Programmable Logic Controller

Subject : Control System II

Mrs. Gulrez Bodhle, Asst. Professor

Semester VII

B.E. Electrical Engineering

Anjuman-I-Islam's Kalsekar Technical Campus
New Panvel - 410206

9/8/2016

CHAPTER 4: PROGRAMMABLE LOGIC CONTROLLER

4.1: INTRODUCTION

A **programmable logic controller (PLC)** is a special form of microprocessor- based controller that uses a programmable memory to store instructions and to implement functions such as logic, sequencing, timing, counting and arithmetic in order to control machines and processes and are designed to be operated by engineers with perhaps a limited knowledge of computers and computing languages. They are not designed so that only computer programmers can set up or change the programs. Thus, the designers of the PLC have pre-programmed it so that the control program can be entered using a simple, rather intuitive, form of language. The term *logic* is used because programming is primarily concerned with implementing logic and switching operations, e.g. if A or B occurs switch on C, if A and B occurs switch on D. Input devices, e.g. sensors such as switches, and output devices in the system being controlled, e.g. motors, valves, etc., are connected to the PLC. The operator then enters a sequence of instructions, i.e. a program, into the memory of the PLC. The controller then monitors the inputs and outputs according to this program and carries out the control rules for which it has been programmed.

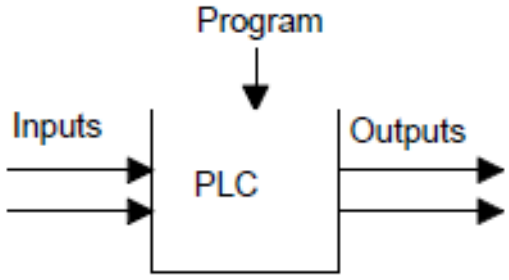


Figure: A Programmable Logic Controller

PLCs have the great advantage that the same basic controller can be used with a wide range of control systems. To modify a control system and the rules that are to be used, all that is necessary is for an operator to key in a different set of instructions. There is no need to rewire. The result is a flexible, cost effective, system which can be used with control systems which vary quite widely in their nature and complexity.

PLCs are similar to computers but whereas computers are optimised for calculation and display tasks, PLCs are optimised for control tasks and the industrial environment. Thus PLCs are:

1. Rugged and designed to withstand vibrations, temperature, humidity and noise.
2. Have interfacing for inputs and outputs already inside the controller.
3. Are easily programmed and have an easily understood programming language which is primarily concerned with logic and switching operations.

The first PLC was developed in 1969. They are now widely used and extend from small self-contained units for use with perhaps 20 digital inputs/outputs to

modular systems which can be used for large numbers of inputs/outputs, handle digital or analogue inputs/outputs, and also carry out proportional-integral-derivative control modes.

4.2: COMPONENTS OF PLC

Typically a PLC system has the basic functional components of processor unit, memory, power supply unit, input/output interface section, communications interface and the programming device.

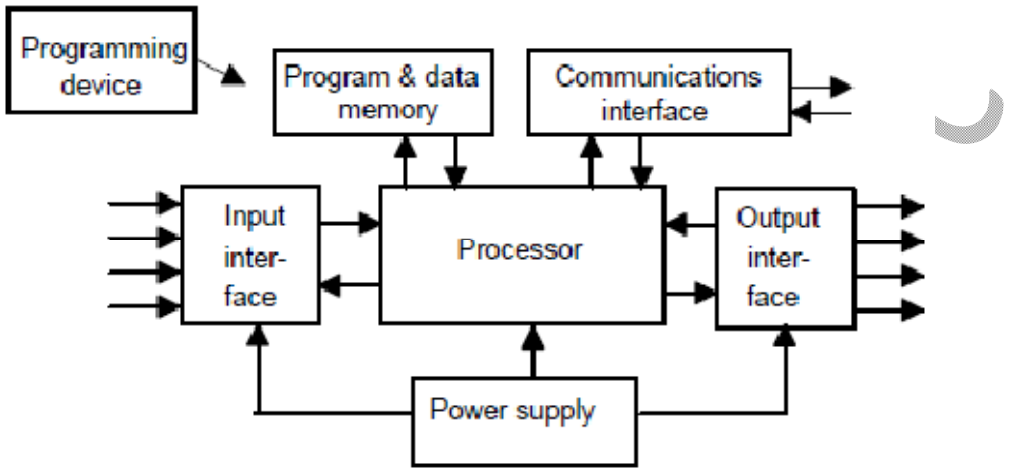


Figure: A Typical PLC System

1. **The Processor Unit**

   The *processor unit* or *central processing unit (CPU)* is the unit containing the microprocessor and this interprets the input signals and carries out the control actions, according to the program stored in its memory, communicating the decisions as action signals to the outputs.

2. **Power Supply Unit**

   The *power supply unit* is needed to convert the mains a.c. voltage to the low d.c. voltage (5 V) necessary for the processor and the circuits in the input and output interface modules.

3. **The Programming Device**

   The *programming device* is used to enter the required program into the memory of the processor. The program is developed in the device and then transferred to the memory unit of the PLC.

4. **The Memory Unit**

   The *memory unit* is where the program is stored that is to be used for the control actions to be exercised by the microprocessor and data stored from the input for processing and for the output for outputting.

5. **The Input and Output Sections**

   The *input and output sections* are where the processor receives information from external devices and communicates information to external devices. The inputs might thus be from switches or other sensors, etc. The outputs might be to motor starter coils, solenoid valves, etc. Input and output devices can be classified as giving signals which are discrete, digital or analogue. Devices giving *discrete* or *digital signals* are ones where the

signals are either off or on. Thus a switch is a device giving a discrete signal, either no voltage or a voltage. *Digital* devices can be considered to be essentially discrete devices which give a sequence of on−off signals. *Analogue* devices give signals whose size is proportional to the size of the variable being monitored.

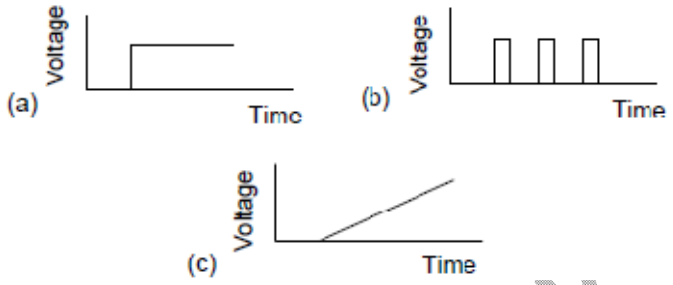


Figure: Signals, a) Discrete b) Digital c) Analog

6. The Communications Interface

The *communications interface* is used to receive and transmit data on communication networks from or to other remote PLCs. It is concerned with such actions as device verification, data acquisition, synchronisation between user applications and connection management.

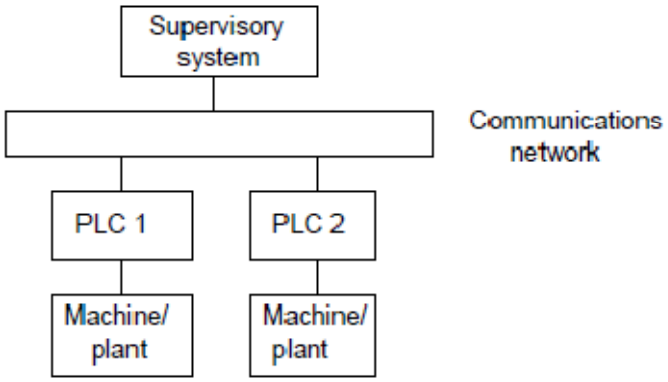


Figure: Basic Communications Module

4.3: THE DIFFERENT LANGUAGES OF PLC

Programs for microprocessor-based systems have to be loaded into them in *machine code*, this being a sequence of binary code numbers to represent the program instructions. However, *assembly language* based on the use of mnemonics can be used, e.g. LD is used to indicate the operation required to load the data that follows the LD, and a computer program called an assembler is used to translate the mnemonics into machine code. Programming can be made even easier by the use of the so-called *high level languages*, e.g. C, BASIC, PASCAL, FORTRAN, COBOL. However, the use of these methods to write programs requires some skill

in programming and PLCs are intended to be used by engineers without any great knowledge of programming. As a consequence, *ladderprogramming* was developed. This is a means of writing programs which can then be converted into machine code by some software for use by the PLC microprocessor.

The IEC (International Electrotechnical Commission) 1131-3 programming languages are:

1) Ladder Diagrams

A very commonly used method of programming PLCs is based on the use of *ladder diagrams*. Writing a program is then equivalent to drawing a switching circuit. The ladder diagram consists of two vertical lines representing the power rails. Circuits are connected as horizontal lines, i.e. the rungs of the ladder, between these two verticals.

