UNIVERSITÀ
DEGLI STUDI
DI BERGAMO

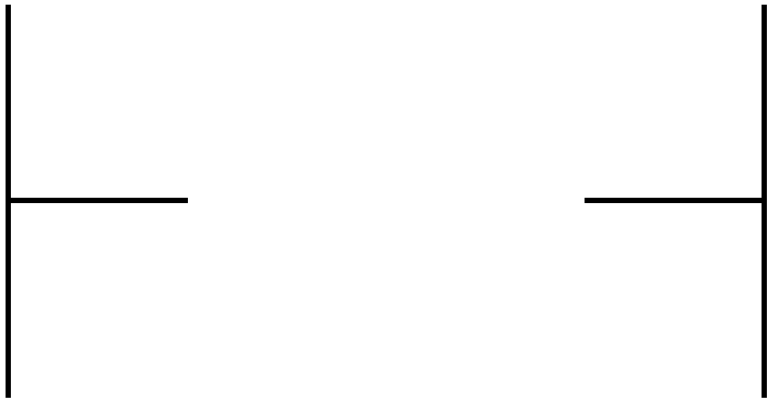
Data Science and Automation

Lesson 18

PLC - Ladder

Basic idea

Ladder is a language contact-based: rungs, composed of various elements, are built between the two sides.



In this language it is supposed that a current flows, depending on the input values, from the left side to the right side. The current sets the output values of the PLC.

Basic rules

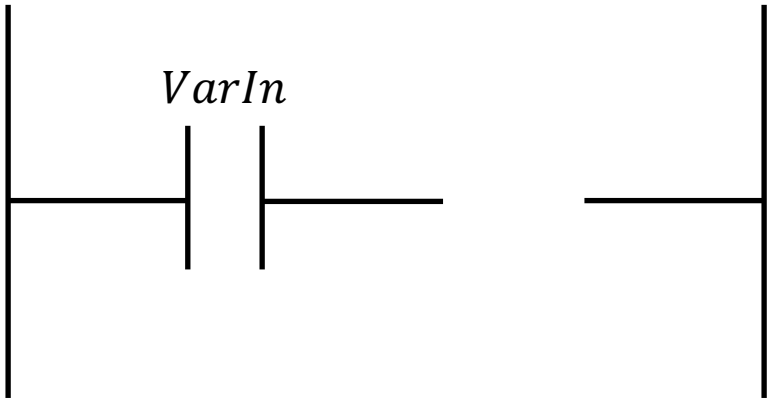
1. The current flows only from left to right.

2. Rungs are explored by the PLC from the first at the top, to the last at the bottom. That's why the order of the rungs is relevant.

3. The synchronization between the variables of the program and the actual input/output of the systems is realized by executing four steps:
    1. Input reading
    2. All the rungs are executed
    3. Output updating
    4. Restart operations

Ladder: Language elements

Contact

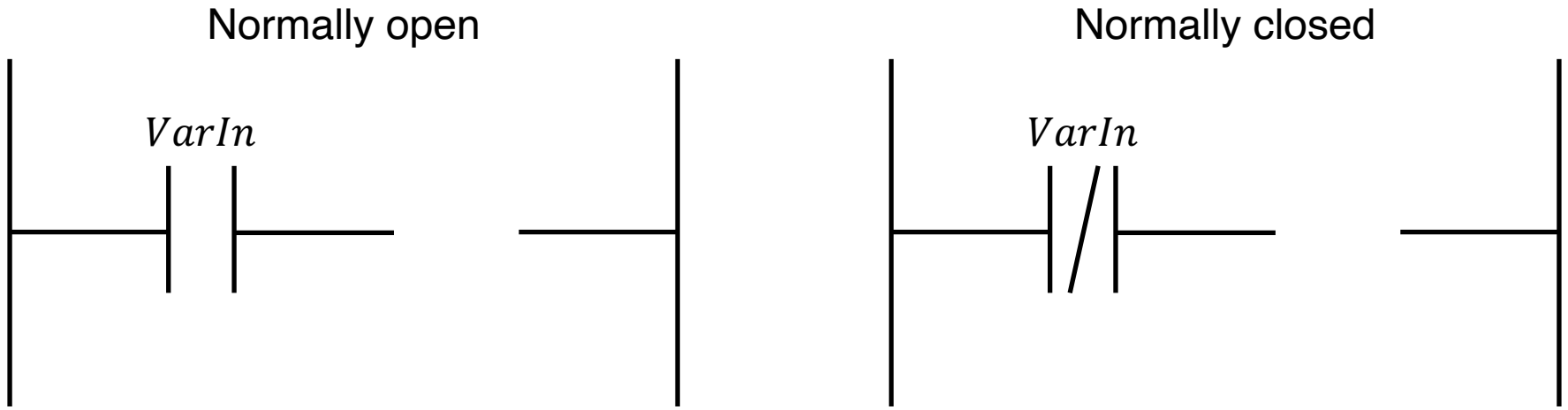


The contact is the element that brings continuity on the right side of the ladder. It is associated to a variable (*VarIn* in the example above).

Usually the contact is connected to the left side.

Ladder: Language elements

Normally open contact – Normally closed contact

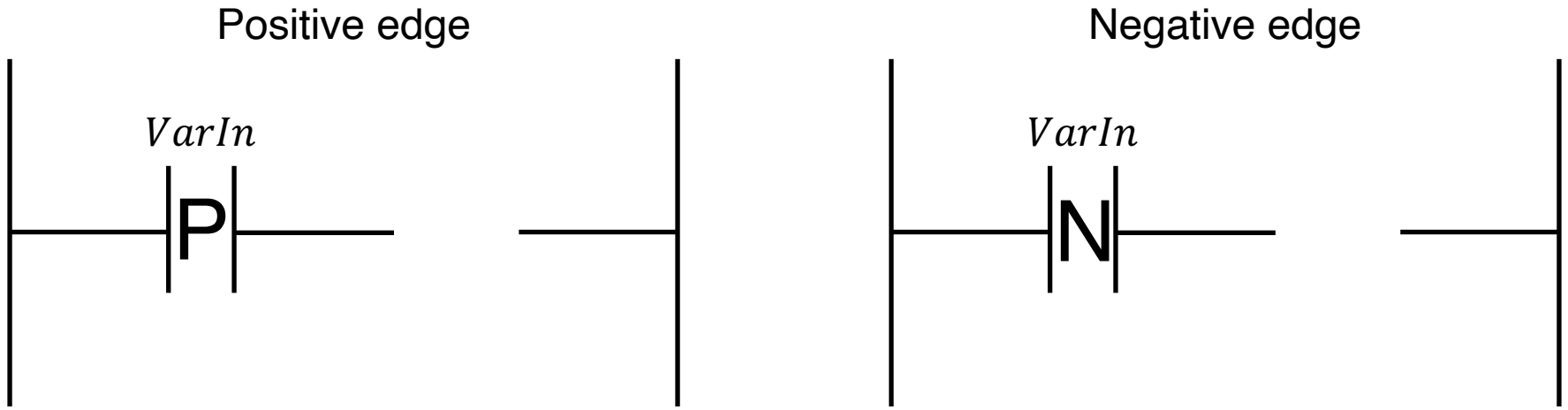


Normally open contact brings continuity on the right side if the value of the associated variable is 1.

Normally closed contact brings continuity on the right side if the value of the associated variable is 0.

Ladder: Language elements

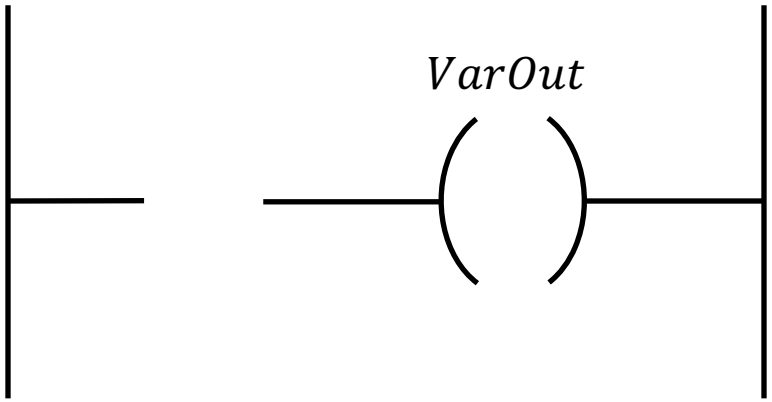
Contacts with edge recognition



The contact brings continuity on the right only during the cycle in which there is the positive (or negative) edge of the signal *VarIn* (from 0 to 1 for the P contacts, from 1 to 0 for the N contacts).

Ladder: Language elements

Coil

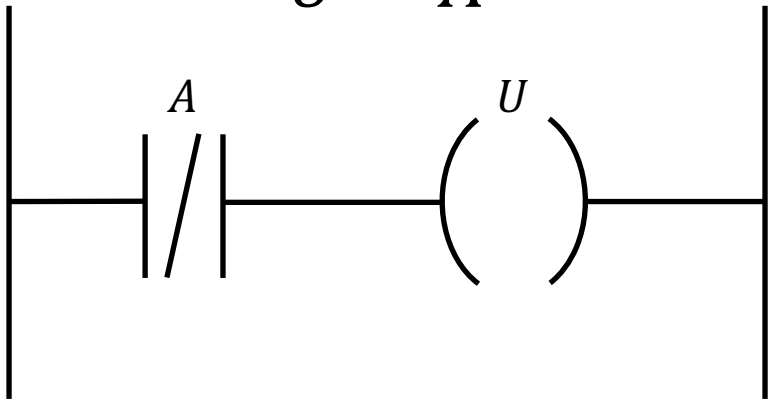


The coil is the final item in a ladder instruction. Usually it is connected to the right side of the ladder and, in case of continuity on its left side, sets to TRUE the value of the related variable *(VarOut)*.

