

STEP 7

Una manera fácil de programar PLC de Siemens

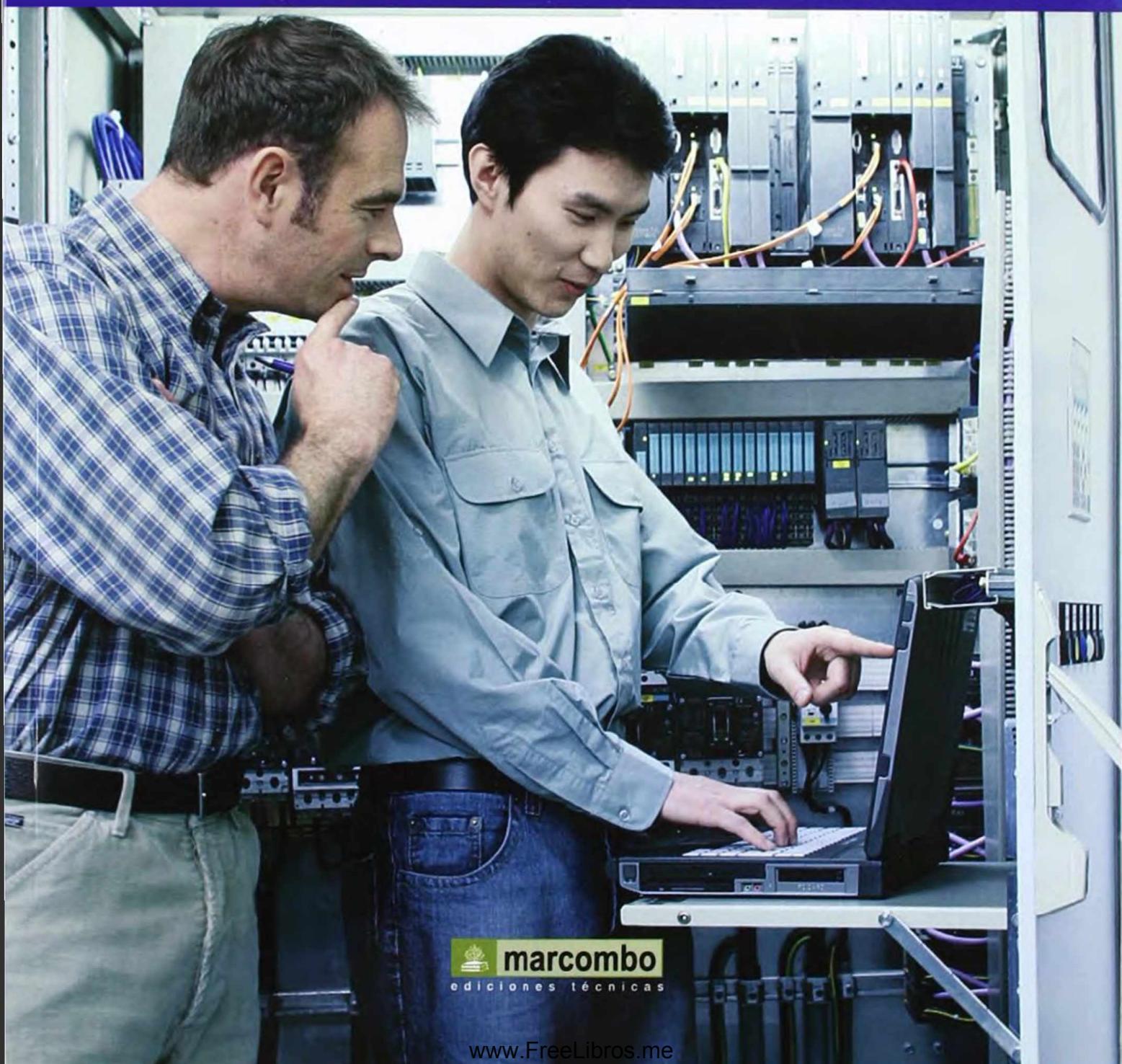
Pilar Mengual



incluye DVD

F

marcombo
FORMACIÓN



 **marcombo**
ediciones técnicas

www.FreeLibros.me

Índice general

Unidad didáctica 3

Operaciones de *byte*, palabras y dobles palabras..... 134

Ejercicios

- 3.1. Instrucciones de carga y transferencia..... 135
- 3.2. Ejercicio de metas..... 140
- 3.3. Trabajar con DB..... 144
- 3.4. Pesar productos dentro de unos límites..... 152
- 3.5. Introducción a la programación estructurada..... 155
- 3.6. Desplazamiento y rotación de bits..... 158
- 3.7. Planta de embotellado..... 162
- 3.8. FC con y sin parámetros..... 164
- 3.9. Crear un DB con la SFC 22..... 170
- 3.10. Sistemas de numeración..... 172
- 3.11. Carga codificada..... 176
- 3.12. Operaciones con enteros..... 179
- 3.13. Conversiones de formatos..... 184
- 3.14. Operaciones con reales..... 186
- 3.15. Control de un gallinero..... 190
- 3.16. Operaciones de salto..... 195
- 3.17. Mezcla de pinturas..... 197
- 3.18. Instrucciones NOT, CLR, SET y SAVE..... 199
- 3.19. Ajuste de valores analógicos..... 201
- 3.20. Ajuste de valores analógicos con funciones de librería..... 205
- 3.21. Ejemplo con UDT..... 207
- 3.22. Operaciones lógicas con palabras..... 210
- 3.23. Alarmas..... 211

Unidad didáctica 4

Operaciones de sistema..... 213

Ejercicios

- 4.1. Detección de errores..... 214
- 4.2. Relación de OB y SFC..... 224
- 4.3. Instrucción LOOP..... 230
- 4.4. Programación OB 80 (SFC 43)... 232
- 4.5. OB 100, 101, Retardo en el arranque..... 235
- 4.6. Programación de alarmas cíclicas..... 237
- 4.7. Programación de alarmas horarias por *hardware*..... 239
- 4.8. Programación de alarmas horarias por *software*..... 241
- 4.9. Programación de alarmas de retardo..... 247
- 4.10. Ajustar la hora..... 249
- 4.11. Formatos fecha y hora..... 250
- 4.12. Hacer funcionar algo un día de la semana..... 254
- 4.13. Convertir archivos de S5 a S7..... 256
- 4.14. Programar archivos fuente y protección de bloques..... 264
- 4.15. Direccionamiento indirecto..... 270
- 4.16. Control de fabricación de piezas..... 274
- 4.17. Cargar longitud y número de DB..... 277
- 4.18. Comparar dobles palabras..... 278
- 4.19. Referencias cruzadas..... 279
- 4.20. Comunicación MPI por datos globales..... 284
- 4.21. Red PROFIBUS DP. Periferia descentralizada..... 291
- 4.22. Utilización del simulador de SIEMENS..... 297
- 4.23. Realizar copias de seguridad..... 301

Prólogo

La importancia del software en la automatización es cada vez mayor, por eso resulta un gran acierto dedicar un libro expresamente al aprendizaje del lenguaje de programación de PLC más divulgado en la industria, el STEP 7 o Lenguaje de los Controladores SIMATIC.

El extraordinario éxito del PLC ha sido su evolución y su creciente aplicabilidad, desde la propia del automatismo o la regulación de magnitudes físicas hasta la seguridad de las personas o del medio ambiente. Su sencillez, versatilidad y amplia divulgación le han dado un carácter universal, pues es difícil encontrar la fábrica o la infraestructura que no utilice algún PLC. Por este motivo, la inversión de tiempo que realice el lector en su aprendizaje, que consiste básicamente en su programación, se verá muy fácilmente recompensada en su vida profesional.

Para mí es una satisfacción observar el trabajo de mi paisana valenciana Pilar Mengual. Tras su carrera de Ingeniería Industrial, destacó por su competencia durante los 5 años que duró su etapa laboral en el Centro de Formación de Siemens donde impartió cursos de Automatización, posteriormente completó su desarrollo profesional en varias empresas de ingeniería donde estuvo desarrollando proyectos desde sus fases de diseño y desarrollo hasta la de puesta en marcha. También conoce lo que significa ser usuario, pues también se ha responsabilizado del servicio técnico postventa.

Precisamente la amplia experiencia de Pilar, que aúna conocimientos teóricos y prácticos, así como su capacidad divulgativa, es lo que confiere a este libro el gran valor de la utilidad, escrita con mayúsculas. Se lanza directamente a la teoría necesaria para, a continuación, acometer su aplicación inmediata en el ejemplo o ejercicio práctico.

Quiero finalmente agradecer a la editorial Marcombo su valiosísima colaboración en la divulgación de las enseñanzas técnicas en su vertiente más práctica.

ÁLVARO ESTEVE VIVES

Director División Industry Automation

SIEMENS S.A.

Unidad 1 Automatismos eléctricos y microcontrolador Siemens LOGO!



En este capítulo:

- 1.1 Introducción
- 1.2 Comentario de ayuda
- 1.3 Automatismos eléctricos
- 1.4 Microcontrolador SIEMENS LOGO!

1.1 Introducción

El objetivo de este libro no es otro que el de hacer más sencilla la tarea de conocer el lenguaje de programación **STEP 7** a todos aquellos que en su quehacer diario van a tener que enfrentarse a estos equipos.

Siemens siempre ha tenido fama de hacer las cosas demasiado complicadas y con este libro se espera que los lectores cambien esta opinión.

El libro está pensado para todas aquellas personas que no tengan amplios conocimientos de electricidad o electrónica pero que se quieran dedicar a la programación de PLC. También les será de gran utilidad a todos aquellos que, aunque no se dediquen a la programación propiamente dicha, tengan que convivir con estos equipos en su trabajo del día a día.

Como se podrá comprobar, todos los ejercicios son aplicaciones sencillas las cuales están resueltas mediante programación en **STEP 7** para un equipo 300 en este caso (resueltos algunos de ellos en AWL, KOP y FUP). También se muestra algún ejemplo de programación con una CPU 400 para analizar las diferencias existentes con los equipos 300.

Se hace más hincapié en la programación AWL aunque se dan las explicaciones suficientes como para poder hacer los mismos ejercicios tanto en KOP como en FUP. En AWL tenemos que conocer el juego de instrucciones para poder programar. En los otros dos lenguajes, al ser gráficos, es "más sencillo" poder seleccionar del catálogo que nos ofrece el programa. No es necesario sabernos de memoria la nemotécnica de cada instrucción. Por otra parte, creemos que al terminar de leer este libro muchos lectores que en un principio piensan que es más sencillo programar en KOP o en FUP, cambiarán de opinión y preferirán el AWL. Veremos que este lenguaje de programación es más versátil. El programador es más libre a la hora de organizar los bloques, los segmentos y el orden de las instrucciones. Además, en cuanto a información se refiere, el sistema nos ofrece algo más en este lenguaje. También veremos que al final consumimos menos memoria dentro de la CPU ya que sólo programamos las instrucciones que necesitamos y no todas las que nos ofrecen los bloques gráficos. Tanto en KOP como en FUP, si alguna opción que nos ofrecen los bloques no la necesitamos, la dejaremos en blanco. A nosotros no nos supone esfuerzo adicional. Pero el programa utiliza memoria para "decir" a la CPU "esta opción no la quiero utilizar". Esto lo podremos ver si el programa hecho en KOP lo traducimos a AWL. Veremos que se crean instrucciones que "no hacen nada", pero en cambio ocupan memoria de la CPU.

Dado que lo que se pretende es ver las cosas de una manera sencilla, en cada una de las aplicaciones se tratan temas diferentes. Cada uno de los ejercicios viene dedicado a una de las posibilidades del equipo. En cada uno de ellos se utilizan una pequeña cantidad de instrucciones de manera que sean más didácticos. Están redactados en un lenguaje "coloquial" de manera que sea accesible para todo el mundo. No se requiere de grandes conocimientos previos para hacer pequeños programas en S7.

Estos ejercicios van a suponer un complemento ideal a la hora de poner en práctica lo que hayamos podido aprender de manera teórica sobre los equipos.

Recuerda . . .

Programar o trabajar con PLCs Siemens no es difícil. Además, el software Step 7 dispone de una excelente ayuda, que si la sabes utilizar, te puede resolver muchas dudas en tu tarea de programar o diagnosticar. En estas páginas, antes de introducirte en el software, tienes unas pequeñas pautas de cómo utilizar esta herramienta.

1.2 Comentario de ayuda

Ejercicio A 

1.2.1. Utilización de la ayuda STEP 7

TEORÍA: A lo largo de este manual, se irá indicando como se puede hacer uso de la ayuda del STEP 7. Si se sabe utilizar correctamente, es muy fácil de encontrar lo que se necesita en cada momento. A modo de introducción y de manera genérica, vamos a ver de dónde podemos obtener dicha ayuda.

Como en la gran mayoría de las aplicaciones para Windows, dentro del propio Administrador de SIMATIC, tenemos el menú de Ayuda. Dentro de él podemos encontrar varias cosas:

1. Introducción al STEP 7

Si vamos al menú "Ayuda -> Introducción", podemos encontrar una explicación de lo que es el STEP 7 y para que lo podemos utilizar.



Fig. 252

2. También podemos encontrar un menú de primeros pasos. Aquí se nos explica como generar un proyecto en S7 básico y enviarlo a la CPU.

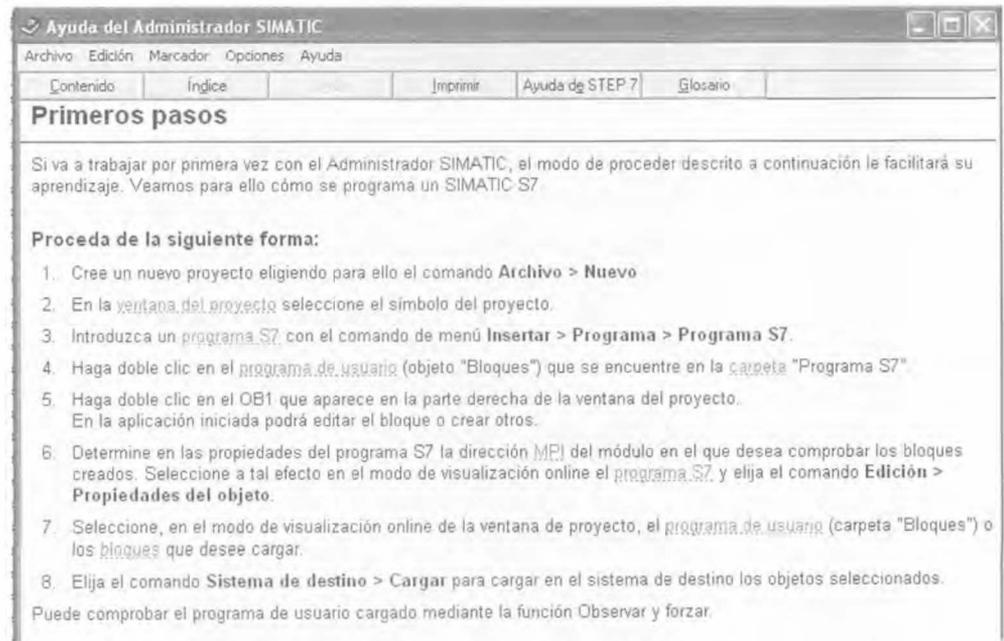


Fig. 253

3.- También podemos encontrar la ayuda completa del software si vamos a “Ayuda -> Temas de ayuda”

Aquí tenemos un índice para buscar los temas que nos interesen en cada caso.

También podemos hacer una búsqueda por temas en la pestaña de contenidos.

Si entramos en esta ayuda, veremos algo similar a esto:

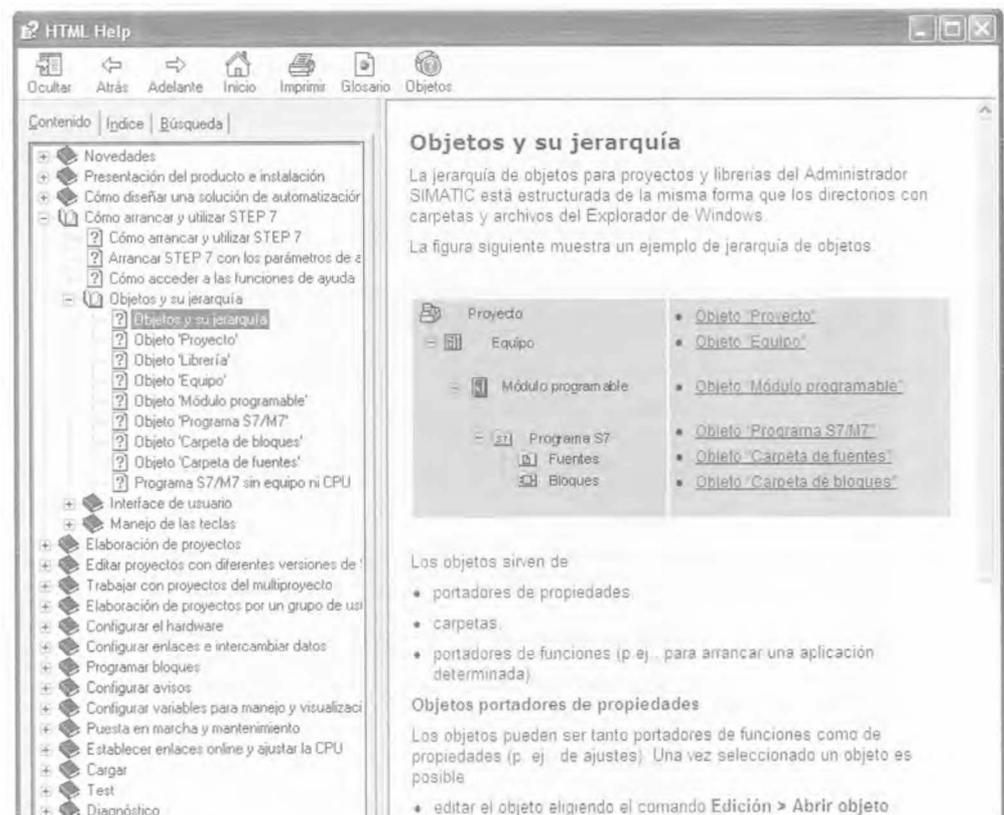


Fig. 254

Esto es lo que tenemos en cuanto a la ayuda del menú "Ayuda" del Administrador de SIMATIC. Evidentemente aquí podemos encontrar todo lo que a la ayuda del *software* se refiere. Pero además de esto, el propio *software* tiene una serie de ayudas contextuales que nos pueden ser de gran utilidad cuando estamos programando.

1.2.2. Ayudas contextuales de STEP 7

En la barra de herramientas del STEP 7 tenemos un botón con un interrogante,



Si con este botón seleccionado pinchamos sobre cualquier sitio del STEP 7, nos saldrá la ayuda del objeto seleccionado. Esto nos puede ser muy útil cuando tengamos que utilizar algún bloque ya programado de librerías o de sistema de las CPU y queramos saber cómo se utiliza y los parámetros que tenemos que rellenar.

A modo de ejemplo mostramos aquí la ayuda de una función de librería:

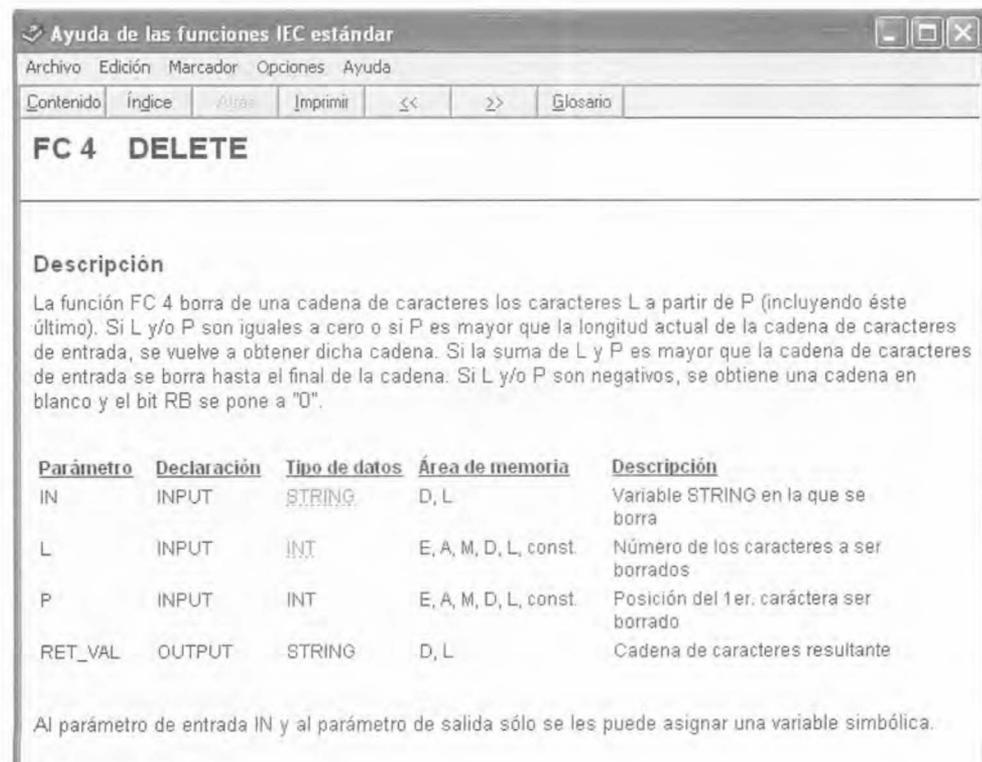


Fig. 256

También tenemos la opción de pulsar F1 cuando estemos introduciendo algún tipo de dato o nos encontremos con alguna opción concreta seleccionada. En este momento obtendremos la ayuda contextual de lo que tengamos seleccionado. Veamos un ejemplo. Tenemos desde el OB 1 una llamada a la FC 3 y nos pide un parámetro. Imaginemos que no sabemos qué tipo de formato nos pide. Si nos paramos con el puntero del ratón sobre el parámetro nos indicará el tipo de dato solicitado.

```
CALL FC 3
  VALOR_IN: =
  IN: INT
```

Fig. 257

Si ahora no sabemos cómo se escribe un entero o que valores podemos introducir como máximo o como mínimo, etc. siempre podemos pulsar en este momento F1 y obtendremos la ayuda del formato solicitado.

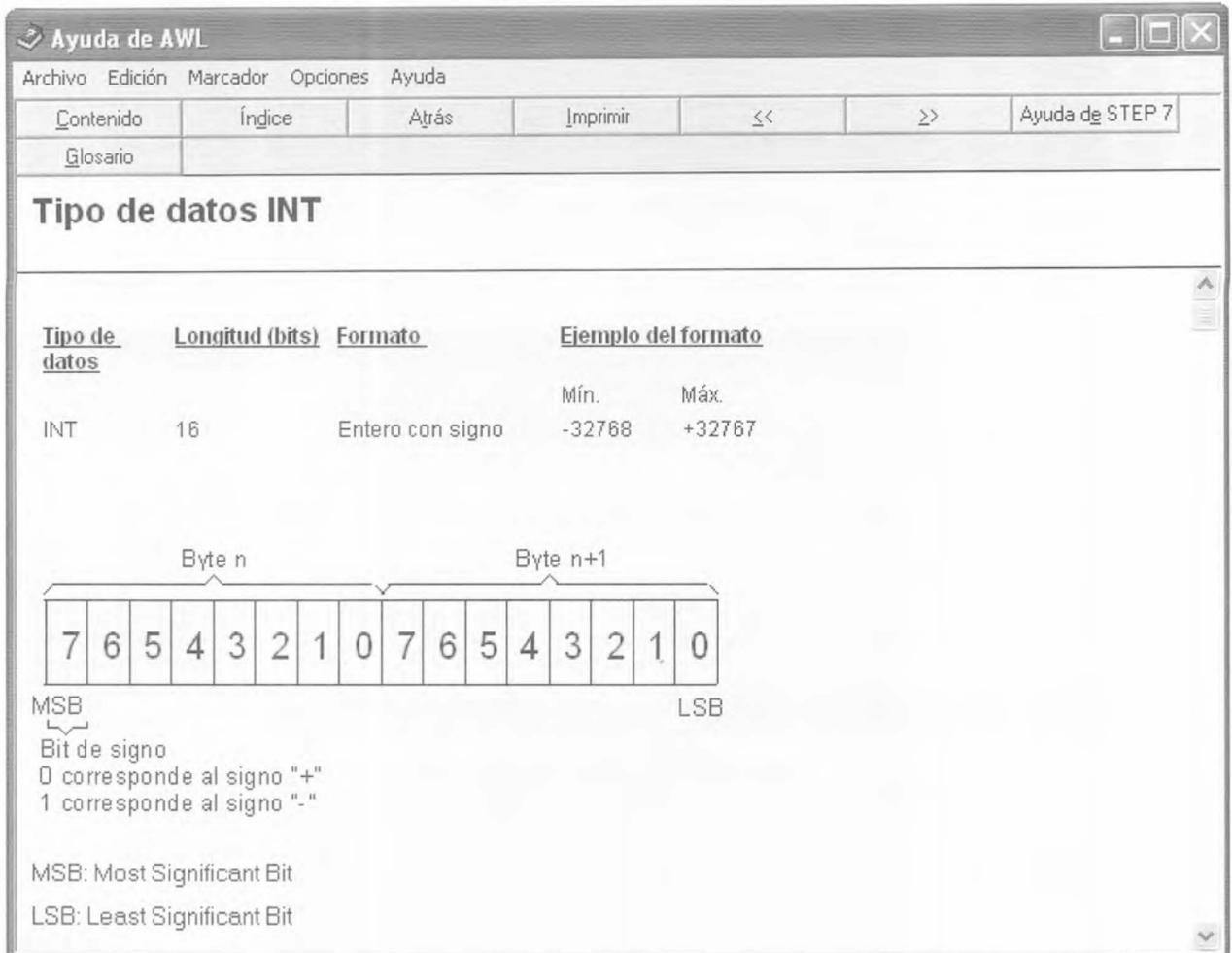


Fig. 258

En cualquier punto del programa que nos encontremos, con F1 obtendremos la ayuda contextual del tema.

1.2.3. Ayuda de instrucciones AWL, KOP y FUP

Otra ayuda que encontramos muy útil y que recomendamos su uso, es la que encontramos dentro del editor de bloques.

Allí encontramos el siguiente menú:

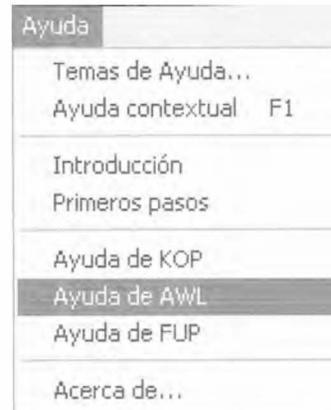


Fig. 259

Dentro de la ayuda de AWL encontramos todas las instrucciones del editor ordenadas por orden alfabético. Es muy útil, sobre todo para interpretar programas que no hemos hecho nosotros. Si encontramos una instrucción que no entendemos, no tenemos más que ir a esta ayuda, seleccionar la ficha "índice" y escribir la instrucción. Por ejemplo, imaginemos que encontramos en un programa la instrucción ITD y no sabemos lo que significa.

Buscamos en la ficha "índice" y encontramos lo siguiente:

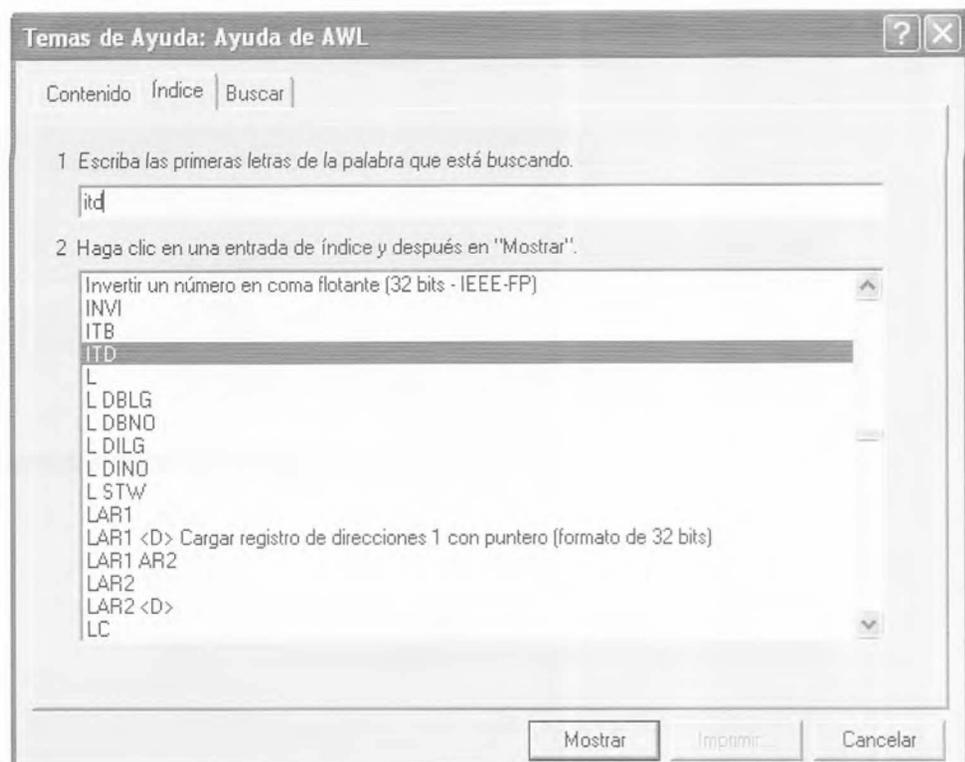


Fig. 260

Si ahora pulsamos “Mostrar” obtenemos lo siguiente:

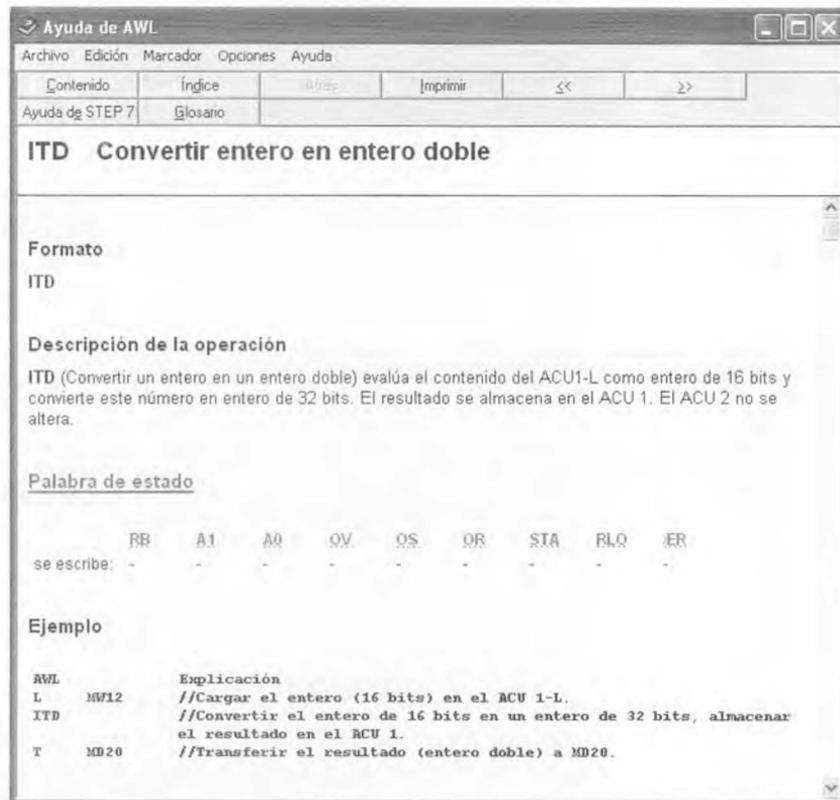


Fig. 261

Además de los nombres de las instrucciones, también tenemos la descripción de lo que hacen por sí lo que necesitamos es buscar la instrucción para hacer una operación en concreto. Aprovechando el ejemplo anterior, a la misma ayuda podemos llegar buscando la funcionalidad en lugar de la instrucción. Buscaríamos “Convertir entero en doble entero”.



Fig. 262

Si pulsamos mostrar, encontraremos la misma ayuda que en el ejemplo anterior. Lo mismo que hemos visto para AWL, lo podemos encontrar para KOP y FUP. Tenemos la ayuda tanto para escribir la instrucción y que nos diga lo que hace, como para escribir lo que queremos hacer y que nos indique la instrucción necesaria. Tanto de una manera como de otra, al pulsar Mostrar, se nos indica lo que hace la instrucción, como se utiliza y un ejemplo de la misma.

Ejercicio B



Recuerda . . .

Además de lo explicado hasta el momento, en la misma ventana en la que te encuentres programando, dispones una ayuda resumida ONLINE en la que puedes ver tanto información sobre errores, como datos básicos sobre símbolos y referencia cruzadas. También puedes hacer un diagnóstico rápido o un forzado de señales.

1.2.4. Vista detalles

En este capítulo, veremos cómo detectar errores de programa. También se explica en el último ejercicio, cómo comparar bloques. En el siguiente caso, a modo de información para el usuario, veremos cómo podemos ver errores tales como las referencias cruzadas, la información adicional, etc. del bloque editado, sin necesidad de abrir las herramientas adicionales del **STEP 7**.

Dentro del editor de bloques, disponemos de una información adicional sobre lo que estamos editando en cada momento. En la parte inferior de la ventana, podemos ver una serie de pestañas que nos muestran errores o información adicional sobre lo que estamos escribiendo. Es posible que no veamos estas pestañas porque por defecto aparecen minimizadas. Pero siempre podemos acercarnos con el ratón y abrir esta zona de información.

Veamos varios ejemplos de la información que podemos obtener de aquí. Imaginemos que estamos programando en AWL y escribimos mal una instrucción. Para empezar, el propio editor nos la marcará en rojo y no nos la aceptará como buena. Imaginemos que en lugar de escribir U E 0.3, escribimos U T 0.3.

En la pestaña inferior de "errores" podremos ver la siguiente información:

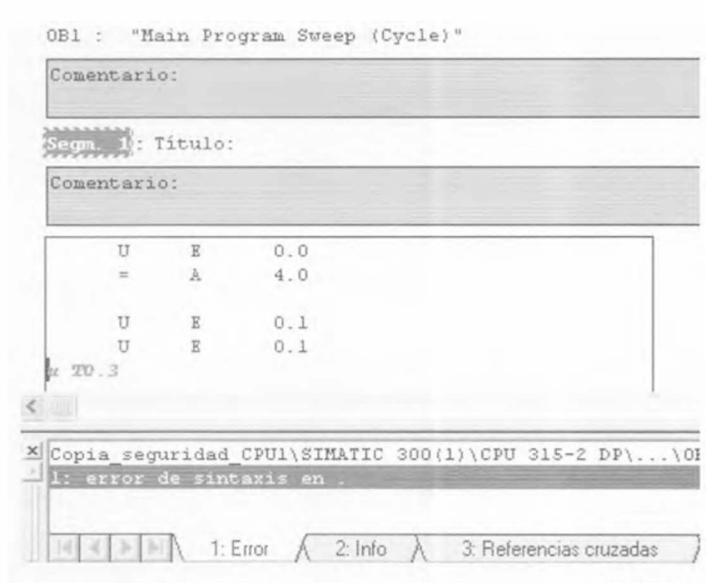


Fig. 263

En la información adicional se nos informa de que tenemos un error de sintaxis. Si hacemos doble clic sobre el error, nos sitúa el cursor del ratón sobre la instrucción que contiene el error.

En la ficha "Info" de la vista de detalles, podemos ver información sobre lo que tenemos seleccionado con el ratón en cada momento.

Veamos un ejemplo:

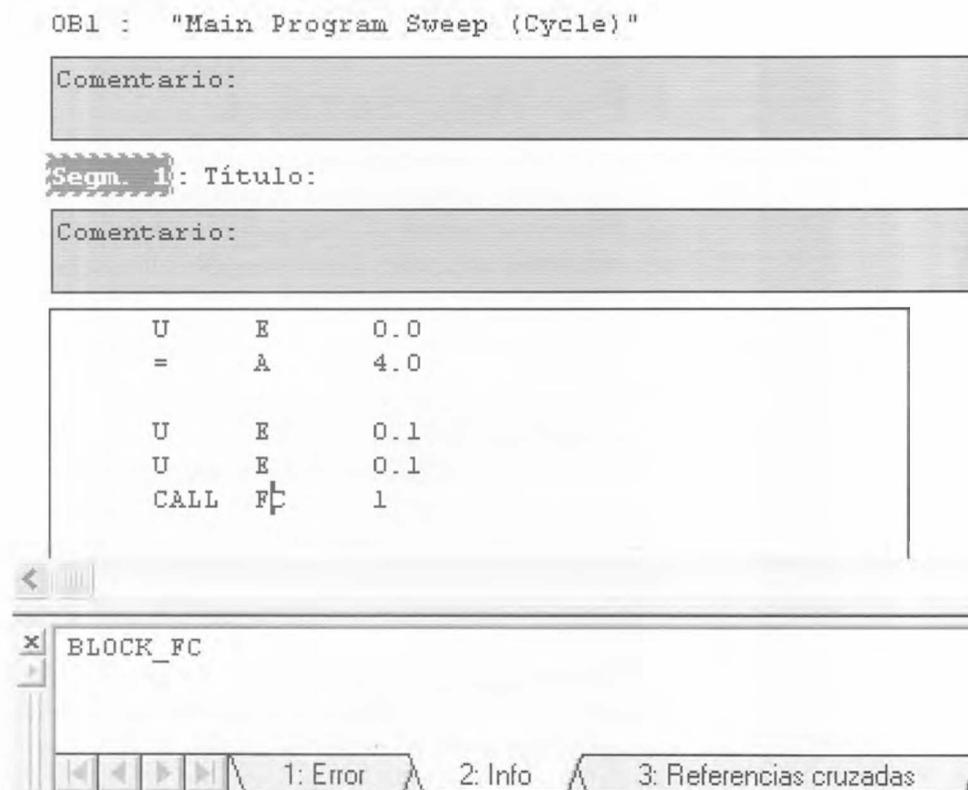


Fig. 264

Aquí tenemos seleccionado con el cursor la llamada a la FC 1. En la ficha de "Info" nos dice que lo que tenemos seleccionado es un bloque de función.

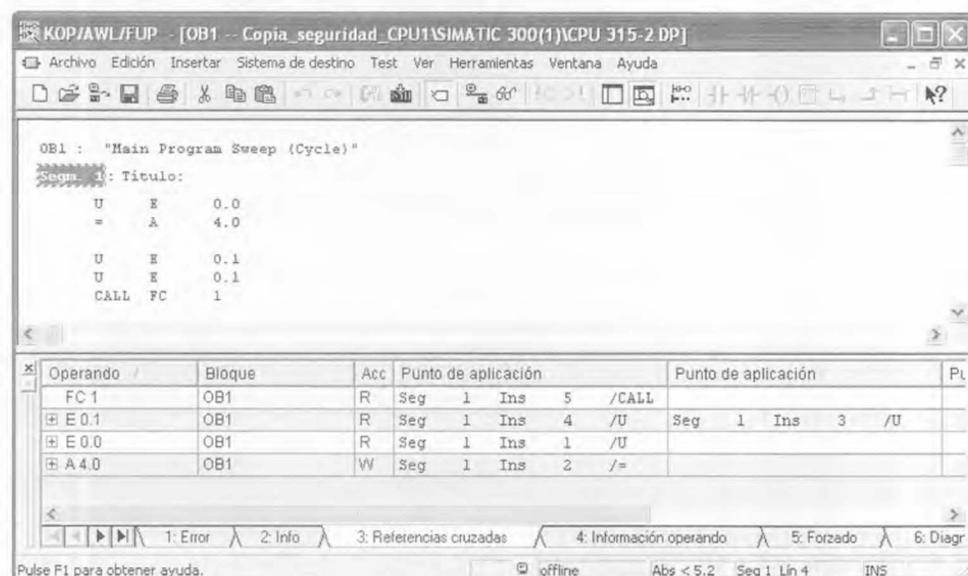


Fig. 265

En la pestaña de referencias cruzadas, podemos ver las referencias de los elementos que estamos utilizando en el bloque de programación que tenemos abierto. Podemos tener esta información siempre presente sin necesidad de abrir el menú específico de las referencias cruzadas.

Si vamos a la siguiente pantalla podemos obtener información del operando que estamos utilizando en el momento. Veamos un ejemplo:

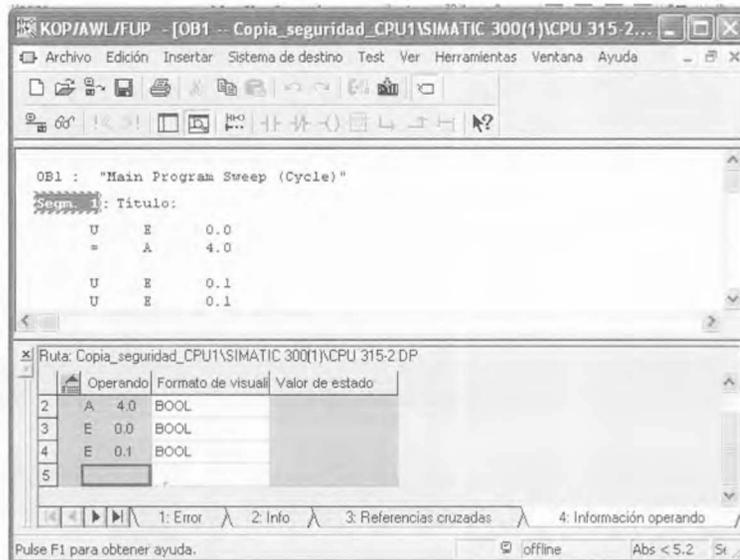


Fig. 266

Como vemos en el ejemplo, en esta ficha podemos ver los operandos que estamos utilizando en este bloque, y también podríamos ver su estado si estuviésemos conectados **ONLINE**.

En la siguiente pestaña de información vemos algo parecido, pero además tenemos la posibilidad de forzar valores del operando que estamos visualizando:

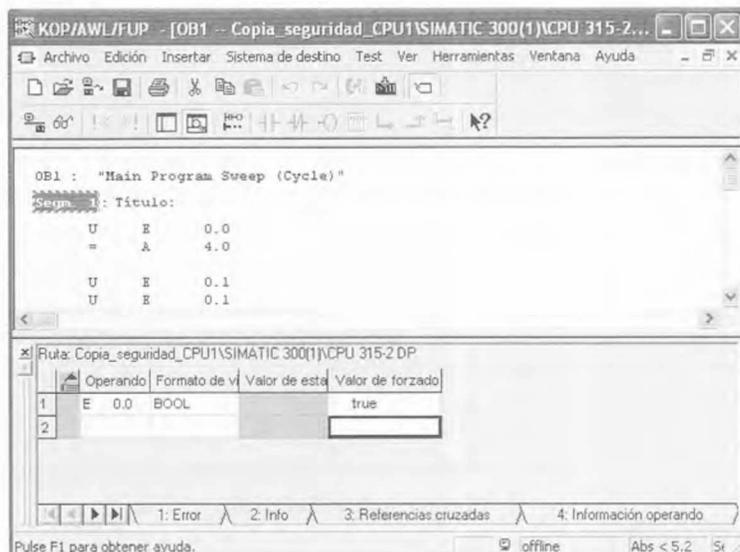


Fig. 267

Tendremos la opción de forzar valores de los operandos, siempre y cuando estemos conectados **ONLINE**.

La ficha de comparación, nos puede ser útil cuando estamos utilizando la opción de comparar bloques explicada en este mismo manual.

Además de lo que podemos ver desde la ventana del Administrador de **SIMATIC**, aquí en la ficha de "comparar", podemos ver todos los datos de utilidad dentro del bloque que estamos comparando.

Aquí podemos ver las diferencias que tenemos en el bloque que estamos comparando. También podemos hacer las modificaciones que consideremos necesarias, y sin salir del editor de bloques, podemos actualizar desde aquí mismo para ir anulando diferencias según vayamos corrigiendo. También tenemos dos botones con los que podemos ir a la siguiente o a la anterior diferencia.

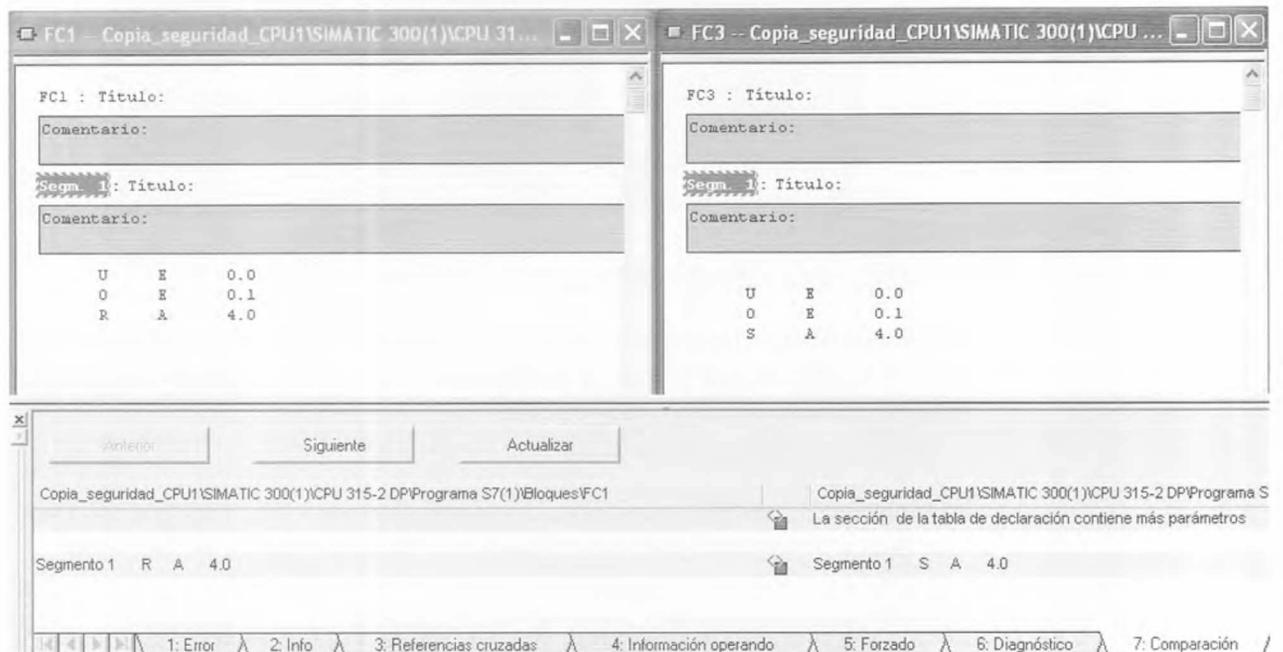


Fig. 268

Como hemos podido ver, con esta vista, no hacemos nada que no podamos hacer desde el Administrador de **SIMATIC**. No nos aporta ninguna herramienta nueva. Lo que sí que nos ofrece es una gran comodidad a la hora de utilizar las herramientas. Disponemos de la información o los datos en el lugar donde nos interesa y del operando o bloque que estamos editando, sin necesidad de ir a la ventana general del Administrador de **SIMATIC**.

Recomendamos al programador que tenga siempre esta vista abierta para facilitar la labor de programación.

1.3 Automatismos eléctricos

Un automatismo eléctrico consta de uno o varios circuitos cuya finalidad es alimentar eléctricamente a unos actuadores encargados de realizar un trabajo. Este trabajo puede ser mecánico, eléctrico, calorífico o puede generar un aviso luminoso o sonoro. El resultado del actuador también podría ser la conexión de sistemas de potencia o generadores eléctricos.

1.3.1. Conveniencia de los circuitos de mando

Cuando se pretende alimentar un actuador o sistema eléctrico permitiendo cierto grado de maniobra no limitada únicamente a la apertura o cierre, es conveniente separar el esquema eléctrico en dos: uno principal o de potencia y otro secundario o de mando (y señalización).

El circuito principal será el encargado de transmitir la potencia al elemento accionado.

Constará de tres o cuatro hilos o conductores en el caso de alimentación alterna trifásica o de dos hilos en caso de alimentación monofásica o de corriente continua y a los niveles adecuados de tensión (220 V o superior). Estos conductores deberán soportar el paso de la corriente para el que las máquinas estén diseñadas.

El circuito de mando será el encargado de realizar las funciones de temporización, autorretención, enclavamiento, etc. que nos permitirán un mayor control del proceso o dispositivo. Consta de dos hilos porque se trabaja generalmente con alimentación alterna monofásica de 220 V o menor. Los elementos que forman parte del circuito de mando no maniobran con elevadas potencias y por tanto no se les exigen las mismas condiciones que los elementos del circuito de potencia (son más baratos).

De este modo, al separar el circuito en dos, se consigue:

- Una simplificación en los esquemas, pues se trabaja con dos esquemas diferentes más sencillos
- Un ahorro en cableado, pues el mando se encarga a un circuito monofásico en vez de trifásico (el usual en la industria)
- Un ahorro en los elementos, pues a los elementos del circuito de mando no se les exigen las mismas características que a los de potencia.

Si el elemento a alimentar es de escasa potencia y la maniobra que se pretende realizar es simple, no suele haber esta separación.

Recuerda . . .

Todas las instalaciones deben estar correctamente protegidas para que no exista peligro para el usuario. Existen normativas que regulan las protecciones que debemos utilizar en cada caso.

1.3.2. Necesidad de los elementos de protección

Además de las acciones de maniobra que pueden englobarse en lo que se denominaría la operación normal de la instalación, existen otras acciones que son necesarias para proteger los elementos de la instalación o para proteger a las personas. De estas acciones se encargan los elementos de protección.

Dentro del primer grupo, los destinados a la protección de los elementos, se encuentran todos los dispositivos encargados de detectar condiciones anormales de funcionamiento y de realizar las acciones oportunas para evitar las consecuencias dañinas de ese mal funcionamiento. Estas acciones generalmente provocan la interrupción de la alimentación del elemento en situación anormal. Esta acción de interrupción a veces es instantánea tras la detección de la situación y otras veces permite cierto retardo en función de la gravedad de la situación. Los principales elementos dentro de este grupo son los relés térmicos o magnetotérmicos y los fusibles, que se encargan de detectar (los relés) o detectar y despejar (los fusibles) las sobrecargas y cortocircuitos.

En este sentido, conviene introducir el concepto de condiciones nominales. Son aquellas por encima de las cuales no está garantizado el perfecto funcionamiento del equipo, durante el periodo de vida del mismo:

- Si se trabaja por encima de la tensión nominal, es posible que los aislamientos no soporten esa tensión y se produzcan descargas y contorneamientos. También puede dar lugar a corrientes mayores de las esperadas.

- Si se trabaja por encima de la intensidad nominal las pérdidas por efecto Joule son demasiado elevadas y es posible que el sistema de refrigeración del equipo no permita disipar ese calor, con lo que la temperatura sube excesivamente y puede dañar el aislamiento. Por otro lado, un par por encima del nominal en una máquina rotativa puede producir una fatiga excesiva del material o directamente ocasionar la rotura del eje.

Dentro del segundo grupo de dispositivos de protección, los que se refieren a la protección de las personas, el principal es el relé diferencial, que detecta fugas de corriente.

1.3.3. Armarios eléctricos convencionales (sin PLC)

Hace unos años, toda la maniobra eléctrica se hacía a base de contactores, *relés*, temporizadores físicos, levas, etc. Los cuadros eléctricos eran por lo general mucho más grandes que hoy en día. La visualización, si es que tenían, se solía hacer con un sinóptico en la puerta del armario, animado con *leds* de colores.

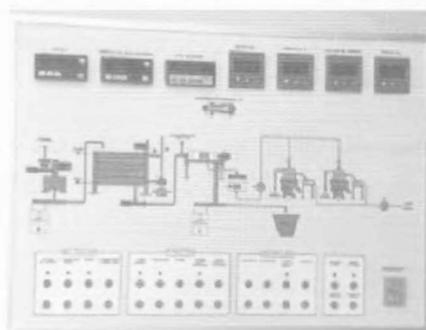


Fig. 1

Recuerda . . .

Hoy en día, la mayoría de cuadros eléctricos que se montan incluyen un PLC, por sencillo que sea el circuito. Y si la aplicación es un poco compleja, se suele incluir una visualización gráfica.

Cada vez más en los cuadros eléctricos se incluye un autómatas o PLC y una pantalla de visualización. El PLC hace las funciones que hacía antes la lógica cableada.

Evidentemente no podemos poner en marcha un aparato si no le llega tensión. Con la lógica cableada, se tenía que construir los circuitos eléctricos a base de contactos físicos para que le llegase tensión al elemento adecuado cuando interesaba ponerlo en marcha. Se debía crear un camino eléctrico para cada accionamiento. O incluso varios caminos si se tenía que activar con diferentes condiciones. En los cuadros modernos, ya no es necesario crear todos estos “caminos”. Ahora todos los sensores y actuadores se cablean a las entradas del PLC. Los accionamientos se cablean a las salidas del PLC. No se necesita tanto cableado como antes. Además el circuito así obtenido es mucho más versátil. Mediante programa, le decimos al PLC que cuando se cumplan una serie de condiciones, dé tensión a una determinada salida. Siempre podremos cambiar la salida en cuestión o las condiciones que la activan sin necesidad de tocar el cableado. Además los temporizadores, contadores y levas que se utilizaban antes, ya no son aparatos físicos que ocupan espacio. Son parte del *software* del PLC.

La visualización se hace a través de pantallas en los propios cuadros o sistemas escadas en ordenadores situados en los puntos de la nave donde interese. Incluso remotamente en lugares fuera de la instalación en la que se encuentra el cuadro eléctrico.

Los armarios se han reducido considerablemente y la versatilidad y funcionalidad ha aumentado muchísimo.

Además de todos estos cambios, cada vez más se utiliza la periferia descentralizada. Esto quiere decir que las entradas y salidas del PLC, así como los aparatos de campo, no es necesario incluirlos en el armario eléctrico principal. Reducimos todavía más el armario. Los elementos de campo y sus conexiones, se instalan en pequeñas cajas al lado de donde son necesarios. Entre estas cajas auxiliares y el armario principal se conecta un cable de *bus* industrial (**PROFIBUS**, **ASI**, **ETHERNET**, etcétera).

Todo esto nos ofrece reducción de cableado, reducción de trabajo, reducción de espacio, y lo que es muy importante, mejor diagnóstico de la instalación.

Este libro está dedicado a la programación de PLC, pero a modo de introducción, se incluye este capítulo dedicado a la lógica cableada y a la introducción de controladores lógicos  como paso intermedio entre la lógica cableada y un PLC propiamente dicho.

Para hablar de la lógica cableada, daremos una serie de definiciones sobre los elementos que la componen. Algunos de estos elementos se siguen utilizando en los cuadros eléctricos en los que se incluye un PLC. Antes daremos algunas definiciones referentes a los PLC y visualización para que el lector pueda entender mejor las explicaciones que se dan en este capítulo.

PLC: PLC significa “Controlador lógico programable”. Son siglas en inglés y es como se conocen comúnmente. En documentación de **SIEMENS**, a veces se hace referencia a ellos como SPS. Son las siglas en alemán de las mismas palabras. También son llamados en castellano “autómatas”.

Son equipos que se hacen funcionar mediante un programa. Dependiendo de los estados de las entradas al PLC, se hace actuar a las salidas del mismo. Existen de muchas marcas, cada uno se programa de una manera diferente y con lenguajes también diferentes. Lo que tienen en común es que son programas de ejecución cíclica. Los programas son leídos en unos pocos milisegundos. En este manual se

hace referencia a los PLC **SIEMENS** y al **STEP 7** que es el lenguaje de programación de estos PLC.

ESCADA: Es una aplicación de PC para visualizar la instalación. Normalmente los escadas se comunican con los PLC y visualizan los datos contenidos en ellos. También es posible introducir datos al PLC a través de variables de entrada / salida. Además, los escadas pueden guardar datos en ficheros históricos, mostrar alarmas de los PLC, hacer gráficas de los datos recogidos o cualquier otra aplicación que se programe en el PC (por ejemplo, comunicar con Excel y rellenar tablas de datos, abrir aplicaciones pulsando un botón, etc.). Los escadas también los hay de diferentes marcas y cada uno se programa de forma diferente. El escada de **SIEMENS** es el Win CC. Los fabricantes de escadas, suelen proporcionar al usuario *drivers* para comunicar sus escadas con otros PLC que no sean de la misma marca.

PANTALLA DE VISUALIZACIÓN: Son pantallas que se instalan cerca de los PLC para su visualización. Cada vez más estas pantallas son táctiles y se denominan TP. Estas pantallas suelen ser de pocas pulgadas (más pequeñas que un monitor de ordenador). Las hay en blanco y negro y en colores. La visualización que se realiza aquí, es más reducida que en un escada. Se ven de forma gráfica o numérica, valores incluidos en el PLC, se pueden introducir datos al PLC mediante campos de entrada / salida o pulsadores, se pueden visualizar alarmas y poco más. También en este caso, las pantallas de las diferentes marcas que hay en el mercado, se programan con software diferentes. Para las pantallas de **SIEMENS**, se utilizaba el PROTOOL. Ahora se utiliza el Win CC flexible. Los fabricantes de pantallas también suelen proporcionar a los usuarios *drivers* para comunicar sus pantallas con PLC de otras marcas. No es necesario que la pantalla sea de la misma marca que el PLC para poder hacer una visualización.

1.3.4. Cuadros de control

En los cuadros de control, se agrupan normalmente todos los dispositivos de control y protección de una instalación. También se suelen instalar en los cuadros eléctricos los aparatos de visualización y medida.

En las instalaciones más antiguas, se solía instalar un armario de control principal (bastante grande) en el que se agrupaban dentro de él todos los dispositivos de control y protección de la instalación. Era muy común instalar un sinóptico en la misma puerta del armario o en un lugar cercano, indicado mediante *leds*, alarmas, flujos de material, averías, alarmas, etc. Estos sinópticos eran un dibujo lineal que representaba la instalación y en el que se insertaban unos leds de colores que indicaban el funcionamiento de la misma.

En la parte interior del armario, los automatismos se generaban a través de contactores e interruptores. En este mismo capítulo veremos algunos automatismos de los más utilizados en la industria y que todavía hoy en día se utilizan en muchas máquinas.

Cada vez más esta distribución y visualización de las instalaciones está cambiando. Ahora ya no es tan común montar un armario eléctrico principal tan grande. Lo que cada vez está más extendido es montar un armario principal en el que se incluye la entrada de corriente, las fuentes de alimentación necesarias, las protecciones de las líneas secundarias y un PLC que controlará la instalación mediante el programa que se le introduzca. Después se montan armarios secundarios o pupitres repartidos por la instalación de manera que se tiene la visualización donde se necesita, así se ahorra en cableado y montaje.

1.3.5. Dispositivos de un cuadro convencional

Como introducción a los automatismos eléctricos, daremos una serie de definiciones de los aparatos más utilizados en los cuadros eléctricos convencionales.

Los agruparemos en tres grandes grupos, aunque algunos de ellos podrían estar incluidos en dos de ellos. Hay dispositivos de potencia que pueden actuar como dispositivos de protección, por ejemplo.

1.3.6. Dispositivos de mando o maniobra seccionador

Es un aparato que permite abrir o cerrar circuitos siempre en vacío. Se suele utilizar para circuitos de alta tensión o altas intensidades. Su función es básicamente de protección a la hora de realizar mantenimientos en la instalación. Con el seccionador abierto, nos aseguraremos de que tenemos las fuentes de alimentación sin tensión y podremos hacer un mantenimiento seguro.

En caso de poder actuar bajo carga, se denominan seccionadores bajo carga o interruptores-seccionadores. Pueden llevar un fusible incorporado y se denominan "ruptofusibles".

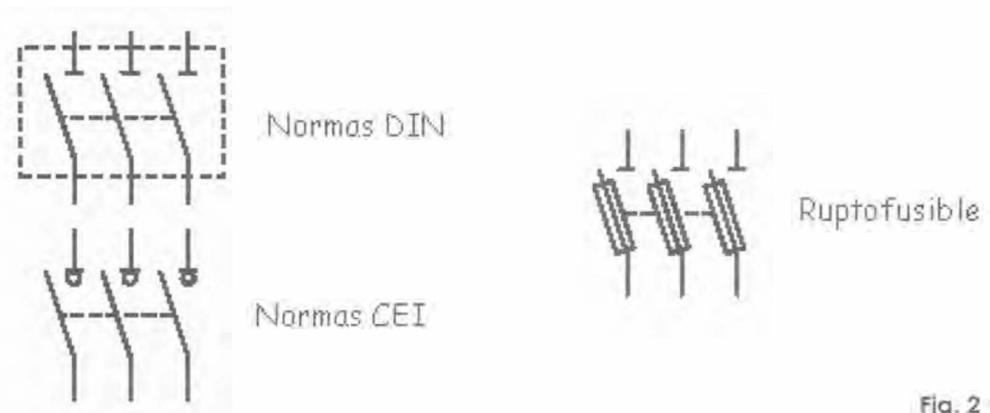


Fig. 2

INTERRUPTOR DE CARGA: Es un aparato manual que permite abrir y cerrar el circuito en condiciones normales de carga. Si hablamos de instalaciones de intensidades nominales elevadas, estos aparatos pueden producir un arco eléctrico entre los contactos que puede llegar a destruirlos. En estos casos, es necesario instalar un sistema de extinción de arco eléctrico.

PULSADOR: Por la parte exterior o visible, es lo que todo el mundo conoce como pulsador o "botón". Es algo que se pulsa o aprieta y, al soltarlo, vuelve a la posición inicial o de reposo. Por la parte interior, es un contacto que tiene una sola posición estable. Si en la posición de reposo permite el paso de corriente, será un pulsador normalmente cerrado o pulsador de apertura (o pulsador de paro), si en la posición de reposo no permite el paso de corriente, será un pulsador normalmente abierto o pulsador de cierre (o pulsador de marcha).

Cuando el pulsador normalmente cerrado es activado manualmente (se pulsa), el contacto se abre, y abre también el circuito durante el tiempo en que se mantiene pulsado no permitiendo el paso de corriente. Se suele utilizar como pulsador de paro por un tema de seguridades. Normalmente, una máquina tendrá sus condiciones de marcha y sus condiciones de paro, pero ante una emergencia, se querrá parar la máquina. En ese caso tendremos que actuar el pulsador de paro. Imaginemos que el pulsador de paro es un contacto normalmente abierto. Y que por una avería se nos ha roto el cable del pulsador. Como mientras esté abierto no actúa, sólo nos daremos cuenta de que está roto, cuando queramos parar la máquina y al pulsar el botón ésta no para. Si la hemos querido parar por una emergencia, podremos haber provocado un accidente. En cambio, si el contacto del pulsador de paro es normalmente cerrado, en el momento en que se rompe el cable por error, en ese momento nos damos cuenta porque se para la máquina. No tendremos que esperar a que haya una emergencia para darnos cuenta de que tenemos un problema de cable roto. Normalmente todas las seguridades las utilizaremos con contactos normalmente cerrados por este motivo.

Cuando el pulsador normalmente abierto es activado, el contacto se cierra, y realiza la conexión eléctrica entre sus contactos. Al dejar de pulsar, el circuito se abre y deja de pasar corriente a través suyo. Si en este caso se nos rompiera el cable, lo que ocurriría es que no podríamos poner la máquina en marcha. En este momento nos daríamos cuenta de la avería pero no tendríamos el riesgo de provocar un accidente. La máquina quedaría parada hasta solventar el problema de cable roto.

La nomenclatura más utilizada para este tipo de pulsadores es: Pulsador normalmente abierto (NA o NO). Pulsador normalmente cerrado (NC).

Se suelen representar como vemos a continuación:



Fig. 3

INTERRUPTOR: Elemento de maniobra con dos posiciones estables. Interrumpe o establece la intensidad normal de funcionamiento. Aunque se confunde este término con el de disyuntor, el interruptor propiamente dicho de baja tensión no interrumpe la intensidad de cortocircuito.

El interruptor, también es conocido como posicionador, selector, interruptor de posición o conmutador. Es un contacto con dos posiciones estables. En una el contacto está abierto (no deja pasar corriente) y en la otra, cerrado (deja pasar la corriente). Algunos tipos de conmutadores, pueden establecer más de un circuito, o bien abrir un circuito a la vez que cierran otro,



Fig. 4

1.3.7. Dispositivos de protección

Son dispositivos para controlar los funcionamientos anómalos de una instalación eléctrica. El reglamento español de baja tensión, obliga a que todas las instalaciones estén protegidas frente a contactos directos o indirectos, sobretensiones, sobre intensidades o sobrecargas.

Veamos de una forma genérica qué son estos peligros y como pueden afectar a la seguridad de los mismos:

- **Sobreintensidades o sobrecargas:** Se producen al sobrecargar la instalación (utilizarla por encima de sus posibilidades) o por cortocircuitos imprevistos. Lo que ocurre en estos casos es que la intensidad aumenta considerablemente produciendo fuertes calentamientos que pueden dañar la instalación e incluso llegar a provocar incendios.

- **Contactos eléctricos:** Se habla de contacto eléctrico cuando una persona entra en contacto físico con una instalación eléctrica, de manera que provoca el paso de corriente a través de su cuerpo. Estos contactos pueden llegar a ser realmente peligrosos y producir incluso la muerte de la persona que entra en contacto con la instalación. La peligrosidad del accidente depende sobre todo de la intensidad que circule por el cuerpo y del tiempo que dure el contacto. Se habla de contacto indirecto, cuando una masa entra en contacto accidental con una parte en tensión. Teóricamente las masas deben tener tensión 0 y no es peligroso tocarlas. Una masa puede ser la carcasa de una lavadora, la base de un ordenador, la puerta de un armario eléctrico por ejemplo.

- **Sobretensiones:** Son aumentos inesperados de tensión que pueden provocar averías, desperfectos o incluso la destrucción de los aparatos o equipos conectados a la red.

Los dispositivos de protección más comunes son: fusibles, relés térmicos, interruptores automáticos, interruptores diferenciales y protectores de sobretensiones. Analicemos de manera genérica qué son y cómo actúan cada uno de ellos:

- **Fusibles:** Es un dispositivo que físicamente se funde y provoca un corte en el circuito eléctrico cuando la intensidad que circula por él es superior a la intensidad nominal durante un cierto tiempo. Cuando uno de estos dispositivos actúa, es necesario sustituirlo por otro ya que se destruyen al fundirse. Este dispositivo protege la instalación de sobreintensidades.

- **Relé térmico:** Es un dispositivo que protege al circuito de sobrecargas (intensidades por encima de la nominal); no actúa instantáneamente sino que el tiempo que tardan en abrirse sus polos (o dar la orden de apertura) depende de cuánto más elevada es la intensidad por encima de la nominal a la que estén trabajando.

- **Interruptor automático:** Es un aparato mecánico de conexión capaz de establecer, soportar e interrumpir corrientes en las condiciones normales del circuito, así como de establecer, soportar durante un tiempo determinado e interrumpir corrientes en condiciones anormales como las de cortocircuito.

Son dispositivos diseñados de forma tal que al detectar cierto tipo de anomalía en el circuito, ordenan su propio disparo, dejando el circuito abierto.

RELÉ: Es un elemento típicamente usado en protección aunque por su funcionamiento puede desempeñar funciones de maniobra. En algunos casos puede ser un dispositivo de mando.

Relés usados en protección:

Son dispositivos que muestrean una o varias magnitudes eléctricas y en función

Recuerda . . .

Muchos de estos componentes se siguen utilizando en los armarios basados en autómatas programables.

de su valor o de la relación entre las magnitudes son capaces de detectar si existe un mal funcionamiento del sistema que están controlando.

Esta condición suele ser la de una excesiva intensidad, pero también puede producirse el disparo (apertura de contactos) por una excesivamente grande o pequeña tensión o frecuencia, por una inadecuada dirección de la potencia (funcionamiento como motor de alternadores), por una baja o elevada intensidad en el circuito de excitación de máquinas síncronas, etcétera.

Al advertir un mal funcionamiento de la magnitud que controlan, o bien se produce la apertura de sus polos (contactos) interrumpiendo la alimentación de un circuito eléctrico o máquina, o bien dan la orden de apertura al dispositivo encargado de la desconexión.

En baja tensión y para pequeñas potencias, al elemento sensible que detecta la condición de apertura se le incorpora, en el mismo cuerpo, otro elemento actuador que realmente produce la apertura de los polos. En alta tensión o con potencias elevadas estos dos elementos están separados en un órgano sensible (*relé* propiamente dicho) y el órgano actuador o mecanismo de disparo (contactor o contactor más disyuntor).

Los principales y más sencillos relés de protección que se encuentran en una instalación son:

Relés térmicos que protegen al circuito frente a sobrecargas (intensidades por encima de la nominal); no actúan instantáneamente sino que el tiempo que tardan en abrir sus polos (o dar la orden de apertura) depende de cuánto más elevada es la intensidad por encima de la nominal.

Relés magnetotérmicos que unen a su característica térmica un elemento que opera instantáneamente por acciones electromagnéticas cuando la intensidad es muy superior a la nominal, previsiblemente porque existe un cortocircuito cercano; la acción magnética puede llevar incorporada un retardo independiente de la intensidad.

Interruptor diferencial

Elemento de protección que detecta los defectos de aislamiento. Da lugar a disparo instantáneo cuando existe una intensidad que se deriva a masa por encima de un determinado valor (30 mA o 300 mA).

Si existe un defecto de aislamiento, un conductor puede quedar unido eléctricamente a la carcasa o a alguna parte accesible por el personal, con lo que dicha parte estaría a una tensión peligrosa para el operario. Para evitarlo, si la instalación tiene una adecuada toma de tierra se derivará una intensidad en el momento en que se produzca dicho defecto de aislamiento y el interruptor diferencial interrumpirá la alimentación no permitiendo la conexión hasta que no se detecte y repare el defecto.

1.3.8. Dispositivos de potencia

CONTACTOR: Es un interruptor comandado a distancia, que vuelve a la posición de reposo, cuando la fuerza que lo acciona deja de actuar sobre él. Al utilizar contactores en un circuito tenemos algunas ventajas con respecto a utilizar la tensión directa sobre los elementos. Existe una mayor seguridad para el operario puesto que la bobina del electroimán que acciona el contactor puede trabajar a tensiones menores que el motor o lo que queramos accionar a través del contactor.

Su misión es la de establecer la corriente de alimentación de un dispositivo eléctrico (normalmente un motor) al ser accionado, o bien modificar la forma en que sea alimentado el dispositivo eléctrico. Esto se consigue aplicando tensión a la bobina del contactor.

Cuando la bobina deja de ser excitada, sus contactos volverán a su estado de reposo dejando de alimentar al motor al que estaba conectado. En definitiva, el contactor permite al ser activado o impide al ser desactivado, el paso de corriente en una parte del circuito de potencia.

Es, por tanto, capaz de establecer, soportar e interrumpir corrientes en condiciones normales del circuito, incluidas las de leve sobrecarga.

El contactor sirve para comunicar las órdenes finales obtenidas del circuito de mando al circuito principal, aunque no hay contacto eléctrico entre ambos.

Los principales componentes son:

1. El electroimán: Formado por un circuito magnético y una bobina. Es el órgano activo. Cuando se aplica una tensión a la bobina, el yugo (parte fija del circuito magnético) atrae al martillo (parte móvil), y éste, en su movimiento, arrastra a todos los contactos que van solidariamente unidos a él. De esta manera, la aplicación de una tensión a la bobina del contactor (dentro del circuito de mando) se transforma en la apertura y cierre de una serie de contactos (del circuito de potencia y también del de mando).

2. Los polos o contactos principales: Son los elementos que establecen y cortan las corrientes del circuito principal. Esto se consigue por unión o separación de sus contactos, lo que produce un arco eléctrico que hay que controlar, especialmente en la desconexión. Por eso, los contactos de los polos son las piezas que están sometidas al trabajo más duro en el contactor.

3. Contactos auxiliares: Son los elementos que establecen y cortan corrientes en el circuito de mando. Realizan las funciones de señalización, enclavamiento y autorretención.

Una de las formas en que se representa el contactor (en este caso un contactor trifásico con dos contactos auxiliares NC y dos NA) es:

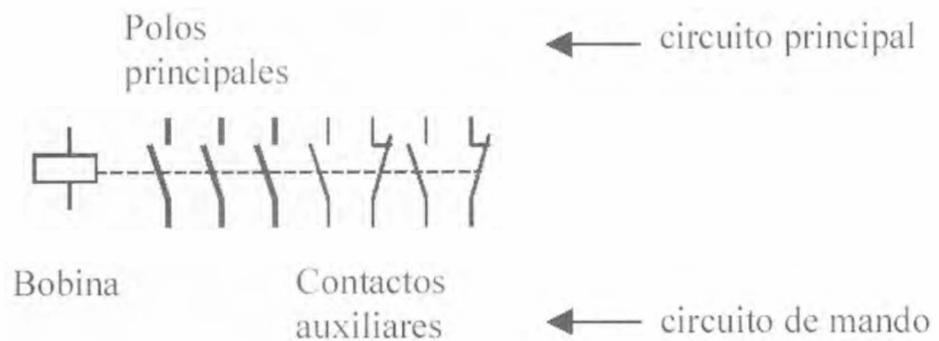


Fig. 5

Es accionado (directa o indirectamente) por pulsadores (marcha) o *relés* térmicos.

DISYUNTOR: Elemento de protección accionado por un *relé*. Es capaz de interrumpir corrientes de cortocircuito (muy elevadas). Su poder de corte (intensidad que es capaz de interrumpir) depende principalmente de las características de los polos y de la capacidad de eliminar el arco que se establece al intentar interrumpir una intensidad elevada por la separación de los polos.

También puede interrumpir intensidades de operación normal o sobrecarga, o establecer la corriente eléctrica (es más fácil cerrar un circuito y establecer una corriente que abrir el circuito e interrumpir la misma corriente).



Disyuntor o interruptor
de potencia.
Normas DIN

1.3.9. Automatismos eléctricos

Simplemente contando con un circuito eléctrico, podemos llegar a construir pequeños automatismos mediante la lógica cableada. Sin necesidad de un PLC se pueden generar circuitos "automáticos". Estos circuitos se construirán a base de *relés* y contactores e incluso temporizadores o contadores independientes. A continuación tenemos unos ejemplos sencillos de automatismos eléctricos.

Este manual está dedicado a la programación mediante PLC. En concreto con PLC **SIEMENS** programados con **STEP 7**. Pero antes de entrar de lleno en lo que es la programación propiamente dicha, se hará una pequeña introducción de los automatismos cableados con algunos ejemplos de los más típicos en la industria.

También se añadirá algún pequeño esquemita eléctrico resuelto con un LOGO de SIEMENS que es un controlador lógico. Digamos que es un intermedio entre la lógica cableada y un PLC.

Ejemplo 1



1.3.10. Mando por contacto permanente

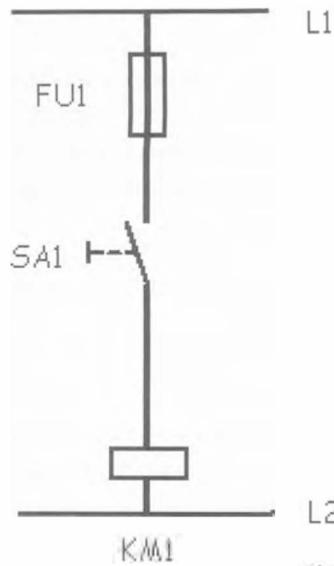


Fig. 7

En este circuito, tenemos un fusible de protección (FU1), un interruptor de mando (SA1) y un contactor (KM1). La bobina del contactor se pondrá en funcionamiento siempre y cuando el elemento SA1 se encuentre en posición de trabajo.

Ejemplo 2



1.3.11. Mando con memoria

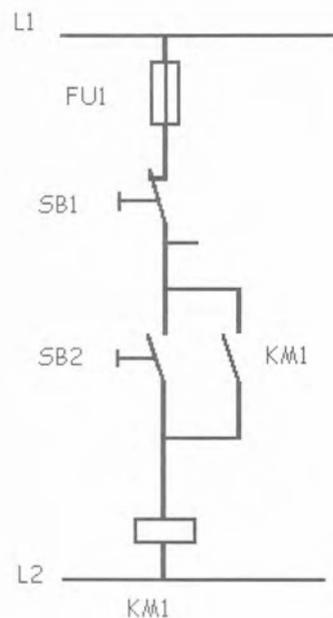


Fig. 8

Al accionar el pulsador SB2 se conecta la bobina del contactor, que a su vez cierra el contacto KM1 realimentando la bobina. Al desactivar el pulsador de marcha el contactor seguirá funcionando, gracias a la realimentación. Si se quiere desconectar el circuito, se deberá pulsar SB1, el cual cortará la señal de la bobina y esta volverá a abrir su contacto auxiliar dejándolo en su posición de reposo.

En este caso para alimentar la bobina, hemos tenido que crear dos caminos eléctricos físicamente con cable para poder alimentarla. Se alimentará a través de SB2 o a través de KM1. Además hemos tenido que cablear un elemento de protección en serie. Si en algún momento queremos que deje de funcionar el enclavamiento, deberemos cambiar el cableado. En cambio, si esto lo hacemos con un PLC, sólo deberemos cablear SB1 y SB2 a dos entradas del PLC. Y KM1 a una salida del PLC. Después haremos un programa como el siguiente:

```

U  E  0.0  Al pulsar SB2
S  A  4.0  Activar KM1 y mantener activado
U  E  0.1  Al pulsar SB1
R  A  4.0  Desactivar KM1 y mantener desactivado.

```

Estas instrucciones están explicadas en el capítulo dedicado a la programación de PLC, pero lo que intentamos explicar aquí, es que no necesitamos cablear de manera diferente si queremos un enclavamiento o si queremos el contacto directo. Para que este mismo circuito funcionase con contacto directo, deberíamos cambiar el cableado y dejarlo como en el ejemplo anterior. Si lo hiciésemos con PLC, no tocaríamos el cableado y programaríamos:

```

U  E  0.0
=  A  4.0

```

Las protecciones no hace falta que las cableemos en serie con todas las maniobras que hagamos. Con un PLC, podemos decirle que si las protecciones o seguridades no están bien, se paren todos los accionamientos. Sólo necesitamos cablearlas a una entrada del PLC. Por programa haremos lo que nos interese.

Sólo habrá que tener en cuenta que las paradas de emergencia, por ley, además de que actúen por programa, deberán actuar eléctricamente y quitar tensión al elemento que deban parar. En este caso tenemos que tener el cableado físico además de las conexiones al PLC.

Ejemplo 3



1.3.12. Arranque directo de un motor de inducción

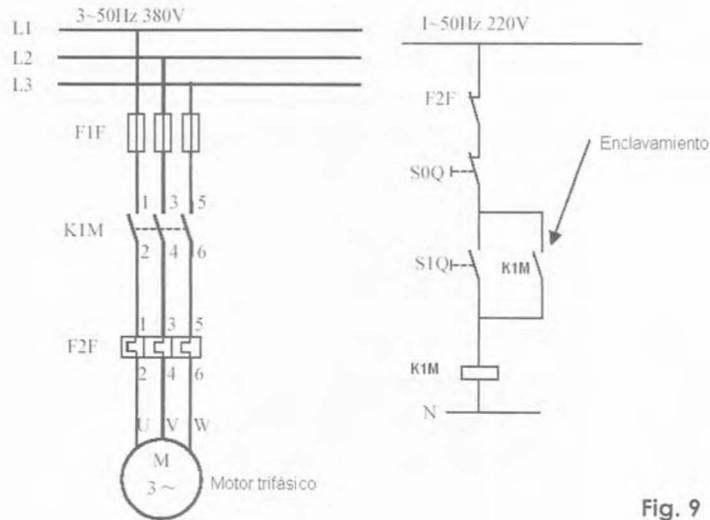


Fig. 9

Ejemplo 4



1.3.13. Arranque con inversión de giro

Se trata de cambiar la secuencia de fases de alimentación al motor, por ejemplo L1 L2 L3 a L3 L2 L1. Para cambiar de un sentido de giro al contrario, no es necesario pulsar antes el pulsador de paro (sí sería necesario si no estuviesen los contactos normalmente cerrados de S1B S2B). El enclavamiento producido por los contactos normalmente cerrados de K2B y K1B impide físicamente que se active una bobina estando excitada la otra, con lo que evita un posible cortocircuito bifásico.

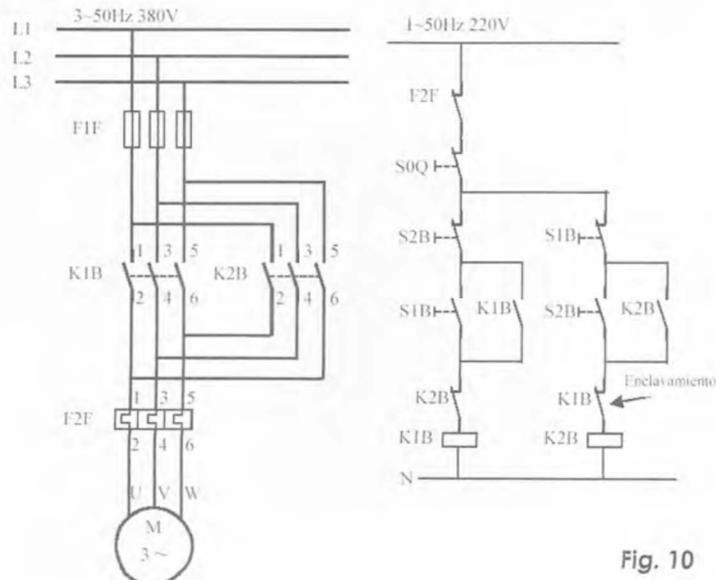


Fig. 10

Ejemplo 5



1.3.14. Arranque estrella - triángulo

Arranque muy utilizado en la industria pues permite disminuir en 1/3 la intensidad de arranque del motor de inducción (por la líneas de alimentación que llegan al motor) y, por tanto, disminuye las caídas de tensión en los equipos cercanos. Se utiliza cuando la configuración final es triángulo. Al activar el pulsador de marcha, el motor arranca con conexión estrella. Una vez cercana a la velocidad nominal del motor, se pasa a triángulo, con lo que el motor trabaja a tensión nominal. En esta situación se quedará hasta que se active el pulsador de paro.

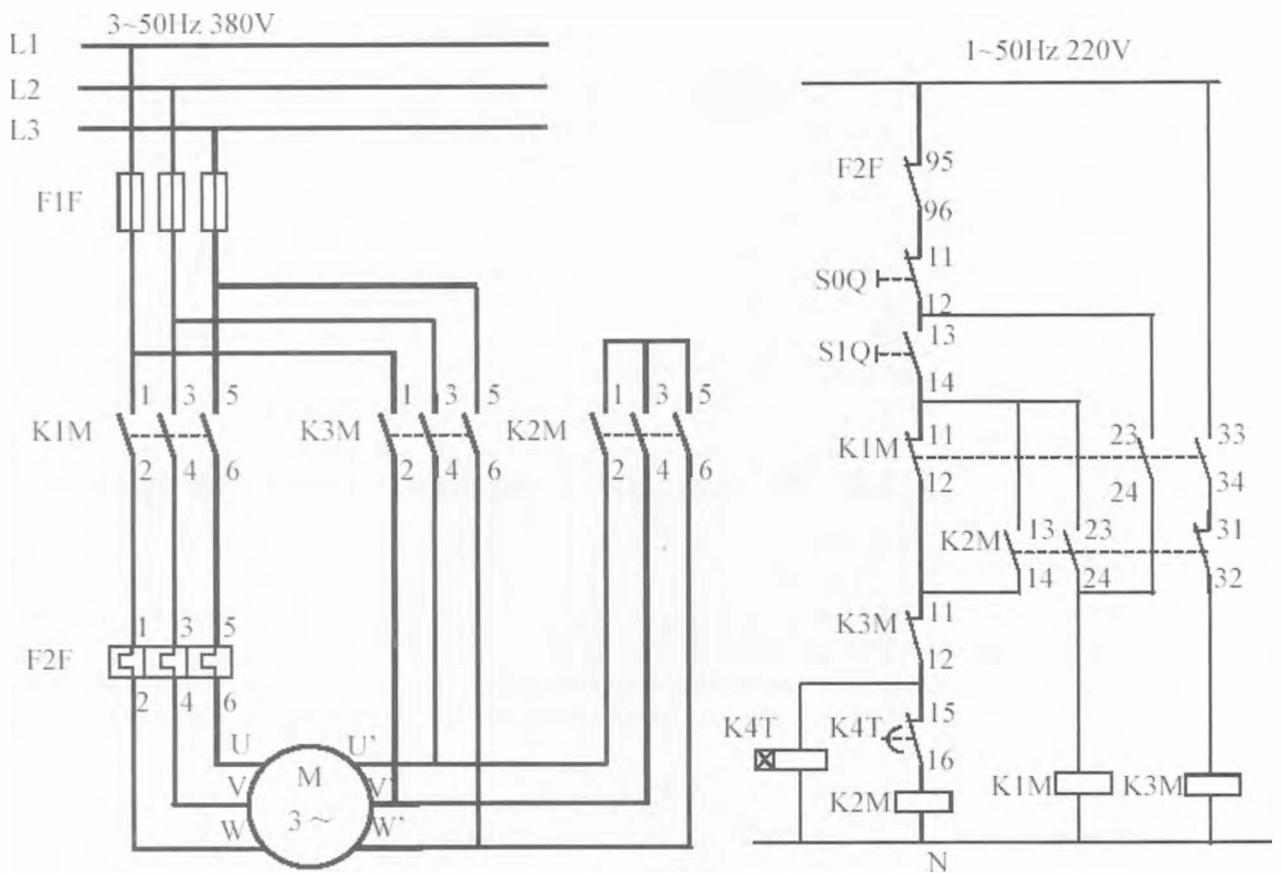


Fig. 11

1.4 Microcontroladores SIEMENS LOGO!

Entre la lógica cableada y un PLC propiamente dicho, **SIEMENS** dispone de un dispositivo llamado LOGO!. Este aparato es un controlador lógico. Viene a ser una especie de "PLC pequeñito" que a través de un programa, hace actuar unas salidas dependiendo del estado de las entradas. No hablamos de PLC porque no dispone de las mismas propiedades y está mucho más limitado que un PLC en cuanto a prestaciones.



Fig. 1

Recuerda . . .

Este dispositivo no llega a ser un PLC, pero nos resuelve, de manera muy sencilla, pequeñas aplicaciones. También podemos utilizar elementos de visualización y comunicaciones. Es muy útil para viviendas y pequeñas máquinas.

Lo que vemos en la imagen es el LOGO! más pequeño y sencillo. Como se puede apreciar, dispone de 8 entradas y 4 salidas. Se puede programar desde la misma pantalla con los botones de menú, o a través del *software* "LOGOSOFT".

Existen LOGO más grandes con 12 entradas y 8 salidas. También los hay con salidas de *relé* o directamente a 220v. También existen dos modelos de LOGO con posibilidad de comunicaciones. Existe el LOGO que podemos conectar a una red ASI para implementarlo en aplicaciones de automatización un poquito más complejas, y existe el LOGO con conexión KNX para utilizarlo en la automatización de viviendas y edificios. En este campo, es un aparato muy versátil con el que se pueden hacer aplicaciones sencillas pero muy útiles. Como se verá más adelante, el LOGO trae implementadas funciones que se pueden aplicar con gran facilidad al mundo de la vivienda, como por ejemplo el interruptor de escalera, los temporizadores semanales, los temporizadores anuales, etcétera.

Como equipo todavía más sencillo, tenemos el LOGO pure.



Fig. 2

Recuerda . . .

Este dispositivo es ampliable con diferentes módulos de funciones. Tanto con módulos de entradas / salidas como con módulos de comunicación.

Este equipo es el mismo que el LOGO normal, pero sin pantalla de programación. Sólo es programable a través de cable. Es muy práctico si nos dedicamos a fabricar muchas máquinas iguales. El equipo es un poco más barato, y sólo necesitamos tener el programa en un ordenador y un cable para programar todos los LOGO.

También existen accesorios complementarios para este equipo. Por un lado, disponemos de las fuentes de alimentación.



Tenemos varias fuentes de hasta 4A. de intensidad.

Con la última generación de LOGO, también ha salido al mercado una minipantalla en la que se pueden visualizar mensajes y mejorar la funcionalidad que ya traía incorporada la propia pantalla del aparato.



Fig. 4

La propia pantalla incluye 4 teclas de función que podemos programar a través del *software* del LOGO. Veremos que tenemos un contacto representado por una F llamado tecla de función. Hace referencia a estas 4 teclas.

También como novedad con respecto a los primeros LOGO, es que ahora disponemos de tarjetas de ampliación. En algunos LOGO, se pueden conectar ampliaciones de entradas / salidas.



Fig. 5

En este manual, veremos algunas aplicaciones prácticas realizadas con el *software* LOGOSOFT *comfort*. No es un manual dedicado expresamente a estos equipos. Veremos por encima sus funciones básicas y algunos ejemplos resueltos. Para conocer mejor estos equipos, recomendamos leer un manual de LOGO!.

Algunos de los ejemplos que veremos con LOGO!, son los mismos que posteriormente veremos resueltos en **STEP 7**. Así se podrá apreciar la diferencia entre el LOGO y un PLC propiamente dicho.

Ejercicio 1



1.4.1. Contactos serie

Si entramos en el software, lo que observamos es una pantalla como la siguiente:

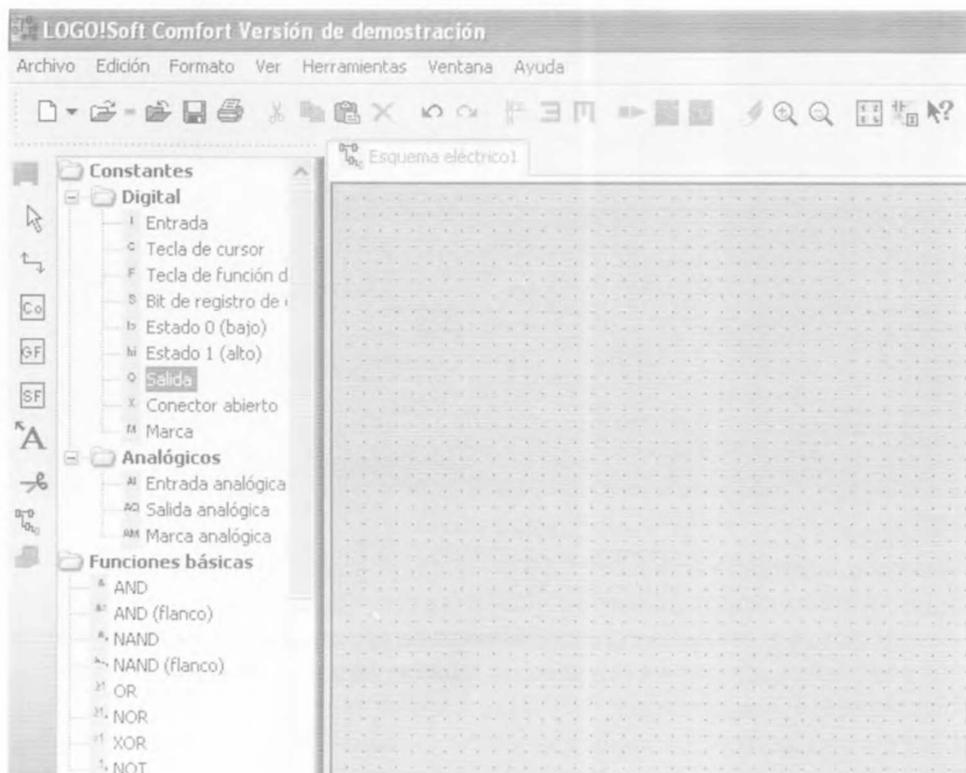


Fig. 6

En la parte izquierda vemos un menú con todas las opciones que tenemos disponibles para programar. En la parte central de la pantalla, disponemos del espacio para generar nuestro circuito eléctrico. En este ejemplo vamos a generar un circuito con dos contactos en serie.