In drawing a ladder diagram, certain conventions are adopted:

- **The vertical lines of the diagram represent the power rails between which circuits are connected. The power flow is taken to be from the left-hand vertical across a rung.**
- **Each rung on the ladder defines one operation in the control process.**
- **A ladder diagram is read from left to right and from top to bottom, Figure below showing the scanning motion employed by the PLC. The top rung is read from left to right. Then the second rung down is read from left to right and so on.**

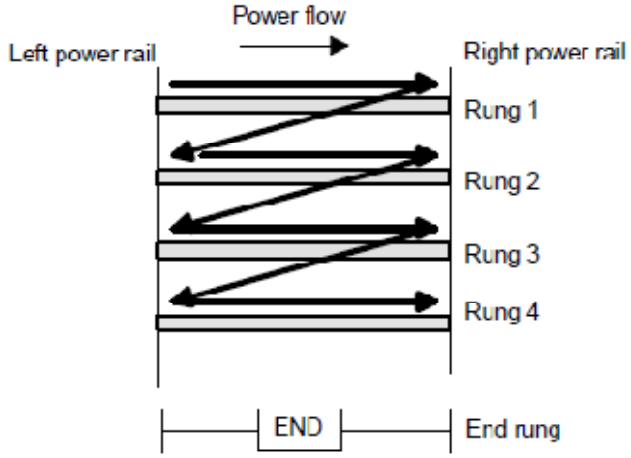


Figure: Scanning The Ladder Diagram

When the PLC is in its run mode, it goes through the entire ladder program to the end, the end rung of the program being clearly denoted, and then promptly resumes at the start. This procedure of going through all the rungs of the program is termed a *cycle*. The end rung might be indicated by a block with the word END or RET for return, since the program promptly returns to its beginning.

- **Each rung must start with an input or inputs and must end with at least one output.**
- **Electrical devices are shown in their normal condition.**
- **A particular device can appear in more than one rung of a ladder. The same letters and/or numbers are used to label the device in each situation.**
- **The inputs and outputs are all identified by their addresses, the notation used depending on the PLC manufacturer.**

2) Function Block Diagrams(FBD)

The term *function block diagram (FBD)* is used for PLC programs described in terms of graphical blocks. It is described as being a graphical language for depicting signal and data flows through blocks, these being reusable software elements. A function block is a program instruction unit which, when executed, yields one or more output values. Thus a block is represented in the manner shown in Figure (a) below with the function name written in the box.

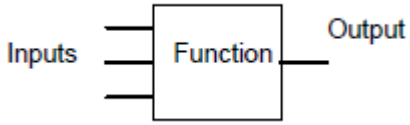
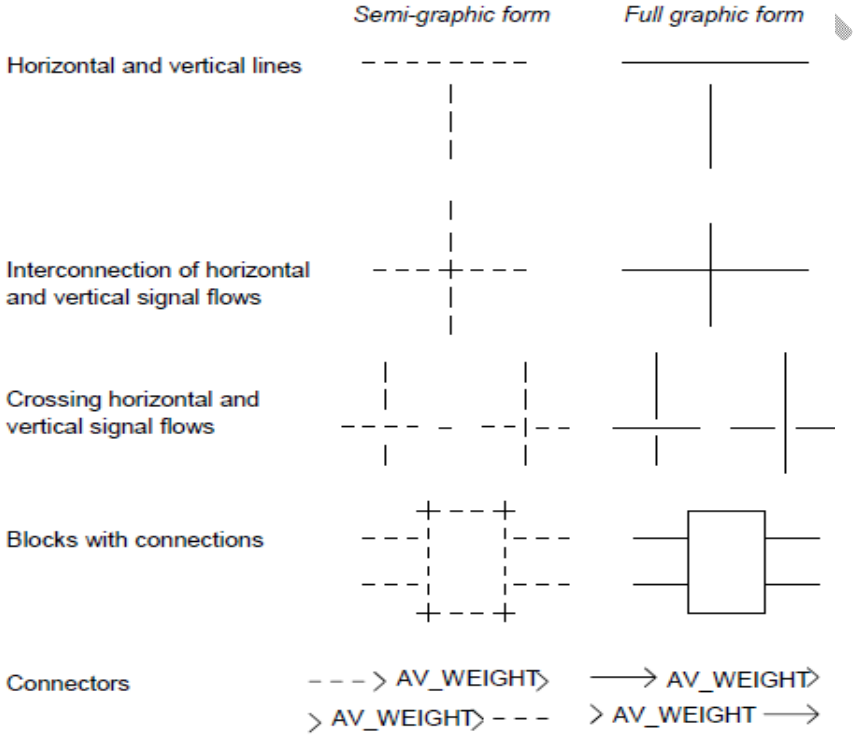


Figure (a): Functional Block

The IEC 113-3 standard for drawing such blocks is shown below:



3) Instruction Lists (IL)

A programming method, which can be considered to be the entering of a ladder program using text, is *instruction lists (IL)*. Instruction list gives programs which consist of a series of instructions, each instruction being on a new line. An instruction consists of an operator followed by one of more operands, i.e. the subjects of the operator. Each instruction may either use or change the value stored in a memory register. The codes used differ to some extent from manufacturer to manufacturer, though a standard IEC 1131-3 has been proposed and is being widely adopted. Table 6.1 shows some of the codes used by manufacturers, and the proposed standard.

As an illustration of the use of IEC 113-1 operators, consider the following:

LD A

```
AND     B
ST      Q
```

LD A is thus the instruction to load the A into the memory register. It can then later be called on for further operations. The next line of the program has the Boolean operation AND performed with A and B. The last line has the result stored in Q, i.e. outputted to Q.

4) Sequential Function Charts(SFC)

The term *sequential function chart* (SFC) is used for a pictorial representation of a system's operation to show the sequence of the events involved in its operation. SFC charts have the following elements:

- **The operation is described by a number of separate sequentially connected states or steps which are represented by rectangular boxes, each representing a particular state of the system being controlled.**
- **Each connecting line between states has a horizontal bar representing the transition condition that has to be realised before the system can move from one state to the next. Two steps can never be directly connected, they must always be separated by a transition. Two transitions can never directly follow from one to another, they must always be separated by a step.**

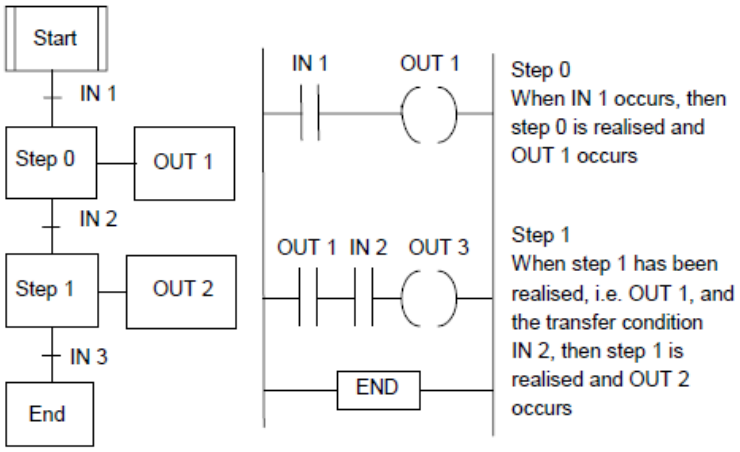


Figure: *SFC and equivalent ladder diagram*

- **When the transfer conditions to the next state are realised then the next state or step in the program occurs.**
- **The process thus continues from one state to the next until thecomplete machine cycle is completed.**
- **Outputs/actions at any state are represented by horizontally linked boxes and occur when that state has been realized.**

5) Structured Text (ST)

Structured text is a programming language that strongly resembles the programming language PASCAL. Programs are written as a series of statements separated by semicolons. The statements use predefined statements and subroutines to change variables, these being defined values, internally stored values or inputs and outputs.

For Example:

```
Light := SwitchA;
```

is used to indicate that a light is to have its 'value' changed, i.e. switched on or off, when switch A changes its 'value', i.e. is on or off. The general

format of an assignment statement is:

X := Y;

where Y represents an expression which produces a new value for the variable X. Another example is:

Light :- SwitchA OR SwitchB;

to indicate that the light is switched on by either switch A or switch B.

4.4: THE DIFFERENT ADDRESSING MODES

The PLC has to be able to identify each particular input and output. It does this by allocating addresses to each input and output. With a small PLC this is likely to be just a number, prefixed by a letter to indicate whether it is an input or an output. Thus for the Mitsubishi PLC we might have inputs with addresses X400, X401, X402, etc., and outputs with addresses Y430, Y431, Y432, etc., the X indicating an input and the Y an output. Toshiba use a similar system.

With larger PLCs having several racks of input and output channels, the racks are numbered. With the Allen-Bradley PLC-5, the rack containing the processor is given the number 0 and the addresses of the other racks are numbered 1, 2, 3, etc. according to how set-up switches are set. Each rack can have a number of modules and each one deals with a number of inputs and/or outputs. Thus addresses can be of the form shown in Figure. For example, we might have an input with address I: 012/03. This would indicate an input, rack 01, module 2 and terminal 03.

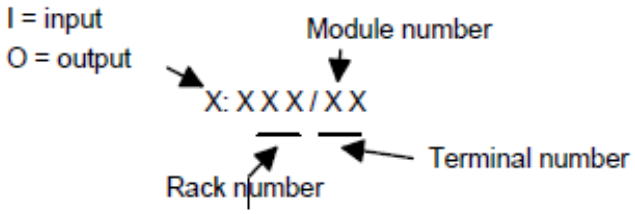


Figure: *Allen-Bradley PLC-5 addressing*

With the Siemens SIMATIC S5, the inputs and outputs are arranged in groups of 8. Each 8 group is termed a byte and each input or output within an 8 is termed a bit. The inputs and outputs thus have their addresses in terms of the byte and bit numbers, effectively giving a module number followed by a terminal number, a full stop (.) separating the two numbers. Figure below shows the system. Thus I0.1 is an input at bit 1 in byte 0; Q2.0 is an output at bit 0 in byte 2.

