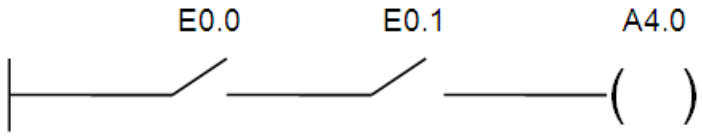
EJERCICIOS DE PROGRAMACIÓN RESUELTOS CON STEP 7

EJERCICIO 1. Contactos en serie (Instrucción U)

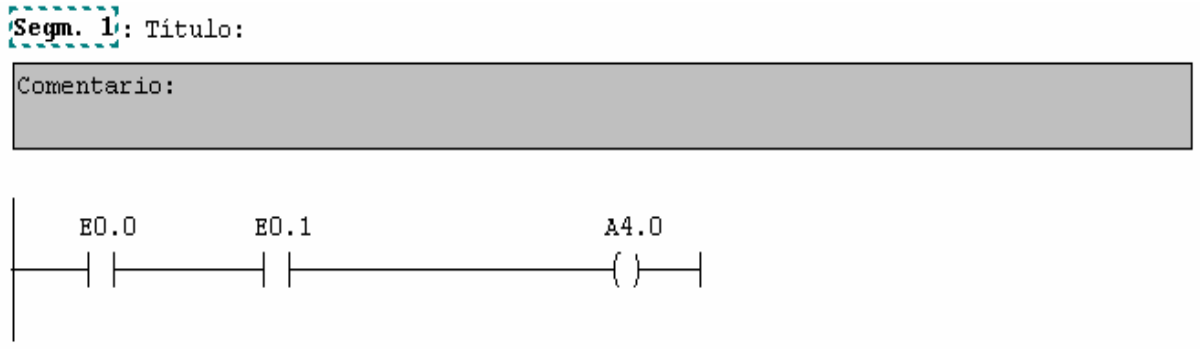
Automatizar el siguiente circuito en los tres lenguajes AWL, KOP y FUP.



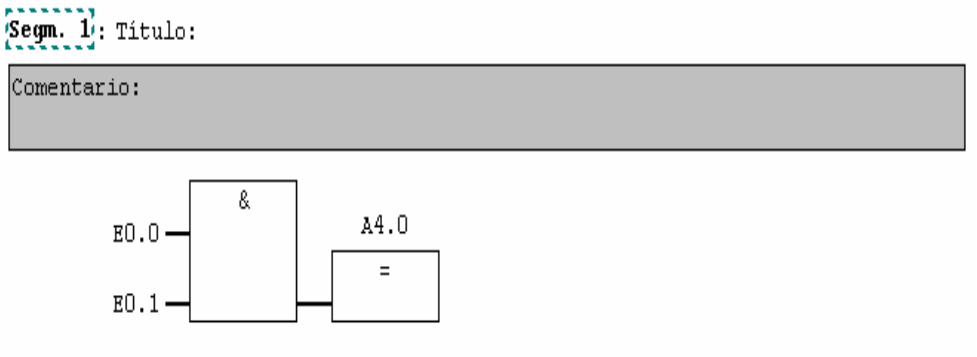
Solución en AWL

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

Solución en KOP

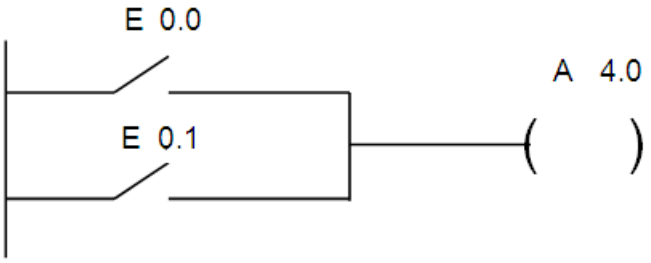


Solución en FUP



EJERCICIO 2. Contactos en paralelo (Instrucción O)

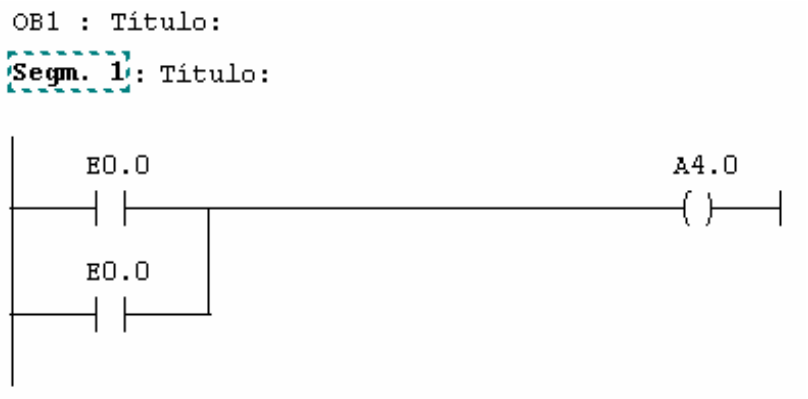
Automatizar el siguiente circuito en los tres lenguajes AWL, KOP y FUP.



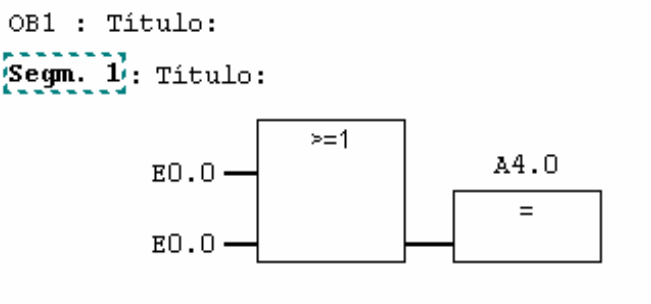
Solución en AWL

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

Solución en KOP

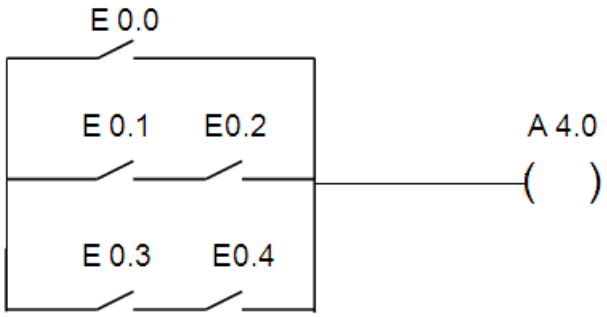


Solución en FUP



EJERCICIO 3. Contactos en Serie-Paralelo (instrucciones con paréntesis)

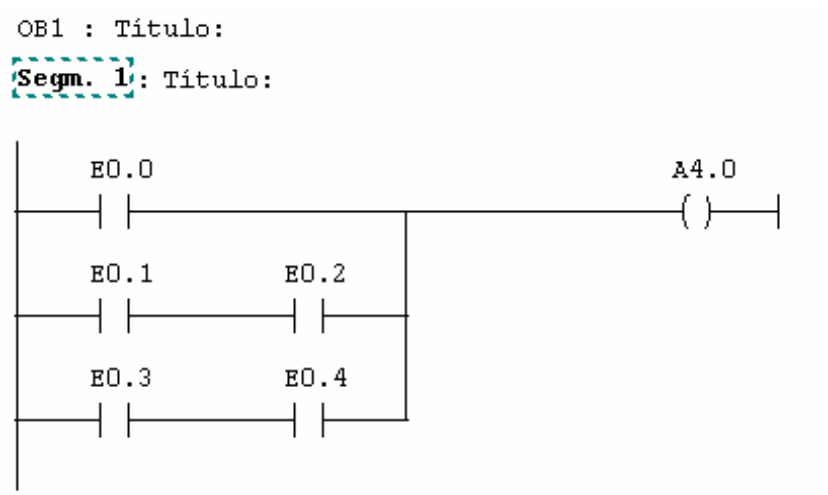
Automatizar el siguiente circuito en los tres lenguajes AWL, KOP y FUP.



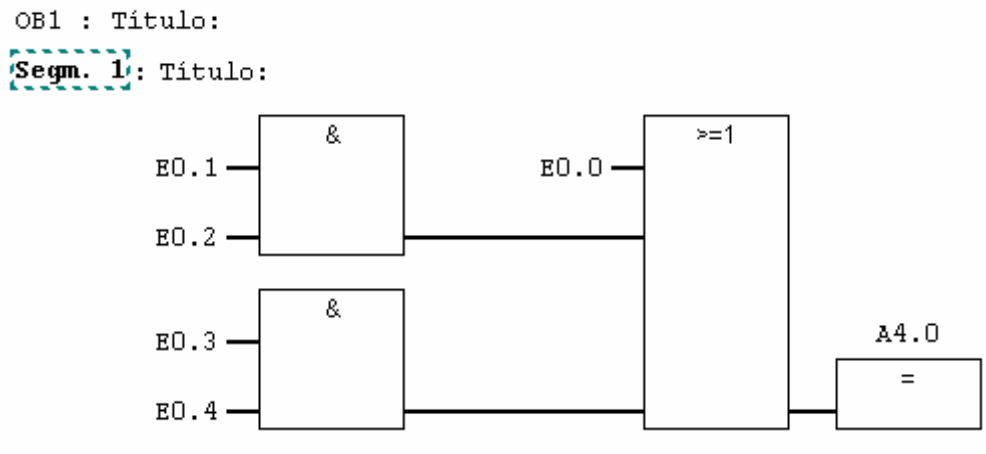
Solución en AWL

```
U   E   0.0
O(
U   E   0.1
U   E   0.2
)
O
U   E   0.3
U   E   0.4
=   A   4.0
```

Solución en KOP

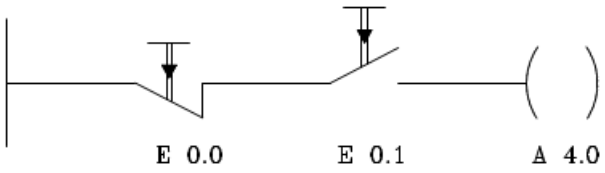


Solución en FUP



EJERCICIO 4. Contactos Negados (instrucciones ON y UN).

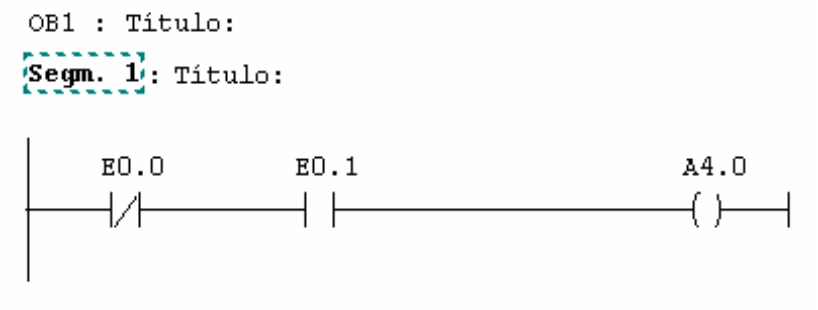
Automatizar el siguiente circuito en los tres lenguajes AWL, KOP y FUP. Se desea que se active la salida cuando accionemos los dos pulsadores. En un contacto queremos que dé señal cuando se cierre físicamente el contacto. En el otro caso queremos que dé señal cuando se abra físicamente el contacto.



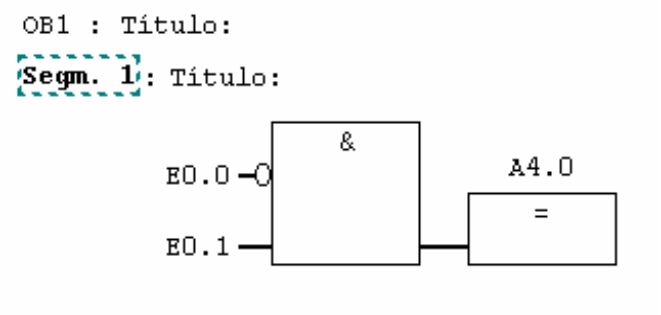
Solución en AWL

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

Solución en KOP



Solución en FUP



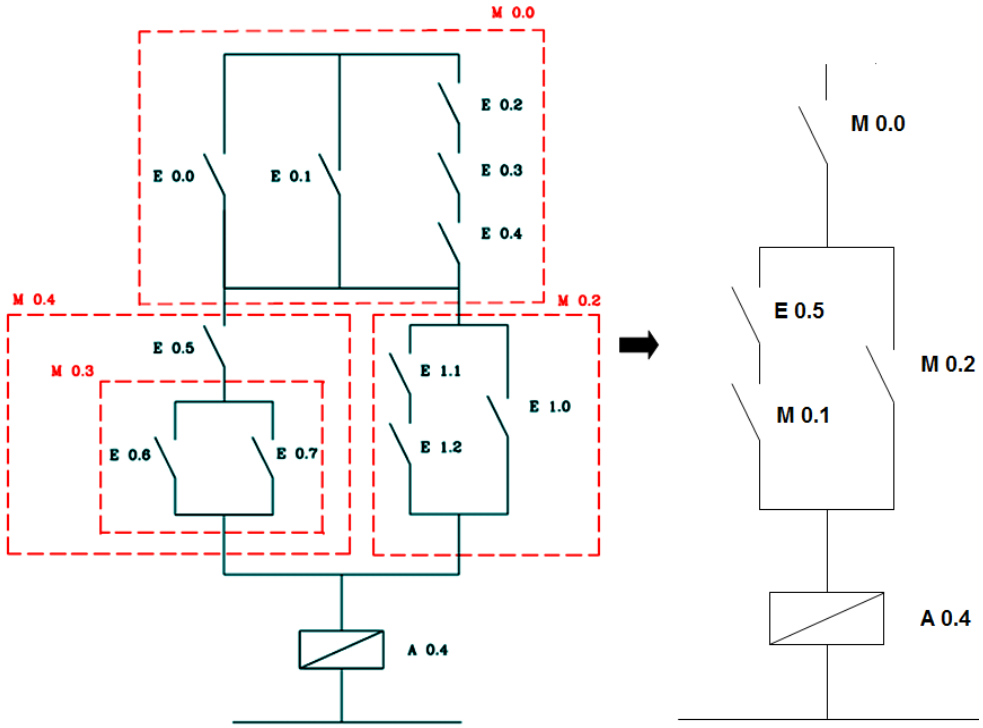
EJERCICIO 5. Programación con Marcas.

Las marcas son bits internos de la CPU. Disponemos de una cantidad limitada de marcas dependiendo de la CPU con la que estemos trabajando. Los bits de marca podemos activarlos y desactivarlos como si fueran salidas, y podremos consultarlos en cualquier punto del programa.
Las marcas se pueden direccionar como bit (M), byte (MB), palabra (MW) o doble-palabra (MD). Por ejemplo:

Bit de marcas:	M 0.0, M 10.7, M 4.5, etc.
Byte de marcas:	MB20, MB100, MB50, etc.
Palabra de marcas:	MW100, MW200, etc.
Doble-Palabra de marcas:	MD100, MD200, etc.

Veamos un ejercicio de aplicación

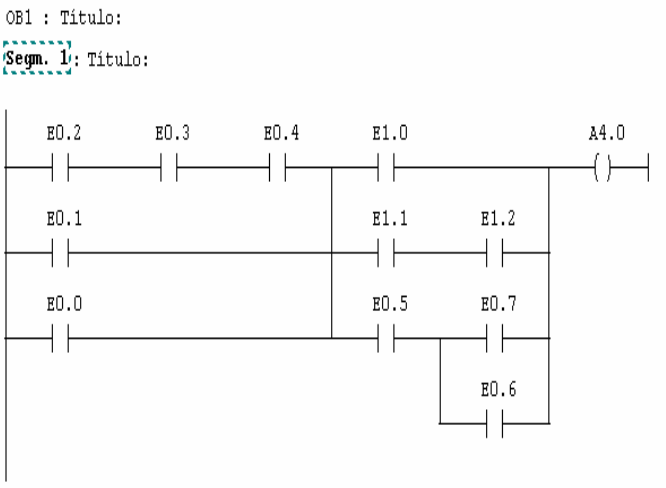
Para resolver el circuito eléctrico de la figura, podemos simplificar el circuito utilizando marcas, quedando por programar un circuito tan sencillo como vemos en la figura.



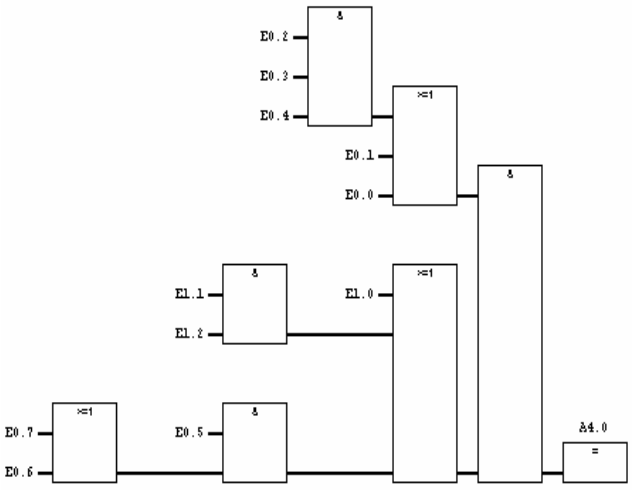
Solución en AWL

```
U     E    0.0
O     E    0.1
O(
U     E    0.2
U     E    0.3
U     E    0.4
)
=     M    0.0
U     E    0.6
O     E    0.7
=     M    0.1
U     E    1.1
U     E    1.2
O     E    1.0
=     M    0.2
U     E    0.5
U     M    0.1
=     M    0.3
U     M    0.0
U(
U     M    0.3
O     M    0.2
)
=     A    4.0
```

Solución en KOP



Solución en FUP



EJERCICIO 6. Instrucciones SET y RESET

Las instrucciones SET y RESET son instrucciones de memoria. Si programamos un SET de una salida o de una marca con unas condiciones, se activará cuando se cumplan dichas condiciones. Aunque las condiciones dejen de cumplirse, no se desactivará hasta que se haga un RESET de la salida o marca.

Estas instrucciones tienen prioridad. Dependen del orden en que las programemos. Siempre va a tener prioridad la última que programemos.

Veamos por qué ocurre esto.

Existen dos registros internos que se llaman PAE (imagen de proceso de entradas) y PAA (imagen de proceso de salidas).

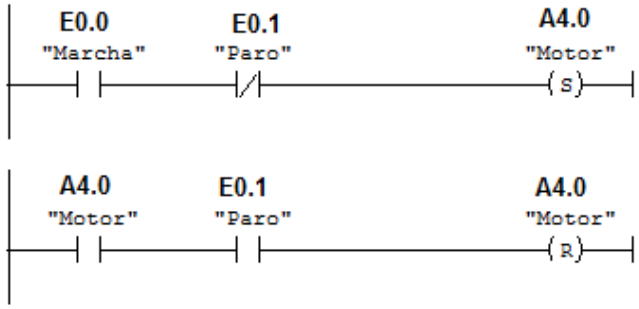
Antes de ejecutarse el OB1, se hace una copia de las entradas reales en la PAE. Durante la ejecución del OB1, el PLC no accede a la periferia real para hacer sus consultas, lo que hace en realidad es acceder a este registro interno. Este registro se refresca cada vez que comienza un nuevo ciclo de scan.

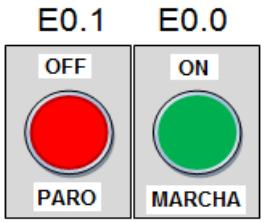
Según se van ejecutando las instrucciones, el PLC no accede a las salidas reales para activarlas o desactivarlas. Accede al registro interno PAA y pone “0” o “1”.

Sólo cuando termina cada ciclo de scan accede realmente a las salidas. Entonces lo que hace es copiar lo que hay en la PAA en las salidas reales. En nuestro caso, si hacemos un SET y un RESET dentro del mismo ciclo de scan, al final de cada ciclo hará efecto lo último que hayamos programado.

Veamos un ejercicio de aplicación

El circuito de la figura corresponde a un enclavamiento eléctrico. Esto hace las funciones de dos pulsadores, uno de marcha y otro de paro. Es la forma más cómoda de programar dos pulsadores.



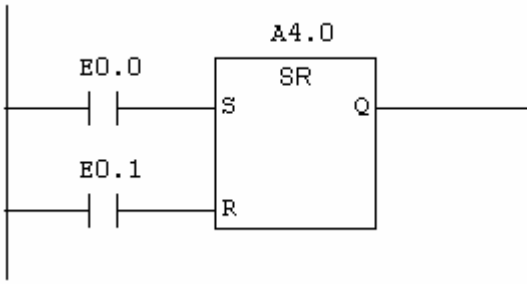


Esto lo podríamos programar tal y como aparece en la figura, pero si lo hacemos mediante instrucciones SET y RESET quedaría de la siguiente manera:

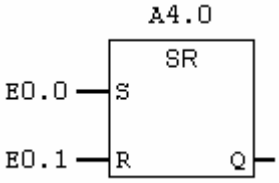
Solución en AWL

```
U   E   0.0
S   A   4.0
U   E   0.1
R   A   4.0
```

Solución en KOP



Solución en FUP

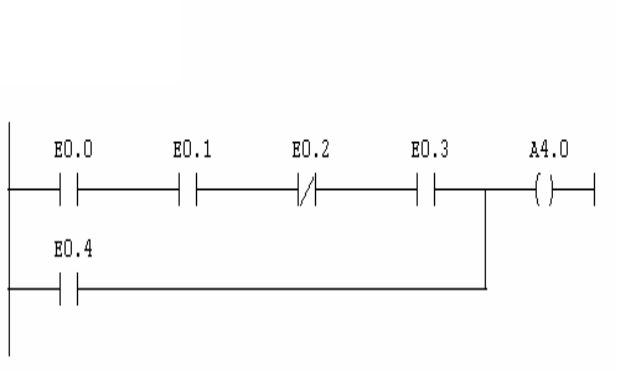


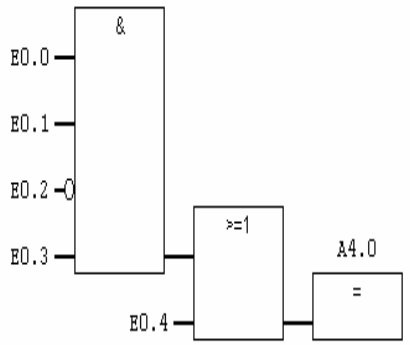
EJERCICIO 7. Observación de variables.

El administrador Step 7 nos permite conocer en todo momento el valor que toma una variable durante la ejecución del programa e incluso forzar su estado. Este es particularmente útil para hacer un seguimiento y para depuración del programa que ejecuta el PLC.

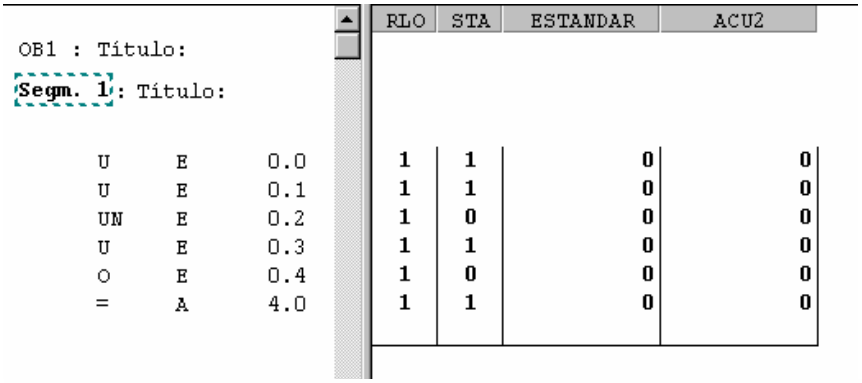
Supongamos a modo de ejemplo el siguiente programa, al que deseamos hacer un seguimiento de su ejecución en el PLC desde el propio Administrador Step 7. Como es lógico, será imprescindible estar conectado al PLC mediante el interface MPI-PC.

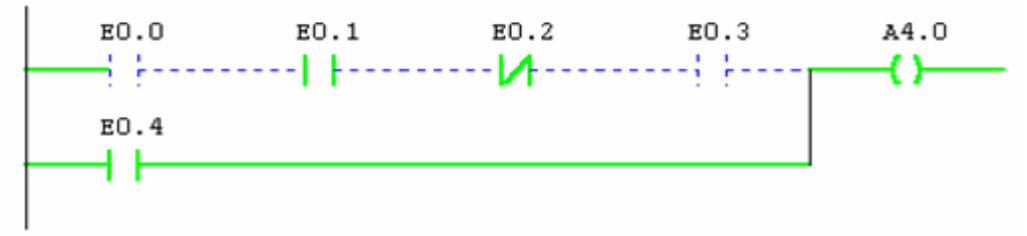
```
U   E   0.0
U   E   0.1
UN  E   0.2
U   E   0.3
O   E   0.4
=   A   4.0
```

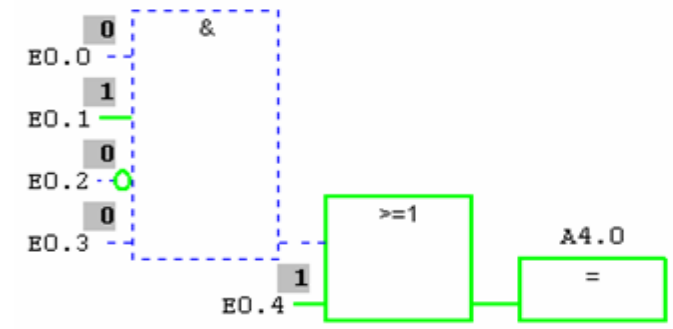




Desde el menú Test > Observar, o bien pulsando sobre el icono que representa unas gafas, podremos conocer en tiempo real el estado de ejecución de nuestro programa. En las imágenes siguientes vemos el resultado según estemos operando en AWL, KOP o FUP. Observe como al accionar alguno de los contactos, varían tanto su valor como su representación gráfica.



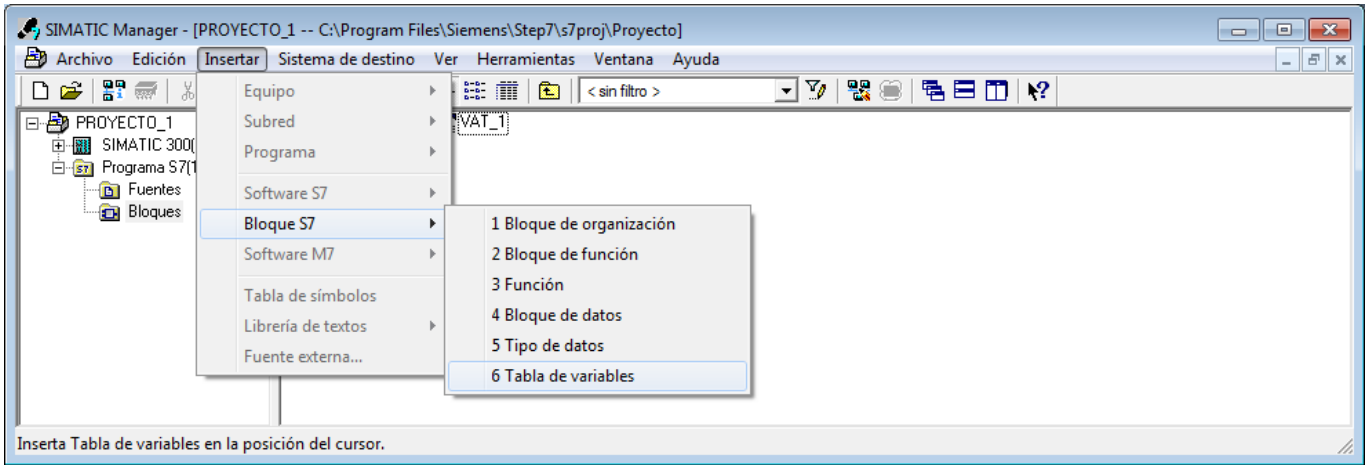




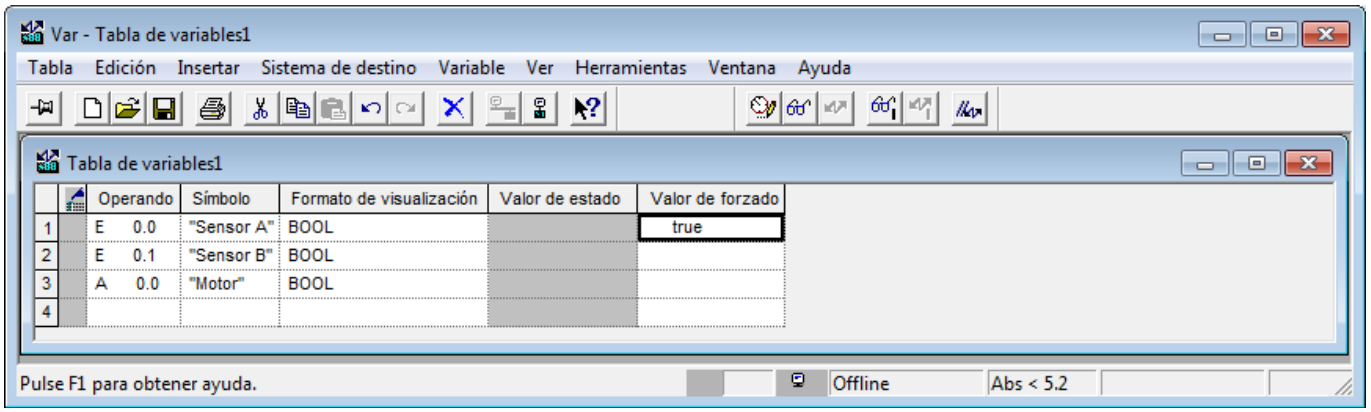
EJERCICIO 8. Tablas de variables (Observación y Forzado de variables)

Además de lo visto en el ejercicio anterior, podemos abrir y configurar una tabla en la que podemos observar las variables que deseemos.

Estas tablas son bloques más dentro del proyecto, y podemos insertarlas desde el administrador abriendo el menú "Insertar" > Bloque S7 > "Tabla de variables". Observe como ahora dentro de nuestro proyecto y junto al bloque de organización OB1 nos aparecerá nuestra tabla con el nombre VAT_1.



Podemos acceder a la tabla, bien abriéndola directamente en la carpeta de Programa S7, o bien a través del menú "Sistema Destino" > "Observar/Forzar variables".



Una vez abierta la tabla deberemos escribir las variables que deseamos observar y/o forzar en la columna "Operando", e indicar el formato de visualización. El estado de las variables se nos mostrará en la columna "Valor de estado". Si deseamos forzar el valor de una variable utilizaremos la columna "Valor de forzado". En la figura hemos forzado E0.0 a "1".

Para poder observar y forzar las variables tenemos que ejecutar nuestro programa en el PLC y conectarnos con el ONLINE mediante el botón "gafas" de la barra de herramientas.

El botón de las que representa unas gafas con una rayita al lado sólo nos permitirá una visualización instantánea de las variables. Si se producen cambios en las variables no los veremos reflejados en la pantalla.

