Process automation
Programmable Logic Controllers (PLCs)
Programming – part 1

BS UNI studies, Fall semester 2024/2025

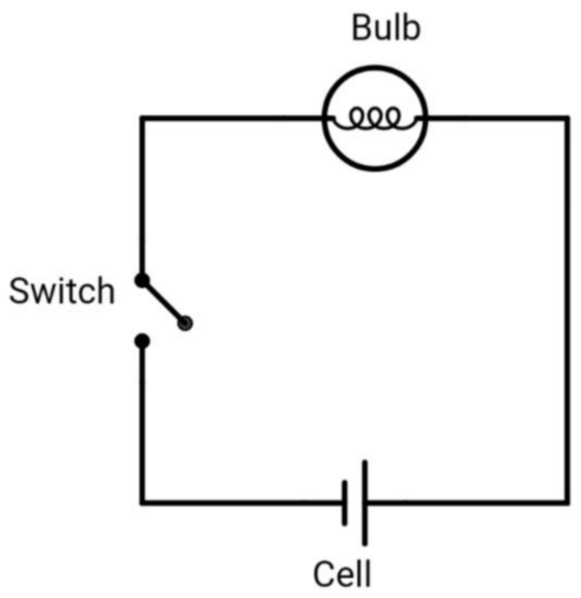
Octavian M. Machidon
octavian.machidon@fri.uni-lj.si

Outline

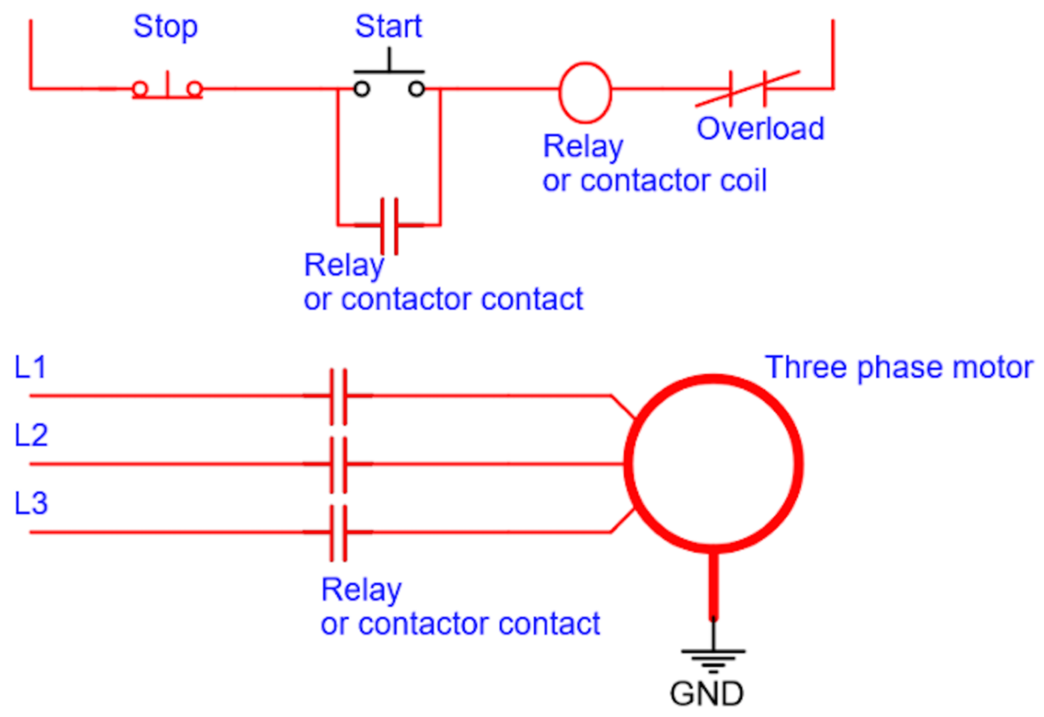
- **Introduction to PLCs and Programming**
  - Overview and significance of PLCs.
  - Early challenges and development of programming standards.
- **IEC 61131-3 Standard**
  - Core programming languages (Ladder, Function Block).
  - Typical program structures, data types, and addressing.
- **Programming Objects and Initialization**
  - Programs, function blocks, tasks.
  - Start-up procedures, memory management, and data persistence.
- **Ladder Diagrams**
  - Basics, examples, and limitations.
  - Extensions: advanced logic, timers, and counters.

Example: control without a PLC

Switch and Lightbulb



Electromotor control



Evolution of standards for PLC programming

Before (Pre-IEC 61131)

- Ladder diagrams were not well-structured.
- Complex high-level languages were used for specific applications, aiming to simplify the programming process.

IEC 61131 Standard

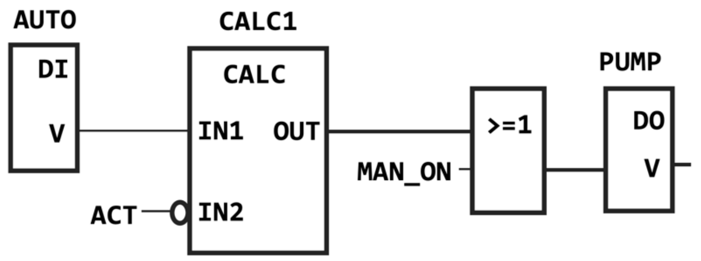
- Established in 1993, with the current 3rd edition released in 2013, describing an object-oriented approach to programming.
- It aimed to standardize various PLC programming concepts.
- Part 3 of IEC 61131 defines the structures and programming languages used in PLC programming, including:
    - **Ladder diagram (LD)**
    - **Function block diagram (FBD)**
    - **Sequential function chart (SFC)**
    - **Structured text (ST)**
    - **Instruction list (IL)** (marked as obsolete in the 3rd edition, note regarding TwinCAT)

Now (Current)

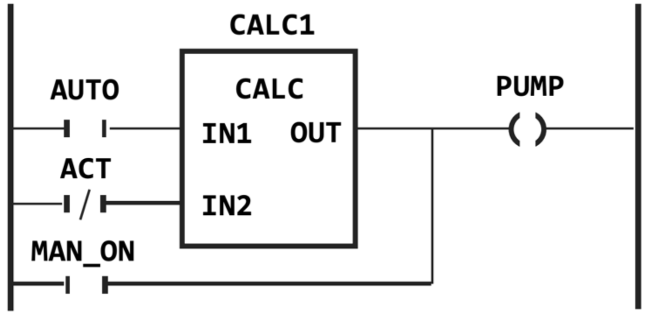
• Programs are almost transferable between different manufacturers or product series, and programming concepts are definitely transferable.

IEC 61131-3 standard

Function Block Diagram (FBD)



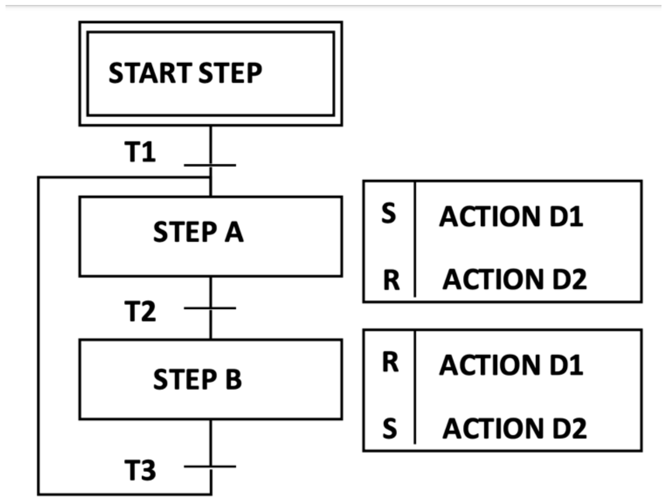
Ladder Diagram (LD)



Instruction List (IL)

```
A: LD   %IX1 (* PUSH BUTTON *)
   ANDN %MX5 (* NOT INHIBIT *)
   ST   %QX2 (* FAN ON *)
```

Sequential Function Chart (SFC)



Structured Text (ST)

