Controladores lógicos programables –PLC– 2

Anexos

MINISTERIO de EDUCACIÓN CIENCIA y TECNOLOGÍA PRESIDENCIA de la NACIÓN

inet Instituto Nacional de Educación Tecnológica

serie/desarrollo de contenidos
colección/fluídica y controladores lógicos programables

Autoridades

Presidente de la Nación
Néstor C. Kirchner

Ministro de Educación, Ciencia y Tecnología
Daniel Filmus

Directora Ejecutiva del Instituto Nacional de Educación Tecnológica
María Rosa Almandoz

Director Nacional del Centro Nacional de Educación Tecnológica
Juan Manuel Kirschenbaum

Especialista en contenidos

- Norberto Molinari

Ministerio de Educación, Ciencia y Tecnología
Instituto Nacional de Educación Tecnológica
Saavedra 789. C1229ACE
Ciudad Autónoma de Buenos Aires
República Argentina

serie/desarrollo de contenidos

Colecciones

- Autotrónica
- Comunicación de señales y datos
- Diseño gráfico industrial
- Electrónica y sistemas de control
- Fluídica y controladores lógicos programables
    1. Tecnología neumática
    2. Controladores lógicos programables –PLC–
- Gestión de la calidad
- Gestión de las organizaciones
- Informática
- Invernadero computarizado
- Laboratorio interactivo de idiomas
- Procesos de producción integrada
- Proyecto tecnológico
- Unidades de cultura tecnológica

Índice

OTROS LENGUAJES DE PROGRAMACIÓN

Anexo 1 / Estructura del lenguaje STL

[1] *Además de los lenguajes* ***Ladder y Lista de instrucciones****, existen otros que, en menor o mayor grado, también son muy adecuados cuando trabajamos con equipos de origen alemán o francés: el lenguaje* ***STL*** *y el lenguaje* ***GRAFCET****. Nos ocupamos de ellos en estos dos documentos anexos que completan nuestro módulo.*

Jerarquía de los elementos STL

***Lenguaje STL* –Statement List–**
Permite al programador resolver tareas de control usando expresiones sencillas en inglés, las que describen la operación deseada del controlador.

La naturaleza modular del lenguaje STL[2] permite al programador resolver tareas complejas de una manera eficiente, sin requerir el conocimiento de la totalidad de estos elementos pero sí la forma en que esos elementos son combinados, ya que influye de modo importante en la operación del programa.

Consideremos cómo está organizado este lenguaje:

Jerarquía de los elementos STL:

PROGRAM

STEP
SENTENCE
CONDITIONAL PART
EXECUTIVE PART

Aunque el uso de la instrucción STEP **–paso–** es opcional, la mayoría de los programas requiere su uso. La instrucción *step* se utiliza para marcar el comienzo de un bloque lógico de código de programa. Cada programa STL puede contener hasta 255 *step*s discretos, pudiendo cada uno de estos *step* incluir una o más ***sentences* – oraciones o enunciados–.**

Cada *step* puede tener asignada una etiqueta –*label*–, a voluntad del programador, la que puede ser requerida para referencia futura del *step*.

Un *label* para un *step* sólo es requerido si el *step* respectivo es asignado como destino por una instrucción de salto –*jump*-.

La *sentence* forma el nivel básico de la organización del programa. Cada *sentence* consiste en una parte condicional y una parte ejecutiva.

La **parte condicional** sirve para enumerar una o más condiciones que deben evaluarse cuando el programa está corriendo y que pueden ser ciertas o falsas. La parte condicional siempre comienza con la expresión If y continúa con uno o más enunciados que describen las condiciones a ser evaluadas.

Si las condiciones programadas son evaluadas como *true* –verdaderas–, entonces cualquier instrucción programada en la **parte ejecutiva** es cumplida. El comienzo de la parte ejecutiva está marcado por la expresión THEN.

Sentencias típicas

Consideremos sentencias típicas en STL sin el uso de la instrucción *step*:

IF I1.0	Si input 1.0 está activa
THEN SETO1.2	entonces conmutar a si output 1.2
IF N I2.0	Si input 2.0 no está activa
THEN SET O2.3	entonces conmutar a si output 2.3

[2] El lenguaje STL, como se describe aquí, se aplica a los controladores de Festo® Modelos FPC100 B/AF, FPC405, FEC, IPC y SF03. La información contenida en este documentos se corresponde con el lenguaje STL según la implemetación del FST Software Versión 3.X.

```
IF I6.0              Si input 6.0 está activa
AND N I2.1           y la input 2.1 no está activa
AND   O3.1           y la output 3.1 está activa
THEN  RESET O2.1     entonces apagar output 2.1
RESET T6             y reset timer 6
```

En el ultimo ejemplo, se ha introducido el principio de condiciones compuestas. Es decir, todas las condiciones establecidas en la sentencia deben cumplirse (ser verdaderas) para que las acciones –posteriores a la expresión THEN– sean ejecutadas.

Más ejemplos:

```
IF I3.2              Si input 3.2 está activa
ORN T6               o timer 6 no está corriendo
THEN  INC CW1        entonces incrementar counter 1
SET T4               y arrancar timer 4 con parámetros
                     preexistentes
```

Este ejemplo muestra el uso de la estructura OR en la parte condicional de una sentencia. Por lo tanto, la sentencia es evaluada como verdadera (y se incrementará el contador 1 y se arrancará el timer 4), si alguna o ambas de las condiciones establecidas son ciertas.

La próxima sentencia introduce el uso de paréntesis en la parte condicional, para establecer la forma en que las condiciones son evaluadas.

```
IF (11.1             Si input 1.1 está activa y
AND   T4 )           Timer 4 está corriendo
OR    (I1.3          O si input 1.3 está activa
AND   I1.2 )         e input 1.2 está activa
```

Hemos usado la instrucción OR para combinar por medio del paréntesis dos condiciones compuestas.

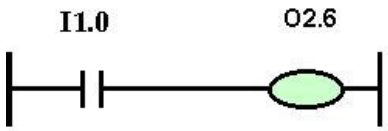
Hemos introducido estos ejemplos para mostrar el uso de sentencias en el *Lenguaje Statement List.* Es posible crear programas enteros que consistan sólo en múltiples sentencias sin usar la instrucción STEP.

A los programas construidos de esta forma se los llama **programas paralelos**. Estos programas se comportan casi de la misma forma que los programas en el lenguaje *Ladder Diagram*. Esto es, sin usar la instrucción STEP.

Para que estos programas puedan funcionar y ser procesados continuamente, es necesario agregar la instrucción PSE.

Para quienes están familiarizados con el *Lenguaje Ladder Diagram*, puede decirse que hay gran similitud entre una sentencia STL y una rama en *Ladder Diagram*.

Por ejemplo, una rama en *Ladder Diagram* para activar una salida (ponerla a ON) siempre que la entrada está activa y apagarla (OFF) cuando la entrada está inactiva, es:



Mientras que la sentencia equivalente en STL es:

```
IF I1.0              Si input 1.0 está activa
THEN SET 02.6        Entonces conmutar a si output 2.6
PSE                  Fin de programa
OTHRW RESET 02.6     De otra forma, apagar output 2.6
PSE                  Fin de programa
```

En este ejemplo se ve la inclusión de la instrucción OTHRW; porque, el lenguaje STL requiere instrucciones explícitas para alterar el estado de cualquier operando (por ejemplo: output, timer, counter).

La instrucción PSE, por su parte, se coloca al final de la sección del programa paralelo para forzar al programa a ejecutarse continuamente, retornando a la primera sentencia del *step* corriente o a la primera sentencia del programa, si no hubiera *steps.*

Instrucción STEP

Los programas que no usan la instrucción STEP son procesados en forma paralela (barrido). Aunque este tipo de programa puede ser apto para resolver cierto tipo de tareas de control, el lenguaje STL provee la instrucción STEP.

Esta instrucción permite que los programas sean divididos en secciones discretas –STEPS– que son ejecutadas independientemente.

En su forma más simple, un STEP incluye al menos una sentencia y toma la siguiente forma:

```
STEP (label)
IF I1.0              Si la input 1.0 es activa
THEN SET 02.4        entonces activar output 2.4 y proceder
                     al próximo step
```

Es importante comprender que el programa espera en este STEP hasta que las condiciones sean ciertas; recién en ese momento las acciones son ejecutadas y entonces, sólo entonces, el programa sigue al próximo STEP.

La etiqueta –*label*– del STEP sólo es requerida si un STEP es el destino de una instrucción de salto –*jump*–.

Debe notarse que cuando el softvvare FST carga los programas STL en el PLC, asigna números relativos de STEP a cada uno de éstos. Estos números son reproducidos en los listados de programas, siendo de gran ayuda para monitorear la ejecución de programas *online* con propósito de búsqueda de fallas –*debugging*–.

Los STEP de un programa pueden incluir múltiples sentencias:

```
STEP
IF I2.2              Si input 2.2 es activa
THEN SET 04.4        Conmutar y activa output 4.4
IF I1.6              Si input 1.6 es activa
THEN RESET 02.5      Apagar output 2.5
SET 03.3             y activar output 3.3
```

En el ejemplo previo, se ha introducido el concepto de múltiples sentencias dentro de un STEP único. Cuando el programa alcanza este STEP, procesa la primera sentencia (en este caso en particular, activando la salida 4.4 si la entrada 2.2 está activa) y, luego, se mueve a la segunda sentencia sin importar si las condiciones de la primera son verdaderas.

Cuando la última (en este caso, la segunda) sentencia de un STEP es procesada:

- Si la parte condicional es verdadera, entonces la parte ejecutiva es llevada a cabo y el programa procede al próximo *step*;
- Si la parte condicional de la última sentencia es no verdadera, entonces el programa retorna a la primera sentencia del *step* actual.

Al desarrollar programas o *step*s que contengan múltiples sentencias, es importante comprender que éstas son procesadas en forma paralela (barrido): Cada vez que la parte condicional de una sentencia es evaluada como verdadera, las instrucciones programadas en la parte ejecutiva son ejecutadas. Esto debe ser considerado para evitar una ejecución múltiple descontrolada tal como SET TIMER (arrancar un timer) o INC/DEC counter (incrementar/decrementar un contador).

El lenguaje STL no usa condiciones de “disparo por flancos”; las condiciones se evalúan por verdadero o falso, nb sin importar el estado previo.

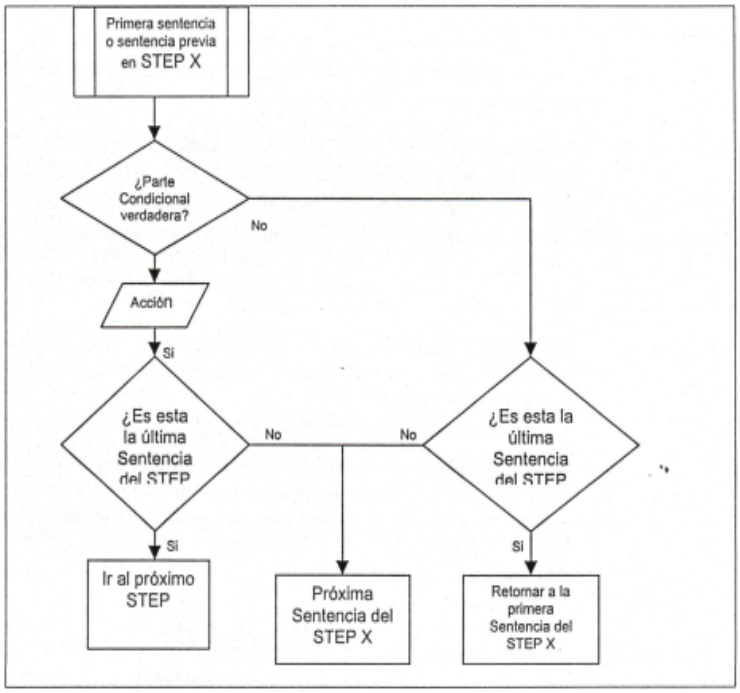
Esta situación es fácilmente manejada usando *step*s, flags u otro medio de control.

Las siguientes líneas pueden aplicarse como guía para determinar cómo los *step*s y las sentencias son procesados por el PLC:

Reglas de ejecución:

- Si las condiciones de una sentencia se cumplen, entonces las acciones programadas son ejecutadas.
- Si las condiciones de la última (o única) sentencia dentro de un *step* se cumplen, entonces las acciones programadas son ejecutadas y el programa procede al próximo *step*.
- Si las condiciones de una sentencia no se cumplen, entonces el programa se mueve a la sentencia siguiente en el *step* actual.
- Si las condiciones de la última (o única) sentencia dentro de un *step* no se cumplen, entonces el programa retorna a la primera sentencia del *step* actual.

La siguiente figura ilustra la estructura de proceso de un *step* STL; usando varias combinaciones de *step*s con una o varias sentencias, el lenguaje STL brinda un amplio rango de posibilidades para resolver las más complejas tareas:



Modificando el flujo del programa

Además de las estructuras de control inherentes en la instrucción STEP, hay varias instrucciones adicionales STL disponibles para modificar el criterio de ejecución de los *step*s y sentencias del programa.

El lenguaje STL tiene las siguientes instrucciones que permiten resolver, en forma rápida y sencilla, tareas de control simples y complejas.

	SUMARIO DE INSTRUCCIONES STL
INSTRUCCIÓN	**PROPÓSITO**
AND	Lleva a cabo una operación Y-lógica en un operando monobit o multibit, y en constantes.
BID	Convierte el contenido del acumulador monobit, de formato binario a BCD.
CFM n	Comienza la ejecución o inicialización de un módulo de función.
CMP n	Comienza la ejecución de un módulo de programa.
CPL	Produce el complemento a dos, del contenido del acumulador monobit.
DEC	Decrementa un acumulador / operando monobit.
DEB	Convierte el contenido del acumulador monobit de formato BCD a binario.
EXOR	Realiza una operación lógica EXOR en operandos monobit o multibit, y constantes.
IF	Este comando marca el comienzo de la parte condicional de una sentencia.
INC	Incrementa un operador/acumulador monobit.
INV	Produce el complemento a uno del contenido del acumulador monobit.
JMP TO (*Step* label)	Hace que el programa siga ejecutándose en el *Step* especificado por el label (etiqueta).
LOAD	Realiza la carga de operandos especificados (monobit o multibit) / constantes al acumulador monobit o multibit.
NOP	Es una instrucción especial que siempre es verdadera (siempre se cumple), en la parte condicional de una sentencia. En la parte ejecutiva, es equivalente a "no hacer nada".
OR	Realiza la operación lógica OR en operandos monobit y monobit, y constantes.
OTHRW	Permite continuar la ejecución del programa, si la parte condicional de una sentencia es *false*.
PSE	–*Program Section End*– Fin de la sección del programa.
RESET	Modifica operandos monobit a su estado lógico "0".
ROL	Rota a izquierda una posición todos los bits contenidos en el acumulador monobit. El bit más significativo se mueve a la posición del menos significativo.

ROR	Rota a derecha una posición todos los bits contenidos en el acumulador monobit. El bit menos significativo se mueve a la posición del más significativo.
SET	Modifica operandos monobit a su estado lógico "1".
SHIFT	Realiza la operación de transferencia –*swap*– entre un operando monobit y el acumulador monobit.
SHL	Desplaza una posición a izquierda a todos los bit contenidos en el acumulador monobit. El bit más significativo se pierde y el menos significativo se llena con un cero (0).
SHR	Desplaza una posición a derecha a todos los bit contenidos en el acumulador monobit. El bit menos significativo se pierde y el más significativo se llena con un cero (0).
SWAP	Intercambia los bytes alto y bajo del acumulador monobit.
TO	Se usa en conjunción con la instrucción LOAD, para especificar un operando de destino.
THEN	Marca el comienzo de la parte ejecutiva de una sentencia.
WITH	Instrucción usada para pasar parámetros con algunas instrucciones del tipo CFM/CMP. En algunos modelos de PLC, especifica velocidades de reloj.