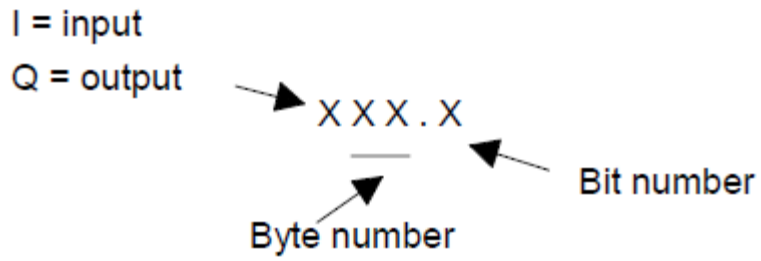


Figure: *Siemens SIMATIC S5 addressing*

The GEM-80 PLC assigns inputs and output addresses in terms of the module number and terminal number within that module. The letter A is used to designate inputs and B outputs. Thus A3.02 is an input at terminal 02 in module 3, B5.12 is an output at terminal 12 in module 5.

In addition to using addresses to identify inputs and outputs, PLCs also use their addressing systems to identify internal, software-created devices, such as relays, timers and counters.

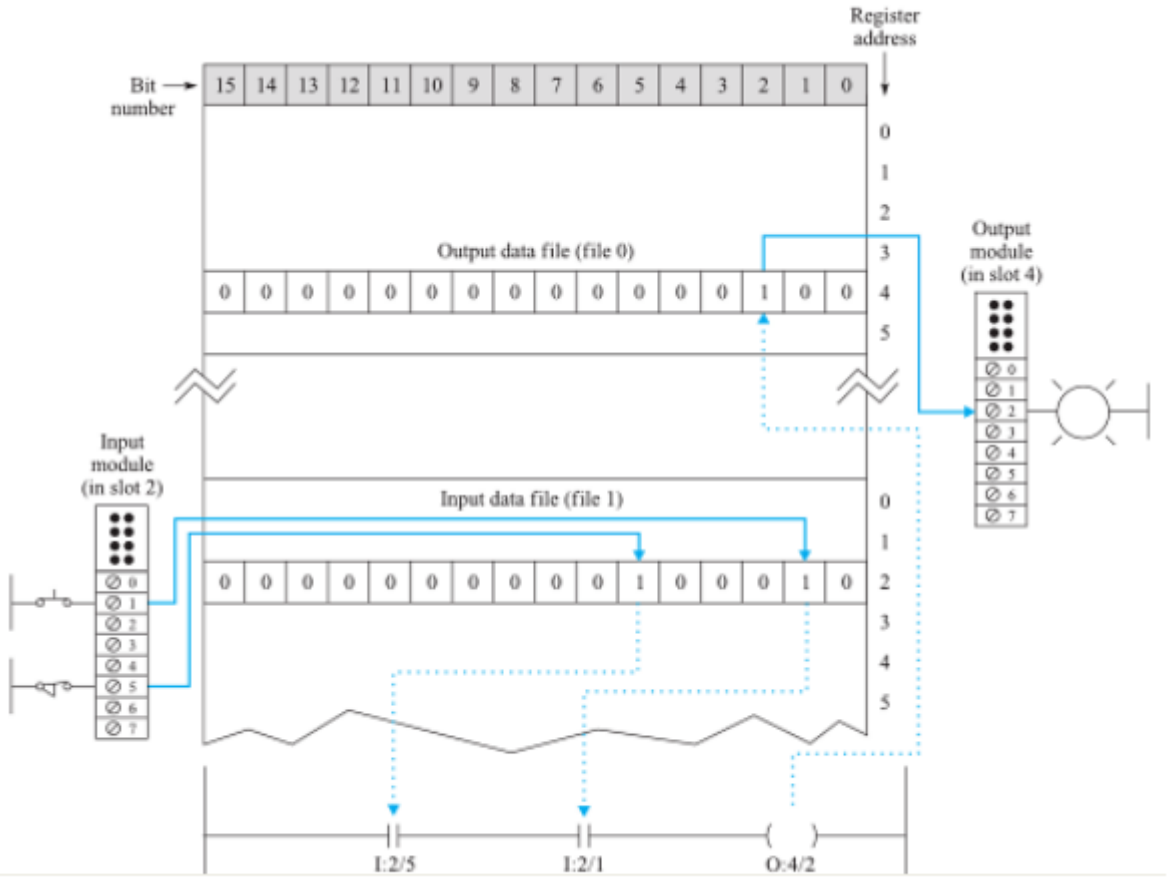
4.5: Relationship of Data File Addresses to I/O Modules

Relationship of Data File Addresses to I/O Modules

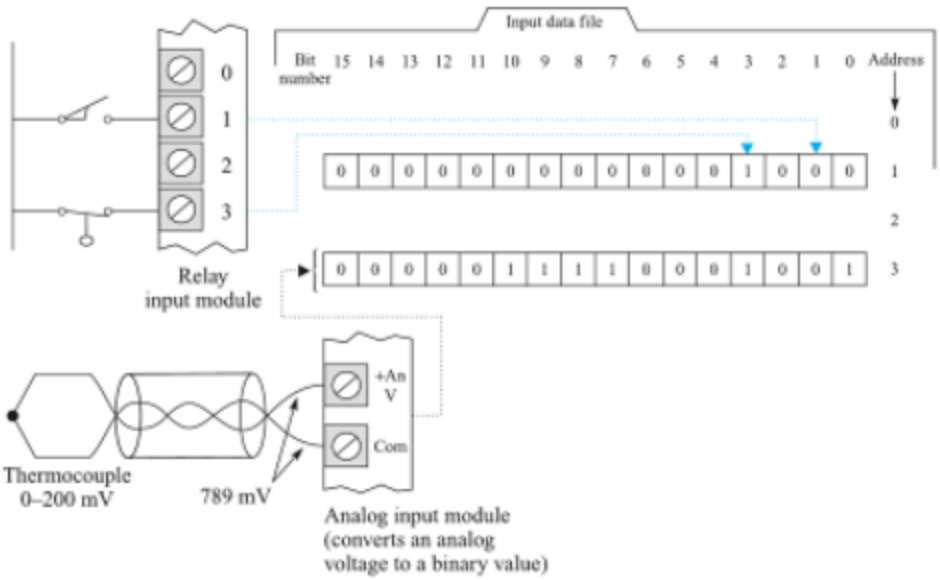
Figure a illustrates the relationship between the number assigned to the data files in memory and the number used by the I/O terminals. The figure contains an input module, a partial memory map of the data file, an output module, and field devices. The ladder diagram at the bottom shows two input switching contacts connected in series with an output symbol. The ladder diagram is programmed into the user program memory (program file 2). A push button is wired to terminal 1, and a limit switch is wired to terminal 5 of the input module located in slot 2. Pressing the push button will cause the corresponding status bit 1 in the input data file to go from 0 to 1, and the limit switch closure causes bit 5 to be set to a 1. This input status creates a True logic condition in the rung and causes bit 2 of the output data file to become a 1. As a result, corresponding output terminal 2 of the module located in slot 4 becomes energized and turns on the output field device to which it is connected. If the push button or the limit switch is open, the corresponding bit in the data file will be a 0, and the logic condition of the rung becomes False and makes bit 2 of the output data file a logic 0. Note that the number following the colon in an I/O address is the same number as the slot position where the module to which the corresponding field device is connected is located. None of the addresses in a data file where the elements are stored can be used more than once.

PLCs are not limited to interfacing with discrete field devices. There are input modules, for example, that convert analog voltages to proportional digital binary values, which are stored in the PLC memory. The digits in the memory register may be pure binary numbers, or BCD numbers that represent decimal digits.

Unlike discrete inputs that require a single memory bit, an analog input requires an entire 16-bit word to represent its value. A pure binary number can represent 65,536 (2^{16}) different voltage levels, and a four-digit BCD number can represent 10,000 (0000 to 9999) different analog voltage levels. Data that represents discrete signals and analog voltages are both stored in the input data file, as shown in Figure b.



a) Relationship of data file addresses to input and output devices



b) Analog and discrete relay signals stored in the input data file

There are also output modules that convert binary numbers from the PLC into proportional analog voltages used by field devices. The binary numbers that represent the analog values and the bits that represent the status of the discrete signals can never be stored in the same register.

6.2: PLC PROGRAM EXECUTION

There are different ways and means of executing a user program. Normally a cyclic execution program is preferred and this cyclic operators are given due priorities. Program processing in a PLC happens cyclically with the following execution:

1. After the PLC is initialised, the **processor** reads the individual inputs. This status of the input is stored in the process- image input table (**PII**).
2. This processor processes the program stored in the program memory. This consists of a list of logic functions and instructions, which are successively processed, so that the required input information will already be accessed before the read in PII and the matching results are written into a process-image output table (**PIQ**). Also other storage areas for counters, timers and memory bits will be accessed during program processing by the processor if necessary.
3. In the third step after the processing of the user program, the status from the PIQ will transfer to the outputs and then be switched on and/or off. Afterwards it begins the execution of the next cycle from step 1.

