PLC programming

Paolo Burgio
paolo.burgio@unimore.it

UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA
High Performance
Real Time Lab

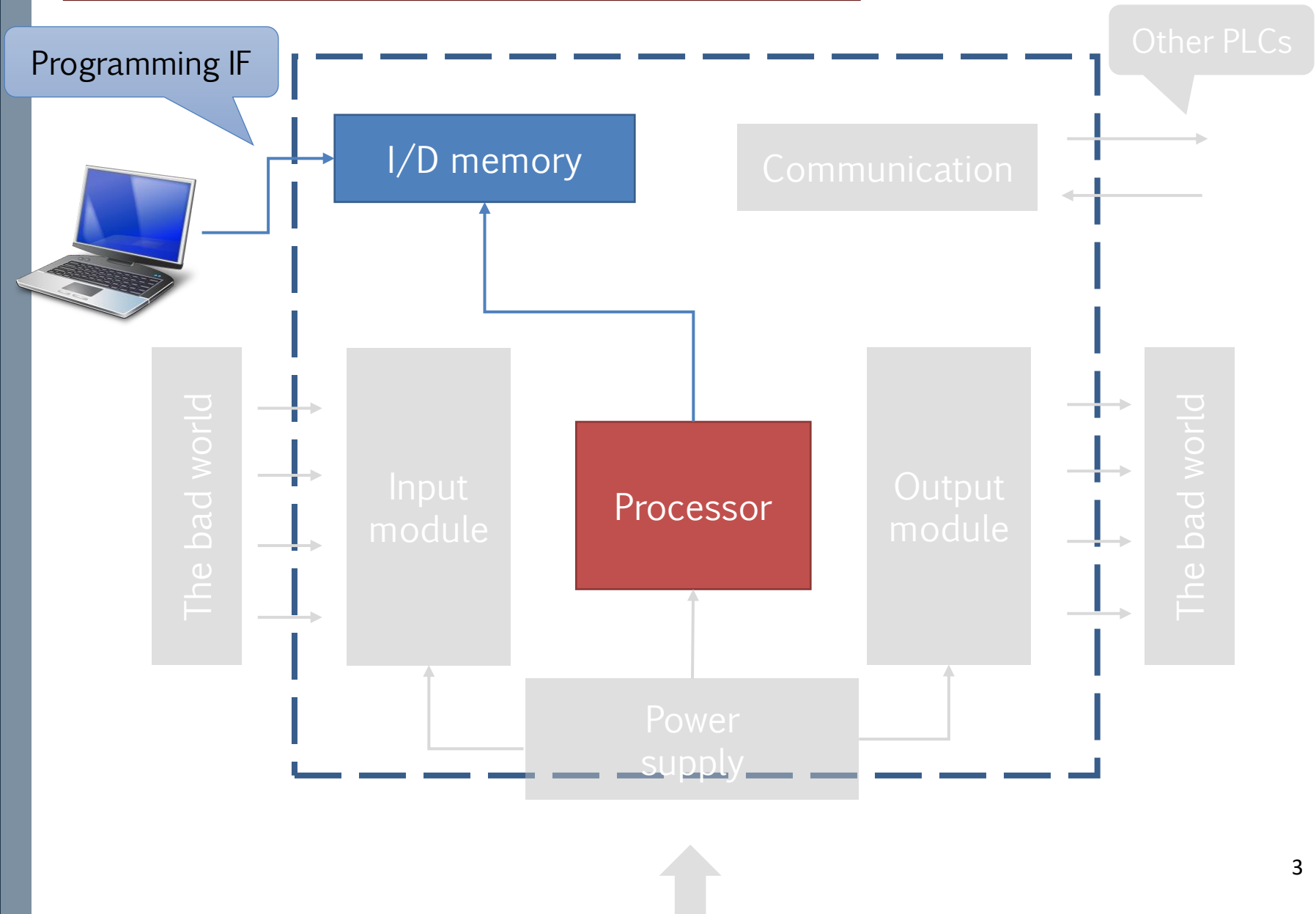# Structure of a PLC

PLC programming with IEC 61131

First attempt of standardization => **IEC 61131-3 standard**

› yr 1993, latest rev 2013

› Before that, "the fish market" of languages

› Still, Ladder was a prominent one...but everyone had its own variant!!

