Ingeniería Informática

# AUTOMATIZACIÓN Y ROBÓTICA

## CURSO 2021/2022

Tema 5. Programación de PLCs

1

## Tema 5. Programación de PLCs

- Introducción
- Diagrama de contactos.
- Texto estructurado.
- Estados de un sistema.
- GRAFCET.
- Ejercicios.

2

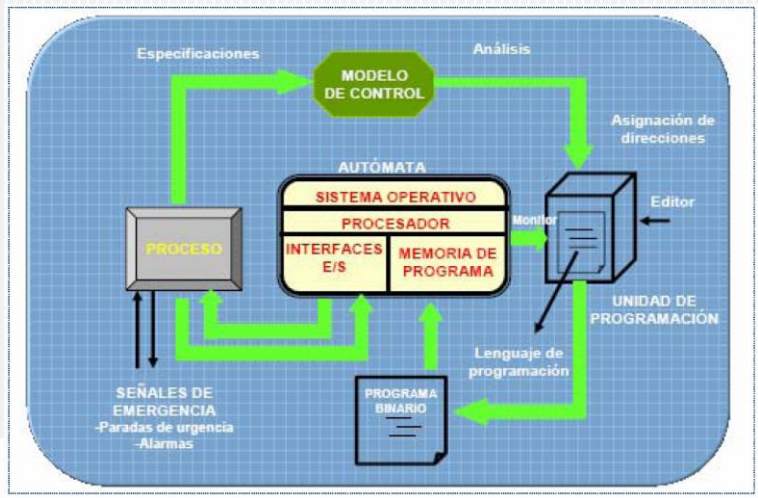
Ingeniería Informática

# INTRODUCCIÓN

3

## Introducción

- Sistemas y recursos involucrados en la programación de un PLC




4

# Introducción

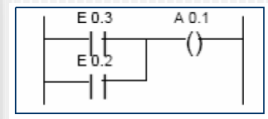
- Lenguajes de programación IEC 61131.
  - Lenguajes literales.
    - Lista de instrucciones (similar a *ensamblador*). AWL
    - Texto estructurado (lenguaje de alto nivel, similar a *Pascal*). STR – ST
  - Lenguajes gráficos.
    - Esquema de contactos. Derivado del lenguaje de Relés. KOP – LD.
    - GRAFCET.

```
U   E   0.0
U   E   0.1
=   A   4.0
```

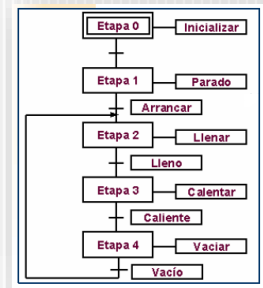
AWL

```
sum:=0;
FOR i:=1 TC 3 DC
   FOR j:=1 TC 2 DC
      (* Si flag esta activo sale*)
      IF FLAG=1 THEN EXIT;
      END_IF ;
      sum:=sum+j;
   END_FOR ;
   sum:=sum+i;
END_FOR;
```

STR




KOP




GRAFCET

5

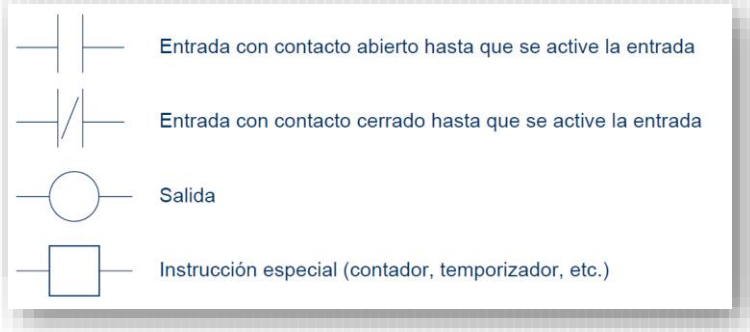
Ingeniería Informática

# DIAGRAMA DE CONTACTOS

6

# Diagrama de contactos

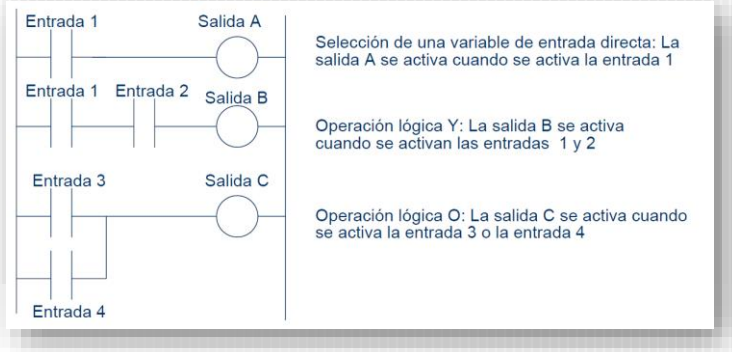
- El diagrama de una sección se compone de circuitos, llamados redes, entre las dos líneas de alimentación.
- Cada red se construye con contactos, bobinas y líneas de conexión. A los contactos y boninas se asocian variables.




