Smart Motor Devices

User manual

STEPMOTOR CONTROLLER

SMSD-1.5Modbus

2020

Precautions and remarks in the text:

Attention
Different types of danger, which may result in damage to property or injuries.

Information
Recommendations, advices or reference to another documentation.

Highlights and formatting of text fragments:

● Label or mark of choice
1) Some actions should be done in a sequential order
— Common enumerations

Contents

1. The intent of the controller

The controller SMSD-1.5Modbus is designed to operate with stepper motors with the maximum current per phase up to 1.5 Amp. The device can be controlled by PLC (via RS-485 Modbus ASCII/RTU) or can operate in a standalone mode according to the preset executing program.

The controller provides fullstep operating or microstepping up to 1/256. The controller provides smooth motion with a low level of vibrations and a high accuracy of positioning.

The controller provides the next control modes:

- Program control mode – standalone operation according to the preset executing algorithm or a real-time control from a PC or PLC by commands given via Modbus protocol.
- Analog speed control – motor rotation speed is adjusted by the potentiometer at the front side of the controller.
- STEP/DIR control mode – control the motor position by pulse logic signals.

SMSD-1.5Modbus has 8 logic inputs and 10 outputs (2 of these outputs are high voltage). The state of I/O can be read or set from a user executing program or directly by Modbus commands. Internal executing program can be download from and upload to the controller through USB or RS-485.

We provide software for controller adjusting, assembling or editing of executing programs and for uploading of the executing program to the controller's memory.

The controller has overheating protection.

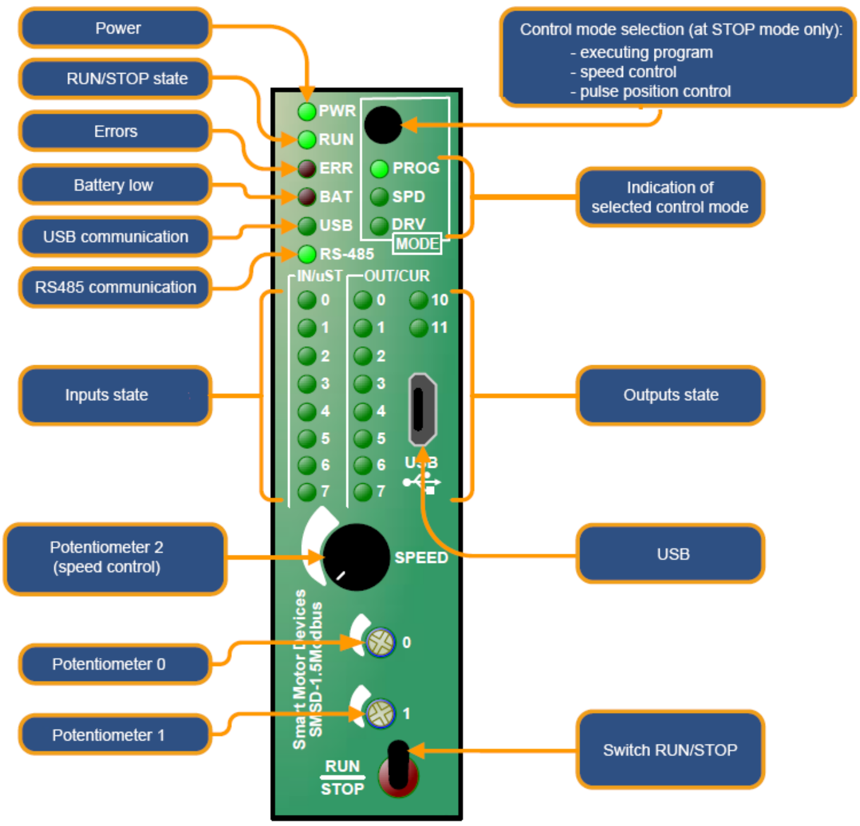


Fig. 1 – Purpose of control elements

Inputs

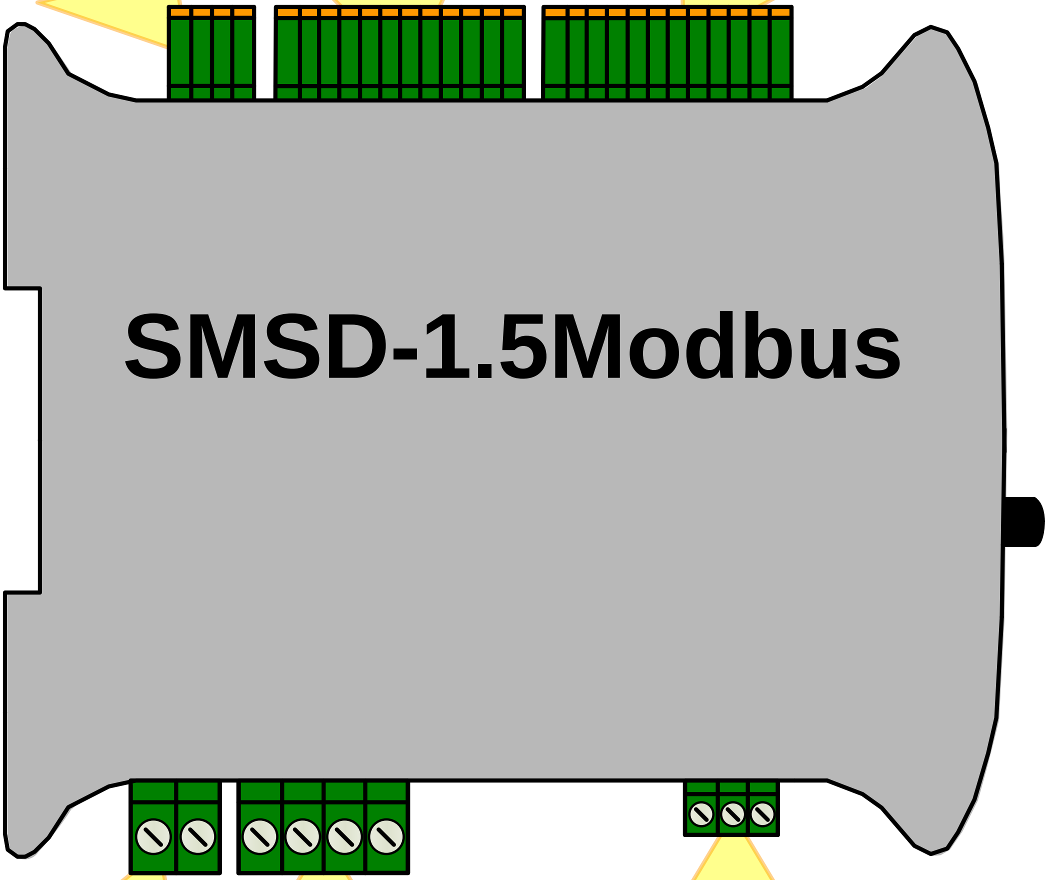
	Name	User program control mode	Speed control mode	Step/Dir control mode
1	COM0			STEP-
2	IN0	X0/Counter/ Encoder A	Speed UP Button/ Encoder A	STEP+
3	COM1			DIR-
4	IN1	X1/Counter/ Encoder B	Speed Down Button/Encoder B	DIR+
5	COM2			COM-
6	IN2	X2	SLOW STOP	ENABLE + / ENABLE+
7	IN3	X3	HARD STOP	INVERT_ENABLE
8	COM3			
9	IN4	X4	ENABLE	
10	IN5	X5	Step mode & Control Selector	
11	IN6	X6	Speed control Lock	
12	IN7	X7	DIR	

Outputs

	Name	User program control mode	Speed control mode	Step/Dir control mode
1	-5B			
2	+5B			
3	COM4			FAULT –
4	OUT0	Y0		FAULT +
5	OUT1	Y1		
6	OUT2	Y2		
7	OUT3	Y3		
8	COM5			
9	OUT4	Y4		
10	OUT5	Y5		
11	OUT6	Y6 / PWM		
12	OUT7	Y7 / PWM		

Outputs

	Name	User program control mode	Speed control mode	Step/Dir control mode
1	OUT10	Y10 RELAY		
2	OUT10'	Y10		
3	OUT11	Y11 RELAY		
4	OUT11'	Y11		



	Supply voltage 12...48 VDC
1	-U
2	+U

	Stepper motor
1	B+
2	B-
3	A-
4	A+

	RS-485
1	D+
2	D-
3	GND

Fig. 2 – Terminal assignment.

The Fig. 1 shows the front panel of the controller with control and indication elements. The **PWR** indicator indicates the presence of supply voltage. RUN indicates the current state of the controller (RUN or STOP). In **PROG** mode (execution of a user program) and **SPD** mode (speed control), the active state of RUN indicator indicates the execution of the program, the inactive state indicates stop state. In **DRV** mode (Step/Dir mode) the inactive **RUN** state indicates the ability to set driver parameters by the control elements of the controller. The active **RUN** state indicates entry to the operating mode, parameter changes are disabled. The ability to switch between the PROG / SPD / DRV modes is disabled in the **RUN** state. In the STOP state switching the control mode can be done by the **MODE** button. **ERR** indicates existing of errors. **BAT** indicates low battery inside the unit. **USB** and **RS485** indicate the process of a Modbus frame transmitting via USB and RS-485. In **PROG** and **SPD** mode the LEDs IN0 ... IN7 indicate the presence of a high level of a logic signal at the corresponding input. The active state of the OUT0 ... OUT11 indicators indicate the open state of the transistor output (see Fig. 3 – Fig. 6). In **DRV** mode the LEDs of the inputs IN0 ... IN7 indicate misrostepping setting, the outputs OUT0 ... OUT3 show the operating current setting, OUT4 ... OUT7 the holding current setting.

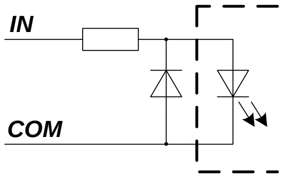


Fig. 3 – Inputs IN0 and IN1

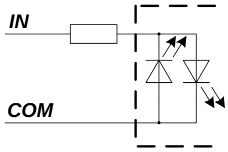


Fig. 4 – Inputs IN2 – IN7

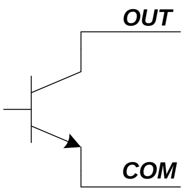


Fig. 5 – Outputs OUT0 – OUT7

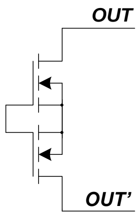


Fig. 6 – Outputs OUT10 and OUT11

The position of potentiometers **0, 1, 2 (SPEED)** are converted into 12-bit values, which are accessible for the further use in an executing program and can be read by Modbus command. In the **SPD** (speed control) mode potentiometer **2 (SPEED)** is used to set the rotation speed, potentiometer 0 - acceleration and deceleration rate, 1 – motor work current. In the **DRV** (Pulse/Dir) mode potentiometer 2 is used to set microstepping, 0 – motor work current, 1 - holding current.

The **RUN/STOP** switch is used to start and stop program execution in **PROG** and **SPD** control modes. It is used to switch between parameters setting and operation in **DRV** control mode.

Fig. 2 shows the controller terminals and their purpose depending on the control mode.

Tallinn Science Park Tehnopol,
Akadeemia tee 21/6, 12618, Tallinn, Estonia
Phone: + 372 6559914,
e-mail: mail@smd.ee
url: https://smd.ee

2. Technical specifications

№	Characteristic	Value	
		min	max
1	Supply voltage, VDC	12	48
2	Max. output current per phase, Amp	0,15	1,5
3	High level of logic inputs, VDC		2,4
4	Low level of logic inputs, VDC	0,7	
5	Voltage of logic transistor output, VDC		80
6	Max current of logic transistor output, mA		50
7	Voltage of logic relay output, VAC/VDC		350
8	Max current of logic relay output DC (AC/DC), mA		250 (~120)
9	Max current of additional output +5VDC, mA		200
10	Duration of high voltage level of STEP signal in Step/Dir control mode (DRV), ns	250(1)	
11	Duration of high voltage level of STEP signal in Step/Dir control mode (DRV), ns	250(1)	
12	Duration of high voltage level of signals at inputs IN0 and IN1, ns	70(1)	
13	Duration of low voltage level of signals at inputs IN0 and IN1, ns	70(1)	
14	Duration of high voltage level of signals at inputs IN2...IN7, mks	5(1)	
15	Duration of low voltage level of signals at inputs IN2...IN7, mks	5(1)	
16	PWM signal generation frequency, Hz	0,015	5000
17	Base instruction time, mks	185(2)	

(1) – on condition of 5VDC high voltage level

(2) - with no regard for returning to zero line, setting of outputs and reading of inputs

Additional information

№	Characteristic	Value
1	Possible baud rates for RS-485 data transmission	600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 57600, 115200, 128000
2	Possible microstepping settings	1/1, 1/2, 1/4, 1/8, 1/16, 1/32, 1/128, 1/256
3	Communication protocol	Modbus RTU, Modbus ASCII
4	Programing language	IL (Instruction List), LD* (Ladder Diagram)

* – *special software is needed to convert LD to IL before writing the program into the controller*

3. Operation sequence

The sequence of the controller operation is the next:

- reading of external devices (logic inputs, Modbus Coils);
- user program processing;
- setting of new states of output devices (logic outputs, Modbus Discrete Inputs, executing of motion).

The user program consists of a sequence of control instructions (commands) that determine the final functionality. The controller executes the commands sequentially one by one. The total program pass is continuously repeated. The time required for one program pass is called the cycle time, and the program passes are called cyclic scanning.

The controllers are able to operate in real-time mode and and can be used both for building local automation nodes and distributed I / O systems with the organization of data exchange via RS-485 interface with Modbus protocol.

We offer special software for assembling and debugging of user programs, which does not require significant computer resources and is a simple tool for users. Two programming languages are used: LD (ladder contact logic or ladder diagrams) and IL (list of instructions).

3.1. The principle of operation of relay-contact circuits and ladder diagrams in the controller

The language of the ladder diagrams is a derivate of the relay-contact circuit diagram in a simplified representation. The relay-contact circuits in the controller have a set of basic components, such as: normally-open contact, normally-closed contact, coil (output), timer, counter, etc., as well as applied instructions: mathematical functions, motor control commands, data processing and a large number of special functions and commands. We can assume that the controller is tens or hundreds of separate relays, counters, timers and memory. All these counters, timers, etc. physically do not exist, but are modeled by the processor and are designed to exchange data between built-in functions, counters, timers, etc.

The relay-contact logic language in the controller is very similar to the basic relay-contact electrical circuits if we compare it's used graphic symbols. There can be two types of logic in relay-contact circuits: combined, i.e. circuits consisting of fragments which are independent each of other, and sequential logic, where all the steps of the program are interconnected and the circuit cannot be parallelized.

Combined logic

The first segment of the circuit consists of one normally open contact X0 and a coil Y0, which determines the state of the output Y0. When the state of contact X0 is open (logical "0"), the state of output Y0 is also open (logical "0"). When the contact X0 is closed, the output Y0 also changes its state to closed (logical "1").

The second segment of the circuit consists of one normally-closed contact X1 and coil Y1, which determines the state of output Y1. In the normal state of contact X1, output Y1 will be closed (logical "1"). When the state of contact X1 changes to open, the output Y1 also changes its state to open.

At the third segment of the circuit, the state of output Y2 depends on a combination of the states of the three input contacts X2, X3, and X4. Output Y2 is closed when X2 is turned off and X4 is turned on or when X3 and X4 are turned on.

The general scheme is a combination of three segments, which operate independently each of other.

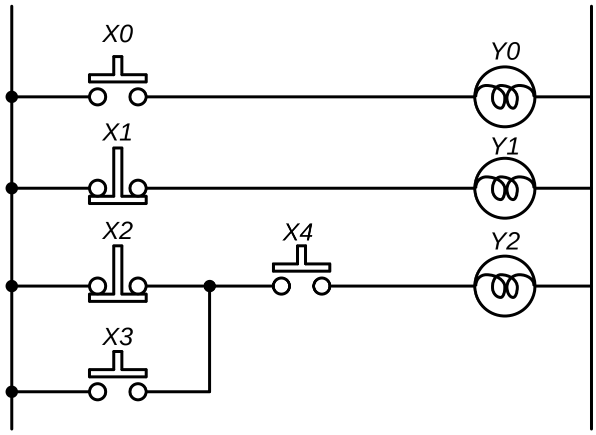


Fig. 7 – relay-contact circuit

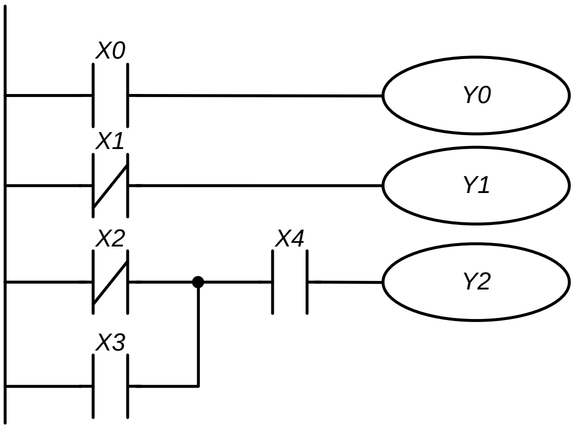


Fig. 8 – ladder diagram in the controller

Sequential logic

In the sequential logic circuits the result of execution at a previous step is an entry condition for the next step. In other words an output at the previous steps is an input at the following step.

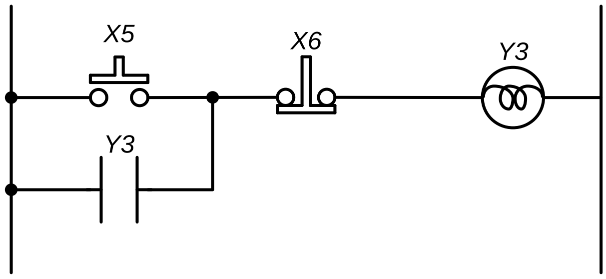


Fig. 9 – relay-contact circuit

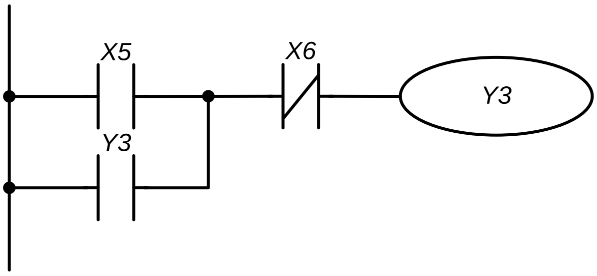


Fig. 10 – ladder diagram in the controller

When X5 contact is closed, the output Y3 changes its state to close. However when X5 is open again, Y3 keeps it's close state till the moment when X6 is open. In this circuit the output Y3 is selflocking.

3.2. Differences between logic of real relay-contact circuits and ladder diagrams in the controller

All specified control processes are performed simultaneously (in parallel) in conventional relay-contact electrical circuits. Each change in the state of the input signals immediately affects the state of the output signals.

A change of the state of the input signals that occurred during the current passage of the program in the controller is recognized only at the next program cycle. This behavior is smoothed out due to the short cycle time.

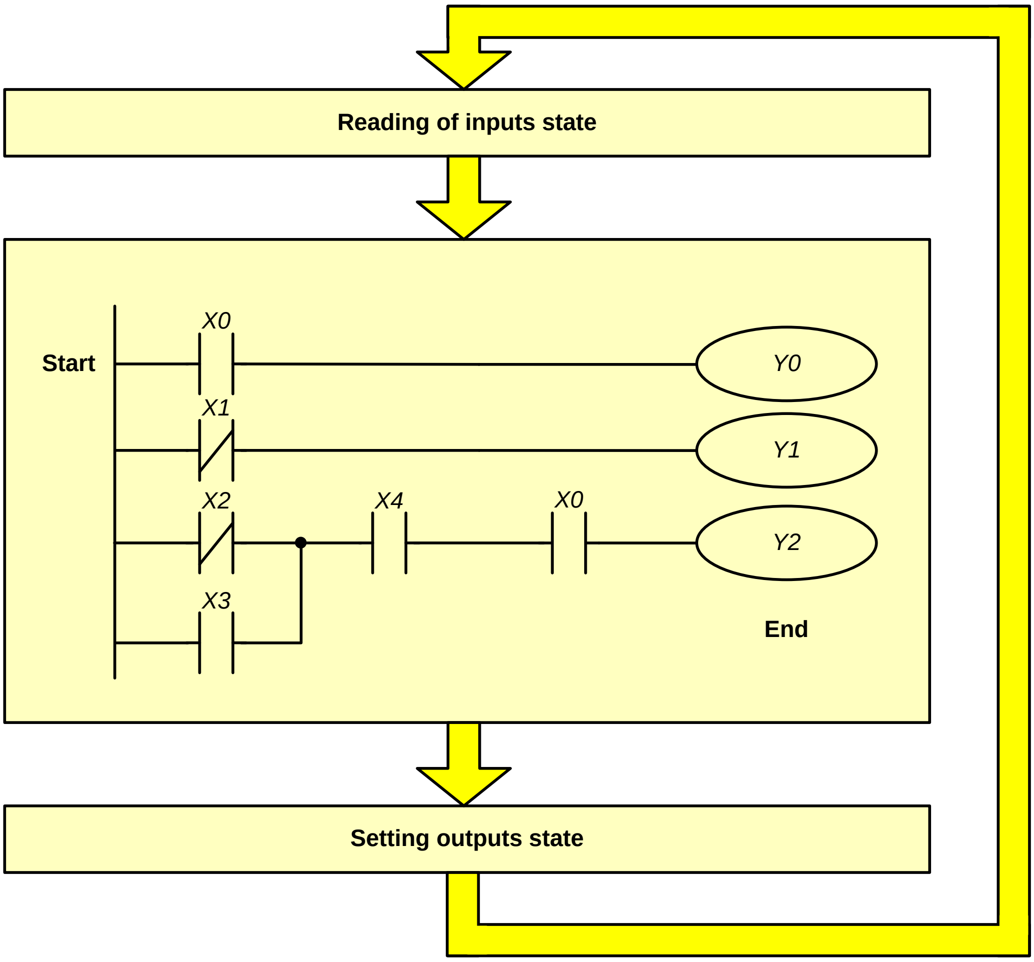


Fig. 11 – operation sequence in the controller

During operation, the controller continuously reads the current state of the inputs and changes the state of the outputs (on/off) depending on the user program.

Fig. 11 shows a flow diagram of one program cycle.

At the first stage the controller reads the state of physical and virtual inputs (the state of which are set via Modbus Coils) and buffers them in the internal memory of the controller.

At the second stage the state of the buffered inputs is processed and the state of the outputs in the controller memory changes according to a given user program. Thus all modifications of the outputs occur without changing their physical condition. During this stage, the state of physical and virtual inputs may change, but the next buffering of the updated state will occur at the first stage of the next cycle of the user program.

At the third stage the controller changes the state of the physical and virtual outputs.

Another difference between the relay-contact logic of the controller and conventional relay-contact electrical circuits is that the user programs run in rows only from left to right and from top to bottom. For example, a circuit with a reverse current direction (section a-b in) will result in an error during compilation in the controller.

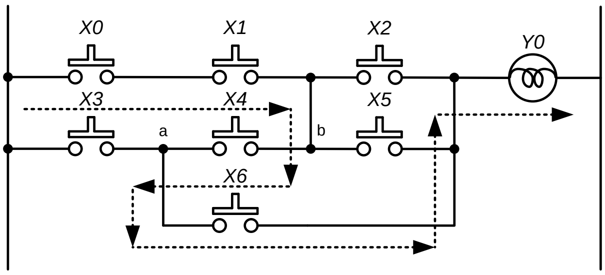
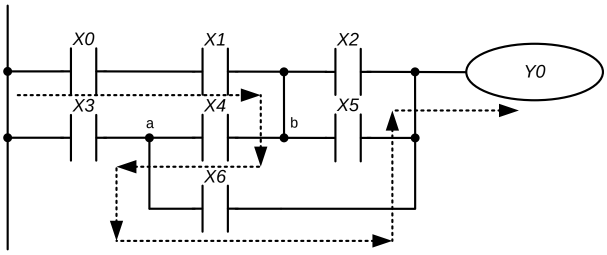


Fig. 12 – electric relay-contact circuit



An error in 3d row

Fig. 13 – relay-contact circuit of the controller

3.3. Operands

All internal objects (devices) of the controller – operands – are divided into few different types and have addresses. Every of types have its own designation and format, which determines what space it takes in the memory of the controller. Thus, for example, input relays are named “X” and have 1-bit format, general purpose data registers are named “D” and have 16-bit (1 word) or 32-bit (2 words) format.

Type and name of the operand		Description
Input	**X**	Input relays. Determine the state of external bit devices, which are connected to input terminals of the controller and state of virtual inputs, which status can be set via Modbus protocol. These operands can take on one of two possible values: 0 or 1. **Addressing is octal**: X0, X1... X7, X10, X11, ...
Output	**Y**	Output relays. Determine the state of output terminals of the controller, to which the load is connected and the state of virtual outputs, which status can be read via Modbus protocol. In the program can be either contacts or coils. These operands can take on one of two possible values: 0 or 1. **Addressing is octal**: Y0, Y1...Y7, Y10, Y11, ...
Merker	**M**	Auxiliary relays. It is a memory for binary intermediary results. In the user program can be either contacts, or coils. These operands can take on one of two possible values: 0 or 1. **Addressing is decimal**: M0, M1...M7, M8, M9,...
Timer	**T**	Time relay. In the program can be used for storage of current timer value and have 16-bit format. Also these operands can be used as contacts and take on one of two states: 0 or 1 **Addressing is decimal**: T0, T1 ...T64
Counter	**C**	Counter is used to implement counting. In the program can be used for storage of current value of the counter and have 16-bit or 32-bit format, and also can be used as a contact and take on one of two possible meanings: 0 or 1 **Addressing is decimal**: C0, C1...C66
Decimal constant	**K**	Determines a **number decimally**
Hexadecimal constant	**H**	Determines a **number hexadecimally**
Floating-point constant	**F**	Determines a **floating-point number**

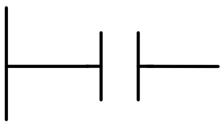
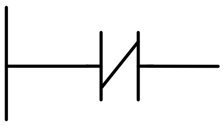
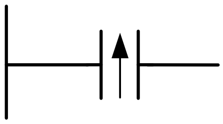
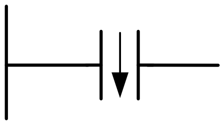
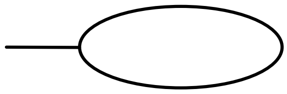
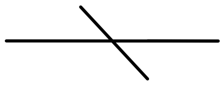
Tallinn Science Park Tehnopol,
Akadeemia tee 21/6, 12618, Tallinn, Estonia
Phone: + 372 6559914,
e-mail: mail@smd.ee
url: https://smd.ee

Data register	**D**	Data storage. 16-bit or 32-bit format. **Addressing is decimal**: D0, D1,..., D391. For 32-bit data one element takes two registers. As an example - for reading 32-bit data from D10 register the data is read from registers D10 and D11.
Index register	**A** **B**	Data storage for intermediary results and for index identification. 16-bit format. Addressing: A0 – A7, B0 – B7 **decimal**
Pointer	**P**	An address for subprogram call. **Decimal**.
Interrupt pointer	**I**	Address of interrupt handling. **Decimal**.

3.4. Graphic symbols of control instructions at a ladder diagram

A relay-contact circuit consists of one vertical line at the left and horizontal lines extending to the right. Vertical line at the left is a bus line, horizontal lines are command lines = steps. There are symbols of entry conditions on command lines leading to commands (instructions) located on the right. The logical combinations of these entry conditions determine when and how right-handed commands are executed.

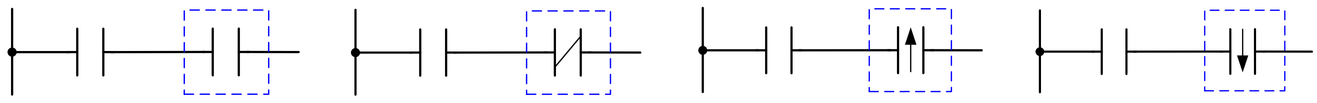
The next symbols are used in relay-contact circuits:

Symbol	**Description**	**Command**	**Operands**
	Input contact – normally open	LD	X, Y, M, T, C
	Input contact – normally closed	LDI	X, Y, M, T, C
	Input pulse contact – rising-edge	LDP	X, Y, M, T, C
	Input pulse contact – falling-edge	LDF	X, Y, M, T, C
	Output signal (coil)	OUT	Y, M
	Basic and application instructions	Refer to chapter XX	Refer to chapter XX
	Logic inversion	INV	-

Input contacts can be combined into serial and parallel blocks:

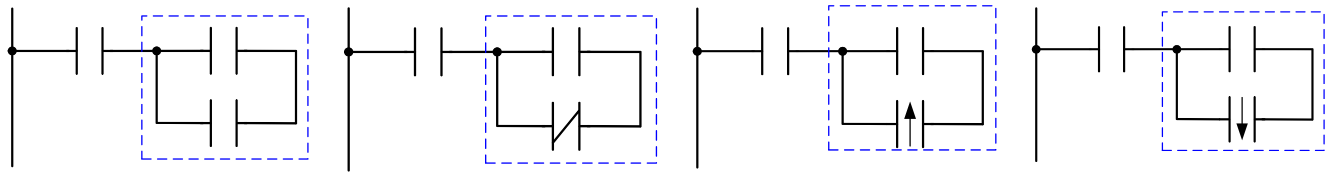
Tallinn Science Park Tehnopol,
Akadeemia tee 21/6, 12618, Tallinn, Estonia
Phone: + 372 6559914,
e-mail: mail@smd.ee
url: https://smd.ee

Serial connections:

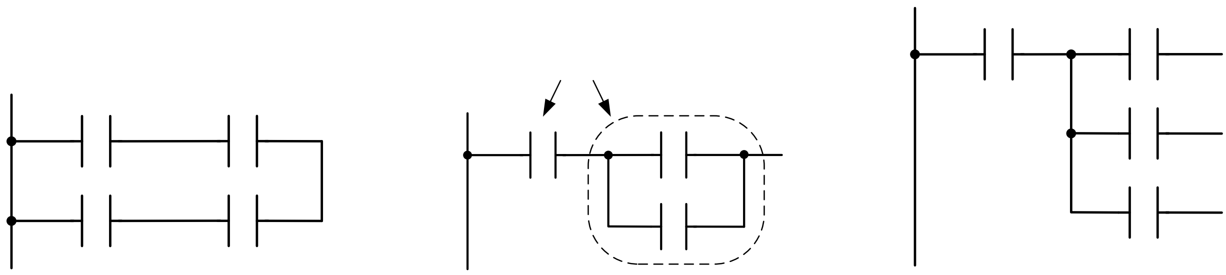


AND ANI ANDP ANDF

Parallel connections:



OR ORI ORP ORF



ORB ANB MPS, MRD, MPP

Fig. 14 – Graphic symbols of control instructions

Program scanning starts from the upper left corner of the diagram and ends in the lower right corner. The following example illustrates the sequence of a program:

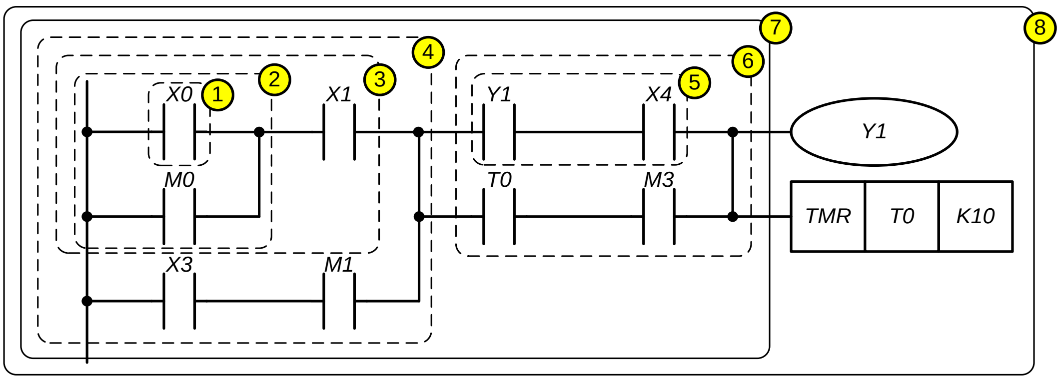
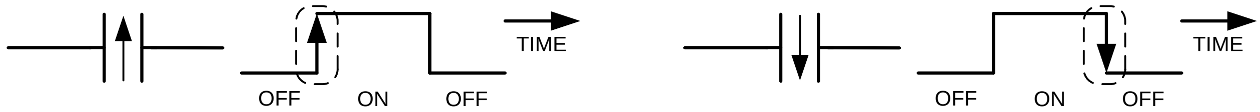


Fig. 15 – sequence of a program

1	LD	X0	
2	OR	M0	
3	AND	X1	
4	LD	X3	
	AND	M3	
	ORB		
5	LD	Y1	
	AND	X4	
6	LD	T0	
	AND	M3	
	ORB		
7	ANB		
8	OUT	Y1	
	TMR	T0	K10

Symbols of input signals with a rising edge (when a signal is switched from 0 to 1) and with a falling edge (when a signal is switched from 1 to 0) are explained below:



Rising edge contact Falling edge contact

Fig. 16 – Edge filtering

The logical block commands ANB and ORB do not correspond to specific conditions on the relay-contact circuit, but describe the relationship between the blocks. The ANB command performs the **LOGIC AND** operation on the execution conditions produced by two logical blocks.

