Programmable Logic Controller (PLC)
Speicherprogrammierbare Steuerung (SPS)

Literature
R. W. Lewis:
“Programming industrial control systems using IEC 1131-3”

PLC Hardware
Software Architecture following IEC 61131-3

- **Common Elements**
  - Configurations
  - Resources
  - Tasks
  - Programs
  - Functions
  - Function Blocks
  - Data Types
  - Variable declarations
  - Initial values

- **Programming Languages**
  - Ladder Diagram (LD)
  - Function Block Diagram (FBD)
  - Instruction List (IL)
  - Structured Text (ST)
  - Sequential Function Chart (SFC)

---

**Code Structure**

**Configuration**
- Defines the software for a complete PLC
- Contains one or more resources
- Is specific to a particular type of PLC product and the arrangement of the PLC hardware
- Considers
  - Processing resources (type of micro processor)
  - Memory addresses (I/O)
  - System capabilities (max. number of tasks and execution rates)

**Resource**
- Corresponds to a processing facility
- Contains variable declarations, tasks and program declarations

**Task**
- Executes periodically or in response to a particular boolean variable state
- Runs a program or a function block

**Program**
- Largest form of program organization
- Contains variable declarations and a body which contains software describing the program’s behavior
- Can call programs, function blocks and functions
- Can only be declared within resources
Functions and Function Blocks

**Functions**
- Calculates a result depending on inputs
- Reusable
- Multiple Input
- Single Output
- Can **not** store values within internal variables
- No declaration
- Examples: ADD, MAX, AND, COS, GT (greater than)

**Function Blocks**
- Calculates a result depending on inputs and local variables
- Reusable
- Multiple Input
- Multiple Output
- Can store values within internal variables
- Declaration needed
- Examples: TON (On delay timer), TOF (Off delay timer), TP (Pulse timer), CTU (Counter up), SR (Set/Reset), R_TRIG (Rising trigger)

![FBD Diagram](represented as FBD)

**Generic Data Types**

<table>
<thead>
<tr>
<th>ANY</th>
<th>In parentheses: number of Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANY_BIT</td>
<td>BOO (1), BYTE (8), WORD (16), DWORD (32), LWORD (64)</td>
</tr>
<tr>
<td>ANY_NUM</td>
<td>ANY_INT</td>
</tr>
<tr>
<td></td>
<td>SINT (8), INT (16), DINT (32), LINT (64)</td>
</tr>
<tr>
<td></td>
<td>USINT (8), UINT (16), UDINT (32), ULINT (64)</td>
</tr>
<tr>
<td></td>
<td>ANY_REAL</td>
</tr>
<tr>
<td></td>
<td>REAL (32)</td>
</tr>
<tr>
<td></td>
<td>LREAL (64)</td>
</tr>
<tr>
<td>TIME (*)</td>
<td></td>
</tr>
<tr>
<td>ANY_DATE</td>
<td>DATE_AND_TIME (*)</td>
</tr>
<tr>
<td></td>
<td>DATE, TIME_OF_DAY (*)</td>
</tr>
<tr>
<td>ANY_STRING</td>
<td>STRING (*)</td>
</tr>
<tr>
<td></td>
<td>WSTRING (*)</td>
</tr>
</tbody>
</table>

(*) required memory depends on the system
Variable Declaration

- Variables are declared at the beginning of each POU
- POU...Program organization unit: program, function block, function
- Syntax of Structured Text
- Internal variables
  ```
  VAR
  AVE_SPEED : REAL;
  Inhibit : INT := 2;
  END_VAR
  ```
- Input variables
  ```
  VAR_INPUT
  SetPoint : REAL;
  MAX_Count : USINT := 150;
  END_VAR
  ```
- Output variables
  ```
  VAR_OUTPUT
  Message : STRING(10);
  Status : BOOL;
  END_VAR
  ```
- Input/Output variables: interface to other POU's
  ```
  VAR_IN_OUT
  counter : INT;
  END_VAR
  ```
- Global variables
  ```
  VAR_GLOBAL
  Job_Num : INT;
  END_VAR
  ```
- Temporary variables
  ```
  VAR_TEMP
  Result : REAL;
  END_VAR
  ```
- Directly represented variables: Adressing memory locations with %
  - First letter code: I...Input, Q...Output, M...Internal memory
  - Second letter code: X...Bit, B...Byte, W...Word, D...Double word, L...Long word
  Examples: %IX2.6, %QW122