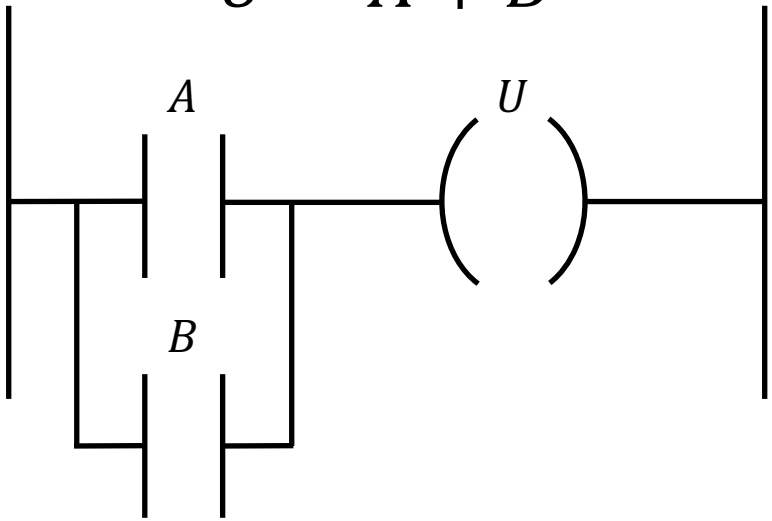
Example

Examples with contacts and coils

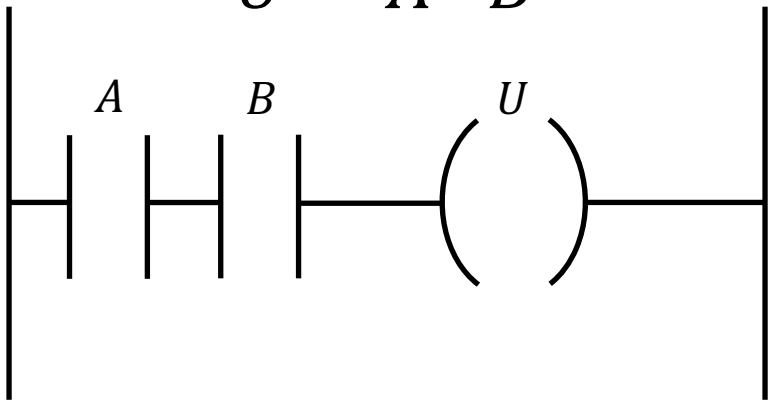
$U = \bar{A}$



$U = A + B$



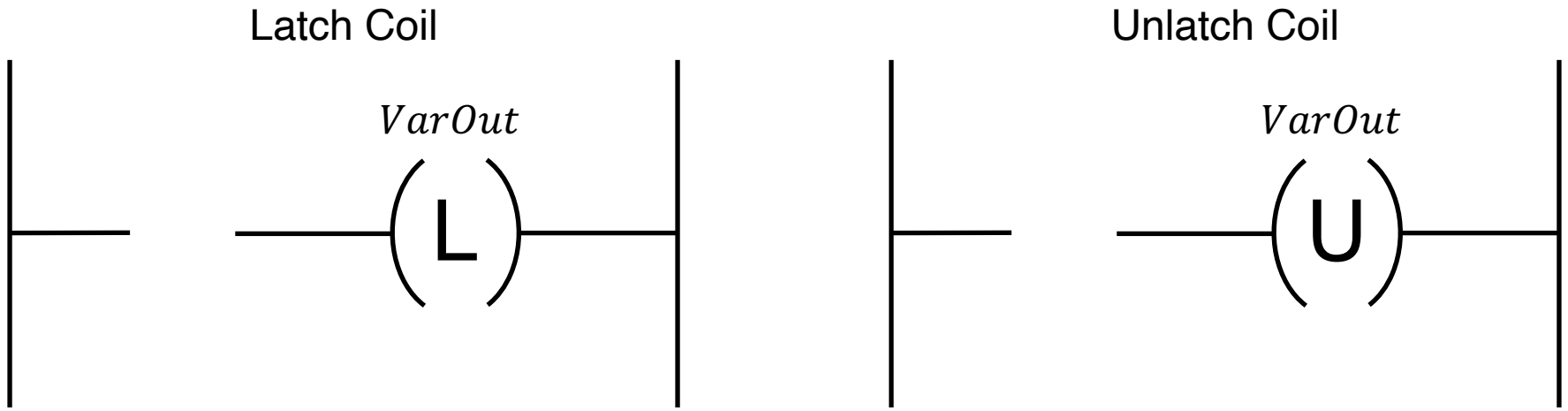
$U = A \cdot B$



N.B.: The declaration of the variables will be shown during the use of Automation Studio.

Ladder: Language elements

Latch/Unlatch coil



The latch coil sets the related *VarOut* if there is continuity on its left side and maintain the variable set even if the continuity is missing. To unset *VarOut* a unlatch coil has to be activated.

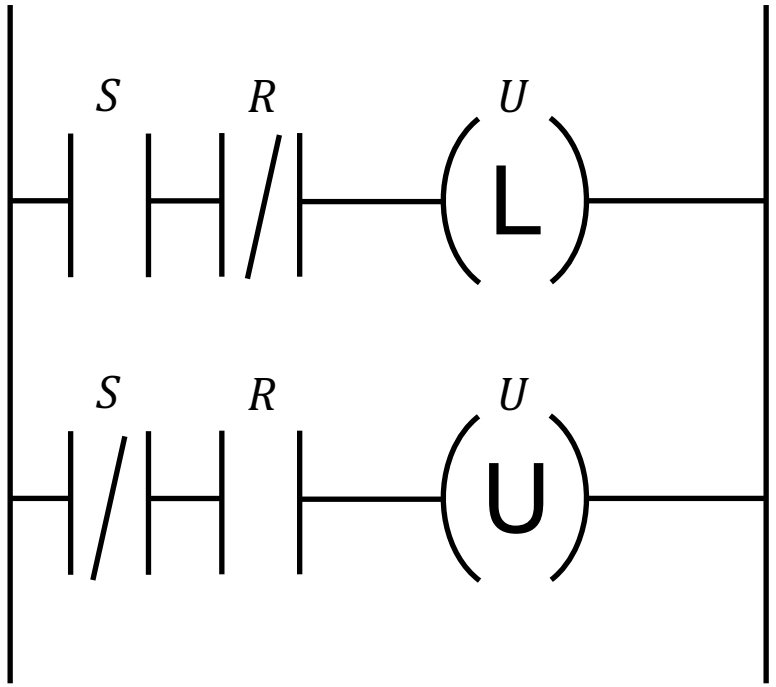
N.B.: In Automation Studio they are called Set/Reset

Example

Example with contacts, latch and unlatch coils

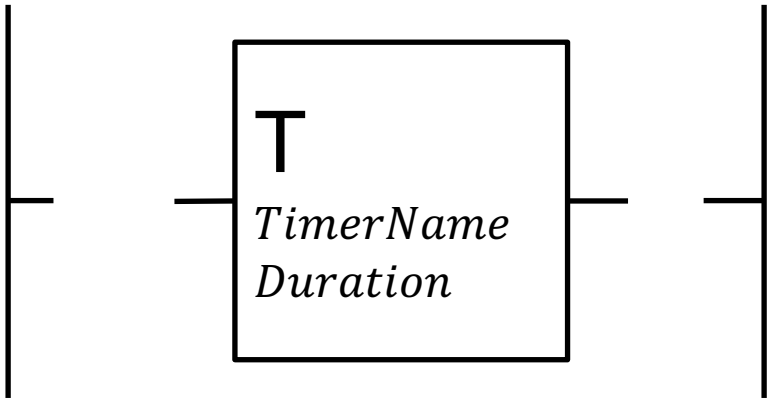
Create a program that simulates the behaviour of a set-reset flip-flop.