Dentro del menú de opciones, vemos que lo primero que tenemos es un apartado llamado "Digital". Allí encontramos 9 constantes para programar. Encontramos las entradas y salidas del equipo, bits siempre a 1 y bits siempre a 0 y marcas, teclas de cursor y teclas de función que podemos utilizar como bits auxiliares.

Si con el interrogante que vemos en la barra superior de herramientas, pulsamos sobre cada una de estas funciones, obtenemos la explicación de lo que hacen y de cómo podemos utilizarlas. En la figura 2, vemos en la parte izquierda de la pantalla 3 botones llamados Co, GF y SF. Si pulsamos cada uno de ellos, aparece en el *software*, en la parte inferior, una barra de herramientas con los contactos, las funciones generales o las funciones especiales para que nos sea más rápido programar.

En el menú desplegable, también podemos ver las "Funciones básicas". Allí encontramos básicamente funciones AND y OR (contactos en serie y en paralelo). Encontramos funciones por contacto mantenido, por flanco y con contactos abiertos o cerrados.

En este primer ejemplo, queremos programar dos contactos abiertos en serie. Para ello utilizaremos dos entradas de LOGO y una AND normal.

Veamos cómo quedaría el ejemplo realizado con el *software*:



Fig. 7

Hemos programado una función "AND" con dos contactos de entrada y un contacto de salida.

Dentro del mismo *software*, podemos hacer una pequeña simulación. Para ello vamos al menú "Herramientas -> simulación".

Veremos lo siguiente:

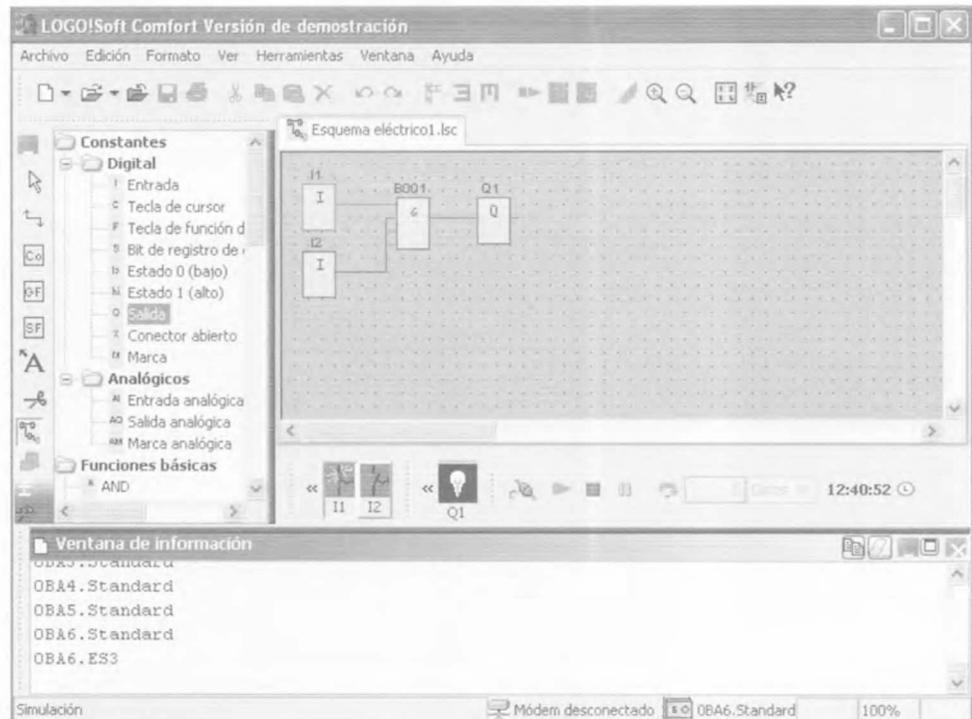


Fig. 8

En el ejemplo vemos que tenemos activa sólo la entrada I1. Por lo tanto no se enciende todavía la salida. En el simulador, aparecen las entradas utilizadas en el programa en la parte inferior de la ventana, y con un clic del ratón las podemos activar o desactivar. También vemos las salidas con un icono que representa una bombilla. Si está encendida quiere decir que tenemos la salida activa.

Todo esto también lo podemos ver en esquema de contactos (KOP). Para ello sólo tenemos que ir al menú “Archivo -> Convertir a KOP”. Con esto veremos el circuito como se muestra a continuación:

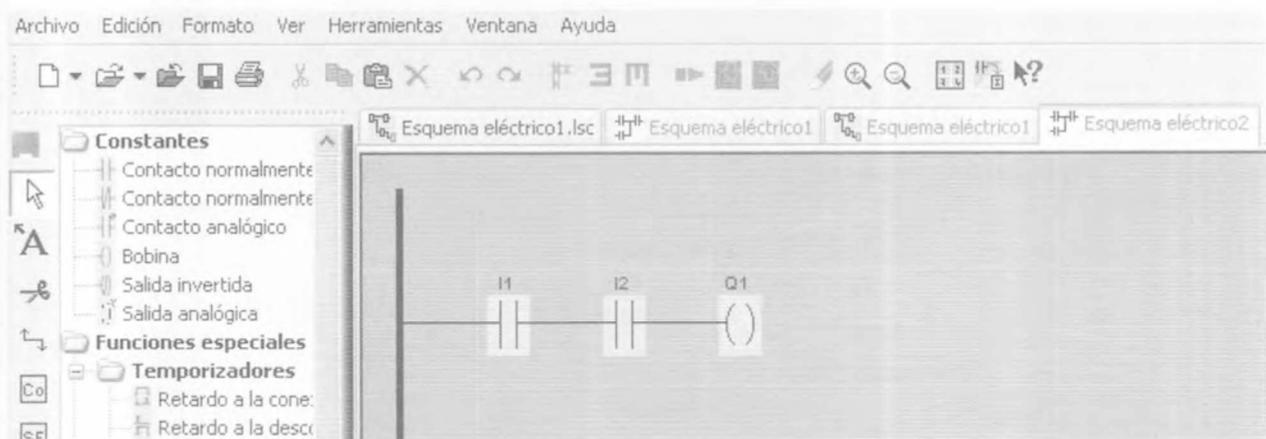


Fig. 9

Si lo programamos directamente en el LOGO, deberemos empezar a programar por la salida. Elegimos primero la salida que queremos activar y vamos programando hacia atrás las condiciones hasta llegar a las entradas. La programación que veremos será FUP no esquema de contactos. La desventaja que tiene programar directamente sobre la pantalla del LOGO es que vemos los elementos de programación de 1 en 1. No podemos ver el esquema eléctrico. En cambio, la ventaja es que no nos hace falta ordenador ni cable. Con el mismo aparato podemos programar o hacer modificaciones sin necesidad de nada más.

Ejercicio 2



1.4.2. Contactos en paralelo

Veamos cómo quedaría un circuito de dos contactos en paralelo.

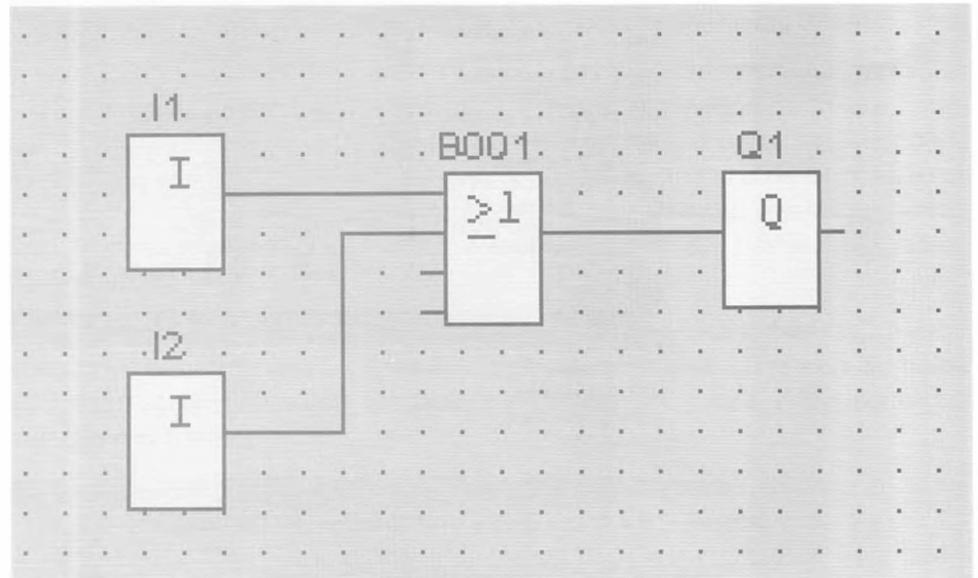


Fig. 10

Si lo visualizamos en KOP veremos lo siguiente:

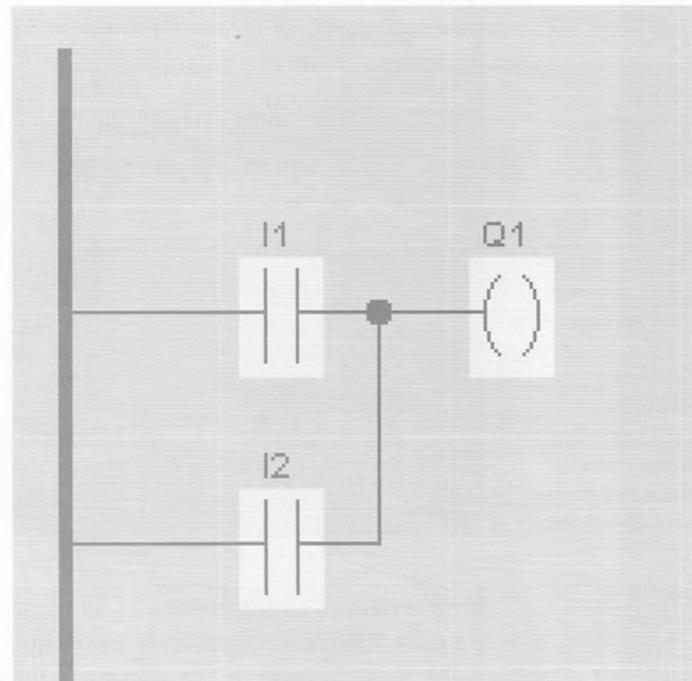


Fig. 11

Si activamos el simulador y accionamos los dos contactos, veríamos lo siguiente:

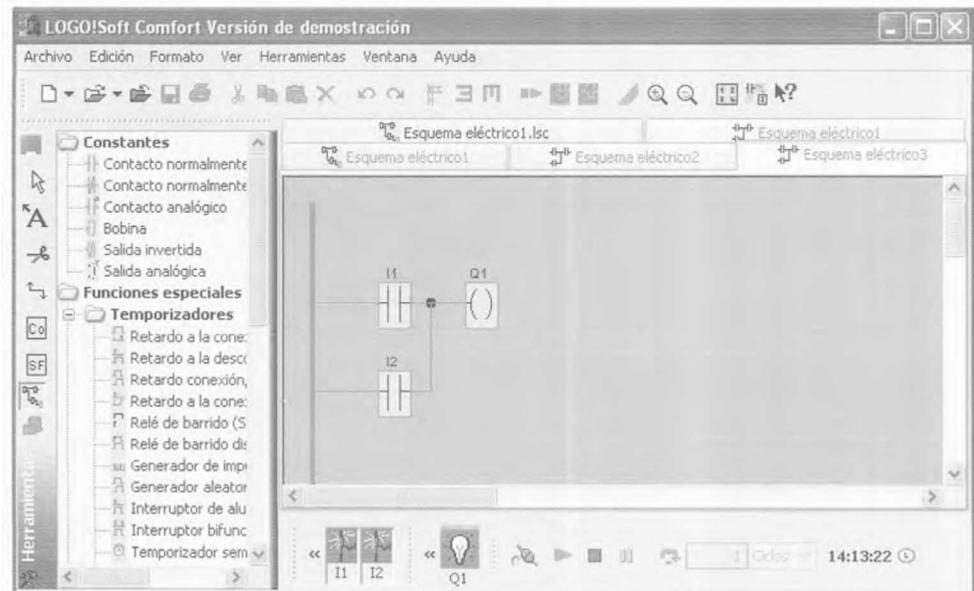


Fig. 12

Una vez el programa hecho, tenemos la posibilidad de seleccionar el LOGO que necesitamos. En el menú herramientas, disponemos la opción "Determinar LOGO". Si pulsamos aquí, abajo en la ventana de información, se nos indica la referencia del LOGO que necesitamos para el programa realizado. Hay que tener en cuenta que esto no es un PLC y que no tenemos posibilidades de ampliación a base de tarjetas.

Ejercicio 3

1.4.3. Temporizadores Aplicación "semáforo"

Para programar un temporizador, tenemos las siguientes opciones:

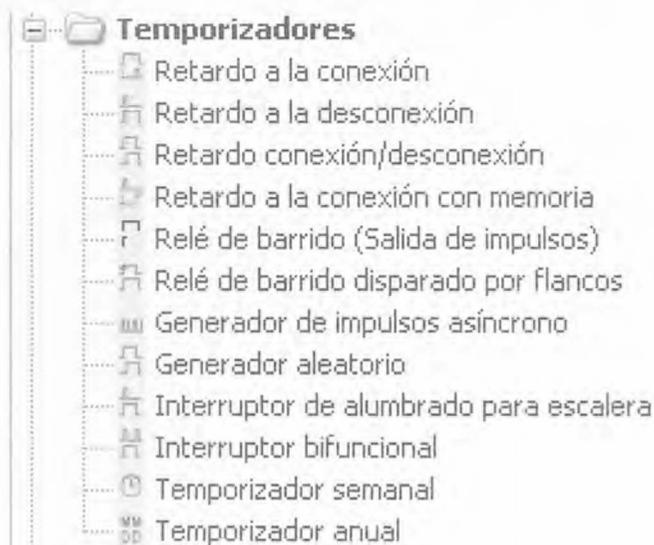


Fig. 13

Encontramos los temporizadores dentro de las funciones especiales.

En el icono de cada temporizador, tenemos un pequeño esquema de cómo funciona. Veamos a modo de ejemplo cómo programamos un temporizador de retardo a la conexión.

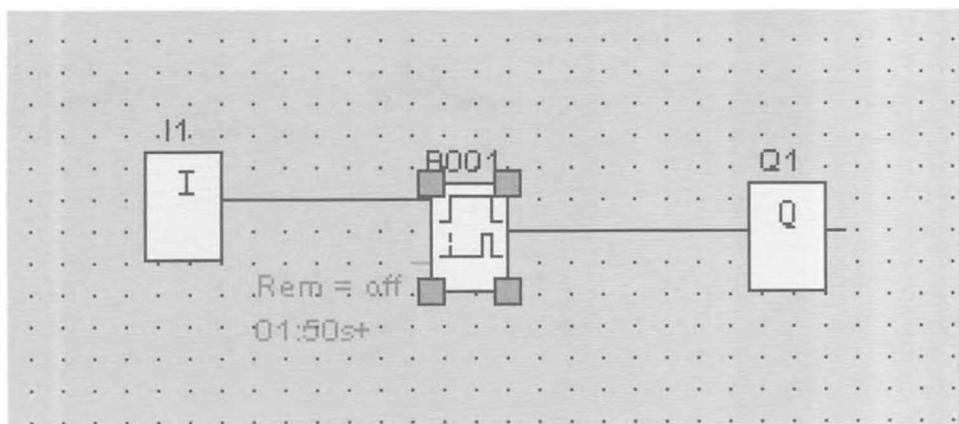


Fig. 14

Lo que hace este programa, es que al cabo de 1,5 segundos de haber activado la entrada 1, se activará la salida 1. Si hacemos doble clic sobre el temporizador, podemos seleccionar el tiempo que queremos ajustar.

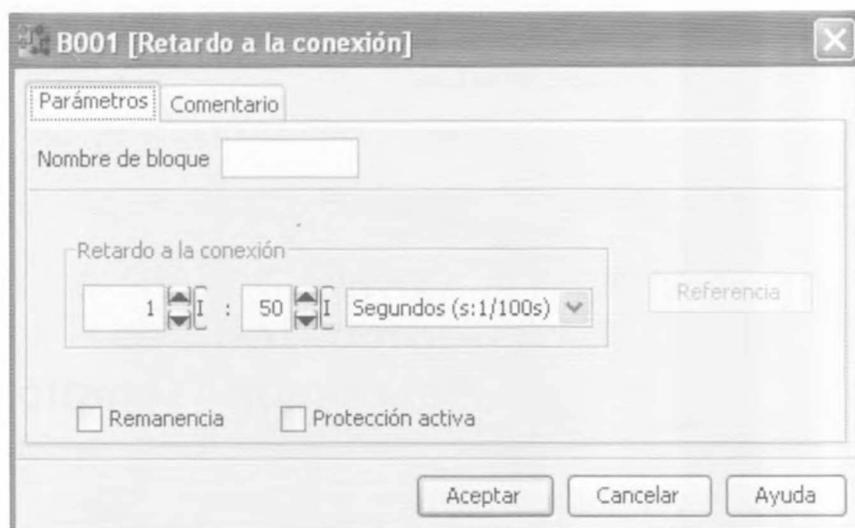


Fig. 15

Aquí podemos elegir la unidad que queremos seleccionar y el tiempo que queremos ajustar.

Con la opción de simulación, podemos probar qué funciona, tal y como lo hemos explicado.

Veamos cómo podemos hacer un semáforo utilizando este temporizador. Queremos que al pulsar marcha (I1), se ponga el semáforo en marcha encendiendo la luz verde (Q3) durante 5 segundos. Al cabo de 5 segundos, queremos que se encienda el amarillo (Q2) durante 2 segundos. Pasados los dos segundos, queremos que se encienda la luz roja (Q1) durante 6 segundos. Pasado este tiempo queremos que vuelva a comenzar el ciclo encendiendo la luz verde. Se mantendrá el ciclo hasta que pulsemos paro (I2) con lo que se apagarán todas las luces del semáforo.

El programa hecho con LOGOSOFT quedaría así:

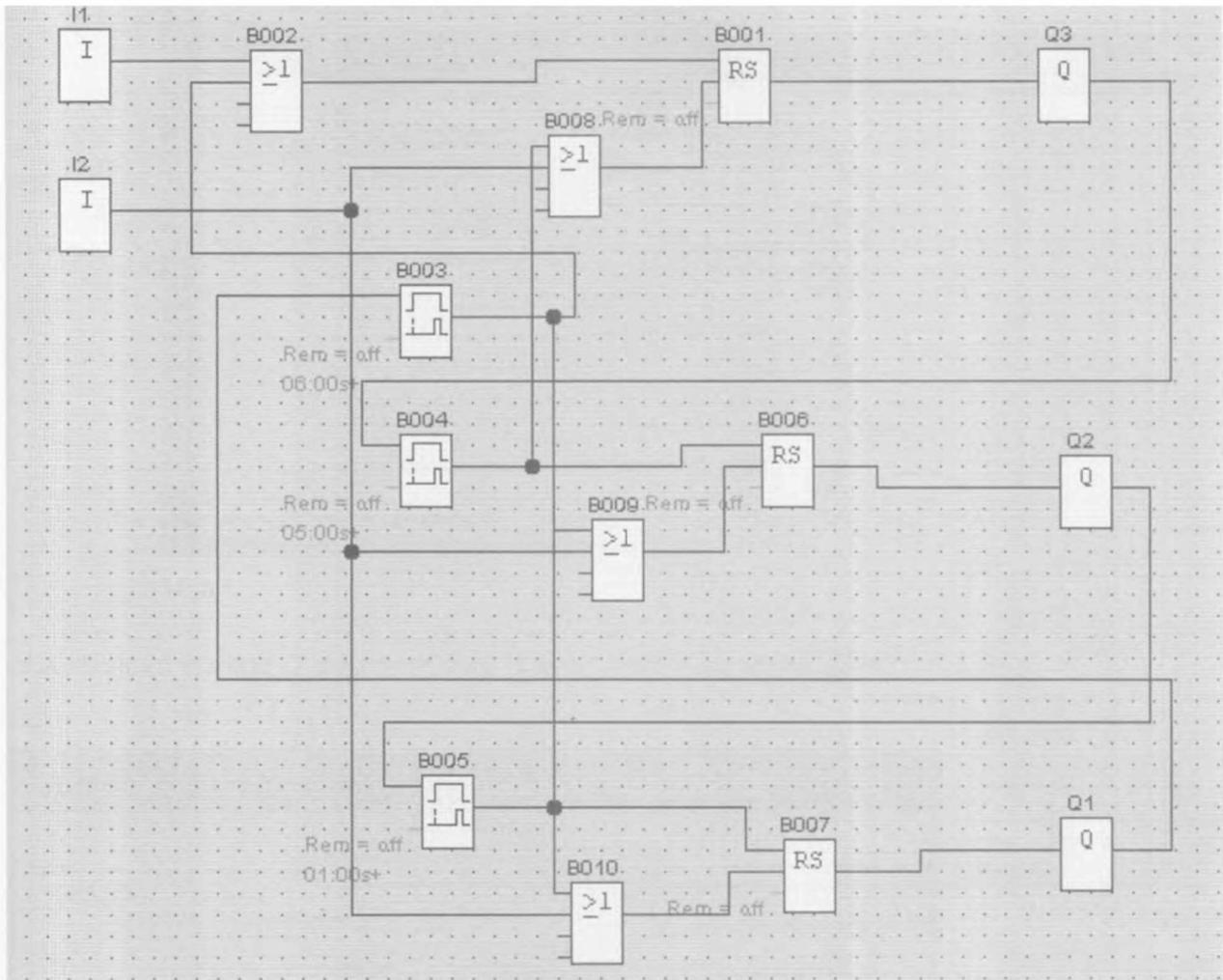


Fig. 16

Ejercicio 4



1.4.4. Temporizador semanal

Además de los temporizadores que hemos visto para el ejercicio anterior, el LOGO dispone de un temporizador semanal y un temporizador anual. Son muy prácticos para realizar funciones repetitivas dentro de la semana o del año. También para hacer algo en una determinada fecha o un día determinado.

Con el temporizador semanal, que es el que vamos a ver en este ejemplo, podemos programar, dentro de la semana de lunes a domingo, hasta 3 franjas de tiempo para que el aparato haga algo. Imaginemos que queremos encender las luces de un jardín, de lunes a jueves de 8 de la tarde a 10 de la noche, y viernes, sábado y domingo de 8 de la tarde a 12 de la noche.

Para ello ponemos en la zona de programación un temporizador semanal y entramos en sus parámetros haciendo doble clic sobre él. Veremos una ventana de parametrización como la que mostramos aquí:



Fig. 17

Entramos dentro de la ficha "Leva 1" para programar la primera franja de activación.

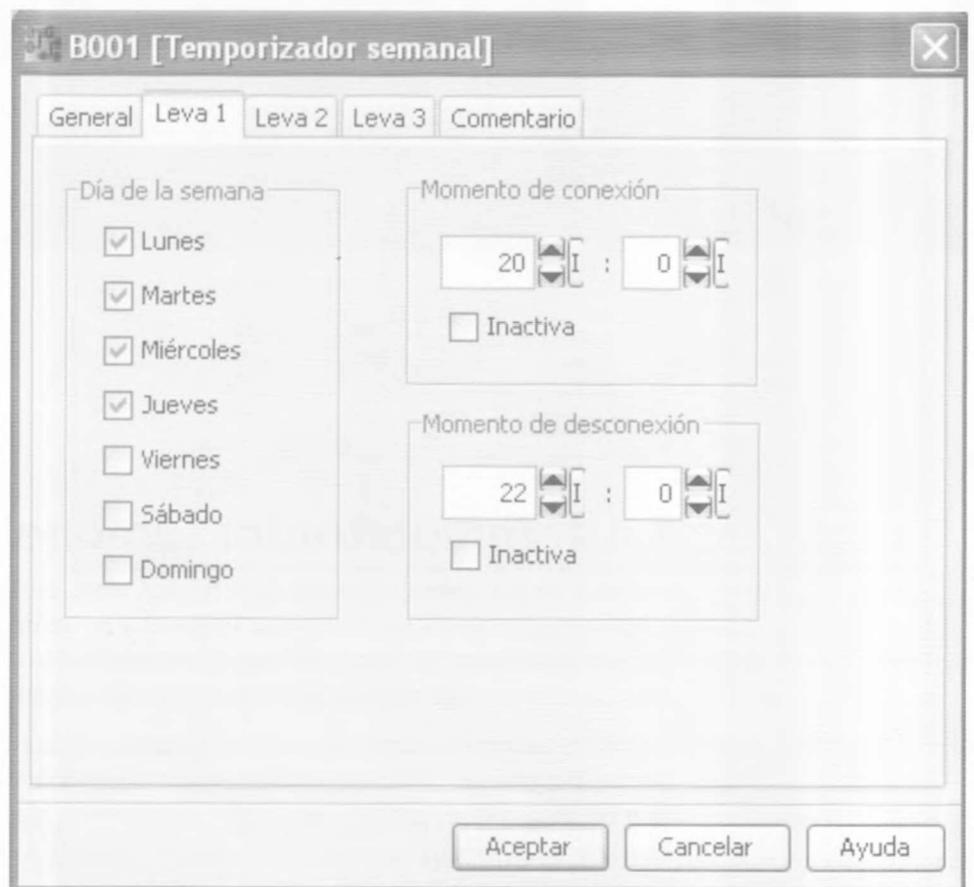


Fig. 18

Aquí hemos programado que queremos que se active la salida del temporizador de lunes a jueves de 8 de la tarde a 10 de la noche.

Ahora nos vamos a la ficha de "Leva 2" para activar la segunda franja horaria que queremos:

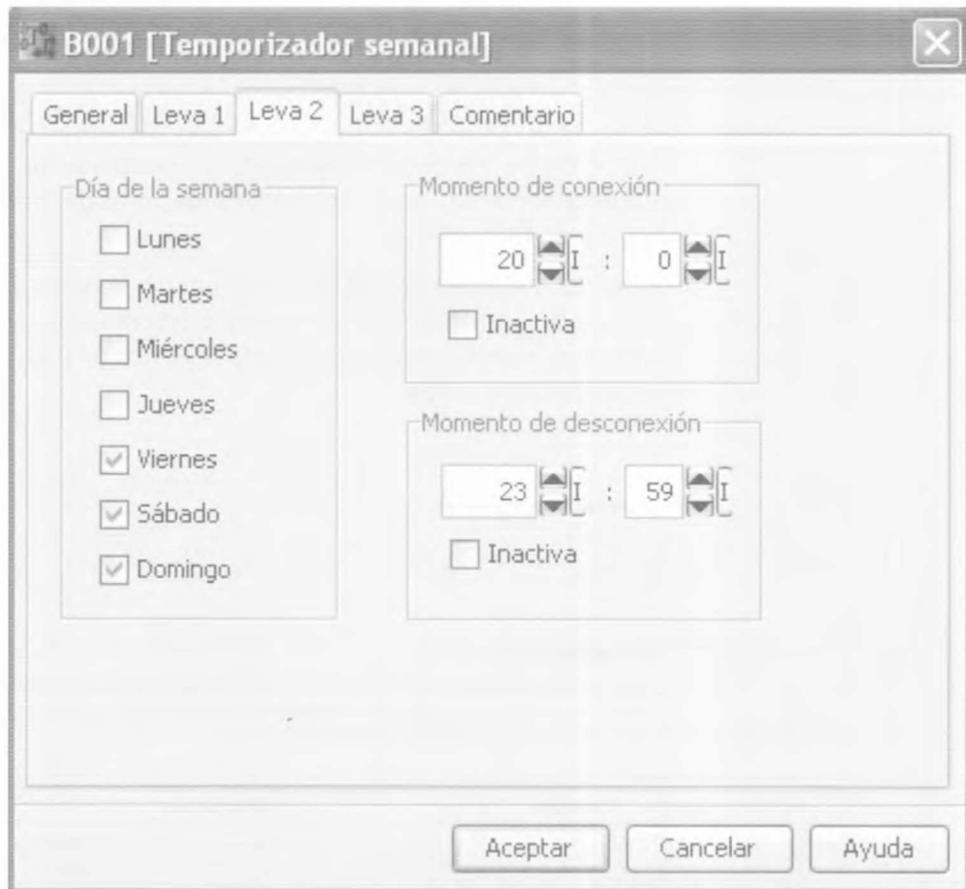


Fig. 19

La ficha de "Leva 3" la dejamos inactiva ya que no necesitamos más franjas para programar lo que nos indica el enunciado del ejemplo.

Ahora sólo tenemos que conectar la salida de las luces del jardín, y ya tenemos el ejemplo terminado:

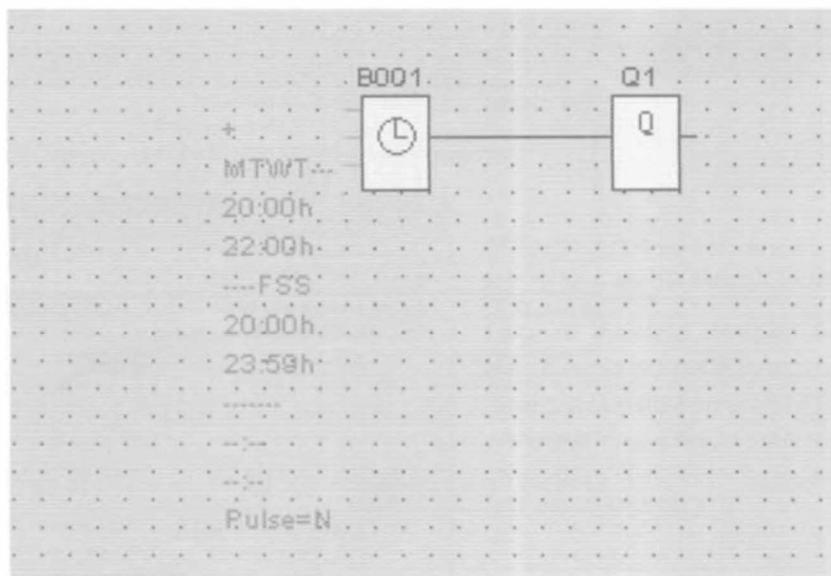


Fig. 20

Ejercicio 5



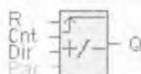
1.4.5. Contadores

Aplicación *parking* coches

Dentro de las funciones especiales, encontramos los contadores. Tenemos un contador adelante / atrás, un contador de horas de funcionamiento y un selector de umbral.

Si cogemos el interrogante que tenemos en la barra de herramientas y pinchamos sobre la función que queremos, obtenemos la ayuda de cómo funciona y que parámetros debemos introducirle. Vemos un ejemplo con la ayuda del contador adelante / atrás:

Contador adelante/atrás



Descripción breve

Según la parametrización, un impulso de entrada incrementa o decrementa un valor de contaje interno. Cuando se alcanzan los umbrales parametrizables se define o se resetea la salida. El sentido de contaje puede cambiarse mediante la entrada Dir.

Conexión	Descripción
Entrada R	Por medio de la entrada R (Reset), el valor de contaje interno y la salida Q se ajustan al valor inicial (StartVal).
Entrada Cnt	La función cuenta en la entrada Cnt los cambios de estado de 0 a 1. Los cambios de estado de 1 a 0 no se cuentan. <ul style="list-style-type: none"> ● Utilice las entradas I3, I4, I5 y I6 para contajes rápidos (sólo en algunos LOGO! 12/24 RC/RCO y LOGO! 24/24a): máx. 5 kHz ● Utilice cualquier otra entrada o un elemento del circuito para contajes lentos (típ. 4 Hz).
Entrada Dir	Por medio de la entrada Dir (Direction) se especifica el sentido de contaje: Dir = 0: Adelante Dir = 1: Atrás
Parámetros	On: Umbral de conexión Rango de valores: 0...999999 Off: Umbral de desconexión Rango de valores: 0...999999 StartVal: Valor inicial a partir del cual se cuenta adelante o atrás Remanencia activada (ON) = el estado se guarda de forma remanente.
Salida Q	Q se activa o desactiva en función del valor real Cnt y de los valores umbral ajustados.

Fig. 21

En esta figura podemos ver parte de la ayuda que ofrece el software.

Aquí nos explica los parámetros y las entradas y salidas necesarias para programar el contador.

Aprovechando este contador, vamos a programar un *parking* para coches. Imaginemos lo siguiente:

Tenemos un *parking* en el que nos caben 10 coches. Tenemos una fotocélula a la entrada (I1) y una fotocélula a la salida (I2) que controlan el flujo de los coches.



Fig. 22

Tenemos también dos salidas que nos indican si el *parking* está libre (Q1) u ocupado (Q2).

Queremos programar las salidas luminosas en función de si tenemos el *parking* lleno o todavía quedan plazas libres.

Para ello tenemos que utilizar el contador que hemos visto anteriormente.

Veamos como quedaría el programa resuelto con LOGO:

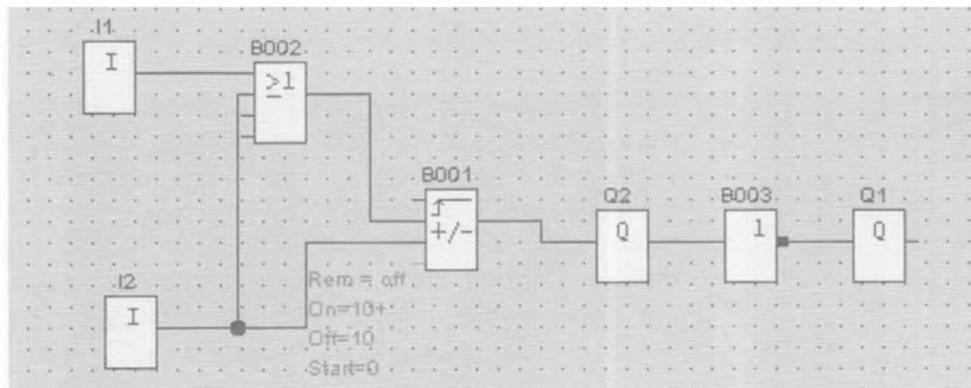


Fig. 23

Los parámetros del contador serán los siguientes:



Fig. 24

Al llegar a 10 se activará su salida. Por debajo de 10 estará desactivada.

En este ejemplo sencillo suponemos que no entran y salen coches a la vez, y que no entrarán más de 10 en el *parking*. Este mismo ejercicio un poco más elaborado, lo tenemos resuelto en **STEP 7**.

Ejercicio 6



1.4.6. Algunas aplicaciones del Logosoft

Dentro del menú “Herramientas”, encontramos la opción “Selección de dispositivos...”



Fig. 25

En esta ventana, podemos ver las propiedades que tiene cada uno de los LOGO que hay en el mercado, y seleccionar el que nos interesa para nuestra aplicación. Aquí podemos ver la memoria de que disponemos, la cantidad máxima de bloques que podemos programar, los parámetros que podemos utilizar, la cantidad de entradas, salidas y marcas que tenemos disponibles, y la profundidad de pila máxima. La profundidad de pila, es la cantidad de bloques que podemos poner entre la primera condición y la salida o marca que queremos actuar. Es más o menos la longitud máxima del segmento.

En el menú “Herramientas -> Opciones”, podemos ver la siguiente ventana:

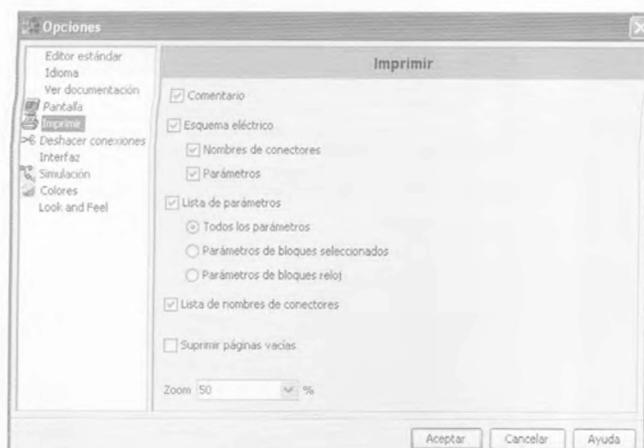


Fig. 26

Aquí podemos seleccionar diversas preferencias de usuario. Podemos seleccionar los colores de visualización, el puerto por el que comunicamos, el editor que queremos por defecto (KOP o FUP) y los parámetros de impresión, entre otras cosas.

En la barra de herramientas de la parte izquierda de la pantalla, tenemos la opción de añadir un texto en el esquema eléctrico.

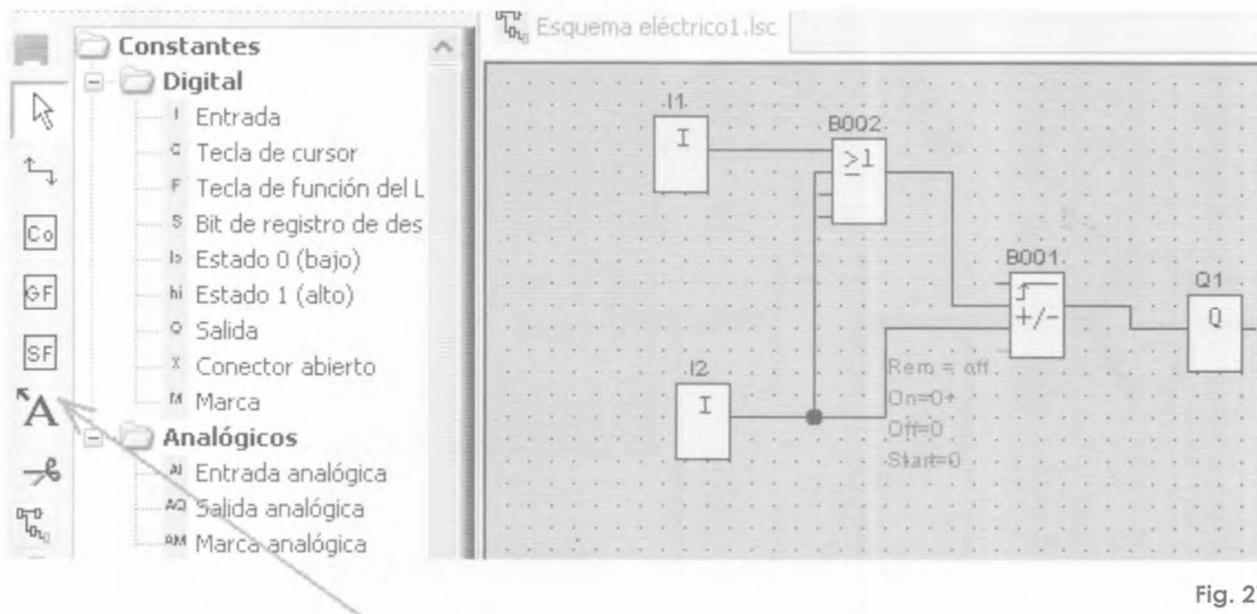


Fig. 27

Con esta opción podríamos añadir comentarios como los que vemos en el ejemplo:

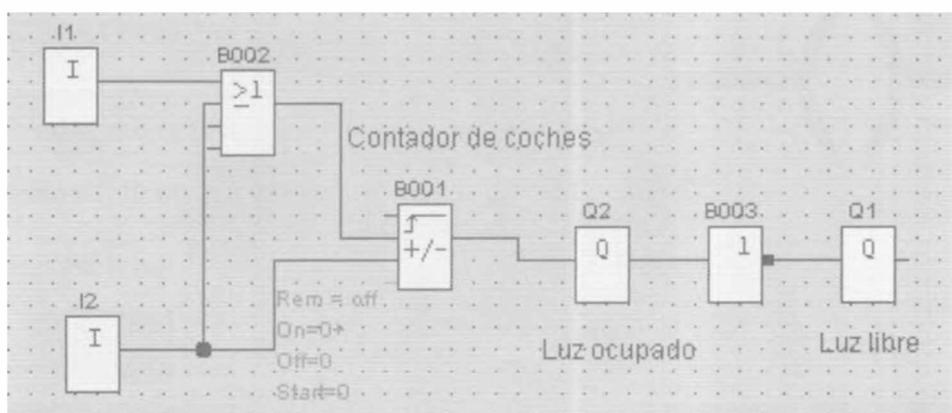


Fig. 28

Otra opción práctica de la que disponemos en el *software*, es la opción "Ir a bloque" que encontramos dentro del menú "Edición".

Si seleccionamos esta opción, nos aparece un listado de todos los bloques que tenemos utilizados en el programa.



Fig. 29

Si seleccionamos aquí sobre el bloque que queremos ver, el *software* nos lo marcará en rojo en el esquema del programa:

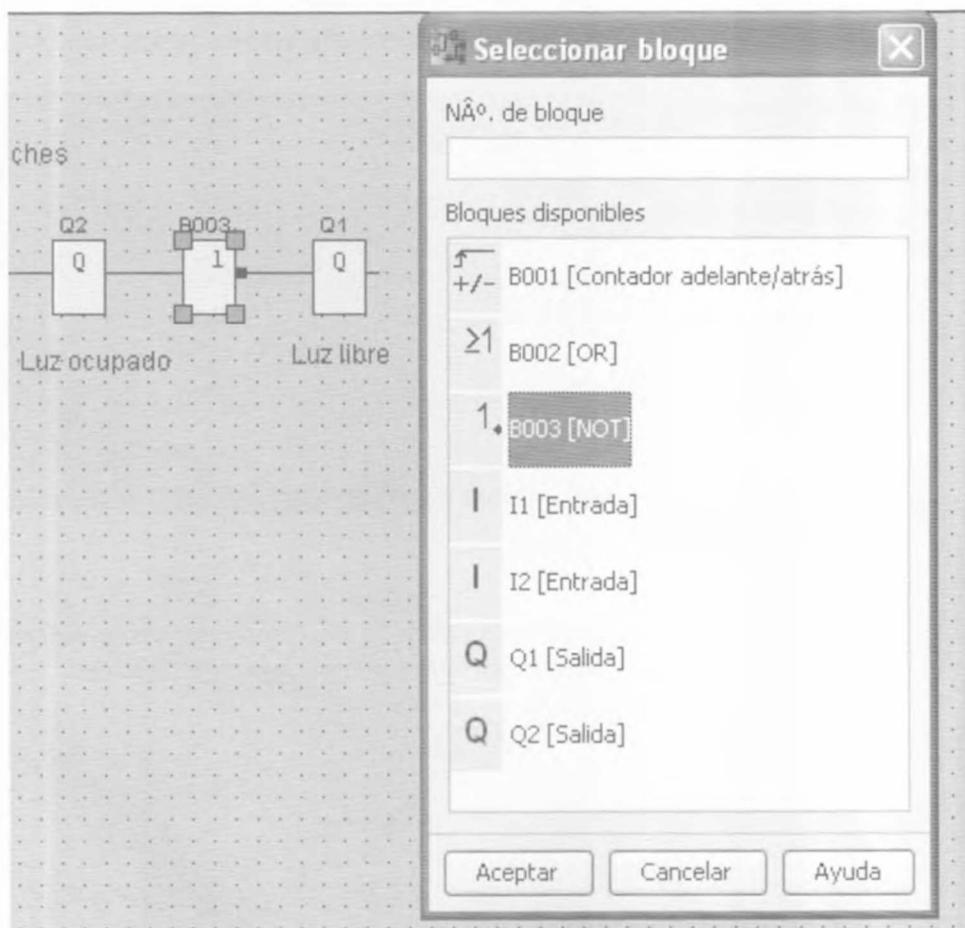
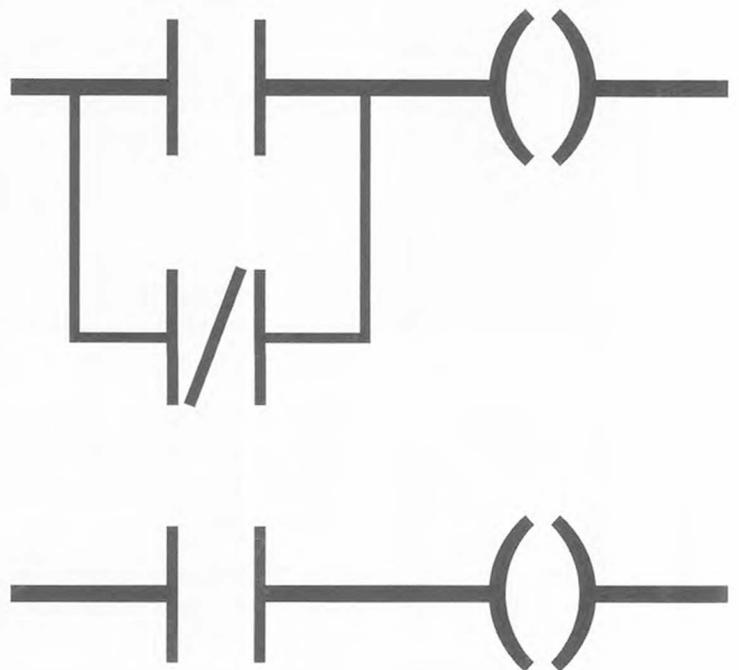


Fig. 25

Unidad 2

Ejemplos con operaciones de *bit*, instrucciones binarias, temporizadores y contadores



En este capítulo:

- 2.1. Creación del primer proyecto
- 2.2. Contactos en serie
- 2.3. Contactos en paralelo
- 2.4. Utilización del paréntesis
- 2.5. Contactos negados
- 2.6. Marcas internas
- 2.7. Instrucciones SET y RESET
- 2.8. Opción TEST > OBSERVAR
- 2.9. Tabla OBSERVAR / FORZAR VARIABLE
- 2.10. Depósito de agua
- 2.11. Semáforo
- 2.12. Simbólico global
- 2.13. Cintas transportadoras
- 2.14. Intermitente
- 2.15. Semáforo con intermitencia
- 2.16. *Parking* de coches
- 2.17. Puerta corredera
- 2.18. Contar y descontar cada segundo
- 2.19. Fábrica de curtidos
- 2.20. Escalera automática
- 2.21. Instrucción MASTER CONTROL RELAY

Recuerda . . .

Dentro de un proyecto de Step 7 introduciremos todos los equipos que tengamos en la instalación. Tanto PLC como equipos de visualización, PC u otros elementos de campo. De este modo será muy fácil establecer las relaciones entre ellos.

2.1 Creación del primer proyecto

Vamos a realizar el primer proyecto en **STEP 7**. Para ello se necesita disponer de un *software* **STEP 7** con su licencia o una versión demo. También es posible tener una copia de un *software* **STEP 7** sin licencia. El inconveniente de esto es que cada vez que intentemos guardar algún cambio nos saldrá un mensaje advirtiéndonos que el *software* utilizado no tiene llave. Dependiendo de qué versión del *software* estemos utilizando también puede ser que no nos deje ni siquiera abrir el programa si no tenemos licencia. Los ejercicios expuestos en este manual están resueltos con la versión 5.3 del *software* con el SP 2. También es posible realizar los mismos ejercicios con versiones anteriores. Algunas pantallas han cambiado de aspecto pero las instrucciones y opciones explicadas en este manual también son válidas para las otras versiones.

Dentro de un proyecto de **STEP 7** introduciremos todos aquellos equipos que vayan a formar parte de nuestra instalación. Introduciremos tanto los equipos de **SIEMENS** que estemos gastando, como los equipos de otras marcas que formen parte del proyecto. También los PC, los equipos de visualización y las diferentes redes en caso de que las hubiera. De este modo, podremos comunicarlo todo de modo sencillo y podremos visualizar todos los equipos de la instalación con un solo PC desde un solo punto de la instalación.

Además, con el *software* NETPRO (incluido en el **STEP 7** a partir de la versión 5.x) podremos visualizar de forma gráfica, las redes y conexiones entre los diferentes equipos. Además podremos manejar estos enlaces de los equipos a las redes existentes con el ratón del PC de forma gráfica e intuitiva.

En el primer proyecto que vamos a realizar en el curso, vamos a insertar un solo equipo (un PLC de **SIEMENS**). En siguientes capítulos se insertará más de un equipo incluyendo PLC, reguladores de frecuencia, sistemas de visualización, ordenadores, etcétera.

Dentro del equipo de **SIEMENS** que vamos a insertar en este primer ejemplo, vamos a incluir tanto el *hardware* que estemos utilizando, como los bloques de programación (programa propiamente dicho).

La inclusión del *hardware* en el proyecto nos aportará varias ventajas. El ordenador “sabrà” el equipo que vamos a gastar en nuestro trabajo. Además “sabrà” las tarjetas que tenemos instaladas y las opciones de las que dispone cada tarjeta. Si intentamos utilizar alguna instrucción que no soporta nuestra CPU, nos avisará indicándonos que aquella instrucción es imposible que sea utilizada en este proyecto. Además cuando entremos en la opción “Propiedades del objeto” de cualquier módulo que tengamos en la instalación, tendremos disponibles las propiedades de ese equipo en concreto.

Además tendremos la posibilidad de ajustar desde el *software* propiedades del propio *hardware*. Por ejemplo nos permitirá cambiar las alarmas, el tiempo de ciclo de scan preestablecido para la propia CPU, las direcciones de cada uno de los objetos de periferia, etc. Se explicará mejor todos estos puntos en los ejercicios dedicados a ello.

Cada vez que hagamos un proyecto nuevo tendremos que definir un *hardware* nuevo para cada uno de los equipos que tengamos en la red. Este paso no es estrictamente necesario. Veremos en este mismo manual que es posible crear y programar proyectos sin necesidad de definir un *hardware*. El único inconveniente de esto es que no podremos acceder o modificar las propiedades de cada módulo.

Veamos cómo haríamos esto en la práctica.

Abrimos el Administrador de **SIMATIC**. Si es la primera vez que se utiliza el *software*, automáticamente se abrirá el asistente de nuevo proyecto.

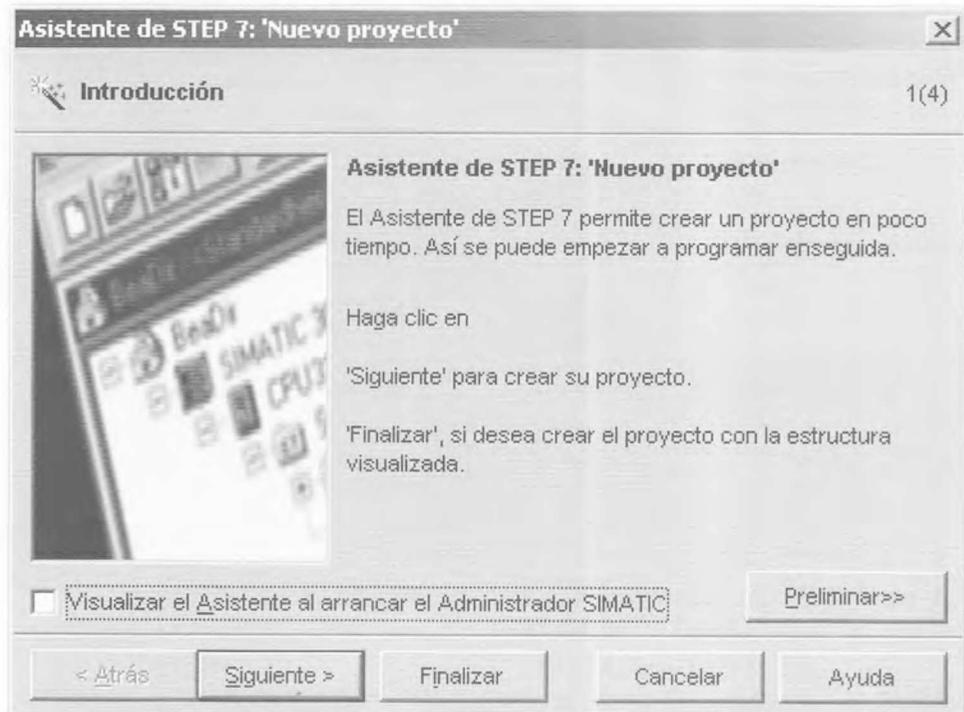


Fig. 1

Si hemos utilizado el *software* más veces, se abrirá mostrando el último proyecto que se utilizó y no se cerró expresamente. También puede ser que en la ventana del asistente se haya seleccionado la pestaña de visualizar el asistente al arrancar el Administrador de **SIMATIC** y la veamos siempre. Si tenemos un proyecto abierto, el aspecto del Administrador de **SIMATIC** será algo parecido a la figura que vemos a continuación:

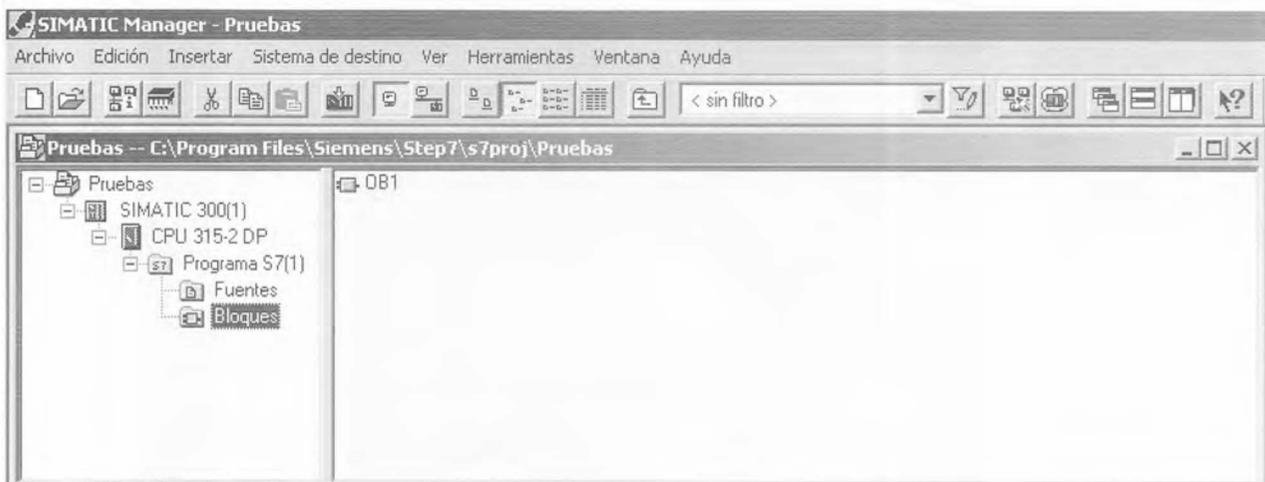


Fig. 2

En el ejemplo vemos que se ha abierto el **STEP 7** y el proyecto “**Pruebas**”. Esto quiere decir que la última vez que se cerró **STEP 7**, se hizo sin cerrar previamente la ventana del proyecto “**Pruebas**”.

Si se utilizó el **STEP 7** y se cerró el Administrador después de haber cerrado el proyecto en uso, veremos una pantalla en blanco (o gris) desde la cual podremos abrir un proyecto existente o crear uno nuevo.

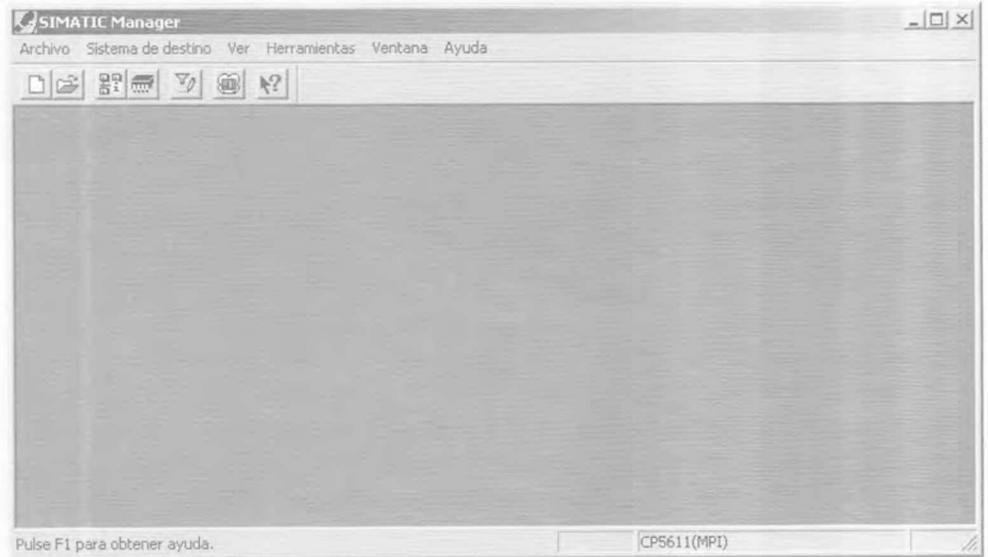


Fig. 3

También podremos abrir el asistente si así lo queremos. Si no se abre el asistente pero queremos utilizarlo, deberemos acceder al menú **“Archivo → Asistente ‘Nuevo proyecto’ ...”**, después de haber cerrado el proyecto abierto en caso de que lo hubiese.

Si queremos utilizar el asistente, no tenemos más que ir contestando a lo que se nos pregunta. Si queremos, podemos cancelar el asistente y generar nosotros nuestro nuevo proyecto.

El primer ejemplo lo haremos utilizando el asistente.

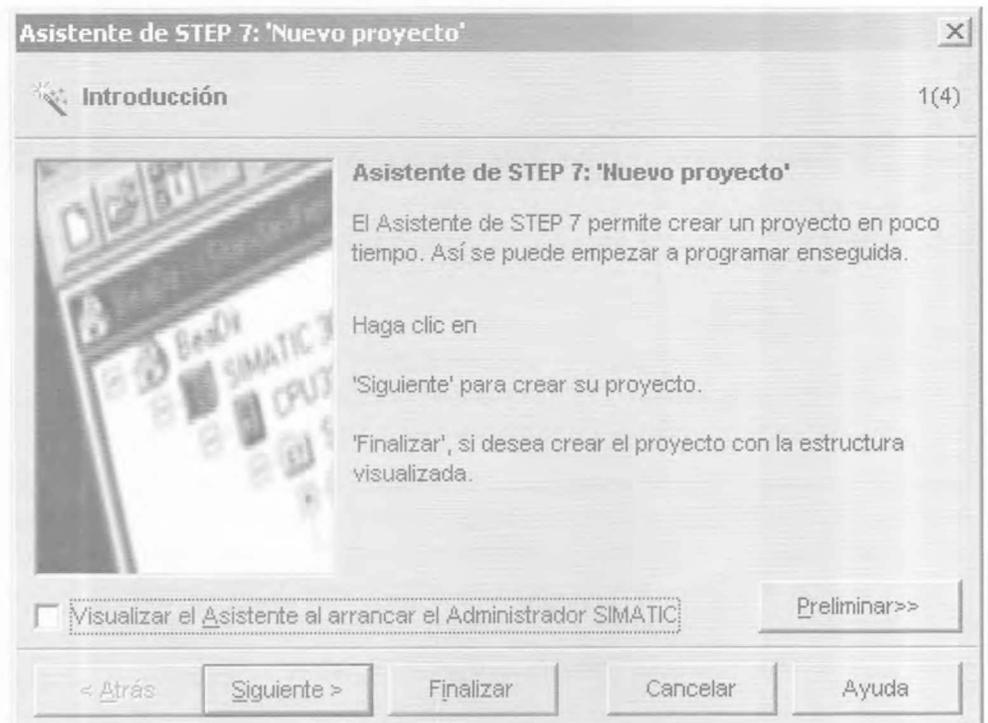


Fig. 4

Si hacemos el proyecto con el asistente, no tendremos en él las tarjetas de entradas y salidas que estamos gastando. Tampoco tendremos incluidos otros equipos que queramos utilizar o comunicar. Se generará un proyecto estándar en el que no dispondremos de los equipos reales con los que vamos a trabajar. Es posible programar con un proyecto generado así si nos adaptamos a la configuración estándar de los equipos. Si queremos cambiar alguna de las opciones configurables en los equipos, deberemos incluirlos de forma manual en nuestro proyecto.

Veamos como quedaría el proyecto generado con el asistente. Posteriormente crearemos uno manualmente y observaremos las diferencias.

Si pulsamos el botón “**Siguiente >**” en la primera pantalla del asistente, observaremos el siguiente menú:

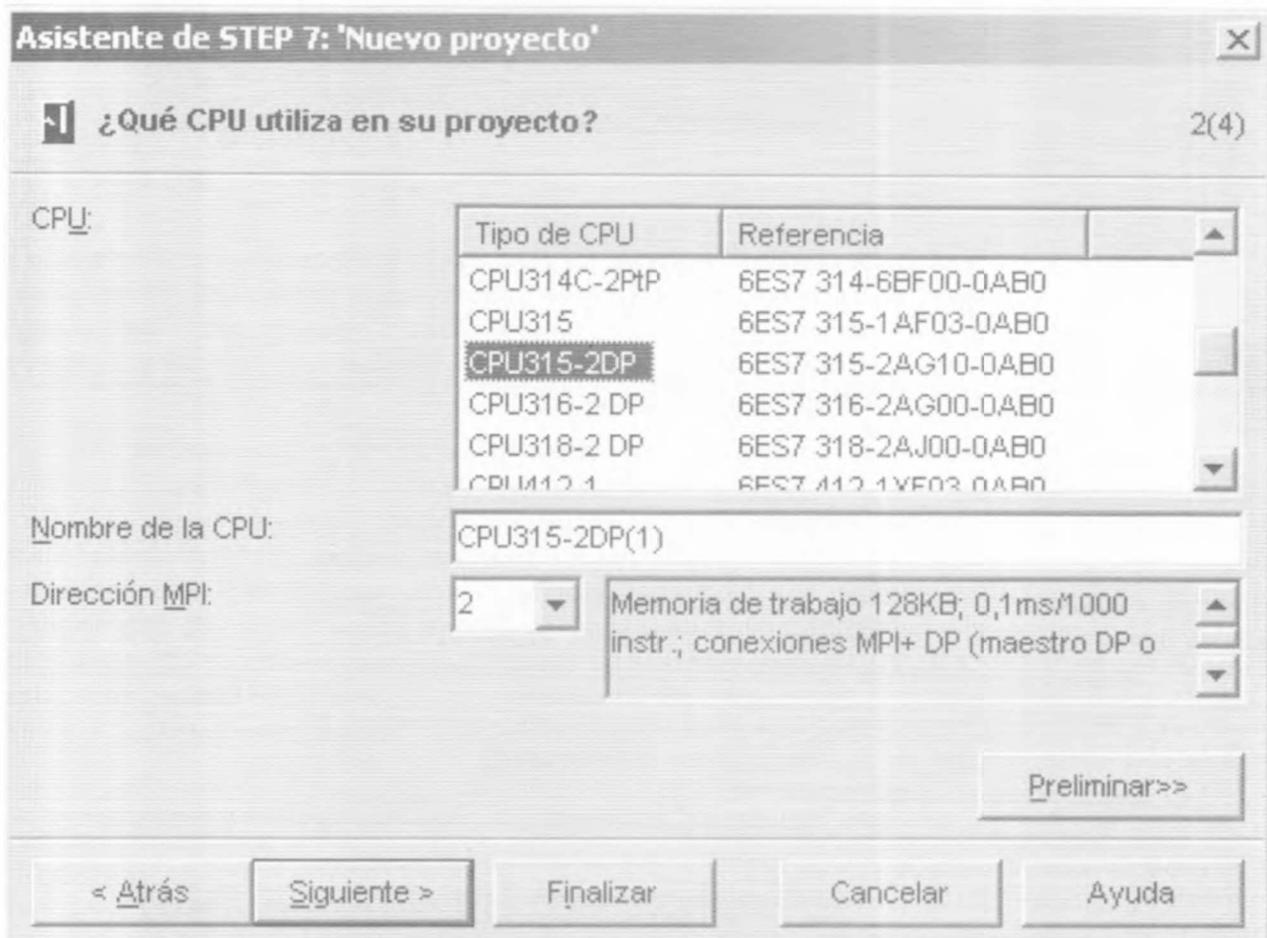


Fig. 5

En esta ventana deberemos seleccionar la CPU que vamos a utilizar en el proyecto. En el ejemplo hay seleccionada una **CPU 315-2DP**. También podemos asignarle una dirección MPI y un nombre. De momento mientras sólo tengamos una CPU la dirección MPI es irrelevante. Podemos dejar la 2 que es la que viene por defecto. El nombre lo dará el usuario. Se recomienda utilizar un nombre que identifique a la CPU dentro de la instalación. En el ejemplo le hemos dejado el nombre por defecto ya que de momento no estamos programando nada en concreto.

Si pulsamos “**Siguiente >**” accedemos a esta ventana:

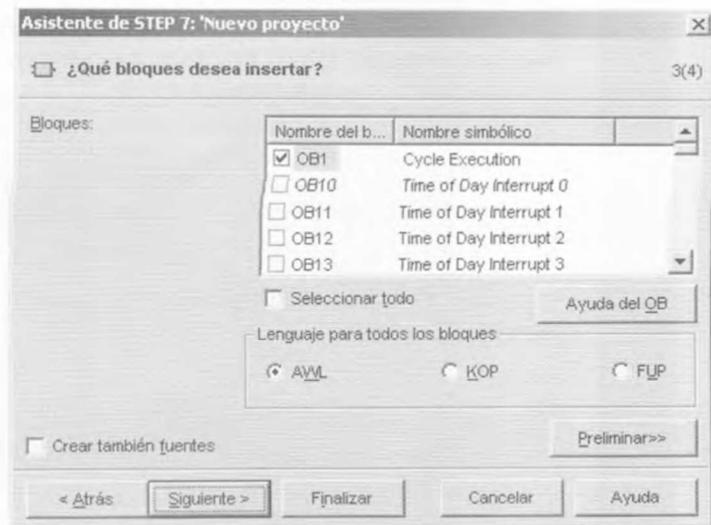


Fig. 6

Aquí se nos permite seleccionar los bloques que queremos tener en el proyecto a priori. En principio por defecto viene seleccionado el bloque OB1 (bloque principal). De momento empezaremos trabajando los primeros proyectos sólo con este bloque. En posteriores capítulos se explicará para qué sirven los demás bloques y se realizarán ejemplos utilizándolos.

También se nos da la opción de seleccionar el tipo de lenguaje que queremos utilizar. Por defecto viene seleccionado AWL (listado de instrucciones). En próximas páginas explicaremos los tres lenguajes incluyendo ejemplos. Aunque aquí seleccionemos uno de los tres lenguajes disponibles, siempre podremos cambiarlo una vez entremos en el bloque para programar. Incluso si se respetan unas sencillas reglas podremos cambiar de lenguaje los bloques una vez hayan sido programados. Conforme vayamos avanzando en la teoría de programación, se irán explicando estas pequeñas normas con las cuales siempre se tendrán los bloques traducibles a los tres lenguajes. De momento en el ejemplo dejamos seleccionado el lenguaje AWL.

Si pulsamos “**Siguiente >**” accedemos a la pantalla que vemos a continuación:

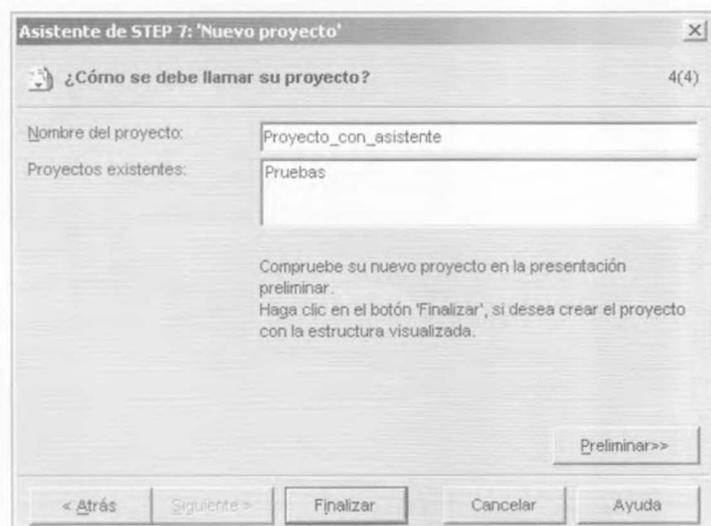


Fig. 7

En esta última ventana se nos permite dar un nombre al proyecto. En el ejemplo se ha llamado "**Proyecto_con_asistente**". En la ventana inferior se observan los proyectos de **STEP 7** existentes en el ordenador en el que estamos trabajando y que se encuentran almacenados en la misma carpeta que vamos a guardar el proyecto que estamos creando.

Una vez tengamos escrito el nombre que queremos para el proyecto, pulsamos el botón "**Finalizar**" y veremos el proyecto que se ha generado con la ayuda del asistente.

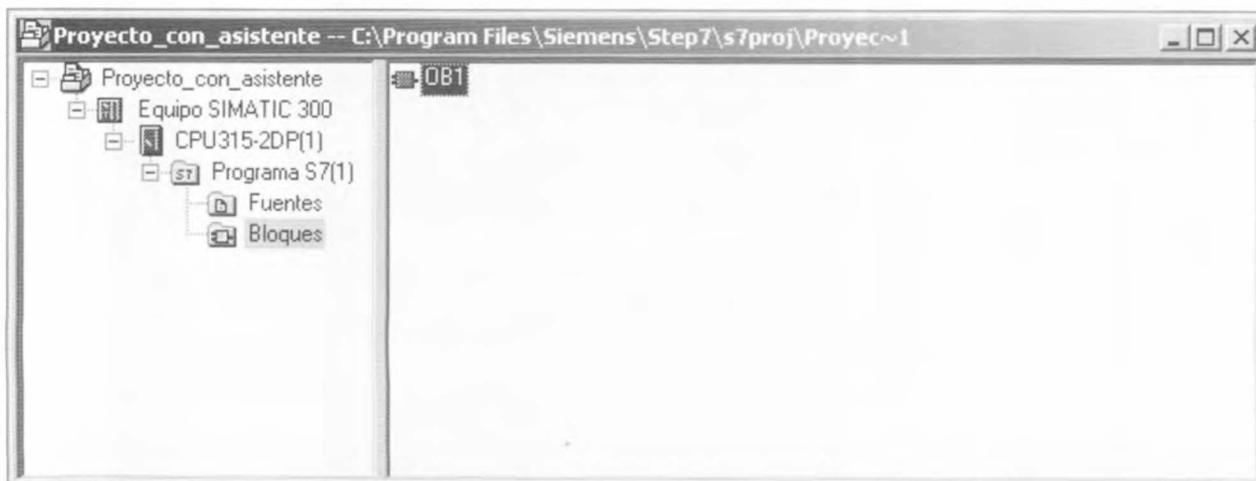


Fig. 8

Esto es una ventana típica de un proyecto en **STEP 7**. De momento tenemos un proyecto sencillo en el que sólo tenemos un PLC con una CPU y un solo bloque de programa. Aunque en el futuro tengamos más equipos y más bloques de programa en los proyectos, la estructura de esta ventana veremos que será la misma. En una misma ventana de proyecto visualizaremos todos los equipos.

Ya hemos visto como se crea un proyecto con el asistente del **STEP 7**. Ahora vamos a cerrar este proyecto y vamos a generar uno manualmente. A la hora de cerrar el proyecto observaremos que tenemos abiertas dos ventanas. Una es el Administrador de **SIMATIC** y otra es el proyecto que acabamos de crear. Ahora queremos cerrar el proyecto pero dejar abierto el Administrador de **SIMATIC**. A lo largo de este curso veremos que dentro del Administrador podremos abrir muchas subventanas con diferentes aplicaciones dentro del Administrador **SIMATIC**.

Una vez tengamos los dos proyectos creados, analizaremos las diferencias.

Para generar un nuevo proyecto, podemos ir bien al menú "**Archivo → Nuevo...**", o bien al botón que tiene como icono una hoja en blanco .

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

En nuestro caso decimos que queremos un proyecto de usuario.

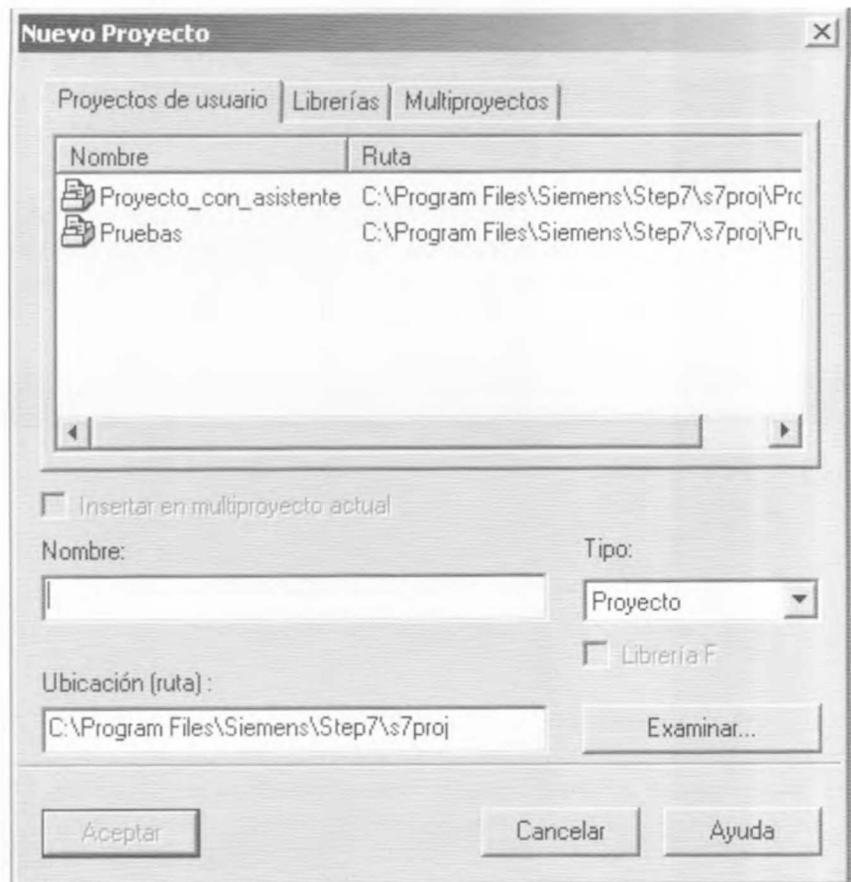


Fig. 9

En la parte superior de la ventana vemos un listado de los proyectos que ya existen en el ordenador que estamos utilizando y que hemos abierto alguna vez desde el **STEP 7**. En la parte inferior deberemos dar un nombre al proyecto que vamos a crear. En el ejemplo le llamaremos "Primer_proyecto". También podemos indicar la carpeta en la cual queremos que se quede almacenado. La que vemos en el ejemplo es la que utiliza la aplicación por defecto. Si queremos guardarlo en otra carpeta, no tenemos más que pulsar el botón "Examinar" y seleccionar la carpeta que nos interese.

Una vez introducido el nombre del proyecto, observaremos que tenemos una ventana con el nombre de nuestro proyecto en la parte izquierda y con la red MPI en la parte derecha. Ya tenemos un proyecto creado el cual de momento está vacío en cuanto a equipos y programa se refiere.

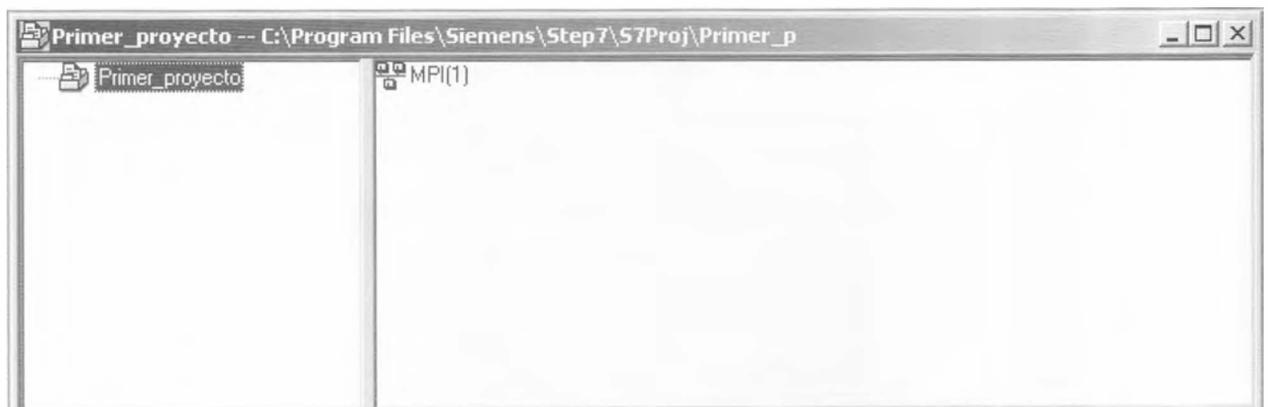


Fig. 10

El icono de la red MPI aparece por defecto. Es necesario que tengamos al menos una red MPI porque la primera programación de la CPU se hace a través del puerto MPI de la misma. En el primer proyecto que vamos a realizar tendremos una red MPI que constará de la CPU del PLC que queremos programar y el ordenador que utilizaremos para programarla.

Posteriormente podremos insertar tantas redes y equipos como nos haga falta.

Si observamos esta ventana, podemos apreciar que tiene el mismo aspecto que el proyecto que generamos con el asistente. Todas las ventanas de proyectos son iguales. La diferencia radica en los elementos que hayamos introducido en el proyecto.

Vamos a rellenar el proyecto con los elementos que vamos a utilizar en este manual de ejercicios. Para poder resolver los ejercicios de este manual, deberemos tener como mínimo una CPU 300, una tarjeta de 16 entradas digitales, una tarjeta de 16 salidas digitales y una tarjeta con 4 entradas y 2 ó 4 salidas analógicas.

Lo primero que tenemos que hacer es insertar los equipos que van a formar parte de nuestro proyecto. En este caso vamos a insertar un solo equipo. Para ello, vamos al menú de "Insertar" y elegimos el equipo con el que vayamos a trabajar. En los primeros ejemplos utilizaremos un equipo de la gama 300. En posteriores ejemplos utilizaremos una CPU 400 y veremos las diferencias entre un tipo y otro de CPU a la hora de realizar los proyectos.

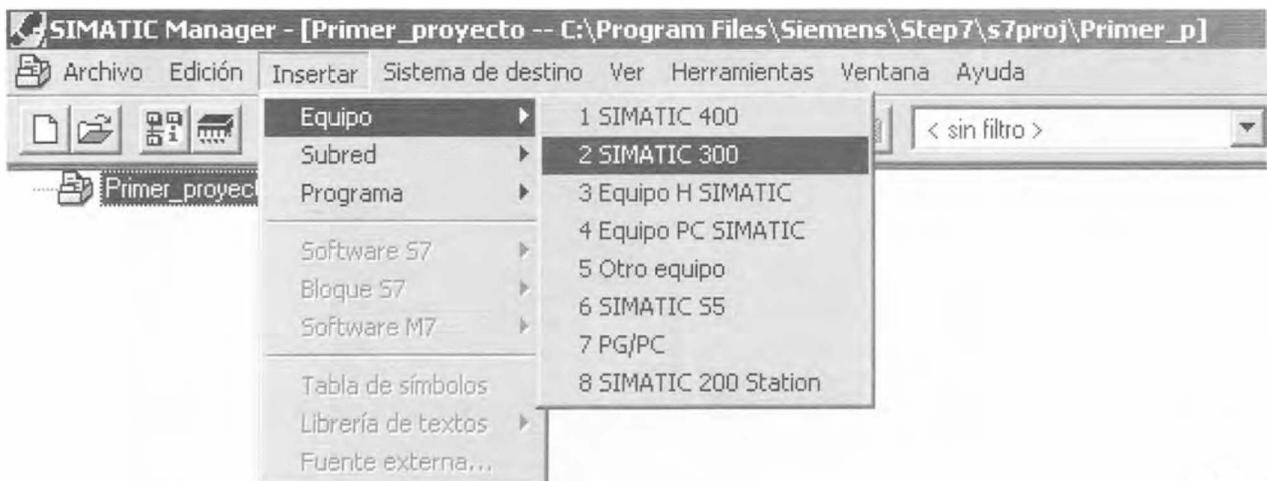


Fig. 11

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

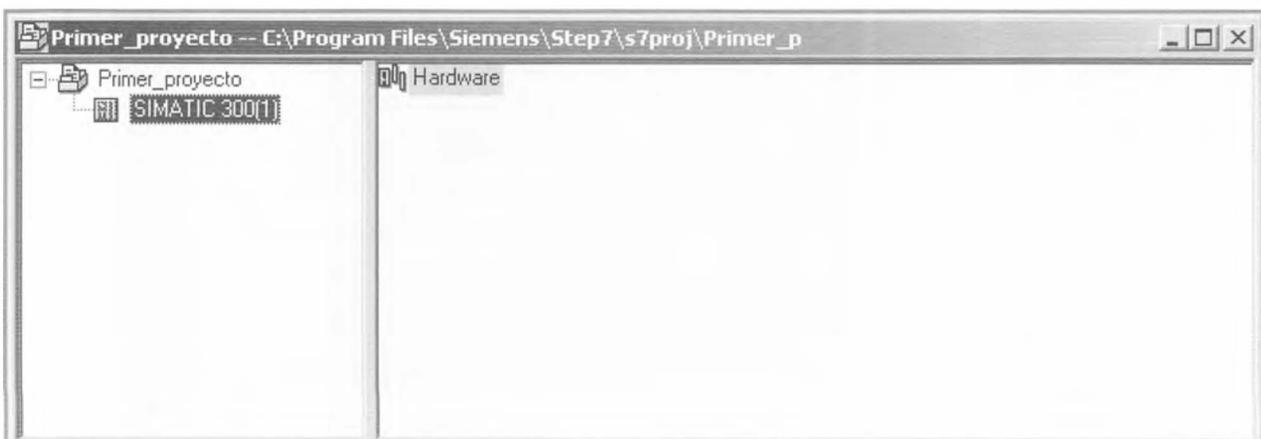


Fig. 12

Recuerda . . .

Al definir todo el hardware que estamos utilizando, siempre podremos acceder a las propiedades de los equipos y configurarlos de manera sencilla.

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

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

Lo primero que tenemos que insertar es un bastidor o perfil soporte. Es la pieza donde irán colocadas las tarjetas que utilizaremos en nuestro autómatas programable. Desplegamos la cortina de los bastidores y vemos que tenemos un perfil soporte.

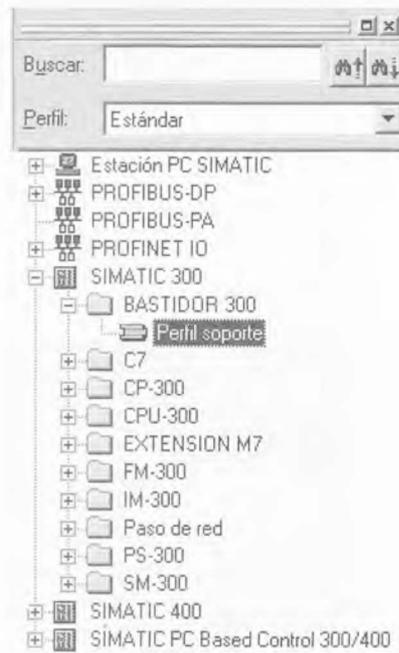


Fig. 13

Hacemos doble clic sobre el perfil soporte. Veremos que en la instalación del *hardware* se sitúa en la posición cero.

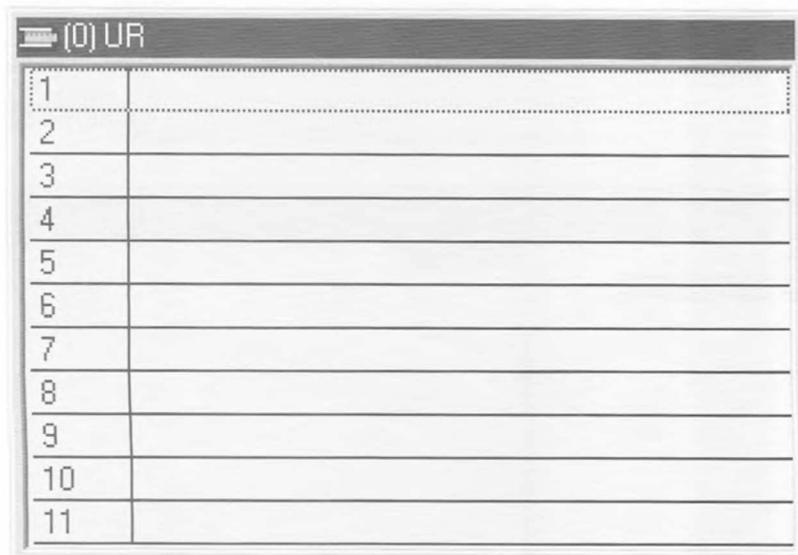


Fig. 14

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

Nos situamos en la posición 1 y vamos a insertar la fuente de alimentación que es lo primero que tenemos en nuestro equipo. Es posible que en una instalación con más equipos, no tengamos la fuente de alimentación en el mismo bastidor que la CPU. La CPU se tiene que alimentar con una fuente de 24v., pero no es necesario que esté situada en el mismo bastidor que ella. En caso de no tener una fuente en el bastidor, dejaríamos esta posición 1 en blanco.

Si tenemos que introducir una fuente de alimentación, desplegamos la cortina de las PS y elegimos la que tengamos en cada caso. En el ejemplo se toma una fuente de 24v. de 5 amperios.



Fig. 15

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

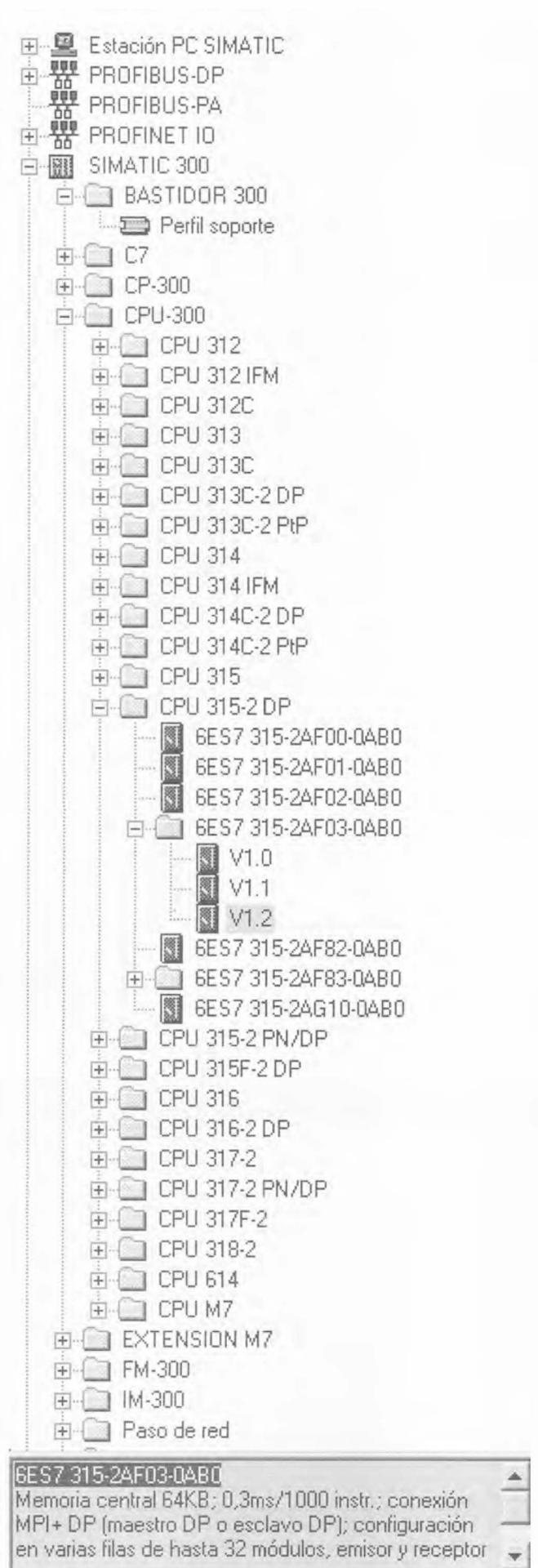


Fig. 16

Tenemos que comprobar que esta referencia coincida con la referencia del elemento que tenemos nosotros. En los equipos de **SIEMENS** suele venir la referencia del elemento en la parte inferior delantera del mismo. También podemos ver que dentro de la misma referencia existen varias versiones. Deberíamos seleccionar el equipo con la misma versión que el equipo del que disponemos. Hay que tener en cuenta que no en todas las referencias existen versiones. Si no se especifica nada, es porque sólo existe una.

En el ejemplo hemos seleccionado una CPU 315-2DP. Si seleccionamos una CPU que tiene algún puerto de red adicional además del MPI, el sistema nos preguntará acerca de los datos de esta nueva red. Veamos el ejemplo al seleccionar una CPU 2DP (con un segundo puerto **PROFIBUS**).

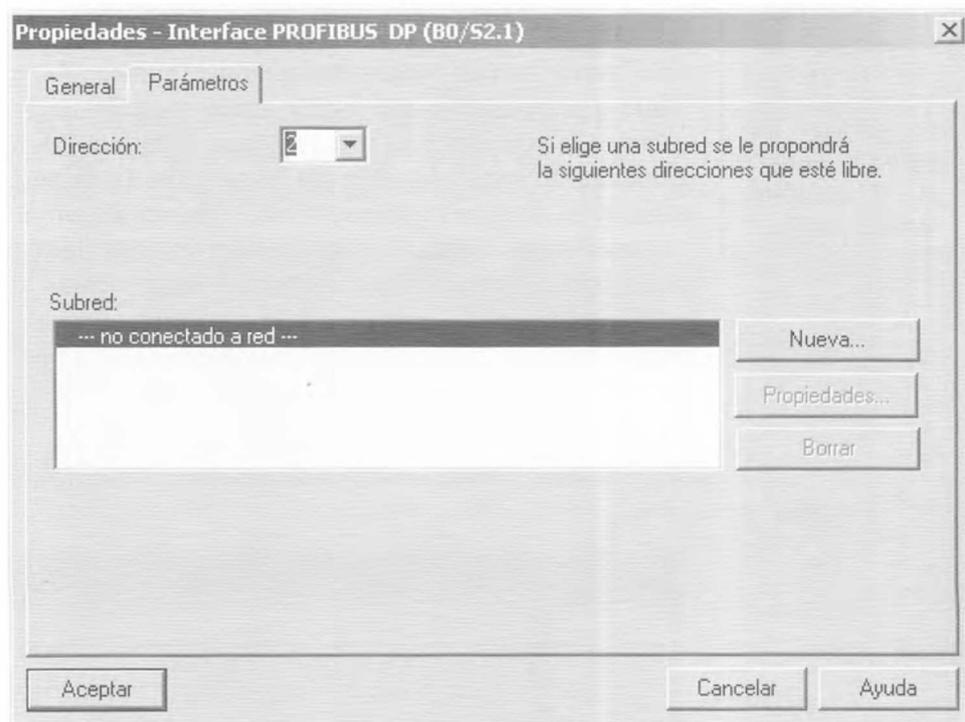


Fig. 17

En esta ventana se nos pregunta por la dirección **PROFIBUS** del puerto de la CPU y nos da la opción de dejarla sin conexión a ninguna red o generar una nueva red para conectar la CPU. Dejaremos la dirección 2 por defecto. Al existir el puerto **PROFIBUS**, necesariamente tenemos que asignarle una dirección. Si recordamos anteriormente también asignamos la dirección 2 al puerto MPI. Al ser redes diferentes no hay ningún problema en que coincida la dirección. En cuanto a la red, de momento para el primer proyecto la dejaremos sin conectar a nada. En ejercicios posteriores volveremos a esta pantalla y crearemos redes y conexiones. Pulsamos el botón de aceptar y tendremos la CPU colocada en la posición 2 del bastidor.

Si la CPU de la que disponemos no tiene ningún puerto adicional además del MPI, no veremos esta pantalla. Simplemente quedará colocada la CPU en la posición 2 del bastidor.

En la posición 3 no podemos insertar cualquier módulo. Es una posición "reservada" para los módulos IM. Estos módulos sirven para realizar configuraciones en más de una línea de bastidor. Si nos fijamos en la ventana que tenemos abierta de *hardware*, veremos que sólo existen 11 posiciones. Esto es porque en un bastidor podemos tener una fuente de alimentación, una CPU y 8 tarjetas de

Recuerda . . .

