

A blurred screenshot of a software interface showing a ladder logic diagram. The diagram consists of several rungs with various colored components (red, cyan, black). A mouse cursor is visible at the bottom right, pointing at one of the rungs. The background is a light blue and white grid.

MANUAL MICROLADDER

EDICION EDITADA PARA NANO LADDER |
DAGEL Sistemas Electrónicos

Índice

1	¿Qué es un Microcontrolador?	4
2	Formas de Programar un microcontrolador.	5
2.1	Editor	5
2.2	Compilador	5
2.3	Grabador.....	5
3	¿Qué es Microladder?	6
4	Funcionamiento de Microladder.	7
4.1	Comunicación entre ML-PC y ML-Chip.....	7
4.2	Modo Run y modo Stop.	7
4.3	La barra de herramientas de Microladder.....	7
5	Primeros pasos con Microladder.	8
5.1	Área de Trabajo	8
5.2	Árbol de Navegación.....	8
5.3	Área Auxiliar.....	8
5.3.1	Ventana “Búsqueda”.....	9
5.3.2	Ventana “Errores”.....	9
5.3.3	Ventana “FD”.....	9
5.3.4	Ventana “ED”.....	9
6	Un ejemplo con Microladder.	10
6.1	Parpadeo de un Lámpara cada segundo.....	11
6.1.1	Configuración de los Pines	11
6.1.2	Creación de las variables	12
6.1.3	Edición del programa.....	13
6.1.3.1	Insertando el contacto	13
6.1.3.2	Insertando la salida.....	14
6.1.4	Transferir el programa al MLCHIP1	14
7	VARIABLES en Microladder	15
7.1	Tipos de datos	15
7.1.1	Bit	15
7.1.2	Byte.....	15
7.1.3	Word y Double Word	15
7.1.4	Integers	16
7.1.5	String	16
7.1.6	Timer	16
7.2	Clases de variables	17
7.2.1	RAM Data	17
7.2.2	EEPROM Data	17
7.2.3	FLASH Data	17
7.2.4	AI.....	17
7.2.5	TIMER.....	17
7.2.6	SPECIAL Data	17
7.3	Creación de variables	19
7.3.1	Dirección	19
7.3.2	Tipo.....	19
7.3.3	Nombre	20
7.3.4	Comentario	20
7.3.5	Campos adicionales “Valor inicial” y “tamaño”	20
7.3.6	Tamaño.....	20
7.3.7	Valor inicial	21
8	CONFIGURACIÓN de los Pines	22
8.1	Tipos permitidos para cada Pin	23
8.2	Configurando los Pines del microcontrolador en ML_PC.....	24
8.3	Combinación de las entradas analógicas:	25

9	LADDER, la programación	26
9.1	<i>Lenguaje.....</i>	26
9.1.1	Contactos	26
9.1.2	Salidas.....	26
9.1.3	Funciones.....	26
9.1.4	Bloques	27
9.2	<i>Construcción de programas en ladder:.....</i>	27
9.2.1	Añadir comentarios al bloque	29
9.3	<i>El lenguaje de nemónicos.....</i>	29
9.4	<i>Métodos abreviados del teclado.....</i>	30
9.4.1	Edición en la ventana de Símbolos:	30
9.4.2	Edición en la ventana de Configuración:	30
9.4.3	Edición en la ventana de Ladder:.....	31
9.5	<i>Insertando funciones en ML-PC.....</i>	32
9.6	<i>Funciones.....</i>	33
9.6.1	Función "SET":	33
9.6.2	Función "RESET":	34
9.6.3	Función "SET_CF":.....	34
9.6.4	Función "RESET_CF":	34
9.6.5	Función "TOGGLE":.....	35
9.6.6	Función "UP":	36
9.6.7	Función "DOWN":.....	37
9.6.8	Función "TON"	38
9.6.9	Función "TOFF":.....	39
9.6.10	Función "TIME_UP":.....	40
9.6.11	Función "TIME_DN":	41
9.6.12	Función "COPY":	42
9.6.13	Función "MOV":	43
9.6.14	Función "MOVI":	44
9.6.15	Función "FILL"	45
9.6.16	Función "AND"	46
9.6.17	Función "OR":.....	47
9.6.18	Función "XOR":.....	48
9.6.19	Función "ROR":.....	49
9.6.20	Función "ROL":.....	50
9.6.21	Función "COMPL":.....	51
9.6.22	Función "++":	52
9.6.23	Función "--":	53
9.6.24	Función "ADD"	54
9.6.25	Función "SUB":.....	55
9.6.26	Función "MUL":.....	56
9.6.27	Función "DIV"	57
9.6.28	Función "CMP":	58
9.6.29	Función "EQUAL":.....	59
9.6.30	Función "GET_EE":.....	60
9.6.31	Función "PUT_EE":.....	61
9.6.32	Función "REFLCD":	62
9.6.33	Función "STRLCD":	63
9.6.34	Función "CHRLCD":.....	65
9.6.35	Función "INTLCD":	66
9.6.36	Función "SCL":	67
9.6.37	Función "CONV":	68
9.6.38	Función "NOP":.....	68
9.6.39	Función "STOP":.....	68
9.6.40	Función "END"	68
9.7	<i>Verificación de Bloques</i>	69
9.8	<i>Compilación de programa</i>	69

10	Monitorización en Microladder.....	70
10.1	<i>Introducción.....</i>	70
10.2	<i>La monitorización.....</i>	70
10.3	<i>Cambio de valores durante la monitorización.....</i>	70
10.4	<i>Monitorización en modo Run y modo Stop.....</i>	70
10.5	<i>Monitorización de las Variables SD asociadas a la comparación.....</i>	71
10.6	<i>Agradecimientos.....</i>	72

1 *¿Qué es un microcontrolador?*

Un microcontrolador es un circuito integrado programable. Los microcontroladores se programan para controlar el funcionamiento de una determinada tarea. Para ello contienen todos los elementos de un computador aunque de manera limitada, estos son:

- *Microprocesador.*
- *Memoria.*
- *Líneas de Entrada / Salida (E/S).*

Todos estos elementos están contenidos dentro del mismo Microcontrolador, solo salen al exterior a través de determinados Pines del chip, las líneas de (E/S) que gobiernan los periféricos. A continuación entraremos un poco más en detalle de cada uno de estos elementos:

El *microprocesador*, o simplemente el micro, es el “cerebro” que se encarga de realizar todas las operaciones lógicas. A veces al microprocesador se le denomina "CPU" (Central Process Unit, Unidad Central de Proceso).

La *memoria* del microcontrolador está formada por; *La memoria de Programa*, que contiene todas las instrucciones del programa de control, el cual está grabado de forma permanente en la misma. Lo que permite el uso de memorias del tipo solo escritura, ROM, EEPROM o FLASH son algunos de los tipos más utilizados y *la memoria de Datos*, que contiene toda la información que varía de forma continua en el programa (como las variables), por lo tanto esta memoria ha de ser de Lectura / Escritura, por lo que es frecuente el uso de la memoria RAM.

Las *líneas de E/S* se utilizan para comunicar el microcontrolador con los diferentes periféricos que el programa va a controlar, la mayoría de las patillas del chip están destinadas a este fin y se agrupan comúnmente en grupos de 8, denominados Puertos.

2 Formas de Programar un microcontrolador.

El “código máquina” es el conjunto de instrucciones con las que trabajan los microcontroladores, sin embargo la programación parte de un lenguaje mucho más sencillo para el programador, posteriormente se transforma a código máquina para poder grabarlo al micro, todo esto se resume en los tres pasos que muestra el siguiente gráfico.



2.1 Editor

El programador utiliza el editor para escribir el código fuente (líneas de instrucciones del programa), dependiendo de la calidad del Editor, ofrecerá un mayor número de prestaciones y ayudas al programador para facilitarle el desarrollo del mismo.

2.2 Compilador

Para poder programar los microcontroladores surge la necesidad de crear un programa que permita generar el código máquina a partir de las instrucciones pertenecientes al lenguaje de programación utilizado por el programador. A partir de este concepto se desarrollaron los “Compiladores”.

En resumen, un Compilador permite traducir un programa hecho con un determinado lenguaje de programación a código máquina.

Muchos de los compiladores que existen en el mercado incorporan un Editor propio.

Se puede clasificar un Compilador en función del lenguaje de programación que va a traducir a código máquina:

- *Ensamblador*: Es el de más bajo nivel de todos, una instrucción de ensamblador se traduce a una instrucción de código máquina, es por tanto el más rápido y el que menos memoria ocupa. Por el contrario, la programación puede ser en un principio más complicada y laboriosa que en otros lenguajes de nivel más alto.
- *Basic*: Es un lenguaje de más alto nivel y que por su sencillez se ha popularizado mucho, es menos rápido que el lenguaje ensamblador y ocupa más memoria.
- *C*: Más próximo al lenguaje Ensamblador que Basic en cuanto a nivel de programación, pero más complejo de programar que éste último.

2.3 Grabador

El grabador es un software que se utiliza para “volcar” el código máquina del programa, al microcontrolador.

El software de grabación se encarga de grabar el programa en el microcontrolador haciendo uso de los puertos de comunicaciones del PC y de un Hardware específico de grabación cuyo diseño depende del microcontrolador que se use.

3 ¿Qué es Microladder?

Microladder es un entorno completo para el desarrollo de aplicaciones de control utilizando un microcontrolador con un firmware* pregrabado y basado en el mismo lenguaje de programación (Ladder) y monitorización que los utilizados por los más avanzados autómatas industriales del mercado. La potencia del entorno se basa en la unión de un firmware en el microcontrolador y un software que corre en el entorno MS-Windows y que permite al programador realizar modificaciones en el programa de usuario a través del puerto serie del ordenador. No requiere ningún conocimiento previo sobre microcontroladores ni sobre lenguajes de programación habituales en microcontroladores como son ensamblador o C.

**firmware: software que corre dentro del mismo microcontrolador.*

Microladder al ser un entorno completo de programación, sube un nivel por encima de los sistemas tradicionales de programación de microcontroladores:

- El editor incorporado, formado por un entorno gráfico basado en ladder, ofrece una edición sencilla, potente, rápida e intuitiva.
- La traducción se realiza directamente a código máquina, sin lenguajes intermedios ofreciendo una rapidez que hasta ahora no se asociaba a entornos similares.
- Desaparece el concepto de software grabador, Microladder incorpora en su firmware un sistema por el cual la grabación se realiza desde el mismo entorno sin necesidad de hardware dedicado adicional.

Y va más allá...

- Ofreciendo la capacidad de monitorización en tiempo real del programa y del estado de las entradas / salidas que controla el mismo, por lo que la depuración se puede hacer incluso desde la misma placa de aplicación.
- La alta integración de Microladder permite realizar todo el proceso de programación desde la edición hasta la monitorización, desde el mismo entorno sin software ni hardware adicional.
- La sencillez del entorno Microladder, permite conseguir grandes resultados sin necesidad de pasar por una larga fase de aprendizaje, lo que hasta ahora no sucedía en las demás alternativas de programación.

4 Funcionamiento de Microladder.

Microladder esta formado por:

- **ML-Chip:** un firmware grabado en el PIC, que contiene un Sistema Operativo y un número determinado de funciones para el control y gestión del microcontrolador.
- **ML-PC:** un Software que corre en windows98 y superiores, desarrollado para que, junto a ML-Chip, se pueda programar y monitorizar el microcontrolador.
ML-PC es un entorno de programación basado en el lenguaje Ladder y permite configurar y programar el chip a través de un interfaz gráfico haciendo uso de las opciones y funciones incluidas en el S.O. de ML-Chip.
Además, ofrece un potente sistema de monitorización para el control y supervisión del programa y de las variables que lo forman así como de las salidas y entradas del microcontrolador.

4.1 Comunicación entre ML-PC y ML-Chip.

La conexión y comunicación entre ambos se realiza desde ML-PC, a través de las siguientes acciones:

Conectar: establece el puerto serie (COM) como puerto de comunicación entre ML-PC y ML –Chip.

Transferir: transfiere el programa de usuario desarrollado en ML-PC a ML-Chip.

Monitorizar: monitoriza desde ML-PC el programa que se ha cargado en ML-Chip.

4.2 Modo Run y modo Stop.

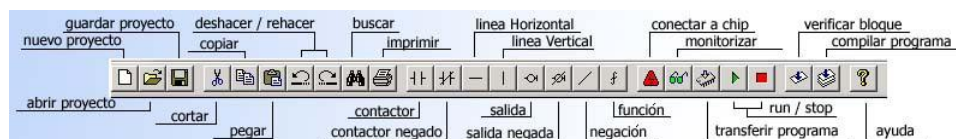
Una vez conectados ML-PC y ML-Chip, el microcontrolador puede estar en dos modos diferentes, modo *Run* y modo *Stop*.

Modo Run: En modo Run se ejecuta el programa de usuario, por lo que el micro ejecutará todas las funciones del programa y activará sus salidas en función de la programación que se le haya cargado desde ML-PC.

Modo Stop: en modo Stop, no se ejecuta el programa de usuario, por tanto, se detiene la ejecución de las funciones y se desactivan las salidas, sin embargo se siguen leyendo las entradas y funcionando las SD de tiempos (SD_0_1seg, SD_0_5seg, SD_1seg).

4.3 La barra de herramientas de Microladder

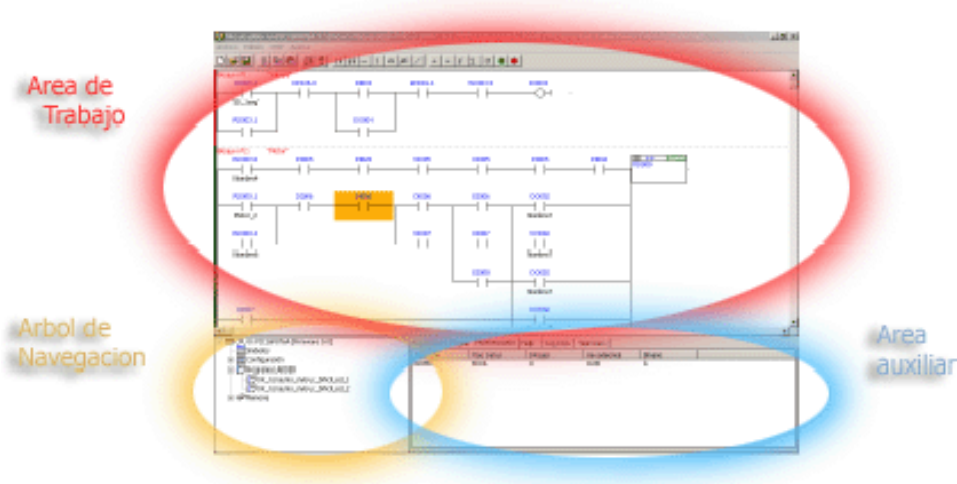
Desde la barra de herramientas, se puede acceder a las principales acciones de comunicación, edición y gestión del proyecto con solo pulsar un botón. La barra de herramientas es accesible desde todas las áreas de trabajo de ML-PC.