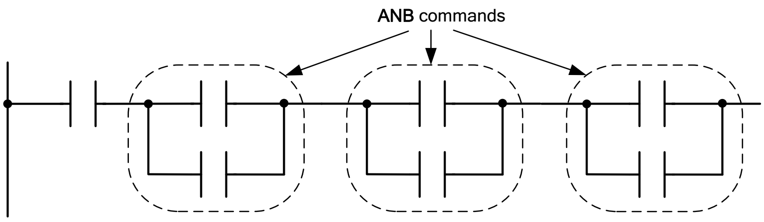


Fig. 17 – ANB instruction

The ORB command performs a **LOGIC OR** operation on the execution conditions produced by two logical blocks.

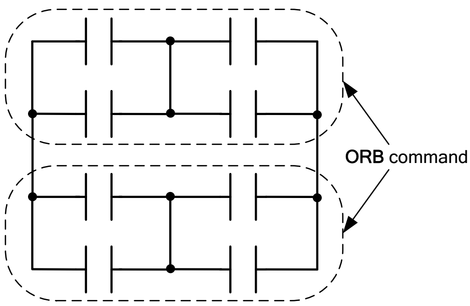
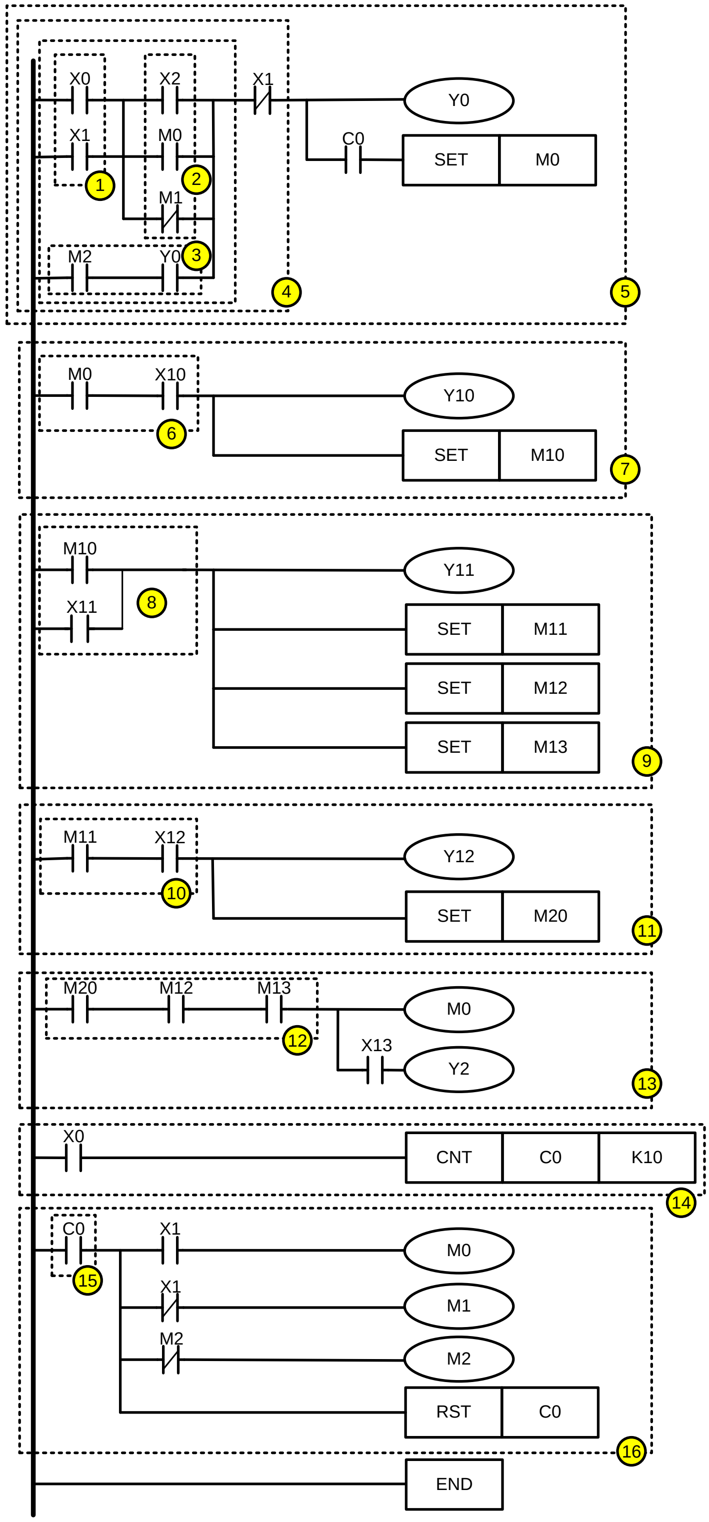


Fig. 18 – ORB instruction

3.5. Convert relay contact circuits (LD) to mnemonic code (IL)

The figure below shows a program presented in the form of relay contact symbols (LD) and a list of instructions - mnemonic code (IL). The figure shows the sequence of converting the ladder diagram (LD) into the code executed by the controller (IL).

Relay contact circuits (LD)




```
LD    X0
OR    X1
LD    X2
OR    M0
ORI   M1
ANB
LD    M2
AND   Y0
ORB
ANI   X1
OUT   Y0
AND   C0
SET   M0
LD    M0
AND   X10
OUT   Y10
SET   M10
LD    M10
AND   X11
OUT   Y11
SET   M11
SET   M12
SET   M13
LD    M11
AND   X12
OUT   Y12
SET   M20
LD    M20
AND   M12
AND   M13
OUT   M0
AND   X13
OUT   Y2
LD    M0
CNT   C0    K10
LD    C0
MPS
AND   X1
OUT   M0
MRD
ANI   X1
OUT   M1
MPP
ANI   M2
OUT   M2
END
```

1 Block OR
2 Block OR
Serial connection of blocks
3 Block AND
4 Parallel connection of blocks
5 The state of the output will be set according to the inputs states during the program executing
Branching
6 Block AND
7 Setting the output and marker
8 Block OR
9 Setting the output and marker
10 Block AND
11 Setting the output and marker
12 Block AND
13 Branching
14
15 Reading of state of C0
16 Branching
End of the program

Fig. 19 – Converting of LD into IL

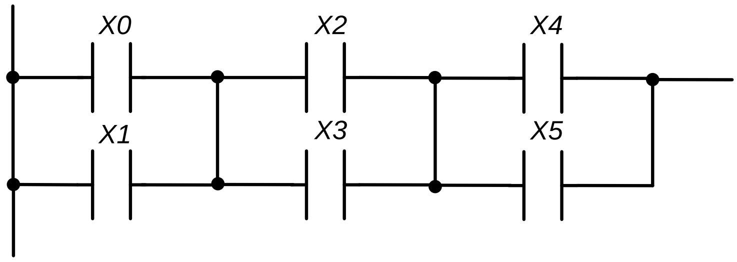
The processing of the relay-contact circuit starts at the upper left corner and ends in the lower right, however, there may be exceptions and various options for converting to mnemonic code, as shown in the following examples:

Example 1

The ladder diagram below can be converted into instruction list in two different ways, but the result will be identical (Fig. 20).

The first encoding method is most preferable as the number of logical blocks is unlimited.

The second method is limited by maximum logic blocks number (max blocks number is 8).

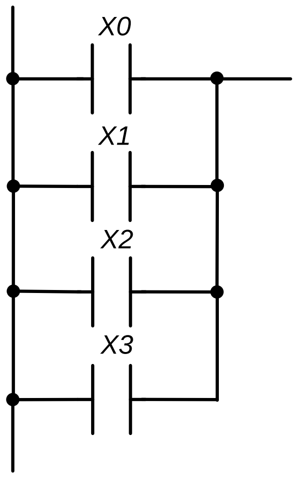


Method 1		Method 2	
LD	X0	LD	X0
OR	X1	OR	X1
LD	X2	LD	X2
OR	X3	OR	X3
ANB		LD	X4
LD	X4	OR	X5
OR	X5	ANB	
ANB		ANB	

Fig. 20 – different methods of using ANB instructions

Example 2

Different encoding methods of parallel connected contacts are shown below (Fig. 21).



Method 1		Method 2	
LD	X0	LD	X0
OR	X1	LD	X1
OR	X2	LD	X2
OR	X3	LD	X3
		ORB	
		ORB	
		ORB	

Fig. 21 – different methods of using ORB instructions

The first method of converting of ladder diagram into instructions list is the most preferable from the point of view of using the controller RAM.

4. Controller functionality

4.1. Operands overview

Type	Operand			Range of addresses		Function
Relays (1-bit memory)	X	External input relays	Physical inputs	X0…X7	Max. 128 points (octal)	Controller inputs
			Virtual inputs (Modbus Coil)	X10…X177		
	Y	External ouput relays	Physical outputs	Y0…Y7	Max. 128 points (octal)	Controller outputs
			Virtual outputs (Modbus Discrete Inputs)	Y10…Y177		
	M	Internal relays (merkers)	General purpose	M0…M99, M111…M127	Max. 128 points	Intermediate binary memory. Corresponds to intermediate relays in electrical circuits.
			Special purpose	M100…M110		
	T	Timers	Resolution 100 ms	T0…T47 (T46, T47 – accumulative)	Max. 64 points	Used as contacts (T), which close when the corresponding timer reaches its set value (TMR command)
			Resolution 10 ms	T48…T63 (T62, T63 – accumulative)		
	C	Counters	Incremental general purpose	C0…C63	Max. 66 points	Used as contacts (C), which close when the corresponding counter reaches its set value (CNT command)
			External pulses	C64, C65		
Registers (16-bit memory)	T	Current timer value		64 points (T0…T63)		Registers for storage of current timers values
	C	Current counter value		66 – 32-bit counters		Registers for storage of current counters values
	D	Data registers	General purpose	D0…D319, D385…D391	Max. 384 points	Used to store data. Special registers configure the controller and display its status
			Nonvolatile(1)	D320…D351		
			Special	D352…D384		
	A	Index registers	Minor 16 bits	A0…A7	Max. 16 points	Can be used for index indication
	B		Major 16 bits	B0…B7		
Pointers	P	Pointers for instructions CALL, CJ		32 points (P0…P31)		Labels for instructions of transitions and subprograms
	I	Interruptions	Communication	I0	Max. 15 points	Labels for subprogram for processing of interruptions
			Timed	I1…I100 (Max. 4 points)		
			External	I1000…I1007		
			Driver's	I2000, I2001		
Constants	K	Decimal constants		K-32768 ...K32767 (16-bit functions) K-2147483648 ...K2147483647 (32-bit functions)		
	H	Hexadecimal constants		H0000...HFFFF (16-bit functions) H00000000...HFFFFFFFF (32-bit functions)		

	F	Floating point constant	F±1.175494351 E-38… 3.402823466 E+38 (32-bit functions only)

(1) – data storage is provided by internal power supply CR2032.

4.2. Addressing and functions of inputs [X] and outputs [Y]

The inputs and outputs in the user program are represented by operands. By specifying the address of the operand, it is possible to refer to the physical and virtual inputs and outputs of the controller during programming.

Discrete inputs/outputs are addressed in octal system, that means the numbers 8 and 9 are not used for inputs and output.

Function of input relays X

Input relays X read the state of external physical devices (buttons, switches, relay contacts, etc.) directly connected to the input terminals of the controller. Each input X can be used in the program an unlimited number of times.

Function of output relays Y

Output relays Y control the state of the physical output contacts of the controller, and therefore the load devices (lamps, relay coils, etc.) directly connected to the output terminals of the controller.

Each output Y can be used in the program an unlimited number of times, but it is recommended to use output coil Y in the program no more than once, because when coil Y is used few times, the output state is determined by the last Y in the scan.

The state of the I/O signals can be read in the program by different instructions.

The process of handling of I/O signals in the controller:

Inputs:

1. The controller reads the state of external input devices, and store it at the beginning of each scan cycle.
2. Changes in the input state during the cycle will not be accepted if the input pulse is very short (less than the time of one scan).

Program:

3. The controller executes the program starting from line 0 and stores the state of all operands in objects memory.

Outputs:

4. After executing the END instruction the state of the output relays Y is written to the memory of the outputs and the states of the output contacts will be changed.

4.3. Addressing and function of internal relays [M]

To store the binary results of logical bindings (signal states "0" or "1"), an intermediate memory (internal relay) is used inside the program. They correspond to intermediate relays in control systems based on relay logic.

Two types of internal relays are used in the controller:

1. General purpose, which are not saved when the power is turned off;
2. Special purpose, which provide the user with additional functionality.

Internal relays are programmed as outputs. They can be used in the program an unlimited number of times. Addressing of internal relays is in decimal format.

Appointment of special merkers:

Merker	Function
M100…M107	These auxiliary relays are used only in conjunction with interruptions from the external inputs I1000 ... I1007 respectively. The value of the merker corresponds to the state of the physical input (IN0 ... IN7) at the moment when the interruption was processed (I1000 ... I1007). The values X0 ... X7 are updated only at the beginning of the next scan of the user program. For example, after getting into the interrupt handler I1004, it is possible to determine the state of input IN4 by requesting the state of M104 (LD M104) (the value of X4 is not relevant in this case).
M108	The rising edge of this auxiliary relay indicates the completion of initialization of the controller peripherals. During subsequent work, the merker maintains a high level value. Resetting the merker reinitializes the controller. For example, after redefining the outputs by the PWM signal generators and the inputs by pulse counters, re-initialization is required. In this case it is necessary to reset M108.
M109	Setting the merker turns on the “ERR” indication on the front panel of the controller, resetting disables it.
M110	Setting this merker and then resetting the M108 will result in a full reboot of the controller.

4.4. Addressing and function of timers [T]

Some control processes require a time relay. Many relay-controlled systems use time relays which switches on delay. The controller uses internal memory elements for these purposes, called timers. The characteristics of the timers can be determined in the program.

Addressing of timers is decimal.

T	Timers	Resolution 100 ms	T0…T47 (T46, T47 – accumulative)	Max. 64 points
		Resolution 10 ms	T48…T63 (T62, T63 – accumulative)	

The required time setting is determined by a decimal constant K, which indicates the number of counted time steps (discrete).

Example: a 100 ms resolution timer set as K5, the actual value of the setting will be 5 x 100 = 500 ms.

The timer operates with on-delay. It is activated with the contact state = 1. After counting the set time value, the timer sets the corresponding input contact T to state “1.” The timer returns to the off state and resets its current value when its input contact is set to “0”.

The setting of the time setting can also be performed indirectly by means of a decimal number recorded earlier in the data register D.

In the controllers the timer begins to count immediately when executes the TMR command.

Explanation of the operation of two types of timers:

General purpose timer

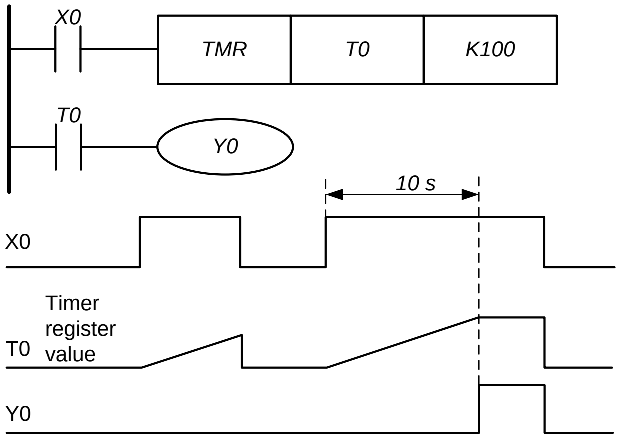


Fig. 22 – General purpose timer operating principle

When the input X0 takes the state "1", the count of the set time begins. After reaching of the programmed 10 seconds, the output Y0 takes the state "1". The timer turns off and the T0 register is reset to zero as soon as the input X0 takes the state "0".

Accumulative tiner

In addition to general purpose timers, the controller has accumulative timers, which, after disabling the control logical connection, save the accumulated time value.

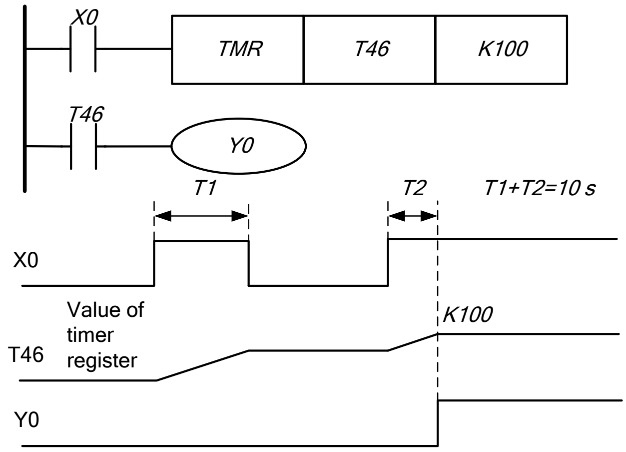


Fig. 23 – Accumulative timer operating principle

4.5. Addressing and function of counters [C]

It is necessary to count impulses (add or subtract) in some control processes. Many relay-controlled systems use pulse counters for this purpose. The controller uses two types or internal memory elements (counters).

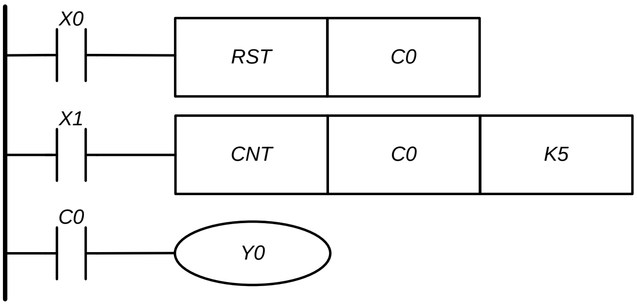
Addressing of timers is decimal.

C	Counters	Incremental general purpose	C0…C63	Max. 66 points
		External pulses (hardware)	C64, C65	

Function of counters:

When the input signal of the counter changes its state from 0 to 1, the current value of counter C increments by one. When it becomes equal to the set value (set point), the counter’s working contact turns on.

```
LD      X0
RST     C0
LD      X1
CNT     C0      K5
LD      C0
OUT     Y0
```



The counter is reset when X0=1: the current value of register C0 = 0, contact C0 is open.

After switching X1 from 0 to 1, the value of C) increments by one.

When register value C0 = 5, contacts C0 and Y0 are closed, all next pulses at input are not counted.

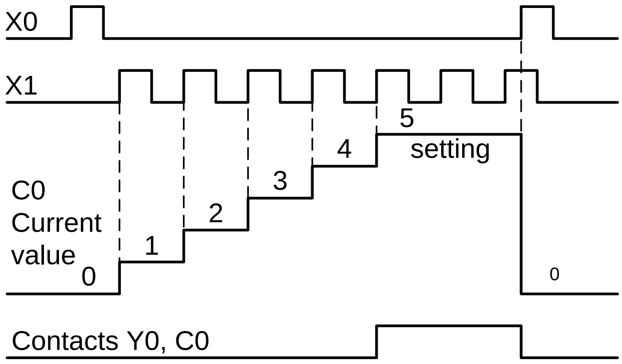
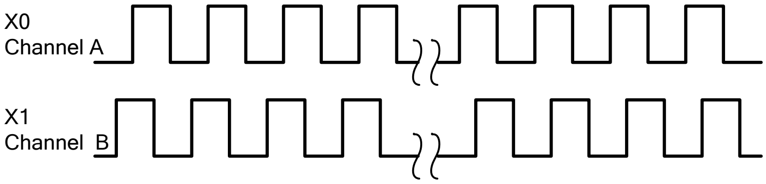


Fig. 24 – counter operating principle

General purpose counters do not count above a threshold, unlike hardware counters, which don't not on the input signal, but only on the physical state of the discrete input, which it refers to. Depending on the value of the configuration register D355, the counters can be configured in the following manner:

Value of the register D355	Configuration
0	The default value. X0 and X1 (IN0 and IN1) operate as discrete inputs.
1	The discrete input X0 refers to the counter C64, which counts rising-edges of pulses. X1 operates as a discrete input.
2	The discrete input X0 refers to the counter C64, which counts falling-edges of pulses. X1 operates as a discrete input.
3	The discrete input X0 refers to the counter C64, which counts both rising-edges and falling edges of pulses. X1 operates as a discrete input.
4	X0 operates as a discrete input. The discrete input X1 refers to the counter C65, which counts rising-edges of pulses.
5	X0 operates as a discrete input. The discrete input X1 refers to the counter C65, which counts falling-edges of pulses.
6	The discrete inputs X0 and X1 refer to the counters C64 and C65 accordingly. The counters count rising-edges of pulses.
7	The discrete inputs X0 and X1 refer to the counters C64 and C65 accordingly. C64 counts falling-edges of pulses, C65 counts rising-edges of pulses.
8	The discrete inputs X0 and X1 refer to the counters C64 and C65 accordingly. C64 counts both rising-edges and falling-edges of pulses, C65 counts rising-edges of pulses.

9	The discrete inputs X0 and X1 refer to the counters C64 and C65. C64 counts rising-edges of pulses, C65 counts falling-edges of pulses.
10	The discrete inputs X0 and X1 refers to the counters C64 and C65 accordingly. The counters count falling-edges of pulses.
11	The discrete inputs X0 and X1 refers to the counters C64 and C65 accordingly. C64 counts both rising-edges and falling-edges of pulses, C65 counts falling-edges of pulses.
12	The discrete inputs X0 and X1 refers to the counter C64 and operate as encoder. A quadrature signal is applied to the inputs. 

4.6. Addressing and function of registers [D], [A], [B]

Data registers [D]

Registers represent the data memory inside the controller. The registers can store numerical values and binary information following one after another.

Data is stored in a 16-bit register (D0, etc.), which can store a number from -32768 to +32767. The joint of two 16-bit registers gives a 32-bit "double register" (D0, D1, etc.), which can store a number from -2147483648 to +2147483647.

Addressing of data registers is decimal. For double-registers (32 bit) addressing starts with the lower 16-bit register.

D	Data registers	General purpose	D0…D319, D385…D391	Max. 384 points
		Non-volatile	D320…D351	
		Special	D352…D384	

There are the next data register types:

General purpose data registers:

These registers are used during user program executing, the data are not saved when the power is off.

Non-volatile data registers:

The data in these registers are saved in the controller memory when the power is off. The memory power supply is provided by internal source CR2032.

Index registers:

This register is used to store intermediate results and to indicate operands.

Special registers:

These registers are used to configure the controller and for access to some special functionality. The numbers of special registers are given in table below:

Register	**Function**	**Values**
D352	The register contains data on the position of the potentiometer “0” on the front panel of the controller.	0…4095

<table>
<tr><td>D353</td><td>The register contains data on the position of the potentiometer “1” on the front panel of the controller.</td><td>0…4095</td></tr>
<tr><td>D354</td><td>The register contains data on the position of the potentiometer “2” “Speed”.</td><td>0…4095</td></tr>
<tr><td>D355</td><td>The register configures input types IN0 and IN1, for more details refer to the section 4.5</td><td>0…12</td></tr>
<tr><td>D356</td><td>The register configures the types of outputs OUT6 and OUT7 for PWM instruction (see section 7. “Application instructions”Error! Reference source not found., PWM instruction).
<table>
<tr><th rowspan="2">Value</th><th rowspan="2">Discretisation time, ms</th><th colspan="2">Function of output</th></tr>
<tr><th>OUT6</th><th>OUT7</th></tr>
<tr><td>0</td><td>–</td><td>output</td><td>output</td></tr>
<tr><td>1</td><td>1</td><td>PWM generator</td><td>output</td></tr>
<tr><td>2</td><td>0,1</td><td>PWM generator</td><td>output</td></tr>
<tr><td>3</td><td>1</td><td>output</td><td>PWM generator</td></tr>
<tr><td>4</td><td>0,1</td><td>output</td><td>PWM generator</td></tr>
<tr><td>5</td><td>1</td><td>PWM generator</td><td>PWM generator</td></tr>
<tr><td>6</td><td>0,1</td><td>PWM generator</td><td>PWM generator</td></tr>
</table>
Refer to the section Error! Reference source not found. (PWM instruction) for detailed information on producing of PWM signal.</td><td>0…6</td></tr>
<tr><td>D357…D384</td><td>Mode-setting and status registers of the stepper motor driver. See the section Error! Reference source not found.” for more details.</td><td>–</td></tr>
</table>

4.7. Index registers [A], [B]

Index registers are used to index operand addresses and change constant values.

The index registers are 16-bit registers.

In 32-bit instructions index registers A and B are used in combination. A contains 16 low-order bits, B contains 16 high-order bits. Index register A is used as the destination address.

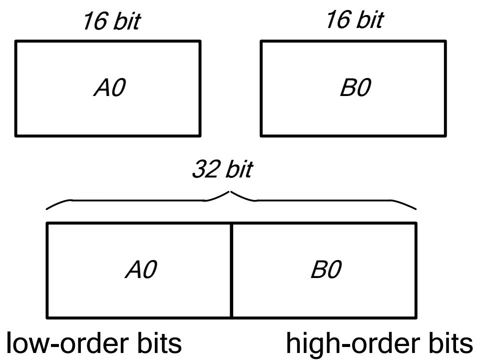
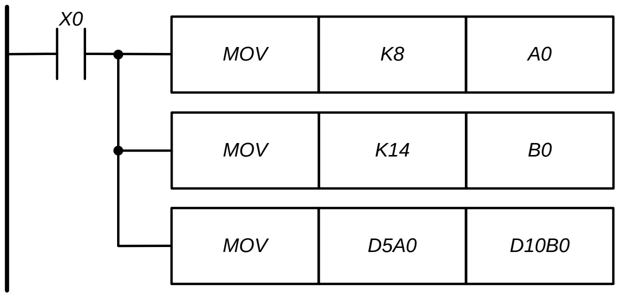


Fig. 25 - Index register structure

Example of data transfer from data register D5A0 to data register D10B0:



When X0 = 1: A0 = 8, B = 14

- Address of transfer source is D5A0 = 5 + 8 = D13
- Destination address is D10B0 = 10 + 14 = 24.
- In this way, data is transferred from the register D13 to the data register D24

Fig. 26 – Data transfer using index registers

Index registers can be used for data transfer and comparison operations in conjunction with byte operands and bit operands.

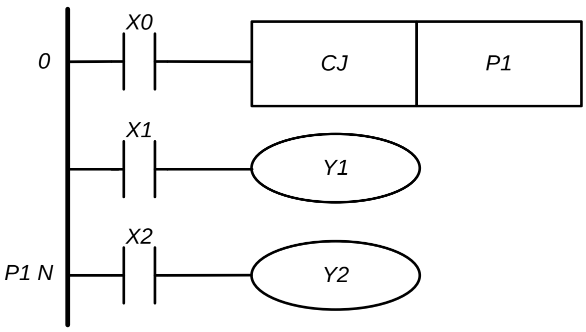
It is also possible to index constants in the same way. When indexing constants, it is required to use the symbol “@”. For example: MOV K10 @ A0 D0B0.

4.8. Pointers [P], [I].

<table>
<tr><td>P</td><td colspan="2">Instruction pointers CALL, CJ</td><td colspan="2">32 points (P0…P31)</td><td>Labels or marks for commands of transition or calling subprograms</td></tr>
<tr><td rowspan="4">I</td><td rowspan="4">Interruptions</td><td>Communication</td><td>I0</td><td rowspan="4">Max. 15 points</td><td rowspan="4">Labels for subprogram for processing of interruptions</td></tr>
<tr><td>Timed</td><td>I1…I100 (Max. 4 points)</td></tr>
<tr><td>External</td><td>I1000…I1007</td></tr>
<tr><td>Driver</td><td>I2000, I2001</td></tr>
</table>

Pointers (P) are used in combination with instructions CJ (transitions) or CALL (subprograms). These pointers are addresses of locations of places or subprograms, which were marked.

An example of executing a CJ jump instruction:



When X0 = 1, after the execution of line 0, the program immediately goes to the line with the pointer P1 and the lines located between them are not executed.

If X0 = 0, the program executes normally step by step.

Fig. 27 – Implementation of the CJ instruction

An example of using subprograms:

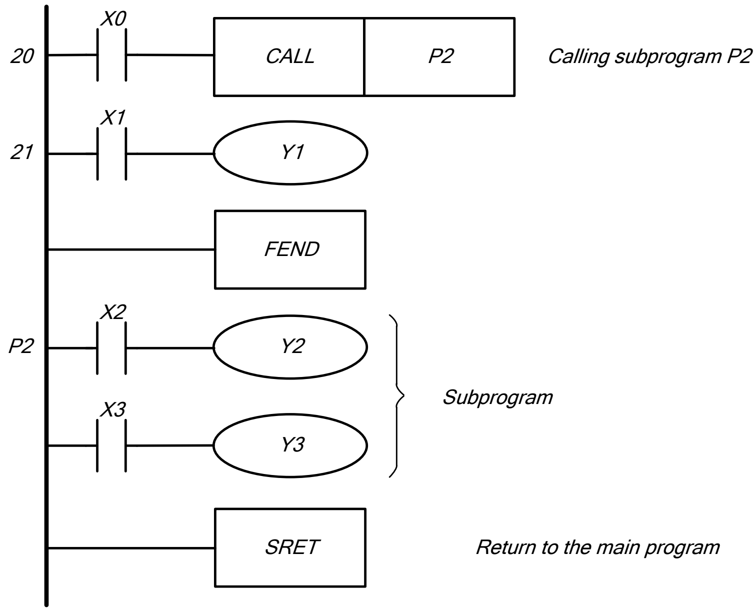


Fig. 28 – Implementation of the CALL instruction

When X0 = 1 at line 20 the program executing goes directly to the line marked P2, the subprogram executes, and after SRET command program executing returns to the line 21.

Interruption pointers (I) are used with instructions EI, DI, IRET for interrupting of main program executing. There are the following types of interruptions:

1. Communication interruption: if the controller receives a broadcast frame via Modbus protocol, it immediately (regardless of the scan cycle) goes to the interrupt processing subprogram which is marked with the pointer I0. It returns to the main program after the IRET instruction is executed.
2. Timed interruption: the interrupt processing subprogram is executed automatically at specified time intervals from 10 to 1000 ms in increments of 10 ms. Totally it is possible to have up to 4 timed interruptions. As an example, inerruptions with pointers I10, I50, I80, I100 will be executed once per 100 ms, 500 ms, 800 ms and 1 s accordingly. Executing returns to the main program after instruction IRET.
3. External interruptions: when the signal at the input IN0 ... IN7 switches from 0 to 1 or from 1 to 0, the controller immediately turns to the execution of the interrupt processing subprogram with the corresponding pointer I (IN0 → I1000, IN1 → I1001, etc.). Return to the main program occurs after the IRET instruction is executed.
4. Driver interruption: when an error occurs during motor phases commutation, which is represented by the special register D381 (ERROR_CODE, more details in the section 8. “Stepper motor driver control”), the controller turns to the execution of the subprogram with the I2000 pointer. When the status of the stepper motor driver changes, register D371 (MOTOR_STATUS, for more details refer to the section 8. “Stepper motor driver control”), the controller turns to the executing of the subprogram with the pointer I2001. The controller returns to the main program after the IRET instruction is executed.

5. Error codes

If the "ERR" LED is on after loading and running the user program, this means that the user program contains an error: a grammatical error or incorrect operand error. Each error that occurs in the controller is recorded in a special register (step number and error code are recorded). This information can be read using a PC or PLC. The table below contains a list of error codes and descriptions.

Address	Type	Size	Description
E004	Input Registers	16-bit	Error code during executing of a user program.
E084	Input Registers	16-bit	Line of the user program, where the error was detected.
E004	Coils		Error flag during user program executing.

Error code	Description
2012h	Unknown command
1007h	Internal error, type of signal collision is not identified.
1005h	Internal error, signal type in case of level collision is not identified.
1006h	Internal error, signal type when inverted level collision is not identified.
2002h	LD instruction, stack overflow.
2001h	The type of the main signal is not identified.
2000h	Processing of LD-type commands, the instruction code has changed.
1001h	Internal error, unknown type of single collision.
1000h	Internal error, single collision at the current value. The signal type of the operand is unknown.
200Bh	Processing of AND-type command when removing the signal from the output stack. Unknown collision type.
1004h	Group collision internal error. Unknown collision type.
1002h	Group collision internal error. The type of operand signal for level collision is not defined.
1003h	Group collision internal error. The type of operand signal for inverse level collision is not defined.
200Ch	Processing of AND-type command, the instruction code has changed.
200Dh	Processing of OR-type command, the instruction code has changed.
2010h	There are not entries in the main stack when the ANB instruction is applied.
200Fh	Applying of ANB instruction error. There are not entries in the output stack, and there is only one entry in the main stack.
200Eh	Unknown signal in the output stack with ANB command.
2011h	The absence of at least two elements in the main stack for applying the ORB instruction.
2013h	Brunching stack overflow, instruction MPS.
2016h	Brunching stack is empty, instructions MRD, MPP.
2015h	Main stack overflow, instructions MRD, MPP.
2014h	Signal type is not recognized when assigning selector, instructions MRD, MPP.
2017h	Stack overflow, instruction NEXT.
3012h	Prescan – index P is ut of range.
3014h	Prescan – index I is ut of range.
3013h	Prescan. Unable to create a new timed interruption, limit on quantity exceeded.
201Dh	Incorrect operand type, instructions CJ/CJP.
201Ch	Operand is out of range, instructions CJ/CJP.
2024h	Incorrect operand type, instructions CALL/CALLP.
2023h	Operand is out of range, instructions CALL/CALLP.