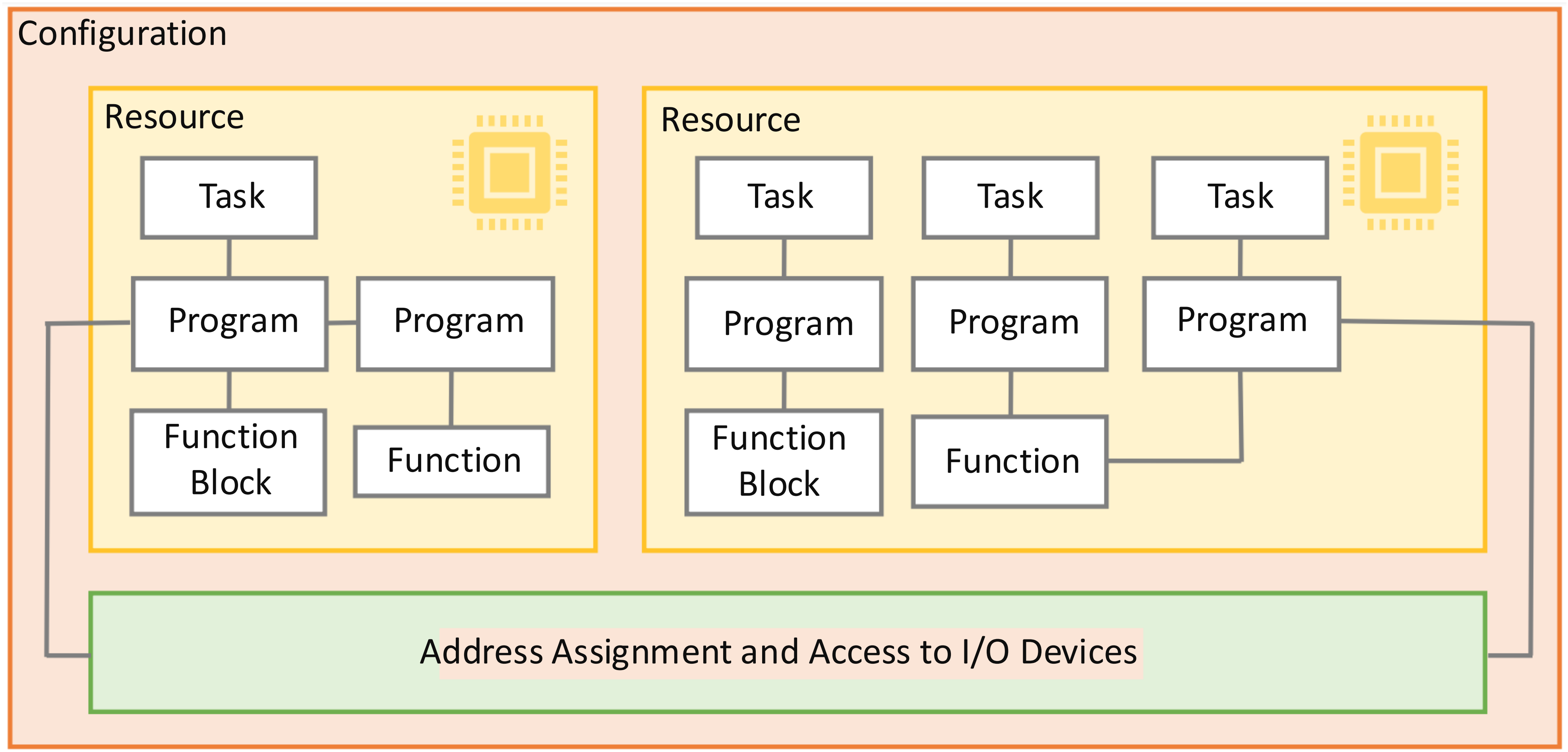
```
VAR CONSTANT
X : REAL := 53.8;Z : REAL; END_VAR
VAR aFB, bFB :  FB_type; END_VAR

bFB(A:=1, B:='OK');
Z := X - INT_TO_REAL (bFB.OUT1);
IF Z > 57.0 THEN aFB(A:=0, B:="ERR");
ELSE  aFB(A:=1, B:="Z is OK");
END_IF
```

Typical programming structures and data types

- **Digital Inputs**: %I*
- **Digital Outputs**: %Q*
- **Memory**: %M*
- **Timers**
  - Measures time when a condition is met. Once the set time is reached, it triggers its contact.
- **Counters**
  - Counts upwards or downwards when a condition is met.
- **Binary Variable Grouping**
  - Byte (8-bit), Word (16-bit), Double Word (32-bit)
- **Integer Variables** (16- or 32-bit)
- **Floating-Point Numbers** (e.g., 3.14, 1.64e+009)
- **Strings** (‘Hello!’)
- **Time** (T#12h34m15s)
- **Date** (D#2023-07-23)

Programming objects - model

- The standard defines a model for allocating resources to individual objects



Program objects

- Objects that Contain Program Code for PLC – POU (program organization unit)
- **Program – PRG**
  - Organizational unit at the highest level.
  - Every project needs at least one program (typically MAIN)
  - A program can contain calls to other programs, function blocks, and functions.
  - You can call a program from another program or function block, but not from a function.
  - The order of program calls in a project is defined in the task object. The task initializes the program.
  - When the program finishes, the values of the variables are retained until the next call of the program.
- **Function Block – FB**
  - Can have multiple inputs (**VAR_INPUT**) and outputs (**VAR_OUTPUT**).
  - Has its own memory for internal variables (**VAR**, **VAR_STAT**).
  - Internal and output variables retain their value until the next call, meaning that a function block returns different values on successive calls with the same input arguments.
  - A function block is always called through instances, i.e., copies. Each instance has its own copy of the variables.
  - The most common type of POU.
- **Function – FUN**
  - Returns exactly one element (can be an array or structure).
  - Can have multiple inputs (**VAR_INPUT**) and outputs (**VAR_OUTPUT**).
  - Does not have memory, so the values of variables between successive calls are not retained, meaning the same input parameters will always yield the same output.
  - **Note:** It is still possible to use static variables (**VAR_STAT**) in certain cases.

Program objects

- Every program object has two parts:
- **Declarations**:
    - Variables that define the interface:
        - Input: VAR_INPUT
        - Output: VAR_OUTPUT
        - Input/Output: VAR_IN_OUT
    - Local variables:
        - VAR
        - VAR_TEMP (used only in PRG and FB): initialized with each call of the block.
        - VAR_STAT (used only in FUN and FB): initialized only on the first call and remains for the entire lifetime (same variable instance even with multiple calls of FUN or FB instances).
    - Global variables (only in GVL, without implementation part): VAR_GLOBAL
- **Implementation**:
    - Program code in the chosen IEC language that implements a specific functionality.

Startup

- Initial startup *("original reset")*:
  - All variables are reset to their initial values, and the project on the controller is also reset.
- Cold startup *("cold reset")*:
  - All variables are reset to their initial values, except those set as PERSISTENT or RETAIN.

Addressing

- Syntax:

%	type	size	number	.	number

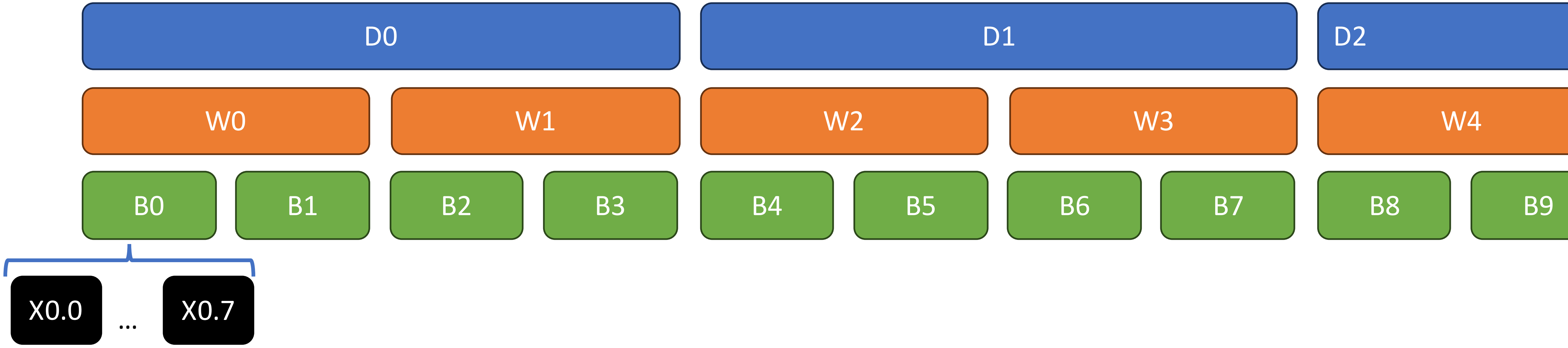
Input	I
Output	Q
Memory	M

X	1 bit
B	8 bits
W	16 bits
D	32 bits

- Examples
  - %IX1.0: Input bit 0 in byte with address 1 (1 bit)
  - %QB2: Output byte at address 2 (8 bits)
  - %MW4: Memory data, 5th consecutive word (16 bits)
  - %QD24: Output double word, 25th consecutive word (32 bits)

Addressing – overlapping

- In memory, bits, bytes, words, and double words are arranged as follows:



- **Warning:** If you write something to %MW0 and then to %MB1, you have written to the overlapping area in memory.
- **Tip:** In TwinCAT, it is recommended to use automatic address assignment with wildcard symbols: %I*, %Q*, %M*. This allows optimization of access and flexibility in address space allocation.

