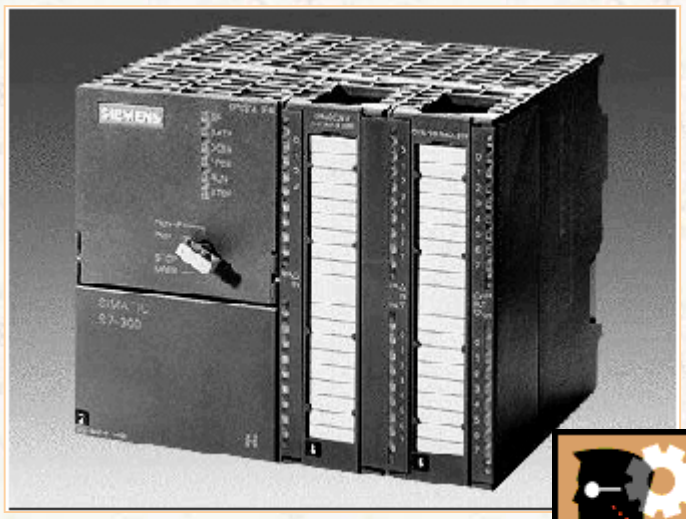
MANUAL DE PROGRAMACIÓN EN SIMATIC S7



Manual de programación en Simatic S7

INDICE

1.1. Distribución

Este documento es de libre distribución y empleo. El autor no se responsabiliza de cualquier modificación efectuada al original.

Se permite la copia parcial o integra de su contenido.

1.2. Objetivos

Con este tutorial se pretende enseñar los conocimientos necesarios para programar en los autómatas de la serie Simatic S7 de Siemens AUT.

No se va a mostrar la utilización del programa Step7, ya que este programa es comercial e incluye los manuales necesarios para su empleo.

Todo lo aquí expuesto ha sido obtenido de los manuales del autómata, pero se ha estructurado de distinta forma, tratando que la persona que lo utilice no se pierda en ningún momento y que acceda de forma rápida a lo que busca.

Por supuesto, este tutorial no es una alternativa al manual que incorpora el autómata, se debe tomar como una guía complementaria. El manual proporciona más ejemplos y mayor detalle que el expuesto aquí.

Es necesario que el lector esté familiarizado con el álgebra de Boole, y si se está familiarizado con algún otro lenguaje de programación (como basic o ensamblador) será mucho más sencillo y rápido.

Para contactar con el autor para cualquier comentario o sugerencia: ambart@terra.es

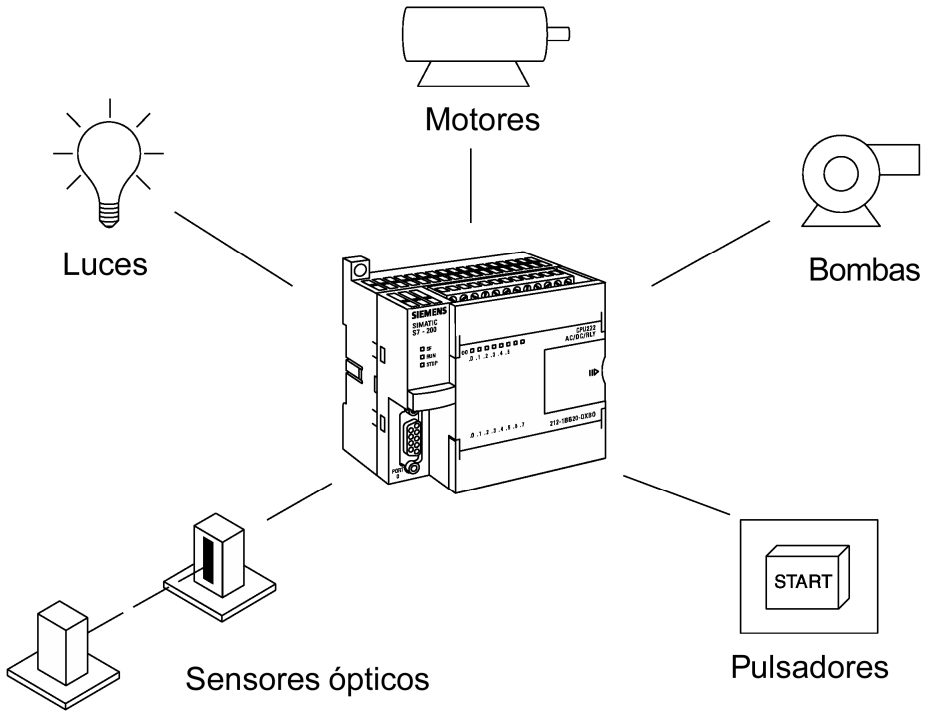
PARTE 1

Programación

2. Introducción

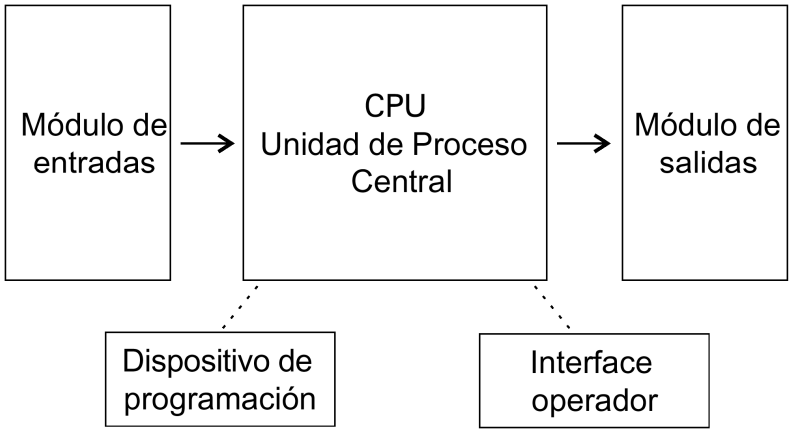
2.1. PLC's

Los Controladores Lógicos Programables (PLCs), también llamados autómatas programables, forman parte de la familia de los ordenadores. Se usan en aplicaciones comerciales e industriales. Un autómata monitoriza las entradas, toma decisiones basadas en su programa, y controla las salidas para automatizar un proceso o máquina. Este curso trata de suministrar la información básica sobre las funciones y las configuraciones de los autómatas programables.

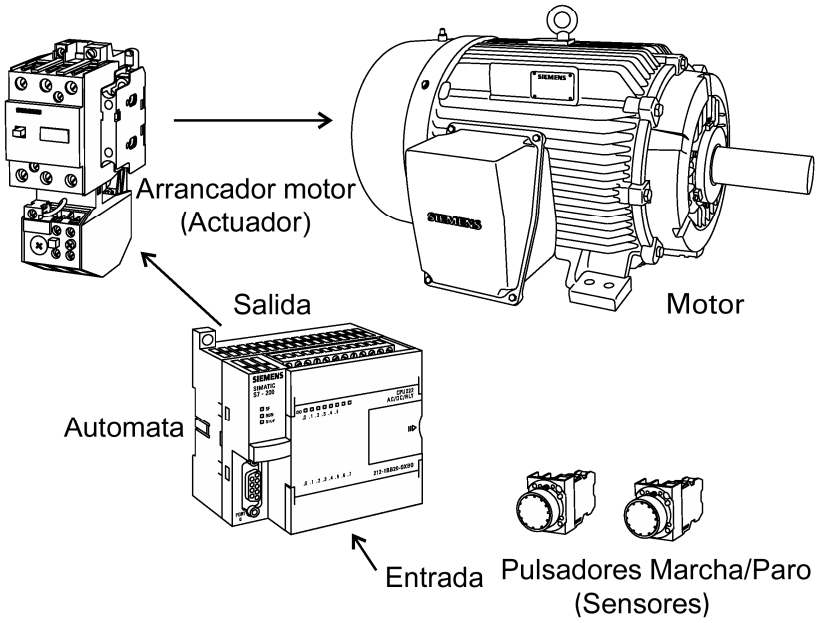


2.2. Funcionamiento básico

Un autómata programable consiste en módulos de entradas, una CPU, y módulos de salidas. Una entrada acepta una gran variedad de señales analógicas o digitales de varios dispositivos de campo (sensores) y los convierte en una señal lógica que puede usar la CPU. La CPU tomas las decisiones y ejecuta las instrucciones de control basadas en las instrucciones del programa de la memoria. Los módulos de salida convierten las instrucciones de control de la CPU en una señal digital o analógica que se puede usar para controlar dispositivos de campo (actuadores). Se usa un dispositivo de programación para introducir las instrucciones deseadas. Estas instrucciones especifican lo que debe hacer el autómata según una entrada especifica. Un dispositivo operador permite procesar la información para ser visualizada e introducir nuevos parámetros de control.

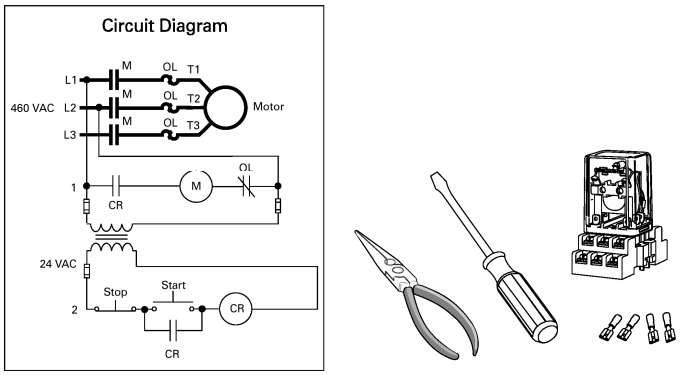


Los pulsadores (sensores), del siguiente ejemplo, conectados a las entradas del autómata, pueden usarse para arrancar y parar un motor conectado a un autómata a través de un arrancador (actuador).



2.3. Control

Anteriormente a los autómatas, muchas de estas tareas de control se solucionaban mediante relés o contactores. Esto con frecuencia se denominaba control mediante lógica cableada. Se tenían que diseñar los diagramas de circuito, especificar e instalar los componentes eléctricos, y crear listas de cableado. Entonces los electricistas debían cablear los componentes necesarios para realizar una tarea especifica. Si se cometía un error, los cables tenían que volver a conectarse correctamente. Un cambio en su función o una ampliación del sistema requería grandes cambios en los componentes y su recableado.



2.4. Autómata

Lo mismo, además de tareas más complejas, se puede hacer con un autómata. El cableado entre dispositivos y los contactos entre relés se hacen en el programa del autómata. Aunque todavía se requiere el cableado para conectar los dispositivos de campo, éste es menos intensivo. La modificación de la aplicación y la corrección de errores son más fáciles de realizar. Es más fácil crear y cambiar un programa en un autómata que cablear y recablear un circuito.

2.5. Ventajas

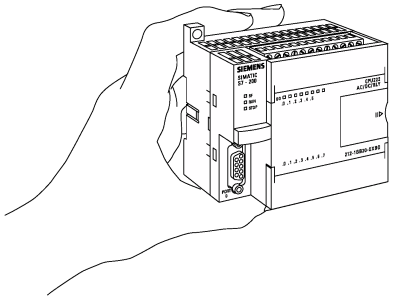
- Menor tamaño físico que las soluciones de cableado
- La realización de cambios es más fácil y más rápida.
- Los autómatas llevan integradas funciones de diagnóstico.
- Los diagnósticos están disponibles centralmente en la PG.
- Las aplicaciones pueden ser inmediatamente documentadas.
- Se pueden duplicar las aplicaciones más rápidamente y con menor coste.

2.6. Autómatas Siemens

Siemens fabrica varios líneas de autómatas de la familia SIMATIC® S7. Son: S7-200, S7-300, y S7-400.

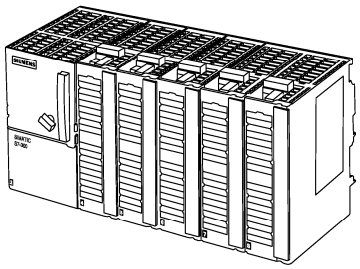
S7-200

Al S7-200 se le denomina microsistema a causa de su pequeño tamaño. El S7-200 tiene un diseño compacto que significa que la fuente de alimentación y las Entradas/Salidas las lleva incorporadas. El S7-200 puede usarse en pequeñas aplicaciones independientes como ascensores, lavado de coches, o máquinas mezcladoras. También puede utilizarse en aplicaciones industriales más complejas como máquinas de embotellado y empaquetado.

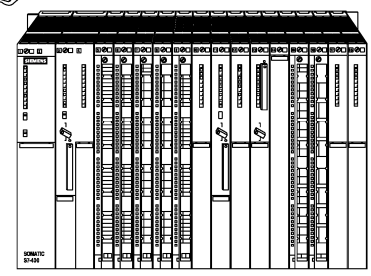


S7-300 y S7-400

Estos autómatas se usan en aplicaciones más complejas que necesitan de un mayor número de Entradas/Salidas. Ambos son modulares y ampliables. La fuente de alimentación y las Entradas/Salidas consisten en módulos independientes conectados a la CPU. La elección entre el S7-300 y el S7-400 depende de la complejidad de la tarea y de una posible ampliación futura.



S7-300



S7-400

2.7. Ciclo de trabajo en el autómata

El autómata va a ejecutar nuestro programa de usuario en un tiempo determinado, el cual va a depender sobre todo de la longitud del programa. Esto es debido a que cada instrucción tarda un tiempo determinado en ejecutarse, por lo que en procesos rápidos será un factor crítico.

En un sistema de control mediante autómata programable tendremos los siguientes tiempos:

1. Retardo de entrada.
2. Vigilancia y exploración de las entradas.
3. Ejecución del programa de usuario.
4. Transmisión de las salidas.
5. Retardo en salidas.

Los puntos 2, 3 y 4 sumados dan como total el tiempo de ciclo del autómata. Tras este ciclo es cuando se modifican las salidas, por lo que si varían durante la ejecución del programa tomarán como valor el último que se haya asignado. También supone que una variación en las entradas no se verá durante la ejecución del programa, hasta que se inicie un nuevo ciclo.

Esto es así debido a que no se manejan directamente las entradas y las salidas, sino una imagen en memoria de las mismas que se adquiere al comienzo del ciclo (2) y se modifica al final de éste (retardo).

En la etapa de vigilancia (watchdog) se comprueba si se sobrepasó el tiempo máximo de ciclo, activándose en caso afirmativo la señal de error correspondiente.

2.8. Lenguajes de programación

Para toda la familia de autómatas Simatic S7 se emplean los siguientes lenguajes de programación:

- Lista de instrucciones (AWL).

- Esquema de contactos (KOP): se representa gráficamente con símbolos eléctricos.

Internamente el autómata solo trabaja con lista de instrucciones, KOP es traducido a AWL por Step7. En este tutorial solo veremos la programación en lista de instrucciones.

Las instrucciones son las órdenes lógicas elementales que el sistema debe obedecer. Suelen ocupar una línea de programa (dos en algunas instrucciones), y no pueden escindirse en instrucciones parciales.

Las instrucciones AWL se dividen en:

- OPERACION: indica la instrucción que se ha de realizar (ej. AND).
- OPERANDO: indica una constante o dirección con la que debe trabajar la operación. Si se trata de una dirección se puede manejar en modo bit, byte o palabra (tal y como veremos más adelante).

```
Operación
   |
   | Identificador del operando
   |  |
   U  E  32.0          //Operación AND lógica
         |              |
       Operando     Comentarios
```

Una instrucción puede no contener operando (ej. NOT).

El operando puede ser sustituido por un nombre simbólico (ej. MOTOR_ON), el cual debe ser especificado al comienzo del programa para indicar a que entrada o salida equivale.

3. Sistemas de numeración

Dado que un autómata es un ordenador, almacena información en forma de condiciones On y Off (1 ó 0), refiriéndose a dígitos binarios (bits).

A veces los dígitos binarios se usan individualmente y otras son utilizados para representar valores numéricos.

3.1. Sistema Decimal

En los autómatas se usan varios sistemas numéricos. Todos los sistemas de números tienen las mismas características: dígitos, base, potencia. El sistema decimal, que es de utilización común en la vida diaria, tiene las características siguientes:

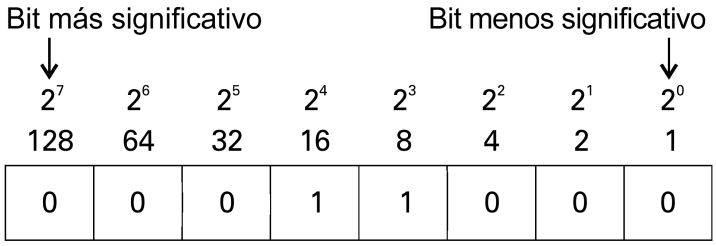
Diez dígitos 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Base 10
Potencias 1, 10, 100, 1000, ...

3.2. Sistema Binario

El sistema binario se usa en los controladores programables. Tiene las siguientes características:

Dos dígitos 0, 1
Base 2
Potencias de base 2 (1, 2, 4, 8, 16, ...)

En el sistema binario los 1s y 0s se ordenan en columnas. Cada columna tiene un peso. La primera columna tiene un peso binario de 2^0. Esto equivale al decimal 1. A éste se le denomina bit menos significativo. El peso binario se dobla en cada columna sucesiva. La siguiente columna, por ejemplo, tiene un peso de 2^1, que equivale al decimal 2. El valor decimal se dobla en cada columna sucesiva. El número más a la izquierda se denomina bit más significativo. En el ejemplo, el bit más significativo tiene un peso binario de 2^7. Es equivalente al decimal 128.

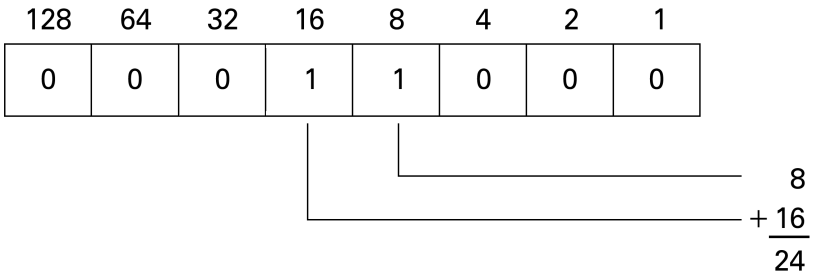


3.3. Convertir el binario

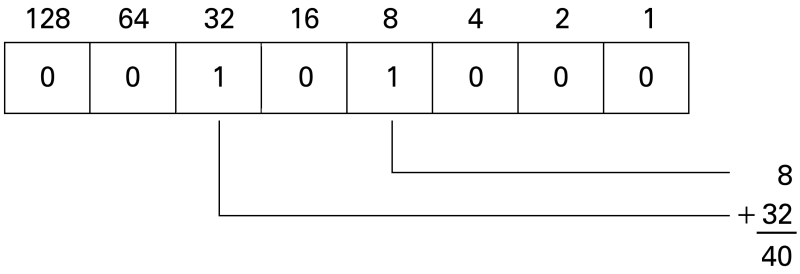
Los siguientes pasos se pueden usar para interpretar un número decimal desde un valor binario:

1. Buscar de derecha a izquierda (de menos significativo a más significativo) los 1s.
2. Escribir la representación decimal de cada columna que contenga un 1.
3. Sumar los valores de esas columnas.

En el ejemplo siguiente, las columnas cuarta y quinta desde la derecha contienen un 1. El valor decimal de la cuarta columna desde la derecha es 8, y el valor decimal de la quinta columna desde la derecha es 16. El decimal equivalente de este número binario es 24. La suma de todas las columnas con peso que contienen un 1 es el número decimal que el autómata ha almacenado.

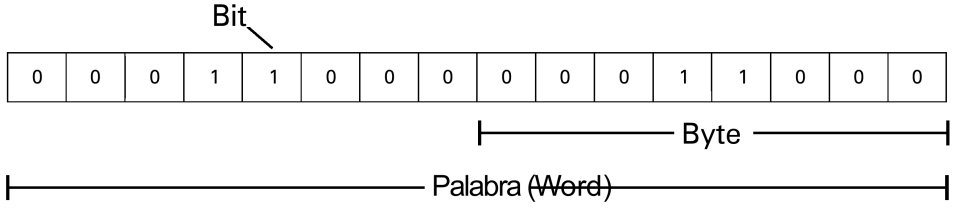


En el siguiente ejemplo las columnas cuarta y quinta desde la derecha contienen un 1. El valor decimal de la cuarta columna desde la derecha es 8, y el valor decimal de la sexta columna desde la derecha es 32. El decimal equivalente de este número binario es 40.



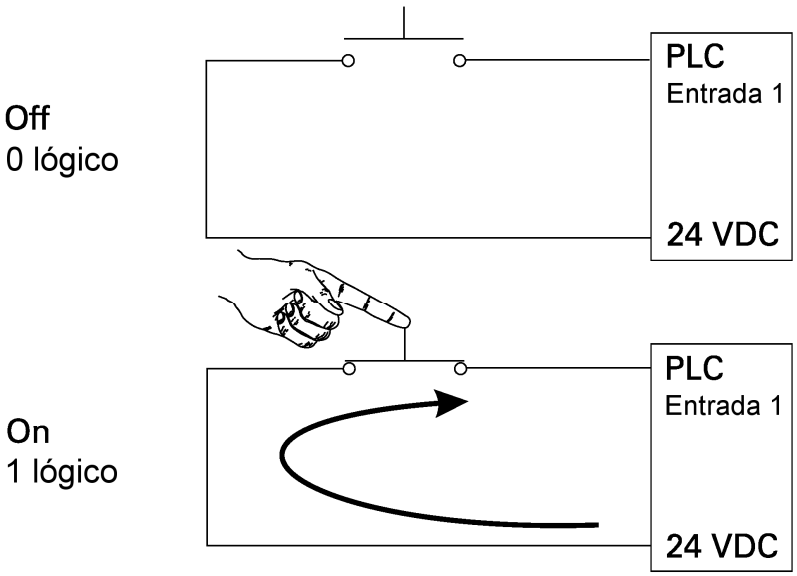
3.4. Bits, bytes, y palabras (words)

Cada unidad binaria de datos es un bit. Cada 8 bits hacen 1 byte. 2 bytes ó 16 bits, hacen 1 palabra.



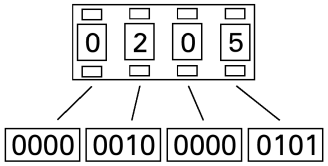
3.5. 0 lógico, 1 lógico

Los controladores programables sólo pueden entender una señal que sea On o Off (presente o no presente). El sistema binario es un sistema en el cual sólo hay dos números, 1 y 0. El binario 1 indica que una señal está presente, o el interruptor está On. El binario 0 indica que la señal no está presente, o el interruptor está Off.



3.6. BCD

El código BCD (Binary-Coded Decimal) son números decimales en los que cada dígito está representado por un número binario de 4 bits. Un contador de vueltas es un ejemplo de un dispositivo de entrada de 4 bits. El BCD se usa comúnmente con dispositivos de entrada y salida. Los números binarios se rompen en grupos de 4 bits, cada grupo representa un decimal equivalente. Un contador de vueltas de 4 dígitos, como el que se muestra abajo, podría controlar 16 (4 x 4) entradas al autómata.



Números Decimales	Números BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

3.7. Hexadecimal

El hexadecimal es otro sistema usado en los autómatas programables. El sistema hexadecimal tiene las características siguientes:

16 dígitos 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
Base 16
Potencias de base 16 (1, 16, 256, 4096 ...)

Se utilizan los diez dígitos del sistema decimal para los primeros diez dígitos del sistema hexadecimal. Se usan las primeras seis letras del alfabeto para los seis dígitos restantes.

A = 10 D = 13
B = 11 E = 14
C = 12 F = 15

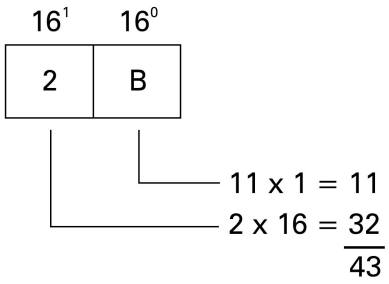
El sistema hexadecimal se utiliza en los autómatas programables porque permite representar el estado de un gran número de bits binarios en un pequeño espacio como es la pantalla de un ordenador o un visualizador de programación. Cada dígito hexadecimal representa el estado exacto de 4 bits binarios. Para convertir un número decimal a un número hexadecimal el número decimal se dividirá por base 16. Por ejemplo, para convertir el decimal 28 a hexadecimal:

$$16 \overline{)\,28} \quad 1 \text{ r } 12$$

El decimal 28 dividido entre 16 es 1 con un resto de 12. Doce es equivalente a C en hexadecimal. El hexadecimal equivalente del decimal 28 será pues 1C.

El valor decimal de un número hexadecimal se obtiene multiplicando individualmente los dígitos hexadecimales por el peso de base 16 y después sumando los resultados. En el siguiente ejemplo el número hexadecimal 2B se convierte a su decimal equivalente que es 43.

$16^0 = 1$
$16^1 = 16$
B =11



3.8. Conversión de números

La siguiente tabla muestra unos pocos valores numéricos en representación decimal, binario, BCD, y hexadecimal.

Decimal	Binario	BCD	Hexadecimal
0	0	0000	0
1	1	0001	1
2	1	0010	2
3	11	0011	3
4	100	0100	4
5	101	0101	5
6	110	0110	6
7	111	0111	7
8	1000	1000	8
9	1001	1001	9
10	1010	0001 0000	A
11	1011	0001 0001	B
12	1100	0001 0010	C
13	1101	0001 0011	D
14	1110	0001 0100	E
15	1111	0001 0101	F
16	1 0000	0001 0110	10
17	1 0001	0001 0111	11
18	1 0010	0001 1000	12
19	1 0011	0001 1001	13
20	1 0100	0010 0000	14
.	.	.	.
.	.	.	.
126	111 1110	0001 0010 0110	7E
127	111 1111	0001 0010 0111	7F
128	1000 1000	0001 0010 1000	80
.	.	.	.
.	.	.	.
510	1 1111 1110	0101 0001 000	1FE
511	1 1111 1111	0101 0001 0001	2FF
512	10 0000 0000	0101 0001 0010	200

4. Estructura de Simatic S7

4.1. Estructura de la memoria en Simatic S7

La memoria del autómata está estructurada en las siguientes zonas:

MEMORIA DE PROGRAMA:

Aquí es donde se va a introducir el programa que hagamos. La capacidad varía según la CPU que utilicemos, para la S7-314 IFM tenemos 24K bytes, lo cual equivale a una media de 8K (8192) líneas de programa. Como se puede observar cada línea de programa suele ocupar 4 bytes de memoria.

IMAGENES DE ENTRADAS Y SALIDAS:

Tal y como vimos en 2.1, el autómata maneja una imagen en memoria de las entradas y las salidas, actualizando éstas al final del ciclo y recogiendo su estado al principio de otro.

MARCAS DE MEMORIA:

Aquí almacenaremos los datos intermedios que deseemos preservar. Solo se admiten datos de 1 bit, aunque pueden manejarse en modo bit, byte, etc.

E/S DE LA PERIFERIA:

Esta zona se emplea para tener acceso directo a los módulos de E/S externos que pueden ser añadidos a la CPU.

ESTADO DE TEMPORIZADORES Y CONTADORES:

El valor de temporización y de contaje, preselección y estado actual, se almacena en esta área. Por batería se pueden retener los valores de contaje y temporización que deseemos.

MODULOS DE DATOS:

Aquí podemos almacenar constantes y valores obtenidos mediante operaciones de cualquier longitud (bit, byte, etc.). Estos módulos pueden ser accesibles desde cualquier módulo de programa.

DATOS TEMPORALES:

Aquí se almacenan distintos datos, como las pilas de salto, que se utilizan durante la ejecución del programa y se pierden al final de cada ciclo.

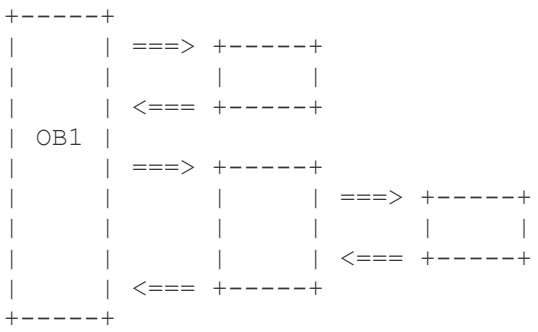
4.2. Tipos de módulos

El Simatic S7 dispone de una serie de módulos que dividen la memoria de programa y la de datos en secciones, permitiendo una programación estructurada y un acceso ordenado a los datos. El número de módulos va a depender del tipo de CPU empleada, disponiendo en general de los siguientes:

Módulos de organización (OB)

Constituyen la forma de comunicación entre el sistema operativo de la CPU y el programa de usuario. Existen 3 tipos de OB, los cuales están accesibles o no según el tipo de CPU:

- OB 1 (ciclo libre): es el módulo principal, el que se ejecuta cíclicamente y del que parten todos los saltos a otros módulos.



- OB de error y alarma: son los que contienen la secuencia de acciones a realizar en caso de que se produzca una alarma o error programado (ver 4.6).

- OB de arranque: en este módulo podemos introducir valores por defecto que permiten el arranque definido a la instalación, bien en un arranque inicial o tras un fallo en la alimentación.

Módulos de código (FC)

Son módulos en los que podemos incluir parte del programa de usuario con lo que obtenemos un programa mucho más estructurado. A estos módulos se pueden acceder desde otro módulo FC o desde un módulo OB.

Módulos de funciones (FB)

Son módulos de programa especiales. Aquí se introducen las partes de programa que aparecen con frecuencia o poseen gran complejidad. Posee una zona de memoria asignada para guardar variables (módulo de datos de instancia). Lo que se hace es enviar parámetros al FB y guardar algunos de los datos locales en el módulo de datos de instancia.

Módulos de datos (DB)

Son áreas de memoria destinadas a contener datos del programa de usuario. Existen módulos de datos globales y de instancia. A los datos contenidos en un módulo de datos es posible acceder de forma absoluta o simbólica. Los datos complejos o compuestos pueden depositarse en forma de estructura. Los módulos de datos pueden ser de dos tipos:

- Módulos de datos globales: se pueden utilizar por cualquier módulo del programa.

- Módulos de datos de instancia: se asignan a un determinado modulo de función y solo pueden manejarse desde dicho módulo. Pueden asignarse varios módulos de datos de instancia a un módulo de función.

Módulos de funciones especiales (SFB)

Se tratan de módulos ya programados, los cuales están preparados para realizar acciones complejas como regulación PID (lazo cerrado), medida de frecuencia, etc...

Módulos de funciones del sistema (SFC)

Son funciones integradas en el sistema operativo de la CPU y que se pueden llamar en caso de necesidad desde el programa de usuario.

4.3. Tipos de datos

Los operandos de las instrucciones se componen de un dato que puede ser de distintos tipos. Los tipos de datos posibles son:

```
E       entrada
A       salida
M       marca
P       periferia (acceso directo)
L       datos locales
T       temporizador
Z       contador
DB      módulo de datos
```

Cada uno de estos tipos se pueden direccionar en 4 posibles modos (salvo T y Z):

- Por defecto (X para DB): Bit.
- B: byte (8 bits).
- W: palabra (16 bits).
- D: palabra doble (32 bits).

4.4. Marcas de memoria

Cuando realicemos nuestro programa y operemos a nivel de bit en operaciones lógicas (and, or, etc.) puede que nos aparezca la necesidad de almacenar el resultado lógico que tengamos en un determinado momento. Para ello disponemos de 256 marcas de memoria de 1 byte, es decir un total de 2048 marcas de 1 bit, que podemos direccionar como:

```
Marcas                          M       0.0 a 255.7
Byte de marcas                  MB      0 a 255
Palabra de marcas               MW      0 a 254
Palabra doble de marcas         MD      0 a 252
```

4.5. Entradas y salidas

Tal y como comentamos anteriormente, manejaremos una imagen de las entradas y las salidas. El número de e/s disponibles dependerá del tipo de CPU que empleemos, además de los módulos externos que tengamos conectados. Como máximo el autómata puede manejar hasta 65536 bytes para cada tipo de e/s. En cada caso podemos direccionar como:

IMAGEN DEL PROCESO DE LAS ENTRADAS (PAE):

Entrada	E	0.0 a 65535.7
Byte de entrada	EB	0 a 65535
Palabra de entrada	EW	0 a 65534
Palabra doble de entrada	ED	0 a 65532

IMAGEN DEL PROCESO DE LAS SALIDAS (PAA):

Salida	A	0.0 a 65535.7
Byte de salida	AB	0 a 65535
Palabra de salida	AW	0 a 65534
Palabra doble de salida	AD	0 a 65532

ENTRADAS EXTERNAS:

Byte de entrada de la periferia	PEB	0 a 65535
Palabra de entrada de la periferia	PEW	0 a 65534
Palabra doble de entrada de la periferia	PED	0 a 65532

SALIDAS EXTERNAS:

Byte de salida de la periferia	PAB	0 a 65535
Palabra de salida de la periferia	PAW	0 a 65534
Palabra doble de salida de la periferia	PAD	0 a 65532

Todas estas entradas y salidas pueden ser de tres tipos:

- E/S digitales: son las e/s más frecuentes y que en mayor cantidad vamos a tener. Ocupan 4 bytes de memoria de direcciones, comenzando desde la 0.0 hasta la 127.7.
- E/S digitales de alarma/error: no son e/s adicionales, se configuran dentro de Step7 y ocupan una de las e/s digitales normales.
- E/S analógicas: estas si son e/s adicionales, pero no obstante hay que configurarlas también desde Step7 para especificar el rango de direcciones que van a ocupar. Ocupan 2 bytes de memoria de e/s (16 bytes por módulo) y se sitúan en el rango de direcciones 256 a 383.

