PRACTICAL WORK BOOK

For Academic Session 2014

**Industrial Electronics (EL-383) For T.E(EL)**

---

Name: 

Roll Number: 

Batch: 

Department: 

Year: 

---

Department of Electronic Engineering
N.E.D. University of Engineering & Technology, Karachi –75270 Pakistan
LABORATORY WORK BOOK

FOR THE COURSE

EL-383 INDUSTRIAL ELECTRONICS

Prepared By:

Syed Riaz un Nabi (Assistant Professor)

Reviewed By:

Muhammad Khurram Shaikh (Assistant Professor)

Approved By:

The Board of Studies of Department of Electronic Engineering
# CONTENTS

<table>
<thead>
<tr>
<th>Lab#</th>
<th>Dated</th>
<th>List of Experiments</th>
<th>Page #</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>Introduction to PLC &amp; Simatic S7-300</td>
<td>04</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>PLC – The Way of Operation &amp; Programming Basics</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>Using PLCSIM (PLC Simulator)</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Structured Programming -Programming with Function Blocks</td>
<td>58</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>Integrated Functions of CPU 314IFM</td>
<td>68</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>To understand Project navigation and to understand system configuration</td>
<td>79</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>To understand the logical operation of PLC such as SR latch operation</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>To develop a Function and to use it during application</td>
<td>81</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td><strong>APENDIX I</strong>&lt;br&gt;Notes for the programming of Simatic S7-300 with step 7</td>
<td>82</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td><strong>APENDIX II</strong>&lt;br&gt;Notes for the application of S7- PLCSIM</td>
<td>97</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td><strong>APENDIX III</strong>&lt;br&gt;Notes for structured programming with FCs and FBs</td>
<td>98</td>
<td></td>
</tr>
</tbody>
</table>
1.1 INTRODUCTION TO PLC

Control Engineering has evolved over time. In the past humans were the main methods for controlling a system. More recently electricity has used for control and early electric control was based on relays. These relays allow the power to be switched on and off without a mechanical switch. It is common to use relays to make simple logical control decisions. The development of low cost computers has brought the most recent revolutions, the Programmable Logic Controllers (PLC). The advent of PLCs began in 1970s, and has become the most common choice for the manufacturing controls.

Programmable logic controllers (PLCs) are the control hubs for a wide variety of automated systems and processes. Programmable logic controllers are used extensively in diverse industrial applications ranging from machining to automated assembly. They were designed to replace the necessary sequential relay circuits for machine control.

PLCs have been gaining popularity on the factory floor and will probably remain predominant for some time to come. Other areas of application of PLCs are industrial automation and control of industrial equipment. Most of this is because of the advantages they offer:

- Cost effective & Flexible
- Computational abilities
- Troubleshooting aids
- Reliable components

One may very correctly ask that why not to use a personal computer for these tasks in place a specialized PLC. The answer is very simple; PLC

- Is intended for use on factory floors & in harsh environments
- Is more durable & Less expensive
- Can be placed in remote or rugged industrial locations
- Can perform at a high level for many years.
- Can withstand shock, vibration, humidity, EMI, RFI, dust, mist, and splash
- Can also be used for compiling data coming from many sources and uploading on a computer network
1.2 **PLC SIGNAL SYSTEM**

PLC uses binary number system for digital signaling. 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”.

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.

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.
2 INSIDE A PLC

2.1 THE CPU MODULE

The voltage coming from the sensor signals is switched to the pin board of the input device. In the CPU (central processing unit), the processor works on the program in the memory and queries whether the individual inputs of the equipment voltage have voltage or not. Dependent on this condition at the inputs and on the program in the memory, the processor instructs the output device to switch the voltage on the appropriate terminals of the terminal strip. Depending on the tensile state at the terminals of the output modules, the attached actuators and/or warning light are switched on or off.

CPU of the PLC:
The address counter successively (serially) queries the program memory instruction for instructions and causes the program-dependent information to transfer from the program memory to the instruction register. All memory of a process is made up of registers. The control mechanism receives its instructions from the instruction register. While the control mechanism works on the current instruction, the address counter pushes the next instruction into the instruction register. After the operations follow the status transfer of the inputs into the processor image input table (PAE), the employment of the timers, counters, accumulators and the transmission of the result of logic operation (RLO) in the processor image output table (PAA). After the processing of the user program’s block end (BE), if a module’s end is recognized, then the transmission of the respective status follows from the PAA to the outputs.

The peripheral bus completes data exchange between the CPU and the peripherals. The analog and digital input and output devices, as well as a timer, counter, and comparator module belong to the peripheral bus.

2.2 THE BUS SYSTEM

The bus system is a collecting line for the transmission of signals. Thus the signal exchange is made in the automation equipment between the processor and the input and output by a process bus system. The bus consists of three parallel signal lines:

- The addresses on the individual modules are addressed with the address bus.
- Data will transfer e.g. from input to output devices with the data bus.
- Signals are conveyed with the control bus for the control and monitoring of the execution of functions within the automation equipment.
2.3 THE POWER SUPPLY MODULE

The power supply module produces the voltage for the electronic devices of the automation equipment from network voltage. The height of this voltage amounts to 24 V. Voltages for sensor signals, actuators and warning lights, which lie over 24 V, supply additional voltage for power supply units and/or control transformers.

2.4 PROGRAM MEMORY

Memory elements are elements, in which information can be deposited (stored) in the form of binary signals.

Semiconductor memory is used predominantly as program memory. A memory consists of 512, 1024 or 2048 memory cells. It is usually recommended to indicate the capacity of the program memory (i.e. the number of memory cells) in multiples of 1 K (1 K stands for 1024). An instruction for control can be written (programmed) into each memory cell with the help of a programming device. Each binary cell of a memory cell can accept the signal status "1" or "0".

2.5 RAM

One designates read/write memory built in semiconductor technology with RAM. The individual storage locations are indicated by addresses, and with the help of the memory cells, can be accessed. The information is often arbitrarily written into the memory cells information. The information is picked out, without the information contents being lost. RAM-memory is however volatile memory i.e. their information contents are lost in case of failure of the supply voltage. RAM memory is electrically deleted. The internal main memory of a SIMATIC S7-300 is this type of RAM. The buffer battery, which can be inserted into the PLC, serves as a safety device for this memory.

2.6 FLASH-EPROM

EPROM stands for ERASABLE, PROGRAMMABLE read-only memory. The contents of EPROM are erasable and again programmable by UV light or a voltage. It is well suited to be transported without overflow. In the SIMATIC S7 300 one has the possibility to secure a program on a Memory card (Flash-EPROM) with the PG and to also quickly restore the system after a power failure. This Flash EPROM is recorded and deleted with a voltage of 5V. Thus it could also be erased if a power failure occurred when it was connected with the CPU.
3 AUTOMATION SYSTEM SIMATIC S7

SIMATIC S7 is a product design of the company SIEMENS and is used for the current series of the programmable controllers.

This SIMATIC S7 - computer family is a module in the automation concept for the manufacturing and process technique: Totally Integrated automation.

Automation system SIMATIC S7 of Siemens consists of following series

**SIMATIC S7-200**

Used in open loop & closed loop control tasks in Mechanical Engineering & Plant building

Applications include:
- Plaster & Cement Misers
- Suction Plants
- Wood Working Machinery
- Hydraulic Lifts
- Conveyer Systems
- Food & Drinks Industry
- Gate Controls

**SIMATIC S7-300**

- Modular mini PLC system for low end applications
- Individually combinable modules
- Various CPUs and range of I/O modules
- Many convenient functions

Application include
- Special Purpose Machines
- Textile Machines
- Packaging Machines
- Machine Tools
- Installation Engineering
- Control System

**SIMATIC S7-400**

- Power PLC for medium and upper performance ranges
- Modular, Ruggedness, easy extension
- Extensive communication facilities
Easy implementation of distributed structures

Applications include:

- Automotive Construction
- Process Engineering
- Special Purpose Machines
- Warehouse Equipments

3.1 SIMATIC S7 - 300

Module Spectrum:

The following kinds of modules, together with the CPU, form the structure of a SIMATIC S7-300:

```
<table>
<thead>
<tr>
<th>PS:</th>
<th>IM:</th>
<th>SM:</th>
<th>FM:</th>
<th>CP:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input:</td>
<td>Send</td>
<td>DoDo</td>
<td>Counters</td>
<td>Point to point</td>
</tr>
<tr>
<td>AC 120V/230V</td>
<td>-Receive</td>
<td>-DC 24V</td>
<td>-Rules</td>
<td>-PROFIBUS DP/FMS</td>
</tr>
<tr>
<td>Output:</td>
<td>Send</td>
<td>-AC 120V/230V</td>
<td>-Position</td>
<td></td>
</tr>
<tr>
<td>DC 24V</td>
<td>-Receive</td>
<td>-Relay</td>
<td>-Cam</td>
<td></td>
</tr>
<tr>
<td>- 2A</td>
<td>AV/AO</td>
<td>-Layer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- 5 A</td>
<td>-Voltage</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-10 A</td>
<td>-Current</td>
<td>FM-ATE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-Resistance</td>
<td>-User technology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-Thermocouple</td>
<td></td>
<td>(MS-DOS/C Platform)</td>
<td></td>
</tr>
</tbody>
</table>
```
Important elements of the power supply and CPU:

![Diagram of PLC components](image)

### Status- and error indication to the CPU:

<table>
<thead>
<tr>
<th>Memory bit</th>
<th>Meaning</th>
<th>Clarification</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF (red)</td>
<td>System error</td>
<td>Diagnostic modules indicate a system error.</td>
</tr>
<tr>
<td>BAF (red)</td>
<td>Battery low</td>
<td>Announcement if the buffer battery does not contain enough voltage.</td>
</tr>
<tr>
<td>DC5V (green)</td>
<td>DC5V-supply for the CPU and rear wall bus</td>
<td>Announcement for the functional internal 5V supply of the CPU.</td>
</tr>
<tr>
<td>FRCE (yellow)</td>
<td>Force</td>
<td>Announcement for the condition of the CPU, in which inputs and outputs are force-actuated by a debug function.</td>
</tr>
<tr>
<td>RUN (green)</td>
<td>Operation state RUN</td>
<td>Flashes by the running of the CPU - static announcement when the CPU is in the RUN state.</td>
</tr>
<tr>
<td>STOP (yellow)</td>
<td>Operation state STOP</td>
<td>Flashes when a memory reset is requested – static announcement, when the CPU is in the STOP state.</td>
</tr>
</tbody>
</table>
Protection concept of the CPU:

Each CPU possesses a code switch for the switching of operation modes. The following operation modes are possible:

- **RUN-P**: Program runs; All PG functions are allowed.
- **RUN**: Program runs; Only read PG functions are allowed.
- **STOP**: Program does not run; All PG functions are allowed.
- **MRES**: With this position, one can accomplish a reset as described.

The protection concept for SIMATIC S7-300 makes it possible to protect certain parts of the automation system against unauthorized access. These are:

- The CPU and all programmable modules
- All objects (like e.g. blocks)

These protected parts can be affected by the PG or by B&B devices.

The CPU protection concept is divided into three protection stages. These protection stages determine what is permitted to a user.

- **Stage 1** – Code switch position Run-P or Stop: no protection, all functions are allowed.
- **Stage 2** – Code switch position Run: write protection, reading functions is allowed, e.g. observation functions, information functions, compiling from the CPU.
- **Stage 3** – Parameter lock over S7-Configuration (Password protection). Only observation and information functions are allowed, if the user does not know the password.

The memory resetting of the CPU can also be accomplished with the code switch as follows:

<table>
<thead>
<tr>
<th>Step</th>
<th>Execution</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Turn the key to the <strong>STOP</strong> position.</td>
<td>STOP indication is shown.</td>
</tr>
<tr>
<td>2</td>
<td>Turn the key to the <strong>MRES</strong> position and hold it in this position (approx. 3 Seconds) until the STOP-memory bit is shown.</td>
<td>The STOP-memory bit expires and after approx. 3 seconds, it will be shown again. With new CPUs, wait until the STOP-Memory bit lights up for the second time. <strong>Important:</strong> Between step 2 and step 3 should a maximum of 3 seconds go by?</td>
</tr>
<tr>
<td>3</td>
<td>Turn the key back to the <strong>STOP</strong> position and within the following 2 seconds restart in the <strong>MRES</strong> position.</td>
<td>The STOP-Indication blinks for approx. 3 seconds and then lights up again normally: When everything is ok. The CPU is reset.</td>
</tr>
</tbody>
</table>
4 **Basic Principle of operation of PLC**

Programmable logic controllers function by interpreting data coming through their inputs and depending upon their state, turning on/off their outputs. They are programmable via software interfaced via standard computer interfaces and proprietary languages and network options. PLCs function using relay ladder logic programming. Unlike traditional programming in which command functions are read, interpreted, and performed in sequential order, ladder logic allows programmable logic controllers to perform any command within the loop at any time without executing previous commands. This allows PLCs to carry on their three basic functions: Control, Input, and Output - as needed.

In addition to controlling output functions, programmable logic controllers are good for compiling data from many sources and uploading this data into a computer network.

5 **Wiring Diagram – Ladder programming**

Ladder programming, one of the programming languages of PLC, is the easiest way of programming a PLC. It is reflection of the wiring diagrams of some control circuits. To get an easy understanding of the Ladder one should have an idea of what wiring diagrams are and how can they be made for some control task. Wiring diagrams

- Provide information about Industrial Control Circuitry
- Serve two main purposes
- Are a source of written communication
- Serves as trouble shooting guide

**Relay Ladder Logic Diagram**

The wiring circuits used in electrical control circuits are referred to as **Relay Ladder Logic Diagram**.

- The term Ladder is derived from the appearance of the diagram.
- The term Logic is derived from the decision making function that is performed by relays

**Example**

- **L1**
- **L2**

Two vertical Lines L1 and L2 represent potential difference. They are referred to as **Rails**.
Various components are located on horizontal lines. Input components are located on the Left portion of the Rung and Output on the Right

**Elements of Ladder Circuit**
- Power Source
- Input Control
- Device Load
- Device
- Interconnection Wires

**Components used as Input Devices.**
- Push
- Buttons
- Selection
- Switches
- Limit
- Switches
- Flow Switches
- Level Switches
- Temperature Switches

**Components used as Output Devices**
- Direct Device
- Lamps
- Actuators

- Indirect Devices
- Relays
- Contactors

**Building a Ladder Diagram**
The function of pumping station is to pump water from storage tank to pressure tank.
When the water level in the pressure tank is too low, the operator must keep the switch depressed to make the pump run until the tank is full. The operator then releases the push button to stop the flow of water into the pressure tank.

**Modification 1**

The circuit is modified so that the operator is not required to keep the button pressed. Here in the circuit the motor pump performs two functions. It fills the pressure tank and it also has a coil that keeps the pump on.

**Modification 2**

Installing a Float switch near the top of the pressure tank the operator is not required to stop the pump when the tank is full. When the water level reaches the float switch FS1 it’s NC connections open thus stops the pump.
Modification 3
Installing a Float switch near the bottom of the pressure tank the pump will automatically start on when the water level in the tank reaches a predetermined low level.

Modification 4
In order to protect the pump a float switch is installed in the Storage Tank FS3. If the water level is extreme low its contacts will be open and the pump will not operate.

PLC Operation
There are three steps of operation of PLC:
- Detect incoming data
- Process it
- Generate outputs to control various devices

PLC achieves machine control as a result of
- program written into its memory
- the power of it’s CPU

Program execution by the CPU takes place in 3 steps. It is called Process Scan Cycle. That is:
- Scan input image table (PII)
Inputs are scanned and **INPUT IMAGE TABLE** is updated.

- Scan program
  - CPU scan program Instructions sequentially.
- Update output image table (PIQ)
  - CPU updates the Output Terminals.

Self diagnostic check is performed by the PLC system

- On switching power on
- Any fault is indicated
- Then the PLC starts the scanning operation

### 7  SIEMENS PLC REVISITED

#### 7.1  Simatic S7-300 Components

- **CPU 314 IFM**
Simulator

1. System jack
2. Power supply
3. Emergency brake
4. Power unit

Input Modules

1. 8 input jacks
2. 8 LED's
3. 8 switches

Switch positions:
- left: switches +24V to input channel
- isolates input jack from input channel
- centre: isolates +24V from input channel
- switches input jack to input channel
- right: momentary-contact function, otherwise as for "left"
Output Modules

Analog I/O
Technical specifications

### Voltages, Currents

<table>
<thead>
<tr>
<th>Power supply</th>
<th>24V DC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Permissible range</td>
<td>20.4 to 28.8 V</td>
</tr>
<tr>
<td>Current consumption</td>
<td>typical 1.0 A</td>
</tr>
<tr>
<td>Inrush current</td>
<td>typical 8A</td>
</tr>
<tr>
<td>External fusing for</td>
<td>Circuit breaker; 2 A</td>
</tr>
<tr>
<td>supply lines</td>
<td>Type B or C</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Integrated inputs/outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addresses of integral</td>
</tr>
<tr>
<td>Digital outputs</td>
</tr>
<tr>
<td>Analog inputs</td>
</tr>
<tr>
<td>Analog outputs</td>
</tr>
</tbody>
</table>

8  USING STEP 7

Ladder logic basics elements

--- | |--- Normally Open Contact (Address)
--- | / |--- Normally Closed Contact (Address)
--- ( ) Output Coil
---[NOT]--- Invert Power Flow
---(S) Set Coil
---(R) Reset Coil
LOGIC DESIGNING USING LADDER LOGIC

- **OR Logic**

- **AND Logic**

- **XOR Logic**

9  **EXERCISES / OBSERVATIONS**

- List all ladder logic programs made during the lab session. (Either draw or attach a printout)
- State Following
  - What is a PLC
• Automation System Simatic S7
• Features of S7-300 PLC with CPU 314IFM
  CPU
  Simulator
• Step 7
1 LEARNING GOAL
In this Lab session, the student will learn about the way of operation of a PLC and basics of programming in a programmable logic controller (PLC) with the programming tool STEP 7.

