2 Operaciones de bit. Instrucciones binarias.

Contenidos del Capítulo 2

2 Operaciones de bit. Instrucciones binarias.

2.1 Indice de ejercicios.

19.- Escalera automática.

20.- Instrucción MASTER CONTROL RELAY.

2.2 Creación del primer proyecto.

Vamos a realizar el primer proyecto en STEP 7. Dentro de un proyecto de STEP 7 introduciremos todos aquellos equipos que vayan a formar parte de nuestra instalación. Introduciremos tanto los equipos de SIEMENS que estemos gastando, así como los equipos de otras marcas, y los PC en caso de que los hubiera. De este modo podemos comunicarlos de modo sencillo y podemos visualizar todos los equipos de la instalación con una sola programadora desde un solo punto de la instalación.

Además, con el software NETPRO (incluido en el STEP 7 a partir de la versión 5.x) podremos visualizar de forma gráfica, las redes y conexiones entre los diferente equipos. Además podremos manejar estos enlaces a las redes con el ratón de la programadora de forma gráfica.

En el primer proyecto que vamos a realizar en el curso, vamos a insertar un solo equipo.

Dentro del equipo vamos a incluir tanto el hardware que estemos utilizando, como los bloques de programación (programa propiamente dicho).

Esto nos aportará varias ventajas. La programadora “sabrá” el equipo que vamos a gastar en nuestro trabajo. Además sabrá las tarjetas que tenemos

instaladas y las opciones de las que dispone cada tarjeta. Si intentamos utilizar alguna instrucción que no soporta nuestra CPU, nos avisará indicándonos que aquella instrucción es imposible. Además cuando entremos en la opción “Propiedades del objeto” de cualquier objeto que tengamos en la instalación, tendremos disponibles las propiedades de ese objeto en concreto.

Además tendremos la posibilidad de ajustar desde el software propiedades del propio hardware. Por ejemplo nos permitirá cambiar las alarmas, el tiempo de ciclo de scan preestablecido para la propia CPU, las direcciones de cada uno de los objetos, etc.

Cada vez que hagamos un proyecto nuevo tendremos que definir un hardware nuevo para cada uno de los equipos que tengamos en la red.

Veamos como haríamos esto prácticamente.

Abrimos el Administrador de SIMATIC. Tenemos disponible un asistente de nuevo proyecto. Si queremos utilizar el asistente, no tenemos más que ir contestando a lo que se nos pregunta. Si queremos podemos cancelar el asistente y generar nosotros nuestro nuevo proyecto.

Si hacemos el proyecto con el asistente, no tendremos en el proyecto las tarjetas de entradas y salidas que estamos gastando.

Para generar un nuevo proyecto, podemos ir bien al menú “Archivo > nuevo”, o bien al botón que tiene como icono una hoja en blanco.

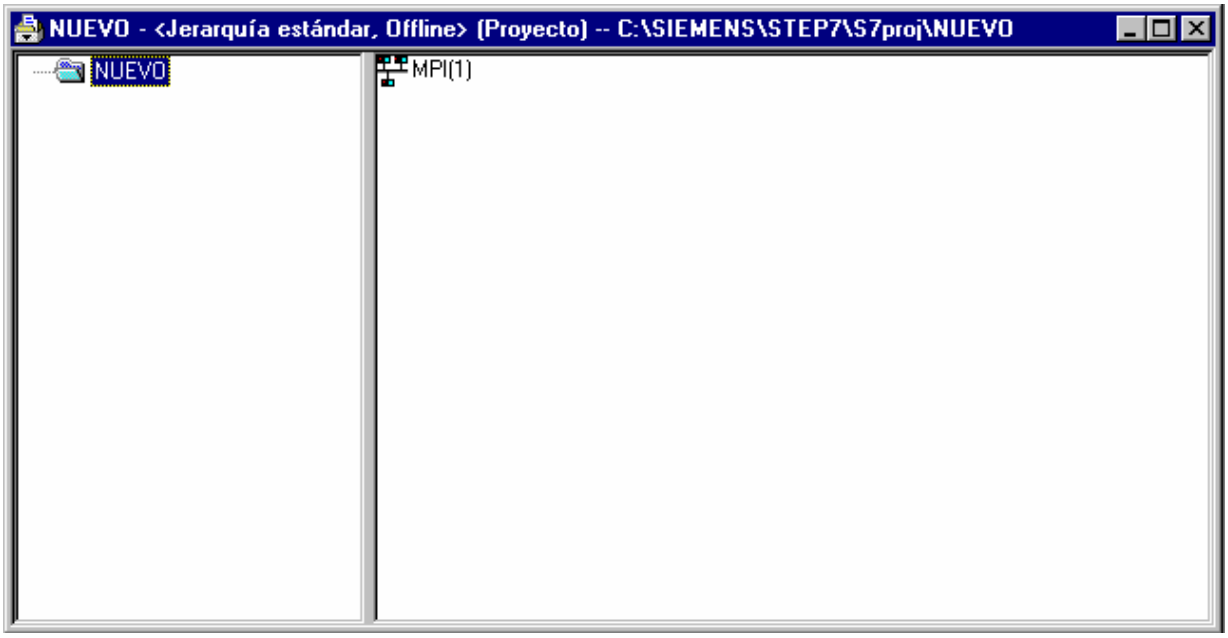
Aparece una ventana en la que podemos decir si queremos crear un proyecto nuevo o una librería nueva.

En nuestro caso decimos que queremos un proyecto nuevo.

A continuación le damos el nombre que queramos al proyecto. Una vez tenemos el nombre, aceptamos y observaremos que tenemos una ventana con el nombre de nuestro proyecto en la parte izquierda y con la red MPI en la parte derecha.

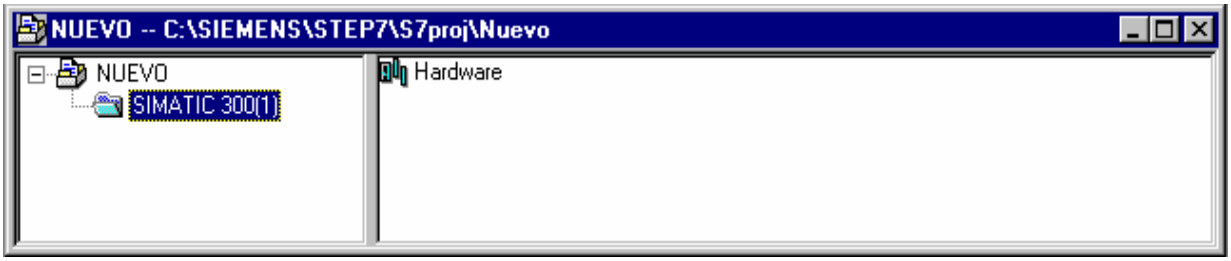
El icono de la red MPI aparece por defecto. Es necesario que tengamos al menos una red MPI porque la programación de la CPU se hace a través del puerto MPI de la CPU.

Posteriormente podremos insertar tantas redes como nos haga falta.

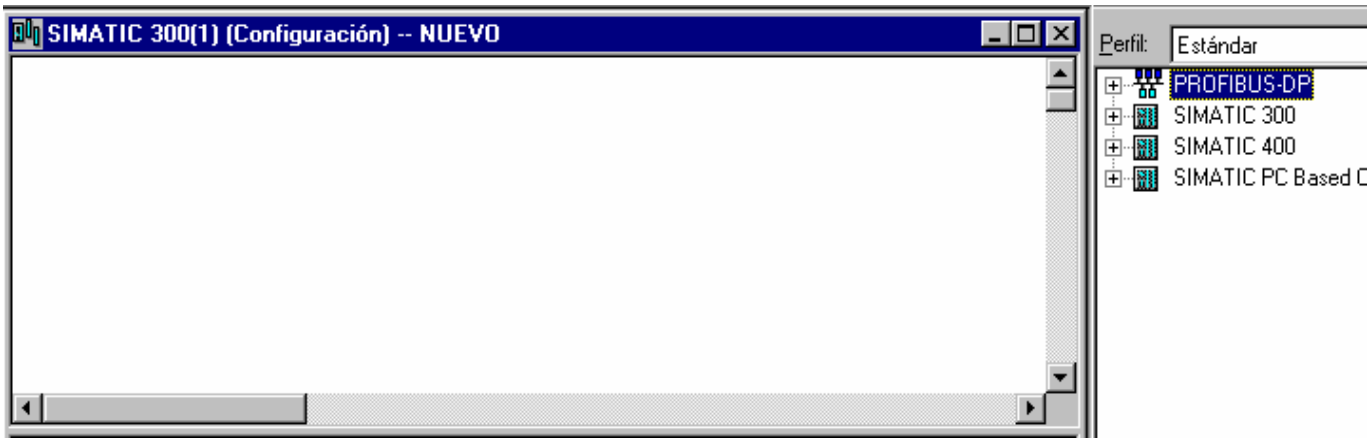


Vamos a rellenar el proyecto. Lo primero que tenemos que hacer es insertar los equipos que van a formar parte de nuestro proyecto. En este caso vamos a insertar un solo equipo. Para ello, vamos al menú de “insertar” y elegimos el equipo con el que vayamos a trabajar. En este caso tenemos un equipo 300.

Veremos que en nuestra ventana del proyecto se ha creado un equipo. Hacemos doble clic sobre el equipo, y en la parte derecha de la ventana, veremos que aparece un icono que se llama “Hardware”.



Hacemos doble clic sobre él y entramos el editor de hardware. En principio veremos que está todo en blanco. Para insertar los módulos que nosotros tenemos en nuestro equipo, tendremos que abrir el catálogo. Suele estar abierto por defecto. Si no lo está, podemos abrirlo con el botón que representa un catálogo, o desde dentro del menú de "Ver", con la opción "Catálogo". (También funciona con la combinación de teclas Ctrl K).



Una vez tengamos el catálogo abierto, desplegamos la cortina del equipo que tengamos que definir. En este caso desplegamos la cortina de SIMATIC 300.

Lo primero que tenemos que insertar es un bastidor. Desplegamos la cortina de los bastidores, y vemos que tenemos un perfil soporte. Hacemos doble clic sobre

el perfil soporte. Veremos que en la instalación del hardware se sitúa en la posición cero.

A continuación tenemos que ir rellenando el resto de posiciones que tengamos ocupadas.

Nos situamos en la posición 1 y vamos a insertar la fuente de alimentación que es lo primero que tenemos en nuestro equipo.

Desplegamos la cortina de las PS, y elegimos la que tengamos en cada caso.

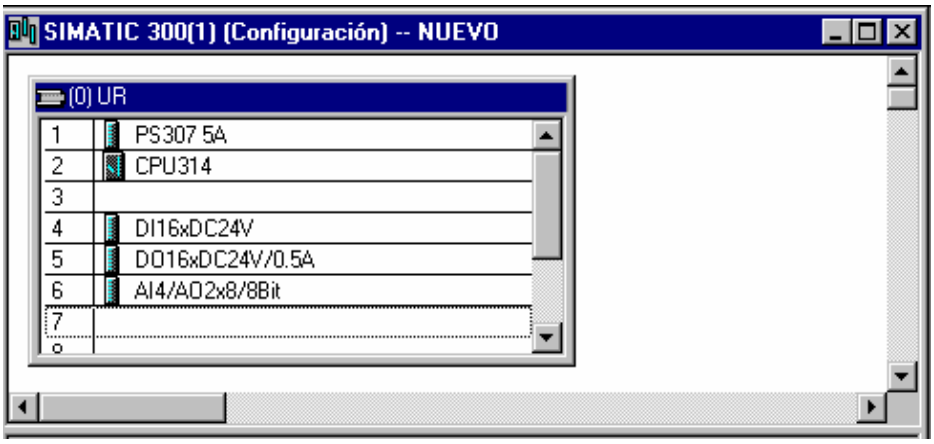
A continuación nos situamos en la posición 2 para insertar la CPU. Desplegamos la cortina de las CPU. Vemos que existen varias del mismo modelo. Si pinchamos una sola vez encima de cada una de las CPU (o cualquier otro elemento del catálogo), vemos que en la parte inferior del catálogo, tenemos una pequeña explicación sobre el elemento seleccionado, y además la referencia del elemento.

Tenemos que comprobar que esta referencia coincida con la referencia del elemento que tenemos nosotros.

En la posición 3 no podemos insertar cualquier módulo. Es una posición reservada para los módulos IM. Estos módulos sirven para realizar configuraciones en más de una línea de bastidor. En nuestro caso tenemos una única línea de bastidor. No tenemos tarjeta IM. En este caso tenemos que dejar la posición 3 libre.

Pasamos a la posición 4. En ella y en las siguientes posiciones, tenemos que insertar los módulos de entradas/salidas que tengamos. Si encontramos en el catálogo varios del mismo modelo, tendremos que comprobar para cada caso la referencia del elemento.

El hardware que tenemos en el equipo didáctico quedaría de la siguiente manera:



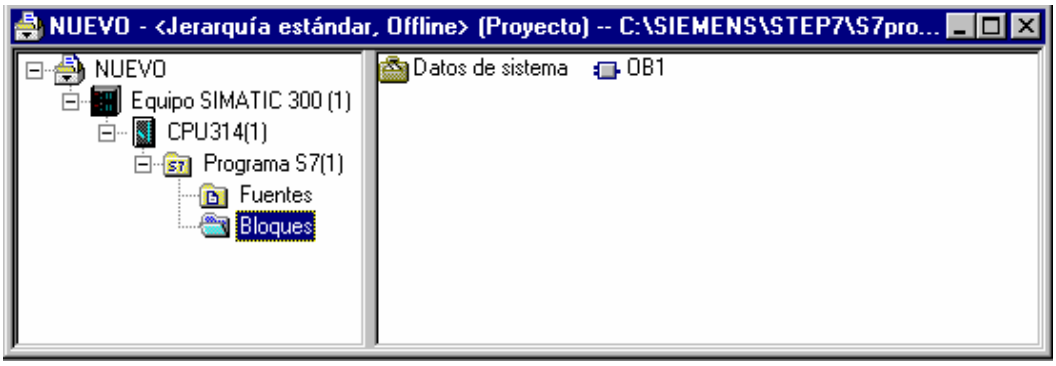
Veremos que debajo de esta tabla, se va creando otra en la que podemos ver los elementos que vamos creando con sus referencias y además con las direcciones que le corresponde a cada uno de los módulos.

Una vez hemos terminado la configuración, tenemos que guardarla. Estamos trabajando con dos CPU a la vez. Estamos trabajando con la programadora y con el PLC. Tenemos que guardar la información en ambos sitios. Con el icono que representa un disquete, guardamos la información en la programadora, y con el icono que representa un PLC y una flecha que entra, guardamos la información en el PLC. Una vez tenemos la información guardada, salimos del editor de hardware.


Veremos que volvemos a la misma ventana en la que estábamos antes, es decir, volvemos al administrador de SIMATIC. En nuestro proyecto, tenemos el equipo. Vemos que al lado del equipo hay un + . Si desplegamos todo lo que

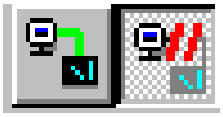
tenemos, vemos que dentro del equipo está la CPU, la carpeta para el programa, los bloques y las fuentes. Si pinchamos encima de los bloques, vemos que en la parte derecha tenemos los datos de sistema y el OB 1.

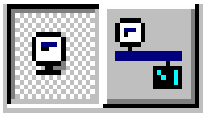
Es el primer bloque que vamos a programar. Aparece por defecto, aunque está vacío.



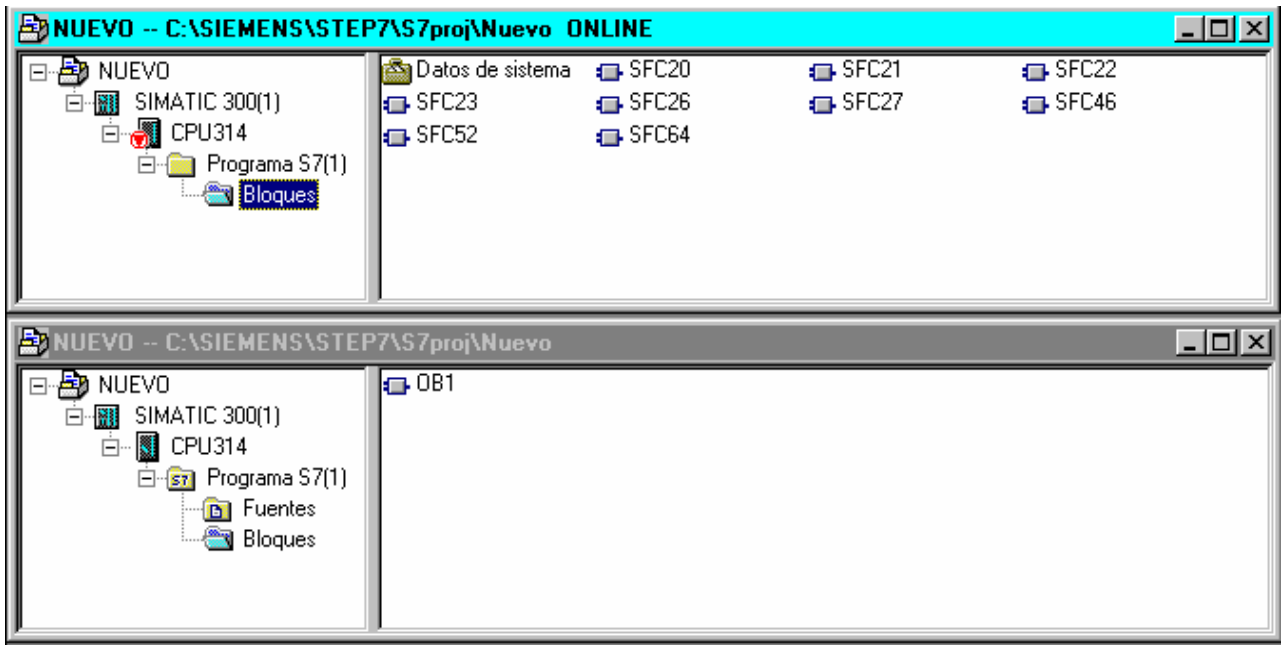
Estando en el administrador de SIMATIC vemos que tenemos abierta una ventana con nuestro proyecto que acabamos de crear. En los iconos de la parte superior vemos que tenemos uno seleccionado que representa un PC y un PLC cortados por una doble barra roja. Este es el icono de “offline”. A su lado vemos que tenemos otro icono que representa el mismo PC y el mismo PLC pero unidos con una línea verde. Este es el icono de “online”.

Versiones 3 y 4 Versión 5





En este momento tenemos apretado el icono de “offline”. Apretamos también el icono de “online”. Vemos que en el administrador de SIMATIC tenemos dos ventanas parecidas. Con May. F2, nos organizamos las ventanas.



Siempre que estemos en “offline”, estamos trabajando en la programadora. Estamos leyendo del disco duro de la programadora. Vemos que en los bloques sólo tenemos el OB 1 que es el que ha creado el proyecto por defecto.

Siempre que estemos en “online”, estamos trabajando directamente en el PLC. Estamos leyendo directamente lo que tenga el PLC. Vemos que tenemos otros bloques. Son los bloques que lleva integrados y protegidos la CPU. Dependiendo de la CPU con la que estemos trabajando, tendremos unos bloques diferentes. Estos bloques no los podemos borrar. Tampoco podemos ver lo que hay programado en ellos. Sólo podemos utilizarlos. Tenemos una ayuda de cada uno de ellos en la que nos explica como se llama cada uno de los bloques, lo que hace y como debemos utilizarlo y rellenarle sus parámetros. Para ver esta ayuda, sólo tenemos que seleccionar el bloque que queremos, y en esta posición pulsar la tecla F1.

Si aparecen bloques que no sean los de sistema (SFC´s o SFB´s) quiere decir que en el PLC tenemos algún programa. Antes de empezar con nuestro programa vamos a borrar todo lo que tenga el PLC.

Para ello pinchamos encima de la CPU de online. En esta posición vemos al menú Sistema Destino, y elegimos la opción “Borrado Total”.

Con esto borramos todos los bloques que tuviera la CPU excepto los de sistema. El hardware no lo borramos.

Si volvemos a pinchar en bloques veremos que sólo tenemos los de sistema.

A la hora de trabajar sobre los distintos bloques, lo podemos hacer tanto en “offline” como en “online”. A la hora de guardar lo que hemos programado, lo podemos guardar tanto en la programadora como en la CPU, tanto si estamos trabajando en “online” como si estamos trabajando en “offline”.

A la hora de trabajar con los distintos bloques, tenemos que tener en cuenta que en un momento dado podemos llegar a estar trabajando con tres bloques con el mismo nombre a la vez. Por ejemplo, supongamos que estamos trabajando con un OB 1. El bloque que estamos viendo en la pantalla de la programadora, mientras no lo guardemos en ningún sitio, lo tenemos únicamente en la RAM. Se puede dar el caso de que tengamos un bloque en el PLC, tengamos otro bloque diferente guardado en disco duro, y acabemos de hacer una modificación y la tengamos en la pantalla de la programadora pero todavía no la hayamos transferido. Si teniendo el bloque en la pantalla pinchamos el icono de “guardar” o el icono de “transferir al autómata”, estaremos guardando el disco duro o en el autómata lo que tengamos en la pantalla. Pero si volvemos a la pantalla principal (Administrador de Simatic) sin

haber guardado previamente el bloque en disco duro, y transferimos algún bloque arrastrándolo con ayuda del ratón, vamos a transferir lo último que hubiésemos guardado en disco duro, y no las últimas midificaciones que hemos hecho en el bloque.

¿Cuántos tipos de bloques podemos programar?

OB	Bloques de sistema.
FC	Funciones.
FB	Bloques de función.
DB	Bloques de datos.
UDT	Tipo de datos.

Veamos lo que podemos hacer con cada uno de estos bloques.

OB: Son bloques de organización. Cada OB tiene una función determinada. El OB 1 es el único bloque de ejecución cíclica. Es el que ejecuta la CPU sin que nadie le llame. Los demás OB´s tienen una función determinada. Se ejecutan cuando les corresponda sin que nadie les llame desde ningún sitio del programa. Tenemos OB´s asociados a diferente errores de la CPU, a alarmas, etc.

FC: Funciones. Son trozos de programa que yo me creo. Realizan una función determinada dentro de mi proyecto. Se ejecutan cuando se las llama desde algún punto de mi programa. Pueden ser parametrizables o no. Además de las FC´s que yo me creo, existen FC´s hechas en librerías. Se utilizan exactamente igual que las que yo programo. No podemos entrar en ellas para ver la programación.

FB: Bloques de función. En principio funcionan igual que las FC. La diferencia está en que las FB se guardan la tabla de parámetros en un módulo de datos. Esto tiene dos ventajas. Una es que podemos acceder a los parámetros desde cualquier punto del programa. Otra es que cada vez que llamemos a la FB no es necesario que le demos todos los parámetros . Los parámetros que no rellenemos, se tomarán por defecto los últimos que hayamos utilizado.

