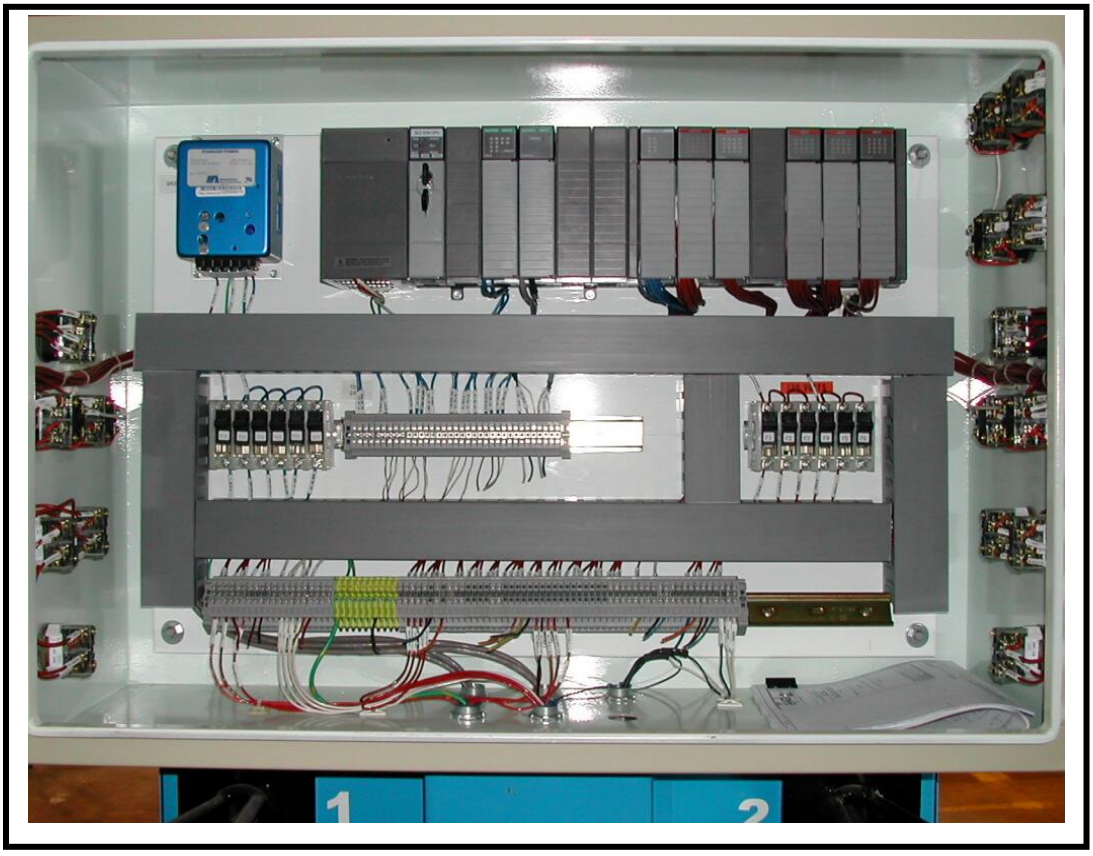
2014

JUBAIL INDUSTRIAL COLLEGE

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING TECHNOLOGY



EEE 422 ADVANCED PLC

EEE 422: ADVANCED PLC

TABLE OF CONTENTS

PART ONE

PROGRAMMABLE LOGIC CONTROLLERS
AN OVERVIEW

PROGRAMMABLE LOGIC CONTROLLERS (PLC's)

The Need for PLC's

- Hardwired panels were very time consuming to wire, debug and change.
- GM identified the following requirements for computer controllers to replace hardwired panels.

    - Solid-state not mechanical
    - Easy to modify input and output devices
    - Easily programmed and maintained by plant electricians
    - Be able to function in an industrial environment

The First Programmable Logic Controllers (PLC's)

- Introduced in the late 1960's
- Developed to offer the same functionality as the existing relay logic systems
- Programmable, reusable and reliable

    - Could withstand a harsh industrial environment -They had no hard drive, they had battery backup
    - Could start in seconds
    - Used Ladder Logic for programming

Programmable Logic Controller

- A programmable logic controller (PLC) is a specialized computer used to control machines and process.

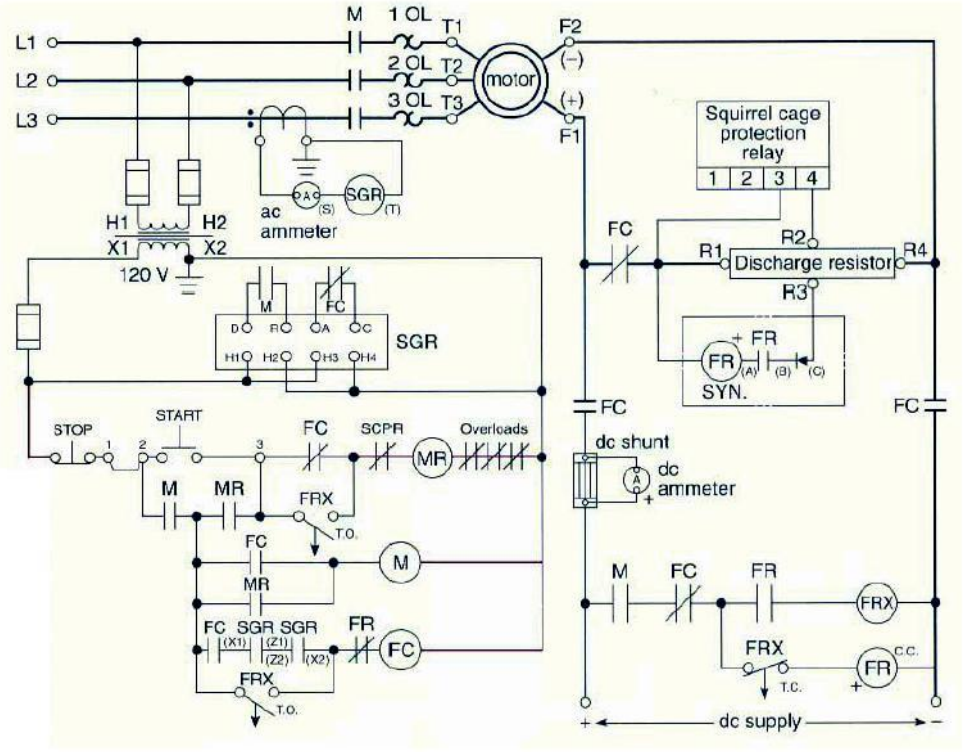
- It uses a programmable memory to store instructions and specific functions that include On/Off control, timing, counting, sequencing, arithmetic, and data handling

Advantages of PLC Control Systems

- Flexible
- Faster response time
- Less and simpler wiring
- Solid-state -no moving parts
- Modular design-easy to repair and expand
- Handles much more complicated systems
- Sophisticated instruction sets available
- Allows for diagnostics "easy to troubleshoot"
- Less expensive

Advantages of a PLC Control System

Eliminates much of the hard wiring, which was associated with conventional relay control circuits.

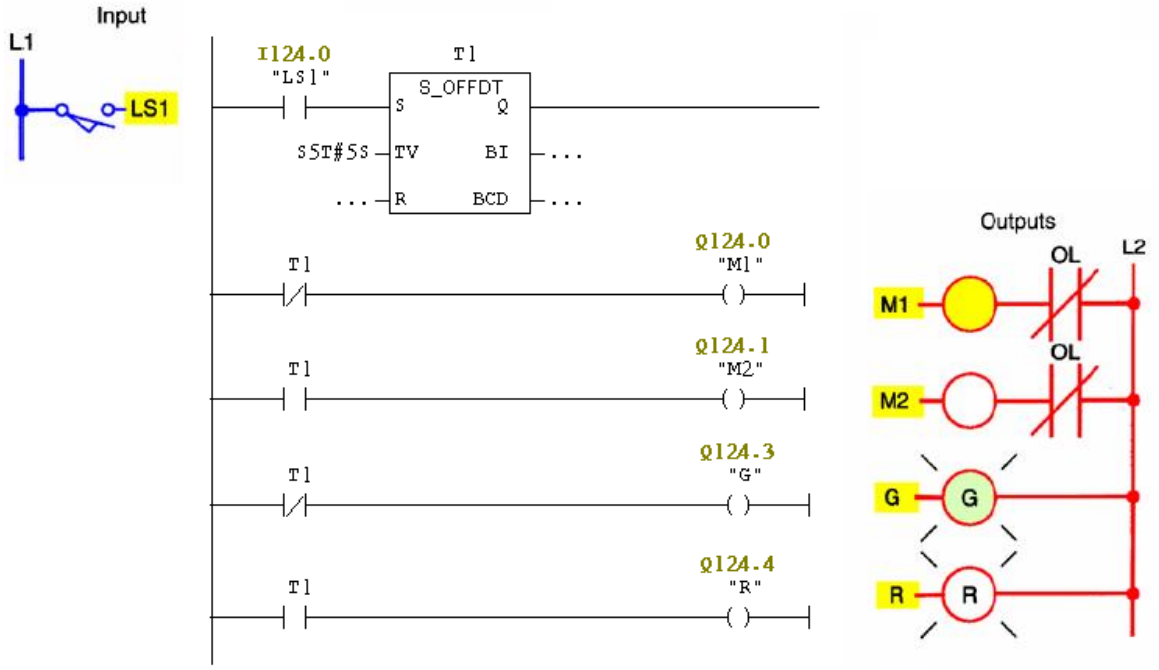


The program takes the place of much of the external wiring that would be required for control of a process.

Increased Reliability:

Once a program has been written and tested it can be downloaded to other PLC's.

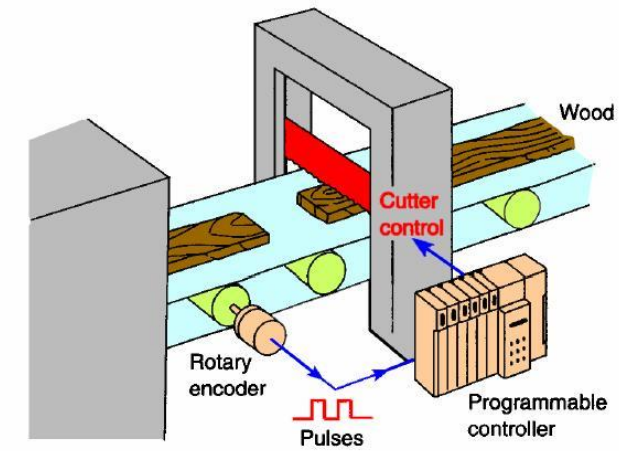
Since all the logic is contained in the PLC's memory, there is no chance of making a logic wiring error.



More Flexibility:

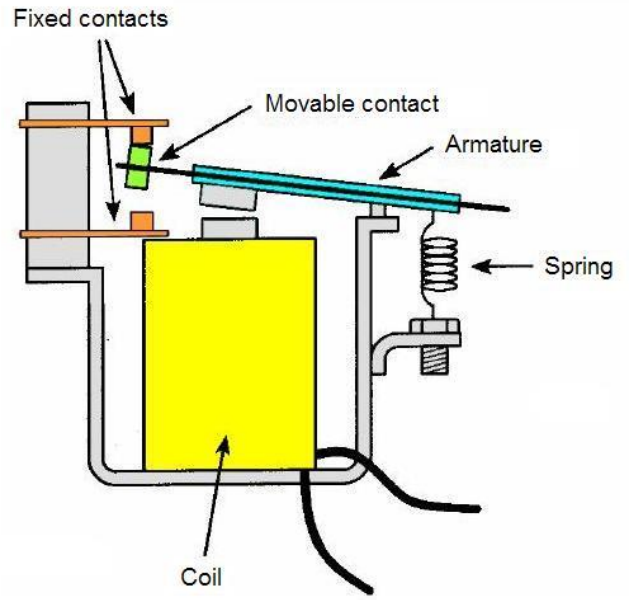
Original equipment manufacturers (OEMs) can provide system updates for a process by simply sending out a new program.

It is easier to create and change a program in a PLC than to wire and rewire a circuit. End-users can modify the program in the field.



Lower Costs:

Originally PLC's were designed to replace relay control logic. The cost savings using PLC's have been so significant that relay control is becoming obsolete, except for power applications.

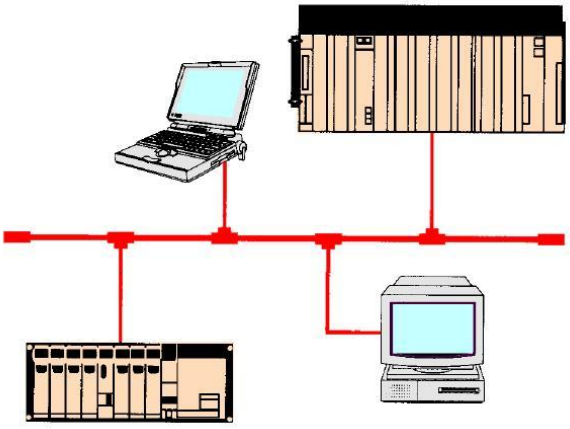


Generally, if an application requires more than about 6 control relays, it will usually be less expensive to install a PLC.

Communications Capability:

A PLC can communicate with other controllers or computer equipment.

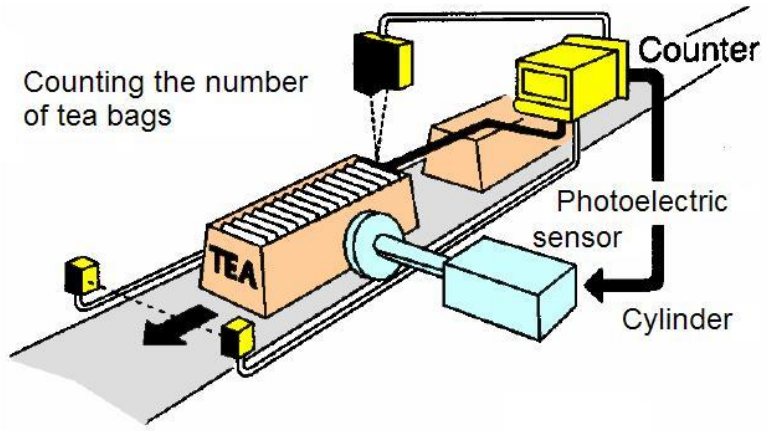
They can be networked to perform such functions as: supervisory control, data gathering, monitoring devices and process parameters, and downloading and uploading of programs.



Faster Response Time:

PLC's operate in real-time which means that an event taking place in the field will result in an operation or output taking place.

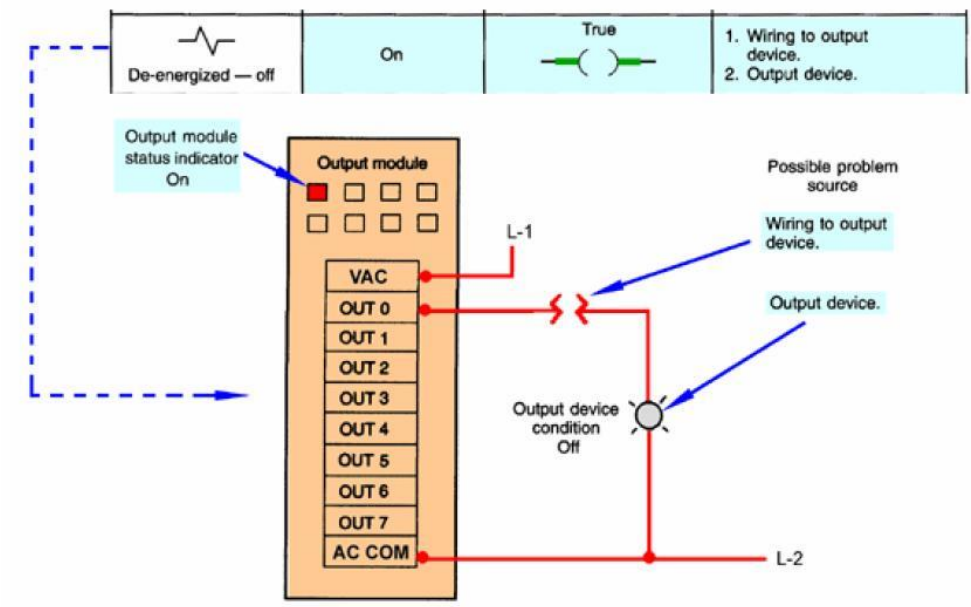
Machines that process thousands of items per second and objects that spend only a fraction of a second in front of a sensor require the PLC's quick response capability.



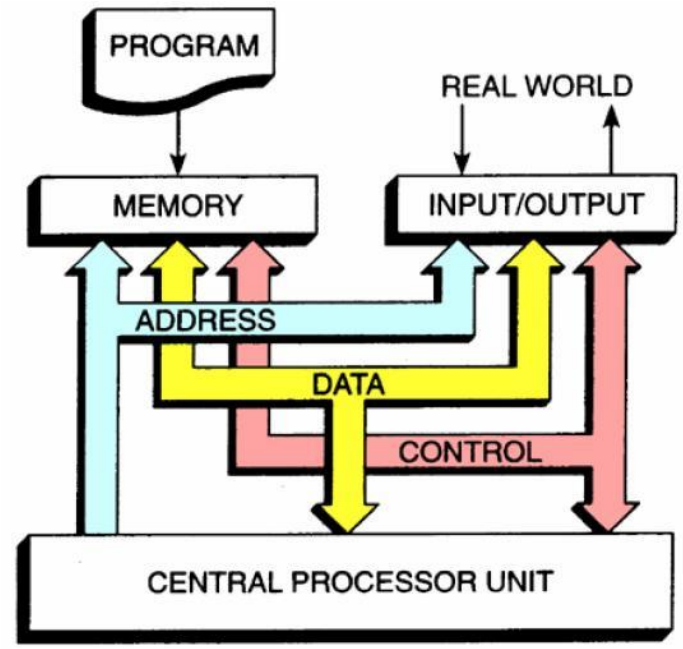
Easier To Troubleshoot:

PLC's have resident diagnostic and override functions allowing users to easily trace and correct software and hardware problems.

The control program can be watched in real-time as it executes to find and fix problems.

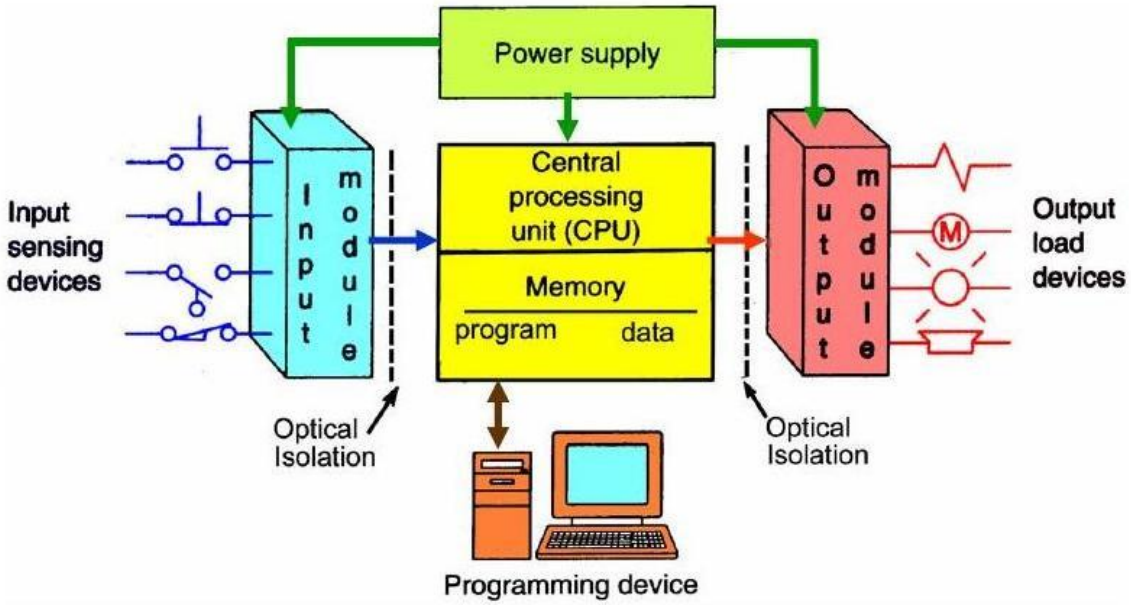


PLC Architecture:



The structure of a PLC is based on the same principles as those employed in computer architecture.

PLC System:



PLC Architecture:

- An *open architecture* design allows the system to be connected easily to devices and programs made by other manufacturers.

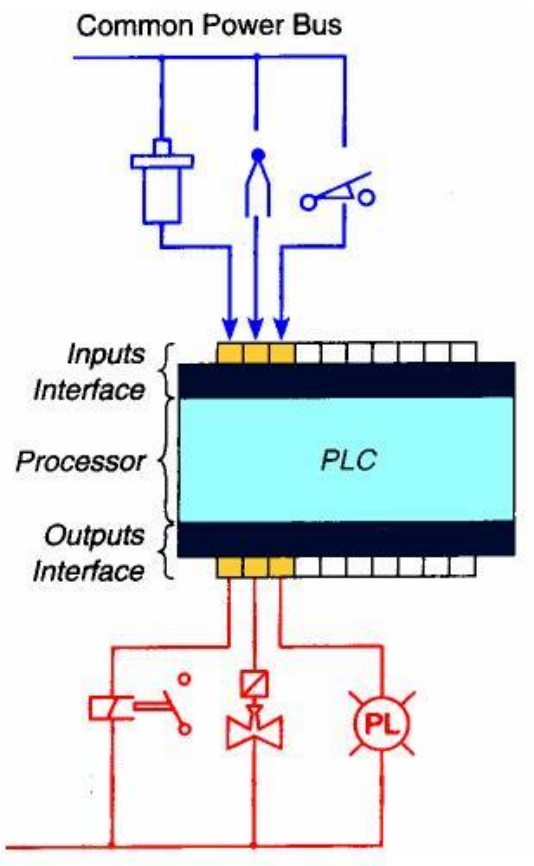
- A *closed architecture* or *proprietary* system, is one whose design makes it more difficult to connect devices and programs made by other manufacturers.

  NOTE: When working with PLC systems that are proprietary in nature you must be sure that any generic hardware or software you use is compatible with your particular PLC.

I/0 Configurations:

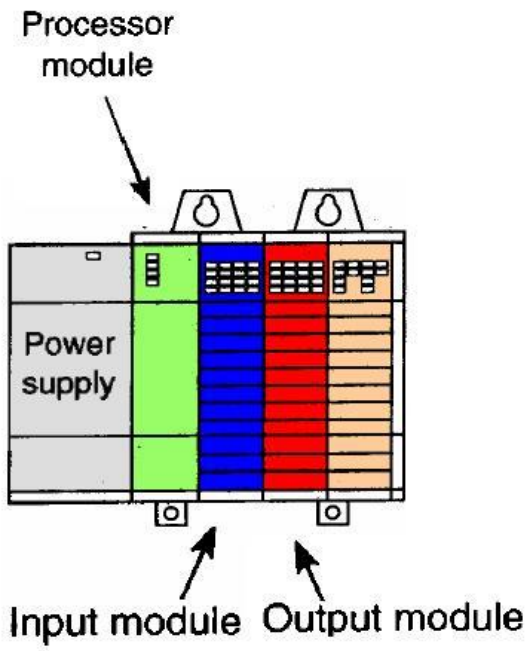
Fixed I/O .

- Is typical of small PLC's
- Comes in one package with no separate removable units
- The processor and I/O are packaged together.
- Lower in cost – but lacks flexibility.



Modular I/0 Modules:

Modular I/O .

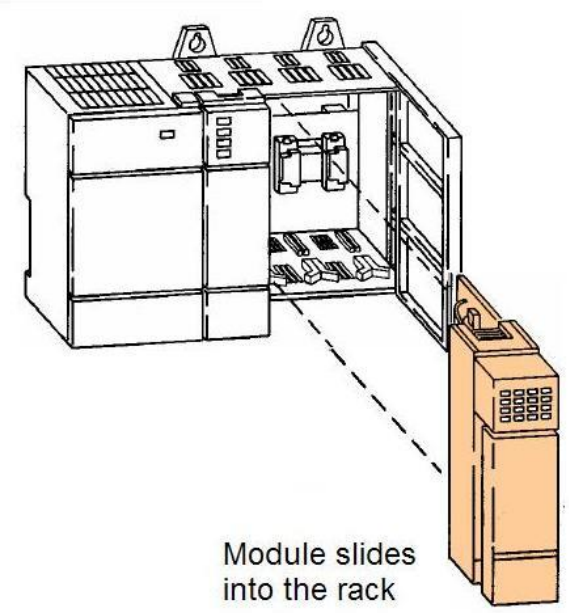


Is divided by compartments into which separate modules can be plugged.

This feature greatly increases your options and the unit's flexibility. You can choose from all the modules available and mix them In any way you desire.

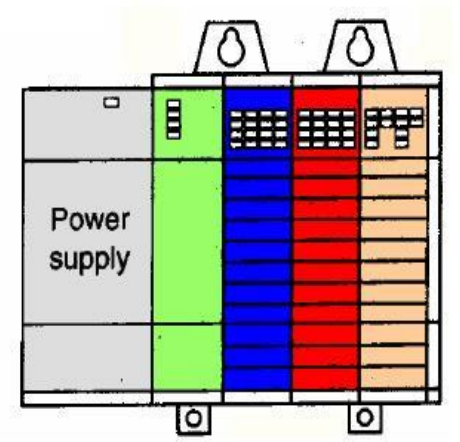
Modular I/O .

When a module slides into the rack, it makes an electrical connection with a series of contacts -called the *backplane.* The backplane is located at the rear of the rack.



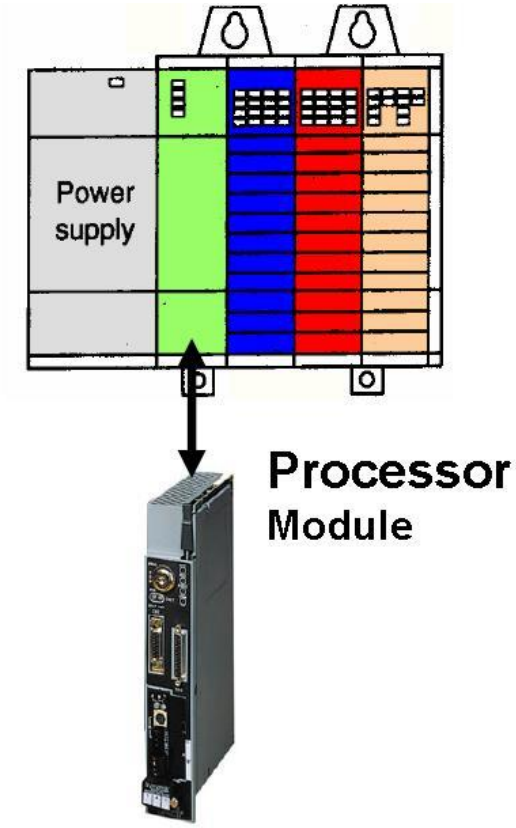
Power Supply:

- Supplies DC power to other modules that plug into the rack.
- In large PLC systems, this power supply does not normally supply power to the field devices.
- In small and micro PLC systems, the power supply is also used to power field devices.

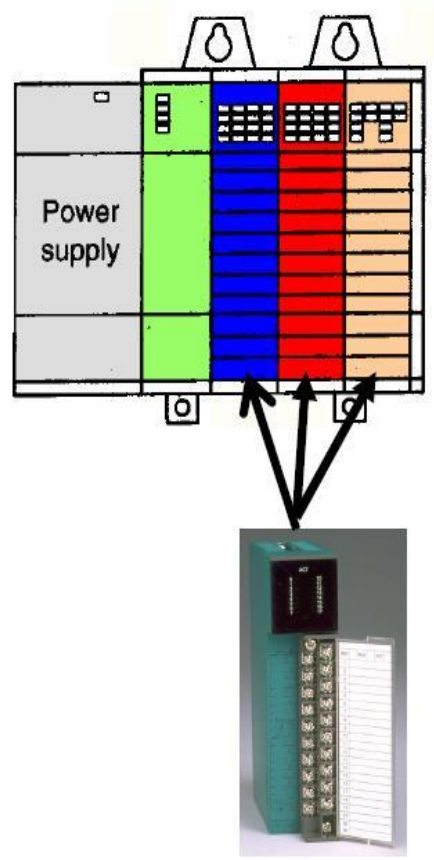


Processor (CPU):

- Is the "brain" of the PLC
- Consists of a microprocessor for Implementing the logic, and supply controlling the communications among the modules.
- Designed so the desired circuit can be entered in relay ladder logic form.
- The processor accepts input data from various sensing devices, executes the stored user program, and sends appropriate output commands to control devices.



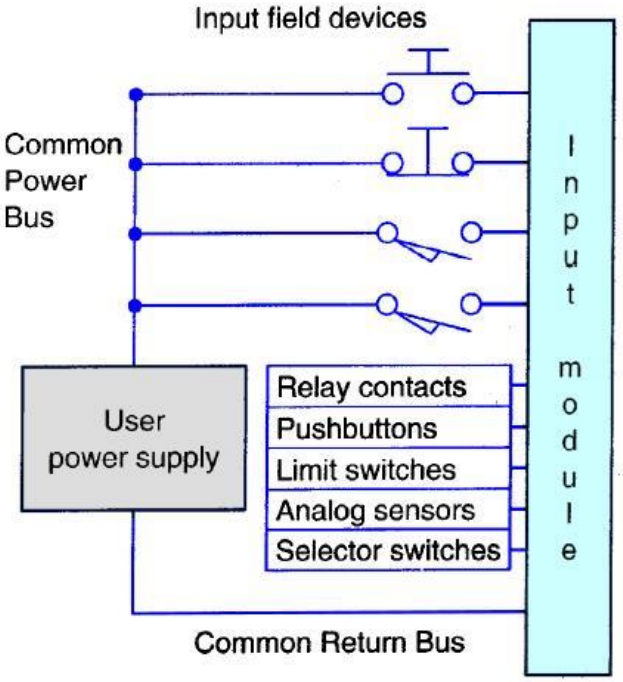
I/0 Sections:



Consists of:

- Input modules supply
- Output modules.

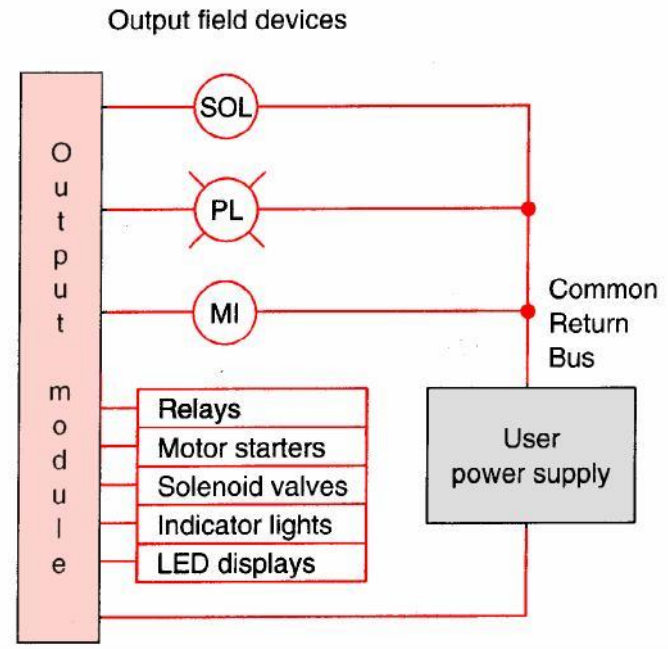
Input Module .



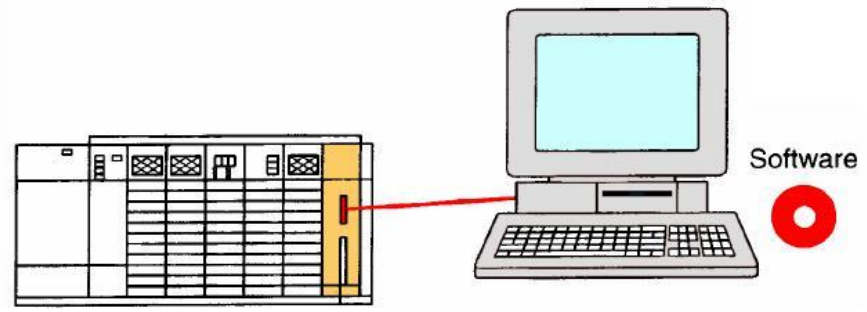
- Forms the interface Power by which input field devices are connected to the controller.
- The terms “field” and the “real world” are sued to distinguish actual external devices that exist and must be physically wired into the system.

Output Module

- Forms the interface by which output field devices are connected to the controller.
- PLC's employ an optical isolator which uses light to electrically isolate the internal components from the input and output terminals.

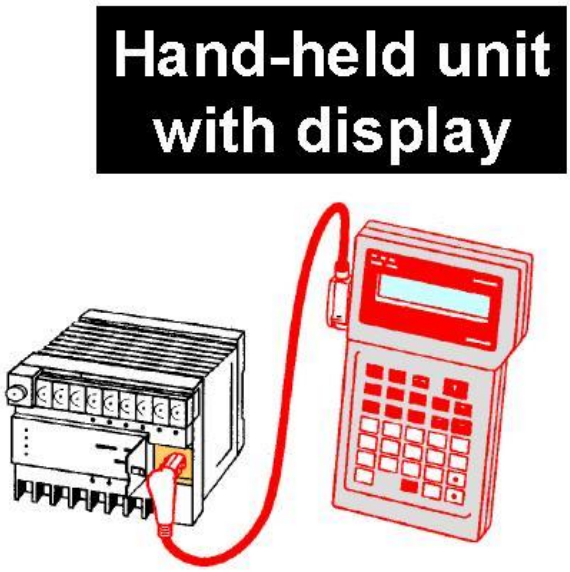


Programming Device:



- A personal computer (PC) is the most commonly used programming device.
- The software allows users to create, edit, document, store and troubleshoot programs.
- The personal computer communicates with the PLC processor via a serial or parallel data communications link

Programming Device:



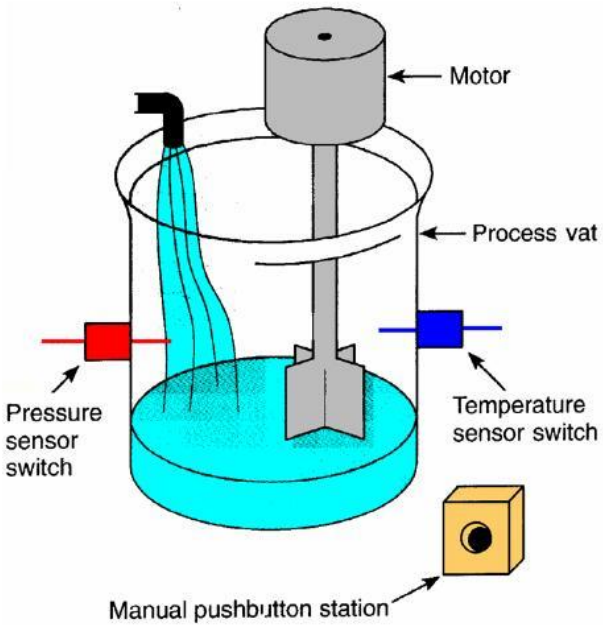
- Hand-held programming devices are sometimes used to program small PLC's
- They are compact, inexpensive, and easy to use, but are not able to display as much logic on screen as a computer monitor
- Hand-held units are often used on the factory floor for troubleshooting, modifying programs, and transferring programs to multiple machines.

PLC Mixer Process Control Problem:

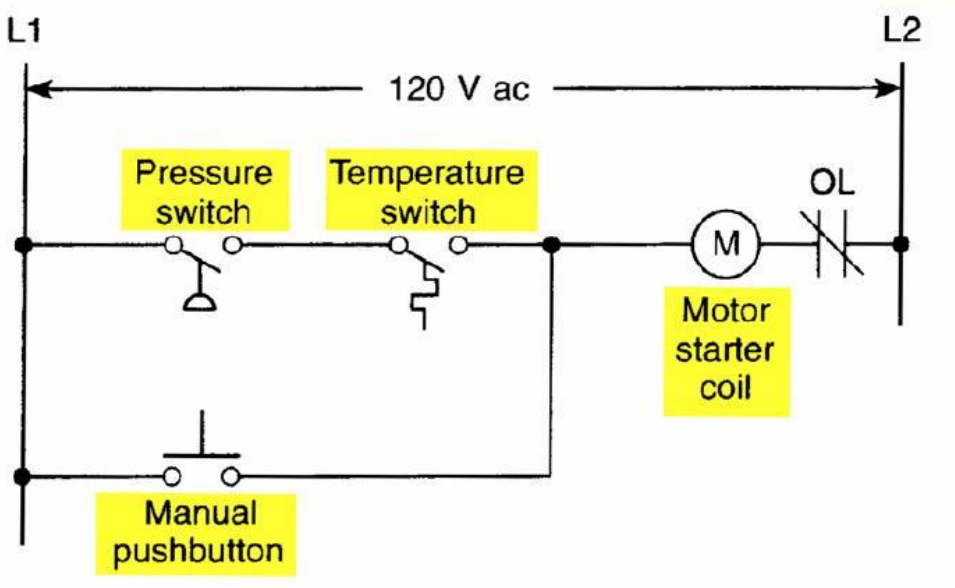
Mixer motor is to automatically stir the liquid in the vat when the temperature and pressure reach preset values.

Alternate manual pushbutton is provided to control the motor.

The temperature and pressure and pressure sensor switches close their respective contacts when conditions reach their preset values.



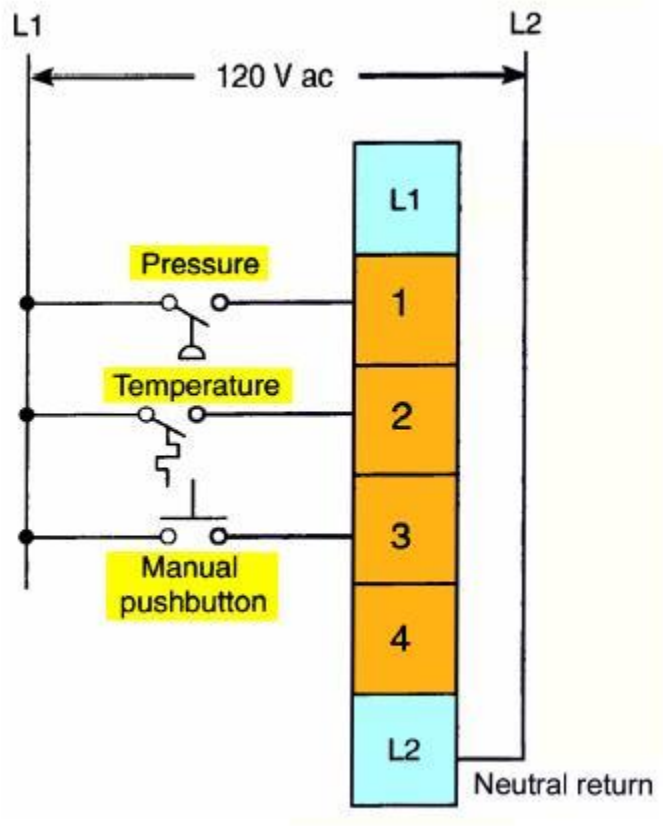
Process Control Relay
Ladder Diagram:



Motor starter coils is energized when both the pressure and temperature switches are closed or when the manual pushbutton is pressed.
The temperature and

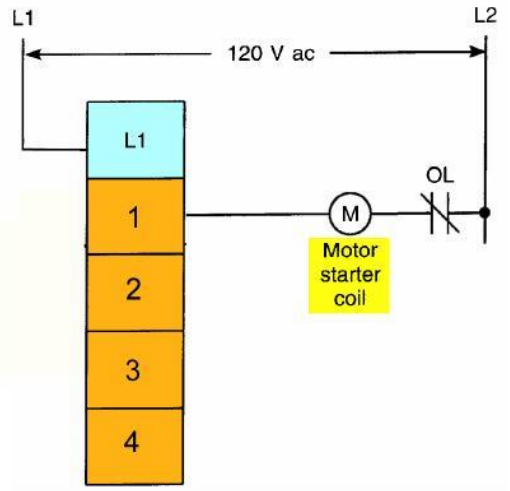
PLC Input Module Connections:

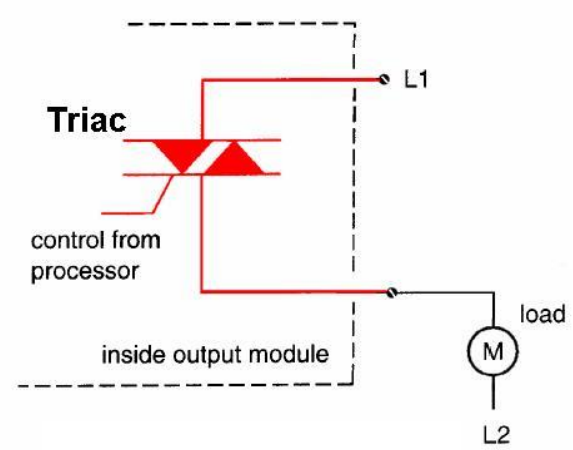
- The same input field devices are used.
- These devices are wired to the input module according to the manufacturer's labeling scheme.



PLC Output Module Connections:

Same output field device is used and wired output module.

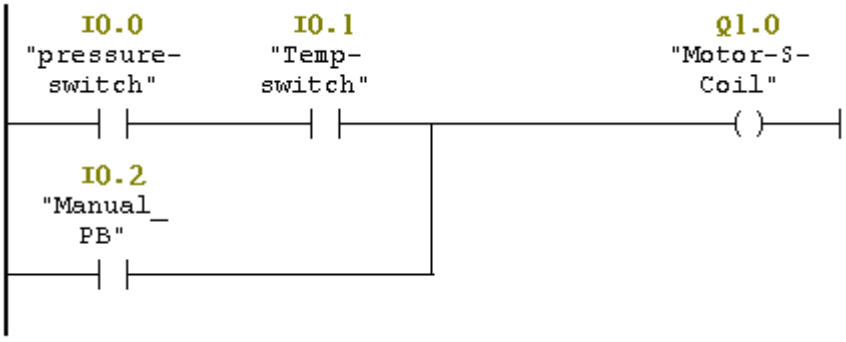


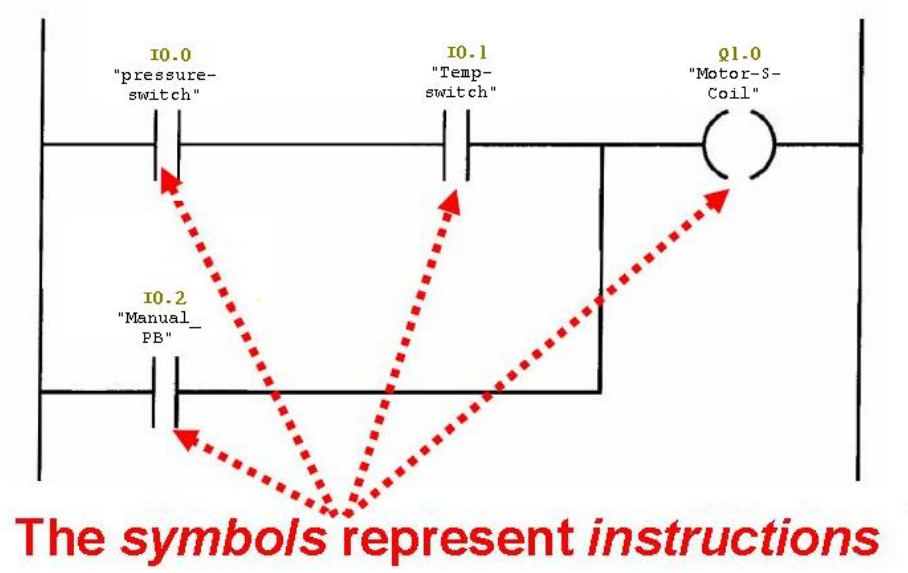


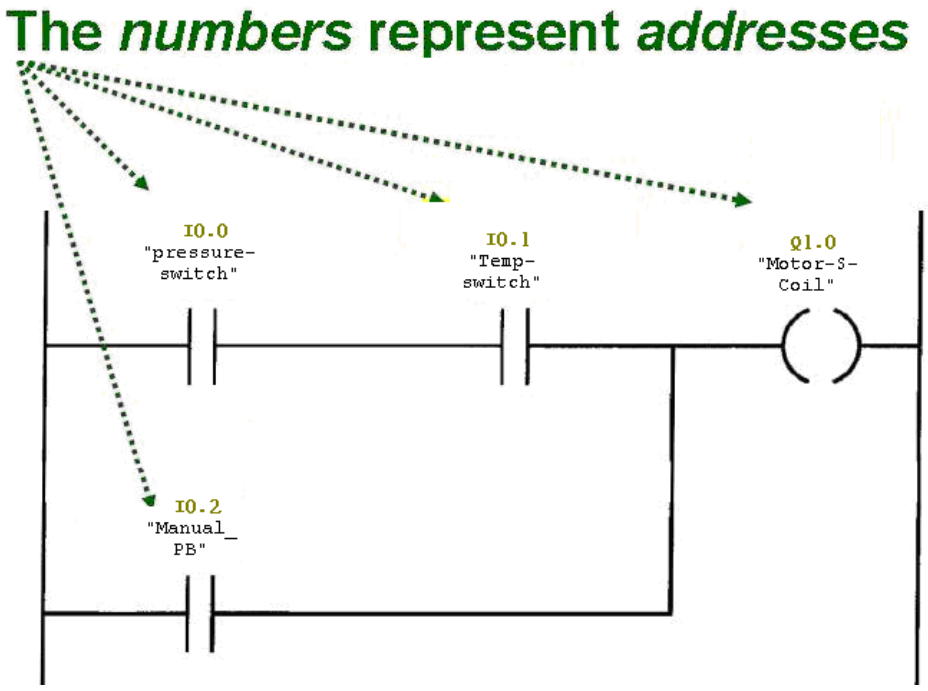
Triac switches motor ON and OFF in accordance with the control signal from the processor.

PLC Ladder Logic Program:

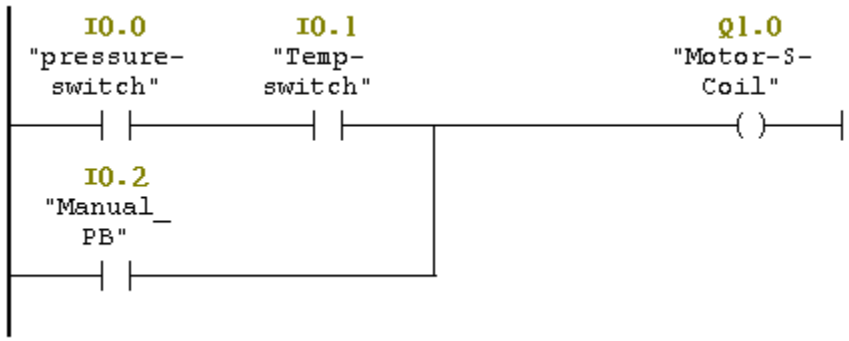
- The format used is similar to that of the hard-wired relay circuit.




I0.0
"pressure-
switch"
I0.1
"Temp-
switch"
Q1.0
"Motor-S-
Coil"
I0.2
"Manual_
PB"
The symbols represent instructions

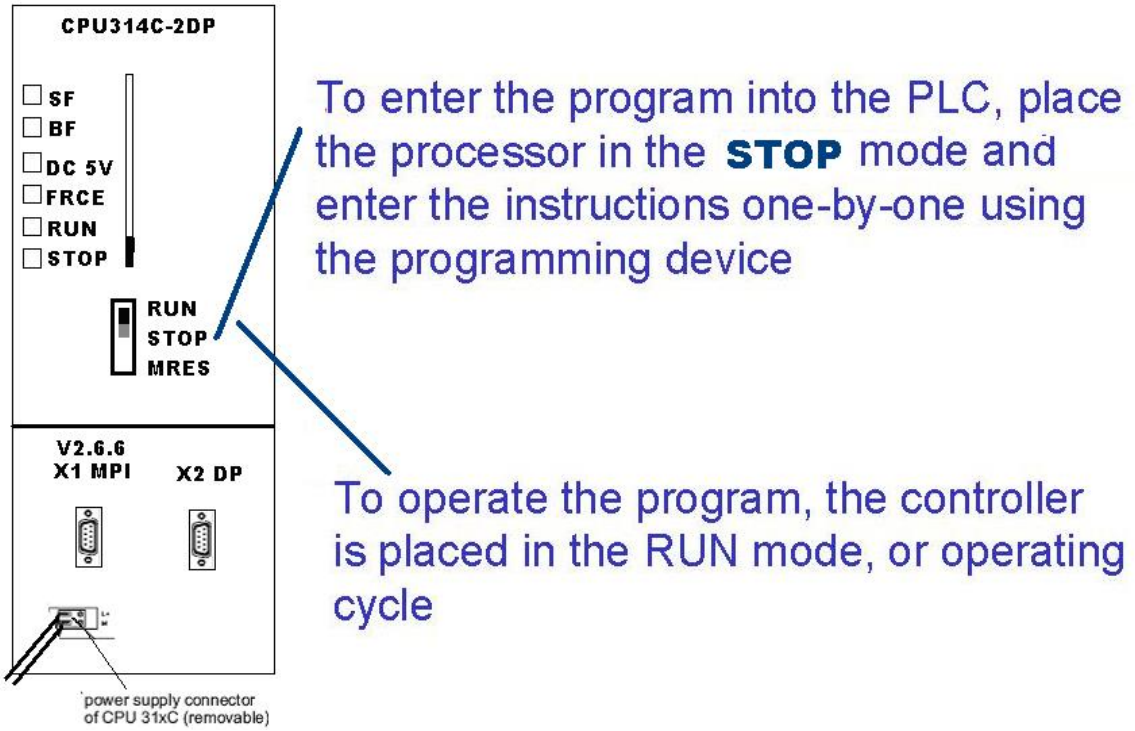

The numbers represent addresses
I0.0
"pressure-
switch"
I0.1
"Temp-
switch"
Q1.0
"Motor-S-
Coil"
I0.2
"Manual_
PB"

PLC ladder logic Program:



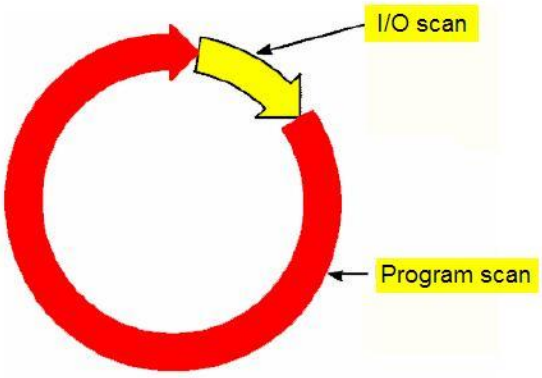
- I/O address format will differ, depending on the PLC manufacturer. You give each input and output device an address. This lets the PLC know where they are physically connected

Entering and Running the PLC Program:

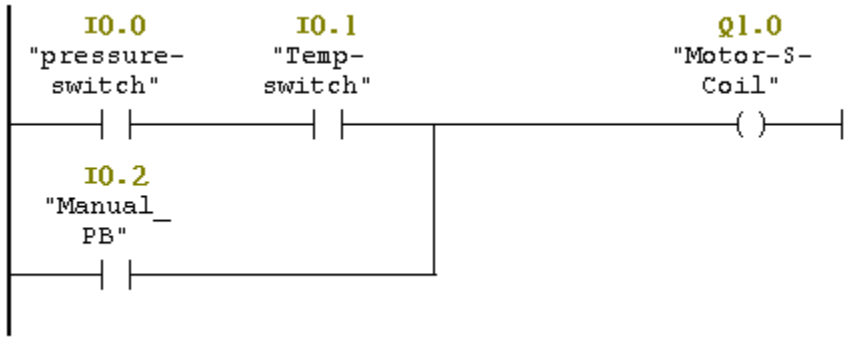


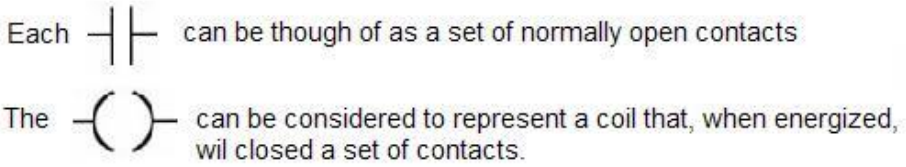
PLC Operating Cycle:

During each operating cycle, the controller examines the status of input devices, executes the user program, and changes outputs accordingly.



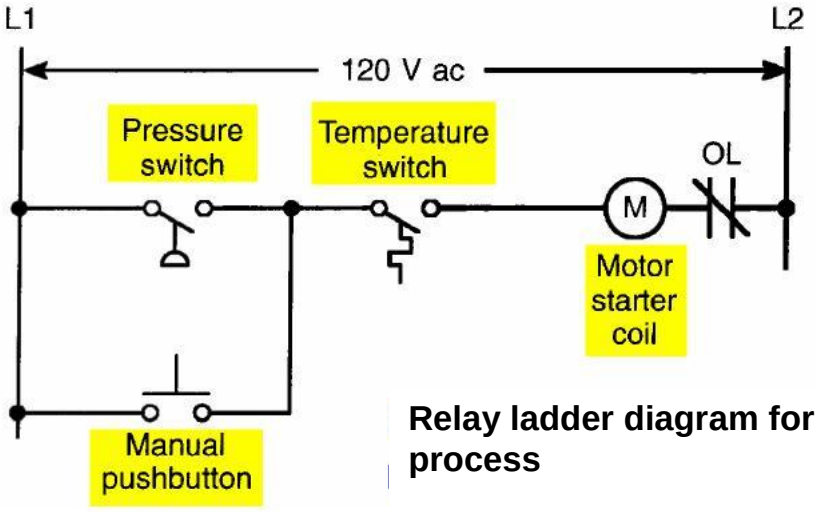
The completion of one cycle of this sequence is called a scan. The scan time, the required for one full cycle, provides measure of the speed of response of the PLC.





Coil O/I is energized when contacts I/1 and I/2 are closed or when contact I/3 is closed. Either of these conditions provides a continuous path from left to right across the rung that includes the coil.

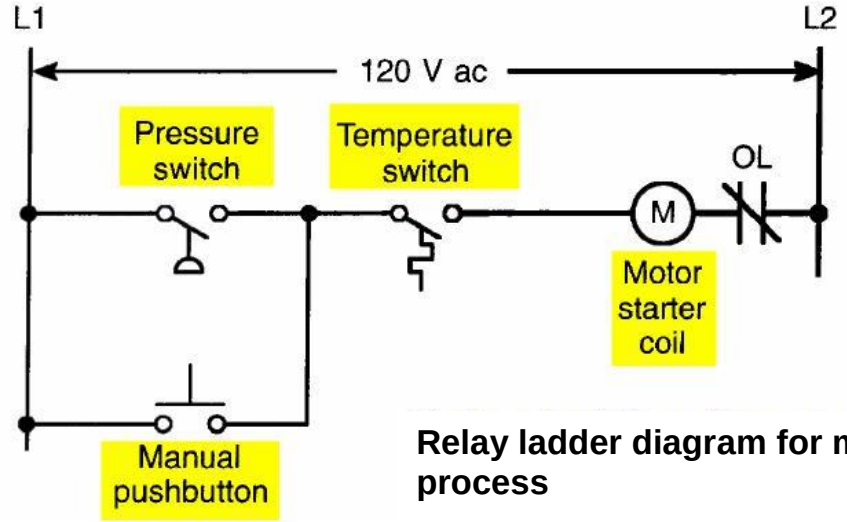
Modifying a PLC Program:



Relay ladder diagram for modified process

- The change requires that the manual pushbutton control should be permitted to operate at any pressure but not unless the specified temperature setting has been reached.

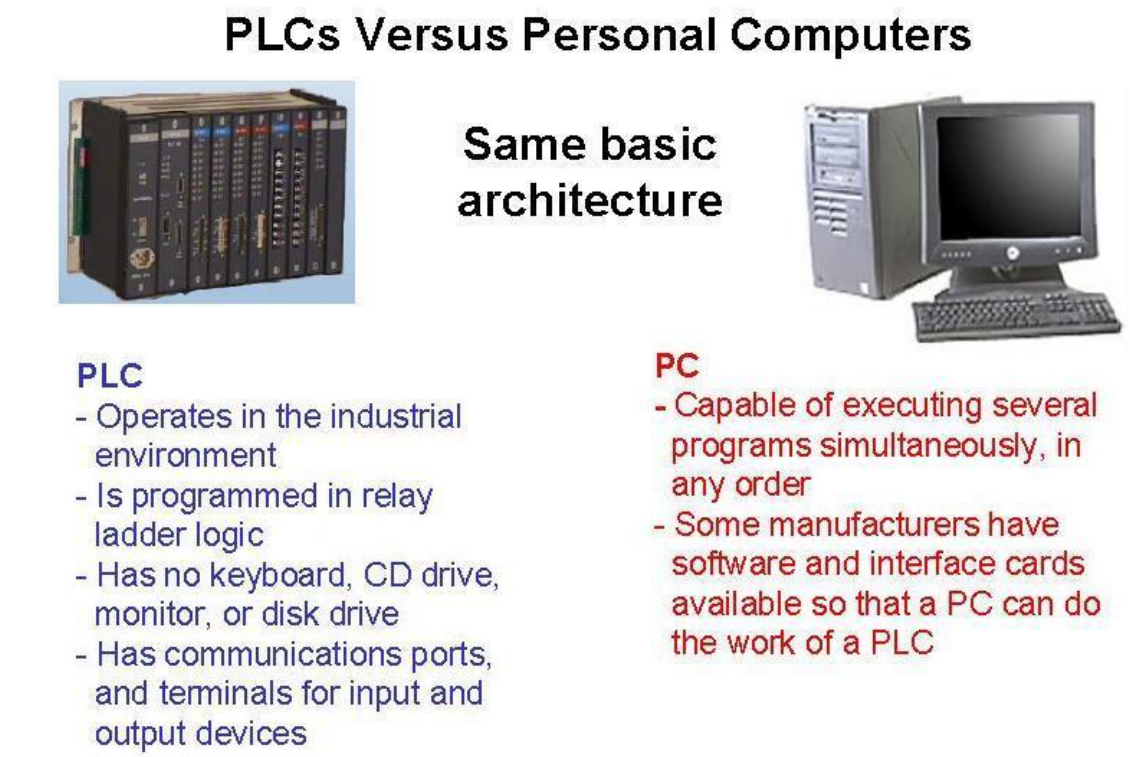
- If a relay system were used, it would require some rewiring of the system, as shown, to achieve the desired change.



Relay ladder diagram for modified process

- If a PLC is used, no rewiring is necessary! The inputs and outputs are still the same. All that is required is to change the PLC program.

PLC's Versus Personal Computers:



PC Based Control Systems:

Advantages:

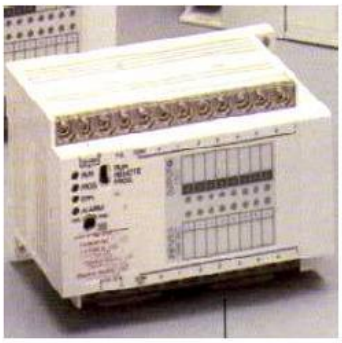
- Lower initial cost
- Less proprietary hardware and software required
- Straightforward data exchange with other systems
- Speedy information processing
- Easy customization

PLC Size Classification:

Criteria

- Number of inputs and outputs (I/O count)
- Cost
- Physical size

NANO PLC
- Smallest sized PLC
- Handles up to 16 I/O points



Micro PLC
- Handles up to 32 I/O points

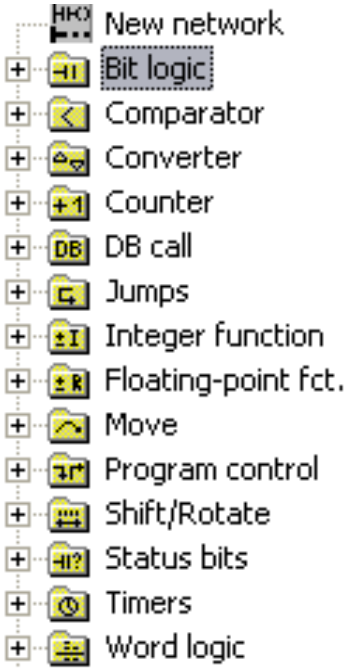
PLC Size Classification

Allen-Bradley SLC-500 Family
- Handles up to 960 I/O points

Allen-Bradley PLC-5 Family
- Handles several thousand I/O points

PLC Instruction Set:

The instruction set for a particular PLC type lists the different types of instructions supported.



An instruction is a command that will cause a PLC to perform a certain predetermined operation.

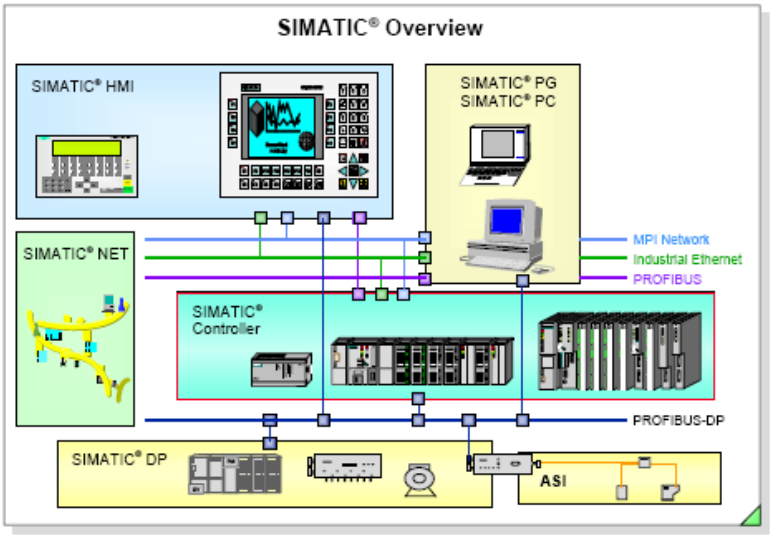
Typical PLC Instructions:

--] [-- NO contact	Examine a bit for an ON condition
--] [-- NC contact	Examine a bit for an OFF condition
--( )-- output coil	Turn ON a bit
--(S)-- output set	Set a bit
--(R)-- output reset	Reset a bit

PART TWO

PLC HARDWARE COMPONENTS
&
SIMATIC S7 SOFTWARE

SIMATIC OVERVIEW



Introduction

In the past, control tasks were solved with individual isolated **P**rogrammable **L**ogic **C**ontrols (PLCs) controlling a machine or process. Today in order for companies to remain competitive, it is not enough to automate only individual processing stations or machines in isolation. The demand for more flexibility with higher productivity can then be fulfilled when the individual machines are integrated in the entire system.

Totally Integrated

Totally Integrated Automation (TIA) provides a common software environment **Automation** that integrates all components, in spite of the diversification of applied technology, into one uniform system. This brings together everything you need to program, configure, operate, handle data, communicate, and maintain your control solutions.

Step 7 SIMATIC Manager, running on Siemens PGs or PCs, provides an integrated set of tools for all system components that allows easy creation, testing, start-up, operation and maintenance of your control solutions. While you are configuring and programming, the Siemens software puts all of your data in a *central database* to which all of the tools have access.

Central Database A common database of all components of Totally Integrated Automation means that data only have to be entered once and are then available for the entire project. The total integration of the entire automation environment is made possible with the help of:

• One common software environment (Step 7 SIMATIC Manager) that integrates all components and tasks into one uniform easy to use system.

• Common data management

• Standard open busses such as Ethernet, PROFIBUS, MPI, AS-interface connect all components to each other, from the management level to the

Hardware and software needed

PC, operating system Windows 2000 Professional starting with SP4 /XP Professional starting with SP1/Server 2003 with 600 MHz and 512 RAM, Free hard disk memory approx. 650 to 900 MB, MS Internet Explorer 6.0

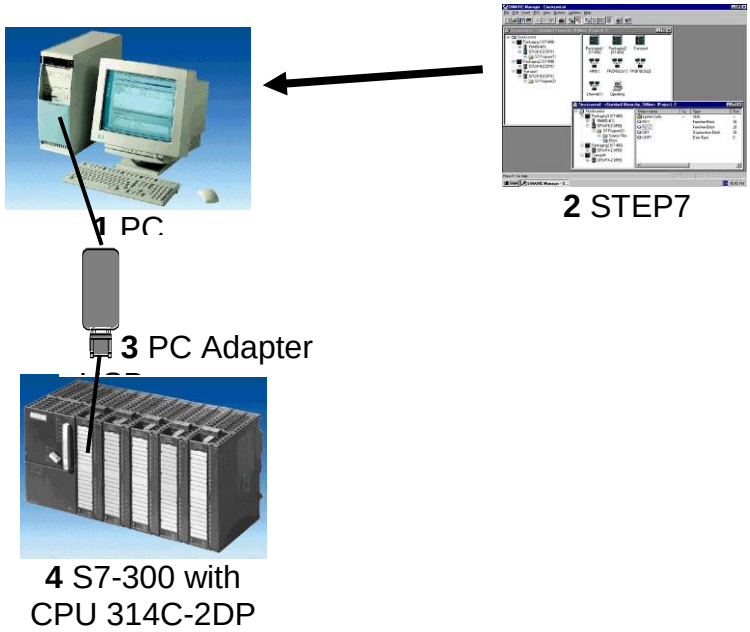
2 Software: STEP7 V 5.4

3 MPI interface for the PC (e.g. PC adapter for USB)

4 SIMATIC S7-300 PLC with the CPU 314C-2DP

Configuration example:

- Power supply unit: PS 307 2A
- CPU: CPU 314-2DP



NOTES ON THE USE OF THE CPU 314C-2DP

The CPU 314C-2DP is shipped with an integrated PROFIBUS DP interface and integrated inputs/outputs.
The following PROFIBUS protocol profiles are available for the CPU 314C-2DP:

- DP interface as master according to EN 50170.
- DP interface as slave according to EN 50170.