7

# Diagrama de contactos

- Similar a un circuito eléctrico.
  - Laterales verticales conectados a alimentación
  - Cada escalón horizontal funcionan como circuitos-interruptores que encienden o apagan las salidas del escalón.




8

# Diagrama de contactos

- Operaciones de SET/RESET.

| | |
|---|---|
| ***<br>S | **Operación SET**: La variable se activa cuando se cierra el circuito conectado en serie. Permanece activa aunque el circuito se abra. *** representa el nombre de la variable (operando). |
| ***<br>R | **Operación RESET**: La variable se desactiva cuando se cierra el circuito conectado en serie. Permanece activa aunque el circuito se abra. *** representa el nombre de la variable (operando). |

- Detección de flancos positivos/negativos

- Flanco positivo:

***
P

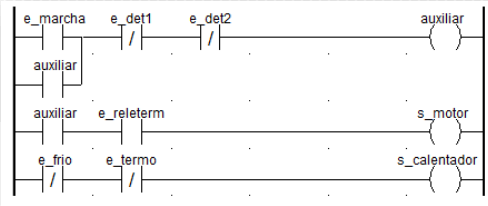
Contacto que se cierra (valor "ON") cuando la variable *** asociada con él pasa de 0 a 1. (**P** = Positive transition).

- Flanco negativo

***
N

Contacto que se cierra (valor "ON") cuando la variable *** asociada con él pasa de 1 a 0. (**N** = Negative transition).

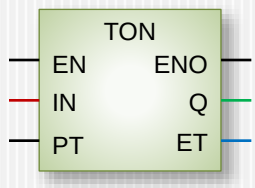
- Ejemplo con el software de Schneider.

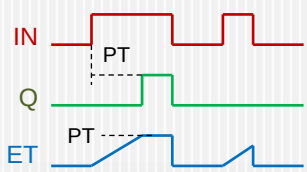


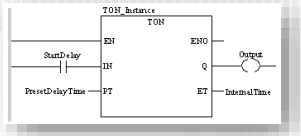

9

# Bloques de temporización IEC

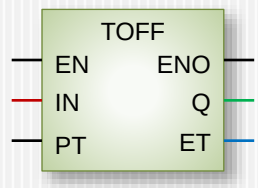
Retardo de conexión

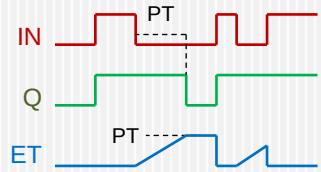


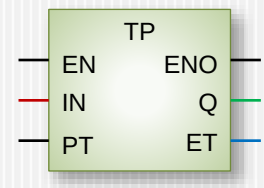



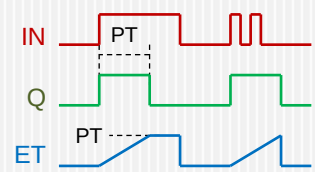

Retardo de desconexión




Generador de pulso



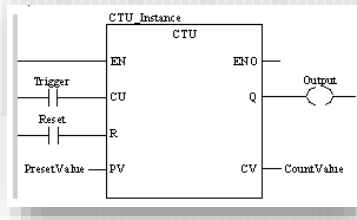



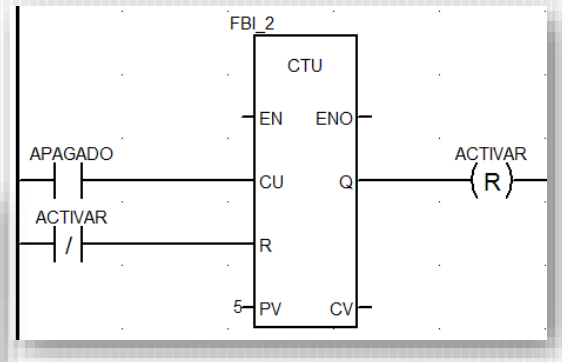

10

# Bloques de cuenta IEC

- Se dispone de contadores para:
  - CTD (Count Down). Cuenta decreciente desde un valor cargado.
  - CTU (Count Up). Cuenta creciente hasta el valor indicado.
  - CTUD. Cuenta creciente o decreciente, según la entrada usada.

Contador de 5 pulsos usando un bloque CTU

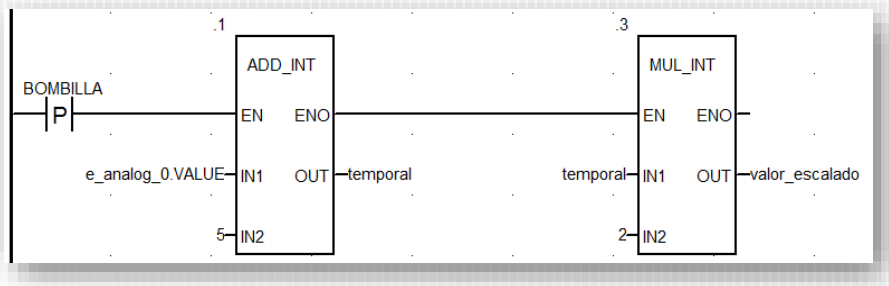



11

# Secuenciación con EN y ENO

- En LD, los bloques de función disponen de:
  - EN (ENable): El bloque se ejecuta cuando esta entrada vale 1. Si no se usa, se interpreta que vale 1.
  - ENO (EN Output): Salida que se pone a 1 cuando el bloque acaba su ejecución. Si hay errores, ENO se pone a 0.




