# IEC 61131-3 Programming Languages

## Lecture 3

# Learning Outcome

- Explain the importance of the IEC standard for PLC
- Describe the different programming languages provided under IEC 61131-3 for PLC programming
- Declaration of variables including Local & Global with correct data types
- Explain the differences between different modes of PLC addressing
- Design SFC for sequential control requirement

# Importance of IEC 61131

There are numerous manufacturers of PLCs, IEC 61131 provides a standard for all manufacturers to follow.

It assure users of the suitability for industrial applications.

The most significant benefit reaped is with the standardisation of programming languages in IEC 61131-3:

- Ease of understanding works from other PLC programmers
- Independent of PLC brands
  - Reduction of training time
  - Buying and selling products and services easily
- Easier reuse of program code

# What is IEC 61131?

It is an IEC standard for PLC that is sub-divided into 10 parts:

| Standard | Details of Standard |
|---|---|
| IEC 61131 – 1 | General Overview<br>Define and identify the principal functional characteristics of PLC and associated peripherals |
| IEC 61131 – 2 | Equipment Requirements & Tests<br>Specify the equipment requirements and related tests for programmable controllers and associated peripherals for industrial controls such as PLC, industrial PC, HMI, DCS |
| IEC 61131 – 3 | Programming Languages<br>Standardize 5 different programming languages for PLC |
| IEC 61131 – 4 | User Guidelines<br>Facilitate communication between PLC user and PLC supplier according to specifications of IEC 61131 |
| IEC 61131 – 5 | Communication<br>Specify the communication aspect of programmable controllers with other electronic systems and PCs |

Source: https://plcopen.org/technical-activities/logic

# What is IEC 61131?

It is an IEC standard for PLC that is sub-divided into 10 parts:

| Standard | Details of Standard |
|---|---|
| IEC 61131 – 6 | Functional Safety<br>Specify as part 1, the intended to be used for safety related systems. It should be read concurrently with other standards such as IEC 61508 |
| IEC 61131 – 7 | Fuzzy Control Programming<br>Defines basic programming elements for fuzzy logic control as used in programmable controllers to integrate fuzzy control applications according to part 3. |
| IEC 61131 – 8 | Guidelines for the application and implementation of programming languages for software developers |
| IEC 61131 – 9 | Single-drop digital communication interface for small sensors and actuators (IO-Link) |
| IEC 61131 – 10 | PLC open XML Exchange Format for export and import of IEC 61131-3 projects to be transferred between different programming environments. Allows exchange of configuration elements, data types and POUs |

Source: https://plcopen.org/technical-activities/logic

IEC 61131-3

# IEC 61131-3 Standard

Programming Languages

Common Elements

# IEC 61131 – 3 Programming Languages

The 5 programming languages as follows

1. Ladder Diagram (LD) - Graphical
2. Instruction List (IL) - Text
3. Function Block Diagram (FBD) - Graphical
4. Sequential Function Chart (SFC) - Graphical
5. Structured Text (ST) - Text

# IEC 61131 - 3

In addition to definition of languages, the standard also covers the important aspects:

- Addressing
- Execution
- Data formats/data structures
- Use of symbols
- Connections between languages

# IEC 61131 – 3

## Common Elements
## Data Typing

Data Typing
Variables

- Definition of Data Type of any parameter used
- Prevent errors in early stage
- Common data types:
  - Boolean
  - Integer
  - Real
  - Byte
  - Word
  - Date
  - Time_of_Day
  - String
- Personal data types could be defined and re-use

# IEC 61131 – 3

# Common Elements Variables

Data Typing Variables

- **Variables**
  - Serve as identification of data objects
  - Efficient way to refer to data object
  - Connected to Inputs, Outputs or Memory of PLC
- **Local Variable**
  - Contained within program block only, hence name can be reused in other parts without conflict
  - Declared in the program block
- **Global Variable**
  - The variable could be addressed or used by another configuration or another program block
  - Unique variable name
  - Assigned to specific physical input / output or PLC memory
  - Parameters able to be assigned with initial value during PLC/CPU start up

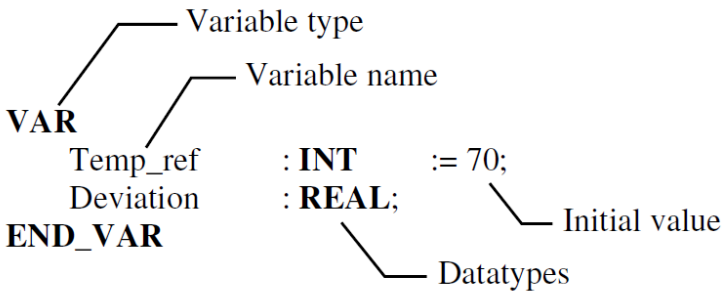
# IEC 61131 – 3

# Common Elements Variables Addressing

- When Global Variables are declared, Data Type and address needs to be specified
- Address shall be % Prefix followed by the address
  - Example: %I0.1 %IX0.1 for Input Channel 0, bit 1

