Técnicas de la automatización
(Cód. 201987)

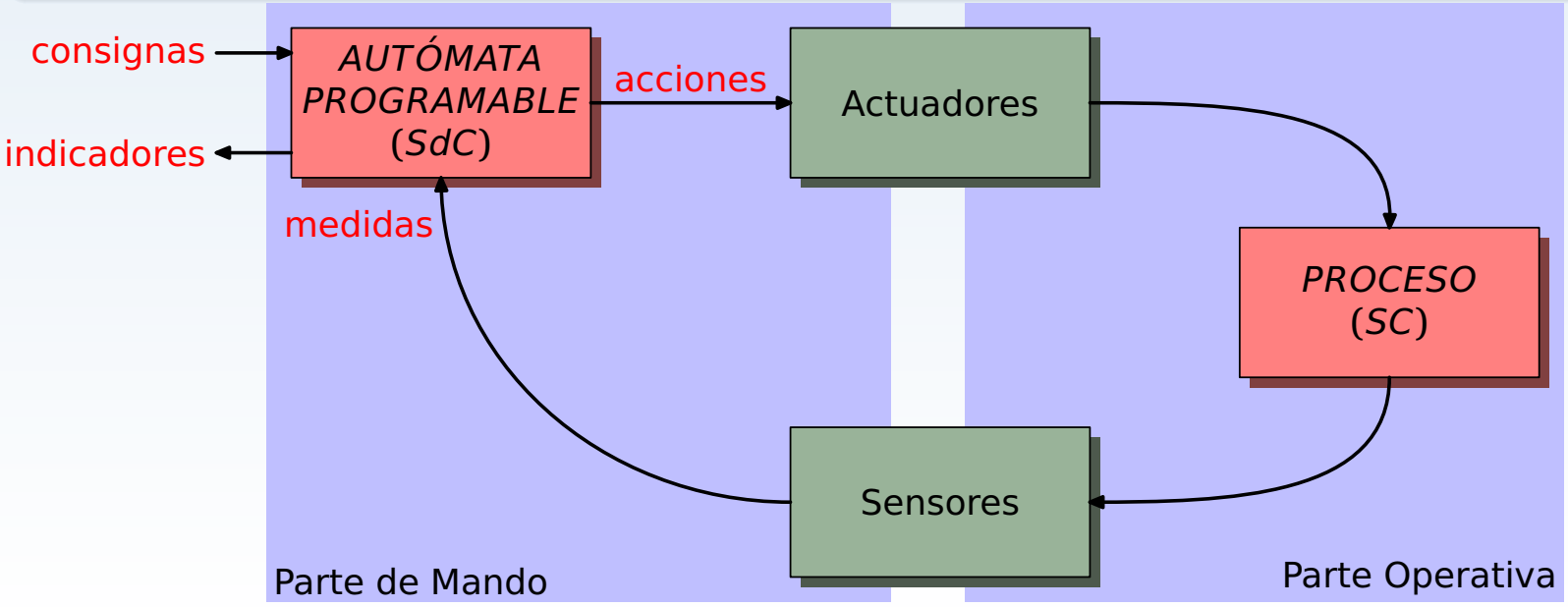
3. PLC-I: Controladores lógicos programables. Programación en lenguajes LD e IL

Escuela Politécnica Superior
UNIVERSIDAD DE ALCALÁ

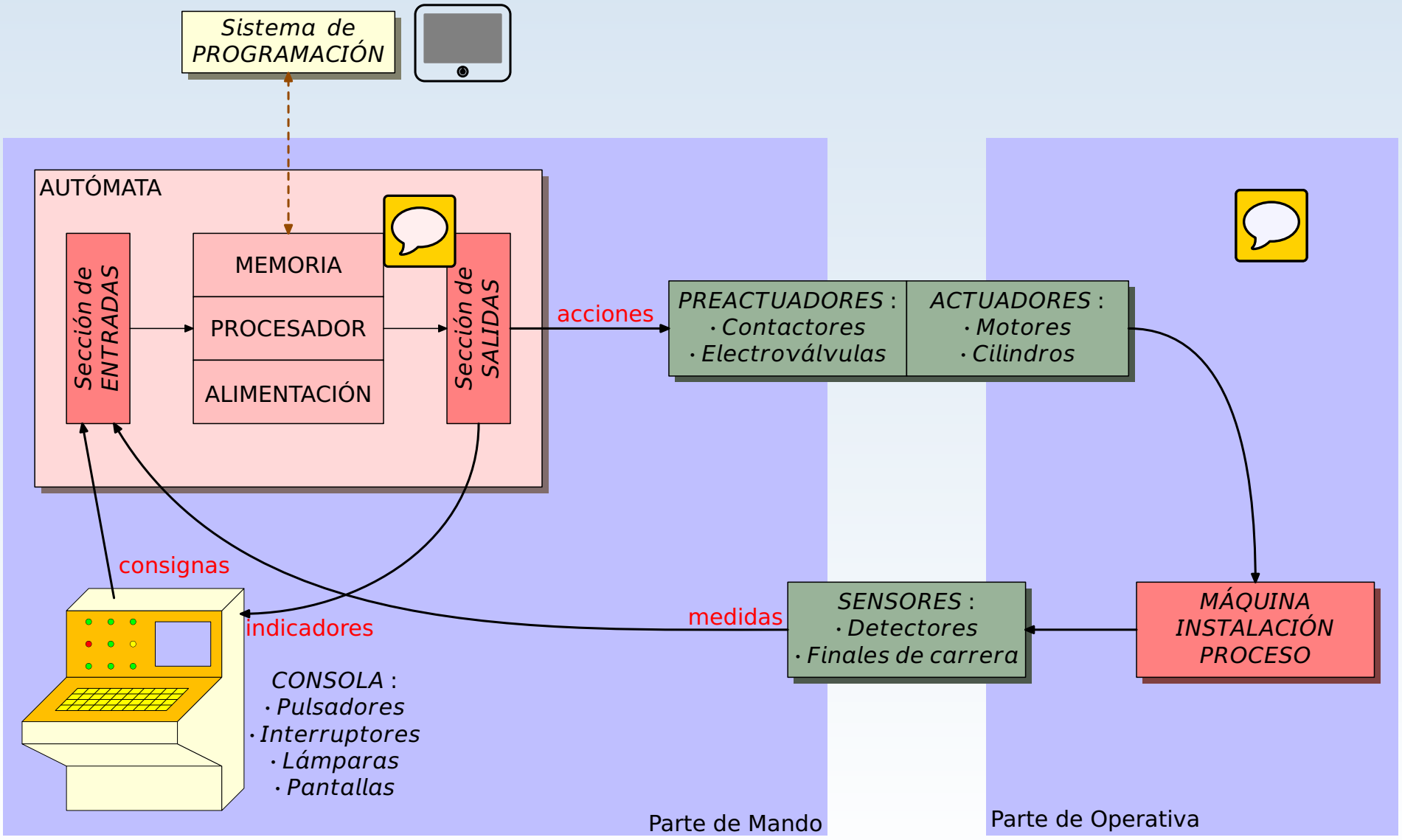
Índice

Autómatas programables

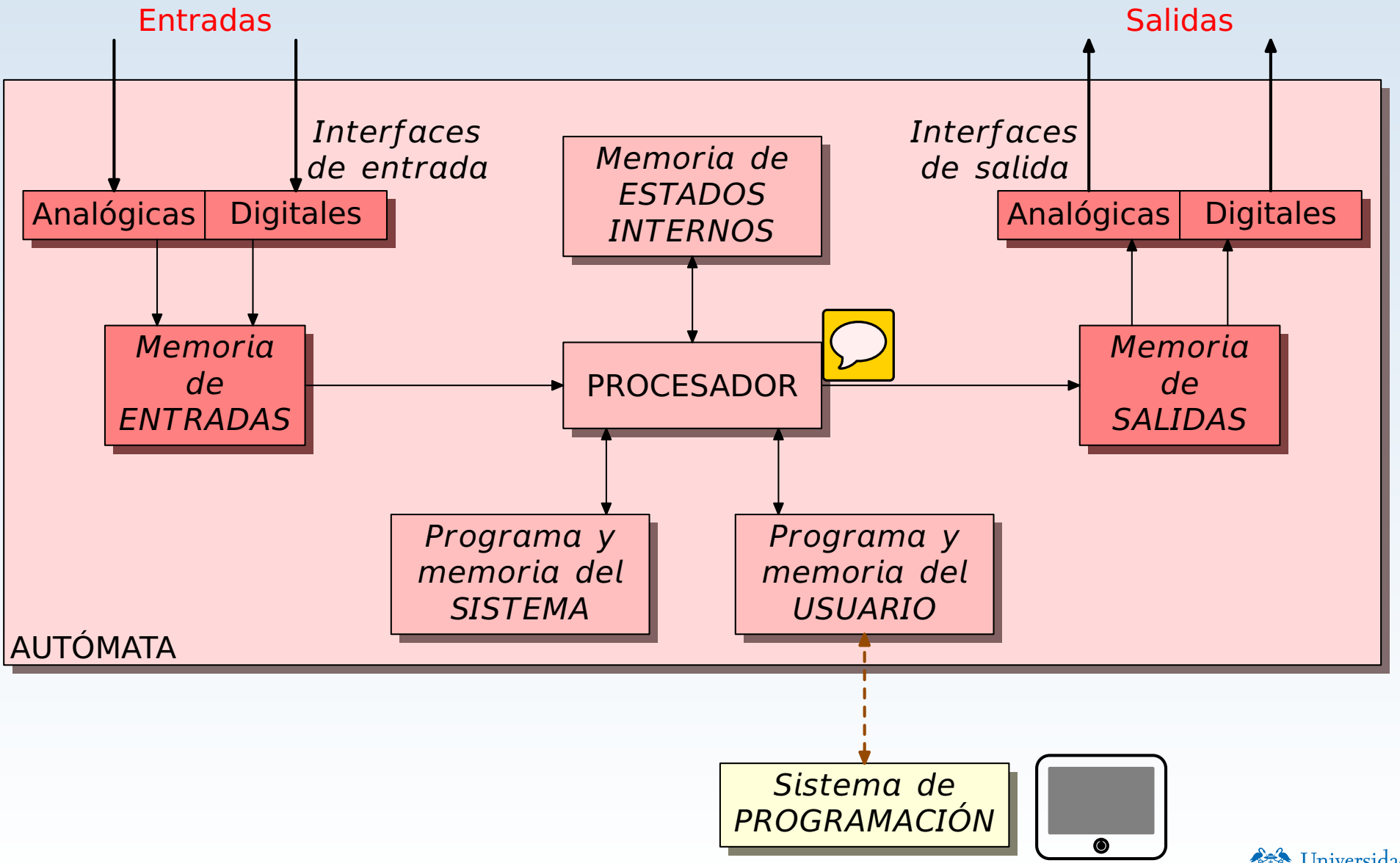
(IEC 61131) Un autómata programable (*PLC — Programmable Logic Controller*) es una máquina electrónica programable diseñada para ser utilizada en un entorno industrial, que utiliza una memoria programable para el almacenamiento interno de instrucciones orientadas al usuario, para implantar soluciones específicas tales como funciones lógicas, secuencias, temporizaciones, recuentos y funciones aritméticas, con el fin de controlar mediante entradas y salidas (digitales y analógicas) diversos tipos de máquinas o procesos.