Ladder Diagram (LD) – Kontaktplan (KOP)

- Historical background: Contactor circuits
- Application: Bit operations
- Structure: Networks
- Representation: Graphical
- Benefits:
  - Detection of signal edges
  - Clear expression for AND and OR operators
  - Easy to understand
- Disadvantages:
  - Bad support for operations except bit operations
  - Complex projects become confused

![Ladder Diagram](image.png)
Ladder Diagram Features

<table>
<thead>
<tr>
<th>Graphical feature</th>
<th>Graphic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normally open contact</td>
<td>P</td>
</tr>
<tr>
<td>Normally closed contact</td>
<td>P</td>
</tr>
<tr>
<td>Positive transition-sensing contact</td>
<td>N</td>
</tr>
<tr>
<td>Negative transition-sensing contact</td>
<td>N</td>
</tr>
<tr>
<td>Coil</td>
<td>M</td>
</tr>
<tr>
<td>Negated coil</td>
<td>M</td>
</tr>
<tr>
<td>SET coil</td>
<td>M</td>
</tr>
<tr>
<td>RESET coil</td>
<td>S</td>
</tr>
<tr>
<td>Retentive memory coil (variable is stored retained in memory)</td>
<td>R</td>
</tr>
<tr>
<td>SET retentive memory coil</td>
<td>M</td>
</tr>
<tr>
<td>RESET retentive memory coil</td>
<td>R</td>
</tr>
<tr>
<td>Positive transition-sensing coil</td>
<td>S</td>
</tr>
<tr>
<td>Negative transition-sensing coil</td>
<td>S</td>
</tr>
</tbody>
</table>

Function Block Diagram (FBD) – *Funktionsbausteinsprache (FBS)*

- Historical background: Electronic circuits
- Application: Logical connections
- Structure: Networks
- Representation: Graphical
- Benefits:
  - Good support for arithmetic operations
  - Easy use of functions and function blocks
  - Easy to understand
- Disadvantages:
  - Needs much space for documentation
Instruction List (IL) – Anweisungsliste (AWL)

- Historical background: Machine assembler language
- Application: Difficult problems with branches
- Structure: Rows
- Representation: Textual
- Benefits:
  - Easier to implement for PLC designer
  - Flexible
  - Suitable for performance optimized execution
- Disadvantages:
  - Hard to follow the program flow (especially when using jumps)

<table>
<thead>
<tr>
<th>Label</th>
<th>Operator</th>
<th>Operand</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>DoorShut</td>
<td>(* If the door is shut *)</td>
<td></td>
</tr>
<tr>
<td>ANDN</td>
<td>DrainShut</td>
<td>(* and the Drain switch is NOT shut *)</td>
<td></td>
</tr>
<tr>
<td>OR</td>
<td>Manual</td>
<td>(* or the manual switch is pressed *)</td>
<td></td>
</tr>
<tr>
<td>ST</td>
<td>PumpDown</td>
<td>(* then turn on the pump *)</td>
<td></td>
</tr>
</tbody>
</table>