States that there are 5 "standard" ways of programming PLCs

› **Ladder diagram**
  – Description of electrical wiring, designed for non-informatics

› **Function Block Diagram – FBD**
  – From electronics

› Sequential Functional Chart – SFC
  – Petri-net style

› Instruction List – IL
  – ASM-like

› **Structured Text – ST**
  – Similar to Pascal/VB

IEC 61131

Covers the complete lifecycle of PLC modules and ~~sw development for PLC~~ PLC programming

› Part 1: Definition of <u>terminology</u> and concepts

› Parte 2: Electronic and mech equipment and <u>verification/testing</u>

› Part 3: programming languages (5 types)

› Part 4: how to choose, install and <u>maintain</u>

› Part 5: how to <u>communicate</u> (MMS – Manufacturing Messaging specification)

› Part 6: communication via fieldbusses/other ind. Standards

› Part 7: Fuzzy control (won't see this..if you don't want)
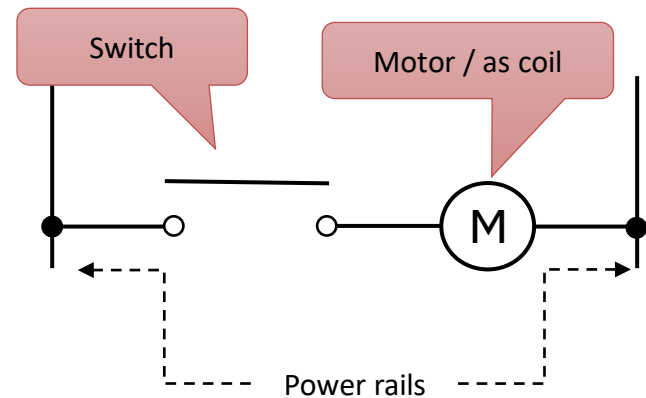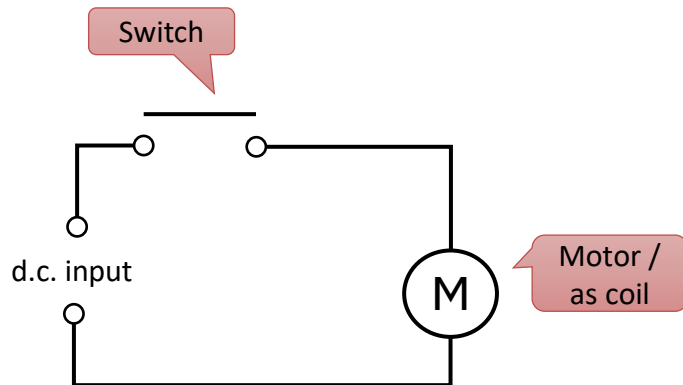
› Part 8: sw dev <u>guidelines</u>

Ladder

Ladder diagrams

Possible circuit to power on a motor

› Left: electrical diagram; right: Ladder

› Does it remind of something…?

Ladder programs

Writing a program is as easy as writing a switching circuit

› Vertical lines are power rails, and power flows from top-left to bottom-right
  – So does "program flow"

› Horizontal lines (rungs) connect power rails
  – Every rung starts with one or more inputs, and ends with exactly one output

