2 Operaciones de byte, palabras y dobles palabras.

Contenidos del Capítulo 2

2 Operaciones de byte, palabras y dobles palabras.

2.1 Indice de ejercicios:

19.- Falncos.

20.- Ajuste de valores analógicos.

21.- Ejemplo con UDT.

22.- Operaciones lógicas con palabras.

23.- Ejemplo de alarmas.

24.- Seleccionar tiempos.

EJERCICIO 1: CARGAR Y TRANSFERIR DATOS

TEORÍA.

INSTRUCCIONES DE CARGA Y TRANSFERENCIA.

Hasta ahora hemos trabajado con bits. Escribíamos instrucciones del tipo:

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

Si quisiésemos hacer esto mismo pero con todas las entradas y todas las salidas, podríamos hacerlo bit a bit, o trabajando directamente con la palabra de entradas y la palabra de salidas en bloque.

Hemos visto como podemos acceder a los bits de entrada, salida o marcas.

E 0.0 A 4.0 M 0.0

También podemos acceder a bytes (8 bits)

EB 0 AB 4 MB 0

Esto es el byte de entradas 0, el byte de salidas 4 y el byte de marcas 0.

También podemos acceder a palabras (16 bits).

EW 0 AW 4 MW 0

Esto son la palabra de entradas 0, la palabra de salidas 4 y la palabra de marcas 0.

También podemos trabajar con dobles palabras.

ED 0 AD 4 MD 0

Esto son la doble palabra de entradas 0, la doble palabra de salidas 4 y la doble palabra de marcas 0.

Para trabajar con todo esto tenemos las instrucciones de carga y transferencia.

Instrucción de carga:	L
Instrucción de transferencia:	T

Cuando cargamos algo, lo cargamos en el acumulador 1. Cuando transferimos, lo que hacemos es coger lo que hay en el acumulador 1 y llevarlo a donde le decimos. El acumulador 1 es un registro interno del autómata de 32 bits.

EJERCICIO 1: INSTRUCCIONES DE CARGA Y TRANSFERENCIA

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Instrucciones de carga y transferencia.

Vamos a hacer una prueba de lo que podemos hacer con está instrucción.

Queremos que lo que metamos por las entradas nos salga por las salidas.

Para ello tenemos dos formas de hacerlo. Una forma sería pasar los bits uno a uno. Por ejemplo:

```
U     E     0.0
=     A     4.0
U     E     0.1
=     A     4.1
.....
.....
```

Otra forma de hacerlo sería utilizando las instrucciones de carga y transferencia.

El programa quedaría del siguiente modo:

```
L     EW    0
T     AW    4
BE
```

Con esto lo que estamos haciendo es una copia de las entradas en las salidas.

Hay que tener en cuenta que las operaciones de carga y transferencia son incondicionales. Esto quiere decir que no debemos hacer programas como el siguiente:

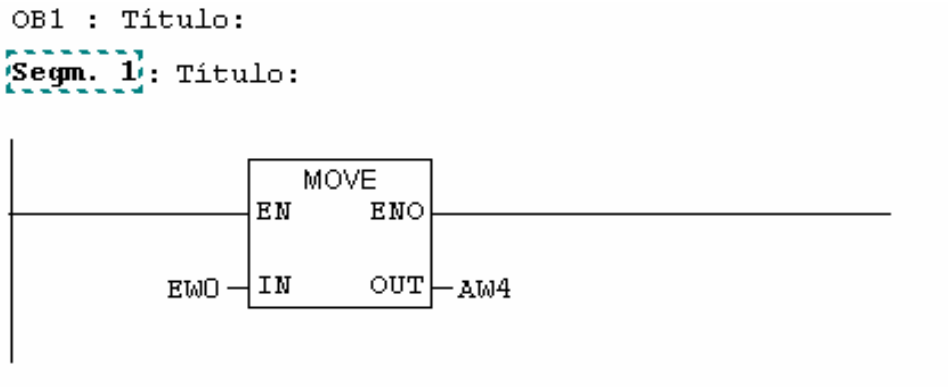
```
U    E    0.0
L    MW   0
T    AW   4
```

En este caso el autómata no daría error pero no haría caso a la entrada. Tanto si estuviera como si no, la operación de carga y transferencia se ejecutaría.

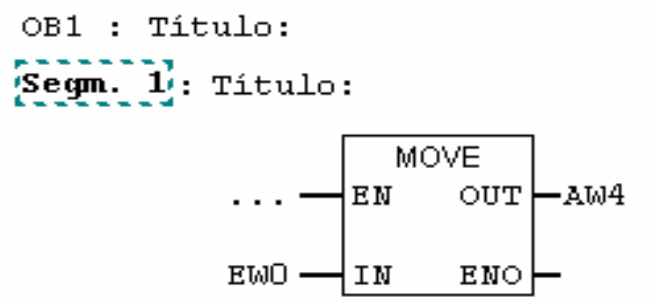
Esta operación también la podemos hacer en KOP y en FUP con la instrucción MOVE.

Veamos como lo haríamos:

Solución en KOP



Solución en FUP



En esta instrucción podemos encontrar alguna diferencia entre la programación en AWL y las programaciones gráficas KOP y FUP.

En lista de instrucciones dijimos que las instrucciones L y T eran incondicionales. Ahora en la programación en KOP y en FUP, vemos que en las

cajas de programación aparecen unos parámetros que se llaman EN y ENO. Lo que nosotros pongamos en EN será la condición que habilite la función que estamos dibujando a continuación. Esto nos sirve igual para la instrucción MOVE que para cualquier otra. En KOP y en FUP todo lo podemos hacer condicional. Sólo tenemos que rellenar en el parámetro EN la condición que queramos.

Además tenemos el parámetro ENO. Si rellenamos aquello con algún bit, éste se activará cuando se esté realizando correctamente la función que tenemos programada. En este caso la carga y transferencia.

EJERCICIO 2: EJERCICIO DE METAS.

TEORÍA

FORMATOS DE CARGA. SALTOS CONDICIONAL E INCONDICIONAL.

Nosotros podemos introducir valores en las palabras, bytes o dobles palabras. Veamos como podemos cargar valores.

Para introducir datos en el acumulador, lo podemos hacer con diferentes formatos. Hagamos un repaso de los que hemos visto hasta ahora.

L	C#.....	Cargar una constante de contador.
L	S5T#.....	Cargar una constante de tiempo.
L	.....	Cargar una constante decimal.

Veamos cuatro formatos nuevos:

L	2#.....	Cargar una constante binaria.
L	B#16#.....	Cargar 8 bits en hexadecimal.
L	W#16#.....	Cargar 16 bits en hexadecimal.
L	DW#16#.....	Cargar 32 bits en hexadecimal.

El autómata por defecto, va leyendo una instrucción detrás de otra. Mientras no le digamos lo contrario esto es lo que va a hacer. Nosotros podemos intercalar en nuestro programa instrucciones de salto. De este modo le decimos al autómata que se vaya a ejecutar un trozo de programa determinado, al que nosotros le habremos puesto un nombre (metas). El nombre de las metas, tiene que estar compuesto como máximo de cuatro dígitos de los cuales el primero necesariamente tiene que ser una letra.

Para saltar, tenemos dos instrucciones:

SPA: Salto absoluto.
SPB: Salto condicional.

Estas instrucciones nos sirven para saltar a trozos de programa que se encuentren dentro del mismo bloque en el que nos encontramos. Con estas instrucciones no podemos ir de un bloque a otro.

Nos sirven para todo tipo de bloques de programación.

EJERCICIO 2: EJERCICIO DE METAS

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Definición de metas. Final BEA. Formatos binario y hexadecimal. Saltos.

Queremos hacer dos contadores. Si el contador nº 1 tiene un valor más grande que el contador nº 2, queremos que se enciendan todas las salidas. Si el contador nº 1 tiene un valor más pequeño que el contador nº 2, queremos que se enciendan solamente las salidas pares. Si los dos contadores tienen el mismo valor queremos que se apaguen todas las salidas.

SOLUCIÓN 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
```

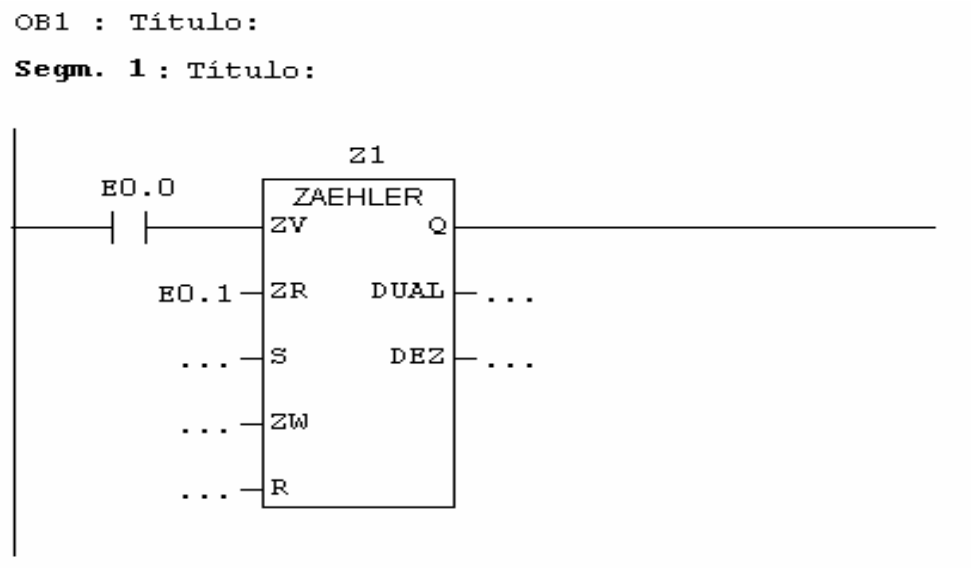
```
            T     AW    4
            BEA

