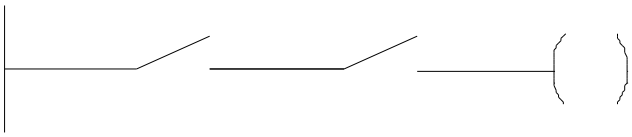
Introducción al lenguaje AWL

Instrucciones básicas

Las instrucciones básicas del lenguaje AWL son:

Operación	Instrucción
Entrada	E
Salida	A
AND	U
OR	O
LD	U o O
NAND	UN
NOR	ON
OUT	=
SET	S
RESET	R
Cargar el acumulador 1	L
Transferir el acumulador 1	T

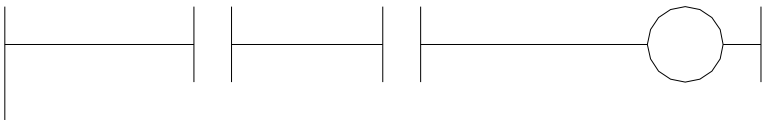
Veamos un ejemplo para un circuito sencillo:



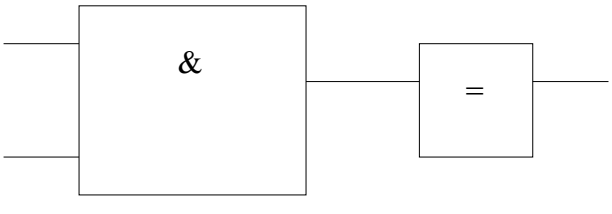
La solución en lenguaje AWL es:

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

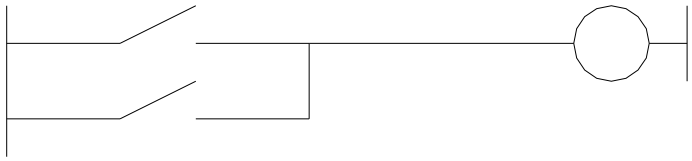
Para comparar, veamos la solución en diagrama de contactos (KOP en el entorno STEP7).



En FUP la solución sería:



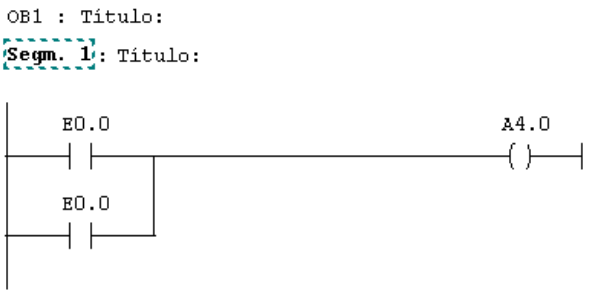
Si queremos hacer un circuito en OR como el siguiente:



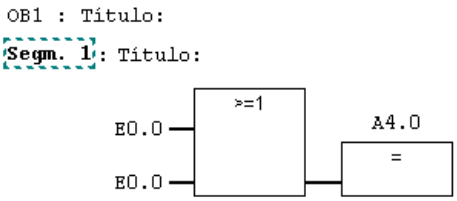
En AWL sería:

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

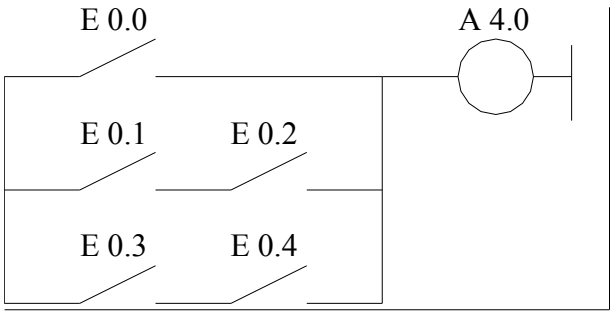
En KOP sería



Y en FUP tendríamos:



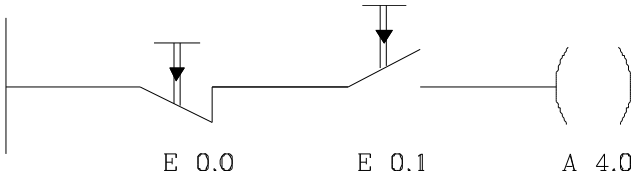
Para poder programar circuitos más complejos necesitaremos utilizar instrucciones con paréntesis. Estas instrucciones son U() y O(). No obstante, podremos obviar el uso del paréntesis y utilizamos las instrucciones U y O solas en una línea.
Veamos como programar el siguiente circuito:



En AWL lo programaríamos así:

```
U       E       0.0
O(                          (Ejemplo con paréntesis)
U       E       0.1
U       E       0.2
)
O                           (Ejemplo sin paréntesis)
U       E       0.3
U       E       0.4
=       A       4.0
BE
```

Para terminar este apartado, veremos un ejemplo de circuito con entradas normalmente cerradas:



El programa en AWL es:

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

Marcas

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.