PROFIBUS-DP (decentralized peripherals) is the protocol profile for connecting decentralized peripherals/field units with very fast reaction times.

The addresses of the input and output modules of this CPU can be parameterized.

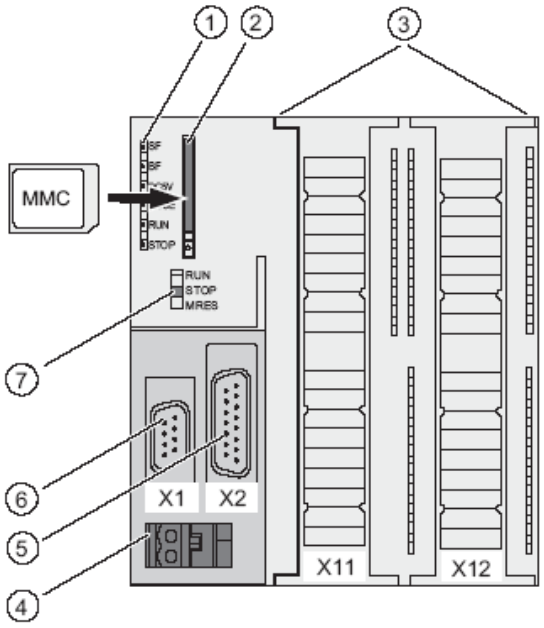
Due to the following performance data, this CPU is especially suitable for training purposes:
- 48 kByte RAM, load memory in the form of a plug-in MicroMemoryCard (MMC), 64 kByte to 4 MByte
- 8192 bytes DI/DO, including 992 bytes central
- 512 bytes AI/AO, including 248 bytes central
- 0.1 ms / 1 K commands
- 256 counters
- 256 timers
- 256 clock memory bytes
- 24 DIs, including 16 which can be used for integrated functions; all can be used as alarm inputs as well
- 16 DOs, integrated; 4 of which are fast outputs
- 4 AIs for current/voltage, 1 AI resistor integrated
- 2 AOs for current/voltage, integrated
- 4 pulse outputs (2.5 kHz)
- 4-channel counting and measuring with 24 V (60 kHz) incremental encoders
- Integrated positioning function

OPERATING THE CPUS 31XC

Operator control and display elements

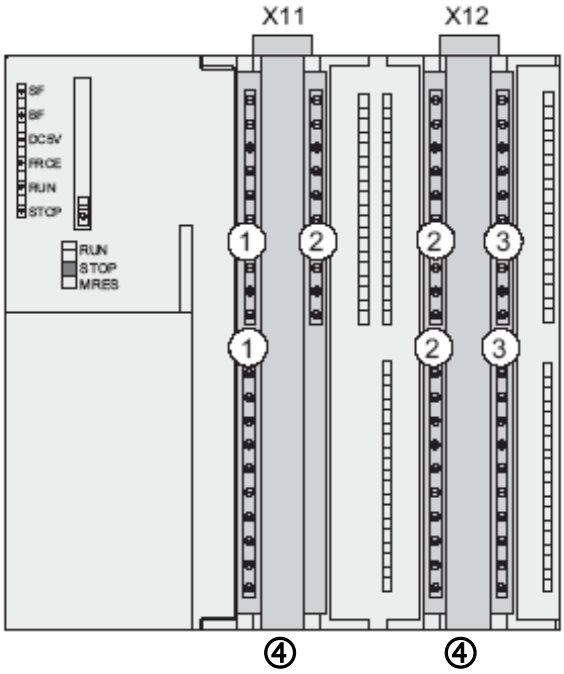
The following illustration shows the operator control and display elements of a CPU 31xC.

The arrangement and number of elements in some CPUs differ from this illustration.



The figures show the following CPU elements:
(1) Status and error displays
(2) Slot for the Micro Memory Card (MMC), incl. the ejector
(3) Connections of the integrated I/O.
(4) Power supply connection
(5) 1st interface X2 (PtP or DP)
(6) 1st interface X1 (MPI)

The following illustration shows the digital and analog inputs/outputs integrated on the CPU.



The figure shows the following integrated I/Os:
(1) Analog I/Os
(2) each with 8 digital inputs
(3) each with 8 digital outputs
(4) Front connectors (front

Status and fault/error displays

The CPU has the following LED displays:

LED designation	Color	Meaning
SF	red	Hardware or software error
BF (for CPUs with DP interface only)	red	Bus error
DC5V	green	5-V power for CPU and S7-300 bus is OK
FRCE	yellow	Force job is active
RUN	green	CPU in RUN The LED flashes during STARTUP at a rate of 2 Hz, and in HOLD state at 0.5 Hz.
STOP	yellow	CPU in STOP and HOLD or STARTUP The LED flashes at 0.5 Hz when the CPU requests a memory reset, and during the reset at 2 Hz.

Slot for the SIMATIC Micro Memory Card (MMC)

A SIMATIC Micro Memory Card (MMC) is used as a memory module for the CPU 31xC. The MMC can be used as a load memory and as a transportable data carrier. The MMC **must** be plugged in before the CPU can be operated because the CPUs 31xC do not have an integrated load memory.

Mode selector

The mode selector can be used to choose the current operating mode of the CPU. The mode selector is designed as a toggle switch with 3 positions.

Positions of the mode selector

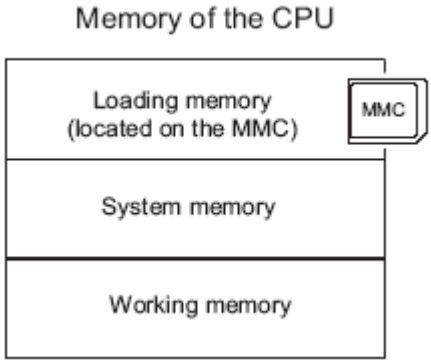
The positions of the mode selector are explained in the same sequence as they occur on the CPU:

Position	Description	Comments
RUN	RUN mode	The CPU is processing the user program
STOP	STOP mode	The CPU is not processing a user program
MRES	Memory Reset	Button position of the operating mode switch for a memory reset of the CPU. A CPU memory reset requires a specific operating sequence (refer to the Installation Manual, Chapter *Commissioning*)

MEMORY AREAS OF THE CPU 31XC

Introduction

The memory of the CPU 31xC can be divided into three areas:



Note

Only with the MMC plugged in is it possible to load user programs and therefore operate the CPU 31xC

Load memory

The load memory is located on a SIMATIC Micro Memory Card (MMC). Its size is exactly the same as that of the MMC. It is used for storing code blocks and data blocks as well as system data (configuration, connections, module parameters, etc.).
Blocks that are marked as not being relevant to program execution are exclusively stored in the load memory. In addition, the complete planning data for a project can be stored on the MMC.

RAM

The RAM is integrated on the CPU and cannot be expanded. It is used for processing the code and processing the data of the user program. The program is executed exclusively in the RAM and the system memory. Once the MMC has been plugged in, the RAM of the CPU is retentive.

System memory

The system memory is integrated on the CPU and cannot be expanded. It contains

- the operands area for clock memories, timers and counters
- the process images of the inputs and outputs
- the local data

Retentivity

Your CPU 31xC has retentive memory. Retentivity is implemented on the MMC and on the CPU. Due to this retentivity, the contents of the retentive memory are retained even after the mains supply has been switched off and the CPU has been restarted (warm restart).

Load memory

You program in the load memory (MMC) is always retentive. During loading, it is stored on the MMC, is powerfail-proof and cannot be cleared.

Work memory (RAM)

Your data in the work memory are backed up on the MMC in the event that the mains supply is switched off. The contents of data blocks are therefore always retained.

System memory

With regard to clock memories, timers and counters, you configure (properties of the CPU, Retentivity tab) which parts are to be retentive and which are to be initialized with "0" when the system is restarted (warm restart).

The diagnostic buffer, MPI address (and baud rate) as well as the runtime meter are generally stored in the retentive memory area on the CPU. The retentive area for the MPI address and the baud rate ensure that your CPU is still able to communicate after a power failure, a complete memory reset or loss of the communication parameters (because the MMC was removed, or the communication parameters were deleted).

Retention of the memory objects

The following table shows which memory objects are retained when transitions between operating modes occur.

Memory Object	**Operating Mode Transition**		
	PowerOn/Off	STOP → RUN	Memory Reset
User Program/User Data (load memory)	X	X	X
Actual values of the DBs	X	X	-
Flags, timers and counters configured as retentive	X	X	-
Diagnostic buffer, hours run meter	X	X	X
MPI address, baud rate	X	X	X

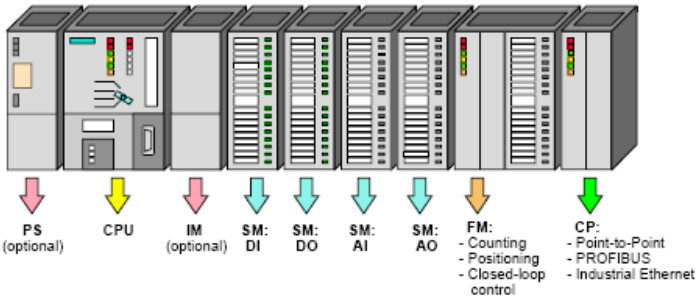
X = retentive; - = not retentive

S7-300 Modules

Features

- Modular small control system for the lower performance range
- Performance-graded range of CPUs
- Extensive selection of modules
- Expandable design with up to 32 modules
- Backplane bus integrated in the modules
- Can be networked with - Multipoint interface (MPI),
  - PROFIBUS or
  - Industrial Ethernet.
- Central PG/PC connection with access to all modules
- No slot restrictions
- Configuration and parameter setting with the help of the "HWConfig" tool.

S7-300™: Modules



Signal Modules (SM)

- Digital input modules: 24 VDC, 120/230 VAC
- Digital output modules: 24 VDC, Relay
- Analog input modules: Voltage, current, resistance, and thermocouple
- Analog output modules: Voltage, current

Interface Modules (IM) The IM360/IM361 and IM365 make multi-tier configurations possible. the interface modules loop the bus from one tier to the next.

Dummy Modules (DM) The DM 370 dummy module reserves a slot for a signal module whose parameters **have** not yet been assigned. A dummy module can also be used, for example, to reserve a slot for Installation of an interface module at a later date.

Function Modules (FM) Perform "special functions":

- Counting
- Positioning
- Closed-loop control.

Communication Processors (CP) Provide the following networking facilities:
- Point-to-Point connections
- PROFIBUS
- Industrial Ethernet.

Accessories Bus connectors and front connectors

S7-300™: CPU Design

Mode Selector
- MRES = Memory reset function (Module Reset)
- STOP = Stop mode, the program is not executed.
- RUN = Program execution, read-only access possible from PG.
- RUN-P = Program execution, read/write access possible from PG.

Status Indicators (LED's)
- SF = Group error; internal CPU fault or fault in module with diagnostics capability.
- BATF = Battery fault; battery empty or non-existent.
- DC5V = Internal 5 VDC voltage indicator.
- FRCE = FORCE; indicates that at least one input or output is forced.
- RUN = Flashes when the CPU is starting up, then a steady light in Run mode.
- STOP = Shows a steady light in Stop mode.
  Flashes slowly for a memory reset request,
  Flashes quickly when a memory reset is being carried out,
  Flashes slowly when a memory reset is necessary because a memory card has been inserted.

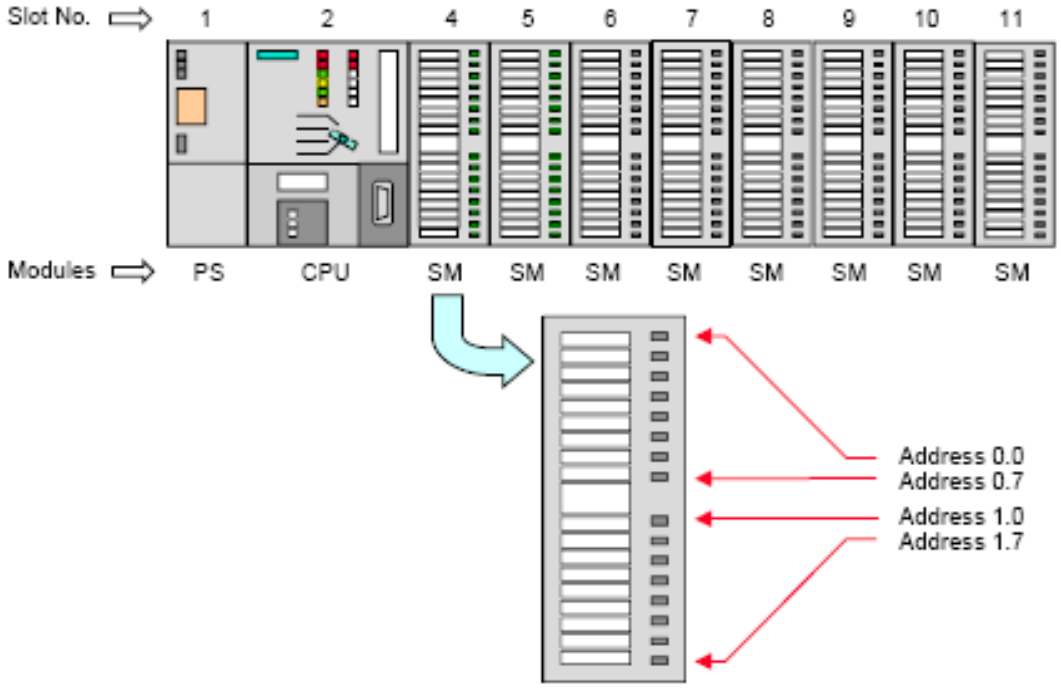
Memory Card A slot is provided for a memory card. The memory card saves the program contents in the event of a power outage without the need for a battery.

Battery Compartment There is a receptacle for a lithium battery under the cover. The battery provides backup power to save the contents of the RAM in the event of a power outage.

MPI Connection Connection for a programming device or other device with an MPI interface.

DP Interface Interface for direct connection of distributed I/Os to the CPU.

Addressing S7-300™ Modules



Slot Numbers The slot numbers in the rack of an S7-300™ simplify addressing in the S7-300™ environment. The position of the module in the rack determine the First address on a module.

Slot 1 Power supply. This is the first slot by default.
A power supply module is not absolutely essential. An S7-300™ can also be supplied with 24V directly.

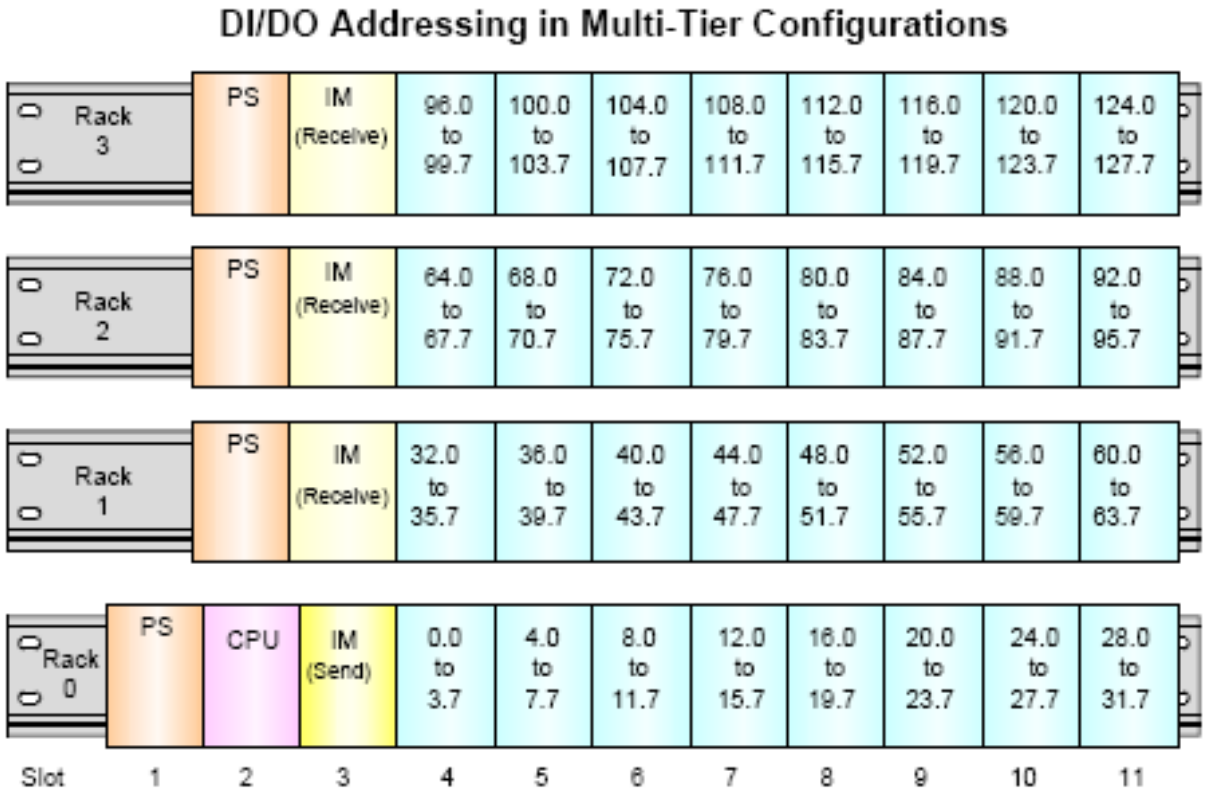
Slot 2 Slot for the CPU.

Slot 3 Logically reserved for an interface module (IM) for multi-tier configurations using expansion racks. Even if no IM is installed, it must be included for addressing purposes.
You can physically reserve the slot (such as for installing an IM at a later date) if you insert a DM370 dummy module.

Slots 4-11 Slot 4 is the first slot that can be used for I/O modules, communications processors (CP) or function modules (FM).

Addressing examples:
- A DI module in slot 4 begins with the byte address 0 .
- The top LED of a DO module in slot 6 is called Q8.0 .

Note Four byte addresses are reserved for each slot. When 16-channel DI/DO modules are used, two byte addresses are lost in every slot!

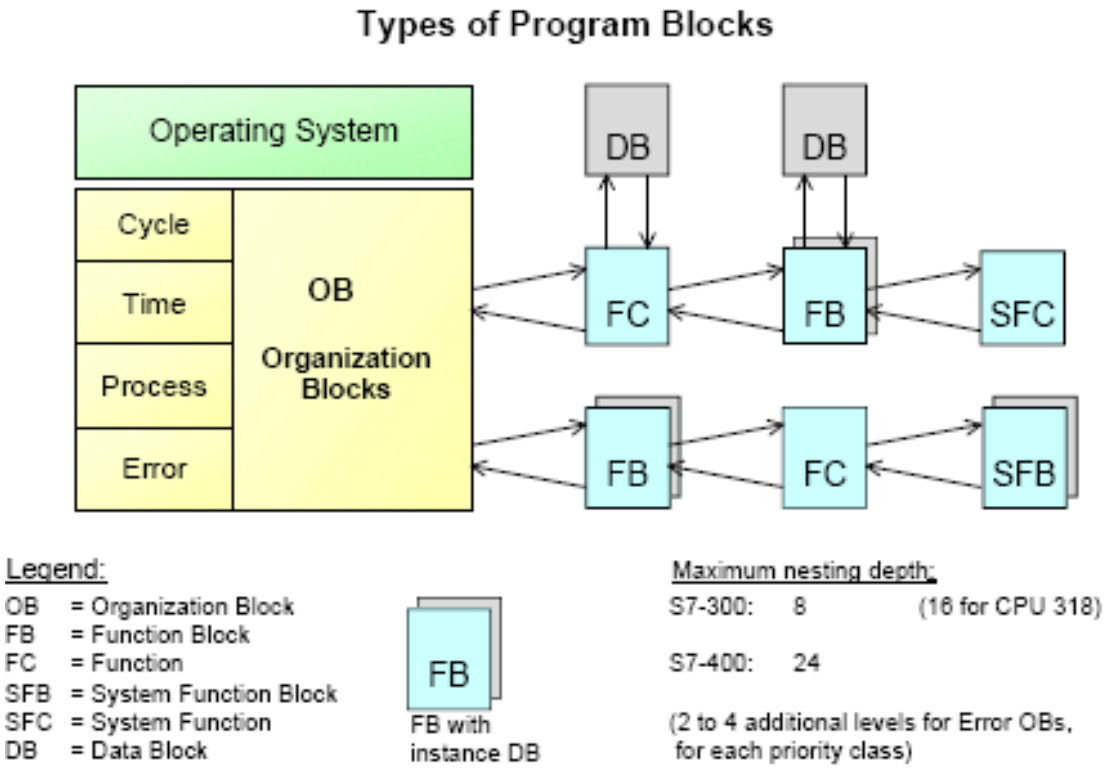


Multi-Tier Configurations

The slots also have fixed addresses in a multi-tier configuration.

Examples:

- Q7.7 is the last bit of a 32-channel DO module plugged into slot 5 of rack 0.
- IB105 is the second byte of a DI module in slot 6 of rack 3.
- QW60 is the first two bytes of a DO module in slot 11 of rack 1.
- ID80 is all four bytes of a 32-channel DI module in slot 8 in rack 2.



Blocks
The programmable logic controller provides various types of blocks in which the user program and the related data can be stored. Depending on the requirements of the process, the program can be structured in different blocks.

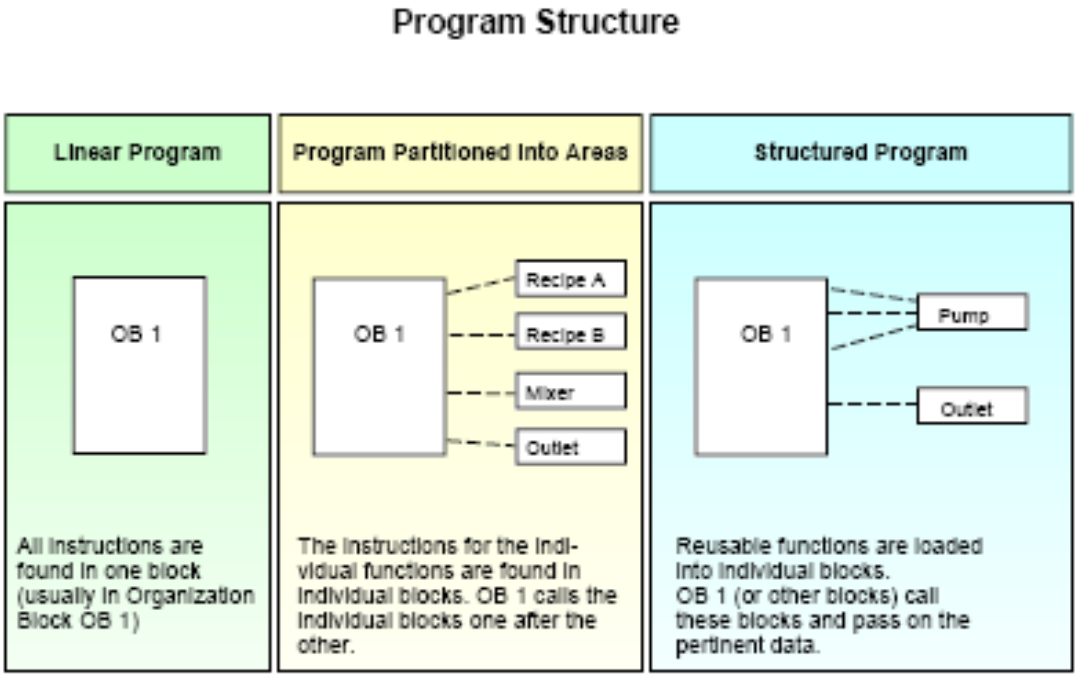
Organization
Organization blocks (OBs) form the interface between the operating system and **Block** the user program. The entire program can be stored in OB1 that is cyclically **OB** called by the operating system (linear program) or the program can be divided and stored in several blocks (structured program).

Function
A function (FC) contains a partial functionality of the program. It is possible to **FC, SFC** program functions so that they can be assigned parameters. As a result, functions are also suited for programming recurring, complex partial functionalities such as calculations.
System functions (SFC) are parameter-assignable functions integrated in the CPU‘s operating system. Both their number and their functionality are fixed. More information can be found in the Online Help.

Function Block
Basically, function blocks offer the same possibilities as functions. In addition, **FB, SFB** function blocks have their own memory area in the form of instance data blocks. As a result, function blocks are suited for programming frequently recurring, complex functionalities such as closed-loop control tasks.
System function blocks (SFB) are parameter-assignable functions integrated in the CPU‘s operating system. Both their number and their functionality are fixed. More information can be found in the Online Help.

Data Blocks
Data blocks (DB) are data areas of the user program in which user data are **DB** managed in a structured manner.

Permissible Operations
You can use the entire operation set in all blocks (FB, FC and OB).



Linear Program
The entire program is found in one continuous program block. This model resembles a hard-wired relay control that was replaced by a programmable logic controller. The CPU processes the individual instructions one after the other.

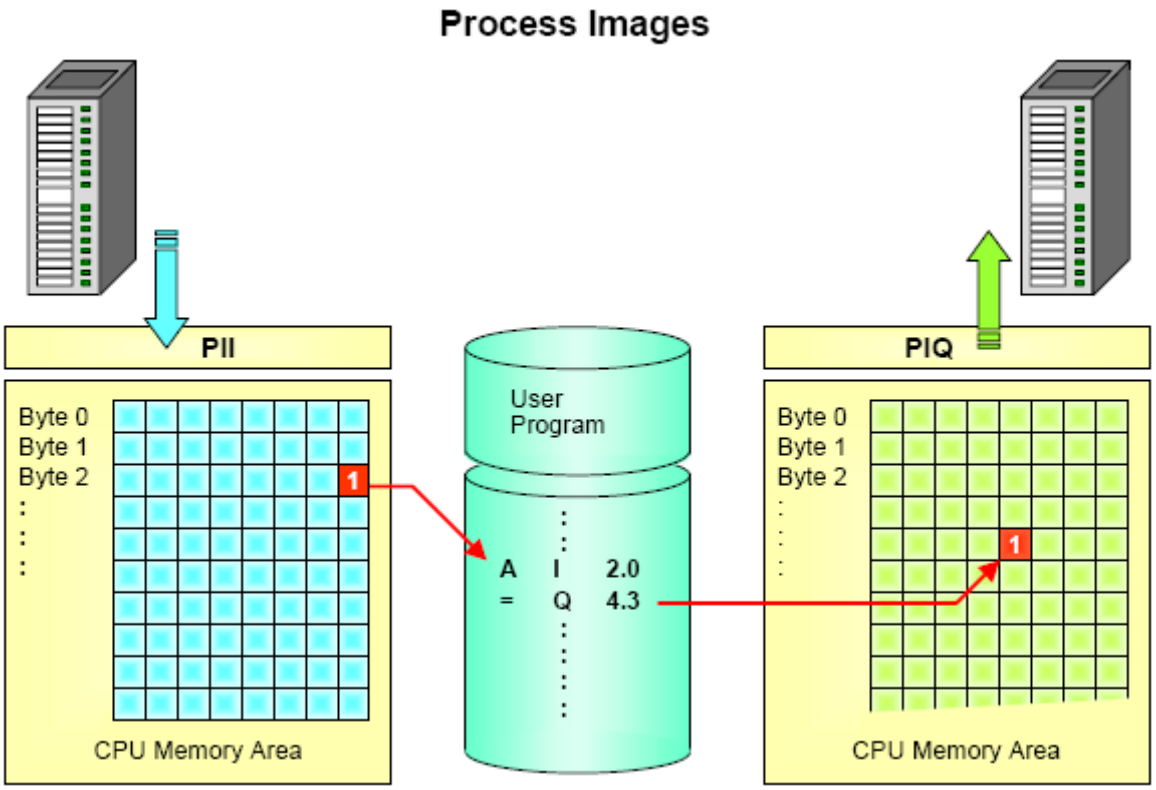
Partitioned Program
The program is divided into blocks, whereby every block only contains the program for solving a partial task. Further partitioning through networks is possible within a block. You can generate network templates for networks of the same type. The OB 1 organization block contains instructions that call the other blocks in a defined sequence.

Structured Program
A structured program is divided into blocks. The code in OB1 is kept to a minimum with calls to other blocks containing code. The blocks are parameter assignable. These blocks can be written to pass parameters so they can be used universally. When a parameter assignable block is called, the programming editor lists the local variable names of the blocks. Parameter values are assigned in the calling block and passed to the function or function block.

Example:
• A "pump block" contains instructions for the control of a pump.

• The program blocks, which are responsible for the control of special pumps, call the "Pump block" and give it information about which pump is to be controlled with which Parameters.

• When the "pump block" has completed the execution of its instructions, the program returns to the calling block (such as OB 1), which continues processing the calling block‘s instructions.



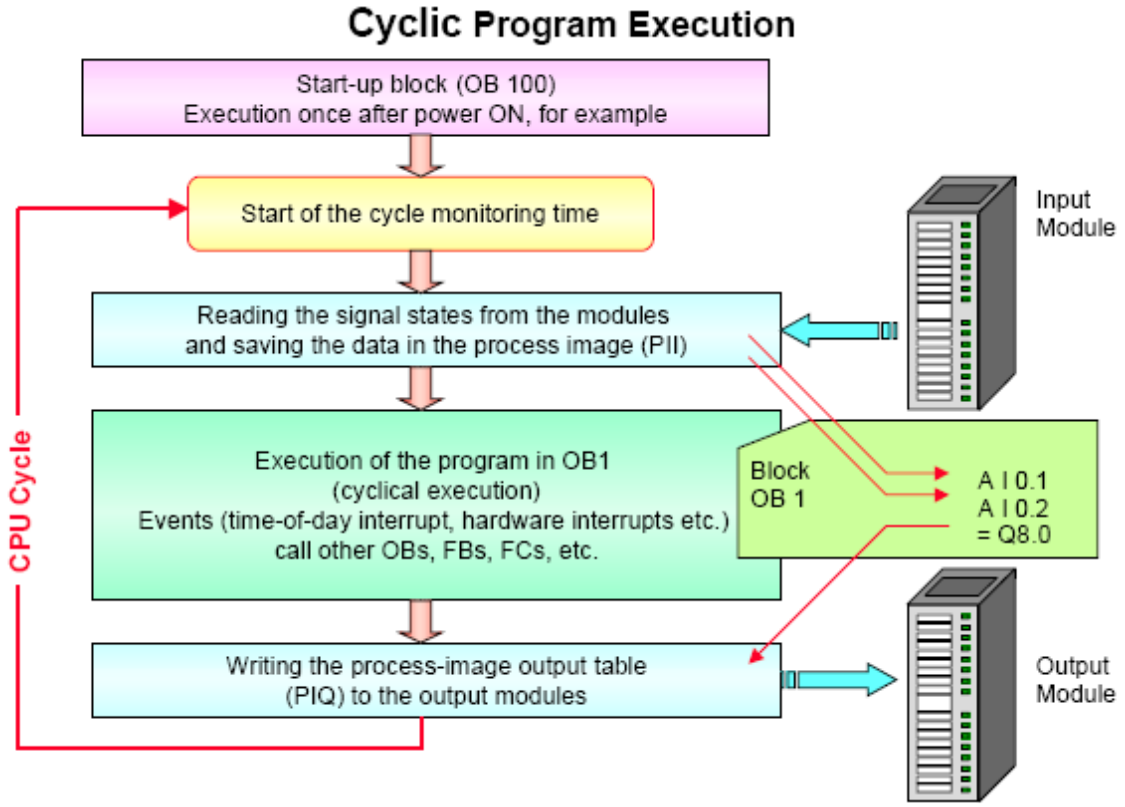
Introduction
The CPU checks the status of the inputs and outputs in every cycle. There are specific memory areas in which the module‘s binary data are stored: PII and PIQ. The program accesses these registers during processing.

PII
The **P**rocess-**I**mage **I**nput table is found in the CPU‘s memory area. The signal state of all inputs is stored there.

PIQ
The **P**rocess-**I**mage Output (**Q**) table contains the output values that result from the program execution. These output values are sent to the actual outputs (Q) at the end of the cycle.

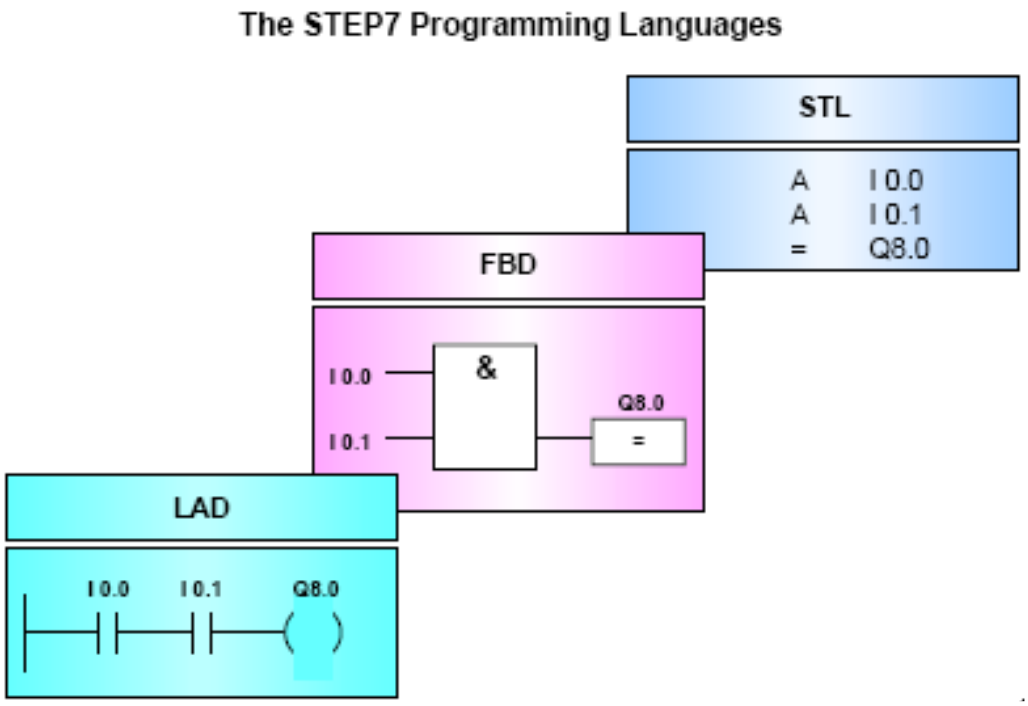
User Program
When you check inputs in the user program with, for example, A I 2.0, the last state from the PII is evaluated. This guarantees that the same signal state is always delivered throughout one cycle.

Note
Outputs can be assigned as well as checked in the program. Even if an output is assigned a state in several locations in the program, only the state that was assigned last is transferred to the appropriate output module.



Starting
The CPU carries out a complete restart (with OB100) when switching on or when switching from STOP --> RUN. During a complete restart, the operating system:
• deletes the non-retentive bit memories, timers and counters.
• deletes the interrupt stack and block stack.
• resets all stored hardware interrupts and diagnostic interrupts.
• starts the scan cycle monitoring time.

Scan Cycle
The cyclical operation of the CPU consists of three main sections, as shown in the diagram above. The CPU:
• checks the status of the input signals and updates the process-image input table.
• executes the user program with the respective instructions.
• writes the values from the process-image output table into the output modules.



Introduction
There are several programming languages in STEP 7 that can be used depending on preference and knowledge. By adhering to specific rules, the program can be created in Statement List and later converted into another programming language.

LAD
Ladder Diagram is very similar to a circuit diagram. Symbols such as contacts and coils are used. This programming language often appeals to those who have a drafting or electrical background.

STL
The Statement List consists of STEP 7 instructions. You can program fairly freely with STL. This programming language is preferred by programmers who are already familiar with other programming languages.

FBD
The Function Block Diagram uses “boxes” for the individual functions. The character in the box indicates the function (such as & --> AND Logic Operation).This programming language has the advantage that even a “non-programmer” can work with it. Function Block Diagram is available as of Version 3.0 of the STEP7 Software.

Absolute and Symbolic Addressing

```
A      I 0.0
=      Q4.1
A      I 0.4
=      Q8.5
Call   FC18
```

```
A      "T_System_ON"
=      "L_SYSTEM"
A      "S_M/A_ModeSelect"
=      "K_RT"
Call   "FC_Count"
```

Symbol	Address	Data Type	Comment
K_RT	Q8.5	BOOL	Run Conveyor Right
FC_Count	FC18	FC18	Count Transported Parts
T_System_ON	I 0.0	BOOL	System ON Switch, Momentary Contact
L_SYSTEM	Q4.1	BOOL	System ON Light
S_M/A_ModeSelect	I 0.4	BOOL	Operating Mode Man=0/Auto=1 Selector Switch

(max. 24 characters) (max. 80 characters)

Absolute Addressing

In absolute addressing, you specify the address (such as input I 1.0) directly. In this case you don‘t need a symbol table, but the program is harder to read.

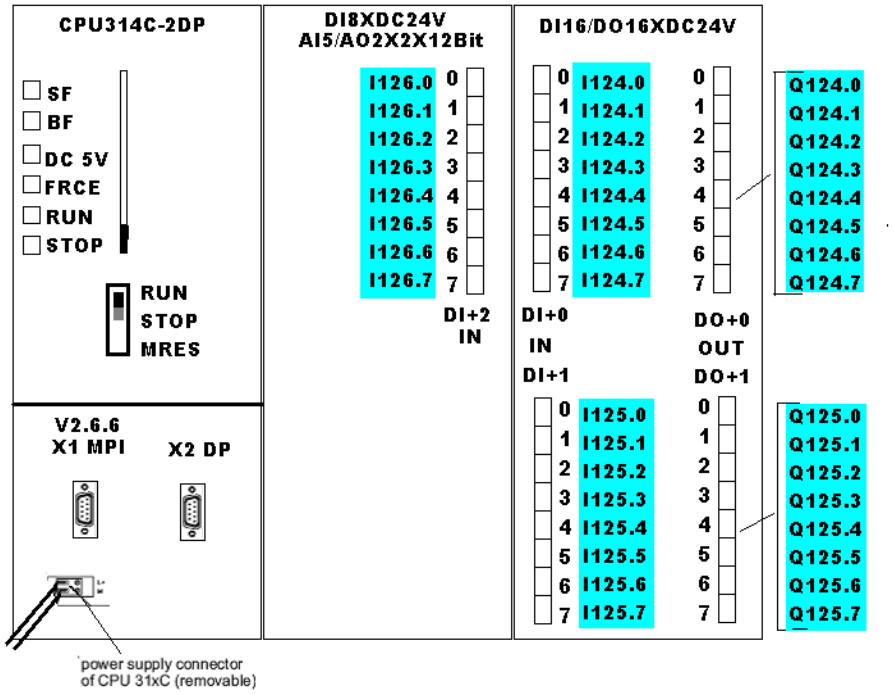
Symbolic Addressing

In symbolic addressing, you use symbols (such as MOTOR_ON) instead of the absolute addresses. You store the symbols for inputs, outputs, timers, counters, bit memories and
blocks in the symbol table.

Note

When you enter symbol names, you don‘t have to include quotation marks. The Program Editor adds these for you.

Configuration of S7-314C-2DP at the LAB

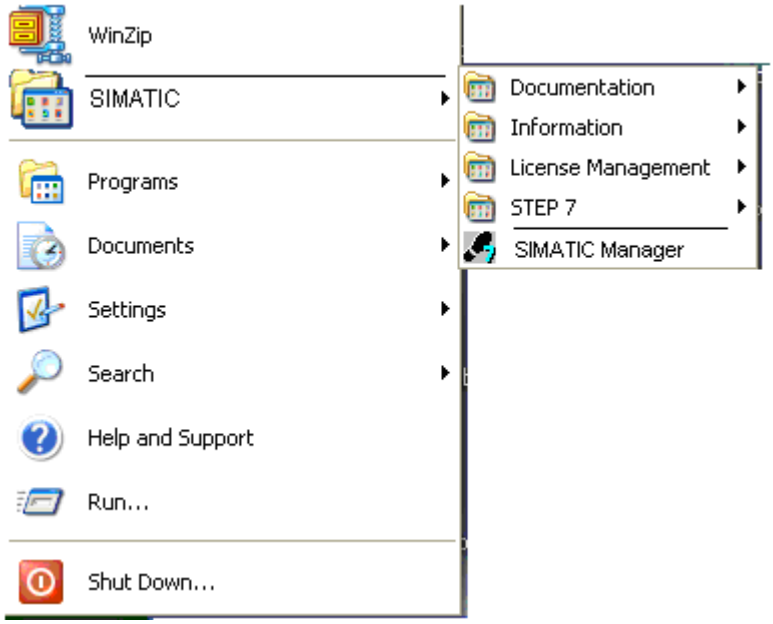


NEW PROJECT CREATION USING SIMATIC MANAGER

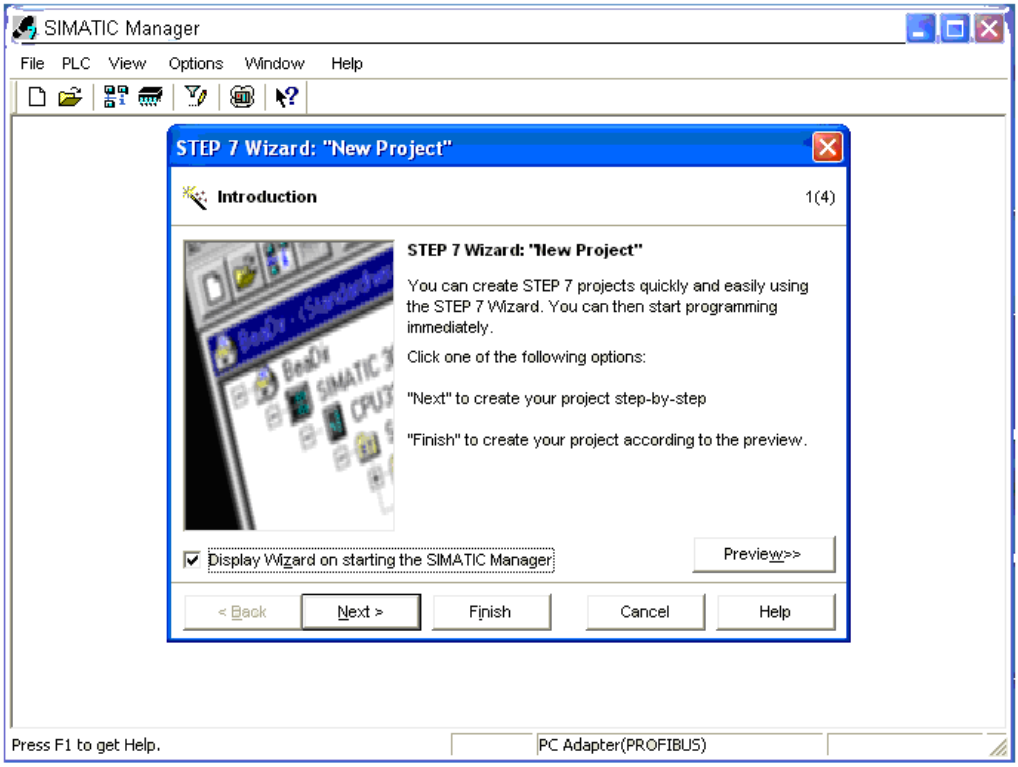
Procedure to run the Simatic manager and How to create a new project

To run the simatic manager, follow the procedure illustrated below.

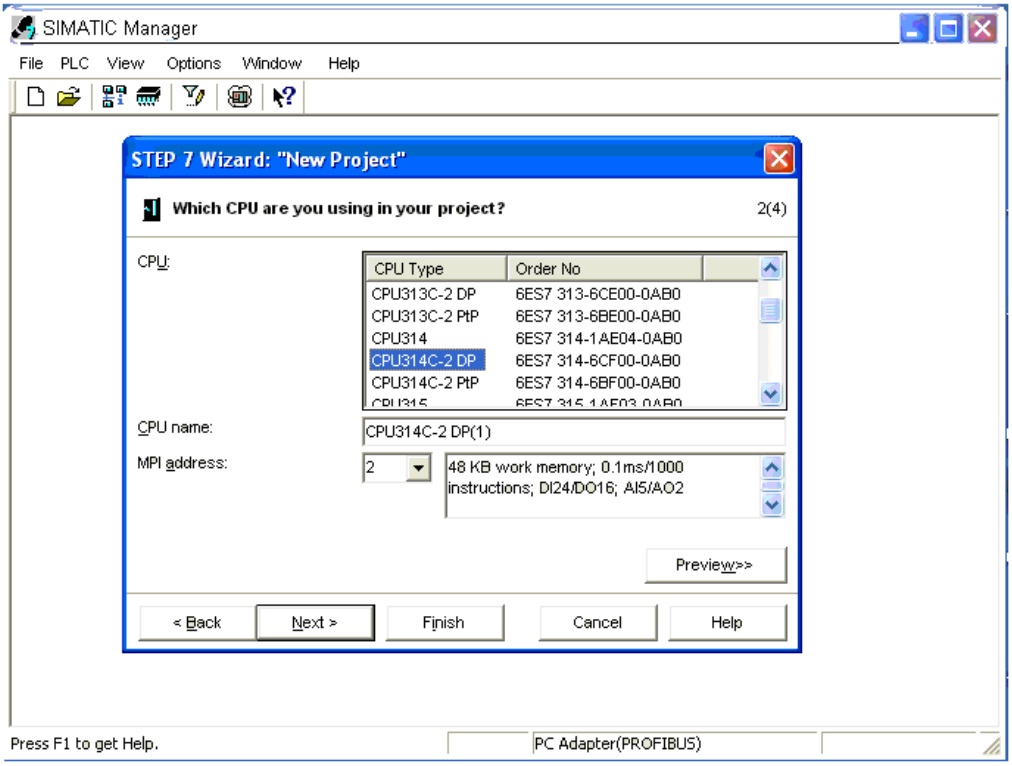
Click on **simatic Manager** under Simatic on the start up menu.



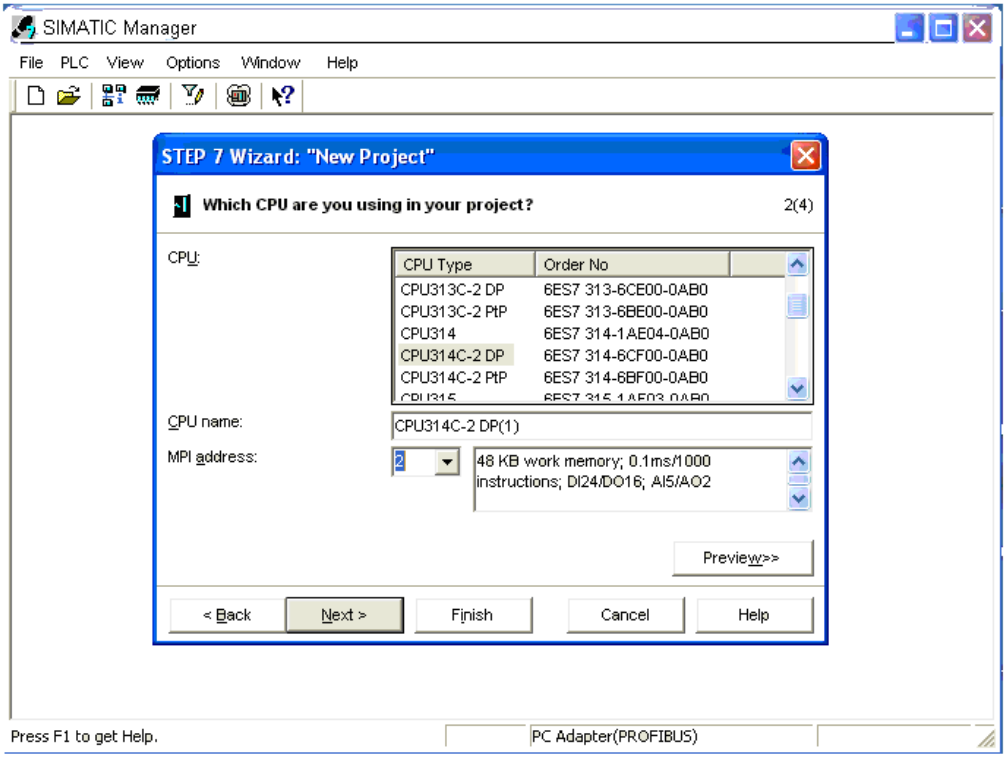
The following screen will appear.



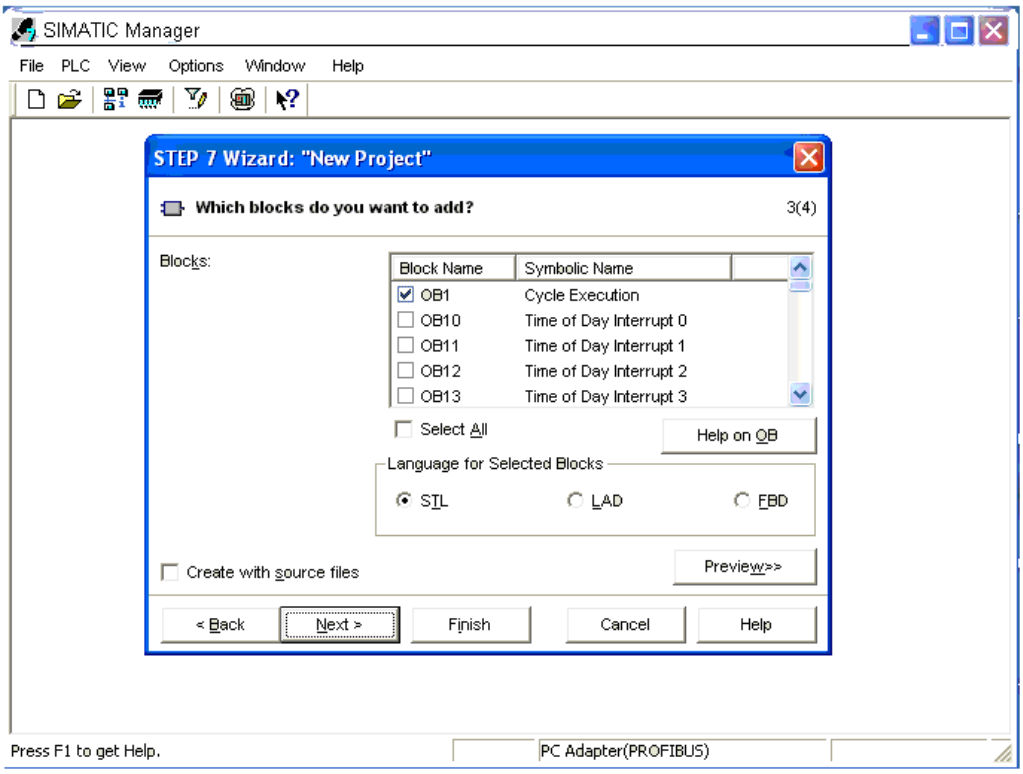
Click on **Next** button and in the following screen choose the **CPU 314C-2DP**



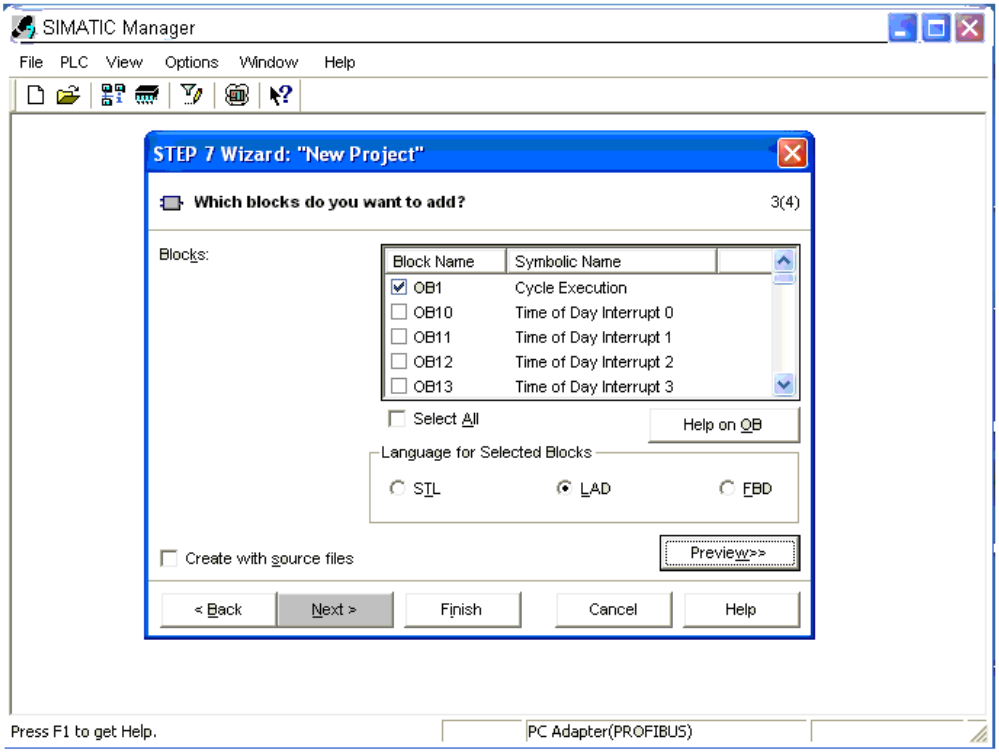
Leave MPI address as it is 2



Click on **next** again and Choose block OB1 from the list

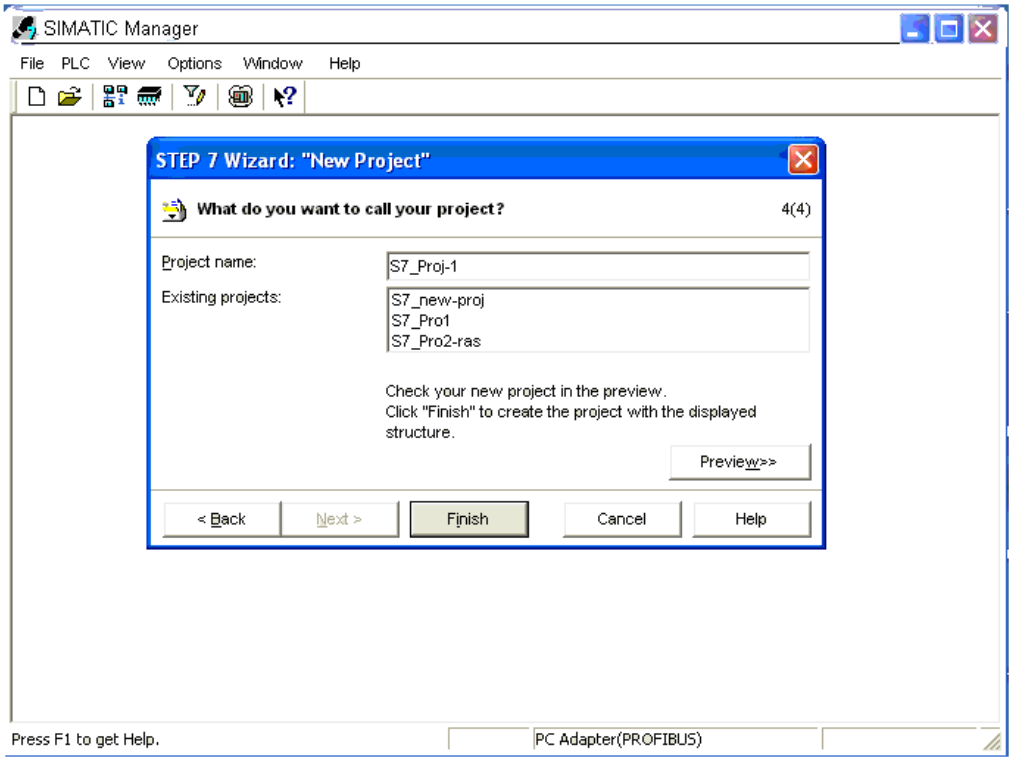


For the programming language select LAD for Ladder

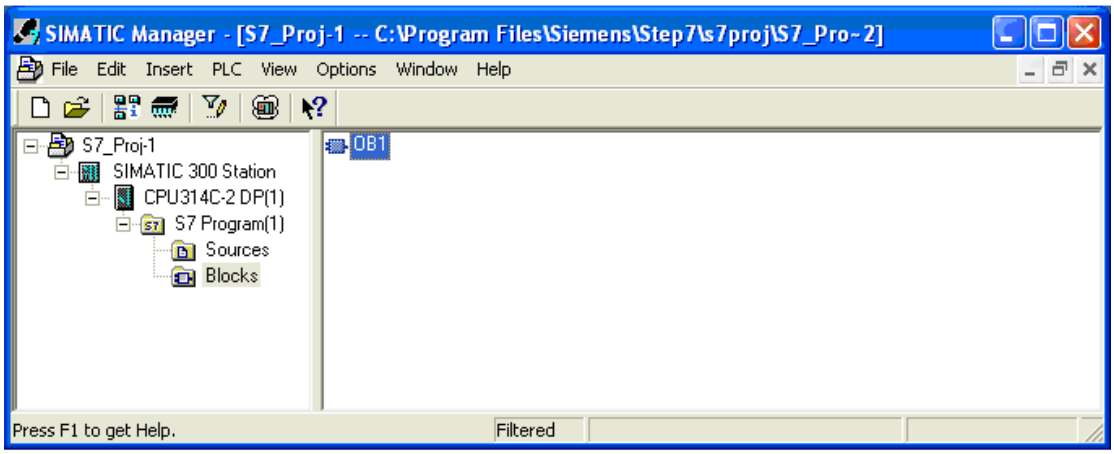


Click on **next**

Type the name of the new project → choose **S7_proj-1**

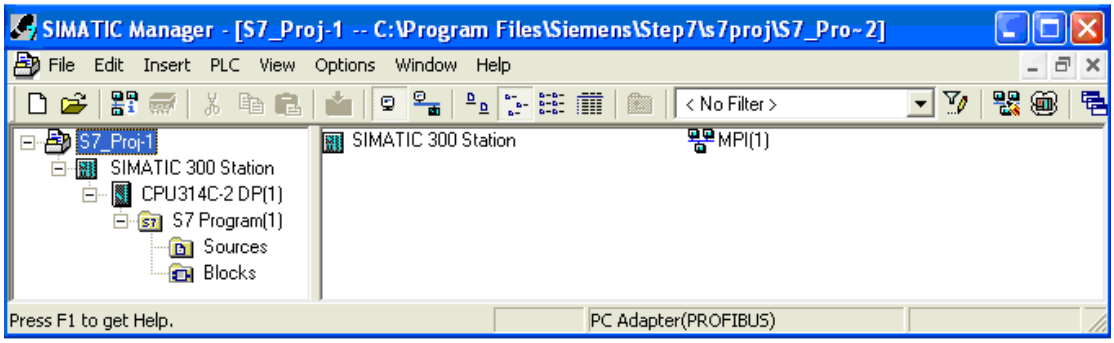


Click **Finish** button, Simatic Manager will create the project S7_proj-1.

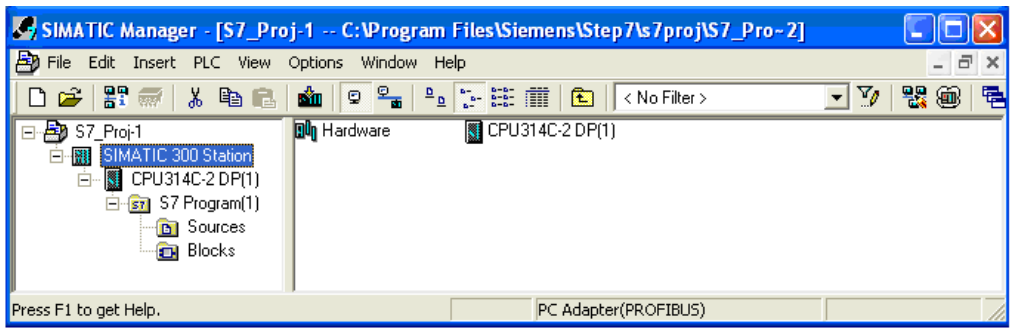


This part is aimed to introduce you to the new projects and related files, follow the procedure below to take a tour and learn more about the Simatic Manager.

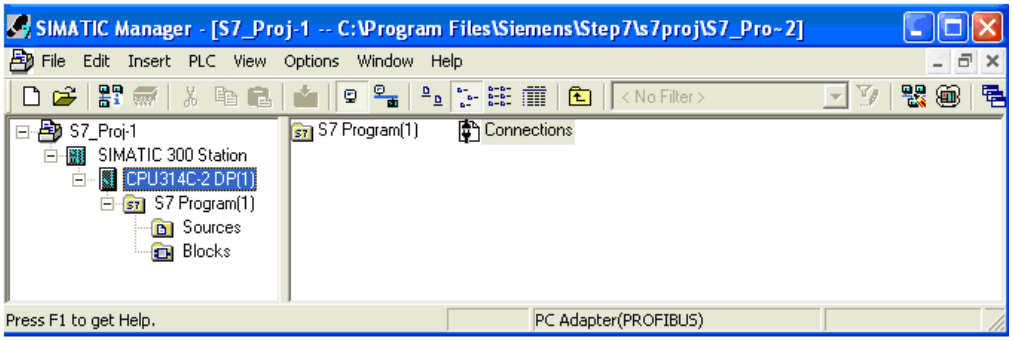
If you highlighted S7_proj-1 you will see 2 entities



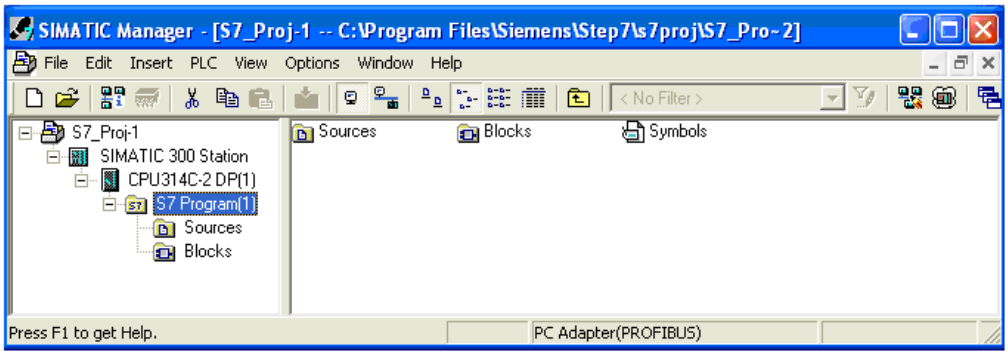
Click on Simatic 300 station



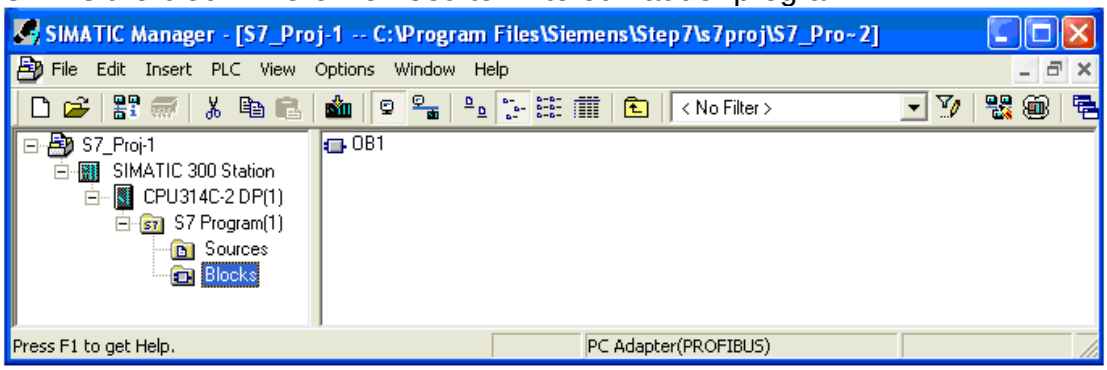
Show what is under CPU314C-2DP



Under S7 program → sources Blocks symbols

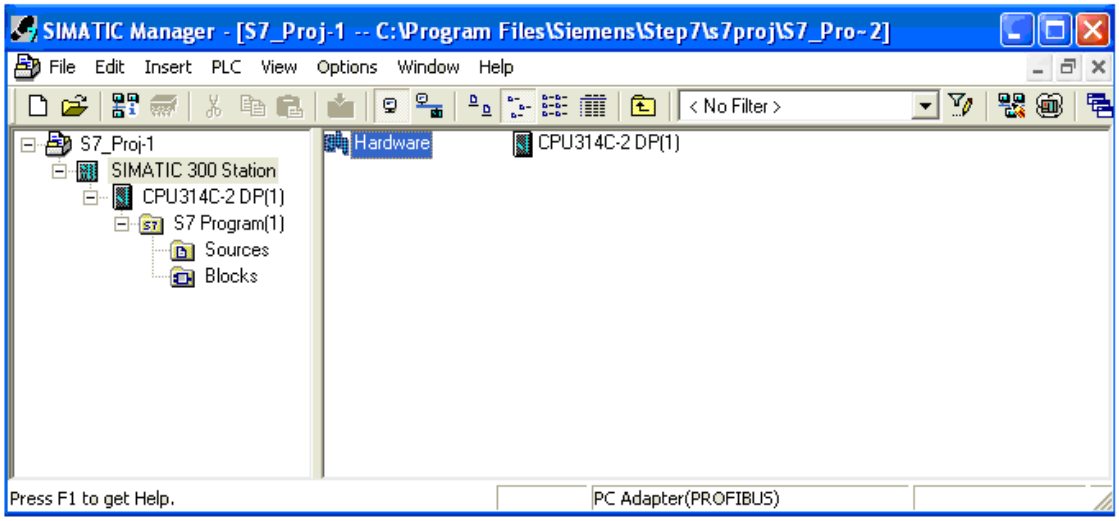


OB1 is the block where we need to write our ladder program

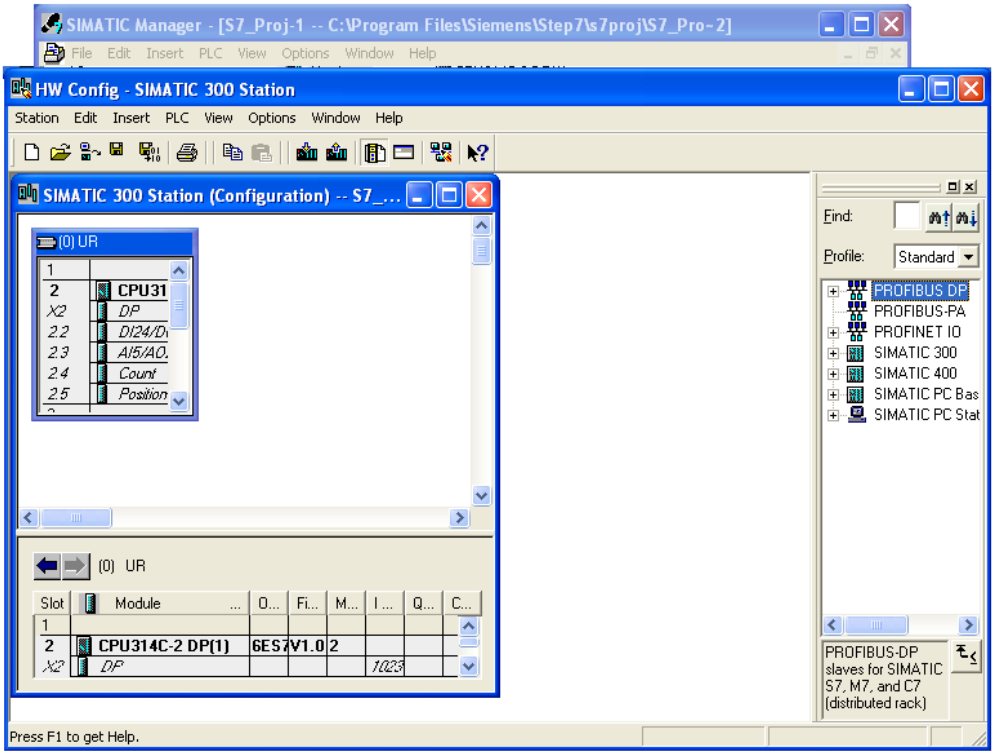


Hardware configuration

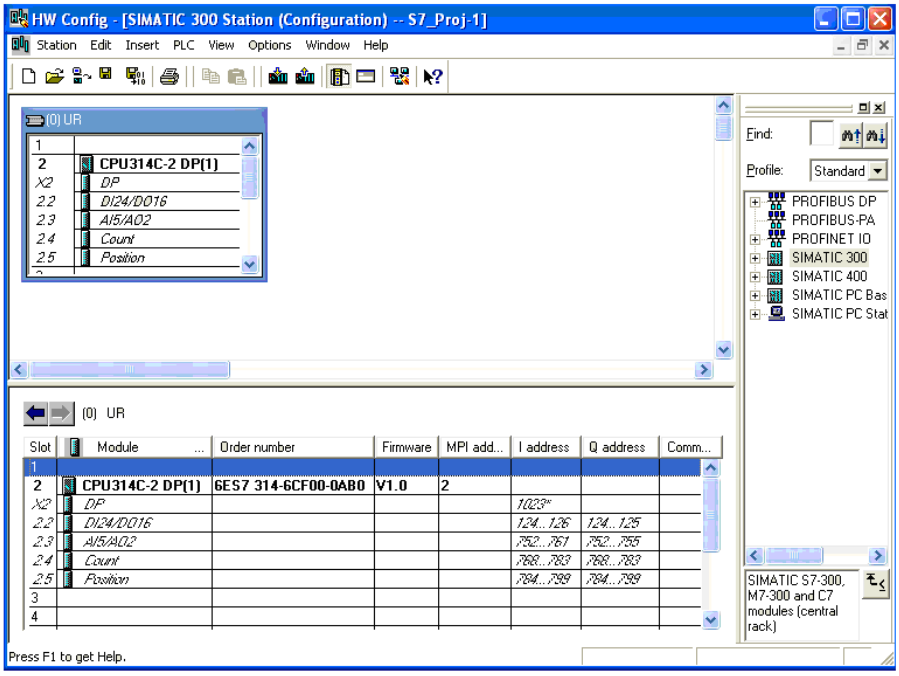
Go back to click on **Simatic 300 station** to see the hardware configuration. Remember that the system is already configured for you.



