PLC e SCADA, Sect.3

Alessandra Flammini
alessandra.flammini@unibs.it
Stefano Rinaldi
stefano.rinaldi@unibs.it
Ufficio 24 Dip. Ingegneria dell'Informazione
030-3715627 Martedì 16:30-18:30

Programmable Logic Controller Programming

PLC needs "Real-Time" languages

Extend procedural languages to express time

("introduce programming constructs to influence scheduling and control flow")

- ADA
- Real-Time Java
- MARS (TU Wien)
- Forth
- "C" with real-time features
- etc…

could not impose themselves

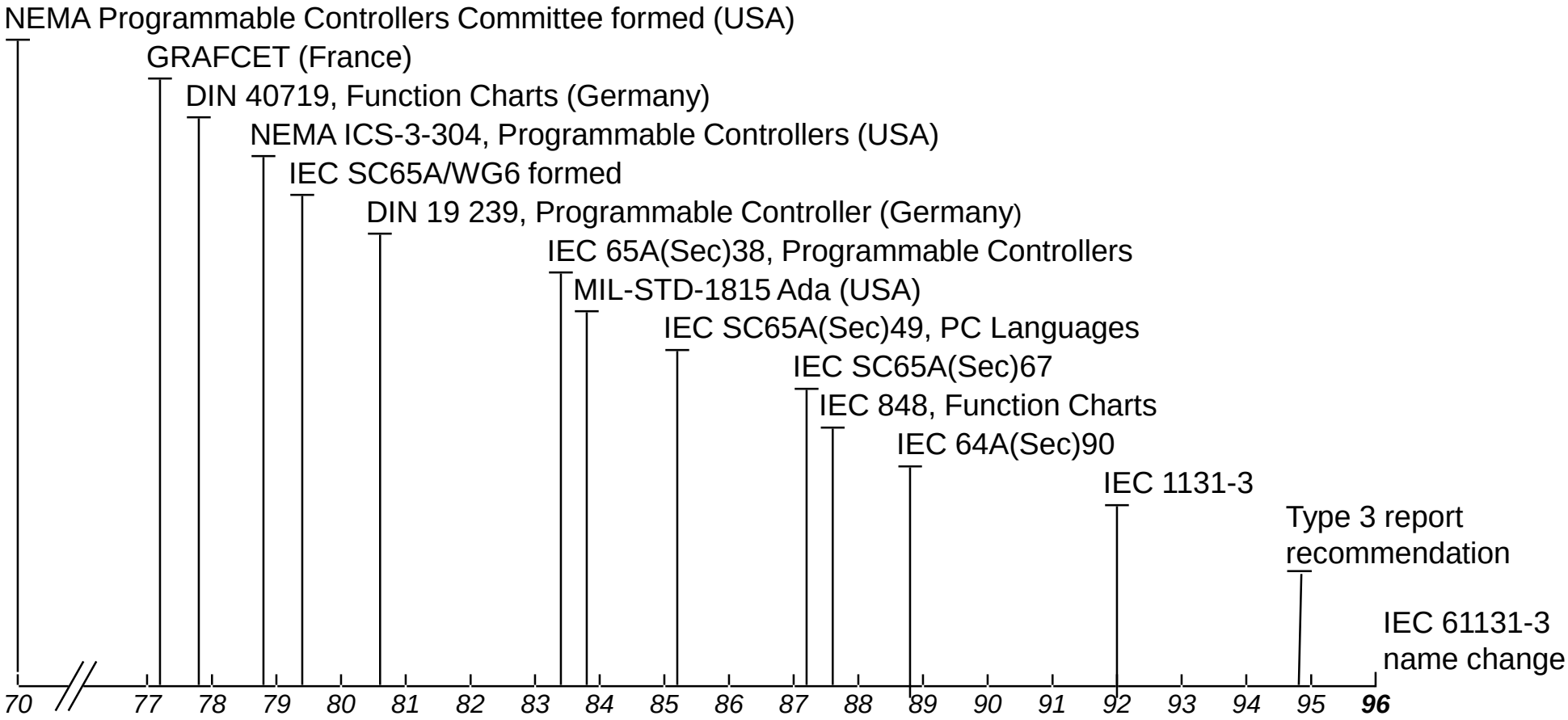
languages developed for cyclic execution and real-time ("application-oriented languages")

- ladder logic
- function block language
- instruction lists
- GRAFCET
- SDL

etc...

wide-spread in the control industry. Now standardized as IEC 61131

The long march to IEC 61131



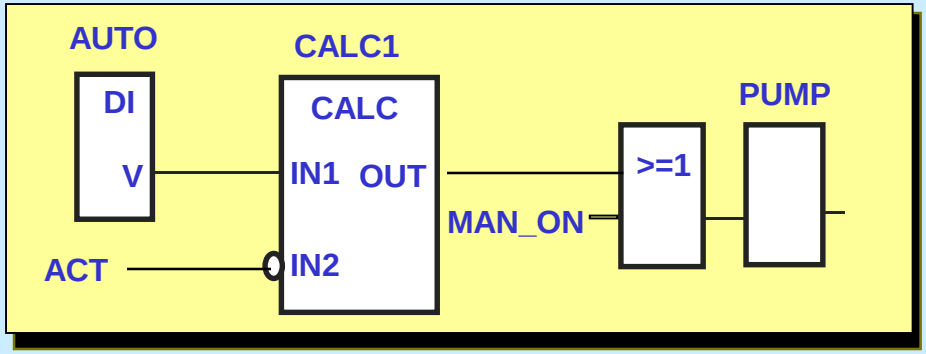
Source: Dr. J. Christensen

it took 20 years to make that standard…

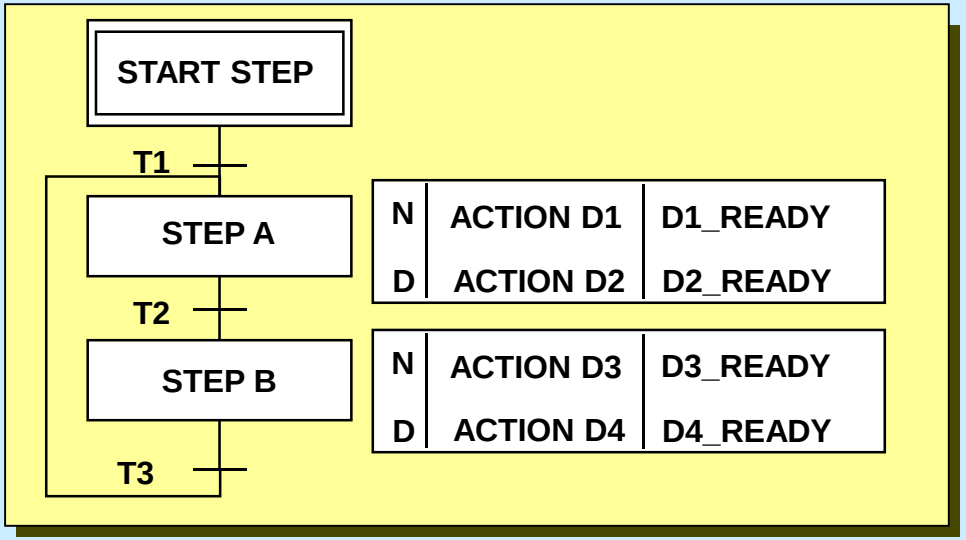
The five IEC 61131-3 Programming languages