Data types

Data type	Lower limit	Upper limit	Size (bits)
BIT	FALSE (0)	TRUE (1)	1
BOOL	FALSE (0)	TRUE (1)	8
BYTE	0 (decimal)	255	8
	2#0000_0000 (binary)	2#1111_1111 (binary)	
	8#000 (octal)	8#377 (octal)	
	16#00 (hexadecimal)	16#FF (hexadecimal)	
WORD	0	65535	16
DWORD	0	4294967295	32
LWORD	0	2^64 – 1	64
TIME	0 = T#0ms	4294967295 = T#49d17h2m47s295ms	32
LTIME	0 = T#0ms	T#213503d23h34m33s709ms551us615ns	64
DATE	0 = D#1970-01-01	4294967295 = D#2106-02-07	32
DATE_AND_TIME	0 = DT#1970-01-01-0:0:0	4294967295 = DT#2106-02-07-06:28:15	32

Data types - continued

Data type	Lower limit	Upper limit	Size (bits)
SINT	-128	127	8
USINT	0	255	8
INT	-32768	32767	16
UINT	0	65535	16
DINT	-2147483648	2147483647	32
UDINT	0	4294967295	32
LINT	-2^63	2^63 - 1	64
ULINT	0	2^64-1	64
REAL	-3.402823e+38	3.402823e+38	32
LREAL	-1.7976931348623158e+308	1.7976931348623158e+308	64

Check out also the types STRING, WSTRING, ARRAY and ANY.

Ladder diagrams: standard

- **The most used language for PLC**
- Graphical language that enables an easy transition from electrical schematics to programming.
- **Standard 61131-3**
- **Vertical Rails**:
  - The foundation of the ladder diagram.
  - The left rail is connected to high voltage and represents logic 1.
  - The right rail is connected to low voltage and represents logic 0.
- **Connected Elements are Represented by Horizontal Connections**:
  - The state of the connected element is defined as “on” or “off” and corresponds to logic values 1 and 0. In the first case, it allows the electrical current to flow, and in the second, it does not.
- **Horizontal Connection**:
  - The state of the horizontal connection changes based on whether or not electric current flows through the element.
  - When the element is turned on, the horizontal connection transfers the state from the left side of the element to the right side.
- **Vertical Connections Link One or More Connected Elements**:
  - The state of the vertical connection is “off” if all the horizontal connections are off.
  - The state of the vertical connection is “on” if at least one horizontal connection is on.
  - The state of the vertical connection is transferred to all horizontal connections on its right.
  - Transferring the state to the left side of the vertical connection is not allowed.

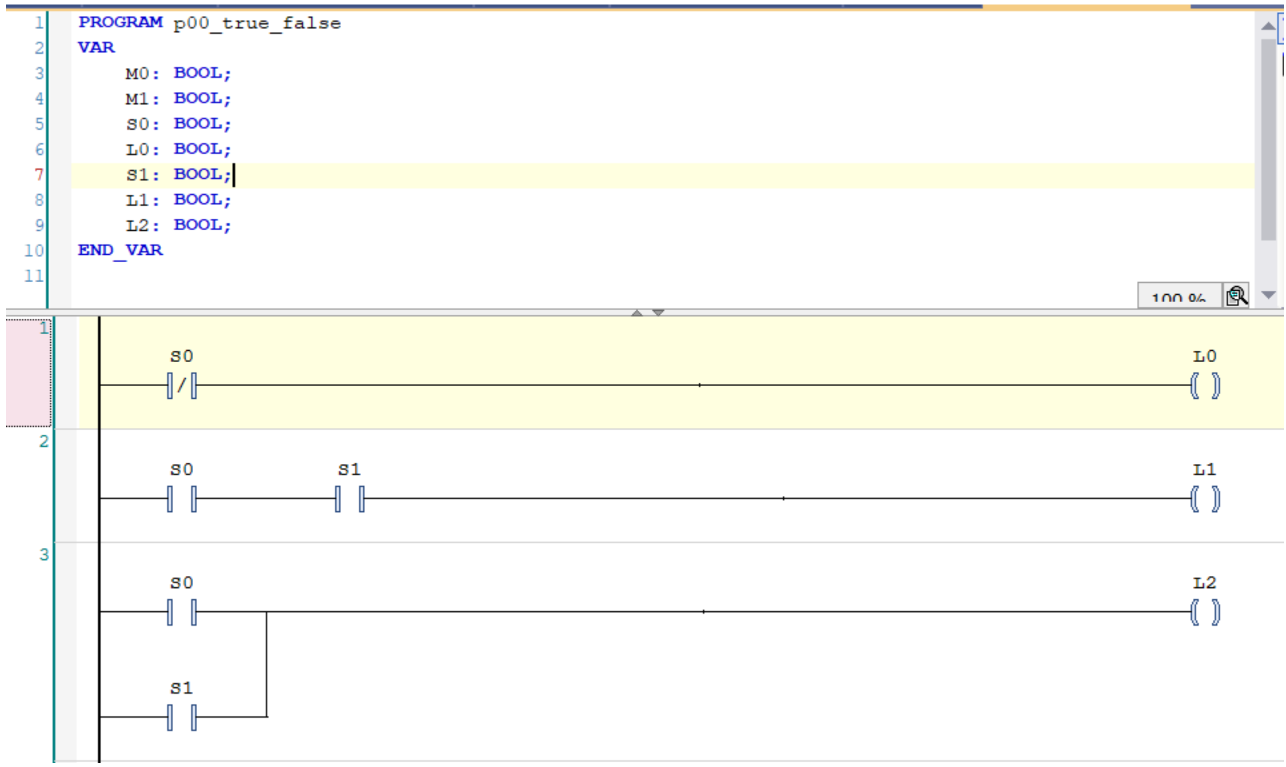
Ladder diagrams: standard

Standard 61131-3

- Elements and connections form ladder rungs (network)
- Typically, each rung has a set of appropriately connected input conditions, which control one or more coils (outputs)
    - Best practice:
        - Each coil (output) should only appear in the program at one place.

Example:

- Basic logical operations

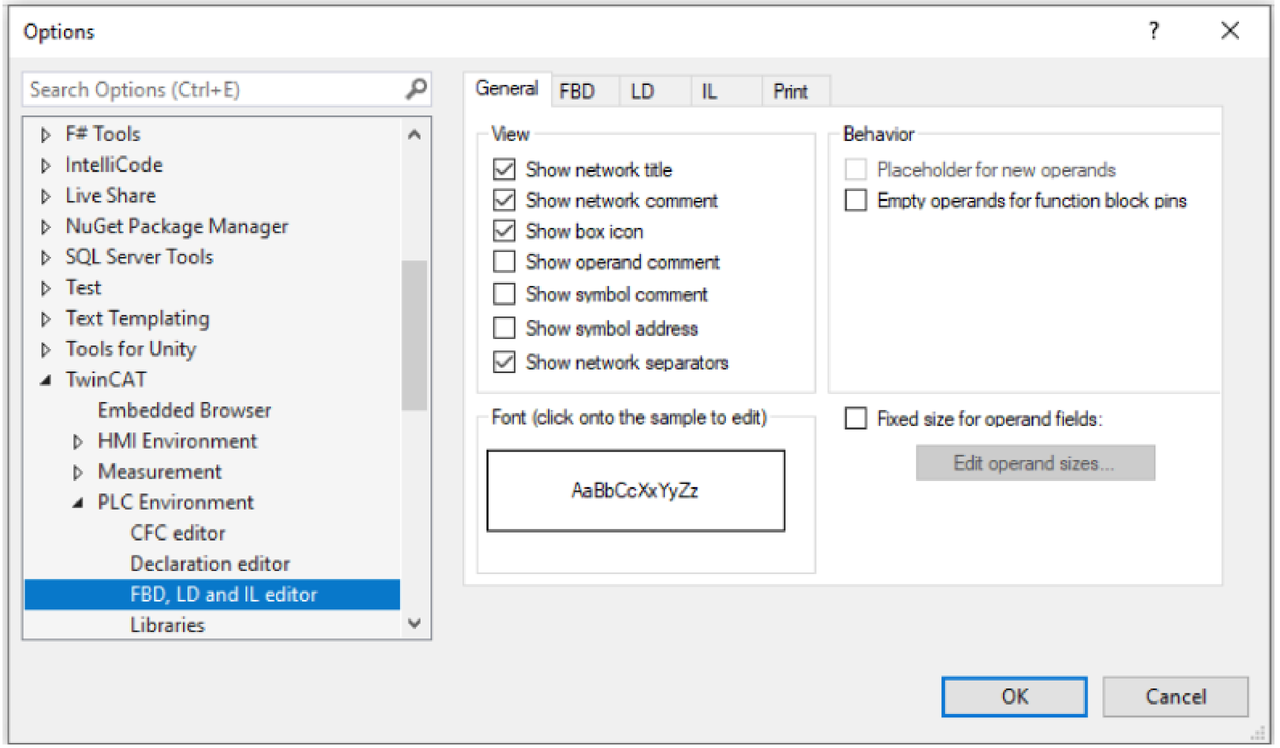


```
PROGRAM p00_true_false
VAR
    M0: BOOL;
    M1: BOOL;
    S0: BOOL;
    L0: BOOL;
    S1: BOOL;
    L1: BOOL;
    L2: BOOL;
END_VAR
```

Ladder diagrams: settings in TwinCAT

Navigation:

- Tools → Options → TwinCAT → PLC Environment
  - Display of titles and comments in the rungs (networks)
  - Font settings.



