Supervisory Control & Monitoring

Topic - Programmable Logic Controllers (PLC)

Teacher - Prof. Elio USAI
eusai@diee.unica.it
Dipartimento di Ingegneria Elettrica ed Elettronica
Università di Cagliari

References

J. Hugh
Automating Manufacturing Systems with PLCs
http://www.theautomationstore.com/product_images/uploaded_images/plcbook5_0.pdf, 2007

P. Chiacchio
PLC ed automazione industriale
McGraw-Hill Libri Italia, 1996

P. Chiacchio, F. Basile
Tecnologie informatiche per l'automazione
McGraw-Hill, 2004

Sommario

- PLC – Definitions
- PLC – Characteristics
- PLC – Structure
- PLC – Modules
- PLC – Languages
- Sequential Functional Chart
- Programming example
- Example: pumping plant
- Ladder Diagram
- Translation from SFC to LD

PLC - Definitions

IEC 1131 Norm

A digitally operated electronic apparatus which has been ruggedized and adapted for industrial applications, which uses a programmable memory for the internal storage of instructions for implementing specific functions such as logic, sequencing, timing, counting, and arithmetic to control, through digital or analog input/output modules, various types of machines or processes.

PLC - Definitions

IEC 1131 Norm

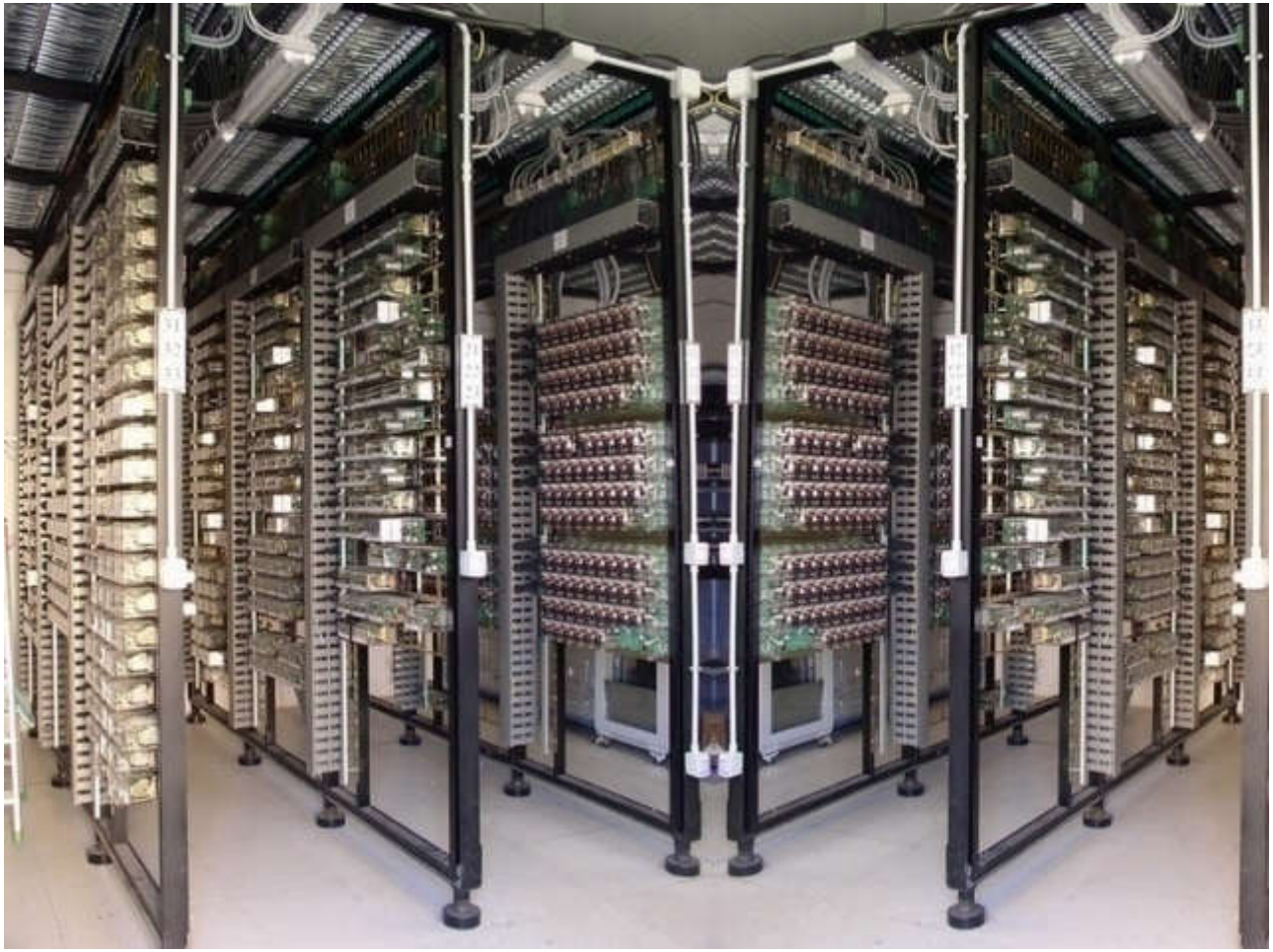
A digitally operated electronic apparatus which has been ruggedized and adapted for industrial applications, which uses a programmable memory for the internal storage of instructions for implementing specific functions such as logic, sequencing, timing, counting, and arithmetic to control, through digital or analog input/output modules, various types of machines or processes.

PLC - Definitions

PLC characteristics

- Electronic apparatus
- Used for industrial applications
- Internal programmable memory
- Implementing logic, sequencing, timing, and counting functions
- Equiped with digital or analog input/output modules

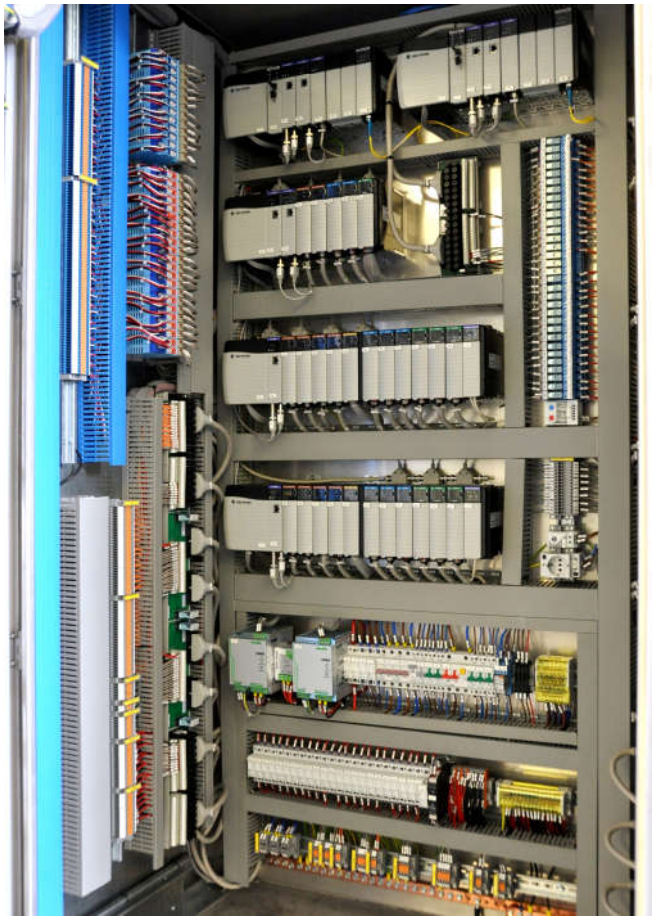
PLC - Definitions



Substitutes the “old” relay cabinet/room

Programmable Logic Controllers (PLC)

PLC - Definitions



Substitutes the “old” relay cabinet/room

Programmable Logic Controllers (PLC)

PLC - Characteristics

- Reconfigurability / Programmability

The logic in the PLC can be changed taking to follow the changes in the plant or in the process

Reduced need of substituting the whole control system

Flexible production

PLC - Characteristics

- Compact

Reduced size such that room is saved

Reduced energy consumption due to the low consumption electronic devices

PLC - Characteristics

- Modularity

New features don't need to change the all control system

Additional control loops can be added without significant difficulties

Simple adaptation to the specific plant needs

PLC - Characteristics

- Low-cost

Cost decreasing of the electronic devices

Save implementation time

Save maintenance costs both for the hardware and the software

PLC - Structure

Minimal configuration

- Rack 19”
- Power supply
- Microprocessor
- Input/Output ports
- Programming device

Compact construction
⇒ Unique Module

PLC - Structure

Espansion modules

- Communication
- Storage memory
- I/O ports
- PID
- Servo-drive
- Encoder
- HMI

PLC - Modules

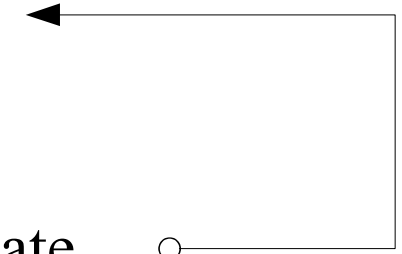
Microprocessor

Scan cycle

Optimized data transfer

Sequential operations

- Inputs scan
- Execution
- Checks
- Outputs update



PLC - Modules

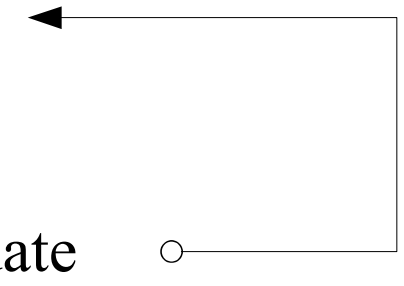
Microprocessor

Scan cycle

Optimized data transfer

Sequential operations

- Inputs scan
- Execution
- Checks
- Outputs update



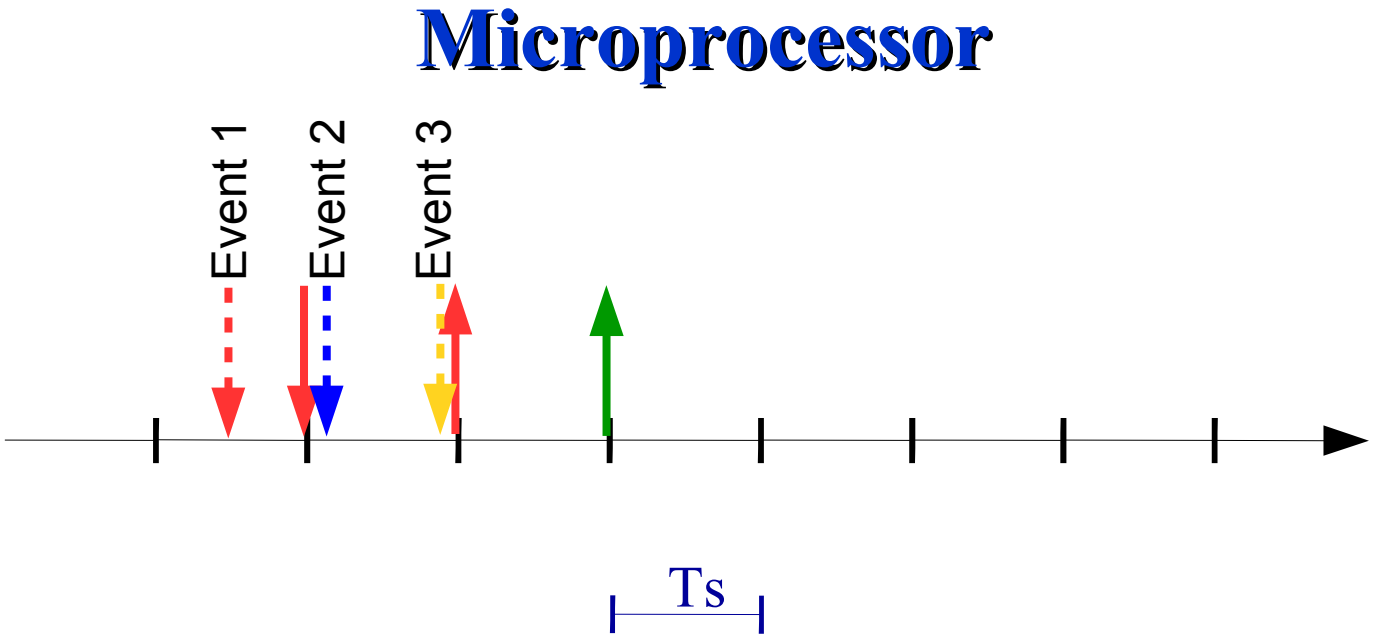
Checks can be executed also after the outputs update

PLC - Modules

Microprocessor

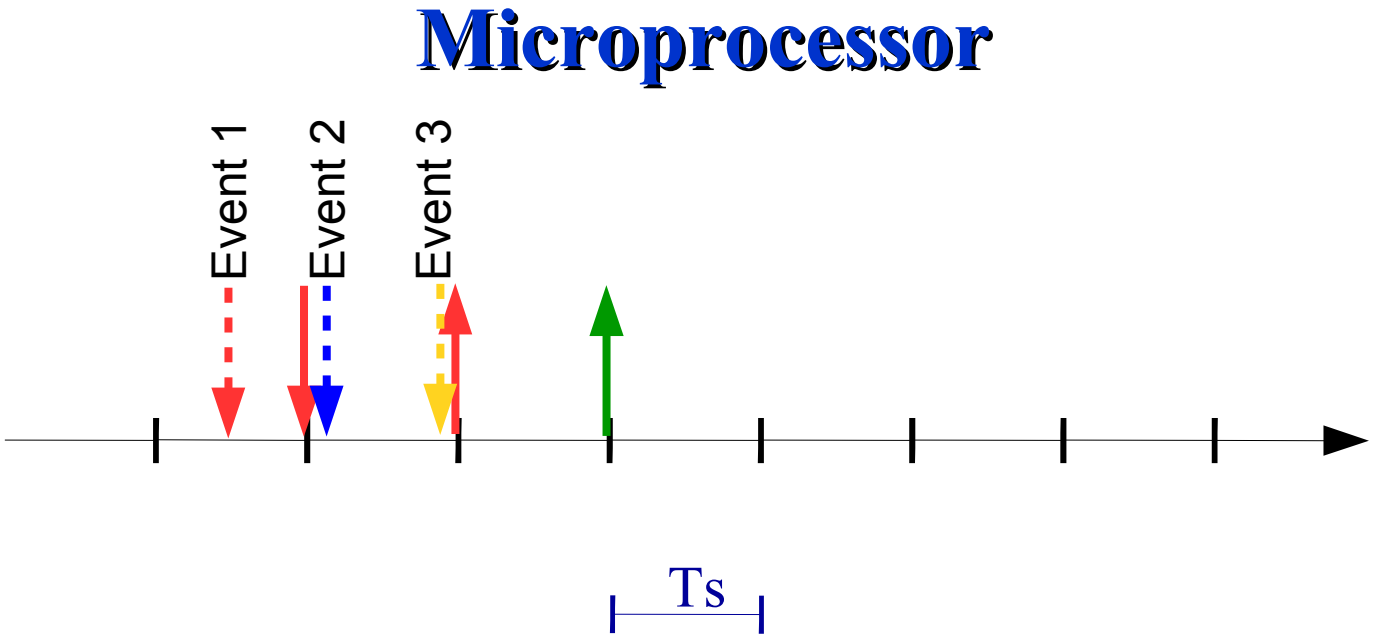
Scan time period	The time needed to complete the full scan cycle
Reaction time	The time interval between the occuring of the event and the execution of the corresponding action

PLC - Modules



What are the Reaction time for each event?

PLC - Modules



Input scan and store

Output commands

$Ts < Tr < 2\ Ts$

Not including communication delays

PLC - Modules

Microprocessor

Scan time period

The time needed to complete the full scan cycle

Factor influencing the scan time period

- Microprocessor speed (FLOPS)
- Code complexity
- Number of inputs
- Number of outputs

PLC - Modules

Microprocessor

Operating System	Management and check of the device, “watch-dog” procedures includes, operations: programming, validation (outputs not active), program execution
Memories	ROM: the OS is stored here RAM: stores the current data EEPROM: stores the execution program SD cards: multiple use

PLC – Modules

I / O ports

Are the interface between the process and PLC while guaranteing the isolation with the field deviced

Galvanic decoupling	Relays Optoisolators High isolation trasformers
Functional decoupling	Voltage follower

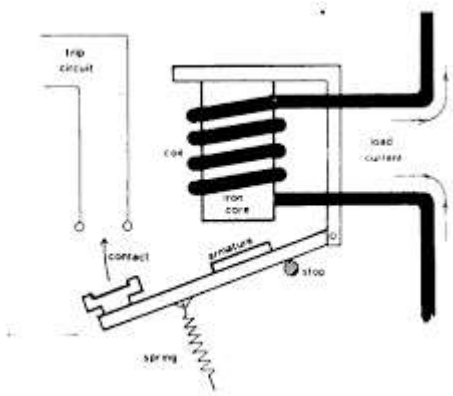
PLC - Modules

I / O ports

Are the interface between the process and PLC while guaranteing the isolation with the field deviced

Galvanic decoupling

Relays



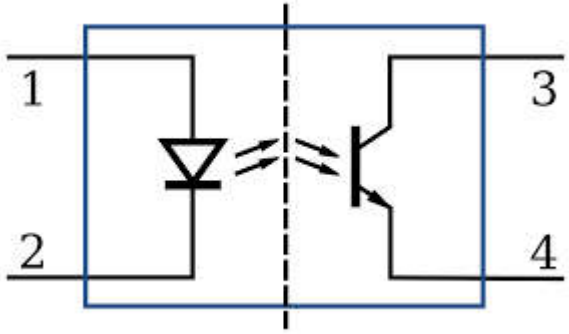
PLC - Modules

I / O ports

Are the interface between the process and PLC while guaranteing the isolation with the field deviced

Galvanic decoupling

Relays
Optoisolators



PLC - Modules

I / O ports

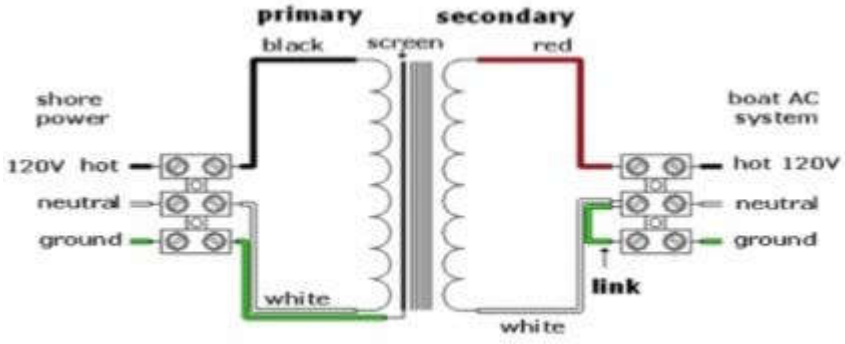
Are the interface between the process and PLC while guaranteing the isolation with the field deviced

Galvanic decoupling

Relays

Optoisolators

High isolation trasformers

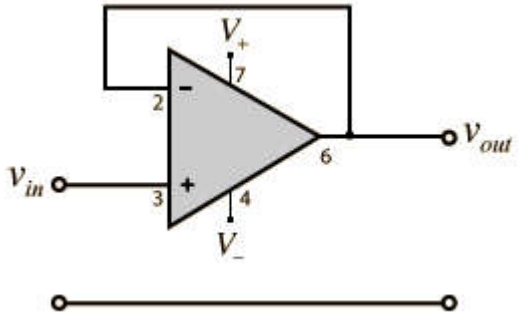


PLC - Modules

I / O ports

Are the interface between the process and PLC while guaranteing the isolation with the field deviced

Functional decoupling



Voltage follower

PLC - Modules

I / O

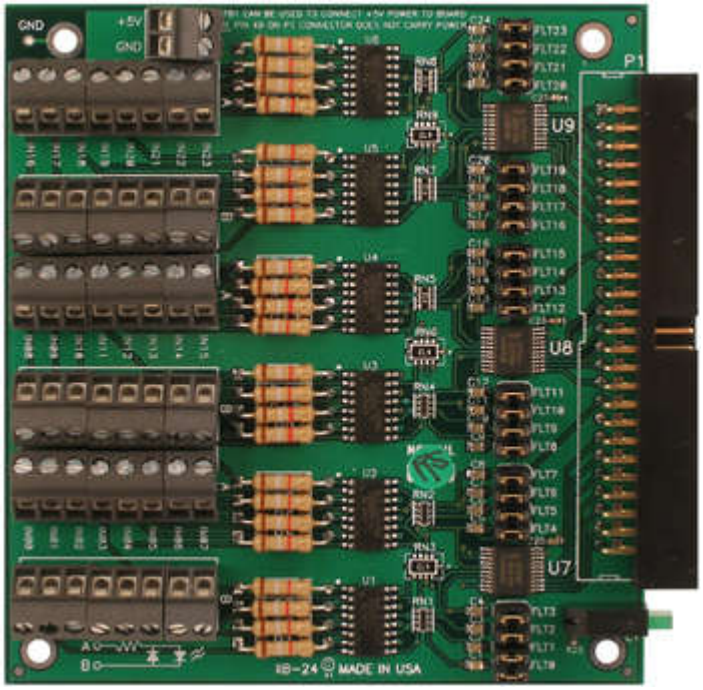
Digital	0÷24 V d.c. 0÷230 V a.c. 50 Hz
Relays	0÷24 V d.c. 0÷230 V a.c. 50 Hz
Analogue	± 5 V, ± 10 V, 0÷10 V, 4÷20 mA For thermocouples and thermistors A/D Convertitor (min. 1) D/A Convertitor (more than 1)

PLC - Modules

I / O

Digital

0÷24 V d.c.
0÷230 V a.c. 50 Hz



PLC - Modules

I / O

Digital	0÷24 V d.c. 0÷230 V a.c. 50 Hz
Relays	0÷24 V d.c. 0÷230 V a.c. 50 Hz

PLC - Modules

I / O

Digital	0÷24 V d.c. 0÷230 V a.c. 50 Hz
Relays	0÷24 V d.c. 0÷230 V a.c. 50 Hz
Analogue	± 5 V, ± 10 V, 0÷10 V, 4÷20 mA For thermocouples and thermistors A/D Convertitor (min. 1) D/A Convertitor (more than 1)

PLC - Modules

Programming devices

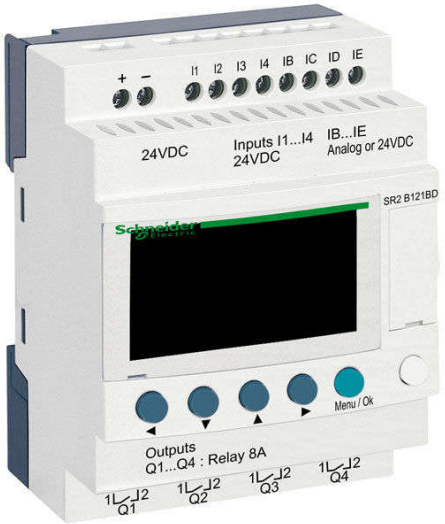
Console	It allows for writing simple programs and can be connected via serial port or wireless.
Keyboard & display	It can be considered as a dedicated lap-top with advanced features.

PLC - Modules

Programming devices

Console

It allows for writing simple programs and can be connected via serial port or wireless.



PLC - Modules

Programming devices

Keyboard & display

It can be considered as a dedicated laptop with advanced features.

PLC - Modules

Power supply unit

- Charecterised by maximal and rated power
- To choose taking into account future expansion
- Turn-off allarm
- Redunded for critical applications

PLC - Modules

Communication

Fieldbus	RS 232C, RS 485 Ethernet Optical fiber
Remote I/O	Data acquisition in large plants
Protocols	Profibus, Fieldbus Fundation Device Net - CAN ModBus CSMA/CD (TCP-IP)

PLC - Modules

Coprocessor

Used for special and complex programs

Programming with specific languages (C, C++, etc

Direct connection to the network

Dedicated memories

Few applications, nowadays

PLC - Modules

Command units

PID

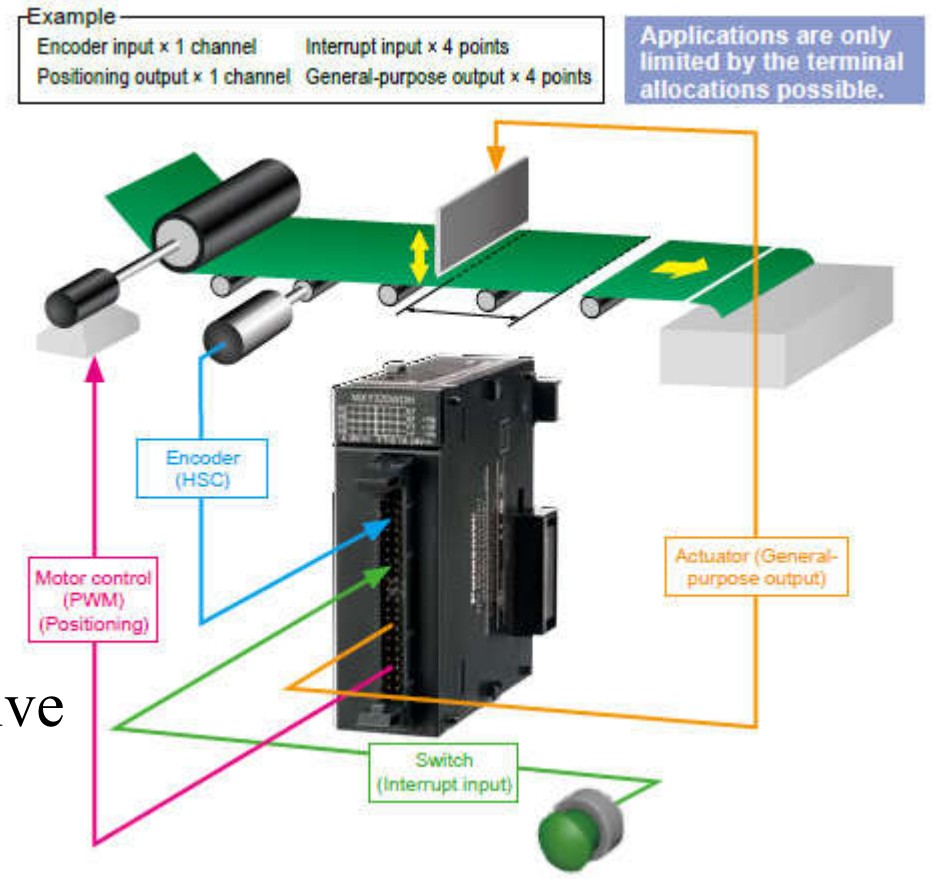
They allow for a simple implementation of PID control logics.
Often not needed – specific codes available

Servo drive

Specialised unit for the servo-drive control and command

PLC - Modules

Command units

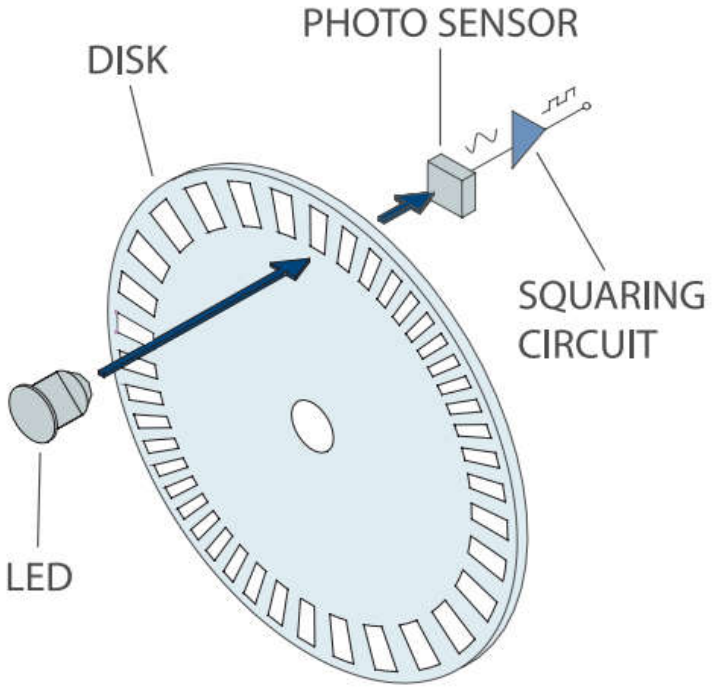


Servo drive

Specialised unit for the servo-drive control and command

PLC - Modules

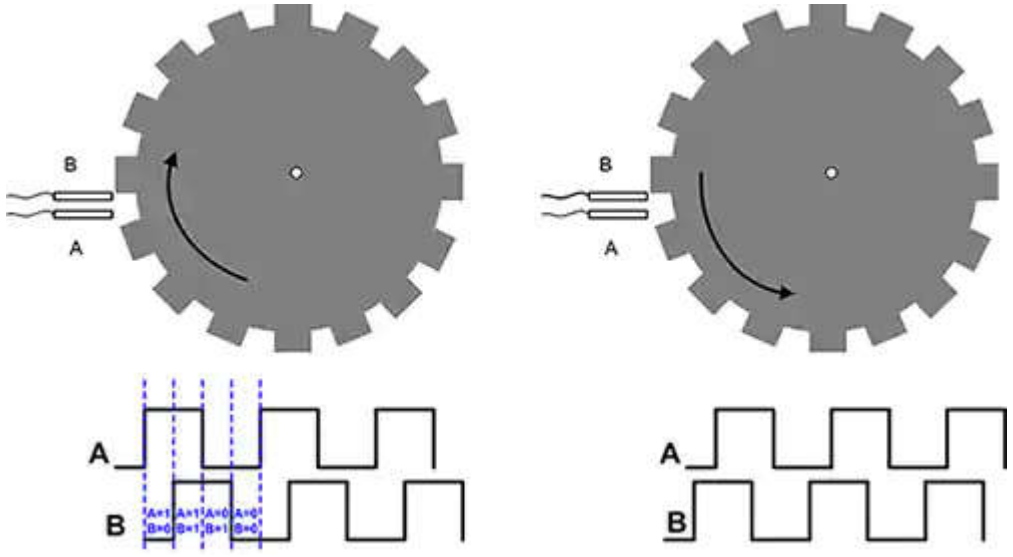
Encoders are
used as sensors
for a servo drive



The shaft position is detected by the
photo sensor (binary signal)

PLC - Modules

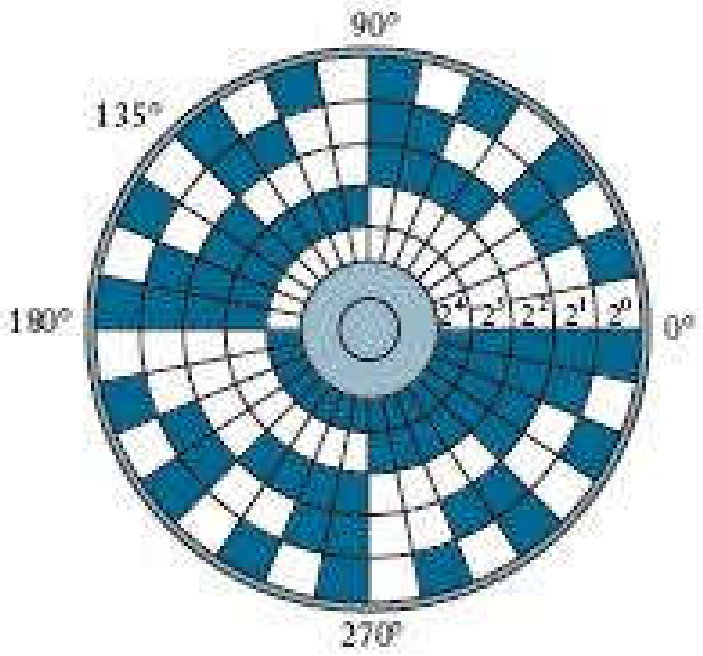
Encoders are used as sensors for a servo drive



The rotation direction can be detected by using two photo sensors.