Consideremos algunas de estas instrucciones:

- **Instrucción NOP:**

La instrucción NOP puede usarse en la parte condicional o en la ejecutiva de una sentencia.

Si NOP es usada en la parte condicional, siempre es evaluada como verdadera. La instrucción NOP puede usarse para la ejecución incondicional de una sentencia.

```
IF NOP          esto siempre es verdadero
THEN SET O1.0   entonces output 1.0 será siempre activada cuando el
                programa ejecute esta línea.
```

El uso típico puede verse en el ejemplo siguiente:

El programador desea que, cuando la ejecución del programa llegue al *step* 50, se verifiquen determinadas condiciones y, en caso de ser éstas verdaderas, se ejecuten las acciones apropiadas.

Sin embargo, sin importar si alguna de las condiciones se cumplen, después de pasar exactamente una vez el programa, encenderá la output 3.6 y procederá al próximo *step*. Esto es porque hemos forzado la última sentencia a ser verdadera, mediante la instrucción NOP

```
STEP 50
IF I1.0           Si  input 1.0 es  activa
THEN SET 02.2     Entonces activar output 2.2
IF N I3.5         Si input 3.5 no es activa
AND I4.4          e input 4.4 es activa
THEN RESET O1.2   entonces apagar output 1.2
IF T3             Si timer 3 está corriendo
THEN  SET F0.0    entonces set flag 0.0
IF NOP            En cualquiera de los casos nos aseguramos que la
                  ultima sentencia sea siempre verdadera.
```

La instrucción NOP puede usarse en la parte ejecutiva de una sentencia. Cuando se la usa de esta forma, un NOP es equivalente a "hacer nada". Se usa a menudo cuando el programa debe esperar por ciertas condiciones y, luego, proceder al próximo *step*.

IF I3.2	Si input 3.2 es activa
THEN NOP	no hacer nada e ir al próximo *step*
THEN SET 03.6	encender output 3.6, salir de este *step* e ir al próximo *step*.

- **Instrucción JMP TO:**

Otra instrucción STL que puede usarse para modificar el flujo de ejecución del programa es la instrucción JMP.

La instrucción JMP permite al programa ramificarse. Modificando el ejemplo que consideramos hace un momento, es posible consultar las condiciones de cada sentencia y, si se cumplen, ejecutar la acción programada y luego saltar –*jump*– al *step* designado del programa.

STEP 50	
IF I1.0	Si input 1.0 es activa
THEN SET O2.2	Encender output 2.2
JMP TO 70	Y saltar a *step* label 70
IF N I3.5	Si input 3.5 no es activa
AND I4.4	e input 4.4 es activa
THEN RESET O1.2	Apagar output 1.2
JMPTO 6	y saltar a *step* label 6
IF T3	Si timer 3 está corriendo
THEN SET F0.0	entonces set flag 0.0
IF NOP	Siempre verdadero, luego...
THEN SET 03.6	Encender output 3.6 e ir al próximo paso.

Puede verse que no solamente hemos alterado el flujo del programa, sino que además hemos establecido prioridades entre las sentencias.

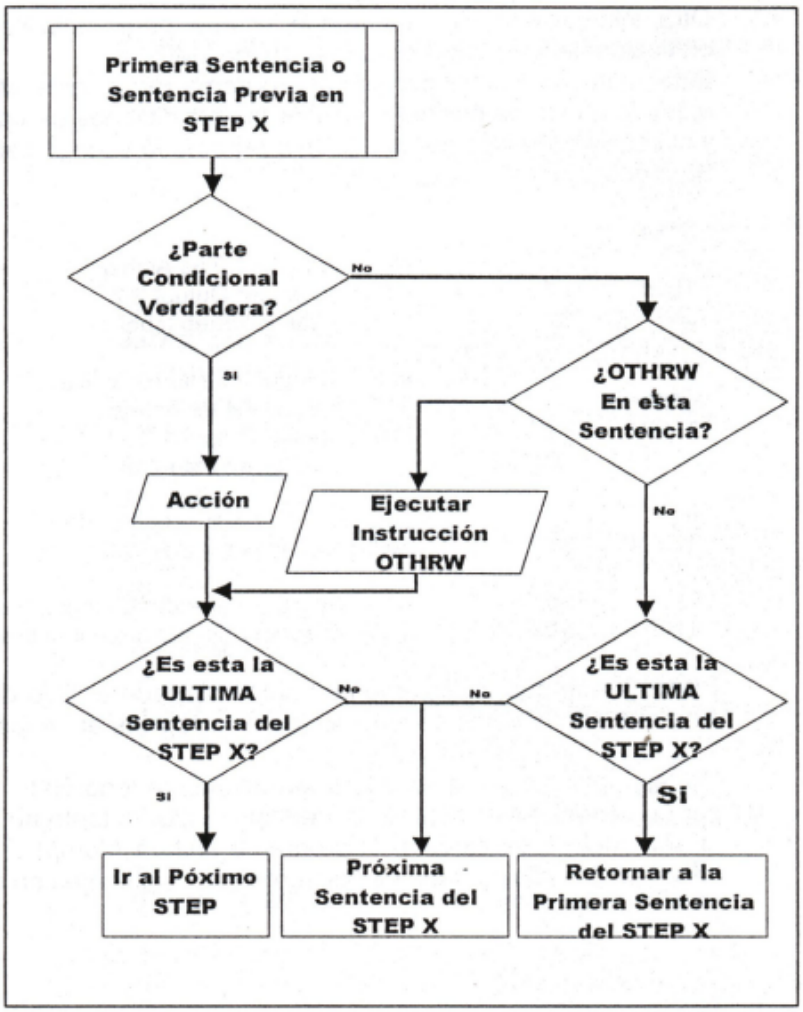
Por ejemplo, las sentencias 2, 3 y 4 solamente tienen la posibilidad de ser procesadas:

- si la sentencia 1 es falsa y, por lo tanto, no ejecutada;
- si la sentencia 1 se ejecuta, el programa salta al *step* 70 sin haber procesado ninguna de las sentencias siguientes en el *step* 50.

- **Instrucción OTHRW**

La instrucción OTHRW –*otherwise*; por otra parte, sino– puede cambiar el flujo del programa. Esta instrucción se ejecuta cuando la última cláusula IF encontrada es evaluada como no válida –*not true*–.

IF I2.0	Si input 2.0 es activa
THEN SET 03.3	encender output 3.3
OTHRW SET 04.5	sino encender output 4.5



Direccionando entradas y salidas –inputs y outputs–

Vamos a detallar ahora cómo acceder a las entradas digitales –*inputs*– y a las salidas digitales –outputs–, usando el lenguaje STL.

Inputs y outputs están conectados a la CPU (en donde los programas de control en STL son almacenados) por medio de las siguientes formas:

- Sistema de field bus.
- Redes.

En unas páginas más nos ocuparemos de ellos.

Los controladores lógicos programables de *Festo*® organizan las inputs y outputs (I/O) como un **word** –grupo–. Dependiendo del modelo de controlador particular (o del módulo de I/O para sistemas modulares), cada grupo, usualmente, consiste en 8 o 16 inputs o outputs discretas.

Estos grupos completos de words se referencian por su tipo, input u output, y por la dirección de la word (n). Este número de dirección es generalmente fijo en controladores pequeños y configurable (por medio de llaves) en sistemas modulares.

- las **word de entrada** –input words–, a menudo, son identificadas con **iwn**; mientras que

- las ***word de salida*** –output words– son declaradas como **own**.

Los ejemplos incluyen:

IW1	Input Word 1
IW7	Input Word 7
OW0	Output Word 0
OW2	Output Word 2

Note que cada entrada y cada salida dentro de un sistema tiene un número de dirección único; no es posible para un sistema tener direcciones duplicadas de I/O.

Sin embargo, es generalmente aceptable para un sistema incluir un *input word* con el mismo número de direcciones que el de un *output word* (por ejemplo, IW1 y OWI).

Las entradas y salidas individuales que están en un grupo de I/O están identificadas de la siguiente forma:

- Si es entrada o salida (I u O) + el número de dirección de la word (n) + "." Seguido por el número particular de posición de I / 0 (Sn).

Los números de la posición pueden estar entre -7 o 0-15, dependiendo del tamaño del grupo de I/O.

Por ejemplo:

I3.2	Posición input 2 de la input word 3
I0.15	Posición 15 de la input word 0
02.7	Posición output 7 de la output word 2
O0.0	Posición output 0 de la output word 0

1. Usando Inputs en programas:

Las inputs son elementos del sistema de control que están diseñados para ser leídos o consultados. Están conectadas a dispositivos externos tales como sensores, llaves, etc., los que pueden o no suministrar una señal a una entrada individual.

Ejecutando las instrucciones STL apropiadas dentro de la parte condicional de una sentencia, el controlador es capaz de determinar el estado corriente de una **entrada discreta**.

IFI1.1	Verifica una señal válida en input 1.1
IF N I3.3	Verifica una señal falsa en input 3.3

Las entradas múltiples, así como otras condiciones, pueden combinarse lógicamente.

Algunas veces, puede ser deseable o necesario verificar el estado de words de entrada –*input words*– completas. Para determinar el estado de una *input word* completa, es necesario leer el valor de la word entera y determinar si cumple el criterio deseado.

Por ejemplo, para verificar si todas las 8 inputs de la input word 2 están recibiendo señales válidas, podríamos hacer un AND lógico a cada input:

```
IF          I2.0
    AND     I2.1
    AND     I2.2
    AND     I2.3
    AND     I2.4
    AND     I2.5
    AND     I2.6
    AND     I2.7
```

Aquí verificamos si todas las 8 inputs de una input word de 8 bit están recibiendo señales válidas o usando la ventaja del lenguaje STL de evaluar words completas.

La siguiente podría ser la secuencia del programa:

```
IF (Iw2      solamente verifica si todas las 8 inputs
= V255)      están en ...11111111 (binario) = 255
```

Tareas más complejas, las que requerirían largos listados si se programaran bit a bit, son fácilmente llevadas a cabo usando input words combinadas con otras instrucciones lógicas.

Para ver si una o más de las inputs 1.5, 1.6 1.7 son válidas, se hace de la siguiente forma:

```
IF  Iw1      primero obtiene la word entera
AND  V224)   = 111 00000 binario
> V31        si el resultado es mayor que... Aquí tendremos
```

Lo que es equivalente a:

```
IF    I1.5
      OR I1.6
      OR I1.7
```

2. Usando outputs en programas:

Las outputs de un controlador programable pueden usarse para controlar distintos tipos de dispositivos eléctricos, mediante instrucciones de programa, lo que activará (SET) o desactivará (RESET) las salidas requeridas (output).

Mientras las inputs pueden sólo ser leídas (consultadas), las outputs pueden ser escritas (SET o RESET) y pueden también ser consultadas en la misma forma que las entradas. Por lo tanto, las referencias a las outputs pueden aparecer tanto en la parte condicional como en la parte ejecutiva de una sentencia STL.

Ejecutando las instrucciones STL apropiadas en la parte ejecutiva de una sentencia, el controlador puede conmutar una output particular a SI o NO.

- La instrucción SET se usa para conmutar a SI una output.
- La instrucción RESET conmuta la output a No.

```
IF                cualesquiera condiciones necesarias
THEN SET O1.2     activar output 1.2
RESET 03.3        apagar output 3.3
```

Setear una output que ya está SET o resetear una output que ya está RESET no tiene ningún efecto. Como se nota, las outputs pueden ser consultadas en la parte condicional.

La siguiente instrucción verifica si la input 2.4 está recibiendo una señal válida y si la output 2.2 está activa en este momento:

```
IF I2.4          Input 2.4 activa
AND 02.2         Output 2.2 activa
THEN ...         Acciones deseadas
```

Algunas veces puede ser deseable o necesario verificar o alterar el estado de *output words* enteras. De la misma forma, las entradas pueden ser manipuladas tomando como base grupos de word.

Los mismos principios que hemos referido a las input se aplican a las outputs.

Por ejemplo, de la sentencia STL:

```
THEN LOAD V0
TO OW2
```

resulta que en la totalidad de las outputs asociadas con la output word 2 sean apagadas.

A partir de aquí analizaremos cómo se trabaja con timers, contadores, registros y flags en el lenguaje STL.

Organizaremos la exposición en cuatro títulos:

- Usando timers
- Usando counters
- Usando registros
- Usando flags y flags words

Usando timers

Cada timer implementado en el lenguaje STL consta de los elementos:

Elemento / Operando	Referencia	Función
Bit de estado del timer	Tn	Permite a un programa consultar si el timer está activo (corriendo). El bit cambia a activo cuando se arranca (SET) y el período del timer está activo (corriendo). Al completarse el tiempo programado o si el timer es detenido (RESET), el bit de estado se vuelve inactivo.
Timer preselect (Preselección del temporizador)	Tpn	Es un operando de 16 bit que contiene el valor que define el período para el timer n.
Timer Word	Twn	Es un operando de 16 bit al que el operando TP –tiempo de preselección– es transferido automáticamente cuando el timer es arrancado (SET). Luego este valor es decrementado por el sistema a intervalos.

Se requiere de varios pasos para usar un timer en un programa STL:

1. Establecer una preselección de timer válida.
2. Establecer una instrucción para arrancar el timer.
3. Comprobar el estado del timer: activo / detenido.

1. Establecer una preselección de timer válida[3]:

Antes que pueda usarse cualquier timer, debe inicializarse el *timer preselect* con un valor correspondiente al período de tiempo deseado.

Esta inicialización sólo es necesaria de ser llevada nuevamente a cabo si el valor preseleccionado de tiempo debe ser cambiado. No es necesario cargar nuevamente el *timer preselect* cada vez que el timer es arrancado.

Esta inicialización necesita ser realizada con un valor (constante) o con cualquiera de los contenidos de un MBO –Monobit Operandos– (por ejemplo, registro, input word, flag word, etc.)

Por ejemplo, para la inicialización de la preselección de timer con la velocidad del reloj:

```
STEP 1          ¡Lo hacemos primero!
Lf NOP          Incondicionalmente
LOAD V10        Valor 10
TO TP4          A la preselección  del timer 4,
WITH SEC        velocidad  reloj = seconds ...timer 4. Será ahora un timer
                de 10 segundos
```

Las velocidades de reloj son:

```
HSC   cientos de segundos
TSC   decenas de segundos
SEC   segundos
MIN   minutos
```

Por ejemplo, para la inicialización de la preselección de timer sin una velocidad de reloj:

```
STEP 1              ¡Lo hacemos primero!
IF          NOP     Incondicionalmente
LOAD V100           valor 100... la velocidad del reloj estándar (sin
                    especificar) será en incrementos de 1/100 de segundo
TO          TPO     a la preselección del timer 0 = 1 seg.
```

Se ha inicializado el Timer 0 para tener una duración de 1 segundo (100 x 1/100 segundo). El rango permitido es de 0 - 65535, lo que habilita a períodos del timer entre 0.01s y 655.35 s (aproximadamente, 10 minutos).