Options

Search Options (Ctrl+E)

- F# Tools
- IntelliCode
- Live Share
- NuGet Package Manager
- SQL Server Tools
- Test
- Text Templating
- Tools for Unity
- TwinCAT
  - Embedded Browser
  - HMI Environment
  - Measurement
  - PLC Environment
    - CFC editor
    - Declaration editor
    - FBD, LD and IL editor
    - Libraries

General | FBD | LD | IL | Print

View
- [✓] Show network title
- [✓] Show network comment
- [✓] Show box icon
- [ ] Show operand comment
- [ ] Show symbol comment
- [ ] Show symbol address
- [✓] Show network separators

Behavior
- [ ] Placeholder for new operands
- [ ] Empty operands for function block pins

Font (click onto the sample to edit)

AaBbCcXxYyZz

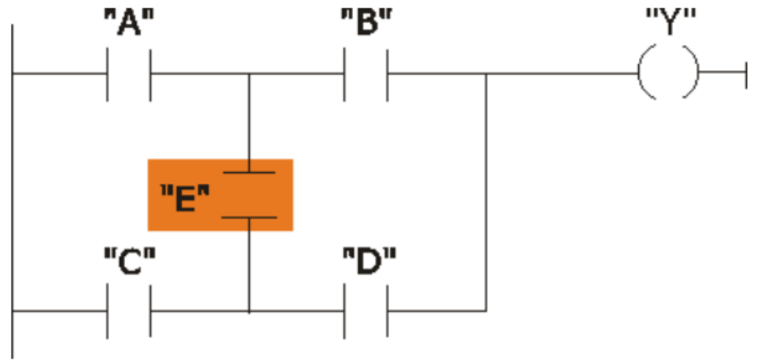
- [ ] Fixed size for operand fields:

Edit operand sizes...

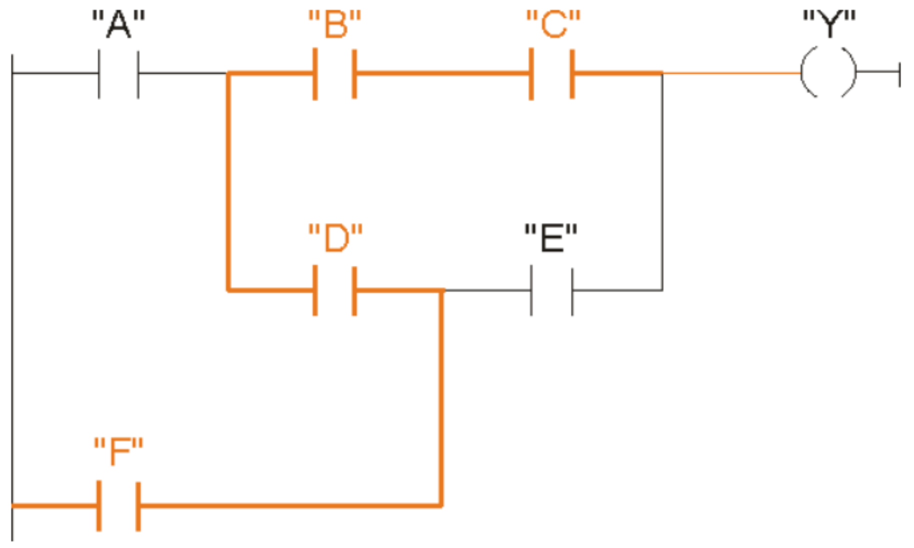
OK | Cancel

Limitations of ladder diagrams

- No vertical connections between elements

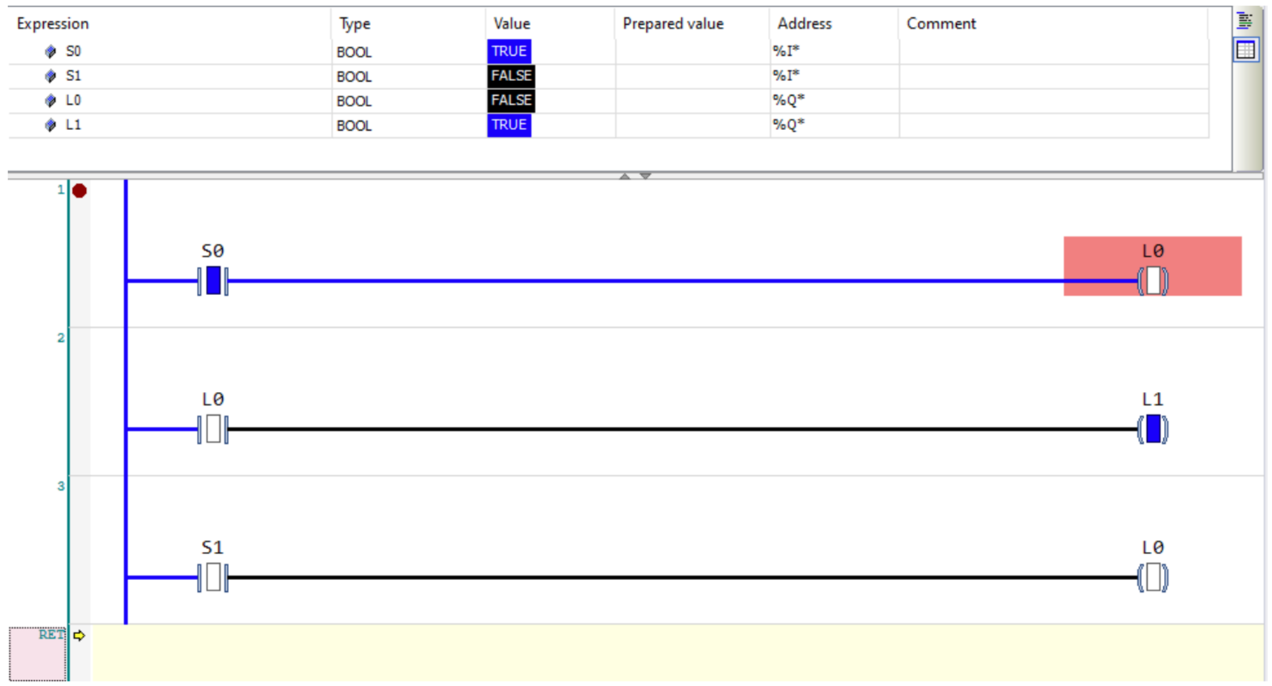


- Current flows only from left to right
  - Tools for creating such connections typically don't allow this



Ladder diagrams: changes

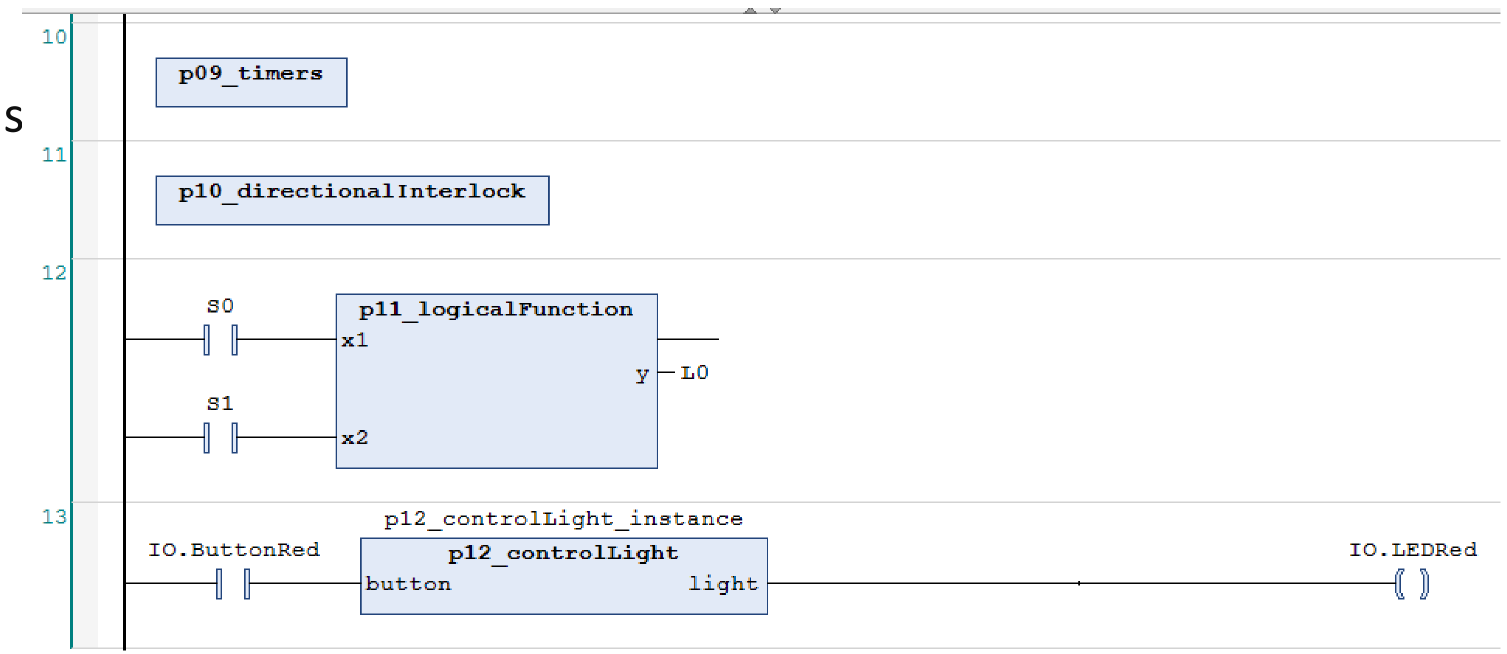
- Changes in Inputs and Outputs
  - Inputs remain constant throughout the entire program cycle (parallel)
  - Outputs change sequentially, rung by rung (sequential)
    - The last output value is transferred to the physical output at the end of the memory image (output buffer).