| % | 1st prefix | 2nd prefix | Specific location | Meaning | |
|---|---|---|---|---|---|
| | **I** | | | Input | |
| | **Q** | | | Output | |
| | **M** | | | Memory | |
| | | **None or X** | | Boolean | : 1 bit |
| | | **B** | | Byte | : 8 bit |
| | | **W** | | Single word | : 16 bit |
| | | **D** | | Double word | : 32 bit |
| | | **L** | | Long word | : 64 bit |

| Address | Meaning |
|---|---|
| %MX0.0 | Bit 0 (LSB) in memory location 0 |
| %M0.0 | ----------"---------- |
| %MB8 | Memory byte 8 |
| %MW12 | Memory word 12 |
| %MD45 | Double word at memory location 45 |
| %ML14 | Quadruple word at memory location 14 |

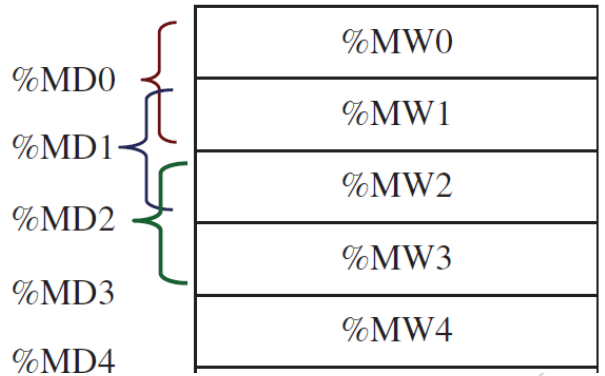


# IEC 61131 – 3

# Common Elements
## Direct Addressing

- Even when the standard introduce variables and recommend using variables. It is possible to use direct addressing
- Direct addressing simply means entry of address directly to the program
- This could be direct addressing of input / output address or directly addressing the memory location
- It is the same way as assigning address to global variables
- Direct addressing of memory location is not recommended as there could be risk of using address that overlap with another function or use


https://youtu.be/mnAhfEFIOsE



# Implementation of IEC 61131-3 Standard

- It is NOT an absolute standard for all manufacturers to follow.
- Instead, the standard define a comprehensive set of guidelines.
- Manufacturers decide the extend to follow the guidelines and document the conformities clearly.

In this way, programming PLC of different brands would largely be the same, with exception of software user interface, graphics and other competitive features could still be implemented to differentiate themselves.