The same cyclic process also acts upon an RLL program.

The time required by the microprocessor to complete one cycle is known as the **scan time**. After all rungs have been tested, the PLC then starts over again with the first rung. Of course the scan time for a particular processor is a function of the processor speed, the number of rungs, and the complexity of each rung.

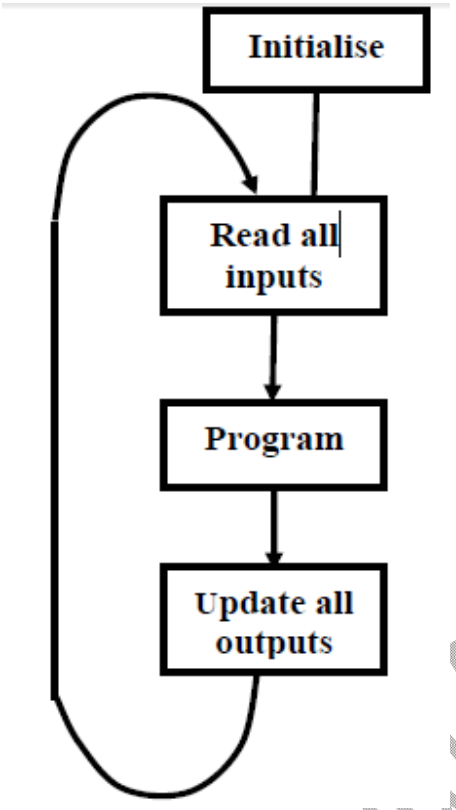


Figure: Cyclic Execution of PLC Programs

Interrupt Driven and Clock Driven Execution Modes:

A cyclically executing program can however be interrupted by a suitably defined signal resulting in an interrupt driven mode of program execution (when fast reaction time is required). If the interrupting signal occurs at fixed intervals we can also realized time synchronous execution (i.e. with closed loop control function). The cyclic execution, synchronized by a real time clock is the most common program structure for a PLC.

Similarly, programmers can also define error-handling routines in their programs. Specific and defined error procedures are then invoked if the PLC operating system encounters fault of given types during execution.

6.5.3 Timers

These are special operands of a PLC, which represent a time delay relay in a relay logic system. The time functions are a fixed component of the central processing unit. The number of these varies from manufacturer to manufacturer and from product to product. It is possible to achieve time delays in the range of few milliseconds to few hours.

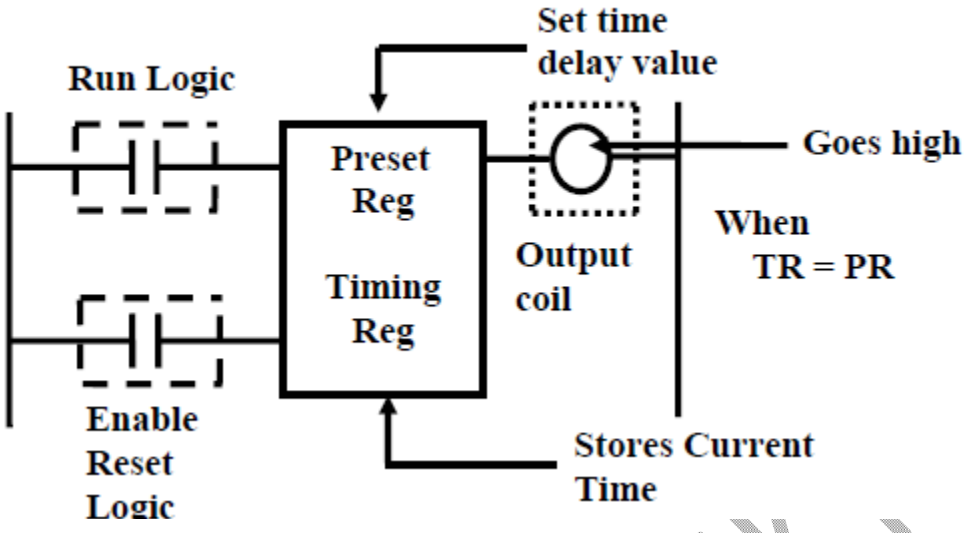


Figure: Structure of a Typical Timer

Representation for timers is shown in Figure above. Timers have a preset register value, which represent the maximum count it can hold and can be set using software/program. The figure shown below has a 'enable reset logic' and 'run logic' in connection with the timer. The counter does not work and the register consists of 'zero' until the enable reset logic is 'on'. Once the 'enable reset logic' is 'on', the counter starts counting when the 'run logic' is 'on'. The output is 'on' only when the counter reaches the maximum count.

Various kinds of timers are explained as follows

1) ON Delay Timer

The input and output signals of the on delay timer are as shown in the Figure below. When the input signal becomes on, the output signal becomes on with certain delay. But when the input signal becomes off, the output signal also becomes off at the same instant. If the input becomes on and off with the time which less than the delay time, there is no change in the output and remains in the 'off' condition even the input is turned on and off i.e., output is not observed until the input pulse width is greater than the delay time.

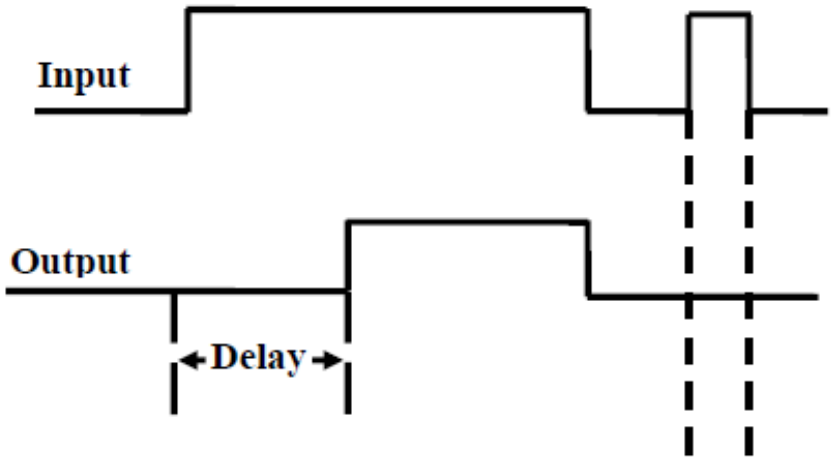


Figure: Input and Output Waveforms of On Delay Timer

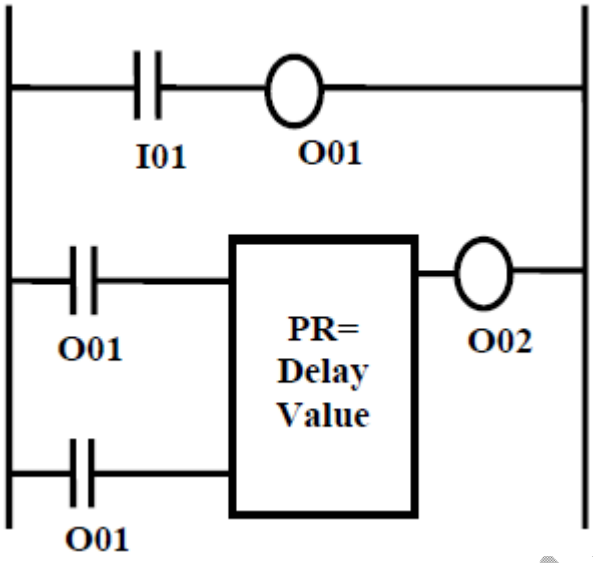


Figure: Realization of On Delay Timer from a General Timer

Realization of on-delay timer: The realization of on-delay timer using the basic timer shown in the previous fig is explained here. The realization is as shown in the Figure above, which shows a real input switch (IN001), coil1 (OP002), two normally opened auxiliary contacts (OP002), coil2(OP002). When the real input switch is 'on' the coil (OP002) is 'on' and hence both the auxiliary switches are 'on'. Now the counter value starts increasing and the output of the timer is 'on' only after it reaches the maximum preset count. The behavior of this timer is shown in figure, which shows the on-delay timer. The value in the counter is 'reset' when the input switch (IN001) is off as the 'enable reset logic' is 'off'. This is a non-retentive timer.

2) OFF Delay Timer

The input and output signals of the off delay timer are as shown in the Figure below. When the input signal becomes on, the output signal becomes on at the same time. But when the input signal becomes off, the output signal becomes 'off' with certain delay. If the input becomes on and off with the time which less than the delay time, there is no change in the output and remains in the 'on' condition even the input is turned on and off i.e., the delay in the output is not observed until the input pulse width is greater than the delay time.