Modelo estructural: PLC y entorno

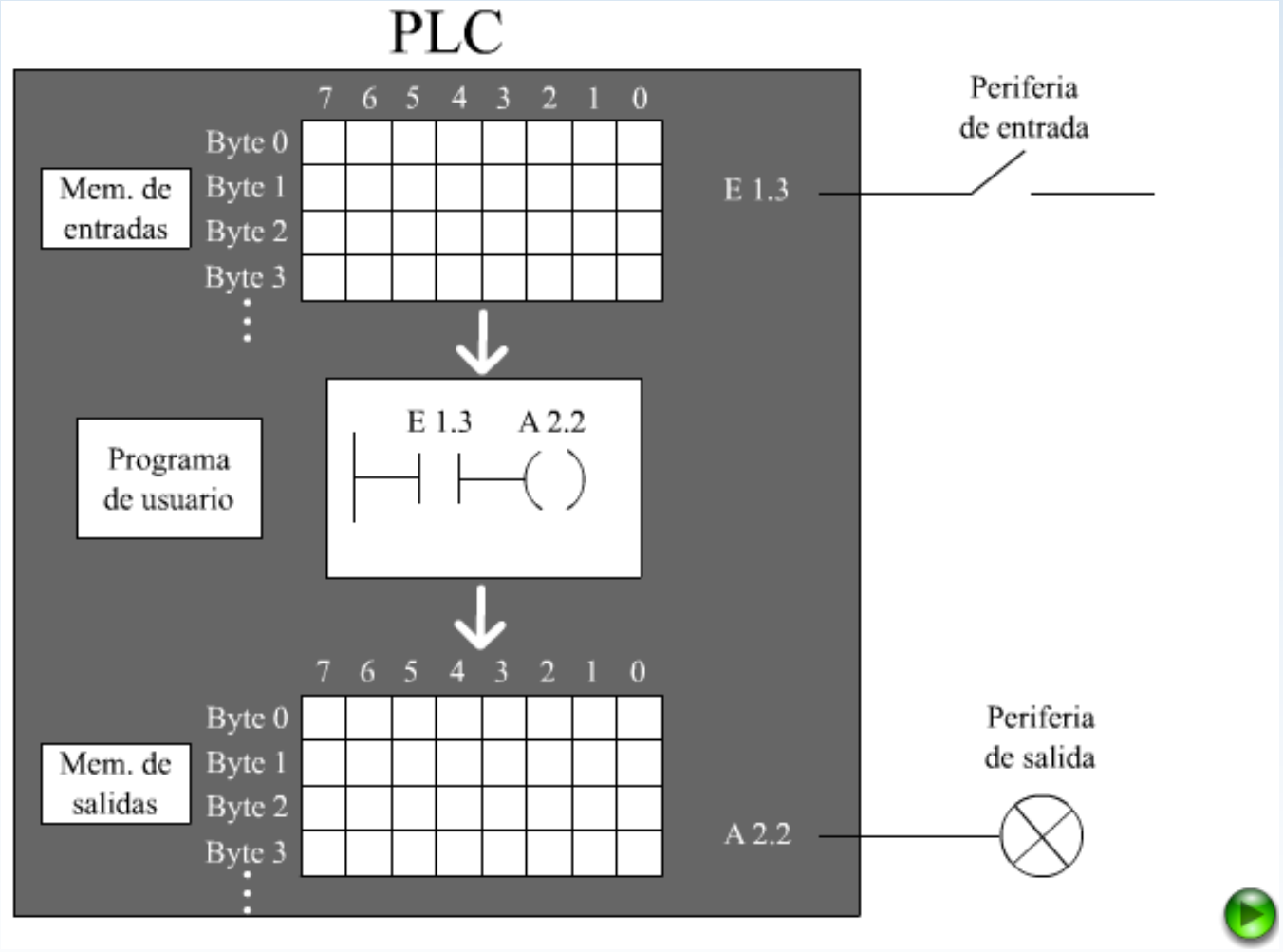


Modelo estructural: arquitectura de un PLC



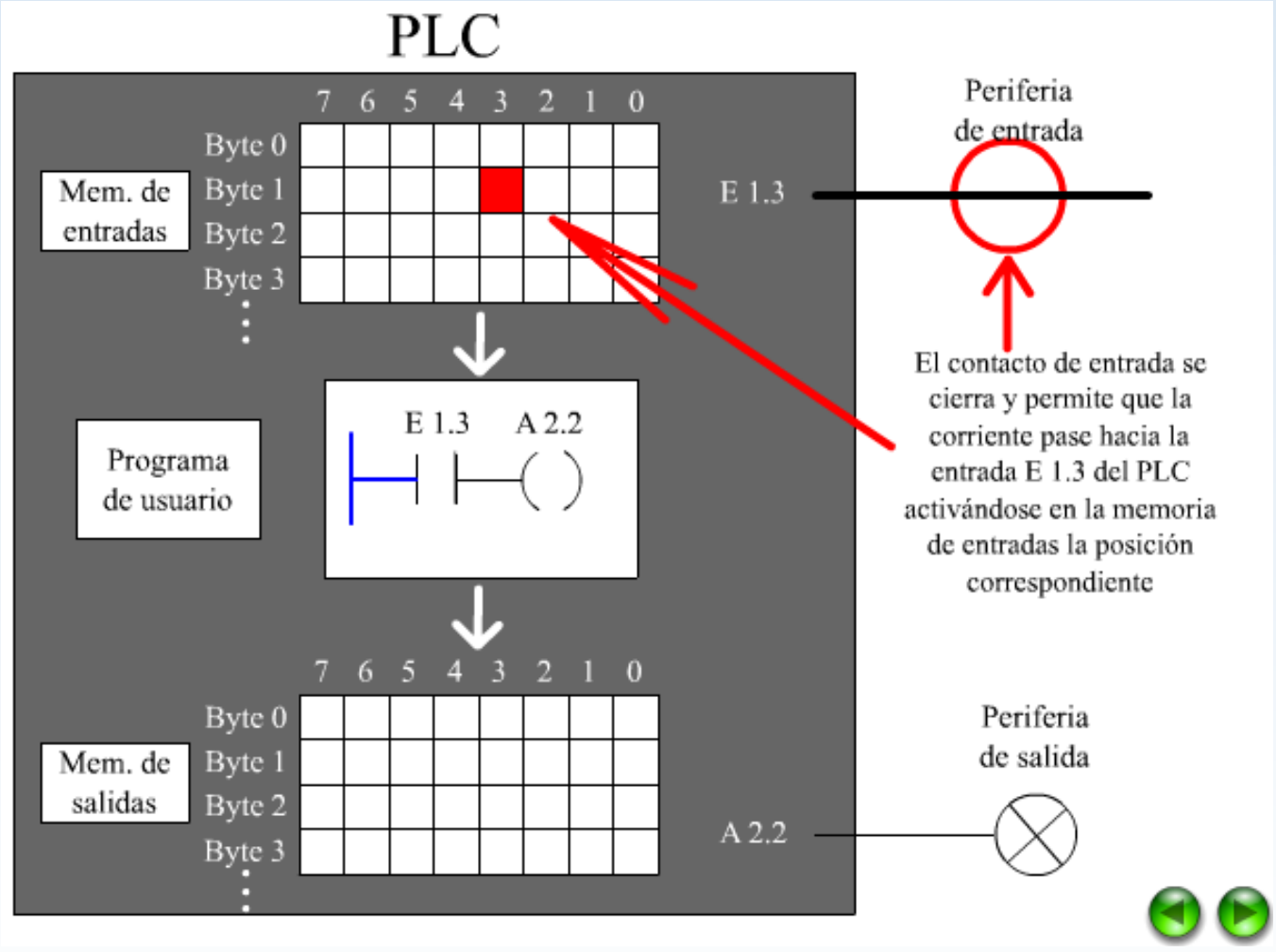
Modelo procesal: ciclo de SCAN (i)