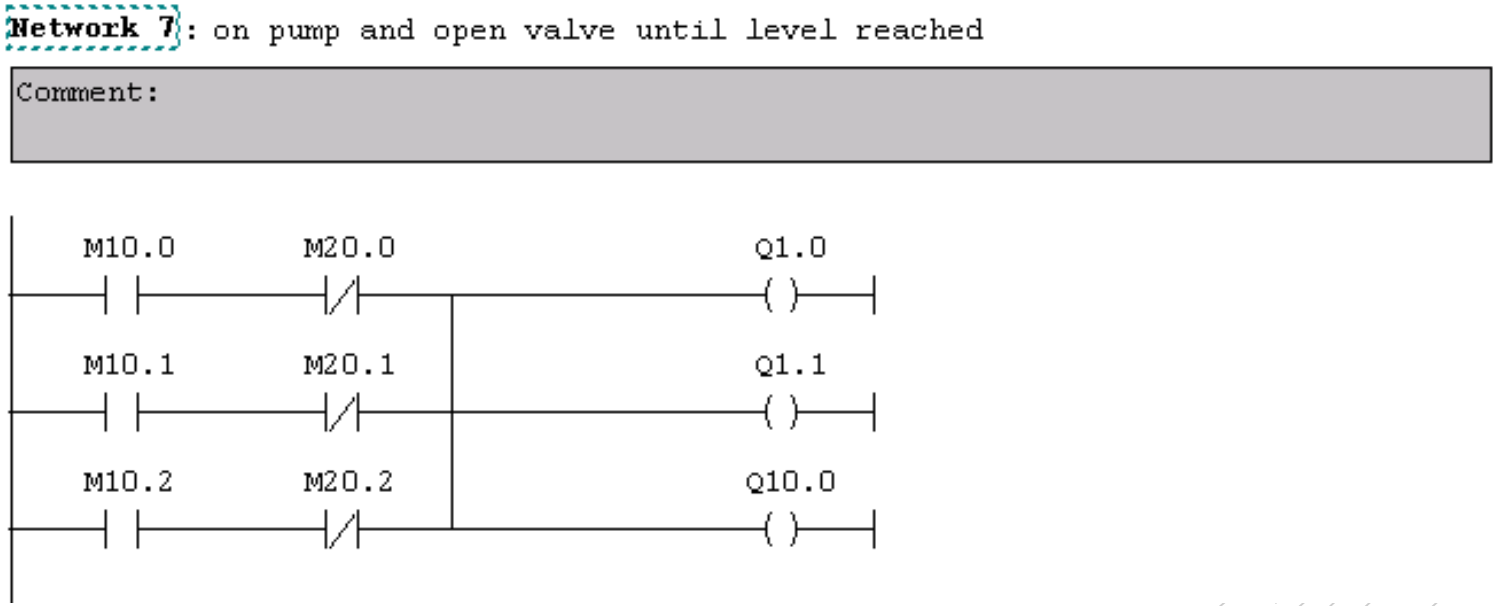
# IEC 61131 – 3 Programming Languages

The 5 programming languages as follows

1. Ladder Diagram (LD) - Graphical
2. Instruction List (IL) - Text
3. Function Block Diagram (FBD) - Graphical
4. Sequential Function Chart (SFC) - Graphical
5. Structured Text (ST) - Text

# Ladder (LD)

- The most commonly language used in PLC as it resembles relay logic that PLC was designed to replace.
- Very suitable for logic operations and interlocking functions
- Most PLC programs are written in LD



# Instruction List (IL)


- Instruction List (IL) is an assembly-like low-level language
- The below instruction performs the same function as the ladder diagram (LD)
- IL was popular when handheld devices were used for programming PLC

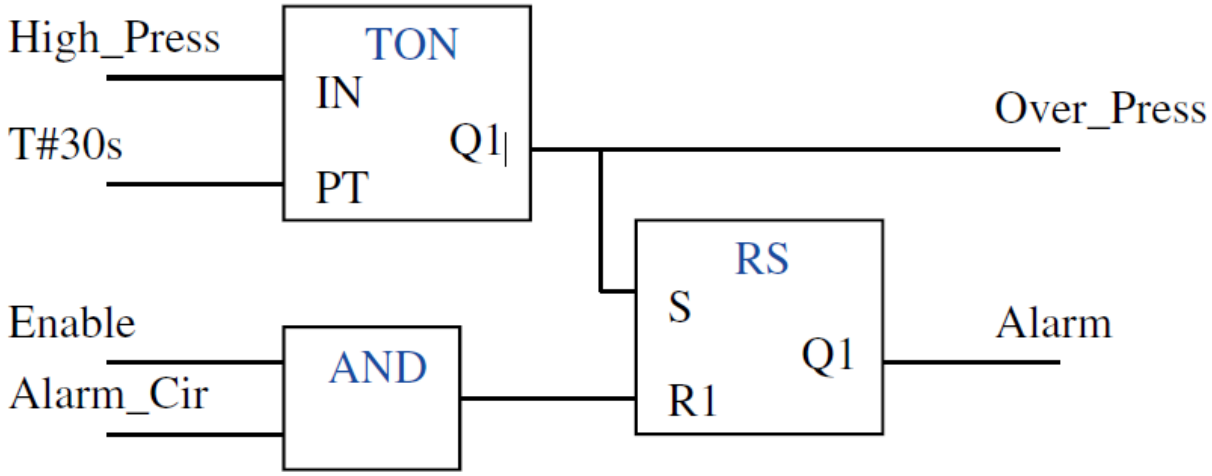
```
Network 7: on pump and open valve until level reached
Comment:

      A     M     10.0
      AN    M     20.0
      O
      A     M     10.1
      AN    M     20.1
      O
      A     M     10.2
      AN    M     20.2
      =     Q      1.0
      =     Q      1.1
      =     Q     10.0
```

# Function Block Diagram (FBD)

FBD is programming language. Do not get mixed up with custom function-blocks!

- Graphical language that is based on connecting functions and function-blocks
- Standard logic functions such as AND, OR, NOT etc and function-blocks such as timers, Set-Reset etc can easily be linked up
- Easy for programmers with digital electronics knowledge
- Practical to implement logical Boolean function with a quick overview of functions



# Structure Text (ST)

- Text based high-level language, resembles Pascal or C
- Very suitable for complex arithmetic functions, manipulation of bits and Words
- Selection branches such as IF-THEN-ELSE and CASE OF
- Iteration loops such as FOR, WHILE, REPEAT
- Ideal for creating custom function-blocks that could be reused