MAYO:       L     W#16#FFFF
            T     AW    4
            BEA
IGUA:       L     0
            T     AW    4
            BE
```

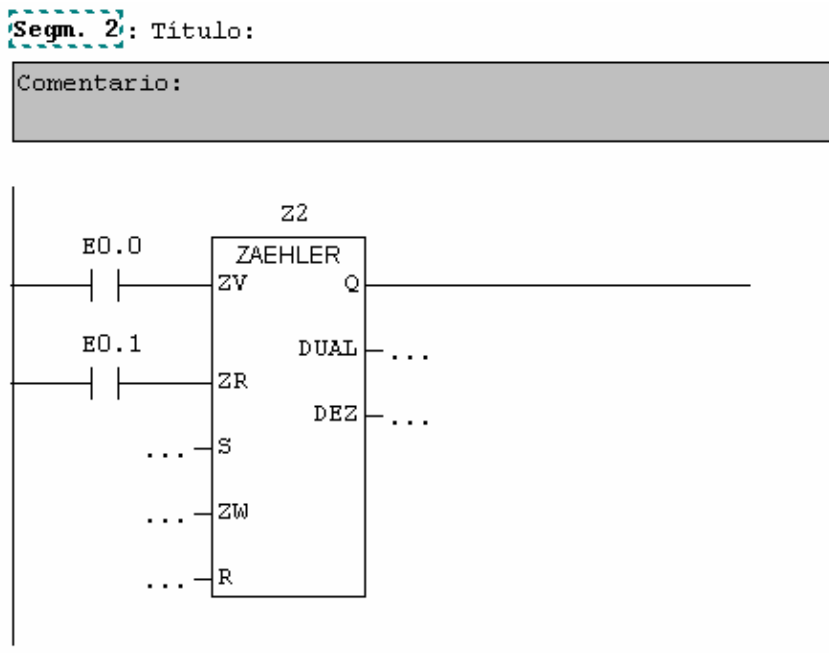
Hay que tener en cuenta siempre poner un BEA antes de empezar con las metas, y otro BEA al final de cada una de las metas.

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.

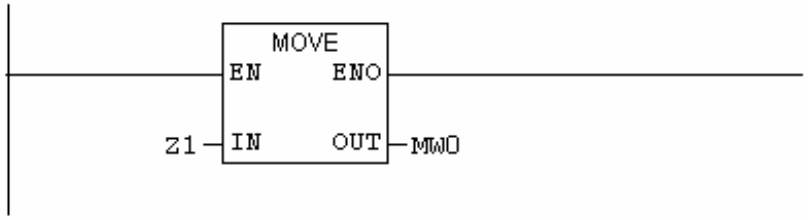
Veamos las soluciones en KOP y en FUP.

Solución en KOP:

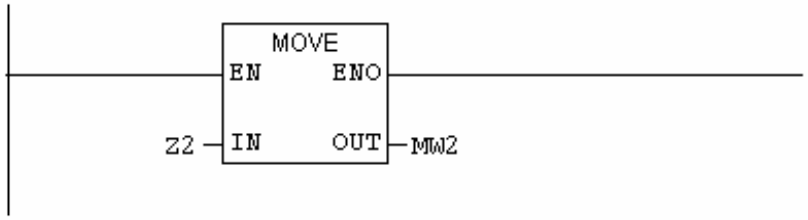




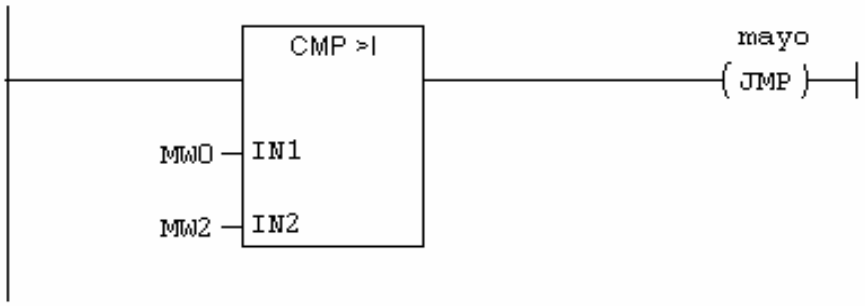
Segm. 3 : Título:



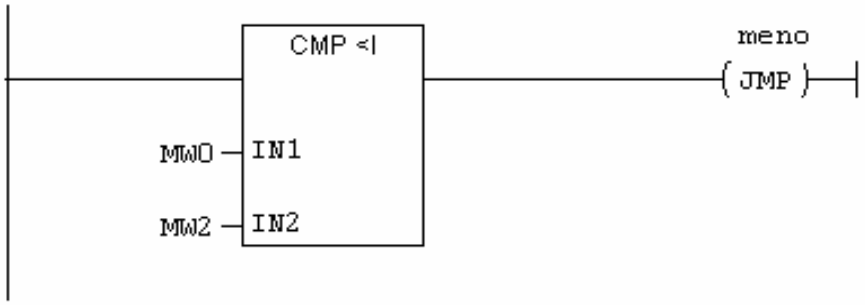
Segm. 4 : Título:



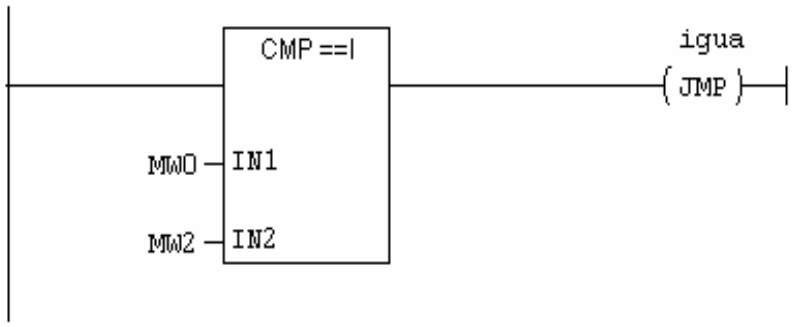
Segm. 5 : Título:



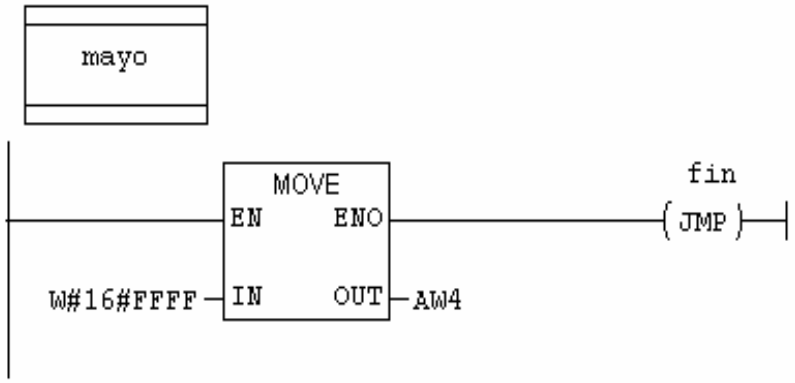
Segm. 6 : Título:



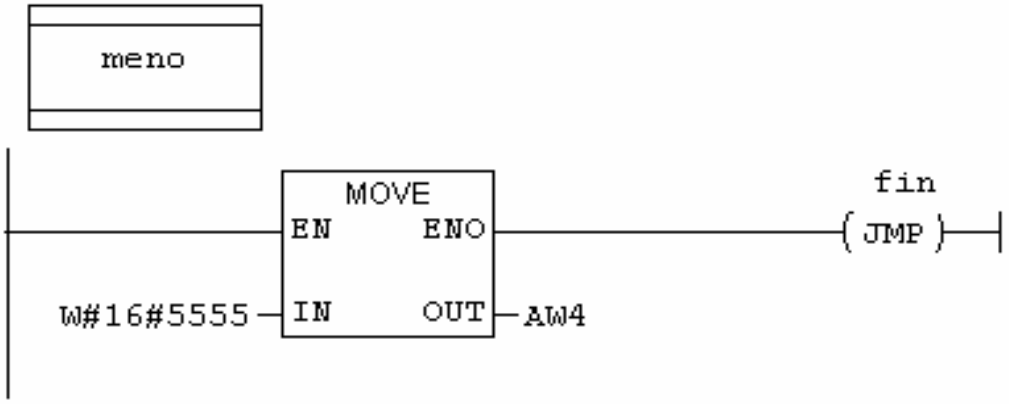
Segm. 7 : Título:



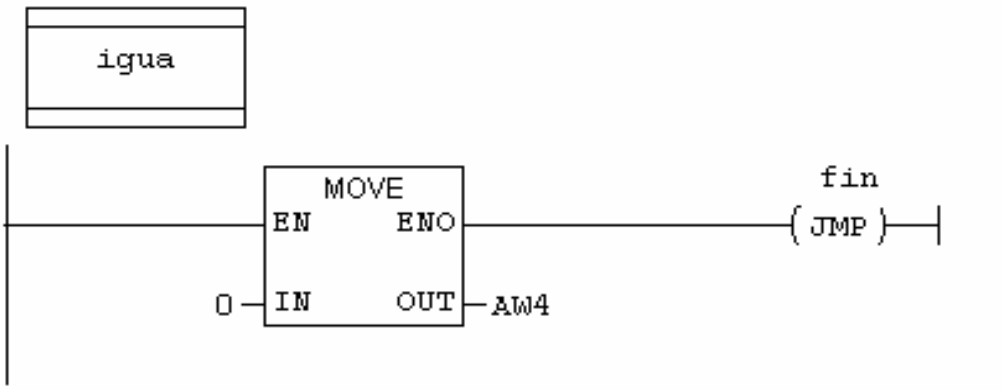
Segm. 8 : Título:



Segm. 9 : Título:



Segm. 10: Título:

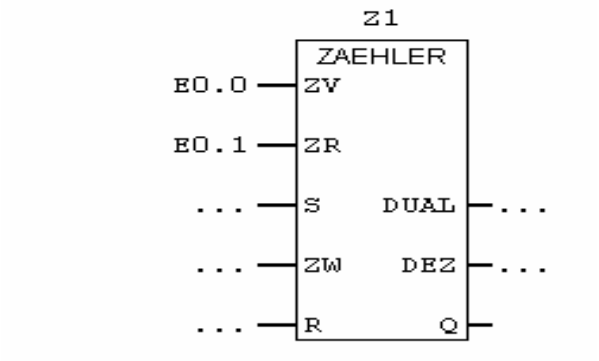


Segm. 11 : Título:

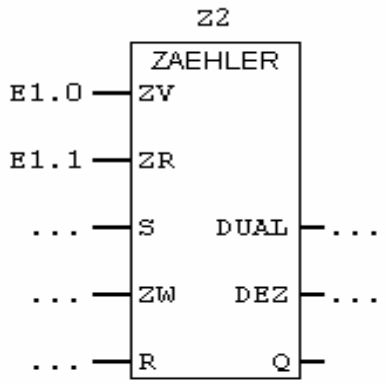
Solución en FUP:

OB1 : Título:

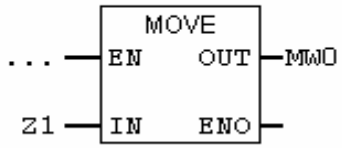
Segm. 1 : Título:



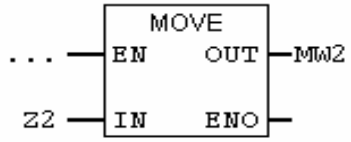
Segm. 2 : Título:



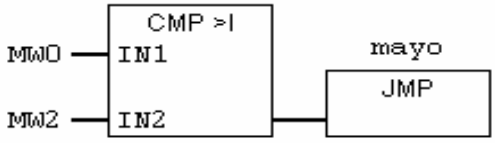
Segm. 3 : Título:



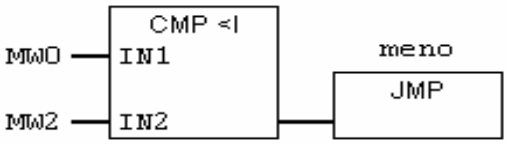
Segm. 4 : Título:



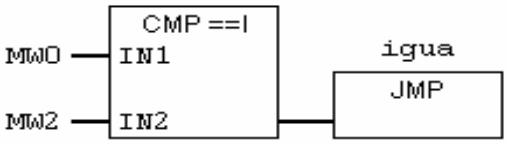
Segm. 5 : Título:



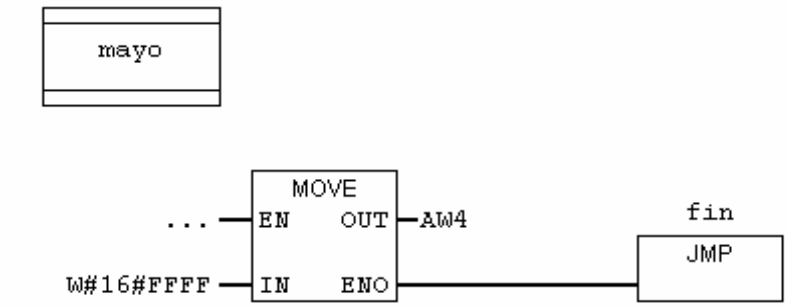
Segm. 6 : Título:



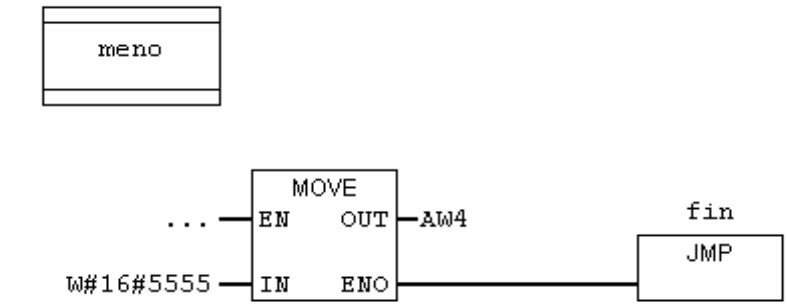
Segm. 7 : Título:



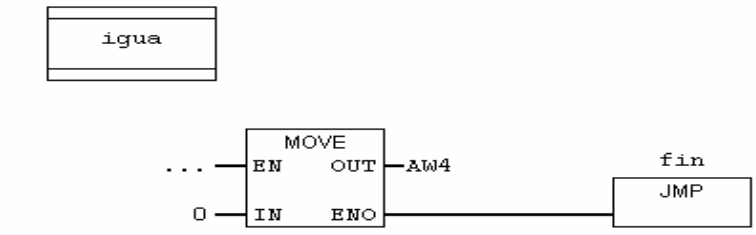
Segm. 8 : Título:

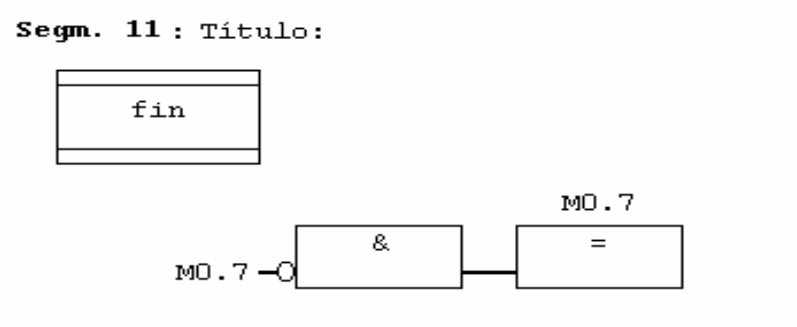


Segm. 9 : Título:



Segm. 10 : Título:




Segm. 11 : Título:
fin
M0.7
&
=
M0.7

Hemos visto las soluciones en KOP y en FUP.

En estos dos lenguajes, no hubiera sido necesario el uso de las metas. La carga y la transferencia la podíamos haber hecho condicional con los resultados de las comparaciones de los valores de los contadores. El programa hubiera quedado más corto. Lo que hemos hecho aquí ha sido exactamente el mismo programa en cada uno de los tres lenguajes para ver como hacemos lo mismo en los diferentes lenguajes.

EJERCICIO 3: TRABAJAR CON BLOQUES DE DATOS.

TEORÍA.

CREACIÓN DE UN BLOQUE DE DATOS

Un bloque de datos es una tabla en la que almacenamos datos. Aquí no vamos a programar nada. Los datos los podemos almacenar en distintos formatos.

Para guardar un dato, tenemos que poner nombre a la variable, definir el formato en el que lo queremos, y el valor inicial.

El valor inicial siempre es el mismo. Su propio nombre ya lo indica, es el valor inicial. Cuando este valor cambie, se almacenará en otra columna que es el valor actual. Aunque al abrir el DB no veamos esta columna, tenemos que tener en cuenta que también existe.

Para poderla ver, tenemos que ir al menú Ver > datos.

Veremos los datos actuales de la programadora o del autómata dependiendo de si estamos en ONLINE o en OFFLINE. Los datos actuales de la programadora siempre serán los mismos que los iniciales. (Siempre y cuando no los cambiemos nosotros). La programadora no ejecuta el programa. Los datos actuales del autómata, serán los últimos datos que haya puesto la ejecución del programa.

Tenemos que tener en cuenta que esta columna de valor actual también la transferimos tanto de ONLINE a OFFLINE como al contrario.

EJERCICIO 3: TRABAJAR CON BLOQUES DE DATOS

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Saber crear un bloque de datos.

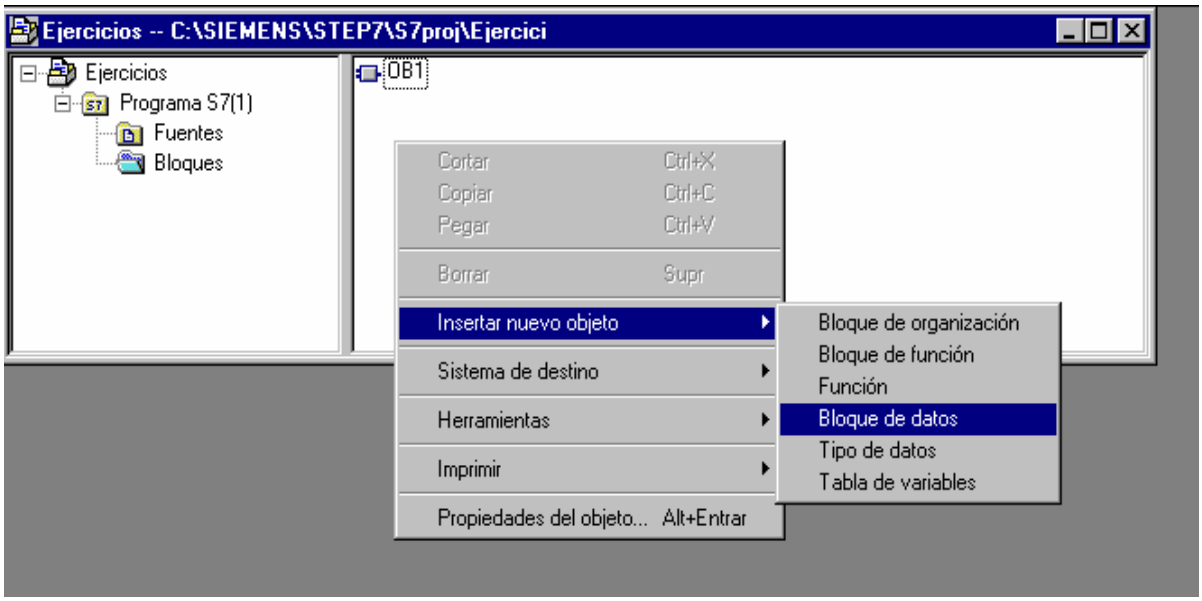
Vamos a crear un bloque nuevo. Vamos a hacer un bloque de datos. En estos bloques lo único que vamos a introducir son datos. No vamos a programar nada. Va a ser una tabla con datos los cuales podemos leer y podemos escribir nuevos valores en los datos.

Para crear un DB vamos al administrador de SIMATIC. Nos situamos en la parte izquierda encima de donde pone bloques. Lo podemos hacer tanto en ONLINE como en OFFLINE. En la parte derecha tenemos todos los bloques que tenemos creados hasta ahora.

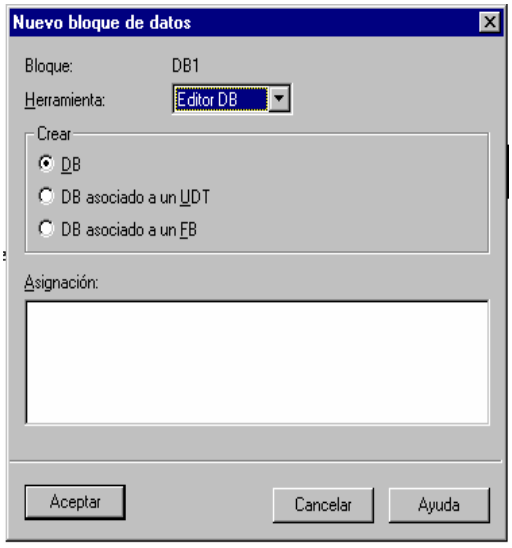
Tenemos que insertar un bloque nuevo. Pinchamos en la parte derecha con el botón derecho del ratón y aparece una ventana donde le decimos que queremos insertar un nuevo objeto.

También lo podemos hacer en el menú de “Insertar” que tenemos en la barra de herramientas.

Insertamos un bloque de datos.



Cuando entramos en el nuevo DB vemos que nos ofrece tres posibilidades. Podemos hacer un DB asociado a una FB, podemos hacer un DB asociado a un UDT, y podemos hacer un DB simple.



En principio lo que queremos hacer es un DB simple.

Una vez lo tenemos creado veremos que sale el icono del DB junto con los demás bloques.

Le podemos dar el número de DB que nosotros queramos. Por ejemplo podemos hacer el DB 1.

Entramos en el DB y vamos a rellenarlo.

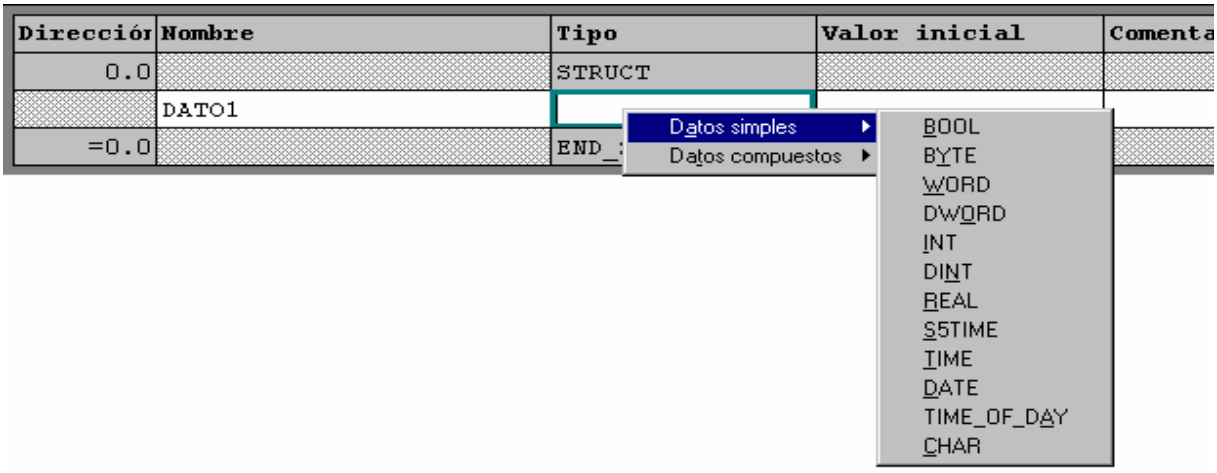
El DB es una tabla en la que tenemos que rellenar los datos. Tenemos varios campos para rellenar.

Dirección	Nombre	Tipo	Valor inicial	Comentario
0.0		STRUCT		
=0.0		END_STRUCT		

El primer campo que tenemos es la dirección. De esto no nos tenemos que preocupar. Es automático. Conforme vayamos creando datos, se le van asignando las direcciones que corresponden.

El siguiente campo es el nombre. Es obligatorio dar un nombre a cada dato que vayamos a introducir.

A continuación tenemos que definir el tipo de datos que va a ser. Si no sabemos como definir un tipo de datos pinchamos en la casilla correspondiente con el botón derecho del ratón y podemos elegir el tipo de datos que queremos.



A continuación tenemos que definir el valor inicial. Tenemos que poner el valor en el formato que corresponde. Si no sabemos el formato o no queremos ningún valor inicial, por defecto nos pone un 0, pero escrito en el formato correcto.

Ya tenemos el primer dato definido.

Vamos a crear la siguiente tabla:

VALOR_1	INT	0
VALOR_2	REAL	3.0
VALOR_3	WORD	W#16#0

Esto en un DB quedaría de la siguiente manera:

Dirección	Nombre	Tipo	Valor inicial	Comentario
0.0		STRUCT		
+0.0	VALOR_1	INT	0	
+2.0	VALOR_2	REAL	0.000000e+000	
+6.0	VALOR_3	WORD	W#16#0	
=8.0		END_STRUCT		

Ya tenemos un DB. Vamos a ver ahora como trabajamos con él.

Vamos a leer el valor 3.0, y vamos a escribir un valor en VALOR_3.

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

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

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

También podemos dar un nombre al DB y acceder a los datos por su nombre. Por ejemplo en el DB que hemos creado antes podríamos acceder a los datos del siguiente modo:

```
L    DATOS.VALOR_1
```

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

Vamos a ver como leemos un valor y lo metemos en una marca, y como escribimos un valor y luego lo vamos.

Vamos a leer el valor 3.0 que tenemos en VALOR_2 y vamos a ponerlo en la doble palabra de marcas 100. Tiene que ser en una doble palabra de marcas porque es un número real.

```
AUF   DB    1
L     DBD   2
T     MD    100
```

Ahora vamos a escribir un valor en VALOR_3.