S	R	U
0	0	Uold
0	1	0
1	0	1
1	1	-



Ladder: Language elements

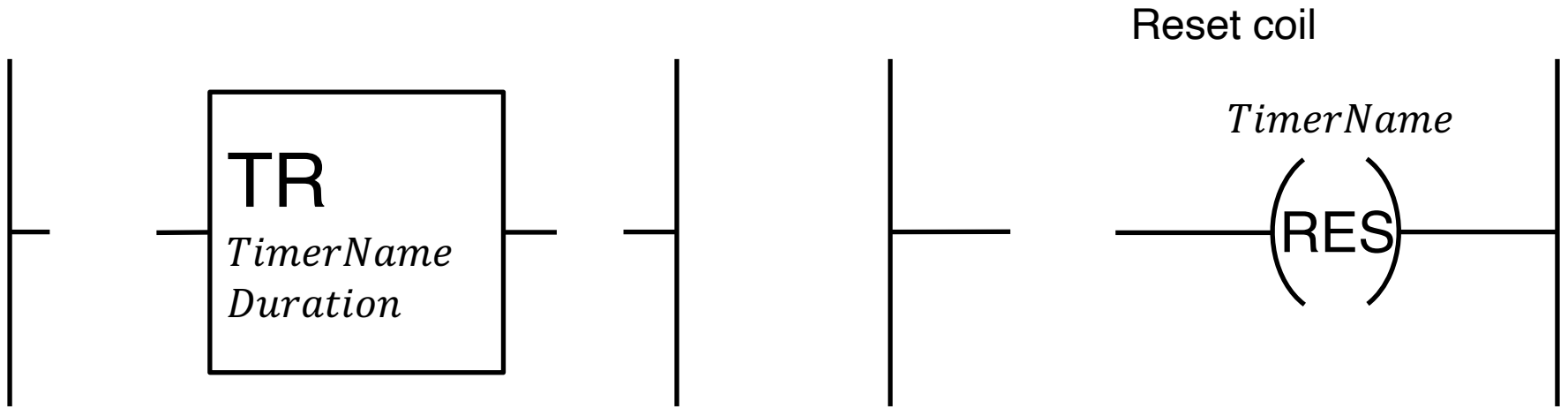
Timer



The timer is the component which performs a verification of the duration, starting from the instant in which there is continuity on the left side. When the value $Duration$ is reached, $TimerName$ is set to 1. If the continuity is missing on the left, the timer is reset (and $TimerName$is set to 0)

Ladder: Language elements

Retentive timer



The only difference between a retentive timer and a normal timer is that, if the continuity on the left side is missing, the value reached by the timer is maintained. The reset of its value can be realized only by a reset coil (on the right in the previous picture)

Example

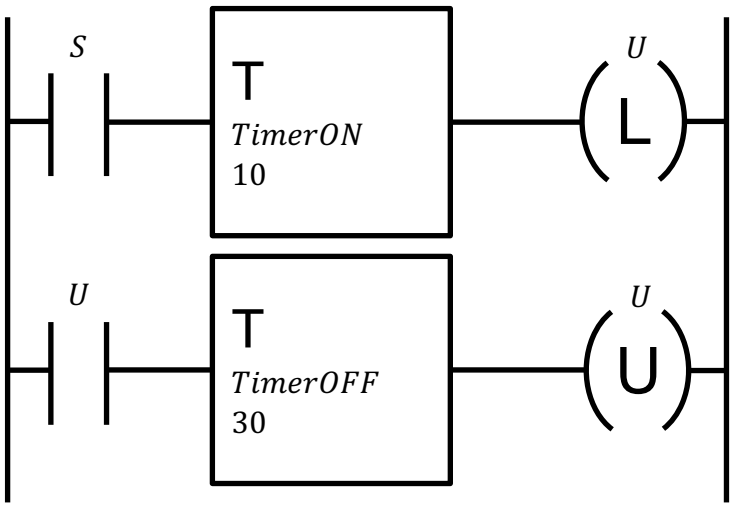
Example: Timer

Create a software that activates an electro-valve U if the overflow sensor S stays to TRUE for more than 10 s and maintains U active for 30 s

Solution n° 1

Let's think about this solution:

if S stays to TRUE for more than 40 s... What happens?



Example

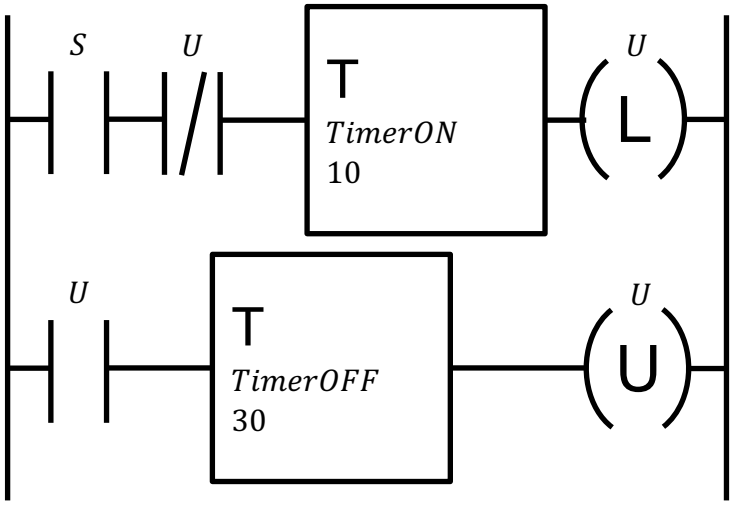
Example: Timer

Create a software that activates an electro-valve U if the overflow sensor S stays to TRUE for more than 10 s and maintains U active for 30 s

Solution n° 2

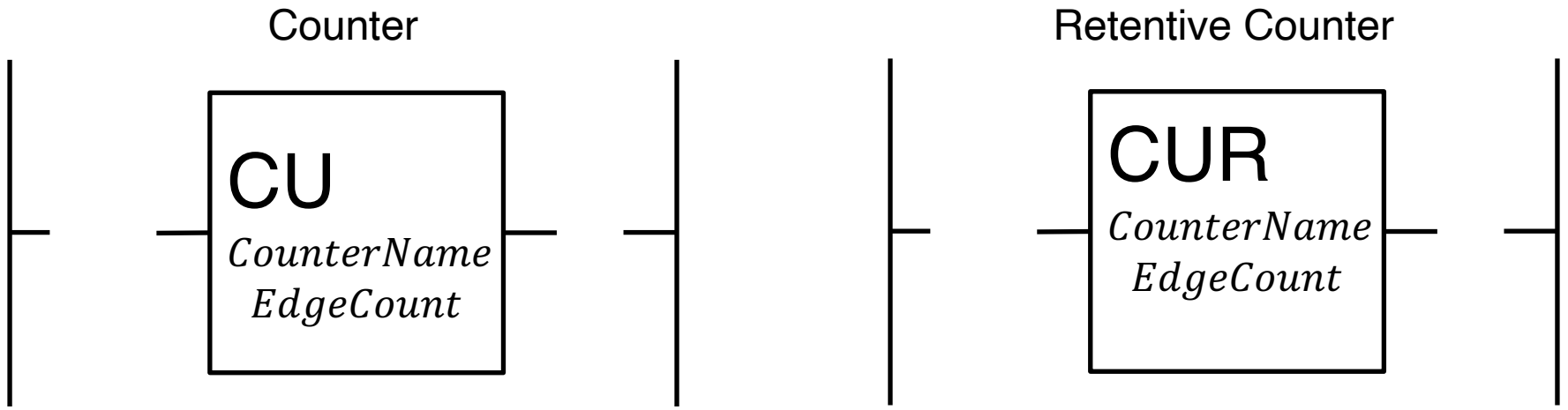
With this solution:

If S remains ON, after 30 s, U is deactivated and TimerON restarts its counting operations.



Ladder: Language elements

Counter

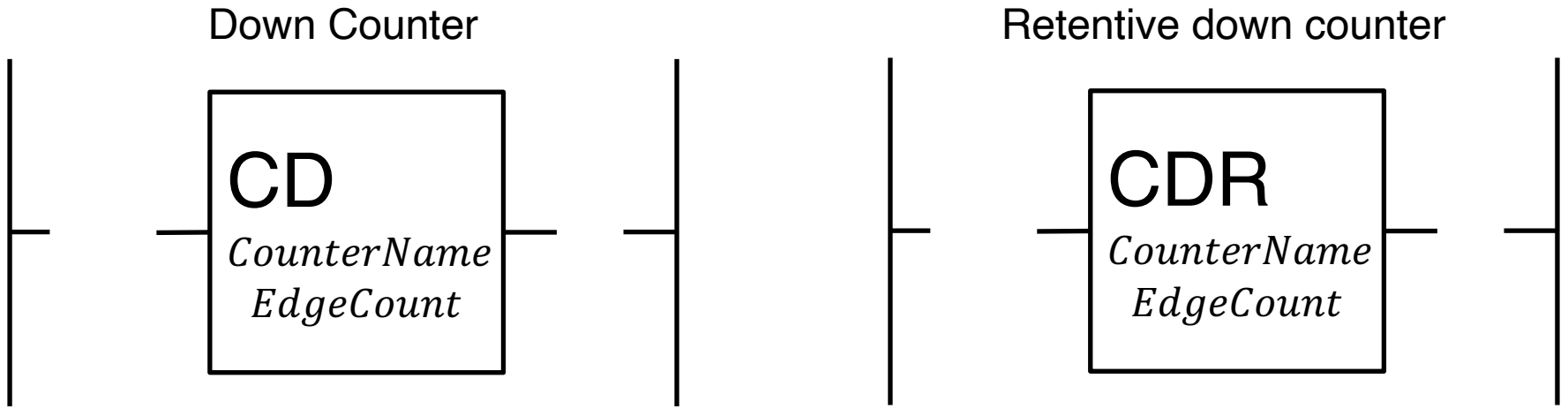


The counter is the element that counts the positive edges of its input until the counted value reach the value $EdgeCount$. In this case the output is activated.

In the case of Retentive Counter, it is the same as retentive timer (you need the reset coil to reset the counted value).

Ladder: Language elements

Down Counter

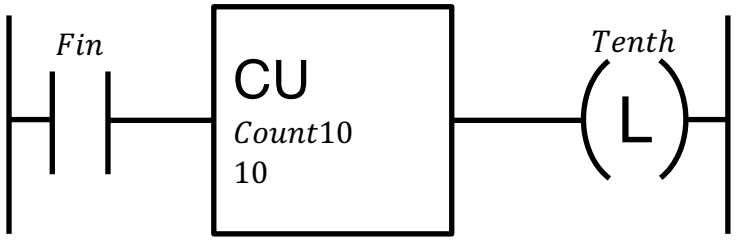


The down counter starts from $EdgeCount$ to 0. When the value 0 is reached, the output is activated.