graphical languages

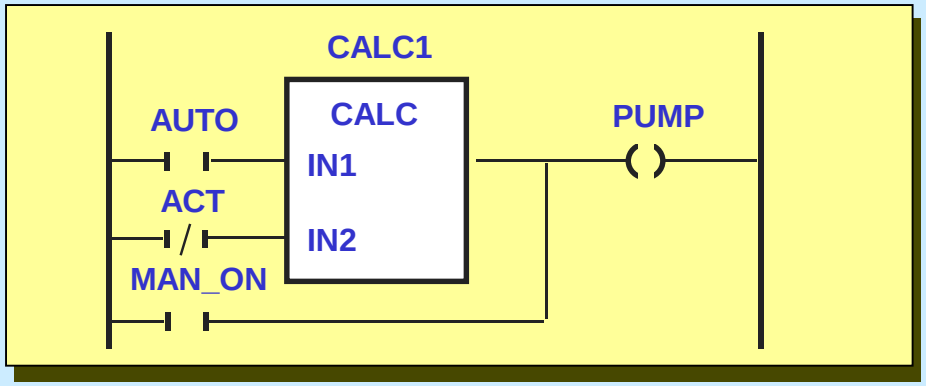
Function Block Diagram (FBD)



Sequential Flow Chart (SFC)



Ladder Diagram (LD)



textual languages

Structured Text (ST)

```
VAR CONSTANT X : REAL := 53.8 ;
Z : REAL; END_VAR
VAR aFB, bFB :  FB_type; END_VAR

bFB(A:=1, B:='OK');
Z := X - INT_TO_REAL (bFB.OUT1);
IF Z>57.0 THEN aFB(A:=0, B:="ERR");
ELSE  aFB(A:=1, B:="Z is OK");
END_IF
```

Instruction List (IL)

```
A: LD    %IX1  (* PUSH BUTTON *)
   ANDN %MX5  (* NOT INHIBITED *)
   ST    %QX2  (* FAN ON *)
```

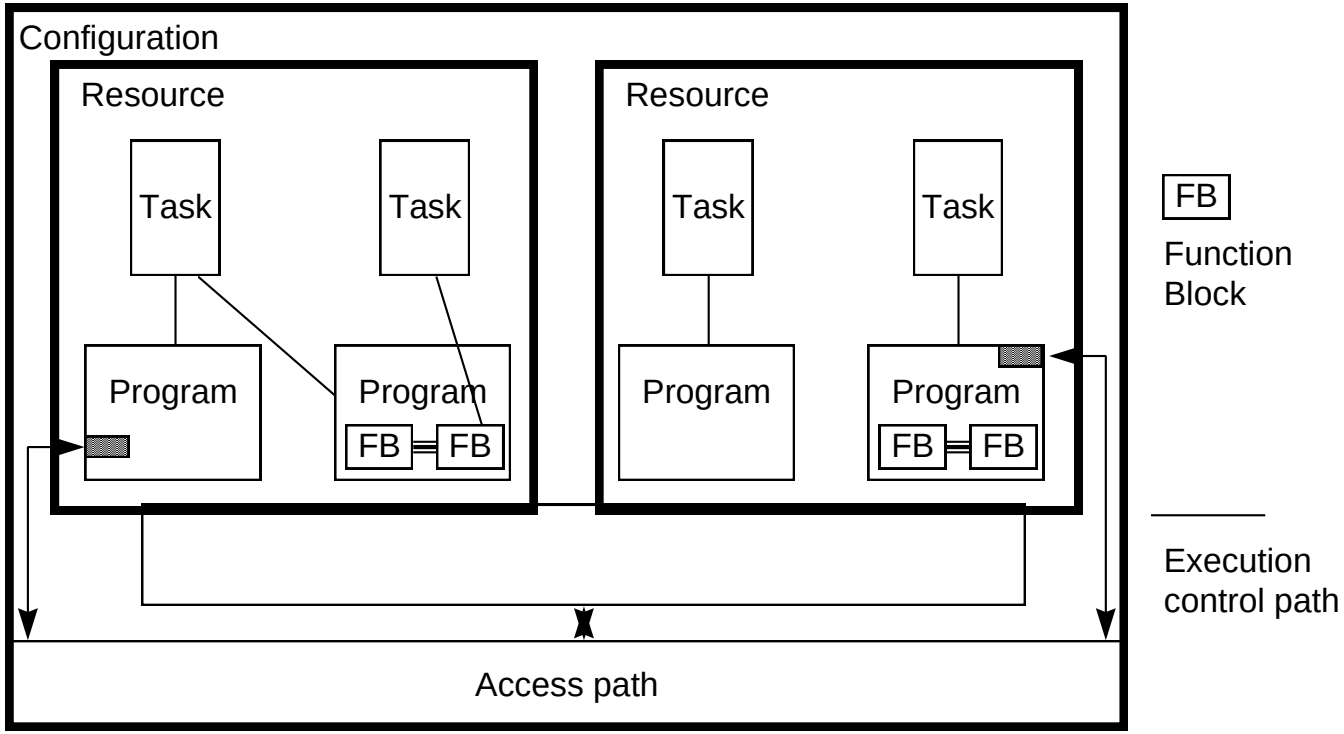
http://www.isagraf.com

Importance of IEC 61131 (not only IEC61131-3)

IEC 61131-3 is the most important automation language in industry.

80% of all PLCs support it, all new developments base on it.

Depending on the country, some languages are more popular.