4.6. Registros

Todas las CPU Simatic S7 disponen de una serie de registros que se emplean durante la ejecución del programa de usuario. No vamos a comentar todos ellos, sólo los que realmente empleemos en la programación:

Acumuladores (ACU1 y ACU2)

El acumulador 1 (ACU 1) y el acumulador 2 (ACU 2) son dos registros universales de 32 bits que se emplean para procesar bytes, palabras y palabras dobles. En estos acumuladores se pueden cargar constantes o valores depositados en la memoria como operandos y ejecutar operaciones lógicas con ellos. También es posible transferir el resultado en ACU 1 a una dirección (un módulo de datos, una salida, etc.).

Cada acumulador puede descomponerse en dos palabras de 16 bits (palabra baja y alta). La palabra baja contiene los bits de menor peso y la alta los de mayor peso lógico.

Todas las posibles operaciones que pueden realizarse son:

- Cargar: que siempre actúa sobre ACU 1 y guarda el antiguo contenido en ACU 2 (perdiéndose el valor antiguo de ACU 2). La carga de una palabra actúa sobre la palabra baja del ACU 1.

- Transferir: copia el contenido de ACU 1 en una dirección de memoria, sin perder el valor de los acumuladores.

- Intercambiar el contenido de los acumuladores: mediante la instrucción TAK.

- Realizar una operación entre los acumuladores, almacenando el resultado en ACU 1 sin variar ACU 2. Las operaciones pueden ser de comparación, de lógica digital y de aritmética.

Palabra de estado

Es un registro de 16 bits que contiene algunos bits a los que puede accederse en el operando de operaciones lógicas de bits y de palabras. Solo nos serán de utilidad los 9 primeros bits, estando reservados el uso de los 7 últimos. A continuación pasaremos a describir cada bit:

- BIT 0 (ER): 0 indica que la siguiente línea se ejecuta como nueva consulta (inhibida). En este estado la consulta se almacena directamente en RLO (ver 4.1).

- BIT 1 (RLO): resultado lógico. Aquí se realizan las operaciones a nivel de bit (como AND, OR, etc.).

- BIT 2 (STA): bit de estado. Solo sirve en el test de programa.

- BIT 3 (OR): se requiere para el proceso Y delante de O. Este bit indica que una operación Y ha dado valor 1, en las restantes operaciones es 0.

- BIT 4 (OV): bit de desbordamiento. Se activa (1) por una operación aritmética o de comparación de coma flotante tras producirse un error (desbordamiento, operación no admisible, o relación incorrecta).

- BIT 5 (OS): bit de desbordamiento memorizado. Se activa junto con OV e indica que previamente se ha producido un error. Solo puede cambiar a cero con la instrucción SPS, una operación de llamada a módulo, o porque se ha alcanzado el fin del módulo.

- BITS 6 (A0) y 7 (A1): códigos de condición. Dan información sobre los resultados o bits siguientes:

  - resultado de una operación aritmética.
  - resultado de una comparación.
  - resultado de una operación digital.
  - bits desplazados por una instrucción de desplazamiento o rotación.

- BIT 8 (RB): resultado binario. Permite interpretar el resultado de una operación de palabras como resultado binario e integrarlo en la cadena de combinaciones lógicas binarias.

Registros 1 y 2 de direcciones

Son dos registros de 32 bits cada uno. Se emplean como punteros en operaciones que utilizan un direccionamiento indirecto de registros.

Pila de paréntesis

Aquí se almacenan los bits RB, RLO y OR, además del código de función que especifica que instrucción lógica ha abierto el paréntesis. Tiene un tamaño de 8 bytes (máximo anidamiento).

Pila Master Control Relay (MCR)

Almacena los bits que indican si se opera dentro de un área MCR. Para el caso de emplear saltos guarda los datos en una pila (8 niveles).

4.7. Temporizadores y contadores

TEMPORIZADORES (T):

En el Simatic S7 vamos a disponer de una serie de temporizadores que nos van a permitir realizar una serie de acciones:

- Realizar tiempos de espera.
- Supervisar acciones durante un tiempo determinado (tiempo de vigilancia).
- Generar impulsos.
- Medir tiempos de proceso.

Para la utilización de los temporizadores vamos a disponer de una serie de instrucciones que nos permitirán emplear los temporizadores de distintas formas para adecuarnos a nuestras necesidades, tal y como veremos en capítulos posteriores.

Vamos a disponer de 256 temporizadores, los cuales direccionaremos como:

T 0 a T 255

CONTADORES (Z):

Al igual que los temporizadores vamos a disponer de una serie de contadores que nos permitirán efectuar contajes, tanto hacia adelante como hacia atrás.

También vamos a emplear una serie de instrucciones que permitirán manejarlos, las cuales se verán en siguientes capítulos.

Disponemos de 256 contadores, los cuales podemos direccionar como:

Z 0 a Z 255

5. Programación en AWL

5.1. Tratamiento de los resultados

Dependiendo del tipo de operando que empleemos, se hará uso de uno o varios de los siguientes registros:

- Bit de resultado lógico (RLO): aquí se almacena el resultado de operaciones lógicas a nivel de bit y primera consulta.
- Acumuladores (ACU 1 y ACU 2): aquí se almacenan los operandos y resultados de operaciones lógicas a nivel de byte, palabra, y doble palabra.

Un operando del tipo bit sería una entrada o salida digital, por ejemplo.

Un operando del tipo byte o superior sería la lectura de una entrada analógica, por ejemplo.

5.2. Primera consulta

Cuando efectuamos una asignación, o se comienza un nuevo ciclo de programa, se está en estado de primera consulta. Es decir, la primera instrucción lógica que se efectúe servirá para situar su operando en el RLO.

Las operaciones S y R también producen que el bit de primera consulta se ponga a 0.

Da igual si se trata de una operación AND, OR, o XOR, en los tres casos se introduce el operando en el RLO de forma directa. Si tratamos con instrucciones NAND, NOR, o XOR se introducirá el operando de forma negada (si es un 0 el bit RLO será 1).

5.3. ASIGNACION

Instrucción "="

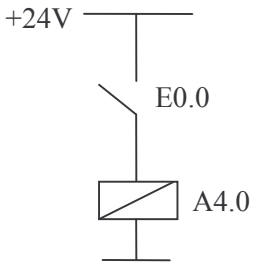
Se copia el contenido del RLO al operando especificado, sin perder el contenido del RLO.

Posibles operandos: E, A, M, DBX, DIX, L

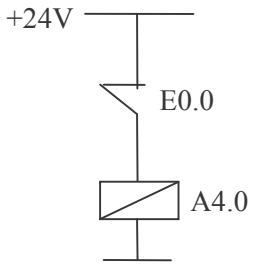
Registros afectados: ER, STA

```
ej. =  E  2.0    //copia el RLO a la entrada E 2.0
```

Esquema eléctrico:



Contacto normalmente abierto



Contacto normalmente cerrado

5.4. Función AND (Y)

Esquema eléctrico:

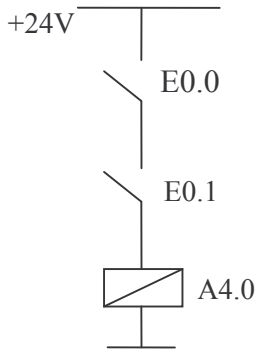


Tabla de verdad		
E0.0	E0.1	A4.0
0	0	0
0	1	0
1	0	0
1	1	1

Instrucción "U"

Realiza la función lógica AND entre el RLO y el operando especificado, almacenando el resultado en RLO (se pierde el valor anterior). Se puede operar con el negado del operando si se adjunta "N" (UN).

Posibles operandos: E, A, M, DBX, DIX, L, T, Z

Registros afectados: RLO, STA

```
ej. U  E  0.0    //realiza un AND entre el RLO y la entrada E 0.0
ej. UN A  1.2    //realiza un AND entre el RLO y la salida A 1.2
negada
```

5.5. Función OR (O)

Esquema eléctrico:

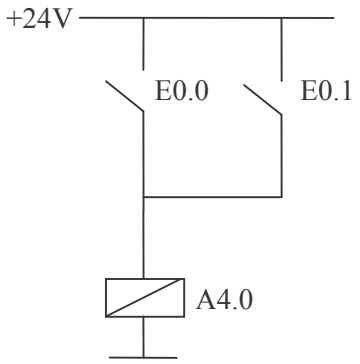


Tabla de verdad		
E0.0	E0.1	A4.0
0	0	0
0	1	1
1	0	1
1	1	1

Instrucción "O"

Realiza la función lógica OR entre el RLO y el operando especificado, almacenando el resultado en RLO (se pierde el valor anterior). Se puede operar con el negado del operando si se adjunta "N" (ON).

Posibles operandos: E, A, M, DBX, DIX, L, T, Z

Registros afectados: RLO, STA

```
ej. O  T  0      //realiza un OR entre el RLO y el estado del temporizador T 0
ej. ON M  5.0    //realiza un OR entre el RLO y la marca M 5.0 negada
```

5.6. Función XOR (O exclusiva)

Esquema eléctrico:

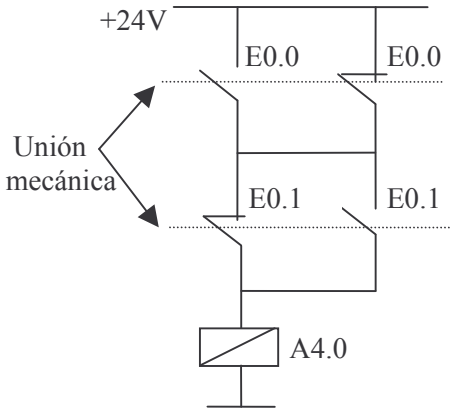


Tabla de verdad		
E0.0	E0.1	A4.0
0	0	0
0	1	1
1	0	1
1	1	0

Instrucción "X"

Realiza la función lógica XOR entre el RLO y el operando especificado, almacenando el resultado en RLO (se pierde el valor anterior). Se puede operar con el negado del operando si se adjunta "N" (XN).

Posibles operandos: E, A, M, DBX, DIX, L, T, Z

Registros afectados: RLO, STA

```
ej. X  Z  0      //realiza un XOR entre el RLO y el estado del contador Z 0
ej. XN A  1.0    //realiza un XOR entre el RLO y la salida A 1.0 negada
```

5.7. Expresiones entre paréntesis

Instrucciones "U(", "UN(", "O(", "ON(", "X(", "XN(", ")" sin operandos

Las operaciones U, O, X, y sus negaciones UN, ON, y XN permiten ejecutar operaciones lógicas con fracciones de una cadena lógica encerradas entre paréntesis (expresiones entre paréntesis).

Los paréntesis que encierran una fracción de una cadena lógica indican que el programa va a ejecutar las operaciones entre paréntesis antes de ejecutar la operación lógica que precede a la expresión entre paréntesis.

La operación que abre una expresión entre paréntesis almacena el RLO de la operación precedente en la pila de paréntesis. A continuación, el programa combina el RLO almacenado con el resultado de las combinaciones lógicas ejecutadas dentro del paréntesis (siendo la primera operación dentro de los paréntesis de primera consulta).

El número máximo de paréntesis anidados que se permiten es 8.

Registros afectados: RLO, STA, RB, pila de paréntesis

Ejemplo:

```
U(
O   E   0.0
U   E   0.1
)
=   A   2.0
```

Veamos los pasos que sigue el programa en este ejemplo:

- Efectúa un AND en primera consulta, con lo que el resultado de las operaciones dentro del paréntesis se introducirá directamente en RLO.
- Efectuamos un OR con la entrada 0.0, al ser en primera consulta (primera operación dentro del paréntesis) lo que sucede es que el contenido de E 0.0 pasa a ser el nuevo valor del RLO.
- Se efectúa un AND entre el RLO obtenido anteriormente y la entrada 0.1, almacenándose el resultado en el RLO.
- Se cierra el paréntesis, con lo que el RLO de las operaciones efectuadas dentro se opera según la instrucción que inicia el paréntesis (en este caso la instrucción U). Tal y como comentamos, al estar la instrucción de inicio al principio del programa se ejecuta como primera consulta, con lo que el RLO pasará a valer lo que el resultado dentro del paréntesis.
- Copiamos el contenido del RLO en la salida 2.0.

En pocas palabras, si ejecutáramos este programa la salida 2.0 valdría 0 a menos que E 0.0 y E 0.1 valiesen 1, con lo que pasaría a valer 0.

Un programa equivalente sería (en este caso):

```
O   E   0.0    //copiamos la E 0.0 en el RLO (primera c.)
U   E   0.1    //efectuamos un AND entre el RLO y la E 0.1
=   A   2.0    //copiamos el resultado a la salida 2.0
```

5.8. Y antes de O

Instrucción "O" sin operando

Si introducimos una instrucción "O" sin operando seguida de una o varias instrucciones AND se evalúa en primer lugar las instrucciones AND y el resultado se combina con el RLO según un OR.

Esta operación equivale a emplear "O(" con instrucciones del tipo AND dentro del paréntesis.

Registros afectados: RLO, STA, OR, pila de paréntesis

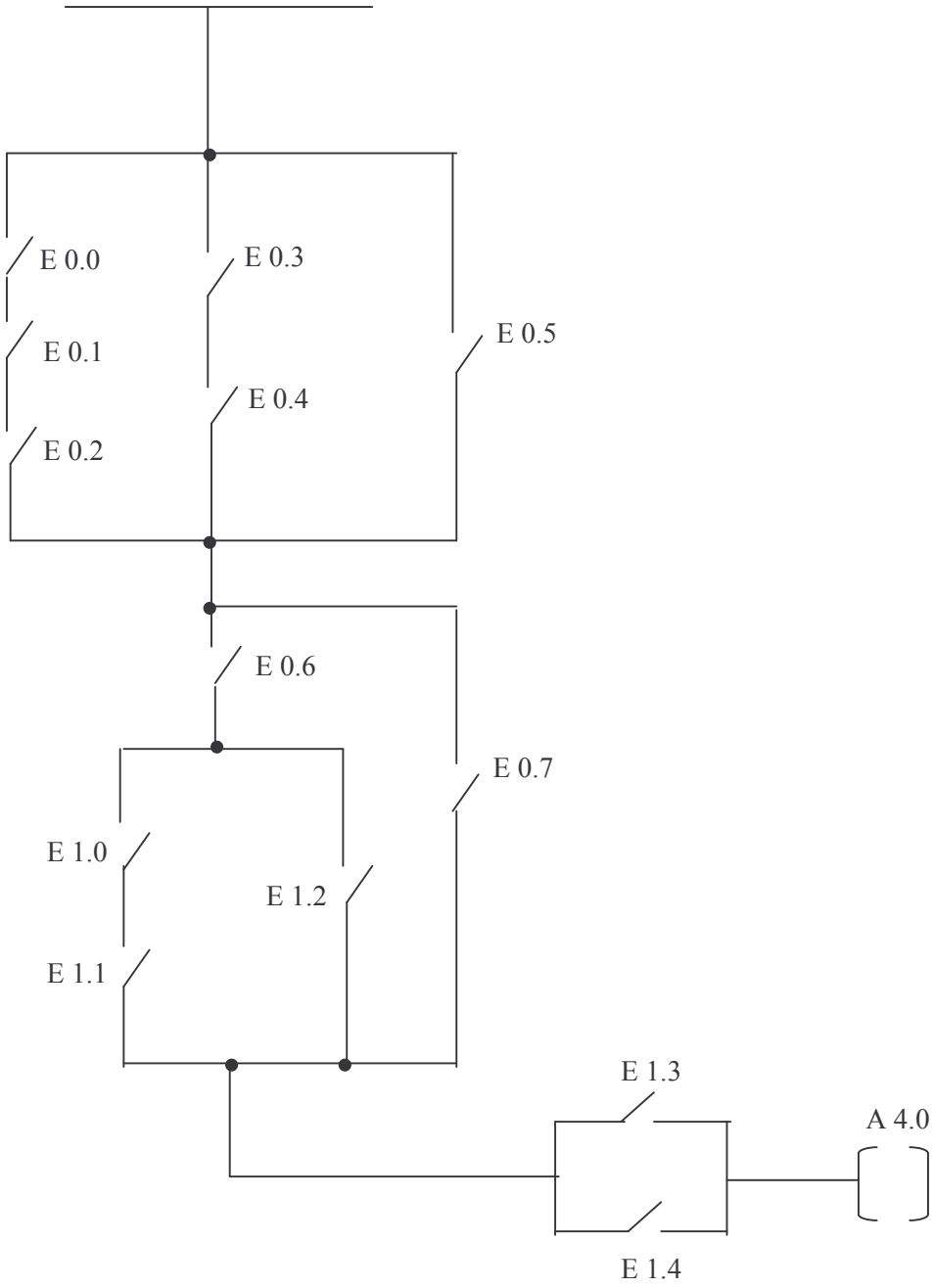
Ejemplo:

```
U   E   0.0    //se introduce en el RLO el valor de la entrada 0.0 (primera c.)
O              //comenzamos una operación Y antes de O
U   E   0.1    //introducimos el valor de la entrada 0.1 en el RLO (primera c.)
U   M   0.3    //efectuamos un AND entre el RLO y la marca 0.3
=   A   4.0    //se finaliza Y antes de O. Se efectúa un OR entre el primer RLO
                 y el RLO resultado de las operaciones AND. Luego se copia el
                 contenido del RLO en la salida 4.0
```

5.9. Ejercicio propuesto

Escribir en AWL el siguiente esquema eléctrico:

1) Sin usar marcas
2) Usando marcas



5.10. Operaciones de flancos

Instrucciones "FP" y "FN"

Las operaciones de flanco positivo (FP) y flanco negativo (FN) pueden utilizarse para detectar cambios de flanco en el RLO. El cambio de 0 a 1 se denomina flanco positivo, mientras que el cambio de 1 a 0 se denomina flanco negativo.

Cada instrucción FP o FN emplea un operando para poder comparar el RLO actual con el que había en el ciclo anterior, se recomienda emplear marcas de memoria.

Si se realiza un cambio de flanco en el sentido de la instrucción empleada, ésta produce un impulso positivo (1) en el RLO durante el ciclo actual.

Posibles operandos: E, A, M, DBX, DIX, L

Registros afectados: RLO, STA
Se emplea una operando para almacenar el RLO

Ejemplo:

```
U  E  1.0    //empleamos la entrada 1.0 para detectar un cambio de flanco
FP M  1.0    //empleamos la marca 1.0 para detectar el cambio de flanco
=  A  4.0    //asignamos el resultado de la operación FP a la salida 4.0
```

En este ejemplo cada vez que introduzcamos un flanco positivo en la entrada 1.0 se producirá un impulso de longitud un ciclo en la salida 4.0, tal y como se muestra en la siguiente figura:

```
E 1.0:  0 0 1 1 1 0 0 0 1 1 0
M 1.0:  0 0 1 1 1 0 0 0 1 1 0
A 4.0:  0 0 1 0 0 0 0 0 1 0 0

ciclo:     1 2 3 4 5 6 7 8 9 10
```

Para el caso de sustituir en el ejemplo FP por FN, se obtendría:

```
E 1.0:  0 0 1 1 1 0 0 0 1 1 0
M 1.0:  0 0 1 1 1 0 0 0 1 1 0
A 4.0:  0 0 0 0 0 1 0 0 0 0 1

ciclo:     1 2 3 4 5 6 7 8 9 10
```

ATENCION:
Es obligatorio no emplear los operandos ocupados por FP y FN para otros fines, ya que entonces se falsifica el RLO almacenado en ellos y por lo tanto se produce un funcionamiento incorrecto del programa.

5.11. Set y Reset

Instrucciones "S" y "R"

La operación set (S) fuerza a uno el operando especificado si el RLO es 1.
La operación reset (R) fuerza a cero el operando especificado si el RLO es 1.

En ambos casos el bit de primera consulta se hace 0.

Posibles operandos: E, A, M, D, DBX, DIX, L

Registro afectados: ER

Ejemplo:

```
U  E  1.0    //copiamos al RLO el valor de la entrada 1.0 (primera c.)
S  A  4.0    //si RLO=1 se fuerza la salida 4.0 a 1
U  E  1.1    //copiamos al RLO el valor de la entrada 1.1 (primera c.)
R  A  4.0    //si RLO=1 se fuerza la salida 4.0 a 0
```

En este ejemplo (báscula S-R) tiene preferencia el reset sobre el set, ya que esta última instrucción se ejecuta al después, es decir si las entradas 1.0 y 1.1 fuesen 1 la salida 4.0 sería 0.

5.12. Negar, activar, desactivar y salvar el RLO

Instrucciones "NOT", "SET", "CLR" y "SAVE" sin operando

NOT

Niega (invierte) el RLO actual al no haberse activado el bit OR.
Registros afectados: RLO se invierte, STA=1

SET

Fuerza el RLO de forma incondicional a 1.
Registros afectados: RLO=1, STA=1, ER=0, OR=0

CLR

Fuerza el RLO de forma incondicional a 0.
Registros afectados: RLO=0, STA=0, ER=0, OR=0

SAVE

Almacena el RLO en el registro de estado (en el bit RB). El RLO almacenado puede ser consultado de nuevo con la instrucción "U BR".
Registros afectados: RB almacena el valor de RLO.

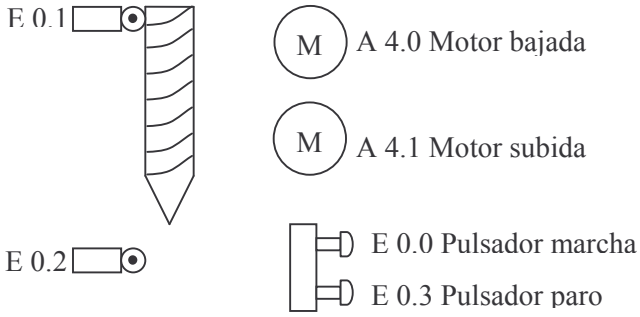
5.13. Ejercicios propuestos

Ejercicio 1: Taladradora

Funcionamiento:

En el estado de reposo la taladradora estará arriba, pisando el final de carrera (E0.1)

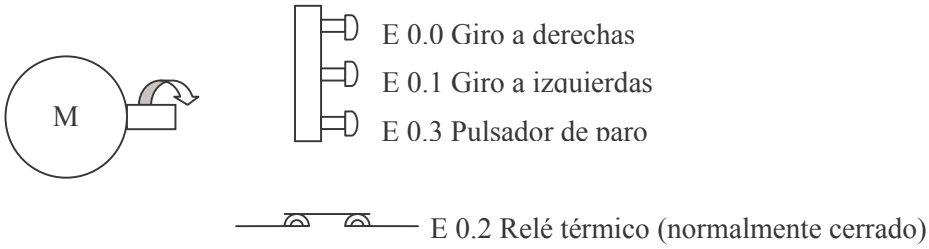
Si se pulsa la marcha (E0.0) la taladradora bajará accionado por el motor de bajada (A4.0). Cuando se active el final de carrera de abajo (E0.2), la taladradora subirá de nuevo. Si en algún momento se pulsa el interruptor de parada (E0.3), la taladradora deberá subir.



Ejercicio 2: Motor

Funcionamiento:

El motor podrá girar a derechas (A 4.0) o izquierdas (A 4.1) según le demos al pulsador correspondiente. Además existe un pulsador de paro (E0.3), y un relé térmico normalmente cerrado (E0.2) que se abrirá cuando en el motor se produzca una sobretemperatura.



6. Acumuladores - Operaciones de carga y transferencia

ACU1 y ACU2 son dos registros de 32 bits para el procesamiento de bytes, palabras y doble palabras. Permiten programar un intercambio de información entre módulos de E/S y áreas de memoria.

- Imagen de proceso de entradas y salidas
- Marcas
- Temporizadores y contadores
- Áreas de datos

Las instrucciones de carga y transferencia transfieren datos a ó desde el ACU1.

Son operaciones incondicionales, o sea, independientes del RLO

Ejemplo:

```
U  E1.0
L  2    //Carga el número 2 en ACU1 independientemente del estado de E1.0
```

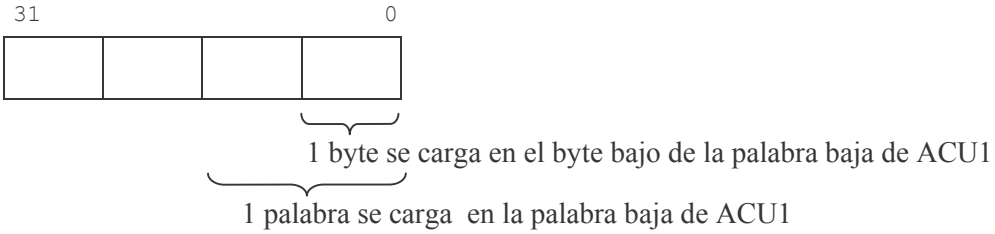
6.1. Operación de carga

Instrucción "L"

Carga en ACU1 constantes y valores de los operadores.

Ejemplo:

```
L  3    //Carga el número entero 3 (16 bits) en el ACU1
L  EB0  //Carga el byte de entradas 0 en ACU1
L  MB20 //Carga el byte de marcas 20 en ACU1
```



Los bytes no aprovechados se ponen a cero.

La información que hubiese en ACU1 se desplaza a ACU2. Y lo que hubiese en ACU2 se pierde.

Ejemplo:

```
L  3    //ACU1=3
L  2    //ACU1=2 ; ACU2=3
L  5    //ACU1=5 ; ACU2=2   se pierde
```

6.2. Operación de transferencia

Instrucción "T"

Transfiere el contenido de ACU1 a una dirección de operando. La transferencia no cambia el contenido de los acumuladores. Transferimos únicamente desde ACU1.

Ejemplo:

```
T   AB10   //Transfiere el byte más bajo del ACU1 al byte de salida 0.
T   MW4    //Transfiero el contenido de la palabra baja de ACU1 a la
             palabra de marcas MW14
```

Observaciones sobre la carga y transferencia:

- Los acumuladores sirven para ejecutar operaciones lógicas entre ACU1 y ACU2 (comparación, aritmeticas, AND, OR...). El resultado siempre se almacena en ACU1, sobreescribiendo lo que hubiese en ACU1 pero el contenido de ACU2 permanece intacto.

Ejemplo:

```
L   8    //ACU1=8
T   2    //ACU1=2 ; ACU2=8
-I       //ACU2-ACU1 ⇒ ACU1=6 ; ACU2=8
T MW10   //Transfiero el resultado de la resta a MW10
```

- En cada ciclo de programa ACU1 y ACU2 se ponen a cero.

Ejemplo:

```
T   MW10   //en el segundo ciclo y sucesivos jamás se transferirá 5
L   5
```

7. Operaciones de contaje

7.1. Operaciones con contadores

Los contadores permiten distintas operaciones, que debemos emplear en su manejo:

- Cargar un valor de contaje (preselección).
- Borrar el contaje.
- Contar hacia adelante y hacia atrás.
- Consultar su estado como un operando más en operaciones lógicas de bit.
- Consultar su valor en ACU 1.

Todas estas operaciones serán explicadas con profundidad en los siguientes puntos.

7.2. Cargar un valor de contaje

Instrucciones: "L C#" y "S Z"

Un contador se pone a un determinado valor cargando dicho valor en la palabra baja del ACU 1, mediante una operación de carga, y luego en el contador, mediante una instrucción Set.

"L C#" introduce un valor de contaje en la palabra baja del ACU 1. El valor de contaje puede ser un valor comprendido entre 0 y 999.

Registros afectados: ACU 1, ACU 2

"S Z" introduce el valor de contaje en ACU 1 en el contador si RLO vale 1.

Registros afectados: ER

Ejemplo:

```
L   C#3       //introduce el valor de contaje 3 en el ACU 1
U   E   1.0   //carga en el RLO el valor de la entrada 1.0
S   Z   1     //introduce el valor 3 (dentro de ACU 1) en el contador 1 si
                la entrada 1.0 es 1
```

7.3. Borrar un contador

Instrucción: "R Z"

Borra el contador especificado (puesta a cero) si el RLO vale 1.

Registros afectados: ER

Ejemplo:

```
U   E   1.0   //carga en el RLO el valor de la entrada 1.0
R   Z   1     //borra el contador 1 (a cero) si la entrada 1.0 es 1 (RLO=1)
```

7.4. Contaje hacia adelante y hacia atrás

Instrucciones: "ZV" y "ZR"

"ZV" incrementa el contador especificado si hay un cambio de flanco ascendente (0 a 1) en el RLO. El incremento se produce en una unidad. Cuando el contador alcanza el límite superior de 999, se detiene y no sigue incrementando.

"ZR" decrementa el contador especificado si hay un cambio de flanco ascendente (0 a 1) en el RLO. El decremento se produce en una unidad. Cuando el contador alcanza el límite inferior de 0, se detiene y no sigue decrementando.

Registros afectados: ER

Ejemplos:

```
U  E  0.0    //carga en el RLO el valor de la entrada 0.0
ZV Z  1      //incrementa el contador 1 si la entrada 0.0 presenta un
               cambio de flanco ascendente

U  E  1.0    //carga en el RLO el valor de la entrada 1.0
ZR Z  1      //decrementa el contador 1 si la entrada 1.0 presenta un
               cambio de flanco ascendente
```

7.5. Consulta del estado de contadores

El programa puede consultar el estado de un contador de la misma manera que consulta el estado de señal de una entrada o salida, pudiendo combinar el resultado de la consulta.

Cuando se consulta el estado del contador con las operaciones U, O, o X el resultado es 1 si el valor de contaje es mayor que 0.

Ejemplo:

```
L  C#5       //introduce el valor de contaje 5 en el ACU 1
U  E  2.0    //carga en el RLO el valor de la entrada 2.0
S  Z  1      //introduce el valor 5 (dentro de ACU 1) en el contador 1 si
               la entrada 2.0 es 1
U  E  1.0    //carga en el RLO el valor de la entrada 1.0
ZR Z  1      //decrementa el contador 1 si la entrada 1.0 presenta un
               cambio de flanco ascendente
U  Z  1      //introduce en el RLO el estado del contador 1
=  A  0.0    //introduce el estado del contador 1 en la salida 0.0
```

7.6. Lectura de un valor de contaje

Instrucciones: "L Z" y "LC Z"

Con la instrucción "L Z" introducimos en el ACU 1 (parte baja) el valor del contador especificado en binario. El valor en ACU 1 puede ser introducido en otro contador.

Con la instrucción "LC Z" introducimos en el ACU 1 (parte baja) el valor del contador especificado en BCD. En esta codificación no es posible pasar el valor de ACU 1 a otro contador.

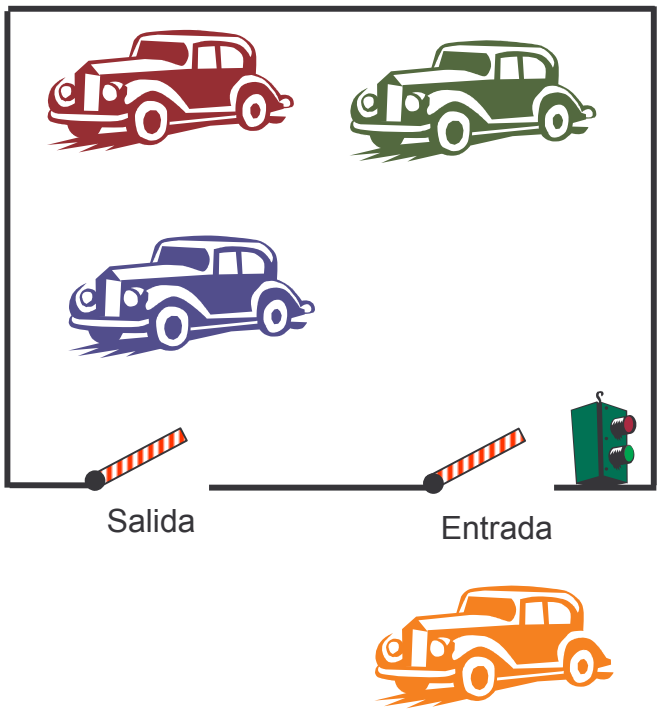
Registros afectados: ACU 1, ACU 2

Ejemplos:

```
L  Z  1       //introduce el valor del contador 1 en el ACU 1
LC Z  2       //introduce el valor del contador 2 en el ACU 1 en BCD
```

7.7. Ejercicios propuestos

***Ejercicio 1*: Control de un garaje**



Automatizar un garaje de cinco plazas de tal forma que si éste se encuentra lleno, se encienda una luz indicándolo y no suba la barrera. En caso contrario deberá estar encendida otra luz indicando "LIBRE".

El garaje consta de 5 plazas.