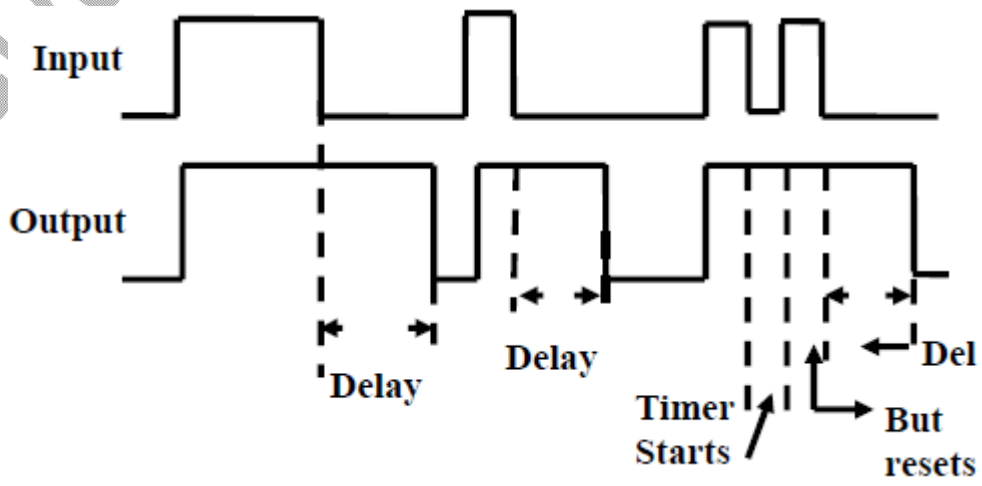


Figure: Typical Input Output Waveforms of OFF Delay Timer

Realization of off-delay timer: The realization of on-delay timer using the basic timer shown in the previous fig is explained here. The realization is as shown in the Figure below, which shows a real input switch (IN001), coil1 (OP002), two normally closed input contacts (IN001), output contacts (OP002, OP003). When the real input switch is 'on', the coil (OP002) is 'on' and both the auxiliary input switches are 'off'. Now the output contact (OP002) becomes 'off' which in turn makes the auxiliary contact (OP002) in the third rung to become 'on' and hence the output contact (OP003) is 'on'. When the real input switch is 'off', the counter value starts increasing and the output of the contact becomes 'on' after the timer reaches the maximum preset count. At this time the auxiliary contact in the third rung becomes 'off' and so is the output contact (OP003). The input and output signals are as shown in the figure, which explain the off-delay timer.

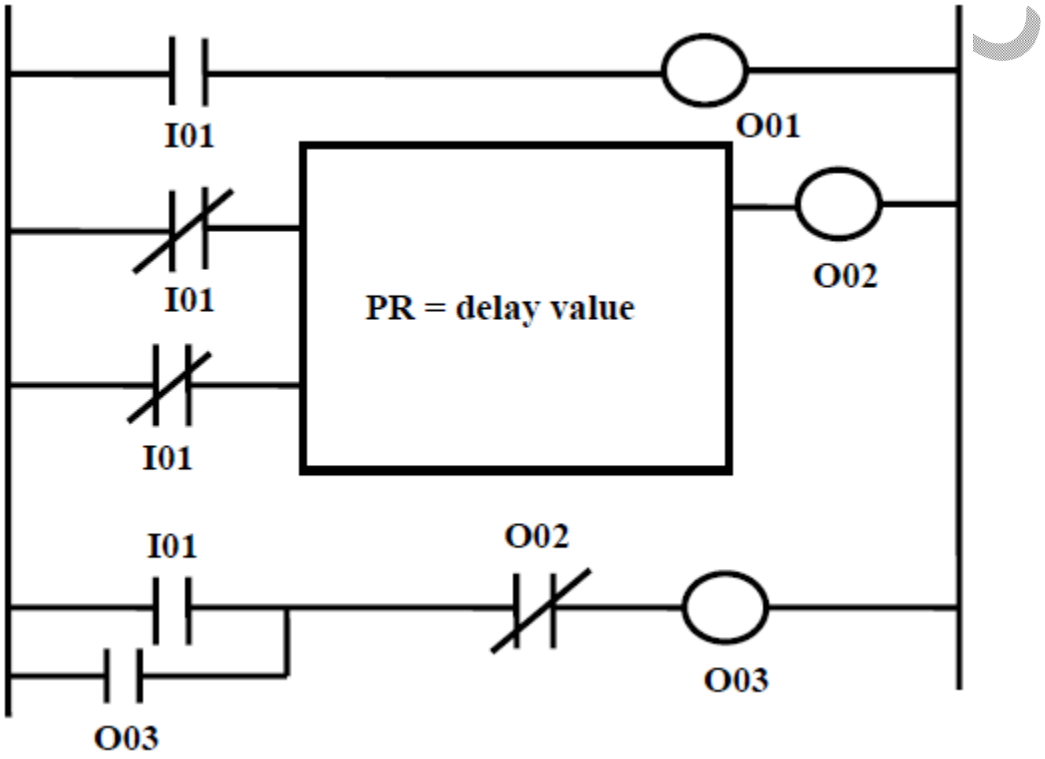


Figure: Realization of OFF Delay Timer from a General Timer

3) Fixed Pulse Width Timer

The input and output signals of the fixed pulse width timer are as shown in the Figure below. When the input signal becomes on, the output signal becomes on at the same time and remains on for a fixed time then becomes 'off'. The output pulse width is independent of input pulse width.

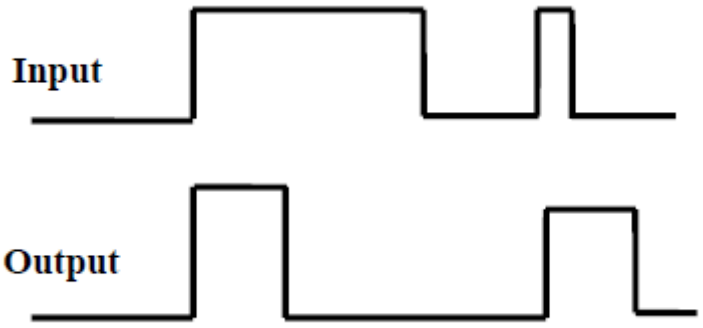


Figure: Input Output Waveforms of Fixed Pulse Width Timer

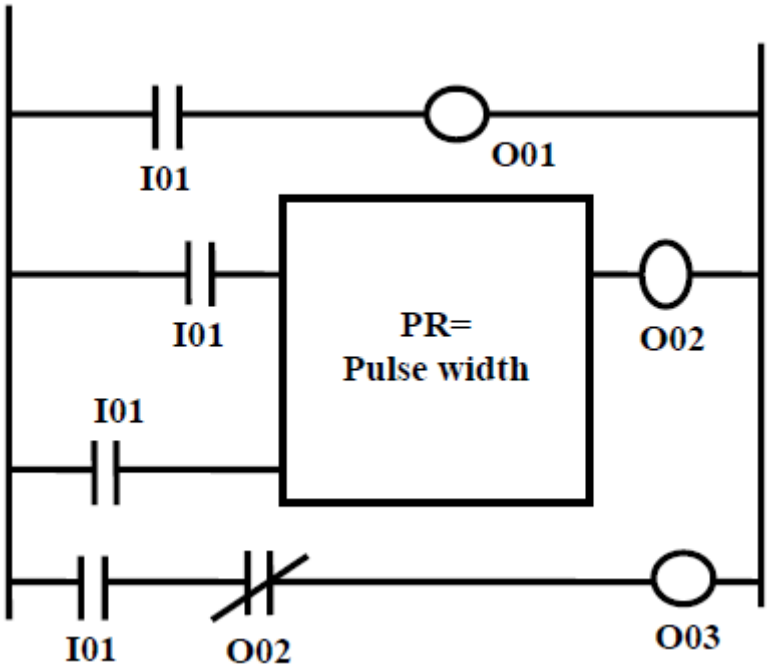


Figure: Realization of Fixed Pulse Width Timer from a General Timer

4) Retentive Timer

The input and output signals of the retentive timer are as shown in the figure below.

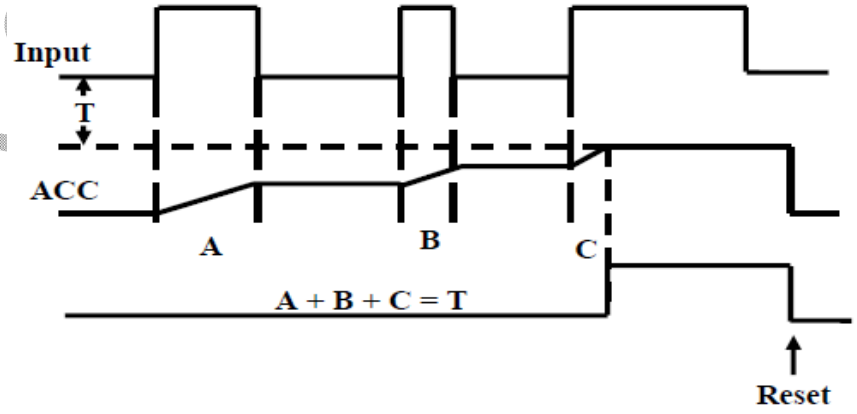


Figure: Input and Output Waveforms of Retentive Timer

This is also implemented internally in a register as in the previous case. When the input is 'on' , the internal counter starts counting until the input is 'off' and at this time, the counter holds the value till next input pulse is applied and then starts counting starting with the

value existing in the register. Hence it is named as 'retentive' timer. The output is 'on' only when the counter reaches its 'terminal count'.

5) Non – Retentive Timer

The input and output signals of the non-retentive timer are as shown in the Figure below. This is implemented internally in a register. When the input is 'on', the internal counter starts counting until the input is 'off' and at this time the value in the counter is reset to zero. Hence it is named as non-retentive timer. The output is 'on' only when the counter reaches its 'terminal count'.