La posición número 3 de cada bastidor siempre debe quedar libre para poder realizar posteriores modificaciones de ampliación cuando sea necesario. En esta posición se introducirá la tarjeta IM para comunicar los diferentes bastidores.

periferia como máximo. La fuente siempre se colocaría en la posición 1, la CPU en la posición 2 y las tarjetas de periferia en las posiciones de la 4 a la 11. En nuestro caso vamos a tener una única línea de bastidor. Como hemos explicado anteriormente sólo vamos a trabajar con 3 tarjetas de periferia. No tenemos tarjeta IM. En este caso tenemos que dejar la posición 3 libre.

En caso de tener más de 8 tarjetas de periferia, deberíamos insertar un segundo bastidor de la misma forma que hemos insertado el primero. En el segundo bastidor no tendríamos CPU puesto que serían tarjetas de periferia correspondientes a la misma CPU. Deberíamos colocar dos tarjetas IM en la posición 3 de ambos bastidores y a través de ellas es por donde estarían conectadas las tarjetas de periferia del segundo bastidor con la CPU situada en el primer bastidor. Físicamente deberíamos hacer lo mismo. Habría que montar dos bastidores y comunicarlos mediante cable eléctrico a través de las tarjetas IM.

En nuestro caso en el ejemplo, como no tenemos IM, pasamos a la posición 4. Si algún día tuviésemos que hacer una ampliación de la instalación, siempre tendríamos la posición 3 libre para añadir la tarjeta IM. No cambiaríamos el direccionamiento inicial del resto de las tarjetas. En la posición 4 y en las siguientes posiciones, tenemos que insertar los módulos de entradas/salidas que tengamos. Las tarjetas de señales las encontraremos en el desplegable llamado **SM 300**. Si encontramos en el catálogo varias del mismo modelo, tendremos que comprobar para cada caso que la referencia del elemento coincide con la tarjeta que tenemos físicamente.

El *hardware* que utilizaremos para los ejemplos, constará de una tarjeta de 16 entradas digitales, una tarjeta de 16 salidas digitales y una tarjeta de 4 entradas analógicas y 2 salidas analógicas de 8 *bits* de resolución.

Slot	Module
1	PS 307 5A
2	CPU 315-2 DP
X2	DP
3	
4	DI16xDC24V
5	DO16xDC24V/0.5A
6	AI4/AO2x8/8Bit
7	
8	
9	
10	

Fig. 18

Con estas tarjetas podrán realizarse todos los ejercicios que se explican en este manual. En caso de tener una configuración diferente, habría que realizar el *hardware* con arreglo a lo que se tiene en la realidad. Una vez insertadas todas las tarjetas que tenemos, el *hardware* quedaría como se ve en la figura 18.

Esto corresponde a una configuración real como la que vemos en la fotografía siguiente:

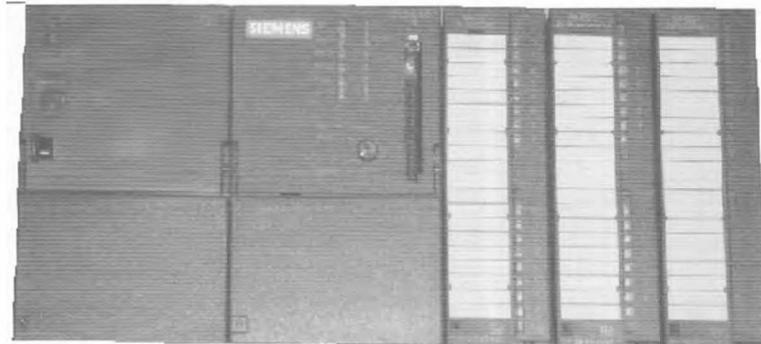


Fig. 19

Veremos que en la parte inferior de la pantalla, se va creando otra tabla en la que podemos ver los elementos que vamos creando con sus referencias y además con las direcciones que se le van asignando a cada uno de los módulos.

Slot	Módulo	Referencia	Firmware	Dirección MPI	Dirección E	Dirección S	Comentario
1	PS 307 5A	6ES7 307-1EA00-0AA0					
2	CPU 315-2 DP	6ES7 315-2AF03-0AB0	V1.2	2			
X2	DP				1023"		
3							
4	DI16xDC24V	6ES7 321-1BH00-0AA0			0...1		
5	DO16xDC24V/0.5A	6ES7 322-1BH00-0AA0				4...5	
6	AI4/AO2x8/8Bit	6ES7 334-0CE00-0AA0			288...295	288...291	
7							
8							
9							

Fig. 20

Una vez hemos terminada la configuración, tenemos que guardarla.

Tenemos que tener en cuenta que estamos trabajando con dos CPU a la vez. Estamos trabajando con el ordenador y con el PLC que tiene su propia CPU y su propia memoria. Debemos guardar la información en ambos sitios si queremos que el PLC tenga su configuración y además poderla tener almacenada en nuestro PC.

Con el icono que representa un disquete, guardamos la información en el ordenador y con el icono que representa un PLC y una flecha que entra, guardaremos la información en el PLC.

Para guardar la información en el PC simplemente pulsamos el botón de guardar.  También existe un icono que representa un disquete con unos ceros y unos debajo.  Este botón nos sirve para guardar y compilar el *hardware* generado.

Si en lugar de guardar pulsamos el botón de guardar y compilar, nos guardará el *hardware* que acabamos de hacer y además el programa analizará si todo lo que hemos puesto en la configuración es coherente. Nos avisará en caso de intentar guardar una configuración imposible o con errores diagnosticables.

Para guardar la información en el PLC tenemos que estar comunicados con su CPU. Vamos a ver cómo hacemos esta primera comunicación con el PLC. Posteriormente se podrá establecer la comunicación por diferentes protocolos y a

Recuerda . . .

Todas las CPU Siemens de las gamas 300 y 400 llevan puerto integrado MPI. Por defecto vienen de fábrica con dirección 2. La comunicación MPI siempre la tendremos disponible para programar, visualizar, diagnosticar o realizar modificaciones.

través de diferentes redes. La primera vez que se establece la comunicación tiene que ser a través del puerto MPI. En esta primera transferencia de información, podemos decir a la CPU si tiene diferentes redes conectadas y las direcciones de cada uno de los puertos de red. A partir de este momento sería posible la conexión a través de otras redes y otros protocolos.

Para la primera conexión necesitamos tener una conexión MPI entre el ordenador y el PLC. Para esto tenemos varias posibilidades. Si el ordenador con el que trabajamos dispone de puerto MPI, tan solo necesitaremos un cable RS-485 (o cable MPI) que conecte ambos puertos. Si por el contrario no disponemos de este puerto en el PC, necesitaremos un cable con conversor a MPI. Existen cables para conectar al puerto serie o al puerto USB. Dependiendo de lo que tengamos, tendremos que hacer una configuración diferente.

Para hacer esta configuración en el **STEP 7** vamos al menú: **Herramientas** → **“Ajustar interface PG/PC”** (de momento dejamos la ventana del *hardware* abierta hasta que enviemos la información al PLC). Tenemos que acceder al menú Herramientas desde la ventana del Administrador de **SIMATIC**. Podemos observar que existe este menú tanto en el Administrador como en el *hardware*, pero el contenido de ambos es diferente.

Veremos una ventana como la que se muestra:



Fig. 21

A esta misma ventana también podemos llegar desde el panel de control. Una vez tengamos instalado el *software* **STEP 7** en el panel de control del ordenador aparecerá un nuevo icono llamado **“Ajustar interface PG/PC”**. Si entramos desde allí llegamos a la misma ventana de configuración.

En esta ventana nos aparece un listado de los *drivers* que tenemos instalados en la máquina para establecer la comunicación con el PLC. En el ejemplo, seleccionamos CP5611 (MPI). La CP 5611 es el puerto MPI / **PROFIBUS** del ordenador que se está utilizando para la resolución de los ejemplos. Entre paréntesis le decimos qué protocolo vamos a utilizar, puesto que por este puerto podemos establecer comunicaciones en varios protocolos (“Idiomas de comunicación”).

Una vez instalado, volvemos a la ventana anterior y ya podremos seleccionarlo.

Una vez seleccionado el puerto y el protocolo y conectado el cable físicamente, deberíamos tener comunicación con el PLC. Para comprobar si existe dicha comunicación podemos hacerlo con el icono de “Estaciones accesibles” desde el Administrador de **SIMATIC**.



Si pulsamos este botón, nos saldrá una ventanita con todas las estaciones que son accesibles por el puerto y el protocolo seleccionado. En este caso nos saldrá una ventana con las estaciones accesibles en MPI. Si hemos hecho todo correctamente deberemos ver una ventana similar a la que se muestra a continuación en la que vemos el PLC que tenemos conectado.

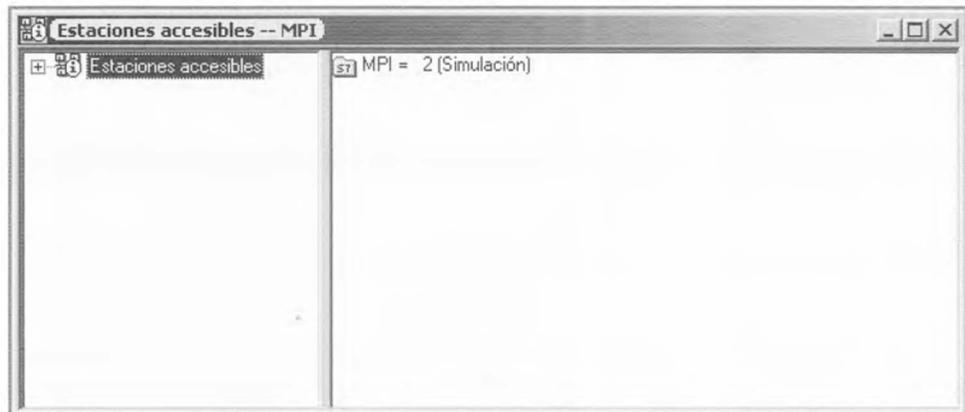


Fig. 23

Recuerda . . .

Al programar un PLC siempre estamos trabajando con 2 CPU. La del ordenador con el que estamos haciendo el programa y la propia CPU del PLC. Siempre tendremos la opción de guardar cosas en ambas CPU. Debemos tener claro qué cosas queremos guardar en cada CPU.

Con esto nos aseguramos de que tenemos comunicación con el PLC. Podemos cerrar esta ventana y continuar con lo que estábamos haciendo. Habíamos creado un *hardware* y lo teníamos guardado en el PC. Ahora teníamos que guardarlo en el PLC. Para ello pulsamos, desde la ventana de *hardware*, el botón que simula un PLC y una flecha que entra.



Al pulsar este botón sale un menú diálogo que nos pregunta a quien queremos enviar esta información. Con el *hardware* que hemos creado nosotros, sólo tenemos la posibilidad de enviar datos a la CPU. Los demás módulos no son programables. El diálogo que vemos es el siguiente:

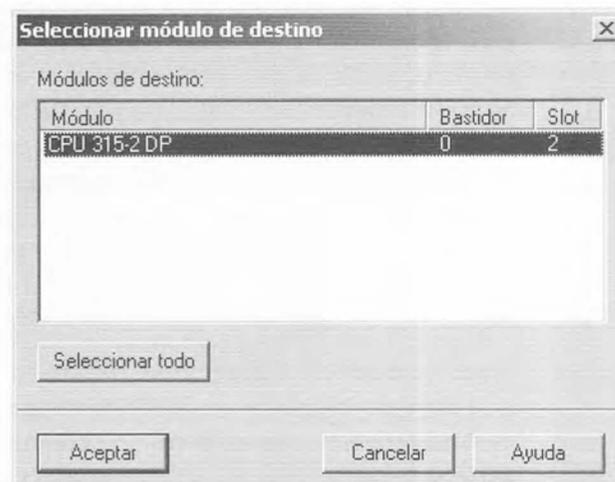


Fig. 24

Aceptamos y con esto ya hemos pasado la información a la CPU. Podemos cerrar la ventana del *hardware* y volver al Administrador de **SIMATIC**.

Una vez cerrada la ventana de *hardware* veremos que volvemos a la misma ventana en la que estábamos antes, es decir, volvemos al Administrador de **SIMATIC**. En nuestro proyecto, tenemos el equipo. Vemos que al lado del equipo hay un signo +. Si desplegamos todo lo que tenemos, vemos que dentro del equipo está la CPU, la carpeta para el programa, los bloques y las fuentes. Si pinchamos encima de los bloques, vemos que en la parte derecha tenemos el **OB 1**.



Fig. 25

Esta ventana es muy parecida a la que obtuvimos generando el proyecto con el asistente del **STEP 7**. Las diferencias las observamos dentro del *hardware*. Si comparamos las dos ventanas del *hardware*, veremos que en el proyecto que hemos hecho a mano, tenemos tanto la CPU como las tarjetas que estamos utilizando perfectamente definidas. Podemos hacer clic sobre cada una de ellas con el botón derecho del ratón y podremos acceder a sus propiedades. En posteriores ejemplos en este manual veremos para que nos puede ser útil modificar dichas propiedades. Si vemos la pantalla del *hardware* generada por el asistente, veremos que sólo existe CPU. No tenemos las tarjetas que vamos a utilizar. Además la CPU que existe en el *software* es una CPU genérica dentro del modelo elegido. Posiblemente no sea la misma que tenemos instalada eléctricamente. De momento vamos a trabajar con el proyecto creado a mano.

El OB1 es el primer bloque que vamos a programar. Aparece por defecto, aunque está vacío. Ahora deberemos entrar en él y generar el programa. Una vez creado nuestro primer programa, deberemos guardarlo en el PC y en el PLC. No siempre es obligatorio guardar las cosas en ambos sitios. Guardarlo en el PC nos sirve para que cuando apaguemos nuestro ordenador, lo que hemos programado nos quede en la memoria y lo podamos abrir o consultar en cualquier momento. Guardarlo en el PLC nos sirve para poder probar lo que hemos hecho en el autómeta. Si por ejemplo estamos haciendo unas pruebas que no sabemos si van a funcionar o no, tenemos la posibilidad de guardarlo sólo en el PLC y probarlo. Podemos hacer tantas pruebas como sea necesario y cuando estemos seguros de que funciona lo que hemos programado, podemos entonces guardar el programa en el PC. También puede ocurrir al revés. Por ejemplo que estemos programando y no tengamos la CPU en este momento. Podemos guardar los programas en el PC y ya los enviaremos al PLC cuando tengamos la posibilidad de conectarnos.

Antes de empezar a programar, hagamos una puntualización. Estando en la pantalla del Administrador de **SIMATIC** vemos que tenemos abierta una ventana con nuestro proyecto que acabamos de crear ("Primer_proyecto"). En los iconos de la parte superior vemos que tenemos seleccionado el icono de **OFFLINE**.



A su lado vemos que tenemos otro icono que representa un PC y un PLC pero unidos con una línea azul. Este es el icono de **ONLINE** y en este momento no lo tenemos seleccionado.

En este momento tenemos apretado el icono de **OFFLINE**. Si apretamos también el icono de **ONLINE** vemos que en el Administrador de **SIMATIC** tenemos dos ventanas parecidas. Con May. F2, nos organizamos las ventanas en forma de mosaico horizontal. Lo que visualizaremos será lo siguiente:

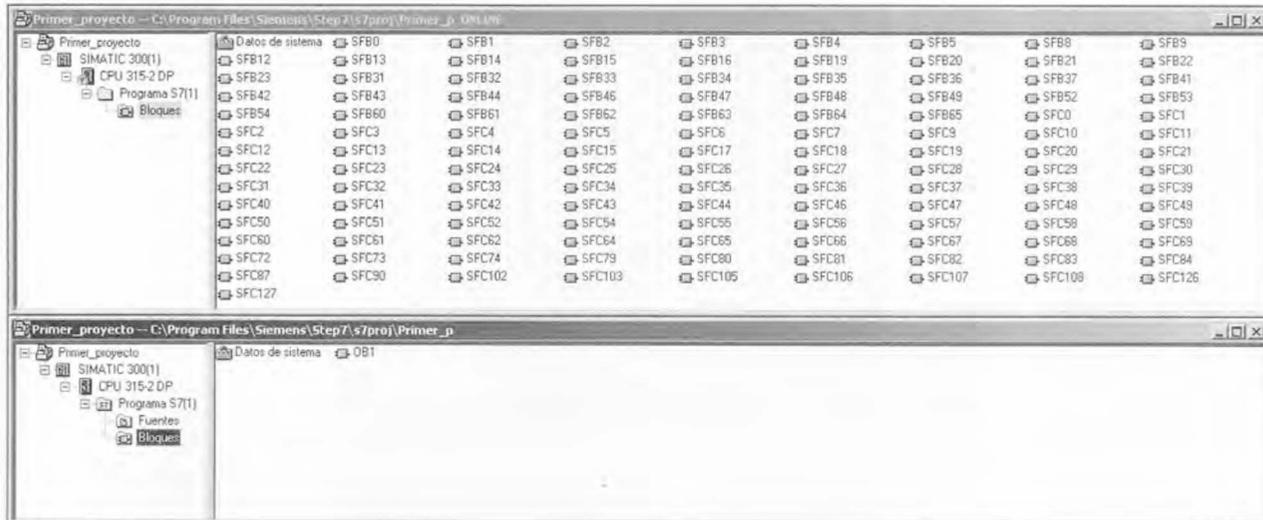


Fig. 26

Siempre que estemos en **OFFLINE**, estamos trabajando en el ordenador. Estamos leyendo del disco duro del PC. Vemos que en los bloques sólo tenemos el OB 1 que es el que ha creado el proyecto por defecto.

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

También tenemos la opción de coger el icono que tiene forma de interrogante en la parte superior de la barra de herramientas y, con él seleccionado, hacer clic sobre el bloque del cual queremos obtener la información.

En posteriores ejemplos en este mismo manual, se verá cómo podemos utilizar estos bloques, tanto las ayudas que nos proporciona el sistema como la propia programación y utilización de los mismos.

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

Para ello pinchamos encima de la CPU de **ONLINE**. En esta posición vamos al menú “Sistema Destino”, “Diagnóstico / Configuración” y elegimos la opción “Borrado Total”.

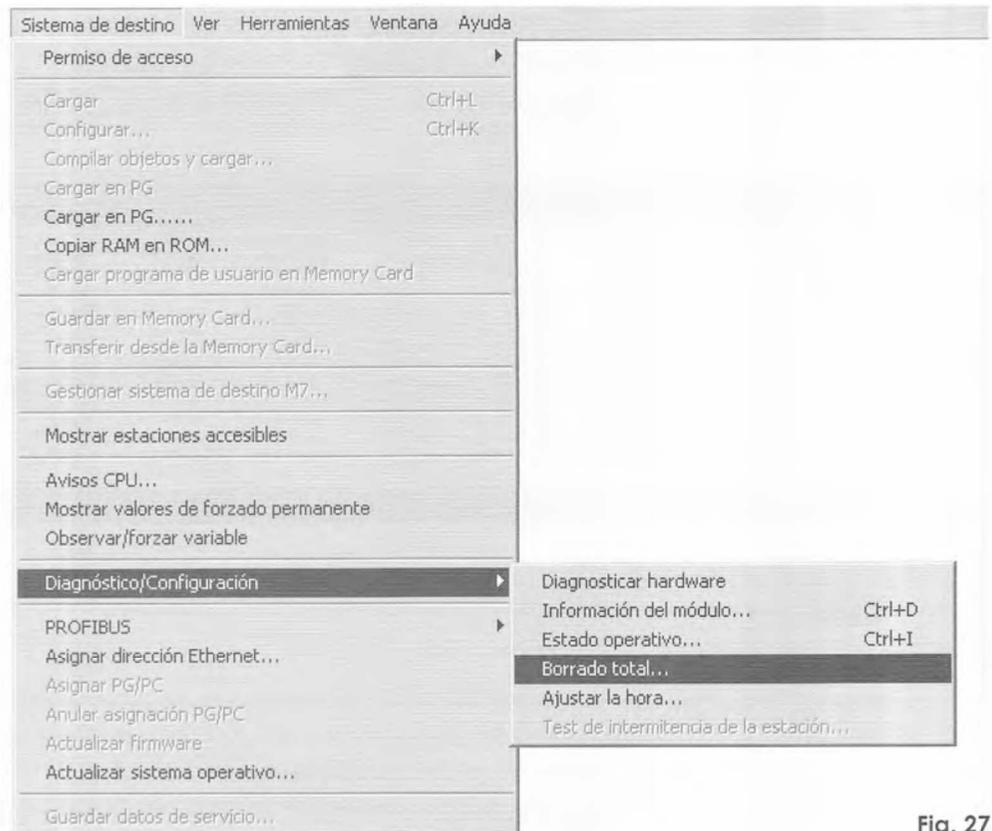


Fig. 27

Con esto borramos todos los bloques que tuviera la CPU excepto los de sistema (que ya hemos dicho que de ninguna manera podemos borrarlos).

Si volvemos a pinchar en bloques veremos que sólo tenemos los de sistema (los que empiezan por S).

También tenemos la opción de hacer un borrado total de la CPU desde el propio PLC. Para ello tenemos que realizar una secuencia especial con la llave azul que lleva insertada la CPU. Se hace mediante esta secuencia especial para que nadie haga un borrado de la CPU sin querer. Si nos fijamos en las posiciones de dicha llave, vemos que tiene 4. La primera **RUN-P**. En esta posición tenemos la CPU funcionando, es decir, ejecutando el programa que tenga cargado y además tenemos la posibilidad de conectarnos con el PC para hacer modificaciones. La segunda opción es **RUN**. En esta posición tenemos al PLC funcionando, es decir, ejecutando el programa que tenga cargado, pero no podemos hacer modificaciones **ONLINE**. Sólo podríamos hacer operaciones de visualización. La tercera opción es **STOP**. En esta posición tendríamos el PLC parado. Es decir, su CPU no estaría leyendo ningún programa. En esta posición si que podemos conectarnos con el ordenador y hacer modificaciones en el programa. La cuarta posición es la que nos servirá para hacer un borrado total de la CPU. Es la posición **MRES**. La llave no se mantiene sola en esta posición. Debemos mantenerla nosotros con la mano. Para proceder al borrado total del PLC deberemos llevar la llave a esta posición y mantenerla mientras la luz de **STOP** se apaga y enciende dos veces. En cuanto haya hecho este parpadeo, sin esperar apenas nada de tiempo, soltamos la llave y la volvemos a llevar a la posición de **MRES**. Ahora la luz de **STOP** deberá parpadear de un modo más rápido. En cuanto deje de parpadear, podemos soltar la llave y ya habremos borrado la CPU del PLC.

Después de haber hecho un borrado total, deberemos volver a enviar el *hardware* para tener la CPU tal y como queríamos. Con el borrado total, se borra tanto el programa como las propiedades de *hardware* que hayamos configurado con antelación.

A la hora de trabajar sobre los distintos bloques, lo podemos hacer tanto en **OFFLINE** como en **ONLINE**. A la hora de guardar lo que hemos programado, lo podemos guardar tanto en la programadora como en la CPU, tanto si estamos trabajando en **ONLINE** como si estamos trabajando en **OFFLINE**. Siempre el icono que muestra un *disquet* será para guardar el programa en el ordenador y el icono que muestra un PLC será para guardar en el PLC.

A la hora de trabajar con los distintos bloques, tenemos que tener en cuenta que en un momento dado podemos llegar a estar trabajando con tres bloques con el mismo nombre a la vez. Podemos tener en un momento dado tres OB1 conteniendo diferente programa a la vez. Por ejemplo, supongamos que estamos trabajando con un OB1. Podemos escribir unas instrucciones y enviarlas al PLC. Luego hacemos una modificación y guardamos en el ordenador. Luego hacemos otra modificación y no la guardamos en ningún sitio. De este modo, tendremos el primer bloque programado dentro del PLC. Con una modificación en el ordenador y con la segunda modificación es lo que estaremos viendo en la pantalla del ordenador pero no está guardado en ningún sitio. Esto no supone ningún problema a priori. Simplemente tendremos que tener claro lo que queremos guardar en cada sitio y saber cómo mirar lo que tenemos en cada CPU.

Tras todas estas explicaciones, vamos a proceder a realizar nuestro primer programa dentro de un OB1.

Una vez tenemos el proyecto creado y el ordenador conectado al PLC, estamos en disposición de empezar a programar. Comenzaremos programando el bloque OB1 que es el que viene creado por defecto. En posteriores ejercicios veremos cómo se van creando y para que sirven otros bloques.

El OB1 es el bloque principal. Todos los programas que hagamos deberán constar al menos de un OB1. Es un bloque necesario aunque luego añadamos más. Cuando la CPU va a leer el programa, siempre va a leer el OB1. Empieza desde la primera instrucción y lee hasta la última. Una vez ha terminado de leer el bloque vuelve a empezar por la primera instrucción. Es un bloque cíclico. Es la base de todos los demás bloques. Cuando queramos que la CPU ejecute otros bloques, deberemos llamarlos desde el OB1 (a excepción de otros OB que más adelante explicaremos cómo se utilizan).

¿Cuántos tipos de bloques podemos programar?

OB	Bloques de organización.
FC	Funciones.
FB	Bloques de función.
DB	Bloques de datos.
UDT	Tipos de datos de usuario.

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

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

FC: Funciones. Son trozos de programa que crea el usuario. Realizan una función determinada dentro del proyecto. Se ejecutan cuando se las llama desde algún punto del programa. Pueden ser parametrizables o no. Además de las FC que yo me creo, existen FC hechas en librerías. Se utilizan exactamente igual que las que yo programo. No podemos entrar en ellas para ver la programación. Las funciones que ya existen dentro de las CPU se llaman SFC.

FB: Bloques de función. En principio funcionan igual que las FC. La diferencia está en que las FB se guardan la tabla de parámetros en un módulo de datos. Esto tiene dos ventajas. Una es que podemos acceder a los parámetros desde cualquier punto del programa. Otra es que cada vez que llamemos a la FB no es necesario que le demos todos los parámetros. Los parámetros que no rellenemos, se tomarán por defecto los últimos que hayamos utilizado o los que existen en el DB correspondiente. También existen FB creadas dentro de cada una de las CPU. En estas FB no podremos entrar y ver el código, pero si podremos llamarlas y utilizarlas. Las FB de sistema se llaman SFB.

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

UDT: Tipo de datos. Nos podemos definir nuestros propios tipos de datos para luego utilizarlos en los DB. Serán adicionales a los tipos de datos ya existentes para el **STEP 7**.

Veremos que en el PLC existen bloque SFC y SFB. Son bloques protegidos a los que no podremos acceder. No podremos ver el código programado pero en cambio sí que los podremos utilizar. Las SFC son lo mismo que las FC pero ya vienen programadas en el sistema. Las SFB son lo mismo que las FB pero ya vienen programadas en el sistema.

Comenzaremos programando el OB1 como habíamos dicho anteriormente.

Abrimos el OB1 de la pantalla de **OFFLINE**. Una vez abierto, el bloque que estamos viendo en la pantalla de la programadora, mientras no lo guardemos en ningún sitio, lo tenemos únicamente en la RAM del ordenador. Si teniendo el bloque en la pantalla pinchamos el icono de “guardar” o el icono de “transferir al autómatas”, estaremos guardando el disco duro o en el autómatas lo que tengamos en la pantalla. Pero si volvemos a la pantalla principal (Administrador de **SIMATIC**) sin haber guardado previamente el bloque en disco duro y transferimos algún bloque arrastrándolo, con ayuda del ratón desde la pantalla de **OFFLINE** hasta la pantalla de **ONLINE**, vamos a transferir lo último que hubiésemos guardado en disco duro y no las últimas modificaciones que hemos hecho en el bloque que veíamos en pantalla.

Recuerda . . .

Tenemos diferentes lenguajes para programar las CPUs de la gama 300 y 400. Por defecto, con el Step 7 disponemos de los lenguajes KOP, FUP y AWL. Como paquetes adicionales existen otros lenguajes, como el GRAPH o el SCL. Elegiremos el lenguaje más apropiado para lo que estemos programando en cada caso.

Vamos a empezar a programar. Para ello abrimos el bloque OB1. Si es la primera vez que lo abrimos, nos saldrá una ventana similar a ésta:

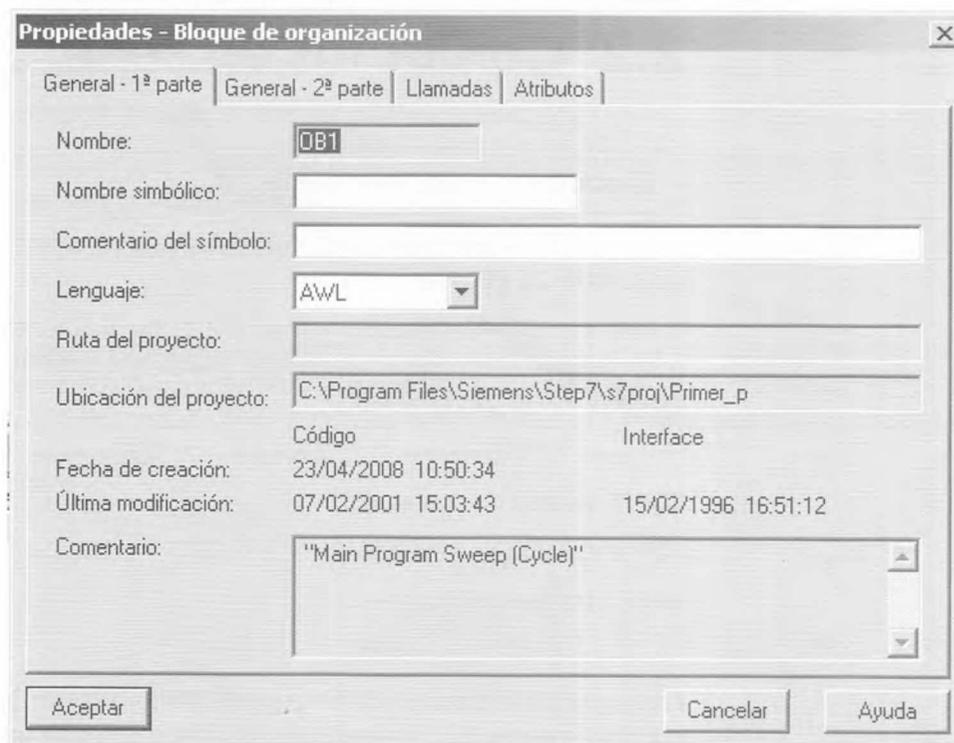


Fig. 28

En esta ventana podemos asignar un nombre simbólico al bloque y podemos decir nuevamente en qué lenguaje queremos programar. De momento por defecto viene seleccionado AWL. Es el que vamos a utilizar para el primer ejemplo. Si pulsamos el botón de aceptar se nos abre el OB1 que tiene este aspecto:

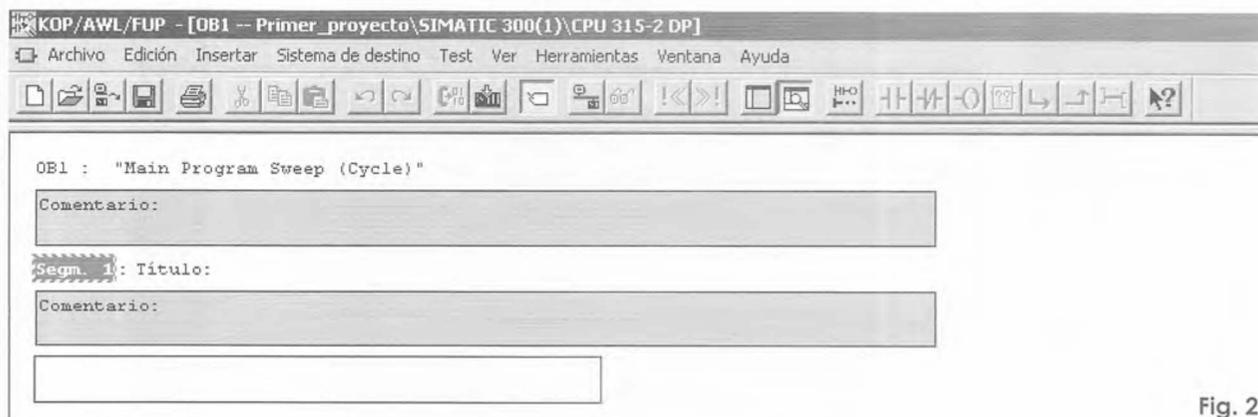


Fig. 29

Hemos abierto otra ventana más adicional al Administrador. Le llamaremos “**editor de bloques**”.

Sabiendo todo esto y teniendo el bloque OB1 abierto, vamos a pasar a la resolución de ejercicios que es la parte principal de este manual. En cada ejercicio se aprenderá algo nuevo sobre instrucciones o bloques programados. Se intentará que cada ejercicio sea corto y sencillo y se centre únicamente en uno o dos conceptos para que se vayan adquiriendo conocimientos poco a poco.

También a lo largo del manual habrá algún ejercicio en el que no se trate nada nuevo sino que se haga un pequeño resumen de lo aprendido anteriormente. Será un ejercicio un poco más extenso y se tratará de utilizar todas las instrucciones y conceptos aprendidos en ejercicios anteriores.

Resolución de ejercicios

2.2 Contactos en serie

Ejercicio 1: Contactos en serie 

ALGO MÁS DE TEORÍA:

Vamos a programar nuestro primer OB 1. De momento habíamos seleccionado como lenguaje de programación AWL. Habíamos visto que teníamos la posibilidad de elegir entre tres lenguajes.

KOP: Esquema de contactos.

FUP: Diagrama de funciones.

AWL: Lista de instrucciones.

En un principio trataremos el lenguaje AWL. Una vez programado el bloque, lo podremos ver en los tres lenguajes posibles y haremos los comentarios sobre cada uno de ellos.

Vamos a empezar por lo más simple que podemos programar en un PLC. Supongamos que lo que queremos programar son dos contactos (dos condiciones) en serie. Y veremos como quedaría hecho en cada uno de los tres lenguajes:

Circuito que queremos programar:



Solución en KOP: Esquema de contactos.



Solución en FUP: Diagrama de funciones.



Solución en AWL: Listado de instrucciones.

```

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

Recuerda . . .

El administrador de Simatic nos ofrece un direccionamiento estándar para poder realizar posteriores ampliaciones de tarjetas. No obstante, siempre se pueden cambiar estas direcciones y asignar las que nos sean más cómodas para nuestro proyecto.

En principio hemos elegido lista de instrucciones. En **STEP 7**, podemos hacer toda la programación que queramos en cada uno de los tres lenguajes. Hay veces que no son exactamente equivalentes los tres lenguajes. Hay instrucciones que existen en unos lenguajes sí y en otros no, pero al final, todo lo que queramos programar es posible programarlo en los tres lenguajes.

Vamos a programar dos contactos en serie.

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

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

Como vemos en el siguiente esquema, se reservan 4 direcciones de *byte* para cada posición de tarjeta. Cada posición ocuparía 4 *bytes* hasta completar las direcciones máximas que permita cada CPU. En el ejemplo, vemos las direcciones correspondientes a las 4 primeras tarjetas digitales.

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

Las direcciones son las mismas independientemente de que las tarjetas sean de entradas o de salidas. También es independiente de que las tarjetas sean de 16 o de 32 *bits*. Si no se ocupa todo el direccionamiento disponible, el resto de direcciones no usadas quedan "reservadas" por si algún día tenemos que cambiar la tarjeta y utilizar otra con más *bits* para ampliar el proyecto. Como vemos, tenemos ocupados 4 *bytes* para cada posición de tarjeta. Si tenemos tarjetas de 2 *bytes*, estamos "perdiendo" estas dos direcciones. Lo de "reservadas" o "perdidas" lo ponemos entre comillas porque más adelante veremos que estas direcciones pueden ser utilizadas como periferia descentralizada.

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

PS	CPU	256	272	288
----	-----	-----	-----	-----

Tendremos *bytes* de entradas 0 y 1, y *bytes* de salidas 4 y 5. Los *bytes* 2 y 3 y los *bytes* 6 y 7 quedan libres en esta configuración. En caso de que en un futuro tengamos la necesidad de ampliar las entradas o las salidas de este PLC, podremos cambiar las tarjetas de 16 *bits* por tarjetas de 32 *bits*, y no deberemos cambiar las direcciones de las entradas y salidas utilizadas en un principio. Las entradas / salidas originales seguirán teniendo direcciones 0, 1 y 4, 5, y las nuevas entradas o salidas de las tarjetas de 32 *bits* serán las 2 y 3 de entradas y 6 y 7 de salidas. La tarjeta de entradas/salidas analógicas, ocupa la posición 288. Tendremos palabras de entrada 288, 290, 292, 294 y palabras de salidas 288 y 290. También aquí quedan direcciones reservadas por si un día necesitamos ampliar.

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

Instrucción de primera consulta es la que se utiliza para analizar la primera condición de una serie de condiciones que terminan realizando una acción.

En el ejemplo que nos ocupa tenemos lo siguiente:

U	E	0.0	(Si está la entrada 0.0. Primera consulta de la serie)
U	E	0.1	(Y está la entrada 0.1. Si además está E0.1 en serie con E0.0)
=	A	4.0	(Activar la salida 4.0 Acción a realizar)

Si queremos programar esto mismo directamente en KOP, lo primero que tenemos que hacer es seleccionar este lenguaje en el editor de bloques. Para ello dejamos el segmento en blanco. Sin tener nada programado, vamos al menú "Ver → KOP". Ahora tenemos un segmento gráfico. Si intentamos programar en AWL veremos que no podemos escribir.

OB1 : "Main Program Sweep (Cycle)"

Comentario:

Segm. 1: Título:

Comentario:



Fig. 30

Para programar en KOP deberemos abrir el catálogo de funciones con el botón que nos muestra un catálogo. 

Se nos abrirá un catálogo como el siguiente en la parte izquierda del editor.



Fig. 31

Tendremos que seleccionar lo que queremos programar dentro de las funciones que nos ofrece el catálogo. Los contactos abiertos los encontramos en **“Operaciones lógicas con bits”**.

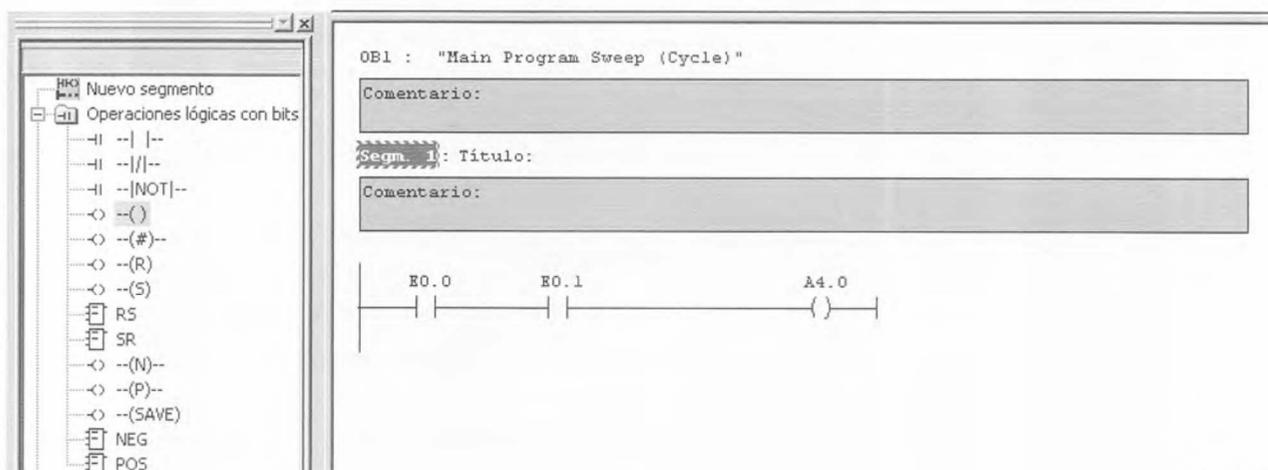


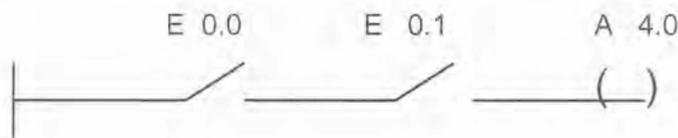
Fig. 32

Ejercicio 1: Contactos en serie ✓

TEORÍA PREVIA: Introducción. Generación de nuevo proyecto.

DEFINICIÓN Y SOLUCIÓN

Supongamos que queremos automatizar el siguiente circuito:



Vemos que lo que tenemos son dos contactos en serie.

Tenemos que asignar nombre a cada uno de los contactos. El **STEP 7** nos permite utilizar dos lenguajes (nemotécnicas) a la hora de programar. Estos lenguajes son independientes de si programamos en AWL, KOP o FUP. Podemos hacer listado de instrucciones en Lenguaje **SIMATIC** (nemotécnica alemana) o Lenguaje **IEC** (nemotécnica inglesa) y también podemos hacer esquemas KOP o FUP tanto en Lenguaje **SIMATIC** como Lenguaje **IEC**. Si utilizamos Lenguaje **SIMATIC**, la mayoría de las instrucciones y nombres de entradas, salidas, marcas, etc. corresponden a iniciales o abreviaturas en alemán. Es el lenguaje que viene por defecto en el *software* y el que se utiliza en el 90% de los programas de mercado. Si utilizamos Lenguaje **IEC**, los nombres de entradas, salidas, marcas, etc. así como las instrucciones, corresponden en su gran mayoría a iniciales en inglés. En este manual se va a utilizar el Lenguaje **SIMATIC** puesto que es el más utilizado en el mercado y entre los programadores. El cambio de un lenguaje a otro es siempre posible sin excepciones. Para aprender a programar deberemos aprender un juego de instrucciones. Si por algún motivo tenemos que tocar o leer un programa hecho por otra persona en el otro lenguaje de programación, siempre será posible hacer la traducción del mismo y tratarlo en el lenguaje que nosotros hemos aprendido. Utilizando el Lenguaje **SIMATIC**, a las entradas les vamos a llamar **E** y a las salidas les vamos a llamar **A**. Esto corresponde a las iniciales en alemán (Lenguaje **SIMATIC**). A la instrucción para unir los contactos en serie le llamaremos **"U"**. En caso de haber elegido Lenguaje **IEC**, a las entradas les llamaríamos **I** y a las salidas les llamaríamos **Q**. A la instrucción para los contactos en serie le llamaríamos **"A"**. De momento vamos a dejar lo que el *software* nos da por defecto que es Lenguaje **SIMATIC**, y una vez tengamos el bloque programado lo traduciremos al otro lenguaje para ver cómo lo podemos hacer.

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

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

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

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

Para unir los dos contactos en serie disponemos de la instrucción **"U"** ya que estamos utilizando Lenguaje **SIMATIC**.

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

Veamos primero la programación en AWL.

SOLUCIÓN EN AWL

```

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

La instrucción BE es opcional. Significa final de bloque de programa. La podremos utilizar en cualquier bloque (OB, FC, FB, etc.). Si no la escribimos no pasa nada. Cuando el autómata lee la última instrucción del OB1 (o cualquier otro bloque) lo dará por terminado. En el caso de ser el OB1, vuelve a empezar la lectura por el principio. El OB1 es el único bloque que se ejecuta de forma cíclica (cuando termina de ejecutarlo, vuelve a empezar por la primera instrucción).

Una vez tengamos el bloque programado y guardado, tendremos la posibilidad de cambiarlo de lenguaje. Veamos primero cómo podemos cambiarlo de Lenguaje **SIMATIC** a Lenguaje **IEC**. Para ello tenemos que cerrar el OB1 después de haberlo guardado. Entonces iremos, desde el Administrador de **SIMATIC**, al menú **"Herramientas -> Preferencias"**.

(**NOTA:** Tener en cuenta que los menús que aparecen en la parte superior de la pantalla no contienen lo mismo si accedemos a ellos desde el Administrador de **SIMATIC** o desde el editor de bloques, o desde cualquier otra aplicación dentro de **STEP 7**. Muchos menús coinciden en el nombre pero no en el contenido. En este manual se hará siempre referencia al menú que queremos entrar según lo que se esté explicando).

Una vez dentro del menú iremos a la ficha llamada **"Idioma"**. Veremos que en la parte izquierda podemos elegir el idioma del *software*. Es decir, el idioma del **STEP 7**. El idioma en el que vemos los nombres de los menús, etc. Dejaremos por defecto español. En la parte derecha podremos cambiar la nemotécnica de alemana a inglesa. Si queremos hacer la prueba de cambiar el idioma, elegiremos nemotécnica inglesa.

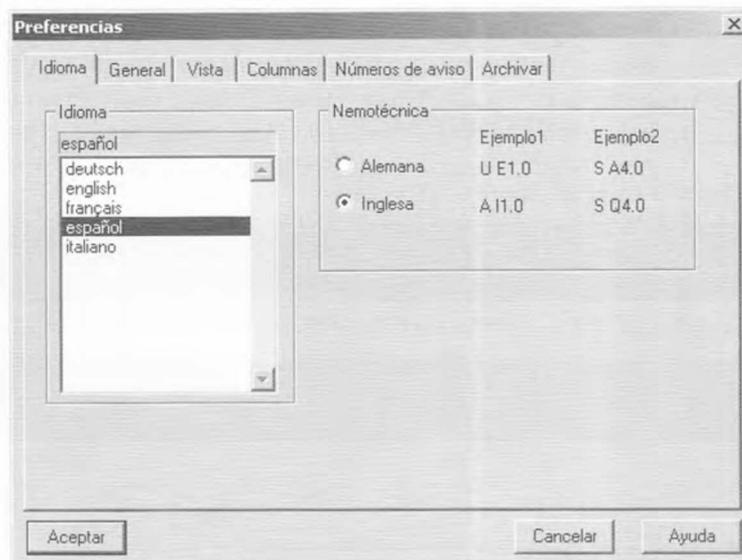


Fig. 33

Recuerda . . .

Siempre podremos convertir un programa de nemotécnica inglesa a alemana y viceversa. Al cambiar la nemotécnica cambia tanto la nomenclatura de los contactos como las propias instrucciones de programa.

Para que el cambio sea efectivo hay que cerrar el **STEP 7** y volverlo a abrir. Veremos que tanto los nombres de las entradas y las salidas como la instrucción utilizada, han cambiado.

Una vez probado volveremos a la nemotécnica alemana.

Ahora estando dentro del bloque, podremos cambiar el lenguaje de programación. Esto lo podremos hacer siempre y cuando el programa sea traducible. Las traducciones de AWL a KOP o FUP no siempre son posibles. En este caso, por ejemplo, la instrucción **BE** no existe ni en KOP ni en FUP. Si intentamos traducir esto nos va a decir que no es traducible. Al principio de este manual dijimos que todo lo podemos programar en cualquiera de los tres lenguajes de programación. Pero también advertimos que no siempre los programas serán traducibles tal cual los tenemos escritos. Debemos respetar unas normas si queremos que se pueda traducir. Lo que si es cierto que cualquier funcionalidad se la podremos dar, de una forma o de otra, en cualquiera de los tres lenguajes.

Si le quitamos el **BE** que no nos aporta nada nuevo al programa, veremos que ya lo podemos traducir. En AWL podemos poner o no poner la instrucción **BE**. El programa funcionará exactamente igual con esta instrucción o sin ella. La instrucción existe porque existía en el antiguo programa **STEP 5** de programación y se ha mantenido. En aquel lenguaje obligatoriamente había que escribir **BE** al terminar cualquier bloque.

Para cambiar el lenguaje del bloque, tenemos que ir al menú **Ver → KOP**

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

SOLUCIÓN AWL

OB1 : "Main Program Sweep (Cycle)"

Comentario:

Segm. 1: Título:

Comentario:

U	E	0.0
U	E	0.1
=	A	4.0

Fig. 34

SOLUCION EN KOP

OB1 : "Main Program Sweep (Cycle)"

Comentario:

Segm. 1: Título:

Comentario:



Fig. 35

SOLUCIÓN EN FUP

OBI : "Main Program Sweep (Cycle)"

Comentario:

Segm. 1: Título:

Comentario:

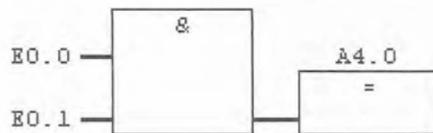


Fig. 36

Ahora deberíamos probar que esto funciona. Enviamos el bloque al PLC y con el simulador que tenemos probamos que funciona. Si activamos las entradas **E0.0** y **E0.1**, debería activarse la salida **A4.0**.

2.3 Contactos en paralelo

Ejercicio 2: Contactos en paralelo

TEORÍA

INSTRUCCIÓN "O"

Para unir dos contactos en paralelo tenemos la instrucción "O".

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

Para la instrucción de primera consulta, podemos utilizar tanto la instrucción "U" como la instrucción "O". Cuando consultamos un primer contacto, de momento no está ni en serie ni en paralelo. Es el segundo contacto el que ponemos en serie o en paralelo con el primero. Lo mismo ocurre al dibujar el segmento en KOP o en FUP. El primer contacto siempre lo dibujamos igual.

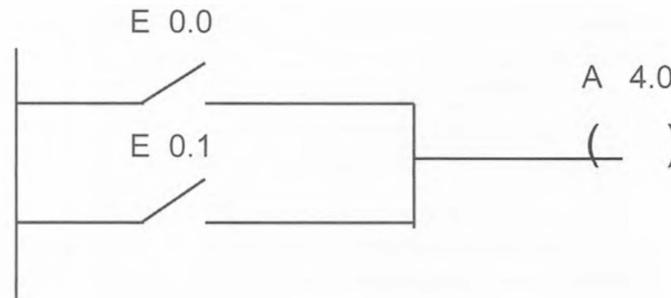
El programa funcionaría exactamente igual utilizando la instrucción "O" o la instrucción "U" como primera consulta. Si queremos añadir más condiciones en paralelo, en AWL las añadimos mediante la instrucción O. En KOP dibujamos más contactos iguales que el primero, pero en paralelo con éste. En FUP deberíamos coger del catálogo la cajita que tiene como símbolo la condición "O" (ver resolución del ejercicio).

Ejercicio 2: Contactos en paralelo ✓

TEORÍA PREVIA: Instrucción "O".

DEFINICIÓN Y SOLUCIÓN

Veamos cómo podríamos resolver el siguiente circuito eléctrico:



Vemos que lo que tenemos son dos contactos en paralelo.

Para programar los contactos en paralelo tenemos la instrucción "O".

SOLUCIÓN EN AWL

```

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

SOLUCIÓN EN KOP

OB1 : "Main Program Sweep (Cycle)"

Comentario:

Segm. 1: Título:

Comentario:

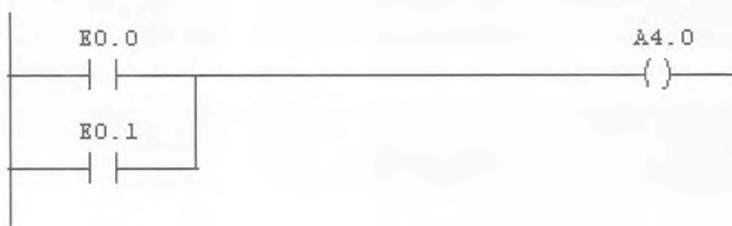


Fig. 37

SOLUCIÓN EN FUP

OB1 : "Main Program Sweep (Cycle)"

Comentario:

Segm. 1: Título:

Comentario:

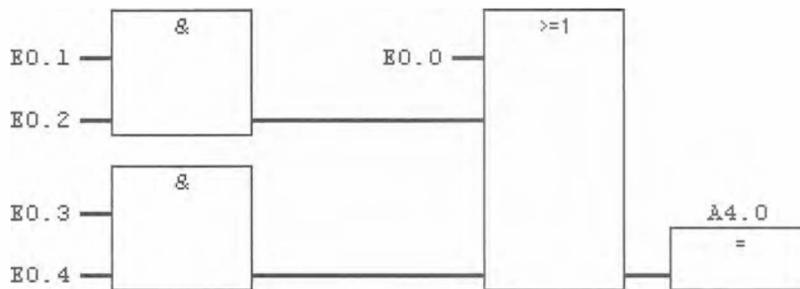


Fig. 40

2.4 Utilización del paréntesis

Ejercicio 3: Utilización del paréntesis

TEORÍA

INSTRUCCIONES O(, U(y)

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

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

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

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

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

El significado sería el mismo.

Ejercicio 3: Utilización del paréntesis

TEORÍA PREVIA: Instrucción paréntesis.

DEFINICIÓN Y SOLUCIÓN

Veamos cómo podríamos programar el siguiente circuito eléctrico:

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

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

SOLUCIÓN EN AWL

```

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

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

En ambos casos el resultado que obtenemos es el mismo.

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

En los lenguajes gráficos no tiene sentido las instrucciones U(, O(. Simplemente dibujamos el segmento de modo gráfico. Si después de haber dibujado gráficamente traducimos a instrucciones, veremos que el propio software incluye los paréntesis.

SOLUCIÓN EN KOP

OB1 : "Main Program Sweep (Cycle)"

Comentario:

Segm. 1): Título:

Comentario:

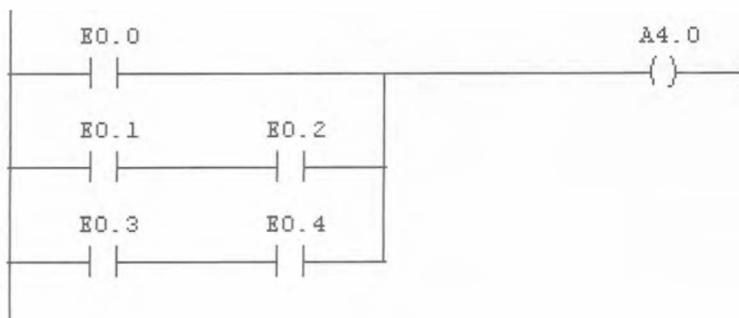


Fig. 39

SOLUCIÓN EN FUP

OB1 : "Main Program Sweep (Cycle)"

Comentario:

Segm. 1: Título:

Comentario:

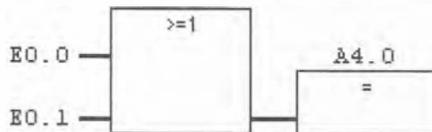


Fig. 40

Recuerda . . .

Al programar un PLC no estamos haciendo lógica cableada. Nosotros podremos activar una salida cuando tengamos un contacto de entrada abierto. Esto no es un circuito eléctrico, son instrucciones de mando para la CPU, que activará las salidas cuando nosotros se lo mandemos a través de las instrucciones de programa.

Ejercicio 4: Contactos negados

TEORÍA

INSTRUCCIONES UN Y ON.

Hemos visto las instrucciones "U" y "O".

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

Son las instrucciones negadas de las anteriores.

Por ejemplo si escribimos:

UN E 0.0

UN E 0.1

.....

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

Esto nos sirve para programar contactos que son normalmente cerrados.

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

Ejercicio 4: Contactos negados ✓

TEORÍA PREVIA: Contacto normalmente cerrado.

DEFINICIÓN Y SOLUCIÓN

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

Para ello utilizaremos las instrucciones "ON" y "UN".

De esta manera estamos negando la instrucción precedente.

Veamos cómo resolveríamos el siguiente circuito eléctrico. Lo que queremos es que se active la salida cuando accionemos los dos pulsadores. En un contacto queremos que dé señal cuando se cierre físicamente el contacto. En el otro caso queremos que dé señal cuando se abra físicamente el contacto.

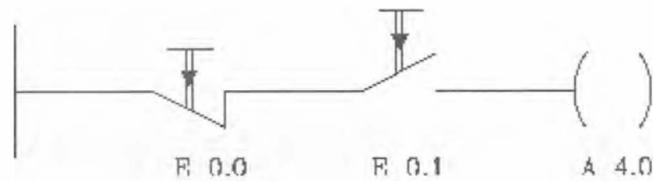


Fig. 41

Hay que tener en cuenta que esto no es un circuito eléctrico propiamente dicho. En un circuito eléctrico, obviamente si el contacto no está cerrado es imposible que pase corriente para encender una bombilla. Pero al tratarse de un programa de PLC, no estamos haciendo conexiones físicas. Estamos dando instrucciones para que llegue tensión a una salida cuando se cumplan una serie de condiciones que nosotros establecemos. Queremos que se encienda la salida A4.0 cuando el contacto E0.0 esté abierto. La tensión no llegará a través de este contacto. Es una instrucción que nosotros damos al PLC.

SOLUCIÓN EN AWL

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

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

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

SOLUCIÓN EN KOP

Segm. 1: Título:

Comentario:

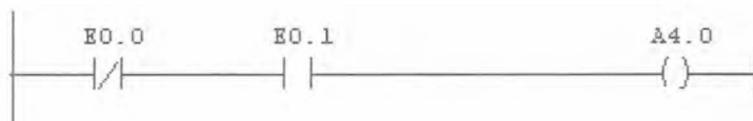


Fig. 42

SOLUCIÓN EN FUP

Segm. 1: Título:

Comentario:

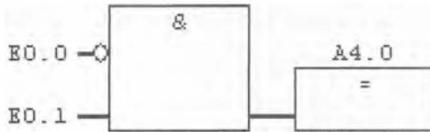


Fig. 43

2.6 Marcas internas

Ejercicio 5: Marcas internas

TEORÍA

DEFINICIÓN DE “MARCA”

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

Estos *bits* podremos activarlos o desactivarlos como si fueran salidas. En cualquier punto del programa los podremos consultar como hemos consultado las entradas hasta ahora.

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

Por ejemplo tenemos las marcas, **M 0.0**, **M 10.7**, **M 4.5**, etcétera.

Ejercicio 5: Marcas

TEORÍA PREVIA: Introducción a las marcas.

DEFINICIÓN Y SOLUCIÓN

Veamos cómo podríamos resolver el siguiente circuito eléctrico:

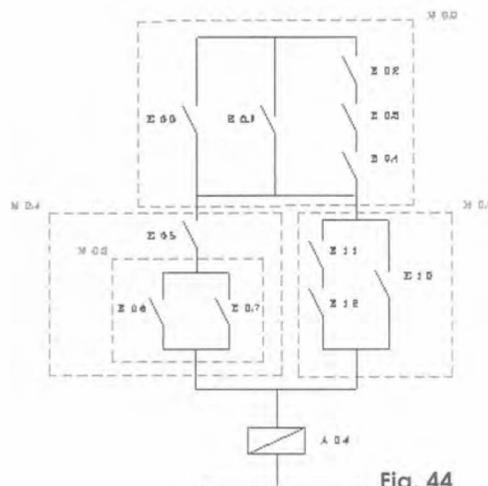


Fig. 44

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

También lo podemos hacer utilizando MARCAS.

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

Veamos cómo podemos simplificar el circuito utilizando las marcas. Cada bloque, señalado en rojo en el circuito anterior, lo convertimos en una marca. El circuito queda mucho más sencillo. Podemos simplificar y agrupar hasta el grado que nosotros queramos.

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

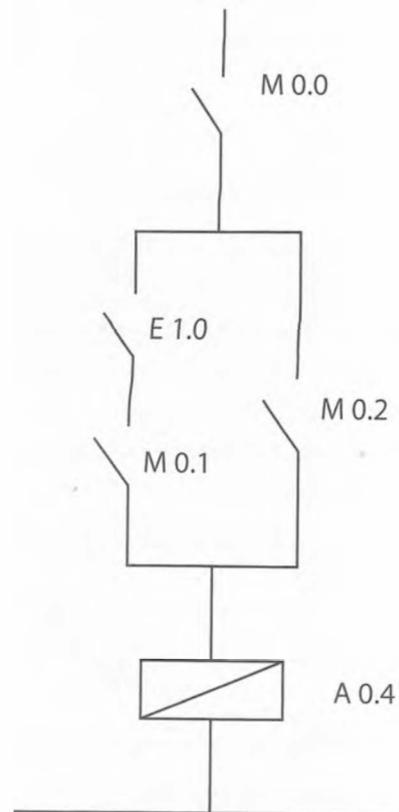


Fig. 45

Veamos cómo quedaría resuelto con MARCAS:

SOLUCIÓN EN AWL

```

U   E   0.0
O   E   0.1
O(
U   E   0.2
U   E   0.3
U   E   0.4
)
=   M   0.0

U   E   0.6
O   E   0.7
=   M   0.1

U   E   1.1
U   E   1.2
O   E   1.0
=   M   0.2

U   E   0.5
U   M   0.1
=   M   0.3

U   M   0.0
U(
U   M   0.3
O   M   0.2
)
=   A   4.0
    
```

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

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

En este caso no tiene sentido mostrar las soluciones en KOP y en FUP porque sería copiar tal cual el circuito inicial tal y como hemos hecho antes.

También podríamos hacer cada marca en KOP o en FUP y después representar el circuito de la figura 45. Esto habría que hacerlo en varios segmentos. El concepto de segmento se ve a continuación aprovechando el siguiente ejercicio.

Recuerda . . .

Ten en cuenta el funcionamiento de la CPU a través de la PAE y de la PAA. Este ejercicio es importante para entender el funcionamiento de los PLC.

2.7 Instrucciones SET y RESET

Ejercicio 6: Instrucciones SET y RESET

TEORÍA

SIGNIFICADO DE SET Y RESET

Las instrucciones **SET** y **RESET** son instrucciones de memoria.

Si programamos un **SET** de una salida o de una marca con unas condiciones, se activará cuando se cumplan dichas condiciones. Aunque las condiciones dejen de cumplirse, no se desactivará hasta que no se haga un **RESET** de la salida o la marca.

Estas instrucciones tienen prioridad. Dependen del orden en que las programemos. Siempre va a tener prioridad la última que tengamos escrita en el programa.

Veamos porqué ocurre esto.

Existen dos registros internos que se llaman **PAE** (imagen de proceso de entradas) y **PAA** (imagen de proceso de salidas).

Antes de ejecutarse el OB1, se hace una copia de las entradas reales en la PAE. Esto quiere decir que el PLC “mira” todas sus entradas y escribe un 1 en el lugar de la PAE correspondiente a las entradas que están activas y un 0 en el lugar de la PAE correspondiente a las estradas que no están activas. El PLC tiene una “foto” del estado de sus entradas. Durante la ejecución del OB1, el PLC no accede a la periferia real para hacer sus consultas, lo que hace en realidad es acceder a este registro interno. No mira si la entrada tiene tensión o no. Lo que mira es si en la foto de esa entrada hay un 0 o un 1. El acceso a periferia es lento y de este modo sólo lo tiene que hacer una vez por ciclo. Este registro se refresca cada vez que comienza un nuevo ciclo de *scan*. Así puede trabajar más rápido.

Según se van ejecutando las instrucciones, el PLC no accede a las salidas reales para activarlas o desactivarlas. Accede al registro interno PAA y pone “0” o “1” en el lugar correspondiente a las salidas reales. También se ahorra tiempo con esto. No se accede a periferia cada vez que hay que modificar una salida. Se crea una foto del estado que deben tener las salidas y, cuando se ha terminado de leer el programa, se envía el nuevo estado a todas las salidas a la vez.

Sólo cuando termina cada ciclo de *scan* accede realmente a las salidas. Entonces lo que hace es copiar lo que hay en la PAA en las salidas reales. Si durante el mismo ciclo de *scan* se activa y desactiva una misma salida varias veces, el registro correspondiente de la PAA cambia de 1 a 0 varias veces pero la salida “no se entera”. Sólo al final del ciclo se le envía el estado a la salida real.

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

Si en un mismo ciclo le digo enciende la salida y, a continuación, le digo que la apague, sólo a final de ciclo es cuando realmente actúa sobre la salida. En este caso, la dejaría apagada.

Ejercicio 6: Instrucciones SET y RESET



TEORÍA PREVIA: Instrucciones **SET** y **RESET**. Diferencia con un "igual".

DEFINICIÓN Y SOLUCIÓN

Vamos a ver cómo podríamos programar un enclavamiento eléctrico.

El circuito eléctrico correspondiente a un enclavamiento sería el siguiente:

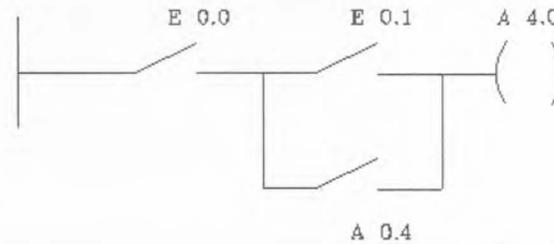


Fig. 46

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

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

Son las instrucciones "**S**" y "**R**" (**SET** y **RESET**)

Veamos cómo quedaría el circuito resuelto:

SOLUCIÓN EN AWL

```

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

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

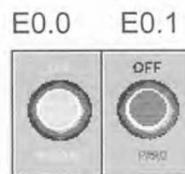


Fig. 47

Si intentamos traducir este programa a KOP o a FUP, veremos que el *software* no puede traducirlo. Esto es porque en KOP o en FUP hay que respetar el sentido del "**segmento**". Un segmento se compone de unas condiciones con las que se realiza una acción. En este ejemplo tenemos dos segmentos. Cuando activemos la E0.0 queremos que se encienda la A4.0. Por otro lado, queremos que al pulsar E0.1 se apague. Son dos acciones diferentes que para poder traducir a KOP tenemos que programar en segmentos diferentes.

En AWL que no es un lenguaje gráfico, podemos utilizar los segmentos más libremente. Podemos desde programar todo el programa en un segmento hasta programar cada instrucción en un segmento distinto. En AWL es indiferente. Como a los segmentos se les puede poner nombre y comentarios, se suelen utilizar para dividir partes de programa separables conceptualmente. Por ejemplo, en un segmento programar un funcionamiento manual, en el siguiente el automático y en el otro las seguridades.

Para ver este ejemplo en KOP y en FUP, primero tendremos que modificar el programa en AWL. Para ello necesitamos añadir un segmento nuevo. Esto lo podemos hacer desde el menú "Insertar" → "Segmento, o con el botón que representa un segmento.



La nueva solución en AWL quedaría de la siguiente manera:

Obl : "Main Program Sweep (Cycle)"

Comentario:

Segm. 1: Título:

Comentario:

U	E	0.0
S	A	4.0

Segm. 2: Título:

Comentario:

U	E	0.1
R	A	4.0

Fig. 48

SOLUCIÓN EN KOP

Una vez tengamos esto programado en AWL, el programa ya es capaz de traducir. Si queremos ver el resultado en KOP obtendremos lo siguiente:

Obl : "Main Program Sweep (Cycle)"

Comentario:

Segm. 1: Título:

Comentario:



Segm. 2: Título:

Comentario:



Fig. 49

SOLUCIÓN EN FUP

OB1 : "Main Program Sweep (Cycle)"

Comentario:

Segm. 1 : Título:

Comentario:



Segm. 2 : Título:

Comentario:

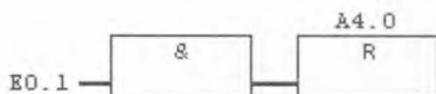


Fig. 50

2.8 Opción TEST > OBSERVAR

Ejercicio 7: Opción "TEST > OBSERVAR" 

TEORÍA

OBSERVAR LA EJECUCIÓN DEL PROGRAMA

Vamos a hacer el siguiente programa:

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

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

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

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

Así podremos “ver” cómo se ejecuta el programa. En AWL la ejecución la vemos a través de 0 y 1 en diferentes columnas.

Veremos que, por defecto, aparecen tres columnas al lado de las instrucciones.

El ejemplo que tenemos quedaría de la siguiente manera:

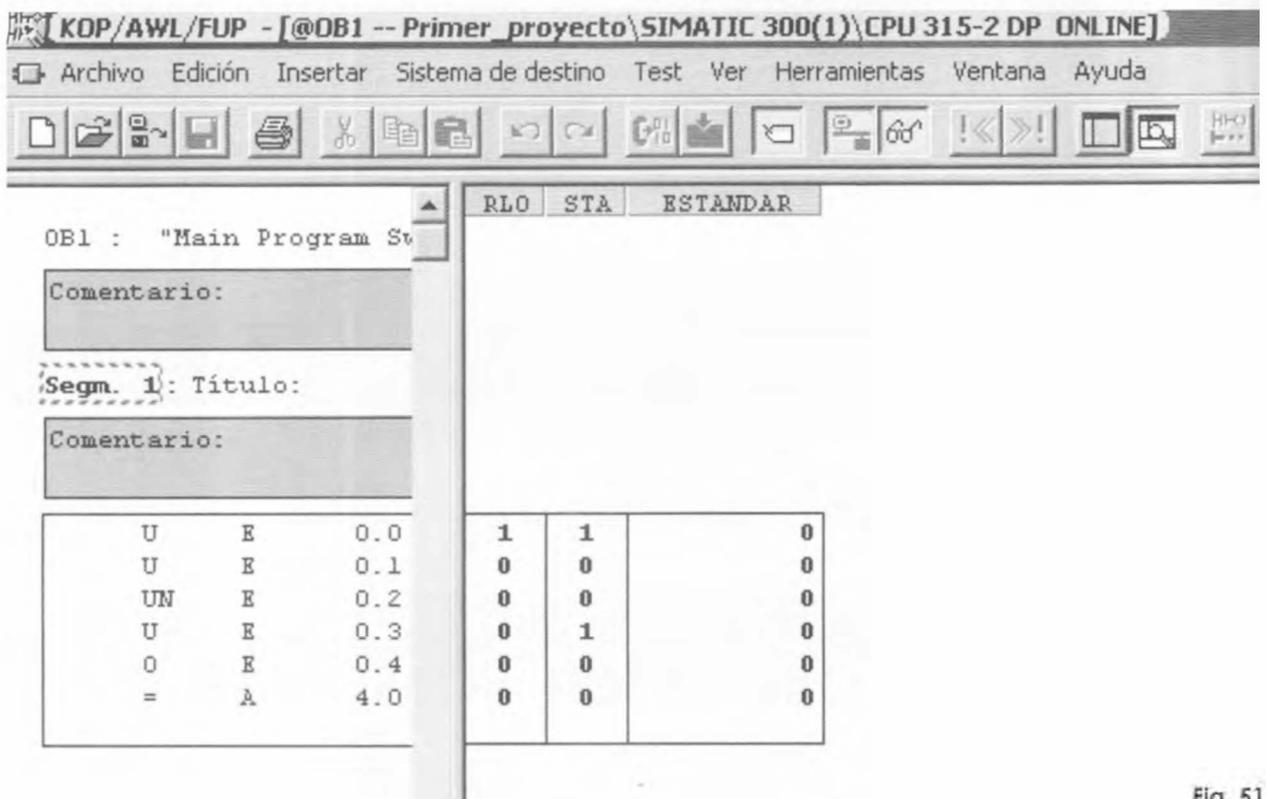


Fig. 51

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

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

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

Se recomienda hacer varias pruebas con el programa ejemplo y analizar los valores obtenidos. En el ejemplo mostrado en la figura 51 se observa que la E0.0 y la E0.3 están activas y las demás no. El primer RLO que vemos a 1 significa lo siguiente. La primera instrucción dice: “si tenemos activo el contacto E0.0...” y efectivamente está activo, con lo cual el RLO es 1. La siguiente instrucción dice: “y si además también tenemos el E0.1...”, como esto no se cumple el RLO es 0. De aquí en adelante, todo lo que vamos diciendo no se cumple, con lo cual los RLO son 0 y no se activa la salida.

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

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

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

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

KOP:

OB1 : "Main Program Sweep (Cycle)"

Comentario:

Segm. 1: Título:

Comentario:

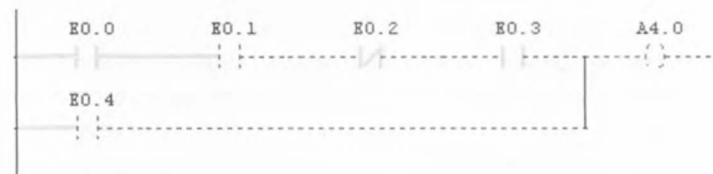


Fig. 52

FUP:

OB1 : "Main Program Sweep (Cycle)"

Comentario:

Segm. 1: Título:

Comentario:

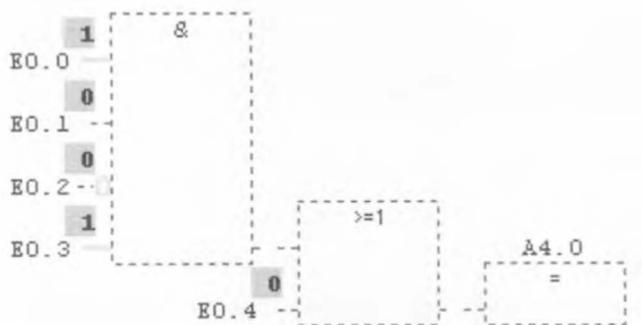


Fig. 53

Se recomienda hacer varias pruebas activando y desactivando contactos, mientras se va observando como se comportan las diferentes visualizaciones.

2.9 Tabla OBSERVAR / FORZAR VARIABLE

Ejercicio 8: Tabla OBSERVAR / FORZAR VARIABLE

TEORÍA

OBSERVAR O FORZAR EL VALOR DE CUALQUIER VARIABLE

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

Estas tablas son un bloque más dentro del proyecto. Hasta ahora teníamos el OB1. Si generamos una tabla, tendremos el OB1 y la tabla que generemos. En versiones anteriores de **STEP 7**, las tablas tenían un nombre fijo igual que el resto de bloques. Se llamaban VAT. Por esto cuando generamos una tabla se nos ofrece como nombre por defecto VAT 1. En la nueva versión de **STEP 7** podemos darle el nombre que nosotros queramos libremente. Podemos tener tantas tablas como queramos. Cuando pinchemos en los bloques del proyecto, veremos las tablas que tenemos junto con los bloques de programa. Las diferenciaremos de los bloques de programa por el nombre y por el icono que aparece delante de ellas.

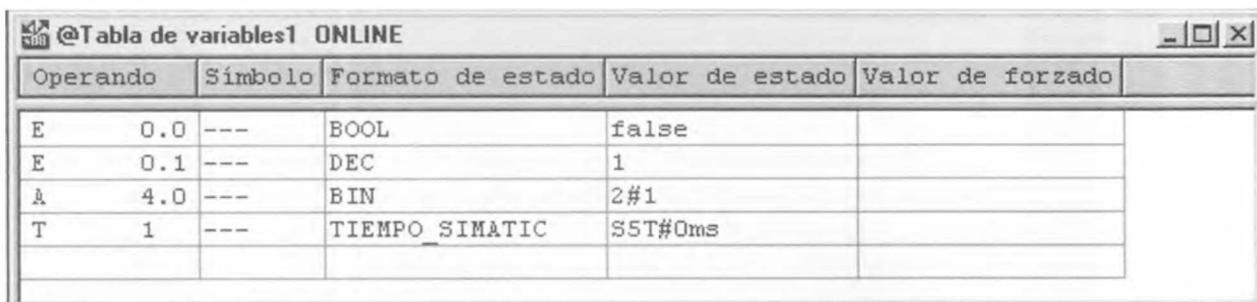
Para poder generar una tabla, tenemos que estar bien dentro de un bloque de **ONLINE / OFFLINE**, o bien desde el Administrador de **SIMATIC** pinchando en la parte izquierda encima del nombre de la CPU en **ONLINE**. En las nuevas versiones lo podemos hacer desde cualquier punto del Administrador de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

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

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

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

TABLA DE VALORES



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

Fig. 54

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



Fig. 55

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

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

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

Una vez forzado un valor, veremos que el valor actual de la variable es el que acabamos de forzar (siempre y cuando el valor no cambie por otras circunstancias. Si nosotros forzamos un valor a una variable pero el programa modifica este valor, la CPU hace caso al programa e instantáneamente perdemos el valor de forzado).

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

2.10 Depósito de agua

Ejercicio 9: Depósito de agua



TEORÍA PREVIA: Contactos, marcas, **SET** y **RESET**.

DEFINICIÓN Y SOLUCIÓN

Los ejercicios que hemos hecho hasta ahora han sido pequeños ejemplos para familiarizarnos con el editor de bloques y empezar con nuestras primeras instrucciones. Los hemos hecho sobre el mismo OB1 machacando unos con otros. A partir de ahora vamos a realizar ejercicios "reales". Vamos a poner en práctica lo que estamos aprendiendo. Si no queremos machacar estos ejercicios y guardarlos para poder consultarlos en un futuro veremos cómo podemos hacerlo.

En este primer ejercicio vamos a hacer un proyecto nuevo. En este caso lo haremos de modo manual y sin introducir *hardware*. Ya dijimos en la introducción que si no necesitamos cambiar las propiedades de las CPU o de las tarjetas utilizadas, no nos hace falta introducir los equipos de que disponemos. Crearemos un proyecto nuevo y le llamaremos "**Mis_primeros_ejemplos**".

Insertaremos directamente una carpeta de programa. Sin insertar previamente ningún equipo. A la carpeta de programa le llamaremos “Depósito de agua”. Nos quedará el proyecto de la siguiente manera:

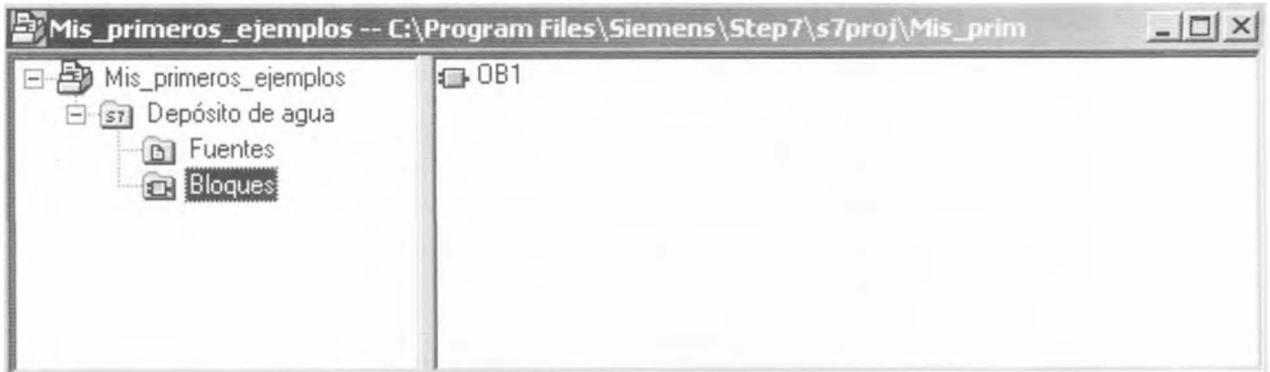


Fig. 56

En este OB1 realizaremos el programa.

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

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

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

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

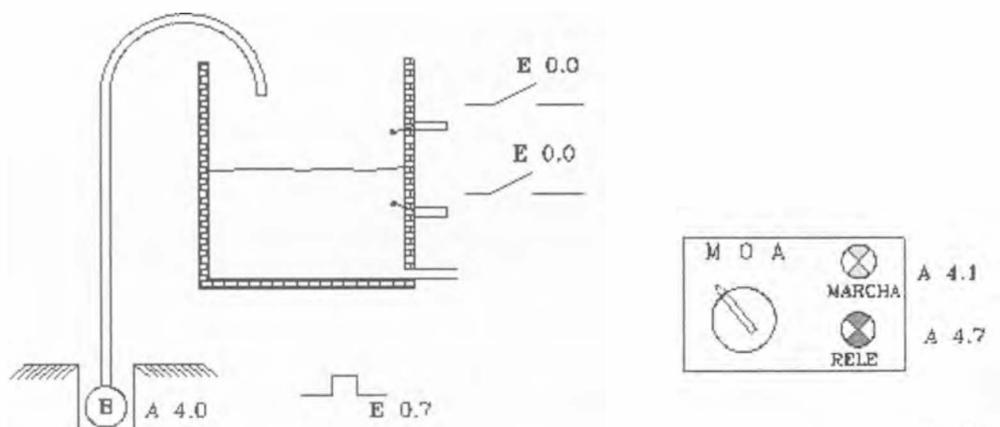


Fig. 57

Veamos cómo podríamos plantear una primera solución a la programación del depósito:

SOLUCIÓN EN AWL

Segmento 1: MANUAL

```

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

Segmento 2: AUTOMÁTICO

```

U    E    0.1          //Si está en automático
U    E    0.7          //Y está bien el relé
U    E    0.2          //Y está activo el nivel de abajo
UN   E    0.3          //Y no está activo el nivel de arriba
S    A    4.0          //Pon en marcha la bomba
S    A    4.1          //Y enciende la luz de marcha
U    E    0.1          //Si está en automático
U    E    0.7          //Y está bien el relé
UN   E    0.2          //Y no está activo el nivel de abajo
U    E    0.3          //Y se ha activado el nivel de arriba
ON   E    0.7          //O ha saltado el relé
R    A    4.0          //Para la bomba
R    A    4.1          //Apaga la luz de marcha
UN   E    0.7          //Si ha saltado el relé
=    A    4.7          //Avísame con la luz de relé
    
```

¿Parece correcta esta programación?

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

Se recomienda hacer varias pruebas tanto en automático como en manual. Actuar la marcha y el paro en ambos modos de funcionamiento y el *relé* térmico.

Se recomienda analizar el programa e intentar saber por qué no funciona correctamente. Si se tiene claro por qué este bloque no funciona, se entenderá bien el funcionamiento del ciclo de *scan* del PLC.

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

Recuerda . . .

NUNCA debes repetir la activación de una salida en un mismo ciclo de programa. A veces puede funcionar, pero no nos será nada útil a la hora de hacer modificaciones, diagnóstico o visualización.

Las salidas no se activan en el mismo instante en el que se lee la instrucción correspondiente. Como se explicó anteriormente, se activa el registro interno denominado PAA (Imagen de proceso de salida), en el que se van almacenando los valores que se tienen que transferir a las salidas cuando finalice el correspondiente ciclo de *scan*. Cuando se lea la instrucción BE (o cuando se termine de leer la última instrucción del bloque), es cuando se mandarán estos valores a las salidas reales. Si hemos enviado varios valores dentro del mismo ciclo de *scan*, el que realmente llegará a las salidas será el último que hemos enviado.

En este caso el segmento de manual siempre envía datos a las salidas. Si está activo envía un 1 y si no está activo envía un 0. En cambio, el modo automático sólo envía valores a las salidas al llegar al nivel mínimo y al llegar al nivel máximo. Cuando se acciona la boya de mínimo mandamos un **SET** al PLC. Queremos que la bomba se ponga en marcha y continúe así aunque la boya deje de estar actuada hasta que llegue el nivel máximo. ¿Qué pasa cuando la boya deja de actuar pero no hemos llegado al nivel máximo? que el segmento de automático no envía nada a la salida (bomba), pero el segmento de manual, como no está en este modo, dice que pare la bomba. Con lo cual queda parada y nunca llenaría en modo automático.

El ejercicio bien resuelto quedaría de la siguiente manera:

Segmento 1: MANUAL

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

Segmento 2: AUTOMÁTICO

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

Segmento 3: INDICADOR DE LED TÉRMICO

```
UN  E   0.7           //Si no está el relé
=   A   4.7           //Activa la luz de relé
```

Ahora nos quedaría asignar las marcas a las salidas.

Añadimos:

Segmento 4: ASIGNACIÓN DE MARCAS A SALIDAS

```

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

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

```

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

Otra posible solución sería programar el paro de la bomba por apertura del *relé* térmico en el último segmento. Los paros de emergencia se suelen programar al final. Como se vio en la explicación de las instrucciones **SET** y **RESET**, sobre las salidas predomina lo último que se programa. Programaremos lo último, lo que queramos que tenga prioridad en caso de emergencia.