Disponemos de una célula fotoeléctrica y una barrera en la entrada y lo mismo en la salida.

Asignación de variables	
E0.0	Célula fotoeléctrica de entrada
E0.1	Célula fotoeléctrica de salida
A4.0	Barrera de entrada
A4.1	Barrera de salida
A4.2	Luz de señalización de "LIBRE"
A4.3	Luz de señalización de "LLENO"

8. Operaciones de temporización

8.1. Operaciones con temporizadores

Los temporizadores permiten distintas operaciones, que debemos emplear en su manejo:

- Funcionamiento en un modo determinado.
- Borrar la temporización.
- Re-arrancar un temporizador (FR).
- Consultar su estado como un operando más en operaciones lógicas de bit.
- Consultar su valor en ACU 1.

Cada temporizador lo podemos hacer funcionar en uno de los siguientes modos:

- Impulso (SI).
- Impulso prolongado (SV).
- Retardo a la conexión (SE).
- Retardo a la conexión con memoria (SS).
- Retardo a la desconexión (SA).

Todas estas operaciones serán explicadas con profundidad en los siguientes puntos.

8.2. Cargar un valor de temporización

El valor de temporización se debe cargar en la parte baja del ACU 1, para desde allí transferirlo al temporizador mediante el set que determine el modo de temporización adecuado.

El tiempo va decrementando hasta ser igual a 0. El valor de temporización puede cargarse en la palabra baja del ACU 1 en formato binario, hexadecimal o BCD. Para ello debemos elegir una base de tiempos y un valor dentro de dicha base, con lo que podemos realizar temporizaciones desde 0 a 9990 segundos (0H_00M_00S_00MS a 2H_46M_30S_00MS).

La siguiente sintaxis permite cargar un valor de temporización predefinido:

L W#16#abcd

a = base de tiempos
bcd = valor de temporización en formato BCD

Base de tiempos y código respectivo:

```
10 ms           0
100 ms          1
1 s             2
10 s            3
```

Registros afectados: ACU 1, ACU 2

Ejemplo:

```
L  W#16#210    //esto introduce un valor de 10 segundos en ACU 1
                 (2 base de 1s, 10 los segundos que deseamos)
```

L S5T#aH_bbM_ccS_ddMS

a = horas, bb= minutos, cc = segundos, dd = milisegundos

En este caso la base de tiempos se selecciona de forma automática, tomándose la de valor más bajo posible. Debido a esto los valores de resolución demasiado alta se redondean por defecto, alcanzando el rango pero no la resolución deseada.

Las posibles resoluciones y rangos son:

```
0,01 s          10MS a 9S_990MS
0,1 s           100MS a 1M_39S_900MS
1 s             1S a 16M_39S
10 s            10S a 2H_46M_30S
```

Registros afectados: ACU 1, ACU 2

Ejemplo:

```
L  S5T#00H02M23S00MS    //esto introduce un valor de temporización de
                          2 minutos y 23 segundos en el ACU 1
```

8.3. Consulta del estado de temporizadores.

El programa puede consultar el estado de un temporizador de la misma manera que consulta el estado de señal de una entrada o salida, pudiendo combinar el resultado de la consulta.

Cuando se consulta el estado del temporizador con las operaciones U, O, o X el resultado es 1 si el valor de la salida del temporizador es 1.

8.4. Temporizador como impulso (SI)

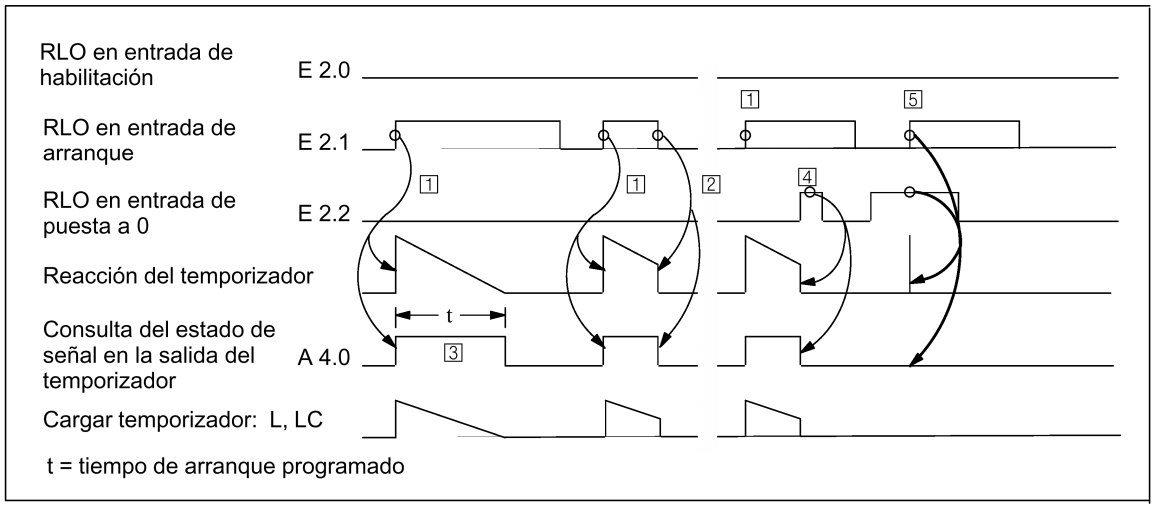
Instrucción: "SI"

Si el RLO (al ejecutar esta instrucción) cambia de 0 a 1, el temporizador arranca. El temporizador marcha con el valor de tiempo indicado en ACU1. Si el RLO cambia de 1 a 0 antes de terminar el tiempo, el temporizador se detiene. La salida del temporizador entrega 1 mientras el temporizador corre.

Registros afectados: ER

Ejemplo:

```
U   E   0.0      //Empleamos la entrada 0.0 como entrada del temporizador
L   S5T#45s      //Introducimos un valor de temporización de 45 segundos
SI  T   2        //Empleamos el temporizador 2 como impulso
U   T   2        //Leemos la salida del temporizador
=   A   0.1      //Asignamos la salida del temporizador a la salida 0.1
```



8.5. Temporizador como impulso prolongado (SV)

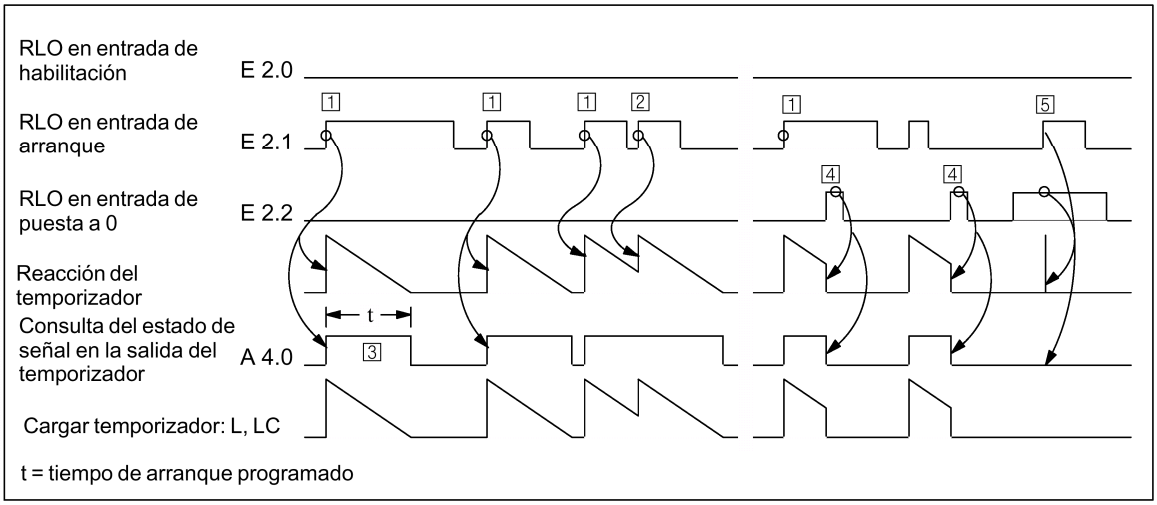
Instrucción: "SV"

Si el RLO (al ejecutar esta instrucción) cambia de 0 a 1, el temporizador arranca y continua en marcha incluso si el RLO cambia a 0 antes de que el temporizador termine. Mientras el tiempo está corriendo, la salida vale 1.

Registros afectados: ER

Ejemplo:

```
U  E  0.2      //Empleamos la entrada 0.2 como entrada del temporizador
L  S5T#85s     //Introducimos un valor de temporización de 85 segundos
SV T  9        //Empleamos el temporizador 9 como impulso prolongado
U  T  9        //Leemos la salida del temporizador
=  A  9.1      //Asignamos la salida del temporizador a la salida 9.1
```



8.6. Temporizador como retardo a la conexión (SE)

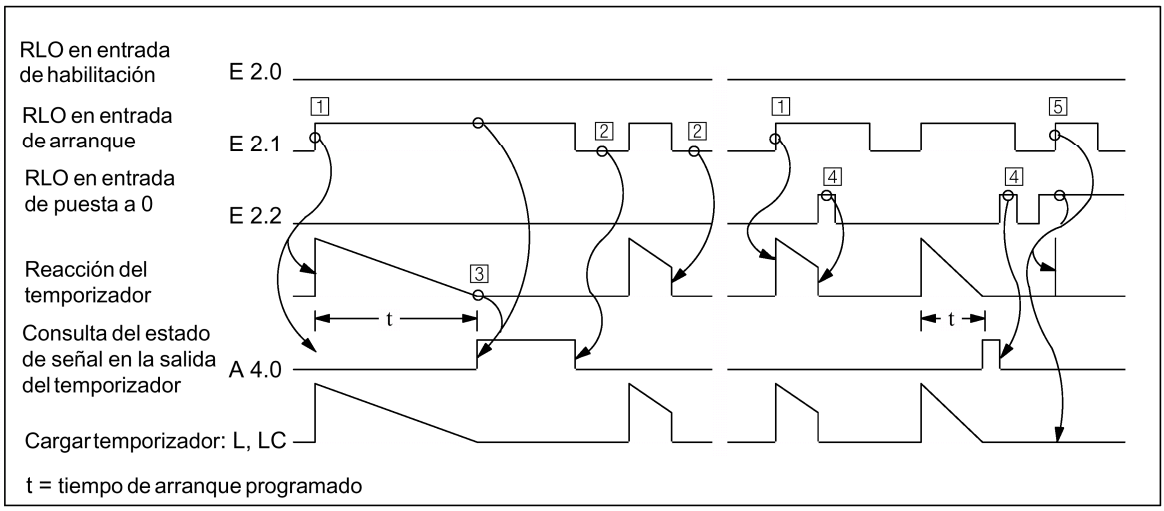
Instrucción: "SE"

El temporizador arranca cuando hay un flanco creciente en el RLO (al ejecutar esta instrucción). El temporizador continua en marcha con el valor de temporización indicado en el ACU 1 mientras sea positivo el estado de señal en la entrada (el RLO). El estado de la salida es 1 si el tiempo ha transcurrido sin errores y si el estado de la entrada (RLO) es 1. Si la entrada (RLO) cambia de 1 a 0 mientras está en marcha el temporizador, éste cambia el estado de la salida a 0.

Registros afectados: ER

Ejemplo:

```
U  E  0.7      //Empleamos la entrada 0.7 como entrada del temporizador
L  S5T#65s     //Introducimos un valor de temporización de 65 segundos
SE T  4        //Empleamos el temporizador 4 como retardo a la conexión
U  T  4        //Leemos la salida del temporizador
=  A  8.1      //Asignamos la salida del temporizador a la salida 8.1
```



8.7. Temporizador como retardo a la conexión con memoria (SS)

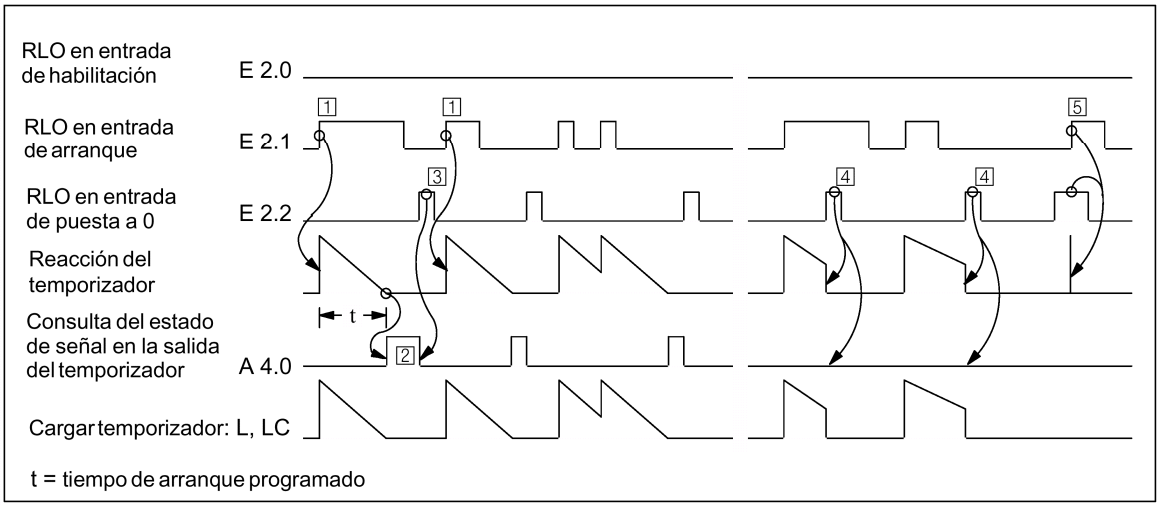
Instrucción: "SS"

Si la entrada (RLO en la ejecución de la instrucción) cambia de 0 a 1, el temporizador arranca y continua corriendo incluso si la entrada (RLO) cambia a 0, antes que el temporizador termine de contar. Si el tiempo ha concluido la salida continua a 1 independientemente del estado de la entrada (RLO). Solo se puede poner a 0 la salida mediante un Reset. El temporizador vuelve a arrancar con el valor de temporización indicado en el ACU 1 si el estado de la señal en la entrada (RLO) cambia de 0 a 1 mientras el temporizador está en marcha.

Registros afectados: ER

Ejemplo:

```
U   E   1.2      //Empleamos la entrada 1.2 como entrada del temporizador
L   S5T#32s      //Introducimos un valor de temporización de 32 segundos
SS  T   2        //Empleamos el temporizador 2 como retardo a la c. con memoria
U   T   2        //Leemos la salida del temporizador
=   A   3.1      //Asignamos la salida del temporizador a la salida 3.1
```



8.8. Temporizador como retardo a la desconexión (SA)

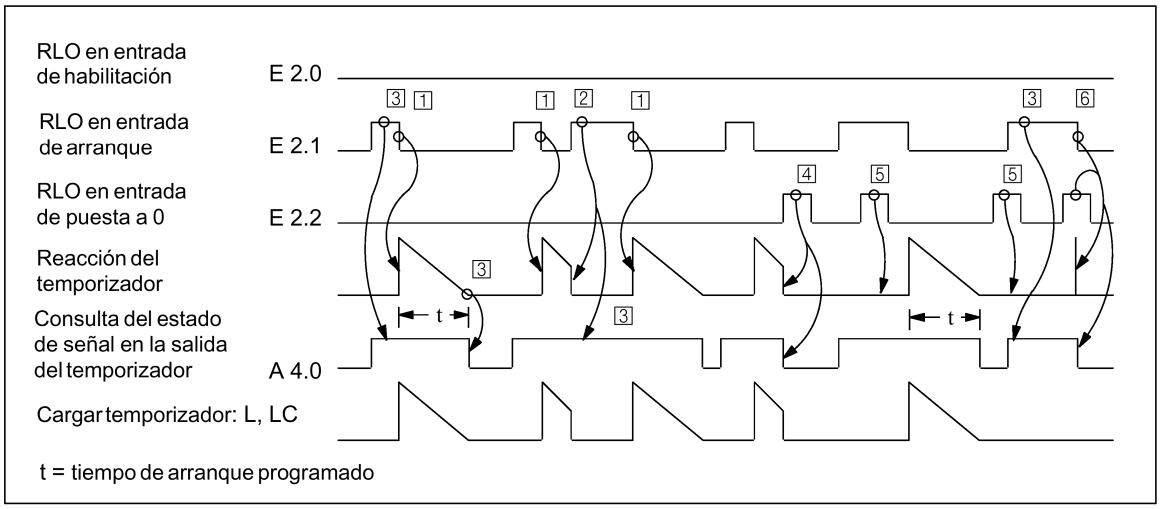
Instrucción: "SA"

Si la entrada (RLO en la ejecución de la instrucción) cambia de 1 a 0, el temporizador arranca y continua corriendo. Si la entrada (RLO) cambia a 1 antes que el temporizador termine de contar, se resetea el temporizador. Mientras el tiempo está corriendo, la salida vale 1.

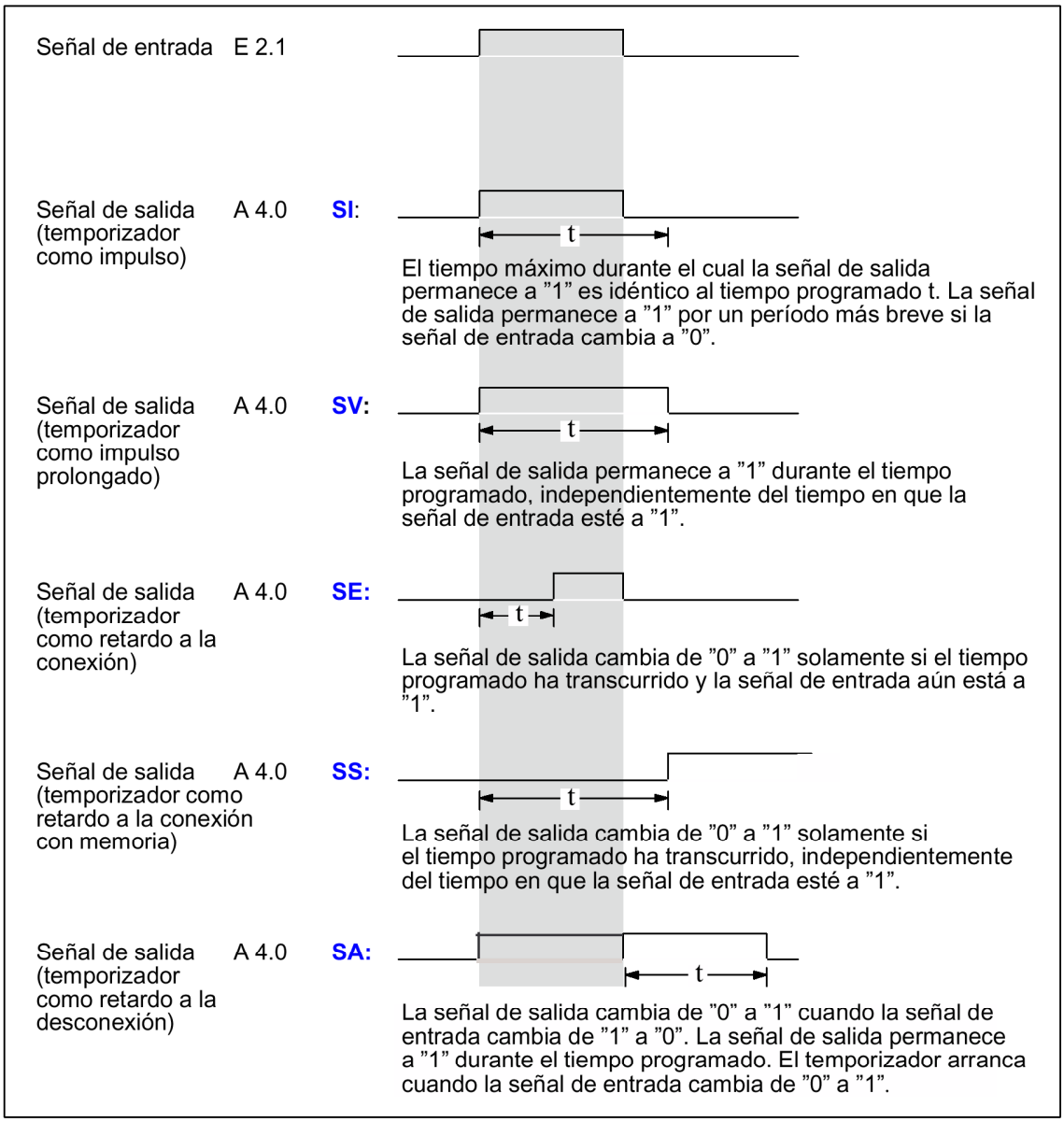
Registros afectados: ER

Ejemplo:

```
U   E   4.2      //Empleamos la entrada 4.2 como entrada del temporizador
L   S5T#32s      //Introducimos un valor de temporización de 32 segundos
SA  T   7        //Empleamos el temporizador 7 como retardo a la desconexión
U   T   7        //Leemos la salida del temporizador
=   A   1.1      //Asignamos la salida del temporizador a la salida 1.1
```



8.9. Elegir el temporizador adecuado



8.10. Borrar una temporización

Instrucción: "R T"

Esta instrucción borra (resetea) el temporizador indicado. El temporizador vuelve al estado de reposo, es decir parado y con la salida igual a 0.

Registros afectados: ER

Ejemplo:

```
U  E  0.0     //Empleamos la entrada 0.0 como entrada del temporizador
L  S5T#2s     //Introducimos un valor de temporización de 2 segundos
SS T  2       //Empleamos el temporizador 2 como retardo a la c. con memoria
U  E  0.1     //Empleamos la entrada 0.1 como entrada de borrado
R  T  2       //Si la entrada 0.1 cambia de 0 a 1 el temporizador 2 se borra
U  T  2       //Leemos la salida del temporizador
=  A  3.1     //Asignamos la salida del temporizador a la salida 3.1
```

8.11. Re-arranque de un temporizador

Instrucción: "FR T"

Cuando el RLO cambia de 0 a 1 (flanco de subida) delante de una operación FR se habilita el temporizador. Este cambio del estado de señal siempre es necesario para habilitar un temporizador.

Para arrancar un temporizador y ejecutar una operación normal de temporizador no hace falta habilitarlo. Esta función se emplea únicamente para redisparar un temporizador que está en marcha, es decir, para rearrancarlo. Este rearranque sólo puede efectuarse cuando la operación de arranque continúa procesándose con un RLO de 1.

Registros afectados: ER

Ejemplo:

```
U  E  2.0     //Empleamos la entrada 2.0 como re-arranque
FR T  1       //Re-arrancamos el temporizador 1 si la E 2.0 pasa a 1
U  E  2.1     //Empleamos la entrada 2.1 como entrada del temporizador
L  S5T#5s     //Introducimos un valor de temporización de 5 segundos
SI T  1       //Empleamos el temporizador 1 como impulso
U  T  1       //Leemos la salida del temporizador
=  A  4.0     //Copiamos la salida del temporizador a la salida 4.0
```

Si el RLO cambia de 0 a 1 en la entrada de re-arranque mientras está en marcha el temporizador, el temporizador vuelve a arrancar. El tiempo programado se emplea como tiempo actual para el re-arranque. Un cambio del RLO de 1 a 0 en la entrada de re-arranque no produce ningún efecto.

Un cambio del RLO de 0 a 1 en la entrada de habilitación no afecta al temporizador si todavía hay un RLO 0 en la entrada del temporizador.

8.12. Lectura de un valor de temporización

Instrucciones: "L T" y "LC T"

Con la instrucción "L T" introducimos en el ACU 1 (parte baja) el valor del temporizador especificado en binario. El valor en ACU 1 puede ser introducido en otro temporizador.

Con la instrucción "LC T" introducimos en el ACU 1 (parte baja) el valor del temporizador especificado en BCD. En esta codificación no es posible pasar el valor de ACU 1 a otro temporizador.

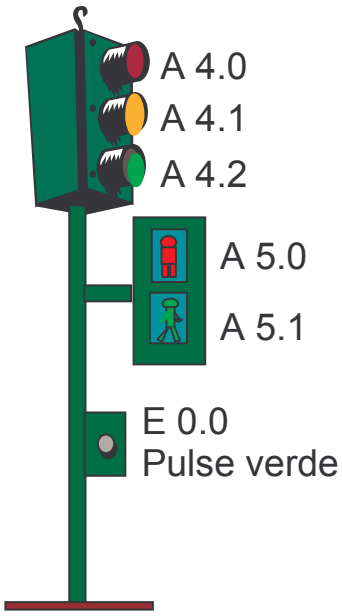
Registros afectados: ACU 1, ACU 2

Ejemplos:

```
L  T  1      //introduce el valor del temporizador 1 en el ACU 1

LC T  2      //introduce el valor del temporizador 2 en el ACU 1 en
BCD
```

8.13. Ejercicios propuestos

***Ejercicio 1*: Control de un semáforo**



Se dispone de un semáforo, el cual en condiciones normales se encuentra del modo siguiente:

• Verde vehículos

• Rojo peatones

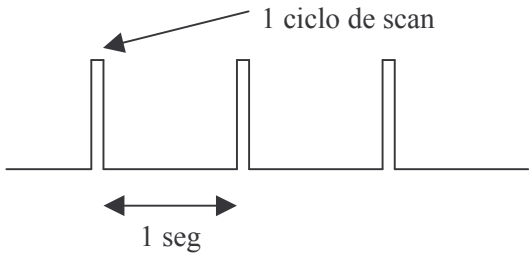
En el mismo instante que un peatón accione sobre el pulsador situado en el semáforo, éste pasará a amarillo para vehículos, estado que durará durante 3". Finalizado éste, pasará a estado rojo para vehículos y verde para peatones.

El tiempo de duración fijado para rojo vehículos: 6"

Finalizado el proceso, el semáforo regresará al estado normal.

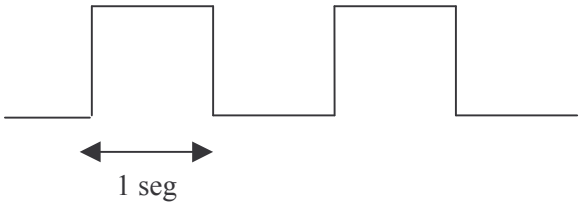
Durante el tiempo de duración del ciclo, deberá evitarse que cualquier nueva activación sobre el pulsador verde, rearme el ciclo.

***Ejercicio 2*: Generador de pulsos**



Funcionamiento: Realizar un tren de pulsos de 1 ciclo de scan con un periodo de 1 seg.

***Ejercicio 3*: Generador de onda cuadrada**



Funcionamiento: Basándose en el generador de pulsos realizar una onda cuadrada periódica de 1 seg.

9. Operaciones de salto

9.1. Operaciones de salto incondicional

Instrucciones: "SPA" y "SPL"

Las operaciones de salto incondicional (SPA) interrumpen el desarrollo normal del programa, haciendo que el mismo salte a una meta determinada (operando de la operación SPA). La meta define el punto en que deberá continuar el programa. El salto se efectúa independientemente de condiciones.

Ejemplo de salto SPA:

```
        U   E  1.0      //cargamos en el RLO el valor de la entrada 1.0
        SPA AQUI        //saltamos de forma incondicional a la línea con meta "AQUI"
        NOP 0           //esta línea no se ejecuta (es saltada)
AQUI:   U   E  2.0      //aquí continua la ejecución del programa
        =   A  3.0      //introducimos el resultado en la salida 3.0
```

La operación Salto a meta (SPL) es un distribuidor de saltos seguido de una serie de saltos incondicionales a metas determinadas (lista de saltos). El salto de la lista se escoge según el valor contenido en el ACU1, es decir si el acu1 vale 0 se escogerá el primer salto incondicional (SPA), si vale 1 se saltará al segundo salto... Si el valor se encuentra fuera de la lista se salta a la meta especificada en SPL.

Una meta se compone de 4 caracteres como máximo. El primer carácter debe ser siempre una letra, no importando si el resto son números o letras. La meta se especifica normalmente en el operando de la instrucción de salto, y seguida de dos puntos frente a la línea que posee la meta (ver ejemplos).

Registros afectados: ninguno

Estructura:

```
        L MB100       //cargamos en el ACU1 un byte
        SPL DEF       //saltamos a DEF si el valor de ACU1 no está en la list
        SPA CERO      //se salta a CERO si ACU1 vale 0
        SPA UNO       //se salta a UNO si ACU1 vale 1
        SPA DOS       //se salta a UNO si ACU1 vale 2

DEF:    ...
        ...

        BEA
CERO:   ...
        ...
        BEA
UNO:    ...
        ...
        BEA
DOS:    ...
        ...
        BEA
```

Ejemplo de salto SPL:

```
        L   MB100       //cargamos en el ACU1 un valor de un módulo de datos
        SPL NORM        //se salta a NORM si el valor de ACU1 no está en lista
        SPA UNO         //se salta a UNO si ACU1 vale 0
        SPA CONT        //se salta a CONT si ACU1 vale 1
        SPA DOS         //se salta a DOS si ACU1 vale 2
NORM:   SPA CONT        //se salta a CONT de forma incondicional
 UNO:   U   E  0.0      //instrucción meta del salto UNO
        SPA CONT        //se salta a CONT de forma incondicional
 DOS:   U   E  1.0      //instrucción meta del salto DOS
        SPA CONT        //se salta a CONT de forma incondicional
CONT:   =   A  2.0      //aquí saltamos finalmente, continuando el programa
```

9.2. Operaciones de salto condicional, en función del RLO

Instrucciones: "SPB", "SPBN", "SPBB", "SPBNB"

Estas instrucciones efectúan un salto en el programa hacia una meta determinada, para el caso de cumplir la condición que necesitan:

SPB: salto si RLO=1
SPBN: salto si RLO=0
SPBB: salto si RLO=1 y RB=1
SPBNB: salto si RLO=0 y RB=1

En todas estas instrucciones, si la condición no es cumplida y no se realiza el salto, se modifican los siguientes registros:

RO=0
STA=1
RLO=1
ER=0

En SPBB y SPBNB se almacena el RLO en el bit RB de la palabra de estado antes de efectuar el salto.

Registros afectados: RB, OR, STA, RLO, ER

Ejemplo de salto SPB:

```
        U   E  2.0      //cargamos en el RLO el valor de la entrada 2.0
        SPB AQUI        //saltamos a la línea con meta "AQUI" si el RLO=1
        U   E  1.0      //esta línea no se ejecuta si se salta
AQUI:   U   E  3.0      //aquí continua la ejecución del programa
        =   A  0.0      //introducimos el resultado en la salida 0.0
```

Como podemos observar en el ejemplo, el resultado de la salida 0.0 depende primeramente del valor de la entrada 2.0, ya que ella decide si se tiene en cuenta también la entrada 1.0 en el resultado final.

9.3. Operaciones de salto condicional, en función de RB u OV/OS

Instrucciones: "SPBI", "SPBIN", "SPO", "SPS"

Estas instrucciones efectúan un salto en el programa hacia una meta determinada, para el caso de cumplir la condición que necesitan:

SPBI: salto si RB=1
SPBIN: salto si RB=0
SPO: salto si OV=1

SPS: salto si OS=1

Las operaciones SPBI y SPBIN ponen los bits OR y ER de la palabra de estado a 0 y el bit STA a 1. La operación SPS pone el bit OS a 0.

Registros afectados: OR, ER, STA, OS

Ejemplo de salto SPS:

```
        SPS AQUI        //saltamos a la línea con meta "AQUI" si OV=1
        SPA SEGU        //esta línea no se ejecuta si OV=1
AQUI:   SET             //forzamos el RLO a 1
        =   A  1.0      //con la salida 1.0 indicamos si hubo un error previo
                          en la anterior ejecución del programa
SEGU:   U   E  3.0      //aquí continua la ejecución del programa normalmente
        =   A  0.0      //introducimos el resultado en la salida 0.0
```

9.4. Operaciones de salto condicional, en función de A1 y A0

Instrucciones: "SPZ", "SPN", "SPP", "SPM", "SPMZ", "SPPZ", "SPU"

Estas instrucciones efectúan un salto en el programa hacia una meta determinada, para el caso de cumplir la condición que necesitan:

SPZ: salto si resultado=0 (ACU 1)
SPN: salto si resultado no es 0
SPP: salto si resultado es mayor que cero
SPM: salto si resultado es menor que cero
SPMZ: salto si resultado es menor o igual que cero
SPPZ: salto si resultado es mayor o igual que cero
SPU: salto si el resultado no es válido (uno de los operandos en una operación de coma flotante no es un número en coma flotante)

A continuación se muestra el estado de A1 y A0 tras una operación con los acumuladores:

A1	**A0**	**Resultado del cálculo**	**Operación de salto posible**
0	0	igual a 0	SPZ
1 o 0	0 o 1	distinto de 0	SPN
1	0	mayor que 0	SPP
0	1	menor que 0	SPM
0 o 1	0 o 0	mayor o igual que 0	SPPZ
0 o 0	0 o 1	menor o igual que 0	SPMZ
1	1	UO (no admisible)	SPU

9.5. Finalizar módulos

Instrucciones: "BEA" y "BEB"

Durante el ciclo del autómata programable, el sistema operativo ejecuta un programa estructurado módulo a módulo. La operación fin de módulo es la que finaliza el módulo en ejecución.