Importance of IEC 61131

Configuration. A configuration is specific to a particular type of control system, including the arrangement of the hardware, i.c. processing resources, memory addresses for I/O channels and system capabilities.
Within a configuration one can define one or more *Resources*. One can look at a resource as a processing facility that is able to execute IEC programs.
Within a resource, one or more *Tasks* can be defined. Tasks control the execution of a set of programs and/or function blocks. These can either be executed periodically or upon the occurrence of a specified trigger, such as the change of a variable.
Programs are built from a number of different software elements written in any of the IEC defined languages. Typically, a program consists of a network of *Functions* and *Function Blocks*, which are able to exchange data. Function and Function Blocks are the basic building blocks, containing a data structure (FB) and an algorithm.
Let's compare this to a conventional PLC: this contains one resource, running one task, controlling one program, running in a closed loop.
IEC 61131-3 adds much to this, making it open to the future. A future that includes multi-processing and event driven programs. And this future is not so far: just look at distributed systems or real-time control systems.
IEC 61131-3 is suitable for a broad range of applications, without having to learn additional programming languages.

http://www.plcopen.org/pages/pc2_training/downloads/index.htm

Standard *IEC 1131* – Lo standard dei PLC

Lo standard *IEC 1131* è diviso in diverse parti:

- Parte 1: Informazioni Generali
- Parte 2: Requisiti e test dei dispositivi
- Parte 3: Linguaggi di Programmazione
- Parte 4: Linea guida per l'utente
- Parte 5: Definizione dei servizi di messaggio
- Parte 6: Comunicazione attraverso bus di campo
- Parte 7: Programmazione controllo Fuzzy
- Parte 8: Linee guida per le applicazioni e l'implementazione dei linguaggi di programmazione

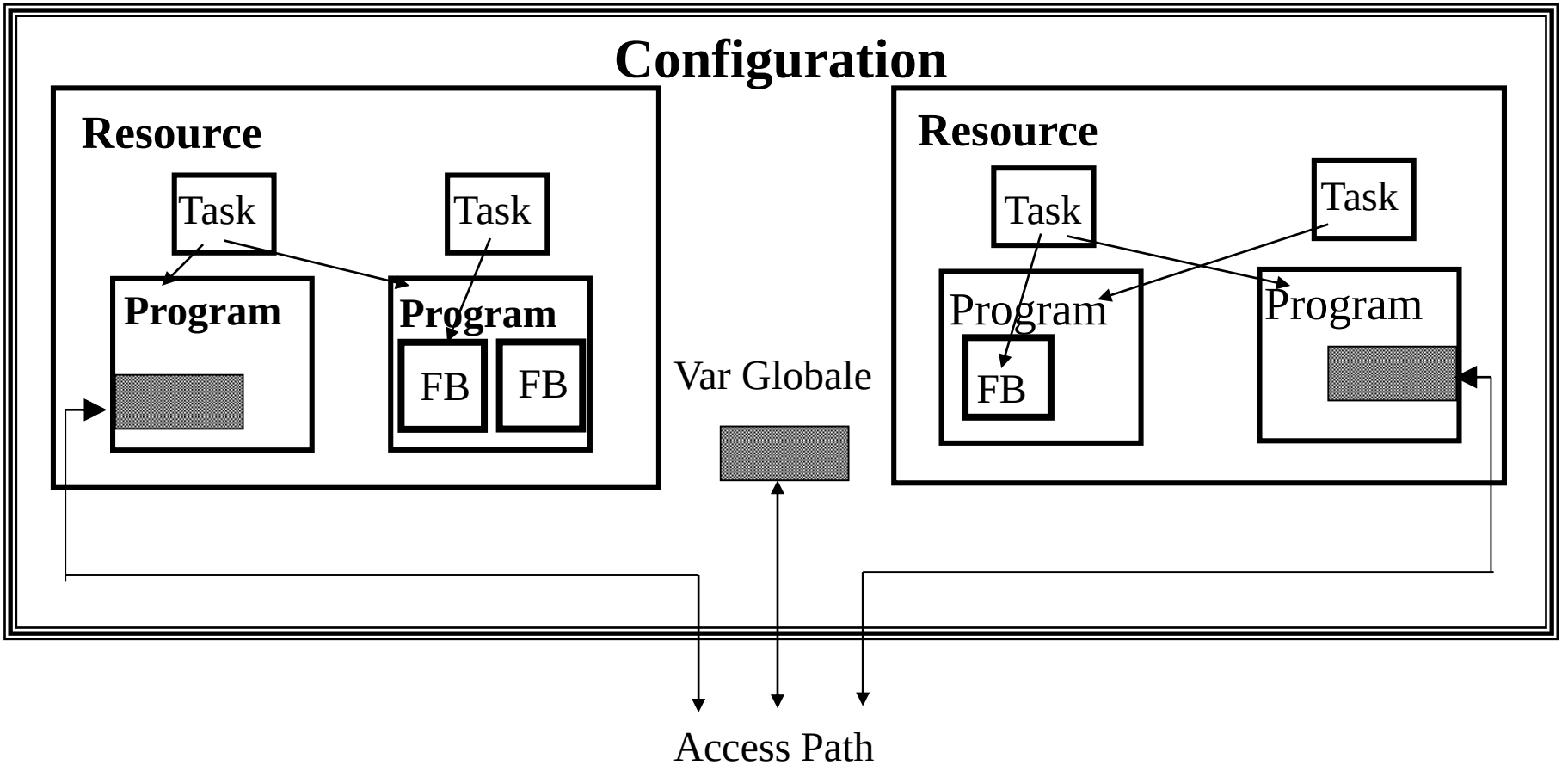
Definizione di PLC "Sistema elettronico a funzionamento digitale, destinato all'uso in ambito industriale, che utilizza una memoria programmabile per l'archiviazione interna delle istruzioni orientate all'utilizzatore per l'implementazione di funzioni specifiche, come quelle logiche, di sequenziamento, di temporizzazione, di conteggio e di calcolo aritmetico, e per controllare, mediante ingressi ed uscite sia digitali che analogiche, vari tipi di macchine e processi."

Caratteristiche principali dello standard IEC 1131-3

- Sviluppo strutturato del software
  - Approccio bottom-up o top-down
  - Suddivisione del programma in elementi funzionali (POU) clonabili e riusabili
- Strong data typing
  - Verifica della coerenza tra tipo di dato e assegnazione e/o operazione
- Pieno controllo dell'esecuzione
  - Suddivisione del programma in task con differenti tempi di scansione
- Coesistenza di più linguaggi (5) per la descrizione di un'applicazione
  - Ladder, Sequential Function Chart (SFC), Instruction List (IL), Function Block Diagram (FBD), Structured Text(ST)
- Supporto alla descrizione delle sequenze
  - Linguaggio SFC (Sequential Flow Chart)
- Supporto alla gestione di strutture dati (es. record e vettori) -> integrazione (database)

- Portabilità del software tra PLC di costruttori differenti

Modello Software dello standard IEC 1131-3



Modello Software dello standard IEC 1131-3

- Il modello software IEC è a strati: ogni strato nasconde molte delle caratteristiche dei gli strati inferiori (astrazione).
- Configuration. Al livello più alto, il software per un particolare problema di controllo è contenuto in una Configuration. Una Configuration definisce il comportamento di un PLC per una specifica applicazione.
- Generalmente una Configuration equivale al software richiesto per un PLC. In applicazioni più complesse, in cui sono necessari parecchi PLC che interagiscono fra di loro, il software per ogni PLC sarà considerato come una Configuration separata.
- Una Configuration è in grado di comunicare con altre Configuration attraverso interfacce definite.