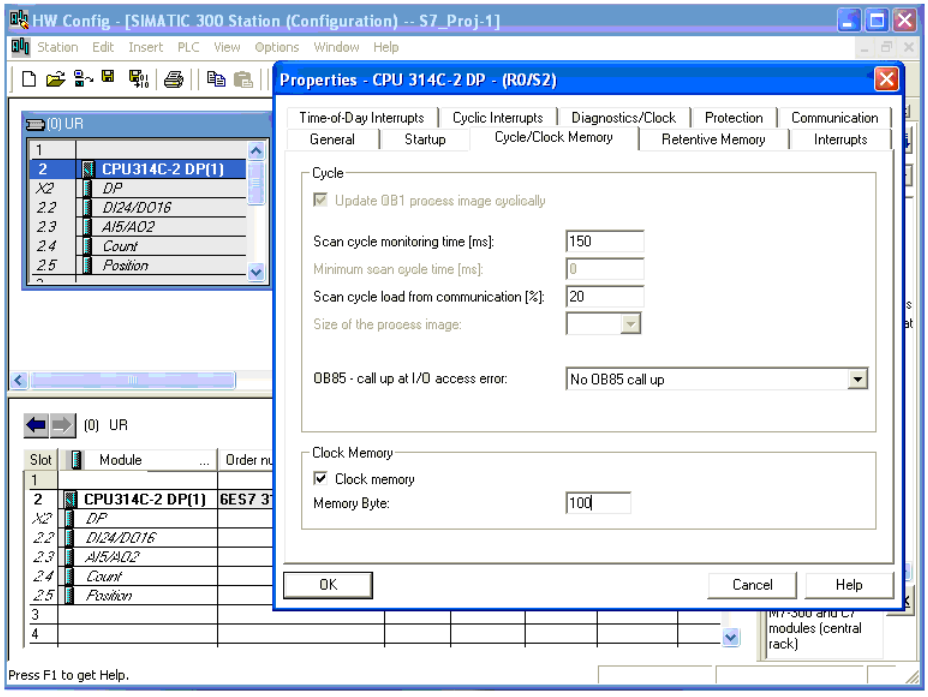
Double click on Hardware icon



Using the mouse Enlarge the screen inside to see more details
The following screen shows the hardware configuration, profile side menu shows more Hardware that can be added.



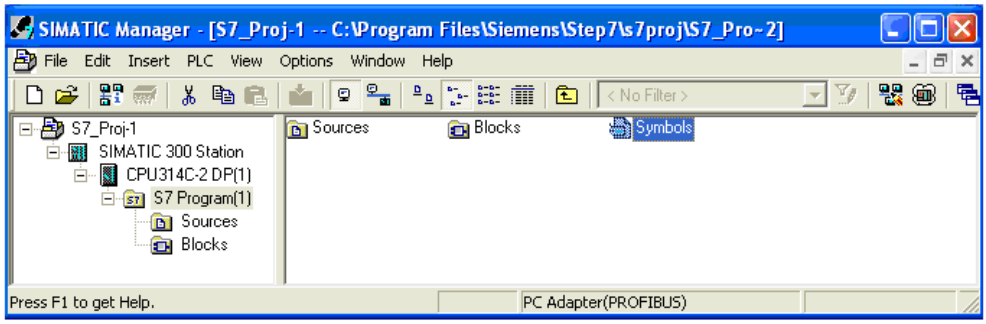
Double Click on CPU314C-2DP(1) icon and select Cycle/Clock Memory card



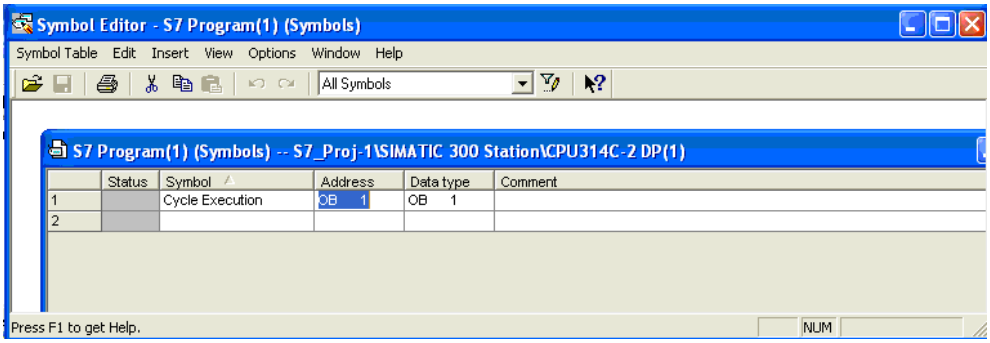
Creating a small Ladder diagram

Symbol table:

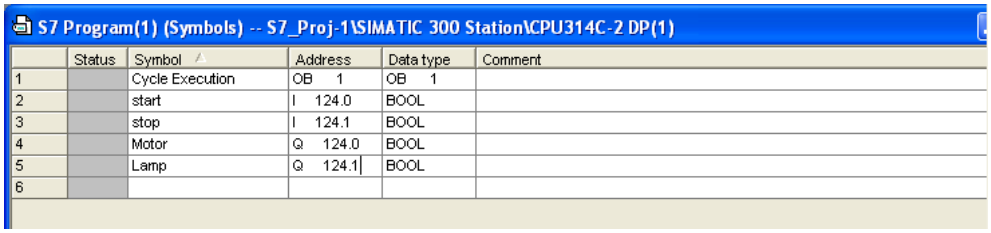
We begin with the symbol table to Click on S7 program icon first and then click on Symbols



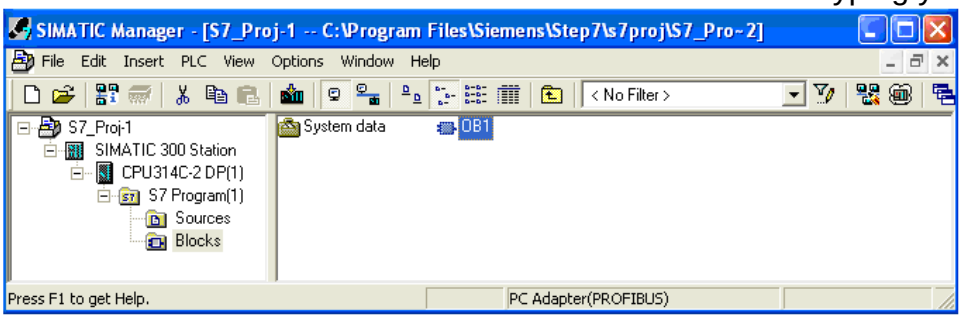
The following screen will appear



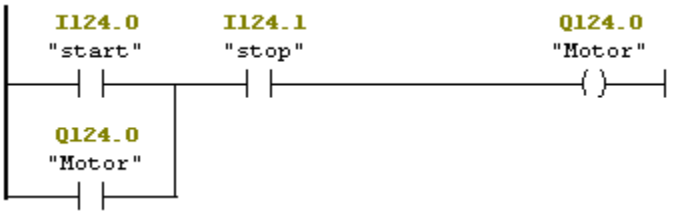
Start typing in the symbols and their addresses as below.



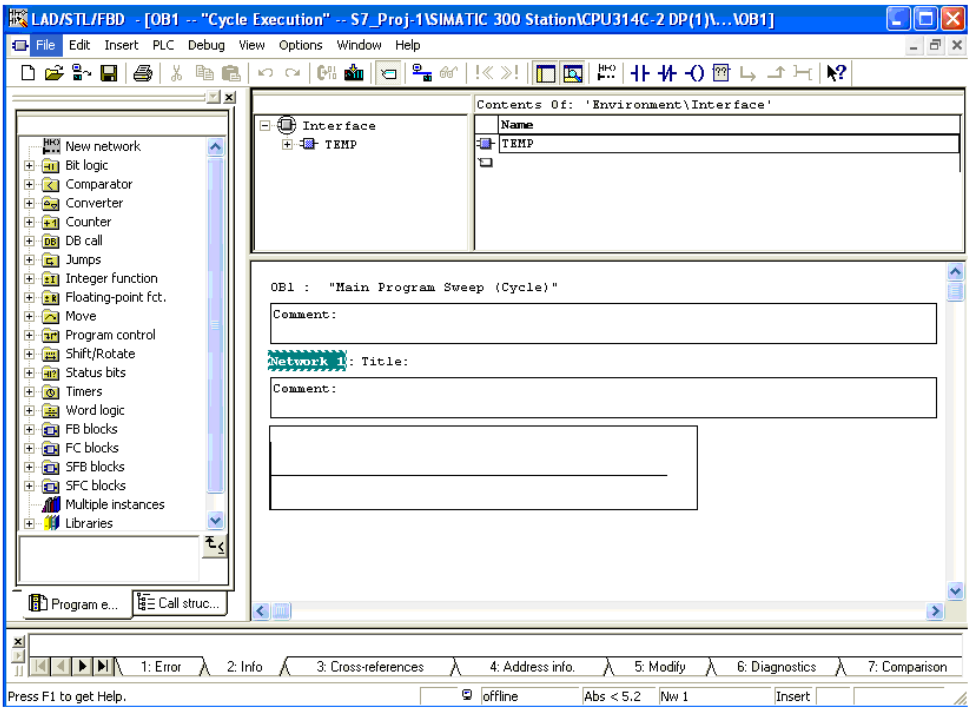
Now click on Blocks and then double click on OB1 to start typing your program.



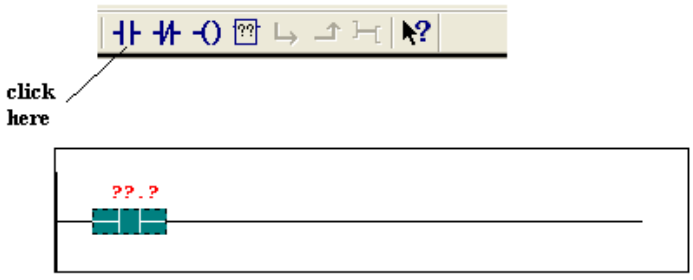
The following screen will appear, follow the procedure correctly to enter your first program in Ladder. The program is to control the start and stop of a motor.



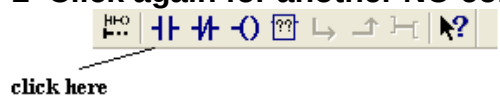
Click on the drawing area under Network-1



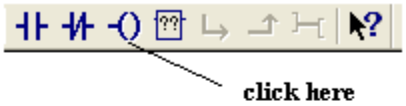
Step-1 When the mouse is in the drawing area click on open contact -] [-

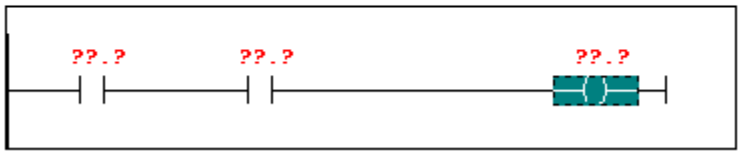


Step-2 Click again for another NO contact -] [-

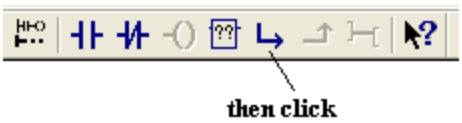


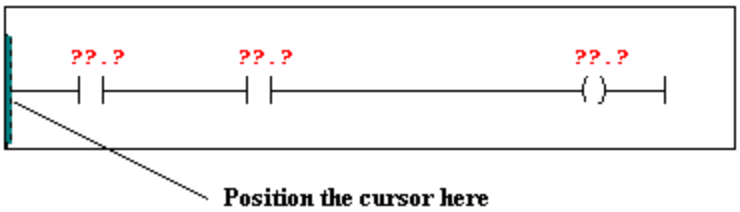
Step-3 Click for the Coil -( )-

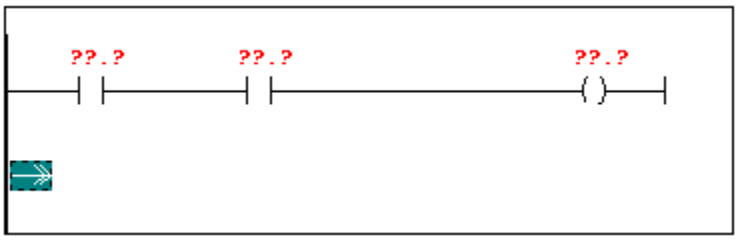




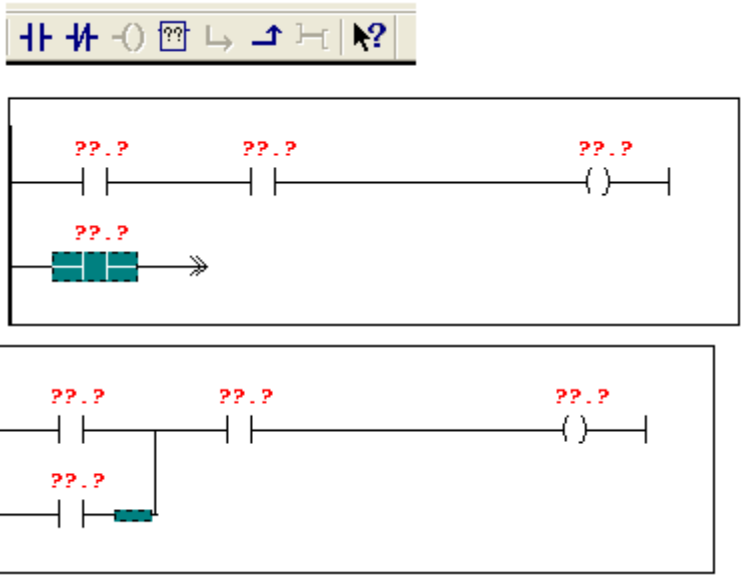
Step-4 Now for parallel contact you have to position the cursor in the beginning of
the rung and click on → as show below.





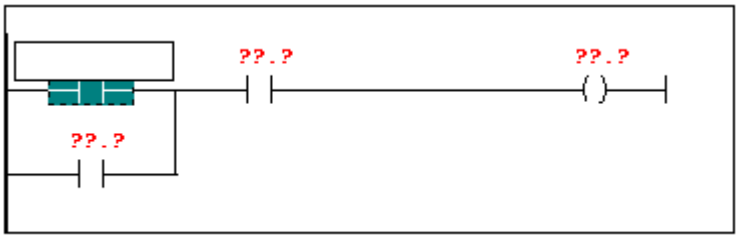


Step-5 click on the NO contact -] [- and then click on ^ to close the branch

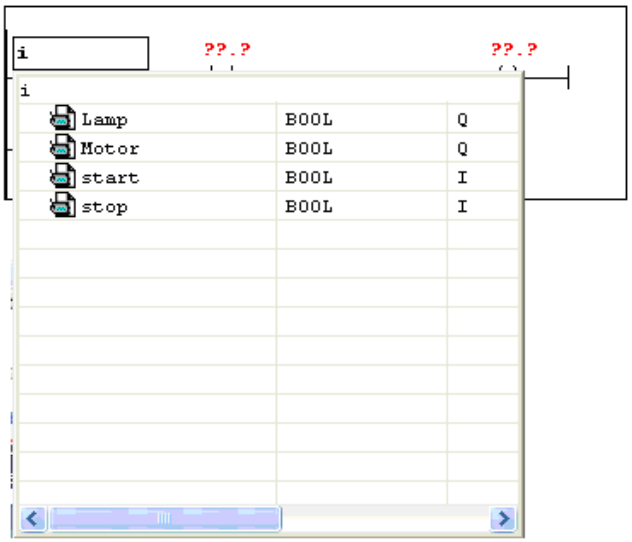


Now you can start editing the address on top of the ??.?

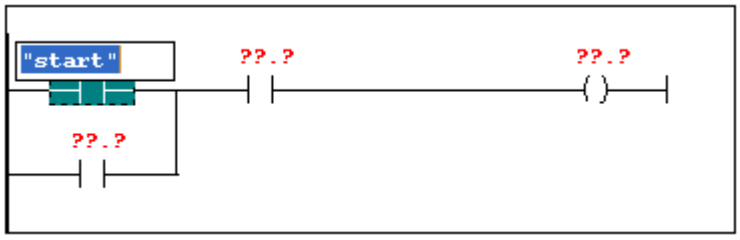
Click on the for contact for Start PB I124.0



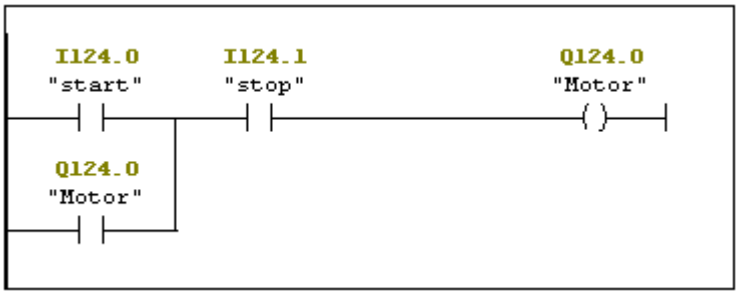
As soon as you press "I" the symbol table will appear



Double Click on start to select it



Do the same for the rest till you complete all.



The symbols will appear only if you have saved the OB1

Downloading and running the program

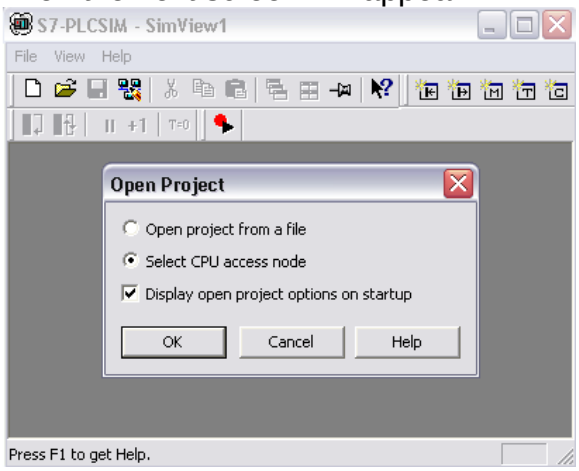
Note : Don't forget to put STOP switch in the on position (NC)

Now you need to download and run your program in the PLC. Also you can simulate the program using PLCSIM

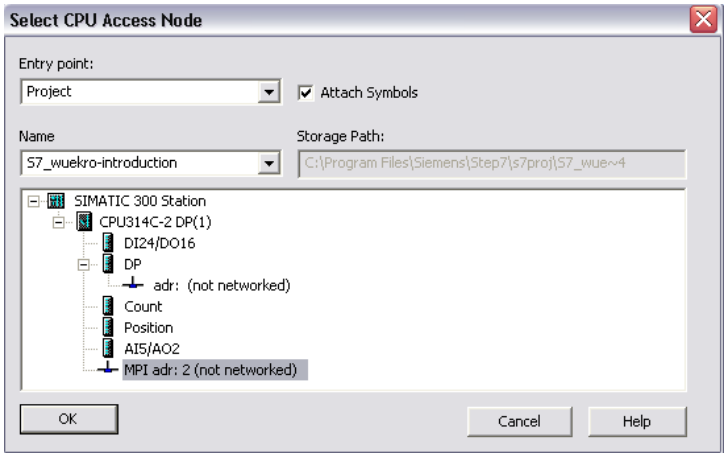
Skip the following steps (between the → marks) if you are not using the PLCSIM.

→ → → ***Skip here if not using PLCsim*** ← ← ←

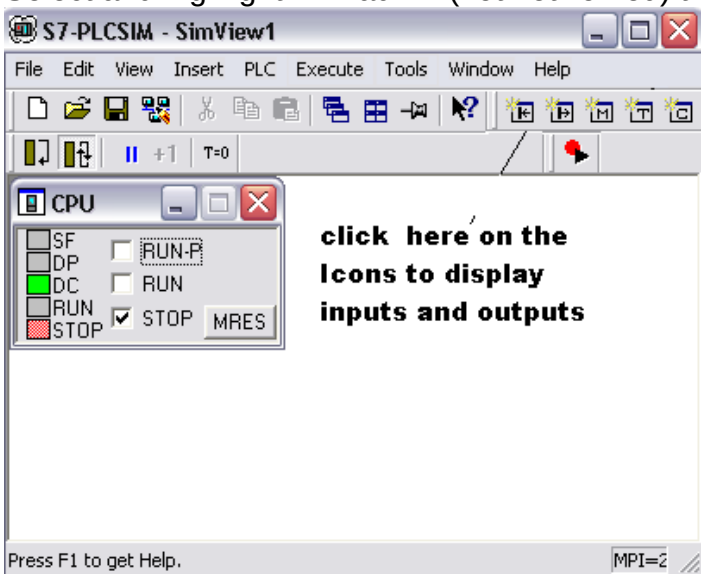
Then the next screen will appear

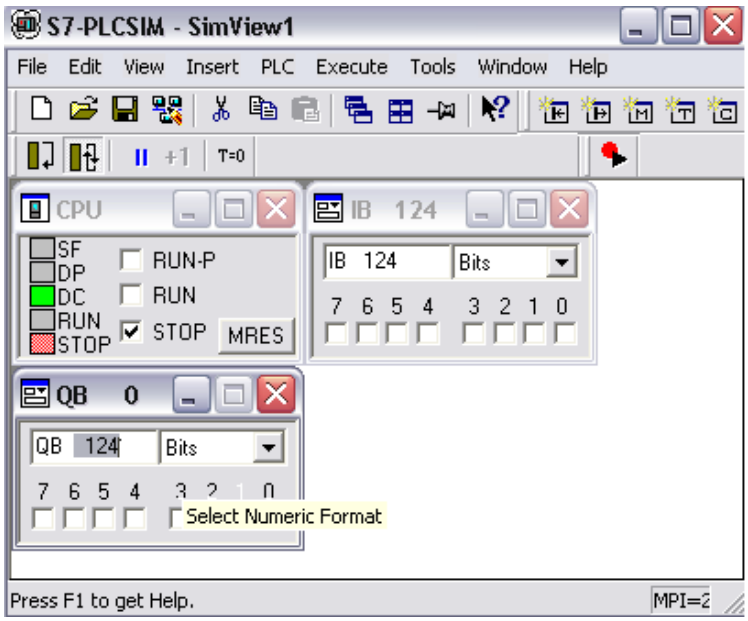


Click ok



Select and highlight MPI adr: 2 (not networked) then press ok

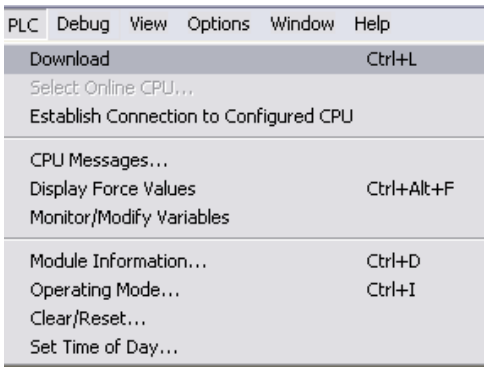




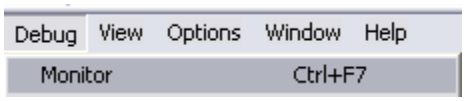
Now this the screen where you need to change the inputs and see the outputs after you download the program.

→ → → ***Continue below*** ← ← ←

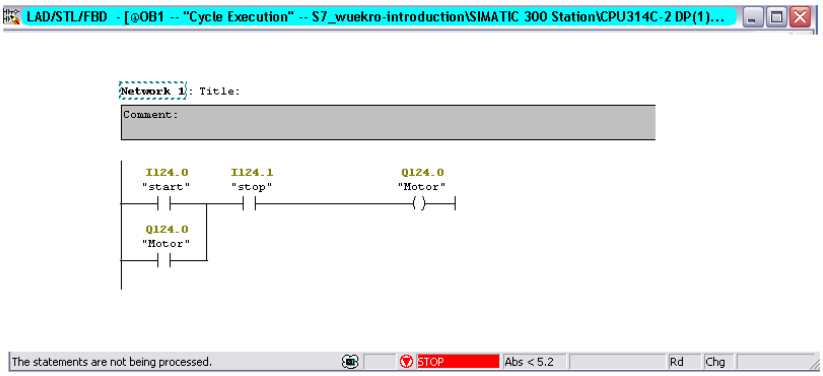
Now make sure that your PLC is in the **STOP** mode and then click on Download on the Program screen.



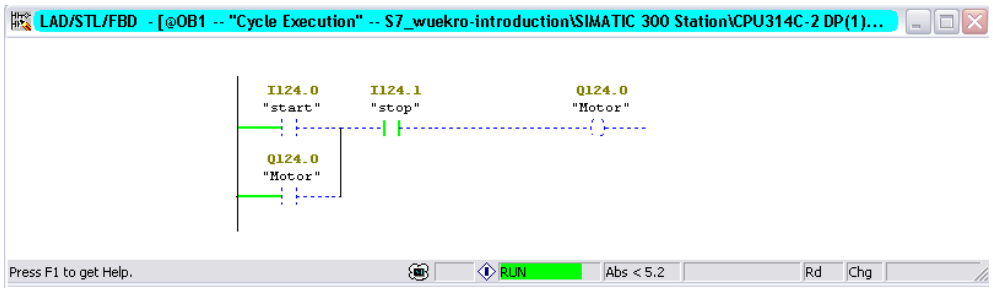
Then click on monitor under Debug menu.



You need to watch the important information on screen. Change in color of the top line to indicate the monitoring is active. Red color in the bottom line to show that the CPU is in STOP mode.

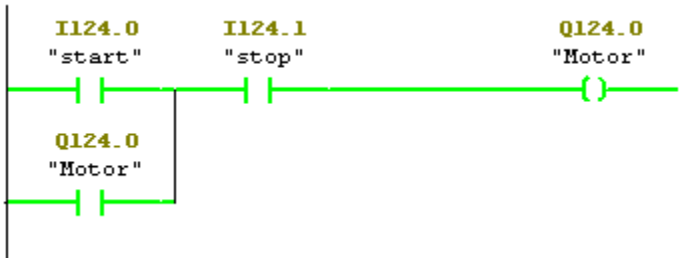


Switch the CPU to the RUN position.



The green solid line means that the contact is in True state "ON", and a dot line in the blue means that the contact is in False state "OFF"

Switch “start” on and then back to off, the output “Motor” will be on.



PART THREE

NUMBER SYSTEM AND CODES

TYPES OF SIGNALS IN CONTROL SYSTEM TECHNOLOGY

The electrical signals which are applied at the inputs and outputs can be, in principle, divided into two different groups:

BINARY SIGNAL

Binary signals can take the value of 2 possible states. They are as follows:

Signal state “1“ = voltage available = e.g. Switch on
Signal state “0“ = voltage not available = e.g. Switch off

In control engineering, a frequent DC voltage of 24V is used as a “control supply voltage“ A voltage level of + 24V at an input clamp means that the signal status is “1“ for this input. Accordingly 0V means that the signal status is “0“. In addition to a signal status, another logical assignment of the sensor is important. It’s a matter of whether the transmitter is a “normally closed” contact or a “normally open” contact. When it is operated, a “normally closed” contact supplies a signal status of “0“ in the “active case“. One calls this switching behavior “active 0“ or “active low“. A “normally open” contact is “active 1”/“active high“, and supplies a “1“ signal, when it is operated.
In closed loop control, sensor signals are “active 1“. A typical application for an “active 0“ transmitter is an emergency stop button. An emergency stop button is always on (current flows through it) in the non actuated state (emergency stop button not pressed). It supplies a signal of “1“(i.e. wire break safety device) to the attached input. If operation of an emergency stop button is to implement a certain reaction (e.g. all valves close), then it must be triggered with a signal status of “0“

Equivalent binary digits:

A binary signal can only take the two values (signal statuses) “0“ or “1“. Such a binary signal is also designated as an equivalent binary digit and receives the designation of “Bit“ in the technical language book. Several binary signals result in a digital signal after a certain assignment (code). While a binary signal only provides a grouping of a bivalent size/e.g. for door open/door close), one can form e.g. a number or digit as digital information by the bundling of equivalent binary digits.
The summarization of n-equivalent binary digits allows the representation of 2n different combinations.

One can show 4 different types of information with e.g. 2 equivalent binary digits 2x2:

0	0	Configuration 1	(e.g. Both switches open)
0	1	Configuration 2	(Switch 1 closed / Switch 2 open)
1	0	Configuration 3	(Switch 1 open / Switch 2 closed)
1	1	Configuration 4	(both switches closed)

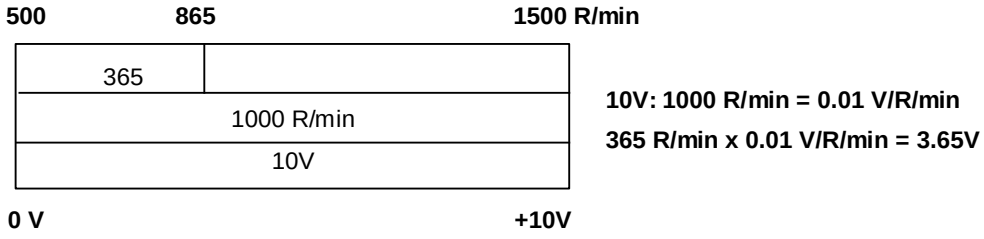
ANALOG SIGNAL

Contrary to a binary signal that can accept only signal statuses („Voltage available +24V“ and “Voltage available 0V“, there are similar signals that can take many values within a certain range when desired. A typical example of an analog encoder is a potentiometer. Depending upon the position of the rotary button, any resistance can be adjusted here up to a maximum value.

Examples of analog measurements in control system technology:

- Temperature -50 ... +150°C
- Current flow 0 ... 200l/min
- Number of revolutions 500 ... 1500 R/min
- Etc.

These measurements, with the help of a transducer in electrical voltages, are converted to currents or resistances. E.g. if a number of revolutions is collected, the speed range can be converted over a transducer from 500... 1500 R/min into a voltage range from 0... +10V. At a measured number of revolutions of 865 R/min, the transducer would give out a voltage level of + 3.65V.



If similar measurements are processed with a PLC, then the input must be converted into digital information to a voltage, current or resistance value. One calls this transformation analog to digital conversion (A/D conversion). This

means, that e.g. a voltage level of 3.65V is deposited as information into a set of equivalent binary digits. The more equivalent binary digits for the digital representation will be used, in order for the resolution to be finer. If one would have e.g. only 1 bit available for the voltage range 0... +10V, only one statement could be met, if the measured voltage is in the range 0.. +5V or +5V....+10V. With 2 bits, the range can be partitioned into 4 single areas, (0... 2.5/2.5... 5/5... 7.5/7.5... 10V).

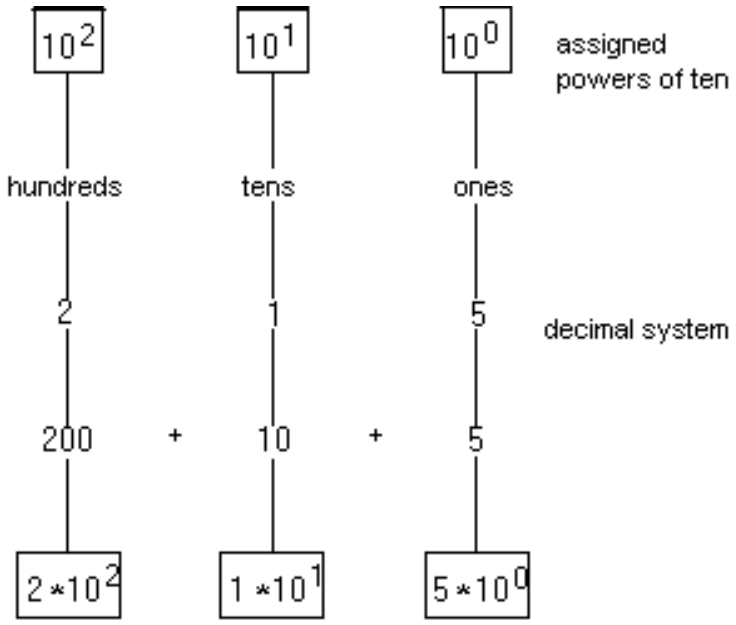
Usually in control engineering, the A/d converter is changed with the 8th or 11th bit. 256 single areas are normally provided, but with 8 or 11 bits, you can have 2048 single areas.

NUMBER SYSTEMS

For the processing of the addresses of memory cells, inputs, outputs, times, bit memories etc. by a programmable controller, the binary system is used instead of the decimal system.

DECIMAL SYSTEM

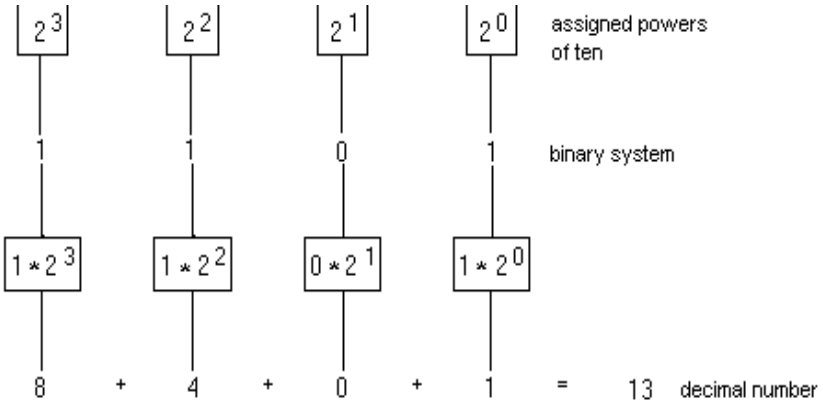
In order to understand the binary number system, it is first necessary to consider the decimal system. Here the number of 215 is to be subdivided. Thereby the hundreds represent the 2, the 1 stand for the tens and the 5 for the ones. Actually, one would have to write 215 in such a way: 200+10+5. If one writes down the expression 200+10+5, with the help of the powers of ten as explained earlier, then one states that each place is assigned a power of ten within the number.



Each number within the decimal system is assigned a power of ten.

BINARY SYSTEM

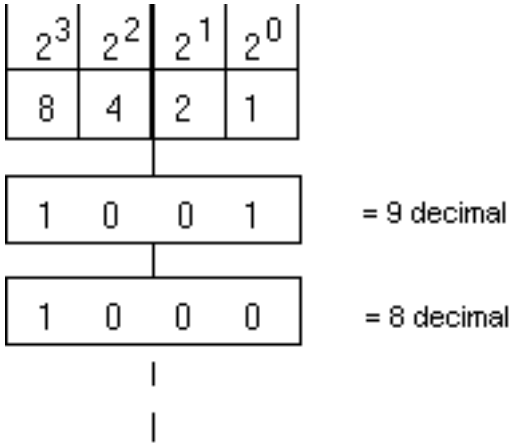
The binary number system uses only the numbers 0 and 1, which are easily represented and evaluated in data processing. Thus it is called a binary number system. The values of a dual number are assigned the power-of-two numbers, as represented below.



Each number assigned within the binary number system is a power-of-two.

BCD - CODE (8-4-2-1-CODE)

In order to represent large numerical values more clearly, the BCD code (binary coded decimal number) is frequently used. The decimal numbers are represented with the help of the binary number system. The decimal digit with the highest value is the 9. One needs to demonstrate the 9 with power-of-two numbers until 2^3, thus using 4 places for the representation of the number.



Because the representation of the largest decimal digit requires 4 binary places, a four-place unit called a tetrad, is used for each decimal digit. The BCD code is thus a 4-Bit-Code

Each decimal number is coded individually. The number of 285 consists e.g. of three decimal digits. Each decimal digit appears in the BCD code as a four-place unit (tetrad).

2	8	5
001 0	100 0	010 1

Each decimal digit is represented by an individually coded tetrad.

HEXADECIMAL NUMBER SYSTEM

The hexadecimal number system belongs to the notational systems because value powers of the number 16 are used. The hexadecimal number system is thus a sixteen count system. Each place within a hexadecimal number is assigned a sixteenth power. One needs altogether 16 numbers, including the zero. For the numbers 0 to 9 one uses the decimal system, and for the numbers 10 to 15 the letters A, B, C, D, E and F are used.

Each digit within a hexadecimal number system is assigned a power of the number 16.

DEMONSTRATION OF THE NUMBER SYSTEMS

decimal number	binary number					hexadecimal number
	16	8	4	2	1	
0					0	0
1					1	1
2				1	0	2
3				1	1	3
4			1	0	0	4
5			1	0	1	5
6			1	1	0	6
7			1	1	1	7
8		1	0	0	0	8
9		1	0	0	1	9
10		1	0	1	0	A
11		1	0	1	1	B
12		1	1	0	0	C
13		1	1	0	1	D
14		1	1	1	0	E
15		1	1	1	1	F
16	1	0	0	0	0	1 0
17	1	0	0	0	1	1 1
18	1	0	0	1	0	1 2
19	1	0	0	1	1	1 3

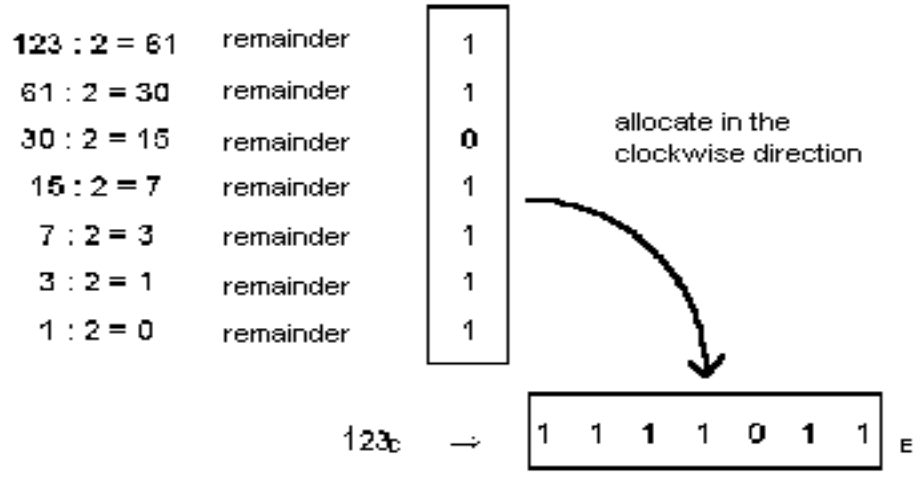
CONVERSION RULES

The transformation of the different number systems is based on simple rules. These rules should be controlled by the PLC users, since they are often used in handling this technology. For the use of a number system on which a given number is based, an index sign is placed at the end of a number. Here “D“ stands for decimal, “B“ for binary, and “H“ for hexadecimal. This marking is often necessary to identify a number system because in each system, different values can be obtained when the same number is used. (e.g.. “111“ in the decimal system has the value 111_D (one hundred eleven). In the binary system it would be 111_B ,which is the decimal value 7 (1x20 + 1x 21 + 1x22). As a hexadecimal number, 111_H would be the decimal value 273 (1x160 + 1x161 + 1x 162).

Converting decimal → binary

Integral decimal numbers are divided by the base 2 until the result of zero is obtained. The remainder obtained with the division (0 or 1) results in a binary number. One needs to also consider the direction that the “remainders“ are written in. The remainder of the first division is the first right bit (low order width unit bit).

e.g.: The decimal number 123 is to be changed into an appropriate dual number.



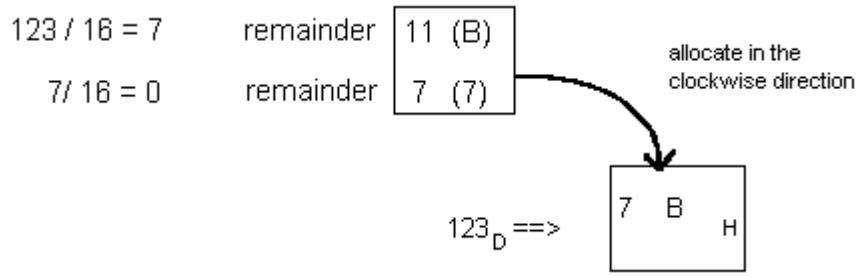
Pattern:

$$1\ \ 1\ \ 1\ \ 1\ \ 0\ \ 1\ \ 1$$
$$1x2^6 + 1x2^5 + 1x2^4 + 1x2^3 + 0x2^2 + 1x2^1 + 1x2^0$$
$$64 + 32 + 16 + 8 + 0 + 2 + 1 = \underline{123}$$

Converting decimal → hexadecimal

This transformation is performed exactly like the decimal → binary transformation. The only difference is that instead of using base 2, we use base 16. Thus the number must be divided by 16 rather than by 2.

E.g. The decimal number 123 is to be changed into the appropriate hex number.

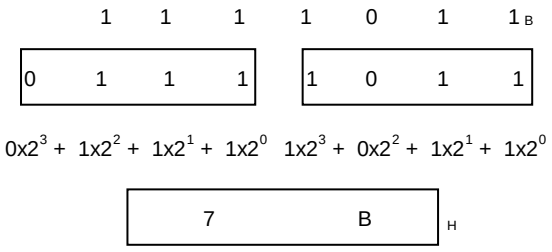


Pattern:

7		B		
$7x16^1$	+	$11x16^0$		
112	+	11	=	123

Converting binary → hexadecimal

For the transformation of a dual number into a Hex number, one could first determine the decimal value of the binary number (addition of the priorities). This decimal number could then be changed into a hexadecimal number with the help of the division:16. In addition, there is the possibility of determining the associated hex value directly from the binary number. First of all, the binary number is divided from the right beginning in the quadripartite groups. Every one of the determined quadripartite groups results in a number of the hexadecimal number system. If necessary, fill the missing bits on the left hand side with zeros e.g. The binary number 1111011 is to be changed directly into a hex number.

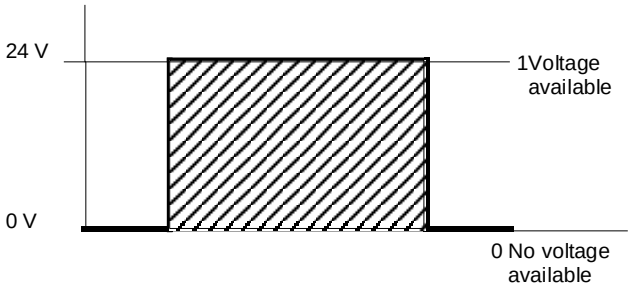


TERMS FROM COMPUTER SCIENCE

In connection with programmable controllers, terms such as BIT, BYTE and WORD are frequently used in the explanation of data and/or data processing.

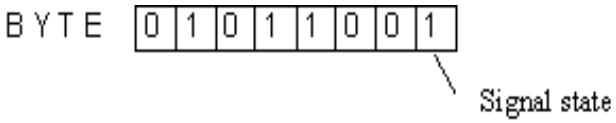
BIT

Bit is the abbreviation for binary digit. The BIT is the smallest binary (bivalent) information unit, which can accept a signal status of “1“or “0“.



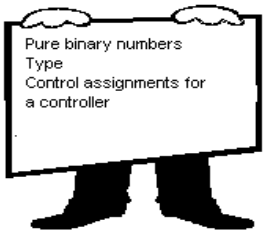
BYTE

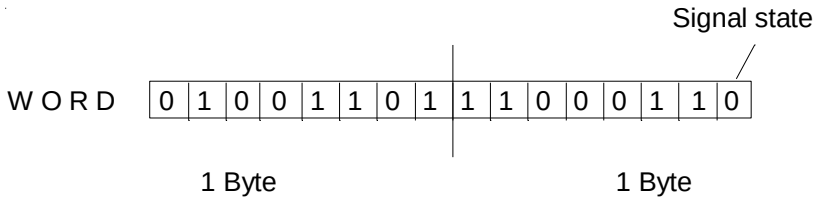
For a unit of 8 binary characters, the term BYTE is used. A byte has the size of 8 bits.



WORD

A word is a sequence of binary characters, which is regarded as a unit in a specific connection. The word length corresponds to the number from 16 binary characters. With words, the following can be represented:





A word also has the size of 2 bytes or 16 bits.

DOUBLE-WORD

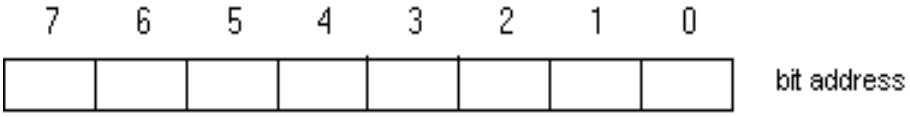
A double-word corresponds to the word length of 32 binary characters.

A double-word also has the size of 2 words, 4 bytes, or 32 bits.

Further units are kilo-bit or kilo-byte, which stand for $2^{10,}$ or 1024 bits, and the mega-bit or mega-byte which stands for 1024 kilo-bits.

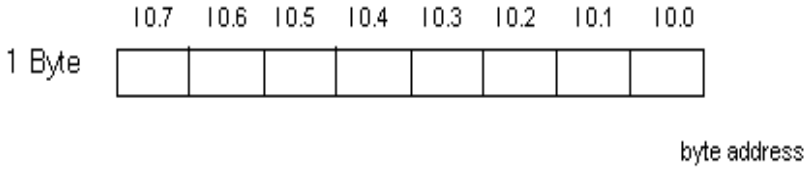
BIT ADDRESS

So that individual bits can be addressed within a byte, each individual bit is assigned a bit location. In each byte the bit gets the bit location 7 on the leftmost side and the bit location 0 on the rightmost side.



BYTE ADDRESS

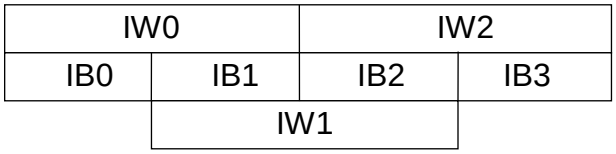
The individual bytes also receive numbers called byte displacements. Additionally, the operand is still marked, so that e.g. IB 2 stands for input byte 2 and QB4 stands for output byte 4. Individual bits are clearly addressed by the combination of bit and byte displacement. The bit location is separated from the byte displacement by one point. The bit location stands to the right of the point, and the byte displacement to the left.



WORD ADDRESS

The numbering of words results in a word address.

Note: The word address is always the smallest address of the two pertinent bytes when using words, e.g. input word(IW),output word(QW), bit memory word(MW), etc. (e.g. With a word that comes from IB2 and IB3, the address is IW2).



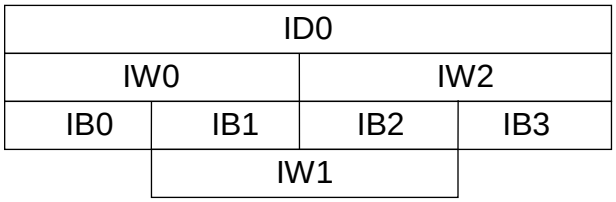
Word address

Note: During word processing it is to be noted that e.g. the input word 0 and the input word 1 are in a byte overlap. In addition, when counting bits, one begins at the rightmost bit. For example the bit0 from IW1 is the bit of I2.0, bit1 is I2.1.... bit7 is I 2.7, bit8 is I1.0.... bit15 ois I1.7. A jump exists between the bits 7 and 8.

DOUBLE-WORD ADDRESS

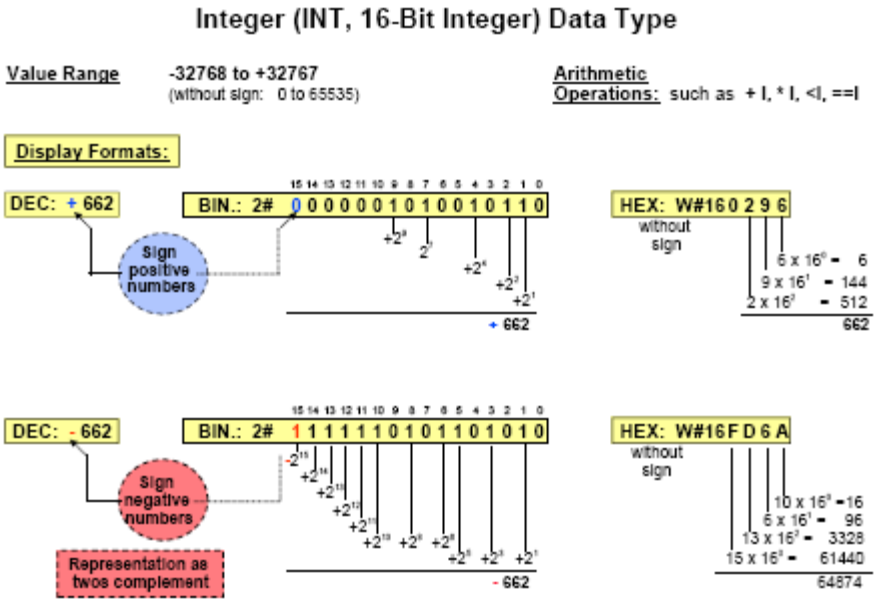
The numbering of double-words results in a double-word address.

Note: When using double-words e.g. ID, QD, MD etc. the double-word address is the smaller word address of the two pertinent words.



Double word address

Data Types



Integer Data Type

An *Integer* data type value is a whole number value, that is, a value without a (16-Bit Integer) decimal point. SIMATIC® S7 stores *Integer* data type values with sign in 16 bit code. This results in the value range shown in the slide above. As well, SIMATIC® S7 provides arithmetic operations for processing Integer values.

Decimal

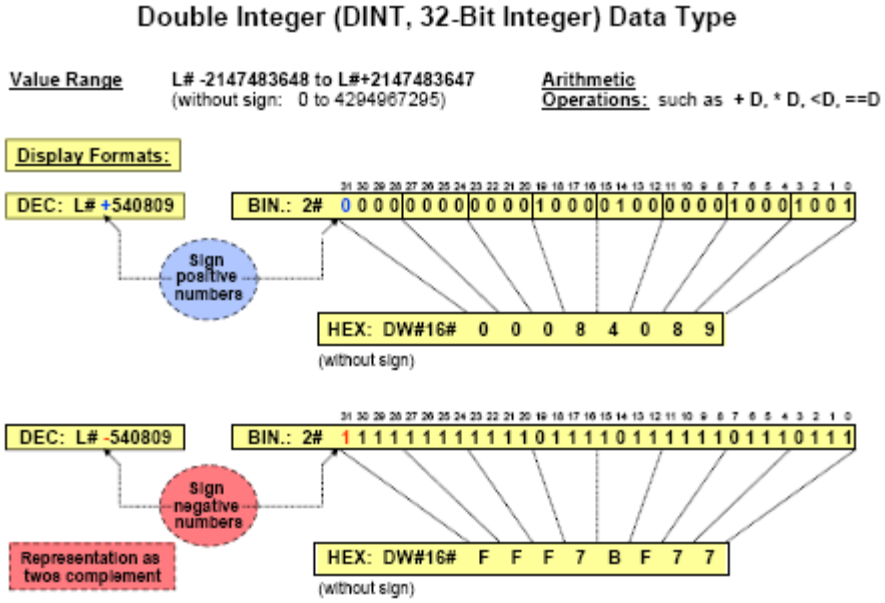
STEP7 uses the *Decimal* (not BCD!) display format to specify the constants of the *Integer* data type. That is, with sign and without explicit format description. The use of constant Integer values in the *Binary* and *Hexadecimal* display formats is possible in principle, but because of the poor legibility, they are more or less not suitable. For this reason, the syntax of STEP7 provides the specification of Integer values only in the decimal display format.

Binary

In a digital computer system, all values are stored in a binary-coded form. Only the digits 0 and 1 are available in the binary number system. Base 2 of this numbers system results from the number of available digits. Accordingly, the value of every position of a binary number results from a power of Base 2. This is also expressed in the format specification **2#**.... .

Negative values are represented as binary numbers in twos complement. In this representation, the most significant bit (bit no. 15 for the Integer data type) has the value - 215. Since this value is greater than the sum of all residual values, this bit also has the sign information. That is, if this bit = 0, then the value is positive; if the bit is = 1, then the value is negative. The conversion of a binary number into a decimal number is made by adding the values of the positions that have a 1 (see slide).

Specifying constants in the binary display format is not only used for specifying Integer values, but more often to specify bit patterns (such as in digital logic operations) in which the Integer value represented by the bit pattern is of no interest. The number of specifiable bits is variable from 1 to 32. Missing bits are filled with leading zero digits.



Double Integer

SIMATIC® S7 stores *Double Integer* data type values with sign as 32 bit code. (32-Bit Integer) This results in the value range shown in the slide above. As well, SIMATIC® S7
provides arithmetic operations for processing DINT values.

Decimal

STEP7 uses a decimal number (not BCD!) to specify a constant of the *Double Integer* data type. That is, with sign and the format **L#** for "long" (double word, 32 bit). When a value smaller than -32768 or greater than 32767 is specified, the format **L#** is automatically added. For negative numbers smaller than -32768, the user must specify the format as **L**# - (for example: L# -32769). This is imperative if the value is to be further processed arithmetically as a double integer. Otherwise you would work with false values (value + sign!)!

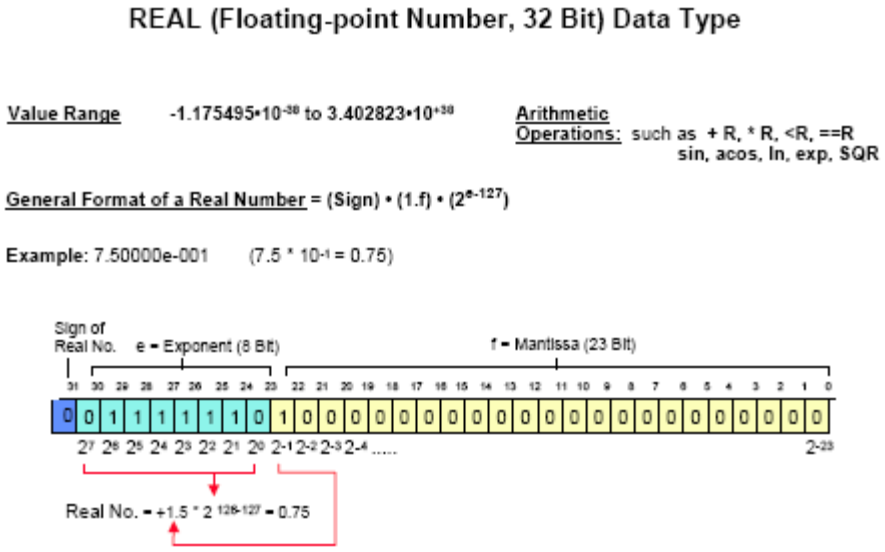
Hexadecimal

The hexadecimal numbers system provides 16 different digits (0 to 9 and A to F). This results in Base 16 of this numbers system. Accordingly, the value of every position of a hexadecimal number results from a power of Base 16. Hexadecimal numbers are specified with the format **W#** for the dimension (W = word = 16 bit) or **DW#** (DW = double word = 32 bit) and **16#** for identifying the basic numbering system. The number of specifiable bits is variable from 1 to 8. Missing bits are filled with leading zero digits.

The digits A to F correspond to the decimal values 10 to 15. The value 15 is the last value that can be binary-coded - without sign - with 4 bits.

This connection results in the simple conversion of a binary number in a hexadecimal number and vice versa. Four binary bits make up one digit of a hexadecimal number.

Constants in the hexadecimal format are therefore not used for specifying integer values. They are used instead of binary numbers for specifying bit patterns in which the integer value represented by the bit pattern is of no interest.



Real

The previously described INT and DINT data types are used to store whole number values with sign. Accordingly, only operations that supply a whole number value as the result can be performed with these data types.
In cases where analog process variables such as voltage, current, and temperature have to be processed, it becomes necessary to use *Real* values (real numbers, "decimal numbers"). In order to be able to represent such values, binary digits have to be defined whose value is less than 1 (power of base 2 with negative exponent).

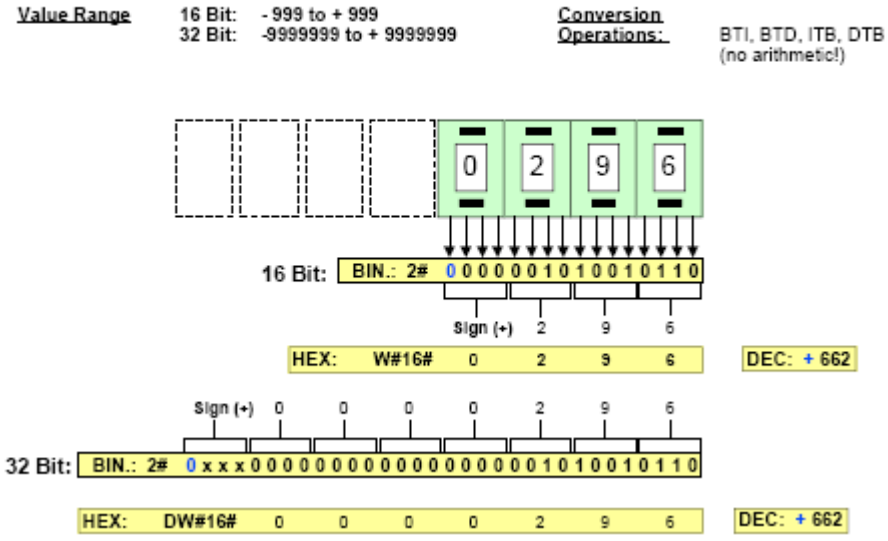
Real Format

In order to be able to form the greatest possible value range within a defined memory capacity (for SIMATIC® S7: double word, 32 bit) (see slide), you must be able to select the decimal point position. Early on, IEEE defined a format for floating-point numbers. This format was laid down in IEC 61131 and was included in STEP 7. This format makes it easy to process a variable decimal point position.

In a binary coded floating-point number, a portion of the binary digits contain the mantissa and the rest contain the exponent and the sign of the floating-point number.
When you specify real values, you do so without specifying the format. After you enter a constant real value (for example: 0.75), the Editor automatically makes a conversion (for example: 7.5000e-001).

Application
Floating-point numbers are used for "analog value processing", among others. A great advantage of floating-point numbers is in the number of operations possible with such numbers. These include, in addition to the standard operations such as: +, -, * , / also instructions such as sin, cos, exp, ln, etc, that are used mainly in closed-loop control algorithms.

The BCD Code for Inputting and Outputting Integers



Origin
In the past, the specification and visualization of whole numbers was done exclusively using simple, mechanical thumbwheel buttons and digital displays. These thumbwheel buttons and digital displays were connected to the PLC‘s digital input and output modules through parallel wiring. The structure could also be cascaded, without having to change the mechanical coding of a digit.

BCD Code
Each digit of a decimal number is encoded in four bit positions. Four bits are used because the highest decimal digit, 9, requires at least four bit positions in binary code. Decimal No. BCD Code Decimal No. BCD Code
0 0000 6 0110
1 0001 7 0111
2 0010 8 1000
3 0011 9 1001
4 0100 10 ... 15 not allowed
5 0101

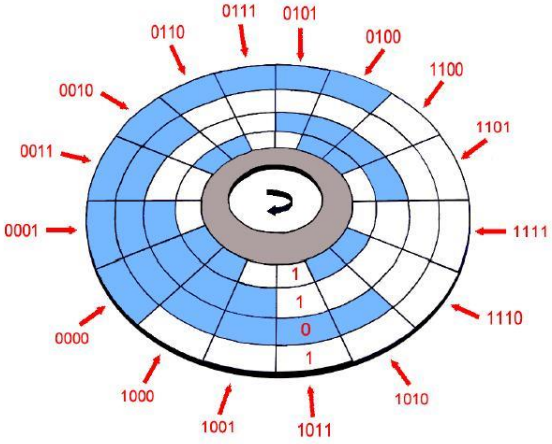
Negative Numbers
So that negative numbers can also be specified using a BCD thumbwheel button, STEP 7 codes the sign in the most significant bit of the most significant digit (see slide). A sign bit = 0 indicates a positive number. A sign bit = 1 indicates a negative number. STEP 7 recognizes 16-bit-coded (sign + 3 digits) and 32-bit-coded (sign + 7 digits) BCD numbers.

Data Formats
There is no data format for specifying BCD-coded values in STEP 7. You can, however, specify the decimal number whose BCD code is to be given, as a HEX number. The binary code of the HEX number and that of the BCD-coded decimal number is identical.
As you can see in the slide, the DEC data format is not suitable for specifying BCD coded numbers!

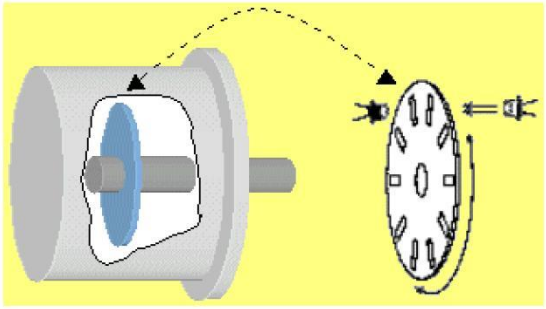
Gray Code:

- The Gray code is a special type of binary code that does *not* use position weighting.
- It is set up so that as we progress from one number to the next, *only one bit changes.*
- For this reason, the Gray code is considered to be an error-minimizing code.
- Because only one bit changes at a time, the speed of transition for the Gray code is considerably *faster than* that for codes such as BCD
- Gray codes are used with position encoders for accurate control of the motion of robots, machine tools, and servomechanisms.

Typical Encoder Disk:



The encoder disk is attached to a rotating shaft and outputs a digital Gray Code signal that is used to determine the position of the shaft.



ASCII Code:

- ASCII stands for American Standard Code for Information Interchange.
- It is an alphanumeric code because it indicates letters as well as numbers.

The keystrokes on the keyboard of a computer are converted directly into ASCII for processing by the computer.