BEA finaliza la ejecución del módulo actual y devuelve el control al módulo que llamó al módulo finalizado. Esta instrucción se ejecuta sin depender del RLO ni de cualquier otra condición.

Ejemplo:

```
         U   E 1.0      //introducimos en el RLO el valor de la entrada 1.0
         SPB NEXT       //si la entrada 1.0 salta a meta NEXT
         L   EW 4
         T   EW 10
         U   E 6.0
         U   E 6.1
         S   M 12.0
         BEA            //aquí finaliza el módulo de forma incondicional
NEXT:    NOP 0
```

Ejemplo 2 :

```
         U   E 1.0      //introducimos en el RLO el valor de la entrada 1.0
         SPB NEXT       //si la entrada 1.0 salta a meta NEXT
         L   EW 4
         T   EW 10
         U   E 6.0
         U   E 6.1
         S   M 12.0
         BEA            //aquí finaliza el módulo de forma incondicional
NEXT:    NOP 0
         BEA            //fin de módulo
         U   E 1.1      //estas instrucciones nunca se ejecutarían
         S   A 4.0
```

BEB finaliza la ejecución del módulo actual y devuelve el control al módulo que llamó al módulo finalizado. Esta acción se realiza si el RLO es 1. Si no es así se continua la ejecución del actual módulo, pero con el RLO a 1.

Ejemplo:

```
U   E   1.0      //introducimos en el RLO el valor de la entrada 1.0
BEB              //si la entrada 1.0 vale 1 el módulo acaba aquí
U   E   2.0
=   A   3.0
BEA              //aquí finaliza el módulo de forma incondicional
```

Si el módulo que finaliza es el OB1 se finaliza el ciclo de ejecución del programa, volviendo a comenzar uno nuevo.

9.6. Loop

Instrucción: "LOOP"

La operación LOOP sirve para llamar varias veces un segmento del programa. Esta operación decrementa la palabra baja del ACU 1 en 1. Después se comprueba el valor depositado en la palabra baja del ACU 1. Si no es igual a 0, se ejecuta un salto a la meta indicada en la operación LOOP. En caso contrario, se ejecuta la siguiente operación normalmente.

Observaciones :

- El contador de bucles es un entero (de 16 bits) sin signo
- El salto puede ser tanto hacia delante como hacia atrás
- El salto sólo se puede ejecutar dentro de un bloque

Registros afectados: ACU 1

Ejemplo:

```
        L      +5      //Hacemos el ACU 1 igual a 5
PROX:   T   MW 10      //transferimos el valor del ACU 1 a la memoria de datos
        -              //En estos guiones estaría el segmento del programa
        -              //que se va a ejecutar 5 veces
        -
        L   MW 10      //leemos el valor de la memoria de datos en ACU 1
        LOOP   PROX    //decrementamos ACU 1 y saltamos a PROX si no es cero
```

Hay que tener precaución con el valor que haya en el ACU 1, ya que si ejecutamos LOOP con un valor de ACU 1 igual a 0 el bucle se ejecutará 65535 veces. Tampoco se recomienda introducir valores enteros negativos en el ACU 1.

9.7. Ejercicios propuestos

Ejercicio 1: SPL

Según el contenido de EB0 :

Si 0 => AW4= FFFF

Si 1 => AW4=AAAA

Si 2 => AW4=5555

Si 3 => AW4=0000

Ejercicio 1: Factorial de un número

Hacer el factorial del nº 7, o sea, 7 x 6 x 5 x 4 x 3 x 2 x 1 mediante un bucle.

10. Operaciones de control de programa

10.1. Llamar funciones y módulos de función con CALL

Instrucción: "CALL"

La operación CALL se emplea para llamar funciones (FC's) y módulos de función (FB's) creados para el usuario para el programa en cuestión o adquiridos en Siemens como módulos de función estándar. La operación CALL llama la función FC o módulo FB indicado como operando, independientemente del resultado lógico o cualquier otra condición.

Si se desea llamar un módulo de función con la operación CALL, se deberá asignar un módulo de datos de instancia (DB de instancia).

La llamada de una función (FC) o de un módulo de función (FB) puede programarse, es decir, es posible asignar operandos a la llamada. El programa ejecutará con estos operandos la función (FC) o el módulo de función (FB). Para ello hay que indicar los operandos que se desean usar para ejecutar la función o el módulo de función. Estos parámetros se denominan parámetros actuales (entradas, salidas, marcas de memoria...). El programa que contiene la función o el módulo de función tiene que poder acceder a estos parámetros actuales, por lo que se deberá indicar en el programa el parámetro formal que corresponda al parámetro actual . Si no se especifica la correspondencia en módulos de función el programa accederá a través del módulo de datos de instancia a los datos del parámetro formal. En la llamada a funciones todos los parámetros formales tienen que ser asignados a parámetros actuales.

La lista de parámetros formales es parte integrante de la operación CALL. El parámetro actual que se indica al llamar un módulo de función tiene que ser del mismo tipo de datos que el parámetro formal.

Los parámetros actuales empleados al llamar una función o un módulo de función se suelen indicar con nombres simbólicos. El direccionamiento absoluto de parámetros actuales sólo es posible con operandos cuyo tamaño máximo no supere una palabra doble.

Registros afectados: ninguno

Ejemplo de llamada a un FB con un DB de instancia y parámetros de módulo:

```
CALL   FB40,DB41     //llamamos al módulo FB40 con el módulo de instancia DB41
ON1:   = E1.0        //ON1 (parámetro formal) es asignado a E1.0 (p. actual)
ON2:   = MW2         //ON2 (parámetro formal) es asignado a MW2 (p. actual)
OFF1:  = MD20        //OFF1 (parámetro formal) es asignado a MD20 (p. actual)
L      DB20          //el programa accede al parámetro formal OFF1.
```

En el ejemplo anterior se ha supuesto que los parámetros formales pertenecen a los siguientes tipos de datos:

ON1: BOOL (binario)
ON2: WORD (palabra)
OFF1: DWORD (palabra doble)

Ejemplo de llamada a un FC con parámetros de módulo:

```
CALL   FC80           //llamamos la función FC80
INK1:  = M1.0         //INK1 (p. formal) es asignado a M 1.0 (p. actual)
INK2:  = EW2          //INK2 (p. formal) es asignado a EW2 (p. actual)
OFF:   = AW4          //OFF (p. formal) es asignado a AW4 (p. actual)
```

En el ejemplo anterior se ha supuesto que los parámetros formales pertenecen a los siguientes tipos de datos:

INK1: BOOL (binario)
INK2: INT (entero)
OFF: WORD (palabra)

Es posible crear una función que dé un valor de retorno. Si se desea crear por ejemplo una operación aritmética con números de coma flotante, entonces puede utilizar este valor de retorno como salida para el resultado de la función. Como nombre de la variable puede introducirse "RE_VAL" y como tipo de datos REAL. Al llamar después esta función en el programa se ha de proveer la salida RET_VAL de una dirección de palabra doble de forma que pueda acoger el resultado de 32 bits de la operación aritmética.

10.2. Llamar funciones y módulos con CC y UC

Instrucciones: "CC" y "UC"

Estas operaciones se emplean para llamar funciones (FC) creadas para el programa del mismo modo como se utiliza la operación CALL. Sin embargo, no es posible transferir parámetros.

CC llama la función indicada como operando si el RLO=1.

UC llama la función indicada como operando, independientemente de cualquier condición.

Las operaciones CC y UC pueden llamar una función con direccionamiento directo o indirecto de la memoria, o a través de una FC transferida como parámetro. El área de memoria es FC más el número del FC.

Máx. área de direccionamiento directo	**Máx. área de direccionamiento indirecto**		
0 a 65535	[DBW] [DIW] [LW] [MW]	0 a 65534	

El nombre del parámetro formal o nombre simbólico para el caso de llamar una FC a través de una FC transferida como parámetro es BLOCK_FC (los parámetros de tipo BLOCK_FC no pueden utilizarse con la operación CC en módulos FC).

Registros afectados: ninguno

Ejemplos:

CC FB12 //llamar a FB12 si RLO=1
UC FB12 //llamar a FB12 independientemente del RLO

10.3. Llamar funciones de sistema integradas

Instrucción: "CALL"

La operación CALL puede emplearse también para llamar funciones del sistema (SFC) y módulos de función del sistema (SFB) integrados en el sistema operativo S7.

Cada SFB o SFC ejecuta una función estándar determinada. Por ejemplo, si se desea averiguar al hora actual del sistema se utiliza al siguiente operación:

CALL SFC64

La operación de llamada CALL solamente puede llamar una SFC o un SFB con direccionamiento directo.

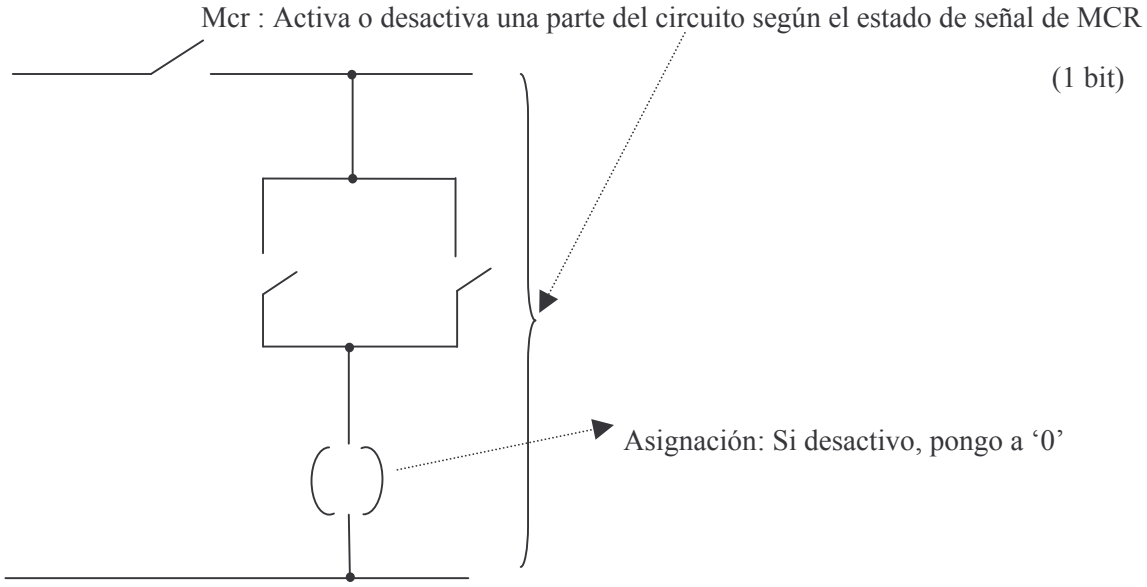
10.4. Función Master Control Relay

Instrucciones: "MCRA", "MCRD", "MCR(:", ")MCR:"

El Master Control Relay (MCR) se emplea para inhibir el funcionamiento de una determinada parte del programa (secuencia de instrucciones que escribe un cero en lugar del valor calculado, o bien no modifican el valor de memoria existente). Las siguientes operaciones dependen del MCR:

=
S
R
T (con bytes, palabras o palabras dobles)

Se utiliza en los esquemas de relés para activar y desactivar el flujo de señales.



- Si el bit de MCR es igual a 1 “ON” --> Ejecución normal
- Si bit MCR=0 “Desactivación”, el MCR afecta a las siguientes instrucciones:
    - Asignación “=” : Escribe 0.
    - Set y Reset “S y R” : No escribe. Permanecen en su estado actual.
    - Transferencia “T” de bytes, palabras y doble palabras: Escribe 0.

Ejemplo:

```
MCRA        //Activa el área MCR
U   E0.0    //Consulto
MCR(        //Inicia el área MCR : Si RLO=1 --> MCR ON
                                   Si RLO=0 --> MCR OFF

U   E0.1    //Si MCR OFF --> A4.0=0 sin considerar E0.1
=   A4.0

U   E0.2    //Si MCR OFF --> Mantiene el valor de A4.1
S   A4.1
U   E0.3
R   A4.1

L   MW20   //Si MCR OFF --> Transfiero 0 a AW10
T   AW10

)MCR       //Finalizo área MCR
MCRD       //Desactivo área MCR


U   E1.0   //Como está fuera del área MCR no depende del bir MCR
=   A5.0
```

11. Formatos de representación de números

Tenemos 3 formatos de representación numérica: Binario, Hexadecimal y BCD

Tenemos 3 formatos de números: Entero, doble entero y real.

11.1. Binario

Representa números naturales mayores o iguales que cero.

0; 01; 10; 11; 100...

Límite 32 bits => $2^{32} - 1 = 4.294.967.295$

Ejemplo:

```
L   2#0110      //Cargo en ACU1 el número 7 en formato binario
```

Rango:

Palabra: 2#0 ÷ 2#1....1 (16 unos)

Doble palabra: 2#0 ÷ 2#1....1 (32 unos)

11.2. Hexadecimal

Ejemplo:

```
L   B#16#1A          //Cargo una constante hexadecimal de 8 bits en ACU1
L   W#16#FAFB        //Cargo una constante hexadecimal de 16 bits en ACU1
L   DW#16#1FFE_1ABC   //Cargo una constante hexadecimal de 32 bits en ACU1
```

Rango:

Byte: B#16#0 ÷ B#16#FF

Palabra: W#16#0 ÷ W#16#FFFF

Doble palabra: DW#16#0 ÷ DW#16#FFFFFFFF

11.3. BCD

BCD "Decimal codificado en binario". Cada 4 bits representan un dígito.

Hay 2 formatos de BCD, de 3 cifras y de siete cifras.

Palabra (16 bits): BCD de 3 cifras con signo

Ejemplo: +310

0000	0011	0001	0000
+	3	1	0

signo: 0000 -> positivo
1111 -> negativo

Rango: -999 ÷ 999

Doble palabra (32 bits): 7 cifras con signo.

Rango: Lo que cabe en ACU1: 9999999

Ejemplo:

```
L   2#0000_0011_0001_000  //Cargo +310 en BCD formato en ACU1
LC  T1                    //Cargo el valor de temporización de T1 en formato
                            BCD en ACU1
LC  Z1                    // Cargo el valor del contador Z1 en formato
                             BCD en ACU1
```

11.4. Números enteros (I)

Los números enteros se denominan como I (de *Integer*).

Un número entero es un número binario de 16 bits que tiene como signo el bit más significativo.

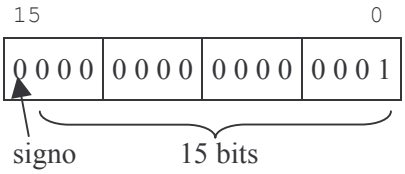
Límite:

Nº positivos: 2^{15}-1 = 32767 (El 0 se considera positivo)

Nº negativos: 2^{15}= -32768

Ejemplo:

```
L   1    //Carga el número entero 1 en ACU1
```



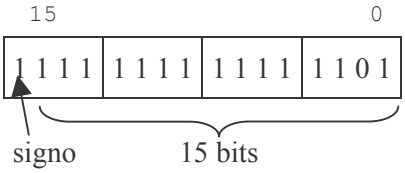
Nº negativo: Se hace el complemento a 2, cambiando ceros por unos y sumando 1.

Ejemplo:

```
L   -3   //Carga el número entero 3 en ACU1
```

3 = 0000 0000 0011
C'1= 1111 1111 1100
+1 = 1111 1111 1101

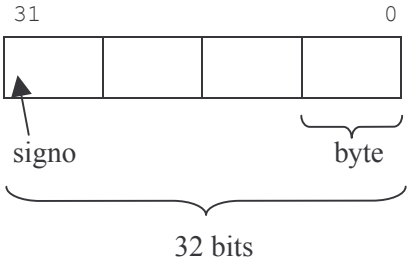
El número –3 por tanto sería:



11.5. Números dobles enteros (D)

Los números dobles enteros se denominan como D.

Son números binarios de 32 bits.



Límite:

Nº positivos: 2^{31}-1 = 2147483647 (El 0 se considera positivo)

Nº negativos: 2^{31} = -21474863648

Ejemplo:

```
L   L#-1 //Carga el número doble entero -1 en ACU1
```

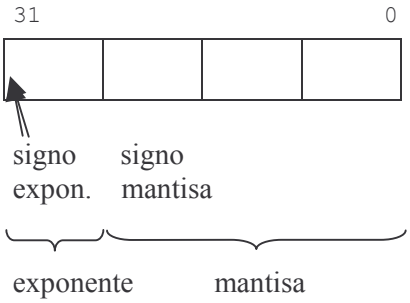
11.6. Números reales (R)

Los números reales se denominan como R.

Son números binarios de 32 bits que constan de 2 partes:

Mantisa : los 3 bytes más altos

Exponente : el byte más alto



Se puede expresar de forma exponencial o como quebrados.

Ejemplo:

```
L    4.83    //Carga el número real 4,83 en ACU1
```

El editor Step-7 lo pasa automáticamente a la forma exponencial:

```
L  4.830000e+000
```

Se redondea con exactitud hasta el 6 dígito

Ejemplo:

```
L    4780000000.0    =  L    4.780000e+010
```

el punto es obligatorio

12. Operaciones de comparación

12.1. Realización de comparaciones

Las operaciones de comparación sirven para comparar los siguientes pares de valores numéricos:

Dos enteros (16 bits)
Dos enteros dobles (32 bits)
Dos números reales (de coma flotante, 32 bits, IEEE-FP)

Los valores numéricos se cargan en los ACU's 1 y 2. Las operaciones de comparación comparan el valor del ACU2 con el valor depositado en el ACU1.

El resultado de la comparación es un dígito binario. Un 1 significa que el resultado de la comparación es verdadero, mientras que un 0 significa que el resultado de la comparación es faso. Este resultado se encuentra almacenado en el bit de resultado lógico (RLO). Este resultado puede emplearse para su posterior procesamiento.

Cuando se ejecuta una comparación también se activan los bits de estado A1 y A0.

12.2. Comparar dos números enteros

Instrucciones y descripción:

Operación	Comparación efectuada
==I	El entero (16 bits) de la palabra baja del ACU2 es igual al entero (16 bits) de la palabra baja del ACU 1.
==D	El entero doble (32 bits) del ACU2 es igual al entero doble (32 bits) del ACU1.
<>I	El entero (16 bits) de la palabra baja del ACU2 no es igual al entero (16 bits) de la palabra baja del ACU 1.
<>D	El entero doble (32 bits) del ACU2 no es igual al entero doble (32 bits) del ACU1.
>I	El entero (16 bits) de la palabra baja del ACU2 es mayor que el entero (16 bits) de la palabra baja del ACU 1.
>D	El entero doble (32 bits) del ACU2 es mayor que el entero doble (32 bits) del ACU1.
<I	El entero (16 bits) de la palabra baja del ACU2 es menor que el entero (16 bits) de la palabra baja del ACU 1.
<D	El entero doble (32 bits) del ACU2 es menor que el entero doble (32 bits) del ACU1.
>=I	El entero (16 bits) de la palabra baja del ACU2 es mayor o igual al entero (16 bits) de la palabra baja del ACU 1.

>=D	El entero doble (32 bits) del ACU2 es mayor o igual al entero doble (32 bits) del ACU1.
<=I	El entero (16 bits) de la palabra baja del ACU2 es menor o igual al entero (16 bits) de la palabra baja del ACU 1.
<=D	El entero doble (32 bits) del ACU2 es menor o igual al entero doble (32 bits) del ACU1.

Estas operaciones afectan al estado de los bits A1 y A0, en función de la condición que se haya cumplido, tal y como se muestra en la tabla siguiente:

Condición	A1	A0
ACU2 > ACU1	1	0
ACU2 < ACU1	0	1
ACU2 = ACU1	0	0
ACU2 <> ACU1	0 o 1	1 o 0
ACU2 >= ACU1	1 o 0	0 o 0
ACU2 <= ACU1	0 o 0	1 o 0

Registros afectados: RLO, A1, A0

Ejemplo:

```
L    MW10          //introducimos en el ACU1 la palabra de marcas MW10
L    EW0           //introducimos en el ACU1 la palabra de entradas EW0, el
                     antiguo contenido del ACU1 (MW10) pasa al ACU2
==I                //comparamos la igualdad de la palabra baja del ACU1 y ACU2
=    A    1.0      //la salida 1.0 se excitará si MW10 y EW0 son iguales
>I                 //comparamos el valor de la palabra baja del ACU2 para
                     ver si es mayor que el valor de la palabra baja del ACU1
=    A    2.0      //la salida 2.0 se excitará si MW10 es mayor que EW0
```

12.3. Comparar dos números reales

Instrucciones y descripción:

Operación	Comparación efectuada
==R	El número de coma flotante de 32 bits IEEE-FP del ACU 2 es igual al número de coma flotante de 32 bits IEEE-FP del ACU 1
<>R	El número de coma flotante de 32 bits IEEE-FP del ACU 2 no es igual al número de coma flotante de 32 bits IEEE-FP del ACU 1

>R	El número de coma flotante de 32 bits IEEE-FP del ACU 2 es mayor que el número de coma flotante de 32 bits IEEE-FP del ACU 1
<R	El número de coma flotante de 32 bits IEEE-FP del ACU 2 es menor que el número de coma flotante de 32 bits IEEE-FP del ACU 1
>=R	El número de coma flotante de 32 bits IEEE-FP del ACU 2 es mayor o igual que el número de coma flotante de 32 bits IEEE-FP del ACU 1
<=R	El número de coma flotante de 32 bits IEEE-FP del ACU 2 es menor o igual que el número de coma flotante de 32 bits IEEE-FP del ACU 1

Estas operaciones activan determinadas combinaciones de los estados de los códigos de condición A1, A0, OV y OS de la palabra de estado para indicar qué condición se ha cumplido, tal y como se muestra en la siguiente tabla:

Condición	A1	A0	OV	OS
==	0	0	0	no aplicable
<>	0 o 1	1 o 0	0	no aplicable
>	1	0	0	no aplicable
<	0	1	0	no aplicable
>=	1 o 0	0 o 0	0	no aplicable
<=	0 o 0	1 o 0	0	no aplicable
UO	1	1	1	1

Registros afectados: RLO, A1, A0, OV, OS

Ejemplo:

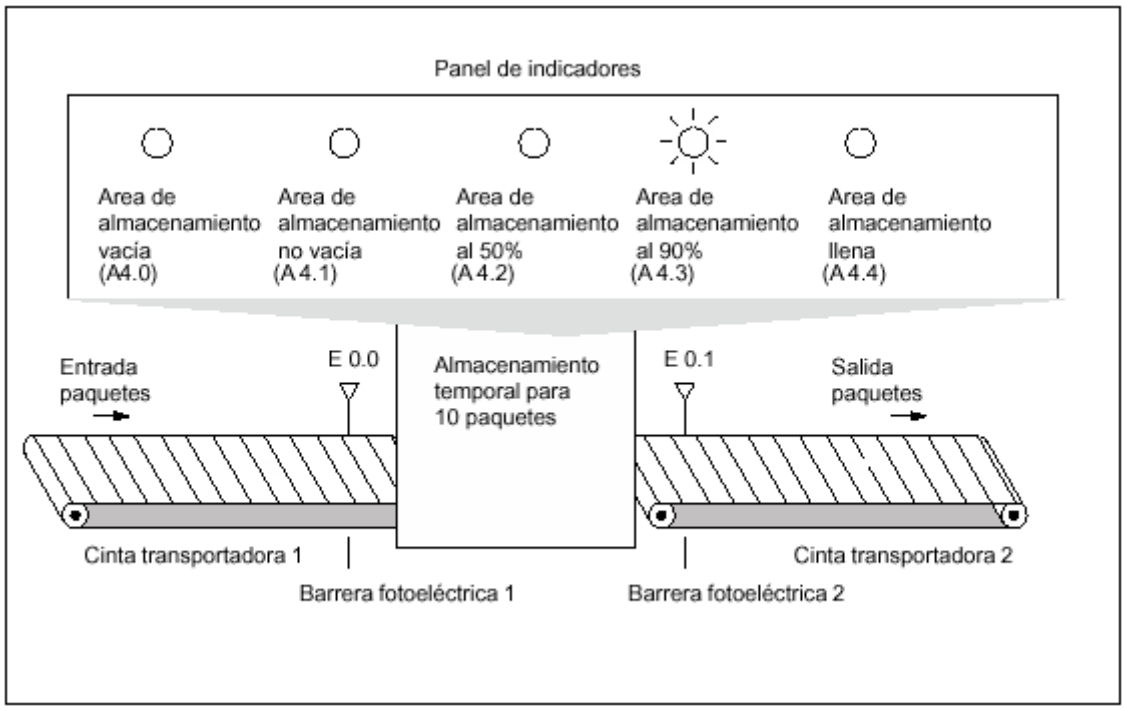
```
L   MD24         //introducimos en el ACU1 la palabra doble de marcas MD24
L   +1.00E+00    //introducimos en el ACU1 un valor en coma flotante de 32
                   bits, el antiguo contenido del ACU1 (MD24) pasa al ACU2
>R               //comparamos el ACU2 para ver si es mayor que el ACU1
=   A   1.0      //la salida 1.0 se excitará si la condición es cierta
```

12.4. Ejercicios propuestos

Ejercicio 1 : Área de almacenamiento

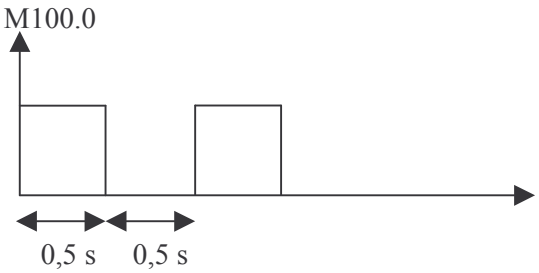
La figura muestra un sistema de dos cintas transportadoras con un área de almacenamiento temporal entre ellas. La cinta 1 lleva paquetes al área de almacenamiento. Una barrera fotoeléctrica situada al final de la cinta 1, junto al área de almacenamiento, se encarga de determinar cuántos paquetes se han suministrado al área de almacenamiento. La cinta 2 transporta paquetes desde el área de almacenamiento temporal a un cargador donde llegan camiones para recoger los paquetes y suministrarlos a los clientes. Una barrera fotoeléctrica al final de la cinta transportadora 2, junto al área de almacenamiento, determina cuántos paquetes salen del área hacia el cargador.

Un panel de cinco lámparas indica el nivel de llenado del área de almacenamiento temporal.



13. Marca de ciclo

Una marca de ciclo es una marca que cambia periódicamente de valor binario.



Se define en las propiedades de la CPU: HW Config → CPU → Pestaña "Ciclo/Marca de ciclo"

Para crear una marca de ciclo :

1. Introducir la dirección del byte de marcas.
2. Activar la casilla de verificación ✓
3. "Guardar y compilar" y "Cargar en módulo"

Cada bit del byte de marca de ciclo lleva asignada una duración de periodo / frecuencia en seg:

Bit:	7	6	5	4	3	2	1	0
Duración del período (seg):	2	1,6	1	0,8	0,5	0,4	0,2	0,1

Ejemplo:

```
U M100.3      //Si defino la marca 100 como marca de ciclo
= A4.0        //el led parpadeará con una frecuencia de 0,5 seg.
```

13.1. Ejercicios propuestos

Ejercicio : Marca de ciclo

Mediante interruptores realizar:

Entradas	Salidas
E0.0	A4.0 parpadee 2 seg.
E0.1	A4.1 parpadee 0,5 seg.
E0.2	A4.0 parpadee 1,6 seg.
	A4.1 parpadee 0,8 seg.

14. Operaciones aritméticas

14.1. Operaciones aritméticas con enteros

Para enteros instrucciones: "+I", "–I", "*I", "/I"

Para dobles enteros instrucciones: "+D", "–D", "*D", "/D"

- Combinan el contenido de ACU1 y ACU2.
- El resultado se deposita en ACU1 y el ACU2 permanece inalterado.
- En el S7-400 después de la operación: ACU2 ← ACU3 ← ACU4
- Son operaciones incondicionales del RLO y no lo afectan.

La instrucción de suma "+I", "+D" suma el contenido de ACU2 con ACU1 y el resultado lo almacena en ACU1.

En el caso de sumar dos enteros esta operación actúa sobre las palabras bajas de los acumuladores y el resultado lo almacena en la palabra baja del ACU1.

Ejemplo:

```
L   8
L   9
+I      //ACU2+ACU1 → ACU1=17; ACU2=8
```

La instrucción de resta "–I", "–D" resta el contenido de ACU2 con ACU1 y el resultado lo almacena en ACU1.

Ejemplo:

```
L   10
L   3
-I      //ACU2-ACU1 → ACU1=7; ACU2=10
```

La instrucción de multiplicación "*I", "*D" multiplica el contenido de ACU2 por ACU1 y el resultado lo almacena en ACU1.

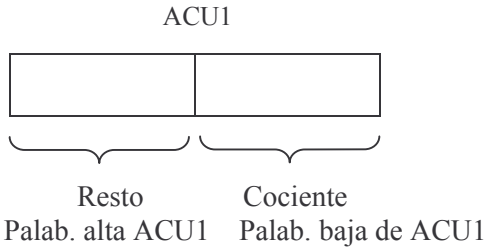
Si se multiplican dos enteros el resultado se almacena como doble entero en ACU1.

Ejemplo:

```
L   L#2
L   L#100
*D          //ACU2*ACU1 → ACU1=200; ACU2=2
```

La instrucción de división "/I", "/D" divide el contenido de ACU2 entre ACU1 y el resultado se almacena de dos formas distintas según se multipliquen enteros y dobles enteros:

- En caso de división de enteros el resultado se almacena en ACU1 en forma de dos enteros de 16 bits:



- En caso de dividir dos enteros dobles el cociente se almacena en ACU1 y el resto de la división se obtiene realizando la operación "MOD".

Ejemplo:

```
L  ED10
L  MD14
/I
T  MD20  //Almaceno el cociente
MOD
T  MD24  //Almaceno el resto
```

14.2. Operaciones aritméticas con números reales

Instrucciones: "+R", "–R", "*R", "/R"

La instrucción de suma "+R", suma el contenido de ACU2 (32bits) con ACU1 (32bits) y el resultado lo almacena en ACU1 (32 bits).

Ejemplo:

```
L   1.0
L   2.4
+R      //ACU2+ACU1 → ACU1=3.4; ACU2=1.0
```

La instrucción de resta "–R" resta el contenido de ACU2 (32bits) con ACU1 (32bits) y el resultado lo almacena en ACU1 (32 bits).

Ejemplo:

```
L   25.0
L   13.4
-R        //ACU2-ACU1 → ACU1=11.6; ACU2=25.0
```

La instrucción de multiplicación "/R" divide el contenido de ACU2 (32bits) entre ACU1 (32bits) y el resultado lo almacena en ACU1 (32 bits).

Ejemplo:

```
L    10.0
L    2.0
/R          //ACU2*ACU1 → ACU1=5.0; ACU2=10.0
```

Instrucción "ABS": Valor absoluto de ACU1, y el resultado se deposita en ACU1.

Instrucción "SQRT": Raíz cuadrada de ACU1 (debe ser ≥ 0), y el resultado se deposita en ACU1.

Instrucción "SQR": Cuadrado de ACU1, y el resultado se deposita en ACU1.

Instrucción "LN": Logaritmo natural de ACU1 (logaritmo con base e), y el resultado se deposita en ACU1.

Instrucción "EXP": Calcula el exponente (valor exponencial con base e) de ACU1, y el resultado se deposita en ACU1.

Instrucciones "SIN", "COS" y "TAN": Calculan el seno, coseno o tangente de un ángulo indicado en radianes, y el resultado se deposita en ACU1.

Instrucción "ASIN" : Calcula el arcoseno de ACU1 (entre –1 y 1). El resultado es un ángulo indicado en radianes: $-\pi/2 \leq \text{arcoseno(ACU1)} \leq \pi/2$

Instrucción "ACOS" : Calcula el arcocoseno de ACU1 (entre –1 y 1). El resultado es un ángulo indicado en radianes: $0 \leq \text{arcocoseno(ACU1)} \leq \pi$

Instrucción "ATAN" : Calcula el arcotangente de ACU1. El resultado es un ángulo indicado en radianes: $\pi/2 \leq \text{arcotangente(ACU1)} \leq \pi/2$

14.3. Ejercicios propuestos

Ejercicio : Multiplicación y división

Con la entrada E0.5 cargo el valor de EB1 en ACU1.
A cada pulso de la entrada E0.0 iré multiplicando ese valor por 2 hasta un máximo de 255.
Además, a cada pulso de la entrada E0.1 podré ir dividiendo ese valor entre 2 hasta un mínimo de 1.

15. Operaciones de conversión

Instrucciones: "BTD", "BTI", "DTB", "DTR","DTR","ITB","ITD","RND","RND+","RND-" y "TRUNC"

- Son operaciones incondicionales, no dependen ni afectan el RLO.
- Se efectúan sobre el ACU1, donde también se almacena el resultado.

Instrucción “BTD” : Convierte el número BCD de 7 dígitos en doble entero.