2.1 INSIDE S7-300
Simatic S7 PLC consists
- CPU
- Buffer battery
- S7 bus
- Inputs and Outputs

2.1.1 CPU
The CPU consists of memory (containing the user program) and the control unit (processor of the user program).

The program can be written into memory by programmer. On CPU there can ROM, EPROM
Plug in memory modules can be
- RAM
- EPROM
- EEPROM

Control unit
- Calls statements in memory one by one
- Executes them
- This is done as
  - Read PII (Process Image of Inputs)
  - Process user program
  - Write PIQ (Process Image of Outputs)

These three steps make up a program cycle and duration of one program cycle is called cycle time which depends upon execution time of individual statements in the program.

This operation of PLC can be best described by the following diagram
In CPU two different programs are always executed

- Operating System (OS)
- User Program (UP)

**Operating System (OS) handles**

- Restart, Updating PII & Outputting PIQ
- Calling UP, detecting & calling interrupts
- Dealing with errors & managing memory
- Communication with peripheral devices

**User Program (UP) is created by user & downloaded into CPU. It handles**

- Specifying condition for restart
- Processing data
- Specifying reaction to interrupts

### 2.1.2 Buffer Battery

This is a back up battery that is used to store the data in case of power failure. With the back up battery all the user data can be saved.

### 2.1.3 S7 Bus

It is the bus on which all the input and output modules are connected.

### 2.1.4 Inputs and Outputs

A wide range of input and output modules can be connected with the S7 PLC. The PLC available in the laboratory has following I/O modules:

- 20 Digital Inputs
- 16 Digital Outputs
2.2 S7 PROGRAMMING LANGUAGES

There are three programming languages in S7 system can be programmed:
- STL (Statement List)
- LAD (Ladder)
- FBD (Function Block Diagram)

STL is pure listing of program. It requires the programmer to memorize the instructions. FBD & LAD, whereas, are graphical programming languages. For both these programmer can simply drag & drop the programming elements in program from catalog of program elements.

Following diagram can very easily distinguish and describe the three programming languages:

Program elements for Ladder programming were explained in lab 1 as Ladder logic program elements. Program elements for Function block diagram are also very easy to understand since the function they perform is displayed as a symbol in each block. Whereas, statement format for the Statement List is as under:

3 PROGRAMMING
Two types of programming are possible in PLCs

- Linear Programming
- Structured Programming

3.1 Linear programming

In linear programming, a set of instruction is cyclically processed. That is, after the last instruction has been processed the first is processed again. All the instructions in that set are processed sequentially.

3.2 Structured programming
4. LAB EXERCISE - 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.

4.1 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.

4.2 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.

4.3 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.
4.4 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).

4.5 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.

4.6 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:

4.7 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 a contact-afflicted circuit, this can be realized only with normally open and closed contacts.

Caution: The exclusive-OR-Operation should only be used with exactly two inputs.
4.8 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.

![Diagram of FBD, STL, and LAD for the query of outputs Q 1.0 and Q 1.1.]

4.9 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.

7.9.1 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.*

### 4.9.2 SET DOMINANT

In accordance with section 4.10.1, the exit Q 2.1 in this example is set with priority.

### 4.10 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.

#### 4.10.1 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.

**4.10.2 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.
4.11 TIMER FUNCTIONS

For the realization of control tasks, different timer functions must be frequently used. The timer functions are integrated in the CPU of the automation equipment. The setting of the desired running time and the starting of the timer function must be made by the user program. The SIMATIC - Automation devices place a certain number of timer elements (CPU dependent) with different timer functions at one’s disposal. A 16-bit-word is assigned to each of the time elements.

The following functions can be programmed with a timer:

4.11.1 RELEASE TIMER (FR) ONLY IN STL

A positive edge change (from “0” to “1”) in the operation result of the release timer operation (FR) will release a timer.

For starting or for the normal function of a timer, the release is not needed. The release is used only in order to re-trigger a current time i.e. to let it start again. This restart is possible only if the starting operation is edited further with the RLO ‘1’.

4.11.2 START TIMER (SI/SE/SD/SS/SF)

With a signal change at the start input (positive edge), the timer is started. In order to start a timer, you must insert three operations in its STL program:

- **Query of a signal status**
- **Load a starting time into ACCU 1**
- **Start operations (alternatively SI, SE, SD, SS or SF)**

\[\text{e.g.:} \]

- A 10.0
- L S5T#2S
- SE T5
4.11.3 TIMER VALUE (TV)

A timer should always execute for a certain time. The length of time value TV can be assigned either as a pre-defined constant in the program or can be given as data to an input word IW, to an output word QW, to a data item DBW/DIW, to a local word LW or to a memory bit word MW. Updating the time decreases the current value in each case by a unit in an interval, which was specified by the time base.

You can load a pre-defined current value with the following syntax:

- **L W#16#abcd**
  - with: a = binary coded time base (e.g. time interval or representation unit; see below)
  - bcd = time value in BCD-Format

- **L S5T#aH_bbM_ccS_dddMS**
  - with: a = hours, bb = minutes, cc = seconds and ddd = Milliseconds
  - The time basis is selected automatically

**Time base:**
The time base defines the interval, in which the time is decreased by a unit. Values with no exact multiple of the time interval are cut off. Values, whose representation unit for the desired range is too large, are rounded off.

<table>
<thead>
<tr>
<th>Time basis</th>
<th>Binary code</th>
<th>Time length</th>
</tr>
</thead>
<tbody>
<tr>
<td>10ms</td>
<td>00</td>
<td>10MS to 9S_990MS</td>
</tr>
<tr>
<td>100ms</td>
<td>01</td>
<td>100MS to 1M_39S_900MS</td>
</tr>
<tr>
<td>1s</td>
<td>10</td>
<td>1S to 16M_39S</td>
</tr>
<tr>
<td>10s</td>
<td>11</td>
<td>10S to 2H_46M_30S</td>
</tr>
</tbody>
</table>

4.11.4 RESET TIMER (R)

A signal at the reset input terminates the processing of the timer. The current time is deleted and the output Q of the time cell is reset.

4.11.5 LOAD TIMER (L/LC)

A time is stored in a binary coded time word. The value in the word can be loaded as a dual number (DUAL) or as a BCD number (DEC) into the ACCU and be transferred from there into other operands - ranges. With STL programming, you have the choice between L T1 for the query of the dual number and LC T1 for the query of the BCD number.

4.11.6 QUERY SIGNAL STATE OF TIMER (Q)

A timer can be queried on its signal status of ("0" or "1"). Signal statuses can be queried - with A T1, AN T1, ON T1, etc.... and can later be used for further logical operations.

You can select five different timers:
4.11.7  PULSE TIMER (SI)

The output of a timer, which is started as a pulse, is fed a signal status 1 after starting. (1). The output is reset, if the programmed length of time has elapsed (2), if the starting signal is reset to zero (3) or if at the reset input of the timer, a signal status 1 is applied (4). A positive edge change (of "0" to "1") in the logical operation result of the operation release (FR), which starts the time again (5). This restart is possible only if the starting operation is edited further with the RLO ‘1’.
4.11.8 EXTENDED PULSE TIMER (SE)

The output of a timer, which is started as an extended pulse, is fed a signal status 1 after starting (1). The output is released, if the given length of time has elapsed (2) or if the resetting input of the timer function is switched on (5).

When the time runs, switching the start input off does not cause the output to reset (locking) (3).

Step - while the time still runs - a renewed signal changes on 1 at the start input and the timer is again started (re-triggered) (4).
**4.11.9 ON-DELAY TIMER (SD)**

The output of a timer, which is started as a signal delay, is fed a signal status 1 after starting only if the programmed time has elapsed and the RLO 1 is applied at the start input (1). The switching on of the start input also causes a switching on of the output Q in the given length of time. The output will reset, if the start input is switched off (2) or if a signal status of 1 is applied at the reset input of the timer (3). The output Q is not switched on if during time running, the start input is switched off or a signal status of 1 at the reset input of the timer is closed.
### 4.11.10 RETENTIVE ON-DELAY TIMER (SS)

The output of a timer, which is started as a retentive ON delay, is fed a signal status 1 after starting only if the programmed time has elapsed (1). The function no longer requires an RLO 1 after starting at the start input, thus it cannot be switched off (locking) (3).

The output is reset only if the reset input of the timer function is switched on (2). As long as the time is running, a switching off and renewed switching on of the start input causes the timer function to become once again started (re-triggered) (4).
4.11.11 OFF-DELAY TIMER (SF)

With a signal change (positive edge) at the start input of a timer which is started as a switched off delay, the output Q of the timer function is switched on (1). If the start input is switched off, the output is still supplied with the signal status 1 until the programmed time has elapsed (2). Switching the start input (negative edge) around the given length of time, causes the switching of the input off. The output of the timer is also switched off, if at the reset input, the signal status 1 is applied (4). While the time runs, the renewed switching on of the time function causes the execution time to be stopped and then again started only by the next switching off of the start input (3).

