LCAMP

Learner Centric Advanced Manufacturing Platform

BASICS OF AUTOMATION WITH PLC

WP6 COLLABORATIVE LEARNING FACTORY

Co-funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

Co-funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

LCAMP partners:

TKNIKA – Basque VET Applied Research Centre, CIFP Miguel Altuna LHII, DHBW Heilbronn – Duale Hochschule, Baden-Württemberg, Curt Nicolin High School, AFM – Spanish Association of Machine Tool Industries, EARLALL – European Association of Regional & Local Authorities for Lifelong Learning, FORCAM, CMQE: Association campus des métiers et des qualifications industrie du future, MV: Mecanic Vallée, KIC: Knowledge Innovation Centre, MADE Competence Centre Industria 4.0; AFIL: Associazione Fabbrica Intelligente Lombardia, SIMUMATIK AB; Association HVC Association of Slovene Higher Vocational Colleges; TSCMB:Tehniški šolski center Maribor, KPDoNE: Kocaeli Directorate Of National Education; GEBKİM OIZ and CAMOSUN college.

Document summary

Document type:	Public Course
Title	Basics of automation with PLC
Author/s	Marjan Bezjak
Reviewer	Samo Čretnik
Date	November 2025
Document status	1.0
Document level	Confidential until its publication
Document description	This course aims to cover configuration of PLC automation and programming hardware, advantages of using PLC in process automation, optimal selection of PLC and extension modules for automation of a specific process, the basic functions of PLC programming, PLC programming on the example of a practical task.
Cite this deliverable as:	Bezjak M., 2025, PLC
Document level	Public

CONTENT TABLE

EXECUTIVE SUMMARY

In this course we provide participants with knowledge about configuration of PLC automation and programming hardware, advantages of using PLC in process automation, optimal selection of PLC and extension modules for automation of a specific process, the basic functions of PLC programming, PLC programming on the example of a practical task (basic combination and stepper control).

1. GENERAL ABOUT YOUR COMPUTER

Computers have a huge impact on our lives in modern times. They date back to the times just before and after the First and Second World Wars, and the first useful computers (e.g. Z3, ENIAC) were created at the end of the second half of the 20th century and were initially accessible only to the army and major research institutions. As semiconductor technology advanced greatly in the 1950s and 1960s, computers became less demanding, more powerful, and cheaper, making them more accessible to the general population.

This led to the development of ideas that the computer could also manage more complex processes, not just the processing of numerical data. The biggest problem was that they were still relatively expensive and large, and thus only suitable for a limited range of applications.

With the advent of microprocessors, the situation changed so significantly that the computer became "the most revolutionary product in the history of mankind" (Gordon Moore, founder of INTEL). Nowadays, it is no longer possible to imagine life without microprocessors, as we encounter them at every step. In the household, we no longer find ourselves without a washing machine or dishwasher, without a microwave oven and some other appliances, and it is also difficult to imagine living without a telephone, radio, television, camera

… Cars are also equipped with microprocessor systems.

In commercial and industrial systems, production control systems, measuring devices, robots, etc., are built on the basis of microprocessors, which are connected to local networks, in which smaller computers to control systems in the final production cells are controlled by more powerful ones. They then manage production using databases, take into account business decisions, etc.

Compared to discrete logic, microcomputer control consists of fewer components and their development is faster and simpler, testing is easier, the price is lower, and the possibility of error is lower. The same electronics can be used by several different devices, since standard microcomputer circuits are universally applicable, and thanks to these circuits, they are able to quickly adapt to changes, because only a new configuration and program are required. Maintenance is simpler (fewer elements, universal modules, possibility of self-testing, standardized procedures).

1.1. THE ROLE OF THE COMPUTER

You can think of a computer as a black box, which is an intelligent part between input and output. In order for a computer to be able to process data, it must be able to:

- read the written program and execute it,
- perform arithmetic and logical operations on data,
- make comparisons between data and make decisions,

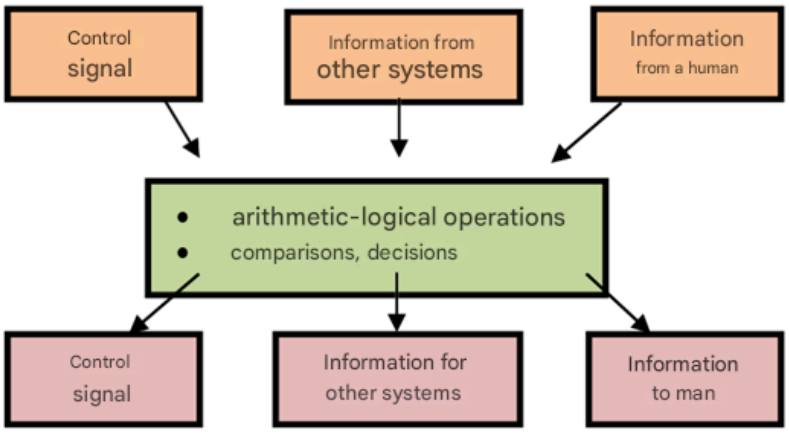


Figure 1: Computer operation

1.2. COMPUTER SHARING

Information computers – in the office, they help the secretary in running the business, and in the development department, the engineers in the development department in the development of new products.

The control computers that control machines and production lines in the production plant are mostly microcomputers, as they have only basic components (RAM, ROM, I/O interfaces) in addition to the microprocessor. PCs can also be used for control computers with the appropriate software tools and control interfaces.

CAD/CAM (Computer Aided Design/Computer Aided Manufactury): Local computer networks enable the connection of information and control computers to CAD/CAM systems and further to systems for managerial production monitoring.

1.3. MICROCOMPUTER AS A PROCESS CONTROLLER PROGRAMMABLE LOGIC CONTROLLER (PLC)

Its function is to replace man in the control and control of machines and production processes in industry.
Classic control with permanently activated program:

Electromechanical controls: The basic building blocks are relays, switches, valves. The mode of operation is determined by the functional plan and the binding scheme.

Electronic controls: They are made up of logic elements in the form of chips and have these elements connected to each other by design. The design is constructed based on the

requirements of a regularity table or logic function using digital circuit minimization methods. The function of the control operation is determined by the interconnection of elements. Here, too, we say that the program of action is determined by the program.

Programmable Logic Controller (PLC):

In the case of large-scale controls for control and control of industrial processes, however, we usually have to change the operating program during the testing period. Sometimes the technological process also changes frequently and requires a new program. Therefore, there is a demand for greater flexibility of the steering to different needs. With such controls, you could change the program without having to change the wiring.

PLC-based control systems consist of:

- apparatus or hardware, or hardware (controller, transducers, contactors, switches, wiring, etc.)
- software – software (a program of control operation that is entered into the program part of the memory). To change the function of the control operation, it is only necessary to write a new program to the program memory, while changes in hardware, as a rule, are not required.

In a controlled or regulated process, signal transmitters (sensors) provide data on the status and value of individual quantities (temperature, pressure, speed, flow, position, etc.). This data is processed in the input unit into binary values that can be processed by the processor. The processor reads individual input data in accordance with the program and determines the states in the process. On the basis of the states in the process and the program, it gives commands in the form of voltage pulses to the executive elements of the control (relays, light and sound detectors, displays, servo motor, stepper motor, etc.) via the output unit.

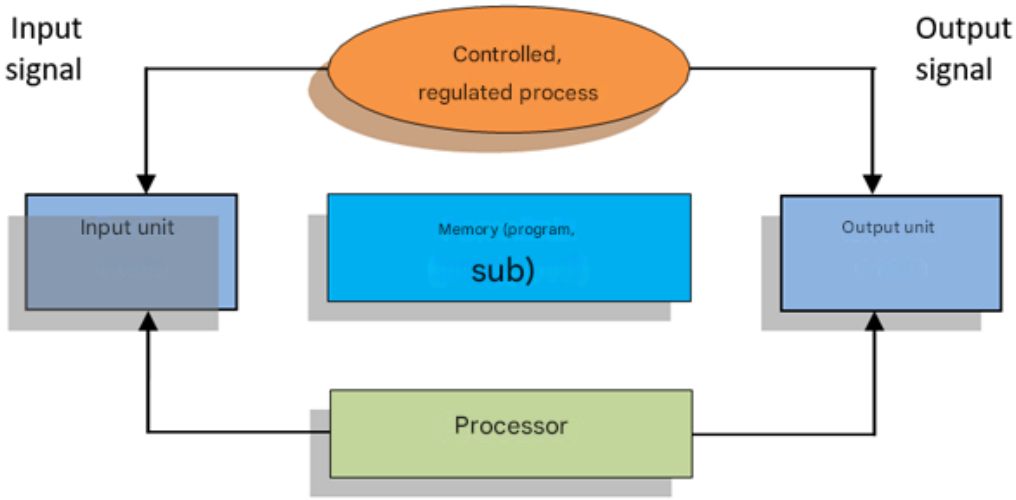


Figure 2: Block display and operation of PLC

With a programmable logic controller (PLC), you can perform the following process functions:

- capturing and editing process data,
- implementation of control and regulation,

- protection of processes in conflict situations,
- analysis and evaluation of the results of the process.

Summary:

Nowadays, the computer has a huge impact on our lives. At the end of the 20th century, computers were only available to the military and some research institutions, but then technology advanced a lot in the 50s and 60s and computers became less demanding, more powerful and cheaper. However, when microprocessors appeared, the computer became the most revolutionary product in the history of mankind, and nowadays we encounter them at every turn. Computers are divided into information, control and CAD/CAM computers.

The Programmable Logic Controller (PLC) replaces humans in the control and control of machines and production processes in industry. It consists of apparatus or hardware (hardware) and software (software). It can be used to capture and edit process data, perform feeds and regulations, protect processes in conflict situations, and analyze and evaluate process results.

Questions:

1. How can you define the term computer?
2. What does a computer need to know to process data?
3. Where can the computer get the data from and where can it send it?
4. What is the difference in usability between information and control computers?
5. What does CAD/CAM mean?
6. What are the permanently harnessed controls?
7. What are the advantages of programmable logic controllers over permanent control wiring?
8. What process functions can a programmable logic controller (PLC) perform?

2. COMPUTER PROGRAMMING METHODS

In this chapter, you will learn about the importance of a computer program and the characteristics of programming languages, as well as the programming tools and phases in creating a program. The concepts of computer, computer program, programmer and programming are explained. Programming languages are presented with examples and divided according to the method of programming, the time of compilation and the method of input. This is followed by a presentation of the Lego Mindstorm, Crocodile Technology and Labview (http://www.ni.com/labview/applications/) data capture and processing programs and a basic approach to programming.

2.1. COMPUTER, COMPUTER PROGRAM AND PROGRAMMING

A computer is an information-processing machine that performs operations or instructions according to a computer program.

A computer program is an instruction to a machine on what and how to process information.

Programming is the recording of a task in a form that is understandable to the processor.

A computer programmer or software developer writes computer programs. In doing so, he must know the problems and solutions to the tasks that will later be solved by the computer with the help of a written program.

2.2. DIVISION OF PROGRAMMING LANGUAGES WITH EXAMPLES

a) **The machine code,** machine language, or machine text of a program. Machine code or machine language) is text or code in executable files created from source text by compilers or compilers. The machine text of a program consists of sequences of executable machine commands that are executed on personal computers by a central processing unit. Each processor system has its own architectural design (platform). This means that the machine language is very different between different processors (but the same for those on the same design), so a different compiler must be used for each platform. Compiler) to translate source code into hardware. Although most dependent on the processor design, the platform is the set of properties of all parts of the computer. This also includes the operating system and other important programs. Software) or machine parts (Eng. Hardware). The central processing unit directly understands the object code, but for successful execution on a given operating system, we also need a certain execution notation. Executable format), which is defined by the ABI interface (ABI). Application Binary Interface) of the operating system. Thus, the executable files contain the hardware code of a particular executable, which is the central reason for the non-portability of computer programs between individual operating

systems. GNU/Linux and the BSD family use, for example, the ELF executable notation. Executable and Linkable Format), Mac uses Mach-o, while Windows PE (Eng. Portable and Executable Format). An executable file with machine code can also be without an executable record. A file that can be run without an operating system on a central processing unit t's called a flat executable. A flat binary file. Machine code is often referred to as the first generation of programming languages.

An example of machine code is shown in Figure 3.

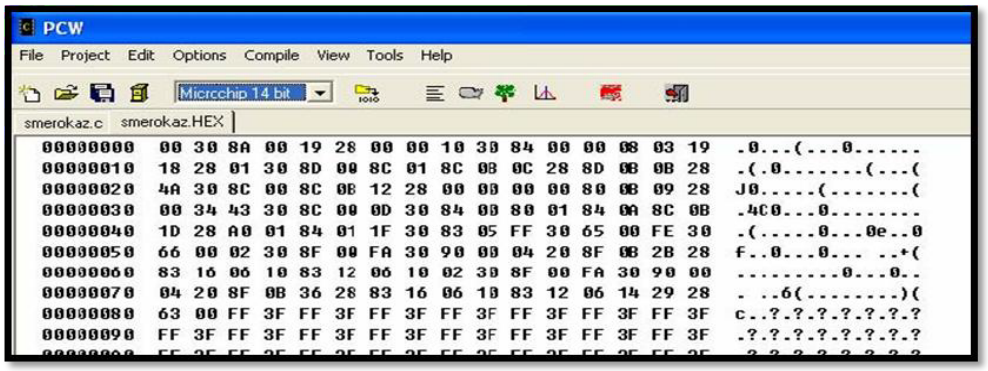


Figure 3: Example of a program in machine code

b) **Assembly language** or assembler is a low-level second-generation programming language (the first-generation language is machine code) that is written with mnemonics. In general, mnemonics are readable versions of binary sequences (zeros and ones) that need to be compiled in order to obtain a code that is understandable to the central processor. Specifically, mnemonics are command codes. Operation codes, abbreviated as Opcodes), which are defined in the central processing unit according to the ISA architecture (ISA). Instruction Set Architecture). This code then usually needs to be connected to certain structures to get a working executable program. Certain assembly software, such as FASM, merely replaces mnemonics and operands, or parameters, with the appropriate instructions in the machine programming language. In this way, we get flat binary execution files that contain (depending on the experience of the individual computer programmer) machine code with exceptional algorithmic efficiency. The program code in the summary is shown in Figure 4.

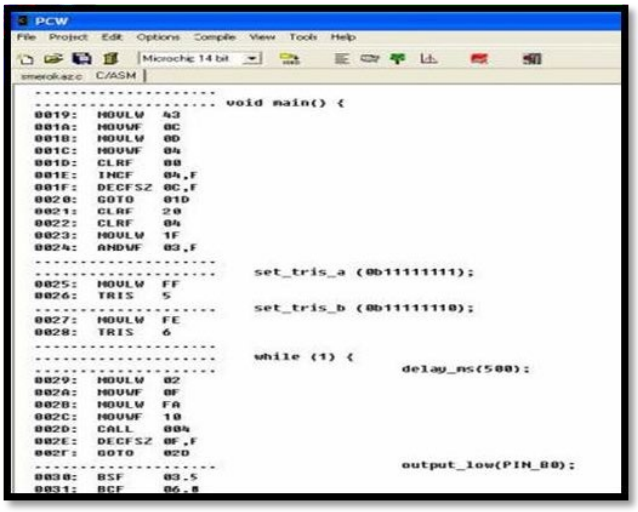


Figure 4: Example of a program in the rollup

A high-level language is a programming language designed to meet the programmer's requirements. It does not depend on the internal machine code of a particular computer. We use high-level languages to solve problems and are often called problem-oriented languages – for example, BASIC was designed to allow beginners to learn quickly, COBOL is used to write business programs, and FORTRAN is used to solve scientific and mathematical problems. Low-level languages, as opposed to high-level languages, strongly reflect the characteristics of the machine code of a particular computer and are therefore also called machine-oriented languages. Unlike low-level languages, high-level languages are relatively easy to learn. Their commands are similar to human language, so the programmer does not need to know in detail the internal structure of the computer. Each command of a high-level language is equivalent to multiple machine commands. High-level programs are therefore more compact than equivalent low-level programs. However, any high-level program must be translated into machine language before it can be run – either by a compiler or by an interpreter. High-level languages are designed to be portable. This means that a program written in a high-level language can be run on any computer that has a compiler or interpreter for that language. The high-level language is C++ shown in Figure 5.

```
*********************************************************************

#case              // Da loči male in velike źrke
#ZERO_RAM       // Briçe RAM po startu programa
/*********************************/

#include <16F84.h>
#USE DELAY (CLOCK=4000000)
#fuses XT,WDT,PUT,NOPROTECT

void main() {

   set_tris_a (0b11111111);
   set_tris_b (0b11111110);

   while (1) {
         delay_ns(500);
         output_low(PIN_B0);
         delay_ns(500);
         output_high(PIN_B0);
            }   // konec while 1

   }     // konec main
```

Figure 5: Example of a programming language in C++

To bring the machine command language closer to the human, let's use the programming language as an intermediate stage. A programming language is not the same as a natural language. Figure 5 shows program code in a C++ environment where you write a program using a set of commands provided by a software tool, such as while. This presents us with a loop, and you can use the include command to include the necessary library.