Barra de herramientas de Microladder

5 Primeros pasos con Microladder.

El entorno Microladder está formado por 3 áreas: Trabajo, Navegación y Auxiliar



5.1 Área de Trabajo

Desde esta zona de la ventana se llevan a cabo todas las funciones principales de edición, hay tres espacios de trabajo diferentes:

- *Símbolos*: En esta ventana se incluyen todas las variables necesarias para desarrollar nuestro programa. Desde aquí se especifica la dirección de memoria de la variable, el tipo de variable, un nombre identificativo, así como una breve descripción libre de la variable.
- *Configuración*: Desde donde se configuran los Pines del microcontrolador correspondientes a los puertos como entradas o salidas y desde donde, además, podemos monitorizar el estado de las mismas.
- *Programa ladder*: En esta ventana se incluyen todas las herramientas para construir y diseñar el programa a través de la programación ladder. Además desde este área también se puede monitorizar la ejecución del programa.

5.2 Árbol de Navegación

Esta zona de ventana contiene un árbol desde el que se puede acceder a cualquiera de las 3 áreas de trabajo (símbolos, configuración y programa ladder).

5.3 Área Auxiliar

El área auxiliar está formada por cuatro ventanas que se pueden visualizar seleccionando su correspondiente pestaña, estas ventanas son:

- Ventana *Búsqueda*.
- Ventana *Errores*.
- Ventana *FD*.
- Ventana *ED*.



5.3.1 Ventana “Búsqueda”.

En esta ventana aparecen los resultados de la acción “*Buscar*”, acción a la que se puede acceder a través de la barra de herramientas.

En esta ventana específica, el bloque y sección donde se encuentra la variable que queremos localizar y específica si se haya como operando de una función o como un contacto. En caso de hallarse contenida en una función específica el nombre y ubicación de la misma.

5.3.2 Ventana “Errores”.

La ventana “Errores” muestran los fallos registrados al verificar un bloque o compilar el programa, para que el usuario pueda corregirlos y pueda grabar o transferir el programa libre de- errores.

Para más información sobre la verificación de bloque o compilación del programa ir al capítulo dedicado a dichas acciones.

5.3.3 Ventana “FD”.

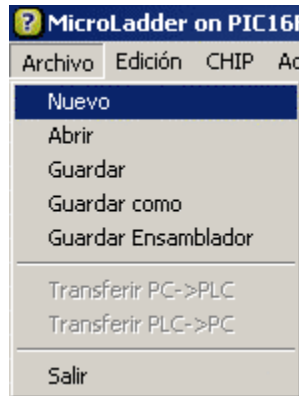
Muestra el valor de las posiciones de memoria Flash del micro accesibles por el usuario.

5.3.4 Ventana “ED”.

Muestra el valor de las posiciones de memoria EEPROM del micro accesibles por el usuario.

6 Un ejemplo con Microladder.

El primer paso consiste en crear un proyecto nuevo, para ello deberemos de ir a:



o pulsar el botón de la barra de herramientas “comenzar un proyecto nuevo”:



A continuación nos pedirá la confirmación para empezar un nuevo proyecto, presionaremos “aceptar”.

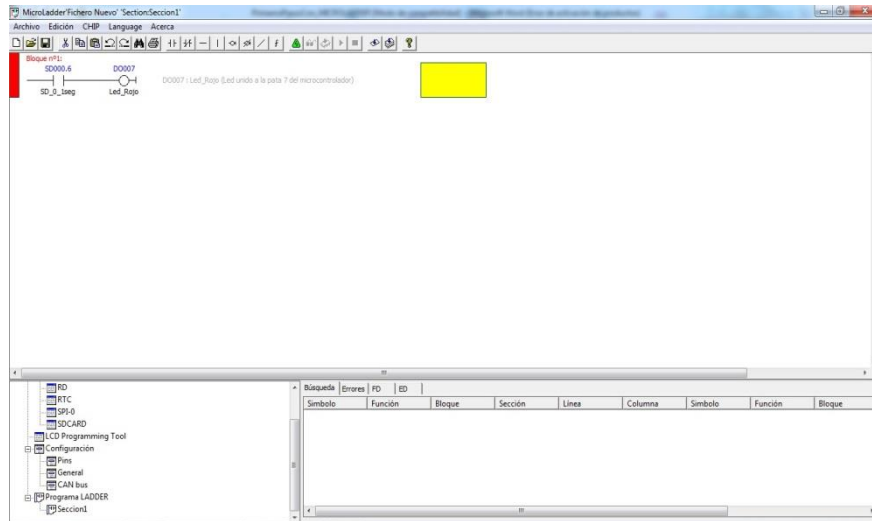
Acabamos de crear un proyecto nuevo, Microladder se inicia en el área de trabajo de “programa ladder” cada proyecto nuevo incluye además unas variables propias del sistema operativo de Microladder utilizables por el usuario (*definidas con detalle más delante en este mismo manual*), con las que ya se puede empezar a programar.

A continuación realizaremos un programa muy sencillo para ir familiarizándonos con las ventanas y entorno Microladder.

6.1 Parpadeo de Una Lámpara cada segundo

El programa que vamos a hacer enciende y apaga en intervalos de un segundo una lámpara que hemos conectado a la Salida 1 NANO LADDER (Pin 7 del Microcontrolador). A través de la programación en ladder nuestro programa terminará con esta apariencia:

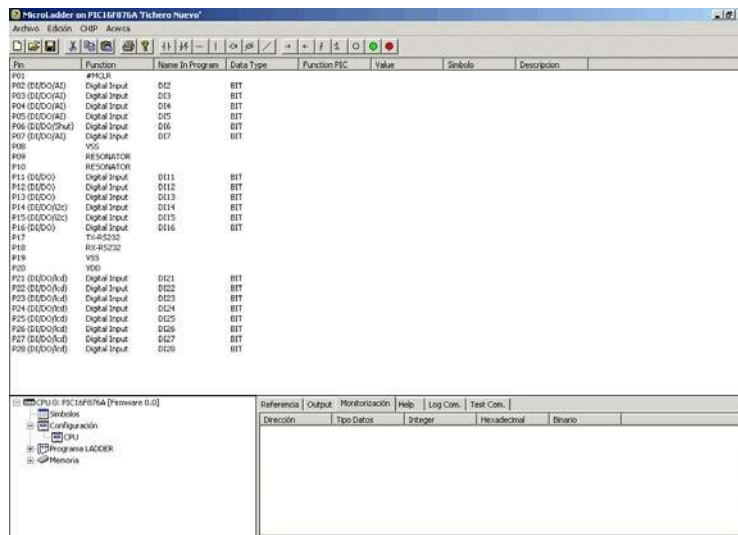
Programa para encender una Lámpara intermitentemente.



Para llegar a realizar este programa seguiremos los siguientes pasos:

6.1.1 Configuración de los Pines

Configuraremos el Pin 7 del micro como salida, para ello: Iremos al *árbol de navegación* y haremos doble clic sobre “configuración” para entrar en esta área.

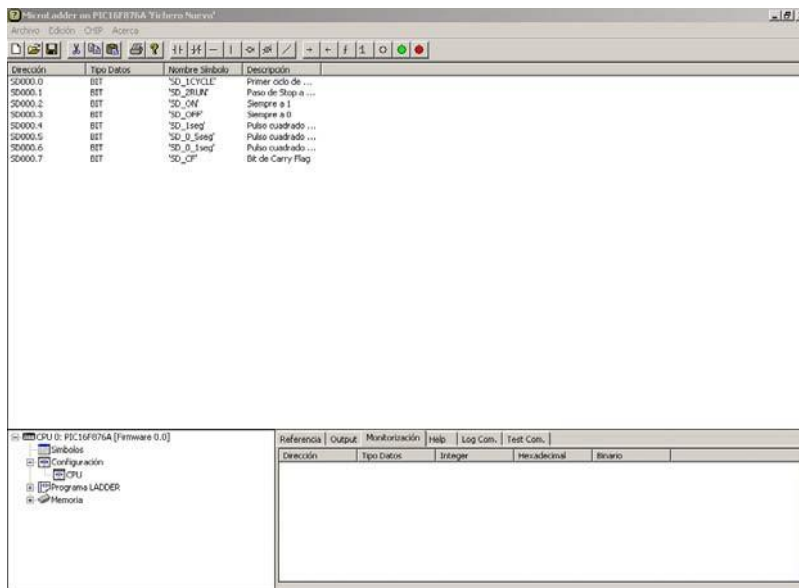


Área de Trabajo Configuración

Una vez en el área de trabajo *configuración*, haremos clic derecho sobre P07 y seleccionaremos desde el menú desplegable “*Como Salida Digital*”. Con esto el Pin 7 del micro queda configurado como salida digital (Salida 1 NANO LADDER) *Ver manual de Usuario NANOLADDER V1 para ver las configuraciones de Salidas.

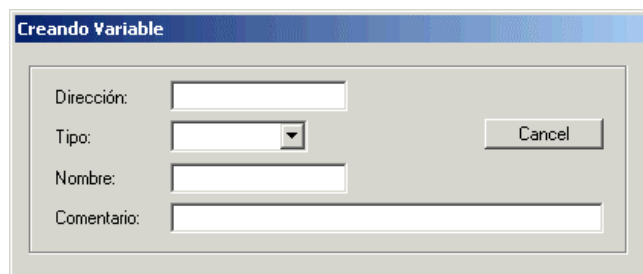
6.1.2 Creación de las variables

Debemos definir la variable de salida, correspondiente al Pin 7 del micro, Iremos al *árbol de navegación* y haremos doble clic sobre “*símbolos*” para entrar en esta área.



Área de Trabajo *Símbolos*.

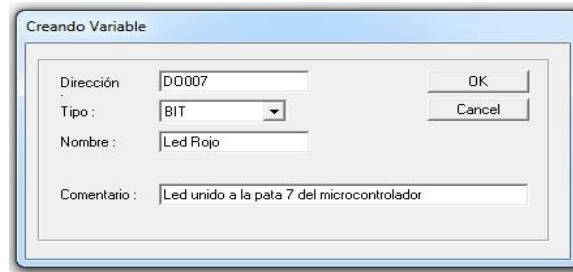
Una vez en el área de trabajo *símbolos*, pulsaremos la tecla “*insertar*” y aparecerá el siguiente cuadro de dialogo.



Rellenaremos los campos de la siguiente manera

Dirección	DO7	<i>Corresponde a Digital Output, Pin 7</i>
Tipo	BIT	<i>Este campo lo completa automáticamente debido a que las Salidas digitales solo pueden ser del tipo “Bit”.</i>
Nombre	Led Rojo	<i>Nos ayuda a identificar la variable</i>
Comentario		<i>En este campo se puede incluir información complementaria relativa a la variable.</i>

De esta manera nos quedará el cuadro de la siguiente manera:



De esta manera ya tenemos creada la variable de salida que controla el Pin 7 y por tanto la Salida 1 de NANO LADDER.

6.1.3 Edición del programa

Una vez que ya tenemos las variables necesarias ya podemos elaborar el programa en Ladder, para ello volveremos al *Árbol de navegación*, y seleccionaremos “programa Ladder”.

Recordemos que nuestro programa encenderá y apagará intermitentemente una lámpara con semiperiodos de 1 segundo.

Para editar en ladder haremos clic sobre el área de trabajo de ladder, el cuadro naranja es el cursor de edición y determina la fila y columna en que nos encontramos, lo llevaremos a la primera fila y columna, desde aquí empezaremos a editar en este ejemplo.

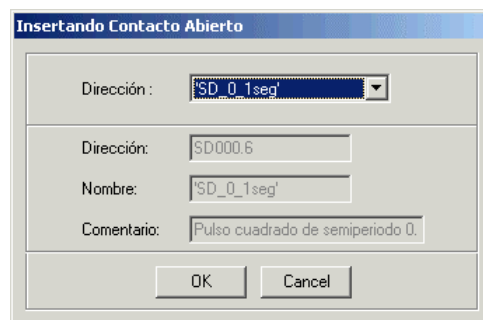
Lo primero es pensar en la estructura que queremos construir, un contacto que se active y desactive cada segundo, cíclicamente y que este conectado a la salida 1. Insertaremos el contacto encargado de la activación cíclica de 1 seg.

6.1.3.1 Insertando el contacto

Pulsaremos “C” (Contacto) o el botón de la barra de herramientas:



Aparece el cuadro de “inserción de contactos”

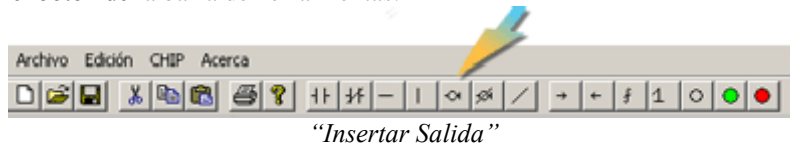


En “Dirección” le indicaremos “SD_0_1seg” que corresponde con una variable especial incluida por Microladder, que conmuta entre “0” y “1” cada segundo, esta variable nos servirá para nuestro ejemplo y será

la encargada de conmutar la salida asociada a la lámpara, para asociarla colocamos la columna siguiente a este contacto, la salida a través de:

6.1.3.2 Insertando la salida

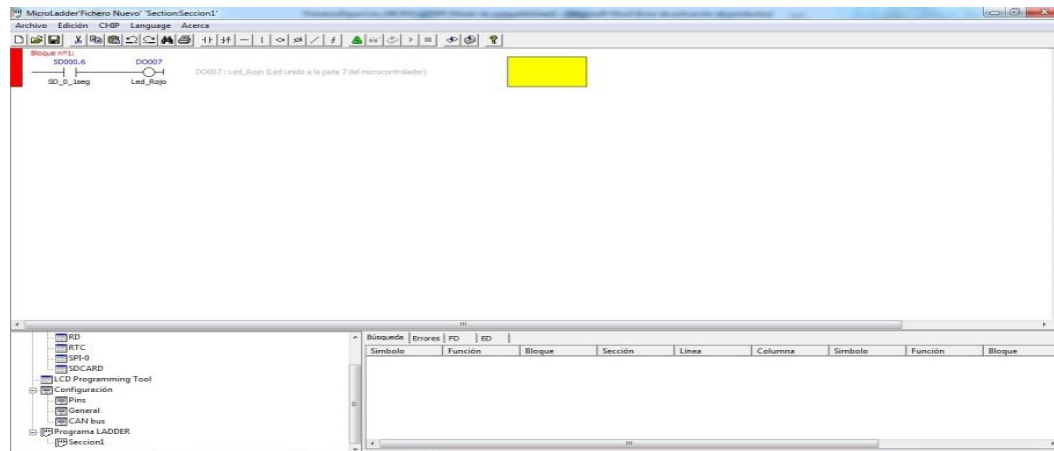
La “O” (output) o el botón de la barra de herramientas:



Aparece un cuadro similar al de la inserción de contactos, seleccionaremos D07 y aceptaremos para insertarla.

Nos encontraremos con un esquema en el área del *programa ladder* como la de la siguiente imagen:

Programa para encender un led intermitentemente.



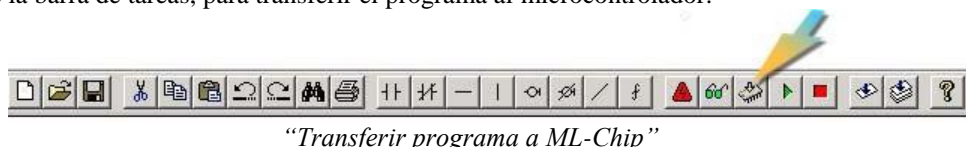
6.1.4 Transferir el programa al MLCHIP1

Nuestro programa para encender el Led o la luminaria ya está listo, ahora deberemos *volcarlo* al microcontrolador, para ello solo seguir dos pasos:

- 1- Hacer clic en: *Menú – CHIP – “Conectar”*, o pulsar el botón correspondiente de la barra de tareas, para establecer comunicación con el microcontrolador.