If both the start input and the reset input of the timer function are fed a signal status 1, the exit of the timer will only be set if the dominate reset is switched off (5).
4.12 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:

4.12.1 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”.

4.12.2 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!)
4.12.3 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!)*

4.12.4 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.

4.12.5 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 $C\#5, 2\#...etc.$

4.12.6 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.

4.12.7 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.

4.12.8 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

STL

LAD

Signal state chart:
4.13 LOAD-AND TRANSFER OPERATIONS (L/T) ONLY IN STL

In the programming language STEP 7, load and transfer operations make byte -, word -, and/or the double-word orientated exchange of information between input and output modules, the process-image of the input and output, the timer, the counter, and memory bit storage as well as data blocks possible. This information exchange is not made directly, but always by the accumulator 1 (ACCU 1). The ACCU 1 is a register in the processor and serves as a buffer.

The information flow is directed as follows:

LOAD: from the source memory into the ACCU 1

TRANSFERRING: of the ACCU into the target memory

While the loading contents of the addressed source memory are copied and written into the ACCU 1, the previous ACCU content is transferred into the ACCU 2. When transferring, the contents of ACCU 1 are copied and written into the addressed target memory.

Since the accumulator content was only copied, it is available for further transfer operations.

STL:

```
: L   IW 0
: T   QW 4
: L   +5
: T   QW 6
: BE
```

*1: Process-image of the input area  *2: Process-image of the output area

Load and transferring are absolute operations, which are implemented independently of the logical operations result in each cyclic circulation.

4.14 COMPARISON FUNCTIONS

The programming language STEP 7 offers the possibility of comparing two numerical values directly and advancing the result of the comparison (RLO) immediately. A condition for it is that both numbers have the same number format. The following pairs of numerical values can be compared:

- two integers (16 Bit, Symbol: I)
- two integers (32 Bit, Symbol: D)
- two real numbers (Floating point numbers, 32 Bit, Symbol: R)

There are 6 different comparison operations to choose from:
With the comparison functions, two values which lie in the ACCUs 1 and 2 are compared directly with each other. With the first load operation, the first operand (e.g. IW 0) is loaded into ACCU 1. With the second load operation, first the first operand is reloaded by the ACCU 1 into ACCU 2 and then the second operand (e.g. IW 2) is loaded into ACCU 1. Afterwards the numerical values in the arithmetic block in both accumulators are compared with one another bit by bit. The result of the comparison is binary. If the desired comparison is satisfied, the operation result becomes 1. If the desired comparison is not satisfied, then the RLO becomes 0.

4.15 PROGRAM ORGANIZATION
4.15.1 BLOCK CALL (CALL)

With the module call CALL, you can call functions (FCs) and functional blocks (FBs) as well as system functions (SFCs) and system function blocks (SFBs). At the same time parameters can be transferred and/or variables described as well as opened in the FB or SFB associated local data blocks (See: further reference function “Variable declaration in code blocks”). If no variables are defined in the called block, then this operation corresponds to the operation UC.

**STL**

```plaintext```

```
CALL FB1, DB20
IN := IW 1
OUT :=
TEST :=
```

LAD/FBD
4.15.2 CONDITIONAL CALL (CC)
With the block call CC you can call functions (FCs) and functional blocks (FBs) as well as system functions (SFCs) and system function blocks (SFBs). However, they cannot transfer any parameters and/or describe variables. The call is implemented only if the logical operation result amounts to a “1”.

4.15.3 UNCONDITIONAL CALL (UC)
With the module call UC, you can call functions (FCs) and functional blocks (FBs) as well as system functions (SFCs) and system function blocks (SFBs). They can transfer however no parameters and/or describe variables. The call is implemented independently from the logical operation result.

4.15.4 OPEN A DATA BLOCK (OPN)
With the operation open data block (OPN), you can open a data block (DB) or instance-data block (DI), in order to access the contained data (e.g. with load and transfer operations).

4.15.5 BLOCK END CONDITIONAL (BEC) ONLY IN STL
Depending on the logical operation result, this operation terminates the processing of the current block and jumps back into the block that was previously called. This operation occurs only if the logical operation result amounts to “1”.

4.15.6 **BLOCK END UNCONDITIONAL (BEU) ONLY IN STL**

This operation terminates the processing of the current block and jumps back into the previous block. This operation occurs independently from the logical operation result.

4.16 **JUMP OPERATIONS**

4.16.1 **JUMP UNCONDITIONAL (JU)**

The operation JU interrupts the normal execution of the program and jumps to the branch label indicated in the operand. The jump occurs independently from the logical operation result.

4.16.2 **JUMP IF RLO=1/RLO=0 (JC/JCN)**

The conditioned jump operations interrupt the normal execution of the program and initiate a jump to the branch label indicated in the operand. The jump takes place as a function of the logical operation result. The following conditioned jump operations can be implemented:

- **JC**: Jump when \( RLO = 1 \)
- **JCN**: Jump when \( RLO = 0 \)
4.16.3 **LOOP (LOOP) ONLY IN STL**

With a program loop (LOOP), you can edit a program section several times. In addition you must load a constant into the low order word from the ACCU 1. This number is then decreased by ‘1’ by the operation LOOP. Afterwards the value of this number is examined for $< 0$. If it does not amount to ‘0’, then a jump is implemented to the label of the operation LOOP; otherwise the next operation is implemented.