Los modelos de controladores que incorporan baterías de resguardo de memoria –*back-up batteries*– mantienen los valores de preselección de los timers durante los períodos en que el controlador está desconectado o apagado. Estos valores también son mantenidos en controladores que no tienen baterías de respaldo pero que tienen tecnologías más modernas como, por ejemplo, FLASHROM, o Zero Power RAM (Estos dispositivos son RAM mantenidos por baterías montadas en el mismo chip).

[3] Dependiendo del modelo de controlador que se esté usando, podría o no requerirse la especificación del reloj, así como el valor del temporizador. Por favor, refiérase al manual del controlador que usted está usando.

2. Establecer una instrucción para arrancar el timer:

Arrancar –*start*– un timer requiere solamente emplear una instrucción SET especificando, además, cuál es el timer a ser puesto en marcha:

Si una instrucción de SET de un timer se ejecuta y el timer especificado ya está funcionando (activo), será rearrancado y un nuevo período de tiempo comenzará.

```
IF          I1.0    Cualquier condición para comenzar
THEN SET    T6      Entonces arrancar timer 6
```

Siempre que la instrucción SET Tn se ejecute, ocurre lo siguiente:

- El valor almacenado en tpn –*timer preselect n* o preselección del timer n– se copia a la twn –timer *word n*–.
- Tn –*timer status n* o estado del timer n– se activa poniéndose en '1' (activo / corriendo).
- El controlador automáticamente decrementa el valor almacenado en Twn a intervalos regulares.
- Cuando el valor almacenado en twn alcanza 0 (cero), Tn –*timer status*– se convierte en '0' (inactivo / detenido).

3. Comprobar el estado del timer: activo / detenido:

Para que los timers puedan ser útiles en el control de procesos, es necesario saber cuándo el tiempo programado ha sido completado. El lenguaje STL permite verificar si el timer está activo, de la misma forma que se hace para verificar si una input está activa.

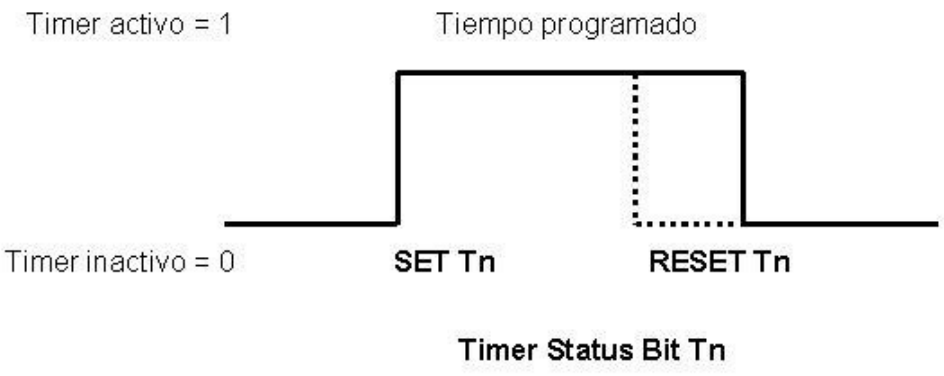
```
      IF    T5    Verifica si el timer 5 está activo (corriendo)
IF    N     T3    Verifica si el timer 3 no está activo (detenido)
```

Detener un timer sólo requiere de una instrucción RESET, además de especificar cuál es el timer que debe detenerse:

```
IF I2.0          Input para detener el timer
THEN RESET T5    Detener timer 5
```

- Si se ejecuta la instrucción RESET Tn, el bit de estado del timer (Tn) se vuelve 0 (inactivo).
- Si el timer ya estaba inactivo, esto no tiene efecto.

Vemos aquí la relación entre las instrucciones bit de estado del timer –*timer status* Bit; Tn–, el SET Tn, el RESET Tn y el período normal de tiempo:



Timer Status Bit Tn

La línea sólida representa una secuencia de temporizado normal en la cual el estado del timer se vuelve activo cuando la instrucción SET Tn se ejecuta; el estado retorna a inactivo cuando el período programado de tiempo se ha completado.

La línea de puntos indica que con una instrucción RESET Tn inmediatamente retorna el estado del timer a inactivo.

Es importante comprender que, al construir programas o *steps* que contengan múltiples sentencias que serán procesadas en una manera paralela (barrido), cada vez que la parte condicional de una sentencia evalúe ésta como verdadera, las instrucciones programadas en la parte ejecutiva serán evaluadas. Esto debe considerarse para evitar ejecuciones múltiples no controladas que incluyan Set timer o INS/DEC Counter Word, SHL, etc.

El lenguaje STL no usa “disparo por flancos”; las condiciones son evaluadas por verdaderas cada vez que se procesan, sin importar su estado previo.

Esta situación es fácilmente manejada usando *steps*, *flags* u otro medio de control. Los siguientes ejemplos muestran dos posibles formas en las cuales estos ejemplos son minimizados.

Podemos evitar arranques indeseados usando la estructura *Step* STL.

Veamos un ejemplo:

Se trata de una sección de programa en la que se desea poner en marcha un motor durante 3 segundos, cada vez que se presiona un botón, si el motor no está ya en marcha y han pasado a por lo menos 9 segundos de la última vez que el motor fue puesto en marcha.

Se ha eliminado en este programa la posibilidad de continuos rearranques de los temporizadores, combinando la estructura *step* del STL con la instrucción N Timer.

```
STEP 1
      IF      NOP          Inicializar al conectar energía
      THEN    LOAD V900  900.  0,1seg unidad de tiempo
              TO         TP0   Timer 0 es 2 seg. tiempo de pausa
              LOAD V300  300   0,1 seg. unidad de tiempo
              TO         TP2   Timer 2 es timer de motor
              SET        T0    Correr pause timer
```

```
STEP 10
      IF    N        T0      Timer 0 ha terminado
            AND   N  T2      Timer 2 no está corriendo
            AND   N  O1.0    Motor no está  en marcha
            AND      I1.2    Botón presionado
      THEN  SET      T2      Arrancar timer
            SET      O1.0    Arrancar motor

STEP 20
      IF             N       T2      Tiempo del motor finalizado
      THEN           RESET   O1.0    Detener el motor
                     SET     T0      Arrancar timer de pausa
                     JMP  TO 10      Arrancar de nuevo
```

También es posible evitar continuos rearranques de timers, al procesar en paralelo.

Para esto, es importante que el programador de STL entienda que el Bit de estado del timer (por ejemplo, T2) puede ser consultado usando las siguientes instrucciones:

```
IF     T2   Esta consulta es verdadera si timer 2 está activo y temporizando
IF N   T2   Esta consulta es verdadera si timer 2 no está activo en este momento
```

Es vital comprender que ninguna de estas instrucciones da información de si el timer 2 ha sido arrancado y se ha completado su temporización. Por ello, al construir programas en STL con sentencias que serán procesadas múltiples veces, es importante tomar las medidas necesarias para evitar resultados inesperados.

El siguiente ejemplo muestra una sección de programa en la que un botón pulsador se usa para hacer que un cilindro extienda su vástago por un tiempo predeterminado –*preset*–.

Manteniendo el botón pulsador presionando, o presionando y soltando el botón múltiples veces dentro del período de tiempo definido, no se altera el tiempo programado.

```
STEP 1                      Inicialización; sólo la primera vez
      THEN      LOAD V0
      TO        OW0         Apagar todas las salidas
      RESET     F3.0        Borrar flag 3.0
      LOAD      V100        Inicializar timer
      TO        TP0         Que el timer T0,  sea de 1 segundo

STEP 2                      Sección de barrido principal
      IF        I1.0        Botón 1 es presionado
      AND   N   T0          y el timer 0 no está corriendo
      AND   N   F3.0        Forma de detección de flanco
      THEN  SET T0          Arrancar timer 0
            SET O1.0        Extender cilindro 1
            SET F3.0        Memorizar flanco positivo de pulsador

      IF    N   T0          Timer 0 no activo
      AND       O1.0        y el cilindro está extendido
      THEN RESET O1.0       luego retraer el cilindro
      IF    N   T0          Timer 0 no activo
```

AND	F3.0	y tuvimos previamente un flanco ascendente
AND N	I1.0	y el botón pulsador es liberado, se detecta el flanco descendente
THEN	RESET F3.0	Estar preparados para el próximo flanco
IF	NOP	Entonces continuar barriendo
THEN JMP TO 2		El *step* actual.

Usando counters[4]

Consideremos algunas funciones básicas para el uso de contadores:

Elemento / Operando	Referencia	Función
Counter Status Bit(Bit de estado delcontador)	Cn	Permite a un programa consultar si el counter es activo (no ha alcanzado su valor final). Este Bit se cambia a activo cuando el counter se arranca –*set*–. Cuando el número de cuentas programadas se alcanza o si es detenido –*reset*–, el bit de estado se vuelve inactivo.
Counter preselect	Cpn	Un operando de 16 Bit que contiene el valor deseado de la cuenta.
Counter Word	Cwn	Un operando de 16 Bit que contiene el número corriente de cuentas almacenadas mediante las instrucciones decrementar o incrementar. Al usar counters estándar y ejecutar la instrucción SET Cn, la counter word es automáticamente cero.

Los modelos de controladores que incorporan baterías de respaldo, mantienen los valores de preselección del counter, bits de estado y words durante los períodos sin energía de red. Igualmente, con los nuevos, equipados con EEPROM o NVRAM.

Vamos a dividir nuestra exposición en dos partes:

- Contadores estándar
- Contadores incrementales / decrementales

- **Contadores estándar:**

Un counter estándar es útil para el conteo de eventos determinados y para llevar a cabo, luego, una acción deseada, cuando se arribe a la cuenta predefinida.

Para operar counters estándar es necesario:

1. Establecer una preselección de counter válida
2. Establecer una instrucción para arrancar el counter
3. Controlar el estado del counter: activado / detenido

1. Establecer una preselección de counter válida:

Antes de que un counter estándar pueda usarse, el respectivo counter *preselect* debe ser inicializado con un valor correspondiente al número de eventos a ser contado.

Esta inicialización sólo necesita ser realizada nuevamente si el valor para actividades de conteo subsecuentes requiere ser cargado. No es necesario cargar de nuevo al counter *preselect* cada vez que el contador es arrancado.

---

[4] No intentamos describir aquí la operación o la implementación de ningún contador de alta velocidad especial ni de aquellos manejados por interruptores, los que están disponibles en algunos modelos de controladores. el uso de counters con estas características especiales puede ser encontrado en el manual de hardware del controlador en cuestión.

El counter *preselect* puede ser cargado con valores absolutos o con el contenido de cualquier MBO (por ejemplo, Registro, Input Word, Flag Word, etc.)

Si inicializamos el counter *preselect* con un valor absoluto:

```
IF                      I1.0    O cualquier condición deseada...
THEN    LOAD V100               Cargamos un valor absoluto de 100
                                como el número de eventos a contar
        TO              CP4     Al counter Preselect 4.
```

Si inicializamos el counter *preselect* con un valor MBO:

```
IF                      I1.0    O cualquier condición deseada
THEN    LOAD  IW1               Input word 1 como el valor de entrada
TO      CP5                     al counter preselect 5
```

Por medio de la instrucción DEB podemos usar llaves externas BCD para establecer la cuenta.

2. Establecer una instrucción para arrancar el counter:

Arrancar un contador sólo requiere de una instrucción SET y especificar qué counter debe arrancarse:

Si una instrucción SET de un counter es ejecutada y el counter especificado está ya activo, es rearrancado y la cuenta actual (en cwn) es puesta a cero (0).

```
IF                      I1.2    Condiciones deseadas
THEN  SET       C2              Activar counter 2
```

Siempre que la instrucción Set Cn es ejecutada, ocurre:

- La respectiva Counter Word (cwn) es cargada con un cero (0).
- Cn (Counter Status n = estado del contador) se vuelve activo (1).

3. Comprobar el estado del counter: activado / detenido:

Para utilizar contadores de forma útil, es necesario poder determinar cuándo la cuenta preseleccionada ha sido alcanzada.

- Una vez que el counter ha sido **activado** –SET–, la cuenta actual es mantenida en la respectiva counter word, la que puede ser actualizada usando tanto la instrucción INC cwn como la DEC cwn.

- Un counter puede ser **detenido** en cualquier momento con la instrucción RESET Cn. Cuando la instrucción RESET Cn es ejecutada, el bit de estado del counter –Counter *Status* Bit Cn– es 0 (cero). El contenido de la counter word permanece sin cambiar.

Consideremos un ejemplo de uso de un counter estándar en conjunción con la estructura *step* para evitar incrementos múltiples descontrolados en los *Step*s 10 y 15:

Un botón pulsador se usa para comenzar un ciclo de máquina. El ciclo arranca una rampa y cuenta las botellas que pasan por un sensor. Una vez que 25 botellas hayan pasado el sensor, la rampa es detenida y un mecanismo posiciona corchos de cierre en cada botella. Finalmente, todos los corchos son presionados en las botellas 2 veces, durante un segundo cada uno.

```
STEP 1                                  Encendido
      THEN      RESET       C0          Contador de botellas
                RESET       C1          Contador de pulsadas
                RESET       O1.0        Apagar rampa
                RESET       O1.1        Apagar presión de corchos
                LOAD                    V25     cuántos para contar
                TO                      CP0     preselección counter 0
                LOAD                    V2      cuántas pulsadas
                TO                      CP2     preselección counter 2
                LOAD                    V100    100 x .01s = 1 segundo
                TO                      TP0     Timer 0 Preselect

STEP 5                                  Esperar botón de marcha
IF                                      I1.0    Botón de marcha
THEN            SET                     C0      Activar contador
                SET                     O1.0    Arrancar rampa

STEP 10                                 Comenzar contando botellas
      IF                                I1.1    una botella fue sensada
      THEN      INC                     CW0     incrementar contador de botellas

STEP 15                                 ¿25 botellas ya?
            IF      N                   C0      hemos hecho todo, entonces...
            THEN    RESET               O1.0    detener rampa
                    SET                 C2      activar contador de pulsadas
                    JMP   TO            50      salir lazo de conteo
            OTHRW                       sino

            IF              N           I1.1    Esperar por la última botella contada;
                                                que se mueva fuera del sensor
            THEN            JMP TO      10      Y seguir contando

STEP 50                                 25 botellas fueron contadas
      THEN      SET         O1.1        Presionar los corchos
                SET         T2          Arrancar timer de presionado
                INC         CW2         contar este presionado

STEP 60                                 Timer espera 1 segundo
      IF        N           T2          tiempo finalizado
      THEN      RESET       O1.1        detener el presionando

STEP 70                                 ¿Hecho?
      IF        N           C2          corchos presionados 2 veces
      THEN      JMP   TO    5           volver a Step 5
      OTHRW     JMP   TO    50          presionar de nuevo
```

En programas o *steps* que contienen múltiples sentencias que son procesadas en paralelo (modo barrido), cada vez que la parte condicional de una sentencia es evaluada como verdadera, las instrucciones programadas en la parte ejecutiva son ejecutadas.

Esto debe ser considerado para evitar ejecuciones múltiples descontroladas de instrucciones que incluyan SET TIMER o INC / DEC Counter Word, SHL, etc.

El lenguaje STL no usa "disparo por flancos": las condiciones son evaluadas por verdaderas cada vez que son procesadas, sin importar sus estados previos.

El próximo ejemplo es el de un contador estándar en una sección de programa tipo paralelo en la que la estructura del *Step* no es usada para evitar incrementos múltiples e incontrolados del counter. En cambio, se usa una solución alternativa mediante un Flag:

El programa espera por el botón de marcha y, entonces, cicla un cilindro entre las posiciones completamente extendida y completamente cerrada, 100 veces.