- 2- Hacer clic en: *Menú – CHIP – “Transferir PC => ML-Chip.”*, o pulsar el botón correspondiente de la barra de tareas, para transferir el programa al microcontrolador.



En unos segundos ya tendremos nuestro programa cargado en el microcontrolador y listo para su uso.

7 VARIABLES en Microladder

7.1 Tipos de datos

Las Variables son valores que se almacenan en la memoria del microcontrolador y cambian dinámicamente su valor según las condiciones especificadas en el programa. El programador se encarga de ir creando las variables a lo largo del proceso de programación según surja la necesidad de su uso. Existen variables de diversos tipos, dependiendo del uso que se le vaya a dar, el programador definirá la variable de un tipo u otro:

DENOMINACIÓN	TIPO DE VARIABLE	BITS	RANGO
Bit	Bit	1	[0 – 1]
Byte	Byte	8	Ejemplo: 0xA2
Word	Word	16	Ejemplo: 0xA23D
DWord	Double Word	32	Ejemplo: 0xACD49FF6
Int8U	Integer 8Bit Unsigned	8	[0 .. 255]
Int8S	Integer 8Bit Signed	8	[-128 .. 127]
Int16U	Integer 16Bit Unsigned	16	[0 .. 65535]
Int16S	Integer 16Bit Signed	16	[-32768 .. 32767]
Int32U	Integer 32Bit Unsigned	32	[0 .. 4294967295]
Int32S	Integer 32Bit Signed	32	[-2147483648 .. 2147483647]
String	Cadena de caracteres	16	[0 - 32] Caracteres
Timer	Temporizador	8	[0 .. 255]

7.1.1 Bit

Es la variable más simple, solo puede adoptar dos valores, [0 – 1], se puede utilizar para almacenar por ejemplo eventos que solo puedan adoptar estos dos valores:

- El estado de un interruptor (encendido o apagado).
- El estado de una alarma (activa o inactiva).

Solo las variables del **tipo Bit** pueden ser usadas como **contactos**.

7.1.2 Byte

Es un conjunto de 8 bits, y se utiliza cuando estos bits tienen algo en común y es necesario acceder a uno o varios de los bits de forma individual y / o conjunta, por ejemplo:

- El estado de un puerto del Microcontrolador (un puerto de 8 E/S).
- Para agrupar un conjunto de bits correspondiente a los estados de las alarmas de un local.

7.1.3 Word y Double Word

Tienen la misma función que el Byte, se diferencia únicamente en el número de bit que asocian. (Word-16bits, Dword-32 bits).

El *Byte*, *Word* y *Dword* se suelen representar normalmente en *Binario*, o en la forma abreviada de binario *Hexadecimal*, la siguiente tabla muestra la correspondencia entre las representaciones Decimal, Binaria y Hexadecimal:

Representación en Hexadecimal

Decimal	Binario	Hexadecimal
0	0000	0x00
1	0001	0x01
2	0010	0x02
3	0011	0x03
4	0100	0x04
5	0101	0x05
6	0110	0x06
7	0111	0x07
8	1000	0x08
9	1001	0x09
10	1010	0x0A
11	1011	0x0B
12	1100	0x0C
13	1101	0x0D
14	1110	0x0E
15	1111	0x0F

Como se puede apreciar el Hexadecimal agrupa en un solo símbolo el equivalente a 4 en binario, precedido por el prefijo “0x”. Aquí tenemos algunos ejemplos de binario y su correspondencia en Hexadecimal:

Binario	0001	0110	0111	1011	1100	1111	0111	0111 1101
Hexadecimal 0x	1	6	7	B	C	F	7	7D

7.1.4 *Integers*

Son variables que se utilizan para almacenar valores numéricos, sirven por ejemplo para:

- Almacenar el valor de un contador.
- Almacenar el valor de una suma.
- Almacenar el valor ofrecido por un sensor de temperatura.
- Etc...

Los integers pueden ser de varios tipos dependiendo del rango de valores que pueden almacenar, y con o sin signo dependiendo de si admiten valores negativos o no. Los integers con signo reparten su rango entre los valores negativos y positivos.

7.1.5 *String*

Son variables que almacenan un determinado número de caracteres, se utilizan para almacenar mensajes alfanuméricos, para ser enviados posteriormente por ejemplo a un LCD. Las cadenas pueden alcanzar un máximo de 32 caracteres.

7.1.6 *Timer*

Este tipo de variables son para usarlas en conjunto con las funciones de temporización y como contactos regidos por las mismas, sobre el uso de los temporizadores se entra con más detalle más adelante en este mismo manual, pueden admitir un valor comprendido entre 0 y 255.

7.2 Clases de variables.

Desde el Área de trabajo de *Símbolos* se definen todas las variables que se utilizan en la programación ladder. Dependiendo de la memoria del micro que utilizemos existen 3 clases diferentes de Variables.

Tipo de Variable		
RAM Data	RD000 a RD095	(96 direcciones)
EEPROM Data	ED000 a ED079	(80 direcciones)
FLASH Data	FD000 a FD199	(200 direcciones)
Timer	T000 a T024	(25 timers)
AI (Analog Input)	AI2, AI3, AI4, AI5, AI7	(5 Pines)

7.2.1 RAM Data

Este tipo de Variables se almacenan en la memoria RAM del micro, por lo tanto, no son permanentes (no mantienen su valor cuando se apaga el micro). Este tipo de variables son las más utilizadas para almacenar los datos utilizados en los procesos que realiza nuestro programa. Microladder puede almacenar hasta 96 posiciones de memoria de este tipo de variables.

7.2.2 EEPROM Data

Las variables de este tipo se almacenan en la memoria EEPROM del micro, por lo que mantienen su valor incluso con el micro apagado, se utilizan para almacenar variables que interesa almacenar de manera permanente, bien por motivos de seguridad o para recuperar su valor cada vez que se inicie el programa. Microladder puede almacenar hasta 50 posiciones de memoria de este tipo de variables.

7.2.3 FLASH Data

La Memoria Flash en Microladder, se utiliza para almacenar Strings (cadena de caracteres) para enviarlas posteriormente a un LCD aprovechando las funciones que Microladder incorpora a tal efecto. Microladder puede almacenar hasta 200 posiciones de memoria de este tipo de variables.

7.2.4 AI

Este tipo de Variable se utiliza para almacenar en un integer de 16 bit sin signo (Int16U), el valor leído en una de las entradas analógicas del micro, por ejemplo:

AI002 almacenara el valor leído por el Pin 2 del micro (siempre que este configurada como analógica). AI004 corresponderá al Pin 4 del micro, etc...

7.2.5 TIMER

Esta variable se utiliza para definir timers o temporizadores que utilizaremos en nuestro programa, Microladder pone a disposición del usuario un total de 25 timers independientes entre sí. Más adelante en este manual se entra en detalle en el uso de los temporizadores.

7.2.6 SPECIAL Data

Las Special Data (SD) son variables que ya están definidas en el S.O. y que ya van incluidas en cada proyecto nuevo de programación. Estas variables están orientadas a ser usadas como contactos en la programación ladder y ofrecen ayuda al programador ya que realizan funciones especiales asociadas a la temporización, comprobación, comparación etc... A continuación, se enumeran las diferentes SD que incorpora Microladder así como su función dentro del entorno de programación.

Variables *Special Data* de Microladder:

<i>Special Data</i>	<i>Condición de Activación</i>	<i>Dirección</i>
<i>SD_1CYCLE</i>	Se activa en el primer ciclo de scan	<i>SD000.0</i>
<i>SD_2RUN</i>	activación cuando se pasa de modo <i>Stop</i> a modo <i>Run</i>	<i>SD000.1</i>
<i>SD_ON</i>	Siempre a “1”	<i>SD000.2</i>
<i>SD_OFF</i>	Siempre a “0”	<i>SD000.3</i>
<i>SD_1seg</i>	Activación en semiperiodos de 1 seg.	<i>SD000.4</i>
<i>SD_0_5seg</i>	Activación en semiperiodos de 500ms.	<i>SD000.5</i>
<i>SD_0_1seg</i>	Activación en semiperiodos de 100 ms	<i>SD000.6</i>
<i>SD_CF</i>	Activación cuando el Carry Flag es “1”	<i>SD000.7</i>
<i>SD_EQ</i>	Activación si el resultado de la función “CMP” es igual	<i>SD001.0</i>
<i>SD_NEQ</i>	Activación si el resultado de la función “CMP” no es igual	<i>SD001.1</i>
<i>SD_GE</i>	Activación si el resultado de la función “CMP” es mayor o igual	<i>SD001.2</i>
<i>SD_GT</i>	Activación si el resultado de la función “CMP” es mayor	<i>SD001.3</i>
<i>SD_LE</i>	Activación si el resultado de la función “CMP” es menor o igual	<i>SD001.4</i>
<i>SD_LT</i>	Activación si el resultado de la función “CMP” es menor	<i>SD001.5</i>
<i>SD_MUL_OVF</i>	Activación si se produce un desbordamiento (overflow) en una multiplicación (función “MUL”) (no se encuentra implementada en ML-CHIP1)	<i>SD001.6</i>
<i>SD_DIV_BY0</i>	Activación si se produce una división por “0” (función “DIV”)	<i>SD001.7</i>
<i>SD_EE_FREE</i>	Activación cuando la EEPROM no esta en proceso de escritura.	<i>SD002.0</i>
<i>SD_WROK</i>	Activación si la escritura en la EEPROM ha sido correcta	<i>SD002.1</i>
<i>SD_WRER</i>	Activación si la escritura en la EEPROM no se ha realizado correctamente	<i>SD002.2</i>
<i>SD_RDOK</i>	Activación si la lectura de la EEPROM se ha realizado correctamente.	<i>SD002.3</i>

El estado que adoptan las SD asociadas a la función “CMP” (comparación), depende del resultado de dicha función. Si a continuación se ejecuta otra comparación, los valores de las SD cambiarán en relación al resultado de esta nueva comparación, sobrescribiendo los de la anterior. Este es un aspecto muy importante a tener en cuenta a la hora de diseñar el programa. Los SD que queramos utilizar como contactos deben de estar después de la comparación a la que queremos asociarlos, y antes de la siguiente comparación, para que su funcionalidad en el programa sea la correcta.

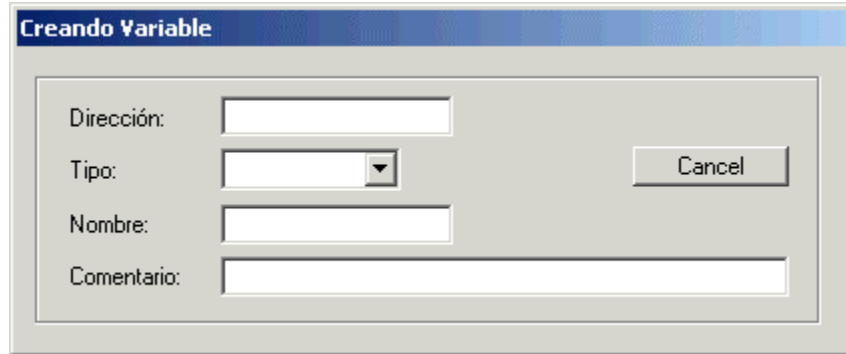
SD de comparación: “SD_EQ”, “SD_NEQ”, “SD_GE”, “SD_GT”, “SD_LE”, “SD_LT”.

7.3 Creación de variables

Para crear Variables en Microladder, deberemos hacerlo desde el área de trabajo “*Símbolos*”:

- *Crear variable*: Tecla “Insert”.
- *Sobre la columna “dirección”, clic derecho, “insertar”*

A continuación se abre el cuadro de diálogo “Crear Variable”



7.3.1 Dirección

En la *Dirección* especificaremos el tipo de variable que queremos crear y la posición de memoria que ocupa, por ejemplo:

- *RD005*: RAM data, posición 5 de memoria (de las 96 posibles)
- *RD025.1*: RAM data, bit 1 de la posición 25 de memoria (de las 96 posibles)
- *FD030*: FLASH data: posición 30 de memoria (de las 200 posibles)
- *ED050*: EEPROM data: posición 50 de memoria (de las 50 posibles)
- *T001*: TIMER: timer número1 (de 25 posibles).
- *AI002*: Analog Input, pata2 del micro (AI002, AI003, AI004, AI005, AI007)

Nota: Microladder auto completa campos, por lo que si escribimos por ejemplo:

- “RD2.2” autocompletará con “RD002.2”.

7.3.2 Tipo

El *Tipo* nos permite seleccionar, que clase de variable va a ser (*bit, byte, int16, int32, string, timer etc...*). Microladder, dependiendo de la *Dirección* de memoria que hayamos especificado, limita este campo a todos los tipos posibles para la dirección especificada, evitando que el programador pueda asignar un tipo incorrecto a la dirección por error.

Por ejemplo evita que le asignemos un tipo *Word* a *RD001.1* ya que esa dirección solo soporta el tipo *BIT*

La siguiente tabla muestra los tipos soportados en función de ejemplos de direcciones de memoria:

	RD001.1	RD005	FD005	ED005	T001	AI7
Bit	X					
Byte		X		X		
Word		X		X		
DWord		X		X		
Int8U		X		X		
Int8S		X		X		
Int16U		X		X		X
Int16S		X		X		
Int32U		X		X		
Int32S		X		X		
String			X			
Timer					X	

7.3.3 Nombre

En el campo *Nombre* podemos insertar una etiqueta que nos ayude a identificar la variable. Es especialmente útil para aquellas variables tipo *BIT* que posteriormente vayamos a utilizar en el programa ladder como contactos o salidas, ya que aparece bajo el contacto o la salida.

7.3.4 Comentario

En este campo podemos añadir un comentario con información completaría de la varia o cualquier información que el programador considere de ayuda.

7.3.5 Campos adicionales “Valor inicial” y “tamaño”

Algunos tipos de direcciones como las FD (FLASH Data) o las ED (EEPROM Data), incorporan campos adicionales:



7.3.6 Tamaño

Especifica el número de caracteres que componen la cadena de caracteres de la FD que definamos, en el ejemplo de la imagen la FD001 la cadena es de 15 caracteres.

7.3.7 Valor inicial

- Este campo aparece en el tipo de direcciones FD (FLASH data) y ED (EEPROM data),
- En el primer caso (FD) sirve para especificar el *contenido* de la cadena de caracteres. En nuestro caso “empezar proceso”, esta cadena será la que almacenará la dirección FD001.
- En el caso de la ED, este campo guardará el *valor* que le asignemos, dependiendo del tipo de variable que contenga (en decimal si es un integer, Hexadecimal si es un byte, word, etc.).

NOTA: Es obligatorio definir este campo para poder crear la variable.

NOTA: cada posición de memoria (ej: RD005) ocupa un byte, si definimos una variable de mayor tamaño (Word , Dword, etc..), ocupara las posiciones siguientes, Microladder evita automáticamente que el usuario defina variables en posiciones que ya están ocupadas por otras.