Expression	Type	Value	Prepared value	Address	Comment
S0	BOOL	TRUE		%I*	
S1	BOOL	FALSE		%I*	
L0	BOOL	FALSE		%Q*	
L1	BOOL	TRUE		%Q*	

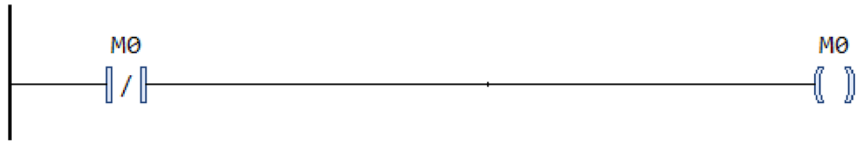
Ladder diagrams: extensions

- Basic ladder diagram only includes switches and relays
- Necessary extensions:
  - Function and subprogram calls
  - Jumps
  - Use of structured data types
  - Analog value processing

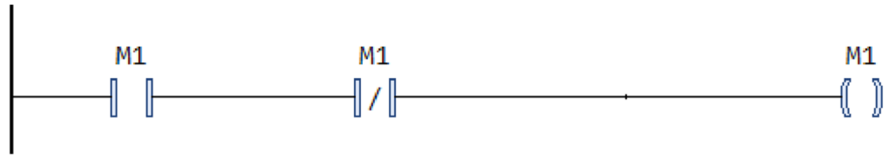


- Challenges:
  - Programs become harder to understand
    - The intuitiveness of switches and relays is lost
  - Not suitable for large, complex projects

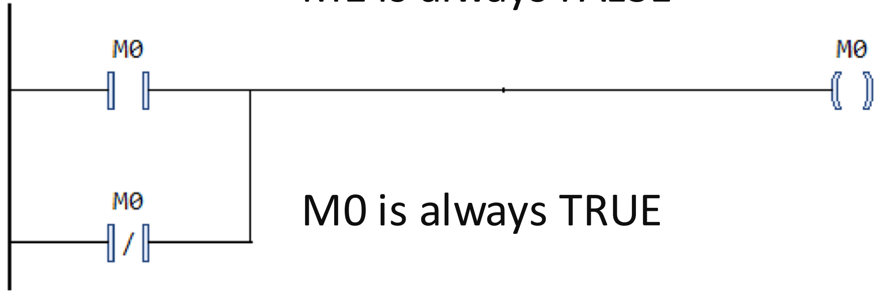
Ladder diagrams: examples



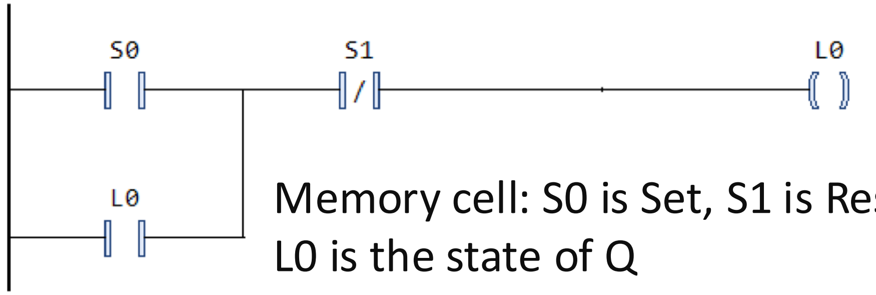
Bit M0 changes its value every cycle – we get a clock.



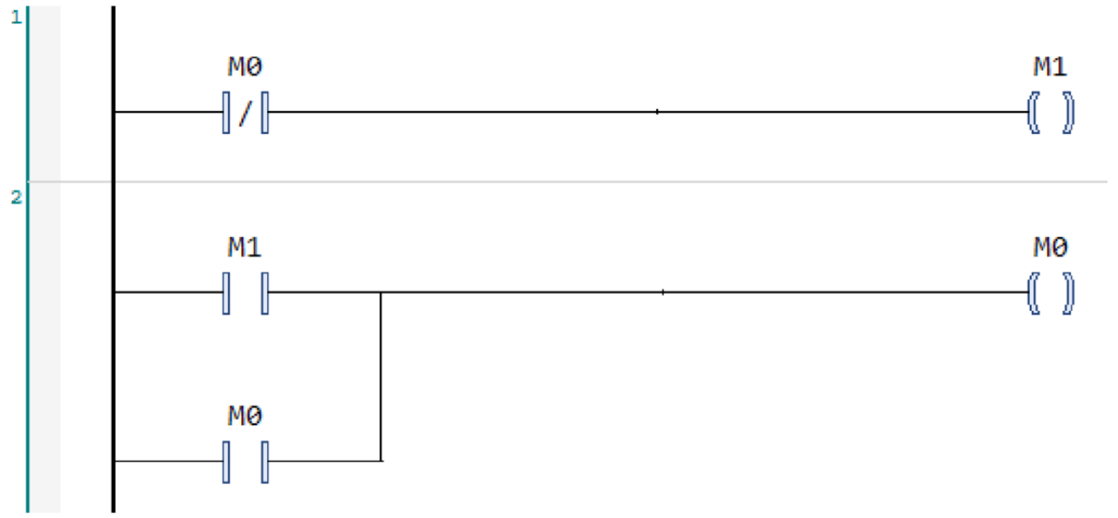
M1 is always FALSE



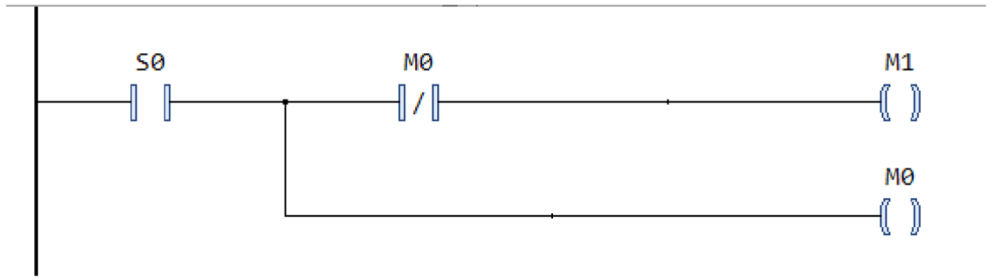
M0 is always TRUE



Memory cell: S0 is Set, S1 is Reset, L0 is the state of Q



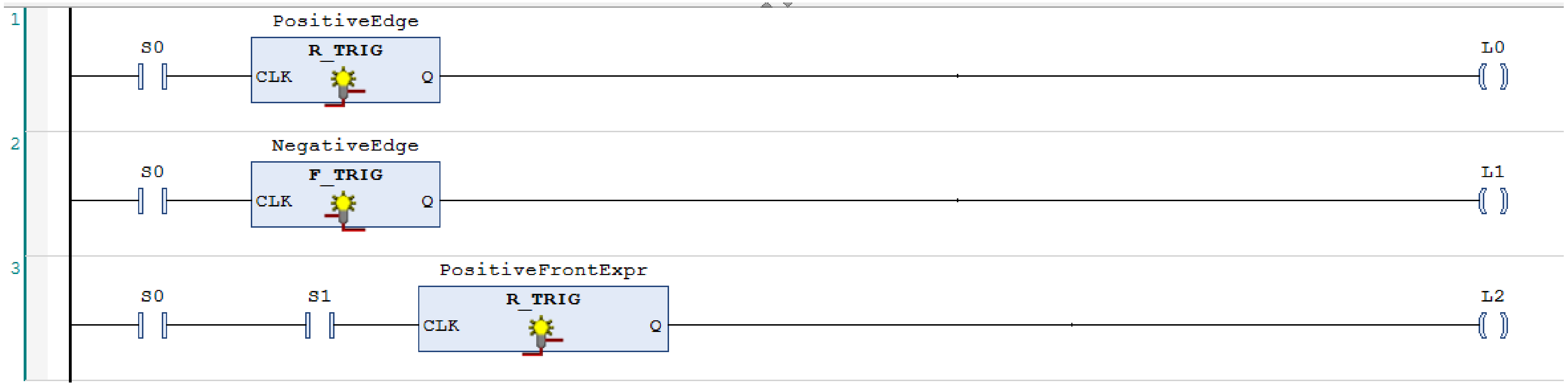
M1 will be TRUE only during the first cycle of program execution.