Modello Software dello standard IEC 1131-3

- Dentro ogni Configuration ci sono una o più Resource. Una Resource fornisce il supporto per tutte le caratteristiche necessarie per l'esecuzione del programma.
- Un programma IEC non potrà funzionare fin quando non è stato caricato in una Resource. Normalmente una Resource esiste dentro un PLC, ma può anche essere simulata dentro ad un personal computer.
- Lo standard permette ad una Configuration di contenere un certo numero di Resource e per ogni Resource più di un programma. Questo permette ad un PLC di mandare in esecuzione un numero di programmi totalmente indipendenti.
- L'esecuzione di differenti parti di un programma può essere controllata usando i task.

Riassumendo:

Configuration
Può essere inteso come il software relativo ad un PLC (In IEC 1131-5 sono definite le modalità di comunicazione tra piu' configuration –access paths-)

Resource
Puo' essere inteso come il software relativo ad una CPU

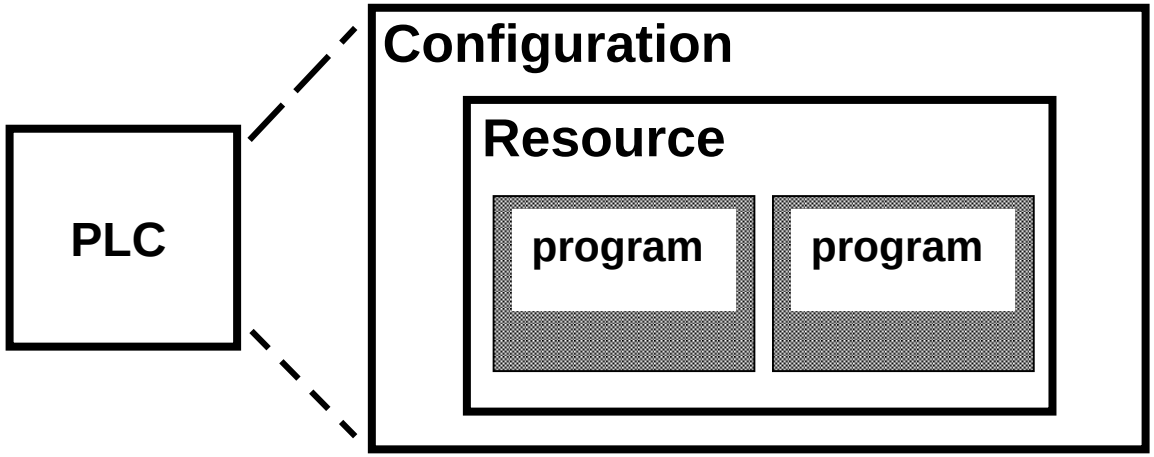
Task
"Motore" di attivazione di più parti di più POU di una resource

- esecuzione su evento o periodica (TIME) con priorità

Program (POU= Program Organisation Unit)
E' un modulo e cioè un insieme organizzato di:

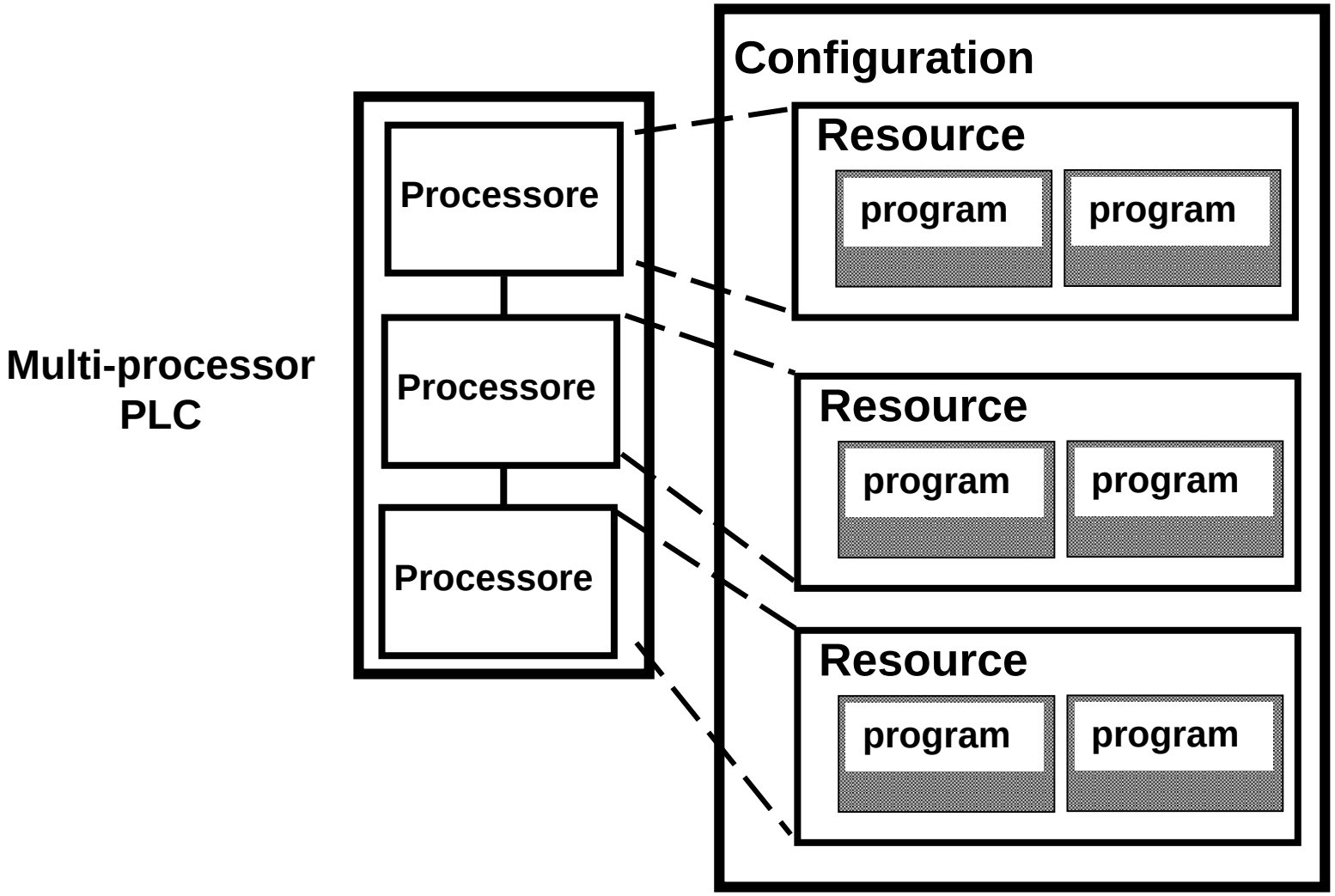
- Function (sistemi a più ingressi e una uscita privi di stati interni)
- Function block (procedure con parametri di ingresso e uscita e variabili locali – Es. PID,… -)