```
L 5
NEXT:T MB 10

L MB 10
LOOP NEXT
```
Using PLCSIM (PLC Simulator)

1. LEARNING GOAL

In this Lab session, the student will learn about the debugging of a STEP 7-Program with the simulation software S7-PLCSIM.

2. LAB EXERCISE

2.1 GENERATION OF A SIMPLE STEP7-PROGRAM

The program which can be debugged is generated with STEP 7. The example shown here turns a lamp (H1) off with an input-button (S1) and an output-button (S2).

Assignment list:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>I 0.1</td>
<td>S1</td>
<td>Input-button</td>
</tr>
<tr>
<td>I 0.1</td>
<td>S2</td>
<td>Output-button</td>
</tr>
<tr>
<td>Q 4.0</td>
<td>H1</td>
<td>Lamp</td>
</tr>
</tbody>
</table>

The user must implement the following steps, in order to provide a project, in which the solution program can be written.

1. The main tool in STEP 7 is the SIMATIC Manager, which can be opened with a double click on the icon (→ SIMATIC Manager).

2. STEP 7- Programs are managed in projects. Each project can be newly created (→ File → New).
3. Give the project the **Name PLCSIM_1** (→ PLCSIM_1 → OK).
4. Insert a new **S7-Program** into the project **PLCSIM_1**. (→ PLCSIM_1 → Insert → Program → S7-Program).
5. In the SIMATIC Manager, select the OB1 block and accept the options with **OK** (→OK).

6. Accept the options of the OB1 block with **OK** (→OK).
7. Now a simple program can be written in OB1 to e.g. the statement list (STL). This program must then be saved and the OB1 must be closed with X (→ Save X).
2.2 START AND CONFIGURATION OF S7-PLCSIM

If this program is to be debugged without a connection between a PC and a hardware PLC, the simulation must be activated. Then all the accesses that are implemented on the interface of the hardware PLC will be simulated internally in the S7-PLCSIM.

8. In order to start a PLCSIM, click on the Simulator button \( \text{Simulator button} \) (\( \rightarrow \) Simulation on/off \( \text{Simulation on/off} \) ).
9. Now an input and output need to be placed in the program in order to debug it. This is done by calling **Insert** and selecting **Input** and **Output**. **BIT MEMORY** and **Counters** can also be inserted (→ Insert → Input → Insert → Output).

10. The desired addresses **IB0** and **QB4**, and the demonstration method **Bits** must be chosen here (→ IB0 → Bits → QB4 → Bits).
2.3 DEBUGGING OF THE STEP7- PROGRAM WITH S7-PLCSIM

The STEP7- Program to be debugged can now be loaded into the PLC simulator. For this example, only OB1 will be debugged. In addition, SDBs (System function blocks), FBs, FCs and DBs can also be downloaded.

11. Highlight OB1, and click Download \( \rightarrow \) OB1 \( \rightarrow \) Download \( \Rightarrow \).
12. Now switch the simulated PLC to **RUN** and switch the individual input bits with the mouse when needed. The active outputs appear similar to switched inputs, but include a check mark \( ∨ \), which means that they are active(\(→ \text{RUN} → ∨\)).

3 EXERCISES / OBSERVATIONS

- List all ladder logic programs made during the lab session. (Either draw or attach a printout)
- List the method of using PLCSIM. (take any example for your convenience)
1 LEARNING GOAL
In this Lab session, the student should learn how a function block with internal variables is generated for structured programming.

- Generating a function block
- Defining internal variables and programming internal variables in a function block
- Calling and parameterizing of a function block in OB1

2 LAB EXERCISE
2.1 GENERATING A FUNCTION BLOCK WITH VARIABLE DECLARATION
When blocks are generated with STEP 7, the quasi as a “Black-Box” in any program functions must be programmed under assignment from variables. Therefore the rules apply that in these blocks, no absolute addressed In/Outputs, memory bits, timers, counters, etc. are allowed to be used. Single variables and constants come here to be assigned.

In the following example, a function block with variable declaration is to be provided which contains a simple AND gate program.

Inputs:
- A1 = I 0.0
- A2 = I 0.1

Outputs:
- Y = 2.0

To create this program example, the following steps must be accomplished (with the production of a hardware configuration):

1. Open SIMATIC Manager with a double click (→ SIMATIC Manager).
2. Create a new project ( → File → New)

3. Generate a new project, allocate the project with a name **Testproject_FB**

   (→ Testproject_FB)
4. Insert a new **S7-Program** (→ Insert → Program → S7-Program).

5. Highlight the folder **Blocks** (→ Blocks).
6. Insert a **Function block** (→ Insert → S7 Block → Function block).

7. Enter the name of **FB1** for the FB and click on **OK** (→ FB1 → OK).
8. Open function block **FB1** with a double click. (→ FB1)

9. With **LAD, STL, FBD: Program blocks**, you now have an editor which gives you the possibility to edit your functions.

   In addition, the variables should be defined and specified in the variable declarations table, which is displayed in the FB1.

   These variables are type ‘in’, 'out', 'in_out', 'stat' and 'temp'.

   **Input parameters (IN) only in FBs, FCs, SFBs and SFCs**
   
   With help of the input parameters, data is assigned for the processing of the block.

   **Output parameters (OUT) only in FBs, FCs, SFBs and SFCs**
   
   With the output parameters, the results are assigned to the called block.

   **In/Out parameters (IN_OUT) only in FBs, FCs, SFBs and SFCs**
   
   With the in/out parameters, data is assigned to the called block, processed and files the results from the called block into the same variables.

   **Statistical data (STAT) only in FBs and SFBs**
   
   Statistical data is the local data of a function block that is saved in an instance data block and therefore remains preserved until the next processing of the function block.
Temporary data (TEMP) in all blocks

Temporary data is local data of a block that is filed during the processing of a block into the local data stack (L-Stack) and is no longer available after processing.

Note: Here the difference between FB/SFB and FC/SFC is stated. In a FC, there are no statistical variables (stat) to regulate because there is no memory for the contents of the variable contents after the processing of the FC. In the FB, these statistical variables are buffer stored in the corresponding local instance DB until the next processing of the FB. Out of this principle, only the FB is suited for the creation of programs in which data like e.g. step memory bits over more program cycles should remain stored away.

Data in a data block must be determined through data types.

The following standard-data types are defined in the S7 below:

<table>
<thead>
<tr>
<th>Type and description</th>
<th>Size in Bits</th>
<th>Format-options</th>
<th>Range and number notation (lowest to highest value)</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL (Bit)</td>
<td>1</td>
<td>Boolean-Text</td>
<td>TRUE/FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>BYTE (Byte)</td>
<td>8</td>
<td>Hexadecimal</td>
<td>B#16#0 to B#16#FF</td>
<td>B#16#10</td>
</tr>
<tr>
<td>WORD (Word)</td>
<td>16</td>
<td>Binary number</td>
<td>2#0 to 2#1111_1111_1111_1111</td>
<td>2#0001_0000_0000_0000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hexadecimal</td>
<td>W#16#0 to W#16#FFFF</td>
<td>W#16#100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BCD</td>
<td>C#0 to C#999</td>
<td>C#998</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Decimal number</td>
<td>B#(0,0) to B#(255,255)</td>
<td>B#(10,20)</td>
</tr>
<tr>
<td>DWORD (Double word)</td>
<td>32</td>
<td>Binary number</td>
<td>2#0 to 2#1111_1111_1111_1111_1111_1111_1111</td>
<td>2#1000_0001_0001_1000_1011_1011_0111_1111</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hexadecimal</td>
<td>DW#16#0000_0000 to DW#16#FFFF_FFFF</td>
<td>DW#16#00A2_1234</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Decimal number</td>
<td>B#(0,0,0,0) to B#(255,255,255,255)</td>
<td>B#(1,14,100,120)</td>
</tr>
<tr>
<td>INT (Integer)</td>
<td>16</td>
<td>Decimal number</td>
<td>-32768 to 32767</td>
<td>1</td>
</tr>
<tr>
<td>DINT (Int,32 bit)</td>
<td>32</td>
<td>Decimal number</td>
<td>L#-2147483648 to L#2147483647</td>
<td>L#1</td>
</tr>
<tr>
<td>REAL (Floating-)</td>
<td>32</td>
<td>IEEE floating-</td>
<td>Upper limit: +/-3.402823e+38</td>
<td>1.234567e+13</td>
</tr>
<tr>
<td>point number</td>
<td>point number</td>
<td>Lower limit: +/-1.175495e-38</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>--------------</td>
<td>--------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S5TIME (Simatic-Time) 16</td>
<td>S7-Time in steps of 10 ms</td>
<td>S5T#0H_0M_0S_10MS to S5T#0H_1M_0S_0MS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIME (IEC-Date) 32</td>
<td>IEC-Time in steps from 1ms, integer signed</td>
<td>T#24D_20H_31M_23S_648MS to T#24D_20H_31M_23S_647MS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATE (IEC-Date) 16</td>
<td>IEC-Date in steps of 1 Tag</td>
<td>D#1990-1-1 to D#2168-12-31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIME_OF_DAY (Time) 32</td>
<td>Time in steps of 1ms</td>
<td>TOD#0:0:0.0 to TOD#23:59:59.999</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHAR (Character) 8</td>
<td>ASCII-Characters</td>
<td>B'</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

10. Now the program can be entered by the use of variable names. (Variables are recognized with the symbol #'). These variables can be seen in the following example in STL. The function block FB1 should be saved and downloaded into the CPU.

11. In SIMATIC Manager, only the OB1 is opened in order to program the call of the FB1.
12. Accept the setting with a click on **OK** (→ OK).

13. With ‘LAD, STL, FBD: Program blocks’, you now have an editor that gives you the possibility to generate your OB1. The FB1 should be called together with its associated instance DB (also called local DB) with the following instruction line.