Error code	Description
2025h	There are no return points in the stack, instruction SRET.
2004h	A command END/FEND was received during interruption processing.
202Ah	A command IRET was received in the main program.
2056h	Instruction END, main stack is not empty.
2057h	Instruction END, brunching stack is not empty.
2058h	Instruction END, cycles stack is not empty.
2059h	Instruction END, subprograms stack is not empty.
2026h	Instruction IRET, main stack is not empty.
2027h	Instruction IRET, brunching stack is not empty
2028h	Instruction IRET, cycles stack is not empty.
2029h	Instruction IRET, subprogram stack is not empty.
2020h	Instruction CALL/CALLP, unknown index operand.
201Eh	Instruction CALL/CALLP, index operand A is out of range.
201Fh	Instruction CALL/CALLP, index operand B is out of range.
201Ah	Instruction CJ/CJP, unknown index operand.
2018h	Instruction CJ/CJP, index operand A is out of range.
2019h	Instruction CJ/CJP, index operand B is out of range.
201Bh	Instruction CJ/CJP, the requested pointer does not exist.
2022h	Instruction CALL/CALLP, the requested mark does not exist.
2021h	Instruction CALL/CALLP, stack overflow.
2003h	Incorrect operand, instruction OUT.
200Ah	Incorrect operand, instruction SET/RST.
2005h	Instruction SET can not be applied to operand C.
2006h	Instruction SET can not be applied to operand T.
2007h	Instruction SET can not be applied to operand D.
2008h	Instruction SET can not be applied to operand A.
2009h	Instruction SET can not be applied to operand B.
202Dh	Instruction INV, unknown signal type.
202Bh	Instruction TMR, the first argument is not typical.
202Ch	Instruction CNT, the first argument is not typical.
202Eh	Instruction INC/DEC, incorrect operand.
2037h	Instruction ADD/SUB/MUL/DIV/WAND/WOR/WXOR, type of 3d operand is incorrect.
2038h	Instruction NEG/ABS, incorrect operand type.
2030h	Instruction CMP, type of 3d operand is incorrect.
2031h	Instruction ZCP, type of 3d operand is incorrect.
202Fh	Instruction MOV/BMOV/FMOV, incorrect type of destination operand.
2039h	Instruction XCH, data type of 1st operand is incorrect.
203Ah	Instruction XCH, data type of 2d operand is incorrect..
203Bh	Instruction ROR/ROL, data type of 1st operand is incorrect.
2033h	Instruction ZRST, operands are not of the same type
2032h	Instruction ZRST, operand type is incorrect.
2036h	Instruction DIV, division by zero of an integer.
2046h	Instruction DECO, type of 2d operand is incorrect.
2047h	Instruction ENCO, type of 2d operand is incorrect.
2048h	Instruction SUM, type of 2d operand is incorrect.
2049h	Instruction BON, type of 2d operand is incorrect
204Bh	Instruction SQR, type of 2d operand is incorrect.

Error code	Description
204Ah	Instruction SQR, negative value.
204Ch	Instruction POW, type of 3d operand is incorrect.
203Ch	Instruction FLT, type of 2d operand is incorrect.
203Dh	Instruction INT, type of 2d operand is incorrect.
203Eh	Instruction PWM, the third operand is not applicable for PWM signal output.
203Fh	Instruction PWM, type of 3d operand is incorrect.
2041h	Instruction DECMP, type of 1st operand is incorrect.
2040h	Instruction DECMP, type of 2d operand is incorrect.
2042h	Instruction DECMP, type of 3d operand is incorrect.
2045h	Instruction DEZCP, type of 3d operand is incorrect.
2044h	Instruction DEZCP, type of 1st operand is incorrect.
2043h	Instruction DEZCP, type of 2d operand is incorrect.
2050h	Instruction DEADD/DESUB/DEMUL/DEDIV/DEPOW, type of 3d operand is incorrect.
204Fh	Instruction DEADD/DESUB/DEMUL/DEDIV/DEPOW, type of 1st operand is incorrect.
204Eh	Instruction DEADD/DESUB/DEMUL/DEDIV/DEPOW, type of 2d operand is incorrect.
204Dh	Instruction DEDIV, divide by zero.
3015h	Prescan error, unknown command detected.
2053h	Instruction DESQR, type of 1st operand is incorrect.
2052h	Instruction DESQR, type of 2d operand is incorrect.
2051h	Instruction DESQR negative value.
2035h	Instruction LD#, stack overflow.
2034h	Instruction LD#, main signal type not recognized.
4000h	Switch interruptions queue overflow.
4001h	Timed interruptions queue overflow TIM0.
4002h	Timed interruptions queue overflow TIM1.
4003h	Timed interruptions queue overflow TIM2.
4004h	Timed interruptions queue overflow TIM3.
4005h	External interruptions queue overflow IN0.
4006h	External interruptions queue overflow IN1.
4007h	External interruptions queue overflow IN2.
4008h	External interruptions queue overflow IN3.
4009h	External interruptions queue overflow IN4.
400Ah	External interruptions queue overflow IN5.
400Bh	External interruptions queue overflow IN6.
400Ch	External interruptions queue overflow IN7.
400Dh	Driver interruptions queue overflow.
400Eh	Motor status change interruption queue overflow.
2054h	Instruction TRD, incorrect operand type.
2055h	Instruction TWR, incorrect operand type.
3000h	Stacks are empty, no signal value.
3001h	The main stack is empty, no signal value.
3003h	The index register value is out of range.
3004h	Index of operand X is out of range.
3005h	Index of operand Y is out of range.
3006h	Index of operand M is out of range.
3007h	Index of operand C is out of range.
3008h	Index of operand T is out of range.

Error code	Description
3009h	Index of operand A/B is out of range.
300Ah	Index of operand D is out of range.
300Bh	Index of operand P is out of range.
300Ch	Index of operand I is out of range.
300Dh	Unknown operand type
300Fh	Impossible to get the operand value.
300Eh	FLOAT number is used with 16-bit instruction.
3010h	Getting operand value - incorrect operand type.
3011h	Getting token of operand - incorrect operand type.
5000h	Power supply + 5V - short circuit
205Ah	Instruction TWR, incorrect time format.

6. Basic instructions

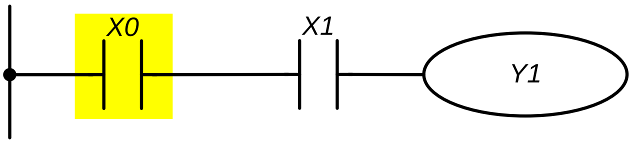
Instruction	Function
LD	Normally open contact

Operand	X	Y	M	T	C	A	B	D
	●	●	●	●	●			

Description:

Instruction LD is used as a normally open contact for programming of starts of logical chains. It is located at left in contact scheme and connected directly to the power bus line.

Use:



LD	**X0**
AND	X1
OUT	Y1

The instruction LD X0 "normally open contact X0" starts the sequential logic connection. If at the inputs X0 and X1 there is simultaneously a signal "1", then the output Y1 will be set to the state "1".

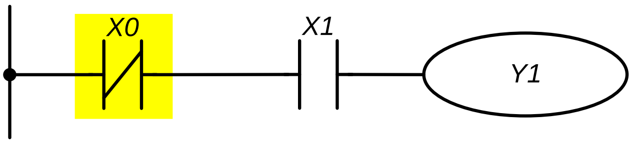
Instruction	Function
LDI	Normally closed contact

Operand	X	Y	M	T	C	A	B	D
	●	●	●	●	●			

Description:

Instruction LDI is used as a normally closed contact for programming of starts of logical chains. It is located at left in contact scheme and connected directly to the power bus line.

Use:



LDI	**X0**
AND	X1
OUT	Y1

The instruction LDI X0 "normally closed contact X0" starts the sequential logic connection. If at the inputs X0 and X1 there is simultaneously a signal "1", then the output Y1 will be set to the state "1".

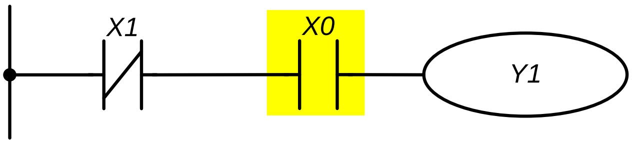
Instruction	Function
AND	Series connection - normally open contact (logic AND)

Operand	X	Y	M	T	C	A	B	D
	●	●	●	●	●			

Description:

Instruction AND is used as a series connected normally open contact for programming of logical multiplication operation (AND). The instruction represents a logical operation and therefore cannot be programmed at the beginning of the sequence. For sequence beginning instructions LD or LDI must be used.

Use:



LDI	X1
AND	**X0**
OU	Y1

The instruction AND X0 "Series connection - normally open contact X0" creates a series logical connection with contact X1 and used to perform the logical multiplication operation. If there is “0” at input X1 and “1” at X0, then output Y1 turns to the state “1”.

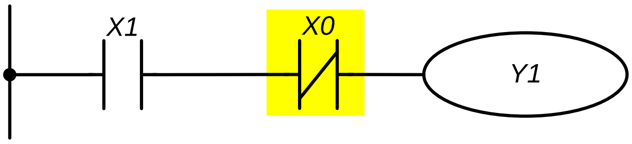
Instruction	Function
ANI	Series connection - normally closed contact (logic NAND)

Operand	X	Y	M	T	C	A	B	D
	●	●	●	●	●			

Description:

Instruction ANI is used as a series connected normally closed contact for programming of logical operation NAND (AND NOT). The instruction represents a logical operation and therefore cannot be programmed at the beginning of the sequence. For sequence beginning instructions LD or LDI must be used.

Use:



LD	X1
ANI	**X0**
OUT	Y1

The instruction "Series connection - normally closed contact X0" creates a series logical connection with contact X1 and used to perform the logical operation NAND. If there is “1” at input X1 and and “0” at X0, then output Y1 turns to the state “1”.

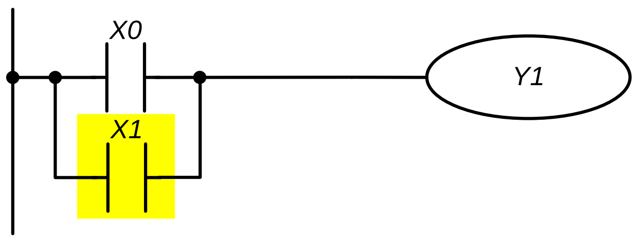
Instruction	Function
OR	Parallel connection – normally open contact (logic OR)

Operand	X	Y	M	T	C	A	B	D
	●	●	●	●	●			

Description:

The instruction OR is used as a parallel connected normally open contact for programming of logical addition (OR). The instruction represents a logical operation and therefore cannot be programmed at the beginning of the sequence. For sequence beginning instructions LD or LDI must be used.

Use:




```
LD    X0
OR    X1
OUT   Y1
```

The instruction "Parallel connection – normally open contact X1" creates a parallel logical connection with contact X0 and used to perform the operation of logical addition. If at least one of the inputs X0 or X1 is "1", then the output Y1 turns to the state "1".

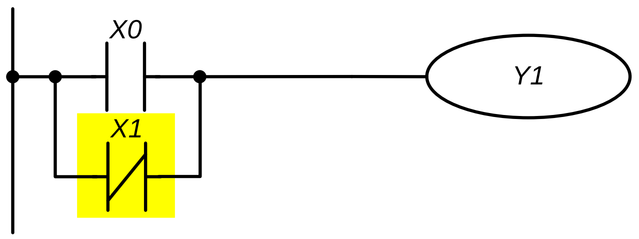
Instruction	Function
ORI	Parallel connection – normally closed contact (logic NOR)

Operand	X	Y	M	T	C	A	B	D
	●	●	●	●	●			

Description:

The instruction ORI is used as a parallel connected normally closed contact for programming of logical operation NOR (OR NOT). The instruction represents a logical operation and therefore cannot be programmed at the beginning of the sequence. For sequence beginning instructions LD or LDI must be used.

Use:




```
LD    X0
ORI   X1
OUT   Y1
```

The instruction "Parallel connection – normally closed contact X1" creates a parallel logical connection with contact X0 and used to perform the operation of logical instruction NOR (OR NOT). If the input X0 is "1" or the input X1 is "0" (one or both conditions at the same time), then the output Y1 turns to the state "1".

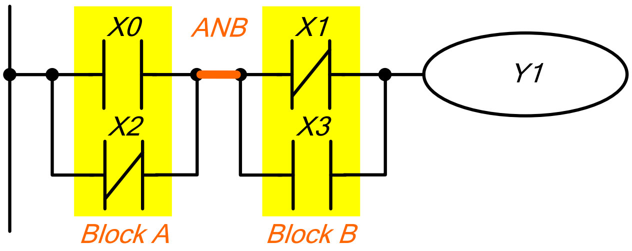
Instruction	Function
ANB	«AND»-block: series connection of blocks

Description:

- The instruction ANB is used for series connection of two logical chains (blocks). Separate blocks of parallel connected elements are entered into the program separately. To connect these blocks in series, an ANB instruction is programmed after each block.
- Start branching programmed using LD or LDI instructions.
- ANB instruction is independent and does not require any operands.
- ANB instruction within the whole user program can be used unlimited times.
- Instruction ANB is shown as a series connection in a contact diagram. The instruction ANB in a list of IL language instructions can be shown in a contact circuit as a jumper.

— If it is necessary to connect few separate blocks one after another, the number of LD/ LDI instructions and also the number of ANB instructions must be limited by 8.

Use:




```
LD    X0
ORI   X2
LDI   X1
OR    X3
ANB
OUT   Y1
```

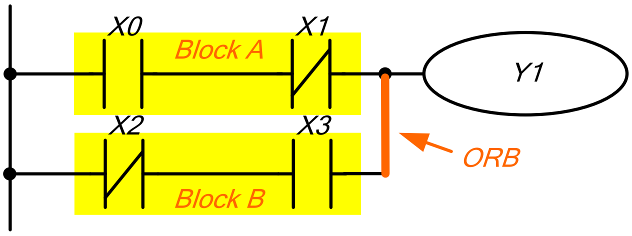
The instruction ANB creates a series logical connection between two logic blocks (Block A and Block B).

Instruction	Function
ORB	«OR»-block: parallel connection of blocks

Description:

— The instruction ORB is used for parallel connection of two or more series connected contacts or blocks. If several series connected blocks are connected in parallel, it is necessary to add ORB instruction after every block.
— The branching start is programmed using the LD or LDI instructions.
— ORB-Instruction is independent and does not require any operands.
— ORB-Instruction within the user program can be used unlimited times
— If several separate blocks are programmed directly one after another, it is necessary to limit the number of LD and LDI instructions and also the number of ORB instructions to 8.
— The ORB instruction is shown as a parallel connection in a contact diagram. The instruction ORB in a list of IL language instructions can be shown in a contact circuit as a jumper.

Use:




```
LD    X0
ANI   X1
LDI   X2
AND   X3
OR
OUT   Y1
```

The instruction ORB creates a series logical connection between two logic blocks (Block A and Block B).

Instruction	Function
MPS	Offset down the stack

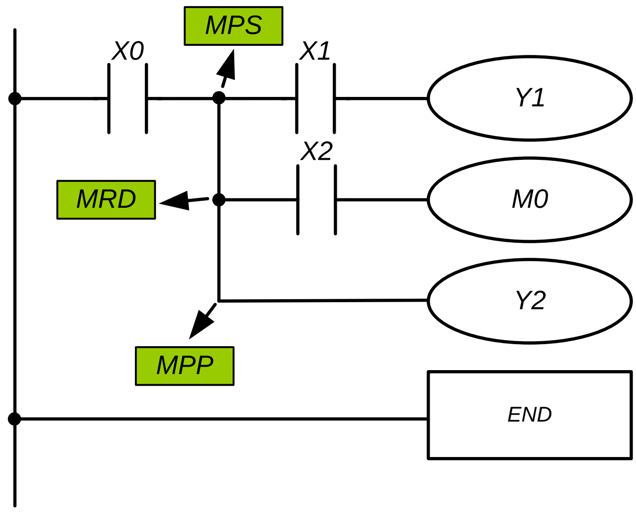
Instruction	Function
MRD	Read value from the stack

Instruction	Function
MPP	Exit from the stack

Description:

- The instructions MPS, MRD, MPP are used to create levels of logical connections (for example, after one initial logical expression, create several logical expressions at the output, i.e., turn on several output coils).
- Using the MPS instruction, the previous result of logical connections (processing of a logical expression) is stored.
- Using the MRD instruction, it is possible to create several independed branches between the beginning (MPS) and the end (MPP) of the branch. The result of processing a logical expression at MPS point is taken into account at each branch.
- The last branch is created by MPP-instruction.
- The branching opened with the MPS instruction must always be closed by the MPP instruction.
- MPS, MRD, MPP instructions don’t need any operands.
- These instructions are not shown in the contact diagram. If programming is done in a contact circuit, the branches are used as usual. When convert a user program from a ladder diagram (LD) to an instruction list (IL), MPS-, MRD- and MPP-instructions should be added to IL.

Use:



LD	X0
MPS	
AND	X1
OUT	Y1
MPD	
AND	X2
OUT	M0
MPP	
OUT	Y2
END	

MPS

An intermediate result (X0 value) at the 1st level of logical connections is listed on the 1st place in the stack memory of intermediate connections. Logical multiplication of X1 with X0 is performed and output Y1 is set.

MRD

Before executing the next instruction, an intermediate result at the 1st place of the memory of logical connections is read. Logical multiplication of X2 with X0 is performed and the output of M0 is set.

MPP

Before executing the next instruction, an intermediate result at the 1st place of the memory of logical connections is read. The output Y2 is set. The operation at the 1st level of intermediate results is completed, and the memory of logical connections is cleared.

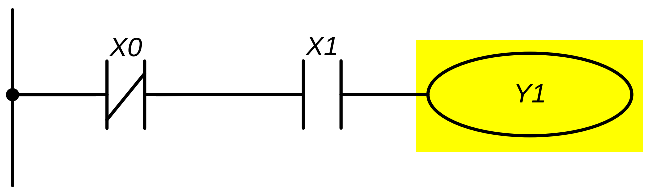
Instruction	Function
OUT	Output coil

Operand	X	Y	M	T	C	A	B	D
		●	●					

Description:

- The instruction OUT is used to set an output coil depending on the result of logical connections (the result of processing the logical expression by the controller).
- Using the OUT instruction, it is possible to end the programming of a line (logical expression).
- Programming several OUT instructions as a result of processing a logical expression is also possible.
- The result of logical connections represented by the OUT instruction can be applied in the next program steps as the state of the input signal, i.e. it can be read many times in many logical expressions.
- The result of logical connections represented by the OUT instruction is active (on) as long as the conditions for its turning on are valid.
- When programming the double recording of the same outputs (their addresses), problems may arise during program execution. Avoid double recording the outputs, as this can lead to interference when running the program.

Use:




```
LDI   X0
AND   X1
OUT   Y1
```

If X0 = 0 and X1 = 1 – the instruction OUT Y1 set the state of the output Y1 = "1".

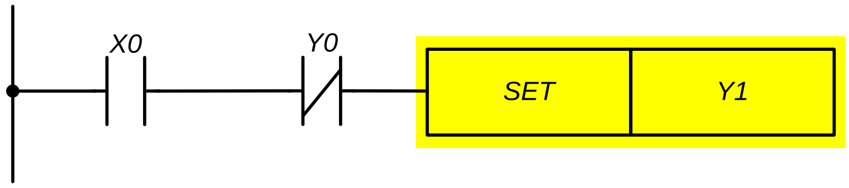
Instruction	Function
SET	Turning on latched output

Operand	X	Y	M	T	C	A	B	D
		●	●					

Description:

- The state of operand can be set directly by the SET instruction.
- Operands Y and M can be turned on by the SET instruction.
- As soon as the entry condition is established for the SET instruction (signal "1"), the corresponding operand turns on.
- If the entry conditions for the SET instruction are no longer satisfied, the corresponding operand remains on.

Use:



LD X0
ANI Y0
SET Y1

Output Y1 turns on when entry conditions (X0, Y0) are satisfied. After that the output Y1 doesn't depend on entry conditions. The only way to turn the output Y1 off is to use RST instruction or to turn off the controller power supply.

Instruction	**Function**
RST	Reset of operand state

Operand	**X**	**Y**	**M**	**T**	**C**	**A**	**B**	**D**
		●	●	●	●	●	●	●

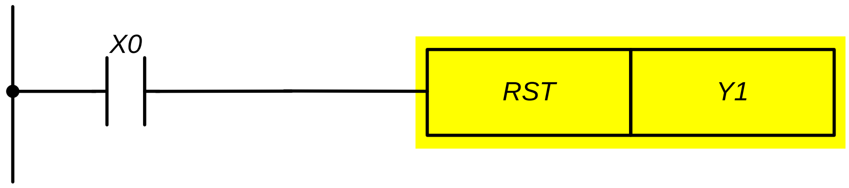
Description:

The state of an operand can be reset directly.

RST-instruction turns of corresponding operands. It means:

— Outputs Y, contacts M are turned off (signal state "0").
— Current values of timers and counters, values of registers D, A and B are reset to "0".
— As soon as the entry condition is established for the RST instruction (signal "1"), the corresponding operand turns off.
— If the entry conditions for the RST instruction are no longer satisfied, the corresponding operand remains off.

Use:



LD X0
RST Y1

The output Y1 turns off when condition X1 is satisfied and remains off even when condition X0 is not met.

Instruction	**Function**
TMR	Timer (16-bit)

Operands	**K**	**H**	**F**	**X**	**Y**	**M**	**T**	**C**	**A**	**B**	**D**
S1							●				
S2	●	●					●	●	●	●	●

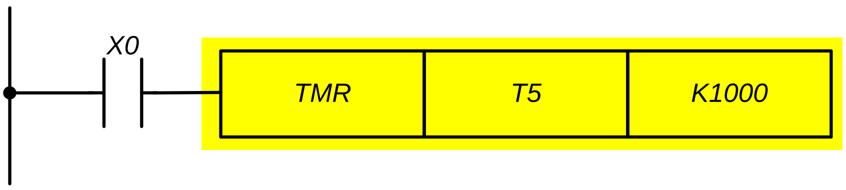
Description:

— The Instruction TMR is used to set a signal state (turn on/off) depending on the result of logical connections after a period of time specified in the instruction.
— Using the TMR instruction, it is possible to end the programming of a line (logical expression).

— The result of logical connections represented by the TMR instruction can be used in the next program steps as the state of the input signal, i.e. can be read many times in many logical expressions

— The result of logical connections represented by the TMR instruction is active (turned on) as long as the entry conditions are valid.

Use:



LD	X0	
TMR	**T5**	**K1000**

Upon condition X0 = 1 the instruction TMR T5 counts until the value in the T5 register reaches the value of K1000 (100 sec). If X0 = 0, the execution of the TMR instruction will stop and T5 will reset to "0".

Instruction		Function
CNT	S1 S2	Counter (16-bit)
DCNT		Counter (32-bit)

Operands	K	H	F	X	Y	M	T	C	A	B	D
S1								•			
S2	•	•					•	•	•	•	•

Information

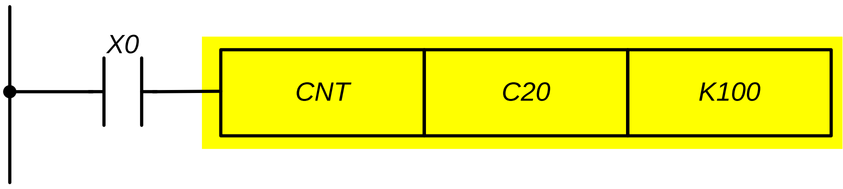
D Usually, to use 32-bit instructions, the prefix "D" is added to the name of the instruction. D – only a 32-bit version of the instruction exists.

P For impulse instructions with a one scan "lifetime", the postfix "**P**" is added. The concept of a *single scan* should be attributed to the used operand. For example, the operand **M0** on line 7 of the main program was set from "0" to "1". Now for all instructions below that are before FEND or END, the **M0** operand has a pulse component on the leading edge (the result for LDP M0 will be "1"), as well as for instructions starting from the 0th line to the 6th line during next scan the operand **M0** will has a pulse component. When go to line 7 (or lower if a CJ command is used), **M0** will have a high signal level without pulse components. Thus, one circle was made along the body of the program - one scan, shifted to the operand change line. In case of interruptions arise before reaching the line 7, the operand **M0** keeps the pulse component until returning to the main program. If the operand **M0** has been modified in an interruption or subprogram, then the place where the operand is changed is considered to be the line from which the transition to the subprogram or the main program line was carried out, before processing of which the interrupt handler was called.

Description:

- The instruction CNT is used to summarize the number of closures of the input contact and assign the signal state (turn on the output) when the current counter value reaches the set value.
- Using the CNT instruction, it is possible to end the programming of a line (logical expression).
- \- The result of logical connections represented by the CNT instruction can be applied in the next program steps as the state of the input signal, i.e. can be read many times in many logical expressions.
- To reset the current value of a counter use RST instruction.
- **Attention:** hardware counters count above the threshold and work regardless of the presence of an input signal

Use:



LD	X0	
CNT	**C20**	**K100**

When X0 changes from “0” to “1” the value of the register C20 increases by 1. It repeats until the value of register C20 reaches K100 (100 pulses). After that the count stops, the contact C20 turns on. To reset the value of the register C20 use the instruction RST C20.

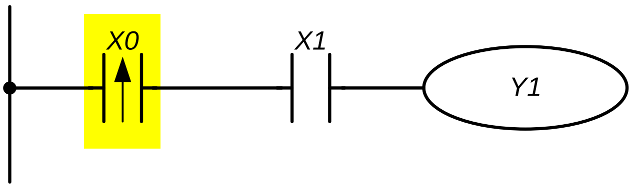
Instruction	**Function**
LDP	Beginning of logical expression with a rising edge polling (impulse)

Operand	**X**	**Y**	**M**	**T**	**C**	**A**	**B**	**D**
	●	●	●	●	●			

Description:

- The instruction LDP is used to program the pulse start of a logical connection.
- The instruction LDP must be programmed at the beginning of the circuit.
- The LDP instruction is also used in conjunction with the ANB and ORB instructions to start branching.
- The LDP instruction after a positive edge is stored for the duration of the program cycle (scan).

Use:



LDP	**X0**
AND	X1
OUT	Y1

The instruction "LDP X0" starts the series logical connection. If the input X0 changes from "0" to "1" (and X1 = 1), then the output Y1 keeps the state "1" during one scan.

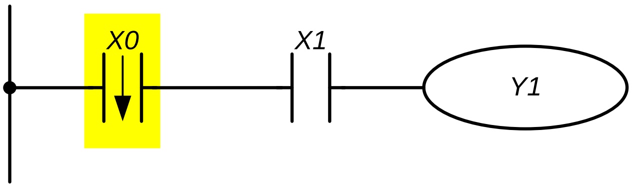
Instruction	**Function**
LDF	Beginning of a logical expression with a falling edge polling (impulse)

Operand	X	Y	M	T	C	A	B	D
	●	●	●	●	●			

Description:

- The instruction LDF is used to program the pulse start of a logical connection.
- The instruction LDF must be programmed at the beginning of the circuit.
- The LDF instruction is also used in conjunction with the ANB and ORB instructions to start branching.
- The LDF instruction after a negative edge is stored for the duration of the program cycle (scan).

Use:



LDF X0 3
AND X1
OUT Y1

The instruction "LDF X0 starts the series logical connection. If the input X0 changes from "1" to "0" (and X1 = 1), then the output Y1 keeps the state "1" during one scan.

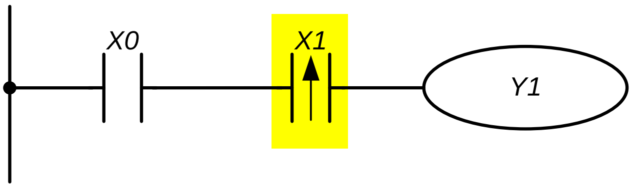
Instruction	Function
ANDP	«AND» with rising edge polling (impulse)

Operand	X	Y	M	T	C	A	B	D
	●	●	●	●	●			

Description:

- The instruction ANDP is used for programming of a series connected pulse contact with rising edge polling (impulse).

Use:



LD X0
ANDP X1
OUT Y1

The instruction "ANDP X1" creates a series logic connection. If input X1 changes from "0" to "1" (and X0 = 1), then the output Y1 keeps state "1" during one scan.

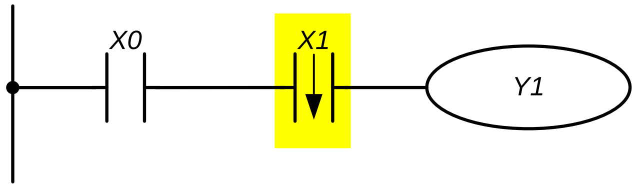
Instruction	Function
ANDF	«AND» with polling on a falling edge (impulse)

Operand	X	Y	M	T	C	A	B	D
	●	●	●	●	●			

Description:

- The instruction ANDF is used for programming of a series connected pulse contact with falling edge polling (impulse).

Use:



LD	X0
ANDF	**X1**
OUT	Y1

The instruction "ANDF X1" creates a series logic connection. If input X1 changes from "1" to "0" (and X0 = 1), then the output Y1 keeps state "1" during one scan.

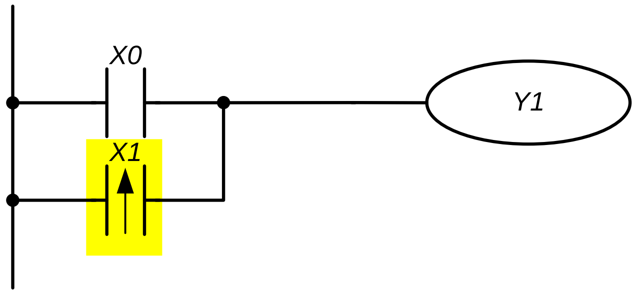
Instruction	Function
ORP	«OR» with rising edge polling (impulse)

Operand	X	Y	M	T	C	A	B	D
	●	●	●	●	●			

Description:

— The instruction ORP is used for programming of a parallel connected pulse contact with rising edge polling (impulse).

Use:



LD	X0
ORP	**X1**
OUT	Y1

The instruction "ORP X1" creates a parallel logic connection. The output Y1 will keep state "1" during one scan if the input X1 changes from "0" to "1" or X0 = 1.

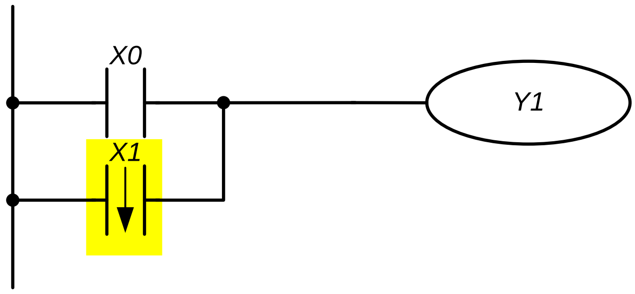
Instruction	Function
ORF	«OR» with falling edge polling (impulse)

Operand	X	Y	M	T	C	A	B	D
	●	●	●	●	●			

Description:

— The instruction ORF is used for programming of a parallel connected pulse contact with falling edge polling (impulse).

Use:



LD	X0
ORF	**X1**
OUT	Y1

The instruction "ORF X1" creates a parallel logic connection. The output Y1 will keep state "1" during one scan if the input X1 changes from "1" to "0" or X0 = 1.

Instruction	Function
END	End of program

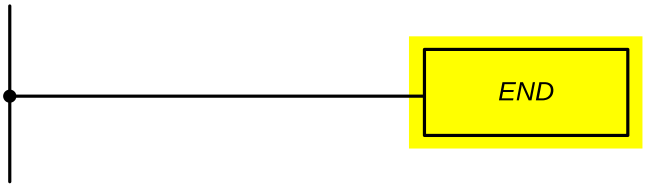
Description:

The end of a user program and transition to the beginning of the program (step 0).

— Each controller program must end with an END instruction.

— If an END instruction is being programmed, then at this point the processing of the program ends. Subsequent areas of the program are no longer taken into account. After processing of the END instruction, the outputs are set and the program starts (step 0).

Use:



END

Instruction	Function
FEND	End of main program

Description:

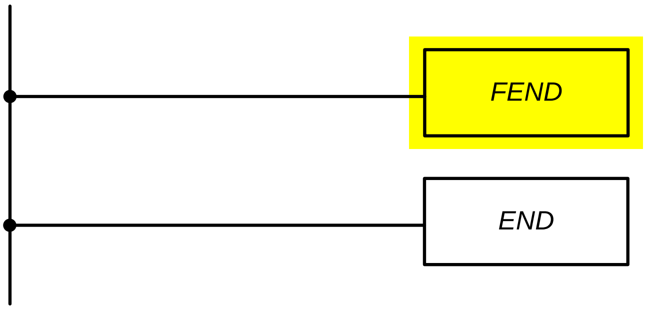
The end of the main user program and the transition to the beginning of the program (step 0). The main differences from the END instruction are the next:

— Processing does not end with the FEND command. The instruction FEND separates the main program from subprograms and interruption handlers, which are located in the area between the FEND and END instructions and are framed by P and SRET, I and IRET.

— If subprograms and interruptions are not used in a user program, the instruction FEND is not required.

— The instruction FEND can be used only once.

Use:



FEND
END

Instruction	Function
NOP	Empty line in the program

Description:

An empty line without logical functions can later be used for any instructions, for example, during assembling of a program or for debugging.

- After successful assembling a program, NOP instructions should be deleted, otherwise they uselessly extend the time of program cycle.
- The number of NOP instructions in a program is not limited.