Sin el uso del Flag, un programa de barrido incrementaría el counter en cada barrido –*scan*– del programa más que en cada vez que el cilindro fue nuevamente extendido.

STEP 1				**Inicialización 1 vez solamente**
	IF		I1.0	Botón de marcha apretado
	THEN	LOAD	V0	
	TO	OW1		Todas las outputs apagadas
		RESET	F3.0	Borrar flag 3.0
		LOAD	V100	Inicializar timer
		TO	CP0	Hacer un contador de ciclo de 100
		SET	C0	Arrancar el counter
STEP 2				**Sección principal de barrido**
	IF	AND	I1.1	Cilindro retraído
		AND	C0	y el Counter 0 está activo
		AND N	F3.0	Para detectar flanco
		AND N	O1.0	Válvula para extender cilindro apagada
	THEN	SET	O1.0	Comenzar a extender cilindro
		SET	F3.0	Perfecto para ver una nueva extensión
		IF	I1.2	Cilindro está extendido,
		AND	F3.0	Nuevo flanco
	THEN	INC	CWO	Contar el ciclo
		RESET	F3.0	Actualizar control de flancos
		RESET	O1.0	Comenzar retraer cilindro
	IF	N	C0	100 cuentas fueron hechas
	THEN	JMP TO	1	Comenzar todo de nuevo

- **Contadores incrementales / decrementales:**

Además de usar los contadores estándar que ya describimos, el lenguaje STL, a través del uso de operandos multibit, crea counters comúnmente denominados como *Up*/*Down* –arriba/abajo, incrementales/decrementales– que se pueden armar con cualquier operando multibit tal como la counter word, registros, etc.

A diferencia de los counters estándar, no hay necesidad de inicialización un counter preselect y no existe bit dedicado al estado del contador. Por ello, tampoco son aplicables las instrucciones SET RESET.

Los *step*s siguientes son los requeridos para usar este tipo de contador:

- Inicializar el apropiado MBO
- El MBO pude ser Incrementado o decrementado
- El MBO puede ser comparado a un valor o a otro MBO

Consideremos un ejemplo de uso de un registro como counter:

En el siguiente ejemplo un proceso es arrancado y corre hasta que 100 partes de buena calidad son producidas.

```
STEP 10                          Esperar por la señal de marcha
     IF                  I1.0    Botón de marcha
     THEN LOAD           V100    Número a producir
          TO             R50     Registro 50 es el contador
          SET            O1.1    Arrancar la máquina

STEP 20                          Mirar por cualquier parte
                                 En él área de calidad
     IF                  (I1-1   Listo para verificar
          AND            I2.3 )  Calidad es buena
     THEN      DEC       R50     1 buena, 1 menos para hacer
          JMP TO         30      Continuar en Step 30
     IF                  (I1.1   Listo para verificar
          AND   N        I2.3 )  Calidad bien sensor
                                 Perdidos = mal
     THEN      NOP               No cuenta las malas

STEP 30                          Ver si ya tenemos las 100
     IF                  (R50
                         = V0)   Trabajo terminado
     THEN RESET          O1.1    Detener la máquina
          JMP   TO       10      Volver al comienzo
     OTHRW NOP                   O si no está hecho, seguir

STEP 40                          Esperar por la última parte
                                 Para mover
     IF              N   I1.1    Área de calidad limpia
     THEN JMP TO         20      Seguir corriendo / verificando
```

Usando registros[5]

Los controladores programables de *Festo*® que pueden programarse usando el lenguaje STL poseen registros de 16 Bit, aún cuando la cantidad exacta de estos registros varía de acuerdo al modelo.

Estos registros son operandos multibit y pueden usarse para almacenar números en el rango de:

- 0 - 65535 enteros sin signo
- +/- 32767 enteros con signo

Se utilizan los registros en conjunción con la instrucción LOAD TO y operaciones lógicas multibit.

Si el modelo de controlador que usted está usando incluye batería de respaldo o algún sistema de memoria permanente, los contenidos de los registros serán mantenidos mientras no haya energía conectada al equipo. Los registros que no hayan sido inicializados contendrán valores aleatorios (en el sistema no existen rutinas de limpieza).

[5] *Aquí se explica el concepto de registro, como se ha implementado en los controladores programables de Festo®.*

Los registros no son direccionables en una base bit por bit; si este tipo de acceso es necesario, los flag words son los aconsejados para la tarea.

Los registros pueden usarse para simplificar el control múltiple de procesos secuenciales dentro de una única sección de programa barrida.

Para usar registros en la parte condicional de una sentencia:

```
IF      R51     Si el contenido del registro 51
        = V111) Es  igual a 111
AND     T7      Y el timer 7 está corriendo
AND     R3      Y el registro 3 es menor que
    <   R8)     El registro 8
THEN...         Hacer lo programado...
```

Para usar registros en la parte ejecutiva de una sentencia:

```
IF                          Condiciones programadas
THEN    LOAD    R12     Load el contenido del registro 12 al MBA
        +       R50     Sumar el contenido del registro 50
        TO      R45     Y almacenar el resultado en registro 45
```

Usando flags y flag words

Se describe a continuación la construcción lógica, y uso de flags y flag words para los controladores programables de *Festo*®.

Las flag words son, de muchas formas, casi idénticas a los registros. Las flag words contienen, cada una, unidades de 16 bits de información. Dentro del lenguaje STL se usa la abreviatura FW.

Si el modelo del controlador que usted está usando incluye memoria del tipo FLASHRAM o ZPRAM, o en su defecto batería de respaldo, el contenido será mantenido durante los períodos de corte o interrupción de la alimentación de energía. Aquellos flags que nunca hayan sido inicializados contendrán valores aleatorios.

Los flag words son capaces de almacenar datos numéricos dentro del rango:

- 0 - 65535 enteros sin signo
- +/- 32767 enteros con signo.

Los flag words difieren de otros operandos multibit en varias formas importantes:

La mayor diferencia entre flags y otros operandos multibit tales como registros, counter words, etc., es que cada 16 Bit Flag Word es también direccionable de a un bit (Por ejemplo, el FPC100 contiene 16 flag words, direccionadas como FWO a FWI 5).

Es también posible direccionar bits individuales –flags– de cada flag word, mediante la siguiente sintaxis:

F(número de Flag Word), número de BIT

Donde el número de bit varía entre 0 y 15.
(Por ejemplo, F7.14 hace referencia al bit 14 del flag 7)

Este esquema de direccionamiento es muy similar al usado cuando se accede a puntos de entrada/salida (I/O) digitales, como ya hemos descripto.

Mientras que los flag words pueden usarse con cualesquiera instrucciones STL para operandos multibit, los flags individuales son sólo accesibles usando instrucciones STL diseñadas para operandos de bit.

Los flags de bit único son, a menudo, usados como medio conveniente para memorizar eventos. En este aspecto son similares a "bobinas internas o relés" encontrados a menudo en el diagrama escalera (Ladder Diagram).

El FPC405, que soporta múltiples módulos de CPU (multiprocesamiento), permite que cualquier programa en cualquiera de las CPU acceda a los flags de FWO a FW23 (FW externa) desde cualquiera otra CPU. Esto es, cada CPU es capaz de leer desde o escribir a los flags de cualquier otra CPU.

Por lo tanto, los flags pueden dar un medio conveniente para implementar comunicaciones entre CPU.

En sistemas con múltiples CPU, cada Word es direccionada como:

CPU número, flag word número
(Por ejemplo, FW2.14 se direcciona flag word 14 en CPU 2)

De la misma manera, es posible direccionar flags monobit en otras CPU, extendiendo la sintaxis de la dirección:

CPU número. F(número de flag word).Bit número
(Por ejemplo, F0, 11.9 se refiere al flag bit 9 en la flag word 11 localizada en la CPU 0)

Flags individuales (así como también flag words) pueden programarse tanto en la parte condicional como en la ejecutiva de una sentencia. En la parte condicional, los flags pueden ser interrogados por sus estados (0 = RESET, 1 = SET); mientras que los flags words pueden ser comparadas a valores u otros MBO's –*multibit operandos*–.

Consideremos el modo de programación en la parte condicional:

```
IF          F1.1   Si el Bit 1 de la flag word 1 es SET
IF          F2.1   Si el Bit 1 de la flag word 2 es SET
AND  N      F4.0   y el Bit 0 de la flag word 4 es No SET.
```

Igual que con los otros operandos monobit o multibit, los flags pueden combinarse con otros operandos:

```
IF          I3.0   Si input 3.0 es válida
AND         F0.0)  y flag 0.0 es SET
OR     ((   FW3    o el valor de todos los 16 bits del Flag Word 3
       =    V500)  son iguales a 500
 AND   N    T7)    y el timer 7 no está activo
```

Ejemplos de la parte ejecutiva:

```
IF          I1.1   Si input 1.1 es válida entonces
THEN
       SET  F2.2   SET bit 2 del flag word 2
IF          T6     Si T6 en la CPU local está corriendo
```

```
THEN
                SET   F3.3   SET   Flag 3.3 de modo que otra CPU pueda
                                   verificar el estado de T6
OTHRW           RESET F3.3
```

En la parte ejecutiva de las sentencias, los flag words pueden usarse como la fuente o destino de cualquier instrucción multibit.

Como los flags puedan direccionarse sobre la estructura de una word, así como desde el punto de vista de una estructura de bits, éstos resultan como un medio conveniente para construir registros de desplazamiento –*shifl registers*–-.

Supongamos que es necesario programar una línea de maquinado en la cual filas de moldes sean cargadas en la estación 0 y, a continuación, varias operaciones deban ser llevadas a cabo a lo largo de las siguientes 15 estaciones. La máquina completa cicla cada 2 segundos y durante ese tiempo una nueva fila de moldes puede o no ser cargada en la estación 1..., lo que puede verificarse por medio de un sensor.

Las estaciones 1-15 no incluyen sensores, pero se requiere que la estación opere solamente cuando haya una parte en el lugar.

Esto presenta una situación ideal para usar un registro de desplazamiento.

Usaremos flag word 6 para tener el seguimiento de cuáles son las estaciones que contienen materiales a ser maquinados. La instrucción SHL –*Shifl Left*–; desplazar a la izquierda– se usará para mover realmente los bits individuales dentro del flag word.

Se usarán las siguientes I/O:

```
                Input 1.0                Botón de marcha
                Input 1.1                Sensor de partes en la estación 0
                Input 2.2                Se indexa línea transfer
                Output 2.0               Indexa línea de maquinado
                Outputs 1.0 – 1.15       Controla la operación de maquinado
                                         en las estaciones 0-15, respectivamente

STEP 10                                  Arranque
    IF                  I1.0             Botón de arranque
        THEN
        AND
        LOAD
        I2.2
                                V200     La línea es indexada 2 segundos
        TO                      TPO      a la preselección timer 0
        LOAD                    V0       Supone nueva producción corriendo
        TO                      FW6      No partes en ninguna estación

STEP 15                                  Esperar hasta que algunas partes estén listas
    IF                          I1.1     Parte fue encontrada en estación 0
    THEN        SET             F6.0     Memorizarlo
    IF                          (FW6     ¿Alguna parte para procesar?
                >               V0 )     ¡Algo existe!
```

```
      THEN    LOAD    FW6
              TO      OW1   Arrancar motor en estaciones con partes
              SET     T0    Arrancar timer proceso

STEP 20                     ¿Timer de maquinado hecho?
      IF      N       T0    Tiempo terminado
      THEN    LOAD    V0    Apagar todos los motores de la estación
              SET     O2.0  Arrancar línea indexado

STEP 25                     Esperar hasta que el index arranque
      IF      N       I2.2  Arrancado al index
      THEN    LOAD    FW6   Obtener el estado todas estacionas
              SHL           Mover bits para adaptar partes
              TO      FW6   Y almacenarlo

STEP 30                     ¿Está completo el index?
      IF                    Nuevo punto de index
      THEN    RESET   O2.0  Detener Index del motor
              JMP TO  15    Volver al Step 15 por más
```

Funciones especiales del lenguaje STL

Aquí vamos a suministrarle información básica concerniente a:

- Entradas / salidas analógicas
- Redes
- Control de posicionamiento
- Field bus

Algunas de estas funciones pueden no aplicarse a todos los modelos de controladores y pueden manejarse de diferentes formas, dependiendo del modelo de controlador.

- **Entradas / salidas analógicas:**

Contrariamente a las entradas / salidas digitales (1 o 0), las señales analógicas toman la forma de una señal variable continuamente dentro de un rango predefinido.

Ya que la CPU sólo es capaz de funcionar internamente usando señales digitales, conectar un PLC a entradas analógicas o salidas analógicas requiere componentes de hardware especial.

Hay varios rangos o tipos de señales analógicas que son populares en el control industrial. Si excluimos señales analógicas especializadas del tipo relacionado al control de temperatura, los rangos comunes que quedan son:

- +/- 10 volts
- 0/4 a 20 miliamperes de corriente

Para que las entradas y salidas analógicas sean útiles, el software de programación debe suministrar los medios de llevar a cabo las funciones deseadas.

Las funciones analógicas básicas usadas en control industrial incluyen:

- Setear un nivel analógico de salida basado en un valor digital
- Convertir una señal de entrada analógica en un valor digital

Para efectuar estas funciones dentro del lenguaje de programación del PLC usado, en general, se integran procedimientos especializados CFM o FN[6].

- **Redes:**

Redes

En el contexto de este material de capacitación y del lenguaje STL, se refiere al hardware y software que proporcionan los medios para interconectar sistemas de control que, de otra forma, serían unidades independientes.

Una red es típicamente empleada para conectar varios elementos de un sistema de procesamiento distribuido en el cual cada subsistema controla una específica sección, física o lógica, de la tarea total.

El uso de una red facilita a estas secciones combinarse de una manera ordenada. Sin importar el lenguaje de programación a ser usado, para implementar la red se requiere hardware especializado así como software de red.

Dependiendo del modelo de controlador, el hardware especializado puede tomar la forma de un procesador de red o un módulo que contiene las rutinas especializadas de software que pueden ser direccionadas por el lenguaje STL.

Las funciones típicas que deben ser llevadas a cabo en una red incluyen:

- Inicializar estaciones de red.
- Requerir a otra estación ejecutar un comando.
- Gerenciar transmisiones de red.

La interfase entre el lenguaje STL y el software especializado de red es efectuada mediante la instrucción CFM.

- **Control de posicionamiento**:

Puede ser necesario controlar rápidamente y con precisión la posición de componentes mecánicos como partes de un sistema de control. Tales movimientos son llevados a cabo, generalmente, usando varios tipos de motores, dependiendo de los requerimientos de:

- velocidad,
- precisión,
- efectividad del costo,
- confiabilidad.

Diversos tipos de motores están disponibles. Estas elecciones incluyen motores de pasos y servomotores, así como motores de múltiple velocidad, los que pueden incorporar componentes de frenado. Variaciones adicionales pueden o no incorporar control a lazo cerrado.

Las más precisas (y costosas) soluciones tales como servomotores, incorporan microprocesadores dedicados; mientras que soluciones menos sofisticadas pueden confiar en la velocidad e inteligencia del controlador programable.

---

[6] *La información general sobre los CFM* –Call Function Module– *puede ser leida en los manuales técnicos de cada Controlador Lógico Programable, puesto que, de acuerdo con la marca y modelo de cada PLC, éstas trabajan de manera distinta.*

A causa de la amplia variedad de subsistemas de posicionamiento que pueden conectarse, no hay instrucciones STL dedicadas para posicionar[7].