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

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

Añado las instrucciones pertinentes.

```

U    E    0.0
UN   E    0.1
    
```

Ahora ya tenemos la base del programa. Podemos añadir todo lo que creamos que sea necesario o conveniente. Por ejemplo, en este caso no he tenido en cuenta la situación de que, después de haber estado en manual o en automático, volvamos a la posición de reposo. En automático he hecho **SET** a ciertas marcas. Cuando volvamos a la posición de reposo esas marcas tendrán que volver a cero. De lo contrario, podría darse el caso de que estando en la posición de reposo, tengamos la bomba en marcha. Para remediar esto podría añadir las siguientes instrucciones:

```

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

2.11 Semáforo

Ejercicio 10: Semáforo

TEORÍA

TEMPORIZADORES “SE” Y “SI”

Para realizar este ejercicio, primero analizaremos lo que son los temporizadores y cómo se programan. Proponemos crear una carpeta de programa nueva que se llame “semáforo”. Aquí haremos las pruebas de los temporizadores y terminaremos machacando las pruebas con el programa definitivo del semáforo. Las pruebas seguramente no nos interesa guardarlas. Para añadir una carpeta de programa nueva, iremos al Administrador de **SIMATIC**. Nos situaremos sobre el nombre del proyecto en la parte izquierda de la ventana. Iremos al menú “insertar” y volveremos a insertar un programa **S7**. Quedará nuestro proyecto de la siguiente manera:

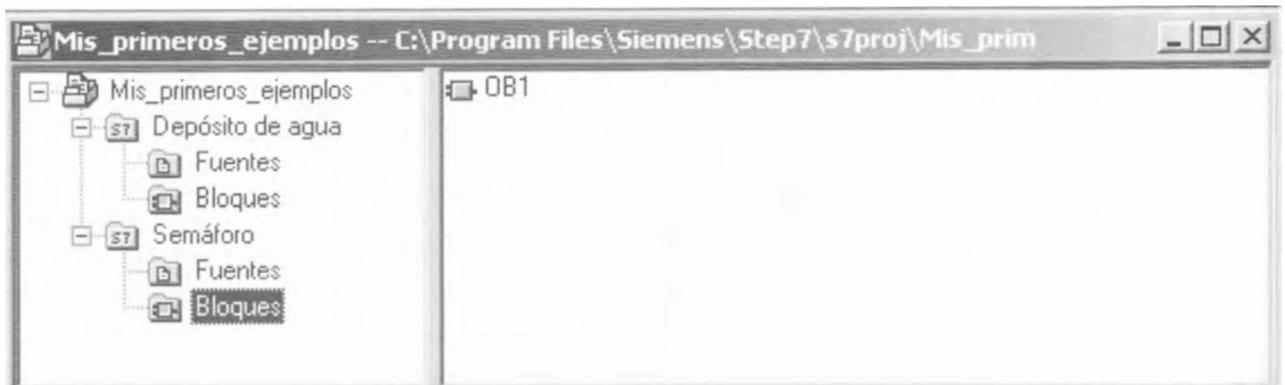


Fig. 58

En este nuevo OB1 haremos las pruebas de los temporizadores y posteriormente el ejemplo del semáforo. En el mismo proyecto tendremos guardado el depósito de agua y el semáforo.

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

Analicemos cada uno de ellos.

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

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

2ª Necesitamos decirle cuanto tiempo tiene que temporizar. Para introducir el tiempo utilizaremos la instrucción “L” que significa “cargar”. Para introducir el tiempo deberemos hacerlo tras el formato de tiempo (S5T#). Esto significa que lo que escribamos a continuación es un tiempo. El tiempo lo podremos escribir en horas, minutos, segundos y milisegundos. Veamos unos ejemplos de tiempos:

- L S5T#2H3M Dos horas y 3 minutos
- L S5T#10S20MS 10 segundos y 20 milisegundos

2ª Necesitamos decirle cuanto tiempo tiene que temporizar. Para introducir el tiempo utilizaremos la instrucción “L” que significa “cargar”. Para introducir el tiempo deberemos hacerlo tras el formato de tiempo (S5T#). Esto significa que lo que escribamos a continuación es un tiempo. El tiempo lo podremos escribir en horas, minutos, segundos y milisegundos. Veamos unos ejemplos de tiempos:

- L S5T#2H3M Dos horas y 3 minutos
- L S5T#10S20MS 10 segundos y 20 milisegundos

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

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

5ª haga algo.

El formato S5T#... significa que utiliza el formato de tiempos del S5. El PLC dispone de unas ondas cuadradas internas de 10 segundos, 1 segundo, 100 ms y 10 milisegundos. Al final lo que hace es contar cuadraditos de estas ondas. El tiempo que nosotros introducimos, el PLC lo almacena en 16 *bits*. Estos *bits* significan lo siguiente:

Los 4 primeros son la base de tiempos. En los siguientes 3 grupos de 4 *bits* almacena la cantidad de tiempo en formato BCD. Esto significa que almacena de 0 a 9 en cada 4 *bits*.

Base de tiempos	0 - 9	0 - 9	0 - 9
-----------------	-------	-------	-------

El PLC dispone de cuatro bases de tiempos. Base 0 cuenta ondas de 10 ms. Base 1 cuenta ondas de 100 milisegundos. Base 2 cuenta ondas de 1 segundo. Y base 3 cuenta ondas de 10 segundos.

Veamos unos ejemplos de lo que hace el PLC cuando introducimos un tiempo para un temporizador. Supongamos que escribimos la siguiente instrucción:

L S5T#3S

El PLC intenta siempre traducir el tiempo introducido en la base más pequeña para tener menor error en la cuenta (máxima resolución). En este caso, el PLC se preguntaría: ¿Cuántos cuadraditos de 10 milisegundos son 3 segundos? La respuesta sería 300. Por lo tanto el PLC almacenaría 300 en base 0.

0	3	0	0
---	---	---	---

Veamos qué ocurre si escribimos la siguiente instrucción:

L S5T#10S

El PLC se preguntaría: ¿Cuántos cuadraditos de 10 ms son 10 segundos? La respuesta sería 1.000. Sólo nos caben 3 dígitos. Con lo cual no puede representar 10 segundos en base cero. Por lo tanto, cambiaría de base y entonces se preguntaría: ¿Cuántos cuadraditos de 100 ms son 10 segundos? La respuesta en este caso sería 100. Con lo cual el PLC almacenaría:

1	1	0	0
---	---	---	---

En función de la cantidad de tiempo que intentemos cargar, el PLC utilizará una base u otra. Cuanto más grande sea la base utilizada, más grande es el error que se puede cometer en la temporización. El PLC cuenta flancos de la onda generada.



Recuerda . . .

Dentro de Step 7 disponemos de herramientas para poder escribir cualquier número en formatos cómodos para el usuario. No es necesario que sepamos cómo trabaja el PLC internamente para poder escribir los datos que necesitemos. Veremos que los formatos se indican escribiendo XXX#..... En el caso del tiempo escribiremos S5T#1M3S, por ejemplo.

Una vez se ha activado el temporizador, cada flanco de onda el PLC incrementa la cuenta. Los cuadraditos son exactos en cuanto al tiempo que duran. El error lo tenemos siempre en el primer cuadradito. Al activar la condición de inicio del temporizador podemos estar en cualquier punto de la primera onda (línea roja en la figura). Si activamos la condición al principio de la onda, al primer flanco de bajada el PLC contará 1 y realmente habrá pasado el tiempo correspondiente a la onda. Pero si activamos la condición justo al final de la línea roja, al primer flanco de bajada el PLC contará 1 y en realidad no habrá pasado el tiempo completo del primer cuadradito. Por eso el error máximo de los temporizadores es de 10ms, 100ms, 1s o 10 s, dependiendo de la base que se esté utilizando. Por eso el PLC siempre utiliza la menor base posible.

Todo esto en principio para el programador es irrelevante. Si yo quiero contar 25 minutos y 14 segundos, escribo L S5T#25M14S. No me preocupo de cómo lo va a escribir el PLC ni de qué base de tiempos utiliza. Para eso sirven las expresiones de formato. Pero en cambio nos viene bien saber cómo trabaja internamente el PLC para entender que no podemos cargar tiempos menores de 10ms ni tiempos mayores de 999 veces 10 segundos. O sea 2 horas 46 min 30 segundos. También nos viene bien saber todo esto para saber con que margen de error trabajan los temporizadores. Y también entenderemos por qué no es posible escribir algunas combinaciones de tiempos. Por ejemplo, no puedo escribir L S5T#2H10MS. Si escribo esto, veré que el propio *software* elimina los milisegundos y escribe sólo 2 horas. Esto es porque al escribir 2 horas, el PLC ha pasado a contar en base 3. No tiene resolución de milisegundos. Lo mínimo que puede contar son saltos de 10 S. El propio sistema elimina lo que no puede contar. Si entendemos cómo funciona internamente sabremos por qué no nos permite ciertas combinaciones de tiempo y podremos ajustar mejor los tiempos a lo que queremos o necesitamos programar.

En cuanto a modos de funcionamiento de los temporizadores, existe 5. Veamos de momento los dos que corresponden a temporizadores sin memoria. Son el SE y el SI.

El modo de funcionamiento SE es el siguiente:



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

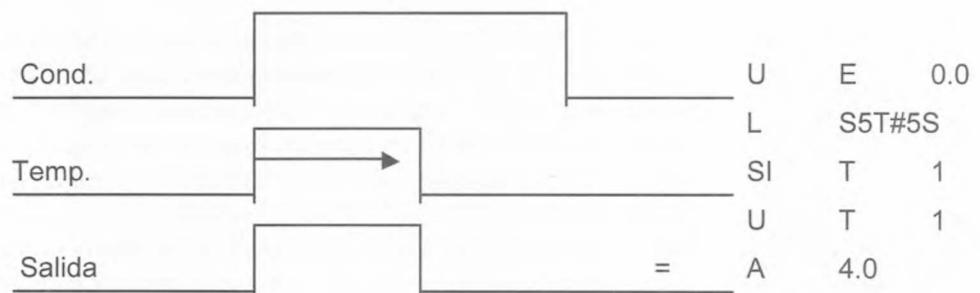
Para ello añadiríamos estas dos instrucciones:

U E 0.1
R T 1

Llamamos temporizadores sin memoria, porque para que terminen de contar su tiempo se debe mantener la condición a 1. En el momento la condición inicial deja de existir, el temporizador vuelve a 0 y “pierde la cuenta”.

El temporizador SI es también sin memoria pero su modo de funcionamiento es diferente.

El modo de funcionamiento SI es el siguiente:



A este temporizador también podemos añadirle un **RESET** en cualquier momento. Para ello escribiríamos estas dos instrucciones:

```

U   E   0,1
R   T   1
    
```

Si se activa la E 0,1 durante el tiempo que al temporizador le corresponde estar a 1, automáticamente bajaría a 0 y perdería la cuenta.

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

KOP

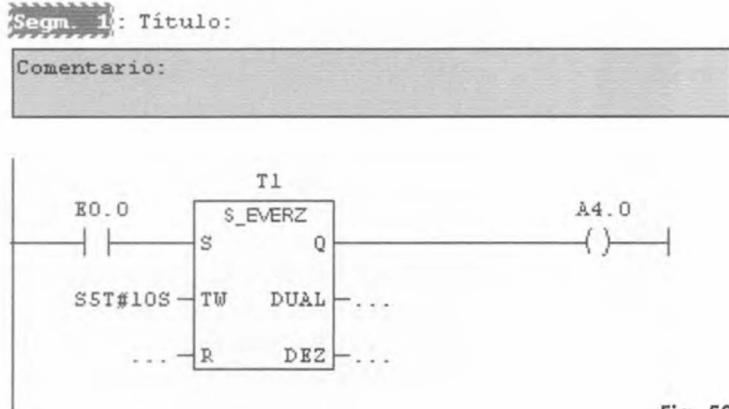


Fig. 59

FUP

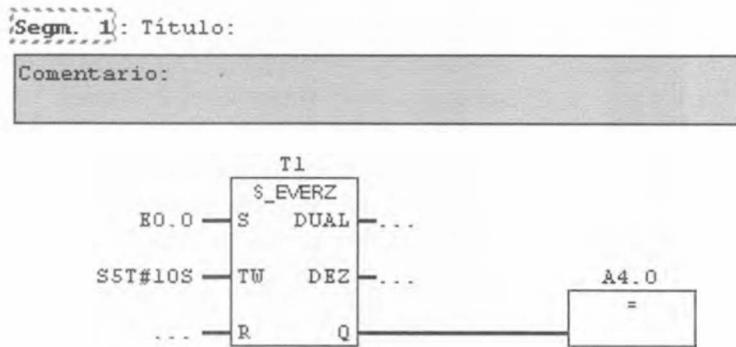


Fig. 60

Si intentamos traducir directamente el programa hecho en AWL a KOP o a FUP, veremos que no es traducible directamente. Si nos fijamos en las cajitas del KOP y del FUP, vemos que han quedado 3 opciones libres. Lo que hemos programado ha sido un temporizador con el mínimo número de instrucciones posibles. Los lenguajes KOP y FUP nos ofrecen más posibilidades en el mismo segmento. Podemos añadir la condición para hacer un **RESET** y también podemos añadir dos variables en las que poder visualizar el tiempo contado en binario o en decimal. En principio estas opciones no las vamos a programar. Las queremos vacías.

Para poder traducir de AWL a KOP deberemos escribir el siguiente programa:

```

U      E      0.0
L      S5T#10S
SE     T      1
NOP    0
NOP    0
NOP    0
U      T      1
=      A      4.0
    
```

La instrucción **NOP 0** significa “nada”. Para que sea traducible, tenemos que decir al programa que en estas opciones que nos da el lenguaje no queremos programar nada.

También podemos hacer directamente el programa en KOP o en FUP. Para ello deberemos tener el editor de textos en el lenguaje deseado antes de empezar a programar. Para insertar los bloques que queremos, deberemos abrir el catálogo de instrucciones. Lo abriremos pulsando el siguiente botón de la barra de herramientas.

 Al pulsar este botón nos aparecerá el catálogo de instrucciones KOP o FUP en la parte izquierda del editor. Aquí tendremos que buscar el bloque deseado. En este caso el temporizador SE.



Fig. 61

Una vez tengamos el bloque en la pantalla del editor, tendremos que rellenar las opciones que nos ofrece con los contactos necesarios. Las opciones que no queramos utilizar las dejaremos en blanco. La traducción de KOP o de FUP a AWL es siempre posible. Si dejamos opciones sin programar, se traducirán como NOP 0.

Ejercicio 10: Semáforo ✓

TEORÍA PREVIA: Temporizadores SE y SI.

DEFINICIÓN Y SOLUCIÓN

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

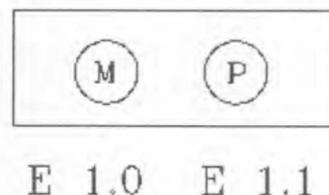
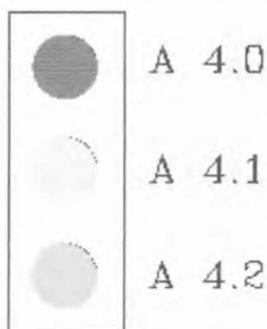


Fig. 62

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

- 1º/ Verde durante 5 seg.
- 2º/ Amarillo durante 2 seg.
- 3º/ Rojo durante 6 seg.

El ciclo es repetitivo hasta que se pulse el botón de paro. En ese momento se apaga todo.

Siempre que le dé al pulsador de marcha quiero que empiece el ciclo por el verde.

Veamos cómo quedaría el ejercicio resuelto utilizando temporizadores SE:

SOLUCIÓN EN AWL

```

U   E   0.0      //Al activar el pulsador de marcha
S   A   4.2      //Encender el verde
U   A   4.2      //Si se ha encendido el verde
L   S5T#5S      //Cuenta 5 segundos
SE  T   1        //Con el temporizador 1
U   T   1        //Y cuando acabes de contar
R   A   4.0      //Apaga el verde
S   A   4.1      //Y enciende el amarillo
U   A   4.1      //Si se ha encendido el amarillo
L   S5T#2S      //Cuenta 2 segundos
SE  T   2        //Con el temporizador 2
U   T   2        //Y cuando acabes de contar
S   A   4.0      //Enciende el rojo
R   A   4.1      //Apaga el amarillo
R   A   4.2      //Y apaga el verde
U   A   4.0      //Si se ha encendido el rojo
L   S5T#6S      //Cuenta 6 segundos
SE  T   3        //Con el temporizador 3
U   T   3        //Cuando acabes de contar
S   A   4.2      //Enciende el verde
R   A   4.0      //Y apaga el rojo
U   E   0.1      //Si se activa el pulsador de paro
R   A   4.0      //Apaga el rojo
R   A   4.1      //Apaga el amarillo
R   A   4.2      //Apaga el verde
    
```

No es necesario resolver el ejercicio con temporizadores SE. Normalmente podremos resolver cualquier problema de tiempos con cualquier tipo de temporizador. Todos los temporizadores cuentan el tiempo preestablecido. Sólo tenemos que tener en cuenta cuando sus *bits* de salida están a 0 o a 1 y consultarlos del modo adecuado para obtener el resultado que nos interesa.

Veamos que este mismo ejercicio lo podemos resolver igualmente con temporizadores SI:

```

U  E  1.0      //Al activar el pulsador de marcha
S  M  0.0      //Nos activamos una marca
U  M  0.0      //Si tenemos el semáforo en marcha
UN A  4.1      //Y no tenemos encendido el amarillo
UN A  4.0      //Y no tenemos encendido el rojo
L  S5T#5S     //Durante 5 segundos
SI  T  1      //Contamos con el T1
U  T  1      //Mientras contamos los 5 segundos
=  A  4.2      //Tenemos encendido el verde
U  M  0.0      //Si tenemos el semáforo en marcha
UN A  4.2      //Y ya se ha apagado el verde
UN A  4.0      //Y no tenemos encendido el rojo
L  S5T#2S     //Cuenta 2 segundos
SI  T  2      //Con el T2
U  T  2      //Mientras contamos los dos segundos
=  A  4.1      //Tenemos encendido el amarillo
U  M  0.0      //Si tenemos el semáforo en marcha
UN A  4.1      //Y ya se ha apagado el amarillo
UN A  4.2      //Y no tenemos encendido el verde
L  S5T#6S     //Contamos 6 segundos
SI  T  3      //Con el T3
U  T  3      //Mientras contamos los 6 segundos
=  A  4.0      //Tenemos encendido el rojo
U  E  1.1      //Si pulsamos paro
R  M  0.0      //Quitamos la marca de semáforo en marcha
    
```

Recomendamos realizar el mismo ejercicio con las siguientes variantes:

- El mismo ejercicio en KOP.
- El mismo ejercicio en FUP.

Los comentarios que aparecen aquí en el libro junto a cada instrucción, también los podemos usar en el editor de bloques de **STEP 7**. Todo lo que vaya precedido de una doble barra (//) el sistema lo interpreta como comentario. Podemos escribir lo que queramos de manera que nos sea más sencillo interpretar el programa que estamos haciendo.

Además de estos comentarios, podemos dar nombre a las variables utilizadas y utilizarlas en el programa. Esto lo vemos resuelto en el siguiente ejercicio. Una vez estudiado el ejercicio 11, se puede volver a los ejercicios anteriores y dar nombre a las variables utilizadas.

2.12 Simbólico global

Ejercicio 11: Direccionamiento simbólico global

TEORÍA

INSERTAR SÍMBOLOS

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

Pero nosotros podemos dar nombre a todo esto. Para ello vamos a la ventana del Administrador de **SIMATIC** y pinchamos en la ventana de **OFFLINE**, en la parte izquierda, encima de donde pone programa **S7** (o el nombre de la carpeta si lo habíamos cambiado anteriormente). En la parte derecha aparece un icono que se llama "Símbolos".



Fig. 63

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

The screenshot shows the 'Semáforo (Símbolos)' dialog box. It contains a table with the following columns: Estado, Símbolo, Dirección, Tipo de dato, and Comentario. The table has one row with the number '1' in the first column.

Estado	Símbolo	Dirección	Tipo de dato	Comentario
1				

Fig. 64

Podemos poner nombre a todo lo que queramos. Tenemos que tener en cuenta que el programa diferencia las mayúsculas de las minúsculas. Si luego intentamos acceder a uno de estos contactos por su nombre, tendremos que escribir el nombre tal y como lo hemos definido diferenciando las mayúsculas de las minúsculas. Una vez observamos estos nombres en el programa, veremos que aparecerán escritos entre comillas. Si definimos los símbolos utilizando solamente números, letras y guiones bajos, podremos programar directamente con los simbólicos y

Recuerda . . .

La tabla de símbolos global será válida para todos los bloques dentro de una carpeta de programa. Encontraremos un icono llamado símbolos dentro de cada carpeta de programa que creemos. El icono de símbolos no aparece en el árbol desplegable.

el editor le pondrá las comillas. Si utilizamos espacios a la hora de definir los símbolos, tendremos que preocuparnos nosotros de poner las comillas para que el editor lo interprete como un solo símbolo. Se recomienda utilizar sólo letras y guiones bajos. También podemos usar números.

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

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

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

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

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

Ejercicio 11: Direccionamiento simbólico global

TEORÍA PREVIA: Insertar símbolos

DEFINICIÓN Y SOLUCIÓN

Vamos a insertar símbolos en el ejercicio del semáforo anterior y veremos como queda en el editor de bloques.

Lo primero que hacemos es rellenar la tabla con los símbolos que queremos utilizar. Daremos nombre a los pulsadores de marcha y paro, a las tres luces del semáforo y a los tres temporizadores utilizados.

Obtendremos una tabla como esta:

	Estado	Símbolo	Dirección	Tipo de dato	Comentario
1		Pulsador_marcha	E 1.0	BOOL	
2		Pulsador_paro	E 1.1	BOOL	
3		Verde	A 4.0	BOOL	
4		Amarillo	A 4.1	BOOL	
5		Rojo	A 4.2	BOOL	
6		Tiempo_de_verde	T 1	TIMER	
7		Tiempo_de_amarillo	T 2	TIMER	
8		Tiempo_de_rojo	T 3	TIMER	
9					

Fig. 65

Si ahora vamos a observar nuestro bloque OB1 con el programa del semáforo, podremos observarlo de las siguientes maneras:

Inicialmente teníamos esto:

```

U E 1.0 //Al activar el pulsador de marcha
S A 4.2 //Encender el verde
U A 4.2 //Si se ha encendido el verde
L S5T#5S //Cuenta 5 segundos
SE T 1 //Con el temporizador 1
U T 1 //Y cuando acabes de contar
S A 4.1 //Enciende el amarillo
    
```

Si queremos ver los símbolos, pulsamos el botón de la barra de herramientas que representa una etiqueta.



También podemos ver los símbolos desde el menú “Ver” → “mostrar” → “representación simbólica”. Entonces veríamos el programa de la siguiente forma:

```

U "Pulsador_marcha" //Al activar el pulsador de marcha
S "Verde" //Encender el verde
U "Verde" //Si se ha encendido el verde
L S5T#5S //Cuenta 5 segundos
SE "Tiempo_de_verde" //Con el temporizador 1
U "Tiempo_de_verde" //Y cuando acabes de contar
S "Amarillo" //Enciende el amarillo
    
```

Fig. 66

También tenemos la opción de ver los contactos reales y los símbolos en una misma línea de programa. Si escogemos esta visualización, perderemos los comentarios con //. Para ello seleccionamos en el menú “Ver” → “mostrar” la opción “información del símbolo”. Quedaría el programa así:

```

U "Pulsador_marcha" E1.0
S "Verde" A4.2
U "Verde" A4.2
L S5T#5S
SE "Tiempo_de_verde" T1
U "Tiempo_de_verde" T1
S "Amarillo" A4.1
    
```

Fig. 67

Si ahora quitamos la opción de “representación simbólica” pero mantenemos la información del símbolo, el bloque quedaría así:

```

U E 1.0 Pulsador_marcha
S A 4.2 Verde
U A 4.2 Verde
L S5T#5S
SE T 1 Tiempo_de_verde
U T 1 Tiempo_de_verde
S A 4.1 Amarillo
    
```

Fig. 68

Tenemos múltiples formas de ver los símbolos y los comentarios. Cada uno puede elegir la que le sea más práctica a la hora de trabajar, programar o interpretar un programa ya hecho.

2.13 Cintas transportadoras

Ejercicio 12: Cintas transportadoras

TEORÍA PREVIA: Temporizadores SE y SI.

DEFINICIÓN Y SOLUCIÓN

Tenemos tres cintas transportadoras dispuestas de la siguiente manera:

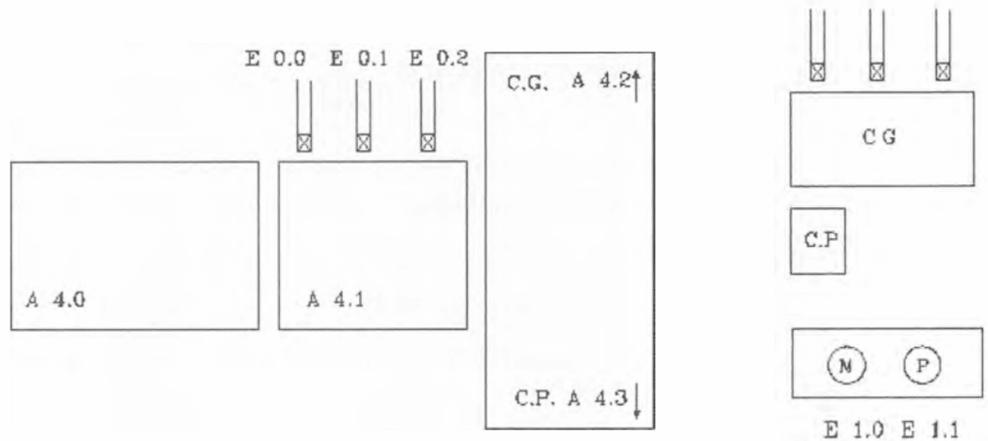
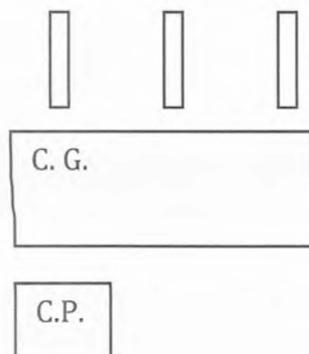


Fig. 69

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



Las cajas grandes son de un tamaño tal que en un momento dado de su recorrido, la caja tapa las tres fotocélulas. Las cajas pequeñas son de un tamaño tal que las fotocélulas las detectan de una en una. Cuando durante el recorrido, empieza a ver la segunda fotocélula, la primera ya ha dejado de detectar.

El funcionamiento que queremos es el siguiente:

Cuando le demos al pulsador de marcha queremos que se ponga en marcha únicamente la cinta nº 1. Cuando llegue la primera caja a la cinta nº 2, queremos que se pare la cinta nº 1 y que se ponga en marcha la cinta nº 2. En la cinta nº 2 detectamos si la caja es grande o pequeña. Si es grande, queremos que se ponga en marcha la tercera cinta hacia la izquierda y, si es pequeña, queremos que se ponga en marcha la tercera cinta hacia la derecha. La cinta nº 2 se para cuando la caja ya esté abandonando la cinta nº 2. La cinta nº 3 se para a los 10 seg. de haberse puesto en marcha. A continuación se pone en marcha de nuevo la primera cinta y vuelve a comenzar el ciclo con la siguiente caja. Las cajas siempre harán el ciclo de una en una.

Tenemos que añadir un nuevo OB1 en el proyecto que ya tenemos, para programar este ejemplo. En este caso vamos a insertar un equipo con la CPU y las tarjetas que tenemos. Veremos que nos interesará modificar alguna propiedad de la CPU. En los dos ejemplos anteriores nos ha sido suficiente crear carpetas de programa sin *hardware*. Ahora insertamos un equipo y sus tarjetas tal y como explicamos al inicio de este manual. El proyecto nos quedará como éste:



Fig. 70

En este OB1 haremos el nuevo programa.

SOLUCIÓN EN AWL

U	E	1.0	//Si le damos al pulsador de marcha
S	A	4.	//Pon en marcha la primera cinta
U	E	0.0	//Cuando la caja cambie de cinta
S	A	4.1	//Pon en marcha la segunda cinta
R	A	4.0	//y para la primera
U	E	0.0	//Si ve la primera célula
U	E	0.1	//Y ve la segunda célula
U	E	0.2	//Y ve la tercera célula
S	A	4.2	//Pon en marcha la cinta de caja grande
UN	E	0.0	//Si no ve la primera célula
U	E	0.1	//Y si que ve la segunda célula

```

UN   E   0.2      //Y no ve la tercera célula
S    A   4.3      //Pon en marcha la cinta de caja pequeña
UN   E   0.0      //Si no ve la primera célula
UN   E   0.1      //Y no ve la segunda célula
U    E   0.2      //Y si que ve la tercera célula
R    A   4.1      //Para la segunda cinta
U    A   4.2      //Si está en marcha la cinta de caja grande
O    A   4.3      //O la cinta de caja pequeña
L    S5T#10S     //Cuenta 10 segundos
SE   T    1      //Con el T 1
U    T    1      //Y cuando acabes de contar
R    A   4.2      //Para la cinta de caja grande
R    A   4.3      //Para la cinta de caja pequeña
S    A   4.0      //Y pon en marcha la primera cinta
U    E   1.1      //Si pulsamos el paro de emergencia
R    A   4.0      //Para la primera cinta
R    A   4.1      //Para la segunda cinta
R    A   4.2      //Para la cinta de caja grande
R    A   4.3      //Para la cinta de caja pequeña

```

UN POCO MÁS DE TEORÍA

Si probamos este programa, en principio funciona bien. Las cajas pasan por las cintas tal y como se ha explicado en las especificaciones. Pero supongamos que por un momento nos quedamos sin tensión. En ese momento, por supuesto, se pararían todas las cintas. Imaginemos que en la tercera cinta (o en la primera) había una caja circulando. Quedaría parada antes de terminar el recorrido. Al volver la tensión, la instalación “no sabrá” que ha quedado una caja a medias. Como allí no hay fotocélulas, no hay manera de detectarla. Se juntaría en el recorrido con la siguiente caja que pasase. Podría darse el caso de que una caja pequeña circulase en el sentido de una grande o viceversa.

Para arreglar esto y para que el sistema “se acuerde” de las cajas que tenía circulando antes de parar, podemos utilizar **marcas remanentes** en lugar de trabajar directamente con las salidas.

De este modo el sistema se “acordaría” de las cajas que estaban circulando antes del corte de suministro y podría arrancar en el momento del ciclo en el que se quedó.

Para hacer esto, cambiaríamos las salidas por marcas en el programa original. Donde ponía A4.0 pondremos M0.0, donde ponía A4.1 pondremos M0.1, donde ponía A4.2 pondremos M0.2 y donde ponía A4.3 pondremos M0.3.

Recuerda . . .

Las áreas de remanencia del PLC las puede configurar el usuario desde dentro de las propiedades de la CPU. Para ello será necesario haber creado previamente un hardware con la CPU utilizada.

Al final del programa añadiremos un segmento como éste para asignar las marcas a las salidas correspondientes.

```

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

Veamos ahora como hacemos que estas marcas sean remanentes y cual será el nuevo funcionamiento del programa.

Marcas remanentes son aquellas que ante un corte de tensión mantienen su valor. También si la CPU pasa a **STOP** y de nuevo a **RUN**. Por defecto, tenemos los primeros 16 *bytes* de marcas remanentes.

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

Veremos que aparecen unas fichas. Una de ellas se llama "Remanencia". Entramos en esta ficha. Vemos que podemos definir la cantidad de marcas remanentes que nosotros queramos. Siempre serán los primeros bytes a partir del 0 los que son remanentes. Por defecto vemos que tenemos los 16 primeros bytes de marcas remanentes.

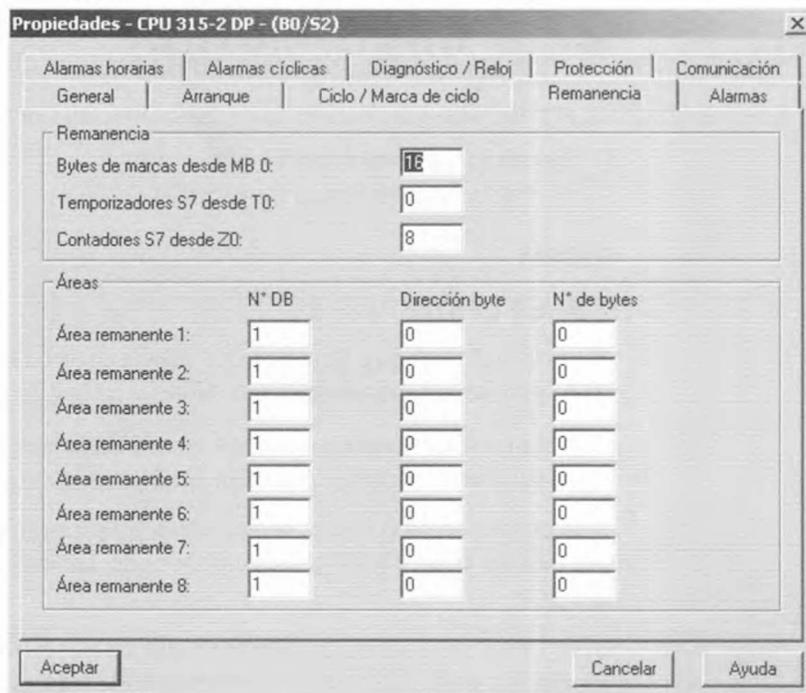


Fig. 71

Si ahora probamos el ejemplo anterior, después de haber hecho el cambio de salidas por marcas, veremos que podemos quitar tensión al PLC o pasarlo a **STOP** y, cuando vuelva a arrancar, estará en marcha la cinta o las cintas que estuviesen antes de la parada.

Podemos cambiar en la ficha de propiedades la cantidad de marcas remanentes y poner 0. Veremos que entonces el PLC “no se acuerda” de nada ante una parada.

También podemos probar a poner 2 temporizadores remanentes (el 0 y el 1) y así veremos que también se “acuerda” del tiempo que le quedaba por contar en la tercera cinta.

Aunque todavía no se han explicado los contadores, vemos en esta ficha de propiedades que podríamos tener contadores remanentes y no remanentes. Esto quiere decir que ante una parada de la CPU se “acordará” o no se “acordará” de la cuenta que llevase hecha.

Este mismo programa también lo podríamos hacer con direccionamiento simbólico.

Si utilizamos el direccionamiento simbólico podríamos hacer lo siguiente:

```

U    "MARCHA"
S    "1ª_CINTA"
U    "CELULA_1"
S    "2ª_CINTA"
R    "1ª_CINTA"

```

.....

2.14 Intermitente

Ejercicio 13: Intermitente

TEORÍA

FINALES EN STEP 7

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

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

Si la condición se cumple, será un final de programa. Si la condición no se cumple, no será un final y la CPU seguirá leyendo las instrucciones que tiene por debajo del BEB.

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

Recuerda . . .

La instrucción BE (final de bloque), no es necesaria. Si la CPU ve que en un bloque no hay más instrucciones, entiende que ha terminado el bloque. Dependiendo del tipo de bloque que estemos programando, la CPU ya sabrá por dónde debe continuar el programa. Si es el final del OB1, volverá al inicio del programa.

Veamos un ejemplo en el que podemos utilizar los BEA.

Supongamos que queremos el siguiente funcionamiento:

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

Veamos como programaríamos esto de modo genérico. Todavía no hemos visto cómo se programan los saltos. Se analizará en posteriores ejercicios. Pero lo explicamos de modo gráfico para entender un poco mejor la necesidad y las diferencias entre los finales explicados.

Esto lo programaríamos del siguiente modo:

```

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

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

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

Veamos un ejemplo de cómo funcionaría el BEB.

```

U    E    0.0
=    A    4.0
U    E    0.1
BEB
U    E    0.2
=    A    4.2
BE
    
```

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

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

Ejercicio 13: Intermitente ✓

TEORÍA PREVIA: Finales BEB y BEA

DEFINICIÓN Y SOLUCIÓN

Vamos a hacer un intermitente utilizando un solo temporizador de 1 segundo. Queremos que una salida esté activa durante un segundo y no activa durante el siguiente segundo. Queremos que haga esto de manera continua y sin ninguna condición previa. Nada más enviar el programa a la CPU queremos que se ponga en marcha el intermitente.



Fig. 72

SOLUCIÓN EN AWL

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

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

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

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

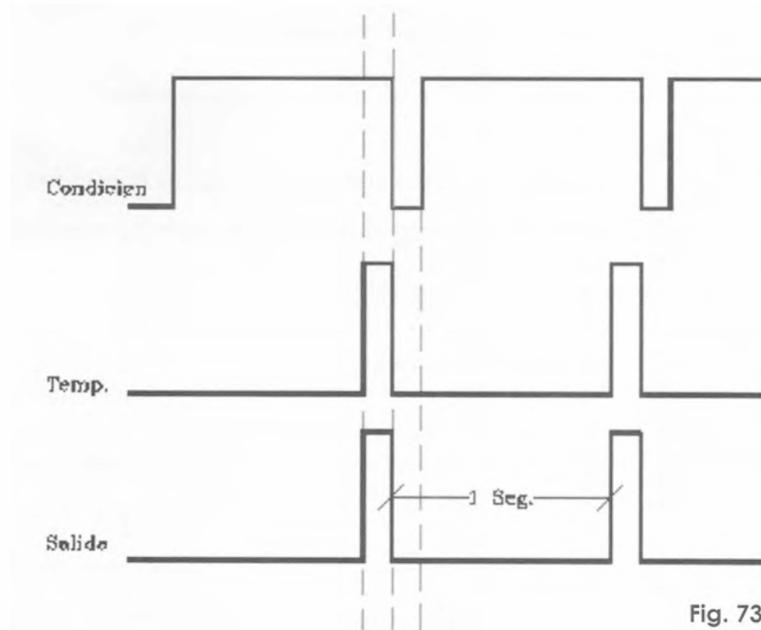


Fig. 73

2.15 Semáforo con intermitencia

Ejercicio 14: Semáforo con intermitencia ✓

TEORÍA PREVIA: Intermitente (BEB).

DEFINICIÓN Y SOLUCIÓN

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

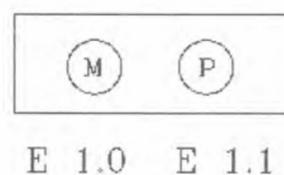
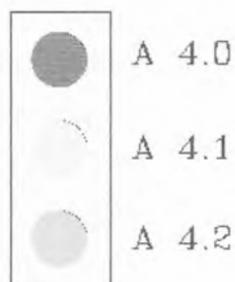


Fig. 74

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

1º/ Verde durante 5 seg.

2º/ Amarillo intermitente durante 2 seg.

3º/ Rojo durante 6 seg.

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

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

SOLUCIÓN EN AWL

U	E	0.0	//Si le damos al pulsador de marcha
S	A	4.2	//Enciende el verde
U	A	4.2	//Si se ha encendido el verde
L	S5T#5S		//Cuenta 5 segundos
SE	T	1	//Con el temporizador 1
U	T	1	//Cuando acabes de contar
R	A	4.2	//Apaga el verde
S	M	10.0	//Y activa la marca 10.0
U	M	10.0	//Si está activa la marca 10.0
U	M	0.1	//Y está activa la marca 0.1
=	A	4.1	//Enciende el amarillo
U	M	10.0	//Si está activa la marca 10.0
L	S5T#2S		//Cuenta 2 segundos
SE	T	2	//Con el temporizador 2
U	T	2	//Cuando acabes de contar
R	M	10.0	//Desactiva la marca 10.0
S	A	4.0	//Y enciende el rojo
U	A	4.0	//Si se ha encendido el rojo
L	S5T#6S		//Cuenta 6 segundos
SE	T	3	//Con el temporizador 3
U	T	3	//Cuando acabes de contar
R	A	4.0	//Apaga el rojo
S	A	4.2	//Y enciende el verde
U	E	0.1	//Si le damos al pulsador de paro
R	A	4.0	//Apaga el rojo
R	M	10.0	//Apaga la marca de amarillo
R	A	4.2	//Apaga el verde

```

UN  M   0.0      //Hacemos que la marca 0.0 se active
L   S5T#200MS   //Una vez cada 200 milisegundos
SE  T    4
U   T    4
=   M   0.0
UN  M   0.0
BEB
UN  M   0.1      //La marca 0.1 estará 200 milisegundos activa
=   M   0.1      //Y 200 milisegundos no activa
    
```

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

2.16 Parking de coches

Ejercicio 15: Parking de coches

TEORÍA

CONTADORES Y COMPARACIONES

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

```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Ejercicio 15: *Parking* de coches ✓

DEFINICIÓN Y SOLUCIÓN

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

Tenemos el siguiente *parking* de coches:

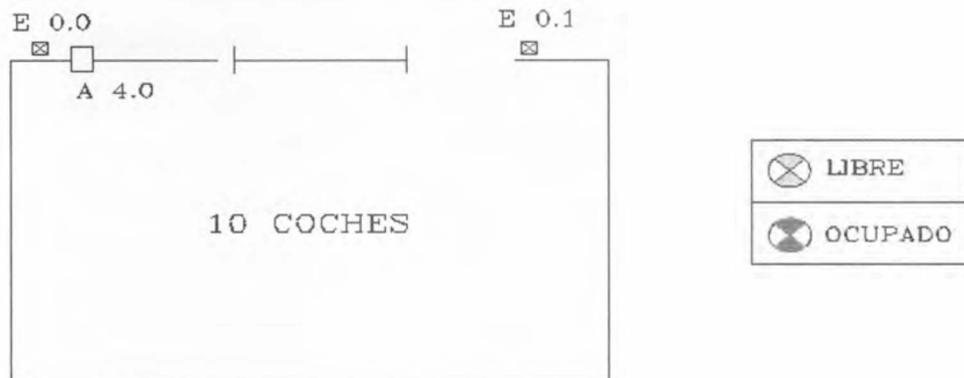


Fig. 75

El funcionamiento que queremos es el siguiente:

Cuando llega un coche y el *parking* está libre, queremos que se abra la barrera. A la salida no tenemos barrera. Cuando sale un coche simplemente sabemos que ha salido.

En el *parking* caben 10 coches. Cuando el *parking* tenga menos de 10 coches queremos que esté encendida la luz de libre. Cuando en el *parking* haya 10 coches queremos que esté encendida la luz de ocupado.

Además queremos que si el *parking* está ocupado y llega un coche no se le abra la barrera.

SOLUCIÓN EN AWL

```

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

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

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

SOLUCIÓN AWL

```

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

En el primer ejercicio se cuentan los coches que entran en el *parking*. En el segundo ejercicio se cuentan las plazas libres que tenemos.

2.17 Puerta corredera



Ejercicio 16: Puerta corredera

TEORÍA

TEMPORIZADORES SS, SV y SA

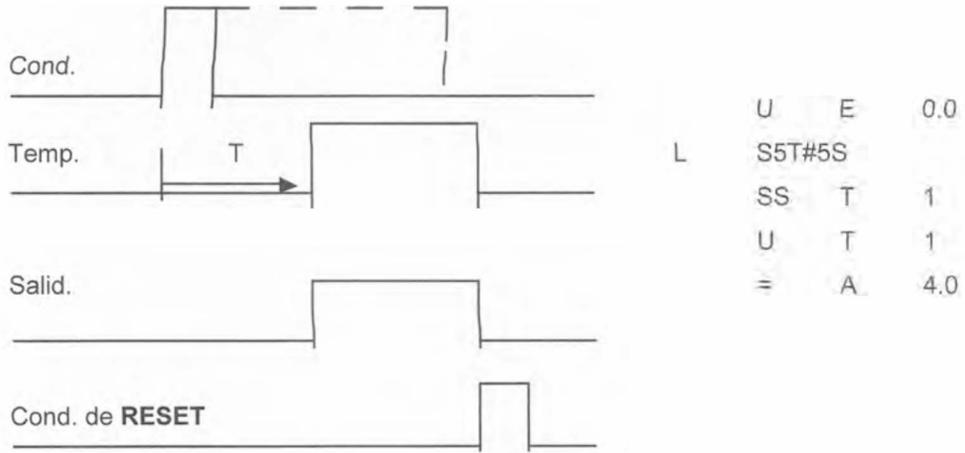
Además de los temporizadores que hemos visto en ejercicios anteriores, tenemos tres más llamados temporizadores con memoria. Son los temporizadores SS, SV y SA.

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

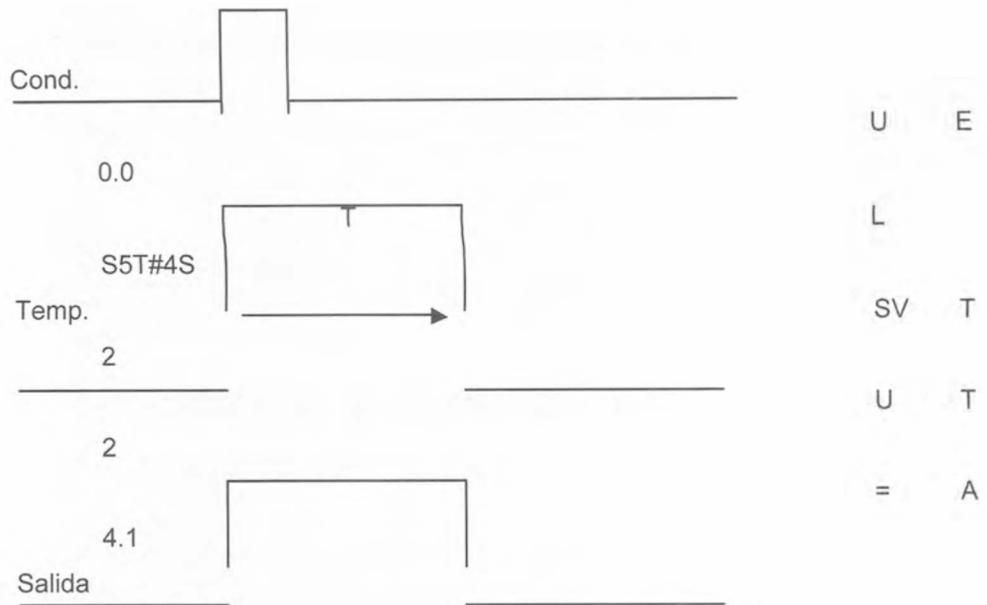
Recuerda . . .

En Step 7 existen cinco tipos de temporizadores para facilitar la programación al usuario. SE, SI, SS, SV y SA. No obstante, sabiendo cómo funciona cada uno de ellos, podríamos programar cualquier aplicación con cualquiera de los cinco tipos de temporizador.

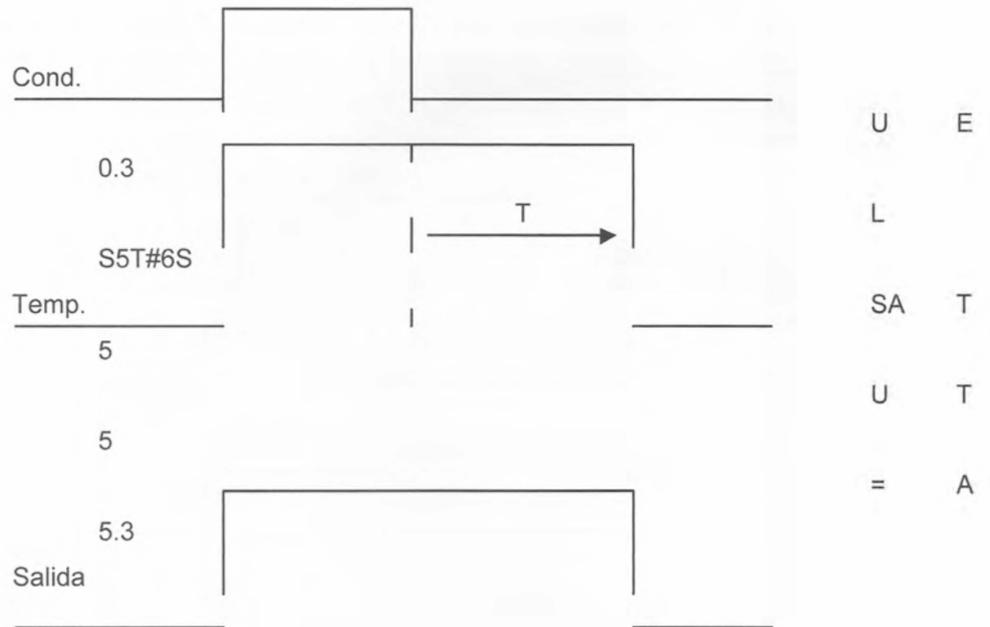
A continuación vemos un esquema del funcionamiento del temporizador. Observamos que tenemos un problema. El temporizador se queda a uno si nadie lo *resetea*. Necesitamos añadir una condición que *resete* el temporizador para que vuelva a su estado inicial y lo podamos volver a utilizar.



El temporizador SV es equivalente al SI. El funcionamiento es el mismo, pero es independiente de la condición de entrada. Una vez se ha detectado un flanco de subida de la entrada se ejecuta todo el ciclo del temporizador. Veamos el esquema de funcionamiento.



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

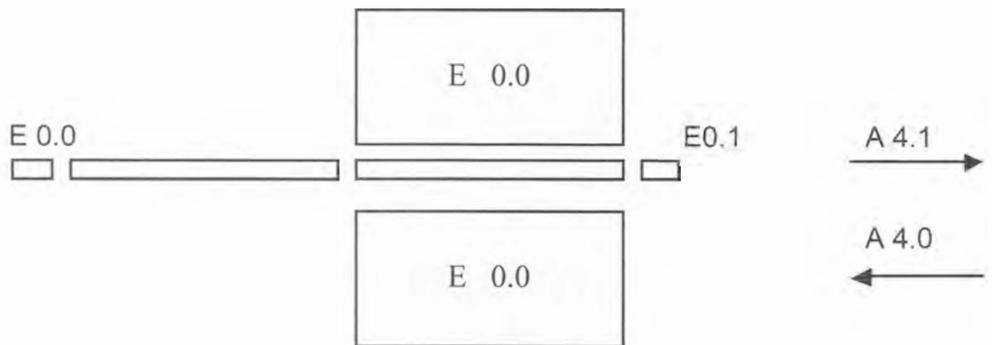


Ejercicio 16: Puerta corredera ✓

TEORÍA PREVIA: Temporizadores + contadores.

DEFINICIÓN Y SOLUCIÓN

Tenemos una puerta corredera. El funcionamiento de la puerta es el siguiente:



Queremos que cuando alguien pise en la goma del suelo, se abra la puerta. Motor de apertura A 4.0. La puerta se está abriendo hasta que llegue al final de carrera. Cuando llega al final de carrera, comienza a cerrarse (Motor A 4.1). Se está cerrando hasta que llega al final de carrera.

Tenemos dos pulsadores de control. El de marcha y el de paro. Cuando le demos al pulsador de marcha queremos que el funcionamiento sea el que hemos explicado anteriormente. Cuando le demos al de paro queremos que deje de funcionar. Es decir, si alguien pisa la goma no queremos que se abra la puerta.

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

Volverá a funcionar cuando desbloquemos la puerta.

SOLUCIÓN EN AWL

```

U   E   0.6   //Si le damos al pulsador de marcha
S   M   0.0   //Activa la marca 0.0
U   E   0.7   //Si le damos al pulsador de paro
R   M   0.0   //Desactiva la marca 0.0
U   E   0.0   //Si alguien pisa la goma
U   M   0.0   //Y está la puerta en marcha
U   Z   1     //Y el contador 1 tiene un valor distinto de 0
S   A   4.0   //Y activa el motor de abrir
U   E   1.0   //Si llega el final de carrera
R   A   4.0   //Para el motor de apertura
S   A   4.1   //Y pon en marcha el motor de cierre
U   E   1.1   //Si se ha cerrado la puerta
R   A   4.1   //Para el motor de cierre
UN  E   1.7   //Si ha saltado el relé térmico
ZR  Z   1     //Descuenta una unidad en el contador 1
R   A   4.0   //Y para el motor de abrir
R   A   4.1   //Y para el motor de cerrar
U   E   1.6   //Si activamos la entrada 1.6
L   C#5     //Carga un 5
S   Z   1     //Y mételo en el contador 1
    
```

2.18 Contar y descontar cada segundo

Ejercicio 17: Contar y descontar cada segundo 

TEORÍA PREVIA: Intermitente + contadores.

DEFINICIÓN Y SOLUCIÓN

Queremos hacer un contador que a partir de que le demos al pulsador de marcha, comience a contar una unidad cada segundo hasta llegar a 60. Cuando llegue a 60 queremos que siga contando una unidad cada segundo pero en sentido descendente.

Queremos que haga la siguiente cuenta:

0, 1, 2, 3, 4,, 58, 59, 60, 59, 58, 57,....., 2, 1, 0, 1, 2,

Cuando le demos al pulsador de paro queremos que deje de contar. Cuando le demos de nuevo al pulsador de marcha probaremos dos cosas:

Que siga contando por donde iba.

Que empiece a contar otra vez desde cero.

SOLUCIÓN EN AWL

```

UN   M   0.0           //Hacemos que la marca 0.0
L    S5T#1S           //Se active un ciclo cada segundo
SE   T    1
U    T    1
=    M   0.0
U    E   0.0           //Si le damos al pulsador de marcha
S    M   0.1           //Se activa la marca 0.1
R    M   0.3           //Y se desactiva la marca 0.3
(R   Z    1)           Empezará desde cero o por donde iba.
U    M   0.1           //Si está activa la marca 0.1
U    M   0.0           //Y llega un pulsa de la marca 0.0
UN   M   0.3           //Y no está activa la marca 0.3
ZV   Z    1           //Cuenta una unidad con el contador 1
L    Z    1           //Carga el contador 1
L    60              //Carga un 60
==|                          //Cuando sean iguales
S    M   0.2           //Activa la marca 0.2
R    M   0.1           //Y desactiva la marca 0.1
U    M   0.2           //Si está la marca 0.2
U    M   0.0           //Y llega un pulso de la marca 0.0
UN   M   0.3           //Y no está la marca 0.3
ZR   Z    1           //Descuenta 1 con el contador 1
L    Z    1           //Carga el contador 1
L    0               //Carga un 0
==|                          //Si son iguales
S    M   0.1           //Activa la marca 0.1
R    M   0.2           //Y desactiva la marca 0.2
U    E   0.1           //Si le damos al paro
S    M   0.3           //Activa la marca 0.3
    
```

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

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

2.19 Fábrica de curtidos

Ejercicio 18: Fábrica de curtidos ✓

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Temporizadores + contadores.

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

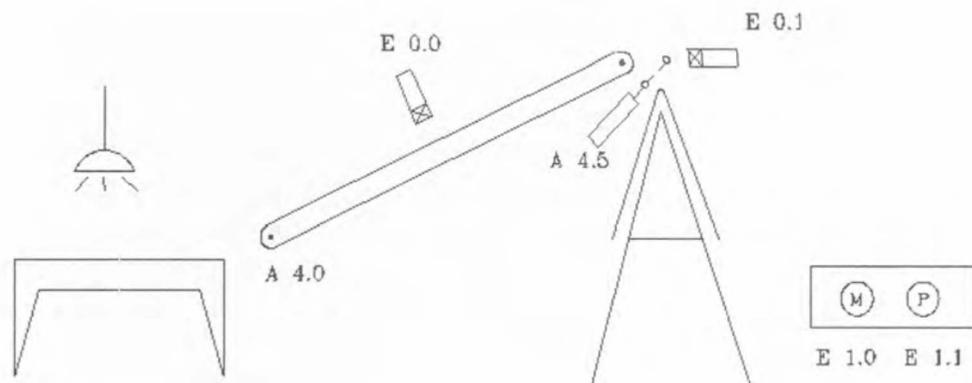


Fig. 76

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

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

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

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

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

SOLUCIÓN EN AWL

```

U   E   1.0           //Si le damos al botón de marcha
S   A   4.0           //Pon en marcha la cinta
UN  M   0.0           //Generamos unos pulsos
L   S5T#10MS         //de 10 milisegundos
SE  T   1             //con la marca 0.0
U   T   1
=   M   0.0
UN  M   0.1           //Generamos unos pulsos
L   S5T#20MS         //de 20 milisegundos
SE  T   2             //con la marca 0.1
U   T   2
=   M   0.1
U   E   0.0           //Mientras esté la primera célula activa
U   M   0.1           //Y lleguen pulsos de frecuencia lenta
ZV  Z   1             //Cuéntalos con el contador 1
U   E   0.1           //Mientras está activa la segunda célula
U   M   0.0           //Y lleguen los pulsos rápidos
ZV  Z   2             //Cuéntalos con el contador 2
L   Z   1             //Carga el contador 1
L   Z   2             //Carga el contador 2
==|
S   A   4.5           //Activa el émbolo
U   A   4.5           //Cuando hayas activado el émbolo
L   S5T#3S           //Cuenta 3 segundos
SE  T   3             //Con el temporizador 3
U   T   3             //Cuando acabes de contar
R   A   4.5           //Desactiva el émbolo
R   Z   1             //Resetea el contador 1
R   Z   2             //Y resetea el contador 2
U   E   1.1           //Si pulsamos el paro
R   A   4.0           //Para la cinta
    
```

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

2.20 Escalera automática

Ejercicio 19: Escalera automática

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Temporizadores SA

Tenemos una escalera automática.

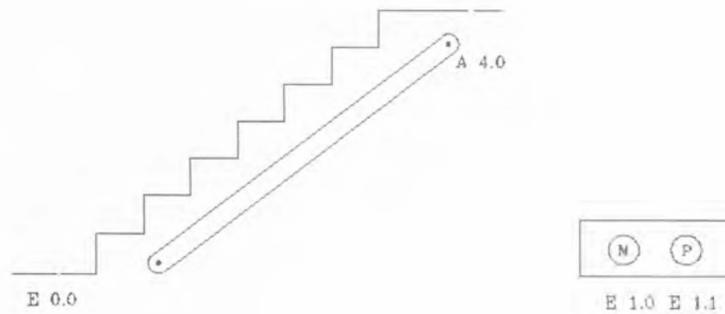


Fig. 77

El funcionamiento que queremos es el siguiente:

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

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

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

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

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

SOLUCIÓN EN AWL

```

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

2.21 Instrucción MASTER CONTROL RELAY

Ejercicio 20: Instrucción MASTER CONTROL RELAY

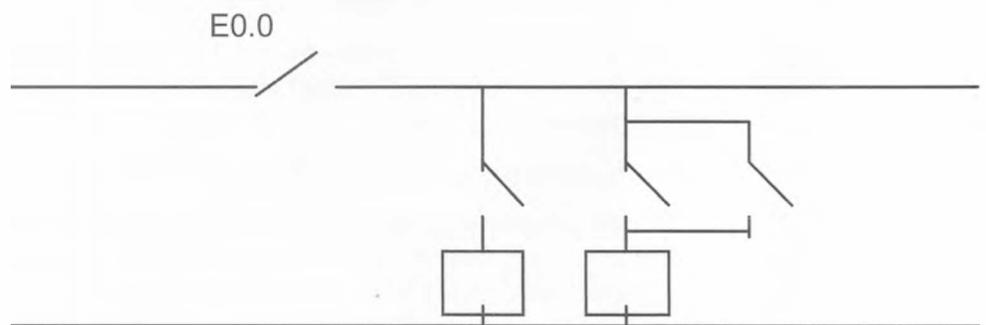
TEORÍA

INSTRUCCIÓN MASTER CONTROL RELAY

El MASTER CONTROL RELAY consta de 4 instrucciones:

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

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



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

Ejercicio 21: Instrucción MASTER CONTROL RELAY

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Introducción teórica a la instrucción **MASTER CONTROL RELAY**.

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

Es la instrucción **MASTER CONTROL RELAY**.

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

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

Está pensado para utilizar contactos con asignaciones “=”. Viene a ser como un circuito eléctrico. Lo que quede activado cuando no se ejecuta lo que hay dentro de los paréntesis del **MASTER CONTROL RELAY**, se desactiva.

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

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

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

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

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

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

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

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

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

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

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

Unidad 3 Operaciones de *byte*, palabras y dobles palabras



En este capítulo:

- 3.1. Instrucciones de carga y transferencia
- 3.2. Ejercicio de metas
- 3.3. Trabajar con DB
- 3.4. Pesar productos dentro de unos límites
- 3.5. Introducción a la programación estructurada
- 3.6. Desplazamiento y rotación de bits
- 3.7. Planta de embotellado
- 3.8. FC con y sin parámetros
- 3.9. Crear un DB con la SFC 22
- 3.10. Sistemas de numeración
- 3.11. Carga codificada
- 3.12. Operaciones con enteros
- 3.13. Conversiones de formatos
- 3.14. Operaciones con reales
- 3.15. Control de un gallinero
- 3.16. Operaciones de salto
- 3.17. Mezcla de pinturas
- 3.18. Instrucciones NOT, CLR, SET y SAVE
- 3.19. Ajuste de valores analógicos
- 3.20. Ejemplo con UDT
- 3.21. Operaciones lógicas con palabras
- 3.22. Ejemplo de alarmas

3.1 Instrucciones de carga y transferencia

Ejercicio 1: Instrucciones de carga y transferencia

TEORÍA

INSTRUCCIONES DE CARGA Y TRANSFERENCIA

Hasta ahora, en el primer capítulo de este libro, hemos trabajado con *bits*. Escribíamos instrucciones del tipo:

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

Si quisiésemos hacer esto mismo pero con todas las entradas y todas las salidas que tenemos disponibles (que cada entrada active una salida), podríamos hacerlo *bit a bit*, o trabajando directamente con la **palabra de entradas** y la **palabra de salidas** en bloque. Una palabra son 16 "*bits* juntos". En un *bit* cabe muy poca información (0 ó 1). En cambio, en una palabra, combinando ceros y unos en los 16 *bits*, cabe muchísima más información. Utilizaremos las palabras para tratar datos con valores diferentes de 1 o 0. Por ejemplo, podremos guardar una temperatura, un peso, una cantidad, una distancia, una lectura de una etiqueta, etcétera.

Además de tratar los *bits* de 16 en 16 (palabras), podemos trabajar con ellos en *bytes* (8 bits) o en dobles palabras (32 *bits*). En un principio no podremos trabajar con formatos de datos de más de 32 *bits* porque los registros internos del PLC son de 32 *bits*. En el siguiente capítulo de este libro, veremos que existen formatos de más de 32 *bits*. Estos datos no los podremos utilizar con las instrucciones que veremos ahora. Ya veremos que existen FC específicas para tratar estos tipos de formatos.

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

```
E 0.0      A 4.0      M 0.0
```

También podemos acceder a *bytes*.

```
EB 0      AB 4      MB 0
```

Esto es el *byte* de entradas 0, el *byte* de salidas 4 y el *byte* de marcas 0. Los *bytes* son 8 *bits* consecutivos en el siguiente orden:

E 0.7	E 0.6	E 0.5	E 0.4	E 0.3	E 0.2	E 0.1	E 0.0
-------	-------	-------	-------	-------	-------	-------	-------

En todos los *bytes* veremos que el *bit* *.0 queda en la parte derecha. Es el de menor peso. El *bit* *.7 queda en la parte izquierda. Es el de mayor peso.

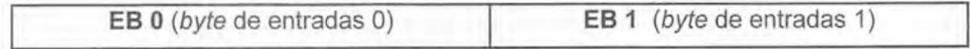
Recuerda . . .

Siempre el *byte* de mayor número será el de menor peso. Debemos tener esto en cuenta sobre todo a la hora de comunicarnos con otras aplicaciones.

También podemos acceder a palabras (16 *bits*).

EW 0 AW 4 MW 0

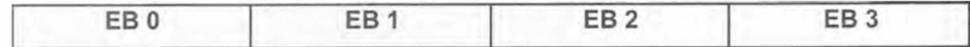
Esto es la palabra de entradas 0, la palabra de salidas 4 y la palabra de marcas 0. Una palabra es dos *bytes* consecutivos dispuestos en el siguiente orden:



También podemos trabajar con dobles palabras.

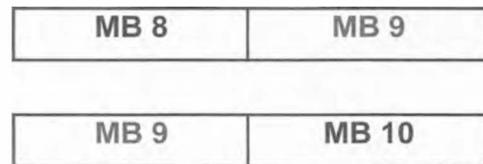
ED 0 AD 4 MD 0

Esto es la doble palabra de entradas 0, la doble palabra de salidas 4 y la doble palabra de marcas 0. Una doble palabra es 4 *bytes* consecutivos dispuestos en el siguiente orden:

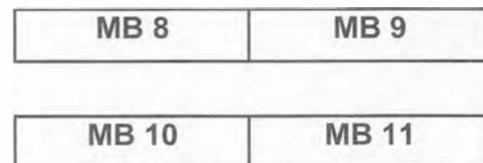


Como podemos observar, el *byte* de entradas 0, la palabra de entradas 0 y la doble palabra de entradas 0 utilizan *bits* comunes. Tendremos que tener esto en cuenta a la hora de utilizar este tipo de datos.

Veamos un ejemplo. Si utilizamos la palabra de marcas 8 y la palabra de marcas 9, estaremos utilizando lo siguiente:



El *byte* de marcas 9 lo estamos gastando en ambas palabras. Si introducimos valores en la palabra de marcas 8, estaremos modificando el valor de la palabra de marcas 9. Esto no es un error de programación. El *software* nos permite hacerlo. Podemos querer pasar el valor de la parte baja de una palabra, a la parte alta de otra. Sería tan fácil como usar estas dos palabras de marcas. Pero si lo que queremos es utilizar valores totalmente independientes deberíamos utilizar palabras de marcas independientes. Que no solapen *bits*. Por ejemplo, podríamos utilizar el *byte* de marcas 8 y el *byte* de marcas 10.



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

Instrucción de carga: L
 Instrucción de transferencia: T

Cuando cargamos algo, lo hacemos en un registro interno de la CPU llamado **acumulador 1 (ACU 1)**. Cuando transferimos, lo que hacemos es coger lo que hay en el acumulador 1 y llevarlo a donde le decimos. El acumulador 1 es un registro interno del autómata de 32 *bits*. Nos sirve para hacer tratamiento de valores que no sean *bits* sueltos y que no sobrepasen los 32 *bits*.

A través del acumulador podremos hacer comparaciones, movimientos de datos, operaciones matemáticas, etcétera.

Las instrucciones de carga y transferencia son siempre incondicionales. Siempre que el PLC lea la instrucción la ejecutará sin consultar el RLO anterior.

Ejercicio 1: Instrucciones de carga y transferencia ✓

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Instrucciones de carga y transferencia.

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

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

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

```

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

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

El programa quedaría del siguiente modo:

```

L      EW      0
T      AW      4
    
```

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

Como ya comentamos en la parte de teoría, hay que tener en cuenta que las operaciones de carga y transferencia son **incondicionales**. Esto quiere decir que **NO** debemos hacer programas como el siguiente:

```

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

En este caso el autómatas no daría error puesto que cada instrucción por separado es correcta, pero no haría caso a la entrada. Tanto si estuviera como si no, la operación de carga y transferencia se ejecutaría igualmente.

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

Veamos cómo lo haríamos:

Solución en KOP

```
OB1 : "Main Program Sweep (Cycle)"
```

Comentario:

Segm. 1: Título:

Comentario:

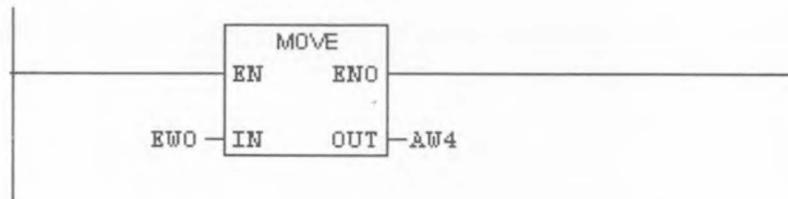


Fig. 78

Solución en FUP

```
OB1 : "Main Program Sweep (Cycle)"
```

Comentario:

Segm. 1: Título:

Comentario:

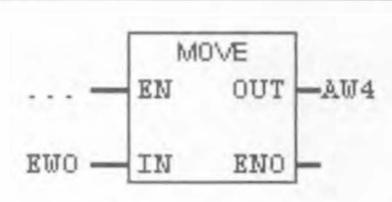


Fig. 79

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

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

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

Como ya dijimos al principio de este manual, todo lo podemos hacer en los tres lenguajes. Lo único es que no todo siempre es traducible o programable con las mismas instrucciones. Si ponemos una condición y traducimos a AWL veremos lo siguiente:

Solución en KOP con condición de entrada:



Fig. 79

Ahora podemos traducir esto a AWL. El programa lo traducirá de la siguiente manera:

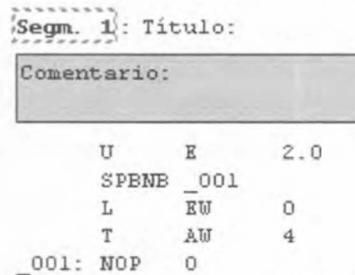


Fig. 81

En este manual todavía no se ha explicado la teoría suficiente para entender este código. No obstante se ha hecho la traducción para observar que aunque no sean instrucciones exactamente equivalentes, todo lo podemos acabar resolviendo en cualquiera de los tres lenguajes de un modo u otro. En este caso, la incondicionalidad de las instrucciones "L" y "T" se ha solventado con una instrucción de salto.

Lo mismo obtendríamos si hiciésemos la traducción del FUP.

3.2 Ejercicio de metas

Ejercicio 2: Ejercicio de metas

TEORÍA

FORMATOS DE CARGA. SALTOS CONDICIONAL E INCONDICIONAL

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

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

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

Veamos cuatro formatos nuevos:

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

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

Para saltar, tenemos dos instrucciones básicas:

SPA:	Salto absoluto.
SPB:	Salto condicional.

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

Nos sirven dentro de todo tipo de bloques de programación.

Recuerda . . .

Los diferentes finales utilizados en AWL como instrucciones no existen en KOP ni en FUP.

No obstante, podemos programar la misma aplicación utilizando los saltos o los bloques MOVE condicionales.

Ejercicio 2: Ejercicio de metas ✓

DEFINICIÓN Y SOLUCIÓN

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

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

Solución EN AWL

```

U      E      0.0
ZV     Z      1
U      E      0.1
ZR     Z      1
U      E      1.0
ZV     Z      2
U      E      1.1
ZR     Z      2
L      Z      1
L      Z      2
<I
SPB    MENO
>I
SPB    MAYO
==I
SPB    IGUA
BEA
    
```

```

MENO: L      W#16#5555
          T      AW      4
          BEA
    
```

```

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

```

IGUA:      L      0
          T      AW      4
    
```

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

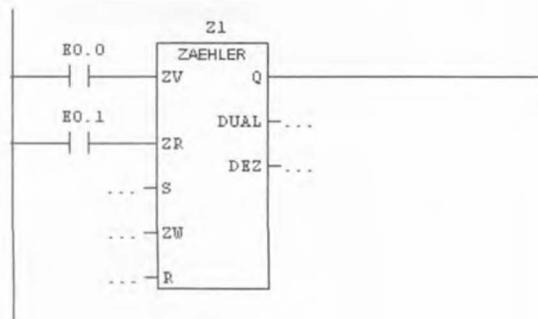
En la última meta podemos dejar sólo el BE ya que es la última instrucción del programa y la función del BE y la del BEA es la misma. También podemos en este caso no poner nada. Como aquí se termina el programa, no es necesario escribir ninguna instrucción de final.

Veamos las soluciones en KOP y en FUP.

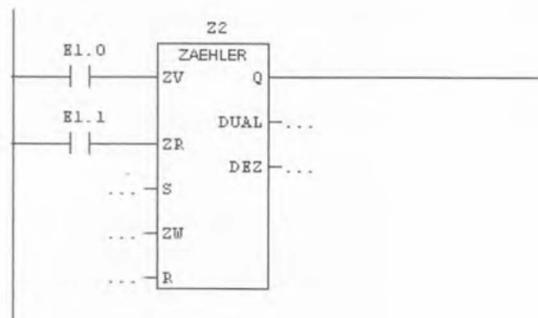
Solución en KOP:

OB1 : "Main Program Sweep (Cycle)"

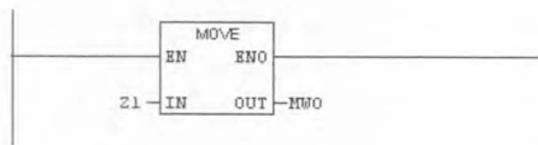
Segm. 1 : Título:



Segm. 2 : Título:



Segm. 3 : Título:



Segm. 4 : Título:

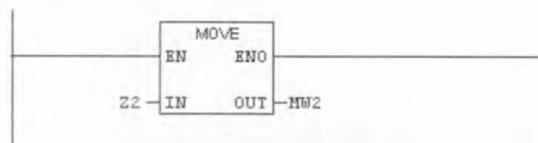
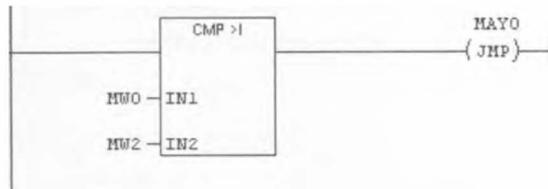
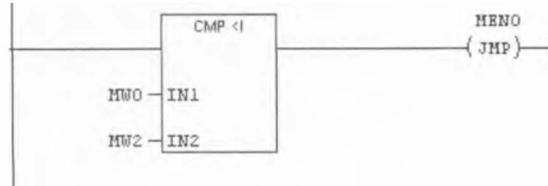


Fig. 82 (parte 1)

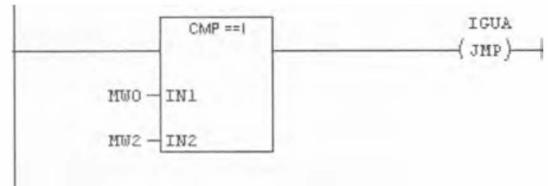
Segm. 5 : Título:



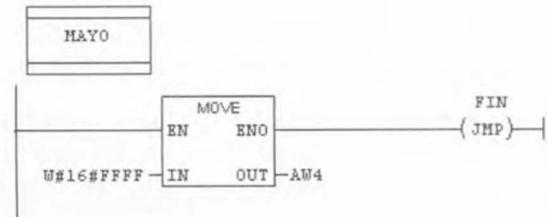
Segm. 6 : Título:



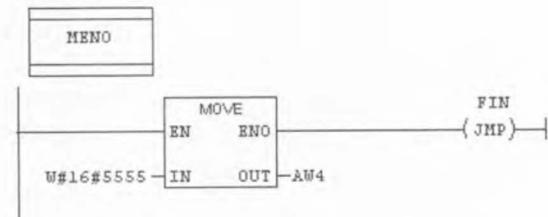
Segm. 7 : Título:



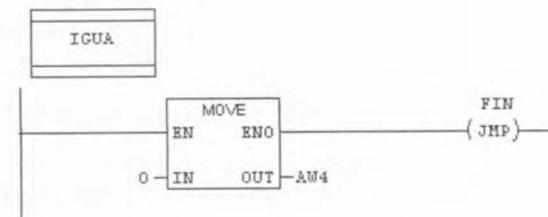
Segm. 8 : Título:



Segm. 9 : Título:



Segm. 10 : Título:



Segm. 11 : Título:



Fig. 82 (parte 2)

Solución en FUP:

La solución en FUP es equivalente a la solución en KOP. Se deja como propuesta resolver el ejercicio en FUP.

Hemos visto la solución en KOP.

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

3.3 Trabajar con bloques de datos

Ejercicio 3: Trabajar con bloques de datos

TEORÍA

CREACIÓN DE UN BLOQUE DE DATOS

Un bloque de datos es una tabla en la que almacenamos datos. Aquí no vamos a programar nada. Tenemos los datos disponibles para poder leerlos desde cualquier punto del programa o para poder escribir sobre ellos nuevos valores. Los datos los podemos almacenar en distintos formatos.

Para guardar un dato, tenemos que poner nombre a la variable, definir el formato en el que lo queremos y el valor inicial. Después tendremos un valor actual que es el que tiene la variable en cada momento. Si nadie escribe nada sobre la variable, el valor actual será siempre el mismo que el valor inicial. En cambio, si se escribe sobre este valor, el dato almacenado en valor actual irá cambiando y no coincidirá con el valor inicial.

El valor inicial siempre es el mismo. Su propio nombre ya lo indica, es el valor inicial. El que tiene cuando se crea el bloque de datos. Es un campo obligatorio de rellenar cuando se crea el DB. Cuando este valor cambie, se almacenará en otra columna que es el valor actual. Aunque al abrir el DB no veamos esta columna, tenemos que tener en cuenta que también existe.

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

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

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

Ejercicio 3: Trabajar con bloques de datos

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Saber crear un bloque de datos.

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

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

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

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

Nos creamos un nuevo proyecto para este segundo capítulo de momento sin *hardware*. A la carpeta de programa le llamaremos "Pesar cajas" y la aprovecharemos para el siguiente ejercicio en el que se utiliza un bloque de datos. En este ejercicio sólo vamos a mostrar cómo se crea el bloque y cómo podemos trabajar con él.

Insertamos un bloque de datos.

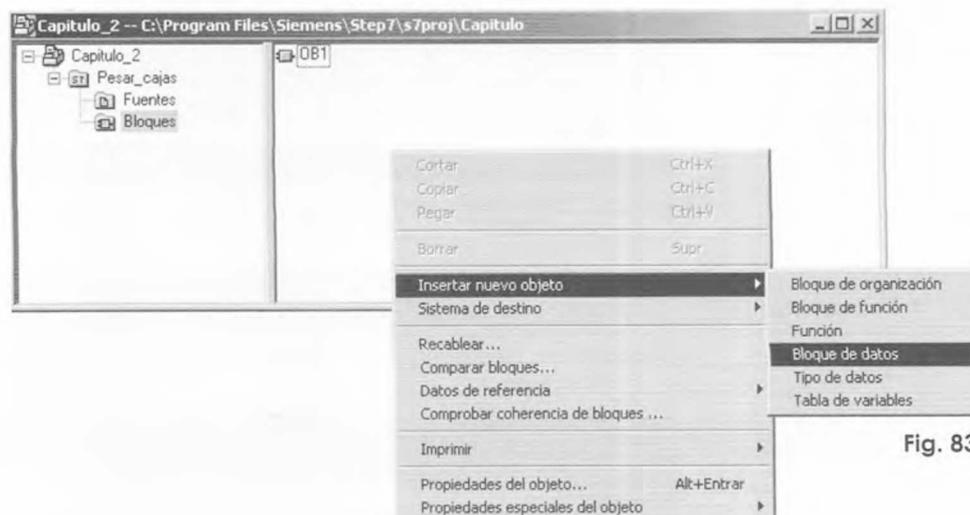


Fig. 83

Al crear el bloque de datos se nos preguntará por el número que le queremos dar y de qué tipo lo queremos generar. El número que le daremos variará entre 1 y el máximo de DB que admita nuestra CPU. A priori todos los DB son iguales. No importa el número que le demos. En el ejemplo hemos creado el DB1. En cuanto al tipo, veremos que desde aquí sólo podemos generar DB de tipo global. En este mismo manual se verán los otros tipos de DB que podemos generar. En el desplegable del lenguaje, vemos que sólo existe DB. Aquí no tenemos lenguaje de programación. Ya hemos dicho que no vamos a programar nada aquí dentro.

Recuerda . . .

En un DB no escribimos programa. Es simplemente una tabla en la que guardamos datos. A los diferentes DB les podremos dar el nombre que queramos.

Veamos cómo queda en el ejemplo el cuadro de diálogo:

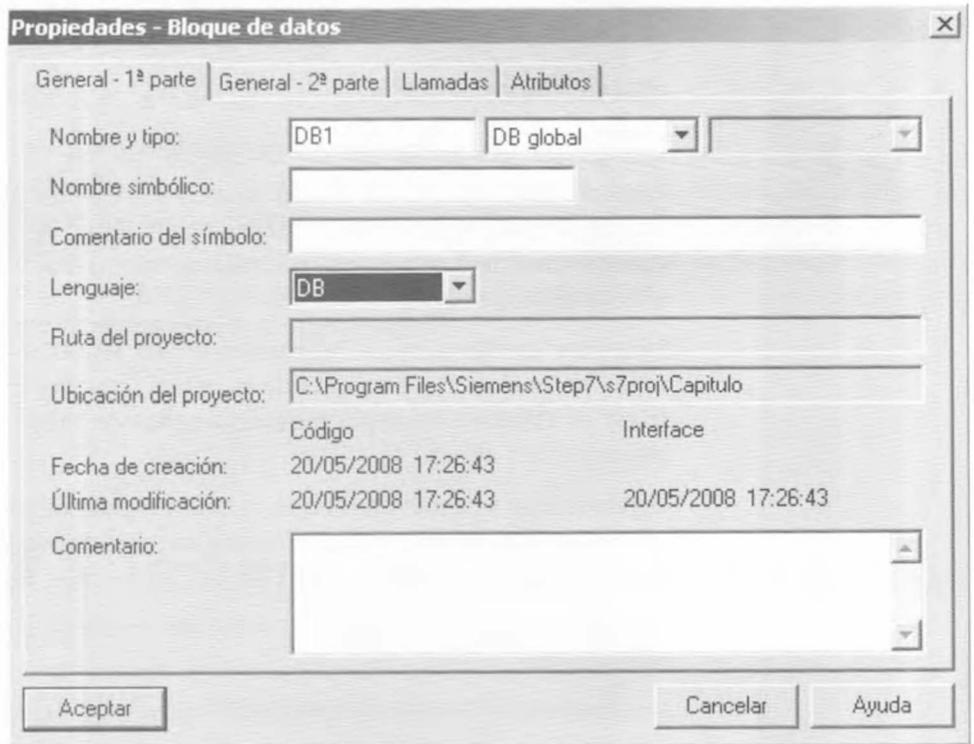


Fig. 84

Una vez lo tenemos creado veremos que sale el icono del DB junto con los demás bloques. En el ejemplo tendremos un OB 1 y un DB1.

Cuando entramos en el nuevo DB por primera vez, vemos que tenemos creada una variable de ejemplo:

Dirección	Nombre	Tipo	Valor inicial	Comentario
0.0		STRUCT		
+0.0	DB_VAR	INT	0	Variable provisional
=2.0		END_STRUCT		

Fig. 85

En principio vamos a borrar esta variable de ejemplo y vamos a definir nosotros las que nos interesen. Para borrarla la seleccionamos desde su dirección y la eliminamos. Nos quedará un DB vacío como el que se muestra a continuación:

Dirección	Nombre	Tipo	Valor inicial	Comentario
0.0		STRUCT		
=0.0		END_STRUCT		

Fig. 86

El primer campo que tenemos es la dirección. De esto no nos tenemos que preocupar. Es automático. Conforme vayamos creando datos, se le van asignando las direcciones que corresponden. En aquello que en la estructura del DB aparece en gris, no tenemos que escribir nada. Rellenaremos las casillas en blanco.

El siguiente campo es el nombre. Es obligatorio dar un nombre a cada dato que vayamos a introducir. En el ejemplo introducimos DATO1.

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

Podemos elegir entre tipos de datos simples y tipos de datos compuestos. Los simples son todos aquellos que ocupan como máximo 32 *bits*. Los datos compuestos son los que ocupan más de 32 *bits*.

En el ejemplo vamos a crear un primer dato de tipo entero.

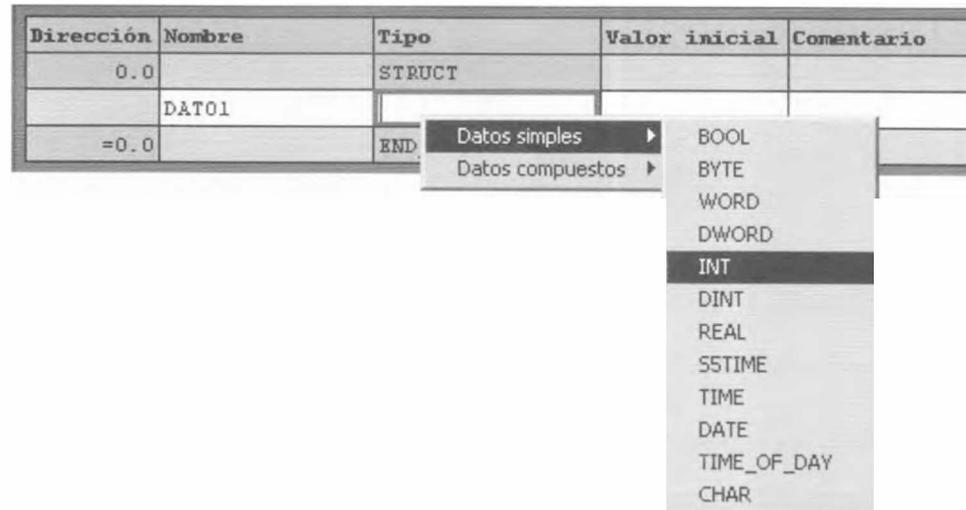


Fig. 87

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

Ya tenemos el primer dato definido.

De la misma manera creamos otros dos datos (DAT02 y DAT03) de formatos real y Word. A DAT02 le daremos valor inicial 3.0.

Obtendremos un DB como el que se muestra en la figura.

Dirección	Nombre	Tipo	Valor inicial	Comentario
0.0		STRUCT		
+0.0	DAT01	INT	0	
+2.0	DAT02	REAL	3.000000e+000	
+6.0	DAT03	WORD	W#16#0	
=8.0		END_STRUCT		

Fig. 88

Ya tenemos un DB. Vamos a ver ahora como trabajamos con él. Para que el PLC pueda acceder a los datos del DB, tendremos que transferirlo a la CPU. Esto lo haremos con el mismo botón que utilizábamos para transferir el OB1 hasta ahora.

Vamos a leer el valor 3.0 que tenemos en DAT02 y vamos a escribir un valor en DAT03.

Para acceder a los datos de un DB utilizaremos las instrucciones de carga y transferencia que vimos en el ejercicio anterior. Para acceder a los *bytes*, palabras o dobles palabras de un DB, tenemos dos posibilidades. Podemos primero “abrir el DB” y luego acceder directamente a la palabra de datos, o podemos decir en cada instrucción a qué DB y a qué palabra de datos queremos acceder. La numeración y disposición de los *bytes* es exactamente igual que vimos con las marcas, entradas o salidas. Todos los DB empiezan por el *byte* 0. Por esto es necesario de una forma u otra decir a qué DB nos estamos refiriendo a la hora de trabajar con los datos. Veamos un par de ejemplos de las dos maneras que tenemos de acceder a los datos del DB:

Si abrimos primero el DB el programa quedaría de la siguiente manera:

```

      AUF  DB  1    //Abre el DB 1
L     DBW  0      //Carga lo que haya en la palabra de datos 0
T     MW   10     //Y envíalo a la palabra de marcas 10

```

Si no abrimos primero el DB, en cada instrucción que hagamos referencia a datos de un DB, tendremos que especificar de que DB. El mismo ejemplo quedaría de la siguiente manera:

```

L     DB1.DBW  0    //Carga la palabra de datos 0 del DB1
T     MW      10   //Y envíala a la palabra de marcas 10

```

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

Hemos visto que los datos dentro del DB tienen nombre. Al crear las variables les llamamos como nosotros queremos. En el ejemplo tenemos DATO1, DATO2 y DATO3. Cuando los hemos creado, no nos hemos preocupado de en qué dirección estaba cada uno. Si nos fijamos, tenemos el DATO1 en la dirección 0, el DATO2 en la dirección 2 y el DATO3 en la dirección 6. El propio **STEP 7** ya sabe lo que ocupa cada tipo de datos y va dejando los *bytes* suficientes para cada dato. Ahora bien, si queremos coger lo que tenemos en DATO3, deberíamos escribir:

```

L     DB1.DBW  6.

```

Aquí si que necesitamos saber en qué dirección tenemos el dato para acceder a él. Imaginemos que tenemos un DB con 1.000 direcciones en las que las variables son *bits*, *bytes*, reales o palabras. Sería muy engorroso trabajar con él y estar pendiente de en qué dirección tenemos cada dato. Para esto tenemos una solución. Si le damos nombre al DB, podremos acceder a las variables por su nombre. Así no tendremos que preocuparnos de las direcciones ni al crear el DB ni al acceder a él. Supongamos que tenemos un DB en el que guardamos 500 temperaturas de un proceso. Y que en un momento dado queremos leer la temperatura 227. Pues sería tan fácil como llamarle al DB TEMPERATURAS. Y a continuación podríamos escribir:

```

L     TEMPERATURAS.TEMPERATURA227

```

En este caso me da igual que sea un entero o un real, que ocupe 2 *bytes* o 4 *bytes*, o que esté en una dirección u otra. Así es más fácil acceder a los datos y también es más fácil interpretar el programa una vez hecho. El día de mañana cuando tengamos que ir a una avería o una modificación, entenderemos mucho mejor el programa con esta simbología que si pone **L DB32.DBD 908**.

Vamos a volver al ejemplo que nos ocupaba. Tenemos que dar nombre a nuestro DB. Para ello vamos a la tabla de símbolos que ya vimos en el capítulo 1. Desde el Administrador de **SIMATIC**, si nos situamos encima del nombre del programa, a la derecha vemos el icono de “**símbolos**”.



Fig. 89

Una vez dentro de la tabla de símbolos damos nombre al DB de la siguiente manera:

	Estado	Símbolo ▲	Dirección	Tipo de dato	Comentario
1		DATOS	DB 1	DB 1	
2					

Fig. 90

DATOS es el nombre del DB y DATO1, DATO2 y DATO3 son los nombres de los datos dentro del bloque.

Vamos a ver como leemos el valor de DATO2 y lo metemos en una marca, y como escribimos un valor en DATO1 y luego lo vemos.

Vamos a leer el valor 3.0 que tenemos en DATO2 y vamos a ponerlo en la doble palabra de marcas 100. Tiene que ser en una doble palabra de marcas porque es un número real y como podemos observar en el direccionamiento del DB, ocupa 4 bytes, o sea, una doble palabra.

Como hemos comentado antes, tenemos dos maneras de hacerlo. Una primera es abriendo previamente el DB y después accediendo a los datos. El ejemplo quedaría así:

```

AUF   DB   1
L     DBD  2
T     MD  100
L     10
T     DBW  0
    
```

Recuerda . . .

Si damos nombre al DB y a los datos guardados en él, podremos acceder a ellos por su símbolo, sin necesidad de saber qué dirección ocupan dentro de la tabla de datos.

Si optamos por esta forma de programar, no podemos utilizar los simbólicos que hemos definido. Veamos cómo quedaría el programa con la otra forma de utilizar los datos de los DB.

```
L   DATOS.DATO2
T   MW   100
L   10
T   DATOS.DATO1
```

De esta segunda forma queda el programa más claro e intuitivo.

Creamos este último programa en el OB1 y lo enviamos al PLC. Previamente tendremos que haber enviado el DB 1 al PLC. Si no lo hemos hecho, el autómeta se nos irá a **STOP**. Le estamos diciendo que acceda al DB a buscar unos datos, que si no tiene, no puede leer y por tanto se va a **STOP**.

Una vez tengamos los dos bloques transferidos, podemos observar lo que ha ocurrido con los valores leídos y escritos.

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

Entramos en el DB y, en cuanto a valores, lo único que vemos es la columna de valores iniciales. Aquí vamos que en DATO1 sigue habiendo un 0.

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

Vemos que el DB tiene una nueva columna que es la de datos actuales. Si observamos los valores, vemos que todavía tenemos un 0 en DATO1. Esto es porque estamos en **OFFLINE**. En el PC, creamos DATO1 con valor 0 y por tanto el valor inicial el 0. Después de hacer el programa, el valor actual sigue siendo 0 porque el PC no ha ejecutado el programa. En principio, mientras nosotros no cambiemos nada, en el PC el valor inicial y el actual será el mismo puesto que no se ejecuta ninguna instrucción.

Si desde esta misma ventana observando la columna de valores actuales, pinchamos el botón que tiene unas gafas (nos conectamos **ONLINE**), veremos que ahora sí que vemos el nuevo valor introducido como valor actual.

Observaremos lo siguiente:

Dirección	Nombre	Tipo	Valor inicial	Valor actual	Comentario
0.0	DATO1	INT	0	10	
2.0	DATO2	REAL	3.000000e+000	3.0	
6.0	DATO3	WORD	W#16#0	W#16#0000	

Fig. 91

Tenemos que tener en cuenta que cuando pulsamos el botón de guardar, estos valores se guardan en disco duro. Cada vez que transfiramos el programa bien de **OFFLINE** a **ONLINE** o viceversa, también se transfieren los valores actuales. Quiere decir que si ahora guardamos este DB en OFLINE, estaremos guardando este 10 como valor actual.

Si queremos dejar el DB como estaba al principio con sus valores iniciales para comenzar el proceso de nuevo, vamos al menú de edición y tenemos la posibilidad de reinicializar bloque de datos. Los valores actuales se cambian por los valores iniciales. A partir de ahí hace lo que se le diga por programa. Esto puede ser interesante para máquinas que parametrizamos a través de un DB. Una vez tengamos la máquina en marcha, guardamos los valores iniciales del primer día. Si posteriormente tenemos una avería en la máquina, inicializando el DB podremos recuperar los datos de la primera puesta en marcha. La opción de inicializar la encontramos aquí:

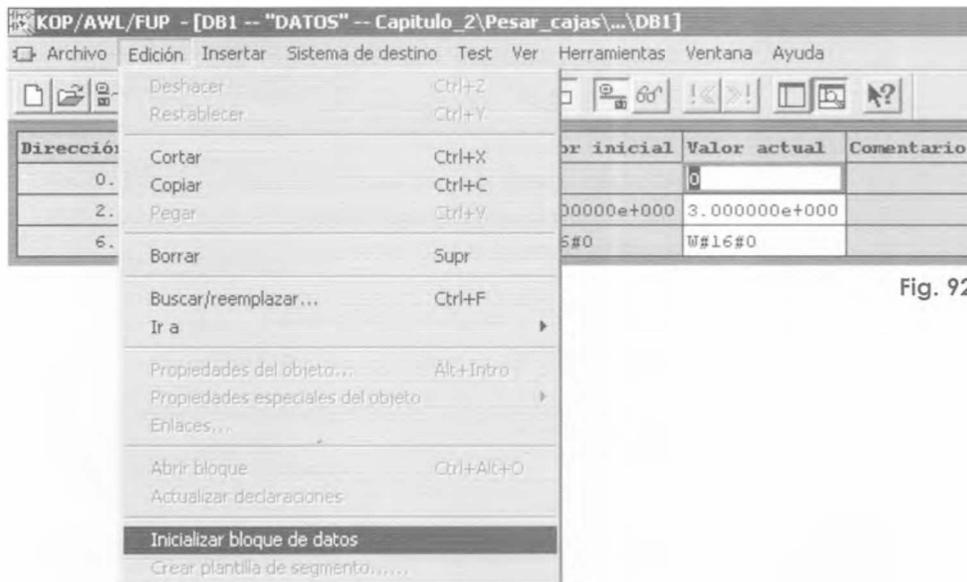


Fig. 92

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

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

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

Se deja como ejercicio repetir esto mismo en KOP o en FUP. En estos lenguajes los DB son exactamente igual. Los bloques de datos no tienen lenguajes de programación. Luego para mover los datos utilizaremos la instrucción MOVE que se vio en ejercicios anteriores.

3.4 Pesar productos dentro de unos límites

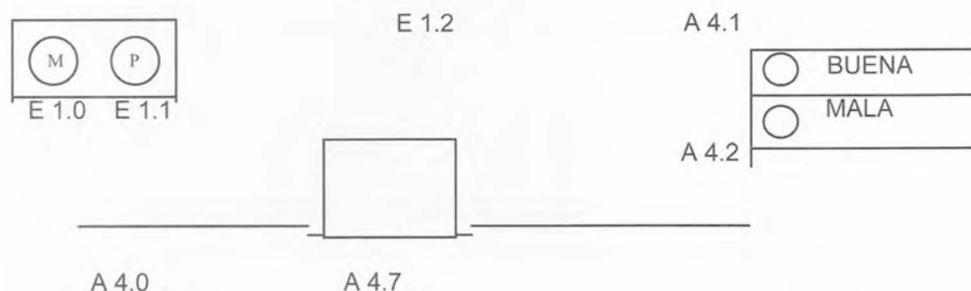
Ejercicio 4: Pesar productos dentro de unos límites

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Creación y manejo de DB.

Tenemos una cinta transportadora que lleva cajas con producto. Lo que queremos es que el peso de los productos esté dentro de unos límites. Las cajas que se salgan del peso estipulado tanto por arriba como por abajo, se deberán desechar.

Además tenemos unas luces indicadoras de caja buena o de caja mala. Queremos que mientras está la caja debajo de la célula detectora, se encienda la luz correspondiente de buena o mala según su peso.



Con los botones de marcha / paro, ponemos en marcha la cinta que simularemos con la salida A4.0. Una vez la caja llegue debajo de la fotocélula de presencia (E1.2), tenemos que hacer la comprobación del peso. Para ello suponemos que tenemos una báscula que nos dará un peso. Como de momento esto no lo podemos simular (no tenemos una báscula que pese), simularemos el peso en la palabra de marcas 10. Aquí forzaremos los valores que queramos para hacer pruebas. Si el peso es correcto, la caja continuará su camino y encenderemos la luz de buena. Si el peso se sale del rango, abriremos la trampilla que desecha las cajas (A 4.7) y encenderemos la luz de mala.

El peso mínimo que deben tener las cajas es de 10 Kg. y el peso máximo es de 15 Kg.

Para ello vamos a tener en un módulo de datos los límites de peso tanto superior como inferior. Podríamos programar todo esto en un OB1 sin necesidad de DB. Podríamos comparar el peso de la báscula con 10 y después con 15 y obtener el resultado. Imaginemos que estamos ante un programa real y tenemos 2.000 instrucciones de programa entre el OB1 y las FC programadas. Y ahora cambiamos el proceso de fabricación y los pesos mínimo y máximo pasan a ser 12 y 18. Deberíamos aprendernos todo el programa y donde ponía 10 poner 12 y donde ponía 15 poner 18. Esto puede suponer mucha pérdida de tiempo. En cambio si lo tenemos en un DB como "PESO_MÍNIMO" y "PESO_MÁXIMO", sólo tenemos que cambiar los datos en el DB y automáticamente habrá cambiado todo el programa. Es mucho más rápido y ordenado que cambiar cada dato por separado. También tenemos menos riesgo de equivocarnos.

Veamos cómo quedaría el programa resuelto. Primero generamos un DB con los pesos mínimo y máximo.

Dirección	Nombre	Tipo	Valor inicial	Comentario
0.0		STRUCT		
+0.0	PESO_MINIMO	INT	10	
+2.0	PESO_MAXIMO	INT	15	
=4.0		END_STRUCT		

Fig. 93

En el simbólico general le damos nombre a este DB y le llamamos PESOS. También podemos dar nombre al resto de entradas y salidas que utilizaremos en el programa. A la palabra de marcas 10 le llamamos báscula. Luego veremos cómo podemos forzar aquí valores.

Una vez generado este DB lo guardamos en el PC y lo enviamos al PLC para poderlo utilizar posteriormente.

En el OB1 hacemos el siguiente programa:

SOLUCIÓN AWL

```

U   PULSADOR_MARCHA //Si pulsamos marcha
S   CINTA             //Arranca la cinta
U   PULSADOR_PARO   //Si pulsamos paro
R   CINTA             //Para la cinta
U   DETECTA_CAJA    //Cuando detecte caja
SPB PESA             //Ve a proceso pesado
R   LUZ_BUENA       //Si no hay caja
R   LUZ_MAL         //Apaga las luces
R   TRAMPILLA       //Y cierra la trampilla
BEA
PESA: L   BASCULA           //Mira el peso real
      L   PESOS.PESO_MAXIMO //Carga peso máximo
      <I           //Si es menor
      =   M   0.0           //Puede que sea buena
      L   BASCULA           //Carga peso real
      L   PESOS.PESO_MINIMO //Carga peso mínimo
      >I           //Si es mayor
      =   M   0.1           //Puede que sea buena
      U   M   0.0           //Si cumple las dos condiciones
      U   M   0.1           //De posibilidad de buena
      S   LUZ_BUENA       //Es buena y encendemos luz
      R   LUZ_MALA       //Apagamos la luz de mala
      NOT           //Si no es buena
      S   LUZ_MALA       //Es mala y encendemos luz de mala
      R   LUZ_BUENA       //Apagamos la luz de buena
      U   LUZ_MALA       //Si está encendida la luz de mala
      =   TRAMPILLA       //Abrimos la trampilla para desechar
    
```

La instrucción **NOT** utilizada en el programa cambia el estado del RLO. Cuando se cumplen las condiciones que escribimos, el RLO estará a 1. Si escribimos un **NOT** se pondrá a cero. Viene a ser como expresar: Si se cumplen las condiciones haz una cosa y si no haz otra.