```
L     8
T     DB1.DBW6
BE
```

Hemos metido un 8 en este valor.

Vamos a entrar en el DB y vamos a ver como vemos este nuevo dato que hemos introducido.

Entramos en el DB y lo único que vemos es la columna de valores iniciales. Aquí vamos que sigue habiendo un 0.

Si vamos al menú de Ver tenemos la opción de Ver > datos.

Vemos que el DB tiene una nueva columna que es la de datos actuales. Aquí veremos que tenemos el nuevo valor. Esta pantalla no refresca valores. Hace una lectura cuando abrimos la ventana. Si hacemos algún cambio y queremos visualizarlo, tendremos que cerrar y abrir de nuevo la pantalla.

Para ver el dato que acabamos de introducir, tenemos que hacer la operación en ONLINE. Si vamos a ver datos en OFFLINE, veremos los datos actuales de la CPU del PC. El PC no ha ejecutado programa. Veremos que los valores iniciales son los mismos que los actuales.

Tenemos que tener en cuenta que estos valores se guardan en disco duro. Cada vez que transfiramos el programa bien de OFFLINE a ONLINE o viceversa, también se transfieren los valores actuales.

Si queremos dejar el DB como estaba al principio con sus valores iniciales para comenzar el proceso de nuevo, vamos al menú de edición y tenemos la posibilidad de reinicializar bloque de datos. Los valores actuales se cambian por los valores iniciales. A partir de ahí hace lo que se le diga por programa.

Si el DB ha adquirido unos valores distintos de los iniciales los guarda en esta columna de valores actuales. Cada vez que transferimos al autómata el DB también estamos transfiriendo esta columna de valores actuales aunque no la veamos.

Si queremos volver a los valores iniciales no tenemos más remedio que reinicializar el DB y transferir estos cambios al autómata.

Si estamos programando en KOP o en FUP, tendremos que utilizar la instrucción MOVE para leer o escribir datos en el DB.

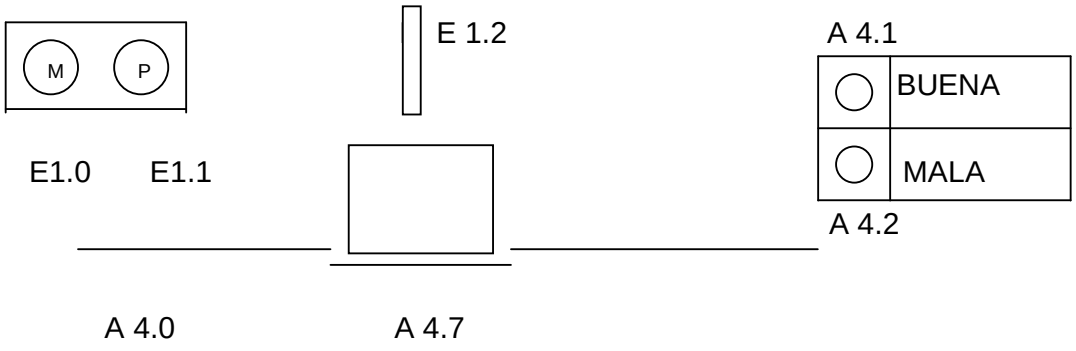
EJERCICIO 5: PESAR PRODUCTOS DENTRO DE UNOS LÍMITES

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Creación y manejo de DB´s.

Tenemos una cinta transportadora que lleva los productos. Lo que queremos es que los productos que se salgan del peso estipulado tanto por arriba como por abajo, se desechen.

Además tenemos unas luces indicadoras de producto bueno o de producto malo. Queremos que mientras está la pieza debajo de la célula detectora, se encienda la luz correspondiente.



Para ello vamos a tener en un módulo de datos los límites de peso tanto superior como inferior.

Con el byte 0 de entradas vamos a simular los pesos de las piezas que circulan por la cinta. Queremos que cuando la pieza tenga un peso comprendido entre los límites correctos se encienda una luz que nos indique que la pieza es correcta. Cuando pase una pieza que su peso se salga de los límites queremos que

nos indique una luz que la pieza es mala y a la vez que se abra una trampilla para que la pieza salga de la cinta.

```
SOLUCIÓN AWL
      U     E     1.0          //Cuando le demos al botón de marcha
      S     A     4.0          //Pon en marcha la cinta transportadora
      U     E     1.1          //Cuando le demos al botón de paro
      R     A     4.0          //Para la cinta
      U     E     1.2          //Cuando pase la pieza por la balanza
      SPB   M001
      R     A     4.1          // Si no hay pieza manten todo apagado.
      R     A     4.2
      R     A     4.7          //Salta a la meta 1
      BEA                      //Si no hay pieza  termina el programa
M001: L     EB    0            //Carga el peso de la pieza
      L     DB1.DBW0           //Carga el límite superior de peso
      <I                       //Si la pieza pesa menos que el limite sup.
      =     M     0.0          //Activa la marca 0.0
      L     EB    0            //Carga el peso de la pieza
      L     DB1.DBW2           //Carga el límite inferior
      >I                       //Si la pieza pesa más que el limite inf.
      =     M     0.1          //Activa la marca 0.1
      U     M     0.0          //Si está activa la marca 0.0
      U     M     0.1          //Y está activa la marca 0.1
      S     A     4.1          //Enciende la luz de pieza buena
      R     A     4.2          //Apaga la luz de pieza mala
      NOT                      //En caso contrario
```

```
S     A     4.2          //Enciende la luz de mala
R     A     4.1          //Apaga la luz de buena
U     A     4.2          //Si la pieza es mala
=     A     4.7          //Abre la trampilla de desecho
BE
```

Hemos visto como hacer una carga y una transferencia condicionada por una entrada. Para hacerlo condicional lo podemos hacer de dos maneras. Bien lo hacemos como en este ejemplo utilizando metas, o bien lo hacemos con llamadas a otros módulos de modo condicional.

Si se cumple la condición salta al módulo que le digamos y allí tendremos la carga y la transferencia. Si no se cumple la condición no salta al módulo en cuestión y por tanto no lee las instrucciones de carga y transferencia.

Para completar el ejercicio, nos falta hacer el módulo de datos donde vamos a guardar el límite superior y el límite inferior de pesado para las piezas.

Vamos a crear el DB1.

En el DB1 vamos a crear dos palabras de datos. Tenemos que definir un tipo para cada uno de los datos. En este caso van a ser dos números enteros. Diremos que son de tipo (INT).

También podemos suponer que son valores reales. Para ello lo deberíamos haber tenido en cuenta a la hora de programar.

Para comparar números reales tenemos otra instrucción, y además los números reales se guardan en dobles palabras, no en palabras simples.

Para guardar los datos deberíamos haber utilizado la doble palabra de datos 0 y la doble palabra de datos 4.

Veamos como quedaría el DB.

Dirección	Nombre	Tipo	Valor inicial	Comentario
0.0		STRUCT		
+0.0	PESO_MAXIMO	INT	25	
+2.0	PESO_MINIMO	INT	20	
=4.0		END_STRUCT		

Ejercicio propuesto: Hacer este mismo ejercicio en KOP y en FUP.

Todas las instrucciones necesarias se han visto en ejercicios anteriores.

EJERCICIO 6. PROGRAMACIÓN ESTRUCTURADA.

TEORÍA

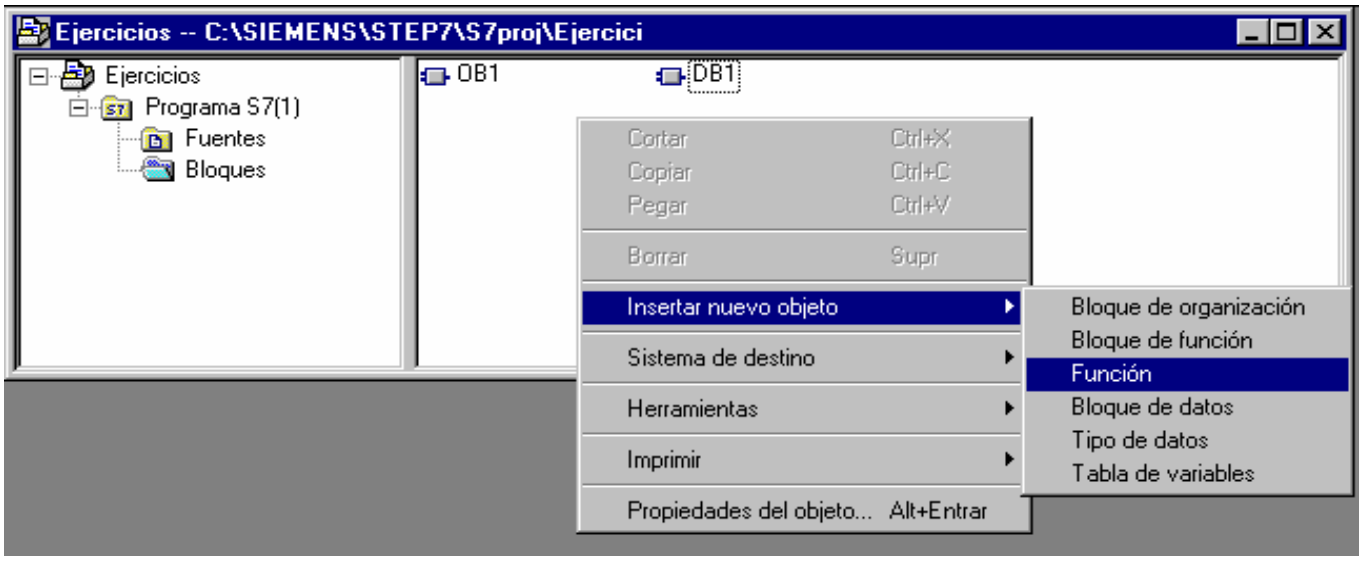
EJEMPLO DE PROGRAMACIÓN ESTRUCTURADA

Creación de FC´s sin parámetros.

Queremos que si la entrada E 0.0 está activa funcione una parte del programa. Y que si no está activa funcione la otra parte del programa.

Para ello vamos a hacer dos módulos FC y desde el OB indicaremos cuando tiene que acceder a uno y a otro.

Para crear las 2 FC´s, lo hacemos del mismo modo que hicimos para crear un nuevo DB. En este caso insertamos una función.



Una vez tengamos creadas las dos FC´s ya podemos entrar en ellas y hacer su programa.

SOLUCIÓN AWL:

OB1:

```
U     E     0.0
CC    FC    1          //Llamada condicional.
UN    E     0.0
CC    FC    2          //Llamada condicional.
BE
```

FC1:

```
U     E     1.0
=     A     4.0
BE
```

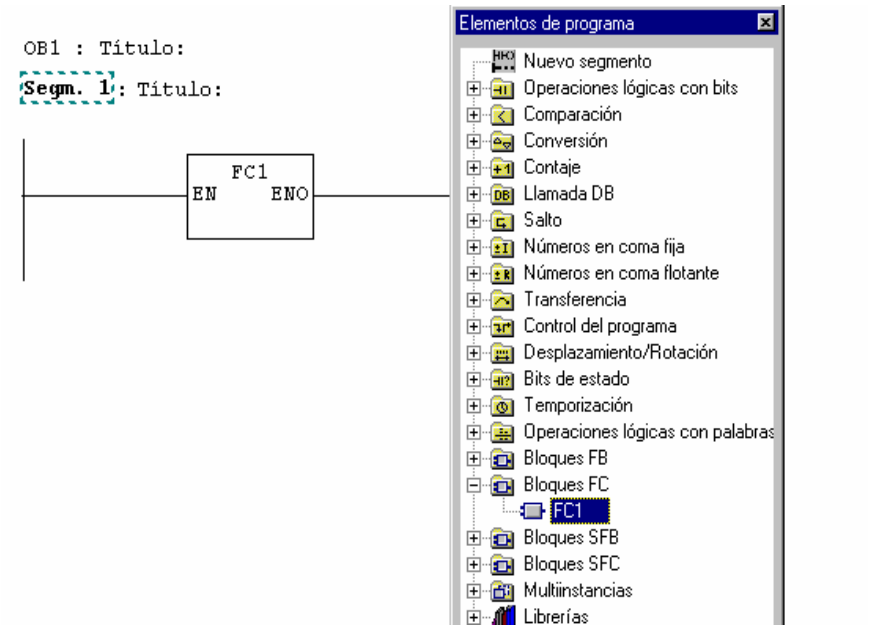
FC2:

```
U     E     1.1
=     A     4.1
BE
```

Hemos visto que para hacer llamadas condicionales a FC´s sin parámetros en AWL, tenemos la instrucción CC. Si quesiéramos hacer una llamada a una FC sin parámetros pero de modo incindicional, utilizaríamos la instrucción UC.

Veamos como podríamos hacer esto en KOP y en FUP.

Para hacer la llamada, tenemos que abrir el catálogo, y seleccionar la función a la que queremos llamar.



Si queremos hacer la llamada de modo condicional, solo tenemos que rellenar el parámetro EN con la condición que queramos.

En FUP haríamos exactamente lo mismo.

EJERCICIO 6: CREAR UN DB CON LA SFC 22

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Creación de un DB.

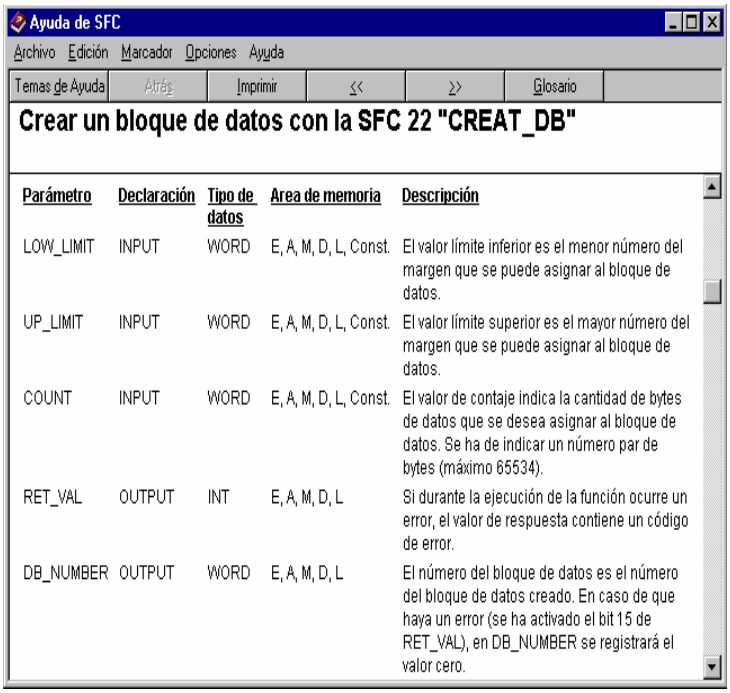
Hemos visto como podemos hacer un DB dato a dato. Si son pocos datos los que queremos hacer, los hacemos uno a uno, pero también tenemos la posibilidad de crear un DB con un número predefinido de palabras de datos.

Por ejemplo si queremos hacer un DB con 100 palabras de datos para rellenar después, no es necesario hacerlo dato a dato y poner nombre a todos ellos.

Para hacerlo disponemos de una SFC que ya lleva integrada la CPU.

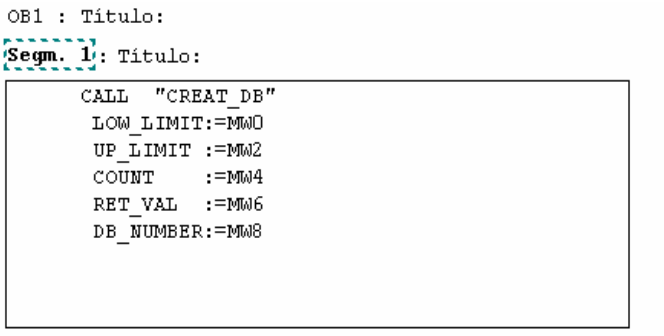
Simplemente tenemos que llamar a la SFC 22 y rellenar los parámetros que nos pide.

Las SFC las tenemos en la CPU. Si vamos a los bloques de ONLINE, veremos que tenemos una SFC 22. Para ver lo que hace y cómo tenemos que utilizarla, solo tenemos que pinchar encima del icono correspondiente con el icono de ayuda que tenemos en el menú de herramientas. (?).



En esta ayuda nos explica lo que hace la SFC 22, y nos dice que al llamarla se nos va a pedir una serie de parámetros. Nos explica lo que significa cada parámetro.

Veamos como haríamos una llamada a esta SFC y veamos como rellenaríamos los parámetros.



Para llamar a una FC o SFC que tiene parámetros, utilizamos la instrucción CALL. En este caso, se va a crear un DB con número comprendido entre el valor de la MW0 y el valor de la MW2. La cantidad de datos que va a tener va a ser la que diga la MW4. Si ocurre algún error en la creación del DB, lo tendremos almacenado en la MW6. El número del último DB que tenemos creado, lo tendremos en la MW8.

Si el DB que queremos crear ya existe, no se crea ninguno y la función nos da un error. Esto es lo que ocurrirá si la llamamos de modo incondicional. En la primera llamada nos generará el DB. En posteriores ciclos de scan cuando intente crearlo de nuevo nos dará un error en la MW6. Vamos a ver lo que tenemos en la MW6, y vamos a buscar en la ayuda lo que significa este valor.

Para hacer el programa en KOP o en FUP, tenemos que seleccionar del catálogo la SFC 22 como en ejercicios anteriores.

EJERCICIO 7: DESPLAZAMIENTO DE BITS

TEORÍA

INSTRUCCIONES DE ROTACIÓN Y DESPLAZAMIENTO

Tenemos 6 instrucciones para desplazar y rotar 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.

Estas instrucciones se gastan seguidas de un número que indica la cantidad de posiciones que se tienen que mover los bits.

Veamos un ejemplo y cómo funcionaría:

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

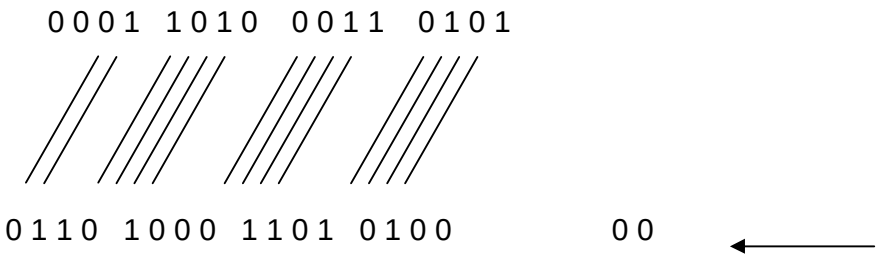
Supongamos que en la MW 0 tenemos lo siguiente: 0001 1010 0011 0101

Al ejecutar este programa veamos lo que ocurriría en el acumulador:

Con la primera carga lo que tendríamos en el acumulador sería:

0001 1010 0011 0101

La siguiente operación que hacemos en el acumulador es el desplazamiento:



Al desplazar, se mueven todos los bits dos posiciones a la izquierda, y por la parte derecha entran dos ceros.

Si en lugar de desplazar lo que hacemos es rotar, lo que ocurre es que lo que se pierde por un lado entra por el lado contrario. Esta operación la hace en el acumulador. Sólo podemos rotar dobles palabras porque el acumulador es de 32 bits.

EJERCICIO 7: DESPLAZAMIETNO DE BITS

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Desplazamiento y rotación de bits.

Queremos que un bit de un byte de salidas vaya corriendo hacia arriba y hacia abajo. Comenzaremos encendiendo la salida 4.0 y queremos que ese bit se desplace hacia abajo. La velocidad que le daremos será de una posición cada medio segundo.

Después probaremos a hacer lo mismo pero en lugar de con el byte de salidas, lo haremos con la palabra de salidas. Veremos que con un solo desplazamiento no podemos conseguir el mismo movimiento de antes.

El movimiento que en el papel vemos de derecha a izquierda, no coincide con el movimiento de arriba hacia abajo de la palabra de salidas.

SOLUCIÓN EN AWL:

OB1

```
U     E     0.0          //Cuando le demos a la entrada 0.0
S     A     4.0          //Encenderemos la salida 4.0
UN    M     0.0          //Hacemos unos pulsos de 500 milisegundos
L     S5T#500MS          //de la marca 0.0
SE    T     1
U     T     1
=     M     0.0
U     M     0.0          //Cuando llegue un pulso de la marca 0.0
UN    M     0.1          //Y no esté la marca 0.1
CC    FC    1                  //Se irá a ejecutar la FC 1
```

