Programación en Simatic S7

Tema 1 Objetivos

Este documento no contiene todas las instrucciones del autómata , se debe tomar como una guía complementaria. El manual proporciona más ejemplos y mayor detalle que el expuesto aquí.

Tema 2 Introducción

2.1. 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.

2.2. 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.

Tema 3 Estructura interna del autómata

3.1. 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 suele colocar datos de 1 bit, aunque pueden manejarse en modo bit, byte, word, doble word 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, este almacenaje o memoria de los contadores o temporizadores se puede seleccionar cuando configuramos las pantallas de la CPU al programar el hardware con **HW ,** consiguiendo que los contadores o temporizadores que queramos sigan manteniendo su contenido aunque apagemos el equipo.

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.

3.2. Tipos de módulos

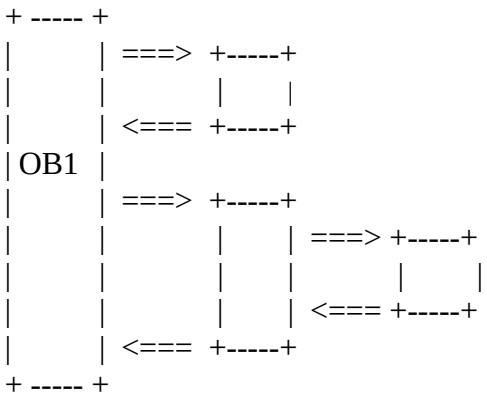
El Simatic S7 dispone de una serie de módulos que dividen la memoria de programa y la de datos en secciones, permitiendo una programación estructurada y un acceso ordenado a los datos.

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. Tiene un tiempo de ejecución en el cual el programa que ejecuta a de llegar al final, sino pasara a STOP la CPU, indicando error, ese tiempo se puede variar al configurar las pantallas de la CPU cuando se programa el hardware con **HW.** Por defecto tiene colocado 150msg.



- 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 3.5).
- 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.

En total podemos manejar hasta 128 módulos de código.

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.

En total podemos manejar hasta 128 módulos de funciones.

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.

En total podemos manejar hasta 127 módulos de datos.

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.

3.3. 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).

3.4. 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

3.5. 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.

3.6. 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
```

3.7. 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).

3.8. 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.

- 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.

TEMA 4: PROGRACIÓN EN AWL

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).

Tema 5 Operaciones lógicas con bits

5.1. 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
```

5.2. AND

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 E0.0

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

5.3. OR

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
```

5.4. XOR (O exclusiva)

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
```

5.5. 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
```

5.6. 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
```

5.7. 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: 0 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: 0 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.

5.8. 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.

5.9. 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.

Tema 6 Operaciones de contaje

6.1. 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.

6.2. 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
```

6.3. 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)
```

6.4. 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
```

6.5. 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
```

6.6. 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
```

Tema 7 Operaciones de temporización

7.1. 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.

7.2. 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
```

7.3. 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.

7.4. Temporizador como impulso

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
```

7.5. Temporizador como impulso prolongado

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
```

7.6. Temporizador como retardo a la conexión

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
```

7.7. Temporizador como retardo a la conexión con memoria

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 sali

```
                                  a del temporizador
=  A  3.1     //Asignamos la salida del temporizador a la salida
              3.1
```

7.8. Temporizador como retardo a la desconexión

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
```

7.9. 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
```

7.10. 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.

7.11. 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
```

Tema 8 Operaciones de salto

8.1. 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.

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

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
```

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
```

8.2. Operaciones de salto condicional, en función del RLO

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

Estas instrucciones efectuan 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.

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

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

Estas instrucciones efectuan 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
```

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

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

Estas instrucciones efectuan 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	0	mayor o igual que 0	SPPZ

o 1	o 0		
0 o 0	0 o 1	menor o igual que 0	SPMZ
1	1	UO (no admisible)	SPU

8.5. 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.

Registros afectados: ACU 1

Ejemplo:

```
        L      +5      //Hacemos el ACU 1 igual a 5
PROX:   T   MB 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   MB 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.

Tema 9 Operaciones de control de programa

9.1. 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.

9.2. 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]	0 a 65534	

	[DIW] [LW] [MW]		

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

9.3. 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 determinmada. 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.

9.4. Funciones 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)

Estado de señal del MCR	**=**	**S o R**	**T**
0	Escribe 0	No escribe	Escribe 0
1	Ejecución normal	Ejecución normal	Ejecución normal

MCRA activa el área MCR.

MCRD desactiva el área MCR.

Las operaciones programadas entre las operaciones MCRA y MCRD dependen del estado de señal del bit MCR. Si falta la operación MCRD las operaciones entre MCRA y una operación BEA dependerán del bit MCR.

Cuando se llama a una función (FC) o a un módulo de función (FB) desde un área MCR las instrucciones ejecutadas no dependerán del MCR, a menos que volvamos a especificarlo con MCRA en el módulo llamado. Una vez se finalice el módulo y se retorne se continuará en área MCR.

Hay que recalcar que las instrucciones dentro del área MCR dependen del bit MCR, pero dicho bit no es 0 (MCR activo) mediante las instrucciones MCRA y MCRD.

Registros afectados: ninguno

MCR(salva el RLO en la pila MCR si es el primer nivel de anidamiento. Para un anidamiento mayor se multiplica (AND lógico) el MCR actual con el RLO, siendo el resultado el nuevo bit MCR (ver ejemplo). El máximo anidamiento es 8.

)MCR finaliza un nivel de anidamiento, tomando como nuevo MCR el que hubiera en la pila MCR almacenado. Hay que destacar que las instrucciones)MCR deben ser en igual cuantía que las MCR((deben ir por pares para no perder datos).

Cuando hablamos de anidamiento nos referimos al proceso mediante el cual una pila sube de nivel, almacenando ciertos datos en ella referidos al nivel anterior para luego recuperarlos.

Registros afectados: pila MCR, bit MCR.

Ejemplo:

```
MCRA             //comenzamos área MCR, suponemos 1 el bit MCR
U   E   2.0      //introducimos en el RLO el valor de la entrada
                  2.0
MCR(             //comenzamos anidamiento en pila MCR, si E2.0=0
                 el MCR=0
O   E   1.0      //copiamos el valor de la entrada 1.0 en el RLO
                            s área MCR, suponemos 1 el bit MCR
U   E   1.0      //introducimos en el RLO el valor de la entrada
                  1.0
MCR(             //comenzamos anidamiento en pila MCR, si E1.0=0
```

```
                  el MCR=0
U   E   2.0       //copiamos el valor de la entrada 2.0 en el RLO
MCR(              //nuevo anidamiento en pila MCR, si E2.0=0 el
                  MCR=0, en caso contrario no varia el bit MCR
O   E   3.0       //copiamos el valor de la entrada 3.0 en el RLO
=   A   4.0       //si el bit MCR=0 la salida 4.0 se hace 0 sin
                  depender del RLO, en caso contrario depende del
                  valor de la entrada 3.0
)MCR              //finalizamos primer anidamiento, el bit MCR se
                  recupera de la pila
)MCR              //finalizamos anidamiento, el bit MCR vuelve a
                  ser 1
MCRD              //fin del área MCR
```

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.

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.

Tema 10 Operaciones de comparación

10.1. 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 falso. 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.

10.2. 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
```

10.3. 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
```