PLC - Modules

Encoders are
used as sensors
for a servo drive



Using a set of photosensors the absolute position can be measured.

Resolution depends on the number of annula. Grey code is used for single bit change.

PLC - Modules

Human Machine Interface

Display and keyboard connected to the PLC, mostly substituted by Laptop PC

PLC - Modules

Back up and reliability

Allow for the commutation between PLCs in the case of faults. Usually between two equal devices (redundancy)

Programmable Logic Controllers (PLC)

PLC - Modules

Back up and reliability

Allow for the commutation between PLCs in the case of faults. Usually between two equal devices (redundancy)

Hot redundancy	The spare part/PLC executes the same program but updates the outputs just when a fault occurs
Cold redundancy	The spare part/PLC is off but ready to be turned on whenever the main PLC is faulty
Intrinsic safety	The commands of at least two PLC are compared and applied only when equal

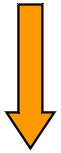
PLC - Languages

Portability

The control program can be easily implemented in several different PLCs

Maintainability

The control program can be easily modified even by third parts to implement new functions



Standard languages

PLC - Languages

IEC 61131-3 standard languages

Text	Instruction list – *similar to assembler*
	Structured text – *any language*
Symbolic	**Ladder** – *similar to electrical schemes*
Object oriented	**S**equential **F**unctional **C**hart – *automata, gerarchical implementation*
	Functional **B**lock **D**iagram – *Boolean algebra*

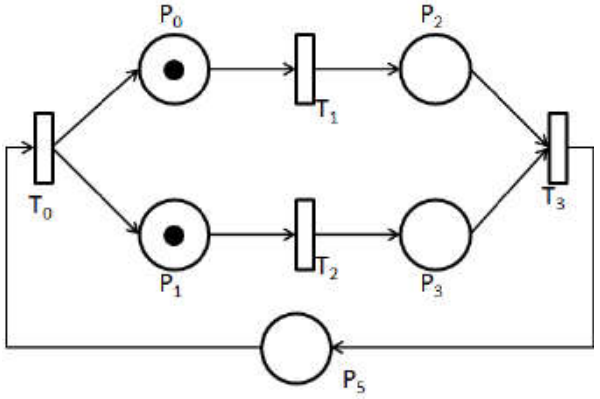
PLC - Languages

IEC 61131-3 standard languages

Object oriented

Sequential **F**unctional **C**hart – *automata, gerarchical implementation*

- Can be simply associated to a Petri-Net



PLC - Languages

IEC 61131-3 standard languages

Object oriented

Sequential **F**unctional **C**hart – *automata, gerarchical implementation*

- Can be simply associated to a Petri-Net
- Simple representation of state-event dynamics
- Scalable representation
- Self-explain process representation

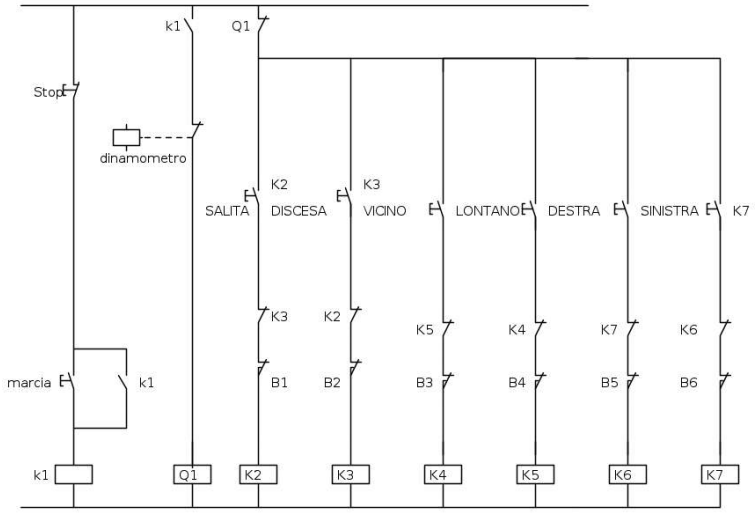
PLC - Languages

IEC 61131-3 standard languages

Symbolic

Ladder – *similar to electrical schemes*

- Similar to electrical functional schemes



Programmable Logic Controllers (PLC)

PLC - Languages

IEC 61131-3 standard languages

Symbolic

Ladder – *similar to electrical schemes*

- Similar to electrical functional schemes
- Representation quite familiar to electricians
- Less effort for transition from relay cabinets
- Difficult to derive the process evolution for complex plants

Sequential Functional Chart

It is the standardised evolution of the GRAPHe de Coordination Etapes-Transitions (*GRAPHCET*) developed in France around '70th -'80th

Standard since 1988 as IEC 848

It is included in IEC 61131-3 standard as S.F.C.

It is a high-level language for the implementation of functional block and programs, but not simple functions

It allows for the implementation of logical/functional sequences

It is a simplified (state – transition) Petri Net

Sequential Functional Chart

Phase It is an **invariant working condition** of the system which can modified only by the occurence of an event that causes a transtion to another phase. A phase is represented by a square.

Transition It establishes the change from an invariant working condition to another one, taking into account the current state and the occurence of an event in the system. A transition is represented by a bar.

Oriented arch It connect a phase with a transition and vice-versa. It is an oriented arch but the arrow can be omitted if the direction is up-down.

Sequential Functional Chart

Phase It is an invariant working condition of the system which can modified only by the occurence of an event that causes a transtion to another phase. A phase is represented by a square.

Transition It establishes the change from an invariant working condition to another one, taking into account the current state and the **occurence of an event** in the system. A transition is represented by a bar.

Oriented arch It connect a phase with a transition and vice-versa. It is an oriented arch but the arrow can be omitted if the direction is up-down.

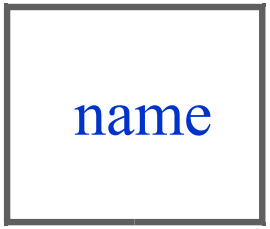
Sequential Functional Chart

Phase It is an invariant working condition of the system which can modified only by the occurence of an event that causes a transtion to another phase. A phase is represented by a square.

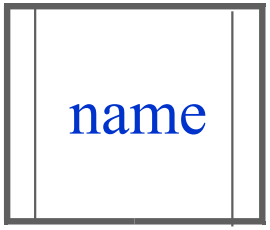
Transition It establishes the change from an invariant working condition to another one, taking into account the current state and the occurence of an event in the system. A transition is represented by a bar.

Oriented arch It **connect** a phase with a transition and vice-versa. It is an oriented arch but the arrow can be omitted if the direction is up-down.

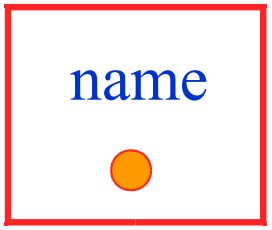
Sequential Functional Chart



A name or number is associated to each phase to clearly identify it



The initial phase is indicated by a doubl square or by two additional vertical lines in the square



An active phase can be indicated by a ball inside the square

Sequential Functional Chart

A boolean variable X_{nome} is associated to each phase

$X_{nome} = 1 \Rightarrow$ **Active** phase

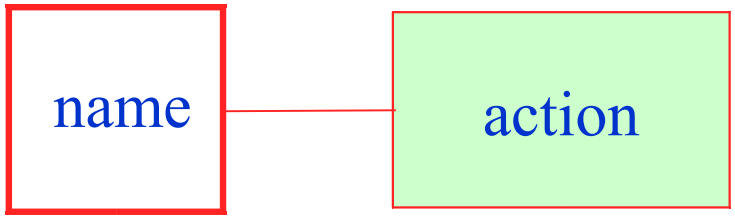
$X_{nome} = 0 \Rightarrow$ **Not active** phase

Set including all the phase variable defins the state of the system. It is named "*marking*" in the di Petri Net paradigm

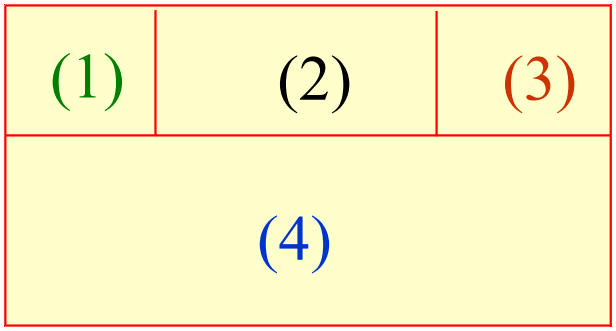
$$\mathbf{M} = \{X_i : i = 1,2,\ldots,N\} \in \{1,0\}^N$$

A real variable t_{name} is associated to each phase, rappresenting the time elapsed since its last activation

Sequential Functional Chart



To a phase can be associated one (or more) action which is executed when the phase is active, according to its qualifier



(1) Action qualifier

(2) Action name

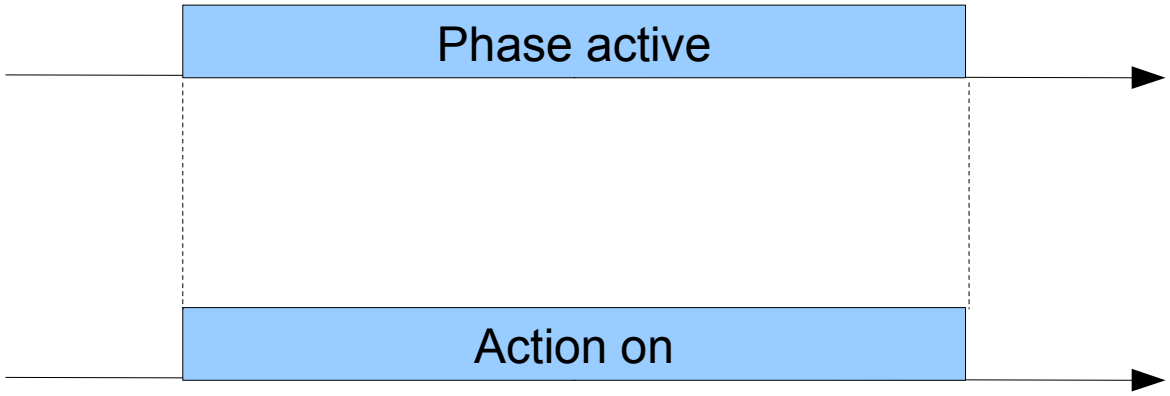
(3) A variable that says the action is ended

(4) Action body

Sequential Functional Chart

Action qualifier

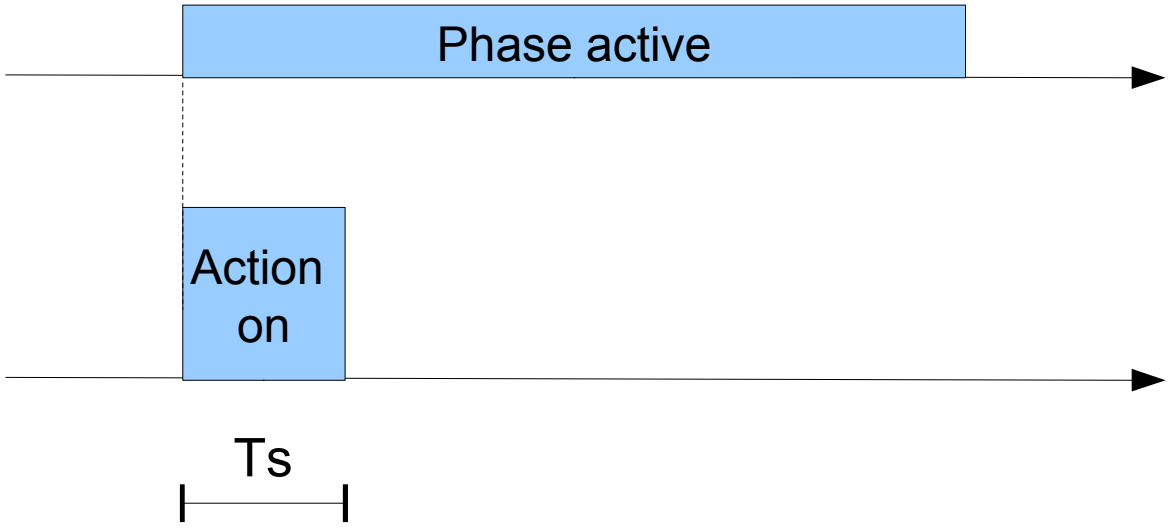
N – normal action, it is executed when the phase is active



Sequential Functional Chart

Action qualifier

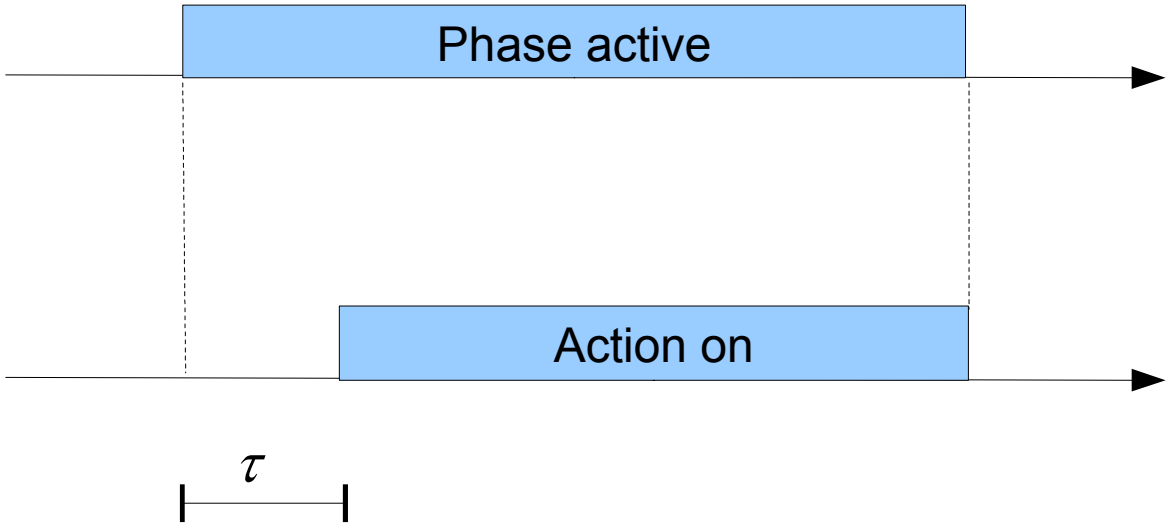
P – impulsive action, it is executed just once when the action is activated



Sequential Functional Chart

Action qualifier

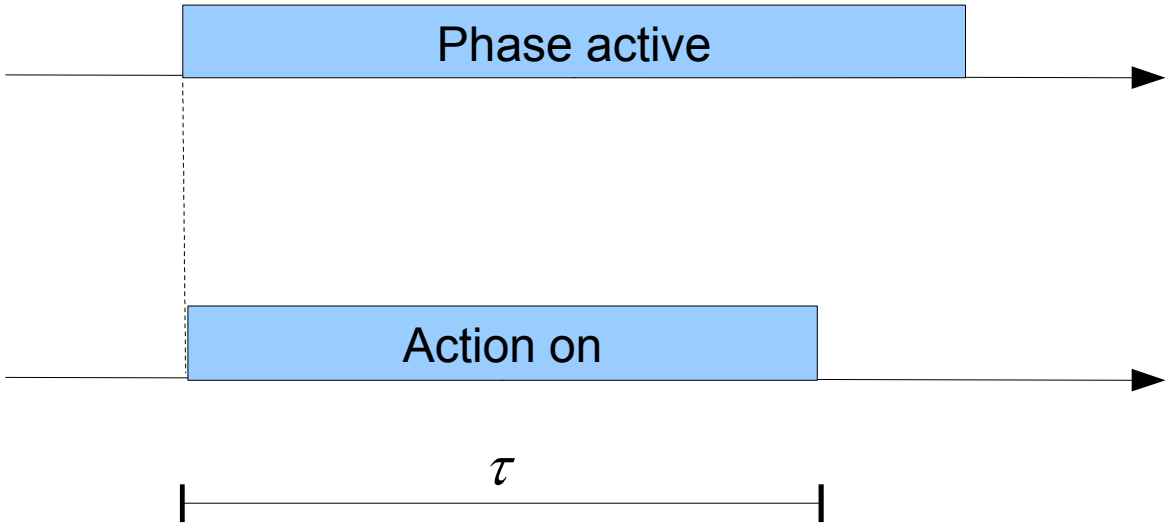
D – delayed, it is executed after the phase is active for the time τ



Sequential Functional Chart

Action qualifier

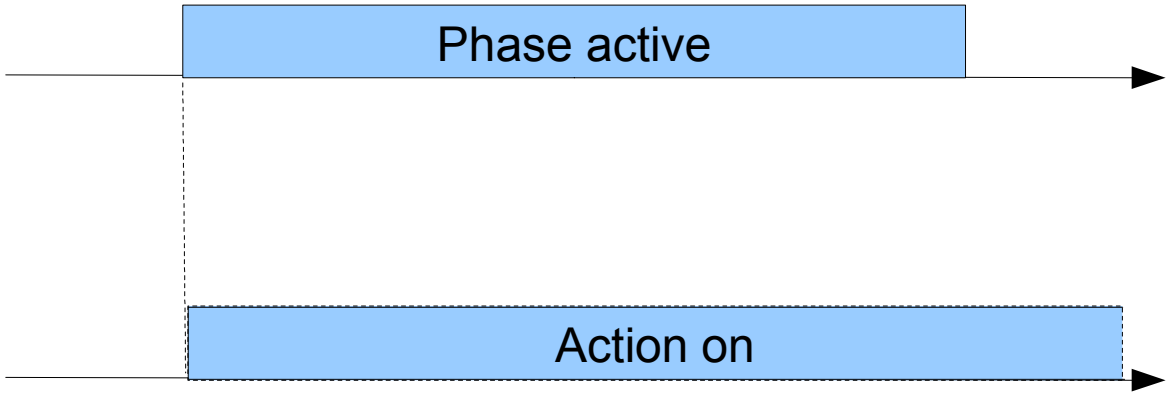
L – limited, it is executed at most for the time τ when the phase is active



Sequential Functional Chart

Action qualifier

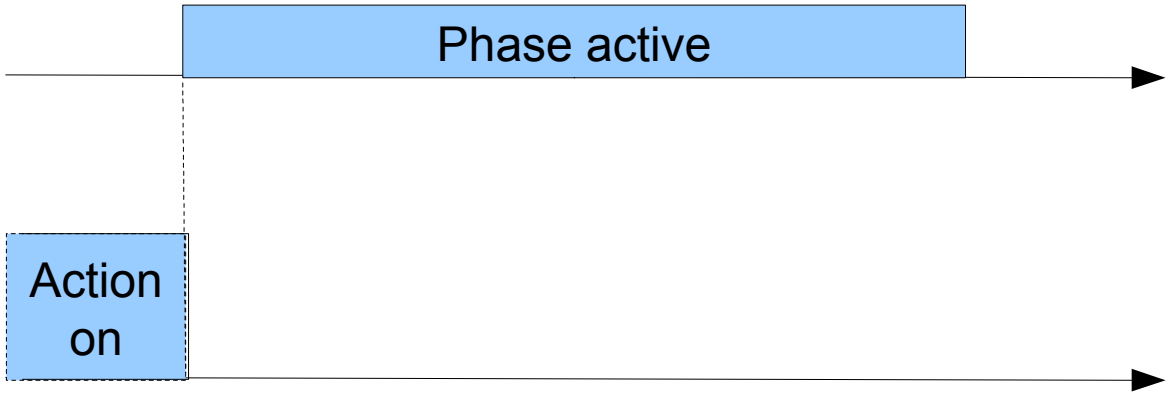
S – set, it is executed even if the phase is deactivated



Sequential Functional Chart

Action qualifier

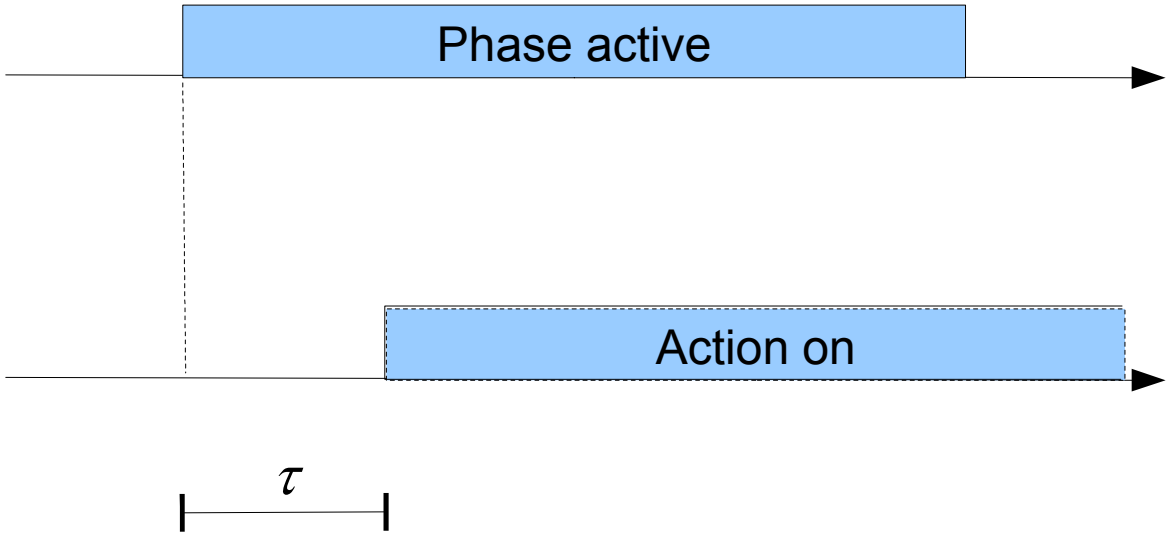
R – reset, it stops the execution of a setted action associated to the phase



Sequential Functional Chart

Action qualifier

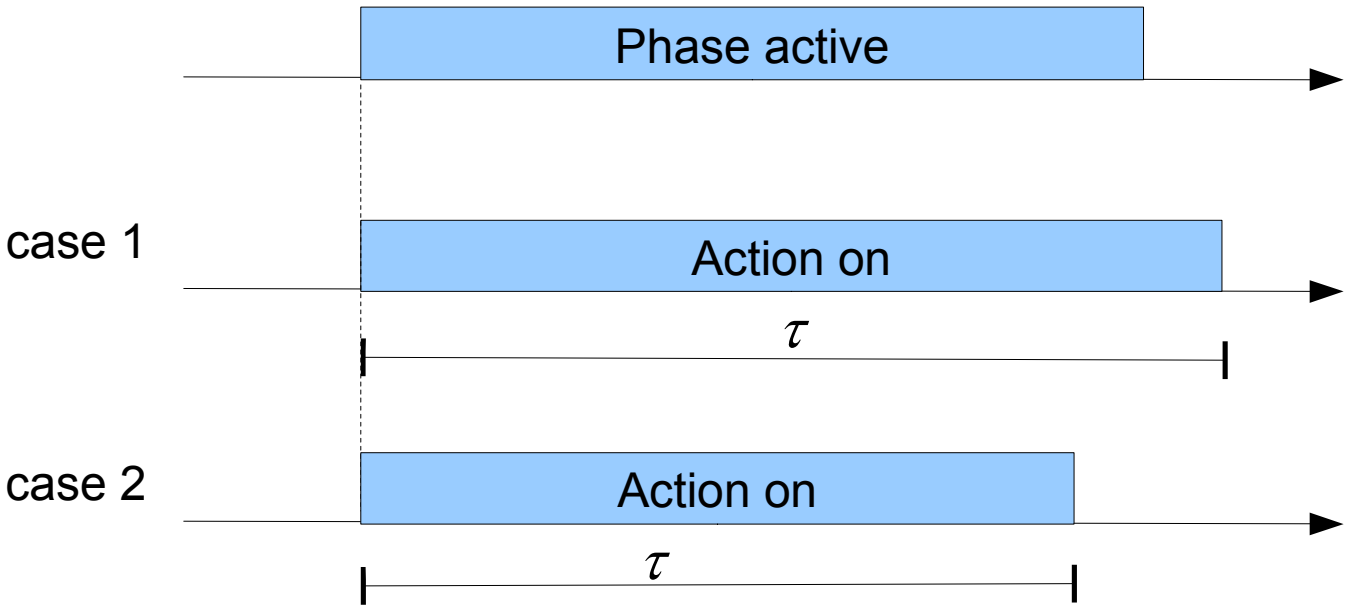
DS – delayed and set



Sequential Functional Chart

Action qualifier

SL – set and limited



Sequential Functional Chart

Action qualifier

N – normal action, it is executed when the pahse is active

P – impulsive action, it is executed just once when the action is activated

D – delayed, it is executed after the phase is active for the time τ

L – limited, it is executed at most for the time τ when the phase is active

S – set, it is executed even if the phase is deactivated

R – reset, it stops the execution of a setted action associated to the phase

Any sensible combination of above qualifiers

Sequential Functional Chart

Action body

It defines the executive commands associated to the action.

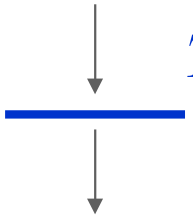
It can be defined using any programmin language, and even by an SFC graph.