El ciclo de SCAN es cada uno de los ciclos de ejecución de un programa:



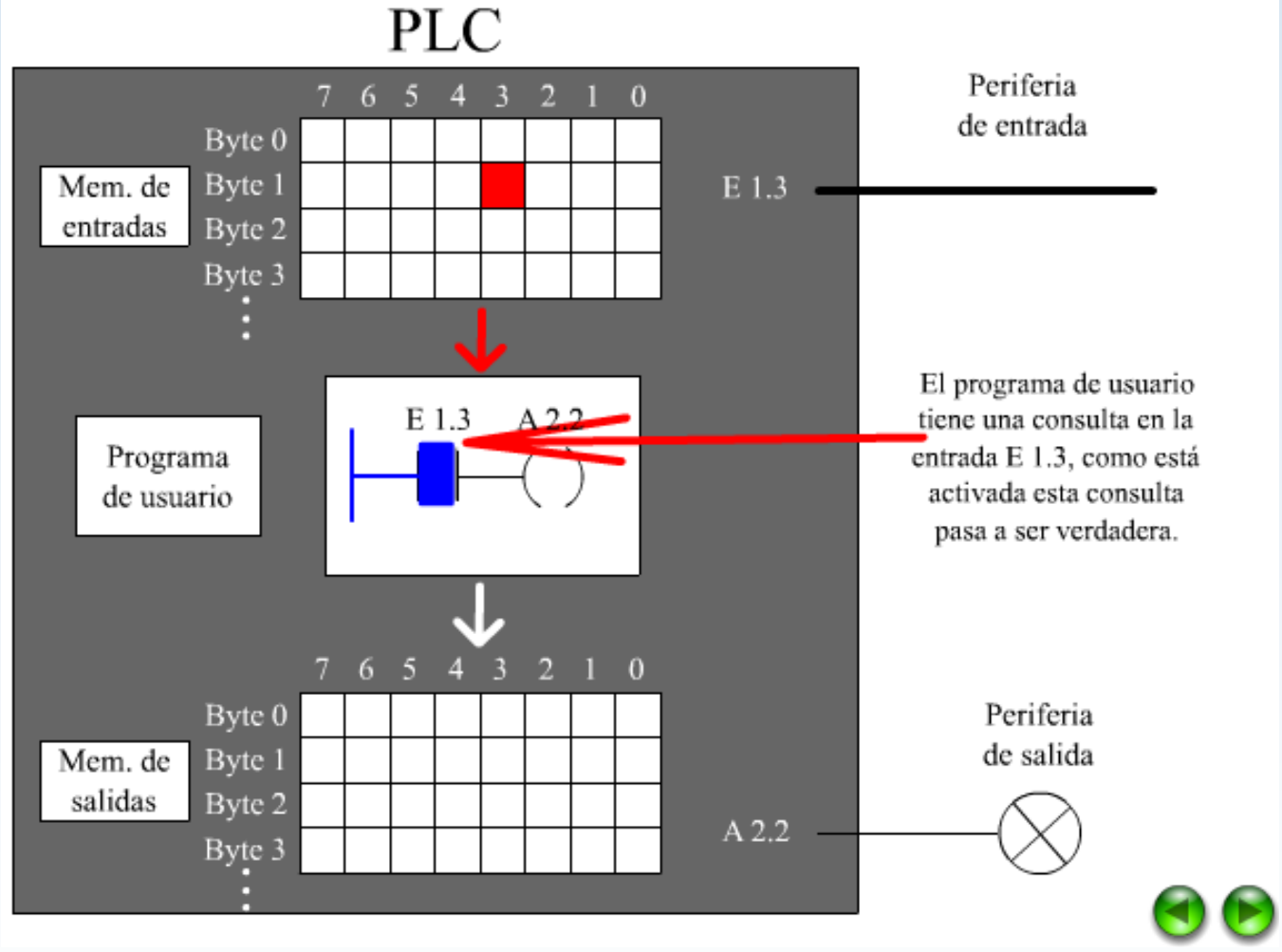
Modelo procesal: ciclo de SCAN (ii)

(a) Lectura de las entradas:



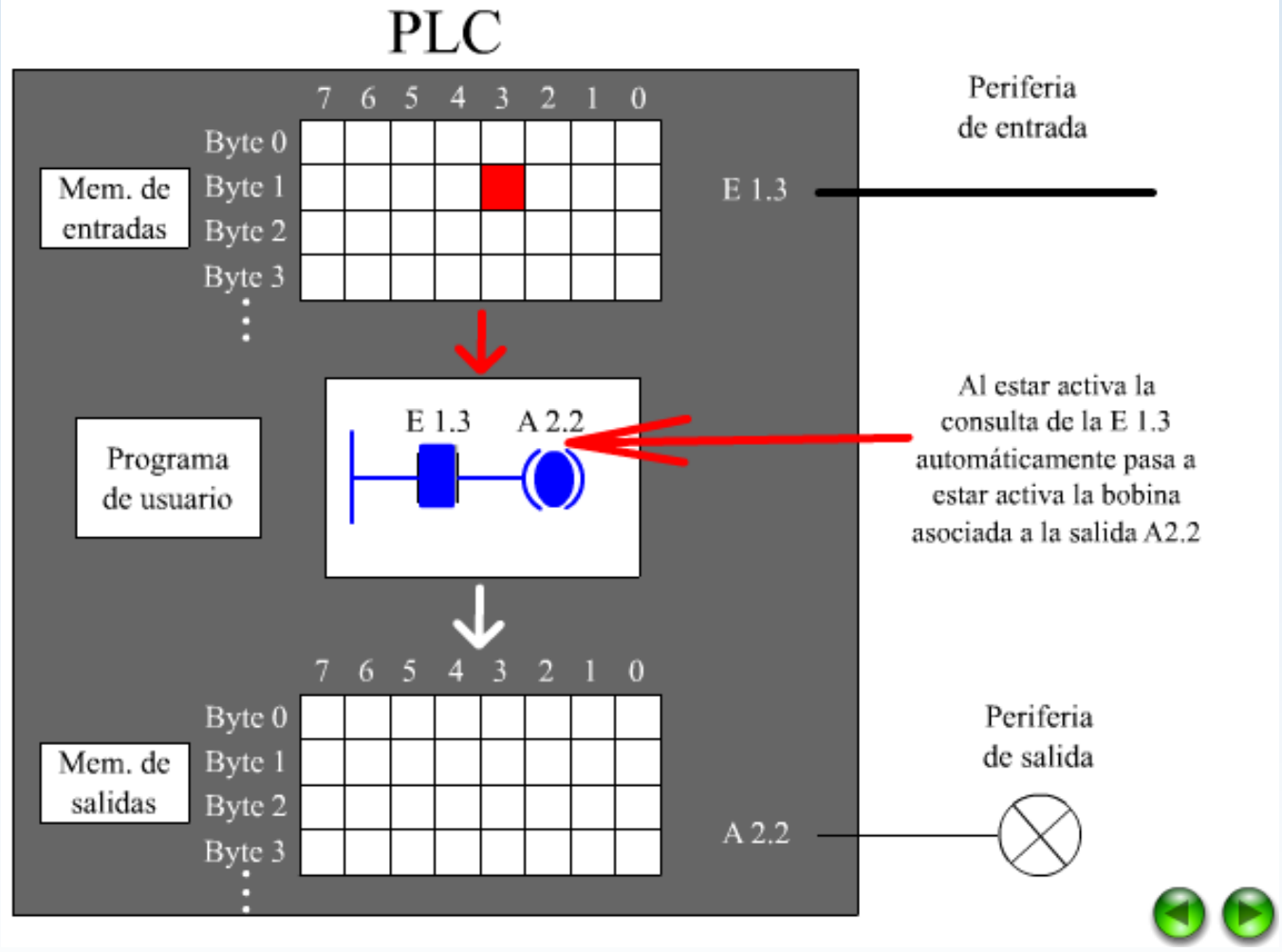
Modelo procesal: ciclo de SCAN (iii)

(b) Ejecución del programa:



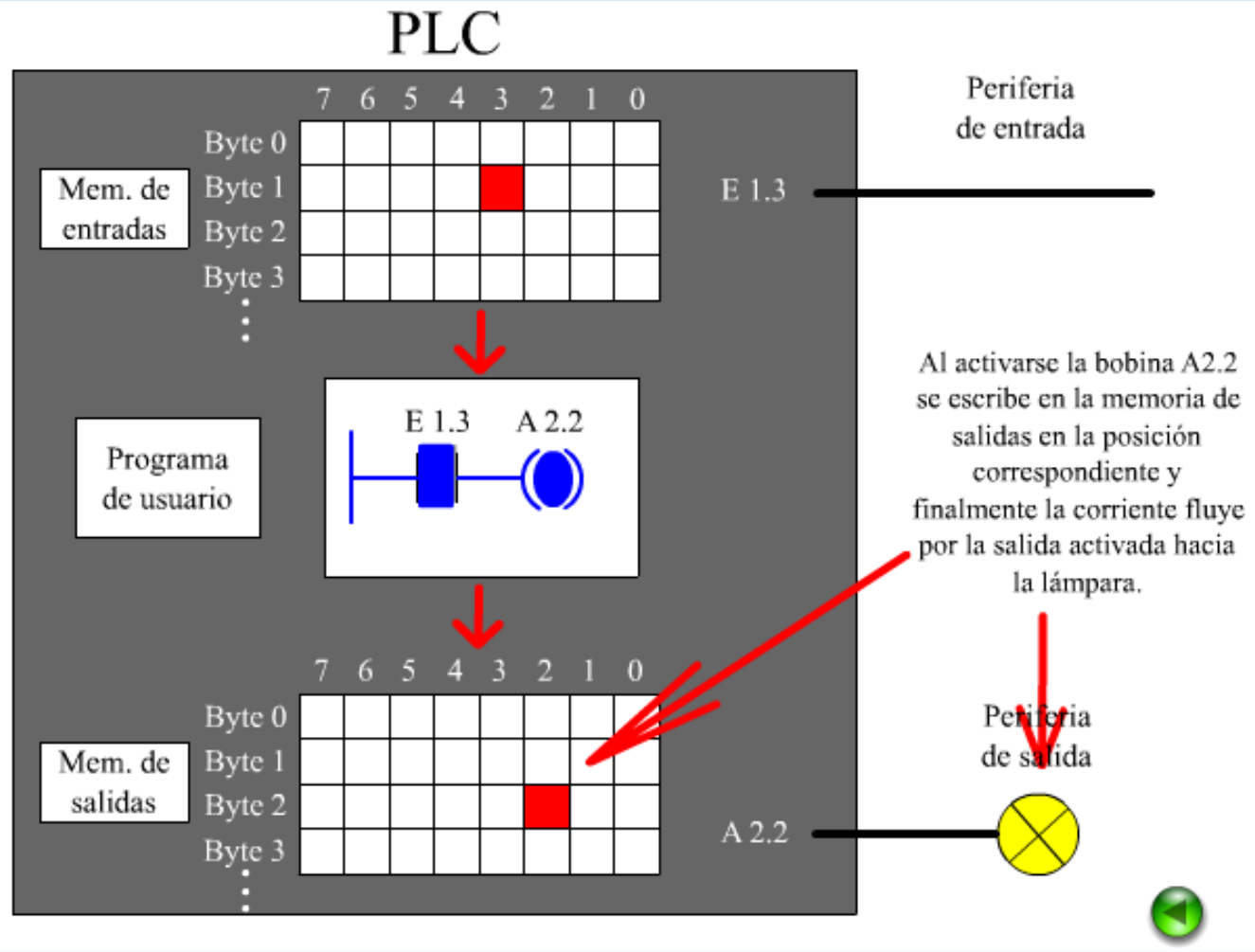
Modelo procesal: ciclo de SCAN (iv)

(b) Ejecución del programa:



Modelo procesal: ciclo de SCAN (v)

(c) Escritura de las salidas:



Índice

La norma IEC 61131

La norma IEC 61131 es el resultado del trabajo de varias multinacionales y especifica los requisitos de los sistemas PLC.

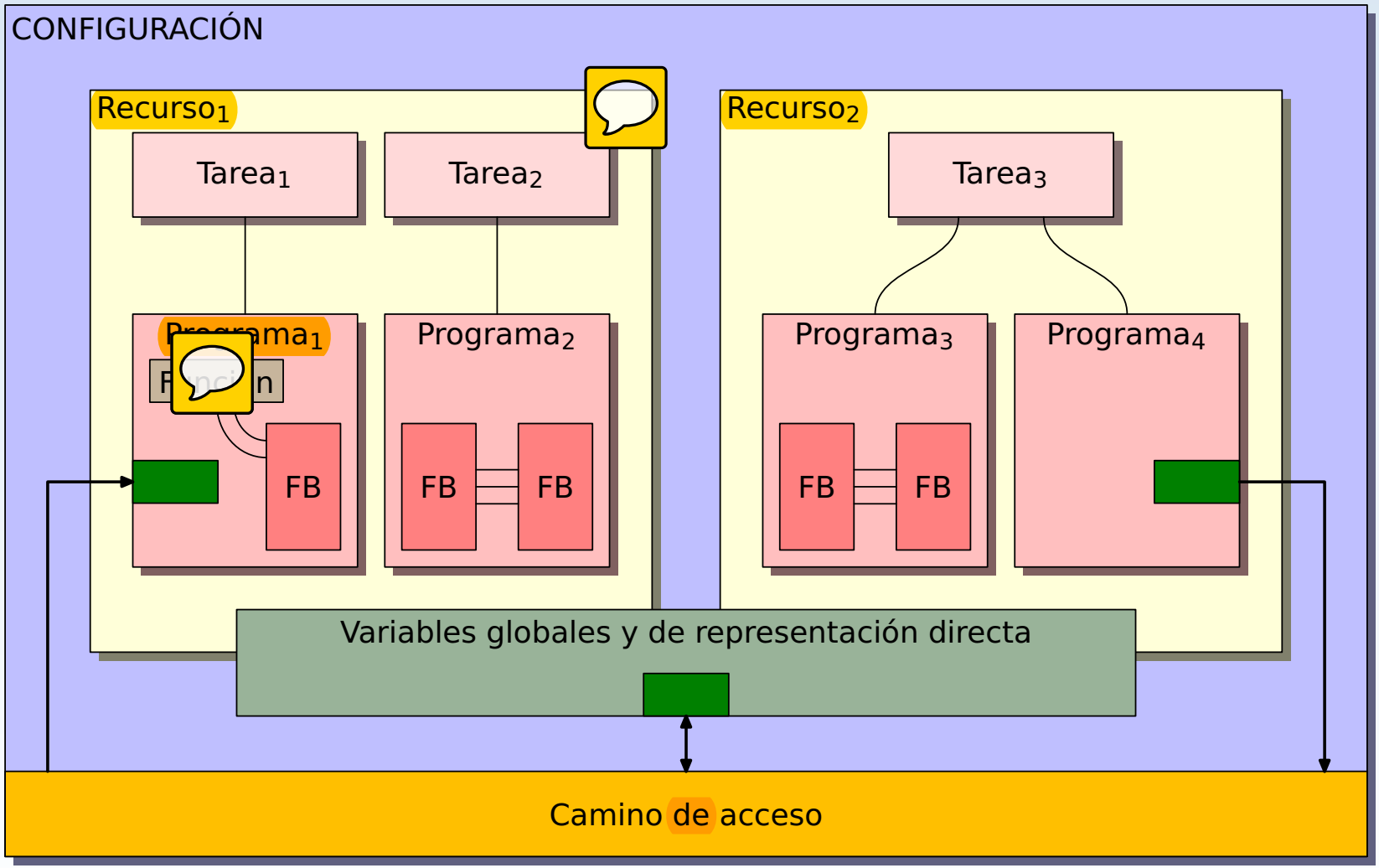
Partes del estándar

- IEC 61131-1 — Información general.
- IEC 61131-2 — Requisitos de equipos y pruebas (hardware).
- IEC 61131-3 — Lenguajes de programación (software) [IEC, 2006].
- IEC 61131-4 — Guías de usuario.
- IEC 61131-5 — Comunicaciones.
- IEC 61131-6 — Seguridad (en preparación).
- IEC 61131-7 — Programación en lógica difusa.
- IEC 61131-8 — Guías para la implementación y aplicación de los lenguajes de programación de autómatas programables.

Cumplir todos los aspectos de la norma IEC 61131 no es fácil, por eso se permiten implementaciones parciales.

El modelo de software IEC 61131-3

Una configuración es el elemento software de más alto nivel requerido para solucionar un problema de control.



Elementos de una configuración

Elementos de una configuración

- Definición de tipos.
- Declaración de variables globales (accesibles desde cualquier recurso).
- Declaración de recursos.
- Declaración de caminos de acceso (para comunicación entre configuraciones).

```
1 CONFIGURATION Configuración_1
2    VAR_GLOBAL ... END_VAR
3    RESOURCE Recurso_1 ON CPU_01 ... END_RESOURCE
4    RESOURCE Recurso_2 ON CPU_02 ... END_RESOURCE
5    ...
6    VAR_ACCESS ... END_VAR (* Camino de acceso *)
7 END_CONFIGURATION
```

Recursos

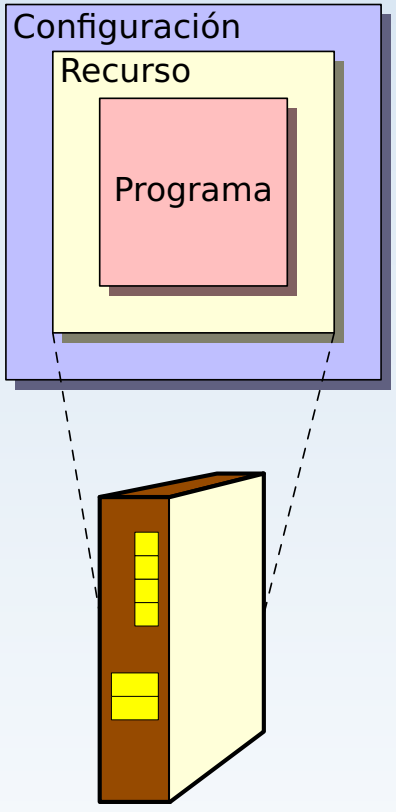
- Dentro de una configuración se pueden definir uno o más recursos.
- Se puede entender cada recurso como un procesador capaz de ejecutar programas IEC.