DB: Módulos de datos. En estos bloques no realizamos programa. Son tablas en las que guardamos datos. Luego podremos leerlos y escribir sobre ellos.

UDT: Tipo de datos. Nos podemos definir nuestros propios tipos de datos para luego utilizarlos en los DB.

2.3 Ejercicios resueltos.

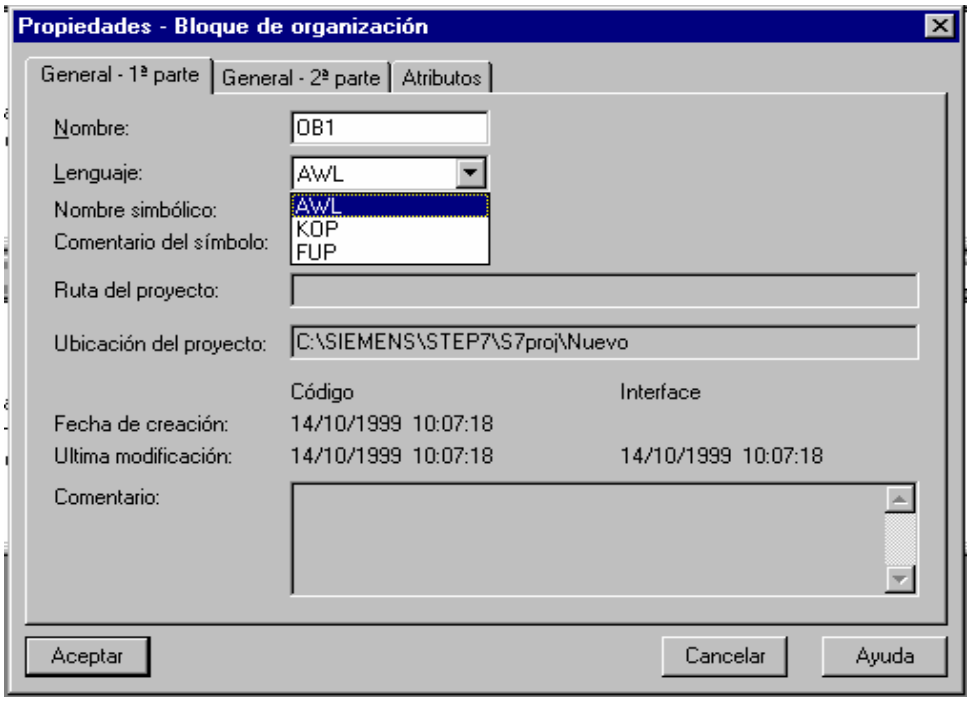
EJERCICIO 1: CONTACTOS EN SERIE.

TEORÍA

CREACIÓN DEL PRIMER BLOQUE

Vamos a programar nuestro primer OB 1. Entramos en el OB 1 de “offline”.

Nos aparece una ventana en la que tenemos que elegir el lenguaje de programación.



Tenemos 3 posibilidades: KOP, FUP y AWL.

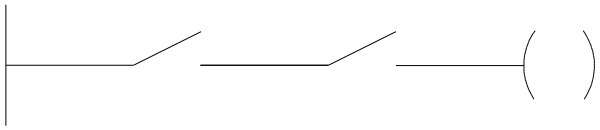
KOP: Esquema de contactos.

FUP: Funciones.

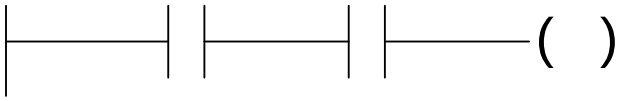
AWL: Lista de instrucciones.

Supongamos que lo que queremos programar son dos contactos en serie. Veamos como quedaría hecho en cada uno de los tres lenguajes:

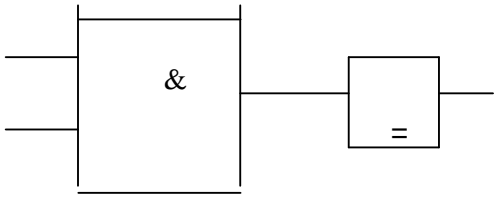
Circuito que queremos programar:



Solución en KOP:



Solución en FUP:



Solución en AWL:

```
U     E     0.0
```

```
U     E     0.1
=     A     4.0
BE
```

En principio elegimos lista de instrucciones. En STEP 7, podemos hacer toda la programación que queremos en cada uno de los tres lenguajes.

Vamos a programar dos contactos en serie.

Tenemos que dar nombre a cada uno de los contactos. Tendremos que direccionar las entradas y salidas de que disponemos.

El direccionamiento de tarjetas en un S7 - 300 es el siguiente:

PS	CPU	0	4	8	13	.....
		1	5	9	14	
		2	6	10	12	
		3	7	11	13	

Las direcciones son las mismas independientemente de que las tarjetas sean de entradas o de salidas. Como vemos, tenemos ocupados 4 bytes para cada posición de tarjeta. Si tenemos tarjetas de 2 bytes, estamos perdiendo estas dos direcciones.

En el caso de tarjetas analógicas, el direccionamiento es el siguiente:

PS	CPU	256	272	288

También es independiente de que las tarjetas sean de entradas o de salidas. Como vemos, para cada puesto de tarjeta tenemos reservados 16 bytes.

En nuestro caso, tenemos la siguiente configuración.

PS	CPU	0 1	4 5	288 290 292 294 288

Tendremos bytes de entradas 0 y 1, y bytes de salidas 4 y 5. La tarjeta de entradas/salidas analógicas, ocupa la posición 288. Tendremos palabras de entrada 288, 290, 292, 294, y palabras de salidas 288 y 290.

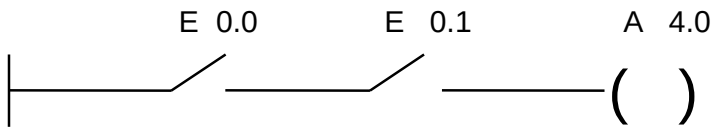
Instrucción “U”: Es la instrucción que utilizaremos para unir varias condiciones en serie. La instrucción sirve tanto para primera consulta como para el resto de condiciones en serie.

EJERCICIO 1: CONTACTOS EN SERIE

TEORÍA PREVIA: Introducción. Generación de nuevo proyecto.

DEFINICIÓN Y SOLUCIÓN

Supongamos que queremos automatizar el siguiente circuito:



Vemos que lo que tenemos son dos contactos en serie.

Tenemos que asignar nombre a cada uno de los contactos. A las entradas les vamos a llamar E y a las salidas les vamos a llamar A. Esto corresponde a las iniciales en alemán (Lenguaje SIMATIC). También podemos cambiar y elegir las iniciales en inglés. (Lenguaje IEC). El cambio lo podemos hacer en el menú “Herramientas” “Preferencias”. De aquí en adelante trabajaremos con lenguaje SIMATIC.

A parte de darles nombre a las entradas y a las salidas, tenemos que darles una numeración.

Como hemos visto anteriormente, el direccionamiento de entradas y salidas, depende únicamente de la posición que ocupen en el rack. Es decir, la primera tarjeta, que en nuestro caso es una tarjeta de entradas, van a ser los bytes 0 y 1.

La siguiente tarjeta en nuestro caso es una tarjeta de salidas, ocupará las posiciones 4 y 5.

Por tanto tendremos disponibles 16 entradas (desde la 0.0 hasta la 1.7) y 16 salidas (desde la 4.0 hasta la 5.7)

Las direcciones que no estamos utilizando, las podríamos utilizar en periferia descentralizada.

Para unir los dos contactos en serie disponemos de la instrucción “U”.

El siguiente ejercicio lo podríamos resolver en los tres lenguajes que nos permite el STEP 7. AWL, KOP y FUP.

Veamos primero la programación en AWL.

SOLUCIÓN EN AWL

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

La instrucción BE es opcional. Significa final de programa. Si no la escribimos no pasa nada. Cuando el autómata lee la última instrucción del OB 1 vuelve a empezar la lectura por el principio.

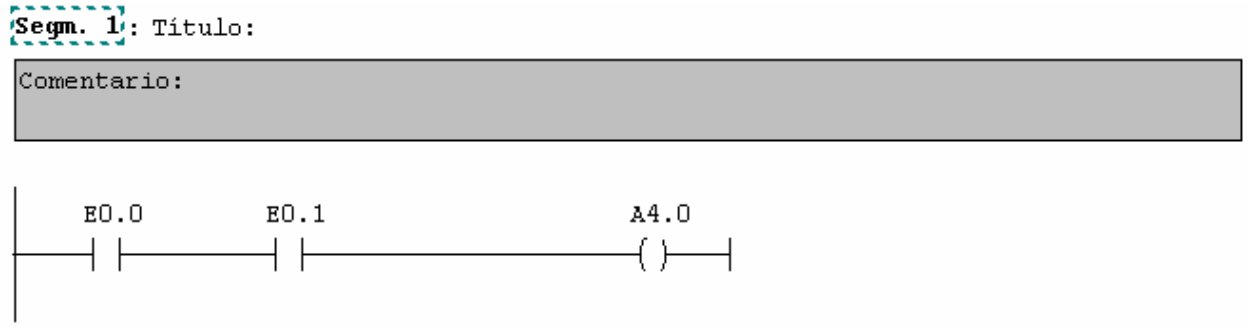
Una vez tenemos el programa hecho, podemos cambiarlo de lenguaje. Esto lo podremos hacer siempre y cuando el programa sea traducible. En este caso, por

ejemplo, la instrucción BE no existe ni en KOP ni en FUP. Si intentamos traducir esto nos va a decir que no es traducible.

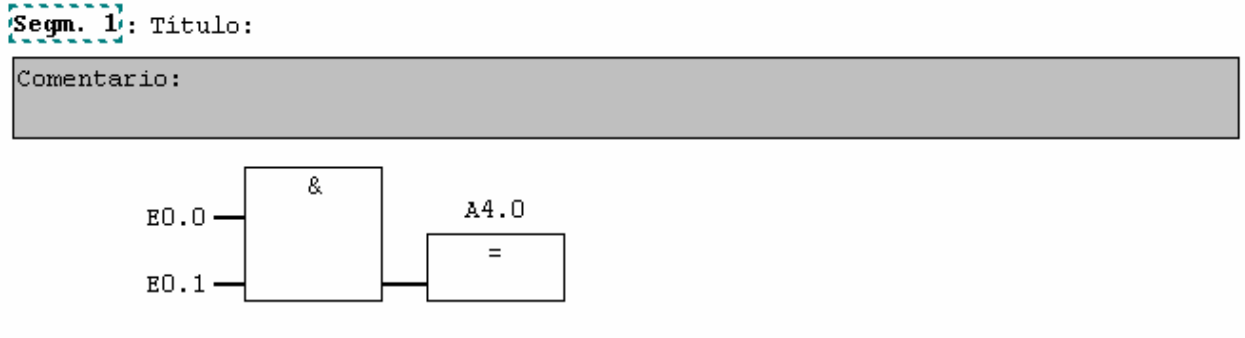
Si le quitamos el BE veremos que ya lo podemos traducir. En AWL podemos poner o no poner la instrucción BE.

Veremos que tenemos el mismo programa en AWL en KOP o en FUP.

SOLUCION EN KOP



SOLUCIÓN EN FUP



EJERCICIO 2. CONTACTOS EN PARALELO.

TEORÍA

INSTRUCCIÓN “O”

Para unir dos contactos en paralelo tenemos la instrucción “O”.

Con esta instrucción unimos varias condiciones en paralelo. La instrucción nos sirve tanto para instrucciones de primera consulta como para el resto de condiciones.

Para la instrucción de primera consulta, podemos utilizar tanto la instrucción “U” como la instrucción “O”.

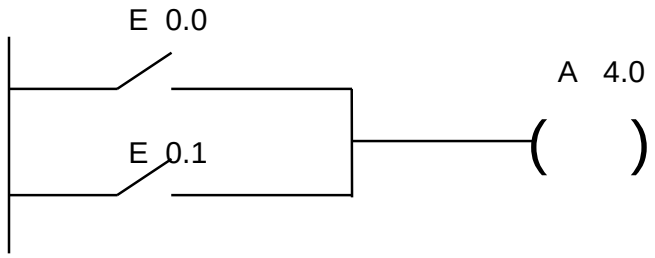
El programa funcionaría exactamente igual.

EJERCICIO 2: CONTACTOS EN PARALELO.

TEORÍA PREVIA: Instrucción “O”.

DEFINICIÓN Y SOLUCIÓN.

Veamos cómo podríamos resolver el siguiente circuito eléctrico:



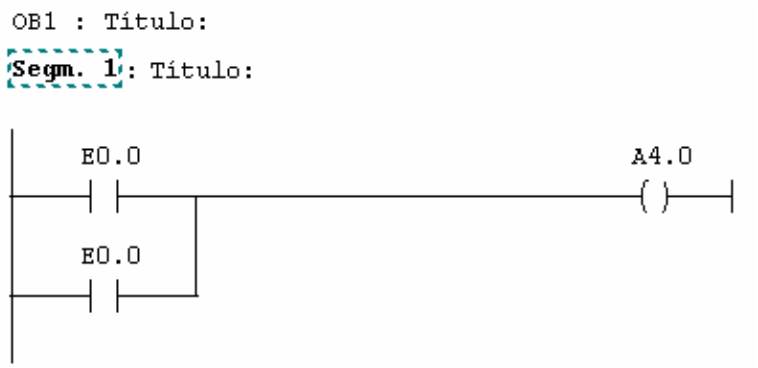
Vemos que lo que tenemos son dos contactos en paralelo.

Para programar los contactos en paralelo tenemos la instrucción “O”.

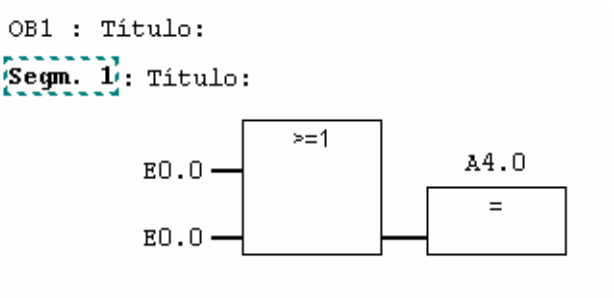
SOLUCIÓN EN AWL

```
U   E   0.0     (también   O   E   0.0)
O   E   0.1
=   A   4.0
BE
```

SOLUCIÓN EN KOP



SOLUCIÓN EN FUP



EJERCICIO 3. UTILIZACIÓN DEL PARÉNTESIS.

TEORÍA

INSTRUCCIONES O(Y U(

Hasta ahora hemos visto la manera de unir varias condiciones en serie y condiciones en paralelo.

También podemos combinar series con paralelos. Para ello, nos hará falta utilizar los paréntesis. (Solamente si hacemos la programación en instrucciones)

Para abrir el paréntesis lo haremos siempre al lado de una instrucción. Por ejemplo U(O(

Para cerrarlo lo haremos en una instrucción el paréntesis solo).

También podemos obviar los paréntesis. Para ello dejaríamos la instrucción que abre el paréntesis sola en una línea, y luego no cerramos el paréntesis.

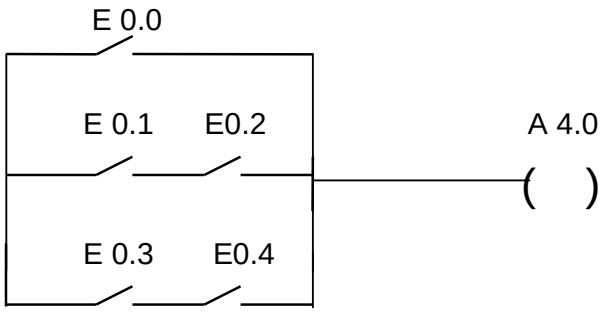
El significado sería el mismo.

EJERCICIO 3: UTILIZACIÓN DEL PARÉNTESIS.

TEORÍA PREVIA: Instrucción paréntesis.

DEFINICIÓN Y SOLUCIÓN.

Veamos cómo podríamos programar el siguiente circuito eléctrico:



Vemos que en el circuito tenemos contactos en serie junto con contactos en paralelo. Ya hemos visto que los contactos en serie se programan con la instrucción “U”, y que los contactos en paralelo se programan con la instrucción “O”. Ahora tenemos que unir ambas instrucciones para formar el circuito que queremos programar.

Para hacer algunas de estas uniones nos hará falta utilizar los paréntesis. Veamos cómo quedaría el circuito resuelto:

SOLUCIÓN EN AWL

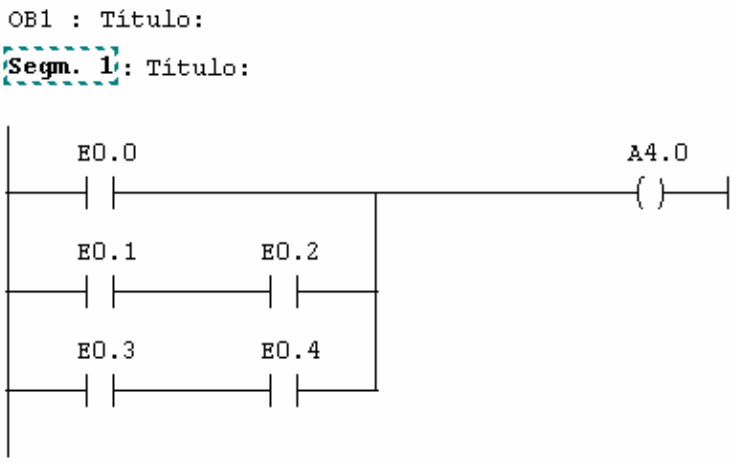
```
U     E     0.0
O(
U     E     0.1
U     E     0.2
)
O
U     E     0.3
U     E     0.4
=     A     4.0
BE
```

Hemos visto dos formas de hacer lo mismo. Vemos que podemos utilizar la instrucción “O(“ o bien podemos utilizar la instrucción “O”, sin abrir y cerrar el paréntesis.

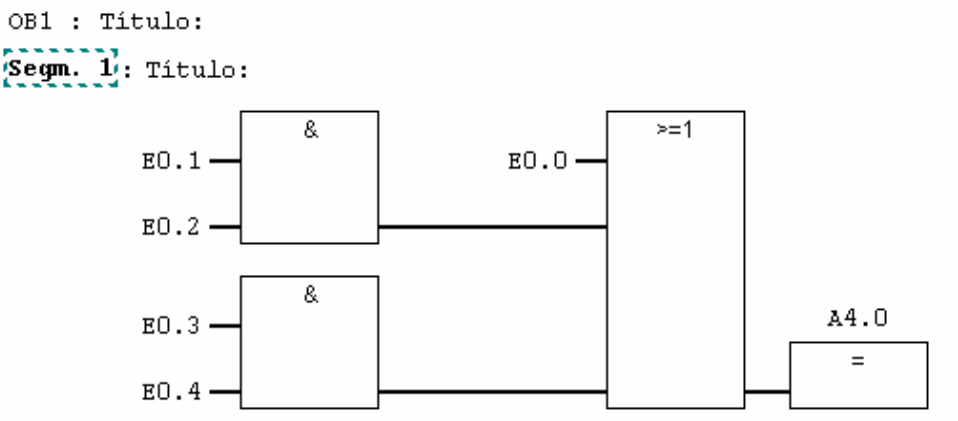
En ambos casos el resultado que obtenemos es el mismo.

Del mismo modo, también podemos utilizar la instrucción del paréntesis para la instrucción “U”.

SOLUCIÓN EN KOP



SOLUCIÓN EN FUP



EJERCICIO 4. CONTACTOS NEGADOS.

TEORÍA

INSTRUCCIONES UN Y ON.

Hemos visto las instrucciones “U” y “O”.

A ambas instrucciones podemos añadirle la letra N a continuación. Se convierten en las instrucciones “UN” y “ON”.

Son las instrucciones negadas de las anteriores.

Por ejemplo si escribimos:

```
UN    E     0.0
UN    E     0.1
.....
```

Esto significa que cuando no esté cerrado el contacto E0.0, y cuando no esté cerrado el contacto E 0.1, ...

Esto nos sirve para programar contactos que son normalmente cerrados.

Veremos a lo largo del curso que la letra N la podemos añadir a más instrucciones. Lo que conseguimos es negar lo que dice la instrucción precedente.

EJERCICIO 4: CONTACTOS NEGADOS

TEORÍA PREVIA: Contacto normalmente cerrado.

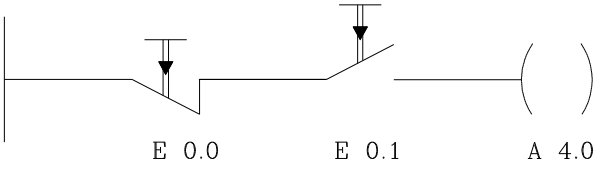
DEFINICIÓN Y SOLUCIÓN.

Vamos a ver cómo podemos programar contactos que son normalmente cerrados y queremos que la actuación sea cuando abrimos el contacto en lugar de cuando lo cerramos.

Para ello utilizaremos las instrucciones “ON” y “UN”.

De esta manera estamos negando la instrucción precedente.

Veamos cómo resolveríamos el siguiente circuito eléctrico. Lo que queremos es 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.



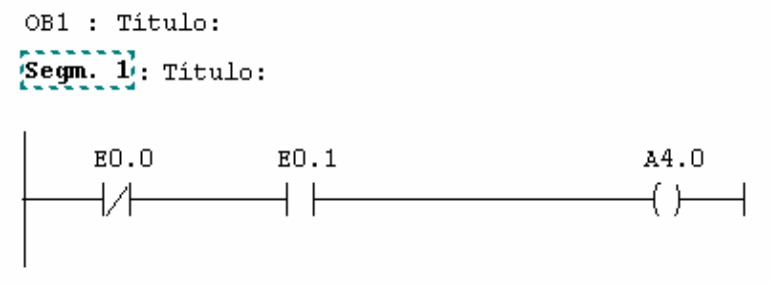
SOLUCIÓN EN AWL

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

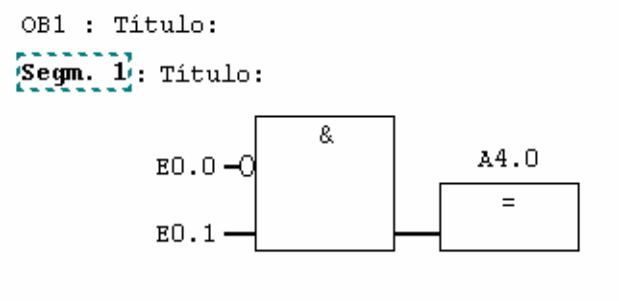
Cuando no esté el contacto E0.0 y si que esté el E0.1, se activará la salida.

En el dibujo, cuando pulsemos en el botón de E 0.0 y cuando pulsemos en el botón de E 0.1, se activará la salida.

SOLUCIÓN EN KOP



SOLUCIÓN EN FUP



EJERCICIO 5: MARCAS.

TEORÍA

DEFINICIÓN DE “MARCA”.