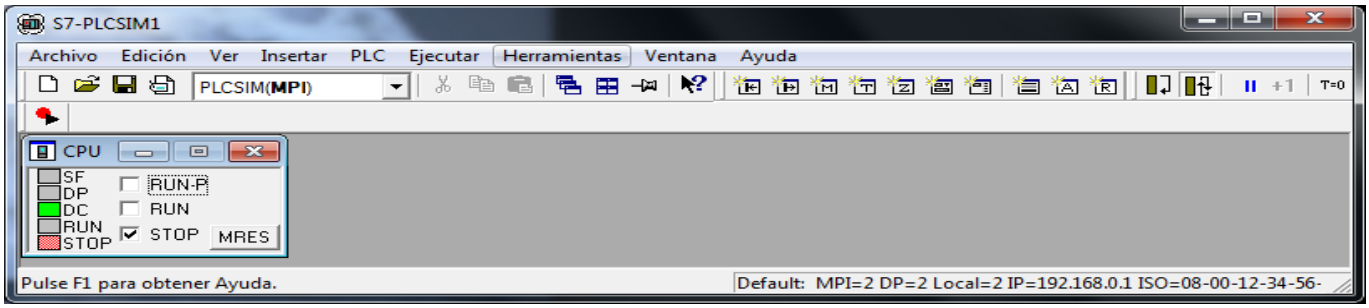
Los botones que representan unos "rayos" son para forzar variables. Podemos hacer un solo forzado o hacer un forzado continuo.

EJERCICIO 9. Simulación con PLCSIM

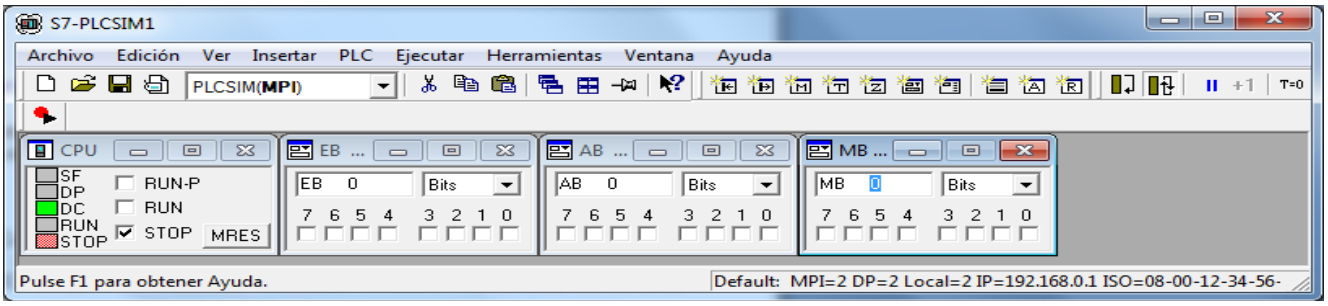
PLCSIM es un simulador incluido dentro del software de SIMATIC STEP 7. Con este simulador podremos comprobar nuestro programa en el PC sin necesidad de tener conectado el autómata.

Los pasos a seguir son:

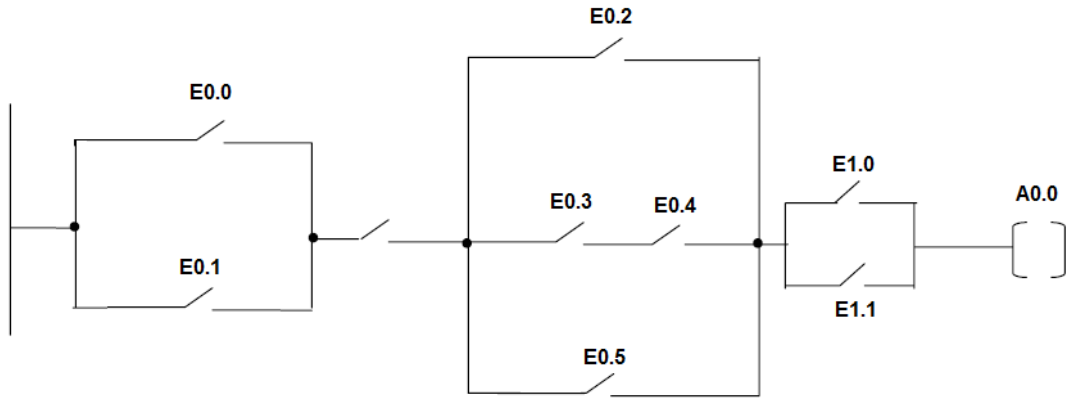
1. Configure el hardware del proyecto. Compile y Guarde la configuración.
2. Escriba el programa del PLC en OB1. Compile y Guarde el programa.
3. Abra PLCSIM desde INICIO>SIMATIC>STEP7>S7-PLCSIM –Simular módulos. Abriremos una ventana tal como muestra la figura.



4. Proceda ahora a la carga de todo el proyecto (hardware+programas) desde la ventana del administrador tal y como lo haría para el PLC.
5. Desde el menú “Insertar” podrá abrir elementos de visualización tales como entradas, salidas, marcas, temporizadores, etc. En nuestro caso hemos insertado un elemento de entradas (EB0), uno de salidas (AB0) y otro de marcas (MB0).

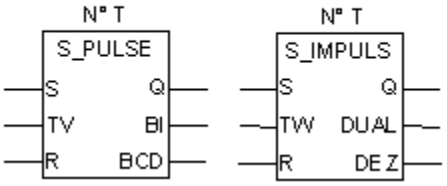
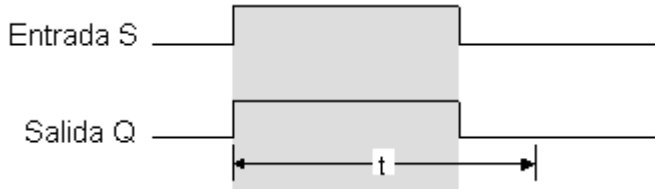
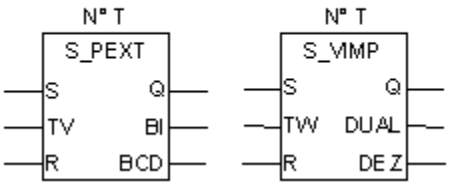
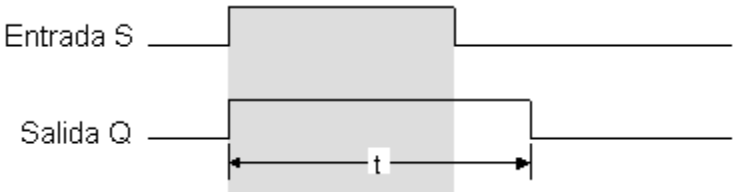
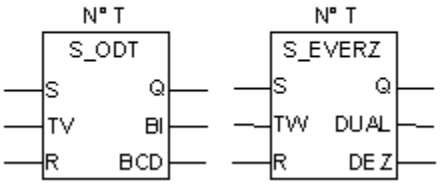
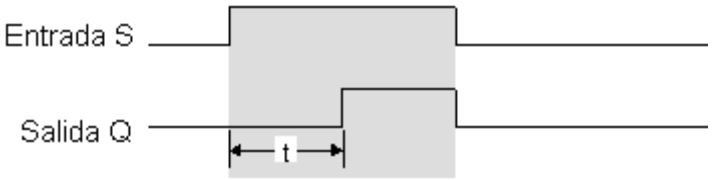
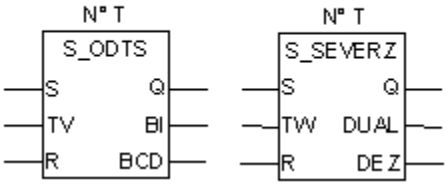
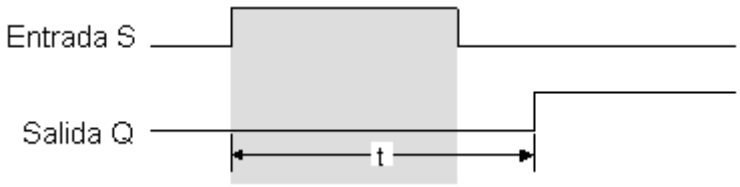
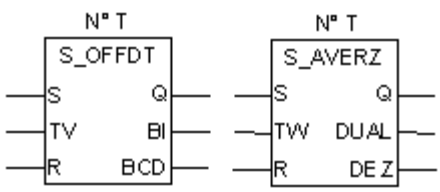
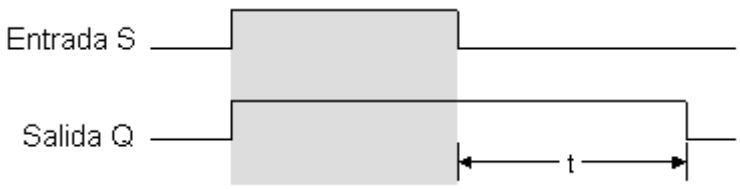


6. Tan sólo nos queda poner en modo “RUN” el simulador y comprobar el funcionamiento del programa. Para activar las entradas del PLC tendrá que hacer clic sobre las pestañas de la ventana EB0. Los bits de salidas y las marcas deberán activarse de acuerdo con el programa cargado.
7. Como ejercicio práctico le proponemos que resuelva con marcas y simule la siguiente función:



TEMPORIZADORES

El administrador Step 7 nos ofrece cinco tipos de temporizadores. Elegir el temporizador apropiado dependerá de las necesidades del programador. Indicamos aquí un breve resumen y descripción de estos:

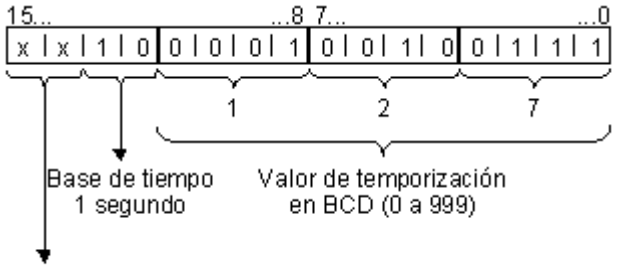
SÍMBOLO	Descripción
S_IMPULS Temporizador de impulso **Inglés** **Aléman** 	El tiempo máximo que la señal de salida permanece a 1 corresponde al valor de temporización t programado. La señal de salida permanece a 1 durante un tiempo inferior si la señal de entrada (S) cambia a 0. 
S_VIMP - Temporizador de impulso prolongado **Inglés** **Aléman** 	La señal de salida permanece a 1 durante el tiempo programado, independientemente del tiempo en que la señal de entrada esté a 1. 
S_EVERZ - Temporizador de retardo a la conexión **Inglés** **Aléman** 	La señal de salida es 1 solamente si ha finalizado el tiempo programado y la señal de entrada sigue siendo 1. 
S_SEVERZ - Temporizador de retardo a la conexión con memoria **Inglés** **Aléman** 	La señal de salida cambia de 0 a 1 solamente si ha finalizado el tiempo programado, independientemente del tiempo en que la señal de salida esté a 1. 
S_AVERZ - Temporizador de retardo a la desconexión **Inglés** **Aléman** 	La señal de salida es 1 cuando la señal de entrada es 1 o cuando el temporizador está en marcha. El temporizador arranca cuando la señal de entrada cambia de 1 a 0. 

Área de memoria

Los temporizadores tienen un área reservada en la memoria de la CPU. Esta área de memoria reserva una palabra de 16 bits para cada operando de temporizador. La programación con KOP asiste 256 temporizadores.

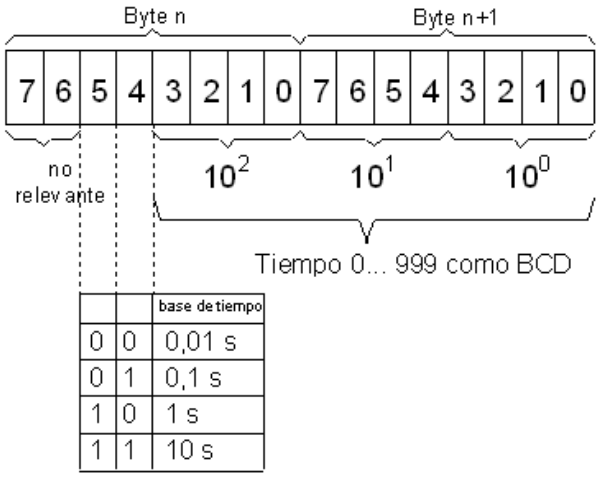
Cuando se dispara un temporizador, el contenido de la palabra de temporización 1 se utiliza como valor de temporización. Los bits 0 a 11 de la palabra de temporización almacenan el valor de temporización en formato decimal codificado en binario (formato BCD: cada grupo de cuatro bits contiene el código binario de un valor decimal). Los bits 12 a 13 almacenan la base de tiempo en código binario.

La figura muestra el contenido de la palabra de temporización cargado con el valor 127 y una base de tiempo de 1 segundo.



Irrelevante: Estos bits se ignoran al arrancar el temporizador

Tipo de datos	Longitud (bits)	Formato	Ejemplos del formato	
			mín.	máx.
S5TIME	16	h min s ó min s ms	S5T#0ms y S5T#10ms	S5T#2h4



		base de tiempo
0	0	0,01 s
0	1	0,1 s
1	0	1 s
1	1	10 s

Valor de temporización. Tipo de datos S5TIME

Los bits 0 a 9 de la palabra de temporización contienen el valor de temporización en código binario. Este valor indica un número de unidades. La actualización decrementa el valor de temporización en una unidad y en el intervalo indicado por la base de tiempo hasta alcanzar el valor 0. El valor de temporización se puede cargar en los formatos binario, hexadecimal o decimal codificado en binario (BCD). El área de temporización va de 0 a 9 990 segundos. Para cargar un valor de temporización redefinido, se observarán las siguientes reglas sintácticas.

El valor de temporización se puede cargar en cualquiera de los siguientes formatos:

- **w#16#wxyz** siendo: w= la base de tiempo (es decir, intervalo de tiempo o resolución). xyz = el valor de temporización en formato BCD

- **S5T#aH_bM_cS_dMS** siendo: H (horas), M (minutos), S (segundos), MS (milisegundos); a, b, c, d los define el usuario

La base de tiempo se selecciona automáticamente y el valor de temporización se redondea al próximo número inferior con esa base de tiempo.

El valor de temporización máximo que puede introducirse es de 9 900 segundos ó 2H_46M_30S.

Ejemplos:

S5TIME#4S --> 4 segundos
s5t#2h_15m --> 2 horas y 15 minutos
S5T#1H_12M_18S --> 1 hora 12 minutos y 18 segundos

Base de tiempo

Los bits 12 y 13 de la palabra de temporización contienen la base de tiempo en código binario. La base de tiempo define el intervalo en que se decrementa en una unidad el valor de temporización. La base de tiempo más pequeña es 10 ms, la más grande 10 s.

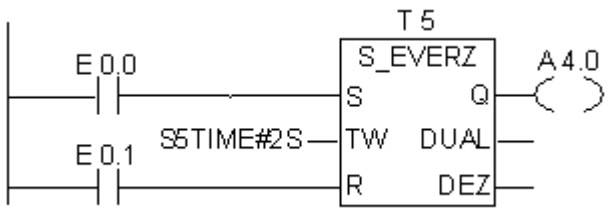
Base de tiempo	**Base de tiempo en código binario**
10 ms	00
100 ms	01
1 s	10
10 s	11

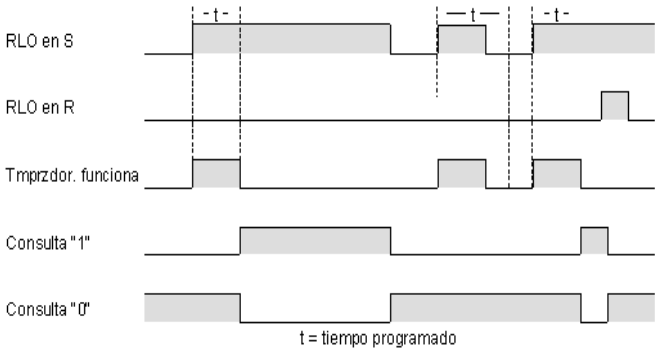
Los valores no deben exceder 2H_46M_30S. Los valores con un margen o una resolución demasiado grandes (p. ej. 2H_10MS) se redondean de tal forma que correspondan a la tabla para el margen y la resolución.

El formato general para el tipo de datos S5TIME tiene los siguientes valores límite para el margen y la resolución:

Resolución	**Márgen**
10MS a 9S_990MS	10MS a 9S_990MS
100MS a 1M_39S_900MS	00MS a 1M_39S_900MS
1S a 16M_39S	1S a 16M_39S
10S a 2H_46M_30S	10S a 2H_46M_30S

Ejemplo con temporizador de retardo a la conexión (S_EVERZ):





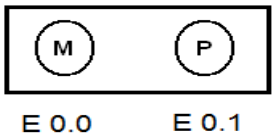
Si el estado de señal de la entrada E 0.0 cambia de "0" a "1" (flanco creciente), se activa el temporizador T5. Si transcurre el tiempo de dos segundos y el estado de señal en la entrada E 0.0 sigue siendo "1", la salida A 4.0 será "1". Si el estado de señal de la entrada E 0.0 cambia de "1" a "0", el temporizador se para y la salida A 4.0 será "0". (Si el estado de señal de la entrada E 0.1 cambia de "0" a "1", el temporizador se pone a 0, tanto si estaba funcionando como si no).

El valor de temporización actual queda depositado en la salida DUAL y DEZ. El valor de temporización en la salida DUAL está en código binario, el valor en la salida DEZ está en formato decimal codificado en binario. El valor de temporización actual equivale al valor inicial de TW menos el valor de temporización que ha transcurrido desde el arranque del temporizador.

Veamos ahora un ejemplo práctico del uso de temporizadores.

EJERCICIO 10. CONTROL DE UN SEMÁFORO

Tenemos un semáforo con las tres luces verde, ámbar y rojo. Disponemos también de dos pulsadores de mando: un pulsador de marcha y un pulsador de paro. El ciclo comienza tras pulsar el pulsador de marcha, siguiendo la siguiente secuencia indicada. El ciclo es repetitivo hasta que se pulse el pulsador de paro. En ese momento se apaga todo. Siempre que le dé al pulsador de marcha quiero que empiece por el verde.



1º/ Verde durante 5 seg.

2º/ Verde + Amarillo durante 2 seg.

3º/ Rojo durante 6 seg

Solución en AWL

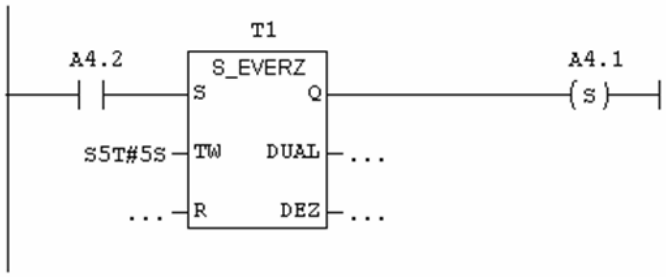
```
U   E   0.0   //Al activar el pulsador de marcha
S   A   4.2   //Encender el verde
U   A   4.2   //Si se ha encendido el verde
L   S5T#5S    //Cuenta 5 segundos
SE  T   1     //Con el temporizador 1
U   T   1     //Y cuando acabes de contar
S   A   4.1   //Enciende el amarillo
U   A   4.1   //Si se ha encendido el amarillo
L   S5T#2S    //Cuenta 2 segundos
SE  T   2     //Con el temporizador 2
U   T   2     //Y cuando acabes de contar
S   A   4.0   //Enciende el rojo
R   A   4.1   //Apaga el amarillo
R   A   4.2   //Y apaga el verde
U   A   4.0   //Si se ha encendido el rojo
L   S5T#6S    //Cuenta 6 segundos
SE  T   3     //Con el temporizador 3
U   T   3     //Cuando acabes de contar
S   A   4.2   //Enciende el verde
R   A   4.0   //Y apaga el rojo
U   E   0.1   //Si se activa el pulsador de paro
R   A   4.0   //Apaga el rojo
R   A   4.1   //Apaga el amarillo
R   A   4.2   //Apaga el verde
```

Solución en KOP

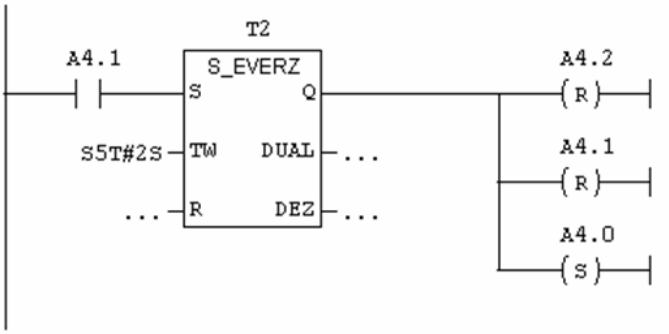
OB1 : SEMAFORO

Segm. 1 : ENCENDER EL VERDE

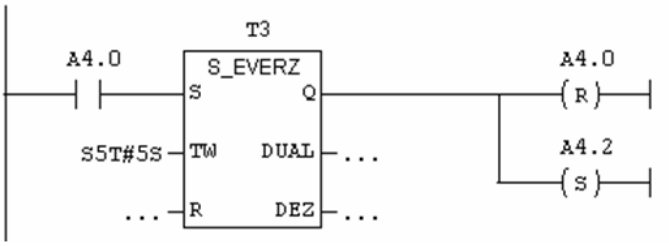
Segm. 2 : A LOS 5 SEGUNDOS ENCENDER EL AMARILLO



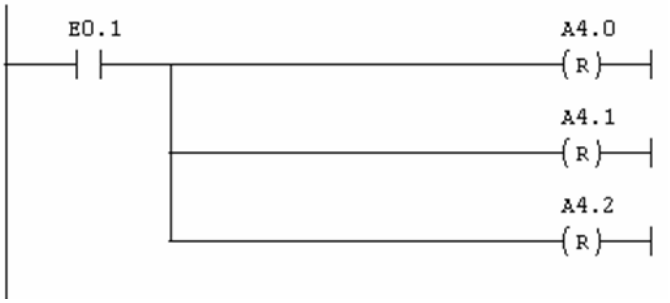
Segm. 3 : A LOS 2 S. APAGAR AMARILLO Y VERDE Y ENCENDER ROJO



Segm. 4 : A LOS 5 SEG. APAGAR ROJO Y ENCENDER VERDE



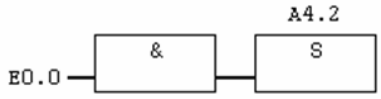
Segm. 5 : SI SE PULSA PARA APAGAR TODO



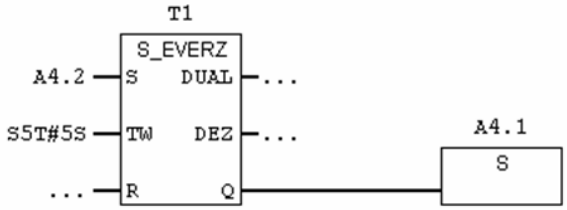
Solución en FUP

OB1 : SEMAFORO

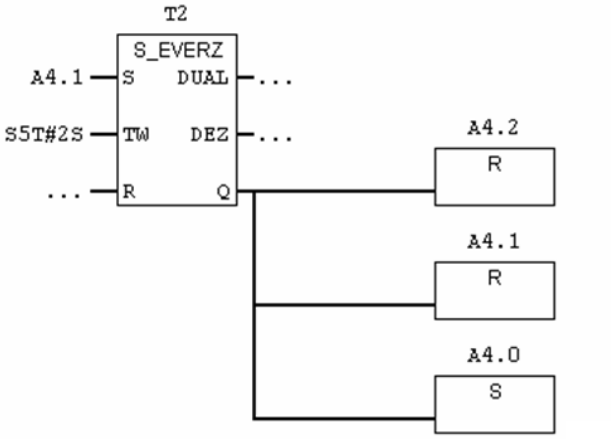
Segm. 1 : ENCENDER EL VERDE



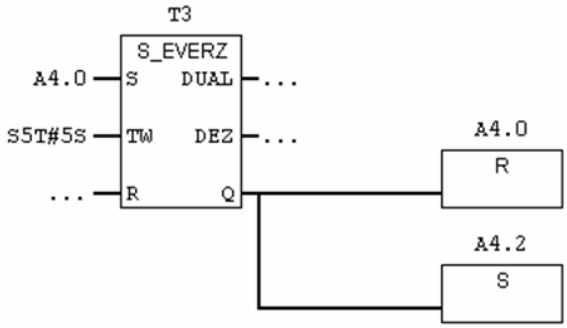
Segm. 2 : A LOS 5 SEGUNDOS ENCENDER EL AMARILLO



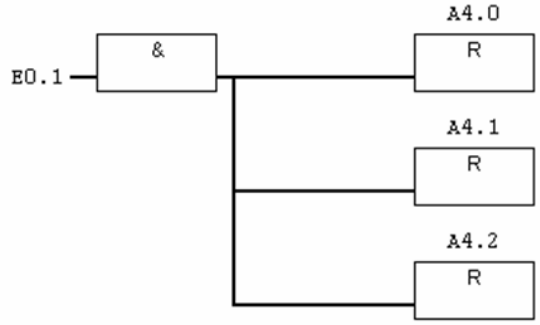
Segm. 3 : A LOS 2 S. APAGAR AMARILLO Y VERDE Y ENCENDER ROJO



Segm. 4 : A LOS 5 SEG. APAGAR ROJO Y ENCENDER VERDE

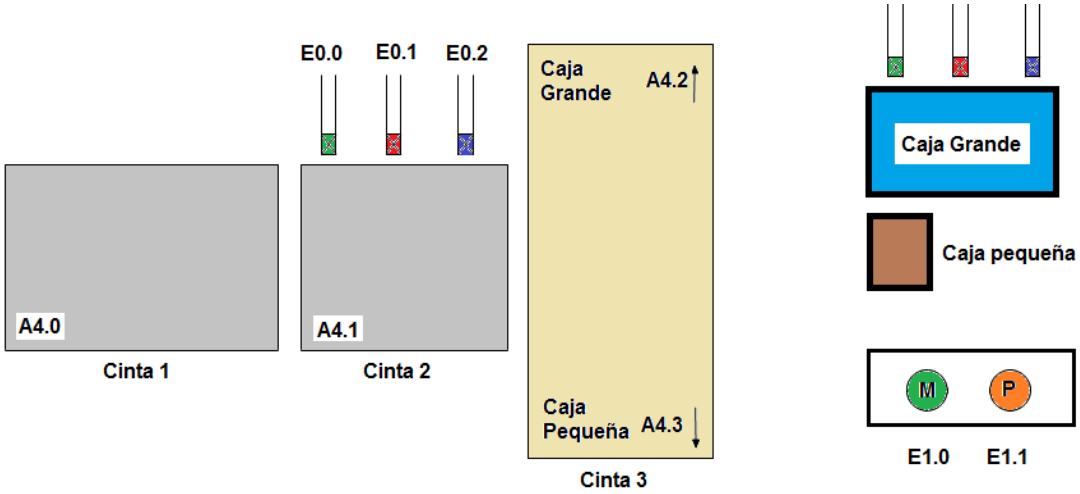


Segm. 5 : SI SE PULSA PARA APAGAR TODO



EJERCICIO 11. Control de un Sistema de Cintas transportadoras (Temporizadores)

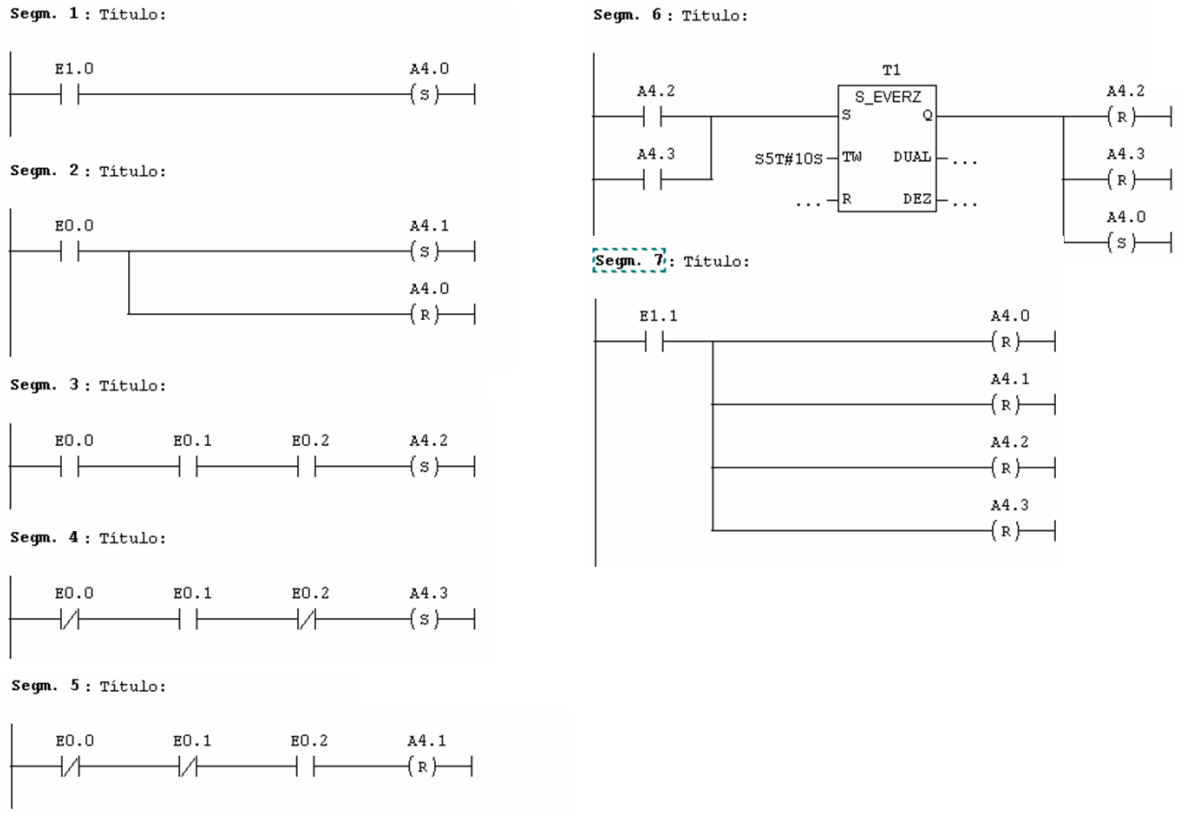
Tenemos tres cintas transportadoras dispuestas como indica la figura. Por las cintas transportadoras van a circular cajas grandes y pequeñas indistintamente. El tamaño de las cajas es detectado por tres sensores. Para cajas grandes los tres sensores se activan. Para las pequeñas sólo el primero de ellos.



El funcionamiento del sistema debe ser el siguiente:

Cuando le demos al pulsador de marcha queremos que se ponga en marcha la cinta nº 1. Cuando llegue la primera caja a la cinta nº 2, queremos que se pare la cinta nº 1 y que se ponga en marcha la cinta nº 2. En la cinta nº 2 detectamos si la caja es grande o pequeña. Si es grande, queremos que se ponga en marcha la tercera cinta hacia arriba, y si es pequeña queremos que se ponga en marcha la tercera cinta hacia abajo. La cinta nº 2 se para cuando la caja ya esté abandonando la cinta n°2. La cinta nº 3 se para a los 10 seg. de haberse puesto en marcha. A continuación se pone en marcha de nuevo la primera cinta y vuelve a comenzar el ciclo.

SOLUCIÓN:



OPERACIONES DE SALTO

Step 7 pone a disposición del programador varias operaciones de salto de programa. Estas operaciones se pueden utilizar en todos los bloques lógicos: bloques de organización (OBs), bloques de función (FBs) y funciones (FCs).