Para probar el ejercicio ahora tenemos que forzar valores en la báscula. Para ello abrimos una tabla de observar / forzar variables como la que ya habíamos visto en ejercicios anteriores. Allí forzaremos el valor de la báscula. Si queremos también podemos observar valores de otras variables como las luces de buena y mala, la trampilla o el motor de la cinta. Veamos como quedaría la tabla y como forzamos los valores del peso.

Los formatos de visualización los elegimos nosotros según lo que nos convenga. Para las cosas que son *bits*, elegiremos formato *BOOL*. Para el peso, en este caso nos interesa verlos como entero. Para ver los valores que tiene cada variable, pulsamos el botón que simula unas gafas sobre la tabla de variables.

Para forzar un valor de paso, por ejemplo 14 Kg, escribimos 14 en la columna de valor de forzado. Para que la variable tome este valor, deberemos pulsar el botón que simula un rayo junto al de las gafas que tenemos pulsado. Forzando diferentes valores de peso, podremos probar que el programa que hemos hecho funciona correctamente.

	Operando	Símbolo	Formato de v	Valor de estado	Valor de forzado
1	A 4.0	"CINTA"	BOOL	false	
2	A 4.1	"LUZ_BUENA"	BOOL	false	
3	A 4.2	"LUZ_MALA"	BOOL	false	
4	A 4.7	"TRAMPILLA"	BOOL	false	
5	MV 10	"PESO"	DEC	14	14
6					

Fig. 94

3.5 Programación estructurada

Ejercicio 5: Programación estructurada

TEORÍA

EJEMPLO DE PROGRAMACIÓN ESTRUCTURADA. PROGRAMACIÓN FC

Hasta ahora hemos programado todos los ejemplos en el OB1. A partir de ahora vamos a generar funciones y hacer el programa en estas funciones en lugar de en el OB1. Aquí encontraremos todo el potencial del **STEP 7**. El OB1 es un bloque de organización y así lo deberemos utilizar. Para organizar los demás bloques. Para desarrollar un proyecto en **STEP 7**, normalmente deberemos programar las distintas funcionalidades en diferentes FC o FB. Después desde el OB1 indicaremos bajo que condiciones tienen que ejecutarse y haremos las llamadas correspondientes. Es decir organizaremos desde el OB1 el resto de programa.

Veamos un ejemplo muy sencillo de cómo creamos las FC, cómo las programamos y cómo podemos llamarlas desde el OB1.

Vamos a generar dos funciones. Para ello en el mismo proyecto que teníamos abierto para los ejercicios de los DB, nos creamos una nueva carpeta de programa que se llame "Funciones".

En la carpeta de programa tendremos el OB1 que viene por defecto. Nosotros vamos a crear dos FC. Dentro de ellas haremos un pequeño programa y desde el OB1 diremos cuando queremos que se ejecute.

Para crear las funciones, junto al OB1 pulsamos con el botón derecho del ratón y seleccionamos "Insertar nuevo objeto → Función".

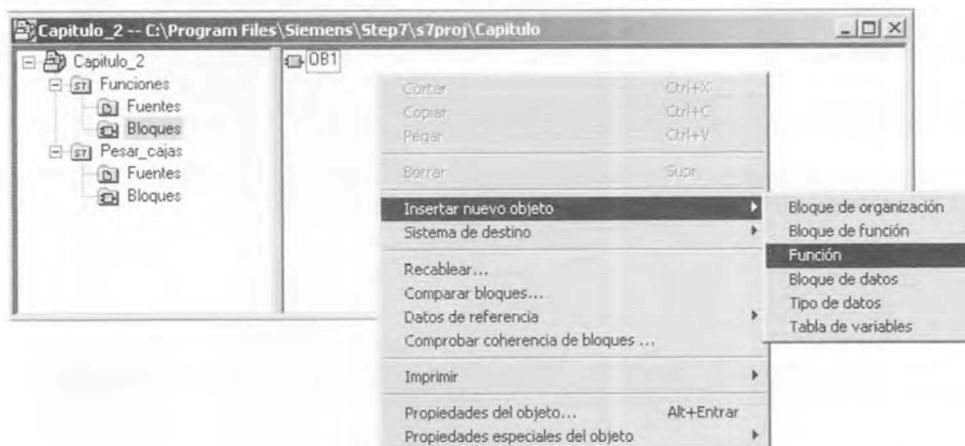


Fig. 95

Recuerda . . .

Normalmente la funcionalidad de los programas hechos en Step 7 se programa en FC y en FB. El bloque OB1 se utilizará para ejecutar estas funciones en el orden y la secuencia correctos.

Una vez le decimos que queremos crear una función, nos sale un diálogo similar al que rellenamos al entrar la primera vez en el OB1.

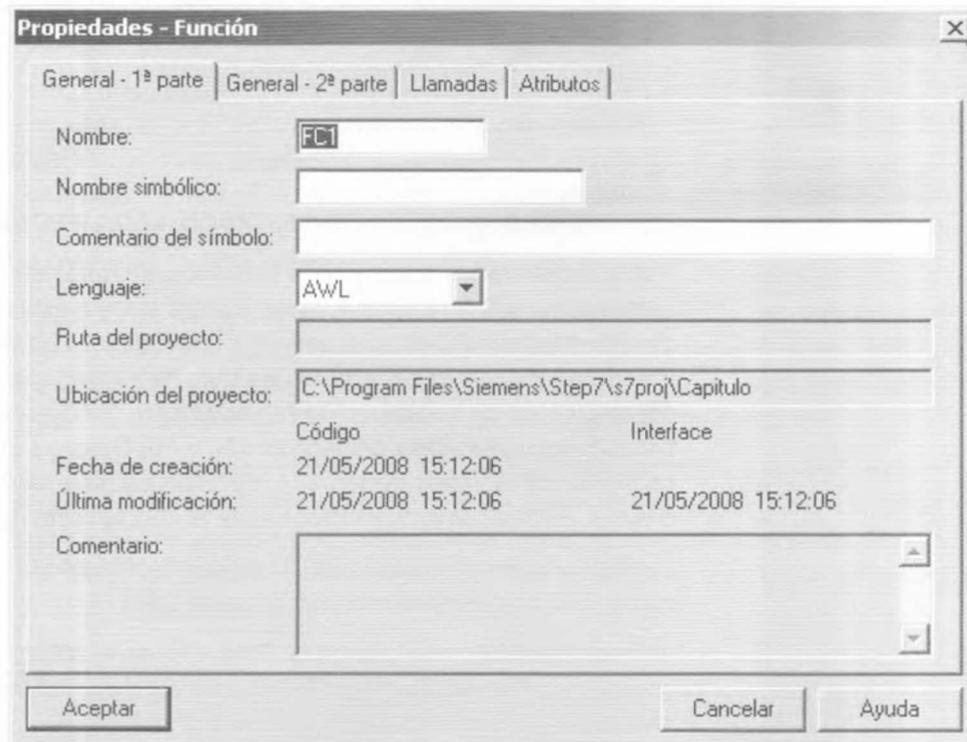


Fig. 96

Aquí tenemos que decir la FC que queremos generar. Podemos crear desde la FC 0, hasta el máximo de funciones permitida por la CPU.

También desde aquí podemos ponerle un nombre simbólico. En la pestaña de lenguaje podremos seleccionar AWL, KOP o FUP. Ocurre lo mismo que con el OB1. Aquí seleccionamos un lenguaje, pero luego lo podemos cambiar desde el propio bloque.

Aceptamos y ya tenemos generada la primera función. Del mismo modo nos creamos una FC2. Ya tenemos tres bloques para programar.

Entramos en la FC1 y generamos el siguiente programa:

FC1

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

Entramos en la FC2 y generamos el siguiente programa:

FC2

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

Ya tenemos dos funciones creadas. Tenemos que guardarlas en el ordenador y transferirlas a la CPU.

Para poder llamar a las funciones sin parámetros como éstas, disponemos de dos instrucciones:

CC: Llamada condicional

UC: Llamada incondicional.

Recuerda . . .

Desde el OB1 se hacen las llamadas a las funciones programadas de forma condicional o incondicional.

Vamos a llamar a la FC1 de manera condicional y a la FC2 de manera incondicional.

SOLUCIÓN AWL

OB1

```

U      E      0.0
CC     FC      1           //Llamada condicional.
UC     FC      2           //Llamada incondicional.
    
```

Lo que conseguimos con este programa es que mientras no esté activa la E0.0, el PLC no leerá la FC1. Con lo cual no funcionará lo que tengamos allí dentro programado. En cambio la FC2 funcionará siempre.

La ventaja de programar en FC en lugar de todo en el OB1 es que tenemos el programa mejor organizado. Aunque hagamos llamadas incondicionales y se ejecuten siempre las FC, a la hora de detectar un error de programa por ejemplo, es más fácil de localizarlo. Sólo miramos la FC que ejecuta la función que falla. No tenemos por qué leer el resto de programa.

Vamos a transferir este OB1 al PLC y vamos a analizar algunas cosas. Mientras no esté activa la E0.0 no funcionará la FC1. Quiere decir que si activamos la E1.0 no pasará nada.

Ahora vamos a activar la E0.0. Con lo cual ya funciona la FC1. Si también activamos la E1.0, se encenderá la A4.0. Si en esta situación quitamos la E0.0, la A4.0 quedará encendida. Aunque quitemos la E1.0 seguirá encendida. El que no funcione una FC no implica que desactive sus salidas. Simplemente el PLC no está leyendo aquel trozo de programa. Si las salidas o variables que actúa esta FC estaban activas cuando dejó de funcionar, pues así se quedarán hasta que volvamos a activar la E0.0 y vuelva a funcionar.

Veamos como podríamos hacer esto en KOP y en FUP.

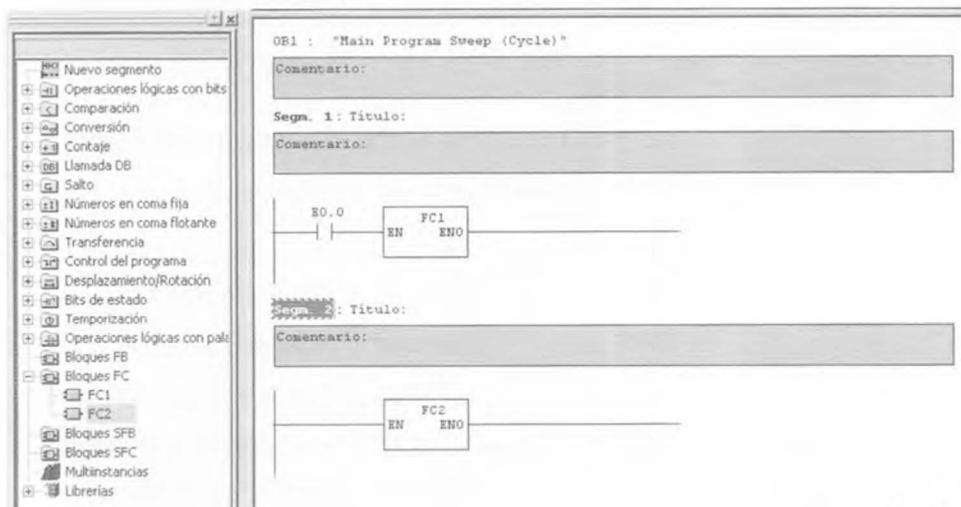


Fig. 97

En el catálogo de la programación en KOP, tenemos un desplegable que se llama "Bloques FC". Allí dentro tendremos todas las funciones que tengamos creadas y guardadas en el PC dispuestas para llamarlas. En el ejemplo, vemos que tenemos la FC1 y la FC2.

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

El movimiento que en el papel vemos de derecha a izquierda, no coincide con el movimiento de arriba hacia abajo de la palabra de salidas. Según la disposición de los *bits* que vimos a principio de este capítulo, ir de la A4.0 a la A4.7, es desplazar hacia la izquierda. E ir de la A4.7 a la A4.0 es desplazar hacia la derecha.

Vamos a aprovechar el concepto de programación estructurada. Vamos a hacer dos FC. Una que desplace a derechas y otra que desplace a izquierdas. Después desde el OB1 diremos cuando se tiene que ejecutar una y cuando se tiene que ejecutar la otra.

FC 1: Desplazamiento a derechas

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

FC 2: Desplazamiento a izquierdas

```
L    AW    4
SLW  1
T    AW    4
```

Ya tenemos dos funciones que desplazan *bits*. Ahora desde el OB1 tendremos que llamarlas cuando corresponda. Veamos como quedaría la programación en AWL del OB1.

OB1

```
U    E    0.0    //Cuando le demos a la entrada 0.0
S    A    4.0    //Encenderemos la salida 4.0

UN   M    0.0    //Hacemos unos pulsos de 500 milisegundos
L    S5T#500MS //De la marca 0.0
SE   T    1
U    T    1
=    M    0.0

U    M    0.0    //Cuando llegue un pulso de la marca 0.0
UN   M    0.1    //Y no esté la marca 0.1
CC   FC    1     //Se irá a ejecutar la FC 1
U    A    4.7    //Cuando se encienda la salida 4.7
S    M    0.1    //Activa la marca 0.1
U    A    4.0    //Si está encendida la salida 4.0
R    M    0.1    //Desactiva la marca 0.1

U    M    0.0    //Si llega un pulso de la marca 0.0
U    M    0.1    //Y está la marca 0.1
CC   FC    2     //Ejecuta la FC 2
```

Veamos cómo se programan las instrucciones de desplazamiento en KOP y en FUP:

Solución en KOP

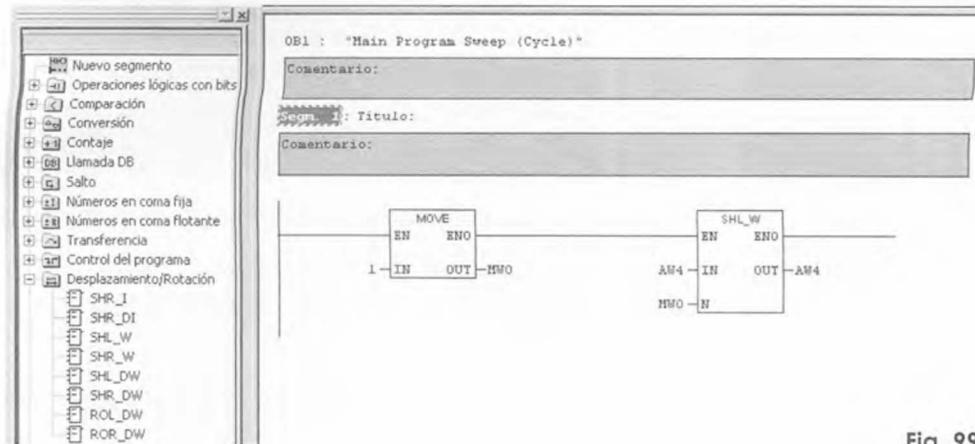


Fig. 99

Las instrucciones de rotación y desplazamiento las tenemos en la carpeta: “**Desplazamiento / Rotación**”. Las instrucciones son equivalentes a las de AWL. Siempre como parámetro EN en KOP podemos poner una condición para que se ejecute. En el ejemplo el parámetro IN es la palabra que queremos desplazar. El parámetro OUT es donde se almacenará la palabra desplazada. En el parámetro N deberemos escribir la cantidad de posiciones que queremos desplazar. Por definición en el parámetro N se espera una palabra y no un entero. Si nos situamos encima con el ratón veremos qué tipo de dato se espera. Nosotros queremos desplazar la palabra una posición pero no podemos escribir directamente un 1 porque es un entero y no una palabra. Tendremos previamente que poner un 1 en una palabra y después utilizar esta palabra como parámetro. Como ya dijimos al principio de este manual, el AWL es más versátil y ofrece más posibilidades al programador.

El programa en FUP sería equivalente al segmento en KOP.

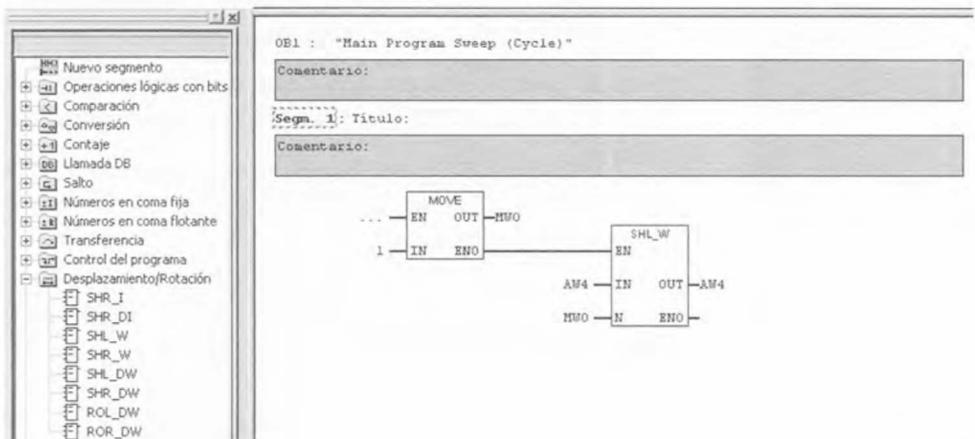


Fig. 100

3.7 Planta de embotellado

Ejercicio 7: Planta de embotellado

DEFINICIÓN Y SOLUCIÓN

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

Tenemos una planta de embotellado distribuida de la siguiente manera:

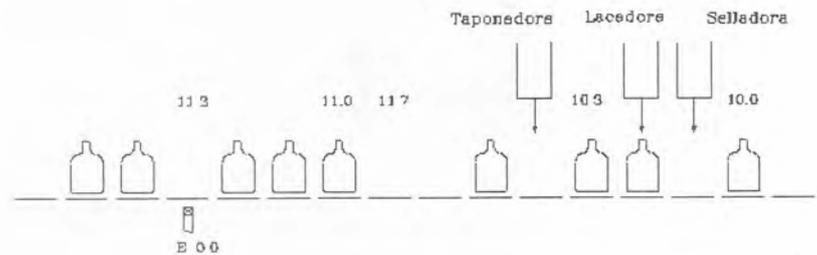


Fig. 101

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

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

Las botellas pasan de posición a posición cada segundo. Con la célula fotoeléctrica que tenemos detectamos cuando pasa una botella o cuando pasa un hueco unas cuantas posiciones antes de las máquinas. Con los datos de una sola fotocélula actuaremos las tres máquinas.

Vamos a resolver el problema utilizando la programación estructurada. Aquí veremos bien para que nos puede servir separar las cosas en FC. Vamos a crear 4 funciones y un OB1 para llamarlas.

En cada FC vamos a programar una de las operaciones que necesitamos realizar. Después desde el OB1 diremos cuando necesitamos realizar cada una de las operaciones.

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

FC 1: Generador de pulsos

```

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

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

Para simular las posiciones por donde circulan las botellas vamos a tomar una palabra de marcas en la que cada bit simulará una posición:

10.7	10.6	10.5	10.4	10.3	10.2	10.1	10.0	10.7	10.6	10.5	10.4	10.3	10.2	10.1	10.0
------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------

MW 10

La fotocélula la tenemos en la posición del bit M 10.5. Las máquinas se encuentran en las posiciones de los bits M11.6, M11.4 y M11.3.

FC 2: Poner botellas (*bit a 1*)

```
U      E      0.0
S      M      10.5
```

En principio ponemos una marca a 1 con un **SET**. No queremos que este 1 se vaya porque la fotocélula parpadee o se mueva la botella. Ponem207 word autoraos un uno fijo, y después otra FC ya moverá este bit a la siguiente posición.

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

FC 3: Desplazamiento a derechas

```
L      MW      10
SLW    1
T      MW      10
```

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

FC 4: Activación de máquinas

```
U      M      11.6
=      A      4.0
U      M      11.4
=      A      4.1
U      M      11.3
=      A      4.2
```

Como hemos podido comprobar, la programación de cada FC es muy sencilla. Programamos las funciones sin preocuparnos de cuando se tienen que ejecutar. Es ahora desde el OB1 cuando nos encargaremos de esta tarea.

OB 1: Bloque de organización

```

UC   FC   1   //Siempre estamos generando pulsos
UC   FC   2   //Siempre miramos si llega botella para colocar el 1
U    M    0.0 //Sólo cada segundo
CC   FC   3   //Hacemos el desplazamiento de bits
UC   FC   4   //Siempre miramos si hay botella para activar las máquinas
    
```

Imaginemos que ahora enviamos todo esto al PLC y no funciona bien. Hay una máquina que no se activa. Entonces sólo tendremos que comprobar la FC4. Sabemos que el resto del programa está bien. Es mucho más sencillo de localizar errores en la programación estructurada.

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

3.8 Diferencia entre FC con y sin parámetros

Ejercicio 8: Diferencia entre FC con y sin parámetros 

TEORÍA

LLAMADAS A LAS FC CON PARÁMETROS. OPERACIONES ARITMÉTICAS

Cuando programamos una FC tenemos que llamarla desde algún sitio para que se ejecute. Ya hemos visto como crear una FC sin parámetros y hemos visto dos instrucciones para llamarlas. En este ejercicio vamos a ver que también podemos crear FC con parámetros y las instrucciones de llamada serán diferentes.

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

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

Como ya hemos visto, para llamar a esta FC podemos utilizar dos instrucciones:

```

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

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

Cuando estamos sumando $A + B$, estamos haciendo una FC parametrizable. Cada vez que la llamemos, tendremos que darle unos valores a A y a B para que el autómata sepa lo que tiene que sumar. El resultado variará dependiendo de lo que valgan A y B. La función nos devolverá también una variable.

Vamos a ver como creamos una FC con parámetros. En principio nos creamos una carpeta de programa como hemos hecho hasta ahora y creamos una FC1. Una vez entremos en la FC tenemos que abrir la tabla de variables que está arriba de la FC. Entre la barra de herramientas y la zona de programación. Por defecto esta tabla viene cerrada. Tendremos que situarnos con el ratón en la línea que divide la barra de herramientas de la zona de programación y abrir la tabla de variables.

En principio veremos la tabla vacía con este aspecto:

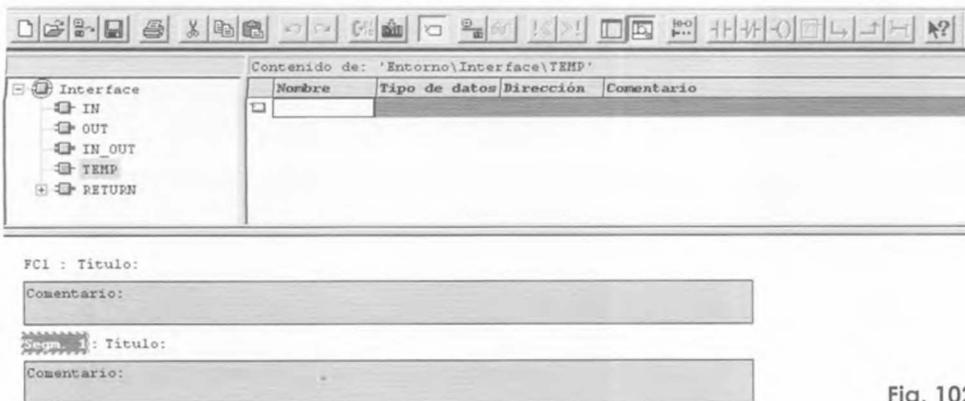


Fig. 102

En el ejemplo que hemos propuesto, necesitamos dos variables de entrada A y B y una variable de salida SUMA. Las variables de entrada las definimos en el grupo de IN y la de salida en el grupo de OUT.

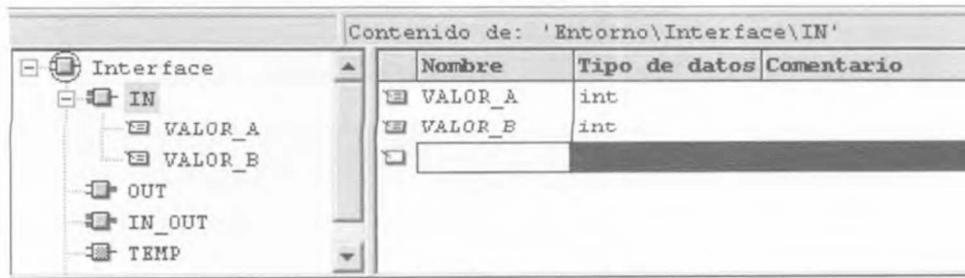


Fig. 103

A las variables no las podemos llamar A y B porque A son las salidas del autómata. No podemos utilizar simbología reservada por el sistema. Por eso hemos llamado a las variables VALOR_A y VALOR_B.

Lo mismo tenemos que hacer con la variable de salida que llamaremos SUMA.

Con estas variables definidas en la tabla, podemos hacer programar la función de la siguiente manera:

FC 1: Suma de dos variables

```
L   VALOR_A
L   VALOR_B
+I
T   SUMA
```

Recuerda . . .

Al realizar operaciones matemáticas, siempre lo deberemos hacer entre números con el mismo formato. Si los formatos no coinciden, el PLC hará la operación con los "0" y "1" que encuentre en el acumulador, pero el resultado será incoherente.

Nosotros programaremos esto y veremos que delante de las variables el sistema les pone #. Esto quiere decir que son variables locales. En la FC1 hemos definido VALOR_A y VALOR_B. Estas variables sólo las podremos utilizar dentro de la FC 1 que es donde las hemos definido. Por eso se llaman variables locales.

Ahora vamos a generar la otra función que suma siempre lo mismo.

FC 2: Suma de dos valores

```
L    2
L    3
+I
T    MW    20
```

En ambas funciones hemos utilizado la instrucción "+I".

Significa que sume los dos valores anteriores con formato de enteros. El programa internamente funcionará de la siguiente manera:

```
L    2    Carga el número 2 en el acumulador 1
L    3    Carga el valor 3 en el acumulador 1 y pasa el valor 2 al acu 2
+I      Suma acu 1 + acu 2. El resultado lo deja en el acumulador 1
T    MW    20  Envía lo que tiene en el acu 1 a la palabra de marcas 20
```

Como vemos, el PLC siempre trabaja con el Acu 1. Siempre que carguemos algo va al acumulador 1. Cuando transferimos algo (instrucción T) siempre es lo que tiene en el acu 1. Al acumulador 2 no podemos acceder directamente. Lo utiliza la CPU para hacer operaciones internamente.

Para hacer operaciones de enteros, tenemos las instrucciones:

+I (suma) **-I** (resta) ***I** (multiplicación) **/I** (división)

Para llamar a una función con parámetros definidos en su tabla de variables, utilizaremos la instrucción **CALL**. Esta instrucción es siempre incondicional. Ocurre lo mismo que con las instrucciones L y T. Si queremos hacer una llamada condicional de una FC con parámetros, lo tendremos que hacer con saltos o poniendo la llamada dentro de una FC a la que sólo se vaya cuando se cumplen las condiciones que nos interesa.

Para poder hacer la llamada a estas funciones correctamente, primero las tenemos que tener guardadas en disco duro. Al hacer la llamada con la instrucción **CALL**, el sistema sabe que esa función tiene parámetros. Entonces accede a la tabla de variables que hemos definido y guardado, y nos pregunta su valor. Si no hemos guardado la función con su correspondiente tabla, al llamar a la FC el sistema pensará que tenemos la tabla vacía, es decir, que no tiene valores que pedir. Cuando el PLC intente sumar A + B y no tenga valores de A ni de B, no podrá hacer la suma y se irá a **STOP**.

Nosotros ya tenemos la FC1 y la FC2 programadas, guardadas y transferidas al PLC. Ahora vamos a programar el OB1 con las dos llamadas.

Una vez tenemos una FC con parámetros programada, la podemos llamar tantas veces como queramos cambiando los valores de las variables.

Vamos a generar un OB1 con varias llamadas.

Al llamar a la FC1 con la instrucción **CALL**, el sistema nos pide las variables que hemos definido en la tabla. Si nos acercamos con el ratón a cada una de las variables, nos indica si es una variable de entrada o de salida, y el tipo de datos que espera. En la figura ejemplo vemos que VALOR_A es una variable de entrada y tenemos que introducir un entero. Si intentamos escribir 3,47 por ejemplo, no nos lo aceptará porque en la definición ya dijimos que esta función sumaría enteros.

```
OB1 : "Main Program Sweep (Cycle)"
Comentario:
Segm. 1: Título:
Comentario:
CALL FC 1
  VALOR_A:=
  VALOR_IN: INT
  SUMA :=
```

Fig. 104

Podemos hacer varias llamadas a la misma FC introduciendo diferentes valores en las variables. Hemos generado una sola FC que suma y ahora podemos sumar lo que queramos. Veamos un ejemplo con varias llamadas:

```
OB1 : "Main Program Sweep (Cycle)"
Comentario:
Segm. 1: Título:
Comentario:
CALL FC 1
  VALOR_A:=2
  VALOR_B:=2
  SUMA :=M00

CALL FC 1
  VALOR_A:=7
  VALOR_B:=15
  SUMA :=M02
```

Fig. 105

Hemos llamado a la FC1 tres veces. Hemos programado una suma y ahora estamos haciendo tres sumas. Esta es la ventaja de programar FC parametrizables. En nuestro caso, en el ejemplo, no es apreciable el ahorro de tiempo. La suma son 4 instrucciones. Pero si en lugar de una suma tenemos programado un proceso largo, lo programaríamos una sola vez y luego lo utilizaríamos tantas veces como nos interese simplemente llamando a la función y rellenando sus parámetros.

En los parámetros de entrada tenemos que introducir valores nosotros. Son parámetros de entrada a la función. En cambio los parámetros de salida, los rellenará la propia función. Nosotros tenemos que indicar dónde queremos que los almacene. Tiene que ser un registro en el que quepa el tipo de datos que queremos almacenar.

Por ejemplo en la primera llamada tenemos como valores de entrada 2 y 2. La función nos los sumará y obtendrá un 4. Lo dejará en la palabra de marcas 0.

Veamos un ejemplo de cómo podemos hacer operaciones matemáticas en KOP y en FUP.

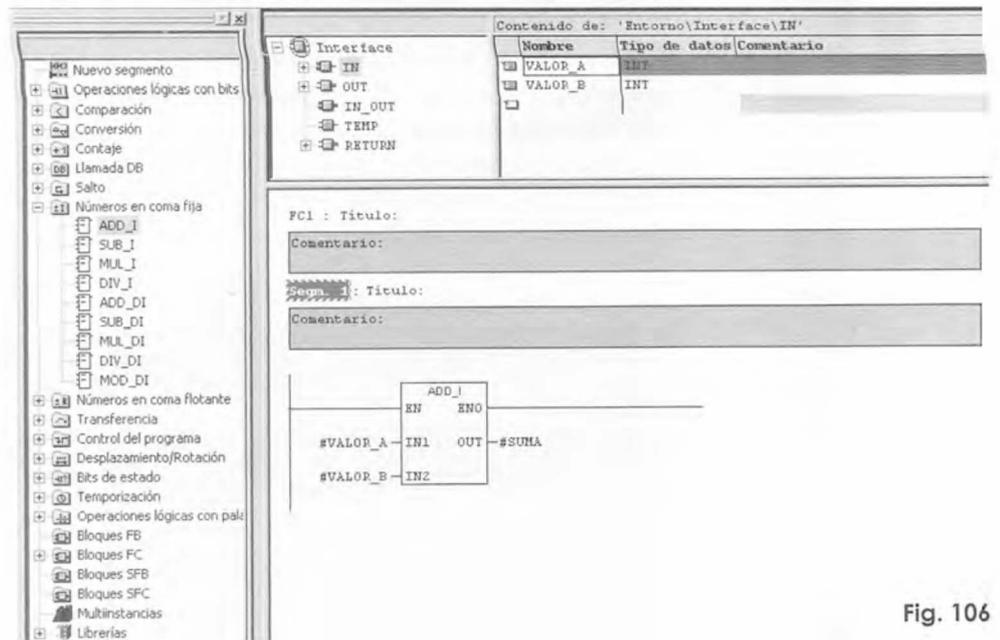


Fig. 106

Las variables las definimos igual que en AWL en la tabla superior de la FC. Para hacer operaciones con enteros, las encontramos en la carpeta de Números en coma fija. Rellenamos los parámetros con las variables definidas.

Para llamarla desde el OB1 lo hacemos igual que hacíamos con las FC sin parámetros. Encontraremos las FC en la misma carpeta. Si tiene parámetros veremos que nos los pide igual que en AWL. Veamos un ejemplo:

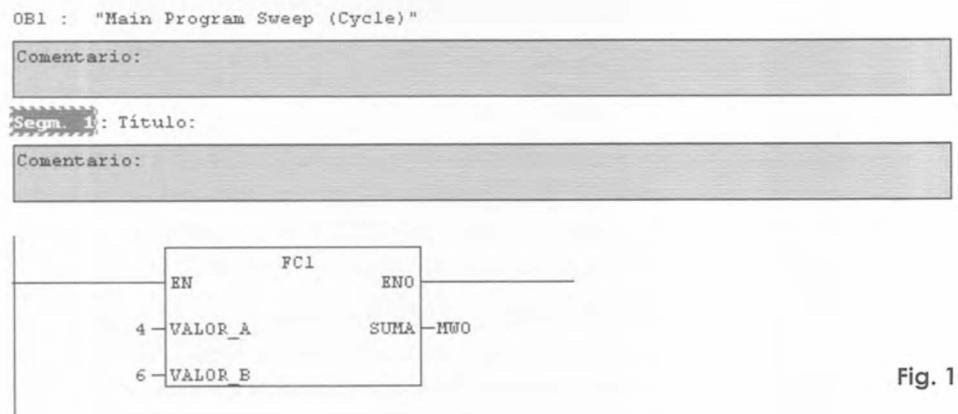


Fig. 107

Ejercicio 8: Diferencia entre fc con y sin parámetros



DEFINICIÓN Y SOLUCIÓN

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

Se deja como ejercicio propuesto crear varias FC que realicen diferentes operaciones matemáticas. Hay que tener en cuenta que la CPU sólo dispone de 2 acumuladores. No puede sumar más de dos valores a la vez. Imaginemos que queremos sumar $2+7+8+4+1$. Imaginemos que hacemos el siguiente programa:

```
L    2
L    7
L    8
L    4
L    1
+I
T    MW    0
```

Esto no sumaría lo que nosotros queremos. El PLC sólo dispone de dos acumuladores. Nosotros siempre cargamos en el 1 y lo que hay en el 1 pasa al 2. En este momento lo que hubiese en el 2 se pierde. En este programa que hemos puesto de ejemplo, los acumuladores quedarían con 1 y 4. Los demás valores se habrían perdido. El resultado de la suma sería 5. Si realmente queremos hacer la suma de todos los valores, habría que hacerlo de la siguiente manera:

```
L    2
L    7
+I           //Suma 7 + 2
L    8           //Carga 8
+I           //Suma el resultado de 7 + 2 + 8
L    4           //Carga 4
+I           //Suma el resultado de 7+2+8 + 4
L    1           //Carga 1
+I           //Suma el resultado de 7+2+8+4 +1
T    MW    0    //Obtenemos 7+2+8+4+1
```

3.9 Crear un DB con la SFC 22

Ejercicio 9: Crear un DB con la SFC 22

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Creación de un DB.

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

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

Para hacerlo disponemos de una SFC que ya lleva integrada la CPU. Las SFC son funciones como las que acabamos de crear en el ejercicio anterior, pero que ya vienen programadas en la CPU. Se llaman funciones de sistema. Dentro de la CPU están siempre. No las podemos borrar. Si queremos tenerla además **OFFLINE** en el proyecto, podemos abrir la ventana de **ONLINE** y **OFFLINE** y arrastrarla con el ratón. Para utilizar una SFC simplemente tenemos que llamarla desde algún punto del programa. No podremos entrar en ella para ver el código programado. Están protegidas.

Todas las SFC del sistema, tienen una ayuda en la que se nos explica lo que hace la función, qué parámetros tiene definidos y cómo y con qué formato tenemos que rellenar cada uno de ellos.

Si vamos a la ventana de **ONLINE** desde el Administrador de **SIMATIC**, veremos todas las SFC y SFB que tiene nuestra CPU. La cantidad de funciones de sistema, depende del modelo de CPU que estemos gastando. La SFC 22 existe en todos los modelos de CPU 300.

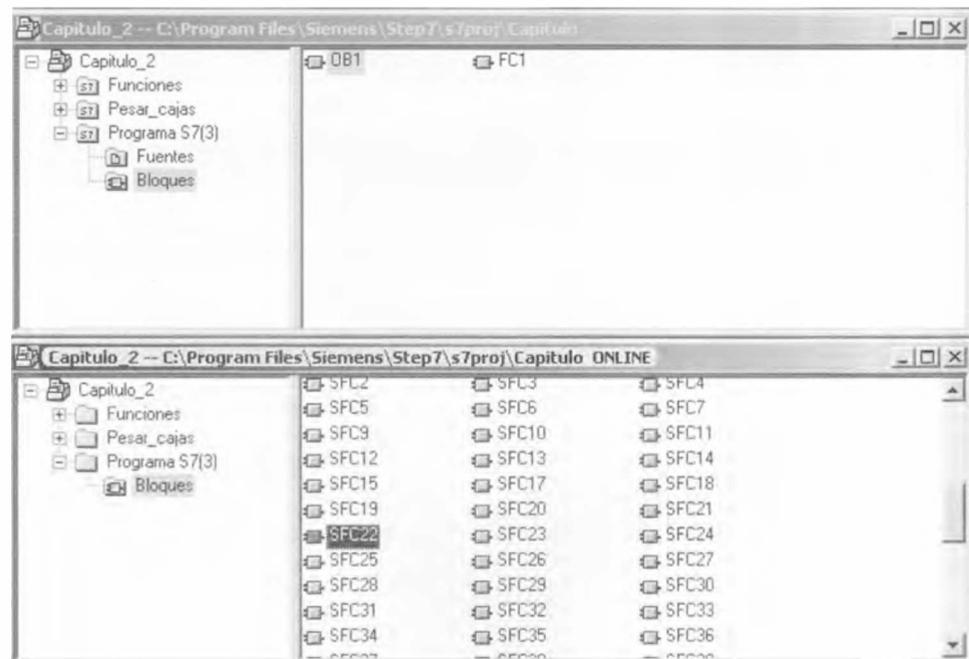


Fig. 108

Recuerda . . .

Si hacemos clic sobre cualquier icono dentro del Administrador de Simatic o sobre los bloques de sistema o de librerías con el interrogante de la barra de herramientas "¿", obtenemos al instante la ayuda de aquello que necesitamos.

Para ver lo que hace esta función y qué parámetros nos va a pedir, pinchamos sobre ella habiendo seleccionado antes el interrogante que tenemos en los botones de la barra de herramientas. Obtendremos la siguiente ayuda:

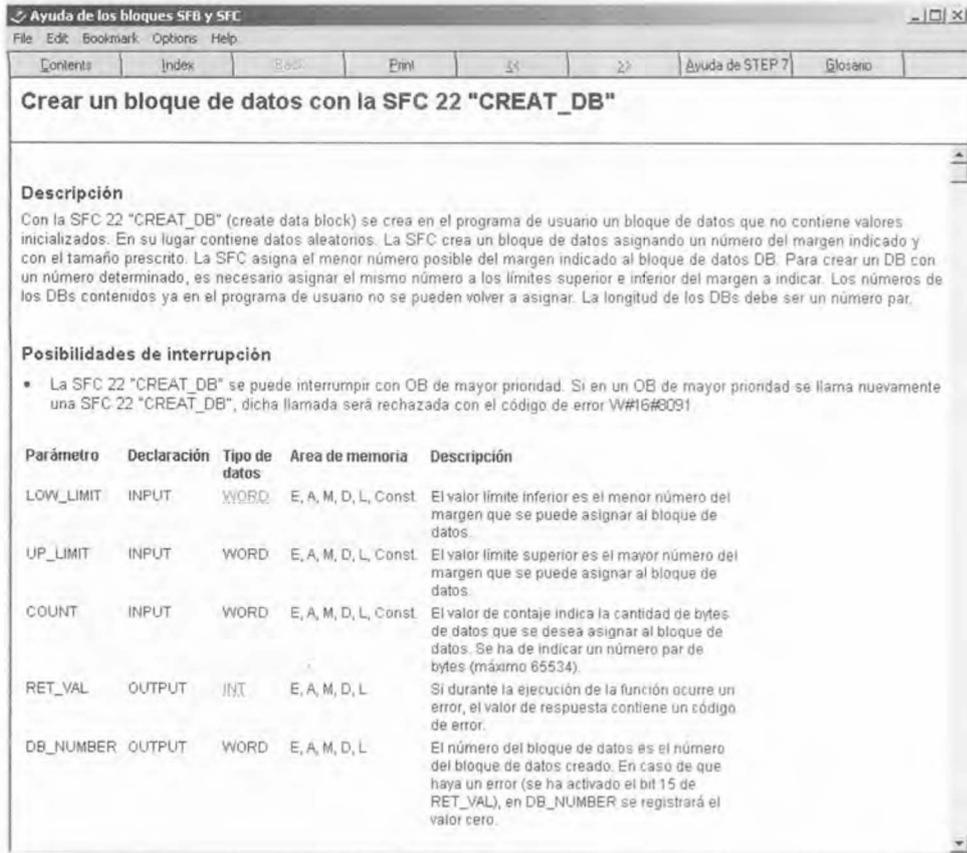


Fig. 109

En esta ayuda nos explica lo que hace la SFC 22 y nos dice que al llamarla se nos va a pedir una serie de parámetros. Nos explica lo que significa cada parámetro. Como valor de salida, nos devuelve un parámetro llamado RET_VAL. Si durante la creación del DB se produce algún error, aquí se almacenará un código que podemos consultar también en esta misma ayuda.

Veamos como haríamos una llamada a esta SFC y veamos como rellenaríamos los parámetros. Nosotros escribiríamos `CALL SFC 22`. Si previamente hemos trasladado la SFC 22 de **ONLINE** a **OFFLINE**, en la carpeta de símbolos se habrá guardado el nombre de la función. Todas las funciones de sistema tienen nombre. Al llamarla, la veremos por su nombre. Veamos un ejemplo en AWL.

```

OB1 : Título:
-----
Segm. 1: Título:
-----
CALL "CREAT_DB"
  LOW_LIMIT:=MW0
  UP_LIMIT :=MW2
  COUNT   :=MW4
  RET_VAL :=MW6
  DB_NUMBER:=MW8
    
```

Fig. 110

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

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

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

3.10 Sistemas de numeración

Ejercicio 10: Sistemas de numeración



TEORÍA

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

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

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

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

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

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

SISTEMA DECIMAL:

Es el sistema al que estamos acostumbrados.

Se llama sistema decimal porque está hecho en base diez.

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

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

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

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

SISTEMA BINARIO:

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

Veamos a que número decimal corresponde el número binario 110.

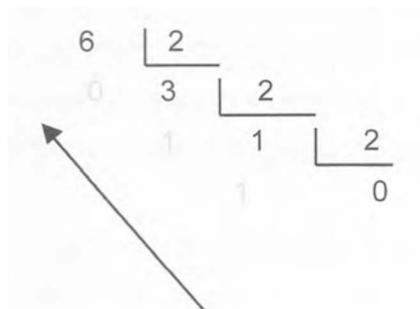
1	1	0
2^2	2^1	2^0

Si hacemos la suma correspondiente nos queda:

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

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

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



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

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

Tenemos que componer una suma de las potencias 23, 22, 21 y 20

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

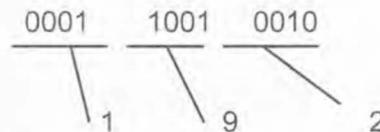
Veamos como haríamos la operación inversa.

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

0001 1001 0010

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

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

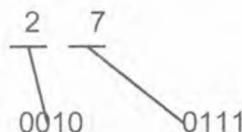


Tenemos el número 192 hexadecimal. Vemos que la conversión es muy sencilla y por grande que sea el número siempre la podemos hacer de cabeza.

SISTEMA BINARIO CODIFICADO EN BCD:

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

Veamos cómo pasamos el número decimal 27 a binario BCD.



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

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

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

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

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

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

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

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

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

3.11 Carga codificada

Ejercicio 11: Carga codificada

TEORÍA

HACER CARGAS DE TEMPORIZADORES Y CONTADORES CODIFICADAS EN BCD

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

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

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

Existen otras instrucciones de carga como las que siguen:

```
L    SST#5S
L    C#10
```

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

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

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

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

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

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

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

Ejercicio 11: Carga codificada



DEFINICIÓN Y SOLUCIÓN

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

Cuando nosotros hacemos un temporizador, escribimos las instrucciones siguientes:

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

Posteriormente podemos utilizar la instrucción:

```
L    T    1
```

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

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

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

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

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

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

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

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

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

```
10.....> 0000 0000 0000 1010
```

Esto es el número 10 en binario.

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

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

Esto es un 10 en BCD. Si ahora escribimos la instrucción de comparar, el autó-mata no sabe que cada cosa está en un formato. Nosotros le hemos introducido dos series de ceros y unos y ahora le preguntamos que si las series son iguales. Evidentemente nos dirá que las series no son iguales. La segunda serie nos dirá que corresponde a un número mayor.

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

Tendríamos que programar de la siguiente manera:

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

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

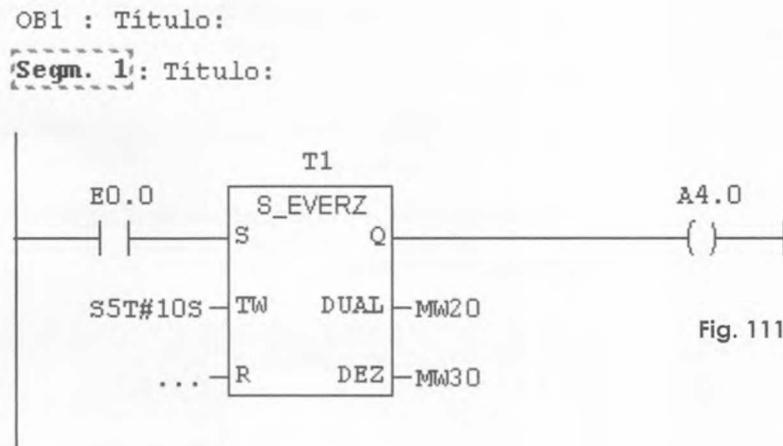
Ahora ya estamos comparando dos cosas en el mismo formato.

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

SOLUCIÓN EN AWL

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

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



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

3.12 Operaciones con números enteros

Ejercicio 10: Sistemas de numeración

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Instrucciones correspondientes a los números enteros.

Vamos a hacer más FC en la que hagamos operaciones con números enteros. Lo haremos en FC parametrizables definiendo variables de entrada, de salida y temporales.

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

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

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

Si no introducimos los datos en el formato correcto el autómata no realiza la operación. No se va a STOP. Simplemente no realiza el cálculo o lo realiza mal.

Vamos a hacer una FC con varias operaciones con enteros. Supongamos que queremos programar la siguiente función:

$$[(A + B) / C] - D = \text{Resultado}$$

Recuerda . . .

En todas las funciones y bloques de función tenemos una tabla de definición de variables. Por defecto aparece oculta, pero siempre podemos visualizarla arrastrando su barra de separación con el ratón.

A nosotros nos interesará introducir los valores A, B, C y D y obtener el resultado. No nos interesará en este caso el resultado de A+B ni el resultado luego de dividirlo entre C.

Para ello podemos crearnos una FC con variables de entrada, variables de salida y variables temporales.

La tabla de variables nos quedará de la siguiente manera:

Variables de entrada:

Contenido de: 'Entorno\Interface\IN'			
	Nombre	Tipo de datos	Comentario
+	VALOR_A	INT	
+	VALOR_B	INT	
+	VALOR_C	INT	
+	VALOR_D	INT	
+			

Fig. 112

Variables de salida:

Contenido de: 'Entorno\Interface\OUT'			
	Nombre	Tipo de datos	Comentario
+	RESULTADO	INT	
+			

Fig. 113

Variables temporales:

Contenido de: 'Entorno\Interface\TEMP'				
	Nombre	Tipo de datos	Dirección	Comentario
+	SUMA	INT	0.0	
+	DIVISION	INT	2.0	
+				

Fig. 114

Las variables temporales nosotros las definimos exactamente igual que las variables de entradas o salidas. Les damos un nombre y un formato. La diferencia está en que al llamar a la función, no se nos va a preguntar los valores de estas variables. Son datos que rellenaremos y utilizaremos dentro de la FC pero sus valores no saldrán de ella ni los podremos consultar desde fuera.

Nuestra FC quedaría de la siguiente manera:

```

FC 1
L   VALOR_A
L   VALOR_B
+I
T   SUMA

L   SUMA
L   VALOR_C
/I
T   DIVISION

L   DIVISION
L   VALOR_D
-I
T   RESULTADO
    
```

En este ejemplo hemos ido haciendo las operaciones por separado y hemos ido guardando los resultados intermedios en las variables temporales. En realidad no hubiese sido necesario el utilizar estas variables. Esto es a gusto del consumidor. Si sabemos lo que va quedando en el acumulador, no hace falta que guardemos resultados intermedios. Lo que ocurre es que a veces no nos importa utilizar 4 instrucciones más en el programa y tenemos la operación mucho más clara a la vista.

Una vez hecha la función, y antes de llamarla, tendremos que guardarla y transferirla al PLC. Hay veces que al guardar o transferir las FC en las cuales hemos escrito algo en su tabla de variables, nos sale un mensaje como el de la figura siguiente:

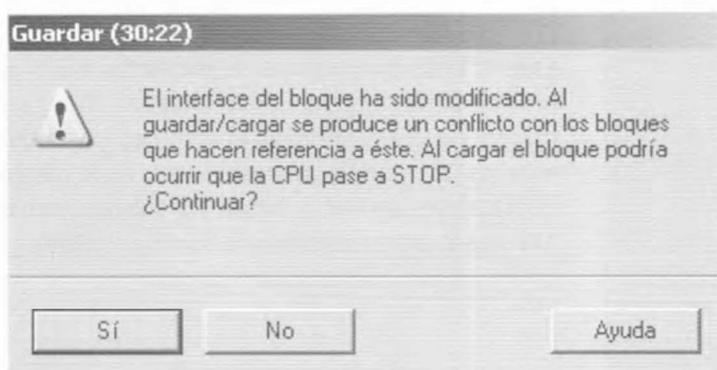


Fig. 115

Esto es porque las FC con parámetros no se pueden ejecutar por si solas. Necesitan que al ser llamadas se les rellenen los parámetros que necesitan. Imaginemos que tenemos una función que suma $A + B$. Y al llamarla desde el OB1 le hemos dado valores a A y a B. Imaginemos que este OB1 está dentro de la CPU y se está ejecutando. Posteriormente, modificamos la FC y queremos que sume $A + B + C$. Si modificamos la FC y la guardamos y enviamos al PLC antes de cambiar el OB1, se nos irá a **STOP**. Habrá un momento antes de corregir el OB1, que la función tiene 3 parámetros de entrada pero nosotros sólo le hemos dado valores a 2 de ellos. El tercero no tendrá valor y no podrá ejecutar sus operaciones. Por eso siempre que se modifica una tabla de variables, nos sale este mensaje de advertencia.

En principio le diremos aceptar. Aunque tengamos un OB1 antiguo en la CPU, como mucho se nos irá la CPU a **STOP**. Siendo esto un curso y unas pruebas no pasa nada.

Vamos ahora al OB1 y vamos a hacer la llamada a la FC que hemos generado:

OB1 : "Main Program Sweep (Cycle)"

Comentario:

Segm. 1: Título:

Comentario:

```
CALL FC      1
  VALOR_A   :=4
  VALOR_B   :=45
  VALOR_C   :=2
  VALOR_D   :=4
  RESULTADO:=MW20
```

Fig. 116

Como podemos observar en el ejemplo, no se nos pide el valor de las variables temporales. No son variables de entrada ni de salida. Sólo son válidas dentro de la función en la que han sido definidas.

Otra cuestión con respecto al programa que hemos hecho es la división de enteros. Hemos utilizado la instrucción `/I`. Al trabajar con este tipo de formato, significa que no podemos obtener decimales. Se hará la división y se obviará el resto. Más adelante en este mismo capítulo veremos como podemos hacer operaciones con números reales.

Este tipo de operaciones lo podemos hacer también en KOP o en FUP como vimos en algún ejercicio anterior. Para poder seleccionar las operaciones en estos otros lenguajes, abrimos el catálogo y vamos a ver las operaciones de números enteros. Este catálogo también nos sirve para acordarnos de cómo se escriben las operaciones en AWL y saber las operaciones que tenemos disponibles. De KOP a AWL los programas siempre son traducibles.

Recuerda . . .

Todas las operaciones matemáticas las podemos realizar tanto en KOP, como en FUP, como en AWL. En KOP y en FUP los números con los que operemos deberán ser del formato que exige el bloque de operación. En AWL podremos operar con formatos diferentes si sabemos que son equivalentes. (Por ejemplo, tanto un formato INT como un formato WORD pueden contener un número entero)

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



Fig. 117

Aquí vemos las operaciones que podemos realizar tanto en KOP como en FUP con números enteros. También tenemos las operaciones que podemos hacer con dobles enteros. Son las mismas pero trabajando con números de 32 *bits*.

3.13 Conversiones de formatos

Ejercicio 13: Conversiones de formatos

TEORÍA PREVIA: Posibles formatos en STEP 7

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

Por ejemplo, si queremos dividir 8 entre 3 y obtener un real como resultado, tendremos que convertir estos números enteros en números reales ya que la división completa es una operación de números reales.

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

Veamos las posibilidades que tenemos de cambio de formato:

BTI:	Cambia de BCD a entero de 16 bits.
ITB:	Cambia de entero de 16 bits a BCD.
DTB:	Cambia de entero de 32 bits a BCD.
BTD:	Cambia de BCD a entero de 32 bits.
DTR:	Cambia de entero doble a real.
ITD:	Cambia de entero (16 bits) a entero doble (32 bits)

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

Por ejemplo, con este tipo de operaciones no podríamos convertir un número real en entero. Si tenemos el número 2,38, al pasarlo a entero tendríamos que dejar un 2. Aquí sí que estamos perdiendo información. Las instrucciones ITR o RTB no existen. Estas operaciones en las que despreciamos parte del valor las hacemos con éstas otras instrucciones:

RND:	Redondea al número entero más cercano.
RND+:	Redondea al número entero más cercano por arriba. El resultado queda en 32 <i>bits</i> .
RND-:	Redondea al número entero más cercano por abajo. El resultado queda en 32 <i>bits</i> .
TRUNC:	Corta la parte decimal. Queda un entero en 16 <i>bits</i> .

Para probar estas instrucciones haremos lo siguiente:

OB 1:

```

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

Aquí estamos gastando instrucciones de más, pero así podemos observar como se van siguiendo los pasos en las palabras de marcas. Para hacer las conversiones no sería necesario transferir los resultados a palabras de marcas.

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

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

Veamos donde tenemos todas estas operaciones en KOP y en FUP

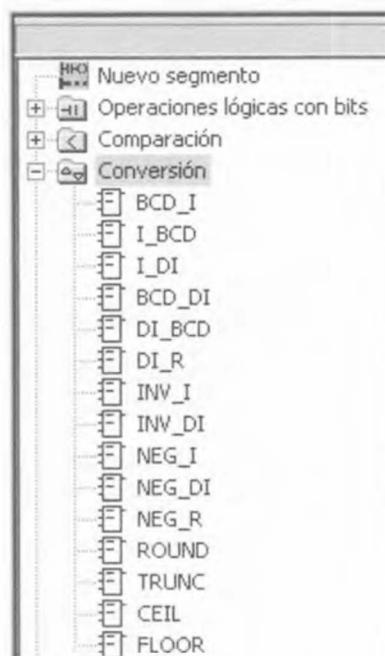


Fig. 118

Si alguna de estas operaciones no sabemos lo que significa, siempre podemos pinchar sobre ellas habiendo seleccionado previamente con el ratón el interrogante de la barra de herramientas.

Por ejemplo las instrucciones INV_I o INV_DI calculan el complemento a 1 de un entero o un doble entero. Las instrucciones NEG_I y NEG_DI calculan el complemento a dos de un entero o un doble entero.

Estas instrucciones no las hemos utilizado en el ejemplo anterior. Se recomienda entrar en la ayuda y analizar lo que hacen.

3.14 Operaciones con números reales

Ejercicio 13: Operaciones con números reales 

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Instrucciones para realizar operaciones de números reales.

Vamos a ver como realizamos las operaciones de números reales. Operaciones de reales tenemos:

+R	Suma de reales
-R	Resta de reales
*R	Multiplicación de reales
/R	División de reales
ABS	Valor absoluto de un real
SQRT	Raíz cuadrada
SQR	Cuadrado de un real
LN	Logaritmo neperiano
EXP	Exponencial
SIN	Seno
COS	Coseno
TAN	Tangente
ASIN	Arco seno
ACOS	Arco coseno
ATAN	Arco tangente

Una forma de hacerlo es utilizar números reales para todo. Así no hay problemas de formatos. Si todos los parámetros que utilizamos son reales, podemos realizar entre ellos cualquiera de estas operaciones.

Vamos a ver cómo haríamos una división de reales.

Nos creamos una FC nueva con la siguiente tabla de variables:

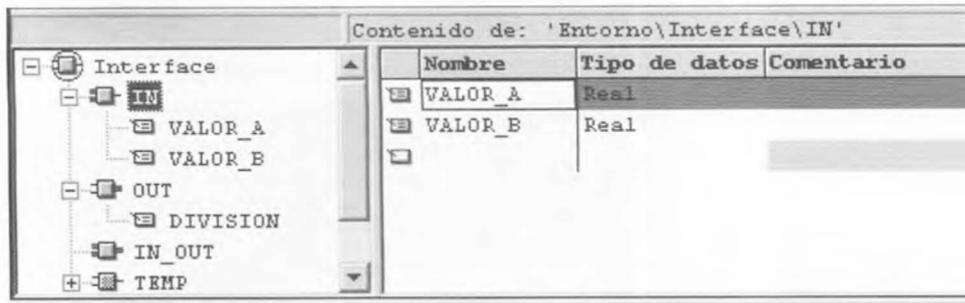


Fig. 119

Tenemos dos variables de entrada de tipo real y una variable de salida que también es de tipo real.

Vamos a programar la FC:

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

Ahora llamamos a esta FC desde el OB 1.

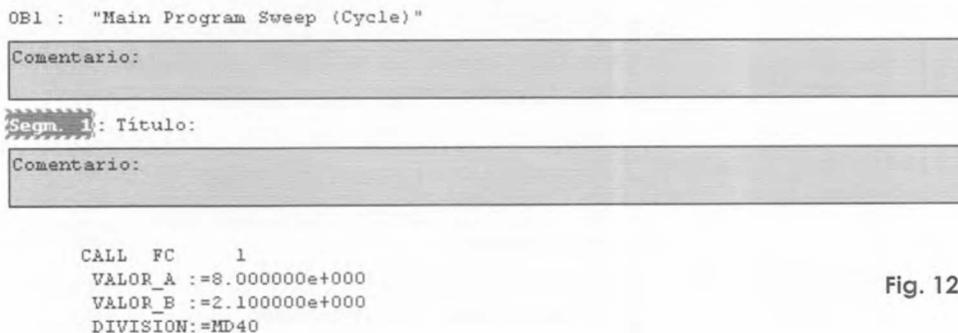


Fig. 120

Los valores que le pongamos tienen que ser reales. Si queremos dividir 8 entre 2,1 como en este caso, tenemos que escribir 8,0 y 2,1.

El resultado lo tendrá que dejar necesariamente en una doble palabra. Los números reales por definición ocupan 32 bits.

Ahora si queremos ver el resultado de la operación, tendremos que ir a la tabla de "observar / forzar variable" y observar la MD 40 en formato real.

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

No sabemos a qué número corresponde. Nosotros sabemos que el resultado es 3,81. Si observamos el resultado en formato decimal, veremos que sale un número grandísimo que no corresponde al resultado. Lo que hace la tabla de observar es traducir "literalmente" la serie de ceros y unos a decimal. Esto no es lo que queremos ver. Veríamos esto:

	Operando	Símbolo	Forma	Valor de estado	Valor de forzado
1	MD 40		DEC	L#1081331518	
2					

Fig. 121

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

Si observamos en formato real, veremos la siguiente tabla:

	Operando	Símbolo	Forma	Valor de estado	Valor de forzado
1	MD 40		REAL	3.809524	
2					

Fig. 122

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

Por ejemplo en el caso que nos ocupa, si pulsamos F1 al intentar rellenar un parámetro obtenemos la siguiente ayuda:

Tipo de datos REAL

Tipo de datos	Longitud (bits)	Formato	Ejemplos del formato	
REAL	32	Número en coma flotante	Min., positivo	Máx. positivo
			+1.175495e-38	+3.402823e+38
			Min., negativo	Máx. negativo
			-1.175495e-38	-3.402823e+38

Los números reales se redondean con exactitud hasta el sexto dígito decimal.

Los valores límite arriba indicados se representan de forma exponencial, pero los valores también pueden indicarse como números quebrados siempre que a cada lado del punto decimal haya, como mínimo, un dígito (p. ej.: 0.123456 ó 12345600000.0). Esta regla también es válida para la representación exponencial (p. ej.: 123.456e-6).

Las CPU de S7 utilizan el formato binario IEEE-FP de 32 bits para procesar números reales.

Cualquier número en coma flotante = $(-1)^s (1.f) (2^{e-127})$

siendo:

- s = bit del signo, (0 corresponde al signo positivo, 1 corresponde al signo negativo)
- e = exponente binario entero = exponente decimal entero + 127 como exponente binario de 8 bits, sin signo, entero ($0 < e < 255$)
- f = mantisa de 23 bits con el MSB igual a 2⁻¹ y el LSB igual a 2⁻²³

Formato de la CPU de S7 para números en coma flotante (32 bits)

Byte n								Bytes n+1, n+2, n+3							
	7					0	-1	-2					-22	-23	
Et de signo (s)	Exponente de 8 bits (e)						Mantisa de 23 bits (f)								

Fig. 123

En esta ayuda nos explica lo que significa cada 0 y cada 1 que hay en el acumulador cuando guardamos un número real o visualizamos en formato real.

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

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

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

Vamos a programar una FC con una tabla de variables en la que introduciremos como entradas VALOR_A Y VALOR_B de tipo enteros. Como variable de salida definiremos RESULTADO de tipo real. El programa dentro de la FC quedaría de la siguiente manera:

```
L   VALOR_A
L   VALOR_B
+I
ITD
DTR
SQRT
T   RESULTADO
```

Hemos tenido que hacer dos conversiones. El entero que resulta de la suma, lo hemos tenido que convertir en un doble entero. Después lo hemos tenido que convertir a real y, finalmente, hemos hecho la raíz cuadrada que es lo que nos interesaba. El resultado que obtenemos es un número real.

Se recomienda hacer más ejemplos utilizando operaciones de reales y operaciones de reales con enteros utilizando las conversiones correspondientes. Para observar mejor cómo funciona el programa, se recomienda transferir cada paso a una palabra de marcas y después observarlo en una tabla de observar / forzar variables.

Todas estas operaciones existen también en KOP y en FUP en la carpeta "números en coma flotante".

3.15 Control de un gallinero

Ejercicio 15: Control de un gallinero

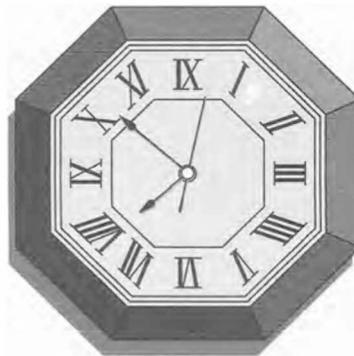


DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Programación estructurada, bloques de datos, flancos.

Vamos a controlar los días de un gallinero. Los gallineros automatizados tienen luz artificial, por lo tanto se puede jugar con las horas de día y las horas de noche que queremos dar a las pobres gallinas. Los días no son de 24 horas. Se optimizan de manera que las gallinas pongan más huevos. En granjas reales, tienen días de unas 20 horas más o menos. Así que las “semanas de las gallinas” tienen 8 días.

El PLC tiene que decidir las horas de luz y las horas de oscuridad que va a tener el gallinero en función de los huevos que pongan las gallinas.



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

El funcionamiento del gallinero debe ser el siguiente:

Llevaremos el control de los huevos que ponen las gallinas.

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

Si hoy han puesto más huevos que ayer, supondremos que las gallinas están contentas. Entonces lo que haremos será disminuir en 8 los minutos de luz y los minutos de oscuridad. De manera que les hacemos el día 16 minutos más corto. Si cada día ponen huevos, contra más cortos sean los días, más días a la semana y más huevos.

Si hoy han puesto menos huevos que ayer, supondremos que las gallinas están tristes. Entonces lo que haremos será aumentar en 5 los minutos de luz y los minutos de oscuridad. De manera que les hacemos el día más largo. Queremos tener cada vez los días más cortos hasta el punto en que dejen de poner huevos. Así ya no serán rentables. Tenemos que alargar un poco los días.

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

Recuerda . . .

Es muy importante que se tengan en cuenta los valores iniciales y actuales a la hora de crear, transferir y visualizar los bloques de datos. Sobre todo después de haber enviado el programa al PLC y haberlo ejecutado.

De esta manera, todos los días cambia la duración del día y de la noche. Así llegará a optimizarse en el punto en que las gallinas ponen más huevos. Si hacemos el día cada vez más corto, la semana cada vez tiene más días y las gallinas ponen más huevos. Llegará un punto en que no puedan descansar y esto repercuta en la puesta. Entonces les hacemos el día un poco más largo.

Para estructurarnos la programación haremos cada cosa en un bloque.

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

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

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

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

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

Finalmente haríamos una OB1 para decidir cuando tenemos que acceder a cada uno de los bloques que hemos hecho.

Veamos como quedaría esto resuelto en AWL:

En el DB1 tendríamos el control de la cantidad de huevos:

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

Fig. 124

En el DB2 tendríamos el control del tiempo:

Dirección	Nombre	Tipo	Valor inicial	Comentario
0.0		STRUCT		
+0.0	MINUTOS_LUZ	INT	720	
+2.0	MINUTOS_OSC	INT	700	
+4.0	DATO_CONTENTAS	INT	8	
+6.0	DATO_TRISTES	INT	5	
+8.0	DATO_INDIFERENTES	INT	1	
=10.0		END_STRUCT		

Fig. 125

FC1: Contadores

```

U    E    0.0
ZV   Z    1
U    E    0.1
ZR   Z    1
U    E    1.0
ZV   Z    2
U    E    1.1
ZR   Z    2
L    Z    1
T    PUESTAS.PUESTAS_HOY
L    Z    2
T    PUESTAS.PUESTAS_AYER
    
```

FC2: Comparaciones

```

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

```

L      TIEMPOS.DATO_TRISTES
+I
T      TIEMPOS.MINUTOS_OSC
BEA
INDI: L      TIEMPOS.MINUTOS_LUZ
      L      TIEMPOS.DATO_INDIFERENTES
      -I
      T      TIEMPOS.MINUTOS_LUZ
      L      TIEMPOS.MINUTOS_OSC
      L      TIEMPOS.DATO_INDIFERENTES
      -I
      T      TIEMPOS.MINUTOS_OSC
    
```

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

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

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

En nuestro caso haremos la comparación cuando le demos a una entrada. Así no tendremos que esperar todo el día para comprobar si funciona el ejemplo que hemos hecho. Supondremos que al activar la E0.0 ha pasado el día completo.

OB1

```

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

Hemos utilizado una instrucción nueva que no habíamos visto hasta ahora. "FP". Significa "flanco positivo". Esta instrucción siempre la tenemos que utilizar con un bit auxiliar que utilizada la CPU internamente para calcular este flanco. ¿Para qué hemos utilizado el flanco en el ejemplo? Si no hubiésemos puesto esta instrucción, mientras estuviese la E0.0 activa, estaríamos sumando o restando minutos una vez cada ciclo de scan. Nosotros no queremos eso. Queremos que al pasar el día se hagan las comparaciones y se sume o reste una sola vez.

Cuando consultamos un bit tenemos 4 estados.

1.- Mientras está activo. Lo consultaríamos así:

```
U     E     0.0
```

2.- Mientras NO está activo. Lo consultaríamos así:

```
UN    E     0.0
```

3.- Justo en el momento en que se activa. Lo consultaríamos así:

```
U      E      0.0
FP     M      0.0
```

4.- Justo en el momento en que se desactiva. Lo consultaríamos así:

```
U      E      0.0
FN     M      0.0
```

Si no ponemos la instrucción de flanco, por deprisa que le demos a la entrada va a suponer más de un ciclo de scan y la comparación se va a hacer más de una vez. Sumaría o restaría en su caso más de una vez y nos quedaríamos sin tiempo en un instante. Con las instrucciones programadas en el ejemplo le estamos diciendo que justo en el momento en que activamos la E0.0 y sólo durante un ciclo de scan, vaya a leer y ejecutar la FC2.

Ejercicios propuestos

Ejercicio propuesto 1: Resolver el problema en KOP y en FUP con las instrucciones vistas anteriormente.

Ejercicio propuesto 2: Resolver el mismo ejercicio pero en lugar de utilizar la entrada para simular el paso del día utilizar un temporizador. Hacer la prueba con segundos en lugar de minutos. También se propone cambiar el funcionamiento de los contadores. Hacer el ejercicio de manera que siempre contemos en HUEVOS_HOY. Al pasar el día, la cuenta debería guardarse como HUEVOS_AYER y HUEVOS_HOY debería valer 0.

Recuerda . . .

En KOP y FUP sólo tenemos dos tipos de saltos. En AWL existen muchos más tipos de saltos. Aunque no sean objeto de este libro siempre se pueden consultar en la ayuda de AWL desde el editor de bloques.

3.16 Operaciones de salto

Ejercicio 16: Operaciones de salto ✓

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Explicación de los saltos.

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

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

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

SPA y SPB

Son los saltos que vimos en el ejercicio de las metas. En **S5** son los saltos que utilizábamos para saltar entre distintos bloques de programa. En **S7** son los saltos que utilizamos para saltar a otros puntos de programa siempre dentro del mismo bloque.

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

El significado de estos dos saltos básicos es el que ya vimos con las metas. El **SPA** es un salto absoluto y el **SPB** es un salto condicional, es decir, depende del RLO.

Veamos dos ejemplos:

```

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

En el primer caso siempre que se lea la instrucción se va a producir un salto a la meta M001. En el segundo caso se saltará a la meta M001 si la E 0.0 está activa. En caso contrario no se producirá el salto.

Tenemos otros saltos como: **SPB SPBN SPBB SPBNB**

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

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

También tenemos saltos combinados como el **SPBNB**.

También tenemos los saltos que se refieren a operaciones matemáticas.

- SPZ** Salta cuando el resultado es distinto de cero.
- SPP** Salta cuando el resultado es positivo.
- SPO** Salta cuando hay desbordamiento.
- SPS** Salta cuando hay desbordamiento.
- SPZP** Salta cuando el resultado es mayor o igual que cero.
- SPU** Salta cuando el resultado no es coherente.

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

El salto SPL es un seleccionador de posiciones.

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

Veamos un ejemplo:

L	#NÚMERO
	SPL ERRO
	SPA M000
	SPA M001
	SPA M002

Si “NÚMERO” vale 0 el programa saltará a la meta M000. Si “NÚMERO” vale 1, el programa saltará a la meta M001. Y si “NÚMERO” vale 2, el programa saltará a la meta M002. Si “NÚMERO” vale cualquier cosa diferente de 0, 1 o 2, el programa saltará a la meta “ERRO”.

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

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

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

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

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



Fig. 126

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

Para situar posteriormente las metas, tenemos LABEL.

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

3.17 Mezcla de pinturas

Ejercicio 17: Mezcla de pinturas



DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Saltos.

Queremos hacer una mezcla de pintura. Tenemos tres posibles colores para hacer. El color ocre es el código 1. El color verde es el código 2. Y el color marrón es el código 3.

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

Para formar el ocre queremos poner 60 gramos de amarillo, 20 gramos de azul y 20 gramos de rojo.

Para formar el verde queremos 50 gramos de amarillo y 50 gramos de azul.

Para formar el marrón queremos 40 gramos de amarillo, 30 de azul y 30 de rojo.

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

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

Nos generamos una FC en la que COLOR será una variable de entradas de tipo entero. Es allí donde seleccionaremos el color deseado.

SOLUCIÓN EN AWL. FC 1

```

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

```

L      20
T      MW   2
L      20
T      MW   4
BEA
VERD: L  50
T      MW   0
L      50
T      MW   2
L      0
T      MW   4
BEA
MARR: L  40
T      MW   0
L      30
T      MW   2
L      30
T      MW   4

```

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

Ahora desde el OB1 llamamos a la FC1 y le decimos el color que queremos formar.

```

OB    1
CALL FC    1
COLOR:= 1

```

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

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

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

3.18 Instrucciones que afectan al RLO (NOT, CLR, SET, SAVE)

Ejercicio 18: Instrucciones que afectan al RLO (NOT, CLR, SET, SAVE) ✓

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Introducción sobre las instrucciones.

NOT: Invierte el RLO.

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

CLR: Pone el RLO a 0 incondicionalmente.

SAVE: Hace una copia en el registro BIE de la palabra de estado del valor actual del RLO.

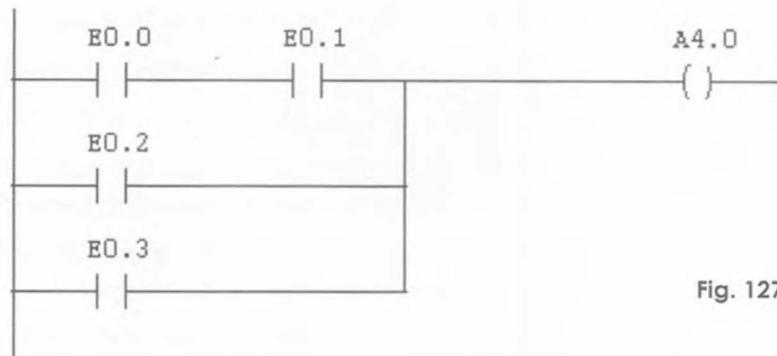


Fig. 127

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

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

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

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

Veamos como quedaría el ejemplo resuelto en AWL.

```

U   E   0.0
U   E   0.1
SAVE
O(
U   E   0.2
U   E   0.3
)
    
```

```

O      E      0.4
=      A      4.0

U      E      1.2
U      E      1.3
=      A      4.1

U      A      4.0    //Si está encendida la salida
U      BIE    //Y este bit está a 1
=      A      4.7    //Se cumplía la primera condición de las 3.
    
```

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

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

Tendremos que hacer tres segmentos.

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

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

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

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

Veamos como quedaría resuelto en KOP.

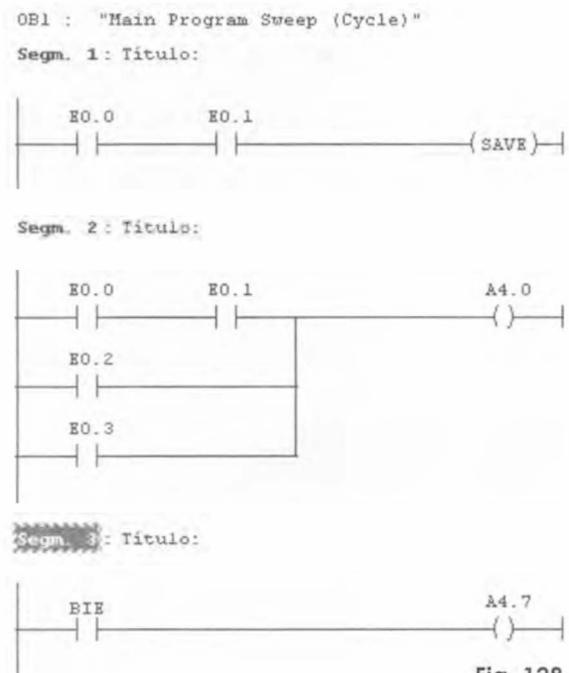


Fig. 128

3.19 Ajuste de valores analógicos

Ejercicio 19: Ajuste de valores analógicos ✓

DEFINICIÓN Y SOLUCIÓN

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

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

Dentro del tanque vamos a tener una sonda de nivel con la que queremos saber los litros de líquido que contiene. Esta sonda de nivel irá cableada a la entrada analógica del PLC. Para nosotros será la PEW 288.

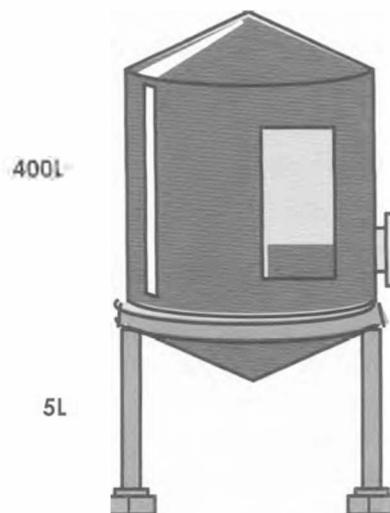


Fig. 129

Hasta ahora a las palabras de entradas les llamábamos EW y su dirección. Dijimos que estos bits correspondían a la PAE. Un registro interno de la CPU que era más rápido de acceder que a las entradas reales. El PLC hace una foto de todas sus entradas antes de empezar el ciclo de scan y es con esta foto con la que trabaja para consultar valores.

Pues bien, este registro interno tiene 256 bytes. Con lo cual todo lo que exceda de esta dirección no lo podemos consultar a través de la PAE. Debemos hacer la consulta directamente a periferia. **L PEW 288** significa cargar la palabra de entradas de periferia 288.

¿Qué tenemos en una entrada analógica? Pues básicamente lo mismo que en las entradas digitales. Una tarjeta de entradas analógicas, no es más que un convertor analógico digital. De modo que a la CPU le llegan 0 y 1 igual que con las tarjetas de entradas digitales.

Las entradas analógicas en campo pueden medir de 0 a 10 voltios, o de 4 a 20 miliamperios normalmente. Cuando la tarjeta hace la conversión de estos voltios o miliamperios a números digitales, obtenemos en todos los casos valores entre 0 y 27.648.

Recuerda . . .

Por defecto, las direcciones de señales analógicas se asignan a partir de la 256. Debemos acceder a ellas como periferia. Es decir, añadiendo una "P" delante. Por ejemplo, PEW 256. Siempre podremos cambiar estas direcciones, si nos interesa, desde el editor de *hardware*.

Cuando trabajamos en un ejemplo real, posiblemente no alcancemos este valor. Siempre hay unas tolerancias porque el potenciómetro no llega a dar lo 10 voltios o porque los miliamperios sufren alguna pérdida en los cables. Como primer ejercicio vamos a pasar la palabra de entradas analógica a una palabra de marcas para poder observarla en la tabla de **observar / forzar variables**.

L	PEW	288
T	MW	10

Ni las entradas ni las salidas de periferia, las podemos ver en la tabla de observar / forzar variables.

En la tabla de variables, observaremos el valor de MW 10. Veremos que el valor va más o menos entre 0 y 27.648. También veremos que no mide de unidad en unidad. Las tarjetas de más resolución tendrán más escalones que las tarjetas de menos resolución. Al final, para que el programa sea siempre equivalente, tanto las tarjetas de mayor resolución como las de menos, medirán como máximo 27.648.

Una vez conozcamos el valor máximo que alcanza nuestro montaje, vamos a hacer los cálculos.

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

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

Vamos a ver como quedaría el programa hecho. Podemos hacer el programa directamente en el OB1 y con los valores de este depósito (máx. 400 y min. 5), o podemos hacer el programa en un FC parametrizable. Si lo hacemos así, nos servirá después para cualquier depósito. Todo lo que pensemos que es susceptible de cambiar de un caso a otro, lo definimos como variable en la tabla de la FC. Así programamos una vez el escalado de analógicas y nos sirve para cualquier escalado que tengamos que hacer posteriormente.

En la tabla de variables introduciremos los siguientes valores:

TABLA DE VARIABLES:

IN	VALOR_SONDA	INT	En este caso será la PEW 288.
IN	NIVEL_SUPERIOR	REAL	En este caso serán 400 litros
IN	NIVEL_INFERIOR	REAL	En este caso serán 5 litros
OUT	VALOR_GRADUADO	REAL	En este caso serán los I actuales
TEMP	V_SONDA_REAL	REAL	Interno para operaciones
TEMP	RANGO	REAL	Interno para operaciones

Las variables temporales, no hace falta que las pensemos en un principio. Lo normal es que nos estructuramos lo que queremos programar y definamos las variables de entrada y salida. Con esto empezamos a hacer el programa. Cuando necesitemos guardar algún valor intermedio, algún resultado, algún bit, es en ese momento cuando definimos la variable temporal y le asignamos el formato que nos conviene.

Por ejemplo en este caso yo definiría VALOR_SONDA como parámetro de entrada. Lo defino de un principio porque es un parámetro que puede cambiar de un ejemplo a otro. Es la entrada donde tenemos la sonda cableada. Si tuviese 3 tanques en lugar de uno, cada sonda estaría cableada a una entrada diferente. Defino el parámetro de entrada de tipo entero. Hemos visto que las analógicas dan valores entre 0 y 27.648 en formato entero.

Una vez me pongo a programar, veo que necesito hacer una regla de tres. Quiero hacer divisiones de reales. El resultado lo quiero en real. El depósito puede tener 27,5 litros y así querré verlo. Por lo tanto, necesito tener el valor de la sonda en formato real. Hago la conversión a reales ¿y dónde dejo este valor? Pues es en este momento que me creo la variable temporal V_SONDA_REAL de tipo real. Yo siempre daré un entero como parámetro de entrada, pero luego dentro de la función haré los cálculos con el valor convertido a real.

FC 1 : Escalado de analógicas

```

L      #VALOR_SONDA
ITD
DTR
T      #V_SONDA_REAL
L      #NIVEL_SUPERIOR
L      #NIVEL_INFERIOR
-R
T      #RANGO
L      #V_SONDA_REAL
L      26624.0      //Valor máximo que daba nuestro simulador
/R
L      #RANGO
*R
L      #NIVEL_INFERIOR
+R
T      #VALOR_GRADUADO
BE

OB1
CALL  FC      3
      VALOR_SONDA:= #PEW288
      NIVEL_SUPERIOR:= 400.0
      NIVEL_INFERIOR:= 5.0
      VALOR_GRADUADO:= MD 0

```

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

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

En la tabla de variables observaríamos lo siguiente:

	Operando	Símbolo	Forma	Valor de estado	Valor de forzado
1	MD 0		REAL	42.55	
2					

Fig. 130

COMENTARIO A CERCA DE LAS VARIABLES TEMPORALES

Cada vez que entramos en una FC, tenemos disponible su tabla de variables. Esta tabla de variables ocupa una zona de memoria del PLC llamada "L". Nosotros también podemos acceder a la LW 0 o a la LD 4. Aunque estos accesos a la zona de variables locales no suele ser habitual. ¿Qué ventaja tenemos utilizando esta zona de variables con respecto a la zona de marcas por ejemplo? En otros PLC de otros fabricantes, se dispone de mayor número de marcas, para en un principio que los **SIEMENS** tengan menos recursos. Pero en realidad no es así. Estas variables locales, tienen la ventaja de que utilizamos la misma zona de memoria en todas las FC. Y no se machacan los valores. Tenemos la memoria muy optimizada. Tanto si son variables de entrada como si son temporales, al entrar en la FC toma sus valores y con ellos trabaja. Al terminar de ejecutar el bloque y saltar a otra función, la misma zona de memoria utiliza otros valores en la misma zona de variables locales.

Esto es una ventaja muy grande. Lo único que tenemos que tener en cuenta es que no podemos suponer que un valor local, quedará permanente con ese valor. Por ejemplo, si tenemos una FC con metas y en una meta ponemos un bit temporal a 1, cuando se deje de ejecutar la meta, no podemos suponer que el bit estará a 1. Si queremos esta funcionalidad, deberemos usar marcas. En cambio, en la gran mayoría de los casos nos serán muy útiles las variables locales. En el ejemplo del escalado de analógicas, no tenemos ningún problema con las variables utilizadas. A las variables de entrada se les da valor al llamar a la FC. Con lo cual no tenemos ningún problema. Y a las dos variables temporales que tenemos, les damos valor durante la FC tras resolver unos cálculos. Siempre dentro de la FC tendremos los valores correctos. Al salir de esta FC, podemos utilizar esa zona de memoria para otras cosas.

3.20 Ajuste de valores analógicos con funciones de librería

Ejercicio 20: Ajuste de valores analógicos con funciones de librería ✓

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Manejo de analógicas.

Lo que hemos visto en el ejercicio anterior, también lo podemos hacer con unas FC ya programadas. Funciones ya programadas tenemos dentro de la CPU (SFC) y FC y FB de librerías que vienen incluidas en el **STEP 7**.

Vamos a utilizar una FC de la librería estándar del **S7**.

Primero nos creamos una carpeta de programa nueva donde vamos a realizar este ejemplo. Después vamos a abrir la librería. Para ello vamos al menú “**Archivo** → **Abrir...**” Entonces veremos un diálogo como el que se muestra a continuación.

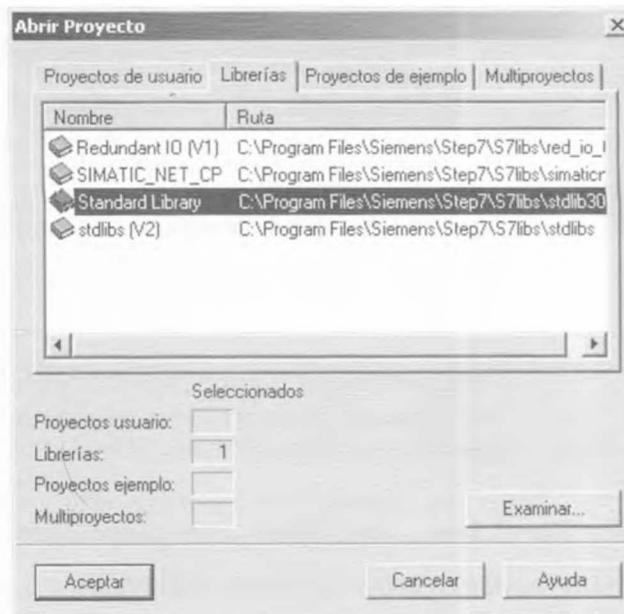


Fig. 131

Tendremos que situarnos en la ficha de “**Librerías**” y seleccionar las librerías estándar como vemos en el ejemplo.

Al pulsar aceptar, vemos las siguientes carpetas de programa:

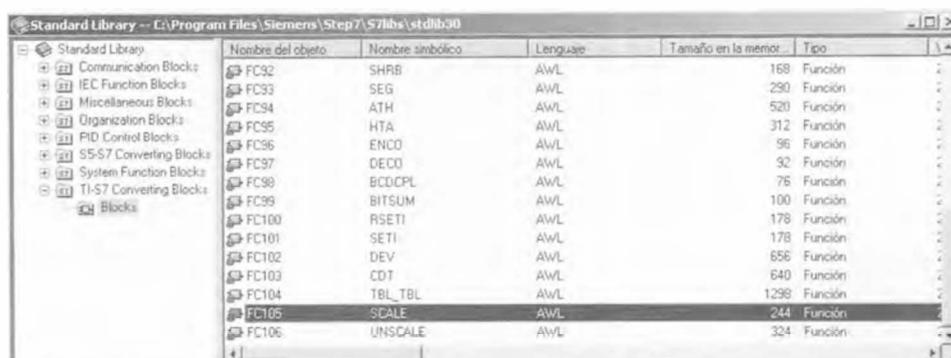


Fig. 132

Recuerda . . .

En las librerías estándar del Administrador de Simatic podremos encontrar numerosas funciones para utilizar en nuestras aplicaciones. Todas ellas tienen su Ayuda, en la que se nos explica cómo funcionan y los parámetros que tienen.

Lo que vemos es igual que los proyectos que estábamos haciendo hasta ahora de ejemplo. Son carpetas de programa sin *hardware* definido y en cada carpeta una serie de FC y FB programadas. Estas funciones también las tenemos protegidas igual que las SFC que vimos anteriormente. Podemos utilizar las funciones pero no podemos ver el código programado dentro de ellas.

Dentro de la carpeta **TI-S7 Converting Blocas** tenemos la FC 105 que se llama SCALE. Nos sirve para escalar valores analógicos. Esto es lo mismo que hemos hecho en el ejercicio anterior.

Si la seleccionamos con el interrogante de ayuda, se nos explica cómo funciona esta FC y los parámetros que nos pedirá cuando la llamemos. Veamos esta ayuda a modo de ejemplo:

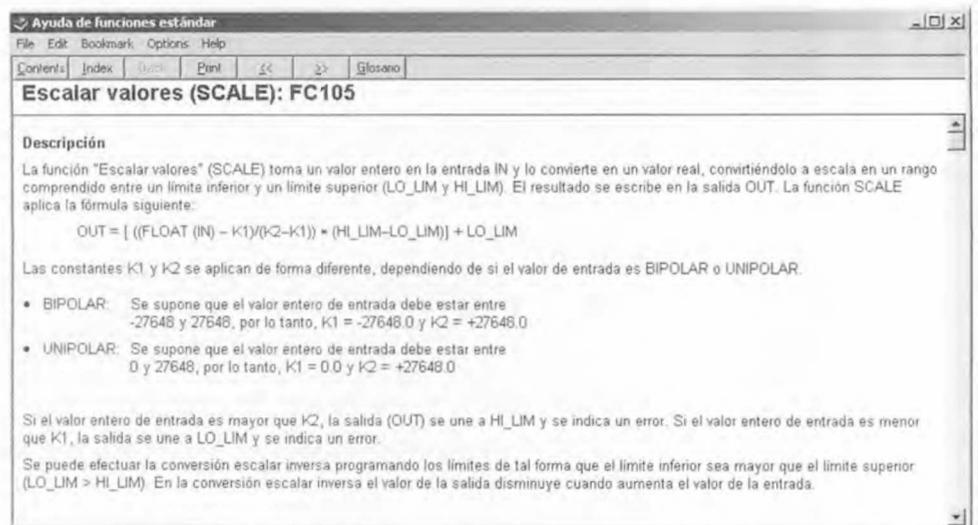


Fig. 133

Como vemos esta función hace prácticamente lo mismo que la que hemos generado nosotros en el proyecto anterior. Además nos da información de posibles errores.

Lo que tenemos que hacer con esta función es arrastrarla a nuestro proyecto. Tendremos que transferirla también al PLC para poderla usar.

Vamos al OB 1 y llamamos a la función. Los parámetros que nos pide son:

- IN: Valor a escalar
- HI_LIM: Límite superior
- LO_LIM: Límite inferior
- BIPOLAR: Si tenemos tarjetas 0 – 10 v. ó -10 – 10 v.
- RET_VAL: Código de error si es que se produce
- OUT: Salida escalada.

```

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

Fig. 134

Simplemente programando esto tendremos lo mismo que habíamos hecho en el ejercicio anterior.

3.21 Ejemplo con UDT

Ejercicio 21: Ejemplo con UDT

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Tipo de datos de usuario.

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

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

Vamos a organizarnos los datos utilizando los UDT.

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

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

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

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

Para generar un UDT 1 tenemos que situarnos en el Administrador de **SIMATIC** e insertar un nuevo objeto tal y como lo habíamos hecho antes para el resto de bloques (DB, FC).

En este caso insertamos un tipo de datos.