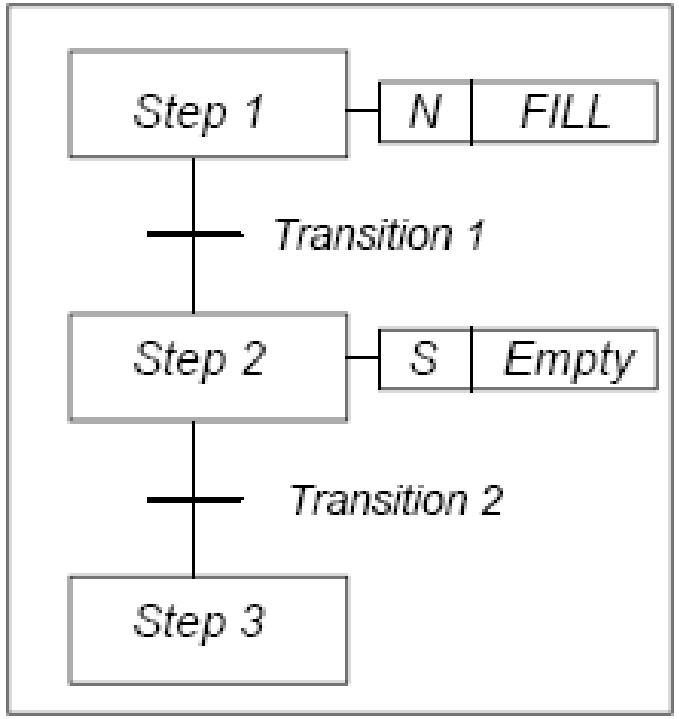
```
I:=25;
WHILE J<5 DO
        Z:= F(I+J);
END_WHILE

IF B_1 THEN
        %QW100:= INT_TO_BCD(Display)
ENDIF

CASE TW OF
        1,5:    TEMP := TEMP_1;
        2:      TEMP := 40;
        4:      TEMP := FTMP(TEMP_2);
ELSE
        TEMP := 0;
        B_ERROR :=1;
END_CASE
```

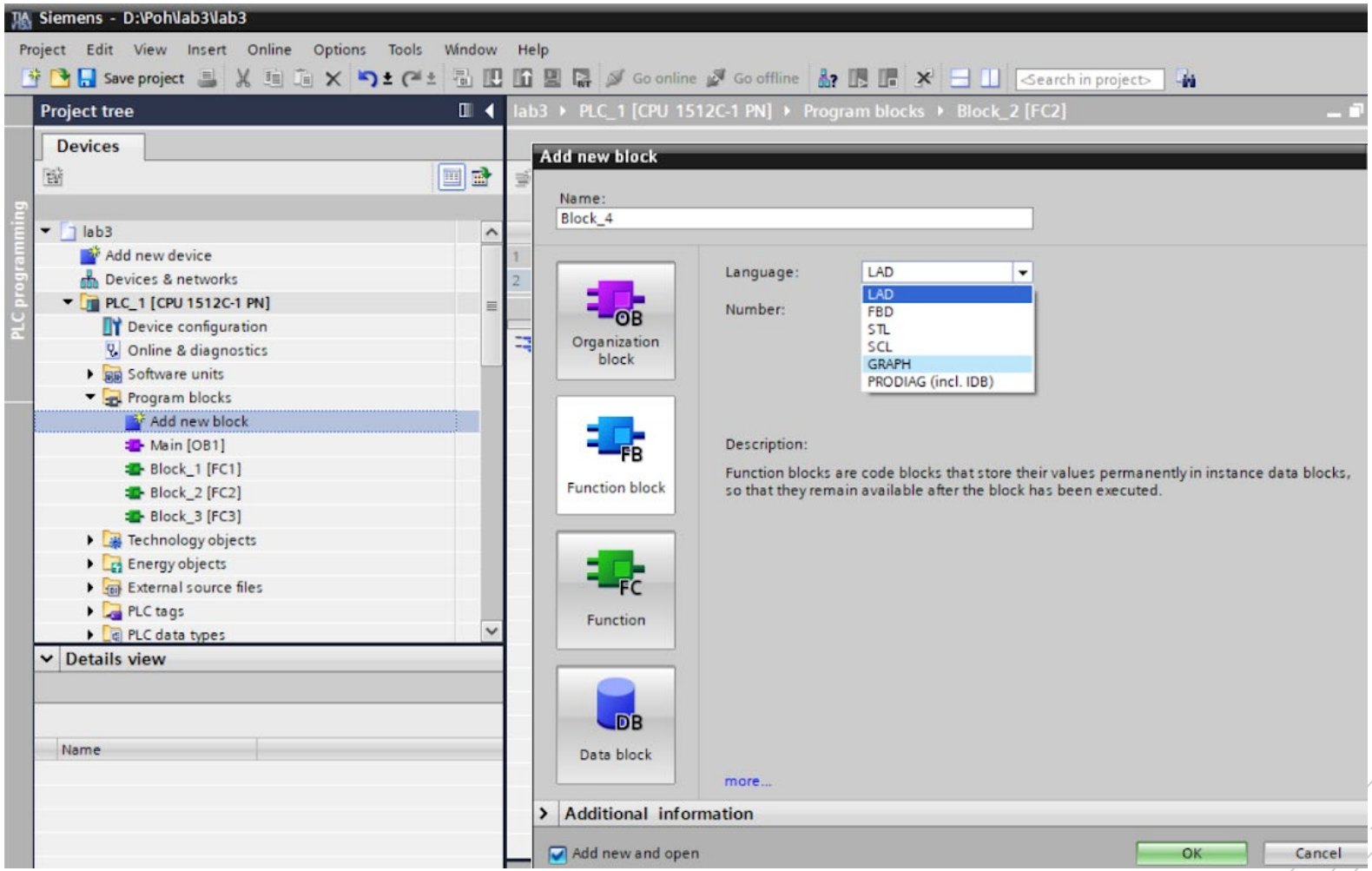
# Sequential Function Chart (SFC)