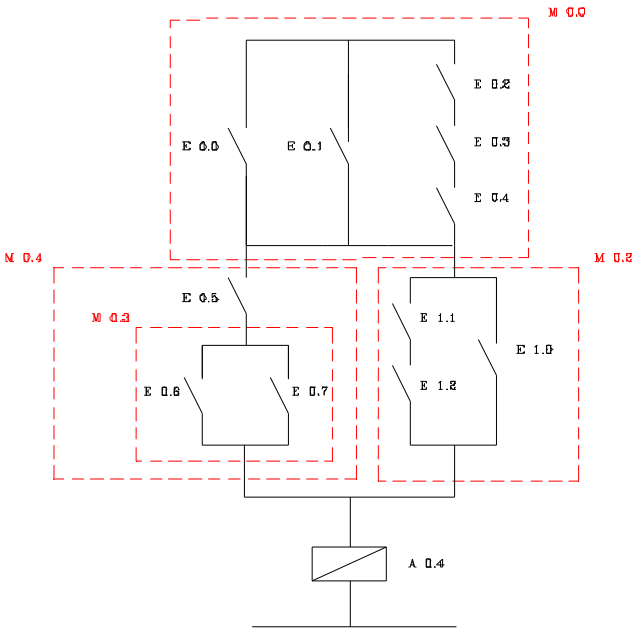
Estos bits podremos activarlos o desactivarlos como si fueran salidas. En cualquier punto del programa los podremos consultar.

A las marcas les llamaremos M. A continuación tenemos que decir a que bit en concreto nos estamos refiriendo.
En lenguaje AWL las marcas se representan por la letra M. Por ejemplo tenemos las marcas, M 0.0, M 10.7, M 4.5, etc.

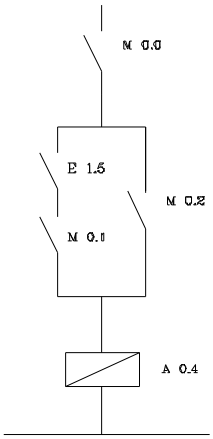
Uno de los usos habituales de las marcas es para simplificar circuitos complejos, como veremos en el siguiente ejemplo.

Queremos resolver el circuito de la figura. 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.

Sin embargo lo resolveremos mediante 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.



Utilizando marcas el circuito equivalente sería el siguiente:



El programa en AWL queda como:

```
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
```

Instrucciones SET y RESET

En AWL las instrucciones SET y RESET se representan por las letras S y R respectivamente.

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.

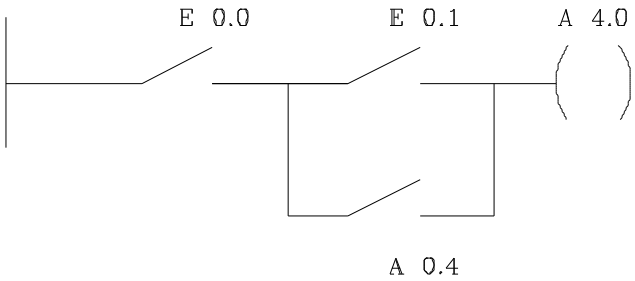
Estas instrucciones tienen prioridad. Dependen del orden en que las programemos. Siempre va a tener prioridad la última que programemos. Veamos porqué ocurre esto.

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, en su lugar accede al registro interno PAA y allí guarda "0" o "1". Sólo cuando termina cada ciclo de scan accede realmente a las salidas. Entonces lo que hace es copiar lo que hay en la PAA en las salidas reales.

En nuestro caso, si hacemos un SET y un RESET dentro del mismo ciclo de scan, al final de cada ciclo hará efecto lo último que hayamos programado.

Veamos como programar el siguiente circuito con instrucciones SET y RESET.



El programa en AWL es:

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

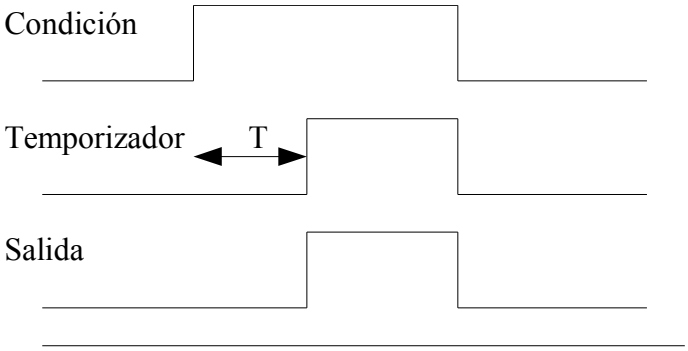
Temporizadores

En los autómatas Siemens tenemos dos modos de funcionamiento para los temporizadores, SE y SI.

El temporizador SE es un temporizador de retardo a la conexión. Para programar un temporizador, necesitamos cinco operaciones como mínimo.

1. Necesitamos una condición a partir de la cual empiece a temporizar. Esta condición puede constar de una sola instrucción o de varias.
2. Necesitamos decirle cuanto tiempo tiene que temporizar.
3. Necesitamos decirle el modo de funcionamiento y número de temporizador que queremos utilizar. (En cada CPU tenemos una cantidad de temporizadores)
4. Queremos que en algún momento dado (mientras temporiza o cuando ha acabado de temporizar) ...
5. ... haga algo !!!.