Para **ELIMINAR** variables, basta con:

- Situarse sobre la variable a eliminar.
- Pulsar la tecla “Supr.”

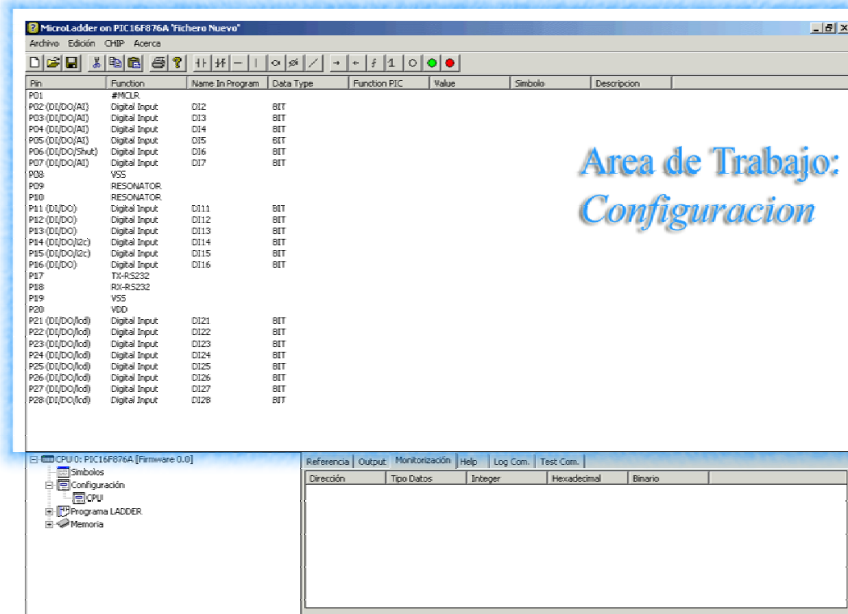
El siguiente cuadro muestra el uso del mapa de memoria del PIC por parte del sistema Microladder:

MEMORIA DEL PIC16F876A ML-CHIP1	
Variables RD	Memoria RAM
Timers (Tx)	
Analog Inputs (AI's)	
Memoria reservada al LCD	
Variables FD	Memoria FLASH
S.O. Microladder	
Programa Usuario	
Variables ED	Memoria EEPROM

Mapa de memoria del PIC

8 CONFIGURACIÓN de los Pines

En Microladder, la configuración nos permite definir cada uno de los Pines del microcontrolador como entradas o salidas (E/S). Toda la configuración se realiza a través del área de trabajo “Configuración” a la que podemos acceder a través del árbol de navegación.



Área de Trabajo Configuración

En esta ventana se reflejan las 28 Pines del microcontrolador para el ML-CHIP 1, y se especifica las opciones de configuración posibles para cada una de ellas. Algunas de los Pines no permiten ser configurados debido a que soportan una función diferente a la de entrada / salida. Es el caso de los Pines de alimentación del micro, transmisión serie y reset.

A continuación se explica el contenido de cada uno de los campos del área de Configuración:

Pin	Incluye el número de Pin (P02, P04...etc.) Posibles opciones de configuración del Pin. (DO, DI...etc.)
Function	Muestra la configuración actual del Pin
Name in program	Muestra el nombre con el que aparecerá en el programa de ladder
Data type	Muestra el tipo de variable con el que se va a tratar el Pin

Tipos de Funciones de las patas:

DI (Digital Input):	Configura el Pin como entrada digital. [0 – 1]
DO (Digital Output)	Configura el Pin como salida digital. [0 – 1]
AI (Analog Input)	Configura el Pin como entrada analógica. [Int16u]
LCD	Configura los Pines P12,P13, P16 y de la P21 a P28 como salida en paralelo para conexión con un display de cristal líquido (LCD).

8.1 Tipos permitidos para cada Pin en NANO LADDER V1

Pin	Función Especial.	DI	DO	AI	LCD
1	MCLR				
2		X			
3		X			
4		X			
5		X			
6		X			
7			X		
8	V _{ss}				
9	Resonator				
10	Resonator				
11			X		
12					X
13					X
14			X		
15			X		
16					X
17	TX-RS232				
18	RX-RS232				
19	V _{ss}				
20	VDD				
21					X
22					X
23					X
24					X
25					X
26					X
27					X
28					X

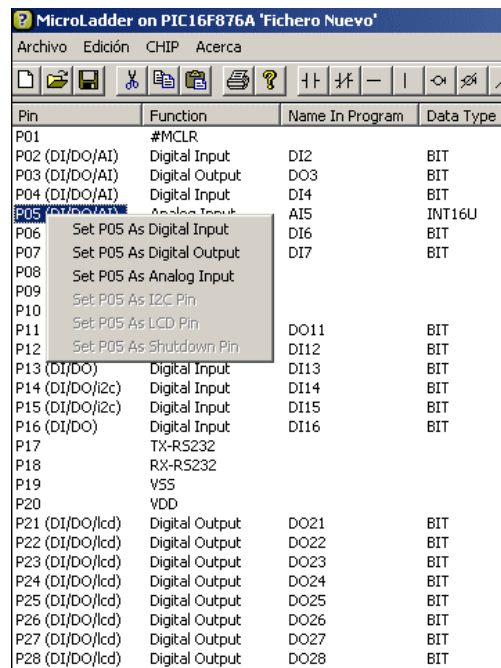
- Pines especiales (no configurables):

#MCLR	(P01)	Reset del micro (Master Clear Reset).
V_{ss}	(P08, P19)	Masa.
VDD	(P20)	Alimentación.
Resonator	(P09, P10)	Entrada cristal del micro
TX-RS232	(P17)	Pin de Transmisión serie UART
RX-RS232	(P18)	Pin de Recepción serie UART.

8.2 Configurando los Pines del microcontrolador en ML_PC

Para configurar un determinado Pin del microcontrolador iremos al área de trabajo *Configuración* y:

- Colocar el puntero del ratón sobre el Pin correspondiente.
- Hacer Clic derecho (para abrir menú desplegable).
- Elegir la opción de configuración deseada.



Pin	Function	Name In Program	Data Type
P01	#MCLR		
P02 (DI/DO/A1)	Digital Input	DI2	BIT
P03 (DI/DO/A1)	Digital Output	DO3	BIT
P04 (DI/DO/A1)	Digital Input	DI4	BIT
P05 (DI/DO/A1)	Analog Input	AI5	INT16U
P06	Set P05 As Digital Input	DI6	BIT
P07	Set P05 As Digital Output	DI7	BIT
P08	Set P05 As Analog Input		
P09	Set P05 As I2C Pin		
P10	Set P05 As LCD Pin		
P11	Set P05 As Shutdown Pin	DO11	BIT
P12		DI12	BIT
P13 (DI/DO)	Digital Input	DI13	BIT
P14 (DI/DO/I2c)	Digital Input	DI14	BIT
P15 (DI/DO/I2c)	Digital Input	DI15	BIT
P16 (DI/DO)	Digital Input	DI16	BIT
P17	TX-RS232		
P18	RX-RS232		
P19	VSS		
P20	VDD		
P21 (DI/DO/lcd)	Digital Output	DO21	BIT
P22 (DI/DO/lcd)	Digital Output	DO22	BIT
P23 (DI/DO/lcd)	Digital Output	DO23	BIT
P24 (DI/DO/lcd)	Digital Output	DO24	BIT
P25 (DI/DO/lcd)	Digital Output	DO25	BIT
P26 (DI/DO/lcd)	Digital Output	DO26	BIT
P27 (DI/DO/lcd)	Digital Output	DO27	BIT
P28 (DI/DO/lcd)	Digital Output	DO28	BIT

Menú desplegable de configuración de Pines.

En la sección *Function* se muestra la configuración actual del Pin.

Nota: Debido a que la opción de configuración “LCD” requiere el uso simultaneo de los Pines 12,13,16 y de la 21 a la 28 (ML-CHIP1), cuando configuremos una de estos Pines como “LCD” las demás cambiarán automáticamente también a la configuración “LCD”.

8.3 Combinación de las entradas analógicas:

Debido a las restricciones del PIC, las combinaciones posibles de configuración de las entradas analógicas / digitales se limitan a las 4 mostradas por la siguiente tabla:

Configuración	P7	P5	P4	P3	P2
5AI	AI7	AI5	AI4	AI3	AI2
2 AI	DI7/DO7	AI5	DI4/DO4	AI3	AI2
1 AI	DI7/DO7	DI5/DO5	DI4/DO4	DI3/DO3	AI2
0 AI	DI7/DO7	DI5/DO5	DI4/DO4	DI3/DO3	DI2/DO2

***Para la versión NANO LADDER V1 no existe la posibilidad de entradas analógicas**

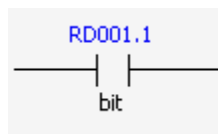
9 LADDER, la programación

9.1 Lenguaje

- La programación *Ladder* está basada en diagramas de contactos, un sistema de programación fuertemente basado en la edición a través de un entorno visual. Los diagramas de contactos son esquemas que representan los elementos de entrada, salida y las funciones que forman el programa.
- La programación Ladder consigue a través de los diagramas de contactos conformar programas que satisfagan cualquier necesidad de programación orientada a automatizar procesos.
- La programación Ladder trabaja con tres tipos de elementos:

9.1.1 Contactos

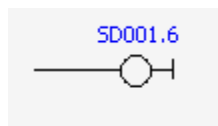
Son los elementos que forman el bloque lógico, su resultado lógico (0 –1) activará o desactivará la salida o la ejecución de la función, pueden ser variables RD (RAM data) a nivel de bit, entradas digitales del micro, timers e incluso las propias salidas digitales se pueden usar como entradas.



Contacto

9.1.2 Salidas

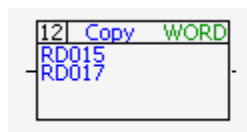
Son los elementos a los que se le va aplicar el resultado de los diferentes procesos del programa, formados por los contactos. Una salida puede estar condicionada por uno o varios de los diferentes procesos que conforman el programa. Las salidas pueden ser variables a nivel de bit tipo RD (RAM data) o las salidas digitales del micro.



Salida

9.1.3 Funciones

Son algoritmos que realizan una operación determinada (sumar, restar, escalar, dividir, incrementar etc...) y se les aplica a las diferentes variables, entradas etc. que tengamos definidas. El uso y elección de las diferentes funciones depende de las necesidades de cada programa.

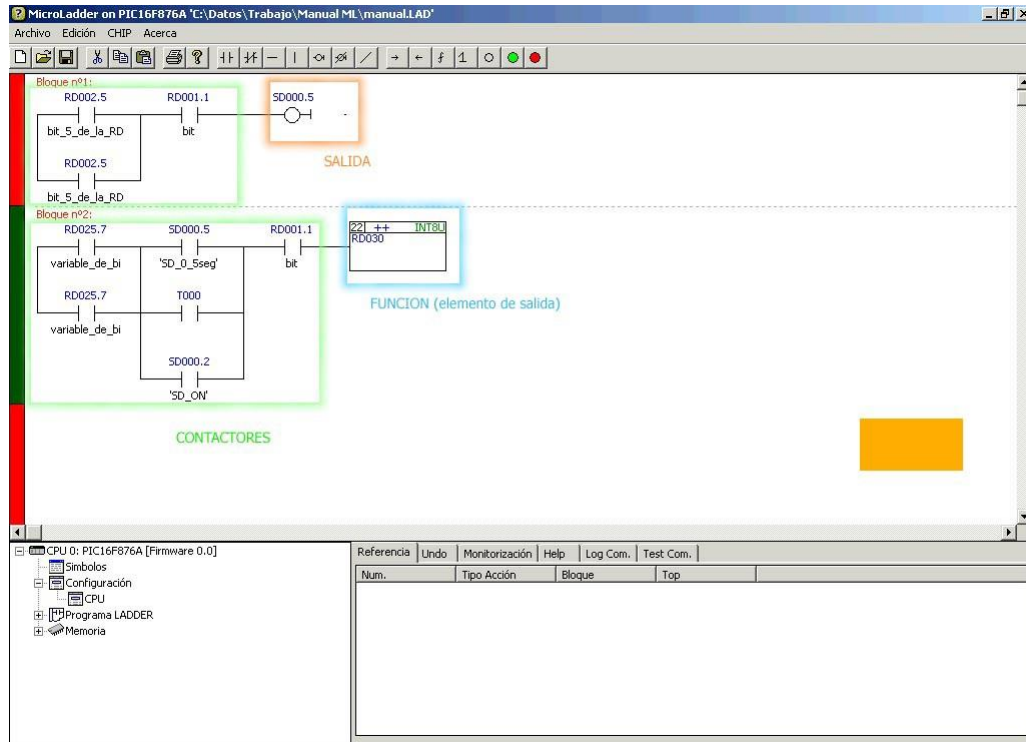


Función "Copy"

9.1.4 Bloques

Los bloques son agrupaciones de los tres elementos anteriores, cada bloque realiza una función lógica determinada que activa una función o una salida, de esta manera un programa en ladder, contiene desde el mínimo de un bloque hasta decenas de ellos.

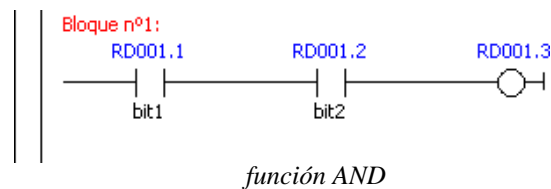
La siguiente imagen corresponde a un ejemplo de programa en Ladder:



Ejemplo de programa en Ladder

9.2 Construcción de programas en ladder:

- Como ya se ha comentado la programación en *Ladder* se basa en los diagramas de contactos. Los diagramas de contactos son agrupaciones de contactos que representan las operaciones lógicas AND u OR según la disposición entre ellos, la combinación de más contactos da como resultado combinaciones múltiples o mixtas de estas dos operaciones.
- La asociación serie de contactos equivale a la **función AND**.

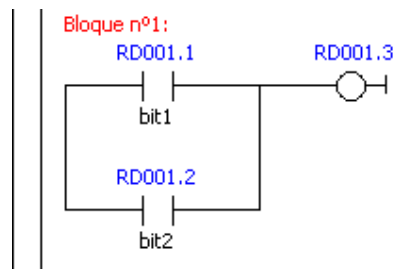


- En la **función AND** todos los contactos han de estar a 1 para que se active la salida o se ejecute la función.

contactos		salida
Bit1	Bit2	Bit1 AND Bit2
0	0	0
0	1	0
1	0	0
1	1	1

Tabla verdad de la función **AND**

- La asociación paralelo de contactos equivale a la **función OR**.



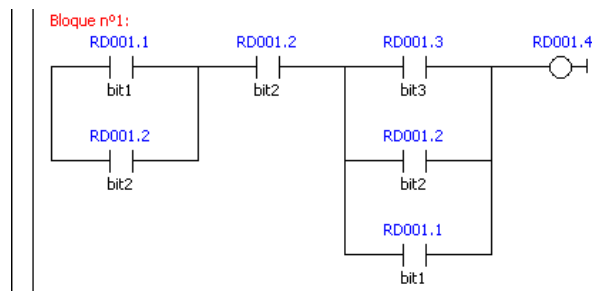
Función **OR**

- En la función OR basta con que un contacto esté a 1 para que se active la salida o se ejecute la función.

contactos		salida
Bit1	Bit2	Bit1 OR Bit2
0	0	0
0	1	1
1	0	1
1	1	1

Tabla verdad de la función **OR**

- La **combinación mixta** de estas dos operaciones lógicas es muy común en los bloques que forman los programas en *Ladder*, la siguiente imagen muestra un ejemplo de combinación mixta de AND y OR.