Las marcas son bits internos de la CPU. Disponemos de una cantidad limitada de marcas. Esta cantidad depende de la CPU con la que estemos trabajando.

Estos bits podremos activarlos o desactivarlos como si fueran salidas. En cualquier punto del programa los podremos consultar.

A las marcas les llamaremos M. A continuación tenemos que decir a que bit en concreto nos estamos refiriendo.

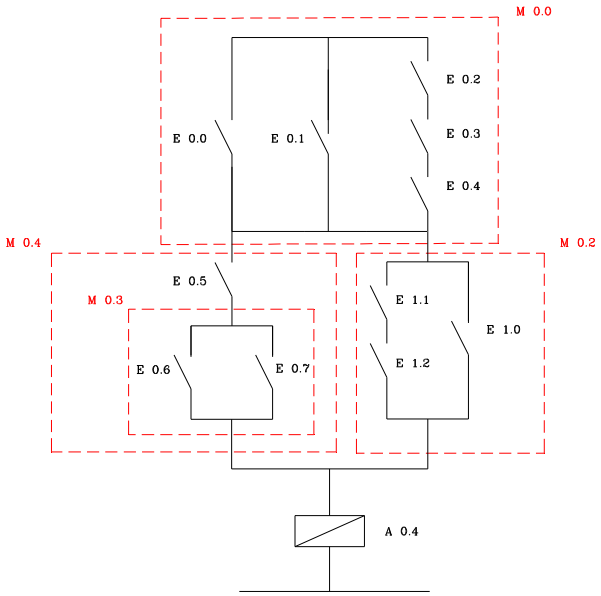
Por ejemplo tenemos las marcas, M 0.0, M 10.7, M 4.5, etc.

EJERCICIO 5: MARCAS.

TEORÍA PREVIA: Introducción a las marcas.

DEFINICIÓN Y SOLUCIÓN.

Veamos cómo podríamos resolver el siguiente circuito eléctrico:



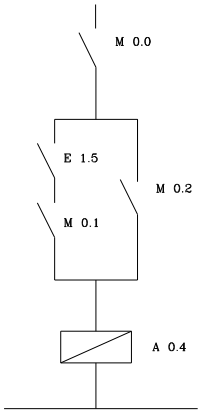
En principio parece que esto es una cosa complicada. Lo podríamos hacer dibujando directamente el circuito en KOP. También lo podemos hacer pensando bien el circuito y con lo visto hasta ahora programarlo a través de paréntesis.

También lo podemos hacer utilizando MARCAS.

Lo que conseguimos utilizando las marcas, es simplificar el circuito todo lo que nosotros queramos. De este modo programamos directamente el AWL de manera sencilla.

Veamos cómo podemos simplificar el circuito utilizando las marcas:

Al final lo que quedaría por programar sería un circuito tan sencillo como este:



Veamos cómo quedaría resuelto con MARCAS:

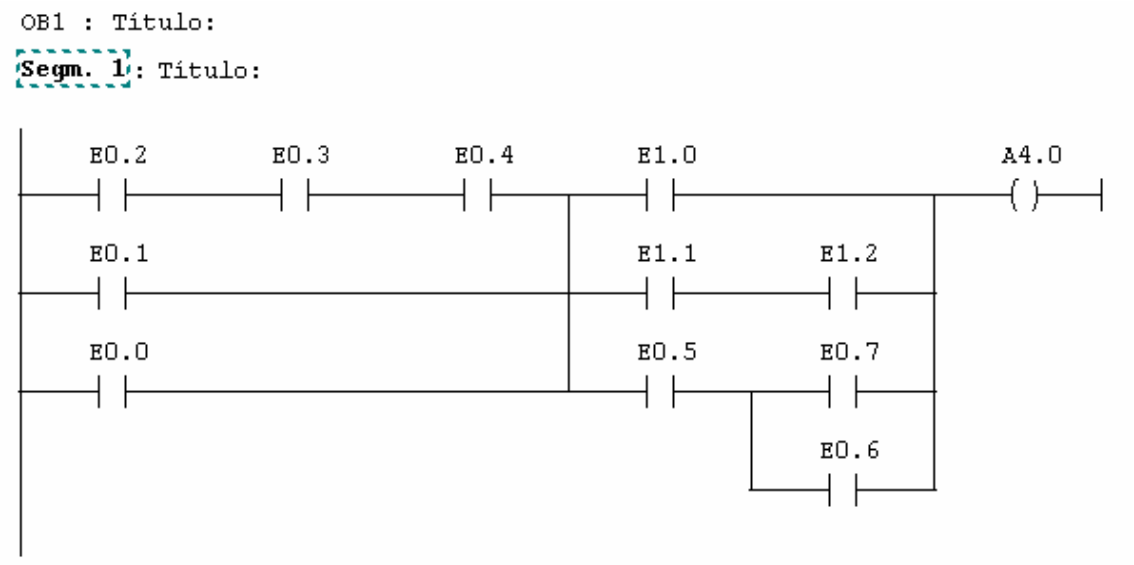
SOLUCIÓN EN AWL

```
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
BE
```

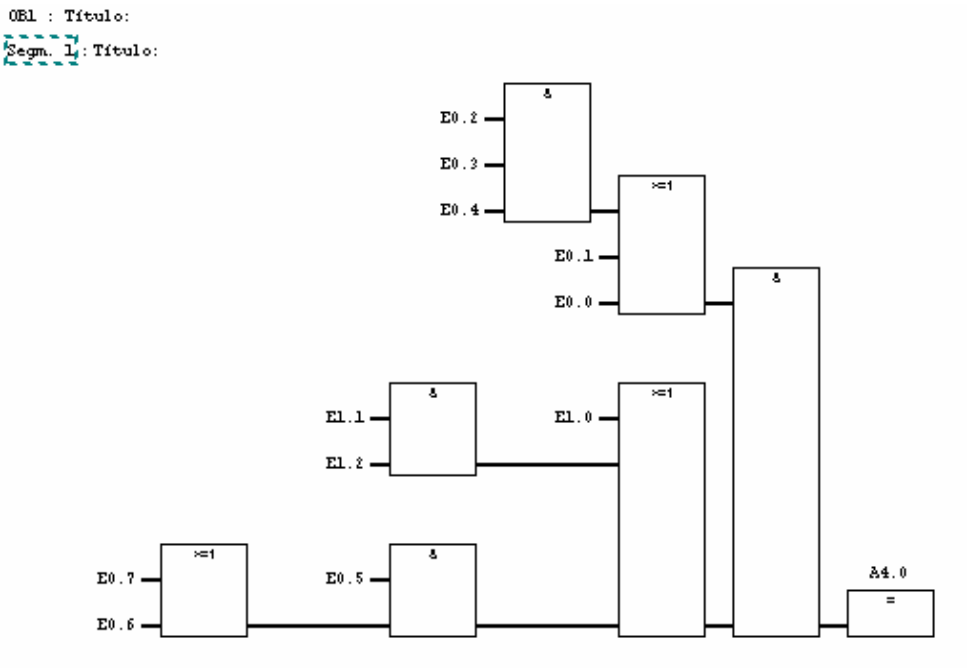
De esta manera, utilizando contactos auxiliares, (marcas) queda resuelto el circuito de manera sencilla.

El ejercicio lo hemos resuelto de modo más largo pero sin tener que pensar mucho.

SOLUCIÓN EN KOP



SOLUCIÓN EN FUP



EJERCICIO 6: INSTRUCCIONES SET Y RESET.

TEORÍA

SIGNIFICADO DE 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 porqué 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.

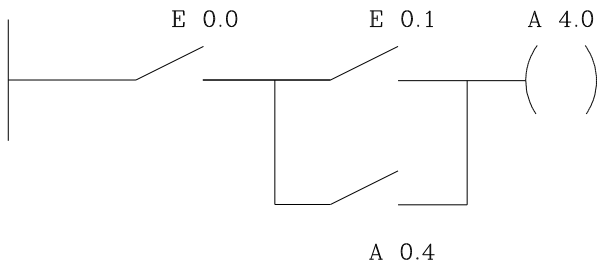
EJERCICIO 6: INSTRUCCIONES SET Y RESET.

TEORÍA PREVIA: Instrucciones SET y RESET. Diferencia con un “igual”.

DEFINICIÓN Y SOLUCIÓN.

Vamos a ver cómo podríamos programar un enclavamiento eléctrico.

El circuito eléctrico correspondiente a un enclavamiento eléctrico sería el siguiente:



Esto lo podemos programar tal cual lo vemos en el circuito eléctrico. Para ello lo haríamos utilizando lo que hemos visto hasta ahora.

También lo podríamos hacer utilizando dos instrucciones nuevas que hacen eso exactamente.

Son las instrucciones “S” y “R”. (Set y Reset)

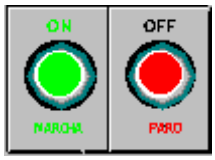
Veamos cómo quedaría el circuito resuelto:

SOLUCIÓN EN AWL

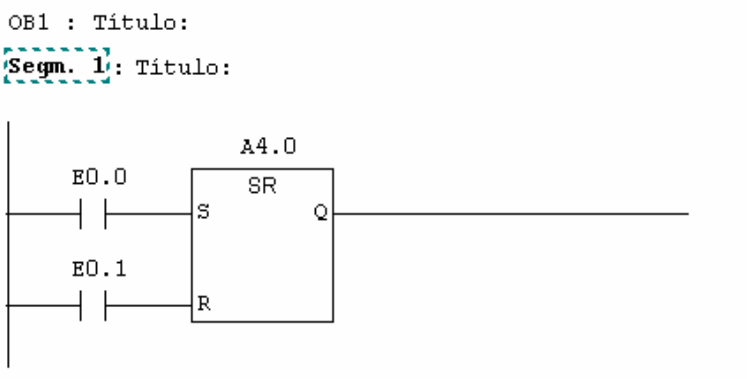
```
U   E   0.0
S   A   4.0
U   E   0.1
R   A   4.0
BE
```

Esto hace las funciones de dos pulsadores, uno de marcha y otro de paro. Es la forma más cómoda de programar dos pulsadores.

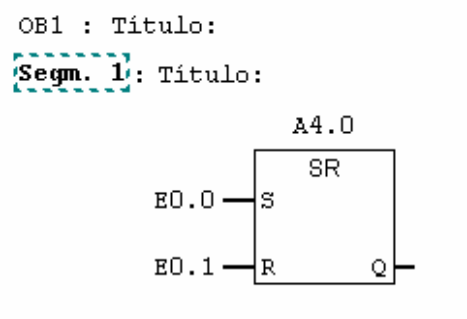
E0.0 E0.1



SOLUCIÓN EN KOP



SOLUCIÓN EN FUP



EJERCICIO 7: OPCIÓN “TEST > OBSERVAR”

TEORÍA

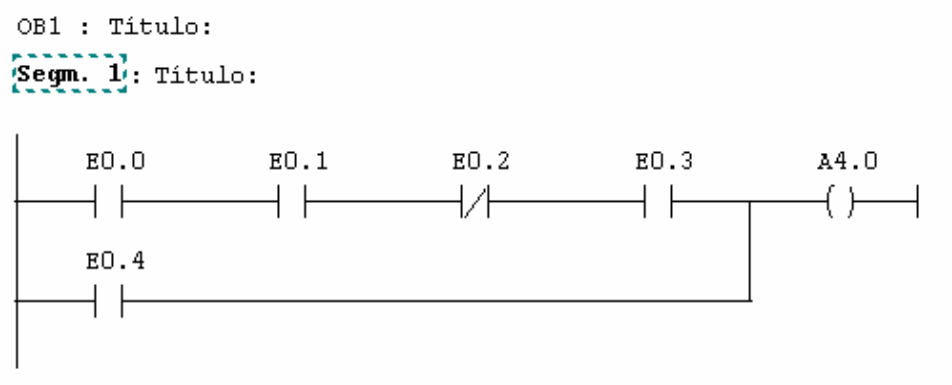
OBSERVAR LA EJECUCIÓN DEL PROGRAMA

Vamos a hacer el siguiente programa:

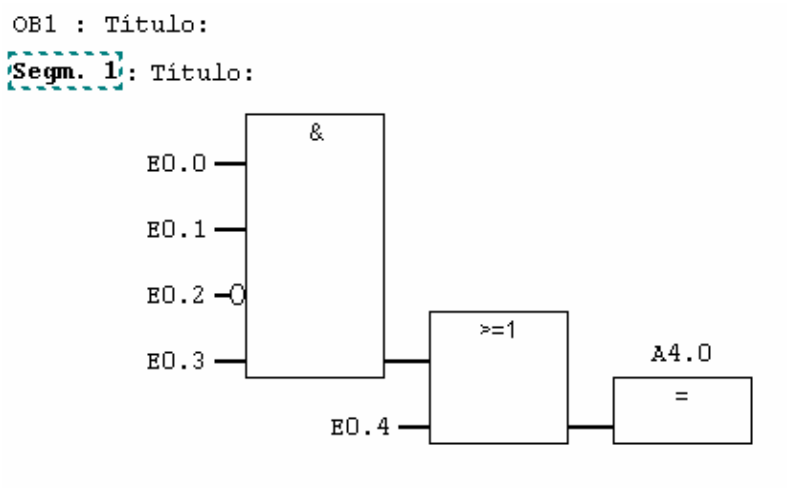
```
U     E     0.0
U     E     0.1
UN    E     0.2
U     E     0.3
O     E     0.4
=     A     4.0
```

Es un programa de un solo segmento que lo podemos ver tanto en AWL como en KOP o en FUP.

PROGRAMA EN KOP



PROGRAMA EN FUP



Vamos a verlo en AWL y vamos a entrar en el menú TEST > OBSERVAR.

También podemos seleccionar esta opción con un icono que representa unas gafas.

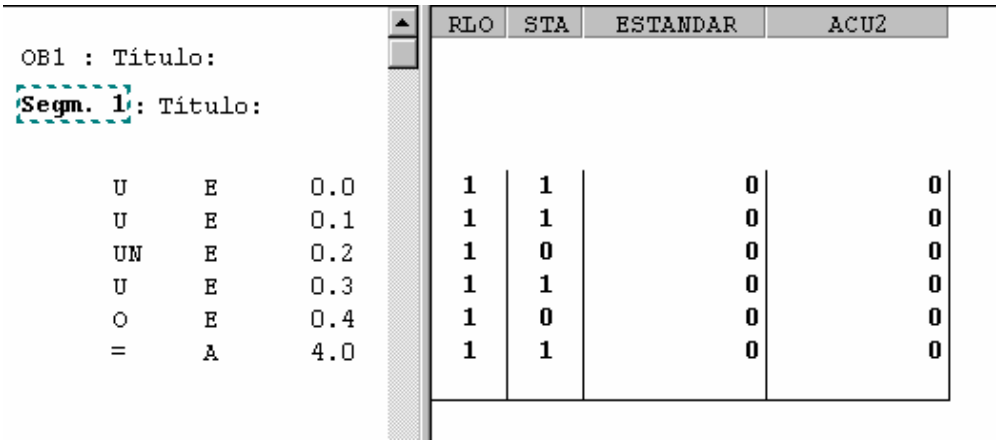
Veremos que aparecen tres columnas al lado de las instrucciones.

Estas columnas las podemos configurar nosotros. Para cambiar la configuración de las columnas, lo podemos hacer entrando en el menú Herramientas > Preferencias. Por defecto vemos tres columnas. Lo que vemos es el estado real del contacto, el RLO (resultado de la operación lógica), y el valor del acumulador 1.

El estado real del contacto será 1 si el contacto en cuestión está cerrado, y será 0 si el contacto está abierto.

El RLO es el resultado de la operación lógica. Comprueba si de ahí hacia arriba se va cumpliendo la operación que le hemos dicho que haga. Si cuando llega a la activación de la salida en el RLO hay 1, la activará. Si hay un 0 no la activará.

Hagamos varias pruebas con el programa ejemplo.



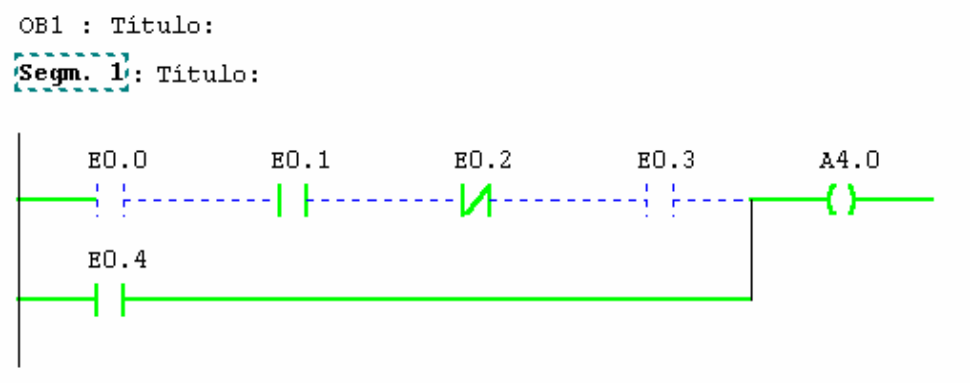
Ahora vamos a ver el programa en KOP y luego en FUP y vamos a ver el mismo menú TEST > OBSERVAR.

Aquí lo que veremos será una línea de color según se van cumpliendo las instrucciones que tenemos escritas.

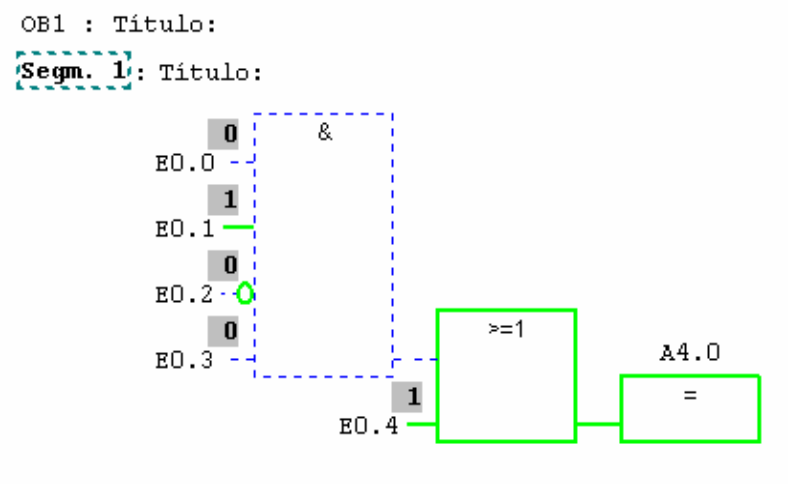
Es lo mismo que hemos visto en AWL pero de modo gráfico.

El significado es el mismo tanto en KOP como en FUP.

KOP


OB1 : Título:
Segm. 1: Título:
E0.0
E0.1
E0.2
E0.3
A4.0
E0.4

FUP


OB1 : Título:
Segm. 1: Título:
&
0
E0.0
1
E0.1
0
E0.2
0
E0.3
1
E0.4
>=1
A4.0
=

EJERCICIO 8: TABLA “OSERVAR / FORZAR VARIABLE”

TEORÍA

OBSERVAR O FORZAR EL VALOR DE CUAIQUIER VARIABLE.

Además de lo que hemos visto en el ejercicio anterior, podemos abrir una tabla en la que podemos observar las variables que nosotros queramos.

Estas tablas son un bloque más dentro del proyecto. Hasta ahora teníamos el OB 1. Si generamos una tabla, tendremos el OB 1 y la VAT 1. Podemos tener más de una tabla. La numeración de las tablas será VAT y a continuación un número. Cuando pinchemos en los bloques del proyecto, veremos las tablas que tenemos.

Para poder abrir la tabla tenemos que estar bien dentro de un bloque de ONLINE/OFFLINE, o bien desde el administrador de SIMATIC pinchando en la parte izquierda encima del nombre de la CPU en ONLINE. En las nuevas versiones lo podemos hacer desde cualquier punto del administrador de SIMATIC.

Vamos al menú “SISTEMA DESTINO” y cogemos la opción de “OBSERVAR / FORZAR VARIABLE”. Tenemos una tabla en la que podemos escribir nosotros las variables con las que queremos trabajar.

Podemos observar bits, bytes, palabras, etc. A continuación podemos decirle en qué formato queremos observar la variable.

A continuación tenemos dos columnas para los valores. En una de ellas vemos el valor actual de las variables, y en la siguiente podemos escribir el valor que queremos que tenga la variable. Esta última columna es para forzar valores.

TABLA DE VALORES

@Tabla de variables1 ONLINE

Operando	Símbolo	Formato de estado	Valor de estado	Valor de forzado
E 0.0	---	BOOL	false	
E 0.1	---	DEC	1	
A 4.0	---	BIN	2#1	
T 1	---	TIEMPO_SIMATIC	S5T#0ms	

Para poder observar y forzar estos valores, tenemos unos botones en la barra de herramientas.

Hay un botón que representa unas gafas con una rayita al lado. Con este botón lo que podemos hacer es una visualización instantánea. Observamos los valores que tienen las variables en ese instante y se quedan fijos en la pantalla. Si se producen cambios en las variables no los vemos reflejados en la pantalla.

Tenemos otro botón que representa unas gafas solamente. Con esto podemos hacer una observación continua. Si se produce algún cambio en las variables, se refleja en la pantalla.

Después tenemos unos botones que representan unos “rayos”. Estos son para forzar variables. Podemos hacer un solo forzado o podemos hacer un forzado continuo.

Una vez forzado un valor, veremos que el valor actual de la variable es el que acabamos de forzar.

Si hacemos un forzado instantáneo y por programa estamos cambiando el valor de la variable, veremos el nuevo valor que ha tomado la variable por programa.

EJERCICIO 9: DEPÓSITO DE AGUA.

TEORÍA PREVIA: Contactos, marcas, set y reset.

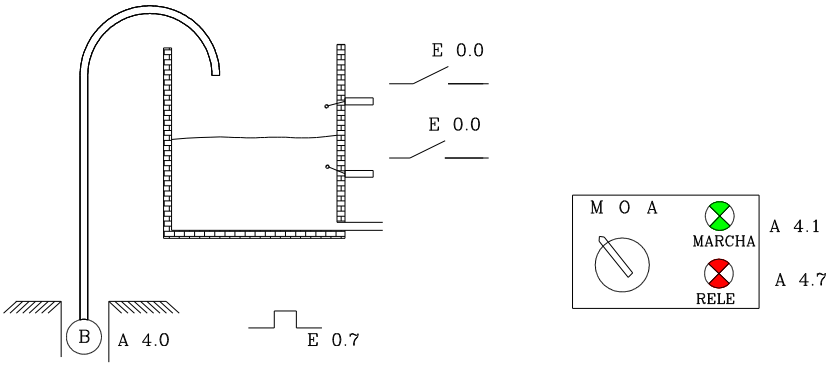
DEFINICIÓN Y SOLUCIÓN.