Use:

```
LD   X0
NOP
OUT  Y0
```

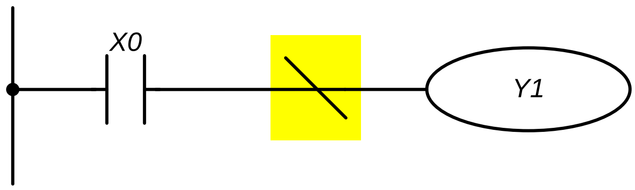
NOP instructions are not displayed in contact diagrams.

Instruction	Function
INV	Inversion - replacing the result of logical connections with the opposite

Description:

- The instruction INV inverts the state of the result signal of the placed before instructions.
- The result of logical connections “1” before INV instruction turns to “0” after it.
- The result of logical connections “0” before INV instruction turns to “1” after it
- The INV instruction can be applied as AND or ANI instructions.
- The INV instruction can be used to reverse the result signal of a complex circuit.
- The INV instruction can be used to reverse the signal result of the pulse instructions LDP, LDF, ANP, etc.

Use:




```
LD   X0
INV
OUT  Y1
```

If the input X0 = 0, the output Y1 = 1. If the input X0 = 1, the output Y1 = 0.

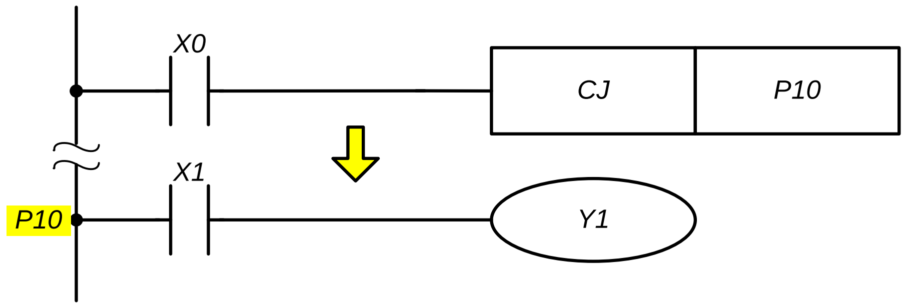
Instruction	Function
P	Addressing a jump point in a program or subprogram

Operand	0…31

Description:

- The P instruction is used to indicate a transition point for instructions CJ, CALL.
- The point number in the program should not be repeated.

Use:




```
LD   X0
CJ   P10
.
.
P    10
LD   X1
OUT  Y1
```

The point P10 indecates the transition address for executing of the instruction CJ P10.

Instruction	Function
I	Addressing of an interruption point

Operand	0…100, 1000…1007, 2000, 2001

Description:

The instruction I is used to indicate the transition point to the interruption handler. Globally interruptions are enabled by the instruction EN and disabled by the instruction DS.

Totally the controller can have 15 interruptions.

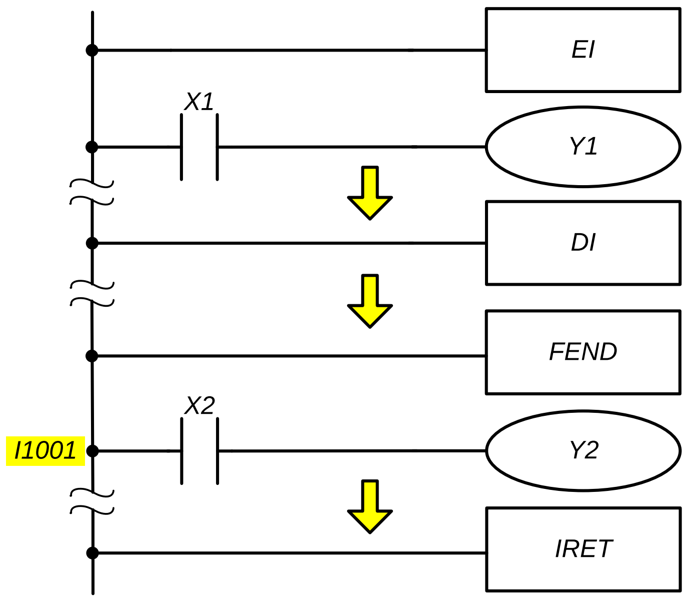
An interruption that occurs when a Modbus (broadcast or addressed to the controller) frame is received through RS-485 is marked as I0

It is possible to organize up to 4 timed interruptions in the controller In, where n is the interruption handler call period of 10ms and can have a value from 1 to 100. So, $n = \frac{T}{10}, ms$, where T is a desired period of call of the handler (measured as ms).

8 external interruptions **I1000**…**I1007** correspond to discrete inputs 0..7. The interruption arises when the level of the input signal is changed.

2 driver interruptions: **I2000** – arises in case of error and **I2001** – arises when the motor status changes (refer to the section 8. “Stepper motor driver control” for more details).

Use:



EI		Interruptions enabling
LD	X1	NO contact X1
OUT	Y1	Output Y1
…		
DI		Interruptions disabling
…		
FEND		End of main program
I	**1001**	Entry point for the interruption handler.
LD	X2	NO contact X2
OUT	Y2	Output Y2
…		
IRET		End of the interruption handler

7. Application instructions

Instruction	Operands	Associated variants	Function
CJ	S	P	Conditional jump

S	Pointers P are used as operands. The operands can be indexed (A, B)

Description:

Using the CJ instruction, a part of the program can be skipped. When applying this instruction, the execution time of the program can be reduced. For example, skipping a section of the program allocated for initializing the peripherals of the controller, turning on interruptions, etc. (refer to the section 4.8 for more details).

CALL	S	P	Calling subprogram

S	Pointers P are used as operands. The operands can be indexed (A, B)

Description:

The instruction CALL is used to call a subprogram.

- A subprogram is marked with points P and can be called by a CALL-instruction.
- The SRET instruction must be placed in the end of subprogram.
- A subprogram must be placed after the instruction FEND and before the instruction END.
- When CALL-instruction is being executed, the controller goes to the marked point. After executing of SRET-instruction, the controller returns to the main program to the instruction, which follows CALL.
- The points can be used with unlimited number of CALL instructions.
- Subprograms can be called from other subprogram. Maximum 8 nesting levels possible.

SRET			End of subprogram

Description:

The instruction SRET defines the end of a subprogram (refer to the section 4.8 for more details).

- Every subprogram must be finished with the SRET instruction.
- The program returns to the instruction following the CALL instruction after processing the SRET.
- The instruction SRET can be used together with the CALL instruction only.

Note: this instruction doesn’t require entry condition (contacts are not needed).

IRET			End of interruption handler

Description:

The instruction IRET defines the end of interruption processing (refer to the section 4.8 for more details).

Note: this instruction doesn’t require entry condition (contacts are not needed).

EI			Global interruptions enabling

Description:
The instruction EI enables interruption processing (refer to the section 4.8 for more details).
Note: this instruction doesn't require entry condition (contacts are not needed).

DI			Global interruptions disabling

Description:
The instruction DI disables interruption processing (refer to the section 4.8 for more details).
Note: this instruction doesn't require entry condition (contacts are not needed).

Calling of an interruption handler subprogram

— When processing an interruption, a transition is made from the main program to the interruption handler.
— After the interruption processing is completed, the controller returns to the main program.
— The start of the interruption subprogram is determined by setting the marking (interruption point).
— The end of the interruption subprogram is determined by the IRET instruction.
— The interruption subprogram must be programmed at the end of the user program after the FEND instruction and before the END instruction.

Note: If neither of the two EI or DI instructions is programmed, the interrupionst mode is not activated, i.e. no one of interruption signal will be processed.

Executing of an interruption subprogram

Several interruption subprograms going one after another are processed in the sequence of their calling.

If several interruption subprograms are called at the same time, the interruption program with a lower point address is processed first.

FOR	S		Start of a loop FOR-NEXT

	K	H	F	X	Y	M	T	C	A	B	D
S	●	●					●	●	●	●	●

Note: this instruction doesn't require entry condition (contacts are not needed).

NEXT			End of a loop FOR-NEXT

Note: this instruction doesn't require entry condition (contacts are not needed).

Cycles

The instructions FOR/NEXT are used for programming of cyclic repetitions of program parts (program loop).

Description:

- The part of the program between FOR- and NEXT instructions is repeated "n" times. After completing the FOR instruction, the program proceeds to the program step after the NEXT instruction.
- The value "n" may be in range from +1 to +32 767. If some value from the range from 0 to -32 768 is set, the loop FOR-NEXT is executed only once.
- Up to 8 nesting levels of FOR-NEXT loops are possible.
- The instructions FOR and NEXT can be used in pair only. Every FOR instruction must be matched with NEXT instruction.

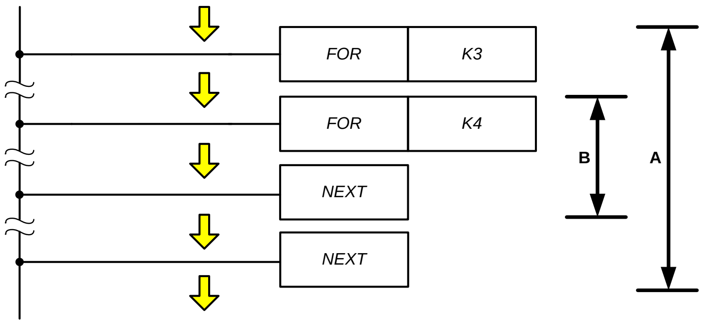
Source of errors

Errors appear in the program in the following cases:

- NEXT instruction goes before FOR instruction.
- The number of NEXT instructions differs from the number of FOR instructions.
- A large number of repetitions "n" can significantly increase a program executing time.

An example of using FOR/NEXT instructions:

The example below shows two FOR-NEXT loops one inside another.
The part of program A is executed 3 times (K3 means decimal number 3).
The part of program B is executed 4 times inside every of repetition of part A (K4 means decimal number 4).



CMP	S1 S2 D	D P	Comparison of numerical data

	K	H	F	X	Y	M	T	C	A	B	D
S1	•	•					•	•	•	•	•
S2	•	•					•	•	•	•	•
D					•	•					

Note: operand D takes 3 addresses.

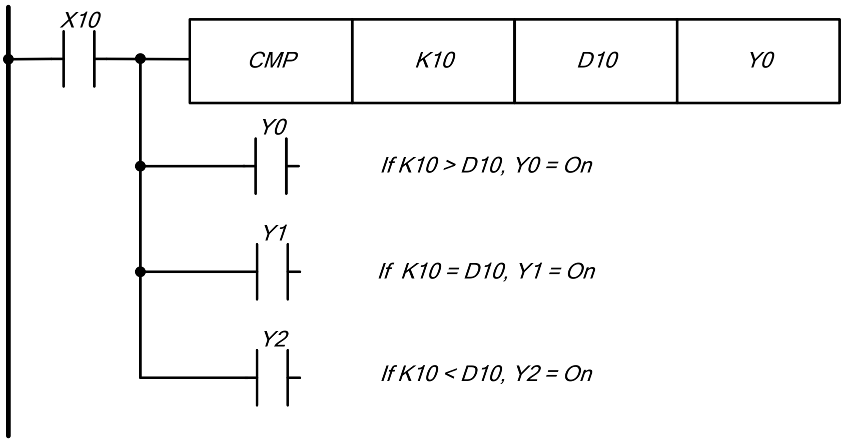
Description:
Comparison of two numerical data values (more, less, equal).

- Data in both sources (S1) and (S2) are compared.

— The result of comparison (more, less, equal) is displayed (indicated) by activating relay M or output Y. Which of the contacts at destination operand (D) is active, is determined by the comparison result:
(S1) > (S2) → (D)
(S1) = (S2) → (D+1)
(S1) < (S2) → (D+2)

— Data in S1 and S2 are processed as signed integer data.

Example:



Y0: is turned on if K10 > data register D10, Y1 and Y2 are turned off.
Y1: is turned on if K10 = data register D10, Y0 and Y2 are turned off.
Y2: is turned on if K10 < data register D10, Y0 and Y1 are turned off.
Y0, Y1, Y2 are not changed if the entry condition X10=0.

To reset the comparison results use instructions RST, ZRST.

ZCP	S1 S2 S S D	D P	Zone comparison of numerical data

	K	H	F	X	Y	M	T	C	A	B	D
S1	●	●					●	●	●	●	●
S2	●	●					●	●	●	●	●
S	●	●					●	●	●	●	●
D					●	●					

Note:

— Operand D takes 3 addresses.

— Operand S1 must be less than S2.

Description:

— Comparison of numerical data values with numerical data areas (more, less, equal)

— Data in source (S) is compared with data in both sources (S1) and (S2)

— The result of comparison (more, less, equal) is displayed (indicated) by activating relay M or output Y. Which of the contacts at destination operand (D) is active, is determined by the comparison result.
(S) < (S1) → (D)
(S1) ≤ (S) ≤ (S2) → (D + 1)

(S) > (S2) → (D + 2)

— If value in (S1) more than value in (S2), all contact in operand (D) are reset.
To reset the comparison results use instructions RST, ZRST.

MOV	S D	D P	Data transfer

	K	H	F	X	Y	M	T	C	A	B	D
S	●	●	●	●	●	●	●	●	●	●	●
D					●	●	●	●	●	●	●

Description:

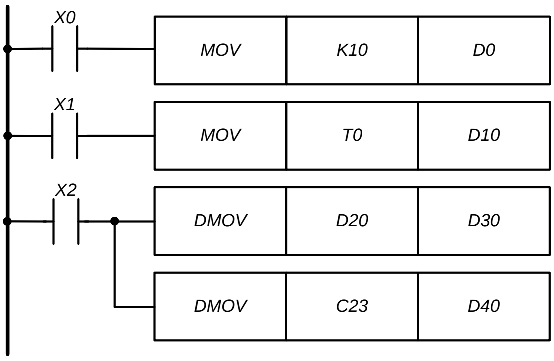
- The MOV instruction is used to transfer data from a data source (S) to a destination (D). The value of the source (S) does not change.
- Data in the data source (S) is read as binary values when executing the MOV instruction.
- Bit operands occupy the number of addresses corresponding to the instruction type - 16 or 32 addresses. In this case it is possible to combine the types of operands for the source and destination. For example, as a result of executing the MOV D3 M0 command, the relays M0 ... M15 display the value of register D3 in binary form.

Example:

If the entry condition X0 is turned on, the value of the register D0 = 10. If X0 is turned off, the value of the register D0 is not changed.

If the entry condition X1 is turned on, the current value of the timer T0 is transferred to the data register D10. If X1 is turned off, the value of the register D10 is not changed.

If the entry condition X2 is turned on, the value of the registers D20 and D21 is transferred to the data registers D30 and D31; the current value of the counter C23 is transferred to the data registers D40 and D41.



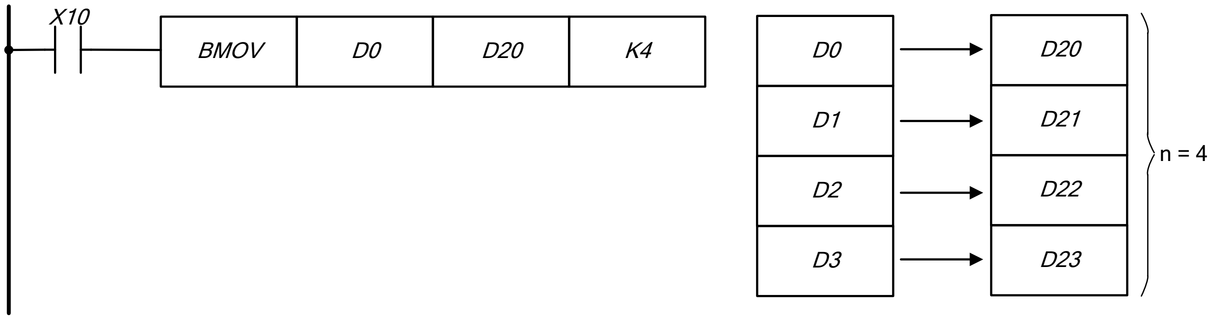
BMOV	S D n	D P	Block data transfer

	K	H	F	X	Y	M	T	C	A	B	D
S	●	●	●	●	●	●	●	●	●	●	●
D					●	●	●	●	●	●	●
n	●	●					●	●	●	●	●

Description:

Copy data packet. The shift during the operation is carried out both for the source operand (S) and for the destination operand (D) to (n) block elements, depending on the instruction (16 bit or 32 bit).

Example:



If X10 is turned on, the values of registers D0 – D3 is transferred to the registers D20 – D23.

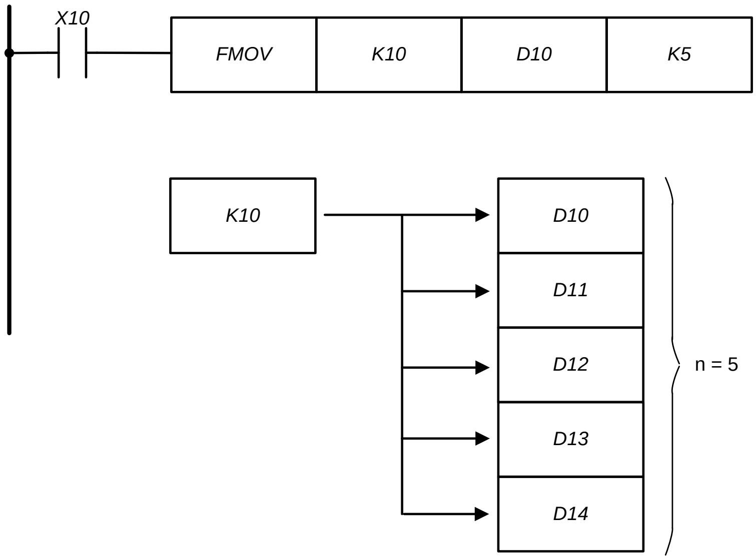
FMOV	S D n	D P	Transferring data to multiple addresses

	K	H	F	X	Y	M	T	C	A	B	D
S	●	●	●	●	●	●	●	●	●	●	●
D					●	●	●	●	●	●	●
n	●	●					●	●	●	●	●

Description:

The value of the source operand (S) are copied to (n) destination operands (D) of the same types.

Example:



The instruction FMOV copies value "10" to the data registers D10...D14.

XCH	D1 D2	D P	Data exchange

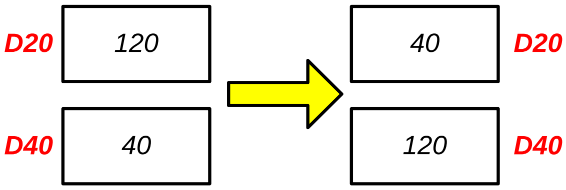
	K	H	F	X	Y	M	T	C	A	B	D
D1							●	●	●	●	●
D2							●	●	●	●	●

Description:

The values of operands (D1) and (D2) are swapped.

Example:

If X0 = 1 the data exchange is done:



ADD	S1 S2 D	D P	Addition of numerical data

	K	H	F	X	Y	M	T	C	A	B	D
S1	●	●					●	●	●	●	●
S2	●	●					●	●	●	●	●
D							●	●	●	●	●

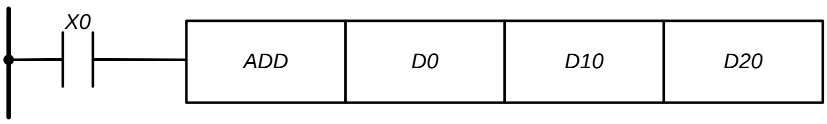
Description:

— Binary data in the source operands (S1) and (S2) are added together. The result of the addition is stored in the destination operand (D). The operation is performed on signed integer data types.
(S1) + (S2) = (D)

— The high bit contains the sign of the result: 0 – sign of a positive number, 1 – sign of a negative number.

— When executing a 32-bit instruction, the lower 16 bits should be indicated in the operand. The following data register contains higher 16 bits.

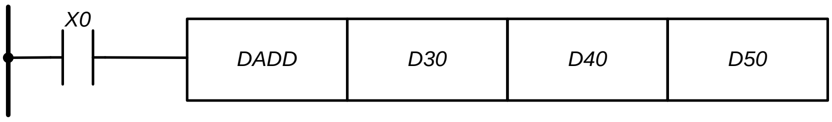
Examples:



(D0) + (D10) = (D20)

If X0 is turned on, the values of data registers D0 and D10 are added together, the result is saved in the data register D20.

Phone: + 372 6559914,
e-mail: mail@smd.ee
url: https://smd.ee



(D31, D30) + (D41, D40) = (D51, D50)

If X0 is turned on, the result of addition the values of registers (D31, D30) and (D41, D40) is saved in the data registers (D51, D50).

SUB	S1 S2 D	D P	Subtraction of numerical data

	K	H	F	X	Y	M	T	C	A	B	D
S1	●	●					●	●	●	●	●
S2	●	●					●	●	●	●	●
D							●	●	●	●	●

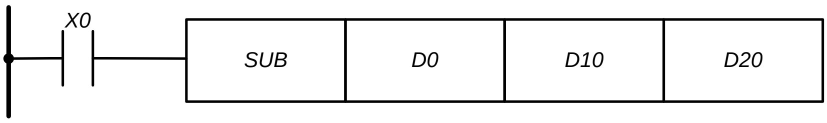
Description:

— The data value in (S2) is subtracted from the data value (S1). The result of the subtraction is stored in the destination operand (D). The operation is performed on signed integer data types.
(S1) – (S2) = (D)

— The high bit contains the sign of the result: 0 – sign of a positive number, 1 – sign of a negative number.

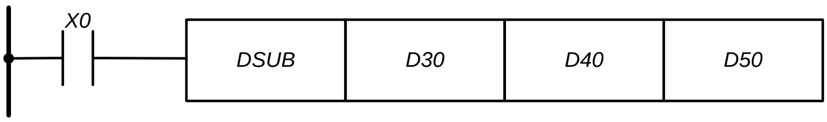
— When executing a 32-bit instruction, the lower 16 bits should be indicated in the operand. The following data register contains higher 16 bits.

Examples:



(D0) - (D10) = (D20)

If X0 is turned on, the difference between the data values in the registers D0 and D10 is calculated. The result is saved in the data register D20.



(D31, D30) - (D41, D40) = (D51, D50)

If X0 is turned on, the difference between the data values in the registers (D31, D30) and (D41, D40) is calculated. The result is saved in the data registers (D51, D50).

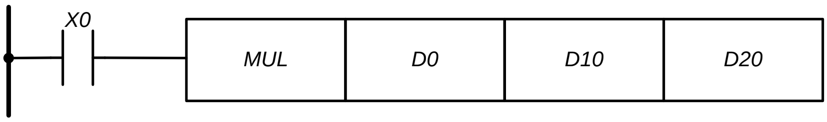
MUL	S1 S2 D	D P	Multiplication of numerical data

	K	H	F	X	Y	M	T	C	A	B	D
S1	●	●					●	●	●	●	●
S2	●	●					●	●	●	●	●
D							●	●	●	●	●

Description:

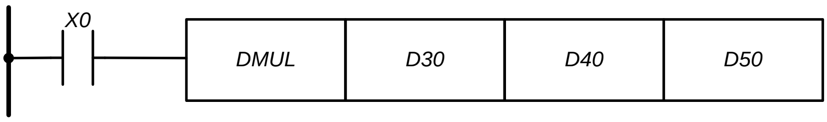
- The data in operands (S1) and (S2) are multiplied together. The result is stored in the destination operand (D). The operation is performed on signed integer data types.
  (S1) x (S2) = (D)
- The high bit contains the sign of the result: 0 – sign of a positive number, 1 – sign of a negative number.
- When executing a 32-bit instruction, the lower 16 bits should be indicated in the operand. The following data register contains higher 16 bits.

Examples:



(D0) * (D10) = (D20)

If X0 is turned on, the values in data registers D0 and D10 are multiplied together. The result is saved in the data register D20.



(D31, D30) * (D41, D40) = (D51, D50)

If X0 is turned on, the values in registers (D31, D30) and (D41, D40) are multiplied together. The result is saved in the data registers (D51, D50).

DIV	S1 S2 D	D P	Division of numerical data

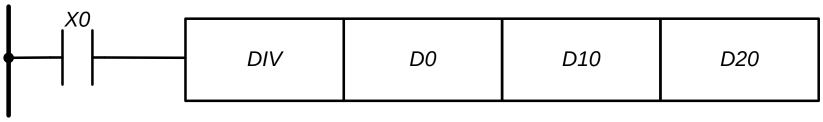
	K	H	F	X	Y	M	T	C	A	B	D
S1	●	●					●	●	●	●	●
S2	●	●					●	●	●	●	●
D							●	●	●	●	●

Description:

- The value of the source operand (S1) is divided by the data value from the source operand (S2). The whole part of the division result is stored in the destination operand (D). The operation is performed on signed integer data types.
  (S1) / (S2) = (D)
- The high bit contains the sign of the result: 0 – sign of a positive number, 1 – sign of a negative number.

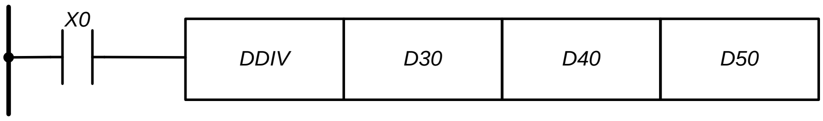
— When executing a 32-bit instruction, the lower 16 bits should be indicated in the operand. The following data register contains higher 16 bits.
— Division by zero leads to an error.

Examples:



(D0) / (D10) = (D20)

If X0 is turned on, division of data values in registers D0 and D10 is done. The result is saved in the data register D20.



(D31, D30) / (D41, D40) = (D51, D50)

If X0 is turned on, division of data values in registers (D31, D30) and (D41, D40) is done. The result is saved in the data registers (D51, D50).

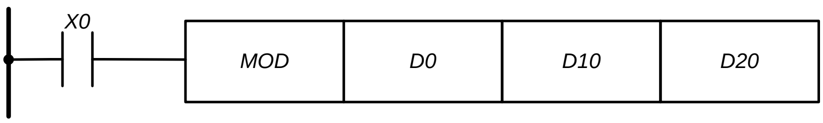
MOD	S1 S2 D	D P	Calculation of the remainder of the division

	K	H	F	X	Y	M	T	C	A	B	D
S1	●	●					●	●	●	●	●
S2	●	●					●	●	●	●	●
D							●	●	●	●	●

Description:

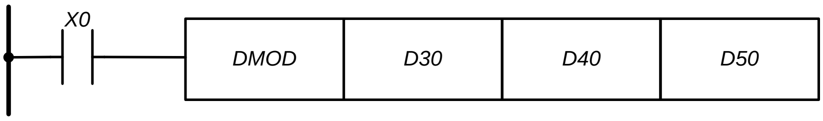
— The value of the source operand (S1) is divided by the data value from the source operand (S2). The remainder of the division is stored in the destination operand (D). The operation is performed on signed integer data types.
(S1) % (S2) = (D)
— The high bit contains the sign of the result: 0 – sign of a positive number, 1 – sign of a negative number.
— When executing a 32-bit instruction, the lower 16 bits should be indicated in the operand. The following data register contains higher 16 bits.
— Division by zero leads to an error

Examples:



(D0) % (D10) = (D20)

If X0 is turned on, division of data values in registers D0 and D10 is done. The result (remainder in division) is saved in the data register D20.



(D31, D30) % (D41, D40) = (D51, D50)

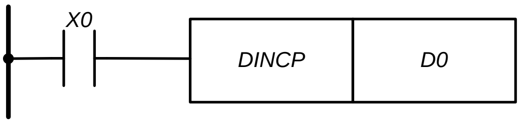
If X0 is turned on, division of data values in registers (D31, D30) and (D41, D40) is done. The result (remainder in division) is saved in the data registers (D51, D50).

INC	D	D P	Increment numerical data

	K	H	F	X	Y	M	T	C	A	B	D
D							●	●	●	●	●

Description:
The value in the operand (D) is incremented by 1.

Example:



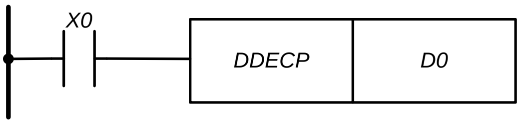
The value in the data registers (D1, D0) is incremented by 1 if the entry condition X0 is turned on. The instruction is activated due to the connected pulse function so that the summing process is not performed in each program cycle.

DEC	D	D P	Decrement numerical data

	K	H	F	X	Y	M	T	C	A	B	D
D							●	●	●	●	●

Description:
The value in the operand (D) is decremented by 1.

Example:



The value in the data registers (D1, D0) is decremented by 1 if the entry condition X0 is turned on. The instruction is activated due to the connected pulse function so that the decrementing is not performed in each program cycle.

WAND	S1 S2 D	D P	Logical multiplication of numerical data (operation "AND")

	K	H	F	X	Y	M	T	C	A	B	D
S1	●	●					●	●	●	●	●
S2	●	●					●	●	●	●	●
D							●	●	●	●	●

Tallinn Science Park Tehnopol,
Akadeemia tee 21/6, 12618, Tallinn, Estonia
Phone: + 372 6559914,
e-mail: mail@smd.ee
url: https://smd.ee

Note: WAND is a 16 bit instruction, **DAND** is a 32 bit instruction.

Description:

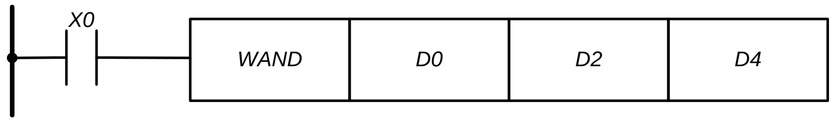
— The operation "logical AND" for numeric data is a bit operation (performed bit by bit).

— The values from the source operands (S1) and (S2) are multiplied bit by bit. The result is stored in the destination operand (D).

— The truth table of logical multiplication:

(S1)	(S2)	(D)
1	1	1
1	0	0
0	1	0
0	0	0

Example:



If X0 = 1, the logic multiplication of values from data registers D0 and D2 is done. The result is saved in the data register D4.

		b15															b00
(S1)	D0	1	1	1	1	1	0	0	0	0	0	1	1	0	0	0	1
									WAND								
(S2)	D2	0	0	1	1	0	1	1	0	0	1	1	0	1	1	0	0
(D)	D4	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0

WOR	S1 S2 D	D P	Logical addition of numerical data (OR operation)

	K	H	F	X	Y	M	T	C	A	B	D
S1	•	•					•	•	•	•	•
S2	•	•					•	•	•	•	•
D							•	•	•	•	•

Note: WOR is a 16 bit instruction, **DOR** is a 32 bit instruction.

Description:

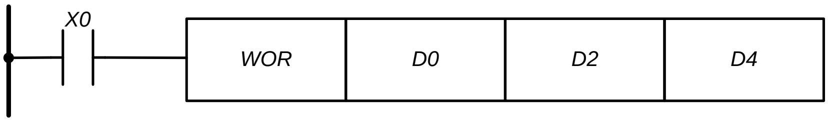
— The operation "logical OR" for numeric data is a bit operation (performed bit by bit).

— The values from the source operands (S1) and (S2) are added bit by bit. The result is stored in the destination operand (D).

— The truth table of logical addition:

(S1)	(S2)	(D)
1	1	1
1	0	1
0	1	1
0	0	0

Example:



If X0 = 1, the logical addition of values from data registers D0 and D2 is done. The result is saved in the data register D4.

		b15															b00
(S1)	D0	1	1	1	1	1	0	0	0	0	0	1	1	0	0	0	1
									WOR								
(S2)	D2	0	0	1	1	0	1	1	0	0	1	1	0	1	1	0	0
(D)	D4	1	1	1	1	1	1	1	0	0	1	1	1	1	1	0	1

WXOR	S1 S2 D	D P	Logical operation "exclusive OR"

	K	H	F	X	Y	M	T	C	A	B	D
S1	•	•					•	•	•	•	•
S2	•	•					•	•	•	•	•
D							•	•	•	•	•

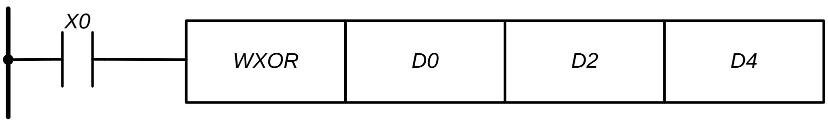
Note: WXOR is a 16 bit instruction, **DXOR** is a 32 bit instruction.

Description:

- The operation "logical exclusive OR" for numeric data is a bit operation (performed bit by bit)
- The values from the source operands (S1) and (S2) are processed bit by bit. The result is stored in the destination operand (D).
- The truth table of logical exclusive OR:

(S1)	(S2)	(D)
1	1	0
1	0	1
0	1	1
0	0	0

Example:



If X0 = 1, the operation "exclusive OR" is performed with values in data registers D0 and D2. The result of the operation is saved in the data register D4.

		b15															b00
(S1)	D0	1	1	1	1	1	0	0	0	0	0	1	1	0	0	0	1
										WXOR							
(S2)	D2	0	0	1	1	0	1	1	0	0	1	1	0	1	1	0	0
(D)	D4	1	1	0	0	1	1	1	0	0	1	0	1	1	1	0	1

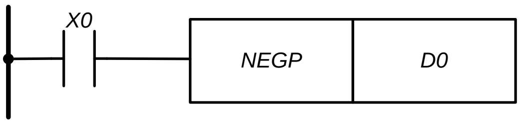
NEG	D	D P	Logical negation

	K	H	F	X	Y	M	T	C	A	B	D
D							●	●	●	●	●

Description:

Logical negation operation (inversion of all bits in binary form and addition with 1) for numerical data.

Example:



If X0 = 1, the operation of logical negation and modification is performed with value in the operand D0.