Se dispone de las operaciones de salto siguientes:

- ---(JMP)--- Salto absoluto
- ---(JMP)--- Salto condicional
- ---(JMPN)--- Saltar si la señal es 0

Las operaciones de salto requieren un operando o meta. La meta indica el destino a donde se desea saltar en el programa. Ésta se introduce encima de la bobina de salto, y se compone de cuatro caracteres como máximo. El primer carácter ha de ser una letra del alfabeto; los restantes caracteres pueden ser letras o números (p.ej. SEG3).

La meta de destino (LABEL) ha de encontrarse siempre al principio de un segmento. Para introducirla hay que seleccionar LABEL en el cuadro KOP.

---(JMP)--- Salto absoluto. En el ejemplo de la figura 1, la operación de salto --(JMP)— tiene como nombre de operando o meta “fin”. Al ejecutarse el segmento 2, el programa saltará a la meta de destino (LABEL) del segmento 4. En este caso, no se ejecutará lo programado en el segmento 3. Esto se conoce como salto absoluto.

---(JMP)--- Salto condicional. La figura 2 muestra que la operación de salto está condicionada a que E0.1 = 1

---(JMPN)--- Salto condicional si la señal es 0. La figura 3 muestra que la operación de salto está condicionada a que E0.1 = 0. Ni no se introduce este contacto, se trataría de un salto absoluto.

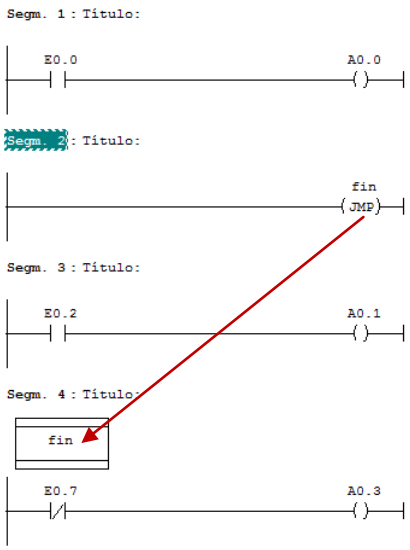


Figura 1. Salto absoluto.

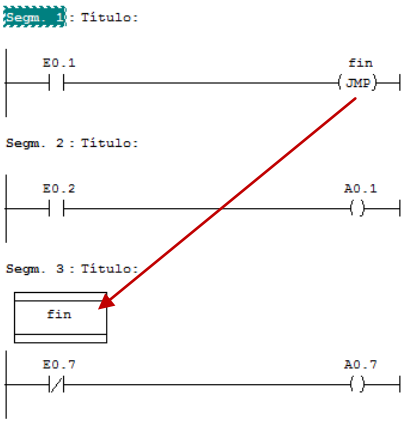


Figura 2. Salto condicional si E0.1=1.

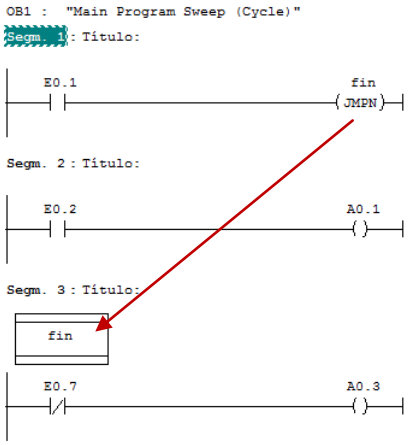
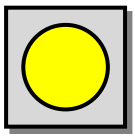


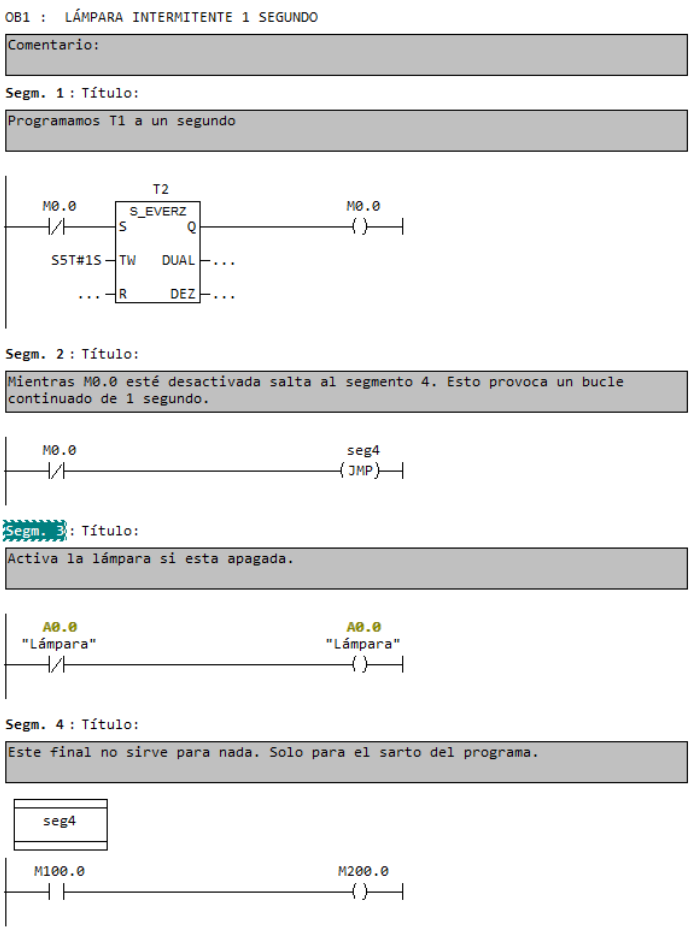
Figura 3. Salto condicional si E0.1=0

EJERCICIO 12. Lámpara intermitente de 1 segundo

Como ejemplo de uso de las operaciones de salto, vamos a programar un intermitente utilizando un solo temporizador de 1 segundo. Queremos que la lámpara esté activa un segundo y no activa otro segundo. Queremos que haga esto sin ninguna condición previa.

Solución en KOP:





Solución en AWL:

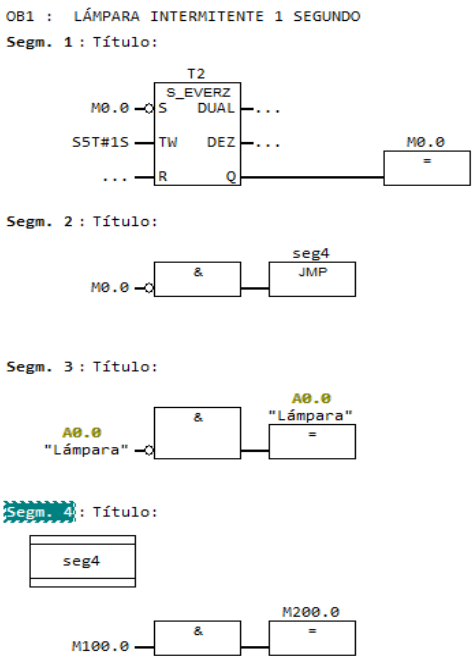
```
OB1 :   LÁMPARA INTERMITENTE 1 SEGUNDO
Segm. 1 : Título:
      UN    M      0.0
      L     S5T#1S
      SE    T      2
      NOP   0
      NOP   0
      NOP   0
      U     T      2
      =     M      0.0

Segm. 2 : Título:
      UN    M      0.0
      SPB   seg4

Segm. 3 : Título:
      UN    "Lámpara"                A0.0
      =     "Lámpara"                A0.0

Segm. 4 : Título:
seg4: U     M    100.0
      =     M    200.0
```

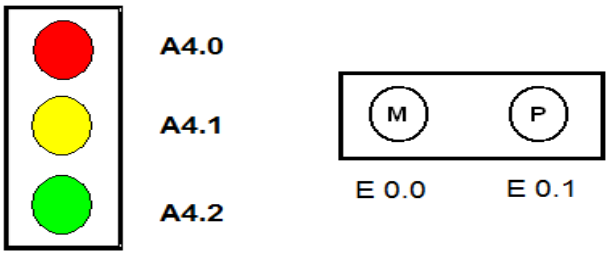
Solución en FUP:



EJERCICIO 13. Control de un semáforo con luz amarilla intermitente (Temporizadores y Saltos)

Tenemos un semáforo con las tres luces verde, ámbar y rojo. Disponemos también de dos pulsadores de mando: un pulsador de marcha y un pulsador de paro. El ciclo comienza tras pulsar el pulsador de marcha, siguiendo la siguiente secuencia indicada. El ciclo es repetitivo hasta que se pulse el pulsador de paro. En ese momento se apaga todo. Siempre que le dé al pulsador de marcha quiero que empiece por el verde.

Como puede verse, el funcionamiento es idéntico al del ejercicio nº 10, pero con el ciclo modificado. Aquí, la luz amarilla queda intermitente.

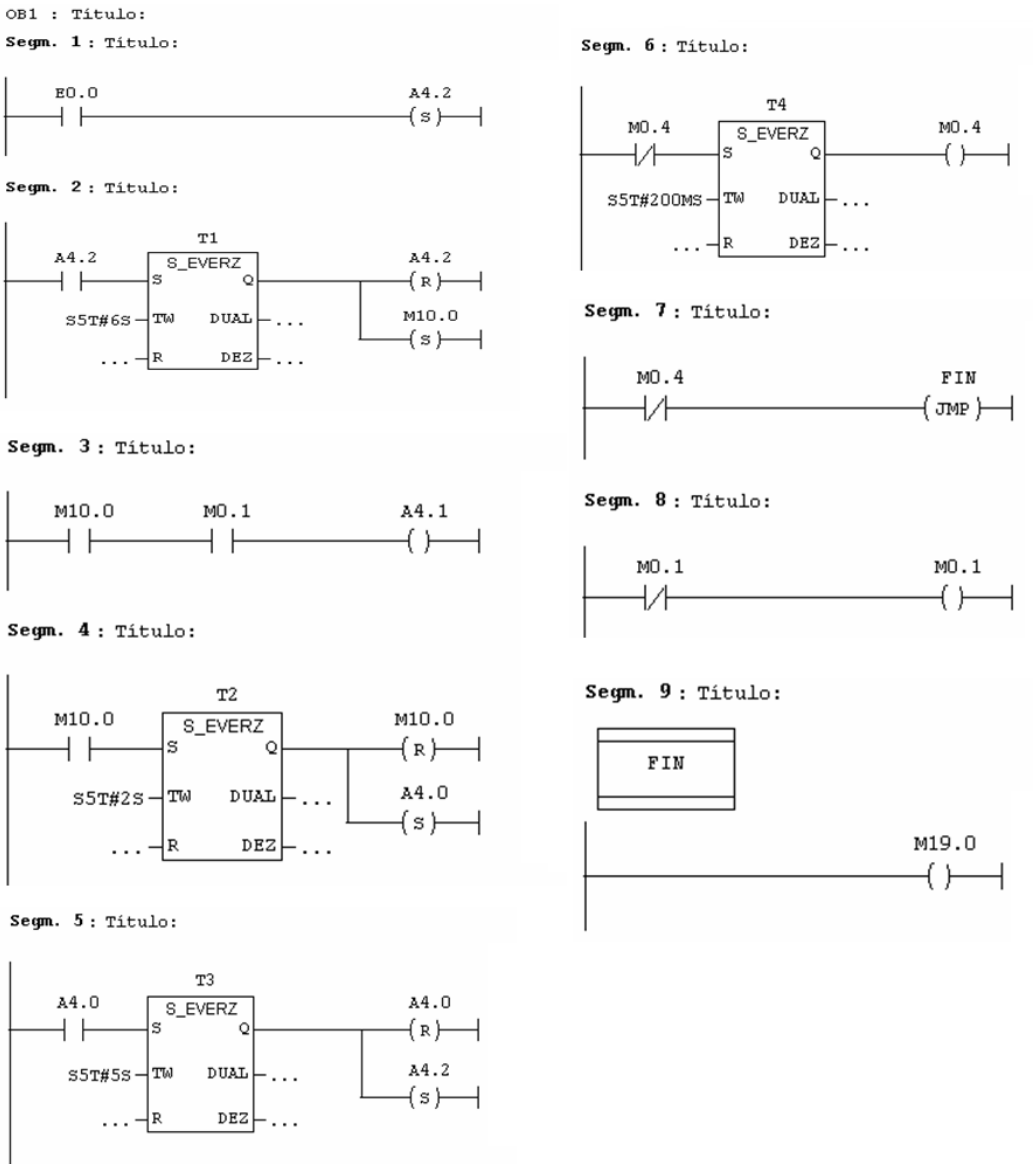


1º/ Verde durante 5 seg.

2º/ Amarillo intermitente durante 2 seg.

3º/ Rojo durante 6 seg

Solución en KOP



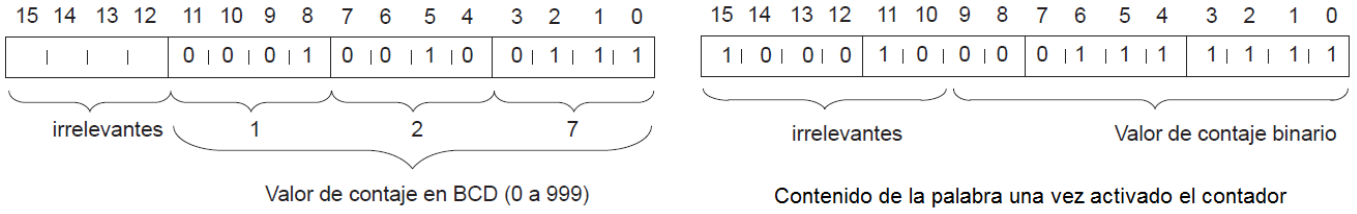
CONTADORES

En Step 7 los contadores se referencian como Z. Para éstos se reserva un área de memoria de 256 palabras de 16 bits por contador. De esta forma podremos utilizar hasta 256 contadores, dependiendo del tipo de CPU que utilicemos.

Los bits 0 a 9 de cada palabra de contaje contienen el valor de contaje en código binario. El valor fijado por el usuario se transfiere del acumulador al contador al activarse éste. El valor de contaje puede estar comprendido entre 0 y 999. Dentro de este margen se puede variar dicho valor utilizando las operaciones Incrementar y Decrementar contador.

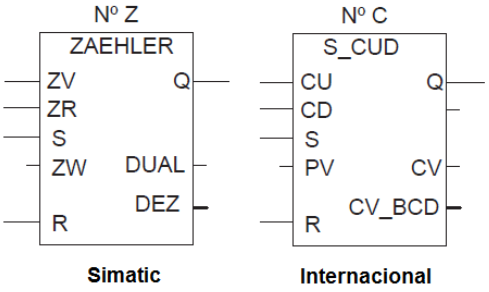
Para poner el contador a un valor determinado hay que introducir un número de 0 a 999, por ejemplo 127, en el siguiente formato: **C# 127.** C# sirve para indicar el formato decimal codificado en binario (formato BCD: cada conjunto de cuatro bits contiene el código binario de un valor decimal).

Los bits 0 a 11 del contador contienen el valor de contaje en formato BCD. La siguiente figura muestra el contenido del contador después de haber cargado el valor de contaje 127 y el contenido de la palabra de contaje después de haber activado el contador.



CONTADOR para Incremento-Decremento *(ZAEHLER)*

El contador se activa al producirse un flanco positivo en las entradas ZV (incremento) o ZR (decremento). Si activamos ZV, el valor del contador se incrementa en 1, siempre y cuando el valor del contador sea menor de 999. Si activamos ZR, el valor del contador se decrementa en 1, siempre y cuando el valor del contador sea mayor de 0. Si activamos ambas entradas a la vez, se procesan las dos operaciones y el valor de contaje se mantiene invariable.



El contador se desactiva cuando se produce un flanco positivo en la entrada R (reset), en cuyo caso el contador pone el valor de contaje a 0.

La salida Q será siempre 1 si el valor de contaje es mayor de 0. La salida Q será 0 si el valor de contaje es 0.

El valor del contador (0-999) se introduce en la entrada ZW con el formato C#xxx (ejemplo C#40).

La entrada S pone el valor del contador al valor de preselección indicado en ZW.

La salida DUAL muestra el valor actual del contador en formato de número entero.

La salida DEZ muestra el valor actual del contador en formato BCD.

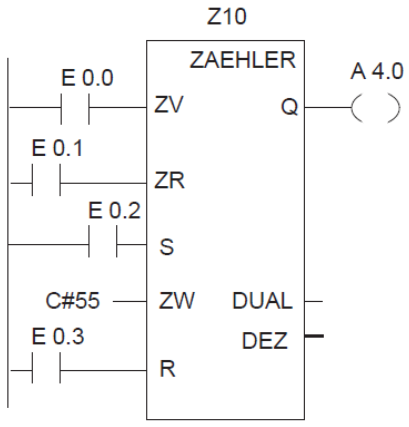
Cuadro Incrementar y decrementar contador y parámetros con abreviatura SIMATIC

Cuadro KOP	Parámetro	Tipo de datos	Area de memoria	Descripción
Nº Z ZAEHLER ZV Q ZR S ZW DUAL DEZ R	Nº del contador	COUNTER	–	Número de identificación del contador. El área depende de la CPU utilizada
	ZV	BOOL	E, A, M, D, L	Entrada ZV: Incrementar contador
	ZR	BOOL	E, A, M, D, L	Entrada ZR: Decrementar contador
	S	BOOL	E, A, M, D, L	Entrada para poner un contador a un valor de preselección
	ZW	WORD	E, A, M, D, L	Valor, comprendido entre 0 y 999 (introducido como C#<valor> para indicar formato BCD) para desactivar el contador
	R	BOOL	E, A, M, D, L	Entrada de desactivación
	Q	BOOL	E, A, M, D, L	Estado del contador
	DUAL	WORD	E, A, M, D, L	Valor de contaje actual (formato entero)
	DEZ	WORD	E, A, M, D, L	Valor de contaje actual (formato BCD)

Cuadro Incrementar y Decrementar contador y parámetros con abreviatura internacional

Cuadro KOP	Parámetro	Tipo de datos	Area de memoria	Descripción
Nº C S_CUD CU Q CD S PV CV CV_BCD R	Nº del contador	COUNTER	–	Número de identificación del contador. El área depende de la CPU utilizada.
	CU	BOOL	E, A, M, D, L	Entrada CU: Incrementar contador
	CD	BOOL	E, A, M, D, L	Entrada CD: Decrementar contador
	S	BOOL	E, A, M, D, L	Entrada para poner un contador a un valor de preselección
	PV	WORD	E, A, M, D, L	Valor comprendido entre 0 y 999 (introducido como C#<valor> para indicar formato BCD) para desactivar el contador
	R	BOOL	E, A, M, D, L	Entrada de desactivación
	Q	BOOL	E, A, M, D, L	Estado del contador
	CV	WORD	E, A, M, D, L	Valor de contaje actual (formato entero)
	CV_BCD	WORD	E, A, M, D, L	Valor de contaje actual (formato BCD)

Ejemplo de funcionamiento:



Un cambio del estado de señal de 0 a 1 en la entrada E 0.2 pone el contador Z 10 al valor 55.

Si el estado de señal de E 0.0 cambia de 0 a 1, el valor del contador Z 10 se incrementa en uno, a menos que el valor de Z 10 sea igual a 999.

Si la entrada E 0.1 cambia de 0 a 1, Z 10 se decrementa en uno, a menos que el valor de Z 10 sea igual a 0.

Si E 0.3 cambia de 0 a 1, el valor del contador de Z10 se pone a 0.
El estado de señal de la salida A 4.0 es 1, a menos que Z 10 sea igual a 0.

CONTADOR para Incremento *(Z_VORW)*

Si activamos mediante un flaco positivo la entrada ZV, el valor del contador se incrementa en 1, siempre y cuando el valor del contador sea menor de 999. El contador se desactiva y se pone a 0 si se produce un flanco positivo en la entrada R. El resto de E/S funcionan igual que el contador anterior.

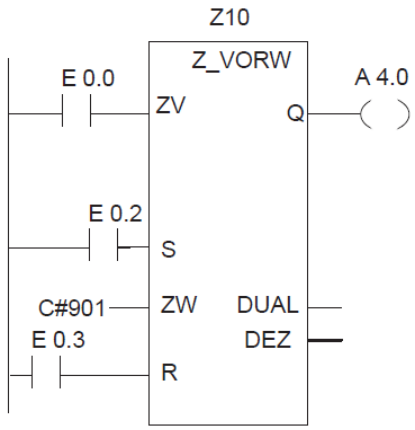
Cuadro Incrementar contador y parámetros con abreviatura SIMATIC

Cuadro KOP	Parámetro	Tipo de datos	Area de memoria	Descripción
Nº Z Z_VORW ZV Q S ZW DUAL DEZ R	Nº del contador	COUNTER	–	Número de identificación del contador. El área depende de la CPU utilizada.
	ZV	BOOL	E, A, M, D, L	Entrada ZV: Incrementar contador
	S	BOOL	E, A, M, D, L	Entrada para poner un contador a un valor de preselección
	ZW	WORD	E, A, M, D, L	Valor, comprendido entre 0 y 999 (introducido como C#<valor> para indicar formato BCD) para desactivación del contador.
	R	BOOL	E, A, M, D, L	Entrada de desactivación
	Q	BOOL	E, A, M, D, L	Estado del contador
	DUAL	WORD	E, A, M, D, L	Valor de contaje actual (formato entero)
	DEZ	WORD	E, A, M, D, L	Valor de contaje actual (formato BCD)

Cuadro Incrementar contador y parámetros con abreviatura internacional

Cuadro KOP	Parámetro	Tipo de datos	Area de memoria	Descripción
Nº C S_CU CU Q S PV CV CV_BCD R	Nº del contador	COUNTER	–	Número de identificación del contador. El área depende de la CPU utilizada.
	CU	BOOL	E, A, M, D, L	Entrada CU: Incrementar contador
	S	BOOL	E, A, M, D, L	Entrada para poner un contador a un valor de preselección
	PV	WORD	E, A, M, D, L	Valor, comprendido entre 0 y 999 (introducido como C#<valor> para indicar formato BCD) para desactivar el contador
	R	BOOL	E, A, M, D, L	Entrada de desactivación
	Q	BOOL	E, A, M, D, L	Estado del contador
	CV	WORD	E, A, M, D, L	Valor de contaje actual (formato entero)
	CV_BCD	WORD	E, A, M, D, L	Valor de contaje actual (formato BCD)

Ejemplo de funcionamiento:



Un cambio del estado de señal de 0 a 1 en la entrada E 0.2 pone el contador Z 10 al valor 901.

Si el estado de señal de E 0.0 cambia de 0 a 1, el valor del contador Z 10 se incrementa en uno, a menos que el valor de Z 10 sea igual a 999.

Si E 0.3 cambia de 1 a 0, el valor del contador Z10 se pone a 0.

El estado de señal de la salida A 4.0 es 1, a menos que Z 10 sea diferente de 0.

CONTADOR para Decremento *(Z_RUECK)*

Si activamos mediante un flaco positivo la entrada ZR, el valor del contador se decrementa en 1, siempre y cuando el valor del contador sea menor de 999. El contador se desactiva y se pone a 0 si se produce un flanco positivo en la entrada R. El resto de E/S funcionan igual que el contador anterior.

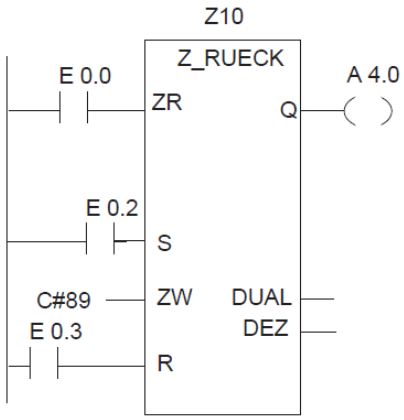
Cuadro Decrementar contador y parámetros con abreviatura SIMATIC

Cuadro KOP	Parámetro	Tipo de datos	Area de memoria	Descripción
Nº Z Z_RUECK ZR Q S ZW DUAL DEZ R	Nº del contador	COUNTER	E, A, M, D, L	Número de identificación del contador. El área depende de la CPU utilizada.
	ZR	BOOL	E, A, M, D, L	Entrada ZR: Decrementar contador
	S	BOOL	E, A, M, D, L	Entrada para poner un contador a un valor de preselección
	ZW	WORD	E, A, M, D, L	Valor, comprendido entre 0 y 999 (introducido como C#<valor> para indicar formato BCD) para desactivación del contador
	R	BOOL	E, A, M, D, L	Entrada de desactivación
	Q	BOOL	E, A, M, D, L	Estado del contador
	DUAL	WORD	E, A, M, D, L	Valor de contaje actual (formato entero)
	DEZ	WORD	E, A, M, D, L	Valor de contaje actual (formato BCD)

Cuadro Decrementar contador y parámetros con abreviatura internacional

Cuadro KOP	Parámetro	Tipo de datos	Area de memoria	Descripción
Nº C S_CD CD Q S PV CV CV_BCD R	Nº del contador	COUNTER	–	Número de identificación del contador. El área depende de la CPU utilizada.
	CD	BOOL	E, A, M, D, L	Entrada CD: Decrementar contador
	S	BOOL	E, A, M, D, L	Entrada para poner un contador a un valor de preselección
	PV	WORD	E, A, M, D, L	Valor, comprendido entre 0 y 999 (introducido como C#<valor> para indicar formato BCD) para desactivación del contador
	R	BOOL	E, A, M, D, L	Entrada de desactivación
	Q	BOOL	E, A, M, D, L	Estado del contador
	CV	WORD	E, A, M, D, L	Valor de contaje actual (formato entero)
	CV_BCD	WORD	E, A, M, D, L	Valor de contaje actual (formato BCD)

Ejemplo de funcionamiento:



Un cambio del estado de señal de 0 a 1 en la entrada E 0.2 pone el contador Z 10 al valor 89.

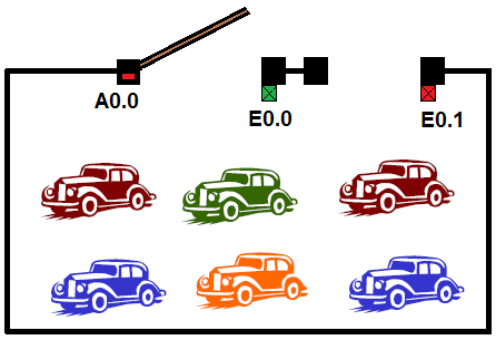
Si el estado de señal de E 0.0 cambia de 0 a 1, el valor del contador se decrementa en uno, a menos que el valor de Z 10 sea igual a 0.

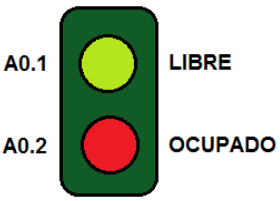
El estado de señal de la salida A 4.0 es 1 si Z 10 es diferente de 0.

Si E 0.3 cambia de 0 a 1, el valor del contador Z10 se pone a 0.

EJERCICIO 14. Control de un Parking. (Contadores y Comparadores)

El funcionamiento parking queremos que sea el siguiente:





Cuando llega un coche y el parking esté libre, queremos que se abra la barrera. A la salida no tenemos barrera. Cuando sale un coche simplemente sabemos que ha salido.

En el parking caben 10 coches. Cuando el parking tenga menos de 10 coches queremos que esté encendida la luz de libre. Cuando en el parking haya 10 coches queremos que esté encendida la luz de ocupado.

Además, queremos que si el parking está ocupado y llega un coche que no se le abra la barrera.

SOLUCIÓN EN AWL

```
OB1 : Parking de Coches
Comentario:

Segm. 1 : Título:
Si llega un coche y el parking está libre, abrimos la barrera

      U     "Sensor Entrada"          E0.0
      U     "Libre"                   A0.1
      =     "Barrera"                 A0.0

Segm. 2 : Título:
Contamos y descontamos el nº de coches

      U     "Barrera"                 A0.0
      ZV    Z      1
      U     "Sensor Salida"           E0.1
      ZR    Z      1
      NOP   0
      NOP   0
      NOP   0
      L     Z      1
      T     MW     10
      NOP   0
      NOP   0

Segm. 3 : Título:
Si el nº de coches es inferior a 10 encendemos la luz verde de "Libre".

      L     MW     10
      L     10
      <I
      =     "Libre"                   A0.1

Segm. 4 : Título:
Si no está libre, activamos la luz de ocupado.

      UN    "Libre"                   A0.1
      =     "Ocupado"                 A0.2
```

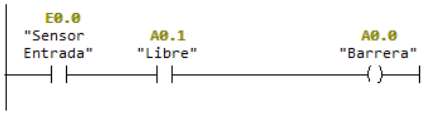
SOLUCIÓN EN KOP

OB1 : Parking de Coches

Comentario:

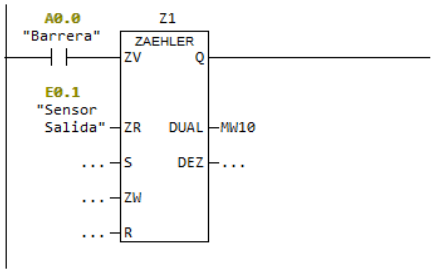
Segm. 1 : Título:

Si llega un coche y el parking está libre, abrimos la barrera



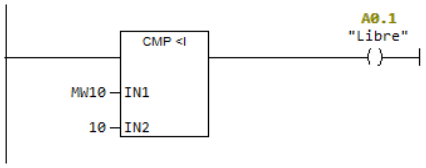
Segm. 2 : Título:

Contamos y descontamos el nº de coches



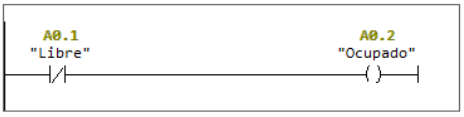
Segm. 3 : Título:

Si el nº de coches es inferior a 10 encendemos la luz verde de "Libre".



Segm. 4 : Título:

Si no está libre, activamos la luz de ocupado.



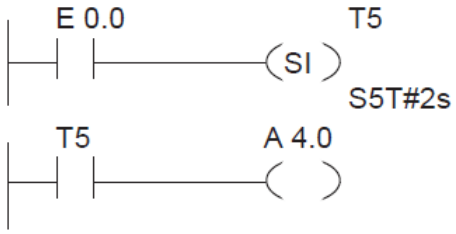
TEMPORIZADORES SI, SE, SS, SV y SA.

Además de los temporizadores que hemos que hemos visto en ejercicios anteriores, tenemos otros dos (SI y SE), y además otros tres (SS,SV y SA) llamados temporizadores con memoria.

Temporizador de impulso (SI)

Este temporizador se comporta igual que el temporizador S_IMPULS. Este arranca con el valor de temporización programado, tras recibir un flanco positivo. El temporizador sigue contando con el valor determinado mientras la entrada esté activa. Si el valor de entrada cambia de 1 a 0 antes de finalizar el tiempo indicado, se para el temporizador. En este caso, la consulta si el estado de señal es 1, produce un 0 como resultado. Las unidades de tiempo son d (días), h (horas), m (minutos), s (segundos) y ms (milisegundos).

Ejemplo:



Si se produce un flanco positivo (cambio de 0 a 1) en E0.0, el temporizador T5 se pone en marcha. El temporizador continúa en marcha con el valor indicado de 2 segundos mientras el estado de señal de E 0.0 sea 1. Si el estado de señal de E 0.0 cambia de 1 a 0 antes de finalizar el tiempo indicado, se para el temporizador.

El estado de señal de la salida A 4.0 es 1 mientras el temporizador esté en marcha.