Instruction List Operators (1)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Modifiers</th>
<th>Operand</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>N</td>
<td>ANY</td>
<td>Load operand into result register</td>
</tr>
<tr>
<td>ST</td>
<td>N</td>
<td>ANY</td>
<td>Store result register into operand</td>
</tr>
<tr>
<td>S</td>
<td>BOOL</td>
<td>Set operand true</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>BOOL</td>
<td>Reset operand false</td>
<td></td>
</tr>
<tr>
<td>AND</td>
<td>N,(</td>
<td>ANY</td>
<td>Boolean AND</td>
</tr>
<tr>
<td>&amp;</td>
<td>N,(</td>
<td>ANY</td>
<td>Boolean AND (equivalent to AND)</td>
</tr>
<tr>
<td>OR</td>
<td>N,(</td>
<td>ANY</td>
<td>Boolean OR</td>
</tr>
<tr>
<td>XOR</td>
<td>N,(</td>
<td>ANY</td>
<td>Boolean exclusive OR</td>
</tr>
<tr>
<td>NOT</td>
<td></td>
<td>ANY</td>
<td>Logical negation (one's complement)</td>
</tr>
<tr>
<td>ADD</td>
<td></td>
<td>ANY</td>
<td>Addition</td>
</tr>
<tr>
<td>SUB</td>
<td></td>
<td>ANY</td>
<td>Subtraction</td>
</tr>
<tr>
<td>MUL</td>
<td></td>
<td>ANY</td>
<td>Multiplication</td>
</tr>
<tr>
<td>DIV</td>
<td></td>
<td>ANY</td>
<td>Division</td>
</tr>
<tr>
<td>MOD</td>
<td></td>
<td>ANY</td>
<td>Modulo-division</td>
</tr>
</tbody>
</table>
### Instruction List Operators (2)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Modifiers</th>
<th>Operand</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>GT</td>
<td>(</td>
<td>ANY</td>
<td>Comparison greater than</td>
</tr>
<tr>
<td>GE</td>
<td>(</td>
<td>ANY</td>
<td>Comparison greater than, equal</td>
</tr>
<tr>
<td>EQ</td>
<td>(</td>
<td>ANY</td>
<td>Comparison equal</td>
</tr>
<tr>
<td>NE</td>
<td>(</td>
<td>ANY</td>
<td>Comparison not equal</td>
</tr>
<tr>
<td>LE</td>
<td>(</td>
<td>ANY</td>
<td>Comparison less than, equal</td>
</tr>
<tr>
<td>LT</td>
<td>(</td>
<td>ANY</td>
<td>Comparison less than</td>
</tr>
<tr>
<td>JMP</td>
<td>C,N</td>
<td>LABEL</td>
<td>Jump to label</td>
</tr>
<tr>
<td>CAL</td>
<td>C,N</td>
<td>NAME</td>
<td>Call function block</td>
</tr>
<tr>
<td>RET</td>
<td>C,N</td>
<td></td>
<td>Return from function or function block</td>
</tr>
<tr>
<td>)</td>
<td></td>
<td></td>
<td>Execute the last deferred operator</td>
</tr>
</tbody>
</table>

### Structured Text (ST) – *Strukturierter Text (ST)*

- **Historical background**: High level programming languages (similar to Pascal)
- **Application**: Complex arithmetic calculations, loops
- **Structure**: Semicolon separated statements
- **Representation**: Textual
- **Benefits**:
  - Clear representation of loops
  - Compact
- **Disadvantages**:
  - No “free” jumps
  - No “SET” and “RESET”

```text
PumpDown := DoorShut AND NOT DrainShut OR Manual; (* Comment *)
```

- **Assignment**  
- **Operand**  
- **Operator**  
- **End of statement**  
- **Comment**
Structured Text Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>(...)</td>
<td>Parenthesis expression</td>
<td>Highest</td>
</tr>
<tr>
<td>Function(...)</td>
<td>Parameter list of a function, function evaluation</td>
<td></td>
</tr>
<tr>
<td>**</td>
<td>Exponentiation, i.e. raising to a power</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>Negation</td>
<td></td>
</tr>
<tr>
<td>NOT</td>
<td>Boolean complement, i.e. value with opposite sign</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td></td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td></td>
</tr>
<tr>
<td>MOD</td>
<td>Modulus operation</td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>Addition</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td></td>
</tr>
<tr>
<td>&lt;, &gt;, &lt;=, &gt;=</td>
<td>Comparison operators</td>
<td></td>
</tr>
<tr>
<td>=</td>
<td>Equality</td>
<td></td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Inequality</td>
<td></td>
</tr>
<tr>
<td>AND, &amp;</td>
<td>Boolean AND</td>
<td></td>
</tr>
<tr>
<td>XOR</td>
<td>Boolean exclusive OR</td>
<td></td>
</tr>
<tr>
<td>OR</td>
<td>Boolean OR</td>
<td>Lowest</td>
</tr>
</tbody>
</table>