Parity Bit:

- Some PLC communications systems use a *parity* bit to check the accuracy of data transmission.
- For example, when data are transferred between PLC's, one of the binary bits may accidentally change states
- Parity is a system where each character transmitted contains *one additional bit* known as a parity bit.
- The bit may be binary 0 or binary 1, depending on the number of 1 's and O's in the character itself
- Two systems of parity are normally used: *odd* and *even*
- Odd parity means that the total number of binary 1 bits in the character, including the parity bit, is odd
- Even parity means that the total number of binary 1 bits in the character, including the parity bit, is even

Character	Even Parity Bit	Odd Parity Bit
0000	0	1
0001	1	0
0010	1	0
0011	0	1
0100	1	0
0101	0	1
0110	0	1
0111	1	0
1000	1	0
1001	0	1

PART FOUR

BASICS OF PLC PROGRAMMING

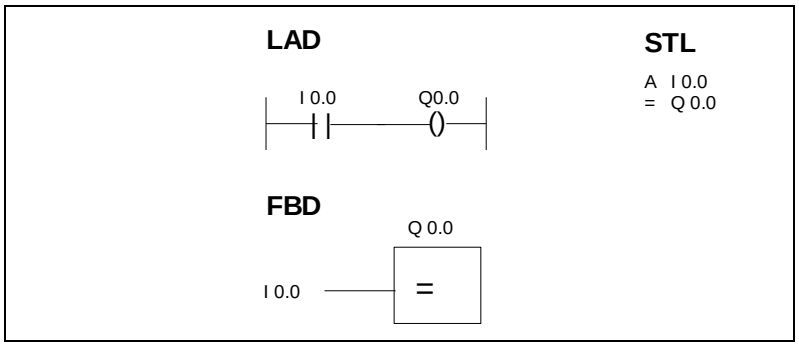
BASICS OF PLC PROGRAMMING

BASIC PROGRAMMING INSTRUCTIONS

The following programming instructions are sufficient for the basics of programming. This is however not a complete listing of all instructions. Information for further instructions in LAD/FBD/STL can be found in the manuals or in the **on-line help** under the point of **language description LAD, FBD** and/or **STL**.

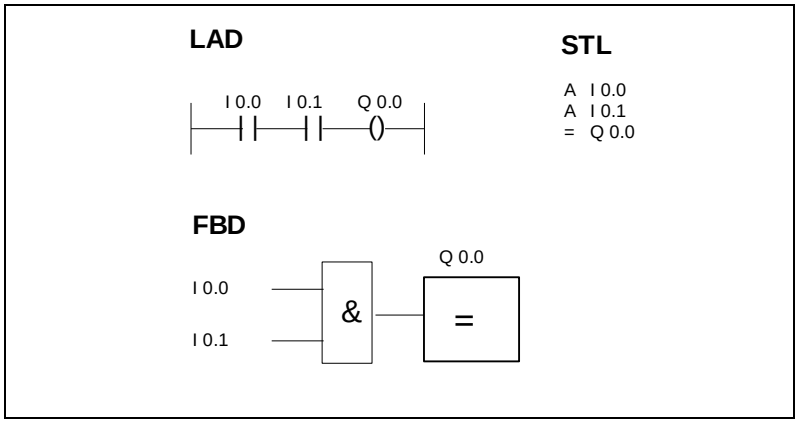
ASSIGNMENT

The assignment (=) copies the logical operation result (RLO) of the preceding operation and assigns it to the following operand. An operation chain can be locked by an assignment.



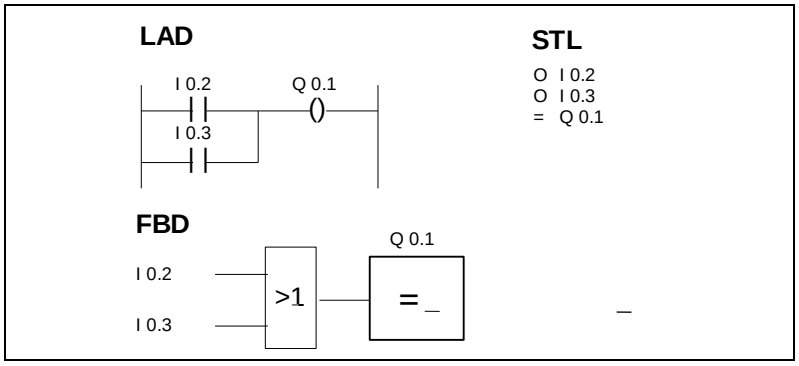
AND - OPERATION

The AND -Operation corresponds to a series connection of contacts in the circuit diagram. At the output Q 0.0, the signal status 1 appears if all inputs exhibit a signal status 1 at the same time. If one of the inputs exhibits a signal status 0, the output remains in a signal status 0.



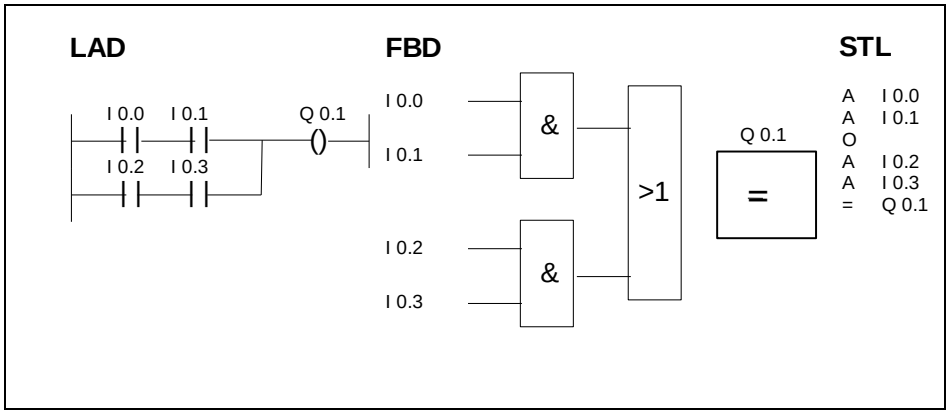
OR - OPERATION

The OR -Operation corresponds to a parallel connection of contacts in the circuit diagram. At the output Q 0.1, a signal status 1 appears if at least one of the inputs exhibits a signal status 1. Only if all inputs exhibit a signal status 0, will the signal status at the output remain on 0.



AND - BEFORE OR - OPERATION

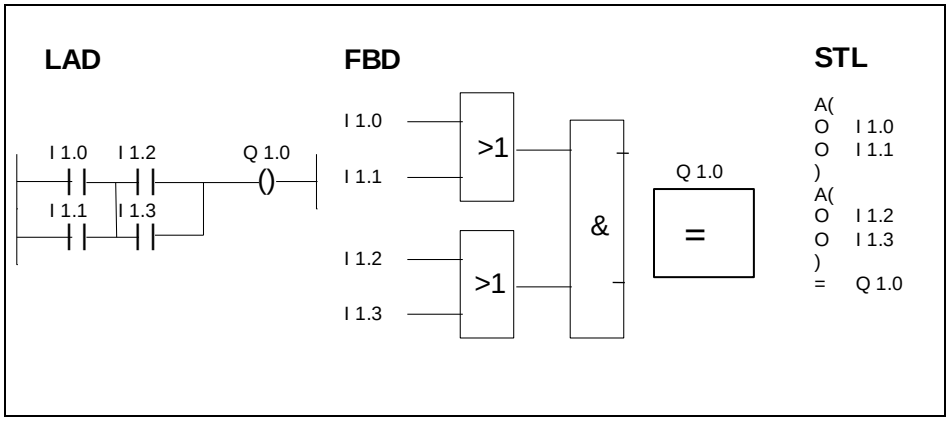
The AND- before -OR -Operation corresponds to a parallel set-up of several contacts in the circuit diagram.
With these branches from rows and parallel circuits aligned together, the output 0.1 is fed the signal status 1, if in at least one branch of all contacts switched in the row are closed (have a signal status 1).
The AND before OR- Operations are programmed without parentheses in the STL representation, however the parallel circuit branches must be separated by the input of the character O (OR function). First the AND functions are edited and from their results the result of the OR function is formed. The first AND function (I 0,0, I 0,1) becomes separated by the second AND function (I 0,2, I 0,3) through the single O (OR function).



The AND- Operations have priority and will always execute before the OR- Operations.

OR - BEFORE AND - OPERATION

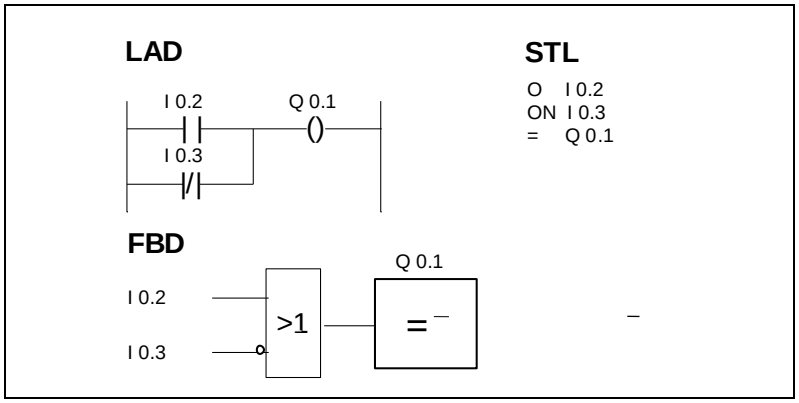
The OR – before -AND operation corresponds to a series connection of several contacts joined in parallel in the circuit diagram.
With these branches from the rows and parallel circuits aligned together, the output 1.0 is fed the signal status 1, if in both branches at least one of the contacts switched in the row is closed (have a signal status 1).



Parenthesis must be used on the OR- Operations so that they will have a higher priority than the AND- Operations.

QUERY ON SIGNAL STATE 0

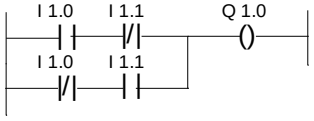
The debugging for the signal status 0 corresponds in a contact-afflicted circuit to an open contact and is realized in the connection AND NOT (AN), OR NOT (ON) and EXCLUSIVE OR NOT (XN).
Example of an OR NOT - Operation:



EXCLUSIVE - OR - OPERATION

The circuit shows an exclusive-OR operation (X), with which the output 1.0 is switched on (signal status 1) if *only one* of the inputs exhibits a signal status of 1. In an contact-afflicted circuit, this can be realized only with normally open and closed contacts.

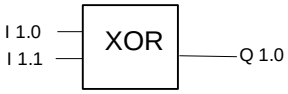
LAD



STL

```
X   I 1.0
X   I 1.1
=   Q 1.0
```

FBD

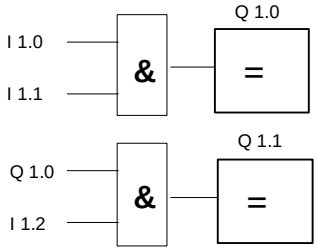


Caution: *The exclusive- OR- Operation should only be used with exactly two inputs.*

QUERY OF OUTPUTS

For the switching on of the outputs Q 1.0 and Q 1.1, different conditions apply. In these cases a current path and/or an operation symbol must be planned for each output. There the automation equipment can query not only the signal status of inputs, outputs, bit memories, etc. It will also query the outputs Q 1.1 and Q 1.0 from the AND operation.

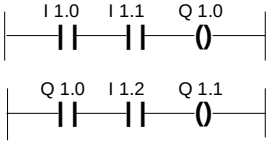
FBD



STL

```
A   I 1.0
A   I 1.1
=   Q 1.0
A   Q 1.0
A   I 1.2
=   Q 1.1
```

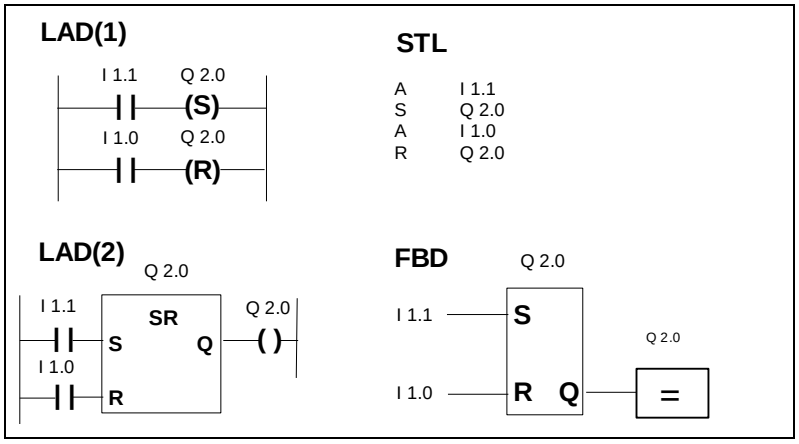
LAD



R - S – STORAGE FUNCTIONS

According to DIN 40900 and DIN 19239, an R-S memory function is represented as a rectangle with the set input S and the reset input R. A signal status 1 at the set input S sets the memory function. A signal status 1 at the reset input R results in the resetting of the memory function. A signal status 0 at the inputs R and S does not change the previously set condition. Should a signal status 1 be applied to both inputs R and S simultaneously, the function will be set or reset. This priority resetting or setting must be considered with programming.

RESET DOMINANT

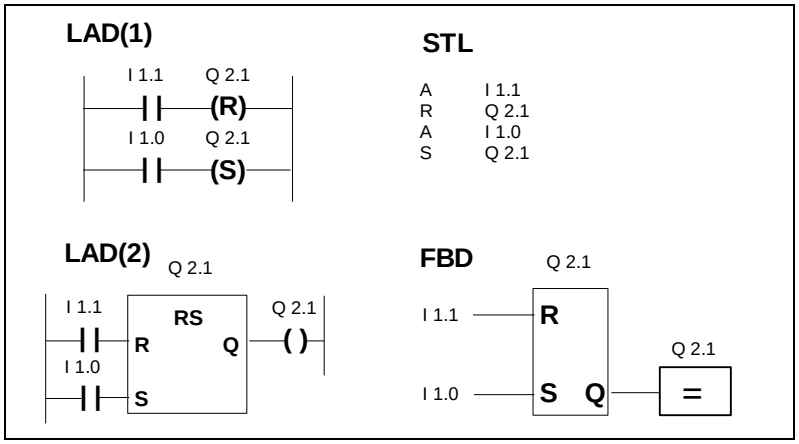


The last operations programmed are worked on by the control with priority. In the example the set operation is first implemented; the output Q 2.0 is again reset and remains reset for the remainder of program processing.

This brief setting of the output is accomplished only in the process image. A signal status on the pertinent I/O rack is not affected during program processing.

SET DOMINANT

The exit Q 2.1 in this example is set with priority.

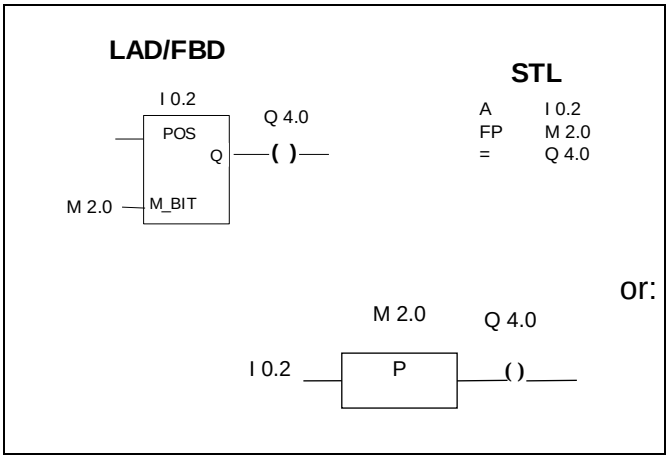


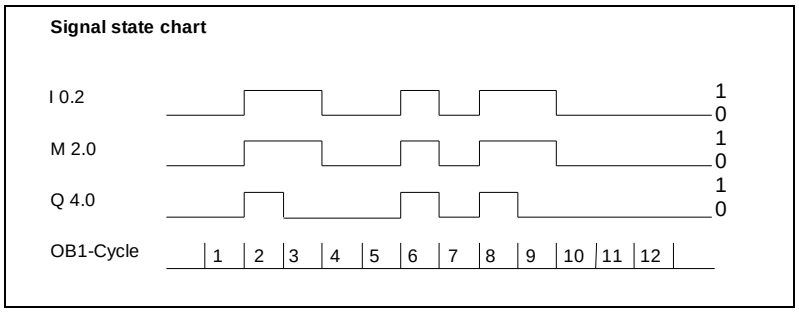
EDGE OPERATIONS

The edge (flank) operations collect in contrary to a static signal status "0" and "1" the *signal change* e.g. of a input. The program of an edge operation corresponds to an edge-recognizing contact in a relay circuit.

POSITIVE EDGE (FP)

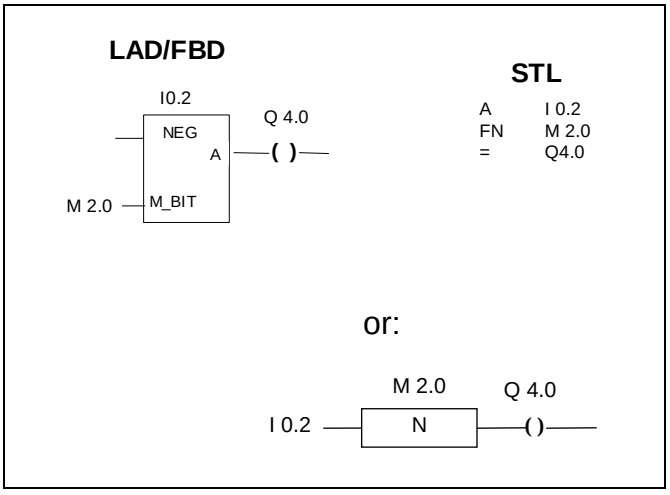
If a rising (positive) edge (change from “0“ to “1“) is recognized by I 0.2, then Q 4.0 for a OB1-Cycle is set to “1“. This output can be again used e.g. to set a memory bit. A rising edge is recognized, as the automation system stores the RLO, which supplied the operation A, in the edge memory bit M 2.0 and compares it with the RLO of the preceding cycle.
The advantage of the second type of representation in LAD/FBD is that logical operations can also be present at the input of the edge operation.

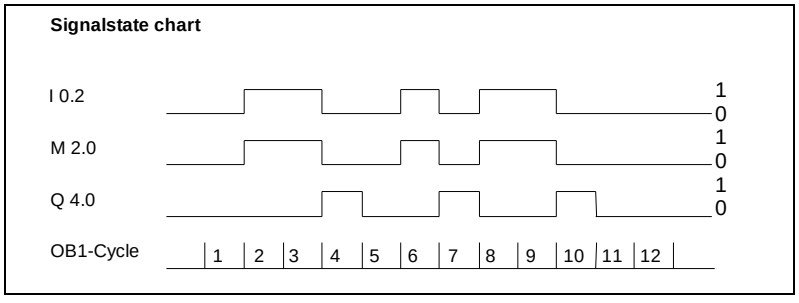




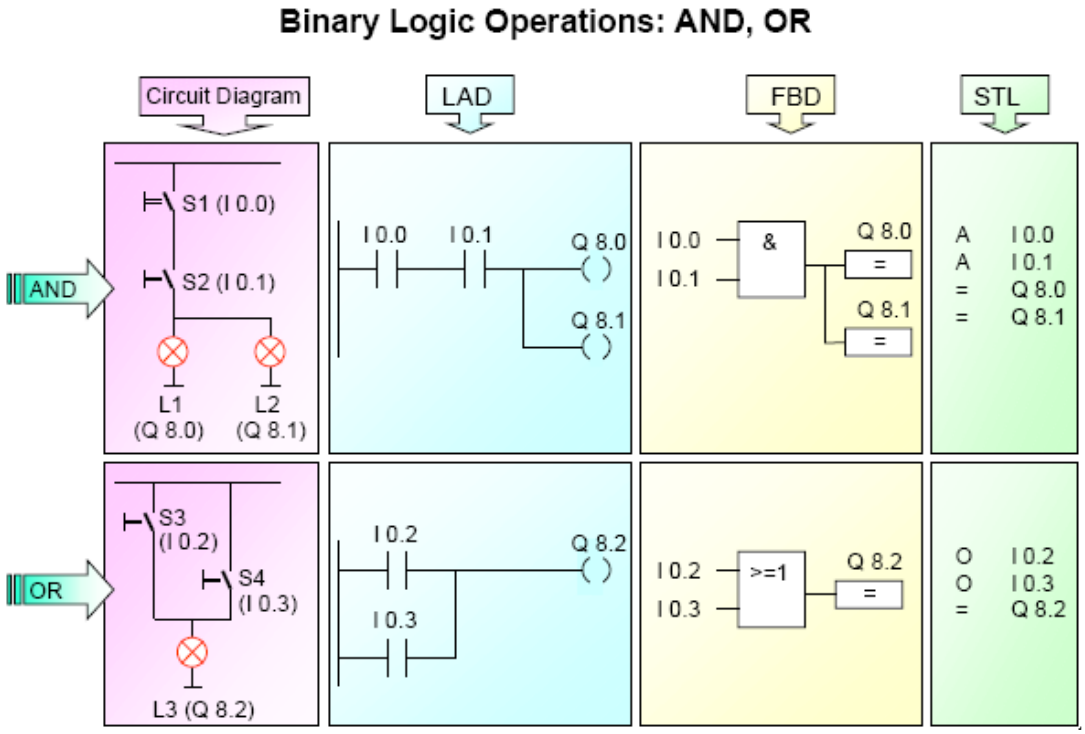
NEGATIVE EDGE (FN)

If a falling (negative) edge (change of “1“ to “0“) is recognized by I 0.2, then Q 4.0 for a OB1-Cycle is set to “1“. This output can be used again e.g. to set a memory bit. A falling edge is recognized, as the automation system stores the RLO, which supplied the operation A in the edge memory bit M 2.0, and compares it with the RLO of the preceding cycle. The advantage of the second type of representation in LAD/FBD is that logic operations can also be present at the input of the edge operation.





Binary Operation:



Logic Table:

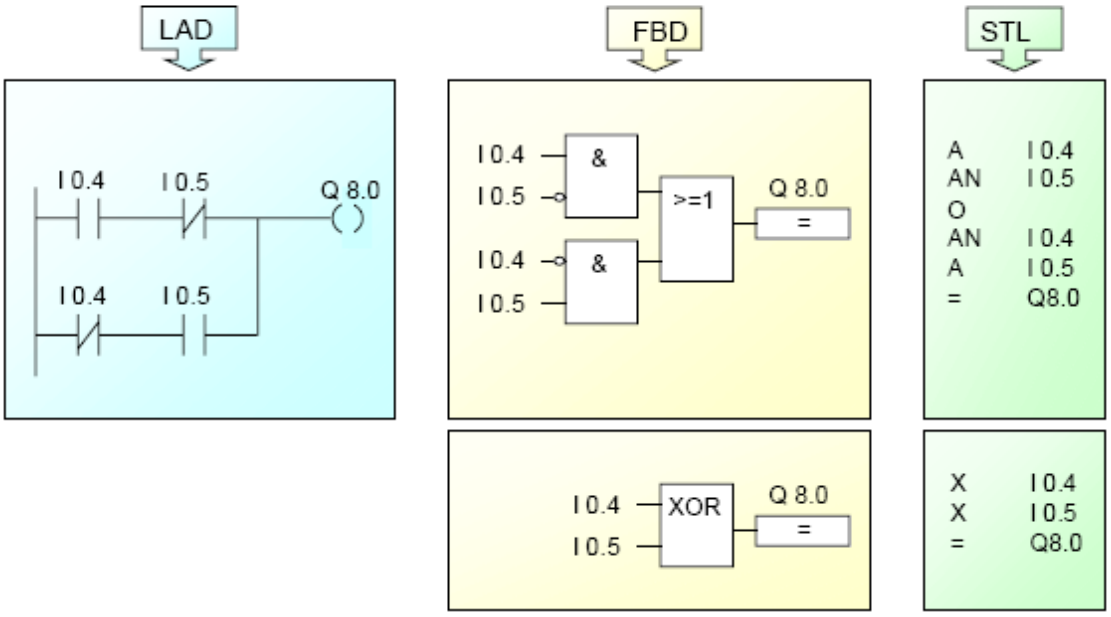
AND

I 0.0	I 0.1	Q 8.0 / Q8.1
0	0	
0	1	
1	0	
1	1	

OR

I 0.2	I 0.3	Q 8.2
0	0	
0	1	
1	0	
1	1	

Binary Logic Operations: Exclusive OR (XOR)



Logic table:

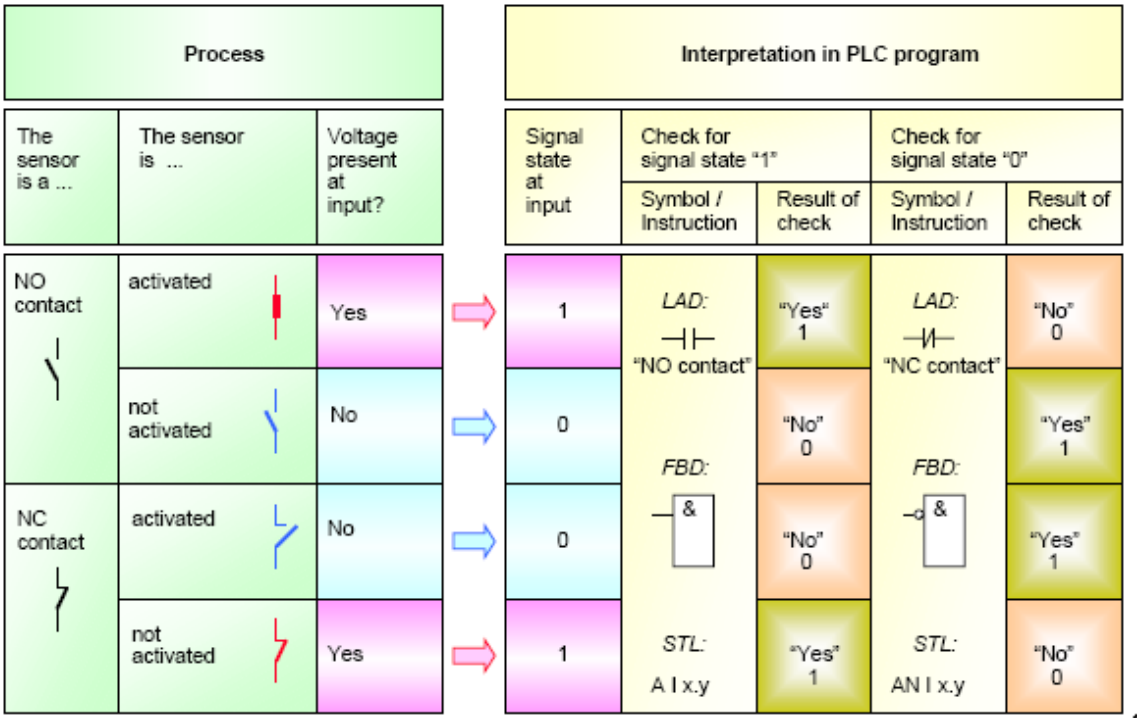
XOR	I 0.4	I 0.5	Q 8.0
	0	0	
	0	1	
	1	0	
	1	1	

Rule

The following rule is valid for the logic operation of two addresses after XOR: the output has signal state "1", when one and only one of the two checks is fulfilled.

Careful! This rule cannot be generalized to "one and only one of n" ! for the logic operation of several addresses after XOR !! As of the third XOR instruction, the old RLO is gated with the new result of check after XOR.

Normally Open and Normally Closed Contacts, Sensors and Symbols



Process
The use of normally open or normally closed contacts for the sensors in a controlled process depends on the safety regulations for that process.
Normally closed contacts are always used for limit switches and safety switches, so that dangerous conditions do not arise if a wire break occurs in the sensor circuit.
Normally closed contacts are also used for switching off machinery for the same reason.

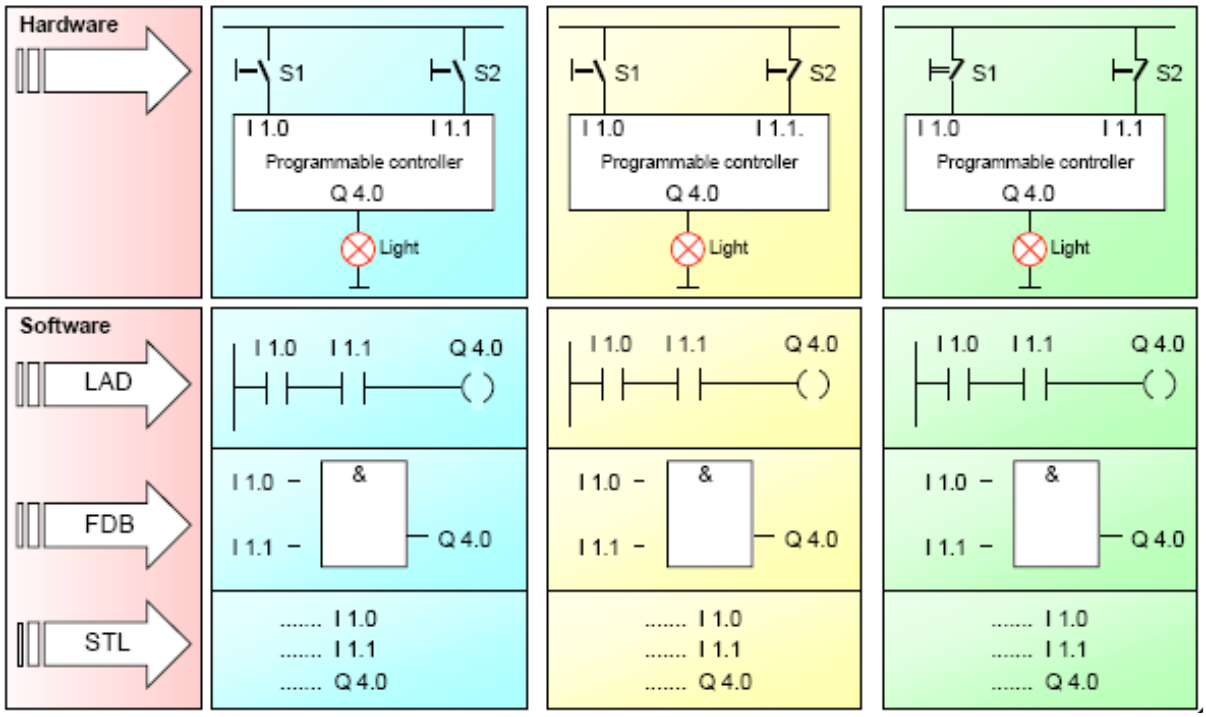
Symbols
In LAD, a symbol with the name "NO contact" is used for checking for signal state "1" and a symbol with the name "NC contact" to check for signal state "0". It makes no difference whether the process signal "1" is supplied by an activated NO contact or a non-activated NC contact.

Example
If an NC contact in the machine is not activated, the signal in the process image table will be "1". You use the NO contact symbol in LAD to check for a signal state of "1".

General:
The "NC contact" symbol delivers the result of check "1" when the checked address state or status is "0".

Exercise

Goal: In all three examples, the light should be on when S1 is activated and S2 is not activated!



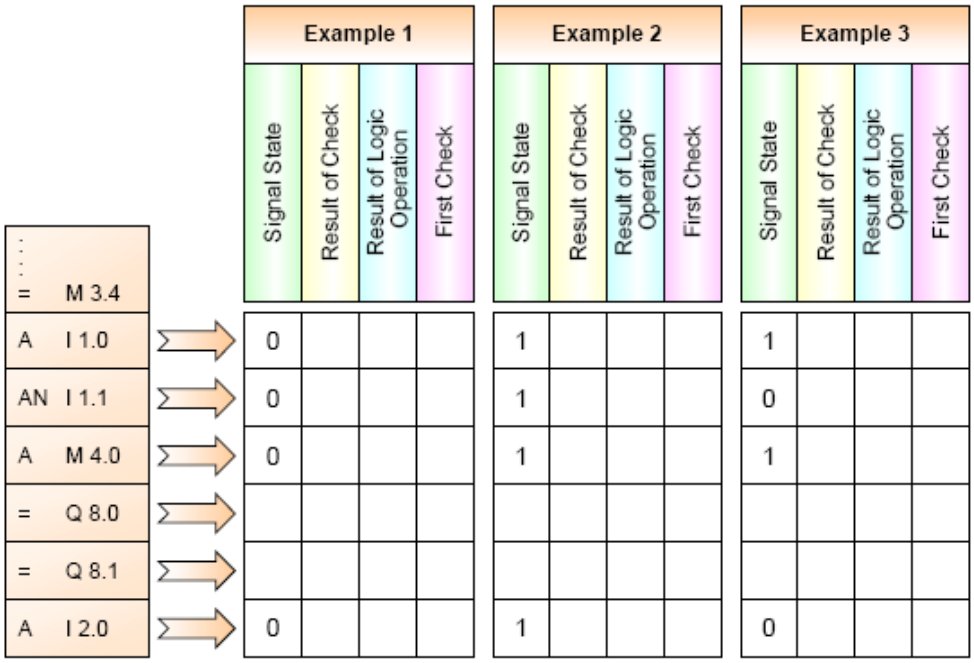
Exercise

Complete the programs above to obtain the following functionality: When switch S1 is activated and switch S2 is not activated, the light should be ON in all three cases.

Note !

The terms "NO contact" and "NC contact" have different meanings depending on whether they are used in the process hardware context or as symbols in the software.

Result of Logic Operation, First Check, and Examples



Signal State

A logic operation is made up of a series of instructions to check the states of signals (inputs **(I**), outputs (**Q**), bit memories (**M**), timers (**T**), counters (**C**) or data bits (**D**)) and instructions to set Q,M,T,C or D.

Result of Check

When the program is executed, the result of check is obtained. If the check condition is fulfilled, the result of check is “1”. If the check condition is not fulfilled, the result of check is “0”.

First Check

The first check that follows an RLO limiting operation (such as S, R, CU, =) or the first check in a logic string is called a First Check (FC) since the result of this check - regardless of the last RLO - is accepted as the new RLO.

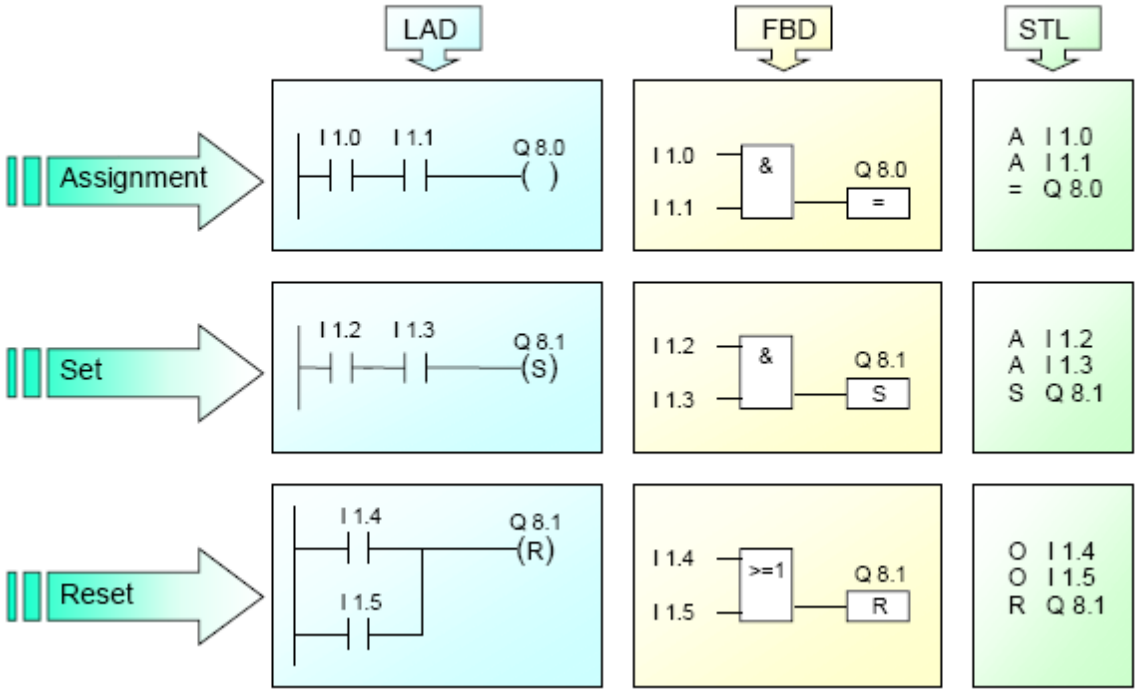
Result of Logic Operation

When the next check instructions are executed, the result of logic operation is gated with the result of check and a new RLO is obtained. When the last check instruction in a logic operation has been executed, the RLO remains the same. A number of instructions using the same RLO can follow.

Note

The result of the first check is stored without being subjected to a logic operation. Therefore, it makes no difference whether you program the first check with an AND or an OR instruction in STL. To convert your program to one of the other programming languages, you should, however, always program using the correct instruction.

Assignment, Setting, Resetting

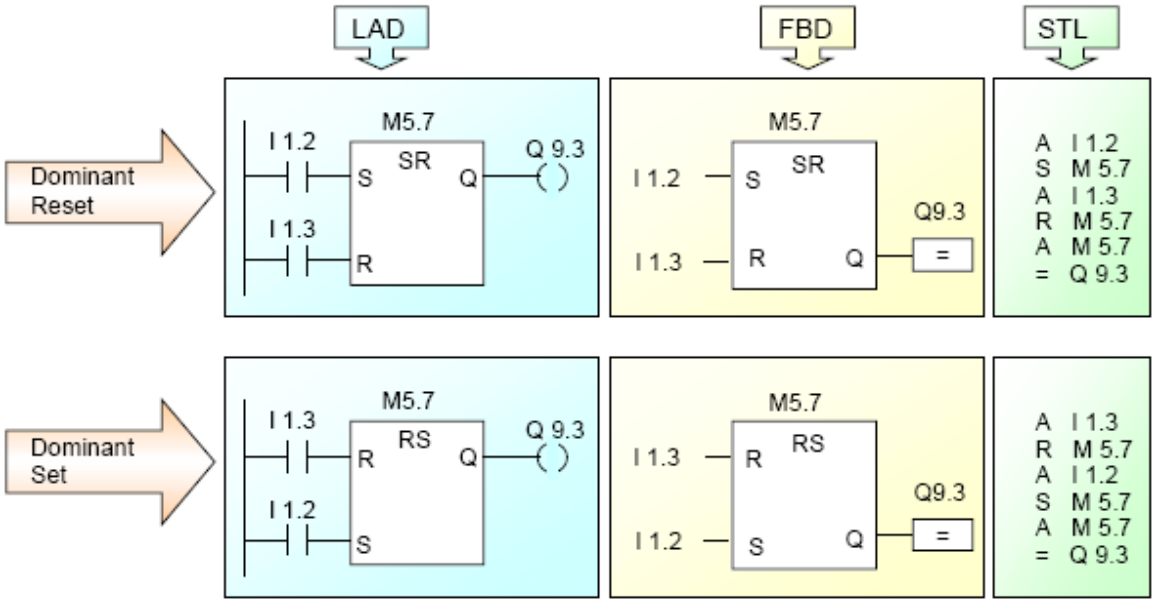


Assignment
An assignment passes the RLO on to the specified address (Q, M, D). When the RLO changes, the signal state of that address also changes.

Set
If RLO= "1", the specified address is set to signal state "1" and remains set until another instruction resets the address.

Reset
If RLO= "1", the specified address is reset to signal state "0" and remains in this state until another instruction sets the address again.

Setting / Resetting a Flip Flop

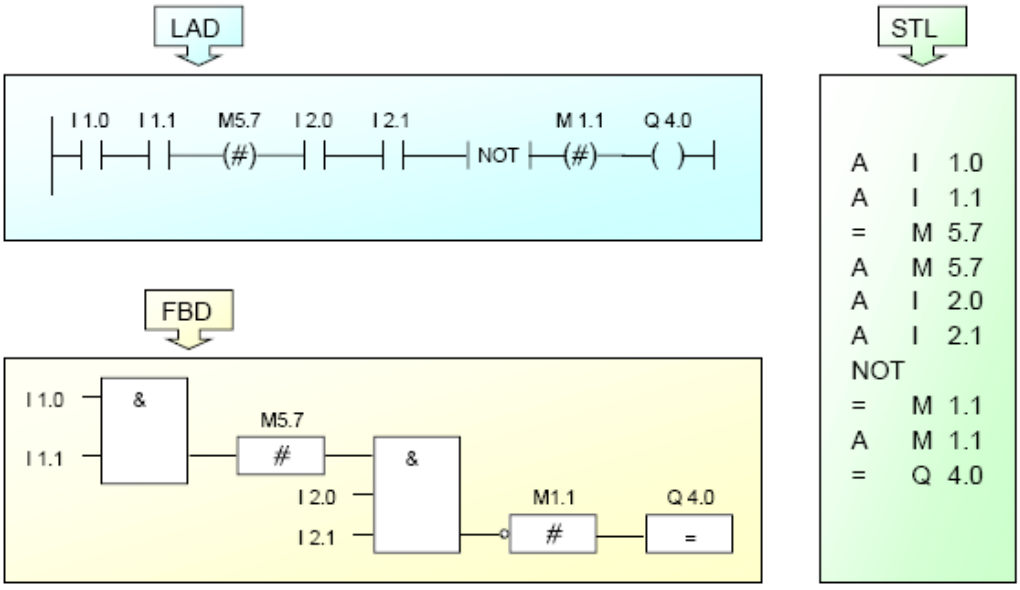


Flip Flop
A flip flop has a Set input and a Reset input. The memory bit is set or reset, depending on which input has an RLO=1. If there is an RLO=1 at both inputs at the same time, the priority must be determined.

Priority
In LAD and FBD there are different symbols for Dominant Set and Dominant Reset memory functions. In STL, the instruction that was programmed last has priority.

Note
If an output is set with a set instruction, the output is reset on a complete restart of the CPU. If M 5.7 in the example above has been declared retentive, it will remain in the set state after a complete restart of the CPU, and the reset output Q 9.3 will be assigned the set state again.

Midline Output Coil

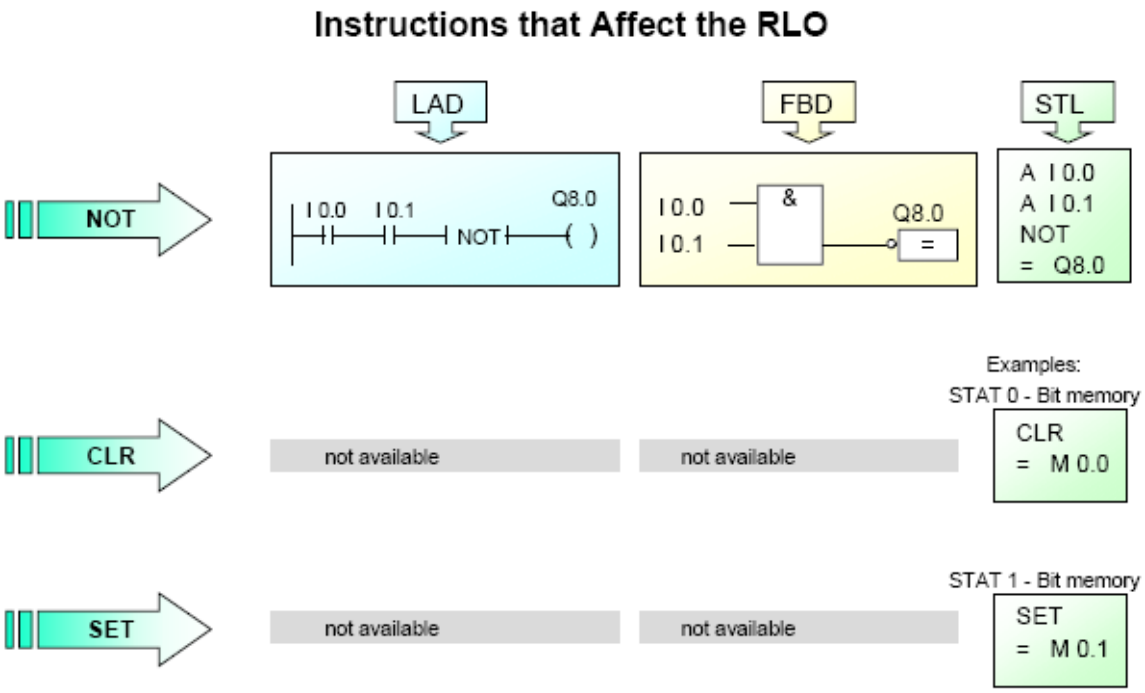


Midline Output Coil

The midline output coil exists only in the LAD and FBD graphic languages. It is an intermediate assignment element with assignment function that assigns the current RLO at a specified address (M5.7 in the slide). The midline output coil provides this same address in the same network for subsequent gating. In the STL language, this is equivalent to

= M 5.7
A M 5.7

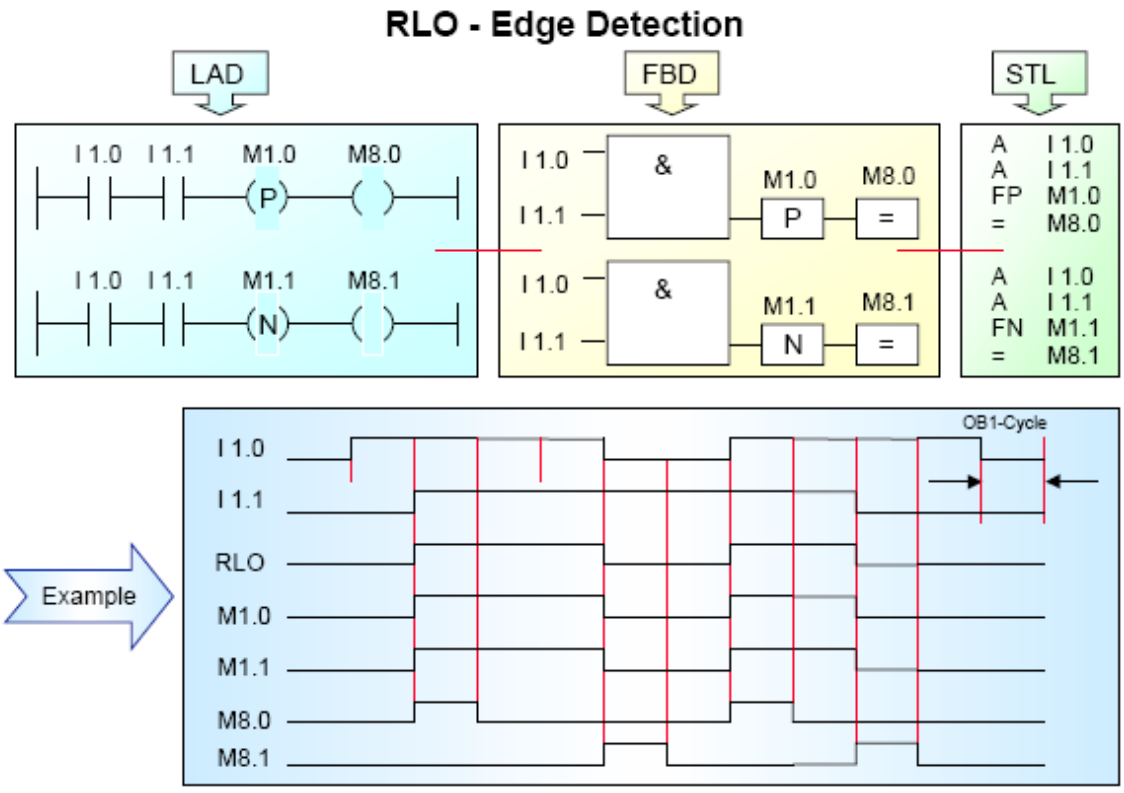
In the LAD language, when connected in series with other elements, the "midline output coil" instruction is inserted in the same way as a contact.



NOT
The NOT instruction inverts the RLO.

CLR
The CLEAR instruction sets the RLO to "0" without pre-conditions (available only in STL at present !). The CLR instruction completes the RLO, thus the next scan becomes a first check.

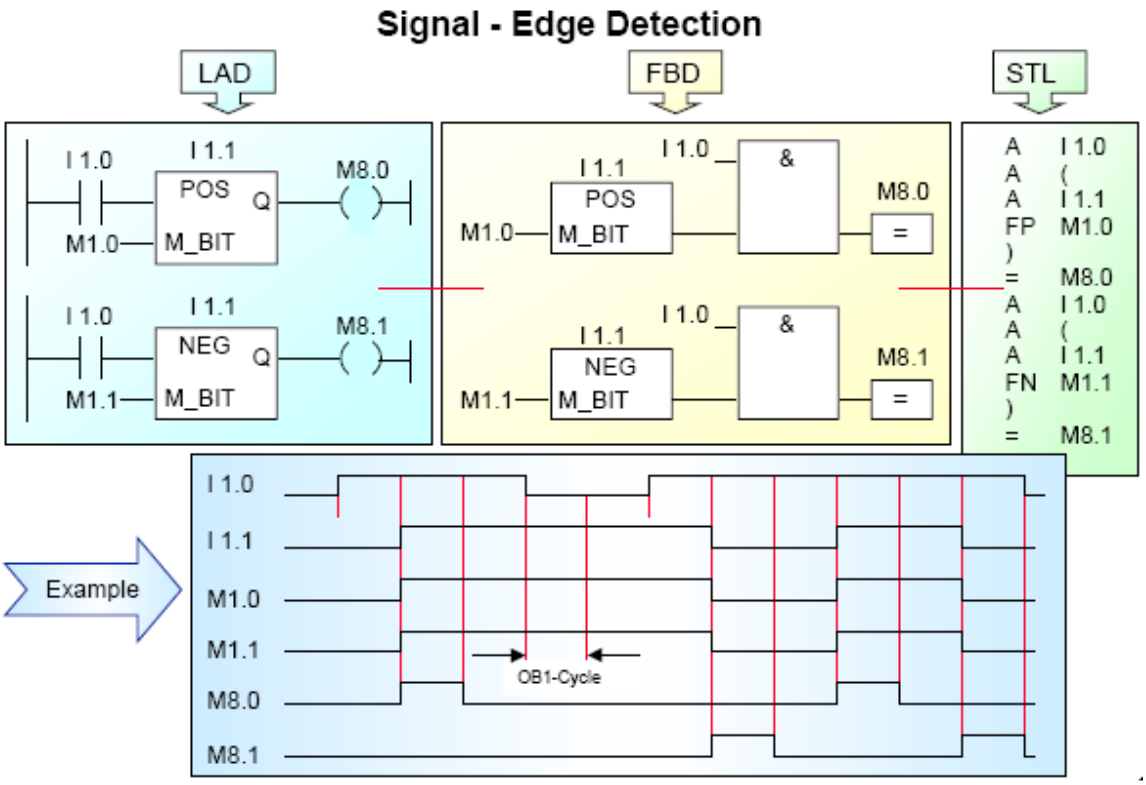
SET
The SET instruction sets the RLO to "1" without pre-conditions (available only in STL at present !). The SET instruction completes the RLO, thus the next scan becomes a first check.



RLO Edge Detection
An "RLO edge" detection is when the result of a logic operation changes from "0" to "1" or from "1" to "0".

Positive Edge
(Positive RLO Edge Detection) detects a signal change in the address (M1.0) from "0" to "1", and displays it as RLO = "1" after the instruction (such as at M 8.0) for one cycle.
To enable the system to detect the edge change, the RLO must be saved in an FP bit memory (such as M 1.0), or a data bit.

Negative Edge
(Negative RLO Edge Detection) detects a signal change in the address (M1.1) from "1" to "0" and displays it as RLO = "1" after the instruction (such as at M 8.1) for one cycle.
To enable the system to detect the edge change, the RLO must be saved in an FN bit memory (such as M 1.1), or a data bit.



Signal Edge Example

A "signal edge" is when a signal changes its state. Input I 1.0 acts as a static enable. Input I 1.1 is to be monitored dynamically and every signal change is to be detected.

Positive Edge

When the signal state at I 1.1 changes from "0" to "1", the "POS" check instruction results in signal state "1" at output Q for one cycle, provided input I 1.0 also has signal state "1" (as in the example above). To enable the system to detect the edge change, the signal state of I 1.1 must also be saved in an M_BIT (bit memory or data bit) (such as M 1.0).

Negative Edge

When the signal state at I 1.1 changes from "1" to "0", the "NEG" check instruction results in signal state "1" at output Q for one cycle, provided input I 1.0 has signal state "1" (as in the example above). To enable the system to detect the edge change, the signal state of I 1.1 must also be saved in an M_BIT (bit memory or data bit) (such as M 1.1).

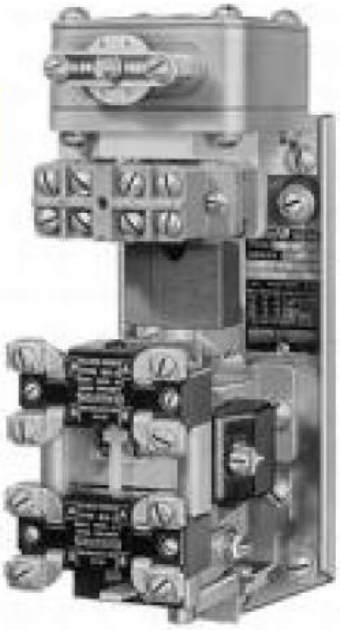
PART FIVE

PROGRAMMING TIMERS

Timers:

There are very few industrial control systems that do not need at least one or two timed functions. They are used to activate or de-activate a device after a preset interval of time.

Time Delay Relay



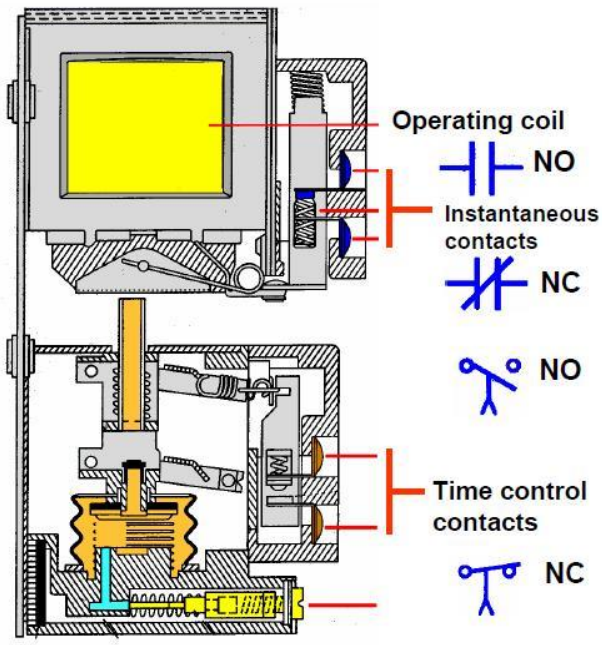
Solid-State Timer

Time delay relays and solid-state timers are used to provide a time delay. They may have displays, pots or other means of operator interface for time settings and electromechanical or solid state outputs.

On-Delay Timing Relay:

Non-timed contacts are controlled directly by the timer coil, as in a general-purpose control relay.

When the coil is energized, the timed contacts are prevented from opening or closing until the time delay period has elapsed. However, when the coil is de-energized, the timed contacts return instantaneously to their normal state..



Timed Contact Symbols

On-Delay Symbols

Normally open, timed closed contact (NOTC)

Contact is open when relay coil is de-energized

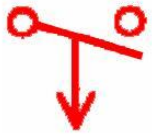
When relay is energized, there is a time delay in closing

Normally closed, timed open contact (NCTO)

Contact is closed when relay coil is de-energized

When relay is energized, there is a time delay in opening

Off Delay Symbols

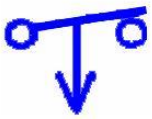


Normally open, timed open contacts (NOTO).

Contact is normally open when relay coil is de-energized.

When relay coil is energized, contact closes instantly.

When relay coil is de-energized, there is a time delay before the contact opens.



Normally closed, timed closed contacts (NCTC).

Contact is normally closed when relay coil is de-energized.

When relay coil is energized, contact opens instantly.

When relay coil is de-energized, there is a time delay before the contact closes.

On-Delay Relay Timer Circuit (NOTC Contact):

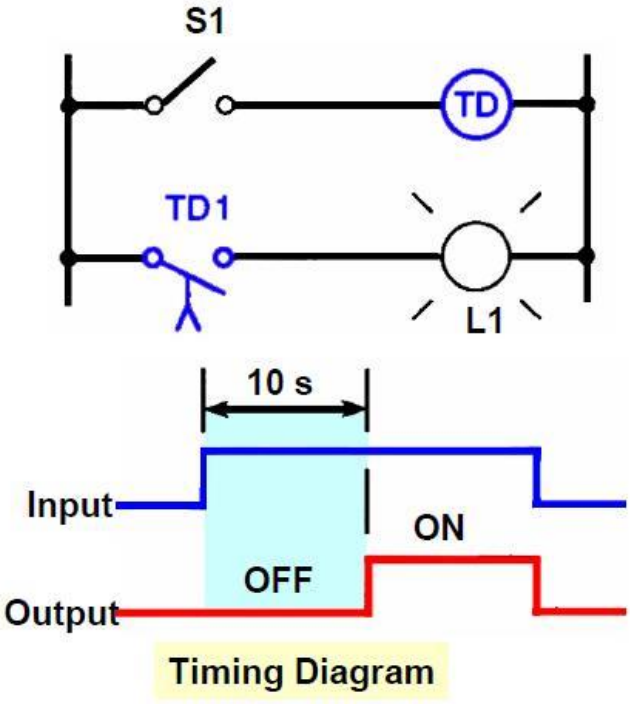
Sequence of Operation

S1 open, TD de-energized, TD1 open, L1 is off.

S1 closes, TD energizes, timing period starts, TD1 still open, L1 is still off.

After 10 s, TD1 closes, L1 is switched on.

S1 is opened, TD de-energizes, TD1 opens instantly, L1 is switched off.



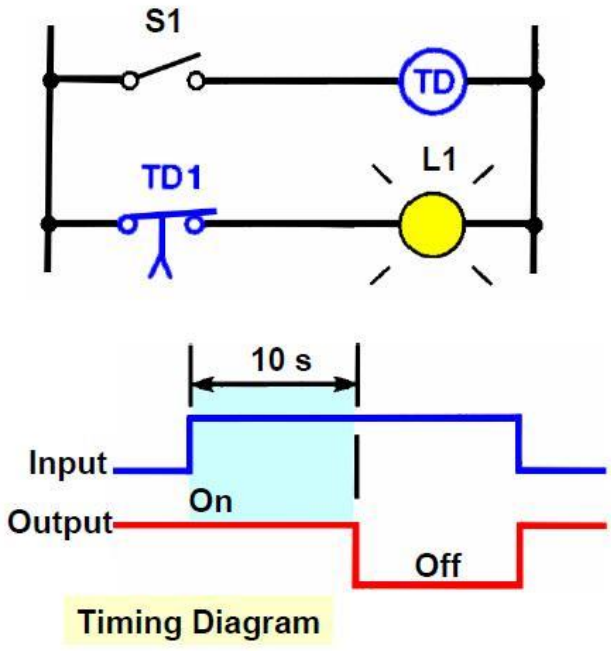
On-Delay Relay Timer Circuit (NCTO Contact):

Sequence of Operation

S1 open, TD de-energized, TD1 closed, L1 is on.

S1 close, TD energizes, timing period starts, TD1 is still closed, L1 is still on.

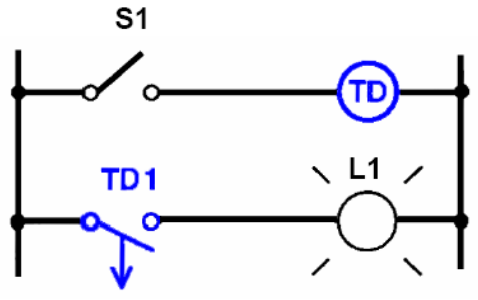
After 10 s, TD1 opens, L1 is switched off.



Off-Delay Relay Timer

Circuit (NOTO Contact):

Sequence of Operation

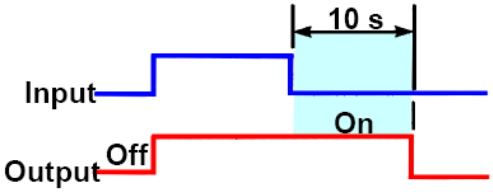


S1 open, TD de-energized, TD1 open, L1 is off.

S1 closes, TD energizes, RD1 closes instantly, L1 is switched on.

S1 is opened, TD de-energizes, timing period starts, TD1 is still closed, L1 is still on.

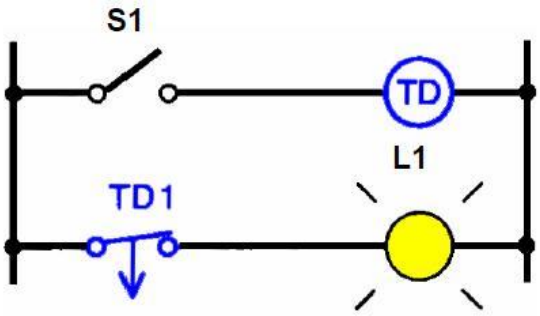
After 10 s, TD1 opens, L1 is switched off.



Timing Diagram

Off-Delay Relay Timer Circuit (NCTC Contact):

Sequence of Operation

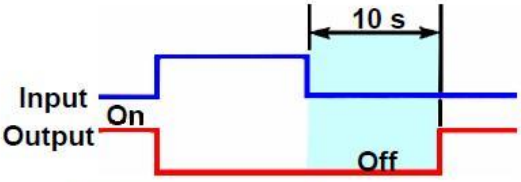


S1 open, TD de-energized, TD1 closed, L1 is on.

S1 closes, TD energizes, TD1 opens instantly, L1 is switched off.

S1 is opened, TD de-energizes, timing period starts, TD1 is still off.

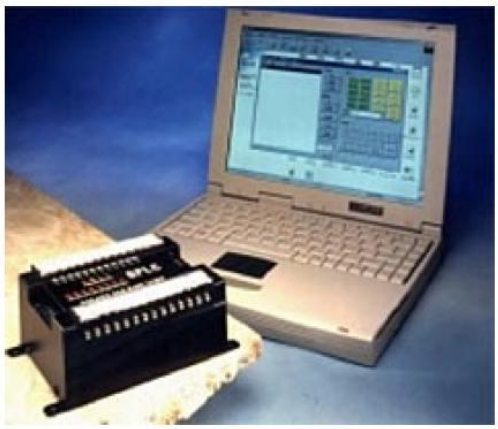
After 10 s, TD1 closes, L1 is switched on.



Timing Diagram

Programmed Timer Instructions:

PLC timers are output instructions that provide the same functions as timing relays and solid state timers.

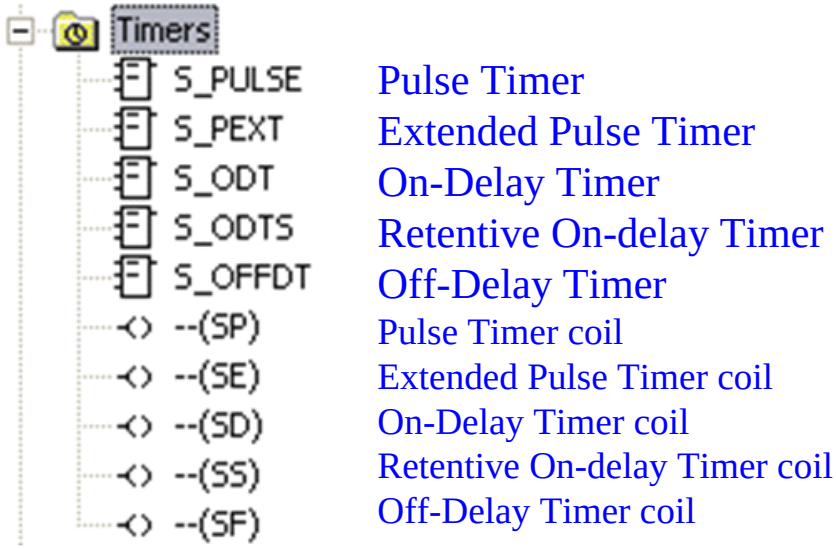


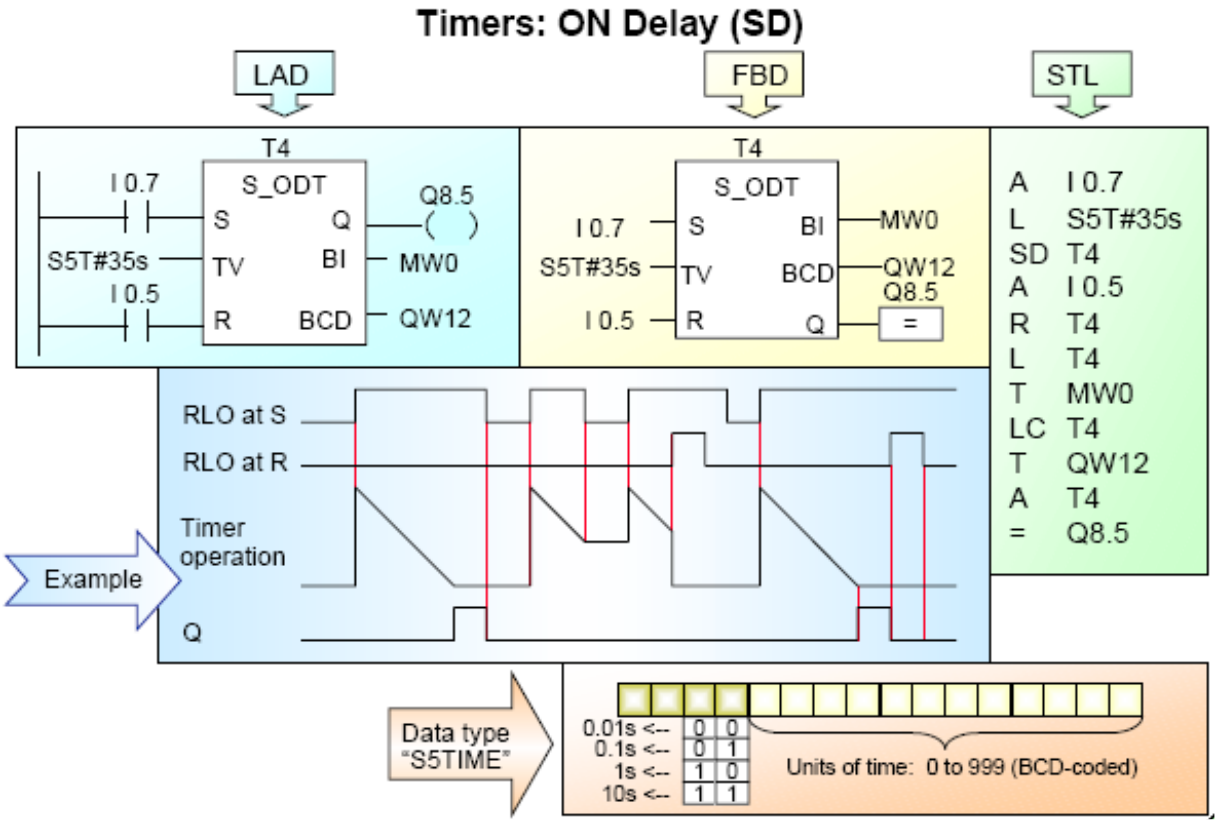
Some advantages of PLC timers:

- Their settings can be altered easily.
- The number of PLC timers used can be increased or decreased by programming changes without wiring changes.
- Timer accuracy and repeatability are extremely high.