Il Modello Software e i Sistemi Reali

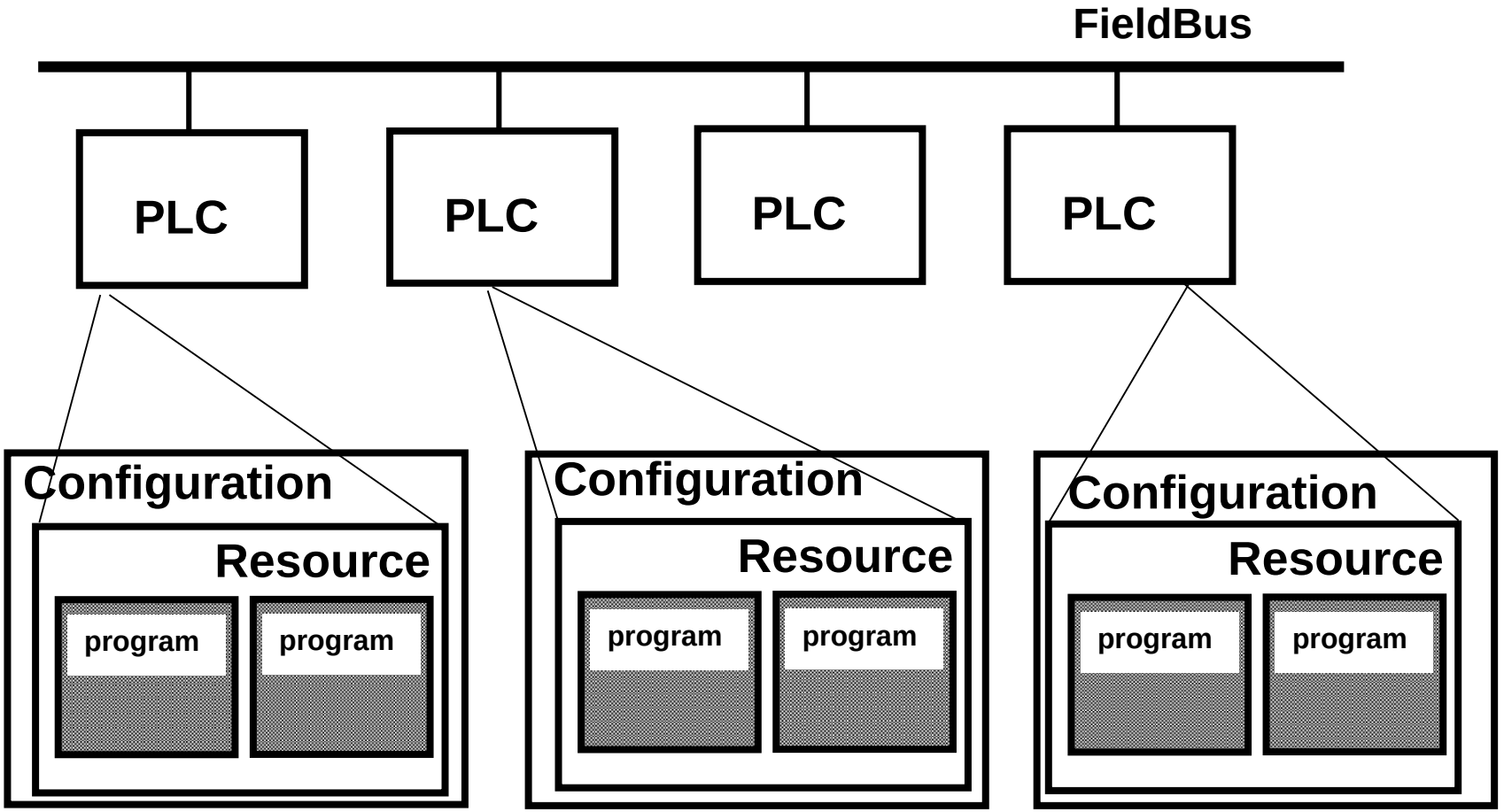


Se il processore non supporta il multi-tasking, il numero di programmi è unitario !