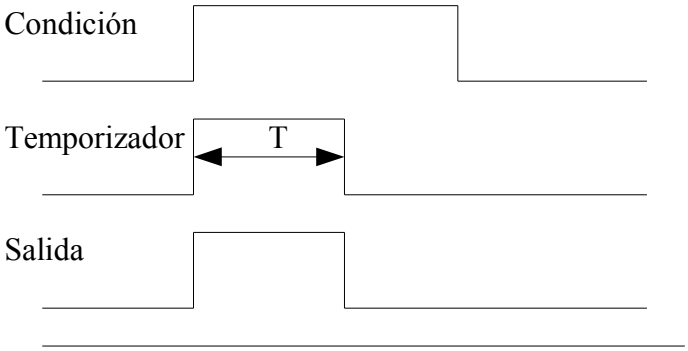
El modo de funcionamiento SE es el siguiente:




```
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:

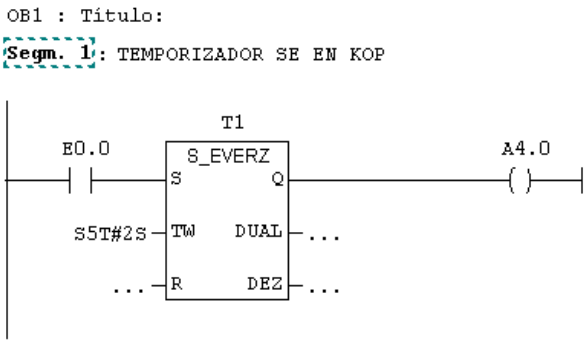



```
U     E 0.0
L     S5T#5S
SI    T 1
U     T 1
=     A 4.0
```

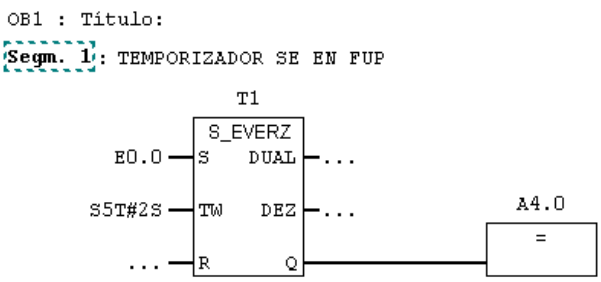
A este temporizador también podemos añadirle un RESET en cualquier momento.

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

KOP

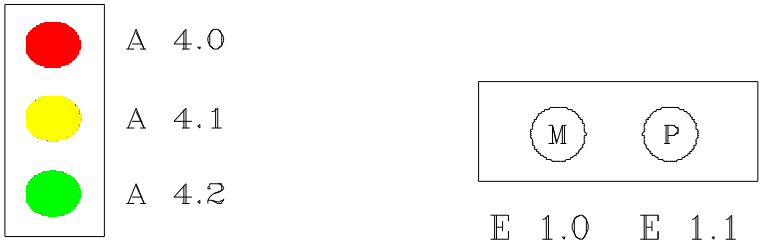


FUP



Veamos un ejemplo de uso de temporizadores.

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.



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 en AWL:

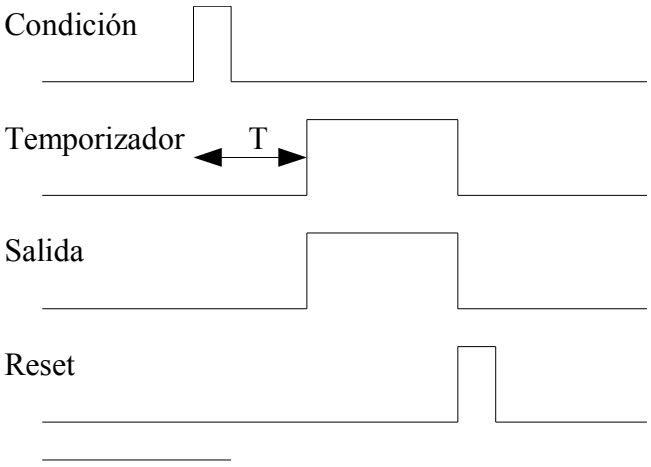
```
U     E     0.0     // Al activar el pulsador de marcha
S     A     4.2     // Encender el verde
U     A     4.2     // Si se ha encendido el verde
L     S5T#5S        // Cuenta 5 segundos
```

```
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
```

Otros tipos de temporizadores

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.

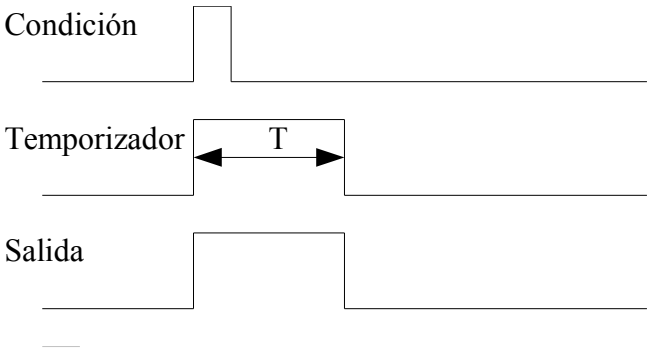
El temporizador SS es equivalente al temporizador SE. El funcionamiento es similar. La diferencia está en que el funcionamiento del temporizador es independiente de la entrada. Una vez se ha detectado un flanco de subida de la entrada se ejecuta el ciclo del temporizador independientemente de lo que hagamos con la entrada.




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

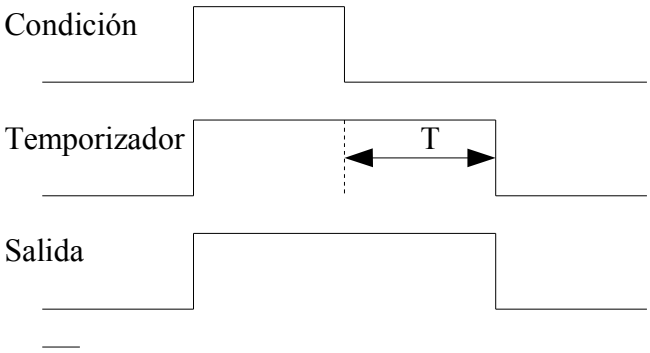
En el esquema de funcionamiento observamos que para que el temporizador se desactive tenemos que activar la señal de reset.

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#5S
SV    T 1
U     T 1
=     A 4.0
```

El temporizador SA es un temporizador de retardo a la desconexión. Veamos el esquema de funcionamiento del temporizador.