Valores admisibles: -9999999 ÷ 9999999

Resultado de la conversión:

Bits 0 ÷ 27 → Núm convertido
Bits 28 ÷ 30 → No se usan
Bit 31 → Bit de signo

Instrucción “BTI” : Convierte el número BCD de 3 dígitos de la palabra baja de ACU1 en un entero (16 bits) que se guarda en la palabra baja de ACU1.

Valores admisibles: -999 ÷ 999

Resultado de la conversión:

Bits 0 ÷ 11 → Núm convertido
Bits 12 ÷ 14 → No se usan
Bit 15 → Bit de signo

Instrucción “DTB” : Convierte un entero doble (32 bits) de ACU1 en un BCD de 7 dígitos.

Resultado de la conversión:

Bits 0 ÷ 27 → Nº BCD
Bits 28 ÷ 31 → Bit de signo: + = 0000
– = 1111

Instrucción “DTR” : Convierte un entero doble (32 bits) de ACU1 en real (32 bits). Si es necesario se redondea el resultado. Un entero de 32 bits es más exacto que un real.

Instrucción “ITB” : Convierte un entero (16 bits) de ACU1 en BCD de 3 dígitos. El resultado lo guarda en la palabra baja del ACU1.

Resultado de la conversión:

Bits 0 ÷ 11 → Nº BCD
Bits 12 ÷ 15 → Bit de signo: + = 0000
– = 1111

Instrucción “ITD” : Convierte un entero (16 bits) de la palabra baja de ACU1 en doble entero (32 bits).

Observación: No hay una instrucción que convierta de entero a real. Hay que hacer dos conversiones consecutivas: ITD y DTR.

Instrucción “RND” : Redondea un número real (32 bits) a entero (32 bits). Primero lo convierte y luego lo redondea al entero más próximo. Si el primer decimal después de la coma es de 0 a 4, redondea a la baja; si el decimal es de 5 a 9, redondea al alza.

Ejemplos: 148.7 → RND = 149
148.4 → RND = 148

Instrucción “RND+” y “RND–“: RND+ redondea el resultado al número entero mayor o igual que el real convertido. RND+ redondea el resultado al número entero menor o igual que el real convertido.

Ejemplos: 148.7 → RND+ = 149
RND– = 148

148.4 → RND+ = 149
RND– = 148

–5.7 → RND+ = –5
RND– = –6

Instrucción “TRUNC” : Convierte un real (32 bits) a doble entero (32 bits) y lo redondea al entero de menor valor absoluto.

Ejemplos: 8.34 → RND = 8
TRUNC = 8

8.7 → RND = 9
TRUNC = 8

–8.7 → RND = –9
TRUNC = –8

16. Operaciones de desplazamiento

Instrucciones: "SRW", "SLW", "SRD", "SLD", "SSI", "SSD"

Desplazan el contenido de ACU1 bit a bit. Son incondicionales, no dependen del RLO.

Podemos desplazar, tanto a la derecha como a la izquierda, palabras, doble palabras, enteros y dobles enteros.

16.1. Desplazar palabras

A la derecha:

"SRW <nº de bits>" Desplaza la palabra baja (bits 0 al 15) de ACU1 nº bits (0-15) a la derecha. Los bits vacíos se rellenan con cero.
Nota: SWR 0 = NOP

"SRW" Desplaza a la dcha. Tantas posiciones según el nº de el byte más bajo de ACU2. (0-255)

A la izquierda:

"SLW <nº de bits>" Desplaza la palabra baja (bits 0 al 15) de ACU1 nº bits (0-15) a la izquierda. Los bits vacíos se rellenan con cero.

"SLW" Desplaza a la izda. según el nº de el byte más bajo de ACU2.

16.2. Desplazar doble palabras

A la derecha:

"SRD <nº de bits>" Desplaza el contenido de ACU1 nº bits (0-32) a la derecha. Los bits vacíos se rellenan con cero.

"SRD" Desplaza a la dcha. según el nº de el byte más bajo de ACU2.

A la izquierda:

"SLD <nº de bits>" Desplaza el contenido de ACU1 nº bits (0-32) a la izquierda. Los bits vacíos se rellenan con cero.

"SLD" Desplaza a la dcha. según el nº de el byte más bajo de ACU2.

16.3. Desplazar enteros

A la derecha:

"SSI <nº de bits>" Desplaza la palabra baja (bits 0 al 15) de ACU1 nº bits (0-15) a la derecha. Los bits vacíos se rellenan con el bit de signo (el bit 15) Nota: SSI 0 = NOP

"SSI" Desplaza a la dcha. según el nº de el byte más bajo de ACU2.

16.4. Desplazar dobles enteros

A la derecha:

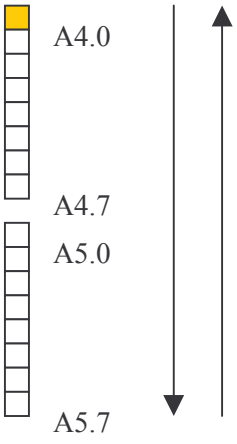
"SSD <nº de bits>" Desplaza el contenido de ACU1 nº bits (0-31) a la derecha. Los bits vacíos se rellenan con el bit de signo (el bit 32)
Nota: SSI 0 = NOP

"SSD" Desplaza a la dcha. según el nº de el byte más bajo de ACU2.

16.5. Ejercicios propuestos

Ejercicio 1 :

Haremos que los leds de la palabra de salida se iluminen consecutivamente durante 0,5 seg desde arriba hacia abajo y viceversa



17. Operaciones de rotación

Instrucciones: "RRD", "RLD"

Rotan el contenido de ACU1 equis posiciones de bit.

17.1. Rotar palabras dobles

A la derecha:

“RRD <nº de bits>” Rota el ACU1 nº bits (0-32) a la derecha. Los bits que van quedando vacíos se llenan con los que salen del ACU1.
Nota: RRD 0 = NOP 0

“RRD” Desplaza a la dcha. según el nº de el byte más bajo de ACU2

A la izquierda:

“RLD <nº de bits>” Rota el ACU1 nº bits (0-32) a la izquierda. Los bits que van quedando vacíos se llenan con los que salen del ACU1.
Nota: RRL 0 = NOP 0

“RLD” Desplaza a la dcha. según el nº de el byte más bajo de ACU2 Ejemplo:

ACU1

Después de RLD 4:

ACU1

17.2. Ejercicios propuestos

Ejercicio 1 :

Hacer lo siguiente:

Pulsador E0.1 : Carga el valor a rotar
Pulsador E0.0 : Rota un bit a la izquierda.

18. Bloques del programa de usuario

El software de programación STEP 7 permite estructurar el programa de usuario, es decir, subdividirlo en distintas partes. Esto aporta las siguientes ventajas:

- los programas de gran tamaño se pueden programar de forma clara
- se pueden estandarizar determinadas partes del programa
- se simplifica la organización del programa
- las modificaciones del programa pueden ejecutarse más fácilmente
- se simplifica el test del programa, ya que puede ejecutarse por partes
- se simplifica la puesta en servicio.

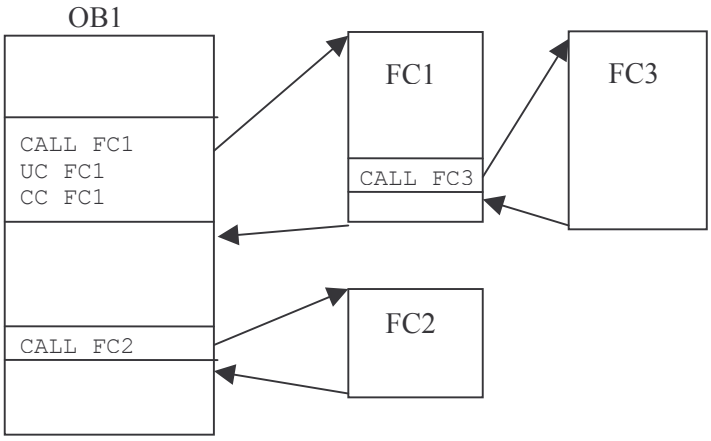
18.1. Tipos de bloques

En un programa de usuario S7 se pueden utilizar diversos tipos de bloques:

Bloque	Descripción breve de la función
Bloques de organización (OB)	Los OBs definen la estructura del programa de usuario.
Bloques de función del sistema (SFBs) y funciones de sistema (SFCs)	Los SFBs y SFCs están integrados en la CPU S7, permitiéndole acceder a importantes funciones del sistema.
Bloques de función (FB)	Los FBs son bloques con "memoria" que puede programar el mismo usuario.
Funciones (FC)	Las FCs contienen rutinas de programa para funciones frecuentes.
Bloques de datos de instancia (DBs de instancia)	Al llamarse a un FB/SFB, los DBs de instancia se asocian al bloque. Los DBs de instancia se generan automáticamente al efectuarse la compilación.
Bloques de datos (DB)	Los DBs son áreas de datos para almacenar los datos de usuario. Adicionalmente a los datos asociados a un determinado bloque de función, se pueden definir también datos globales a los que pueden acceder todos los bloques.

Los OBs, FBs, SFBs, FCs y SFCs contienen partes del programa, por lo que se denominan también bloques lógicos. El número permitido de bloques de cada tipo y su longitud admisible dependen de la CPU.

18.2. Módulos de función (FC)



Los módulos de función, también llamados FC, son módulos en los que podemos incluir parte del programa de usuario, así obtenemos un programa mucho más estructurado.

A estos módulos se puede acceder desde el OB1 o cualquier otro FC o FB.

En el FC podemos almacenar variables temporales “temp”, pero se pierden tras el tratamiento del FC. Por ello para memorizar los datos se suelen utilizar bloques de datos globales.

Las funciones se pueden utilizar para:

- devolver un valor de función al bloque invocante (ejemplo: funciones matemáticas)
- ejecutar una función tecnológica (ejemplo: control individual con combinación binaria).

Creación de un FC : Hacer clic con el botón izquierdo del ratón en la carpeta de Bloques e “Insertar nuevo objeto...” -> “Función”. Se le puede dar un nombre simbólico. Para abrirlo hacer doble clic sobre él.

Tiene una tabla de declaración de variables estándar.

18.3. Tabla de declaración de variables

Dirección	Declaración	Nombre	Tipo	Valor inicial	Comentario
0.0	in	valor1	WORD		
2.0	out	tiempo	S5TIME		
4.0	in_out	memoria	BOOL		
0.0	temp	fecha	DATE		

En las tabla de declaración de variables se definen las variables locales, que pueden ser temporales “temp” y estáticas “stat” (sólo FB’s) y los parámetros formales de bloque, que pueden ser de entrada “in”, de salida “out” o de entrada_salida “in_out”.

Parámetros:

- in: Son valores de entrada al bloque. Pueden ser constantes, direcciones absolutas o simbólicas.
- out: Son valores que devuelve el bloque al programa que lo llamó después de la ejecución del bloque. Pueden ser direcciones absolutas o simbólicas.
- in_out: Recogen y mandan información. . Pueden ser direcciones absolutas o simbólicas.

Variables:

- temp.: Variables o datos locales que se usan dentro del bloque y no salen fuera. Pierden su valor cuando salimos del bloque.
- stat: Variables que mantienen su valor al salir del módulo. Los FC no permiten este tipo de variables. Sólo para bloques FB.

18.4. Llamadas a bloques

Instrucciones: "CALL", "UC" y "CC"

"CALL" llama a funciones FC y FB independientemente del RLO. Una vez procesado el bloque, el programa invocante seguirá procesándose.

Se puede llamar al bloque de forma absoluta o simbólica.

Ejemplo:

```
CALL FC1     //Llama a la función FC1 de forma incondicional
```

Se pueden transferir parámetros a una función de la siguiente forma:

1. Crear en el FC una "Tabla de declaración de variables", con sus parámetros in, out e in_out según corresponda.
2. Cuando en el bloque invocante se introduzca la función CALL a ese FC, automáticamente se crea una "tabla de parámetros formales" que son cogidos de la tabla de declaración de variables del FC.

```
CALL FC1
     Num de motor:=   }
     Velocidad:=      }Parámetros formales
     Potencia:=       }
```

3. Hay que asignar a esos parámetros formales los parámetros actuales del bloque invocante.

```
CALL FC1
    Num de motor:= MW100 }
    Velocidad:= MW110    }Parámetros actuales
    Potencia:= A0.1      }
```

En el caso de los bloques FC no se puede dejar ningún parámetro actual sin definir.

La instrucción “UC” llama incondicionalmente al bloque. Es igual que la función CALL, pero no podemos transferir parámetros.

Ejemplo:

```
UC FC1      //Llama a la función FC1 de forma incondicional
```

La instrucción “CC” llama de forma condicional al bloque, en función del RLO. No podemos transferir parámetros.

Ejemplo:

```
U   E0.0
CC  FC1      //Llama a la función FC1 si E0.0=1
```

18.5. Ejemplo de función FC

Vamos a crear una función matemática que nos realice la suma de dos números enteros y nos devuelva el resultado.

1. Creamos la función FC1, al que podemos darle el nombre simbólico de “Suma”, y rellenamos su tabla de declaración de variables con dos parámetros de entrada que serán los dos operandos a sumar y un parámetro de salida que será el resultado de la suma:

Dirección	Declaración	Nombre	Tipo	Valor inicial	Comentario
0.0	in	valor1	INT		
2.0	in	valor2	INT		
4.0	out	resultado	INT		
	in_out				
	temp				

En el área de instrucciones del FC escribimos el programa en AWL:

```
L    #valor1
L    #valor2
+I
T    #resultado
```

El símbolo # significa variable temporal, se pone automáticamente.

Es importante guardar el bloque FC1.

2. Desde el OB1 llamaremos a la función. Al introducir la instrucción CALL automáticamente nos crea la tabla de parámetros formales que rellenaremos con los valores que queramos.

```
L    2
L    MW20
CALL    “Suma”
   valor1:= MW20
   valor2:= 3
   resultado:= MW100
```

Guardamos el OB1.

Para terminar hay que transferir al autómata los dos bloques, OB1 y FC1.

Marcando la carpeta “Bloques” y pinchando en el icono “Carga” transferimos a la CPU todos los bloques del susodicho programa.

Comprobar con “Observar/Forzar” el valor de MW100.

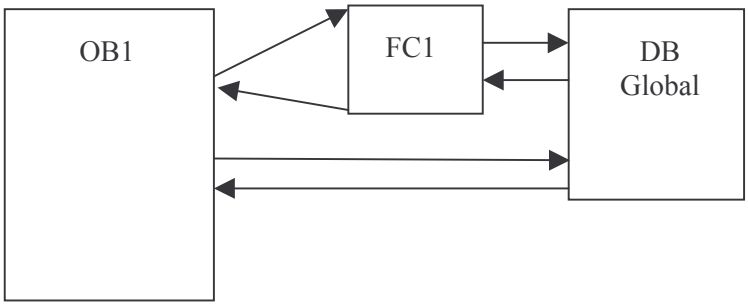
18.6. Bloques de datos (DB)

Los bloques de datos, también llamados DB's, son áreas de datos donde se almacenan datos de usuario. Un DB no contiene instrucciones S7.

Hay dos tipos de bloques de datos:

- Bloques de datos globales: A sus datos pueden acceder todos los bloques.
- Bloques de datos de instancia: Asociados a un bloque de función FB.

18.7. Bloque de datos global



Los DB Globales sirven para depositar datos con los que trabaja el programa de usuario. Estos datos los puede utilizar cualquier tipo de bloque (OB, FC y FB) para leer datos o escribir datos que se conservan aún después de salir del DB.

Un DB no tiene área de instrucciones S7, sino que es una tabla de datos que forma una estructura:

Dirección	Nombre	Tipo	Valor inicial	Comentario
0.0		STRUCT		
=0.0		END_STRUCT		

Los datos almacenados estarán comprendidos entre STRUCT y END_STRUCT.

Creación de un DB : Hacer clic con el botón izquierdo del ratón en la carpeta de Bloques e "Insertar nuevo objeto..." -> "Bloque de datos". Se le puede dar un nombre simbólico. Para abrirlo hacer doble clic sobre él.

1. Seleccione el bloque de datos global, es decir, un bloque no asociado a ningún UDT o FB.
2. Active la vista "Declaración" (en menú "Ver") del bloque de datos en caso de que ésta no se visualice todavía.
3. Defina la estructura rellenando la tabla visualizada conforme a los datos que se indican a continuación:

Columna	Explicación
Dirección	Indica la dirección que STEP 7 asigna automáticamente a la variable al terminar de introducir una declaración.
Designación	Introduzca el nombre que debe asignar a cada variable.
Tipo	Introduzca el tipo de datos de la variable (BOOL, INT, WORD, ARRAY, etc.). Las variables pueden pertenecer a los tipos de datos simples, a los tipos de datos compuestos, o bien, a los tipos de datos de usuario.
Valor inic.	Indique el valor inicial, en caso de que el software no deba tomar el valor predeterminado del tipo de datos introducido. Todos los valores introducidos deben ser compatibles con los tipos de datos. Cuando guarde por primera vez el bloque de datos, el valor inicial será adoptado como valor actual de la variable, a menos que defina expresamente su valor actual.
Comentario	En este campo puede introducir un comentario para documentar la variable. El comentario no debe tener más de 80 caracteres.

Acceso a los datos de un DB:

Hay 2 métodos:

1) Para acceder a los datos de un DB primero hay que abrirlo.

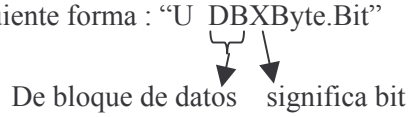
Instrucción: "AUF"

Ejemplo:

```
AUF  DB1     //Abre el DB1
AUF  Datos   //Abre el bloque de datos cuyo simbólico es "Datos"
```

Una vez abierto el DB el acceso a los datos se hace mediante carga (L) y transferencia (T)

Se puede consultar un bit de la siguiente forma : "U DBXByte.Bit"



Ejemplo:

```
U   DBX3.1    //Consulta el bit 3.1 del bloque de datos abierto
```

Además se pueden cargar y transferir bytes, palabras y doble palabras.

Ejemplos:

```
L   DBB1    //Carga en ACU1 el byte 1 del DB abierto

L   5       //Cargo 5 en ACU1
T   DBW4    //y lo transfiero a la palabra 4 del DB abierto

L   L#3     //Cargo el doble entero 3
T   DBD2    //y lo transfiero a la doble palabra 2 del DB abierto
```

No existe ninguna instrucción para cerrar un DB, sólo se cierra abriendo otro DB.

Ejemplo:

```
AUF   DB5      //Abro DB5
L     DBW8     //Carga en ACU1 la palabra 8 del DB5
AUF   DB7      //Abro DB5
T     DBW6     //Transfiero el contenido de ACU1 a la palabra 6 del DB7
```

2) La forma más sencilla de acceder a los datos de un DB es indicando en la misma instrucción el número del DB. En este caso no es necesario abrir el DB con la instrucción "AUF".

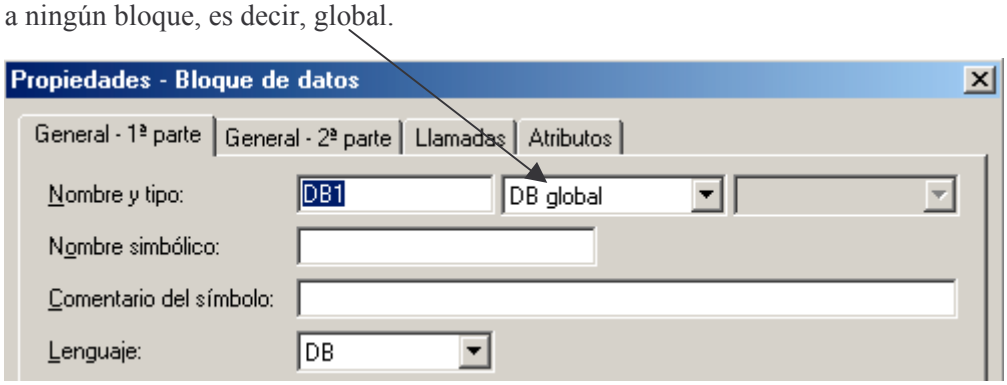
Ejemplo:

```
U     DB1.DBX3.0   //Consulto el bit 3.0 del DB1
L     DB1.DBB7     //Carga en ACU1 el byte 7 del DB1
L     DB2.DBW4     //Carga la palabra 4 del DB2
L     DB1.DBD4     //Cargo la doble palabra 4 del DB1
L     Datos.Dato   //Cargo la variable "Dato" del DB llamado "Datos"
```

18.8. Ejemplo de bloque de datos global

Vamos a crear una función matemática que nos realice la multiplicación de dos números enteros y nos devuelva el resultado. Todos los datos van a estar almacenados en un DB Global.

1. Creamos un DB y le damos el valor simbólico "Datos". Lo seleccionamos como no asociado a ningún bloque, es decir, global.



2. El siguiente paso es declarar los datos. Para ello vamos al menú "Ver" y marcamos "Declaración". Definimos la estructura rellenando la tabla:

Dirección	Nombre	Tipo	Valor inicial	Comentario
0.0		STRUCT		
+0.0	valor1	INT	5	
+2.0	valor2	INT	3	
+4.0	resultado	INT	0	
=6.0		END_STRUCT		

El valor inicial es valor predeterminado del tipo de variable que el programa usará si no hemos definido nosotros ningún valor actual.

Para definir los valores actuales vamos a menú "Ver" y marcamos "Datos". Los valores actuales son en realidad los que están almacenados en el DB y que podemos leer mediante la instrucción L de carga y escribir mediante la instrucción T de transferencia.

Dirección	Nombre	Tipo	Valor inicial	Valor actual	Comentario
0.0	valor1	INT	5	2	
2.0	valor2	INT	3	0	
4.0	resultado	INT	0	0	

A la variable valor1 le damos valor actual 2. Si a la variable valor2 no le damos valor actual alguno, al guardar el DB el valor inicial se transferirá al actual, o sea, 3.

Guardamos el DB.

3. En el OB1 escribimos el programa

```
L    Datos.valor2
L    Datos.valor1
*I
T    Datos.resultado
```

Transferimos los dos bloques, OB1 y DB1 al autómata.

Para ver el valor del resultado ir a "Observar/Forzar"

18.9. Formato de datos en los DB

Datos simples:

- BOOL : 0 (FALSE) ó 1 (TRUE)
- BYTE: Formato: B#16#3F
  No permite introducirlo de la forma: 2#10010001
- WORD: W#16#AF
  Permite introducirlo de la forma 2#1001..., pero lo traduce a W#16#..
- DWORD: DW#16#FFFFAF2D
  Permite introducirlo de la forma 2#1001..., pero lo traduce a W#16#..
- INT: Números entre-32768 y 32767
- DINT: Números entre 32768 y 2147483647 y entre –32769 y –2147483648
  También se pueden introducir de la forma: L#5, L#-10
- REAL: Número Entre 1 (1.000000e+000)
- S5TIME: Máx: S5T#9990S

- TIME: Máx: T#24D20H31H23S647MS
- DATE: D2000-1-24 (Máx: D#2168-12-31)
- TIME_OF_DAY: Máx: TOD#23:59:59.999
- CHAR: Formato: 'C'
  Sólo un carácter que ocupa 1 byte

Datos compuestos:

- DATE_AND_TIME: DT#00-11-21-23:59:59.999
- STRING[Nº de caracteres]: Cadena de caracteres entre comillas: 'Siemens' STRING[7]
  El máximo de caracteres es 254
- ARRAY: Matrices
- STRUCT: Es una estructura dentro del DB. Sirve para agrupar conjuntos de datos de distinto tipo, simples o complejos. Empieza en STRUCT y acaba en END_STRUCT. Se pueden anidar hasta 8 subestructuras.

Ejemplo:

Dirección	Nombre	Tipo	Valor inicial	Comentario
0.0		STRUCT		
+0.0	Ficha	STRUCT		
+0.0	Nombre	STRING[4]	'pepe'	
+6.0	Edad	INT	0	
+8.0	telefono	STRUCT		
+0.0	fijo	INT	0	
+2.0	movil	INT	0	
=4.0		END_STRUCT		
=12.0		END_STRUCT		
=12.0		END_STRUCT		

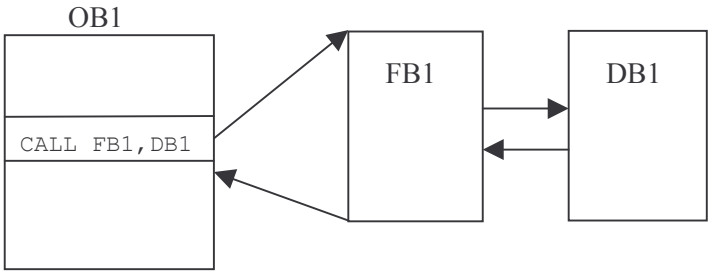
Ejemplos de acceso a los datos:

```
L   Datos.Ficha.Nombre
T   Datos.Ficha.telefono.fijo
L   DB5.DB8.DW0   //accede a Datos.Ficha.telefono.fijo
```

18.10. Bloques de función (FB)

Los bloques de función FB son bloques programables. Un FB es un bloque "con memoria". Dispone de un bloque de datos DB asignado como memoria (bloque de datos de instancia). Los parámetros que se transfieren al FB, así como las variables estáticas “stat”, se memorizan en el DB de instancia. Las variables temporales se memorizan en la pila de datos locales.

Los datos memorizados en el DB de instancia no se pierden al concluir el tratamiento del FB.
Los datos memorizados en la pila de datos locales se pierden al concluir el tratamiento del FB.

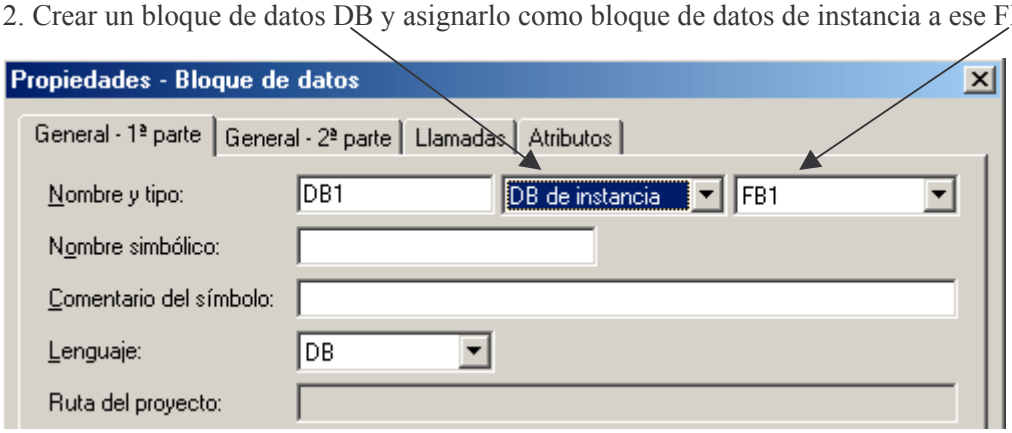


Creación de un FB :

1. Hacer clic con el botón izquierdo del ratón en la carpeta de Bloques e “Insertar nuevo objeto...” -> “Bloque de función”. Se le puede dar un nombre simbólico.

Rellenar su tabla de declaración de variables (véase capítulo 16.3)

2. Crear un bloque de datos DB y asignarlo como bloque de datos de instancia a ese FB:



Al asociar el DB a ese FB, el DB será una tabla de variables copia exacta de la tabla de declaración de variables del FB.

18.11. Llamada al FB

Instrucción: "CALL FBn ,DBn"

“CALL FBn , DBn” llama al FB independientemente del RLO.

Hay que asignarle un bloque de datos de instancia.

Una vez procesado el FB invocado, el programa del bloque invocante seguirá siendo procesado.

Ejemplo:

```
CALL FB1,DB1     //Llama al bloque de función FB1 de forma incondicional
```

Se pueden intercambiar datos con el bloque FB:

4. Crear en el FB una “Tabla de declaración de variables”, con sus parámetros in, out e in_out según corresponda.

5. Cuando en el bloque invocante se introduzca la función CALL FBn,DBn, automáticamente se crea una “tabla de parámetros formales” que son cogidos de la tabla de declaración de variables del FB.

```
CALL FB1,DB1
     Num de motor:=    }
     Velocidad:=       }Parámetros formales
     Potencia:=        }
```

6. Hay que asignar a esos parámetros formales los parámetros actuales del bloque invocante.

```
CALL FB1,DB1
    Num de motor:= MW100 }
    Velocidad:= MW110    }Parámetros actuales
    Potencia:= A0.1      }
```

Observaciones:

- Si se llama varias veces al FB, sólo hay que introducir los parámetros actuales que han cambiado respecto a la última llamada, ya que estos se han guardado en el DB de instancia una vez procesado el FB.

- Si el parámetro actual es un DB, se debe indicar siempre la dirección absoluta completa.

- Asignación de parámetros actuales a los parámetros formales de un FB:

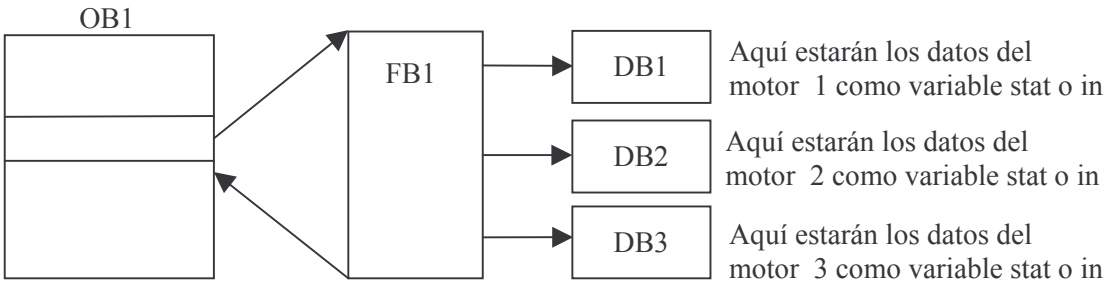
    - Si al FB se le indican los parámetros actuales en la instrucción de llamada, las operaciones del FB utilizan estos.

    - Si en la instrucción de llamada no se indica ningún parámetro actual, el FB utiliza los valores memorizados en el DB de instancia. A su vez pueden ser los valores iniciales declarados en el FB (excepto los temporales).

18.12. Multiinstancia : Un DB de instancia para cada instancia

Un mismo DB puede estar asociado a varios DB's, a estos DB's se les llama instancias, y cada una de esas instancias contiene distintos datos; dependiendo de cómo se llame al FB éste utilizará un DB u otro.

Por ejemplo, si se asignan varios bloques de datos de instancia aun bloque de función FB que controle un motor, se puede utilizar el mismo FB para controlar varios motores.



Dependiendo de cómo se llame al FB1:

`CALL FB1,DB1` → Utiliza los datos del motor 1
`CALL FB1,DB2` → Utiliza los datos del motor 2
`CALL FB1,DB3` → Utiliza los datos del motor 3

Ejemplo: Dependiendo de si es motor 1 (E0.0) o motor 2 (E0.1) activamos ese motor (con E0.2) un tiempo de 2 seg. o 5 seg. respectivamente.

Primero creamos un FB1. Y en su tabla de declaración de variables declaro la variable "tiempo"como parámetro tipo in o stat:

Dirección	Declaración	Nombre	Tipo	Valor inicial	Comentario
	in				
	out				
	in_out				
0.0	stat	tiempo	S5TIME	S5T#0MS	
	temp				

En el área de código del FB1:

```
U     E0.2
L     #tiempo
SV    T0
U     T0
=     A4.0
```

Ahora es el momento de crear el DB asociado al FB: después de definir las variables en la tabla de declaración de variables del FB. Creamos pues un DB1 asociado al FB1. Al abrir el DB1 vemos que contiene la variable "tiempo", en la cual introducimos el valor actual de tiempo para

el motor 1 (menú "Ver" → Datos):

Dirección	Declaración	Nombre	Tipo	Valor inicial	Valor actual	Comentario
0.0	stat	tiempo	S5TIME	S5T#0MS	S5T#2S	

Por el momento no creamos el DB2.