Elementos de un recurso

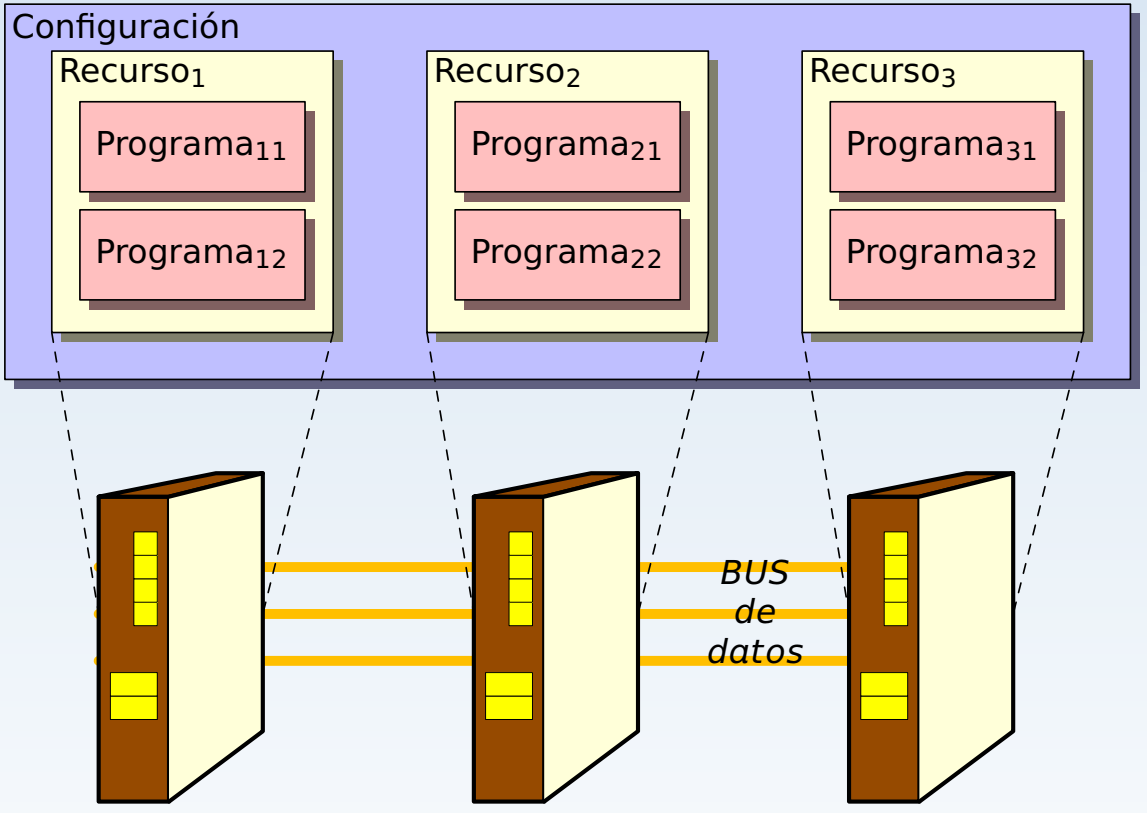
- Declaraciones globales.
- Declaración de tareas y sus programas asociados.

```
1 RESOURCE Recurso_1 ON CPU_01
2    VAR_GLOBAL ... END_VAR
3    TASK Tarea_1 ...
4    PROGRAM Programa_1 WITH Tarea_1 :...
5    TASK Tarea_2 ...
6    PROGRAM Programa_2 WITH Tarea_2 :...
7    ...
8 END_RESOURCE
```

Asignación de autómatas a recursos (i)

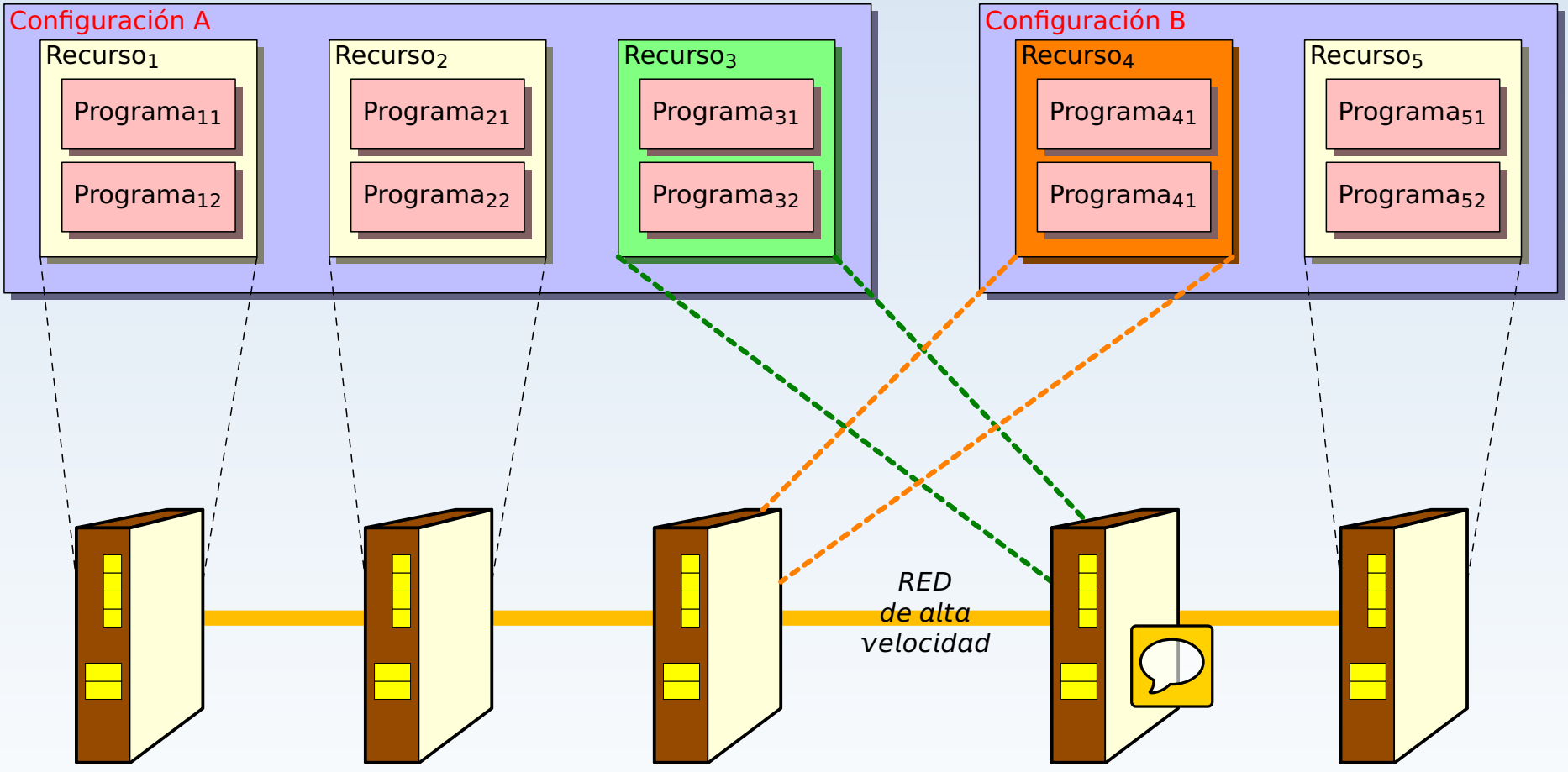


Sistema MONOPROCESADOR



Sistema de autómatas MULTIPROCESADOR

Asignación de autómatas a recursos (ii)



Sistema de autómatas DISTRIBUIDO

Tareas

Tareas

- En un recurso pueden estar definidas una o más tareas.
- Cada tarea controla la ejecución de un conjunto de programas y/o bloques de función.

Declaración de una tarea

```
1 TASK Nombre_Tarea (propiedades de la tarea);
2 PROGRAM Nombre_Programa WITH Nombre_Tarea: (interfaz del programa);
```

Propiedades de una tarea

- SINGLE:=V. Con cada flanco de subida de V la tarea ejecuta una vez el programa.
- INTERVAL:=t#T. La tarea ejecuta el programa con la periodicidad indicada en T.
- PRIORITY:=P. En procesadores que soporten multitarea, la tareas son despachadas según su prioridad.

Ejemplo

```
1 TASK T_Motor (INTERVAL:=t#8ms, PRIORITY:=3);
```

Índice

Programas y unidades organizativas de programas (POU's)

Un POU (*Program Organization Unit*) es la unidad de software independiente más pequeña de un programa de usuario.