Natural language has complex rules and extensive grammatical rules.

A programming language is unambiguous and formally describable – it is closer to machine language.

The programming language must be able to:

- **description of the problem** (description of data, results and relationships between them). Example: Product calculation
  - a description of the numbers, the result – their product;

- description of the procedure (description of the sequence of steps that leads us to the result). Multiplication algorithm written in basic steps.

A programming language is a collection of agreed commands, usually in English, that perform a specific action. A higher programming language is not tied to the type of microprocessor. You must select an appropriate compiler that translates the written program from the source code (e.g., C) into the appropriate machine code understandable to the processor (npr. EXE). Each command is translated into multiple machine codes. A programmer needs several programming tools. The source code is typed in the editor.

Editor – is translated into the machine code module with the help of the appropriate compiler.

Compiler – the compiled module, together with the previously prepared modules from the Library, is combined into an executable program using a connector.

Linker – the operation of the program is tested gradually with the help of a debugger.

Debugger – usually all these tools are grouped together in a programming environment with menus, e.g. DEVCPP, CCSC, Visual Studio NET, etc.

2.3. DIVISION OF PROGRAMMING

LANGUAGES Depending on the translation time:

1. Interpreter (INTERPRETER – RUN TIME).
2. The compiler forms the code for the processor on which the compiler itself runs. (e.g. TurboC, Delphi for PC).
3. A CROSS COMPILER runs on a development computer (e.g., a PC) and creates the software code for another type of microprocessor, for various microcontrollers (e.g., PICC, BASCOM, and industrial controllers (e.g., Mitshubishi-Melsec MEDOC or Siemens STEP 7 – http://www.crocodile-clips.com/en/Crocodile_Technology/.)

Depending on the method of programming:

- **Procedural** – BASIC, C-language, PASCAL (suitable for microcontrollers and larger computers).
- **Object-oriented** – C++, Delphi, JAVA (use CLASS object classes).

Depending on how the program is entered:

- **Text-based (textual)** – these are the programming languages mentioned so far.
- **Graphical** – Visual Basic, Visual C, LabVIEW.

These are the languages of the new generation; the programming takes place in the graphical interface. The programmer stacks cubes, modules, and the corresponding program code is formed in the background.

2.4. APPROACH TO PROGRAMMING

We approach the development of the program similarly to the development of other products or products. Planning and development process:

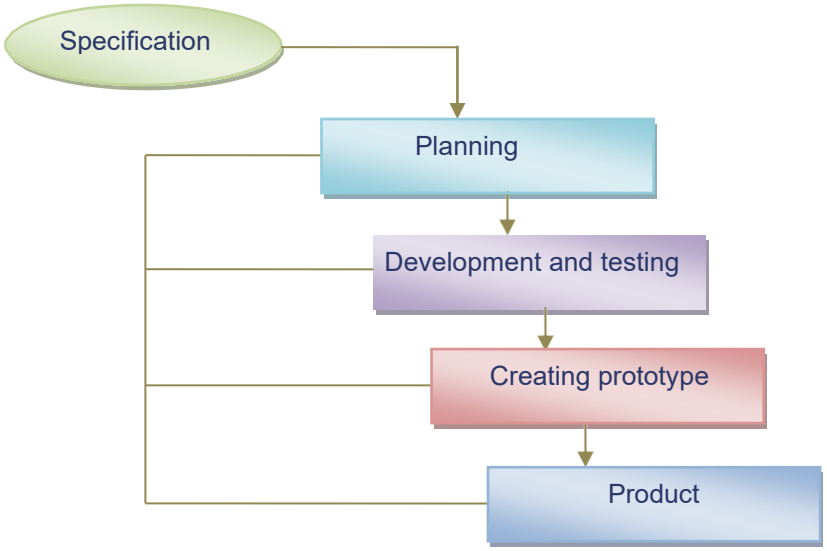


Figure 6: Planning and development process

2.4.1.STAGES OF PROGRAMMING

a) Planning (algorithm, flowchart).
b) Coding (writing in the selected programming language).
c) Translation and testing (the compiler finds spelling errors, and the debugger allows you to find logical errors as well).
d) Program documentation (comments and descriptions for later understanding of what is written and instructions to users of the program).

2.4.2.ALGORITHM

The solution of the task is carried out in steps, in an algorithm.

Definition: An algorithm is a step-by-step description of a procedure by which a problem can be solved.

Example: Create a program that adds two numbers

- Declaring variables a, b, c;
- Enter the first number and enter it in the variable a;
- enter another number and enter it in variable B;
- Add the variables a and b;
- Print the variable C.

2.4.3.FLOW CHART

It is a graphically displayed algorithm, i.e. the course of events described in a human-understandable way. It is used in the planning of programs and also in the description of other activities (e.g. home doctor in book form, Municipal instructions for obtaining permits, etc.).

It is wise to approach the software problem in the way of a flowchart to clarify the implementation step by step and anticipate what the computer will need to do.

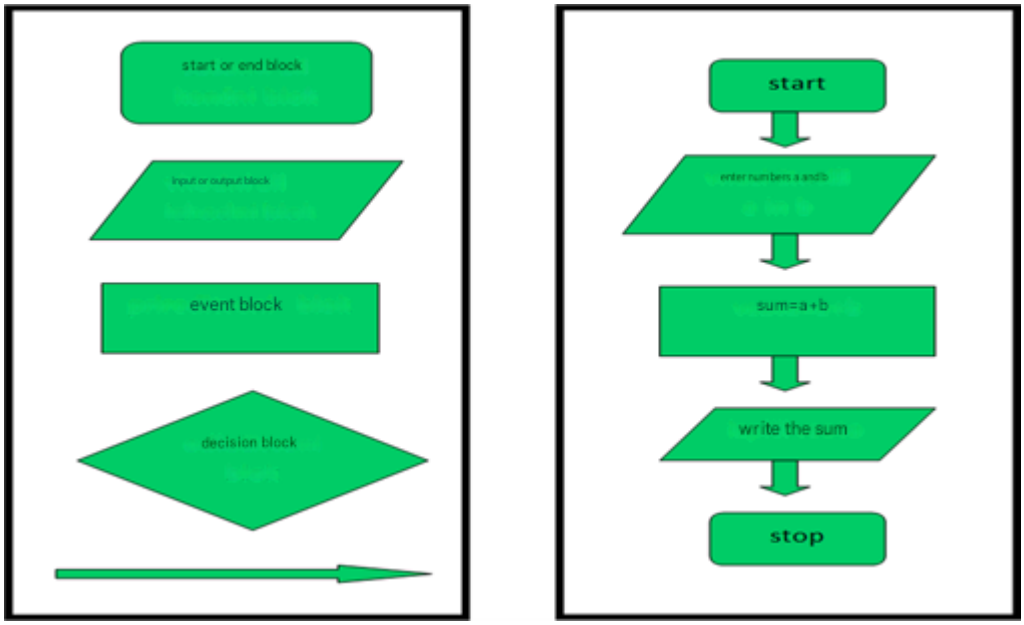


Figure 7: Flowchart blocks

Some learning programs allow you to compose a program directly in a flowchart. The program compiled in this way can be tested or translated into word form, which greatly simplifies learning programming.

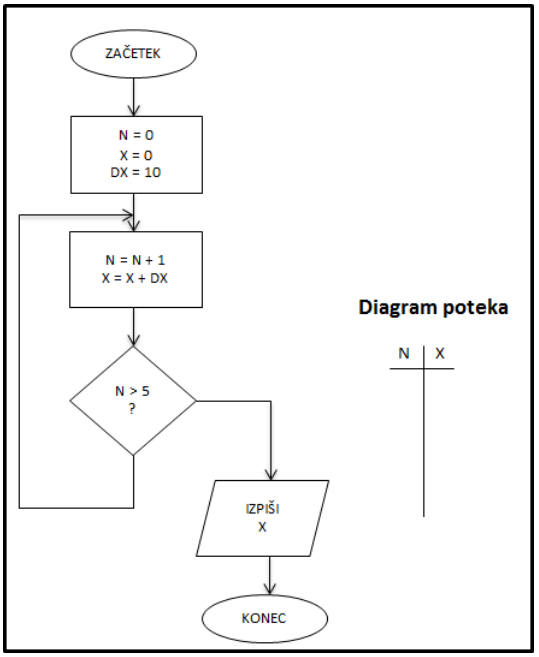


Figure 8: Example of a flowchart

2.5. PROGRAMMING LEGO MINDSTORM AND FLOWCODE MECHATRONIC DEVICES

LEGO MINDSTORM and SIMBOT are well suited for learning programming. Build a car with a built-in microcomputer from LEGO bricks. You write a program for this computer to control the engines and take into account the signals from the sensors.

Figure 9: Lego Mindstorm
Source: http://mindstorms.lego.com/en-us/Default.aspx

One of the programming options is Flowcode.

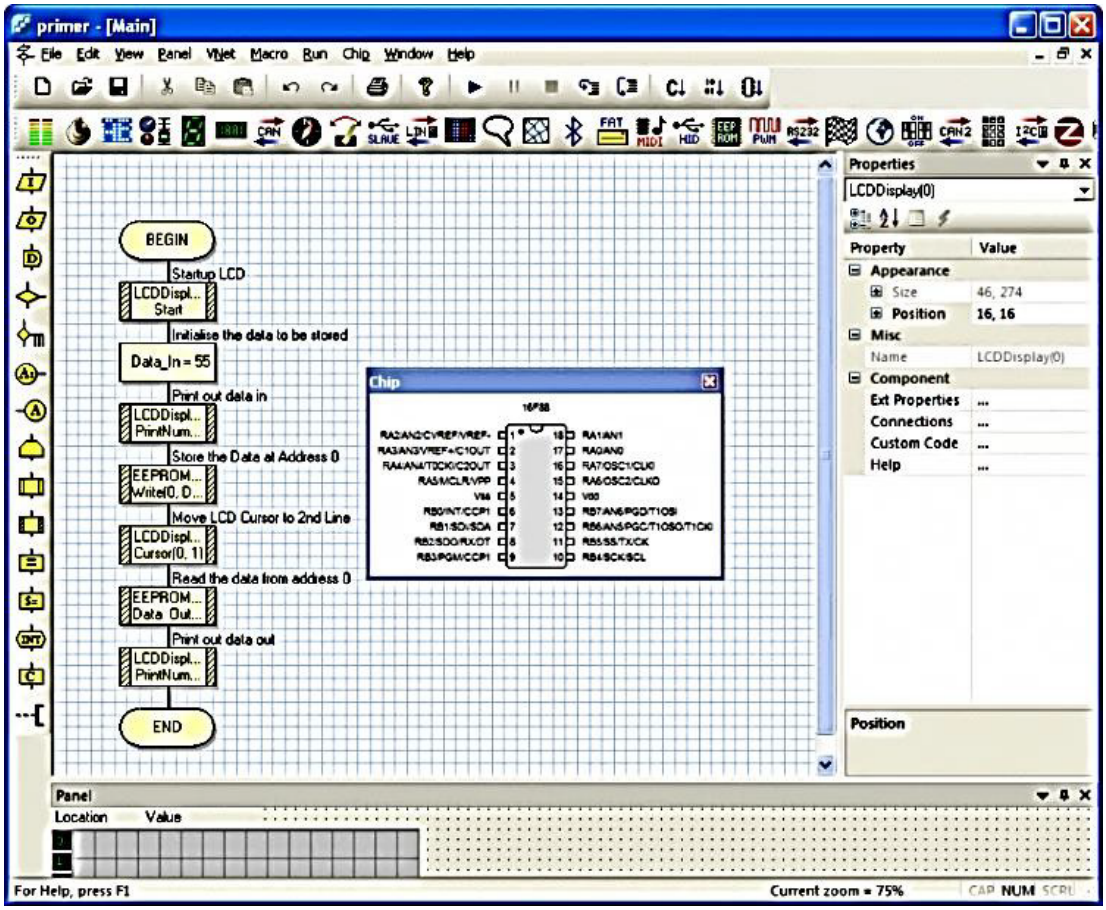


Figure 10: Example of a program in Flowcode
Source: http://www.imagesco.com/microcontroller/flowcode-compiler.html

2.6. CROCODILE TECHNOLOGY SIMULATION PROGRAM

The Crocodile Technology simulation program is suitable for learning how to program the PIC microcontroller. In the program, you create an electrical circuit diagram with elements from the library, and in the next step, you can also build a program flow diagram. In the diagram, you can use any program commands, conditional jumps, loops. The advantage that allows us this way of programming is certainly greater transparency and easier presentation of what is happening in the program. When the simulation program starts, the activity of each input or output is also visible.

The given program also allows you to measure signal values with virtual meters from the plot. For example, you can observe what is happening at the output of the microcontroller.

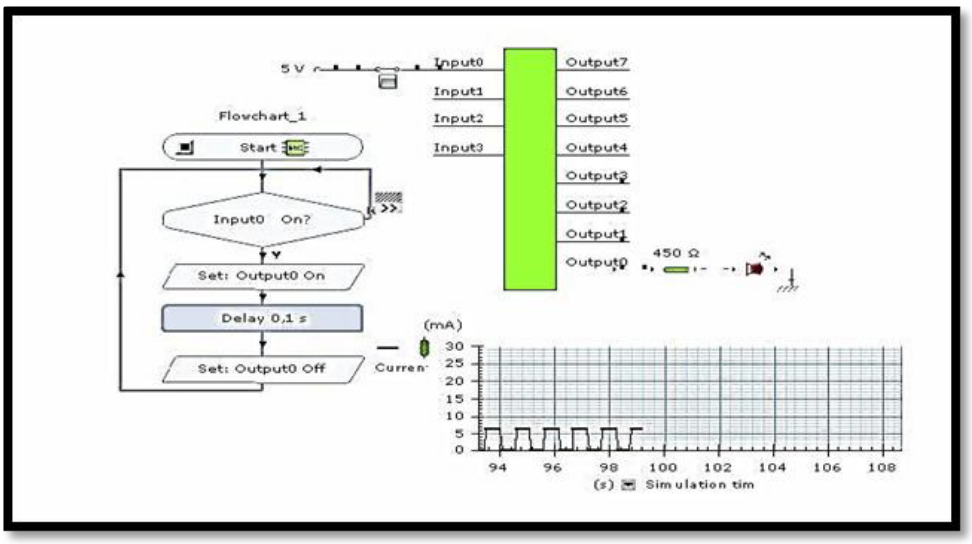


Figure 11: Example of a software environment for programming mobile robots
Source: http://www.crocodile-clips.com/en/Crocodile_Technology/

The program can also be exported to BASIC or entered directly through the programmer into a real chip.

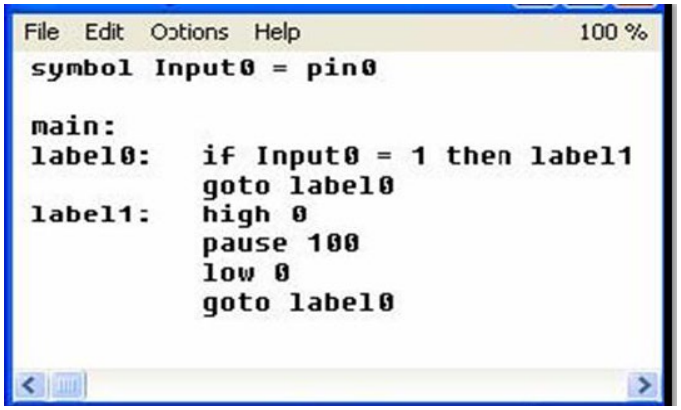


Figure 12: Example of program code in Basic
Source: http://www.justbasic.com/learnmore.html

2.7. MICROCONTROLLER PROGRAMMING – PLC

Also, the programming of industrial controllers can be graphically and quickly understood. The PLC software tools support three programming methods:

- via the relay electrical control scheme (LADDER),
- with elements of digital electronics – block diagram (FBD),
- by verbal commands – instructions (INSTR),
- and some additional graphic mode.