Sequential Function Chart (SFC) – Ablaufsprache (AS)

- Historical background: Automata, Petri nets, Grafcet
- Application: Sequential control, structuring of processes, not useful for low level
- Structure: Steps and Transitions
- Representation: Graphical
- Benefits:
  - Visualization of main states
  - Clear overview
  - Well structured
- Disadvantages:
  - Bad support for operations
Sequential Function Chart Branching

Alternative branching
• Only one leg is executed

Simultaneous branching
• All parallel legs are executed

Transition Conditions for SFC

• Conditions are assigned to the Transitions by Structured Text, Function Block Diagram or Ladder Diagram
• Result of an expression must be a boolean value
### SFC Action Qualifiers

<table>
<thead>
<tr>
<th>Qualifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Non-stored, default, same as ‘N’.</td>
</tr>
<tr>
<td>N</td>
<td>Non-stored, executes while associated step is active.</td>
</tr>
<tr>
<td>R</td>
<td>Resets a stored action.</td>
</tr>
<tr>
<td>S</td>
<td>Sets an action active, i.e. stored.</td>
</tr>
<tr>
<td>L Time</td>
<td>Time limited action, terminates after a given period.</td>
</tr>
<tr>
<td>D Time</td>
<td>Time delayed action, starts after a given period.</td>
</tr>
<tr>
<td>P</td>
<td>A pulse action that only executes once when a step is activated, and once when the step is deactivated.</td>
</tr>
<tr>
<td>P1</td>
<td>A pulse action that only executes once when a step is activated.</td>
</tr>
<tr>
<td>P0</td>
<td>A pulse action that only executes once when a step is deactivated.</td>
</tr>
<tr>
<td>SD Time</td>
<td>Stored and time delayed. The action is set active after a given period, even if the associated step is deactivated before the delay period.</td>
</tr>
<tr>
<td>DS Time</td>
<td>Action is time delayed and stored. If the associated step is deactivated before the delay period, the action is not stored.</td>
</tr>
<tr>
<td>SL Time</td>
<td>Stored and time limited. The action is started and executes for a given period.</td>
</tr>
</tbody>
</table>

---

### Call Function Blocks and Programs

**Declaration**

```plaintext
VAR
  c            : BOOL;
  SetReset     : SR;
END_VAR
```

**IL**

- `CAL SetReset(SET1:=TRUE, RESET:=FALSE)
- `LD SetReset.Q1
- `ST c`

**ST**

- `SetReset(SET1:=TRUE, RESET:=FALSE);
- `c:=SetReset.Q1;`
### SET Operator

**Declaration**

```
VAR
  a : BOOL := TRUE;
  c : BOOL;
END_VAR
```

### Arithmetic Operators

**Declaration**

```
VAR
  a: INT := 4;
  b: INT := 22;
  c: REAL;
END_VAR
```

**Equation**

\[ c := (2^a - b) \times a; \]
### Comparison Operators

**Declaration**

```plaintext
VAR
  a : INT := 44;
  b : INT := 42;
  ab : INT; (*LD*)
  c : BOOL;
END_VAR
```

```plaintext
LD a  
GE b  
AND (  
  LD b  
  NE 42 )  
ST c
```

```plaintext
c := (a >= b) AND (b < 42);
```

### Jump Operator / Loop

**Declaration**

```plaintext
VAR
  a : INT := 0;
  b : INT; (*LD*)
END_VAR
```

```plaintext
X:  
LD 1  
ADD a  
LT 10  
JMPC X
```

```plaintext
REPEAT
  a := a + 1;
UNTIL a >= 10
END_REPEAT
```