```
CALL FB1, DB10 <Enter>
```

Therefore, the instance DB (DB10) can automatically be generated when the question is answered with **Yes** (→ Call FB1, DB10 → Yes).
14. Then all variables from type ‘in’, ‘out’ and ‘in_out’ are displayed, so that these variables can be assigned actual parameters (e.g.: I 0.0, MW2 etc ...).

15. In our example, the allocation follows as shown. If the allocation is as follows, the organization block OB1 can be saved and downloaded. The mode switch of the CPU must be on STOP! (→)

Note: On this type, the FB1 can be called several times between the indication of different data blocks and in/output addresses. Thus it represents a standard block for this special setting of tasks.

16. Now in ‘SIMATIC Manager’, the instance DB (local DB) ‘DB10’ is chosen and downloaded into the CPU. The mode switch of the CPU must be on STOP!(→)

17. By switching the mode switch to RUN the program is started. The motor switches on when switch I0.0 is activated. It is switched off, as the switch I0.1 is activated. In the memory bit MD20, how often the FB1 from the OB1 is called, is taken into account. The memory bits get a feeling for the cycle time of the OB1. This happens with a high frequency, since the program cycle is very short in the OB1.
3 EXERCISES / OBSERVATIONS

- List all ladder logic programs made during the lab session. (Either draw or attach a printout)
- List the complete method of programming a function block—from creating FB to calling it in OB and downloading and simulating.
1 LEARNING GOAL
In this Lab session, the student should learn how to use integrated functions of CPU 314IFM.
- Frequency Meter Integrated Function
- Counter Integrated Function

2 EXPLANATION – FREQUENCY METER
2.1 Integrated Inputs/Outputs
Following table lists the special integrated inputs/outputs of the CPU 312 IFM and CPU 314 IFM for the Frequency Meter integrated function.

<table>
<thead>
<tr>
<th>CPU312 IFM</th>
<th>CPU314 IFM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>I 124.6</td>
<td>I 126.0</td>
<td>Measurement digital input</td>
</tr>
</tbody>
</table>

2.2 Function Overview
Following is an overview diagram (block diagram) for the Frequency Meter integrated function. The block diagram contains the main components of the integrated function and all its inputs and outputs.
The Frequency Meter integrated function enables continuous measurement of a frequency ≤ 10 kHz.
2.3 How the Frequency Meter Integrated Function Operates

The Frequency Meter calculates the current frequency from the measured signal and the sample time. The measured signal is connected via the Meter digital input of the integrated CPU inputs/outputs. The Frequency Meter counts the positive edges of the measured signal within a sample time in order to calculate the frequency.

The CPU calculates the frequency according to two different measuring principles:

- Measuring principle 1 is applied with a sample time of 0.1 s, 1 s or 10 s
- Measuring principle 2 is applied with a sample time of 1 ms, 2 ms or 4 ms

The Frequency Meter calculates the frequency according to the following formula:

\[
\text{Frequency} = \frac{\text{Number of positive edges}}{\text{Sample time}}
\]

The Frequency Meter calculates the frequency by measuring the time interval between two incoming positive edges at the meter’s digital input. You configure the sample time with STEP 7. You can choose between a sample time of 1 ms, 2 ms, 4 ms, 0.1 s, 1 s or 10 s.

The measurement process is restarted immediately after the sample time expires, with the result that the current frequency is always available. The sample time is 1 s. 6500 positive edges were counted during one sample period.

\[
\text{Frequency} = \frac{6500}{1\text{s}} = 6500 \text{ Hz}
\]

The sample times from 0.1 s to 10 s were introduced for the measurement of high frequencies. The higher the frequency, the more accurate is the result of the measurement. With high frequencies, this measuring principle is associated with:

- High measurement accuracy
- Low load on the cycle

The sample times from 1 s to 4 s were introduced for the measurement of low frequencies. The lower the frequency, the more accurate is the result of the measurement. With low frequencies, this measuring principle is associated with:

- High measurement accuracy
- High-speed response to process events (e.g. process interrupt triggering)
- A high load on the cycle

2.4 Function of the Comparator

The Frequency Meter integrated function has two integrated comparators with which you can monitor adherence to a specific frequency range. The upper limit comparator intervenes if the frequency FREQ exceeds a defined comparison value U_LIMIT. In this case, status bit STATUS_U at SFB 30 is enabled. The lower limit comparator intervenes if the frequency FREQ falls below a defined comparison value L_LIMIT. In this case, status bit STATUS_L at SFB 30 is enabled. You can evaluate the status bits in your user program. Until the first valid frequency value is displayed, the signal state of the status bits at SFB 30 is 0. If the value
exceeds the U_LIMIT comparison value or falls below the L_LIMIT comparison value, a corresponding process interrupt is triggered if configured in STEP 7 (sample time 1, 2 or 4 ms and process interrupt activated).

Following Figure illustrates the function of the comparator. The shaded areas indicate when a lower or upper limit is exceeded.

2.5 Assigning Parameters
You assign the parameters for the integrated function with STEP 7. Following table lists the parameters for the Frequency Meter integrated function.
2.6 System Function Block 30

The Frequency Meter integrated function is assigned to SFB 30. A graphical illustration of SFB 30 is shown below

### Input Parameters of SFB 30

In following table there is a description of the input parameters of SFB 30.
### 2.6.2 Output Parameters of SFB 30

In following table there is a description of the output parameters of SFB 30.

<table>
<thead>
<tr>
<th>Output Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENO</td>
<td>Output parameter ENO indicates whether an error occurred during execution of the SFB. If ENO = 1, no error occurred. If ENO = 0, the SFB was not executed or an error occurred during execution.</td>
</tr>
<tr>
<td>Data type: BOOL</td>
<td>Address ID: I, Q, M, Value range: 0/1 (FALSE/TRUE)</td>
</tr>
<tr>
<td>FREQ</td>
<td>The measured frequency is output in kHz in this parameter.</td>
</tr>
<tr>
<td>Data type: DINT</td>
<td>Address ID: I, Q, M, Value range: from -1 to 1000000</td>
</tr>
<tr>
<td>U_LIMIT</td>
<td>The current U LIMIT comparison value is output in this output parameter.</td>
</tr>
<tr>
<td>Data type: DINT</td>
<td>Address ID: I, Q, M, Value range: from -2147483648 to 2147483647</td>
</tr>
<tr>
<td>L_LIMIT</td>
<td>The current L LIMIT comparison value is output in this output parameter.</td>
</tr>
<tr>
<td>Data type: DINT</td>
<td>Address ID: I, Q, M, Value range: from -2147483648 to 2147483647</td>
</tr>
<tr>
<td>STATUS_U</td>
<td>The output parameter STATUS_U indicates the position of the frequency relative to the comparison value U LIMIT:</td>
</tr>
<tr>
<td></td>
<td>Frequency FREQ &gt; comparison value U_LIMIT: output parameter STATUS_U enabled</td>
</tr>
<tr>
<td></td>
<td>Frequency FREQ = comparison value U_LIMIT: output parameter STATUS_U not enabled</td>
</tr>
<tr>
<td>Data type: BOOL</td>
<td>Address ID: I, Q, M, Value range: 0/1 (FALSE/TRUE)</td>
</tr>
<tr>
<td>STATUS_L</td>
<td>The output parameter STATUS_L indicates the position of the frequency relative to the comparison value L LIMIT:</td>
</tr>
<tr>
<td></td>
<td>Frequency FREQ ≥ comparison value L_LIMIT: output parameter STATUS_L not enabled</td>
</tr>
<tr>
<td></td>
<td>Frequency FREQ &lt; comparison value L_LIMIT: output parameter STATUS_L enabled</td>
</tr>
<tr>
<td>Data type: BOOL</td>
<td>Address ID: I, Q, M, Value range: 0/1 (FALSE/TRUE)</td>
</tr>
</tbody>
</table>
3 EXPLANATION – COUNTER

3.1 Integrated Inputs/Outputs

Following table lists the special integrated inputs/outputs of the CPU 312 IFM and CPU 314 IFM for the Counter integrated function.

<table>
<thead>
<tr>
<th>CPU 312 IFM</th>
<th>CPU 314 IFM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1124.6</td>
<td>1126.0</td>
<td>Digital input up</td>
</tr>
<tr>
<td>1124.7</td>
<td>1126.1</td>
<td>Digital input down</td>
</tr>
<tr>
<td>1125.0</td>
<td>1126.2</td>
<td>Digital input direction</td>
</tr>
<tr>
<td>1125.1</td>
<td>1126.3</td>
<td>Digital input hardware start/stop</td>
</tr>
<tr>
<td>Q 124.0</td>
<td>Q 124.0</td>
<td>Digital output A</td>
</tr>
<tr>
<td>Q 124.1</td>
<td>Q 124.1</td>
<td>Digital output B</td>
</tr>
</tbody>
</table>

3.2 Function Overview

Following is an overview diagram (block diagram) for the Counter integrated function. The block diagram contains the main components of the integrated function and all its inputs and outputs.

The Counter integrated function enables the measurement of counting pulses up to a frequency of 10 kHz. The Counter integrated function can count up and down.

3.3 How the Counter Operates

The counter calculates the actual value of the counter from the counting pulses (up and down). The counting pulses are measured via two digital inputs on the CPU: Up digital input
and Down digital input. You use STEP 7 to configure whether the digital inputs are evaluated and, if so, whether positive or negative edges are evaluated.

The counter calculates the actual value according to the following formula:
Actual value = no. of edges on Up DI – no. of edges on Down DI

Figure below shows an example to illustrate how the actual value of the counter is changed by the counting pulses at the two digital inputs. The positive edges are evaluated on the Up digital input and the negative edges are evaluated on the Down digital input.

You can start or stop the Counter integrated function in one of the following ways:

- From the integrated inputs/outputs: HW_Start/Stop digital input
- From the user program: input parameter EN_COUNT at SFB 29

The digital input and the input parameter are ANDed. This means that the Up and Down digital inputs are only evaluated when both are enabled. You can define the start value at which the counter begins counting with input parameter PRES_COUNT at SFB 29. The start value is accepted by the counter:

- On a positive edge on input parameter SET_COUNT of SFB 29
- On the occurrence of a counter event, for example, comparison value of the counter reached from below (parameterized with STEP 7).

You can change the counting direction of the Up and Down digital inputs with the Direction digital input. While the signal status of the Direction digital input is 0, the Up digital input counts down and the Down digital input counts up. The Counter integrated function counts pulses up to a frequency of 10 kHz.
3.4 System Function Block 29

The Counter integrated function is assigned to SFB 29. A graphical illustration of SFB 29 is shown in figure below.

3.4.1 Input Parameters of SFB 29

In table below you will find a description of the input parameters of SFB 29.
LAB#5

<table>
<thead>
<tr>
<th>Output Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENO</td>
<td>Output parameter ENO indicates whether an error occurred during execution of SFB 29. If ENO = 1, no error occurred. If ENO = 0, SFB 29 was not executed or an error occurred during execution. Data type: BOOL Address ID: I Q, M, Value range: 0/1 (FALSE/TRUE)</td>
</tr>
<tr>
<td>COUNT</td>