12

## Orden de ejecución

- La ejecución se realiza línea a línea, de arriba abajo.
- Cada línea se ejecuta completamente, hasta determinar todas las salidas, antes de pasar a la siguiente línea.
- Si se usan bloques de función, estos requieren disponer de todas las entradas para ejecutarse.
- Se puede alterar el flujo de ejecución con elementos de salto, pero su abuso puede dificultar la interpretación de un programa.

13

Ingeniería Informática

# TEXTO ESTRUCTURADO

14

# Características del texto estructurado (ST)

- Lenguaje de alto nivel, al estilo de Pascal o Basic.
- Una sección es un programa estructurado con una lista de instrucciones compuestas de:
  - Instrucciones de control del flujo del programa.
  - Expresiones de evaluación y asignación de variables.
  - Llamadas a bloques de funciones y subrutinas.
- Cada instrucción debe acabar en punto y coma (;).
- Particularmente adecuado para programar cálculos aritméticos y algoritmos complejos.

15

# Expresiones

- Operadores básicos:
  - Asignación de un resultado: **:=**
  - Comparación: **>** , **<** , **=** , **>=** , **<=** , **<>**
  - Lógicos: **AND**, **&**, **OR**, **XOR**, **NOT**
  - Aritméticos: **+**, **-**, *, /, **MOD** (resto división entera), ** (potencia)
  - Paréntesis para forzar orden de evaluación: **(** , **)**
  - Llamadas a funciones: TON_ins1 (inicio, tiempo, salida, cuenta);
- Resulta fácil construir expresiones complejas:
  - s_motor := e_marcha AND NOT (e_alarma1 OR e_alarma2);
  - a := 5 + a * cos(INT_TO_REAL(%CH0.3.0.VALUE));
  - var1 := var2 := var3 := 1;
  - timeVar := 10 * T#10s;
  - salida := Instancia1_TON.Q;

16

# Instrucciones de control del programa

- Ejecución condicional de operaciones

```
(* ejemplo 1*)
IF ABS(contador) > 10 THEN
        s_refrigeracion:=1;
        contador:=0;
END_IF;

(* ejemplo 2*)
IF marcador1=TRUE THEN
        %Q0.1.16:=1;
ELSE
        %Q0.1.17:=1;
END_IF;
```

```
(*ejemplo 3*)
IF e_analog_0.VALUE>10 THEN
        temporal:=e_analog_0.VALUE;

ELSIF e_analog_0.VALUE<10 THEN
        temporal:=ABS(e_analog_0.VALUE);

ELSE
        temporal:=0;
END_IF;
```

17

# Instrucciones de control del programa

- Bucles para repetir operaciones

```
(* contador ascendente*)
FOR contador:=0 TO 4 DO
  temporal:=temporal+
        BYTE_TO_INT(tabla[contador]);
END_FOR;

(* contador descendente*)
FOR contador:=4 TO 0 BY -1 DO
  temporal:=temporal+
        BYTE_TO_INT(tabla[contador]);
END_FOR;
```

```
(* contador ascendente*)
contador:=0;
WHILE contador<5 DO
  temporal:=temporal+
        BYTE_TO_INT(tabla[contador]);
  contador:=contador+1;
END_WHILE;

(* contador descendente*)
contador:=5;
REPEAT
  contador:=contador-1;
  temporal:=temporal+
        BYTE_TO_INT(tabla[contador]);
UNTIL contador=0
END_REPEAT;
```

18

# Instrucciones de control del programa

- Selección de alternativas
  - CASE funciona como varias instrucciones IF-THEN-ELSIF, pero es más comprensible.
  - Para cada opción se puede especificar un bloque de instrucciones o expresiones.
  - Opcionalmente, se puede añadir la opción ELSE al final (*default*).

```
CASE num_estado OF
        0: IF bt_marcha THEN
                num_estado:=1;
                sal_izq:=TRUE;
           END_IF;
        1: IF bt_fc_der THEN
                num_estado:=2;
                sal_der:=TRUE;
           END_IF;
        2: IF bt_fc_izq THEN
                num_estado:=0;
                sal_izq:=FALSE;
                sal_der:=FALSE;
           END_IF;
        ELSE    error:=TRUE;
                num_estado:=0;
END_CASE;
```