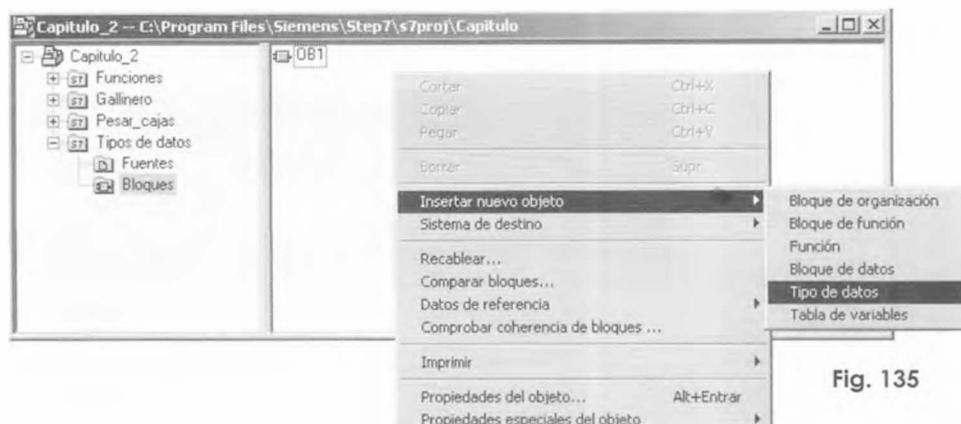


Fig. 135

Veamos cómo quedaría nuestro UDT 1.

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

Fig. 136

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

Dirección	Nombre	Tipo	Valor inicial	Comentario
0.0		STRUCT		
+0.0	MAQ_1	UDT1		
+8.0	MAQ_2	UDT1		
+16.0	MAQ_3	UDT1		
+24.0	MAQ_4	UDT1		
=32.0		END_STRUCT		

Fig. 137

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

Ahora cada planta será de tipo UDT 2.

Dirección	Nombre	Tipo	Valor inicial	Comentario
0.0		STRUCT		
+0.0	PLANTA_1	UDT2		
+32.0	PLANTA_2	UDT2		
+64.0	PLANTA_3	UDT2		
=96.0		END_STRUCT		

Fig. 138

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

Sin escribir demasiadas líneas tenemos un montón de datos acumulados.

Ahora tenemos que hacer el DB.

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

Le llamamos DATOS.

Dentro de datos ponemos:

Dirección	Nombre	Tipo	Valor inicial	Comentario
0.0		STRUCT		
+0.0	POLI_1	UDT3		
+96.0	POLI_2	UDT3		
=192.0		END_STRUCT		

Fig. 139

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

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

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

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

44.0	POLI_1.PLANTA_2.MAQ_2.MARCHA	BOOL	FALSE	FALSE
46.0	POLI_1.PLANTA_2.MAQ_2.PIEZAS	INT	0	0
48.0	POLI_1.PLANTA_2.MAQ_3.TEMPERATURA	REAL	0.000000e+000	0.000000e+000
52.0	POLI_1.PLANTA_2.MAQ_3.MARCHA	BOOL	FALSE	FALSE
54.0	POLI_1.PLANTA_2.MAQ_3.PIEZAS	INT	0	0
56.0	POLI_1.PLANTA_2.MAQ_4.TEMPERATURA	REAL	0.000000e+000	0.000000e+000
60.0	POLI_1.PLANTA_2.MAQ_4.MARCHA	BOOL	FALSE	FALSE
62.0	POLI_1.PLANTA_2.MAQ_4.PIEZAS	INT	0	0
64.0	POLI_1.PLANTA_3.MAQ_1.TEMPERATURA	REAL	0.000000e+000	0.000000e+000
68.0	POLI_1.PLANTA_3.MAQ_1.MARCHA	BOOL	FALSE	FALSE
70.0	POLI_1.PLANTA_3.MAQ_1.PIEZAS	INT	0	0
72.0	POLI_1.PLANTA_3.MAQ_2.TEMPERATURA	REAL	0.000000e+000	0.000000e+000
76.0	POLI_1.PLANTA_3.MAQ_2.MARCHA	BOOL	FALSE	FALSE
78.0	POLI_1.PLANTA_3.MAQ_2.PIEZAS	INT	0	0
80.0	POLI_1.PLANTA_3.MAQ_3.TEMPERATURA	REAL	0.000000e+000	0.000000e+000
84.0	POLI_1.PLANTA_3.MAQ_3.MARCHA	BOOL	FALSE	FALSE
86.0	POLI_1.PLANTA_3.MAQ_3.PIEZAS	INT	0	0
88.0	POLI_1.PLANTA_3.MAQ_4.TEMPERATURA	REAL	0.000000e+000	0.000000e+000
92.0	POLI_1.PLANTA_3.MAQ_4.MARCHA	BOOL	FALSE	FALSE
94.0	POLI_1.PLANTA_3.MAQ_4.PIEZAS	INT	0	0

Fig. 140

Esto es parte del DB que hemos creado.

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

DATOS.POLI1.PLANTA3.MAQ2.TEMP

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

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

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

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

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

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

Recuerda . . .

Para poder utilizar los símbolos de dentro de los UDT y de los DB, antes debemos asignar un nombre al DB desde la tabla de definición del simbólico global.

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

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

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

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

3.22 Operaciones lógicas con palabras

Ejercicio 22: Operaciones lógicas con palabras 

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Concepto de operación lógica.

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

Las operaciones que podemos hacer son las operaciones lógicas del álgebra: **AND**, **OR** y **XOR**.

Esto lo utilizaremos básicamente para hacer enmascaramientos de *bits*.

Las instrucciones para realizar estas operaciones son las siguientes: **UW**, **OW**, **XOW**

Las utilizaremos de la siguiente manera:

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

Cargamos las dos palabras con las cuales queremos hacer la operación y a continuación escribimos la instrucción. El resultado quedará en el acu 1 y lo podemos transferir donde queramos para verlo.

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

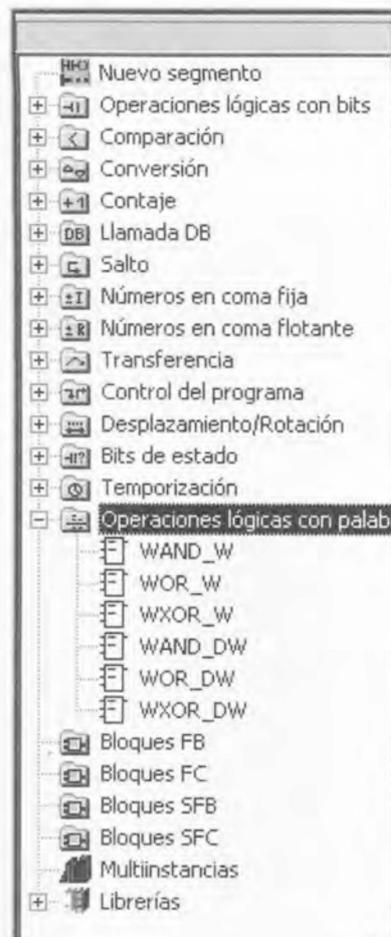


Fig. 141

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

3.23 Alarmas

Ejercicio 23: Alarmas

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Concepto de operación lógica.

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

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

Veamos un ejemplo:

```

0001_1001_0101_0110
-----
0101_0000_1110_1000
0100_1001_1011_1110
    
```

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

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

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

11100010

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

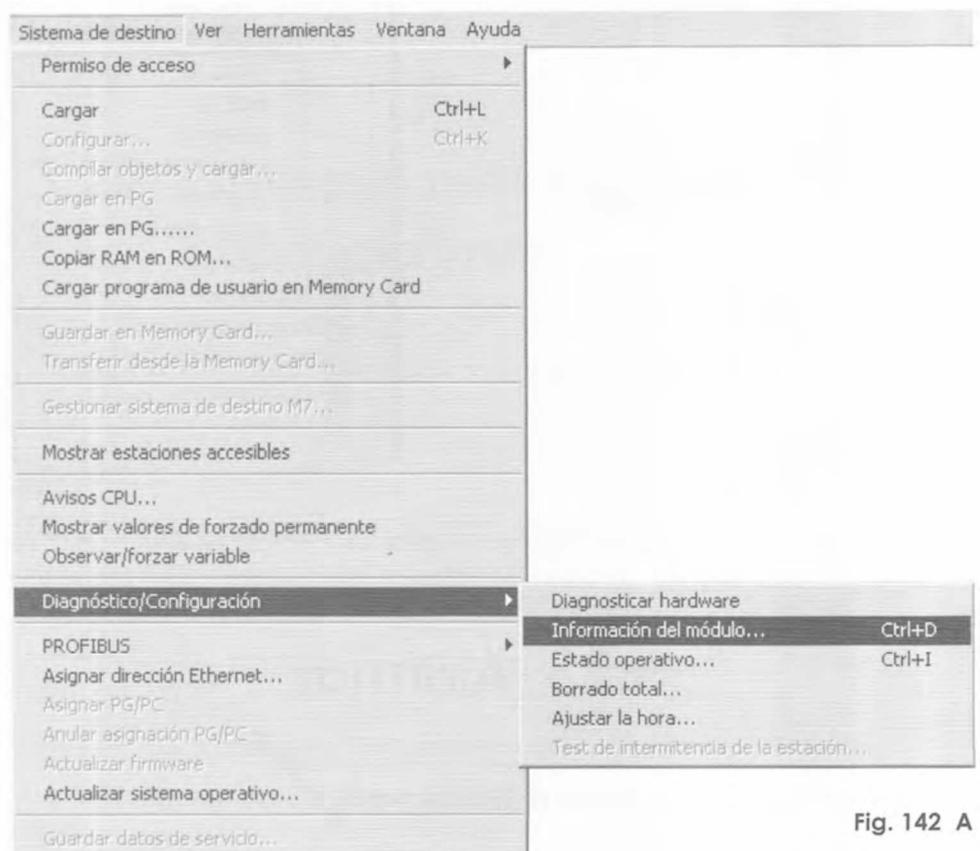
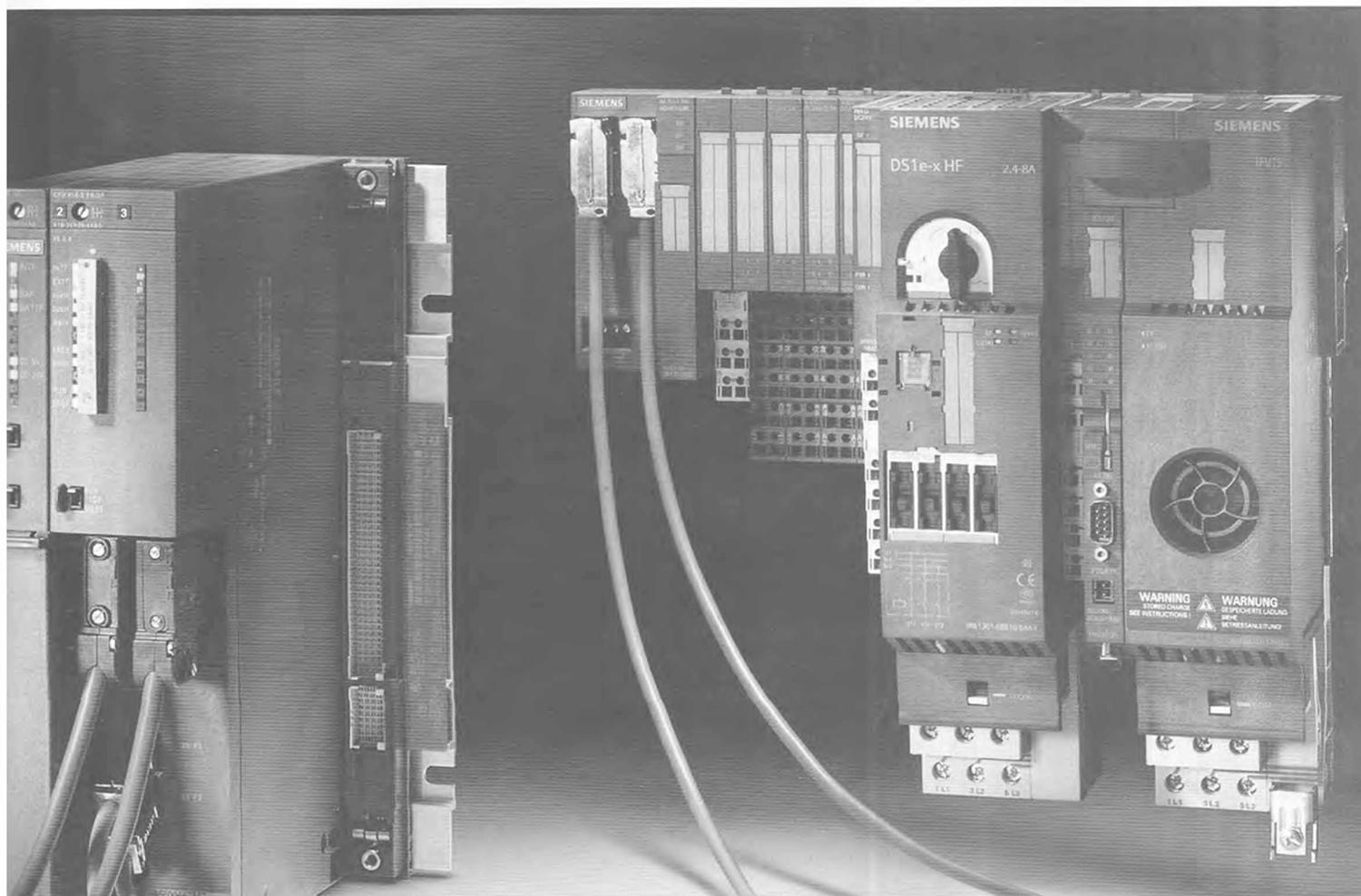


Fig. 142 A

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

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

Unidad 4 Operaciones de sistema



En este capítulo:

- 4.1. Detección de errores
- 4.2. Relación de OB y SFC
- 4.3. Instrucción LOOP
- 4.4. Programación OB 80 (SFC 43)
- 4.5. OB 100, 101. Retardo en el arranque
- 4.6. Programación de alarmas cíclicas
- 4.7. Programación de alarmas horarias por *hardware*
- 4.8. Programación de alarmas horarias por *software*
- 4.9. Programación de alarmas de retardo
- 4.10. Ajustar la hora
- 4.11. Formatos fecha y hora
- 4.12. Hacer funcionar algo un día de la semana
- 4.13. Convertir archivos de S5 a S7
- 4.14. Programar archivos fuente y protección de bloques
- 4.15. Direccionamiento indirecto
- 4.16. Control de fabricación de piezas
- 4.17. Cargar longitud y número de DB
- 4.18. Comparar dobles palabras
- 4.19. Referencias cruzadas
- 4.20. Comunicación MPI por datos globales
- 4.21. Red PROFIBUS DP. Periferia descentralizada
- 4.22. Simulador de CPU
- 4.23. Realizar copias de seguridad

Resolución de ejercicios

Recuerda . . .

Step 7 nos ofrece una amplia información para detectar errores y averías de nuestra CPU. También podemos utilizar éstas y otras herramientas para detectar averías de nuestra instalación sin que sean un fallo del PLC.

4.1 Detección de errores

Ejercicio 1: Detección de errores 

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Menú "Información del módulo".

Además de todo lo que hemos visto en cuanto a programación, el **STEP 7** nos ofrece muchas herramientas en cuanto a diagnóstico se refiere. Cuando tengamos algún problema con los equipos utilizados o con las comunicaciones entre equipos, vamos a disponer de mucha información que nos ayudará a encontrar rápido los problemas y a resolverlos.

Vamos a empezar analizando qué ocurre y qué herramientas podemos utilizar cuando la CPU se nos va a **STOP** o simplemente cuando se le enciende la luz de error.

Si nos fijamos en la CPU, junto a la llave de arranque, dispone de unos leds informativos. Tenemos una luz llamada SF. Se encenderá en rojo cuando tengamos un fallo de sistema. Tenemos una luz naranja llamada Batt. Se encenderá cuando tengamos baja la batería de la CPU. Tenemos una luz verde llamada DC. Se encenderá cuando tengamos la CPU alimentada. Tenemos otra luz verde llamada **RUN**. Se encenderá cuando tengamos la CPU en **RUN**. Es decir, ejecutando el programa. Por último, tenemos una luz naranja llamada **STOP**. Se encenderá cuando tengamos la CPU en **STOP**. Bien porque la hayamos puesto nosotros con la llave o bien porque tenga algún problema y el sistema la haya enviado a este estado. Si tenemos una CPU con algún bus integrado, también tendremos las luces rojas que nos indicarán que tenemos fallo en el bus.

Cuando el PLC se va a **STOP** o simplemente nos da un error, nos ofrece un menú de información donde nos dice cual es el problema por el cual se encuentra en este estado.

Vamos a hacer un ejemplo en el cual introduzcamos errores para que el PLC se vaya a **STOP** y nos dé un error. Luego le pediremos información sobre el error y analizaremos las opciones de las que disponemos.

Vamos a crear tres bloques como los que se muestran a continuación:

OB	1	
U	E	0.0
CC	FC	1

FC	1	
UC	FC	2

FC	2	
U	E	0.7
CC	FC	4

En principio, no le vamos a programar ninguna FC 4. Enviamos estos tres bloques a la CPU. El PLC ejecutará el programa y veremos que inicialmente no pasa nada. EL PLC no da ningún tipo de error. Esto es porque de momento sólo está leyendo el OB1. Como no encuentra activa la E0.0, no busca la FC1. Cuando acaba de leer este bloque, vuelve a empezar. Aquí no encuentra ningún error. Cuando esté activa la E 0.0, el PLC irá a leer la FC1. De allí lo mandamos a la FC2. Mientras no esté activa la E0.7, no buscará la FC4. Con lo cual volverá al OB1 y terminará el programa sin problemas. Será en el momento en que tengamos activa la E0.0 y activemos la E0.7, cuando intente saltar a la FC4 y no la encuentre. Entonces el PLC dará un fallo y se irá a **STOP** la CPU. Veremos que se encienden las luces de SF y **STOP**. Aunque nosotros pasemos la CPU a estado **RUN**, ella sola volverá al estado de **STOP**.

Una vez lo tengamos en esta situación vamos a ir al Administrador de **SIMATIC**. Desde allí vamos al menú “**Sistema destino → Diagnóstico / configuración → Información del módulo...**”.

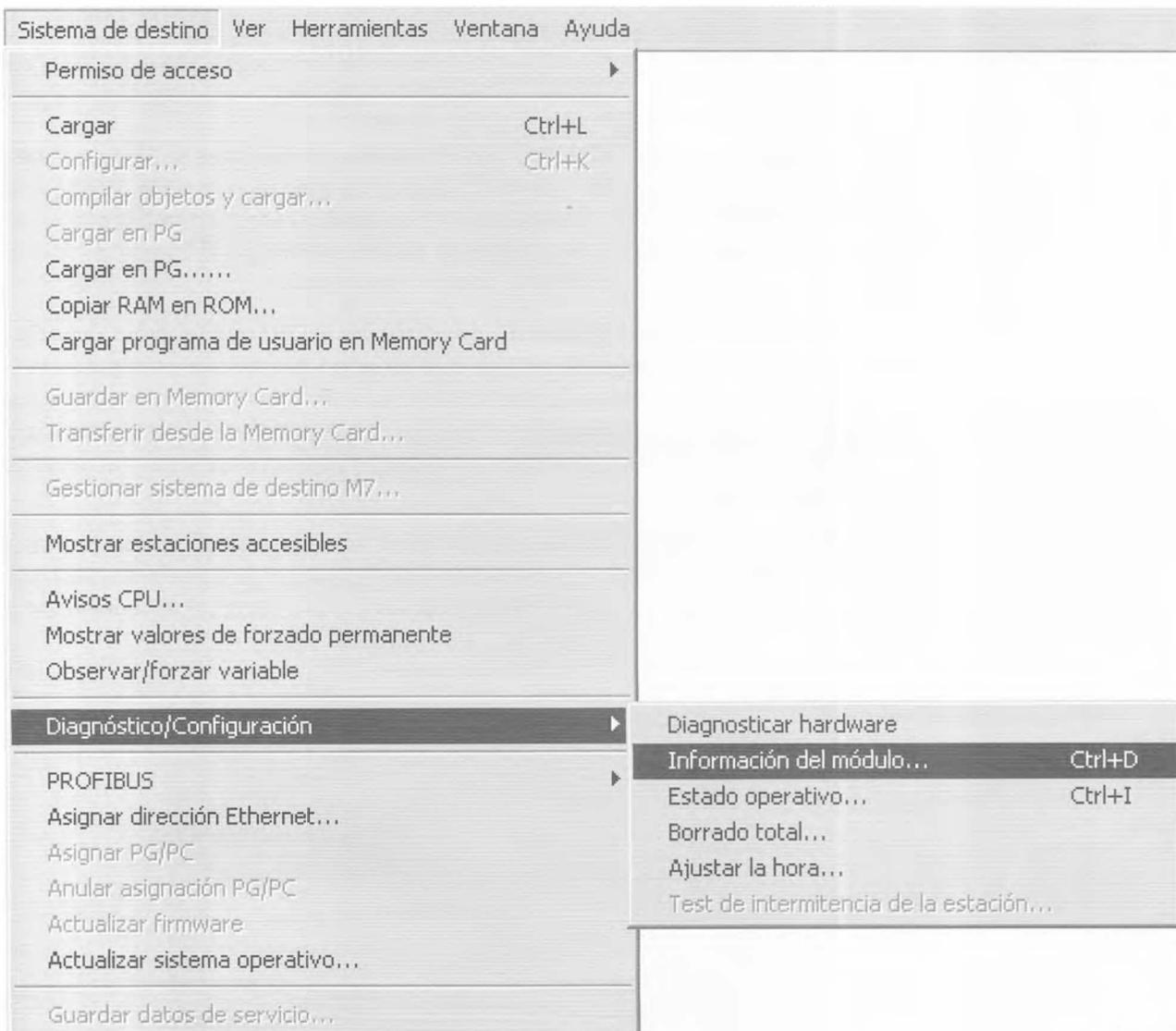


Fig. 142

Se nos va a abrir una ventana en la que podemos encontrar mucha información. Veamos un ejemplo de lo que podemos ver en este caso:

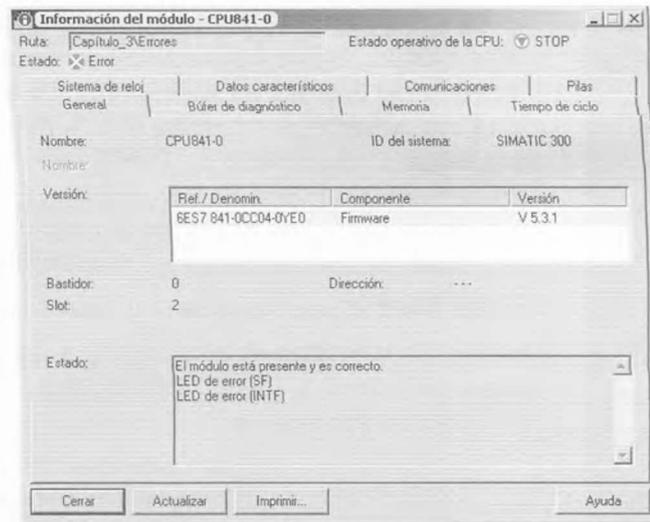


Fig. 143

Nada más entrar en este menú podemos observar que tenemos la CPU en **STOP**. Esto nos será muy útil cuando estemos haciendo diagnóstico remoto. Arriba a la derecha vemos el estado operativo de la CPU de modo gráfico. Además en la parte superior izquierda vemos que su estado es de Error. También lo vemos de forma gráfica.

En la primera ficha llamada **“General”**, podemos ver la CPU que tenemos conectada y los *leds* que tiene encendidos. Hay que tener en cuenta que en algunas versiones de STEP 7, esta información no se refresca sola. Si la CPU pasase a RUN, no nos daríamos cuenta hasta que no pulsásemos el botón de actualizar o al cerrar y volver a abrir este diálogo. En las últimas versiones del *software* si que se refresca al cambiar el estado de la CPU.

Pasamos ahora a la segunda ficha llamada **“Búfer de diagnóstico”**. En esta ventana podremos ver las 100 últimas causas (o las que tengamos predeterminadas) por las cuales el PLC se ha ido a **STOP** o ha tenido algún fallo aunque no se haya ido a **STOP**.



Fig. 144

Recuerda . . .

Existen diferentes bloques OB asociados a distintos errores de la CPU. La cantidad de OB disponibles depende de la CPU que estemos utilizando. Con ellos podemos conseguir que la CPU no se muestre Stop ante determinados tipos de errores.

En este ejemplo vemos los 11 últimos errores o eventos que ha tenido la CPU. Si pulsamos el botón de ajuste, podremos decirle que cantidad de errores queremos que almacene hasta un máximo de 1024 (también depende de la CPU que estemos utilizando).

El error número 1 siempre va a ser el último que se ha producido en el PLC. Si nos situamos encima del error, en la parte inferior de la ventana, obtenemos un poco más de información sobre el mismo. Situándonos encima del error número 1, simplemente nos dice que ha habido una petición de pasar a **STOP** y que la CPU se encuentra en este estado. Nos informa que es necesario hacer un rearranque para que funcione. De aquí obtenemos poca información. Si nos situamos en el segundo error podemos ver algo más. Como causa del error podemos leer **“STOP debido a error de programación (OB no cargado o imposible cargarlo)”**. Este error lo encontraremos prácticamente siempre que veamos la CPU en **STOP** y no sea porque la hemos llevado a este estado manualmente. Para casi todos los errores que se puedan dar en el PLC existe un OB asociado.

¿Cómo funcionan estos OB? Nosotros tenemos la opción de no programar ningún OB asociado a errores. Con lo cual, cuando la CPU encuentre un error se irá a **STOP**. O también tenemos la opción de programar los OB correspondientes a ciertos errores y allí programar lo que queremos que haga la CPU cuando existe uno de estos fallos. El número del OB que tenemos que programar no es aleatorio. A los OB no se les llama. Cada uno tiene una función predeterminada y se ejecuta cuando “le toca” según el número de OB. Los OB asociados a cierto tipo de errores, se ejecutan mientras en la CPU se produzcan eventos del tipo de fallo asociado. En el ejercicio siguiente se ve una lista de los posibles OB que podemos programar y a qué eventos los tenemos asociados.

Si pinchamos sobre el error número 3, veremos lo siguiente:

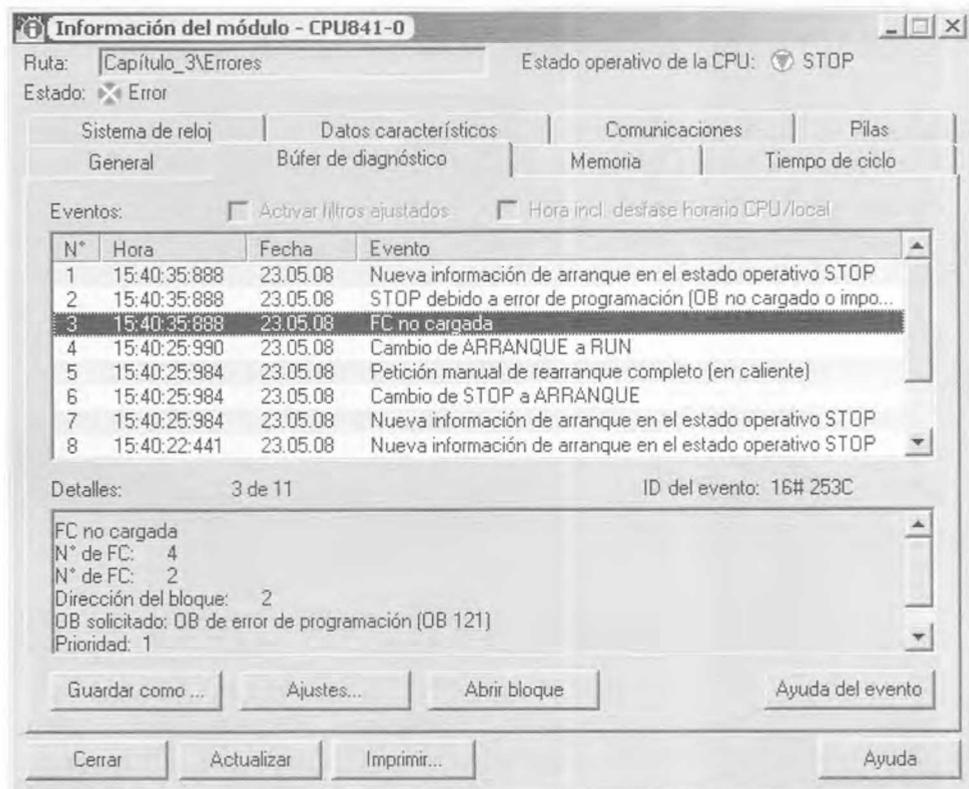


Fig. 145

El error número 3 es FC no cargada. En la parte inferior de la ventana, nos informa que la FC no cargada es la FC4, y que el PLC estaba leyendo la instrucción 2 de la FC2. También nos dice que el OB solicitado ante este error era el OB 121. En realidad la CPU ha visto que no tenía la FC4. Entonces ha ido a buscar el OB 121 esperando ver allí lo que tenía que hacer. Como no había OB 121 y nadie le ha dicho lo que tenía que hacer, entonces se ha ido a **STOP**.

Si queremos más información, podemos ir a la ficha de **"Pilas"**. Aquí podemos ver el camino que ha seguido el PLC antes de irse a **STOP**. En nuestro caso veremos que ha ejecutado el OB 1, ha saltado a la FC1, después a la FC2, y allí es donde ha quedado parado.

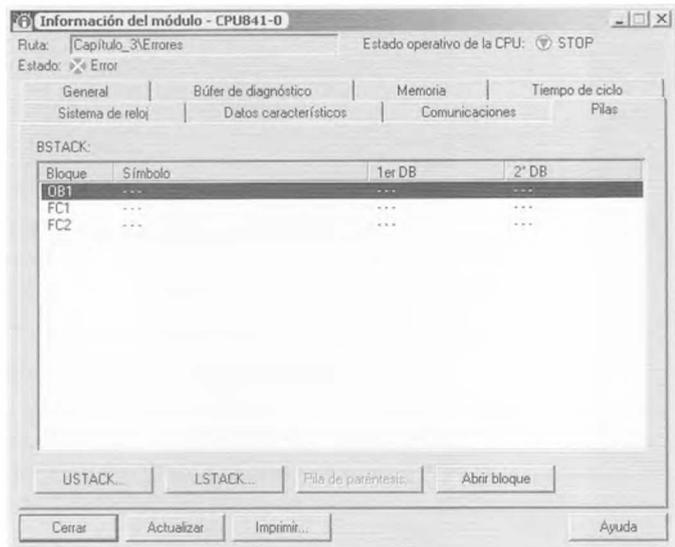


Fig. 146

En la parte inferior izquierda, vemos que tenemos un botón que se llama **"USTACK"**. Si pinchamos allí veremos el estado en el que estaba la CPU en el instante en el que se fue a **STOP**.

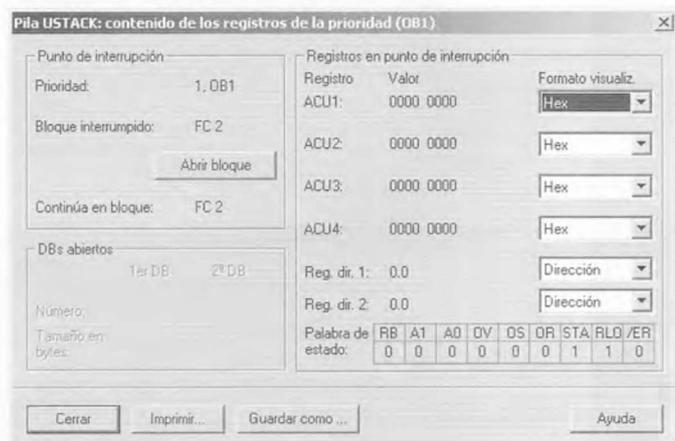


Fig. 147

En la parte derecha de la ficha nos da información del estado del PLC cuando se fue a **STOP**. Vemos en este caso que los cuatro acumuladores (o 2 acumuladores dependiendo de la CPU que estemos gastando), tenían valor 0. Podemos ver estos valores en distintos formatos. También podemos ver los valores de los diferentes *bits* de la palabra de estado. En este caso vemos por ejemplo que el valor del RLO era 1. Se cumplía la última condición que había leído. Como estaba activa la E0.7, el RLO era 1 y había ido a buscar la FC4. Al no encontrarla es cuando se ha ido a **STOP**. Además tenemos una ventana a la parte izquierda que nos indica el bloque

interrumpido y tenemos un botón que dice **“Abrir bloque”**. Si pinchamos en este botón se abrirá el bloque que contenía el error en **ONLINE**, y nos señalará con el ratón la instrucción que contiene el error.

Con todo esto hemos llegado a saber cuál era el error de la CPU, dónde estaba y por qué se había ido a **STOP**.

Además tenemos otras fichas en las cuales se nos da más información del módulo. Podemos ver el tiempo máximo, el tiempo mínimo y el tiempo real del ciclo de *scan*. Si vamos a la ficha **“Tiempo de ciclo”** veremos lo siguiente:

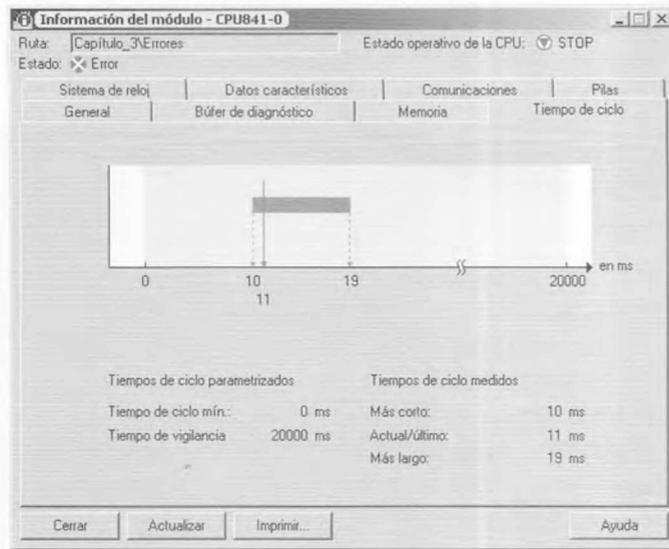


Fig. 148

Aquí podemos ver el tiempo de ciclo máximo, el mínimo y el actual. Si vamos pulsando el botón de **“Actualizar”**, veremos que el tiempo de ciclo de *scan* actual va cambiando. Tenemos un máximo y un mínimo porque el PLC no siempre ejecuta el mismo número de instrucciones. Si, por ejemplo, no se realiza la llamada a una FC, no se lee su contenido.

Si vamos a la ficha llamada **“Memoria”**, podemos ver la cantidad de memoria utilizada en el PLC. Tanto la memoria de carga como la memoria de trabajo. En el ejemplo vemos que estamos utilizando el 0% porque estamos simulando una CPU muy potente y el ejemplo programado es realmente pequeño para lo que puede almacenar una CPU de estas características. En la parte inferior de la ventana, se indica, en valor numérico la memoria utilizada.

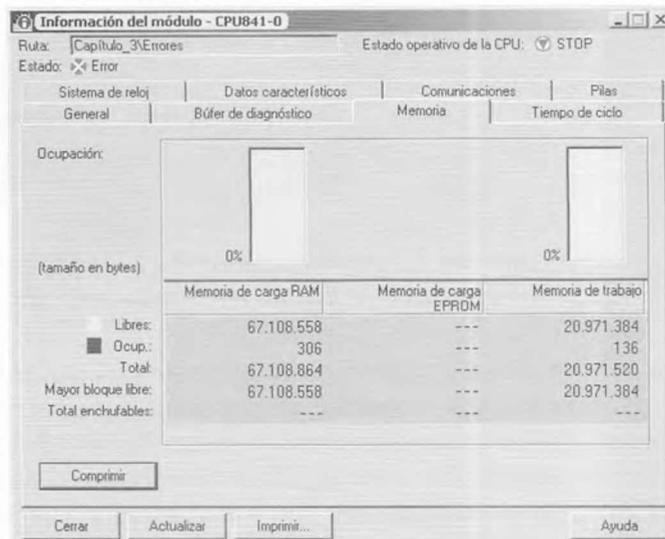


Fig. 149

En la parte izquierda, podemos ver un botón para comprimir la memoria. Cuando enviamos muchos bloques a la CPU, porque estamos haciendo y depurando un proyecto, muchas veces quedan en la memoria bloques que no se están utilizando en el programa. También se ocupa memoria si enviamos varias veces un mismo módulo con distinta cantidad de instrucciones. Si vemos que hay utilizada mucha cantidad de memoria en la CPU, siempre podemos pulsar este botón para que se optimice la memoria utilizada. En cualquier caso podemos pulsar el botón de comprimir memoria. Si es posible se comprimirá y si no es posible, nos quedaremos con la misma memoria que teníamos utilizada.

Si vamos a la ficha llamada “sistema de reloj”, veremos lo siguiente:

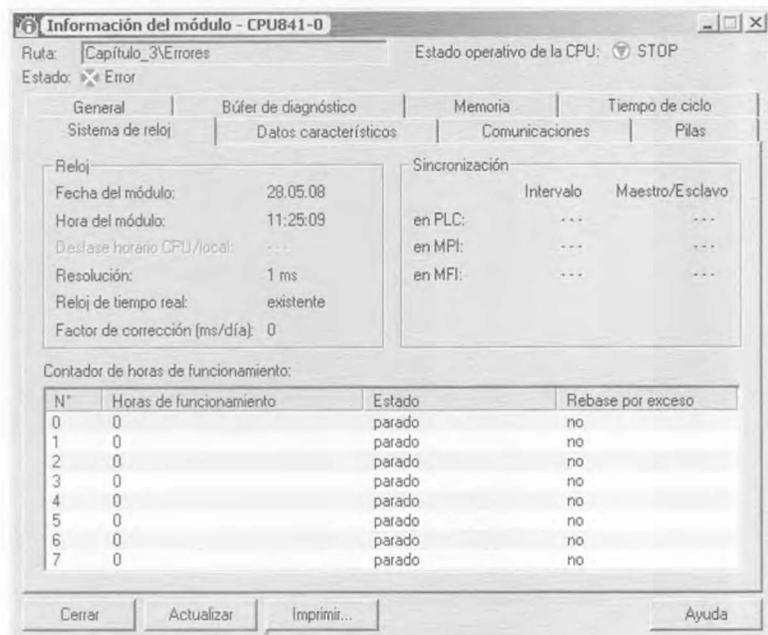


Fig. 150

Recuerda . . .

Toda esta información es **ONLINE**. Significa que se está leyendo directamente del PLC y será diferente dependiendo del modelo de autómatas que tengamos conectado.

Aquí podemos ver la fecha y la hora del módulo en el momento en que abrimos esta ventana. Lo que vemos en la parte inferior son los contadores de horas que tenemos activos. En el ejemplo vemos que no tenemos ninguno funcionando. Los contadores de horas los podemos utilizar mediante SFC de sistema. No es un tema que se trate en este libro.

Podemos obtener más información de la CPU si vamos a la ficha llamada “Datos característicos”.

En esta ficha podemos ver los OB que tenemos disponibles para programar en la CPU que estamos utilizando. No todos los OB los tenemos disponibles en todos los modelos de CPU. También podemos ver las SFC y las SFB de que dispone nuestra CPU integradas. Este diálogo que tenemos abierto es **ONLINE**. Toda la información que vemos hace referencia a la CPU que tenemos conectada. Si no estamos conectados **ONLINE**, no podremos abrir este menú que estamos analizando.

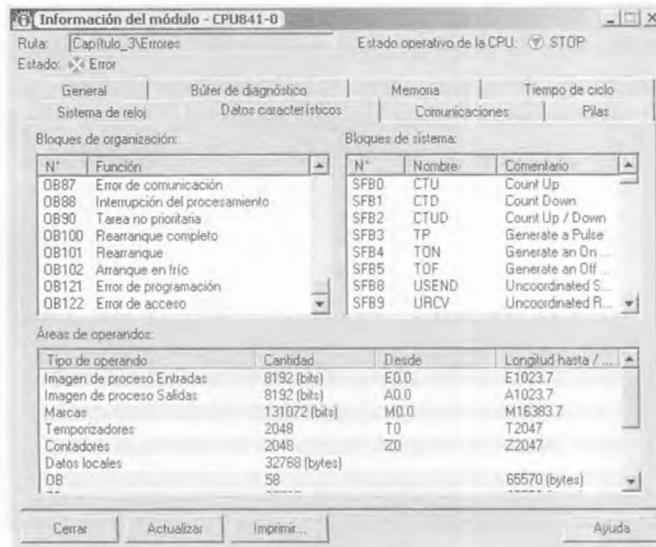


Fig. 151

En la parte inferior de la ventana vemos la cantidad de memoria que tenemos en la CPU para cada tipo de datos que podemos utilizar. En el ejemplo vemos que tenemos muchísima memoria disponible. La CPU que tenemos es una 841-0. Esta CPU no existe en el mercado como tal. Esta ficha se ha abierto utilizando el simulador de **STEP 7**. Es una aplicación adicional al Administrador de **SIMATIC**, con la cual se puede simular y probar los programas hechos **OFFLINE**. Al final de este manual hay una pequeña descripción de cómo utilizar este simulador. También vemos la lista de los OB y las SFC y SFB que tenemos disponibles en esta CPU. En la lista de OB podemos ver cómo se llama cada uno y a qué lo tenemos asociado. Por ejemplo, vemos que el OB 121 está asociado a un error de programación.

Si ahora vamos a la ficha de comunicaciones veremos lo siguiente:

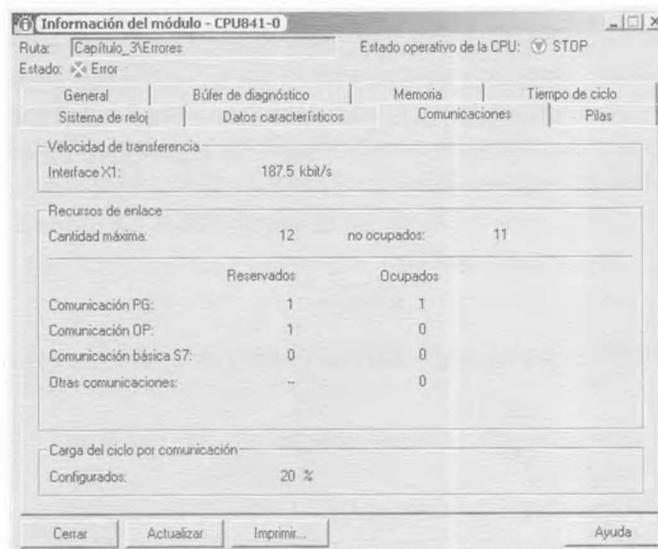


Fig. 152

Aquí podemos ver los enlaces de los que dispone la CPU que tenemos conectada y cuántos de ellos tenemos utilizados. Este concepto de enlace no es tratado en este manual. A modo de resumen, comentaremos que siempre vamos a tener un enlace reservado para la PG o el PC. En el ejemplo vemos que lo tenemos utilizado. Siempre quedará otro enlace reservado para poner un panel de visualización.

El resto de enlaces servirán para conectar al PLC otro tipo de equipos. En el ejemplo vemos que tenemos 11 enlaces libres (uno de ellos reservado para una OP). Podemos hacer 10 conexiones más, que ocupen recurso de enlace de la CPU. No todo lo que conectamos al PLC ocupa un enlace. Por ejemplo, todo lo que conectamos a través de periferia descentralizada no ocupará recursos de enlace.

En la primera ficha de información, vimos que el PLC había ido a buscar el OB121. Vamos a ver cómo podemos utilizar este OB. En el Administrador de **SIMATIC**, añadimos un OB121. Lo hacemos igual que hemos insertado el resto de bloques. Al insertar un bloque de organización, por defecto, nos dirá si queremos crear el OB 2. Nosotros escribimos en este caso, OB 121. Una vez tengamos el bloque creado, pinchamos sobre él con el interrogante que tenemos en la barra de herramientas superior. Primero pinchamos sobre el interrogante y con él seleccionado pinchamos sobre el OB 121. Entonces obtendremos la información del bloque OB 121. Veremos cómo se llama, la función que realiza y lo que necesitamos para programarlo.

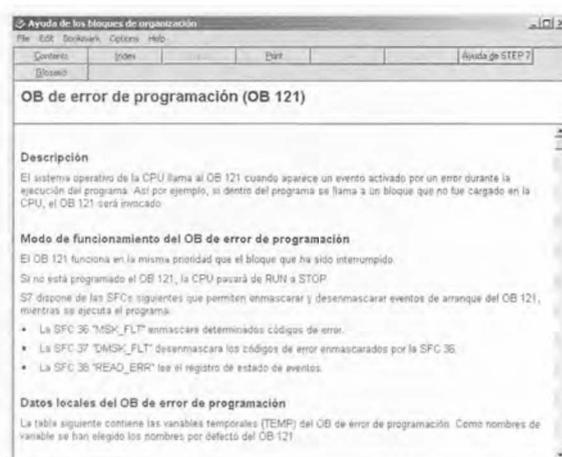


Fig. 153

En esta ayuda podemos ver como se llama el bloque, cómo funciona y los datos locales de que dispone. Los datos locales son las variables temporales que vimos en las tablas de las FC. Si abrimos la tabla, veremos que los OB no tienen variables de entrada ni de salida. Esto es porque nunca los llamaremos desde el programa y por tanto nunca tendremos la posibilidad de darles valores. En cambio, las variables temporales sí que tienen unos valores dentro del bloque y podemos utilizarlos si nos interesa. En la ayuda vemos de qué datos disponemos y lo que significa cada uno de ellos.

Vamos a entrar en el OB 121 y vamos a escribir el siguiente código de programa:

OB 121

SET

S A 4.7

La instrucción **SET** nos pone el RLO a 1. Por tanto siempre que el programa lea estas instrucciones, encenderá la salida A 4.7. La instrucción S es condicional. Si no escribimos **SET** como primera instrucción y sólo escribimos S A 4.7, la salida se encenderá si tenemos el RLO a 1. Y no se encenderá si lo tenemos a 0. Nosotros no sabemos cuándo se va a leer el OB 121. Normalmente no sabemos cuándo se va a producir un error de programación. En este caso yo quiero que siempre que haya un error de este tipo, me avise encendiendo la salida A 4.7. Para ello lo hago como se ha escrito en el ejemplo.

Transferimos este OB 121 y volvemos a probar el programa. Para ello tenemos que llevar la CPU a **RUN** mediante el selector manual. También lo podemos hacer a través del menú “sistema destino → estado operativo”. Mientras no tengamos ninguna entrada activa, no ocurrirá nada. Cuando activemos E0.0 y E0.7, se encenderá la luz de fallo de sistema SF y se encenderá la salida A 4.7 porque se lo hemos indicado nosotros en el OB 121. El PLC no se habrá ido a **STOP**. Si ahora vamos a la ficha de información del módulo, obtendremos la siguiente información:

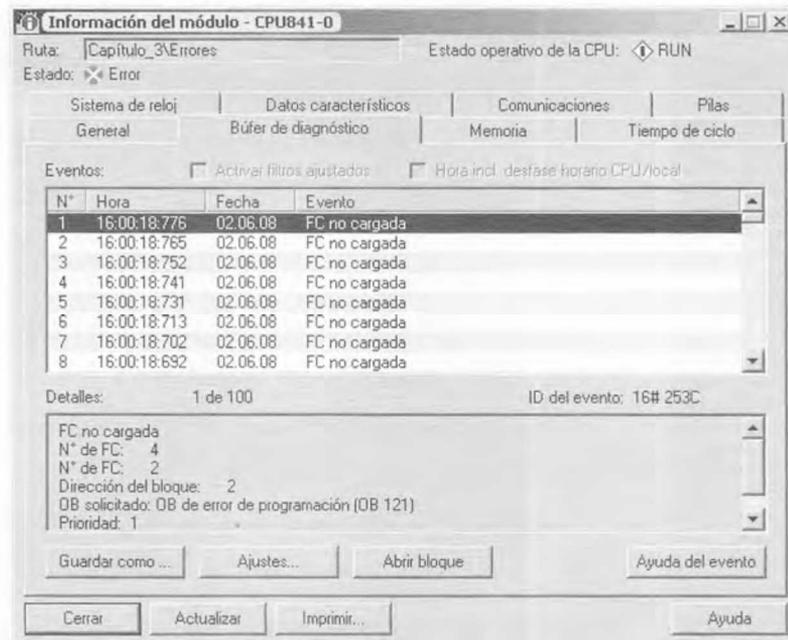


Fig. 154

Lo primero que vemos en la parte superior de la ficha, es que tenemos la CPU con fallo de sistema, pero su estado es **RUN**.

Además, vemos que cada ciclo de scan encuentra un fallo “FC no cargada”. En cambio no se va a **STOP**. En el OB 121, le estamos diciendo que si encuentra un error de programación, que encienda la A4.7 y que continúe con el resto de instrucciones. Nosotros en este OB 121 podemos programar lo que queramos. Que salte a una FC especial, que se vaya la CPU a **STOP**, que guarde una serie de datos antes de parar, que avise con una bocina, etcétera.

Veamos qué pasa si borramos todo lo que contiene el OB 121. Es decir borramos las instrucciones que habíamos escrito y enviamos el OB 121 sin instrucciones.

Pasamos la CPU a **STOP** y nuevamente a **RUN** con el selector manual, después de haber transferido los bloques con este nuevo OB 121.

Observaremos que la CPU nos da fallo de sistema pero no se va a **STOP**. Nosotros en este caso, le estamos diciendo, que ante un fallo de programación no haga nada. No es lo mismo decirle que no haga nada (OB 121 vacío) que no decirle nada (sin OB 121). En este segundo caso ya hemos visto que la CPU se va a **STOP**.

4.2 Otras OB y datos característicos de las SFC

Ejercicio 2: Otras OB y datos característicos de las SFC

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Significado de los OB

Los OB son la comunicación entre el sistema operativo de la CPU y el programa de usuario. Con los OB podemos:

- Arrancar la CPU.
- Ejecución cíclica o intermitente temporalmente.
- Ejecución a determinadas horas del día.
- Ejecución después de transcurrir un tiempo preestablecido.
- Ejecución al producirse errores.
- Ejecución al producirse alarmas de proceso.

En el ejercicio anterior, hemos podido observar los OB que podemos programar con la CPU que tenemos conectada. Si queremos ver todos los OB que tenemos disponibles en **STEP 7**, podemos acceder a través de una librería y hacer la consulta. No obstante cada CPU sólo admite una lista de OB predeterminada que es la que hemos visto en la conexión **ONLINE** del ejercicio anterior. Veamos lo que podemos consultar en las librerías.

Para abrir una librería, vamos al Administrador de **SIMATIC** y pulsamos el botón de abrir como si fuésemos a abrir un proyecto ya existente. Una vez tengamos abierta la ventana de “**abrir proyecto**”, seleccionamos la ficha de “**librerías**”.

Seleccionamos las librerías estándar tal y como vemos en la figura siguiente:

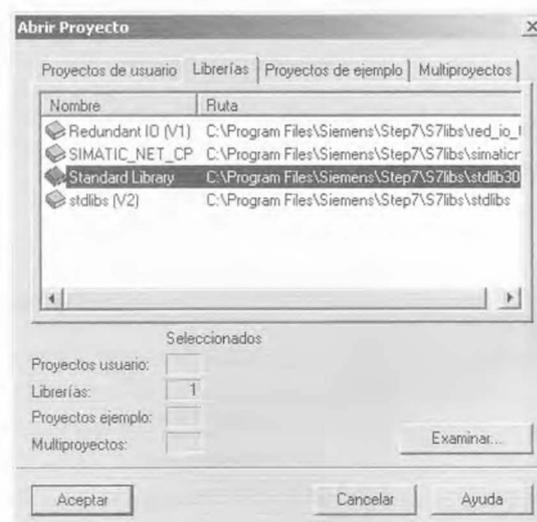


Fig. 155

Una vez abierta la librería, observaremos que es como un proyecto de usuario. Tiene el mismo aspecto que los proyectos ejemplo que venimos haciendo a lo largo de este manual sin definir *hardware*. Esto son bloques que vienen ya programados y protegidos. Podemos copiarlos o arrastrarlos a nuestro proyecto y utilizarlos. Todos disponen de su correspondiente ayuda en la que se explica cómo funcionan y los parámetros que nos van a pedir en caso de que existan.

Si dentro de la librería nos vamos a la carpeta de **“Organization Blocks”** veremos el listado de todos los OB que se podrían programar en **STEP 7**.

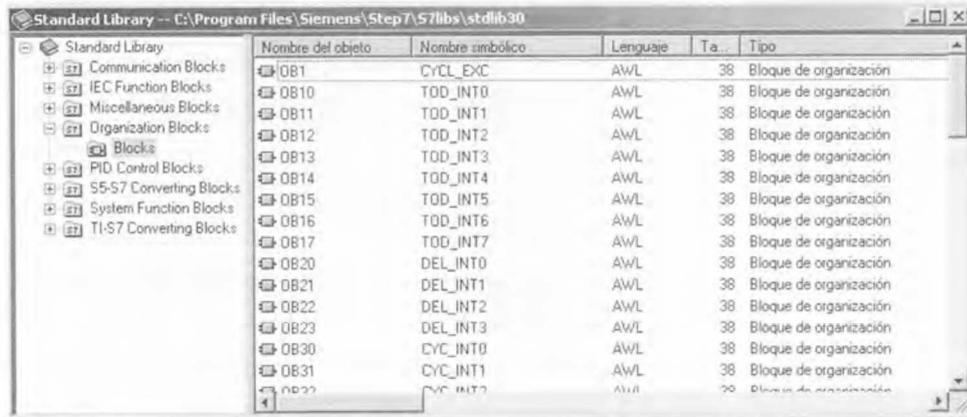


Fig. 156

Hagamos un resumen de todos estos OB:

RELACIÓN DE OB

OB1	
OB10.....OB17	Alarmas horarias
OB20.....OB23	Alarmas de retardo (SFC 32)
OB30.....OB38	Alarmas cíclicas
OB40.....OB47	Alarma de proceso
OB80	Error de tiempo (SFC 43 redispara tiempo)
OB81	Fallo de alimentación
OB82	Alarma de diagnóstico
OB83	Alarma insertar/extraer módulos
OB84	Avería CPU
OB85	Error de ejecución
OB86	Fallo de DP o en bastidor
OB87	Error de comunicación
OB100	Rearranque completo
OB101	Rearranque
OB121	Error de programación
OB122	Error acceso a la periferia

Recuerda . . .

No todos los OB que encontramos en la lista de bloques están disponibles en todas las CPU. Esta funcionalidad depende del PLC que estemos utilizando.

Comentarios sobre algunos OB.

OB 1: Programa cíclico.

Es el único bloque del CPU que es cíclico. Cuando acaba de leer las instrucciones que lo componen, vuelve a empezar por el principio.

Cuando termina el OB 1 envía los datos globales. Esto es un dato a tener en cuenta si se utiliza la comunicación por datos globales. Existe un ejercicio ejemplo hacia el final de este manual.

S7 ofrece supervisión del tiempo de ciclo máximo. Es lo que se llama ciclo de *scan*. Es un tiempo que nosotros parametrizamos como máximo. Mientras no nos de un error de tiempo, nos aseguramos que en menos del tiempo preestablecido, el programa se lee de arriba abajo.

ALARMAS HORARIAS

Se activan llamando a la SFC 30 o por *hardware* directamente. Si no están activadas la CPU no las lee.

Se programan con la SFC 28 o desde el *hardware* más el bloque correspondiente.

Cuando se ejecutan una vez se anulan. También se pueden ejecutar periódicamente si así lo programamos.

Si la CPU realiza un rearranque completo, cada OB de alarma horaria configurado mediante una SFC adopta otra vez la configuración ajustada con **STEP 7**.

Los ejercicios 7 y 8 de este manual están dedicados a este tipo de alarmas.

OB DE ALARMA DE RETARDO

Arrancan mediante una llamada a la SFC 32. Arrancan después de un tiempo preestablecido.

Puede anularse una alarma que todavía no ha sido arrancada (SFC 33).

Un rearranque completo borra cualquier evento de arranque de un OB de alarma de retardo.

El ejercicio 9 de este manual está dedicado a este tipo de alarmas.

OB DE ALARMA CÍCLICA

Tienen un valor de tiempo prefijado.

No puede durar más tiempo la ejecución del OB que el tiempo de volver a arrancar.

El ejercicio 6 de este manual está dedicado a este tipo de alarmas.

OB DE ALARMA DE PROCESO

Dentro de las alarmas de proceso, las prioridades se establecen con **STEP 7**. La activación se parametriza con **STEP 7**. No se pueden ejecutar dos a la vez.

Si el evento ocurre en otro canal del mismo módulo, no puede activarse momentáneamente ninguna alarma de proceso. En **S7 300** se pierden. En **S7 400** no se pierden. Se ejecutan cuando acaban.

OB DE ALARMA DE MULTIPROCESAMIENTO

En caso de operación en modo multiprocesador, la alarma de multiprocesamiento permite que las CPU asociadas puedan reaccionar de forma sincronizada a un evento. Al contrario de las alarmas de proceso - que sólo pueden ser desencadenadas por módulo de señales - la alarma de multiprocesamiento sólo puede ser emitida exclusivamente por las CPU.

Se activa llamando a la SFC 35.

En este manual no tratamos este tema.

OB DE ERROR DE TIEMPO

Si no es programado, la CPU pasa a **STOP** cuando se produce un error de tiempo.

Si en un mismo ciclo se llama dos veces al OB 80 debido a la superación del tiempo de ciclo, la CPU pasa a **STOP**. Es posible evitar esto llamando a la SFC 43 en un lugar adecuado.

Esta SFC se utiliza en el ejercicio de este manual dedicado a los errores por tiempo de ciclo.

OB 81. FALLO DE ALIMENTACIÓN

Sólo está permitida en el **S7 400**. Si no está programada y ocurre un fallo de alimentación, la CPU pasa a **STOP**.

RELACION DE SFC Y SFB

Las SFC de que dispone nuestra CPU, las podemos ver desde el Administrador de **SIMATIC**, conectándonos **ONLINE** tal y como vimos en el capítulo 1 de este manual. En la carpeta de bloques de **OFFLINE**, veremos los bloques que nosotros hemos programado. En la carpeta de bloques de **ONLINE**, veremos lo que nosotros hayamos transferido a la CPU y todas las SFC y SFB que tiene la CPU. Cada modelo tiene incluidas unas funciones y no podremos transferir más. Aunque las obtengamos de otras CPU o de librerías, no podríamos transferirlas a una CPU que no las trae de fábrica.

Si queremos consultar todas las SFC y SFB que existen en **S7**, podemos hacerlo a través de la librería igual que hicimos con los OB. Abrimos de nuevo las librerías estándar y vamos a ver los bloques de la carpeta **“system function blocks”**.

Nombre del objeto	Nombre simbólico	Lenguaje	Tama.	Tipo	Verzi.	Nombre (encabezado)
SFB65	SERVE_RK	AWL	--	SFB	1.0	SERVE_RK
SFB75	SALRM	AWL	--	SFB	1.0	SALRM
SFB81	RD_DPAR	AWL	--	SFB	1.0	RD_DPAR
SFC0	SET_CLK	AWL	--	SFC	1.0	SET_CLK
SFC1	READ_CLK	AWL	--	SFC	1.0	READ_CLK
SFC2	SET_RTM	AWL	--	SFC	1.0	SET_RTM
SFC3	CTRL_RTM	AWL	--	SFC	1.0	CTRL_RTM
SFC4	READ_RTM	AWL	--	SFC	1.0	READ_RTM
SFC5	GADR_LGC	AWL	--	SFC	1.0	GADR_LGC
SFC6	RD_SINFO	AWL	--	SFC	1.0	RD_SINFO
SFC7	DP_PRAL	AWL	--	SFC	1.0	DP_PRAL
SFC9	EN_MSG	AWL	--	SFC	1.0	EN_MSG

Fig. 157

Recuerda . . .

La cantidad de funciones de sistema que tenemos disponibles depende de la CPU que estemos utilizando. Desde la ventana de ONLINE siempre podremos obtener todas las funciones de sistema incluidas en nuestra CPU.

RESUMEN DE LAS SFC MÁS IMPORTANTES**FUNCIONES DE COPIA CON BLOQUES**

- Con **SFC 20** podemos copiar. Se copia el contenido de un área de memoria en otra.

Con esto NO se pueden copiar:

- FB, SFB, FC, SFC, OB, SDB.
- Contadores.
- Temporizadores.
- Áreas de memoria de la periferia.

Copia lo que cabe tanto si el campo destino es mayor o menor que el campo fuente. No existen informaciones de error específicas.

- **SFC 21:** Es posible inicializar un área de memoria (campo destino) con el contenido de otra área de memoria (campo fuente). La SFC copia en el campo de destino indicado, el contenido hasta que el área de memoria esté escrita por completo.

La escritura se realiza por el orden sucesivo de direcciones ascendentes en el área de memoria.

- **SFC 22:** Con esta función podemos crear un bloque de datos. Se hace con el parámetro "CREAT_DB". No contiene valores inicializados. La longitud de los DB debe ser un número par.

- **SFC 23:** Con esta función podemos borrar un bloque de datos. "DEL_DB".

- Podemos comprobar un bloque de datos con la SFC 24 "TEST_DB".

- Podemos comprimir la memoria de usuario con la SFC 25 "COMPRESS".

Al borrar y recargar repetidamente bloques, pueden surgir huecos, tanto en la memoria de carga como también en la memoria interna, que reducen el área de memoria aprovechable.

Con esta función podemos aprovechar el espacio. No hay control sobre si la compresión se ha realizado correctamente. Se puede controlar llamando a la **SFC 25** cíclicamente.

- Transferir valor de sustitución a ACU 1 con la **SFC 44** "REPL_VAL".

Con la SFC 44 "REPL_VAL", se transfiere un valor al ACU 1 del nivel de programa causante del error.

La **SFC 44** sólo debe ser llamada en el OB de error síncrono (OB 121, OB 122)

Puede continuar el programa en caso de error. Utiliza el nuevo valor que hay en el ACU 1.

SFC PARA CONTROL DE PROGRAMA

- La **SFC 43** redispara el tiempo de vigilancia.

No ofrece informaciones de error. En el ejercicio 4 de este manual hay un ejemplo en el que se utiliza esta SFC y se explica su funcionamiento.

- La **SFC 46** pone la CPU en STOP.

No ofrece informaciones de error.

- La **SFC 47** permite programar retardos o tiempos de espera en el programa de usuario.

Esta función se puede interrumpir con un OB de mayor prioridad.

No ofrece informaciones de error.

- La **SFC 35** dispara la alarma de multiprocesamiento. Esto conduce el arranque sincronizado del OB 60 en todas las CPU asociadas.

El parámetro de entrada JOB permite identificar la causa definida por el usuario para la alarma.

A la SFC 35 puede llamarse en cualquier punto del programa de usuario.

SFC PARA GESTIÓN DEL RELOJ

- **SFC 0** "SET_CLK" para ajustar la hora. Con esto se ajusta la fecha y la hora de la CPU. Si el reloj es un maestro, la CPU arranca también la sincronización de la hora al llamar a la SFC 0. Los intervalos de sincronización se ajustan con STEP 7.

La fecha y la hora se indican con el tipo de datos DT. Ej: DT#1998-01-15-10:30:30.

Se ha de tener en cuenta que el tipo de datos DT debe ser generado previamente con la FC3 de librerías, "D_TOD_DT" antes de asignarlo al parámetro de entrada.

- **SFC 1** "READ_CLK" lee la hora. Se obtienen la fecha y hora actuales.

No ofrece informaciones de error específicas.

- **SFC 48**: Sincronización de relojes esclavos. Transmite la fecha y la hora desde el reloj maestro de un segmento de *bus* a todos los relojes esclavos de este mismo segmento de *bus*.

SFC PARA GESTIÓN DEL CONTADOR DE HORAS DE FUNCIONAMIENTO

Contador de horas de funcionamiento: Cuenta las horas de funcionamiento de la CPU.

- Ajustar el contador de horas de funcionamiento con la **SFC 2**. Con esto se ajusta el contador de horas de funcionamiento de la CPU a un valor preestablecido. Es posible ajustar una cantidad específica de contadores para cada CPU.

- **SFC 3** arranca y para el contador de horas de funcionamiento.

- **SFC 4** lee el contador de horas de funcionamiento. La SFC 4 suministra como datos de salida, la cantidad actual de horas de funcionamiento y el estado del contador, es decir, parado o contando.

- **SFC 64** lee el cronómetro del sistema de la CPU. Si se sobrepasa el cronómetro del sistema, se comienza a contar desde cero.

SFC PARA TRANSFERIR REGISTROS

- Escribir y leer registros:

Existen zonas de memoria en las que sólo se puede escribir, o sólo se puede leer.

- **SFC 35** transfiere el registro RECORD al módulo direccionado.

- **SFC 56** transfiere el registro con el número RECNUM, desde el correspondiente SDB al módulo direccionado. Carece de significado si se trata de un registro estático o dinámico.

- Parametrizar el módulo con la **SFC 57**.

Con la SFC 57 se transfieren todos los registros de un módulo que han sido configurados con **STEP 7** en el correspondiente SDB, al módulo.

Carece de sentido si se trata de un registro estático o dinámico.

- **SFC 58** se transfiere el registro RECORD al módulo direccionado.
- **SFC 59** lee el registro con el número RECNUM del módulo direccionado.

SFC PARA GESTIÓN DE ALARMAS HORARIAS

Una alarma horaria es la causa de la llamada controlada por tiempo de un OB de alarma horaria. La alarma horaria se puede parametrizar con **STEP 7** y activar en el programa de usuario.

- **SFC 28** Ajustar alarmas horarias.
- **SFC 29** Anular alarmas horarias.
- **SFC 30** Activar alarmas horarias.
- **SFC 31** Consultar alarmas horarias.

4.3 Instrucción LOOP

Ejercicio 3: Instrucción LOOP

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Introducción a la instrucción.

Vamos a ver una nueva instrucción.

Es la instrucción **LOOP**. Esta instrucción permite hacer un pequeño bucle dentro de un bloque. Permite repetir una serie de instrucciones varias veces antes de terminar el bloque.

Veamos un ejemplo. Las instrucciones necesarias para utilizar la instrucción son las siguientes:

```

L            20            //Cargar un entero
META: T    MB            0            //Almacenarlo si es necesario
.....
.....
L            MB            0            //Recuperar la cuenta
LOOP        META            //Instrucción LOOP
.....
    
```

La primera vez que se lea LOOP META, el acumulador 1 valdrá 8. Suponemos que ha sumado $0 + 8$ y se habrá quedado con el resultado. Al leer LOOP, dirá $8 - 1 = 7$. Y se preguntará ¿7 es distinto de 0? Sí. Por lo tanto saltará a META. La segunda vez que lea el LOOP, el acumulador 1 valdrá 16. Habrá sumado $8 + 8$. Como 16 es diferente de 0 saltará de nuevo a la META. Como podemos comprobar, esto no hace 5 bucles como queríamos. Intentará hacer infinitos bucles y la CPU se irá a STOP por ciclo de scan excesivo.

Veamos otro ejemplo:

```

L      5
META: U  E      0.0
        =      A      4.0
        LOOP META

```

En este caso (aunque no conseguimos nada especial con el LOOP) sí que se ejecutaría 5 veces. Antes de leer la instrucción LOOP, lo último que se cargó en el acumulador fue un 5. Cada vez que llega a LOOP META, se decrementa en 1 y salta a META.

Si en lo que vamos a introducir dentro del bucle, tenemos que modificar el valor del acumulador, antes de ejecutar ninguna instrucción en el bucle transferimos lo que hay en el acumulador a un *byte* de marcas (de datos locales o de un DB o donde nos interese guardarlo). A continuación trabajamos con el acumulador. Antes de la instrucción LOOP, cargamos lo que hay en el *byte* de marcas auxiliar utilizado. La propia instrucción LOOP decrementa este número en uno. A continuación cargamos de nuevo este número en el *byte* auxiliar.

Si dentro del bucle no vamos a utilizar el acumulador, no son necesarias las instrucciones de carga y transferencia.

Veamos un ejemplo en el que comprobemos cuantas veces se ejecuta el bucle:

```

L      5
META: T  MB      0
L      5
L      MB      0
==|
S      A      4.0
L      4
L      MB      0
==|
S      A      4.1
L      3
L      MB      0
==|
S      A      4.2
L      2
L      MB      0
==|
S      A      4.3
L      1
L      MB      0
==|
S      A      4.4
L      0
L      MB      0
==|
S      A      4.5
L      MB      0
LOOP          META

```

Con esto veremos que se encienden las salidas 4.0, 4.1, 4.2 4.3 y 4.4. La 4.5 no se enciende. Dentro del bucle la MB 0 nunca vale cero. Cuando vale cero ya no se ejecuta el bucle.

Con esto simplemente comprobamos que se ha ejecutado el bucle 5 veces.

Esta instrucción será útil utilizarla en combinación con direccionamientos indirectos para leer valores de tablas, o escribir en ellas por ejemplo. Hay que tener mucho cuidado de no hacer bucles infinitos que no permitan a la CPU terminar su ciclo de *scan*.

4.4 Error de tiempo, OB 80 y SFC 43

Ejercicio 4: Error de tiempo, OB 80 y SFC 43



DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Instrucción LOOP

Vamos a hacer un programa con el cual la CPU se vaya a **STOP** por un error de tiempo. Si en lugar de hacer 4 o 5 bucles con la instrucción LOOP como hicimos en un ejemplo anterior, le decimos que ejecute 30.000 bucles, seguramente la CPU se irá a **STOP**. No podemos decir que ejecute más de 32.767 bucles puesto que lo almacena en un entero de 16 *bits* y este es el valor máximo que admite. Nosotros podemos escribir **L 60.000**, pero no quiere decir que ejecute sesenta mil bucles. Ejecutará la cantidad que corresponda a la parte baja en binario de 60.000. Es decir, si escribimos 60.000 en binario, ocupará más de 16 *bits*. Si quitamos los *bits* que sobran de la parte izquierda y nos quedamos con los 16 *bits* de la parte derecha, esta cantidad será los bucles que ejecute la CPU. El tiempo que tarde en irse a **STOP** o la cantidad de bucles necesarios para ello, dependerá del tiempo de ciclo que tengamos predeterminado y de la velocidad de la CPU que estemos gastando. En ejemplos posteriores veremos como modificar los tiempos de ciclo. En principio por defecto, las CPU vienen con un tiempo de ciclo de *scan* de 150 ms.

Vamos a hacer el siguiente programa:

OB1

```

L      30000
META: U  E      0.0
        =      A      4.0
        LOOP      META
  
```

Como vemos este programa lo que hace es ejecutar 30.000 veces este bucle. Si ponemos un número suficientemente grande, no se podrá ejecutar el bloque completo dentro del tiempo definido como ciclo de *scan*. En cuanto esto ocurra, la CPU se irá a **STOP** por un error de tiempo.

Si una vez tengamos la CPU en **STOP** entramos en la información del módulo veremos la siguiente información:

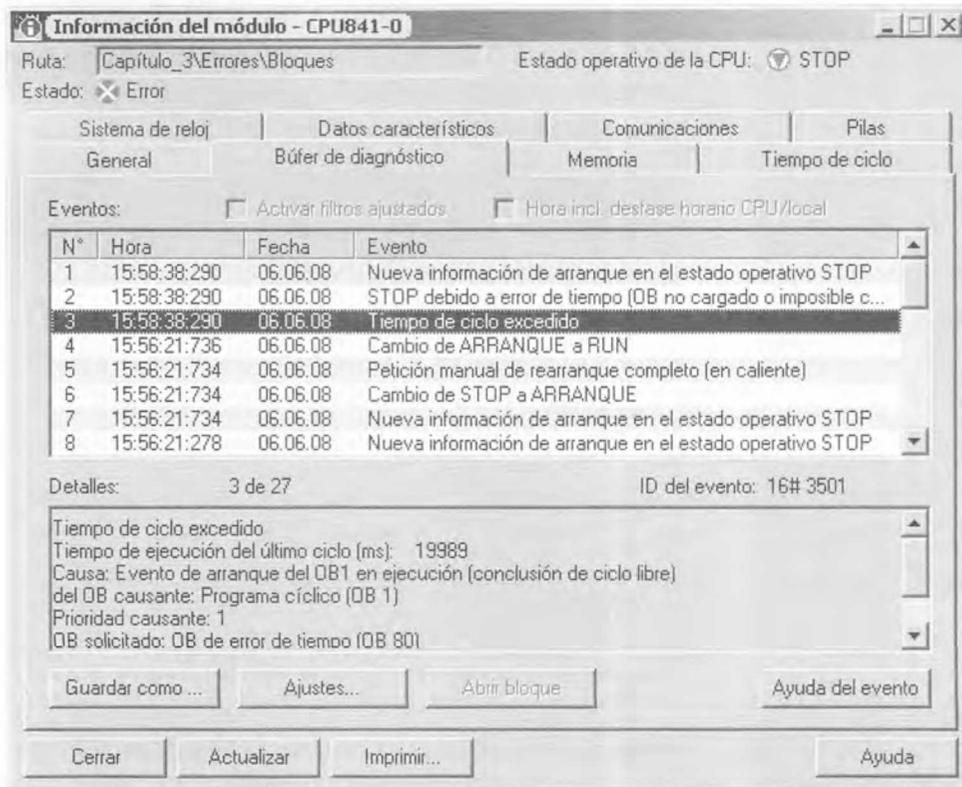


Fig. 158

En la ficha de "Búfer de diagnóstico" vemos que la CPU se ha ido a **STOP** por tiempo de ciclo excedido. En la parte inferior, nos dice que ha ido a buscar el OB 80 que es el OB asociado a este tipo de errores. Como no lo ha encontrado, se ha ido a **STOP**.

¿Qué podemos hacer para que la CPU no se vaya a **STOP**?

Podemos programar algo en el OB 80. De este modo, cuando ocurra un fallo de tiempo, en lugar de irse la CPU a **STOP**, lo que hará será ejecutar lo que ponga en este bloque.

Si no tenemos programado el OB 80, cuando ocurre un error de tiempo, el PLC va a leer lo que pone en el OB 80 y, como no lo encuentra, se va a **STOP**. Si vamos a ver la información del módulo, el primer error que veremos será "OB no cargada".

Se ha ido a **STOP** porque le faltaba un módulo. Si vamos a ver el segundo error veremos que es un error de tiempo.

Si programamos el OB 80, cuando ocurra un error de tiempo irá al OB 80. Como ahora sí que lo encontrará, la CPU no se irá a **STOP**. Ejecutará lo que diga el OB 80.

Para probarlo, vamos al Administrador de **SIMATIC** y nos creamos un OB 80. Podemos escribirle alguna instrucción o simplemente podemos dejarlo en blanco.

Probamos ahora el programa. Veremos que el autómata sigue dando un error pero no se va la CPU a **STOP**. Además podemos comprobar que sigue funcionando.

Vemos que tarda "mucho" en reaccionar. Es el tiempo real que tarda la CPU en ejecutar el OB 1. Sabemos que está tardando más de lo que tiene parametrizado como tiempo de ciclo de scan.

Tenemos una “solución” para que la CPU no nos de este fallo.

Dentro del bucle podemos hacer una llamada a la SFC 43. Esta función lo que hace es relanzar el tiempo de ciclo.

Para ver lo que hace la SFC 43, podemos ir al Administrador de **SIMATIC** y pinchar con el interrogante de ayuda encima de la SFC 43. Se abrirá una ventana en la que nos explica lo que hace la SFC 43.

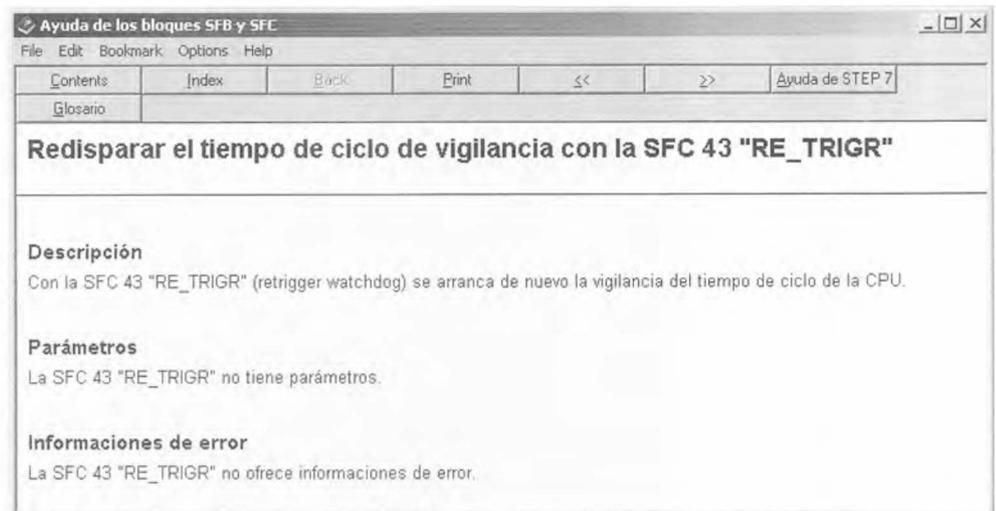


Fig. 159

Recuerda . . .

El tiempo de ciclo de scan de un PLC debe ser moderado. Hay que tener en cuenta que el funcionamiento de un programa de PLC se basa en el tiempo de ciclo de scan. Si alguien actúa y hay una emergencia en nuestra instalación, deberemos estar seguros de que la máquina no tardará más del tiempo de ciclo de scan en reaccionar. No deberemos utilizar tiempos excesivamente largos.

Vemos que lo que hace es relanzar el tiempo de ciclo. Es decir, cada vez que se lee la SFC 43, el tiempo de ciclo de *scan* vuelve a cero y empieza a correr de nuevo.

Esto que vamos a hacer en el ejemplo, no lo deberíamos hacer nunca dentro de una instrucción de bucle. Lo haremos a modo de ejemplo para entender bien el funcionamiento.

De este modo cada vez que se ejecuta el ciclo, tenemos más tiempo de ciclo de *scan*. Podemos dejar el PLC bloqueado y que se quede durante varios segundos sin terminar el OB1.

Si hacemos lo que hemos dicho, conseguiríamos que el OB1 tardara en ejecutarse mucho tiempo y que no diera ningún error y que no se fuera a **STOP**.

Esto es peligroso porque se está leyendo el bloque cada varios segundos. Si tenemos alarmas programadas u otras operaciones que tenga que ejecutar el PLC, sólo se dará cuenta de que lo tiene que ejecutar cada vez que lea la instrucción.

Puede darse el caso de que salte una alarma y hasta el cabo de unos segundos no reaccione el PLC. Imaginemos que la E0.0 que tenemos programada es una seta de emergencia y que la salida debe activar las seguridades de la máquina. Sería un peligro que no actuasen hasta el cabo de unos segundos.

En un programa de PLC deberemos asegurarnos que de cualquier modo el programa completo se ejecuta en una cantidad de milisegundos razonable. 200ms ya es un tiempo más que considerable para un programa de PLC. Lo normal es que en menos de 100ms se lea el programa completo por complejo que sea. Todo esto dependerá de la longitud del programa y de la velocidad de la CPU utilizada.

La SFC 43 se puede utilizar para alguna transferencia de datos masiva a un ordenador o algo que queramos hacer que sobrecarga la CPU. Deberíamos asegurarnos que tenemos un ciclo de *scan* algo más largo en el momento justo que necesitamos sobrecargar la CPU por alguna operación concreta. Nunca deberíamos tener ciclos de *scan* excesivamente largos en una máquina que funcione normalmente con operarios o peligros mecánicos o de cualquier otro tipo.

4.5 OB 100, 101

Retardo en el arranque

Ejercicio 5: OB 100, 101. Retardo en el arranque



DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Relación de OB.

Cuando la CPU realiza un re arranque completo, accede al OB 100 antes de leer el OB 1. El OB 100 es un bloque que se lee una sola vez al arrancar la CPU. Hasta que no haya un nuevo paso de **STOP** a **RUN**, no se volverá a leer.

Vamos a hacer una prueba de que esto es así.

Vamos a programar el OB 100. Vamos a decirle en el OB 100 que ponga unos valores a unas palabras de marcas.

Luego haremos un OB 1 que utilice estos datos de las palabras de marcas y posteriormente que los modifique. Veremos que cada vez que re arranquemos el autómata, toma los valores que hemos definido como iniciales.

OB100

```
L    10
T    MB    100
L    8
T    MB    101
```

```
OB    1
L    MW    100
T    AW    4
U    E    0.0
SPB  M001
BEA
```

```
M001: L    W#16#FFFF
      T    MW    100
      BE
```

Cada vez que activemos la E 0.0, estaremos cambiando el valor de la palabra de marcas 100. Se quedará con este valor hasta que re arranquemos el autómata.

Al re arranar tomará el valor que le estamos dando con el OB100.

Este OB se suele utilizar para inicializar bloques de datos.

En él cargamos todos los datos importantes de inicialización de proceso. Cada vez que arranquemos de nuevo, se tomarán los valores iniciales.

Si no hacemos esto, los DB se guardan los valores actuales. Si por cualquier causa en un DB hay datos que no son los iniciales, cada vez que transfiramos al autómeta, estamos transfiriendo los valores actuales.

El OB 101, sólo lo tenemos disponible en los S7 - 400.

Es similar al OB 100.

El OB 100 también es útil para retardar el arranque. El tiempo de ciclo de scan sólo afecta al OB 1. Nosotros podemos programar un bucle con un temporizador de 3 segundos dentro del OB 100, de manera que cuando ponemos la CPU en marcha, hasta los tres segundos no se empieza a leer el OB 1 y por consiguiente no se empieza a ejecutar el programa.

Esto nos puede ser útil en instalaciones en las que tengamos, por ejemplo, variadores de frecuencia que necesiten unos segundos para estar a punto y empezar el programa.

Solución en AWL

OB 100

```
M001: UN   M   0.0
      L    S5T#3S
      SE   T    1
      UN   T    1
      SPB  M001
      BE
```

A continuación se leería el OB 1 de forma cíclica.

4.6 Programación de alarmas cíclicas

Ejercicio 6: Programación de alarmas cíclicas

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Las alarmas cíclicas, son unos OB que se ejecutan cada cierto tiempo que nosotros parametrizamos. Son independientes del ciclo de programa. Puede que se ejecuten dos veces por ciclo de *scan* o una vez cada 100 ciclos. Dependen del tiempo que le ajustemos a la alarma y del tiempo que tarde el ciclo de *scan*. Además no siempre se ejecutarán en el mismo punto del programa. La CPU va ejecutando el programa que tiene en su memoria y va saltando a los bloques que le corresponde. Está donde esté, cuando se cumple el tiempo de la alarma, deja lo que esté haciendo y va a ejecutar el OB correspondiente. Cuando termina de ejecutar este OB, continúa por donde iba en el ciclo normal de programa.

Vamos a ver cómo podemos programar una alarma cíclica. El tiempo de ejecución de estos OB es algo que se modifica desde el *hardware* de la CPU. Por tanto necesitamos crear un nuevo proyecto en el que introduzcamos el *hardware* que tenemos conectado.

Una vez tengamos el proyecto creado, vamos a la ventana del hardware. Desde allí pinchamos con el botón derecho encima de la CPU. Entramos en propiedades del objeto. En cualquier otro lugar del **STEP 7** donde veamos el nombre de la CPU, también podemos pulsar con el botón derecho y acceder a las propiedades del objeto. Lo que ocurre es que accederemos en modo "sólo lectura". Si queremos modificar valores, deberemos acceder a esta ventana desde la definición del *hardware* de **OFFLINE**.

Pulsamos con el botón derecho sobre la CPU y seleccionamos: "Propiedades del objeto".

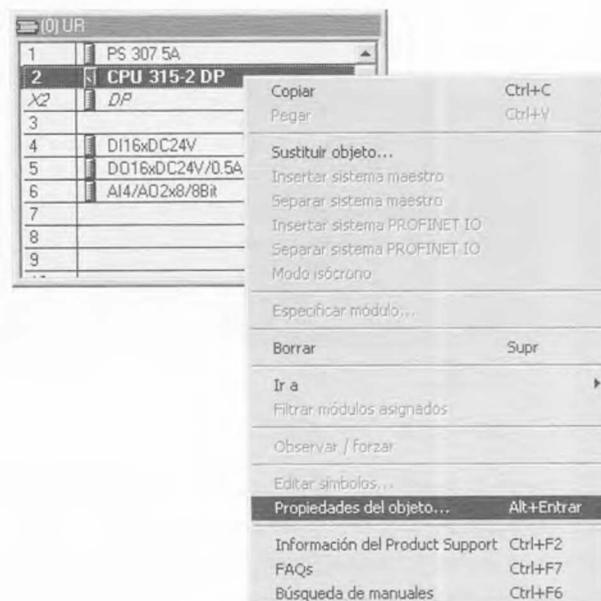


Fig. 160

Dentro de propiedades del objeto vamos a la ficha de alarmas cíclicas.

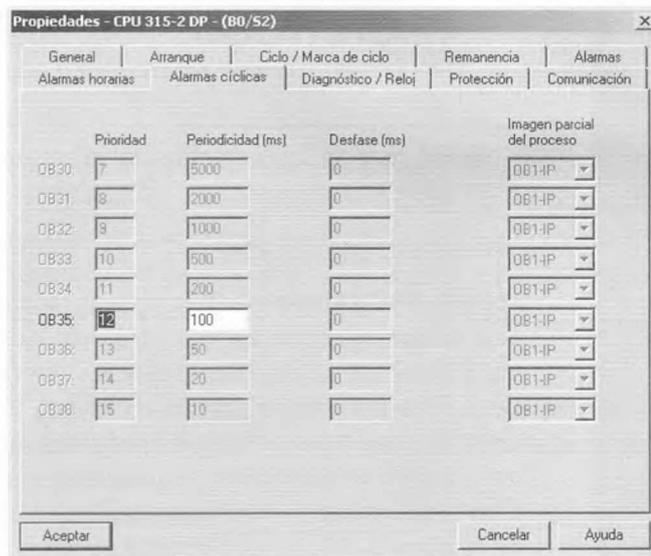


Fig. 161

En esta ficha podemos programar los tiempos de todas las alarmas cíclicas que tengamos disponibles en nuestra CPU. En este caso vemos que sólo tenemos una disponible y corresponde al OB 35. Por defecto tenemos 100ms. Vamos a cambiar este tiempo y vamos a poner 1000 ms. Es decir, 1 segundo. También podemos ver, aunque no modificar, la prioridad de este bloque. Vemos que tiene prioridad 12. La prioridad se utiliza en el caso en que la CPU se encuentre en el caso de tener que ejecutar dos bloques al mismo tiempo. Por ejemplo, si cuando se cumple el tiempo de la alarma, la CPU estaba leyendo una instrucción de salto a una FC ¿qué bloque se ejecutará primero? El que tenga mayor prioridad. El OB 1 tiene prioridad 1 y siempre es el último que se ejecuta.

Una vez tenemos modificadas estas propiedades de la CPU, tenemos que enviar el *hardware* de nuevo para que sean efectivas.

En este caso vamos a hacer un ejemplo que ejecute el OB 35 cada segundo.

El tiempo siempre se lo damos en milisegundos. Este tipo de alarmas sólo nos vale para tiempos cortos. Para tiempos más largos disponemos de las alarmas horarias.

Para programar la alarma, tenemos que ir al Administrador de **SIMATIC** y crear el OB 35. Lo creamos como cualquier otro bloque. Desde la ventana de bloques, pinchamos con el botón derecho del ratón, seleccionamos insertar nuevo objeto → bloque de organización. Por defecto el sistema nos ofrecerá crear el siguiente que tengamos libre. Es decir, si ya tenemos un OB 1, se nos ofrecerá crear el OB 2. Nosotros modificamos el bloque que queremos y escribimos OB 35.

Una vez dentro del bloque creamos el siguiente programa:

```

OB 35
UN   A   5.0
=    A   5.0
    
```

Una vez enviemos esto al PLC, veremos que tenemos un intermitente de 1 segundo. A este OB no hace falta llamarlo desde ningún sitio. Se ejecuta por tiempo.

Si queremos dejar de tener la alarma cíclica, no tenemos más que borrar el OB 35.

4.7 Programación de alarmas horarias por *hardware*

Ejercicio 7: Programación de alarmas horarias por *hardware* ✓

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Relación de OB.

Las alarmas horarias son similares a las cíclicas. Se utilizan para rangos de tiempo mayores. La programación es algo diferente como veremos en el presente ejercicio.

Para programar este tipo de alarmas, tenemos que hacerlo a través del *hardware* o por programa utilizando las SFC correspondientes. En este primer ejercicio veremos cómo hacerlo a través del *hardware*.

Para ello vamos a hacer un proyecto nuevo (o aprovechamos el del ejercicio anterior) y vamos a entrar en el *hardware*. Una vez lo tengamos definido, pinchamos Veremos que tenemos varias fichas. Vamos a la ficha de alarmas horarias.

Dependiendo de la CPU que tengamos, tendremos unos OB disponibles para programar alarmas. Por ejemplo, si tenemos una CPU de la serie 300, veremos que sólo tenemos disponible el OB10.

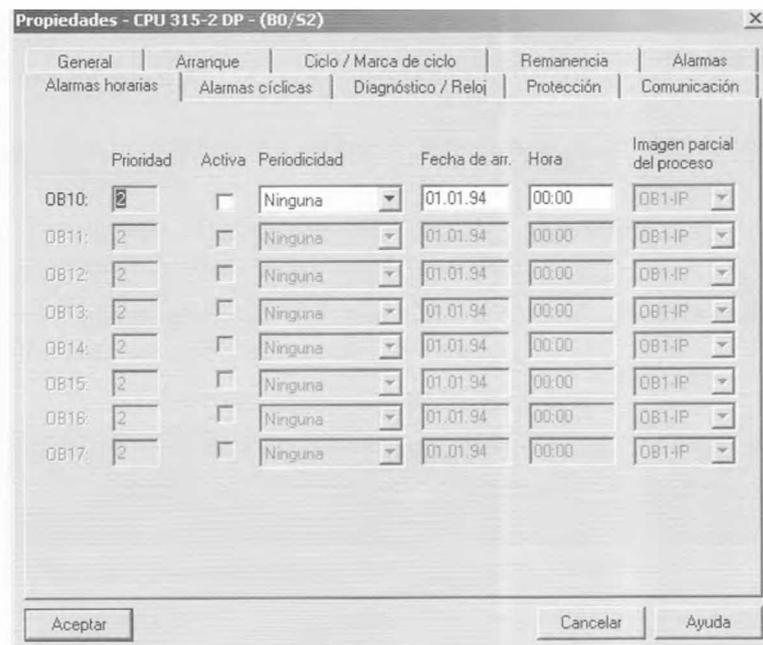


Fig. 162

Para programar este tipo de alarmas lo primero que tenemos que hacer es decir que la alarma está activa. Para ello tenemos que seleccionar la casilla de activación de la alarma que queremos. Además tendremos que decirle cada cuanto tiempo queremos que se produzca y desde cuando queremos que se active.

Esto significa que cada cierto tiempo, se va a acceder al OB 10. Se ejecutará lo que allí diga y el programa seguirá luego por donde iba. El funcionamiento es como el de las alarmas cíclicas.

Luego tendremos que programar el OB10. Lo que no habrá que hacer, será programar una llamada al OB 10. A los OB no se les llama.

Vamos a ver un ejemplo de este tipo de alarmas.

Vamos a decirle que se ejecute el OB 10 cada minuto desde la hora actual. Esto lo haríamos así:

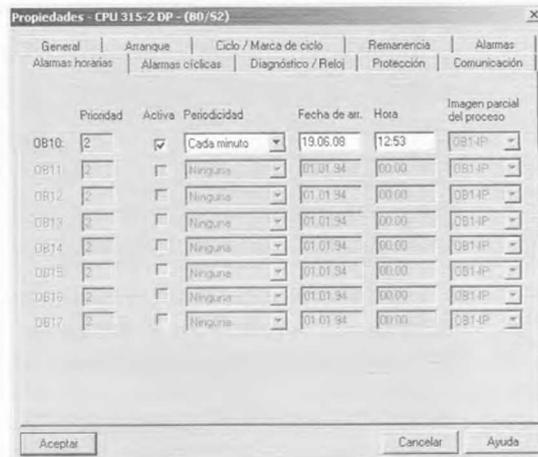


Fig. 163

En la OB 10 vamos a programar:

Vamos a programar a modo de ejemplo el siguiente OB 10. Antes tenemos que transferir el hardware a la CPU para que los cambios sean efectivos.

