

Basics of Ladder Diagram

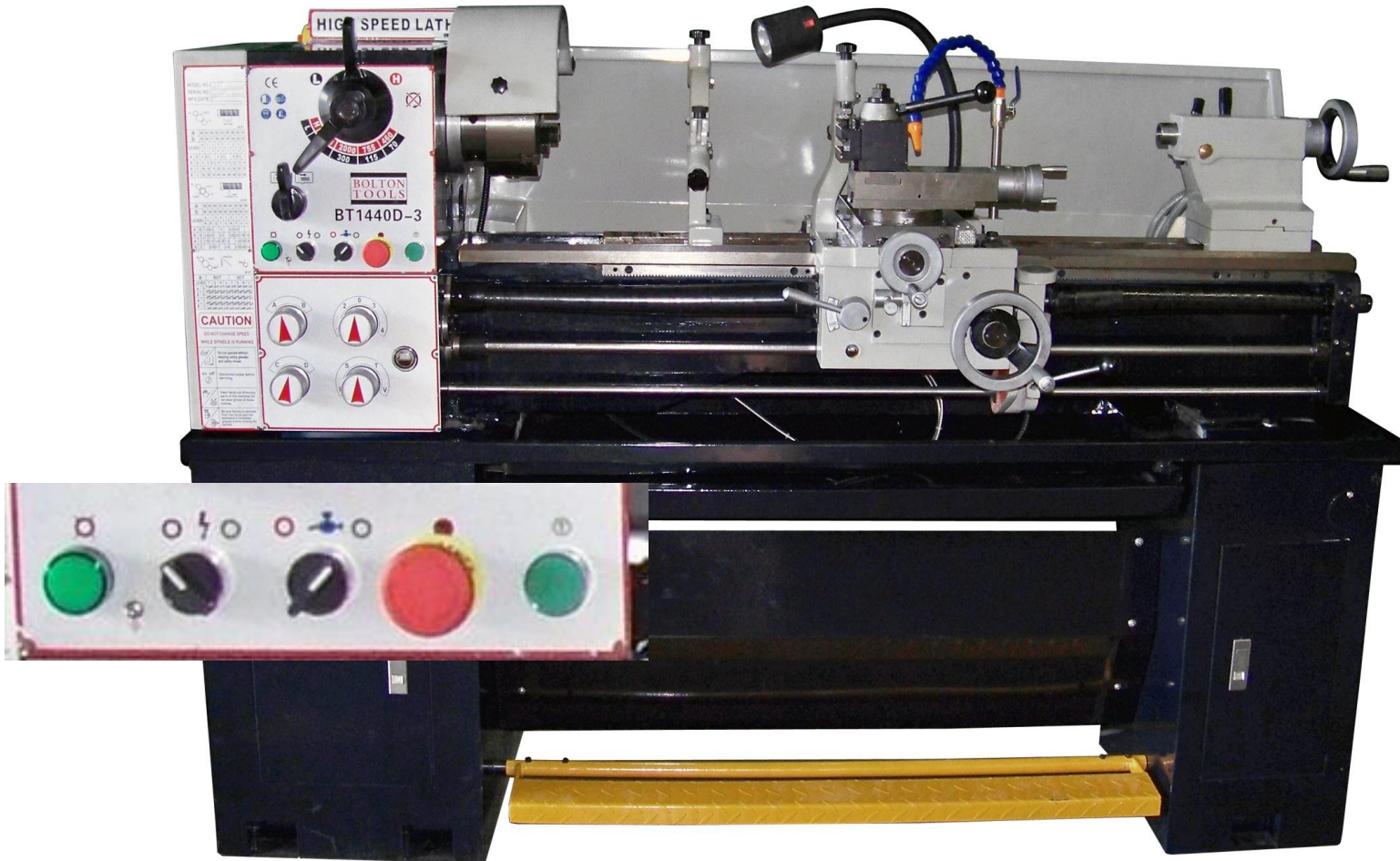
Industrial control

KOVÁCS Gábor

gkovacs@iit.bme.hu



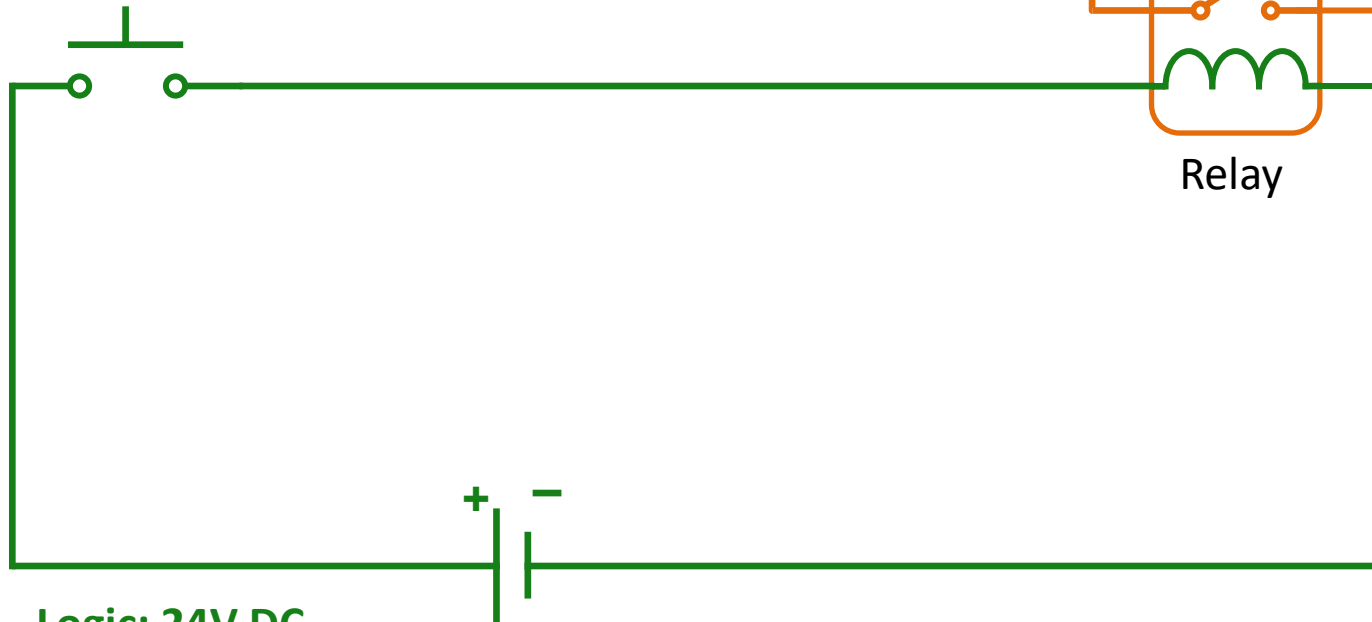
From wired logic to PLC code



The machine is operated by a pushbutton

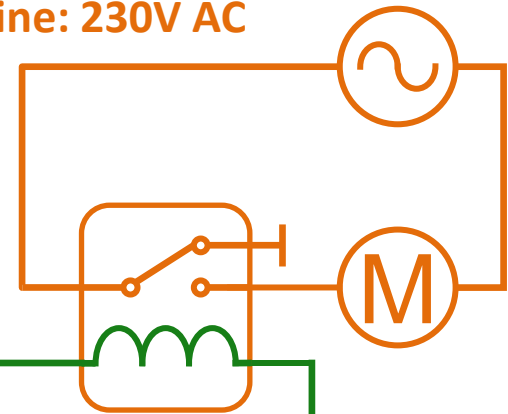


Normally Open (NO)
pushbutton

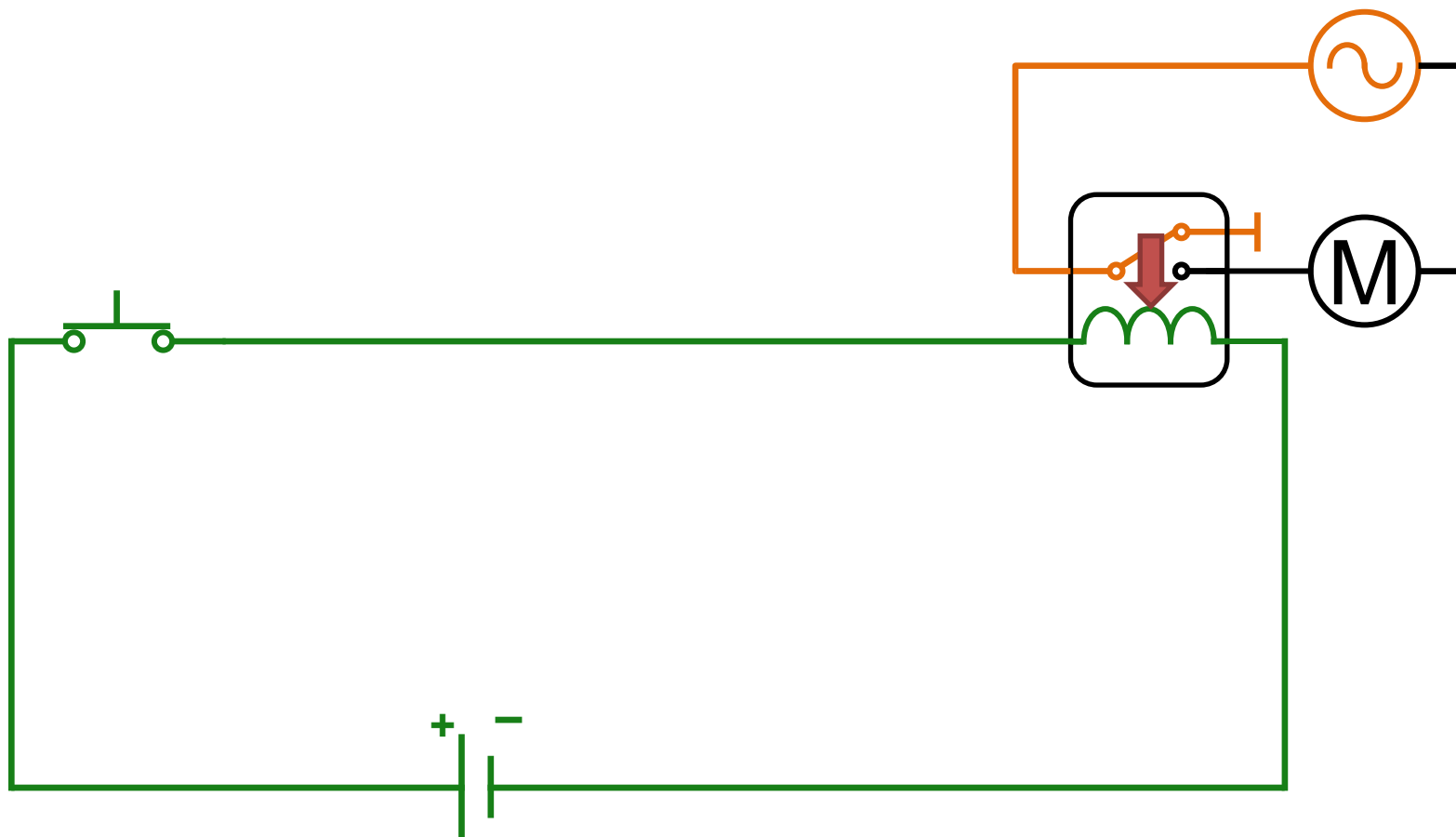


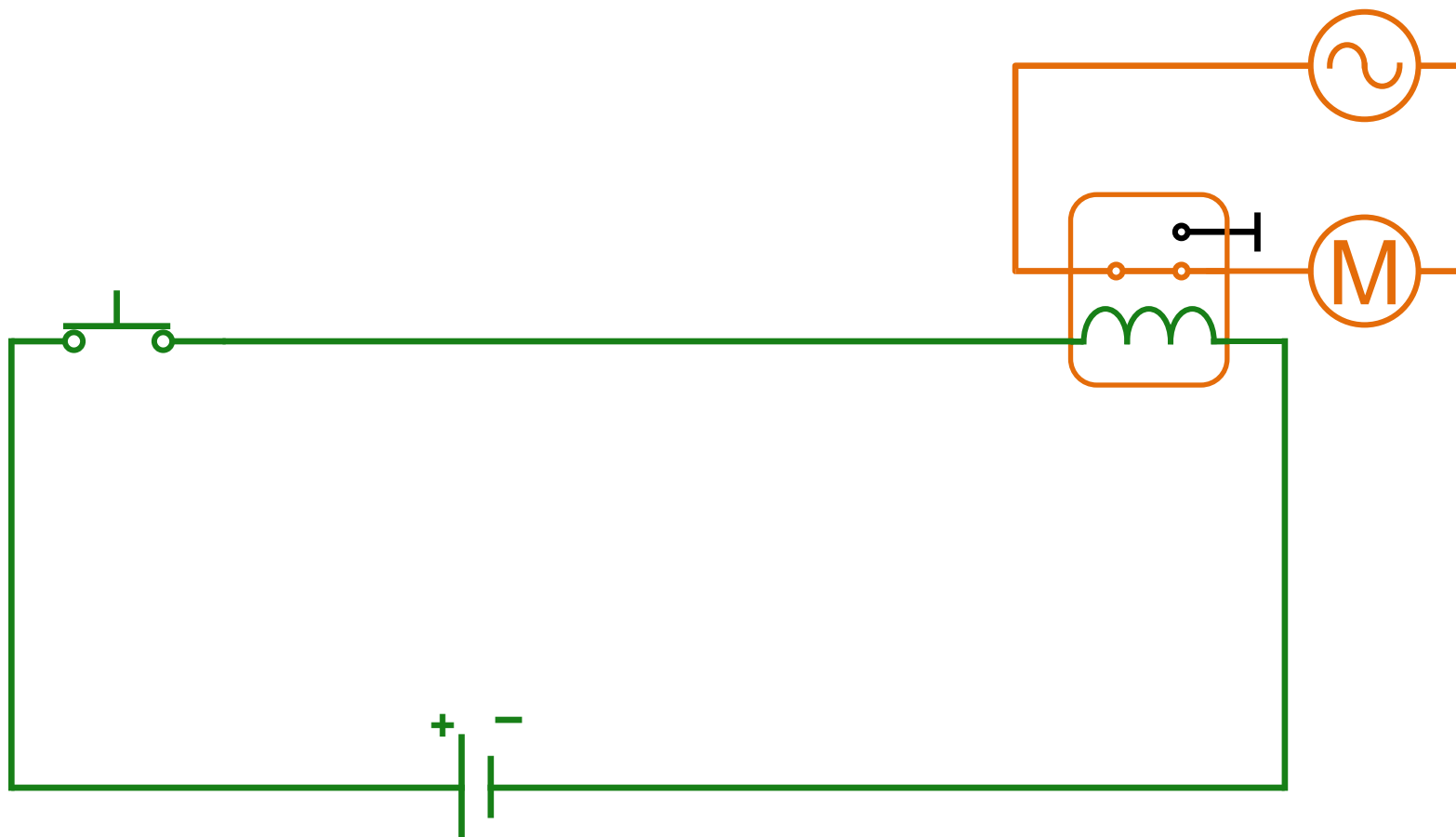
Logic: 24V DC

Machine: 230V AC

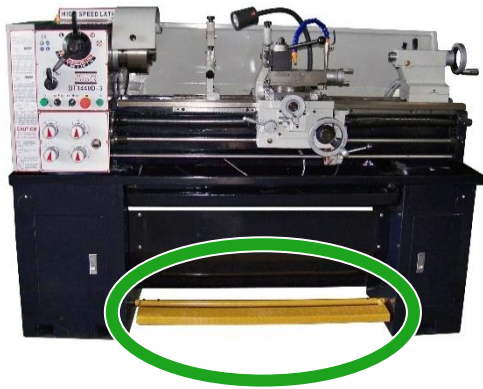