19

# Control del flujo de ejecución

- JMP *etiqueta*: Salto a la etiqueta indicada. No es estándar, y no está recomendado.
- RETURN: retorno anticipado de una subrutina.
- EXIT: fuerza la finalización de una instrucción de repetición antes de que se cumpla la condición de final.

```
sum:=0;
FOR i:=1 TO 3 DO
   FOR j:=1 TO 2 DO
      (* Si flag esta activo sale*)
      IF FLAG=1 THEN EXIT;
      END_IF ;
      sum:=sum+j;
   END_FOR ;
   sum:=sum+i;
END_FOR;
```

20

Ingeniería Informática

# ESTADOS DE UN SISTEMA (EVENTOS DISCRETOS)

21

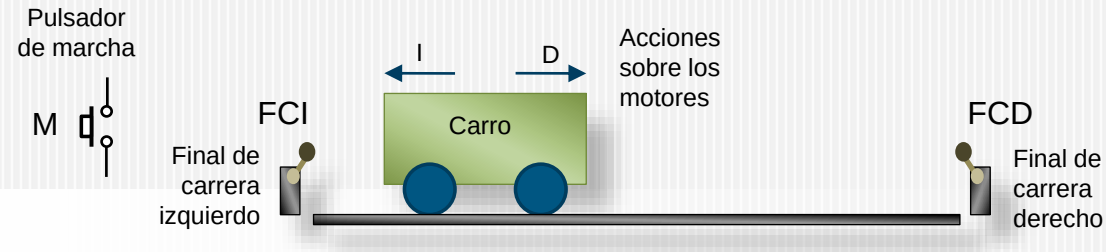
## Sistemas de eventos discretos

- Muchas aplicaciones de automatización se basan en desarrollar una secuencia ordenada de acciones.
- El controlador no comienza una acción hasta que la acción anterior no ha terminado.
- La situación del sistema durante el desarrollo de una acción estable se denomina **estado**.
- Un evento es la transición de un estado a otro.
- Eventos discretos:
  - Las transiciones de estado son instantáneas y ocurren en instantes discretos de tiempo.
  - En un intervalo finito de tiempo hay un numero finito de transiciones de estado.

22

# Ejemplo de sistema basado en eventos

- El sistema sigue esta secuencia de funcionamiento:
  - Inicialmente el carro está en reposo en el lado izquierdo.
  - Mediante M, el carro se pone en marcha hacia la derecha.
  - Cuando el carro toca FCD, se mueve a la izquierda.
  - Al tocar FCI, el carro se para, la espera de pulsar M de nuevo.



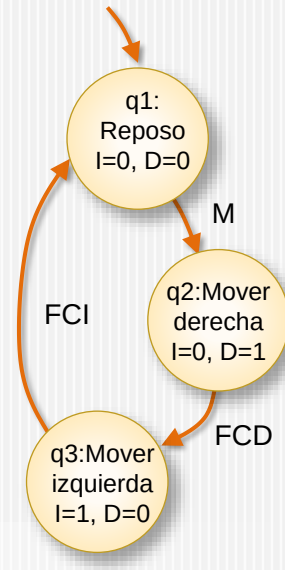

23

# Grafo de estados

- El estado siguiente ($q_s$) depende del estado actual (q) y de las entradas (e).
- Las entradas provocan las transiciones de estado.
- Las salidas (s) dependen del estado.

Q: Conjunto de estados
E: Conjunto de entradas
S: Conjunto de salidas

$q_s = f(q, e)$
$s = g(q)$
$q, q_s \in Q \quad e \in E \quad s \in S$

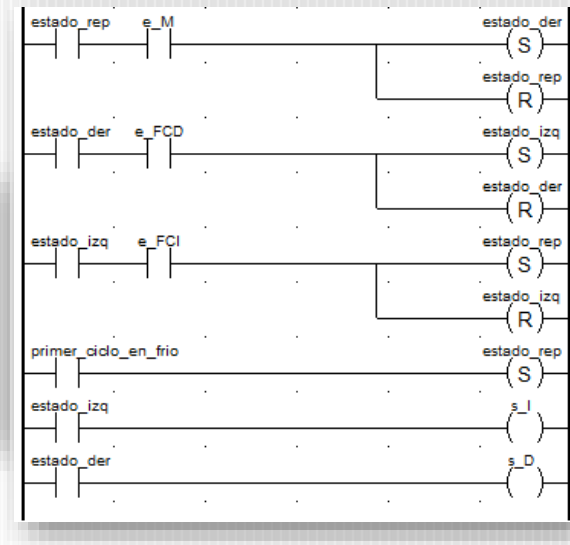



24

# Implementación con diagrama de contactos

- El estado se puede representar con variables booleanas.
- Hay que inicializar el programa al primer estado.

| Nombre | Tipo | Dirección |
|---|---|---|
| e_FCD | EBOOL | %I0.1.1 |
| e_FCI | EBOOL | %I0.1.2 |
| e_M | EBOOL | %I0.1.0 |
| estado_der | EBOOL | |
| estado_izq | EBOOL | |
| estado_rep | EBOOL | |
| primer_ciclo_en_frio | BOOL | %S0 |
| s_D | EBOOL | %Q0.1.16 |
| s_I | EBOOL | %Q0.1.17 |