Combinación mixta de **AND** y **OR**

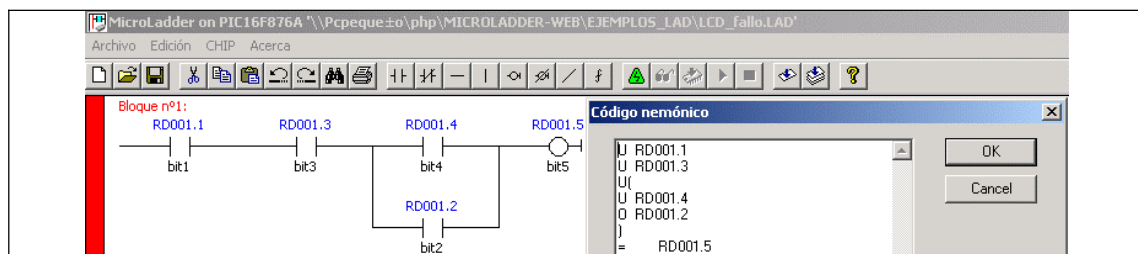
- La combinación mixta de las operaciones lógicas AND y OR permite realizar bloques con comportamientos lógicos más complejos.

9.2.1 Añadir comentarios al bloque.

Si se desea añadir un comentario al bloque, basta con hacer doble clic sobre el nombre del bloque (bloque nº1, nº2, etc.).

9.3 El lenguaje de nemónicos

- **Microladder** transfiere el programa de usuario al microchip en dos pasos:
 - Traduciendo el programa *Ladder* a *ensamblador*.
 - Transfiriendo a través de la función “*transferir PC->ML-CHIP1*” el programa al microchip.
- Antes de traducir de *Ladder* a *Ensamblador*, Microladder traduce a un lenguaje intermedio de nemónicos.
- El lenguaje de nemónicos utilizado por Microladder está inspirado en AWL de Siemens que utiliza en autómatas como el S7-300.
- Como ya hemos comentado, los diagramas de bloques se pueden traducir como un conjunto de operaciones AND y OR, el resultado de esa traducción es el lenguaje de nemónicos de Microladder.
- Microladder permite al usuario ver la traducción a este lenguaje intermedio con la intención de que tenga la oportunidad de pueda aprender el lenguaje utilizado por los autómatas de la familia Siemens.
- El *lenguaje de nemónicos* de Microladder, traduce los bloques lógicos de contactos a una secuencia de líneas de texto que contemplan la relación de OR y AND representadas en el diagrama de contactos.

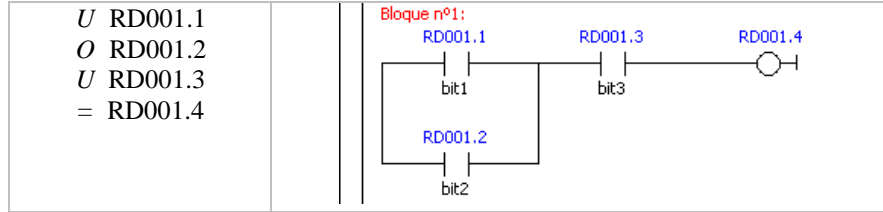


Traducción de Ladder a Lenguaje de Nemónicos.

- La primera línea del lenguaje de nemónicos siempre empieza con una función AND y por el primer contacto del diagrama (el primero por arriba a la izquierda).
- Las funciones AND se representan con letra “U” y las OR con la letra “O”. De esta manera:

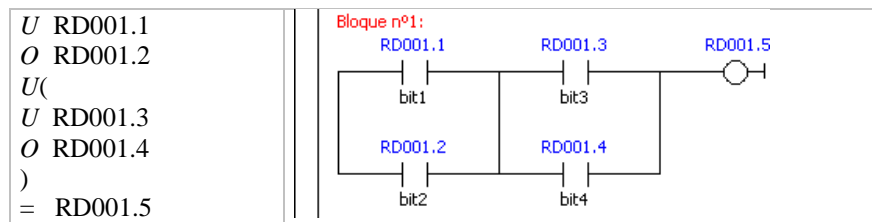
```
U RD001.1
U RD001.2
= RD001.4
```

- Significa que RD001.4 depende del resultado de una AND entre RD001.1 y RD001.2



- RD001.4 depende del resultado de : RD001.1 OR RD001.2 y a continuación una AND con RD001.1

- Los paréntesis agrupan al igual que en las operaciones aritméticas comunes, subconjuntos de operaciones jerárquicamente inferiores.



- En este ejemplo se puede apreciar como las operaciones entre paréntesis se tratan como si fuesen un solo elemento para el resto de las operaciones lógicas del bloque.
- De esta manera las operaciones de este ejemplo serían:
 - RD001.1 OR RD001.2
 - A continuación se hace la AND con el resultado de la operación RD001.3 OR RD001.4
 - El resultado activará o desactivará la salida RD001.5

9.4 Métodos abreviados del teclado

9.4.1 Edición en la ventana de Símbolos:

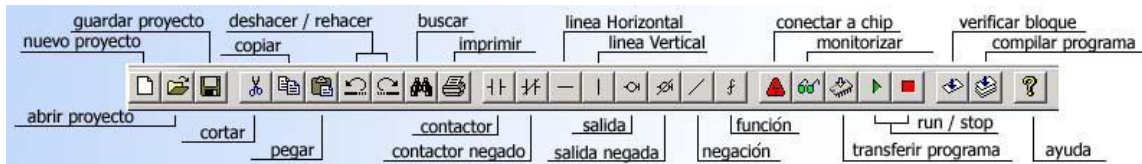
Acción	Método abreviado
Insertar variable	Insert (o botón derecho: <i>insertar</i>)
Borrar variable	Supr. (o botón derecho: <i>eliminar</i>)
Modificar variable	Doble clic, Enter (o botón derecho: <i>editar</i>)

9.4.2 Edición en la ventana de Configuración:

Acción	Método abreviado
Configurar Pin	Botón derecho

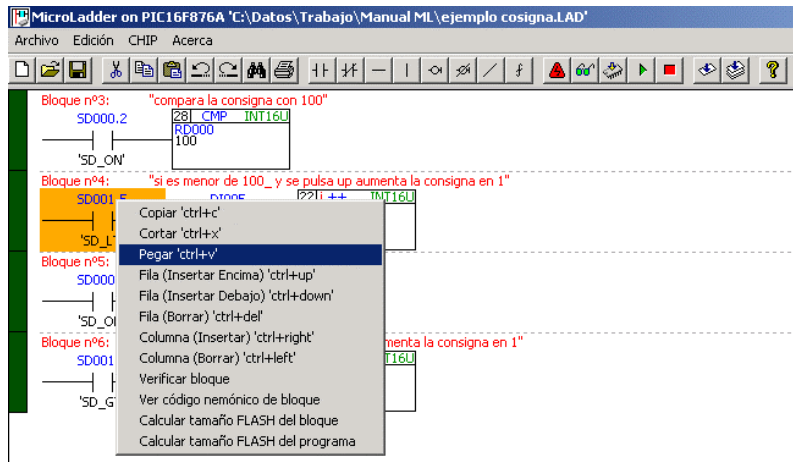
9.4.3 Edición en la ventana de Ladder:

La edición en Ladder es muy sencilla, ya que se realiza básicamente con los cursores y el ratón. La mayoría de las acciones se pueden llamar directamente desde los menús de la ventana o con la barra de herramientas:



Barra de herramientas de Microladder

A través del menú contextual del ratón (botón derecho). Obtenemos también acceso a algunas opciones de edición, al lenguaje de neumónicos generado por el programa y a opciones para calcular el espacio en la memoria flash que ocupa un bloque o el programa entero.



Las opciones de edición también son accesibles a través de los métodos abreviados del teclado.

Elemento	Método abreviado
Contacto	C
Salida	O
Función	F
Barra horizontal	H
Barra Vertical	V
Negar contacto o salida	/

Edición	Método abreviado
Fila, (insertar encima)	Ctrl. + Up
Fila, (insertar debajo)	Ctrl. + Down
Fila, Borrar	Ctrl. + Supr.
Columna, Insertar	Ctrl. + Right
Columna, borrar	Ctrl. + Left
Bloque, (insertar arriba)	Ctrl. + A
Bloque, (insertar debajo)	Ctrl. + S
Bloque, Borrar	Ctrl. + D

9.5 Insertando funciones en ML-PC

Microladder-PC (ML-PC) consta de 42 tipos de funciones diferentes. Las funciones se acceden desde el área de trabajo *Ladder*, ya sea:

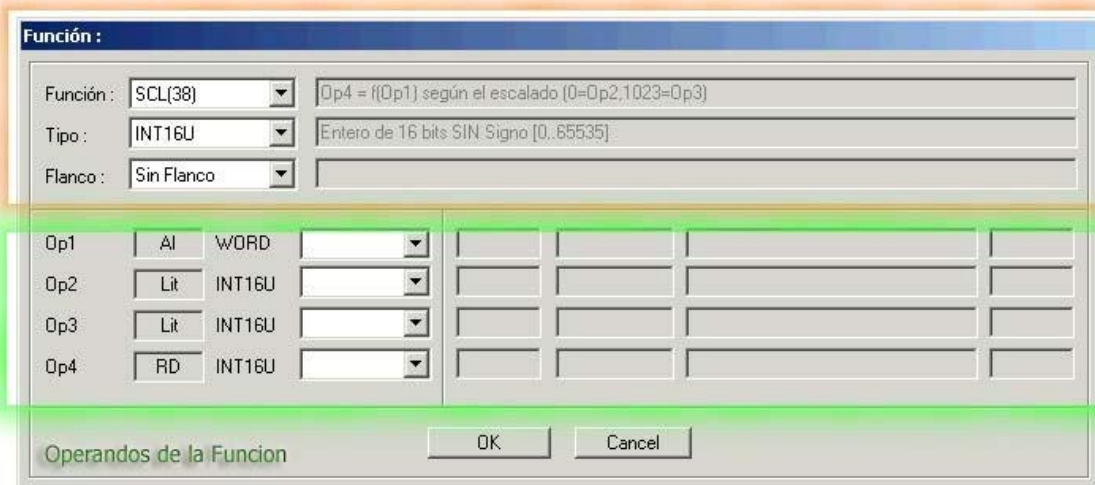
- Pulsando la tecla “F”.
- o con el botón de la barra de tareas:



Botón “Insertar Función”

Aparece así el cuadro de inserción de Funciones:

Parámetros generales de la función.



Ventana de inserción de Funciones

- *Parámetros Generales de la función:*

- **Función:** Este campo Especifica la función que vamos a insertar en el programa.
- **Tipo:** Especifica el tipo de variable que va a tratar la Función.
- **Flanco:** Indica si la Función se activa a través de Flanco (ascendente o descendente) o sin Flanco (con niveles).

- *Operandos de la Función:*

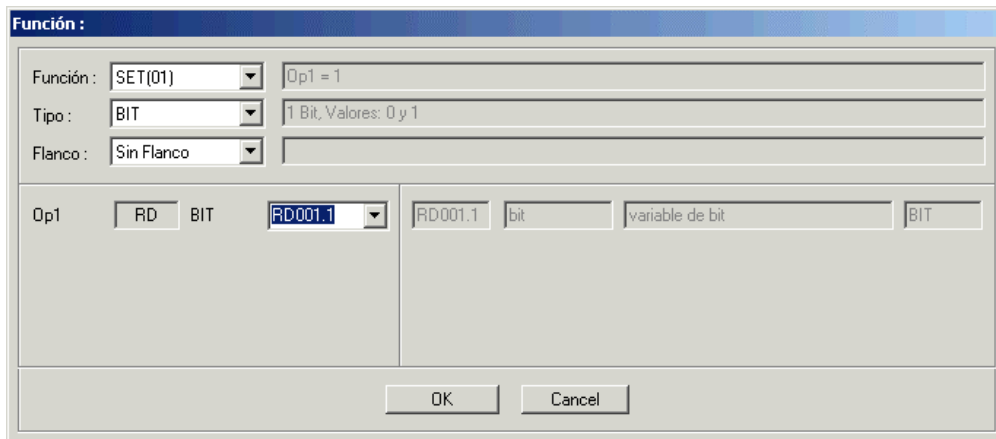
- Los operandos las variables o datos que determinan la entrada, salida y parámetros de la función. El número de operandos varía dependiendo del tipo de función.

A continuación se detalla las características y función de cada una de ellas:

9.6 Funciones

9.6.1 Función "SET":

- Esta función se encarga de poner a "1" las variables de tipo "BIT".
 - Operandos:
 - Op1: corresponde a la variable que queremos poner a "1".

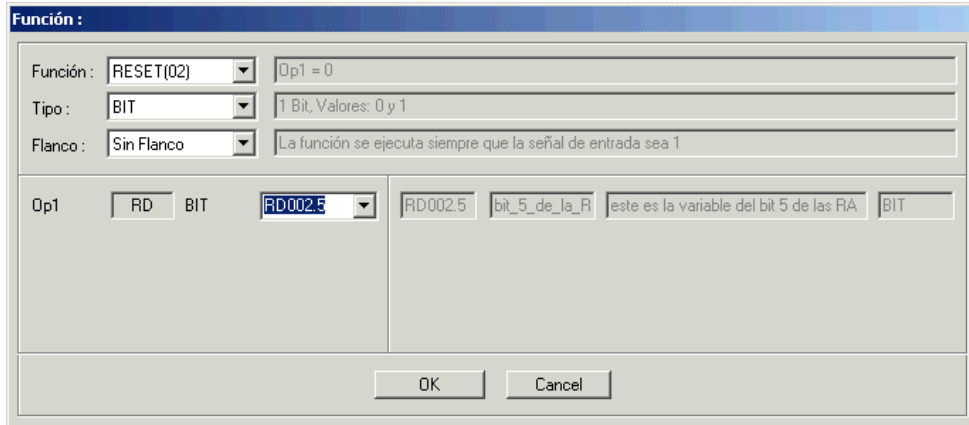


Ejemplo función BIT

- En este ejemplo la función *BIT* pone a "1" el bit 1 de la RD001 (variable RD001.1)

9.6.2 Función “RESET”:

- Esta función se encarga de poner a “0” las variables de tipo “BIT”.
 - Operandos:
 - Op1: corresponde a la variable que queremos poner a “0”.