Relay

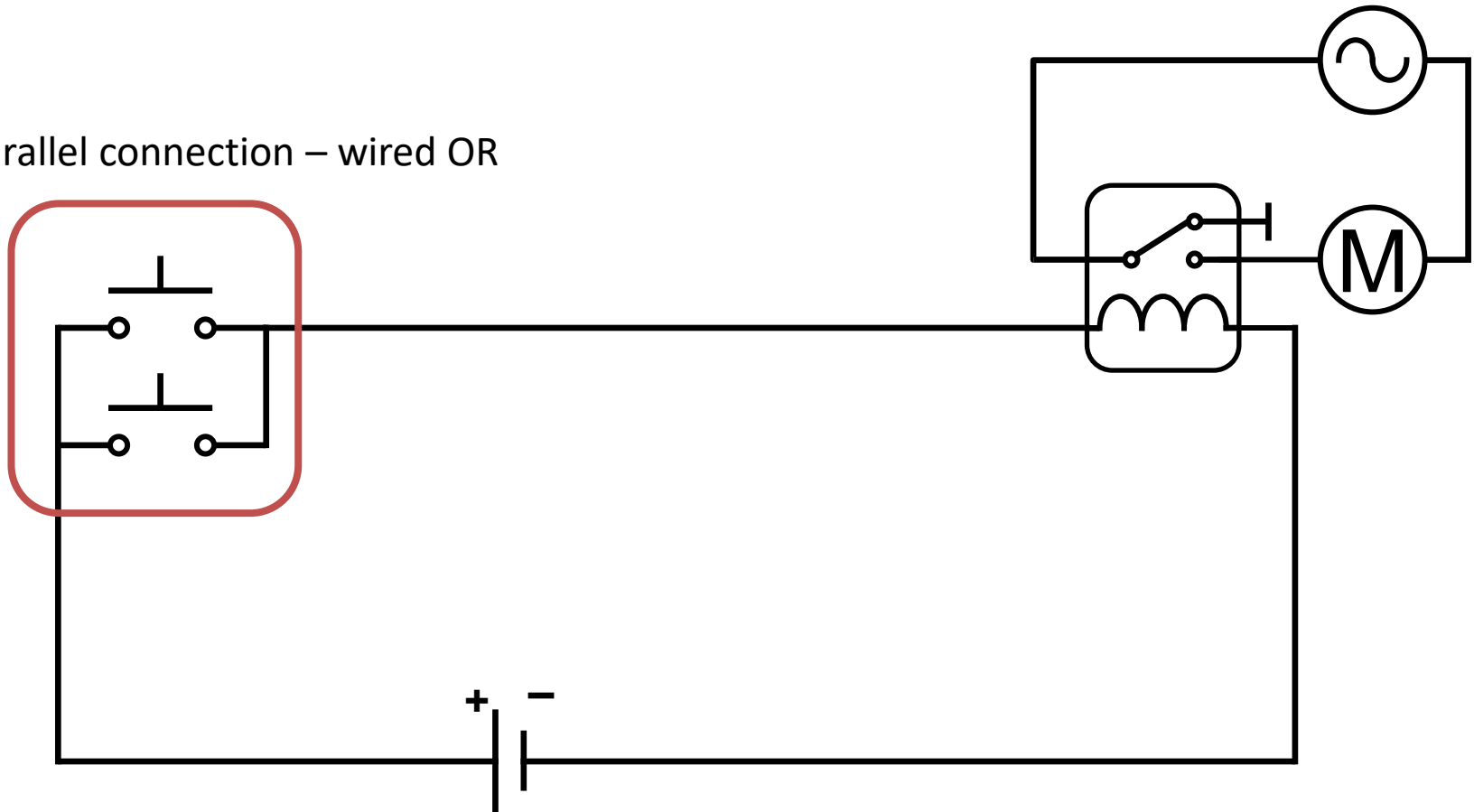




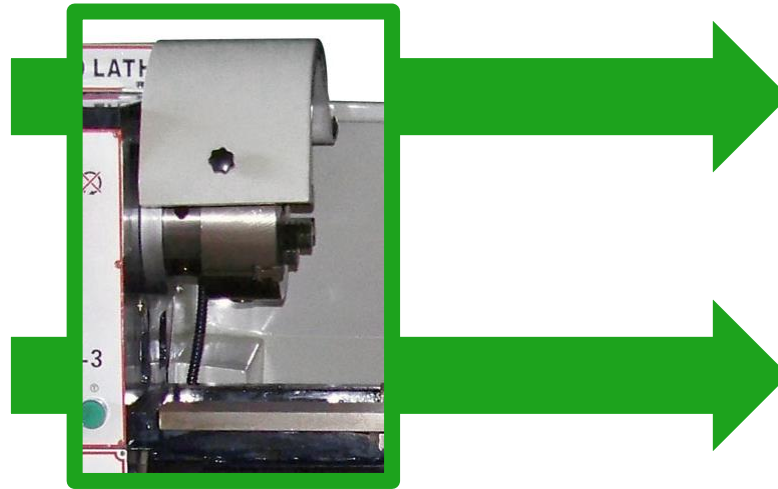
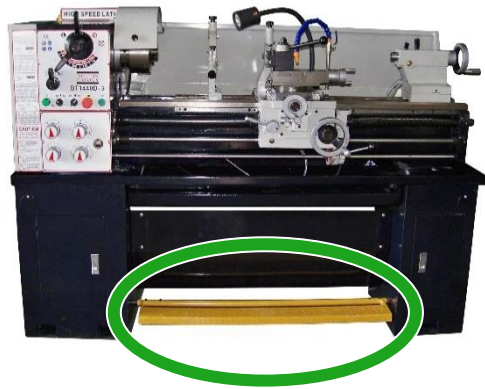
The machine can be operated either by a pushbutton or a foot switch (OR)



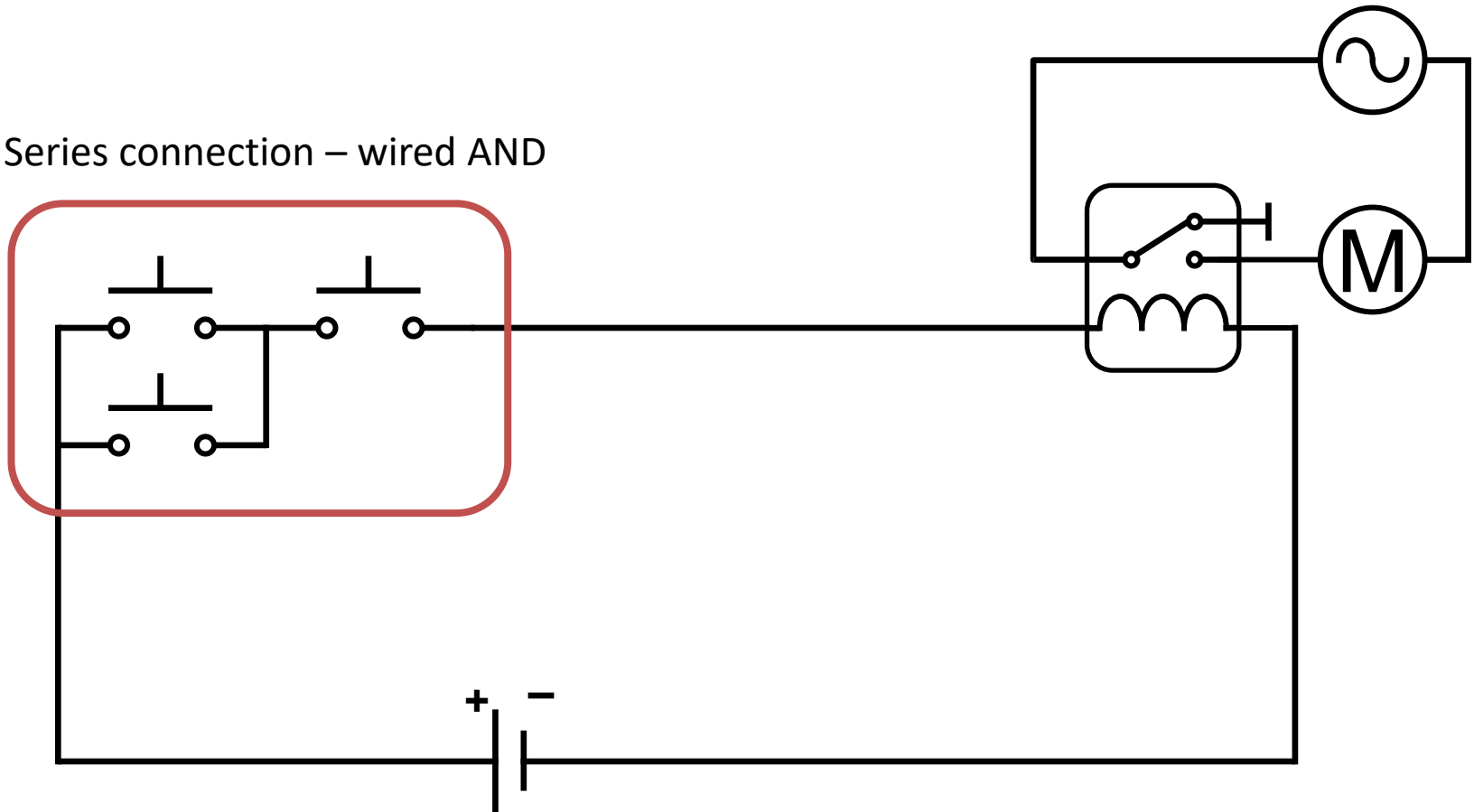
Parallel connection – wired OR



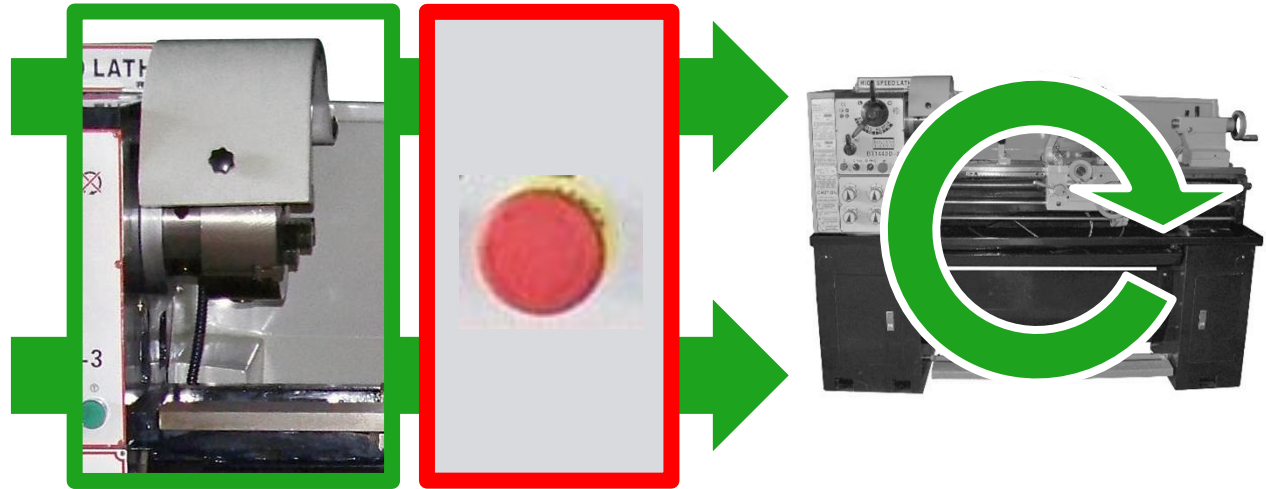
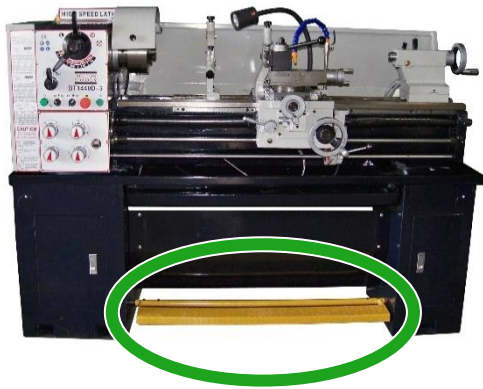
Safety feature: the machine can operate only with the cover closed (AND)