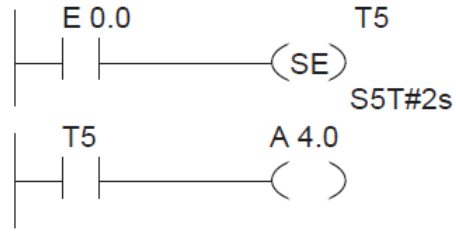
Ejemplos de valores de temporización:

S5T#2s = 2 segundos S5T#12m_18s = 12 minutos y 18 segundos

Temporizador de retardo a la conexión (SE)

Este es equivalente al temporizador S_EVERZ. Cuando la señal de activación produce un flanco positivo, el temporizador arranca con el valor de temporización programado. El valor de salida será 1 mientras la entrada de activación sea 1. Si la señal de activación pasa de 0 a 1 el temporizador se para y la salida se hace 0.

Ejemplo:



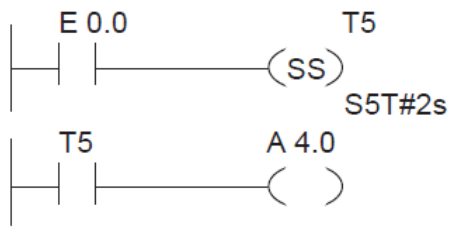
Si el estado de señal de la entrada E 0.0 cambia de 0 a 1 (es decir, si se produce un flanco positivo en el RLO), el temporizador T5 arranca. Si al finalizar el tiempo indicado, el estado de señal de la entrada E 0.0 aún es 1, la salida A 4.0 es 1. Si el estado de señal de la entrada E 0.0 cambia de 1 a 0, el temporizador se para y la salida A 4.0 es 0.

Temporizador de retardo a la conexión con memoria (SS)

El temporizador SS es equivalente al temporizador SE. El funcionamiento es similar. La diferencia está en que el funcionamiento del temporizador es independiente de la entrada.

Este temporizador una vez arrancado (flaco positivo a la entrada) continúa con el valor de temporización aunque la entrada cambie a 0 antes de finalizar el tiempo. La salida es 1 cuando finaliza el tiempo programado. El temporizador arranca (dispara) nuevamente con el valor indicad o si el RLO cambia de 0 a 1 mientras el temporizador está en marcha.

Ejemplo:



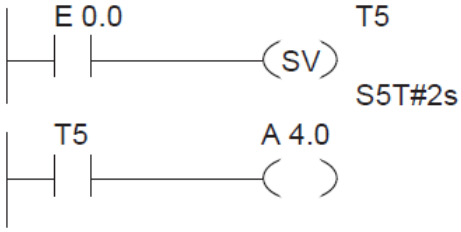
Si el estado de señal de la entrada E 0.0 cambia de 0 a 1 (es decir, si se produce un flanco positivo en el RLO), el temporizador T 5 arranca. El temporizador continúa funcionando aunque cambie de 1 a 0 en la entrada E 0.0. Si el estado de señal de E 0.0 cambia de 0 a 1 antes de finalizar el tiempo, el temporizador arranca nuevamente. El estado de señal de la salida A 4.0 es 1 cuando el tiempo ha finalizado.

Temporizador de impulso prolongado (SV)

Este temporizador una vez arrancado (flaco positivo a la entrada), continúa funcionando con el valor programado aunque la entrada cambie de 1 a 0 antes de finalizar el valor temporizado. Mientras el temporizador está en marcha la salida está a 1.

El temporizador arranca (se dispara) nuevamente si la entrada cambia de 0 a 1 mientras el temporizador está en marcha.

Ejemplo:

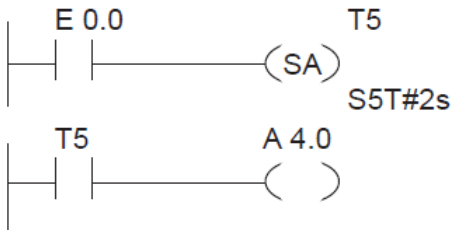


Si el estado de señal de E 0.0 cambia de 0 a 1, el temporizador T5 se pone en marcha. Este continúa en marcha sin atender al flanco negativo en el RLO. Si el estado de señal de E 0.0 cambia de 0 a 1 antes de finalizar el tiempo indicado, el temporizador se dispara nuevamente. El estado de señal de la salida A 4.0 es 1 mientras el temporizador esté en marcha

Temporizador de retardo a la desconexión (SA)

El temporizador arranca si se produce un flanco negativo en el RLO (es decir, si el RLO cambia de 1 a 0). Mientras está en marcha o la entrada es 1 la salida será 1. El temporizador vuelve a ponerse a 0 si el RLO cambia de 0 a 1 mientras está en marcha. El temporizador no vuelve a arrancar mientras el RLO no cambie de 1 a 0.

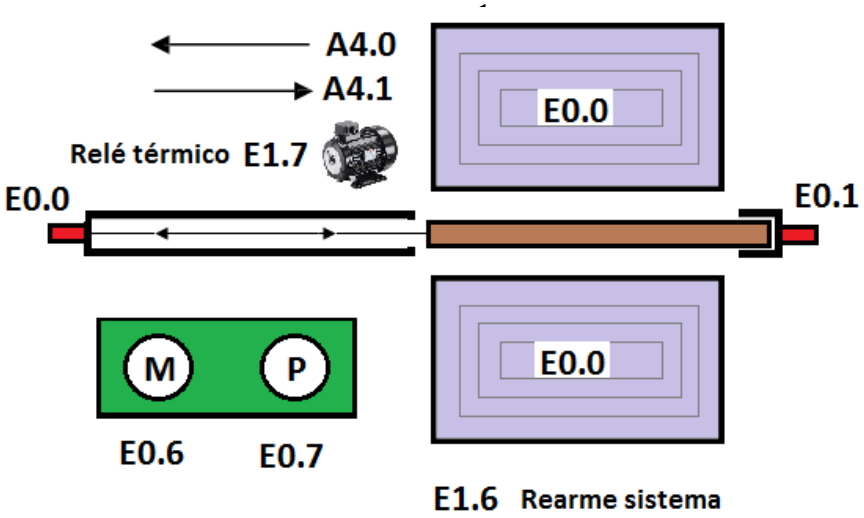
Ejemplo:



El temporizador arranca si el estado de señal de la entrada E 0.0 cambia de 1 a 0. Si el estado de la señal de E 0.0 cambia de 0 a 1, el temporizador se desactiva. El estado de señal de la salida A 4.0 es 1 si el estado de señal de la entrada E 0.0 es 1 o si el temporizador está en marcha.

EJERCICIO 15. Control de una Puerta Corredera (Temporizadores + Contadores).

Queremos controlar un acceso mediante una puerta corredera. El funcionamiento de la puerta es el siguiente:



Queremos que cuando alguien pise en la goma del suelo, se abra la puerta (Motor A4.0). La puerta se abre hasta que llegue al final de carrera (E0.0). Cuando llega al final de carrera, comienza a cerrarse (Motor A 4.1), hasta que llega al final de carrera (E0.1).

Tenemos dos pulsadores de control. El de marcha (M) y el de paro (P). Cuando le demos al pulsador de marcha queremos que el funcionamiento sea el que hemos explicado anteriormente. Cuando le demos al de paro queremos que deje de funcionar. Es decir, si alguien pisa la goma no queremos que se abra la puerta.

También queremos que cuando salte el relé térmico (E1.7) se pare la puerta hasta que lo rearmemos (E1.6). Cuando haya saltado el relé térmico 5 veces queremos que se bloquee la puerta.

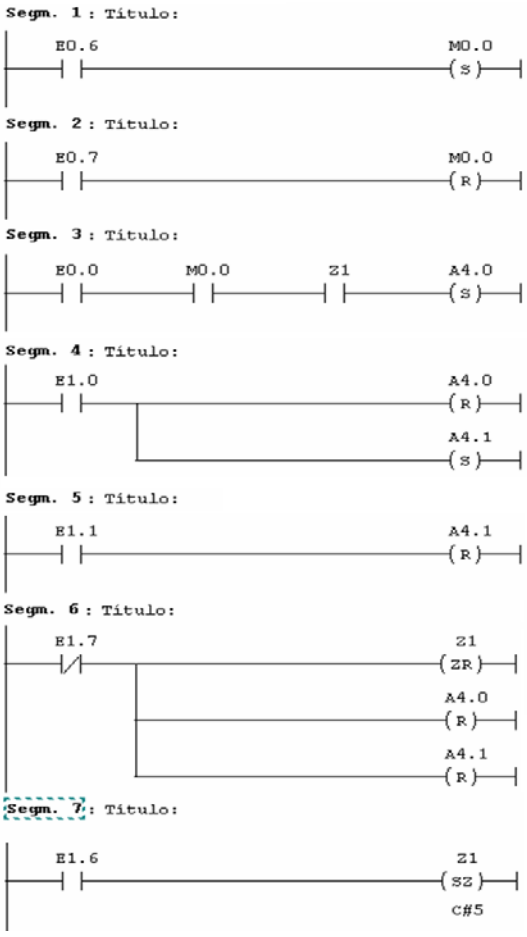
El sistema volverá a funcionar cuando desbloqueemos la puerta.

SOLUCIÓN EN AWL

```
U   E   0.6   //Si le damos al pulsador de marcha
S   M   0.0   //Activa la marca 0.0
U   E   0.7   //Si le damos al pulsador de paro
R   M   0.0   //Desactiva la marca 0.0
U   E   0.0   //Si alguien pisa la goma
U   M   0.0   //Y está la puerta en marcha
U   Z   1     //Y el contador 1 tiene un valor distinto de 0
S   A   4.0   //Y activa el motor de abrir
U   E   1.0   //Si llega el final de carrera
R   A   4.0   //Para el motor de apertura
S   A   4.1   //Y pon en marcha el motor de cierre
U   E   1.1   //Si se ha cerrado la puerta
R   A   4.1   //Para el motor de cierre
UN  E   1.7   //Si ha saltado el relé térmico
ZR  Z   1     //Descuenta una unidad en el contador 1
R   A   4.0   //Y para el motor de abrir
R   A   4.1   //Y para el motor de cerrar
U   E   1.6   //Si activamos la entrada 1.6
L   C#5       //Carga un 5

S   Z   1     //Y mételo en el contador 1
BE            // Fin del programa
```

SOLUCIÓN EN KOP



EJERCICIO 16: CONTAR Y DESCONTAR CADA SEGUNDO (Intermitente + contadores).

Queremos hacer un contador que a partir de que le demos al pulsador de marcha, comience a contar una unidad cada segundo hasta llegar a 60. Cuando llegue a 60 queremos que siga contando una unidad cada segundo pero en sentido descendente.

Queremos que haga la siguiente cuenta:

0, 1, 2, 3, 4, ..., 58, 59, **60**, 59, 58, 57,....., 2, 1, 0, 1, 2, ...

Cuando le demos al pulsador de paro queremos que deje de contar. Cuando le demos de nuevo al pulsador de marcha probaremos dos cosas:

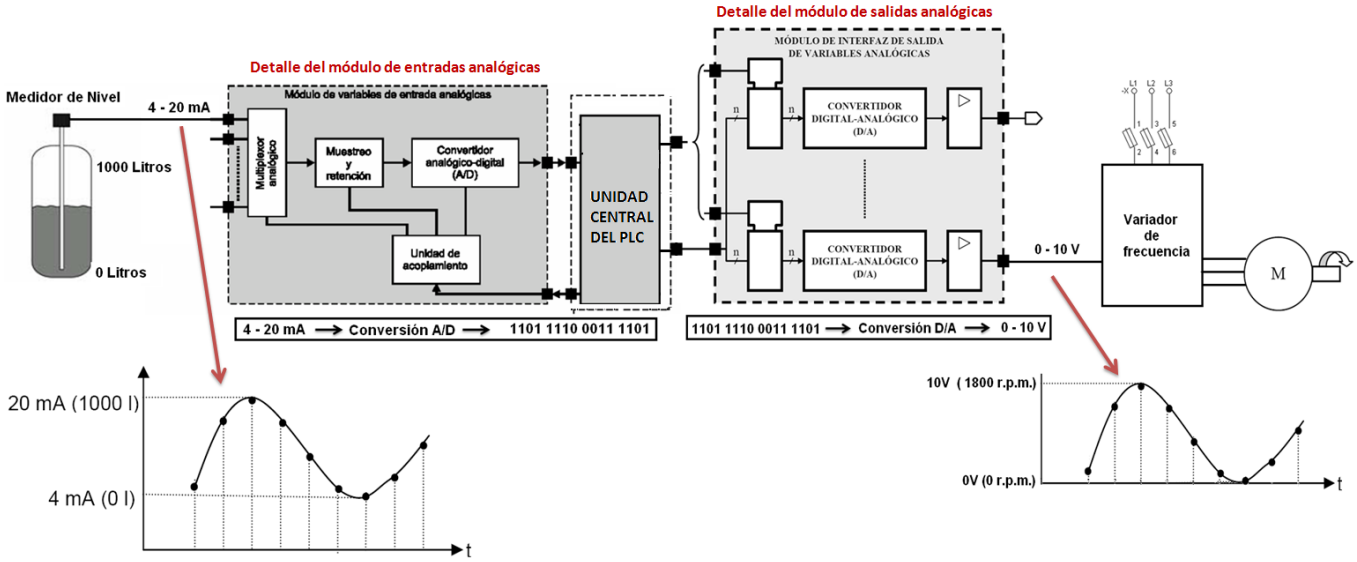
1) Que siga contando por donde iba.
2) Que empiece a contar otra vez desde cero.

SOLUCIÓN EN AWL

```
UN    M     0.0          //Hacemos que la marca 0.0
L     S5T#1S             //Se active un ciclo cada segundo
SE    T     1
U     T     1
=     M     0.0
U     E     0.0          //Si le damos al pulsador de marcha
S     M     0.1          //Se activa la marca 0.1
R     M     0.3          //Y se desactiva la marca 0.3
(R    Z     1)           Empezará desde cero o pro donde
U     M     0.1          //Si está activa la marca 0.1
U     M     0.0          //Y llega un pulsa de la marca 0.0
UN    M     0.3          //Y no está activa la marca 0.3
ZV    Z     1            //Cuanta una unidad con el
                         contador 1
L     Z     1            //Carga el contador 1
L     60                 //Carga un 60
==I                      //Cuando sean iguales
S     M     0.2          //Activa la marca 0.2
R     M     0.1          //Y desactiva la marca 0.1
U     M     0.2          //Si está la marca 0.2
U     M     0.0          //Y llega un pulso de la marca 0.0
UN    M     0.3          //Y no está la marca 0.3
ZR    Z     1            //Descuenta 1 con el contador 1
L     Z     1            //Carga el contador 1
L     0                  //Carga un 0
==I                      //Si son iguales
S     M     0.1          //Activa la marca 0.1
R     M     0.2          //Y desactiva la marca 0.2
U     E     0.1          //Si le damos al paro
S     M     0.3          //Activa la marca 0.3
BE
```

VALORES ANALÓGICOS DE ENTRADA/SALIDA

Aunque los PLC´s trabajan internamente con datos digitales, también pueden admitir señales analógicas que pueden tomar valores infinitos en el tiempo. Para ello es preciso que dichos PLC´s estén provistos de módulos que realicen conversiones A/D para las entradas y conversiones D/A para las salidas. De esta forma, cualquier PLC puede procesar y admitir valores analógicos procedentes por ejemplo de un sensor o dispositivo analógico, al mismo tiempo que pueden entregar señales analógicas ya procesadas para cualquier actuador analógico.



Estas entradas y salidas analógicas pueden ser en forma de **tensión** o de **corriente** dependiendo del tipo de PLC utilizado**.** Los valores de configuración típicos son:

TENSIÓN:	**+/- 10 V , +/- 5 V, 0 a 5 V y de 0 a 10 V.**
INTENSIDAD:	**+/- 20 mA, 0-20 mA y de 4-20 mA**

Las señales E/S son procesadas por el PLC como información en tamaño **palabra (16 bits).** El acceso a esta palabra se realiza con las instrucciones:

PEW x	para Entrada analógica en S/300	→ **PEW 0 a 65534**	→ **IW 0 a 65534** para S7-1200
PAW x	para Salida Analógica en S7-300	→ **PAW 0 a 65534**	→ **QW 0 a 65534** para S7-1200

El direccionamiento de las palabras de entrada/salida analógicas depende de la dirección de comienzo del módulo. Si el módulo analógico se coloca en el slot 4, su dirección de comienzo estándar es 256. La dirección de comienzo de cada módulo adicional se incrementa en 16 bytes. Esta dirección estándar se puede comprobar en la tabla de configuración hardware en la vista detallada.

Los valores de las E/S analógicos se transforman en un número entero de 16 bits.

Los números enteros son números que incluye a los números naturales distintos positivos (1, 2, 3, ...), los negativos (..., −3, −2, −1) y al 0. **Los números enteros no tienen parte decimal.**

EJEMPLO: +2, -1, 0, +12, -435, etc.

En realidad esto es así porque los convertidores A/D y D/A generalmente utilizan registros de 2 Byte (16 bit) de memoria para su funcionamiento.

WORD	BYTE ALTO								BYTE BAJO							
Posición del bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Peso del bit	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
Dato	S	VALOR DEL DATO DIGITAL														

Resolución Nº de BITS	Incremento mínimo del dato		Dato															
	Decimal	Hexadecimal	Byte alto								Byte bajo							
8	128	40H	S	0	0	0	0	0	0	0	1	x	x	x	x	x	x	x
9	64	30H	S	0	0	0	0	0	0	0	0	1	x	x	x	x	x	x
10	32	20H	S	0	0	0	0	0	0	0	0	0	1	x	x	x	x	x
11	16	10H	S	0	0	0	0	0	0	0	0	0	0	1	x	x	x	x
12	8	8H	S	0	0	0	0	0	0	0	0	0	0	0	1	x	x	X
13	4	4H	S	0	0	0	0	0	0	0	0	0	0	0	0	1	x	X
14	2	2H	S	0	0	0	0	0	0	0	0	0	0	0	0	0	1	x
15	1	1H	S	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

No obstante, no **debemos confundir estos 16 bits con la resolución de los convertidores A/D y D/A.** La resolución oscilará según el tipo de tarjeta entre 15 bits mas signo (+S) y 8 bits. Como es lógico, cuantos más bits de resolución tengan los convertidores, mayor definición y precisión, y mayor será la exactitud de la lectura o escritura. Por contra, también será mayor el tiempo de conversión y por lo tanto las variaciones en el proceso tardarán más en reflejarse en el PLC.

Si la resolución de los convertidores es menor a 15 bits (+ S) el dato es justificado a la izquierda con ceros en los bits menos significativos.

De forma genérica la distribución del **área de entradas y Salidas** que se utiliza es la siguiente:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS	2^{11} *2048*	2^{10} *1024*	2^{9} *512*	2^{8} *256*	2^{7} *128*	2^{6} *64*	2^{5} *32*	2^{4} *16*	2^{3} *8*	2^{2} *2*	2^{1} *2*	2^{0} *1*	X	E	D

ÁREA DE ENTRADAS

Bit de signo
BS = 0 valor positivo
BS = 1 valor negativo

Bits de datos 12 bits (resolución convertidor A/D)

Bit de error E= 1 Error

Bit de overflow D= 1 Desbordamiento

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS	2^{10} *1024*	2^{9} *512*	2^{8} *256*	2^{7} *128*	2^{6} *64*	2^{5} *32*	2^{4} *16*	2^{3} *8*	2^{2} *2*	2^{1} *2*	2^{0} *1*	X	X	X	X

ÁREA DE SALIDAS

Bit de signo
BS = 0 valor positivo
BS = 1 valor negativo

Bits de datos 11 bits (resolución convertidor D/A)

Rango de datos para E/S analógicas

WORD	BYTE ALTO								BYTE BAJO							
Posición del bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Peso del bit	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
Dato	S	VALOR DEL DATO DIGITAL														

En el área de 16 bits los valores posibles son: **2^{16} = 65536** (valor entero) (E/S unipolar).

Si la E/S es bipolar (+/-) necesitamos un bit de signo (S) y ahora los valores serían: **-32768 a +32768.**

EJEMPLO: Para un módulo de entradas de 12 bits la resolución será: 2^{12} = 4096. Pero nosotros veremos X cantidad de resolución, dependiendo de la tarjeta analógica.

- Tarjeta 8 bits= 28 =256 (2^{8}) de resolución x 128 (2^{7}) = 32768 cantidad de resolución.

- Tarjeta 12 bits= 212 =4096 (2^{12}) de resolución x 8 (2^{3}) = 32768 cantidad de resolución.

Dado que hay que tener en cuenta los valores de rebasamiento y desbordamiento, **la resolución en la práctica es 27648.**

Por ejemplo, para una entrada de +/- 10 V y medidas entre 0 y 1000 litros obtendré:

0 Litros ------ 0 500 Litros ------ 13824 1000 Litros ------ 27648

Límites de lectura

Lectura +/-10 V:

Estado de lectura	Tensión en la entrada anal.	Valor de la entrada anal.
Desbordamiento positivo	>= 11,759	32.767
Rebase positivo	De 11,7589 a 10,0004	De 32511 a 27649
Valor nominal	De 10,0 a –10,0	De 27648 a –27648
Rebase negativo	De -10,0004 a -11,759	De –27649 a –32512
Desbordamiento negativo	<= -11,76	-32.768

Lectura 0 a 20 mA:

Estado de lectura	Corriente (mA)	Valor de la entrada anal.
Desbordamiento positivo	>= 23,516	32.767
Rebase positivo	De 23,515 a 20,0007	De 32511 a 27649
Valor nominal	De 20 a 0	De 27648 a 0
Rebase negativo	De –0,0007 a –3,5185	De -1 a –4864
Desbordamiento negativo	<= -3,5193	-32.768

Lectura 4 a 20 mA:

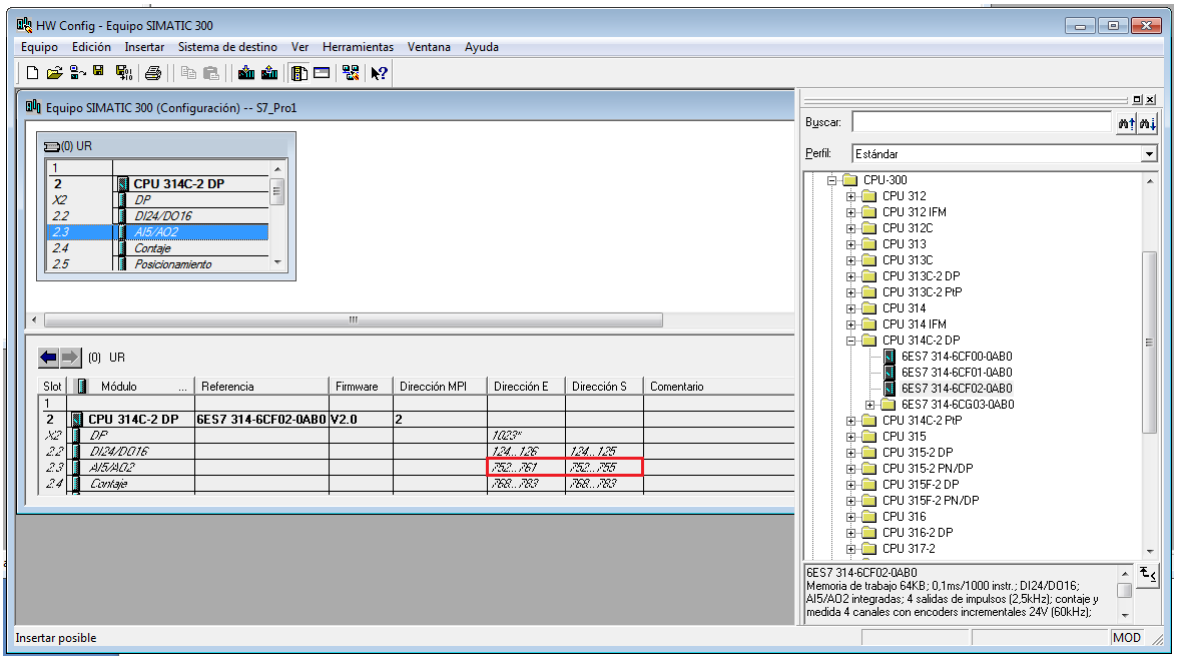
Estado de lectura	Corriente (mA)	Valor de la entrada anal.
Desbordamiento positivo	>= 22,815	32.767
Rebase positivo	De 22,810 a 20,0005	De 32511 a 27649
Valor nominal	De 20 a 4	De 27648 a 0
Rebase negativo	De 3,9995 a 1,1852	De -1 a –4864
Desbordamiento negativo	<= 1,1845	-32.768

EJERCICIO 17. Configuración del PLC para E/S analógicas. Lectura y Escritura en E/S analógicas.

Supongamos que disponemos de un sensor de presión que nos da una tensión entre 0 y 10 V para valores de presión comprendidos entre 0 y 100 mBares. Queremos que dicho sensor nos controle linealmente la apertura de una electroválvula que dispone para su apertura de una entrada de tensión con un rango entre 0 V (cerrada) y 10 V (máxima apertura).

También queremos que cuando se alcance una presión de 80 mBares se active una señal luminosa de aviso.

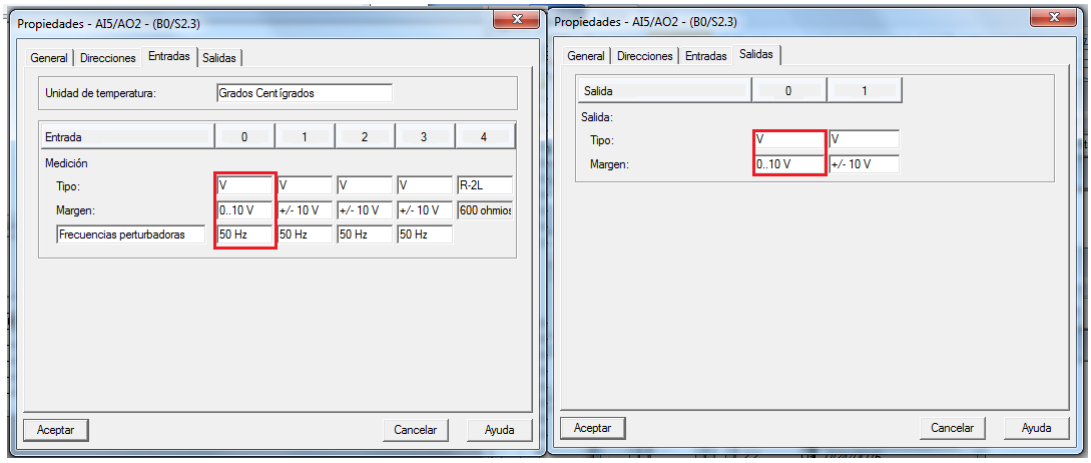
En primer lugar procederemos a la configuración de las E/S analógicas del PLC. En HW Config veremos las direcciones de E/S asignadas para las E/S analógicas. En nuestro caso la direcciones asignadas por defecto han sido: PEW 752 … PEW761 para las 5 entradas analógicas *(AI5),* y PAW 752 … PAW 755 para las 2 salidas analógicas *(A02).* Estas direcciones se pueden cambiar actuando sobre las propiedades de lasa E/S.



Si no cambiamos nada, la dirección de la primera entrada analógica será PEW 752 y la dirección para la primera salida analógica será PAW 752.

A continuación, deberemos asegurarnos que tanto la entrada elegida, así como la salida, estén configuradas para trabajar en los rangos enunciados anterioremente, 0-10 V para la entrada y 0-10V para la salida.

Esto la haremos abriendo las propiedades de E/S picando dos veces sobre el slot 2.3 del bastidor. El resultado deberá ser el siguiente:

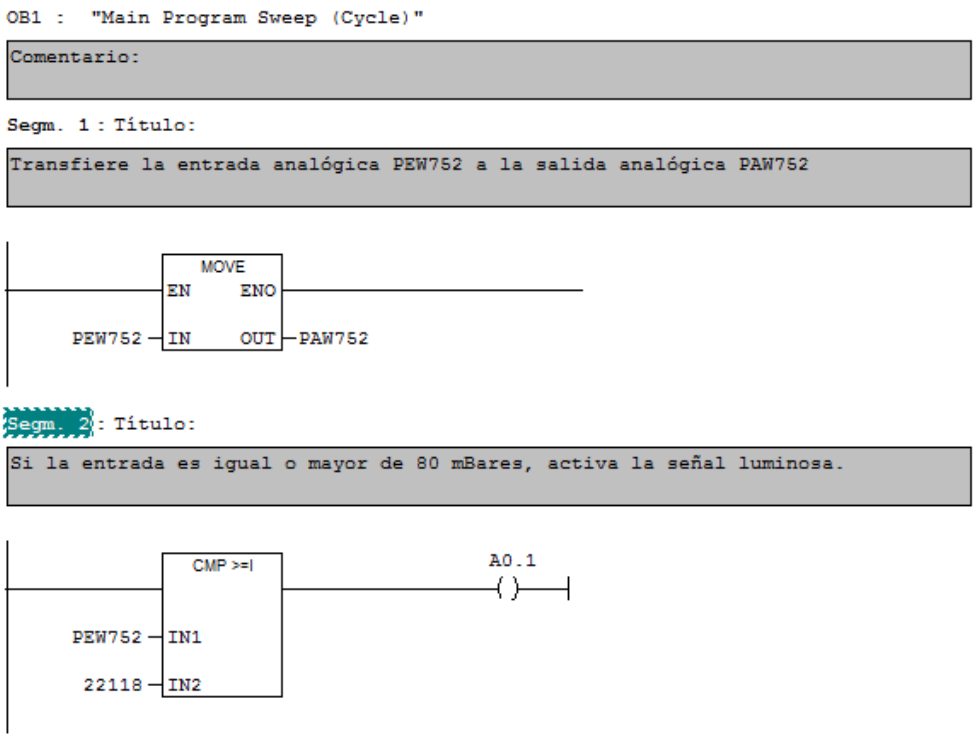


Finalmente, procederemos a realizar la programamción del OB1.

Como los valores de la entrada podrán variar entre 0 y 27648 para los valores de 0 y 10V respectivamente. Haciendo una regla de tres tendrémos el valor de la entrada para 80 mBares.