<td>The actual value of the counter is output in this parameter. When the value range is exceeded, the following apply: • Upper limit exceeded: the counting process continues with the minimum value in the value range. • Lower limit exceeded: the counting process continues with the maximum value in the value range. Data type: DINT Address ID: I Q, M, Value range: from -2147483648 to 2147483647</td>
</tr>
<tr>
<td>COMP_A</td>
<td>The current COMP_A comparison value is output in this output parameter. Data type: DINT Address ID: I Q, M, Value range: from -2147483648 to 2147483647</td>
</tr>
<tr>
<td>COMP_B</td>
<td>The current COMP_B comparison value is output in this output parameter. Data type: DINT Address ID: I Q, M, Value range: from -2147483648 to 2147483647</td>
</tr>
<tr>
<td>STATUS_A</td>
<td>The output parameter STATUS_A indicates the position of the actual value relative to comparison value COMP_A: • Actual value COUNT ≥ comparison value COMP_A: output parameter STATUS_A enabled • Actual value COUNT &lt; comparison value COMP_A: output parameter STATUS_A not enabled Data type: BOOL Address ID: I Q, M, Value range: 0/1 (FALSE/TRUE)</td>
</tr>
<tr>
<td>STATUS_B</td>
<td>The output parameter STATUS_B indicates the position of the actual value relative to comparison value COMP_B: • Actual value COUNT ≥ comparison value COMP_B: output parameter STATUS_B enabled • Actual value COUNT &lt; comparison value COMP_B: output parameter STATUS_B not enabled Data type: BOOL Address ID: I Q, M, Value range: 0/1 (FALSE/TRUE)</td>
</tr>
</tbody>
</table>

4 LAB EXERCISE

4.1 Frequency Counter

After creating the project in OB1, draw the network 1 as under and then save OB1. After this open the hardware configuration by double clicking hardware in SIMATIC 300 STATION. In this hardware configurations right click CPU and select properties. In properties select Integrated Function and activate Frequency Meter and save settings. Finally, download the SIMATIC 300 STATION and verify the results as explained.
4.2 Counter

After creating the project in OB1, draw the network as under and then save OB1. After this open the hardware configuration by double clicking hardware in SIMATIC 300 STATION. In this hardware configurations right click CPU and select properties. In properties select Integrated Function and activate Counter and save settings. Finally, download the SIMATIC 300 STATION and verify the results as explained.
5 EXERCISES / OBSERVATIONS

- List frequency measured in Lab, also list the two limits of frequency and what happened when frequency crossed those limits.
- List values of count observed in the Lab, values of comparators, and what happened when count value crossed those values.
To understand Project navigation and to understand system configuration

Objective:
- To understand the project organization in step7.
- To configure the system.
- To develop OBI, download OBI and simulate OBI.
- To observe the effect of memory reset.

Task:
1. Double click the Simatic Manager and create your project wizard by selecting CPU 314IFM and OBI.
2. Click Simatic 300 Station and then hardware.
3. Using Insert > Hardware Catalog arrange your rack by dragging and dropping the relevant modules.
4. Now double click the CPU and different tabs will be displayed.
5. Click the Startup tab and you will see that only warm restart is possible in your selected CPU.
6. Click the Retentive Memory tab and set the retentive areas according to the requirement.
7. Download your configuration and exit.
8. Click the block folder and then OBI.
9. From menu view, select LAD mode of representation and develop the drawn networks.
10. Download the block OBI and simulate it.
11. From project window open the block OBI online as View > Online.
12. Reset the memory as PLC > Clear/Reset and open the block OBI.
13. Again download OBI and then save contents of RAM to integrated EPROM as PLC>Save RAM to ROM.
14. Reset the memory again and then open OBI online. Whether the block is opened this time? Give the reason.
15. Conclude your observations.

To understand the logical operation of PLC such as SR latch operation
Objective:

- Understand the logical Operation of PLC
- Understand the S R operation

Task:

Belt conveyor 1 for the lifting table is started when momentary contact push button S1 is actuated. The pallet rolls over the inclined roller train onto the moving conveyer. As soon as pallet actuated limit switch S2, belt conveyor 1 stops and the lifting table begin moving upward. When the limit S4 is actuated, the lifting table stop moving and belt conveyor 1 and 2 are started. Both belt conveyor stop when limit switch S5 is actuated. The lifting table descends until limit switch S3 is actuated.
To develop a Function and to use it during application.

Objective:

- To develop a function and to program its call
- To understand the SR function
- To Create and use variable table

Task:

A collecting basin for waste water is emptied using two pumps. The system is started when ENABLE button E0 is pressed.

**PUMP 1:**

**Start:** The pump is started either manually by pressing momentary contact push button S2 or automatically by float switch B1. When the water level is exceeded

**STOP:** if the water level falls below float switch B0, the pump switch off automatically at any time by pressing push button S1 or by thermal over current release F1

**PUMP 2:**

**Start:** The pump is started either manually by pressing momentary contact push button S4 or automatically by float switch B4. When the water level is exceeded.

**Stop:** If the water level falls below float switch B3, the pump switch off automatically at any time by pressing push button S3 or by thermal over current release F2.

Lamp H0 to H3 indicates the operating state of the pumps. Whole of the system is shut down when stop push button S0 is pressed. Hooter H4 must sound when the water level reaches float switch B2 or when a pump fails because the associated thermal over current release has tripped.
NOTES FOR THE PROGRAMMING OF SIMATIC S7-300 WITH STEP 7

1. Introduction
This document contains a brief account of the SIEMENS SIMATIC S7-300 PLC hardware information and its software Step 7. It has been obtained from SEIMENS website.

2. NOTES FOR THE PROGRAMMING OF SIMATIC S7-300 WITH STEP 7

2.1 AUTOMATION SYSTEM SIMATIC S7-300

The automation system SIMATIC S7-300 is the modular miniature control system for the low and medium power ranges. There is a comprehensive module spectrum for the optimal adjustment in the automation task.

The S7-Controller consists of a power supply (PS), a central processing unit (CPU) and signal modules for in and/or output devices (I/O devices). If necessary, communication processors (CPs) and function modules (FMs) can be used for specific tasks (e.g. stepping motor control).

The programmable logic controller (PLC) supervises and controls a machine or a process in conjunction with an S7 program. The I/O devices are addressed in the S7-Program via the Input (I) and Output addresses (Q).

The system is programmed with the software STEP 7.

2.2 PROGRAM SOFTWARE STEP 7

The software STEP 7 is the program tool for the automation systems
- SIMATIC S7-300
- SIMATIC S7-400
- SIMATIC WinAC

With STEP 7, the following functions can be used for the automation construction:
- Configuring and parameterization of hardware
- Generation of a user program
- Debug, commissioning, and service
3. WHAT IS PLC AND WHAT ARE PLCS USED FOR?

3.1 WHAT IS THE CONCEPT OF A PLC?

PLC is an abbreviation for programmable logic control. This describes equipment that controls a process (e.g. a printing machine for printing newspapers, a bagging plant to bag cement, a press for pressing plastic-shaped parts, etc...). This process occurs according to the instructions of a program in the memory of the equipment.

![Diagram of PLC process]

3.2 HOW DOES A PLC DRIVE A PROCESS?

The PLC controls the process, in which Actuators are wired as Outputs to designated connections of a PLC with a control supply voltage of e.g. 24V. Motors can be switched on and off, valves extended or retracted, or lamps switched on and off through this connection.
3.3 FROM WHERE DOES A PLC GET INFORMATION ABOUT THE STATE OF A PROCESS?

A PLC receives information about the process from Signal generators which are wired to the inputs of the PLC. These signal generators can be e.g. sensors which recognize whether a working part, switches or buttons lie in a certain position. This position can be closed or opened. Please note the variation between NC contacts, which are inactive when closed, and NO contacts, which are inactive when open.

3.4 WHERE DOES THE DIFFERENCE BETWEEN A NORMALLY OPEN (NC) AND CLOSED (NC) CONTACT LIE?

The variation between NO contacts and NC contacts is within a signal generator.
The switch shown here is a NO contact, i.e. it is closed when it is active.

![Diagram of NO contact]

The switch shown here is a NC contact. i.e. it is closed when it is not active.

![Diagram of NC contact]

3.5 HOW DOES A PLC COMMUNICATE WITH IN/OUTPUT SIGNALS?

The designation of a certain input or output within the program is referred to as addressing.

The inputs and outputs of the PLCs are mostly defined in groups of eight on digital input and/or digital output devices. This eight unit is called a byte. Every such group receives a number as a byte address.

Each in/output byte is divided into 8 individual bits, through which it can respond with. These bits are numbered from bit 0 to bit 7. Thus one receives a bit address.

The PLC represented here has input bytes 0 and 1 as well as output bytes 4 and 5.
Here e.g. the fifth input from the higher bits responds with the following address:

\[ I \ 0.4 \]

Here the address type is specified as Input, 0 the byte address and 4 the bit address.
The byte address and bit address are always separated with a point.

**Note:** For the bit address here, the 4 stands for the fifth input because the count begins at 0.

Here e.g. the lower bits respond with the following address:

\[ Q \ 5.7 \]

Q. Here the address type is specified as Output, 5 the byte address and 7 the bit address.
The byte address and bit address are always separated with a point.

**Note:** For the bit address here, the 7 stands for the eighth output, because the count begins at 0.

### 3.6 HOW DOES THE PROGRAM WORK IN A PLC?
Program processing in a PLC happens cyclically with the following execution:

1. After the PLC is switched on, the **processor** (which represents the brain of the PLC) questions if the individual inputs have been transmitted or not. This status of the input is stored in the process-image input table (**PII**). Leading inputs become the information 1 or High when enabled, or the information 0 or Low when not enabled.

2. This processor processes the program deposited into the program memory. This consists of a list of logic functions and instructions, which are successively processed, so that the required input information will already be accessed before the read in PII and the matching results are written into a process-image output table (**PIQ**). Also other storage areas for counters, timers and memory bits will be accessed during program processing by the processor if necessary.

3. In the third step after the processing of the user program, the status from the PIQ will transfer to the outputs and then be switched on and/or off. Afterwards it continues to operate, as seen in point 1.

1. Input status in the PII memory.

2. Handling of a program command for a directive with access from the PII and PIQ as well as the timer, counter and mem. bit.