› Typical exec time: 1ms for 1k bytes of program, so usually approx 10-50ms

Program flow

› Store input status in mem

› Read inputs from memory, run program, store out in mem

› Update all outputs

Power flow

Rung 1

Rung 2

Rung 3

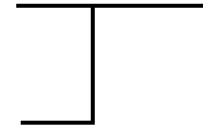END Rung
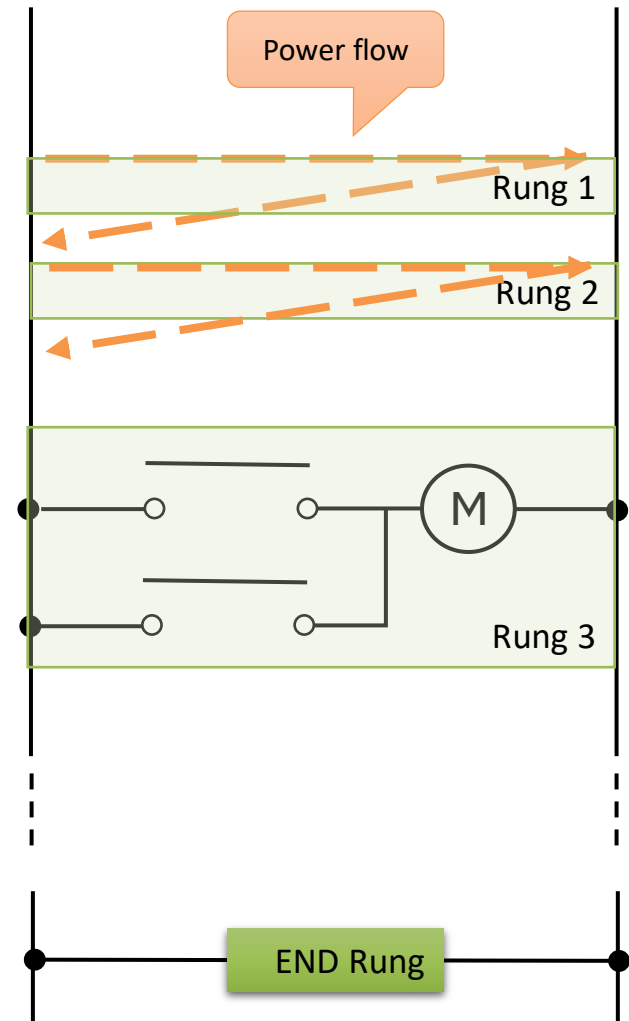
Ladder symbols

› Power rails (Vert lines)

› Rugs (Horiz lines)

› Left/right connection between power rail and rugs

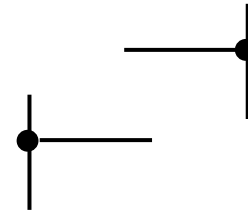› Dual connection

› Normally open (NO) contact

› Normally closed (NC) contact

› Output coil (from a lamp, a motor...)
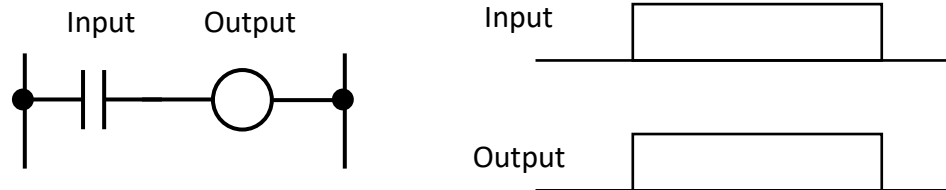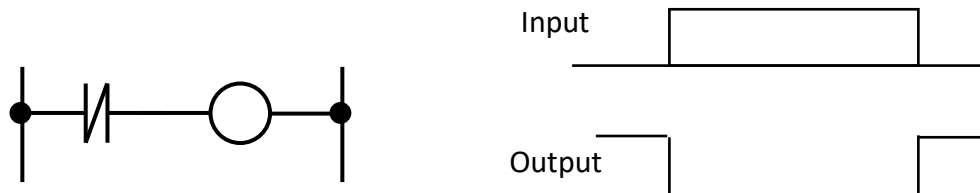
› A switch