Il Modello Software e i Sistemi Reali



Il Modello Software e i Sistemi Reali

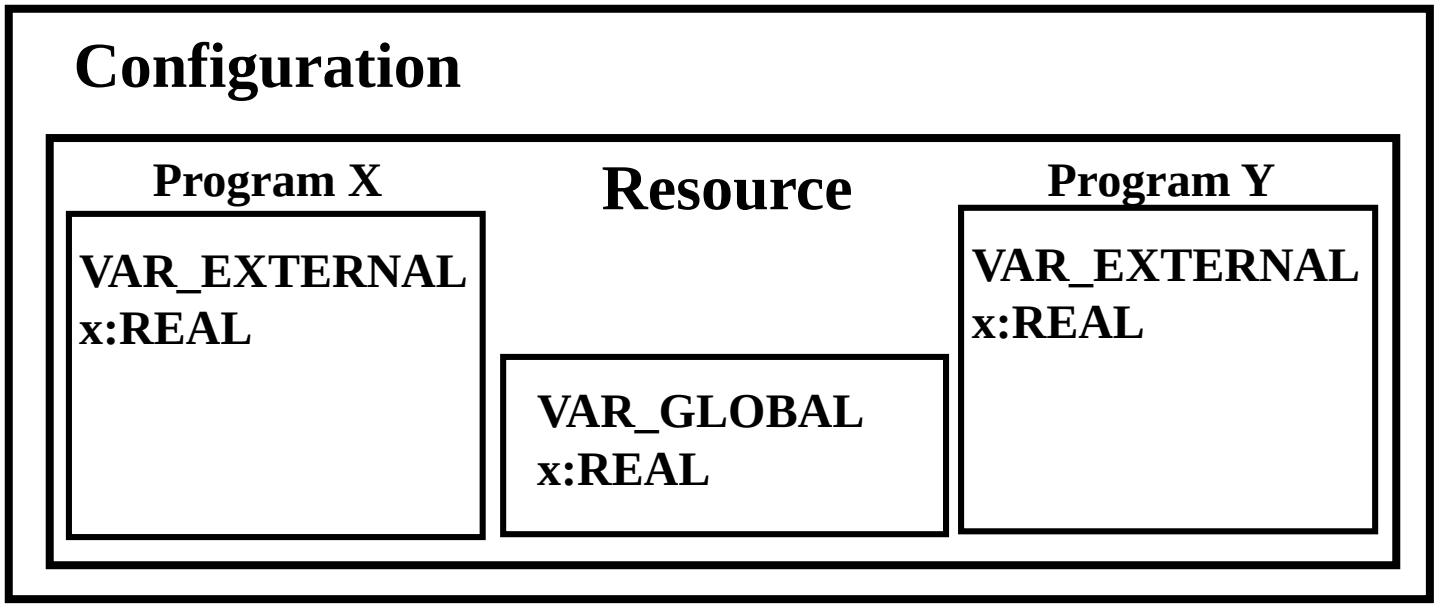


Riusabilità del software: Program Organization Units

- Program Organization Units (POUs):
  - Programma
  - Blocchi Funzioni (FB)
  - Funzioni (FC)
- La definizione di un POU permette il suo utilizzo un numero di volte illimitato:
  - Chiamate di funzioni
  - Istanze di Programma
  - Blocchi Funzioni (FB)
- Ciascuna istanza di un Programma o di un FB condivide lo stesso codice, ma ha la sua area privata di memoria
- Nell'ambito della stessa Configurazione è possibile utilizzare più istanze dello stesso programma in differenti Risorse
- Nell'ambito dello stesso Programma è possibile utilizzare più istanze dello stesso FB
- Non è ammessa alcuna ricorsione nelle POUs

Comunicazioni tra POUs

- Avviene tramite variabili globali definite a livello di:
  - Configuration
  - Resource
  - Program



Variable declarations

Keyword	Variable usage
VAR	Internal to organization unit
VAR_INPUT	Externally supplied, not modifiable within organization unit
VAR_OUTPUT	Supplied by organization unit to external entities
VAR_IN_OUT	Supplied by external entities. Can be modified within org. unit
VAR_EXTERNAL	Supplied by configuration via VAR_GLOBAL .Can be modified within organization unit
VAR_GLOBAL	Global variable declaration
VAR_ACCESS	Access path declaration
RETAIN	Retentive variables
CONSTANT	Constant
AT	Location assignment

Example

```
VAR
    CONDITION_RED : BOOL;
    IBOUNCE : WORD;
    MYDUB : DWORD;
    AWORD, BWORD, CWORD: INT;
    OKAY : STRING[10] := 'OK';
END_VAR
```

Data Types

such as:

BOOL, BYTE, WORD, DOUBLE WORD, QUAD WORD

INTEGER : SINT, INT, DINT, LINT, USINT, UINT, UDINT, ULINT

REAL, LREAL

DATE, TIME_OF_DAY, DATE_AND_TIME

STRING

e.g. DATE#2012-03-06 or D#2012-03-06 TOD#16:32:45.02

Derived Datatypes

Direct derivation from elementary types, e.g.: TYPE R : REAL ; END_TYPE

Enumerated data types, e.g.: TYPE ANALOG_SIGNAL_TYPE : (SINGLE, DIFFERENTIAL) ; END_TYPE

Subrange data types, e.g.:TYPE ANALOG_DATA : INT (-4095..4095) ; END_TYPE

Array data types, e.g.:TYPE ANALOG_16_INPUT_DATA : ARRAY [1..16] OF ANALOG_DATA ;
END_TYPE

www.plcopen.org

Directly Represented Variables

Prefix	Meaning	Default data type
I	Input location	
Q	Output location	
M	Memory location	
X	Single bit size	BOOL
None	Single bit size	BOOL
B	Byte (8 bits) size	BYTE
W	Word (16 bits) size	WORD
D	Double word (32 bits) size	DWORD
L	Long (quad) word (64 bits) size	LWORD

FUNCTIONS (FC) and FUNCTION BLOCKS (FB)

- Functions does not have internal states (no internal memory)
  - Standard functions (ADD, MUL, SQRT, …)
  - User-defined functions (defined by the user)

- Function Blocks have algorithm plus memory
  - Standard Function Blocks –like standard IC- (counters, timers, FF,…)
  - User-defined Function blocks -like ASIC-

- **The same functionality can be realized (and runs):**
  - **As a long sequence of statements (OB0)**
  - **With non parametric FC**
  - **With parametric (input and output) FC**
  - **With FB**

Function (FC)

* Standard numerical functions (one input)

  ABS, SQRT, LN, LOG, EXP, SIN, COS, TAN, ASIN, ACOS, ATAN

* Standard arithmetic functions (two inputs)

  ADD, MUL, SUB, DIV, MOD, EXPT, MOVE

* Standard bit string functions (one input): SHL, SHR, ROR, ROL

* Standard logic functions: AND, OR, XOR, NOT

* Standard selection functions: SEL (between 2), MUX (among N), MAX, MIN, LIMIT

* Standard comparison functions : GT, GE, EQ, LE, LT

* Character string functions: LEN(length), CONCAT, INSERT,...

* Other types of functions (e.g. Data conversion)

www.plcopen.org

Function (FC)

* There are libraries of additional functions

* Your own defined functions:

```
FUNCTION SIMPLE_FUN : REAL
        VAR_INPUT
                A, B      : REAL;
                C          : REAL := 1.0;
        END_VAR
SIMPLE_FUN := A*B/C;
END FUNCTION
```

www.plcopen.org

Function Block (FB)

* Standard Function Blocks include FF, Timers and Counters

* Your own defined Function Block can run all the functionalities of a FC

* In addition, your own defined Function Block can also contain timers and counters (static)

* Every time you recall your own defined Function Block, a new Instance data Block is automatically generated

www.plcopen.org

TASK (1)

Concetti fondamentali della schedulazione di processi:

- Un processo può trovarsi nello stato di pronto, di attesa o di esecuzione
- Un processo nello stato di pronto viene posto in esecuzione in base alla politica di scheduling del S.O.
- E' possibile assegnare una priorità ai processi in modo da aiutare il S.O. nella scelta del processo da porre in esecuzione tra i processi pronti

Task nello standard IEC 61131-3:

- Ha il compito di "risvegliare un processo" ponendolo nello stato di pronto
- Permette di assegnare differenti controlli sulla esecuzione di Programmi o di Function Blocks appartenenti alla stessa Resource

TASK (2)

- Ad ogni Programma e FB viene associato un task.
- Esistono tre tipi di tasks:
  - Cyclic tasks: sono attivati ad intervalli temporali e il programma è eseguito periodicamente (o ciclicamente)
  - System (or Error) tasks: sono attivati se un evento di sistema (errore di sistema) avviene durante l'esecuzione di un programma, ad esempio stack overflow
  - Event (or Interrupt) tasks: sono attivati all'occorrenza di certi eventi, ad esempio se una variabile ha raggiunto un certo valore o al sopraggiungere di un interrupt
- Un programma senza task associato ha la più bassa priorità e viene posto in stato di pronto appena termina