Step7 Timer Commands:

Simatic S7 has 5 types of timers





Start

The timer starts when the RLO at the Start input "S" changes from "0" to "1". The timer starts with the time value specified at the Time Value "TV" for as long as the signal state at input "S" =1.

Reset

When the RLO at the Reset input "R" changes from "0" to "1", the current time value and the time base are deleted and the output "Q" is reset.

Digital Outputs

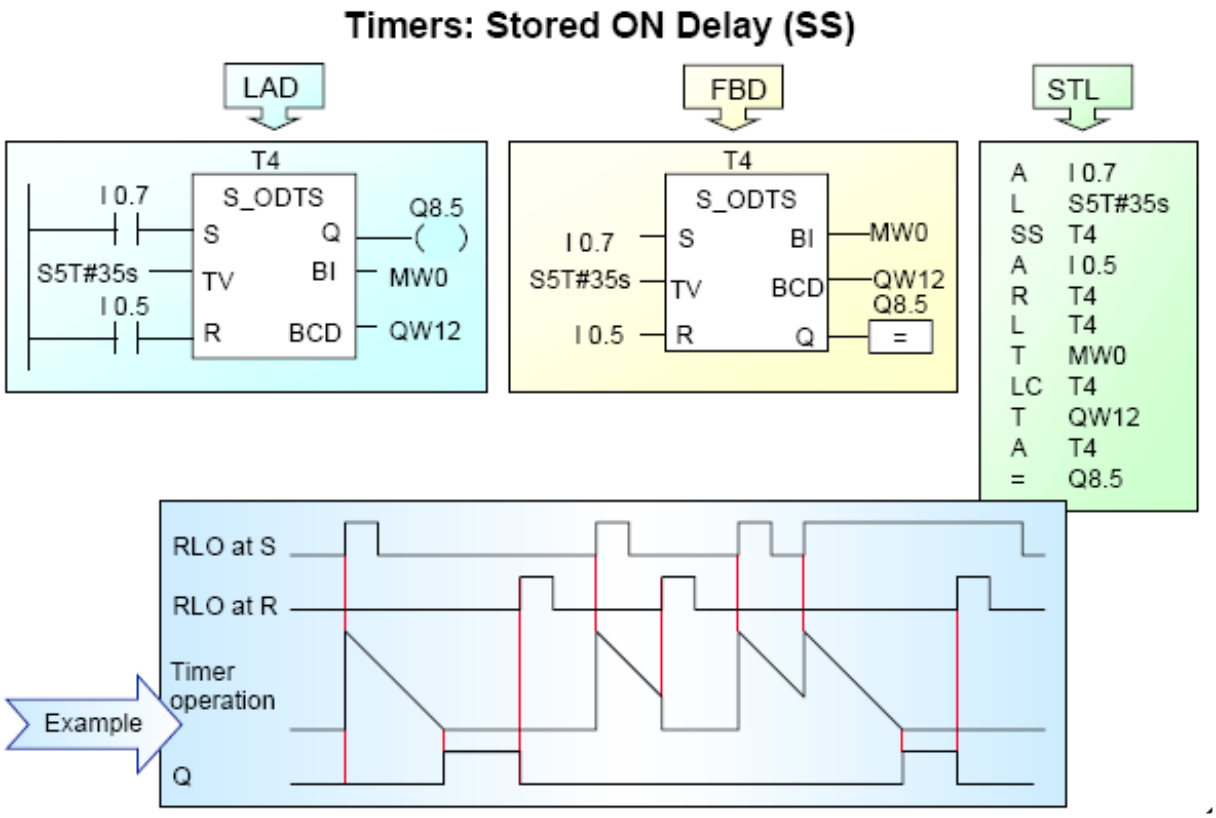
The current time value can be read as a binary number at the "BI" output and as a BCD number at the "BCD" output. The current time value is the initial value of "TV" minus the value for the time that has elapsed since the timer was started.

Binary Output

The signal at the "Q" output changes to "1" when the timer has expired without error and input "S" has signal state "1". If the signal state at the "S" input changes from "1" to "0" before the timer has expired, the timer stops running and output "Q" has a signal state "0".

Note

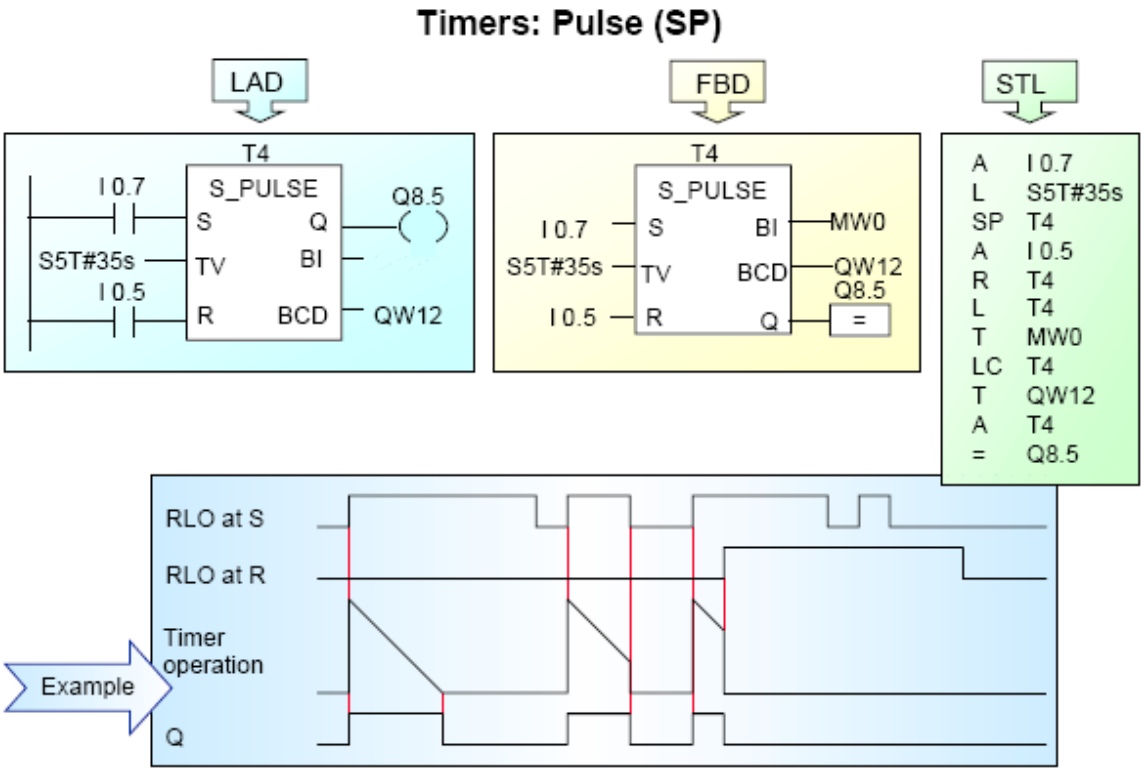
In STEP 7, you can also implement IEC conforming timers using SFBs. The use of system function blocks is dealt with in an advanced programming course.



Start
The stored-on-delay timer starts when the RLO at the "S" input changes from "0" to "1". The timer runs starting with the time value specified at input "TV" and continues to run even if the signal at input "S" changes back to "0" during that time. If the signal at the start input changes from "0" to "1" again while the timer is still timing down, the timer starts again from the beginning.

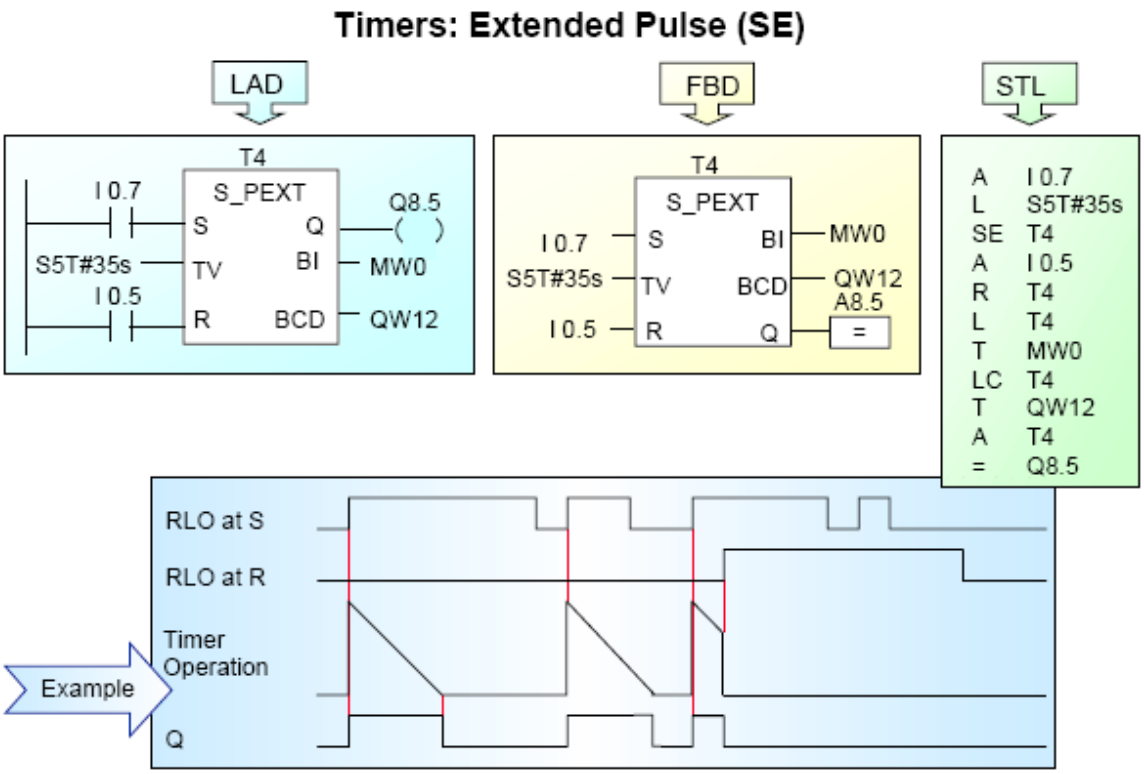
Reset
When the RLO at reset input "R" changes from "0" to "1", the current time value and the time base are deleted and output "Q" is reset.

Binary Output
The signal state at output "Q" changes to "1" when the timer has expired without error, regardless of whether the signal state at input "S" is still "1".



Start
The pulse timer starts when the RLO at the "S" input changes from "0" to "1". Output "Q" is also set to "1".

Reset
Output "Q" is reset when:
• the timer has expired, or
• the start "S" signal changes from "1" to "0", or
• the reset input "R" has a signal state of "1".



Start

The extended pulse timer starts when the RLO at the "S" input changes from "0" to "1". Output "Q" is also set to "1". The signal state at output "Q" remains at "1" even if the signal at the "S" input changes back to "0". If the signal at the start input changes from "0" to "1" again while the timer is running, the timer is restarted.

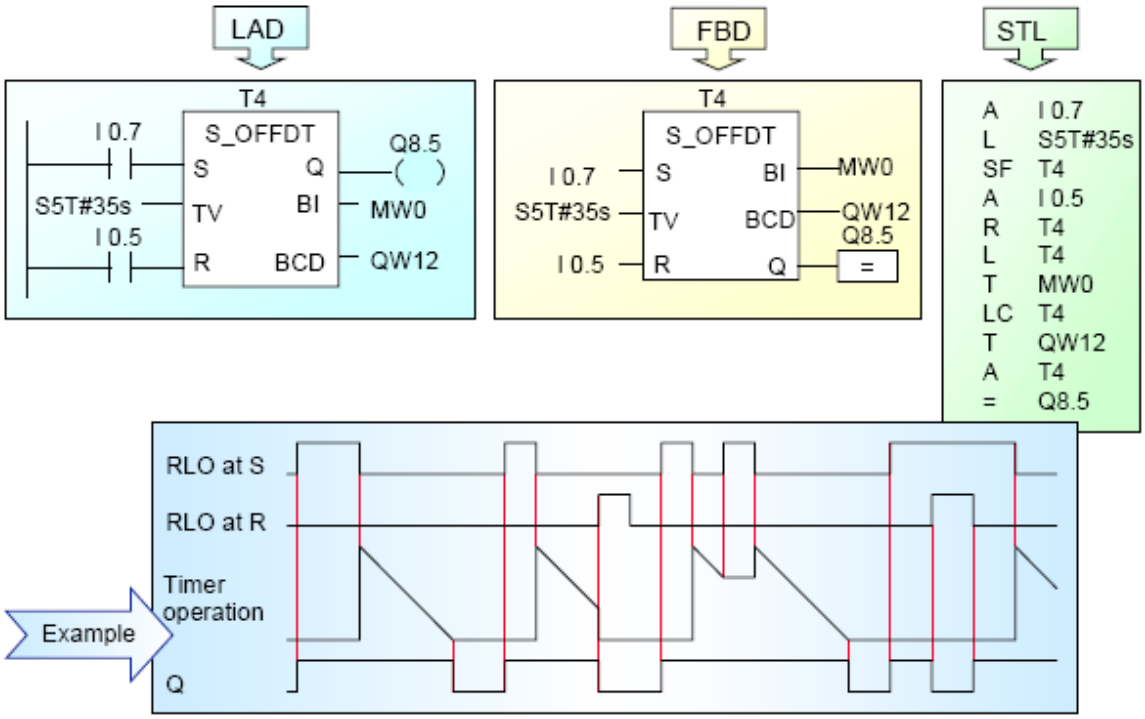
Reset

Output "Q" is reset when:

• the timer has expired, or

• the reset input "R" has a signal state of "1".

Timers: OFF Delay (SF)



Start

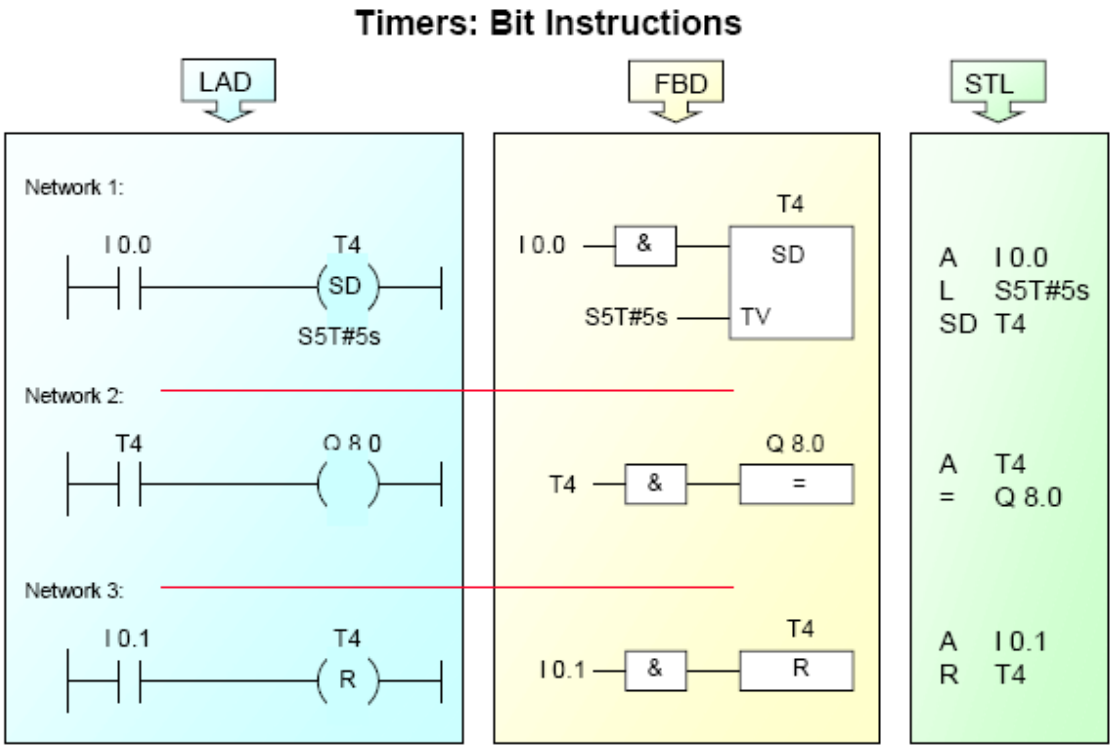
The off-delay timer starts when the RLO at the "S" input changes from "1" to "0". When the timer has expired, the signal state at output "Q" changes to "0". If the signal state at the "S" input changes from "0" to "1" while the timer is running, the timer stops. The next time the signal state at the "S" input changes from "1" to "0", it starts again from the beginning.

Reset

When the RLO at reset input "R" is "1", the current time value and the time base are deleted and output "Q" is reset. If both inputs (S and R) have signal states of "1", output "Q" is not set until the dominant reset is deactivated.

Binary Output

Output "Q" is activated when the RLO at the "S" input changes from "0" to "1". If input "S" is deactivated, output "Q" continues to have signal state of "1" until the programmed time has expired.



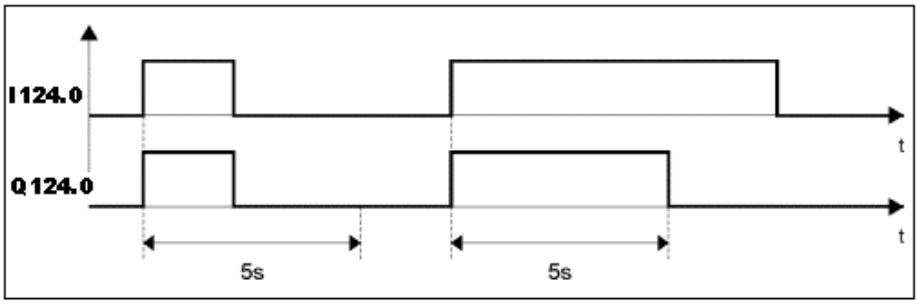
Bit Instructions

All timer functions can also be started with simple bit instructions. The similarities and differences between this method and the timer functions discussed so far are as follows:

- Similarities:
  - Start conditions at the "S" input
  - Specification of the time value
  - Reset conditions at the "R" input
  - Signal response at output "Q"
- Differences (for LAD and FBD):
  - It is not possible to check the current time value (there are no "BI" and "BCD" outputs).

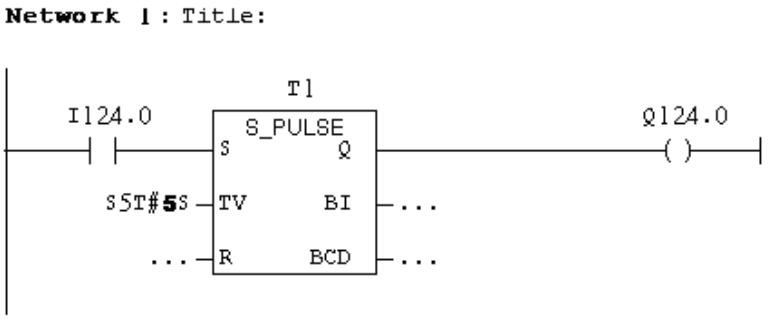
PULSE TIMER S_PULSE

The output Q124.0 is activated by the closure of input I124.0 and deactivated 5 seconds later. If the input is reopened during this period of time, the output is immediately deactivated.

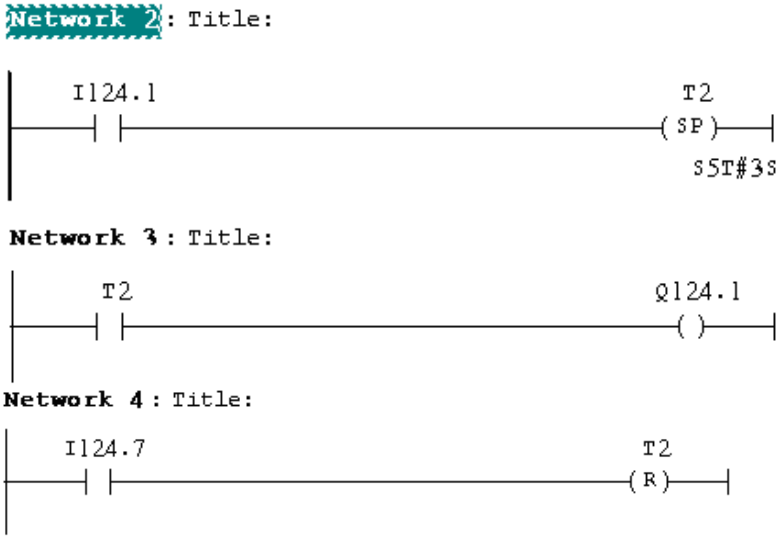


Time diagram of the pulse timer

The operation of the pulse timer is shown in the time diagram above. The first line represents the input signal and the second line its output.
The program ladder has been designed using a timer S_PULSE activated by the NO contact of I124.0, with a time constant equal to 5 seconds and the output connected to the coil of Q124.0.

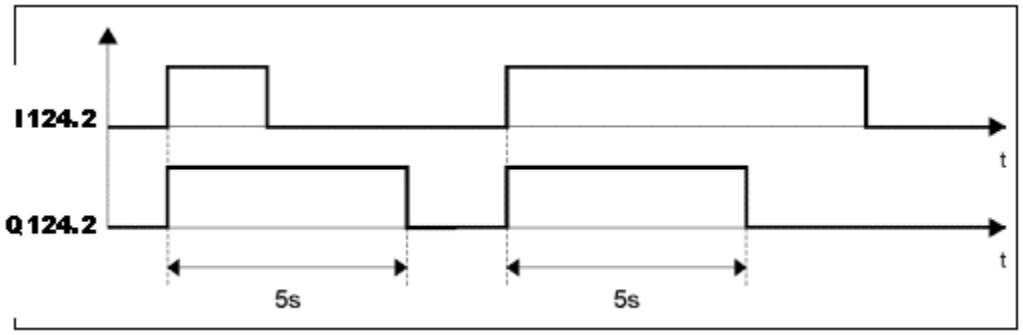


PULSE TIMER COIL -(SP)-



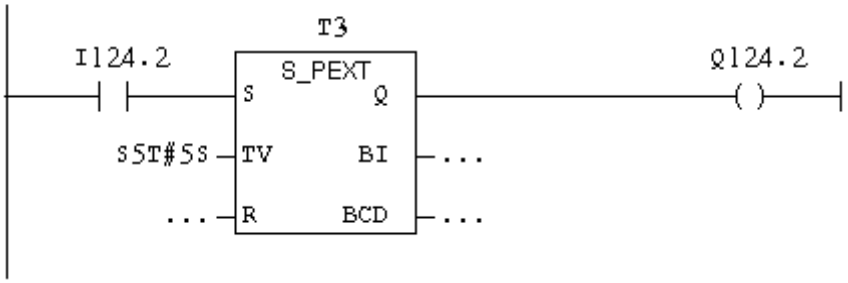
EXTENDED PULSE TIMER S_PEXT

The output Q124.2 is activated when inputI124.2 is closed and deactivated 5 seconds later, irrespective of whether the input is opened again during this period of time.



The operation of the pulse timer is shown in the time diagram above. The first line represents the input signal and the second line its output.
The program ladder has been designed using a timer S_PEXT (extended pulse) activated by the NO contact of I124.2 , with a time constant equal to 5 seconds and the output connected to the coil of Q124.2.

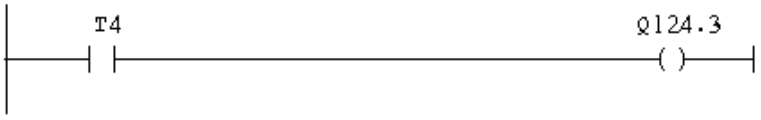
Network 5 : Title:



EXTENDED PULSE TIMER COIL -(SE)-

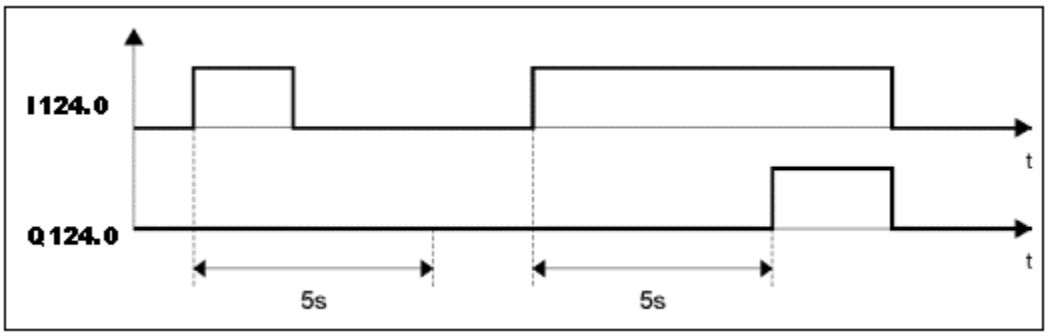
Network 6 : Title:

Network 7 : Title:

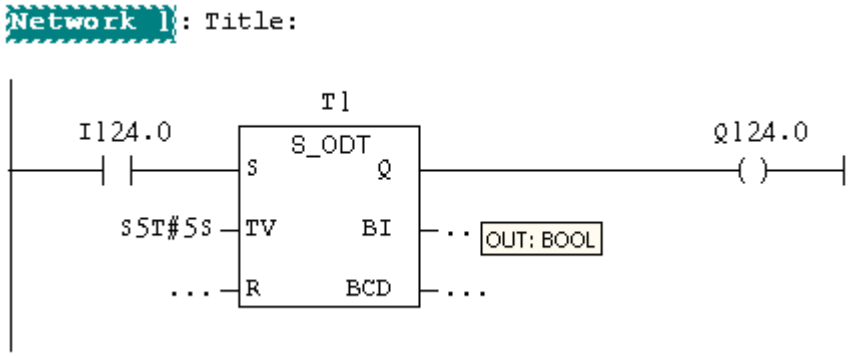


ON-DELAY TIMER S_ODT

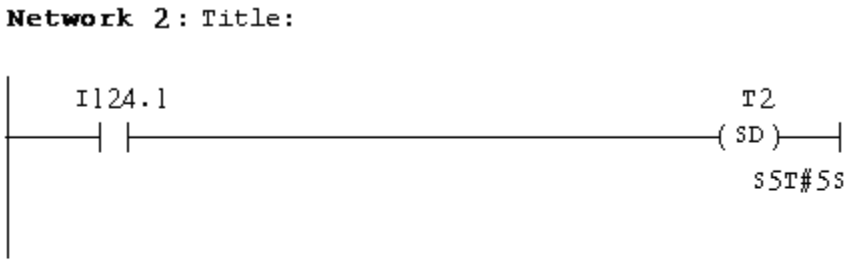
The output Q124.0 is activated 5 seconds after input I124.0 is closed. When the input is reopened, the output is deactivated.



The operation of the pulse timer is shown in the above time diagram. The first line represents the input signal and the second line is the corresponding output. The program ladder has been designed using a timer S_ODT (delayed activation) activated by the NO contact of I124.0, with a time constant equal to 5 seconds and the output connected to the coil of Q124.0.

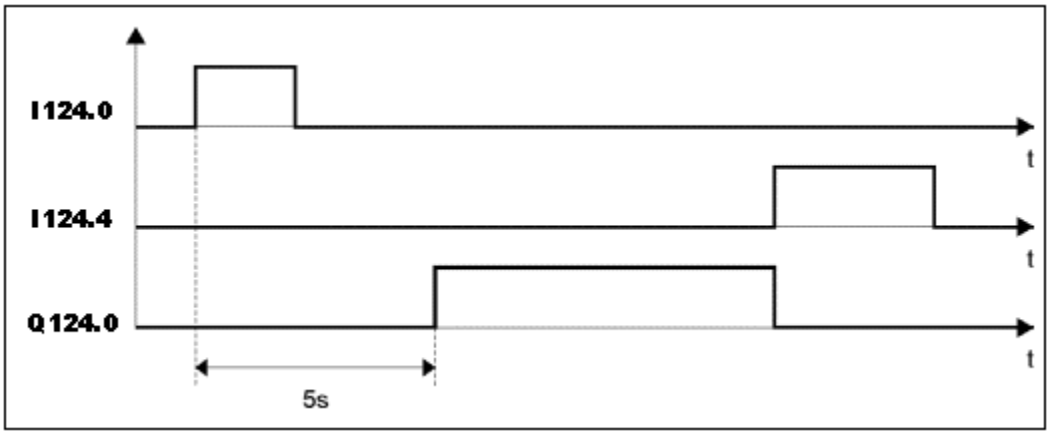


ON-DELAY TIMER COIL -(SD)-

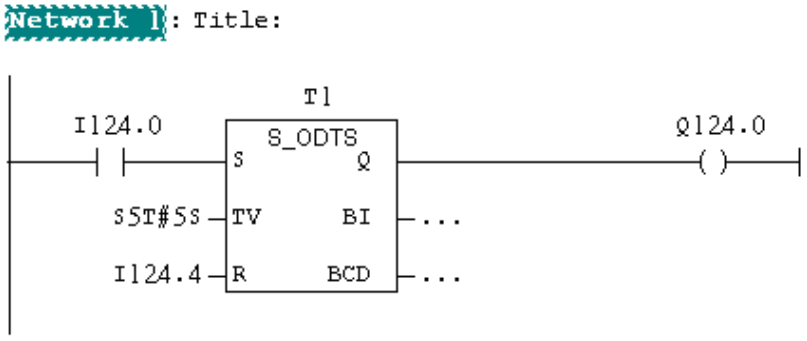


RETENTIVE ON-DELAY TIMER S_ODTS

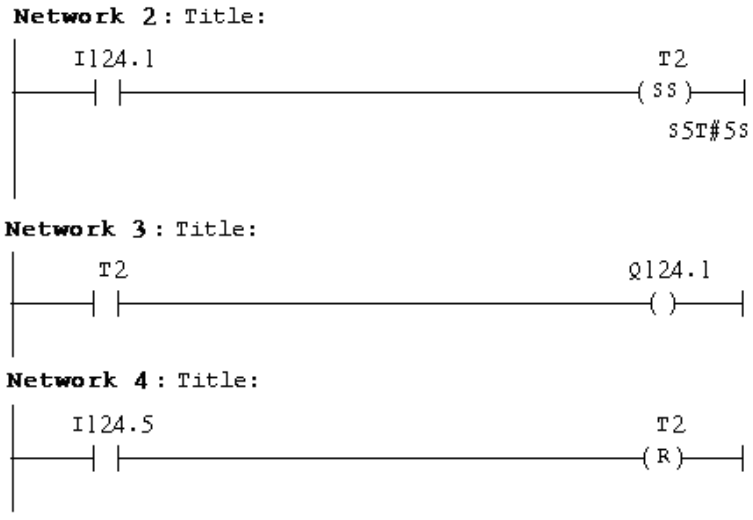
Output Q124.0 is activated 5 seconds after input I124.0 is closed (even though this input is opened again during this time) and deactivated with the closure of input



The operation of the retentive On-delay timer with reset can be obtained by comparing the first two lines and the last line of the above time diagram. The first two lines represent the input signals, and the last line is the resulting output.
The program ladder has been designed using a timer S_ODTS (retentive On-delay) activated by the NO contact of I124.0, with a time constant equal to 5 seconds, the reset connected to a NO contact of I124.4 and the output to the coil of Q124.0.

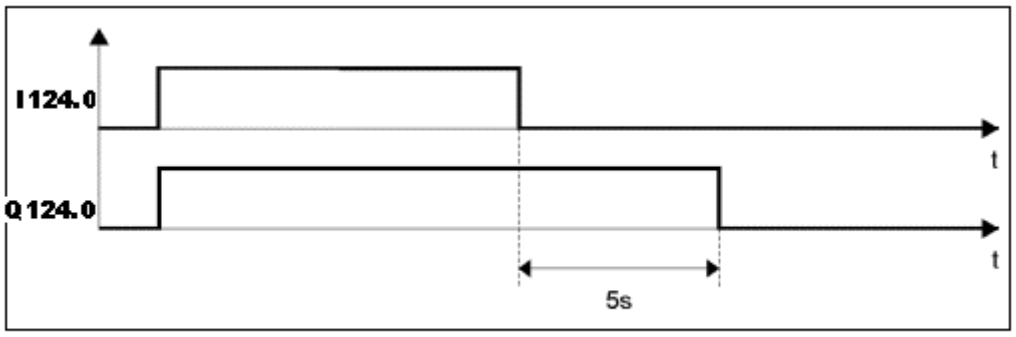


RETENTIVE ON-DELAY TIMER COIL –(SS)-

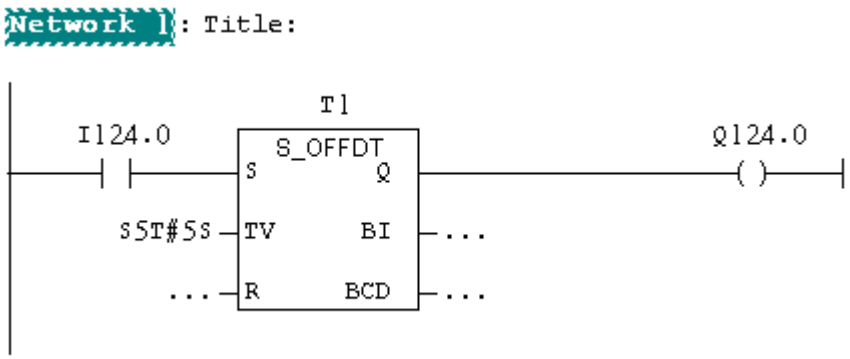


OFF-DELAY TIMER S_OFFDT

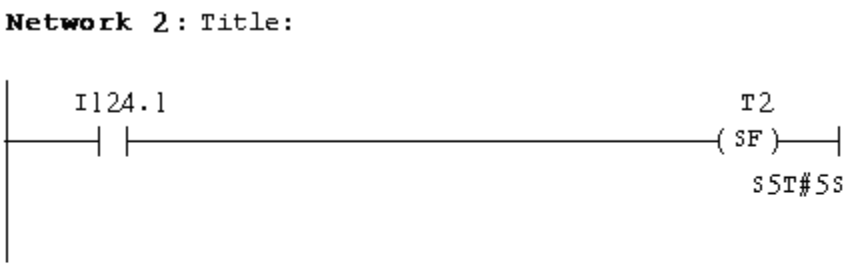
Output Q124.0 should activate when input I124.0 is closed and deactivate itself 5 seconds after it is reopened.



The operation of the Off-delay timer is shown in the time diagram. The first line represents the input signal and the second line is the corresponding output. The program ladder has been designed using a S_OFFDT (Off-delay timer) activated by the NO contact of I124.0, with a time constant equal to 5 a second and the output connected to the coil of Q124.0.



OFF-DELAY TIMER -(SF)-



PART SIX

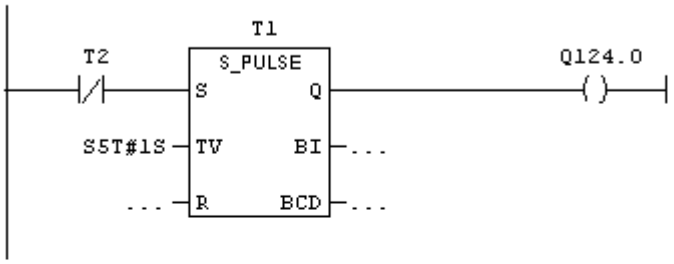
TIMER APPLICATIONS

FLASHING USING PULSE TIMERS

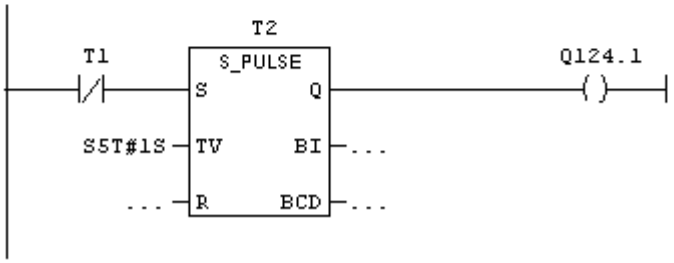
One of the most important applications of timers is to produce flashing signals. It is possible to obtain a flashing signal of the rate of one second using two pulse timers as shown below. This way of linking the two pulse timers can start the flashing automatically as soon as the CPU is in the RUN mode.

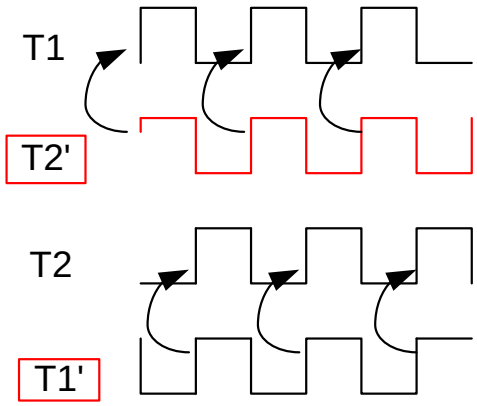
OB1 : "Main Program Sweep (Cycle)"

Network 1: FIRST TIME SCAN T2 IS OFF SO T2' Ignate the timer T1 to start



Network 2 : Waiting for T1 to finish T1=off T1' will enable T2 to start.

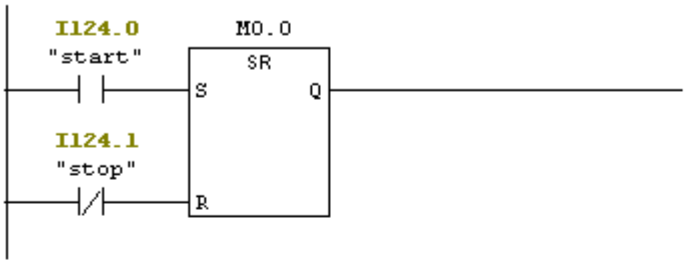




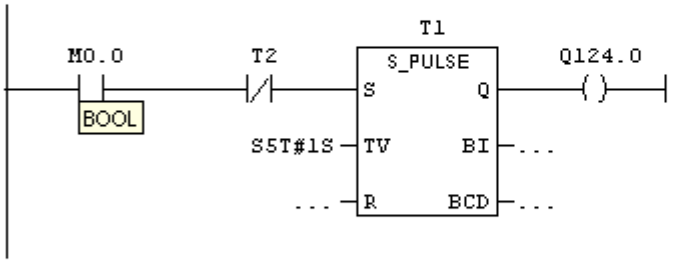
To control the flashing we can add a pushbutton with SR block to enable the timers. The following diagram illustrates that; please note the nature of the different timers when using them in the applications (refer to previous chapter).

OB1 : "Main Program Sweep (Cycle)"

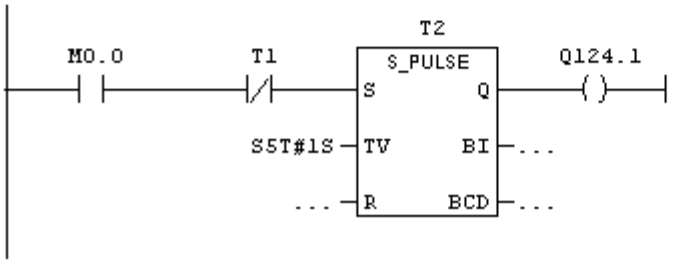
Network 1: start PB to start the flashing and stop PB to stop the flashing



Network 2: FIRST TIME SCAN T2 IS OFF SO T2' Ignate the timer T1 to start

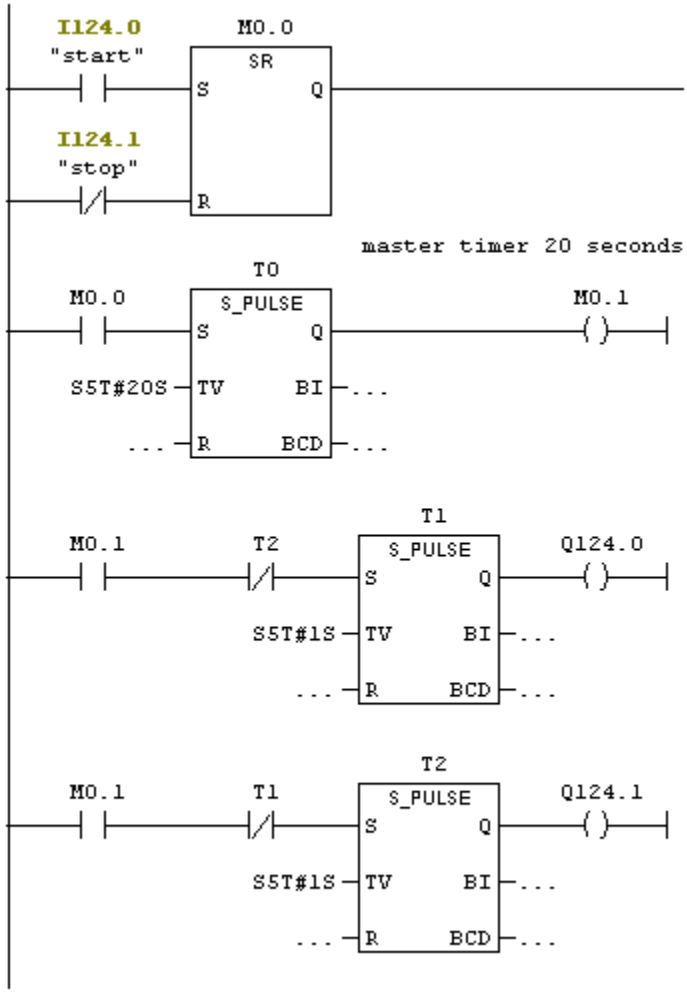


Network 3: Waiting for T1 to finish T1=off T1' will enable T2 to start.



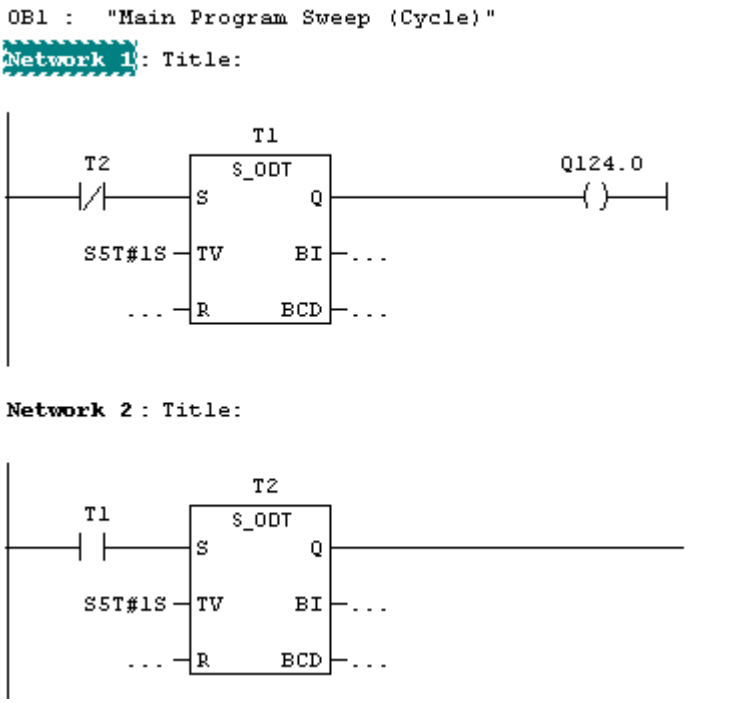
Flashing for a certain amount of time

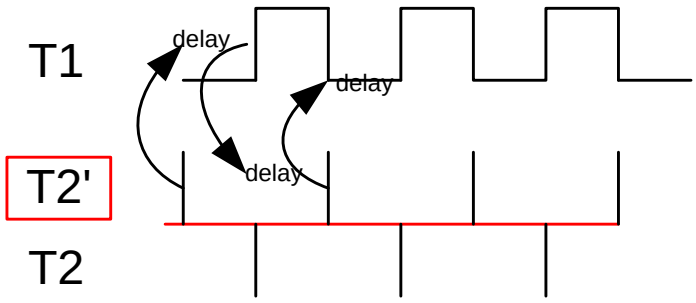
To obtain flashing for a certain amount of time like a flashing for 10 second, an extra timer will be added to the above ladder diagram. This additional timer will act as a Master timer and the other two timer works as a slave.



Flashing using on-delay timers

Similarly, two on delay timer can be used to obtain flashing for 1 second interval. The following ladder diagram illustrates flashing using on-delay timers.





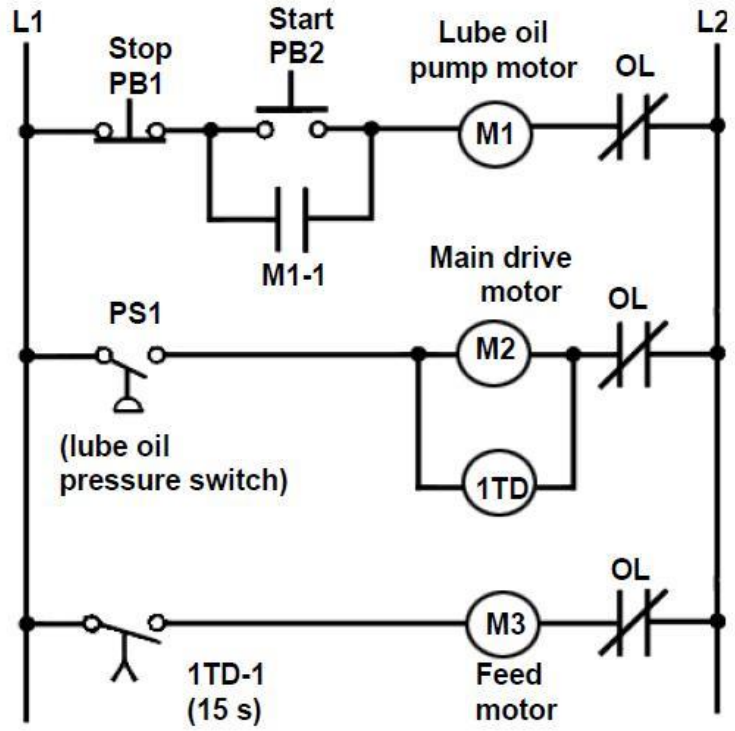
Sequence of operations using timers

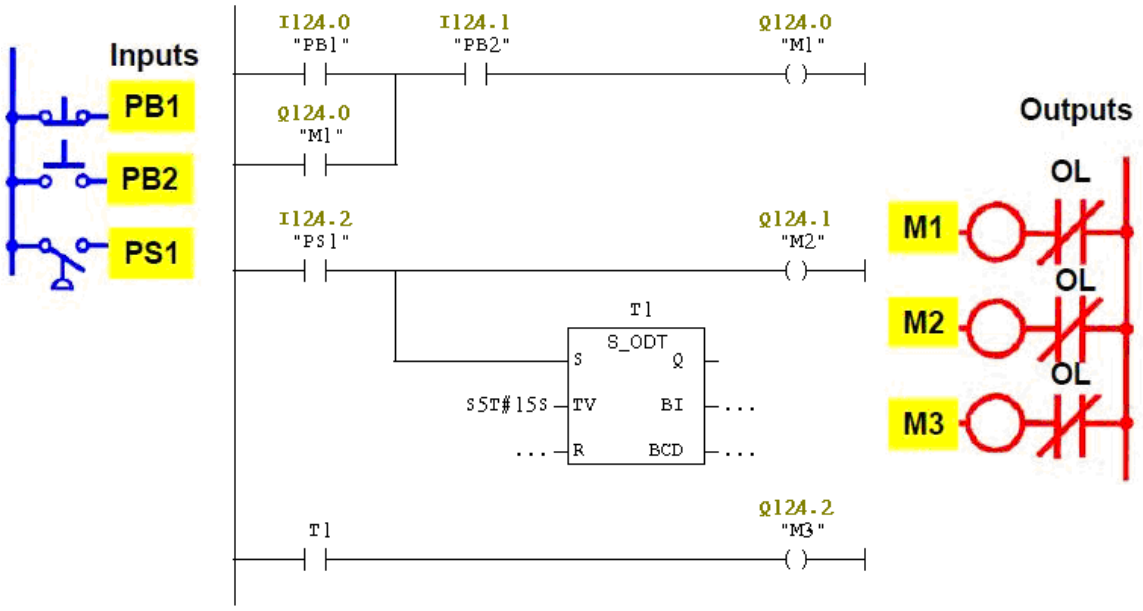
Automatic Sequential Control System:

Timers are often used as part of automatic sequential control systems. The following schematic shows how a series of motors can be started automatically with only one start/stop control station.

According to the relay ladder schematic, lube-oil pump motor started coil M1 is energized when the start pushbutton PB2 is momentarily actuated. As a result, M1-1 control contact closes to seal in M1, and the lube-oil pump motor starts. When the lube-oil pump builds up sufficient oil pressure, the lube-oil pressure switch PS1 closes. This in turn energizes coil M2 to start the main drive motor and energizes coil 1TD to begin the time-delay period. After the preset time-delay period of 15 s, 1TD-1 contact closes to energize coil M3 and start the feed motor. The ladder logic program shows how the circuit could be programmed using a PLC.

Relay Ladder Schematic Diagram




Inputs
PB1
PB2
PS1
I124.0
"PB1"
I124.1
"PB2"
Q124.0
"M1"
Q124.0
"M1"
I124.2
"PS1"
Q124.1
"M2"
T1
S_ODT
S
Q
S5T#15s
TV
BI
R
BCD
T1
Q124.2
"M3"
Outputs
OL
M1
OL
M2
OL
M3

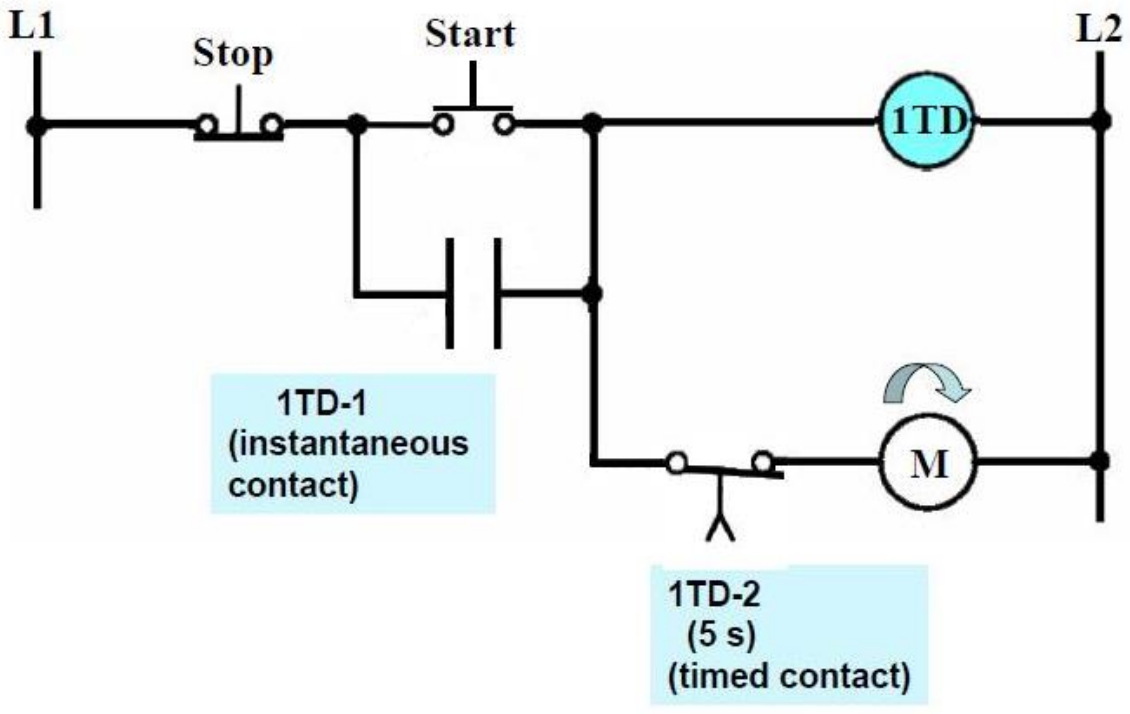
Applications using On-Delay timers

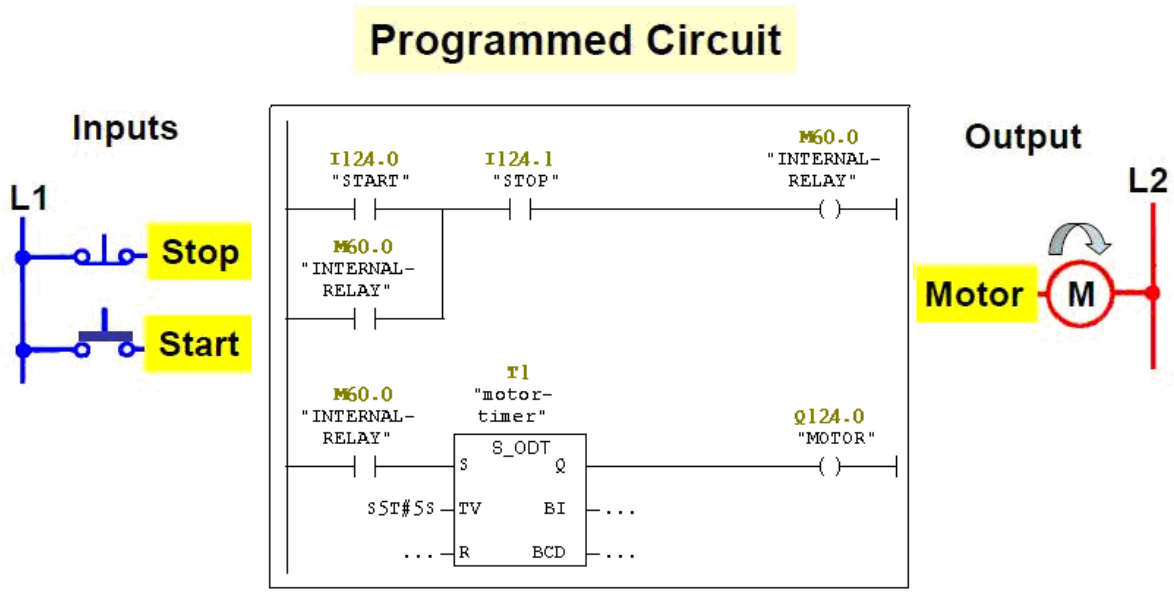
On-Delay timer with instantaneous output:

Timers may or may not have an instantaneous output (also known as the enable bit) signal associated with them. If an instantaneous output signal is required from timer and it is not provided as part of the timer instruction, an equivalent instantaneous contact instruction can be programmed using an internally referenced relay coil.

The following figure shows an application of this technique. According to relay ladder schematic diagram, coil M is to be energized 5 s after the start pushbutton is pressed. Contact 1TD-1 is the instantaneous contact, and contact 1TD2 is the timed contact.

The Ladder logic program shows that a contact instruction referenced to an internal relay is now used to operate the timer. The instantaneous contact is referenced to the internal relay coil, whereas the time-delay contact is referenced to the timer output coil.



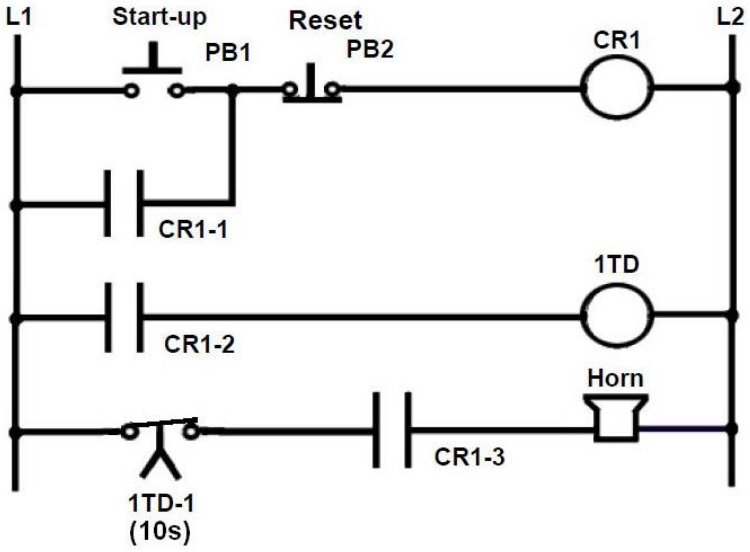


S7_timer-applic-01

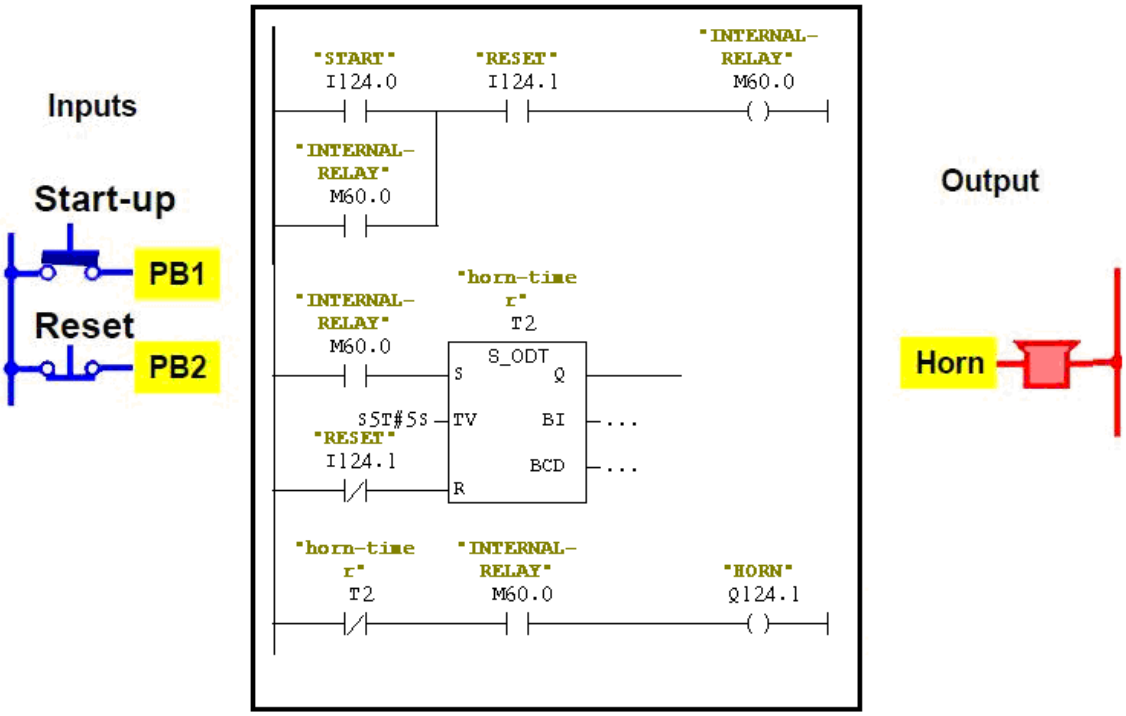
Start-up Warning Signal Circuit:

The schematic below shows the application of an on-delay timer that uses an NCTO contact. This circuit is used as a warning signal when moving equipment, such as a conveyor motor, is about to be started. According to the relay ladder schematic diagram, coil CR1 is energized when the start pushbutton PB1 is momentarily actuated. As a result, contact CR1-1 closes to seal in CR1, contact CR1-2 closes to energize timer coil 1TD, and contact CR1-3 closes to sound the horn. After a 10-s time delay period, timer contact 1TD-1 opens to automatically switch the horn off. The ladder logic diagram shows how the circuit could be programmed using a PLC.

Relay Ladder Schematic Diagram



Programmed Circuit

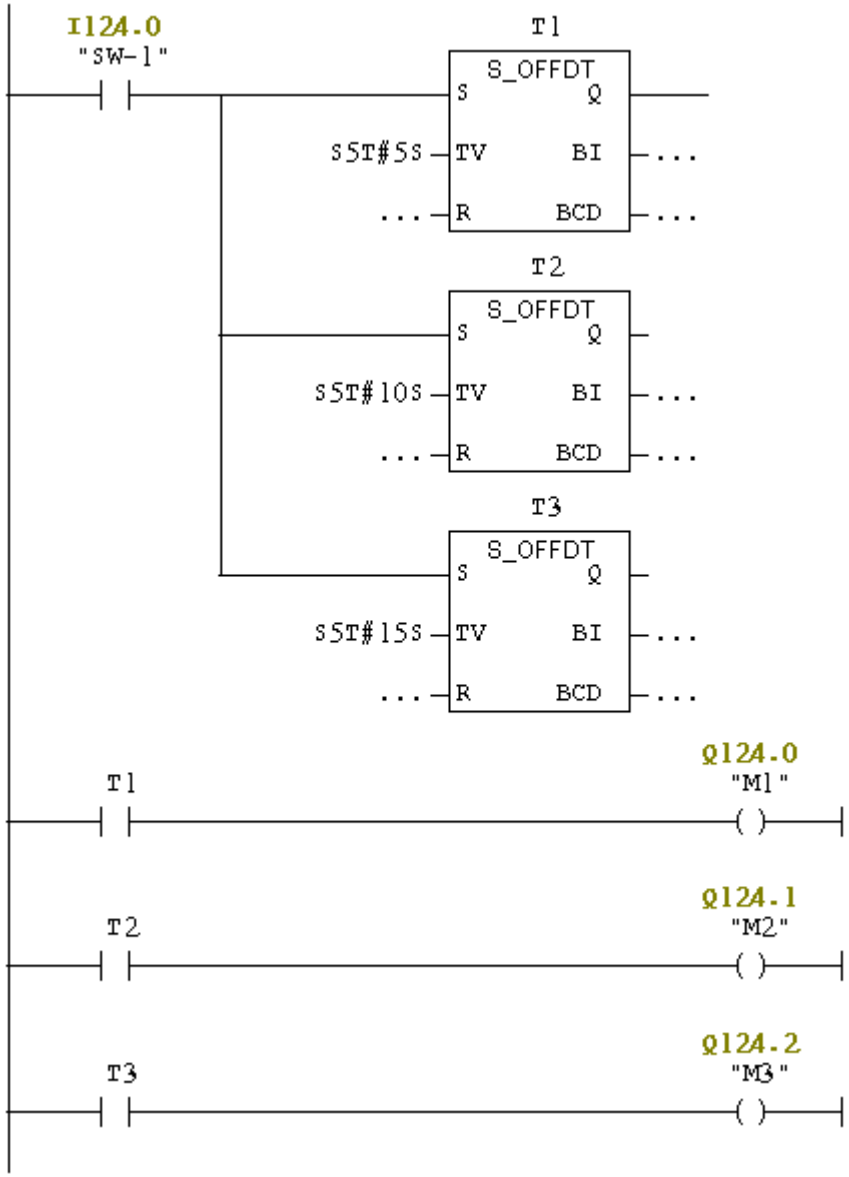


S7_timer-applic-02a

Applications using OFF-Delay timers

Off-Delay Timer Used to Switch Motors Off at a 5 s intervals:

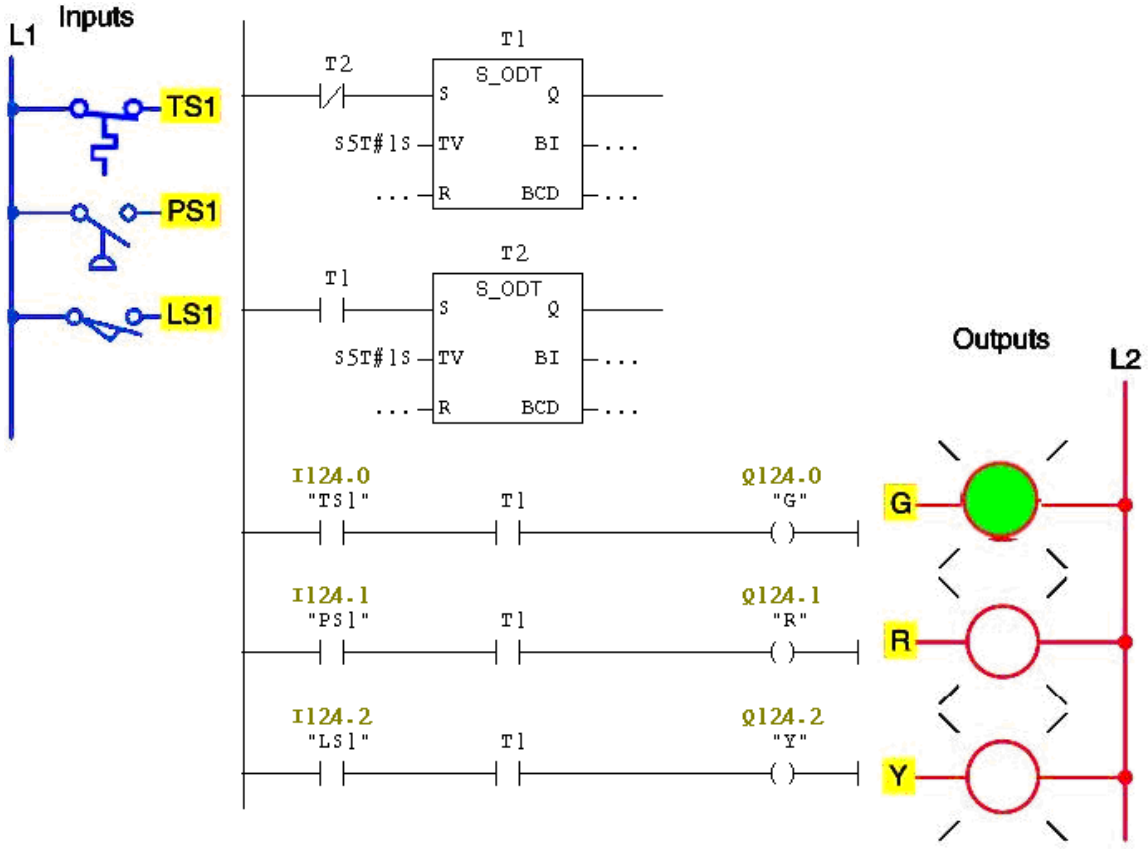
Closing the sw-1 immediately turns on motors M1, M2, and M3. When the switch is opened, motors M1, M2, and M3 turns off at 5-s intervals.