Tenemos un depósito de agua. Para manejarlo tenemos un selector de mando. Podemos seleccionar modo manual o modo automático. Si seleccionamos modo manual, lo que queremos es que mientras esté conectada, la bomba esté funcionando, y cuando desconectemos que se pare la bomba. No queremos que se haga caso a las boyas de nivel.

Si lo tenemos en modo automático queremos que el nivel se mantenga entre las dos boyas. Cuando el agua llegue al nivel de abajo queremos que se ponga en marcha la bomba, y cuando el agua llegue al nivel de arriba queremos que se pare la bomba.

Además tenemos un relé térmico que actúa tanto cuando tenemos la bomba en funcionamiento manual como cuando la tenemos en funcionamiento automático. Cuando salta el relé, queremos que se pare la bomba y que nos avise con un indicador luminoso en el cuadro de mando.

Además tenemos una luz de marcha que nos indica cuando está en marcha la bomba.



Veamos cómo haríamos la programación del depósito:

SOLUCIÓN EN AWL

Segmento 1: MANUAL

U	E	0.0	//Si activamos en modo manual
=	A	4.0	//Pon en marcha la bomba
=	A	4.1	//Enciende la luz de marcha

Segmento 2: AUTOMÁTICO

U	E	0.1	//Si está en automático
U	E	0.7	//Y está bien el relé
U	E	0.2	//Y está activo el nivel de abajo
UN	E	0.3	//Y no está activo el nivel de arriba
S	A	4.0	//Pon en marcha la bomba
S	A	4.1	//Y enciende la luz de marcha
U	E	0.1	//Si está en automático

```
U     E     0.7          //Y está bien el relé
UN    E     0.2          //Y no está activo el nivel de abajo
U     E     0.3          //Y se ha activado el nivel de arriba
ON    E     0.7          //O ha saltado el relé
R     A     4.O          //Para la bomba
R     A     4.1          //Apaga la luz de marcha
UN    E     0.7          //Si ha saltado el relé
=     A     4.7          //Avísame con la luz de relé
BE
```

Si hacemos la prueba de este circuito veremos que no funciona correctamente. Vemos que en modo manual sí que funciona pero en modo automático no para la bomba cuando debería.

Para resolver este circuito correctamente, nos hace falta utilizar marcas auxiliares. En un mismo bloque no podemos activar la misma salida dos veces con condiciones diferentes porque se interfieren entre ellas.

Las salidas no se activan en el mismo instante en el que se lee la instrucción correspondiente. Existe un registro interno que se denomina PAA (Imagen de proceso de salida), en el que se van almacenando los valores que se tienen que transferir a las salidas cuando finalice el correspondiente ciclo de scan. Cuando se lea la instrucción BE es cuando se mandarán estos valores a las salidas reales. Si hemos enviado varios valores dentro del mismo ciclo de scan, el que realmente llegará a las salidas, será el último que hemos enviado.

El ejercicio bien resuelto quedaría de la siguiente manera:

Segmento 1 : MANUAL

U	E	0.0	//Si está en manual
=	M	0.0	//Activa la marca 0.0
=	M	0.1	//Y activa la marca 0.1

Segmento 2: AUTOMÁTICO

U	E	0.1	//Si está en automático
U	E	0.7	//Y está el relé bien
U	E	0.2	//Y está activo el nivel inferior
UN	E	0.3	//Y no está activo el nivel superior
S	M	0.2	//Activa la marca 0.2
S	M	0.3	//Y activa la marca 0.3
U	E	0.1	//Si está en automático
U	E	0.7	//Y está el relé bien
UN	E	0.2	//Y no está activo el nivel inferior
U	E	0.3	//Y se ha activado el nivel superior
ON	E	0.7	//O ha saltado el relé
R	M	0.2	//Desactiva la marca 0.2
R	M	0.3	//Y desactiva la marca 0.3
UN	E	0.7	//Si no está el relé
=	A	4.7	//Activa la luz de relé.

Ahora nos quedaría asignar las marcas a las salidas.

Añadimos:

```
U     M     0.0          //Si está activa la marca 0.0
O     M     0.2          //O está activa la marca 0.2
=     A     4.0          //Pon en marcha la bomba
U     M     0.1          //Si está activa la marca 0.1
O     M     0.3          //O la marca 0.3
=     A     4.1          //Enciende la luz de marcha
```

Ahora ya no funciona el térmico en el modo manual. Al utilizar marcas diferentes para cada tipo de funcionamiento, el térmico sólo actúa sobre las marcas de modo automático. Sólo estamos haciendo un reset de una de las marcas que activan la bomba. Nos falta resetear la otra marca. Tendremos que añadir las siguientes líneas.

```
UN    E     0.7          //Si ha saltado el relé
R     M     0.0          //Desactiva la marca 0.0
R     M     0.1          //Y desactiva la marca 0.1
BE
```

Otra posible solución sería programar el paro de la bomba por apertura del relé térmico en el último segmento. Los paros de emergencia se suelen programar al final.

Ahora podemos hacer todas las objeciones que queramos y corregir sobre lo que ya tenemos hecho.

Por ejemplo, puedo querer asegurarme que cuando se pone en marcha en modo manual no está a la vez en modo automático. Puedo suponer que por error se pueden dar las dos circunstancias a la vez y quiero evitar ese error.

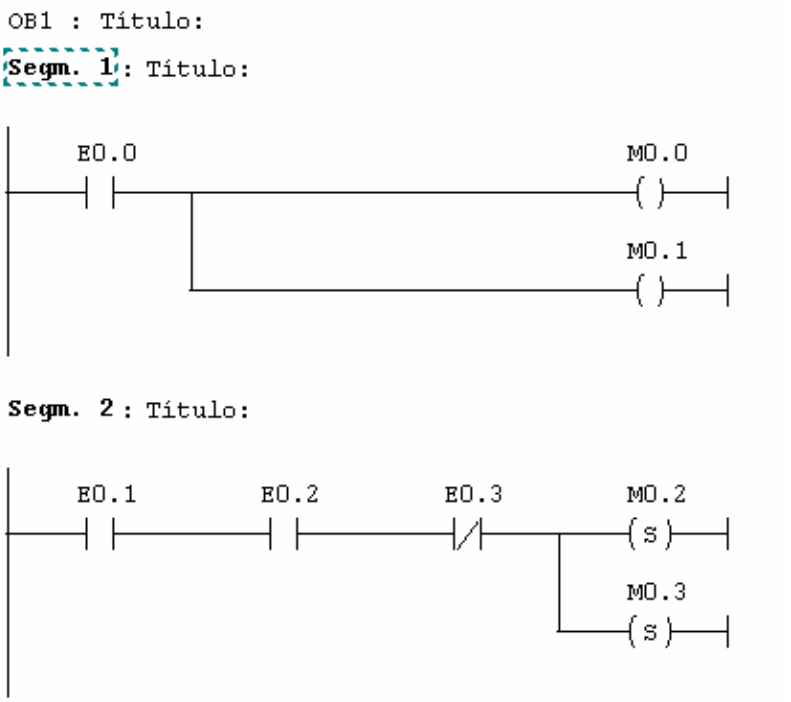
Añado las instrucciones pertinentes.

U E 0.0

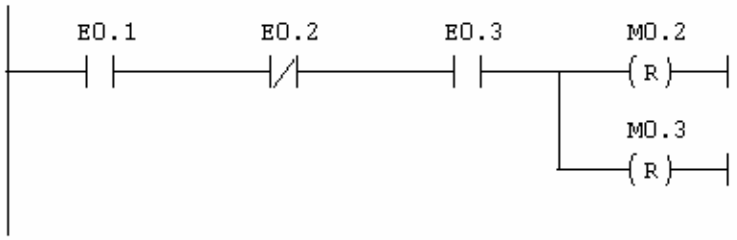
UN E 0.1

.....

KOP



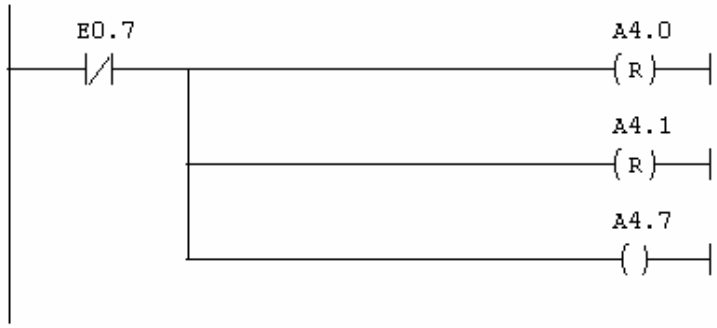
Segm. 3 : Título:



Segm. 4 : Título:

Segm. 5 : Título:

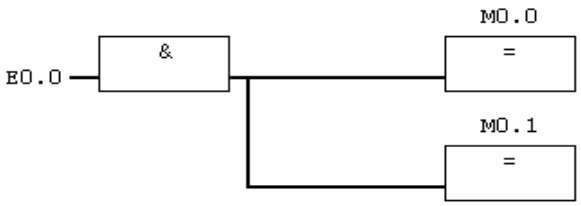
Segm. 6 : Título:



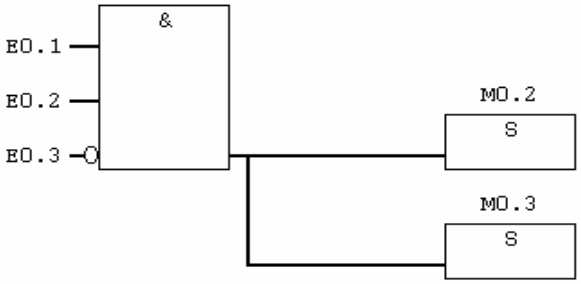
FUP

OB1 : Título:

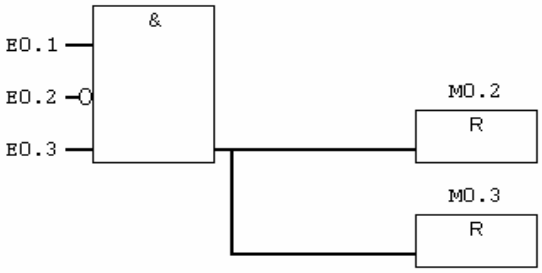
Segm. 1: Título:



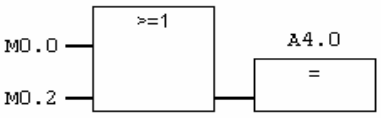
Segm. 2 : Título:

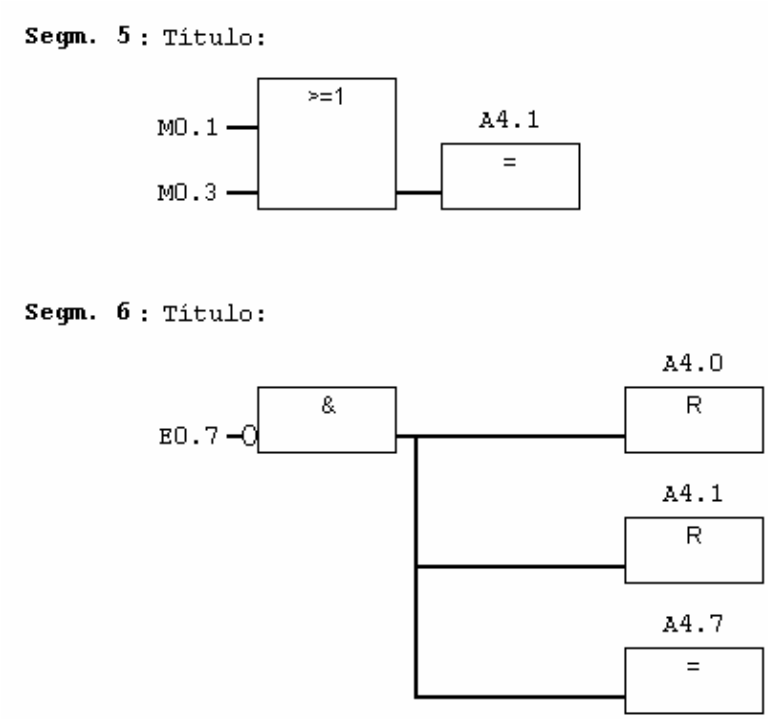


Segm. 3 : Título:



Segm. 4 : Título:





Ahora ya tenemos la base del programa. Podemos añadir todo lo que creamos que sea necesario o conveniente. Por ejemplo, en este caso no he tenido

en cuenta la situación de que después de haber estado en manual o en automático, volvamos a la posición de reposo. En automático he hecho sets a ciertas marcas. Cuando volvamos a la posición de reposo esas marcas tendrán que volver a cero. De lo contrario podría darse el caso de que estando en la posición de reposo, tengamos la bomba en marcha. Para remediar esto podría añadir las siguientes instrucciones:

```
UN    E    0.0
UN    E    0.1
R     A    4.0
R     A    4.1
```

EJERCICIO 10. SEMÁFORO

TEORÍA

TEMPORIZADORES “SE” Y “SI”

Temporizadores sin memoria: Tenemos los temporizadores “SE” y ”SI”.

Analicemos cada uno de ellos.

Temporizador “SE”: Es un temporizador de retardo a la conexión. Para programar el temporizador, necesitamos cinco operaciones como mínimo.

1ª Necesitamos una condición a partir de la cual empiece a temporizar. Esta condición puede constar de una sola instrucción o de varias.

2ª Necesitamos decirle cuanto tiempo tiene que temporizar.

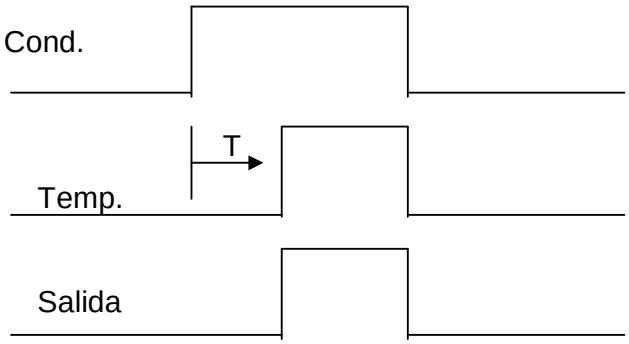
3ª Necesitamos decirle el modo de funcionamiento y nº de temporizador que queremos utilizar. (En cada CPU tenemos una cantidad de temporizadores)

4º Queremos que en algún momento dado, (mientras temporiza, cuando ha acabado de temporizar, etc.)

5º haga algo.

Alguna de estas operaciones, puede constar de más de una instrucción.

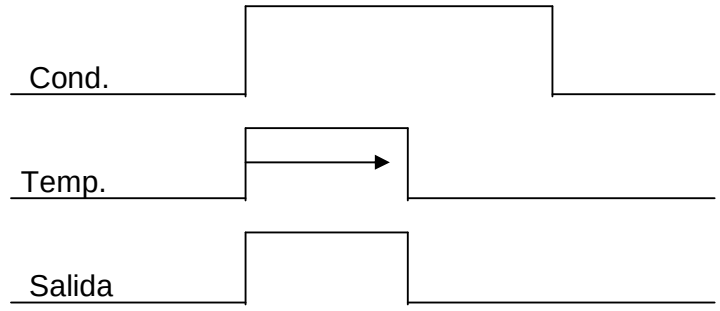
El modo de funcionamiento SE es el siguiente:




```
U   E   0.0
L   S5T#5S
SE  T   1
U   T   1
=   A   4.0
```

Además de lo que hemos visto, en cualquier momento podemos hacer un RESET del temporizador. Para hacer un RESET necesitamos una condición. En el momento se cumpla si al temporizador le correspondía estar a 1, automáticamente se pondrá a cero aunque por su modo de funcionamiento no le corresponda.

El modo de funcionamiento SI es el siguiente:

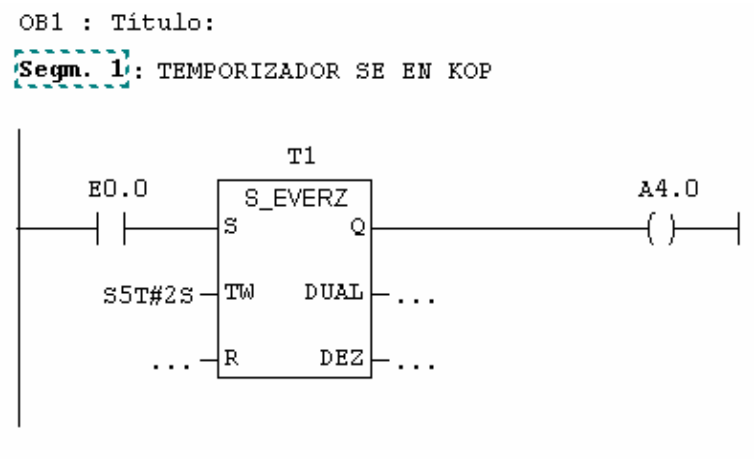



```
U   E   0.0
L   S5T#5S
SI  T   1
U   T   1
=   A   4.0
```

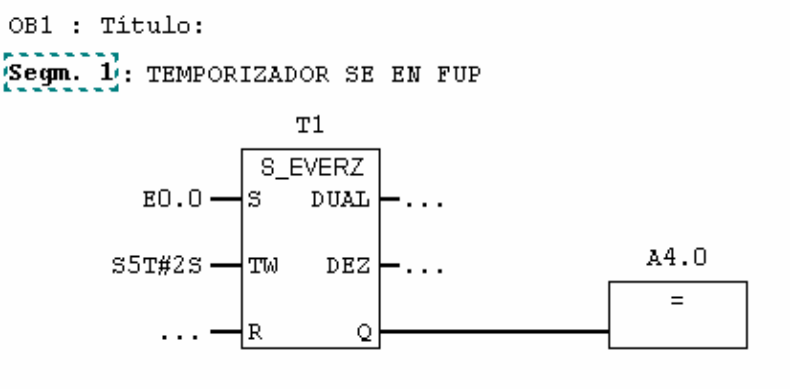
A este temporizador también podemos añadirle un RESET en cualquier momento.

Veamos como podríamos programar estos dos temporizadores en KOP y en FUP respectivamente.

KOP



FUP

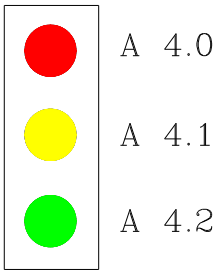


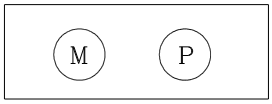
EJERCICIO 10: SEMÁFORO

TEORÍA PREVIA: Temporizadores SE y SI.

DEFINICIÓN Y SOLUCIÓN

Tenemos un semáforo con las tres luces verde, amarillo y rojo. Tenemos dos pulsadores de mando: un pulsador de marcha y un pulsador de paro.





E 1.0 E 1.1

Con el pulsador de marcha quiero que comience el ciclo. El ciclo de funcionamiento es el siguiente:

1º/ Verde durante 5 seg.

2º/ Verde + Amarillo durante 2 seg.

3º/ Rojo durante 6 seg.

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.

Veamos cómo quedaría el ejercicio resuelto:

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
```

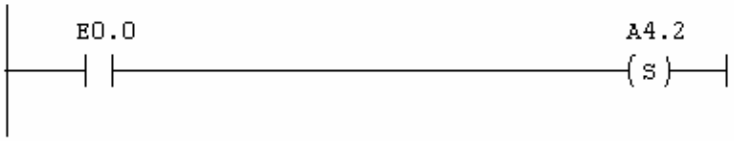
```
R    A    4.0        //Apaga el rojo
R    A    4.1        //Apaga el amarillo
R    A    4.2        //Apaga el verde
BE
```

Veamos como quedaría resuelto el circuito en KOP y en FUP.

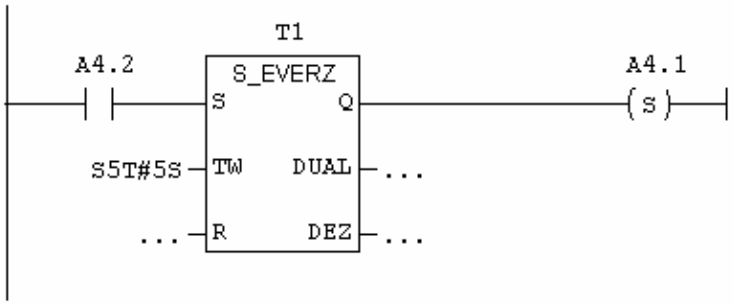
KOP

OB1 : SEMAFORO

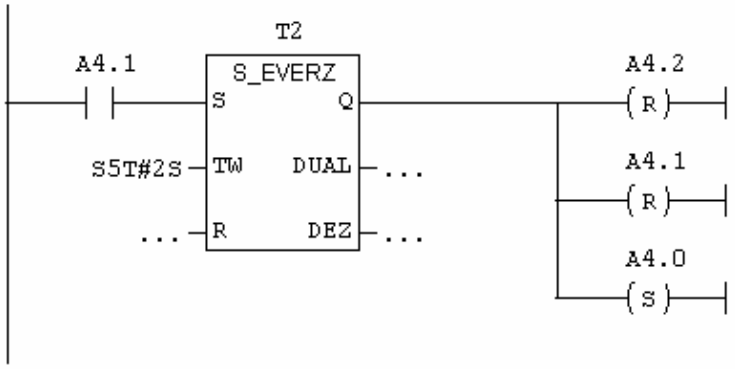
Segm. 1 : ENCENDER EL VERDE



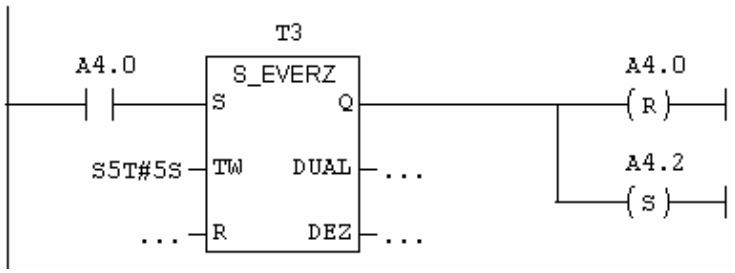
Segm. 2 : A LOS 5 SEGUNDOS ENCENDER EL AMARILLO



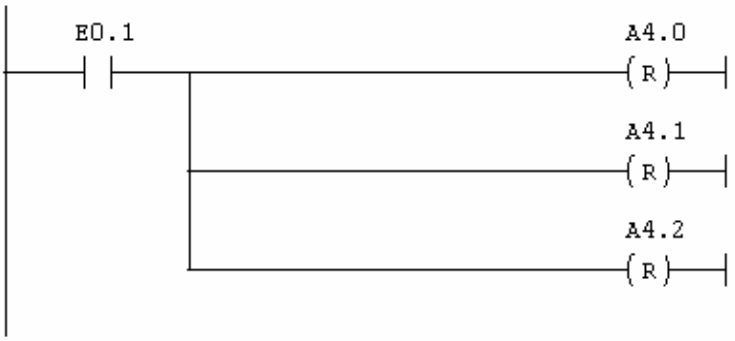
Segm. 3 : A LOS 2 S. APAGAR AMARILLO Y VERDE Y ENCENDER ROJO



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



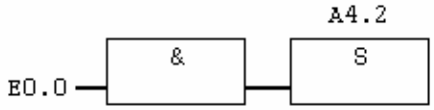
Segm. 5: SI SE PULSA PARA APAGAR TODO



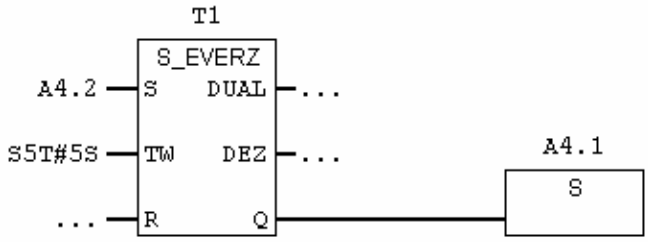
FUP

OB1 : SEMAFORO

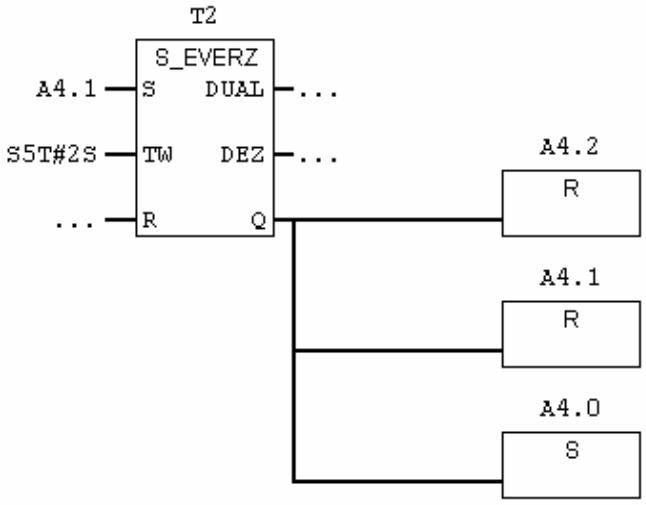
Segm. 1 : ENCENDER EL VERDE



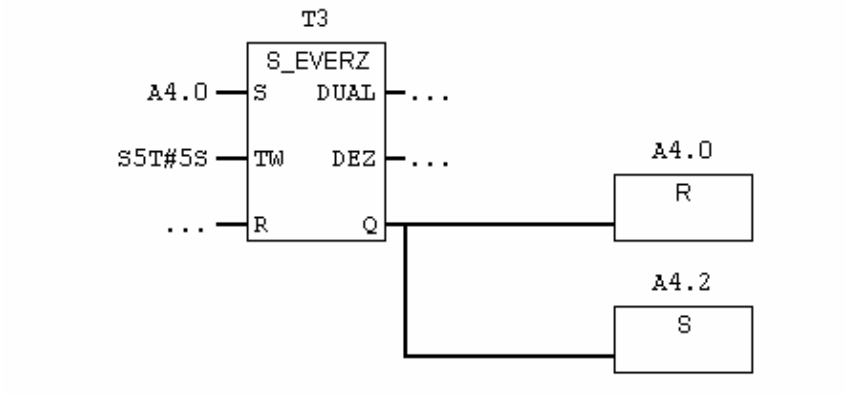
Segm. 2 : A LOS 5 SEGUNDOS ENCENDER EL AMARILLO



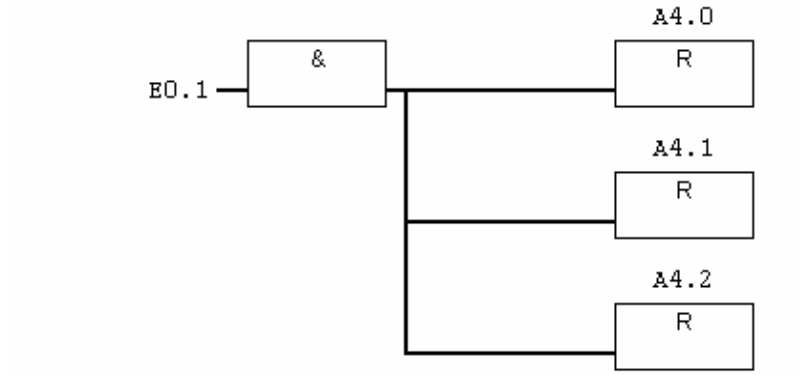
Segm. 3 : A LOS 2 S. APAGAR AMARILLO Y VERDE Y ENCENDER ROJO



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



Segm. 5 : SI SE PULSA PARA APAGAR TODO



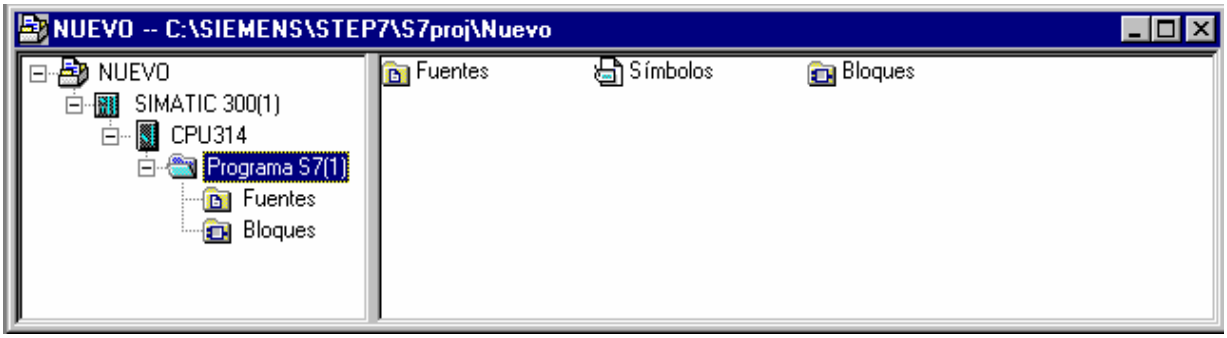
EJERCICIO 11: DIRECCIONAMIENTO SIMBÓLICO GLOBAL

TEORÍA

INSERTAR SÍMBOLOS

Hasta ahora hemos llamado a cada contacto por su nombre. Dependiendo de si es entrada, salida o marca, tienen unos nombres predefinidos (E, A, M,...). Veremos que lo mismo ocurre con los temporizadores, contadores, DB, etc.

Pero nosotros podemos dar nombre a todo esto. Para ello vamos a la ventana del administrador de SIMATIC, y pinchamos en la ventana de OFLINE, en la parte izquierda, encima de donde pone programa S7. En la parte derecha aparece un icono que se llama “Símbolos”.



Hacemos doble clic encima de “Símbolos”. Entramos en una tabla donde podemos definir los nombres que queramos y decir a qué contacto corresponde cada nombre.

Podemos poner nombre a todo lo que queramos. Tenemos que tener en cuenta que el programa diferencia las mayúsculas de las minúsculas. Si luego intentamos acceder a uno de estos contactos por su nombre, tendremos que escribir el nombre tal y como lo hemos definido diferenciando las mayúsculas de las minúsculas.

Los nombres que definamos aquí son de ámbito global. Los podremos utilizar en cualquier bloque del programa.

Al escribirlos en el programa, sabremos que son de ámbito global porque aparecerán escritos entre comillas.

Nosotros escribiremos el nombre que hemos definido, y veremos que el programa le añade unas comillas. Esto nos indica que es un símbolo global. Luego veremos que también podemos tener símbolos locales.

A la hora de ver el programa en AWL, KOP o FUP, podremos ver o no estos símbolos.

Tenemos dentro del menú VER > MOSTRAR > REPRESENTACIÓN SIMBÓLICA para ver o no los símbolos. También tenemos la opción “información sobre el símbolo” para ver a qué contacto corresponde cada uno de los símbolos.

Nosotros podremos acceder a estos contactos por su nombre en cualquier sitio del programa.

Veamos unos pequeños ejemplos en cada uno de los lenguajes.

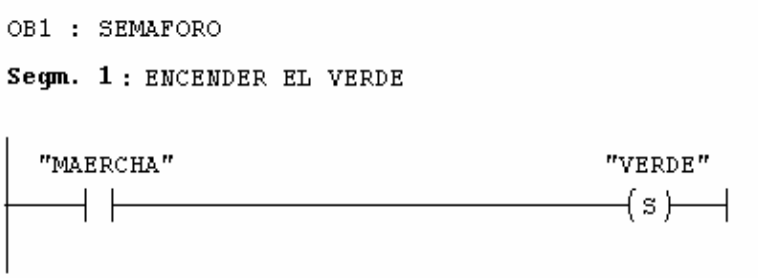
AWL

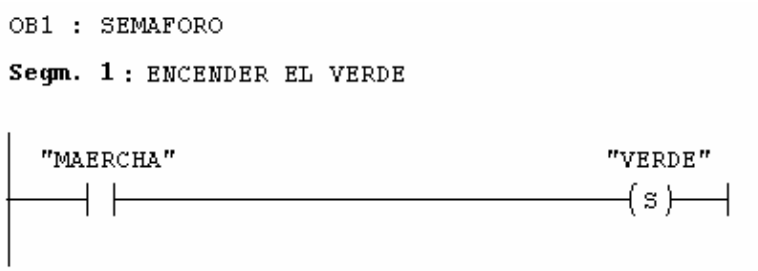
```
OB1 : SEMAFORO
Segm. 1 : ENCENDER EL VERDE
      U     "MAERCHA"
      S     "VERDE"
```

```
OB1 : SEMAFORO
Segm. 1 : ENCENDER EL VERDE
      U     "MAERCHA"                E0.0
      S     "VERDE"                  A4.2
```

```
OB1 : SEMAFORO
Segm. 1 : ENCENDER EL VERDE
      U     E      0.0                    "MAERCHA"
      S     A      4.2                    "VERDE"
```

KOP



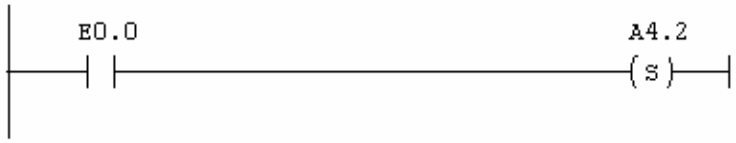


Información del símbolo:

E0.0	MAERCHA
A4.2	VERDE

OB1 : SEMAFORO

Segm. 1 : ENCENDER EL VERDE



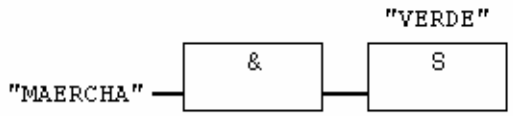
Información del símbolo:

E0.0	MAERCHA
A4.2	VERDE

FUP

OB1 : SEMAFORO

Segm. 1 : ENCENDER EL VERDE



OB1 : SEMAFORO

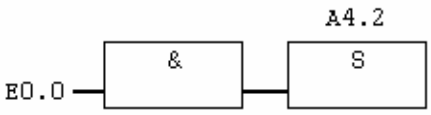
Segm. 1 : ENCENDER EL VERDE

Información del símbolo:

E0.0	MAERCHA
A4.2	VERDE

OB1 : SEMAFORO

Segm. 1 : ENCENDER EL VERDE



Información del símbolo:

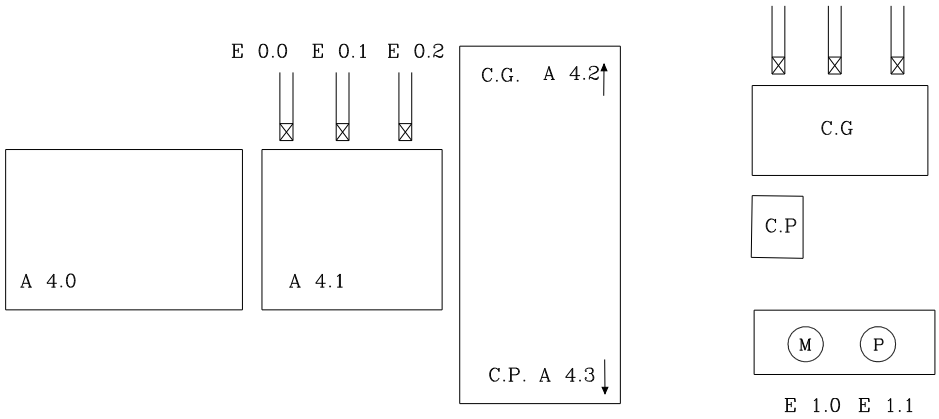
E0.0	MAERCHA
A4.2	VERDE

EJERCICIO 12: CINTAS TRANSPORTADORAS

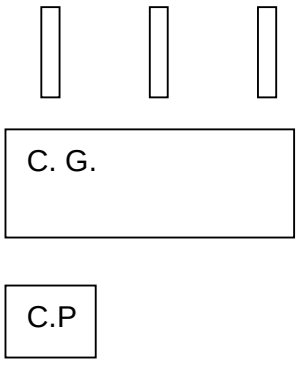
TEORÍA PREVIA: Temporizadores SE y SI.

DEFINICIÓN Y SOLUCIÓN

Tenemos tres cintas transportadoras dispuestas de la siguiente manera:



Por las cintas transportadoras van a circular cajas grandes y pequeñas indistintamente. El tamaño de las cajas con respecto a las células que tenemos en la segunda cinta es el siguiente:



El funcionamiento que queremos es 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 nº2. 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 EN AWL

```
U   E   1.0    //Si le damos al pulsador de marcha
S   A   4.0    //Pon en marcha la primera cinta
U   E   0.0    //Cuando la caja cambie de cinta
S   A   4.1    //Pon en marcha la segunda cinta
R   A   4.0    //y para la primera
U   E   0.0    //Si ve la primera célula
U   E   0.1    //Y ve la segunda célula
U   E   0.2    //Y ve la tercera célula
S   A   4.2    //Pon en marcha la cinta de caja grande
UN  E   0.0    //Si no ve la primera célula
U   E   0.1    //Y si que ve la segunda célula
UN  E   0.2    //Y no ve la tercera célula
S   A   4.3    //Pon en marcha la cinta de caja pequeña
UN  E   0.0    //Si no ve la primera célula
UN  E   0.1    //Y no ve la segunda célula
U   E   0.2    //Y si que ve la tercera célula
```

```
R     A     4.1          //Para la segunda cinta
U     A     4.2          //Si está en marcha la cinta de caja grande
O     A     4.3          //O la cinta de caja pequeña
L     S5T#10S            //Cuanta 10 segundos
SE    T     1            //Con el T 1
U     T     1            //Y cuando acabes de contar
R     A     4.2          //Para la cinta de caja grande
R     A     4.3          //Para la cinta de caja pequeña
S     A     4.0          //Y pon en marcha la primera cinta
U     E     1.1          //Si pulsamos el paro de emergencia
R     A     4.0          //Para la primera cinta
R     A     4.1          //Para la segunda cinta
R     A     4.2          //Para la cinta de caja grande
R     A     4.3          //Para la cinta de caja pequeña
BE
```

De este modo, si se va la luz en un momento determinado, al volver el ciclo empezaría otra vez desde cero.

Si se hubiera quedado una caja en el camino, no lo sabríamos y tendríamos un ciclo con dos cajas circulando. Es una cosa que no hemos tenido en cuenta. El sistema podría reaccionar erróneamente.

Para arreglar esto podemos utilizar marcas remanentes en lugar de trabajar directamente con las salidas.

De este modo el sistema se “acordaría” de las cajas que estaban circulando antes del corte de suministro.

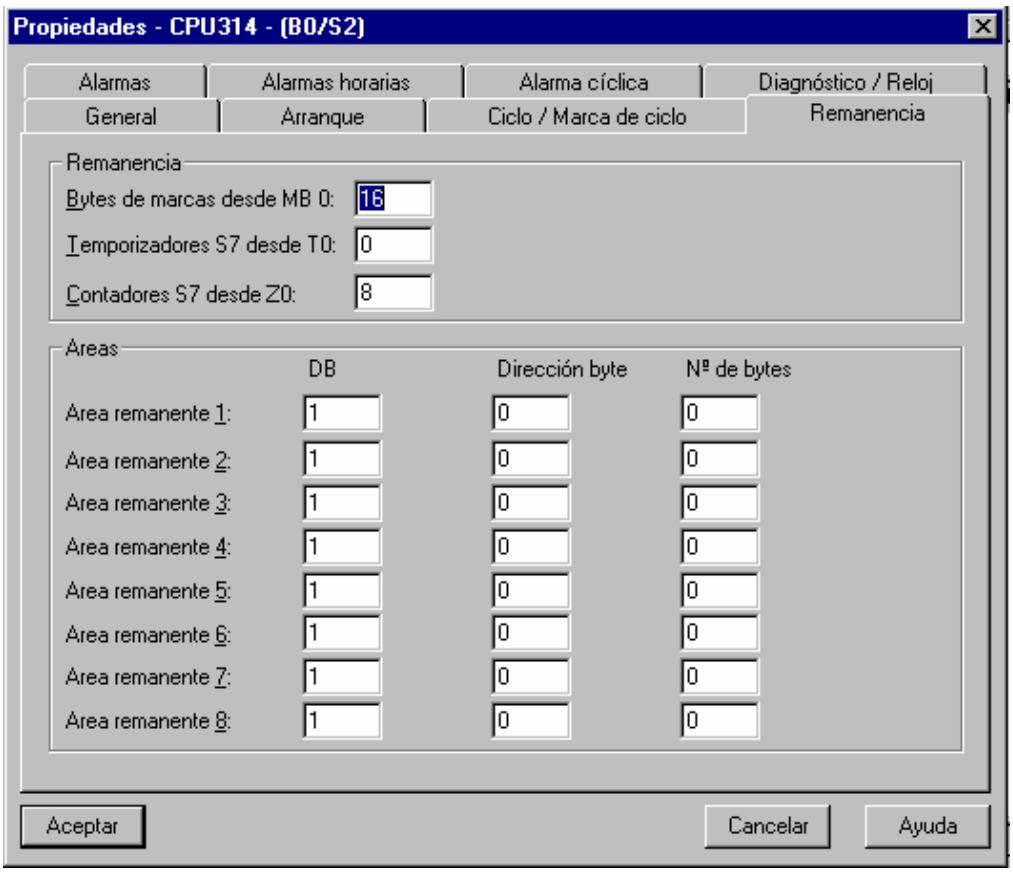
Cambiaríamos las salidas por marcas remanentes y añadiríamos:

```
U    M    0.0
=    A    4.0
U    M    0.1
=    A    4.1
U    M    0.2
=    A    4.2
U    M    0.3
=    A    4.3
```

Marcas remanentes son aquellas que ante un corte de tensión mantienen su valor. Por defecto, tenemos los primeros 16 bytes de marcas remanentes.

No obstante, la cantidad de marcas remanentes que queremos las definimos nosotros. Para ello, vamos al administrador de SIMATIC. Pinchamos encima de “Equipo 300”. En la parte derecha aparece el icono del Hardware. Entramos en el hardware. Una vez dentro, pinchamos con el botón derecho encima de la CPU. Entramos en el menú “Propiedades del objeto”.

Veremos que aparecen unas fichas. Una de ellas se llama “Remanencia”. Entramos en esta ficha. Vemos que podemos definir la remanencia de las marcas como nosotros queramos. Le decimos la cantidad de bytes remanentes de marcas que queremos.



Este mismo programa también lo podríamos hacer con direccionamiento simbólico.

Si utilizamos el direccionamiento simbólico podríamos hacer lo siguiente:

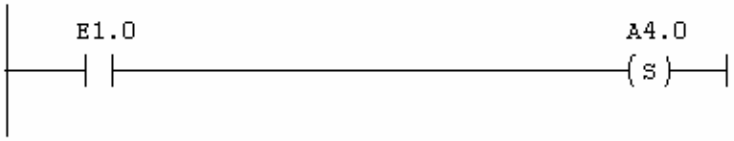
U “MARCHA”
S “1ª_CINTA”
U “CELULA_1”
S “2ª_CINTA”
R “1ª_CINTA”
.....
.....

Veamos como quedaría el ejercicio resuelto en KOP y en FUP.

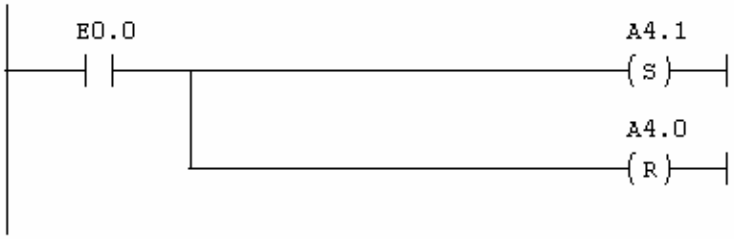
KOP

OB1 : SEMAFORO

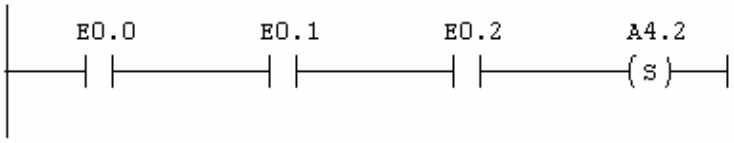
Segm. 1 : Título:



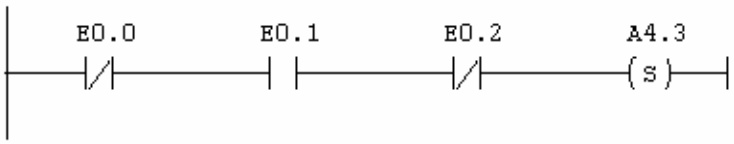
Segm. 2 : Título:



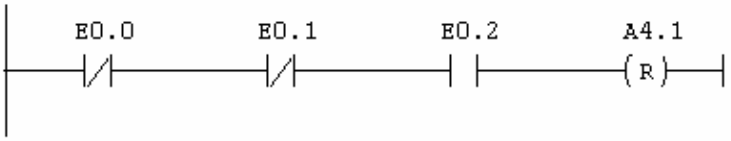
Segm. 3 : Título:



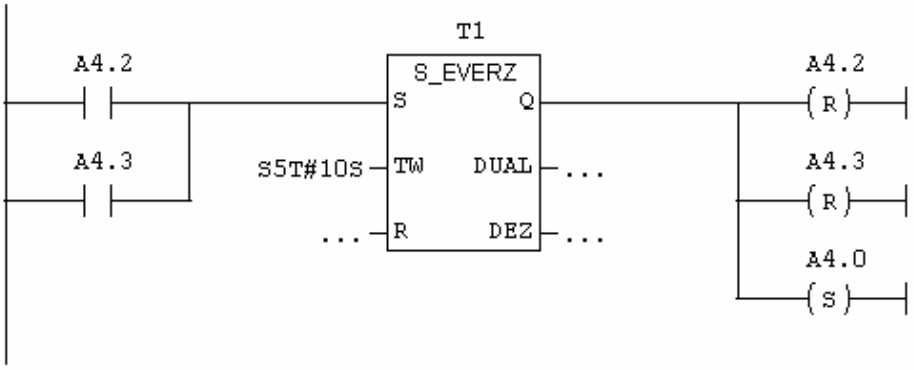
Segm. 4 : Título:



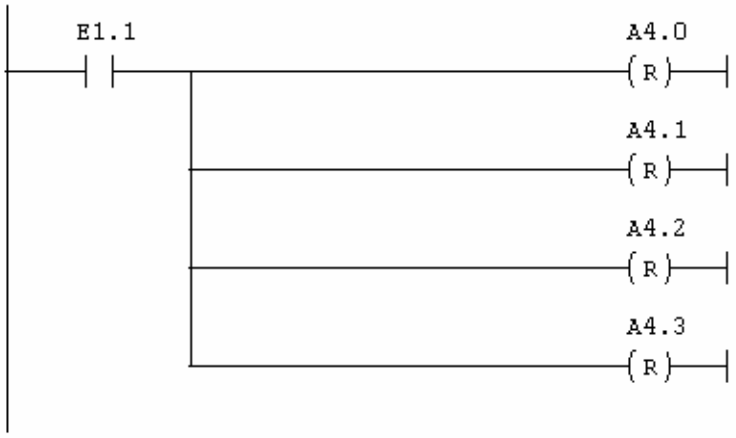
Segm. 5 : Título:



Segm. 6 : Título:



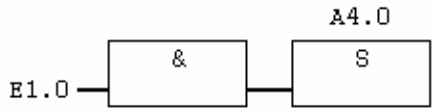
Segm. 7 : Título:



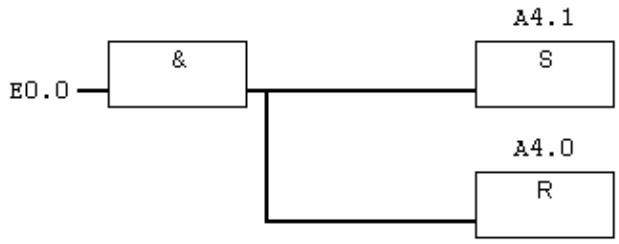
FUP

OB1 : SEMAFORO

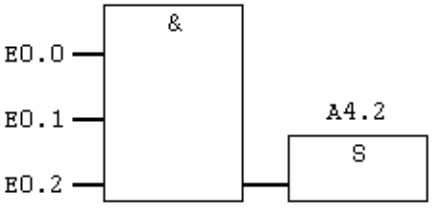
Segm. 1 : Título:



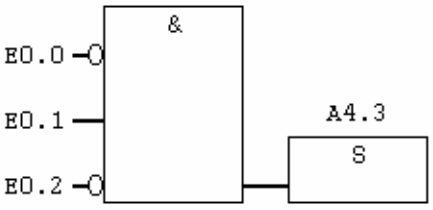
Segm. 2 : Título:



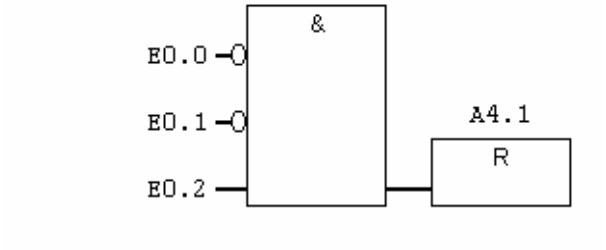
Segm. 3 : Título:



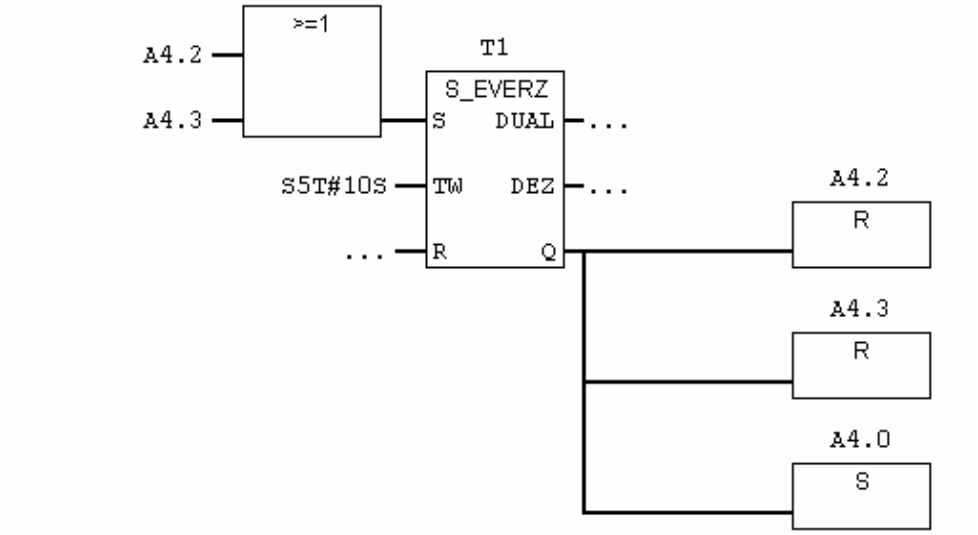
Segm. 4 : Título:

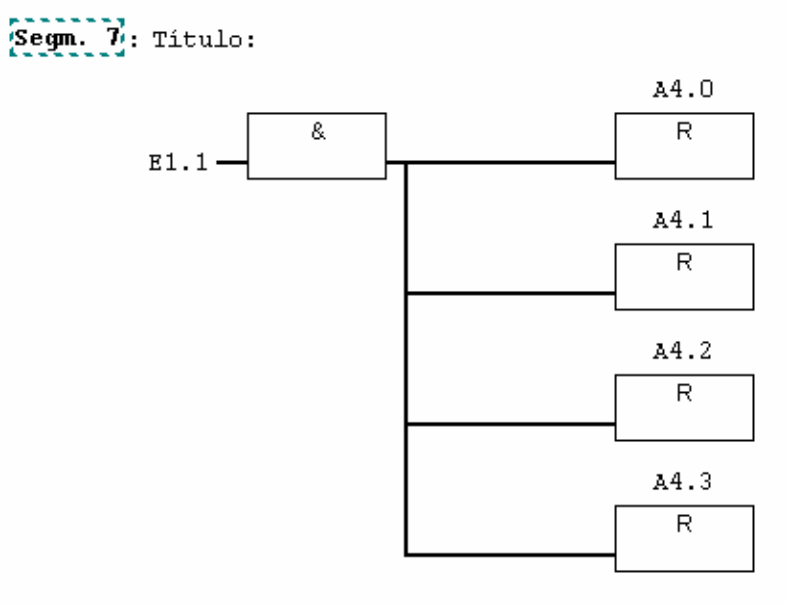


Segm. 5 : Título:

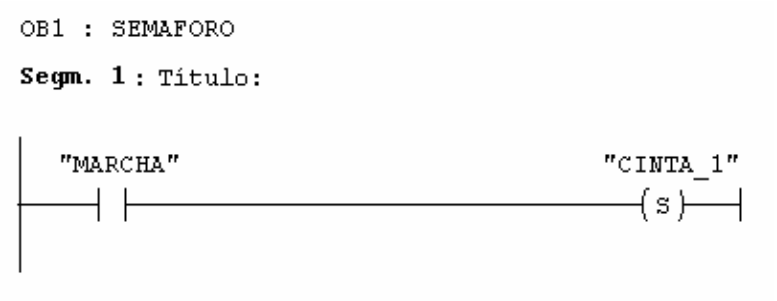


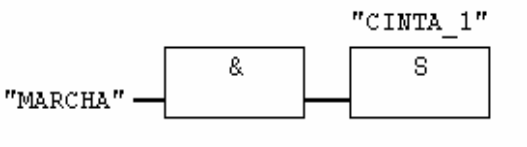
Segm. 6 : Título:





Tanto en KOP como en FUP, también podemos utilizar el direccionamiento simbólico. El programa (a modo de ejemplo) quedaría de la siguiente manera:





EJERCICIO 13. INTERMITENTE

TEORÍA

FINALES EN STEP 7

A parte del final que hemos visto (BE), existen otros dos finales. Estos son BEB y BEA. Veamos para qué podemos utilizar cada uno de ellos.

BEB: Es un final condicional. Esto quiere decir que será un final o no dependiendo de si se cumple o no la condición (RLO) que tenemos antes del BEB.

Si la condición se cumple, será un final de programa. Si la condición no se cumple, no será un final.

BEA: Es un final absoluto. Siempre que se lea la instrucción BEA terminará el programa. La diferencia con el BE es que podemos escribir detrás de él.

Veamos un ejemplo en el que podemos utilizar los BEA.

Supongamos que queremos el siguiente funcionamiento:

Si está activa la entrada E 0.0 queremos que funcione un trozo de programa. Si está activa la entrada E 0.1 queremos que funcione otro trozo de programa. Si no está activa ninguna de las dos, no queremos que funcione nada.

Esto lo programaríamos del siguiente modo:

```
        U       E       0.0
        Salta a meta 1
        U       E       0.1
        Salta a meta 2
        BEA
Meta1:.............
        .................
        .................
        BEA
Meta2:.............
        .................
        .............
        BE
```

Si no tuviésemos el primer BEA, aunque no estuviera ni la E 0.0 ni la E 0.1 se ejecutaría la primera meta. Si el PLC no encuentra una instrucción de fin, va ejecutando una instrucción detrás de otra.

El BEA es una instrucción incondicional. Cada vez que el PLC la lea va a terminar el programa.

Veamos un ejemplo de como funcionaría el BEB.

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

```
U     E     0.2
=     A     4.2
BE
```

Si no está activa la E 0.1 funcionaría todo el programa. Si está activa la E 0.1 sólo funcionaría la primera parte del programa.

En este caso no tenemos operación equivalente en KOP ni en FUP. La misma función la podríamos desarrollar utilizando metas. (Ver ejercicio más adelante)

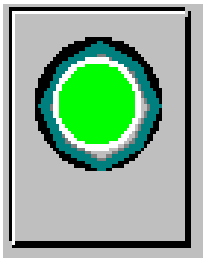
EJERCICIO 13: INTERMITENTE

TEORÍA PREVIA: Finales BEB y BEA

DEFINICIÓN Y SOLUCIÓN

.

Vamos a hacer un intermitente utilizando un solo temporizador de 1 segundo. Queremos que una salida esté activa un segundo y no activa otro segundo. Queremos que haga esto sin ninguna condición previa.



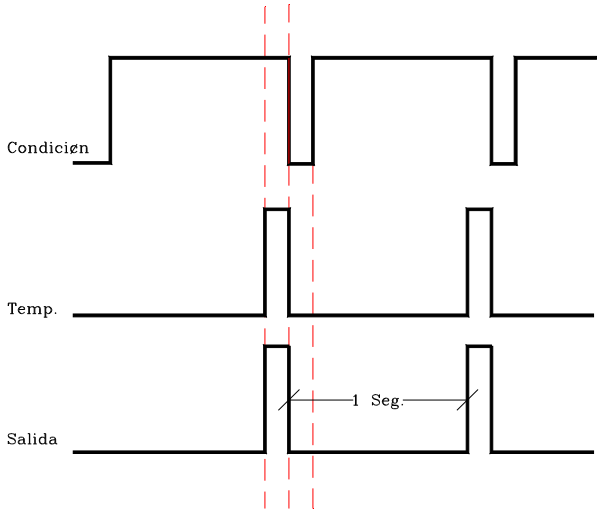
SOLUCIÓN EN AWL

```
UN    M     0.0
L     S5T#1S
SE    T     1
U     T     1
=     M     0.0
UN    M     0.0
BEB
UN    A     4.0
=     A     4.0
BE
```

Si añadimos más BEB, con otras salidas tenemos intermitentes cada uno con doble frecuencia que el anterior. El programa continuaría de la siguiente manera:

```
UN    A    4.0
=     A    4.0
BEB
UN    A    4.1
=     A    4.1
BEB
UN    A    4.2
=     A    4.2
BEB
UN    A    4.3
=     A    4.3
BEB
UN    A    4.4
=     A    4.4
BEB
.....
.....
BE
```

Veamos en un esquema lo que está ocurriendo con las marcas y porqué esto actúa como un intermitente:


Condición
Temp.
1 Seg.
Salida

EJERCICIO 14: SEMÁFORO CON INTERMITENCIA

TEORÍA PREVIA: Intermitente (BEB).

DEFINICIÓN Y SOLUCIÓN

Vamos a programar el semáforo del ejercicio anterior, pero modificando el ciclo.

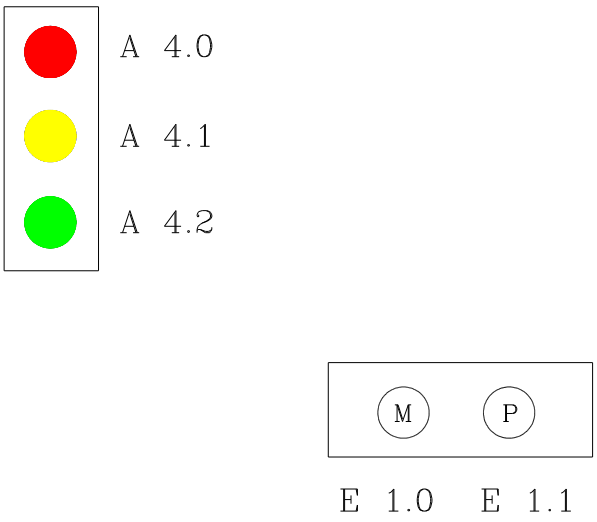
Cuando lo ponemos en marcha queremos que el ciclo funcione de la manera siguiente:

1º/ Verde durante 5 seg.

2º/ Amarillo intermitente durante 2 seg.

3º/ Rojo durante 6 seg.

Cuando le demos al pulsador de paro queremos que se pare todo.



Cuando le demos al pulsador de marcha queremos que el ciclo siempre empiece con el verde.

SOLUCIÓN EN AWL

```
U   E   0.0     //Si le damos al pulsador de marcha
S   A   4.2     //Enciende 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       //Cuando acabes de contar
R   A   4.2     //Apaga el verde
S   M   10.0    //Y activa la marca 10.0
U   M   10.0    //Si está activa la marca 10.0
U   M   0.1     //Y está activa la marca 0.1
=   A   4.1     //Enciende el amarillo
U   M   10.0    //Si está activa la marca 10.0
L   S5T#2S      //Cuanta 2 segundos
SE  T   2       //Con el temporizador 2
U   T   2       //Cuando acabes de contar
R   M   10.0    //Desactiva la marca 10.0
S   A   4.0     //Y enciende el rojo
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
R   A   4.0     //Apaga el rojo
S   A   4.2     //Y enciende el verde
U   E   0.1     //Si le damos al pulsador de paro
R   A   4.0     //Apaga el rojo
R   M   10.0    //Apaga la marca de amarillo
```

```
R     A     4.2          //Apara el verde
UN    M     0.0          //Hacemos que la marca 0.0 se active
L     S5T#200MS          //una vez cada 200 milisegundos
SE    T     4
U     T     4
=     M     0.0
UN    M     0.0
BEB
UN    M     0.1          //La marca 0.1 estará 200 milisegundos activa
=     M     0.1          //y 200 milisegundos no activa
BE
```

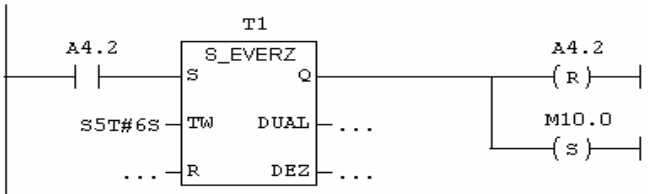
Lo que hemos hecho ha sido sustituir la luz de amarillo por una marca. (M 10.0) La marca 10.0 estará activa durante 2 segundos, igual que en el ejercicio anterior lo estaba la luz de amarillo. Además nos hemos hecho un intermitente con la marca 0.1 igual que el ejercicio pasado. La luz de amarillo la encedemos cuando coincidan las dos marcas.

KOP

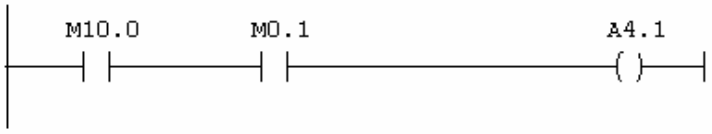
OB1 : Título:

Segm. 1 : Título:

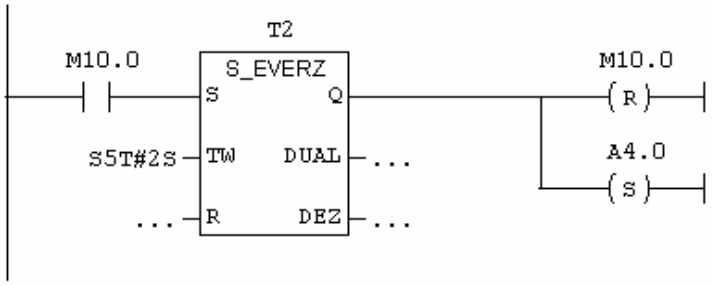
Segm. 2 : Título:



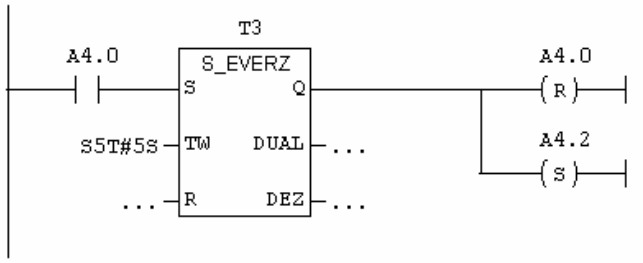
Segm. 3 : Título:



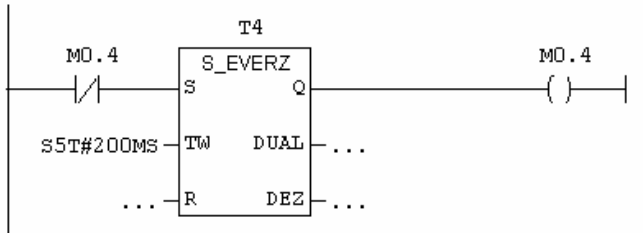
Segm. 4 : Título:



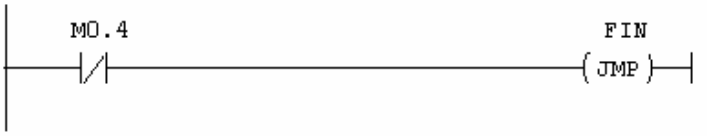
Segm. 5 : Título:



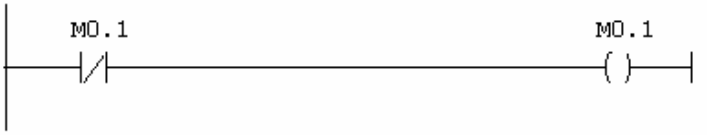
Segm. 6 : Título:

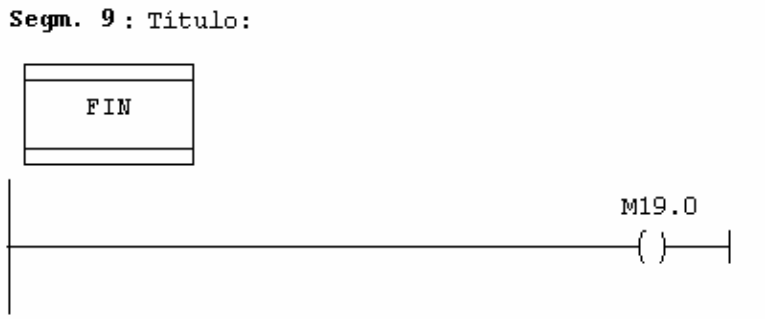


Segm. 7 : Título:

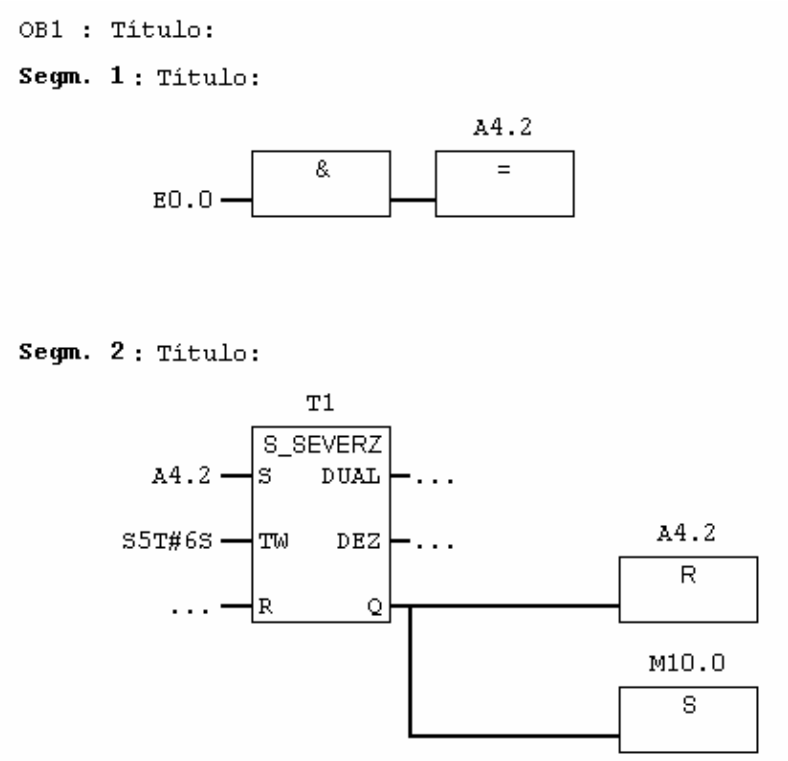


Segm. 8 : Título:

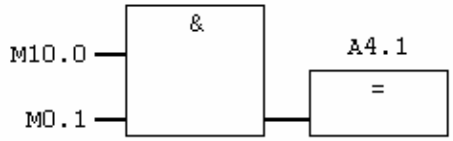




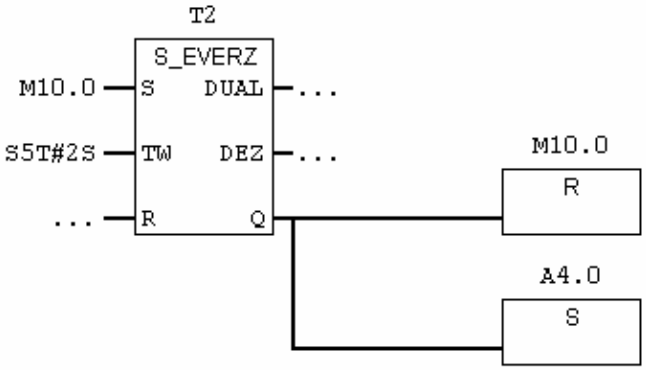
Solución en FUP



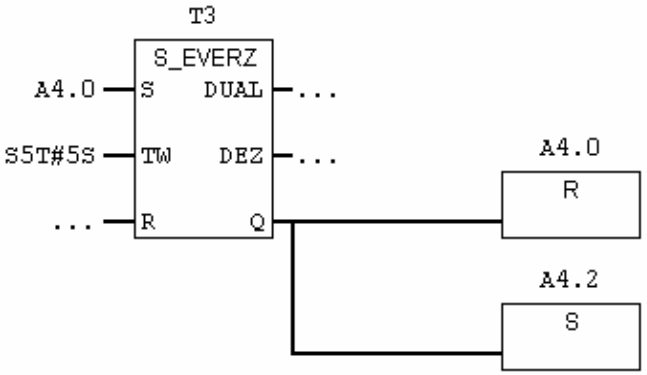
Segm. 3 : Título:



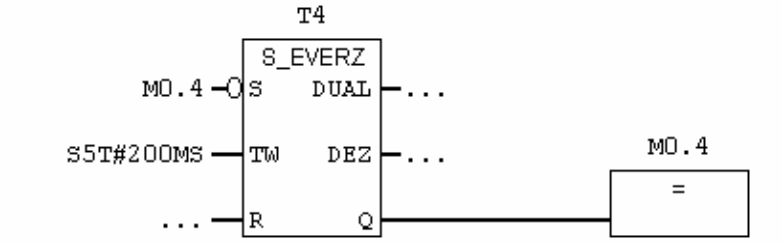
Segm. 4 : Título:



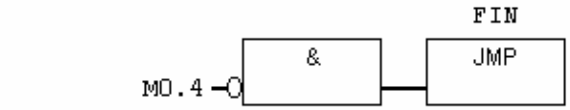
Segm. 5 : Título:



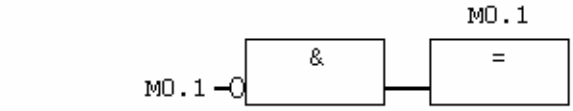
Segm. 6 : Título:



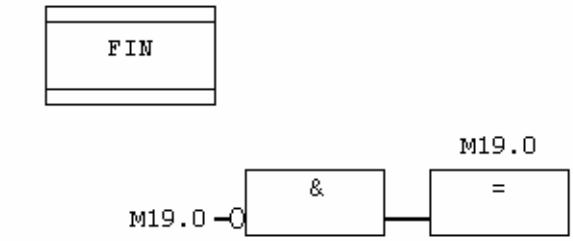
Segm. 7 : Título:



Segm. 8 : Título:



Segm. 9 : Título:



EJERCICIO 15: PARKING.

TEORÍA

CONTADORES Y COMPARACIONES

Veamos como podemos programar un contador. A los contadores les llamaremos Z. Veamos todo lo que podemos hacer con un contador:

```
      U     E     0.0
      ZV    Z     1        Contar una unidad con un flanco positivo de E0.0
-------------------------------------------------
      U     E     0.1
      ZR    Z     1        Descontar una unidad con un flanco positivo de E0.1
-------------------------------------------------
      U     E     0.2
      L     C#10
      S     Z     1        Setear con un valor. Inicializar el contador.
-------------------------------------------------
      U     E     0.3
      R     Z     1        Resetear el contador (poner a cero).
-------------------------------------------------
      U     Z     1        Consultar el bit de salida.
      =     A     4.0
-------------------------------------------------
      U     E     0.4      Utilizar una entrada para contar y descontar.
      FR    Z     1
```

Esto es todo lo que podemos hacer con un contador. No es necesario que para cada contador utilicemos todas las posibilidades ni en este orden.

Z1 es el contador que estamos gastando en este ejemplo. El número de contadores que podemos gastar depende de la CPU que estemos gastando.

El contador va a almacenar un valor. Será la cuenta que lleve el contador en cada momento.

A parte de esto, nosotros también podemos acceder a Z1 con instrucciones de bit. De este modo estamos consultando el bit de salida del contador.

Este bit estará a 0 siempre y cuando el contador esté a 0. Este bit estará a 1 siempre y cuando el contador tenga un valor distinto de cero. (Los contadores no cuentan números negativos).

Además de esto podemos consultar el valor del contador y trabajar con él como número entero.

Con los contadores, podemos trabajar de dos modos distintos. Una forma es cargar inicialmente un valor en el contador. Luego podemos saber cuando ha llegado a cero. Tenemos un bit de salida que nos da cambio cuando pasamos de un valor distinto de cero a cero.

Otra forma de trabajar con los contadores, es comenzar a contar desde cero y comparar con los valores con los cuales queramos que ocurra algo.

Para esto nos hará falta comparar dos valores. Para comparar, al PLC le hace falta tener estos valores en dos registros internos que son el acumulador 1 y el acumulador 2.

Para meter los valores en los acumuladores, tenemos la instrucción de carga. (L).

Cuando cargamos un valor, siempre se carga en el acumulador 1. Cuando volvemos a cargar otro valor, también se guarda en acumulador 1. Lo que tenía en el acumulador 1 pasa al acumulador 2, y lo que tenía en el acumulador 2 lo pierde.

En nuestro caso, cargaremos el valor de Z1 y a continuación cargaremos el valor con el que queremos comparar.

Una vez tengamos los valores en el acumulador, tendremos que compararlos. Para ello tenemos las siguientes instrucciones:

>	>	>=	<=	==	<>
Mayor	Menor	Mayor o igual	Menor o igual	Igual	Dist.

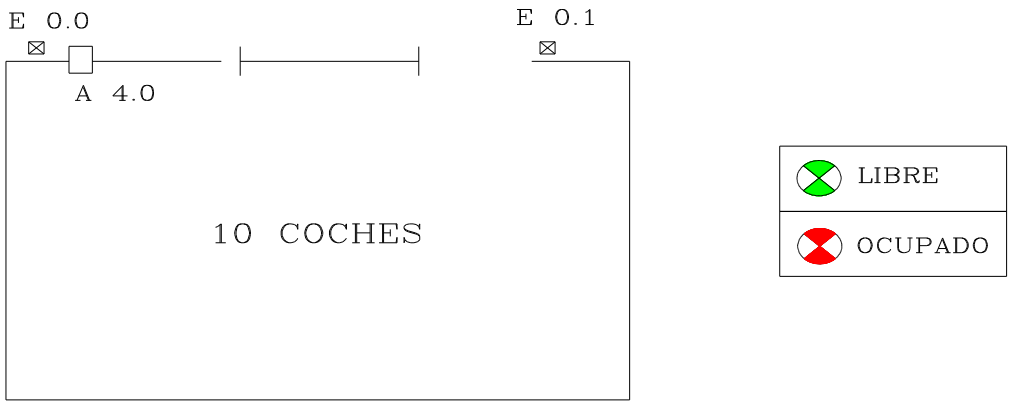
A continuación del símbolo de comparación pondremos una I si lo que estamos comparando son dos números enteros. Pondremos una R si lo que estamos comparando son números reales.

EJERCICIO 15: PARKING

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Contadores y comparaciones. (Operaciones de carga).

Tenemos el siguiente parking de coches:



El funcionamiento que queremos es 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

```
U     E     0.0          //Si llega un coche
U     A     4.6          //Y está libre
=     A     4.0          //Abre la barrera
U     A     4.0          //Si se he abierto la barrera
ZV    Z     1            //Cuenta uno con el contador 1
U     E     0.1          //Si sale un coche
ZR    Z     1            //Descuenta 1 con el contador 1
L     Z     1            //Carga el contador 1
L     10                 //Carga un 10
<I                       //Si en el contador hay menos de 10
S     A     4.6          //Enciende la luz de libre
R     A     4.7          //Y apaga la de ocupado
==I                      //Si el contador de coches vale 10
R     A     4.6          //Apaga la luz de ocupado
S     A     4.7          //Y enciende la luz de libre
BE
```

Esto sería una forma de hacerlo utilizando un contador y la función de comparación.

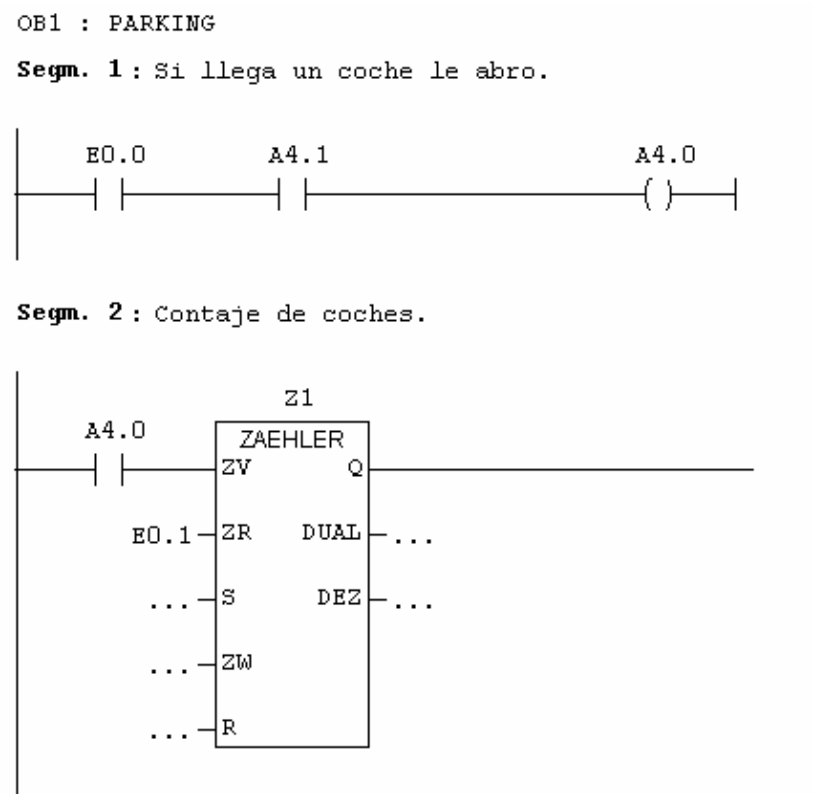
Como sólo queremos hacer una comparación, es decir sólo nos interesa la cantidad de 10 coches para hacer el cambio de las luces, tenemos otra posibilidad de programarlo. Podemos hacerlo utilizando la salida binaria del contador. Veamos como quedaría resuelto en AWL.

SOLUCIÓN AWL

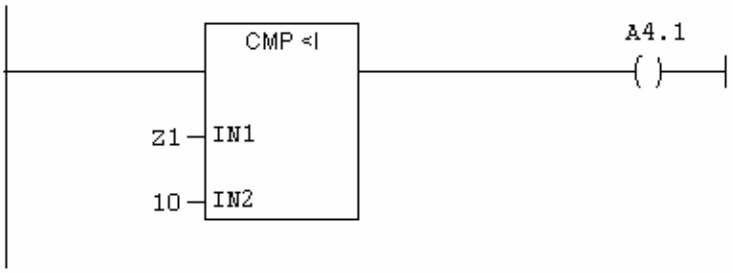
```
U     E     0.7      //Si activamos la entrada 0.7
L     C#10           //Carga un 10
S     Z     1        //Mete el 10 en el contador
U     E     0.0      //Si llega un coche
U     A     4.6      //Y está libre
=     A     4.0      //Abre la barrera
U     A     4.0      //Si se ha abierto la barrera
ZR    Z     1        //Descuenta 1 en el contador 1.1 plaza libre menos
U     E     0.1      //Si sale un coche
ZV    Z     1        //Cuenta 1 en el contador 1. 1 plaza libre mas.
UN    Z     1        //Si en el contador 1 hay un 0
=     A     4.7      //Enciende la luz de ocupado
UN    A     4.7      //Si no está ocupado
=     A     4.6      //Enciende la luz de libre
BE
```

Pasemos a ver las soluciones en KOP y AWL

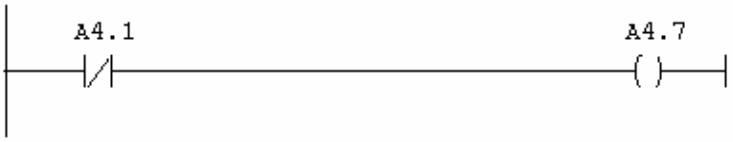
Solucion en KOP



Segm. 3: Con menos de 10 coches está libre.



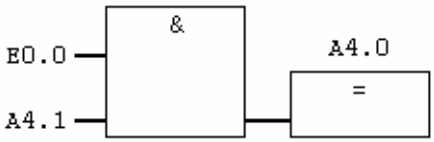
Segm. 4: Si no está libre, está ocupado.



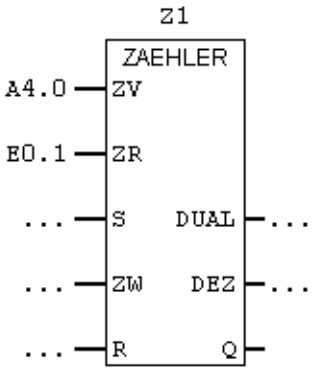
Solucion en FUP

OB1 : PARKING

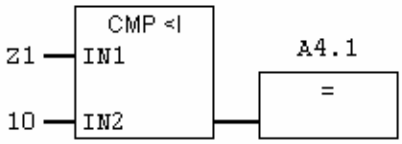
Segm. 1 : Si llega un coche le abro.



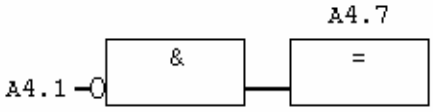
Segm. 2 : Contaje de coches.



Segm. 3 : Con menos de 10 coches está libre.



Segm. 4 : Si no está libre, está ocupado.



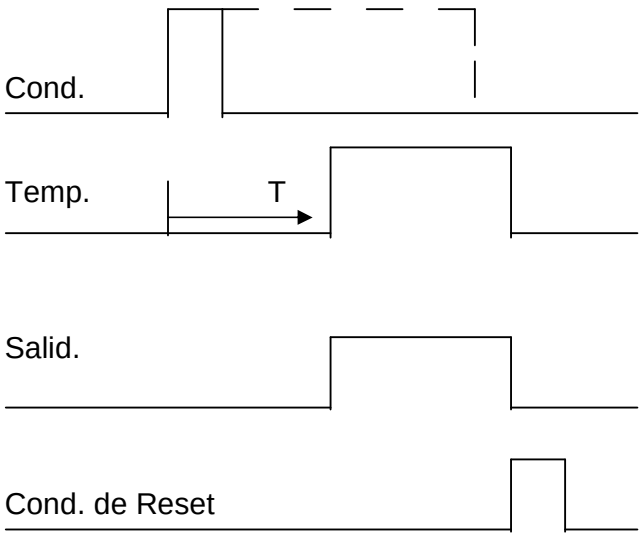
EJERCICIO 16: PUERTA CORREDERA

TEORÍA

TEMPORIZADORES SS, SV y SA.

Además de los temporizadores que hemos que hemos visto en ejercicios anteriores, tenemos tres más llamados temporizadores con memoria. Son los temporizadores SS, SV y SA.

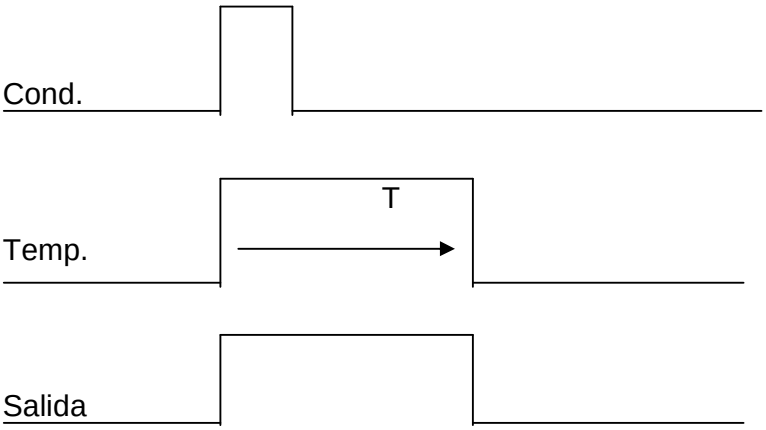
El temporizador SS es equivalente al temporizador SE. El funcionamiento es similar. La diferencia está en que el funcionamiento del temporizador es independiente de la entrada. Una vez se ha detectado un flanco de subida de la entrada se ejecuta el ciclo del temporizador independientemente de lo que hagamos con la entrada. A continuación vemos un esquema del funcionamiento del temporizador. Observamos que tenemos un problema. El temporizador se queda a uno si nadie lo resetea. Necesitamos añadir una condición que resetee el temporizador para que vuelva a su estado inicial y lo podamos volver a utilizar.




```
U     E     0.0
L     S5T#5S
SS    T     1
U     T     1
=     A     4.0
```

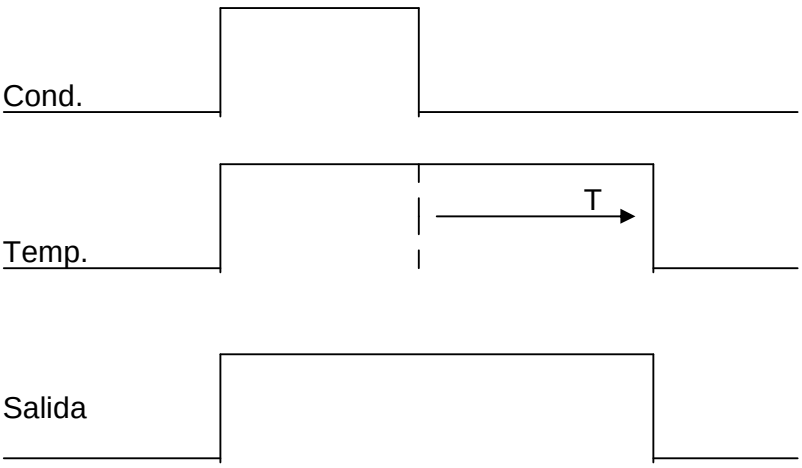
El temporizador SV es equivalente al SI. El funcionamiento es el mismo, pero es independiente de la condición de entrada. Una vez se ha detectado un flanco de

subida de la entrada se ejecuta todo el ciclo del temporizador. Veamos el esquema de funcionamiento.




```
U    E   0.0
L    S5T#4S
SV   T   2
U    T   2
=    A   4.1
```

También disponemos de un temporizador de retardo a la desconexión Es el temporizador SA. Veamos el esquema de funcionamiento del temporizador.



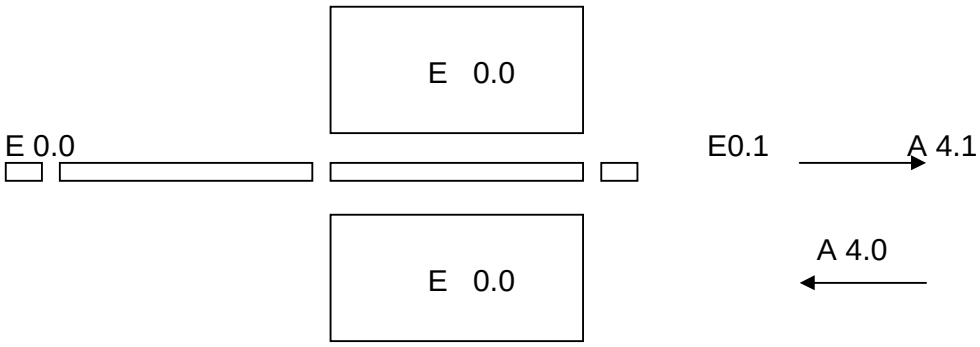

```
U    E   0.3
L    S5T#6S
SA   T   5
U    T   5
=    A   5.3
```

EJERCICIO 16: PUERTA CORREDERA

TEORÍA PREVIA: Temporizadores + contadores.

DEFINICIÓN Y SOLUCIÓN

Tenemos 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 de apertura A 4.0. La puerta se está abriendo hasta que llegue al final de carrera. Cuando llega al final de carrera, comienza a cerrarse. (Motor A 4.1). Se está cerrando hasta que llega al final de carrera.

Tenemos dos pulsadores de control. El de marcha y el de paro. 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.

Además tenemos un relé térmico. Queremos que cuando salte el relé térmico se pare la puerta hasta que lo rearmemos. Cuando haya saltado el relé térmico 5 veces queremos que se bloquee la puerta.

Volverá a funcionar cuando desbloqueemos la puerta.

SOLUCIÓN EN AWL

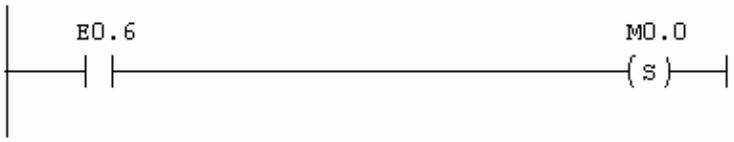
```
U   E   0.6     //Si le damos al pulsador de marcha
S   M   0.0     //Activa la marca 0.0
U   E   0.7     //Si le damos al pulsador de paro
R   M   0.0     //Desactiva la marca 0.0
U   E   0.0     //Si alguien pisa la goma
U   M   0.0     //Y está la puerta en marcha
U   Z   1       //Y el contador 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
```

Veamos la solucion en KOP y en FUP.

KOP

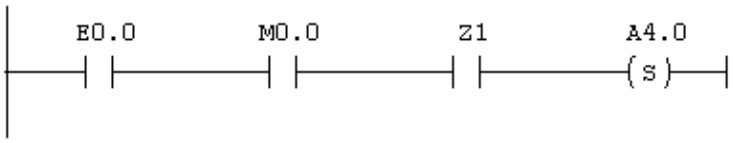
OB1 : Título:

Segm. 1 : Título:

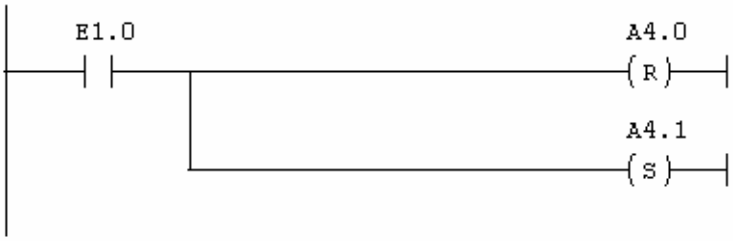


Segm. 2 : Título:

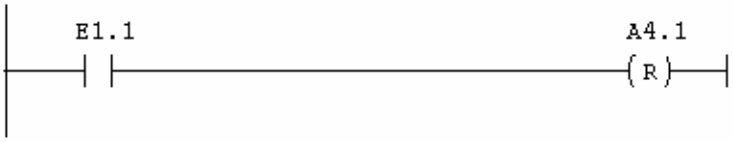
Segm. 3 : Título:



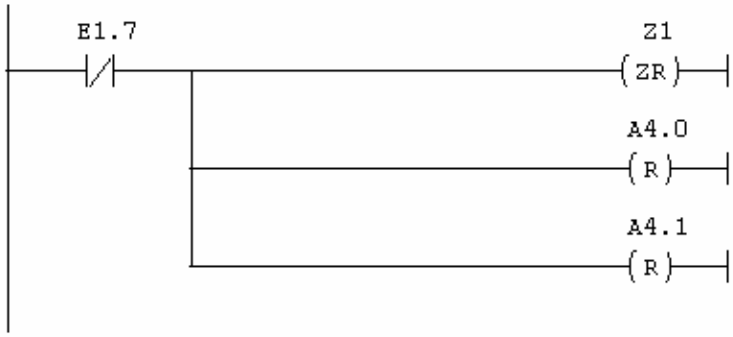
Segm. 4 : Título:



Segm. 5 : Título:



Segm. 6 : Título:

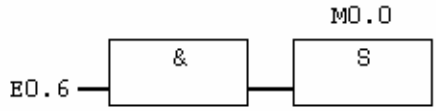


Segm. 7 : Título:

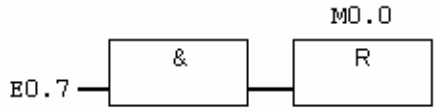
FUP

OB1 : Título:

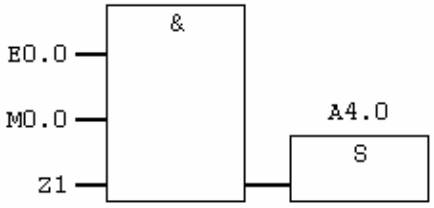
Segm. 1 : Título:



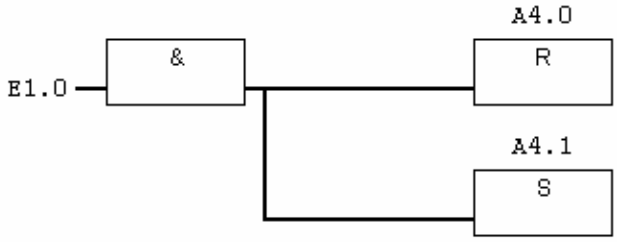
Segm. 2 : Título:



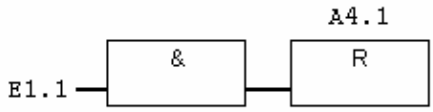
Segm. 3 : Título:



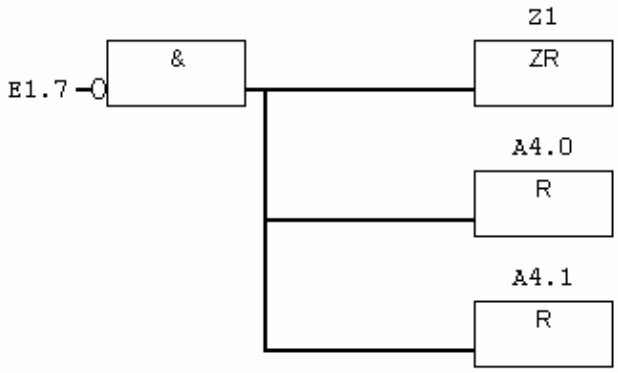
Segm. 4 : Título:



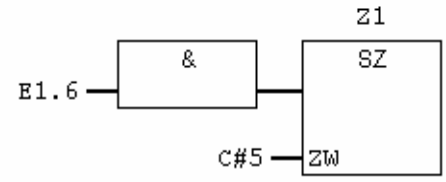
Segm. 5 : Título:



Segm. 6 : Título:



Segm. 7 : Título:



EJERCICIO 17: CONTAR Y DESCONTAR CADA SEGUNDO

TEORÍA PREVIA: Intermitente + contadores.

DEFINICIÓN Y SOLUCIÓN

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:

Que siga contando por donde iba.

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 iba.
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
```