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

Finales de programa condicionales y absolutos

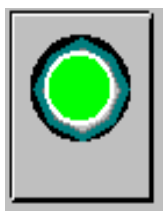
A parte del final que hemos visto (BE), existen otros dos finales. Estos son BEB y BEA. Veamos para qué podemos utilizar cada uno de ellos.

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 donde estas instrucciones pueden ser de utilidad.

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:

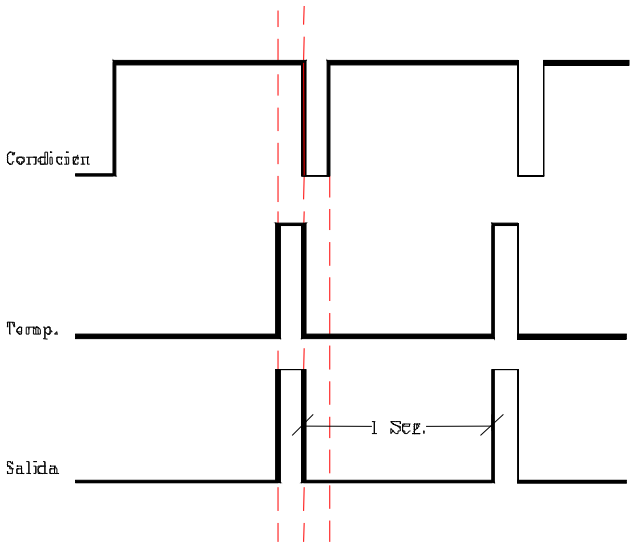
```
UN    M     0.0
L     S5T#1S
SE    T     1
```

```
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
```

En el siguiente diagrama de tiempo podemos ver qué es lo que está ocurriendo:



Contadores y comparadores

Los contadores en lenguaje AWL se representan por la letra Z. Para acceder al estado de cada contador (si su salida está activa o no) nos referiremos a ellos como Z 1, Z 2, etc. El número de contadores disponibles dependerá del autómata que estemos utilizando.

Los contadores pueden contar hacia delante (instrucción ZV), hacia atrás (instrucción ZR), por flanco positivo (cuando una entrada o una salida pase de 0 a 1) o por flanco negativo (caso contrario). También es posible inicializar un contador con un determinado valor utilizando la instrucción LOAD (L).

Resumen de operaciones con contadores:

```
      U       E       0.0
      ZV      Z       1       Contar una unidad con flanco positivo de E0.0
-----------------------------------------------
      U       E       0.1
      ZR      Z       1       Descontar una unidad con flanco positivo
                              de E0.1
-----------------------------------------------
      U       E       0.2
      L       C#10
      S       Z       1       Inicializar el contador con un valor.
-----------------------------------------------
      U       E       0.3
      R       Z       1       Poner a cero el contador.
-----------------------------------------------
      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
```

La salida del contador está a 0 cuando el valor del contador sea 0, y 1 cuando el contador tenga un valor distinto de cero. Los contadores no cuentan números negativos.

Podemos trabajar con los contadores de dos formas distintas. La primera y más sencilla es comprobar si su salida en 0 o 1 en función de si la cuenta vale cero o no vale 0 respectivamente. La segunda es comprobar el valor del contador utilizando instrucciones de comparación.

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, usaremos 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.

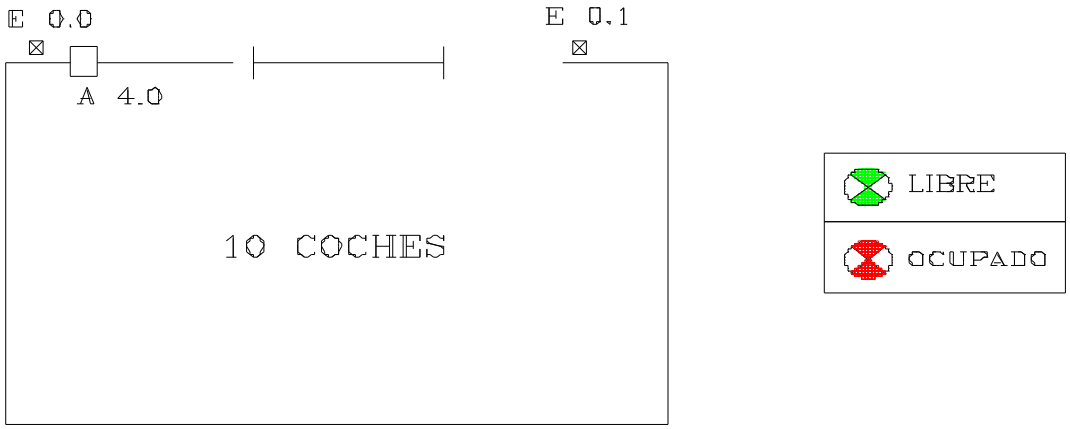
Una vez tengamos los valores en el acumulador, tendremos que compararlos. Para ello tenemos las siguientes instrucciones:

>	<	>=	<=	= =	<>
Mayor	Menor	Mayor o igual	Menor o igual	Igual	Distinto

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.

Veamos un ejercicio que puede resolverse haciendo uso de los contadores.