TASK (3)

La dichiarazione dei task è caratterizzata dai seguenti parametri:

- Task Sistema/Interrupt
  Definizione dell'evento (strettamente legato al PLC)
- Task Cyclic
  - Definizione dell'intervallo. Si noti che il task può essere eseguito anche dopo intervalli superiori all'intervallo specificato, in dipendenza del S.O.
  - WatchDog Time. Specifica l'intervallo temporale dopo il quale viene controllato se l'intervallo Periodico è stato superato o no. Si sceglie generalmente inferiore o uguale alla durata dell'intervallo.
- Priorità
  Viene assegnata una priorità al task, generalmente in ordine decrescente (0 la più alta). Si usa nella scelta tra processi pronti.

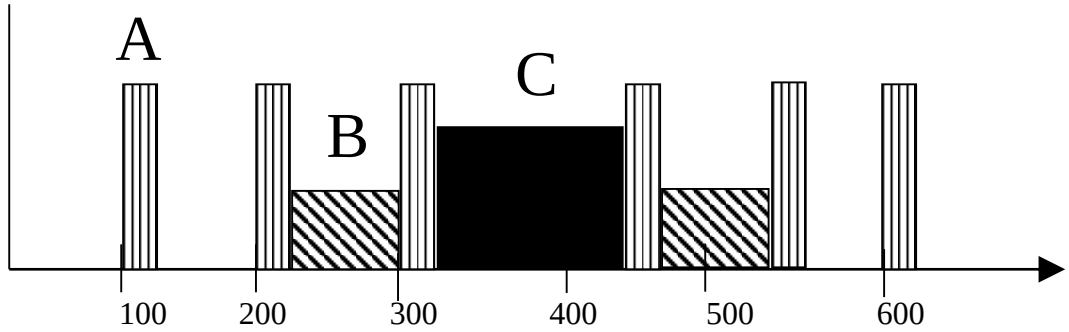
TASK (4)

I Task sono degli strumenti per controllare l'esecuzione dei processi, ma l'esecuzione dipende dal Sistema Operativo (preemptive, non-preemptive)

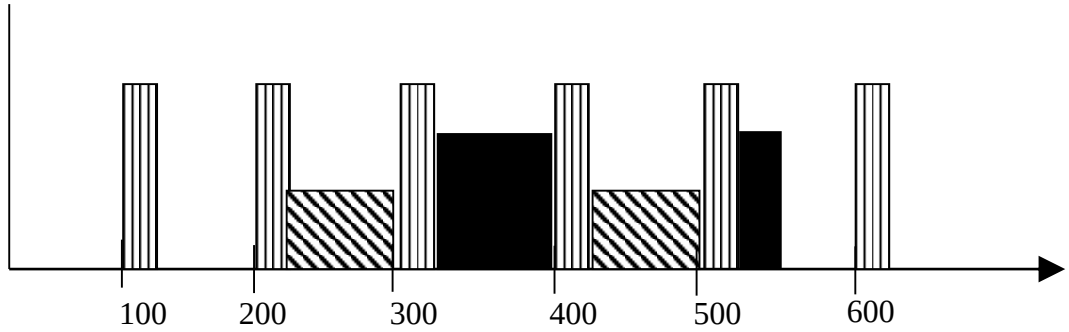
Esempio: tre tasks:

- Task A, Cyclic, Interval 100ms, priorità 0, durata 10
- Task B, Cyclic, Interval 200ms, priorità 1, durata 90
- Task C, Cyclic, Interval 300ms, priorità 2, durata 120

Non-preemptive schedule



Preemptive schedule



Elementi in Comune tra i 5 linguaggi IEC 61131-3

Identificatori

- Un identificatore può essere costituito da una sequenza di lettere e numeri purché siano soddisfatte le seguenti condizioni:
  - il primo carattere non sia un numero
  - non ci siano più di due caratteri "_" consecutivi
  - non vi siano spazi
- Lo standard impone che almeno i primi 6 caratteri debbano differenziare due identificatori

Keywords

- Lo standard definisce un set di keywords (ad esempio VAR, VAR_EXTERNAL, VAR_ACCESS). Si deve evitare l'uso di identificatori uguali alle keywords.

Commenti

- Vengono messi tra (* *)

Tipi di dati predefiniti

- Interi, Reali, Time, Date, String, Boolean

NOTA: i 5 linguaggi coesistono

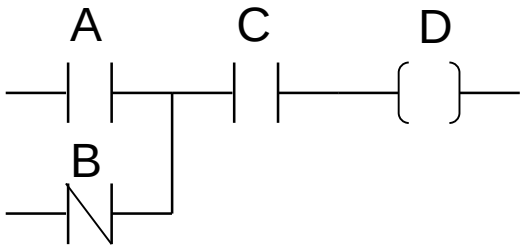
Linguaggi di programmazione tradizionali

Ladder diagram (schemi a contatti, KOP per Siemens)
• Orientato ai manutentori
• Richiama gli schemi funzionali a relais
• “Input operators followed by output operators”
Input operators: True? False? GE? LE? EQ?
Output operator: assign, sum, inc, convert, …

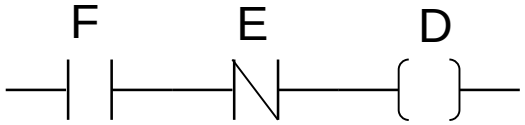
Instruction List (lista di istruzioni, AWL per Siemens)
• Orientato al personale informatico
• Pseudoassembler
• “Accumulator := operator & operand”

Function Block diagrams (Schemi funzionali, FUP per Siemens)
• Orientato al personale elettronico (è come realizzare una scheda partendo dai componenti e collegandoli tra loro)
• Segue lo standard ANSI/IEEE Std.91

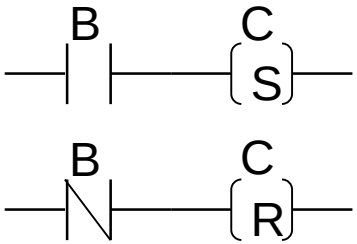
LADDER DIAGRAM: contacts and coils



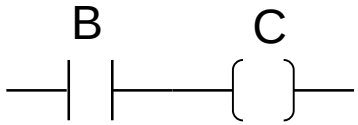
If (A+!B)&C then D:=1 else D:=0



If F&!E then D:=1 else D:=0
This instruction destroy effect of the previous



is equal to



…if B does not change between the two tests

In HW logics (relays), functions are executed in parallel, in a ladder program segments are executed in a sequence