A boolean variable Y_{nome} is associated to each action

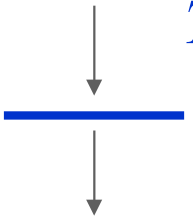
$Y_{nome} = 1 \Rightarrow$ **Active** action

$Y_{nome} = 0 \Rightarrow$ **Not active** action

Sequential Functional Chart



A name or a number is associated to each transition to clearly identify it

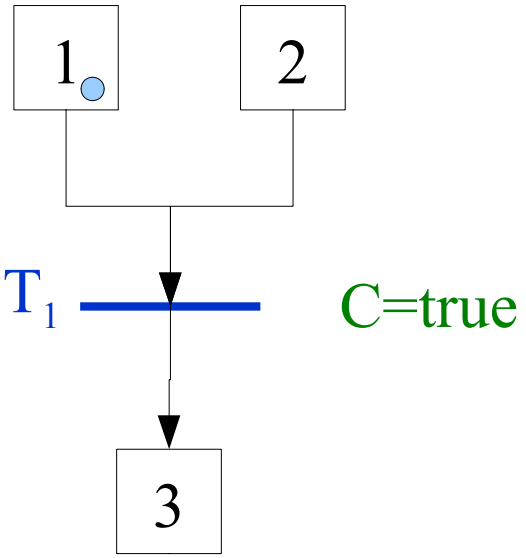


A condition can be associated to each transition; if omitted it corresponds to a true condition

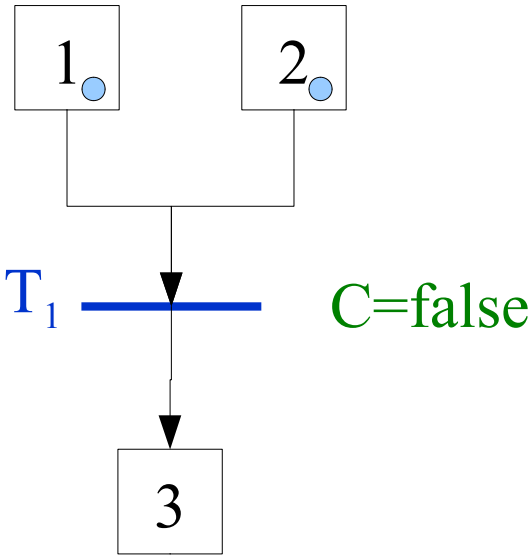
The transition is active if all the upstream phases are active

The transition can be fired (occurs) if it is active and the conition is true

Sequential Functional Chart

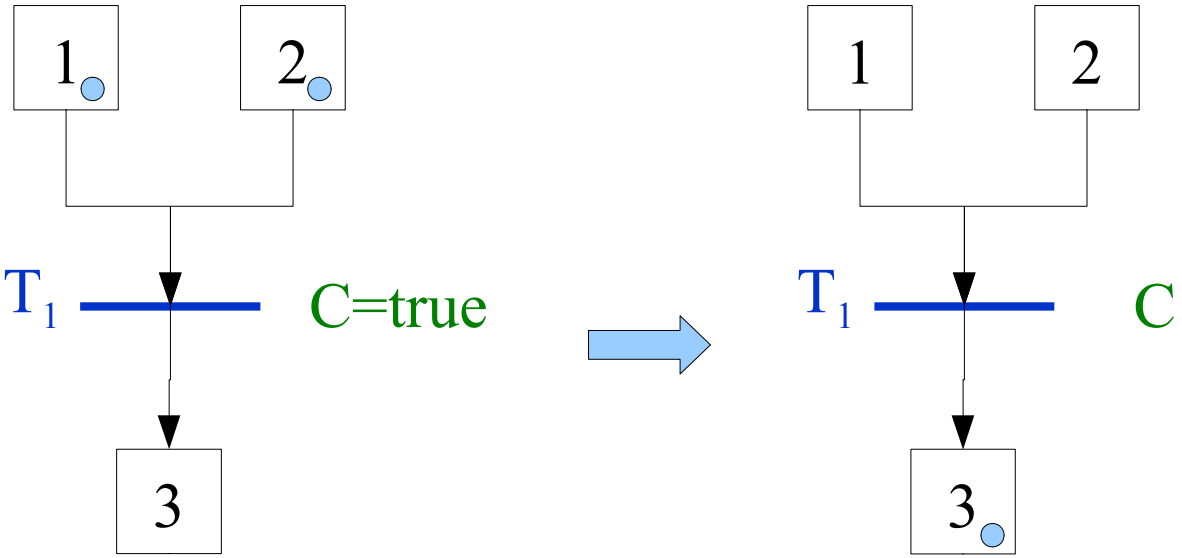


Transition is not active



Transition is active but cannot fire

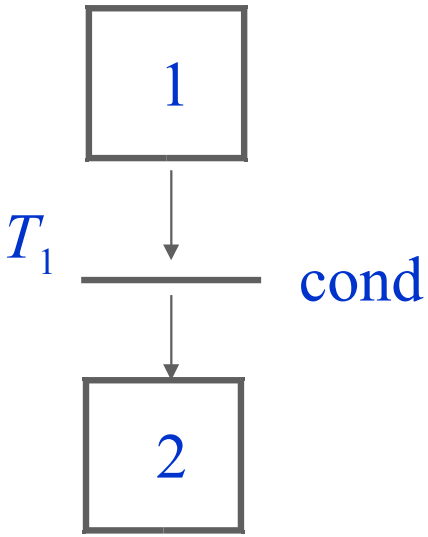
Sequential Functional Chart



Transition fires

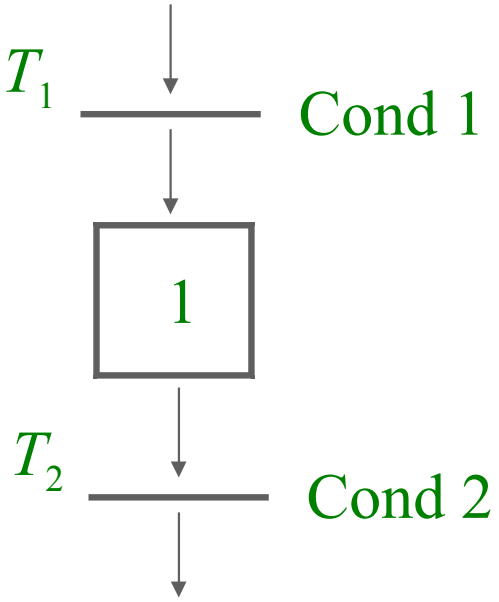
Phases 1 and 2 are disactivated
Phase 3 is activated

Sequential Functional Chart

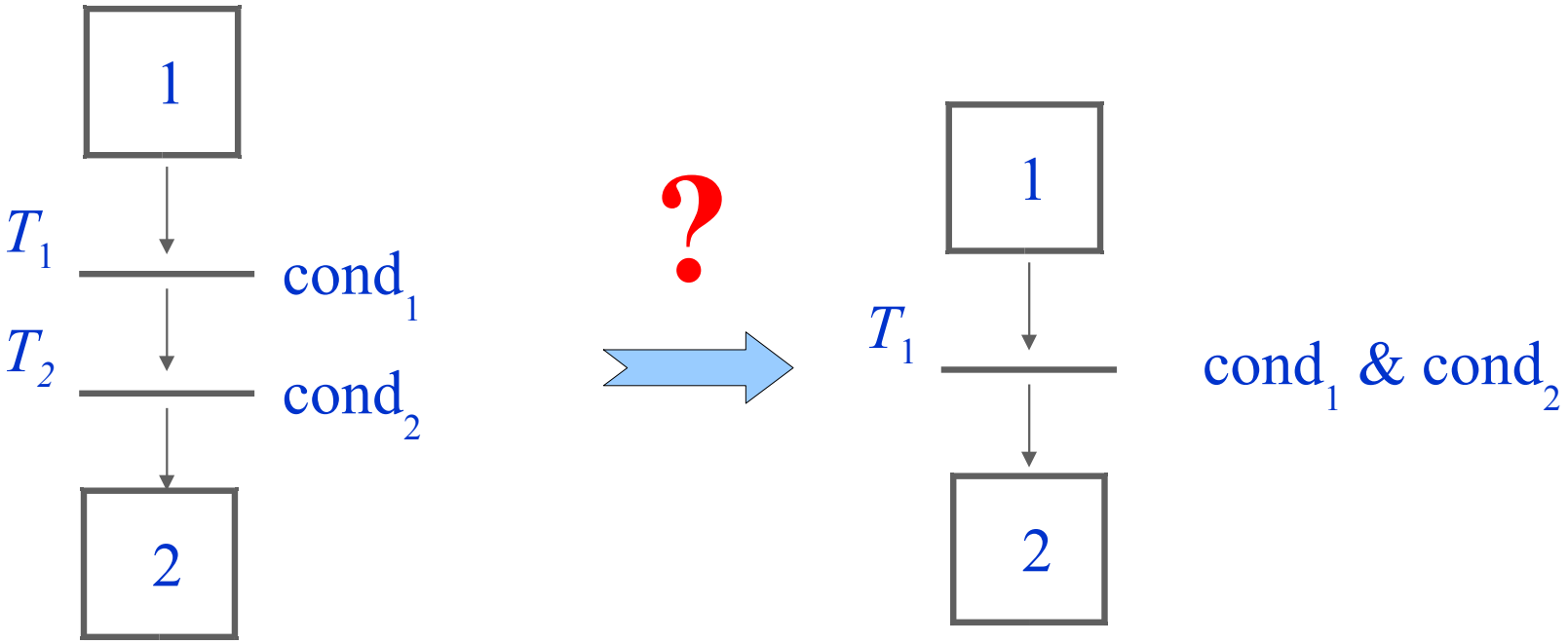


Between two connected phases there must be a transition

Between two connected transitions there must be a phase

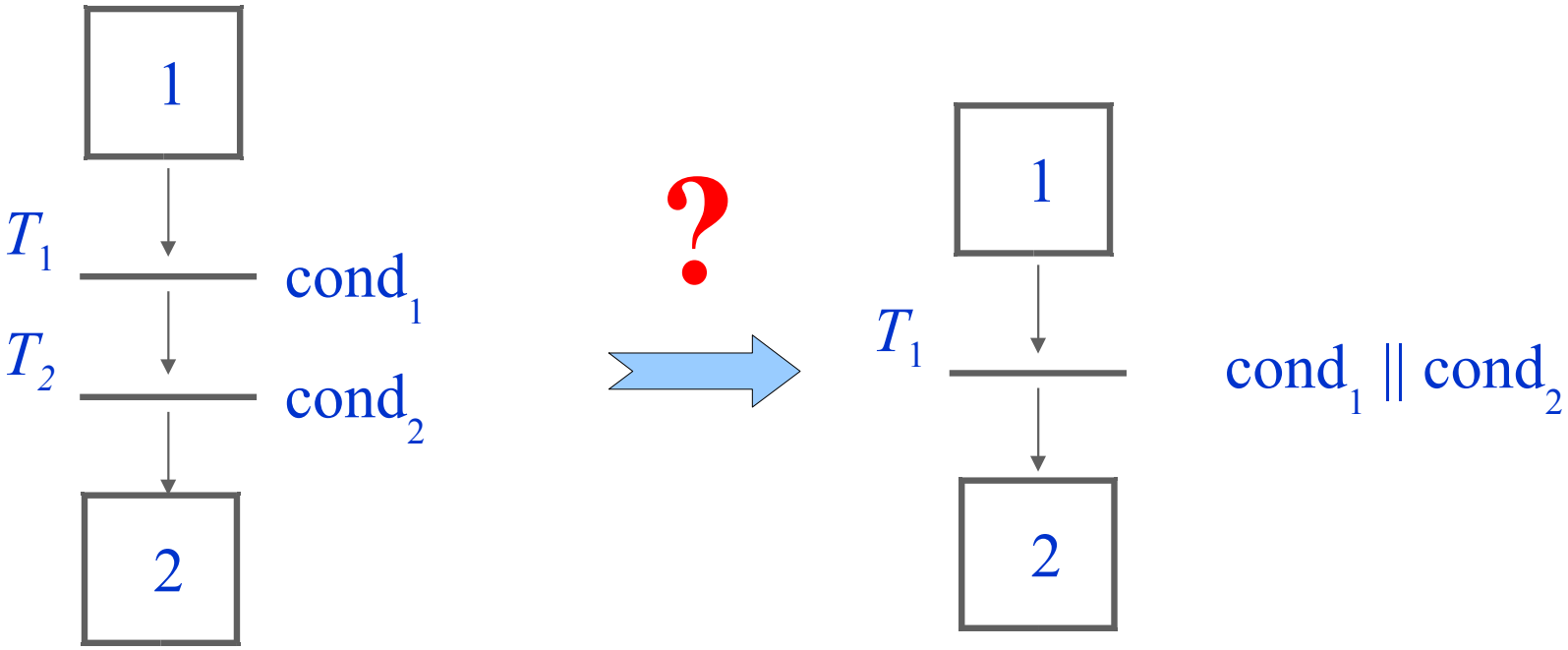


Sequential Functional Chart



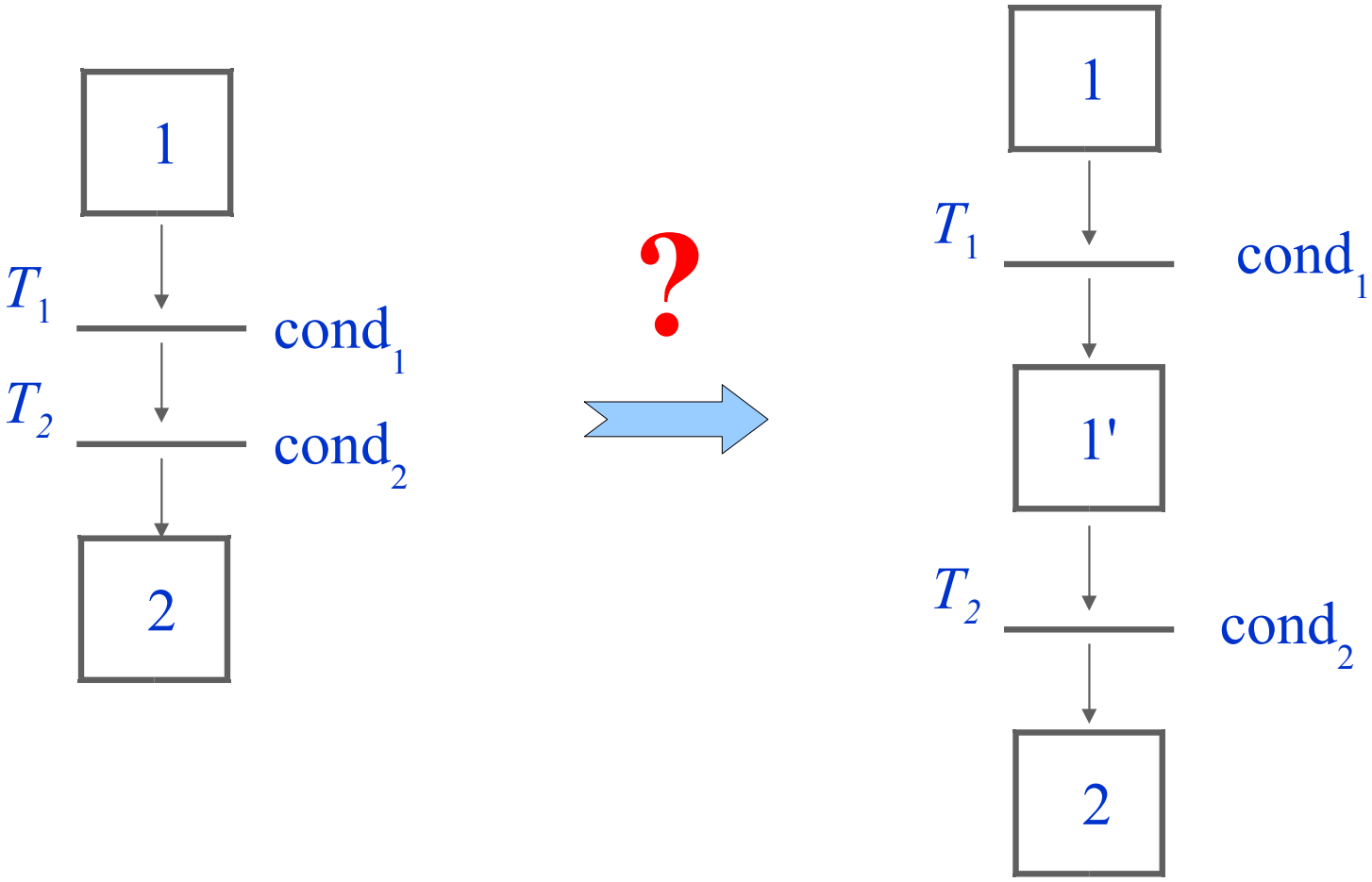
Between two connected phases there must be a transition

Sequential Functional Chart



Between two connected phases there must be a transition

Sequential Functional Chart



Between two connected phases there must be a transition

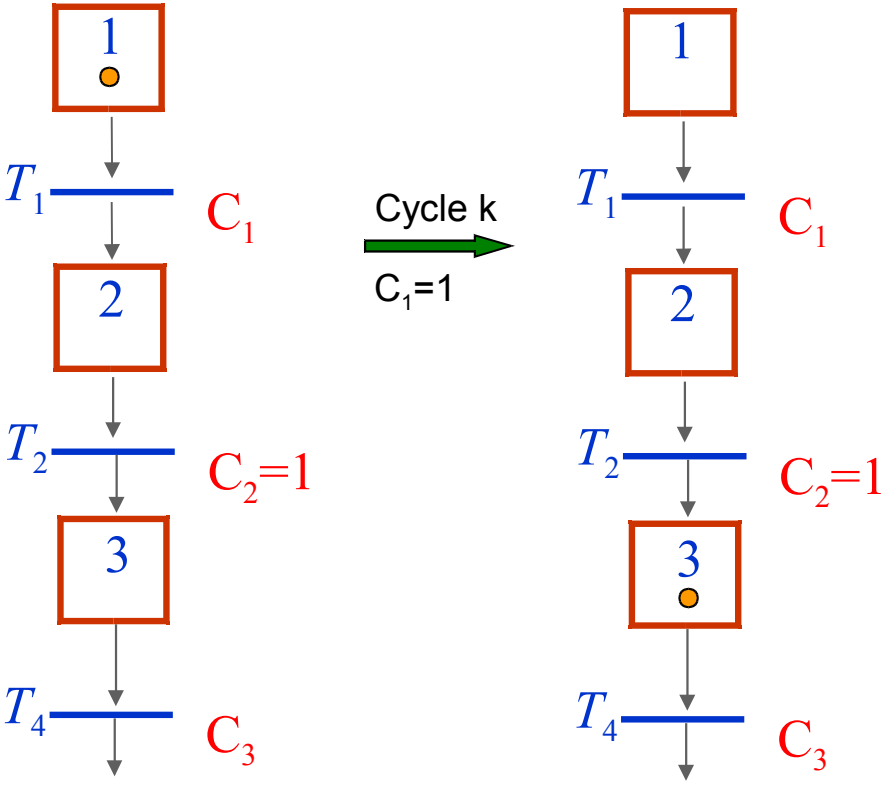
Sequential Functional Chart

Graph evolution rules

- If a transition can be fired, it is!
- When a transition fires, all upstream phases are de-activated and all the downstream phases are activated
- If there is more than a transition that can be fired at the same time, they are all fired contemporarely
- An unstable phase (i.e., a downstream transition can be fired at the same time instant its upstream phase is activated) the phase is activated for one scan cycle, at least

Traduzione da SFC a LD

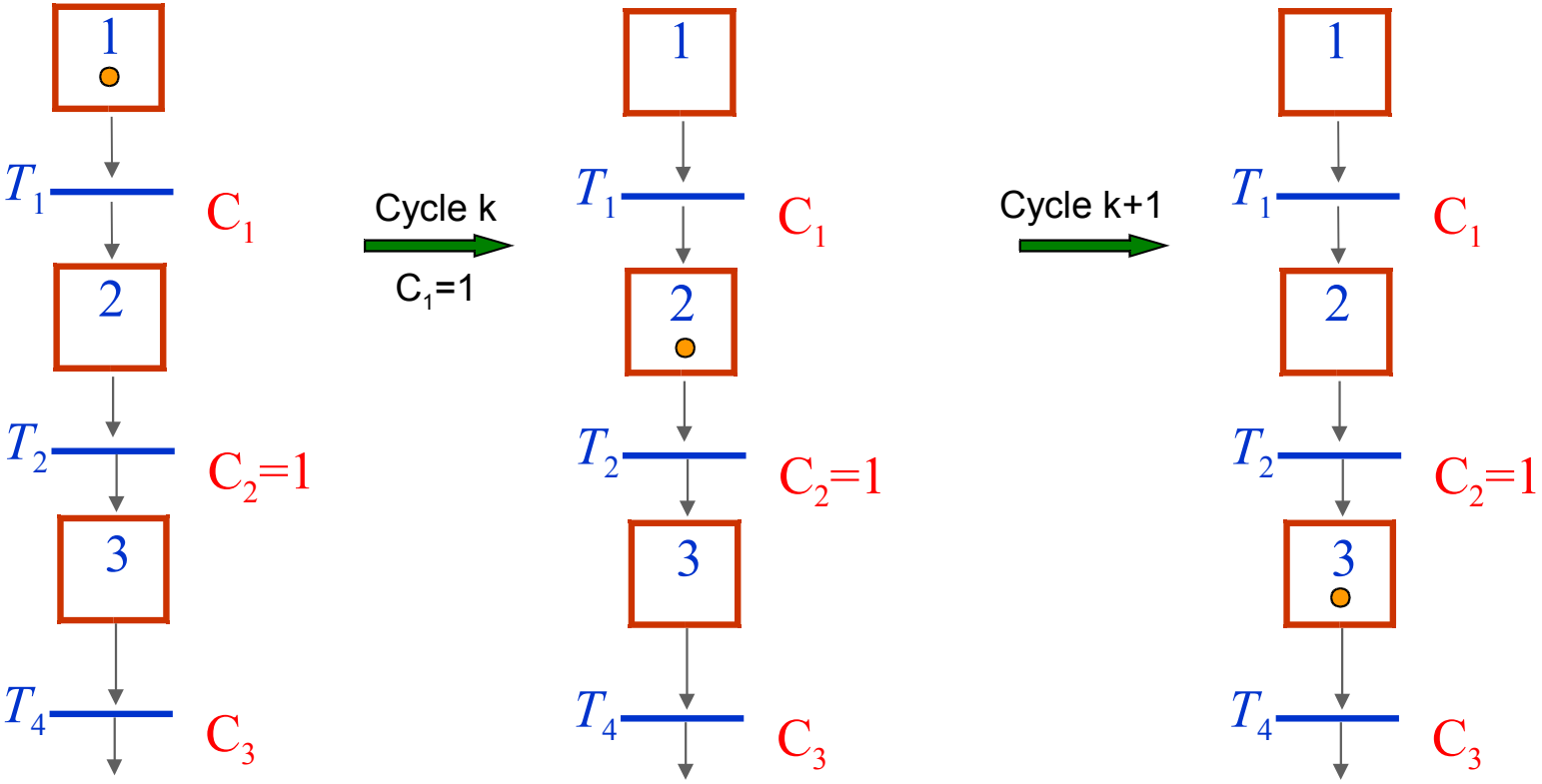
Phase 2 is an unstable phase



The action eventually associated to phase 2 is not executed

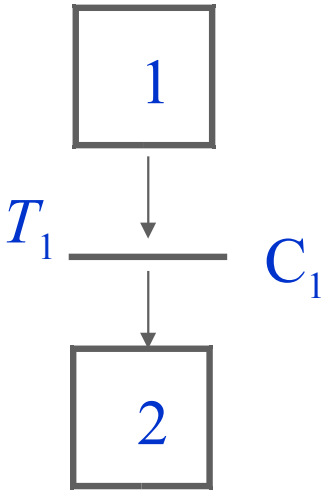
Traduzione da SFC a LD

Phase 2 is an unstable phase but it is activated for 1 cycle



Sequential Functional Chart

Sequence

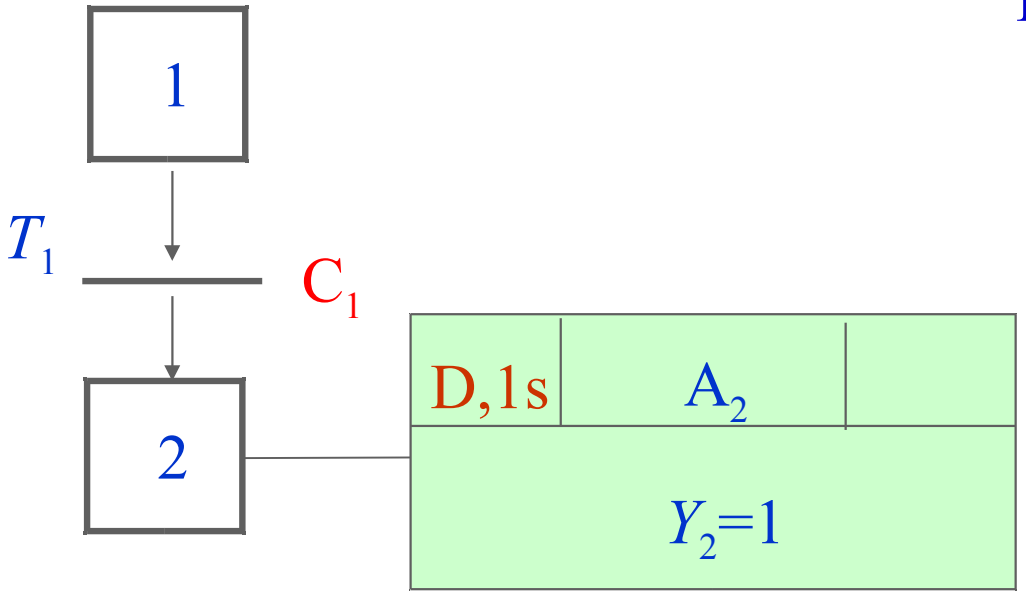


Equivalent MATLAB code

```
if and((x1==1),(C1==1))
    x1=0;
    x2=1;
else
    x1=x1;
    x2=x2;
end
```

Sequential Functional Chart

Sequence

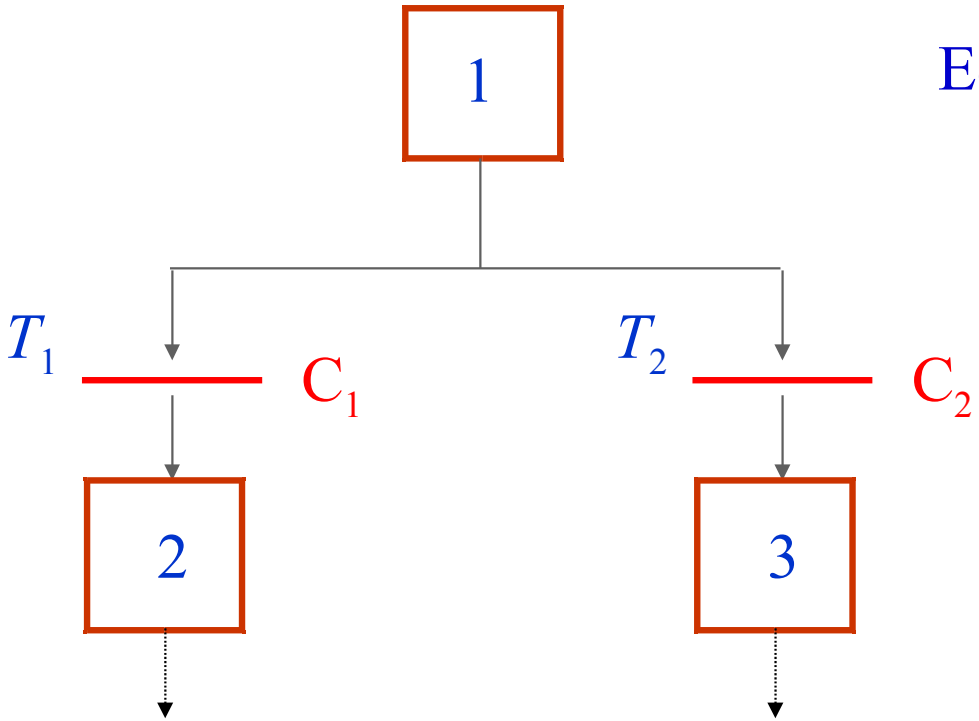


Equivalent MATLAB code

```
if and((x1==1),(C1==1))
    x1=0;
    x2=1;
else
    x1=x1;
    x2=x2;
end
if and((x2==1),(t2>1))
    Y2=1;
else
    Y2=0;
end
```