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.

Tipos de acceso

Hasta ahora los ejemplos que hemos visto accedían a las entradas y salidas del autómata bit a bit. Es posible sin embargo acceder a la memoria como bytes (8 bits), palabras (16 bits) o dobles palabras (32 bits). Para ello utilizaremos los sufijos B para referirnos a los bytes, W para referirnos a las palabras y D para referirnos a las dobles palabras.

Para referirnos a bits se utiliza la letra X, aunque normalmente no es necesario indicarlo (como hemos visto en los ejemplos hasta ahora) puesto que el formato x.y (donde x es la zona de memoria e y el número de bit) lleva ya implícita la referencia a nivel de bit.

De este modo, si queremos transferir 8 bits de la entrada E124 a la salida A124 haríamos:

```
L     EB    124
T     AB    124
```

Para transferir palabras haríamos:

```
L     EW    124
T     AW    124
```

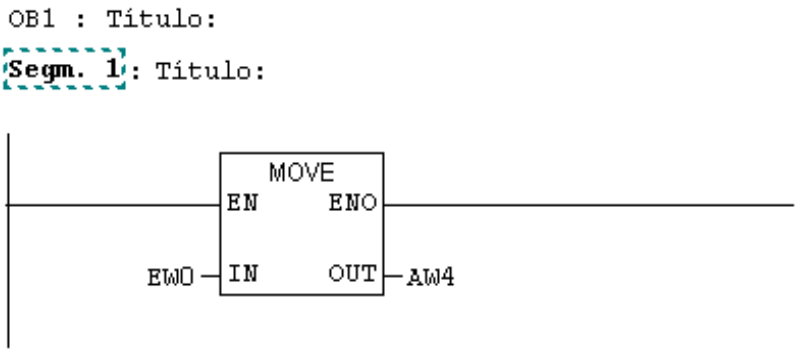
Y para dobles palabras:

```
L     ED    124
T     AD    124
```

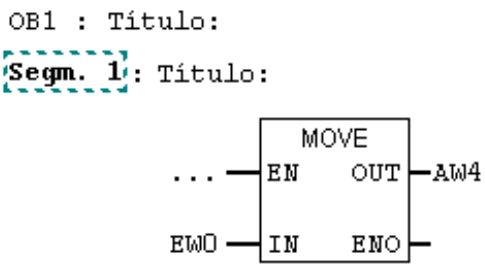
En estos ejemplos hemos usado las instrucciones L y T. Estas instrucciones sirven para transferir el operando desde o hacia el acumulador 1. El acumulador 1 es un registro interno del autómata de 32 bits. La instrucción L carga el operando en el acumulador, mientras que la instrucción T transfiere el contenido del acumulador 1 al operando.

Las operaciones de transferencia se hacen en lenguaje KOP y FUP mediante la instrucción MOVE.

En KOP:



En FUP:



La diferencia fundamental entre las instrucciones en KOP y FUP con respecto a las instrucciones L y T en AWL es que en KOP y FUP la instrucción MOVE es *condicional*, es decir, se ejecutará o no en función de la entrada EN, mientras que las instrucciones L y T de AWL son *incondicionales*, lo que significa que siempre se ejecutarán. Además en los bloque MOVE disponemos de una salida ENO que indica si la operación se ha realizado correctamente (1) o no (0).

Siguiendo con el acumulador, este puede ser cargado con distintos valores:

L	C#	Carga una constante de contador
L	S5T#	Carga una constante de tiempo
L	<número>	Carga número decimal
L	2#	Carga constante binaria
L	B#16#	Carga 8 bits en hexadecimal
L	W#16#	Carga 16 bits en hexadecimal
L	D#16#	Carga 32 bits en hexadecimal

Instrucciones de salto

Hemos visto que las instrucciones L y T se ejecutan siempre. Para solventar este problema (entre otros) disponemos de las instrucciones de salto condicional y absoluto:

SPA	Salto absoluto
SPB	Salto condicional

La instrucción SPA no tiene misterio, salta siempre a la dirección que le indiquemos. La instrucción SPB salta si el resultado de la última operación de comparación en los acumuladores es cierta.

Veamos un ejemplo de uso de estas instrucciones:

Queremos hacer dos contadores. Si el contador 1 tiene un valor más grande que el contador 2, queremos que se enciendan todas las salidas. Si el contador 1 tiene un valor más pequeño que el contador 2, queremos que se enciendan solamente las salidas pares. Si los dos contadores tienen el mismo valor queremos que se apaguen todas las salidas.

Programa en AWL:

```
        U       E       0.0
        ZV      Z       1
        U       E       0.1
        ZR      Z       1
        U       E       1.0
        ZV      Z       2
        U       E       1.1
        ZR      Z       2
        L       Z       1
        L       Z       2
        <I
        SPB     MENO
        >I
        SPB     MAYO
        ==I
        SPB     IGUA
        BEA
MENO:   L       W#16#5555
        T       AW      4
        BEA
MAYO:   L       W#16#FFFF
        T       AW      4
        BEA
IGUA:   L       0
        T       AW      4
        BE
```

Hay que tener en cuenta siempre poner un BEA antes de empezar con las metas, y otro BEA al final de cada una de las metas. En la última meta dejamos sólo el BE ya que es la última instrucción del programa y la función del BE y la del BEA es la misma.