Tipos de unidades organizativas

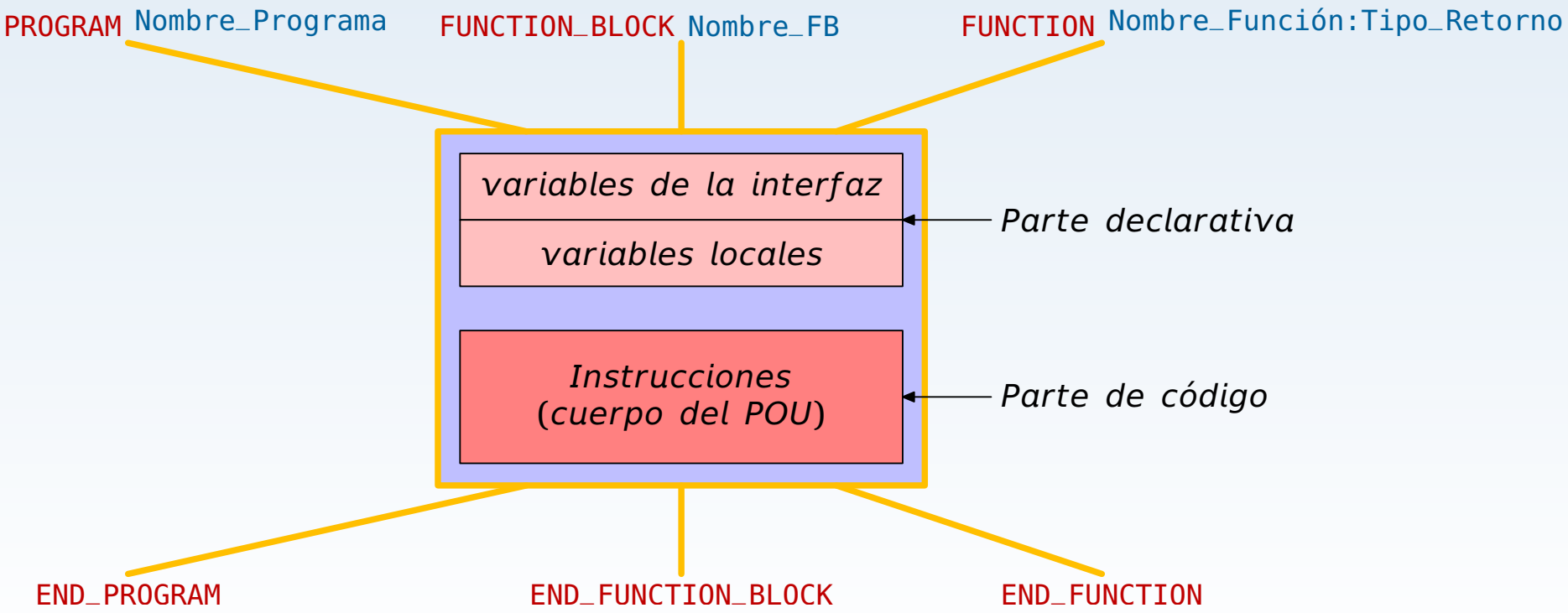
1. Función. Secuencia de instrucciones que trabajan sobre unos datos de entrada para producir una salida. Carece de memoria o estado interno.
2. Bloque de función. Estructura de datos encapsulados e independientes junto con los algoritmos que trabajan sobre esos datos. Puede tener un estado interno.
3. Programa. Secuencia de instrucciones, llamadas a funciones y llamadas a bloques de función para formar el ciclo principal de ejecución.

Com.— Los POU's no pueden contener llamadas recursivas.

Estructura de un POU

Partes de un POU

1. Tipo y nombre (y tipo de retorno en el caso de funciones)
2. Parte declarativa.
3. Parte de código (cuerpo con las instrucciones).



POU: parte declarativa – secciones de variables

En la parte declarativa se definen todas las variables que se utilizarán en la unidad.

Secciones de la parte declarativa

```
1  (* Variables de la interfaz *)
2  (* Interfaz de la llamada – Parámetros formales *)
3  VAR_INPUT ... END_VAR (* Parámetros de entrada *)
4  VAR_OUTPUT ... END_VAR (* Parámetros de salida *)
5  VAR_IN_OUT ... END_VAR (* Parámetros de entrada/salida *)
6  (* Interfaz global *)
7  VAR_EXTERNAL ... END_VAR (* Var. externas, declaradas en otros POU's *)
8  VAR_GLOBAL ... END_VAR (* Var. globales, accesibles para otros POU's *)
9  VAR_ACCESS ... END_VAR (* Var. en el camino de acceso de la »
           » configuración *)
10 (* Variables locales *)
11 VAR_TEMP ... END_VAR; (*Var. que no conservan valores entre llamadas.*)
12 VAR ... END_VAR;
```

POU: parte declarativa – atributos de las variables

Cada sección de var. puede tener alguno de los siguientes atributos

- CONSTANT: el valor asignado a la variable no cambia durante la ejecución del programa.
- RETAIN/NON_RETAIN:[a] la variable conserva/no conserva su valor tras un corte de alimentación.
- R_EDGE/F_EDGE:[b] la variable se activa[c] con un flanco de subida/bajada.
- READ_ONLY/READ_WRITE: la variable no puede/puede ser modificada.

[a] *retentiva/no retentiva.*
[b] *rising edge/falling edge.*
[c] Toma el valor TRUE.

```
1 VAR CONSTANT
2     Cte: BYTE := 16#FC;
3 END_VAR;
```

POU: parte declarativa – atributos permitidos

Sección/atrib. [1]	RETAIN NON_RETAIN	CONSTANT	R_EDGE F_EDGE	READ_ONLY READ_WRITE
VAR_INPUT	×		×	
VAR_OUTPUT	×			
VAR_IN_OUT				
VAR_EXTERNAL		×		
VAR_GLOBAL	×	×		
VAR_ACCES				×
VAR_TEMP		×	×	
VAR	×	×		

[1] [John, 2010, p. 94]

POU: parte declarativa – tipos de variables

En cada sección se pueden definir una o más variables:

```
Nombre_Variable: Tipo_Variable;
```

```
1 VAR_INPUT
2     Var_1:REAL;
3     Flag_1, Flag_2:BOOL;
4 END_VAR
```

Tipos elementales

Bits	Ent. con signo	Ent. sin signo	Coma flot.	otros
BOOL (1 bit)				TIME
BYTE (8)	SINT	USINT		DATE
WORD (16)	INT	UINT		STRING
DWORD (32)	DINT	UDINT	REAL	
LWORD (64)	LINT	ULINT	LREAL	

Direccionamiento directo

Una variable puede estar asociada a una dirección física:

```
1 VAR
2    Var_1 AT %IX2.10: BOOL; (* Bit (X) nro. 10 de entrada (I) del »
           » canal nro. 2. *)
3    Var_2 AT %QW7: WORD; (* Palabra (W) de salida (Q) nro. 7. *)
4 END_VAR
```

Códigos de direccionamiento directo:

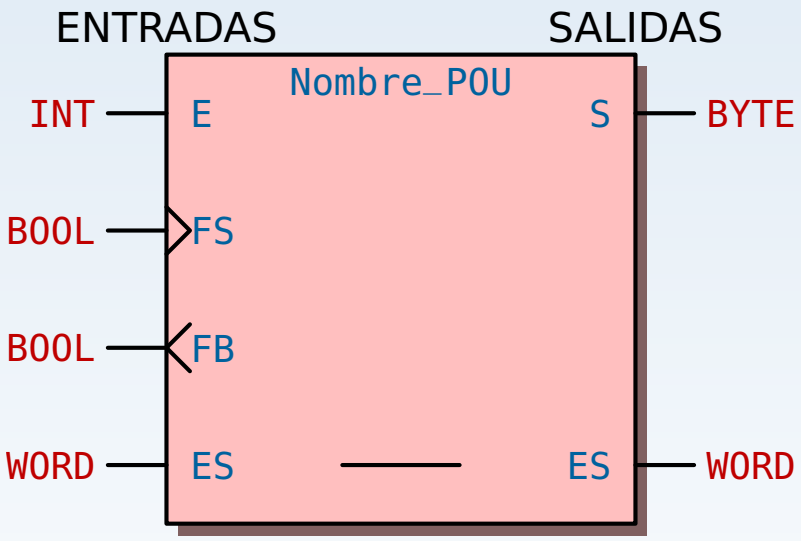
%	I (entrada)	X (opcional)	byte.bit[2]
	Q (salida)	B (byte)	
	M (relé/memoria)	W (palabra)	
		D (doble palabra)	
		L (cuádruple palabra)	

[2] Depende del fabricante.

POU: parte declarativa – representación gráfica

Los parámetros formales (interfaz de la llamada) de un POU se pueden representar gráficamente:

```
1  VAR_INPUT
2      E: INT;
3      FS: BOOL R_EDGE;
4      FB: BOOL F_EDGE;
5  END_VAR;
6  VAR_OUTPUT
7      S: BYTE; END_VAR;
8  VAR_IN_OUT
9      ES: WORD;
10 END_VAR;
```



POU: parte de código

La parte de código contiene las instrucciones que procesará el autómata.
Se programa con alguno de los siguientes lenguajes:

1. IL (*Instruction List*)[3], lista de instrucciones. Lenguaje de bajo nivel parecido al lenguaje ensamblador.
2. LD (*Ladder Diagram*)[4], diagrama de contactos. Lenguaje gráfico que trabaja con contactos y relés (variables lógicas) y está basado en los automatismos eléctricos.
3. ST (*Structured Text*), texto estructurado. Lenguaje de alto nivel parecido a Pascal o ADA.
4. FBD (*Function Block Diagram*)[5], diagrama de bloques de función. Lenguaje gráfico que trabaja con conexiones de funciones y bloques de función. Se asemeja a los diagramas con circuitos integrados.
5. SFC (*Sequential Function Chart*), diagrama funcional secuencial. Lenguaje gráfico basado en el Grafcet y las redes de Petri.