Annunciator flasher program

Two timers can be interconnected to form an oscillator circuit. The oscillator logic is basically a timing circuit programmed to generate periodic output pulses of any duration. They can be used as part of an annunciator system to indicate an alarm condition.

The oscillator circuit output is programmed in series with the alarm condition. If the alarm condition is true, the appropriate output indicating light will flash.



Application “Tank filling system”

Suppose that you have a **Tank Filling System**. Design a Ladder Program for the system with the following conditions:

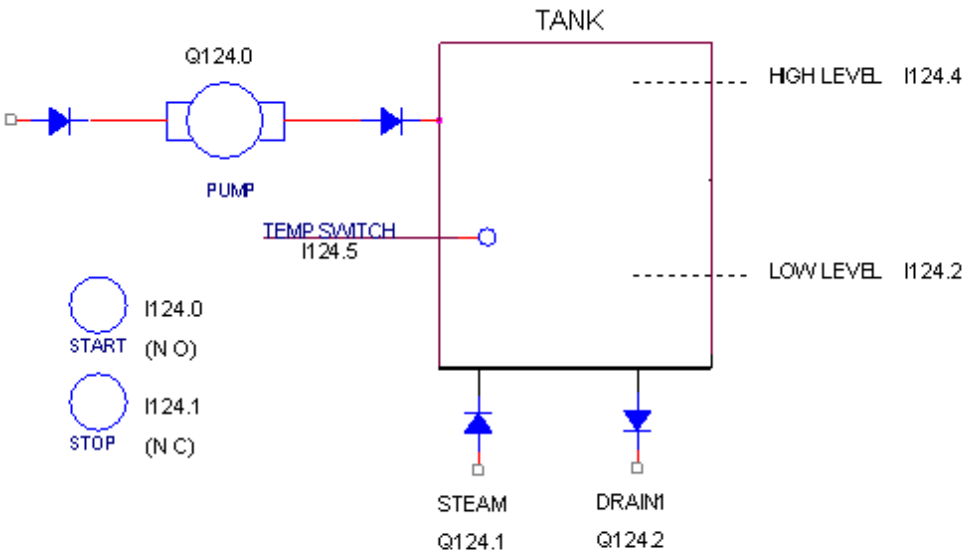
i) When the START push button is pressed, the Pump starts to fill the tank until the HIGH level is reached. Then Pump stops.

ii) Then, STEAM valve opens, raising the temperature, until the temperature switch I124.5 is activated.

iii) Ten minutes after the desired temperate is reached the Steam shuts off and the Drain1 valves open.

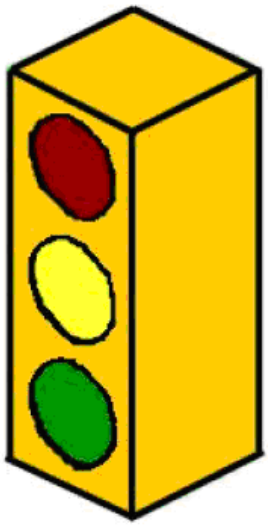
iv) After the Tank is Empty and water level reaches LOW LEVEL sensor, the Drain1 valve also shuts off.

v) The process can be stopped whenever by pressing the STOP push button.

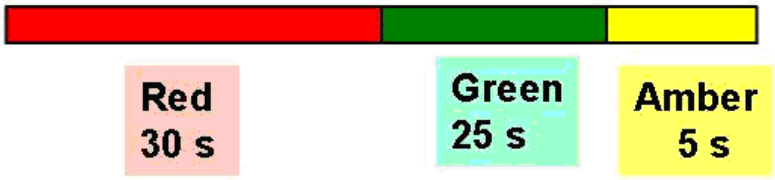


Solution

Application “Traffic light of one street”



Control of Traffic in One Direction
Sequence of Operation



Homework: Design the Ladder Program to perform this function

SOLUTION

Application “Traffic light of two streets”

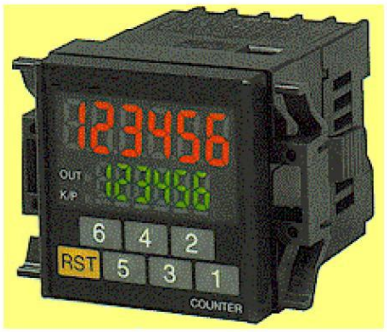
PART SEVEN

PROGRAMMING COUNTERS - I

PROGRAMMING COUNTERS I

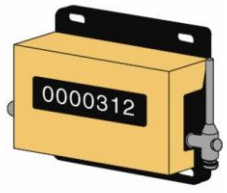
Counters:

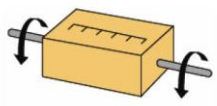
Common applications of counters include keeping track of the number of items moving past a given point, and determining the number of times a given action occurs.



A preset counter can control an external circuit when its counted total matches the user-entered preset limits.

Mechanical Counters:



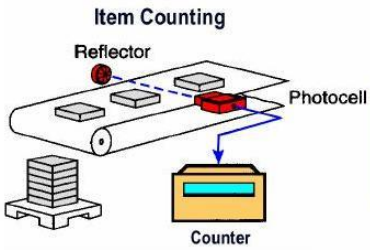


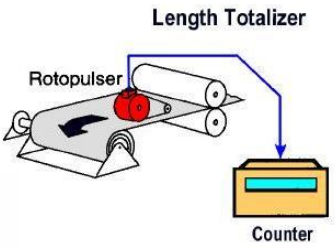
Programmed counters can serve the same functions as mechanical counters.

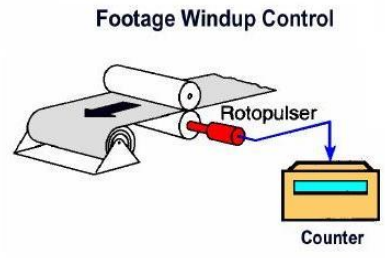
Every time the actuating lever is moved over the counter adds one number, while the actuating lever returns automatically to its original position. Resetting to zero is done with a pushbutton located on the side of the unit.

Electronic Counters:

Electronic counters can count up, count down, or be combined to count up and down. They are dependent on external sources, such as parts traveling past a sensor or actuating a limit switch for counting.



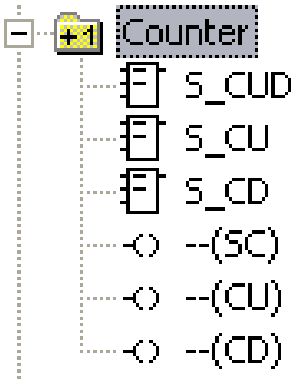




Step 7 Counters

COUNTER OPERATIONS

In control engineering, counter functions are needed for collecting the number of items or pulses and for the evaluation of times and distances. In the SIMATIC S7, counters are already integrated in the CPU. These counters possess their own reserved storage area. The range of the count value lies between 0 and 999. The following functions can be programmed with a counter:



UP/Down Counter
UP Counter
Down Counter
Set Counter Value
Up Counter coil
Down Counter coil

RELEASE COUNTER (FR) ONLY IN STL

A positive edge change (of "0" to "1") in the logical operation of the operation release (FR) releases a counter.
A counter release is not needed for setting a counter or for normal counting operations. However, if one wants to set a counter without a rising edge before the appropriate counting operation (CU, CD or S), then this can take place with a release. This is however possible only if the RLO bit before the appropriate operation (CU, CD or S) has a signal status "1".

The operation release (FR) only exists in the programming language STL.

COUNTER UP (CU)

The value of the addressed counter is increased by 1. The function becomes effective only with a positive edge change of the logical operation programmed before CU. If the count value achieves the upper limit of 999, it is no longer increased. (*a carry is not generated!*)

COUNTER DOWN (CD)

The value of the addressed counter is reduced by 1. The function becomes effective only with a positive edge change of the logical operation programmed before CD. If the count value achieves the lower limit 0, it is no longer reduced. *(Only positive counter values!)*

SET COUNTER (S)

In order to set a counter, you must insert three operations into its STL program:

- Query a signal status
- Load a count value
- Set a counter with the loaded count of the function.
  This function is only edited by a positive edge change of the query.

COUNTER VALUE (CV)

If a counter is set, then the contents of ACCU 1 are used as the count. There is a possibility to code the count value either as binary or BCD code. The following operands are possible:

- *Input word* *IW ..*
- *Output word* *QW ..*
- *Memory bit word* *MW ..*
- *Data word* *DBW/DIW ..*
- *Local data word* *LW ..*
- *Constant* *C#5, 2#...etc.*

e.g.:

```
A    I 2.3
L    C#5
S    C1
```

RESET COUNTER (R)

The counter is set to zero (to reset) with RLO 1. The counter remains unchanged with RLO 0. Resetting a counter works statically. During a satisfied resetting condition, a counter can be neither set nor counted.

LOAD COUNTER (L/LC)

A count is stored in a counter word binary code. The value in the counter can be loaded as a dual number (DU) or as BCD number (DE) into the ACCU and be transferred from there into other operand ranges. With STL programming, you have the choice between *L C1* for the query of the dual number and *LC C1* for the query of the BCD number.

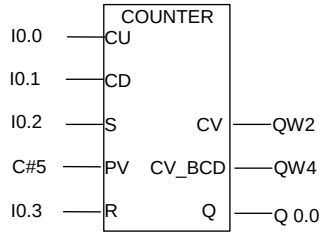
QUERY SIGNAL STATE OF COUNTER (Q)

A counter can be tested for its signal status. The meaning of the signal states are:

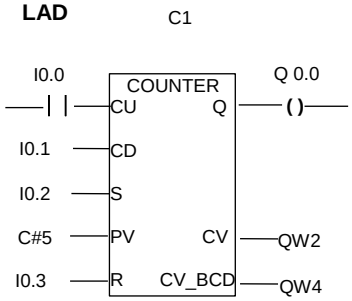
Signal state 0 = Counter stays on the value 0;
Signal state 1 = Counter runs, i.e. it is count ready.

Signal statuses can be queried with A C1, AN C1, ON C1, etc.... and can be used for further logical operations.

FBD



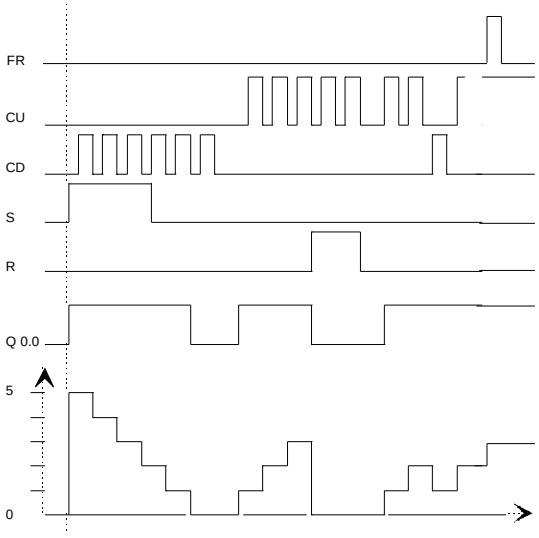
LAD

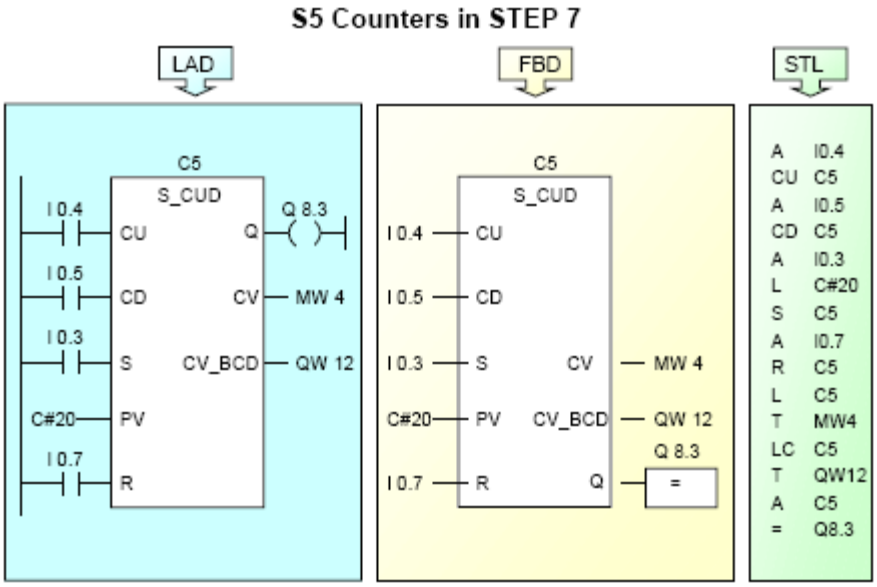


STL

```
A    I 0.7     Release (only in STL)
FR   C1
A    I 0.0
CU   C1        Count up
A    I 0.1
CD   C1        Count down
A    I 0.2
L    C#5       Load counter with default value
S    C1        Set counter with default value
A    I 0.3
R    C1        Rest counter C1
L    C1        Load counter C1 DUAL-coded
T    QW2
LC   C1        Load counter C1 BCD-coded
T    QW4
A    C1        Query of the counter C1
=    Q 0.0
```

Signal state chart:





Counter Value
A 16-bit word is reserved for each counter in the system data memory. This word is used for storing the counter's value (0 to 999) in binary code.

Count Up
When the RLO at the "CU" input changes from "0" to "1", the counter's current value is incremented by 1 (upper limit = 999).

Count Down
When the RLO at the "CD" input changes from "0" to "1", the counter's current value is decremented by 1 (lower limit = 0).

Set Counter
When the RLO at the "S" input changes from "0" to "1", the counter is set to the value at the "PV" input.

Reset Counter
When the RLO at the Reset changes from "0" to "1", the counter's value is set to zero. If the reset condition is fulfilled (stays "high"), the counter cannot be set and counting in either direction is not possible.

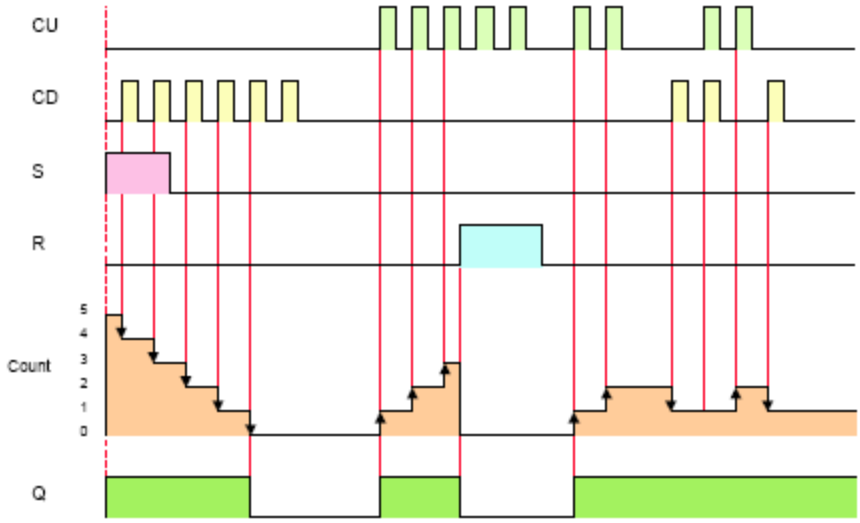
PV The preset value (0 to 999) is specified in BCD format at the "PV" input as:
- As a constant (C#...)
- A BCD format through a data interface.

CV / CV_BCD
The counter value can be loaded as a binary number (CV) or BCD number (CV_BCD) into accumulator 1 and then transferred to other addresses.

Q The signal state of the counter can be checked at output "Q":
- Count = 0 -> output Q = 0
- Count >< 0 -> output Q = 1

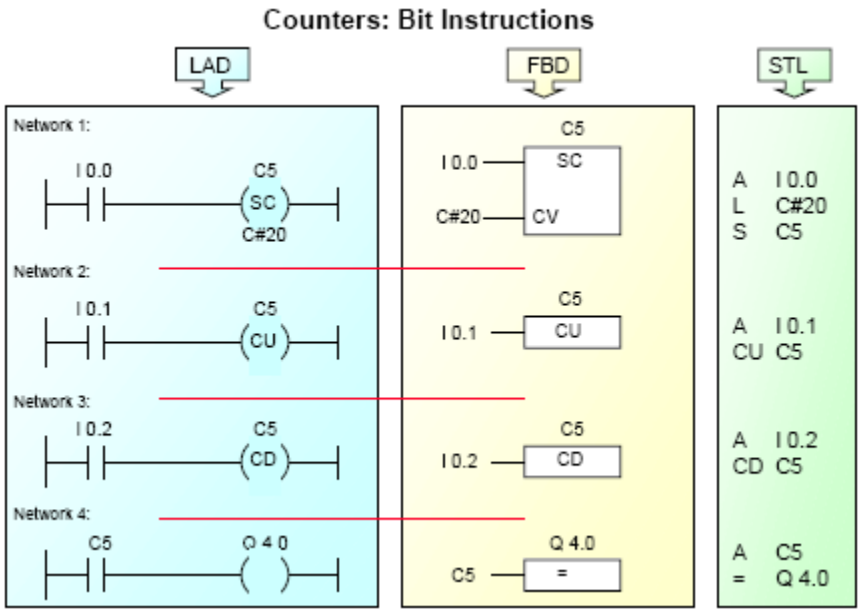
Types of Counters
- S_CU = Up counter (counts up only)
- S_CD = Down counter (counts down only)
- S_CUD = Up/Down counter.

Counters: Function Diagram



Notes

When the counter reaches its maximum value (999), the next count up signal does not affect the counter. Likewise, when the counter reaches its minimum value (0), the next count down signal does not affect the counter. The counters do not count above 999 of lower than zero. If an up count and a down count signal occur at the same time, the count remains the same.



Bit Instructions

All counter functions can also operate with simple bit instructions. The similarities and differences between this method and the counter functions discussed so far are as follows:

• Similarities:

- Setting conditions at the "SC" input
- Specification of the counter value
- RLO change at the "CU" input
- RLO change at the "CD" input

• Differences:

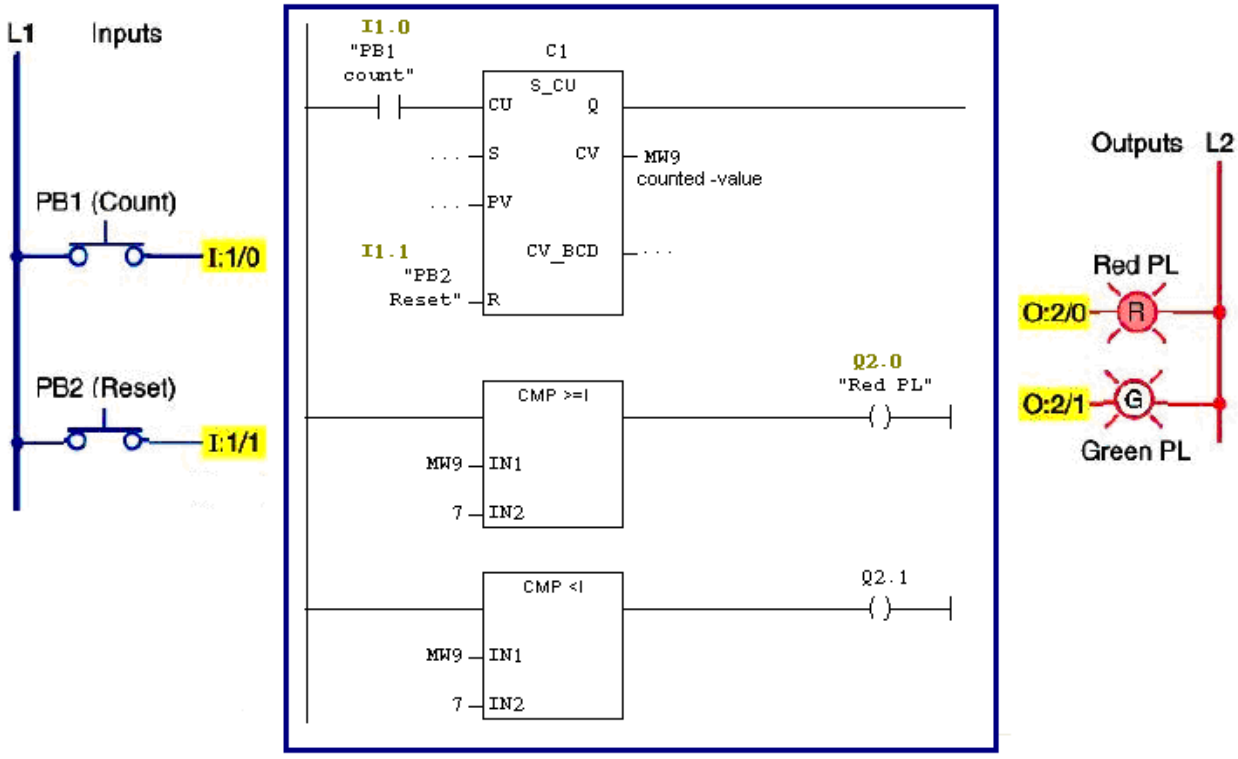
- It is not possible to check the current counter value since there are no Binary (CV) or BCD (CV_BCD) outputs.
- There is no binary output Q in the graphical representation.

Note

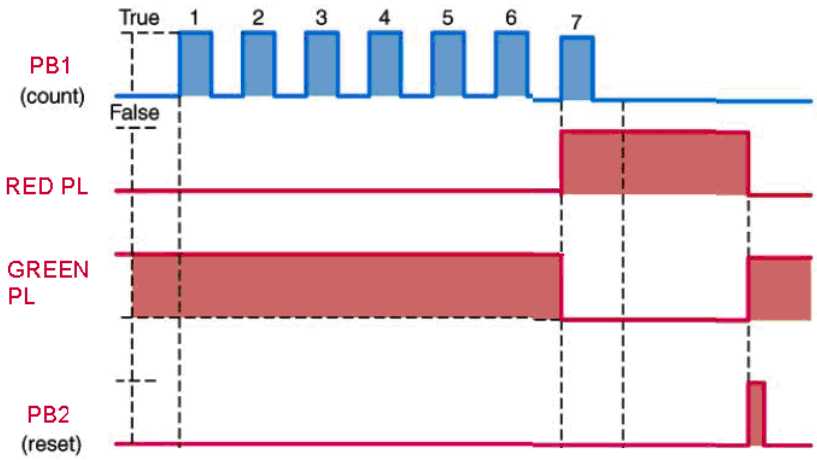
IEC-compliant counters can also be implemented in STEP 7. The use of system function blocks for implementing IEC counters is dealt with in an advanced programming course.

Simple Up-Counter Program:

This simple up counter is designed to turn the red pilot light on and the green pilot light off after an accumulated count of 7. Operating pushbutton PB1 provides the off-to-on transition pulses that are counted by the counter. **[Unlike the other plc's Siemens counter requires the use of a comparator to check for the preset value]**. In this case the counter preset value is 7. PB2 pushbutton is used to reset the counter value.



S7_counter-applic-01


True
1
2
3
4
5
6
7
PB1
(count)
False
RED PL
GREEN
PL
PB2
(reset)

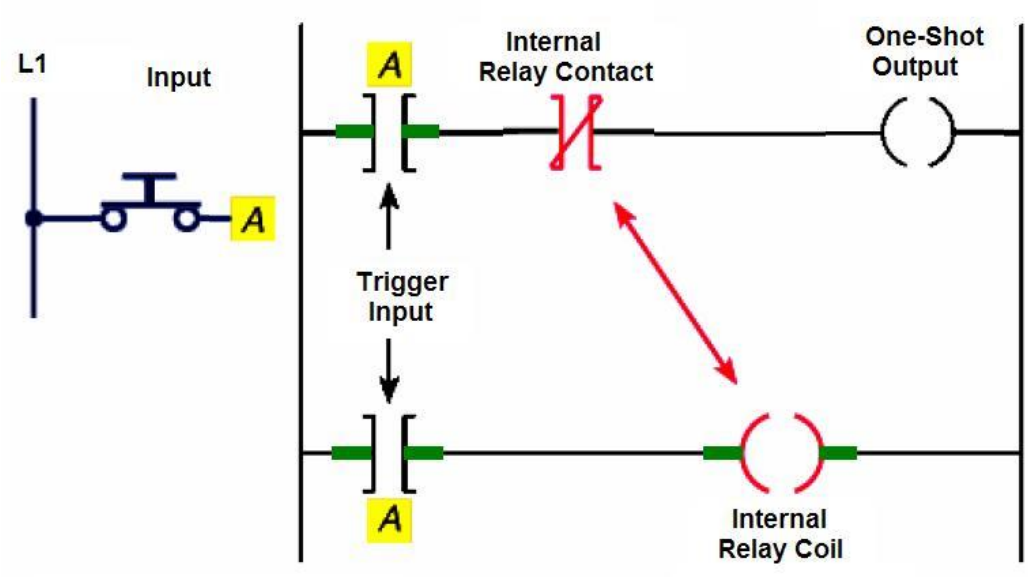
PART EIGHT

PROGRAMMING COUNTERS - II

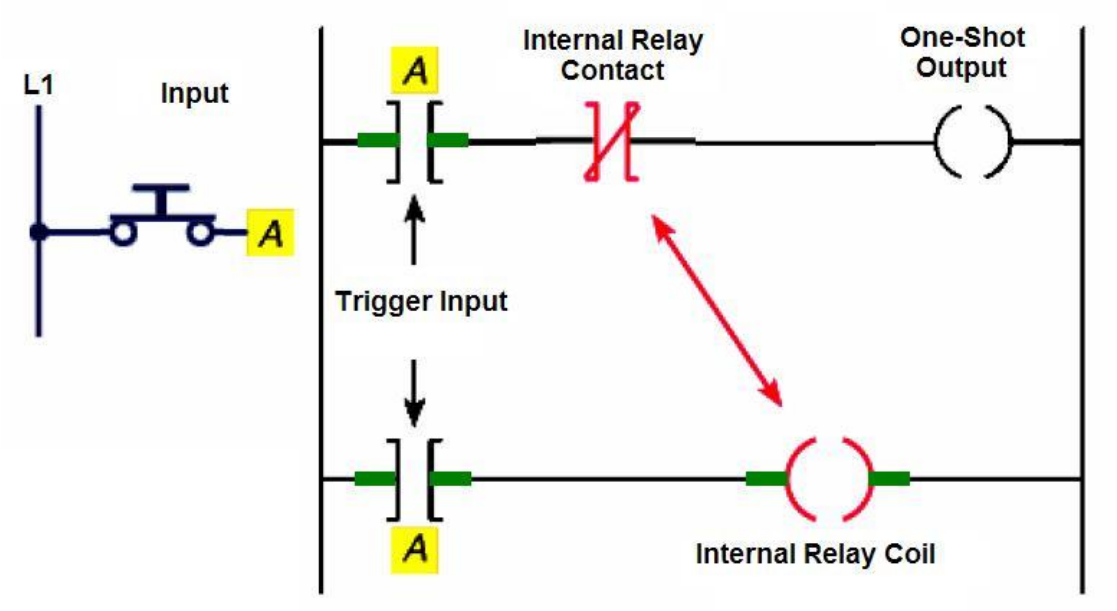
PROGRAMMING COUNTERS II

One-Shot, or Transitional, Contact Program:

The transitional or one shot contact program can be used to automatically clear or reset a counter. The program is designed to generate an output pulse that, when triggered, goes on for the duration of one program scan and then goes off.

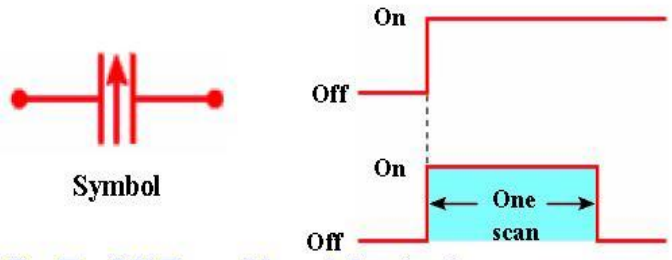


The transitional or one shot contact program can be used to automatically clear or reset a counter. The program is designed to generate an output pulse that, when triggered, goes on for the duration of one program scan and then goes off.



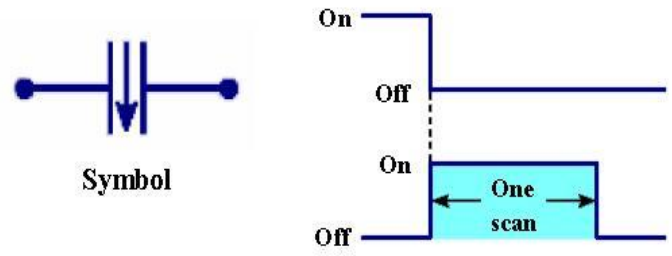
Types of Transitional Contacts:

Off-To-On Transitional Contact



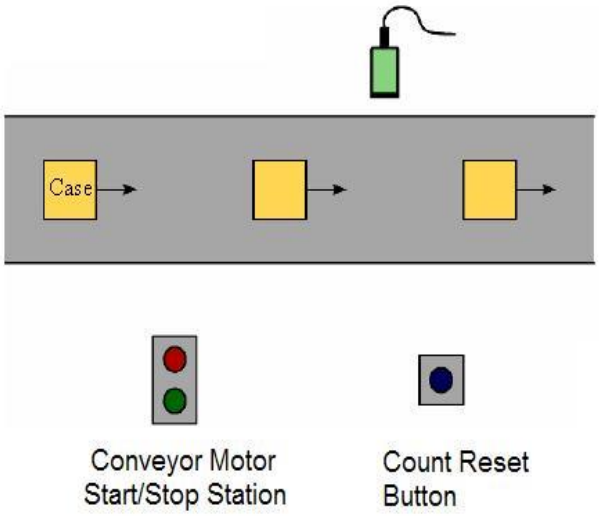
Is programmed to provide a one-shot pulse when the referenced trigger signal makes a positive (off-to-on) transition.

On-To-Off Transitional Contact



Is programmed to provide a one-shot pulse when the referenced trigger signal makes a negative (on-to-off) transition

Conveyor Motor Circuit That Uses a Programmed One-Shot Reset Circuit:



Sequential Task:

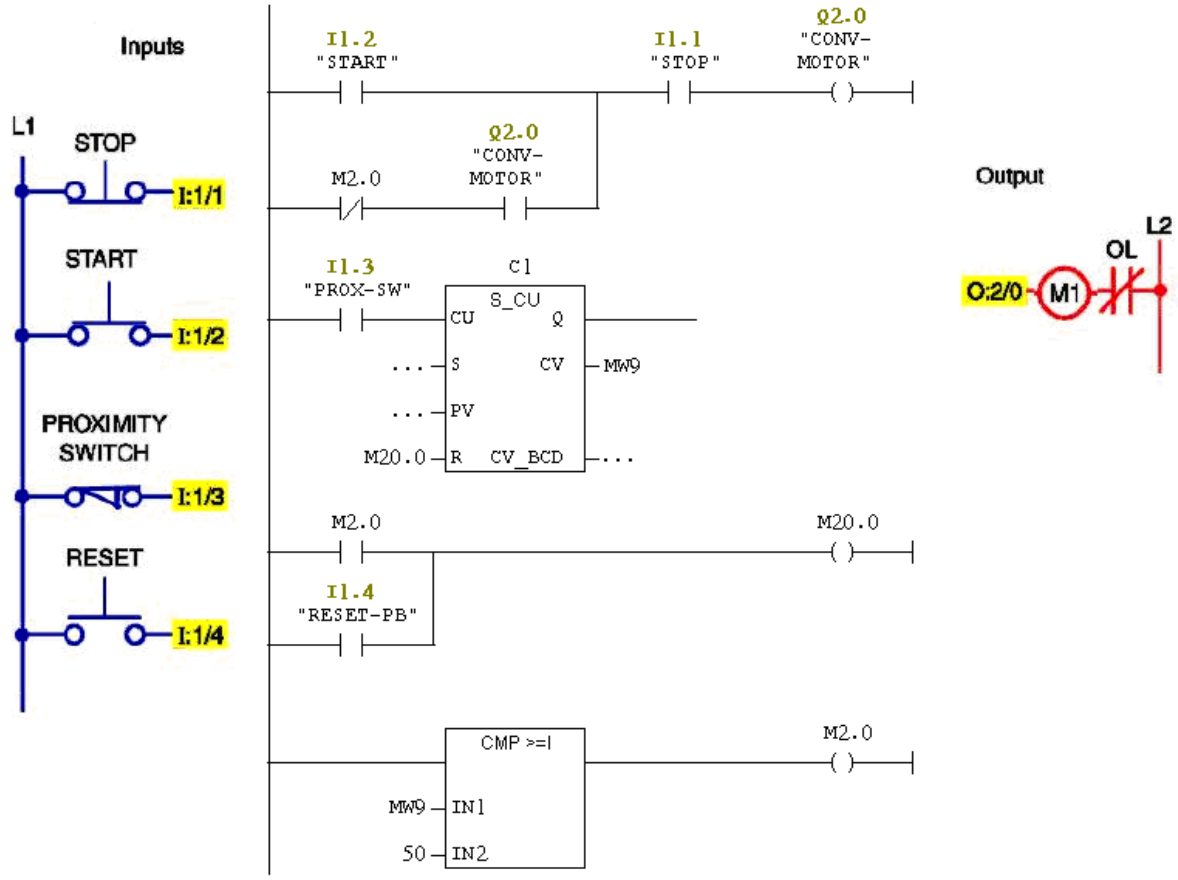
The start button is pressed to start the conveyor motor.

Cases move pass the proximity switch and increment the accumulated value.

After a count of 50, the conveyor motor stops automatically and the counters accumulated value is reset to zero.

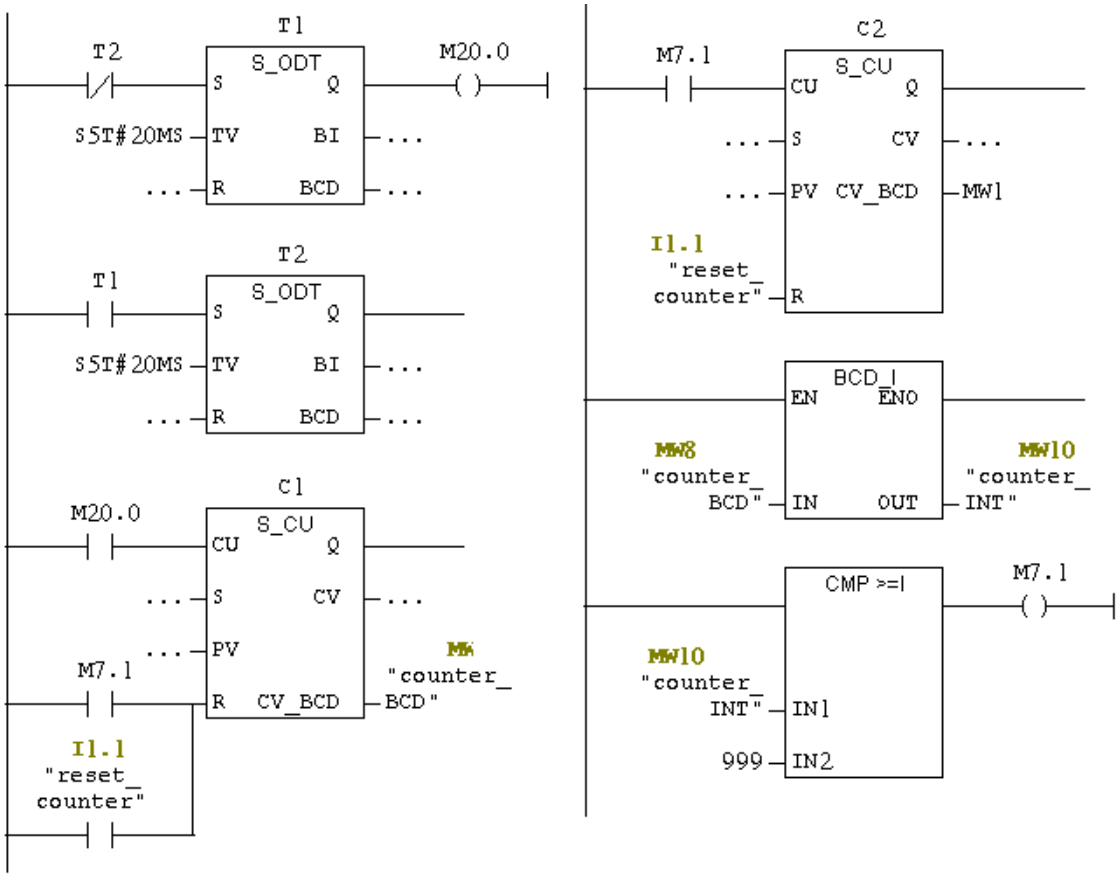
The conveyor motor can be stopped or started manually at anytime without loss of the accumulated count.

Conveyor Motor Circuit That Uses a Programmed One-Shot Reset Circuit:



Counter Cascading:

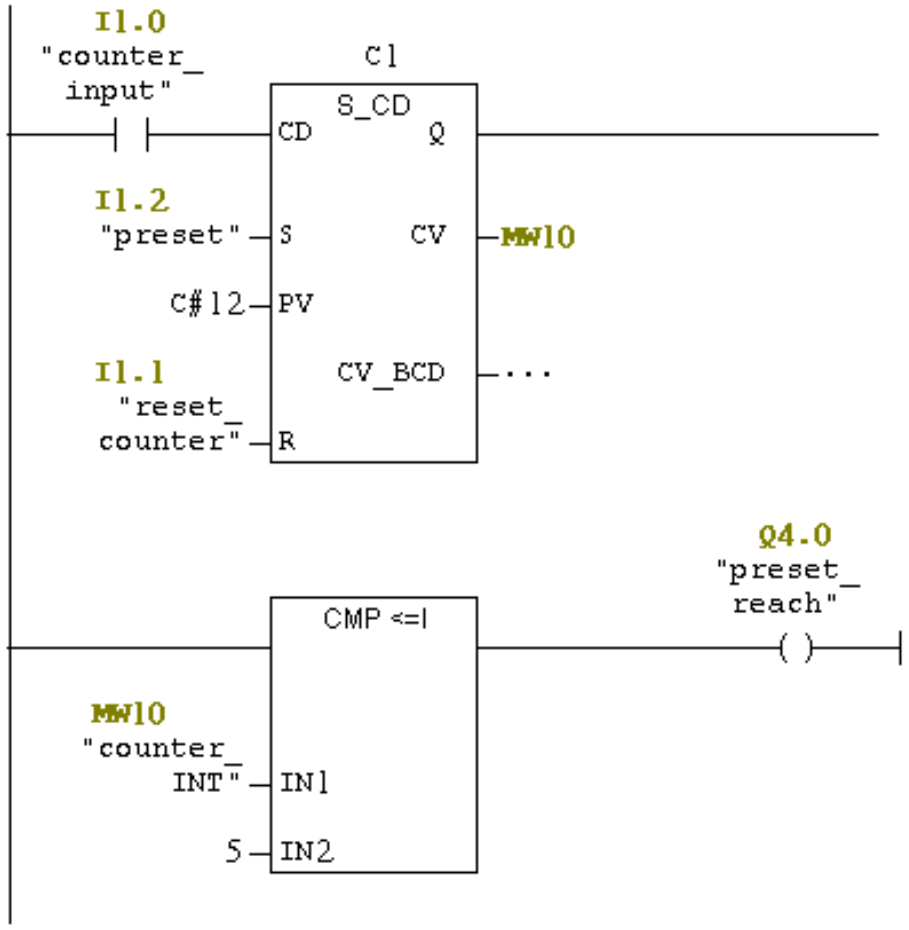
Note the use of CV_BCD



Down-Counter:

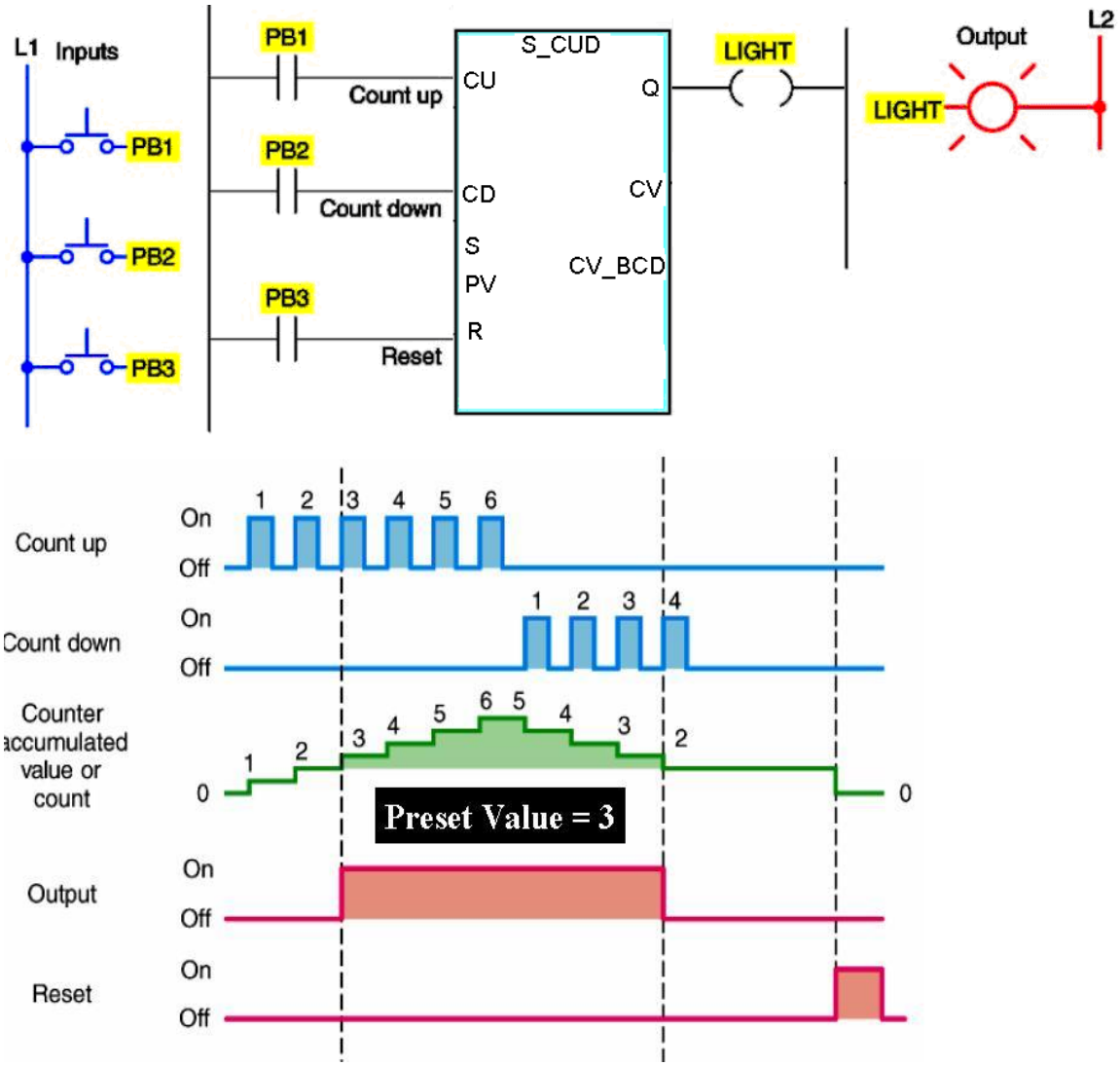
The down-counter output instruction will count down or decrement by 1 each time the counted event occurs. Each time the down-count event occurs, the accumulated value is decremented. Normally the down-counter is used in conjunction with the up counter to form an up/down counter.

Simple down counter in simatic s7



S7_down_counter

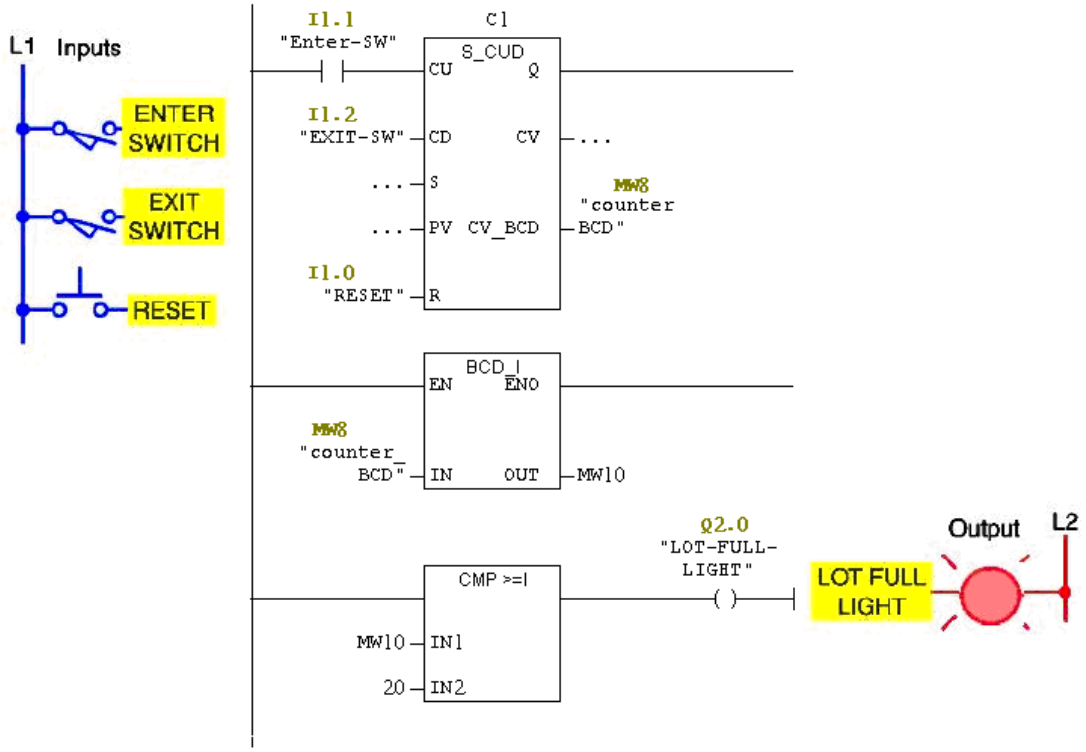
Up/Down Counter Timing Diagram:



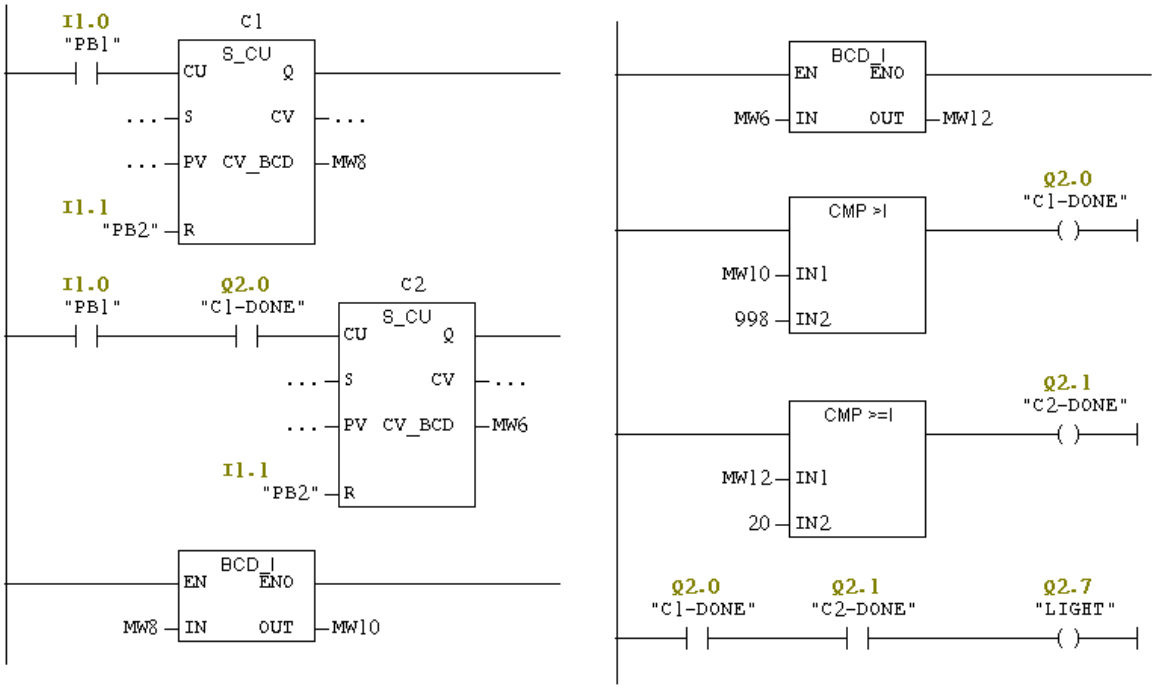
Parking Garage Counter Program:

- As a car enters, it triggers the up-counter output instruction and increments the accumulated count by 1.
- As a car leaves, it triggers the down-counter output instruction and decrements the accumulated count by 1.
- Since both the up- and down-counters have the same address, the accumulated value will be the same in both.
- Whenever the accumulated value equals the preset value, the counter output is energized to light up the Lot Full sign.

Parking Garage Counter Program:



Counting Beyond the Maximum Count

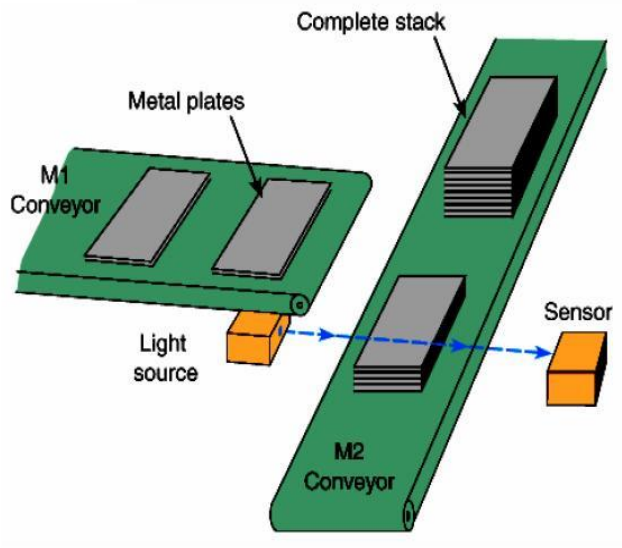


Counter Speed:

The maximum speed of transitions you can count is determined by your program's scan time. Any counter input signal must be fixed for one scan time to be counted reliably.

If the input changes faster than one scan period, the count value will become unreliable because' counts will be missed. When this is the case you need to use a high-speed counter.

Combining Counter and Timer Functions:

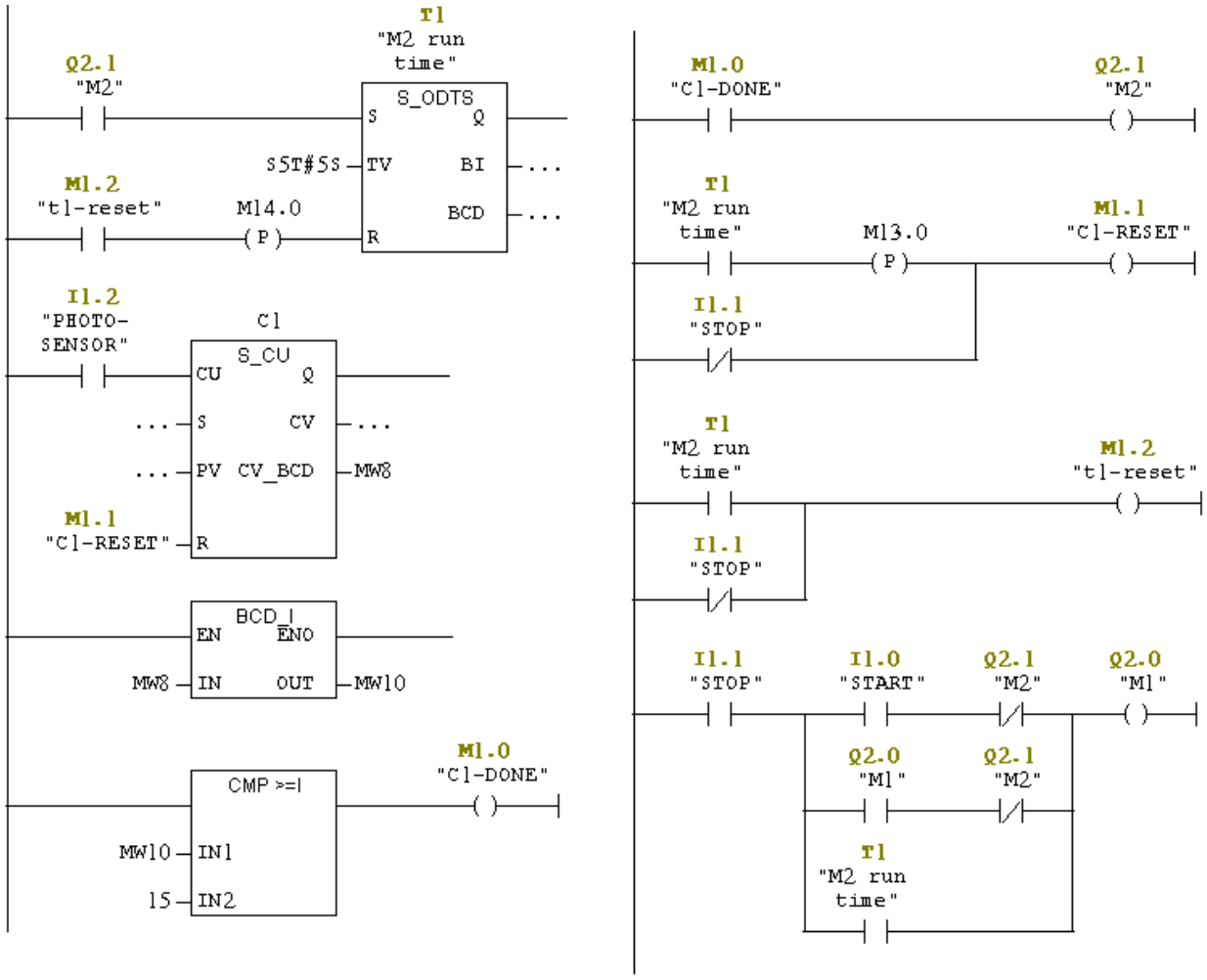


When the start button is pressed, conveyor M1 begins running.

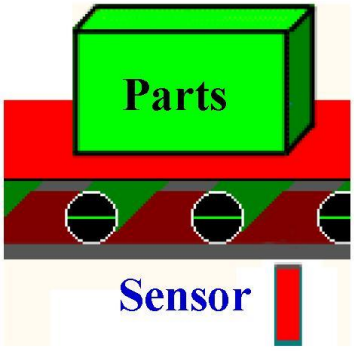
After 15 plates have been stacked, conveyor M1 stops and conveyor M2 begins running.

After conveyor M2 has been operated for 5 s, it stops and the sequence is repeated automatically.
The done bit of the timer resets the timer and counter, and provides a momentary pulse to Automatic Stacking Process automatically restart conveyor.

Automatic Stacking Program:



Product Flow Rate Program:



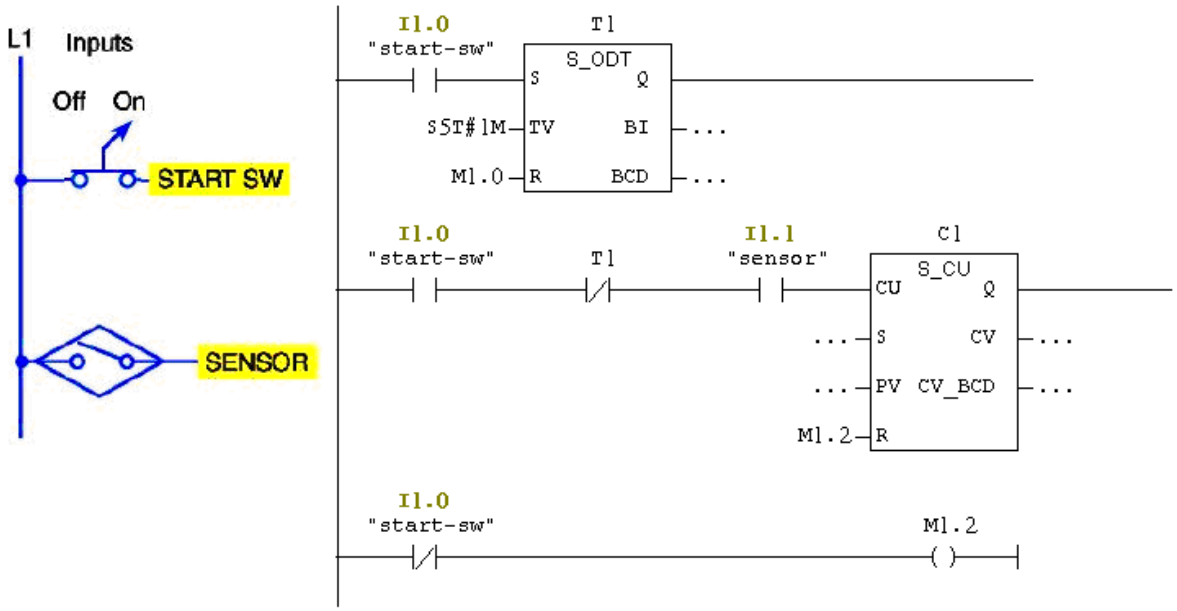
This program is designed to indicate how many parts per minute pass a given process point.

When the start switch is closed, both the counter and timer are enabled.

The counter is pulsed for each part passing the sensor.

The counting begins and the timer starts timing through its 1-min time interval. At the end of 1 min, the timer done bit causes the counter rung to go false. Sensor pulses continue but do not affect the PLC counter. The number of parts for the past minutes is represented by the accumulated value of the counter.

Product Flow Rate Program:



Example:

- Write a program to operate a light according to the following sequence:

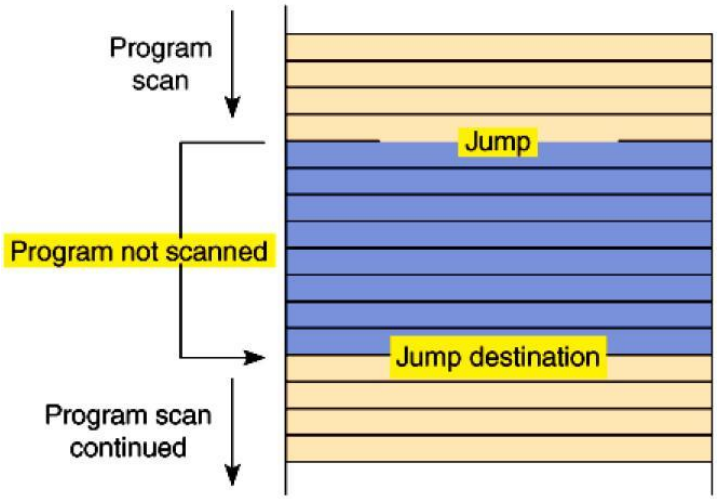
  - A momentary switch is pressed to start the sequence -The light is switched on and remains on for 2 s.
  - The light is then switched off and remains off for 2 s -A counter is incremented by 1 after this sequence.
  - The sequence then repeats for a total of 4 counts.
  - After the fourth count, the sequence will stop and the counter will be reset to zero.

PART NINE

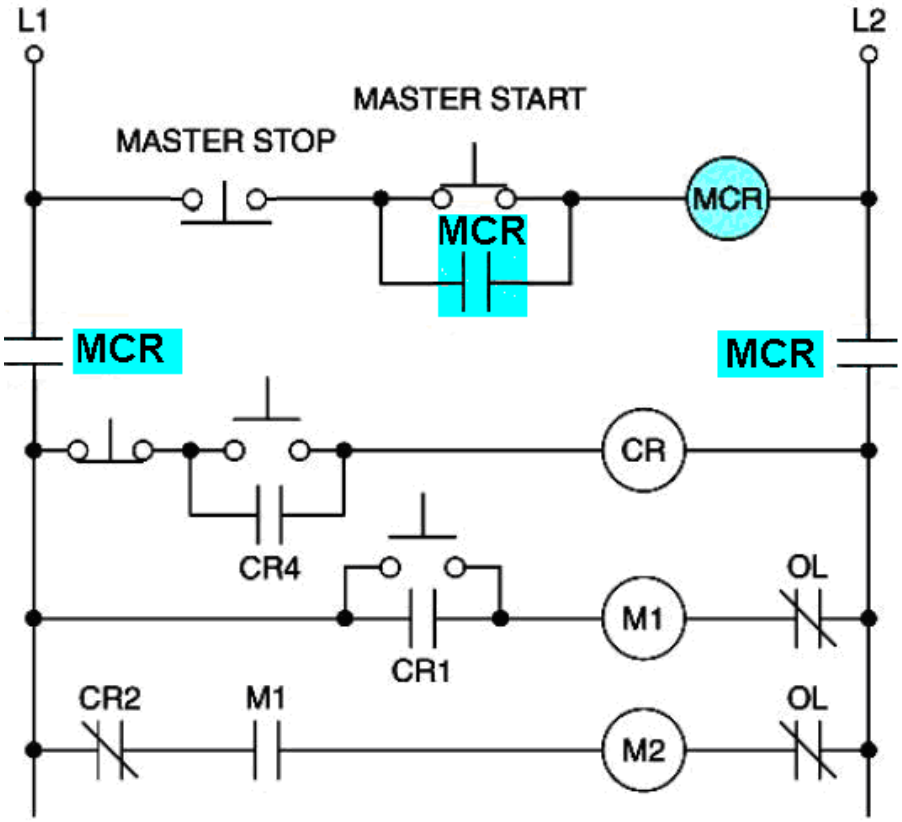
PROGRAM CONTROL INSTRUCTION

PROGRAM CONTROL INSTRUCTIONS

- Program control instructions are used to alter the program scan from its normal sequence.
- Sometimes referred to as override scan instructions, they provide a means of executing sections of the control logic if certain conditions are met.
- They allow for greater program flexibility and greater efficiency in the program scan.



Hardwired Master Control Relay Circuit:

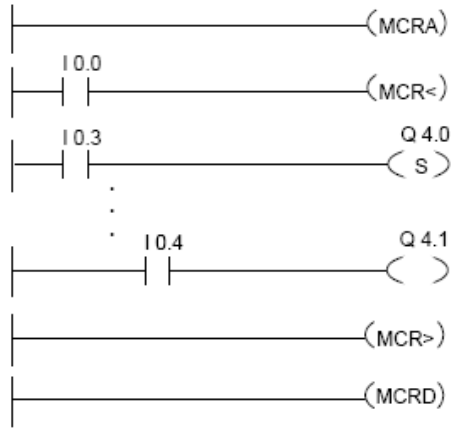


MCR Instruction:

- The master control reset (MCR) instruction can be programmed to control an entire circuit or to control only selected rungs of a circuit.
- When the MCR instruction is false, or de-energized, all *non-retentive* (***non-latched***) rungs below the MCR will be de-energized even if the programmed logic for each rung is true.
- All retentive rungs will remain in their *last state.*
- The MCR instruction establishes a zone in the user program in which all non-retentive outputs can be turned off simultaneously.
- Therefore, retentive instructions should not normally be placed within an MCR zone because the MCR zone maintains retentive instructions in the last active state when the instruction goes false.

In SIMATIC S7, the following instructions have to be in the sequence below. You may have more than one MCR zone.

(MCRA) Master Control Relay Activate
(MCR<) Master Control Relay On
(MCR>) Master Control Relay Off
(MCRD) Master Control Relay Deactivate

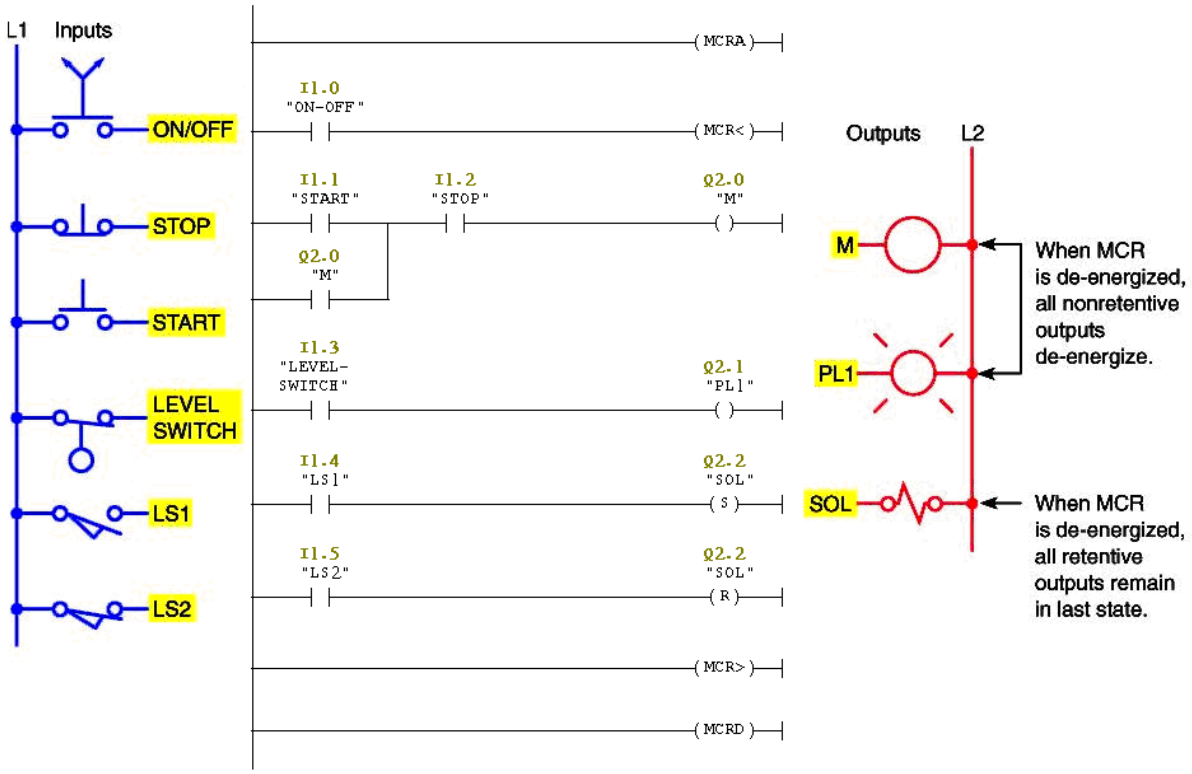


The instruction —(MCRA) activates the function MCR up to the next MCRD. The instructions between —(MCR<) and —(MCR>) are processed dependent on the MA bit (here I 0.0):

- If the signal state of input I 0.0 = 1, the following conditions can exist:
  - Output Q 4.0 is set to 1 if the signal state of input I 0.3 is 1.
  - Output Q 4.0 remains unchanged if the signal state of input I 0.3 is 0.
  - The signal state of input I 0.4 is assigned to output Q 4.1.
- If the signal state of input I 0.0 = 0, the following conditions exist:
  - Output Q 4.0 remains unchanged regardless of the signal state of input I 0.3.
  - Output Q 4.1 is 0 regardless of the signal state of input I 0.4.