En el OB1 escribimos el siguiente código:

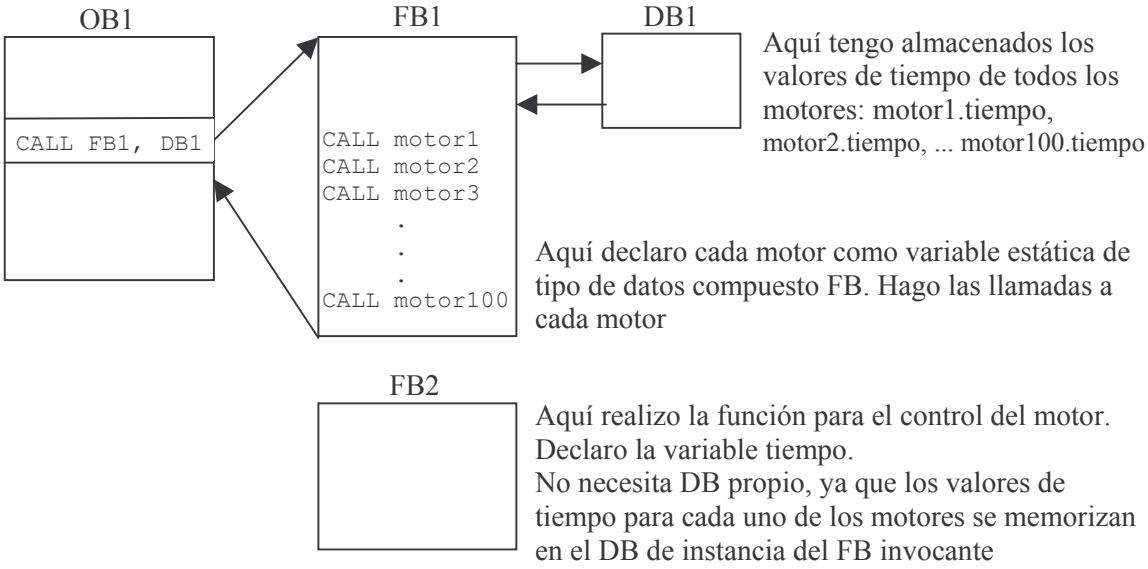
```
      U    E0.0
      SPB  mot1
      U    E0.1
      SPB  mot2
      BEA

mot1: CALL FB1, DB1
      BEA

mot2: CALL FB1, DB2    //Como el DB2 no existe, al introducir la llamada,
      BEA                 automáticamente el programa nos pregunta si
                          queremos crearlo. En el DB2 introduzco el valor
                          actual de tiempo para el motor2.
```

18.13. Multiinstancia : Un DB de instancia para varias instancias de un FB

En el caso anterior si tuviéramos 100 motores, habría que hacer 100 instancias = 100 DB's de instancia. Esto consume mucha memoria en el autómata.

A un FB se pueden transferir conjuntamente en un DB de instancia los datos de instancia para diferentes motores. A tal efecto, la llamada de los controles de motores se ha de efectuar en otro FB y en el área de declaración del FB invocante se deben declarar las variables estáticas con el tipo de datos de un FB para las diferentes instancias.

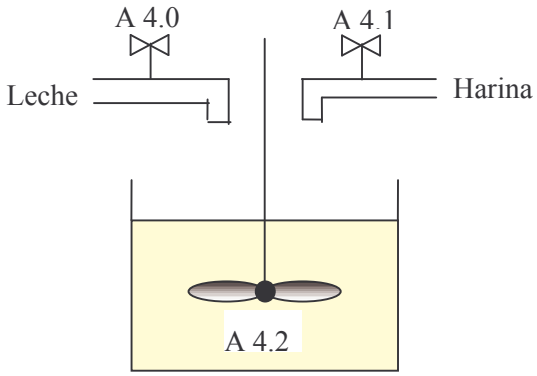
Utilizando un DB de instancia para varias instancias de un FB se ahorra capacidad de memoria y optimiza el uso de los bloques de datos.

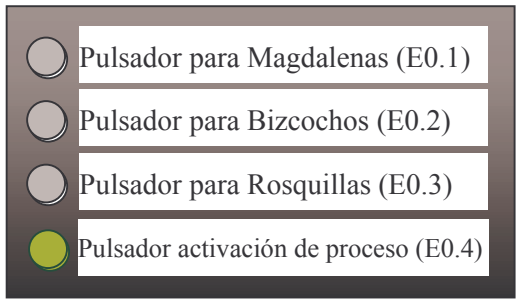


18.14. Ejercicio propuesto

El programa consiste en un mezclador de repostería . Hay una válvula para introducir leche (A4.0) y otra para introducir harina (A4.1); la mezcla se agita mediante una paleta accionada por un motor (A4.2). Dependiendo de los tiempos de introducción de los ingredientes y de funcionamiento de la paleta, podremos hacer tres diferentes productos según la tabla siguiente:

	Leche – A4.0	**Harina – A4.1**	**Agitador – A4.2**
Magdalenas	5 seg	2 seg	4 seg
Bizcochos	2 seg	3 seg	5 seg
Rosquillas	3 seg	4 seg	2 seg





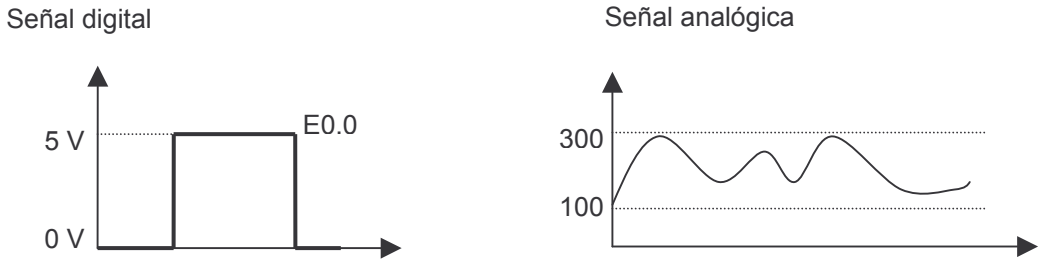
Realizar el automatismo de una manera estructurada:

1) Usando FC´s
2) Usando un DB global
3) Usando FBs con DBs de instancia
4) Usando DB de multiinstancia

19. Tratamiento de señales analógicas

Entradas analógicas: Se leen directamente de la periferia (PEW), no de la PAE.

Salidas analógicas: Se escriben directamente en la periferia, no en la PAA.

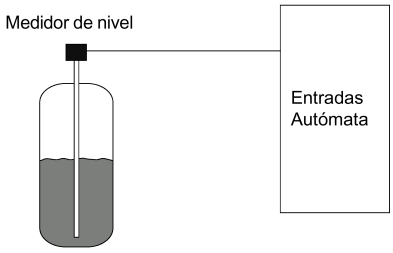


19.1. Entrada analógica

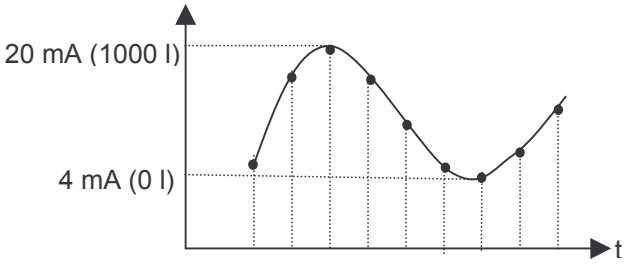
Una entrada analógica es una señal de entrada que tiene una señal continua (ininterrumpida).

Las entradas analógicas típicas varían de 0 a 20 mA 4 a 20 mA, o de 0 a 10 V.

En el ejemplo siguiente, un medidor de nivel de 4 a 20mA monitoriza el nivel de liquido en un tanque. Rango de medida 0 a 1000 litros. Dependiendo del medidor de nivel, la señal al autómata puede incrementarse o reducirse en la misma medida que el nivel se incremente o disminuya.



El autómata incorpora un convertidor analógico digital (A/D), en cada ciclo muestrea un valor y después lo traduce a bit. Cuantos más bits, más definición y precisión.



Estos valores van a ser un número entero, con la siguiente resolución máxima:

2^{16} = 65536 unipolar
-32767 a +32767 bipolar

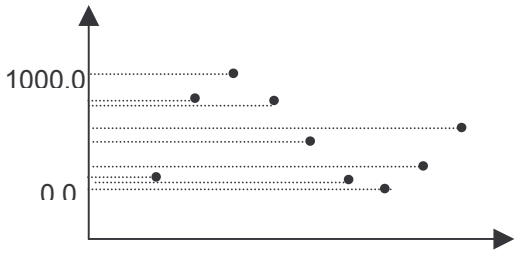
Para un módulo de entradas de 12 bits la resolución será: $2^{12} = 4096$. Pero nosotros veremos X cantidad de resolución, dependiendo de la tarjeta analógica.

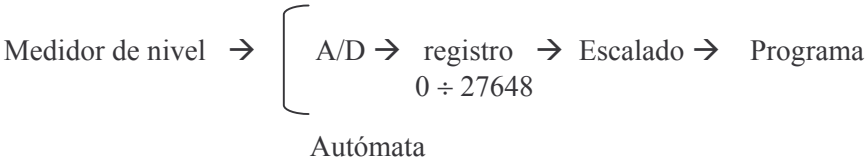
Tarjeta 8 bits= 2^8 =256 de resolución x 128 =32768 cantidad de resolución
Tarjeta 12 bits= 2^{12} =4096 de resolución x 8 =32768 cantidad de resolución

La resolución en la práctica es 27648.

Así, y en el ejemplo, con 0 litros obtendré el entero 0, con 500 litros obtendré 13824, y con 1000 litros obtendré 27648.

Después escala esos valores y los convierte en un número real.

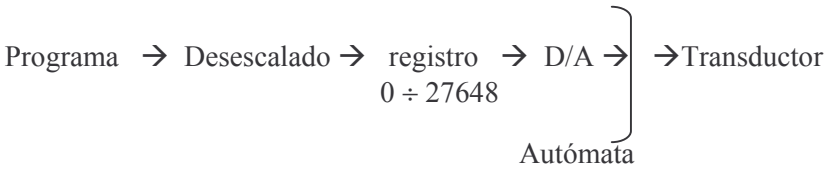




19.2. Salida analógica

Una salida analógica es una señal de salida que tiene una señal continua. La salida puede ser algo tan simple como un tensión de 0-10 VCC que acciona un aparato de medida analógico.

El tratamiento se hace mediante un convertidor digital-analógico (D/A)



19.3. Direccionamiento señales analógicas

La CPU accede directamente a las entradas y salidas de módulos de entradas/salidas analógicas.

Direccionamiento de periferia (acceso directo): Las entradas y salidas disponen de su propia área de direcciones.

La dirección de un canal de entrada o salida analógico es siempre una dirección de palabra. La dirección de canal depende siempre de la dirección inicial del módulo.

ENTRADAS EXTERNAS:

Byte de entrada de la periferia	PEB	0 a 65535
Palabra de entrada de la periferia	PEW	0 a 65534
Palabra doble de entrada de la periferia	PED	0 a 65532

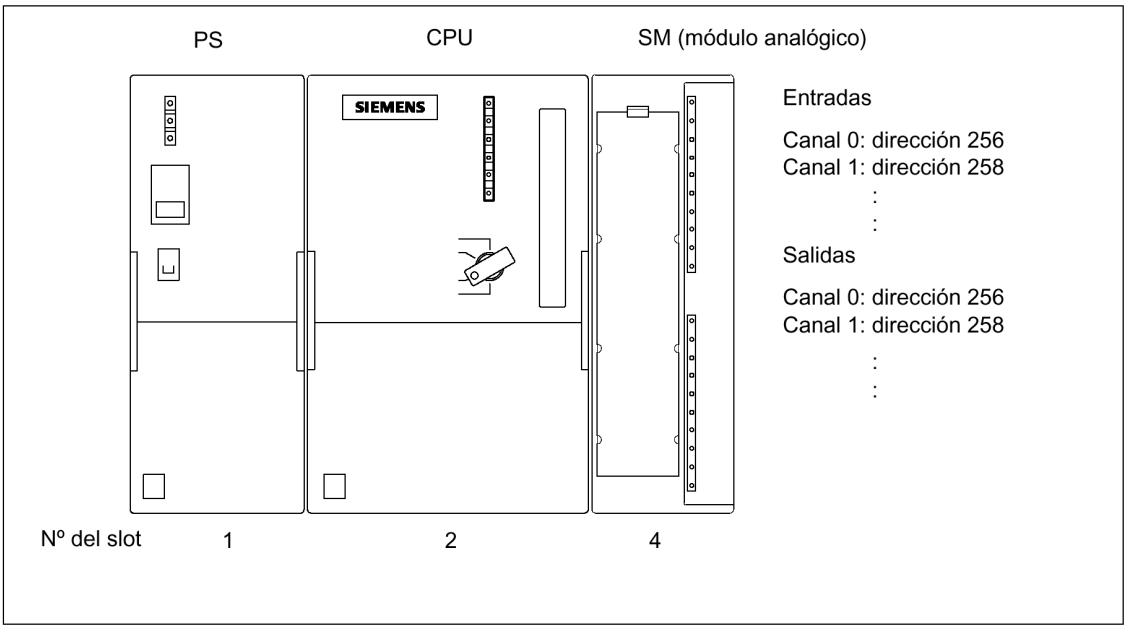
SALIDAS EXTERNAS:

Byte de salida de la periferia	PAB	0 a 65535
Palabra de salida de la periferia	PAW	0 a 65534
Palabra doble de salida de la periferia	PAD	0 a 65532

Si p.ej. el primer módulo analógico está enchufado en el slot 4, tiene la dirección inicial prefijada 256. La dirección inicial de cada módulo analógico añadido se incrementa en 16 por cada slot.

Un módulo de entrada/salida analógica tiene las mismas direcciones iniciales para los canales de entrada y salida analógicos.

Ejemplo para módulos analógicos:



19.4. Función de escalado de entradas analógicas (FC105)

Función: FC105 “SCALE”

Está función se inserta:
Menú Insertar → Elementos de programa --> Librerías --> Standard Library --> TI-S7 Converting Blocks --> FC105

```
CALL  "SCALE"
 IN     :=
 HI_LIM :=
 LO_LIM :=
 BIPOLAR:=
 RET_VAL:=
 OUT    :=
```

La función toma un valor entero a la entrada IN y lo convierte a un valor real, convirtiéndolo a escala entre un rango comprendido entre un límite inferior y un límite superior (LO_LIM y

HI_LIM). El resultado se escribe en la salida (OUT).

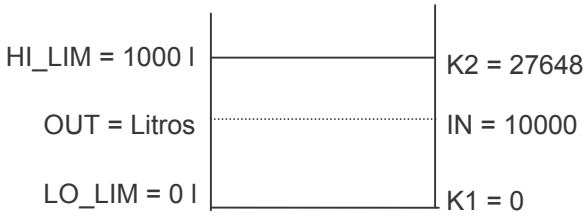
Formula:

$$\text{OUT} = \frac{(IN - K1)\, x\, (HI\_LIM - LO\_LIM)}{K1 - K2} + LO\_LIM$$

donde K1, K2:

- Bipolar: Se supone que el valor entero de entrada deberá estar entre K1= –27648 y K2= 27648.
- Unipolar: K1=0 y K2=27648

Por ejemplo, ¿cuántos litros se corresponden con el valor interno de tarjeta de 10000?



$$\text{Litros} = \frac{(10000 - 0)\, x\, (1000 - 0)}{27648 - 0} + 0 = 361'68\ l$$

Parámetros de la función SCALE (FC105):

Parámetros de entrada:

IN (INT) : Valor de entrada a escalar en valor real

HI_LIM (REAL) : Límite superior del rango de escala

LO_LIM (REAL) : Límite inferior del rango de escala

BIPOLAR (BOOL) : 1 para entrada bipolar, 0 para entrada unipolar

Parámetros de salida:

OUT (REAL) : Resultado de la conversión a escala

RET_VAL (WORD) : Código de retorno. Si devuelve el código W#16#0000 es que no se han producido errores. Para otros códigos véase la ayuda.

Observaciones:

- Los números reales (32 bits) se introducen con el formato correspondiente. Ej.: 10.0
- La OUT al ser un número real se deberá guardar como doble palabra. Ej.: MD100

19.5. Función de desescalado de salidas analógicas (FC106)

Función: FC106 “UNSCALE”

```
CALL  "UNSCALE"
 IN     :=
 HI_LIM :=
 LO_LIM :=
 BIPOLAR:=
 RET_VAL:=
 OUT    :=
```

La función toma en la entrada IN un valor real que está ajustado a escala entre un rango comprendido entre un límite inferior y un límite superior (LO_LIM y HI_LIM), y lo convierte a un valor entero. El resultado se escribe en la salida (OUT).

Formula:

$$\mathrm{OUT} = \frac{(IN - LO\_LIM)\, x\, (K2 - K1)}{(HI\_LIM - LO\_LIM)} + K1$$

donde K1, K2:

- Bipolar: K1= –27648 y K2= 27648.
- Unipolar: K1=0 y K2=27648

Parámetros de la función UNSCALE (FC106):

Parámetros de entrada:

IN (REAL) : Valor de entrada a desescalar, convirtiéndolo en un valor entero

HI_LIM (REAL) : Límite superior del rango de escala

LO_LIM (REAL) : Límite inferior del rango de escala

BIPOLAR (BOOL) : 1 para entrada bipolar, 0 para entrada unipolar

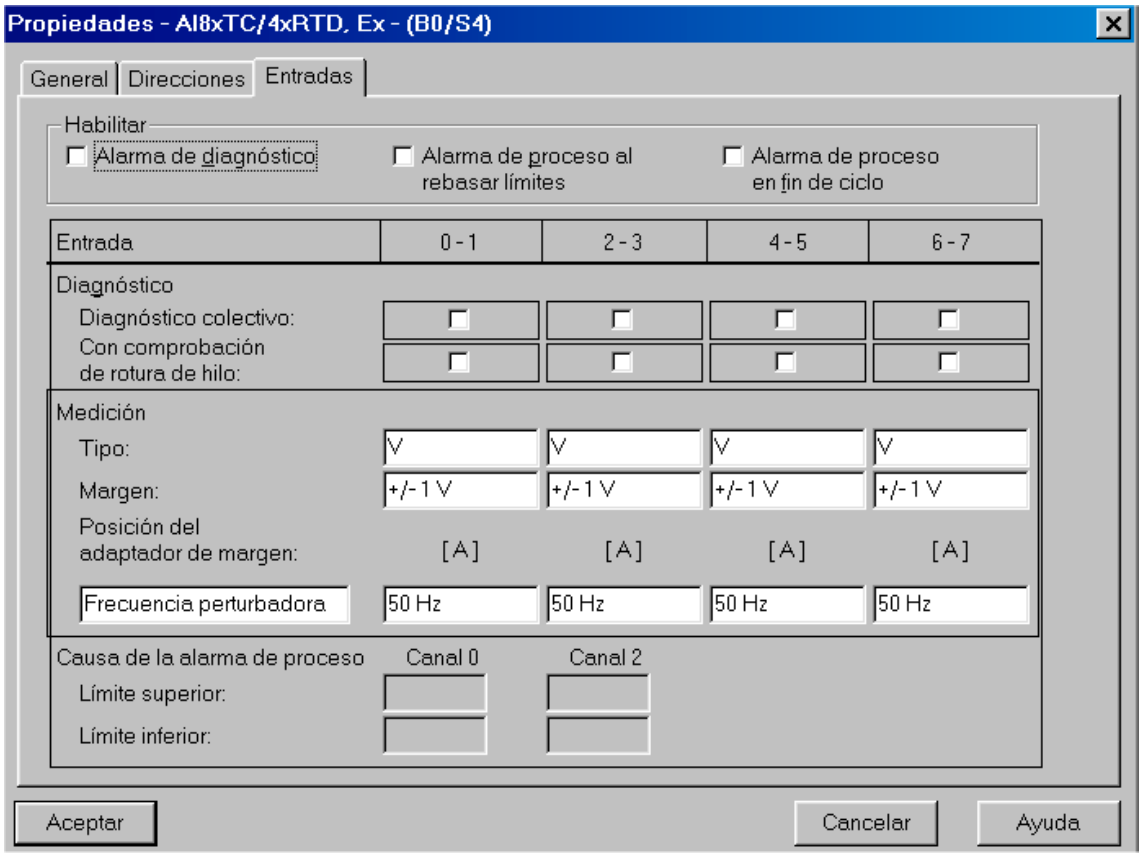
Parámetros de salida:

OUT (INT) : Resultado del desescalado

RET_VAL (WORD) : Código de retorno. Si devuelve el código W#16#0000 es que no se han producido errores. Para otros códigos véase la ayuda.

19.6. Parámetros de las tarjetas analógicas

Para configurar los parámetros de las tarjetas analógicas, ir al HW Config y hacer doble clic sobre la tarjeta.



Parámetros:

- Habilitar :

"Alarma de diagnóstico" : Si está activada y se presenta un evento de diagnóstico, el módulo desencadena una alarma de diagnóstico (OB80), que se produce por:

Error de configuración / parametrización
Error de modo común
Rotura de hilo (Requisito: La comprobación de la rotura de hilo está activada)
Rebase por defecto del margen de medida
Margen de medida rebasado
Falta tensión de carga L+

“Alarma de proceso al rebasar límites” : Cuando el valor de entrada sale del margen limitado por "Límite superior" y "Límite inferior", entonces el módulo dispara una alarma de proceso (=alarma de límite).

“Alarma de proceso al final del ciclo” : Un ciclo abarca la conversión de los valores medidos en todos los canales de entrada activos en el módulo con entradas analógicas. El módulo procesa los canales uno tras otro. Tras la conversión de valores medidos, el módulo notifica que todos los canales presentan valores de medición nuevos a la CPU mediante una alarma de proceso (=alarma fin de ciclo). Se puede utilizar la alarma de proceso para cargar continuamente los valores analógicos actuales convertidos. Para reducir el tiempo de ciclo desactive los canales

que no utilice.

"Diagnóstico colectivo“ : Si está activado y se produce un evento de diagnóstico, entonces se introduce una información al respecto en el área de datos de diagnóstico del módulo. Regla: si hay un grupo de canales que no está conectado, seleccione "desactivado". Entonces se actualizarán a intervalos más breves los demás valores de entrada.

“Tipo” : Haga clic en este campo para hacerse mostrar los tipos de medición disponibles (tensión, intensidad...) y elegir uno de ellos.

“Margen” : Haga clic en este campo para hacerse mostrar los márgenes de medida disponibles para el respectivo tipo de medición y elija uno de ellos.

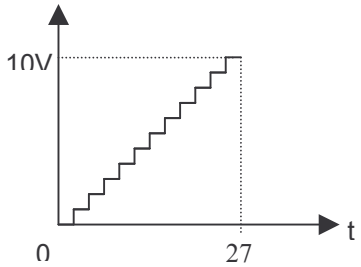
19.7. Ejercicios propuestos

Ejercicio 1: Detector de humo

Cuando la señal del detector de humo, para nosotros la entrada del potenciómetro del entrenador (rango 0 ÷ 10V), sea mayor de 5V, activar la señal de alarma A4.0.

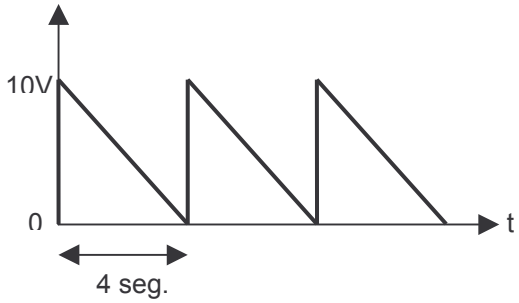
Ejercicio 2: Contador analógico

Realizar un contador analógico de 0 a 27 pasos que se visualice en el voltímetro de 0 a 10V.



Ejercicio 2: Onda “diente de sierra”

Simular una salida analógica en diente de sierra, pero utilizando instrucciones digitales (no entradas analógicas).



20. Eventos de alarma y error asíncrono

La serie Simatic S7 dispone de la capacidad de poder interrumpir el programa de usuario para poder atender de forma inmediata o retardada a un determinado evento. Las respuestas a las alarmas se deben programar, para definir los módulos OB a los cuales se saltará cuando se produzcan.

Se puede definir la prioridad de las alarmas, dando un orden de preferencia en la respuesta de las mismas, lo cual es imprescindible en aquellas situaciones en que se presenten varias alarmas.

También se puede bloquear el tratamiento de las alarmas y eventos de error, aunque no puede ser desactivado por la llamada de una FC estándar, si esta FC estándar incluye también los citados eventos que se habilitan nuevamente.

Para la programación de los eventos de alarma y error asíncrono se emplean las SFC 39 a 42 (ver Manual STEP7 Diseño de programas).

Las alarmas están subdivididas en diferentes clases. La siguiente tabla contiene todas las clases de alarmas y sus OB correspondientes:

Clase de alarma	**OB**
Alarmas horarias	OB 10 a OB 17
Alarmas de retardo	OB 20 a OB 23
Alarmas cíclicas	OB 30 a OB 38
Alarmas de proceso	OB 40 a OB 47
Alarmas de comunicación	OB 50 y OB 51
Alarmas de error asíncrono	OB 80 a OB 87 (siguiente tabla)
Alarmas de error síncrono	OB 121 y OB 122 El tratamiento de las alarmas de error asíncrono se enmascara o desenmascara con las SFC 36 a 38.

La siguiente tabla contiene los eventos de error asíncrono, a los cuales se puede reaccionar llamando el OB correspondiente en el programa de usuario.

Eventos de error asíncrono	**OB**
Error de tiempo (ej. sobrepasar el tiempo de ciclo)	OB 80
Fallo de la alimentación (ej. pila agotada)	OB 81
Alarma de diagnóstico (ej. fusible defectuoso en un módulo de señales)	OB 82
Fallo de inserción del módulo (ej. módulo sacado o mal insertado)	OB 83
Error de hardware de la CPU (ej. cartucho de memoria sacado)	OB 84

Error de proceso del programa (ej. OB no fue cargado)	OB 85
Ha fallado toda la fila	OB 86
Error de comunicación (ej. error de datos globales)	OB 87

Dependiendo de la CPU se dispondrá de unos determinados módulos OB accesibles. Por ejemplo, en la CPU 314 IFM disponemos de:

```
OB 1    ciclo libre
OB 35   control por tiempo
OB 10   control en tiempo real
OB 40   interrupción (alarma)
OB 100  recomienzo
```

20.1. Módulo de arranque OB100

Módulos de arranque dependiendo de CPU y tipo de arranque: OB100, OB101, OB102

Se ejecuta al producirse un arranque de la CPU. Un arranque se produce cuando:

- Después de que se conecta la alimentación ON.
- Si se cambia la llave de STOP a RUN

En estos casos el OB100 sólo se ejecutará una vez, un ciclo, y después se ejecuta el OB1 cíclicamente.

El OB100 nos puede ser útil para inicializar valores, por ejemplo inicializar un contador:

```
OB100

L C#5
S Z0
```

20.2. Alarma cíclica OB35

Alarmas cíclicas disponibles dependiendo de la CPU : OB30÷OB38
Para los S7-300 sólo dispondremos del OB35.

Interrumpe al OB1 cada cierto intervalo para llamar al OB de alarma cíclica.

El OB de alarma cíclica se configura en las propiedades de la CPU:

HW Config → Doble click sobre la CPU → Pestaña “Alarma cíclica”

Establecemos la periodicidad en ms con respecto al momento de arranque (STOP→RUN)

No olvidarse de “Guardar y compilar” y “Cargar en módulo”

Tan sólo nos queda crear el OB35 con el código que queremos que se ejecute en la interrupción, y transferirlo a la CPU.

20.3. Alarma horaria OB10

Alarmas cíclicas disponibles dependiendo de la CPU : OB10 ÷ OB17

Interrumpen el OB1 cíclico y ejecutan el código del OB de alarma horaria en una fecha especificada o a intervalos especificados.

El OB de alarma horaria se configura en las propiedades de la CPU:

HW Config → Doble click sobre la CPU → Pestaña "Alarmas horarias"

Configuración:

- *Activa* : Activa / desactiva la alarma
- *Periodicidad* :

  Una vez: Se ejecuta una vez, en el momento especificado.
  Cada minuto: Se ejecuta en intervalos de 1 minuto, desde el momento especificado.
  Cada hora : Se ejecuta en intervalos de 1 hora, desde el momento especificado.
  Cada día : Se ejecuta 1 vez al día, desde el momento especificado.
  Cada mes
  Cada semana
  Cada año

- *Fecha de arranque / hora* : A partir de la cual empieza la periodicidad, ejemplo:

  Fecha : 30.03.95
  Hora : 22:15:00

No olvidarse de "Guardar y compilar" y "Cargar en módulo"

Tan sólo nos queda crear el OB de alarma horaria con el código que queremos que se ejecute en la interrupción, y transferirlo a la CPU.

Observaciones :

- Si ponemos una fecha en pasado con respecto al reloj interno de la CPU y de periodicidad una sola vez, el OB de alarma horaria se ejecutaría 1 sola vez tras un rearranque completo (caliente) o un rearranque en frío.
- Para ajustar el reloj de la CPU:

  Administrador Simatic → Sistema de destino → Ajustar la hora...

20.4. Interrupción de retardo OB20

Alarmas cíclicas disponibles : OB20 ÷ OB23

El OB de retardo se arranca con la función de sistema SFC32 "SRT_DINT". El funcionamiento es que tras transcurrir el tiempo de retardo parametrizado llama al OB de alarma de retardo.

```
CALL  "SRT_DINT"
 OB_NR  :=
```

```
DTIME  :=
SIGN   :=
RET_VAL:=
```

Descripción de los parámetros:

Parámetros de entrada:

OB_NR (INT) : Número del OB que se arrancará al transcurrir el tiempo de retardo (OB20÷OB23)

DTIME (TIME) : Valor del retardo (1 a 60000 ms) Ej.: T #200MS

SIGN (WORD) : Signo que, al llamar el OB de alarma de retardo, aparece en la información de eventos de arranque del OB.

Parámetros de salida:

RET_VAL (INT) : Código de error, si ocurre un error al procesar la función. Códigos:
0000 No ha ocurrido ningún error.
8090 Parámetro OB_NR erróneo.
8091 Parámetro DTIME erróneo.

20.5. Más OB's

- En cada caso consultar la ayuda online para una descripción detallada.
- OBs de alarma de proceso (OB 40 hasta OB 47)
- OB de alarma de multiprocesamiento (OB 60)
- OB para errores de redundancia en la periferia (OB 70)
- OB para errores de redundancia en las CPU (OB 72)
- OB para errores de redundancia en la comunicación (OB 73)
- OB de error de tiempo (OB 80)
- OB de fallo de alimentación (OB 81)
- OB de alarma de diagnóstico (OB 82)
- OB de presencia de módulo (extraer/insertar) (OB 83)
- OB de avería de CPU (OB 84)
- OB de error de ejecución del programa (OB 85)
- OB de fallo del bastidor (OB 86)
- OB de error de comunicación (OB 87)
- OB de tarea no prioritaria (OB 90)
- OBs de arranque (OB 100, OB 101 y OB 102)

- OB de error de programación (OB 121)
- OB de error de acceso a la periferia (OB 122)

21. Direccionamiento indirecto

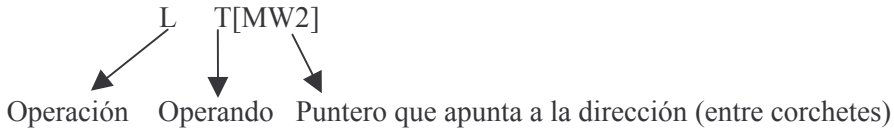
Hay tres tipos de direccionamiento indirecto:

- Direccionamiento indirecto por memoria
- Direccionamiento indirecto por registro intraárea
- Direccionamiento indirecto por registro interárea

21.1. Direccionamiento indirecto por memoria

La dirección del operando viene indicada por un puntero.

Formato de instrucción con direccionamiento indirecto por memoria:



En este ejemplo la instrucción es una carga en ACU1 del valor del temporizador cuyo número está contenido en MW2.

Según el identificador del operando, la operación evalúa los datos depositados en la dirección como puntero en formato palabra o doble palabra.

Puntero en formato palabra, indica el número en formato decimal de:

- Temporizadores (T)
- Contadores (Z)
- Bloques de datos (DB, DI)
- Bloques lógicos (FC, FB)

Ej.:

```
L  5        //Carga 5 en ACU1
T  MW2      //Transfiero 5 a MW2
L  T[MW2]   //Carga el valor actual del temporizador T5 en ACU1
```

Puntero en formato doble palabra, indica la dirección de:

- Bit
- Byte
- Palabra
- Doble palabra

Un puntero tiene el siguiente formato como doble palabra:

P#Byte.Bit

Ej.:

```
L   P#4.1           //Carga el puntero en ACU1
T   MD2             //Transfiero el resultado a una doble palabra
U   E[MD2]          //Consulta el estado de la señal en la entrada E4.1
=   A[MD2]          //y asigna ese estado de señal a la salida A4.1
```

Para direccionar byte, palabra y doble palabra el bit de puntero siempre debe ser 0.

Ej.:

```
L   P#4.0           //Carga el puntero en ACU1
T   MD2             //Transfiero el resultado a una doble palabra
L   EB[MD2]         //Cargo el ACU1 el byte de entradas 3
```

Observaciones:

- El puntero deberá almacenarse en:

  Marcas (M)

  Bloques de datos (DB)

  Datos locales (L)

- Direccionamiento indirecto de bloques de datos:

  Ej.:

```
L    1              //Cargo 1 en ACU1
T    MW10           //Transfiero 1 a MW10
L    P#4.0          //Cargo el puntero en ACU1
T    MD2            //Transfiero el puntero a MD2
AUF  DB[MW10]       //Abro el DB1
L    DBD[MD2]       //En ACU1 cargo DBD4 (La doble palabra 4 del DB)
```

No se podría hacer así: `L DB[MW10].DBD[MD2]`

Además, en el ejemplo deberá existir previamente el DB1, si no, da fallo.

- Los punteros se pueden sumar.