Positive edge detection on S0. M1 will be TRUE for exactly one cycle – when S0 changes from FALSE to TRUE

Ladder diagrams: edge detection

- **Front** refers to a change in the value of a bit and lasts for one cycle.
  - **Positive Edge**: When the value changes from **FALSE** to **TRUE**. This is detected using the function block **R_TRIG**.
  - **Negative Edge**: When the value changes from **TRUE** to **FALSE**. This is detected using the function block **F_TRIG**.

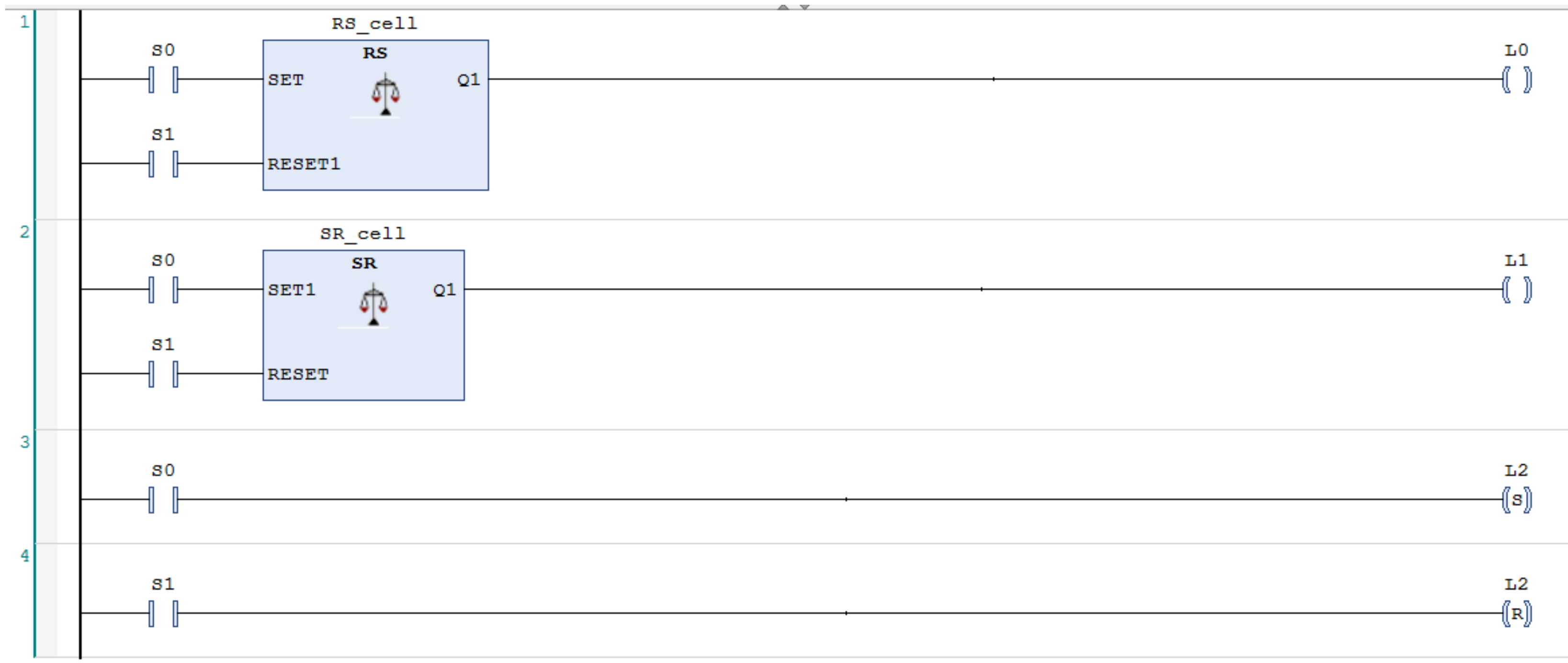


Ladder diagrams: memory cell

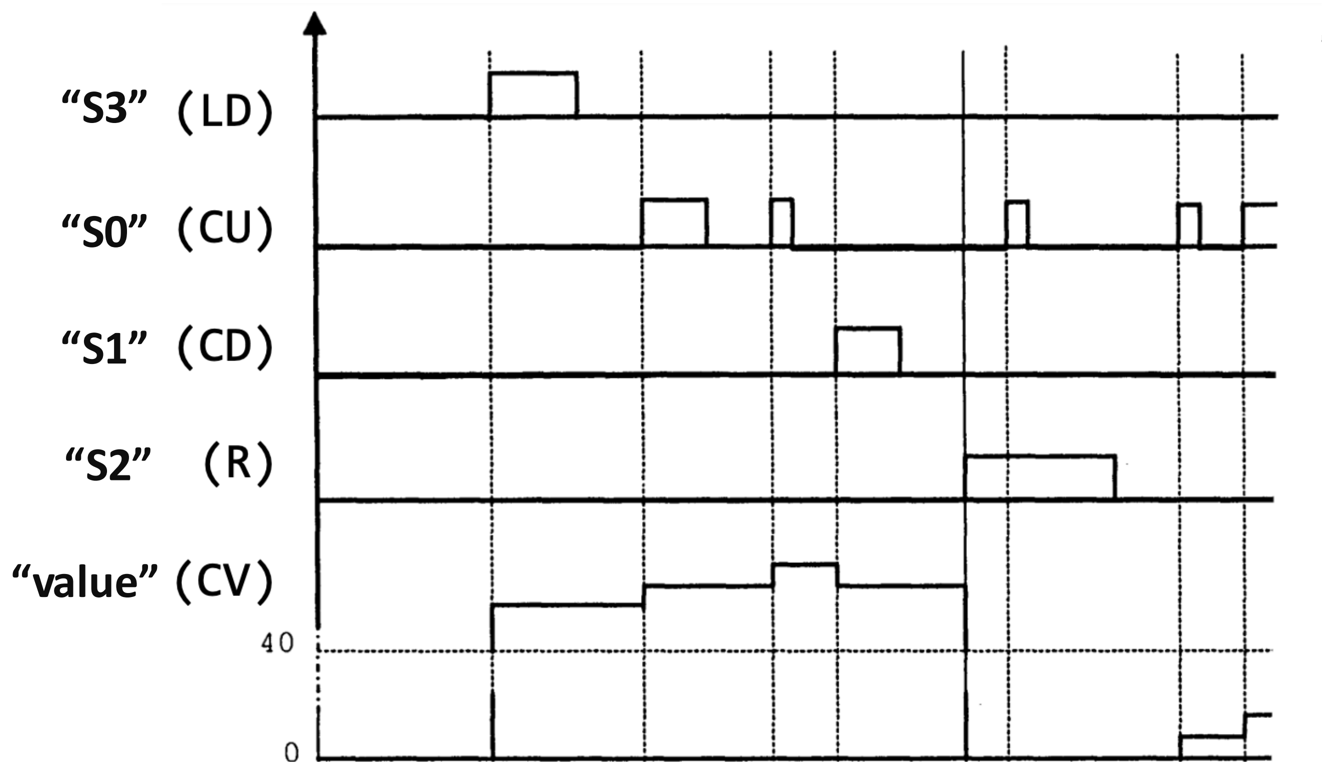
- Memory cells RS (RESET takes precedence over SET) and SR (SET takes precedence)