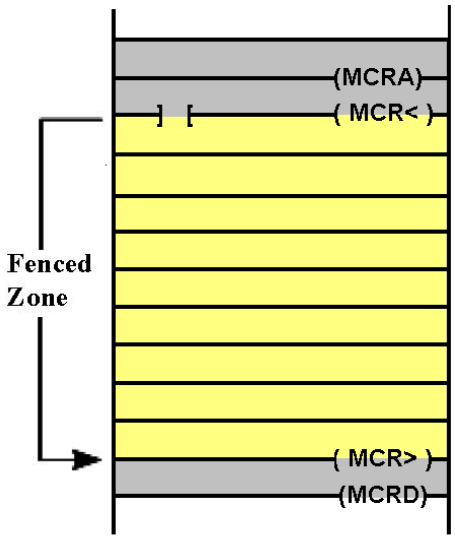
Status Word Bits									
	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	$\overline{FC}$
Write	–	–	–	–	–	–	–	–	–

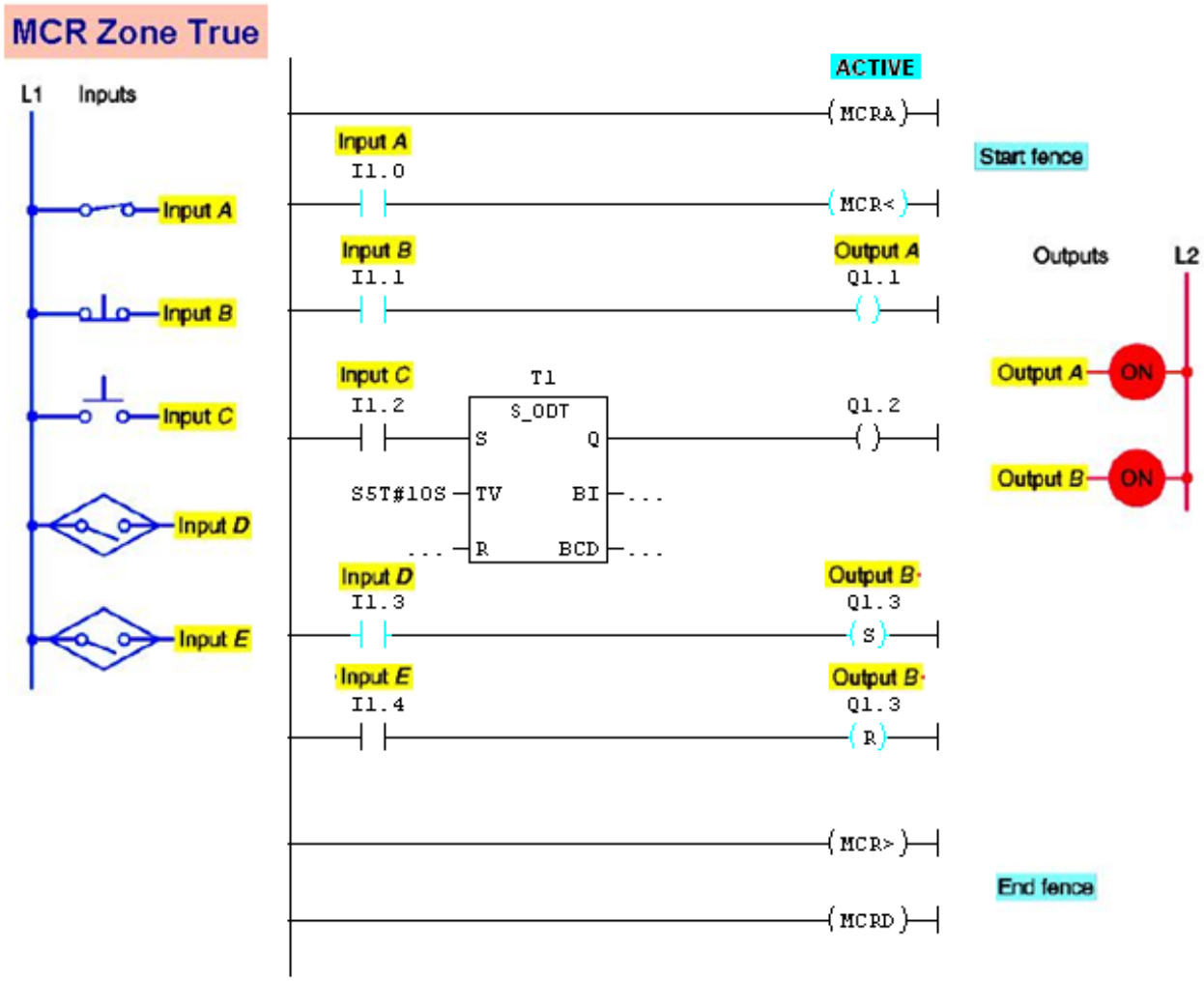
MCR functionality is activated by the MCRA rung. It is then possible to create up to eight nested MCR zones. In the example above there is only one MCR zone.



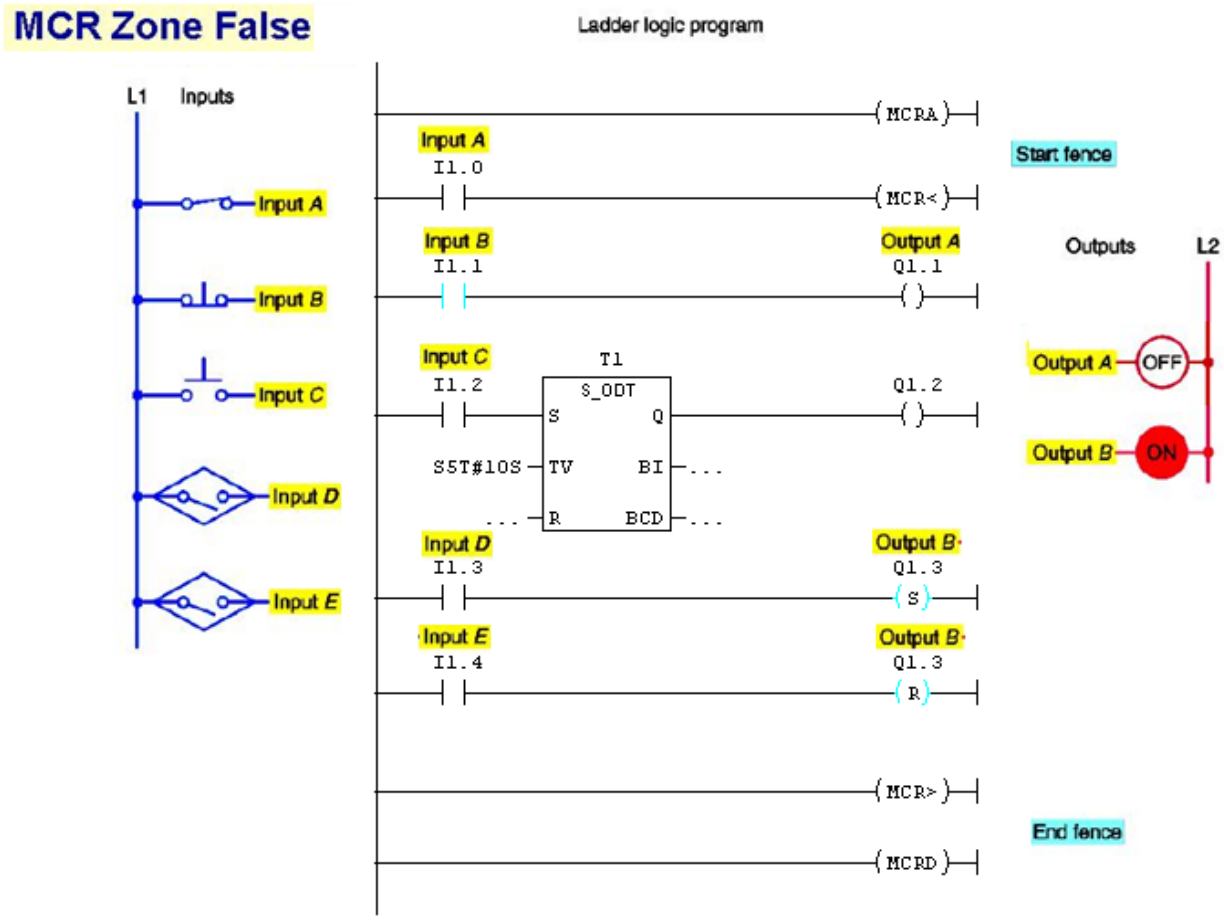
MCR Instruction Programmed To Control a Fenced Zone:

The Master Control Reset Activate (MCRA) and Master Control Reset Deactivate (MCRD) instruction are used in pairs to disable or enable a zone within a ladder program and has ***no address.*** You program the first zone (MCR<) with input instructions in the rung and the ending the zone by (MCR>) without any other instructions in the rung.





Try to use both retentive and non-retentive timers to see the effect of controlled zone.


MCR Zone False
Ladder logic program
L1
Inputs
Input A
Input B
Input C
Input D
Input E
(MCRA)
Input A
I1.0
Start fence
(MCR<)
Input B
I1.1
Output A
Q1.1
( )
Outputs
L2
Input C
I1.2
T1
S_ODT
S
Q
Q1.2
S5T#10S
TV
BI
R
BCD
Output A
OFF
Output B
ON
Input D
I1.3
Output B
Q1.3
(S)
Input E
I1.4
Output B
Q1.3
(R)
(MCR>)
End fence
(MCRD)

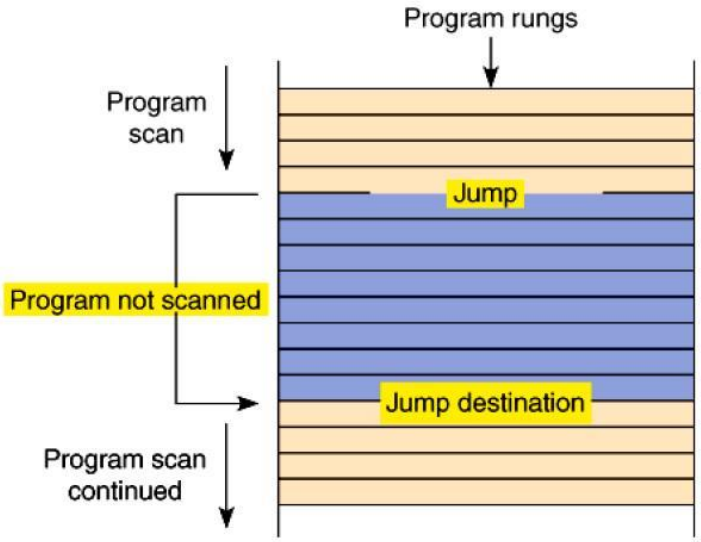
Jump Instruction

As in computer programming, it is sometimes desirable to be able to jump over certain program instructions. The jump instruction (JMP) is an output instruction used for this purpose. The advantages to the jump instruction include:

- The ability to reduce the processor scans time by jumping over instructions not pertinent to the machines operation at that instant.
- The PLC can hold more than one program and scan only the program appropriate to operator requirements
- Sections of a program can be jumped when a production fault occurs.

Jump Operation:

By using the jump instruction, you can branch or skip to different portions of a program and freeze all affected outputs in their last state.



Jumps are normally allowed in scan both the forward and backward directions.

Jumping over counters and timers will stop them from being incremented.

Description

With Siemens PLC, You can use logic control instructions in all logic blocks: organization blocks (OBs), function blocks (FBs), and functions (FCs).

There are logic control instructions to perform the following functions:

· ---(JMP)--- Unconditional Jump

· ---(JMP)--- Conditional Jump

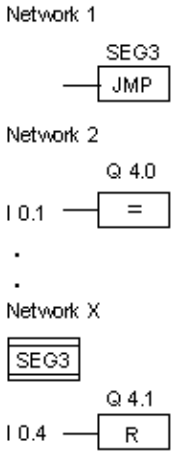
· ---(JMPN)--- Jump-If-Not

Label as Address

The address of a Jump instruction is a label. A label consists of a maximum of four characters. The first character must be a letter of the alphabet; the other characters can be letters or numbers (for example, SEG3). The jump label indicates the destination to which you want the program to jump.

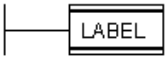
Label as Destination

The destination label must be at the beginning of a network. You enter the destination label at the beginning of the network by selecting LABEL from the ladder logic browser. An empty box appears. In the box, you type the name of the label.



LABEL Label

Symbol

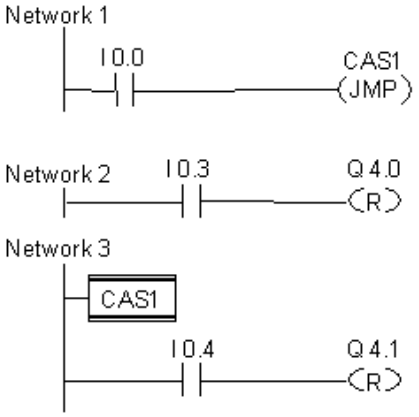


Description

LABEL is the identifier for the destination of a jump instruction.
The first character must be a letter of the alphabet; the other characters can be letters or numbers (for example, CAS1).
A jump label (LABEL) must exist for every ---(JMP) or ---(JMPN).

Example

Example



If I0.0 = "1", the jump to label CAS1 is executed. Because of the jump, the instruction to reset output Q4.0 is not executed even if there is a logic "1" at I0.3.

---(JMP)--- Unconditional Jump

Symbol

<label name>
---(JMP)

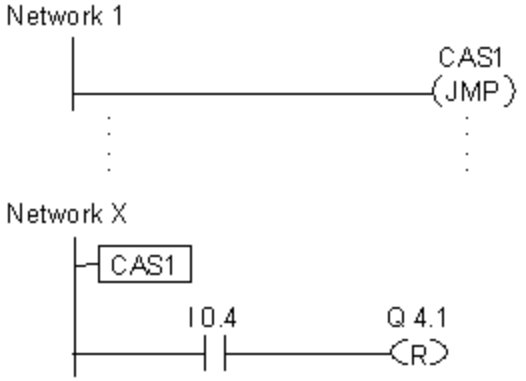
Description

---(JMP) (jump within the block when 1) functions as an absolute jump when there is no other Ladder element between the left-hand power rail and the instruction (see example).
A destination (LABEL) must also exist for every ---(JMP).
All instructions between the jump instruction and the label are not executed.

Status word

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
writes:	-	-	-	-	-	-	-	-	-

Example



The jump is always executed and the instructions between the jump instruction and the jump label are missed out.

---(JMP)--- Conditional Jump

Symbol

<label name>
---(JMP)

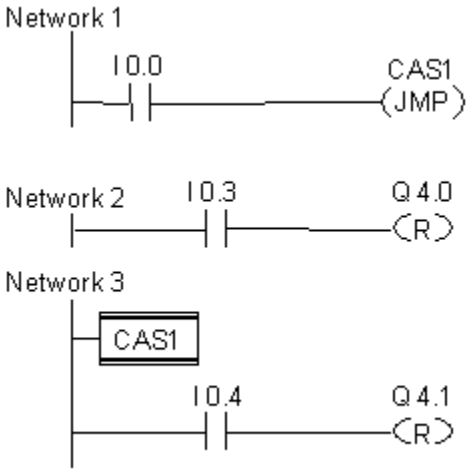
Description

---(JMP) (jump within the block when 1) functions as a conditional jump when the RLO of the previous logic operation is "1".
A destination (LABEL) must also exist for every ---(JMP).
All instructions between the jump instruction and the label are not executed.
If a conditional jump is not executed, the RLO changes to "1" after the jump instruction.

Status word

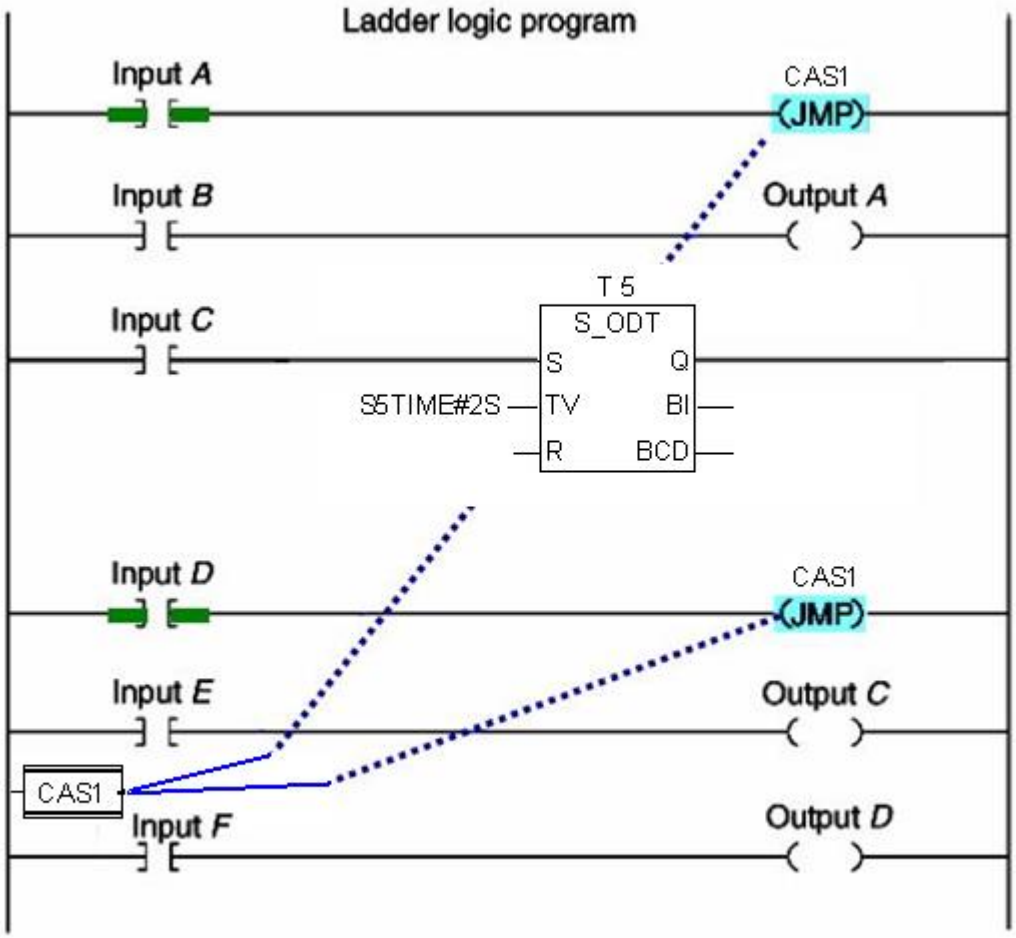
	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
writes:	-	-	-	-	-	0	1	1	0

Example



If I0.0 = "1", the jump to label CAS1 is executed. Because of the jump, the instruction to reset output Q4.0 is not executed even if there is a logic "1" at I0.3.

Jump-To-Label from Two Locations:

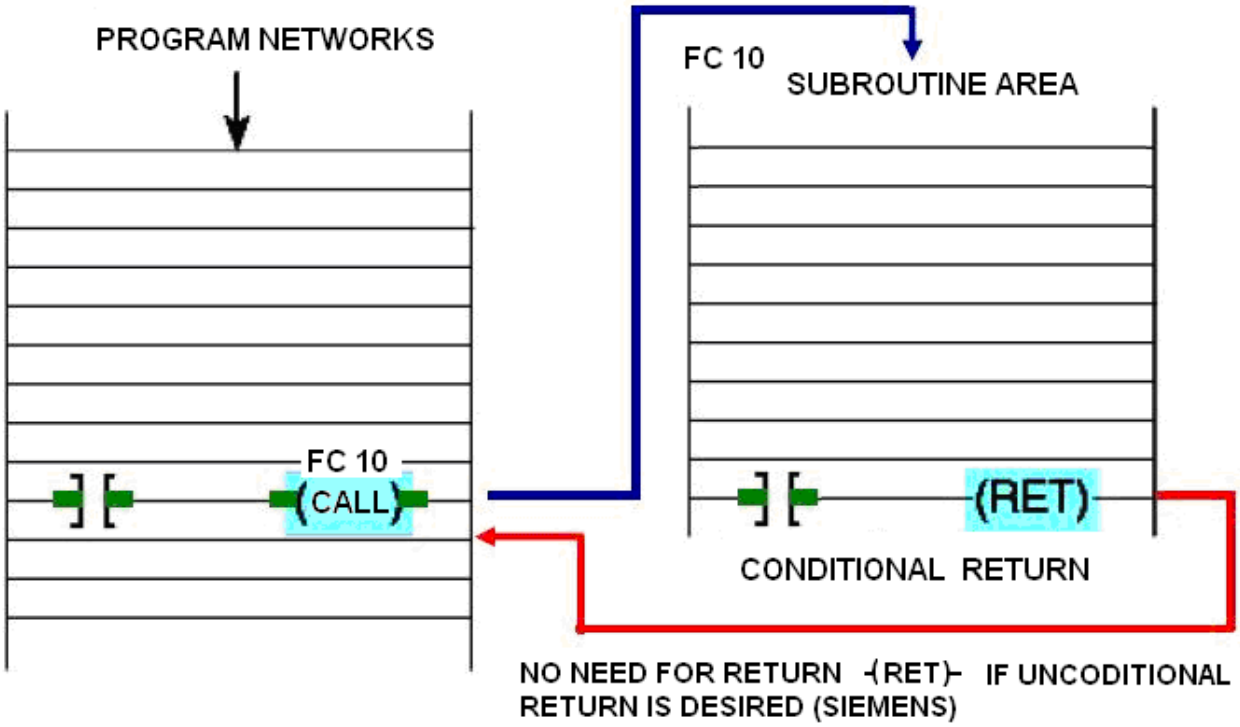


Avoid jumping backwards in the program too many times as this may increase the scan beyond the maximum allowable time. The processor has a watchdog timer that sets the maximum time for a total program scan. If this time is exceeded, the processor will indicate a fault and shut down.

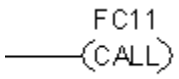
You should never jump into an MCR zone. Instructions that are programmed within the MCR zone starting at the LBL instruction and ending at the end MCR instruction will always be evaluated as though the MCR zone is true, without consideration to the state of the start MCR instruction.

CALL Subroutine:

Another valuable tool in PLC programming is to be able to escape from the main program and go to a program subroutine to perform certain functions and then return to the main program.

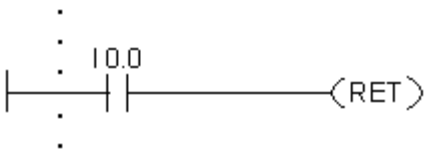


SIEMENS Subroutine-Related Instructions:



The *CALL* instruction causes the scan to jump to the program file designated in the instruction. It is the only parameter entered in the instruction.

When rung conditions are true for this output instruction, it causes the processor to jump to the targeted subroutine file.

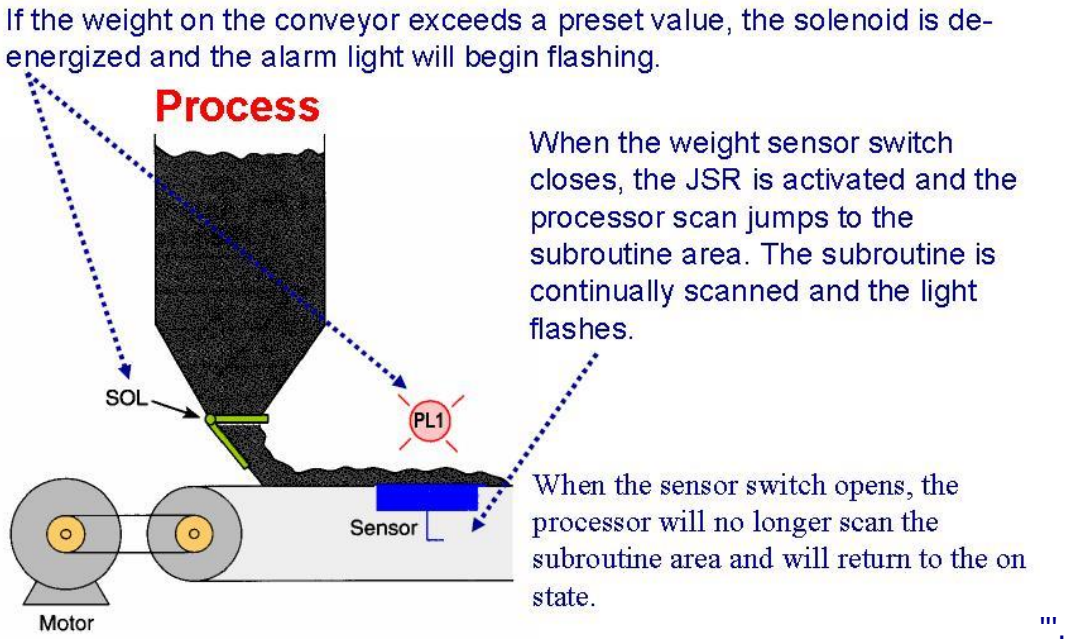


The ***RET*** instruction is an output instruction that marks the end of the subroutine file. It causes the scan to return to the main program at the instruction following the JSR instruction where it exited the program.

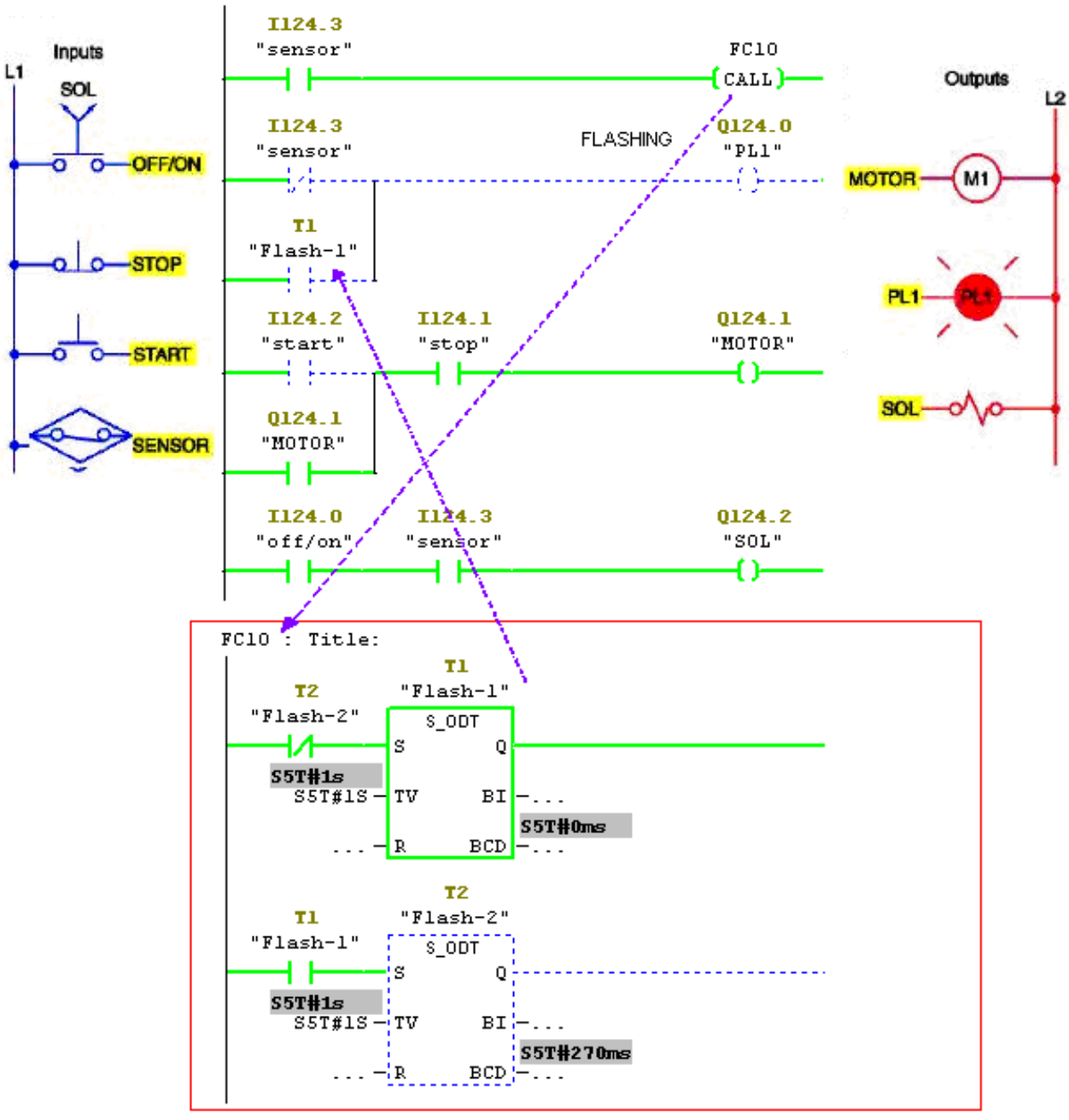
The scan returns from the end of the file if there is no RET instruction. The rung containing the RET instruction may be conditional if this rung precedes the end of the subroutine. In this way, the processor omits the balance of a subroutine only if its rung condition is true.

Flashing Pilot Light Subroutine:

If the weight on the conveyor exceeds a preset value, the solenoid is de-energized and the alarm light will begin flashing.



Flashing Pilot Light Subroutine Program:



Fault Routine:

PLC controllers allow you to design a subroutine file as a fault routine. If used, it determines how the processor responds to a programming error.

There are two kinds of major faults that result in a processor fault: recoverable and non-recoverable faults.

When there is a fault routine, and the fault is recoverable, the fault routine is executed.

If the fault is non-recoverable, the fault routine is scanned once and shuts down.

Either way, the fault routine allows for an orderly shutdown.

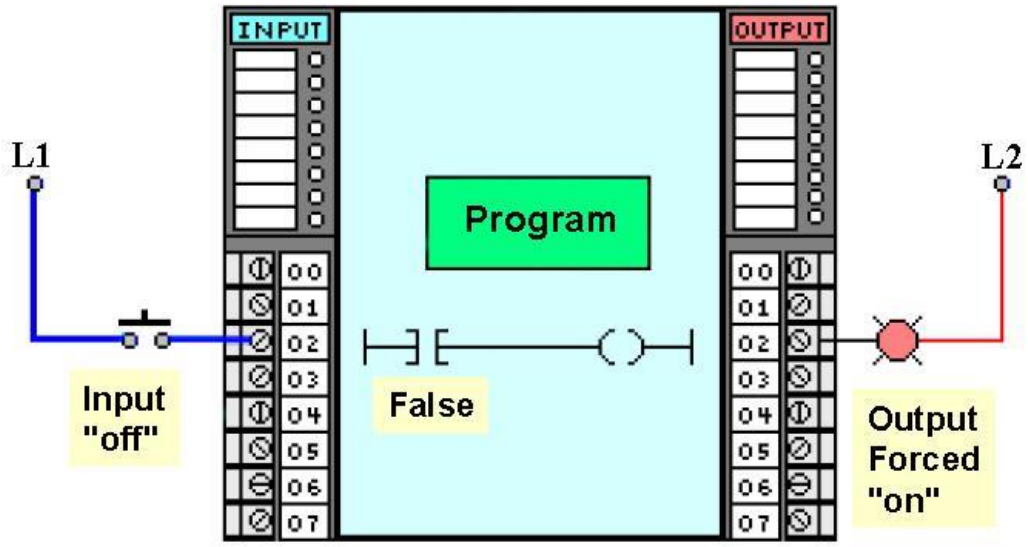
PART TEN

FPRCING EXTERNAL IO ADDRESSES

Forcing External *I0* Addresses

Forcing External *I0* Addresses:

The forcing capability of a PLC allows the user to turn an external input or output "on" or "off" from the keyboard of the programmer.

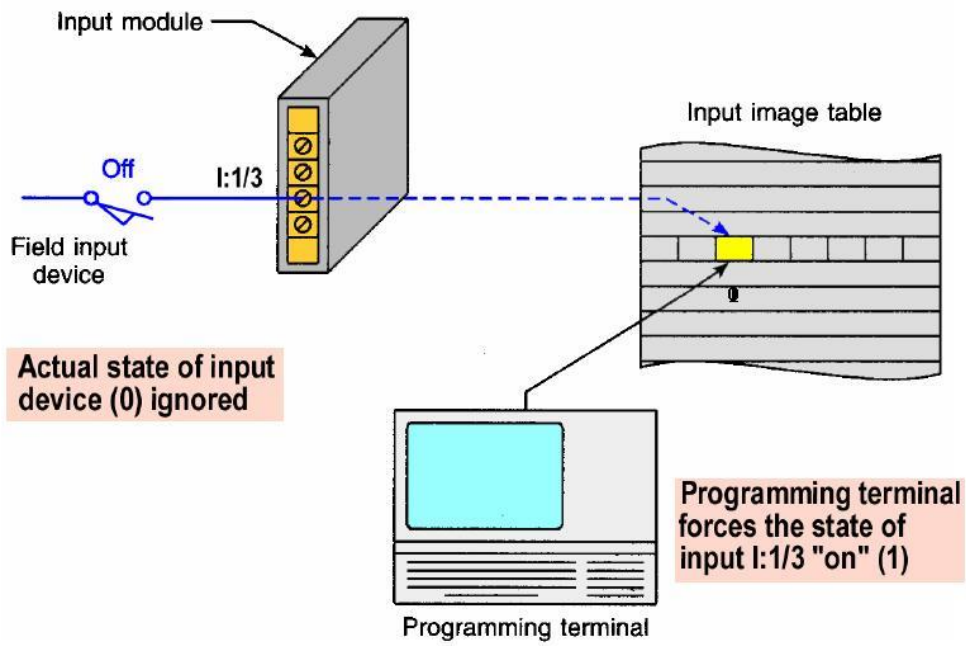


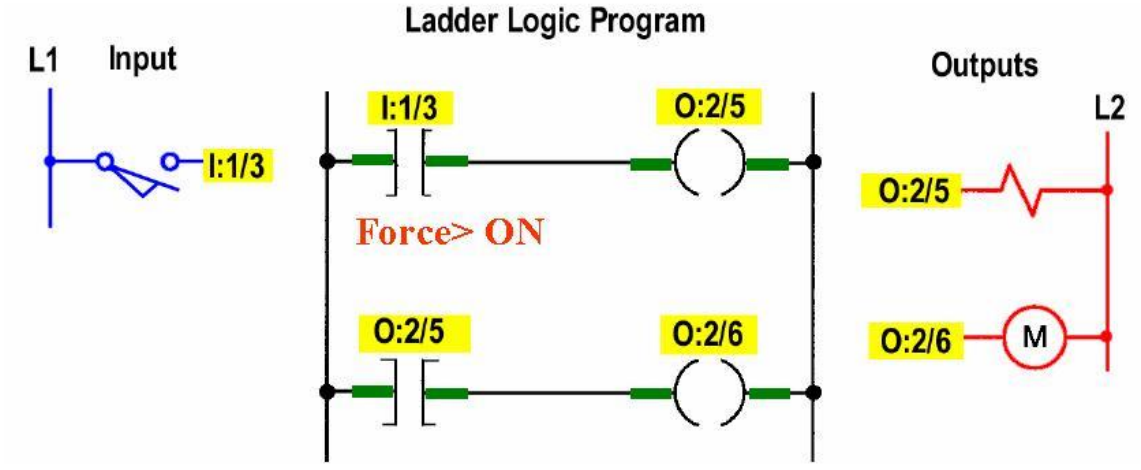
Forcing Inputs:

Overriding of physical inputs on conventional relay control systems can be accomplished by installing hardwire jumpers. With PLC control this is not necessary as the input data table values can be forced to an "on" or "off" state.

Forcing inputs manipulates the input image table file bits and thus affects all areas of the program that uses those bits.

Forcing an Input Address On:





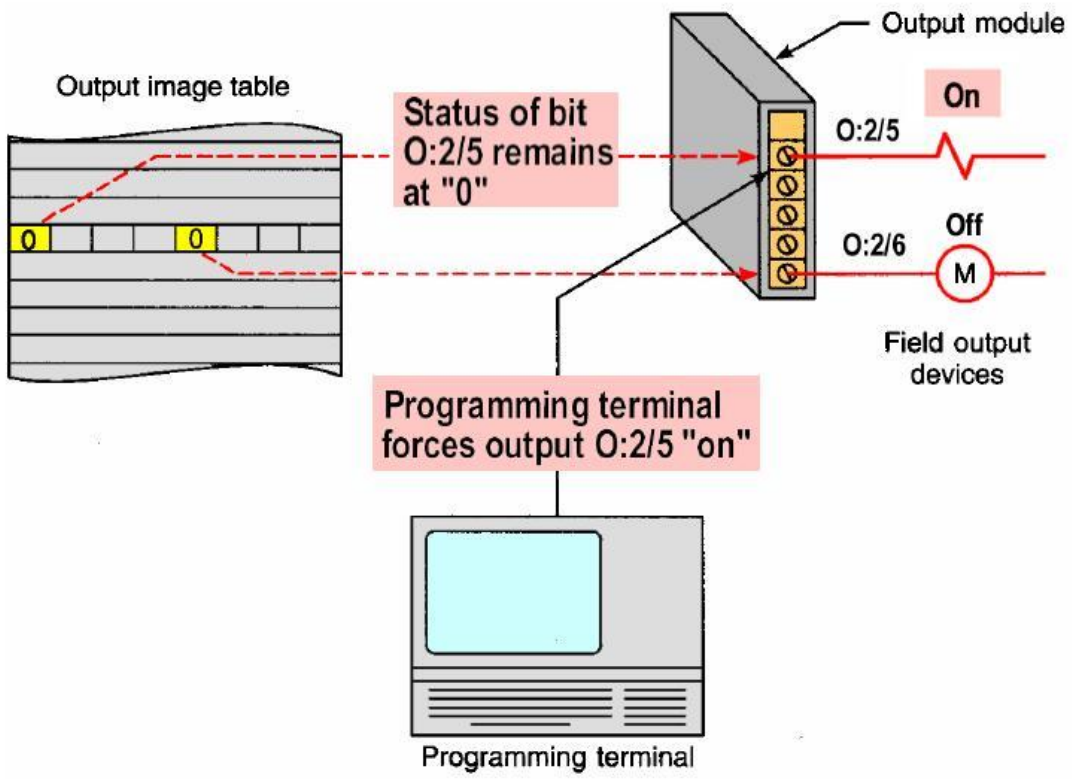
Forcing Outputs:

Forcing outputs affects only the addressed output terminal. When we force an output address; we are forcing only the output terminal to an on or off state.

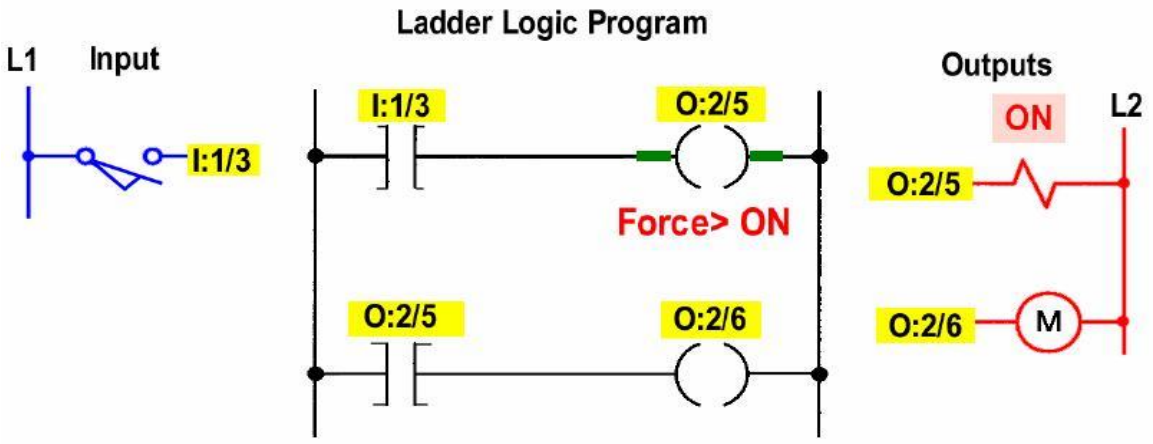
The output image table file bits are unaffected; therefore, your program will be unaffected. The forcing of outputs is done just before the output image table file is updated.

By forcing outputs "off" you can prevent the controller from energizing those outputs, even though the ladder logic, which normally controls them, may be true.

Forcing an Output Address On:



Forcing an Output Address On:



Forcing outputs affects only the addressed output terminal.

Using Forcing Functions:

The Force functions can be applied when the processor is in the run mode.

An understanding of the potential effect that forcing given inputs or outputs will have on the machine operation is essential to avoid possible personal injury and equipment damage.

Most programming terminals provide some visible means of alerting the user that a force is in effect.

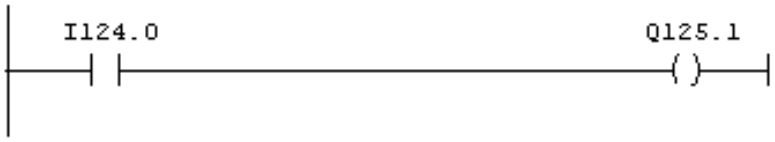
SIEMENS PLC

Force in/Force out

Step 1: Enter program

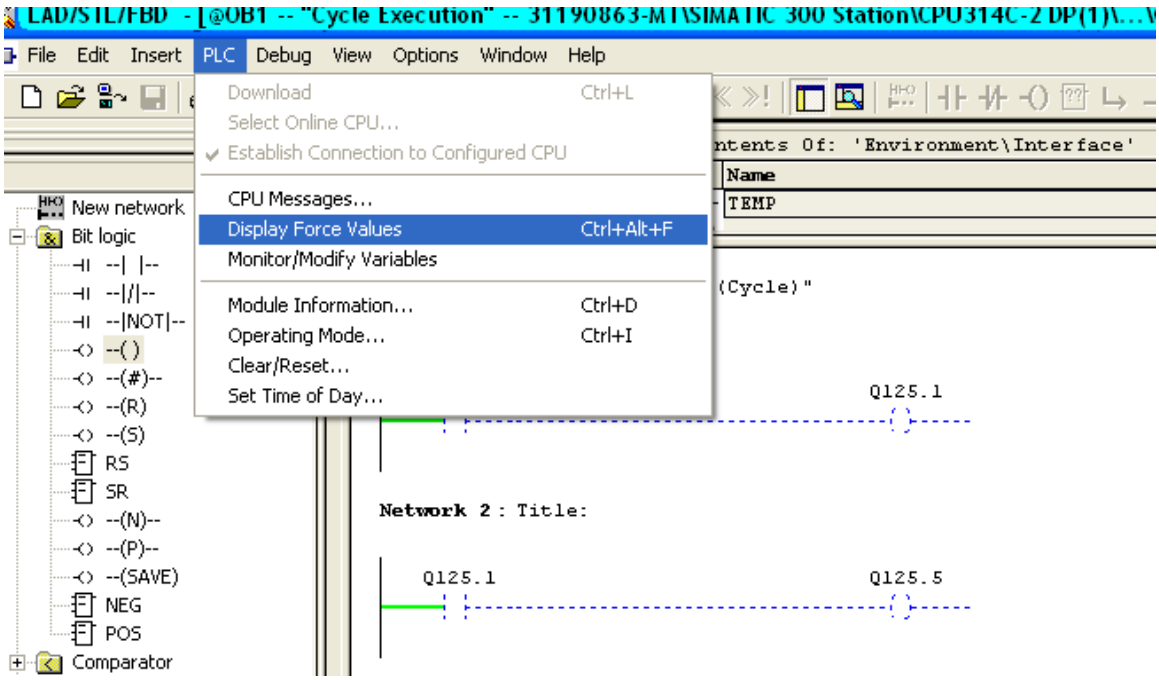
OB1 : "Main Program Sweep (Cycle)"

Network 1 : Title:

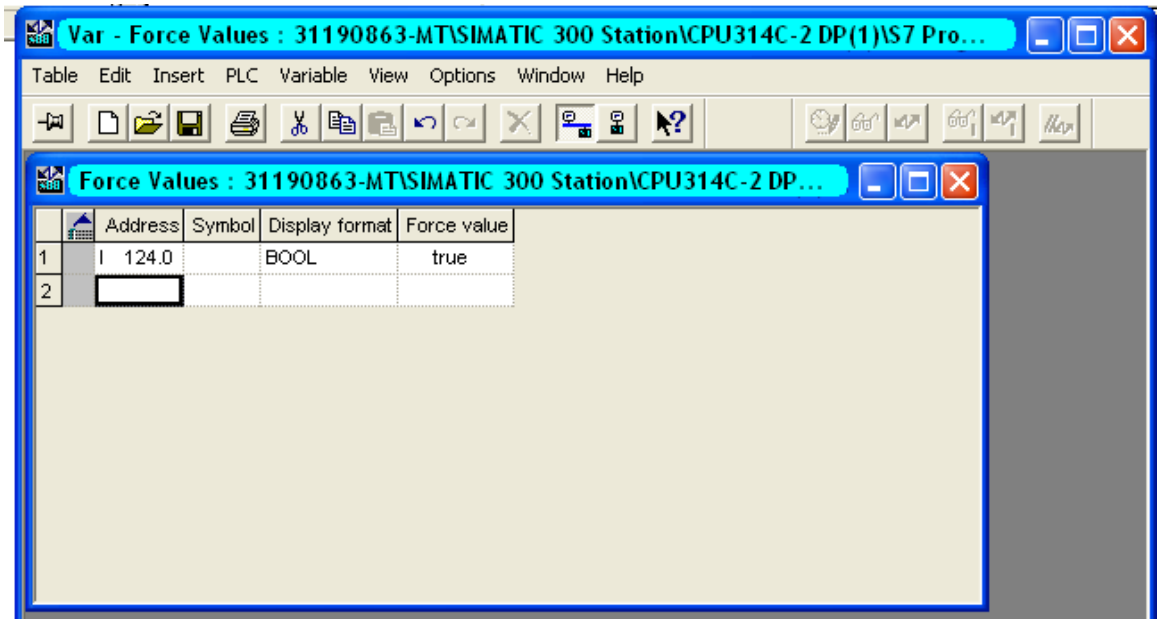


Network 2: Title:

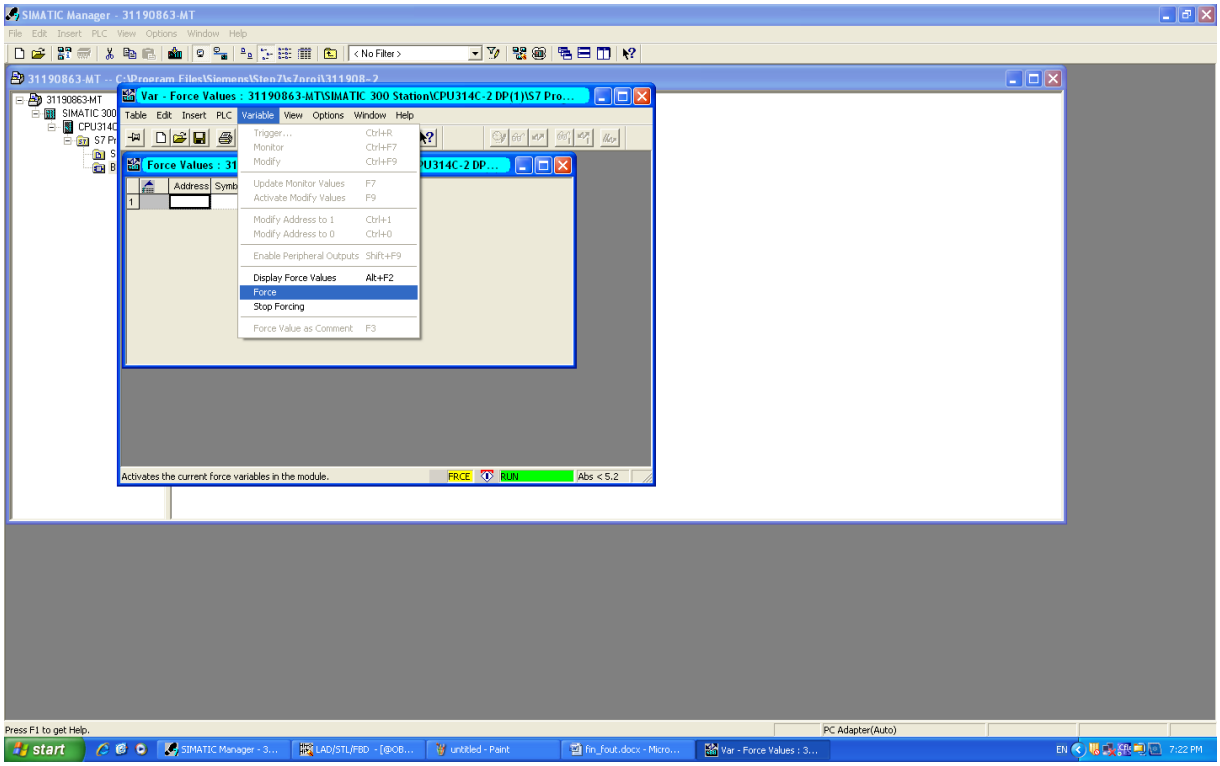
Step 2: Download program and monitor ON
Step 3: To, Activate forcing on: GOTO PLC – DISPLAY FORCE VALUES



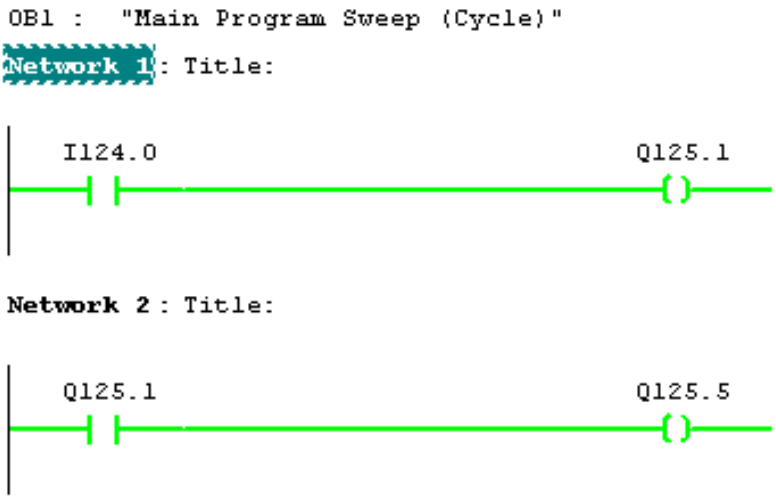
Step 4: Enter values in table for address and force value.



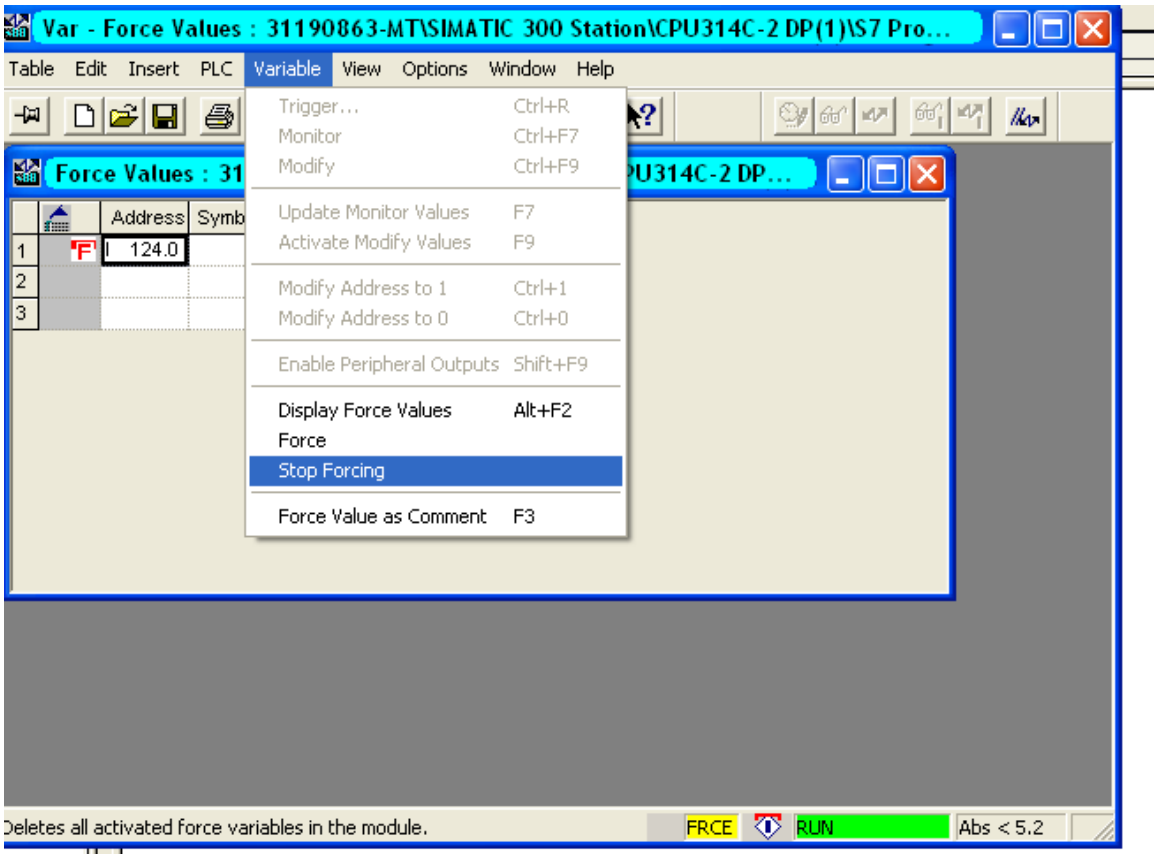
Step 5: GOTO VARIABLE and press FORCE: Observe Output



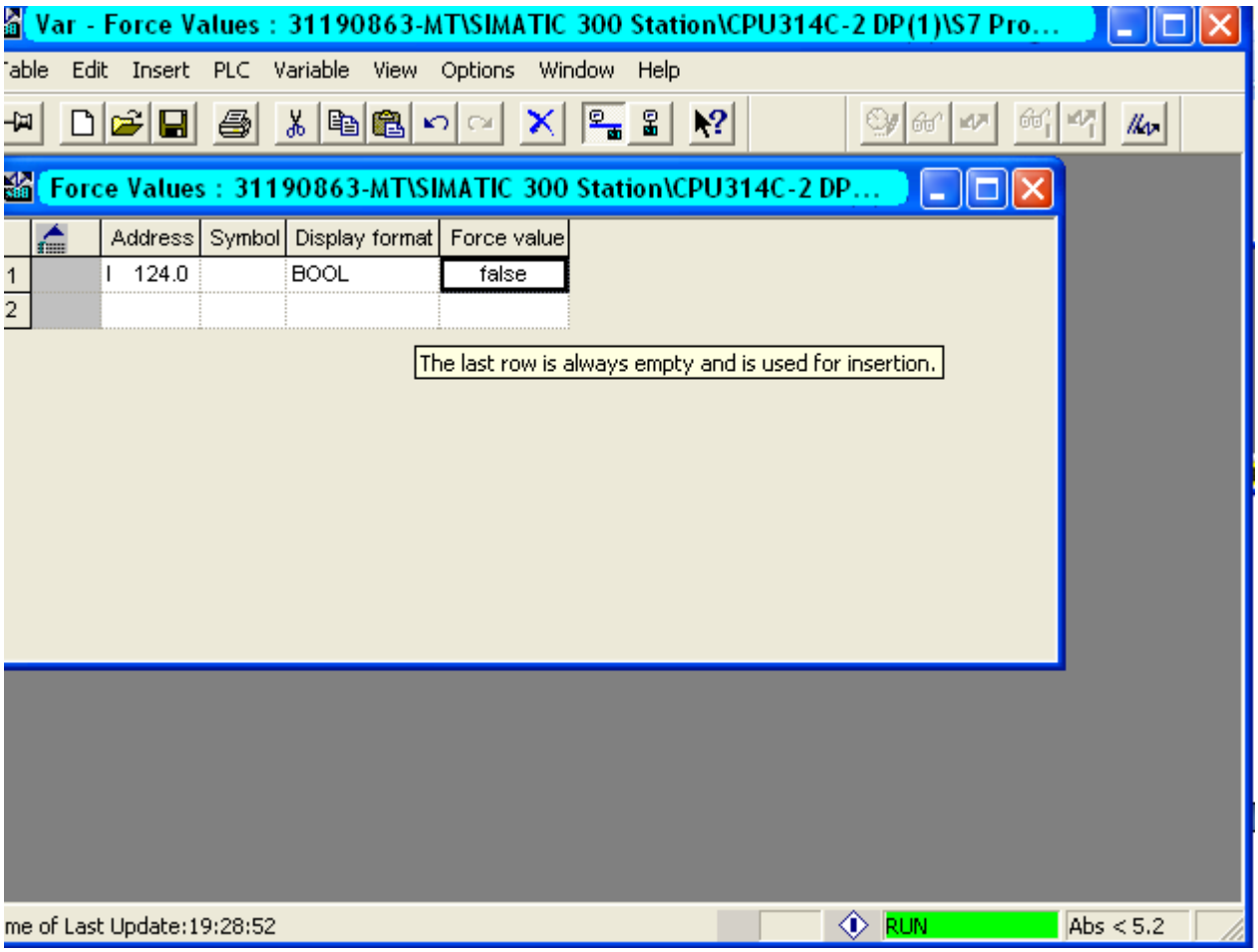
Step 6: Observe program, because of forcing input, both outputs are HIGH



Step 7: To UNDO forcing, press STOP FORCING, both OUTPUTS will be low



Step 7: To make forcing OFF, switch on outputs manually from inputs. Turn ON the outputs and then in the variable table, enter the inputs and FORCE VALUE – 0, it will turn off the outputs.



Wiring of Stop Push Buttons:

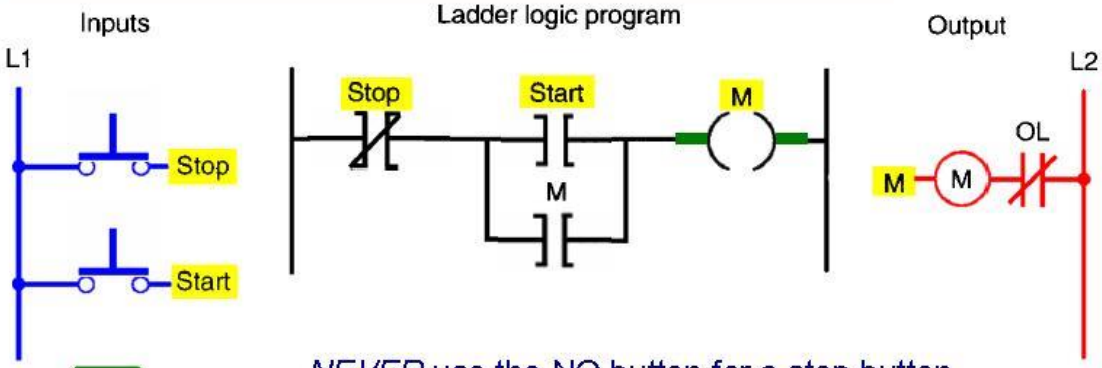
The wiring of stop buttons is another important safety consideration. A stop button is generally considered a safety function as well as an operating function.

NC Pushbutton

Always use a NC button for a stop button

As such, it should be wired using a NC contact and programmed to examine for an on condition. Using a NO contact programmed to examine for an off condition will produce the same logic, however, this is not preferred and is considered to be not as safe.

Normally open pushbutton stop configuration



NEVER use the NO button for a stop button.
If a NO button is used and the circuit between the button and the input point were to be broken the PLC logic control could never react to the stop command - since the input would never be true.

NO Pushbutton

PART ELEVEN

SHIFT & ROTATE INSTRUCTIONS

11 Shift and Rotate Instructions

11.1 Shift Instructions

11.1.1 Overview of Shift Instructions

Description

You can use the Shift instructions to move the contents of input IN bit by bit to the left or the right (see also CPU Registers). Shifting to the left multiplies the contents of input IN by 2 to the power n (2^n); shifting to the right divides the contents of input IN by 2 to the power n (2^n). For example, if you shift the binary equivalent of the decimal value 3 to the left by 3 bits, you obtain the binary equivalent of the decimal value 24 in the accumulator. If you shift the binary equivalent of the decimal value 16 to the right by 2 bits, you obtain the binary equivalent of the decimal value 4 in the accumulator.

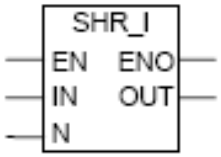
The number that you supply for input parameter N indicates the number of bits by which to shift. The bit places that are vacated by the Shift instruction are either filled with zeros or with the signal state of the sign bit (a 0 stands for positive and a 1 stands for negative). The signal state of the bit that is shifted last is loaded into the CC 1 bit of the status word. The CC 0 and OV bits of the status word are reset to 0. You can use jump instructions to evaluate the CC 1 bit.

The following shift instructions are available:

- SHR_I Shift Right Integer
- SHR_DI Shift Right Double Integer
- SHL_W Shift Left Word
- SHR_W Shift Right Word
- SHL_DW Shift Left Double Word
- SHR_DW Shift Right Double Word

11.1.2 SHR_I Shift Right Integer

Symbol

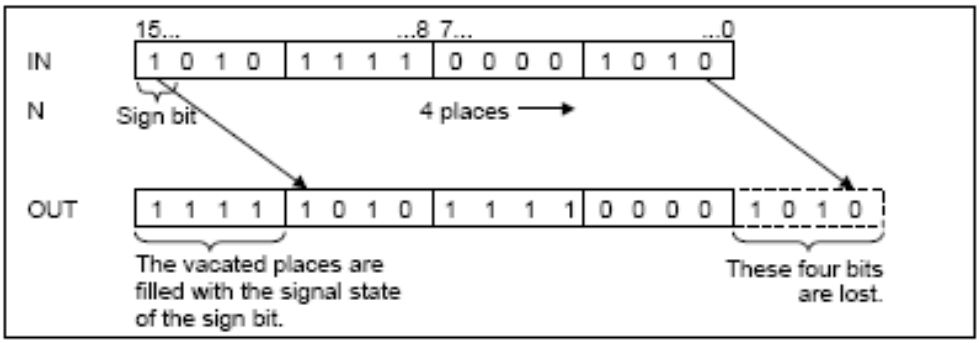


Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, L, D	Enable input
ENO	BOOL	I, Q, M, L, D	Enable output
IN	INT	I, Q, M, L, D	Value to shift
N	WORD	I, Q, M, L, D	Number of bit positions to shift
OUT	INT	I, Q, M, L, D	Result of shift instruction

Description

SHR_I (Shift Right Integer) is activated by a logic "1" at the Enable (EN) Input. The SHR_I instruction is used to shift bits 0 to 15 of input IN bit by bit to the right. Bits 16 to 31 are not affected. The input N specifies the number of bits by which to shift. If N is larger than 16, the command acts as if N were equal to 16. The bit positions shifted in from the left to fill vacated bit positions are assigned the logic state of bit 15 (sign bit for the integer). This means these bit positions are assigned "0" if the integer is positive and "1" if the integer is negative. The result of the shift instruction can be scanned at output OUT. The CC 0 bit and the OV bit are set to "0" by SHR_I if N is not equal to 0.

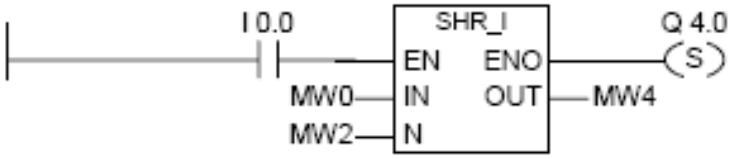
ENO has the same signal state as EN.



Status word

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
writes:	x	x	x	x	-	x	x	x	1

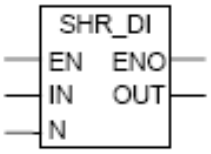
Example



The SHR_I box is activated by logic "1" at I0.0. MW0 is loaded and shifted right by the number of bits specified with MW2. The result is written to MW4. Q4.0 is set.