25

# Implementación con texto estructurado

- El estado se puede representar con una variable entera.

```
(* Inicio *)
IF primer_ciclo_en_frio THEN
        num_estado:=0;
        sal_motor:=FALSE;
        sal_izqder:=FALSE;
END_IF;

(* Evolución de estados *)
CASE num_estado OF
        0: IF bt_marcha THEN
                num_estado:=1;
           END_IF;
        1: IF bt_fc_der THEN
                num_estado:=2;
           END_IF;
        2: IF bt_fc_izq THEN
                num_estado:=0;
           END_IF;
END_CASE;
```

| Nombre | Tipo | Dirección |
|---|---|---|
| bt_fc_der | BOOL | %I0.1.1 |
| bt_fc_izq | BOOL | %I0.1.2 |
| bt_marcha | BOOL | %I0.1.0 |
| num_estado | INT | |
| primer_ciclo_en_frio | BOOL | %S0 |
| sal_izqder | BOOL | %Q0.1.16 |
| sal_motor | BOOL | %Q0.1.17 |

```
(* Salidas *)
IF num_estado=1 OR num_estado=2 THEN
        sal_motor:=TRUE;
ELSE
        sal_motor:=FALSE;
END_IF;

IF num_estado=2 THEN
        sal_izqder:=TRUE;
ELSE
        sal_izqder:=FALSE;
END_IF;
```

26

# Generación de las salidas

- Se puede realizar el procesamiento de salidas en los cambios de estado pero el programa no queda tan claro, y es fácil cometer errores.
- Es mejor determinar las salidas según las variables de estado, como en las diapositivas anteriores.

```
(* Inicio *)
IF primer_ciclo_en_frio THEN
        num_estado:=0;
END_IF;

(* Evolución de estados y salidas *)
CASE num_estado OF
        0: IF bt_marcha THEN
                num_estado:=1;
                sal_motor:=TRUE;
                sal_izqder:=FALSE;
           END_IF;
        1: IF bt_fc_der THEN
                num_estado:=2;
                sal_motor:=TRUE;
                sal_izqder:=TRUE;
           END_IF;
        2: IF bt_fc_izq THEN
                num_estado:=0;
                sal_motor:=FALSE;
                sal_izqder:=FALSE;
           END_IF;
END_CASE;
```

27

# Metodología

- Determinar cuales son los estados estables del sistema.
- Modelar el funcionamiento con un grafo de estados.
- Identificar variables de mando, entradas y salidas.
- Definir variables de estado.
- Iniciar las variables de estado (y salidas).
- Programar los cambios de estado, según las entradas.
- Programar la generación de salidas según el estado.

28

# Problemas de los grafos de estado

- Un estado representa la totalidad del sistema.
  - El número de estados crece exponencialmente al aumentar la complejidad de la aplicación.
- Sólo sirve para problemas sencillos.
- Difícil sincronización de grafos de procesos paralelos.
- Se requiere una herramienta más potente:
  - Redes de Petri.
  - Grafcet.

29

Ingeniería Informática

# GRAFCET

30

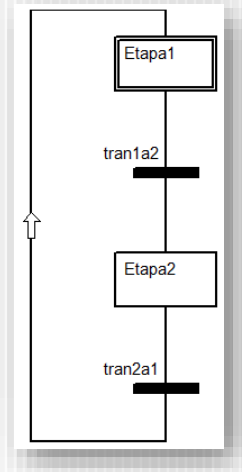
# GRAFCET (SFC)

- Creado por la extinta AFCET (Asociation Francaise pour la Cibernétique Economique et Technique) en 1977.
- Inspirado en redes de Petri.
- Como otros grafos de estados, trabaja con estados, eventos y acciones.
- Adecuado para automatismos secuenciales que se pueden estructurar en etapas.
- Definido como lenguaje de Diagrama Secuencial de Funciones (SFC) para PLCs en el IEC 61131.

31

# GRAFCET (SFC)

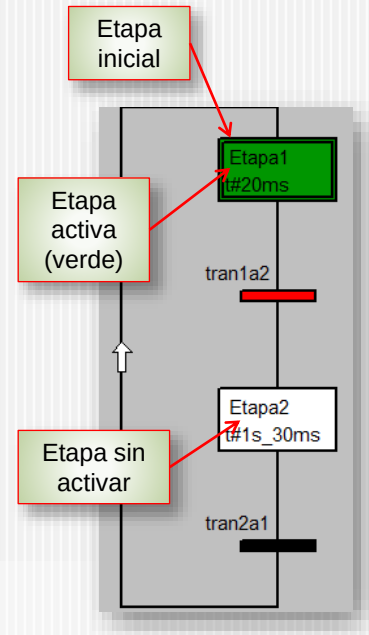
- El software de diseño de los PLCs actuales permite la especificación gráfica de Grafcet de forma directa.
- Elementos principales:
  - Etapas o pasos: Estados estables del automatismo, que tienen asociadas acciones.
  - Transiciones: Determinan los cambios de etapas en base a condiciones booleanas.
  - Conexiones direccionales: conectan una etapa con una transición o viceversa.
  - Sólo se indica la dirección hacia arriba: el resto se supone hacia abajo.