Valor de entrada para 80 mBares = (80x27648)/100 = 22118,4

SOLUCIÓN EN KOP



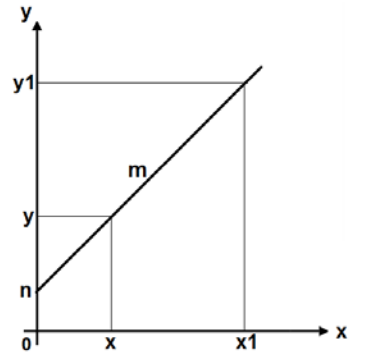
ESCALADO Y DESESCALADO DE VALORES ANALÓGICOS

El escalado de los valores analógicos nos permite trabajar y comparar en las mismas unidades que la variable controlada, en vez de trabajar con los valores numéricos dados por los convertidores A/D de las entradas analógicas. De esta forma podremos directamente operar en nuestro programa con unidades tales como grados, metros, gramos, litros, etc. Por ejemplo:

Entrada (V)	Entrada (V)	Entrada (I)	Entrada (I) con desviación del 20%	Rango nominal de conversión		Temperatura Rango bipolar	Nivel	Presión
+10 V	+5 V	20 mA	20 mA	+27648		300 º C	800 litros	100 mB
0	0	0	4	0 (V-I)	6400 I(20%)	0 ºC	0	0
-10 V				-27648		-300 ºC		

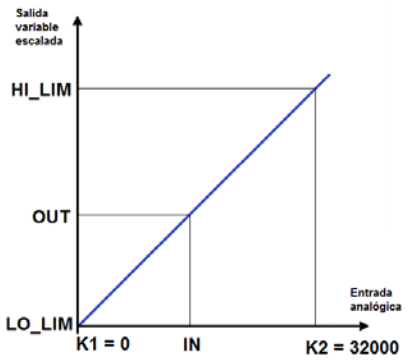
Escalado Unipolar:

Dado que lo que se pretende en el escalado es establecer una relación lineal entre los valores que entrega el convertidor A/D de la entrada analógica (0 a 32000), y los valores de la variable controlada (litros, grados, etc.), utilizaremos la ecuación de la recta para establecer dicha relación. Las siguientes gráficas nos permitirán deducir la expresión matemática que nos exprese el valor de dicha variable.

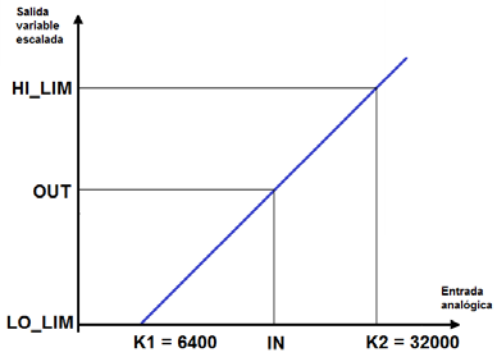


Ecuación general de la recta $y = mx + n$

$$m = \frac{y1 - y}{x1 - x} \qquad y = \frac{y1 - y}{x1 - x} \cdot (X) + n$$



Escalado Unipolar



Escalado unipolar con desviación del 20%

El escalado unipolar solo se desarrolla en el rango de valores positivos (0 a 32000) o negativos (0 a -32000).

Con un escalado unipolar con una desviación del 20% como es el caso de las entradas analógicas con valores 4-20 mA, el límite inferior del rango de valores esta al 20% del valor superior (6400 en lugar de 32000).

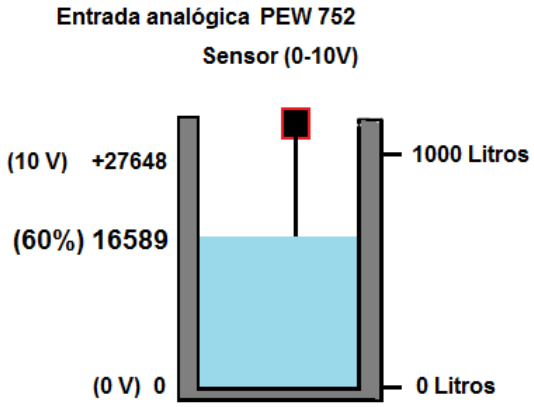
Finalmente, la expresión matemática para el escalado de valores analógicos la obtendremos aplicando la propia ecuación de la recta:

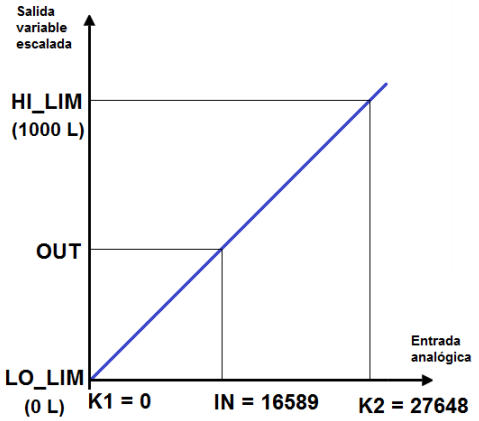
OUT = [(HI_LIM – LO_LIM) / (K2 – K1)] x (IN – K1) + LO_LIM

Parámetro	Descripción
OUT	Valor de la variable escalada (litros, metros, mB, etc.)
HI_LIM	Valor límite superior de la escala (por ejemplo: 1000 Litros, 500 ºC, etc.)
HI_LIM	Valor límite inferior de la escala (por ejemplo: 0 Litros, 20 ºC, etc.)
K2	Valor límite superior de la entrada analógica (32000)
K1	Valor límite inferior de la entrada analógica (0) o 6400 para entradas 4-20 mA
IN	Valor de la entrada analógica (0-32000) --- 0-10V , 0-20 mA, etc.

Veamos un ejemplo:

Disponemos de un depósito de agua con capacidad entre 0-1000 litros. Para la lectura de nivel tenemos un sensor analógico conectado a la entrada PEW752 y calibrado para una tensión unipolar entre 0-10V.



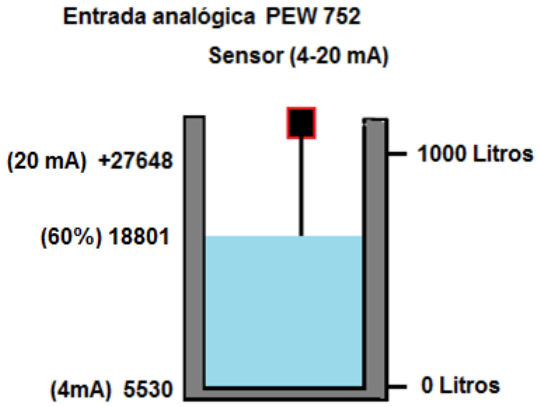


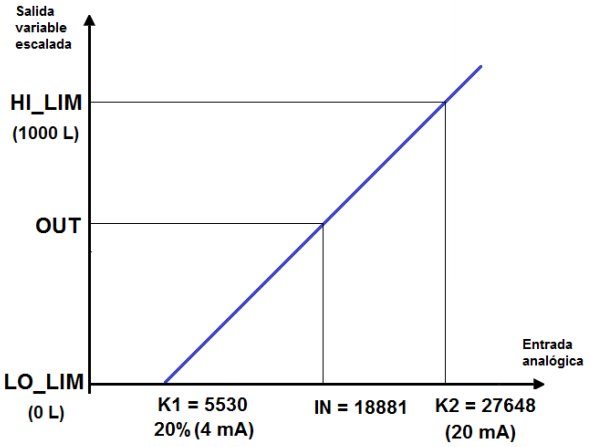
Si el depósito esta al 60 % de capacidad, el valor escalado resultante aplicando la anterior ecuación será:

Valor de IN (60%) = 27648 x 0,6 =**16589**

OUT = [(HI_LIM – LO_LIM) / (K2 – K1)] x (IN – K1) + LO_LIM = [(1000 – 0)/(27648-0)] x (16589 – 0) + 0 = **600 L**

Si en lugar de un sensor de tensión (0-10V) utilizamos un sensor de corriente 4-20 mA, se hace necesario utilizar un offset del 20% del valor analógico en la CPU para compensar.





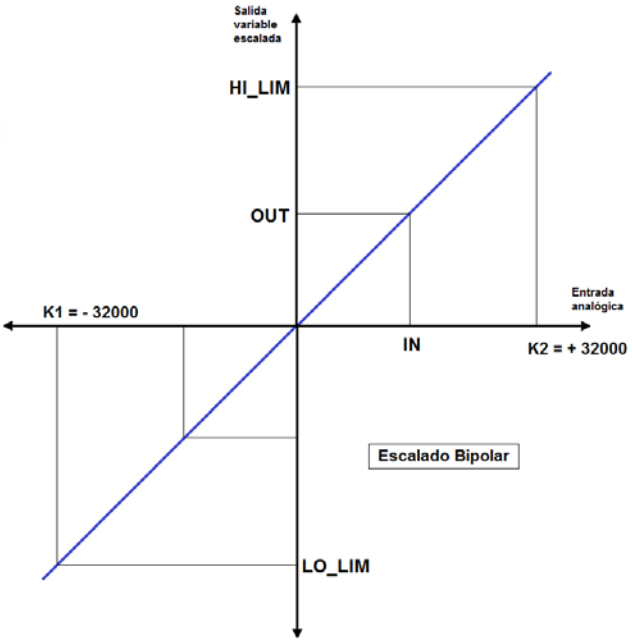
El resultado para el llenado del 60% sería ahora el siguiente:

Valor de IN (60%) = (27648-5530) x 0,6 + 5530 =**18801**

OUT = [(HI_LIM – LO_LIM) / (K2 – K1)] x (IN – K1) + LO_LIM = [(1000 – 0)/(27648-5530)] x (18801 – 5530) + 0 = **600 L**

Escalado Bipolar

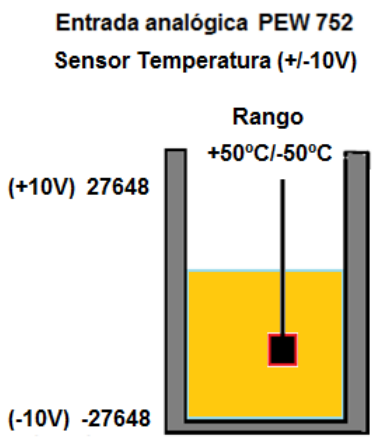
El escalado bipolar se desarrolla en el rango de valores positivo y negativo. La figura muestra un ejemplo de un valor de entrada analógica que va de -32000 a +32000.

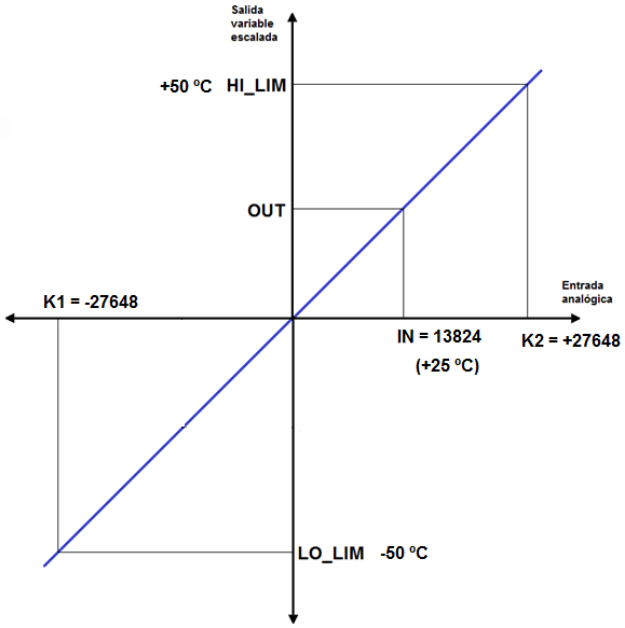


NOTA ACLARATORIA: En las gráficas explicativas hemos utilizado valores de hasta 32000 para indicar el valor máximo de la señal analógica. Sin embargo, en la teoría anteriormente explicada sobre la conversión A/D que se produce en los módulos analógicos, este valor es realmente de 32768 (registro de 16 bit equivalente a 15 bit + signo). Sin embargo, recuerde que en la práctica el límite máximo que se utiliza es de **27648**. Los valores comprendidos entre 27648 y 32768 son utilizados por la CPU para conocer e indicar el posible rebose y desbordamiento que se pueda producir en los niveles de entrada.

Veamos el ejemplo de aplicación con escalado bipolar:

Disponemos de un depósito de líquido y deseamos conocer su temperatura en un rango comprendido entre +50 ºC y -50 ºC. Para ello hemos utilizado una sonda de temperatura en la entrada PEW752 que comprende medidas entre +50 ºC y -50 ºC, y cuyo acondicionador está calibrado para la entrega de una tensión bipolar de +/- 10 V. Deseamos hacer el escalado de la entrada analógica para cualquier valor.





Como ejemplo haremos los cálculos para +25ºC. El resultado sería el siguiente:

Valor de IN (+25ºC) = (27648-0) x 0,25 = **13824**

OUT = [(HI_LIM – LO_LIM) / (K2 – K1)] x (IN – K1) + LO_LIM = [(50 –(-50)/(27648-(-27648)] x (13824 – (-27648) + (-50) = (100/55296) x 41472-50 = **25 ºC**

TIPOS DE DATOS EN STEP 7

En SIMATIC S7 existen diferentes tipos de datos, bajo los cuales pueden representarse diferentes formatos numéricos. A continuación, se muestra una lista completa de los tipos de datos.

Tipo y descripción	Tamaño en Bits	Formato-Opciones	Rango y notación numérica (Valores máximo y mínimo)	Ejemplo
BOOL (Bit)	1	Texto Booleano	TRUE/FALSE	TRUE
BYTE (Byte)	8	Número Hexadecimal	B#16#0 a B#16#FF	B#16#10
WORD (Palabra)	16	Número Binario	2#0 a 2#1111_1111_1111_1111	2#0001_0000_0000_0000
		Número Hexadecimal	W#16#0 a W#16#FFFF	W#16#1000
		BCD	C#0 a C#999	C#998
		Número Decimal sin signo	B#(0,0) a B#(255,255)	B#(10,20)
DWORD (Doble Palabra)	32	Número Binario	2#0 a 2#1111_1111_1111_1111_1111_1111_1111_1111	2#1000_0001_0001_1000_1011_1011_0111_1111
		Número Hexadecimal	DW#16#0000_0000 a DW#16#FFFF_FFFF	DW#16#00A2_1234
		Número Decimal sin signo	B#(0,0,0,0) a B#(255,255,255,255)	B#(1,14,100,120)
INT (Entero)	16	Número Decimal con signo	-32768 a 32767	1
DINT (Int,32 bit)	32	Número Decimal con signo	L#-2147483648 a L#2147483647	L#1
REAL (Número en coma flotante)	32	Número en coma flotante IEEE	Máximo: +/-3.402823e+38 Mínimo: +/-1.175495e-38	1.234567e+13
S5TIME (Tiempo Simatic)	16	Tiempo S7 en pasos de 10 ms	S5T#0H_0M_0S_10MS a S5T#2H_46M_30S_0MS and S5T#0H_0M_0S_0MS	S5T#0H_1M_0S_0MS S5TIME#1H_1M_0S_0MS
TIME (Tiempo IEC)	32	Tiempo IEC en pasos desde 1ms, entero con signo	-T#24D_20H_31M_23S_648MS a T#24D_20H_31M_23S_647MS	T#0D_1H_1M_0S_0MS TIME#0D_1H_1M_0S_0MS
DATE (Fecha IEC)	16	Fecha IEC en pasos de 1 día	D#1990-1-1 a D#2168-12-31	DATE#1994-3-15
TIME_OF_DAY (Fecha y Hora)	32	Tiempo en pasos de 1ms	TOD#0:0:0.0 a TOD#23:59:59.999	TIME_OF_DAY#1:10:3.3
CHAR (Carácter)	8	Caracteres ASCII	´A´, ´B´ etc.	´B´

CALCULOS CON NÚMEROS ENTEROS (INT Y DINT)

Con números enteros, son posibles las operaciones unitarias matemáticas de suma, resta, multiplicación y división. No obstante, no se tienen en cuenta los lugares tras el punto decimal, lo cual genera errores de redondeo con la división.

Operación	Tamaño en Bits	Función
+I	16	Suma el contenido de la palabra baja de los ACCUs 1 y 2 y guarda el resultado en la palabra baja del ACCU 1.
-I	16	Resta el contenido de la palabra baja de los ACCUs 1 y 2 y guarda el resultado el la palabra baja del ACCU 1.
***I**	16	Multiplica el contenido de la palabra baja de los ACCUs 1 y 2 y guarda el resultado (32 Bit) en ACCU 1.
/I	16	Divide el contenido de la palabra baja del ACCU 2 con la palabra baja del ACCU 1. El resultado es almacenado en la palabra baja del ACCU 1. El resto es almacenado en la palabra alta del ACCU 1.
+D	32	Suma los contenidos de los ACCUs 1 y 2 en el ACCU 1.
-D	32	Resta los contenidos de los ACCUs 1 y 2 en el ACCU 1.
***D**	32	Multiplica los contenidos de los ACCUs 1 y 2 en el ACCU 1.
/D	32	Divide el contenido del ACCU 2 con el contenido del ACCU 1 y guarda el resultado en el ACCU 1.
MOD	32	Divide el contenido del ACCU 2 con el contenido del ACCU 1 y guarda el resto en el ACCU 1.

CÁLCULO CON NÚMEROS EN COMA FLOTANTE (REAL)

Con números en coma flotante, se pueden elaborar múltiples operaciones matemáticas. Aquí se consideran las posiciones a la derecha del punto decimal.

Operación	Función
+R	Suma de números en coma flotante (32 Bit, IEEE-FP) contenidos en los ACCUs 1 y 2 y guarda el resultado (32 bits) en el ACCU 1.
-R	Resta de números en coma flotante (32 Bit, IEEE-FP) contenidos en los ACCUs 1 y 2 y guarda el resultado (32 bits) en el ACCU 1.
***R**	Multiplicación de números en coma flotante (32 Bit, IEEE-FP) contenidos en los ACCUs 1 y 2 y guarda el resultado (32 bits) en el ACCU 1.
/R	División de números en coma flotante (32 Bit, IEEE-FP). Se divide el contenido del ACCU 2 por el del ACCU 1. El resultado (32 bits) se guarda en el ACCU 1.
SQRT	Calcula la raíz cuadrada del número en coma flotante (32 Bit, IEEE-FP) contenido en el ACCU 1 y guarda el resultado (32 bits) en el ACCU 1.
SQR	Calcula el cuadrado del número en coma flotante (32 Bit, IEEE-FP) contenido en el ACCU 1 y guarda el resultado (32 bits) en el ACCU 1.
LN	Calcula el logaritmo neperiano del número en coma flotante (32 Bit, IEEE-FP) contenido en el ACCU 1 y guarda el resultado (32 bits) en el ACCU 1.
EXP	Calcula el número e del número en coma flotante (32 Bit, IEEE-FP) contenido en el ACCU 1 y guarda el resultado (32 bits) en el ACCU 1.
SIN	Calcula el seno del número en coma flotante (32 Bit, IEEE-FP) contenido en el ACCU 1 y guarda el resultado (32 bits) en el ACCU 1.
COS	Calcula el coseno del número en coma flotante (32 Bit, IEEE-FP) contenido en el ACCU 1 y guarda el resultado (32 bits) en el ACCU 1.
TAN	Calcula la tangente del número en coma flotante (32 Bit, IEEE-FP) contenido en el ACCU 1 y guarda el resultado (32 bits) en el ACCU 1.
ASIN	Calcula el arcoseno del número en coma flotante (32 Bit, IEEE-FP) contenido en el ACCU 1 y guarda el resultado (32 bits) en el ACCU 1.
ACOS	Calcula el arcocoseno del número en coma flotante (32 Bit, IEEE-FP) contenido en el ACCU 1 y guarda el resultado (32 bits) en el ACCU 1.
ATAN	Calcula el arcotangente del número en coma flotante (32 Bit, IEEE-FP) contenido en el ACCU 1 y guarda el resultado (32 bits) en el ACCU 1.

TIPOS DE DATOS- OPERACIONES DE CONVERSIÓN

Dado que frecuentemente los números no existen para posteriores procesamientos de formatos numéricos, estos números deben de ser ajustados con la ayuda de operaciones de conversión.

Operación	Función
BTI	Conversión BCD a entero (16 Bit). Esta operación convierte un número BCD contenido en el ACCU 1 en un entero (16 Bit). El resultado se deposita en el ACCU1
BTD	Conversión BCD a entero (32 Bit). Esta operación convierte un número BCD contenido en el ACCU 1 en un entero (32 Bit). El resultado se deposita en el ACCU1
ITB	Entero (16 Bit) convertido a BCD. Esta operación convierte un número entero (16 bits) contenido en el ACCU 1 en un número BCD. El resultado se deposita en el ACCU1
ITD	Entero (16 Bit) convertido a entero (32 bits). Esta operación convierte un número entero (16 bits) contenido en el ACCU 1 en un número entero (32 bits). El resultado se deposita en el ACCU1
DTB	Entero (32 Bit) convertido a BCD. Esta operación convierte un número entero (32 bits) contenido en el ACCU 1 en un número BCD. El resultado se deposita en el ACCU1
DTR	Entero (16 Bit) convertido a real (32 bits, IEEE-FP). Esta operación convierte un número entero (16 bits) contenido en el ACCU 1 en un número real (32 bits, IEEE-FP). El resultado se deposita en el ACCU1 (32 Bit, IEEE-FP).
RND	Redondeo a entero. Esta operación redondea el número convertido al entero superior. Cuando la fracción del número convertido sea de 5 o superior, se redondea al entero superior.
RND+	Redondeo al siguiente entero superior. Esta operación redondea el número convertido al siguiente entero superior.
RND-	Redondeo al entero inferior. Esta operación redondea el número convertido al valor de su parte entera.
TRUNC	Redondeo truncado. Esta operación toma sólo la parte entera del número.

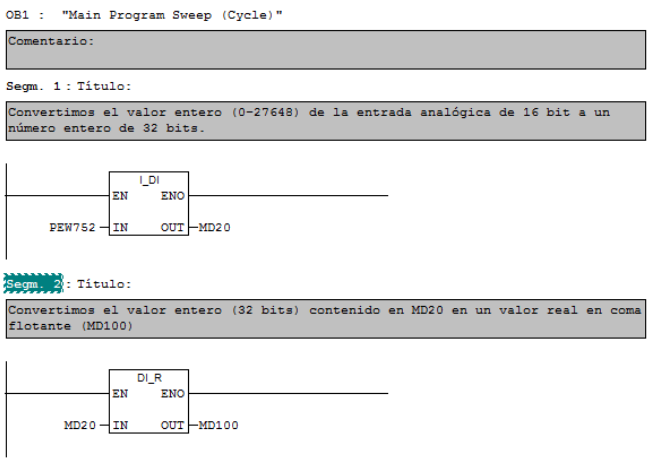
Conversión de número entero (16 bits) a número real en coma flotante (32 bits)

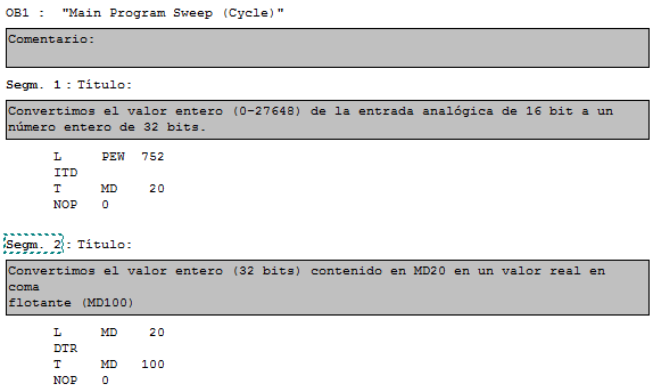
Las entradas analógicas, como por ejemplo PEW752 nos dan valores enteros (0-27648). Si queremos procesar estos valores para hacer cálculos matemáticos o simplemente un escalado de valores, es preciso realizar una conversión de entero a real.

Dado que dicha conversión no es directa, primero haremos una conversión de entero de 16 bits a entero de 32 bit. En AWL lo haremos con la instrucción **ITD** y en KOP con la función **I_DI**.

Posteriormente, convertiremos el valor entero de 32 bits en un número Real en coma flotante de 32 bits. En AWL lo haremos con la instrucción **DTR** y en KOP con la función **DI_R**.

El resultado en AWL y en KOP sería el siguiente:

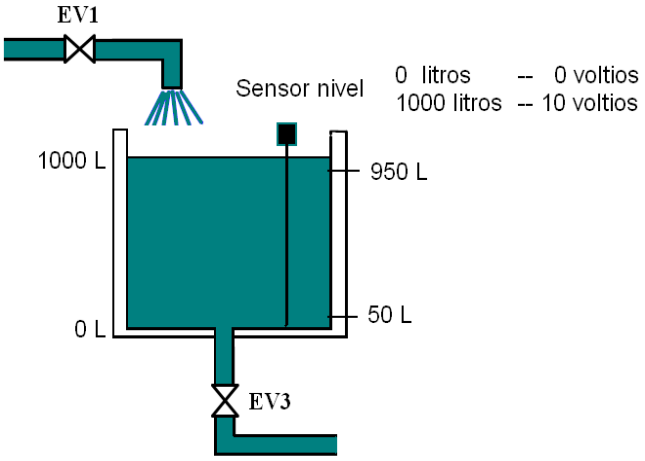




El valor en formato real estaría contenido en la doble palabra MD100

EJERCICIO 18. Control de llenado de un depósito mediante sensor analógico. Escalado mediante la función FC105

Se desea supervisar el llenado de un depósito de manera que la electroválvula EV1 se active para el llenado del depósito cuando a éste le queden tan sólo 50 litros, y que se desactive cuando tenga 950 litros. Para el control, se dispone de un sensor de nivel analógico calibrado para la lectura entre 0 y 1000 litros. La señal que entrega el sensor de nivel está comprendida entre 0 y 10 V para los niveles mínimo y máximo respectivamente.

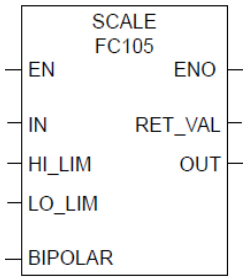


System			Voltage measuring range		Range
	dec.	hex.	1 to 5 V	0 to 10 V	
118.515 %	32767	7FFF	5.741 V	11.852 V	Overflow
117.593 %	32512	7F00			
117.589 %	32511	7EFF	5.704 V	11.759 V	Overrange
	27649	6C01			
100.000 %	27648	6C00	5 V	10 V	
75 %	20736	5100	3.00 V	7.5 V	
0.003617 %	1	1	1 V + 144.7 µV	0 V + 361.7 µV	Nom. range
0 %	0	0	1 V	0 V	
-0.003617 %	-1	FFFF	1 V - 144.7 µV	negative values not possible	Underrange
-17.593 %	-4864	ED00	0.296 V		
	-4865	ECFF			Underflow
≤ -17.596 %	-32768	8000			

Como podemos observar en la tabla, para los valores unipolares entre 0 y 10 V de la entrada analógica, el convertidor A/D interno del PLC entregará valores enteros comprendidos entre 0 y 27648 respectivamente.

El escalado de estos valores lo realizaremos mediante la función “Escalar valores” SCALE (FC105) disponible en Step 7.

La función FC105 toma un valor entero en la entrada IN y lo convierte en un valor real, convirtiéndolo a escala en un rango comprendido entre un límite inferior (LO-LIM) y un límite superior (HI_LIM). El resultado de la función SCALE es un número real que se obtiene en la salida OUT.



La función FC 105 realiza la siguiente operación:

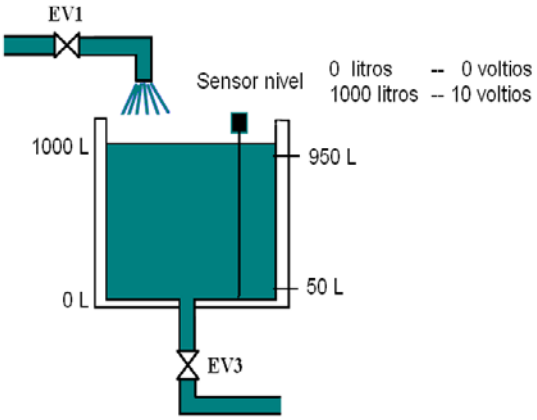
$$OUT = [\ FLOAT\ (IN) - K1/(K2-K1)\ *\ (HI\_LIM - LO\_LIM)] + LO\_LIM$$

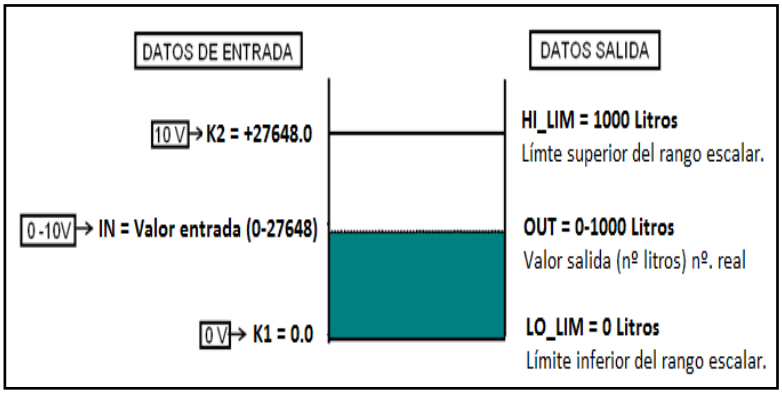
Las constantes K1 y K2 se aplican de forma diferente, dependiendo de si el valor de entrada es BIPOLAR o UNIPOLAR.
BIPOLAR: Se supone que el valor entero de entrada debe estar entre –27648 y 27648, por lo tanto, K1 = –27648.0 y K2 = +27648.0
UNIPOLAR: Se supone que el valor entero de entrada debe estar entre 0 y 27648, por lo tanto, K1 = 0.0 y K2 = +27648.0

Parámetros de la función FC 105

Parámetro	Declaración	Tipo de datos	Area de memoria	Descripción
EN	Entrada	BOOL	E, A, M, D, L	La entrada de habilitación con estado de señal 1 activa el cuadro.
ENO	Salida	BOOL	E, A, M, D, L	La salida de habilitación tiene el estado de señal 1 si la función se ejecuta sin errores.
IN	Entrada	INT	E, A, M, D, L, P, o constante	Valor de entrada a convertir a escala en valor REAL.
HI_LIM	Entrada	REAL	E, A, M, D, L, P, o constante	Límite superior del rango escalar.
LO_LIM	Entrada	REAL	E, A, M, D, L, P, o constante	Límite inferior del rango escalar.
BIPOLAR	Entrada	BOOL	E, A, M, D, L	El estado de señal 1 indica que el valor de entrada es bipolar; con el estado de señal 0 indica que es unipolar.
OUT	Salida	REAL	E, A, M, D, L, P, o constante	Resultado de la conversión a escala.
RET_VAL	Salida	WORD	E, A, M, D, L, P	Da el valor W#16#0000 cuando la función se ejecuta sin errores; si los valores son distintos de W#16#0000, véase la información sobre errores.

En primer lugar, vamos a representar gráficamente los valores necesarios para la función FC 105.



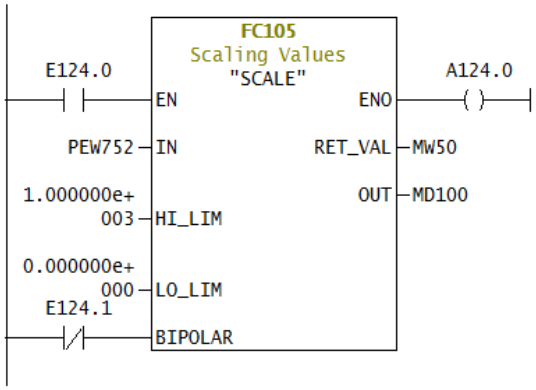


Antes de resolver el ejercicio, vamos a proceder a la simulación con PLCSIM para conocer el comportamiento de la función FC105.