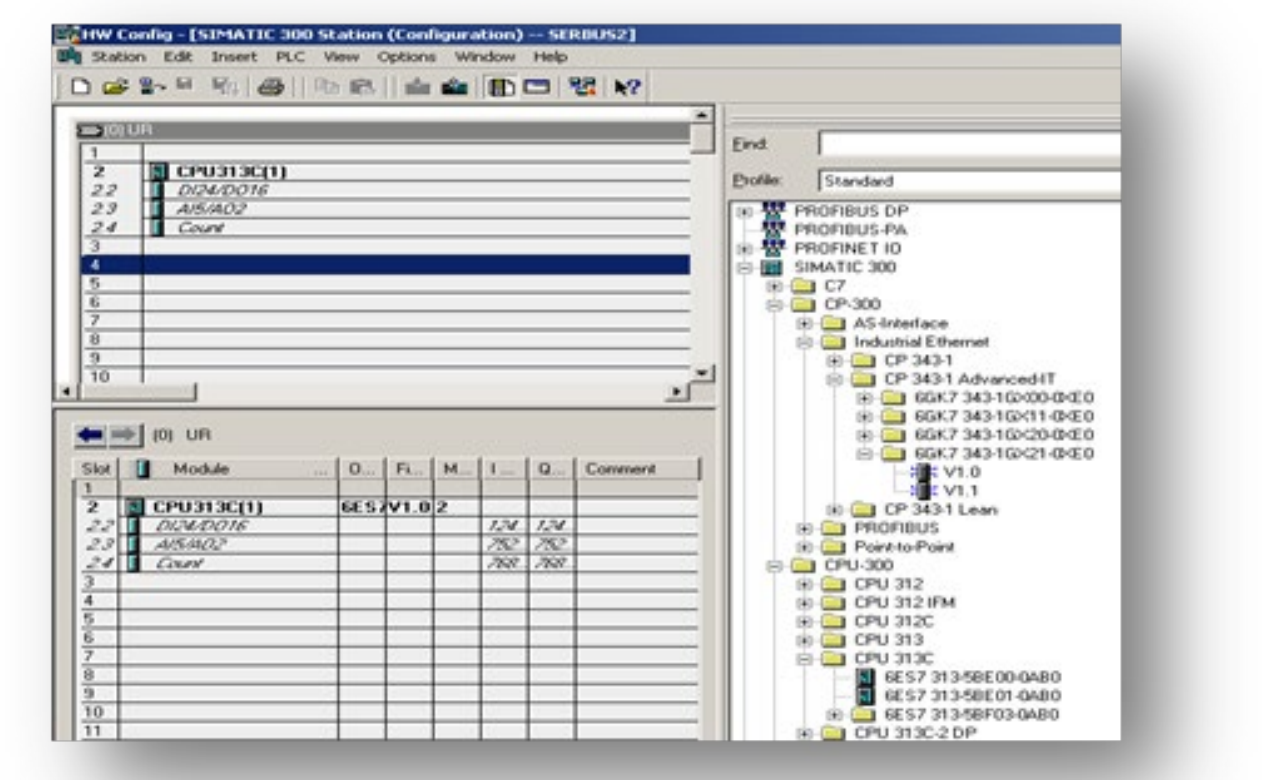


Figure 13: Example of PLC programming languages
Source: http://support.automation.siemens.com

Such software tools enable monitoring and testing of the program after loading it into the controller, and some also allow simulation of operation on a PC (off-line).

2.8. VIRTUAL INSTRUMENTATION PROGRAMMING – LABVIEW

A PC can be used as an industrial controller in the automation of measurement processes. For this purpose, along with the apparatus equipment (measuring interfaces for PCs, etc.), graphical programming environments are also obtained. In Slovenia, the company's equipment is widely used

National Instruments and their LabVIEW software tool (http://www.ni.com/labview/applications/).

This software tool is designed to automate measurements. You need measurement modules and interfaces that you build into your computer (DAQ cards), and these interfaces transmit analog and digital signals between the meter and the computer. Classic measuring instruments that have a computer connection are also useful, as they can be outdoor units.

The creation of programs is simple and transparent. Two program windows are open at the same time: the front panel (Panel), on which you apply and draw displays and control buttons, and the block scheme (Diagram), in which you connect elements from the front panel with

wires in the background and enable data flow. From a rich library of functions, you select the appropriate blocks to process the measurement data, which you link to the schematic.

The displays and control buttons for the VIM are applied to the front panel (VI-Virtual Instrument).

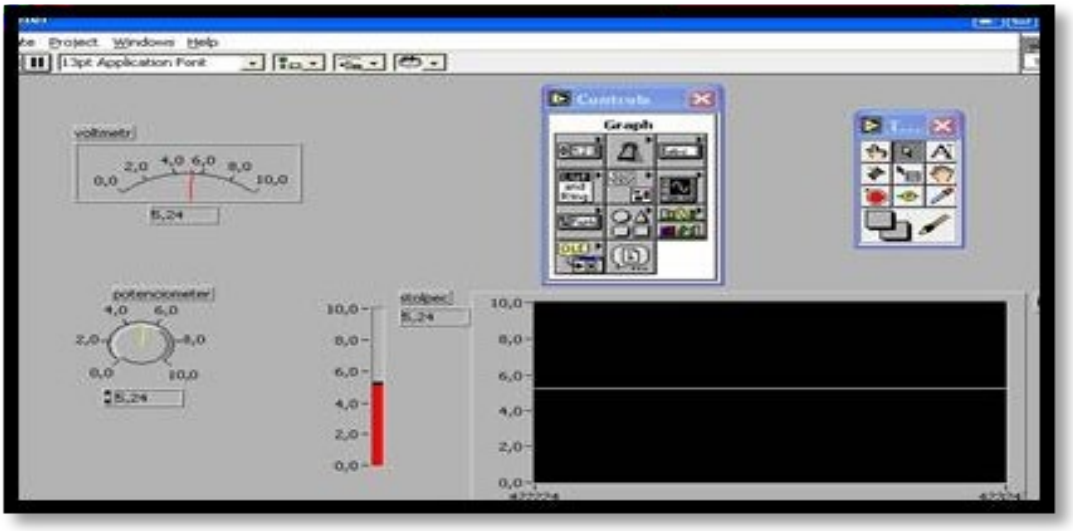


Figure 14: Demonstration of programming in LabVIEW
Source: http://www.ni.com/labview/

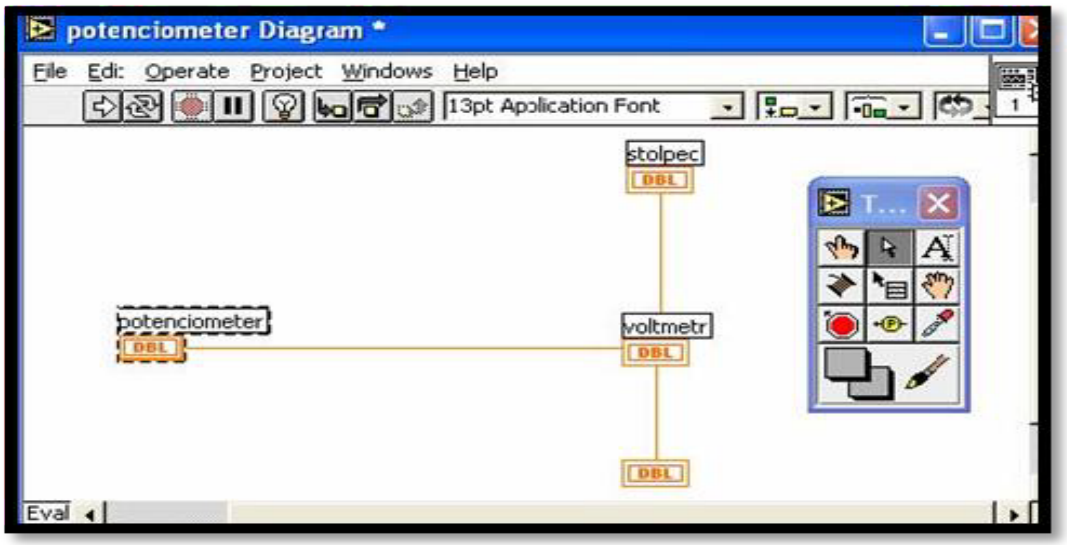


Figure 15: Diagram
Source: http://www.ni.com/labview/

In the background, in the program window (Diagram), you connect the sources and consumers of the data stream and build program loops.

You can compile the program into an executable (exe) file or run it directly on the development environment. The computer reads the measured data from external measuring instruments or measuring interfaces and, based on them, decides on the appropriate action via the control outputs from the PC, writing to a file on disk, reporting on the screen or via the Internet.

You can select work tools from the TOOLS palette, which is accessible in both windows. When building a front panel, you have the CONTROLS palette on the right mouse button, and in a block scheme, the FUNCTIONS palette in the same place, in which the function library is located. Good help in English makes it easier for us to work and detect errors. The created program from both windows is saved under the name

*.VI (virtual instrument). You can help yourself learn with a multitude of examples that have been made.

We start the program with  on the workbar.

If we want it to be repeated automatically, then with  .

For a better understanding of how it works, we include a display of the data flow with  .

2.9. PROGRAMMING OF NUMERICALLY CONTROLLED MACHINES – CNC G-CODE

A CNC (Computer Numerical Control) machine is a lathe or similar metal-processing machine controlled by a computer controller. G codes are the standard for numerically controlled machine tools.

CNC programmers write programs from technological drawings and feed them into the machine controller. Good mechanical CAD programs can automatically create a G-code from a drawn 3D object to create an object on a CNC machine.

Figure 16: CNC Machine

The figure below shows an example of code for programming a CNC milling machine controlled by a PC. Under each of the objects, the code for the construction of the object is also indicated.

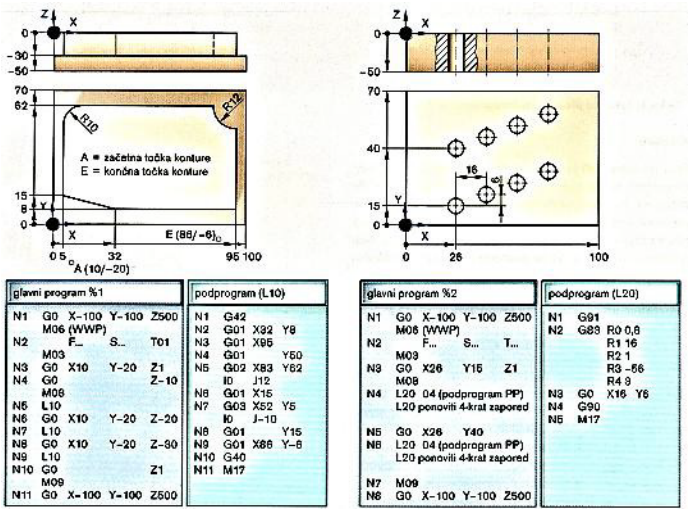


Figure 16: Example of CNC code with included subroutines
Source: J. Bartenschlager, MECHATRONICS 2009

2.10. ROBOTIC PROGRAMMING LANGUAGE: INDUSTRIAL HOOK ROBOTS

The simulation program allows you to learn movements and actions in the simulator. You can build a robot cell and write a robot program. You can then test it and transfer it to the robot controller. Before building a real robotic cell, it is good to make a simulation and present all the boundary operating conditions in it.

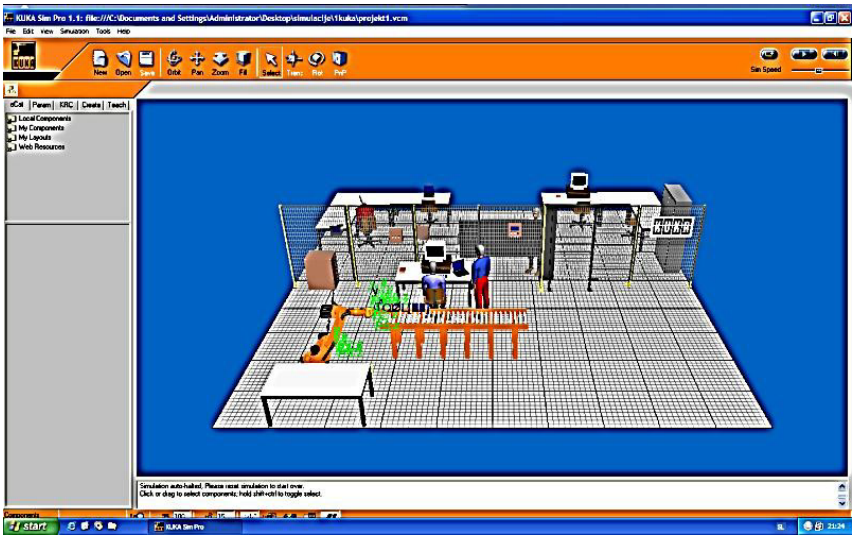


Figure 17: KUKA Sim Pro simulation program
Source: http://www.kuka-robotics.com/en/products/software/

In addition to the development programming environment, you can also connect the simulator to Office Lite, which allows you to simulate a programming device.

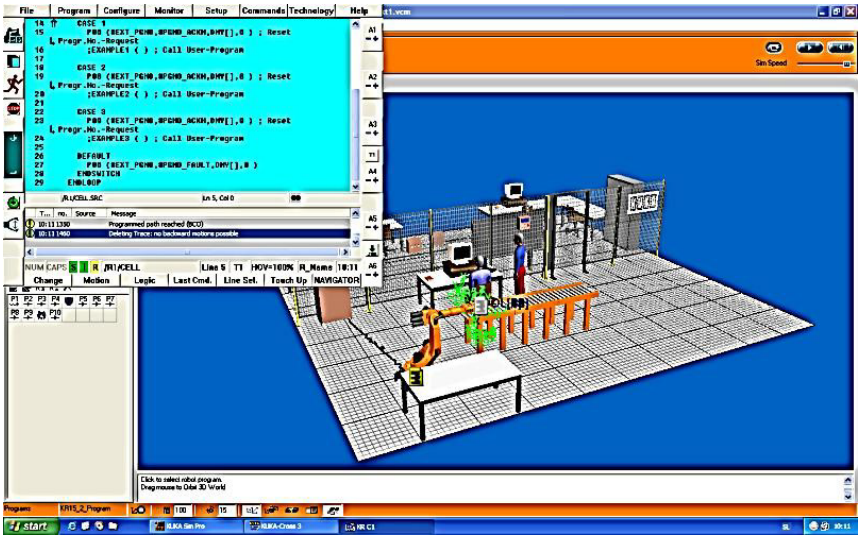


Figure 18: Simulation program
Source: http://www.kuka-robotics.com/en/products/software/

The image above shows the connection between the simulation program KUKA Sim Pro and the simulation subprogram of the Office Lite operating panel.

In the case of connecting the Kuka Sim pro software package and Office Lite 4.1, you can perform the same actions on a PC as on a real robot. This is shown in the image above. You can perform movements, loops, set speeds and transmit the finished program directly over the network to the robotic KRC controller.

Figure 19: Motion Recording Programming Device (Teach Box KR C2)

Summary:

Programming languages can basically be divided into three parts, namely machine code, assembly and higher programming languages. The machine code of a program consists of sequences of executable machine commands that are executed on personal computers by a central processing unit.

Collection (Assembly Language, Assembler) is a low-level second-generation programming language (the first generation is machine code) that is written with mnemonics. In general, mnemonics are readable versions of binary sequences (zeros and ones) that need to be compiled in order to obtain a code that is understandable to the central processor.

A high-level language is a programming language designed to meet a programmer's requirements. It does not depend on the internal machine code of a particular computer. We use high-level languages to solve problems and are often called problem-oriented languages.

Before starting programming, it is definitely necessary to work out the problem itself, which we will solve with the program. We do this with an algorithm, where we break down the problem into individual steps and it is easier to solve.