Sequential Functional Chart

Choice C_1 AND $C_2 \equiv 0$



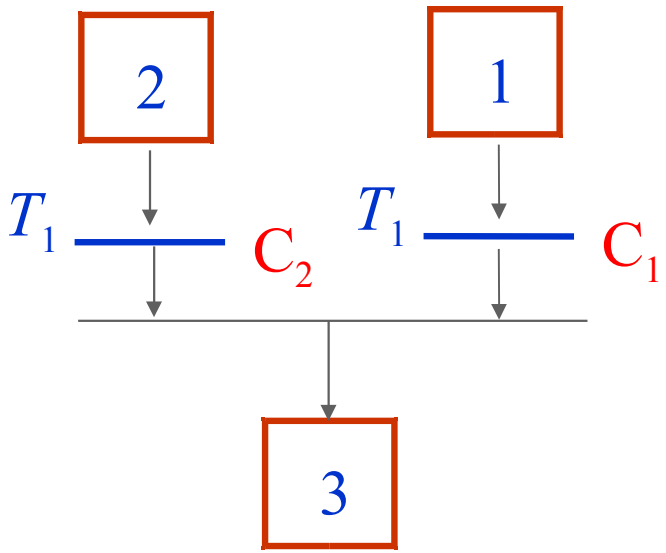
Alternative paths

Equivalent MATLAB code

```
if and((x1==1),(c1==1))
    x1=0;
    x2=1;
    x3=x3;
elseif and((x1==1),(c2==1))
    x1=0;
    x2=x2;
    x3=1;
else
    x1=x1;
    x2=x2;
    x3=x3;
end
```

Sequential Functional Chart

Convergence



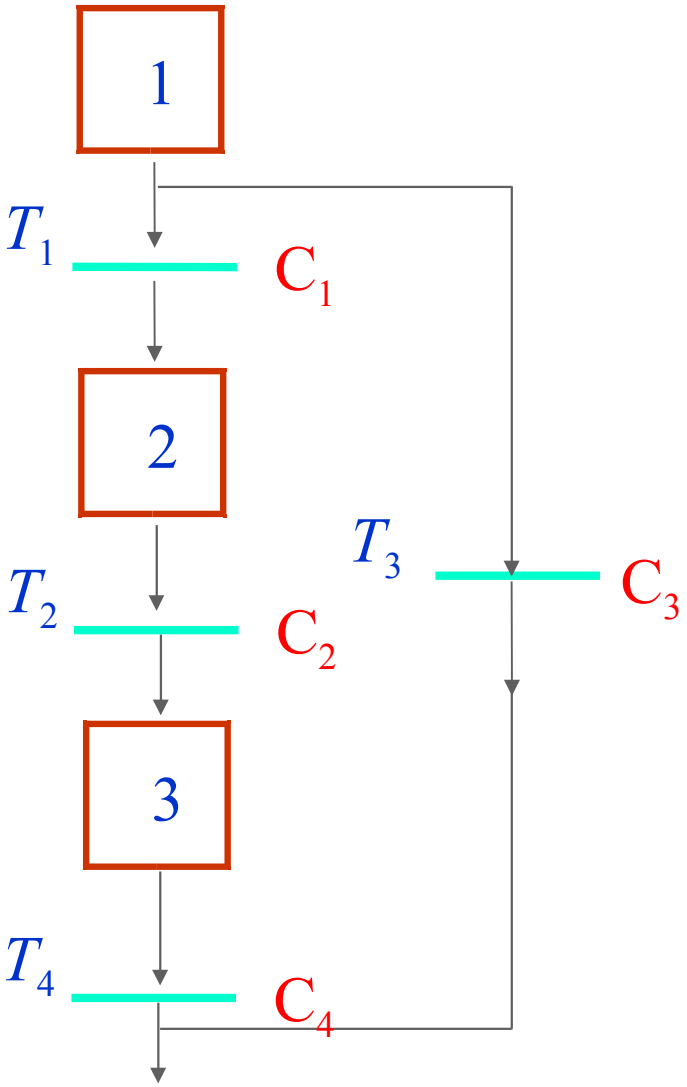
Different paths converge to the same phase

Convergence to phase 3 of the different paths can occur at different time instants

Equivalent MATLAB code

```
if or(and((x1==1),(c1==1)),and((x2==1),(c2==1)))
   if and((x1==1),(c1==1))
      x1=0;  x2=x2;
   end
   if and((x2==1),(c2==1))
      x1=x1; x2=0;
   end
   x3=1;
else
   x1=x1; x2=x2; x3=x3;
end
```

Sequential Functional Chart



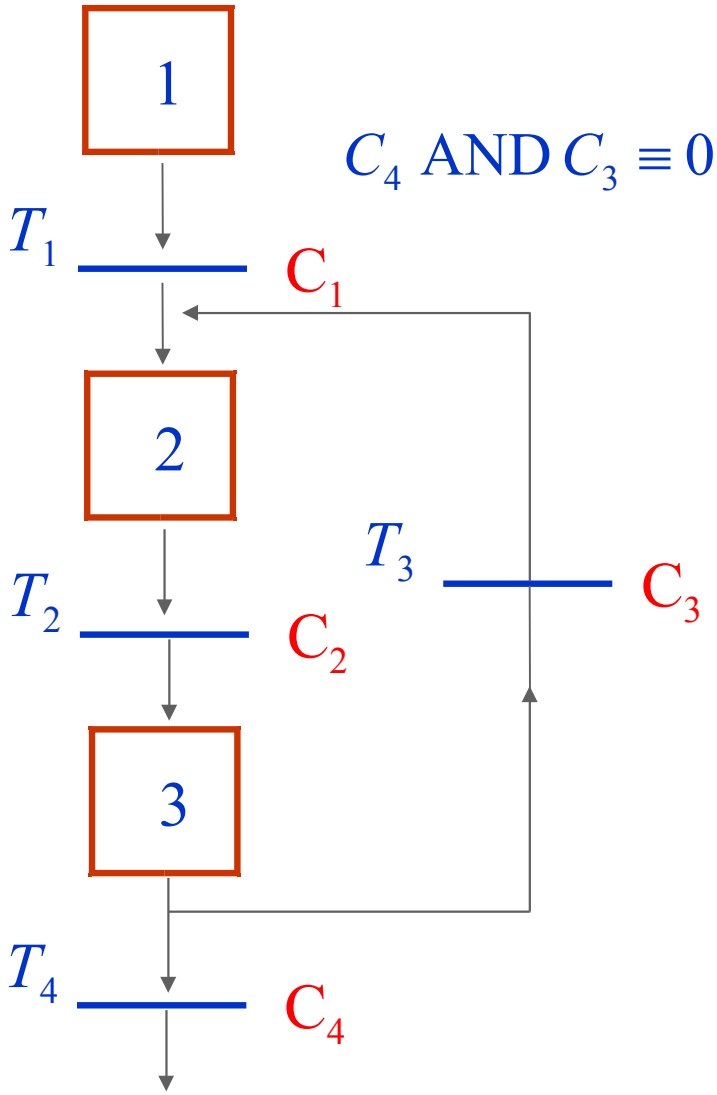
Sequence jump

C_1 AND $C_3 \equiv 0$

It is a special case of choice between two alternative paths

If C_3=1 a specific section of the program is not executed

Sequential Functional Chart

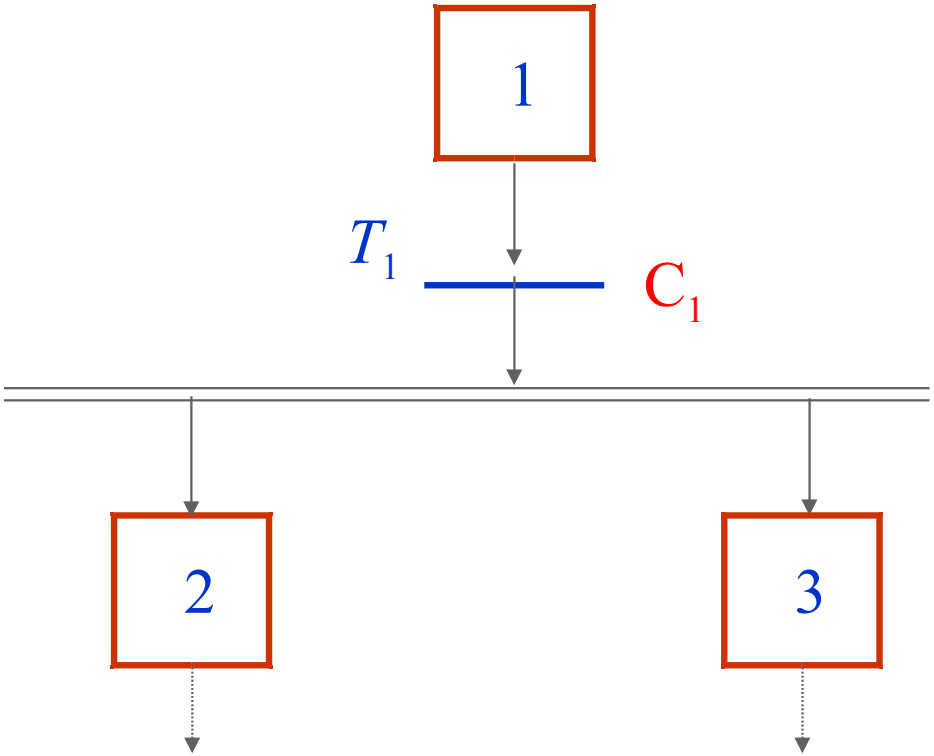


It is a special case of choice between two alternative paths

REPEAT
  IF X_2 AND C_2
    X_3=1, X_2=0
  ELSE
    X_2= X_2, X_3=X_3
  END IF
  IF X_3 AND C_3
    X_2=1, X_3=0
  ELSE
    X_3= X_3, X_2=X_2
  END IF
UNTIL (C_4=1) AND (X_3=1)

Sequential Functional Chart

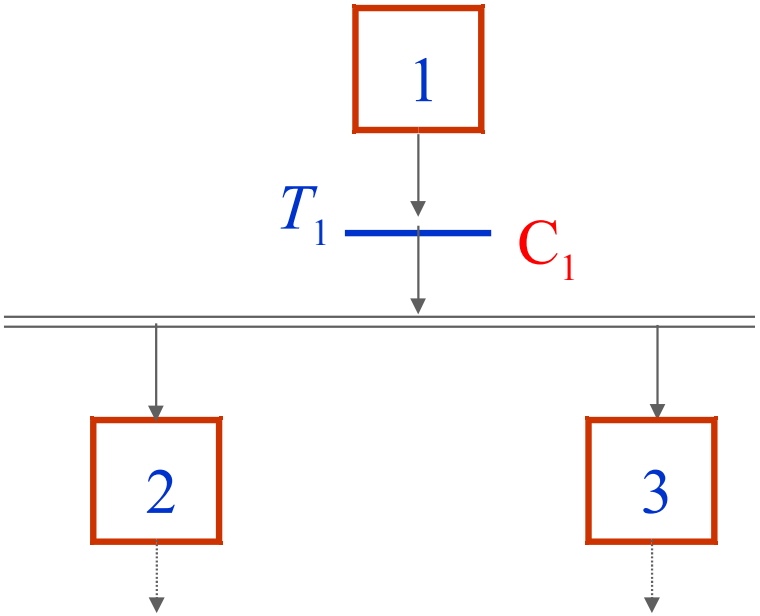
Paralell paths

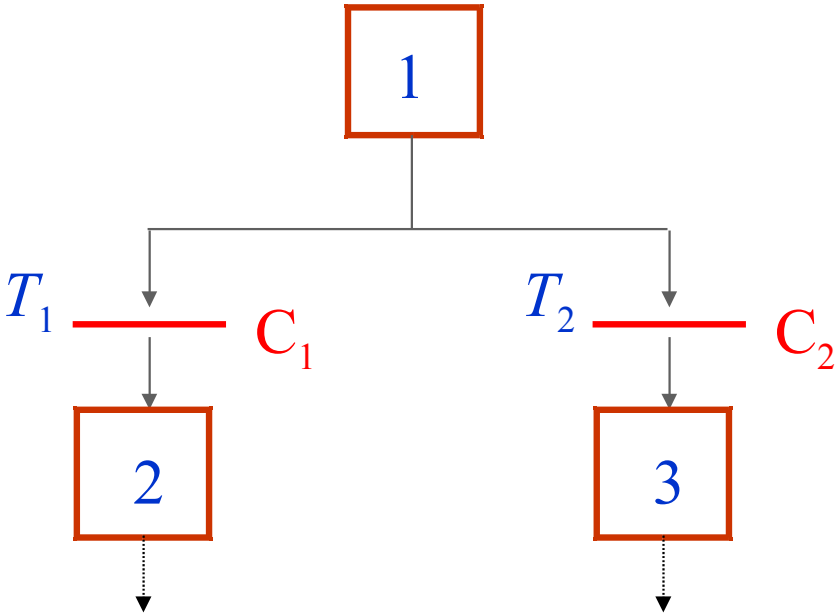


IF X_1 AND C_1
    X_2=1
    X_3=1
    X_1=0
ELSE
    X_1= X_1, X_2= X_2, X_3= X_3
END IF

Sequential Functional Chart

Difference between Paralell paths and Choice

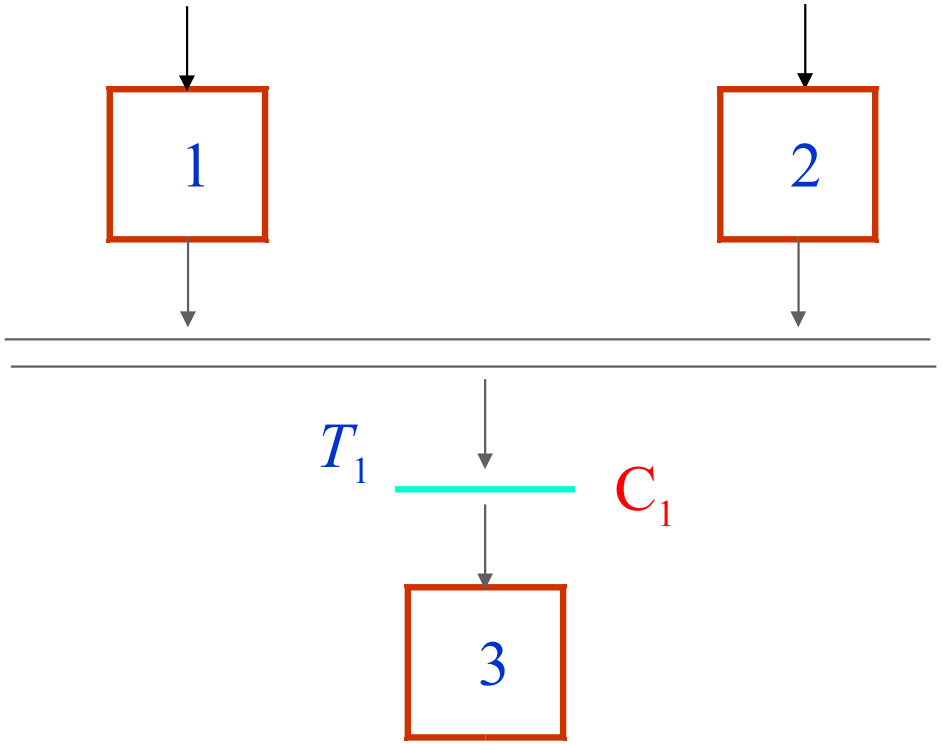




C_1 & $C_2 = 0$

Sequential Functional Chart

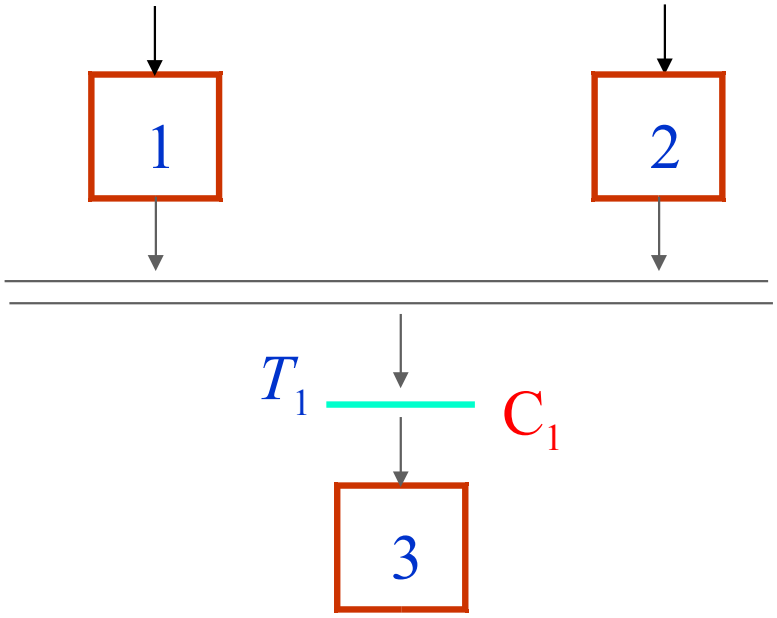
Synchronization

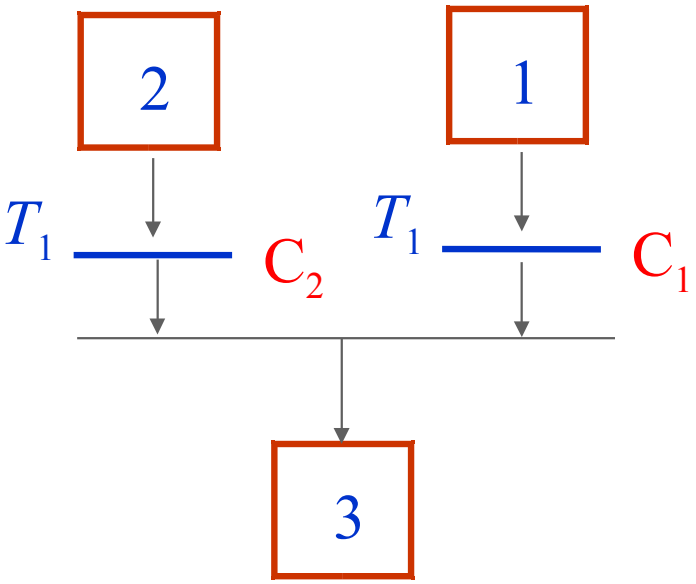


IF X_1 AND X_2 AND C_1
    X_1=0
    X_2=0
    X_3=1
ELSE
    X_1= X_1, X_2= X_2, X_3= X_3
END IF

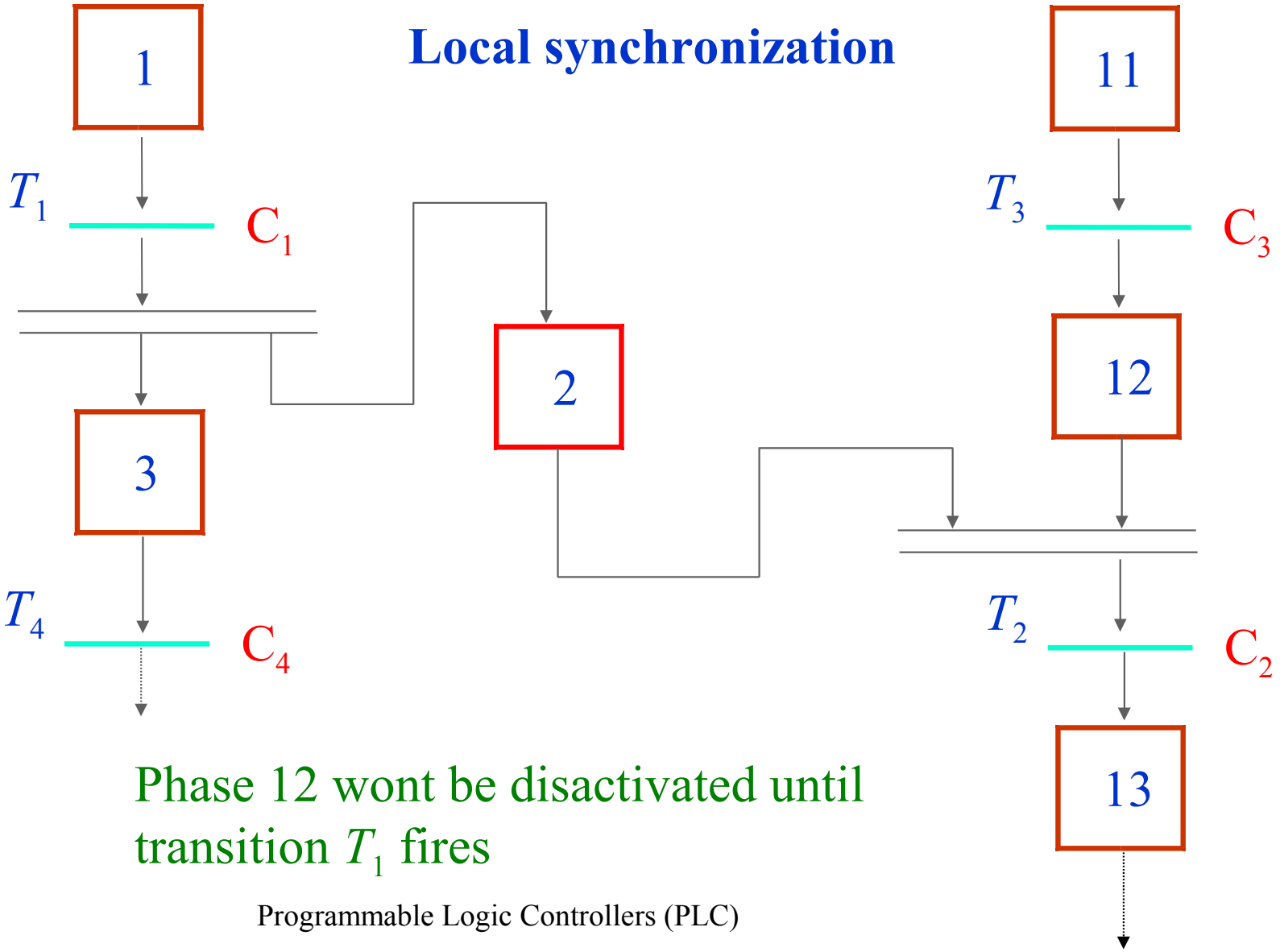
Sequential Functional Chart

Difference between Synchronization and Convergence





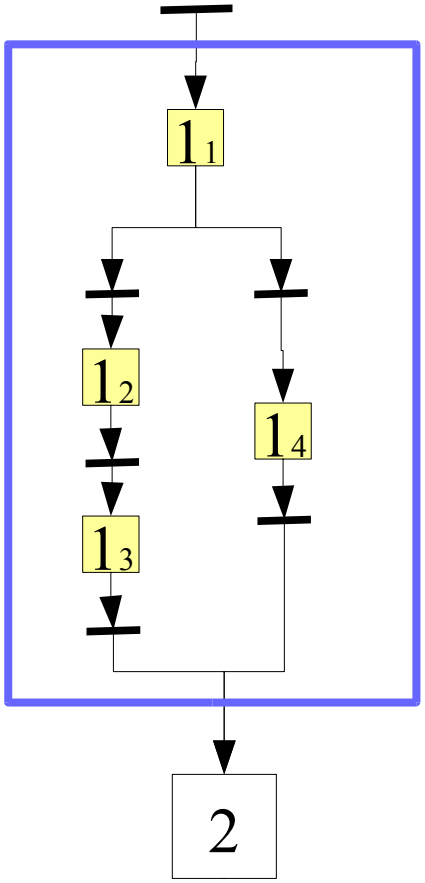
Sequential Functional Chart



Phase 12 wont be disactivated until transition T_1 fires

Sequential Functional Chart

Macrophase It is a phase with one input and one output embedding an FSC



Phase 1

Care should be taken with the evolution rules of the phases within the macro-phase.

Particularly for the internal phases with actions.

Sequential Functional Chart

Macroaction — Allows for the interaction and gerarchy between PLC*s*

Forcing

Assign X:{Y}; i.e., **M**(X):=**Y**

It assign a specific marking Y to the PLC state X

Idle

It is a peculiar forcing to the null state

Assign X:{0}; i.e., **M**(X):=**0**

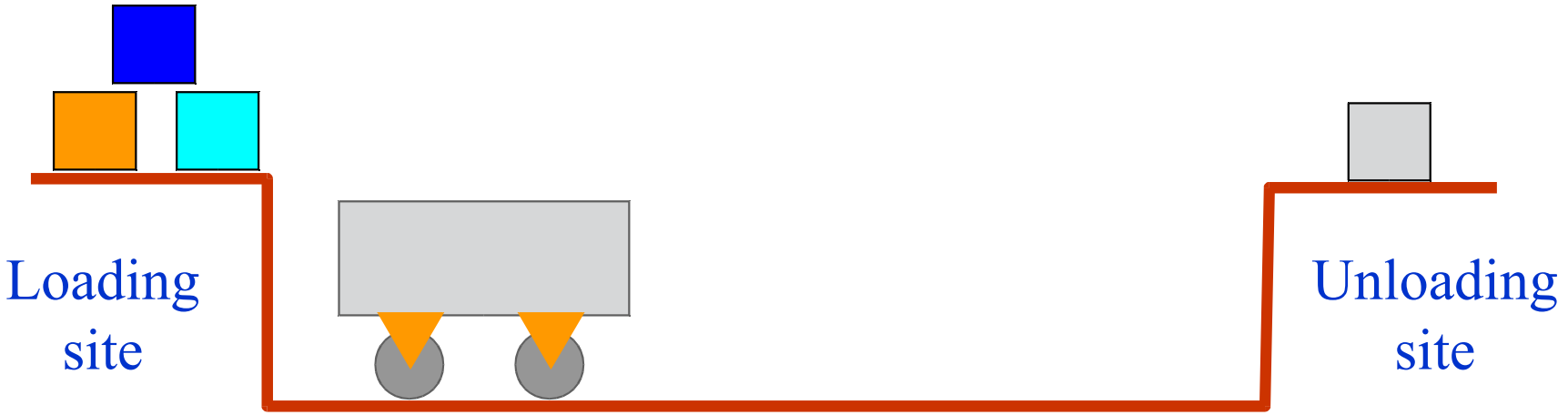
Block

Assign X:{*}; i.e., **M**(X):=**X**

All transitions are inhibited

Programming example

Sequential control of a trolly carrying the load



A PLC on the trolley control its motion to move the packages from the left to the right without external connection.