Además de SPA y SPB tenemos otras instrucciones de salto:

SPBB	Igual que el SPB pero guardamos el valor del RLO en RB
SPZ	Salta cuando el resultado es cero
SPP	Salta cuando el resultado es positivo
SPO	Salta cuando hay desbordamiento
SPS	Salta cuando hay desbordamiento
SPZP	Salta si el resultado es mayor o igual que cero
SPU	Salta cuando el resultado no es coherente

A estas instrucciones si les añadimos el sufijo N se convierten en negadas, así por ejemplo:

SPBN	Igual que el SPB pero negado
SPZN	Salta si el resultado es distinto de cero

En S7 tenemos una instrucción más de salto que no existe en S5. Dicha instrucción es SPL. SPL salta el número de instrucciones que indiquemos en el acumulador. Para entender bien cómo usar esta instrucción veamos un ejemplo.

Queremos hacer una mezcla de pintura. Tenemos tres posibles colores para hacer.

El color ocre es el código 1. El color verde es el código 2. Y el color marrón es el código 3.

Dependiendo del código que introduzcamos queremos que la mezcla sea distinta.

Para formar el ocre queremos poner 60 gramos de amarillo, 20 gramos de azul y 20 gramos de rojo. Para formar el verde queremos 50 gramos de amarillo y 50 gramos de azul. Para formar el marrón queremos 40 gramos de amarillo, 30 de azul y 30 de rojo.

Los colores los van a simular unas palabras de marcas. Es decir si queremos que se forme el ocre, lo que queremos es que en la palabra de marcas 0 haya un 60, etc.

En el ejemplo, si seleccionamos como código del color 0, saltará al primer SPA. En este salto estamos diciendo que salte a la meta de ERRO. Si el código del color es 1 saltará al segundo SPA. Es decir, saltará a la meta de OCRE. Si el código del color es 2, saltará al tercer SPA. Es decir, iremos a la meta de VERD. Por último, si el código de color es 3, saltaremos a la meta de MARR.

Programa en AWL

```
        0.0     COLOR   INT

        L       #COLOR
        SPL     ERRO
        SPA     ERRO    //Llegará aquí si el código de color es 0
        SPA     OCRE    //Llegará aquí si el código de color es 1
        SPA     VERD    //Llegará aquí si el código de color es 2
        SPA     MARR    //Llegará aquí si el código de color es 3
ERRO:   L       0       //Llegará aquí si el código de color es otro valor
        T       MW 0
        T       MW 2
        T       MW 4
        BEA
OCRE:   L       60
        T       MW 0
        L       20
        T       MW 2
```

```
        L       20
        T       MW 4
        BEA
```

En KOP y FUP sólo tenemos dos instrucciones de salto, JMP y JPMN. Determinamos si la instrucción se ejecuta o no en función de lo que pongamos en su entrada.

Instrucción LOOP

La instrucción LOOP sirve para ejecutar un número determinado de veces un trozo de programa. Veamos un ejemplo:

```
        L       10
cont:   T       MW 0        // Hace MW = 10

        .....
        .....               // Instrucciones que se repetirán

        L       MW 0
        LOOP    cont        // Decrementa ACU1 y si es distinto de 0 salta a cont
```

Bloques de datos

Para referirnos a los bloques de datos en AWL utilizaremos el formato

```
<nombre del bloque>.<dirección>                 o bien
<nombre del bloque>.<nombre de variable>
```

Ejemplos:

```
DB1.DBW 0           // Se refiere a la palabra 0 del DB1
DB1.DBX 0           // Se refiere al primer bit (el 0) del DB1
DB1.VALOR_1         // Se refiere a la variable VALOR_1 definida en el DB1
```

En general es preferible referirnos a las variables de los DB1 por los nombres que les hayamos asignado, pues de esta forma no tenemos que especificar el tipo de variable al que nos referimos.

Para cómo insertar bloques de datos y otros ejemplos consultar el manual "Introducción a Step7".

Funciones

Los autómatas Siemens permiten definir bloques de funciones, o FCs. Estas funciones tienen partes de código que puede ser llamado desde el programa principal en OB1, o desde otras funciones.
Las funciones pueden ser sin parámetros o con parámetros. Para llamar a una función sin parámetros utilizaremos las instrucciones CC y UC. La primera, CC, es una llamada condicional y sólo se ejecuta si el RLO de la operación anterior es cierta. UC por el contrario en una llamada incondicional y se ejecutará siempre.

```
        U       E 0.0
        CC      FC 1        // Llama a la FC1 si la entrada E 0.0 está activada
        U       E 0.1
        UC      FC 2        // Llama a la FC2 siempre
```

Para llamar a funciones con parámetros utilizaremos la instrucción CALL y a continuación le indicaremos los parámetros de entrada y salida que requiera la función.

Dentro de la función podremos definir los parámetros de entrada salida y el tipo de cada uno de ellos. Nos referiremos a ellos por su #nombre.

Supongamos que queremos hacer una función que sume dos números. Quedaría algo así:

```
L   #VALOR_A          // Carga VALOR_A en acumulador 1 (ac1)
L   #VALOR_B          // Carga VALOR_B en ac1, VALOR_A pasa a ac2
+I                    // ac1 = ac1 + ac2
T   #RESULTADO        // Transfiere ac1 a RESULTADO
```

Para llamarla utilizaremos el siguiente código:

```
CALL FC 1
  VALOR_A := 7
  VALOR_B := 8
  RESULTADO := MW10
```

Esto sumará 7 + 8 y dejará el resultado en la marca 10.

Consultar "Introducción a Step7" para ver más información.