- **Field Bus:**

El field bus está basado en el estándar eléctrico RS485, el cual define los parámetros de la estructura de un bus serie de alta velocidad.

Debe hacerse una diferenciación entre los elementos que forman la estructura del bus interna del controlador y el sistema de field bus.

Las I/O estándar están conectadas de forma muy cercana, tanto eléctrica como físicamente a la estructura de bus paralela interna del controlador.

Mientras que esta estructura provee accesos de alta velocidad, su naturaleza pone límites al hardware en cuanto al número de I/O únicas direccionables posibles.

El concepto de field bus utiliza el antes mencionado bus serie para unir una estación maestra –Master– y múltiples estaciones esclavas, a velocidades de transmisión hasta 375,000 bits por segundo.

Ya que estas estaciones pueden localizarse en relativamente largas distancias (300-1200 metros), éstas son a menudo genéricamente referidas como “I/O Remotas”.

Las altas velocidades de transmisión, cuando se combinan con el ahorro en costos al usar un par de cables trenzados para la conexión del bus, hacen que el concepto de field bus se vuelva realmente atractivo.

Inputs y outputs localizadas en las estaciones esclavas del field bus pueden ser interrogadas y controladas por la estación field bus master. El lenguaje STL permite acceder a estas I/O usando las mismas instrucciones SET y RESET que usan las I/O estándar ubicadas físicamente en el controlador

Para acomodar la configuración extendida usando el field bus, la sintaxis de instrucciones I/O debe también ser extendida. Inputs y outputs son, por lo tanto, direccionadas de la siguiente forma:

- Ipa[,m.s] para Inputs
- Opa[.m.s] para Outputs

Donde:

P = Dirección de sistema del procesador master del field bus
A = Dirección estación esclava de field bus [1... 99]
M = Dirección de modulo opcional [0... 15]
S = Número de terminal opcional [0...15]

[7] *Sin embargo, Festo® suministra programas especializados y módulos de programa que han sido optimizados y/o adaptados para control de posición.*

Programas de ejemplo

Aquí presentamos algunos ejemplos de problemas de control y soluciones usando el lenguaje STL. Los ejemplos planteados son generales, para que puedan resultar útiles al lector, sin importar que modelo de controlador en particular que vaya a usar.

La mayoría de las tareas de control se divide en tres categorías:

- Completamente secuencial
- Mayormente secuencial con algunos eventos aleatorios
- Completamente aleatorio

Además, en muchas situaciones, se presenta la situación de tener que manejar varias secuencias de control simultáneamente.

- **Ejemplo 1. Completamente secuencial:**

Las tareas completamente secuenciales son especialmente aptas para el lenguaje STL a causa de la estructura implícita del *step*.

La tarea secuencial consiste en controlar 3 cilindros neumáticos por medio de 3 válvulas solenoide 3/2, en una secuencia definida.

Cuando se aplica alimentación de energía al sistema y el botón de arranque es presionado, el cilindro A debe extenderse completamente por 3 segundos y luego retraerse.

A continuación, el cilindro B debe extenderse completamente y retraerse cuatro veces; y, luego, extenderse completamente y permanecer extendido.

Finalmente, el cilindro C debe extenderse completamente, en cuyo momento el cilindro A debe otra vez extenderse.

Después, el cilindro A es otra vez extendido y todos los tres cilindros se retraen y esperan la presión del botón de arranque.

Se utilizan las siguientes conexiones:

Input 1.0	Botón de arranque
Lnput 1.1	Cilindro A retraído
Lnput 1.2	Cilindro A extendido
Lnput 1.3	Cilindro B retraído
Lnput 1.4	Cilindro B extendido
Lnput 1.5	Cilindro C retraído
Lnput 1.6	Cilindro C extendido
Output 1.0	Cilindro A extendido (solenoide)
Output 1.1	Cilindro B extendido (solenoide)
Output 1.2	Cilindro C extendido (solenoide)

Resolución:

STEP 1				**Inicialización en el arranque**
	IF	NOP		Hacer siempre esto
	THEN	LOAD	V0	Incondicionalmente apagar todas las
	TO		OW1	Outputs
		LOAD	V300	Preparar timer 0 para 3 segundos
		TO	TP0	Unidades = 0.01 segundos
		LOAD	V4	Preparar counter 2
		TO	CP2	
STEP 5				**Asegurarse todas las posiciones ok**
	IF		I1.0	Botón arranque presionado
		AND	I1.1	Cilindro A está retraído
		AND	I1.3	Cilindro B está retraído
		AND	I1.5	Cilindro C está retraído
	THEN	SET	OI.0	Comenzar extendiendo cilindro A
STEP 10				**Cilindro A, ¿completamente extendido?**
	IF		I1.2	Ahora está totalmente extendido
	THEN	SET	T0	Arrancar timer 3 segundos
STEP 12				**Esperar 3 segundos**
	IF	N	T0	Si timer completó tiempo
	THEN	RESET	O1.0	Comenzar retraer cilindro A
STEP 15				**Cilindro A, ¿completamente retraído?**
	IF		I1.1	Cilindro A está retraído
	THEN	SET	C2	Set counter 2 - 4 cuentas
		SET	O1.1	Comenzar extender cilindro B
STEP 20				**Cilindro B, ¿completamente extendido?**
	IF		I1.4	Ahora está completamente extendido
	THEN	INC	CW2	Contar este ciclo
		RESET O1.1		Comenzar retraer cilindro B
STEP 22				**¿Es ésta la 4° extensión ?**
	IF		I1.3	Cilindro B retraído y 4
		AND	C2	Carreras no terminado
	THEN	SET	O1.1	Comenzar a extender cilindro B
	JMP	TO	20	Ciclos continuos
	IF		I1.3	Cilindro B retraído y 4
	AND	N	C2	Carreras hechas
	THEN	SET	O1.1	Comenzar a extender cilindro B
STEP 30				**Cilindro B, ¿completamente extendido?**
	IF		I1.4	Cilindro B completamente extendido 5 x
	THEN	SET	O1.2	Comenzar extendiendo cilindro C
STEP 35				**Cilindro C, ¿completamente extendido?**
	IF		I1.6	Cilindro C completamente extendido
	THEN	SET	O1.0	Comenzar extendiendo cilindro A

```
STEP 40                                  ¿Todos cilindros extendidos?
      IF                    I1.2         Cilindro A completamente extendido también
      THEN    RESET O1.0                 Retraer Cilindro A
              RESET O1.1                 Retraer Cilindro B
              RESET O1.2                 Retraer Cilindro C
              JMP   TO  5                Volver al Step 5
```

- **Ejemplo 2. Mayormente secuencial con eventos aleatorios:**

Mientras que algunas máquinas simples pueden ser completamente secuenciales en la operación, existen una o más excepciones que cambian la clasificación de las tareas, de modo de no ser ésta totalmente secuencial.

Si la mayoría de las tareas de control fuera secuencial y el modelo del controlador lógico permitiese multitarea, una solución posible es la de dividir la parte secuencial y los procesos de procesamiento aleatorio en programas separados. Sin embargo, es posible manejar esas situaciones con un único programa STL.

Si el evento aleatorio o los eventos aleatorios a ser monitoreados son pocos y el balanceo de los programas es relativamente simple, entonces podrían manejarse los requerimientos adicionando una sentencia de programa en cada *step*[8].

Vamos a insertar una sentencia de programa en cada *step* existente del programa presentado en el ejemplo 1, como medio de detectar y responder a un simple botón pulsador de "pausa"; el cual al ser presionado resulta en la suspensión de la ejecución del programa hasta que éste esté liberado nuevamente.

Resolución:

```
STEP 1                                   Inicialización en el arranque
      IF        NOP                      Hacer siempre esto
      THEN      LOAD        V0           Incondicionalmente apagar todas las
      TO                    OW1          Outputs
                LOAD        V300         Preparar timer 0 para 3 segundos
                TO          TP0          Unidades = 0.01 segundos
                LOAD        V4           Preparar Counter 2
                TO          CP2

STEP 5                                   Asegurarse todas las posiciones ok
      IF                    I1.0         Botón arranque presionado
      AND                   I1.1         Cilindro A está retraído
      AND                   I1.3         Cilindro B está retraído
      AND                   I1.5         Cilindro C está retraído
      AND       N   I1.7                 Botón de pausa no activo
      THEN      SET         OI.0         Comenzar extendiendo cilindro A
```

[8] *Otras posibles soluciones incluyen el uso de procesamiento por interrupciones (sólo soportada en algunos controladores) o construyendo la secuencia entera como una sección de programa en paralelo (barrido). Retomaremos esta resolución en el ejemplo 3.*

STEP 10				**Cilindro A, ¿completamente extendido?**
	IF		I1.7	Botón de pausa
	THEN	JMP TO	10	Si es así, quedarse acá
	IF		I1.2	Ahora completamente extendido
	THEN	SET	T0	Arrancar timer 3 segundos
STEP 12				**Esperar 3 segundos**
	IF		I1.7	Botón de pausa
	THEN	JMP TO	12	Si es así, quedarse acá
	IF	N	T0	Si timer completó tiempo
	THEN	RESET O1.0		Comenzar retraer cilindro A
STEP 15				**Cilindro A, ¿completamente retraído**?
	IF		I1.7	Botón de pausa
	THEN	JMP TO	15	Si es así, quedarse acá
	IF		I1.1	Cilindro A está retraído
	THEN	SET	C2	Set counter 2 - 4 cuentas
		SET O1.1		Comenzar extender cilindro B
STEP 20				**Cilindro B, ¿completamente extendido?**
	IF		I1.7	Botón de pausa
	THEN	JMP TO	20	Si es así, quedarse acá
	IF		I1.4	Ahora está completamente extendido
	THEN	INC	CW2	Contar este ciclo
	RESET	O1.1		Comenzar retraer cilindro B
STEP 22				**¿Es ésta la 4° extensión ?**
	IF		I1.7	Botón de pausa
	THEN	JMP TO	22	Si es así, quedarse acá
	IF		I1.3	Cilindro B retraído y 4
	AND	C2		Carreras no terminado
	THEN	SET	O1.1	Comenzar a extender cilindro B
	JMP	TO	20	Ciclos continuos
	IF		I1.3	Cilindro B retraído y 4
	AND	N	C2	Carreras hechas
	THEN	SET	O1.1	Comenzar a extender cilindro B
STEP 30				**Cilindro B, ¿completamente extendido?**
	IF		I1.7	Botón de pausa
	THEN	JMP TO	30	Si es así, quedarse acá
	IF		I1.4	Cilindro B completamente extendido 5 x
	THEN	SET	O1.2	Comenzar extendiendo cilindro C
STEP 35				**Cilindro C, ¿completamente extendido?**
	IF		I1.7	Botón de pausa
	THEN	JMP TO	35	Si es así, quedarse acá
	IF		I1.6	Cilindro C completamente extendido
	THEN	SET	O1.0	Comenzar extendiendo cilindro A
STEP 40				**¿Todos cilindros extendidos?**
	IF		I1.7	Botón de pausa
	THEN	JMP TO	40	Si es así, quedarse acá
	IF		I1.2	Cilindro A completamente extendido también

THEN	RESET	O1.0	Retraer Cilindro A
	RESET	O1.1	Retraer Cilindro B
	RESET	O1.2	Retraer Cilindro C
	JMP TO	5	Volver al *Step* 5

En resumen, es posible manejar cantidades limitadas de condiciones en paralelo, dentro de lo que de otra forma sería un estricto proceso secuencial, usando la instrucción *Step*.

- **Ejemplo 3. Eventos completamente aleatorios:**

Algunas situaciones de control no pueden organizarse en una secuencia lógica ya que las operaciones pueden surgir en un orden totalmente aleatorio.

Un ejemplo típico de una situación así podría ser el control de la programación de puesta a punto –*setup*– de una máquina cualquiera. La operación es definida por el operador de la máquina presionando distintos botones del tipo pulsador, cada uno de los cuales tiene asignada una función única.

El siguiente programa desarrolla el programa de Setup para una máquina de moldeo de inyección de plástico.

Resolución:

STEP 10			**inicialización**
IF	NOP		Siempre cierto
THEN	LOAD	V0	
TO	OW1		Apagar todas outputs
STEP 20			**Step de barrido**
IF		I1.0	Botón pulsador cierre molde
THEN	SET	O1.0	Cierre de molde
IF		I1,1	Botón pulsador inyecta plástico
	AND	I2.0	Sensor molde cerrado
THEN	SET	OI.3	Solenoide inyección
OTHRW	RESET	OI.3	
F		I1.2	Botón pulsador apertura molde
AND N	OI.3	Inyección no activa	
THEN	RESET	O1.0	Apertura molde
IF		I1.3	Mecanismo rota tornillo
THEN	SET	O1.1	Solenoide rota tornillo
OTHRW	RESET	O1.1	Mecanismo detención tornillo
IF		I1.4	Sensor molde completamente abierto
	AND	I1.5	Botón pulsador eyector de molde
THEN	SET	O1.4	Solenoide eyector de molde
OTHRW	RESET	O1.4	Proceso de detención de eyección
IF	NOP		Hacer siempre
THEN	JMPTO	20	Seguir procesando

OTROS LENGUAJES DE PROGRAMACIÓN

Anexo 2 / Estructura del lenguaje GRAFCET

GRAFCET –gráfico de mando etapa transición– Es un diagrama funcional que describe la evolución del proceso que se pretende automatizar, indicando las acciones que hay que realizar sobre el proceso y qué informaciones las provocan; partiendo de él se pueden obtener las secuencias que ha de realizar el autómata programable.

El empleo del lenguaje GRAFCET para resolver tareas de automatización facilita el diálogo entre personas con niveles de formación técnica diferente, tanto en el momento del análisis del proceso a automatizar como, posteriormente, en el mantenimiento y reparación de averías.

El GRAFCET surge en Francia a mediados de los años '70, debido a la colaboración entre algunos fabricantes de autómatas, como *Telemecanique* y *Aper* con dos organismos oficiales, AFCET –Asociación Francesa para la Cibernética, Economía y Técnica– y ADEPA –Agencia Nacional para el Desarrollo de la Producción Automatizada–. Homologado en Francia –NFC–, Alemania –DIN– y, posteriormente, por la Comisión Electrotécnica Internacional (norma IEC 848, año 1988) y regulado por la Norma francesa (NF C03-190).

Actualmente es una herramienta imprescindible cuando se trata de automatizar procesos secuenciales de cierta complejidad con autómatas programables.

A continuación se describen los símbolos normalizados utilizados en el GRAFCET en lo que hace a:

- etapas y
- condiciones de transición.

- **Etapas:**

Para representar la evolución de un proceso con GRAFCET, se considera que el proceso a automatizar y el autómata que se emplea como controlador forman un solo sistema.

El nexo de unión entre las actuaciones que hay que hacer sobre el proceso (activar un motor, cerrar una válvula, etc.) y el programa de usuario, cargado en el autómata, que da origen a aquellas es la **etapa**.

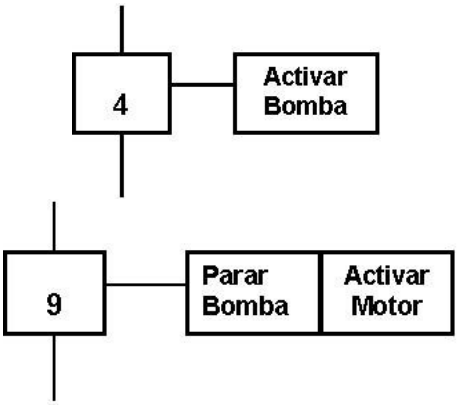
Por tanto, la representación gráfica de la evolución de un proceso con GRAFCET está formada por una serie de etapas, y cada una de ellas lleva asociada una o varias **acciones** a realizar sobre el proceso.