Example

<u>Counter example</u>

Develop a software that signals the passage of the 10th client through a door, using a photocell on the door.

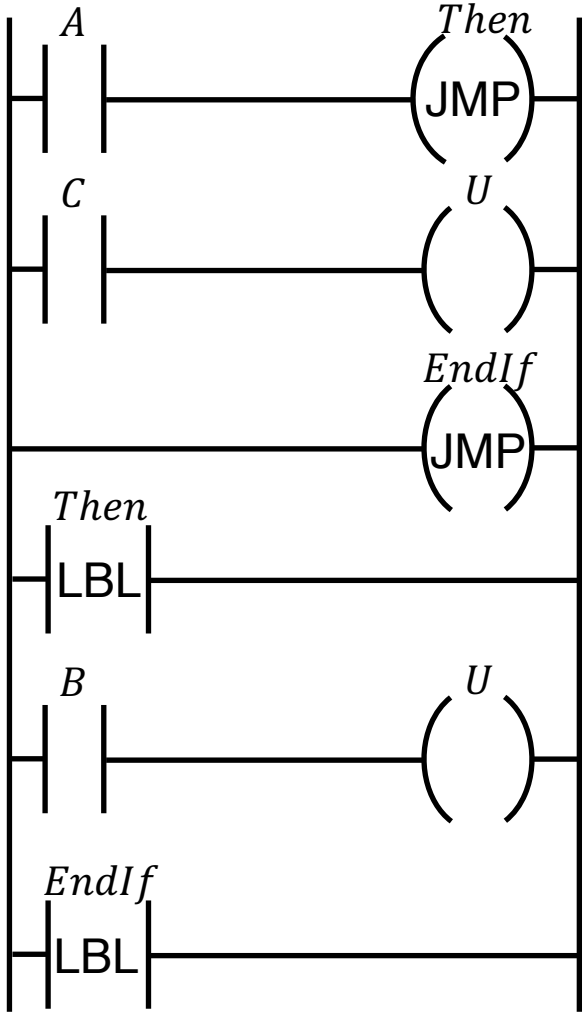


Flow control instructions

Jump to a label

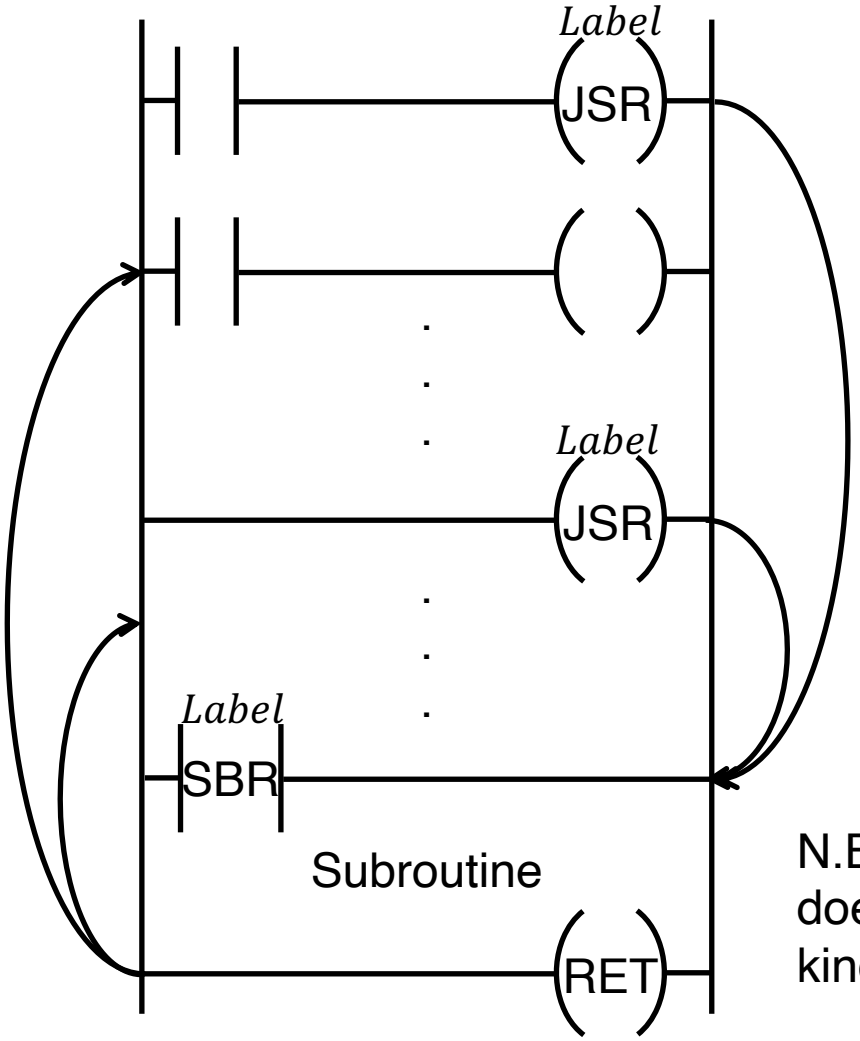
Let's see a quick example!

```
if(A) {
        U = B;
}
else {
        U = C;
}
```



Flow control instructions

Jump to a subroutine



N.B.: Automation Studio does not support these kind of jumps

Arithmetic/Logic operations

Arithmetic/Logic operations

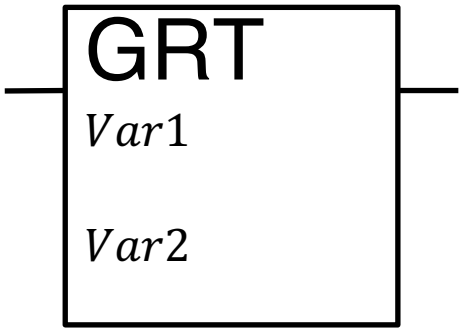
If the component has continuity on its left side, the variables $Variable1$ and $Variable2$ are added. The result is stored into the variable $Result$.

As well as ADD there are: SUB, MUL, DIV, AND, OR, etc.

N.B.: Some PLCs deals only with integer numbers, others also with real number. The type of each variable is defined inside the .var file.

Comparison operations

Comparison operations



If the component has continuity on its left side, the variables $Var1$ and $Var2$ are compared:
if $Var1 > Var2$, then the continuity on the right is activated.

As well as GRT there are: EQU, NEQ, GEQ, LEQ, LES, etc.

Example

«Non boolean» operations

Create a software that, receiving in input an analog signal composed of 10 bit (0 ÷ 1024), scales the signal with offset 512 and gain 10/512. The software has to set an alarm flag if the magnitude of the measure overpasses the value 5.

Other instructions

Other Instructions/Blocks

- MOVE → Sets the value of a non-boolean variable
- ADR → Returns the address of a variable
- SIZEOF → Returns the size of a variable, expressed in byte
- PID → Allows the use of a PID controller
- …

Exercices

Exercise 1

We want to create a system that allows the transportation of stones with a cart. The operator starts the system by pressing the button START. The cart follows the rail from left to right and stops itself, waiting the loading of the stones.

When the stones are accumulated into a tank, they are transferred into the cart.

After that, the cart has to move automatically down the rail from right to left.

Exercise 1

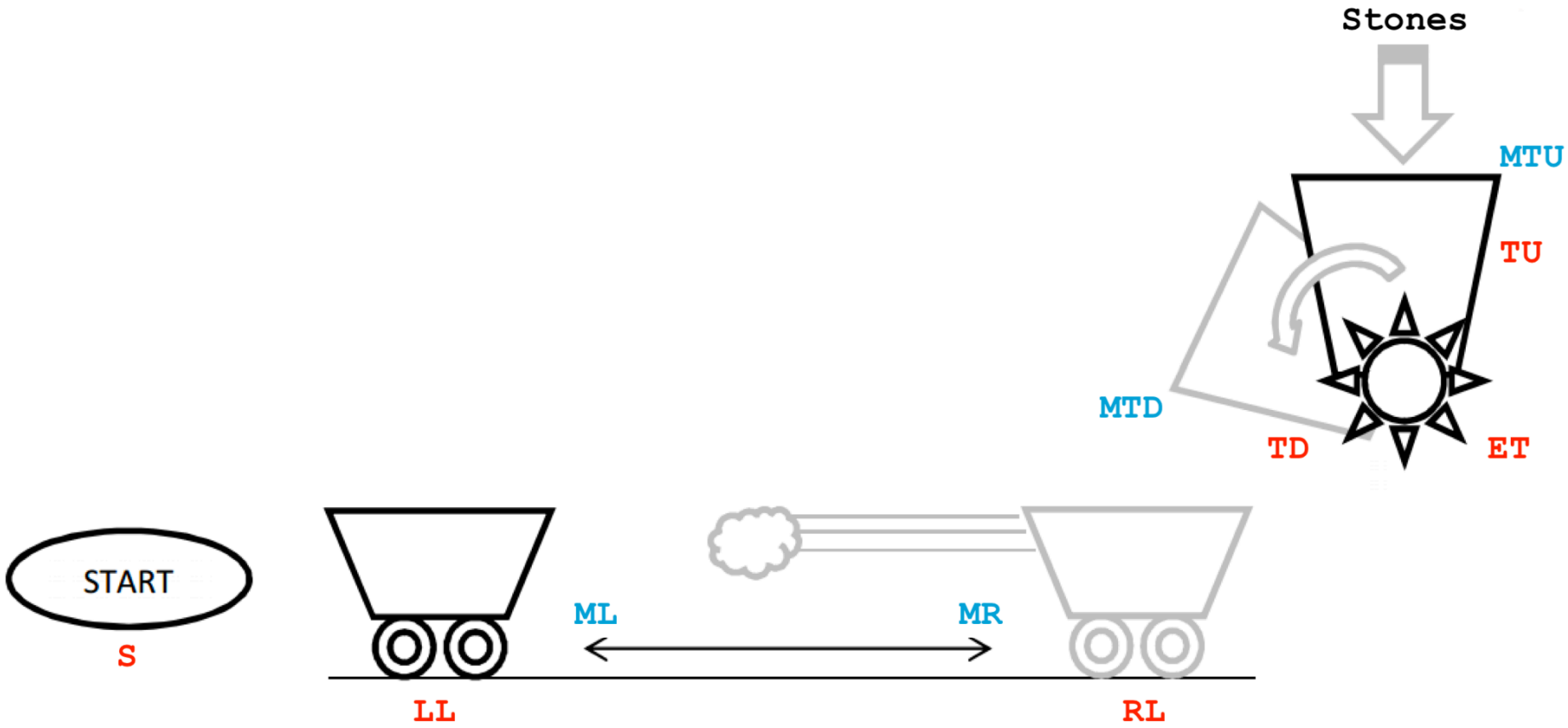
We have six sensors as INPUTS:

- S : Start
- LL : Left Limit-switch
- RL : Right Limit-switch
- ET : Empty tank
- TD : Tank down
- TU : Tank up

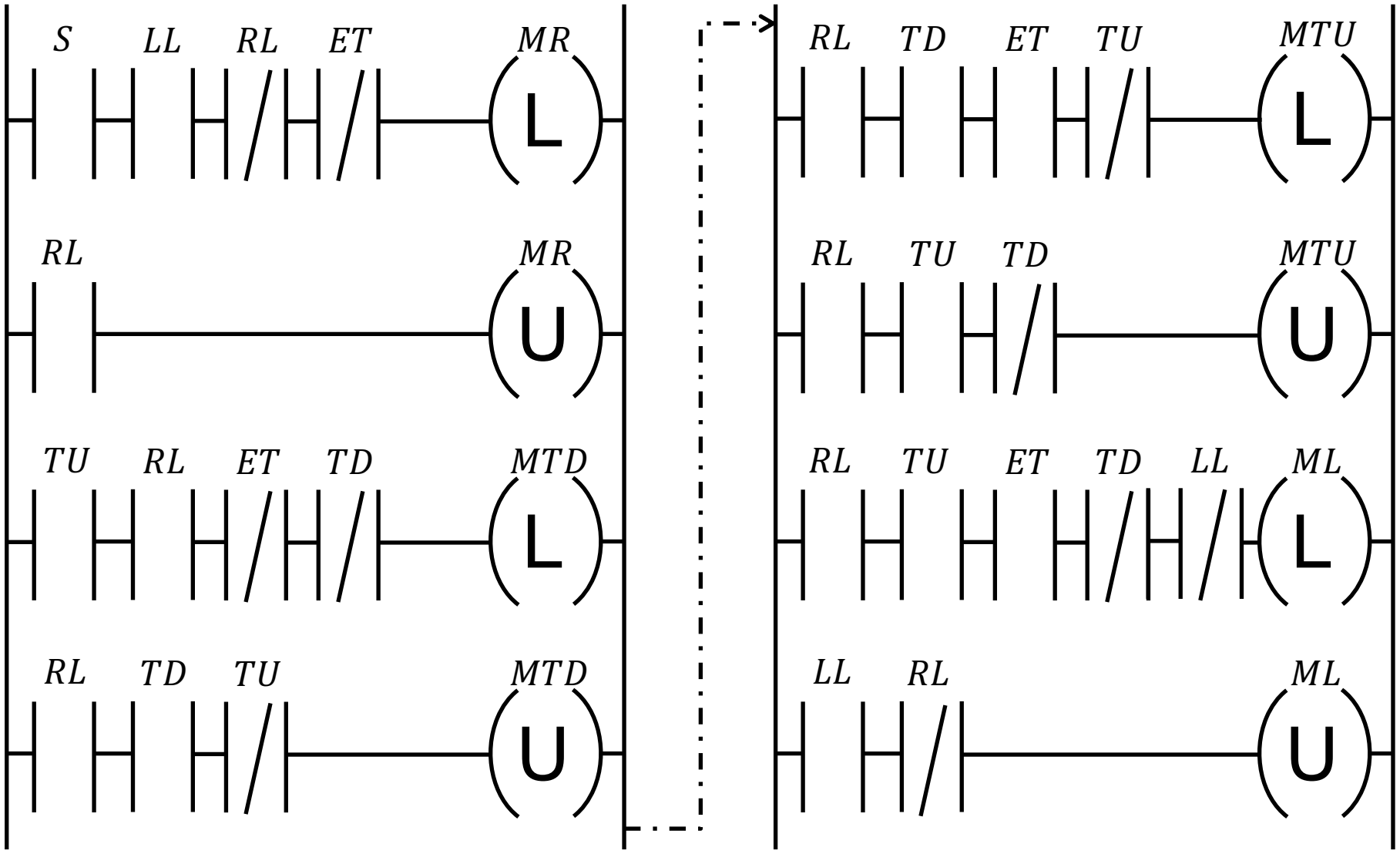
We have the following OUTPUTS:

- MR : Cart motor to the right
- ML : Cart motor to the left
- MTD : Tank's motor down
- MTU : Tank's motor up

Exercise 1



Exercise 1


S
LL
RL
ET
MR
L
RL
MR
U
TU
RL
ET
TD
MTD
L
RL
TD
TU
MTD
U
RL
TD
ET
TU
MTU
L
RL
TU
TD
MTU
U
RL
TU
ET
TD
LL
ML
L
LL
RL
ML
U

Exercise 2

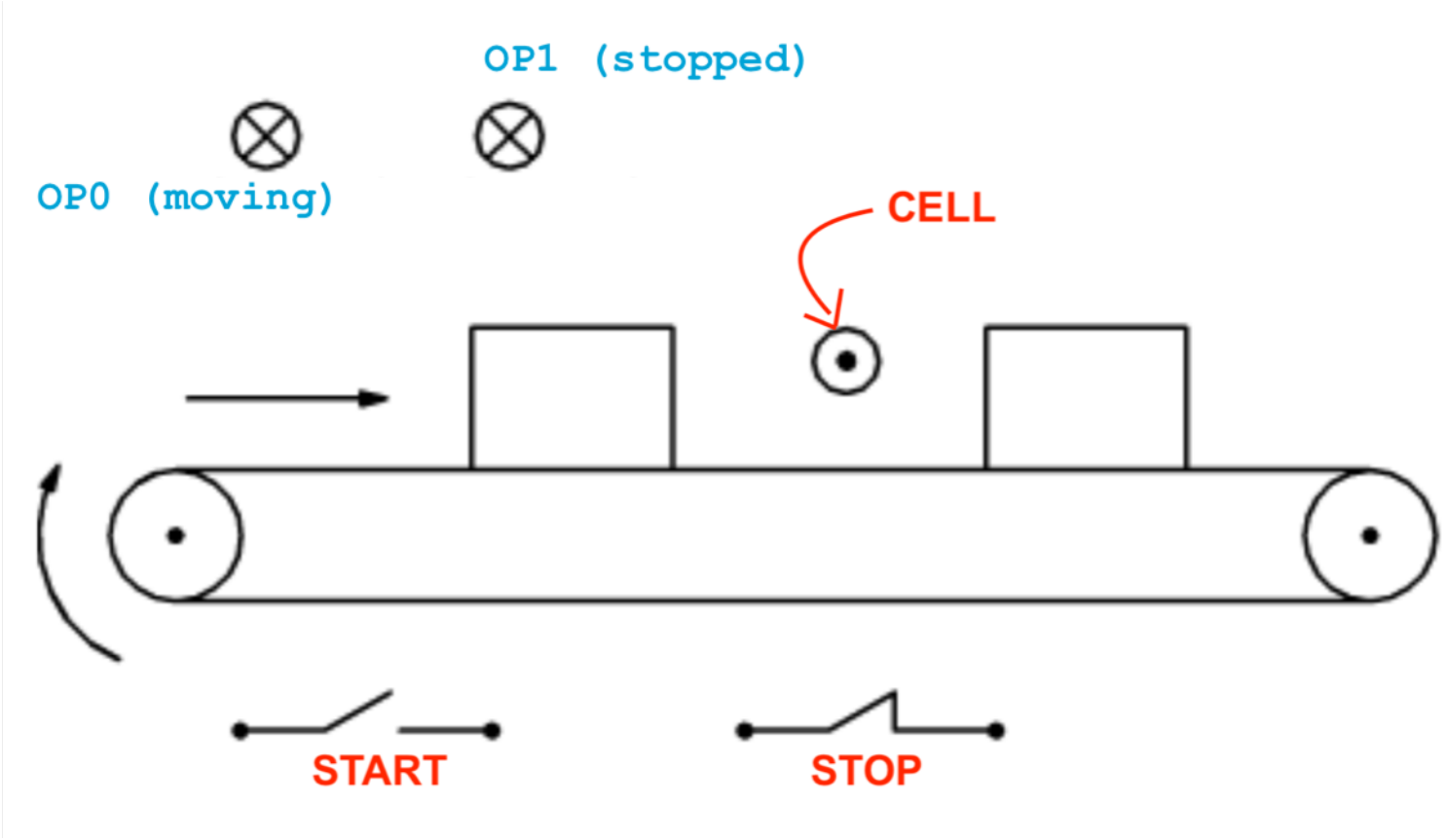
Consider a piece-counter with a conveyor belt.