32

# Etapas

- Cada etapa tiene un nombre.
- Durante un instante de la ejecución, hay dos tipos de etapas:
  - Activas. Se están ejecutando.
  - Sin activar. En espera de que una transición las active.
- Todo Grafcet tiene una etapa inicial que se activa al iniciar el programa.




33

# Variables de una etapa

- Cada etapa tiene asignada una variable con su nombre, con estos elementos:
  - Etapa1.**t**: Tiempo de duración de la etapa. Si se desactiva la etapa, se mantiene el valor.
  - Etapa1.**x**: 1 o 0 según la etapa este activa o no.
- Esas variables sólo se pueden leer.
- Las variables se pueden usar en transiciones y acciones, y en otras secciones del programa.

| Nombre | Tipo |
|---|---|
| Etapa1 | SFCSTEP_STATE |
| t | TIME |
| x | BOOL |
| tminErr | BOOL |
| tmaxErr | BOOL |
| Etapa2 | SFCSTEP_STATE |
| t | TIME |
| x | BOOL |
| tminErr | BOOL |
| tmaxErr | BOOL |

34

## Descriptores para las acciones

- **N** (None). La acción sólo está a 1 cuando la etapa esté activa.
- **R** (Reset). Cuando la etapa se activa, la acción se pone a 0. La acción se pudo activar previamente con S.
- **S** (Set). Al activarse la etapa, la acción se pone a 1, hasta que se use R.
- **L** (Limit). Si la etapa está activa, la acción se pone a 1 durante el tiempo indicado y luego se deja a 0. Si la etapa se desactiva, la acción se pone a 0.
- **P** (Pulse). Si la etapa está activa, la acción se establece a 1, y se mantiene así durante un ciclo de programa. Luego se pone a 0.

35

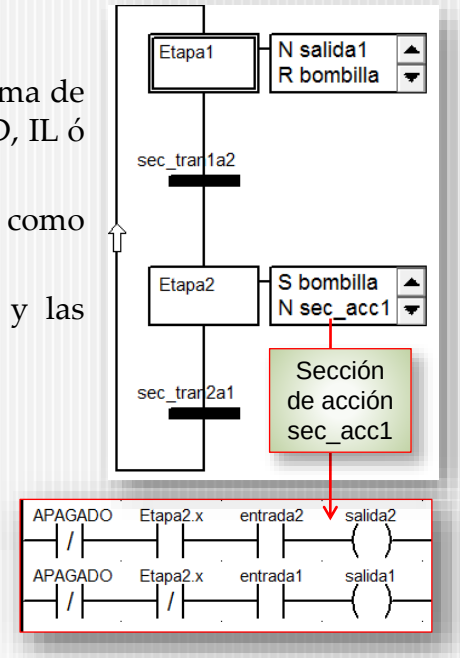
## Descriptores para las acciones

- **D** (Delay). Si la etapa se activa, se inicia un temporizador y, una vez pasado el tiempo indicado, la acción se establece en 1. Si la etapa se desactiva, la acción se pone a 0.
- **DS** (Delayed Set). Actúa como S, pero la acción no se pone a 0 automáticamente al desactivarse la etapa.
- **P1** (Pulse to 1). Cuando la etapa pasa de inactiva a activa, la acción se establece en 1 y permanece así durante un ciclo de programa.
- **P0** (Pulse to 0). Cuando la etapa pasa de activa a inactiva, la acción se establece en 1 y permanece así durante un ciclo de programa.

36

# Secciones de acción

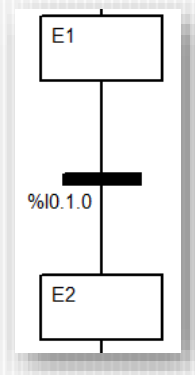
- Se pueden definir acciones complejas en forma de secciones de programa en lenguaje FBD, LD, IL ó ST.
- Una misma sección de acción se puede usar como acción de multiples estapas.
- No se puden usar bloques de funciones, y las llamadas a subrutinas tienen limitaciones.




37

# Transiciones

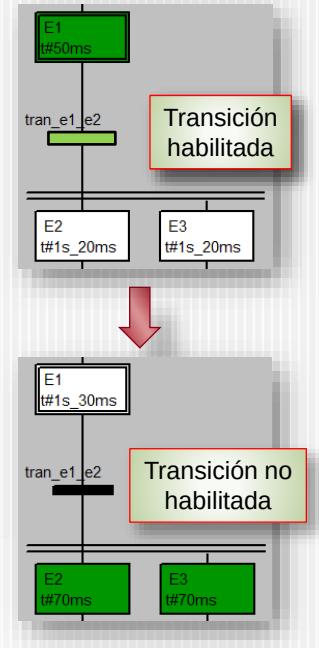
- Definen las condiciones que permiten evolucionar de una o más etapas activas a otras etapas no activas.
- Una transición se habilita si:
  - Todas las etapas de entrada están activas.
  - La condición asociada a la etapa es verdadera o 1.
- La condición de una transición puede ser:
  - Una variable o dirección booleana.
  - Una sección que devuelve un valor booleano.
  - El valor literal 1 (verdadero).