OB10

```
L    W#16#FFFF
T    AW    4
BE
```

Ahora tenemos que programar el OB1.

OB1

```
U    A    4.7
L    S5T#10S
SE   T    1
U    T    1
R    A    4.7
R    A    4.6
R    A    4.5
R    A    4.0
U    E    0.0
=    A    4.0
```

De este modo podremos comprobar que cada minuto se ejecuta el OB 10, y que mientras tanto, se está ejecutando el resto del programa.

El OB10, se ejecutará cada minuto desde la fecha y hora que le dimos de comienzo. Para que se active la alarma, el reloj de la CPU debe pasar por la fecha y hora de inicio. Si el reloj de la CPU tiene una fecha y hora posterior a la de la activación, nunca se activará y por tanto no funcionará la alarma.

4.8 Programación de alarmas horarias por software

Ejercicio 8: Programación de alarmas horarias por software

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Relación de OB.

Esto mismo también lo podemos conseguir por *software* en lugar de modificar el *hardware* de la CPU. Esto puede ser interesante si queremos tener la alarma horaria condicionada a algún proceso. Podemos querer alarma si se cumplen unas condiciones. O podemos querer que se ejecute cada minuto si ocurre una cosa o cada día si ocurre otra. Esto no podríamos conseguirlo si hacemos la programación por *hardware* como en el ejemplo anterior.

Para hacer la programación por *software*, lo haremos utilizando las funciones de sistema SFC 28 y SFC 30.

Vamos a ir realizando el programa poco a poco y analizaremos en detalle cada uno de los parámetros que nos piden las funciones.

Para activar la alarma deberemos llamar a la SFC 30. Veamos primero la ayuda de esta función para ver qué parámetros nos va a pedir.

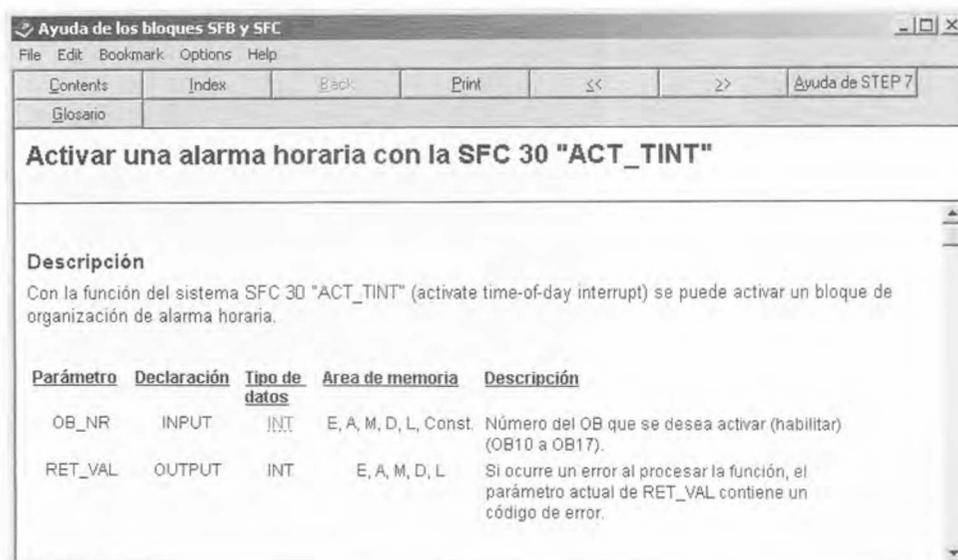


Fig. 164

Aquí podemos ver que al llamar a la SFC 30 se nos va a pedir el número de OB que queremos activar en formato entero. Para nosotros será "10".

También se nos va a pedir una variable de tipo entero en la que guardar el código de error en caso de existir. Para ello podemos generar una variable temporal llamada "ERROR_ACTIVACION" de tipo entero. En la ayuda podemos ver el significado de los códigos de error.

Para decirle cuando queremos que se active la alarma y a partir de cuando queremos que lo haga, tenemos que hacer una llamada a la SFC 28. Vamos a ver la ayuda de esta función y a analizar los parámetros que nos pide.

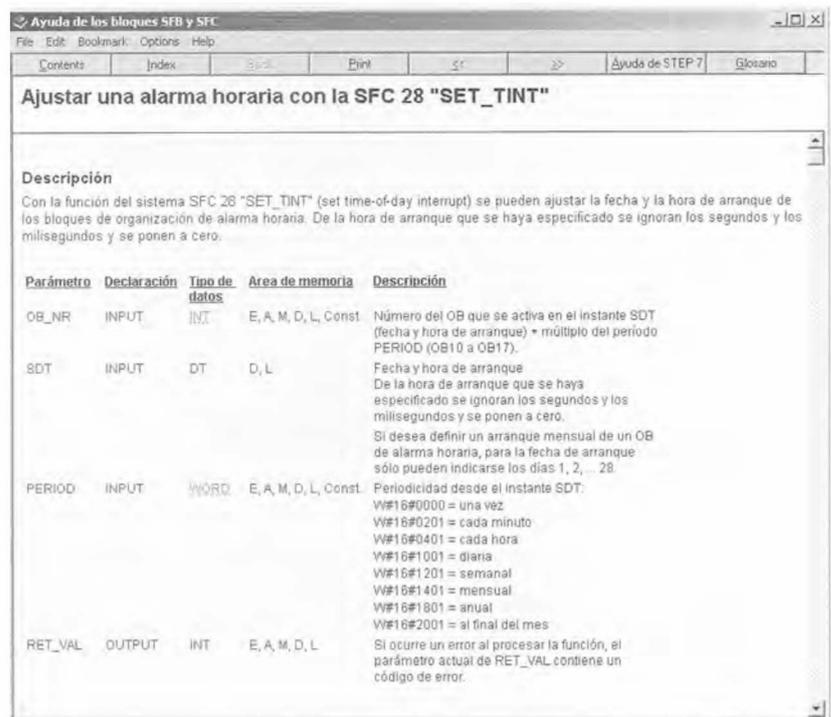


Fig. 165

Como primer parámetro nos pide el número de OB al que hacemos referencia en formato entero. Aquí escribiremos 10. Como segundo parámetro nos pide la fecha y hora de arranque. Hasta ahora en este manual no hemos visto este tipo de formato. Si hacemos la llamada a la función, podemos situarnos con el ratón en el parámetro correspondiente (como vemos en la figura) y desde esta posición pulsamos F1.

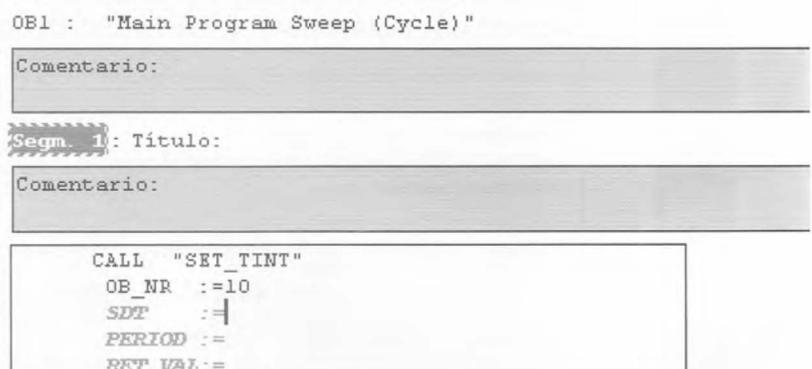


Fig. 166

Siempre que se nos esté solicitando un parámetro y pulsemos F1, nos saldrá la ayuda del tipo de datos que se requiere en ese momento. En este caso obtendríamos la siguiente ayuda:

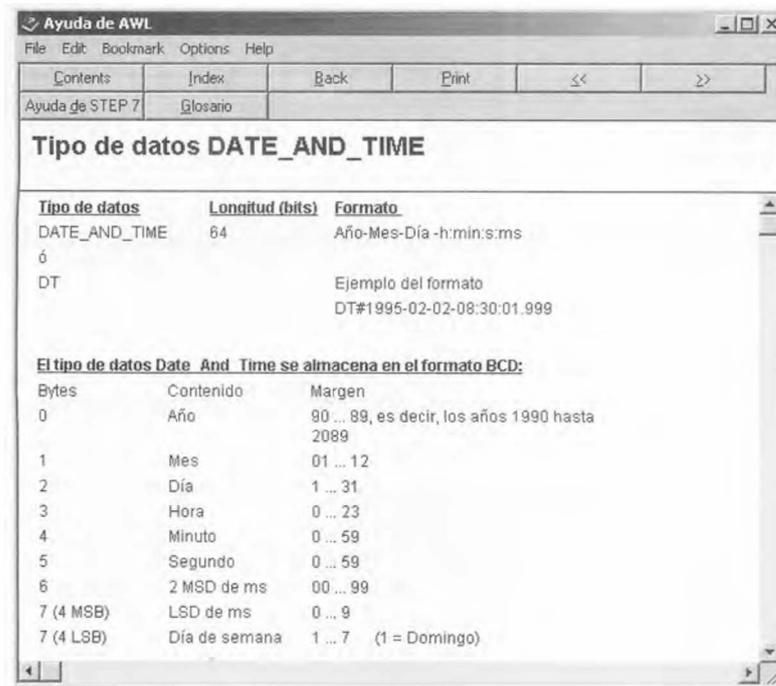


Fig. 167

Aquí podemos ver que el formato *date_and_time* es un formato de 64 bits. Veamos un ejemplo de cómo escribir un dato. Y además se nos explica cada *byte / bit* lo que significa.

Si en la llamada que estábamos haciendo intentamos introducir una fecha y hora, veremos que el sistema no lo admite.

```
CALL "SET_TINT"
OB_NR :=35
SDT :=DT#2008-06-19-09:00:00.00
PERIOD :=
RET_VAL :=
```

Fig. 168

Nosotros no podemos manejar formatos de fecha y hora. Este formato es de 64 bits. No podemos moverlo a través del acumulador. En la ayuda de la SFC ya se nos indica que el tipo de datos que podemos introducir en este parámetro es D o L (de bloques de datos o variables locales). Para poder utilizar variables locales, tenemos que estar dentro de una función. Vamos a hacer la llamada desde una FC. En la tabla de la FC tendremos que definir una variable de tipo DT para utilizarla como parámetro en la llamada a la SFC 28. Sólo podemos definirla como variable temporal. Si la definimos como variable de entrada o salida, tendremos el mismo problema de antes. No podremos utilizarla en la SFC 28 porque se nos pide una variable de tipo L o D. Si la definimos como variable de entrada, dentro de la función será local, pero a esta FC habrá que llamarla y darle valores a la variable. Entonces ya no nos servirá como variable local.

La tabla de variables de nuestra FC quedará como sigue:

The screenshot shows a software interface with a tree view on the left and a table on the right. The tree view shows a folder 'Interface' containing sub-items 'IN', 'OUT', 'IN_OUT', 'TEMP', and 'RETURN'. The 'TEMP' item is highlighted. The table on the right is titled 'Contenido de: 'Entorno\Interface\TEMP'' and has the following data:

Nombre	Tipo de datos	Dirección	Comentario
Fecha_y_hora	Date_And_Time	0.0	

Fig. 169

Desde el OB1 sólo podemos introducirle como parámetros formatos de 32 bits como máximo. Si la definimos como temporal no se nos pedirá los valores desde el OB1.

Ahora ya tenemos una variable que podemos introducir en la llamada a la SFC 28. La llamada completa quedaría como podemos ver en la figura siguiente:

```
OB1 : "Main Program Sweep (Cycle)"
```

Comentario:

Segm. 1: Título:

Comentario:

```
CALL "SET_TINT"
OB_NR :=10
SDT :=#FECHA_Y_HORA
PERIOD :=W#16#201
RET_VAL:=#ERROR
```

Fig. 170

Como vimos en la ayuda de la SFC 28, W#16#201, significa que queremos ejecutar la alarma cada minuto. Igual que habíamos hecho en el ejercicio anterior a través del *hardware*. En el parámetro RET_VAL se nos almacenará el error en caso de haberlo, en formato INT. Lo podemos ver en la ayuda de la SFC 28. Nos definimos una variable temporal de tipo INT llamada "Error" para almacenar el error. Dentro de esta FC lo podremos consultar cuando queramos y en cambio no consumimos memoria de marcas ni de entradas / salidas.

Ya tenemos solventado el tema de la llamada a la SFC 28. Ahora tenemos un "problema". De algún modo tenemos que dar valores a la variable fecha_y_hora. Y este valor no se lo podemos dar desde fuera de la función porque tiene más de 32 bits.

Para solventar el tratamiento de este tipo de datos, STEP 7 dispone de unas FC de librerías que nos ayudan a manejar estos tipos de datos largos.

Vamos a las librerías *Standard* y abrimos los bloques "IEC function blocks". Veremos lo siguiente:



Fig. 171

Vamos a utilizar la función FC3. La función se llama D_TOD_DT. Significa que tomará como entrada un formato D (*date*, fecha), un formato TOD (*time of day*, hora del día) y nos devolverá un formato DT con la fecha y hora que nosotros introduciremos por separado. En el ejercicio posterior a las alarmas, se verán más funciones de esta misma librería tratando formatos de fecha y hora. De momento vamos a continuar con este ejemplo y veamos cómo introduciríamos los valores que necesitamos.

Si con el interrogante de ayuda pulsamos sobre la FC3, obtendremos la ayuda de la función y veremos los parámetros que nos va a pedir y en qué formato.



Fig. 172

Aquí vemos cómo va a trabajar la función y además tenemos la posibilidad de pulsar encima de los formatos solicitados y tener ayuda de cada uno de ellos.

Para completar nuestro programa, deberíamos hacer una llamada a la FC3 de esta librería antes de la llamada a la SFC 28. Para ello primero debemos arrastrar esta FC3 a nuestro proyecto. Luego nos definimos unas variables de entrada llamadas Fecha y Hora. A estas variables les daremos valores desde el OB1. Nuestra FC1 quedaría como podemos ver en la figura:

```

Comentario:

Segm. 1: Título:
Comentario:

CALL "D_TOD_DT"
  IN1  :=#FECHA
  IN2  :=#HORA
  RET_VAL:=#FECHA_Y_HORA

CALL "SET_TINT"
  OB_NR :=10
  SDT   :=#FECHA_Y_HORA
  PERIOD :=W#16#Z01
  RET_VAL:=#ERROR
    
```

Fig. 173

Deberíamos, a continuación de la llamada a la SFC 28, tener la llamada a la SFC 30 para activarla. Como parámetros simplemente nos pide el OB al que vamos a hacer referencia. O sea, qué alarma horaria queremos activar. Y la variable donde queremos almacenar el error en caso de que se produzca. Podemos utilizar la misma variable que definimos para la llamada a la SFC 28. En caso de haber error, vendremos aquí en modo **ONLINE** y veremos en cada llamada el valor de esta variable sin haber consumido memoria del PLC.

Para parametrizar la fecha y la hora desde la cual queremos que empiece a funcionar la alarma, deberemos llamar a la FC1 desde el OB1.

La llamada quedaría así:

```
OB1 : "Main Program Sweep (Cycle)"
```

```

Comentario:

Segm. 1: Título:
Comentario:

CALL FC 1
  FECHA:=D#2008-10-8
  HORA :=TOD#15:30:0.0
    
```

Fig. 174

Los formatos de fecha y hora que se nos piden, los podemos ver pulsando F1 en el momento que nos situamos delante de donde se nos solicita el parámetro. Aunque la FC1 la hayamos creado nosotros, también disponemos de esta ayuda.

En este ejemplo, hemos hecho una llamada incondicional a la FC que activa la alarma. Esto es lo mismo que hacerlo por *hardware*.

Conociendo el funcionamiento de estas funciones e instrucciones, el usuario ya podrá darle la utilidad que necesite. Se pueden hacer llamadas condicionales dependiendo de las condiciones que nos interesen, o cambiar los intervalos de ejecución dependiendo de valores de variables, fechas o lo que queramos programar.

En el ejemplo simplemente hemos visto la funcionalidad y las opciones de las que disponemos con el **STEP 7**.

4.9 Programación de alarmas de retardo

Ejercicio 9: Programación de alarmas horarias de retardo

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Alarmas

Para programar este tipo de alarmas, acudimos de nuevo al *hardware*, y entramos con el botón derecho en propiedades del objeto.

Vamos a la ficha de alarmas. Veremos que dependiendo de la CPU de la que dispongamos, tenemos unos OB disponibles. Por ejemplo, si tenemos una CPU de la serie 300, veremos que el OB que tenemos disponible es el OB 20.

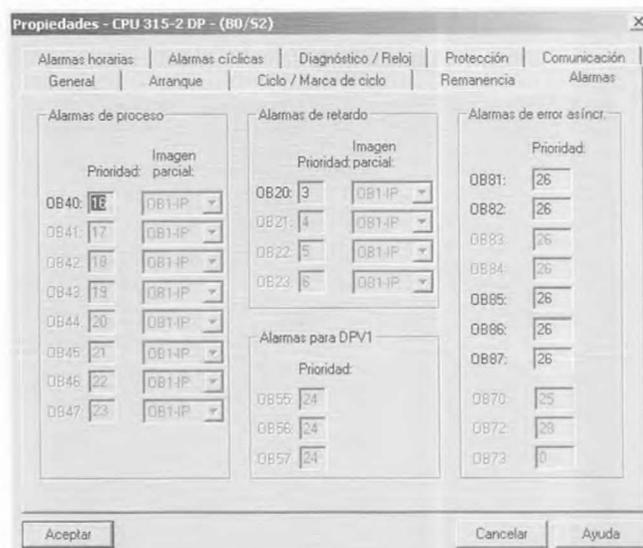


Fig. 175

Para programar estas alarmas tenemos que llamar a la SFC 32. Sólo podemos hacer programación por *software*.

El funcionamiento de las alarmas de retardo es el siguiente:

Nosotros programamos una acción (por ejemplo activar una entrada) y un tiempo. Al cabo de este tiempo que definimos después de haberse dado la acción, se ejecutará el OB que corresponda.

Tenemos que tener cuidado de no programar este tipo de alarmas en un bloque que se ejecute todos los ciclos. Si hacemos esto, nunca acabará de contar el tiempo definido y, por tanto, no se ejecutará el OB en cuestión.

Únicamente tenemos que programar en el OB1 una llamada a la SFC 32. Hay que tener en cuenta no volver a llamar a la SFC 32 antes de haber pasado el tiempo programado en la alarma. De lo contrario nunca se ejecutaría la acción que hayamos programado.

```

OB1 : "Main Program Sweep (Cycle)"
Comentario:
Segm. 1: Título:
Comentario:

      U      E      O.0
      SPBN  META
      CALL  "SRT_DINT"
          OB_NR   :=20
          DTIME  :=T#3M
          SIGN   :=#SIGNO
          RET_VAL:=#ERROR

      META: NOP  0-

```

Fig. 176

En el ejemplo que hemos programado, si mantenemos activa la E 0.0 más de tres minutos, pasado este tiempo se ejecutará lo que hayamos programado en el OB 20. Tenemos que haber transferido dicho OB a la CPU previamente.

En este caso hemos hecho la llamada desde el OB1. Este en un bloque cíclico, lo que pasa es que la llamada a la SFC de la alarma no se ejecuta cada ciclo de *scan*. Hemos condicionado la llamada con una meta y el estado de una entrada.

4.10 Ajustar la hora



DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Conocimiento de las SFC de gestión del reloj.

Además de las SFC de gestión del reloj que veremos en los ejercicios posteriores, tenemos una opción en el menú con la que podemos cambiar la hora de la CPU desde la maleta de programación.

Estando dentro de un bloque de **ONLINE** o bien desde el Administrador de **SI-MATIC**, vamos al menú de Sistema destino y cogemos la opción **“Ajustar la hora”** dentro de **“Diagnóstico / Configuración”**. En las nuevas versiones lo podemos hacer desde cualquier punto del programa.

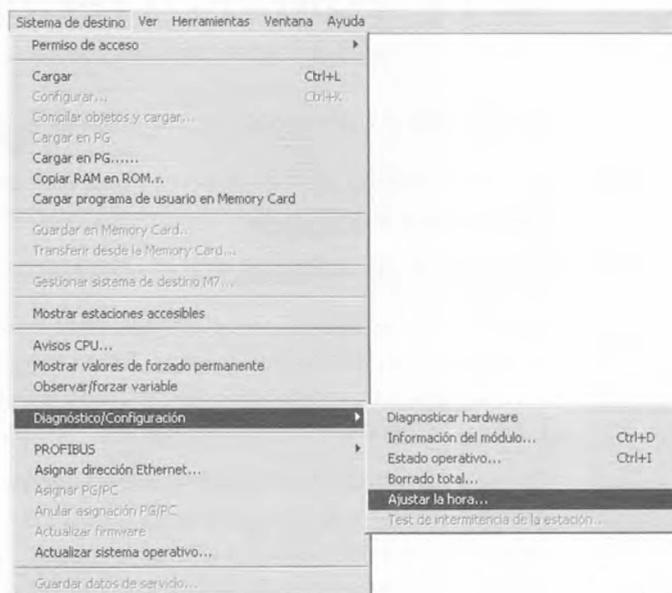


Fig. 177

Veremos una ventana con la hora actual de la CPU. Situándonos encima con el ratón, podemos cambiar la hora y la fecha.

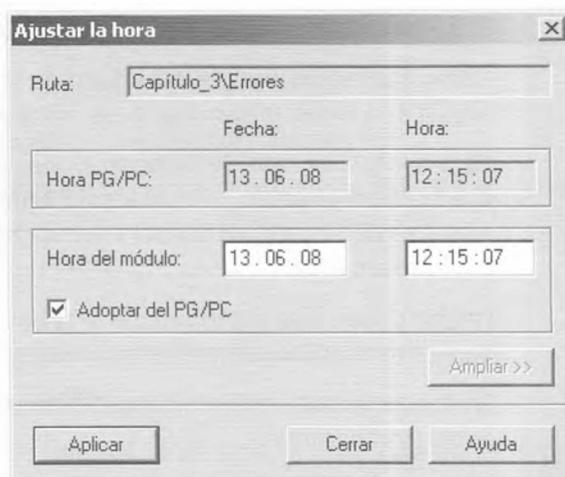


Fig. 178

En esta ventana también vemos la fecha y la hora que tenemos ajustada en el PC. En el PLC podemos poner una hora diferente o podemos decirle que adopte la fecha y la hora del ordenador si seleccionamos la casilla correspondiente.

En el momento pulsemos el botón de “Aplicar”, la CPU tomará la fecha y la hora.

Además esta operación la podemos hacer por programa. Tenemos la función SFC 0 para escribir la fecha y hora de la CPU. Como hemos visto anteriormente, no podemos mover un formato de fecha y hora. Tendremos que hacer una FC en la que llamemos a la SFC 0. Primero tenemos que montarnos el formato de fecha y hora con la FC3 de la librería. Como parámetros de la nueva FC que estamos haciendo, tendremos dos palabras. Una será de tipo fecha y la otra será de tipo hora. Será lo que nos pida cuando hagamos la llamada desde el OB 1. Le daremos como parámetro dos datos de 32 *bits*.

4.11 Formatos fecha y hora

Ejercicio 11: Formatos fecha y hora



DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Introducción de formatos de fecha y hora

Hasta ahora en este manual, hemos visto diferentes tipos de formatos para expresar valores de diferentes formas. Estos valores, los hemos cargado y leído con el ACU 1 de la CPU. Para ello utilizábamos las instrucciones de carga y transferencia L y T.

En este ejercicio vamos a ver nuevos formatos (algunos ya se han visto para programar las alarmas por *software*). En este ejercicio los trataremos más en profundidad. Entre ellos vamos a ver el formato DT (*date and time*). Con este formato podemos expresar una fecha y hora concretas. El problema que va a tener este formato es que ocupa 64 *bits*. Los acumuladores de las CPU de 57 tienen 32 *bits*. Con lo cual no podremos utilizar las instrucciones de carga y transferencia para jugar con este tipo de formato. Veamos en este ejercicio cómo tratamos estos formatos de más de 32 *bits* además de analizar otros formatos cortos.

Queremos hacer un ejercicio en el que se active una luz durante un minuto (de 9:00 horas a 9:01 horas) todos los días del año.

Como operación final de este ejercicio tendremos que comparar si la fecha y hora actual es a la vez mayor que la de inicio y menor que la de fin.

Para hacer este tipo de comparaciones tenemos unas funciones dentro de las librerías que ya están hechas y sólo tenemos que utilizarlas. Cómo son formatos mayores de 32 bits, no nos servirán las funciones de comparación que vimos en el capítulo anterior.

Las funciones que están hechas nos hacen esta comparación pero no sólo con la hora, sino con formato de fecha + hora.

En consecuencia, tendremos que fabricarnos unos formatos de fecha + hora para poder compararlos.

Lo que vamos a hacer es en principio leer la fecha + hora de la CPU. Con esto tendremos la fecha + hora actual. Esto lo hace la función SFC 1.

Recuerda . . .

Existen numerosas funciones en las librerías estándar de Step 7 que podemos utilizar para desarrollar nuestros proyectos. Todas ellas disponen de su ayuda ONLINE, que se nos mostrará pulsando F1 o pinchando con el interrogante de la barra de herramientas sobre ellas.

Vamos a separar este formato en fecha y hora y nos quedaremos con la fecha de hoy. Esto lo hace la FC6 de la librería.

Esta fecha la juntaremos con la hora de inicio y con la hora de fin que nosotros queramos para activar la salida.

De este modo ya tenemos la hora de inicio y la hora de fin con la fecha de cada día. Hemos conseguido lo que queríamos.

Vamos a comparar esta hora + fecha de inicio y de fin con la fecha + hora actual.

En los tres formatos la fecha siempre va a coincidir. Siempre será la fecha de hoy (del día en que nos encontremos). Lo que en realidad estamos comparando es la hora.

Veamos como haríamos esto en AWL.

Antes que nada vamos a ver las funciones que tenemos que gastar de las librerías.

Necesitamos la FC3, la FC6, la FC14 y la FC23.

Además vamos a gastar la SCF 1 que la lleva integrada la CPU.

Primero vamos a entrar con la ayuda en cada una de estas funciones para ver los parámetros que me va a pedir cada una de ellas.

Vamos primero a la **SFC 1**. Vemos que como parámetros tiene:

RET_VAL: Nos va a pedir un valor entero donde nos dejará algún código de error en caso de producirse.

CDT: Será una variable de formato fecha y hora donde nos dejará la fecha y la hora leídas de la CPU.

Vamos a la **FC3:**

IN1: Nos va a pedir una entrada de tipo fecha.

IN2: Nos va a pedir una entrada de tipo hora.

RET_VAL: Nos va a devolver la fecha y la hora que le hemos dado juntas en un formato fecha y hora.

Vamos a la **FC6:**

IN: Nos va a pedir una entrada de formato fecha y hora.

RET_VAL: Nos va a devolver la fecha sola del formato anterior.

Vamos a la **FC14:**

DT1: Nos pide una entrada de formato fecha y hora.

DT2: Nos pide una entrada de formato fecha y hora.

RET_VAL: Nos pide una salida de tipo binario. Este bit se activará si la primera entrada que hemos metido es mayor que la segunda.

Vamos a la **FC23:**

DT1: Nos pide una entrada de formato fecha y hora.

DT2: Nos pide una entrada de formato fecha y hora.

RET_VAL: Nos pide una salida de tipo binario. Este bit se activará si la primera entrada es más pequeña que la segunda entrada.

Para poder utilizar todas estas funciones tenemos que tenerlas dentro de nuestro proyecto. La primera cosa que tenemos que hacer es traerlas a nuestro proyecto para poderlas llamar y poderlas utilizar.

Ahora nosotros nos tenemos que crear una FC con todas las variables que nos van a hacer falta para darlas como parámetros a las FC que vamos a llamar.

Vamos a crear una FC que no tenga el mismo número que las que tenemos que utilizar. Vamos a hacer por ejemplo la FC2.

Su tabla de variables quedaría como vemos a continuación. Las variables las hemos ido definiendo según nos han ido haciendo falta para programar. Sobre todo las temporales.

FC2

IN	HORA_INICIO	TIME_OF_DAY
IN	HORA_FIN	TIME_OF_DAY
OUT	SALIDA	BOOL
TEMP	ERROR	INT
TEMP	FECHA_Y_HORA_ACTUAL	DATE_AND_TIME
TEMP	FECHA_ACTUAL	DATE
TEMP	F_H_INICIO	DATE_AND_TIME
TEMP	F_H_FIN	DATE_AND_TIME
TEMP	BIT_1	BOOL
TEMP	BIT_2	BOOL

Segmento 1: Lectura del valor de la fecha y hora actual.

```
CALL SFC 1
RET_VAL:= #ERROR
CDT:= #FECHA_Y_HORA_ACTUAL
```

Segmento 2: Separa la fecha y hora actuales en sólo fecha actual.

```
CALL FC 6
IN:= #FECHA_Y_HORA_ACTUAL
RET_VAL:= #FECHA_ACTUAL
```

Segmento 3: Une FECHA_ACTUAL a la hora de inicio y crea fecha y hora de inicio.

```
CALL FC 3
IN1:= #FECHA_ACTUAL
IN2:= #HORA_INICIO
RET_VAL:= F_H_INICIO
```

Segmento 4: Une FECHA_ACTUAL a la hora de fin y crea fecha y hora de fin.

```
CALL FC 3
IN1:= FECHA_ACTUAL
IN2:= #HORA_FIN
RET_VAL:= F_H_FIN
```

Segmento 5: Compara si fecha y hora actual > fecha y hora de inicio.

```
CALL FC 14
DT1:= #FECHA_Y_HORA_ACTUAL
DT2:= #F_H_INICIO
RET_VAL:= #BIT_1
```

Segmento 6: Compara si fecha y hora actual < fecha y hora de fin.

```
CALL FC 23
DT1:= #FECHA_Y_HORA_ACTUAL
DT2:= #F_H_FIN
RET_VAL:= #BIT_2
```

Segmento 7: Activación de la salida.

```
U #BIT_1
U #BIT_2
= #SALIDA
BE
```

Ahora nos queda hacer un OB1 para decir cuando tiene que ejecutarse esta FC y además con qué valores tiene que ejecutarse.

OB1

```
CALL FC 2
HORA_INICIO:= TOD#9:00:00
HORA_FIN:= TOD#9:01:00
SALIDA:= A 4.0
BE
```

En estos parámetros le decimos a la hora que queremos que se active la salida y a la hora que queramos que se desactive la salida, y además la salida que queremos que se active.

El hacerlo como parámetro nos da, además, la ventaja de poder cambiar desde aquí estos datos sin tener que entrar en la FC que hemos programado.

Si nos interesa guardar esta FC en una librería, lo que tenemos que hacer es el proceso inverso al de antes. Abrimos o creamos una librería nueva, y arrastramos la FC a la librería que nos interesa.

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

4.12 Hacer funcionar algo un día de la semana

Ejercicio 12: Hacer funcionar algo un día de la semana



DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Formatos de fecha y hora

Vamos a hacer un programa en el que se active una salida un día de la semana determinado que yo le diga desde un parámetro. Si yo en el parámetro le digo "LUNES" quiero que todos los lunes se active la salida. Si yo le digo "MARTES", quiero que todos los martes se active la salida, etcétera.

El autómata, en el formato de fecha y hora, tiene unos *bits* que nos indican el número de día dentro de la semana. Estos *bits* contendrán 1 si es domingo, contendrán 2 si es lunes, 3 si es martes, 4 si es miércoles, 5 si es jueves, 6 si es viernes y 7 si es sábado.

Luego lo que tenemos que hacer es una comparación entre este código interno y el parámetro que yo le estoy introduciendo del día de la semana que yo quiero.

Estos *bits* son los cuatro últimos del formato de fecha y hora.

Este formato está compuesto por 8 *bytes* (64 *bits*).

Nosotros no podemos manejar 64 *bits*. No podemos pasar como parámetro 64 *bits*, ni podemos cargar y transferir. Los acumuladores son de 32 *bits*. Como máximo podemos mover formatos de 32 *bits*.

Para trabajar con este tipo de formatos acudimos de nuevo a las FC hechas en las librerías que nos permiten cambiar este formato de 64 *bits* y transformarlo en formatos de 32 *bits* o menos.

Como ya hemos visto en ejercicios anteriores, tenemos una FC que nos corta el formato de fecha y hora y nos lo convierte en formato de sólo fecha.

Tenemos otra FC que nos corta el formato fecha y hora y nos devuelve sólo la hora.

Tenemos otra FC que lee el formato fecha y hora y nos devuelve un número entero que corresponde al día de la semana.

Esta es la FC7 dentro de la librería IEC.

Si cogemos el interrogante que tenemos en el menú superior de herramientas y nos situamos encima de la FC7, obtenemos la ayuda de lo que hace y de cómo funciona.



Fig. 179

Recuerda . . .

Los formatos de más de 32 bits no los podemos utilizar mediante los acumuladores. Debemos utilizarlos en variables locales de las funciones o en datos de un DB.

Vemos que como parámetros nos va a pedir un formato de fecha y hora y nos devuelve un entero que es el día de la semana con el código que podemos ver en la ayuda.

Ahora sólo nos queda hacer el programa. Tendremos primero que llamar a la SFC 1 para leer la fecha y la hora que tenemos en la CPU. Una vez tengamos la fecha y la hora, tendremos que sacar de allí el día de la semana.

Una vez tengamos el día de la semana, compararemos con un número entero para decirle el día de la semana que queremos que haga algo.

Veamos como quedaría el programa en AWL.

FC3

TEMP	FECHA_HORA	DATE_AND_TIME
TEMP	ERROR	WORD
TEMP	DIA_SEMANA	INT
OUT	SALIDA	BOOL
TEMP	DIA	INT

```
CALL      SFC      1
          SDT:= FECHA_HORA
          RET_VAL:= ERROR
```

```
CALL      FC      7
          SDT:= FECHA_HORA
RET_VAL:= DIA_SEMANA
          L      DIA_SEMANA
          L      DIA
          ==|
          = SALIDA
```

OB1

```
CALL      FC      3
          SALIDA:= A4.0
          DIA:= 3
```

4.13 Convertir archivos de S5 a S7

Ejercicio 13: Convertir archivos de S5 a S7



DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Conocer algo de S5

A fecha de hoy el **STEP 5** de **SIEMENS** ya lo tenemos descatalogado. Ya no se venden equipos nuevos con esta tecnología. Lo que ocurre es que en el mercado todavía quedan muchos equipos de esta serie. Posiblemente la persona que se dedica a la programación de PLC y se mueve en el mundo de **SIEMENS**, se encuentre alguno de estos equipos en su trabajo.

Quizás alguna vez tengamos que transformar un programa hecho en **S5** a otro para un equipo nuevo **S7**. Para esto el **STEP 7** dispone de una herramienta que nos puede ayudar. Esta herramienta será simplemente una ayuda. No traduce todo lo que se tenga en **S5** pero si una gran mayoría de las instrucciones. Todo lo relativo a redes y comunicaciones no nos lo va a traducir. Tampoco introduce instrucciones nuevas para optimizar el programa. Tampoco redirecciona automáticamente si hemos cambiado la periferia existente. Después de utilizar esta herramienta, se requerirá siempre la revisión del programa obtenido y muchas veces el retoque del programador para obtener lo que realmente se necesita.

A modo de ejemplo, nosotros vamos a hacer un pequeño programa en **S5** y lo convertiremos a **S7**. Veremos que con algo tan sencillo la herramienta de traducción lo hace perfectamente.

El programa que vamos a hacer en **S5** es el siguiente:

OB1

```
U      E      0.0
SPB    PB1
SPA    PB2
BE
```

PB1

```
U      E      0.1
L      KT     5.2
SE     T      1
U      T      1
=      A      2.0
BE
```

PB2

```
U      E      1.2
=      A      2.1
BE
```

Para poder hacer este programa necesitamos tener instalado el **STEP 5**.

Este programa lo hacemos con el editor de **S5**. Una vez lo tenemos hecho, salimos de la aplicación y volvemos al Administrador de **SIMATIC**.

En **S5** tenemos que saber dónde tenemos almacenado el programa. Dentro de los ajustes de **S5** podemos ver (y elegir) en que directorio tenemos el proyecto. Luego nos hará falta buscarlo en su directorio correspondiente y con su nombre correspondiente.

Para abrir la aplicación de traducción, lo hacemos desde el menú de inicio del ordenador. "Inicio à **SIMATIC** à Convertir archivo **S5**". El idioma en que veamos este diálogo dependerá del idioma en el que lo instalemos en su día. Dependiendo de la instalación que hayamos hecho en el ordenador, es posible que el menú esté en: "Inicio -> Todos los programas -> **SIMATIC** -> Convertir archivo **S5**".

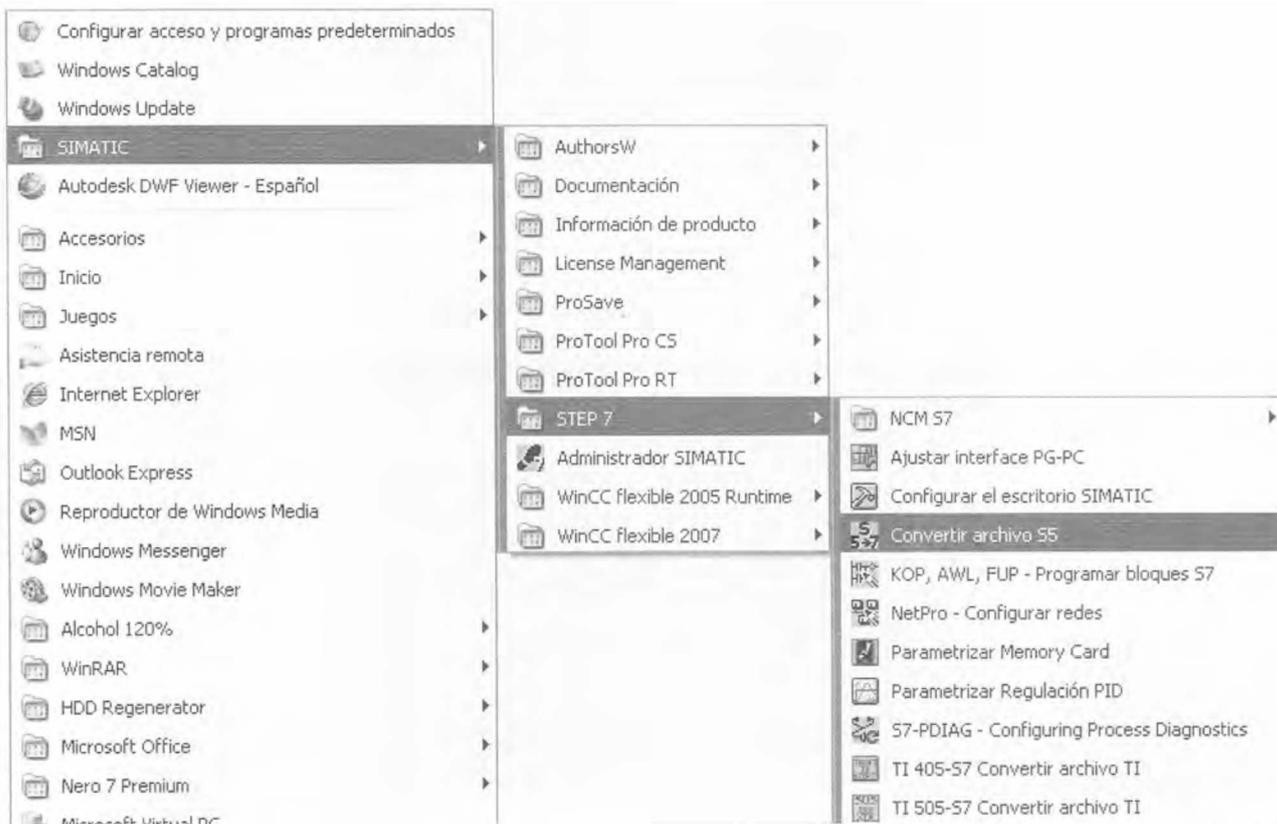


Fig. 180

Entramos en la aplicación y vemos una ventana como esta:

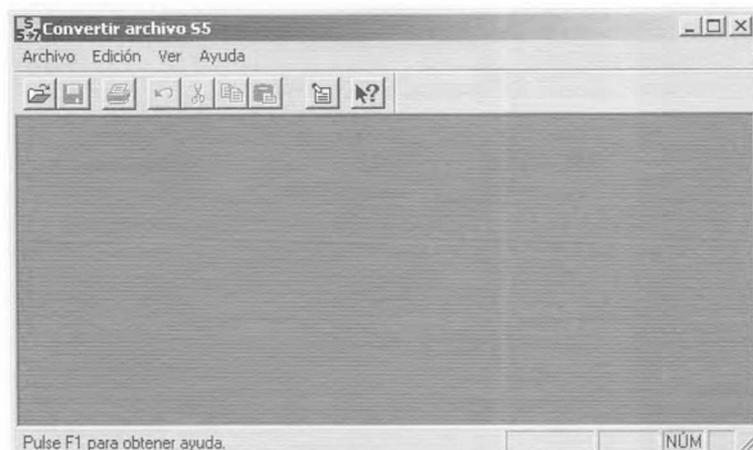


Fig. 181

Pulsamos el botón de abrir y abrimos el archivo que hemos generado en **STEP 5**.

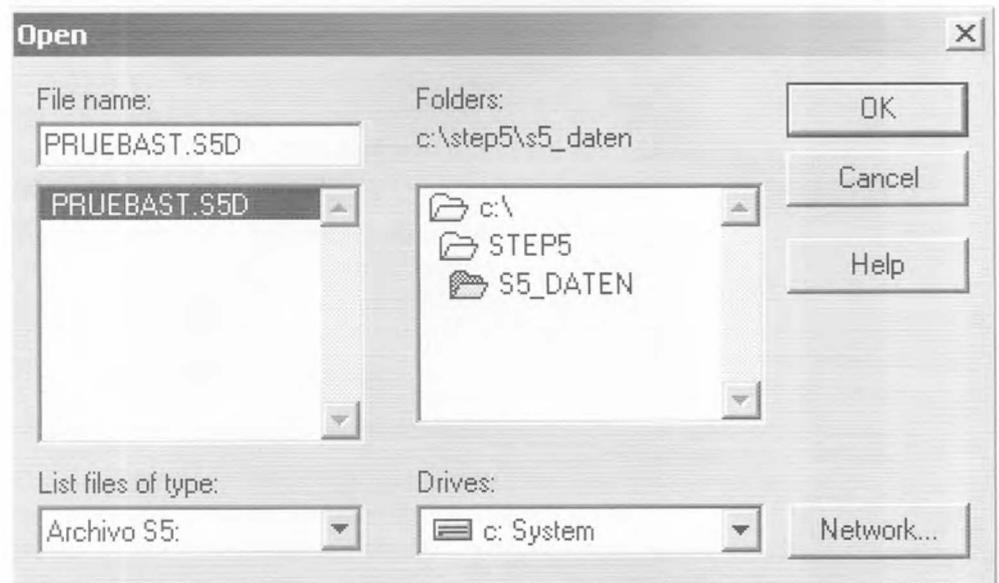


Fig. 182

Cuando abramos el fichero, veremos un diálogo como el siguiente:

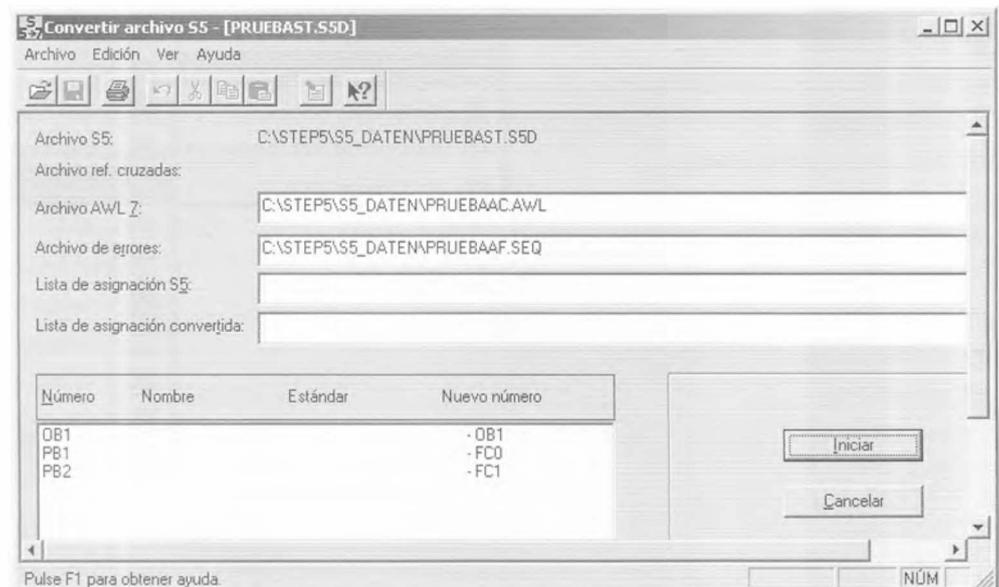


Fig. 183

En esta ventana, podemos ver un resumen de lo que va a hacer la aplicación de conversión. Nos dice dónde va a dejar el archivo AWL (programa traducido) y dónde va a dejar el archivo con los errores si es que los encuentra. También nos explica cómo va a quedar la conversión de bloques. El OB1 lo va a dejar como OB1, el PB 1 lo va a convertir en FC 0 y el PB 2 lo va a convertir como FC 1.

Si ahora pulsamos el botón de "Iniciar", comenzará la traducción del fichero.

El propio programa nos dice donde va a guardar el programa convertido. Nos tenemos que fijar dónde lo va a guardar para luego sacarlo desde **STEP 7**.

Cuando haya terminado veremos un mensaje como el siguiente:



Fig. 184

Aquí nos hace el balance de los errores o comentarios de la traducción. En este caso vemos que no tenemos errores ni advertencias.

Una vez terminada la traducción, cerramos esta aplicación y volvemos al Administrador de **SIMATIC**.

En **STEP 5**, se hacían los bloques directamente. No incluíamos el hardware dentro de lo que es el proyecto. Ahora en **STEP 7** esto es diferente. Tenemos que crear un proyecto nuevo con la configuración que tenemos ahora en **STEP 7**. Una vez lo hemos creado, tendremos un proyecto con su configuración pero sin programa.

En la parte izquierda de la ventana del proyecto en el Administrador de **SIMATIC**, pinchamos encima de la carpeta de fuentes. Veremos que de momento no tenemos nada en la parte derecha.

Nos ponemos en la parte derecha de la ventana. Pinchamos con el botón derecho y elegimos la opción "**Fuente externa**".



Fig. 185

Insertamos la fuente que nos ha creado el conversor de archivos.

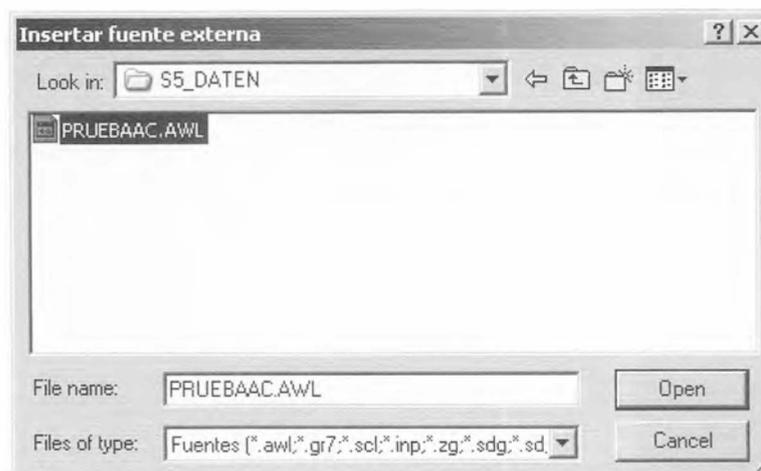


Fig. 186

Una vez tenemos la fuente, la veremos así dentro del proyecto de SIMATIC.

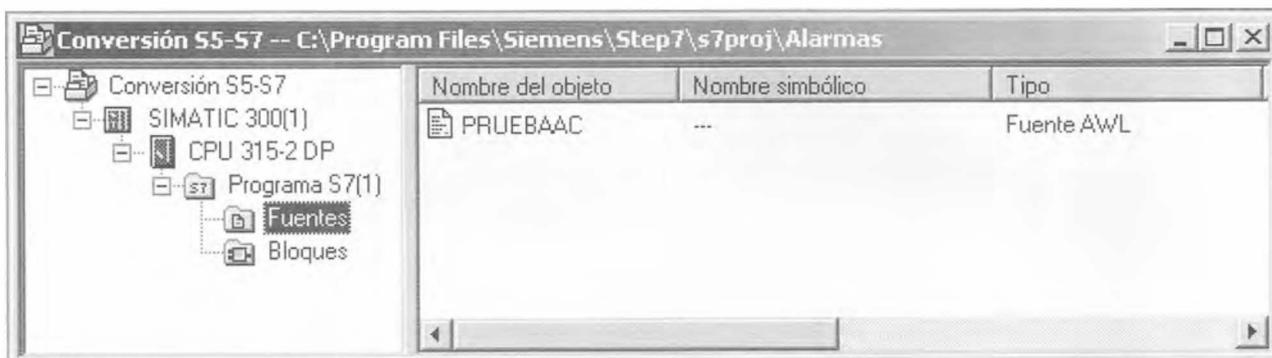


Fig. 187

Hacemos doble clic sobre ella y la abrimos.

Vemos que tenemos el programa convertido en un lenguaje un poco diferente al **STEP 7**. Las instrucciones son las mismas pero el encabezado y el final de los bloques son distintos.

Además vemos que todos los bloques están juntos. Tenemos uno a continuación de otro.

Todo lo que podemos ver al principio de la fuente en color verde y con una doble barra delante, son comentarios. No afecta al programa. En estos comentarios, podemos ver a modo de una ayuda sobre esta aplicación.

Nuestra fuente con el programa traducido la podemos ver de la siguiente manera:

```

ORGANIZATION_BLOCK OB 1
VAR_TEMP
  OB1_EV_CLASS : BYTE ; //Bits 0-3 = 1 (Coming event), Bits 4-7 = 1 (Event class 1)
  OB1_SCAN_1 : BYTE ; //1 (Cold restart scan 1 of OB 1), 3 (Scan 2-n of OB 1)
  OB1_PRIORITY : BYTE ; //1 (Priority of 1 is lowest)
  OB1_OB_NUMBER : BYTE ; //1 (Organization block 1, OB1)
  OB1_RESERVED_1 : BYTE ; //Reserved for system
  OB1_RESERVED_2 : BYTE ; //Reserved for system
  OB1_PREV_CYCLE : INT ; //Cycle time of previous OB1 scan (milliseconds)
  OB1_MIN_CYCLE : INT ; //Minimum cycle time of OB1 (milliseconds)
  OB1_MAX_CYCLE : INT ; //Maximum cycle time of OB1 (milliseconds)
  OB1_DATE_TIME : DATE_AND_TIME ; //Date and time OB1 started
END_VAR
BEGIN
NETWORK
  U E 0.0;
  SPEN X000;
  CALL FC 0;
X000: NOP 0;
  CALL FC 1;

END_ORGANIZATION_BLOCK

FUNCTION FC 0 : VOID
BEGIN
NETWORK
  U E 0.1;
  L S5TIME#S5;
  SE T 1;
  U T 1;
  = A 2.0;

END_FUNCTION

FUNCTION FC 1 : VOID
BEGIN
NETWORK
  U E 1.2;
  = A 2.1;

END_FUNCTION
    
```

Fig. 188

Ahora tenemos que compilar esto para dejarlo ya como bloques de **STEP 7**. Compilamos.

Para ello tenemos un botón como el siguiente:



Fig. 189

Una vez compilados, nos saldrán una serie de advertencias y una serie de errores si es que los tiene la aplicación. Esto lo podemos ver en la parte inferior de la ventana del editor de bloques. Por defecto esta parte inferior de las ventanas, viene minimizada. Si queremos ver el resultado de la compilación, tenemos que abrir esta parte de la ventana arrastrando con el ratón el extremo superior de la misma.

Si la tenemos abierta veremos algo similar a lo que se muestra a continuación:

```

FUNCTION FC 1 : VOID
BEGIN
NETWORK
    U   E 1.2;
    =   A 2.1;

END_FUNCTION
    
```

X Compilar: Conversión S5-S7\SIMATIC 300(1)\CPU 315-2 DP\Programa S7(1)\Fuentes\PRUEBAAC
 E Lín 000079 Col 015: no se ha encontrado ninguna descripción del tipo de AS para el módulo llamado o direccionado FC 0.
 A Lín 000079 Col 015: no se ha encontrado ninguna descripción de tipo ASCII offline sobre el bloque llamado o direccionado FC 0.
 E Lín 000081 Col 017: no se ha encontrado ninguna descripción del tipo de AS para el módulo llamado o direccionado FC 1.
 A Lín 000081 Col 017: no se ha encontrado ninguna descripción de tipo ASCII offline sobre el bloque llamado o direccionado FC 1.
 Resultado compilación: 2 errores, 2 advertencias

1: Error 2: Info 3: Referencias cruzadas 4: Información operando 5: Forzado 6: Diagnóstico 7: Comparación

Fig. 190

En el ejemplo que hemos hecho vemos que tenemos un error en la llamada al bloque. Las llamadas a bloques las ha traducido como CALL. Los bloques que hemos hecho nosotros no tienen parámetros. La llamada CALL es para bloques con parámetros. Para llamadas a bloques sin parámetros se utilizan las instrucciones UC o CC. En el resumen de errores de la parte inferior, si hacemos doble clic sobre cada uno de ellos, automáticamente el programa nos señala la instrucción dentro del programa que tiene el error.

Sabiendo esto, corregimos las dos llamadas y volvemos a compilar. El programa quedará así:

```

        U   E 0.0;
        SPEN X000;
        CC FC 0;
X000: NOP 0;
        UC| FC 1;

END_ORGANIZATION_BLOCK

FUNCTION FC 0 : VOID
BEGIN
NETWORK
    U   E 0.1;
    L   S5TIME#5s;
    SE  T 1;
    U   T 1;
    =   A 2.0;

END_FUNCTION

FUNCTION FC 1 : VOID
BEGIN
NETWORK
    U   E 1.2;
    =   A 2.1;

END_FUNCTION

```

Fig. 191

Ahora ya saldrá bien la compilación. Ya no tenemos ningún error. Una vez está compilado, cerramos la fuente. No es necesario cerrar el editor de bloques.

Una vez compilado correctamente ya tenemos la conversión hecha.

Si ahora vamos a la carpeta de bloques de **STEP 7**, vemos que tenemos todos los bloques creados. Veremos que tenemos un OB1, una FC 0 y una FC1.

Si entramos en cada uno de ellos vemos como los ha traducido.

Estos bloques ya están listos para enviarlos al PLC de **S7**. Al haber traducido el programa directamente de **STEP 5**, posiblemente habrá entradas o salidas que no tengan la dirección que queramos. Por ejemplo, en el fichero que hemos hecho de **S5** estamos gastando salidas 2.0 y 2.1, y en la configuración que tenemos en **STEP 7** tenemos *bytes* 4 y 5 de salidas. Para nosotros, con el PLC que hemos descrito como ejemplo, no existe el *byte* 2 de entradas.

Esto lo podemos resolver entrando en los bloques y cambiando las salidas por las que nos interesan en este caso, o utilizar la herramienta de que dispone el **S7** para ello. En este caso es muy fácil hacerlo a mano puesto que es un programa muy cortito. Pero si tuviésemos un programa grande sería muy complicado leer todas las instrucciones y hacer los cambios pertinentes. Además correremos el riesgo de dejarnos alguna salida por cambiar o de equivocarnos en alguno de los cambios.

Si queremos hacerlo utilizando la herramienta del **STEP 7**, vamos al Administrador de **SIMATIC**. Teniendo seleccionado en la parte izquierda de la ventana la carpeta de bloques, vamos al menú herramientas y, dentro del menú herramientas, a la opción de "recablear".

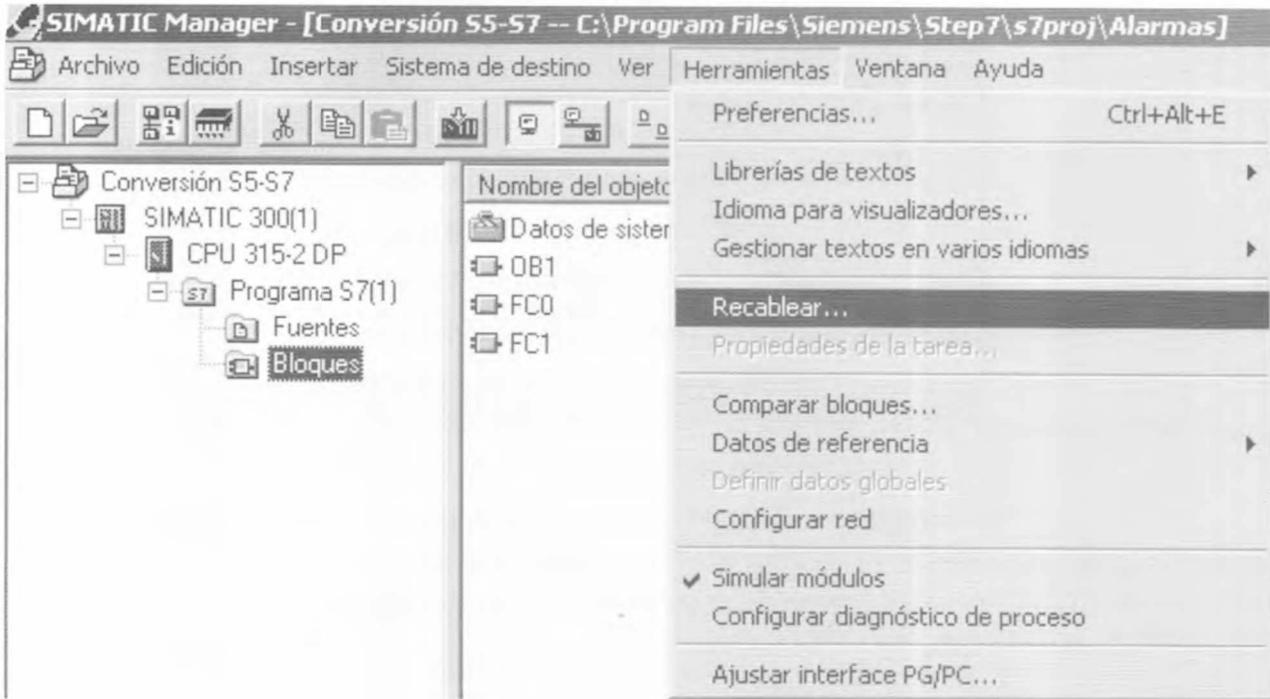


Fig. 192

Aquí podemos renombrar los contactos que queramos. Aparecen dos columnas. En la columna izquierda ponemos el nombre que tiene el contacto actualmente y, en la parte derecha, ponemos el nuevo direccionamiento que queremos que tenga.

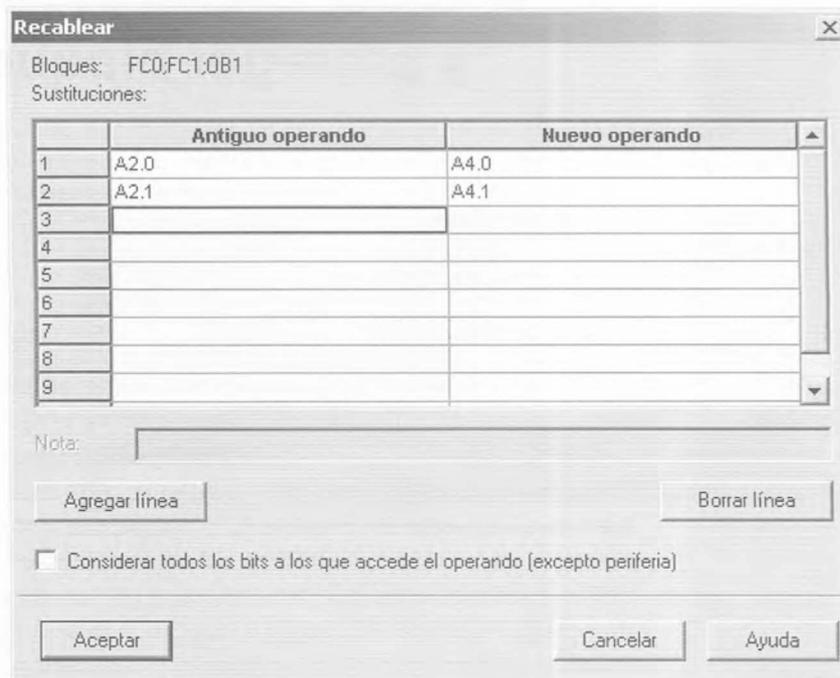
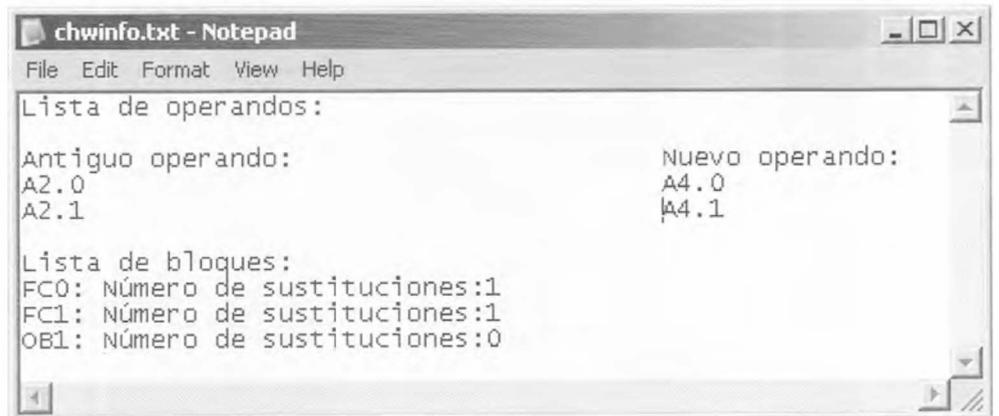


Fig. 193

En el momento pulsamos el botón de aceptar, el cambio se produce en todos los sitios donde aparezca el contacto.

El propio programa nos da un informe de todos los cambios que ha efectuado y dónde ha hecho los cambios.



```

chwinfo.txt - Notepad
File Edit Format View Help
Lista de operandos:
Antiguo operando:
A2.0
A2.1
Nuevo operando:
A4.0
A4.1
Lista de bloques:
FC0: Número de sustituciones:1
FC1: Número de sustituciones:1
OB1: Número de sustituciones:0
  
```

Fig. 194

Deberemos transferir al PLC todos los bloques que han tenido cambios para poder trabajar correctamente con el nuevo programa.

Ya tenemos el proyecto en **STEP 7** como nosotros lo queremos.

4.14 Crear archivos fuente y proteger bloques

Ejercicio 14: Crear archivos fuente y proteger bloques ✓

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA

También podemos crear un programa como archivo fuente o como archivo de texto en un editor de texto. En algún caso se podría hacer el programa sin tener el **STEP 7** y posteriormente compilarlo para enviarlo a la CPU. Aunque esto no es lo habitual. No están pensadas las fuentes para esto. Las fuentes se suelen utilizar bien para convertir archivos de **S5** como hemos visto en el ejercicio anterior o, por ejemplo, para “crear programas automáticamente”. Si tenemos cosas muy repetitivas o instalaciones que tenemos que copiar varias veces, podemos crear unas macros que generen el código y luego compilarlo para obtener los bloques de **S7**. Quizá al usuario se le ocurra alguna otra aplicación que puedan tener las fuentes después de ver lo que son y como podemos generarlas y trabajar con ellas.

Los programas de la CPU están formados básicamente por los bloques que programamos, la configuración y la conexión a las distintas redes que vayamos a utilizar.

Para programar archivos fuente disponemos en el Administrador de **SIMATIC** de un editor de textos con compilador.

Podemos hacer todos los bloques juntos y compilarlos de una sola pasada.

Con esto tenemos las siguientes “ventajas”:

- Podemos programar varios bloques en un archivo fuente.
- El archivo fuente se puede guardar aunque tenga errores.
- Hasta que no se compile no nos informa de los errores.
- Podemos crear el bloque con otros editores de texto e importarlo.
- Podemos generar fuentes a través de macros.
- Podemos utilizar instrucciones que al compilarlas no veremos en **STEP 7** como por ejemplo las instrucciones para proteger bloques.

Esto no son en si ventajas de la programación. Simplemente son posibilidades que tenemos con el compilador de **S7**. No es habitual utilizarlo para programar. Aunque tiene algunas utilidades como veremos en este ejemplo. Veremos cómo podemos crear un bloque protegido a través de una fuente.

Hemos dicho que el programa se compone de los bloques + la configuración. Con el editor de textos podemos hacer los bloques, pero no la configuración.

Para crear un proyecto completo, tendremos que hacer la configuración normalmente con el Administrador de **SIMATIC** y luego crear los bloques a parte en un editor de textos.

El programa lo podemos hacer directamente con el editor de textos. No nos hace falta para nada el **STEP 7**. Pero a la hora de transferir el programa a la CPU, tendrá que estar completo con su configuración y hecho en **STEP 7**.

Veamos como haríamos un programa en el Wordpad por ejemplo:



Fig. 195

Tenemos que tener en cuenta escribir un ; detrás de cada instrucción. Además tenemos que respetar los espacios que luego se nos van a requerir en **STEP 7**.

Creamos un proyecto en **STEP 7**. Lo podemos crear sin decirle el equipo que tenemos.

Dentro de fuentes creamos una fuente nueva como fuente AWL.

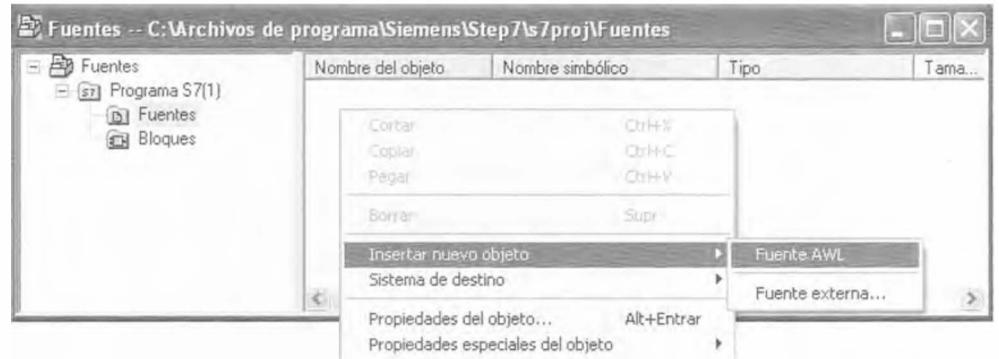


Fig. 196

Desde esta fuente nueva podemos entrar en el editor haciendo doble clic en ella.

Dentro del editor de textos tenemos dos opciones. Podemos copiar la fuente que creamos con el Wordpad, o podemos escribir aquí mismo la fuente utilizando el editor. De cualquier manera, tenemos que generar una nueva fuente como la que hemos creado en el ejemplo.

Para crear los bloques en el editor de textos podemos utilizar plantillas ya existentes. Lo que habíamos escrito en el Wordpad por ejemplo, el sistema no sabrá si es el OB 1 o cualquier otro bloque si no se lo indicamos con las cabeceras de bloque. Recordemos que en las fuentes programamos todos los bloques juntos.

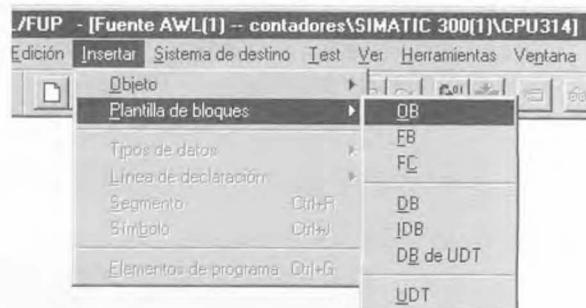


Fig. 197

Las plantillas las podemos modificar si incluyen algo que no nos interesa. O podemos añadir algo más como veremos a continuación en este mismo ejemplo.

En cualquier momento podemos guardar el archivo que hemos generado. No es preciso que esté compilado y correcto.

Usando el comando de menú **Archivo > comprobar consistencia**, se puede comprobar en cualquier momento la coherencia y la sintaxis del archivo fuente, sin que ello inicie la creación de bloques. Nos indicará el número de líneas compiladas y la cantidad de errores y advertencias.

Al compilar es cuando se generan los bloques. Sólo se generan los que no contengan errores.

Tenemos que tener en cuenta escribir un ; detrás de cada instrucción. Además tenemos que respetar los espacios que luego se nos van a requerir en **STEP 7**.

Creamos un proyecto en **STEP 7**. Lo podemos crear sin decirle el equipo que tenemos.

Dentro de fuentes creamos una fuente nueva como fuente AWL.

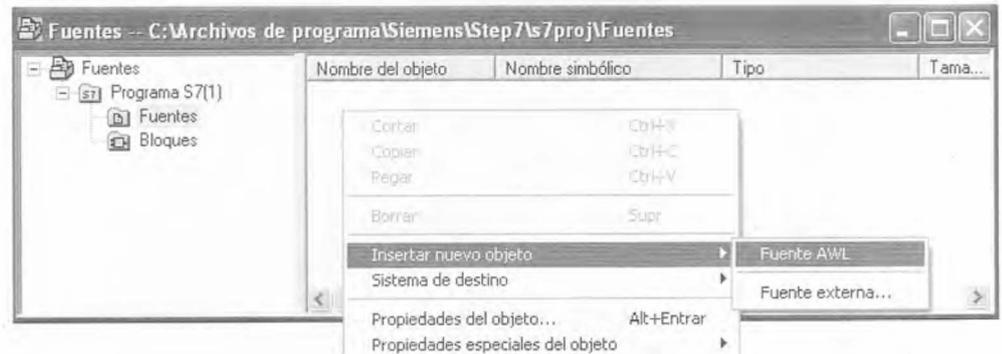


Fig. 196

Desde esta fuente nueva podemos entrar en el editor haciendo doble clic en ella.

Dentro del editor de textos tenemos dos opciones. Podemos copiar la fuente que creamos con el Wordpad, o podemos escribir aquí mismo la fuente utilizando el editor. De cualquier manera, tenemos que generar una nueva fuente como la que hemos creado en el ejemplo.

Para crear los bloques en el editor de textos podemos utilizar plantillas ya existentes. Lo que habíamos escrito en el Wordpad por ejemplo, el sistema no sabrá si es el OB 1 o cualquier otro bloque si no se lo indicamos con las cabeceras de bloque. Recordemos que en las fuentes programamos todos los bloques juntos.



Fig. 197

Las plantillas las podemos modificar si incluyen algo que no nos interesa. O podemos añadir algo más como veremos a continuación en este mismo ejemplo.

En cualquier momento podemos guardar el archivo que hemos generado. No es preciso que esté compilado y correcto.

Usando el comando de menú **Archivo > comprobar consistencia**, se puede comprobar en cualquier momento la coherencia y la sintaxis del archivo fuente, sin que ello inicie la creación de bloques. Nos indicará el número de líneas compiladas y la cantidad de errores y advertencias.

Al compilar es cuando se generan los bloques. Sólo se generan los que no contengan errores.

Para generar un archivo fuente, tenemos que tener en cuenta las siguientes reglas generales:

- La sintaxis de las instrucciones AWL es la misma que la del editor incremental. Una excepción a esta regla son las llamadas a bloques y la declaración de Arrays y estructuras.
- El editor de textos no distingue generalmente entre mayúsculas y minúsculas. De esta regla quedan exceptuados los nombres de las variables.
- Se debe señalar el final de todas las instrucciones AWL y declaraciones de variables escribiendo un punto y coma al final. Se puede introducir más de una instrucción por línea.
- Los comentarios deben comenzar con dos barras inclinadas (//) y la entrada de comentarios debe finalizarse con la tecla enter.

En lo que se refiere al orden que debe haber entre los distintos bloques, se deben tener en cuenta las siguientes reglas:

- El OB1 que es el que llama a otros bloques tiene que estar al final. Cada uno de los otros bloques tiene que estar detrás de los bloques a los que llama.
- Aquellos bloques que llame en el archivo fuente, pero que no programe en ese mismo archivo fuente, ya tienen que haber sido creados en el programa de usuario correspondiente cuando se vaya a compilar el archivo.

Estructura básica de los bloques:

- Comienzo del bloque con indicación del bloque.
- Título del bloque.
- Comentario del bloque.
- Atributos de sistema para los bloques.
- Propiedades del bloque.
- Tabla de declaración.
- Área de instrucciones de bloques lógicos o asignación de valores actuales en bloques de datos.
- Fin de bloque.

Veamos una utilidad de este tipo de editor. Vamos a ver cómo podemos proteger bloques.

Para ello tenemos que pasar obligatoriamente por el formato de fuente. Podemos hacer un bloque a través de una fuente y protegerlo o tener un bloque ya programado desde el **STEP 7** y querer protegerlo.

Para ello tenemos una opción que convierte los bloques en fuentes.

Veamos como haríamos eso.

Supongamos que tenemos el siguiente bloque programado en **STEP 7**:

```

OB1 : Título:
Segm. 1): Título:
      U      E      0.0
      U      E      0.1
      =      A      4.0
      CC     FC      1
    
```

Fig. 198

Una vez tengamos el bloque programado, tenemos una opción en el menú de “**archivo**”, que nos genera la fuente.

Para poder acceder a esta opción, tenemos que estar dentro del bloque del cual queremos tener la fuente.



Fig. 199

Para crear la fuente, nos pide un nombre:

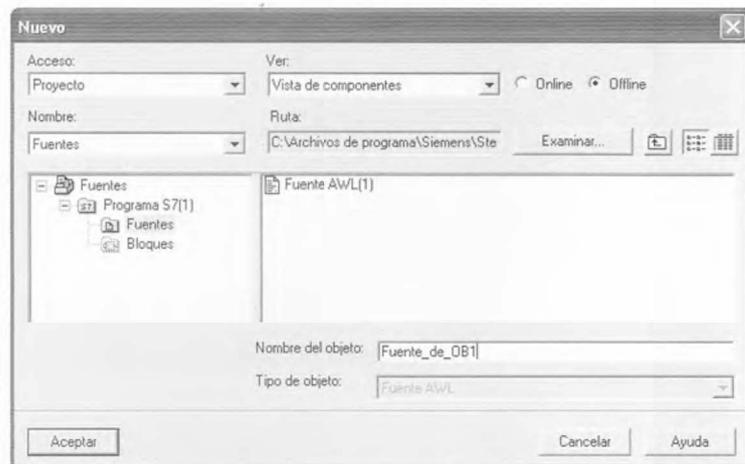


Fig. 200

Aquí le damos el nombre que queremos que tenga la fuente. En el ejemplo le hemos llamado “Fuente_de_OB1”, ya que vamos a generar el código fuente del bloque OB1.

Una vez le ponemos nombre, nos permite seleccionar qué bloques de los que tenemos creados, son los que queremos incluir en la fuente. En este caso hemos generado una fuente que se llama Fuente_de_OB1, y vamos a incluir solamente el OB1.

En la hoja de selección de bloques solamente pasamos a la parte derecha el icono del OB1. Veamos como queda la selección.

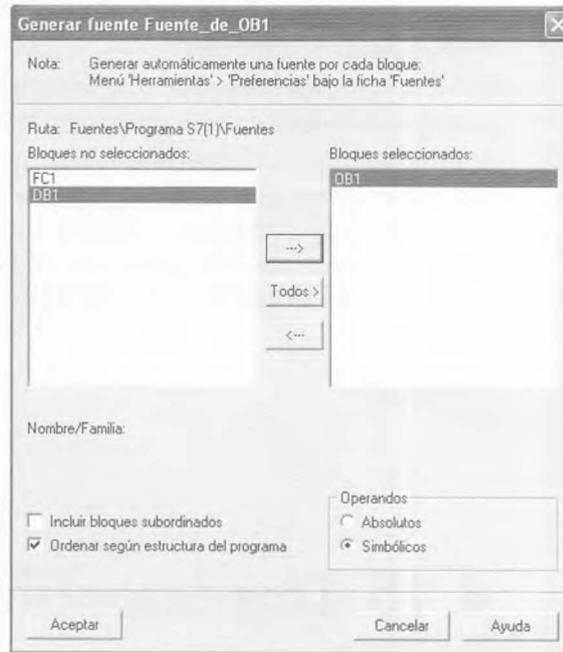


Fig. 201

Una vez le digamos aceptar, ya tenemos la fuente generada.

Ahora entramos en la fuente.

Para proteger bloques, tenemos que estar en formato fuentes. Podemos llegar aquí bien porque hemos creado directamente una fuente o porque hemos convertido un bloque como acabamos de hacer en este caso.

Para que el bloque quede protegido en la fuente tenemos que añadir la palabra: KNOW_HOW_PROTECT delante de la definición de las variables del bloque.

Veamos como quedaría:

```

KOP/AWL/FUP - [ejemplo -- ALARMAS\SIMATIC 300(1)\CPU314]
Archivo Edición Insertar Sistema de destino Test Ver Herramientas Ventana Ayuda
ORGANIZATION_BLOCK OB 1
TITLE =
VERSION : 0.1
Know_how_protect|
VAR_TEMP
  OB1_EV_CLASS : BYTE ; //Bits 0-3 = 1 (Coming event), Bits 4-7 = 1 (Event class 1)
  OB1_SCAN_1 : BYTE ; //1 (Cold restart scan 1 of OB 1), 3 (Scan 2-n of OB 1)
  OB1_PRIORITY : BYTE ; //1 (Priority of 1 is lowest)
  OB1_OB_NUMBR : BYTE ; //1 (Organization block 1, OB1)
  OB1_RESERVED_1 : BYTE ; //Reserved for system
  OB1_RESERVED_2 : BYTE ; //Reserved for system
  OB1_PREV_CYCLE : INT ; //Cycle time of previous OB1 scan (milliseconds)
  OB1_MIN_CYCLE : INT ; //Minimum cycle time of OB1 (milliseconds)
  OB1_MAX_CYCLE : INT ; //Maximum cycle time of OB1 (milliseconds)
  OB1_DATE_TIME : DATE_AND_TIME ; //Date and time OB1 started
END_VAR
BEGIN
NETWORK
TITLE =
    
```

Fig. 202

Ahora sólo tenemos que compilar el bloque. Cuando compilamos estas fuentes, se generan automáticamente los bloques que tengamos en la fuente. Tenemos que tener en cuenta que no existan ya como bloques en el **STEP 7**, en el proyecto en el que hemos creado la fuente. Si existen, nos dará un error a la hora de compilar y no se generará el bloque.

Cuando creamos el bloque con una plantilla de bloques, esta palabra de protección ya viene escrita. Veremos que aparece con // delante. De momento está como comentario. Si queremos proteger estos bloques, sólo tenemos que quitar esta doble barra. Si lo que hemos hecho ha sido generar una fuente de un bloque existente en **S7**, deberemos borrarlo antes de compilar para que se pueda generar desde la fuente con los cambios introducidos. Si compilamos la fuente mientras el bloque existe todavía en **STEP 7** nos dará un error en la compilación y no se generará.

4.15 Direccionamiento indirecto

Ejercicio 15: Direccionamiento indirecto



DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Introducción al direccionamiento indirecto

Vamos a ver varios tipos de direccionamiento indirecto. Con este tipo de instrucciones podremos optimizar mucho, programas complejos que sin este tipo de instrucciones se harían larguísimo o muchas veces imposibles de programar en instrucciones independientes. En los ejemplos mostrados en este manual, simplemente veremos la programación de los direccionamientos indirectos de forma sencilla. No haremos ejemplos complejos para que el lector pueda centrarse en lo que realmente hace el direccionamiento sin meterse en ejemplos complejos. Estos ejemplos podrían realizarse con instrucciones independientes sin mucha complicación, pero nos dan una idea de lo que podemos hacer con ejemplos o programas más complejos. Si se entienden bien estos ejercicios, no le costará al usuario ponerlos en práctica cuando se encuentre ante una situación en la que tiene que programar algo realmente complejo con este tipo de direccionamientos. Incluso le será fácil entender otros programas hechos por otros programadores en los que se utilice este tipo de instrucciones.

Veamos y analicemos con los ejemplos varios tipos de direccionamiento indirecto.

Hasta ahora hemos utilizado instrucciones del tipo:

AUF	DB	3
U	T	2
CC	FC	7
L	Z	4

.....

En todas estas instrucciones estamos gastando números enteros que no tienen nada que ver con los bits. Hacen referencia al DB nº 3, o al temporizador nº 2, a la FC nº 7, etcétera.

Veamos cómo podemos sustituir estos números por un direccionamiento indirecto. Vamos a escribir en el lugar donde antes estaba el número, un registro variable que contenga el número que a nosotros nos interese en cada caso. Será un número que podrá ir variando dependiendo de las condiciones.

Como vemos todos estos números son “números enteros”. Los sustituiremos por un registro en el que tengamos un entero. Esto será una palabra. La palabra podrá ser de datos, de marcas, de salidas, de variables locales, etcétera.

Veamos en unos ejemplos muy sencillos como hacemos esta sustitución y comprobemos que funciona.

L	5			AUF	DB	10
T	MW	2		L	20	
AUF	DB	[MW2]	T	DBW	10	
.....				U	T	[DBW10]
					

En el primer ejemplo estamos diciendo que se abra el DB 5. En el segundo ejemplo estamos consultando el T 20. En estos casos, la variable que utilizamos para el direccionamiento indirecto, siempre tiene el mismo valor. Pero podríamos cambiarlo dependiendo de ciertas condiciones en otra parte del programa. En el primer caso, podríamos meter en la MW 2 un 5 si ocurren ciertas condiciones o un 6 si se cumplen otras. De este modo, la instrucción de abrir DB, a veces abriría el DB 5 y otras veces abriría el DB 6.

Vamos a hacer un ejemplo sencillo simplemente para observar cómo funciona este tipo de direccionamiento. Entendemos que no es un ejercicio muy práctico pero es una manera sencilla de poder comprobar cómo funciona este tipo de direccionamiento. En pocas instrucciones podemos hacer la prueba. Normalmente cuando se utiliza este tipo de instrucciones es porque el programa es complejo y por eso lo requiere. En este manual, tratamos de hacer sólo ejemplos muy cortos y sencillos para que el lector conozca las posibilidades del **STEP 7**. Si nos metemos en ejercicios complejos, podemos perdernos en el desarrollo del ejercicio y no entender bien la aplicación a la que se hace referencia.

Vamos a programar unos temporizadores de la siguiente manera:

L	EB	1
T	MW	10
U	E	0.0
L	S5T#5S	
SE	T	[MW10]
U	T	[MW10]
=	A	4.0

Recuerda . . .

Deberemos diferenciar los números que hacen referencia a bits y los que no. El direccionamiento indirecto se utiliza de diferente manera si nos referimos a bits o a números enteros decimales.

Con esto tenemos programados 255 temporizadores que son todas las combinaciones que podemos poner en el *byte* de entradas 1. Si esto lo hiciésemos con el direccionamiento normal visto hasta ahora, tendríamos que programar $5 * 255$ instrucciones para programar todos los temporizadores.

Vamos a la tabla de observar/forzar variables y veremos en cada caso con el temporizador que estamos contando. Podemos observar el T1, el T2, el T3, el T4, etc. y observar que el temporizador que funciona es el que le indicamos con el *byte* de entradas 1. Ya hemos dicho antes que este ejercicio no es muy práctico a la hora de introducirlo en un proyecto. Siempre con la misma entrada, al cabo de 5 segundos se enciende la misma salida. Simplemente podemos observar que tenemos programados 255 temporizadores con muy pocas instrucciones.

Veamos otros tipos de hacer direccionamientos indirectos.

Otras veces hemos utilizado instrucciones de este otro tipo:

```
L    MW    10
T    AB    4
L    DBD   3
T    EW    0
.....
```

En estos casos también estamos gastando números enteros. Estamos accediendo a la palabra de marcas 10 o al *byte* de salidas 4, a la doble palabra de datos 3, etc. La diferencia con los casos anteriores es que en estos casos sí que nos estamos refiriendo a *bits*.

Por ejemplo, al hacer referencia a la palabra de marcas 10, nos referimos a los *bits* desde el 10.0 hasta el 11.7 de marcas.

A la hora de hacer un direccionamiento indirecto tenemos que tener en cuenta que esto son *bits*. Tenemos que diferenciar entre el 3 de decir L T 3 y el 3 de decir L MW 3. En el primer caso no son *bits* y en el segundo son los *bits* desde el 3.0 hasta el 4.7 de marcas. Tendremos que cambiar estos números por unos registros en los que se contenga la dirección de los *bits* a partir de los cuales tenemos que tomar 8, 16 o 32 *bits* dependiendo de si queremos acceder a un *byte*, palabra o doble palabra. Para poner la dirección de los *bits* utilizaremos el formato puntero que ocupa 32 *bits*. Es decir, utilizaremos dobles palabras para hacer direccionamientos indirectos. El formato puntero, lo escribiremos como P#X.Y. Dentro del PLC este formato utiliza 29 *bits* para la parte X y 3 *bits* para la parte Y. Como hacemos referencia a *bits* en la parte Y del puntero, como mucho podremos escribir un 7. Nunca podremos escribir P#3.8 por ejemplo. Por eso con 3 *bits* tenemos suficiente para escribir valores entre 0 y 7.

Veamos unos ejemplos:

Formato de puntero:

Ejemplo 1

```
L    P#4.7
T    MD    2
U    E     0.0
=    A     [MD2]
BE
```

Ejemplo 2

```
L    P#1.0
T    MD    2
L    EB    [MD2]
T    MW    [MD2]
BE
```

En el ejemplo 1 se activará la A 4.7 cuando tengamos activa la E0.0. En el ejemplo 2 estamos transfiriendo lo que tengamos en el *byte* de entradas 1 a la palabra de marcas 1. En este caso, para el direccionamiento indirecto estamos utilizando la misma MD 2 que contiene 1.0. Al cargar las entradas, escribimos L EB Estamos indicando que queremos cargar un *byte*, es decir, 8 *bits* a partir del 1.0. Esto es el *byte* de entradas 1. En la transferencia decimos T MW, Decimos que queremos que transfiera a una palabra de marcas. Esto es decir, a 16 *bits* de marcas desde el 1.0. Esto es la palabra de marcas 1.

Veamos algo más de lo que podemos hacer con este tipo de direccionamiento. Podemos utilizar los registros AR1 y AR2 del PLC. Son dos registros internos parecidos a los acumuladores ACU 1 y ACU 2. En estos registros guardaremos normalmente valores de punteros. Nosotros podemos cargar valores tanto en el AR1 como en el AR2. Para cargar valores en estos registros, primero deberemos cargar el valor en el ACU 1 con la instrucción L como hemos hecho hasta ahora en los ejercicios anteriores. Después escribiremos LAR1 o LAR2 para pasar el valor al registro AR1 o AR2 según queramos nosotros.

Veamos en unos ejemplos cómo podemos utilizar estos registros.

Ejemplo 3

```
L      P#0.3
LAR1
U      E      [AR1,P#1.2]
=      A      4.0
```

Ejemplo 4

```
L      P#0.0
LAR2
L      EB      [AR2,P#1.0]
T      MW      [AR2,P#100.0]
```

En el ejemplo 3 estamos cargando 0.3 en formato puntero. A continuación pasamos este valor al registro de direcciones AR1. Para introducir valores en estos registros, escribimos **LAR1** o **LAR2**. En este caso, y a diferencia de las cargas en los acumuladores, podemos cargar indistintamente en el AR1 o en el AR2. Es a voluntad del programador. Al escribir estas instrucciones, pasa al registro, lo que tengamos en ese momento en el ACU 1.

La instrucción que contiene el direccionamiento indirecto dice: si tenemos activa la entrada (lo que contenga AR1 + 1.2), activaremos la salida A 4.0. En este caso estamos haciendo referencia a la entrada 0.3 + 1.2. o sea la entrada E 1.5.

Al final lo que hace el ejemplo es que si está activa la E 1.5, se activará la salida 4.0.

En el ejemplo 4 estamos utilizando el registro de direcciones AR2. Allí cargamos 0.0. El ejercicio dice: Carga 8 *bits* de entrada desde el (0.0 + 1.0) y transfírelos a 16 *bits* de marcas desde el (0.0 + 100.0). En definitiva, lo que tenemos programado sería:

```
L      EB      1
T      MW      100
```

Ejemplo 5 (incorrecto)

```
L      P#0.3
LAR2
L      EB      [AR2,P#4.0]
T      MB      10
```

Este ejercicio sería incorrecto. Si lo enviamos a la CPU se nos irá a **STOP**. Analicemos lo que “le estamos diciendo” al PLC. Queremos que cargue 8 *bits* de entradas desde el (0.3 + 4.0). O sea, 8 *bits* de entradas desde el *bit* 4.3. Esto serían los *bits* 4.3, 4.4, 4.5, 4.6, 4.7, 5.0, 5.1 y 5.2. ¿Qué *byte* es este? Es un trozo del *byte* 4 y un trozo del *byte* 5. Esto no es ningún *byte* de entradas. Con el direccionamiento indirecto no podemos hacer nada que no hiciésemos antes con el direccionamiento directo. Simplemente lo escribimos de otra forma. Los *bytes*, palabras o dobles palabras, comienzan siempre en el *bit* X.0.

Aquí hemos visto varias formas de hacer direccionamiento indirecto.

Vamos a intentar hacer un ejemplo sencillo con sentido para entender cómo podemos dar uso a este tipo de instrucciones.

4.16 Control de fabricación de piezas

Ejercicio 16: Control de fabricación de piezas 

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Direccionamiento indirecto

Supongamos que queremos controlar la cantidad de piezas que lleva fabricada una empresa. La empresa abre a las 8 de la mañana y queremos controlar la cantidad de piezas que tenemos fabricadas cada 2 horas, hasta las 6 de la tarde.

El control lo llevaremos en un DB. Nos haremos el DB 1 de la siguiente manera:

0.0	Piezas_ocho_horas	INT	0
2.0	Piezas_diez_horas	INT	0
4.0	Piezas_doce_horas	INT	0
6.0	Piezas_catorce_horas	INT	0
8.0	Piezas_dieciseis_horas INT	INT	0
10.0	Piezas_dieciocho_horas	INT	0

Queremos rellenar este DB con un direccionamiento indirecto. Vamos a hacer una sola carga y transferencia para rellenar todas las posiciones. Si lo hiciésemos con direccionamiento directo, necesitaríamos hacer 6 cargas y 6 transferencias para completar este DB.

Vamos a comenzar el programa. Primero tenemos que hacer un contador que nos cuente la cantidad de piezas que llevamos hechas.

OB1

```
U    E    0.0
ZV   Z    1
```

Nosotros haremos que pasen los valores al DB cada 10 segundos en lugar de cada 2 horas para poderlo probar.

Para ello nos hará falta un generador de pulsos de 10 segundos.

Continuamos con el OB1:

```
UN    M    0.0
L     S5T#10S
SE    T    1
U     T    1
=     M    0.0
```

Ahora, la escritura de las distintas casillas del DB las vamos a hacer en la FC1. Entraremos a la FC1 cada vez que se active la marca 0.0. Además tenemos que tener un control de la cantidad de veces que hemos entrado a la FC para desde allí dentro saber a qué casilla tenemos que acceder.

Continuamos con el OB1.

```
U    M    0.0
ZV   Z    2
CC   FC    1
BE
```

Ahora vamos a programar la FC 1.

Lo primero que tenemos que saber es la cantidad de veces que hemos entrado para saber a qué casilla tenemos que acceder. Si es la primera vez que entramos, queremos acceder a la casilla que tiene dirección 2 (serán las 10 de la mañana). Si es la segunda vez que entramos, tenemos que acceder a la dirección 4. Si es la tercera vez que entramos tenemos que acceder a la dirección 6, etcétera.