```
U     A     4.7      //Cuando se encienda la salida 4.7
S     M     0.1      //Activa la marca 0.1
U     A     4.0      //Si está encendida la salida 4.0
R     M     0.1      //Desactiva la marca 0.1
U     M     0.0      //Si está la marca 0.0
U     M     0.1      //Y está la marca 0.1
CC    FC    2        //Ejecuta la FC 2
BE
```

FC2 : DESPLAZAMIENTO A IZQUIERDAS

A esta función saltaremos cuando toque desplazar a izquierdas.

Segm. 1: Desplazamiento a izquierdas.

Desplazamiento desde la A 4.0 hasta la A 4.7

```
L     AW     4
SRW   1
T     AW     4
BE
```

FC1 : DESPLAZAMIENTO A DERECHAS

A esta función saltaremos cuando toque desplazar a derechas.

Segm. 1: Desplazamiento a derechas

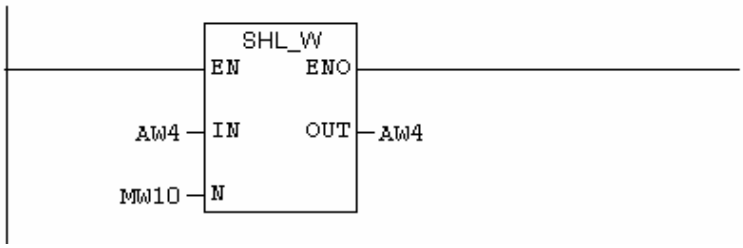
Desplazamiento desde la A 4.7 hasta la A 4.0

```
L     AW     4
SRW   1
T     AW     4
BE
```

A esta función saltaremos cuando toque desplazar a izquierdas.

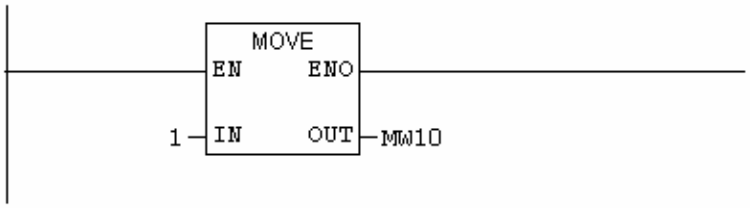
Segm. 1: Desplazamiento a izquierdas.

Desplazamiento desde la A 4.0 hasta la A 4.7

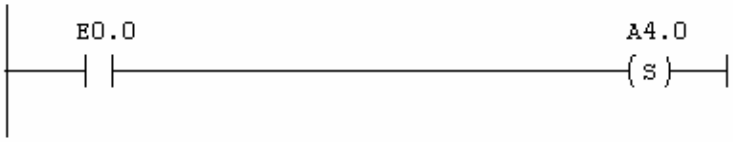


OB1 : Título:

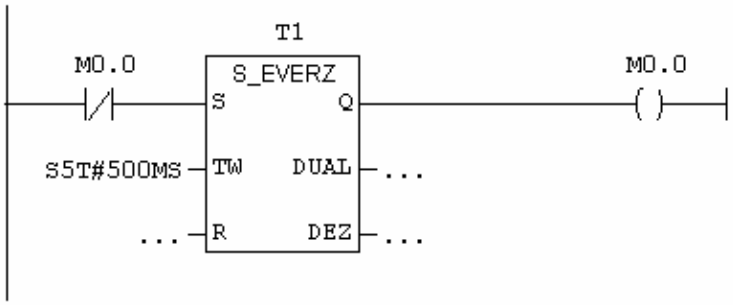
Segm. 1 : Título:



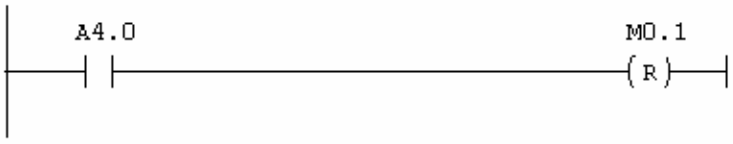
Segm. 2 : Título:



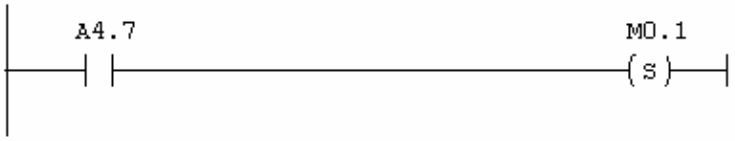
Segm. 3 : Título:



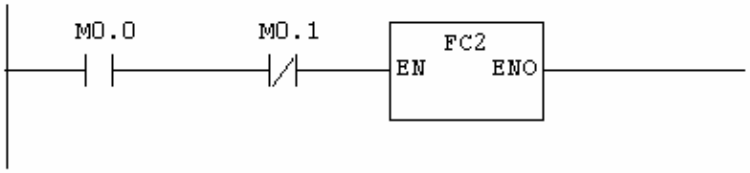
Segm. 4 : Título:



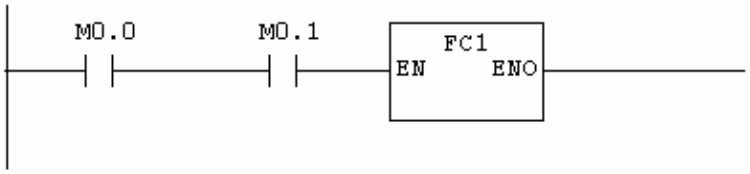
Segm. 5 : Título:



Segm. 6 : Título:



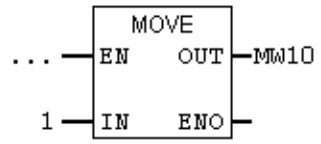
Segm. 7 : Título:



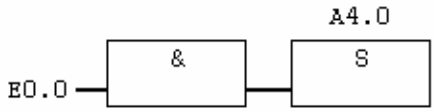
Solución en FUP

OB1 : Título:

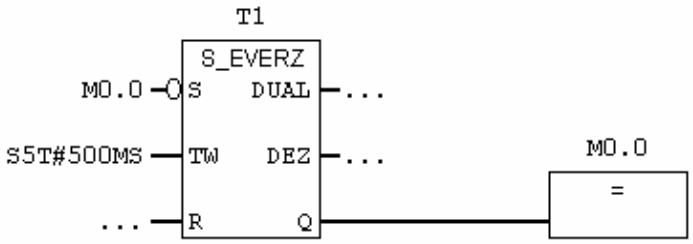
Segm. 1 : Título:



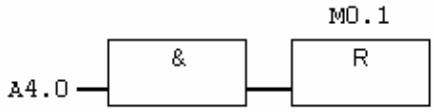
Segm. 2 : Título:



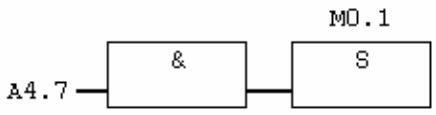
Segm. 3 : Título:



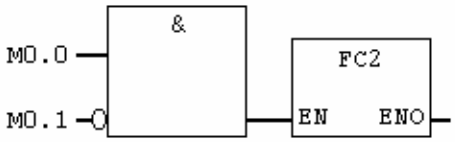
Segm. 4 : Título:



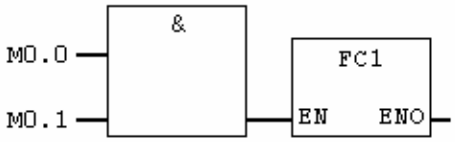
Segm. 5 : Título:



Segm. 6 : Título:



Segm. 7 : Título:

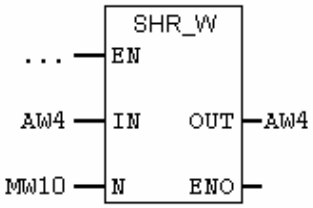


FC1 : DESPLAZAMIENTO A DERECHAS

A esta función saltaremos cuando toque desplazar a derechas.

Segm. 1: Desplazamiento a derechas

Desplazamiento desde la A 4.7 hasta la A 4.0

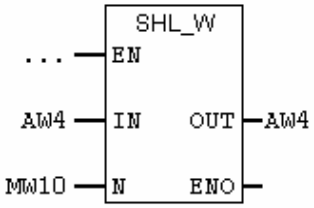


FC2 : DESPLAZAMIENTO A IZQUIERDAS

A esta función saltaremos cuando toque desplazar a izquierdas.

Segm. 1: Desplazamiento a izquierdas.

Desplazamiento desde la A 4.0 hasta la A 4.7

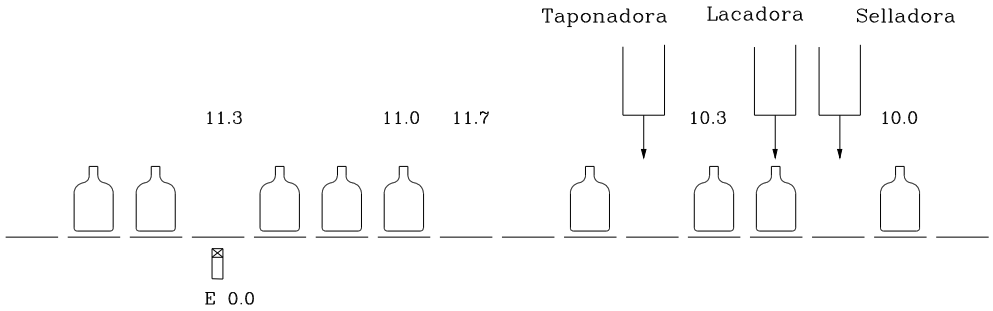


EJERCICIO 8: PLANTA DE EMBOTELLADO

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Desplazamiento y rotación de bits. Programación estructurada.

Tenemos una planta de embotellado distribuida de la siguiente manera:



Vemos que en la línea tenemos tres máquinas. Una taponadora, una lacadora y una selladora.

Queremos que cuando las botellas lleguen debajo de las máquinas, éstas se pongan en marcha, pero si llega un hueco no queremos que las máquinas actúen.

Las botellas pasan de posición a posición cada segundo. Con la célula fotoeléctrica que tenemos detectamos cuando pasa una botella o cuando pasa un hueco.

Vamos a resolver el problema utilizando 4 FC´s y una OB1.

En cada FC vamos a programar una de las operaciones que necesitamos realizar. Después desde la OB 1 diremos cuando necesitamos realizar cada una de la operaciones.

En la primera FC vamos a hacer un generador de pulsos de un segundo para poder mover las botellas.

```
FC1 : GENERADOR DE PULSOS
Segm. 1: Título:
      UN    M      0.0
      L     S5T#1S
      SE    T      1
      U     T      1
      =     M      0.0
```

En la siguiente FC vamos a meter 1 en el lugar donde van las botellas. Es decir cada vez que la célula fotoeléctrica detecte que ha pasado una botella colocará un 1 en su lugar correspondiente. Luego desde otra FC ya veremos cómo tenemos que mover ese 1.

```
FC2 : PONER BIT CUANDO LLAGA LA BOTELLA
Segm. 1: Desplazamiento a izquierdas.
      U     E      0.0
      S     M     11.3
```

En otra FC vamos a programar el desplazamiento de los bits.

```
FC3 : DESPLAZAMIENTO DE LAS BOTELLAS
Segm. 1: Título:
      L     MW    10
      SRW   1
      T     MW    10
```

En otra FC vamos a hacer la activación de las máquinas cuando lleguen las botellas debajo de ellas.

```
FC4 : ACTIVACIÓN DE LAS MÁQUINAS
Segm. 1: Título:
      U     M     11.4
      =     A      4.0
      U     M     11.2
      =     A      4.1
      U     M     11.1
      =     A      4.2
```

Ahora nos queda organizar cuándo se tienen que efectuar cada una de estas FC. Desde el OB1 diremos cuando se tiene que ejecutar cada una de las FC.

```
OB1 : Título:
Segm. 1: Título:
      UC    FC     1
      UC    FC     2
      U     M      0.0
      CC    FC     3
      UC    FC     4
```

En la programación estructurada, hacemos cada cosa en bloque. De este modo la programación se hace mucho más sencilla.

Para programar cada una de las FC´s sólo nos preocupamos de programar la maniobra o la acción que queremos hacer. No nos preocupamos de cuando la tiene que hacer.

Después desde la OB1 ya pensaremos cuando se tiene que ejecutar cada una de las operaciones que hemos definido.

Ejercicio propuesto: resolver este ejercicio en KOP y en FUP con las instrucciones que hemos visto en ejercicios anteriores.

EJERCICIO 9: DIFERENCIA ENTRE FC CON Y SIN PARÁMETROS

TEORÍA

LLAMADAS A LAS FC CON Y SIN PARÁMETROS

Cuando programamos una FC tenemos que llamarla desde algún sitio para que se ejecute.

A continuación vamos a hacer una FC que sume 2 + 3, y otra FC que sume A + B.

Cuando estamos sumando 2 + 3, estamos haciendo una FC sin parámetros. Siempre suma lo mismo y el resultado siempre será 5.

Para llamar a esta FC podemos utilizar dos instrucciones:

```
UC   FC   1        Llamada incondicional.
CC   FC   1        Llamada condicional.
```

De este modo se ejecuta la función de manera condicional o incondicional, con los datos que hemos definido dentro de la propia FC.

Cuando estamos sumando A + B, estamos haciendo una FC parametrizable. Cada vez que la llamemos, tendremos que darle unos valores a A y a B para que el autómata sepa lo que tiene que sumar.

Para darle estos valores, lo que tenemos que hacer es llamar a la FC con CALL FC 1.

Al hacer la llamada de este modo, el programa accede a la FC y nos pide todos los parámetros de entrada y salida que hayamos definido. Entonces nosotros le damos los valores que queremos que sume cada vez que le llamamos.

Una vez la tenemos programada, la podemos llamar tantas veces como queramos. La instrucción CALL que utilizamos para llamar a las FC con parámetros, es siempre incondicional.

Veamos a continuación un ejemplo de FC con parámetros.

EJERCICIO 9: DIFERENCIA ENTRE FC CON Y SIN PARÁMETROS

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Diferencia entre llamadas a FC con y sin parámetros.

Vamos a hacer dos FC similares. Van a ser dos bloques que van a realizar la misma operación pero en una de ellas va a ser con unos valores fijos, y en la otra va a ser parametrizable.

Veamos el primero de los casos:

Para crear una FC nueva, vamos a la ventana del administrador de SIMATIC. Pinchamos en la parte izquierda encima de bloques. En la parte derecha pinchamos con botón derecho y cogemos la opción de insertar nuevo objeto. Insertamos una función.

Una vez tenemos la FC1 hecha, tenemos que programarla.

```
FC1 : Título:
Segm. 1: Título:
      L     2
      L     3
      +I
      T     AW     4
```

Esta función va a sumar 3 + 2 y el resultado nos lo va a dejar en la palabra de marcas AW4.

Siempre que se ejecute está función nos sumará los mismos valores.

Para que la función se ejecute, alguien tendrá que llamarla. Si no, la CPU sólo ejecuta la OB1.

Desde la OB1 tendremos que hacer una llamada a una FC sin parámetros.

Tenemos dos posibilidades de hacer la llamada. Podemos hacer una llamada condicional o incondicional.

Podríamos hacer lo siguiente:

```
OB1
U     E     0.0
CC    FC    1
BE
```

O bien hacer lo siguiente:

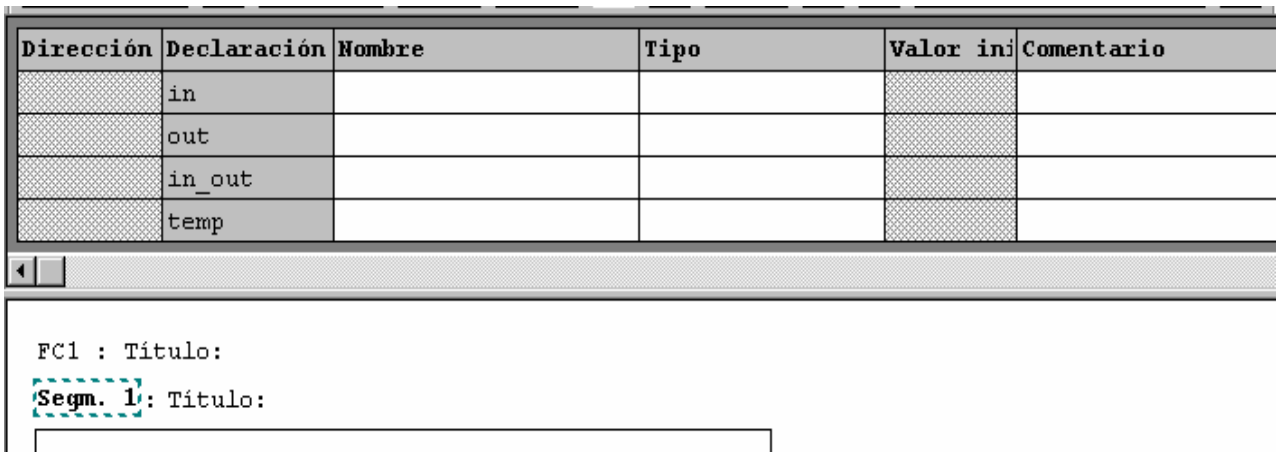
```
OB1
UC    FC    1
BE
```

Son los dos tipos de llamada que tenemos para funciones sin parámetros. Con el primer bloque accedería a la FC 1 siempre y cuando estuviera la entrada E 0.0 activa.

En el segundo caso, accedería siempre a la FC 1.

Vamos a ver como podríamos hacer una FC parametrizable. Esto quiere decir que no siempre tendría que sumar los mismos valores.

Cuando entramos en la FC vemos que en la parte superior hay una tabla. En esta tabla es donde tenemos que definir los parámetros.



Vemos que tenemos varios campos para definir cosas.

Tenemos una línea de IN, otra de OUT, otra línea de IN/OUT y otra línea de TEMP.

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

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

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

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

En nuestro caso vamos a tener dos valores de entrada y un valor de salida.

Confeccionamos la tabla de la siguiente manera:

Dirección	Declaración	Nombre	Tipo	Valor ini	Comentario
0.0	in	VALOR_A	INT		
2.0	in	VALOR_B	INT		
4.0	out	RESULTADO	INT		
	in_out				

A cada una de las variables, tenemos que definirle un tipo. En este caso son todas de tipo entero.

A la hora de programar haremos lo siguiente:

```
FC1 : Título:
Segm. 1: Título:
      L     #VALOR_A
      L     #VALOR_B
      +I
      T     #RESULTADO
```

La almohadilla no la escribimos nosotros. Esto nos indica que es una variable local. Sólo la podemos utilizar con su nombre dentro de la FC 1 que es donde la hemos definido.

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

A la hora de llamar a la función tendremos que decirle cuanto vale cada valor.