```
PLC program in program memory
1. Statement
2. Statement
3. Statement
4. Statement
...
last statement
```

PII

Timer

Counter

Mem. bit

PIQ
3. Status from the PIQ transferred to the outputs.

Note: The time that the processor requires for this execution is called a cycle time. This time is independent from the number and types of commands.

3.7 HOW DO THE LOGIC OPERATIONS IN A PLC PROGRAM APPEAR?

Logic functions can be used in order to specify conditions for the toggling of outputs. These functions can be provided to the PLC-Program in the programming languages ladder diagram (LAD), function block diagram (FBD) or statement list (STL). For the sake of descriptiveness, we will limit ourselves here to FBD. A wide range of different logic operations can be used in PLC programs. AND as well as OR- operation and NEGATION of an input can be frequently used. Basic examples are briefly described below.

Note: Further information of logic operations can be quickly found in the online help section.

3.7.1 AND- OPERATION

Example of an AND- OPERATION:

A lamp should ignite when two switches at a closed contact are active at the same time.

Circuit diagram:
Comment:
The lamp lights when both switches are active.
When the switch S1 and S2 are active, the lamp H1 will light up.

A PLC circuit:
In order to implement logic in a PLC program, both switches must be naturally attached at the inputs of the PLC. Here S1 is wired to the input I 0.0 and S2 to the input I 0.1.
In addition, the lamp H1 must be attached to an output e.g. Q 4.0.

AND-Operation in FBD:
In the function diagram FBD, the AND-Operation is programmed and is shown by figurative representation below:

AND-Operation inputs.
There can be more than 2 inputs!
3.7.2 OR- OPERATION

Example of an OR- Operation:

A lamp should ignite when one or both switches at a normally open circuit are active.

Circuit diagram:

Comment:
The lamp lights when one or both switches are active.
When the switch S1 or S2 is active, lamp H1 will light up.

A PLC circuit:
In order to implement logic in a PLC program, both switches must be naturally attached at the inputs of the PLC. Here S1 is wired to the input I 0.0 and S2 to the input I 0.1.

In addition, the lamp H1 must be attached to an output e.g. Q 4.0.

OR-Operation in FBD:

In the function diagram FBD, the OR-Operation is programmed and is shown by figurative representation below:

OR-Operation inputs. There can be more than 2 inputs!

Output, where the assignment is allocated.
### 3.7.3 NEGATION

In logical functions it is often required to know whether a **NO contact** is **not active** or if a **NC contact** is **active** so that there will be no voltage against the appropriate inputs. This can be achieved with the use of a **Negation** on the input of the AND/OR Operation.

In the function diagram FBD, the negation of the inputs of the AND-Operation is programmed and is shown by figurative representation below:

The output Q 4.0 has the correct value when I 0.0 is not active and I 0.1 is active.

### 3.8 HOW IS A PLC-PROGRAM GENERATED? HOW DOES IT ARRIVE IN THE MEMORY OF THE PLC?

The PLC program is provided with the software STEP 7 on a PC and buffered there.
After the PC is connected with the MPI interface of the PLC, the program can be loaded with a loading function into the memory of the PLC.

1. PLC- Program created with STEP 7 on a PC.
2. PC connected with MPI- Interface of the PLC.
3. Program from the PC built in the PLC memory.

Notice: The exact execution of the program will be described step by step in chapters 8 through 10.

4. ASSEMBLY AND OPERATION OF THE SIMATIC S7-300.

Device spectrum:

The SIMATIC S7-300 is a modular miniature control system and provides the following device spectrum:
- Central processing units (CPUs) with different power ranges, partly integrated with In-/Outputs (e.g. CPU312IFM/CPU314IFM) or integrated with a PROFIBUS-Interface (e.g. CPU315-2DP)
- Power supply devices (PS) with 2A, 5A or 10A.
- Interface modules (IMs) for a more interconnecting design of the SIMATIC S7-300
- Signal modules (SMs) for digital and analog in- and output.
- Function modules (FMs) for special functions (e.g. stepping motor control)
- Communication processors (CP) for network connection.

**Note:** Only a current supply device, any CPU as well as a digital in and output is required for this module.

**Important elements of a voltage supply and CPU:**
MPI Interface:
Each CPU possesses an MPI interface for the networking of program devices (e.g. PC adapter). This is found behind a flap at the front of the CPU.

Mode selector:
Each CPU possesses a code switch for the switching of the modes of operation. Certain programmed functions are allowed depending upon the position of the code switch. The following modes of operation are possible:

- **RUN-P**: Program runs; All PG functions are allowed.
- **RUN**: Program runs; Only read PG functions are allowed.
- **STOP**: Program does not run; All PG functions are allowed.
- **MRES**: With this position, one can accomplish a reset as described.

Memory reset:
Memory reset erases all user data on the CPU each time the program is begun.

This is performed in the following three steps:
<table>
<thead>
<tr>
<th>Step</th>
<th>Execution</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Turn the key to the <strong>STOP</strong> position.</td>
<td>STOP indication is shown.</td>
</tr>
<tr>
<td>2</td>
<td>Turn the key to the <strong>MRES</strong> position and hold it in this position (approx. 3 seconds) until the <strong>STOP</strong>- indicator is shown.</td>
<td>The STOP-Indicator expires and after approx. 3 seconds, it will be shown again. With new CPUs, wait until the STOP-Indicator lights up for the second time. <strong>Important:</strong> Between step 2 and step 3 a maximum of 3 seconds should go by.</td>
</tr>
<tr>
<td>3</td>
<td>Turn the key back to the <strong>STOP</strong> position and within the following 2 seconds restart in the <strong>MRES</strong> position.</td>
<td>The STOP-Indication blinks for approx. 3 seconds and then lights up again normally: When everything is ok.; The <strong>CPU is reset.</strong></td>
</tr>
</tbody>
</table>
1. **INTRODUCTION**

   This document contains a brief account of the PLCSIM feature of software Step 7. It has been obtained from SEIMENS website.

2. **NOTES FOR THE APPLICATION OF S7- PLCSIM**

   The area of application of S7-PLCSIM is mainly a test of the provided STEP 7-Programs for the SIMATIC S7-300 and SIMATIC S7-400 when one can not immediately and directly debug the hardware. This problem can have the following reasons:

   - Smaller program modules, whose execution cannot yet be debugged at a machine.
   - The application is so critical that damage for a person and a machine is feared if programming errors arise. With a simulation, these errors can be eliminated without causing physical harm.

   There is also a possibility to use this application for the purpose of practice, if a hardware PLC is not present.

   With the employment of SIMATIC-PLCSIM, the following points should be considered:

   - The software package that should be used is the STEP 7 Professional or the STEP 7 Student version (Not STEP 7 Mini!)
   - Projects for all SIMATIC S7-300 and S7-400 CPUs as well as SIMATIC WinAC can be debugged here.
   - The use of function modules (FMs) and communication processors (CPs) cannot be simulated.
   - Timer functions do not correspond to the real time requirement, since their execution depends on the speed of the assigned computer.
1. INTRODUCTION
This document contains a brief account of the structured programming using function blocks and functions. It has been obtained from SEIMENS website.

2. NOTES FOR STRUCTURED PROGRAMMING WITH FCS AND FBS
The program execution is written in blocks in STEP 7. The organization block OB1 is already available.

The program execution describes the interface to the operation system of the CPU and is called automatically from this block and executed cyclically.

By extensive control tasks, one cuts the program into small, manageable and ordered program blocks in functions.

These blocks are then called from the organization block over the block call instructions 
(Call xx / UC xx / CC xx). If the block end was realized, the program executes further in the previously called block call.

For structured programming, STEP 7 offers the following:

- FB (Function block):
  The FB has an assigned storage area. If a FB is called, it can be assigned a data block (DB). From the data in this instance, the DB can be accessed by a call from the FB. A FB can be assigned different DBs. Further FBs and FCs can also be called over block call instructions in a function.

- FC (Function):
  A FC does not possess an assigned storage area. The local data of a function is lost after the editing of the function. Further FBs and FCs can be called over block call instructions in a function.
The structure of a program can look as follows:

Note: In order to use the blocks, they must first be generated. There is also a possibility to program these FCs and FBs in the form of standard blocks under the use of internal variables. Then any function can be called often, whereas another local instance DB must access a FB each time.