The conveyor belt is moved by a motor which is controlled by two buttons: START and STOP. There are two lamps that signal the state of the conveyor belt(stopped/moving).

Each piece is placed at the beginning of the conveyor belt and, for each piece that passes in front of a photo-cell, a counter has to be incremented. The system has to automatically stop itself every 50 pieces.

The conveyor belt can be stopped at any time by pressing the STOP button. The restarting of the conveyor belt can occur by means of the pression of the START button, but, in this case, the counter have to retain its value.

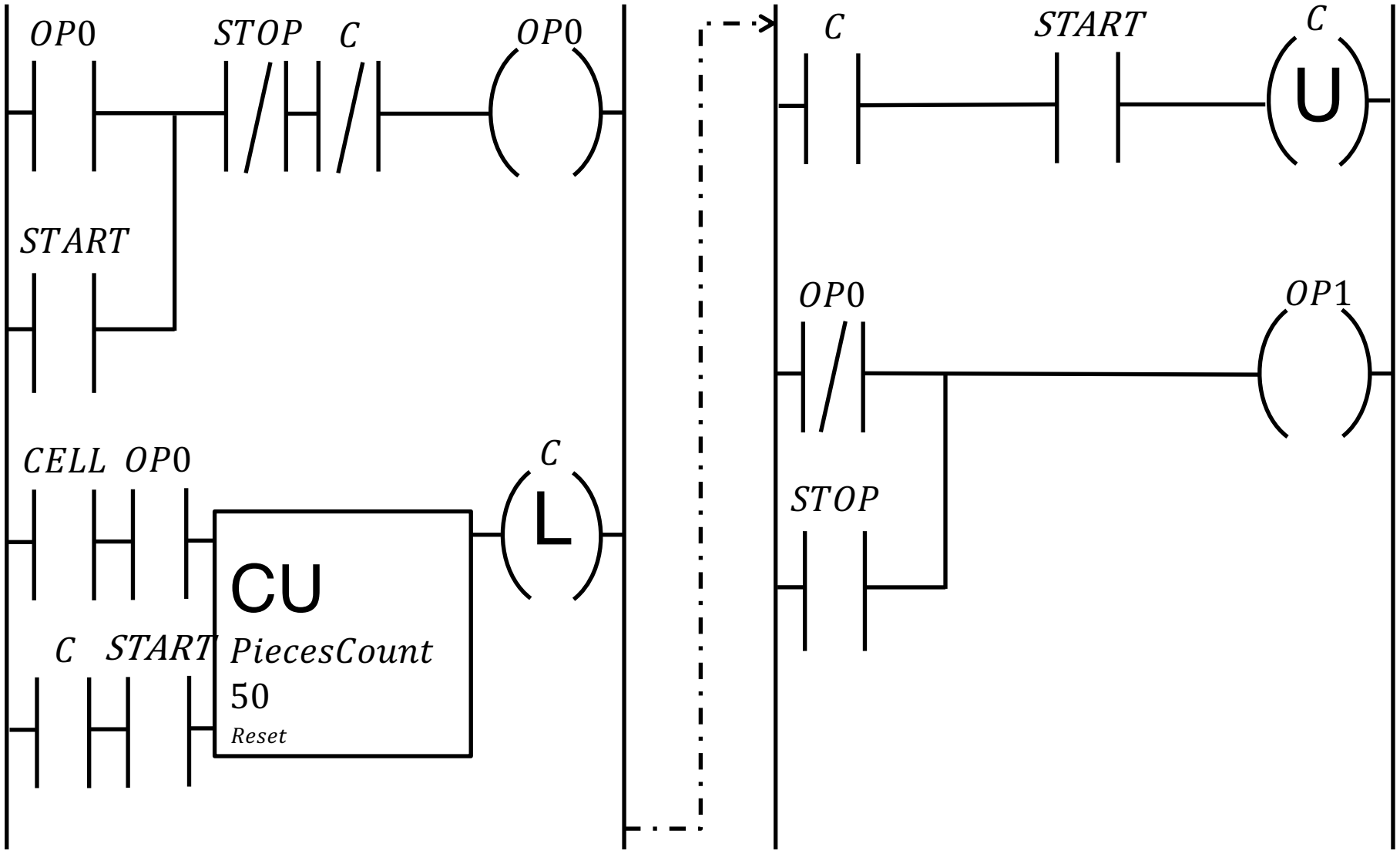
Exercise 2



Exercise 2

- At startup, the motor that drives the conveyor belt must be halted, so only *OP1* has to be on.
- By pressing the *START* button, the motor starts: OP0 has to turn on and OP1 has to turn off.
- Every time the motor is running and a new piece is detected in front of the photocell *CELL*, the counter has to be increased.
- When the counter reach the value 50, the conveyor belt has to be stopped: OP1 has to turn on and OP0 has to turn off. *(in a following restart, the counter must start from the value 0).*
- When the conveyor belt is stopped before the value 50 the counter must retain its value.

Exercise 2


OP0
STOP
C
OP0
START
CELL
OP0
C
L
CU
C
START
PiecesCount
50
Reset
C
START
C
U
OP0
OP1
STOP

Exercise 3

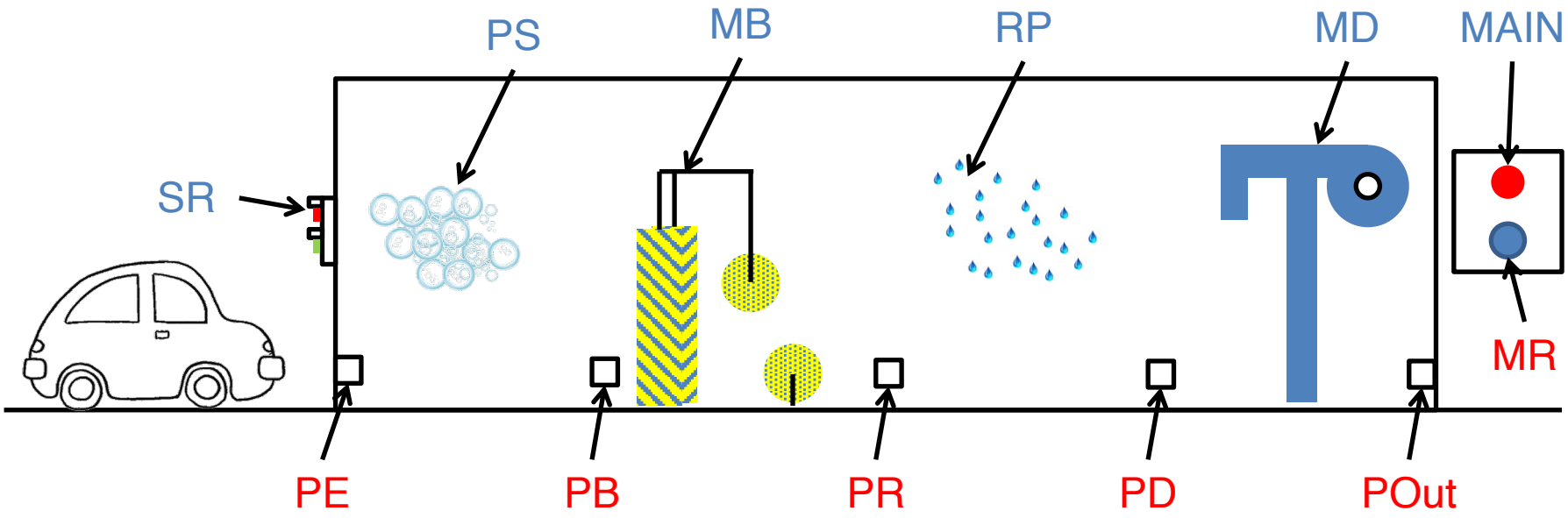
Consider an automatic carwash system.

The customer approaches to the conveyor belt when the semaphore is green. The phases of the washing are: soaping, brushing, rinsing e drying.

All the phases are preceded by a photocell that signals the arrival of the car in that new section of the carwash.

Every 1000 washing, the carwash system have to block itself waiting for the maintenance, that is executed by an operator.

Exercise 3



Input		Output	
PE	Entry photocell	SR	Stop semaphore (0=GREEN, 1=RED)
PB	Brushing photocell	PS	Soaping pump
PR	Rinsing photocell	MB	Brushing motor
PD	Drying photocell	RP	Rinsing pump
POut	Out photocell	MD	Drying motor
MR	Maintenance reset	MAIN	Halt for maintenance

Exercise 3

The system can be developed as a set of sub-systems:

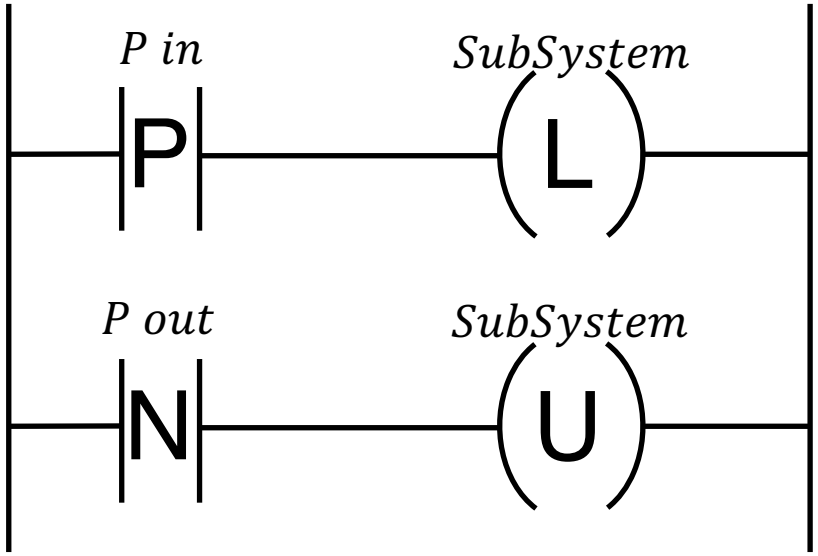
- Soaping
- Brushing
- Rinsing
- Drying

Each of this sub-systems has to start when the previous photocell activates its output and has to stop when the next photocell deactivates its output.