Las etapas se representan con un cuadrado y un número, o una E con un número como subíndice; en ambos casos, el número indica el orden que ocupa la etapa dentro del GRAFCET. Para distinguir el comienzo del GRAFCET, la primera etapa se representa con un doble cuadrado.

Representación de etapas

Representación de etapas iniciales

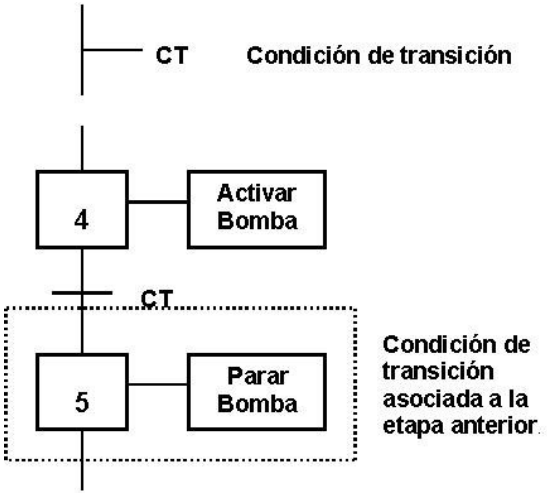
Las acciones que llevan asociadas las etapas se representan con un rectángulo donde se indica el tipo de acción a realizar.



- **Condiciones de transición:**

Un proceso secuencial se caracteriza porque una acción determinada se realiza en función del resultado de la acción anterior.

En GRAFCET, el proceso se descompone en una serie de etapas que son activadas una tras otra. Por tanto, tiene que existir una condición que se ha de cumplir para pasar de una a otra etapa; se la llama **condición de transición –CT–**.



En la figura se representan dos etapas y una condición de transición entre ellas. Para que el proceso evolucione de la etapa 4 a la etapa 5, es necesario que la etapa 4 esté activa y, además, que se cumpla la activación de la condición CT; entonces, se produce la activación de la etapa 5.

Sólo puede existir una etapa activa; por tanto, cuando se produce la activación de la etapa 5 se desactiva la etapa 4. La condición de transición está siempre asociada a la etapa posterior –en este caso, a la 5–.

La condición de transición puede ser una o varias de las variables que intervienen en el proceso (por ejemplo, una señal de un final de carrera, la activación de un motor, un tiempo, etc.)

Para la condición de transición se emplea lógica positiva y podemos tomar los dos valores CT=1 y CT=0; a continuación se indican algunos ejemplos.

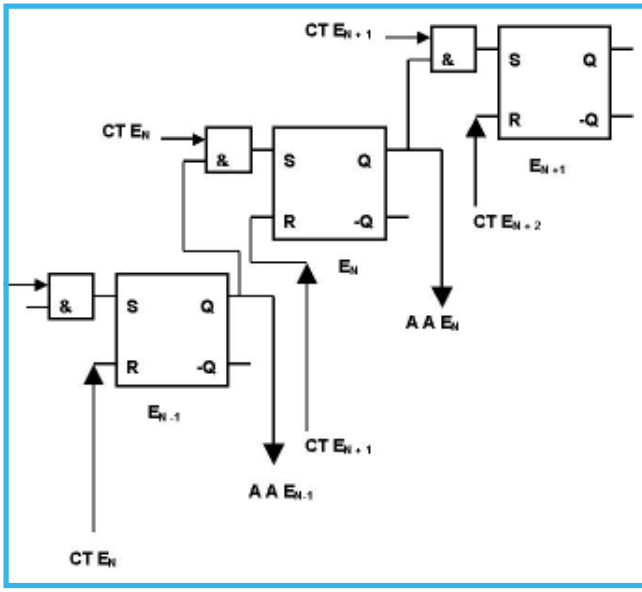
- Condición activa: CT = F1. La activación de la etapa 5 se produce cuando el final de carrera F1 está activado.
- Condición inactiva: CT = NF1. La activación de la etapa 5 se produce cuando el final de carrera F1 esta inactivo.
- Condición por tiempo: CT = t/3/10 seg. La activación se produce cuando el temporizador activado en la etapa 3 alcanza los 10 segundos.
- Condición de varias variables: CT = F1; NF2; F3. La activación se produce si los finales de carrera F1 y F3 están activos y F2 está inactivo.
- Condición incondicional: CT = 1. La activación de la etapa 5 se produce al activarse la etapa 4.
- Condición flanco descendente: CT = A1 '!. La activación se produce cuando la señal A1 pasa de 1 a 0.
- Condición flanco ascendente: CT = A1 "!. La activación se produce cuando la señal A1 pasa de 0 a 1.

Podemos resumir una serie de reglas básicas que hay que tener en cuenta para aplicar GRAFCET:

- El proceso se descompone en etapas, que son activadas de forma secuencial.
- Una o varias acciones se asocian a cada etapa. Estas acciones sólo están activas cuando la etapa está activa.
- Una etapa se hace activa cuando la precedente lo está y la condición de transición entre ambas etapas ha sido activada.
- La activación de una condición de transición implica la activación de la etapa siguiente y la desactivación de la precedente.
- La etapa inicial E_0 tiene que ser activada antes de que se inicie el ciclo del GRAFCET.
- Un ciclo está formado por todas las etapas posteriores a la etapa inicial.

Ecuaciones lógicas

Una vez representado, el GRAFCET permite obtener las ecuaciones lógicas que controlan la activación de cada etapa y la evolución del ciclo. Una de las formas de obtener las ecuaciones se basa en el funcionamiento de un controlador asíncrono con biestables R-S.



Suponiendo que el biestable E_{N-1} tiene su salida Q a "1", la etapa E_{N-1} está activa. Si, posteriormente, la condición de transición de la etapa E_N se activa, la etapa E_N se activará y se desactivará la etapa E_{N-1}. Para desactivar la etapa E_N y activar la etapa E_{N+1}, es necesario activar la condición de transición E_{N+1}. Mientras las etapas están activas (Q = 1), las acciones que llevan asociadas también lo están.

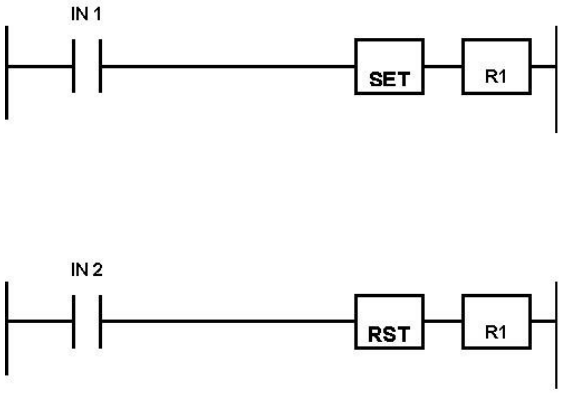
Consideraremos, a continuación:

- Instrucciones Set y RESET.
- Elección condicional entre varias secuencias.
- Secuencias simultáneas.
- Salto condicional a otra etapa.

- **Instrucciones SET y RESET:**

Para utilizar este sistema en el GRAFCET, se asocia a cada una de las etapas una variable interna. La condición de transición, situada entre dos etapas, es la encargada de activar la etapa posterior y desactivar la anterior; para ello se utilizan las instrucciones SET y RESET del autómata.

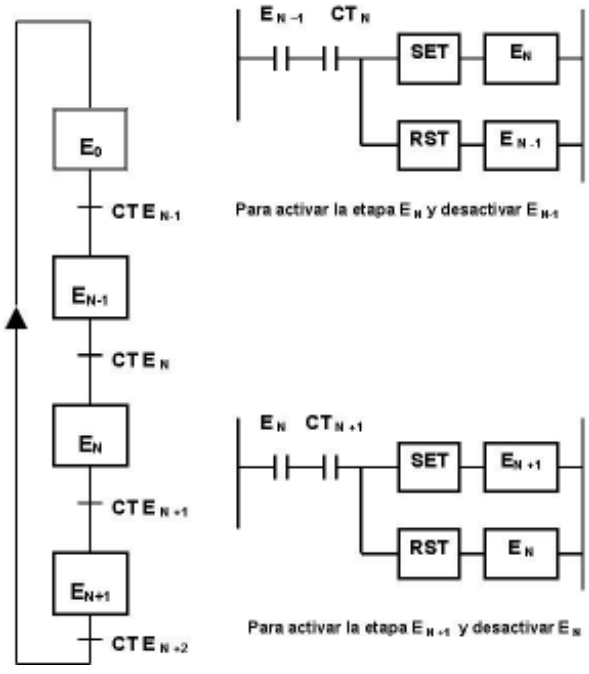
Las instrucciones SET y RESET se utilizan junto con las variables internas asociadas a cada una de las etapas del GRAFCET.



IN 1: Instrucción SET; IN 2: Instrucción RST

Cuando la entrada 1 del autómata se activa, la instrucción SET activará el relé interno R1 que permanecerá activado aunque se desactive la entrada 1; para desactivarlo será necesario emplear una instrucción RESET con otra entrada distinta; cuando se active la entrada 2, R1 se desactivará hasta que, de nuevo, se utilice la entrada 1 para activarlo.

En GRAFCET este tipo de instrucciones se utiliza de forma tal que es la condición de transición la que al cumplirse activa la etapa posterior y desactiva la etapa anterior.

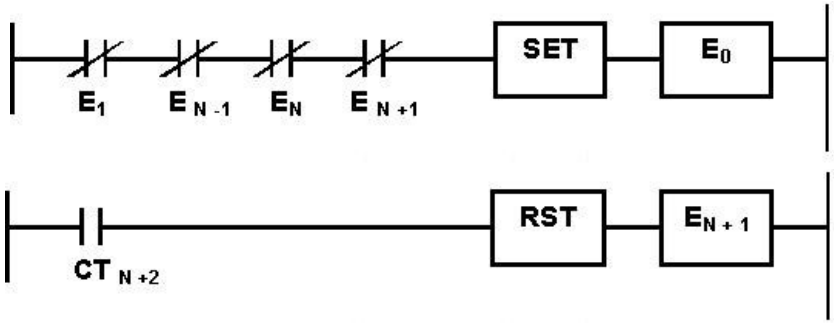


En la figura se representa un ciclo de GRAFCET, y las ecuaciones que permiten la activación y desactivación de la etapa E_n:

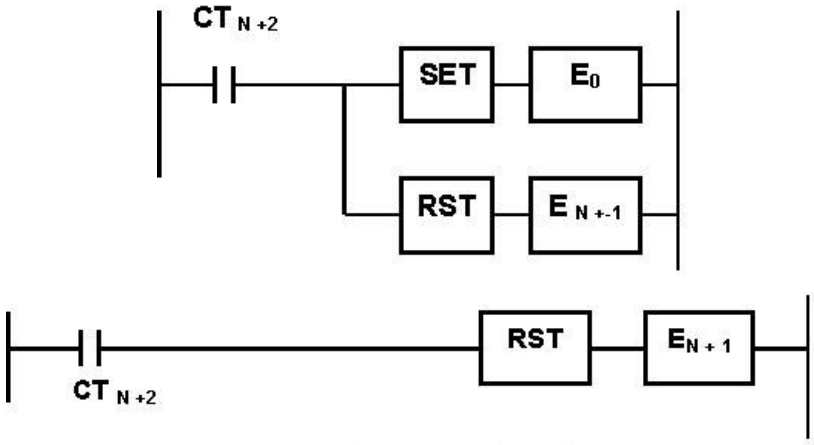
- para activarla se realiza la función AND entre la etapa anterior y la condición de transición asociada a E_n;
- para desactivarla se realiza la función AND entre la propia E_n y la condición de transición asociada a la etapa posterior

Para que comience a ejecutarse el GRAFCET es necesario activar la etapa E_0. Esto puede hacerse de varias formas:

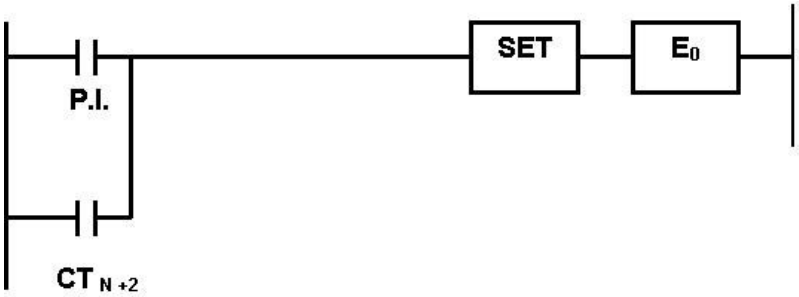
- **Primera forma.** En el ciclo actual del autómata, anulamos la ultima etapa activa; y, si todas las etapas están desactivadas, en el próximo ciclo se activa inicialmente E_0.



- **Segunda forma.** Con la última condición de transición activamos E_0 y desactivamos la última etapa activa.



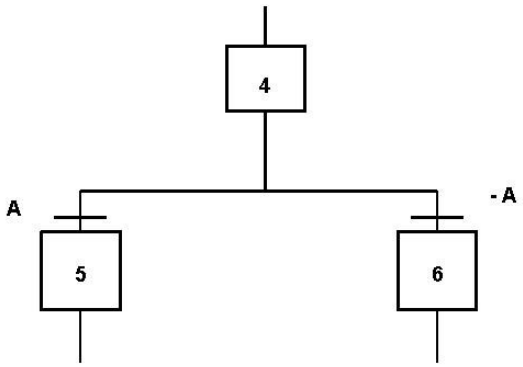
- **Tercera forma.** Muchos autómatas tienen una serie de variables internas específicas; la más común es el impulso inicial al pasar a modo RUN; PI. Este impulso inicial, en la mayoría de tos autómatas, permanece activo durante el primer ciclo.



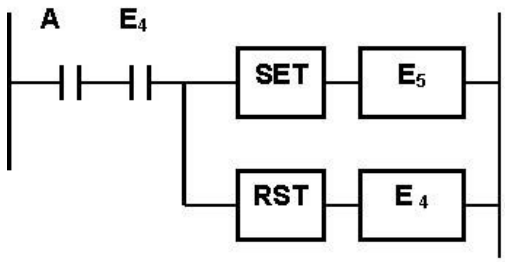
Cualquiera de las tres formas para activar la etapa E_0 es válida. El empleo de una de ellas en concreto es función del proceso que se pretende automatizar y del autómata que se utilice.

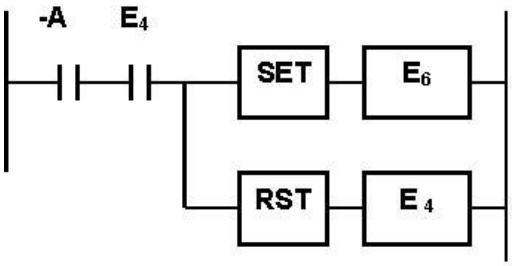
- **Elección condicional entre varias secuencias:**

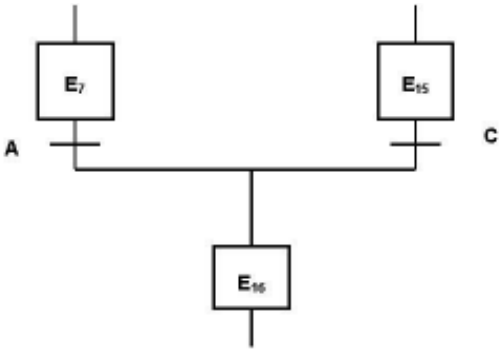
Suele ocurrir que, en un proceso, se llegue a un punto del ciclo en el que hay que efectuar una elección entre varias secuencias posibles, en función de las variables que intervienen en el proceso.