Instrucciones de rotación y desplazamiento de bits

Disponemos de 6 instrucciones para rotar y desplazar bits:

SLW	Desplazar palabra a la izquierda.
SRW	Desplazar palabra a la derecha.
SLD	Desplazar doble palabra a la izquierda.
SRD	Desplazar doble palabra a la derecha.
RLD	Rotar doble palabra a la izquierda.
RRD	Rotar doble palabra a la derecha.

Cada una de estas instrucciones requieren de un parámetro que indica el número de bits a desplazar o rotar. Estas operaciones se realizan sobre el acumulador 1, por lo que antes de ejecutarlas tendremos que utilizar la instrucción L.

```
L       MW 0
SRW     2
T       MW 2
```

El programa carga la palabra de la marca 0, la desplaza dos bits a la derecha y la guarda en la marca 2.

Al desplazar los bits sobrantes se descartan, y los nuevos se rellenan con 0. Por otro lado, al rotar los bits que salen por un lado entran por el otro.

Cálculos y conversiones

El entorno Step7 provee de una exhaustiva biblioteca de funciones para poder hacer operaciones básicas como sumas, restas, multiplicaciones, divisiones, etc.

AWL	KOP	Operación
+I	ADD_I	Suma
-I	SUB_I	Resta
*I	MUL_I	Multiplicación
/I	DIV_I	División
MOD	MOD_DI	Resto de la división
INC x	-	Sumarle al acumulador 1 x
DEC x	-	Restarle al acumulador 1 x

También dispone de un buen número de funciones para convertir tipos de datos.

BTI	Cambia de BCD a entero de 16 bits.
ITB	Cambia de entero de 16 bits a BCD.
DTB	Cambia de entero de 32 bits a BCD.
BTD	Cambia de BCD a entero de 32 bits.
DTR	Cambia de entero doble a real.
ITD	Cambia de entero a entero doble (32 bits).
RND	Redondeo
RND+	Redondea por arriba
RND-	Redondea por abajo
TRUNC	Trunca la parte decimal. Queda una entero de 16 bits

Asimismo, disponemos de funciones para operar con números reales.

AWL	KOP	Operación
+R	ADD_R	Suma
-R	SUB_R	Resta
*R	MUL_R	Multiplicación
/R	DIV_R	División
	ABS	Valor absoluto
	SQRT	Raíz cuadrada
	SQR	Calcula el cuadrado
	LN	Logaritmo natural o neperiano
	EXP	Elevar a *e*
	SIN	Seno
	COS	Coseno
	TAN	Tangente
	ASIN	Arco-seno
	ACOS	Arco-coseno
	ATAN	Arco-tangente

Instrucciones que afectan el RLO

Son las siguientes:

NOT	Invierte el RLO
SET	Pone el RLO a 1
CLR	Pone el RLO a 0
SAVE	Guarda el RLO en el registro BIE

Detección de flancos

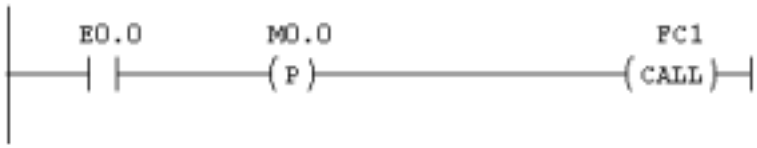
Cuando queramos detectar que una entrada o variables ha pasado de 0 a 1 o de 1 a 0 utilizaremos las instrucciones FP (flanco positivo) y FN (flanco negativo).

Por ejemplo:

```
U       E 0.0
FP      M 0.0
CC      FC 1
BE
```

Ejecutará la función FC1 cuando la entrada E0.0 pase de 0 a 1.

En KOP esto se programaría así:



Mapa de memoria de la CPU 314 IFM

20 ED	E124.0 a E126.3	8 bits (1 byte)
16 SD	A124.0 a A125.7	8 bits (1 byte)
4 EA	EW128 a EW 134	16 bits (1 word)
1 SA	AW128	16 bits (1 word)

Marcas	M0 a M255	256
Contadores	Z0 a Z63	64
Temporizadores	T0 a T127	128

Ejemplos de uso de temporizadores y contadores

Ejemplo 1: Contar y descontar cada segundo

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.