- It's a graphical tool ideal for programming sequential controls and implement state-based control
- It's a graphical approach to structuring program code
- Implementation would be with a mixture of the other languages

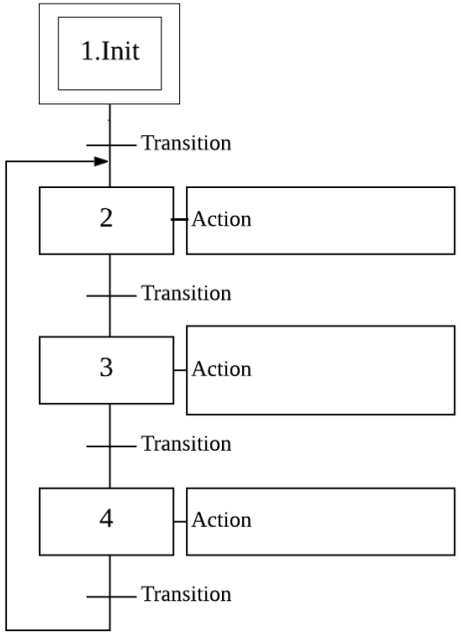


# Which Language to Use?

LAD – Ladder Diagram
FBD – Function Block Diagram
STL – Instruction List
SCL – Structured Text
GRAPH - SFC



# Sequential Function Chart (SFC)

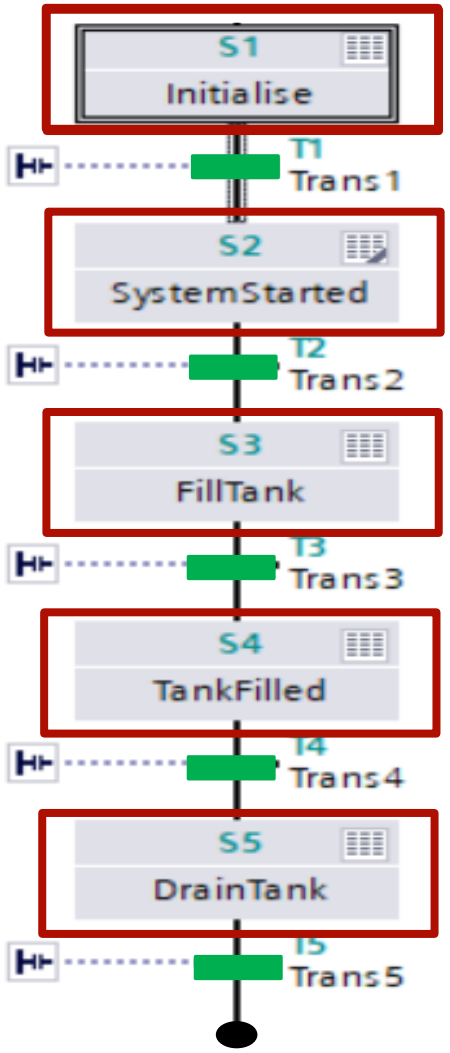



- Graphic programming language for creating sequential control systems
- Sequences programmed in fast straightforward manner using sequencers
- The process is broken down into individual steps
  - Each with a clear scope of functions, and organized into sequencers.
  - Actions to be executed are defined within the individual steps

Transitions are between the steps.