Questions:

1. What does the term "computer program" mean and what does "programmer" mean?
2. What are the features of a programming language and what does it enable?
3. What software tools does a programmer need and what are they for?
4. Divide the programming languages and give some examples.
5. What stages does a programmer have to go through when creating a program?
6. What does the word algorithm mean?
7. Write an algorithm for a coffee maker and explain it.
8. What symbols are used in a flowchart and what do they mean?
9. What program writing methods are used in programmable logic controllers?
10. What does a program for a CNC machine look like? Write at least 3 commands.
11. How are robots programmed? Describe an example.

3. BASICS OF PROGRAMMING LANGUAGES

In the chapter PROGRAMMING LANGUAGES, we will get acquainted with programming languages and their basic commands. The commands are illustrated with examples.

Today's computers only understand bit language, which is also called machine code. So if you want our computer to perform a task or task, you have to give it in a form that it understands, i.e. in ones and zeros. However, since it is not the most natural for us humans to think in bitwise language, there was a need for some higher, more human-understandable language, which can then be translated into machine code. A programming language allows us to give commands and tasks to the computer using everyday expressions, such as "A" and "A".

"Read and write." One of the development programming tools is the C++ language, which is versatile in use. It can be found in the programming of microprocessors, the creation of programs for printing travel orders, and it is increasingly present in the development of web applications.

3.1. COMMANDS IN C++

A program is a sequence of commands, and programming is writing commands for the processor. The commands are different in each programming language. This chapter will describe commands in the C++ programming language.

In addition to C++ and C#, we also know programming languages such as Visual basic, Delphi, Java, SQL, Python, etc.

A brief history of C++:

In the late 1970s, Bjarne Stroustrup began experimenting with extending the widely used C programming language. The C++ programming language is an extension of the C programming language, and Bjarne Stroustrup drew some additions from existing languages such as Simula67 and Algol68. The name C++ first appears in 1983, and in the second half of the eighties, C++ underwent radical changes.

3.2. COMMANDS IN C++:

Main program (block of commands)

All programs must have a main program (main), otherwise they may not be executed. This is the core that you will start programming into. Include the main() method in each program, followed by curly brackets {}. You need to specify the beginning and end of the program in the program, and it is appropriate for the block of commands to be uniquely marked.

```
int main() (start of the main program)
{                          - the beginning of the block
cout << "This is a program in c++" ; return 0;
}                          - end of block
```

On the left side, you can see a program that displays a sentence on the screen – this is a C++ program.

LIBRARIES

We know a lot of libraries. Booklets are files where the machine code of functions is written. When writing programs, you need to include the libraries used in the program, e.g. *(#include<iostream>)* and include an input output stream of data. The most commonly used libraries are listed below:

```
#include <iostream> I/O commands (library used only in C++)
#include <stdlib.h> Standard Commands
#include <string> To make it easier to work with strings, the library is valid only in C++
```

COMMENT

The comment is intended for the programmer to facilitate the understanding of the program. In the case of more complex programs that are not consistently commented on, there are great difficulties in the event that you want to change or correct anything. A feature of the comment is that the compiler skips it. You write your comments briefly and concisely; They can be single-line, denoted by (//) or multi-line, denoted by (/* .....*/).

VARIABLES

Variables are a piece of memory, separated by type and size. The type of variable tells us what all these beings occupied by a variable actually mean. Whether it's a letter, a number, or something else.

RULES FOR NAMING VARIABLES:

- The first character should be a letter, not a number.
- The remaining characters can be letters, numbers, or underscores "_".
- There are no spaces in the names.
- The size of the letters is important. You have to write them down exactly the same every time. For example,
- *A car* is not the same as *a car*.
- A special feature important for Slovenes – do not use murmurs (č,ţ,š).
- It is customary to start the names of variables with a lowercase letter, and for compound words, start each subsequent one with a capital letter. Some examples: *ageChild, speedOf the car*, *sizeFeet.*
- The names of the variables should be meaningful. Use words that tell the meaning of the variable, e.g. *diameterCircle*, *lengthBars*.

Pseudocode for variable declaration:

TYPE IME_SPREMENLJIVKE;

Pseudocode for variable definition:

IME_SPREMENLJIVKE = NEKA_VREDNOST;

A variable must first be declared, only then can it be defined. These two steps can be combined:

TYPE IME_SPREMENLJIVKE = NEKA_VREDNOST;

The minimum size of a variable that can be addressed in the C++ programming language is one (1) byte, i.e. eight bits, which amounts to 256 (2^8) different states. Such a type is called **char** (character) in C++, and integers **can be entered into it**. In addition to it, we also know the integer **int**, which usually occupies 4 bytes. There are also smaller and larger integer types that occupy 2 or 8 bytes, but their declaration varies from compiler to compiler. You can also use signs when defining variables.

Example:

```
/*
Examples of declarations and definitions of integer variables.
*/
Declaration of Char A;
int b;
Declaration and definition, use of the sign
char c = 10; int d = +10;
char e = -10;
int f = +10;
```

The program on the left shows t
declaration of variables. It begins w
the display of a full-line comme
followed by a declaration of variabl
e.g. variable b is of type integer.

C++ also knows **type modifiers** (only for integer types): *short, long, unsigned, signed.* **Short** and **long** are used in conjunction with **the int type**, and specify the size of the int type (depending on the individual compilers).

Pseudocode:

MODIFIER TYPE IME_SPREMENLJIVKE;

TYPE MODIFIER IME_SPREMENLJIVKE = NEKA_VREDNOST;

Example:

```
// uporaba short ter long

shortint majhno = 100;   // the magnitude of the variable should be 2 bytes

longint veliko = 10000; // The size of the variable should be 8 bytes
```

When using modifiers, you can omit **the type int** because the compiler determines that it is an integer type **of int.**
The signed **and** unsigned **modifiers** determine whether our integer types also have a sign, or whether you can also write negative numbers with them. All basic integer types can also write negative numbers (default mode). For example, type **char**. Since it is 1 byte in size, you can represent 256 different possibilities (states) with it. So you can use it to write down numbers from

128 to +127 (also 0). If **you use** the unsigned modifier on **the char type**, the negative numbers are eliminated, so you can write numbers from 0 to 255 in it. This also applies to all other integer types.

Example:

```
// use of modificators signed and,unsigned

char  a;       // -127 do +128, signed as default value
signedchar b; // -127 do +128
unsignedchar c;  // 0 do 255
int d;          // - 21474836648 do +21474836647
unsignedint e; //    0 do 42944967296
```

You can also write down letters and associated symbols. You have more options. Somehow the most used are **ASCII** and **UNICODE** notation. The symbol notation according to the ASCII table is one byte in size, while UNICODE uses 2 bytes for the notation. The UNICODE notation is particularly suitable for developing downloadable programs for foreign markets (or languages). ASCII, on the other hand, provides us with a simple and, above all, less memory-intensive record. The ASCII format uses a so-called code table, which depends on the system settings (language and country, hence the noise problems). UNICODE, however, uses a different notation, namely each symbol has its own unique value. Letters and symbols are written as numbers in the computer, only when we start to "look" at them or print them out, they get their true shape, which we understand. So, for example, the capital letter A has a value of 65, the letter B has a value of 66, and so on.

Example:

```
char   a ='A';       // ascii
wchar_t b ='B';   // unicode
char   c = 65;     // ascii, used as ascii value ('A')
```

In addition to these basic types, C++ also knows the logical **bool** type, which is identical in size and structure to **the char** type, and the so-called null void type.

Variable names must not have the same names as keywords.
A list of all keywords that should not be used for variable names:

asm, auto, break, case, catch, char, class, const, continue, default, delete, do, double, else, enum, extern, float, for, friend, goto, if, inline, int, long, new, operator, private, protected, public, register, return, short, signed, sizeof, static, struct, switch, template, this, throw, try, typedef, union, unsigned, virtual, void, volatile, wchar_t, while, etc.

Variable names should also not begin with a number (e.g. *100abcd*), but may contain digits in any other position (e.g. *abcd123).*

DECISION (IF SENTENCE)

As in everyday life, where constant decisions are required, you also use a decision statement (*if) in programming; for example,* if variable a is greater than variable b, write the value c. In a conditional sentence, we have two values of the solution, namely the value true *and* the value false. A condition is a statement (record) that can contain parentheses, values of different types, variables, constants of different types, and operators. Operators are special symbols that you can use to combine variables or values.

The first group of operators represents mathematical operations such as addition, subtraction, multiplication and division.

Table 1: Operator Group

Operator	Name	Example	Put into words
+	Plus	1 + 2	One plus two
-	Minus	7 – 3	Seven minus three
*	Times	5 * 2	Five times two
/	Shared	9 / 3	nine divided by three

Relational operators compare two values with each other to determine whether they are the same or whether the first is greater or less than the second.

Table 2: Group of relational operators

Operator	Meaning	Example	Put into words
==	is the same	a == b	A is equal to B.
!=	Not the same	c != d	C is not the same as D.
<	less than	x = 10	x is less than 10
>	greater than	y = 0	y is greater than 10
<=	less than or equal to	age <= 25	Age less than or equal to 25 (includes 25)
>=	greater than or equal to	height >= 180	height greater than or equal to 180 (includes 180)

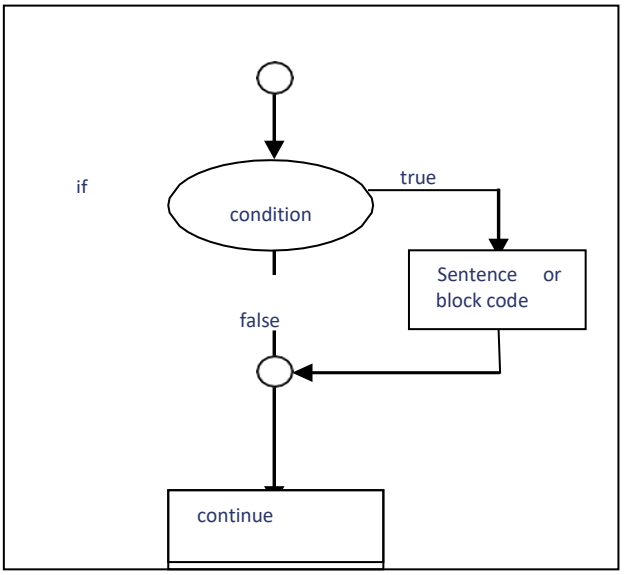
The conditions are divided into:

- Simple
- composite (must contain at least 1 logical operator).

Commands written within the framework of a conditional clause are executed if the given logical condition is met. Use the IF conditional statement when you want some code to be executed only if the condition is met.

The general form of an IF sentence is as follows:

```
if (pogoj)              // YES action
{
        true
} else                  // NO action
{
        false
```



If the condition is approved and the value is true, the action will be xecuted. In the event hat the condition is not satisfied, the program will continue without a program action.

Example of a program with if: (the program counts and prints numbers from 1 to 10):

```
#include<iostream>
using namespace std; int
main()
{
int stevec; stevec=1;
nazaj:
cout<<stevec<<endl;
stevec=stevec+1;
if(stevec>10)
{
system("Pause");    return
0;
}
else
{
goto nazaj;
}
}
```

The program on the left counts and prints numbers from one to ten. First, include the input output stream (iostream) in the program, then the main program starts running. You declare the variable counter and set it to a value of one. The value of the numerator variable is incremented by one until the value of the numerator variable is greater than ten.

PRINTOUT

In C++, we know the **cout command,** which is used to print some text or numbers on the screen. Similar commands to coutu are *izpisit (*prints text) *and izpist (*prints numbers). You use these two commands in "subroutines", which will be described a little later.

Building:

```
cout<<" Neko besedilo ki se izpise na ekran "<<endl

Primer izpisa na ekran:


#include<iostream>
using namespace std;
int main()

{

cout<<"To je izpis na ekran (cout)"<<endl;
system("pause");

return 0;

}
```

READING

To read in C++, you use **the cin** and **getline commands.**

Cin reads to the first blank character and is therefore not suitable for the multicharacter type. This problem is solved **by the getline command, which is in the string library.**

You have different options, namely:
getline(cin,ime_prostora); read until you press enter – to the end of the line; getline(cin, ime_prostora,character); read until you encounter the correct character; getline(ime_prostora,100); you read as many times as it is written (in this case, 100); getline(ime_prostora; 500,b); you read until we get to B, it can be a maximum of 500 characters, no more, in which case there are more than 500 characters, you can read up to 500 characters, or until we get to character B).

Example of a program:

```
#include <iostream> using
namespace std;

int main () {
char name[256], title[256];

cout << "Vase ime je: ";
cin.getline (name,256);

cout << "Vas najljubsi film: ";
cin.getline (title,256);

cout << name << " vas najljubsi film je: " << title;

system("pause");
return 0;
}
```

CHOICE

Choice – you choose from several variants.

In C++, there is a selection command called **switch/case**. Structure of the switch case:

```
switch (expression)
{
case value A; block of commands A
break;
case value XY; block of XY breaks;
default;                no mandatory default
action;
```

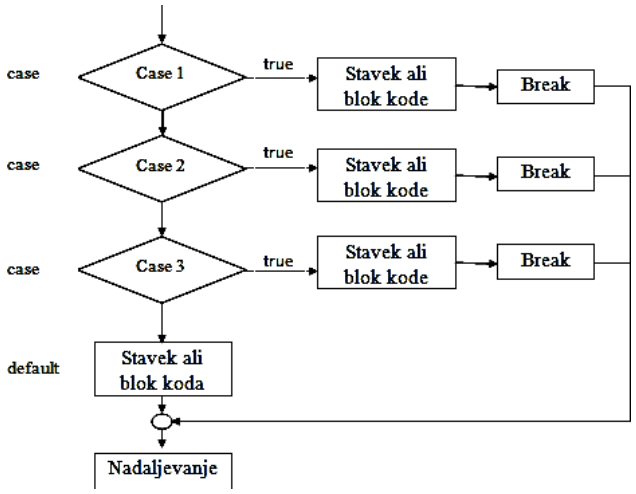


Figure 21: Display of the choice command (switch/case

Command:
break – skip the entire structure;
continue – one step, the current action is interrupted;
Goto – Use this sentence when you don't want sentences in program code to run in order (avoid the goto command because it can lead to unstructured code!!).

Example of a program with switch:

```
#include <iostream>
#include   <stdlib.h>   using
namespace std;

int main()
{
   int   a;
nazaj:

   cout<<"1.    Ime"<<endl;    cout<<"2.
   Priimek"<<endl;              cout<<"3.
   Razred"<<endl;

 cout<<"Izhod iz programa"<<endl;
   cout<<"Vnesi stevilko za prikaz podatkov: "<<endl; cin>>a;

   switch(a)
   {
         case 1:
           cout<<"Alen\n";
                                 break;

                                 case  2:  cout<<"Korosec\n";
                                 break;

                                 case            3:
                                 cout<<"2.a\n";
                                 break;

                                 case 9:
                                 cout<<"Izhod iz programa\n";
         system("pause");
         return 0;

         default:
         cout<<"Stevilo ni veljavno! Vnse drugo"<<endl;

   }
   system("pause");
   system("cls");
   goto nazaj;
```

ARRAY

Fields are groups of identical objects or data. They are numbered from 0 onwards. A group of integers (int) will be represented. Note the format: {12, 7, 32, 15, 113, 0, 7}.

Let's see how these values would be written on a computer, and let's give the current numbers of each data in the group:

Tabela 3: Display values written in a field

Index	0	1	2	3	4	5
Values	12	7	32	15	113	7

The current number of data in such a field is called an index. To get an individual element of the field, you need to specify the name of the field, and write the index number after the name in square brackets [ ], after the name. Let's say you give such a field the name *of the pointCompetitors*.

A field is a data structure that is used when you have several variables of the same type and for the same purpose.

TIP_IME[n] (field declaration)
n - index - number of elements
n-T The index is prohibited in C++.

Example of a program with an array: (the program prints numbers from 0□9)

```
#include<iostream>
using namespace std;

int main()
{
int polje[10]; int
i;

for(i=0;i<10;i=i+1)
{
polje[i]=i;
}

for(i=0;i<10;i=i+1)
{

cout<<polje[i]<<endl;
}

system("pause");    return
0;
}
```

The program on the left shows the use of the field.
An array of type integer and a variable i, also of type integer, are declared.

FOR THE LOOP SAYS:
As long as i is less than ten, increase i by one, or make an increment of i, and write the values in the array. Use the cout command to extract values from an array.