Función :

Función : RESET(02) Op1 = 0

Tipo : BIT 1 Bit, Valores: 0 y 1

Flanco : Sin Flanco La función se ejecuta siempre que la señal de entrada sea 1

Op1 RD BIT RD002.5 RD002.5 bit_5_de_la_RA este es la variable del bit 5 de las RA BIT

OK Cancel

Ejemplo función RESET

- En este ejemplo la función *BIT* pone a “0” el bit 5 de la RD002 (variable RD002.5)

9.6.3 Función “SET_CF”:

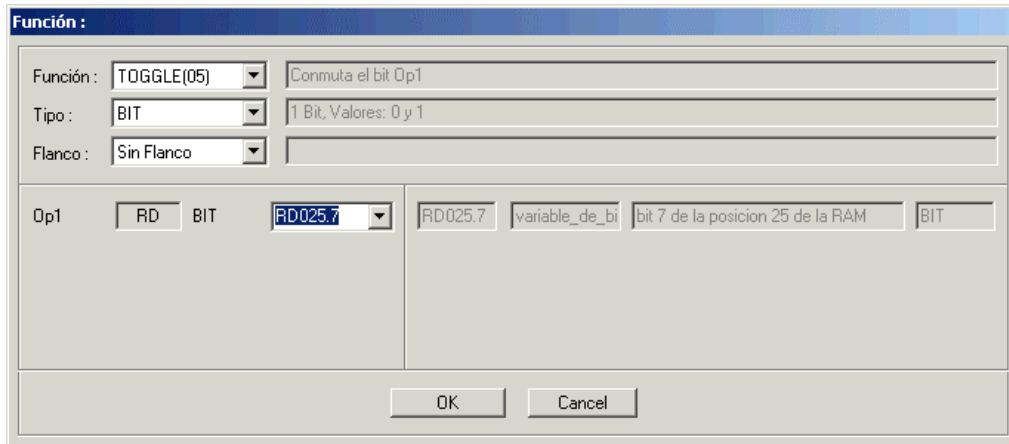
- Esta función se encarga de poner a “1” el Carry flag (registro de Carry).

9.6.4 Función “RESET_CF”:

- Esta función se encarga de poner a “0” el Carry flag (registro de Carry).

9.6.5 Función “TOGGLE”:

- Esta función se encarga conmutar el valor de las variables tipo BIT.
 - Operandos:
 - Op1: corresponde a la variable que queremos conmutar.

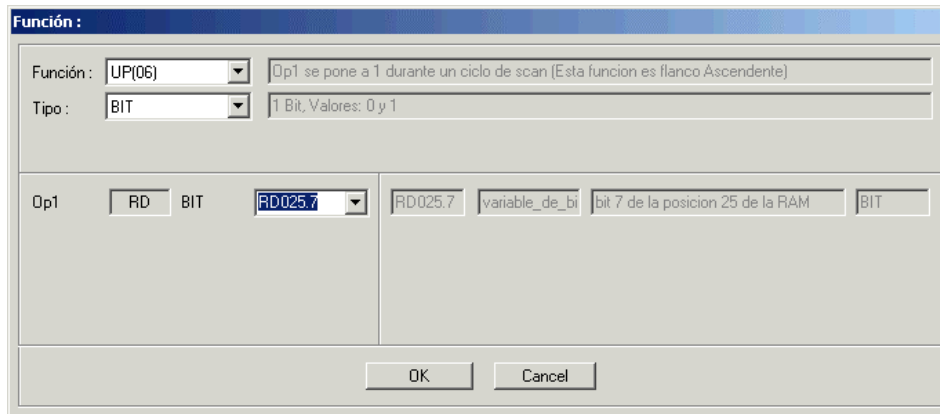


Ejemplo función “Toggle”

- En este ejemplo la función “toggle” conmutara el valor de la variable RD025.7, (bit 7 de la posición 25 de la RAM), a “1” si esta estaba en “0” y viceversa.

9.6.6 Función “UP”:

- Esta función se encarga de poner a 1 las variables tipo BIT durante 1 ciclo de scan, cuando detecta un flanco *ascendente*., Pasado el ciclo de scan la variable pasa a ser “0”.
 - Operandos:
 - Op1: corresponde a la variable que queremos poner a 1.



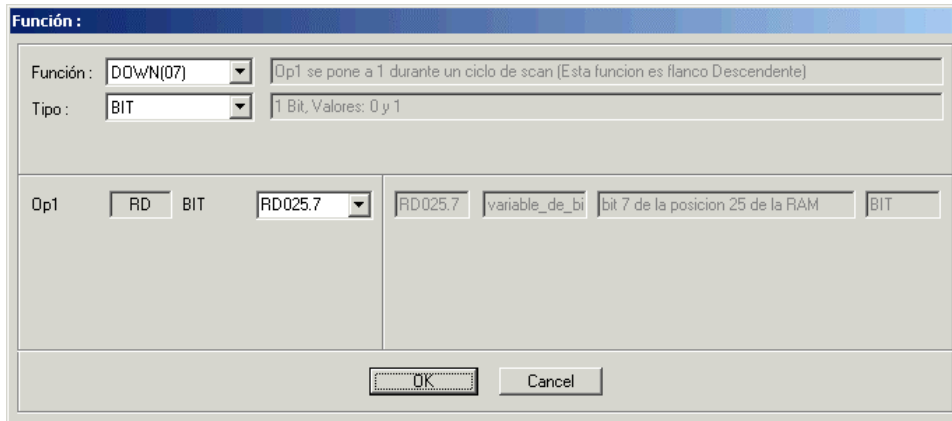
Ejemplo función “Up”

- En este ejemplo la función “up” pone a 1 la variable RD025.7, (bit 7 de la posición 25 de la RAM), durante un ciclo de scan, cuando ha detectado un flanco ascendente. Pasado ese ciclo de scan la variable pasa a ser “0”

9.6.7 Función “DOWN”:

- Esta función se encarga de poner a 1 las variables tipo BIT durante 1 ciclo de scan, cuando detecta un flanco *descendente*. Pasado el ciclo de scan la variable pasa a ser “0”.

- Operandos:
- Op1: corresponde a la variable que queremos poner a 0.

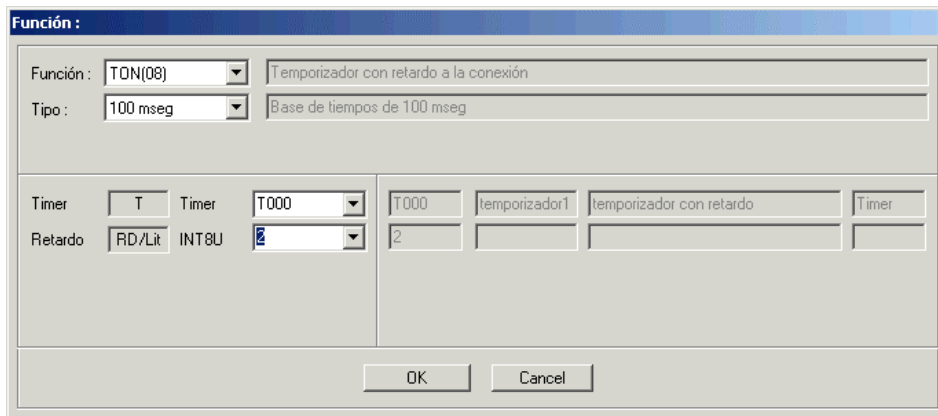


Ejemplo función “Down”

- En este ejemplo la función “down” pone a 0 la variable RD025.7, (bit 7 de la posición 25 de la RAM), durante un ciclo de scan, cuando a detectado un flanco descendente. Pasado ese ciclo de scan la variable pasa a ser “0”

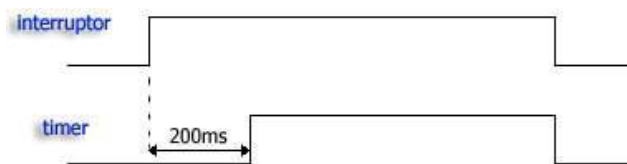
9.6.8 Función “TON”:

- Esta es función activa un temporizador con retardo, pasado ese retardo el temporizador se pone a “1” hasta que la señal de entrada que lo activa baje a “0” y se puede utilizar como contactor.
 - Tipo: especifica la base de tiempos con que trabaja el temporizador. (De 1ms, 10ms, 100ms y 1000ms)
 - Operandos:
 - Timer: especifica el timer (temporizador) que vamos a utilizar (debe estar definido en *Símbolos*)
 - Retardo: es el factor de multiplicación de la base de tiempos, puede ser un literal (2, 30, 50 etc...) o el contenido de una variable INT8U que hayamos definido (RD025, etc...)



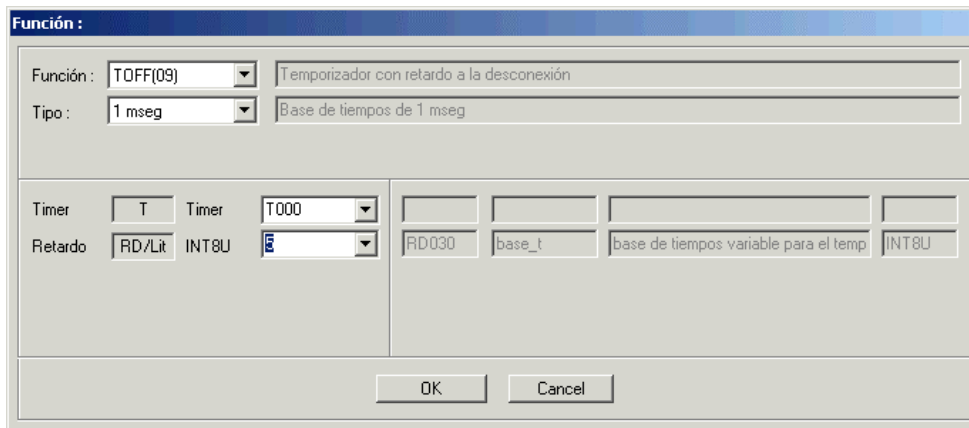
Ejemplo función “TON”

- En este ejemplo la función “TON” activa el temporizador T000, que se pone a “1” después de 2 x 100ms (*Retardo x Tipo*).
- Supongamos un contacto correspondiente a un interruptor conectado a un temporizador de la función “Ton” cuando se active el interruptor, el contacto correspondiente se pondrá a “1”, el timer esperará 200ms (2 x 100 ms) y a continuación se pondrá a “1”, hasta que el interruptor vuelva a “0”
- El siguiente dibujo muestra diagrama de tiempos entre el interruptor y el timer (T000):



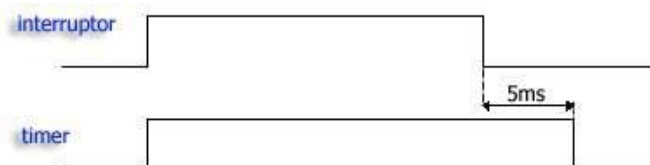
9.6.9 Function “TOFF”:

- Esta es función activa temporizador, que se activa con la señal de entrada a “1” y cuando esta baja a “0” sigue un determinado tiempo a “1”, pasado ese retardo el temporizador se pone a “0”.El temporizador se puede utilizar en otra línea del programa como contacto.
 - Tipo: especifica la base de tiempos con que trabaja el temporizador. (De 1ms, 10ms, 100ms y 1000ms)
 - Operandos:
 - Timer: especifica el timer (temporizador) que vamos a utilizar (debe estar definido en *Símbolos*)
 - Retardo: es el factor de multiplicación de la base de tiempos, puede ser un literal (2, 30, 50 etc...) o una variable INT8U que hayamos definido (RD025, etc...)



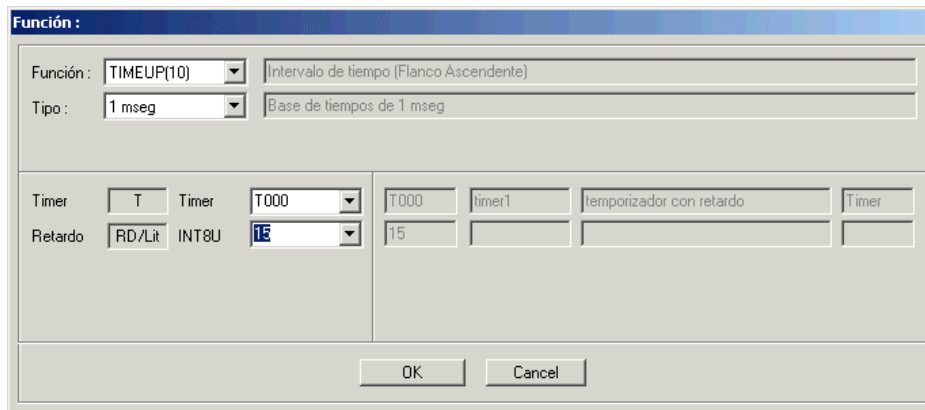
Ejemplo función “TOFF”

- En este ejemplo la función “TOFF” activa el temporizador T000, que se pone a “0” después de 5 x 1ms (*Retardo x Tipo*).
- El siguiente dibujo muestra diagrama de tiempos entre el interruptor y el timer T000:



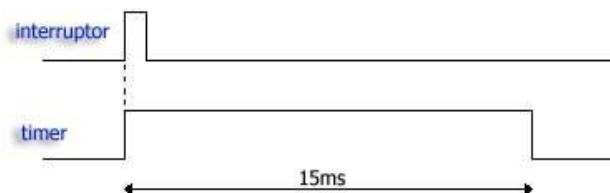
9.6.10 Función "TIME_UP":

- Esta función cuando detecta un flanco *ascendente* activa un temporizador que se pone a "1" durante un tiempo determinado, pasado este tiempo baja a "0". El temporizador se puede utilizar en otra línea del programa como contacto.
 - Tipo: especifica la base de tiempos con que trabaja el temporizador. (De 1ms, 10ms, 100ms y 1000ms)
 - Operandos:
 - Timer: especifica el timer (temporizador) que vamos a utilizar (debe estar definido en *Símbolos*)
 - Retardo: es el factor de multiplicación de la base de tiempos, puede ser un literal (2, 30, 50 etc...) o una variable INT8U que hayamos definido (RD025, etc...)

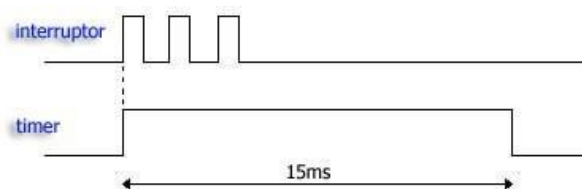


Ejemplo función "Time Up"

- En este ejemplo la función "Time Up" activa el temporizador T000, que se pone a "1" durante 15 x 1ms (*Retardo x Tipo*).
- El siguiente dibujo muestra diagrama de tiempos entre el interruptor y el timer T000:

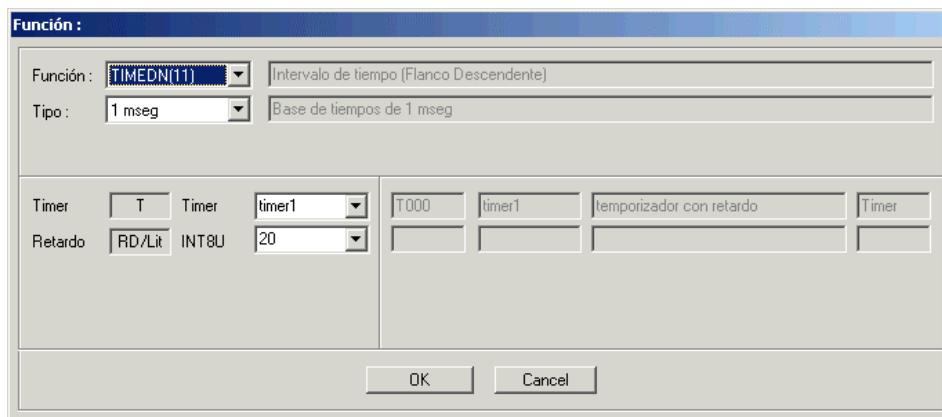


- El timer ignora los pulsos que se produzcan a su entrada una vez que se ha activado...