Example: a (lamp?) witch

› When a NO contact is closed, the attached coil propagates the signal

› When a NC contact is opened, the attached coil propagates the input signal

Let's see this in action

Structured Text

Structured text

› Inspired by Pascal language, also Basic and VB programmer might find it familiar

› Procedural language (can define Functions, and Function Blocks)

› Or define a Program (and call it from within your Application MainTask)

Defining variables

Can also initialize them

› Don't forget semicolon **;**

IEC 61131.3 defines several datatypes

› Few examples

```
Contact1: BOOL;

Contact2: BOOL := FALSE;
```

IEC Data Type	Format	Range
SINT	Short Integer	-128 … 127
INT	Integer	-32768 … 32767
DINT	Double Integer	-2^{31} … $2^{31}-1$
UINT	Unsigned Integer	0 … $2^{16}-1$
BOOL	Boolean	1 bit
BYTE	Byte	8 bits
WORD	Word	16 bits
STRING	Character String	'My string'
TIME	Duration of time after an event	T#10d4h38m57s12ms TIME#10d4h38m
DATE	Calendar date	D#1989-05-22 DATE#1989-05-22

Variables: simple operations

› Assigning variables

```
Contact1 := FALSE;
Input1 := 11;
Output1 := 5;
```

```
Contact1: BOOL;
Input1: INT;
Output1: BYTE;
```

› Also, with operators in R-values

```
Output1 := Input1 – Output1 / ( Input6 + 3);
```

Let's see this in action

Structured Text Operators (and their priority)

Operators	Description
(. . .)	Parenthesized (brackets) expression
Function (. . .)	List of parameters of a function
**	Power
-, NOT	Negation, Boolean NOT
* , / , MOD	Multiplication, division, modules operations
+, -	Add, subract
< , > , <= , >=	Comparison
=, <>	Comparison
AND, OR	Boolean operator
XOR	Exclusive OR
OR	Boolean OR

Priority

Variables: nomenclature

› ST is not case sensitive; in case you might want to use capital letter for clarity

› Use the `AT` keyword to fix the memory location of a variable

```
Contact1: BOOL AT %MX100; (* Internal memory Bit at address 100 *)

Input1: INT AT %IW200; (* Input memory Word at address 200 *)

Output1: BYTE AT %OB300; (* Output memory Byte at address 300 *)
```

```
% M X 100
```

Memory Address

Memory type
M - internal
I - input
O - output

Data type
X (bit) – 1 bit
B (byte) – 8 bit
W (word) – 16 bits
D (double word) – 32 bits
L (long word) – 64 bits

16

If-Then-Else

```
IF Contact1 = TRUE THEN;
  Coil1 := TRUE;
ELSE;
  Coil1 := FALSE;
END_IF;
```

```
IF NOT Contact1 = TRUE;
  Coil1 := TRUE;
END_IF;
```

```
IF Contact1 = TRUE OR Input1 = 11;
  Coil1 := TRUE;
END_IF;
```

Switch-case

› Multiple instructions on the same line

› Default with `ELSE` keyword

› Can also use ranges

```
CASE (STATE) OF
  1: NEXT_STATE := 2; ERROR := FALSE;
  2: NEXT_STATE := 3; ERROR := FALSE;
ELSE
  NEXT_STATE := 4; ERROR := TRUE;
END_CASE;
```

```
CASE (TEMPERATURE) OF
  0...40: Furnace_switch := ON;
  40...100: Furnace_switch := OFF;
END_CASE ;
```

Defining Program, Functions, Function Blocks

Very simple program

› P174

Loops

Functional Block Programming

Implementing state machines into IEC 61131

References

Course website

› http://hipert.unimore.it/people/paolob/pub/Industrial_Informatics/index.html

My contacts

› paolo.burgio@unimore.it

› http://hipert.mat.unimore.it/people/paolob/

Resources

› W. Bolton, ''Programmable Logic Controllers'', 6th edition, Newnes

› "Industrial informatics" course by Proff. Vezzani and Pazzi @UNIMORE

› A "small blog"
    – http://www.google.com