LOOP

A loop is a basic command that you use when a piece of code is repeated (sequentially) for the same purpose.

Theoretical description of loops:
Each loop has a head and a body. In your head, you decide whether to repeat the command or not, and in the body there are commands to repeat. There's a condition in the header, and there's at least one variable in the condition that you can influence.

FOR LOOP

The For loop repeats until the condition is true. It is suitable when the number of repetitions is known in advance (it is not performed if the condition is not met).

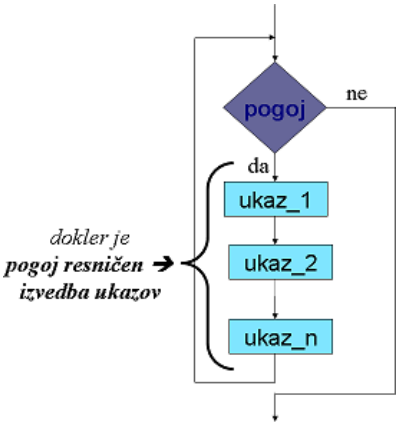


Figure 20: Example For loop

For loop shape:

```
for ( ;pogoj; )

{
Blok ukazov
Vpliv na spremenljivke, ki se nahajajo v pogoju
 }
```

Example of a program with a for loop (the program prints the name on the screen 10 times):

```
{
 int index;

for(index = 0 ; index < 10 ; index = index + 1)
cout<<"Alen"<<endl;
system     ("pause");
return 0;
}
```

The program on the left shows how the for loop works.
For loops are basically a condensed notation for:

- initiation of the meter (or other condition),
- setting and checking the condition, and
- increasing the counter.

WHILE LOOP

While Loop Shape:

```
while (condition)

{

Command Block

}
```

While the loop continues to run as long as some condition is true. When the condition becomes unfulfilled, the loop is broken. So the loop does exactly what its name suggests.

Example of a program with a While loop (the program prints the name on the screen 10 times):

```
#include <stdio.h>
#include   <iostream>   using
namespace std;

int main() /
{
int count;

 count = 0;
 while (count < 10)
{
 cout<<"Alen"<<endl;  count  =
 count + 1;
 }
system     ("pause");
return 0;
}
```

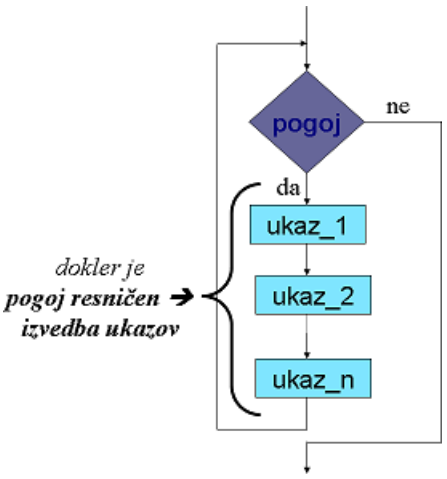


Figure 21: Example of a While loop

DO WHILE LOOP

It is recommended to use this loop when we have the data and based on it we decide whether to repeat it or not. The loop repeats as long as the condition is true. A Do While loop, unlike a While loop, is executed at least once, even if the condition is not met.

Do While Loop Shape:

```
do
{
}while();
```

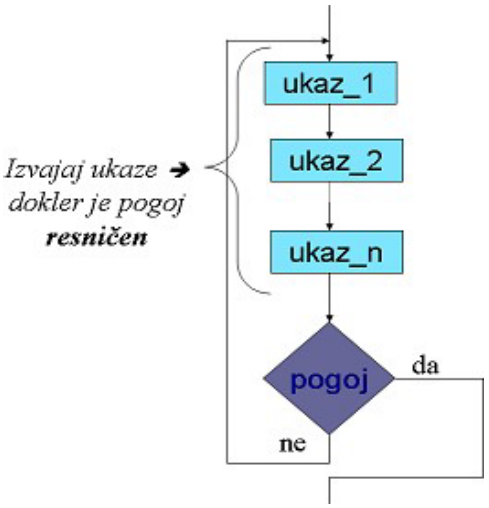


Figure 22: Example of Do While loop

Example of a program with a Do While loop (the program prints the name on the screen 10 times):

```
#include <stdio.h>
#include <iostream>
using namespace std;

int main() /
{
int i;

   i = 0;

   until
   {
      cout<<"Alen"<<endl; i = i + 1;
   } while (i < 10);
system("pause"); return 0;
}
```

A Do While loop is used in a given program. The loop is executed as long as the variable i is less than 10. Each time the loop is executed, the variable i is incremented by 1.

SUBPROGRAMS

When the main programs are executed, the calling of subprograms occurs. For the most part, subroutines are used when a piece of code is repeated in different places, and if the code is repeated sequentially, but each time with different data. A feature of a subroutine is that it must work on its own and not depend on other programs in the code. Subroutines are divided into procedures and functions (procedure – the name of the subroutine is not used to transmit results, function – the name of the subroutine is used to transmit the result).

Structure of the sub-program:

* HEADER (The header contains the type of subroutine and the name of the subroutine).

Parameters – are variables used only in the subroutine (in the header).

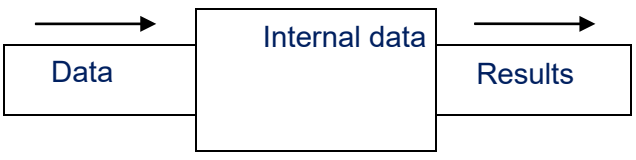
* BODY

- starts with the block start character { and ends with the block end character };
- There is code in the body (in the code can be everything we know, except we are not allowed to start a new subroutine).

Division of sub-programmes:

- Procedure (we do not use the **name of the subprogram** to transmit results);
- Function (we use the name of the subprogram to transmit the results).

To transfer data to a subroutine:



a) Global variables.

b) Transfer via parameters:
   - by value (data changes are not preserved),
   - by reference (changes to the data are preserved). The reference affects the location of the call.

c) Via file;

d) We can't stand anything and just read.

Transfer of data from the sub-programme:

a) Global variable:
   - Any change shall also be maintained outside the sub-programme.

b) Use of parameters:
   - Any change in a parameter affects the variable that gave the data to that parameter.

c) Use the file.

Each programming language has a special command so that we can use the function name to transfer the result outside the subroutine. In C++, this is **the return command.**

Summary:

A program is a sequence of commands, and programming is writing commands for the processor. Commands are different in each programming language, and this chapter describes commands in the C++ programming language. We know a lot of libraries for different applications. Booklets are files where the machine code of functions is written and must be included in the program if we want to use these files in the program. Variables are a piece of memory, separated by type and size. The type of variable tells us what all these beings occupied by a variable actually mean. Each loop has a head and a body. In our heads, we decide whether to repeat or not. In the body, however, there are commands for repetition. There's a condition in the head, and there's at least one variable in the condition that we can influence.

QUESTIONS:

1. List the properties of the main program.
2. What are libraries?
3. What does the type of variable tell us?
4. What happens if we use the unsigned modifier on the char type?
5. Explain the differences between ASCII and UNICODE modifier records.
6. Write down an example of the printout on the screen.
7. Write a program that will require you to enter two numbers and will calculate and print the sum, product, difference, and quotient.
8. What is a loop? List some loops and describe the Do While loop.
9. Draw a flowchart and write a program that will require you to enter 10 numbers and will plot them in order from largest to smallest.
10. What types of variables are known in the C++ programming language?
11. Comment on the program below:

```
#include<stdio.h>
int main ()

{
   int a,b,c;


   a=5;
   b=6;
   c=7;
printf(˝Spremenljivke: %d, %d, %d, a, b, c =;
```

4. PROGRAMMABLE LOGIC CONTROLLER (PLC)

Over the past decade, the field of control technology has changed considerably. In the past, we used only hard-wired controls, but in modern times the so-called software controllers are increasingly used (the English abbreviation for programmable logic controller is PLC – Programmable Logic Controller).

The biggest advantage that controllers have compared to all older control systems is that they have the ability to program operation. Programming can be used to determine the operation of the controller, thus enabling great flexibility of the system, as the programming possibilities are almost unlimited. Classic controls contained a fixed logic – it was very complicated and, above all, expensive to change or correct.

The Programmable Logic Controller (PLC) section shows how a PLC is built and how it works. The control of the device (in open and closed loop) is also presented. We are presented with sequential controls. The Siemens controller and hardware are presented in more detail. A few words are also devoted to software and programming.

A PLC is a digitally functioning electronic device that, based on commands stored in programmable memory, performs logical, sequential, temporal and arithmetic operations, thereby directing various devices and processes through digital and analog inputs and outputs.

4.1. BUILDING

The input unit is a unit for capturing process variables and the variables of the operator, who transforms and adjusts their signals and transmits them to the processing unit. Galvanic separation of signals is important here.

SIGNAL PROCESSING UNIT

a) In permanently wired systems, this is often a circuit with electronic logic, time, count, computational ... elements that eliminate the steering function;
b) in free-programmable controllers, this is the central processing unit (microprocessor), together with the memories and peripherals; A control program is stored in the program memory, which eliminates the control function.

The output unit is a unit for transforming, adapting and transmitting control signals to the executive members of the process. Again, galvanic separation of signals is important here.

A power supply unit is a unit that supplies a supply voltage to the control gear. With PPK, standard versions with a DC voltage of 24 V are used.

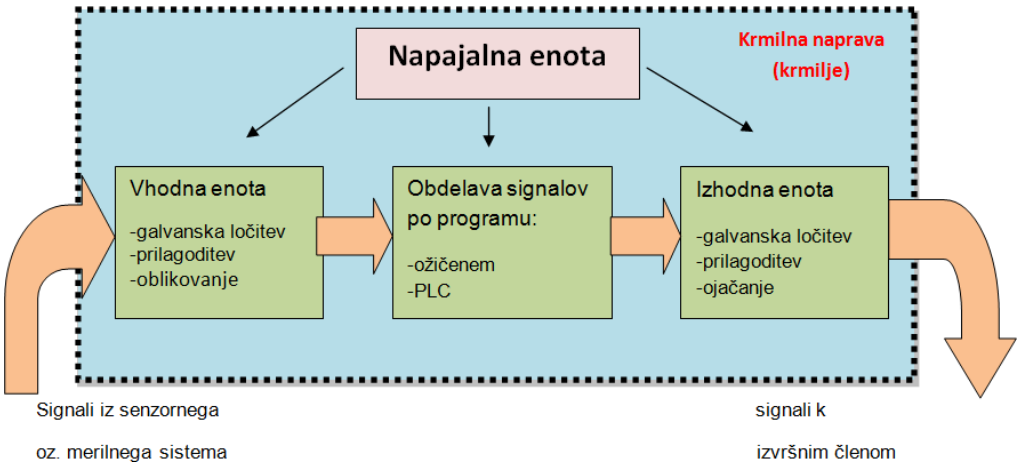


Figure 23: Structure of the control systemInput modules

In order not to transmit the disturbances, an octopter or octocopter is used, which removes the disturbances.

In addition to the tasks mentioned above (level adjustment, galvanic separation, short-circuit protection), there are additional options:

- the possibility of setting the inlet filter time (elimination of quick switches at the input due to interference),
- different number of inputs (expandable),
- indication of the logical unit.

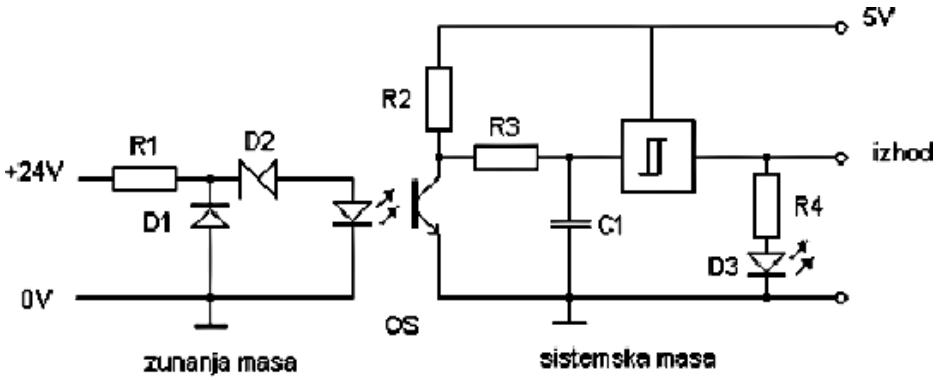


Figure 24: Electrical diagram of DC input digital module
Source: http://support.automation.siemens.com

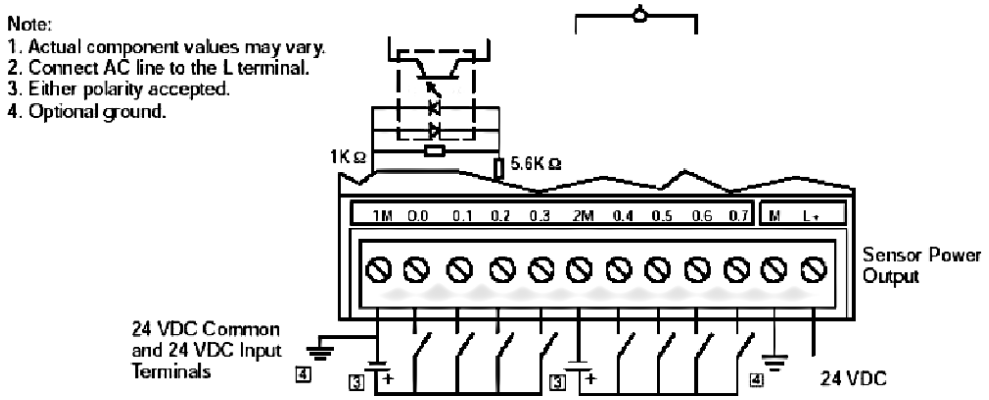


Figure 25: Connecting the inputs to the input digital module at S7-300
Source: http://support.automation.siemens.com

OUTPUT MODULES

- They are designed to transmit and form a signal transmitted by the central controller unit. Signals from the central processing unit are usually TTL-level (5 V power supply).
- Higher signal levels can be achieved with a transistor amplifier, auxiliary relays or tapes.
- Current loads are in the range of a few mA to a few A.
- Galvanic separation of system work from external, process.
- Indication of the unit of separation.

The consumer can only be connected to DC voltage (lower current loads).

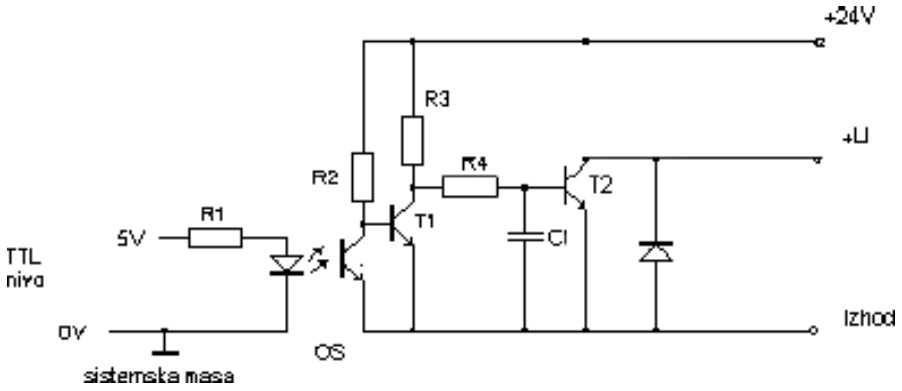


Figure 26: Digital output module with transistor output
Source: http://support.automation.siemens.com

The consumer can be connected to DC or AC voltage.