```
OB1 : Título:
Segm. 1: Título:
      CALL  FC     1
       VALOR_A  :=7
       VALOR_B  :=8
       RESULTADO:=MW10
```

Vemos que para hacer una llamada a una FC con parámetros, hay que utilizar la instrucción CALL.

Cuando utilizamos esta instrucción el programa va a ver los parámetros que tiene definidos esa función y nos pide los valores de estos parámetros.

Sólo nos pide los valores de lo que hayamos definido como entrada o como salida.

Lo que hayamos definido como temporal no nos lo pide. Lo que hemos definido como temporal, son resultados intermedios que nosotros no tenemos que darles ningún valor.

Al hacer esta llamada, la FC 1 se ejecuta con los valores que le hemos dado. Hace la suma correspondiente, y el resultado nos lo deja donde le hemos dicho.

Cuando sale de la FC 1, ya no se acuerda de los valores que le hemos dado. Cada vez que tengamos que ejecutar la FC 1, tendremos que darle de nuevo unos valores.

Si tenemos una FC con parámetros y hacemos una llamada como si fuese una FC sin parámetros la CPU se irá a STOP. Si una FC tiene parámetros, le tendremos que decir en la llamada cuanto valen los parámetros para que pueda ejecutar la operación. Si no es así no puede ejecutar nada.

Por ejemplo, en el caso que tenemos en este ejemplo, la FC realiza una suma. Si no le decimos los valores que tiene que sumar y dónde me tiene que dejar el resultado, no lo podrá hacer.

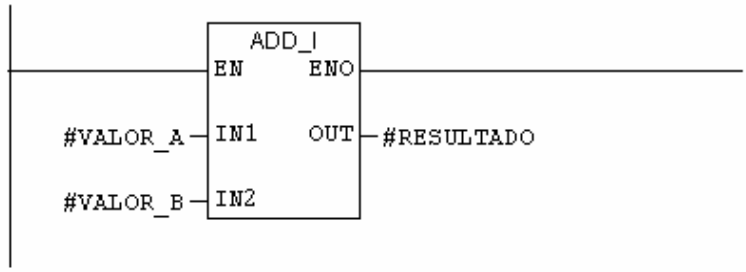
Veamos como hacemos una FC que sume en KOP y en FUP y vemos como la llamamos y le rellenamos los parámetros.

Solución en KOP

Dirección	Declaración	Nombre	Tipo	Valor ini	Comentario
0.0	in	VALOR_A	INT		
2.0	in	VALOR_B	INT		
4.0	out	RESULTADO	INT		
	in_out				

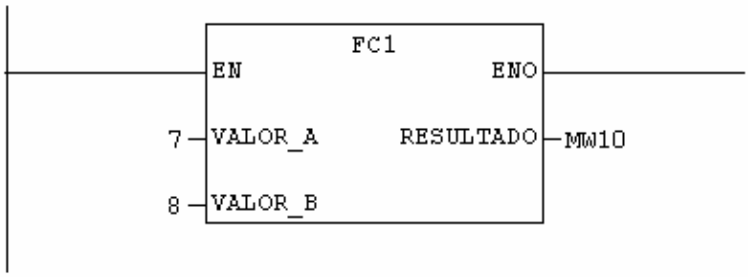
FC1 : Título:

Segm. 1: Título:



OB1 : Título:

Segm. 1: Título:



La solución en FUP sería igual.

Como podemos observar, en KOP y en FUP todo puede ser condicional. Basta con poner una condición en el parámetro EN.

EJERCICIO 10: SISTEMAS DE NUMERACIÓN

TEORÍA

Los sistemas de numeración más usuales en la programación son el sistema decimal, el sistema binario, y el sistema hexadecimal. También utilizaremos el sistema binario codificado en BCD.

Veamos las razones por las cuales se gastan estos sistemas de numeración.

El sistema decimal, lo utilizamos porque es el más corriente para nosotros. Es el que más dominamos y el que más fácil nos resulta de entender.

El sistema binario es el sistema que utilizan los PLC. Se compone de 1 y 0 que es realmente lo que ocurre en las máquinas. Hay tensión o no hay tensión.

El sistema hexadecimal lo utilizamos como sistema de paso. Traducir un número de binario a decimal y viceversa, nos lleva un tiempo. No es una operación inmediata para números grandes. En cambio para pasar de hexadecimal a binario y viceversa es una operación inmediata. Muchas veces sabemos lo que queremos en binario. Sabemos qué bits queremos que se activen pero no sabemos a qué número corresponde en decimal. Para ello utilizamos el sistema hexadecimal que tenemos que escribir menos y fácil de traducir y de entender.

Después tenemos el sistema binario codificado en BCD. Es un sistema binario (ceros y unos), pero tiene las ventajas del sistema hexadecimal a la hora de traducir a decimal que es lo que a nosotros nos resulta más cómodo.

SISTEMA DECIMAL:

Es el sistema al que estamos acostumbrados.

Se llama sistema decimal porque está hecho en base diez.

Veamos el significado de un número. Por ejemplo el 214.

2	1	4
10^2	10^1	10^0

El número 214 es: 2 * 100 + 1 * 10 + 4 * 1 = 214

Los dígitos 2, 1 y 4 corresponden a la cantidad de potencias de diez con el índice del lugar que ocupan, que tenemos que sumar.

SISTEMA BINARIO:

El sistema binario se forma exactamente igual que el sistema decimal sólo que con potencias de 2 y con dos dígitos.

Veamos a que número decimal corresponde el número binario 110.

1	1	0
2^2	2^1	2^0

Si hacemos la suma correspondiente nos queda:

1 * 4 + 1 * 2 + 0 * 1 = 6

El 110 binario, representa el 6 decimal. Vemos que tenemos que calcular una serie de potencias y después efectuar la suma. Si tuviéramos que traducir el número: 11101111000101011, nos llevaría un rato ver a qué número decimal corresponde.

Veamos cómo haríamos la operación inversa. Supongamos que tenemos el número 6 decimal y queremos saber a qué número binario corresponde.

```
6 | 2
0   3 | 2
      1   1 | 2
            1   0
```

Vamos dividiendo por dos hasta que el resultado nos de cero. Los restos de abajo hacia arriba son el número en binario. En este caso nos queda 110.

Vamos que también nos lleva un trabajo un poco costoso. Si quisiésemos traducir el número 29.856 nos llevaría un rato de trabajo.

Por ello utilizamos el sistema hexadecimal.

SISTEMA HEXADECIMAL:

El sistema hexadecimal se forma a partir de potencias de 16 y con 16 dígitos.

Como no conocemos 16 números para formar los 16 dígitos, utilizaremos letras. Los 16 símbolos del sistema hexadecimal son: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Veamos a qué número decimal corresponde el número hexadecimal 2B1.

2	B	1
16^2	16^1	16^0

La suma a realizar sería: $2 * 16^2 + B * 16 + 1 * 1 = 2 * 256 + 11 * 16 + 1 = 689$

Vemos que la sistemática es la misma que para el sistema decimal y que para el sistema binario.

Veamos cómo haríamos la operación inversa:

Tenemos el número 689 en decimal y queremos traducirlo a hexadecimal.

Hacemos lo mismo que con el sistema binario:

```
689 | 16
049   43 | 16
 01    11    | 2   16
             2     0
```

Vamos dividiendo sucesivamente entre dos hasta que el resultado de cero. Con los restos de abajo hacia arriba tenemos el número hexadecimal.

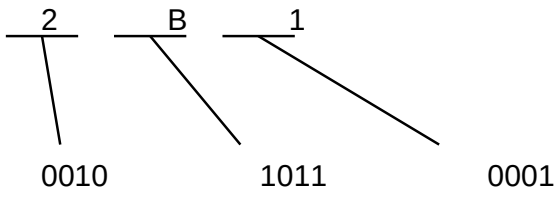
En este caso tenemos: 2 11 1. Como estamos en hexadecimal tenemos 16 dígitos. El 11 se representa por el dígito B. Luego el número hexadecimal que nos queda es 2B1 que es el número inicial que teníamos.

Vemos que esto también nos lleva una faena.

¿Dónde está la ventaja del sistema hexadecimal?

La ventaja está en el paso de hexadecimal a binario y viceversa.

Veamos como escribiríamos este mismo número en binario:



El número en binario sería el 0010 1011 0001. ¿Cómo hemos hecho la conversión? Vamos traduciendo dígito a dígito. Cada dígito hexadecimal corresponde a 4 dígitos en binario.

Tenemos que componer una suma de las potencias 2^3, 2^2, 2^1 y 2^0

Nunca vamos a tener potencias mayores. Estos cálculos los podemos hacer de cabeza. Siempre son sencillos.

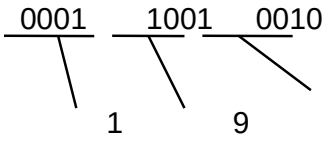
Veamos como haríamos la operación inversa.

Supongamos que tenemos el número binario 110010010 y queremos pasarlo a hexadecimal. Lo primero que tenemos que hacer es dividirlo en grupetos de 4:

0001 1001 0010

Si nos faltan dígitos completar un grupeto de 4 añadimos ceros a la izquierda.

Ahora sabemos que cada grupeto de 4 corresponde a un dígito hexadecimal:

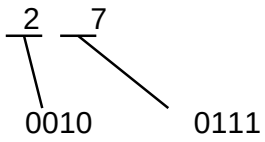


Tenemos el número 192 hexadecimal. Vemos que la conversión es muy sencilla y por grande que sea el número siempre la podemos hacer de cabeza.

SISTEMA BINARIO CODIFICADO EN BCD:

Crear un número codificado en BCD es muy sencillo desde el sistema decimal. Es la misma sistemática que pasar un número de hexadecimal a binario.

Veamos cómo pasamos el número decimal 27 a binario BCD.



El número 27 decimal es el número 0010 0111 en binario BCD.

Tenemos que tener en cuenta que no es lo mismo binario que binario BCD.

Por ejemplo veamos como traducimos el número 10 decimal a binario y a binario BCD.

Decimal 10 ---------- Binario 1010 ---------- Binario BCD 0001 0000

En el código BCD no existen todas las combinaciones de dígitos. En cambio en binario si.

Por ejemplo la combinación 1101 no corresponde a ningún número en BCD.

Si intentásemos traducir quedaría: 8 + 4 + 0 + 1 = 13

No podemos representar el número 13 en decimal con una sola cifra, luego 1101 no es ningún número en código BCD.

Si este número fuese binario normal correspondería al número 13 decimal.

EJERCICIO 11: CARGA CODIFICADA

TEORÍA

HACER CARGAS DE TEMPORIZADORES Y CONTADORES CODIFICADAS EN BCD

Si nosotros queremos cargar el valor de un contador o de un temporizador, utilizaremos instrucciones del tipo:

```
L     T     1
L     Z     3
```

Al ejecutarse este tipo de instrucciones, lo que ocurre es que se carga en el acumulador el valor que en ese momento tengan los contadores o temporizadores en binario.

Existen otras instrucciones de carga como las que siguen:

```
L     S5T#5S
L     C#10
```

Que cargan el valor en el acumulador, pero codificado en BCD.

Si queremos comparar dos cosas, tendrán que estar en el mismo formato.

Para poder hacer esto, existen instrucciones como las que siguen:

```
LC    T     1
LC    Z     3
```

Estas instrucciones hacen una carga de los valores que tengan los contadores o temporizadores, pero codificada en BCD.

Por ejemplo, si queremos comparar un temporizador con la cantidad de 10 segundos, tendremos que cargar en el acumulador el temporizador en cuestión y los 10 segundos para posteriormente comparar.

Para ello tengo que tener en cuenta el formato en el que se carga en el acumulador cada una de las cosas para comparar cosas iguales.

EJERCICIO 11: CARGA CODIFICADA

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Carga de temporizadores y contadores. Binario codificado BCD.

Cuando nosotros hacemos un temporizador, escribimos las instrucciones siguientes:

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

Posteriormente podemos utilizar la instrucción:

```
L     T     1
```

Con esto estamos cargando el valor que tenga en ese instante el temporizador. Podemos hacer lo que queramos con este valor.

Al escribir la instrucción L T 1, se carga en el acumulador el valor que tiene en ese instante el temporizador. Se carga este valor en binario.

Si nosotros quisiéramos comparar el temporizador con un valor de tiempo, haríamos lo siguiente:

```
L     T     1
L     S5T#3S
<I
=     A     4.1
```

Con esto lo que pretendemos hacer es que cuando el temporizador tenga un valor inferior a 3 segundos, se active la salida 4.1.

Lo que ocurre si hacemos esto es que estamos mezclando formatos.

Al escribir la instrucción L S5T#3S, la carga se hace en BCD.

Veamos lo que ocurriría si quisiésemos comparar un temporizador que va por 10 segundos con el valor de 10 segundos. En realidad estamos comparando dos valores que son iguales.

Al poner L T 1, el valor que estaríamos cargando sería:

10.....> 0000 0000 0000 1010

Esto es el número 10 en binario.

Al poner L S5T#10S, el valor que estaríamos cargando sería:

10.....> BCD 0000 0000 0001 0000

Esto es un 10 en BCD. Si ahora escribimos la instrucción de comparar, el autómata no sabe que cada cosa está en un formato. Nosotros le hemos introducido dos series de ceros y unos y ahora le preguntamos que si las series son iguales.

Evidentemente nos dirá que las series no son iguales. La segunda serie nos dirá que corresponde a un número mayor.

Para subsanar este problema, tenemos que decirle que haga las dos cargas en el mismo formato.

Tendríamos que programar de la siguiente manera:

```
L     S5T#10S
LC    T     1
=I
.....
```

De esta manera estamos haciendo una carga codificada del temporizador. El valor de los 10 segundos me lo hace por defecto en BCD. Ahora la carga del valor que tenga el temporizador también me la va a hacer en BCD porque le he dicho que me haga una carga codificada.

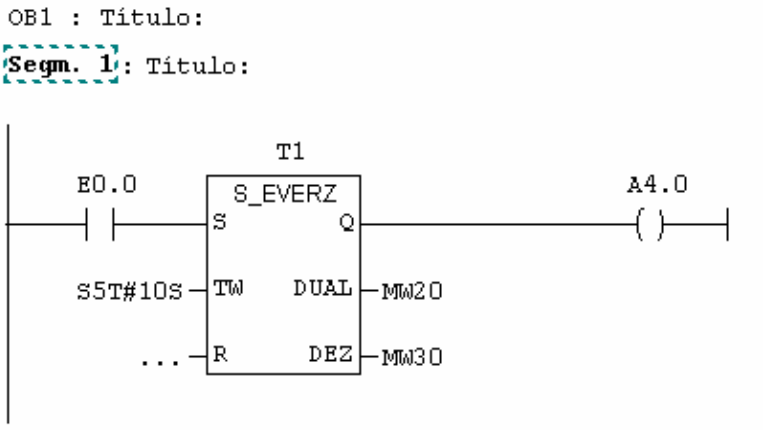
Ahora ya estamos comparando dos cosas en el mismo formato.

Vamos a hacer un ejemplo. Vamos a hacer un temporizador que temporice 8 segundos. Queremos que los primeros 3 segundos se active la salida 4.0 y los últimos segundos se active la salida 4.1.

SOLUCIÓN EN AWL

```
U     E     0.0
L     S5T#8S
SE    T     1
L     S5T#5S
LC    T     1
<I
=     A     4.0
>I
=     A     4.1
BE
```

Esta instrucción no existe como tal en KOP ni en FUP. Veamos como podríamos hacer estas comparaciones en estos lenguajes.



Como podemos ver, el propio temporizador lleva dos parámetros de salida llamados DUAL y DEZ. En este caso, en la MW20, tendremos el valor del temporizador codificado en BCD, y en la MW30 tendremos el valor del temporizador en binario.

EJERCICIO 12: OPERACIONES CON NÚMEROS ENTEROS

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Instrucciones correspondientes a los números enteros.

Vamos a hacer una FC en la que hagamos operaciones con números enteros. Vamos a hacer una FC parametrizable y le definiremos los parámetros que luego vayamos a utilizar en la operación. En la tabla de declaraciones le diremos de qué tipo son los valores.

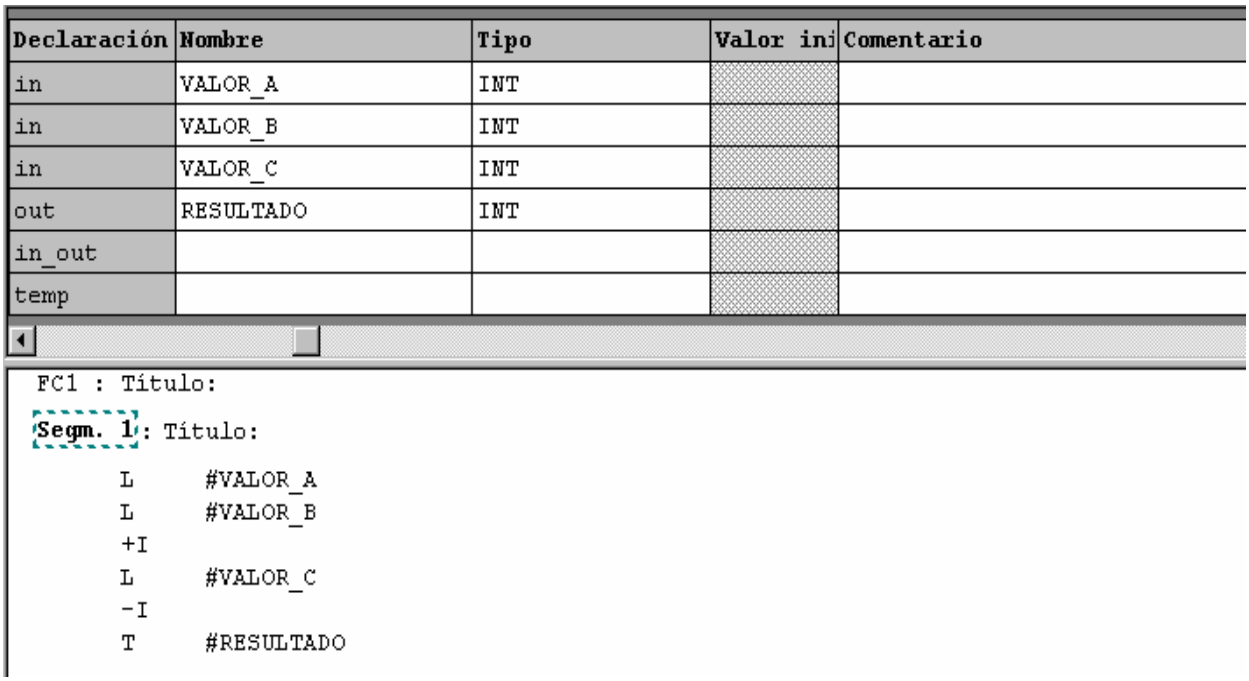
Tenemos que tener muy en cuenta el tipo de datos que estamos gastando y la operación que queremos realizar.

Por ejemplo, hay operaciones que son exclusivas de los números reales. No podremos utilizarlas con números enteros. Por ejemplo no podremos utilizar la raíz cuadrada, ni tampoco podremos utilizar la división ni cualquier operación típica de números reales.

Cuando realizamos una operación, todos los operandos que intervienen tienen que tener el mismo formato. Por ejemplo, si el resultado de una raíz es un número real, lo que introduzcamos para calcular la raíz cuadrada también tendrá que ser un número real.

Si no introducimos los datos en el formato correcto el autómata no realiza la operación. No se va a STOP.

Vamos a hacer una FC con varias operaciones con enteros.



Declaración	Nombre	Tipo	Valor ini	Comentario
in	VALOR_A	INT		
in	VALOR_B	INT		
in	VALOR_C	INT		
out	RESULTADO	INT		
in_out				
temp				