[3] AWL (*Anweisungsliste*) — Siemens.
[4] KOP (*Kontaktplan*).
[5] FUP (*Funktionsplan*).

Índice

Lenguaje IL (*Instruction List*)

- IL es un lenguaje de bajo nivel similar al lenguaje ensamblador. [6]
- A menudo se emplea como lenguaje intermedio al cuál son traducidos el resto de lenguajes gráficos o textuales.
- Un programa IL se compone de una secuencia de líneas de instrucción elementales.

Formato de una línea de instrucción

```
Etiqueta: Operador Operandos
```

1. `Etiqueta` (opcional). Se utiliza para nombrar esta línea en una instrucción de salto.
2. `Operador`. Puede ser el nombre de una instrucción o el de una función.
3. `Operandos`. Lista de operandos sobre los que actúa el operador.

[6] [IEC, 2006, pp. 123–129] y [John, 2010, pp. 100–115].

Operadores lógicos y de almacenamiento

CR - *Current Result*

- CR es un registro que gestiona una pila donde se almacenan los resultados de las instrucciones.
- Algunas instrucciones también usan CR como operando.
- Un operando puede ser una sublista de intrucciones entre paréntesis.

Instrucción	Significado	Instrucción	Significado
LD V	CR := V	LDN V	CR := $\overline{\text{V}}$
ST V	V := CR	STN V	V := $\overline{\text{CR}}$
AND V	CR := CR · V	ANDN V	CR := CR · $\overline{\text{V}}$
OR V	CR := CR + V	ORN V	CR := CR + $\overline{\text{V}}$
XOR V	CR := CR $\oplus$ V	XORN V	CR := CR $\oplus$ $\overline{\text{V}}$
NOT V	CR := $\overline{\text{V}}$		
S V	V := 1,[7] si CR = 1	R V	V := 0,[8] si CR = 1

[7] Set.

[8] Reset.

Ejemplos de listas de instrucciones

$S := V_1 \cdot \overline{V_2} + \overline{V_3} \cdot V_4 + V_5 \cdot V_6$

```
1 LD V1
2 ANDN V2
3 OR( NOT V3
4 AND V4
5 )
6 OR( V5
7 AND V6
8 )
9 ST S
```

$S := (V_1 + V_2) \cdot (\overline{V_3} + V_4) \cdot (\overline{V_5} + V_6)$

```
1 LD V1
2 OR V2
3 AND( NOT V3
4 OR V4
5 )
6 AND( NOT V5
7 OR V6
8 )
9 ST S
```

Otros operadores

Aritméticos			
Instrucción	Significado	Instrucción	Significado
`ADD V`	CR := CR + V	`SUB V`	CR := CR − V
`MUL V`	CR := CR × V	`DIV V`	CR := CR/V
`MOD V`	CR := CR mód V		

Comparación			
`GT V`	CR := (CR > V)	`LT V`	CR := (CR < V)
`GE V`	CR := (CR ⩾ V)	`LE V`	CR := (CR ⩽ V)
`EQ V`	CR := (CR = V)	`NE V`	CR := (CR ≠ V)

Control de secuencia			
`JMP etq`	salta a la línea `etq:`		
`JMPC etq`	salta si CR = 1	`JMPCN etq`	salta si CR = 0
`CAL fb`	llamada a `fb`		
`CALC fb`	llamada si CR = 1	`CALCN fb`	llamada si CR = 0
`RET`	retorna		
`RETC`	retorna si CR = 1	`RETCN`	retorna si CR = 0

Llamadas a funciones

- Supongamos que la función `f` tiene los parámetros formales $\texttt{pf}_1$, $\texttt{pf}_2$ y $\texttt{pf}_3$.
- Tres de las posibles formas de hacer la llamada $\texttt{res} := \texttt{f}(\texttt{d}_1, \texttt{d}_2, \texttt{d}_3)$ son:

Notación posicional

```
1 LD d1
2 f d2, d3
3 ST res
```

Notación nombrada

```
1 f(
2   pf1:=d1,
3   pf2:=d2,
4   pf3:=d3
5 )
6 ST res
```

Orden cambiado

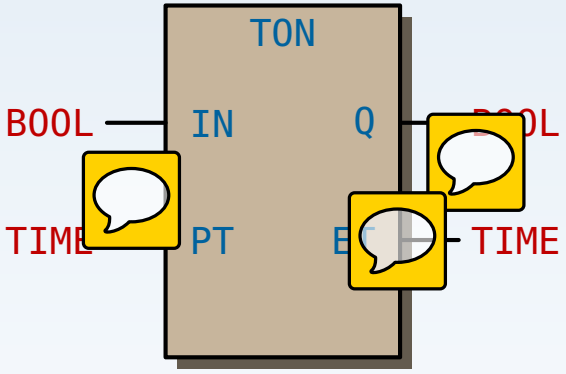
```
1 f(
2   pf2:=d2,
3   pf3:=d3,
4   pf1:=d1
5 )
6 ST res
```

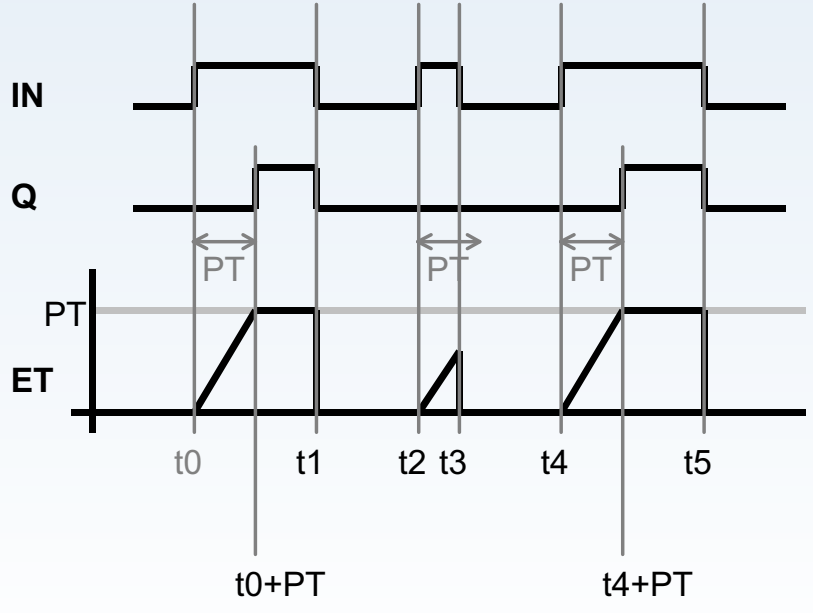
Llamada a bloques de función

La norma IEC 61131-3 describe tres formas

1. Llamar al bloque con una lista de parámetros actuales entre paréntesis.
2. Cargando los parámetros formales antes de llamar al bloque.
3. Llamando implícitamente usando los parámetros formales como operadores (solo es válido para bloques de función estándar).

Ejemplo: temporizador con retardo a la conexión `TON`:





1. `IN`↑⇒`ET:=0` y empieza a contar tiempo.
2. `ET` alcanza `PT`⇒`Q:=TRUE`.
3. `IN`↓⇒`Q:=FALSE` y `ET:=0`.

Utilización del temporizador TON

```
1 Activa, Salida: BOOL := 0;
2 Temp: TON; (* Ejemplar de TON. *)
3 Valor: TIME;
```

Método 1

```
1 CAL Temp(
2    IN:=Activa,
3    PT:=t#500ms,
4    Q=>Salida,
5    ET=>Valor
6 )
```

Método 2

```
1 (* Carga de par. *)
2 LD t#500ms
3 ST Temp.PT
4 LD Activa
5 ST Temp.IN
6 (* Llamada. *)
7 CAL Temp
```

Utilización de los parámetros de salida:

```
1 LD Temp.Q
2 ST Salida
3 LD Temp.ET
4 ST Valor
```

Índice

Partes de un diagrama de contactos

- El lenguaje LD procede del campo de los automatismos eléctricos. [9]
- Fue diseñado para procesar, principalmente, señales lógicas:

$$1 \equiv \texttt{TRUE} \equiv \texttt{ON}, \quad 0 \equiv \texttt{FALSE} \equiv \texttt{OFF}$$

- La parte de código de un POU se divide en segmentos. [10]
- El programa describe el «flujo de potencia» de izquierda a derecha a través de los segmentos del POU.

Partes de un segmento

- Etiqueta. Utilizada por las instrucciones de salto.
- Comentario. `(* Ejemplo de comentario. *)`
- Gráfico del segmento. Contactos, relés, funciones y bloques de función conectados entre sí entre dos «rieles lógicos».

[9] [IEC, 2006, pp. 139–142] y [John, 2010, pp. 147–168].