11.1.3 SHR_DI Shift Right Double Integer

Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, L, D	Enable input
ENO	BOOL	I, Q, M, L, D	Enable output
IN	DINT	I, Q, M, L, D	Value to shift
N	WORD	I, Q, M, L, D	Number of bit positions to shift
OUT	DINT	I, Q, M, L, D	Result of shift instruction

Description

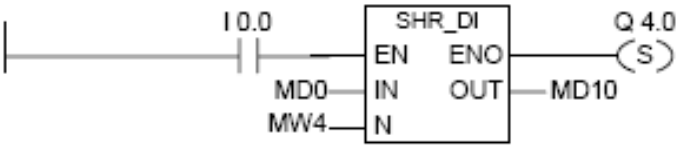
SHR_DI (Shift Right Double Integer) is activated by a logic "1" at the Enable (EN) Input. The SHR_DI instruction is used to shift bits 0 to 31 of input IN bit by bit to the right. The input N specifies the number of bits by which to shift. If N is larger than 32, the command acts as if N were equal to 32. The bit positions shifted in from the left to fill vacated bit positions are assigned the logic state of bit 31 (sign bit for the double integer). This means these bit positions are assigned "0" if the integer is positive and "1" if the integer is negative. The result of the shift instruction can be scanned at output OUT. The CC 0 bit and the OV bit are set to "0" by SHR_DI if N is not equal to 0.

ENO has the same signal state as EN.

Status word

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
writes:	x	x	x	x	-	x	x	x	1

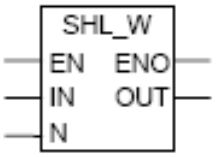
Example



The SHR_DI box is activated by logic "1" at I0.0. MD0 is loaded and shifted right by the number of bits specified with MW4. The result is written to MD10. Q4.0 is set.

11.1.4 SHL_W Shift Left Word

Symbol

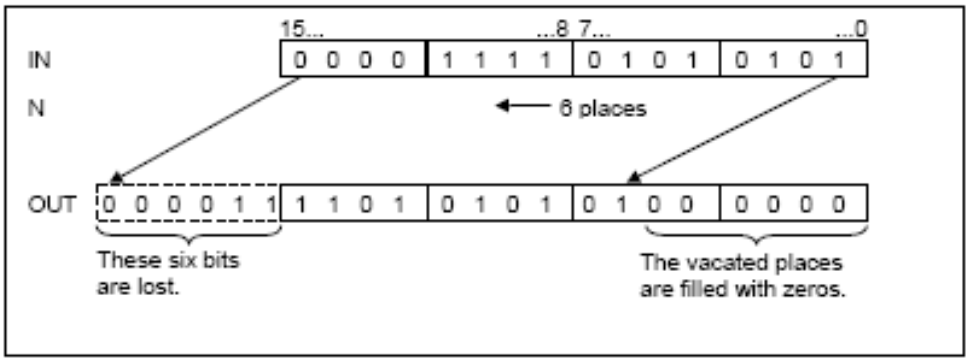


Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, L, D	Enable input
ENO	BOOL	I, Q, M, L, D	Enable output
IN	WORD	I, Q, M, L, D	Value to shift
N	WORD	I, Q, M, L, D	Number of bit positions to shift
OUT	WORD	I, Q, M, L, D	Result of shift instruction

Description

SHL_W (Shift Left Word) is activated by a logic "1" at the Enable (EN) Input. The SHL_W instruction is used to shift bits 0 to 15 of input IN bit by bit to the left. Bits 16 to 31 are not affected. The input N specifies the number of bits by which to shift. If N is larger than 16, the command writes a "0" at output OUT and sets the bits CC 0 and OV in the status word to "0". N zeros are also shifted in from the right to fill vacated bit positions. The result of the shift instruction can be scanned at output OUT. The CC 0 bit and the OV bit are set to "0" by SHL_W if N is not equal to 0.

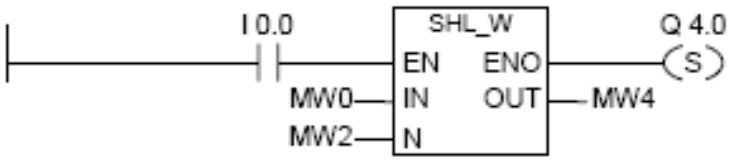
ENO has the same signal state as EN.



Status word

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
writes:	x	x	x	x	-	x	x	x	1

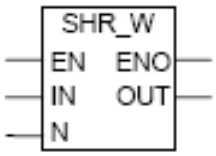
Example



The SHL_W box is activated by logic "1" at I0.0. MW0 is loaded and shifted left by the number of bits specified with MW2. The result is written to MW4. Q4.0 is set.

11.1.5 SHR_W Shift Right Word

Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, L, D	Enable input
ENO	BOOL	I, Q, M, L, D	Enable output
IN	WORD	I, Q, M, L, D	Value to shift
N	WORD	I, Q, M, L, D	Number of bit positions to shift
OUT	WORD	I, Q, M, L, D	Result word of shift instruction

Description

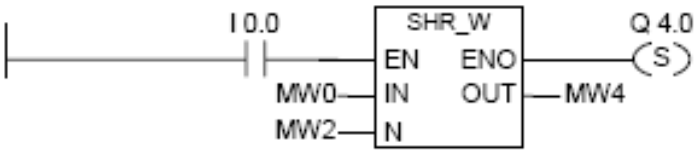
SHR_W (Shift Right Word) is activated by a logic "1" at the Enable (EN) Input. The SHR_W instruction is used to shift bits 0 to 15 of input IN bit by bit to the right. Bits 16 to 31 are not affected. The input N specifies the number of bits by which to shift. If N is larger than 16, the command writes a "0" at output OUT and sets the bits CC 0 and OV in the status word to "0". N zeros are also shifted in from the left to fill vacated bit positions. The result of the shift instruction can be scanned at output OUT. The CC 0 bit and the OV bit are set to "0" by SHR_W if N is not equal to 0.

ENO has the same signal state as EN.

Status word

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
writes:	x	x	x	x	-	x	x	x	1

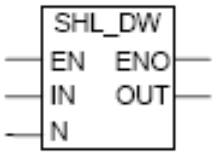
Example



The SHR_W box is activated by logic "1" at I0.0. MW0 is loaded and shifted right by the number of bits specified with MW2. The result is written to MW4. Q4.0 is set.

11.1.6 SHL_DW Shift Left Double Word

Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, L, D	Enable input
ENO	BOOL	I, Q, M, L, D	Enable output
IN	DWORD	I, Q, M, L, D	Value to shift
N	WORD	I, Q, M, L, D	Number of bit positions to shift
OUT	DWORD	I, Q, M, L, D	Result double word of shift instruction

Description

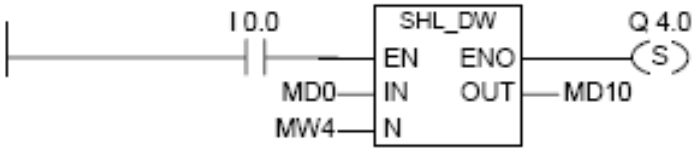
SHL_DW (Shift Left Double Word) is activated by a logic "1" at the Enable (EN) Input. The SHL_DW instruction is used to shift bits 0 to 31 of input IN bit by bit to the left. The input N specifies the number of bits by which to shift. If N is larger than 32, the command writes a "0" at output OUT and sets the bits CC 0 and OV in the status word to "0". N zeros are also shifted in from the right to fill vacated bit positions. The result double word of the shift instruction can be scanned at output OUT. The CC 0 bit and the OV bit are set to "0" by SHL_DW if N is not equal to 0.

ENO has the same signal state as EN.

Status word

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
writes:	x	x	x	x	-	x	x	x	1

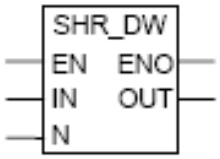
Example



The SHL_DW box is activated by logic "1" at I0.0. MD0 is loaded and shifted left by the number of bits specified with MW4. The result is written to MD10. Q4.0 is set.

11.1.7 SHR_DW Shift Right Double Word

Symbol

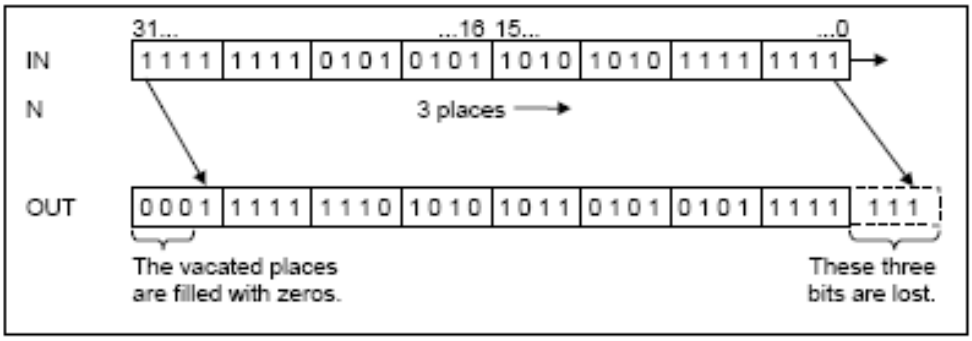


Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, L, D	Enable input
ENO	BOOL	I, Q, M, L, D	Enable output
IN	DWORD	I, Q, M, L, D	Value to shift
N	WORD	I, Q, M, L, D	Number of bit positions to shift
OUT	DWORD	I, Q, M, L, D	Result double word of shift instruction

Description

SHR_DW (Shift Right Double Word) is activated by a logic "1" at the Enable (EN) Input. The SHR_DW instruction is used to shift bits 0 to 31 of input IN bit by bit to the right. The input N specifies the number of bits by which to shift. If N is larger than 32, the command writes a "0" at output OUT and sets the bits CC 0 and OV in the status word to "0". N zeros are also shifted in from the left to fill vacated bit positions. The result double word of the shift instruction can be scanned at output OUT. The CC 0 bit and the OV bit are set to "0" by SHR_DW if N is not equal to 0.

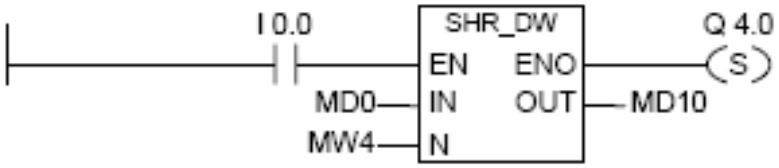
ENO has the same signal state as EN.



Status word

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
writes:	x	x	x	x	-	x	x	x	1

Example



The SHR_DW box is activated by logic "1" at I0.0. MD0 is loaded and shifted right by the number of bits specified with MW4. The result is written to MD10. Q4.0 is set.

11.2 Rotate Instructions

11.2.1 Overview of Rotate Instructions

Description

You can use the Rotate instructions to rotate the entire contents of input IN bit by bit to the left or to the right. The vacated bit places are filled with the signal states of the bits that are shifted out of input IN.

The number that you supply for input parameter N specifies the number of bits by which to rotate.

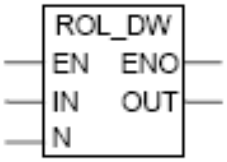
Depending on the instruction, rotation takes place via the CC 1 bit of the status word. The CC 0 bit of the status word is reset to 0.

The following rotate instructions are available:

- ROL_DW Rotate Left Double Word
- ROR_DW Rotate Right Double Word

11.2.2 ROL_DW Rotate Left Double Word

Symbol

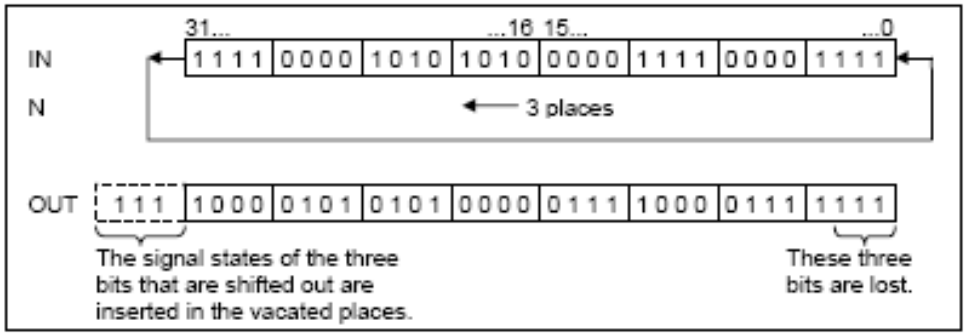


Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, L, D	Enable input
ENO	BOOL	I, Q, M, L, D	Enable output
IN	DWORD	I, Q, M, L, D	Value to rotate
N	WORD	I, Q, M, L, D	Number of bit positions to rotate
OUT	DWORD	I, Q, M, L, D	Result double word of rotate instruction

Description

ROL_DW (Rotate Left Double Word) is activated by a logic "1" at the Enable (EN) Input. The ROL_DW instruction is used to rotate the entire contents of input IN bit by bit to the left. The input N specifies the number of bits by which to rotate. If N is larger than 32, the double word IN is rotated by ((N-1) modulo 32)+1 positions. The bit positions shifted in from the right are assigned the logic states of the bits which were rotated out to the left. The result double word of the rotate instruction can be scanned at output OUT. The CC 0 bit and the OV bit are set to "0" by ROL_DW if N is not equal to 0.

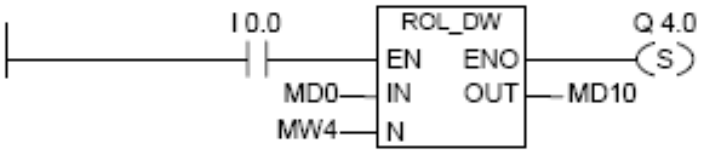
ENO has the same signal state as EN.



Status word

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
writes:	x	x	x	x	-	x	x	x	1

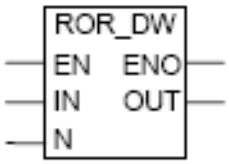
Example



The ROL_DW box is activated by logic "1" at I0.0. MD0 is loaded and rotated to the left by the number of bits specified with MW4. The result is written to MD10. Q4.0 is set.

11.2.3 ROR_DW Rotate Right Double Word

Symbol

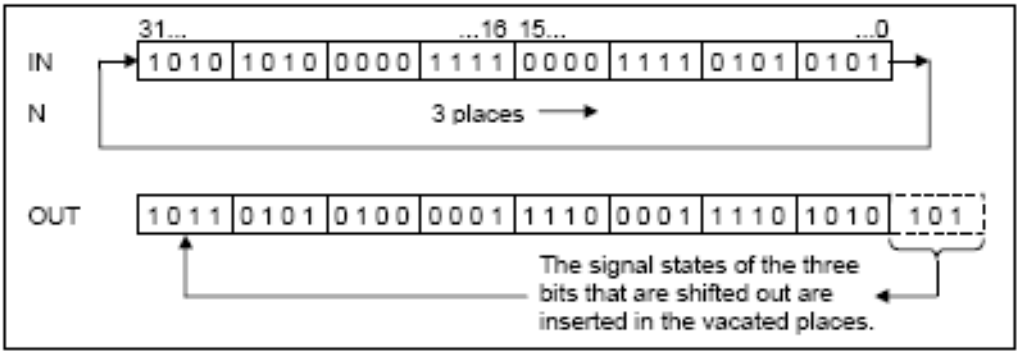


Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, L, D	Enable input
ENO	BOOL	I, Q, M, L, D	Enable output
IN	DWORD	I, Q, M, L, D	Value to rotate
N	WORD	I, Q, M, L, D	Number of bit positions to rotate
OUT	DWORD	I, Q, M, L, D	Result double word of rotate instruction

Description

ROR_DW (Rotate Right Double Word) is activated by a logic "1" at the Enable (EN) Input. The ROR_DW instruction is used to rotate the entire contents of input IN bit by bit to the right. The input N specifies the number of bits by which to rotate. If N is larger than 32, the double word IN is rotated by ((N-1) modulo 32)+1 positions. The bit positions shifted in from the left are assigned the logic states of the bits which were rotated out to the right. The result double word of the rotate instruction can be scanned at output OUT. The CC 0 bit and the OV bit are set to "0" by ROR_DW if N is not equal to 0.

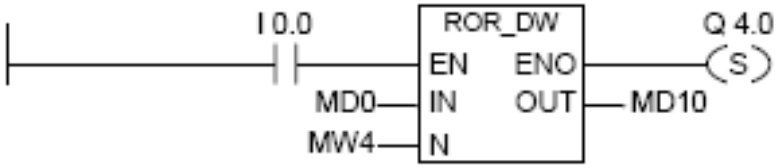
ENO has the same signal state as EN.



Status word

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
writes:	x	x	x	x	-	x	x	x	1

Example

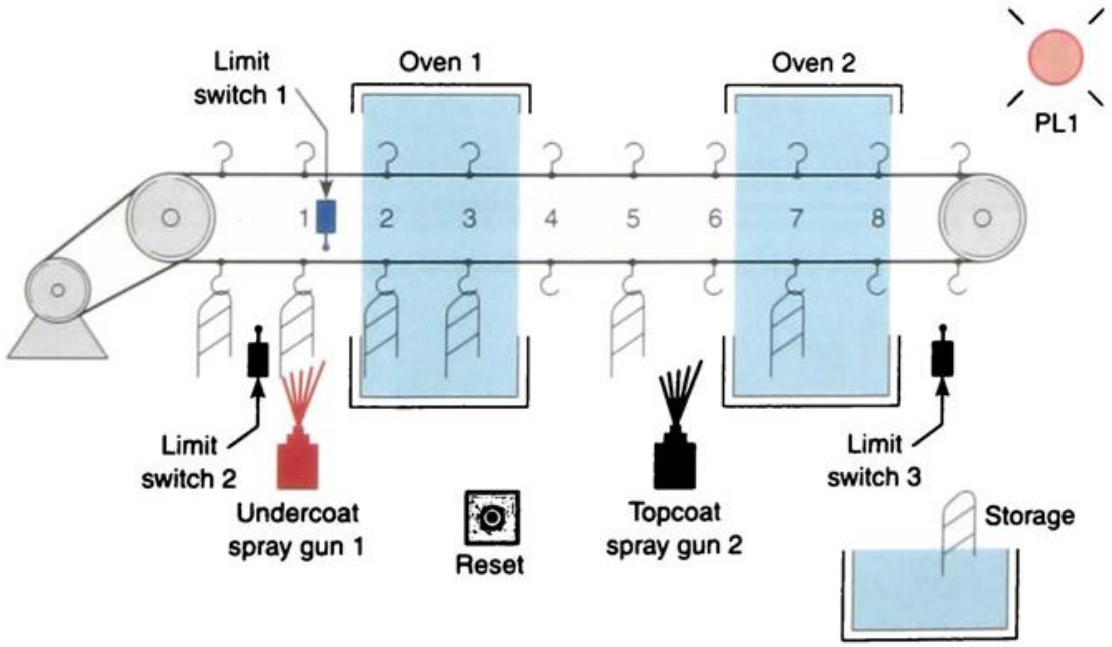


The ROR_DW box is activated by logic "1" at I0.0. MD0 is loaded and rotated to the right by the number of bits specified with MW4. The result is written to MD10. Q4.0 is set.

Application s of shift operations

The program of figure 12-26 illustrates a spray-painting operation controlled by a shift register. Each file bit location represents a station on the line, and the status of the bit indicates whether or not a part is present at that station.

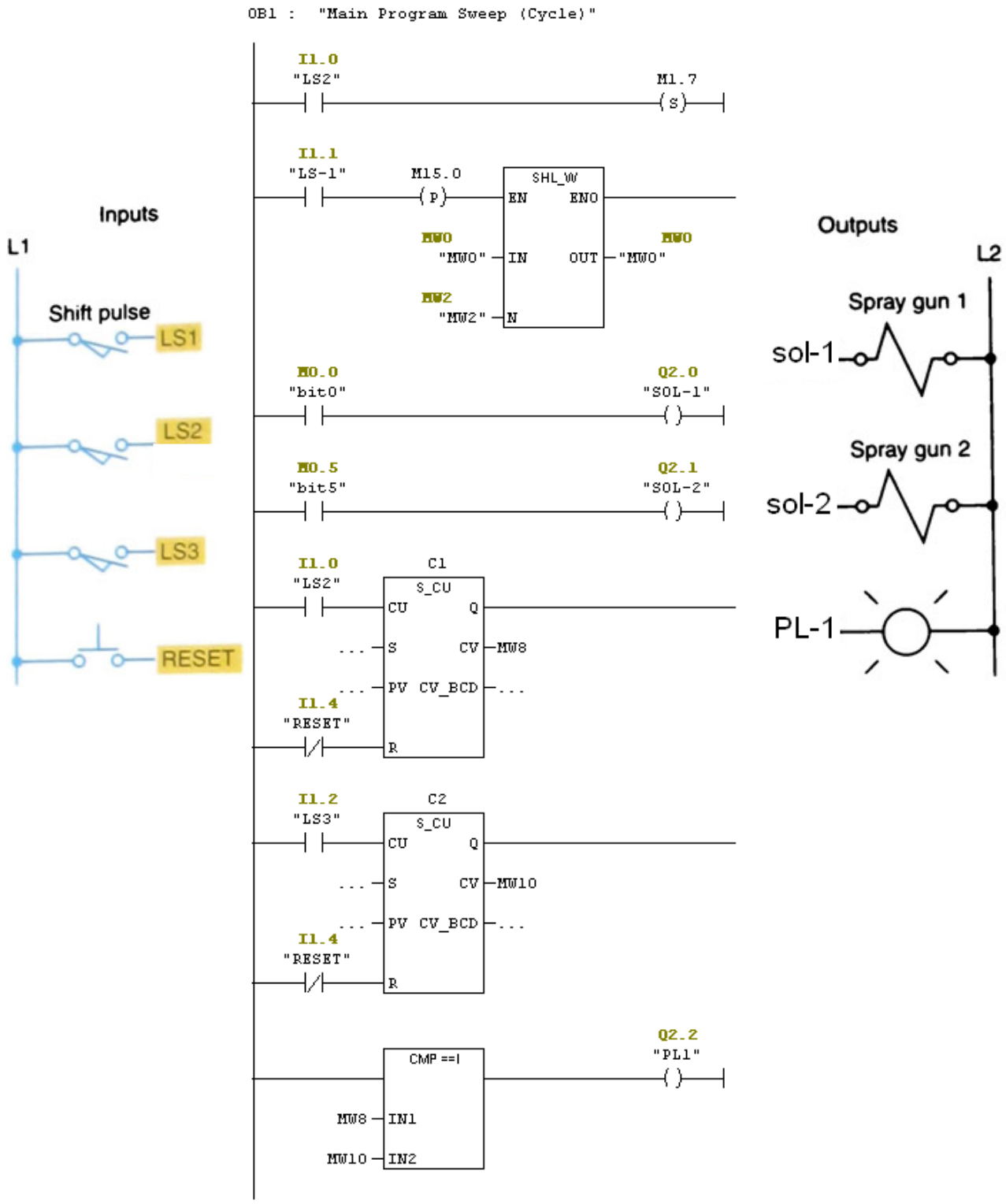
The bit address, I 1.0, detects whether a part has come on the line. The shift register's function is used to keep track of the items to be sprayed. A bit shift left instruction is used to indicate a forward motion of the line. As the parts pass along the production line, the shift register bit patterns represent the items on the conveyor hangers to be painted. LS1 is used to detect the hanger and LS2 detects the part. The logic of this operation is such that when a part to be painted and a part hanger occur together (indicated by the simultaneous operation of LS1 and LS2), a logic 1 is input into the shift register.
The logic 1 will cause the undercoat spray gun to operate, and five steps later, when a 1 occurs in the shift register, the topcoat spray gun is operated. Limit switch 3 counts the parts as they exit the oven. The count obtained by limit switch 2 and limit switch 3 should be equal at the end of the spray-painting run (PL1 is energized) and is an indication that the parts commencing the spray-painting run equal the parts that have completed it. A logic 0 in the shift register indicates that the conveyor has no parts on it to be sprayed, and it therefore inhibits the operation of the spray guns.



(a) Process

FIGURE 12-26 Shift register spray-painting application.

The Solution using Siemens S7



PART TWLEVE

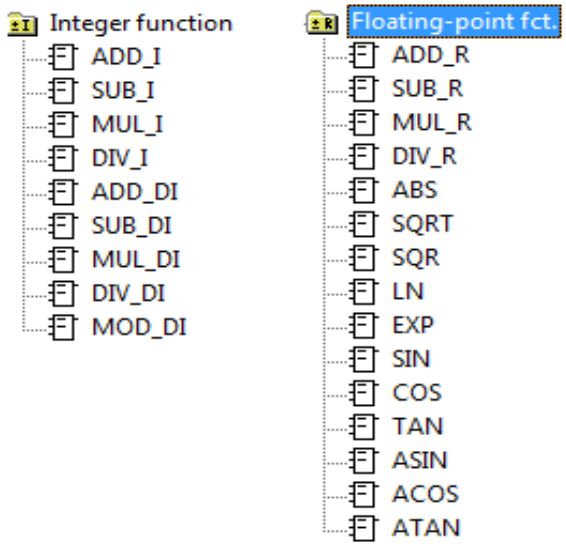
MATH INSTRUCTIONS

MATH INSTRUCTIONS

PLC math instructions allow you to perform arithmetic functions on values stored in memory words.

For example, assume you are using a counter to keep track of the numbers of parts manufactured and you would like to display how many more must be produced in order to reach a certain quota. This would require the data in the accumulated value of the counter to be subtracted from the quota required.

SIEMENS STEP7 MATH ICTIONS

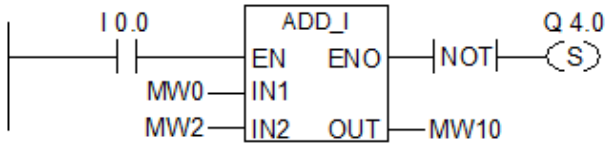


ADD Instruction:

The ADD instruction is an output instruction that performs the addition of two values stored in the referenced memory locations.

ADD_I Add Integer

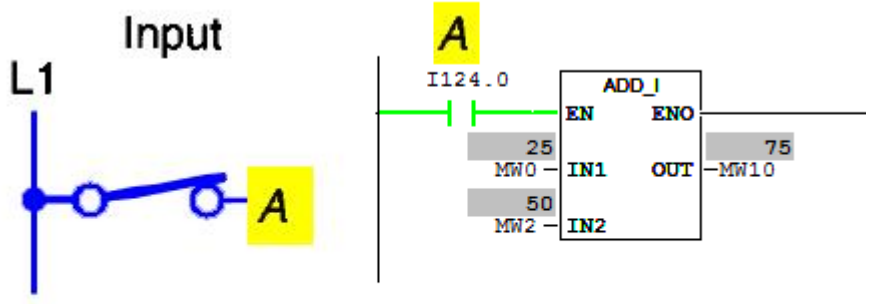
Example



The ADD_I box is activated if I0.0 = "1". The result of the addition MW0 + MW2 is output to MW10.

If the result was outside the permissible range for an integer, the output Q4.0 is set.

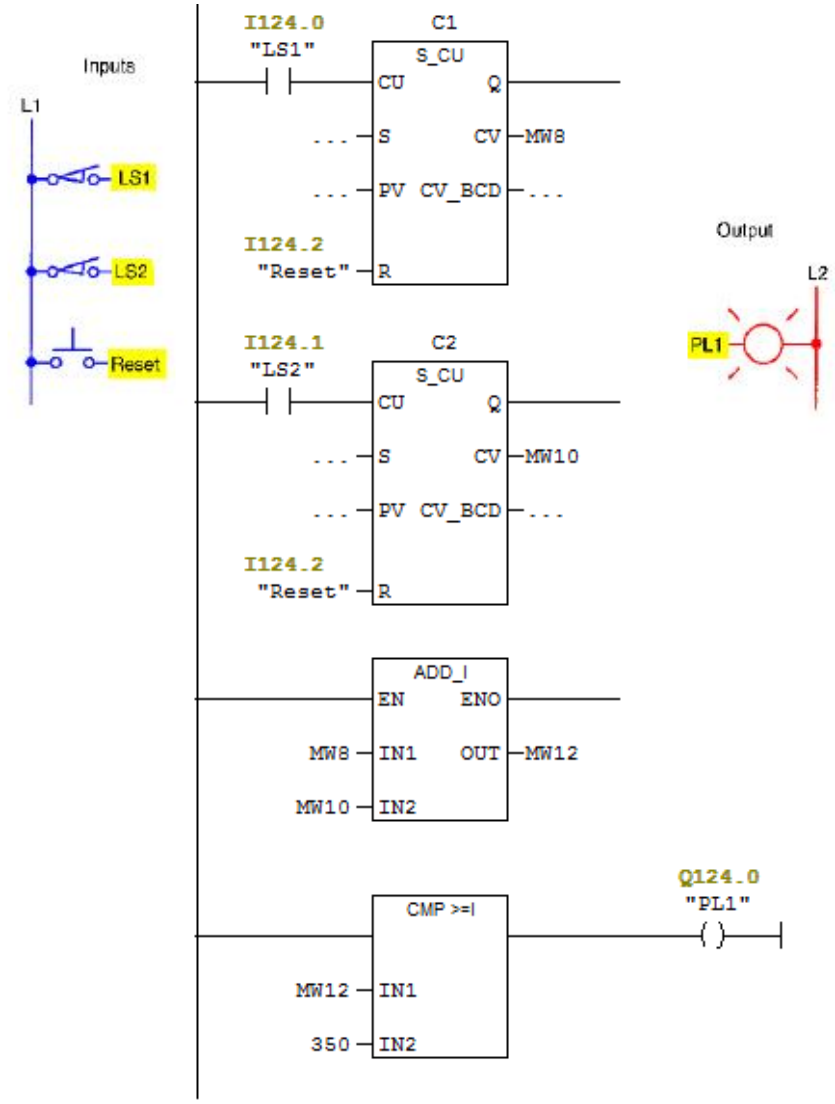
When the rung is true, the value stored at the MW0 address, MW0=(25), is added to the value stored at the MW2 = (50), and the answer (75) is stored at the destination address, MW10.



Counter Program That Uses the ADD Instruction:

The program of the following slide shows how the ADD instruction can be used to add the accumulated counts of two up-counters. This application requires a light to come on when the sum of the counts from the two counters is equal to or greater than 350. Input1 of the ADD instruction is addressed to store the accumulated value of counter C1, while input2 is addressed to store the accumulated value of counter C2.

The value at MW8 is added to the value at MW10 and the result (answer) is stored at destination address MW12. Input 1 of the **GREATER THAN OR EQUAL** instruction is addressed to store the value of the destination address MW12, while Input 2 contains the constant value of 350. Therefore the **GREATER THAN OR EQUAL** instruction will be logic true whenever the accumulated values in the two counters are equal to or greater than the constant value 350. A reset button is provided to reset the accumulated count of both counters to zero.

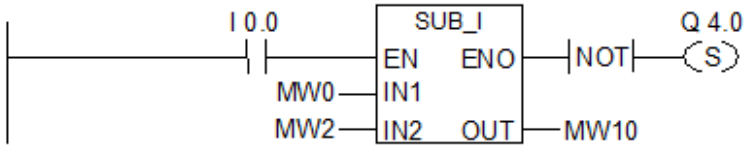


SUBTRACT Instruction:

The **SUBTRACT** instruction is an output instruction that subtracts one value from another and stores the result in the destination address.

SUB_I Subtract Integer

Example

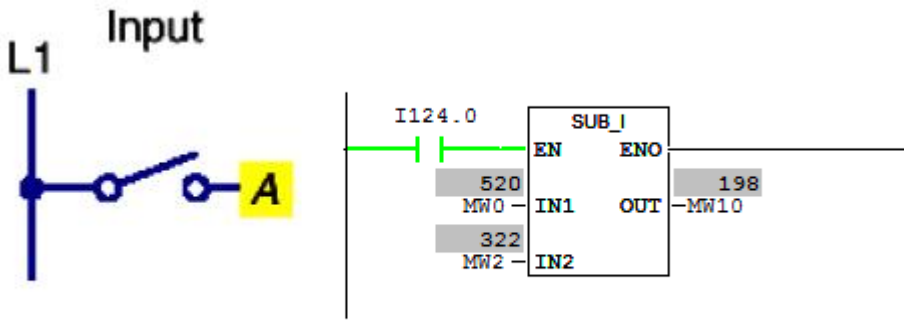


The SUB_I box is activated if I0.0 = "1". The result of the subtraction MW0 - MW2 is output to MW10.

If the result was outside the permissible range for an integer or the signal state of I0.0 = 0, the output Q4.0 is set.

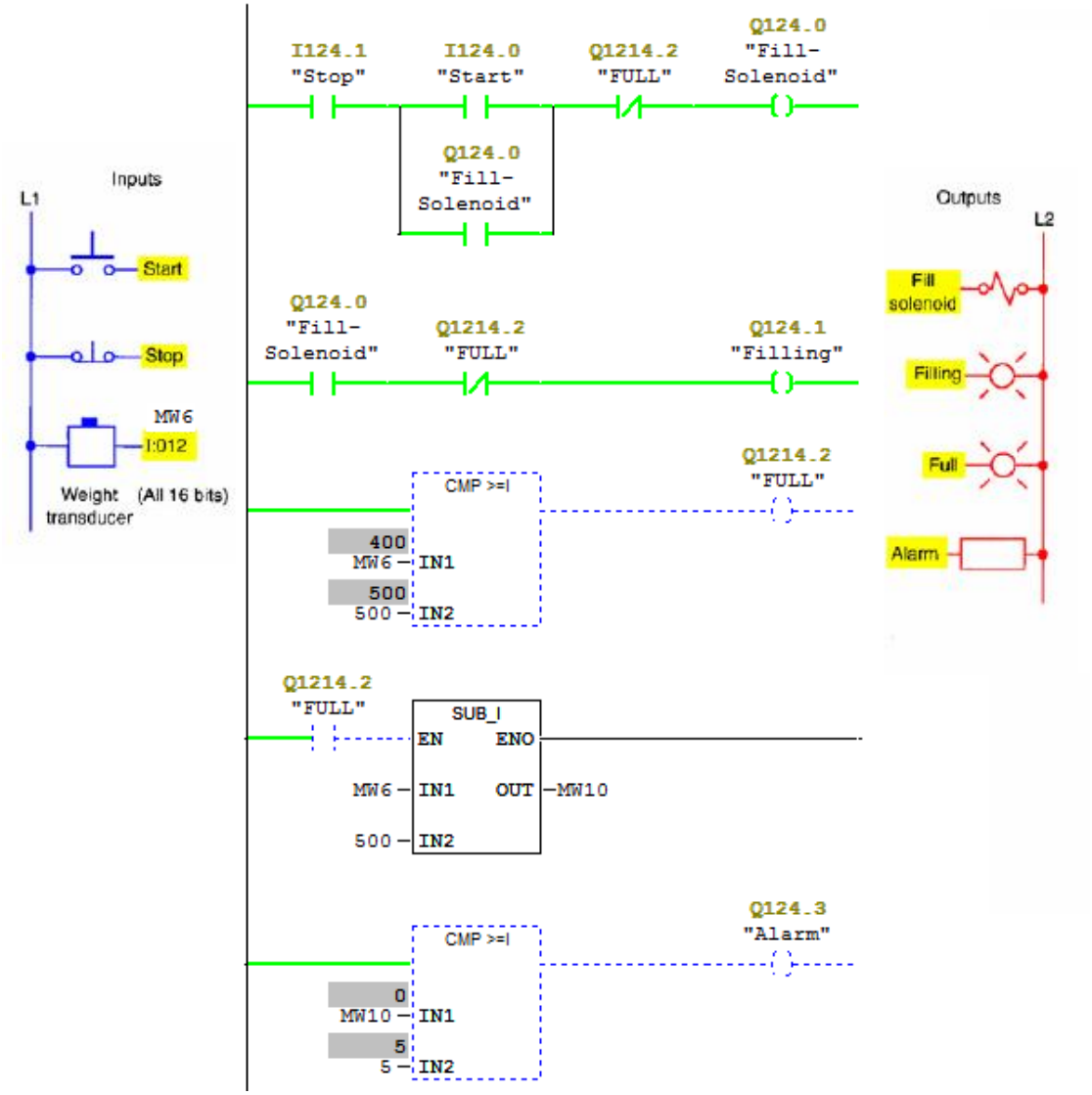
When rung conditions are true, the subtract instruction subtracts IN2 from IN1 and stores the result in the destination.

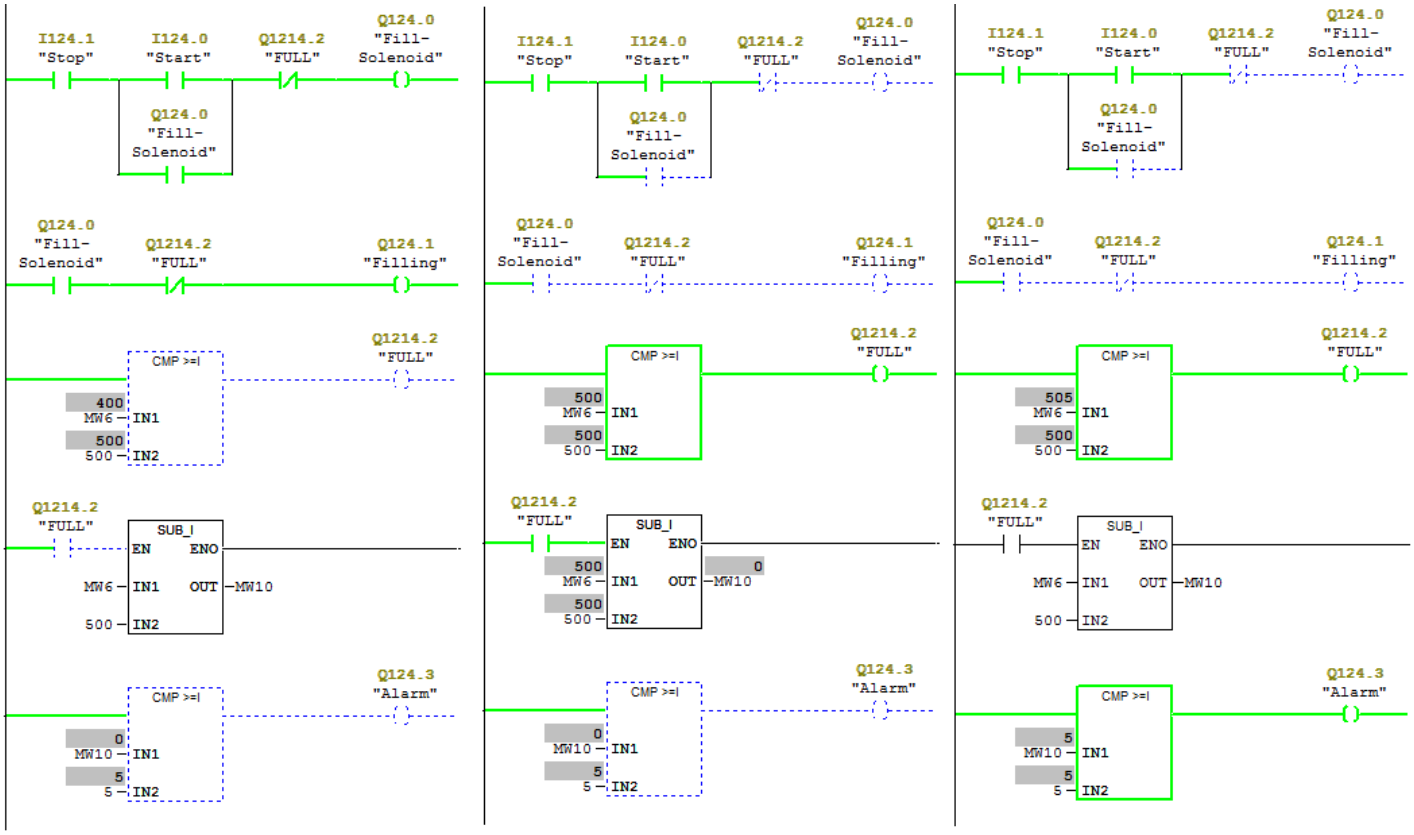
When the rung is true, the value stored at the IN2 address, MW2 (322), is subtracted from the value stored at the IN1 address, MW0 (520), and the answer (198) is stored at the destination address, MW10.



Overfill Alarm Program:

The program of the following side shows how the SUBTRACT function can be used to indicate a vessel overfill condition. This application requires an alarm to sound when a supply system leaks 5 Ib or more of raw material into the vessel after a preset weight of 500 Ib. has been reached. When the start button is pressed, the fill solenoid (rung 1) and filling indicating light (rung 2) are turned on and raw material is allowed to flow into the vessel. The vessel has its weight monitored continuously by the PLC program (rung 3) as it fills. When the weight reaches 500 Ib, the fill solenoid is de-energized and the flow is cut off. At the same time, the filling pilot light indicator is turned off and the full pilot light indicator (rung 3) is turned on. Should the fill solenoid leak 5 Ib or more of raw material into the vessel, the alarm (rung 5) will energize and stay energized until the overflow level is reduced below the 5 Ib overflow limit.



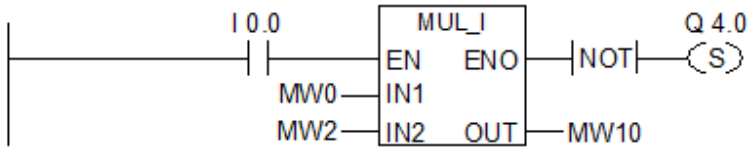

I124.1
"Stop"
I124.0
"Start"
Q1214.2
"FULL"
Q124.0
"Fill-
Solenoid"
Q124.0
"Fill-
Solenoid"
Q124.0
"Fill-
Solenoid"
Q1214.2
"FULL"
Q124.1
"Filling"
CMP >=I
400
MW6
IN1
500
500
IN2
Q1214.2
"FULL"
Q1214.2
"FULL"
SUB_I
EN
ENO
MW6
IN1
OUT
MW10
500
IN2
CMP >=I
0
MW10
IN1
5
5
IN2
Q124.3
"Alarm"
I124.1
"Stop"
I124.0
"Start"
Q1214.2
"FULL"
Q124.0
"Fill-
Solenoid"
Q124.0
"Fill-
Solenoid"
Q124.0
"Fill-
Solenoid"
Q1214.2
"FULL"
Q124.1
"Filling"
CMP >=I
500
MW6
IN1
500
500
IN2
Q1214.2
"FULL"
Q1214.2
"FULL"
SUB_I
EN
ENO
500
MW6
IN1
OUT
0
MW10
500
500
IN2
CMP >=I
0
MW10
IN1
5
5
IN2
Q124.3
"Alarm"
I124.1
"Stop"
I124.0
"Start"
Q1214.2
"FULL"
Q124.0
"Fill-
Solenoid"
Q124.0
"Fill-
Solenoid"
Q124.0
"Fill-
Solenoid"
Q1214.2
"FULL"
Q124.1
"Filling"
CMP >=I
505
MW6
IN1
500
500
IN2
Q1214.2
"FULL"
Q1214.2
"FULL"
SUB_I
EN
ENO
MW6
IN1
OUT
MW10
500
IN2
CMP >=I
5
MW10
IN1
5
5
IN2
Q124.3
"Alarm"

MULTIPLY Instruction:

The **MULTIPLY** instruction is an output instruction that multiplies two values and stores the result in the destination address.

MUL_I Multiply Integer

Example

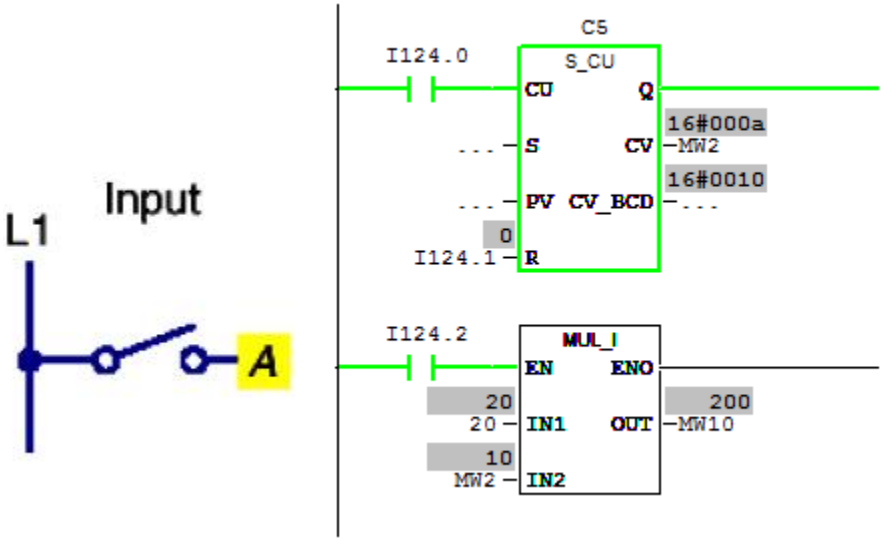


The MUL_I box is activated if I0.0 = "1". The result of the multiplication MW0 x MW2 is output to MD10.

If the result was outside the permissible range for an integer, the output Q4.0 is set.

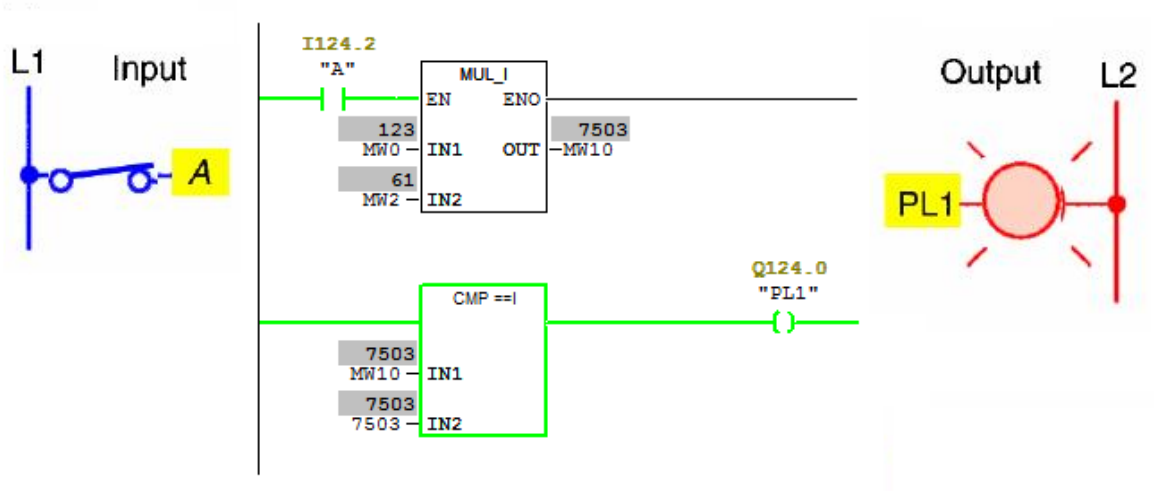
When rung conditions are true, the multiply instruction multiplies IN1 by IN2 and stores the result in the destination “OUT”.

When the rung is true, the data in IN1 (the constant, 20) will be multiplied by the data in IN2 (the accumulated value of counter C5), with the result being placed in the destination MW10



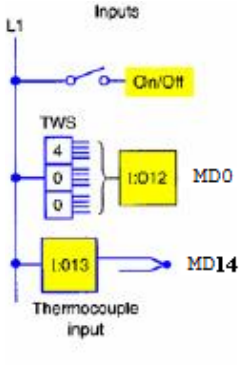
As with previous math instructions, IN1 and IN2 can be values (constants) or addresses that contain values.

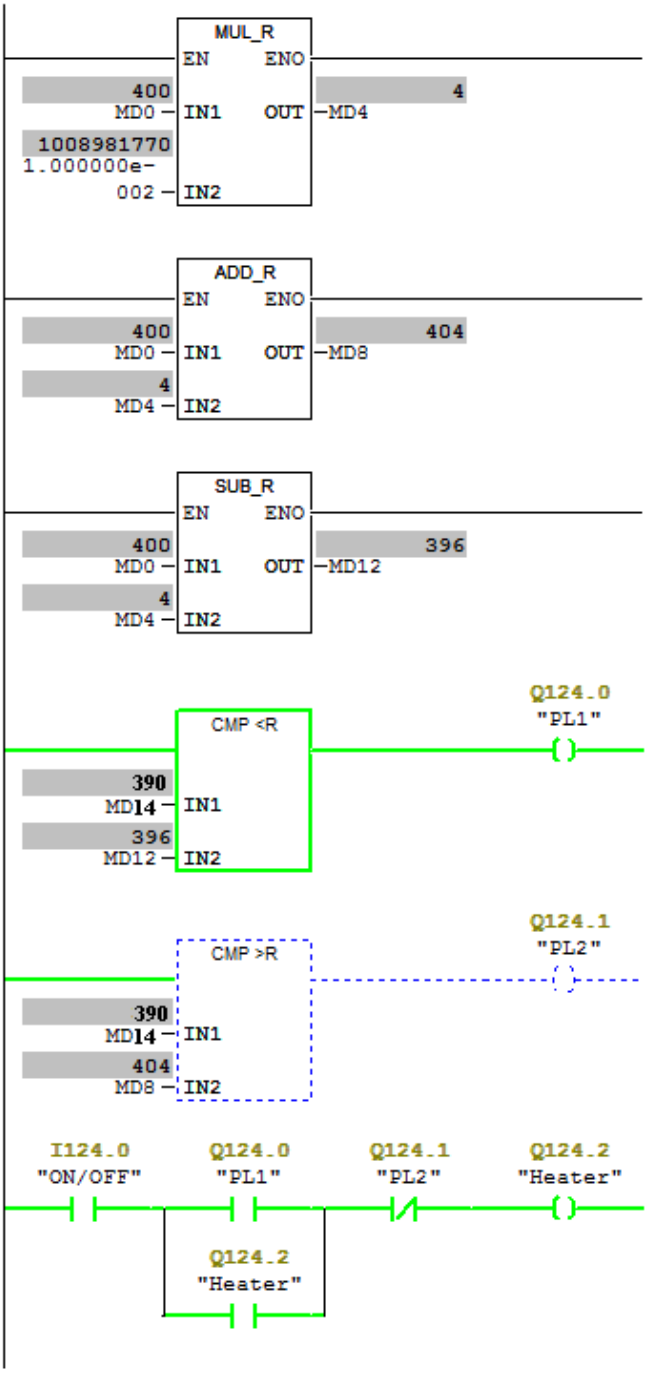
Simple MULTIPLY Program:

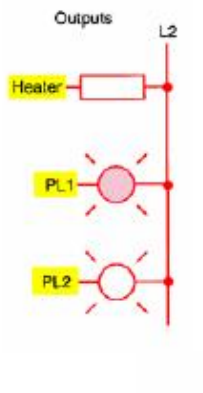


Oven Temperature Control Program:

The program of the following side shows how the **MULTIPLY** instruction is used as part of an oven temperature control program. In this program, the PLC calculates the upper and lower ***dead band*** or off/on limits about the set point. The upper and lower limits are set automatically at ± 1 % regardless of the set-point value. The set-point temperature is adjusted by means of the thumbwheel switch. An analog thermocouple interface module is used to monitor the current temperature of the oven. In this example, the set-point temperature is 400°F. Therefore, the electric heaters will be turned on when the temperature of the oven drops to less than 396°F and stay on until the temperature rises above 404 °F. If the set-point is changed to 100°F, the dead band remains at ± 1%, with the lower limit being 99° of and the upper limit being 101°F. The number stored in word MD4 represents the upper temperature limit, while the number stored in word MD12 represents the lower limit.







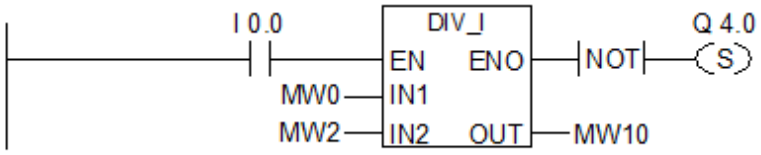
Temperature Control Program

DIVIDE Instruction:

The **DIVIDE** instruction divides the value in IN1 by the value in IN2 and stores the result in the destination OUT and math register.

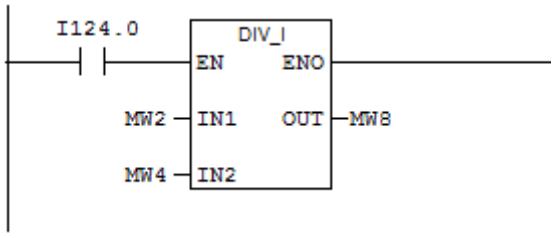
DIV_I Divide Integer

Example

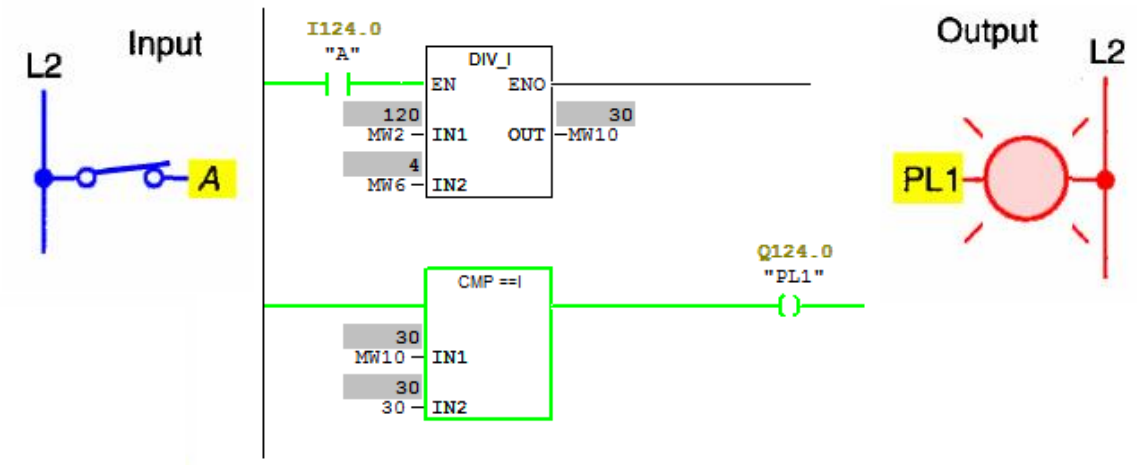


The DIV_I box is activated if I0.0 = "1". The result of the division MW0 by MW2 is output to MW10.
If the result was outside the permissible range for an integer, the output Q4.0 is set.

When the rung is true, the data in MW2 will be divided by the data in MW4, with the result being placed in the destination MW8



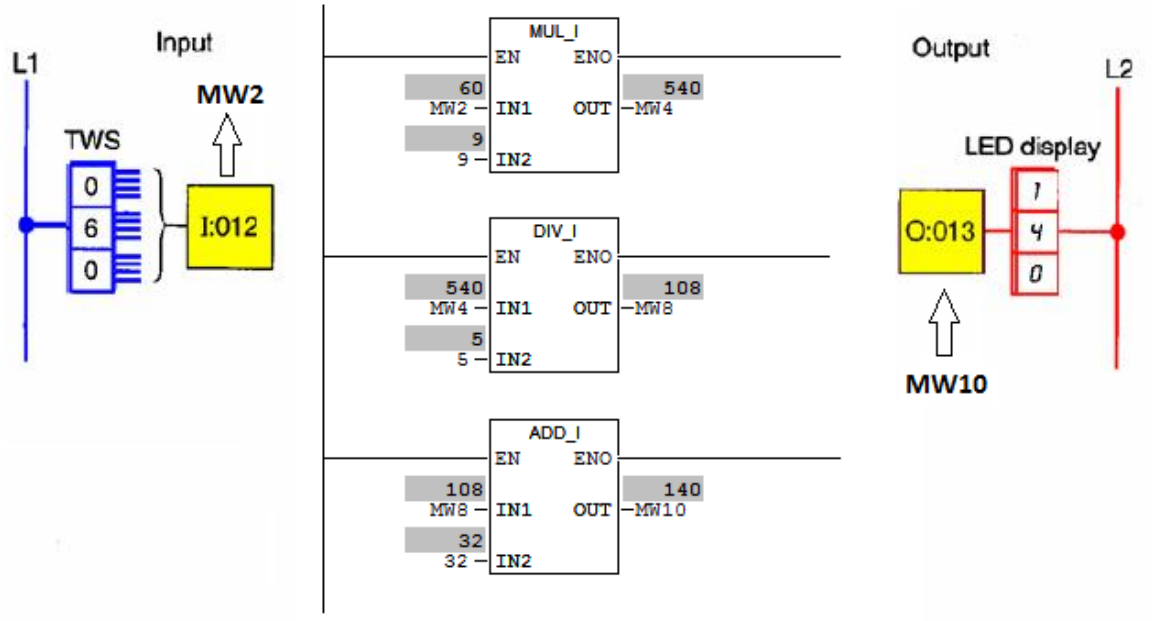
Simple DIVIDE Program:



Converting °C to °F Program:

The program of the following side shows how the **DIVIDE** instruction is used as part of a program to convert Celsius temperature to Fahrenheit. In this application, the thumbwheel switch connected to the input module indicates Celsius temperature. The program is designed to convert the recorded Celsius temperature in the data table to Fahrenheit values for display. The formula: F = (9/5 x C) + 32 forms the basis for the program. In this example, a current temperature reading of 60 °C is assumed. The **MULTIPLY** instruction multiplies the temperature (60°C) by 9 and stores the product (540) in address N7:0. Next, the **DIVIDE i**nstruction divides 5 into the 540 and stores the answer (108) in address N7:1. Finally, the **ADD** instruction adds 32 to the value of 108 and stores the sum (140) in address O:13. Thus 60°C = 140°F.

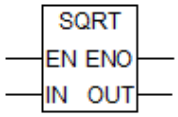
Converting °C to °F Program:



Square Root (SQRT) Instruction:

The **Square Root (SQRT)** instruction is an output instruction that determines the square root of a number.

SQRT Establish the Square Root



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, L, D	Enable input
ENO	BOOL	I, Q, M, L, D	Enable output
IN	REAL	I, Q, M, L, D or constant	Input value: floating-point
OUT	REAL	I, Q, M, L, D	Output value: square root of floating-point number

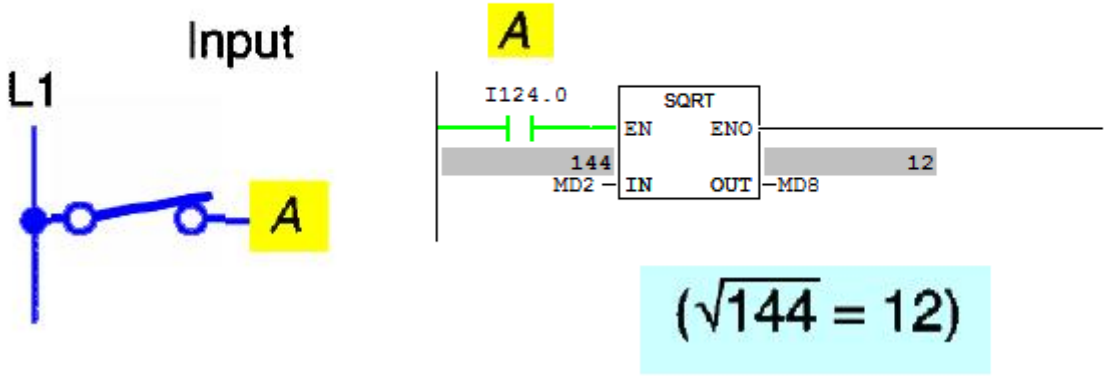
Description

SQRT establishes the square root of a floating-point number. This instruction issues a positive result when the address is greater than "0". Sole exception: the square root of -0 is -0.

When rung conditions are true, the square root instruction calculates the square root of the number stored at IN and places the answer in the OUT.

Square Root (SQR) Instruction:

When the rung is true, the square root of the number in IN, MD2 (144), will be calculated and the answer (12) placed in the destination OUT, MD8.



Negate Instruction (NEG):

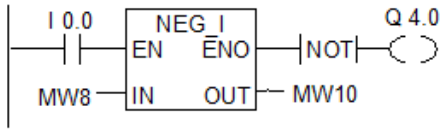
The **Negate (NEG)** instruction is an output instruction that negates (changes the sign of) of a value.

NEG_I Twos Complement Integer

Description

NEG_I (Twos Complement Integer) reads the content of the IN parameter and performs a twos complement instruction. The twos complement instruction is equivalent to multiplication by (-1) and changes the sign (for example: from a positive to a negative value). ENO always has the same signal state as EN with the following exception: if the signal state of EN = 1 and an overflow occurs, the signal state of ENO = 0.

Example



If I0.0 is "1", then the value of MW8 with the opposite sign is output by the OUT parameter to MW10.

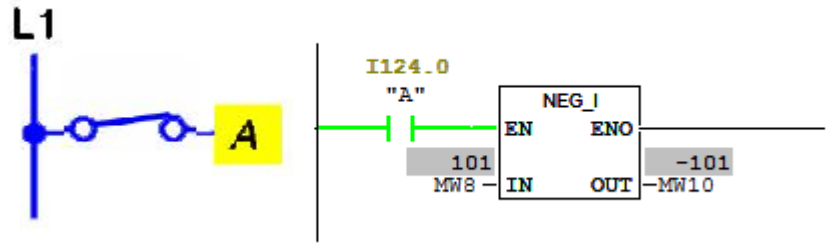
MW8 = + 10 results in MW10 = - 10. The output Q4.0 is "1" if the conversion is not executed (ENO = EN = 0).

If the signal state of EN = 1 and an overflow occurs, the signal state of ENO = 0.

When rung conditions are true, the negate instruction changes the sign of IN and stores the result in the destination OUT.

Negate Instruction (NEG)

When the rung is true, the sign of the number IN, MW8 (101), will be changed and the result (-101) placed in the destination OUT, MW10.

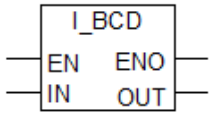


Positive numbers will be stored in straight binary format, and negative numbers will be stored in two's complement.

Convert Integer to BCD Instruction:

The ***convert to*** **I_BCD** output instruction is used to convert 16-bit integers into binary coded decimal (BCD) values.

I_BCD Integer to BCD

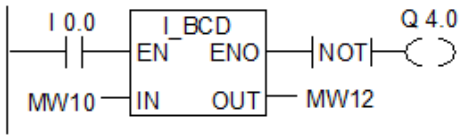


Description

I_BCD (Convert Integer to BCD) reads the content of the IN parameter as an integer value (16-bit) and converts it to a three-digit BCD coded number (+/- 999). The result is output by the parameter OUT.

If an overflow occurred, ENO will be "0".

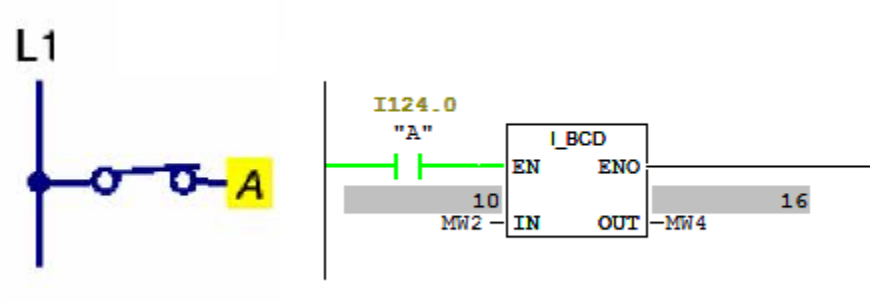
Example



If I0.0 is "1", then the content of MW10 is read as an integer and converted to a three-digit BCD coded number. The result is stored in MW12. The output Q4.0 is "1" if there was an overflow, or the instruction was not executed (I0.0 = 0).

When rung conditions are true, the I_BCD instruction converts the 16-bit integer stored at IN to BCD and places the answer in the destination OUT. This instruction could be used when transferring data from the processor (which stores data in binary format) to an external device, such as an LED display, that functions in BCD format.

When input A is true, the I_BCD instruction will convert the binary bit pattern at the IN address, MW2, into a BCD bit pattern of the same decimal value at the destination address MW4

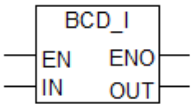


The IN displays the value 10, which is the correct decimal value; however, the destination OUT displays the value 16. Since the processor interprets all bit patterns as binary, the value 16 is the binary interpretation of the BCD bit pattern. The bit pattern for 10 BCD is the same as the bit pattern for 16 binary.

Convert From BCD to Integer Instruction:

The ***convert from*** **BCD_I** output instruction is used to convert binary coded decimal (BCD) values to integer values.

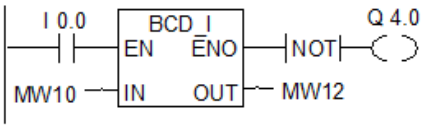
BCD_I BCD to Integer



Description

BCD_I (Convert BCD to Integer) reads the contents of the IN parameter as a three-digit, BCD coded number (+/- 999) and converts it to an integer value (16-bit). The integer result is output by the parameter OUT. ENO always has the same signal state as EN.

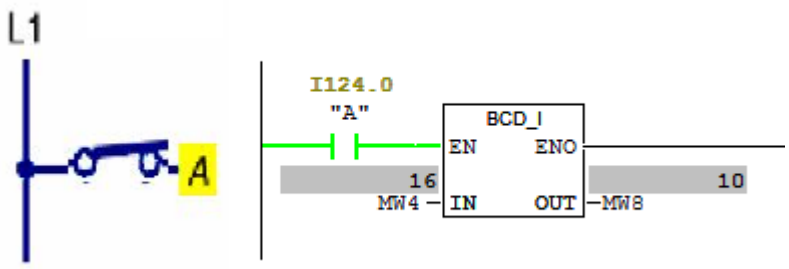
Example



If input I0.0 is "1" , then the content of MW10 is read as a three-digit BCD coded number and converted to an integer. The result is stored in MW12. The output Q4.0 is "1" if the conversion is not executed (ENO = EN = 0).

When rung conditions are true, the BCD_I instruction converts the BCD value to the equivalent integer value and stores the converted value in the destination. This instruction could be used to convert data 'from a BCD external source, such as a BCD thumbwheel switch, to the binary format in which the processor operates.

When input A is true, the BCD_I instruction will convert the BCD bit pattern stored at the source address, MW4, into a binary bit pattern of the same decimal value at the destination address, MW8.



REFRENCES

Text Books:

Programmable Logic Controllers By: Frank D. Petruzella
McGrawHill
ISBN 0-07-829852-0

Lab Manual for Programmable Logic Controllers with LogixPro PLC simulator By: Frank D. Petruzella

SIEMENS Manuals:

Ladder Logic (LAD) for S7-300 S7-400 Programming Manuals

Siemens CTRAIN Documents.