LADDER DIAGRAM: RLC

OB, FC or FB are composed by segments

• Each segment is performed according to an accumulator

RLC (Result logic combinatorial)

• RLC is updated at every logic step of the segment according to elementary operations:

RLC:= RLC.operand.operator

Stack operators and Stack operands

• Temporary values and operators of RLC are stored in stacks

LADDER DIAGRAM: RLC-based operators

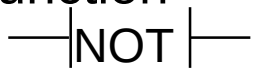
“Intermediate” operators (not input, not output)

Connectors

- RLC value can be stored in “connectors”
- Connector is not an output

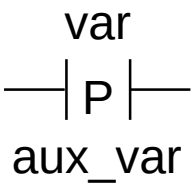
“NOT”

- NOT negates the RLC …that is NOT negates preceding function



Special operator

- Threshold (P = rising edge, N = falling edge)
- Aux_var is needed for old_value of var
- Aux_var is updated after execution of this instruction (one-time use)

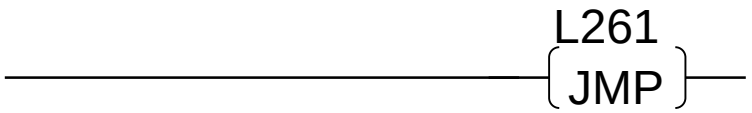


How to generate an “always 0” or “always 1” memory

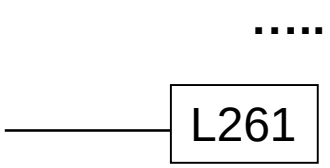
LADDER DIAGRAM: JUMP

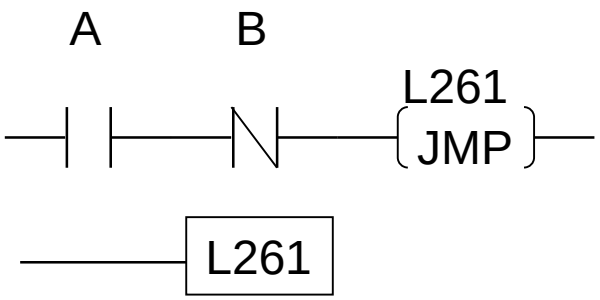
JUMPS (unconditioned or conditioned) are allowed in STEP7 both fourth and back

- Back JUMPS could “lock” OB1



Unconditioned (RLC-independent)





Conditioned (RLC-independent)
JMP if RLC=1
JMPN if RLC=0

LADDER DIAGRAM: DATA

- Bit, strings of bits (bit array), byte (8 bit), word (16 bit), Dword (32 bit)

- Logic operations

- Integer: short (S, 8 bit), normal (16 bit), Double (D, 32 bit), signed or unsigned (U)

- Real: simple $(-1)^s \cdot 1.m \cdot 2^e$ (32bit, s,8e,23m) and double precision (L, 64 bit, s,11e,52m)

- Math operations

- Time (32 bit, in ms), from T#-24d20h31m23s648ms to T#+24d20h31m23s647ms

- Timers operations

Memory Area

- I (input), Q (output), P (peripheral): bit, byte, word, Dword (no retention)
- M (merker), possible retention, variables, data blocks

LADDER DIAGRAM: SEQUENCES

Sequences can be implemented in ladder (one transaction only at a cycle)

1) Check transactions
• conditional transactions (if State=… and “condition” then assign Next_State=…)
• unconditional transactions (if “condition” then assign Next_State=…)
Note: if no transactions Next_state still remains the same

2) Execute transactions
• Assign Old_State <= State
• Assign State <= Next_state

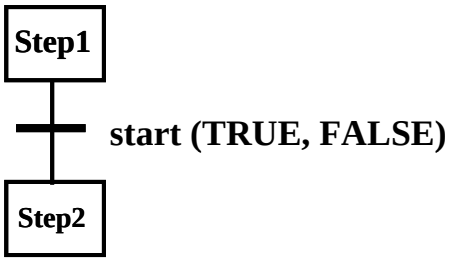
3) Execute actions within steps (stable actions within a state)
• if State=…then Action1, Action2,…

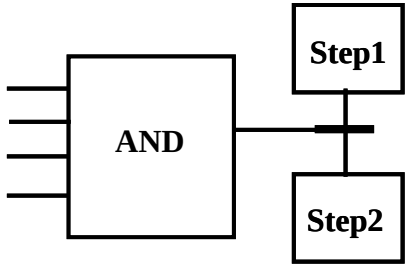
4) Execute actions within steps (actions on an edge)
• if State=… and Old_State<>State then Action1, Action2,…

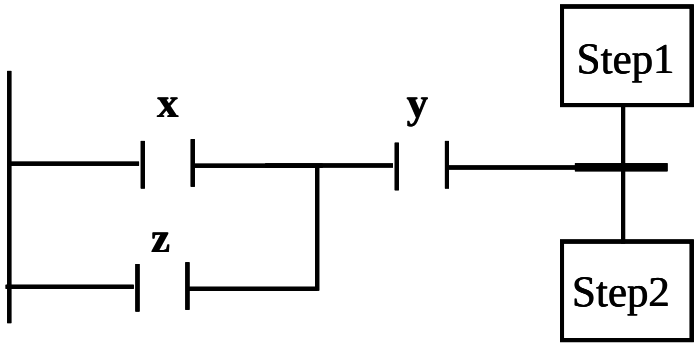
Suitable only for very simple state machines

Linguaggio Sequential Function Chart (SFC)

- Linguaggio orientato alle sequenze
- Deriva dallo standard IEC 848 basato su alcune idee tipiche del Grafcet
  - Linguaggio Grafico Grafcet: Standard Francese basato sulle Reti di Petri
- Vantaggi: Programmazione Top-Down
- I principali componenti sono: passi, transizioni e azioni
- Passi e transizioni sono collegati
- Possibili azioni connesse ai passi
- NO Action -> WAIT step
- Condizioni di transizione
- Adatto a descrivere FB e POU
- Non descrive FC







Un esempio di transizione

Linguaggio Structured text (ST)

Structured Text (ST)

- E' un linguaggio di programmazione ad alto livello
- E' strutturato a blocchi
- Simile al Pascal

- Supporta istruzioni e cicli
  - Loop iterativi (REPEAT-UNTIL; WHILE-DO)
  - Esecuzioni condizionali (IF-THEN-ELSE; CASE)
  - Funzioni (SQRT(), SIN())

Un esempio di macchina a stati

```
(* macchina a stati *)
TxtState := STATES[StateMachine];
CASE StateMachine OF 1:
ClosingValve();
ELSE ;;
BadCase();
END_CASE;
```