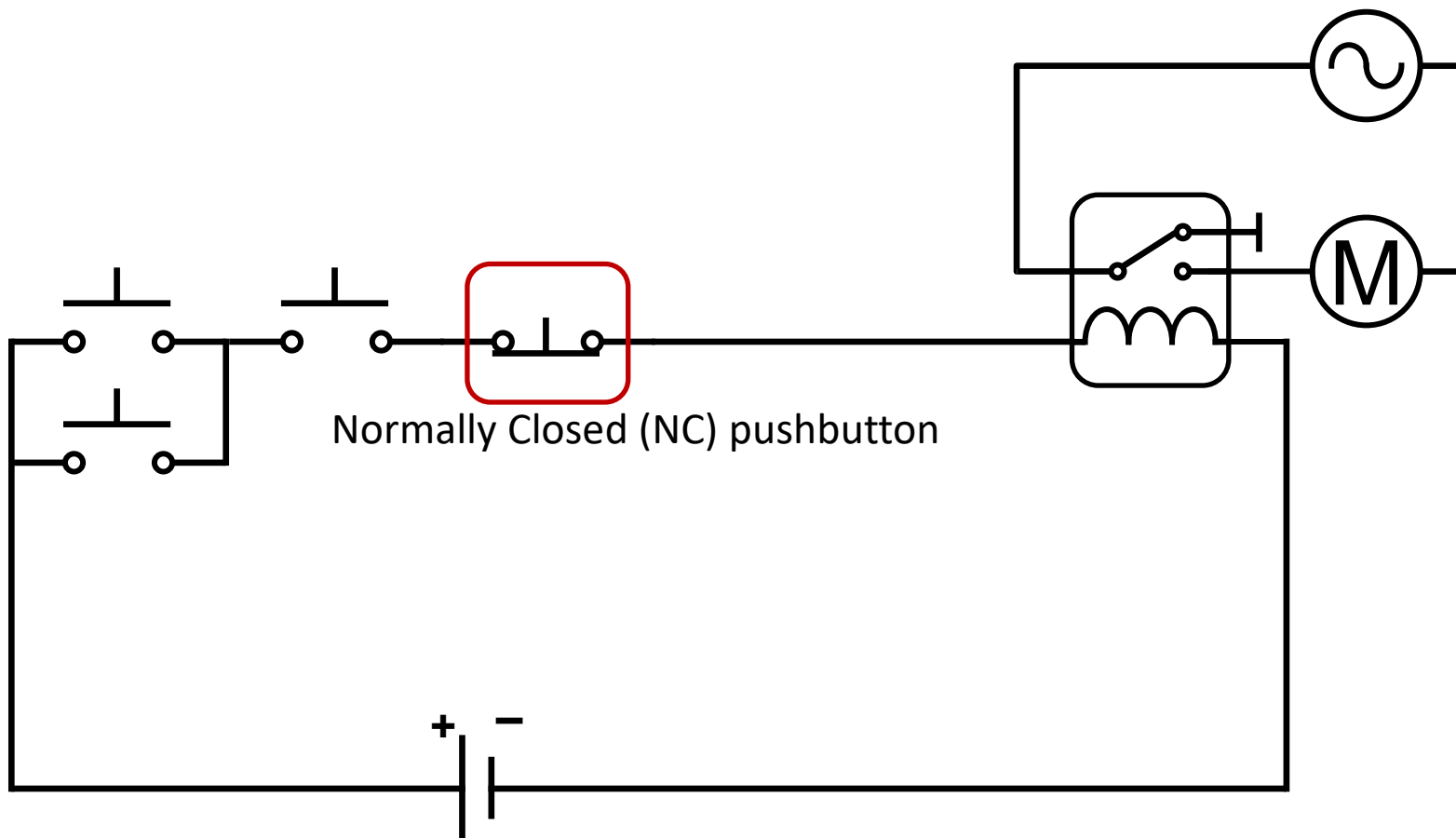


Series connection – wired AND

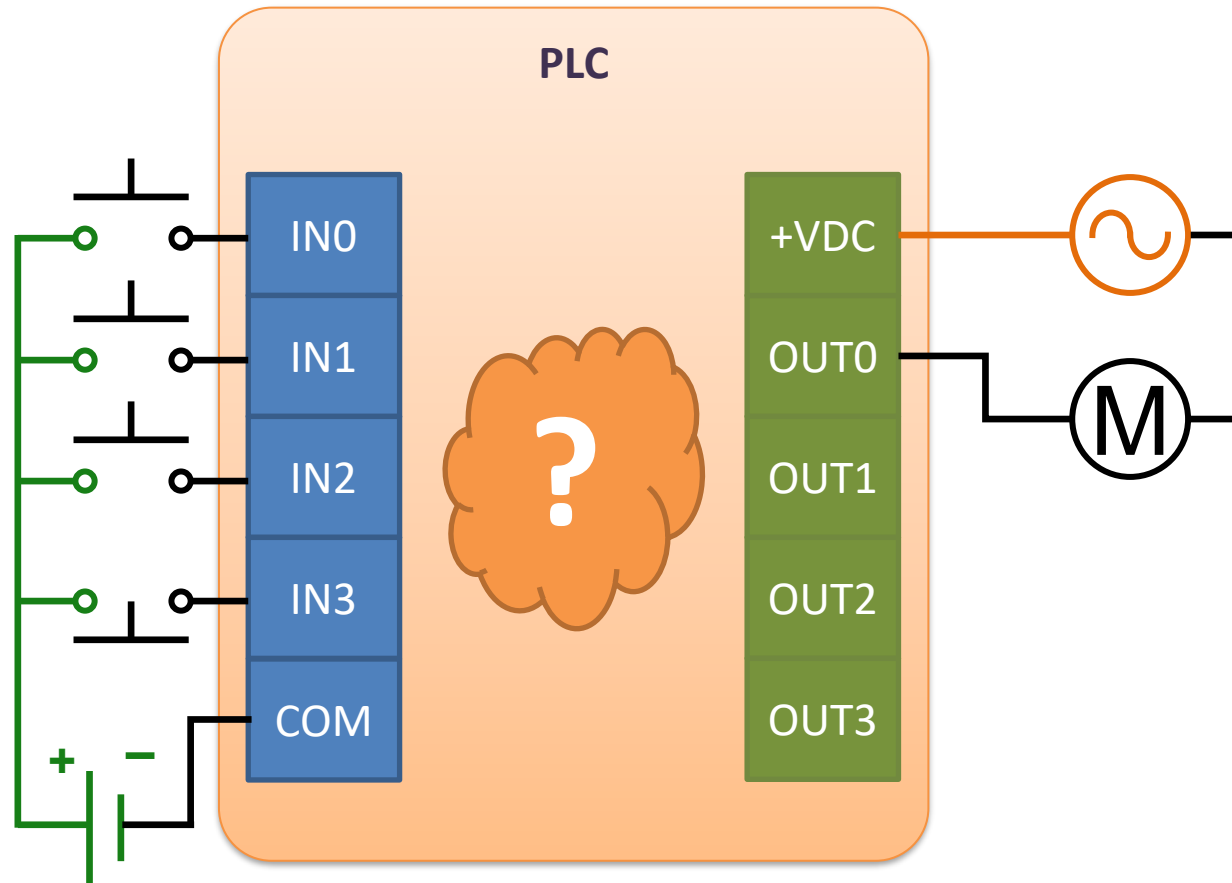


Emergency switch: the machine should not operate if the emergency switch it is pressed
(AND NOT)



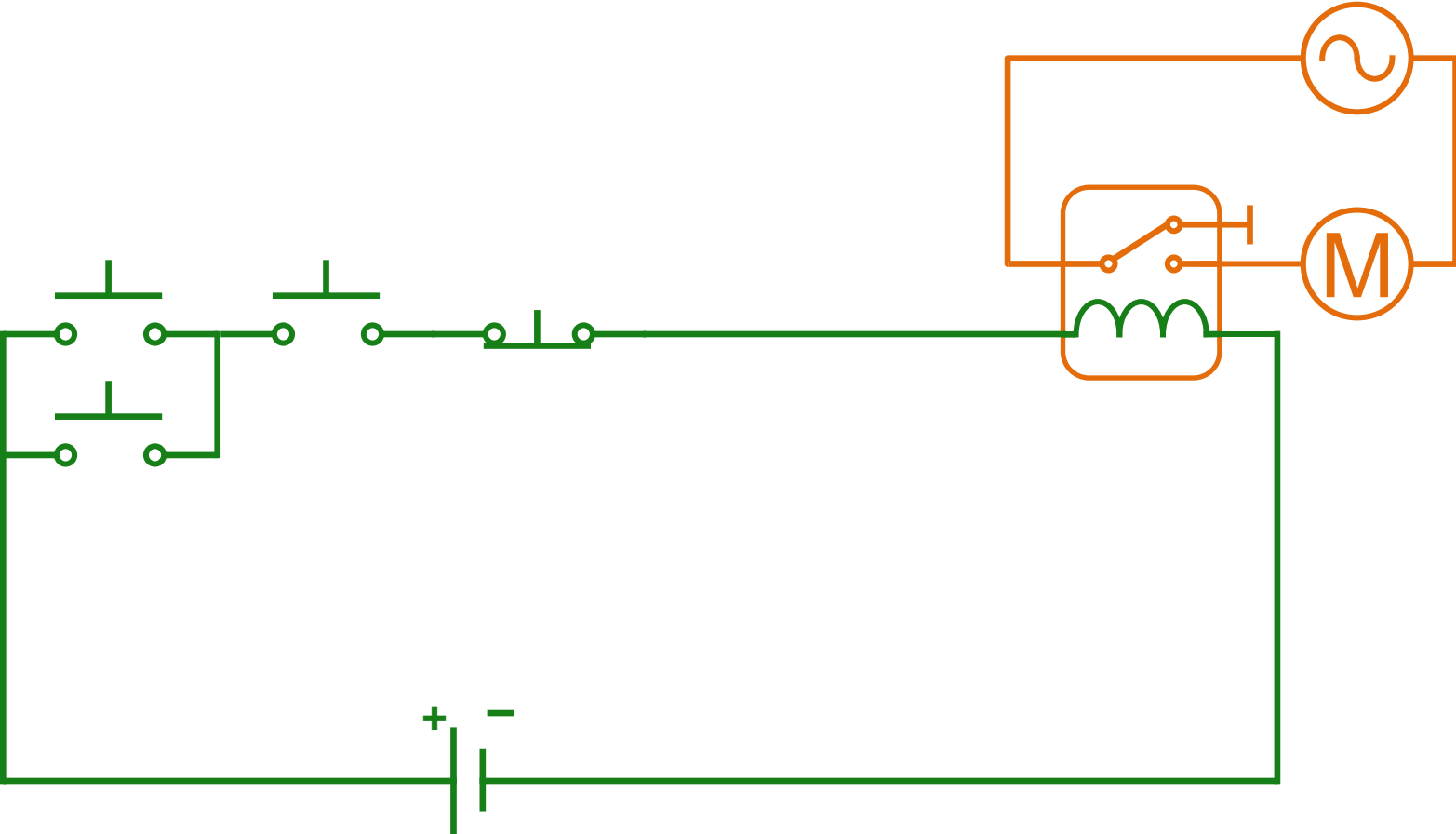


Connecting the PLC IOs

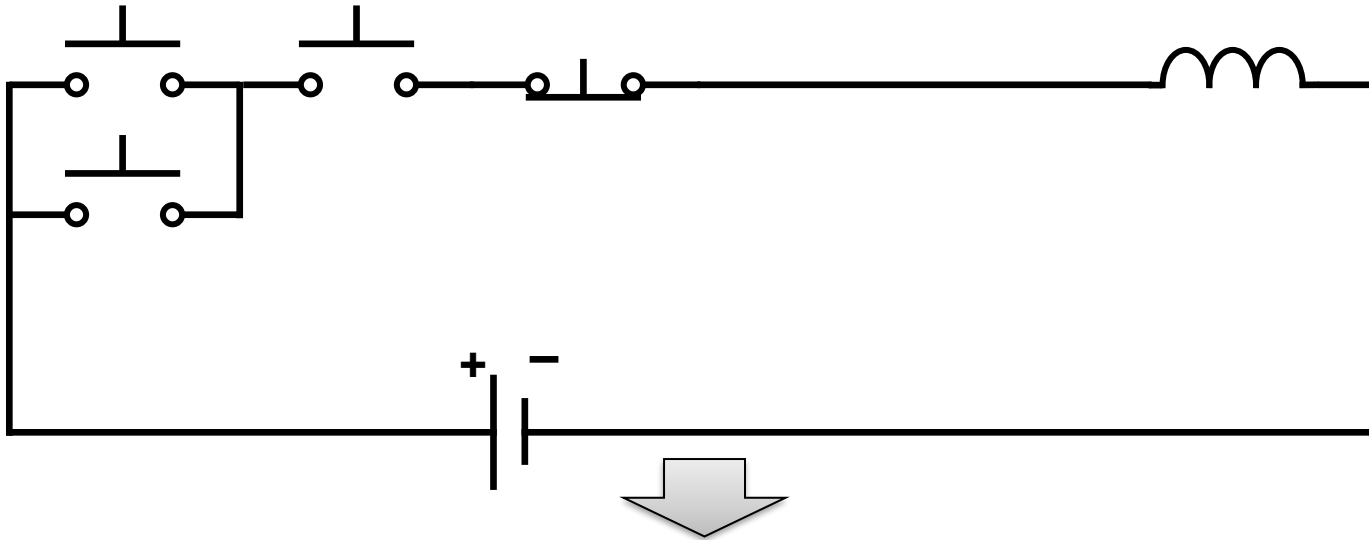


Voltage of most PLC outputs is 24VDC.
Do not connect high voltage to such modules!

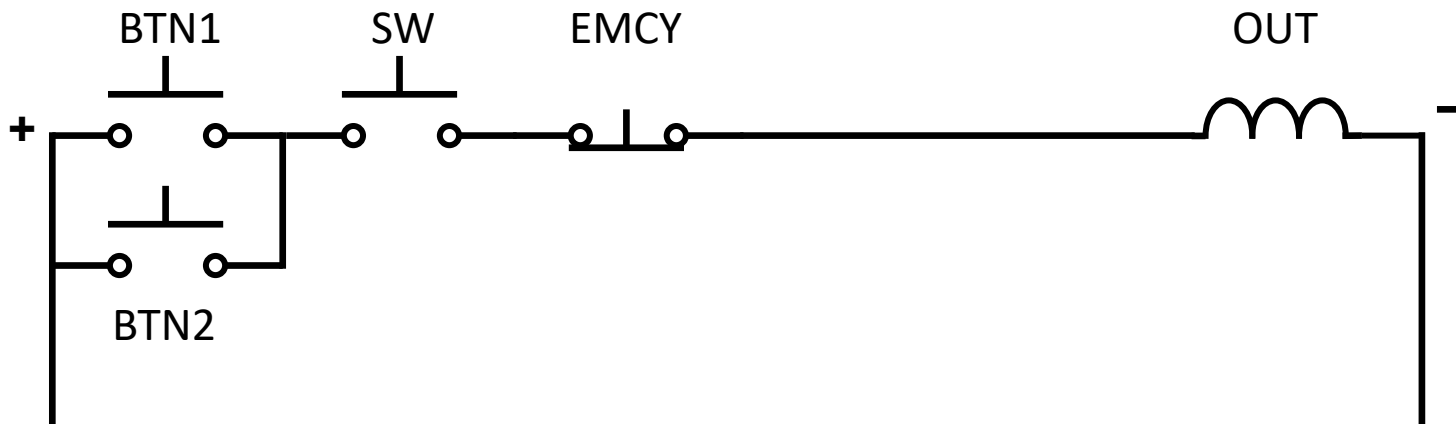
How to transform the electrical circuit to a PLC program?

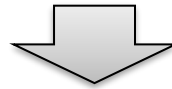
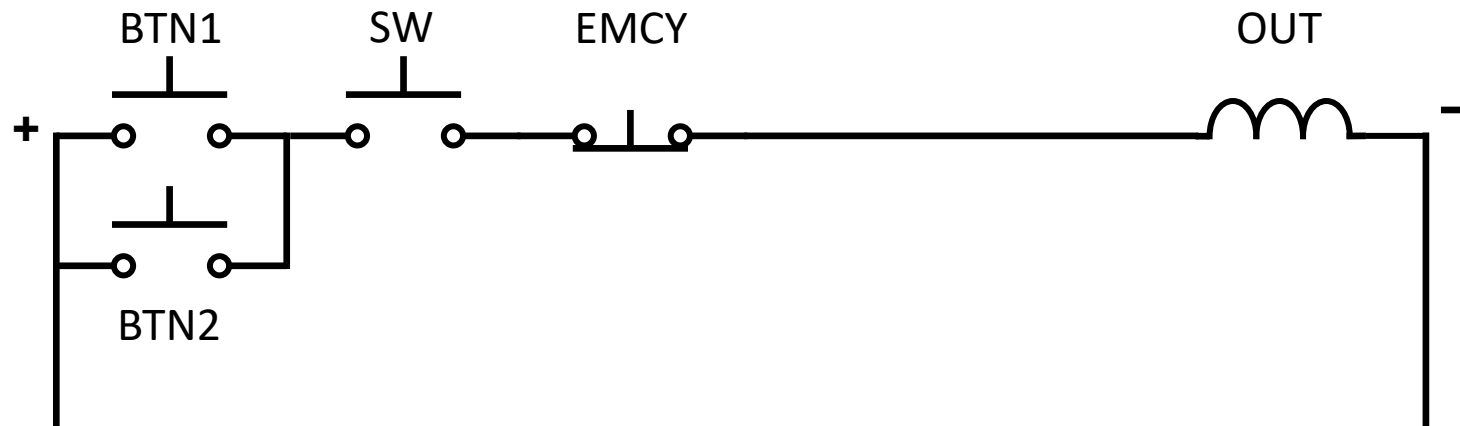


1. Consider only the part corresponding to the control logic!

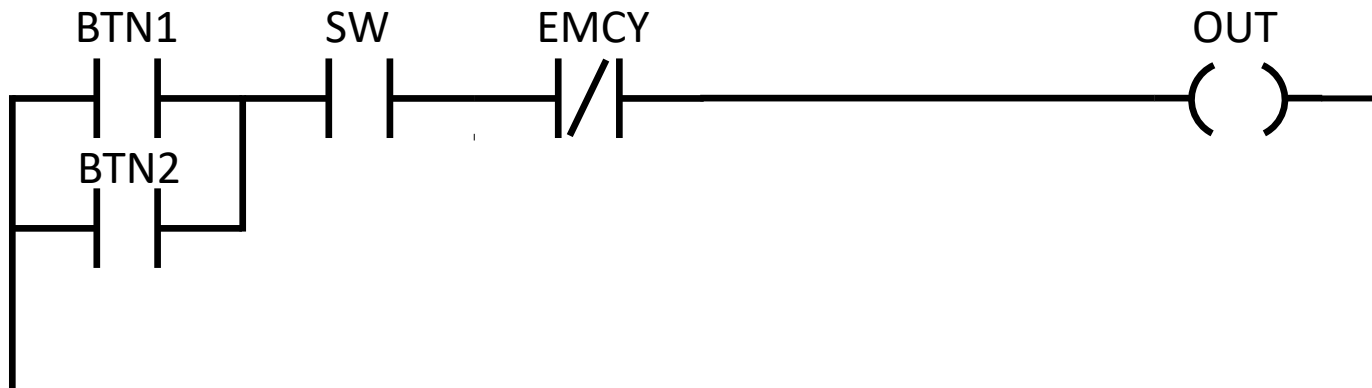
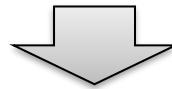


2. Replace the power source by two vertical rails at both sides and label the elements according to their roles!

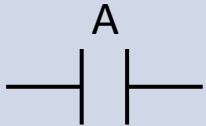



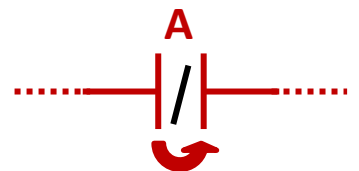
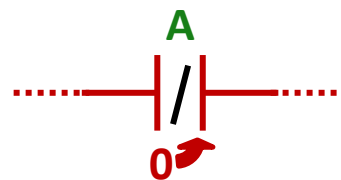
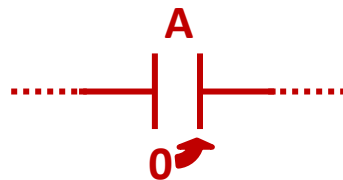
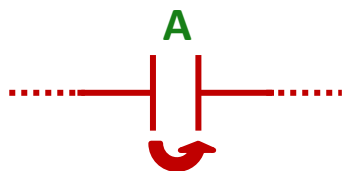
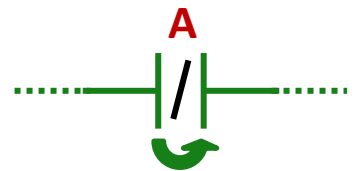
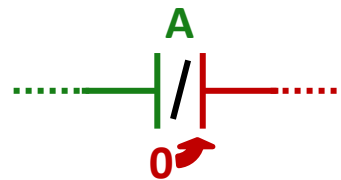
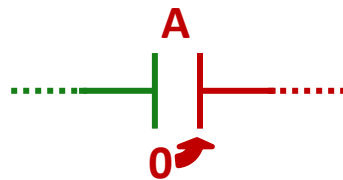
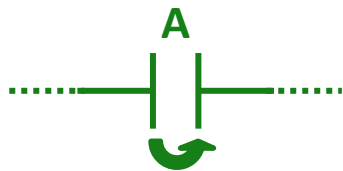