We must to use edge recognition contacts!

Exercise 3

For <u>each sub-system</u> we will have the following schema:

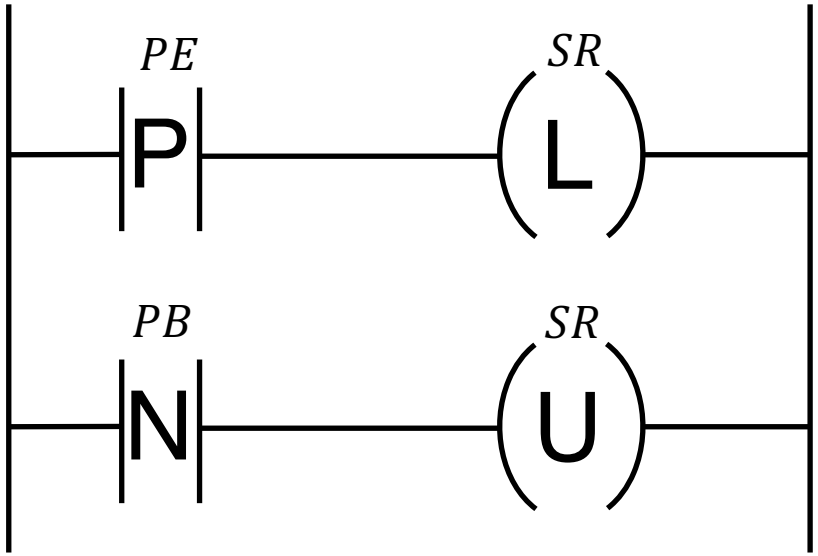


Obviously, we have to manage also:

- Semaphore
- Maintenance

Exercise 3

The semaphore will be GREEN until the photocell PE does not signal a car at the entry of the carwash. At that time, the semaphore has to turn RED (SR=1). The semaphore will stay RED until PB does not signal the complete entry of the car inside the brushing subsystem. The schema is the following:



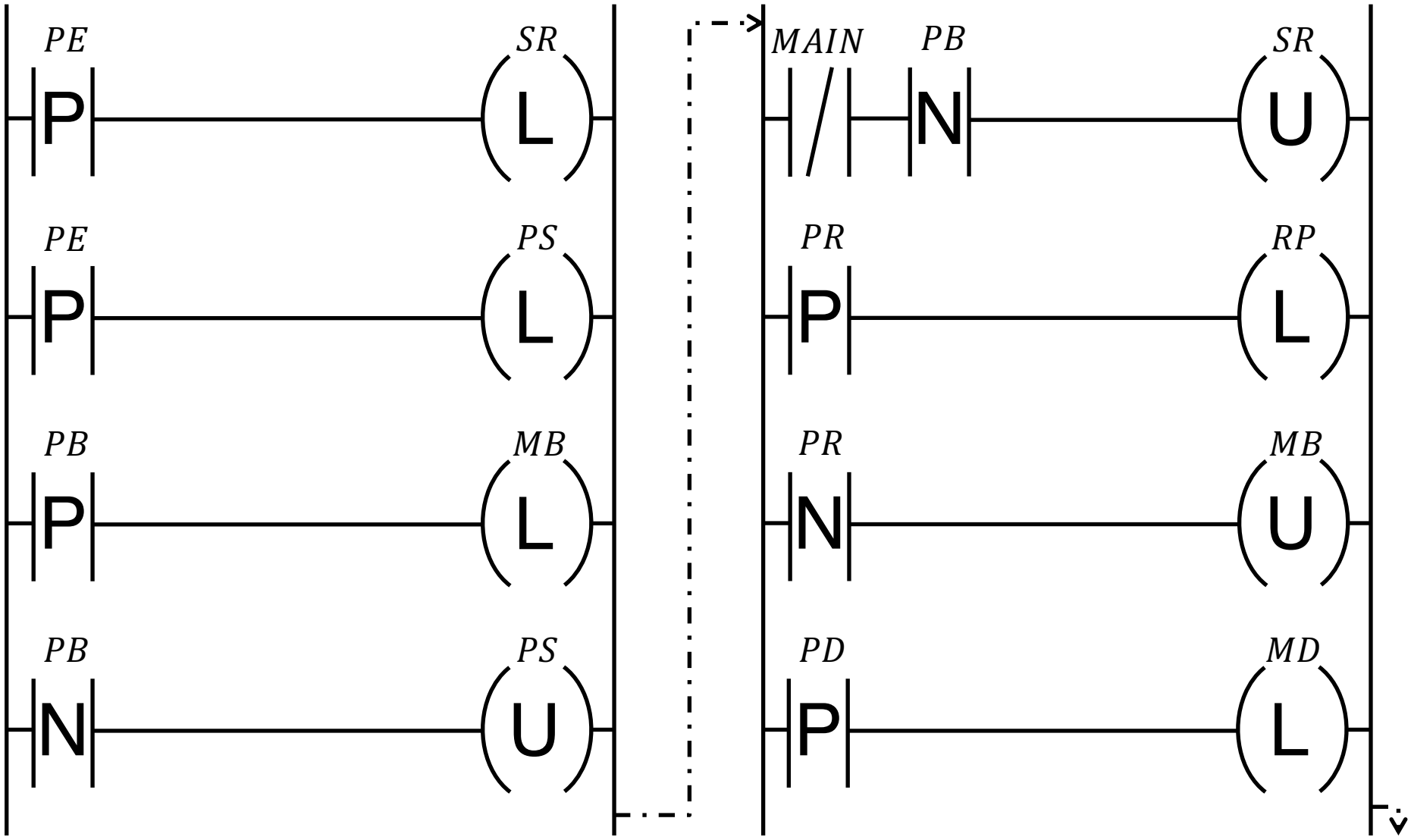
Exercise 3

For the maintenance management we have to count the cars that use the carwash (counter that count the positive edges of PE).

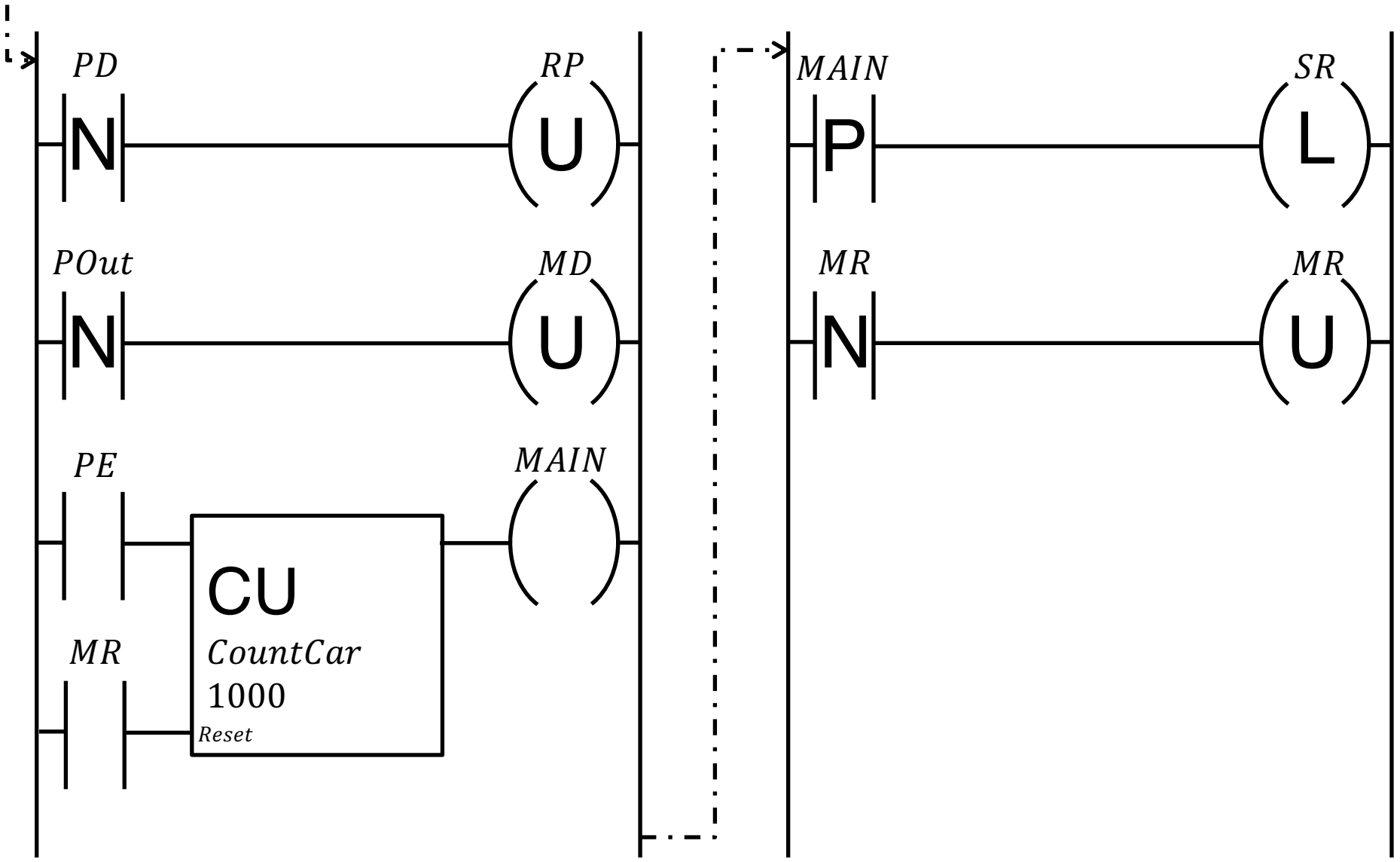
When the value 1000 is reached by the counter, the semaphore has to stay RED until the maintenance reset button (MR) is pressed.