```
FC1 : Título:
Segm. 1: Título:
      L     #VALOR_A
      L     #VALOR_B
      +I
      L     #VALOR_C
      -I
      T     #RESULTADO
```

Este programa va a sumar lo que valga VALOR_A con lo que valga VALOR_B, al resultado le va a resta el VALOR_C y el resultado final me lo va a dejar en la variable que se llama RESULTADO.

Ahora tenemos que definirle cuanto vale cada valor.

Desde la OB 1 hacemos una llamada a la FC1

```
OB1 : Título:
Segm. 1: Título:
      CALL  FC     1
       VALOR_A  :=4
       VALOR_B  :=5
       VALOR_C  :=2
       RESULTADO:=MW20
```

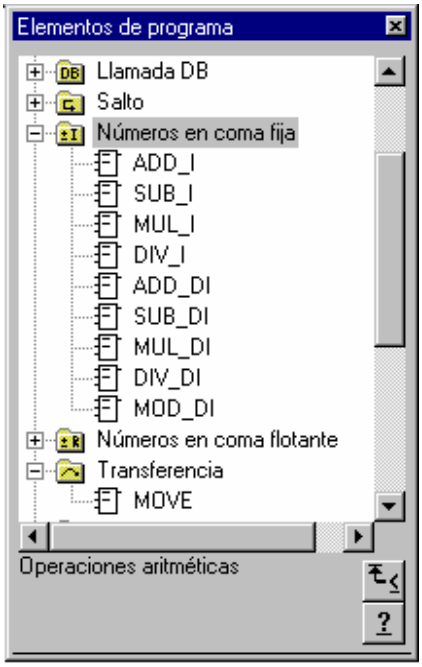
Si vamos a observar lo que tenemos en la MW 20, veremos que tenemos el valor 7. Es el resultado de la suma.

Podemos probar toda clase de operaciones de números enteros.

Lo que no podemos hacer son operaciones de números reales. Por ejemplo, la división y multiplicación son operaciones de números reales. Si nosotros queremos multiplicar dos por cuatro, tendremos que convertir primero estos números a números reales.

Todo este tipo de operaciones lo podemos hacer también en KOP o en FUP. Para poder seleccionar las operaciones en estos otros lenguajes, abrimos el catálogo y vamos a ver las operaciones de números enteros. Este catálogo también nos sirve para acordarnos de como se escriben las operaciones en AWL y saber las operaciones que tenemos disponibles.

Veamos las operaciones que podemos realizar. Las podemos realizar en cualquiera de los tres lenguajes.



Aquí vemos las operaciones que podemos realizar tanto en KOP como en FUP con números enteros.

EJERCICIO 13: CONVERSIONES DE FORMATOS

TEORÍA PREVIA: Posibles formatos en STEP 7

Para realizar otro tipo de operaciones y sobre todo para mezclar operaciones de números reales con números enteros, nos va a hacer falta muchas veces cambiar de formato las variables que tengamos definidas.

Por ejemplo si queremos dividir 8 entre 4, tendremos que convertir estos números enteros en números reales ya que la división es una operación de números reales.

Además los números enteros los tenemos en 16 bits y los números reales los tengo en 32 bits. Tendremos que hacer una transformación de longitud y luego una transformación de formato.

Veamos las posibilidades que tenemos de cambio de formato:

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).

Con estas operaciones, lo que hacemos son cambios de formato sin perder información. El número que queda después de la conversión es exactamente el mismo que antes de ella solo que en otro formato.

Por ejemplo, con este tipo de operaciones no podríamos convertir un número real en entero. Si tenemos el número 2.3, al pasarlo a entero tendríamos que dejar un 2. Aquí si que estamos perdiendo información. Las instrucciones ITR o RTB no existen. Estas operaciones en las que despreciamos parte del valor las hacemos con estas otras instrucciones:

RND:	Redondea al número más cercano.
RND+:	Redondea al número más cercano por arriba. Queda en 32 bits.
RND-:	Redondea al número más cercano por abajo. Queda en 32 bits.
TRUNC:	Corta la parte decimal. Queda un entero en 16 bits.

Para probar estas instrucciones haremos lo siguiente:

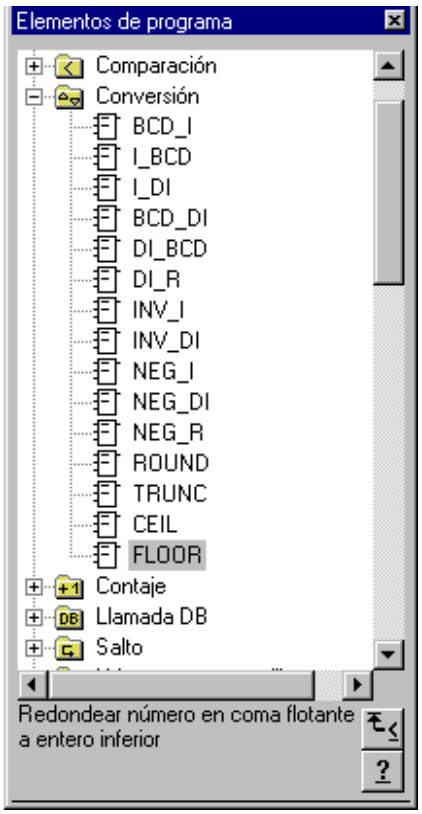
```
L     8
T     MW    10
L     MW    10
ITD
T     MD    20
DTR
T     MD    30
SQRT
T     MD    40
TRUNC
T     MW    50
```

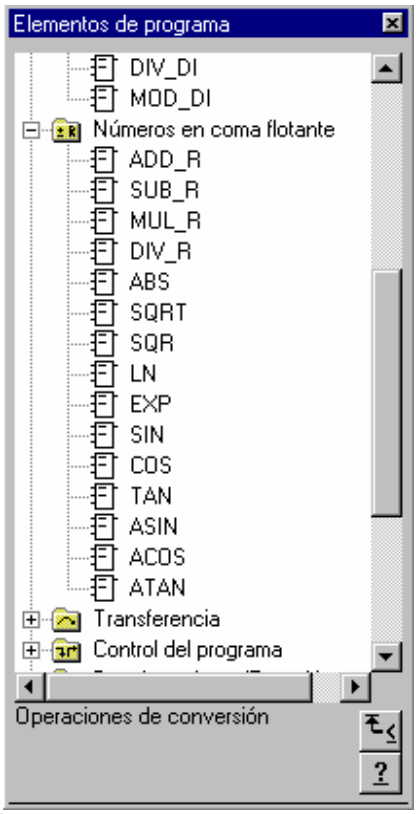
Aquí estamos gastando instrucciones de más, pero así podemos observar como se van siguiendo los pasos en las palabras de marcas.

Las operaciones las va realizando en el acumulador 1 y el resultado lo va dejando en el acumulador 1. Podríamos ejecutar todas las operaciones seguidas sin necesidad de transferir a palabras de marcas. Esto lo hacemos para poder observar los valores que va guardando.

En la tabla de “observar / forzar variables” podemos ver cómo hace los cambios y en qué se convierte el 8 inicial que hemos introducido.

Veamos donde tenemos todas estas operaciones en KOP y en FUP





EJERCICIO 14: OPERACIONES CON NÚMEROS REALES

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Instrucciones para realizar operaciones de números reales.

Vamos a ver como realizamos las operaciones de números reales.

Una forma de hacerlo es utilizar números reales para todo. Así no hay problemas de formatos. Vamos a ver cómo haríamos una división de reales.

Entraríamos en una FC. En la tabla de declaración definiríamos los siguientes valores:

0.0	VALOR_A	REAL
4.0	VALOR_B	REAL
8.0	RESULTADO	REAL

Si nos damos cuenta en las direcciones que nos ha asignado, veremos que cada real ocupa una doble palabra.

Vamos a programar la FC:

```
L     #VALOR_A
L     #VALOR_B
/R                    // Ponemos R porque son reales.
T     #RESULTADO
```

Ahora llamamos a esta FC desde el OB 1.

```
OB 1
CALL  FC     1
VALOR_A:= 8.0
VALOR_B:= 4.0
RESULTADO:= MD 0
```

Los valores que le pongamos tienen que ser reales. Si queremos dividir 8 entre 4 como en este caso, tenemos que escribir 8.0 y 4.0 para distinguir el 8 y 4 enteros del 8.0 y 4.0 reales.

El resultado lo tendrá que dejar necesariamente en una doble palabra.

Ahora si queremos ver el resultado de la operación, tendremos que ir a la tabla de “observar / forzar variable”, y observar la MD 0 en formato real.

La MD 0 estará en el acumulador. Lo que habrá allí será una serie de ceros y unos. Si nosotros observamos esto en binario veremos la serie de ceros y unos. Veremos algo parecido a esto: 001111001.....

No sabemos a qué número corresponde. Nosotros sabemos que el resultado es 2. Para nosotros es un entero pero en realidad el resultado de una división es siempre un número real. El resultado es 2.0. Si observamos el resultado en formato decimal, veremos que sale un número grandísimo que no corresponde al resultado. Lo que hace la tabla de observar es traducir “literalmente” la serie de ceros y unos a decimal. Esto no es lo que queremos ver.

Tendremos que decirle que queremos observarlo en formato real. Así a la hora de traducir ya sabe que una parte de los ceros y unos corresponde a la parte entera, y otra corresponde a la parte decimal.

Si observamos en formato real, veremos que el resultado es 2.0

Si nos situamos con el cursor en el lugar donde tenemos que rellenar un parámetro, al pulsar F1 nos sale directamente la ayuda del tipo de datos que se nos está pidiendo en ese parámetro. Nos dice la longitud del tipo de datos, como se forma en el acumulador y un ejemplo de cómo tenemos que escribirlo.

Veamos lo que haríamos si quisiésemos mezclar números reales con números enteros.

Por ejemplo, supongamos que queremos sumar 5+9 y hacer su raíz cuadrada.

5 y 9 son dos números enteros. En consecuencia el resultado de la suma será un número entero. Después queremos hacer una raíz que es una operación de números reales.

Vamos a programar una FC.

Dirección	Declaración	Nombre	Tipo	Valor ini	Comentario
0.0	in	VALOR_A	INT		
2.0	in	VALOR_B	INT		
4.0	out	RESULTADO	INT		
	in_out				
0.0	temp	SUMA	REAL		

```
FC2 : Título:
Segm. 1: Título:
      L     #VALOR_A
      L     #VALOR_B
      +I
      T     #SUMA
      L     #SUMA
      ITD
      DTR
      SQRT
      TRUNC
      T     #RESULTADO
```

Ahora hacemos la llamada desde el OB 1.

```
OB1 : Título:
Segm. 1: Título:
      CALL  FC     2
       VALOR_A  :=5
       VALOR_B  :=7
       RESULTADO:=MW10
```

El resultado Es un número entero. Lo observamos en una palabra de marcas.

Hemos tenido que hacer dos cambios de formato. Teníamos que convertir un entero en un real. En principio un entero está en 16 bits y un real son 32 bits. Primero hemos hecho un cambio de tamaño y luego un cambio de formato.

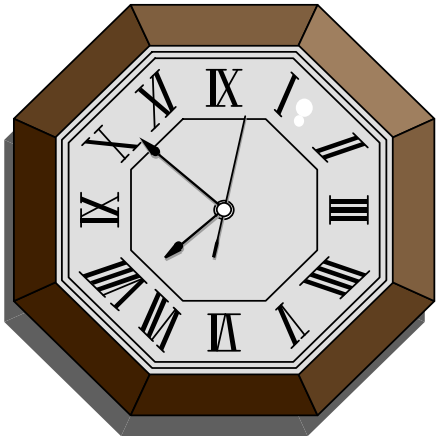
A la hora de sacar el resultado, lo hemos truncado para que nos vuelva a quedar un entero.

Ejercicio propuesto: Hacer varias de estas operaciones en KOP y en FUP con los instrucciones que se han visto anteriormente.

EJERCICIO 15: CONTROL DE UN GALLINERO

DEFINICIÓN Y SOLUCIÓN
TEORÍA PREVIA: Programación estructurada, bloques de datos, flancos.

El PLC tiene que decidir las horas de luz y las horas de oscuridad que va a tener el gallinero.



Se trata de optimizar las horas de luz y las horas de oscuridad para obtener la mayor cantidad de huevos posible.

El funcionamiento del gallinero debe ser el siguiente:

Llevaremos el control de los huevos que ponen las gallinas.

En nuestro ejemplo, esto lo haremos con dos contadores. Llevaremos la cuenta de los huevos que han puesto hoy en un contador, y los huevos que pusieron ayer en otro contador.

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

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

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

Para estructurarnos la programación haremos cada cosa en un bloque.

En principio haremos dos DB´s para guardarnos los datos que luego vamos a utilizar. Haremos un DB que se llame puestas, y allí guardaremos los huevos que han puesto hoy y los huevos que pusieron ayer.

A continuación haremos una FC 1. Allí haremos los dos contadores para simular los huevos que pusieron ayer y los huevos que han puesto hoy.

Desde esta FC, meteremos los valores en su correspondiente DB.

Haremos otro DB que se llame TIEMPOS. Allí meteremos los minutos iniciales de luz y de oscuridad que van a tener las gallinas, y además la cantidad de minutos de luz y de oscuridad que tendríamos que sumar o restar en su caso.

En otro FC haríamos la comparación de las puestas de huevos de ayer y de hoy, y efectuaríamos la operación correspondiente dependiendo de la comparación.

Finalmente haríamos una OB 1 para decidir cuando tenemos que acceder a cada uno de los bloques que hemos hecho.

Veamos como quedaría esto resuelto en AWL:
En el DB1 tendríamos el control de la cantidad de huevos:

Dirección	Nombre	Tipo	Valor inicial	Comentario
0.0		STRUCT		
+0.0	HUEVOS_HOY	INT	0	
+2.0	HUEVOS_AYER	INT	0	
=4.0		END_STRUCT		

En el DB 2 tendríamos el control del tiempo.

Dirección	Nombre	Tipo	Valor inicial	Comentario
0.0		STRUCT		
+0.0	MINUTOS_LUZ	INT	0	
+2.0	MINUTOS_OSC	INT	0	
+4.0	DATO_C	INT	0	
+6.0	DATO_T	INT	0	
=8.0		END_STRUCT		

FC1

```
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
      T     PUESTAS.PUESTAS_HOY
      L     Z     2
      T     PUESTAS.PUESTAS_AYER
      BE
```

FC2

```
      L     PUESTAS.PUESTAS_HOY
      L     PUESTAS.PUESTAS_AYER
      >I
      SPB   CONT
      <I
      SPB   TRIS
      ==I
      SPB   INDI
      BEA
CONT: L     TIEMPOS.MINUTOS_LUZ
       L    TIEMPOS.DATO_CONT
      -I
      T     TIEMPOS.MINUTOS_LUZ
      L     TIEMPOS.MINUTOS_OSC
```

```
        L     TIEMPOS.DATO_CONT
        -I
        T     TIEMPOS.MINUTOS_OSC
        BEA


TRIS:   L     TIEMPOS.MINUTOS_LUZ
        L     TIEMPOS.DATO_TRIS
        +I
        T     TIEMPOS.MINUTOS_LUZ
        L     TIEMPOS.MINUTOS_OSC
        L     TIEMPOS.DATO_TRIS
        +I
        T     TIEMPOS.MINUTOS_OSC
        BEA
INDI:   L     TIEMPOS.MINUTOS_LUZ
        L     1
        -I
        T     TIEMPOS.MINUTOS_LUZ
        L     TIEMPOS.MINUTOS_OSC
        L     1
        -I
        T     TIEMPOS.MINUTOS_OSC
        BE
```

Ahora tenemos que hacer la OB1 para decir cuando tiene que acceder a cada una de las FC´s.

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

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

En nuestro caso haremos la comparación cuando le demos a una entrada. Así no tendremos que esperar todo el día para comprobar si funciona.

OB1

```
UC    FC    1
U     E     0.0
FP    M     0.0
CC    FC    2
BE
```

Si no ponemos la instrucción de flanco, por deprisa que le demos a la entrada va a suponer más de un ciclo de scan, y la comparación se va a hacer más de una vez. Sumaría o restaría en su caso más de una vez y nos quedaríamos sin tiempo en un instante.

Ejercicio propuesto: Resolver el problema en KOP y en FUP con las instrucciones vistas anteriormente.

EJERCICIO 16: OPERACIONES DE SALTO

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Explicación de los saltos.

Tenemos muchos tipos de saltos. Vamos a probar unos cuantos tipos de saltos.

Hay algunos que coinciden con los que existían en S5. Hay otros que son nuevos.

Veamos los saltos que existen tanto en S5 como en S7.

SPA y SPB

En S5 son los saltos que utilizamos para saltar entre distintos bloques de programa. En S7 son los saltos que utilizamos para saltar a otros puntos de programa siempre dentro del mismo bloque.

SPA y SPB no sirven en STEP 7 para saltar de unos bloques a otros. Para ello tenemos las llamadas a otras FC´s que hemos visto antes. Tenemos las llamadas condicionales y las llamadas incondicionales.

El significado de los saltos es el mismo. El SPA es un salto absoluto, y el SPB es un salto condicional, es decir, depende del RLO.

Veamos dos ejemplos:

```
U     E     0.0          U     E     0.0
=     A     4.0          SPB   M001
SPA   M001                .........
.......
```

En el primer caso siempre que se lea la instrucción se va a producir un salto a la meta 1. En el segundo caso se saltará a la meta 1 si la E 0.0 está activa. En caso contrario no se producirá el salto.

Tenemos otros saltos como: SPB SPBN SPBB SPBNB SPO SPS

Si al salto SPB le añadimos una N, tenemos el salto SPBN. Estamos negando el significado del salto SPB. Significa que saltara cuando no se cumpla la condicion.

Si al salto SPB le añadimos la letra B, tenemos el salto SPBB. Con este salto, saltamos cuando se cumple la condición y además nos guardamos el valor del RLO en ese instante.

También tenemos saltos combinados como el SPBNB.

También tenemos los saltos que se refieren a operaciones matemáticas.

SPZ	Salta cuando el resultado es distinto de cero.
SPP	Salta cuando el resultado es positivo.
SPO	Salta cuando hay desbordamiento.
SPS	Salta cuando hay desbordamiento.

SPZP	Salta cuando el resultado es mayor o igual que cero.
SPU	Salta cuando el resultado no es coherente.

Como salto nuevo en S7 que no existía en S5 tenemos el SPL.

El salto SPL es un seleccionador de posiciones.

Antes de utilizar el salto tenemos que cargar un número. Dependiendo del número que pongamos aquí saltará a una meta o a otra. Si ponemos un número que se sale del rango, (cantidad de SPA que tenemos a continuación del salto) saltará a una meta determinada. A la que indicamos en el salto SPL.

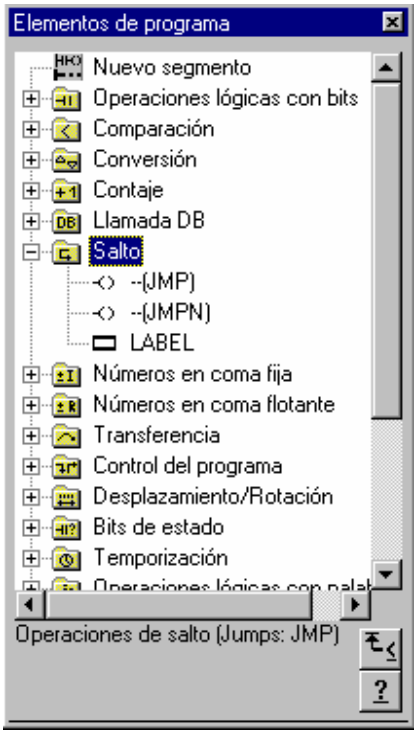
En el ejemplo que vamos a hacer, si nos salimos del rango no queremos que haga nada.

Haremos una meta en la que digamos que no haga nada.

Si nos vamos al catálogo de KOP o de FUP, veremos que no tenemos tantos tipos de saltos. Sólo tenemos el salto llamado JUMP. Con esta instrucción, saltaremos si se cumple la condición. En KOP y en FUP, siempre podemos poner delante de la instrucción que sea un contacto con la condición que queremos que se ejecute.

Nosotros saltaremos siempre con este salto y pondremos delante la condición que queramos.

Veamos en el catálogo dónde podemos encontrar los saltos en KOP / FUP.



También tenemos el salto JMPN. Esto significa que saltaremos cuando no se cumpla la condición precedente.

Para situar posteriormente las metas, tenemos LABEL.

En KOP y en FUP las metas las identificaremos por un número.

EJERCICIO 17. MEZCLA DE PINTURAS.

DEFINICIÓN Y SOLUCIÓN.

TEPRÍA PREVIA: Saltos.

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.

SOLUCIÓN EN AWL

FC 1

```
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
VERD:        L    50
             T    MW   0
             L    50
```

```
        T    MW   2
        L    0
        T    MW   4
        BEA
MARR:   L    40
        T    MW   0
        L    30
        T    MW   2
        L    30
        T    MW   4
        BE
```

Cuando utilizamos el SPL no ponemos BEA antes de las metas. Directamente detrás de los SPA que queramos poner, ponemos la meta a la que tiene que saltar en caso de que nos salgamos de rango. A continuación ponemos las metas que nosotros queremos definir para los saltos SPA.

Ahora desde la OB 1 llamamos a la FC 1 y le decimos el color que queremos formar.

```
OB   1
CALL FC   1
COLOR:= 1
BE
```

Probaremos a formar los tres colores y en la tabla de observar/forzar variables veremos como se van formando las mezclas dependiendo de los gramos que pongamos en cada una de las palabras de marcas.

Veremos que si ponemos un código de color que no existe, el programa no hace nada. Estará haciendo lo que pone en la meta ERRO, que es finalizar el programa sin hacer nada.

Ejercicio propuesto: Realizar el programa en KOP y en FUP con las instrucciones vistas anteriormente.

EJERCICIO 18: INSTRUCCINES QUE AFECTAN AL RLO (NOT, CLR, SET, SAVE)

DEFINICION Y SOLUCION

TEORÍA PREVIA: Introducción sobre las instrucciones.

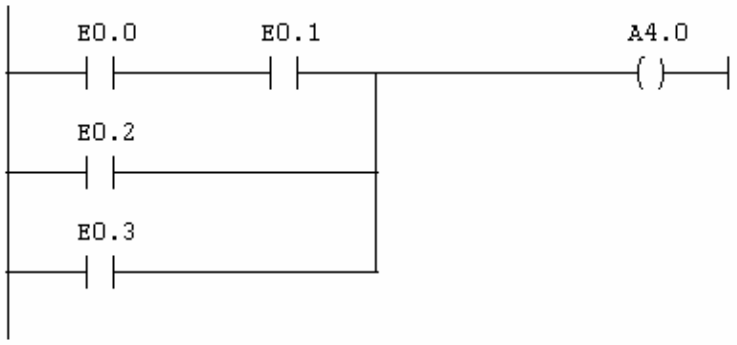
NOT: Invierte el RLO.

SET: Pone el RLO a 1 incondicionalmente. Cuando queremos que una instrucción condicional se ejecute siempre, añadimos la instrucción SET justo antes de la instrucción en cuestión.

CLR: Pone el RLO a 0 incondicionalmente.

SAVE: Hace una copia en el registro BIE del valor actual del RLO.

Supongamos que tenemos el siguiente circuito:



La salida se activará si se cumple cualquiera de las tres condiciones. (Tres ramas)

En un momento determinado del programa nos puede interesar saber si la salida se activó porque la condición que se cumplía era la primera.

Después de escribir las condiciones de la primera rama, escribiremos la instrucción SAVE.

Cuando queramos consultar si este bit está activo, escribiremos la instrucción: “U BIE”

Veamos como quedaría el ejemplo resuelto en AWL.

```
U     E     0.0
U     E     0.1
SAVE
O(
U     E     0.2
U     E     0.3
)
O     E     0.4
=     A     4.0
U     E     1.2
U     E     1.3
=     A     4.1
U     A     4.0
U     BIE
=     A     4.7
BE
```

Si se activó la salida A 4.0 porque se cumplió la primera condición, también se activará la salida 4.7. Si la salida 4.0 se activó por cualquiera de las otras dos condiciones no se activará la A 4.7.

Estas instrucciones también las tenemos en KOP y en FUP. Lo que ocurre es que su utilización es diferente. El comando SAVE en KOP está definido como una bobina. Esto quiere decir que tiene que ser un final de segmento. La rama por la que queremos controlar si pasó la señal, tenemos que dibujarla dos veces.

Tendremos que hacer tres segmentos.

En el primer segmento, dibujamos la primera rama y le asignamos el SAVE.

En el segundo segmento dibujamos el circuito que queremos resolver. Dibujamos las tres ramas y la asignación de la salida correspondiente.

En un tercer segmento, hacemos la consulta del bit guardado con el SAVE y le asignamos la salida 4.7.

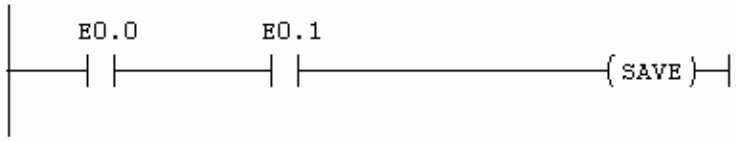
Si ahora traducimos esto a AWL veremos que tenemos más cantidad de instrucciones. Si queremos traducir lo que hemos hecho antes a KOP veremos que no es traducible.

Veamos como quedaría resuelto en KOP.

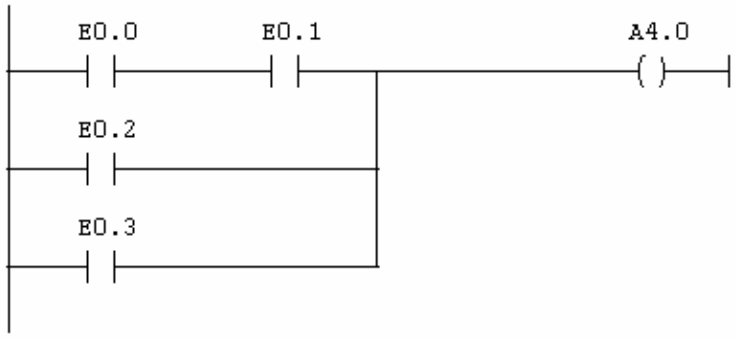
KOP

OB1 : Título:

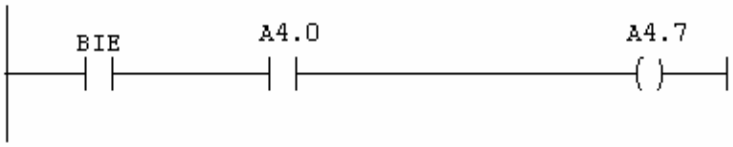
Segm. 1 : Título:



Segm. 2 : Título:



Segm. 3 : Título:



En FUP se haría exactamente igual.

EJERCICIO 19: FLANCOS

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Concepto de flanco.

En ocasiones queremos detectar únicamente un flanco de subida o un flanco de bajada de una señal. Queremos que solamente cuando se detecte uno de estos cambios sea cuando se ejecute algo. No queremos que se está ejecutando la acción durante todo el tiempo que se cumple la condición.

Por ejemplo, tenemos una FC que hace un desplazamiento de bits. Queremos que haga el desplazamiento cuando le demos a la entrada E 0.0.

Si escribimos:

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

Mientras tengamos activa la E 0.0 estará saltando a la FC 1.Por rápido que hagamos el cambio a la entrada seguro que se ejecuta más de un ciclo de scan. Se hará más de un desplazamiento de bits.

Si queremos que sólo lo haga una vez, deberíamos utilizar los flancos.

```
U    E    1.1
FP   M    0.0
CC   FC   1
BE
FC1
```

```
L     EB    0
T     MW    2
L     MW    2
SRW 1
T     AW    4
BE
```

Sólo haremos el desplazamiento en el momento que pasemos la entrada de cero a uno.

La instrucción también la podemos gastar para cosas que no sean precisamente una entrada. Por ejemplo, queremos que se haga el desplazamiento cuando un contador pase a ser mayor que otro.

La FC que teníamos seguirá siendo la misma. En la OB 1 programaríamos:

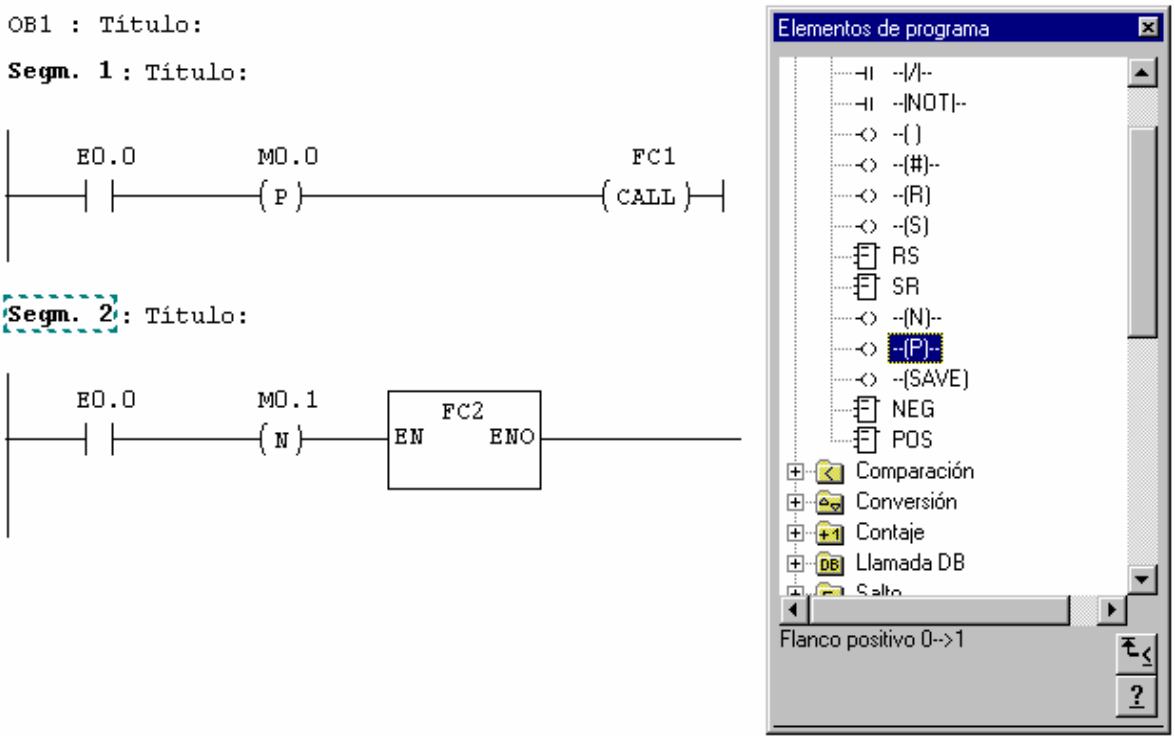
```
U     E     1.0
ZV    Z     1
U     E     1.1
ZV    Z     2
L     Z     1
L     Z     2
>I
FP    M     0.0
CC    FC    1
BE
```

En el momento en que Z1 pase a ser mayor que Z2 se hará el desplazamiento de los bits. Posteriormente aunque Z1 siga siendo mayor que Z2 ya no se hará más saltos.

Estas mismas instrucciones las tenemos en KOP y en FUP.

Para detectar un flanco positivo o negativo de la entrada E 0.0 lo haríamos del siguiente modo:

KOP



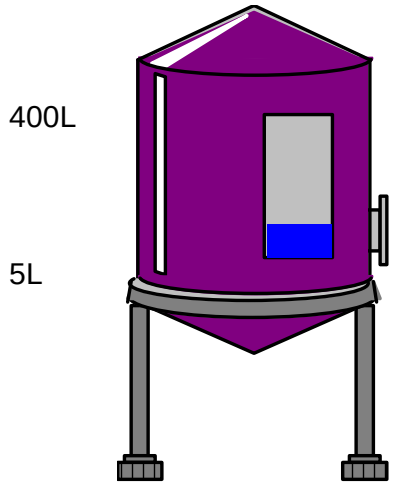
EJERCICIO 20: AJUSTE DE VALORES ANALÓGICOS

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Lectura y escritura de valores analógicos. Conversión formatos.

Vamos a hacer un ajuste de valores analógicos. Vamos a suponer que tenemos un tanque de líquido que como mínimo va a contener 5 litros de líquido, y como máximo va a contener 400 litros.

Dentro del tanque vamos a tener una sonda de nivel con la que queremos saber los litros de líquido que contiene.



Por otro lado sabemos que nuestra sonda puede medir entre 0 y 27648 teóricamente.

Lo primero que vamos a hacer es calcular el valor real al que podemos llegar con la entrada analógica que tenemos.

Vamos a transferir el valor de la entrada analógica a una palabra de marcas y vamos a observar el valor de la palabra de marcas.