13931 → -13931 + 1 = -13931

		b15															b00
(D)	D0	0	0	1	1	0	1	1	0	0	1	1	0	1	0	1	1
(D)	D0	1	1	0	0	1	0	0	1	1	0	0	1	0	1	0	1

-13931 → 13930 + 1 = 13931

		b15															b00
(D)	D0	1	1	0	0	1	0	0	1	1	0	0	1	0	1	0	1
(D)	D0	0	0	1	1	0	1	1	0	0	1	1	0	1	0	1	1

ROL	D n	D P	Cycle shift to the left

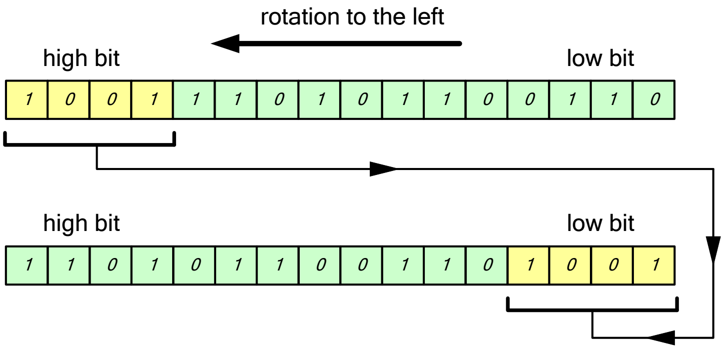
	K	H	F	X	Y	M	T	C	A	B	D
D							●	●	●	●	●
n	●	●					●	●	●	●	●

Description:

Bits rotation on (n) places to the left.

Example:

If X0 = 1, the bits of the value in the data register D10 rotate on 4 bits to the left and the value is modified.



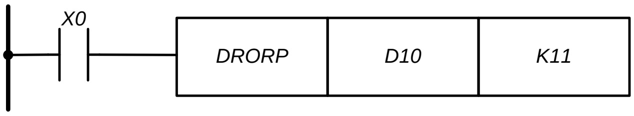
ROR	D n	P	Cycle shift to the right

	K	H	F	X	Y	M	T	C	A	B	D
D							●	●	●	●	●
n	●	●					●	●	●	●	●

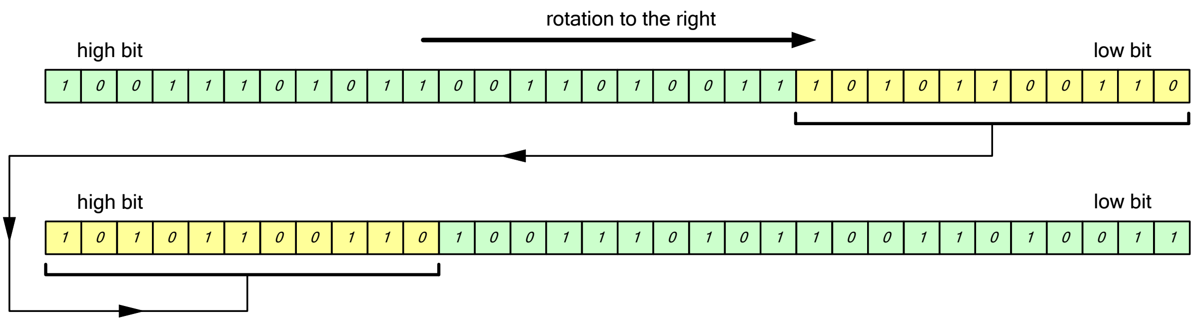
Description:

Bits rotation on (n) places to the right.

Example:



If X0 = 1, the bits of the value in the data register D10 rotate on 11 bits to the right and the value is modified.



ZRST	D1 D2	D P	Group reset of operands

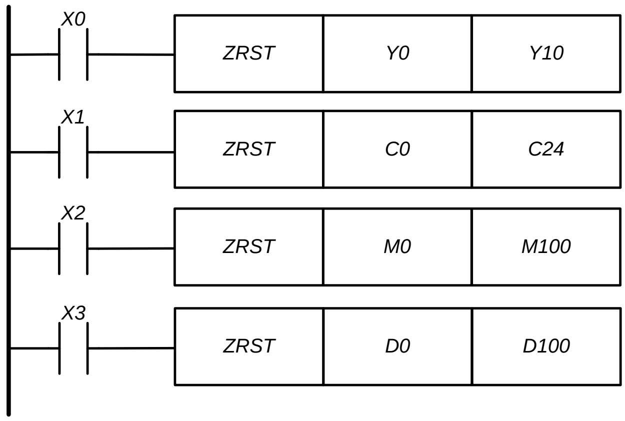
	K	H	F	X	Y	M	T	C	A	B	D
D1					●	●	●	●	●	●	●
D2					●	●	●	●	●	●	●

Description:

The values of several operands following one after another (operand region) can be reset by the instruction ZRST. Bit contacts are turned of, the registers are set to a value of "0".

— Operands (D1) and (D2) determine the region to be reset.
— The operands in (D1) and (D2) must be of the same type.
— (D1) – the first address of the region, (D2) – the last address of the region.
— (D1) must be less than (D2).

Example:



When the entry conditions are satisfied, the bit operands Y0 ... Y10, M0 ... M100 are turned off (turns to the state "0"). The numeric operands C0 ... C24, D0 ... D100 are turned to actual value "0". The corresponding coils and contacts are switched off.

DECO	S D n	P	Decoder 8 → 256 bit

	K	H	F	X	Y	M	T	C	A	B	D
S	●	●		●	●	●	●	●	●	●	●
D					●	●	●	●	●	●	●
n	●	●					●	●	●	●	●

Note: If (D) is a bit operand: (n) = 1…8. If (D) is a numeric operand: (n) = 1…4. If (n) is out of possible range, the instruction is executed with the maximum possible (n) depending on (D).

Description:

Decoding data. Data in (n) operands is decoded starting from the start address, which is specified in (S). Operand (D) determines the starting address of the destination (where the result of decryption is written).

(n) is the number of operands whose data should be decoded. When specifying the bit operand in (D), the following must be observed: $1 \leq (n) \leq 8$. When specifying the numeric operand in (D), the following must be observed: $1 \leq (n) \leq 4$.

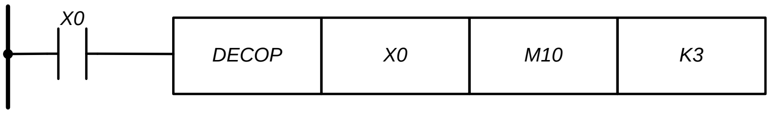
(S) is a start address of operands whose data should be decoded.
2^n – number of operands to be decoded.
(D) is a start address the destination operand.

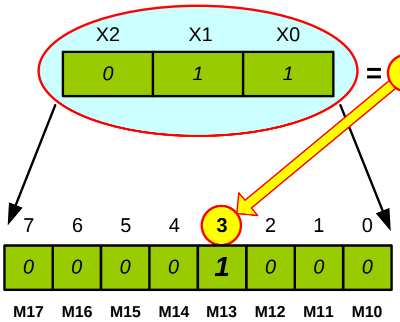
Attention! The instruction does not execute if (n) = 0.

Accordingly, the output remains active if the input conditions at the end of the action turn off again.

Example:

Use of a DECO instruction with bit operands in (D).

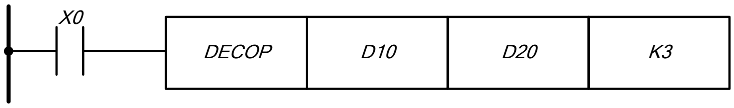


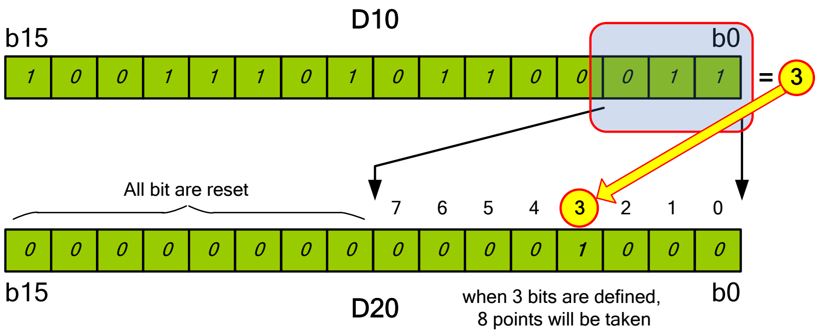


If (n) = 3, the input operands X0, X1 and X2 are processed. $2^n = 2^3 = 8$ addresses are used as a destionation M10…M17.

Value of input operands is 1 + 2 = 3. So the 3d address of the destination, i.e. relay M13 is turned on. If the value of the input operand "0" is processed, then the relay M10 is activated.

Use of a DECO instruction with numeric operands in (D).





The lower 3 bits of the D10 data register are decoded. The decoding result 1 + 2 = 3 is transferred to the data register D20. The 3rd bit is turned on in this data register.

If the value for (n) <3, then all unnecessary bits of a higher number in the destination addresses are set to zero.

ENCO	S D n	P	Encoder 256 → 8 bit

	K	H	F	X	Y	M	T	C	A	B	D
S	●	●		●	●	●	●	●	●	●	●
D					●	●	●	●	●	●	●
n	●	●					●	●	●	●	●

Note: If (D) is a bit operand: (n) = 1…8. If (D) is a numeric operand: (n) = 1…4. If (n) is out of possible range, the instruction is executed with the maximum possible (n) depending on (D).

Description:

Encoding data. Data in 2^n operands is encoded starting from the start address, which is specified in (S). Operand (D) determines the starting address of the destination (where the result of decryption is written).

2^n is the number of operands whose data should be encoded.

(n) – the number of destination operands.

When specifying the bit operand in (S), the following must be observed: $1 \leq (n) \leq 8$. When specifying the numeric operand in (S), the following must be observed: $1 \leq (n) \leq 4$.

(S) is a start address of operands whose data should be encoded.

(D) is a start address the destination operand.

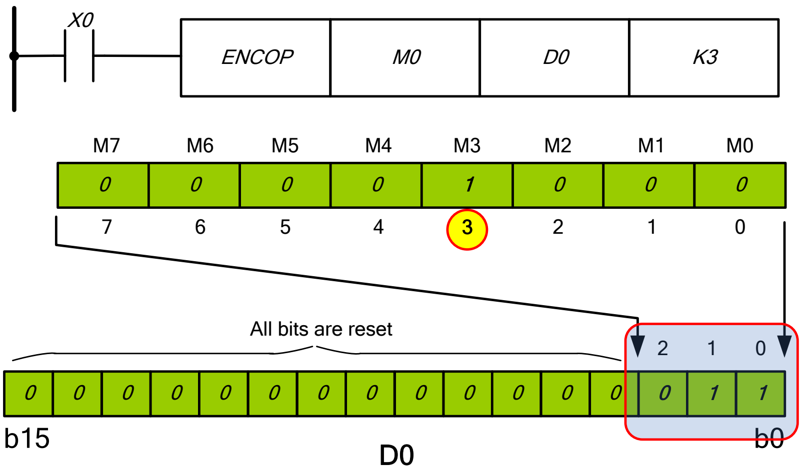
If several operands specified in (S) have the value “1”, then only the low bit is processed.

Attention! The instruction does not execute if (n) = 0.

Accordingly, the output remains active if the input conditions at the end of the action turn off again.

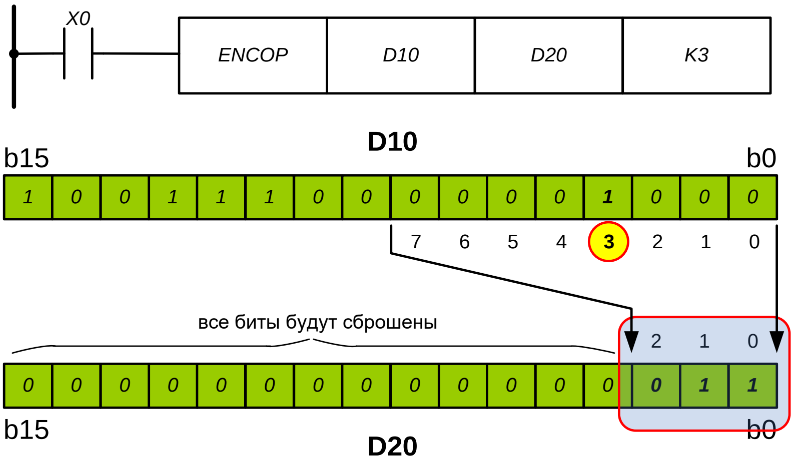
Example:

Use of a ENCO instruction with bit operands in (S).



If 2n = 23 = 8 then output relay addresses are M0...M7. As the 3d output is turned on (i.e. M3 = 1), the value 3 is written to the data register D0.

Use of a ENCO instruction with numeric operands in (S).



The 3d bit is on in the data register D10. So the value 3 is encoded and saved in the data register D20.

SUM	S D	D P	Sum of single bits

	K	H	F	X	Y	M	T	C	A	B	D
S	●	●					●	●	●	●	●
D							●	●	●	●	●

Description:

- Determining the number of active bits in a data word. The instruction counts turned on bits in (S).
- The result value is written to (D).

If a 32-bit instruction is processed, then the high 16 bits (D + 1) of the destination operands (D) are set to zero, since the maximum number of turned on bits in (S) is 32.

BON	S D n	D P	Check bit state

	K	H	F	X	Y	M	T	C	A	B	D
S	●	●					●	●	●	●	●
D					●	●					
n	●	●					●	●	●	●	●

Note: The necessary condition: (n) = 0…15 (16 bit), (n) = 0…31 (32 bit).

Description:

A single bit is checked inside the data word. If the bit (n) is turned on in (S), then the corresponding bit in (D) is turned on.

SQR	S D	D P	Square root calculation

	K	H	F	X	Y	M	T	C	A	B	D
S	●	●					●	●	●	●	●
D											●

Description:

Square root calculation $(D) = \sqrt{(S)}$

The square root of the value in the operand (S) is being calculated, the result is be rounded off to an integer and written to the operand (D). The operation is performed on signed integer data types.

Attention! The square root of a negative number always leads to an error.

FLT	S D	D P	Convert integer to floating point

	K	H	F	X	Y	M	T	C	A	B	D
S	●	●					●	●	●	●	●
D											●

Description:

The instruction FLT converts integer signed number to floating point format.

- The integer in the operand (S) is converted to a floating point number. The result is saved in the operand (D).
- The result of converting is always 32-bit number.

PWM	S1 S2 D		PWM pulse output

	K	H	F	X	Y	M	T	C	A	B	D
S1	●	●					●	●	●	●	●
S2	●	●					●	●	●	●	●
D					●						

Note: The value of the operand (S1) must be less than or equal to the value of the operand (S2).

Description:

(S1) – pulse width, t.

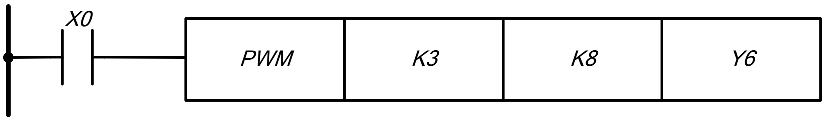
(S2) – period duration, T.

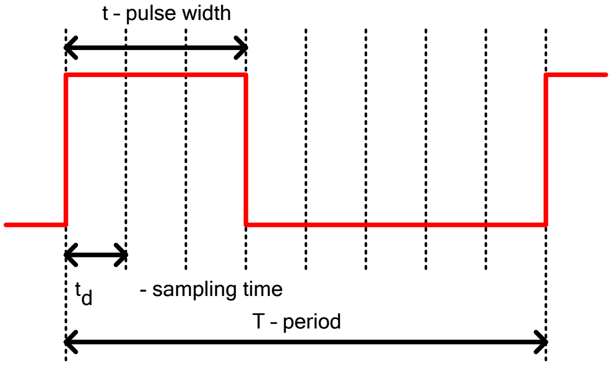
(D) – output address, Y6 or Y7.

Valid values for (S1) and (S2) are from 1 to 32767. (S1) and (S2) are the number of sampling intervals. The sampling period is set simultaneously for both channels 100 µs or 1 ms by the special register D356 (see the section 4.6 for more details).

A PWM signal is present at the output as long as the signal at the input of the PWM instruction is active.

Example:





Let the sampling period be 1 ms. When the input condition X0 = 1 is satisfied, the PWM signal with a period of 8 x 1 ms = 8 ms and a pulse duration of 3 x 1 ms = 3 ms appears at the output Y6.

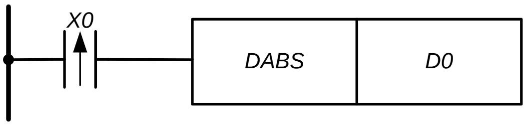
ABS	D	D P	Absolute value

	K	H	F	X	Y	M	T	C	A	B	D
D							●	●	●	●	●

Description:

The instruction ABS writes absolute value of a number into the operand (D). If the value in (D) is negative, then after executing the ABS instruction the sign "-" is discarded and the number becomes positive. If (D) has a positive value, then no changes occur.

Example:



When the input condition is satisfied, the module of the number in the register (D1, D0) is determined.

POW	S1 S2 D	D P	Raising to a power

	K	H	F	X	Y	M	T	C	A	B	D
S1	●	●					●	●	●	●	●
S2	●	●					●	●	●	●	●
D											●

Description:

Raising to a power: $(D) = (S1)^{(S2)}$.

The instruction POW rises the value in the operand (S1) to the (S2) power. The result is saved in the operand (D). The operation is performed on signed integer data types.

DECMP	S1 S2 D	D P	Comparison of floating point numbers

	K	H	F	X	Y	M	T	C	A	B	D
S1	●	●	●								●
S2	●	●	●								●
D					●	●					

Note:

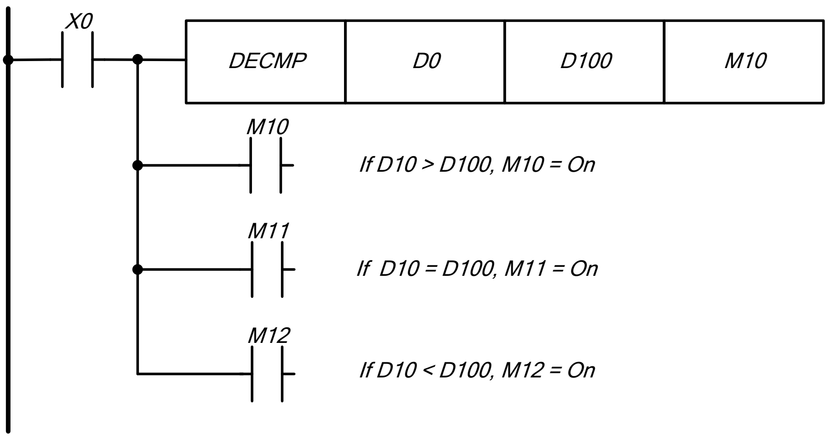
- 32-bit instruction only.
- The operand (D) takes 3 consecutive addresses.
- K and H types are not converted to F, but are projected onto a memory area. To convert integer data types to floating point data, use the **FLT** instruction**.**

Description:

The instruction DECMP performs comparison of two binary floating point numbers and outputs the result of the comparison.

- The DECMP instruction compares the floating point number in (S1) with the floating point number in (S2).
- The comparison result is stored in 3 successive addresses.
- If the number in (S2) is less than the number in (S1), then the bit operand (D) is turned on.
- If the number in (S2) is equal to the number in (S1), then the bit operand ((D) +1) is turned on.
- If the number in (S2) is greater than the number in (S1), then the bit operand ((D) +2) is turned on.
- The polled output operands remain turned on after disabling the entry conditions of the instruction DECMP.

Example:



When contact X0 is turned on, the floating point number specified in D100 (S2) is compared with the floating point number specified in D0 (S1). If the number in D100 is less than the number D0, then the relay M10 is activated. If the number in D100 is equal to the number D0, then the relay M11 is turned on. If the number in D100 is greater than the number D0, then relay M12 is activated.

To obtain the comparison results in the form: $\leq \geq \neq$, you can use parallel contact combinations M10 - M12. To reset the result, you can use the instructions RST, ZRST.

DEZCP	S1 S2 S S D	D P	Zone floating point comparison

	K	H	F	X	Y	M	T	C	A	B	D
S1	●	●	●								●
S2	●	●	●								●
S	●	●	●								●
D					●	●					

Note:

- 32-bit instruction only.
- The operand (D) takes 3 consecutive addresses.
- K and H types are not converted to F, but are projected onto a memory area. To convert integer data types to floating point data, use the **FLT** instruction**.**

Description:

Comparison of the floating point number with the selected (indicated) area and the output of the comparison result.

- The instruction DEZCP compares the floating point number in the operand (S1) with the area between (S1) and (S2).
- The comparison result is stored in 3 successive addresses.
- If the number in the operand (S) is less than the numbers in the operands (S1) and (S2), then the bit operand (D) is turned on.
-
- If the number in the operand (S) is equal to the numbers between (S1) and (S2), then the bit operand ((D) +1) is turned on.
- If the number in the operand (S) is greater than the numbers between (S1) and (S2), then the bit operand ((D) +2) is turned on
- The polled output operands remain on after the DEZCP instruction entry conditions are disabled.
- If the number in the operand (S1) is greater than the number in the operand (S2), then all bits in (D), (D+1), (D+2) will be reset.

DEADD	S1 S2 D	D P	Addition of floating point numbers

	K	H	F	X	Y	M	T	C	A	B	D
S1	●	●	●								●
S2	●	●	●								●
D											●

Note:

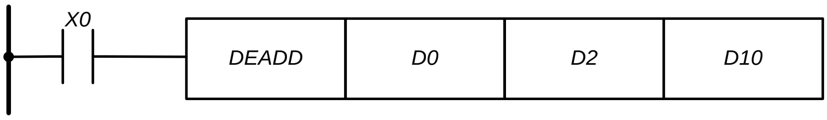
- 32-bit instruction only.
- K and H types are not converted to F, but are projected onto a memory area. To convert integer data types to floating point data, use the **FLT** instruction**.**

Description:

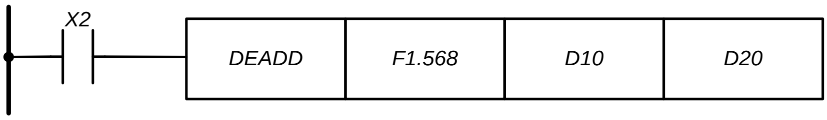
Calculating the sum of two numbers in binary floating point format.

- The floating point numbers specified in (S1) and (S2) are added together. The result of the addition is stored in the destination operand (D).
- Two consecutive registers are used for each operand.
- The same operand can be used for the source and for the destination. In this case, the calculated result is again stored in the source operand and can be used for the next calculation. This process is repeated in each program cycle.

Examples:



When the input X0 is turned on, the floating-point number in (D3, D2) is added to the floating-point number in (D1, D0). The result is saved in (D11, D10).



When the input X2 is turned on, the floating-point number in (D11, D10) is added to the constant F1.568. The result is saved in (D21, D20).

DESUB	S1 S2 D	D P	Subtraction of floating point numbers

	K	H	F	X	Y	M	T	C	A	B	D
S1	●	●	●								●
S2	●	●	●								●
D											●

Note:

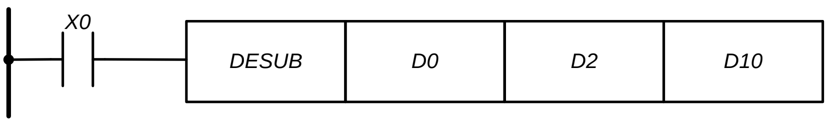
- 32-bit instruction only.
- K and H types are not converted to F, but are projected onto a memory area. To convert integer data types to floating point data, use the **FLT** instruction**.**

Description:

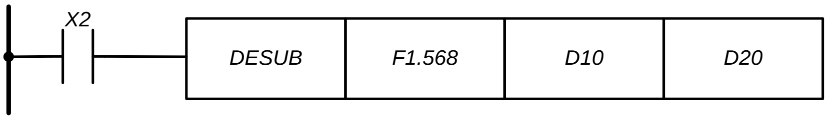
The instruction DESUB Computing the difference of two numbers in binary floating point format.

- The floating point number specified in (S2) is subtracted from the floating point number specified in (S1). The result is saved in (D).
- Two consecutive registers are used for each operand.
- The same operand can be used for the source and for the destination. In this case, the calculated result is again stored in the source operand and can be used for the next calculation. This process is repeated in each program cycle.

Examples:



When input X0 is turned on, the floating point number in (D3, D2) is subtracted from the floating point number in (D1, D0). The result is saved in (D11, D10).



When the input X2 is turned on, the floating-point number in (D11, D10) is subtracted from the constant F1.568. The result is saved in (D21, D20).

DEMUL	S1 S2 D	D P	Multiplication of floating point numbers

	K	H	F	X	Y	M	T	C	A	B	D
S1	●	●	●								●
S2	●	●	●								●
D											●

Note:

— 32-bit instruction only.

— K and H types are not converted to F, but are projected onto a memory area. To convert integer data types to floating point data, use the **FLT** instruction**.**

Description:

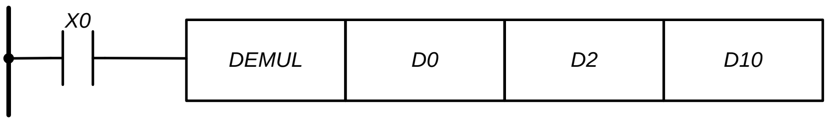
Multiplication of of two numbers in binary floating point format.

— The floating point number specified in the operand (S1) is multiplied by the floating point number in the operand (S2). The result is saved in the operand (D).

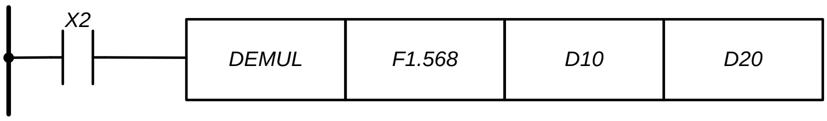
— Two consecutive registers are used for each operand.

— The same operand can be used for the source and for the destination. In this case, the calculated result is again stored in the source operand and can be used for the next calculation. This process is repeated in each program cycle.

Examples:



When input X0 is turned on, the floating point number in (D1, D0) is multiplied by the floating point number in (D3, D2). The result is saved in (D11, D10).



When input X2 is turned on, the constant F1.568 is multiplied by the floating point number in (D11, D10). The result is saved in (D21, D20).

DEDIV	S1 S2 D	D P	Floating point numbers division

	K	H	F	X	Y	M	T	C	A	B	D
S1	●	●	●								●
S2	●	●	●								●
D											●

Note:

— 32-bit instruction only.

— K and H types are not converted to F, but are projected onto a memory area. To convert integer data types to floating point data, use the **FLT** instruction.

Description:

Calculation of the quotient of dividing two numbers in binary floating-point format.

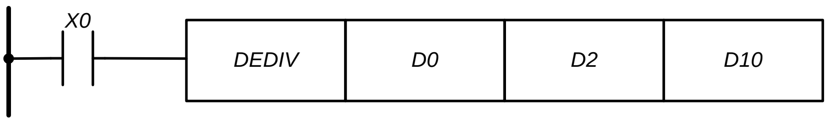
— The floating point number specified in the operand (S1) is divided by the floating point number specified in the operand (S2). The result is saved in (D).

— Two consecutive registers are used for each operand.

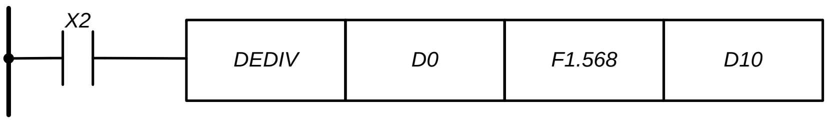
— The same operand can be used for the source and for the destination. In this case, the calculated result is again stored in the source operand and can be used for the next calculation. This process is repeated in each program cycle.

— The operand (S2) S2) cannot be zero because division by zero is not allowed.

Examples:



When the input X0 is turned on, the floating point number in (D1, D0) is divided by the floating point number in (D3, D2). The result is saved in (D11, D10).



When the input X2 is turned on, the floating-point number in (D1, D0) is divided by the constant F1.568. The result is saved in (D11, D10).

DESQR	S D	D P	Square root in floating point format

	K	H	F	X	Y	M	T	C	A	B	D
S	●	●	●								●
D											●

Note:

— 32-bit instruction only.

— K and H types are not converted to F, but are projected onto a memory area. To convert integer data types to floating point data, use the **FLT** instruction.

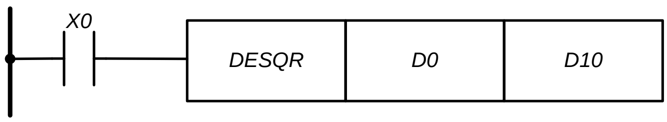
— Necessary condition: (S) ≥ 0

Description:

Calculating the square root of a binary floating-point number.

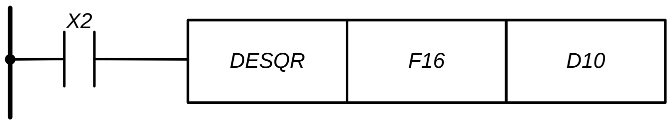
- The square root is calculated from the floating point number specified in the operand (S). The result is saved in (D).
- Two consecutive registers are used for each operand.
- The same operand can be used for the source and for the destination. In this case, the calculated result is again stored in the source operand and can be used for the next calculation. This process is repeated in each program cycle.

Examples:



$$\sqrt{(D1, D0)} \rightarrow (D11, D10)$$

When the input X0 is turned on, the square root of the floating point number in (D1, D0) is calculated. The result is saved in (D11, D10).



When the input X0 is turned on, the square root of the constant F16 is calculated. The result is saved in (D11, D10).

DEPOW	S1 S2 D	D P	Raising to a power in floating point format

	K	H	F	X	Y	M	T	C	A	B	D
S1	●	●	●								●
S2	●	●	●								●
D											●

Note:

- 32-bit instruction only.
- K and H types are not converted to F, but are projected onto a memory area. To convert integer data types to floating point data, use the **FLT** instruction**.**

Description:

Raising a number to a power in binary floating point format.

- The number specified in (S1) is raising to the (S2) power. The result is saved in the operand (D).
  $(S1)^{(S2)} = (D)$.
- Two consecutive registers are used for each operand.
- The same operand can be used for the source and for the destination. In this case, the calculated result is again stored in the source operand and can be used for the next calculation. This process is repeated in each program cycle.

Tallinn Science Park Tehnopol,
Akadeemia tee 21/6, 12618, Tallinn, Estonia
Phone: + 372 6559914,
e-mail: mail@smd.ee
url: https://smd.ee

INT	S D	D P	Converting a floating point number to an integer

	K	H	F	X	Y	M	T	C	A	B	D
S	·										●
D							●	●	●	●	●

Description:

The instruction INT converts floating point numbers to integers, rounded to the nearest.

— The floating point number specified in (S) is rounded to the nearest integer value and saved in (D).

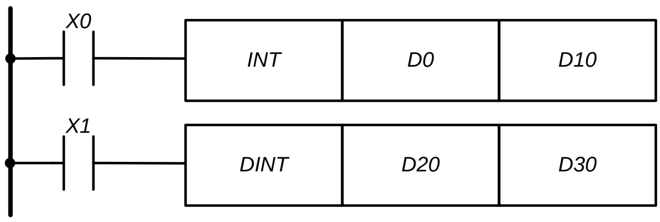
— The source operand is always a double word.

— When using the instruction INT, the word operand is the operand of the destination.

— When using the DINT instruction, the destination operand is a double word operand.

— The INT instruction is the inverse function of the FLT instruction.

Example:



When input X0 is turned on, the floating point number in (D0, D1) is rounded to the nearest lower integer value. The result is saved in D10.

When input X1 is turned on, the floating-point number in (D20, D21) is rounded to the nearest lower integer value. The result is saved in (D30, D31).

TRD	D	P	Reading time data

	K	H	F	X	Y	M	T	C	A	B	D
D											●

Note: The operand D takes 3 consecutive addresses.

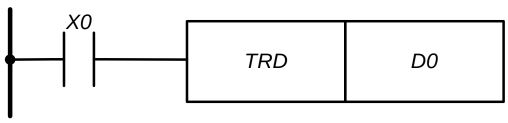
Description:

Reading the current value of the real-time clock.

— Using the TRD instruction, real-time data is read (hours, minutes, seconds).

— This data is saved in 3 consecutive operand addresses (D).

Example:



When input X0 is turned on, real-time data is read and saved in the registers D0 ... D2

Register	Function	Value	Example	
D0	Seconds	0…59	20	12:36:20
D1	Minutes	0…59	36	
D2	Hours	0…23	12	

TWR	S	P	Recording time data

	K	H	F	X	Y	M	T	C	A	B	D
S											●

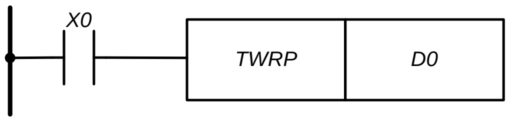
Note: The operand (S) takes 3 consecutive addresses.

Description:

The instruction TWR is used to change the real-time data (hours, minutes, seconds). The data is taken from 3 consecutive addresses, specified in (S).

If the values in (S) exceed the allowed range of values, an error arises.

Example:



When the entry condition is satisfied, the real-time clock of the controller is set to the values indicated in the registers D0 ... D2.

Register	Function	Value	Example	
D0	Seconds	0…59	42	03:11:42
D1	Minutes	0…59	11	
D2	Hours	0…23	3	

LD#	S1 S2	D	Contact type logical operations

	K	H	F	X	Y	M	T	C	A	B	D
S1	●	●		●	●	●	●	●	●	●	●
S2	●	●		●	●	●	●	●	●	●	●

Note:

— The symbol # is &, |, ^.