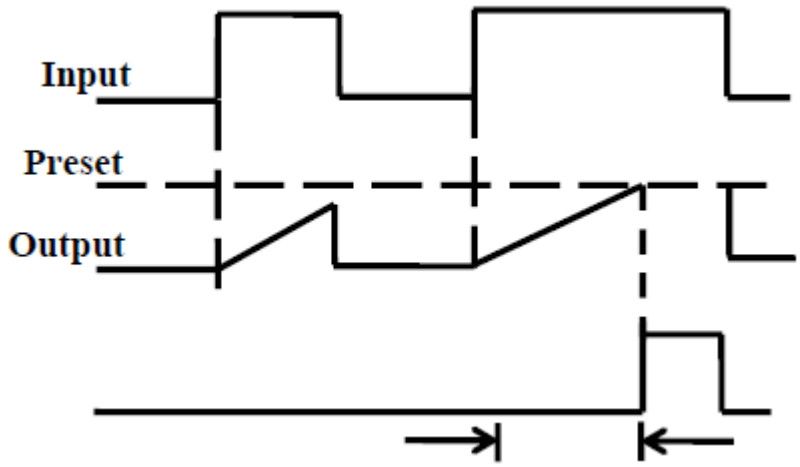


Figure: Input and Output Waveforms of a Non-Retentive Timer

6.5.4 COUNTER

The counting functions (C) operate as hardware counters, but are a fixed component of the central processing unit. The number of these varies for each of the programmable controllers. It is possible to count up as well as to count down. The counting range is from 0 to 999. The count is either dual or BCD coded for further processing.

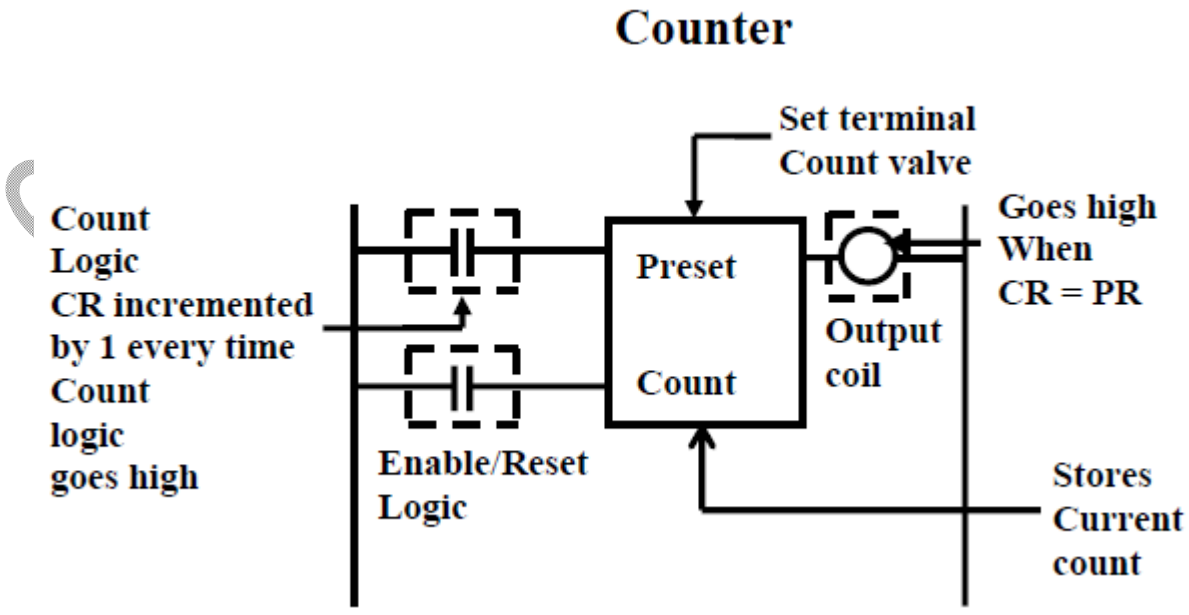


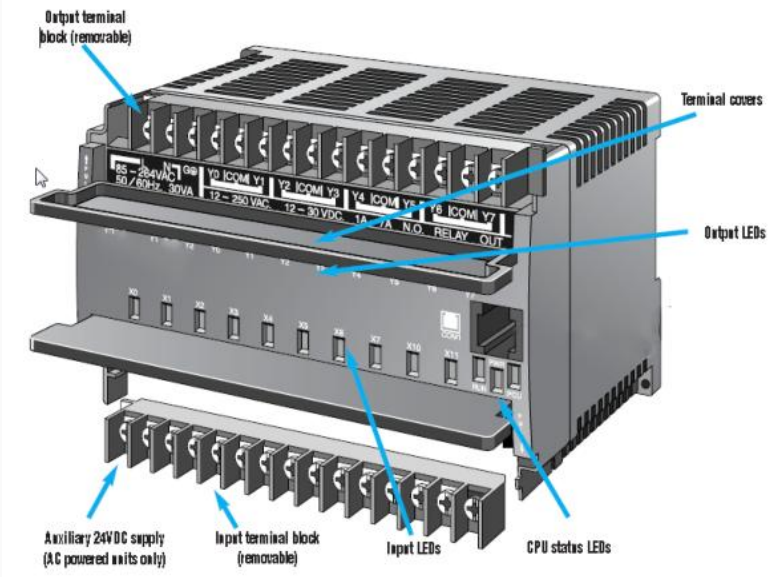
Figure: Structure of a Typical Counter

Modular and Fixed PLCs

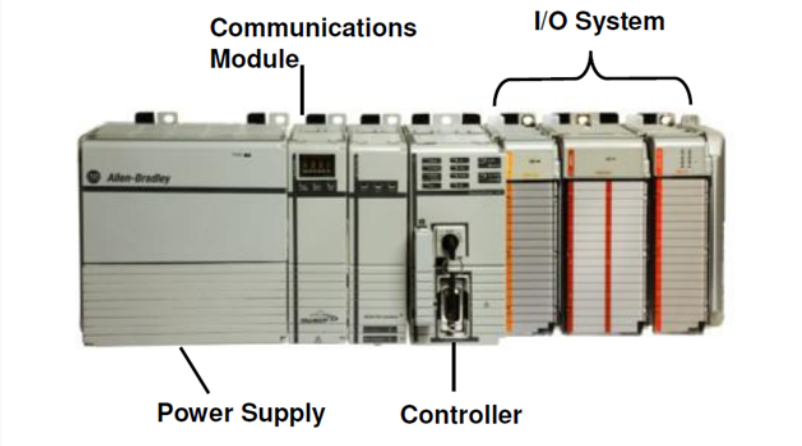
As mentioned, PLCs have two main categories: modular or fixed. Both have the same basic functions. A PLC is much like a personal computer. It consists of a central processing unit (CPU) and an input/output (I/O) interface system. The difference between a PC and PLCs is that PLCs handle multiple configurations and carry out control functions. All I/O systems consist of message or information carriers - inputs - and controllable devices - outputs. Any switches, sensors or other similar devices are physically linked to the main I/O system and all of the activity throughout the entire system is controlled by the CPU.

This is the same regardless of whether you're using a modular or a fixed PLC. However, these two systems are very different in application and execution.

- **Fixed PLCs** are typically designed to perform basic functions. They are small and consist of a power supply and the CPU and I/O systems, which are all housed in one entity. The proper functioning of all processes largely depends on the flawless execution of every component.



- **Modular PLCs**, also known as rack-mounted units, consist of bases allowing for many independent components, such as the installation of numerous I/O modules. These are easier to repair.



Advantages of Modular PLCs

What are the main advantages of a modular PLC?

Memory

Fixed PLCs are only capable of housing so much information. This is because of the fact that all the necessary parts are housed in one entity. Also, there isn't a lot of room for expanding information beyond the very basic functions it requires. Modular PLCs have far more memory and has the capability to store a higher volume of information.

Greater Number of Available I/O Modules

The fixed PLC has limited I/O modules thus limiting its performance. The modular PLC has the capacity to accomplish more complex processes making it more advantageous.

Room for Expansion

As the need to expand your operation arises, the fixed PLC will not be able to grow with the company. Modular PLCs were designed for that purpose, to expand and customize processes for seamless growth.

Easier to Troubleshoot, Less Downtime

Downtime costs money- no one wants a system that drains the profits. Fixed PLCs do not alert the operator of problems. Thus, when they crash it takes considerable time to troubleshoot the problem. However, the modular PLCs can easily troubleshoot issues, keeping some processes operational while fixing the issue. While fixed PLCs may be less expensive, initially, the modular PLC has greater economic security long-term.

PLC Advantages

- Increased Reliability
- More Flexibility
- Lower Cost
- Faster Response
- Easier to troubleshoot
- Remote control capability
- Communication Capability

PLC Disadvantages

- **NO Compatibility between manufacturers**

JUMP AND LOOP INSTRUCTIONS AND PLC TROUBLESHOOTING

6.1: Jump Instruction

A function often provided with PLCs is the *conditional jump*. We can describe this as:

IF (some condition occurs) THEN
perform some instructions
ELSE
perform some other instructions.