La cantidad de veces que hemos entrado la tenemos en el contador 2. Necesitamos tener este dato multiplicado por dos. Al utilizar enteros para rellenar el DB 2, las direcciones de los datos van de dos en dos. Un entero ocupa dos *bytes*.

FC 1

```

L    Z    2
T    MW   0
L    MW   0
SLW  1
T    MW   2

```

Para multiplicar por 2, podemos utilizar tanto la multiplicación de enteros como un desplazamiento a la izquierda de una posición. Obtenemos el mismo resultado.

En la MW 2 tenemos 2, 4, 6, 8, dependiendo la vez que estemos entrando. Son las direcciones a las que queremos ir. En realidad las direcciones tienen formato de puntero. Es decir, las direcciones a las que queremos ir son 2.0, 4.0, 6.0, Queremos escribir en 16 *bits* a partir del 2.0, 4.0, 6.0, etcétera.

Para ello vamos a utilizar dobles palabras. Además tendremos que utilizarlas en formato puntero. Nosotros tenemos la parte entera. Nos falta el .0 de cada una de estas direcciones. Esto lo conseguimos desplazando la doble palabra tres posiciones a la izquierda. Recordamos que el formato puntero era 29 *bits* para la parte entera y los 3 últimos *bits* para el .X. En nuestro caso siempre queremos que sea .0 porque las direcciones son 2.0, 4.0 etc. Si hacemos un desplazamiento de la doble palabra, la parte derecha se rellenará con 3 ceros que es justamente lo que buscamos.

```

L    MW   2
T    MD   4
SLD  3
T    MD   8

```

En la MD 8 ya tenemos la dirección que queremos en cada caso. Continuamos la FC 1.

```

AUF  DB   1
L    Z    1
T    DBW  [MD8]

```

Si lo dejamos así, cuando acabe con las seis posiciones que hemos definido, se irá a **STOP**, porque buscará la dirección 12.0 para escribir en el DB y no la encontrará.

Cuando llegue a la posición 10, queremos que vuelva a empezar. Para ello añadimos este pequeño trozo de programa.

```

L    Z    2
L    10
==|
R    Z    1
R    Z    2
BE

```

4.17 Cargar longitud y número de DB

Ejercicio 17: Cargar longitud y número de DB 

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Instrucciones de carga

Tenemos las instrucciones DBLG y DBNO.

La instrucción DBLG lo que hace es cargar la longitud del DB que tenemos abierto en el acumulador. La instrucción DBNO lo que hace es cargar en el acumulador el número de DB que tenemos abierto.

Estos comandos se utilizan con la instrucción de carga delante.

Quedaría de la siguiente manera:

```
AUF DB 3
L DBNO           Estamos cargando un tres en el acumulador.
L 3
==|
.....
```

```
AUF DB 3
L DBLG           Estamos cargando la longitud en bytes del DB3.
L 3
==|
.....
```

En este caso no vamos a hacer ningún ejemplo utilizando estas instrucciones. Simplemente proponemos al lector que cree un proyecto con un DB 3 y que compruebe lo que se carga en el acumulador con estas dos instrucciones.

4.18 Comparar dobles palabras

Ejercicio 18: Comparar dobles palabras

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Comparación de palabras

Hasta ahora hemos visto que para comparar valores teníamos las instrucciones compuestas por el símbolo de comparación y a continuación una I o una R dependiendo de si lo que vamos a comparar son números reales o son números enteros.

Además podemos comparar dos dobles palabras independientemente de que sean enteros o reales.

Para ello utilizaremos el símbolo de la comparación y a continuación una D de doble palabra.

Veamos un ejemplo.

```
L      MD      0
L      MD      4
>D
=      A      4.0
BE
```

Con esto comparamos si la serie de ceros y unos que hay en una doble palabra es mayor o menor que la serie que tenemos en la otra palabra. No importa el formato en el que tengamos las palabras. El PLC comparará como si fueran números enteros. Si lo que tenemos es otro tipo de formato, deberemos ser nosotros quienes decidamos si la comparación es válida o no. Hay formatos que comparando *bit a bit* obtenemos una comparación válida. En cambio con otros formatos no ocurre así por lo que significan los *bits* del formato. Por ejemplo, un tiempo lo podemos comparar *bit a bit* pero un formato de reales no.

4.19 Referencias cruzadas

Ejercicio 19: Referencias cruzadas

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA: Localización en el menú de las referencias cruzadas

Vamos a hacer un pequeño programa utilizando varios bloques de programación, llamadas entre bloques, algún temporizador y algunos simbólicos de los contactos utilizados.

Una vez tengamos el programa hecho y guardado, veremos qué información podemos obtener del sistema en cuanto a referencias cruzadas se refiere.

Vamos a programar los siguientes bloques con las siguientes instrucciones:

OB1

U	E	0.0
CC	FC	1

FC1

U	E	0.1
CC	FC	2

FC2

U	E	1.0
L	S5T#3S	
SE	T	1
U	T	1
=	A	4.0
L	EW	0
T	MW	10

Además vamos a programar una FC a la que no llamemos desde ningún sitio.

FC3

U	E	1.1
=	A	4.7

Ahora vamos a ir a la tabla de símbolos globales y vamos a dar nombre a la E 0.0, a la E 0.1 y a la E 2.7.

	Estado	Símbolo ^	Dirección	Tipo de dato	Comentario
1		INTERRUPTOR_MARCHA	E 0.0	BOOL	
2		PULSADOR_ARRANQUE	E 0.1	BOOL	
3		DISPARO_TERMICO	E 2.7	BOOL	
4					

Fig. 203

Una vez tenemos esto, vamos al Administrador de **SIMATIC**.

Pinchamos encima de bloques en la ventana de **OFFLINE**. Estando aquí vamos al menú de herramientas y dentro de él vamos a referencias cruzadas.

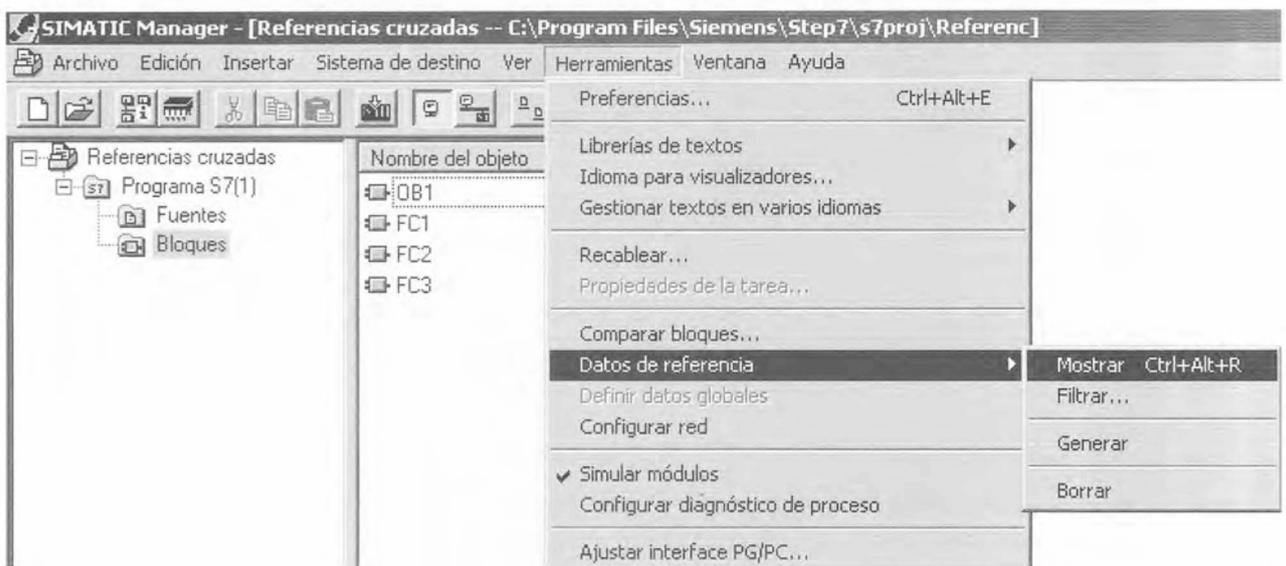


Fig. 204

Dentro de referencias cruzadas, tenemos arriba una serie de botones. Si nos paramos encima de ellos con el ratón, nos va indicando la función de cada uno de ellos.

Operando (símbolo)	Referencias cruzadas (símbolo)	Acc	Lengua	Punto de aplicación			
A 4.0	FC2	W	AWL	Seg	1	Ins 5	/=
A 4.7	FC3	W	AWL	Seg	1	Ins 2	/=

Fig. 205

Vamos a ir pinchando en todos y vamos a ir viendo lo que hace cada uno aprovechando el programa que hemos hecho antes.

Operando (símbolo)	Bloque (símbolo)	Acc	Lengua	Punto de aplicación	Punto de aplicación
A 4.0	FC2	W	AWL	Seg 1 Ins 5	/=
A 4.7	FC3	W	AWL	Seg 1 Ins 2	/=
E 0.0 (INTERRUPTOR_...	OB1	R	AWL	Seg 1 Ins 1	/U
E 0.1 (PULSADOR_AR...	FC1	R	AWL	Seg 1 Ins 1	/U
E 1.0	FC2	R	AWL	Seg 1 Ins 1	/U
E 1.1	FC3	R	AWL	Seg 1 Ins 1	/U
EW 0	FC2	R	AWL	Seg 1 Ins 6	/L
MW 10	FC2	W	AWL	Seg 1 Ins 7	/T
T 1	FC2	R	AWL	Seg 1 Ins 4	/U

Fig. 206

Esto es la ventana de las referencias cruzadas. En ella podemos ver todos los elementos que estamos gastando. Vemos en que bloque lo estamos gastando, si el dato es de lectura o de escritura, en el lenguaje que lo estamos utilizando, en el segmento en el que lo tenemos localizado y la instrucción que realiza.

Si pulsamos el siguiente botón tenemos esta otra pantalla:

Entradas, salidas, marcas									
/	7	6	5	4	3	2	1	0	B W D
EB 0							X	X	
EB 1							X	X	
AB 4	X							X	
MB10									
MB11									

Temporizadores, Contadores										
/	0	1	2	3	4	5	6	7	8	9
T0-9		T1								
Z0-9										

Fig. 207

Aquí podemos ver los bits que estamos utilizando, los que estamos gastando como bits sueltos, los que estamos gastando como byte completo y los que estamos gastando como palabra.

Donde nos señala con una equis, es un bit que lo estamos gastando suelto. Además en el lateral derecho, con líneas azules nos indica si estamos utilizando los bytes como tales o como palabras o cómo dobles palabras. Como vemos en el ejemplo se puede dar el caso en el que hayamos utilizado una palabra o un byte como tal, y además hayamos utilizado bits sueltos dentro de esta palabra.

En la parte derecha de la pantalla, vemos los temporizadores y contadores que tenemos utilizados en la aplicación. En este caso vemos que hemos utilizado el T1 y ningún contador. Según vayamos utilizando más temporizadores o más contadores, aquí irán apareciendo más líneas para indicarnos lo que tenemos usado.

Si entramos en la siguiente opción del menú superior, vemos lo siguiente:

Bloque (símbolo), DB de instancia (símbolo)	Datos I	Lenguaje	Punto de aplicación	Datos locales (para bloque)
Programa S7				
OB1 [máximo: 20]	[20]			[20]
FC1	[20]	AWL	Seg 1 Ins 2	[0]
FC2	[20]	AWL	Seg 1 Ins 2	[0]
FC3	[0]			[0]

Fig. 208

Aquí lo que vemos es la estructura en árbol del proyecto. Vemos los módulos que son llamados y desde qué módulos son llamados. En este caso vemos que a la FC1 se le llama de modo condicional desde el OB1. A la FC2 se le llama de modo condicional desde la FC2.

También vemos que tenemos una FC3 que no la llama nadie en este proyecto. Nos lo indica con una cruz, también vemos que no viene de ninguna rama del árbol. En la parte derecha vemos además los datos locales que tenemos definidos en cada bloque. Vemos que no tenemos nada en las FC en cambio sí tenemos datos locales en el OB1. Como vimos en ejemplos anteriores, los OB vienen con una lista de variables temporales en las que hay datos de sistema que podemos utilizar si nos interesan.

En todas las ventanas que hemos ido entrando, vemos lo que tenemos seleccionado en el filtro. El filtro es el botón que lleva dibujado un embudo y un lápiz. Allí le decimos lo que queremos ver (entradas, salidas, marcas, etc.) y además le decimos los *bytes* que queremos ver. Por ejemplo, si queremos buscar *bits* de DB, no aparecen por defecto seleccionados. Debemos entrar en el filtro y decirle que queremos ver los *bits* de los DB.

La ventana del filtro tiene el siguiente aspecto:

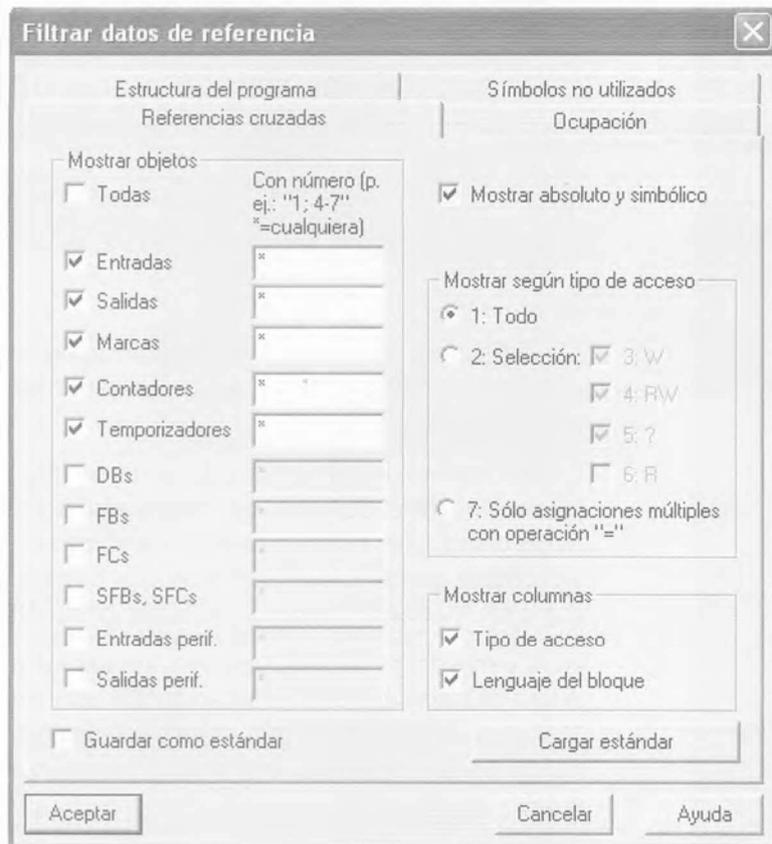


Fig. 209

Aquí vemos que no tenemos los DB seleccionados para su visualización. También vemos que para cada una de las ventanas que hemos visto hasta ahora, tenemos un filtro diferente. Lo que estamos viendo en el ejemplo es el filtro de la ventana de referencias cruzadas. En cada pestaña de esta ventana, podemos seleccionar el filtro que nos interese.

Recuerda . . .

Esta herramienta es muy útil, tanto para hacer modificaciones o ampliaciones de un programa como para diagnosticar errores de una aplicación que ya está funcionando. También tiene gran utilidad a la hora de elaborar un programa para no repetir bits internos, contadores, temporizadores, etc.

Veamos lo que podemos ver en la siguiente ventana.



Fig. 210

Aquí vemos aquellos operandos a los que le hemos dado nombre, pero no hemos utilizado en el proyecto. Esto se supone que será un error. Si le he dado nombre se supone que lo debería haber gastado en el programa.

Si vamos al menú siguiente:

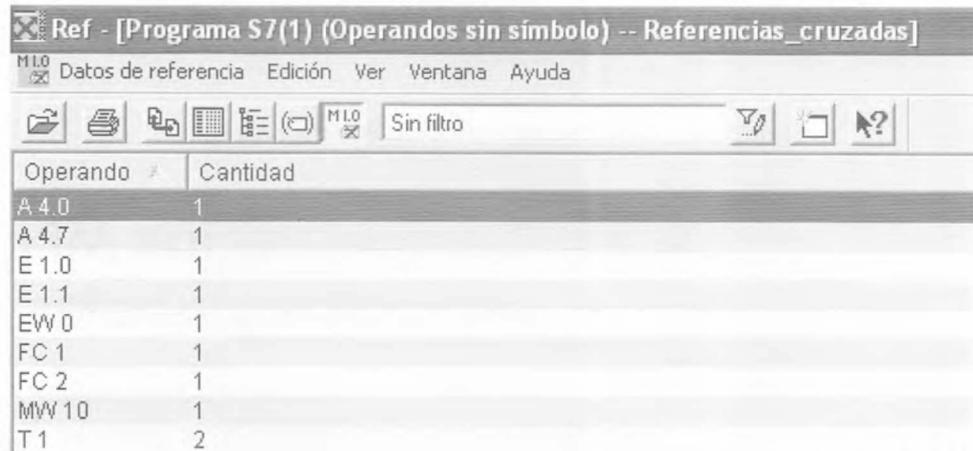


Fig. 211

Aquí vemos lo contrario que en el menú anterior. Vemos los operandos que hemos gastado pero no les hemos dado nombre en el simbólico general.

Si lo que queremos es buscar un operando en concreto, tendremos que volver a la ventana de referencias cruzadas y utilizar el filtro para que nos sea más sencillo de localizar.

Una vez que estemos allí, tendremos que buscar el operando mirando la lista que nos muestra, o bien podemos ir al menú Edición > buscar y escribir el operando que necesitamos consultar.

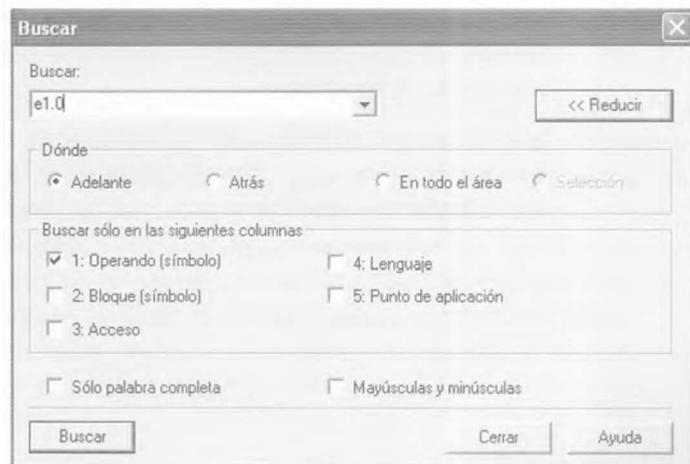


Fig. 212

Aquí seleccionamos el operando que queremos buscar. Podemos escribir tanto el nombre nemónico como el nombre simbólico si es que lo tiene.

Si pulsamos en el botón de buscar, nos buscará en la lista de las referencias cruzadas en todos los lugares donde aparezca el operando. Si queremos acceder a él sólo tenemos que hacer doble clic sobre él y se abre el bloque donde está contenido y con el ratón nos señala allí donde lo estamos utilizando.

Hemos de tener en cuenta que en estas listas no se muestran los operandos que están utilizados en el programa como direccionamientos indirectos. Esto quiere decir que si vemos por ejemplo que la MW 2 está libre, no quiere decir que no se esté utilizando en el proyecto como direccionamiento indirecto. Tendremos que tener esto en cuenta sobre todo si lo que tenemos que hacer es una modificación sobre un programa existente.

4.20 Comunicación MPI por datos globales

Ejercicio 20: Comunicación por datos globales



DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA:

Vamos a montar una red MPI entre varios autómatas y vamos a pasar unos datos entre unos y otros. En el ejemplo que haremos aquí conectaremos 4 PLC con sus correspondientes CPU.

Para ello vamos a tener que hacer un proyecto nuevo en el que insertaremos 4 equipos.

Dentro de un mismo proyecto tenemos que poner todos los equipos. En el menú de insertar, insertamos todos los equipos de los que va a constar la red. Dentro de cada uno de los equipos, vamos a poner el *hardware* de cada uno de los autómatas que vamos a comunicar.

Tendremos que tenerlos todos conectados a una red MPI. Cada uno de ellos deberá tener una dirección diferente. Al crear la CPU en el *hardware*, pinchamos sobre ella con el botón derecho y entramos en sus propiedades.

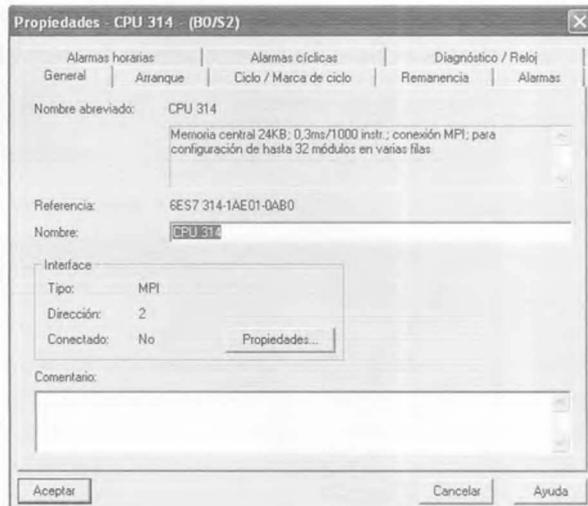


Fig. 213

En este caso vemos que tiene dirección MPI, pero no lo tenemos conectado a ninguna red. Pulsamos el botón de propiedades de la red. Veremos lo siguiente:



Fig. 214

Aquí tenemos que dejar conectada la CPU a la red MPI y asignarle una dirección. En este caso es el primer PLC y le asignamos la dirección 2.

Deberemos hacer lo mismo con el resto de equipos. Deberán estar todos conectados a la misma red y cada uno con una dirección diferente. En el ejemplo les daremos direcciones 2, 3, 4 y 5.

Una vez tengamos definidos los *hardware*, tenemos que transferirle a cada CPU el suyo.

A la hora de transferir, deberemos hacerlo a los equipos uno a uno y sin tenerlos conectados entre ellos. De momento por defecto todos son dirección MPI 2. Si los conectamos antes de cambiarles la dirección, no sabremos a quien estamos transfiriendo cada *hardware*. A la hora de transferir veremos que el sistema nos pregunta a quien queremos enviar la información. Nos aparece una lista de los equipos disponibles. La primera vez que hagamos la transferencia, siempre nos ofrecerá cargar a la dirección MPI 2. Es la que tienen por defecto. Una vez hecha la transferencia del *hardware*, ya podremos conectar todos los equipos en red y, al a hora de transferir, podremos distinguirlos por su dirección.

Una vez tenemos los equipos en el mismo proyecto y con su dirección MPI, tenemos que decirles los datos que queremos que se intercambien.

Veamos gráficamente cómo hacemos todo esto.

El proyecto generado nos habrá quedado del siguiente modo:

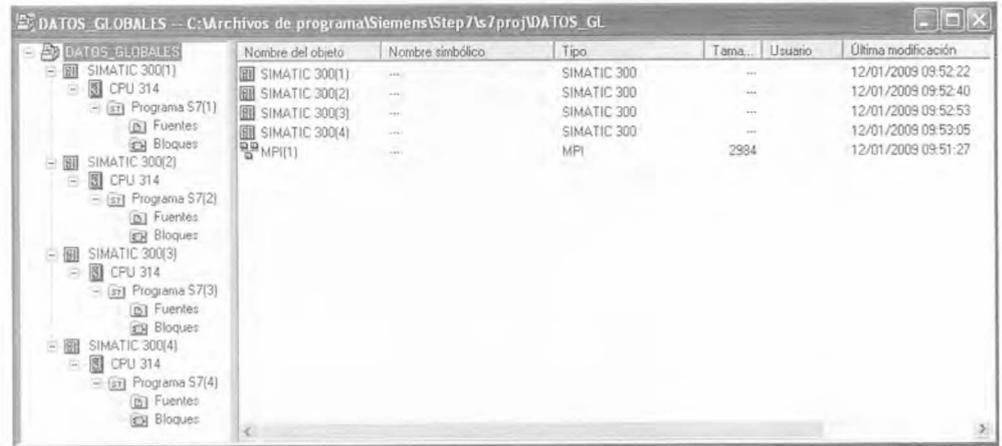


Fig. 215

Si queremos ver los equipos conectados y saber a qué red están conectados cada uno, volvemos al Administrador de **SIMATIC**. Pinchamos encima del nombre del proyecto. En este caso del ejemplo, nos situamos encima de donde pone "DATOS GLOBALES".

Veremos que en la parte izquierda aparece un icono con el nombre de la red. Si hubiésemos creado más de una red, tendríamos varios iconos con los nombres de las diferentes redes.

Si hacemos doble clic en uno de ellos entraremos en un *software* que se llama NETPRO. Es un *software* que viene incluido en el **STEP 7**. Nos sirve para hacer el tratamiento de las redes de forma gráfica. Aquí veremos los equipos que tenemos, así como las redes creadas y las conexiones de cada uno de ellos a las mismas. Aquí podremos crear o borrar enlaces de los equipos a las diferentes redes.

Lo que veremos será lo siguiente:

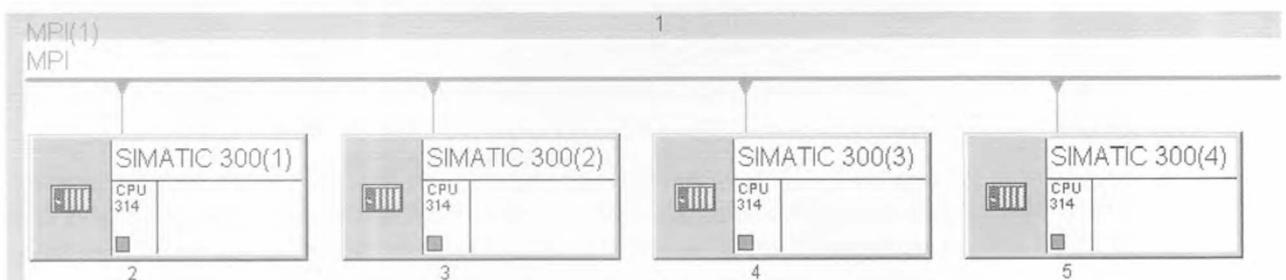


Fig. 216

Aquí veo que tengo 4 equipos conectados a la red uno con la dirección 2, otro con dirección 3, otro con dirección 4 y otro con la dirección 5.

Recuerda . . .

Una red MPI es una red muy sencilla con la que no podemos aspirar a grandes pretensiones. No podemos superar los 100 m de cable, no podemos hacer topologías en árbol ni enviar gran cantidad de datos. Pero, por el contrario, es algo muy fácil de programar que nos puede resolver de forma muy barata alguna aplicación sencilla.

Este *software* no me ofrece nada nuevo que no pueda hacer con el **STEP 7**. Me lo ofrece de modo gráfico y más sencillo. Si tuviera varias redes y quisiera cambiar la comunicación de uno de los equipos, en **STEP 7** tendría que entrar en las propiedades de la CPU, posteriormente en las propiedades de las redes y cambiar la red de conexión.

Si quiero hacer esto desde el NETPRO. No tengo más que arrastrar con el ratón el enlace correspondiente y situarlo donde yo quiera.

También puedo hacer aquí las conexiones que no haya hecho desde el *hardware*. Por ejemplo, si me he dejado una CPU sin conectar y sin cambiar la dirección MPI, aquí la podría conectar de modo gráfico con el ratón. Pulsando sobre la CPU con doble clic, accedo al mismo menú de propiedades que desde la definición de *hardware*. Todos los cambios que haga aquí, tendré que transferirlos a las CPU para que sean efectivos. Vemos que desde la ventana del NETPRO, tenemos el mismo botón de transferir que desde el Administrador de **SIMATIC**.

Una vez tenga todos los equipos conectados a la red MPI 1, tendré que decirles los datos que quiero que se comuniquen.

Para ella volvemos al Administrador de **SIMATIC**.

Pinchamos en la parte izquierda encima del nombre del proyecto. A la parte derecha tenemos un icono que pone MPI. Pinchamos una vez encima del icono de manera que quede seleccionado y entonces vamos al menú de herramientas y entramos en **DEFINIR DATOS GLOBALES**.

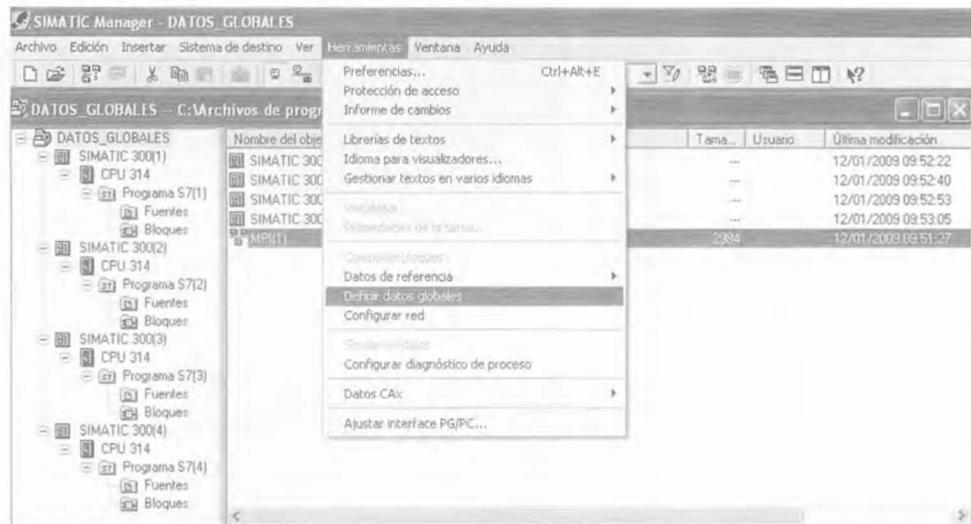


Fig. 217

Si entramos en esta opción lo que tenemos es una tabla en la que tendremos que definir los datos que cada equipo va a emitir a la red, y los equipos que van a recibir los datos.

La tabla que vemos es la siguiente:

	Identificador GD				
1	GD				
2	GD				
3	GD				
4	GD				
5	GD				
6	GD				
7	GD				
8	GD				
9	GD				
10	GD				
11	GD				
12	GD				
13	GD				
14	GD				

Fig. 218

De momento vemos que tenemos la tabla de datos vacía. Tenemos que asignar un equipo a cada columna.

Para ello hacemos doble clic sobre la parte gris de la primera columna. Elegimos la primera CPU de las que se tienen que comunicar. A continuación hacemos doble clic sobre la parte gris de la segunda columna. Elegimos la segunda CPU. Así sucesivamente hasta que tengamos todos los equipos.

La tabla quedará como vemos a continuación:

	Identificador GD	SIMATIC 300(1)\ CPU 314	SIMATIC 300(2)\ CPU 314	SIMATIC 300(3)\ CPU 314	SIMATIC 300(4)\ CPU 314
1	GD				
2	GD				
3	GD				
4	GD				
5	GD				
6	GD				
7	GD				
8	GD				

Fig. 219

Ahora, tenemos que decirles qué datos queremos que se comuniquen.

Queremos que las entradas de una CPU salgan por las salidas del otro y viceversa. Además queremos que algunas palabras de marcas pasen de una CPU a otra.

Veamos como quedaría la tabla rellena con unos datos de comunicación como ejemplo.

	Identificador GD	SIMATIC 300(1)\ CPU 314	SIMATIC 300(2)\ CPU 314	SIMATIC 300(3)\ CPU 314	SIMATIC 300(4)\ CPU 314
1	GD	>BWD	AW4		
2	GD	AW4	>BWD		
3	GD	>MW0	MW10	MW10	MW10
4	GD			>BWD	AW4
5	GD			AW4	>BWD
6	GD		MW20	>MW100	MW20
7	GD				
8	GD				

Fig. 220

Para señalar quienes son emisores y quienes son receptores, tenemos arriba dos iconos con forma de rombo. Tenemos un rombo con una flecha que sale y otro con una flecha que entra.



Fig. 221

El que tiene la flecha que sale es el emisor y el que tiene la flecha que entra es el receptor.

En cada línea de la tabla establecemos una comunicación entre las CPU que queramos. En el ejemplo vemos que en la primera línea decimos que lo que tenga la primera CPU en su palabra 0 de entradas, queremos que salga por la palabra de salidas 4 de la segunda CPU. En la tercera línea de comunicaciones, estamos diciendo que lo que tenga la primera CPU en la palabra de marcas 0, queremos que vaya a la palabra de marcas 10 de cada una de las otras tres CPU.

Una vez tenemos la tabla rellena, tenemos que compilarla para comprobar que no tienen errores.

Para ello pulsamos el siguiente icono:



Una vez compilada, tenemos que transferirla a cada una de las CPU.

Podemos hacer la transferencia de golpe a todas las CPU o una a una.

Si transferimos a las cuatro CPU de golpe, necesitaríamos un cable con el que poder comunicar todos los equipos a la vez. Si no tenemos el cable, también lo podemos hacer. Veremos que a mitad de la transferencia, nos da un aviso de que no encuentra una de las CPU. En ese momento cambiamos el cable a la otra CPU y termina la transferencia.

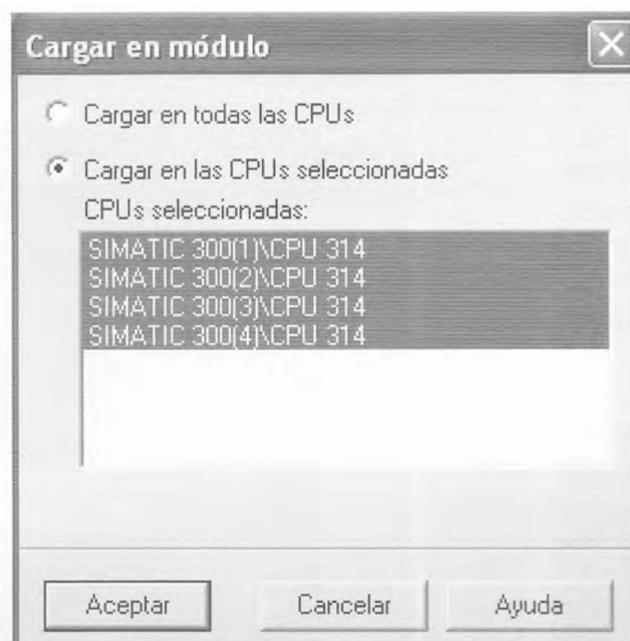


Fig. 222

Ahora ya lo tenemos todo hecho. Sólo nos queda comunicar los autómatas.

Con un cable MPI unimos las cuatro CPU y ya tiene que funcionar.

Lo que hemos hecho es simplemente una transferencia de datos. Cada una de las CPU sigue ejecutando su programa. Sigue ejecutando cada uno lo que tuviera en su OB1.

Vamos a probar ahora a comunicar las entradas y salidas analógicas de dos de las CPU.

Queremos que la entrada analógica de uno salga por la salida analógica del otro y viceversa.

Para ello, en principio tendríamos que hacer el mismo proceso de antes.

Ahora ya no podemos hacer la comunicación directamente como hemos hecho antes. En la tabla de datos globales no podemos poner las entradas analógicas ya que éstas se tratan como datos de periferia. Tendremos que hacer la comunicación a través de marcas.

Para ello tendremos que poner un OB1 en cada una de las CPU.

Las dos OB que tenemos que programar son las siguientes:

OB1				OB1			
L	PEW	288		L	PEW	288	
T	MW	100		T	MW	200	
L	MW	10		L	MW	20	
T	PAW	288		T	PAW	288	
BE				BE			

De este modo lo que tenemos que transferir de uno a otro serán las palabras de marcas.

El primero emitirá por la MW 100 y recibirá por la MW 10. El segundo emitirá por la MW 200 y recibirá por la MW 20.

Si hacemos el mismo proceso de antes, veremos que al mover el voltímetro de un autómata, se mueve el amperímetro del otro y viceversa.

4.21 Red PROFIBUS DP Periferia descentralizada

Ejercicio 21: Red PROFIBUS



DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA:

Vamos a crear una red **PROFIBUS** en la que vamos a conectar un variador de frecuencia. Si lo que queremos conectar es cualquier otro equipo. Los pasos a seguir serán exactamente los mismos. Únicamente deberemos conocer los *bits* que tenemos que enviar y recibir del equipo y lo que significan cada uno de ellos. Este manual no está dedicado a las comunicaciones. Por lo tanto, simplemente es un ejemplo de cómo se configura una red sencilla y qué funciones tenemos que utilizar para programar la transferencia de datos. Se verá que el ejemplo “no está acabado”. No se entra en detalle de cómo funciona el variador ni qué datos tenemos que enviar o recibir. Una vez configurada la red, se dan las nociones de cómo se debería programar el envío y recepción de datos. Ya depende del usuario y del equipo que conecte en **PROFIBUS**, el terminar el ejemplo y hacer que funcione.

Lo primero que necesitamos para hacer una red **PROFIBUS** es que nuestra CPU tenga un puerto **PROFIBUS**. Si estamos trabajando con una CPU 314, no tenemos salida **PROFIBUS**. Deberemos utilizar una CPU con puerto DP. Por ejemplo, una CPU 315-2DP.

Si tenemos cualquier otra CPU que no disponga de puerto **PROFIBUS**, podremos hacer la red añadiendo a nuestra configuración un módulo adicional de comunicaciones. Por ejemplo, una CP-342-5. Aunque no lo tengamos, podemos simular que lo tenemos para hacer las pruebas y generar el proyecto. Lo que no podremos será probar los resultados.

Si tenemos puerto integrado o tarjeta de comunicaciones, la programación será un tanto diferente. Vamos a hacer este ejemplo utilizando una tarjeta adicional de comunicaciones. En este manual, no se trata el tema de las comunicaciones. Para entrar en profundidad sobre estos temas se recomienda consultar un manual de **PROFIBUS**.

Hacemos un proyecto nuevo. Entramos en la configuración del *hardware*. Allí ponemos todo lo que tiene nuestro automático y además la añadimos una CP 342-5. Este es el módulo de comunicaciones para **PROFIBUS** de la serie 300.

Al crear el *hardware* y poner la CP de **PROFIBUS**, automáticamente nos pregunta por la red **PROFIBUS** y la dirección que queremos asignar.

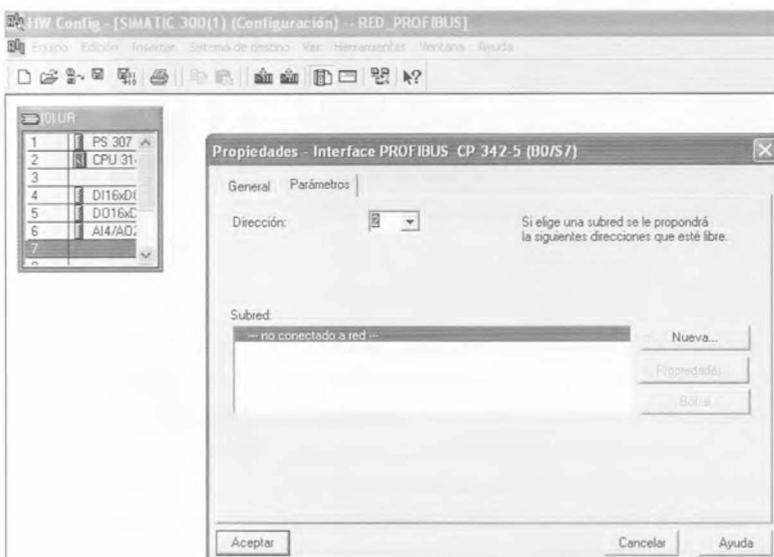


Fig. 223

Por defecto no tenemos ninguna red **PROFIBUS** creada. En este diálogo le diremos que queremos generar una red nueva. Se nos preguntará lo siguiente:

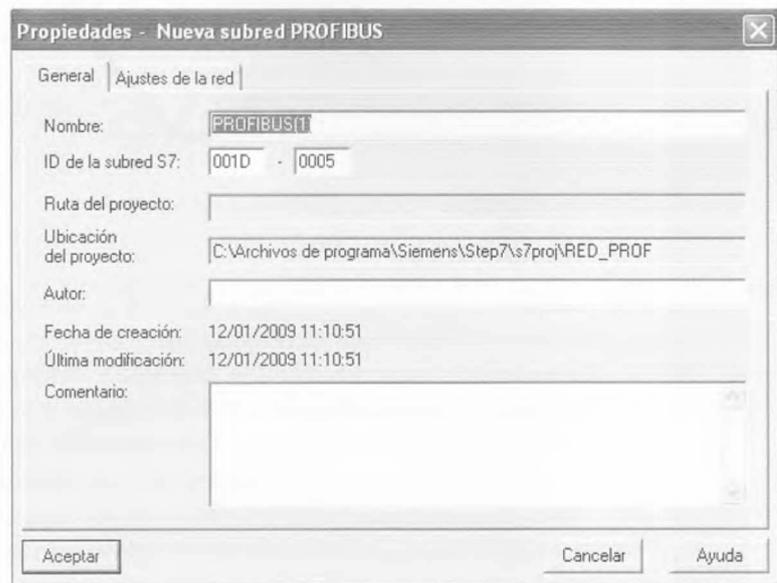


Fig. 224

En esta primera ficha le podemos dar nombre a la red que estamos creando. En el ejemplo dejamos el que nos viene por defecto. Si vamos a la segunda ficha de ajustes de la red, veremos lo siguiente:

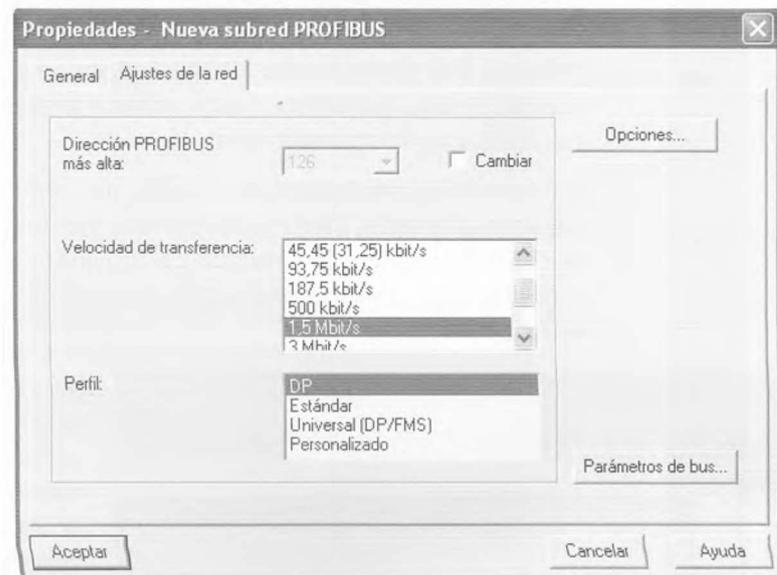


Fig. 225

Aquí tenemos que asignar una velocidad de red. Tendremos que tener en cuenta que la velocidad que asignemos aquí deberá ser la misma para todos los equipos. En principio el sistema ya lo tiene en cuenta. Si conectamos diferentes equipos a una misma red y tratamos de darles velocidades diferentes, nos avisará de que no es posible. Tenemos que tener en cuenta la velocidad máxima de comunicación que permite cada equipo. La velocidad nos vendrá marcada por la velocidad del equipo más lento. En el ejemplo hemos seleccionado *1,5Mbits/seg.*

Recuerda . . .

La red de pro-
fibus DP la po-
demos hacer
utilizando una
CPU con un
segundo puer-
to PROFIBUS o
poniendo una
tarjeta de co-
municaciones
válida para
hacer comuni-
cación DP.

También nos pregunta el perfil de la red que queremos. Al seleccionar un perfil, el sistema ajusta los parámetros del *bus* para optimizar las comunicaciones. No es necesario que tomemos un perfil estándar. Podemos entrar en parámetros de *bus* y asignar nosotros los parámetros que creamos conveniente. Para esto deberemos saber bastante de redes. Si no tenemos tantos conocimientos, el sistema nos proporciona unos perfiles estándar. Si mayoritariamente tenemos equipos en periferia descentralizada, elegiremos perfil DP. Si lo que tenemos son comunicaciones en FDL (no tratadas en este manual), seleccionaremos perfil estándar. Si lo que tenemos mayoritariamente son comunicaciones en FMS (no tratadas en este manual), seleccionaremos perfil Universal (DP/FMS). Con esto el sistema nos hará los ajustes de *bus* optimizados a nuestra comunicación. En nuestro caso vamos a conectar un variador de frecuencia a periferia descentralizada. Por lo tanto, seleccionamos perfil DP.

Una vez hayamos aceptado, veremos que tenemos un *hardware* como este:

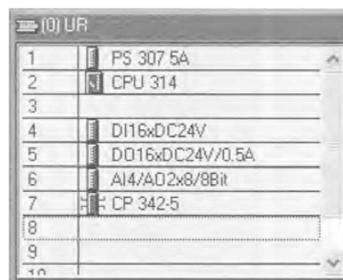


Fig. 226

A este módulo le tenemos que decir que va a ser el maestro de la red.

Para ello tenemos que entrar desde el *hardware* en las propiedades de la CP. Pulsamos sobre ella con el botón derecho y entramos en propiedades. Entramos en modo de operación y le decimos que va a trabajar como maestro DP. Veremos que por defecto pone sin DP.

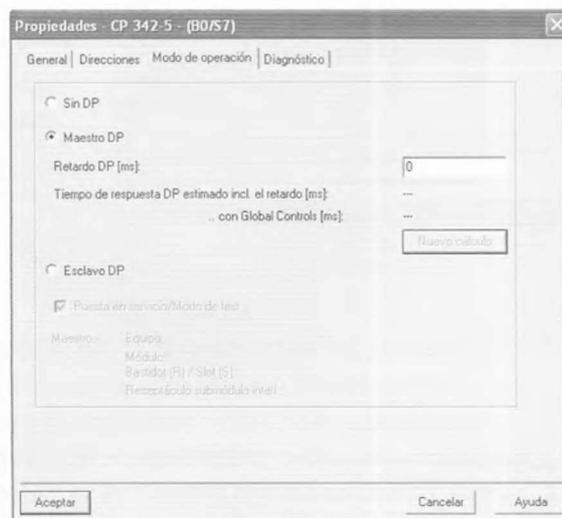


Fig. 227

Al decirle que es un maestro, ya sabe que va a tener unos esclavos. Al cerrar esta ventana, veremos que en el *hardware* nos muestra una red de momento sin nada conectado.

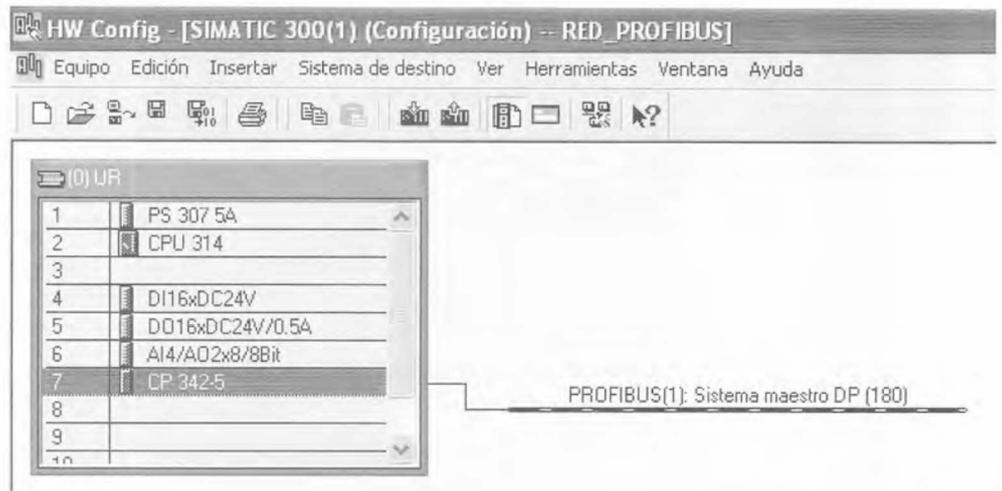


Fig. 228

Para ponerle los esclavos, vamos al catálogo y entramos en el menú de **PROFIBUS**. Allí tenemos todo lo que podemos colgar del maestro. En principio aparecerán los equipos de **SIEMENS** que son los que vienen con el **STEP 7** al instalarlo. Si lo que queremos conectar es de otra marca, deberemos instalar primero el fichero GSD del equipo. Todos los equipos que son para **PROFIBUS**, el fabricante los suministra con el fichero GSD. En la ventana del propio *hardware*, en el menú de herramientas, tenemos la opción de instalar archivo GSD. Todo lo que vayamos instalando, lo veremos también en el catálogo general aunque no sea de **SIEMENS**.

Para realizar el ejemplo, vamos a coger dentro del menú de SIMOVERT, un *micro master*.

Lo cogemos del catálogo y lo arrastramos hasta colgarlo de la línea que tenemos dibujada.

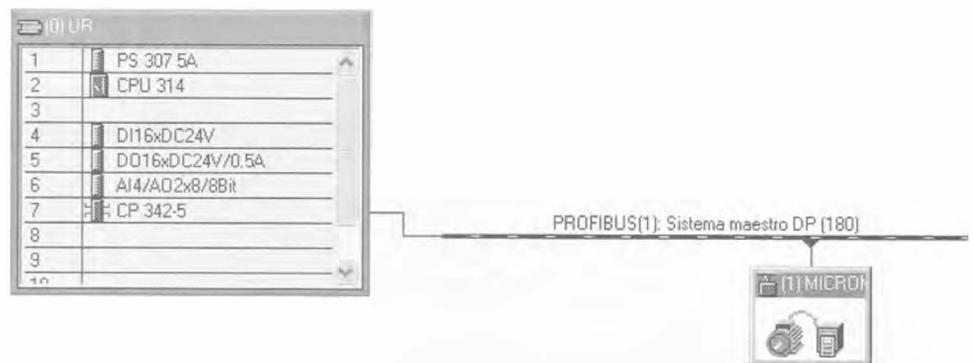


Fig. 229

Ya tenemos la red montada. Al variador de frecuencia le hemos asignado la dirección **PROFIBUS 1**. La CP tenía la dirección 2. Ya hemos terminado la configuración. Si pulsamos sobre el variador, en la parte inferior de la ventana del *hardware*, observamos que tenemos que asignarle los *bytes* de comunicaciones que vamos a utilizar. En el catálogo dentro del variador seleccionado, tenemos varios modos de comunicación. Para el ejemplo seleccionamos 4PKW, 2PZD (PP01). Aquí indicamos los *bytes* que queremos comunicar con el variador. Es un tema que no se trata en este manual. Con un variador, podemos enviar datos de marcha, paro frecuencia, etc. para funcionamiento, o podemos además querer cambiar parámetros del equipo a través de **PROFIBUS**. Con el protocolo elegido, sólo podremos enviar y leer datos de mando y no parámetros.

La configuración del variador quedará así:

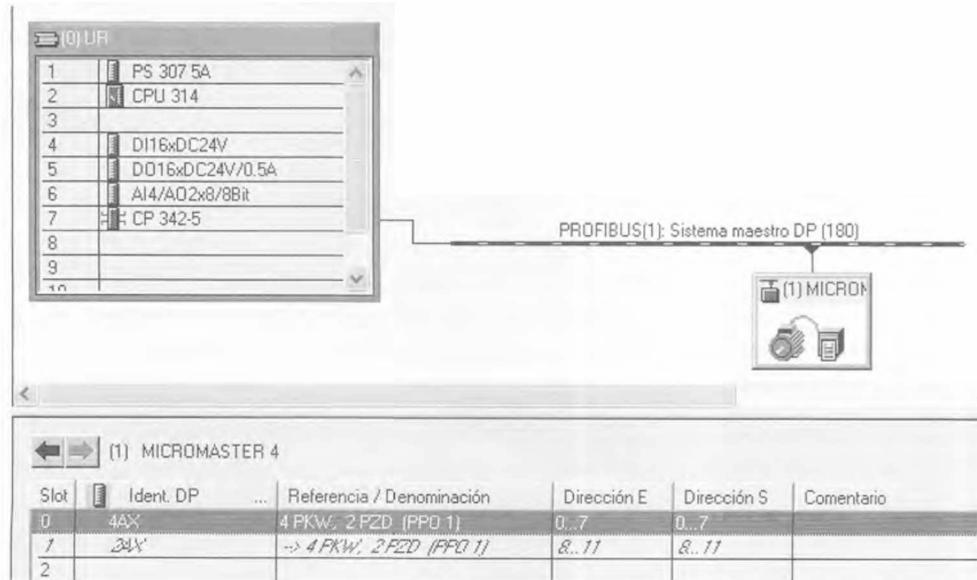


Fig. 230

Transferimos al autómatas la configuración y cerramos la ventana.

Volvemos al Administrador de **SIMATIC**. Para enviar y recibir datos a través de la tarjeta de comunicaciones, deberemos utilizar unas funciones que ya nos vienen programadas en las librerías estándar que ya hemos visto en otros ejemplos de este manual.

Abrimos las librerías estándar y dentro de la carpeta de "communication blocks", encontramos dos FC llamadas DP_SEND y DP_RECIVE.



Fig. 231

Si pinchamos con el interrogante de ayuda encima de las dos funciones, podremos ver lo que hace cada una de ellas y además un ejemplo de cómo utilizarlas.

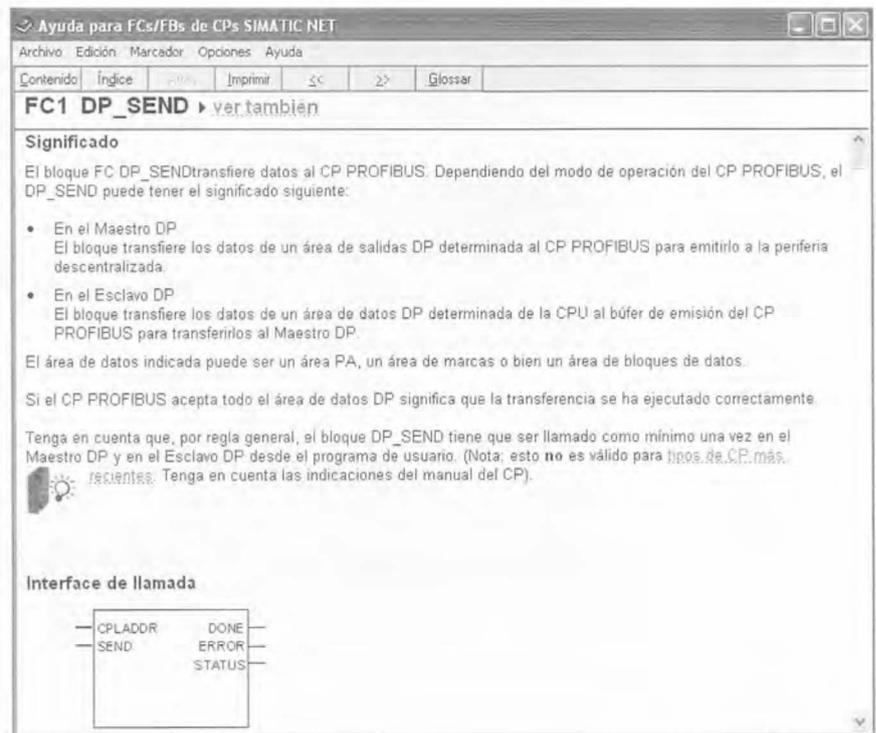


Fig. 232

Nosotros tendremos que arrastrar las dos FC a nuestro proyecto.

En estas FC no podemos entrar para verlas. Son bloques protegidos. Lo que podemos hacer es utilizarlas según hemos visto en la ayuda de cada una de ellas.

Vamos a crear un OB1. Desde aquí vamos a llamar a las dos FC.

Hemos visto que son FC con parámetros. En consecuencia la llamada que tendremos que hacer será un CALL.

Al hacer las llamadas a las dos FC de comunicaciones, nos quedará un OB como el siguiente:

```
OB1 : "Main Program Sweep (Cycle)"
```

Comentario:

Segmento: Título:

Comentario:

```
CALL "DP_SEND"
CPLADDR :=W#16#130
SEND   :=P#DB1.DEX 0.0 BYTE 12
DONE   :=M0.0
ERROR  :=M0.1
STATUS :=M10
```

```
CALL "DP_RECV"
CPLADDR :=W#16#130
RCV     :=P#DB2.DEX 0.0 BYTE 12
NDR     :=M20.0
ERROR  :=M20.1
STATUS :=M30
DPSTATUS:=M40
```

Fig. 233

Analicemos lo que estamos haciendo con estas dos llamadas a las FC de comunicaciones. En el parámetro CPLADDR tenemos que escribir la dirección de la tarjeta por la que queremos comunicar. En un mismo PLC podemos tener varias tarjetas de comunicaciones y establecer diferentes enlaces con diferentes equipos. Tenemos que especificar en cada caso por dónde queremos enviar y recibir los datos. Las tarjetas de comunicaciones, llevan el mismo direccionamiento que las tarjetas de analógicas. Dijimos al principio de este manual que las direcciones de analógicas empezaban por la dirección 256 y se reservan 16 *bytes* para cada posición. En nuestro ejemplo tenemos la tarjeta de comunicaciones en la posición 4. En direcciones de analógicas corresponde a la dirección $256+16+16+16=304$. En el parámetro de la FC nos pide esta dirección en hexadecimal. Si escribimos 304 en hexadecimal, obtenemos 130 que es lo que hemos escrito en el ejemplo.

En los parámetros *SEND* y *RECIVE*, tenemos que escribir lo que queremos enviar y recibir por esta tarjeta. Tal y como vimos al crear el *hardware*, necesitamos 12 *bytes* para comunicarnos con el variador de frecuencia. En el ejemplo nos generaremos dos DB de comunicaciones con 12 *bytes* cada uno. En uno tendremos los datos de envío y en otro tendremos los datos de recepción. Aquí escribimos los parámetros en formato puntero. Para saber lo que significa cada bit o cada byte y poder hacer una aplicación, deberíamos mirar en el manual del variador (o del equipo que estemos conectando ya que todos funcionan igual).

Después tenemos unos *bits* de marcas que nos indican si hay errores en la comunicación y un código de error si es que lo hay. En la ayuda de la función podemos ver lo que significan estos *bits*.

Ahora sólo quedaría hacer la aplicación en la que se activen las palabras de comunicaciones para manejar el variador como nos interese.

4.22 Utilización del simulador de SIEMENS

Ejercicio 22: Utilización del simulador de SIEMENS ✓

DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA

Como aplicación adicional al **STEP 7**, **SIEMENS** dispone de un simulador de CPU. Es un *software* que se vende y se instala independiente del Administrador de SIMATIC, aunque luego se ejecuta desde dentro del mismo. También lleva su licencia independiente. Si tenemos el *software* instalado, se nos activará el botón del simulador en el Administrador de **SIMATIC** como vemos en el ejemplo.



Fig. 234

Si no tenemos el simulador instalado, este botón aparecerá desactivado (en color gris).

Para utilizar el simulador tenemos que tener el botón pulsado. Al pulsarlo veremos una pantalla como la que mostramos a continuación:

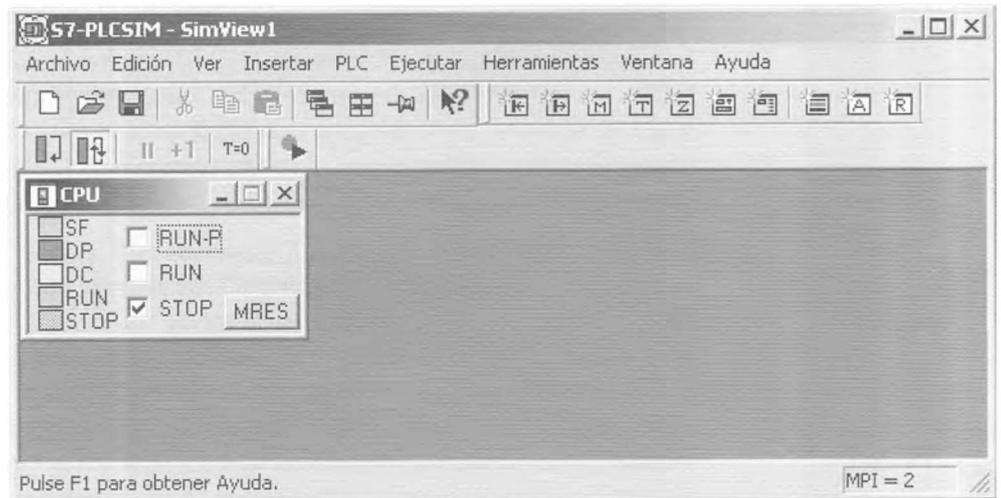


Fig. 235

Vemos simulado el estado de la CPU. Vemos las luces indicadoras y mediante los *check list* podemos simular la llave de los PLC. En este caso tenemos la CPU en **STOP**. También nos indica que tenemos fallo de **PROFIBUS**.

Con un clic del ratón podemos poner la CPU en **RUN** o en **RUN-P**.

Veamos cómo simularíamos un programa que hagamos en el **STEP 7**. Nos creamos un proyecto nuevo y en el OB 1 hacemos la siguiente programación:

```

OB1 : "Main Program Sweep (Cycle)"
Comentario:

Segm. 1: Título:
Comentario:

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

Fig. 236

Ahora para transferirlo a la CPU, deberemos tener en cuenta que no queremos que salga del ordenador. Queremos enviar el programa al simulador. Para ello en los ajustes del driver deberemos seleccionar "Ninguno".

Para ello vamos a la ventana de “Ajustar interface PC/PG” y hacemos la siguiente selección:

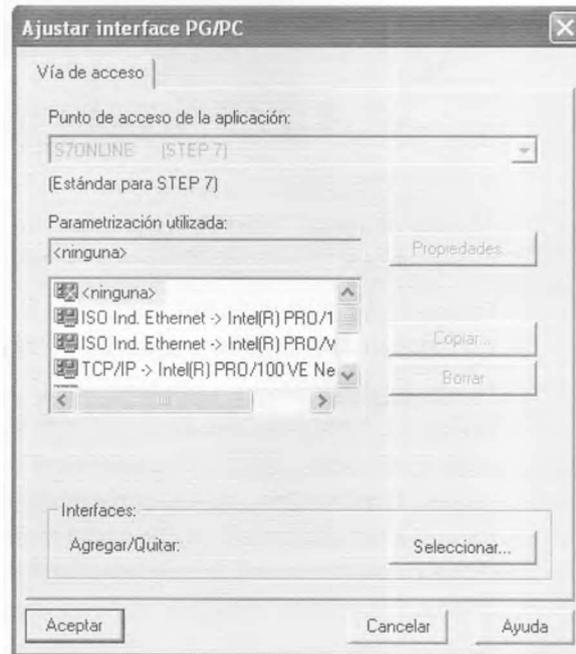


Fig. 237

Una vez tengamos esta selección hecha, podemos volver al bloque OB 1 y pulsar el botón de transferir normalmente como lo hacíamos en los ejemplos anteriores de este manual. El programa se transfiere al simulador.

Ahora vamos a probar que lo que hemos programado funciona. Para ello vamos a la ventana del simulador, y nos insertamos las señales que necesitemos. En este caso hemos gastado *bits* del *byte* 0 de entradas y *bits* del *byte* 4 de salidas. Necesitamos insertar dos *bytes* de señales, direccionarlos tal y como necesitamos y observarlos en modo binario para ver los *bits* uno a uno.

En el simulador vamos al menú “Insertar à entrada”. Nos permite seleccionar el *byte* de entradas que queremos ver y el formato en el que lo queremos ver. Seleccionamos el *byte* 0 en formato “*bits*”. Nos quedará como vemos a continuación:



Fig. 238

Ahora tenemos que insertar un *byte* de salidas. Vamos al menú “Insertar à salida”. En este caso seleccionamos el *byte* 4 y lo representamos en *bits*.

Si ponemos el simulador en **RUN-P** y activamos las entradas E0.0 y E0.1, veremos que se activa la salida A4.0.



Fig. 239

Al abrir el menú “Insertar” podemos observar que se pueden simular tanto entradas y salidas, como marcas temporizadores y contadores.

Con este simulador, nunca podremos simular más de una CPU con sus comunicaciones. Tampoco podremos simular comunicaciones con otros equipos.

También deberemos tener en cuenta que el programa lo está ejecutando el microprocesador del ordenador y no un PLC. Varía mucho la velocidad de un procesador al otro. Veremos que si programamos tiempos y los simulamos, no son tiempos reales. Tampoco se respetan los ciclos de *scan* tal y como se hace en el PLC. Si tenemos un programa con tiempos críticos o maniobras que dependen de pocos ciclos de *scan*, no nos deberemos fiar de la simulación que hagamos con este programa.

En cambio, si no son críticos los tiempos utilizados, podemos utilizar perfectamente el simulador para verificar programas de contactos, contadores, transferencias de datos, tiempos (siempre y cuando sepamos que no son reales), etc. Es una herramienta muy útil para programas sencillos y sin comunicaciones. Podemos asegurarnos de que lo que hemos hecho funciona antes de transferirlo a la máquina o instalación real.

Si hacemos una vista rápida por el menú del simulador, podremos encontrar las siguientes opciones.

Dentro del menú “Archivo”, podemos abrir o cerrar simulaciones y guardarlas en el disco duro.

Dentro del menú edición podemos deshacer la última acción y copiar y pegar datos.

Dentro del menú “Ver”, se nos permite ver otros *bits* internos de la CPU. Podemos ver los acumuladores, la palabra de estado, *bits* de DB y pilas de anidamiento.

En el menú insertar, ya hemos visto que podemos insertar entradas, salidas, marcas, temporizadores, etc. También tenemos una opción de insertar una variable en general que nosotros definiremos.

En el menú “PLC” podemos dar o quitar “tensión” al PLC con las teclas de ON y OFF. Evidentemente siempre simulado. Podemos observar el comportamiento del PLC ante una caída de tensión. También podemos hacer un borrado total de la CPU y asignar una dirección MPI. El borrado total no lo podemos simular con la secuencia de la llave azul que indicamos a principio de este manual, puesto que no tenemos llave. El mismo efecto conseguimos pulsando con el ratón en esta opción.

En el menú ejecutar, también podemos hacer rearranques de la CPU. Podemos cambiar el selector de modo. También podemos ejecutar un solo ciclo de CPU.

En el menú de herramientas (o con el botón que encontramos en la barra de herramientas y que simula los símbolos de grabar o parar), podemos hacer grabaciones de simulaciones de funcionamiento para poder probar nuestro programa de un modo más real. Sin esta herramienta, a veces sería imposible probar un proceso que hemos programado. No daríamos abasto con el ratón a activar y desactivar *bits*. Al tener que hacer clic con el ratón, estamos limitados.

4.23 Realizar copias de seguridad

Ejercicio 23: Realizar copias de seguridad



DEFINICIÓN Y SOLUCIÓN

TEORÍA PREVIA

Es muy importante hacer copias de seguridad en muchas ocasiones. Dependiendo de lo que queramos hacer y de las circunstancias, la manera de proceder será diferente.

1.- Copia de seguridad del programa que estamos realizando.

Cuando nosotros hemos hecho un proyecto y estamos trabajando sobre él, será interesante ir haciendo copias de seguridad indicando la fecha en que la realizamos o la versión del programa por la que vamos. Es interesante guardar las cosas que vamos haciendo por si tenemos cualquier problema con el PC o por si realizamos cambios en el proyecto que terminan por no funcionar bien o no gustarnos. Si tenemos copias de seguridad guardadas, siempre podremos volver a un estado anterior del proyecto.

Para ello tenemos en **STEP 7** dos funciones.

Dentro del menú "Archivo", tenemos la opción "Guardar como" y la opción "Archivar".

La opción "guardar como" funciona como en cualquier otra aplicación de Windows. Guardará el proyecto en la ruta que le indiquemos. Normalmente los proyectos de **STEP 7** ocupan bastante memoria. Por ello tenemos también la función "Archivar".

Para poder archivar un proyecto, deberemos ir al menú "Archivo" y seleccionar "Archivar". Nos aparecerá un diálogo como el siguiente:



Fig. 240

Aquí seleccionamos el proyecto que queremos archivar. A continuación veremos el siguiente diálogo:

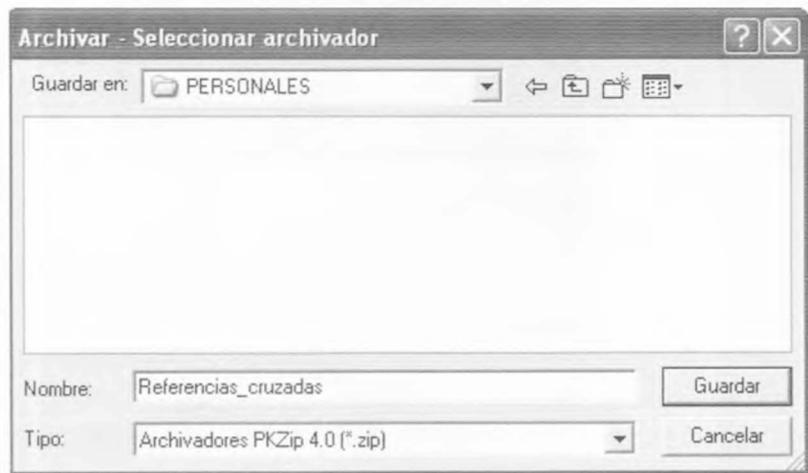


Fig. 241

Aquí podemos seleccionar la carpeta en la que queremos guardar el proyecto, el nombre que le queremos dar al fichero comprimido y el tipo de archivador que queremos utilizar. Podemos comprimir en .zip o en .rar. No es necesario que tengamos ningún *software* de compresión instalado. El **STEP 7** ya lo trae por defecto.

En el nombre del proyecto que aquí utilizamos, es donde podríamos poner la fecha en que la hacemos o la versión del proyecto que estamos guardando. El nombre que damos aquí es independiente del nombre del proyecto. Una vez lo abramos, tendrá el nombre original que se le dio al crearlo en **STEP 7**.

Una vez tengamos seleccionado todo esto y pulsemos aceptar, se nos hará la siguiente pregunta:

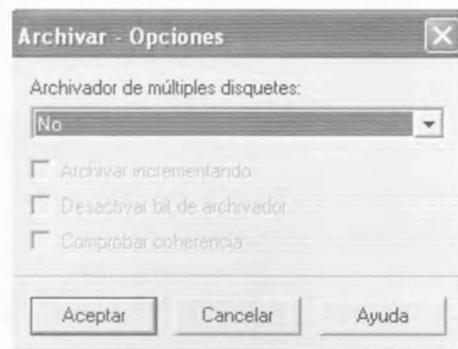


Fig. 242

Le diremos que si, si queremos crear varios volúmenes para guardar en disquetes de 1,44 Mb. Si sólo queremos generar un archivo único, aquí le diremos "no".

Al pulsar aceptar, se hará la compresión y nos generará el archivo. Ya tendremos el proyecto completo guardado y comprimido.

Cuando necesitemos abrirlo, deberemos utilizar la opción "Archivo"-> "desarchivar".

Aquí seleccionamos el proyecto que queremos, lo descomprimimos y si queremos lo abrimos.

Con esto lo que hacemos es guardar o rescatar un proyecto que estaba en nuestro ordenador y que hemos hecho nosotros.

Otro tema diferente será cuando vayamos a una instalación y queramos hacer una copia de seguridad del programa que tiene la máquina.

2.- Copia de seguridad de un PLC **ONLINE**.

En este caso, nosotros no tenemos el proyecto. Nos encontramos ante un PLC que está funcionando pero que no sabemos qué *hardware* tiene, si tiene enlaces de comunicación o los bloques de programa que está usando.

Veamos como procederíamos para hacer una copia de seguridad.

Nos interesa hacer estas copias cuando vamos a modificar una máquina de una instalación. Siempre es conveniente guardar lo que tenemos en el momento en que llegamos a la instalación, por si lo que hacemos no funciona bien, o por si tenemos que poner la máquina en marcha de nuevo antes de terminar con la modificación. También nos interesa hacer copias de seguridad de las máquinas instaladas por si tenemos cualquier avería en ellas y tenemos que cambiar el PLC.

Para generar estas copias de programa, primero deberemos crear un proyecto vacío. En principio no tenemos proyecto y todo lo que copiemos deberá ser sobre un proyecto generado por nosotros.

Para ello desde **STEP 7** vamos al menú Archivo -> Nuevo e insertamos un equipo de la serie que tengamos que hacer la copia. En el ejemplo hemos insertado un equipo 300. El proyecto vacío quedará de la siguiente manera:

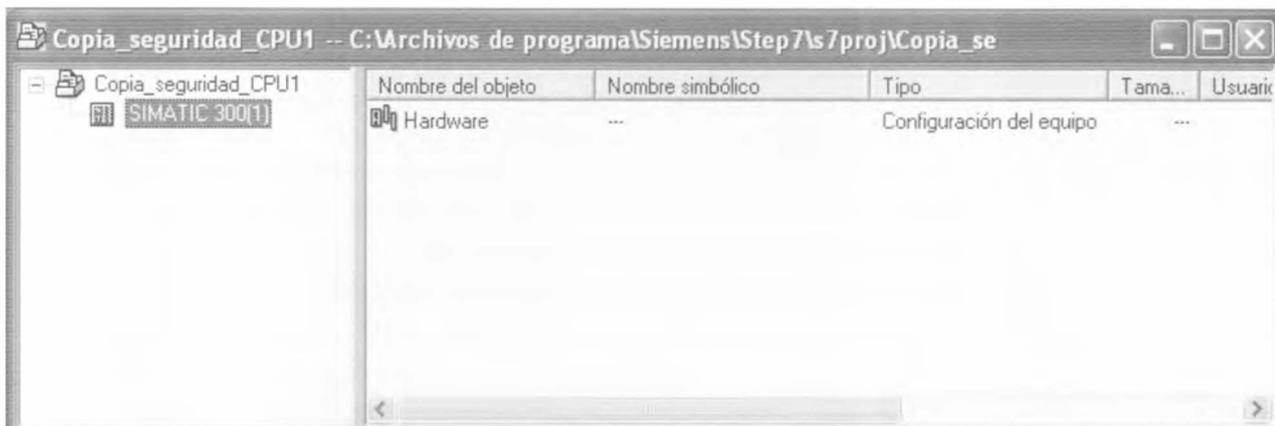


Fig. 243

Lo primero que haremos será copiar el *hardware*. Como hemos visto a lo largo de este manual, un mismo programa no funciona igual en una CPU que en otra que tenga diferentes propiedades. En el *hardware* estaban las marcas de ciclo, la programación de alarmas, configuración de remanencias, etcétera.

Para hacer la copia, entramos en el *hardware* que en principio estará vacío. Pulsamos el botón de leer de la CPU.



Una vez pulsemos este botón, se nos preguntará en que proyecto queremos que lo almacene.

Deberemos seleccionar el proyecto que acabamos de crear vacío. A continuación nos saldrá un diálogo como el que se muestra:

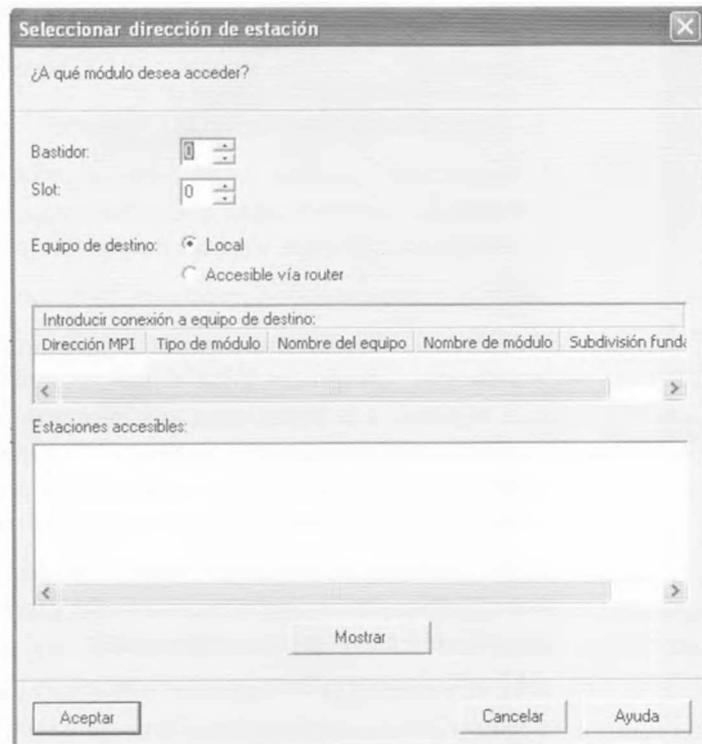


Fig. 245

Aquí se nos está preguntando desde dónde queremos leer el *hardware*. En principio no nos muestra ningún equipo. Podemos dar aceptar si sabemos que no existe ninguna red y que estamos conectados sólo a la CPU de la que queremos hacer la copia. Si donde nos hemos conectado hay más cables que el propio de la CPU, quiere decir que hay más equipos conectados, no sólo la CPU. Podemos tener conectada una pantalla de visualización, perifera descentralizada, otras CPU, etc. Si es el caso, deberemos pulsar el botón "Mostrar".

Nos saldrá entonces una lista de los equipos accesibles en la red que nos hemos conectado. Deberemos seleccionar la CPU de la cual queremos hacer la copia.

A la hora de conectarse, deberemos tener cuidado sobre todo si nos conectamos a través del puerto **PROFIBUS**. Si nos conectamos en MPI no hay problema. No ocurrirá nada al conectar la PG. Pero si nos conectamos a la CPU en **PROFIBUS** y hay más equipos conectados, podríamos provocar un fallo en la CPU incluso llevarla a **STOP**. El ordenador tiene partes metálicas que pueden hacer de antena en la red y hacerla funcionar mal. Esto no quiere decir que siempre que nos conectemos en **PROFIBUS** vaya a funcionar mal la CPU. Dependerá de la red y de las condiciones ambientales. Simplemente hay que tenerlo en cuenta si es una máquina que no puede parar o que un mal funcionamiento aunque sea momentáneo pueda ser peligroso. Para este tipo de conexiones, existe un cable especial con unas resistencias de cierre que nos permiten hacer estas conexiones de modo seguro.

Una vez hayamos seleccionado el equipo del que queremos leer, pulsamos aceptar y hacemos la copia de seguridad de la CPU que tenemos conectada. Una vez volvamos al Administrador de **SIMATIC**, veremos que tenemos 2 equipos. Uno es el que nosotros insertamos para poder acceder a la ventana de *hardware*. El otro es el que hemos leído del equipo **ONLINE**. El que insertamos nosotros, que está vacío, ya lo podemos borrar.

Nos quedará el proyecto de la siguiente manera:

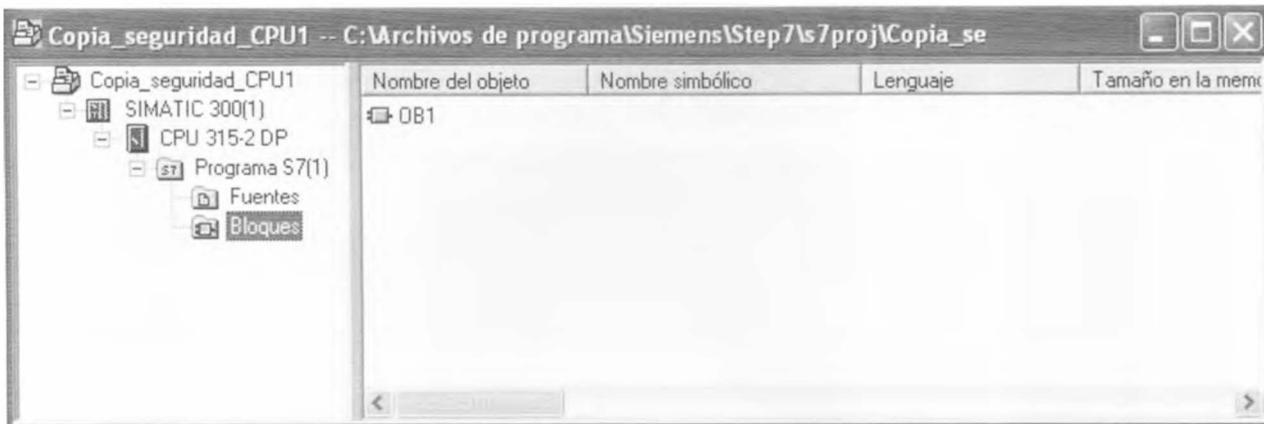


Fig. 246

De momento hemos copiado el *hardware* pero no tenemos programa. Para ello deberemos pulsar el botón de conectarse **ONLINE** y veremos los bloques que tenemos en la máquina.

Ahora podemos arrastrar con el ratón a nuestro proyecto todos los bloques de la CPU **ONLINE**. Veremos que tiene los bloques SFC y SFB propios de la CPU. Éstos no es necesario que los copiemos. Incluso si se están utilizando no es necesario hacer la copia. Todas las CPU llevan estos bloques cargados.

Con esto tendremos el proyecto con el *hardware* y los bloques de programa.

También es necesario ver si tiene algún enlace programado con otros equipos. Para ello vamos al programa NETPRO. Para ello pulsamos el botón que nos da acceso a la configuración de redes:



Fig. 247

Aquí veremos los equipos que tenemos y las conexiones entre ellos si es que las hay. Veamos un ejemplo de un proyecto con más de una red:

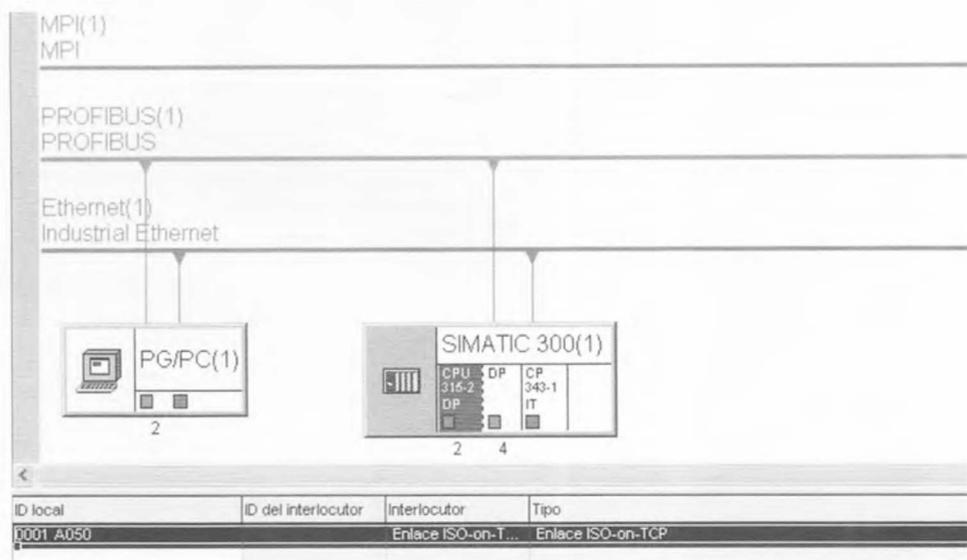


Fig. 248

Recuerda . . .

Para hacer una copia de seguridad completa de un proyecto, siempre deberíamos copiar el *hardware*, los bloques de programa y los posibles enlaces desde el NETPRO.

En el ejemplo hemos representado un proyecto en el que además de la CPU que estamos copiando, hay un PC que está unido a la red **PROFIBUS** y a la red Ethernet igual que la CPU. Al pinchar sobre la CPU, vemos que tiene un enlace configurado. Es importante que incluyamos esto en la copia de seguridad. Si el proyecto lo estamos viendo **ONLINE** y no tenemos copias anteriores, posiblemente no veamos los PC que hay en la red, pero si veremos los enlaces de la CPU que es lo que nos interesa. Debemos pulsar el botón de leer de la CPU para guardar los enlaces si es que existen.

Si al entrar en el NETPRO no vemos más redes que la MPI y no hay nada conectado y la CPU no tiene enlaces, no es necesario que copiemos nada.

Una vez copiadas estas tres cosas: *hardware*, bloques y enlaces, ya tenemos hecha la copia de seguridad.

3.- Comparar bloques.

Analicemos alguna herramienta más que tiene el **STEP 7** y que nos puede venir bien a la hora de enfrentarnos a una máquina. Puede darse el caso de que nos den una copia y nos digan que es la copia del programa que hay en la máquina. Siempre será mejor trabajar sobre un copia original que sobre la que nosotros podamos hacer **ONLINE**. En la copia original tendremos los simbólicos del proyecto y los posibles comentarios que haya hecho el programador. En la copia **ONLINE** no tendremos nada de esto.

Aunque nos den una copia del programa, yo siempre recomiendo hacer una **ONLINE** por lo que podamos encontrar después. Si queremos trabajar con la copia que nos proporcionan, deberemos asegurarnos de que los bloques coinciden con los de la CPU. Para ello tenemos la opción de que **STEP 7** haga la comparación.

Deberemos tener el proyecto abierto y seleccionar la carpeta de bloques. También podemos seleccionar un bloque en concreto. La comparación la hará de lo que tengamos seleccionado a la hora de entrar en el menú de la comparación.

Seleccionando la carpeta de bloques desde el administrador, vamos al menú “Herramientas -> Comparar bloques”

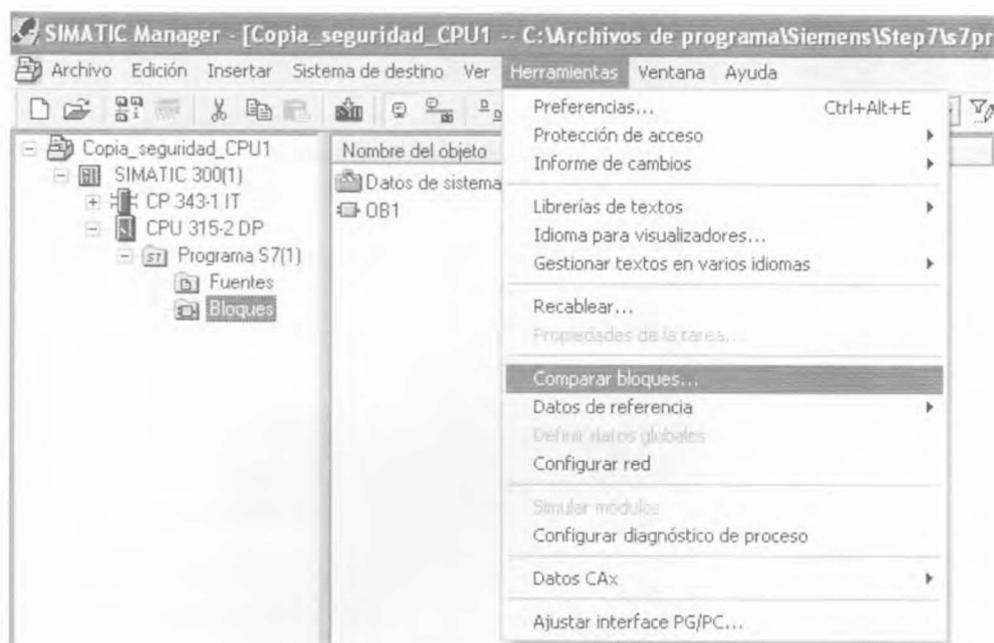


Fig. 249

Al pulsar aquí tenemos la opción de comparar todos los bloques del programa con los que tiene la CPU o con los que tiene otra copia de programa **OFFLINE**.

Al hacer la comparación puede darse el caso de que todos los bloques sean totalmente idénticos. Con lo cual obtendríamos lo siguiente:

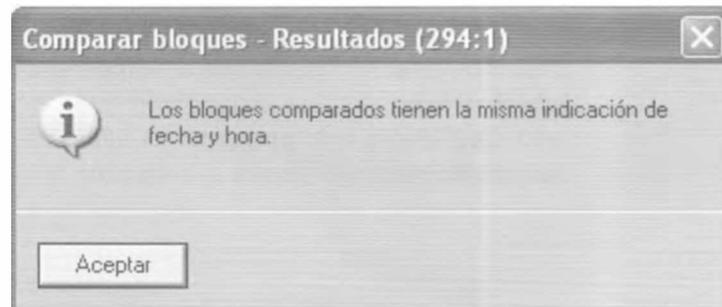


Fig. 250

También puede darse el caso de que haya alguna diferencia en los bloques comparados.

Si fuese el caso, el **STEP 7** nos lo indicaría de la siguiente forma:

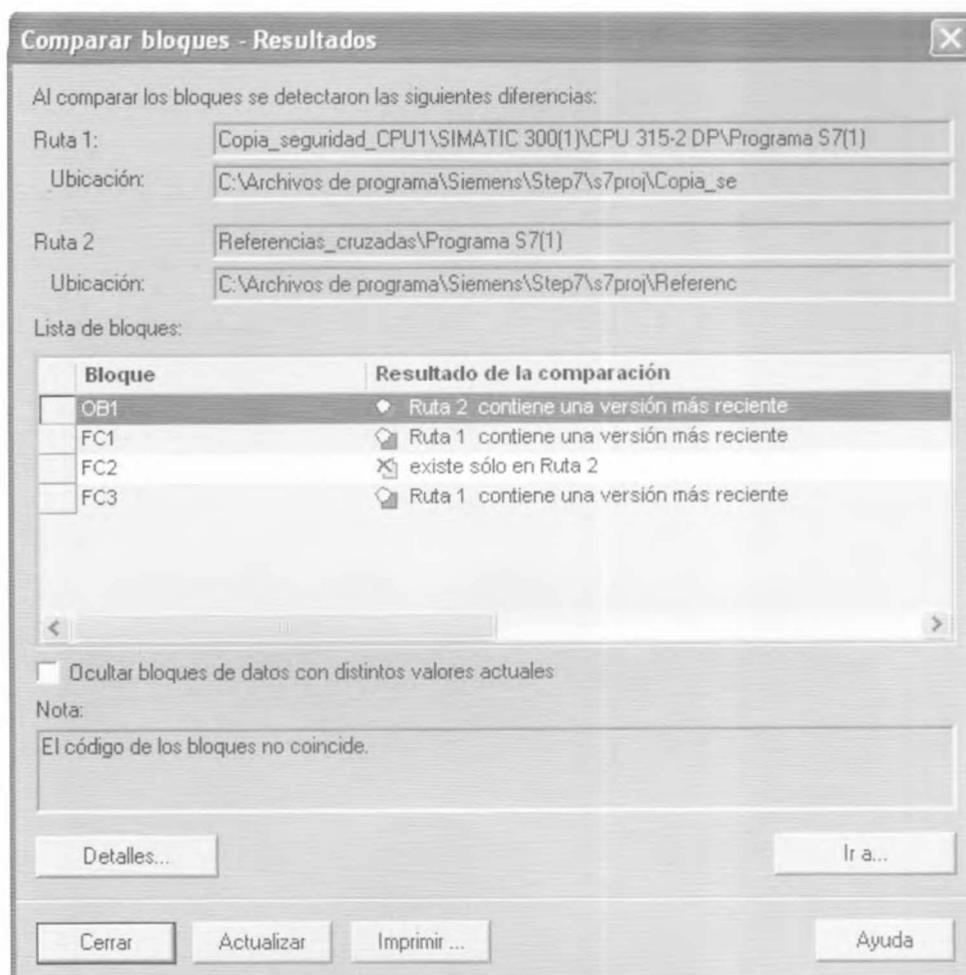


Fig. 251

Algo parecido a esto veremos cuando los bloques comparados no sean iguales. En el ejemplo vemos que tenemos diferencia en los bloques OB1, FC1 y FC3, y que el bloque FC2 sólo coincide en una de las dos rutas comparadas.

En la parte de debajo de esta ventana, nos dice las diferencias encontradas. En este caso nos dice "El código de los bloques no coincide". Además tenemos un botón que dice "Ir a...". Si lo pulsamos, nos abrirá los dos bloques (de las dos rutas seleccionadas) y nos señalará con el ratón donde encuentre las diferencias.

Debemos tener en cuenta que al comparar DB, también se compararán los datos actuales. Normalmente estos datos serán diferentes **ONLINE** y **OFFLINE** sin que esto signifique que el programa es diferente. En **ONLINE** los datos actuales van cambiando según se ejecuta el programa.