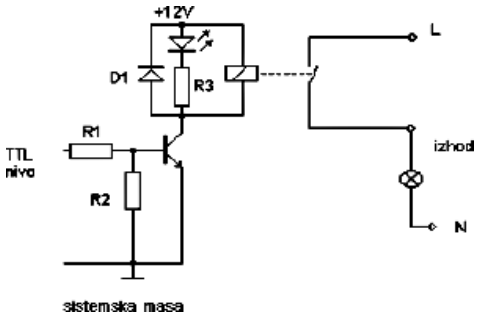


Figure 27: Digital output module with relay output
Source: http://support.automation.siemens.com

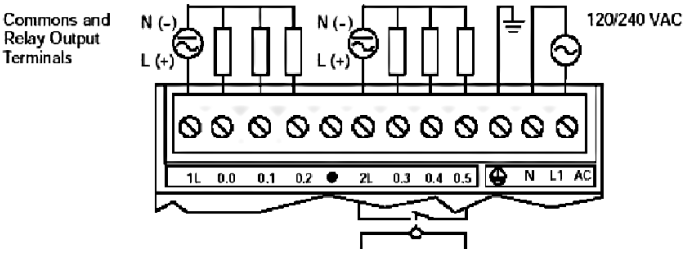


Figure 28: Connecting the output units to the digital relay output module
Source: http://support.automation.siemens.com

SIGNAL TRANSMITTERS

The task of signal transmitters (sensors, detectors, measuring elements) is to measure or detect the quantity in the control system, convert the signal into the appropriate form (amplification, temp. compensation, offset, etc.) and transmit the information to the control device.
There are different types of transmitters:

- **mechanical** ones convert the force acting on the encoder into an electrical signal (limit switches, pushbuttons, but they can also convert the force into current or voltage),
- **Resistives** use a property to change the resistance of an element. Bridge circuits are used (resistive sheets, thermistors, linear and rotary potentiometers...),

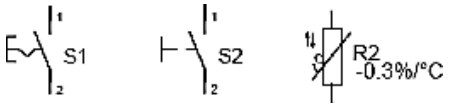


Figure 29: Mechanical and Resistive Position Encoder

- **inductive –** the measured physical quantity affects the change in inductance, the output can be binary with working or rest contact,
- **capacitive** works on the change in capacitance, can measure displacements, levels, use as zoom limit switches,
- **optoelectrons** act on the change in light, which they convert into an electrical voltage (e.g. photocell), or they act on the change in ohmic resistance (e.g. photoresistor, photodiode, probes use UV or IR light),
- **CMOS-sensors** (sensor and electronics in one chip – sensor effect in Si, lower price, higher quality, less interference).

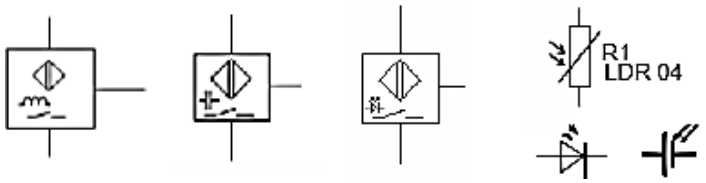


Figure 30: Inductive, capacitive, optoelectronic and CMOS signal encoders

EXECUTIVE MEMBERS – ACTUATORS

The task of the executive members is to execute orders, report and signal about the status of the process.

- Electric motor drives

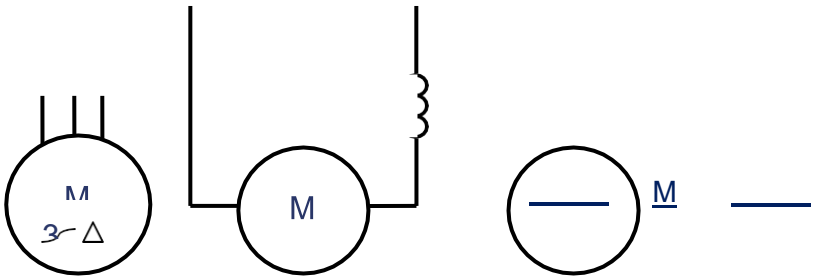


Figure 31: Symbols for some electric motors

- Pneumatic cylinders

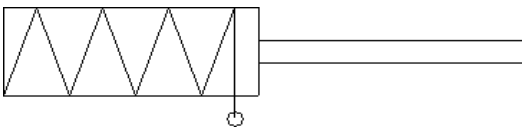


Figure 32: Pneumatic cylinder

- Lights, heaters, fans

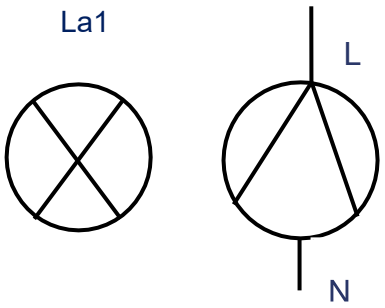


Figure 33: Lamps, heaters, fans

- Valves

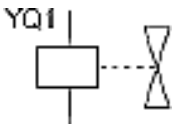


Figure 34: Valves

- Announcements

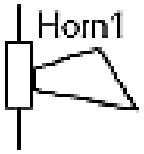


Figure 35: Reporting

- Displays

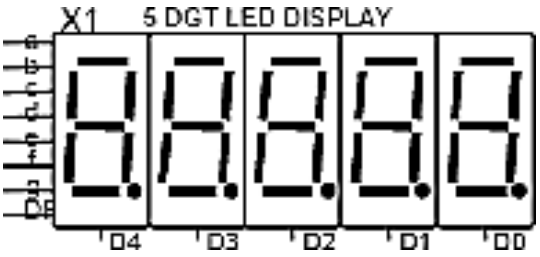


Figure 36: Display

- Traffic lights

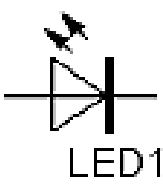


Figure 37: Traffic lights

CONTROLLING THE DEVICE

a) **Open-loop control** (a combination of input variables with the help of a control device activates control signals and transmits them to executive members through the output unit – there is no feedback from the process).

The operation or management of the process takes place in two phases:

- Decision-making (processing of data, signals).
- Action (transmission and execution of commands, usually accompanied by appropriate signalling).

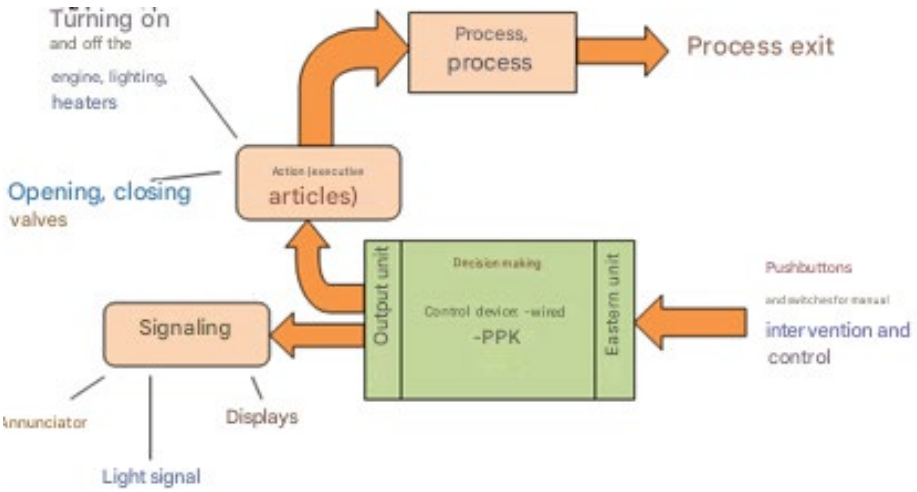


Figure 38: Open-loop control

b) **Closed-loop control** (the most common method, especially when controlling industrial processes; signals from the process come to the input of the control device).

The operation or management of the process includes an additional phase:

- **Observation** (collection of information and data)

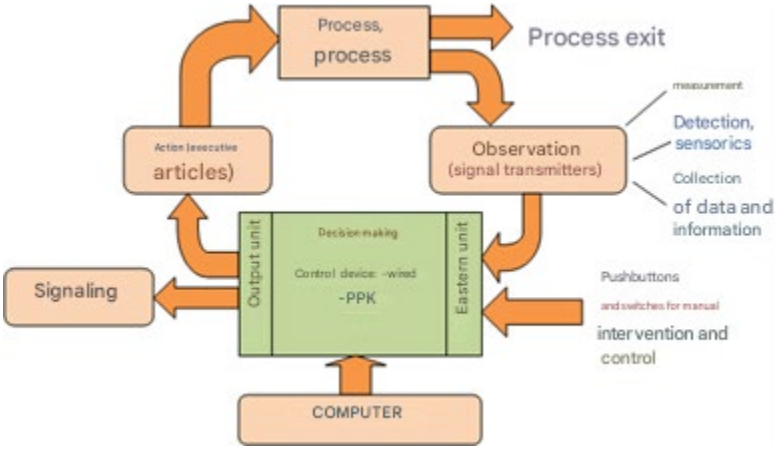


Figure 39: Closed-loop control

4.2. SEQUENTIAL STEERING

4.2.1.FEATURES OF SEQUENTIAL CONTROLS

- The output depends not only on the current state of the inputs, but also on the previous states of the system.
- In addition to the basic logic functions (combination circuit), sequential controls also consist of memory elements, time, count and other additional functions.
- The same combination of input states can be mapped to different output combinations.
- You can compare them with asynchronous sequential circuits and describe and design them using a state diagram or state table or step chains.

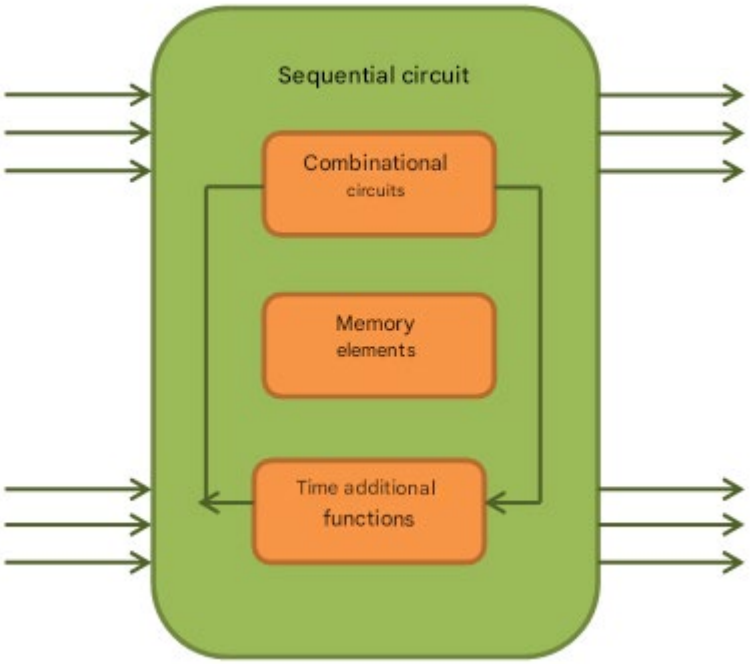


Figure 40: Sequential circuit

4.2.2.ADVANTAGES OF SEQUENTIAL CONTROLS

a) It is a simpler - smaller and simpler controller program.
b) Less sensitivity to noise signals and interference.
c) A system of stable states; the transition to a new state is carried out only under certain conditions; The system does not respond to other changes.
d) Easier error detection.

Sequential controls are divided into:

- **Free-running controls** – any combination can appear at the entrance in any sequence.
- **Step-by-step controls** – combinations of inputs always appear in a certain sequence.

4.2.3.BUILDING BLOCKS OF SEQUENTIAL CONTROLS

Memory (memory) functions:

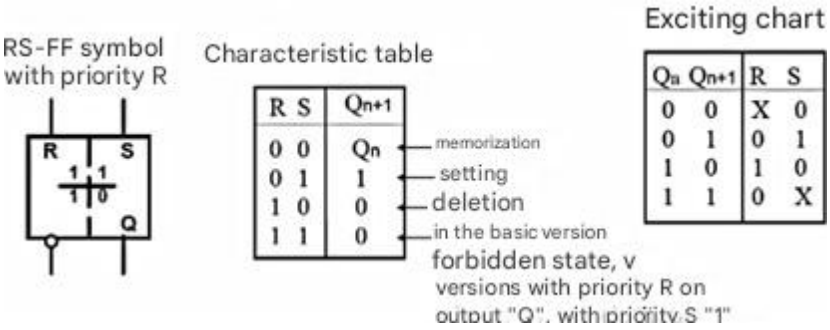


Figure 41: RS Functions
Source: Simatic Step7 Help

Memory (memory) functions in a scale diagram:

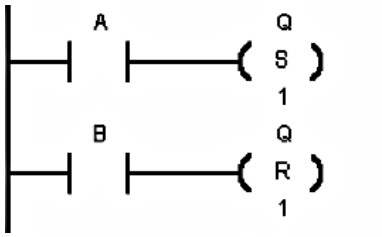


Figure 42: RS-function (scale diagram)

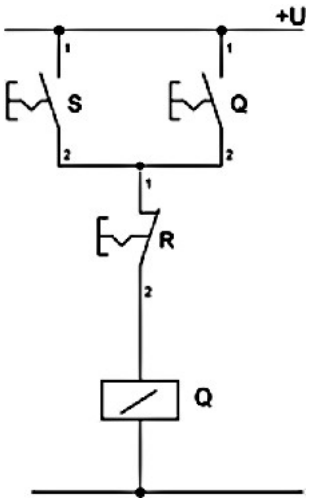


Figure 43: RS- function
Source: Simatic Step7 Help

In digital technology, there are three more types of FF: JK-, D- and T-FF (counters and registers). As solo FFs, they don't often perform in the controls. If we need them, we can easily implement (program) them based on the basic RS-FF.

TEMPORAL FUNCTIONS

Time functions are used to determine the duration of logic signals – you can shorten, lengthen or shift them in time.

There are three basic types of time functions:

- shortening of long pulses,
- prolongation of short pulses,
- time shifts of impulses.

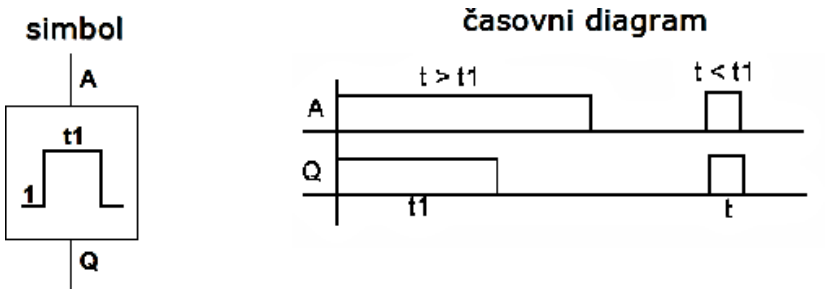


Figure 44: Pulse shortening
Source: Simatic Step7 Help

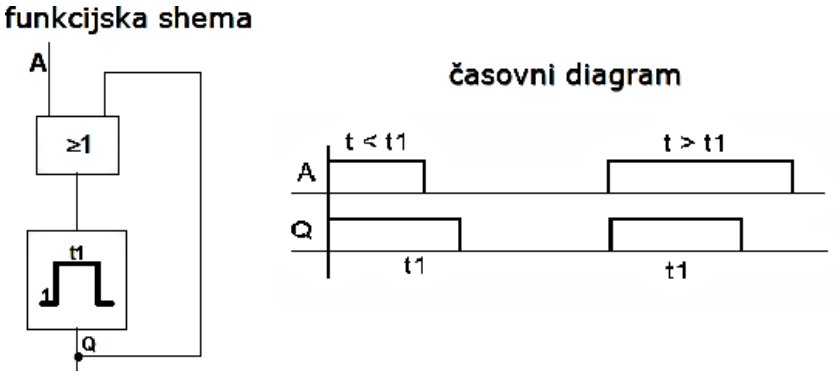


Figure 45: Pulse elongation
Source: Simatic Step7 Help

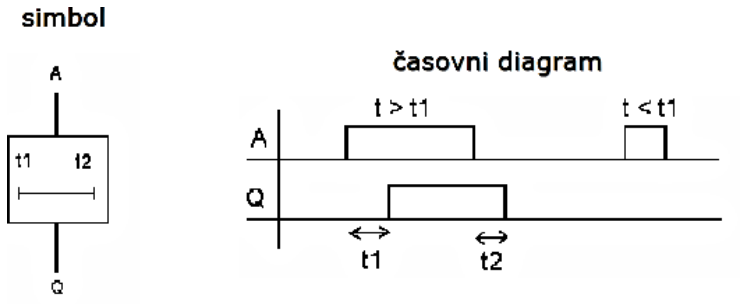


Figure 46: Moving pulses
Source: Simatic Step7 Help

Special Time Features:

In control technology, two other modes of special timing functions are often used. These are the delays that are realized by means of time functions and logical gates:

- Switch-on delay.
- Shutdown delay.