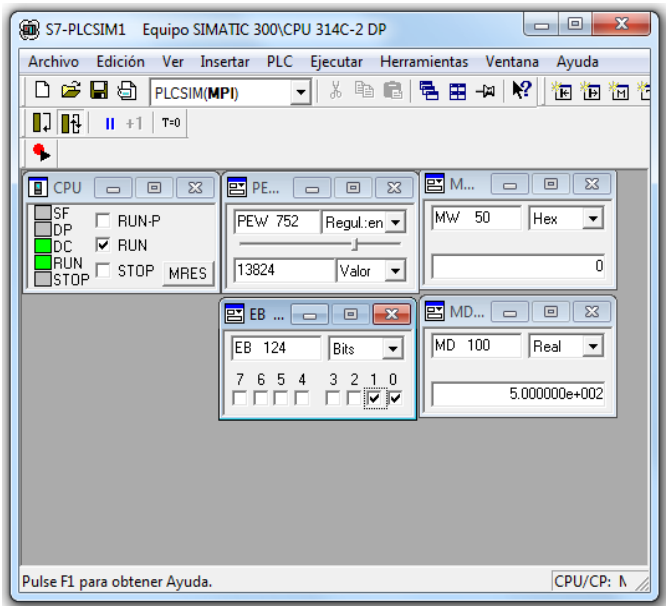
La función se ejecuta cuando el estado de señal de la entrada E 124.0 es 1 (activada). Dado que nuestro sensor es unipolar (0-10V), la entrada BIPOLAR la hacemos 0 mediante E124.1.

En este ejemplo, hemos introducido en la entrada PEW752 el valor entero 13824 correspondiente a un nivel de 5 voltios en el sensor. Dicho valor quedara escalado en la salida OUT entre los valores 0.0 (LO_LIM) y 1000.0 (HI_LIM). El resultado de la operación será ahora un número real que obtenemos a través de la salida OUT y quedará escrito en la doble palabra MD100

Si la función se ejecuta sin errores, los estados de señal de la salida de habilitación (ENO) y de la salida A124.0 se ponen a 1 (activadas), y el valor de respuesta toma el valor W#16#0000.



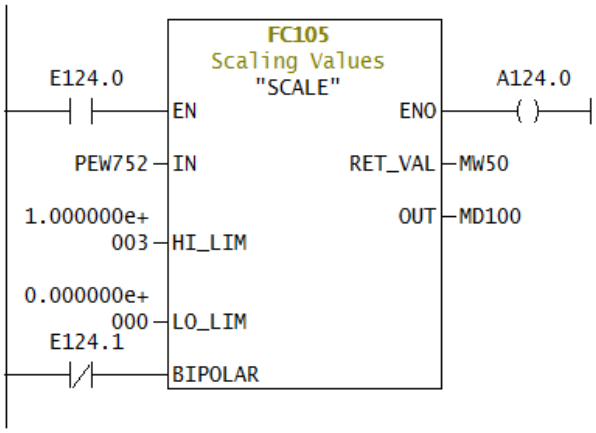
IN → **PEW752 = 13824 (0 a 27648)**
HI_LIM → **1000.0**
LO_LIM → **0.0**
OUT → **MD100 = 500 Litros**



SOLUCIÓN

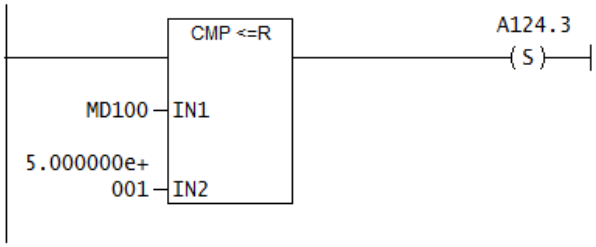
OB1 : "Main Program Sweep (Cycle)"

Segm. 1: Título:



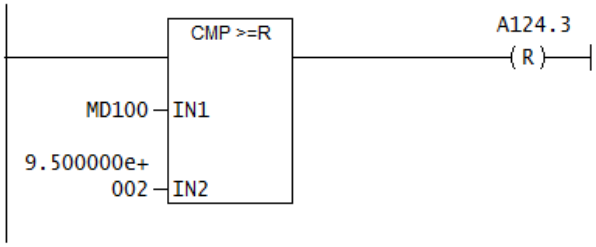
Escalamos los valores contenidos en la entrada PEW752 (0-27648) a valores entre 0 y 1000 Litros.
El resultado es un número real depositado en MD100.

Segm. 2: Título:



Si el nivel del depósito es menor o igual a 50 litros, abrimos la electroválvula EV1 (A124.3)

Segm. 3: Título:



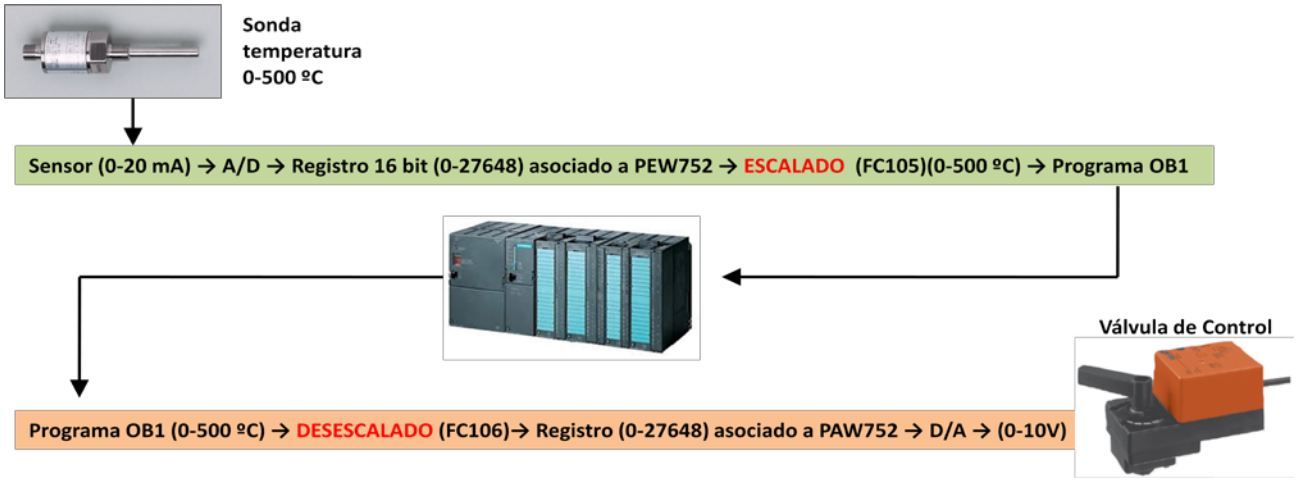
Si el nivel del depósito es mayor o igual a 950 litros, cerramos la electroválvula EV1 (A124.3)

EJERCICIO 19. Regulación de velocidad de un ventilador. Desescalado mediante la función FC106

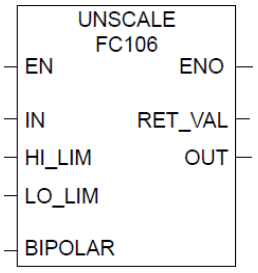
Un ventilador gira entre 0 y 1400 r.p.m en función de los valores 0 - 10 V obtenidos en la salida analógica PAW752. Queremos que al accionar la entrada E124.1 gire a 500 r.p.m., y al accionar la entrada E124.2 gire a 1000 rpm.

Hemos visto anteriormente que cuando trabajamos con entradas analógicas, el escalado nos permite programar directamente con valores de ingeniería (temperatura, presión, caudal, nivel, etc), lo cual nos facilita la programación.

Ahora trataremos el caso contrario, es decir, necesitamos controlar la tensión (V) o intensidad (I) que deberá entregar una salida analógica, según valores de ingeniería (en nuestro rpm). Necesitaremos por tanto, hacer un "desescalado" de valores de ingeniería a valores enteros entre 0-27648, que son los que realmente entiende el convertidor D/A del módulo analógico del PLC. La figura siguiente explica el proceso:



El desescalado lo realizaremos mediante la función "Desescalar valores" (UNSCALE) FC106. Ésta función toma en la entrada IN un valor real que está ajustado a escala en un rango comprendido entre un límite inferior y un límite superior (LO_LIM y HI_LIM), y lo convierte en un valor entero. El resultado se escribe en la salida OUT.



La función FC 106 realiza la siguiente operación:

$$OUT = [\ ((IN - LO\_LIM)/(HI\_LIM - LO\_LIM)) * (K2 - K1)\ ] + K1$$

Las constantes K1 y K2 se aplican de forma diferente, dependiendo de si el valor de entrada es BIPOLAR o UNIPOLAR.
BIPOLAR: Se supone que el valor entero de entrada debe estar entre –27648 y 27648, por lo tanto, K1 = –27648.0 y K2 = +27648.0
UNIPOLAR: Se supone que el valor entero de entrada debe estar entre 0 y 27648, por lo tanto, K1 = 0.0 y K2 = +27648.0

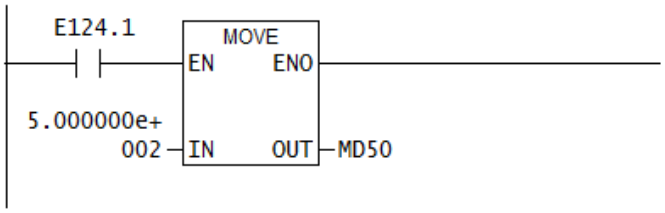
Parámetros de la función FC 105

Parámetro	Declaración	Tipo de datos	Area de memoria	Descripción
EN	Entrada	BOOL	E, A, M, D, L	La entrada de habilitación con estado de señal 1 activa el cuadro.
ENO	Salida	BOOL	E, A, M, D, L	La salida de habilitación tiene el estado de señal 1 si la función se ejecuta sin errores.
IN	Entrada	REAL	E, A, M, D, L, P, o constante	Valor de entrada a desescalar convirtiéndolo en un valor entero.
HI_LIM	Entrada	REAL	E, A, M, D, L, P, o constante	Límite superior del rango escalar.
LO_LIM	Entrada	REAL	E, A, M, D, L, P, o constante	Límite inferior del rango escalar.
BIPOLAR	Entrada	BOOL	E, A, M, D, L	El estado de señal 1 indica que el valor de entrada es bipolar; con el estado de señal 0 indica que es unipolar.
OUT	Salida	INT	E, A, M, D, L, P	Resultado del desescalado.
RET_VAL	Salida	WORD	E, A, M, D, L, P	Da el valor W#16#0000 cuando la función se ejecuta sin errores; si los valores son distintos de W#16#0000, véase la información sobre errores.

SOLUCIÓN

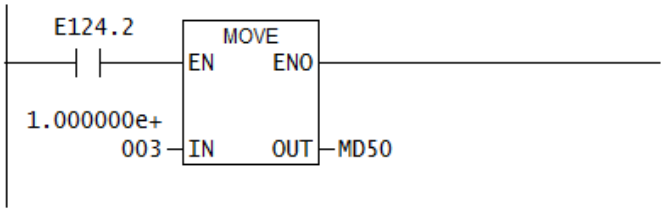
OB1 : "Main Program Sweep (Cycle)"

Segm. 1: Título:



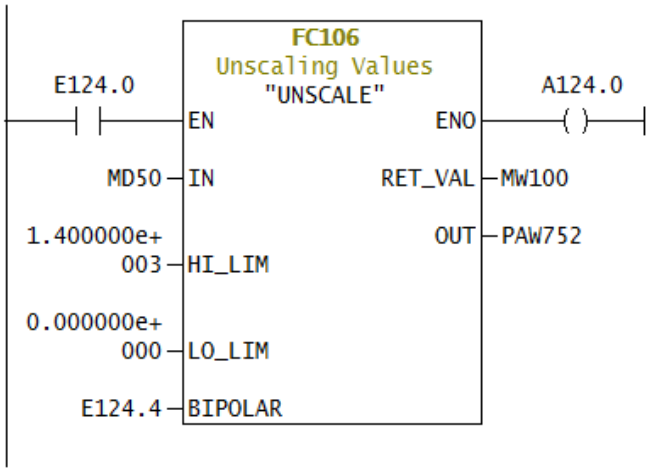
Cuando activemos la entrada E124.1 transferimos a la doble palabra MD50 (real) el dato 500 rpm.

Segm. 2: Título:



Cuando activemos la entrada E124.2 transferimos a la doble palabra MD50 (real) el dato 1000 rpm.

Segm. 3: Título:



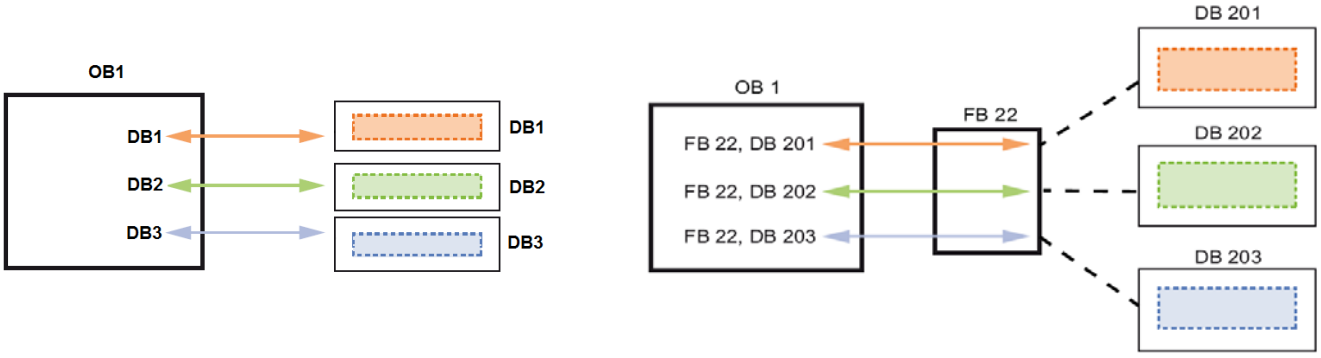
Al activar E124.0, el valor contenido en MD50 es desescalado entre 0 y 1400 rpm. El valor resultante lo obtenemos en la salida analógica PAW752.

Si la salida analógica PAW752 la hemos configurado previamente para entregar tensiones entre 0 a 10 V. La tensión que obtendremos para 1000 r.p.m será de 7,15 V y de 3,58 V para 500 r.p.m.

TRABAJAR CON BLOQUES DE DATOS (DB)

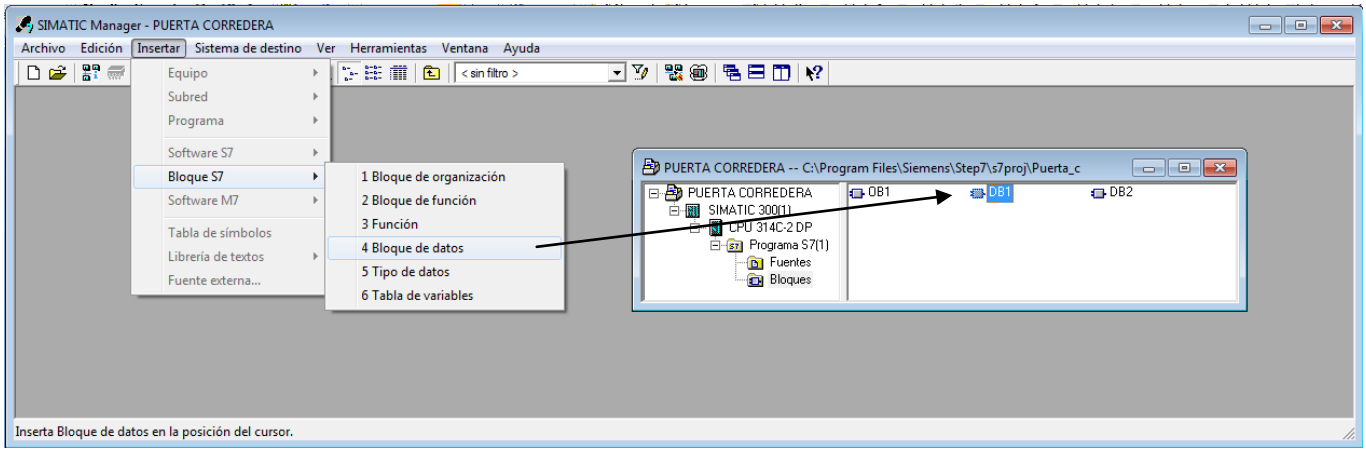
Los bloques de datos (DB) son áreas de datos del programa de usuario (OB1) en las que se almacenan datos en forma de una tabla organizada. Estos datos pueden ser utilizados directamente desde el programa de usuario (OB1) o desde bloques de función (FB).

En un bloque de datos (DB) no se programa nada. Se utilizan para almacenar valores iniciales y leer o escribir datos actuales de determinadas variables, que posteriormente pueden ser usadas por el programa de usuario.



Para guardar un dato, tenemos que poner nombre a la variable, definir el formato en el que lo queremos, y el valor inicial. El valor inicial siempre es el mismo, su propio nombre ya lo indica. Cuando este valor cambie, se almacenará en otra columna que es el valor actual.

Los bloques de datos se introducen en Step 7 mediante la secuencia indicada en la figura:



Para poderla ver el contenido, tenemos que ir al menú Ver > datos, o abrir directamente el bloque en la carpeta del proyecto.

A continuación, definiremos el nombre de la variable, el tipo y el valor inicial.

Dirección	Nombre	Tipo	Valor inicial	Comentario
0.0		STRUCT		
+0.0	TEMPERATURA	REAL	4.500000e+001	Temperatura tanque
+4.0	VALOR_1	INT	100	Variable provisional
+6.0	CONTADOR_1	INT	5	valor contador 1
=8.0		END_STRUCT		

Una vez completado el bloque, procederemos a guardarlo y a cargarlo en el autómata junto al OB1.

Para acceder a los datos de un DB tenemos dos posibilidades. Podemos abrir primero un DB y luego acceder directamente a la palabra de datos, o podemos decir cada vez a que DB y a que palabra de datos queremos acceder.

```
AUF   DB 1                L   DB1.DBW 0
L     DBW 0               T   MW 10
T     MW 10                   BE
BE
```

Para acceder a un dato, le llamamos DB.... Puede ser DBB si es un byte, DBW si es una palabra, DBD si es una doble palabra o DBX 0.0 si es un bit.

También podemos dar un nombre al DB y acceder a los datos por su nombre.

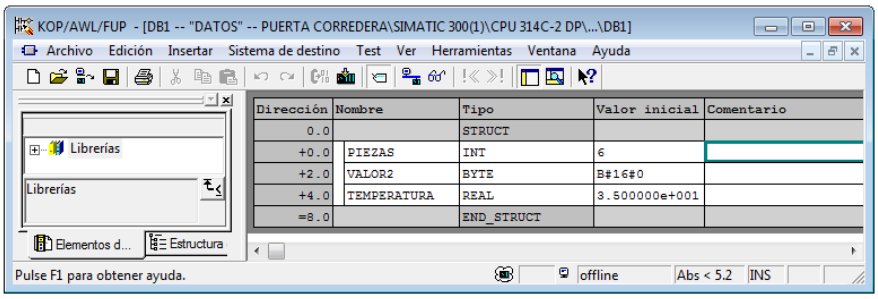
Por ejemplo en el DB que hemos creado antes podríamos acceder a los datos del siguiente modo:

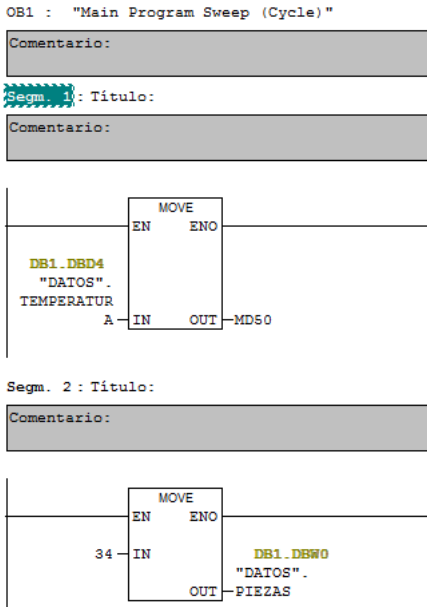
L DATOS.VALOR_1 DATOS es el nombre del DB y VALOR_1 es el nombre del dato.

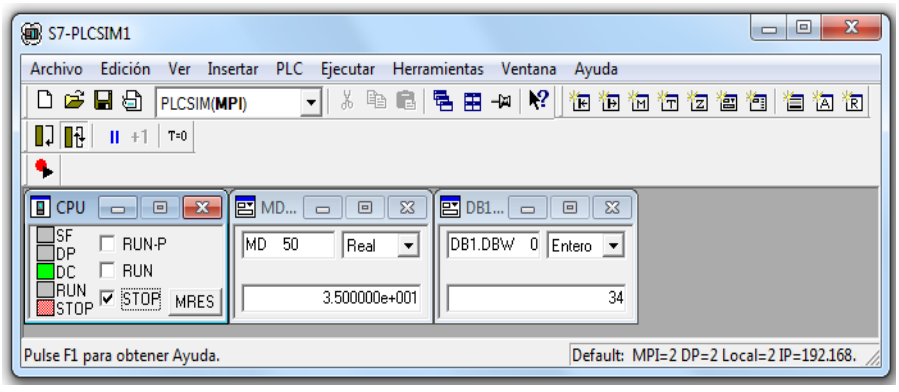
Ejemplo:

Leemos el valor “Temperatura” y la llevamos a MD50. Escribimos el valor 34 en la variable “PIEZAS”. Observe que aunque el valor inicial de “piezas” era 6, al ejecutar el programa la variable ha pasado a valer 34.

Hemos utilizado el simulador PLCSIM para comprobar el funcionamiento de nuestro programa.







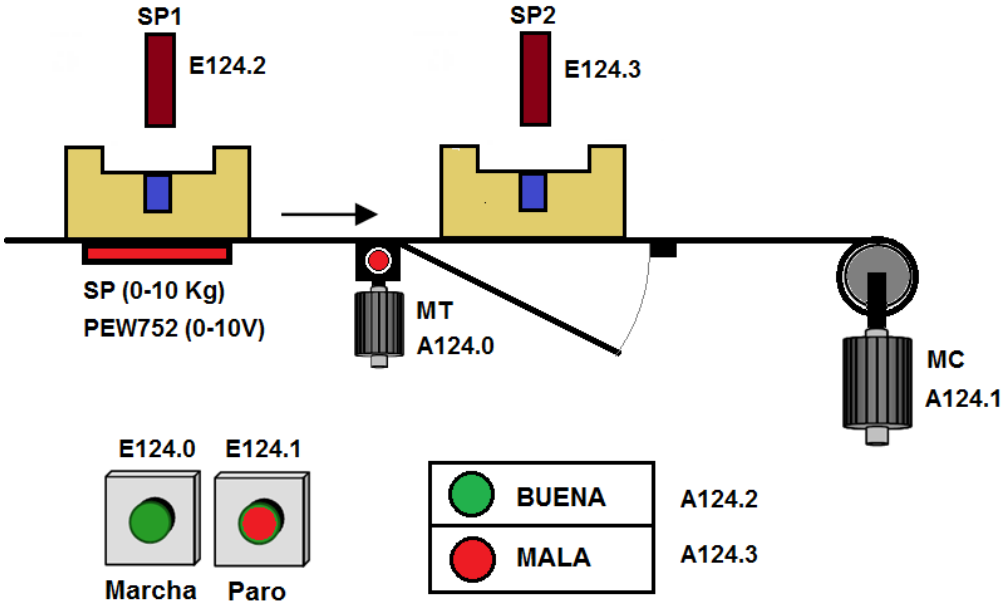
EJERCICIO 20. CONTROL DE PESADO. Creación y manejo de DB´s.

Tenemos una cinta transportadora por donde se desplazan piezas. Queremos hacer la selección de éstas según su peso, de forma que las piezas que se salgan de un margen de peso estipulado tanto por arriba como por debajo se desechen a través de una trampilla. Tenemos unas luces indicadoras de pieza buena o mala. Queremos que mientras está la pieza debajo del sensor de peso se encienda la luz correspondiente.

La cinta se ha de detener para el pesado de las piezas durante 5 segundos; pasado ese tiempo continuará su recorrido. La trampilla se abrirá solo durante 3 segundos cuando la pieza defectuosa se sitúe sobre el sensor de posición SP2.

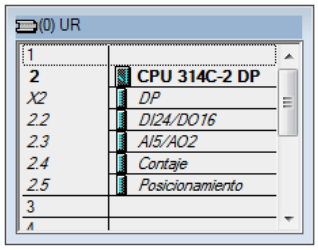
Se consideran piezas buenas aquellas que tengan un peso entre 4 y 5 Kg.

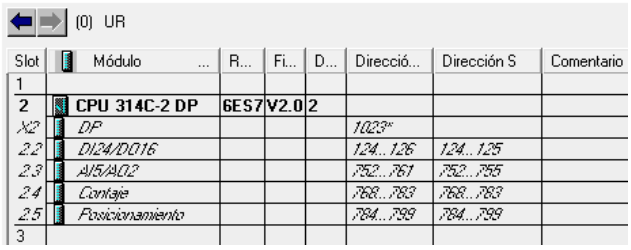
El pesado lo vamos a realizar mediante una sonda analógica calibrada para pesos entre 0 y 10 Kg. Para este rango de medida, la sonda nos dará valores de tensión entre 0 y 10 Voltios.

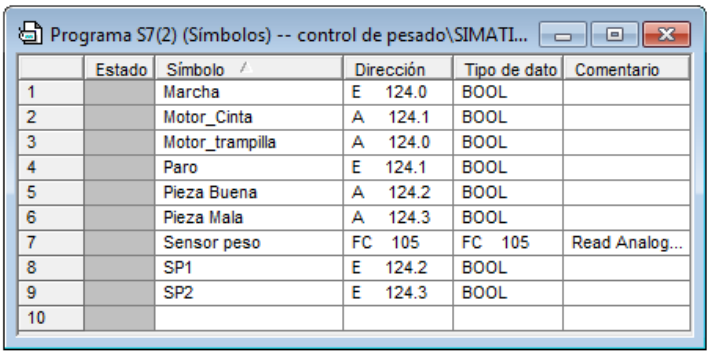


SOLUCIÓN:

Configuramos en primer lugar la CPU, en nuestro caso una S7-314C-2DP, y definimos nuestra tabla de símbolos.

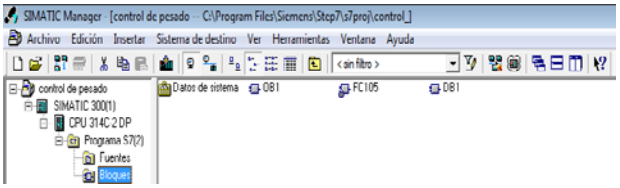


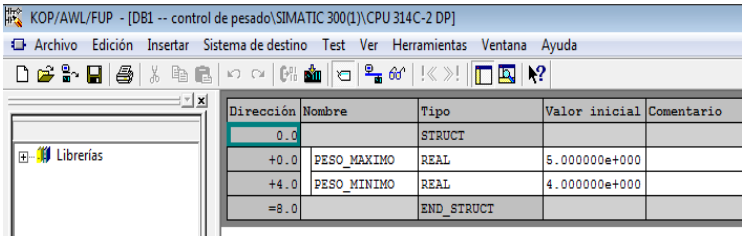




	Estado	Símbolo	Dirección	Tipo de dato	Comentario
1		Marcha	E 124.0	BOOL	
2		Motor_Cinta	A 124.1	BOOL	
3		Motor_trampilla	A 124.0	BOOL	
4		Paro	E 124.1	BOOL	
5		Pieza Buena	A 124.2	BOOL	
6		Pieza Mala	A 124.3	BOOL	
7		Sensor peso	FC 105	FC 105	Read Analog...
8		SP1	E 124.2	BOOL	
9		SP2	E 124.3	BOOL	
10					

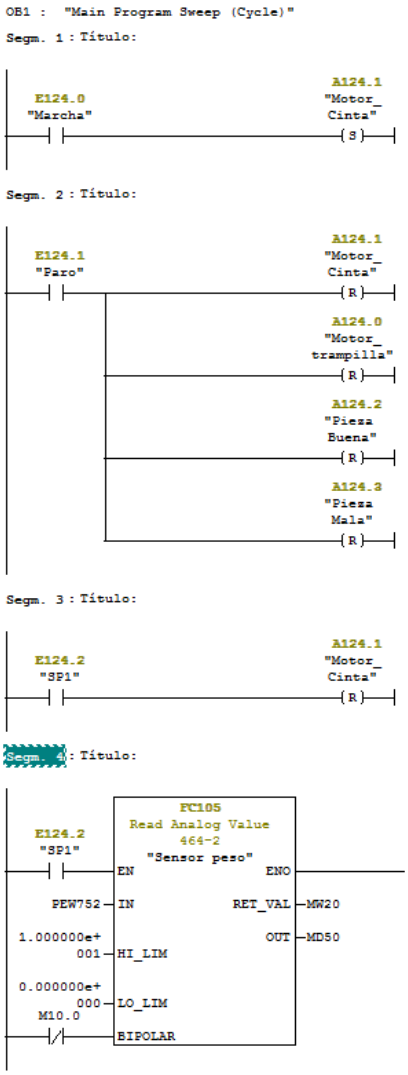
Seguidamente procederemos a insertar un bloque de datos DB1 que utilizaremos para definir los pesos límite inferior y superior. Una vez configurado queda como indica la siguiente figura:

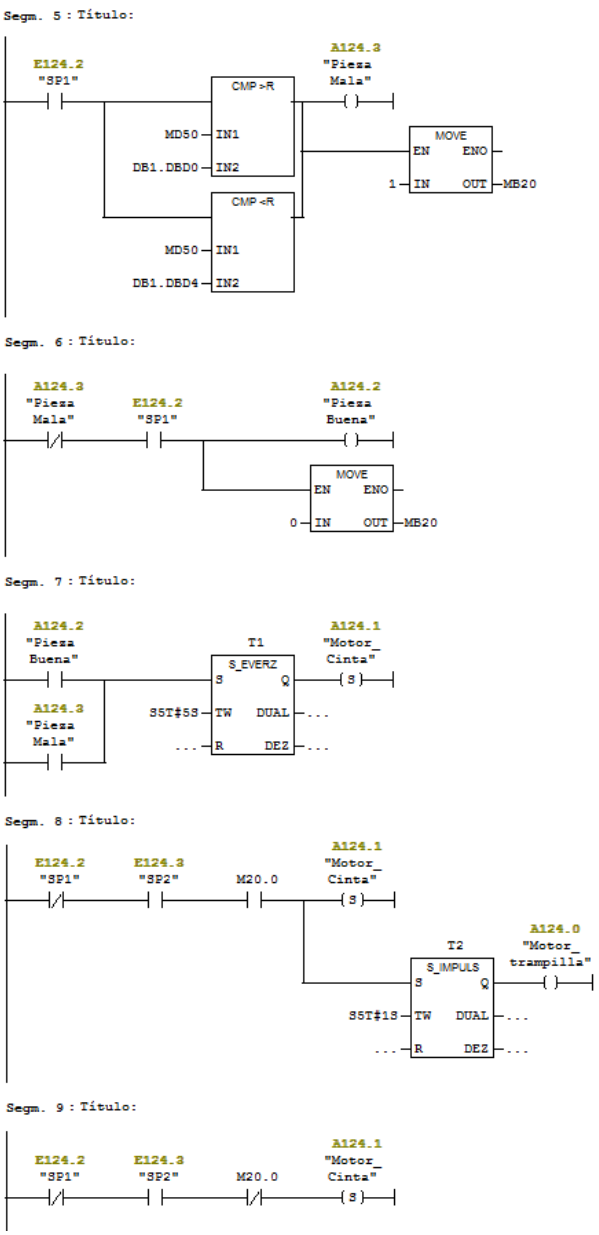




Dirección	Nombre	Tipo	Valor inicial	Comentario
0.0		STRUCT		
+0.0	PESO_MAXIMO	REAL	5.000000e+000	
+4.0	PESO_MINIMO	REAL	4.000000e+000	
=8.0		END_STRUCT		

Programa en KOP





PROGRAMACIÓN ESTRUCTURADA. Trabajar con Funciones (FC´s)

La programación estructurada se refiere al control de ejecución de un programa. La regla general es que las instrucciones se ejecuten sucesivamente una tras otra, pero diversas partes del programa se ejecutan o no dependiendo de que se cumpla alguna condición. Además, hay instrucciones (los bucles) que deben ejecutarse varias veces, ya sea en número fijo o hasta que se cumpla una condición determinada.