  Ej.:

```
L    P#124.2            //Cargo puntero
L    P#0.7              //Cargo puntero
+D                      //Los sumo: 124.2 + 0.7 = P#125.1
```

  Ej.:

```
L    P#124.2            //Cargo puntero
L    3                  //Sumo 3 bits
```

```
+D                          //Resultado: P#124.5
```

- Se puede observar el puntero con “Observar/Forzar Variable”.

Ej.:

Operando	Formato de estado
MD4	Puntero

21.2. Direccionamiento indirecto por registro intraárea

Existen dos registros de direcciones de 32 bits, AR1 y AR2, que almacenan los punteros intraárea o interárea que utilizan las instrucciones que usan el direccionamiento indirecto por registro.

El formato de una instrucción que utilice el direccionamiento indirecto por registro intraárea es el siguiente:

Operación *Identificador del operando[Registro de direcciones AR1 ó AR2, Operando]*

La ventaja es que se puede modificar dinámicamente el operando.

El operando lo indican dos punteros, por ejemplo:

```
=  A[AR1,P#1.1]
  (*)     (**)
```

(*) Identificador de operando, puede ser:

- Operaciones lógicas que direccionan bits: E, A, M, L, DBX
- Con la función carga (L) que direcciona bytes palabras y doble palabras: E, A, M, L, D y PE
- Con la función transferencia (L) que direcciona bytes palabras y doble palabras: E, A, M, L, DB, DI y PA

(**) Puntero de doble palabra, es el *offset* (desplazamiento) que se suma al contenido del registro de direcciones donde previamente debe estar cargado otro puntero en formato doble palabra (intrucción “LAR1”). La suma de los dos punteros es byte+byte y bit+bit.

Ej.:

```
L  P#8.7          //Cargo el valor del puntero en ACU1
LAR1              //Transfiero el puntero de ACU1 a AR1
U  E[AR1,P#0.0]   //Consulto el estado de señal de la entrada
                    E8.7 (8.7 + 0.0 = 8.7)
=  A[AR1,P#1.1]   //y lo asigno a la salida A10.0
                    (8.7 + 1.1 = 9.8, que no existe, por tanto
                     será el siguiente al 9.7, que es 10.0)
```

Para acceder a un byte, palabra o doble palabra el nº de bit del puntero debe ser cero, ej.:

```
L   P#4.0
LAR1
L   EB[AR1,P#10.0] //Cargo en ACU1 el byte de entradas EB14.0
```

21.3. Direccionamiento indirecto por registro interárea

Es similar al direccionamiento indirecto por registro intraárea, pero ahora el área de memoria del operando va incluida en el puntero, por ejemplo:

```
=  [AR1,P#E8.7]
```

área puntero interárea

Ej.:

```
L   P#E8.7
LAR1
L   P#A8.7
LAR2
U   [AR1,P#0.0]    //Consulta la entrada E8.7
=   [AR2,P#1.1]    //Lo asigna a la salida A10.0
                      (8.7 + 1.1 = 9.8 = 10.0)
```

Para acceder a un byte, palabra o doble palabra, el nº de bit de puntero debe ser 0:

```
L   P#MB0
LAR1
L   B[AR1,P#10.0]  //Carga el byte MB10 en ACU1
```

Otro ejemplo:

```
L   P#MD1
LAR2
T   D[AR2,P#53.0]  //Transfiere el contenido de ACU1 a la doble
                     palabra MD54
```

21.4. Operaciones con el registro de direcciones

Instrucciones: “+AR1”, “+AR2”, “+AR1 P#Byte.Bit” y “+AR2 P#Byte.Bit”

“+AR1” y “+AR2” suma el contenido de ACU1 (un entero de 16 bits que será el desplazamiento) al contenido de AR1. La suma se hace a nivel de bits.
Esto es: ACU1+AR1->AR1

Ej:

```
L   P#4.1
LAR1
L 8
+AR1               //4.1 + 8 bits = 5.1
U   E0.0
=   A[AR1,P#0.0]   //Se activa la salida A5.1
```

“+AR1 P#Byte.Bit” me suma el puntero P#Byte.Bit al puntero contenido en AR1, por ejemplo:

```
L   P#4.1
LAR1
```

```
+AR1 P#5.0          //4.1 + 5.0 = 9.1
U  E0.0
=  A[AR1,P#0.0]    //Se activa la salida A9.1
```

21.5. Ejercicio propuesto

Sistema de adquisición de datos

Se supone que nos van llegando datos mediante comunicación con otro autómata, y se van almacenando desde la palabra marca MW0, MW2, MW4... hasta la MW10. Crear una función FC1 la cual se encargue de pasar esos datos al bloque de datos DB1 (mediante flanco positivo de E0.0) donde quedarán almacenados en DBW0...DBW10.

22. Array - Matrices

Un Array combina varios datos de un solo tipo (simples o compuestos) formando una unidad.

Para definir un array en un DB o en una tabla de declaración de variables:

1. Indicar el nombre del array.
2. Declararlo con la clave `ARRAY[]`
3. Indicar el tamaño del array a través de un índice. Se ha de indicar el primer y el último número de las dimensiones (6 máximo) [límite inferior..límite superior]
4. Indicar el tipo de datos a memorizar en el array.

Ejemplo: Matriz unidimensional

Array de 1 dimensión: `ARRAY [1..3]`

lím. inferior lím. superior

Ejemplo: En el DB1 defino un Array[1..3] de tipo de datos INT:

Dirección	Nombre	Tipo	Valor inicial	Comentario
0.0		STRUCT		
+0.0	Matriz	ARRAY[1..3]		
*2.0		INT		
=6.0		END_STRUCT		

Para acceder a estos datos: Matriz[1], Matriz[2] y Matriz[3]:

Dirección	Nombre	Tipo	Valor inicial	Valor actual	Comentario
0.0	Matriz[1]	INT	0	0	
2.0	Matriz[2]	INT	0	0	
4.0	Matriz[3]	INT	0	0	

Nota: Para introducir valores iniciales en el array:

- Individualmente: Valores separados por coma. En nuestro ej.: 5,7,-100
- Indicar factor de repetición: x(y)

nº de elementos valor inicial

Ejemplo: 3(8) En nuestro ejemplo le daría a los tres elementos valor 8

1, 2(37) Al primero le doy valor 1 y a los dos últimos valor 37.

Ejemplo: Matriz bidimensional

`ARRAY [1..2,1..3] = Matriz de 2 x 3 ,` primer elemento= `[1,1]`

último elemento= `[2,3]`

Para acceder a estos datos: Matriz[1,1], Matriz[1,2], Matriz[1,3], Matriz[2,1], Matriz[2,2] y Matriz[2,3]

Observaciones:

- El índice puede ser un valor entre –32768 y 32767

  Ej.: Datos[-1,1] tiene como elementos: Datos[-1], Datos[0] y Datos[1]

- Las matrices se pueden usar como parámetros. Si es así, se debe transferir como array completo y no como elementos individuales

  Ej.: Declaro en el OB1 una variable temporal tipo Array, y en el FC1 un parámetro de entrada "in" tipo Array y de las mismas dimensiones.

OB1:

20.0	temp	Datos	ARRAY[1..3]		
*2.0	temp		INT		

FC1:

Dirección	Declaración	Nombre	Tipo	Valor inicial	Comentario
0.0	in	Datos_temp	ARRAY[1..3]		
*2.0	in		INT		
	out				
	in_out				
	temp				

```
CALL FC1
  Datos_temp:=#Datos
```

Se puede mandar sólo un elemento:

```
CALL FC1
  Datos_temp:=#Datos[1]
```

23. Comunicación de datos globales

La comunicación de datos globales (comunicación GD) es una variante de comunicación sencilla integrada en el sistema operativo de las CPUs S7-300/S7-400.

La comunicación GD permite intercambiar datos cíclicamente entre CPUs a través del interface MPI. El intercambio cíclico de datos se lleva a cabo con la imagen normal del proceso.

La comunicación de datos globales se configura con STEP 7; la transferencia de los datos globales es cosa del sistema, por lo que no se tiene que programar.

Para comunicar varias CPU's a través de esta red MPI (máx. 15 CPU's) se configura la Tabla de Datos Globales. Está diseñada para cantidades de datos reducidos. Estos datos se transmiten cíclicamente entre CPU's.

Se pueden transmitir:

- Entradas y salidas (de la imagen del proceso PAE y PAA)
- Marcas
- Áreas de bloques de datos
- Temporizadores y contadores (no recomendables, porque los valores del emisor ya no son actuales; configurables sólo como áreas de operandos)

Las áreas de la periferia (PE y PA) y los datos locales no se pueden utilizar para la comunicación mediante datos globales.

El intercambio de datos globales se realiza de la siguiente forma:

- CPU emisora: Envía datos al final de un ciclo cada X ciclos.
- CPU receptora: Recibe datos al inicio de un ciclo cada X ciclos.

No se acusa recibo de los datos globales. El emisor no recibe información alguna acerca de si hay un receptor que ha recibido los datos globales enviados y, en caso de haberlo, cuál es.

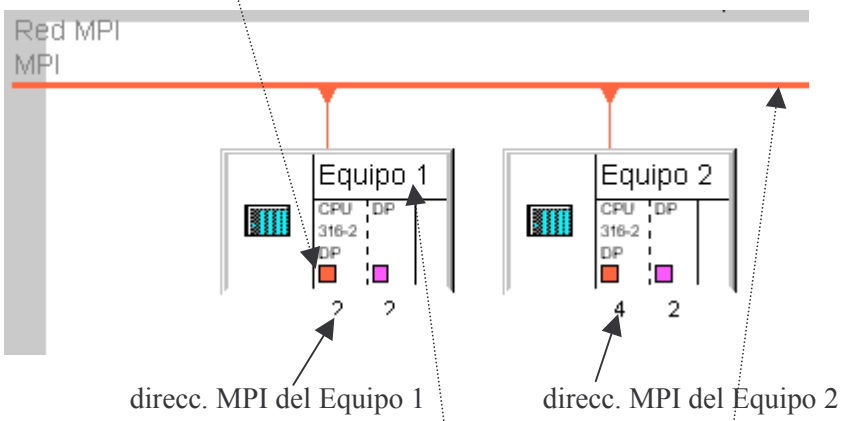
23.1. Configuración de una red GD

1. Establecer la dirección MPI para la PG ("Ajustar interface PG/PC") y la CPU (En HWConfig doble clic sobre la CPU → Pestaña "General" → Propiedades del Interface).

2. Insertar y configurar el hardware de los dos equipos a comunicar:

3. Con el programa "NetPro" configuramos la red MPI. Para acceder al "NetPro": Menú "Herramientas" --> "Configurar red", o bien haciendo clic en el icono 

Si todavía no hemos configurado la red MPI lo podemos hacer en el NetPro. Haciendo doble clic sobre el cuadrado rojo, que representa el puerto MPI, o también simplemente haciendo clic sobre ese puerto y arrastrando hasta encontrar a la línea roja que representa la red MPI, conectaremos la CPU a la red MPI. Además nos aparecerá una ventana para configurar el interface MPI para esa CPU. Es importante asignarle a cada CPU una dirección MPI distinta dentro de la red.



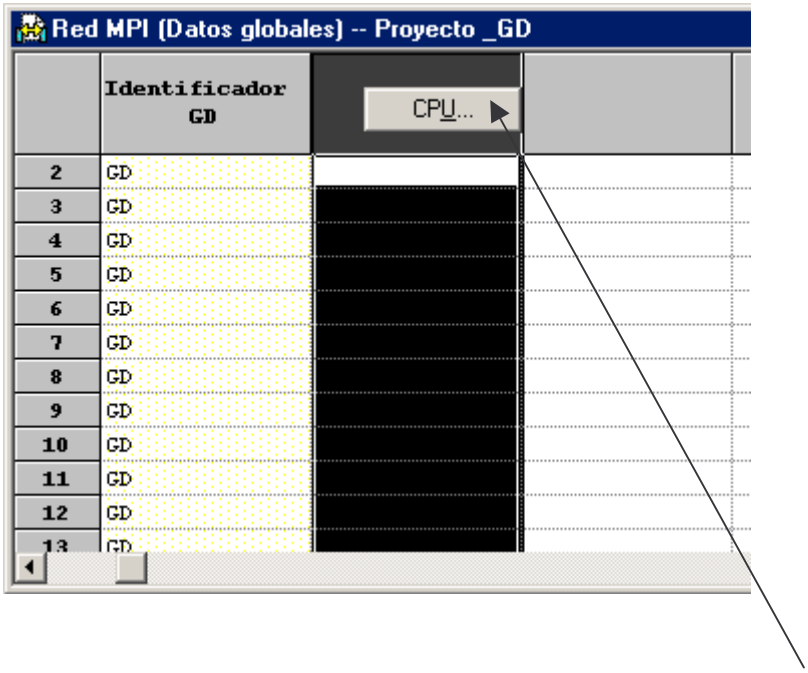
4. "Guardar y compilar" y, pinchando en cada equipo, cargar esta configuración en las dos CPU's con el icono: 

5. Ahora habrá que configurar la Tabla de Datos Globales para el intercambio de datos. A la tabla podemos acceder de dos maneras:

- Desde el NetPro, seleccionado la red MPI (la línea roja).
- Desde el Administrador Simatic, dentro de la carpeta de nuestro proyecto, seleccionamos el icono que representa a la red MPI :

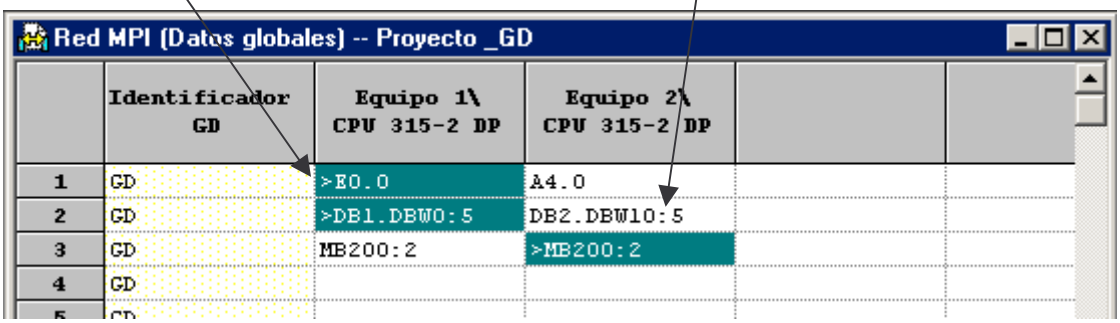
Y en ambos casos menú "Herramientas" --> "Definir datos globales"

Nos aparecerá la Tabla de datos Globales que deberemos rellenar.



Insertamos los equipos de nuestra red por columnas. Seleccionamos la primera columna, y con el botón derecho del ratón --> "CPU...", y en la ventana siguiente escogemos la CPU de un equipo.

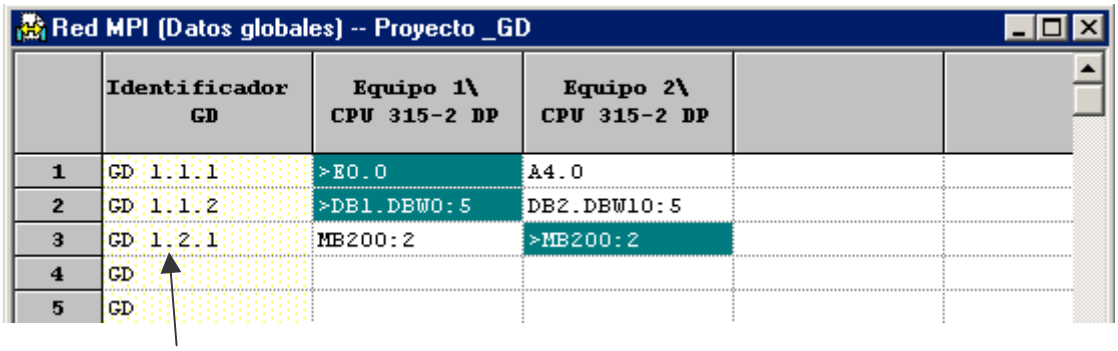
Después rellenamos los datos globales línea a línea. Un dato que se emite por un equipo se deposita en un área de memoria de otro equipo. El dato que se emite se identifica por el símbolo ">" delante de él. El número de datos que se envían se indica con el símbolo ":".



En este ejemplo el equipo 1 transfiere el estado de su entrada E0.0 a la salida A4.0 del equipo 2; el equipo 1 transfiere las 5 primeras palabras de su DB1 a 5 palabras del DB2 (empezando por la palabra 10) del equipo 2; el equipo 2 transfiere el byte 200 y 201 al byte 200 y 201 del equipo 1.

Observación: Para cambiar los datos de una casilla utilizar la tecla F2.

Después de rellenar la tabla se hace una primera compilación (menú "Tabla GD" --> "Compilar", o icono).

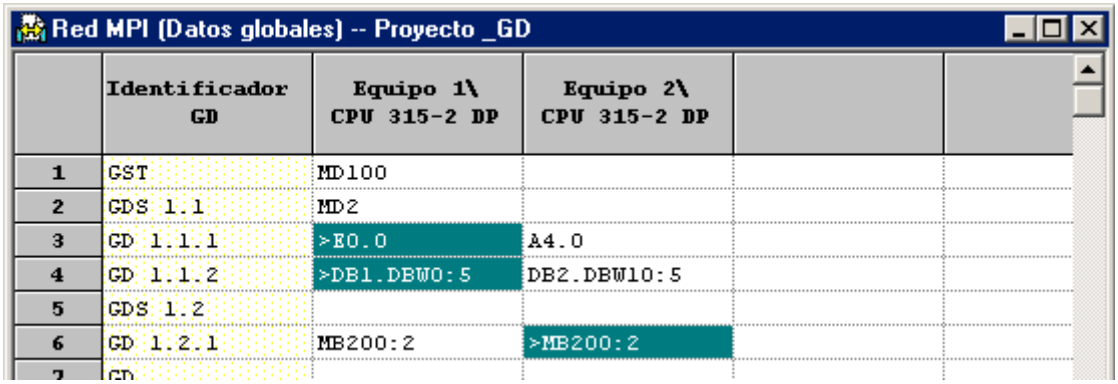


Tenemos las líneas de datos globales con su identificador.

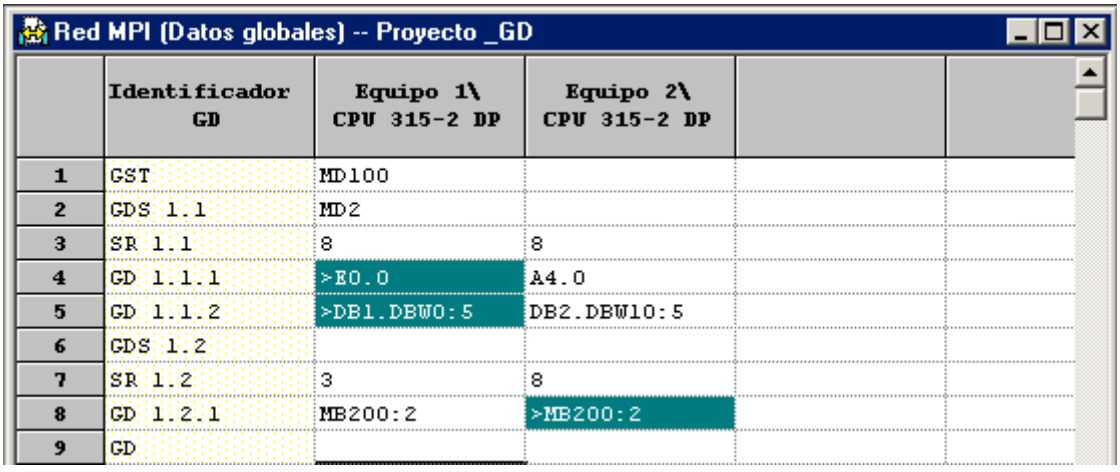
Podemos insertar otros tipos de líneas en la tabla:

- Línea de estado (GSD): Se puede definir para cada paquete de GD un palabra doble de estado para cada CPU.
- Línea de estado global (GST): Se puede definir para cada paquete de GD un palabra doble de estado para cada CPU. Es una combinación OR de todas las líneas de estado.

Estas líneas se pueden ver después de la primera compilación mediante el menú "Ver" --> "Estado GD".



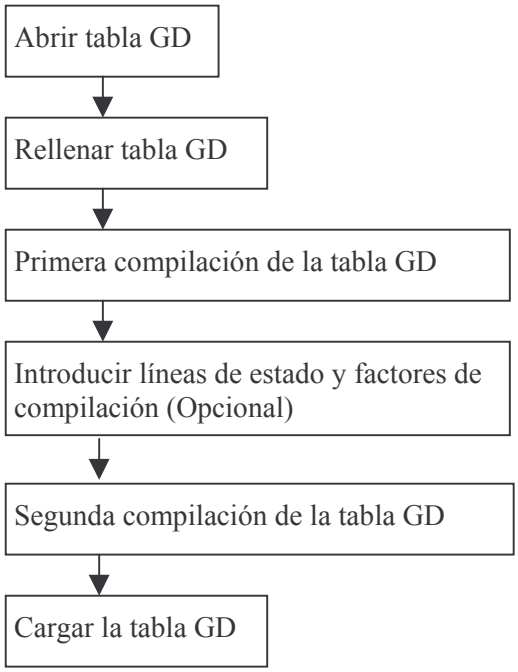
También podemos insertar la Línea de Factor de Ciclo (SR): Indica después de cuantos ciclos la CPU debe actualizar los datos globales. Para ver/modificar los factores de ciclo predefinidos: Menú "Ver" --> "Factores de ciclo". Factores de ciclo validos: 0;1-255. Hay que tener en cuenta que factores de ciclo demasiado bajos sobrecargan la CPU.



En el ejemplo el Equipo 1 emite cada 8 ciclos (al final de ciclo) y recibe cada 3 ciclos (al principio de ciclo).

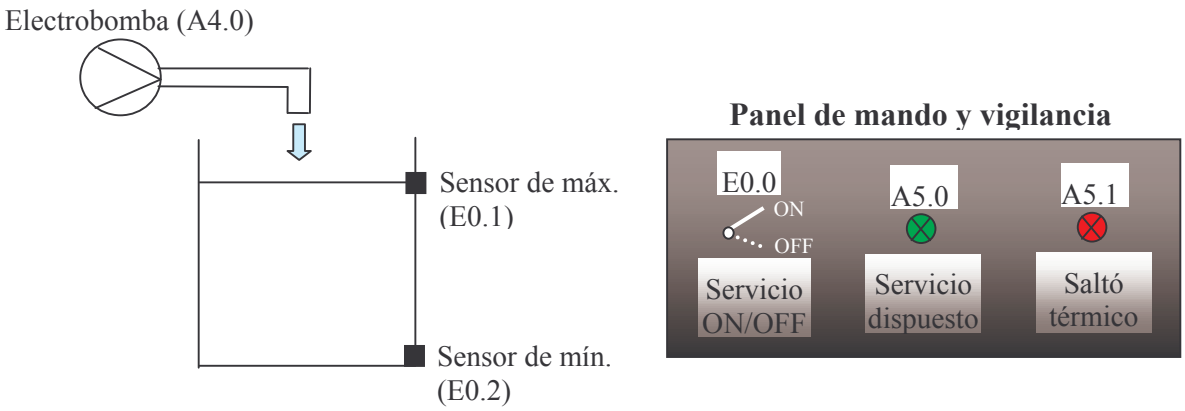
Para terminar hay que hacer una segunda compilación y cargar la tabla GD en cada CPU mediante el icono correspondiente o el menú “Sistema de destino” --> “Cargar en módulo...”.

Resumen :



24. Ejercicios propuestos

24.1. Bomba de agua



El estado de disponibilidad lo indica un selector de dos posiciones: Conectado/desconectado.

La electrobomba se pondrá en marcha cuando el sensor de mínimo esté activado y se apagará cuando, o bien se active el sensor de máximo, o bien salte el térmico o bien la desconectemos mediante el sensor.

Si la electrobomba está en servicio deberá lucir una lámpara indicándolo.

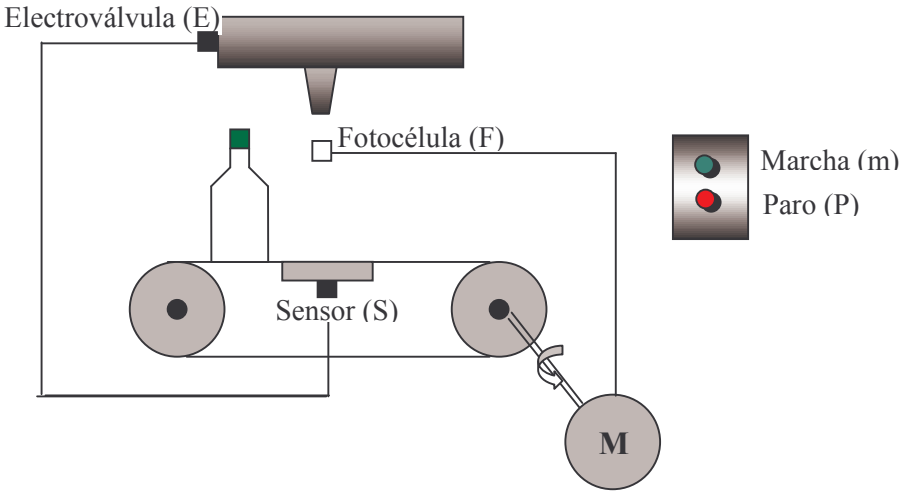
Si salta el térmico se encenderá otra lámpara avisándolo.

24.2. Control de llenado de botellas

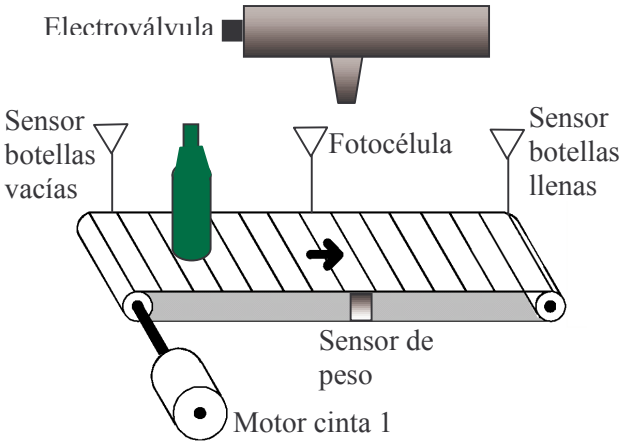
Diseñar un automatismo para el llenado de botellas hasta un cierto nivel, de acuerdo con el siguiente programa de trabajo:

- Al pulsar sobre m (marcha) el motor M de la cinta transportadora arrancará.
- Cuando la fotocélula F detecte una botella, el motor M se parará y se activará la electroválvula E para el llenado de la botella.
- Cuando la botella pese lo deseado, el sensor S pasará a 1, mandando una señal de paro a la electroválvula.
- Pasados 2 seg. el motor se pondrá de nuevo en marcha, parándose en la próxima detección.
- La cinta podrá parase en cualquier momento mediante un pulsador de paro P y arrancarse, si se desea, a continuación mediante m.

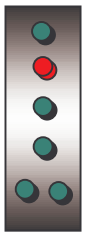
Asignación de variables	
Marcha (m) = E0.0	Motor (M) = A4.0
Paro (P) = E0.1	Electroválvula (E) = A4.1
Fotocélula (F) = E0.2	
Sensor (S) = E0.3	



24.3. Control de llenado de botellas 2



Panel de mando:



Marcha
Paro
Manual /automático
Acuse de recibo
Adelante /atrás

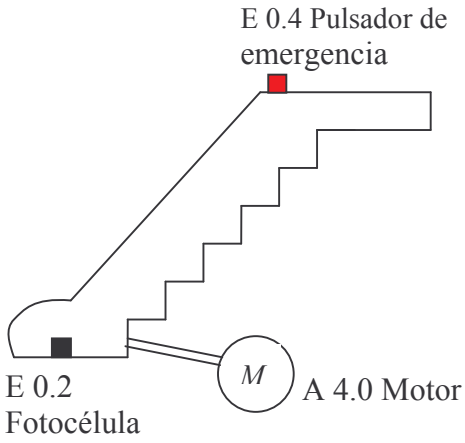
Diseñar un automatismo para el llenado de botellas hasta un cierto nivel, de acuerdo con el siguiente programa de trabajo:

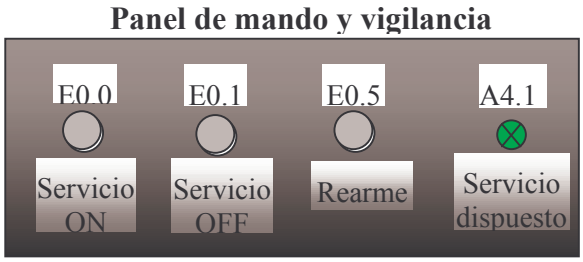
- *Modo automático* (por defecto) :
  - Al pulsar sobre E0.0 (marcha) el motor de la cinta transportadora arrancará y funcionará hacia delante.
  - Cuando la fotocélula E1.0 detecte una botella, el motor se parará y se activará la electroválvula A4.2 para el llenado de la botella.
  - Cuando la botella pese lo deseado, el sensor de peso E1.1 mandará una señal de paro a la electroválvula.
  - Pasados 2 seg. el motor se pondrá de nuevo en marcha.
  - Las botellas que se llenen van pasando a la cinta transportadora 2 (A4.3) la cual funcionará durante 4 seg.
  - Cada vez que se contabilicen 5 botellas, el proceso deberá detenerse, y el operario deberá pulsar el botón de acuse de recibo E0.3 para que el proceso pueda continuar.
  - La cinta podrá pararse en cualquier momento mediante un pulsador de paro E0.1, y rearrancarlo, si se desea, a continuación mediante el pulsador de marcha.
- *Modo manual* (por si se ha de rectificar la posición de la botella) :
  - Funcionamiento similar al automático, pero el operario tendrá que mandar la cinta transportadora 1 mediante los botones de adelante y atrás.

Además, y con el fin de evitar atascos en la línea, mediante los dos sensores de botellas vacías y llenas se ha de comprobar que las botellas que salen se corresponden con las que han entrado. Si se detectase un atasco, activar una luz de fallo, que se detenga el proceso, y para reiniciar el ciclo habrá que pulsar antes el botón de acuse de recibo.

Asignación de variables	
Marcha = E0.0	Motor adelante = A4.0
Paro = E0.1	Motor atrás = A4.1
Manual /auto. =E0.2	Electroválvula = A4.2
Acuse de recibo = E0.3	Cinta 2 = A4.3
Motor adelante = E0.4	Luz manual = A5.0
Motor atrás = E0.5	Luz auto. = A5.1
Fotocélula = E1.0	Luz 5 botellas = A5.2
Sensor de peso = E1.1	Luz de atasco = A5.3
Sensor botellas vacías = E1.2	
Sensor botellas llenas = E1.3	

24.4. Control de una escalera mecánica



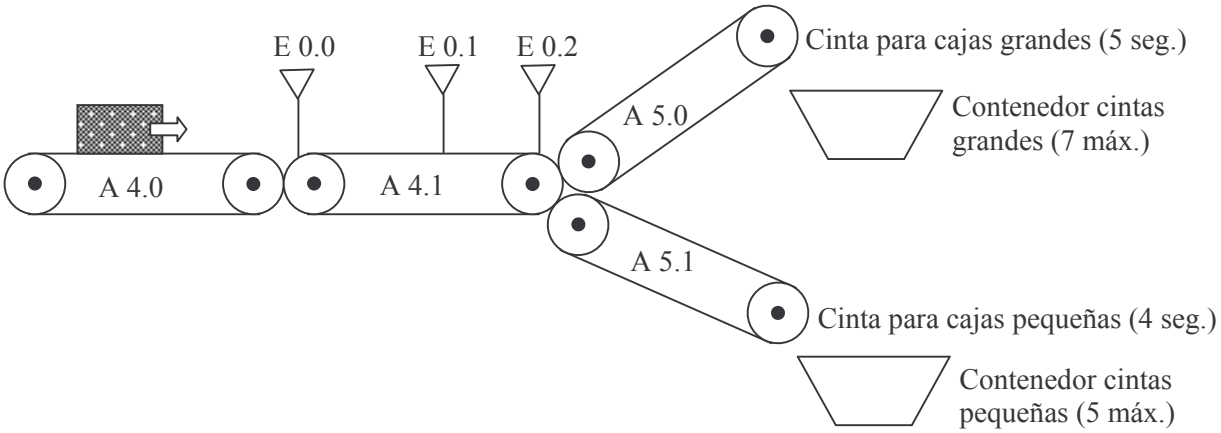


Para la que escalera funcione deberá accionarse el pulsador de servicio "ON" (E0.0), iluminándose la luz de "Servicio dispuesto". Cuando se accione el pulsador de "Servicio OFF" (E 0.1), la escalera se parará. Cuando se accione el pulsador de emergencia (E0.4), la escalera se parará, y para que vuelva a ser operativa habrá que pulsar el rearme (E0.5).

Cuando la fotocélula de abajo (E0.2) detecte que ha llegado una persona, se activa el motor de la escalera (A4.0) durante 5 seg.