The packages are positioned on the trolley by a small crane not connected with the PLC

Programming example

A reversible electrical motor with two velocities is used to move the trolley

The following sensors are installed on the trolley:

Strain-gauge on the trolley platform

One distance sensor at each side (front, back)

The PLC can be turned on at any position along the path

Programming example

Input

F – Strain-gauge (anologue variable)

Sl – left distance sensor (anologue variable)

Sd – right distance sensor (anologue variable)

Output

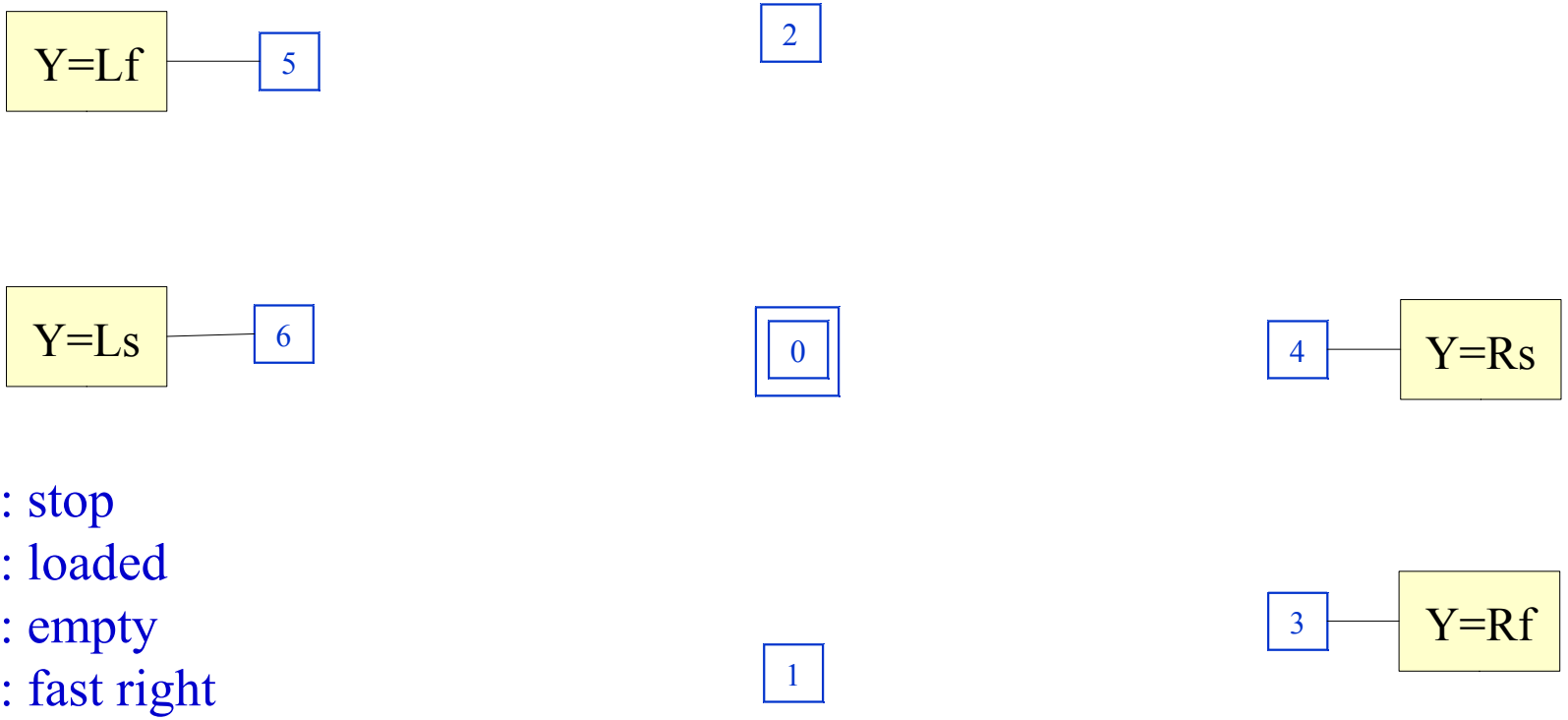
Rs – slow right motion command

Rf – fast right motion command

Ls – slow left motion command

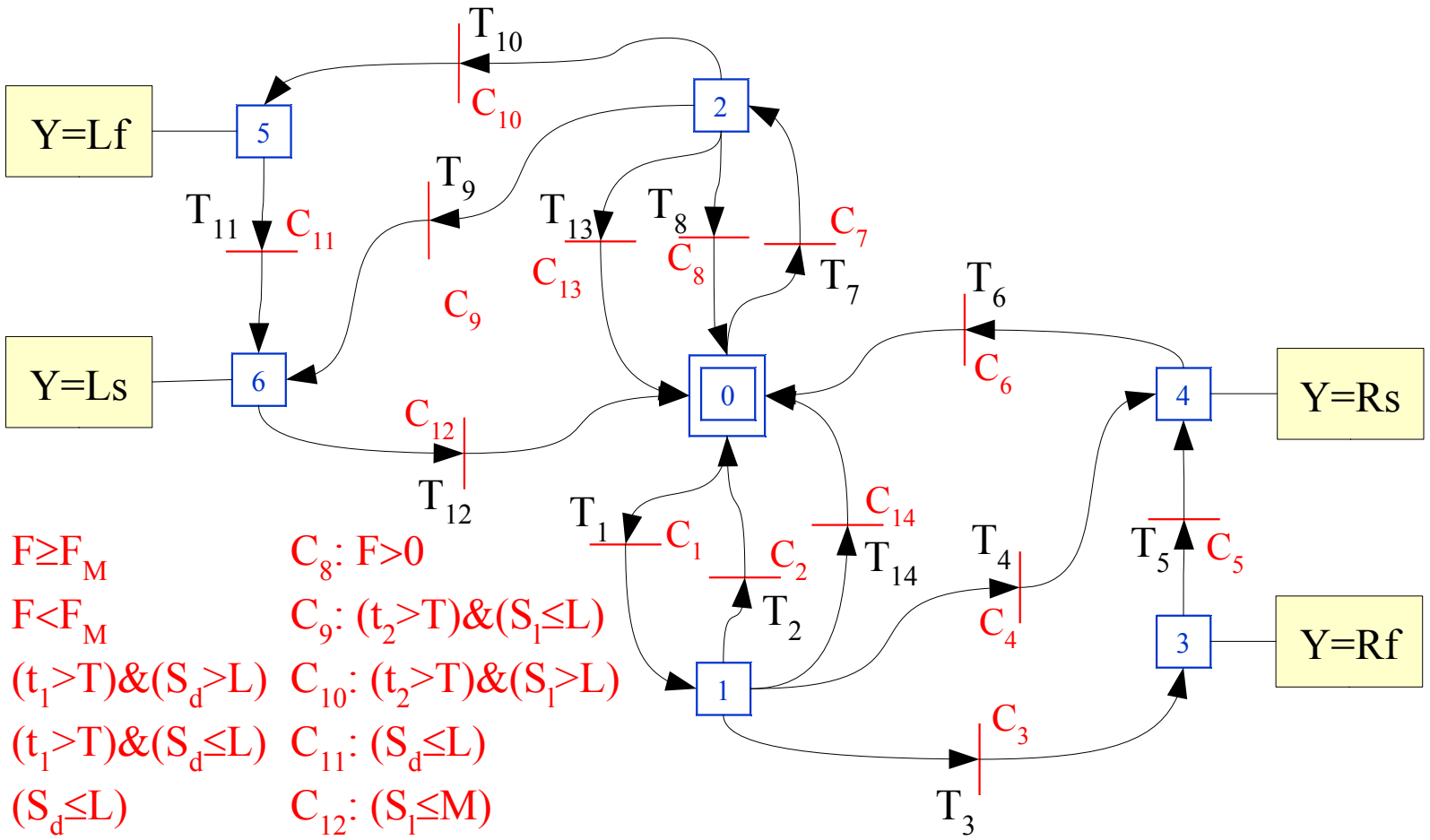
Lf – fast left motion command

Programming example



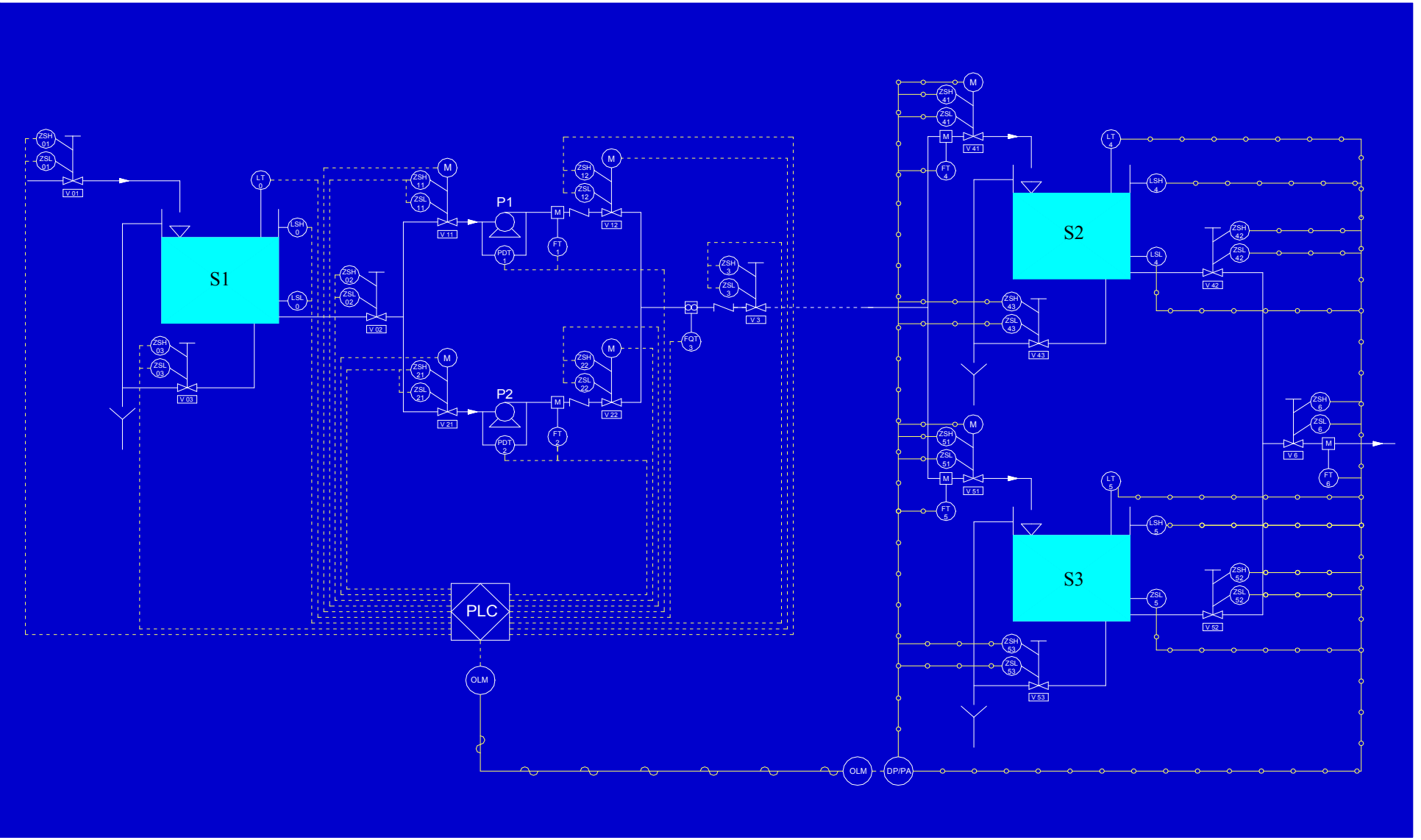
0: stop
1: loaded
2: empty
3: fast right
4: slow right
5: fast left
6: slow left

Programming example



C_1: $F \geq F_M$
C_2: $F < F_M$
C_3: $(t_1 > T) \& (S_d > L)$
C_4: $(t_1 > T) \& (S_d \leq L)$
C_5: $(S_d \leq L)$
C_6: $(S_d \leq M)$
C_7: $F = 0$
C_8: $F > 0$
C_9: $(t_2 > T) \& (S_l \leq L)$
C_{10}: $(t_2 > T) \& (S_l > L)$
C_{11}: $(S_d \leq L)$
C_{12}: $(S_l \leq M)$
C_{13}: $(S_l \leq M) \& F = 0$
C_{14}: $(S_d \leq M) \& F > 0$

Example: pumping plant



Example: pumping plant

Fieldbus

Lower site (S1): 27 digital inputs,5 analogue inputs,
12 digital outputs.

Upper site (S2): 23 digital inputs, 4 digital outputs

Fieldbus: Profibus DP

In the site: twisted pair cable connections

Connection between the sites: optical fiber with opto-electronic conversion devices

Control: 1 master unit (PLC) in the lower site. The master manages all slaves (transmitters/ *actuators*).

Example: pumping plant

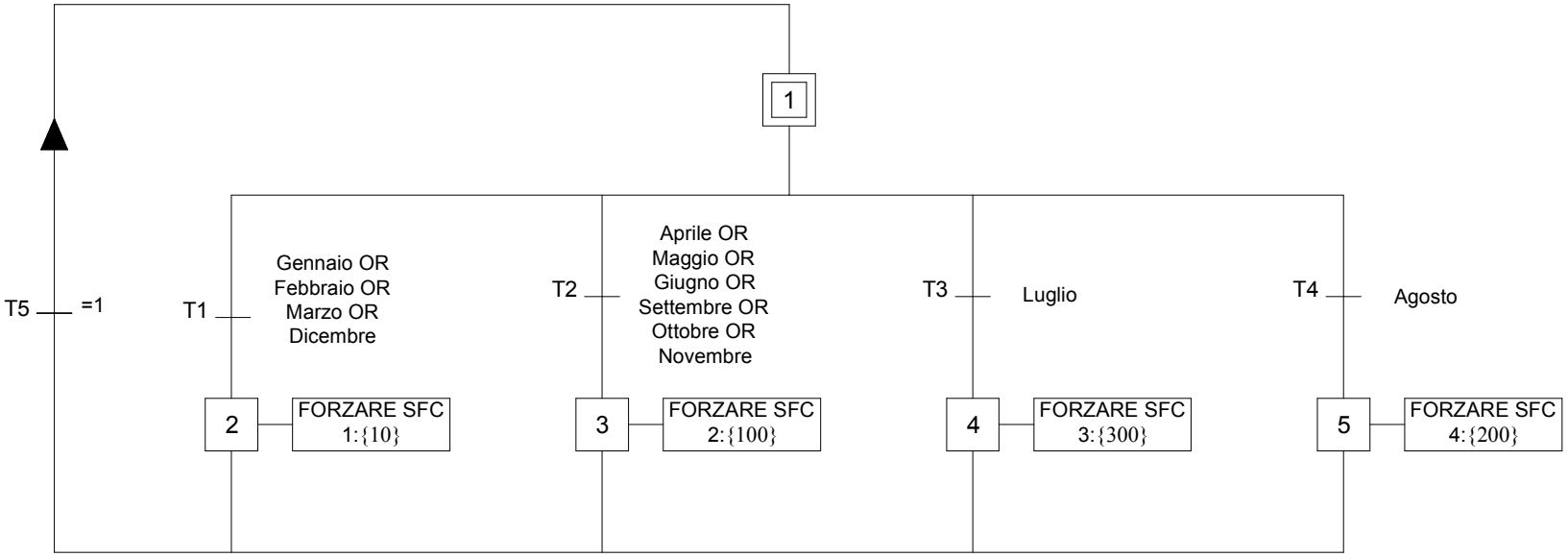
✓ Pumps shold work taking into account water levels in the tanks, water uses in different periods and energy costs

✓ Pumps should preferebly work when the energy cost is less

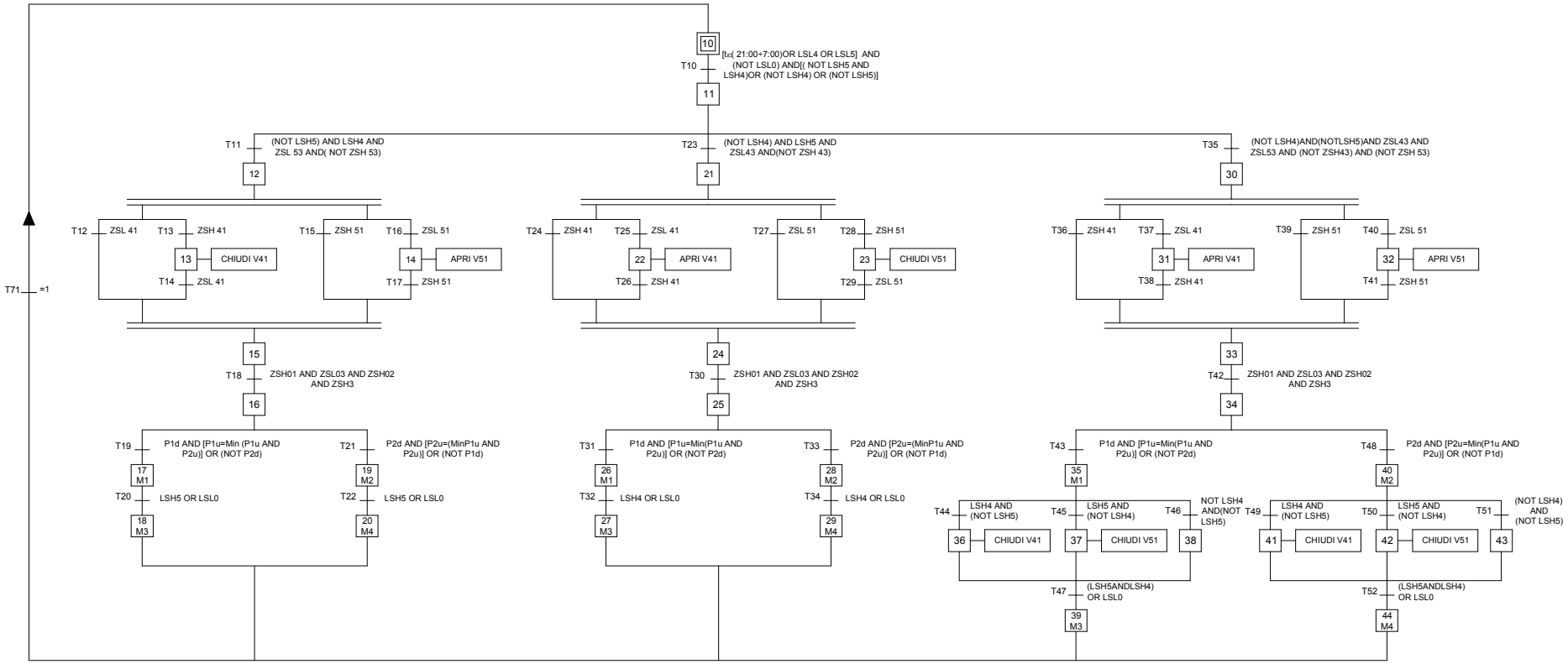
✓ The working sequence of the pumps must take into account the plant situation/configuration

✓ All pumps should be working for almost the same time

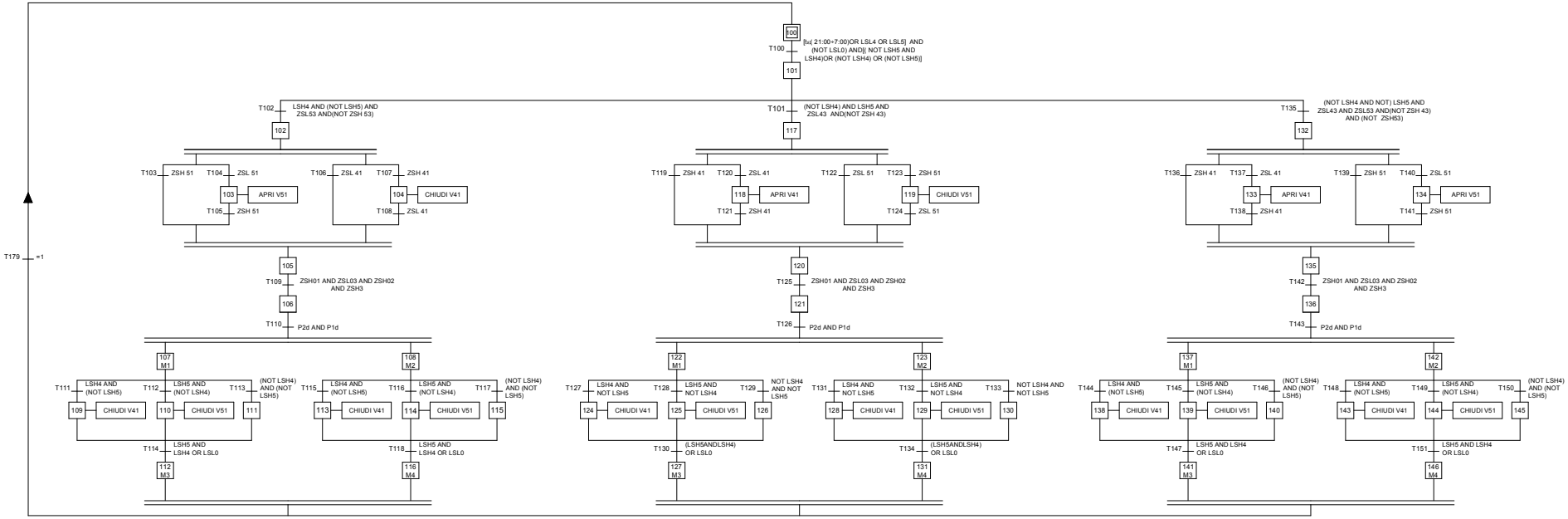
Example: pumping plant



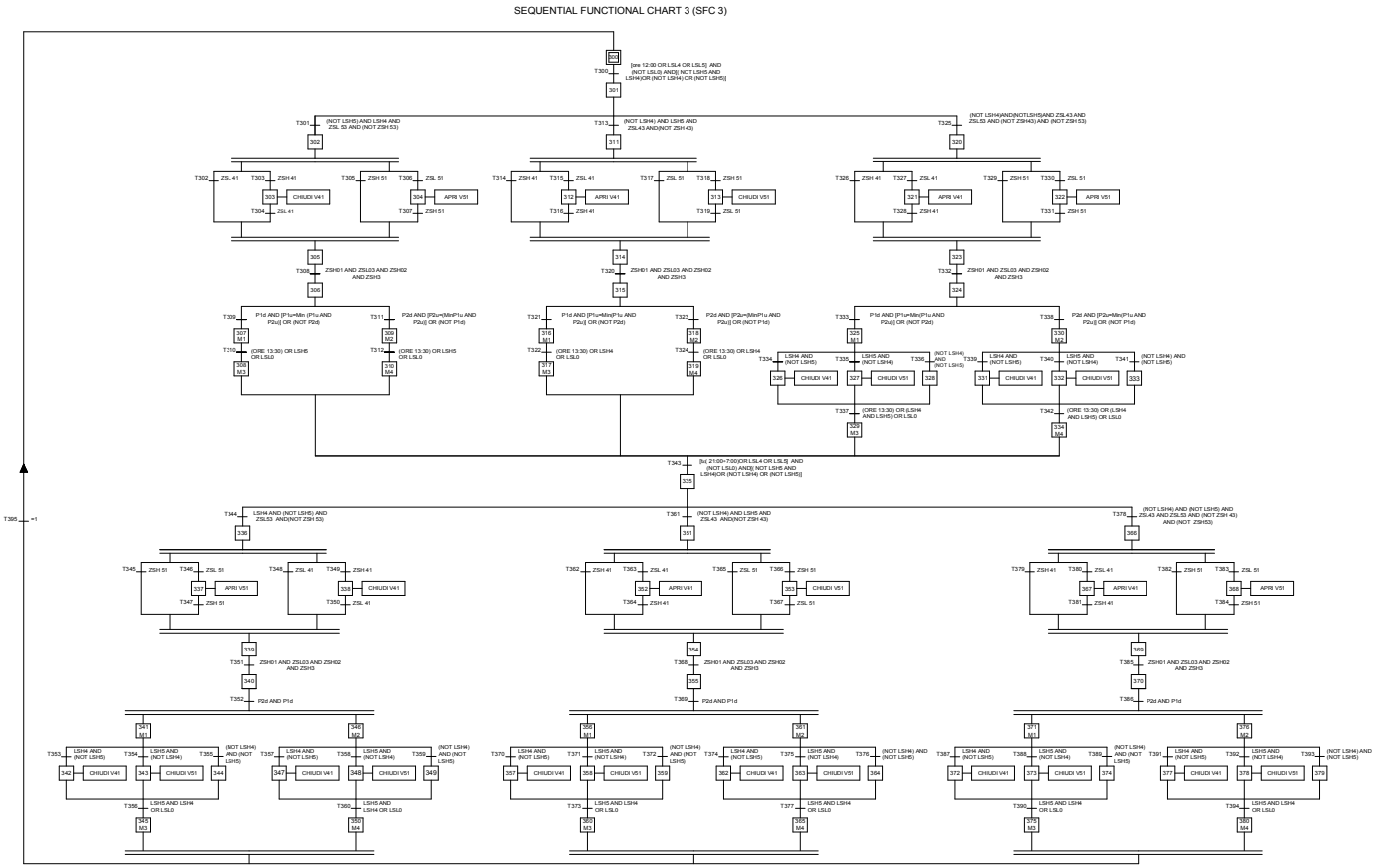
Example: pumping plant



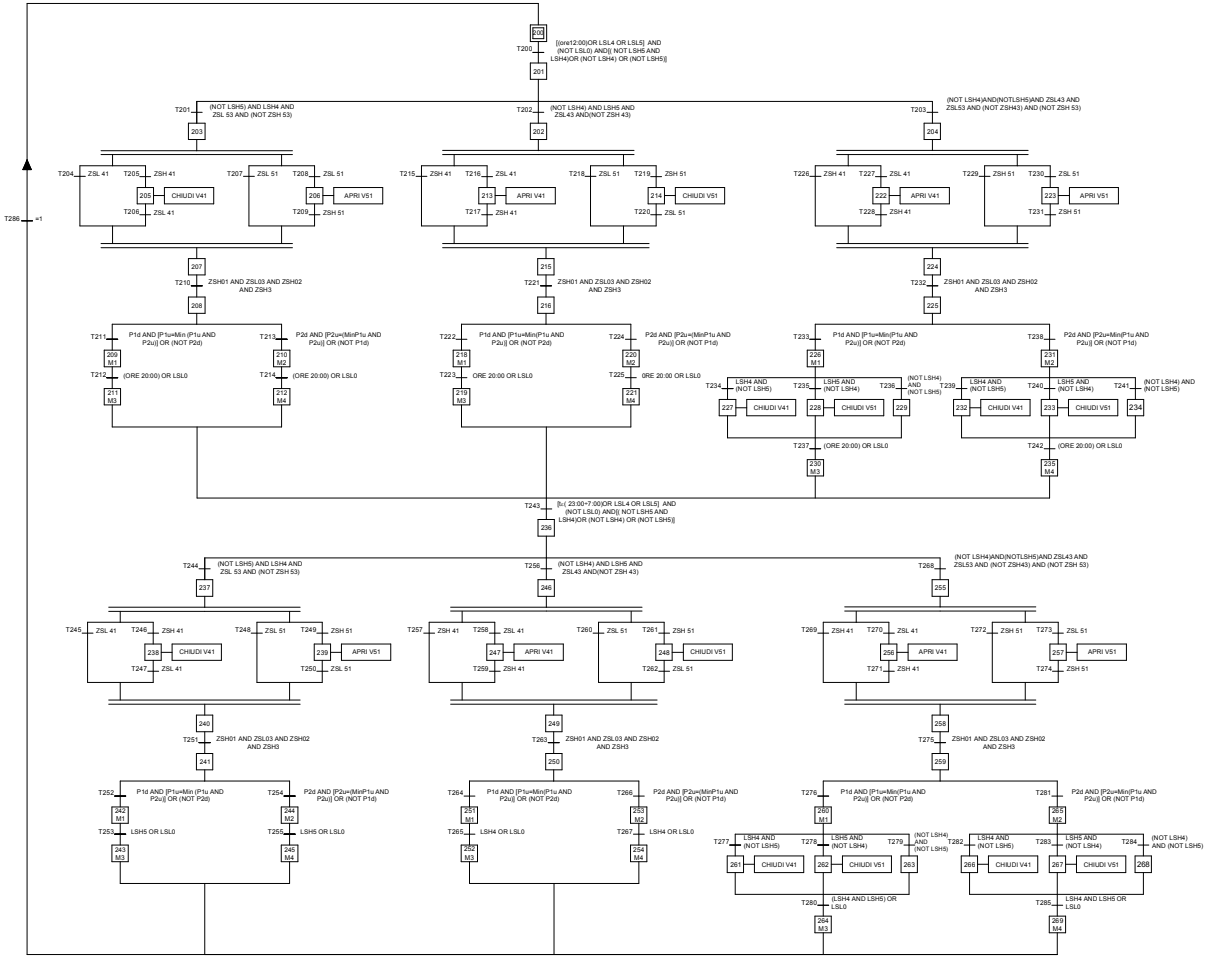
Example: pumping plant



Example: pumping plant



Example: pumping plant



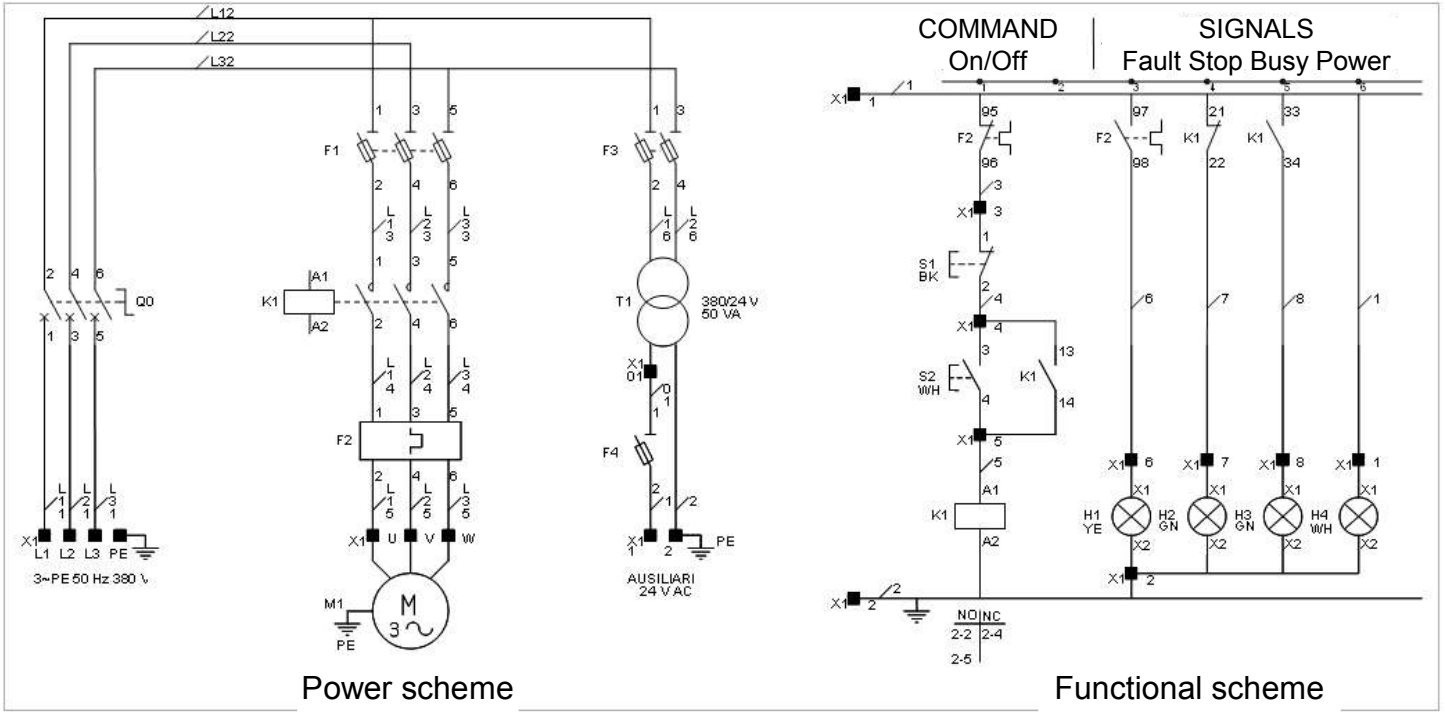
Example: pumping plant

Not taken into account

- Emergency Inputs
- Reset command by the operator
- Signals and warnings from the plant
- Blocks
- Storage of data
- Real-time optimization

Ladder diagram

Ladder Diagram (LD): *it is a simbolic programming language for PLC based on the rapresentation of the control logic by means of contacts (inputs/arguments) and coils (output/results) that mimics electrical schemes representing electromechanical logics used until '80th.*



Ladder diagram

Ladder Diagram (LD): *it is a simbolic programming language for PLC based on the rapresentation of the control logic by means of contacts (inputs/arguments) and coild (output/results) that mimics electrical schemes representing electromechanical logics used until '80th.*

Practically all industrial PLCs have LD as a programming tool option.