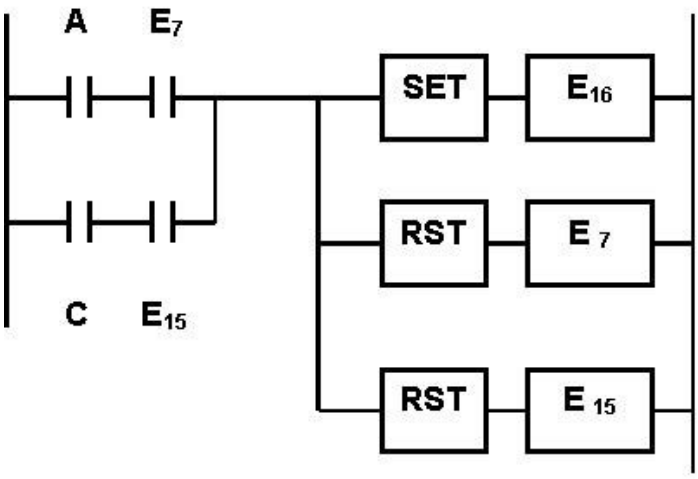


Las ecuaciones para el inicio de sentencia condicionales son:









Partiendo de la etapa E_4 activada, se puede realizar solamente una de las dos secuencias:

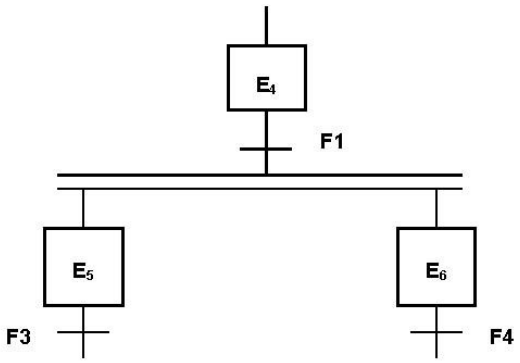
- Si se cumple la transición A se activará la etapa E_5.
- Si se cumple la transición A negada, se activará la etapa E_6.

La transición condicional implica que sólo una de las etapas posteriores se activará. Por tanto, la condición de transición asociada a la etapa 4 tiene que ser opuesta a la condición de transición asociada a la etapa E_5. La primera de las condiciones de transición que se cumpla, desactiva la etapa 3.

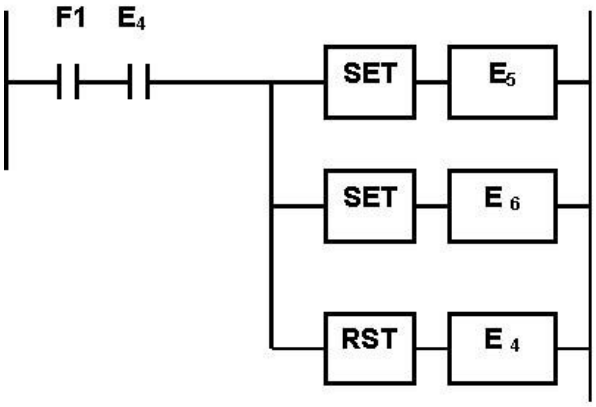
El final de dos secuencias condicionadas se produce cuando una de las dos condiciones de transición asociadas a la etapa E_{16} se cumple. Por ejemplo, si la etapa E_{16} está activa y se cumple la condición de transición A se activará la etapa E_{16} y se desactivará la E_7 o la E_{15}.

- **Secuencias simultáneas:**

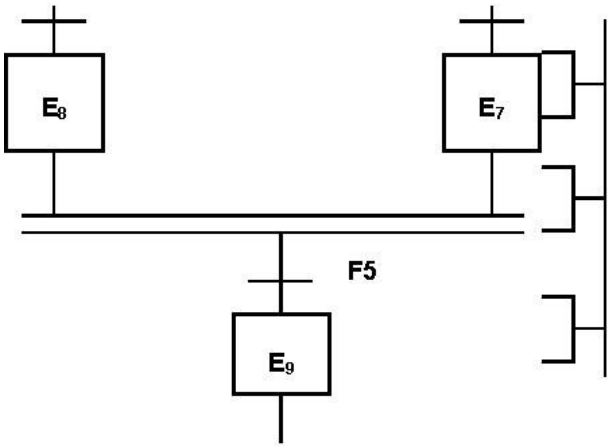
Puede darse el caso de que sea necesario el desarrollo de más de una secuencia a la vez, cuyas etapas no tengan ninguna interrelación. Para poder representar este funcionamiento simultáneo, se utiliza un par de trazos paralelos que indican el principio y el final de estas secuencias.



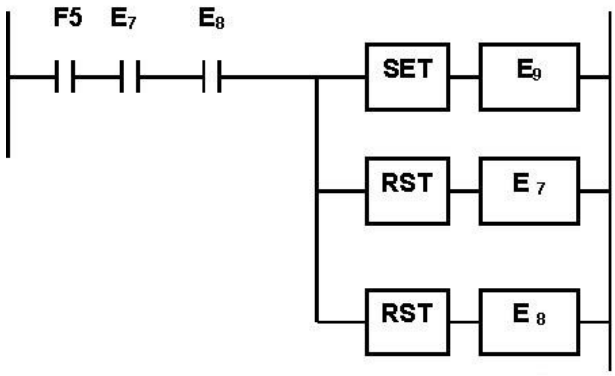
La activación de las secuencias simultáneas es:



El comienzo de las dos secuencias simultáneas se produce cuando se cumple la condición de transición F. El final se produce cuando las dos etapas E_7 y E_8 están activas, y se cumple la condición de transición F_5.

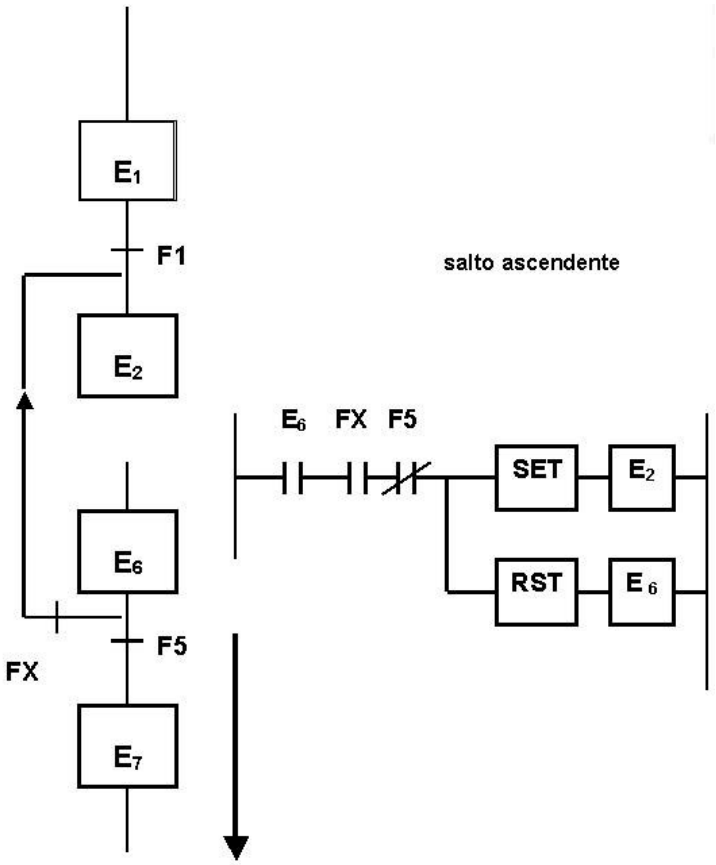


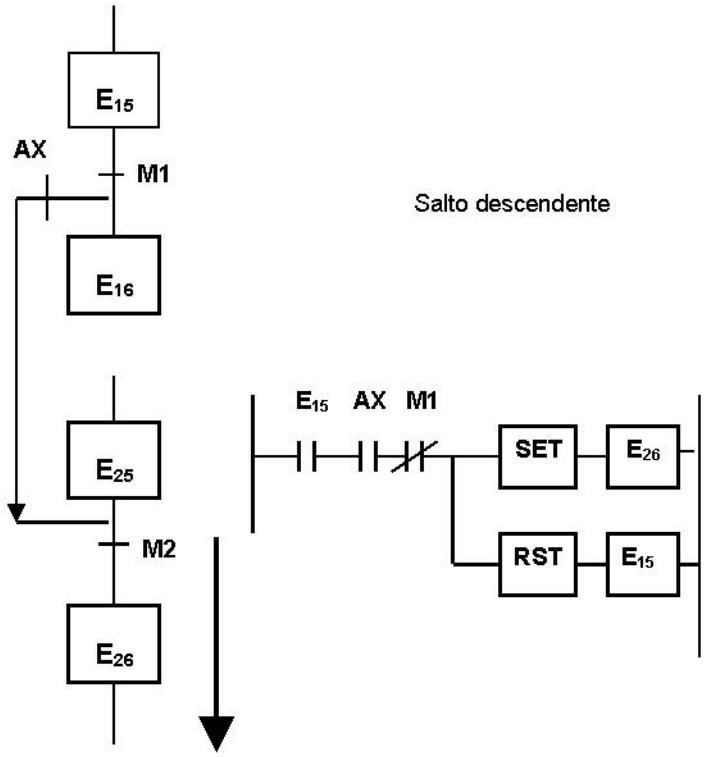
El final de las sentencias simultáneas:



- **Salto condicional a otra etapa:**

El salto condicional a otra etapa permite pasar de una etapa a otra sin activar las etapas intermedias. El salto condicional se puede hacer tanto en el sentido de evolución del GRAFCET como en el sentido inverso. El sentido del salto viene indicado por las flechas:


E1
F1
E2
E6
F5
FX
E7
salto ascendente
E6 FX F5
SET
E2
RST
E 6

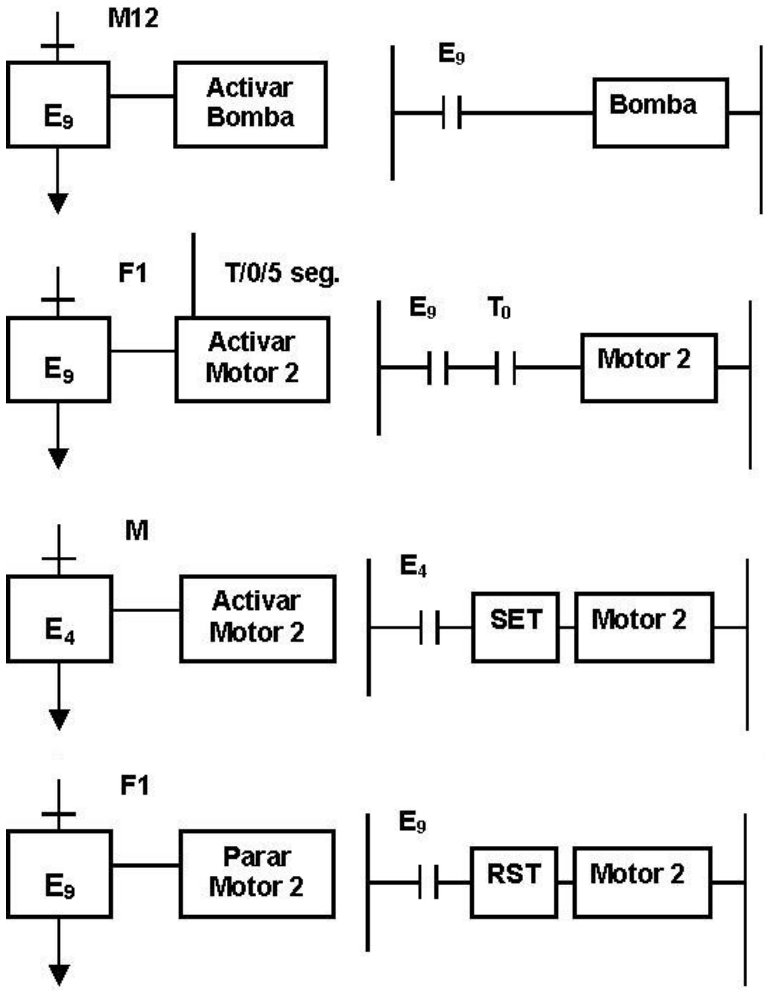

E15
AX
M1
E16
E25
M2
E26
Salto descendente
E15 AX M1
SET
E26
RST
E15

Acciones asociadas a las etapas

Una vez que la etapa está activa, las acciones asociadas pueden activarse con las correspondiente ecuaciones. Hay varias posibilidades:

- **Acciones activas.** Mientras está activa la etapa activa, la bomba está activa mientras E_9 está activa.
- **Acciones condicionadas por una variable.** El motor 2 se activará si E_9 está activa y han transcurrido 5 seg. desde que se activó el temporizador T_0 en la etapa E_0.
- **Acciones activadas en una etapa y desactivadas en otra posterior.** El motor 2 se activa al activarse E_4, y permanece activo hasta que se active la etapa E_9, que lo desactiva.
- **Etapas que no llevan asociada ninguna acción.**

Para que la acción se active es necesario que la condición y la etapa estén activas conjuntamente.



Por lo general, la etapa E_0 no lleva asociada ninguna acción; sólo se emplea para iniciar el ciclo una vez que ha sido activada.

Cuando se realizan dos secuencias simultáneas, es posible que el tiempo que cada una de estas secuencias tarda en realizarse sea distinto, en función del número de tareas asociadas a las etapas, en función de cuándo se activen las condiciones de transición, etc.

Para terminar dos secuencias simultáneas es necesario que las etapas últimas de cada una de ellas estén activas; una o las dos pueden ser etapas de espera para que la secuencia más rápida aguarde el final de la secuencia más lenta.

La conclusión para activar la etapa de espera es la terminación de todas las acciones asociadas a la etapa anterior a ella.

Programas del usuario

Primeramente, analizamos los datos que nos proporcionan las especificaciones iniciales del proceso: qué secuencias se han de realizar, en qué orden se activan, qué variables intervienen, cuáles son las condiciones de seguridad, etc.

Partiendo de estos datos, descomponemos el programa de usuario en tres apartados.

1. **Las condiciones iniciales.** En este apartado se incluyen todas aquellas acciones que el sistema de control ha de activar con prioridad en cualquier momento de la evolución del proceso y que no son secuenciales –como, por ejemplo, las condiciones de emergencia (alarmas de fallos, relés térmicos, etc.)–. Con las variables utilizadas en este apartado se implementan las ecuaciones necesarias para obtener, en caso de funcionamiento correcto, la señal que active la primera etapa del GRAFCET. En algunos casos, el apartado "condiciones iniciales" no existe o está formado por una sola variable, dependiendo del grado de seguridad que se quiera tener sobre el funcionamiento del proceso.