3. Change the symbols to create your first PLC program!

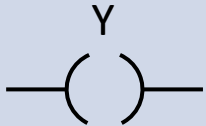
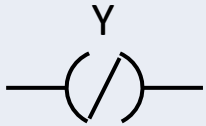


Contacts

Symbol	Name	Operation	Analogy
	NO contact (contact)	„Conducts” if A=1	Normally open pushbutton
	NC contact (negated contact)	„Conducts” if A=0	Normally closed pushbutton

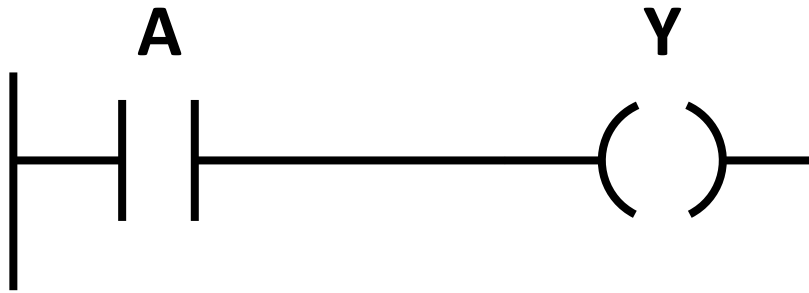


Coils

Symbol	Name	Operation	Analogy
	NO coil, coil	Sets Y to 1 if „powered”, to 0 otherwise	Normally open relay circuit
	NC coil, negated coil	Sets Y to 0 if „powered”, to 0 otherwise	Normally closed relay circuit



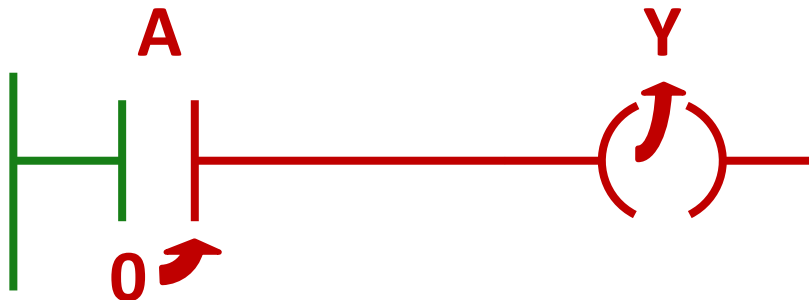
Evaluation of contacts and coils



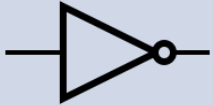
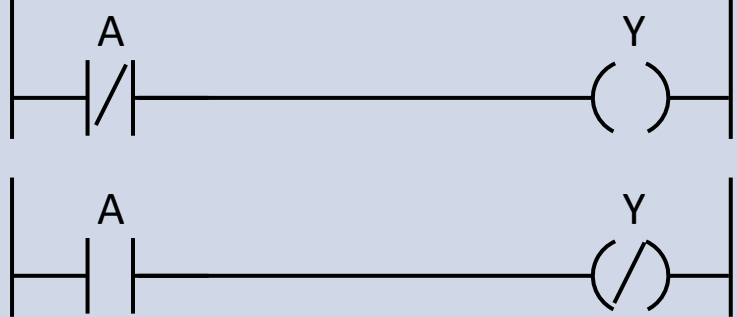

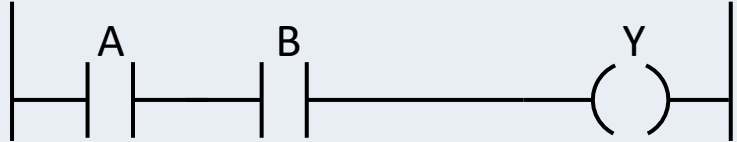


IF (A=1) THEN Y:=1 ❌




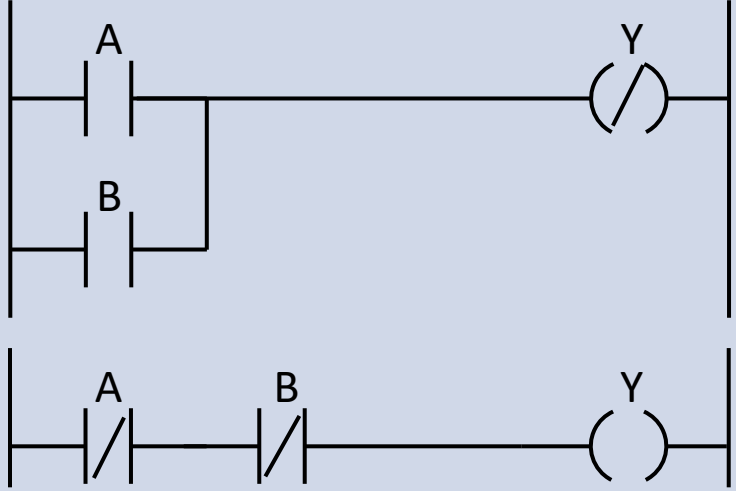

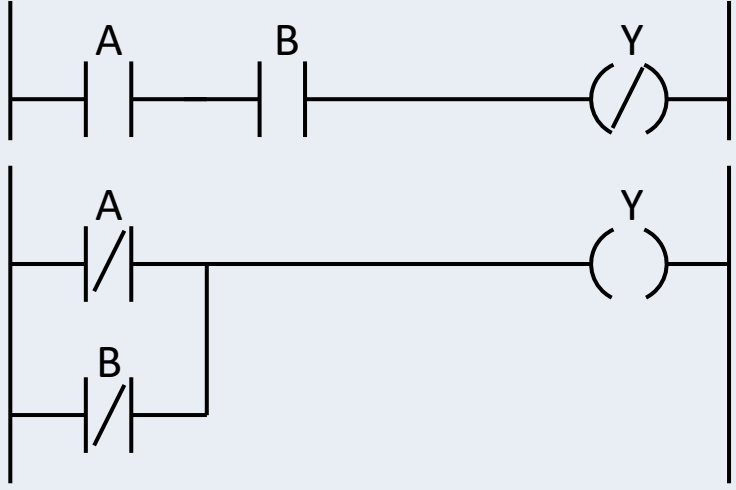
IF (A=1) THEN Y:=1
ELSE Y:=0




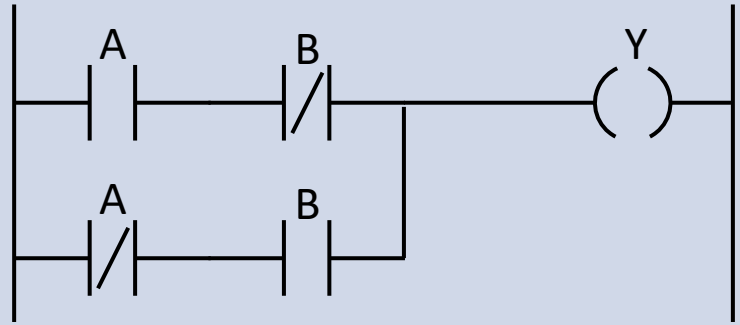

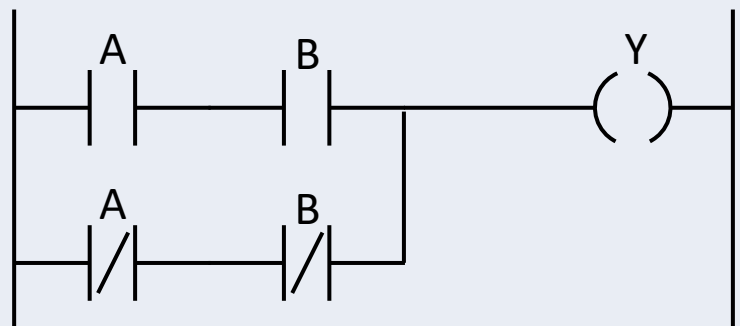
Boolean logic

Operation	Symbol	Algebraic notation	Ladder logic realization
NOT		$Y = \bar{A}$	
AND		$Y = A \cdot B$ $Y = A \& B$	
OR		$Y = A + B$	

Boolean logic

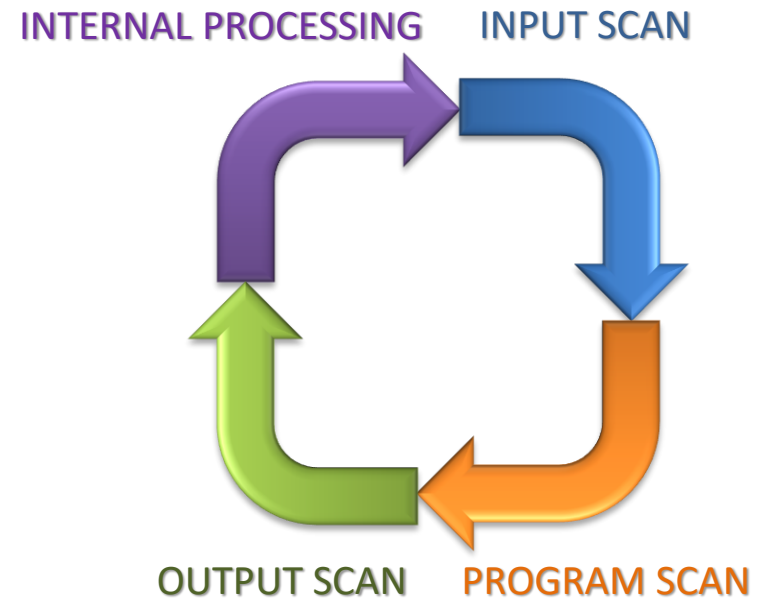
Operation	Symbol	Algebraic notation	Ladder logic realization
NOR		$Y = \overline{A + B}$ $Y = \bar{A} \cdot \bar{B}$	
NAND		$Y = \overline{A \cdot B}$ $Y = \bar{A} + \bar{B}$	

Boolean logic

Operation	Symbol	Algebraic notation	Ladder logic implementation
Antivalence (XOR)		$Y = A \oplus B$	
Equivalence (NXOR, EOR)		$Y = \overline{A \oplus B}$ $Y = A \odot B$	

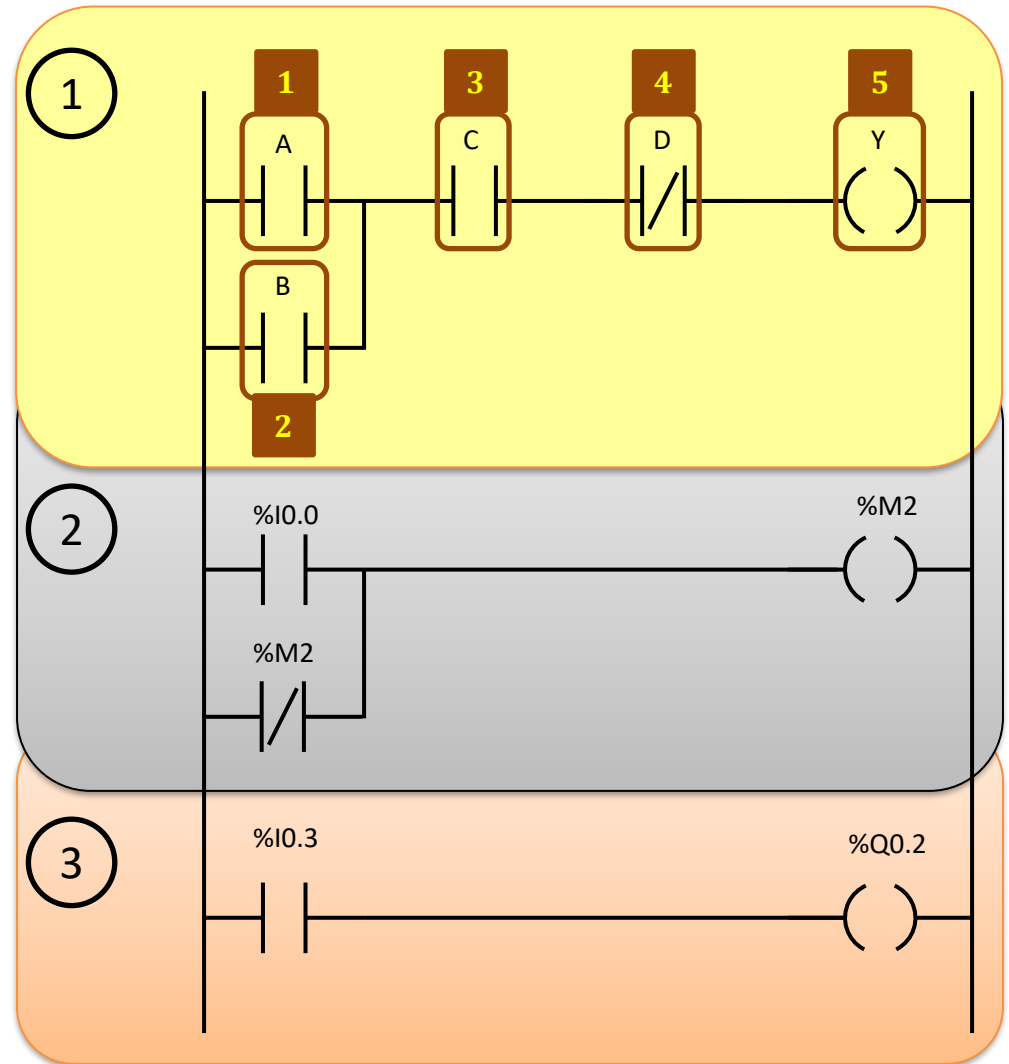
Evaluation of the ladder diagram

- Programs are executed as part of the PLC cycle
- The whole code is executed during the program scan
- All rungs of the diagram are evaluated in each PLC cycle



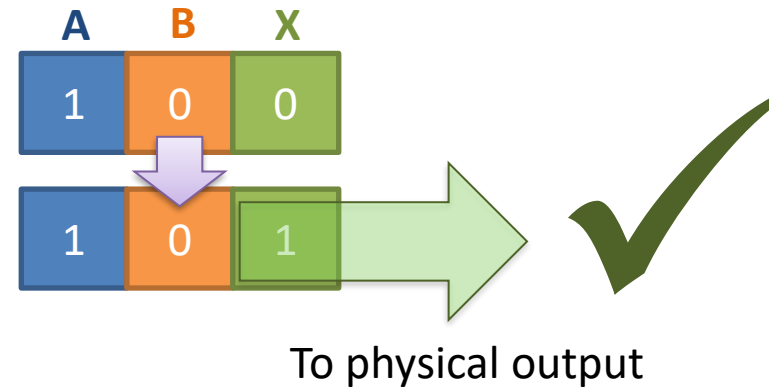
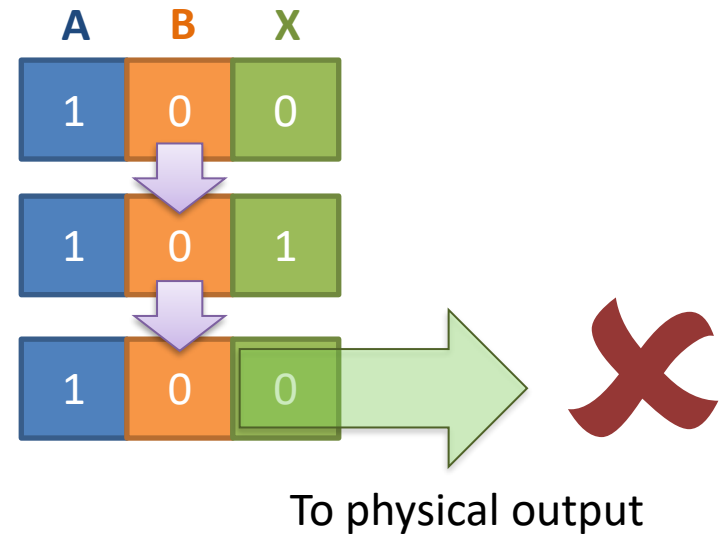
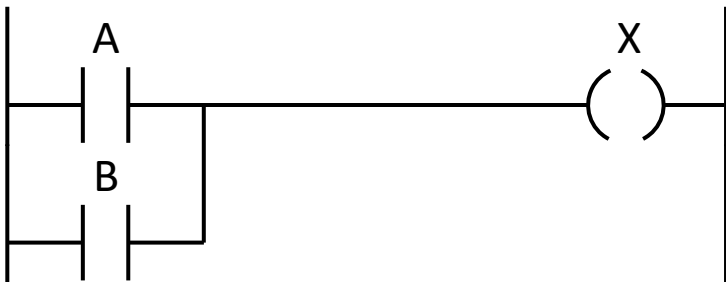
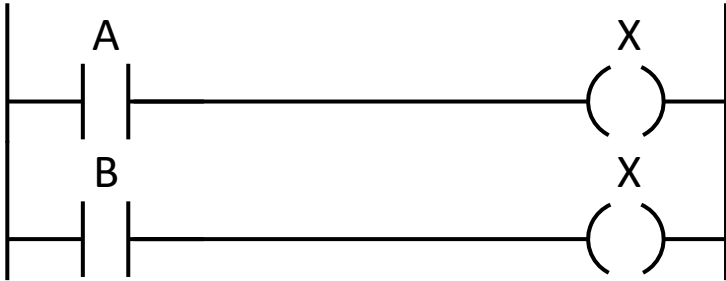
Evaluation of the ladder diagram