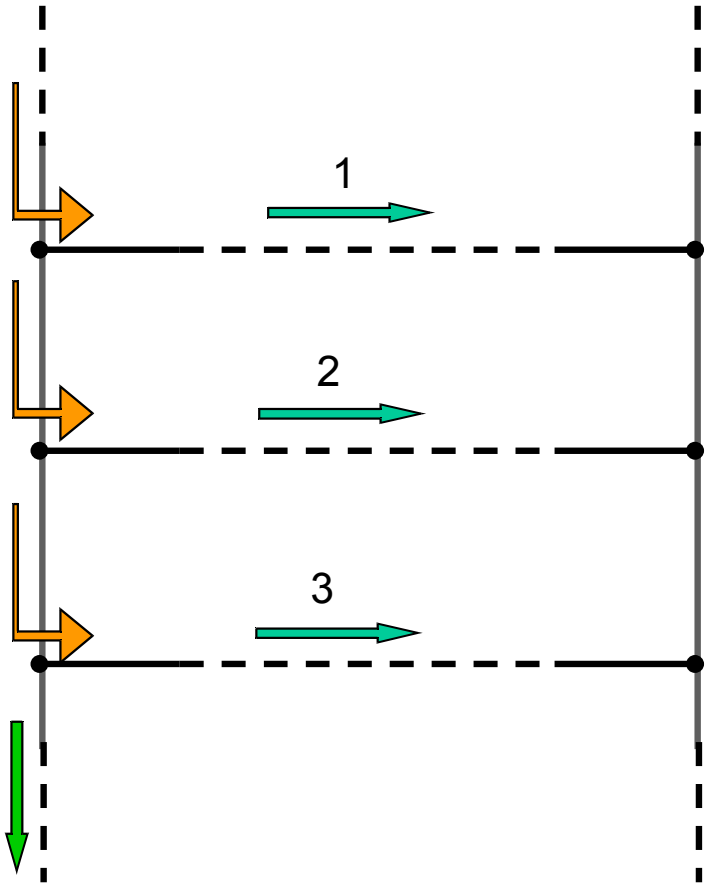
LD can be easily used also by technicians who were used to deal with electromechanical systems, both for their implementation and maintenance, even in the absence of a fully adequate professional update.

LD could be hardly coded (de-coded) when the required (implemented) program is not simple.

The programmer is required to have specific professional skills and experience in order to guarantee the effectiveness and reliability of the program.

Ladder diagram

Basic elements: the ladder



It is the framework within with the program is developed.

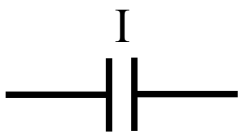
Any logical command is implemented in a ***rung*** (horizontal line) that is red from left to right.

The command sequence is red top-down.

No backward path is allowed.

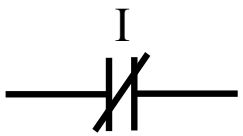
Ladder diagram

Basic elemnts: contacts



(NO)

u=I



(NC)

u=NOT(I)

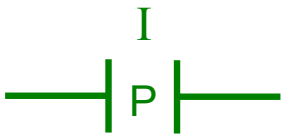
It is the smallest logical element (bit) which define the independent variables of the logical operatore defined by the *rung*

If the associated logical variable is true (I=1) the value of the bit can be true (u=1; normally open contact), or false (u=0; normally closed contact).

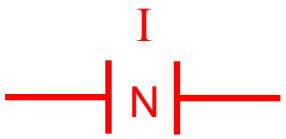
It represents only a logical state of the variable and not the phisycal state of a possibly associated relay.

Ladder diagram

Basic elemnts: contacts



$u_k = (I_k == 1)\ \text{AND}\ (I_{k-1} == 0)$



$u_k = (I_k == 0)\ \text{AND}\ (I_{k-1} == 1)$

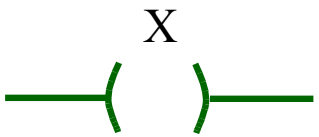
Special types of contacts that represent the transition of the associated variable are available

The rising contact (P) gives to a true variable (u=1) if the associated variable (I) changes from the false to the true state in two subsequent cycles of the routine.

On the contrary the falling contact (N) gives a true vatiable if the transition of the associated variable is from true to false.

Ladder diagram

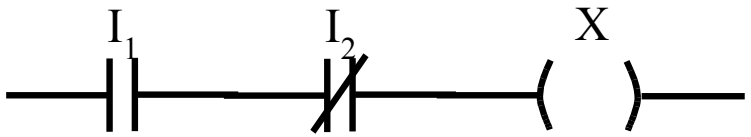
Basic elemnts: coils



X=*rung result*

It represents the booleian variable (X) associated to the result of the logical operator defined by the *rung*

Variable X can be either an internal variable of the program (state) or an ouput variable (command).

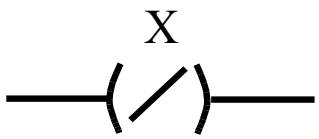


$X=I_1$ AND (NOT(I_2))

The set including all state variable is the state of the LD

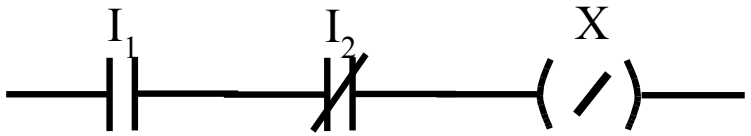
Ladder diagram

Basic elemnts: coils



X=NOT(*rung result)*

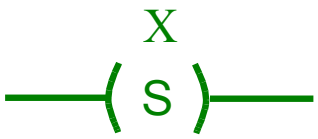
If the result of the logical operation defined by the rung is true, the associated variable X can be true (X=1; simple coil) or false (X=0; denied coil).



X=NOT(I_1 AND (NOT(I_2)))

Ladder diagram

Basic elemnts: coils



Once the variable is set to the true value (X=1) it is memorized

If the rung result is true, the variable associated to the Reset coil is set to zero (X=0)

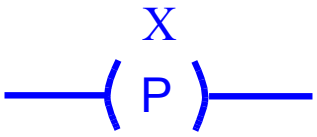
Coils with memory are available such that the value of the associated variable X does not change even if the rung result does.

The ***Set*** (**L**atch) coil puts the value of the associated variable as true (X=1) and it is stored until the correspondant ***Reset*** (**U**nlatch) coil is "powered".

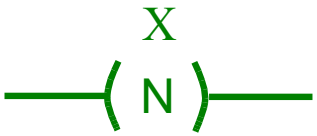
If the power supply of the PLC is out of service all the variables associated to Set coils are set to the false value.

Ladder diagram

Basic elemnts: special coils



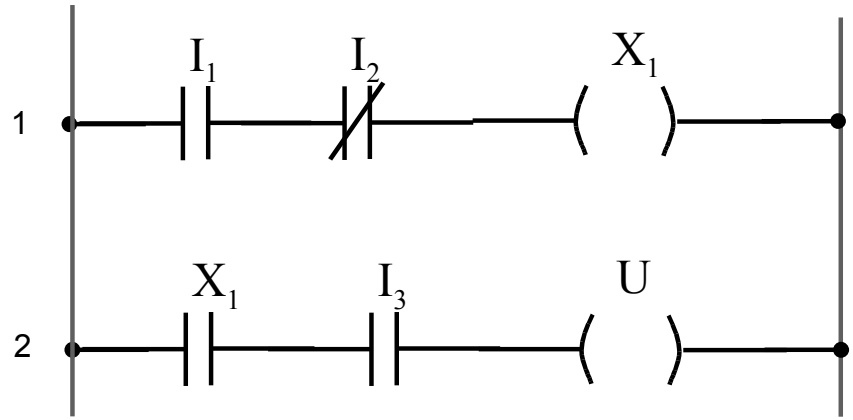
Raising: *X will be true if the rung result passes from false to true within two subsequent cycles*



Falling: *X will be true if the rung result passes from true to false within two subsequent cycles*

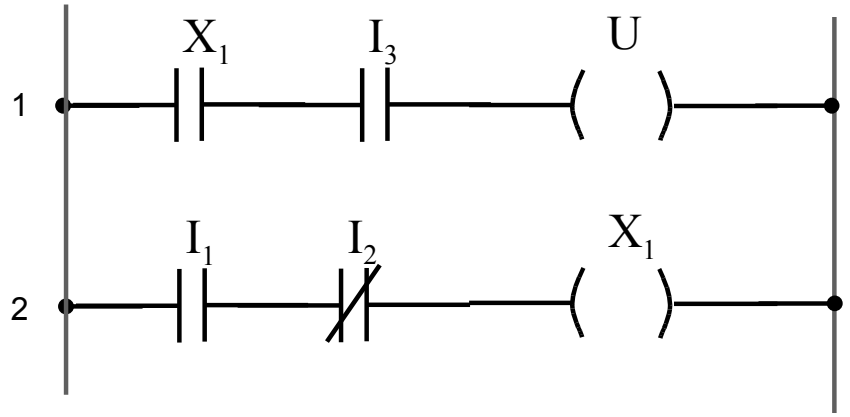
With memory: *the value of the associated variable is memorized in a EEPROM such that it will be maintained even if the power supply of the PLC is down*

Ladder diagram



The *rung* sequence is important

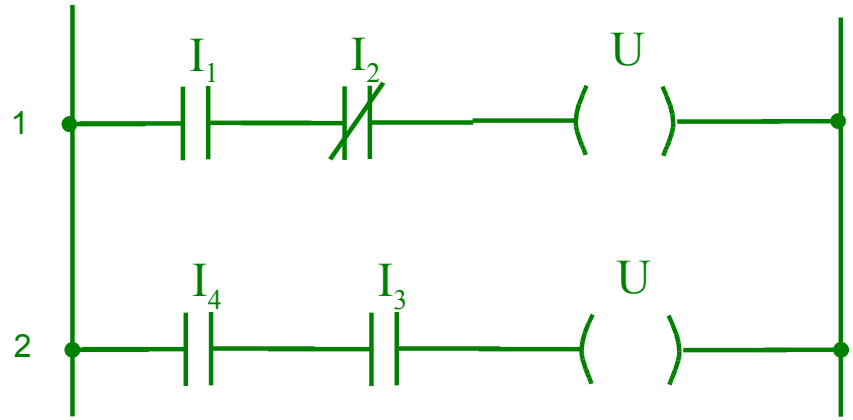
$U = I_1 \text{ AND } (\text{NOT}(I_2)) \text{ AND } I_3$



$U = X_1 \text{ AND } I_3$

If X_1 is not set by another subsequent rung in a previous cycle, X_1 is false and therefore U=0

Ladder diagram



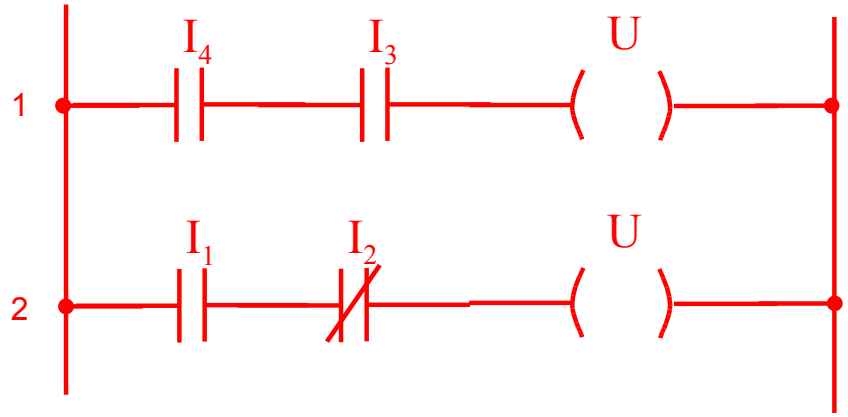
Outputs are assigned only at the end of a program cycle

Input values

$I_1 = 1$
$I_2 = 1$
$I_3 = 1$
$I_4 = 1$

Case a)

$U = I_4$ AND $I_3 = 1$

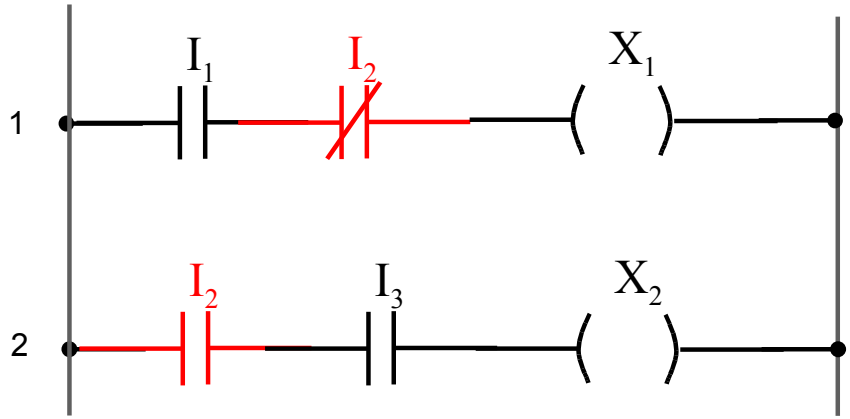


Case b)

$U = I_1$ AND (NOT I_2) = 0

Ladder diagram

Choice

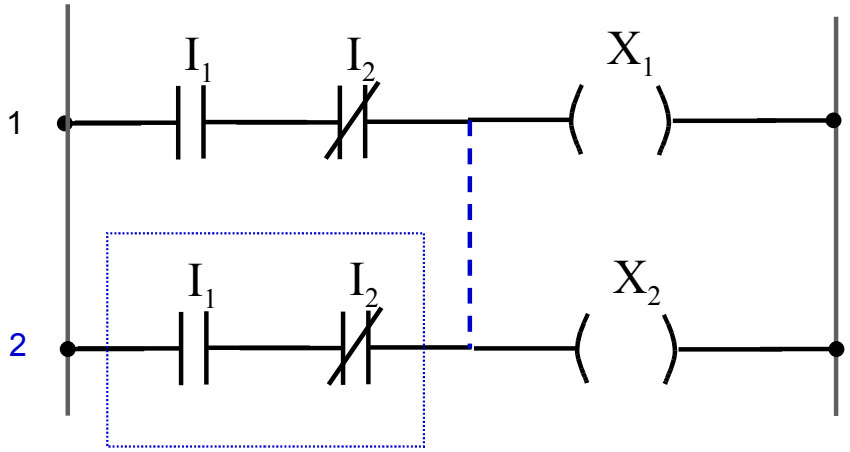


The boleian functions defined by two subsequent rungs must be "mutually excluding each other".

```
IF I2=1 THEN
    IF I3=1 THEN
        X2=1
    ELSE
        X2=0
    END IF
ELSE
    IF I1=1 THEN
        X1=1
    ELSE
        X1=0
    END IF
END IF
```

Ladder diagram

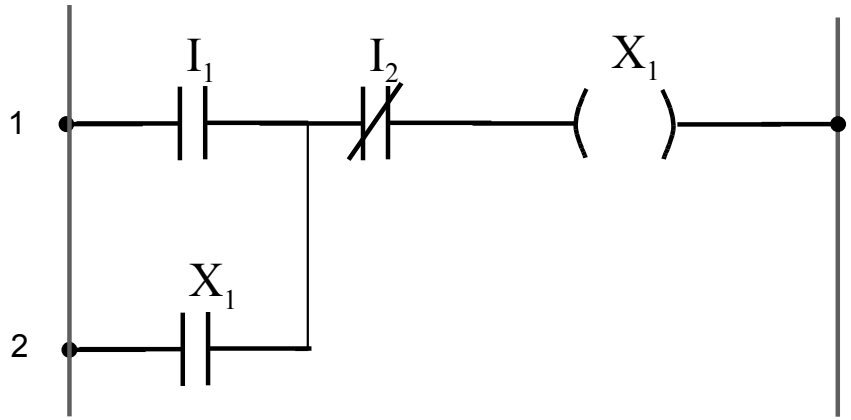
Parallelism



IF I_2=0 AND I_1=1 THEN
    X_2=1
    X_1=1
ELSE
    X_2=0
    X_1=0
END IF

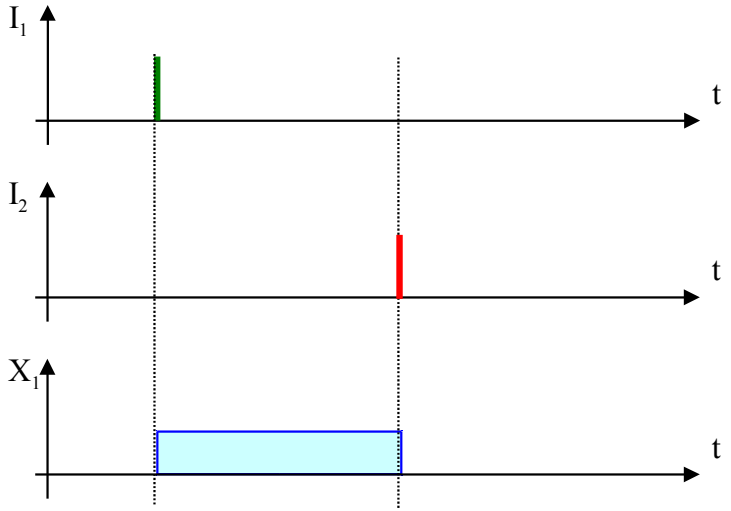
The first part of *rung* 2 can be eliminated

Ladder diagram



Restreint scheme : *variable X_1 is set to the true value as soon as I_1 is true and I_2 is false, and X_1 stay true until I_2 is false once again.*

It is used for START commands



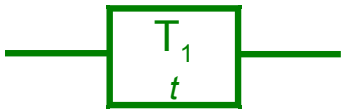
I_1: ON button

I_2: STOP button

X_1: supply coil

Ladder diagram

Additional elements: timers



Timers allow for the timing of the operation sequence and volountary operation delays.

The logic vaiable T_1 becomes true if the logic function defined by the rung remains true continuously for at least the time *t*

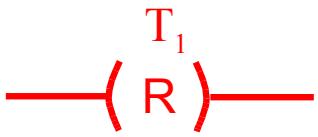
As soon as the logic function in the rung is false the time variable is resetted and the variable T_1 becomes false.

t is ofter expressed using "*cs*" (0.01 seconds)

Ladder diagram

Additional elements: timers

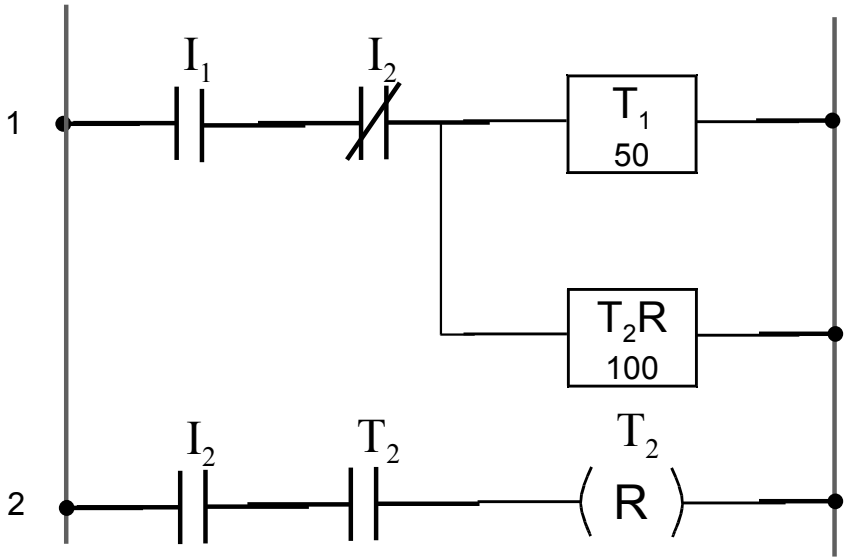
Timers with memory the internal time variable can be resetted only by the corresponding reset coil.

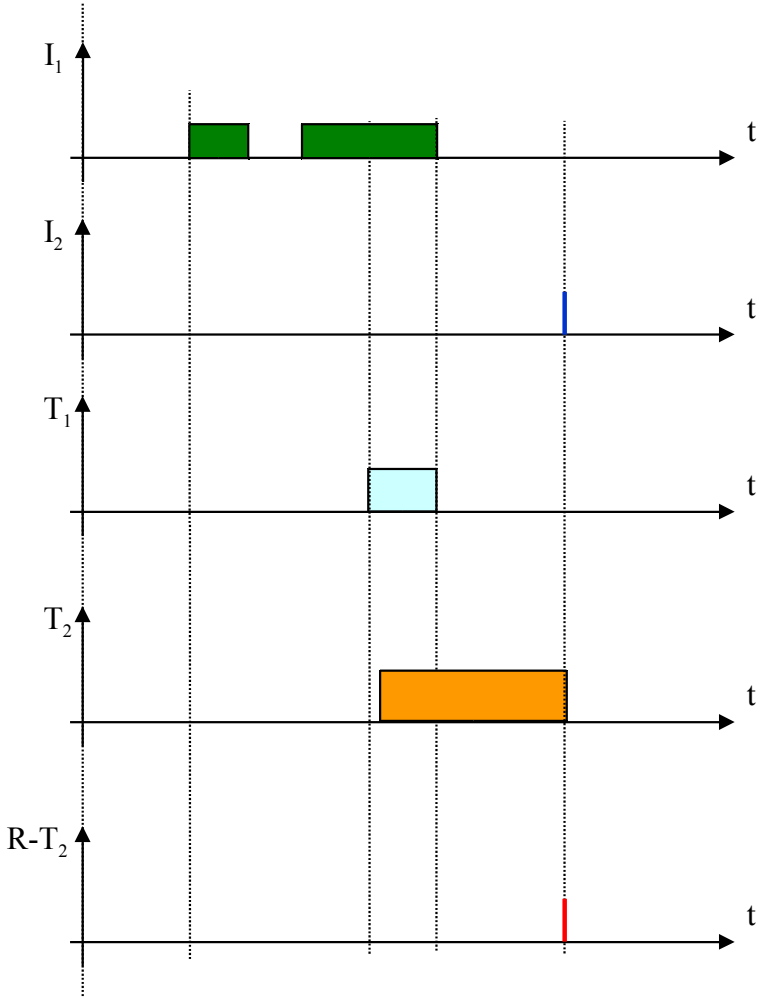


Variable T_1 becomes true if the logical function in the rung is true for at least a time period *t*, **even non continuously**.

Ladder diagram

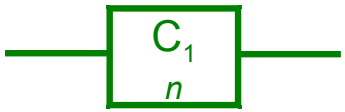
Additional elements: timers





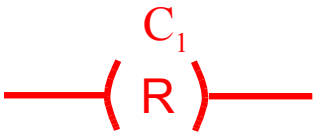
Ladder diagram

Additional elements: incremental counters



Counters allow for implementing the operation taking into account the occurence of a number of equal/similar events.

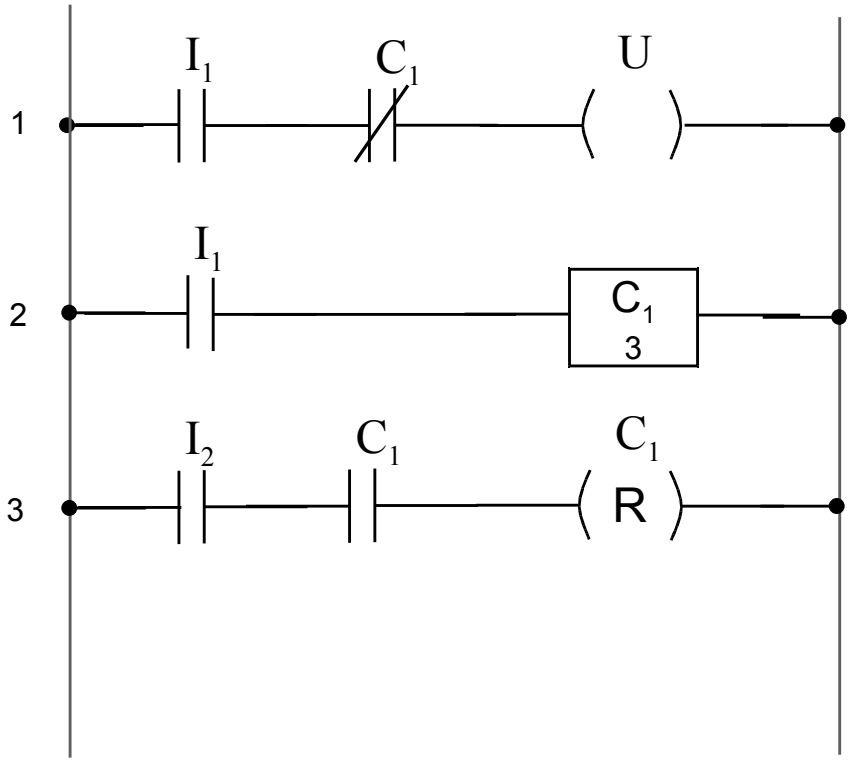
The logical variable C_1 becomes true if the logical function defined by the rung passes from false to true at least *n* times in previous running cycles.

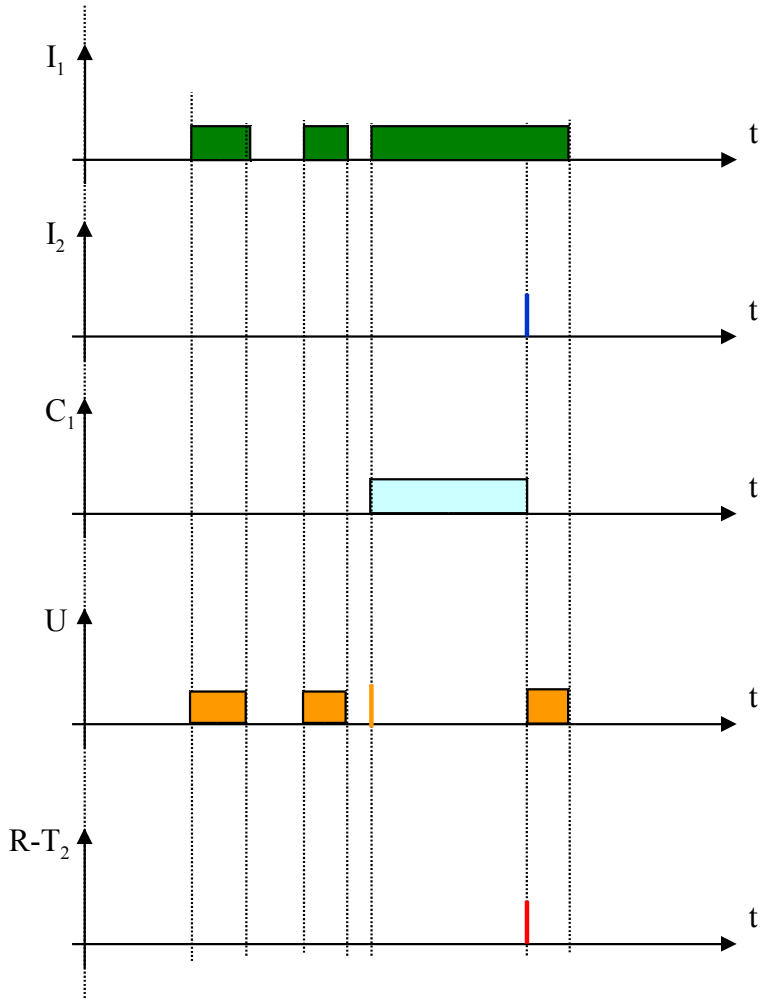


Counters need a reset coil.