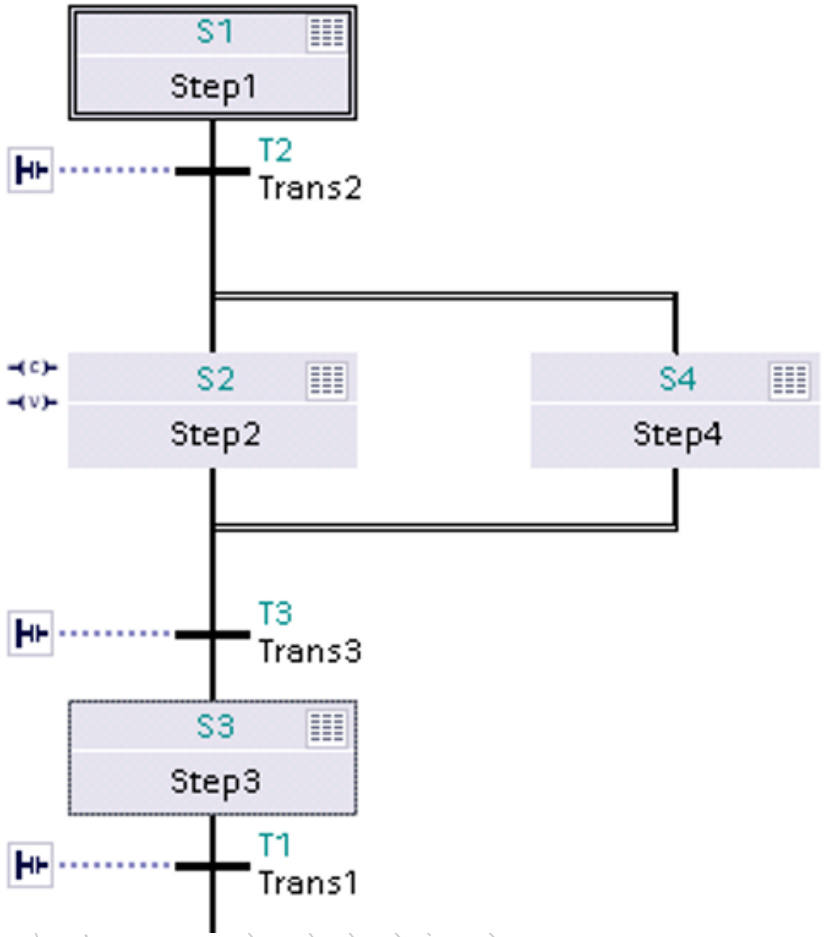
  - Contain conditions for advancing to the next step