- Rungs are evaluated from the top to the bottom
- An element of a rung can be evaluated if all preceding elements (to its left) are evaluated



Effects of evaluation order

$X := A \text{ OR } B$

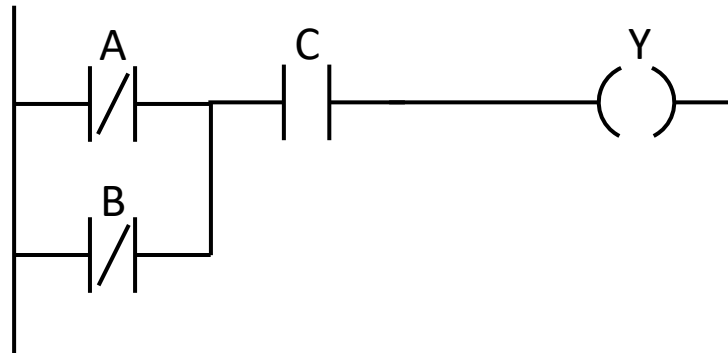


Simplification

$$Y = \overline{(A \cdot B)} \cdot C$$



$$Y = (\overline{A} + \overline{B}) \cdot C$$

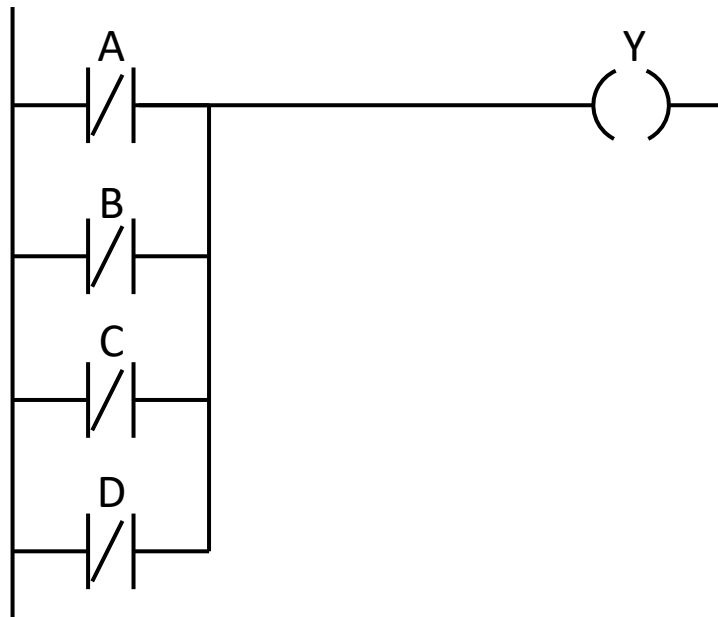


Simplification

$$Y = \overline{(A \cdot B)} + \overline{(C \cdot D)}$$

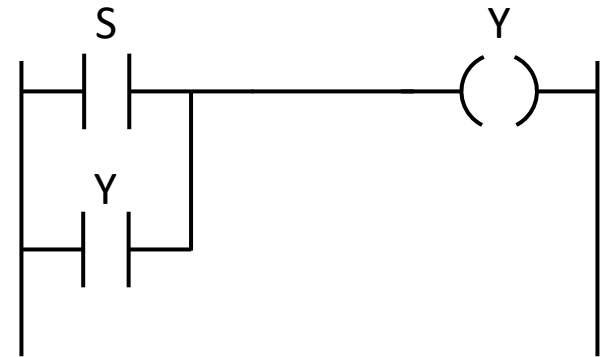
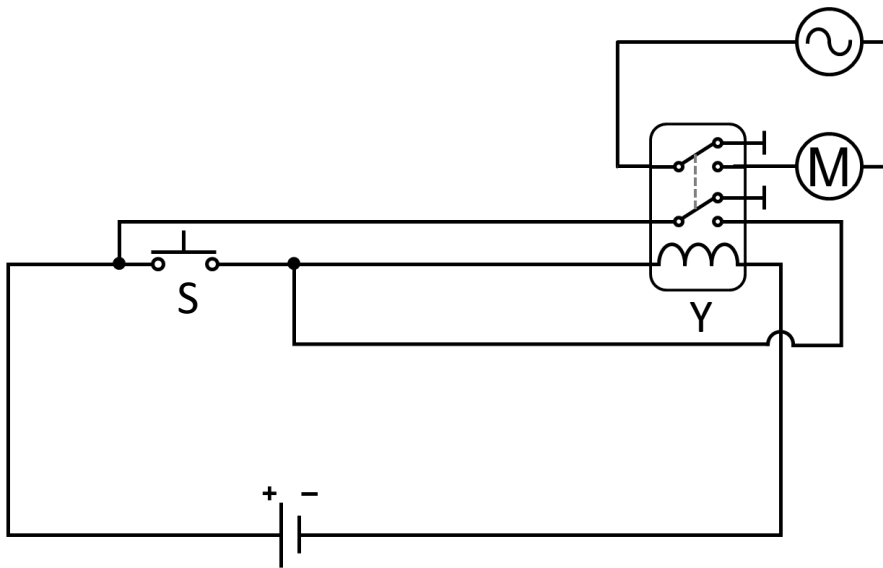


$$Y = \overline{A} + \overline{B} + \overline{C} + \overline{D}$$



Latching

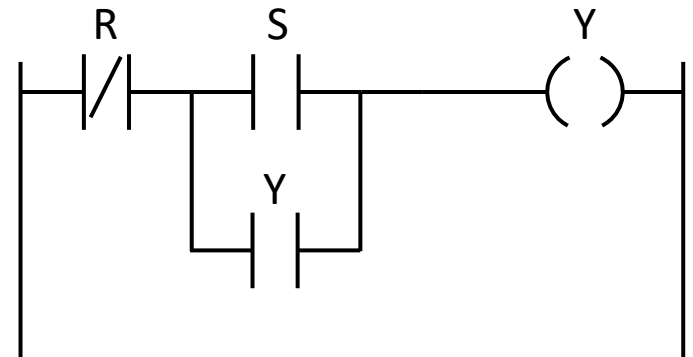
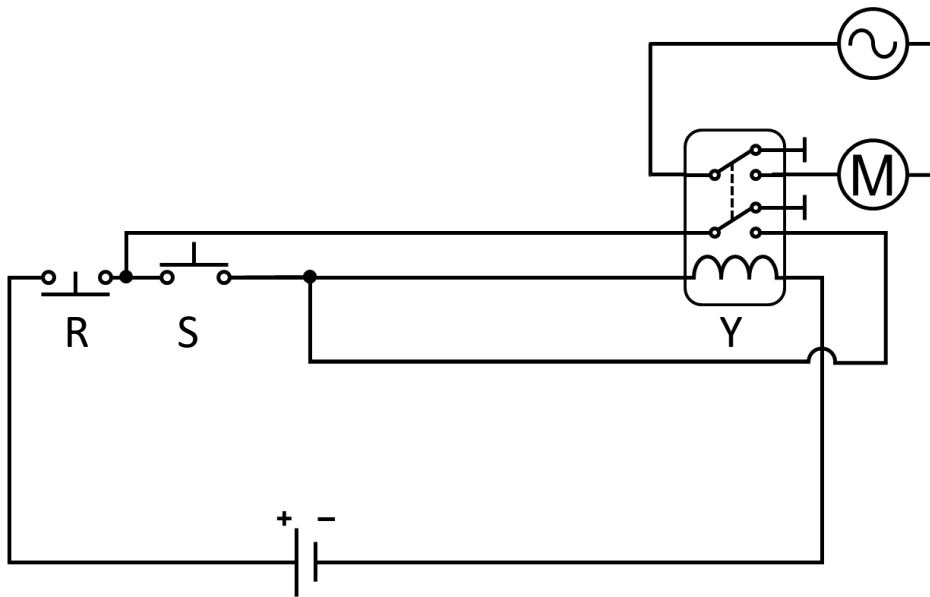
- The machine should start when a button is pressed but should continue its operation even after the button is released



$$Y = S + Y$$

Latching and unlatching

- The machine continues operating after the button is released
- How should we stop it?

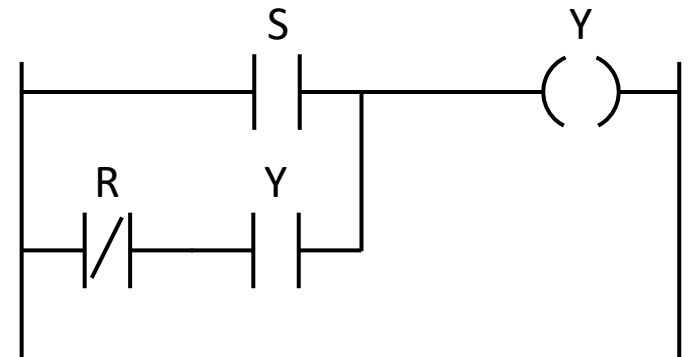
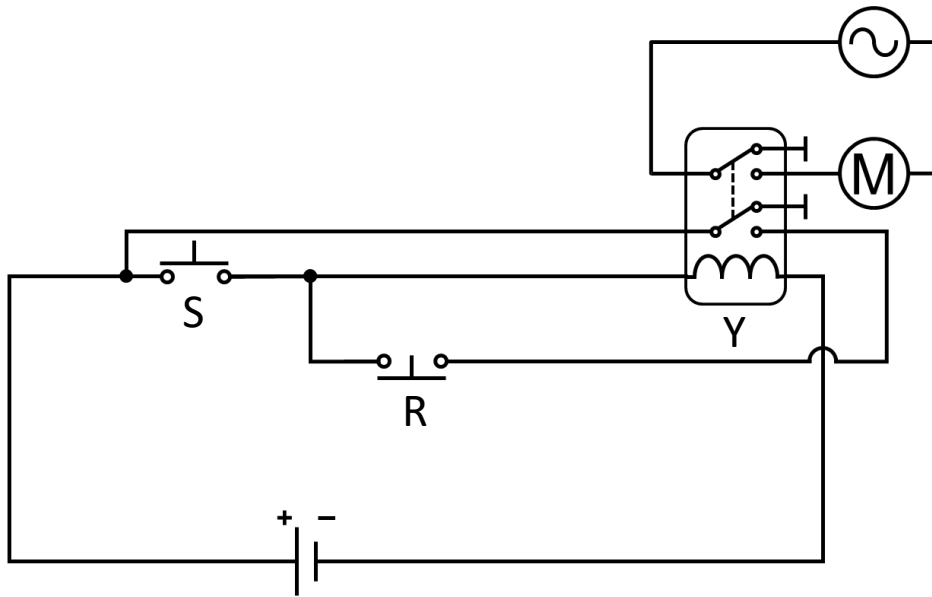


$$Y = \bar{R} \cdot (S + Y)$$

Latching and unlatching

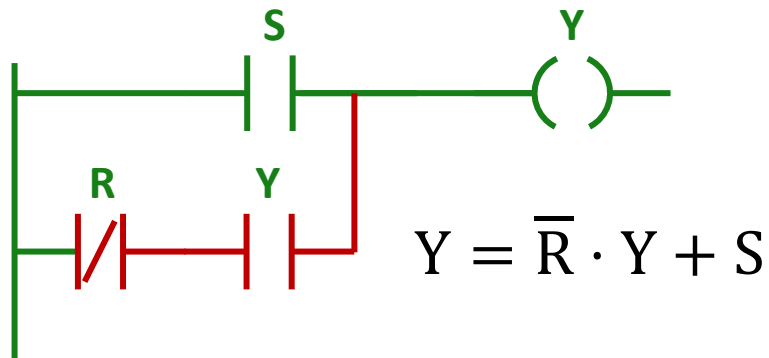
Alternative solution

- The machine continues operating after the button is released
- How should we stop it?

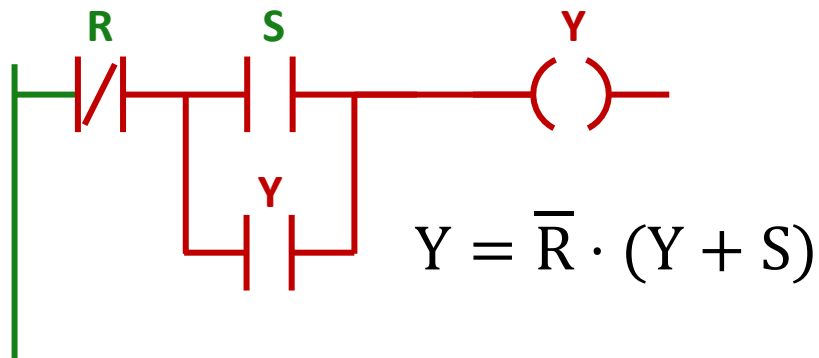


$$Y = \bar{R} \cdot Y + S$$

Latching and unlatching



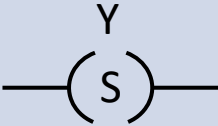
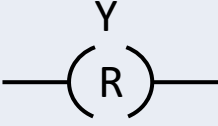
Overriding SET



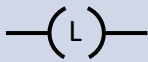
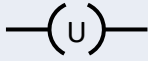
Overriding RESET

Latched coils

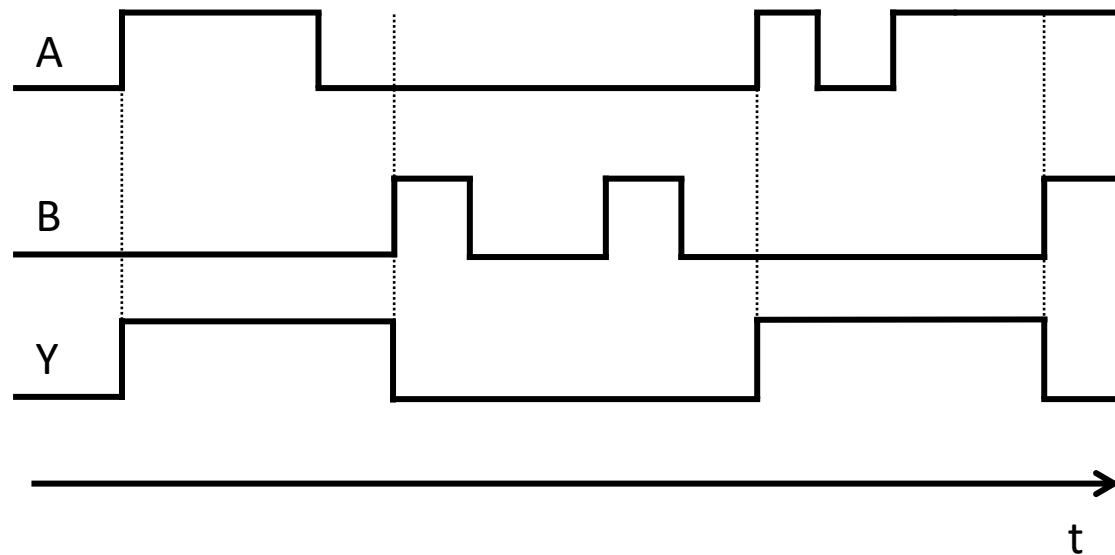
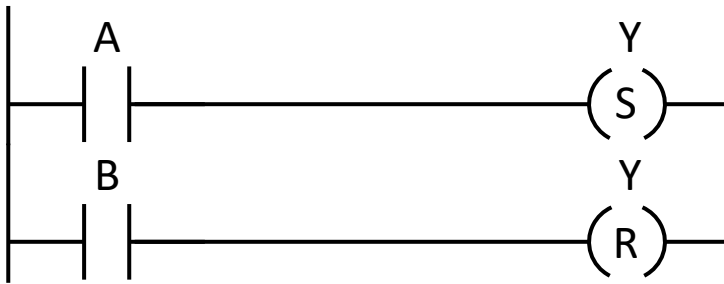
Standard notation