Ladder diagram

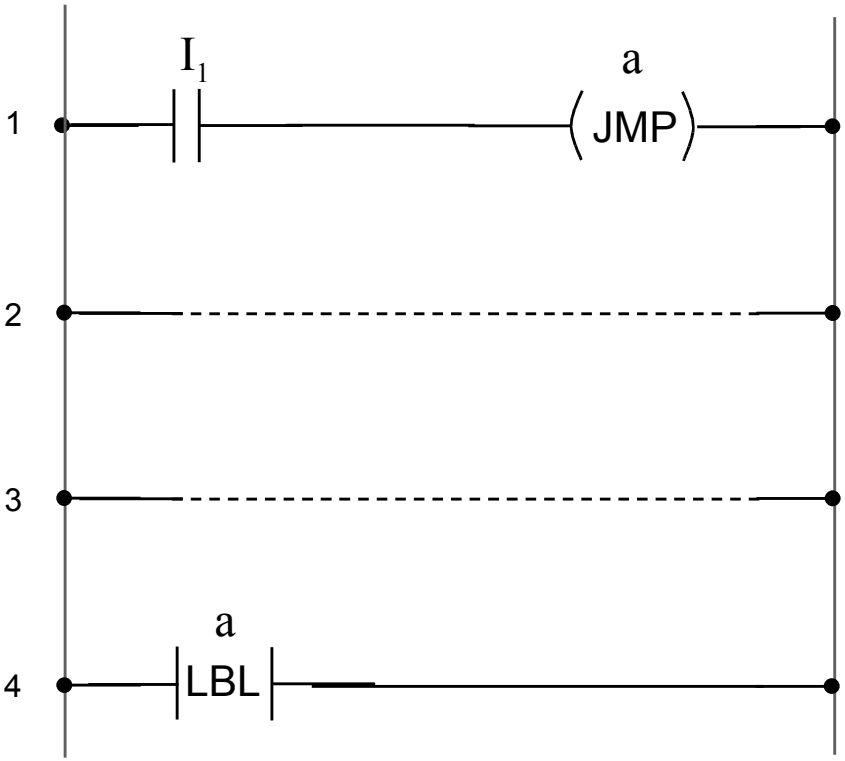
Additional elements: incremental counters





Ladder diagram

Elements for controlling the program sequence: jump

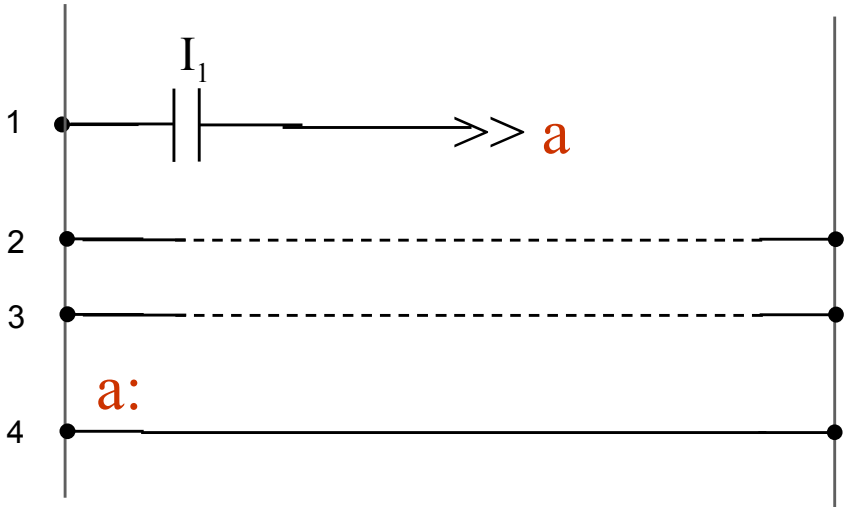


If condion I_1 is true the instructions in rungs 2 e 3 are not executed and the program passes to the rung 5. If condition I_1 is false the instruction after rung 4 are executed after those in rung 2 and 3, respectively.

In simple cases the jump command can be substituted by a choice structure.

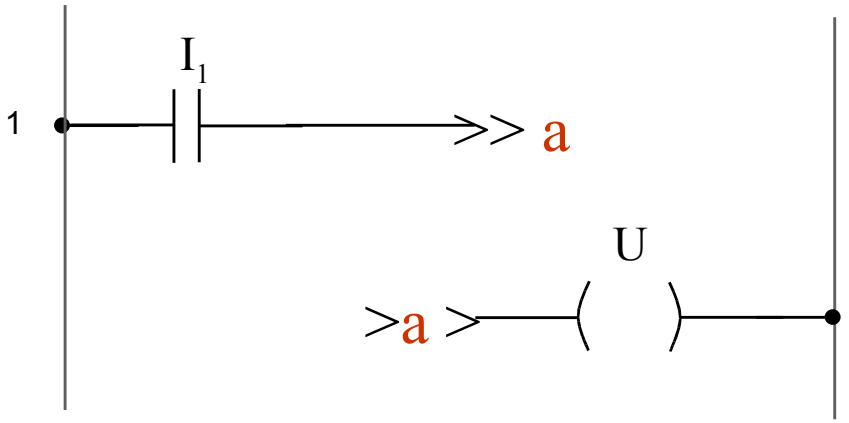
Ladder diagram

Elements for controlling the program sequence: jump



The jump sintax can be dependent on the specific PLC device.

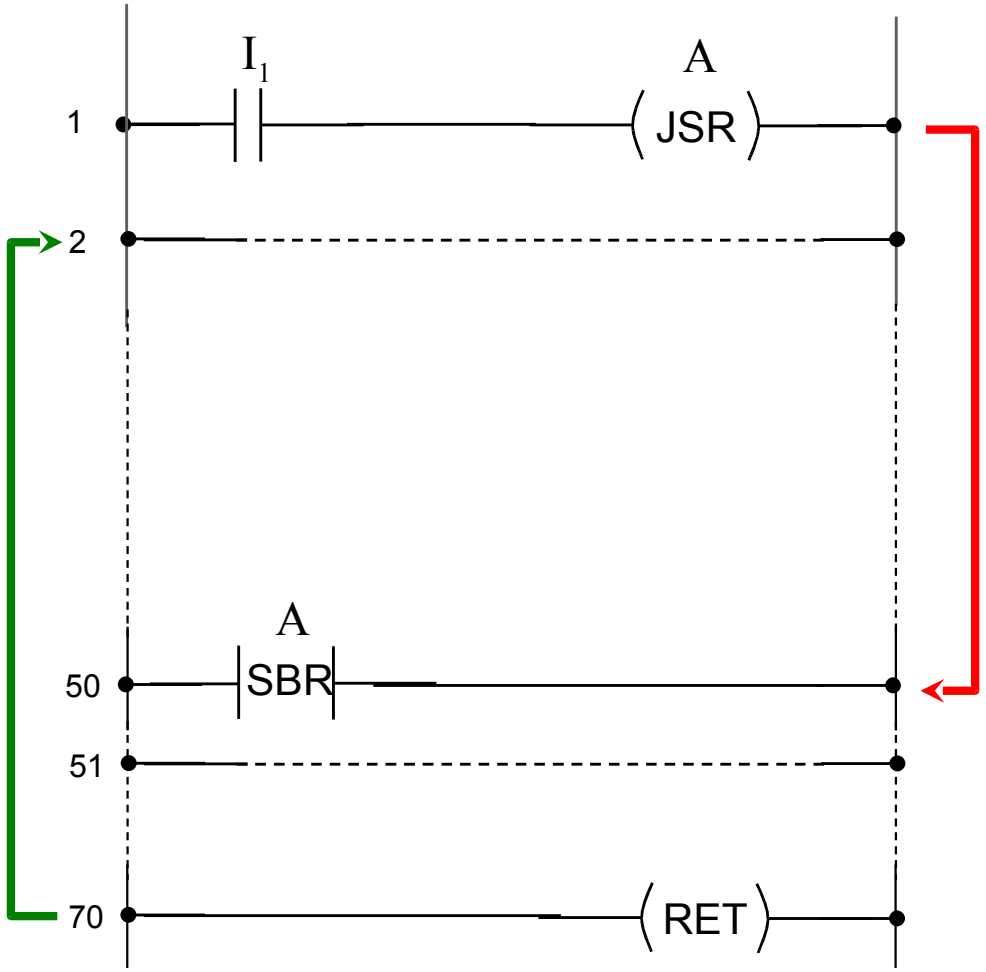
The connection between the jump command and the arriving rung is defined by a proper label.



The jump sintax can be used also to divide a unique logical function into two, or more, lines whenever it is too long to be contained in a unique rung.

Ladder diagram

Elements for controlling the program sequence: jump to subroutine

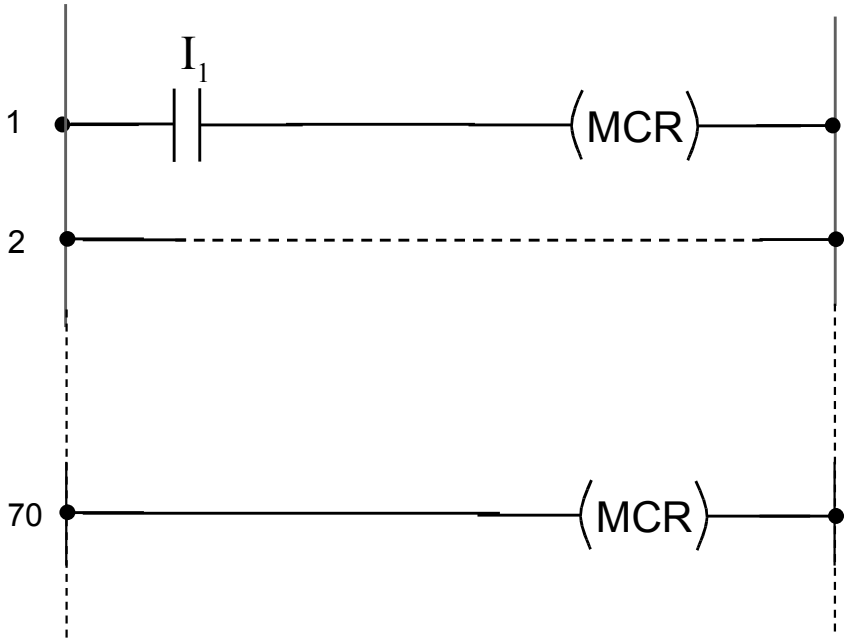


If condition I_1 is true the command within the subroutine (rungs 51-69) are executed and therefore the program continues with rung 2.

The same subroutine can be used in different points of the program.

Ladder diagram

Elements for controlling the program sequence: Master Control Relay



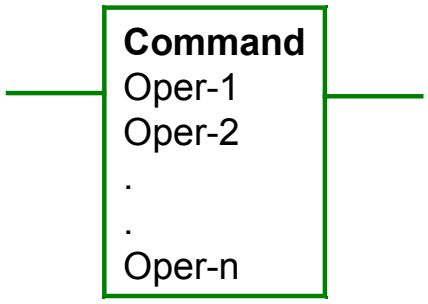
If condition I_1 is true the operations between the two MRC coils are executed.

If condition I_1 is false the operations between the two MRC coils are not executed and the within corresponding coils are set to zero, unless they have memory.

Similar structure has the Zone Control Last State (**ZCL**), but in such a case the insternal coils are maintained in the last state if the ZCL rungs are not executed.

Ladder diagram

Special commands: include logical, algebraic funtions, as well as fuctions of memory and data transfer management.



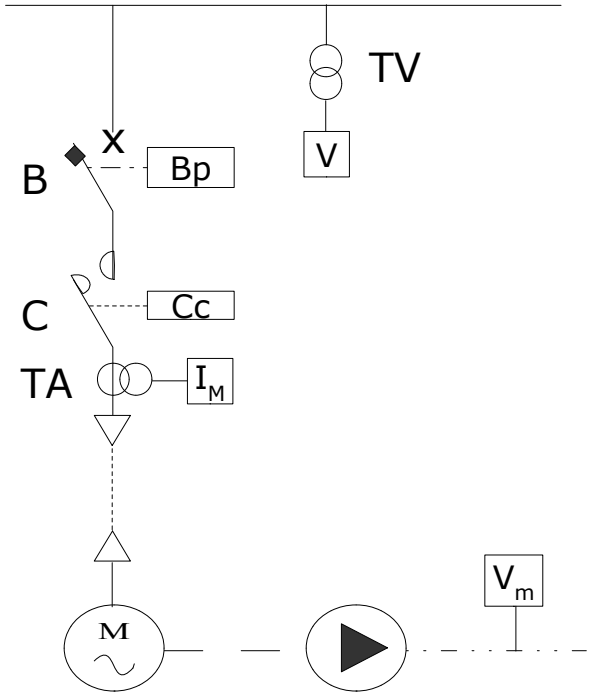
In the example the fuction **Command** is executed using the argoments **Oper-1** ed **Oper-2, …, Oper-n**

Esempi

ADD:	sum
MUL:	product
AND:	binary product bit by bit
OR:	binary sum bit by bit
NEQ:	comparison for inequality
PID:	PID controller
SEND:	data send to a connected PLC
GET:	data aquisition from a connected PLC
RSD:	one bit right shift of a word

Ladder diagram

Exercise: define the ladder program that starts the electro-pumping device feeded by an electrical network. In case the pump doesn't achieve the working speed within 2 seconds for three times the starting procedure should be stopped and the operator action is required



Inputs

$V=1 \rightarrow$ line voltage out-of-range

$I_M=1 \rightarrow$ overcurrent

$V_m=1 \rightarrow$ nominal rotor speed

$B=1 \rightarrow$ breaker closed

$P_{on}=1 \rightarrow$ start command

$P_{off}=1 \rightarrow$ stop command

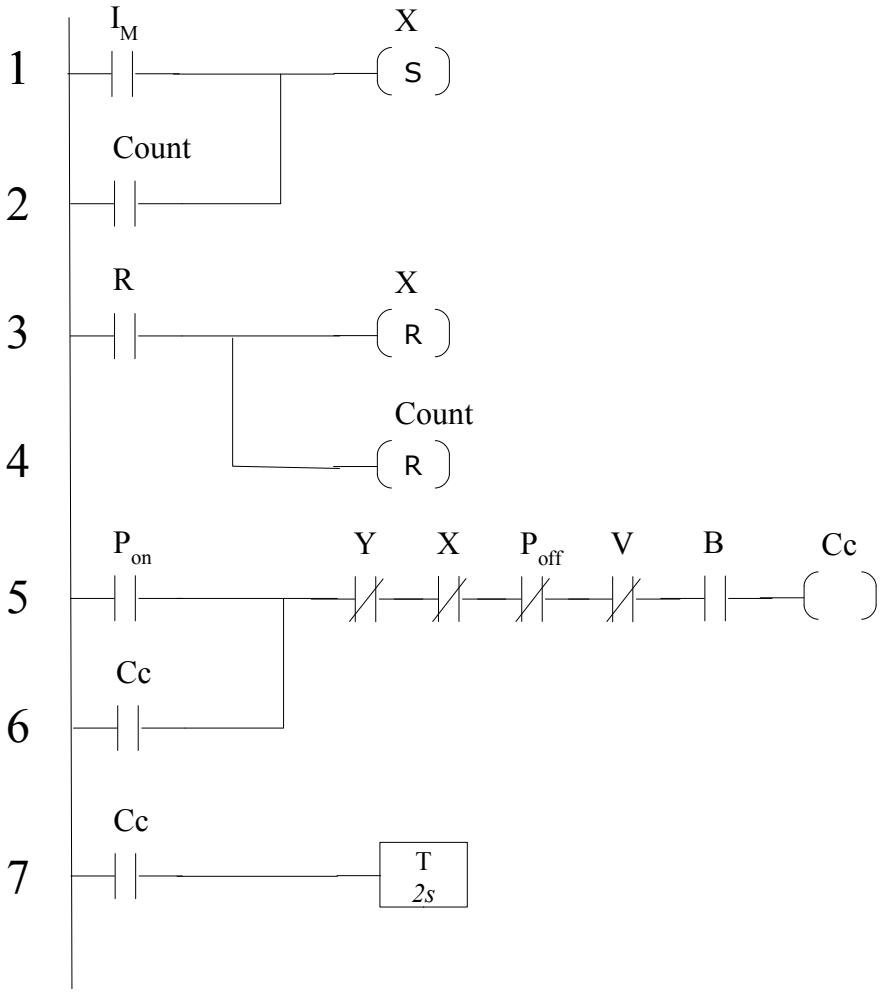
$R=1 \rightarrow$ reset command

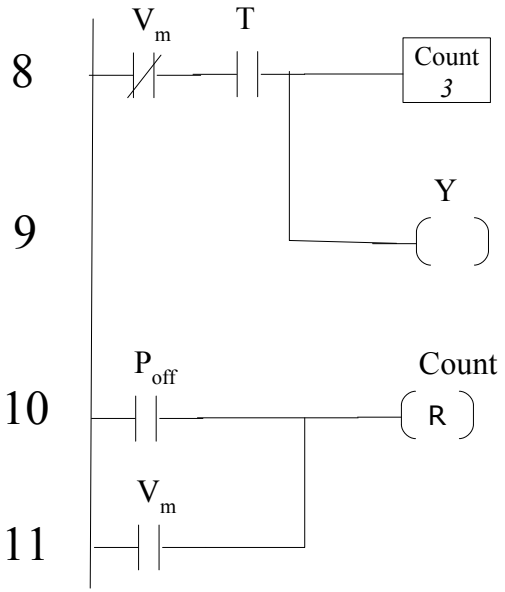
Outputs

$Cc=1 \rightarrow$ contactor close command

Ladder diagram

Exercise: define the ladder program that starts the electro-pumping ……..





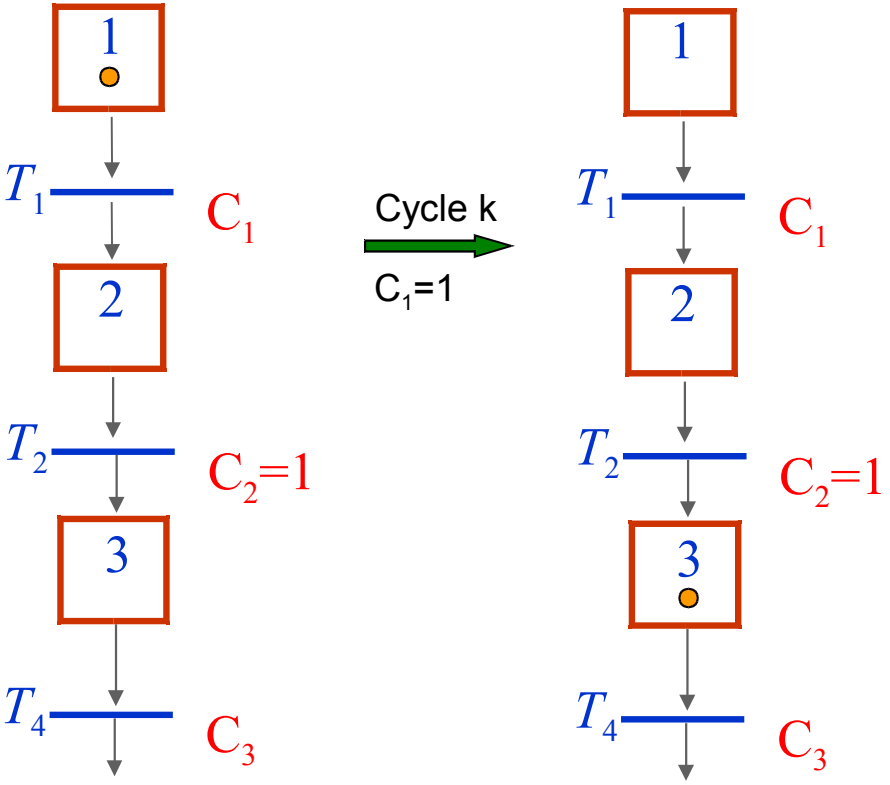
Translation from SFC to LD

A **S**equential **F**unctional **C**hart can be translated into a **L**adder **D**iagram to allow for its implemenation on PLC that doen't support SFC.

- Standard and automatic translation procedure
- Evolution without serch for stability
- Not optimal coding
- Execution time not much increased
- EEPROM use much increased

Traduzione da SFC a LD

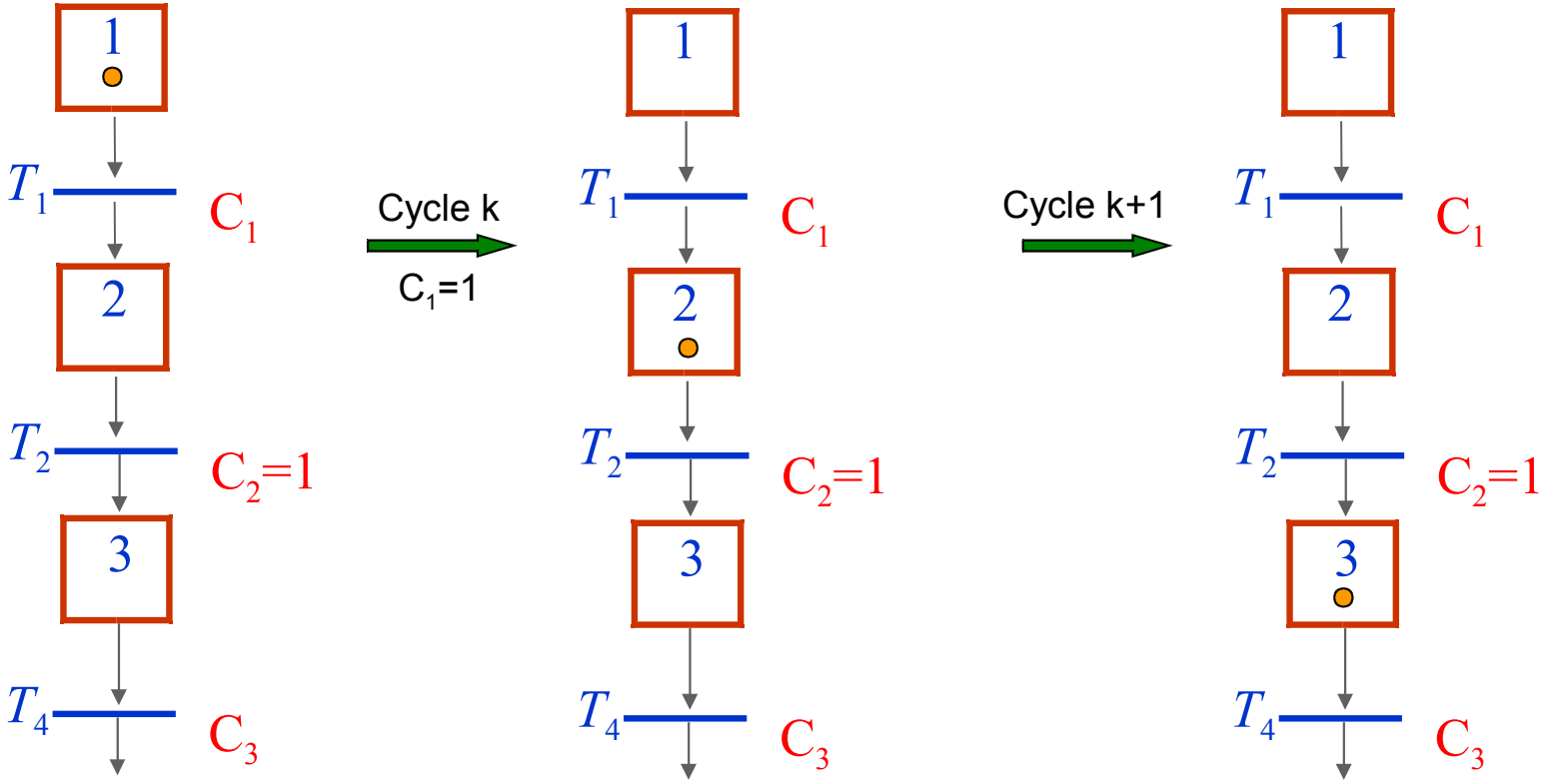
Algorithm with stability search: *all transitions that can be fired in a cycle are fired*



The action eventually associated to phase 2 is not executed

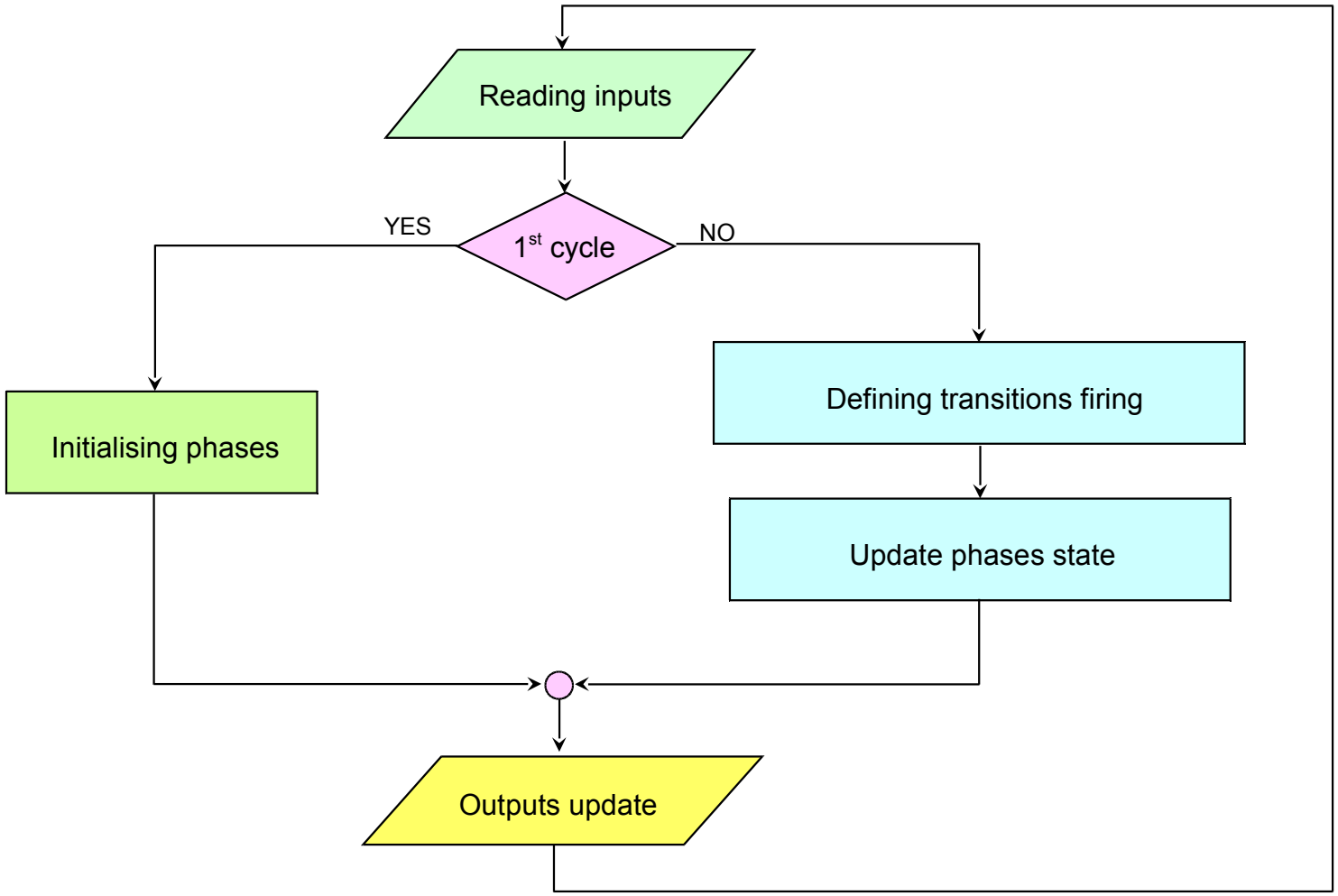
Traduzione da SFC a LD

Algorithm without stability search: *transitions are fired only taking into account the current conditions and phases without update within the cycle.*