N.B.: It is important, when the semaphore is set to RED, assure that the code line that sets it to GREEN is disabled!

Exercise 3


PE
P
SR
L
PE
P
PS
L
PB
P
MB
L
PB
N
PS
U
MAIN
PB
/
N
SR
U
PR
P
RP
L
PR
N
MB
U
PD
P
MD
L

Exercise 3


PD
N
RP
U
POut
N
MD
U
PE
MR
CU
CountCar
1000
Reset
MAIN
MAIN
P
SR
L
MR
N
MR
U

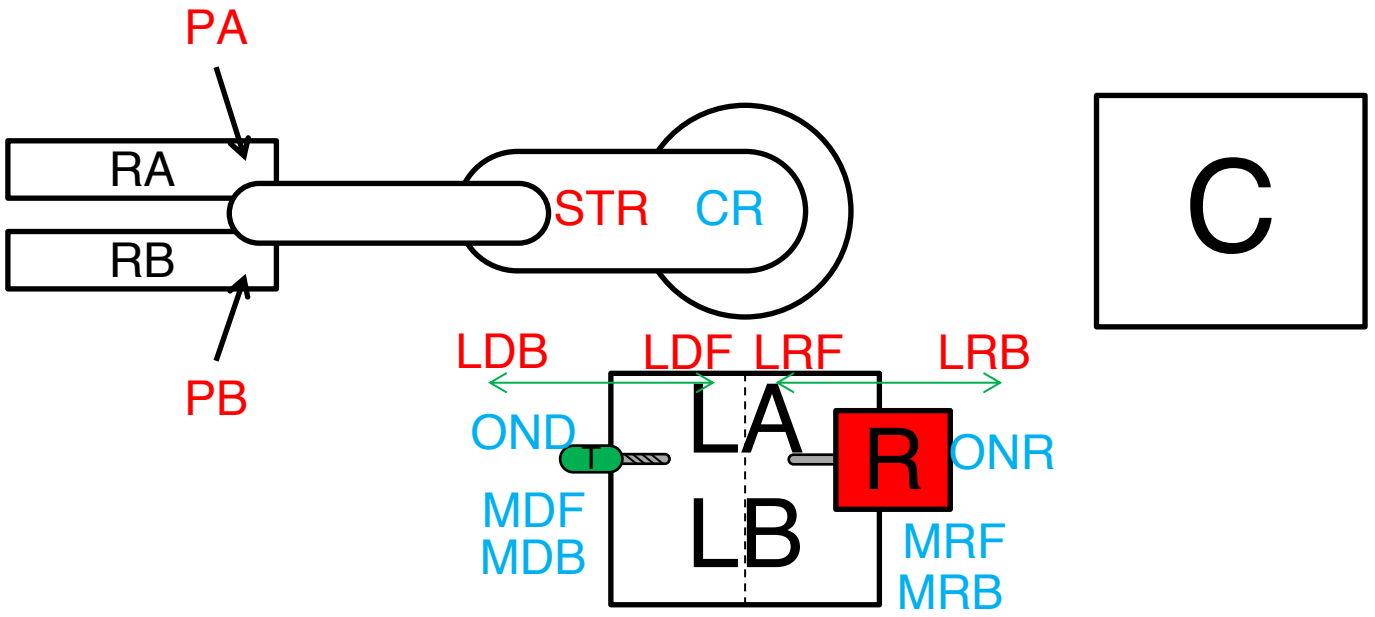
Exercise 4

Consider a system used for automatic drilling and riveting metal sheets.

When the two sheets are available, a robot takes them (one by one) and places them over an assembly mask. After the end of the placing, the pieces are perforated by an automatic drill (5 seconds) and rivetted by a riveter (10 seconds).

At the end of the process, the robot move the produced product in a container.

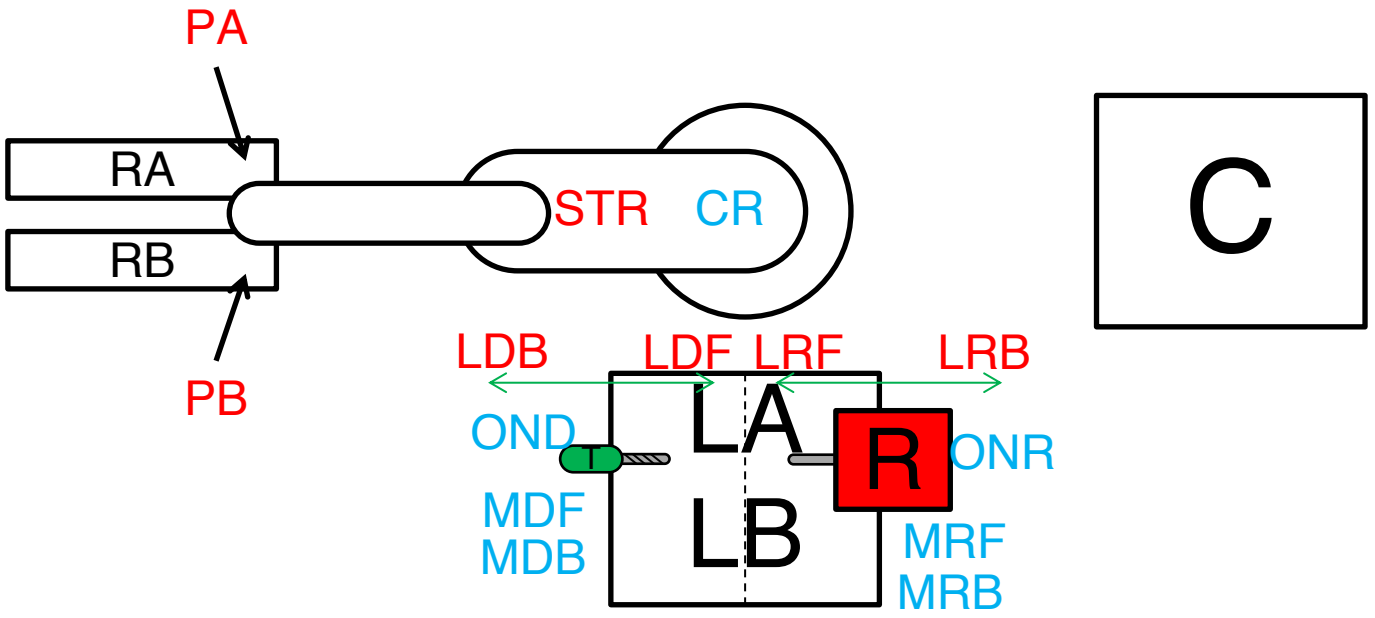
Exercise 4



Input

PA	Piece A available	LDF	Drill forward limit
PB	Piece B available	LRB	Riveter backward limit
STR	Status robot (0=END, 1=EXECUTING)	LRF	Riveter forward limit
LDB	Drill backward limit		

Exercise 4



Output

CR	Robot command (0=STOP, 1=take piece A, 2=take piece B, 3=deposit final product in C)	OND	Drill ON
		MRF	Riveter motor forward
MDF	Drill motor forward	MRB	Riveter motor backward
MDB	Drill motor backward	ONR	Riveter ON

Exercise 4

Let's think about all the conditions that enable some action in our system:

- PA=1 and PB=1 --> we have to send the commands 1 and 2 to the robot
- At the end of the moving, we can enable MDF and, when the forward limit is reached, the drilling can
- …

What if, in the meantime, there are two new pieces in PA and PB?

We must insert a lot of conditions to avoid the sending of the command 1 to the robot…

Exercise 4

The steps to be executed to create a finite product are:

1)Wait until PA=1 and PB=1
2)Send command 1 to the robot and wait the end of its execution
3)Send command 2 to the robot and wait the end of its execution
4)Move the drill forward until the forward-limit is reached
5)Operate the drill for 5 seconds
6)Move the drill backward until the backward-limit is reached
7)Move the riveter forward until the forward-limit is reached
8)Operate the riveter for 10 seconds
9)Move the riveter backward until the backward-limit is reached
10)Send command 3 to the robot and wait the end of its execution
11)Send command 0 to the robot

Exercise 4

Let’s see the code inside the B&R development environment (Automation Studio)

We will use:

- Contacts
- Coils
- Logical ports
- Move
- Timer

Conclusions

Final consideration

Ladder is a simple language but it's not suitable when the system can be represented with a «state machine».

SFC (another language in the IEC 61131 norm, that we won't see) or Structured Text are better choices to develop automatons and state machines.