Symbol	Name	Operation
	Set coil (Latch coil)	Sets Y to 1 if „powered”, does not change its value otherwise
	Reset coil (Unlatch coil)	Sets Y to 0 if „powered”, does not change its value otherwise

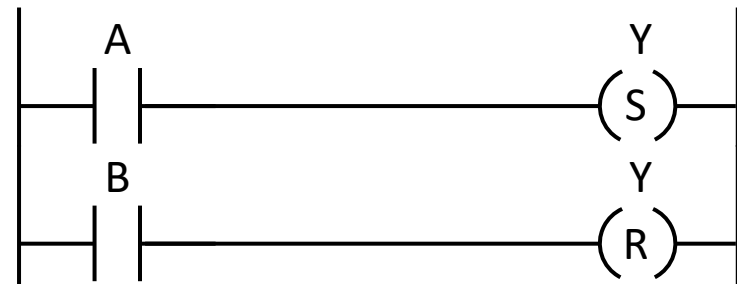
RSLogix notation

Symbol	Name
	Output Latch (OTL)
	Output Unlatch (OTU)

Latched coils



Interpretation of latching



IF (A=1) THEN Y:=1

IF (B=1) THEN Y:=0



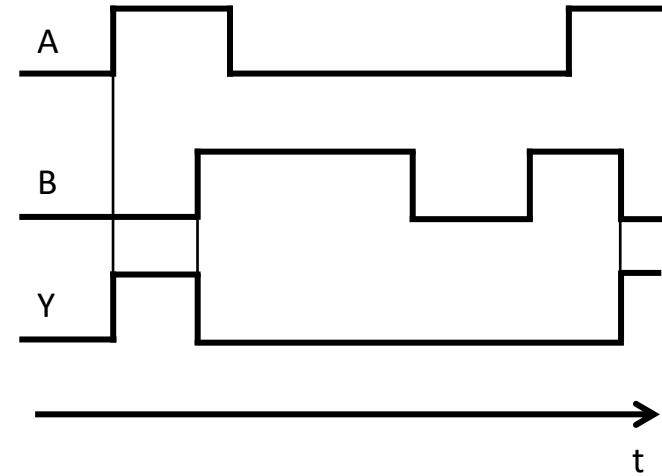
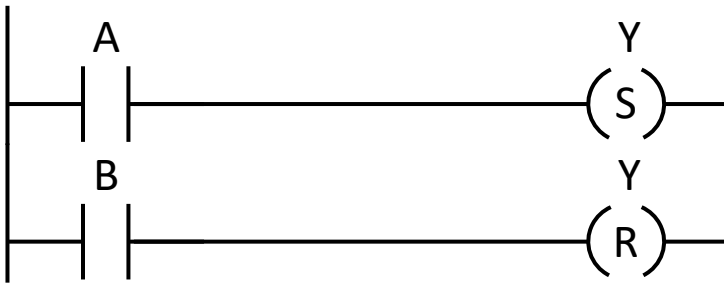
IF (B=1) THEN Y:=0

ELSE

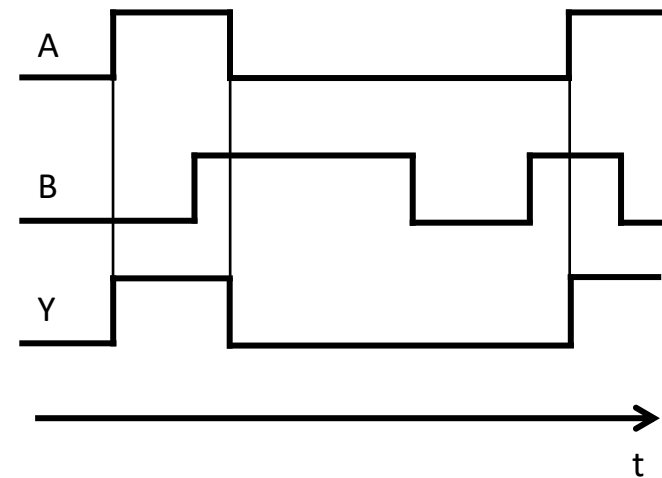
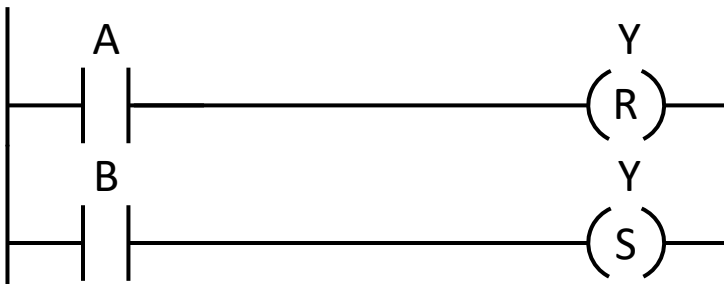
IF (A=1) THEN Y:=1

Set / Reset priority

Dominating Reset



Dominating Set

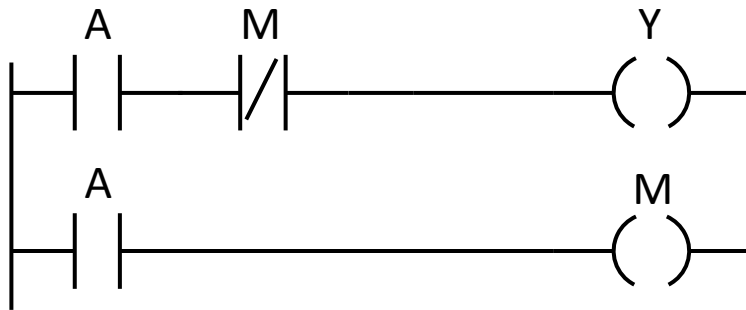


The latter operation (corresponding to the rung below) might overwrite the result of the former one!

Transition sensing

- How should we detect the transition of the value of a Boolean variable?
- Store its value in an auxiliary variable and during the next transition check whether a $0 \rightarrow 1$ or $1 \rightarrow 0$ transition has occurred!

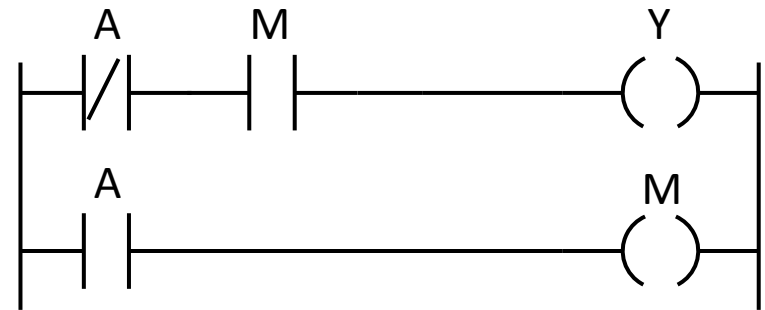
Rising edge sensing ($0 \rightarrow 1$)



$Y := A \text{ AND NOT } (M)$

$M := A$

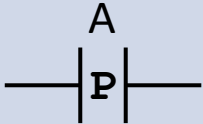
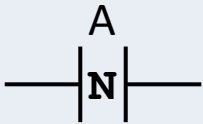
Falling edge sensing ($1 \rightarrow 0$)



$Y := \text{NOT } (A) \text{ AND } M$

$M := A$

Transition-sensing contacts

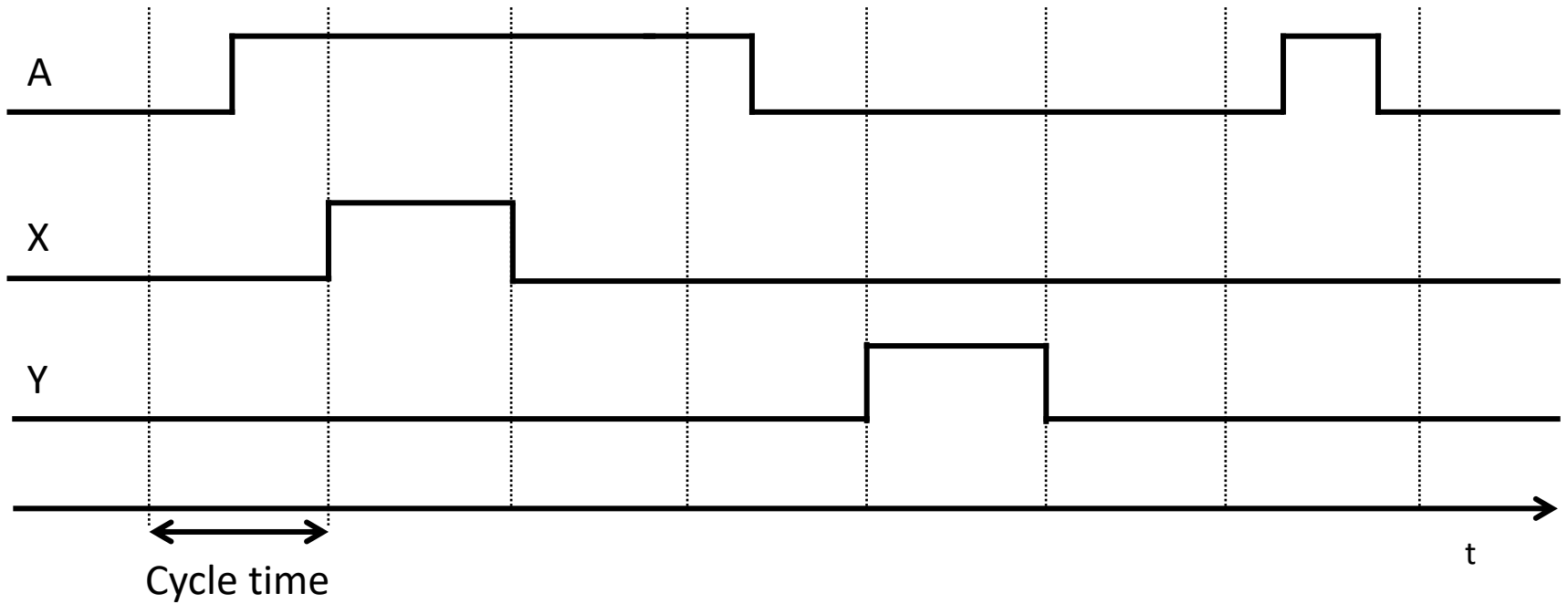
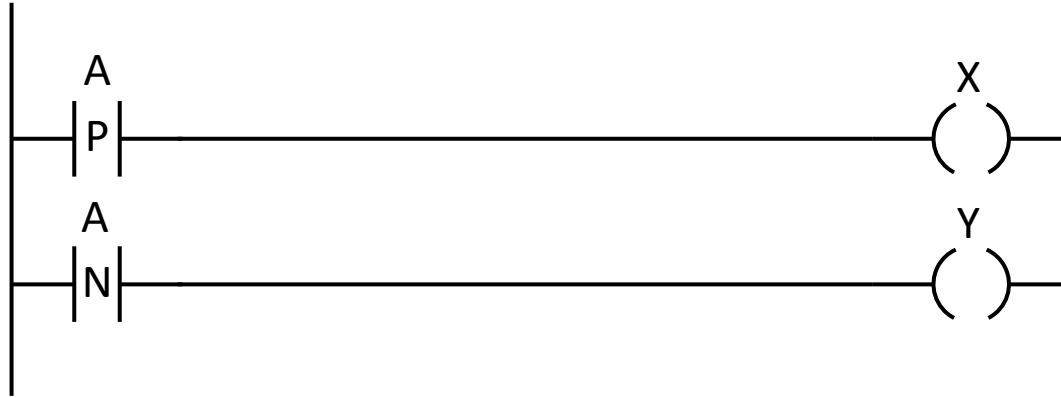
Symbol	Name	Operation
	Positive transition-sensing contact (rising edge contact , one shot rising)	„Conducts” if value of A • was 0 during the previous evaluation • is 1 during the current evaluation
	Negative transition-sensing contact (falling edge contact, one shot falling)	„Conducts” if value of A • was 1 during the previous evaluation • is 0 during the current evaluation

Transition-sensing contacts conduct only until their next evaluation (next cycle)!

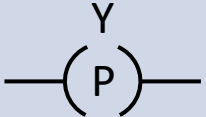
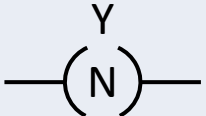
Notes

- Certain development environments (e.g. Siemens Step7) require the developer to specify explicitly the auxiliary variable in which the previous value of the variable is stored. In that case a different auxiliary variable should be assigned for each transition-sensing contact.
- Certain development environments (e.g. CoDeSys 2) does not support transition-sensing contact. Transition detection can be realized by function blocks in such environments.

Transition-sensing contacts

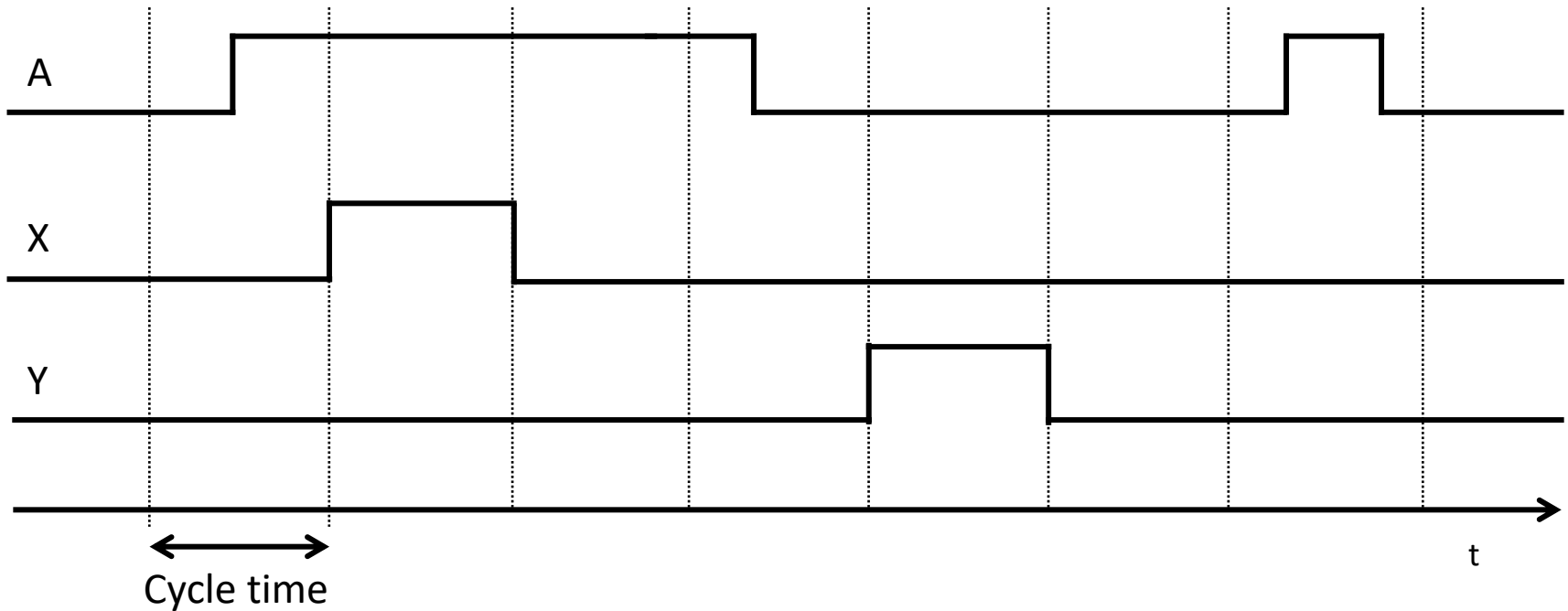
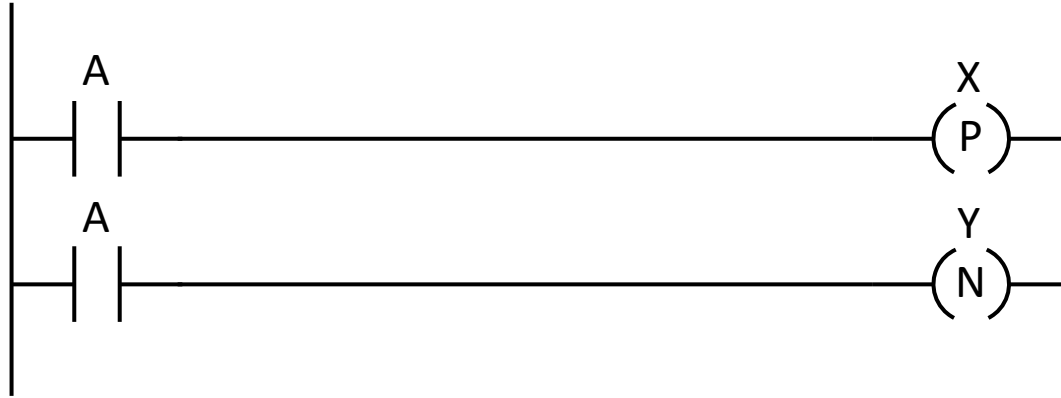


Transition-sensing coils

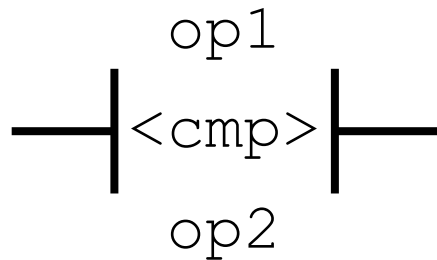
Symbol	Name	Operation
	Positive transition-sensing coil	Sets Y to 1 until its next evaluation if detects a 0→1 transition of its “power supply”
	Negative transition-sensing coil	Sets Y to 1 until its next evaluation if detects a 1→0 transition of its “power supply”

Transition-sensing contacts are standard elements but are not used in the practice and hence not supported by most development environments.