— Bit operands are taken by 16 or 32, depending on the type of instruction, and are converted to an integer data type for further processing.

Description:

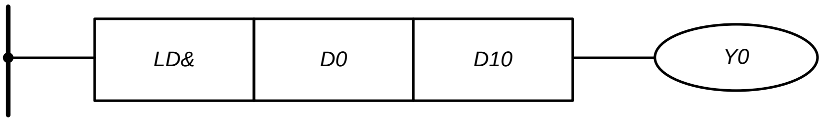
Performing the logical operation "AND", "OR", "EXCLUSIVE OR" on the operands (S1) and (S2), and turning on the LD-contact, depending on the result of the operation.

The instructions LD# in the program are located on the left and open a logical connection or are conditions for the execution of commands at right.

16-bit instructions	32-bit instructions	Contact closed if:	Contact open if:
LD&	**D**LD&	(S1) & (S2) ≠ 0	(S1) & (S2) = 0
LD|	**D**LD|	(S1) | (S2) ≠ 0	(S1) | (S2) = 0
LD^	**D**LD^	(S1) ^ (S2) ≠ 0	(S1) ^ (S2) = 0

&: logical multiplication (AND)
|: logical addition (OR)
^: exclusive OR (XOR)

Example:



AND#	S1 S2	D	Contact type logical operations Serial connection

	K	H	F	X	Y	M	T	C	A	B	D
S1	●	●		●	●	●	●	●	●	●	●
S2	●	●		●	●	●	●	●	●	●	●

Note:

— The symbol # is &, |, ^.
— Bit operands are taken by 16 or 32, depending on the type of instruction, and are converted to an integer data type for further processing.

Description:

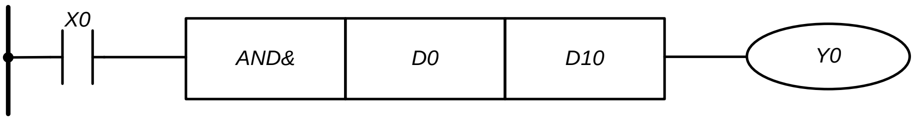
Performing the logical operation "AND", "OR", "Exclusive OR" on the operands (S1) and (S2), and turning on the AND-contact depending on the result of the operation.

The AND# instructions in the program are located after the LD commands and create a logical AND connection.

16-bit instructions	32-bit instructions	Contact closed if:	Contact open if:
AND&	**D**AND&	(S1) & (S2) ≠ 0	(S1) & (S2) = 0
AND|	**D**AND|	(S1) | (S2) ≠ 0	(S1) | (S2) = 0
AND^	**D**AND^	(S1) ^ (S2) ≠ 0	(S1) ^ (S2) = 0

&: logical multiplication (AND)
|: logical addition (OR)
^: exclusive OR (XOR)

Example:



OR#	S1 S2	D	Contact type logical operations Parallel connection

	K	H	F	X	Y	M	T	C	A	B	D
S1	●	●		●	●	●	●	●	●	●	●
S2	●	●		●	●	●	●	●	●	●	●

Note:

- The symbol # is &, |, ^.
- Bit operands are taken by 16 or 32, depending on the type of instruction, and are converted to an integer data type for further processing.

Description:

Performing the logical operation "AND", "OR", "Exclusive OR" on the operands (S1) and (S2), and turning on the OR-contact depending on the result of the operation.

The OR# instructions in the program are located at left in parallel to LD instruction and create a logical OR connection.

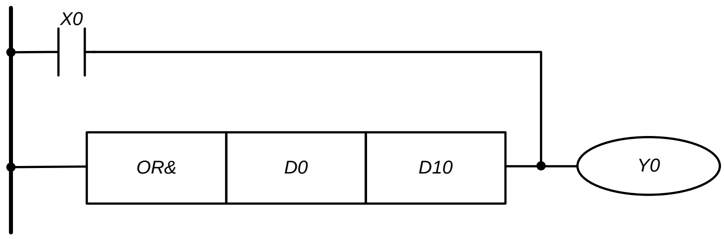
16-bit instructions	32-bit instructions	Contact closed if:	Contact open if:
OR&	**D**OR&	(S1) & (S2) ≠ 0	(S1) & (S2) = 0
OR|	**D**OR|	(S1) | (S2) ≠ 0	(S1) | (S2) = 0
OR^	**D**OR^	(S1) ^ (S2) ≠ 0	(S1) ^ (S2) = 0

&: logical multiplication (AND)

|: logical addition (OR)

^: exclusive OR (XOR)

Example:



LD*	S1 S2	D	Contact type comparison operations

	K	H	F	X	Y	M	T	C	A	B	D
S1	●	●		●	●	●	●	●	●	●	●
S2	●	●		●	●	●	●	●	●	●	●

Note:

- The symbol # is =, >, <, <>, ≤, ≥.
- Bit operands are taken by 16 or 32, depending on the type of instruction, and are converted to an integer data type for further processing.

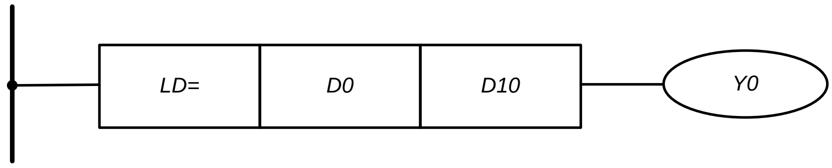
Description:

Comparison of the values of the operands (S1) and (S2), and turning on an LD contact, depending on the result of the operation.

- The LD * instructions in the program are located at the left and begin a logical connection or are conditions for the execution of instructions at right.
- If the comparison result is true, the LD contact is turned on.
- If the result of the comparison is false, the LD contact is turned off.

16-bit instructions	32-bit instructions	Contact closed if:	Contact open if:
LD=	**D**LD=	(S1) = (S2)	(S1) ≠ (S2)
LD>	**D**LD>	(S1) > (S2)	(S1) ≤ (S2)
LD<	**D**LD<	(S1) < (S2)	(S1) ≥ (S2)
LD<>	**D**LD<>	(S1) ≠ (S2)	(S1) = (S2)
LD<=	**D**LD<=	(S1) ≤ (S2)	(S1) > (S2)
LD>=	**D**LD>=	(S1) ≥ (S2)	(S1) < (S2)

Example:



AND*	S1 S2	D	Contact type comparison operations Serial connection

	K	H	F	X	Y	M	T	C	A	B	D
S1	●	●		●	●	●	●	●	●	●	●
S2	●	●		●	●	●	●	●	●	●	●

Note:

- The symbol **#** is =, >, <, <>, ≤, ≥.
- Bit operands are taken by 16 or 32, depending on the type of instruction, and are converted to an integer data type for further processing.

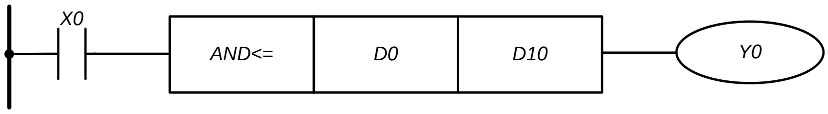
Description:

Comparison of the values of the operands S1 and S2, and turning on AND-contact, depending on the result of the operation.

- The AND* instructions in the program are located after the LD commands and create a logical AND connection..
- If the comparison result is true, the AND contact is turned on.
- If the result of the comparison is false, the AND contact is turned off.

16-bit instructions	32-bit instructions	Contact closed if:	Contact open if:
AND=	**D**AND=	(S1) = (S2)	(S1) ≠ (S2)
AND>	**D**AND>	(S1) > (S2)	(S1) ≤ (S2)
AND<	**D**AND<	(S1) < (S2)	(S1) ≥ (S2)
AND<>	**D**AND<>	(S1) ≠ (S2)	(S1) = (S2)
AND<=	**D**AND<=	(S1) ≤ (S2)	(S1) > (S2)
AND>=	**D**AND>=	(S1) ≥ (S2)	(S1) < (S2)

Example:



OR*	S1 S2	D	Contact type comparison operations Parallel connection

	K	H	F	X	Y	M	T	C	A	B	D
S1	●	●		●	●	●	●	●	●	●	●
S2	●	●		●	●	●	●	●	●	●	●

Note:

— The symbol **#** is =, >, <, <>, ≤, ≥.

— Bit operands are taken by 16 or 32, depending on the type of instruction, and are converted to an integer data type for further processing.

Description:

Comparison of the values of the operands S1 and S2, and turning on OR-contact, depending on the result of the operation.

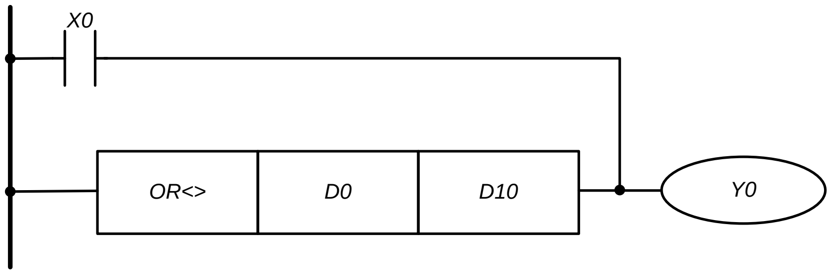
— The OR* instructions in the program are located at left in parallel to LD instruction and create a logical OR connection.

— If the comparison result is true, the OR contact is turned on.

— If the result of the comparison is false, the OR contact is turned off.

16-bit instructions	32-bit instructions	Contact closed if:	Contact open if:
OR=	**D**OR=	(S1) = (S2)	(S1) ≠ (S2)
OR>	**D**OR>	(S1) > (S2)	(S1) ≤ (S2)
OR<	**D**OR<	(S1) < (S2)	(S1) ≥ (S2)
OR<>	**D**OR<>	(S1) ≠ (S2)	(S1) = (S2)
OR<=	**D**OR<=	(S1) ≤ (S2)	(S1) > (S2)
OR>=	**D**OR>=	(S1) ≥ (S2)	(S1) < (S2)

Example:



8. Stepper motor driver control

The stepper motor driver is controlled by commands that specify the parameters of rotation or movement. All commandss are divided into two groups: **RUN** and **MOVE**. The RUN group is designed to control the current speed of the drive, and MOVE - to control movement. To start a rotation after selecting a command and setting driver parameters, the **SPIN** instruction is called.

SPIN		P	Perform specified movement
Address	**Object type**	**Description of the Modbus object**	
5100h	**Coils**	Writing “1” to this object (even if it is not reset) triggers a parameterized movement, resetting is ignored.	

Description:

The instruction starts a parameterized rotation. The movement parameters are set in the service registers, which, like the instructions of the stepper motor driver, are accessible via the Modbus protocol in RUN mode. The instruction SPIN has lower priority than xSTOP and xHIZ. To avoid errors, it is recommended to check the BUSY_MOVE and BUSY_RUN flags before calling the instruction SPIN. Below is a detailed description of the driver service registers.

Address	Object type	Sevice register		Size (bit)	Description
		Number	Name		
5000h	Holding Registers	D357	SPEED	32	The set (target) motor speed (**in pulses per second, pps**) for the commands of the RUN group and the maximum speed for the commands of the MOVE group. The lower threshold is 8 pps, the upper limit is 120000pps under the condition of FS_SW_EN reset. If FS_SW_EN is set, the upper limit is $SPEED_{max}=120000*2^{U\_STEP}$.
5002h	Holding Registers	D359	MIN_SPEED	32	The minimum rotation speed for the commands of the RUN group. For the MOVE group, it is also the minimum speed if the CMIN_SPD_EN flag is not set. If CMIN_SPD_EN is set, the optimal minimum speed is calculated automatically.
5004h	Holding Registers	D361	ACC	16	Acceleration, pps^2.
5005h	Holding Registers	D362	DEC	16	Deceleration, pps^2.
5006h	Holding Registers	D363	ABS	32	Current position. The unit of value is equal to the displacement by the amount of one microstep.

<table>
<tr><th rowspan="2">Address</th><th rowspan="2">Object type</th><th colspan="2">Sevice register</th><th rowspan="2">Size (bit)</th><th rowspan="2">Description</th></tr>
<tr><th>Number</th><th>Name</th></tr>
<tr><td>5008h</td><td>Input Registers</td><td>D365</td><td>U_POS</td><td>16</td><td>The current microstep position in four full steps. Indicates the position of the motor rotor relative to the stator poles. The register is read-only.</td></tr>
<tr><td>5009h</td><td>Holding Registers</td><td>D366</td><td>U_STEP</td><td>16</td><td>Microstepping
<table>
<tr><th>Register value</th><th>Microstepping</th></tr>
<tr><td>0</td><td>1/1</td></tr>
<tr><td>1</td><td>1/2</td></tr>
<tr><td>2</td><td>1/4</td></tr>
<tr><td>3</td><td>1/8</td></tr>
<tr><td>4</td><td>1/16</td></tr>
<tr><td>5</td><td>1/32</td></tr>
<tr><td>7</td><td>1/128</td></tr>
<tr><td>8</td><td>1/256</td></tr>
</table>
A value “6” is not valid.</td></tr>
<tr><td>500Ah</td><td>Holding Registers</td><td>D374</td><td>DIR</td><td>16</td><td rowspan="2">Rotation direction: "1" – forward, "0" – backward.</td></tr>
<tr><td>500Ah</td><td>Coils</td><td colspan="3">DIR</td></tr>
<tr><td>500Bh</td><td>Holding Registers</td><td>D377</td><td>FS_SPD_THR</td><td>32</td><td>The threshold value of turning from microstepping to fullstep mode, measured in full steps per second.</td></tr>
<tr><td>500Dh</td><td>Holding Registers</td><td>D379</td><td>FS_SW_EN</td><td>16</td><td rowspan="2">Setting the object to “1” turns on the morphing function – the controller turns to the fullstep mode after reaching the speed specified in FS_SPD_THR. This function allows to get greater torque at high speeds. Resetting the object to “0” turns off this function (morphing/torque boost).</td></tr>
<tr><td>500Dh</td><td>Coils</td><td colspan="3">FS_SW_EN</td></tr>
<tr><td>500Eh</td><td>Holding Registers</td><td>D372</td><td>TARGET_POS</td><td>32</td><td>The target position to be reached. The unit of value is equal to the displacement by one microstep.</td></tr>
<tr><td>5010h</td><td>Holding Registers</td><td>D376</td><td>CMD</td><td>16</td><td>A movement command to the driver (refer to the table below).</td></tr>
</table>

Tallinn Science Park Tehnopol,
Akadeemia tee 21/6, 12618, Tallinn, Estonia
Phone: + 372 6559914,
e-mail: mail@smd.ee
url: https://smd.ee

Address	Object type	Sevice register Number	Sevice register Name	Size (bit)	Description
5011h	Holding Registers	D375	SW_INPUT	16	The input number for sensor connection – for commands GOUNTIL_... and …RELEASE. Values **0…7**. **Attention:** The declaration of interruption in the main program is necessary for the used inputs, The interruption handle can be left empty. **Example:** A sensor is connected to the input X3 (IN3), the user program must include the next part: ***FEND*** ***I 1003*** ***IRET*** ***END***
5012h	Holding Registers	D367	ACC_CUR	16	Acceleration current, mA. Valid values range: 150…1500.
5013h	Holding Registers	D368	DEC_CUR	16	Deceleration current, mA. Valid values range: 150…1500.
5014h	Holding Registers	D369	STEADY_CUR	16	Constant speed current, mA. Valid values range: 150…1500.
5015h	Holding Registers	D370	HOLD_CUR	16	Holding current, mA. Valid values range: 150…1500.
5016h	Holding Registers	D382	CMIN_SPD_EN	16	"1" - use automatic calculation of the start and final speed of movement for the MOVE group commands, "0" - use MIN_SPEED as the start and final speed.
5017h	Holding Registers	D380	ERROR_SET_HIZ	16	Bits of this register determine which driver errors must lead to de-energizing the motor (the shaft rotates freely) - the **HiZ** state.
5017h	Coils	TERMAL_OVER_CURRENT_ERROR_SET_HIZ			0-bit of the register D380. If the bit is set, an error TERMAL_ERROR (the driver circuit overheating – the register D381) causes de-energizing the motor (HiZ state).
5018h	Coils	SOFTWARE_ERROR_SET_HIZ			1-bit of the register D380. If the bit is set, an error SOFTWARE_ERROR (the controller internal error – the register D381) causes de-energizing the motor (HiZ state).

Address	Object type	Sevice register		Size (bit)	Description
		Number	Name		
5019h	Coils	CMD_ERROR_SET_HIZ			2-bit of the register D380. If the bit is set, an error CMD_ERROR unabe to process an incoming command – the register D381) causes de-energizing the motor (HiZ state).
501Ah	Coils	DATA_ERROR_SET_HIZ			3-bit of the register D380. If the bit is set, an error DATA_ERROR (incorrect data entry in ACC, DEC, U_STEP – the register D381) causes de-energizing the motor (HiZ state).
501Bh	Coils	OUT_OF_LIM_MIN_SPD_ ERROR_SET_HIZ			4-bit of the register D380. If the bit is set, an error OUT_OF_LIM_MIN_SPD_ERROR _SET_HIZ (set speed less than minimum – the register D381) causes de-energizing the motor (HiZ state).
501Ch	Coils	OUT_OF_LIM_MAX_SPD_ ERROR_SET_HIZ			5-bit of the register D380. If the bit is set, an error OUT_OF_LIM_MAX_SPD_ERRO R_SET_HIZ (exceeding the maximum possible speed – the register D381) causes de-energizing the motor (HiZ state).
501Dh	Coils	UNREACHABLE_FS_SPD_ ERROR_SET_HIZ			6-bit of the register D380. If the bit is set, an error UNREACHABLE_FS_SPD_ERRO R_SET_HIZ (unable to reach fullstepspeed threshold in torque boost mode – the register D381) causes de-energizing the motor (HiZ state).
501Eh	Coils	NOT_APP_FS_PARAM_ ERROR_SET_HIZ			7-bit of the register D380. If the bit is set, an error NOT_APP_FS_PARAM_ERROR_ SET_HIZ (transition from torque boost is not possible while the motor is rotating – the register D381) causes de-energizing the motor (HiZ state).
5027h	Holding Registers	D381	ERROR_CODE	16	Error code. See below the description of register bits (errors).
5027h	Coils	TERMAL_OVER_CURRENT_ ERROR			0-bit of the register D381 TERMAL_OVER_ CURRENT_ERROR – overheating of the driver circuit or excess current in the motor windings.

Address	Object type	Sevice register		Size (bit)	Description
		Number	Name		
5028h	Coils	SOFT_ERROR			1-bit of the register D381 SOFTWARE_ERROR – controller internal error.
5029h	Coils	CMD_ERROR			2-bit of the register D381 CMD_ERROR – unable to process an incoming command.
502Ah	Coils	DATA_ERROR			3-bit of the register D381 DATA_ERROR – Incorrect data entry in ACC, DEC, U_STEP
502Bh	Coils	OUT_OF_LIM_MIN_SPD_ERROR			4-bit of the register D381 OUT_OF_LIM_ MIN_SPD_ERROR – set speed is less than minimum.
502Ch	Coils	OUT_OF_LIM_MAX_SPD_ERROR			5-bit of the register D381 OUT_OF_LIM_ MAX_SPD_ERROR – exceeding the maximum possible speed.
502Dh	Coils	UNREACHABLE_FS_SPD_ERROR			6-bit of the register D381 UNREACHABLE _FS_SPD_ERROR – unable to reach fullstepspeed threshold in torque boost mode
502Eh	Coils	NOT_APP_FS_PARAM_ERROR			7-bit of the register D381 . NOT_APP_FS_ PARAM_ERROR – transition from torque boost is not possible while the motor is rotating
5037h	Input Registers	D371	MOTOR_STATUS	16	The register shows the current state of the motor and control system. The description of the register bits is below.
5037h	Discrete Inputs	HIZ			0-bit of the register D371. HiZ-state of the motor (phases are de-energized).
5038h	Discrete Inputs	STOP			1-bit of the register D371. Holding mode.
5039h	Discrete Inputs	ACCELERATING			2-bit of the register D371. Acceleration.
503Ah	Discrete Inputs	DECELERATING			3-bit of the register D371. Deceleration.
503Bh	Discrete Inputs	STEADY			4-bit of the register D371. Constant speed rotation.
503Ch	Discrete Inputs	BUSY_MOVE			5-bit of the register D371. Flag of the impossibility of applying the commands of the MOVE group.
503Dh	Discrete Inputs	BUSY_RUN			6-bit of the register D371. Flag of the impossibility of using the commands of the RUN group.

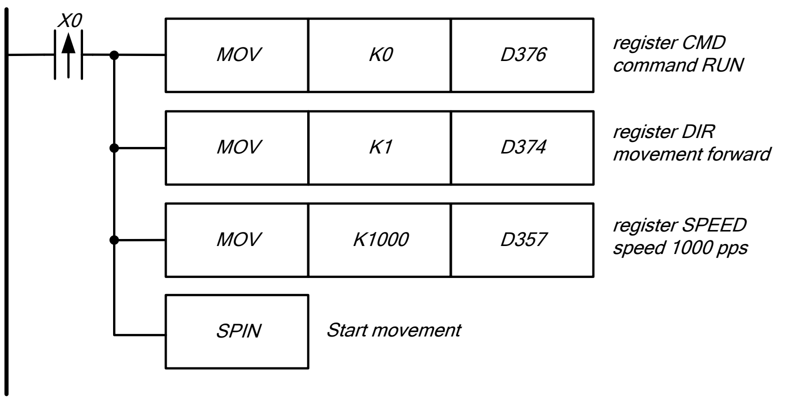
Address	Object type	Sevice register		Size (bit)	Description
		Number	Name		
5047h	Input Registers	D383	CURRENT_SPD	32	Current speed, pps. (It is recommended to use the STEADY flag as an event of reaching a given speed).
5100h	Coils	SPIN (APPLY_CMD)			Setting the object to “1” or applying the **SPIN** instruction activates the execution of the command specified in the CMD register (D376), with the specified parameters.
5101h	Coils	TORQUE (APPLY_CURRENT)			Setting an object to “1” or applying a **TORQUE** instruction applies current values ACC_CUR, DEC_CUR, RUN_CUR, HOLD_CUR for the motor.
5102h	Coils	HSTOP (HARD_STOP)			Setting an object to “1” or applying the **HSTOP** instruction immediately stops the motor and turns to holding mode.
5103h	Coils	HHIZ (HARD_HIZ)			Setting an object to “1” or applying the **HHIZ** instruction immediately turns the motor to HiZ state.
5104h	Coils	SSTOP (SLOW_STOP)			Setting an object object to “1” or applying the **SSTOP** instruction stops the motor according to the DEC and then turns to holding mode.
5105h	Coils	SHIZ (SLOW_HIZ)			Setting an object to “1” or applying the SHIZ instruction stops the motor according to the DEC and then turns to HiZ state.

Movement command (CMD-register)

Value	Group	Name	Description
0	RUN	RUN	Rotation according set speed SPEED, acceleration ACC, deceleration DEC, direction DIR.
1	MOVE	MOVE	**Displace by** the specified number of steps TARGET_POS with the given motion parameters SPEED, ACC, DEC, DIR.
2	MOVE	GOTO	**Move to** the specified position TARGET_POS with the given motion parameters SPEED, ACC, DEC. DIR depends on current position, the gived value is not taken into account.
3	MOVE	GOTO_DIR	**Move to** the specified position TARGET_POS with the given motion parameters SPEED, ACC, DEC, DIR.

4	MOVE	GOHOME	Move to the position **"0"** with the given motion parameters SPEED, ACC, DEC. The command is equal to GOTO **"0".**
5	RUN	GOUNTIL_SLOWSTOP	Motion with a set speed SPEED, acceleration ACC, direction DIR until the sensor SW_INPUT is triggered on a rising edge with an initial check of the input level with following decelerating and stop according to a set DEC.
6	RUN	GOUNTIL_FRONT_SLOWSTOP	Motion with a set speed SPEED, acceleration ACC, direction DIR until the sensor SW_INPUT triggers on a rising edge, with following decelerating and stop according to a set DEC.
7	RUN	GOUNTIL_HARDSTOP	Motion with a set speed SPEED, acceleration ACC, direction DIR until the sensor SW_INPUT is triggered on a rising edge with an initial check of the input level and then turn to holding mode.
8	RUN	GOUNTIL_FRONT_HARDSTOP	Motion with a set speed SPEED, acceleration ACC, direction DIR until the sensor SW_INPUT is triggered on a rising edge and then turn to holding mode.
9	RUN	RELEASE	Motion with a set speed SPEED, ACC acceleration, DIR direction until the sensor SW_INPUT triggers on the falling edge with an initial check of the input level and then turn to holding mode.
10	RUN	FRONT_RELEASE	Motion with a set speed SPEED, acceleration ACC, direction DIR until the sensor SW_INPUT triggers on the falling edge and then turn to holding mode.

Example:



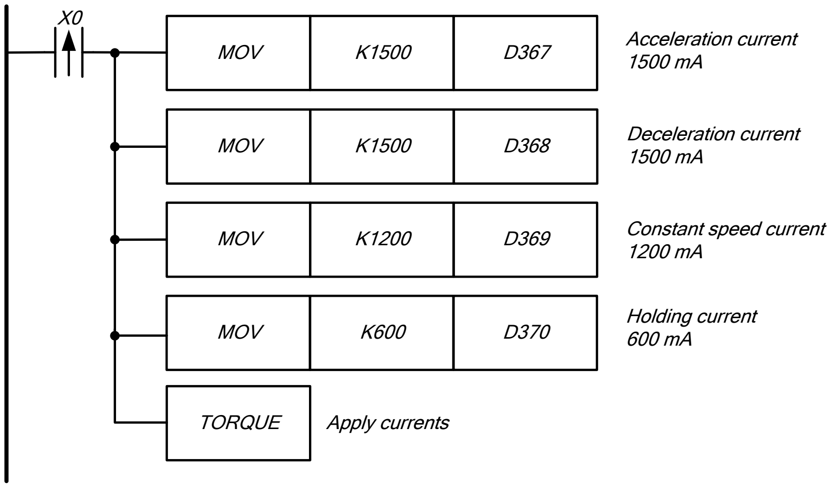
When the entry condition is satisfied, the motion command, direction and speed of rotation are set in the service registers. The followed instruction SPIN starts moving.

TORQUE		P	Apply the set currents to the motor
5101h	**Coils**	Writing “1” to an object (even if it is not reset) applies to the motor the values of current, which are set in service registers. Reset (value = “0”) is ignored.	

Description:

Applying of this instruction sets the operating currents of the motor indicated in the registers ACC_CUR (D367), DEC_CUR (D368), RUN_CUR (D369), HOLD_CUR (D370).

Example:

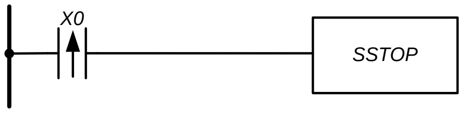


SSTOP		P	The motor stops according to the DEC parameter and then goes into holding mode.
5104h	**Coils**	Writing “1” to an object (even if it is not reset) applies motor braking according to DEC and then goes to holding mode, reset is ignored.	

Description:

Applying of braking according to DEC and then turning to holding mode. This instruction overrides the SPIN operation, has the same priority as SHIZ, but can be overridden by the HSTOP and HHIZ instructions.

Example:

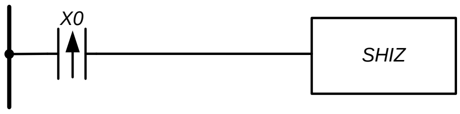


SHIZ		P	The motor stops according to the DEC parameter and then goes to HiZ mode.
5105h	**Coils**	Writing “1” to an object (even if it is not reset) applies motor braking according to DEC and then goes to HiZ mode, reset is ignored.	

Description:

Applying of braking according to DEC followed by de-energization of the windings. This instruction overrides the SPIN operation, has the same priority as SSTOP, but can be overridden by the HSTOP and HHIZ instructions.

Example:

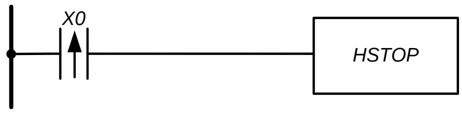


HSTOP		P	Immediate stops the motor and then goes to holding mode.
5102h	**Coils**	Writing “1” to an object (even if it is not reset) immediately stops the motor and then goes to holding mode, reset is ignored.	

Description:

Immediate stops the motor and then goes to holding mode. This instruction overrides SPIN, SSTOP, SHIZ, and has the same priority as HHIZ.

Example:



HHIZ		P	Immediate stops the motor and then goes to HiZ mode.
5103h	**Coils**	Writing “1” to an object (even if it is not reset) immediately stops the motor and then goes to HiZ mode, reset is ignored.	

Description:

Immediate stops the motor and then goes to HiZ mode (the motor is de-energized, the shaft rotates freely). This instruction overrides SPIN, SSTOP, SHIZ, and has the same priority as HSTOP.

Example:

9. Communication parameters

The controller has a USB and RS-485 interfaces, both have the same access to registers and bit operands. The USB interface is a virtual COM port (VCP), it is mainly intended for configuring of the controller and recording of user program, therefore it has fixed communication parameters: Modbus ASCII, ID 1, 115200 baud, 7, Even, 1. Parameter variations for RS-485 are indicated in Appendix A in section “RS-485 interface communication parameters”.

Change communication settings for RS-485

Set the required communication parameters according to the section “RS-485 interface communication parameters” of Appendix A. For the changes to take effect, reboot the device. This can be done by turning the power off and on or by setting the Coils 8101h (Reset) object.

Example:

It is necessary to change communication parameters to the next: Modbus RTU, ID 100, 128000 baud, 8, Odd, 1. There are all possible combinations of communication parameters in the Appendix A

Action sequence:

1) Writing the value 100d into the Holding Registers 8103h – change device address (ID) to 100.
2) Setting Coils 8100h – protocol selection RTU.
3) Writing the value 128000d into the Holding Registers 8100h – setting data transfer speed 128000 baud.
4) Writing the value 1d into the Holding Registers 8102h – parity type selection Odd.
5) Setting Coils 8101h – reboot the controller.

Modbus Protocol

It is strongly recommended to read the protocol specification on the site. http://modbus.org/. Supported protocol functions are presented in the table below:

			Function	**Code**
Data access	Discrete Inputs	1-bit	Read Discrete Inputs	02(02h)
	Coils		Read Coils	01(01h)
			Write Single Coil	05(05h)
			Write Multiple Coils	15(0Fh)
	Input Registers	16-bit	Read Input Register	04(04h)
	Holding Registers		Read Holding Registers	03(03h)
			Write Single Register	06(06h)
			Write Multiple Registers	16(10h)
			Read/Write Multiple Registers	23(17h)
			Mask Write Register	22(16h)

Protocol error codes are presented in the table below:

Error code	Description
01(01h)	*ILLEGAL FUNCTION* The function code cannot be processed.
02(02h)	*ILLEGAL DATA ADDRESS* The address of the register specified in the request is not available.
03(03h)	*ILLEGAL DATA VALUE* The value contained in the request data field is invalid.
04(04h)	*SERVER DEVICE FAILURE* An unrecoverable error occurred while performing the requested action.

Error codes recorded during the processing of protocol packets are presented in the tables below.

Address	Type	Size	Description
E003h	Input Registers	16-bit	Error code while processing Modbus frame.
E003h	Coils	1-bit	Flag of an error during the exchange via Modbus protocol.

Error code	Description
0001h	Memory allocation error.
0002h	Checksum error.
0003h	An error occurred while receiving and processing a broadcast packet.
0004h	Frame size mismatch.
0005h	Function error (0Fh). Not all bits have been overwritten..
0006h	Function error (10h). Not all registers have been overwritten..
0007h	Function error (17h). Not all registers have been overwritten..
0008h	Lost frame due to DMA error.
0009h	Lost frame due to overflowing frame processing stack.

If the device is the end in the RS-485 communication line, then connect a terminal resistor by turning on the toggle switch next to the RS-485 connector, as shown in the Fig. 29.

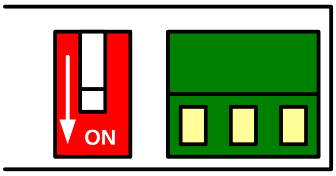


Fig. 29 – Terminal resistor connection.

10. Setting the real time clock

The controller has a real-time clock, which is powered by an internal source (CR2032 battery), which ensures the operation of the clock while the main power is off. The same battery is used for the operation of non-volatile registers and safety settings of the controller communication parameters. The indicator **BAT** lights up in case of the absence or soon failure of the internal CR2032 battery. The real-time clock can be set via the user program using the TWR instruction or via the Modbus protocol in the following order:

1) Disable auto overwrite of the Holding Registers 8110h…8112h by resetting of the Coils 8110h.
2) Recording the current time value into the Holding Registers 8110h, 8111h, 8112h seconds, minutes, hours, respectively (refer to the section “Clock setting” in the Appendix A).
3) Set a new time value by setting the Coils 8111h.
4) Enable auto overwrite of the Holding Registers 8110h…8112h by setting of the Coils 8110h.

11. A user program - loading to and reading from the controller

The controller has two areas for downloading programs: general purpose and special.

The general purpose area is intended for loading a user program with maximum length up to 59754 lines (the area is empty by default). The maximum length of the special area is 1928 lines. This area contains a program for controlling the speed of a stepper motor using a potentiometer, buttons and encoder. If necessary, this area can be overwritten.

Below is an example of a user program. The list of the registers involved in these operations is given in Appendix A in the section “Working with ROM”.