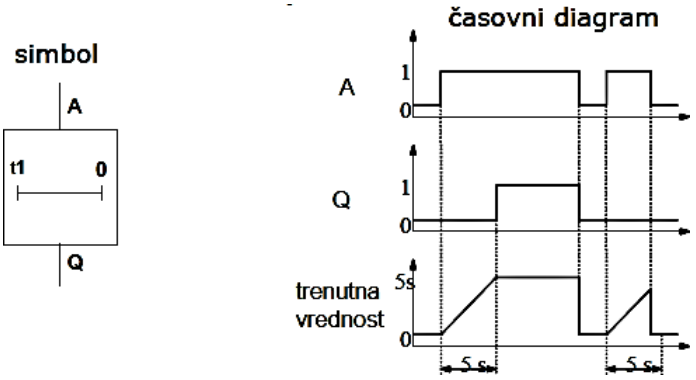


Figure 47: Explanation of power-up delay
Source: Simatic Step7 Help

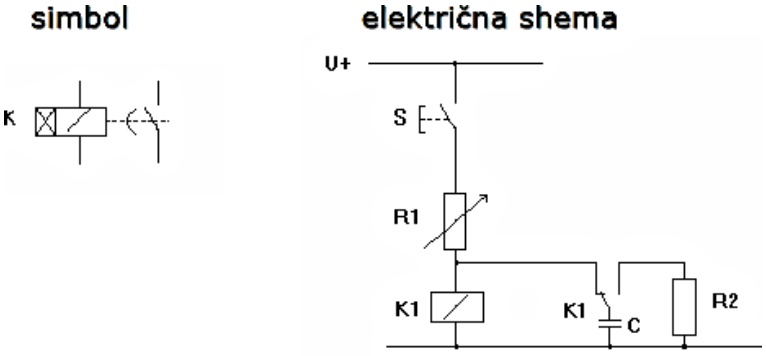


Figure 48: Power-up delay (relay)
Source: Simatic Step7 Help

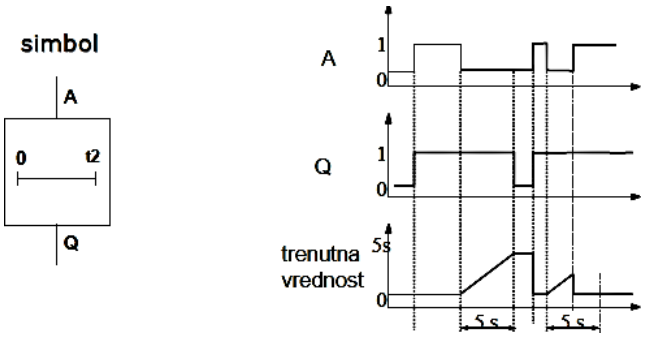


Figure 49: Explanation of shutdown delay
Source: Simatic Step7 Help

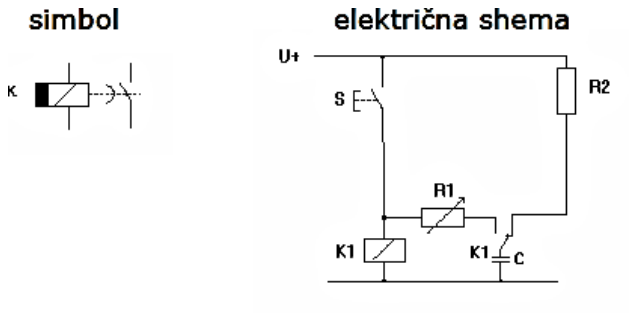


Figure 50: Implementation of the shutdown delay (relay)
Source: Simatic Step7 Help

Counters (event counting)

CU: Count Up (Dynamic) CD: Count Down (Dynamic) R: Reset
PV: set value
LD: load PV value (dynamic) Q: output (bool)
CV: Counter value (integer value)

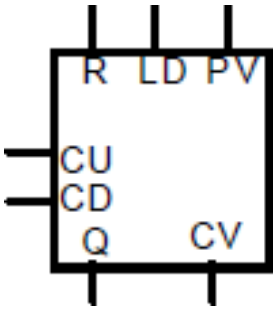


Figure 51: Counter
Source: Simatic Step7 Help

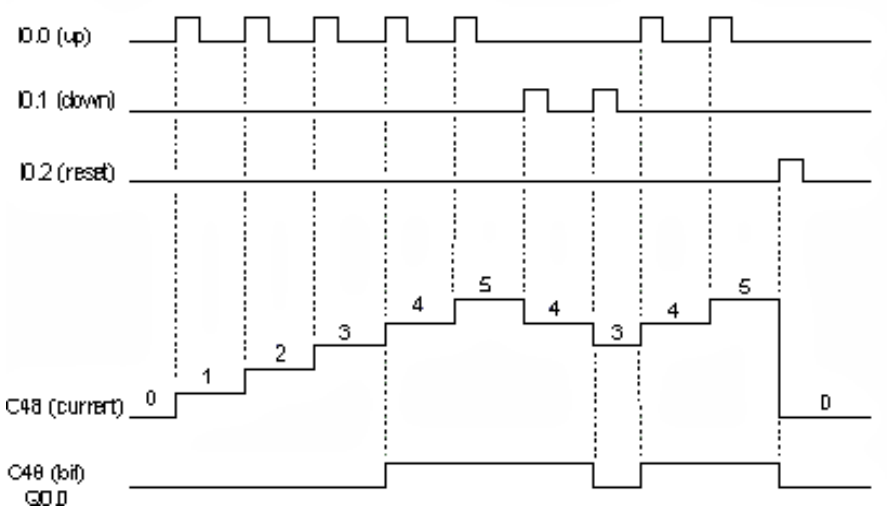


Figure 52: Time diagram of UP/DOWN meter operation
Source: Simatic Step7 Help

4.3. FEATURES OF THE SIEMENS CONTROLLER

Freely programmable controllers have been an essential part of system automation and control since their inception. Their use has become so widespread that it has almost completely replaced the use of so-called hard-wired logic and analog controllers. The reason for their implementation should be sought primarily in the ability to program quickly without changing the wiring.

Due to the mass production and use of freely programmable controllers in industry, these have become:

- It's relatively cheap,
- very reliable in operation,
- Easy to use (programming, installation and servicing).

The use of freely programmable controllers is widespread today. The main areas of application are:

- capture and processing of analogue and digital data,
- regulation of various systems,
- performing computer operations,
- as an interface for performing remote commands,
- for communication.

Freely programmable controllers are a computer without a display and keyboard, but it has all the other components, such as the processor, memory, input/output interfaces, and communication interfaces for connecting to other systems that run on the basis of a user program. It is adapted to operate in an industrial environment. With the development of controllers, various software tools have been developed, with the help of which we write user programs for different types of controllers. With the help of user programs, we assign the controller the tasks that it needs to perform. We write programs most often with computers. In doing so, care must be taken to use the correct communication interface for the operation of the controller.

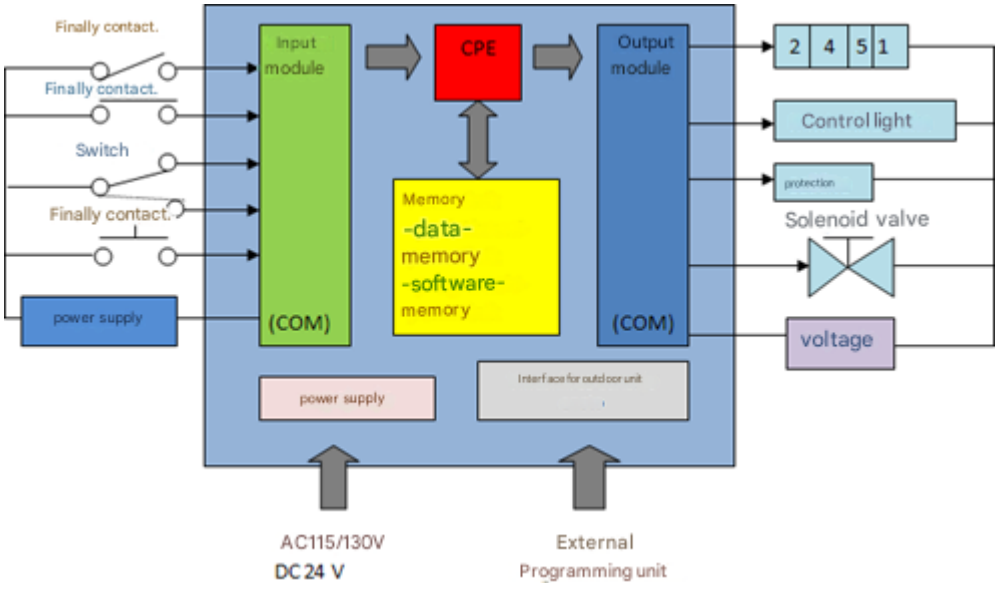


Figure 53: Block Controller Structure

4.4. HARDWARE SIMATIC SIEMENS

SIMATIC control systems are divided into three subgroups, namely:

- SIMATIC S7 programmable controllers,
- SIMATIC M7 systems and
- Complete control system SIMATIC C7.

SIMATIC S7 programmable controllers are also divided according to their size or capacity:

- lower class of controllers, S7-200 series;
- mid-range controllers, S7-300 series;
- upper class of controllers, S7-400 series.

SIMATIC M7 systems enable the integration of mathematical operations and the processing of databases into the STEP7 control program.

SIMATIC C7 systems include the S7-300 family programmable controller and control panel (OP) in one device.

Communication between the operator and the device takes place through the devices for operation and observation. SIMATIC offers a whole range of these devices, which mainly support SIMATIC systems as well as some other systems from other manufacturers.

All of these systems can be programmed with special PG programmed devices, which are very expensive, so it is better to use personal computers or laptops on which the STEP 7 software package is loaded.

In order for systems to work properly, they must also communicate with each other. We know different systems for communication, which differ from each other in speed, transmission capacity and, of course, price.

The systems most commonly found in the industry are:

- PROFIBUS,
- PROFINET,
- SAFETYBUS and
- Industrial thernet

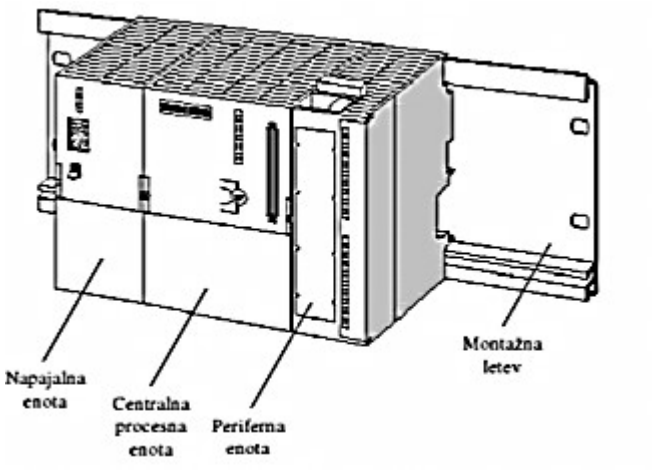


Figure 54: Display of individual modules of the Siemens control system Source: http://support.automation.siemens.com

4.5. COMPONENTS OF A FREELY PROGRAMMABLE SYSTEM POWER SUPPLY UNIT (PS 307 5A)

The power supply unit allows us to change the voltage from 230V to 24V. This allows us to transformer that is in it. A phase conductor, a neutral conductor and a ground conductor are brought to the unit, a voltage of 230V flows through them. From the unit, we get a voltage of 24V, to which all other components can be connected.

Figure 55: Power supply unit

CENTRAL PROCESSING UNIT (CPU 313C)

In our case, we used a 313c CPU, which takes care of processing, processing, and upgrading data. This component could be described as the most important and demanding in the entire modular controller. Of course, there are also inputs and outputs (DI 16/DO 16* DC 12V) on the central processing unit, where the signals of sensors, motor, relay and robot are located.

Figure 56: Central Processing Unit

NETWORK LAYOUT COMPONENTS (CP 343-1 ADVANCED, SCALANCE X208)

To set up a network, you first need a so-called network module, which contains the Simatic Net protocol, which runs on the CP 343-1 Advanced module. This module takes care of setting up the network, but the problem arises, which is that this module has only one RJ 45 interface, so you need a splitter, the so-called Scalance X208.

Figure 57: Communication processor X208 CP 343-1 Advanced

Figure 58: Switch Scalance

4.6. SOFTWARE

For software support for users of Simatic Group products, Siemens has developed the STEP 7 software package. The software package is compatible with Microsoft Windows operating systems and complies with EN 61131-3 [7].

We divide it into four groups of software products:

- Basic programming tools (*Standard Tools*),
- *Engineering Tools*,
- software tools for real-time work (*Runtime Software*) and
- software tools for human-machine interface (*Human Machine Interface*).

STRUCTURE OF THE STEP 7 PROGRAM:

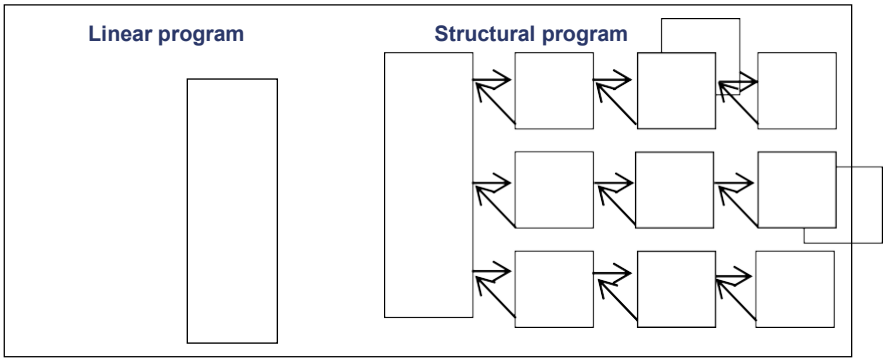


Figure 59: Structure of the STEP 7 program

The entire user program is concluded in one block (OB1). The processor executes the program linearly by sentence as typed. Such an organization of the user program is found in the S7-200 automation system. In addition, the technique of subroutines can also be used in it.

STRUCTURAL PROGRAM

The entire user program is divided into blocks. A block is an automatic part of the program, which differs from the rest in its function, task and priority.

WE DISTINGUISH SEVEN TYPES OF DIFFERENT BLOCKS:

Organizational blocks (OB) regulate cyclical, time, and alarm processing of programs.
Properties:

- the user's access to the operating system,
- priority by tiers, and
- additional start-up information about the local fund.

Function blocks (FBs) contain technology functions and have a reserved portion of memory to store local variables until the next block call.
Properties:

- uses static data,

- the assigned data block, and
- Suitable for programming complex, frequently used functions.

Functions (FC) also contain technological functions, but they are without a memory function for their local variables.
Properties:

- returns its result (without static data) to the program, and
- Mostly out of memory (they don't have an assigned data block).

Data blocks (DBs) are used to store various user data that are available to all parts of the application.
Properties:

- A structured store of local data, and
- Structural storage of global data (data is available to the entire application).

System blocks (SFB, SFC, SDB) are blocks that contain system functions. These blocks are an integral part of the operating system and do not use user memory. The blocks can be used by the user in their application program. SDBs contain data for configuring individual modules and communications.
Properties:

- System function blocks (SFBs) are integrated into the CPU operating system,
- system functions (SFC) are integrated into the CPU operating system and can only be used by the user, but cannot be modified; similar to the FC block, the SFC block is devoid of static variables and
- System data blocks (SDBs) contain data for system configuration.