# Sequential Function Chart (SFC)

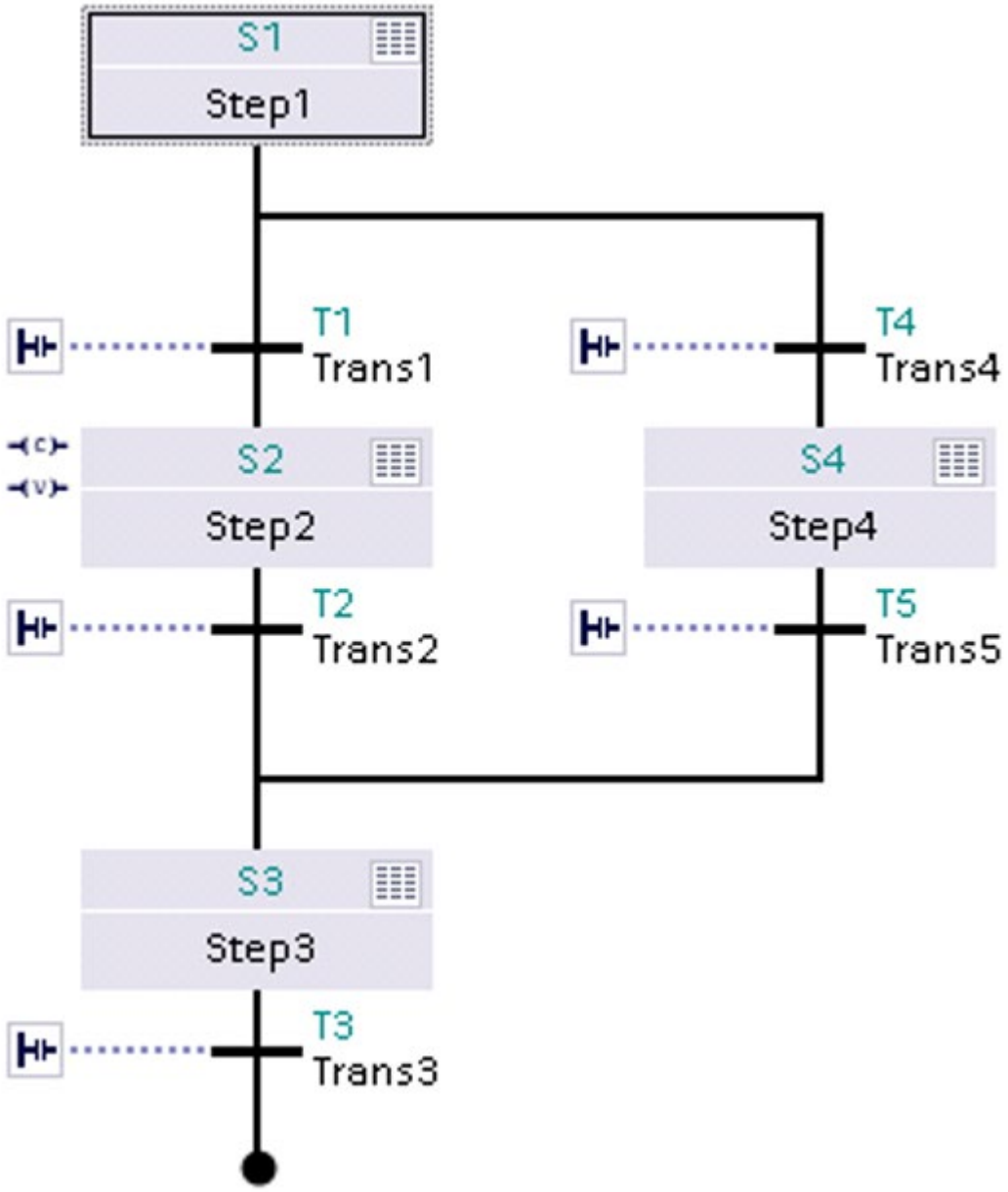



- The simplest case, the steps are processed linearly
- Sequencer starts with initial step
  - Execute actions defined in initial step
- When the condition of the next transition is met, the next step will be active
  - Execute actions defined
- At the end of a sequencer, you can use jumps to enable cyclic processing of the sequencer
- Or simply terminate the sequencer

# Simultaneous Branch

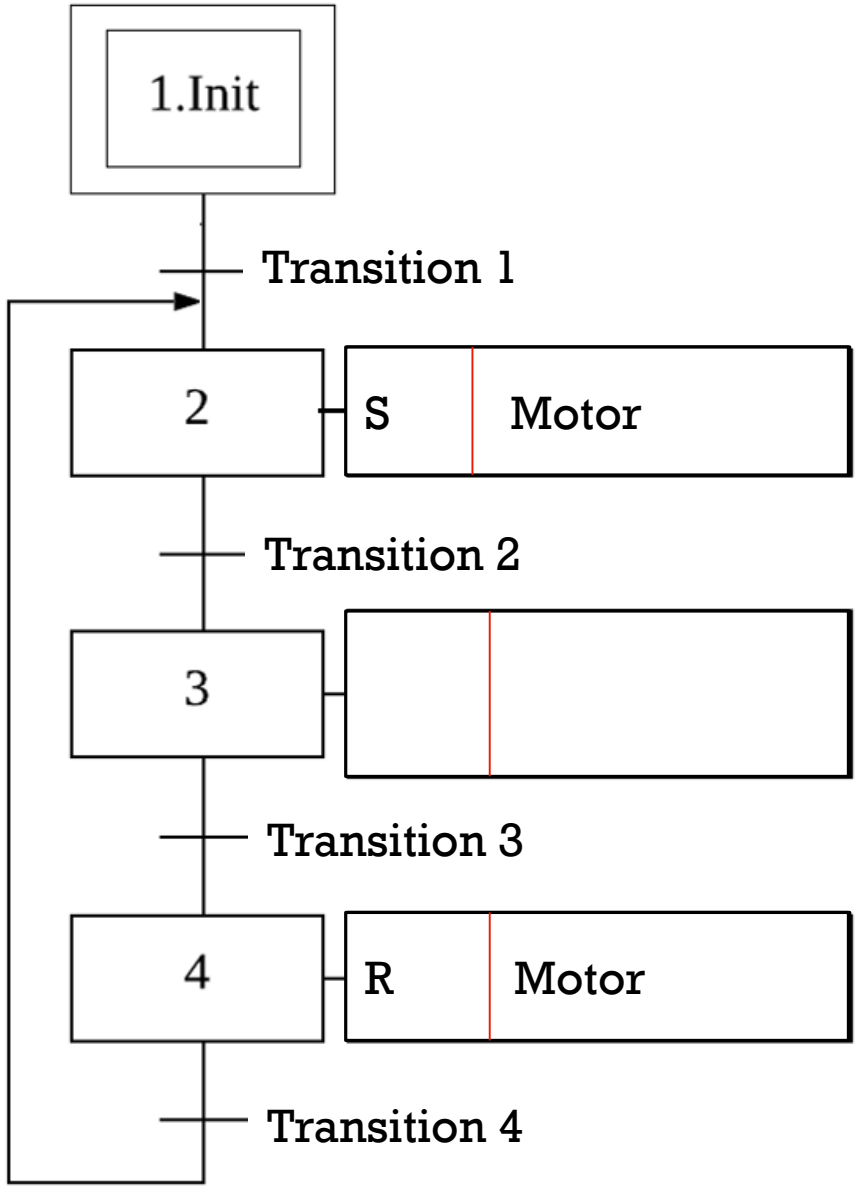



- Simultaneous branch can be used is for programming AND branches
- Note that simultaneous branches always start with a step
- One transition to activate multiple steps
- The actions of the simultaneous steps execute till the condition of the next transition is met
- When the branches that join together in a transition will only advance to the next step when all branches are completely processed