38

# Transiciones

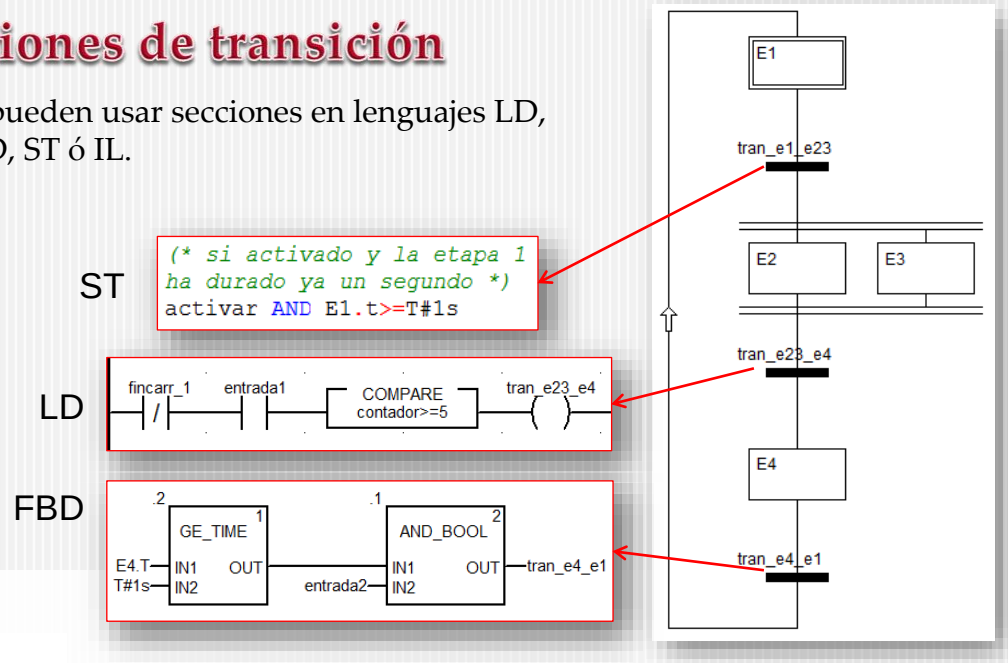
- Si una transición habilitada se ejecuta:
  - Se desactivan todas las etapas inmediatamente anteriores.
  - Se activan todas las etapas inmediatamente posteriores.
- Si hay varias transiciones habilitadas, se disparan todas ellas.
- El tiempo de ejecución de una transición puede ser muy pequeño, pero no cero.




39

# Secciones de transición

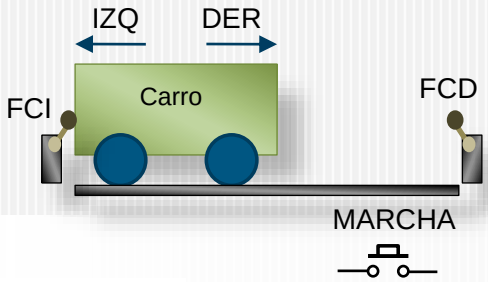
- Se pueden usar secciones en lenguajes LD, FBD, ST ó IL.

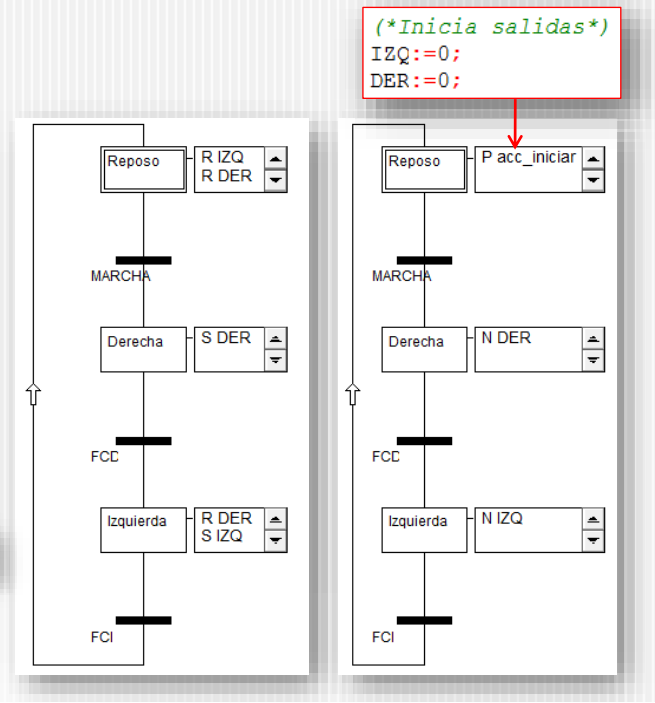



40

# Secuencia única

- Es una sucesión de etapas que son activadas una a continuación de otra.