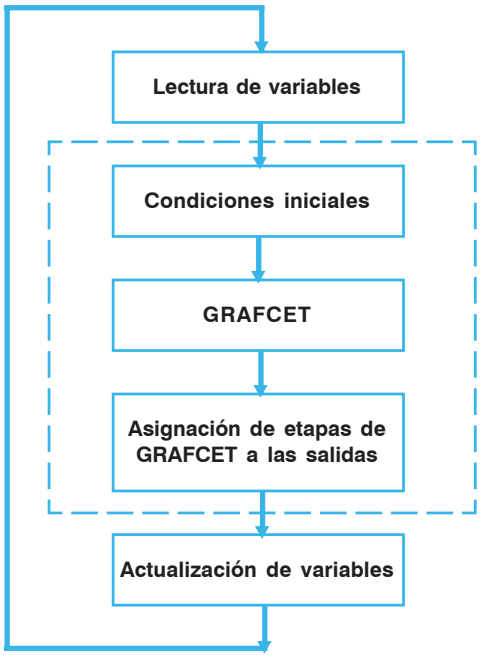
2. **GRAFCET.** En este apartado se representan las secuencias que ha de realizar el proceso y las ecuaciones necesarias para activar las etapas en función de las condiciones de transición. Las condiciones de transición están formadas por las señales proporcionadas por los sensores a las entradas del autómata y por las variables internas asociadas a otras etapas anteriores.

3. **La asignación de variables de etapa del GRAFCET a las salidas del PLC**. Las acciones que han de realizar las etapas del GRAFCET sobre el proceso a controlar se asignan a las salidas del autómata, de forma que pueda existir una conexión física entre el sistema de control y el proceso a controlar.

Comprobados y depurados los tres apartados, el paso siguiente es el de realizar la asignación de las variables que intervienen en el proceso a las entradas, salidas y variables internas del autómata concreto que se va a utilizar.

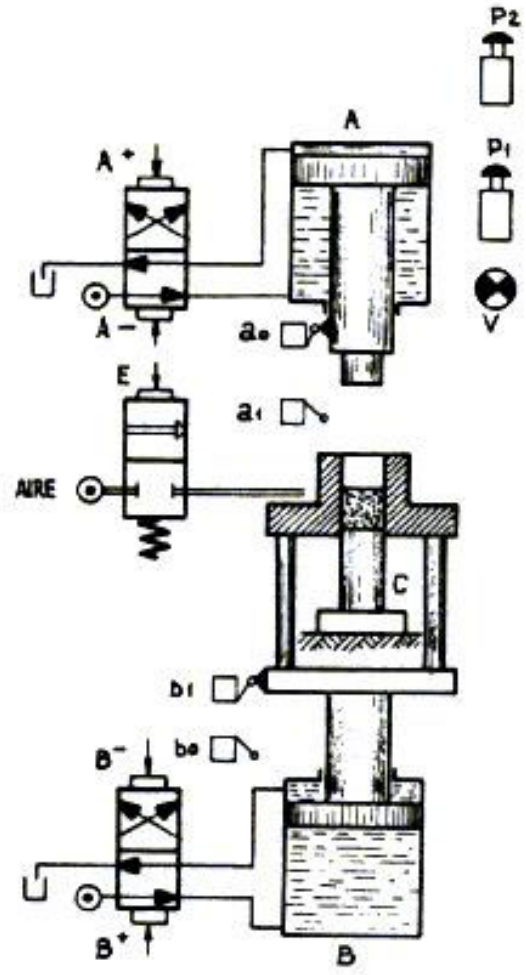
Empleando uno de los lenguajes de programación disponibles, esquema de contactos o lista de instrucciones, introducimos las ecuaciones que hemos obtenido en los tres apartados.

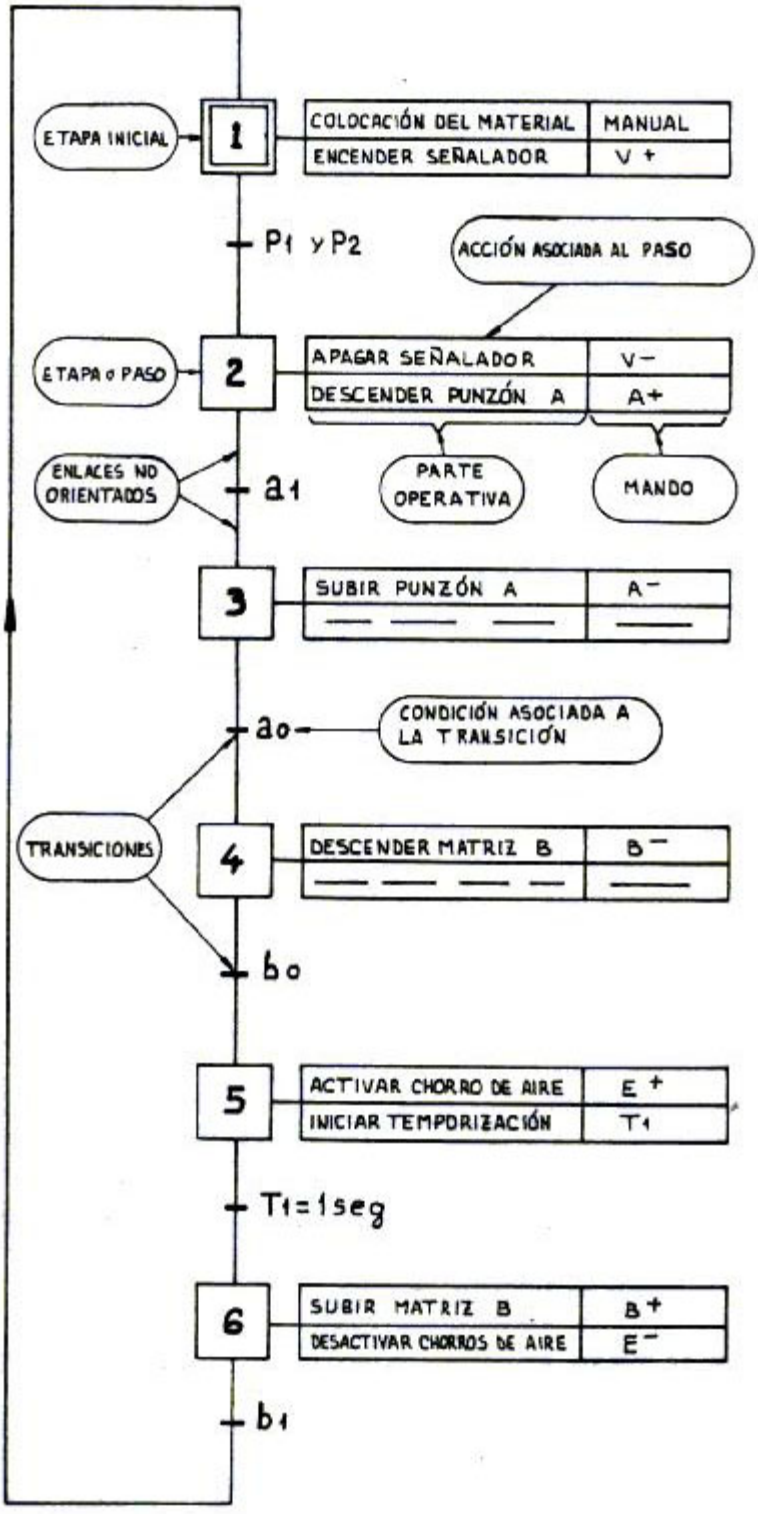
Cuando se transfiera el programa al autómata y se active el modo RUN, el ciclo que realiza constantemente es:



Consideremos algunos ejemplos:

Vamos a representar la secuencia de operación de una prensa destinada a la fabricación de piezas a partir de polvos metálicos.





- La colocación del material está asegurada manualmente por el operador. Un indicador luminoso V está encendido durante todo el curso de la colocación. Terminada aquélla, el operador autoriza la continuación de las operaciones, presionando dos pulsadores simultáneamente.

- Los movimientos del punzón superior y de la matriz son efectuados por cilindros hidráulicos de doble efecto. Las posiciones alta y baja del punzón y de la matriz son controladas con la ayuda de captadores de fin de carrera (respectivamente a0 y a1, b1 y b0) de naturaleza eléctrica.

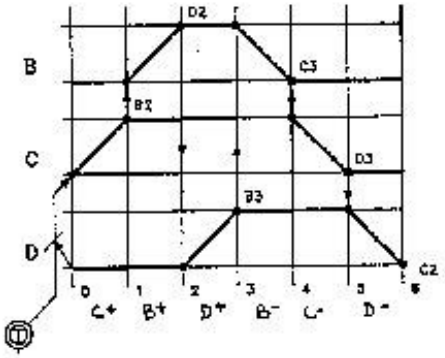
- La evacuación de la pieza es obtenida por medio de un chorro de aire, con una duración de un segundo. Este chorro de aire está comandado por una electroválvula E.

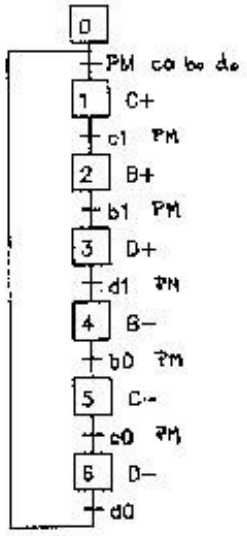
- **Ejercicio 1:**

Se desea realizar la siguiente secuencia: C+, B+, D+, B-, C-, D-, avanzando paso a paso cada vez que se oprime el pulsador PM.

Entradas:
PM = I 300
FC3 = (b0) I 100
FC4 = (b1) I 101
FC5 = (co) I 102
FC6 = (c1) I 200
FC7 = (do) I 201
FC8 = (d1) I 0

Salidas:
EVB = O 0
EVC = O 1
EVD = O 100





Solución:

Tipo de PLC	PRX 20
Tiempo de Scan	30000 μs
Base de tiempo TMR 00 a 15	0,1 seg.
Base de tiempo TMR 16 a 31	0,1 seg.

00		INI	Inicialización
01			
02	LD	H 0707	
03	INI	WG00	Gracet 00
04			
05	LD	H 0707	
06	INI	Z02	Entradas
07			
08	LD	H 0707	
09	INI	Z03	Salidas
10			
11			
12			
13		TRANS	Transiciones
14			
15	LD	I 0102	c0 = Vástago retraído cil. C
16	AND	I 0100	b0 = Vástago retraído cil. B
17	AND	I 0201	d0 = Vástago retraído cil. D
18	AND	I 0300	PM = Pulsador de marcha
19	STR	UO000	
20			

```
21   LD     I 0200      c1 = Vástago afuera cil. C
22   AND    I 0300      PM = Pulsador de marcha
23   STR    UO001
24
25   LD     I 0200      b1 = Vástago afuera cil. B
26   AND    I 0300      PM = Pulsador de marcha
27   STR    UO002
28
29   LD     I 0000      d1 = Vástago afuera cil. D
30   AND    I 0300      PM = Pulsador de marcha
31   STR    UO003
32
33   LD     I 0100      b0 = Vástago retraído cil. B
34   AND    I 0300      PM = Pulsador de marcha
35   STR    UO004
36
37   LD     I 0102      c0 = Vástago retraído cil. C
38   AND    I 0300      PM = Pulsador de marcha
39   STR    UO005
40
41
42
43          GRAFCET
44
45   IST    G0000
46   TR     U0000       c0 b0 d0 PM
47   STP    G0001       C+
48   TR     U0001       PM c1
49   STP    G0002       B+
50   TR     U0002       PM b1
51   STP    G0003       D+
52   TR     U0003       PM d1
53   STP    G0004       B-
54   TR     U0004       PM b0
55   STP    G0005       C-
56   TR     U0005       PM c0
57   STP    G0006       D-
58   TR     I 0000      d0
59   JP     G0000
60
61          SALIDAS
62
63   LD     G0001
64   LD     G0002
65   LD     G0003
66   LD     G0004
67   STR    O0000       C+
68
69   LD     G0003
70   LD     G0004
71   STR    O0000       B+
72
73   LD     G0003
74   LD     G0004
```

```
75    LD      G0005
76    STR     O0100          D+
77
78    END
```

- **Ejercicio 2:**

A partir del planteo del ejercicio 1, introducimos la variante de elección del tipo de ciclo mediante la llave 0-1-2, donde:

- 0 = avance paso a paso,
- 1 = ciclo simple y
- 2 = ciclo automático.

El pulsador de parada PP permite detener la secuencia, reanudándose ésta al oprimirse otra vez el pulsador de marcha PM.

Entradas adicionales:
LLO = (paso a paso) No conectada
LL1 = (ciclo simple) I 202
LL2 = (ciclo automático) I 700
PP = I 301

Solución:

Tipo de PLC	PRX 20
Tiempo de Scan	30000 μs
Base de tiempo TMR 00 a 15	0,1 seg.
Base de tiempo TMR 16 a 31	0,1 seg.

```
00          INI     Inicialización
01
02    LD    H 0707
03    INI   WG00    Gracet 00
04
05    LD    H 0707
06    INI   Z02     Entradas
07
08    LD    H 0707
09    INI   Z03     Salidas
10
11          TRANS   Transiciones
12
13    LD    I 0102  c0 = Vástago retraído cil. C
14    AND   I 0100  b0 = Vástago retraído cil. B
15    AND   I 0201  d0 = Vástago retraído cil. D
16    INT   I 0300  PM = Pulsador de marcha
17    OR    U1000
18    STR   UO000
19
20    LD    I 0200  c1 = Vástago afuera cil. C
21    INT   I 0300  PM = Pulsador de marcha
22    OR    U1000
23    STR   UO001
```

24			
25	LD	I 0200	b1 = Vástago afuera cil. B
26	AND	I 0300	PM = Pulsador de marcha
27	STR	UO002	
28			
29	LD	I 0101	d1 = Vástago afuera cil. D
30	INT	I 0300	PM = Pulsador de marcha
31	OR	U1000	
32	STR	UO003	
33			
34	LD	I 0000	b0 = Vástago retraído cil. B
35	INT	I 0300	PM = Pulsador de marcha
36	OR	U1000	
37	STR	UO004	
38			
39	LD	I 0002	c0 = Vástago retraído cil. C
40	INT	I 0300	PM = Pulsador de marcha
41	OR	U1000	
42	STR	UO005	
43			
44	LD	I 0300	PM = Pulsador de marcha
45	INT	I 0202	LL1 = ciclo simple
46	OR	I 0700	LL2 = automático
47	STR	U1000	
48			
49	LD	I 0300	
50	AND	I 0202	
51	AND	I 0700	
52	LD	I 0301	PP = Pulsador de parada
53	LD	I 0302	No = LL1
54	AND	I 0300	No = LL2
55	PUT	U1000	
56			
57			
58			
59			
60			
61			
62		GRAFCET	
63			
64	IST	G0000	
65	TR	U0000	c0 b0 d0 (PM + U1000)
66	STP	G0001	C +
67	TR	U0001	(PM + LL1 +LL2) c1
68	STP	G0002	B +
69	TR	U0002	(PM + LL1 +LL2) b1
70	STP	G0003	D +
71	TR	U0003	(PM + LL1 +LL2) d1
72	STP	G0004	B -
73	TR	U0004	(PM + LL1 +LL2) b0
74	STP	G0005	C -
75	TR	U0005	(PM + LL1 +LL2) c0
76	STP	G0006	D -

```
77   TR      I 0201          d0
78   JP      G0000
79
80
81
82           SALIDAS
83
84   LD      G0001
85   LD      G0002
86   LD      G0003
87   LD      G0004
88   STR     O0001           C +
89
90   LD      G0002
91   LD      G0003
92   STR     O0000           B +
93
94   LD      G0003
95   LD      G0004
96   LD      G0005
97   STR     O0100           D +
98
99   END
```