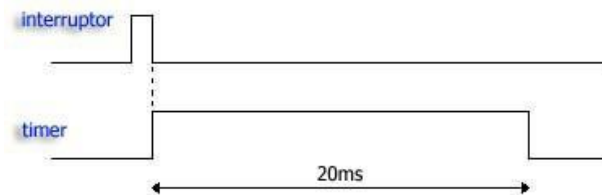
9.6.11 Función “TIME_DN”:

- La función “Time Down” cuando detecta un flanco *descendente* activa un temporizador que se pone a “1” durante un tiempo determinado, pasado este tiempo el temporizador baja a “0”. El temporizador se puede utilizar en otra línea del programa como contacto.
 - Tipo: especifica la base de tiempos con que trabaja el temporizador. (De 1ms, 10ms, 100ms y 1000ms)
 - Operandos:
 - Timer: especifica el timer (temporizador) que vamos a utilizar (debe estar definido en *Símbolos*)
 - Retardo: es el factor de multiplicación de la base de tiempos, puede ser un literal (2, 30, 50 etc...) o una variable INT8U que hayamos definido (RD025, etc...)

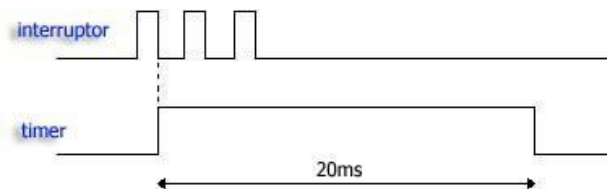


Ejemplo función “Time Down”

- En este ejemplo la función “Time Down” activa el temporizador T000, que se pone a “1” durante 20 x 1ms (*Retardo x Tipo*).
- El siguiente dibujo muestra diagrama de tiempos entre el interruptor y el timer T000:

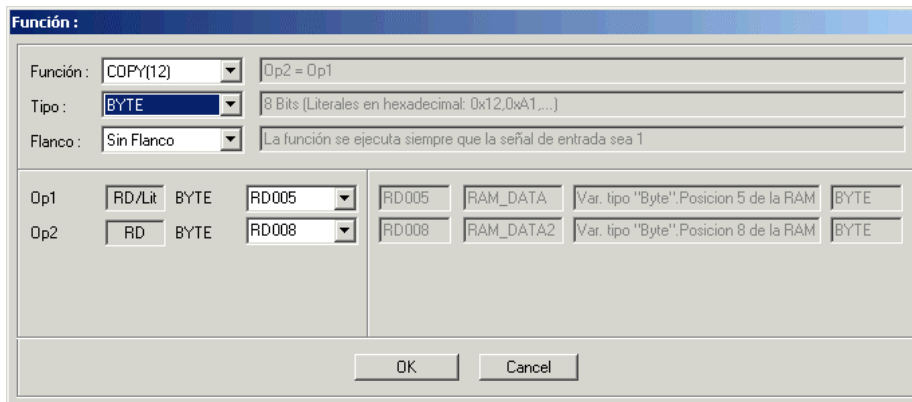


- El timer ignora los pulsos que se produzcan a su entrada una vez que se ha activado.



9.6.12 Función “COPY”:

- Esta función se encarga de copiar el contenido de una variable a otra del mismo tipo.
 - Tipo: permite seleccionar el tipo de variable a copiar (Byte, Word, Dword, e integers de 8, 16 y 32 con y sin signo).
 - Operandos:
 - Op1: corresponde a la variable que queremos copiar.
 - Op2: corresponde a la variable de destino de la copia.



Función :

Función : Op2 = Op1

Tipo : 8 Bits (Literales en hexadecimal: 0x12,0xA1,...)

Flanco : La función se ejecuta siempre que la señal de entrada sea 1

Op1	<input type="text" value="RD/Lit"/>	<input type="text" value="BYTE"/>	<input type="text" value="RD005"/>	<input type="text" value="RD005"/>	<input type="text" value="RAM_DATA"/>	<input type="text" value="Var. tipo 'Byte'. Posicion 5 de la RAM"/>	<input type="text" value="BYTE"/>
Op2	<input type="text" value="RD"/>	<input type="text" value="BYTE"/>	<input type="text" value="RD008"/>	<input type="text" value="RD008"/>	<input type="text" value="RAM_DATA2"/>	<input type="text" value="Var. tipo 'Byte'. Posicion 8 de la RAM"/>	<input type="text" value="BYTE"/>

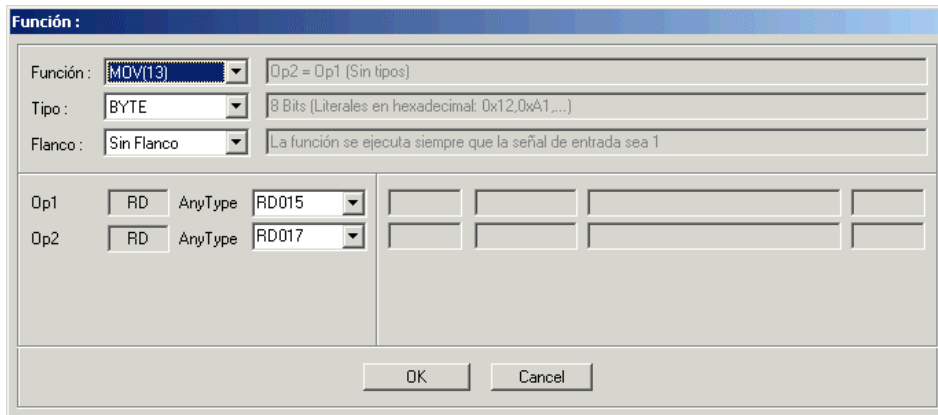
OK Cancel

Ejemplo función “Copy”

- En este ejemplo la función “copy” copia la variable R005 a la variable RD008, ambas del tipo Byte.

9.6.13 Función “MOV”:

- La función “mov” se encarga de copia el primer Byte de una variable al primer byte de otra. “mov” puede copiar entre variables de cualquier tipo.
 - Operandos:
 - Op1: corresponde a la variable cuyo primer byte queremos copiar.
 - Op2: corresponde a la variable de destino de la copia.



Función :
Función : MOV(13) Op2 = Op1 (Sin tipos)
Tipo : BYTE 8 Bits (Literales en hexadecimal: 0x12,0xA1,...)
Flanco : Sin Flanco La función se ejecuta siempre que la señal de entrada sea 1
Op1 RD AnyType RD015
Op2 RD AnyType RD017
OK Cancel

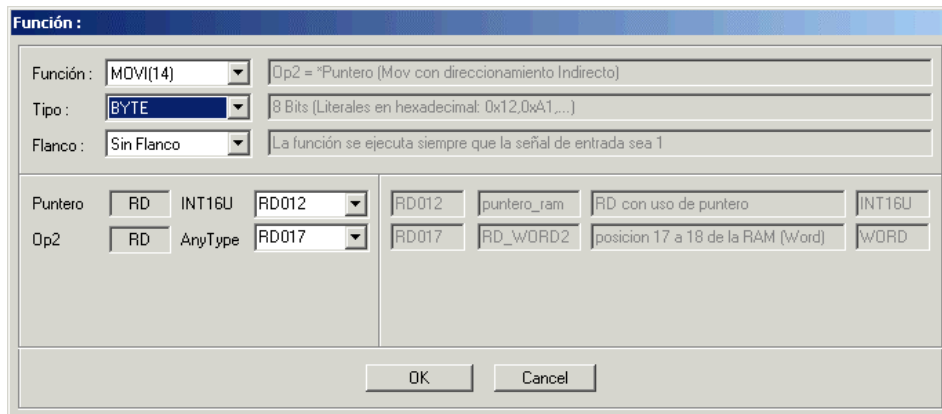
Ejemplo función “mov”

- En este ejemplo la función “move” copia la variable RD015 a la variable RD017, ambas del tipo Word.

Nota: en los operandos que contienen tipos “Any Type”, las variables han de ser introducidas manualmente, ya que no aparecen automáticamente en dichos campos.

9.6.14 Función “MOVI”:

- La función “movi.” (movimiento indirecto) se encarga de copiar el primer Byte de la dirección contenida en un puntero al primer byte de otra.
 - Operandos:
 - Op1: corresponde a dirección que actúa como puntero y que contiene la dirección de la que se va a copiar el primer byte.
 - Op2: corresponde a la variable de destino de la copia.



Función :
Función : MOVI(14) Op2 = *Puntero (Mov con direccionamiento Indirecto)
Tipo : BYTE 8 Bits (Literales en hexadecimal: 0x12,0xA1,...)
Flanco : Sin Flanco La función se ejecuta siempre que la señal de entrada sea 1

Puntero RD INT16U RD012 RD012 puntero_ram RD con uso de puntero INT16U
Op2 RD AnyType RD017 RD017 RD_WORD2 posicion 17 a 18 de la RAM (Word) WORD

OK Cancel

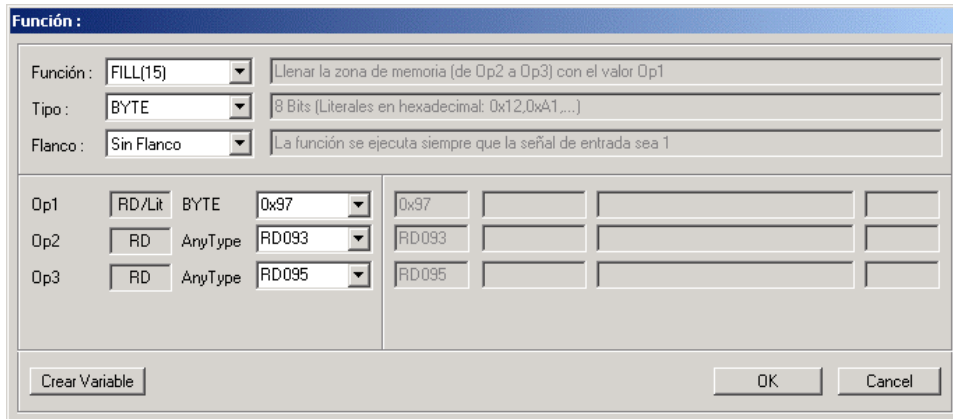
Ejemplo función “movi”

- En este ejemplo la función “movi” copia la variable RD012 a la variable RD017.

Nota: la dirección que va actuar de puntero ha de ser del tipo INT16U. La variable *destino* de la copia puede ser de cualquier tipo...

9.6.15 Función “FILL”:

- La función “fill.” Copia un valor numérico (*literal*) o variable desde una posición inicial de la memoria hasta otra final.
 - Operandos:
 - Op1: contiene, el valor que queremos copiar al rango de direcciones. (debe ser introducido en Hexadecimal).
 - Op2: corresponde a la dirección inicial de la RAM a copiar.
 - Op3: corresponde a la dirección final de la RAM a copiar.



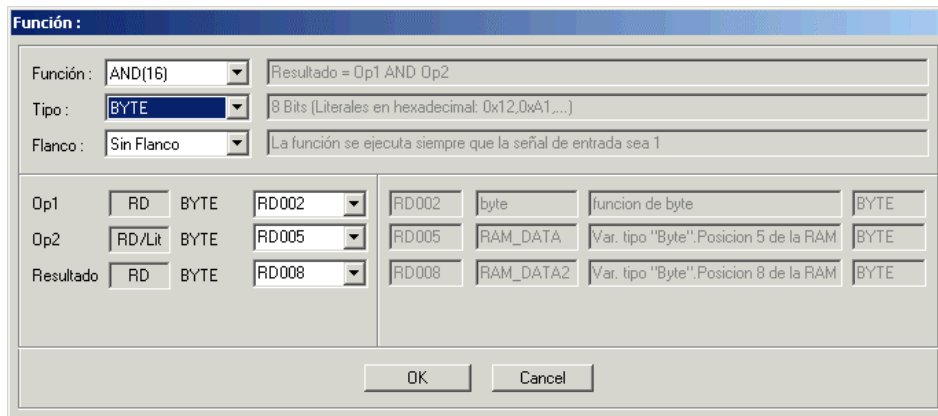
Función :	
Función :	FILL(15) Llenar la zona de memoria (de Op2 a Op3) con el valor Op1
Tipo :	BYTE 8 Bits (Literales en hexadecimal: 0x12,0xA1,...)
Flanco :	Sin Flanco La función se ejecuta siempre que la señal de entrada sea 1
Op1	RD/Lit BYTE 0x97 0x97
Op2	RD AnyType RD093 RD093
Op3	RD AnyType RD095 RD095

Ejemplo función “fill”

- En este ejemplo la función “fill” copia el valor “151” (0x97 en Hexadecimal) en las direcciones comprendidas entre la RD093 y RD095, ambas inclusive.

9.6.16 Función “AND”:

- La función “and” realiza un “and lógico” entre dos variables del mismo tipo y coloca el resultado en una de ellas o en otra variable diferente.
 - Operandos:
 - Op1: contiene, la dirección del primer operando de la operación “and”.
 - Op2: contiene, el valor (literal en hexadecimal) o dirección del segundo operando de la operación “and”.
 - Resultado: corresponde a la dirección donde se va a colocar el resultado de la operación “and”.



Función :

Función : AND(16) Resultado = Op1 AND Op2

Tipo : BYTE 8 Bits (Literales en hexadecimal: 0x12,0xA1,...)

Flanco : Sin Flanco La función se ejecuta siempre que la señal de entrada sea 1

Op1	RD	BYTE	RD002	RD002	byte	funcion de byte	BYTE
Op2	RD/Lit	BYTE	RD005	RD005	RAM_DATA	Var. tipo "Byte", Posicion 5 de la RAM	BYTE
Resultado	RD	BYTE	RD008	RD008	RAM_DATA2	Var. tipo "Byte", Posicion 8 de la RAM	BYTE

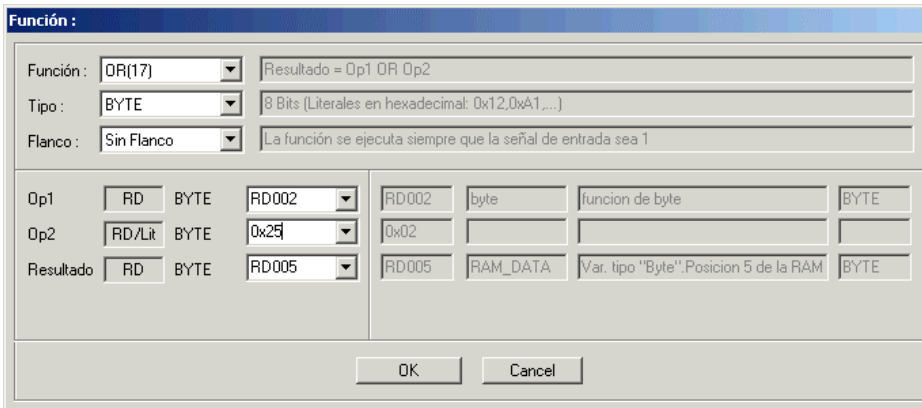
OK Cancel

Ejemplo función “and”

- En este ejemplo la función “and” realiza un AND lógico entre los bytes RD002 y RD005 y coloca el resultado en el byte RD008.