Transition-sensing coils



Compare contacts



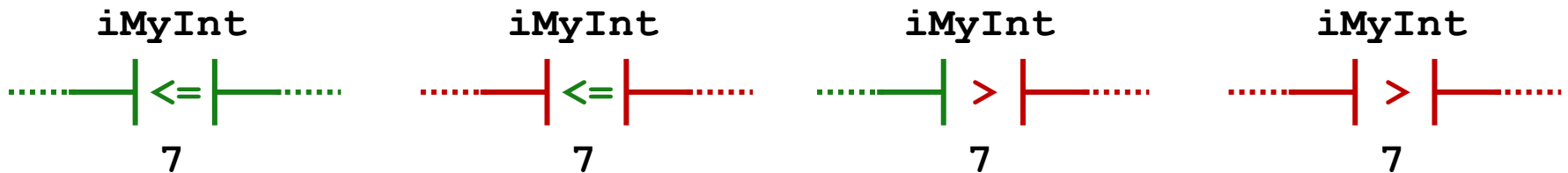
`op1, op2` : non-Boolean operands

`<cmp>` : comparison operators

- `=, <>`
- `<, <=`
- `>, >=`
- etc

The contact conducts if the relation `op1 <cmp> op2` is true.

Value of `iMyInt` is 4



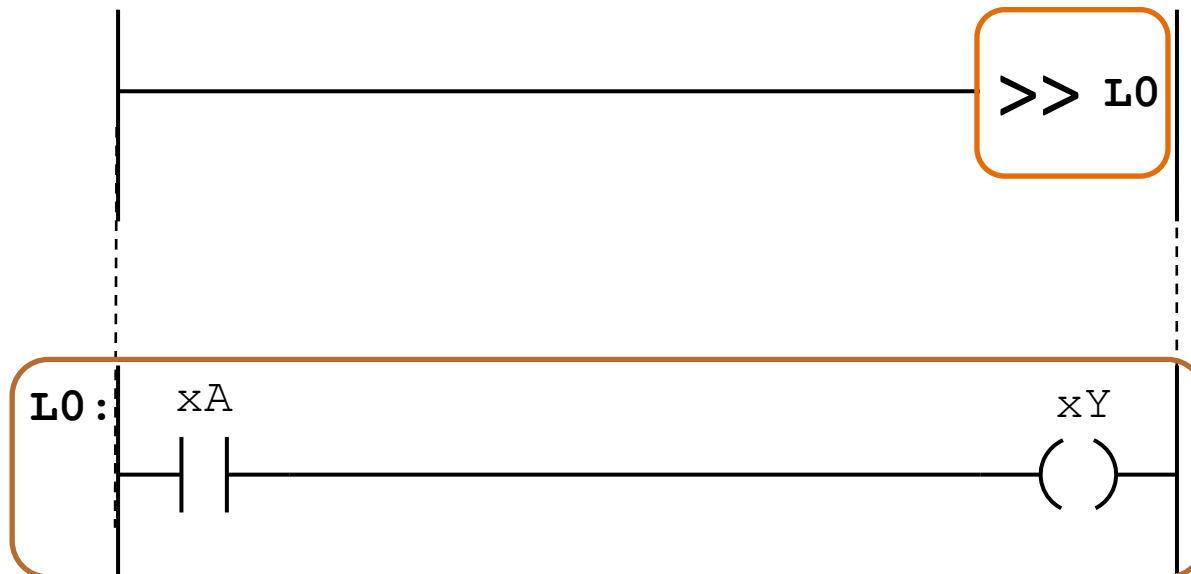
The Boolean variable associated with a compare contact is the result (return value) of a comparison function, e.g. `LEQ(iMyInt, 7)`. Compare contacts are available in only a few development environments.

Control flow instructions

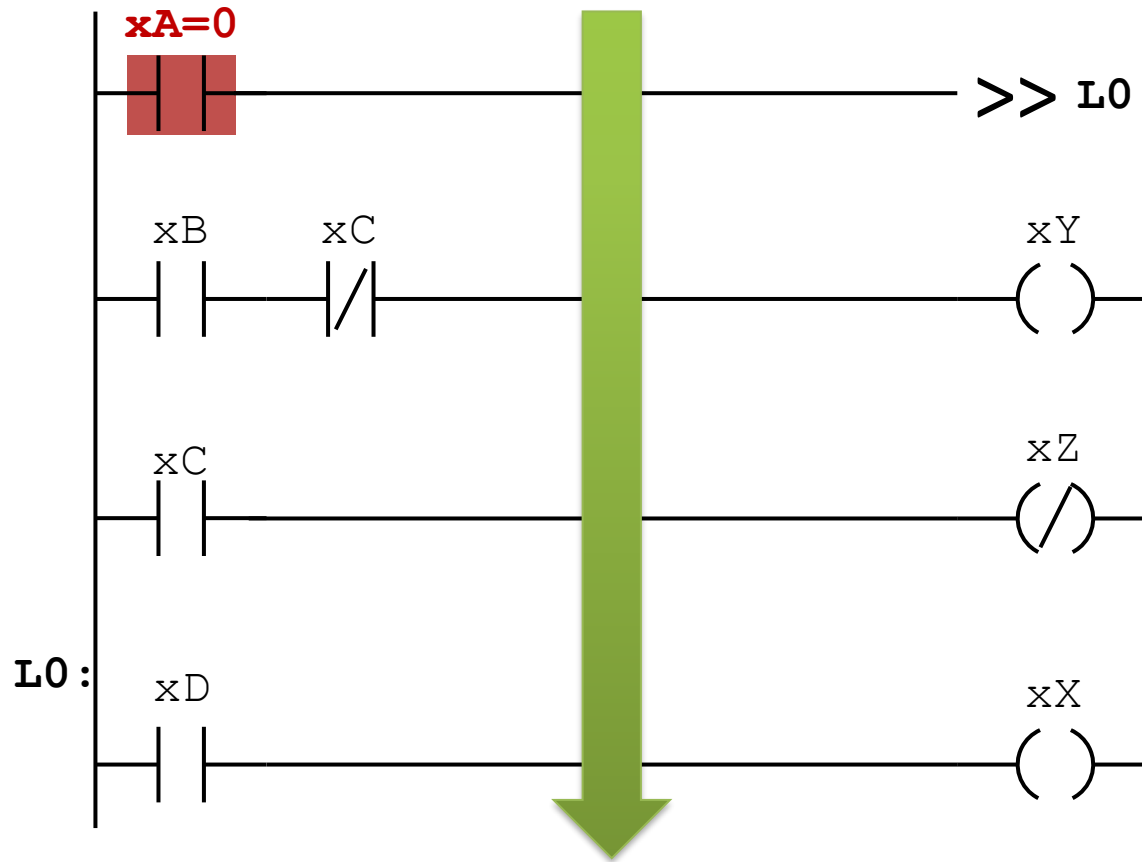
- Control flow instructions allow interrupted or nonlinear execution of programs
- Such instructions act on the program scan phase of the PLC cycle
 - no effect on input scan and output scan
 - input map and output map are treated as usual

Jump instruction

- Unique labels can be assigned to rungs (networks)
- A jump instruction results in continuing the execution of the program from the referenced rung
- A jump instruction can be inserted to a rung as a coil

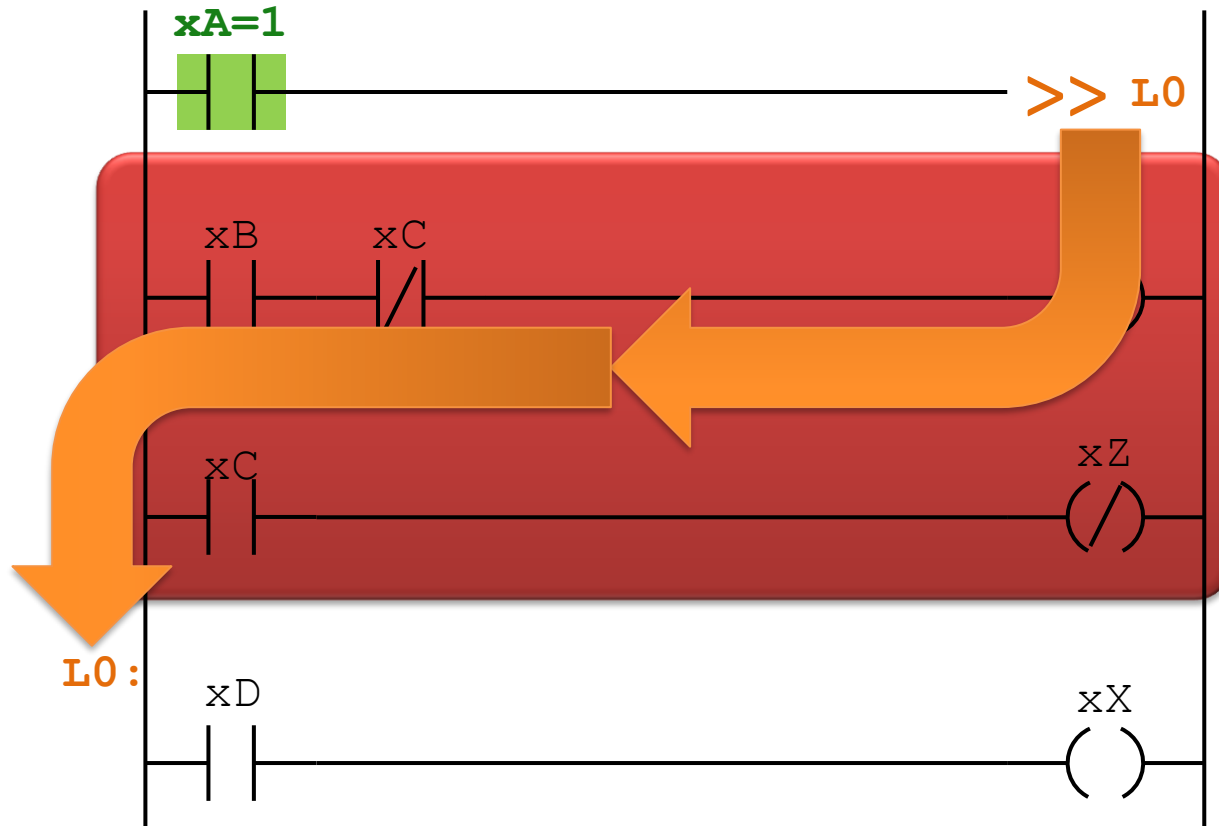


Jump instruction



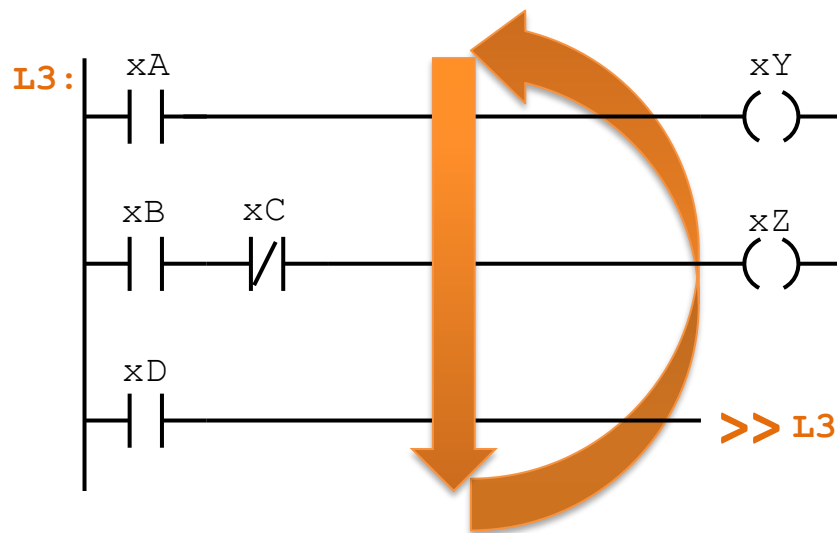
Since the value of xA is 0, the jump is not executed, all rungs are evaluated.

Jump instruction



Since the value of $\bar{x}A$ is 1, a jump is executed and the two rungs marked by red are not evaluated.