Programa 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
```

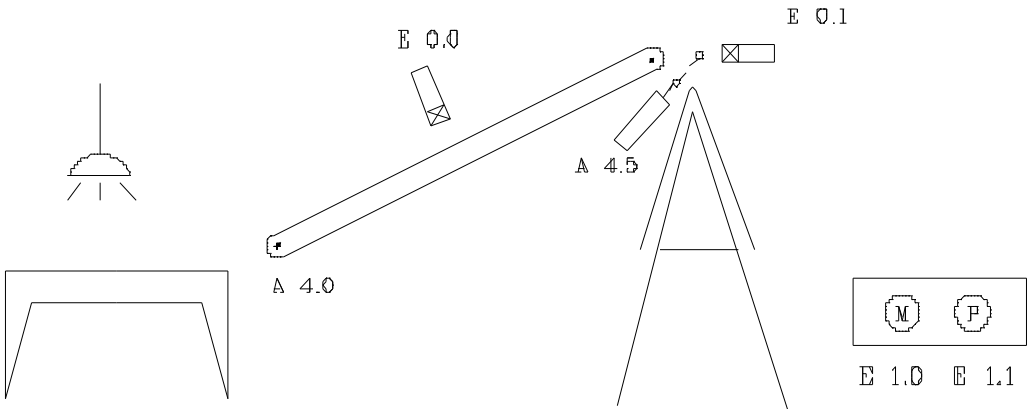
De esta manera podríamos temporizar tiempos más grandes de los que me permiten los temporizadores. Con un temporizador lo máximo que puedo temporizar es 999 unidades de la base de tiempos 3. Esto viene a ser dos horas y pico. Si quiero temporizar más tiempo, puedo generarme yo una base de tiempos con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

Ejemplo 2: Fábrica de curtidos

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.

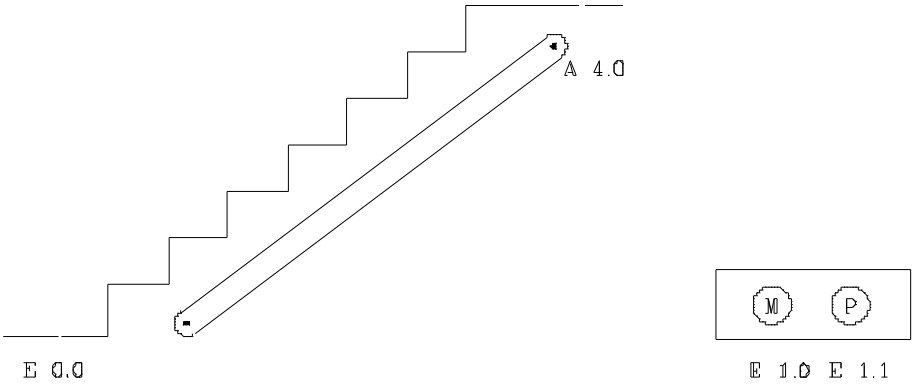
Programa en AWL

```
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.

Ejemplo 3: Escalera automática

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.

Programa en AWL

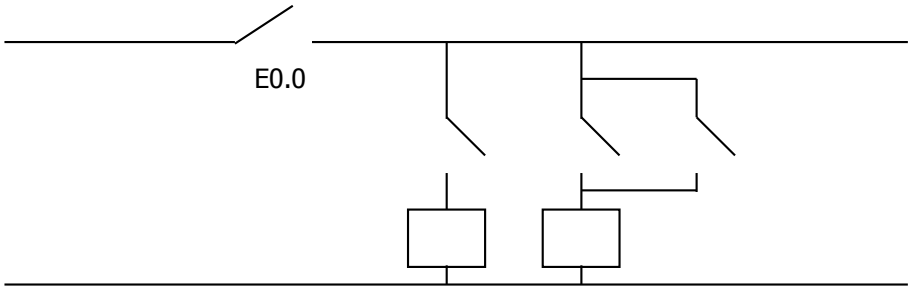
```
U     E     1.0     // Si le damos al pulsador de marcha
S     M     0.0     // Activa la marca 0.0
U     M     0.0     // Si está activa la marca 0.0
U     E     0.0     // Y llega una persona
L     S5T#5S        // Cuenta 5 segundos
SA    T     1       // A partir de cuando empiece a subir
U     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
```

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.

El Master Control Relay 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 sí 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.

Delante de cada paréntesis siempre 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.

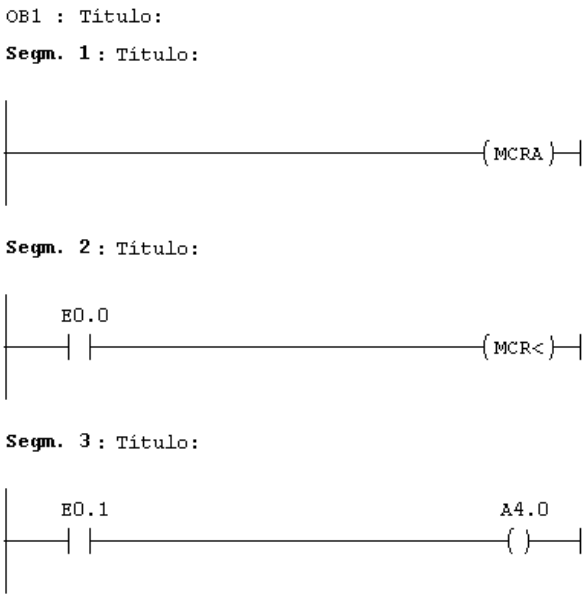
Esto no ocurre si el trozo de programa se deja de leer por cualquier otra causa. Si hacemos un salto o una meta, o programamos otros bloques, cuando no se ejecuta una parte del programa, las salidas se quedan como estaban. Si estaban activas cuando dejó de ejecutarse ese trozo de programa, continúan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

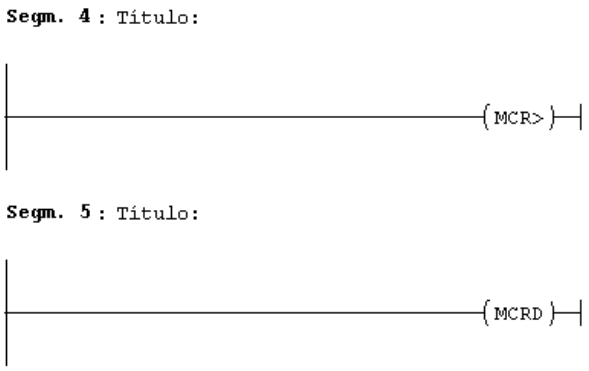
En el ejemplo que hemos hecho, la última parte no está dentro de ningún paréntesis, aunque si que está dentro de la activación del MCR. Esta parte de programa si que funciona siempre.

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