Cuando se hayan producido 5 saltos de relé térmico (E0.3) –contacto normalmente cerrado–, la escalera no podrá rearmarse, hasta que no se accione el rearme (E0.5)

24.5. Cintas transportadoras



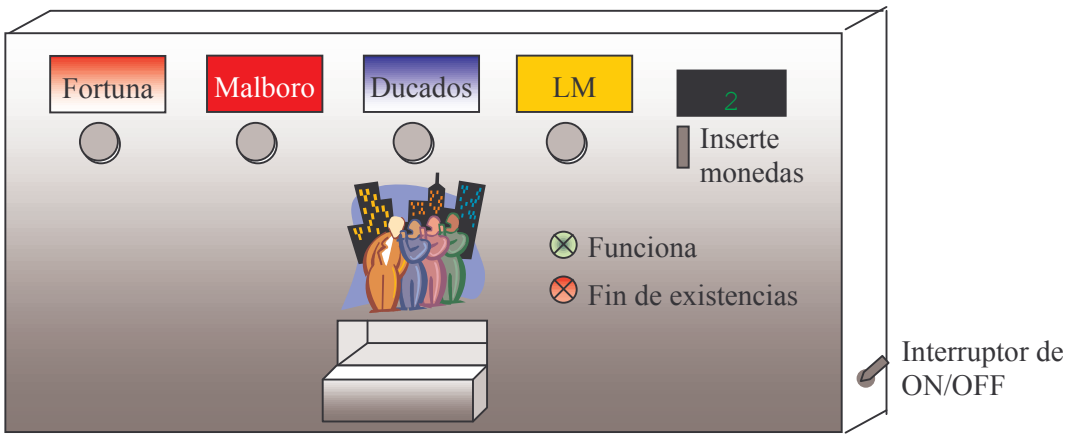
Funcionamiento: Es un automatismo para seleccionar cajas por su tamaño. La cinta transportadora A4.0 está activa esperando una caja. Cuando se activa el sensor E0.0 la cinta A4.1 se pone en marcha, y la cinta A4.0 se detiene. Para seleccionar la caja hay dos sensores: E0.1 y E0.2, este último también indica que la caja sale de la cinta.

- Si la caja es grande se activarían los 2 sensores a la vez, por lo que la caja se desplazaría por la cinta transportadora A5.0 que funcionaría durante 5 seg.
- Si la caja es pequeña nunca se activarían los dos sensores a la vez, y pasaría a la cinta para cajas pequeñas A5.1 que funcionaría 4 seg.

Una vez que la caja salga de la cinta intermedia A4.1 podremos procesar otra caja activando A4.0.

Las cajas van llenando dos contenedores. Como opción, se podrá poner dos contadores que cuenten las cajas grandes y pequeñas, y que cuando lleguen a 7 cajas grandes y 5 pequeñas, se detenga el proceso hasta que el operario sustituya los contenedores por otros vacíos y le dé a un pulsador de acuse de recibo.

24.6. Máquina expendedora de tabaco



El programa consiste en un máquina expendedora de paquetes de tabaco. Dispone de cuatro marcas con los siguientes precios:

	Precio Euros
Fortuna	3
Malboro	3,5
Ducados	2
LM	2,8

La introducción de las monedas la simulamos con pulsos de las siguientes entradas del autómata:

E 0.0	2 Euros
E 0.1	1 Euro
E 0.2	50 ctms.

La máquina no devuelve monedas, si hemos metido más cantidad de dinero que el precio exacto, lo perderemos con la entrega del paquete.

Tenemos un interruptor de activación la maquina, que siempre debe estar "ON": ***E1.0***

Además necesitamos reponer los paquetes de tabaco de cada marca, lo hacemos con las entradas: ***E1.1*** (Fortuna), ***E1.2*** (Malboro), ***E 1.3*** (Ducados) y ***E1.4*** (LM).

Los pulsadores que habrá que activar para que se produzca la entrega del paquete serán: ***E0.3*** (Fortuna), ***E0.4*** (Malboro), ***E 0.5*** (Ducados) y ***E0.6*** (LM).

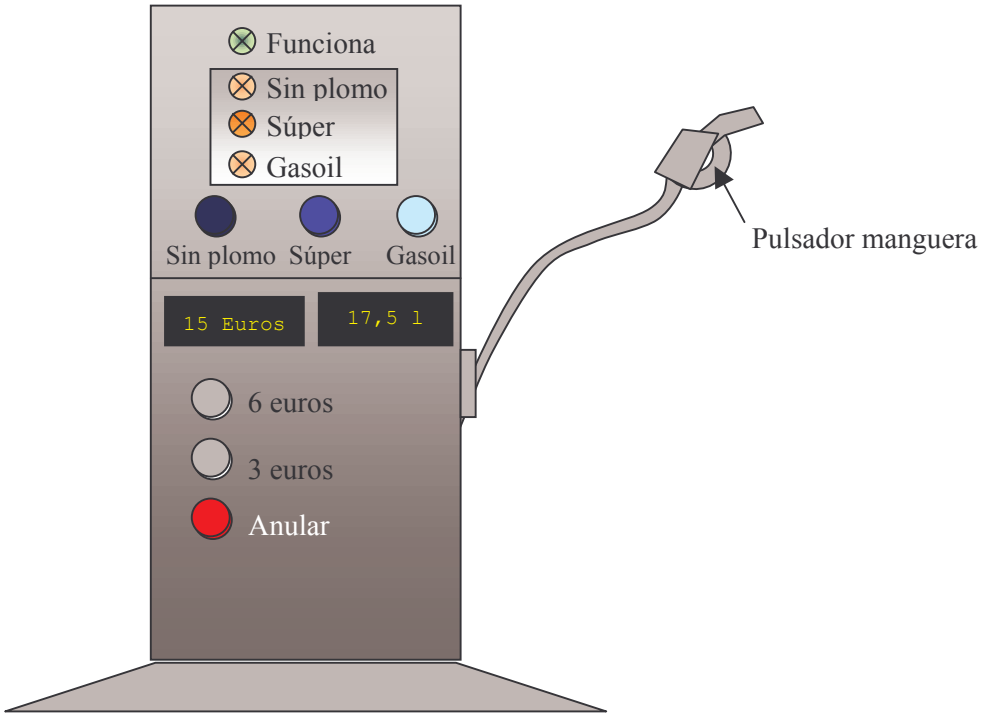
Como salidas tendremos:

A4.0 : Luz que indica funcionamiento de la máquina (cuando la máquina está activa).

A4.1 : Luz que indica fin de existencias, se iluminará durante un cierto tiempo cuando al darle a uno de los pulsadores de entrega no exista esa marca en concreto.

A4.2 : Luz que simula que la entrega de tabaco se ha producido (con un destello es suficiente).

24.7. Gasolinera



Simularemos el funcionamiento de un surtidor de gasolina. Introduciremos la cantidad de dinero y seleccionaremos el tipo de gasolina que queramos cuyo precio por litro es el siguiente:

	Precio/litro
Sin plomo	1 euro
Súper	1,2 euros
Gasoil	0,88 euros

Cuando usemos la manguera, que simularemos como pulsador, nos saldrán *X* litros de gasolina a razón de 1 litro por segundo (la salida de gasolina la simulamos temporizando una salida del autómata).

Botón 6 euros	E0.0	***Luz funciona***	A4.0
Botón 3 euros	E0.1	***Luz sin plomo***	A5.0
Anular	E0.2	***Luz súper***	A5.1
Pulsador sin plomo	E0.3	***Luz gasoil***	A5.2
Pulsador súper	E0.4	***Salida gasolina***	A4.1
Pulsador gasoil	E0.5		
Pulsador manguera	E1.0		

25. Soluciones ejercicios propuestos

25.1. Ejercicio operaciones lógicas

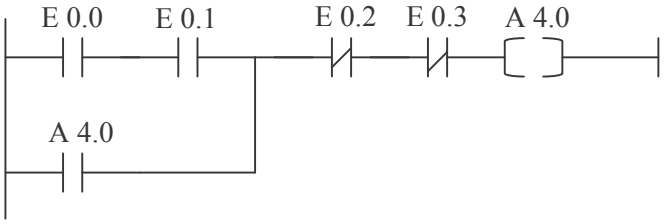
```
U(
U   E0.0
U   E0.1
U   E0.2
O
U   E0.3
U   E0.4
O   E0.5
)

U(
U   E0.6
U(
U   E1.0
U   E1.1
O   E1.2
)
O   E0.7
)

U(
O   E1.3
O   E1.4
)

=   A4.0
```

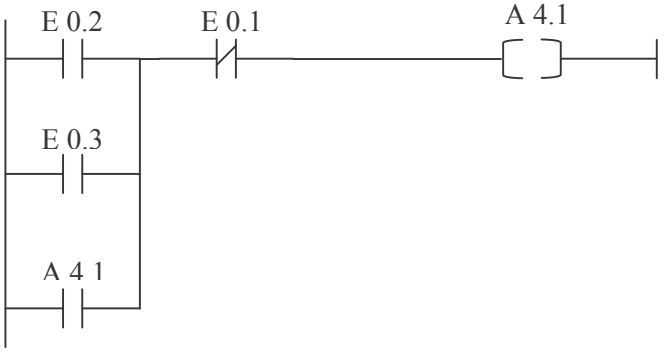
25.2. Ejercicio taladradora

Lista de instrucciones (AWL):

```
//Bajada de taladradora:

U(
U     E 0.0       //(Si "marcha" y
U     E 0.1       //final de carrera arriba pisado
O      A 4.0      //O el motor ya está bajando)
)
UN     E 0.2      //Y además no está pulsado el final de carrera de abajo
UN     E 0.3      //y no está dado el paro
=      A 4.0      //Activo el motor de bajada

//Subida de taladradora:

U(
O      E 0.2      //Si está pisado el Final de Carrera abajo
O      E 0.3      //O está dado el paro
O      A 4.1      //O ya está subiendo
)
UN    E 0.1       //Y además no está pisado el final de carrera de arriba
=     A 4.1       //Activo el motor de subida
```

Diagrama de contactos :



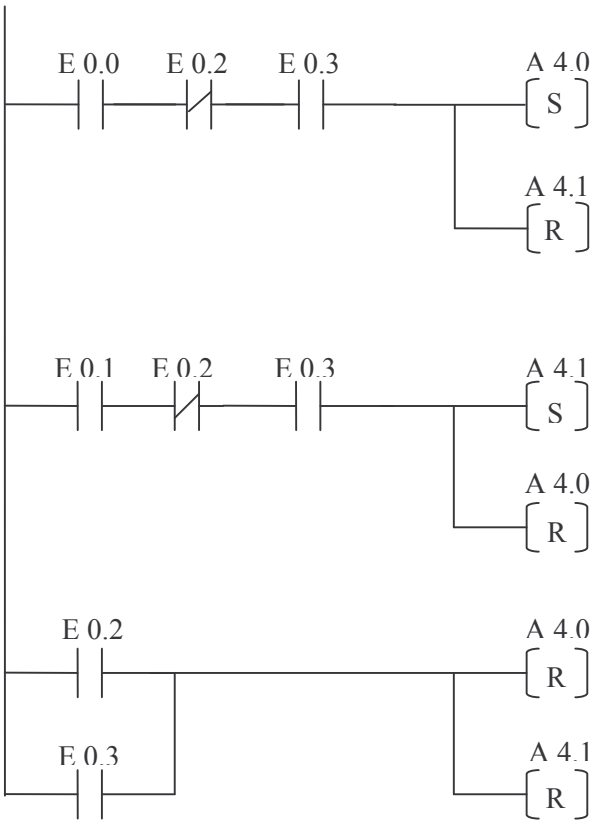


25.3. Ejercicio motor

Programa AWL:

```
//Giro derechas:

      U     E 0.0     //(Si se ha dado derechas
      UN    E 0.2     //y no está dado el paro
      U     E 0.3     //y el relé está cerrado)
      S     A 4.0     //activar derechas
      R     A 4.1     //desactivar izquierdas

//Giro izquierdas:

      U     E 0.1     //(Si se ha dado derechas
      UN    E 0.2     //y no está dado el paro
      U     E 0.3     //y el relé esta cerrado)
      S     A 4.1     //activar izquierdas
      R     A 4.0     //desactivar derechas

//Relé térmico o paro:

      U     E 0.2     //(Si está dado el paro
      ON    E 0.3     //o el relé se abre)
      R     A 4.0     //desactivo derechas
      R     A 4.1     //desactivo izquierdas
```

Programa KOP:



25.4. Ejercicio de control de un semáforo

Resuelto con temporizadores SV :

```
UN    A 4.0    //Si no está en rojo
UN    A 4.1    //ni en amarillo
=     A 4.2    //entonces verde

U     E 0.0    //Si pulso
U     A 4.2    //y está en verde
L     S5T#3S   //temporizo 3 seg.
SV    T0

U     T0      //mientras se temporiza los 3 seg.
=     A 4.1   //se enciende la luz amarilla
U     A 4.1   //Cuando luz amarilla off después de 3 seg.
FN    M 100.1
L     S5T#6S
SV    T1       //temporizo 6 seg.

U     T1     //mientras se temporizan los 6 seg
=     A 4.0  //activo la luz de rojo

U     A 4.0  //Si está en rojo
=     A 5.1  //verde para peatones

UN    A 4.1  //Si no está en rojo
=     A 5.0  //rojo para peatones
```

Resuelto con temporizadores SE :

```
UN    A 4.0  //Si no está en rojo
UN    A 4.1  //ni en amarillo
=     A 4.2  //entonces verde

U     E 0.0  //Si pulso
U     A 4.2  //y está en verde
S     A 4.1  //activo amarillo

U     A 4.1   //Si está amarillo
L     S5T#3S  //temporizo 3 seg.
SE    T0

U     T0     //cuando se cumplan los 3 seg de T0
R     A 4.1 //apago amarillo
S     A 4.0 //enciendo rojo

U     A 4.0   //Si está rojo
L     S5T#6S  //temporizo 6 seg.
SE    T1

U     T1      //cuando se cumplan los 6 seg de T1
R     A 4.0   //apago rojo

U    A 4.0  //Si está en rojo
=    A 5.1  //verde para peatones

UN   A 4.1  //Si no está en rojo
=    A 5.0  //rojo para peatones
```

25.5. Ejercicios generador de pulsos y onda cuadrada

Generador de pulsos:

```
UN    M0.0
L     S5T#1s
SE    T1
U     T1
=     M0.0
=     A4.0
```

Generador de onda cuadrada:

```
UN    M0.0
L     S5T#1s
SE    T1
U     T1
=     M0.0
UN    M0.0
BEB
UN    A4.0
=     A4.0
```

25.6. Ejercicio área de almacenamiento

```
U   E 0.0     //Cada impulso generado por la barrera fotoeléctrica 1 aumenta
ZV  Z1        //el valor del contador Z1 en una unidad, contando así el número
              //de paquetes transportados al área de almacenamiento.

U   E 0.1     //Cada impulso generado por la barrera fotoeléctrica 2 disminuye
ZR  Z1        //el valor del contador Z1 en una unidad, contando así los
              // paquetes que salen del área de almacenamiento.

UN  Z1        //Si el valor de contaje es 0, se enciende la lámpara indicadora
=   A 4.0     //"Área de almacenamiento vacía".

U   Z1        //Si el valor de contaje no es 0, se enciende la lámpara
=   A 4.1     // indicadora "Área de almacenamiento no vacía".

L   +5        //Si el valor de contaje es menor o igual a 5, se enciende la
L   Z1        //lámpara indicadora "Área de almacenamiento al 50%".
<=I
=   A 4.2

L   +9        //Si el valor de contaje es mayor o igual a 9, se enciende la
>=I           //lámpara indicadora "Área de almacenamiento al 90%".
=   A 4.3

L   Z1        //Si el valor de contaje es mayor o igual a 10, se enciende la
L   10        //lámpara indicadora "Área de almacenamiento llena". (También se
>=I           //puede utilizar la salida A 4.4 para bloquear la cinta
=   A 4.4     //transportadora 1.
```

25.7. Ejercicio factorial de un número

1)

```
      L     1       //Inicializo
      T     MD100   //y lo guardo
      L     7       //Cargo para el primer bucle

FACT: T     MW104   //Guardo el valor del contador del bucle (=7 para el primer bucle)

      L     MD100   //Cargo el valor de la multiplicación (=1 en el primer bucle)
      *I            //Multiplico (en el primer bucle = 1 de ACU1 x 7 de ACU2)
      T     MD100   //Guardo el valor de la multiplicación
      L     MW104   //Cargo el valor de contador del bucle en ACU1
      LOOP  FACT    //Decremento en 1 en ACU1 y salto a meta FACT si <>0
```

2)

```
      L     7       //Inicializo
      T     MD100   //y lo guardo
      L     6       //Cargo para el primer bucle

FACT: T     MW104   //Guardo el valor del contador del bucle (=6 para el primer bucle)
      L     MD100   //Cargo el valor de la multiplicación (=7 en el primer bucle)
      *I            //Multiplico (en el primer bucle = 7 de ACU1 x 6 de ACU2)
      T     MD100   //Guardo el valor de la multiplicación
      L     MW104   //Cargo el valor de contador del bucle en ACU1
      LOOP  FACT    //Decremento en 1 en ACU1 y salto a meta FACT si <>0
```

25.8. Ejercicio multiplicación y división

```
      U    E0.5
      FP   M10.0
      SPB  CARG          //Salto a carga
      U    E0.0
      FP   M10.1
      SPB  MUL           //Salto a multiplicación
      U    E0.1
      FP   M10.2
      SPB  DIV           //Salto a división
      BEA                //Fin de módulo

CARG: L    EB1
      T    MW100         //Cargo el valor en una palab. de marca donde iré
      BEA                  guardando el resultado

 MUL: L    255
      L    MW100
      <=I                //Comparo resultado con 255
      SPB  fin           //si es mayor que 255, salto
      L    2             //si es menor, continúo aquí,
      *I                 //y multiplico por 2
      T    MW100         //Transfiero el resultado
      BEA

 DIV: L    1             //Comparo resultado con 1
      L    MW100         //si es igual a 1, salto
      >=I                //si es mayor, continúo aquí,
      SPB  fin           //y divido entre 2
      L    2             //Transfiero el resultado
      /I
      T    MW100
      BEA

fin:  BEA
```

25.9. Ejercicio desplazamiento

OB 100

```
      L   1
      T   AB5
```

OB1

```
      UN   M30.0
      L    S5T#250MS
      SE   T0
      U    T0
      =    M30.0

      U    A5.0
      R    M20.0
      U    A4.7
      S    M20.0

      U    M20.0
      U    M30.0
      SPBN IZQ
      L    AW4
      SRW  1
      T    AW4
      BEA

IZQ:  UN   M20.0
      U    M30.0
      SPBN FIN
      L    AW4
      SLW  1
      T    AW4

FIN:  BE
```

25.10. Ejercicio cintas transportadoras

Programa:

```
//Activación de las cintas:

UN    A4.1  //Si no está activada la cinta intermedia
=     A4.0  //que se activa la primera cinta
U     E0.0  //Si sensor de llegada de caja a la cinta intermedia
S     A4.1  //activar la cinta intermedia

//Selección por tamaño:

U     E0.1
U     E0.2  //Si se activan los dos sensores
S     M0.0  //Es que es una caja grande, lo guardo en una marca

U     E0.2  //Cuando se activa el sensor de salida de la cinta
FN    M0.1  //intermedia por flanco positivo
=     M0.2  //Lo guardo en una marca

//Según el tamaño activo la cinta de salida:

U     M0.2        //Cuando sale la caja
U     M0.0        //y es grande
L     S5T#5S
SV    T1
U     T1
=     A5.0        //activo la cinta para grandes durante 5 seg.

U     M0.2        //Cuando sale la caja
UN    M0.0        //y no es grande
L     S5T#4S
SV    T2
U     T2
=     A5.1        //activo la cinta para pequeñas durante 4 seg

U     M0.2        //Cuando sale la caja
R     A4.1        //Desactivo la cinta intermedia
R     M0.0        //Reseteo la marca que me indica el tamaño
```

25.11. Ejercicio detector de humo

```
CALL  "SCALE"
   IN     :=PEW128
   HI_LIM :=10.0
   LO_LIM :=0.0
   BIPOLAR:=FALSE
   RET_VAL:=MW100
   OUT    :=MD102

L     MD102
L     5.0
>=R
=     A124.0

L     PEW128
T     PAW128
```

25.12. Ejercicio contador analógico

```
U     E124.0
ZV    Z0

L     Z0
ITD
DTR
T     MD100

L     Z0
L     28
==I
R     Z0

CALL  "UNSCALE"
   IN     :=MD100
   HI_LIM :=27.0
   LO_LIM :=0.0
   BIPOLAR:=FALSE
   RET_VAL:=MW104
   OUT    :=PAW128
```

25.13. Ejercicio onda diente de sierra

```
UN    T0
L     S5T#4S
SE    T0

L     T0
ITD
DTR
```

```
T     MD100

CALL  "UNSCALE"
   IN     :=MD100
   HI_LIM :=40.0
   LO_LIM :=0.0
   BIPOLAR:=FALSE
   RET_VAL:=MW103
   OUT    :=PAW128
```

25.14. Ejercicio repostería

1) Usando FCs

OB1

```
UC    FC4  //También se puede poner: CALL FC4
U     E0.0
ZV    Z0

L     Z0
L     0
==I
CC    FC1

L     Z0
L     1
==I
CC    FC2

L     Z0
L     2
==I
CC    FC3

L     Z0
L     3
>=I
R     Z0
```

FC1

```
L     S5T#5S
T     MW10

L     S5T#2S
T     MW12

L     S5T#4S
T     MW14
```

FC2

```
L     S5T#2S
T     MW10

L     S5T#3S
T     MW12

L     S5T#5S
T     MW14
```

FC3

```
L     S5T#3S
T     MW10

L     S5T#4S
T     MW12

L     S5T#2S
T     MW14
```

FC4

```
U     E0.1
L     MW10
SV    T1

L     MW12
SV    T2
```

```
L     MW14
SV    T3

U     T1
=     A4.0

U     T2
=     A4.1

U     T3
=     A4.2
```

2) Usando un DB Global

OB1

```
UC    FC4  //También se puede poner: CALL FC4

U     E4.0
ZV    Z0

L     Z0
L     0
==I
CC    FC1

L     Z0
L     1
==I
CC    FC2

L     Z0
L     2
==I
CC    FC3

L     Z0
L   3
>=I
R     Z0
```

FC1

```
L     "Tiempos".Magdalenas.Leche// L     DB1.DBW    0
T     MW10

L     "Tiempos".Magdalenas.Harina     // L     DB1.DBW    2
T     MW12

L     "Tiempos".Magdalenas.Agitador   // L     DB1.DBW    4
T     MW14
```

FC2

```
L     "Tiempos".Bizcochos.Leche  // L     DB1.DBW    6
T     MW10

L     "Tiempos".Bizcochos.Harina// L     DB1.DBW    8
T     MW12

L     "Tiempos".Bizcochos.Agitador    // L     DB1.DBW   10
T     MW14
```

FC3

```
L     "Tiempos".Rosquillas.Leche// L     DB1.DBW   12
T     MW10

L     "Tiempos".Rosquillas.Harina     // L     DB1.DBW   14
T     MW12

L     "Tiempos".Rosquillas.Agitador   // L     DB1.DBW   16
T     MW14
```

FC4

```
U     E124.1
L     MW10
SV    T1

L     MW12
SV    T2

L     MW14
SV    T3

U     T1
=     A4.0

U     T2
=     A4.1

U     T3
=     A4.2
```

- Hay que crear un DB Global:

Crear el DB global e introducir su estructura (Ver > Declaración) :

Dirección	Nombre	Tipo	Valor inicial	Comentario
0.0		STRUCT		
+0.0	Magdalenas	STRUCT		
+0.0	Leche	S5TIME	S5T#0MS	
+2.0	Harina	S5TIME	S5T#0MS	
+4.0	Agitador	S5TIME	S5T#0MS	
=6.0		END_STRUCT		
+6.0	Bizcochos	STRUCT		
+0.0	Leche	S5TIME	S5T#0MS	
+2.0	Harina	S5TIME	S5T#0MS	
+4.0	Agitador	S5TIME	S5T#0MS	
=6.0		END_STRUCT		
+12.0	Rosquillas	STRUCT		
+0.0	Leche	S5TIME	S5T#0MS	
+2.0	Harina	S5TIME	S5T#0MS	
+4.0	Agitador	S5TIME	S5T#0MS	
=6.0		END_STRUCT		
=18.0		END_STRUCT		

STRUCT: Tipo de datos compuestos cuyos elementos pueden ser simples o compuestos. Se pueden anidar hasta 8 niveles. El tipo de datos STRUCT debe comprender uno o varios componentes que se encuentren entre STRUCT y END_STRUCT. Una estructura dentro de una estructura es considerada como un solo componente.

Definir los valores actuales : (Ver > Datos)

3) **Usando FBs con DBs de instancia**

OB1

```
U     E0.0
ZV    Z0

L     Z0
L     0
==I
SPB   magd

L     Z0
L     1
==I
SPB   bizc

L     Z0
L     2
==I
SPB   rosq

L     Z0
L     3
>=I
R     Z0

BEA


magd: CALL  FB1 , DB1
      BEA


bizc: CALL  FB1 , DB2
      BEA


rosq: CALL  FB1 , DB3
      BEA
```

FB1

```
U     E0.1
L     #leche
SV    T1
```

```
L     #harina
SV    T2

L     #agitador
SV    T3

U     T1
=     A4.0

U     T2
=     A4.1

U     T3
=     A4.2
```

DB1

Dirección	Declaración	Nombre	Tipo	Valor inicial	Valor actual	Comentario
0.0	stat	leche	S5TIME	S5T#0MS	S5T#5S	
2.0	stat	harina	S5TIME	S5T#0MS	S5T#2S	
4.0	stat	agitador	S5TIME	S5T#0MS	S5T#4S	

DB2

Dirección	Declaración	Nombre	Tipo	Valor inicial	Valor actual	Comentario
0.0	stat	leche	S5TIME	S5T#0MS	S5T#2S	
2.0	stat	harina	S5TIME	S5T#0MS	S5T#3S	
4.0	stat	agitador	S5TIME	S5T#0MS	S5T#5S	

DB3

Dirección	Declaración	Nombre	Tipo	Valor inicial	Valor actual	Comentario
0.0	stat	leche	S5TIME	S5T#0MS	S5T#3S	
2.0	stat	harina	S5TIME	S5T#0MS	S5T#4S	
4.0	stat	agitador	S5TIME	S5T#0MS	S5T#2S	

4) **Usando FBs con DB de multiinstancia**

OB1

CALL FB1, DB1

FB1

Dirección	Declaración	Nombre	Tipo	Valor inicial	Comentario
	in				
	out				
	in_out				
0.0	stat	magdalenas	FB2		
6.0	stat	bizcochos	FB2		
12.0	stat	rosquillas	FB2		
	temp				

```
U     E0.0
ZV    Z0

L     Z0
L     0
==I
SPB   magd

L     Z0
L     1
==I
SPB   bizc

L     Z0
L     2
==I
SPB   rosq

L     Z0
L     3
>=I
R     Z0

BEA


magd: CALL  #magdalenas
       leche   :=S5T#5S
       harina  :=S5T#2S
       agitador:=S5T#4S
BEA
```

```
bizc: CALL  #bizcochos
       leche   :=S5T#2S
       harina  :=S5T#3S
       agitador:=S5T#5S
BEA


rosq: CALL  #rosquillas
       leche   :=S5T#3S
       harina  :=S5T#4S
       agitador:=S5T#2S
BEA
```

FB2

Dirección	Declaración	Nombre	Tipo	Valor inicial	Comentario
0.0	in	leche	S5TIME	S5T#0MS	
2.0	in	harina	S5TIME	S5T#0MS	
4.0	in	agitador	S5TIME	S5T#0MS	
	out				

```
U     E0.1
L     #leche
SV    T1

L     #harina
SV    T2

L     #agitador
SV    T3


U     T1
=     A4.0

U     T2
=     A4.1

U     T3
=     A4.2
```

25.15. Sistema de adquisición de datos

Crear el bloque de datos global DB1 e introducir las variables tipo Word:

Dirección	Nombre	Tipo	Valor inicial
0.0		STRUCT	
+0.0	dato1	WORD	W#16#0
+2.0	dato2	WORD	W#16#0
+4.0	dato3	WORD	W#16#0
+6.0	dato4	WORD	W#16#0
+8.0	dato5	WORD	W#16#0
+10.0	dato6	WORD	W#16#0
=12.0		END_STRUCT	

OB1:

```
U   E 0.0
FP  M123.6
CC  FC1
```

Con direccionamiento interárea:

FC1:

```
      LAR1  P#0.0             //Inicializo AR1 con P#0.0
      L     6
bucl: T     MW50
      L     MW [AR1,P#0.0]
      AUF   DB1
      T     DBW [AR1,P#0.0]
      L     P#2.0             //Incremento puntero en 2 bytes=1 palabra
      +AR1                    //ACU1+AR1 -> AR1
      L     MW50
      LOOP  bucl
```

Con direccionamiento por memoria:

```
      L     P#0.0
      T     MD20
      L     6
bucl: T     MW50
      L     MW [MD20]
      AUF   DB1
      T     DBW [MD20]
      L     MD20
      L     P#2.0
      +D
      T     MD20
      L     MW50
      LOOP  bucl
```

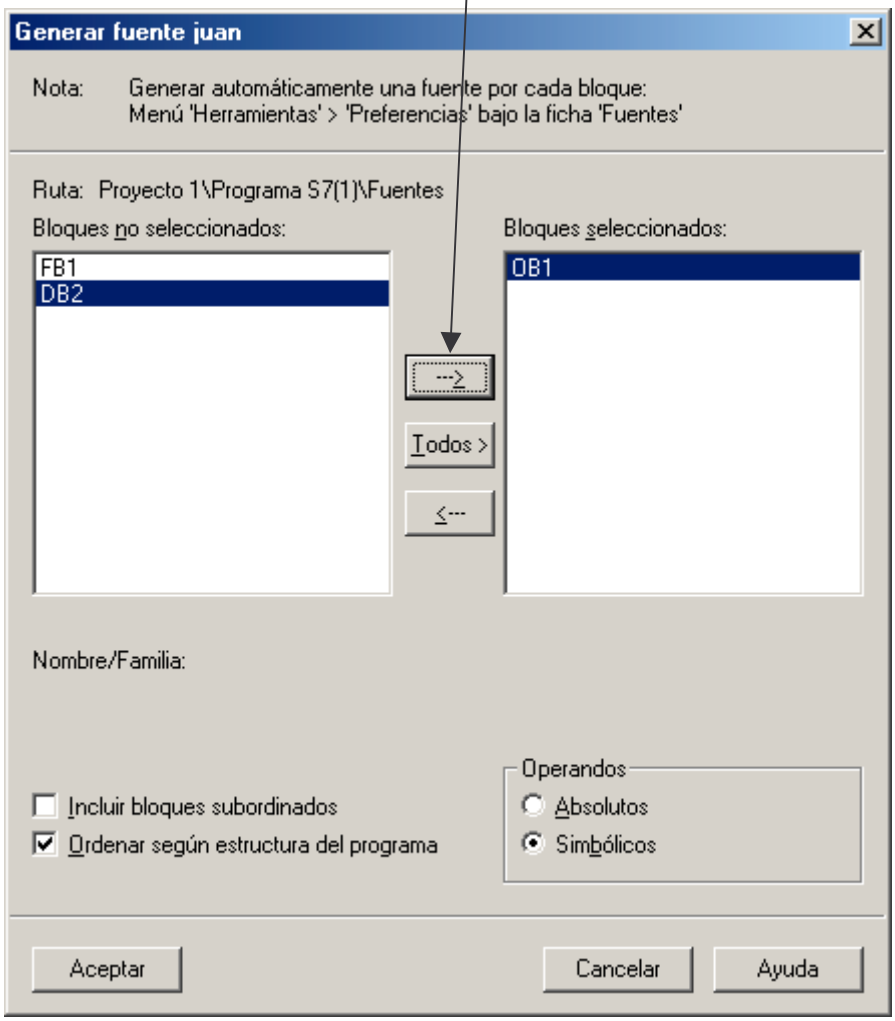
PARTE 2

Configuración y Análisis

26. Protección de bloques (Know-How-Protect)

Se puede proteger un bloque para que no se pueda acceder al código de programa.

1. Abrir el bloque que se quiere proteger.
2. Generar la fuente: “Archivo” → “Generar fuente”
3. Se le da un nombre a la fuente.
4. En la ventana que se abre, seleccionar con los bloques con los que quiere generarse la fuente:



5. La fuente queda depositada en la carpeta “Fuentes” del programa S7.

6. Abrir la fuente y colocar en la parte de declaración del módulo "**KNOW_HOW_PROTECTION**":

```
ORGANIZATION_BLOCK OB 1
TITLE = "Main Program Sweep (Cycle)"
VERSION : 0.1
KNOW_HOW_PROTECTION
```

7. Guardar y compilar la fuente ("Archivo" → "Compilar").

8. Se habrá generado un bloque protegido contra lectura y modificación.