De esta manera podríamos temporizar tiempos más grandes de los que me permiten los temporizadores. Con un temporizador lo máximo que puedo temporizar es 999 unidades de la base de tiempos 3. Esto viene a ser dos horas y pico. Si quiero temporizar más tiempo, puedo generarme yo una base de tiempos con un

generador de pulsos, y luego con un contador lo que hacemos es contar esos pulsos que me acabo de generar.

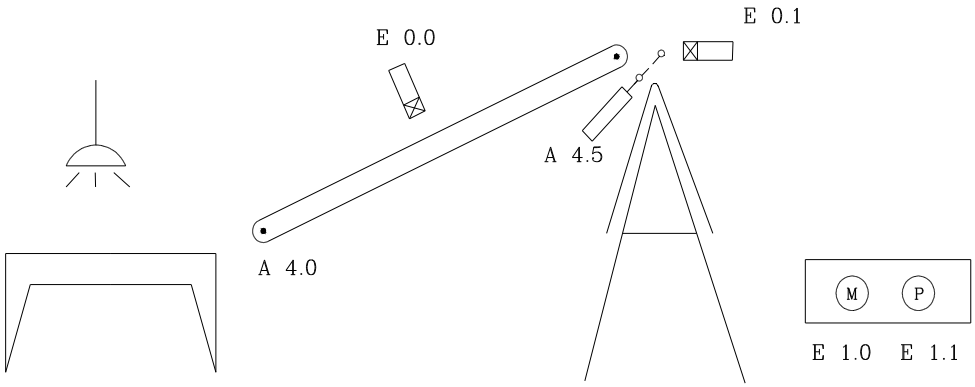
Ejercicio propuesto: Resolver el programa en KOP y en FUP con las instrucciones que se han visto para ejercicios anteriores.

EJERCICIO 18: FÁBRICA DE CURTIDOS

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Temporizadores + contadores.

Tenemos una fábrica de curtidos. Tenemos una mesa de trabajo, una cinta transportadora y un caballete dispuestos del siguiente modo:



Cuando le demos al pulsador de marcha, queremos que se ponga en marcha la cinta transportadora. La piel va cayendo por un lado del caballete. Cuando llegue a la mitad, queremos que se active el émbolo y que doble la piel por la mitad.

Lo que pretendemos es que tenga el tamaño que tenga la piel, siempre doble por la mitad.

Tenemos que medir la piel de algún modo. Lo que vamos a hacer es generar dos trenes de impulsos de frecuencia uno el doble que el otro.

Mientras esté la primera célula activa, estaremos contando los pulsos de frecuencia menor con un contador. Mientras esté activa la segunda célula estaremos contando los pulsos de frecuencia mayor con otro contador.

Cuando la cuenta de los dos contadores sean iguales querrá decir que la piel está por la mitad. Activaremos el émbolo durante 3 segundos.

SOLUCIÓN EN AWL

```
U     E     1.0          //Si le damos al botón de marcha
S     A     4.0          //Pon en marcha la cinta
UN    M     0.0          //Generamos unos pulsos
L     S5T#10MS           //de 10 milisegundos
SE    T     1            //con la marca 0.0
U     T     1
=     M     0.0
UN    M     0.1          //Generamos unos pulsos
L     S5T#20MS           //de 20 milisegundos
SE    T     2            //con la marca 0.1
U     T     2
=     M     0.1
U     E     0.0          //Mientras esté la primera célula activa
U     M     0.1          //y lleguen pulsos de frecuencia lenta
ZV    Z     1            //Cuéntalos con el contador 1
U     E     0.1          //Mientras está activa la segunda célula
U     M     0.0          //Y lleguen los pulsos rápidos
ZV    Z     2            //Cuéntalos con el contador 2
L     Z     1            //Carga el contador 1
L     Z     2            //Carga el contador 2
==I                      //Cuando sean iguales
```

```
S     A     4.5          //Activa el émbolo
U     A     4.5          //Cuando hayas activado el émbolo
L     S5T#3S             //Cuenta 3 segundos
SE    T     3            //Con el temporizador 3
U     T     3            //Cuando acabes de contar
R     A     4.5          //Desactiva el émbolo
R     Z     1            //Resetea el contador 1
R     Z     2            //Y resetea el contador 2
U     E     1.1          //Si pulsamos el paro
R     A     4.0          //Para la cinta
BE
```

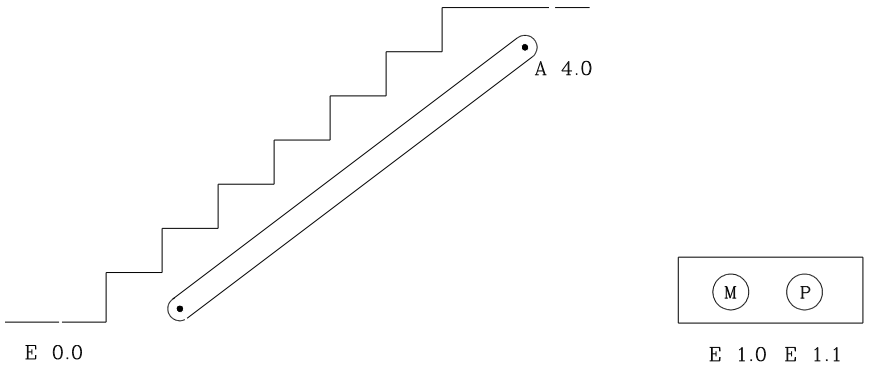
Ejercicio propuesto: Resolver el problema en KOP y en FUP con las instrucciones que se han visto anteriormente.

EJERCICIO 19: ESCALERA AUTOMÁTICA.

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Temporizadores SA

Tenemos una escalera automática.



El funcionamiento que queremos es el siguiente:

Cuando le demos al pulsador de marcha, queremos que la escalera esté activa. Eso no quiere decir que se ponga en marcha. Se pondrá en marcha cuando llegue una persona.

Cuando esté en marcha, el funcionamiento que queremos es el siguiente:

Cuando una persona pise, queremos que la escalera se ponga en marcha. A partir de cuando la persona suba al primer escalón, queremos que esté en marcha 5 seg. que es lo que le cuesta a la persona subir.

Si antes de acabar el ciclo sube otra persona queremos que también llegue al final de su trayecto. En resumen, queremos que la escalera esté en marcha 5 seg. desde que la última persona subió al primer escalón.

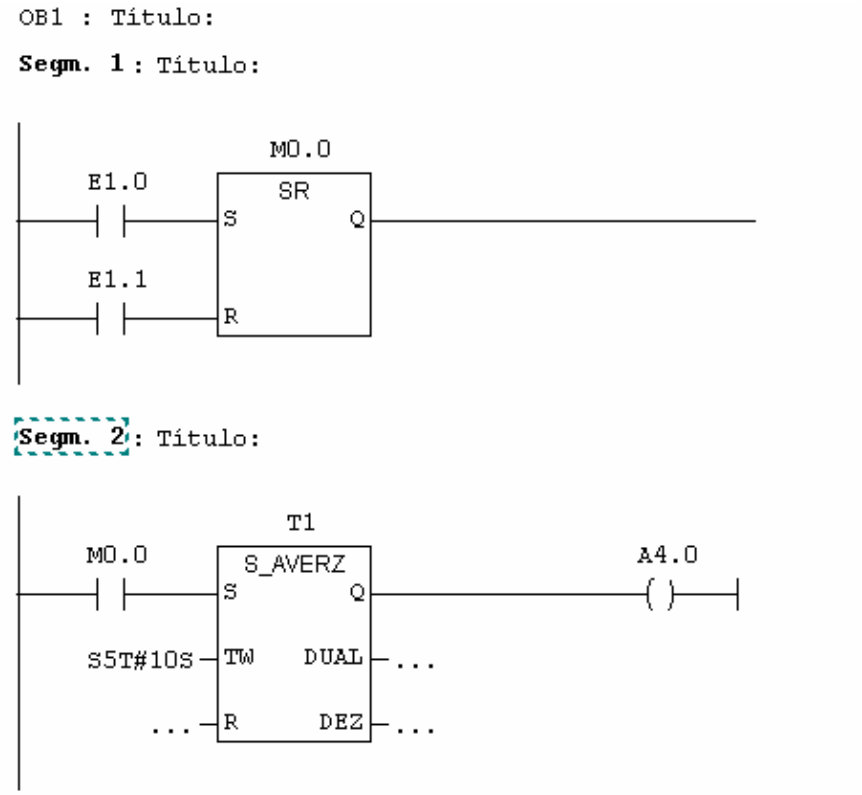
Cuando le demos al pulsador de paro, queremos que si hay alguna persona que está subiendo llegue al final de su trayecto, pero si llega otra persona ya no pueda subir.

SOLUCIÓN EN AWL

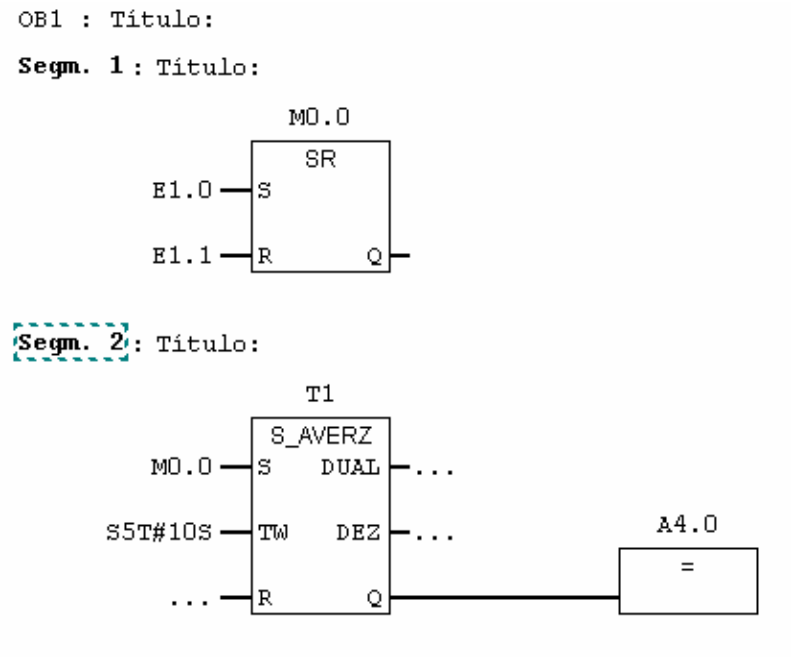
```
U    E    1.0          //Si le damos al pulsador de marcha
S    M    0.0          //Activa la marca 0.0
U    M    0.0          //Si está activa la marca 0.0
U    E    0.0          //Y llega una persona
L    S5T#5S            //Cuenta 5 segundos
SA   T    1            //A partir de cuando empiece a subir
U    T    1            //Mientras no hayas acabado de contar
=    A    4.0          //Estará en marcha la escalera
U    E    1.1          //Si le damos al paro
R    M    0.0          //Resetea la marca 0.0
BE
```

Veamos las soluciones en KOP y en FUP.

Solución en KOP.



Solución en FUP:



EJERCICIO 20: MASTER CONTROL RELAY.

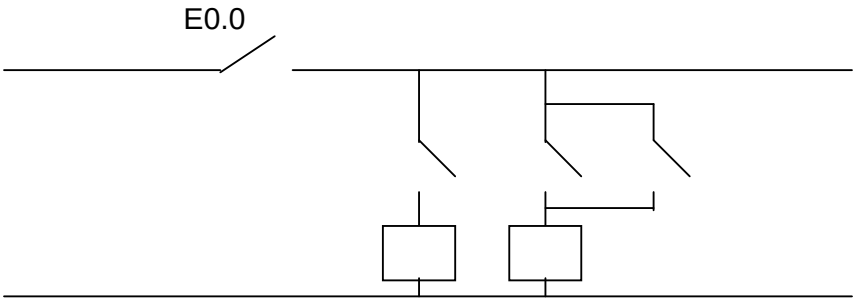
TEORÍA

INSTRUCCIÓN MASTER CONTROL RELAY

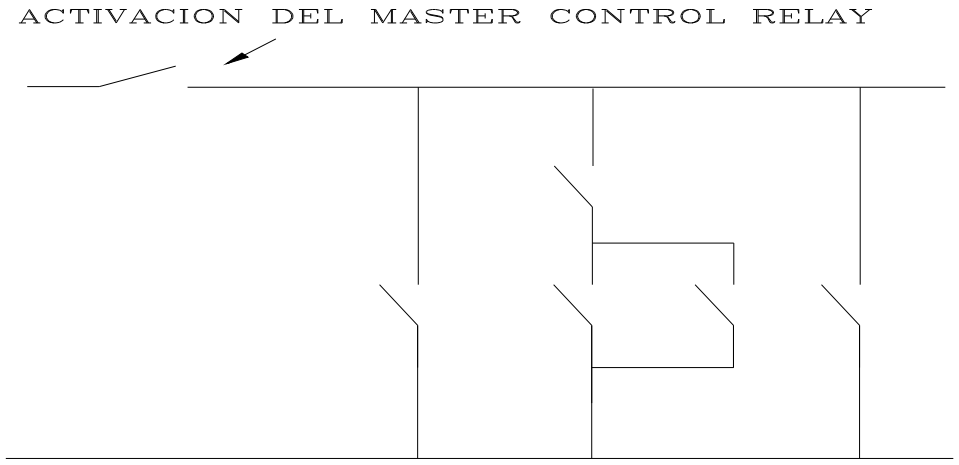
El Master Control Relay consta de 4 instrucciones:

MCRA	Activar el Master Control Relay.
MCR(	Abrir el paréntesis. (Necesita una condición previa).
)MCR	Cerrar el Master Control Relay.
MCRD	Desactivar el Master Control Relay.

Esta instrucción la utilizaremos para programar esquemas como el que sigue:



Delante de cada paréntesis que abramos tendremos que poner una condición que hará las funciones del contacto E 0.0 en el esquema.


ACTIVACION DEL MASTER CONTROL RELAY

EJERCICIO 21: MASTER CONTROL RELAY

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Introducción teórica a la instrucción Master Control Relay.

Vamos a ver una instrucción nueva que no existía en S5.

Es la instrucción MASTER CONTROL RELAY.

Esto viene a ser como una activación a desactivación de un trozo de programa. La función que realiza es la conexión o desconexión de un circuito que represente un esquema eléctrico.

Esto sólo sirve para operaciones de contactos. Dentro del MASTER CONTROL RELAY no podemos poner temporizadores o llamadas a otros bloques. El programa si que nos permite hacerlo pero no funciona correctamente.

Está pensado para utilizar contactos con asignaciones "=". Viene a ser como un circuito eléctrico. Lo que quede activado cuando no se ejecuta lo que hay dentro de los paréntesis del Master Control Relay, se desactiva.

Si dentro del Master Control Relay utilizamos instrucciones SET y RESET, no funciona como hemos dicho. Cuando deja de actuar lo que hay dentro de los paréntesis, si estaba activado con un SET se mantiene activado.

Si no hacemos un RESET desde fuera, no se desactiva.

Veamos cuales son las instrucciones necesarias para hacer un MASTER CONTROL RELAY:

```
MCRA                    Activar al MCR
U    E    0.0
MCR(
U    E    0.1
=    A    4.0
)MCR
U    E    0.2
MCR(
U    E    0.3
=    A    4.1
)MCR
U    E    0.7
=    A    4.7
MCRD                    Desactivar el MCR.
```

Tenemos dos instrucciones para activar y desactivar el MCR. Dentro de estas instrucciones, podemos abrir y cerrar hasta 8 paréntesis. Los podemos hacer anidados o independientes.

Siempre, delante de cada paréntesis tenemos que poner una condición. Hace la función del contacto E 0.0 del gráfico anterior.

Vemos que cada paréntesis funciona sólo cuando tenemos activa su condición. Cuando su condición no está activa el trozo de programa en cuestión deja de funcionar y las salidas se desactivan. Es como si realmente quitásemos tensión a ese trozo de programa.

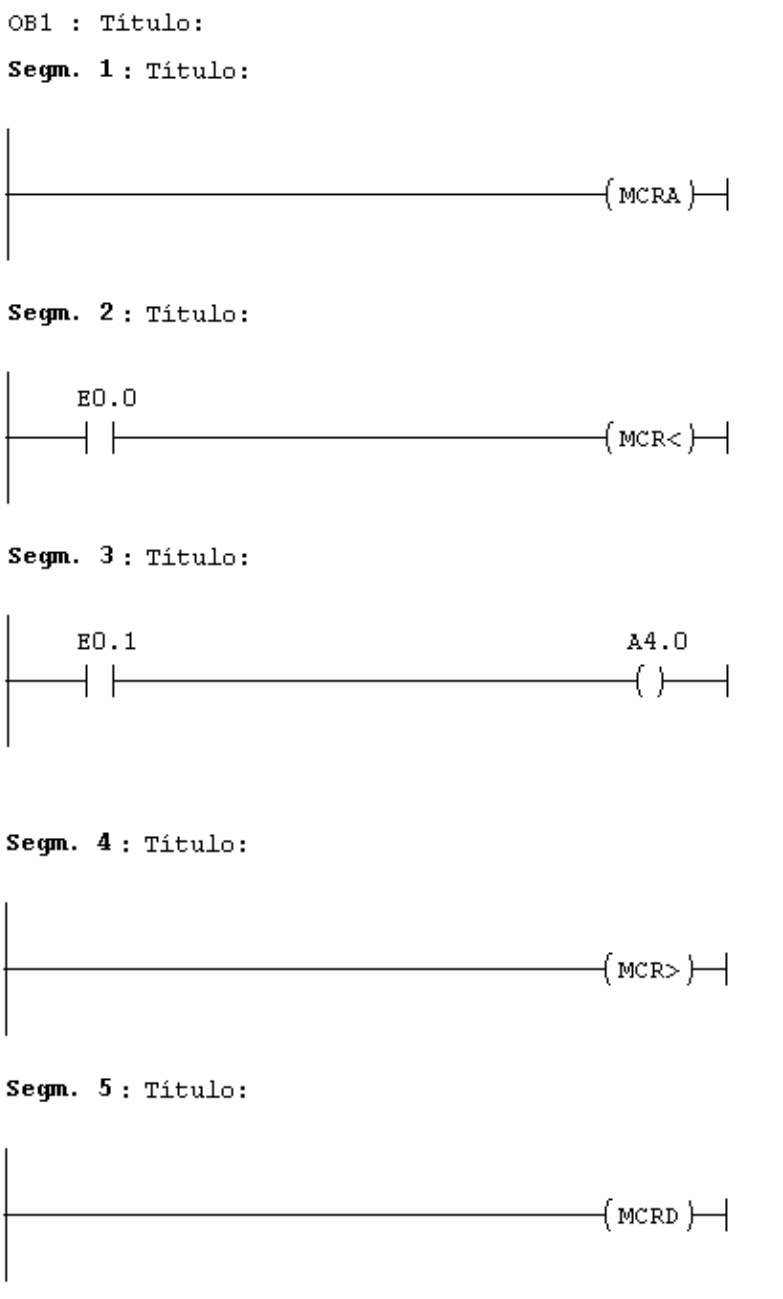
Esto no ocurre si el trozo de programa se deja de leer por cualquier otra causa. Si hacemos un salto o una meta, o programamos otros bloques, cuando no se ejecuta una parte del
programa, las salidas se quedan como estaban. Si estaban activas cuando dejó de ejecutarse ese trozo de programa, continúan activas. Esto no ocurre con el MCR.

En el ejemplo que hemos hacho, la última parte no está dentro de ningún paréntesis, aunque si que está dentro de la activación del MCR. Esta parte de programa si que funciona siempre.

Lo que podemos activar o desactivar es lo que tenemos dentro de los paréntesis y siempre va precedido de una condición.

Igualmente esto lo podemos hacer en cualquiera de los otros dos lenguajes.

Solución en KOP:



Solución en FUP