Algunos lenguajes de programación más antiguos como Fortran o Basic, se apoyaban en una sola instrucción para modificar la secuencia de ejecución de las instrucciones mediante una transferencia incondicional de su control (con la instrucción goto, del inglés "go to", que significa "ir a"). Pero estas transferencias arbitrarias del control de ejecución hacen los programas muy poco legibles y difíciles de comprender. A finales de los años sesenta, surgió una nueva forma de programar que reduce a la mínima expresión el uso de la instrucción "***goto***" y la sustituye por otras más comprensibles.

Esta forma de programar se basa en un famoso teorema, desarrollado por Edsger Dijkstra, que demuestra que todo programa puede escribirse utilizando únicamente las tres estructuras básicas de control siguientes:

- **Secuencia:** el bloque secuencial de instrucciones, instrucciones ejecutadas sucesivamente, una detrás de otra.

- **Selección:** la instrucción condicional con doble alternativa, de la forma " **if** *condición* **then** *instrucción-1* **Else** *instrucción-2* ".

- **Iteración:** el bucle condicional "**while** *condición* **do** *instrucción*", que ejecuta la instrucción repetidamente mientras la condición se cumpla.

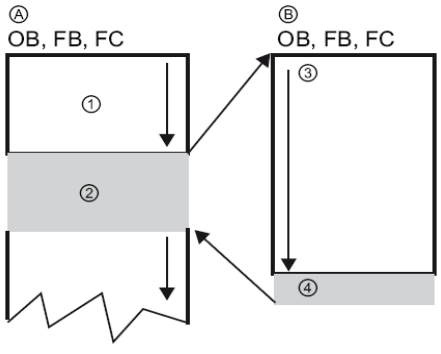
Los programas que utilizan estas estructuras de control básicas, se llaman ***programas estructurados***.

Ésta es la noción clásica de lo que se entiende por programación estructurada (llamada también programación sin goto) que hasta la aparición de la programación orientada a objetos se convirtió en la forma de programar más extendida. Una característica importante en un programa estructurado es que puede ser leído en secuencia, desde el comienzo hasta el final sin perder la continuidad de la tarea que cumple el programa, lo contrario de lo que ocurre con otros estilos de programación.

Cuando en la actualidad se habla de programación estructurada, nos solemos referir a la división de un programa en partes más manejables (usualmente denominadas segmentos, módulos o subrutinas). Así, la visión moderna de un programa estructurado es un compuesto de segmentos, los cuales puedan estar constituidos por unas pocas instrucciones. Cada segmento, módulo o subrutina tiene solamente una entrada y una salida, asumiendo que no poseen bucles infinitos y no tienen instrucciones que jamás se ejecuten.

En el Step 7, las posibilidades de estructuración del programa pueden ser diversas, tal como indica la siguiente figura:

A	Bloque que llama
B	Bloque llamado (o que interrumpe)
①	Ejecución del programa
②	Instrucción o evento que inicia la ejecución de otro bloque
③	Ejecución del programa
④	Fin del bloque (regresa al bloque que llama)



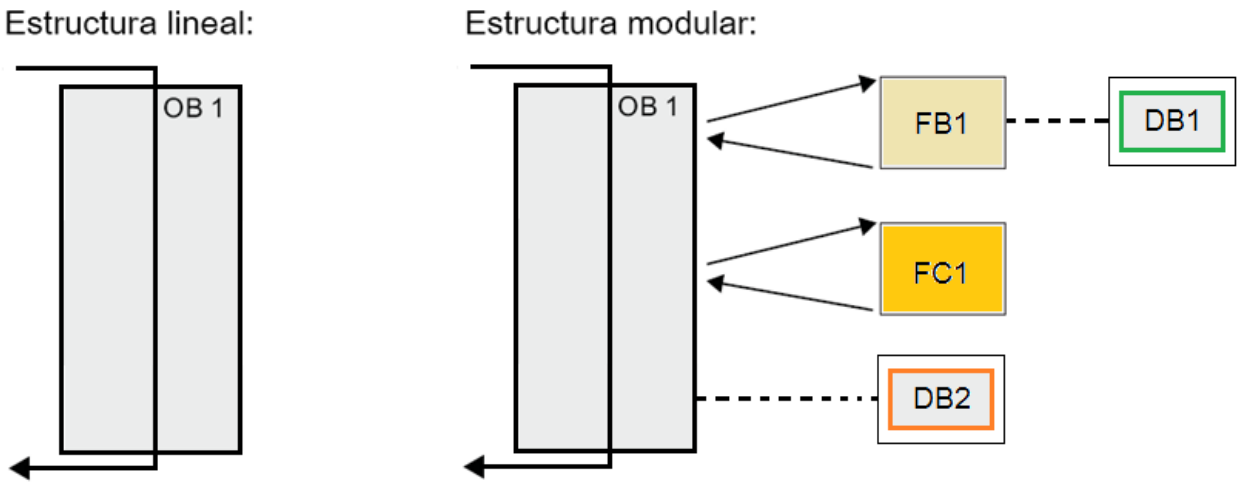
Dependiendo de los requerimientos del proceso, el programa puede ser estructurado en bloques diferentes, donde podemos almacenar el programa de usuario y demás datos relacionados.

- **Bloques de Organización (OB‘s).** Estos bloques constituyen la interface entre el sistema operativo del PLC y el programa de usuario. El programa completo puede almacenarse en un OB, que es ejecutado cíclicamente por el sistema operativo (programa lineal) o puede dividirse y almacenarse en distintos bloques (programa estructurado). El programa principal es siempre el bloque de organización llamado OB1.

- **Funciones (FC´s). Una función (FC) es como una subrutina**. Una FC es un bloque lógico que generalmente realiza una operación específica con una serie de valores de entrada. La FC almacena los resultados de esta operación en posiciones de memoria. Éstas se utilizan para realizar operaciones estándar y reutilizables, p. ej. en cálculos matemáticos, operaciones lógicas, etc. Los FC´s se puede llamar varias veces en diferentes puntos de un programa. Esto facilita la programación de tareas que se repiten con frecuencia. Las FC no tienen bloques de datos DB asociados. Step 7 dispone de una extensa librería de bloques FC ya programados. Un ejemplo claro son las ya conocidas FC105 y FC106 que hemos utilizado anteriormente para el escalado y desescalado de E/S analógicas.

- **Bloques de función (FB). Un bloque de función (FB) es como una subrutina con memoria**. El FB almacena los parámetros de E/S en una memoria variable integrada en un bloque de datos (DB), o en un DB "instancia". Los bloques de función están concebidos para tareas muy repetitivas o funcionalidades complejas, como tareas de control en lazo cerrado.

- **Bloques de datos (DB).** Los bloques de datos (DB) son áreas de datos del programa de usuario en las que los datos son distribuidos de forma estructurada.

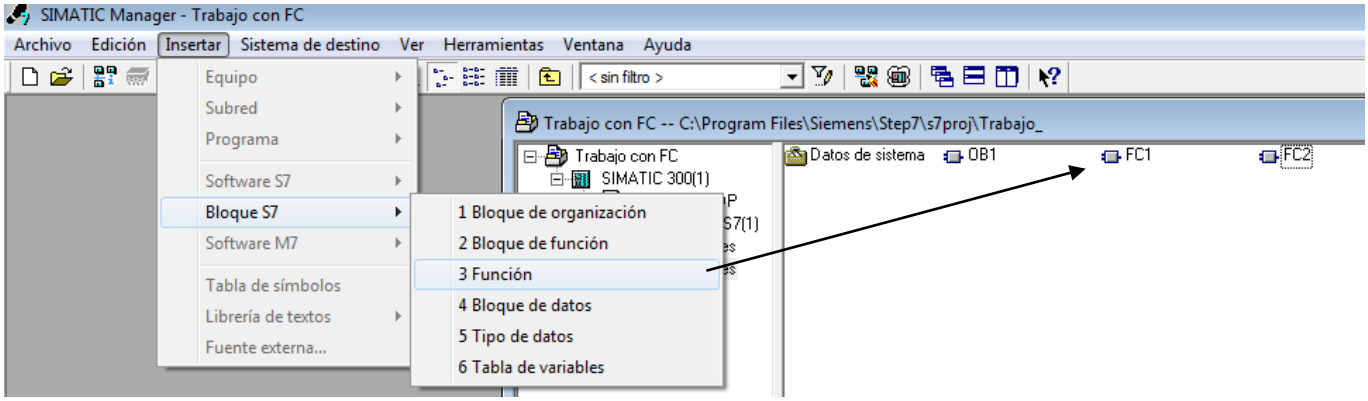


EJERCICIO 21. Programación estructurada con FC´s

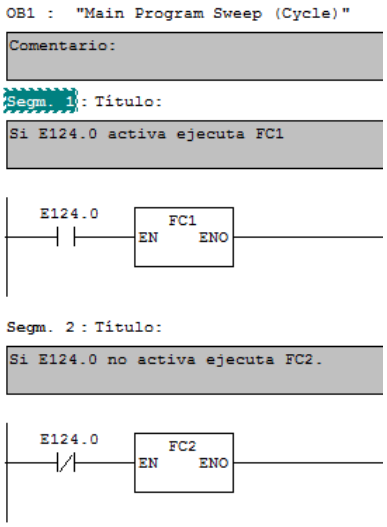
Queremos que si la entrada E 124.0 está activa funcione una parte del programa contenido en FC1, y que si no está activa funcione otra parte del programa contenido en FC2. Desde el OB1 indicaremos cuando tiene que acceder a uno y a otro.

Programa en FC1: Cuando pulsemos E124.1 activar la salida A124.0 durante 5 segundos.
Programa en FC2: Cuando pulsemos E124.1 activar la salida A124.0 durante 10 segundos.

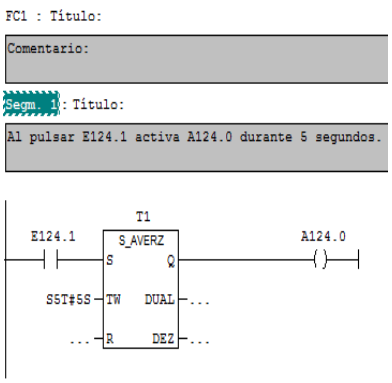
Para crear las 2 FC´s, lo hacemos del mismo modo que hicimos para crear un nuevo DB. Una vez tengamos creadas las dos FC´s ya podemos entrar en ellas y escribir su programa.



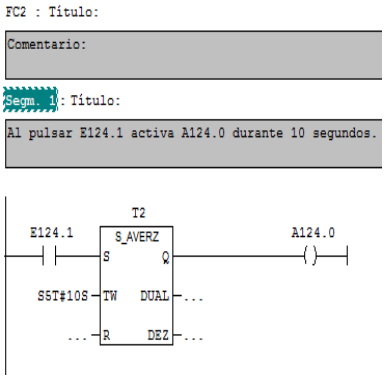
PROGRAMA OB1



PROGRAMA FC1

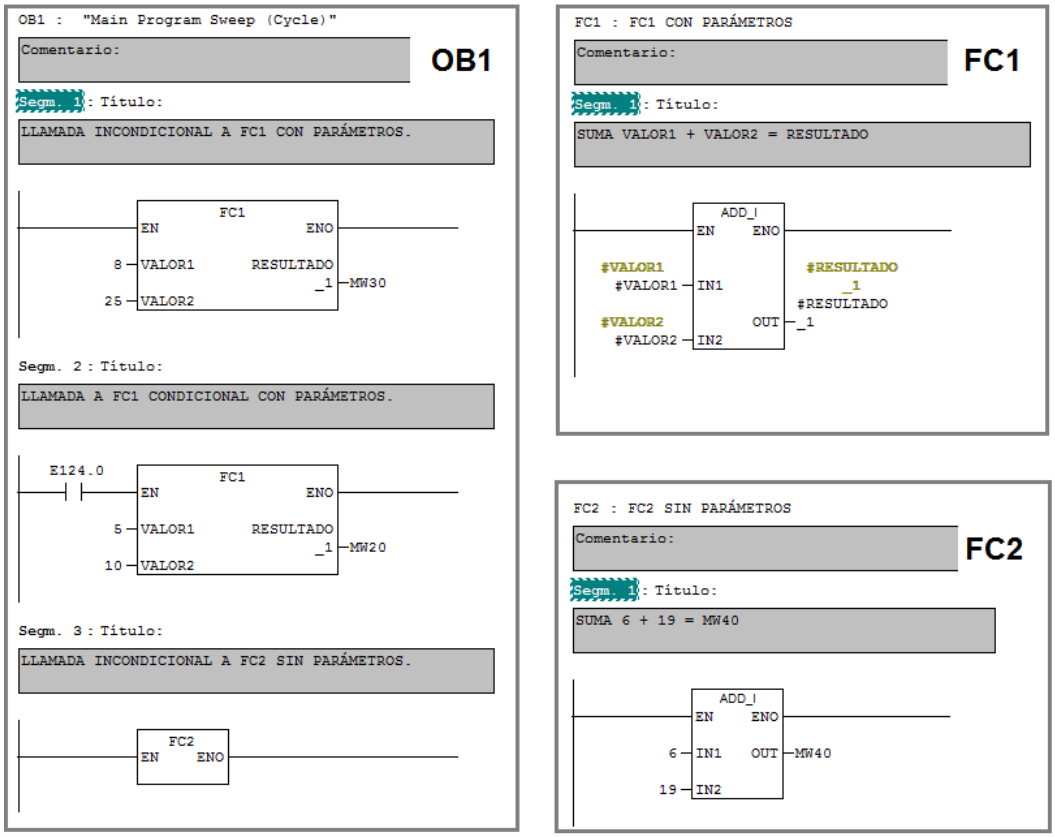


PROGRAMA FC2



LLAMADAS A LAS FC´s CON Y SIN PARÁMETROS

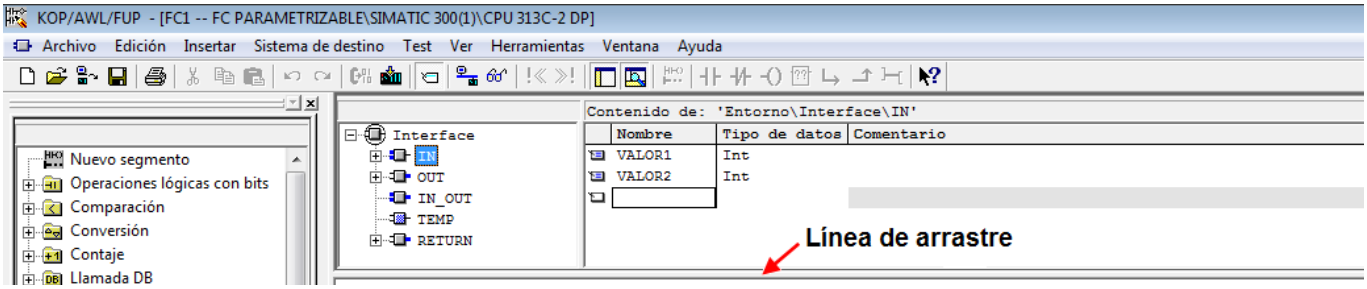
Cuando programamos con FC´s tenemos que llamarla desde el OB1 para que se ejecute. Esta llamada puede ser de tipo incondicional o condicional. Además, podemos programar las FC´s con o sin parámetros. La siguiente figura muestra la diferencia.



En el segmento 3 del OB1, hacemos una llamada incondicional a la función FC2 sin parámetros. Ésta siempre sumará lo mismo (6+19), y el resultado siempre será 25. El resultado lo tendremos presente en MW40.

En los segmentos 1 y 2 estamos haciendo una llamada incondicional y condicional respectivamente a la función FC1. Sin embargo, observe que esta función es parametrizable, dado que desde el OB1 se nos pedirá los valores a sumar. Estos valores pueden ser ahora diferentes. En el primer segmento, la FC1 nos suma 8+25 y deposita el resultado en MW30, en el segundo segmento sumamos 5+10 y depositamos el resultado en MW20.

Para hacer una función FC con parámetros, tan sólo tenemos que rellenar la tabla de variables superior que nos aparece en la pantalla al abrir la FC.



NOTA: si la tabla de variables está oculta pulse y arrastre con el ratón sobre la línea indicada.

Veremos que tenemos varios campos para definir:

La de IN sirve para definir los datos de entrada a la función.

La línea de OUT sirve para definir las salidas de la función.

La línea de IN/OUT sirve para definir los valores que dentro de la función pueden ser entradas y salidas.

La línea de TEMP sirve para definir valores intermedios. Son valores que no son entradas ni salidas de la función. Son resultados intermedios que no nos interesa ver desde fuera de la función.

Tenemos que definir también el nombre de la variable y el tipo de datos. Observe que en nuestro caso hemos introducido “VALOR1” y “VALOR2” como entradas INPUT. Como salida OUT hemos definido “RESULTADO”. La almohadilla no la escribimos nosotros. Esto nos indica que es una variable local. Sólo la podemos utilizar con su nombre dentro de la FC 1 que es donde la hemos definido.

De este modo tenemos una función que suma números enteros pero no le hemos dicho qué números. Cada vez que la utilicemos serán números distintos.

EJERCICIO 22. Control de producción de un gallinero. (Programación estructurada, bloques de datos, flancos.)

Para llevar el control de producción del gallinero, tenemos que optimizar las horas de luz y las horas de oscuridad que va a tener el gallinero con objeto de obtener la mayor cantidad de huevos posible.

Llevaremos el control de los huevos que ponen las gallinas mediante dos contadores; uno para llevar la cuenta de los huevos puestos hoy, y otro con los huevos que pusieron el día anterior.

El funcionamiento deberá ser el siguiente:

Si hoy han puesto más huevos que ayer, supondremos que las gallinas están contentas. Entonces lo que haremos será disminuir en 8 los minutos de luz y los minutos de oscuridad. De manera que les hacemos el día más corto.

Si hoy han puesto menos huevos que ayer, supondremos que las gallinas están tristes. Entonces lo que haremos será aumentar en 5 los minutos de luz y los minutos de oscuridad. De manera que les hacemos el día más largo.

Si hoy han puesto los mismos huevos que ayer, supondremos que las gallinas están indiferentes. Entonces lo que haremos será disminuir en 1 los minutos de luz y los minutos de oscuridad. Iremos haciendo el día más corto poco a poco hasta que pongan menos huevos.

SOLUCIÓN:

Haremos una programación estructurada por bloques.

En primer lugar haremos dos bloques de datos (DB´s) donde guardaremos los datos que luego vamos a utilizar.

DB1 – Se llamará “PUESTAS”, y en el guardaremos los huevos puestos hoy y los que pusieron ayer.

Dirección	Nombre	Tipo	Valor inicial	Comentario
0.0		STRUCT		
+0.0	HUEVOS_HOY	INT	0	Puestas de huevos Hoy
+2.0	HUEVOS_AYER	INT	0	Puestas de huevos Ayer
=4.0		END_STRUCT		

Haremos un DB2 que se llame “TIEMPOS”. Allí meteremos los minutos iniciales de luz y de oscuridad que va a tener el gallinero, y además la cantidad de minutos de luz y de oscuridad que tendríamos que sumar o restar en su caso.

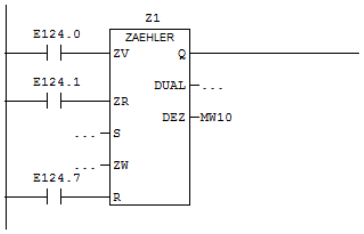
Dirección	Nombre	Tipo	Valor inicial	Comentario
0.0		STRUCT		
+0.0	MINUTOS_LUZ	INT	0	
+2.0	MINUTOS_OSCURIDAD	INT	0	
+4.0	DATO_CONTENTAS	INT	0	
+6.0	DATO_TRISTES	INT	0	
+8.0	DATO_INDIFERENTES	INT	0	
=10.0		END_STRUCT		

Para contar los huevos, haremos una función FC1 llamada “CONTADOR” con dos contadores; uno para contar los huevos puestos hoy y otro para los huevos puestos ayer. Los datos de los contadores los enviaremos a sus correspondientes variables en DB1. Las puestas de huevos la simularemos mediante contactos.

FC1 : CONTADOR DE HUEVOS

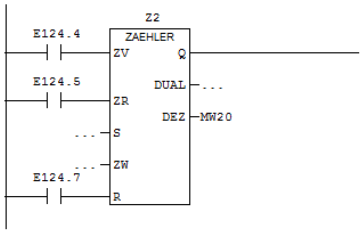
Segm. 1 : Título:

E124.0 Incrementa los huevos puestos hoy.
E124.1 Decrementa los huevos puestos hoy.



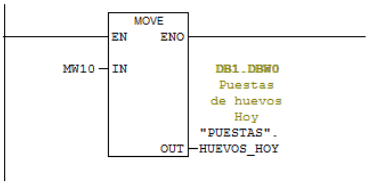
Segm. 2 : Título:

E124.4 Incrementa los huevos puestos ayer.
E124.5 Decrementa los huevos puestos ayer.



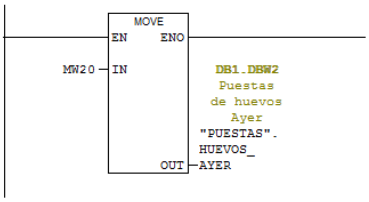
Segm. 3 : Título:

El número de huevos puestos hoy los enviamos al bloque de datos DB1 "PUESTAS".HUEVOS_HOY.



Segm. 4 : Título:

El número de huevos puestos ayer los enviamos al bloque de datos DB1 "PUESTAS".HUEVOS_AYER.

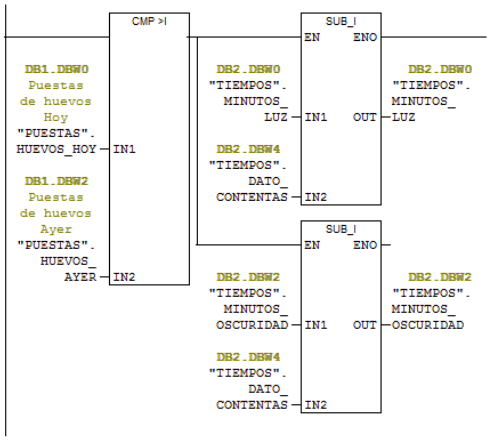


Con otra función FC2 llamada “COMPARATIVA” haremos la comparación de las puestas de huevos de ayer y de hoy, y según los resultados llevaremos a cabo las operaciones correspondientes.

FC2 : COMPARATIVA Y ACCIONES.

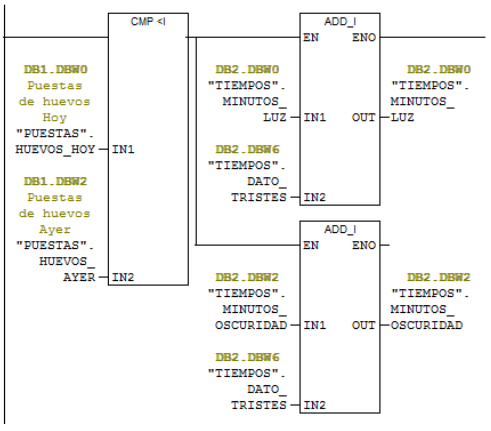
Segm. 1 : Título:

Si hoy han puesto más huevos que ayer, supondremos que las gallinas están contentas. Entonces lo que haremos será disminuir en 8 los minutos de luz y los minutos de oscuridad. De manera que les hacemos el día más corto.



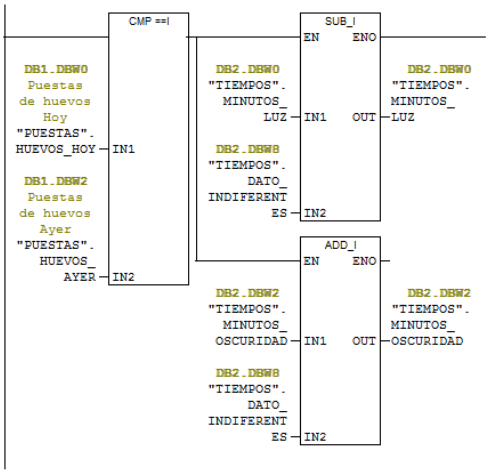
Segm. 2 : Título:

Si hoy han puesto menos huevos que ayer, supondremos que las gallinas están tristes. Entonces lo que haremos será aumentar en 5 los minutos de luz y los minutos de oscuridad. De manera que les hacemos el día más largo.



Segm. 3 : Título:

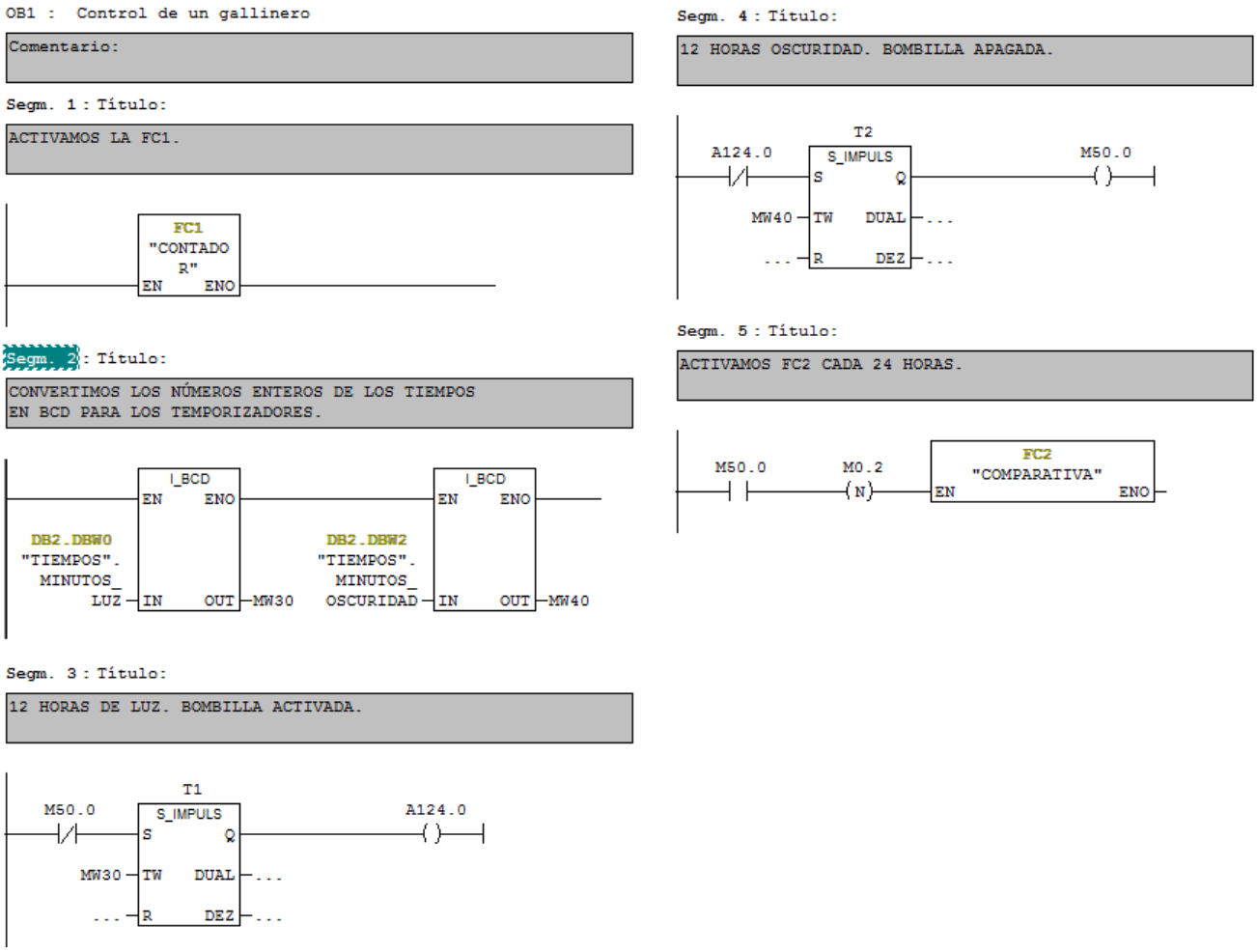
Si hoy han puesto los mismos huevos que ayer, supondremos que las gallinas están indiferentes. Entonces lo que haremos será disminuir en 1 los minutos de luz y los minutos de oscuridad. Iremos haciendo el día más corto poco a poco hasta que pongan menos huevos.



Finalmente, en el programa principal OB1 decidiremos cuando tenemos que acceder a cada uno de los bloques que hemos hecho.

La FC1 la va a tener que estar siempre activa. Siempre tiene que estar vigilando si hay un nuevo huevo y registrar la cuenta en su correspondiente DB.

La FC2 la tendrá que ejecutarse cada día. Tendríamos que hacer un temporizador con la suma de los minutos de luz y de oscuridad y cuando pase el día que haga la comparación y la suma o resta de los tiempos.

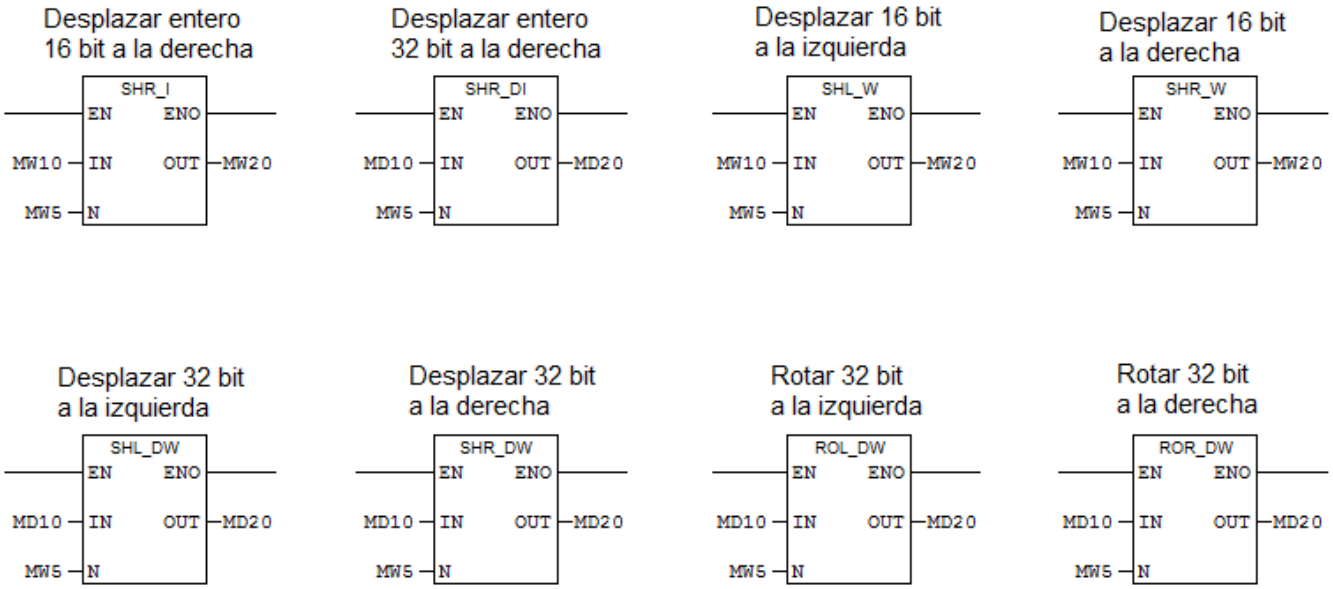


Podemos acortar los tiempos para la simulación con PLCSIM.