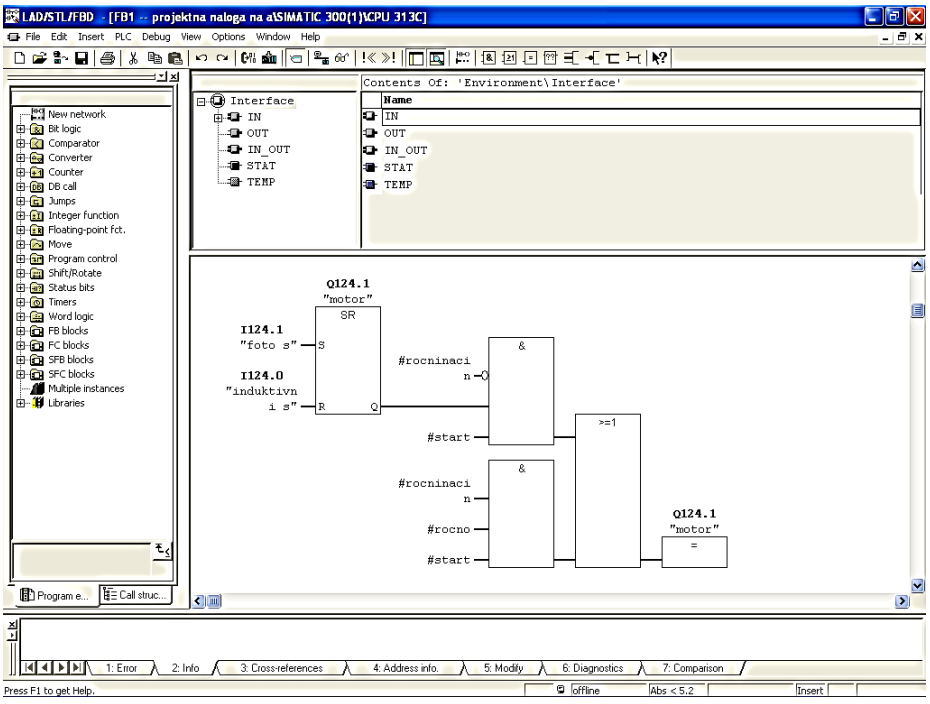


Figure 60: Program in FB1

4.7. STEP 7 – SOFTWARE AND CONFIGURATION EQUIPMENT FOR SIMATIC

STEP 7 is the software and configuration software for the SIMATIC S7. It consists of several individual applications, each of which performs a specific function. Thus, we have functions that accompany us from the beginning of the creation of the project to its completion.

Functions can be grouped into the following groups:

- Hardware configuration functions
- Functions for configuring networks,
- Programming functions,
- Testing and servicing functions
- Documentation and archiving functions.

The main graphical interface in STEP 7 is the SIMATIC Manager. It collects all the necessary data from various applications to design the project as a whole. Within the project itself, data is divided by function and is presented as objects. When we want to work with an individual project, the appropriate tool for working with that object is also activated.

There are two ways to program:

- direct entry of the program into the central unit (on-line programming) and
- Programming Memory Module into Programmed Device without Links with an automated device.

The memory module is subsequently inserted into the central unit (off-line programming). Programming of the user program takes place due to the structure of the controller on a separate programmed device. The transfer of the program from the programmed device to the controller can be done by transferring the program to the EPROM memory and by later insertion into the controller, or it is programmed directly into the controller's memory. This requires a multipoint connection (MPI) between the programmed device and the controller.

The STEP 7 programming language allows us to program with three programming methods that are characteristic of memory-programmable controls:

- Function Block Diagram *(FBD*)
- contact mode – LAD (*Ladder Logic*),
- set of commands – STL (*Statement List*).

FUNCTIONAL PLAN – FBD:

FBD is a programming method in which commands are entered using the graphical method. The function plan uses logic blocks known from Boolean algebra. You can use it for complex functions that can be represented directly by logic blocks. Its great advantage is the direct input of the program, since programming with logic blocks is easy and quick to understand. However, this programming method is not suitable for programming computational operations and more

complex processing of digital values.

CONTACT PLAN – LAD:

LAD is also a graphical programming method where commands are represented by symbols derived from the contact technique. The main elements of a contact plan are a work switch and a peace switch, so it's easy to learn. The disadvantage of this software method is opacity, so it is not suitable for complex tasks.

COMMAND SET – STL:

STL is a programming method where commands can be entered in writing. This programming language is very similar to an assembly language and consists of a series of mnemonic commands – these are execution commands. Each command corresponds to the processor's step through the program. The advantage of the command set lies in the fast execution of the program, as both the processing time and the memory location are optimized. In the editor itself, the work is even easier, as you can use symbolic description, and there are also functions for searching and checking the correct entry in real time. The editor allows you to save or store reused software parts in a standard software booklet, which enables later reuse. Each command in STL consists of an operation and an operand. The operand is further divided into an operand designation and a parameter. In the command itself, it is also possible to enter a tag that is used for jumps, and in the editor we are also allowed to enter comments for easier understanding of the program.

CONTROLLERS FOR SIMATIC S7:

- The standard PID controller enables the use of continuous regulators, regulators for pulse and stair responses in the user program,
- A modular PID controller is used if the standard PID is not sufficient to solve the automation task, and
- *Fuzzy* controllers are used to create logic systems. These systems are used when a process is difficult or impossible to describe mathematically, when processes behave unpredictably, when dealing with nonlinear events, but experience with processes is available.

SOFTWARE TOOLS FOR HUMAN-MACHINE CONNECTION:

HMI is a special *software* for operator control and *monitoring*:

- SIMATIC WinCC SCADA is an open process visualization,
- SIMATIC ProoTool and SIMATIC ProoTool/Lite are modern tools for configuring SIMATIC operator panels and SIMATIC C7 compact units (the Lite version is only for text panels) and
- Pro Agent enables fast, planned process diagnostics in plans and machines, in search of information about the location and causes of errors.

4.8. CONTROLLER PROGRAMMING

First, you need to write a symbol table for input/output signals, which you later supplement with markers, counters and timers. Also, the symbol table is written according to the standard, all locations in the controller are predetermined. The symbol table is used for easier programming, because in the case of writing signals only by locations in the controller, such a program would be very opaque and would probably only be difficult to complete it.

The figure below shows part of the symbol table for the input signals. One row of the table represents one signal, namely the name (*symbol*), address or location in the controller (*address*), data type (*data type*) and comment (comment) of the signal. The flags that we see at the beginning of the line say that these signals are used in the security blocks.

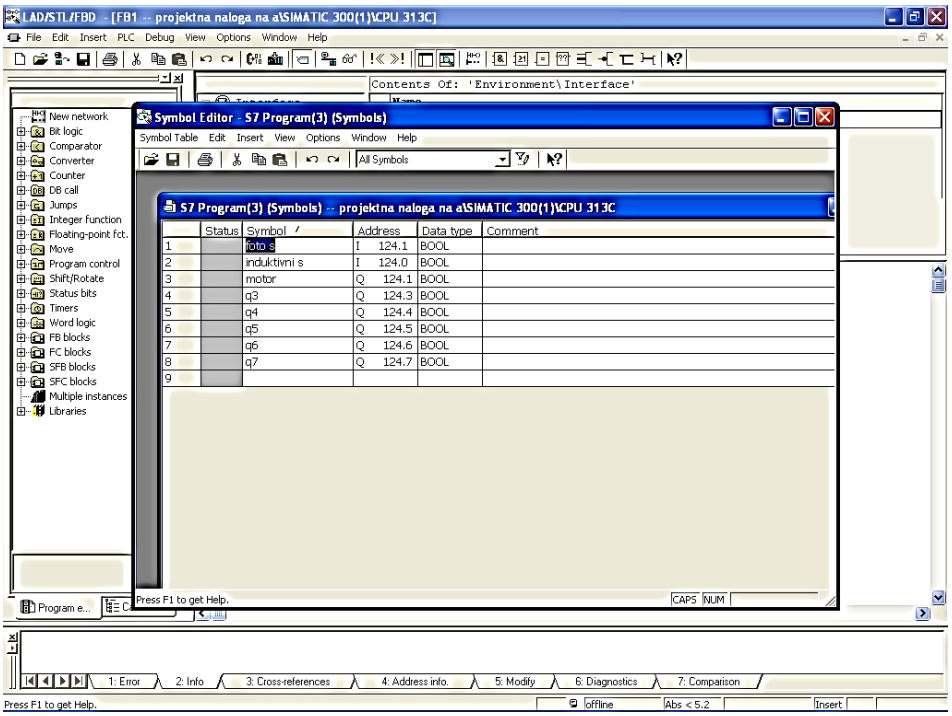


Figure 61: Symbol table

4.9. CONNECTING THE CONTROLLER TO AN ETHERNET NETWORK

Ethernet is the most popular protocol for local area networks. This industry standard has been adopted by many network hardware manufacturers. Today, many of the problems that could arise from the incompatibility of hardware from different manufacturers are practically non-existent with Ethernet. Today, Ethernet networks operate at speeds of 10, 100 and 1,000 Mbps (1 Gbps), allowing applications from home and small business networks to high-performance network backbones.

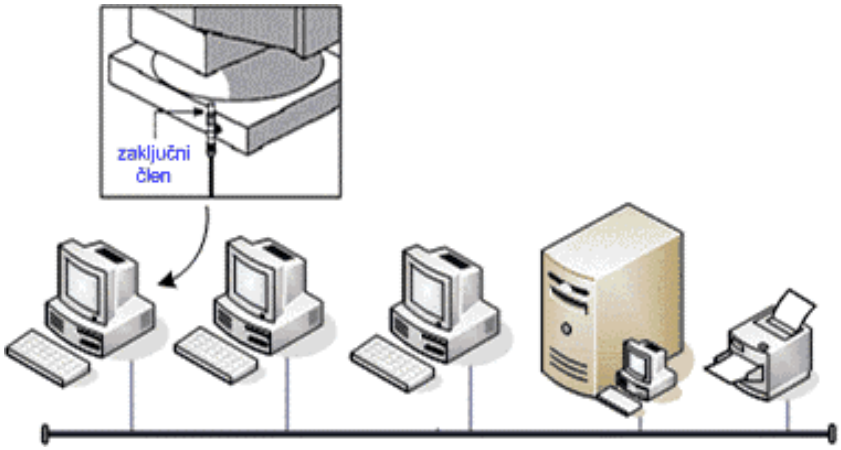


Figure 62: Network connection
Source: http://www.s-sers.mb.edus.si/gradiva/w3/omrezja/01_omrezja/01_omrezja.html

The Physical Layer Standards of the Ethernet standard describe the types of cables from which you can build a network, specify the topology, the maximum length of the cable segment, and the number of restorers that can be used. It is important to follow the standards, as the CSMA/CD mechanism is sensitive to crosstalk and attenuation.

Table 3: Standards

Label	Cable	Topology	Velocity	Segment
10Base5	RG8 Coaxial	Guide	10 Mbps	500 meters
10Base2	RG58 Coaxial	Guide	10 Mbps	185 meters
10BaseT	Category 3 UTP	Star	10 Mbps	100 meters
FOIRL	Multi-Gender Optical Fiber 62.5/125	Star	10 Mbps	1,000 meters
10BaseFL	Multi-Gender Optical Fiber 62.5/125	Star	10 Mbps	2,000 meters
10BaseFB	Multi-Gender Optical Fiber 62.5/125	Star	10 Mbps	2,000 meters
10BaseFP	Multi-Gender Optical Fiber 62.5/125	Star	10 Mbps	500 meters

100BaseTX	Category 5 UTP	Star	100 Mbps	100 meters
100BaseT4	Category 3 UTP	Star	100 Mbps	100 meters

4.10. UTP ETHERNET NETWORK

All other physical Ethernet topologies use a star topology, in which each segment connects the computer to the hub. The most popular cable in today's Ethernet is unshielded pair (UTP). It allows connections at speeds of 10 Mbps, 100 Mbps, and 1000 Mbps. 10BaseT uses only two pairs, one for receiving and one for transmitting.

Fast Ethernet (IEEE 802.3u) describes two specifications that also allow a maximum segment length of 100 meters. 100BaseTX requires Category 5 cables that have better signal transmission. 100BaseT4, on the other hand, allows you to upgrade your network with Category 3 cables. 100BaseT4 uses all four pairs.

Most standards for Gigabit Ethernet prescribe fiber optic cables, but there is also the option of using UTP according to the IEEE 802.3ab standard. The 1000BaseT standard describes the possibility of upgrading an existing network with Category 5 cables (even better if the cables are often designated as 5E according to the new, stricter criteria). 1000BaseT achieves higher speeds by using all four pairs and using PAM–5 pulse-amplitude modulation.

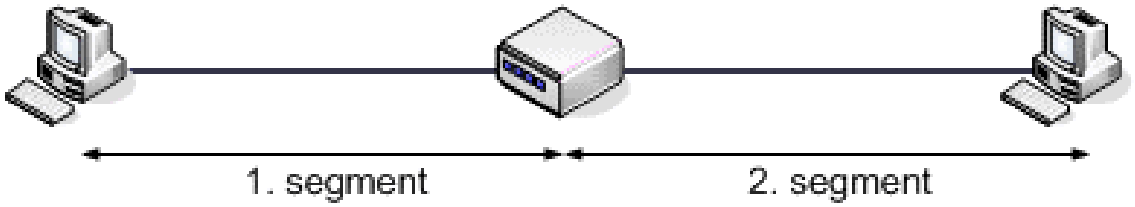


Figure 63: UTP Ethernet
Source: http://www.s-ers.mb.edus.si/gradiva/w3/omrezja/01_omrezja/01_omrezja.html

4.11. NETWORK PROFINET

Parallel to industrial Ethernet for 15 years, the most widely used and most widely used real-time protocol, Profibus, which is backed by Siemens, has existed. The Profibus network is designed for fast and reliable communication in automation processes. The maximum data transfer rate is 12 Mbit/s, which was revolutionary at the time of its introduction, but today it no longer covers the requirements. Problems can also arise with the number of stations connected to the network, which can be up to 127, and each repeater that increases the range of the network also represents one of these devices. The Profibus network is thus reaching the limit of its capacity due to these shortcomings

Siemens is working on a new protocol with the help of some partners. The high transmission speeds of Ethernet open up new possibilities for real-time communications. Ethernet in its classical embodiment does not provide deterministic behavior, and for some systems, the determinism of the transmission time is crucial. In industry, processes are divided by reaction time. For most devices, a reaction time of between 10 and 100 ms is allowed, which is satisfied by the use of classic industrial Ethernet with appropriate drivers. Profinet V2 offers optimised Software-based communication channel. Clock synchronization, on the other hand, presents even higher demands for real-time behavior.

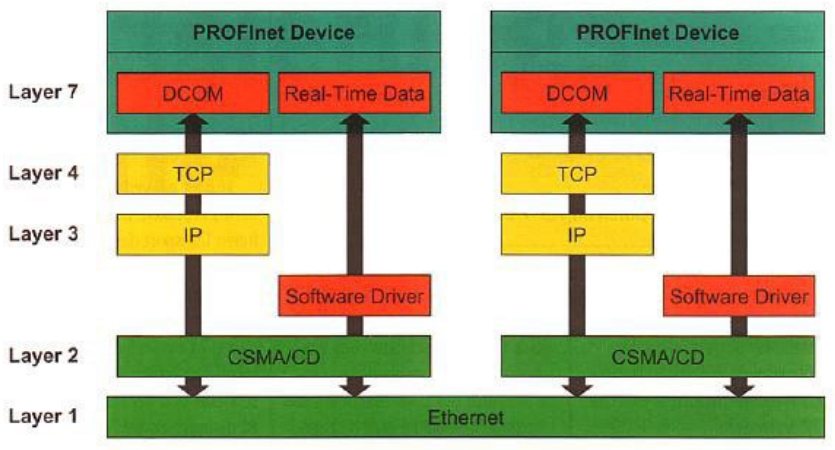


Figure 64: Profinet

Summary:

The controller is a programmable device that contains its own microprocessor, RAM, high-speed input-output units. It enables the regulation and control of the entire production or just one device, it can receive signals via sensors or directly from the control object. These signals are then processed in the controller. The processed signals travel back to the control object. Communication between a PC and a programmable logic controller usually takes place via a serial port or TCP/IP protocol. A programmable controller is basically a CPU (*Central Processing Unit*) that contains a program and is connected to input/output devices. The program is controlled by the PC (*Programmable Controller*) in this sense: when the input signal on the input device changes, the corresponding response is generated (according to the program written in the controller's memory). Input devices can be photoelectric sensors, switches, limiters, and limit switches. The output signals can be 24V voltage, current signal, signal that turns on the relay.

QUESTIONS:

1. The structure of the control system. What is a power supply unit?
2. Types of signal transmitters.
3. Features and benefits of sequential controls – what do we divide them into?
4. What do we use temporal functions and their types for?
5. List the main areas of application of programmable controllers.
6. Division of SIMATIC control systems into subgroups and according to size.
7. Describe and sketch the central processing unit.
8. What do you know about Siemens software?
9. Describe the contact plan (LAD).
10. What is the difference between a compact and a modular PLC, list the advantages and disadvantages of each type.

5. INDEX OF IMAGES

6. INDEX OF TABLES

7. ANNEXES

LCAMP

Learner Centric Advanced Manufacturing Platform

Co-funded by
the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.