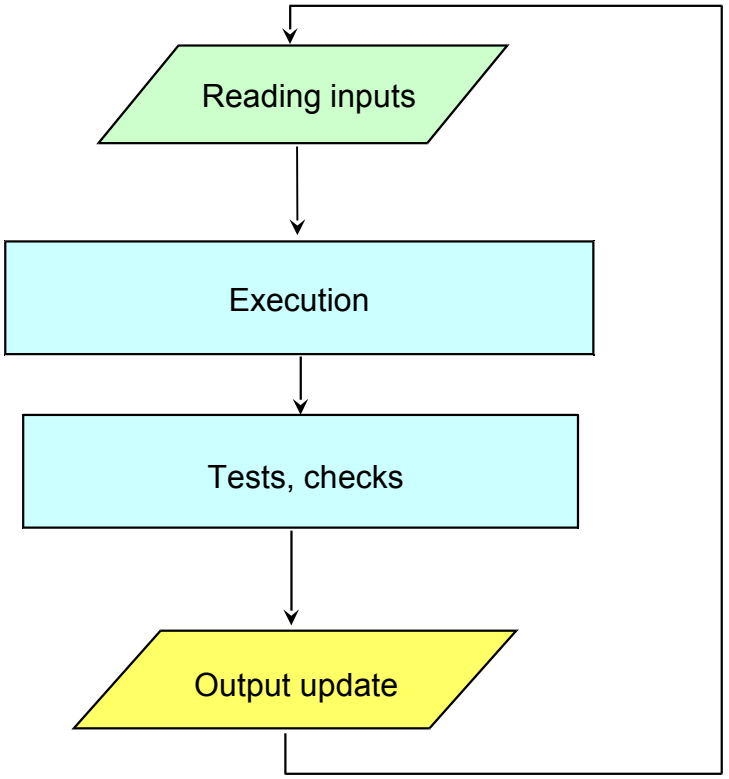


Traduzione da SFC a LD

Algorithm without stability search



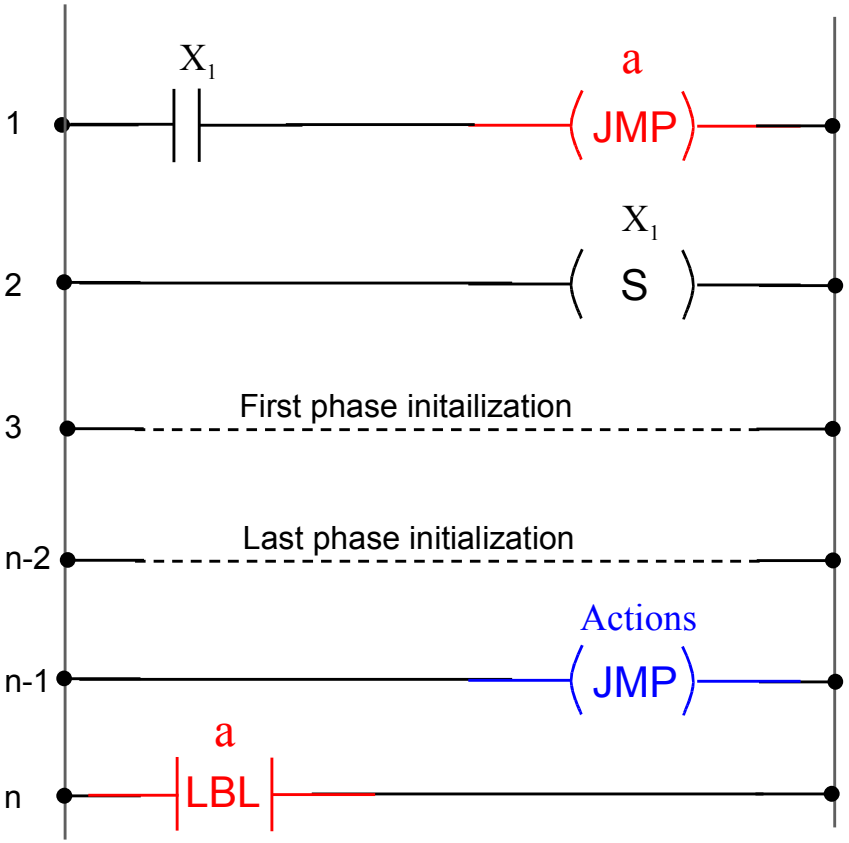
Traduzione da SFC a LD

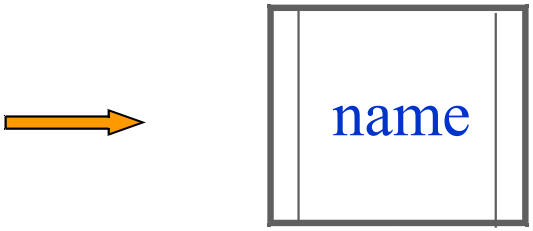


Algorithms without stability search correspond to the working cycle of PLCs

Traduzione da SFC a LD

Initialization: *must be executed only when the PLC is turned on*





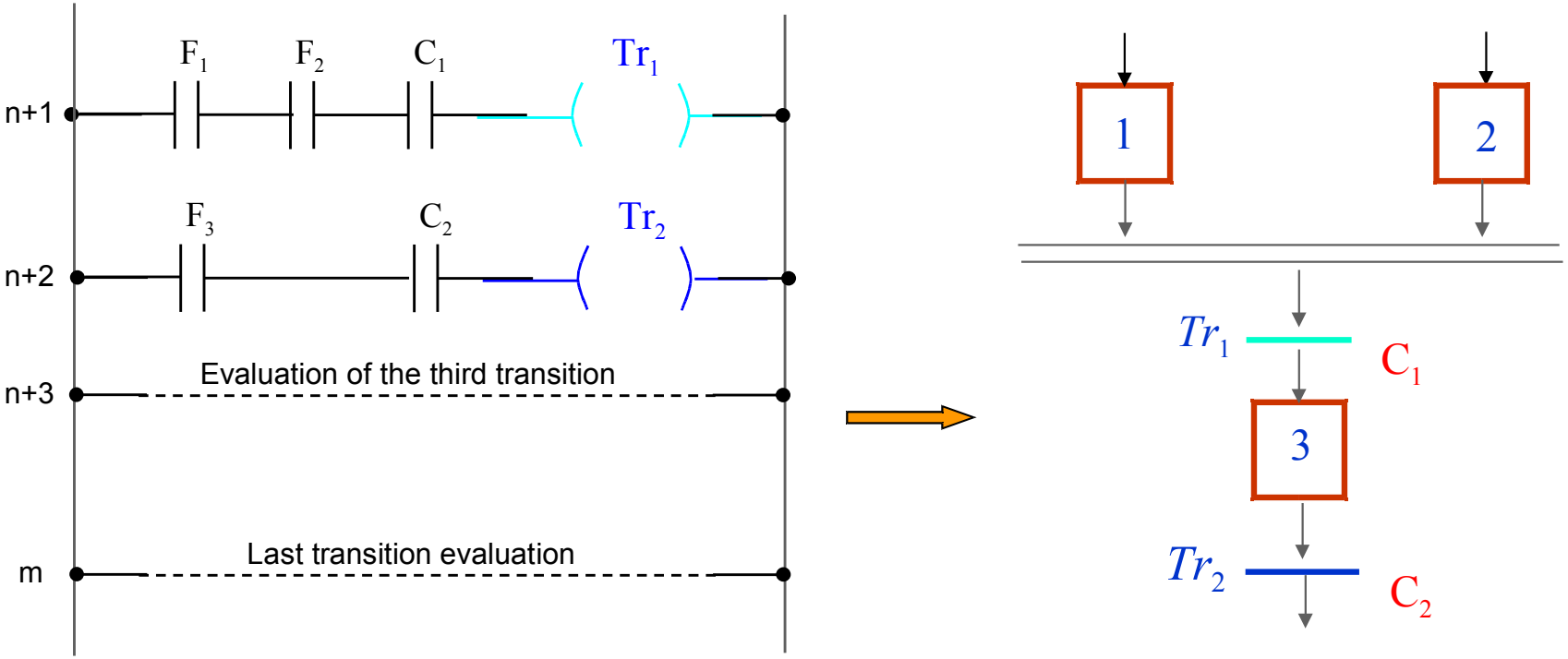
It defines all the phases that must be active at the very beginning of the program

It can correspond to the initial phases

Once the initialization is completed the program jumps to Actions: (outputs update)

Traduzione da SFC a LD

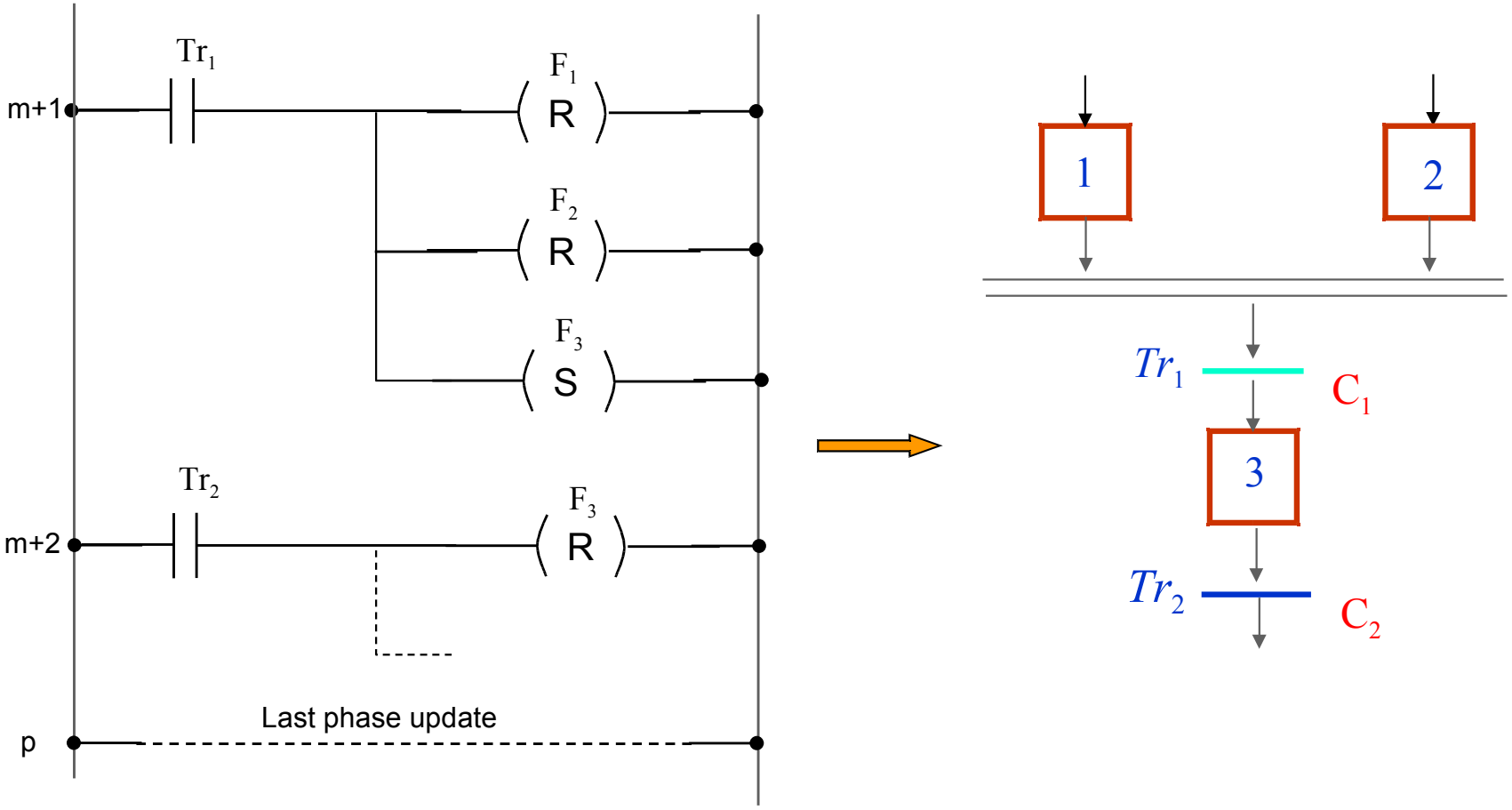
Defining transitions firing



The evaluation of the transitin firing depends on the state of the phases in the previous cycle and on the current conditions with the new inputs

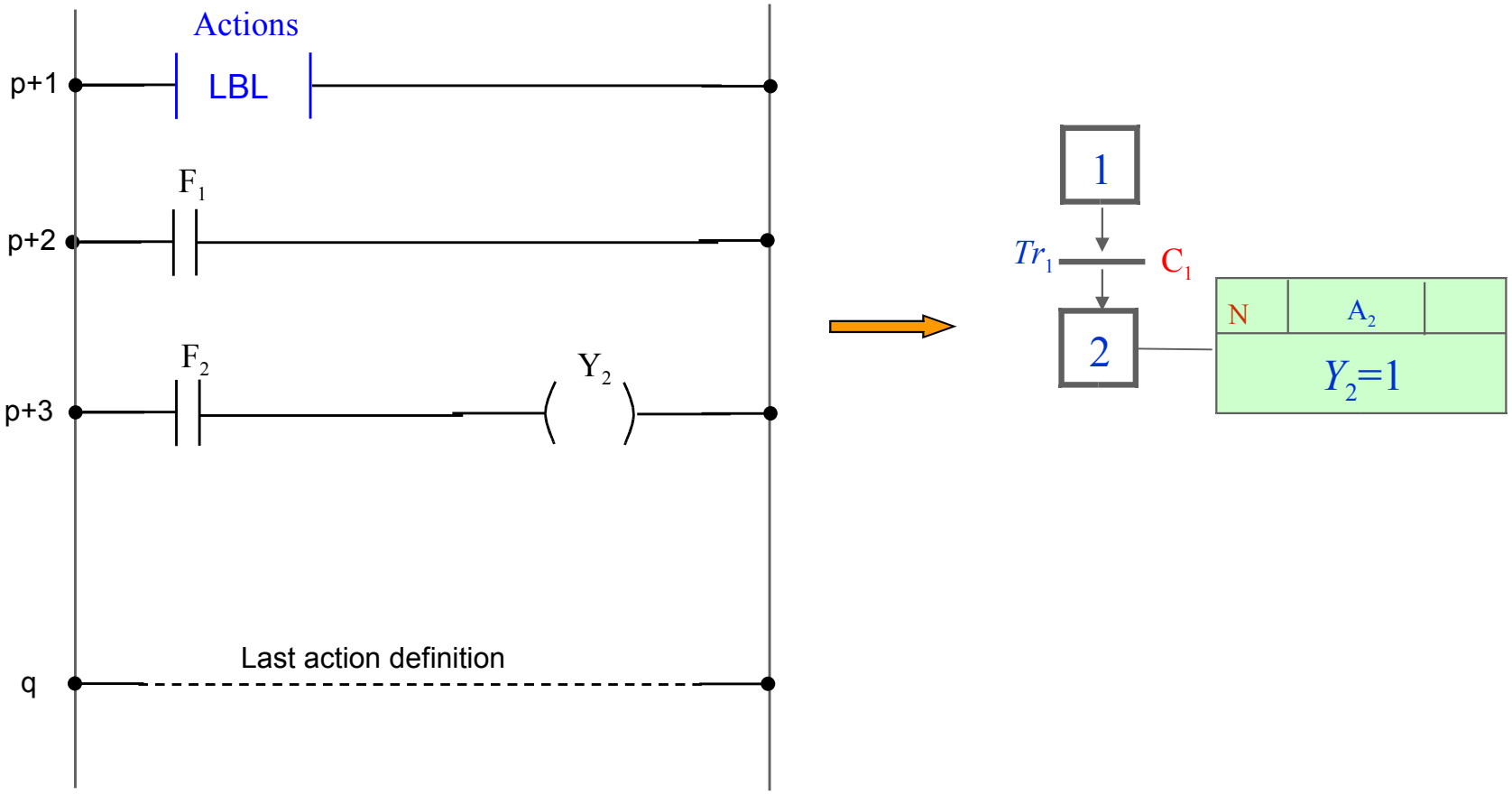
Traduzione da SFC a LD

Update phases state: *the new active phases are determined*



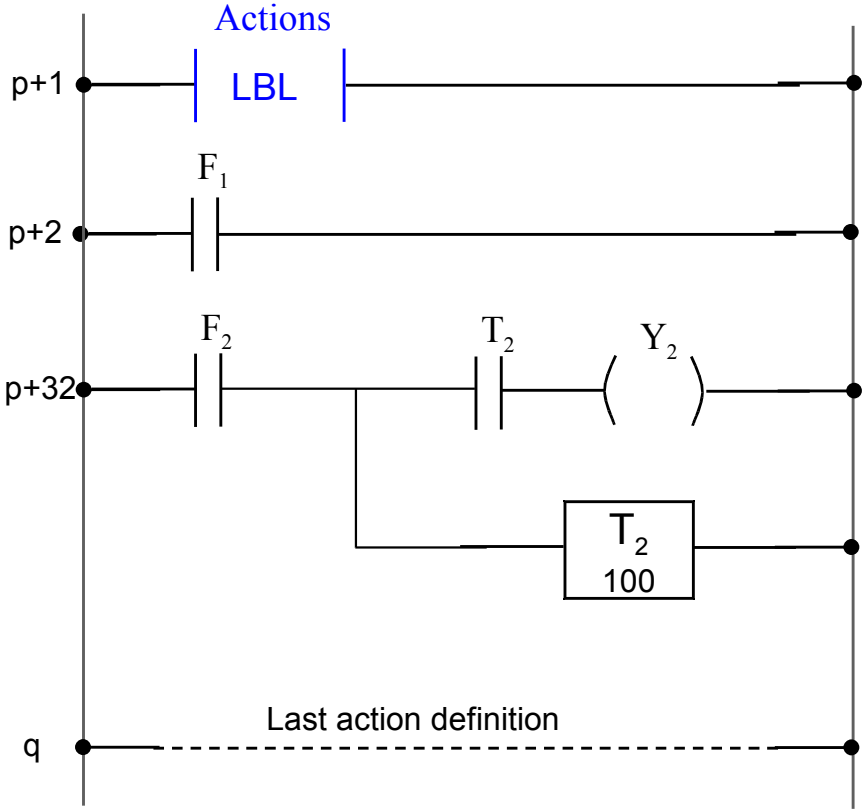
Traduzione da SFC a LD

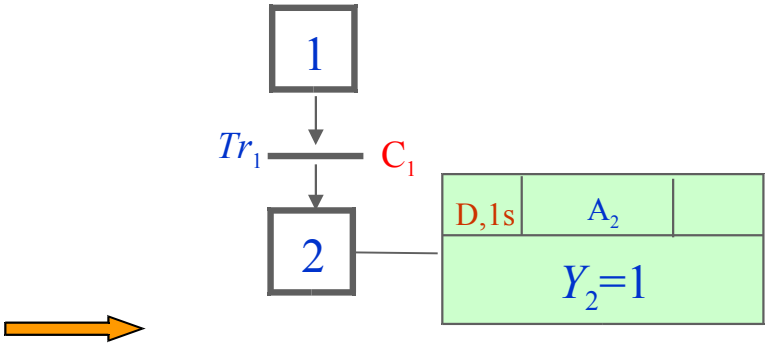
Outputs update: *the new outputs are defined taking into account the new active phases*



Traduzione da SFC a LD

Outputs update: *the new outputs are defined taking into account the new active phases*

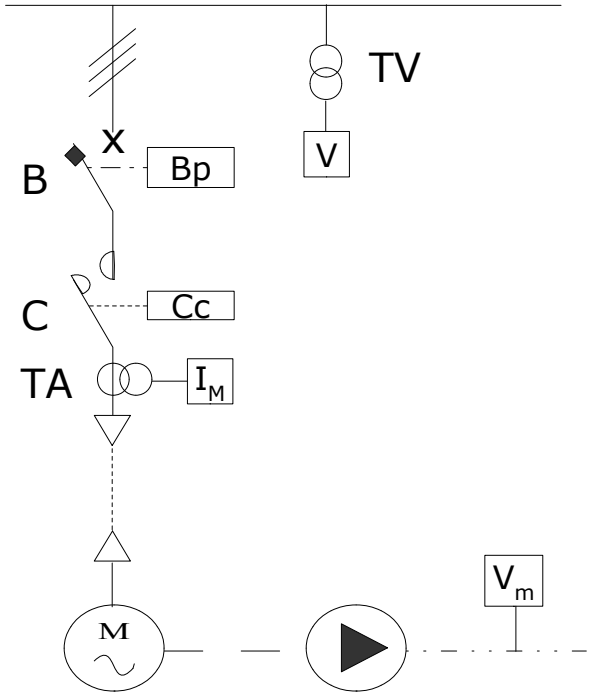




Timers and counters can be used to implement action specifications

Example of a simple project

Project: define the control system that starts the electro-pumping device feeded by three-phase electrical network. In case the pump doesn't achieve the working speed within 2 seconds for three times the starting procedure should be stopped and the operator action is required



PLC logical Inputs

V=1 → line voltage out-of-range

I_M=1 → overcurrent

V_m=1 → nominal rotor speed

B=1 → breaker closed

P_{on}=1 → start command

P_{off}=1 → stop command

R=1 → reset command

Outputs

Cc=1 → contactor close command

Example of a simple project

Project: define the control system …….

The SFC

Fase 0: Motor not available
Fase 1: motor available (***initial***)
Fase 2: motor started
Fase 3: motor's start not completed
Fase 4: motor working

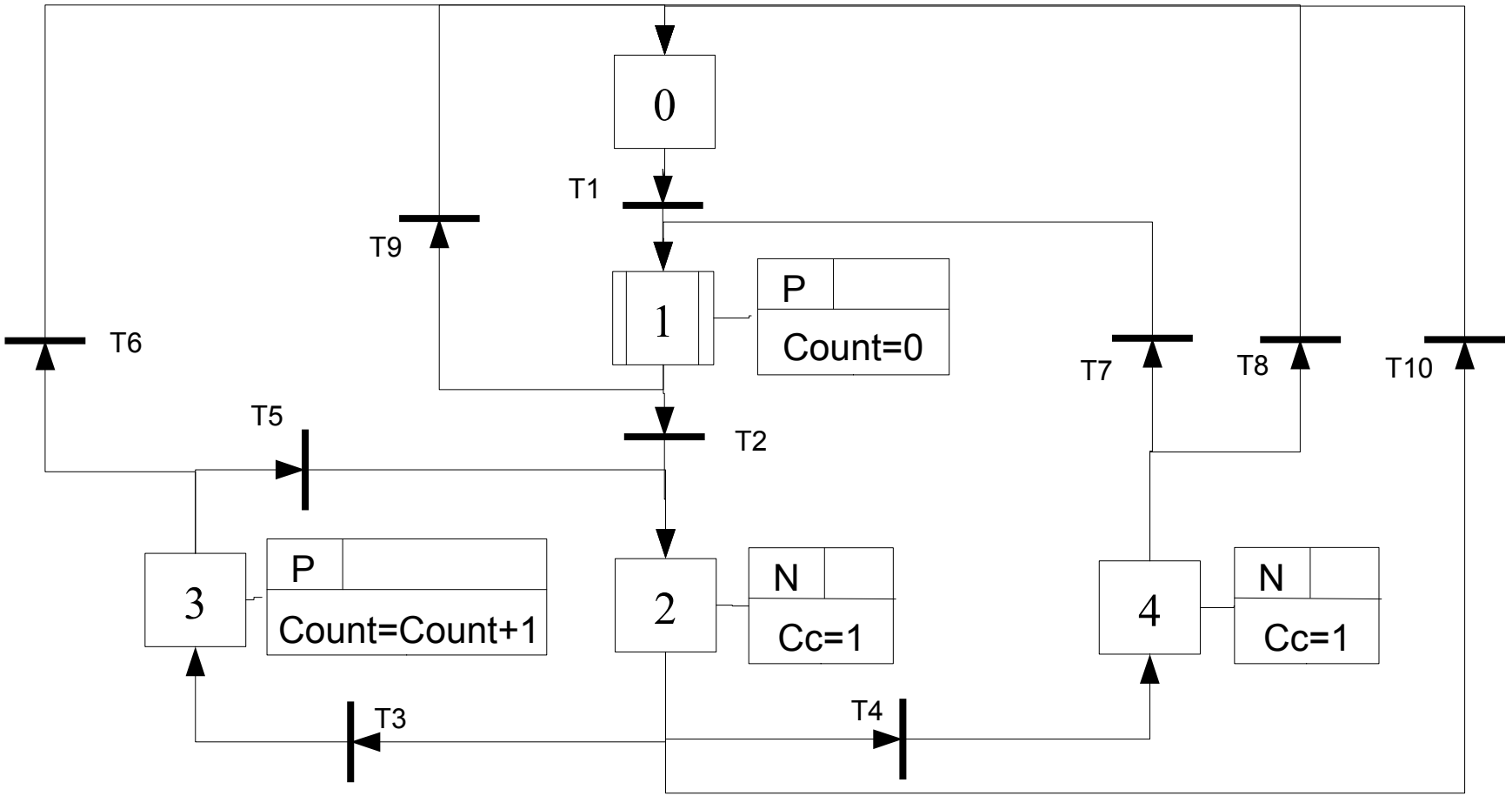
From to ⇨	Fase 0	Fase 1	Fase 2	Fase 3	Fase 4
Fase 0		T1			
Fase 1	T9		T2		
Fase 2	T10			T3	T4
Fase 3	T6		T5		
Fase 4	T8	T7			

Transition	Condition
T1	R
T2	P_{on} *AND* B *AND (NOT* (I_M OR V))
T3	(t_2>2 s) AND (NOT V_m)
T4	V_m
T5	(Count < 3) AND P_{on}
T6	Count ≥ 3
T7	(P_{off} OR V) AND (NOT I_M)
T8	I_M OR (NOT B)
T9	I_M OR (NOT B)
T10	I_M OR (NOT B)

Example of a simple project

Project: define the control system …….

The SFC



Example of a simple project

Project: define the control system …….

The PLC I/O

Analogue Inputs

V_{ab} → phase *ab* RMS voltage

V_{bc} → phase *bc* RMS voltage

V_{ca} → phase *ca* RMS voltage

V_{shaft} → nominal rotor speed

Digital Inputs

I_M → overcurrent relay contact

B → breaker closed contact

P_{on} → start buttom command

P_{off} → stop buttom command

R → reset command

Digital Outputs

Cc → contactor close command

Example of a simple project

Project: define the control system …....

The PLC I/O

Analogue Inputs

V_{ab} → phase *ab* RMS voltage

V_{bc} → phase *bc* RMS voltage

V_{ca} → phase *ca* RMS voltage

N° 3 voltage single phase transducers 0÷660 V ac
PhoenixContact MACX MCR-VAC – 2906239
https://www.phoenixcontact.com

Output 4-20 mA

$V=\text{NOT } (340<V_{ab}<420) \text{ AND } (340<V_{bc}<420) \text{ AND } (340<V_{ca}<420))$

Example of a simple project

Project: define the control system …....

The PLC I/O

Analogue Inputs

V_{ab} → phase *ab* RMS voltage

V_{bc} → phase *bc* RMS voltage

V_{ca} → phase *ca* RMS voltage

N° 1 SIEMENS Multifunctional Transducer
SICAM T 7KG966 code 7KG 9661-2FA00-1AA0
https://support.industry.siemens.com/cs/start?lc=it-IT

Output 4-20 mA

V=NOT (340<V_{avg}<420)

Example of a simple project

Project: define the control system …....

The PLC I/O

Analogue Inputs

$V_{shaft} \rightarrow$ nominal rotor speed

N° 1 ROTEX Elettromeccanica D.C. TACHOGENERATOR
Type SD
Output ±10 V
https://www.rotex-el.com/

$V_m = (V>5)$

Example of a simple project

Project: define the control system …....

The PLC

Solution 1

N° 1 SIEMENS Multifunctional Transducer
SICAM T 7KG966 code 7KG 9661-2FA00-1AA0

N° 1 ROTEX Elettromeccanica D.C. TACHOGENERATOR
Type SD; Output ±10 V

N° 1 SIEMENS SIMATIC S7-1200, CPU 1211C
compact CPU, DC/DC/DC,
Onboard I/O: 6 DI 24 V DC; 4 DO 24 V DC; 2 AI 0-10 V DC,
Power supply: DC 20.4-28.8V DC,
Program/data memory 50 KB

Example of a simple project

Project: define the control system …….

The PLC

Solution 2

N° 3 voltage single phase transducers 0÷660 V ac
PhoenixContact MACX MCR-VAC – 2906239

N° 1 ROTEX Elettromeccanica D.C. TACHOGENERATOR
Type SD; Output ±10 V

N° 1 SIEMENS SIMATIC S7-1200, CPU 1211C
compact CPU, DC/DC/DC,
Onboard I/O: 6 DI 24 V DC; 4 DO 24 V DC; 2 AI 0-10 V DC,
Power supply: DC 20.4-28.8V DC, Program/data memory 50 KB

N° 1 SIEMENS SIMATIC S7-1200, Analog input, SM 1231
4 AI, +/-10 V, +/-5 V, +/- 2.5 V, +/-1.25 or 0-20 mA/4-20 mA,
15 bit+sign bit