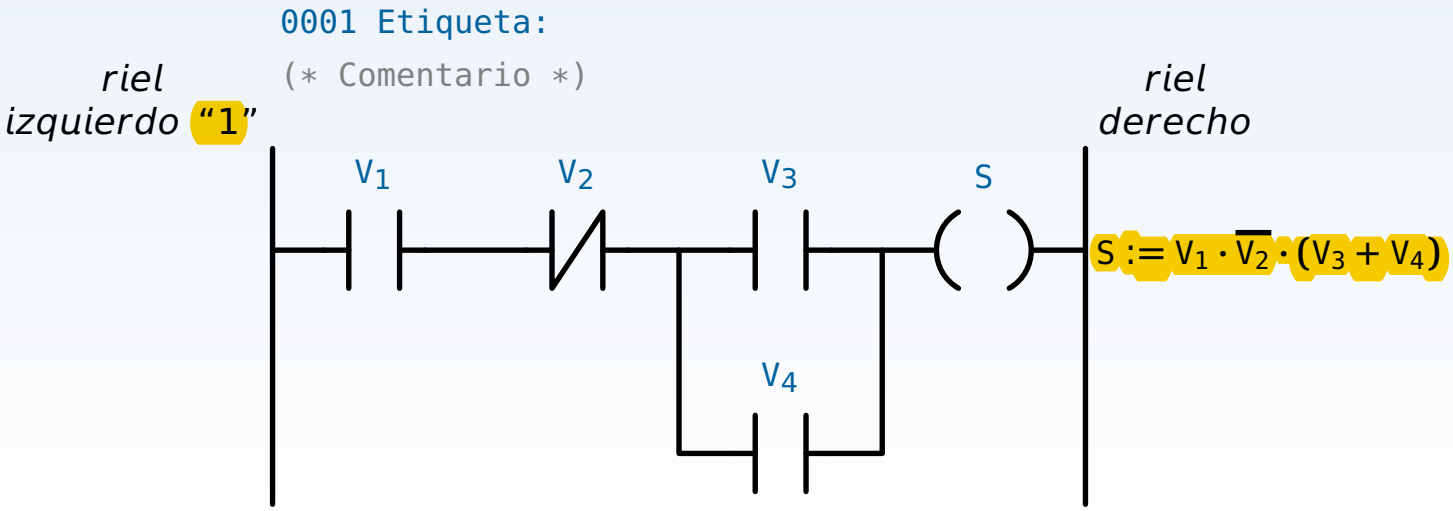
[10] En la norma se denomina *network* a cada segmento.

Distribución de los elementos gráficos

Los elementos gráficos de un segmento se sitúan entre dos rieles.

Rieles lógicos

1. Hay uno a la izquierda y otro a la derecha.
2. El riel de la izquierda está «alimentado» con el valor lógico 1.
3. Este valor «fluye» por el segmento dependiendo del estado de los elementos que encuentra en él.
4. Junto al riel derecho se conectan los elementos que almacenan resultados (los relés).

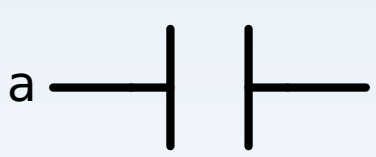
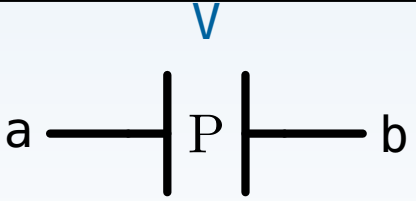
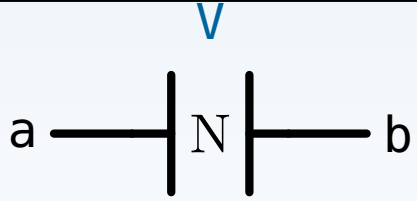


Objetos gráficos de un segmento

Los segmentos de programa se forman conectando entre sí objetos de los siguientes tipos:

- Contactos y relés (se asocian con variables).
- Control de secuencia (saltos).
- Ejemplares de funciones y bloques de función (llamadas a estos POU's).

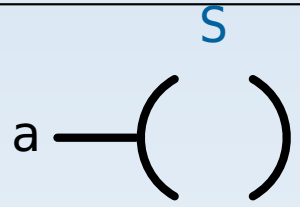
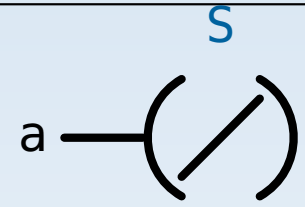
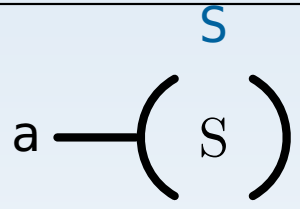
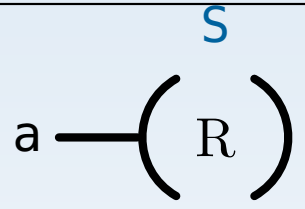
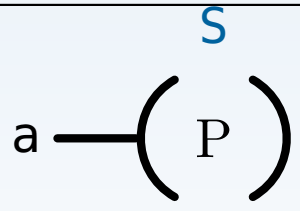
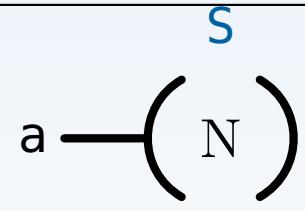
Tipos de contactos:

	$b := a \cdot V$		$b := a \cdot \overline{V}$
	$b := a \cdot V^{\lrcorner 11}$		$b := a \cdot V^{\urcorner 12}$

[11] $V^{\lrcorner}$(flanco de subida) $:= V^k \cdot \overline{V^{k-1}}$.

[12] $V^{\urcorner}$(flanco de bajada) $:= \overline{V^k} \cdot V^{k-1}$.

Tipos de relés

	$S := a$		$S := \overline{a}$
	$S := \begin{cases} 1 & \text{si } a = 1 \\ —^{13} & \text{si } a = 0 \end{cases}$		$S := \begin{cases} 0 & \text{si } a = 1 \\ — & \text{si } a = 0 \end{cases}$
	$S := \begin{cases} 1 & \text{si } a\_\!\!\lrcorner\!\!\ulcorner \\ — & \text{si } \overline{a\_\!\!\lrcorner\!\!\ulcorner} \end{cases}$		$S := \begin{cases} 1 & \text{si } a\ulcorner\!\!\_\!\!\urcorner \\ — & \text{si } \overline{a\ulcorner\!\!\_\!\!\urcorner} \end{cases}$

[13] No cambia la variable S.

Control de secuencia

Saltos

No condicionado:

```
|
+---->>etiqueta
|
```

Condicionado:

```
|   v1
+-| |---->>etiqueta
|
...
etiqueta:
|   v2
+-| |----...----( )
|
```

Retornos

No condicionado:

```
|
+-----<RETURN>
|
```

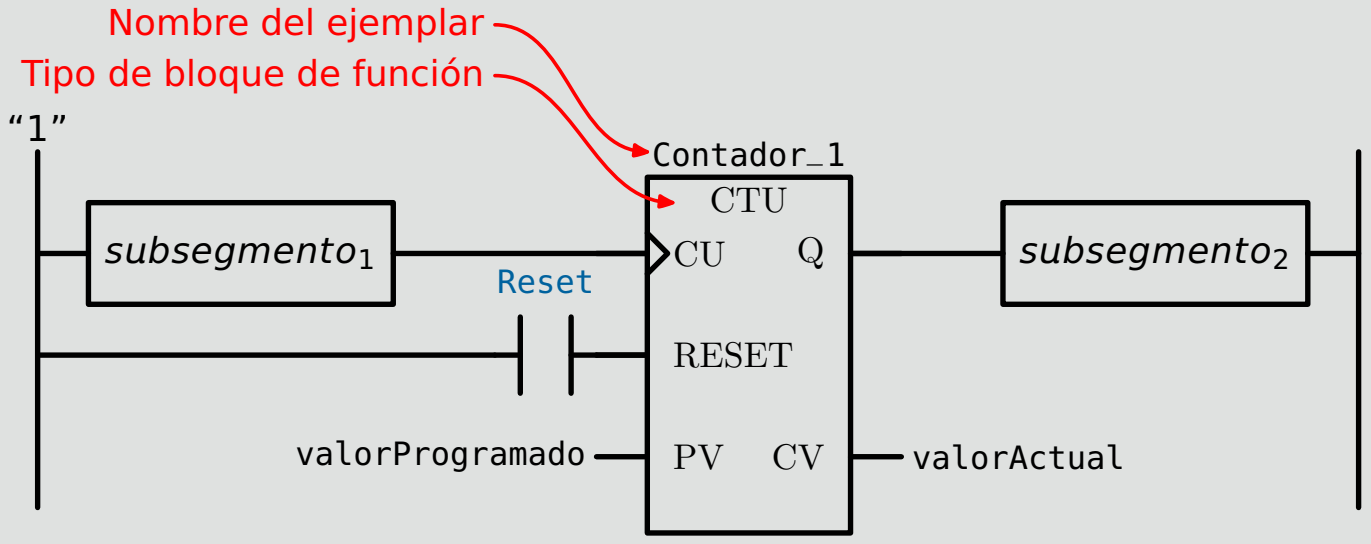
Condicionado:

```
|   v
+-| |-----<RETURN>
|
```

Conexión con funciones y bloques de función

- En un diagrama de contactos, las funciones y bloques de función se representan mediante cajas rectangulares.
- Los parámetros formales aparecen en el interior de la caja.
- Los parámetros reales se colocan en las líneas de acceso a la caja.

Conexión de un contador ascendente



CU – count up
RESET
PV – program value
CV – current value
Q – cuenta finalizada

Índice

Referencias


International Electrotechnical Commission.
Programmable controllers - Part 3: Programming languages.
IEC, 2006.


Karl-Heinz John and Michael Tiegelkamp.
IEC 61131–3: Programming Industrial Automation Systems.
Springer, 2010.