# Alternative Branch

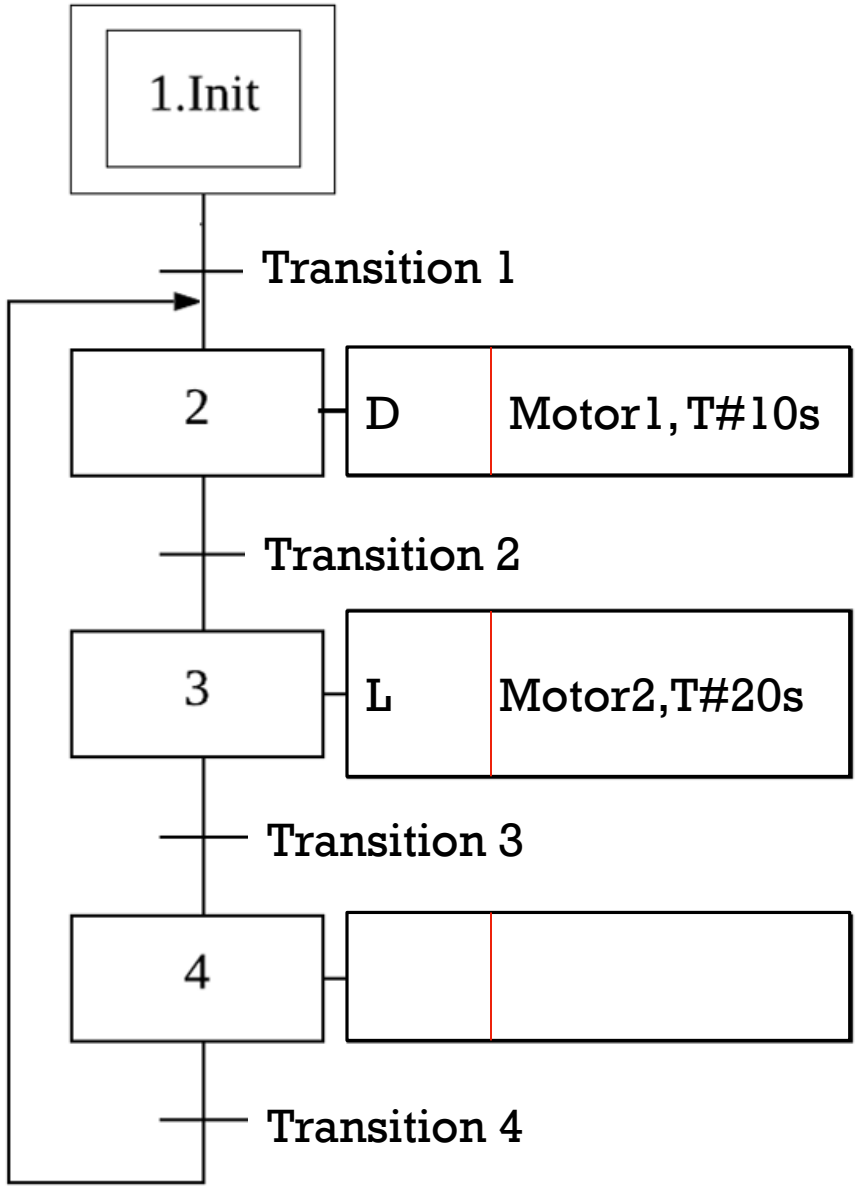
- Alternative  branch can be used is for programming OR branches
- Depending of which transition's condition is satisfied first, the corresponding branch is executed
- Close branch would close the alternative branch
  - Note that transition is required before closing alternative branch




# Actions

Standard actions are executed as long as the step is active

| Identifier | Meaning |
|---|---|
| N - Set as long as step is active | The signal state of the operand is "1" as long as the step is active. |
| S - Set to 1 | As soon as the step is active, the operand is set to "1" and then remains at "1". |
| R - Set to 0 | As soon as the step is active, the operand is set to "0" and then remains at "0". |



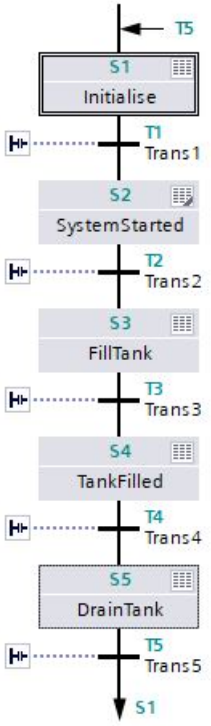

# Actions

Standard actions are executed as long as the step is active

| Identifier | Meaning |
|---|---|
| D - On delay | n seconds after the step activation, the operand is set to "1" and remains at "1" for the duration of the step activation. |
| L - Set for limited time | When the step is active, the operand is set to "1" for n seconds. The operand is then reset. |

Watch the SFC video with alternative way to program time delay -> https://youtu.be/t8jqKQmF-LM

# SFC Initialise Sequence




## INIT_SQ

- Activate initial step
- Reset Sequencer