Such a facility enables programs to be designed such that if certain conditions are met then certain events occur; if they are not met then other events occur. Thus, for example, we might need to design a system so that if the temperature is above 60^{o}C a fan is switched on, and if below that temperature no action occurs.

Thus, if the appropriate conditions are met, this function enables part of a ladder program to be jumped over. Figure 6.1 illustrates this in a general manner. When there is an input to In 1, its contacts close and there is an output to the jump relay. This then results in the program jumping to the rung in which the jump end occurs, so missing out intermediate program rungs. Thus, in this case, when there is an input to Input 1, the program jumps to rung 4 and then proceeds with rungs 5, 6, etc. When there is no input to Input 1, the jump relay is not energised and the program then proceeds to rungs 2, 3, etc.

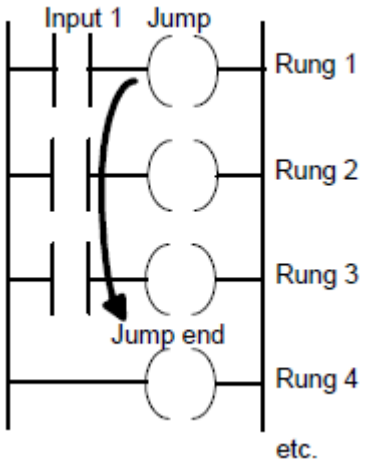


Figure 6.1: Jump Instruction

Figure 6.2 shows the above ladder program in the form used by Mitsubishi. The jump instruction is denoted by CJP (conditional jump) and the place to which the jump occurs is denoted by EJP (end of jump). The condition that the jump will

occur is then that there is an input to X400. When that happens the rungs involving inputs X401 and X403 are ignored and the program jumps to continue with the rungs following the end jump instruction with the same number as the start jump instruction, i.e. in this case EJP 700.

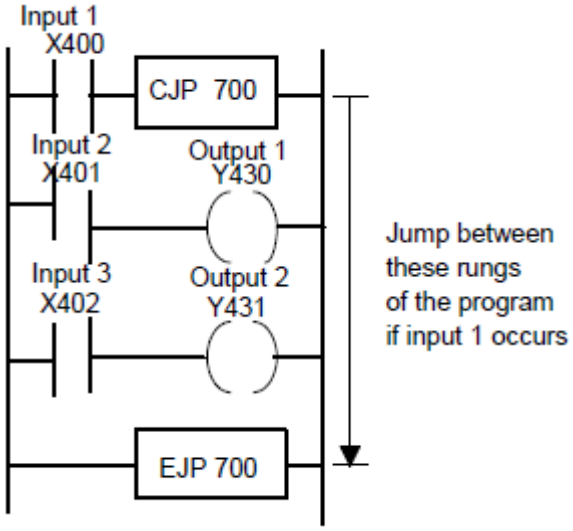


Figure 6.2: Jump Instruction in MITSUBISHI program

With the Allen-Bradley PLC-5 format the jump takes place from the jump instruction (JMP) to the label instruction (LBL). The JMP instruction is given a three-digit number from 000 to 255 and the LBL instruction the same number. Figure 6.3 shows a ladder program in this format.

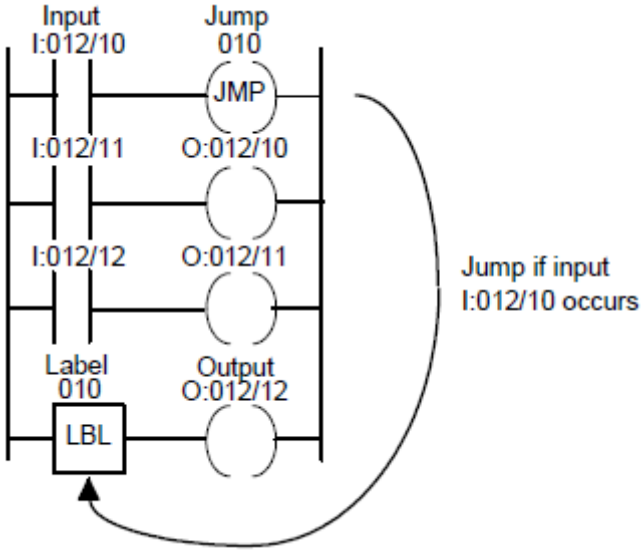


Figure 6.3: JUMP Instruction in Allen Bradley Program

With Siemens' programs, conditional jumps are represented as shown in Figure 6.4, there being a jump instruction JMP which is executed if the input is a 1 and another jump instruction JMPN which is executed if the input is 0. The end of both instructions is the label DEST.

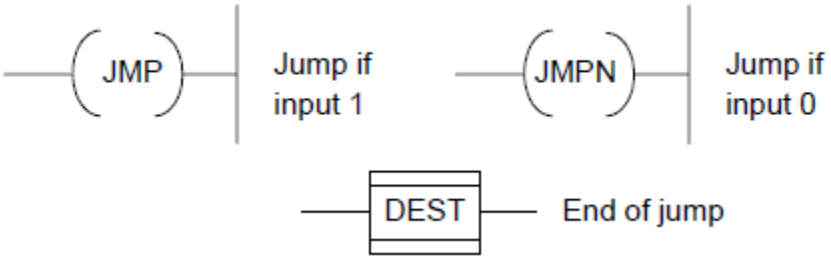


Figure 6.4: Siemens' Jump Instructions

6.1.1: Jump within Jumps

Jumps within jumps are possible. For example, we might have the situation shown in Figure 6.5. If the condition for the jump instruction 1 is realized then the program jumps to rung 8. If the condition is not met then the program continues to rung 3. If the condition for the jump instruction 2 is realized then the program jumps to rung 6. If the condition is not met then the program continues through the rungs.

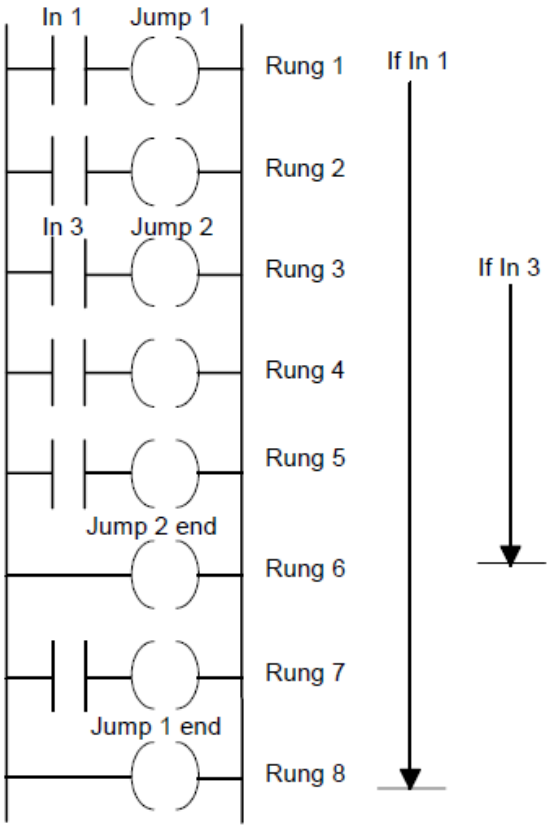


Figure 6.5: Jumps Within Jumps

Thus if we have an input to In 1, the rung sequence is rung 1, 8, etc. If we have no input to In 1 but an input to In 3, then the rung sequence is 1, 2, 6, 7, 8, etc. If we have no input to In 1 and no input to In 3, the rung sequence is 1, 2, 3, 4, 5, 6, 7, 8, etc. The jump instruction enables different groups of program rungs to be selected, depending on the conditions occurring.

6.2: SUBROUTINE OR CALL OR LOOP INSTRUCTION

Subroutines are small programs to perform specific tasks which can be called for use in larger programs. Thus with a Mitsubishi program we might have the situation shown in Figure 6.6. When input 1 occurs, the subroutine P is called. This is then executed, the instruction SRET indicating its end and the point at which the program returns to the main program. To clearly indicate where the main program ends the FEND instruction is used.