User program in LD form:

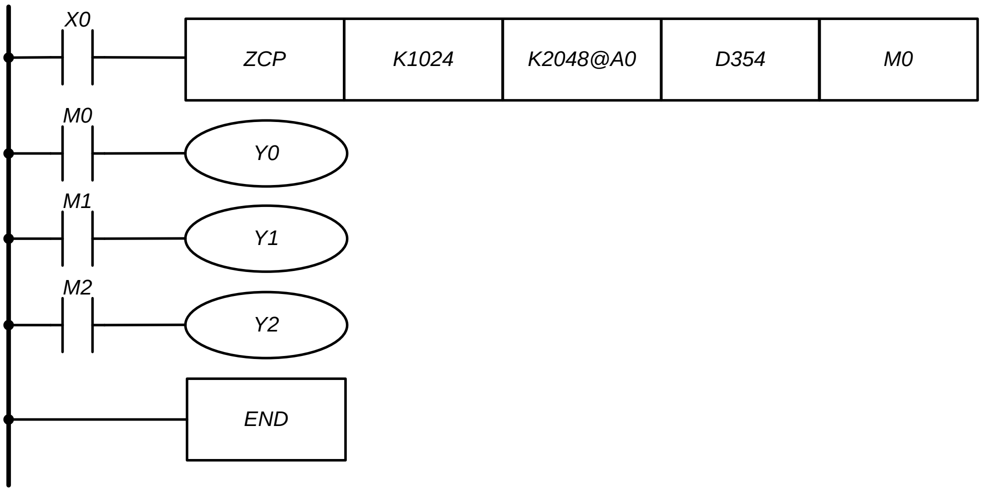


Fig. 30 – User program

The user program converted into IL:

LD	X0	*;entry condition for zone comparison operation*
ZCP	K1024 K2048@A0 D354 M0	*;zone comparison, determining the position of the; potentiometer SPEED (2)*
LD	M0	*;if the value in the register D354 is less than 1024, then Y0 is turned ON, otherwise – turned off*
OUT	Y0	
LD	M1	*;if the value in the register D354 is greater than or equal to 1024 and less than or equal to the sum of 2048 and the value of A0 then Y1 is turned on, otherwise – turned off.*
OUT	Y1	
LD	M2	*if the value in the register D354 is greater than the sum of 2048 and the value of A0, then Y2 is turned on,*
OUT	Y2	
END		*;end of the program*

All supported by the controller instruction codes are presented in Appendix B. Use it when assembling a user program.

1) Make sure that the controller is in the STOP mode. Changing the user program in the RUN mode is impossible. To check the RUN/STOP state of the controller, read the value of Discrete Inputs F001h. It is reset in the STOP mode, it is set in the RUN mode.

2) To read the program from the controller: check if it is not read-protected before reading a program from the controller. There are two read-protection objects in the controller: Coils F001h (for protection of a user program) and F002h (for protection of a service program). It is impossible to read the program if the protection is set for the program. If the protection is not set, go to the step 5 to read the program from the controller.
3) Before writing a new program to the controller it is necessary to erase the previous one. Set the Coils F003h to erase the user program or Coils F004h to erase a service program. In this example the main program is writing, so it is necessary to set the Coils F003h.
4) After setting the Coils F003h (or F004h), wait until Discrete Inputs F000h is reset, this will indicate the completion of the erase procedure and the readiness of the ROM for further work.
5) After erasing previous program it is necessary to set the operation type – read or write. To write a new program set the Coils F005h, to read the program from the controller – reset the Coils F005h.
6) Select the type of program. For the user program reset the Coils F006h, for the service program set the Coils F006h.
7) Set the line number for writing/reading the program using Holding Registers F100h. Numbering starts from 0. For the read operation go to step 10. To write a new program set it's value 0.
8) Fill the download sector F300h ... F314 according to the following example:

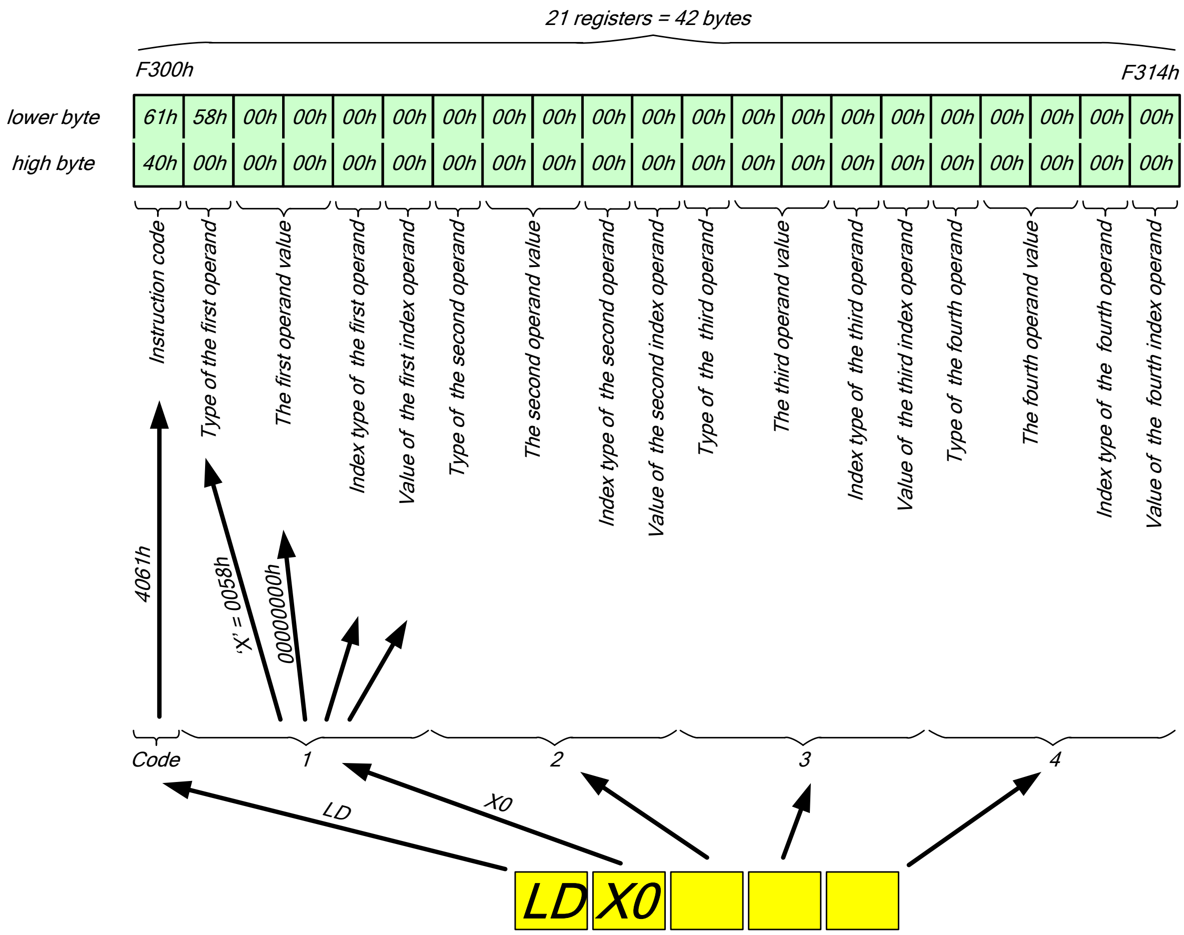


Fig. 31 – Projecting an instruction with operands into the address space of the Modbus protocol.

9) Setting the Coils F000h starts the operation parameterized in Coils F005h and F006h. In this example - writing the first line to the controller ROM. Thus, repeating steps 7 - 9, moving down the program to the end, incrementing Holding Registers F100h, the program is recorded in the controller.
   As an example, below is the formation of the downloading sector from the second line of the program.

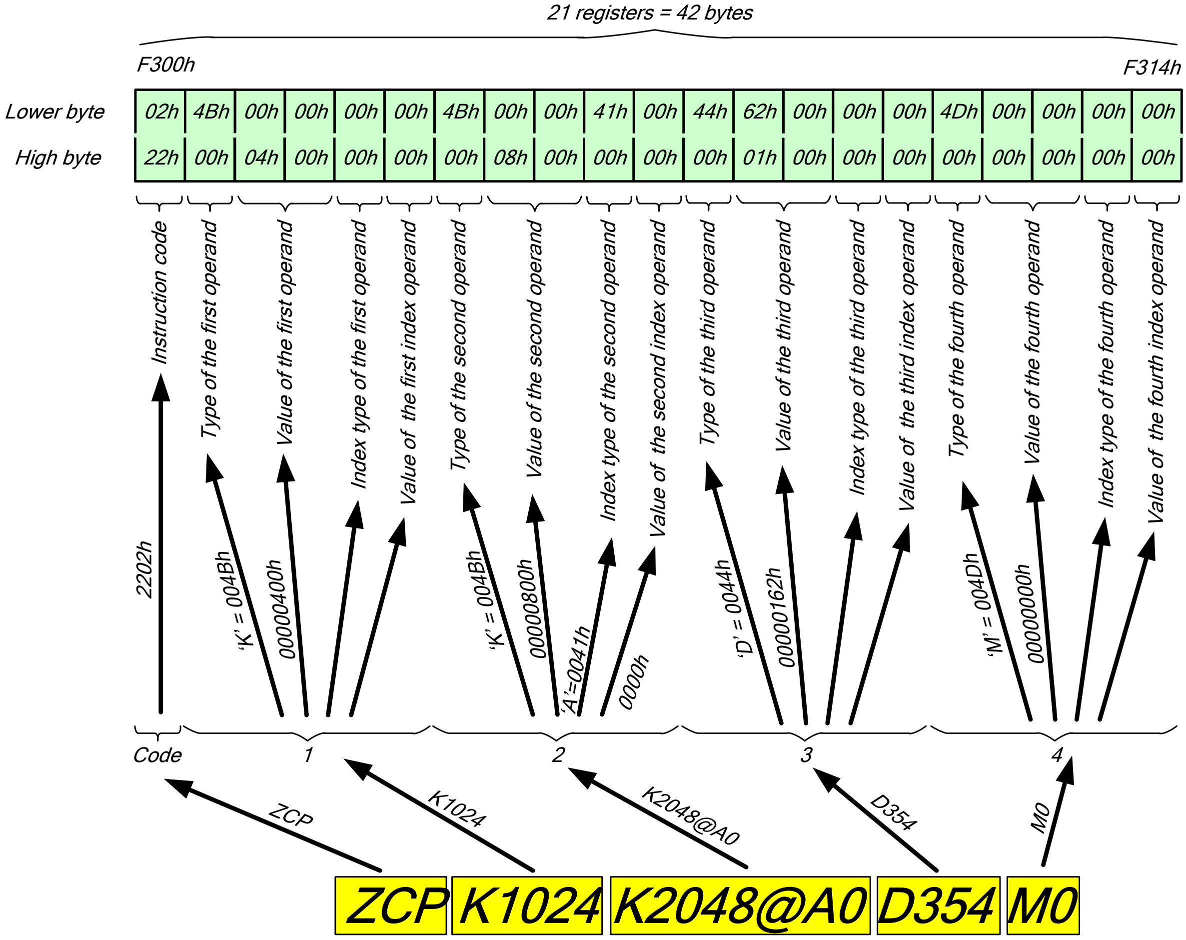


Fig. 32 – Projecting an instruction with operands into the address space of the Modbus protocol.

10) To read the program, the opposite operation is needed. The difference is that the upload sector has the address Input Registers F200h ... F214h and the parameterized operation is performed first by setting Coils F000h, and then reading the sector and then incrementing the line number until the END instruction arrives.

The operand types codes are shown in the table below:

Operand	Code
K	4Bh
H	48h
F	46h
X	58h
Y	59h
M	4Dh
T	54h
C	43h
A	41h

B	42h
D	44h
P	50h
I	49h

Error codes that occur when working with ROM are presented in the tables below:

Address	Type	Size	Description
E002h	Input Registers	16-bit	Error code when working with ROM
E002h	Coils		Flag of an error in the process of working with ROM.

Error code	Description
0001h	Read protection for the main program has not been set.
0002h	Read protection for the service program has not been set.
0003h	Failed to erase the main program sector.
0004h	Failed to erase the service program sector.
0005h	Failed to write the instruction to the main program.
0006h	Failed to write the instruction to the service program.

12.Speed control mode

This mode is intended for controlling the rotation speed of a stepper motor using the built-in potentiometer “SPEED” (2), buttons or an encoder.

To enter the speed control mode, set the controller to the STOP state, then use the mode select button to set the SPD mode. Assemble and connect to the controller the circuit shown in the Fig. 33 – Connection of control elements

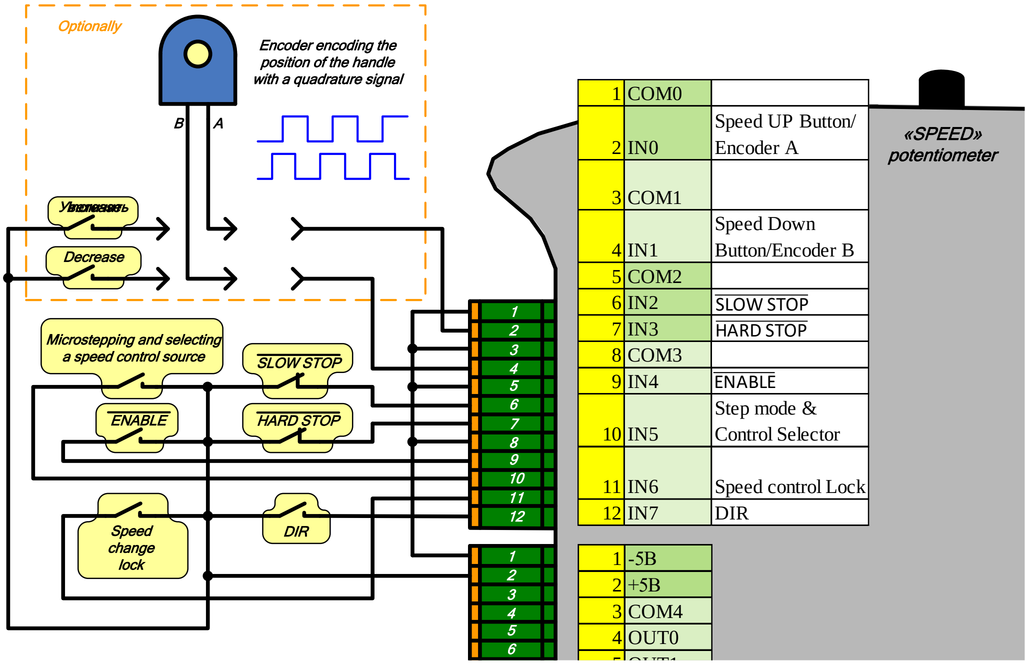


Fig. 33 – Connection of control elements

Attention
Turning the controller to the **RUN** mode while SLOW STOP and HARD STOP switches are closed and ENABLE switch is open, will **cause the motor rotation**. To avoid uncontrolled rotation, turn the “SPEED” potentiometer to the minimum position or change the position of any of the above switches to the opposite one indicated in the diagram.

In the **RUN** state, select the required microstepping by pressing the corresponding button. The method of the speed control is selected by the IN5 input, more details in the table below.

LED indication OUT0…7	Microstepping
OUT0	1/1
OUT1	1/2
OUT2	1/4
OUT3	1/8
OUT4	1/16
OUT5	1/32
OUT6	1/128
OUT7	1/256

LED indication OUT10…11	Speed control source
Both are turned off	The speed is adjusted by the potentiometer «SPEED».
OUT10	The speed is adjusted by the buttons “Increase” and “Decrease” (IN0 and IN1). The speed change increment is set by the potentiometer «SPEED».
OUT11	The speed is adjusted by an encoder, connected to the inputs IN0 and IN1. The changing speed increment for one encoder event is set by the potentiometer «SPEED».

When the speed change lock is turned on, the controller stops responding to the speed controls. This option is designed to prevent accidental mechanical impact on the potentiometer, encoder, buttons.

DIR – changes the motor rotation direction.

ENABLE – controls energizing of stepper motor phases.

$\overline{\text{HARD STOP}}$ – opening of the circuit immediately stops and turns the motor into the holding mode. The holding current is 50% of the work current (150…750 mA). The value of work current is set by the potentiometer (**1**) from 150 to 1500 mA.

$\overline{\text{SLOW STOP}}$ – opening of the circuit causes the motor stop according to the deceleration set by the potentiometer (0) (acceleration value is also set by the potentiometer (0)).

The code of the service program is given in the Appendix D. Code of the service program “Stepper Motor Speed Control”. The code can be modified to meet specific requirements.

13. Step/Dir pulse position control mode

The controller provides pulse position control mode by pulse step signals STEP (the inputs STEP +, STEP-) and direction signal DIR (the inputs DIR +, DIR. The ENABLE + input controls the the motor phases energizing. INVERT_ENABLE + inverts the ENABLE signal. The FAULT output indicates alarm states: overcurrent and overheating, or missing steps due to these two reasons (see Fig. 2).

To turn the controller to pulse position control mode, first turn it to the **STOP** state and then use the mode select button to switch to **DRV** mode. In this state the motor phases are de-energized. Select necessary microstepping, work current and holding current. (the transition to the mode is carried out after one-second after the detection of the last Step signal on the leading edge of the pulse).

Microstepping is set by the SPEED potentiometer, the values of work current – by the potentiometer (0), holding current – by the potentiometer (1). Set parameters are displayed on the LED panel, more details in the tables below:

LED indication IN0…7	**Microstepping**
IN0	1/1
IN1	1/2
IN2	1/4
IN3	1/8
IN4	1/16
IN5	1/32
IN6	1/128
IN7	1/256

OUT0	**OUT1**	**OUT2**	**OUT3**	**Work current**
OUT4	**OUT5**	**OUT6**	**OUT7**	**Holding current**
●				150 mA
	●			245 mA
●	●			340 mA
		●		440 mA
●		●		535 mA
	●	●		630 mA
●	●	●		730 mA
			●	825 mA
●			●	920 mA
	●		●	1020 mA
●	●		●	1115 mA
		●	●	1210 mA
●		●	●	1310 mA
	●	●	●	1400 mA
●	●	●	●	1500 mA

When the controller is in the **RUN** state, the above parameters are fixed and saved after the power is off. Use the inputs and outputs of the controller according to the pin assignment table (see Fig. 2).

14.User program control mode

The controller provides control mode according a user program and control by Modbus commands. The controller indicates this control mode by LED indicator **PROG**. A user program can be sent to the controller memory when the controller is in the **STOP** state. After turning to the **RUN** state the controller starts executing of the user program. It is also possible to control the state of the controller, user program, physical outputs, stepper motor driver and monitor the status of physical inputs via RS-485 interface using Modbus protocol.

Examples of user programs that demonstrate the basic functionality of the controller are described in the Appendix C. Examples of user programs

Appendix A. Registers of the controller

Address	Type	Size	Description
	RS-485 interface communication parameters		
0x8100	Coils	-	Communications protocol selection Reset — Modbus ASCII. Set — Modbus RTU. Changes take effect after the controller reboot.
0x8100	Holding Registers	32-bit	Baud rate. Allowable values: 110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 57600, 115200, 128000, 256000. Changes take effect after the controller reboot..
0x8102	Holding Registers	16-bit	Parity: 0 – NONE 1 – ODD 2 – EVEN
			The following data transfer parameters are available.: *Modbus RTU:* ■ *8-bit / EVEN / 1 STOP* ■ *8-bit / ODD / 1 STOP* ■ *8-bit / NONE / 2 STOP* *Modbus ASCII:* ■ *7bit / EVEN / 1 STOP* ■ *7bit / ODD / 1 STOP* *Stop bit parameters are set automatically.*
0x8103	Holding Registers	16-bit	ID of the controller (device address). Allowable values: 1...247.
	Clock setting		
0x8110	Holding Registers	16-bit	Seconds. Allowable values: 0...59.
0x8111	Holding Registers	16-bit	Minutes. Allowable values: 0...59.
0x8112	Holding Registers	16-bit	Hours. Allowable values: 0...23.
0x8110	Coils	-	Automatic registers update. Set — in registers 0x8110 - 0x8112 actual value of time. Reset — automatic updating of data is disabled, recording of user values is allowed.
0x8111	Coils	-	Setting the object sets the time from the registers 0x8110 - 0x8112. It is allowed to turn on automatic updating again after setting the object.
	Additional		
0x8101	Coils	-	Setting the object causes a reboot of the controller.

Address	Type	Size	Description
	Working with ROM		
0xF001	Discrete Inputs	-	RUN/STOP toggle switch state. ROM operations are not possible when the controller is in the RUN state. Reset —STOP state. Set —RUN state.
0xF000	Discrete Inputs	-	Indication of ROM state. Reset — ROM is ready for operation. Set — ROM is busy.
0xF000	Coils	-	Control object. Setting the object starts the operation parameterized in objects 0xF005 and 0xF006.
0xF001	Coils	-	Read protection of the main (user) program. Set — not protected. Reset — read protection is set. Attempting to set the object causes erasing of the main program.
0xF002	Coils	-	Read protection of the service program. Set — not protected. Reset — read protection is set. Attempting to set the object causes erasing of the service program.
0xF003	Coils	-	Erasing the main program. Setting the object starts the erase procedure of the main program. Resetting the object is ignored.
0xF004	Coils	-	Erasing the service program. Setting the object starts the erase procedure of the service program. Resetting the object is ignored.
0xF005	Coils	-	Operation type selection. Reset — read. Set — write.
0xF006	Coils	-	Program selection (main/service) Reset — main. Set — service.
0xF100	Holding Registers	16-bit	The line number in the program to be read or overwritten (0 ... 59753 - for the main program, 0 ... 1927 - for the service).
	ROM reading sector		
0xF200	Input Registers	16-bit	Instruction code
0xF201	Input Registers	16-bit	Type of the first operand
0xF202	Input Registers	32-bit	Value of the first operand
0xF204	Input Registers	16-bit	Index type of the first operand

Address	Type	Size	Description
0xF205	Input Registers	16-bit	Value of the first index operand
0xF206	Input Registers	16-bit	Type of the second operand
0xF207	Input Registers	32-bit	Value of the second operand
0xF209	Input Registers	16-bit	Index type of the second operand
0xF20A	Input Registers	16-bit	Value of the second index operand
0xF20B	Input Registers	16-bit	Type of the third operand
0xF20C	Input Registers	32-bit	Value of the third operand
0xF20E	Input Registers	16-bit	Index type of the third operand
0xF20F	Input Registers	16-bit	Value of the third index operand
0xF210	Input Registers	16-bit	Type of the fourth operand
0xF211	Input Registers	32-bit	Value of the fourth operand
0xF213	Input Registers	16-bit	Index type of the fourth operand
0xF214	Input Registers	16-bit	Value of the fourth index operand
	ROM writing sector		
0xF300	Holding Registers	16-bit	Instruction code
0xF301	Holding Registers	16-bit	Type of the first operand
0xF302	Holding Registers	32-bit	Value of the first operand
0xF304	Holding Registers	16-bit	Index type of the first operand
0xF305	Holding Registers	16-bit	Value of the first index operand
0xF306	Holding Registers	16-bit	Type of the second operand
0xF307	Holding Registers	32-bit	Value of the second operand

Address	Type	Size	Description
0xF309	Holding Registers	16-bit	Index type of the second operand
0xF30A	Holding Registers	16-bit	Value of the second index operand
0xF30B	Holding Registers	16-bit	Type of the third operand
0xF30C	Holding Registers	32-bit	Value of the third operand
0xF30E	Holding Registers	16-bit	Index type of the third operand
0xF30F	Holding Registers	16-bit	Value of the third index operand
0xF310	Holding Registers	16-bit	Type of the fourth operand
0xF311	Holding Registers	32-bit	Value of the fourth operand
0xF313	Holding Registers	16-bit	Index type of the fourth operand
0xF314	Holding Registers	16-bit	Value of the fourth index operand
	Errors		
0xE000	Coils	-	Setting the object resets all types of errors that are valid for the current state of the controller.
0xE000	Discrete Inputs	-	General error. It is always set when at least one of the types of errors from 0xE001 to 0xE004 appears.
0xE001	Discrete Inputs	-	The set state of the object indicates a discharged CR2032 battery inside the controller. Replacement is required.
0xE002	Discrete Inputs	-	The set state of the object indicates an error during ROM operation. The error code is specified in the Input Registers 0xE002.
0xE003	Discrete Inputs	-	The set state of the object indicates an error during the exchange process using the Modbus protocol. The error code is specified in the Input Registers 0xE003.
0xE004	Discrete Inputs	-	The set state of the object indicates an error during the user program execution. The error code is specified in the Input Registers 0xE004. The caused an error line of the program is indicted in 0xE084.
0xE002	Input Registers	16-bit	ROM operation error code.
0xE003	Input Registers	16-bit	Modbus protocol error code.

Address	Type	Size	Description
0xE004	Input Registers	16-bit	User program error code.
0xE084	Input Registers	16-bit	The number of a line, caused error in the user program (numbering starts from 0, see description of the Coils 0xF100 above).
	Access to program operands ***Discrete outputs***		
0x1000	Discrete Inputs	-	Discrete output Y0
0x1001	Discrete Inputs	-	Discrete output Y1
⋮	⋮	⋮	⋮
0x107F	Discrete Inputs	-	Discrete output Y177
	State of discrete physical inputs		
0x2000	Discrete Inputs	-	Discrete input X0
⋮	⋮	⋮	⋮
0x2007	Discrete Inputs	-	Discrete input X7
	Discrete inputs		
0x2008	Coils	-	Discrete input X10
0x2009	Coils	-	Discrete input X11
⋮	⋮	⋮	⋮
0x207F	Coils	-	Discrete input X177
	General purpose data registers D192...D255		
0x3000	Input Registers	16-bit	Register D192
0x3001	Input Registers	16-bit	Register D193
⋮	⋮	⋮	⋮
0x303F	Input Registers	16-bit	Register D255

Address	Type	Size	Description
	General purpose data registers D256...D319		
0x4000	Holding Registers	16-bit	Register D256
0x4001	Holding Registers	16-bit	Register D257
⋮	⋮	⋮	⋮
0x403F	Holding Registers	16-bit	Register D319
	Non-volatile data registers D320...D327		
0x3100	Input Registers	16-bit	Register D320
⋮	⋮	⋮	⋮
0x3107	Input Registers	16-bit	Register D327
	Non-volatile data registers D328...335		
0x4100	Holding Registers	16-bit	Register D328
⋮	⋮	⋮	⋮
0x4107	Holding Registers	16-bit	Register D335
	Hardware counters		
0x4200	Holding Registers	32-bit	Counter C64
0x4202	Holding Registers	32-bit	Counter C65
	Analog-to-digital converters		
0x3200	Input Registers	16-bit	Register D352, data from the potentiometer «0»
0x3201	Input Registers	16-bit	Register D353, data from the potentiometer «1»
0x3202	Input Registers	16-bit	Register D354, data from the potentiometer «SPEED»
	Hardware and software versions		
0x8001	Input Registers	16-bit	Major hardware version

Address	Type	Size	Description
0x8002	Input Registers	16-bit	Minor hardware version
0x8003	Input Registers	16-bit	Major software version
0x8004	Input Registers	16-bit	Minor software version
0x8005	Input Registers	16-bit	Major bootloader version
0x8006	Input Registers	16-bit	Minor bootloader version
	Stepper motor control		
0x5000	Holding Registers	32-bit	Register D357 – SPEED.
0x5002	Holding Registers	32-bit	Register D359 – MIN_SPEED.
0x5004	Holding Registers	16-bit	Register D361 – ACC.
0x5005	Holding Registers	16-bit	Register D362 – DEC.
0x5006	Holding Registers	32-bit	Register D363 – ABS.
0x5008	Input Registers	16-bit	Register D365 – U_POS.
0x5009	Holding Registers	16-bit	Register D366 – U_STEP.
0x500A	Holding Registers	16-bit	Register D374 – DIR.
0x500A	Coils	-	Register D374 – DIR.
0x500B	Holding Registers	32-bit	Register D377 – FS_SPD_THR.
0x500D	Holding Registers	16-bit	Register D379 – FS_SW_EN.
0x500D	Coils	-	Register D379 – FS_SW_EN.
0x500E	Holding Registers	32-bit	Register D372 – TARGET_POS.
0x5010	Holding Registers	16-bit	Register D376 – CMD.
0x5011	Holding Registers	16-bit	Register D375 – SW_INPUT.
0x5012	Holding Registers	16-bit	Register D367 – ACC_CUR.

Address	Type	Size	Description
0x5013	Holding Registers	16-bit	Register D368 – DEC_CUR.
0x5014	Holding Registers	16-bit	Register D369 – RUN_CUR.
0x5015	Holding Registers	16-bit	Register D370 – HOLD_CUR.
0x5016	Holding Registers	16-bit	Register D382 – CMIN_SPD_EN.
0x5017	Holding Registers	16-bit	Register D380 – ERROR_SET_HIZ.
0x5017	Coils	-	TERMAL_ERROR_SET_HIZ
0x5018	Coils	-	SOFTWARE_ERROR_SET_HIZ
0x5019	Coils	-	CMD_ERROR_SET_HIZ
0x501A	Coils	-	DATA_ERROR_SET_HIZ
0x5027	Holding Registers	16-bit	Register D381 – ERROR_CODE.
0x5027	Coils	-	TERMAL_ERROR_OVER_CURRENT
0x5028	Coils	-	SOFT_ERROR
0x5029	Coils	-	CMD_ERROR
0x502A	Coils	-	DATA_ERROR
0x5037	Input Registers	16-bit	Register D371 – MOTOR_STATUS.
0x5037	Discrete Inputs	-	HIZ
0x5038	Discrete Inputs	-	STOP
0x5039	Discrete Inputs	-	ACCELERATING
0x503A	Discrete Inputs	-	DECELERATING
0x503B	Discrete Inputs	-	STEADY
0x503C	Discrete Inputs	-	BUSY_MOVE
0x503D	Discrete Inputs	-	BUSY_RUN
0x5100	Coils	-	Instruction SPIN – APPLY_CMD.
0x5101	Coils	-	Instruction TORQUE – APPLY_CURRENT.
0x5102	Coils	-	Instruction HSTOP – HARD_STOP.
0x5103	Coils	-	Instruction HHIZ – HARD_HIZ.

Address	Type	Size	Description
0x5104	Coils	-	Instruction SSTOP – SLOW_STOP.
0x5105	Coils	-	Instruction SHIZ – SLOW_HIZ.