Dirección	Nombre	Tipo	Valor inicial	Comentario
0.0		STRUCT		
+0.0	MINUTOS_LUZ	INT	60	
+2.0	MINUTOS_OSCURIDAD	INT	60	
+4.0	DATO_CONTENTAS	INT	8	
+6.0	DATO_TRISTES	INT	5	
+8.0	DATO_INDIFERENTES	INT	1	
=10.0		END_STRUCT		

INSTRUCCIONES DE DESPLAZAMIENTO Y ROTACIÓN

Step 7 dispone de 8 instrucciones para desplazamiento y rotación de bit:



La entrada IN contiene la palabra o doble palabra a desplazar o rotar.
La entrada N nos indica la cantidad de posiciones que se tienen que mover los bit.
El resultado se entrega en OUT.

Veamos un ejemplo con la instrucción SHR_W (Desplazar 16 bit a la derecha)

Si MW10 contiene 00001011001110101 y MW = 3 , el desplazamiento será de 3 posiciones a la derecha. El resultado en MD20 sería el siguiente:

MW10 **0001 0110 0111 0101**

MW20 **0000 0010 1100 1110** **101** (se introducen 3 ceros desde la izquierda)

Si en lugar de desplazar lo que hacemos es rotar, lo que ocurre es que lo que se pierde por un lado entra por el lado contrario. Esta operación la hace en el acumulador. Sólo podemos rotar dobles palabras porque el acumulador es de 32 bits. Veamos un ejemplo con ROR_DW (Rotar 32 bit a la derecha).

Para MW5 = 2

MD10 11001010001100000011111110010001

111001010001100000011111110010001

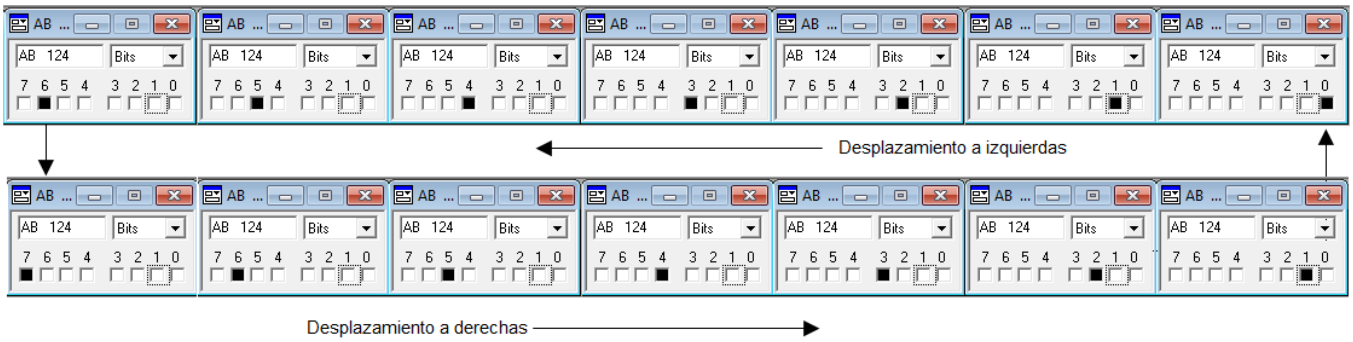
0111001010001100000011111110010001

MD20 **01110010100011000000111111100100**

EJERCICIO 23. Desplazamiento de bits.

Queremos que un bit del byte de salidas AB124 vaya desplazándose de izquierda a derecha y de derecha a izquierda, comenzando por el bit de la salida A124.0. La velocidad de desplazamiento será de 1 segundo.

El resultado visto con PLCSIM sería el siguiente:



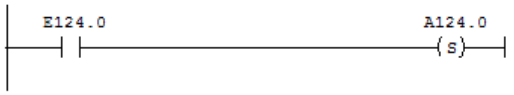
SOLUCIÓN:

OB1 : "Main Program Sweep (Cycle)"

DESPLAZAMIENTO DE BIT

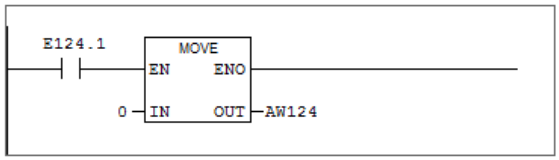
Segm. 1 : Título:

Tras pulsae E124.0 activa a124.0 permanentemente.



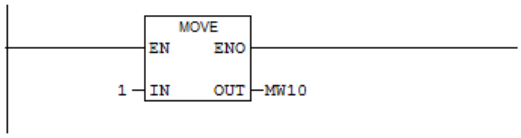
Segm. 2 : Título:

Tras pulsar E124.1 pon a 0 todas las salidas de AW124.



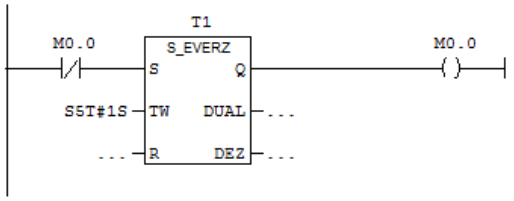
Segm. 3 : Título:

Desplaza 1 bit.



Segm. 4 : Título:

Tiempo de desplazamiento de 1 segundo

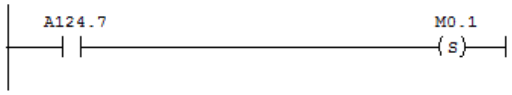


Segm. 5 : Título:

Cuando A124.0 se activa resetea M0.1 para desplazar los bit a la izquierda

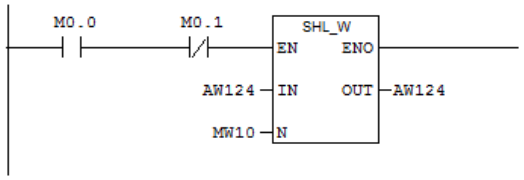
Segm. 6 : Título:

Cuando A124.7 este activa pon a 1 M0.1 para desplaz los bit a la derecha.



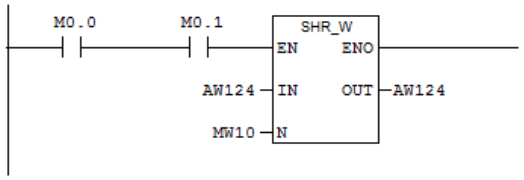
Segm. 7 : Título:

Desplaza bit a la izquierda.



Segm. 8 : Título:

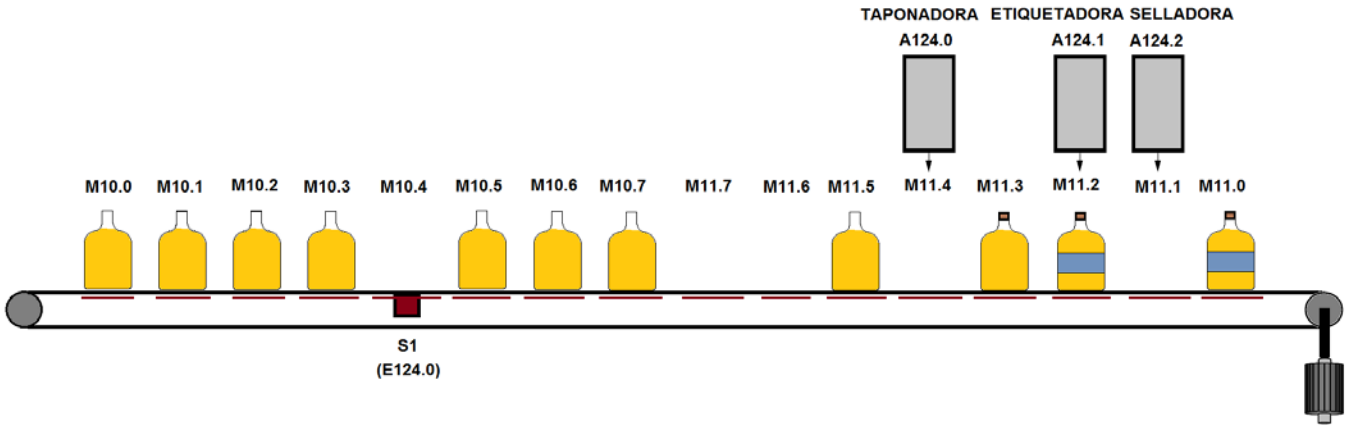
Desplaza bit a la derecha.



EJERCICIO 24. Planta Embotelladora (Desplazamiento y rotación de bits. Programación estructurada).

Tenemos una planta de embotellado distribuida en línea con tres máquinas: una taponadora, una etiquetadora y una selladora.

Las botellas se desplazan de posición a posición en la cinta cada segundo. Queremos que cuando las botellas lleguen debajo de las máquinas, éstas se pongan en marcha, pero si llega un hueco no queremos que las máquinas actúen. Con el sensor S1 (E124.0) detectamos cuando pasa una botella o cuando pasa un hueco.

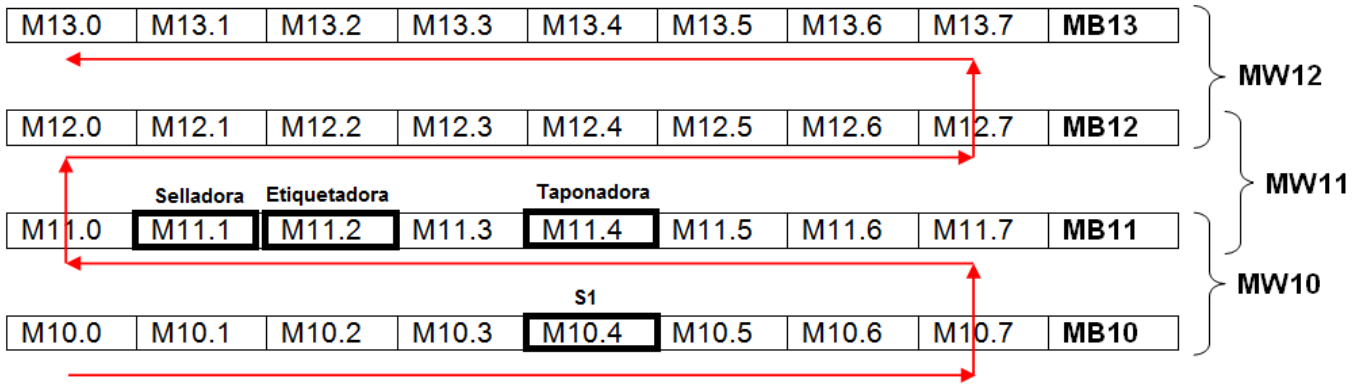


SOLUCIÓN:

Platearemos el problema, considerando 16 posiciones sobre la cinta. Cada posición estará relacionada con un bit en un registro de 16 bits (en nuestro caso la palabra MW10).

Cuando S1 detecte una botella, la marca M10.4 se pondrá a 1 y procederemos al desplazamiento hacia la derecha de este bit. El tiempo en cada posición lo fijaremos en 1 segundo mediante un generador de pulsos. Cuando este bit se sitúe en las marcas M11.4 (taponadora), M11.2 (etiquetadora) y M11.1 (selladora) se activará la salida correspondiente a cada máquina para que ejecute su proceso.

Para entender la distribución de marcas sobre la figura tenemos que tener en cuenta como se produce el desplazamiento de un bit hacia la derecha con la instrucción SHR_W. La figura siguiente nos lo aclara:



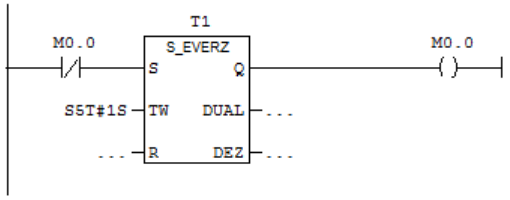
Para conocer mejor la utilidad de las funciones FC, vamos a resolver el problema utilizando 4 FC´s y una OB1.

En cada FC vamos a programar una de las operaciones que necesitamos realizar. Después desde el OB1 diremos cuando necesitamos realizar cada una de las operaciones.

FC1 – Generador de pulsos de 1 segundo

FC1 : GENERADOR DE PULSOS

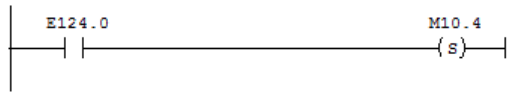
Segm. 1: Título:



FC2 – Con esta función vamos a meter un 1 en el bit 10.4 del registro de marcas MW10 cada vez que se detecte una botella.

FC2 : PONER BIT CUANDO LLEGE UNA BOTELLA

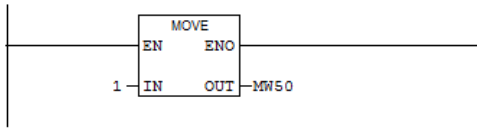
Segm. 1: Título:



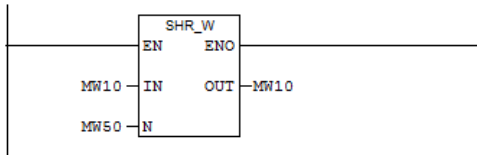
FC3 – Con esta función vamos a desplazar el bit M10.4 hacia la derecha. Observe en el gráfico anterior como se produce este desplazamiento.

FC3 : DESPLAZAMIENTO DE BOTTELLAS A LA DERECHA

Segm. 1: Título:



Segm. 2 : Título:

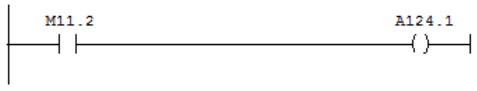


FC4 - En esta vamos a proceder a la activación de las máquinas cuando lleguen las botellas debajo de ellas.

FC4 : ACTIVACIÓN DE MÁQUINAS

Segm. 1: Título:

Segm. 2 : Título:

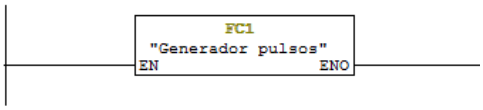


Segm. 3 : Título:

Ahora tan sólo nos queda programar el OB1 para organizar cuándo se tienen que ejecutar cada una de estas FC.

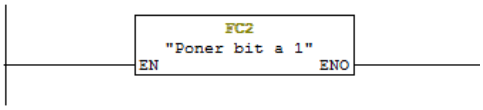
OB1- Programa principal.

```
OB1 :  "Main Program Sweep (Cycle)"
Segm. 1 : Título:
```



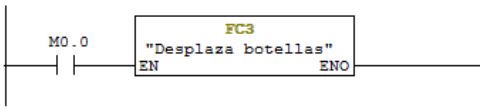
Ejecución incondicional de FC1.

```
Segm. 2 : Título:
```



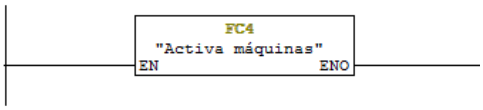
Ejecución incondicional de FC2.

```
Segm. 3 : Título:
```



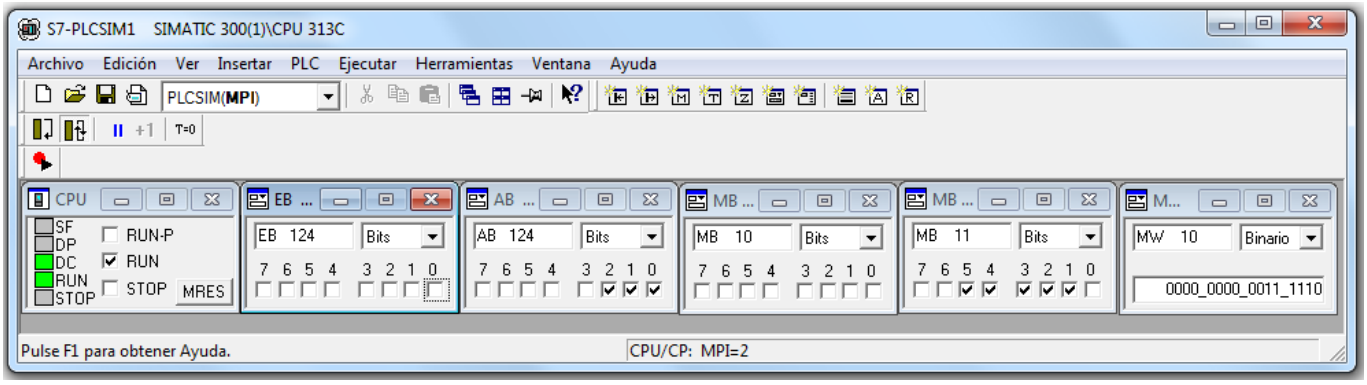
Ejecución condicional de FC3. Ejecuta FC3 cuando M0.0 se active. En nuestro caso cada segundo.

```
Segm. 4 : Título:
```



Ejecución incondicional de FC4.

La simulación del programa con PLCSIM le resultará útil

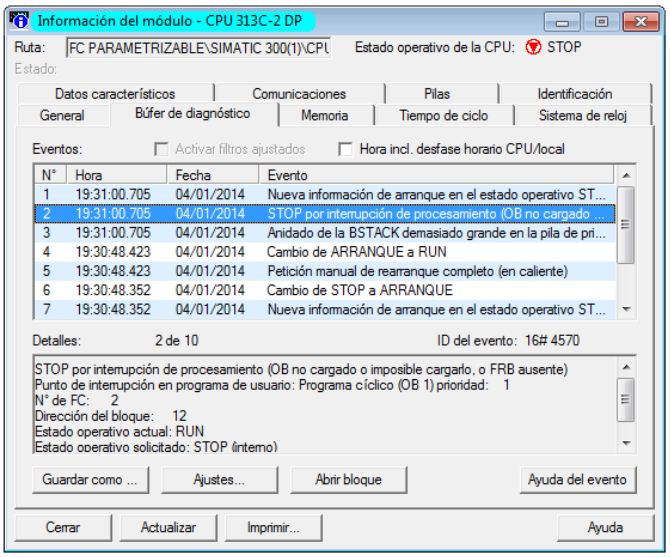


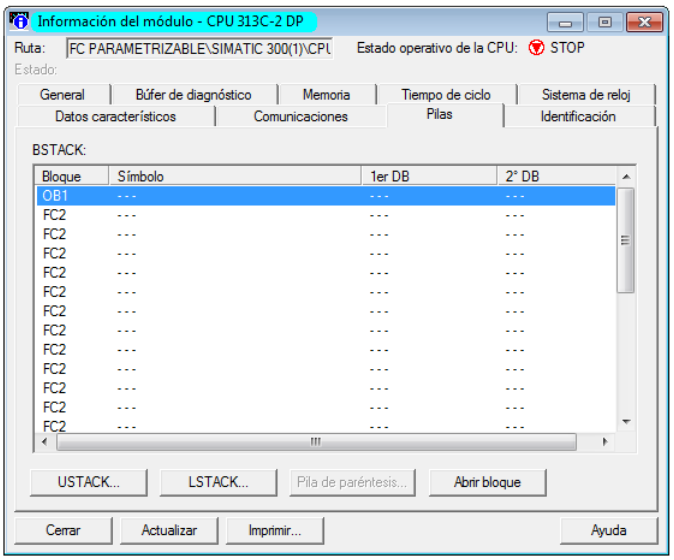
DETECCIÓN DE ERRORES. Menú “Información del módulo”.

En ocasiones, la CPU se nos va a STOP o simplemente nos da un error. Para estos casos, STEP 7 nos ofrece un menú de información donde nos dice cual es el problema por el cual se encuentra la CPU en ese estado.

Desde el menú principal de la ventana de programación, podemos acceder a dicho menú a través de ***Sistema de destino>información del módulo.*** Se nos mostrará una ventana como la siguiente:

En la pestaña de “*Búfer de diagnóstico”* se nos indica los pasos que ha seguido la CPU, con indicación del error.





La pestaña “Pilas” se nos indicará el camino que ha seguido la CPU antes de irse a STOP o generar el error.

Si tenemos activo el botón USTACK veremos el estado en el que estaba la CPU en el instante en el que se fue a Stop. Además tenemos el botón “Abrir bloque”. Si pinchamos en este botón se abrirá el bloque que contenía el error y nos señalará con el ratón la instrucción que contiene el error.

Además tenemos otras fichas en las cuales nos da más información del módulo. Podemos ver el tiempo máximo, el tiempo mínimo y el tiempo real del ciclo de scan.

Podemos ver la cantidad de memoria que estamos ocupando o el ciclo de scan que teníamos con nuestro programa, etc.

BLOQUES DE ORGANIZACIÓN (Otros OB´s)

Los bloques de organización (OB´s) son la comunicación entre el sistema operativo de la CPU y el programa de usuario. Con los OB´s podemos:

- Arrancar la CPU.
- Ejecución cíclica o intermitente temporalmente.
- Ejecución a determinadas horas del día.
- Ejecución después de transcurrir un tiempo preestablecido.
- Ejecución al producirse errores.
- Ejecución al producirse alarmas de proceso.

Cada uno de los OB´s tienen una prioridad asignada.

Tenemos la posibilidad de cambiar nosotros esta prioridad. No se pueden ejecutar nunca dos OB´s a la vez.

Relación de OB´s disponibles en Step 7:

OB1	Principal.
OB10.....0B17	Alarmas horarias.
OB20.....OB23	Alarmas de retardo (SFC 32)
OB30.....OB38	Alarmas cíclicas.
OB40.....OB47	Alarma de proceso.
OB80	Error de tiempo (SFC 43 redispara tiempo)
OB81	Fallo de alimentación.
OB82	Alarma de diagnóstico.
OB83	Alarma insertar/extraer módulos.
OB84	Avería CPU.
OB85	Error de ejecución.
OB86	Fallo de DP o en bastidor.
OB87	Error de comunicación.
OB100	Rearranque completo.
OB101	Rearranque.
OB121	Error de programación.
OB122	Error acceso a la periferia.

Descripción de algunas OB´s.

OB 1: Programa cíclico principal.

OB10...0B17 ALARMAS HORARIAS:

Se activan llamando a la SFC 30. Si no están activadas la CPU no las lee. Se programan con la SFC 28. Cuando se ejecutan una vez se anulan. También se pueden ejecutar periódicamente.

Si la CPU realiza un rearranque completo, cada OB de alarma horaria configurado mediante una SFC adopta otra vez la configuración ajustada con STEP 7. No puede ejecutar dos OB´s de alarma a la vez.

OB20...OB23 ALARMAS DE RETARDO:

Arrancan mediante una llamada a la SFC 32. Arrancan después de un tiempo preestablecido. Puede anularse una alarma que todavía no ha sido arrancada. (SFC 33). Un rearranque completo borra cualquier evento de arranque de un OB de alarma de retardo.

OB30.....OB38 ALARMAS CÍCLICAS:

Tienen un valor de tiempo prefijado. No puede durar más tiempo la ejecución del OB que el tiempo de volver a arrancar.

OB40...OB47 ALARMAS DE PROCESO:

Dentro de las alarmas de proceso, las prioridades se establecen con STEP 7. La activación se parametriza con STEP 7. No se pueden ejecutar dos a la vez.

OB DE ERROR DE TIEMPO:

Si no es programada la CPU pasa a STOP cuando se produce un error de tiempo. Si en un mismo ciclo se llama dos veces al OB 80 debido a la superación del tiempo de ciclo, la CPU pasa a STOP. Es posible evitar esto llamando a la SFC 43 en un lugar adecuado.

OB 100 REARRANQUE COMPLETO:

Cuando la CPU realiza un rearranque completo, accede al OB 100. Lee este bloque una sola vez y a continuación pasa a leer el OB 1 de manera cíclica. Hasta que no se ejecuta un nuevo rearranque no se vuelve a leer el OB 100. Normalmente se utiliza para establecer las condiciones iniciales de un proceso.

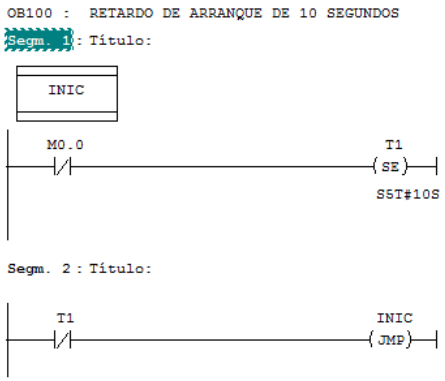
El OB 100 también es útil para retardar el arranque. El tiempo de ciclo de scan sólo afecta al OB 1. Nosotros podemos programar un bucle con un temporizador de 3 segundos dentro del OB 100, de manera que cuando ponemos la CPU en marcha, hasta los tres segundos no se empieza a leer el OB 1 y por consiguiente no se empieza a ejecutar el programa. Esto nos puede ser útil en instalaciones en las que tengamos, por ejemplo, variadores de frecuencia que necesiten unos segundos para estar a punto y empezar el programa.

EJERCICIO 25. (Trabajo con OB100). Retardo de arranque del PLC.

Retardar la ejecución del OB1 durante 10 segundos utilizando el OB100.

SOLUCIÓN:

Programaríamos en OB100 lo siguiente:

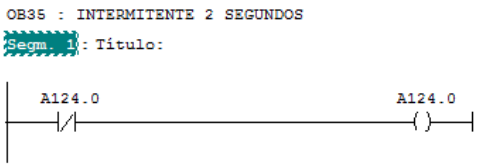


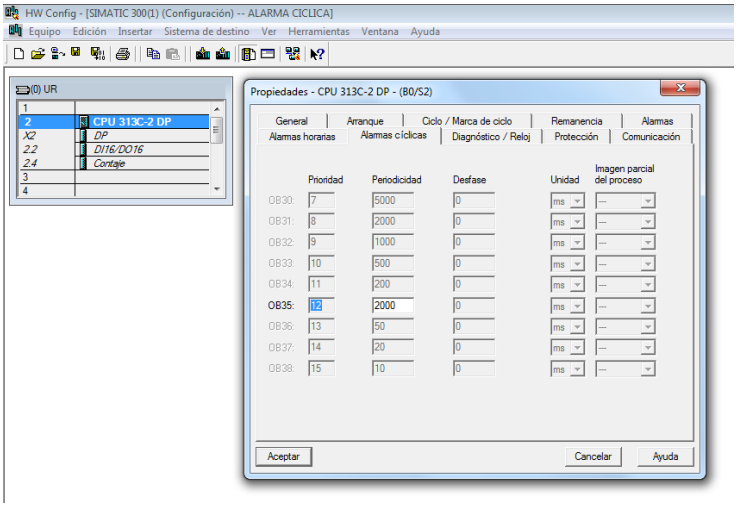
EJERCICIO 26. (Trabajo con OB's de Alarma Cíclica). Intermitente de 2 segundos.

Queremos que la salida A124.0 del PLC se ponga en modo intermitente cada 2 segundos.

Para esto vamos a configurar la alarma cíclica OB35 del PLC. Esto lo hacemos abriendo desde HW Config las propiedades de la CPU. En la pestaña "Alarmas cíclicas" definimos el tiempo de ejecución cíclico de OB35 en 2000 mseg.

A continuación, insertaremos un OB35 junto al OB1 y lo programamos de la siguiente manera:





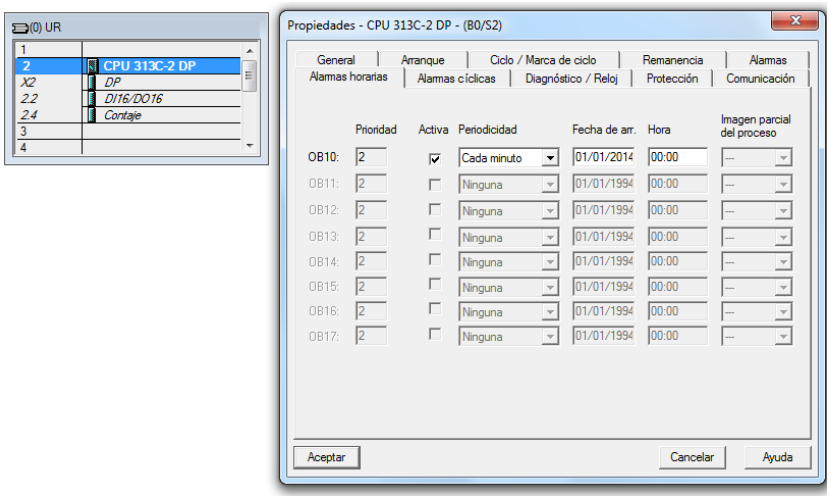
Finalmente, cargaremos sobre la CPU, la configuración y los correspondientes OB1 y OB35.

EJERCICIO 27. (Trabajo con OB's de Alarma Horaria).

Queremos que cada minuto a partir de una fecha y hora determinada se ejecute un programa que nos active el byte de salidas AB124.

Para programar este tipo de alarmas, tenemos que hacerlo a través del hardware. Vamos a hacer un proyecto nuevo, y vamos a entrar en el hardware. Una vez lo tengamos definido, pinchamos encima de la CPU con el botón derecho, y entramos en propiedades de la CPU.

Veremos que tenemos varias fichas. Vamos a la ficha de alarmas horarias. Dependiendo de la CPU que tengamos, tendremos unos OB disponibles para programar alarmas. Por ejemplo, si tenemos una CPU 313C-2DP veremos que sólo tenemos disponible el OB10.



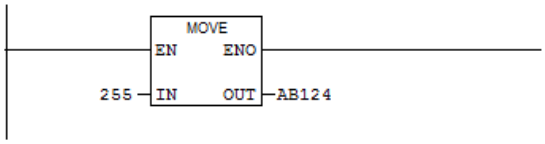
Para programar la alarma tendremos que activar la casilla "activa". Además tendremos que decirle cada cuanto tiempo queremos que se produzca, y desde cuando queremos que se active.

Esto significa que cada cierto tiempo, se va a acceder a la OB 10. Se ejecutará lo que allí diga y el programa seguirá luego por donde iba.

Luego tendremos que programar el OB10. Lo que no habrá que hacer, será programar una llamada al OB 10. A los OB no se les llama.

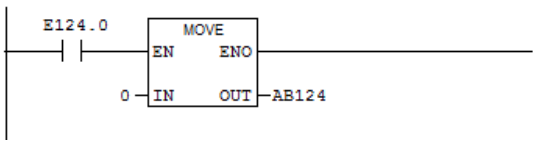
En OB10 tendríamos lo siguiente:

```
OB10 :  "Time of Day Interrupt"
Segm. 1: Título:
```



OB10 *- Programa que se ejecutará cada minuto.*

```
OB1 :  "Main Program Sweep (Cycle)"
Segm. 1: Título:
```



0B1*- Para poner las salidas nuevamente a cero.*