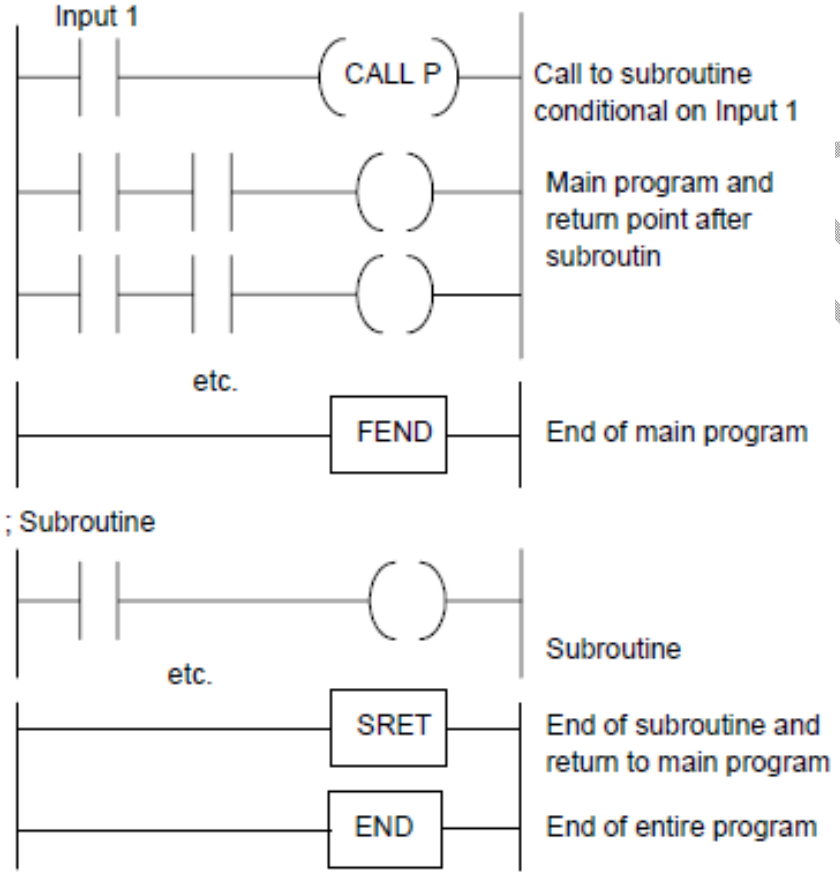


Figure 6.6: Subroutine Call with MITSUBISHI PLC

With Siemens a similar format can be adopted, using CALL to call up a subroutine block and RET to indicate the return instruction to the main program. However, a function box approach (Figure 8.6) can be used and is particularly useful where there is a library of subroutine functions to be called. If the EN (enable) block input is connected directly to the left power rail then the call is without conditions and always executed. If there is a logic operation preceding EN then the block call is only executed if the logic condition is fulfilled, in Figure 6.7 this is closure of contacts of Input 1. Several blocks can be connected in series by connecting the ENO, enable output, of one to the EN input of the next.

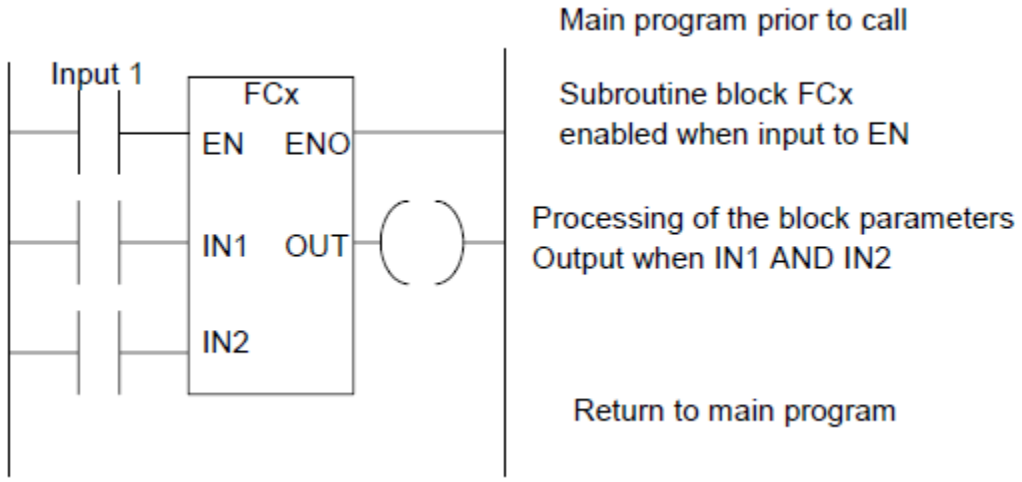


Figure 6.7: Call to subroutine blocks with Siemens PLC

With Allen-Bradley, subroutines are called by using a jump-to subroutine JSR instruction, the start of the subroutine being indicated by SBR and its end and point of return to the main program by RET (Figure 6.8).

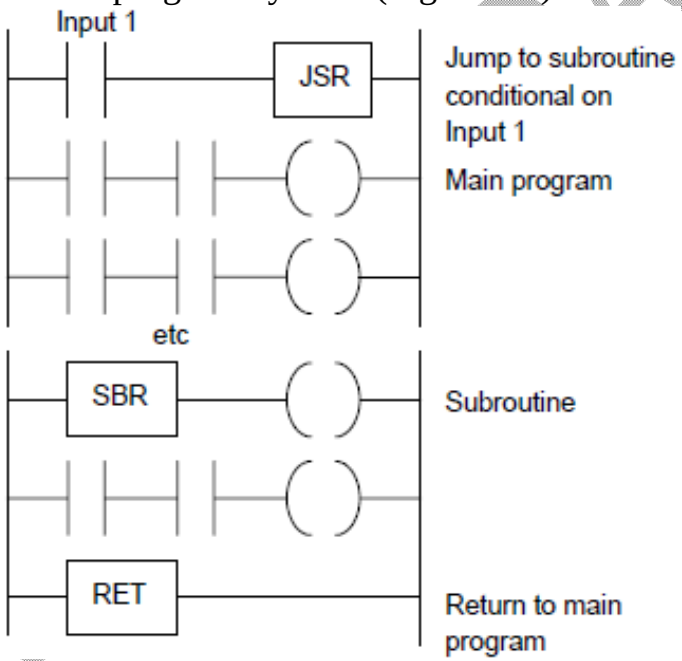


Figure: Jump to subroutine call with Allen-Bradley PLC

6.3: PLC TROUBLESHOOTING

6.3.1 Program Troubleshooting

There are several causes of alteration to the user program:

1. Extreme Environmental Conditions
2. Electromagnetic Interfrenece
3. Improper Grouding
4. Improper Wiring Connections
5. Unauthorized Tampering

If you suspect the memory has been altered, check the program against a previously saved program on an EEPROM, UVPROM or flash EPROM module.

6.3.2 Hardware Troubleshooting

If the installation and start-up procedures were followed closely, controller will give reliable service. If a problem occurs, the first step in the troubleshooting procedure is to identify the problem and its source. Do this by observing your machine or process and by monitoring the diagnostic LED indicators on the CPU, Power Supply and I/O Modules.

By *observing* the diagnostic indicators on the front of the processor unit and I/O modules, the majority of faults can be located and corrected. These indicators, along with error codes identified in the programming device user manual and programmer's monitor, help trace the source of the fault to the user's input/output devices, wiring, or the controller.

6.3.3 Troubleshooting Controller

In identifying the source of the controller's operation problem use troubleshooting considerations table including status indication, trouble description, probable causes and recommended action.

To receive the maximum benefits follow these steps:

- Identify Power Supply and CPU LED status indicators;
- Match processor LEDs with the status LEDs located in troubleshooting tables;
- Once the status LEDs are matched to the appropriate table, simply move across the table identifying error description and probable causes;
- Follow the recommended action steps for each probable cause until the cause is identified;
- If recommended actions do not identify the cause, contact manufacturer or distributor for assistance.

6.3.4 Troubleshooting Input Modules

An input circuit responds to an input signal in the following manner:

- An input filter removes false signals due to contact bounce or electrical interference
- Optical isolation protects the backplane circuits by isolating logic circuits from input signals
- Logic circuits process the signal
- An input LED turns on or off indicating the status of the corresponding input device
- The processor receives the input status for use in processing the program logic

6.3.5 Troubleshooting Output Modules

An output circuit controls the output signal in the following manner:

- The processor determines the output status;
- Logic circuits maintain the output status.
- An output LED indicates the status of the output signal,
- Optical isolation separates logic and backplane circuits from field signals;
- The output driver turns the corresponding output on or off.

Power distribution

The master control relay must be able to inhibit all machines motion by removing power to the machine I/O devices when the relay is de-energized. The DC power supply should be powered directly from the fused secondary of the transformer. Power to the DC input, and output, circuits is connected through a set of master control relay contacts. Interrupt the load side rather the AC line power. This avoids the additional delay of power supply turn-on and turn-off.

POWER LED

The POWER LED on the power supply indicates that DC power is being supplied to the chassis. This LED could be off when incoming power is present when the: Fuse is blown; Voltage drops below the normal operating range; Power supply is defective.

Safety Considerations

Actively thinking about the safety of yourself and others, as well as the condition of your equipment, is of primary importance.

When troubleshooting, pay careful attention to these general warnings:

- Have all personnel remain clear of the controller and equipment when power is applied.
- The problem may be intermittent and sudden unexpected machine motion could result in injury.
- Have someone ready to operate an emergency-stop switch in case it becomes necessary to shut off power to the controller equipment.
- Never reach into a machine to actuate a switch since unexpected machine motion can occur and cause injury.
- Remove all electrical power at the main power disconnects switches before checking electrical connections or inputs/outputs causing machine motion.
- Never alter safety circuits to defeat their functions. Serious injury or machine damage could result.

Calling for assistance

If you need to contact manufacturer or local distributor for assistance, it is helpful to obtain the following (prior to calling):

- Processor type,
- Series letter
- Processor LED status
- Processor error codes
- Hardware types in system (I/O modules, chassis)
- Revision of programming device (HHT or APS).

System documentation

The documentation is the main guide used by the users and for troubleshooting and fault finding with PLCs. The documentation for a PLC installation should include:

- A description of the plant.
- Specification of the control requirements.
- Details of the programmable logic controller.
- Electrical installation diagrams.
- Lists of all inputs and outputs connections.
- Application program with full commentary on what it is achieving.
- Software back-ups.
- Operating manual, including details of all start up and shut down procedures and alarms.