Appendix B – List of instructions

Instruction API	Code	Description
		Basic instructions
LD	0x4061	Normally open contact
LDI	0x4001	Normally closed contact
AND	0x4065	Series connection - normally open contact (logic AND)
ANI	0x4005	Series connection - normally closed contact (logic NAND)
OR	0x4066	Parallel connection – normally open contact (logic OR)
ORI	0x4046	Parallel connection – normally closed contact (logic NOR)
LDP	0x4821	Beginning of logical expression with rising edge polling (impulse)
LDF	0x4841	Beginning of a logical expression with polling on a falling edge (impulse)
ANDP	0x4825	«AND» with a rising edge polling (impulse)
ANDF	0x4845	«AND» with a falling edge polling (impulse)
ORP	0x4806	«OR» with rising edge polling (impulse)
ORF	0x4826	«OR» with falling edge polling (impulse)
TMR	0x2014	Timer (16-bit)
CNT	0x2015	Counter (16-bit)
DCNT	0x3015	Counter (32-bit)
INV	0x4016	Inversion - replacing the result of logical connections with the opposite
ANB	0x4007	«AND»-block: series connection of blocks
ORB	0x4008	«OR»-block: parallel connection of blocks
MPS	0x4009	Offset down the stack
MRD	0x402A	Read value from the stack
MPP	0x400A	Exit from the stack
SET	0x2024	Turning on latched output (setting the logical “1”)
RST	0x2004	Reset of the operand state
OUT	0x2002	Output coil - assignment to the output of the result of a logical expression
FEND	0x6003	End of main program
NOP	0x8011	Empty line in the program
P	0x6051	Addressing a jump point in a program or subprogram
I	0x6031	Addressing of an interruption point
END	0x6023	End of program
		Instructions for loops, transitions, subprogram
CJ	0x200D	Conditional jump - go to the specified program line

Phone: + 372 6559914,
e-mail: mail@smd.ee
url: https://smd.ee

CJP	0x280D	Conditional jump - go to the specified program line with rising edge polling (impulse)
CALL	0x200E	Calling subprogram
CALLP	0x280E	Calling subprogram with rising edge polling (impulse)
SRET	0x600F	End of subprogram
FOR	0x400B	Start of a loop FOR-NEXT
NEXT	0x400C	End of a loop FOR-NEXT
	Interruptions	
IRET	0x6010	End of interruption handler
EI	0x8012	Global interruptions enabling
DI	0x8013	Global interruptions disabling
	Data transfer and comparison	
CMP	0x2201	Comparison of numerical data
CMPP	0x2A01	Comparison of numerical data with rising edge polling (impulse)
DCMP	0x3201	Comparison of numerical data, 32-bit instruction
DCMPP	0x3A01	Comparison of numerical data, 32-bit instruction with rising edge polling (impulse)
ZCP	0x2202	Zone comparison of numerical data
ZCPP	0x2A02	Zone comparison of numerical data with rising edge polling (impulse)
DZCP	0x3202	Zone comparison of numerical data, 32-bit instruction
DZCPP	0x3A02	Zone comparison of numerical data, 32-bit instruction with rising edge polling (impulse)
MOV	0x2018	Data transfer
MOVP	0x2818	Data transfer with rising edge polling (impulse)
DMOV	0x3018	Data transfer, 32-bit instruction
DMOVP	0x3818	Data transfer, 32-bit instruction with rising edge polling (impulse)
BMOV	0x2038	Block data transfer
BMOVP	0x2838	Block data transfer with rising edge polling (impulse)
DBMOV	0x3038	Block data transfer, 32-bit instruction
DBMOVP	0x3838	Block data transfer, 32-bit instruction with rising edge polling (impulse)
FMOV	0x2058	Transferring data to multiple addresses
FMOVP	0x2858	Transferring data to multiple addresses with rising edge polling (impulse)
DFMOV	0x3058	Transferring data to multiple addresses, 32-bit instruction
DFMOVP	0x3858	Transferring data to multiple addresses, 32-bit instruction with rising edge polling (impulse)
XCH	0x220A	Data exchange
XCHP	0x2A0A	Data exchange with rising edge polling (impulse)

DXCH	0x320A	Data exchange, 32-bit instruction
DXCHP	0x3A0A	Data exchange, 32-bit instruction with rising edge polling (impulse)
	Arithmetic operations (integers)	
ADD	0x2208	Addition of numerical data
ADDP	0x2A08	Addition of numerical data with rising edge polling (impulse)
DADD	0x3208	Addition of numerical data, 32-bit instruction
DADDP	0x3A08	Addition of numerical data, 32-bit instruction with rising edge polling (impulse)
SUB	0x2228	Subtraction of numerical data
SUBP	0x2A28	Subtraction of numerical data with rising edge polling (impulse)
DSUB	0x3228	Subtraction of numerical data, 32-bit instruction
DSUBP	0x3A28	Subtraction of numerical data, 32-bit instruction with rising edge polling (impulse)
MUL	0x2248	Multiplication of numerical data
MULP	0x2A48	Multiplication of numerical data with rising edge polling (impulse)
DMUL	0x3248	Multiplication of numerical data, 32-bit instruction
DMULP	0x3A48	Multiplication of numerical data, 32-bit instruction with rising edge polling (impulse)
DIV	0x2268	Division of numerical data
DIVP	0x2A68	Division of numerical data with rising edge polling (impulse)
DDIV	0x3268	Division of numerical data, 32-bit instruction
DDIVP	0x3A68	Division of numerical data, 32-bit instruction with rising edge polling (impulse)
MOD	0x22E8	Remainder of the division
MODP	0x2AE8	Remainder of the division with rising edge polling (impulse)
DMOD	DMOD	Remainder of the division, 32-bit instruction
DMODP	DMODP	Remainder of the division, 32-bit instruction with rising edge polling (impulse)
INC	0x2037	Increment numerical data (increase by 1)
INCP	0x2837	Increment numerical data (increase by 1) with rising edge polling (impulse)
DINC	0x3037	Increment numerical data (increase by 1), 32-bit instruction
DINCP	0x3837	Increment numerical data (increase by 1), 32-bit instruction with rising edge polling (impulse)
DEC	0x2017	Decrement numerical data (decrease by 1)
DECP	0x2817	Decrement numerical data (decrease by 1) with rising edge polling (impulse)
DDEC	0x3017	Decrement numerical data (decrease by 1), 32-bit instruction

DDECP	0x3817	Decrement numerical data (decrease by 1), 32-bit instruction with rising edge polling (impulse)
WAND	0x2288	Logical multiplication of numerical data (operation "AND")
WANDP	0x2A88	Logical multiplication of numerical data (operation "AND") with rising edge polling (impulse)
DAND	0x3288	Logical multiplication of numerical data (operation "AND"), 32-bit instruction
DANDP	0x3A88	Logical multiplication of numerical data (operation "AND"), 32-bit instruction with rising edge polling (impulse)
WOR	0x22A8	Logical addition of numerical data (OR operation)
WORP	0x2AA8	Logical addition of numerical data (OR operation) with rising edge polling (impulse)
DOR	0x32A8	Logical addition of numerical data (OR operation), 32-bit instruction
DORP	0x3AA8	Logical addition of numerical data (OR operation), 32-bit instruction with rising edge polling (impulse)
WXOR	0x22C8	Logical operation "exclusive OR"
WXORP	0x2AC8	Logical operation "exclusive OR" with rising edge polling (impulse)
DXOR	0x32C8	Logical operation "exclusive OR", 32-bit instruction
DXORP	0x3AC8	Logical operation "exclusive OR", 32-bit instruction with rising edge polling (impulse)
NEG	0x2209	Logical negation
NEGP	0x2A09	Logical negation with rising edge polling (impulse)
DNEG	0x3209	Logical negation, 32-bit instruction
DNEGP	0x3A09	Logical negation, 32-bit instruction with rising edge polling (impulse)
ABS	0x2229	Absolute value
ABSP	0x2A29	Absolute value with rising edge polling (impulse)
DABS	0x3229	Absolute value, 32-bit instruction
DABSP	0x3A29	Absolute value, 32-bit instruction with rising edge polling (impulse)
SQR	0x2215	Square root calculation
SQRP	0x2A15	Square root calculation with rising edge polling (impulse)
DSQR	0x3215	Square root calculation, 32-bit instruction
DSQRP	0x3A15	Square root calculation, 32-bit instruction with rising edge polling (impulse)
POW	0x2216	Raising to a power
POWP	0x2A16	Raising to a power with rising edge polling (impulse)
DPOW	0x3216	Raising to a power, 32-bit instruction
DPOWP	0x3A16	Raising to a power, 32-bit instruction with rising edge polling (impulse)

Shift operations		
ROR	0x220B	Cycle shift to the right
RORP	0x2A0B	Cycle shift to the right with rising edge polling (impulse)
DROR	0x320B	Cycle shift to the right, 32-bit instruction
DRORP	0x3A0B	Cycle shift to the right, 32-bit instruction with rising edge polling (impulse)
ROL	0x222B	Cycle shift to the left
ROLP	0x2A2B	Cycle shift to the left with rising edge polling (impulse)
DROL	0x322B	Cycle shift to the left, 32-bit instruction
DROLP	0x3A2B	Cycle shift to the left, 32-bit instruction with rising edge polling (impulse)
Data processing		
ZRST	0x2203	Group reset of operands in a given range
ZRSTP	0x2A03	Group reset of operands in a given range with rising edge polling (impulse)
DECO	0x2211	Decoder 8 → 256 bit
DECOP	0x2A11	Decoder 8 → 256 bit with rising edge polling (impulse)
ENCO	0x2212	Encoder 256 → 8 bit
ENCOP	0x2A12	Encoder 256 → 8 bit with rising edge polling (impulse)
SUM	0x2213	Sum of single bits in the register
SUMP	0x2A13	Sum of single bits in the register with rising edge polling (impulse)
DSUM	0x3213	Sum of single bits in the register, 32-bit instruction
DSUMP	0x3A13	Sum of single bits in the register, 32-bit instruction with rising edge polling (impulse)
BON	0x2214	Check a bit state with setting an output
BONP	0x2A14	Check a bit state with setting an output with rising edge polling (impulse)
DBON	0x3214	Check a bit state with setting an output, 32-bit instruction
DBONP	0x3A14	Check a bit state with setting an output, 32-bit instruction with rising edge polling (impulse)
FLT	0x220C	Convert integer to floating point
FLTP	0x2A0C	Convert integer to floating point with rising edge polling (impulse)
DFLT	0x320C	Convert integer to floating point, 32-bit instruction
DFLTP	0x3A0C	Convert integer to floating point, 32-bit instruction with rising edge polling (impulse)
Floating point operations		
DECMP	0x220F	Comparison of floating point numbers
DECMPP	0x2A0F	Comparison of floating point numbers with rising edge polling (impulse)
DEZCP	0x2210	Zone floating point comparison

DEZCPP	0x2A10	Zone floating point comparison with rising edge polling (impulse)
DEADD	0x2217	Addition of floating point numbers
DEADDP	0x2A17	Addition of floating point numbers with rising edge polling (impulse)
DESUB	0x2237	Subtraction of floating point numbers
DESUBP	0x2A37	Subtraction of floating point numbers with rising edge polling (impulse)
DEMUL	0x2257	Multiplication of floating point numbers
DEMULP	0x2A57	Multiplication of floating point numbers with rising edge polling (impulse)
DEDIV	0x2277	Floating point numbers division
DEDIVP	0x2A77	Floating point numbers division with rising edge polling (impulse)
DESQR	0x2218	Square root in floating point format
DESQRP	0x2A18	Square root in floating point format with rising edge polling (impulse)
DEPOW	0x2297	Raising to a power in floating point format
DEPOWP	0x2A97	Raising to a power in floating point format with rising edge polling (impulse)
INT	0x220D	Converting a floating point number to an integer
INTP	0x2A0D	Converting a floating point number to an integer with rising edge polling (impulse)
DINT	0x320D	Converting a floating point number to an integer, 32-bit instruction
DINTP	0x3A0D	Converting a floating point number to an integer, 32-bit instruction with rising edge polling (impulse)
	Time and PWM	
TRD	0x2219	Reading the current value of the real-time clock
TRDP	0x2A19	Reading the current value of the real-time clock with rising edge polling (impulse)
TWR	0x221A	Changing the value of a real-time clock
TWRP	0x2A1A	Changing the value of a real-time clock with rising edge polling (impulse)
PWM	0x220E	Pulse width-modulation (PWM) output
	Contact type logical operations	
LD&	0x4204	Contact is closed if S1 & S2 ≠ 0
DLD&	0x5204	Contact is closed if S1 & S2 ≠ 0, 32-bit instruction
LD|	0x4224	Contact is closed if S1 | S2 ≠ 0
DLD|	0x5224	Contact is closed if S1 | S2 ≠ 0, 32-bit instruction
LD^	0x4244	Contact is closed if S1 ^ S2 ≠ 0
DLD^	0x5244	Contact is closed if S1 ^ S2 ≠ 0, 32-bit instruction
AND&	0x4205	Serial contact closed if S1 & S2 ≠ 0
DAND&	0x5205	Serial contact closed if S1 & S2 ≠ 0, 32-bit instruction
AND|	0x4225	Serial contact closed if S1 | S2 ≠ 0

DAND|	0x5225	Serial contact closed if $S1 \mid S2 \neq 0$, 32-bit instruction
AND^	0x4245	Serial contact closed if $S1$ ^ $S2 \neq 0$
DAND^	0x5245	Serial contact closed if $S1$ ^ $S2 \neq 0$, 32-bit instruction
OR&	0x4206	Parallel contact closed if $S1$ & $S2 \neq 0$
DOR&	0x5206	Parallel contact closed if $S1$ & $S2 \neq 0$, 32-bit instruction
OR|	0x4226	Parallel contact closed if $S1 \mid S2 \neq 0$
DOR|	0x5226	Parallel contact closed if $S1 \mid S2 \neq 0$, 32-bit instruction
OR^	0x4246	Parallel contact closed if $S1$ ^ $S2 \neq 0$
DOR^	0x5246	Parallel contact closed if $S1$ ^ $S2 \neq 0$, 32-bit instruction
	Contact type comparison operations	
LD=	0x4264	Contact is closed if $S1 = S2$
DLD=	0x5264	Contact is closed if $S1 = S2$, 32-bit instruction
LD>	0x4284	Contact is closed if $S1 > S2$
DLD>	0x5284	Contact is closed if $S1 > S2$, 32-bit instruction
LD<	0x42A4	Contact is closed if $S1 < S2$
DLD<	0x52A4	Contact is closed if $S1 < S2$, 32-bit instruction
LD<>	0x42C4	Contact is closed if $S1 \neq S2$
DLD<>	0x52C4	Contact is closed if $S1 \neq S2$, 32-bit instruction
LD<=	0x42E4	Contact is closed if $S1 \leq S2$
DLD<=	0x52E4	Contact is closed if $S1 \leq S2$, 32-bit instruction
LD>=	0x4304	Contact is closed if $S1 \geq S2$
DLD>=	0x5304	Contact is closed if $S1 \geq S2$, 32-bit instruction
AND=	0x4265	Serial contact closed if $S1 = S2$
DAND=	0x5265	Serial contact closed if $S1 = S2$, 32-bit instruction
AND>	0x4285	Serial contact closed if $S1 > S2$
DAND>	0x5285	Serial contact closed if $S1 > S2$, 32-bit instruction
AND<	0x42A5	Serial contact closed if $S1 < S2$
DAND<	0x52A5	Serial contact closed if $S1 < S2$, 32-bit instruction
AND<>	0x42C5	Serial contact closed if $S1 \neq S2$
DAND<>	0x52C5	Serial contact closed if $S1 \neq S2$, 32-bit instruction
AND<=	0x42E5	Serial contact closed if $S1 \leq S2$
DAND<=	0x52E5	Serial contact closed if $S1 \leq S2$, 32-bit instruction
AND>=	0x4305	Serial contact closed if $S1 \geq S2$
DAND>=	0x5305	Serial contact closed if $S1 \geq S2$, 32-bit instruction
OR=	0x4266	Parallel contact closed if $S1 = S2$
DOR=	0x5266	Parallel contact closed if $S1 = S2$, 32-bit instruction

OR>	0x4286	Parallel contact closed if S1 > S2
DOR>	0x5286	Parallel contact closed if S1 > S2, 32-bit instruction
OR<	0x42A6	Parallel contact closed if S1 < S2
DOR<	0x52A6	Parallel contact closed if Parallel contact closed if S1 < S2, 32-bit instruction
OR<>	0x42C6	Parallel contact closed if S1 ≠ S2
DOR<>	0x52C6	Parallel contact closed if S1 ≠ S2, 32-bit instruction
OR<=	0x42E6	Parallel contact closed if S1 ≤ S2
DOR<=	0x52E6	Parallel contact closed if S1 ≤ S2, 32-bit instruction
OR>=	0x4306	Parallel contact closed if S1 ≥ S2
DOR>=	0x5306	Parallel contact closed if S1 ≥ S2, 32-bit instruction
Stepper motor control		
SPIN	0x2207	Start preset movement
SPINP	0x2A07	Start preset movement with rising edge polling (impulse)
TORQUE	0x2227	Apply the set currents to the motor
TORQUEP	0x2A27	Apply the set currents to the motor with rising edge polling (impulse)
HSTOP	0x2247	Switch to hold mode immediately
HSTOPP	0x2A47	Switch to hold mode immediately with rising edge polling (impulse)
HHIZ	0x2267	Deenergize motor phases immediately (the shaft rotates freely)
HHIZP	0x2A67	Deenergize motor phases immediately (the shaft rotates freely) with rising edge polling (impulse)
SSTOP	0x2287	Decelerate until full stop and switch to hold mode
SSTOPP	0x2A87	Decelerate until full stop and switch to hold mode with rising edge polling (impulse)
SHIZ	0x22A7	Decelerate until full stop and deenergize motor phases (the shaft rotates freely)
SHIZP	0x2AA7	Decelerate until full stop and deenergize motor phases (the shaft rotates freely) with rising edge polling (impulse)

Appendix C. Examples of user programs

Example 1. Usage of RUN command

LDP	X0			*;catch the front of the pulse at the input X0 (button)*
DMOV	K8	D359		*;set minimum speed 8 pps*
DMOV	K120000	D357		*;set maximum speed 120000 pps*
FMOV	K30000	D361	K2	*;set acceleration and deceleration 30000 pps^2*
MOV	K3	D366		*;microstepping 1/8 (refer to the description of the instruction SPIN)*
MOV	K1	D374		*;direction – forward*
DMOV	K6000	D377		*;set fullstep speed 6000 pps/sec*
MOV	K1	D379		*;enable to turn to the fullstep mode when reach fullstep speed*
MOV	K0	D376		*;command RUN*
FMOV	K1500	D367	K2	*;acceleration and deceleration currents 1500 mA*
MOV	K1200	D369		*;constant speed current 1200 mA*
MOV	K600	D370		*;holding current 600 mA*
TORQUE				*;apply current values*
FMOV	K0	D380	K3	*;no error response, errors reset, use ;MIN_SPEED*
SPIN				*;start motion*
LDP	X1			*;catch the front of the pulse at the input X1 (button)*
SSTOP				*;stop according to the preset DEC and turn to the holding mode*
LDP	X2			*;catch the front of the pulse at the input X2 (button)*
SHIZ				*;stop according to the preset DEC and turn to the HiZ mode*
LDP	X3			*;catch the front of the pulse at the input X3 (button)*
HSTOP				*;immediately turn to the holding mode*
LDP	X4			*;catch the front of the pulse at the input X4 (button)*
HHIZ				*;immediately turn to the HiZ mode*
END				*;end of the program*

Example 2. Usage of commands MOVE, GOTO, GOHOME

LD	M0			*;to skip the initialization section, check the condition M0*
CJ	P1			*;and jump to the line marked P1*
LDP	M108			*;M108 leading edge after initialization only*
DMOV	K120000	D357		*;set the maximum speed 120000 pps*
FMOV	K30000	D361	K2	*;set the acceleration and deceleration 30000pps^2*
MOV	K3	D366		*;microstepping 1/8 (refer to the description of the instruction SPIN)*
DMOV	K6000	D377		*;set fullstep speed 6000 pps/sec*
MOV	K1	D379		*;enable to turn to the fullstep mode when reach fullstep speed*
FMOV	K1500	D367	K2	*;acceleration and deceleration currents 1500 mA*
MOV	K1200	D369		*;constant speed current 1200 mA*
MOV	K600	D370		*;holding current 600 mA*
TORQUE				*;apply current values*
FMOV	K0	D380	K2	*;no error response, errors reset*

MOV	K1	D382	*;use automatic calculation of start and final speed*
DMOV	K0	D363	*;zero the current position*
SET	M0		*;turn on the driver initialization bypass condition*
P	1		*;transition mark*
LDP	X0		*;catch the front of the pulse at the input X0 (button)*
AND&	D371	K3	*;only if the motor is in the HiZ or Hold mode*
DMOV	K10000	D372	*;move 10000 microsteps*
MOV	K1	D374	*;in the forward direction*
MOV	K1	D376	;is *performed by the MOVE command*
SPIN			*;start motion*
LDP	X1		*;catch the front of the pulse at the input X1 (button)*
AND&	D371	K3	*;only if the motor is in the HiZ or Hold mode*
DMOV	K100000	D372	*;moving to a position with coordinate 100000*
MOV	K2	D376	;is *performed by the GOTO command*
SPIN			*;start motion*
LDP	X2		*;catch the front of the pulse at the input X2 (button)*
AND&	D371	K3	*;only if the motor is in the HiZ or Hold mode*
MOV	K0	D374	*;movement to the "0" position in the backward direction*
MOV	K4	D376	;is *performed by the GOHOME command*
SPIN			*;start motion*
LDP	X3		*;catch the front of the pulse at the input X3 (button)*
SSTOP			*;stop according to the preset DEC and turn to the holding mode*
LDP	X4		*;catch the front of the pulse at the input X4 (button)*
SHIZ			*;stop according to the preset DEC and turn to the HiZ mode*
LDP	X5		*;catch the front of the pulse at the input X5 (button)*
HSTOP			*;immediately turn to the holding mode*
LDP	X6		*;catch the front of the pulse at the input X6 (button)*
HHIZ			*;immediately turn to the HiZ mode*
END			*;end of the program*

Example 3. Usage of commands GOUNTIL_SLOWSTOP and RELEASE

Using the GOUNTIL_SLOWSTOP and RELEASE commands as an example of moving to the origin position along the positive limit switch (see Fig. 34).

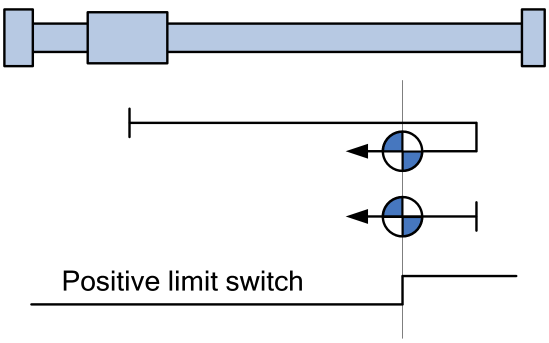


Fig. 34 – Move to the origin

LD	M0		*;to skip the initialization section, check the condition M0*
CJ	P1		*;and jump to the line marked P1*
LDP	M108		*;M108 leading edge after initialization only*

FMOV	K30000	D361	K2	*;set the acceleration and deceleration 30000pps^2*
MOV	K3	D366		*;microstepping 1/8 (refer to the description of the instruction SPIN)*
DMOV	K6000	D377		*;set fullstep speed 6000 pps/sec*
MOV	K1	D379		*;enable to turn to the fullstep mode when reach fullstep speed*
FMOV	K1500	D367	K2	*;acceleration and deceleration currents 1500 mA*
MOV	K1200	D369		*;constant speed current 1200 mA*
MOV	K600	D370		*;holding current 600 mA*
TORQUE				*;apply current values*
FMOV	K0	D380	K2	*;no error response, errors reset*
MOV	K1	D382		*;use automatic calculation of start and final speed*
MOV	K7	D375		*;the switch limit is connected to the input IN7*
SET	M0			*;turn on the driver initialization bypass condition*
P	1			*;transition mark*
LDP	X0			*;catch the front of the pulse at the input X0 (button)*
AND&	D371	K3		*;only if the motor is in the HiZ or Hold mode*
DMOV	K20000	D357		*;set the maximum speed 20000 pps*
MOV	K5	D376		*;command GOUNTIL_SLOWSTOP*
MOV	K1	D374		*;direction – forward*
SPIN				*;start motion*
SET	M1			*;set the flag to start the first stage*
LD	M1			*;wait when the limit switch is activated at the first stage*
AND&	D371	K2		*;and the motor stops*
RST	M1			*;reset the flag of the first stage*
DMOV	K1000	D357		*;decrease speed*
MOV	K9	D376		*;movement in the opposite direction until the limit switch*
MOV	K0	D374		*;opens*
SPIN				*;start motion*
SET	M2			*;and move on to the second stage*
LD	M2			*;waiting for the limit switch to open and the motor to stop*
AND&	D371	K2		*;at the second stage*
RST	M2			*;reset the flag of the second stage*
DMOV	K0	D363		*;and reset the current position, it becomes the origin now*
LDP	X1			*;catch the front of the pulse at the input X1 (button)*
SSTOP				*;stop according to the preset DEC and turn to the holding mode*
LDP	X2			*;catch the front of the pulse at the input X2 (button)*
SHIZ				*;stop according to the preset DEC and turn to the HiZ mode*
LDP	X3			*;catch the front of the pulse at the input X3 (button)*
HSTOP				*;immediately turn to the holding mode*
LDP	X4			*;catch the front of the pulse at the input X4 (button)*
HHIZ				*;immediately turn to the HiZ mode*
FEND				*;end of the main program*
I	1007			*;interruption handler for the input IN7 (required for GOUNTIL _... and ... RELEASE commands, may be left empty)*
IRET				*;return to the main program*
END				*;end of the program*

Tallinn Science Park Tehnopol,
Akadeemia tee 21/6, 12618, Tallinn, Estonia
Phone: + 372 6559914,
e-mail: mail@smd.ee
url: https://smd.ee

Appendix D. Code of the service program "Stepper Motor Speed Control"

LD	M0			*;initialization bypass condition*
CJ	P1			
LDP	M108			*;initialization part*
MPS				
LD=	D320	K6		*; D320 stores microstepping value*
OR>	D320	K8		
OR<	D320	K0		
ANB				
MOV	K0	D320		
MRD				
MOV	D320	D366		
MOV	D320	D0		*;D0 –the service register for visualization of microstepping*
MOV	K0	D2		
MRD				
AND>	D0	K6		*;as the controller doesn't support microstepping 1/64,*
DEC	D0			*;skip this value*
MRD				
DECO	D0	Y0	K3	*;visualization of microstepping on the outputs scale*
MOV	K0	D379		*;initial setup of the stepper motor driver*
MOV	K0	D376		
MOV	K250	D359		
MOV	K0	D380		
MRD				
LD<	D321	K0		*;D321 stores control method data:*
OR>	D321	K2		*;potentiometer/buttons/encoder*
ANB				
MOV	K0	D321		
MRD				
SET	M0			
MOV	D321	Y10		*;visualization of the control method*
MRD				
AND<>	A0	D321		
MPS				
AND=	D321	K2		*;if an* ***encoder*** *is selected, then the peripherals*
MOV	K12	D355		*;of the controller must be set accordingly*
MOV	D321	A0		
RST	M108			*;and restart the program*
MPP				
AND<>	D321	K2		
MOV	K0	D355		
MOV	D321	A0		
RST	M108			
MRD				
MUL	D353	K10	D4	*;data of the potentiometer* ***1***
DIV	D4	K27	D4	
FMOV	D4	D367	K3	*;set acceleration, deceleration and constant speed current*
DIV	D4	K2	D370	*;holding current – 50% of work current*
TORQUE				
MRD				

AND	X7		
MOV	K1	D374	
MRD			
ANI	X7		
MOV	K0	D374	
MRD			
DLD<	D322	K250	*;D322, D323 speed set by buttons*
DOR>	D322	K120000	
ANB			
DMOV	K250	D322	
MRD			
DMOV	D322	D5	
MRD			
DLD<	D324	K250	*;D324, D325 speed set by the encoder*
DOR>	D324	K120000	
ANB			
DMOV	K250	D324	
MRD			
DMOV	D324	D15	
MRD			
DMOV	K0	C64	
DMOV	C64	D7	
MPP			
ANI	X4		
HSTOP			
LD	X2		
OUT	M102		
EI			;enable *interruptions, the end of the initialization block*
P	1		
LD	X4		
AND&	D371	HFE	
HHIZ			
LDI	X4		
MPS			
AND	M102		
AND	X3		
AND&	D371	K3	
CALL	P10A0		
SPIN			
MPP			
LDI	X3		
ANB			
AND&	D371	HFD	
HSTOP			
LDI	M102		
AND	X3		
ANI	X4		
AND&	D371	H15	
SSTOP			
LD	M109		*;if there was an error and the ERR indicator was on -*
TMR	T0	K10	*;start the timer to turn it off*

AND	T0		
RST	M109		
LD<>	A0	K1	
CJ	P19		
LD=	A0	K1	
MPS			
ANDP	X0		
DADD	D5	D354	D5
MPS			
DAND>	D5	K120000	
DMOV	K120000	D5	
MPP			
DMOV	D5	D322	
MPP			
ANDP	X1		
DSUB	D5	D354	D5
MPS			
DAND<	D5	K250	
DMOV	K250	D5	
MPP			
DMOV	D5	D322	
P	19		
LD<>	A0	K2	
CJ	P20		
LD=	A0	K2	
MPS			
DSUB	C64	D7	D9
DADD	D7	D9	D7
DCMP	D9	K0	M1
MRD			
AND	M1		
MOV	D354	D1	
DMUL	D9	D1	D11
DADD	D11	D15	D15
MPP			
AND	M3		
MOV	D354	D1	
ABS	D9		
DMUL	D9	D1	D11
DADD	D15	D11	D15
LD	M1		
OR	M3		
MPS			
DAND<	D15	K250	
DMOV	K250	D15	
MRD			
DAND>	D15	K120000	
DMOV	K120000	D15	
MPP			
DMOV	D15	D324	
P	20		

FEND				*;end of the main program*
P	10			*;subprogram of setting the speed by the potentiometer*
LD	M108			
MOV	K0	D2		
MOV	D354	D1		
MPS				
AND=	D1	K0		
MOV	K1	D1		
MRD				
DMUL	D1	K29	D13	
MRD				
DAND<	D13	K250		
DMOV	K250	D13		
MPP				
DMOV	D13	D357		
CALL	P0			
SRET				
P	11			*; subprogram of setting the speed by buttons*
LD	M108			
DMOV	D322	D357		
CALL	P0			
SRET				
P	12			*; subprogram of setting the speed by the encoder*
LD	M108			
DMOV	D324	D357		
CALL	P0			
SRET				
P	0			*;subprogram of updating acceleration and deceleration*
LD	M108			
MOV	D352	D3		
MPS				
AND=	D3	K0		
MOV	K1	D3		
MPP				
MUL	D3	K14	D361	
MOV	D361	D362		
SRET				
I	50			*;500 ms interruption to update of the currents*
LD	M108			
MUL	D353	K10	D4	
DIV	D4	K27	D4	
FMOV	D4	D367	K3	
DIV	D4	K2	D370	
TORQUE				
IRET				
I	10			*;interruption with a period of 100 ms to update the speed*
LDI	X6			
AND	X2			
AND	M102			
AND	X3			
ANI	X4			

```
BON     D371    M60     K6
ANI     M60
CALL    P10A0
SPIN
IRET
I       1005                    ;interruption from the input IN5
LD      M105
INC     D320
MPS
AND=    D320    K6
INC     D320
MRD
AND=    D320    K9
MOV     K0      D320
INC     D321
MPS
AND=    D321    K3
MOV     K0      D321
MRD
MOV     D321    Y10
MOV     D321    A0
MRD
AND=    D321    K2
MOV     K12     D355
RST     M108
MPP
AND<>   D321    K2
MOV     K0      D355
RST     M108
MRD
MOV     D320    D0
MOV     D320    D366
MRD
AND>    D0      K6
DEC     D0
MPP
DECO    D0      Y0      K3
IRET
I       1004                    ;interruption from the input IN4
LD      M104
HHIZ
LDI     M104
MPS
AND     X2
AND     X3
AND&    D371    K3
CALL    P10A0
SPIN
MPP
LDI     X2
ORI     X3
```

```
ANB
HSTOP
IRET
I       1003                ;interruption from the input IN3
LDI     M103
ANI     X4
HSTOP
LD      M103
AND     X2
ANI     X4
AND&    D371    K3
CALL    P10A0
SPIN
IRET
I       1002                ;interruption from the input IN2
LDI     M102
AND     X3
ANI     X4
SSTOP
LD      M102
AND     X3
ANI     X4
AND&    D371    K3
CALL    P10A0
SPIN
IRET
I       1007                ;interruption from the input IN7
LD&     D371    H1C
MPS
AND     M107
AND=    D374    K0
SSTOP
MPP
ANI     M107
AND=    D374    K1
SSTOP
LD      M107
MOV     K1      D374
AND     X2
AND     X3
ANI     X4
AND&    D371    K3
CALL    P10A0
SPIN
LDI     M107
MOV     K0      D374
AND     X2
AND     X3
ANI     X4
AND&    D371    K3
CALL    P10A0
```

SPIN			
IRET			
I	2000		*;interruption when a driver error occurs*
LD&	D381	K1	
MOV	K0	D381	
SET	M109		
MOV	K0	T0	
IRET			
END			

Tallinn Science Park Tehnopol,
Akadeemia tee 21/6, 12618, Tallinn, Estonia
Phone: + 372 6559914,
e-mail: mail@smd.ee
url: https://smd.ee

Appendix E. The lifetime of the fronts of the operands M and Y

All of the information below is valid for operands M and Y, both for the leading and trailing edges.

Example 1

Consider the lifespan of the leading edge of the operand M0, with guaranteed passage of the start of life point on the next scan.

Line	Instruction	Operands, order number		M0 front lifetime, scan		Explanation
		1	2	1	2	
1	LDP	M108		1		The front edge of the M108 exists only on the first scan of the program
2	ZRST	D0	D2			Setting to zero D0…D2
3	LD	M108				
4	OUT	M0		2		The beginning of the life of the leading edge of the operand M0 is in the first scan and the end is in the second
5	P	1				
6	LD	M0				
7	CALL	P0				The current state M0 is saved for every jump to the subprogram P0.
8	LDP	M0				It will work only once, since the condition of re-passing through the beginning point of the front of the operand M0 in the next program scan is satisfied.
9	INC	D0				The result of the program is D0 = 1.
10	FEND					
11	I	0				The first interruption occurs after line 2 at the first scan. At this moment the leading edge of M0 is absent. The second interruption is processed after line 6. The presence of a leading edge at M0 is transmitted to the interruption processing, so the result of the work is D2 = 1.
13	LDP	M0				
14	INC	D2				
15	IRET					
16	P	0				
17	LDP	M0				The condition is met at the first scan. The condition is not met at the second scan.
18	INC	D1				The result is D1 = 1.
19	SRET					
20	END					

- interruption
- existence of a front of the operand

Example 2

Consider the lifespan of the leading edge of the operand M0, in the absence of passing the start point of life on the next scan.

Line	Instruction	Operands, order number		M0 front lifetime, scan			Explanation
		1	2	1	2	3	
1	LDP	M0					
2	CJ	P1					Bypassing the start point of life of the front.
3	LDP	M108					The leading edge of the M108 exists only at the firdt scan of the program.
4	ZRST	D0	D2				Setting to zero D0…D2.
5	LD	M108					
6	OUT	M0					The start of the life of the leading edge of the operand M0 in the first scan. The next pass of this point will be only in the third scan.
7	P	1					
8	LDP	M0					It will work twice, since the start point of the life of the front was skipped by the command handler in the second scan, and the lifetime was increased until the END / FEND command was received.
9	INC	D0					The result of the program is D0 = 2.
10	END						The lifetime of the leading edge of the operand M0 is increased to the end of the scan due to the absence of passage of the start point of the lifetime of the front M0.

- existence of a front of the operand

If a single pass between points 7 ... 9 is required, then, for example, the optional relay contact M1 can be used. The program will be changes as below:

1	LDP	M0		
2	CJ	P1		
3	LDP	M108		
4	ZRST	D0	D2	
5	ZRST	M1		*;Initialization M1*
6	LD	M108		
7	OUT	M0		
8	P	1		
9	LDP	M0		
10	ANI	M1		*;Additional condition*
11	INC	D0		;The result of the program is *D0 = 1*
12	SET	M1		*;Locking*
13	END			