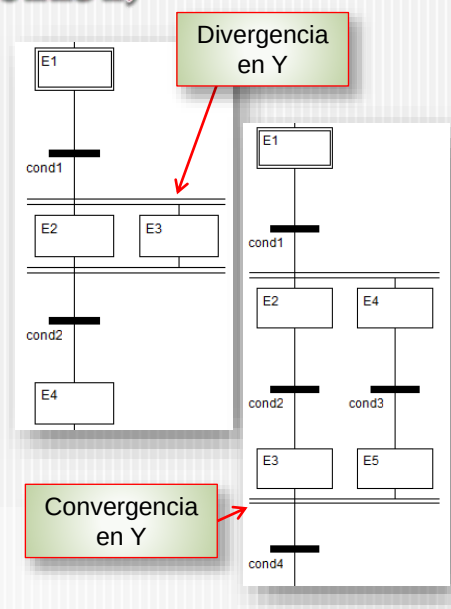



41

# Secuencias simultáneas (IEC 61131)

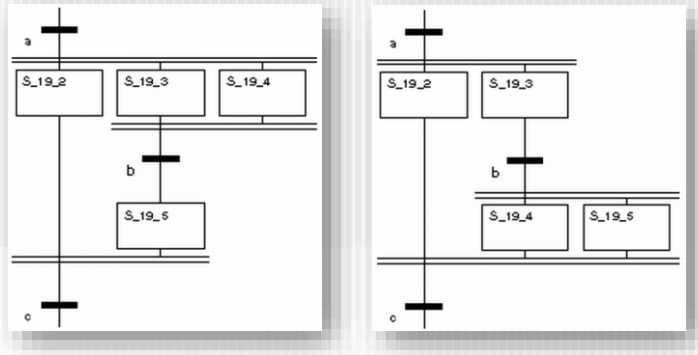
- La salida de una transición se conecta a varias etapas.
  - El disparo de la transición lleva a la activación simultánea de varias etapas, o secuencias de etapas.
- Las secuencias simultáneas deben converger en una transición.
  - La transición se habilitará si todas sus etapas previas están activas.
  - Las etapas paralelas se desactivan.




42

# Secuencias simultáneas (IEC 61131)

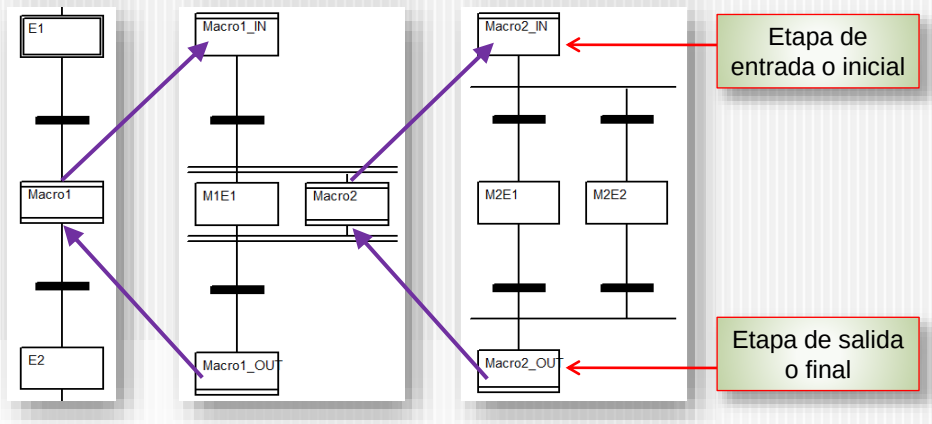
- Según el IEC 61131, una divergencia en Y siempre debe cerrarse con una convergencia en Y.
- El número de divergencias simultáneas no tiene porque coincidir con el número de convergencias simultáneas.




43

# Macroetapas

- Son etapas que contienen un Grafcet dentro.
- Permiten estructurar el programa por niveles de detalle.




44

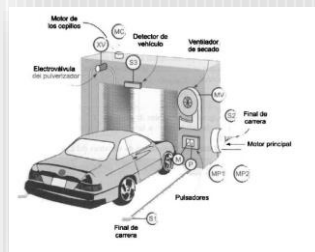
Ingeniería Informática

# EJERCICIOS

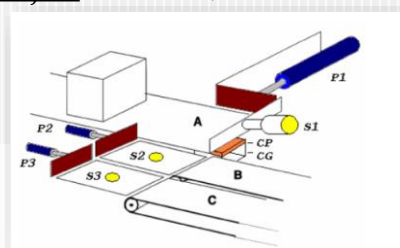
45

## Lista de ejercicios

- Tren de lavado de vehículos → Estados, KOP/STR.




- Posicionado de cajas → Estados, KOP/STR.




46

Ingeniería Informática

# AUTOMATIZACIÓN Y ROBÓTICA

## CURSO 2021/2022

Tema 5. Programación de PLCs

47