```
L     PEW  288
T     MW   10
```

Una vez conozcamos el valor, vamos a hacer los cálculos.

Lo único que tenemos que hacer es una regla de tres. Para hacer los cálculos tenemos que tener en cuanta un par de cosas. Lo primero que tenemos que saber es que los valores reales sólo los podemos almacenar en dobles palabras.

Otra cosa importante que tenemos que saber es que no podemos hacer operaciones de números reales con números enteros. Tenemos que tener mucho cuidado con los formatos. La lectura de les entradas analógicas son números enteros. Luego tendremos que hacer divisiones que son operaciones de números reales. Tendremos que cambiar de formato.

Vamos a ver como quedaría el programa hecho:

SOLUCIÓN EN AWL

```
FC    3
IN          VALOR_SONDA       INT
IN          NIVEL_SUPERIOR    REAL
```

IN	NIVEL_INFERIOR	REAL
OUT	VALOR_GRADUADO	REAL
TEMP	V_SONDA_REAL	REAL
TEMP	RANGO	REAL

```
L     #VALOR_SONDA
ITD
DTR
T     #V_SONDA_REAL
L     #NIVEL_SUPERIOR
L     #NIVEL_INFERIOR
-R
T     #RANGO
L     #V_SONDA_REAL
L     26624.0
/R
L     #RANGO
*R
L     #NIVEL_INFERIOR
+R
T     #VALOR_GRADUADO
BE
```

OB1

```
CALL  FC    3
      VALOR_SONDA:= #PEW288
      NIVEL_SUPERIOR:= 400.0
      NIVEL_INFERIOR:= 5.0
```

VALOR_GRADUADO:= MD 0

En la doble palabra de marcas podemos ver el valor graduado del nivel que está midiendo. Es necesario que sea una doble palabra de marcas porque el valor que queremos observar es un valor real. Es el resultado de operaciones reales.

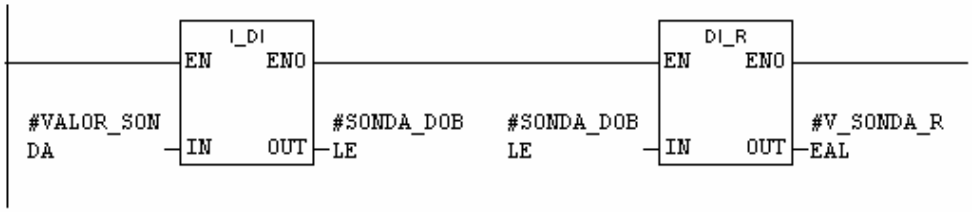
Además a la hora de observar el valor, lo vamos a tener que hacer en formato de número real. Si lo intentamos ver en formato de número entero veremos un valor de algo que no sabemos interpretar.

Veamos como resolveríamos el mismo ejercicio en KOP y en FUP

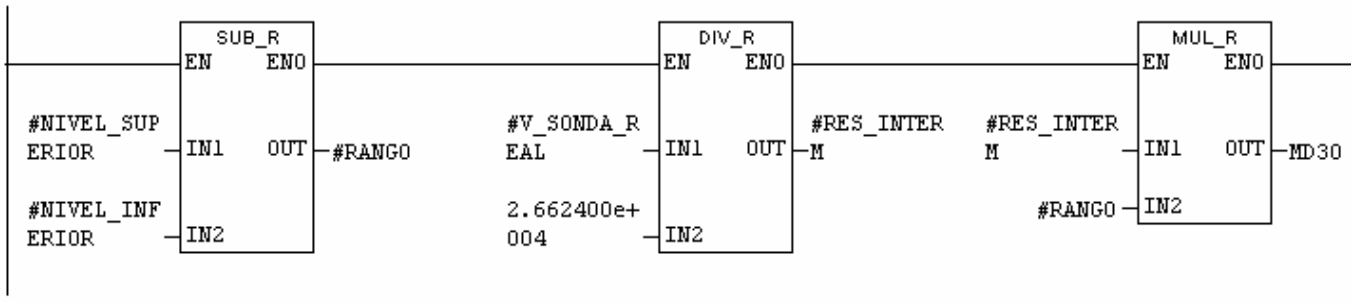
Dirección	Declaración	Nombre	Tipo	Valor ini	Comentario
0.0	in	VALOR_SONDA	INT		
2.0	in	NIVEL_SUPERIOR	REAL		
6.0	in	NIVEL_INFERIOR	REAL		
10.0	out	VALOR_GRADUADO	REAL		
	in_out				
0.0	temp	V_SONDA_REAL	REAL		
4.0	temp	RANGO	REAL		
8.0	temp	SONDA_DOBLE	DINT		
12.0	temp	RES_INTERM	REAL		

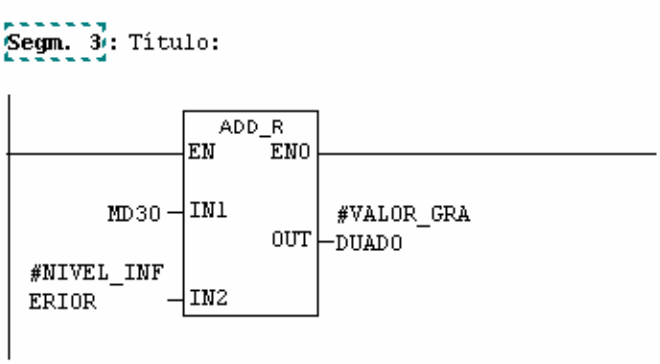
FC1 : Título:

Segm. 1 : Título:

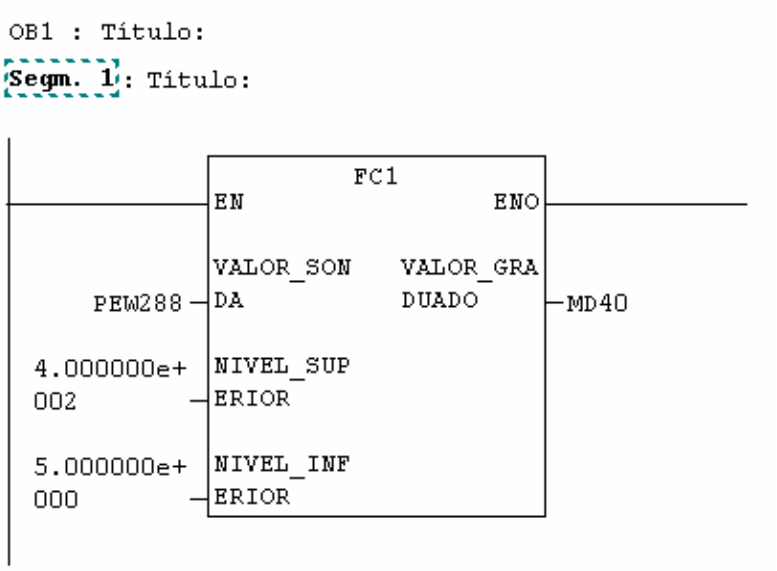


Segm. 2 : Título:





Ahora tendríamos que hacer la llamada desde el OB 1.



Ejercicio propuesto: Resolver el problema en FUP.

EJERCICIO 20: AJUSTE DE VALORES ANALÓGICOS

DEFINICIÓN Y SOLUCIÓN

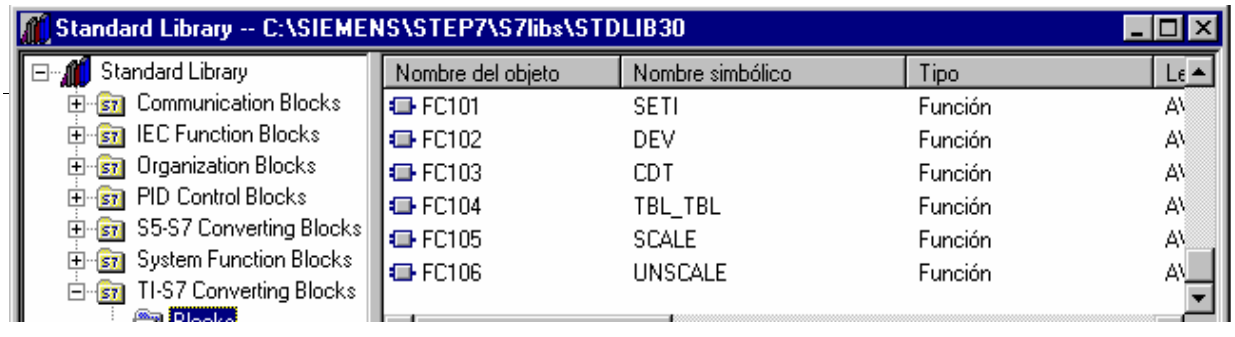
TEORÍA PREVIA: Manejo de analógicas.

Lo que hemos visto en el ejercicio anterior, también lo podemos hacer con unas FC´s ya programadas.

Estas FC las tenemos en una librería. Veamos como accedemos a ellas.

Primero vamos al menú Archivo > Abrir > librería:

Elegimos las Estándar Library de la versión 3.



Vemos que la FC 105 se llama SCALE. Nos sirve para escalar valores analógicos. Esto es lo mismo que hemos hecho en el ejercicio anterior.

Lo único que tenemos que hacer es traer esta función a nuestro proyecto y hacer una llamada. Rellenamos los parámetros que nos pide y ya tenemos la función hecha.

Al utilizar estas funciones ya hechas, tenemos una ventaja. Nos ofrecen información en caso de producirse algún tipo de error en el escalado.

```
OB1 : Título:
Segm. 1: Título:
      CALL  "SCALE"
       IN     :=PEW288
       HI_LIM :=4.000000e+002
       LO_LIM :=5.000000e+000
       BIPOLAR:=FALSE
       RET_VAL:=MW20
       OUT    :=MD30
```

EJERCICIO 21: EJEMPLO CON UDT

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Tipo de datos de usuario.

Vamos a suponer que tenemos que automatizar muchas máquinas de las cuales queremos tener información de tres datos de cada una de ellas.

Queremos tener controlado su temperatura, el interruptor de marcha/paro, y la cantidad de piezas que llevan hechas al día.

Para organizarnos los datos lo vamos a hacer utilizando los UDT.

Vamos a suponer que tenemos las máquinas distribuidas en dos polígonos. Dentro de cada polígono tenemos tres plantas y dentro de cada planta tenemos 4 máquinas.

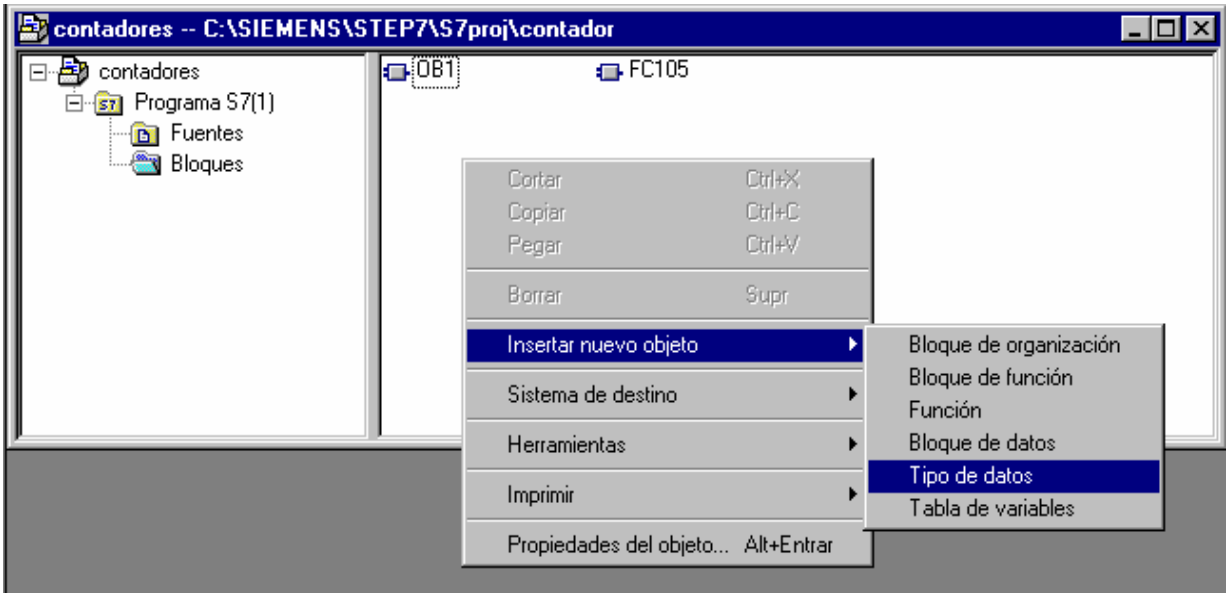
De cada una de esas máquinas tenemos que controlar los datos que hemos comentado antes.

Para ello nos vamos a crear unos UDT del tipo que nos interese.

Para nosotros el UDT 1 va a ser el tipo de datos que asignaremos a cada una de las máquinas.

Para generar un UDT 1 tenemos que situarnos en el administrador de Simatic e insertar un nuevo objeto tal y como lo habíamos hecho antes para el resto de bloques.

En este caso insertamos un tipo de datos.



Veamos como quedaría nuestro UDT 1.

Dirección	Nombre	Tipo	Valor inicial	Comentario
0.0		STRUCT		
+0.0	TEMP	REAL	0.000000e+000	
+4.0	MARCHA	BOOL	FALSE	
+6.0	PIEZAS	INT	0	
=8.0		END_STRUCT		

Ya tenemos definido un tipo de datos. Ahora cada máquina será de tipo UDT 1.

Dirección	Nombre	Tipo	Valor inicial	Comentario
0.0		STRUCT		
+0.0	MAQ1	UDT1		
+8.0	MAQ2	UDT1		
+16.0	MAQ3	UDT1		
+24.0	MAQ4	UDT1		
=32.0		END_STRUCT		

Dentro de cada máquina estamos incluyendo implícitamente cada uno de los tres datos que hemos dicho antes.

Ahora cada planta será de tipo UDT 2.

Dirección	Nombre	Tipo	Valor inicial	Comentario
0.0		STRUCT		
+0.0	PLANTA1	UDT2		
+32.0	PLANTA2	UDT2		
+64.0	PLANTA3	UDT2		
=96.0		END_STRUCT		

Dentro de cada planta, estamos incluyendo implícitamente cuatro máquinas y sabemos que dentro de cada máquina van sus tres datos.

Sin escribir demasiadas líneas tenemos un montón de datos acumulados.

Ahora tenemos que hacer el DB.

Primero vamos a la tabla de símbolos y le ponemos nombre al DB que vamos a gastar.

Le llamamos DATOS.

Dentro de datos ponemos:

Dirección	Nombre	Tipo	Valor inicial	Comentario
0.0		STRUCT		
+0.0	POLI1	UDT3		
+96.0	POLI2	UDT3		
=192.0		END_STRUCT		

Dentro de cada una de estas líneas, están implícitas tres plantas dentro de cada una de las cuales hay cuatro máquinas, dentro de cada una de las cuales hay tres datos.

Ahora la ventaja que tenemos a parte de habernos ahorrado escribir muchos datos, es la manera de acceder a los datos.

Si de memoria tenemos que saber en que dirección se encuentra cada uno de los datos, tendríamos que estar mirando cada vez el DB para saber a que dirección tenemos que acceder.

Ahora si dentro del DB seleccionamos el menú Ver > datos veremos todos los datos que hemos creado con el nombre que tienen.

28.0	POLI1.PLANTA1.MAQ4.MARCHA	BOOL	FALSE	FALSE
30.0	POLI1.PLANTA1.MAQ4.PIEZAS	INT	0	0
32.0	POLI1.PLANTA2.MAQ1.TEMP	REAL	0.000000e+000	0.000000e+000
36.0	POLI1.PLANTA2.MAQ1.MARCHA	BOOL	FALSE	FALSE
38.0	POLI1.PLANTA2.MAQ1.PIEZAS	INT	0	0
40.0	POLI1.PLANTA2.MAQ2.TEMP	REAL	0.000000e+000	0.000000e+000
44.0	POLI1.PLANTA2.MAQ2.MARCHA	BOOL	FALSE	FALSE
46.0	POLI1.PLANTA2.MAQ2.PIEZAS	INT	0	0
48.0	POLI1.PLANTA2.MAQ3.TEMP	REAL	0.000000e+000	0.000000e+000
52.0	POLI1.PLANTA2.MAQ3.MARCHA	BOOL	FALSE	FALSE
54.0	POLI1.PLANTA2.MAQ3.PIEZAS	INT	0	0
56.0	POLI1.PLANTA2.MAQ4.TEMP	REAL	0.000000e+000	0.000000e+000

Esto es parte del DB que hemos creado.

De este modo, podemos acceder por su nombre al dato que queramos. Por ejemplo, podemos acceder a la temperatura de la máquina dos de la planta tres del polígono 1 de la siguiente manera:

DATOS.POLI1.PLANTA3.MAQ2.TEMP

De este modo no tengo por qué saber qué dirección tiene esta línea dentro del DB. Accedo a cada cosa por su nombre.

Vamos a ver como ejemplo cómo accederíamos a dos de los datos que tenemos en el DB.

Por ejemplo vamos suponer que queremos poner en marcha una de las máquinas con la entrada E0.0. (Máquina tres de la planta dos del polígono 1)

Esto corresponde a un bit. En consecuencia lo trataremos igual que cualquier bit.

```
U     E     0.0
= DATOS.POLI1.PLANTA2.MAQ3.MARCHA
```

Si hacemos esto y luego vamos a observar el valor actual del módulo de datos, veremos que hemos introducido un uno en este bit.

Vamos a ver como leeríamos una temperatura. Suponemos que la entrada analógica va a ser la temperatura de la máquina.

```
L     PEW   288
T     DATOS.POLI1.PLANTA1.MAQ1.TEMP
```

Si quisiéramos sacar este dato por la salida analógica, escribiríamos:

```
L     DATOS.POLI1.PLANTA1.MAQ1.TEMP
T     PAW   288
```

EJERCICIO 22: OPERACIONES LÓGICAS CON PALABRAS

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Concepto de operación lógica.

Podemos hacer operaciones lógicas con palabras. Las operaciones lógicas se realizan bit a bit.

Las operaciones que podemos hacer son AND. OR y XOR.

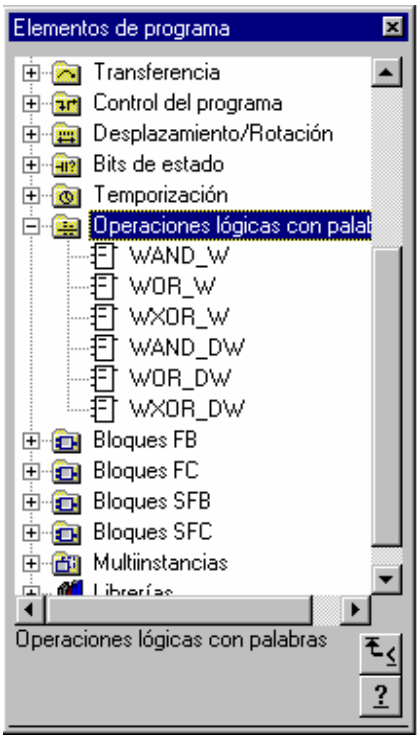
Esto lo utilizaremos para hacer enmascaramientos de bits.

Las operaciones son las siguientes: UW, OW, XOW

Las utilizaremos de la siguiente manera:

```
L     MW   0
L     MW   2
UW
T     MW   4
```

También tenemos las instrucciones en el catálogo de KOP y de FUP.



A continuación veremos un ejemplo en el que utilizaremos estas operaciones.

EJERCICIO 23: ALARMAS

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Concepto de operación lógica.

Vamos a exponer un ejercicio en el que utilizaremos la operación XOR. Si hacemos un XOR entre dos palabras, el resultado que obtendremos será:

Si los dos bits coinciden el resultado será cero. Si los dos bits son distintos e, resultado será 1.

Veamos un ejemplo:

0001_1001_0101_0110
0101_0000_1110_1000

0100_1001_1011_1110

Si observamos el resultado, allá donde veamos un 1 significa que en ese bit difieren las dos palabras anteriores de las cuales hemos hecho una XOR.

Supongamos que nosotros tenemos una serie de alarmas en la instalación. Algunas de ellas serán contactos normalmente cerrados y otros serán contactos normalmente abiertos. Unos deberán permanecer a uno para estar en estado correcto, y otros deberán estar a cero para estar correctamente.

Supongamos que la secuencia buena de las alarmas es la siguiente:

11100010

Lo que queremos es que en caso de que alguna alarma salte, parpadee la salida A4.0. y además nos indique la alarma que ha sido disparada en al byte de salidas 5.

```
OB1 : Título:
Segm. 1: Título:
      L     2#11100010
      T     MW    10
      L     MW    10
      L     EW     0
      XOW
      T     MW    20
      L     MW    20
      L     0
      <>I
      U     M      0.0
      =     A      4.0
      L     MB    21
      T     AB     5
      UN    M      0.1
      L     S5T#500MS
      SE    T      1
      U     T      1
      =     M      0.1
      UN    M      0.0
     BEB
     UN    M      0.0
     =     M      0.0
```

Ejercicio propuesto: Resolver esto mismo en KOP y en FUP utilizando las instrucciones vistas anteriormente.

Ejercicio propuesto: Hacer ejemplos utilizando las otras operaciones lógicas con palabras, en cada uno de los tres lenguajes.