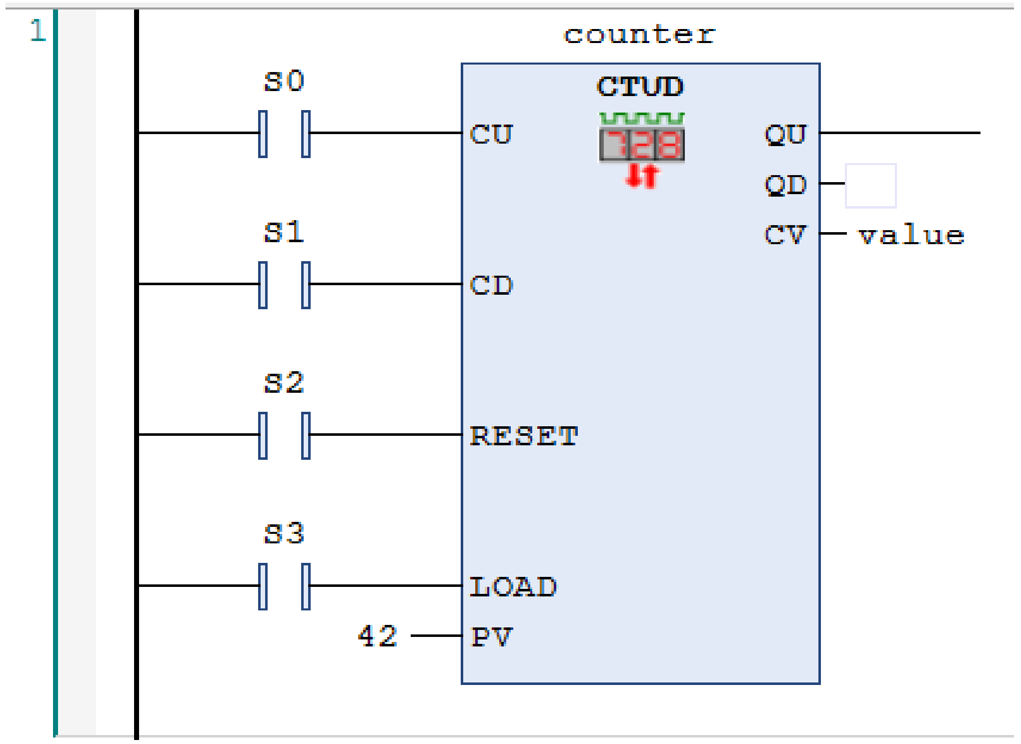
- **Standard method using function blocks**
  - Rungs 1 and 2

- **Alternative method with coils**
  - Rungs 3 and 4



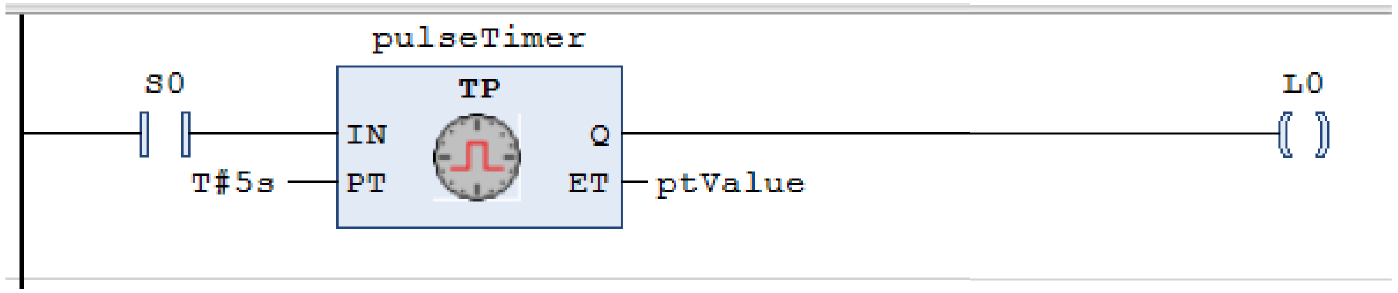
Ladder diagrams: counters



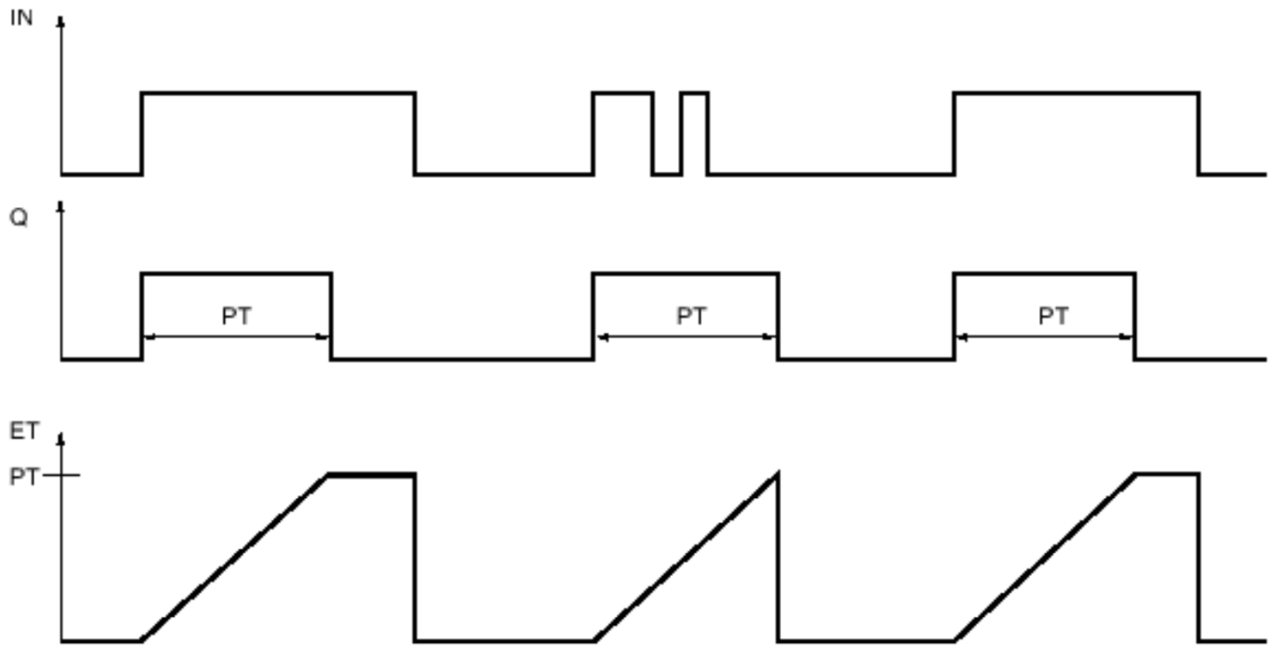


Ladder diagrams: timer TP

Pulse generator using a timer:

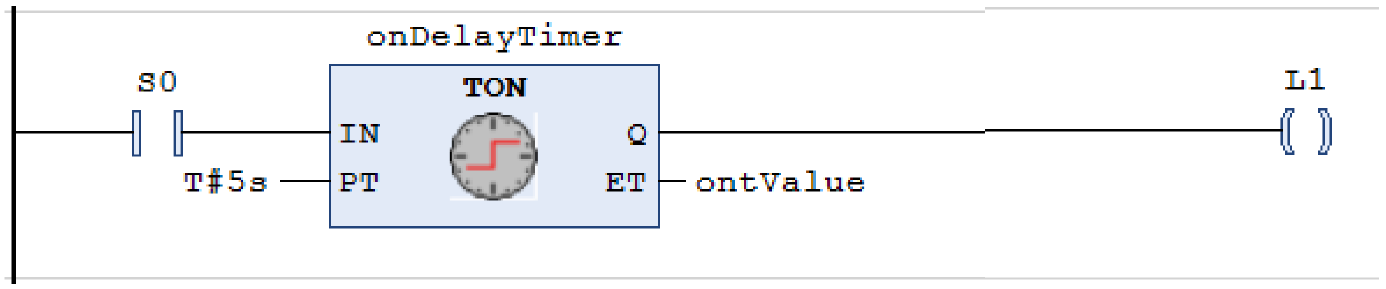


Time diagram:

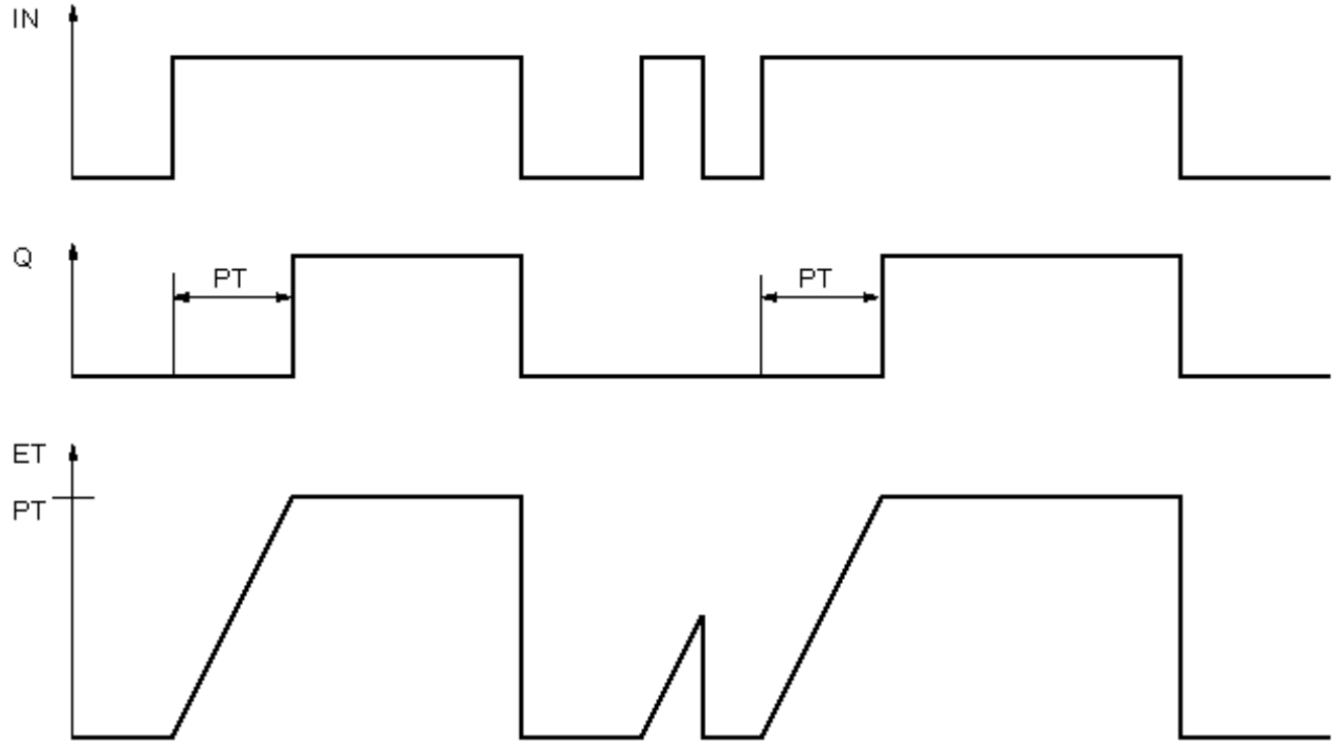


Ladder diagrams: TON timer

Power-on delay (timer-on delay):

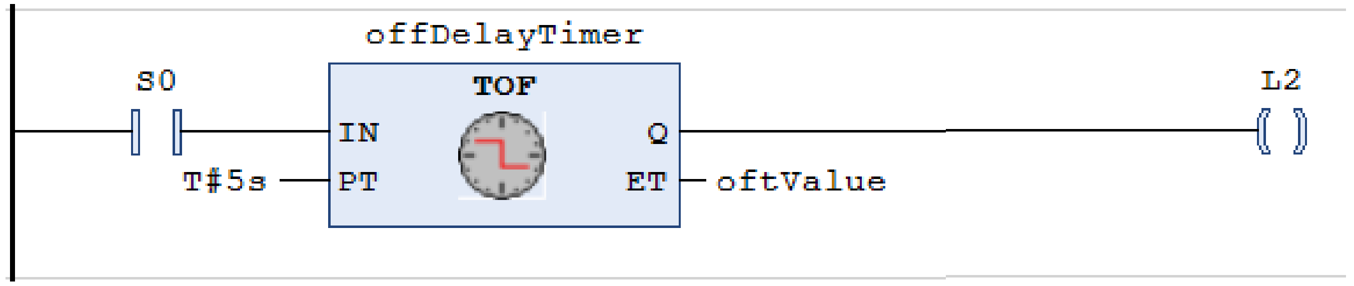


Time diagram:

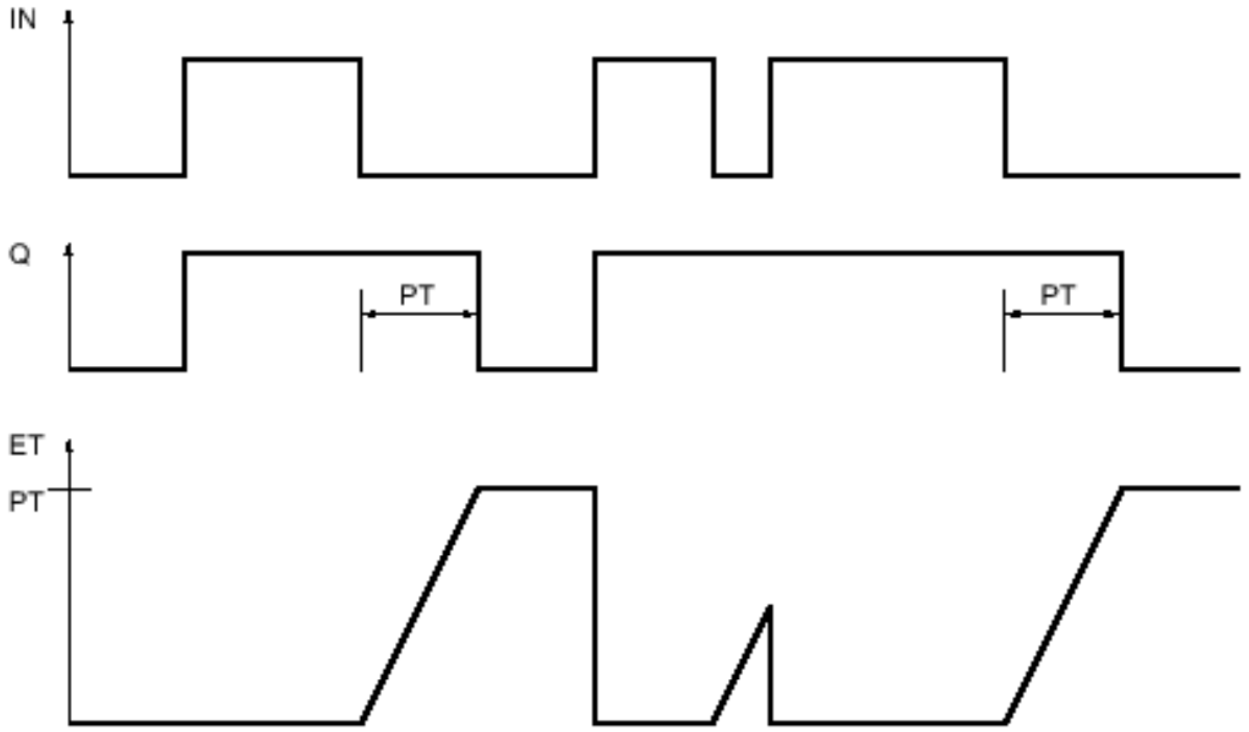


Ladder diagrams: TOF timer

Off-delay (timer-off delay):

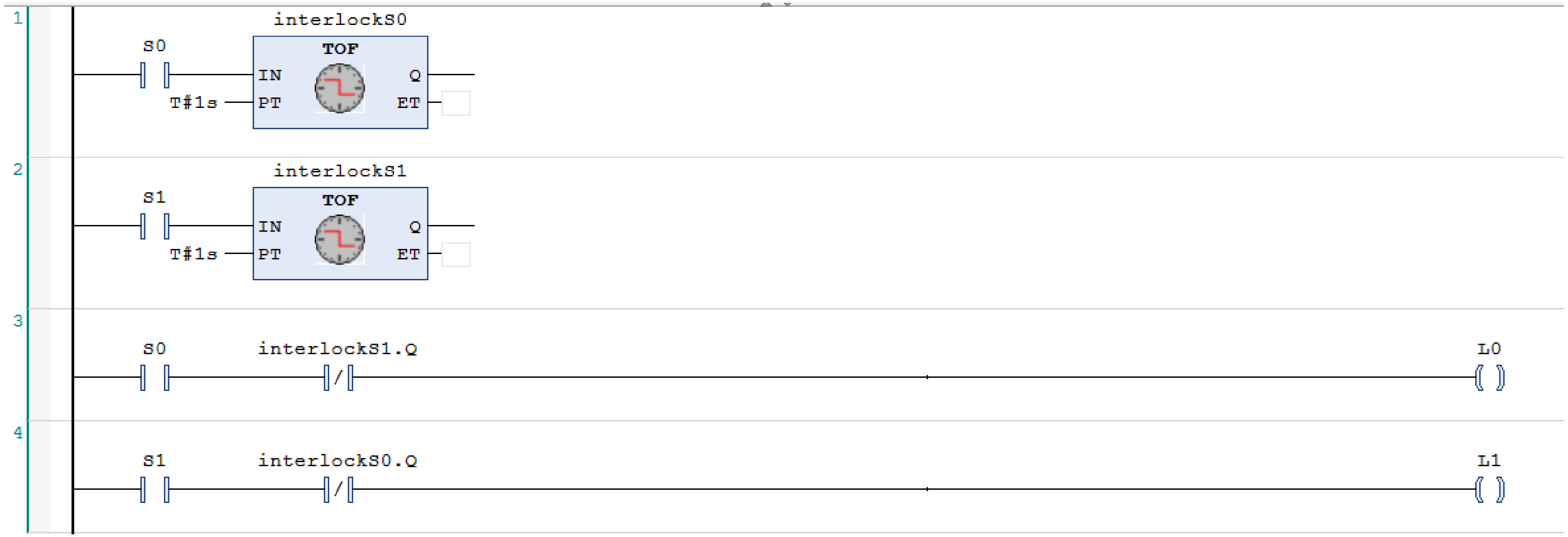


Time diagram:



Ladder diagrams: timers

- **Example:** Prevention of premature reversal of motor direction through **directional interlock**


1
interlockS0
S0
TOF
IN
Q
PT
ET
T#1s
2
interlockS1
S1
TOF
IN
Q
PT
ET
T#1s
3
S0
interlockS1.Q
L0
4
S1
interlockS0.Q
L1