Jumping back



- Danger of creating an infinite loop
- Shall be never used (with some exceptions)

Jump instruction

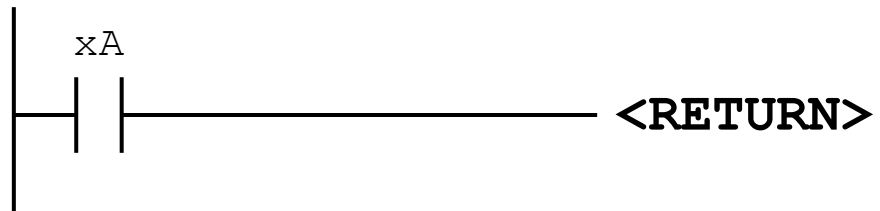


- **Jump instructions deteriorate the cycle time significantly**
- **In case of any error, the maximal cycle time might be exceeded**

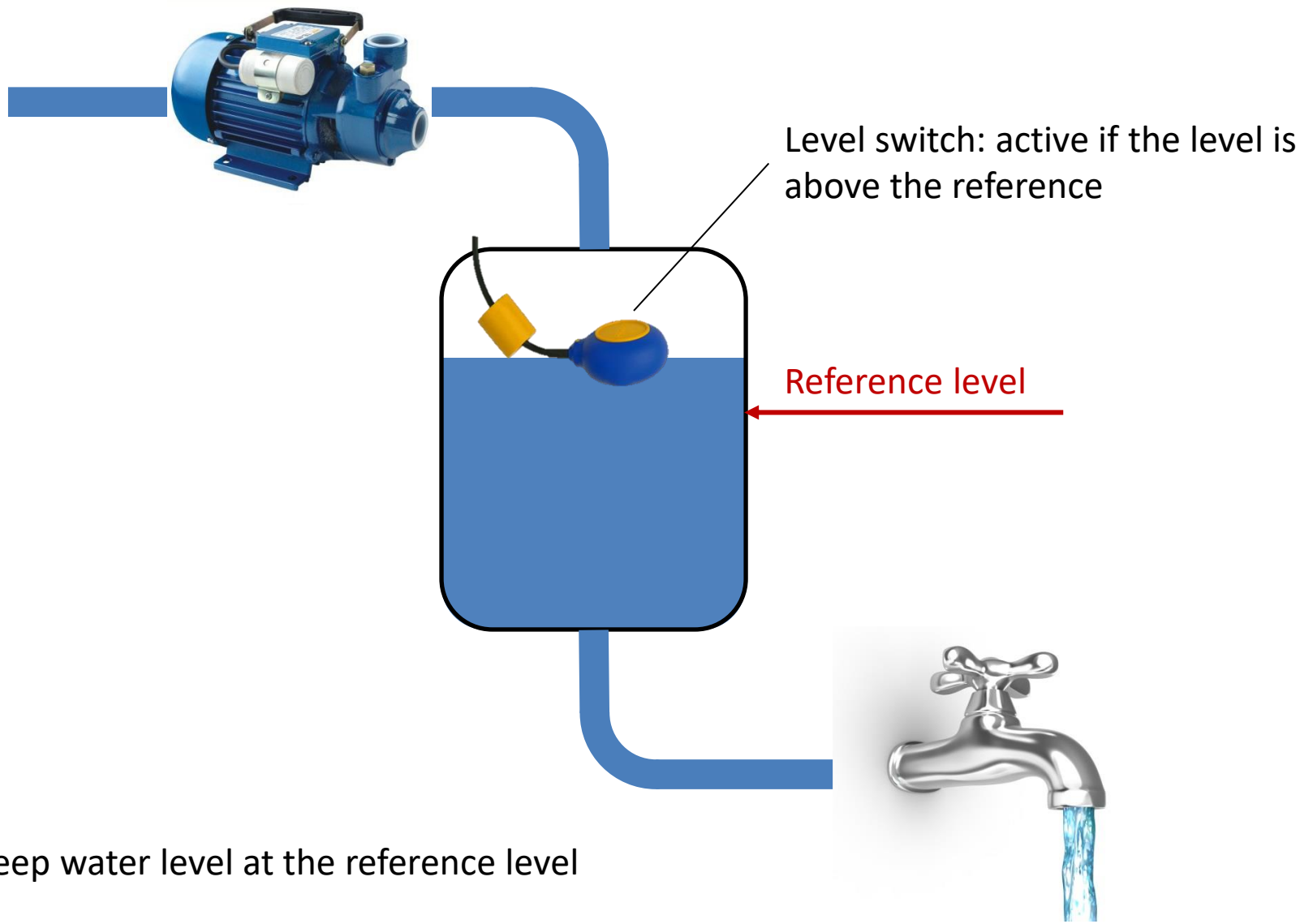
Use of jump instructions is strongly discouraged – features of standard program organization units shall be used instead.

Return

- Immediate return to the caller POU in case of functions and function blocks (further rungs are not evaluated)
- Termination of execution in case of programs (further rungs are not evaluated)
- Might be inserted to a rung as a coil
- Result (return value) must be set beforehand in case of functions
- Shall be used exceptionally

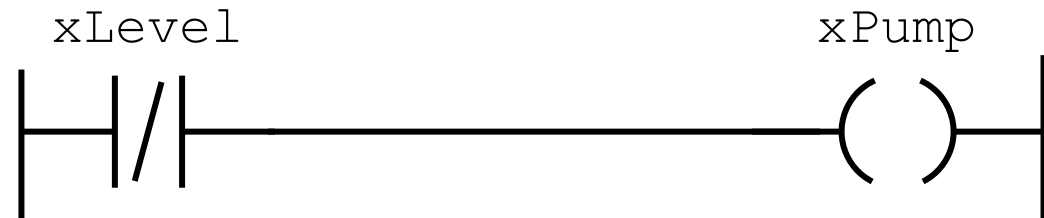
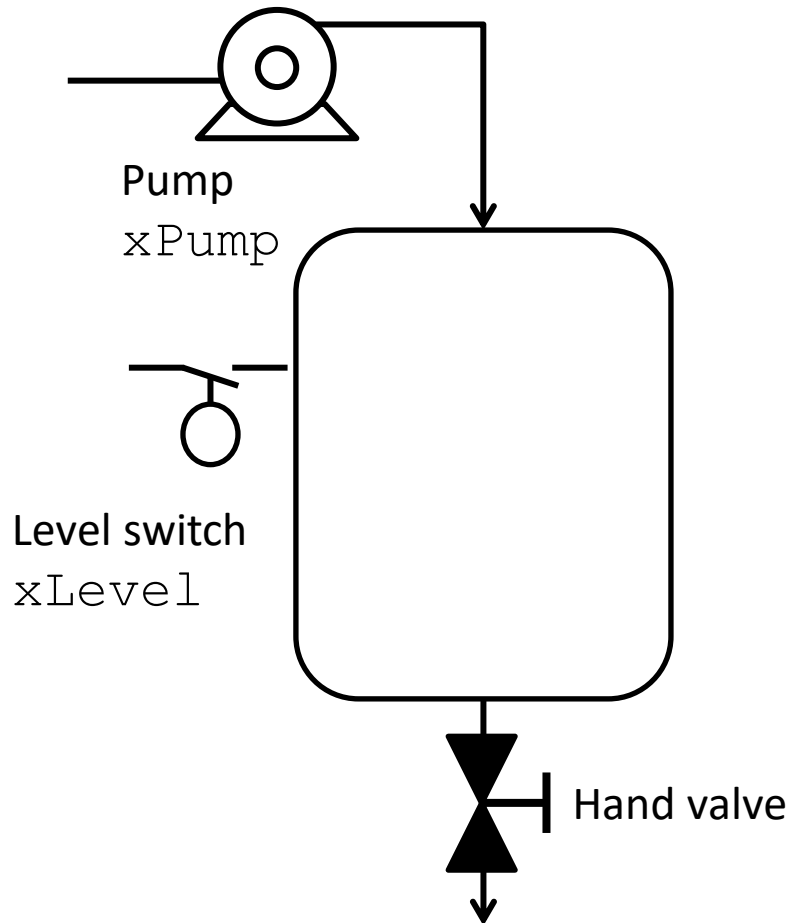


Problem

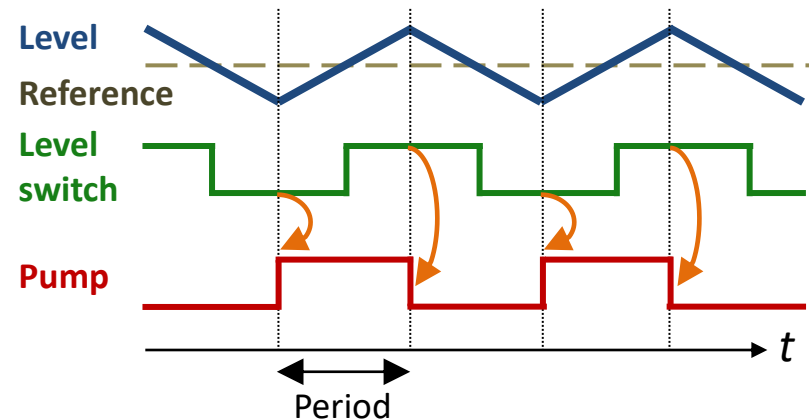


Task: keep water level at the reference level

Solution

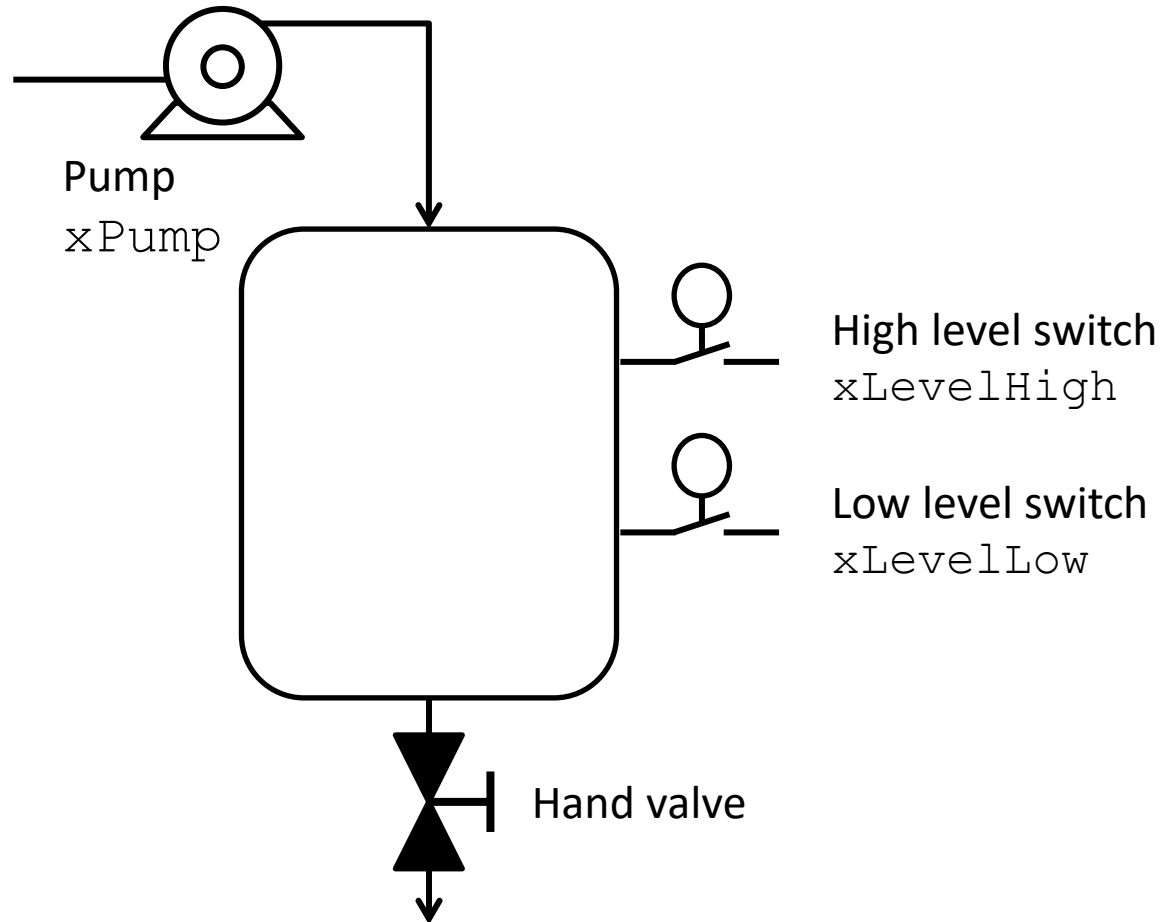


Problem: If the hand valve is opened, the pump is switched on and off with a high frequency, leading to damage of the equipment.

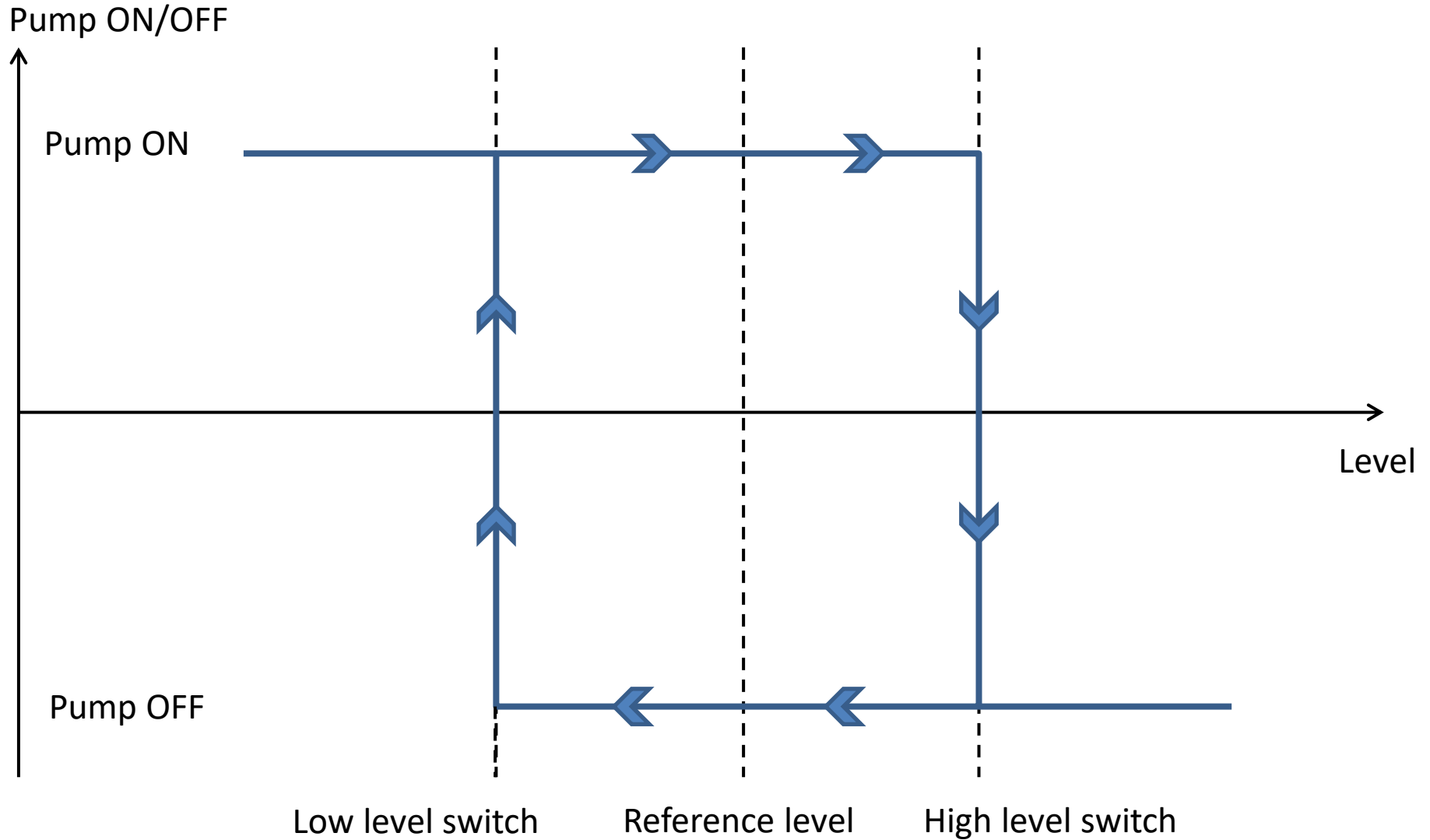


The figure above shows the case of periodic execution of the PLC program with cycle time neglectable compared to the period

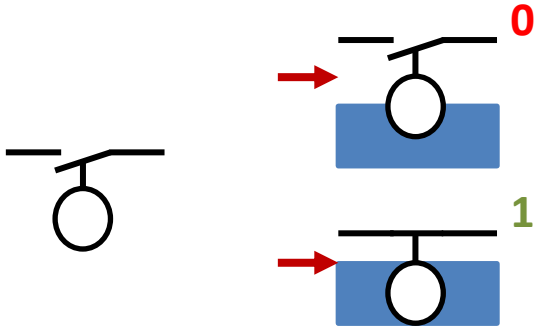
Problem – a better solution



Hysteresis control



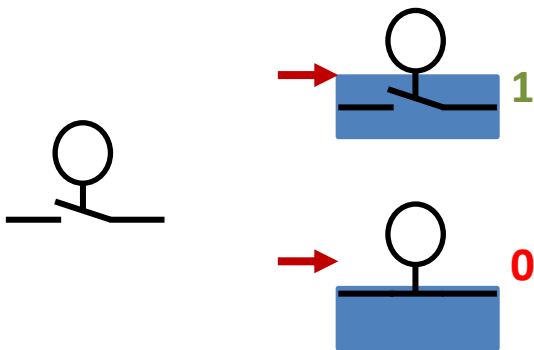
How to install the level switches?



Level switch provides an **active** signal if the level of the liquid is **low**.

In case of wire break, the inactive signal means the level is low, the pump is switched on.

If no safety equipment is installed, the tank is overfilled, which leads to leaks and flooding the environment of the tank.



Level switch provides an **active** signal if the level of the liquid is **high**.

In case of wire break, the inactive signal means the level is high, the pump is switched off.

The system stays in a safe state.



Problem – a better solution