9.6.17 Función “OR”:

- La función “or” realiza un “or lógico” entre dos variables del mismo tipo y coloca el resultado en una de ellas o en otra variable diferente.
 - Operandos:
 - Op1: contiene, la dirección del primer operando de la operación “or”.
 - Op2: contiene, el valor (literal en hexadecimal) o dirección del segundo operando de la operación “or”.
 - Resultado: corresponde a la dirección donde se va a colocar el resultado de la operación “or”.



Función : OR(17) Resultado = Op1 OR Op2

Tipo : BYTE 8 Bits (Literales en hexadecimal: 0x12,0xA1,...)

Flanco : Sin Flanco La función se ejecuta siempre que la señal de entrada sea 1

Op1	RD	BYTE	RD002	RD002	byte	funcion de byte	BYTE
Op2	RD/Lit	BYTE	0x25	0x02			
Resultado	RD	BYTE	RD005	RD005	RAM_DATA	Var. tipo "Byte". Posicion 5 de la RAM	BYTE

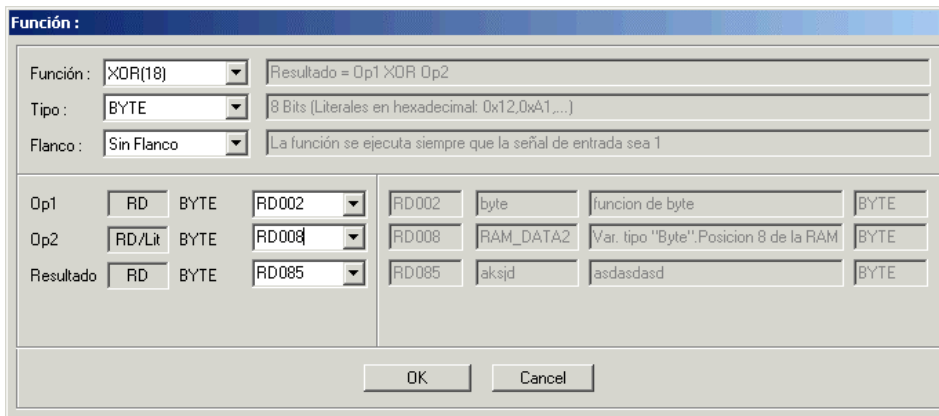
OK Cancel

Ejemplo función “or”

- En este ejemplo la función “or” realiza un OR lógico entre los bytes RD002 y el literal 0x25 y coloca el resultado en el byte RD005.

9.6.18 Función "XOR":

- La función "xor" realiza un "xor lógico" entre dos variables del mismo tipo y coloca el resultado en una de ellas o en otra variable diferente.
 - Operandos:
 - Op1: contiene, la dirección del primer operando de la operación "xor".
 - Op2: contiene, el valor (literal en hexadecimal) o dirección del segundo operando de la operación "xor".
 - Op3: corresponde a la dirección donde se va a colocar el resultado de la operación "xor".



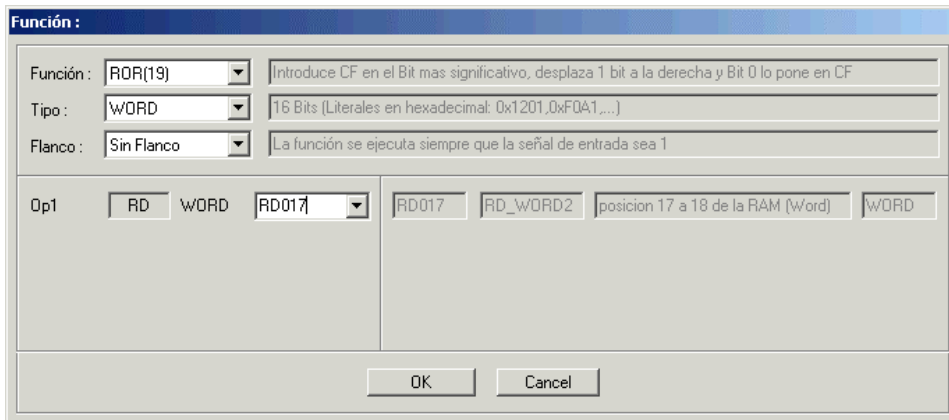
Función :							
Función :	XOR(18)	Resultado = Op1 XOR Op2					
Tipo :	BYTE	8 Bits (Literales en hexadecimal: 0x12,0xA1,...)					
Flanco :	Sin Flanco	La función se ejecuta siempre que la señal de entrada sea 1					
Op1	RD	BYTE	RD002	RD002	byte	funcion de byte	BYTE
Op2	RD/Lit	BYTE	RD008	RD008	RAM_DATA2	Var. tipo "Byte". Posicion 8 de la RAM	BYTE
Resultado	RD	BYTE	RD085	RD085	aksjd	asdasdasd	BYTE
OK Cancel							

Ejemplo función "xor"

- En este ejemplo la función "xor" realiza un XOR lógico entre los bytes RD002 y el literal RD008 y coloca el resultado en el byte RD085.

9.6.19 Función “ROR”:

- La función “ror” (rotate right) desplaza los bits de la RD especificada una posición hacia la derecha, introduciendo el valor del CF (registro Carry Flag) en el bit de más significativo y almacena el bit menos significativo en el CF.
 - Operandos:
 - Op1: contiene la variable RD que queremos rotar.



Función: ROR(19) Introduce CF en el Bit mas significativo, desplaza 1 bit a la derecha y Bit 0 lo pone en CF

Tipo: WORD 16 Bits (Literales en hexadecimal: 0x1201,0xF0A1,...)

Flanco: Sin Flanco La función se ejecuta siempre que la señal de entrada sea 1

Op1 RD WORD RD017 RD017 RD_WORD2 posicion 17 a 18 de la RAM (Word) WORD

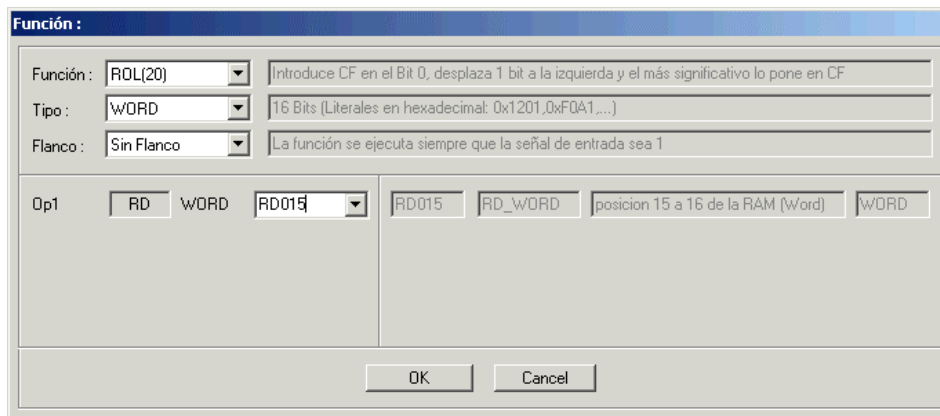
OK Cancel

Ejemplo función “ror”

- En este ejemplo la función “ror” desplaza hacia la derecha los bits de RD017, introduce el valor del CF en el bit más significativo de RD017 y el bit menos significativo (bit 0) pasa a ser almacenado en el CF.

9.6.20 Función “ROL”:

- La función “rol” (rotate left) desplaza los bits de la RD especificada una posición hacia la izquierda, introduciendo el valor del CF (registro Carry Flag) en el bit de menos significativo (bit 0) y almacena el bit más significativo en el CF.
 - Operandos:
 - Op1: contiene la variable RD que queremos rotar.



Función :

Función: ROL(20) | Introduce CF en el Bit 0, desplaza 1 bit a la izquierda y el más significativo lo pone en CF

Tipo: WORD | 16 Bits (Literales en hexadecimal: 0x1201,0xF0A1,...)

Flanco: Sin Flanco | La función se ejecuta siempre que la señal de entrada sea 1

Op1: RD | WORD | RD015 | RD015 | RD_WORD | posicion 15 a 16 de la RAM (Word) | WORD

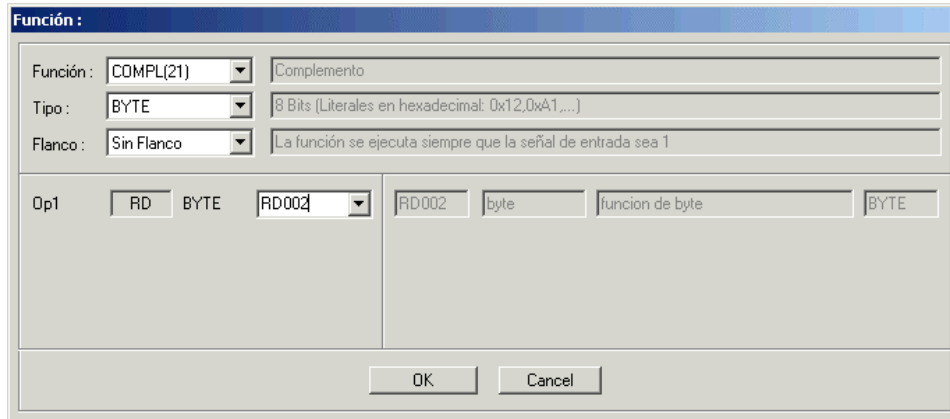
OK Cancel

Ejemplo función “rol”

- En este ejemplo la función “rol” desplaza hacia la izquierda los bits de RD015, introduce el valor del CF en el bit menos significativo de RD015 y el bit de más significativo pasa a ser almacenado en el CF.

9.6.21 Función “COMPL”:

- La función “compl” (complemento) conmuta el valor de los bits de las variables RD. De “0” a “1” y viceversa.
 - Operandos:
 - Op1: contiene la variable RD a la que queremos aplicarle la función complemento.



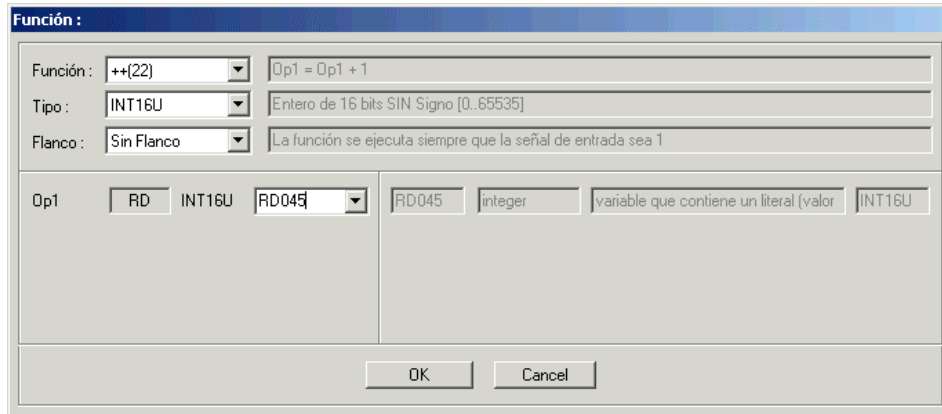
Ejemplo función “compl”

- En este ejemplo la función “compl” cambia el valor de cada uno de los bits, que componen la RD002. Por ejemplo podemos suponer el siguiente ejemplo

RD002 antes de aplicarle “compl”	0110 1100
RD002 después de aplicarle “compl”	1001 0011

9.6.22 Función “++”:

- La función “++” (incremento unario) suma **uno** a las variables tipo **Int.** (integers)
 - Operandos:
 - Op1: contiene la variable RD del tipo **Int.** a la que queremos sumarle **uno** a su valor.



Ejemplo función “++”

- En este ejemplo la función “++” suma uno a RD045 (INT16 sin signo), un ejemplo sería el caso siguiente.

RD002 antes de aplicarle “++”	2002
RD002 después de aplicarle “++”	2003

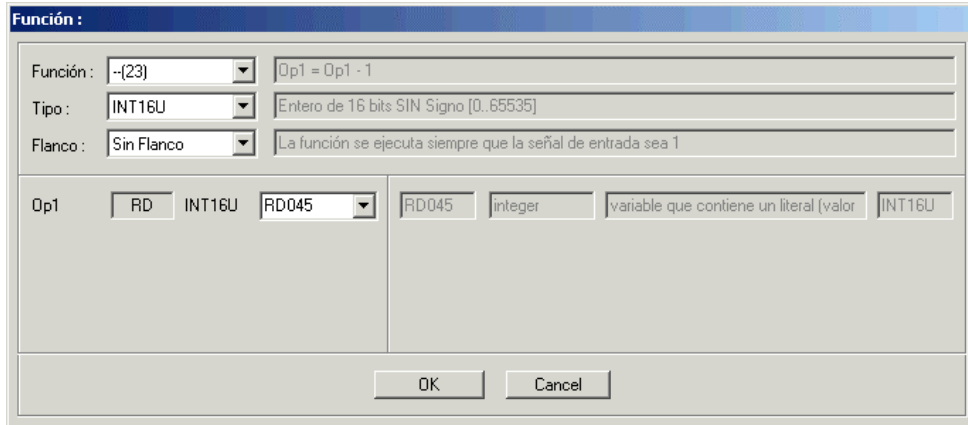
Nota: al igual que pasa en cualquier entorno de programación cuando la función ++ suma por encima del límite de la variable su valor pasa automáticamente a 0 empezando a contar otra vez a partir de ahí.

Ej: en el caso de un Int8U cuyo valor es 255 (límite superior) si lo incrementamos:

255 ++ = 0

9.6.23 Función "--"

- La función "--" (decremento unario) resta **uno** a las variables tipo **Int.** (integers)
 - Operandos:
 - Op1: contiene la variable RD del tipo **Int.** a la que queremos restarle **uno** a su valor.



Ejemplo función "--"

- En este ejemplo la función "--" resta uno a RD045 (INT16 sin signo), un ejemplo sería el caso siguiente.

RD002 antes de aplicarle "--"	2002
RD002 después de aplicarle "--"	2001

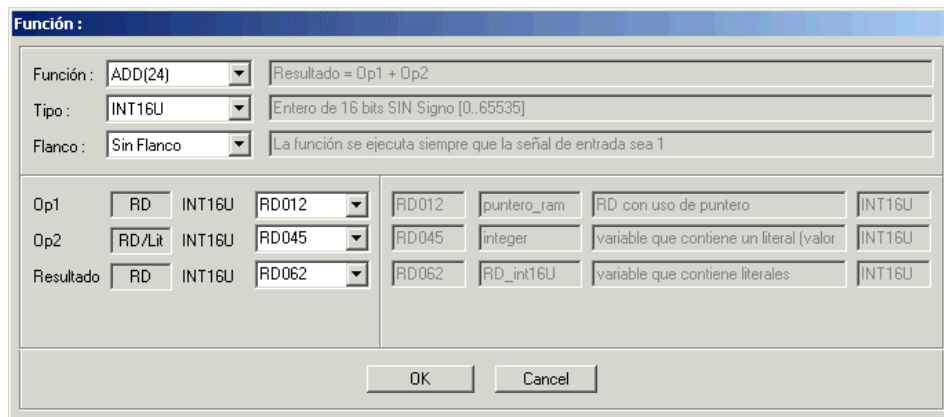
Nota: al igual que pasa en cualquier entorno de programación cuando la función -- por debajo del límite de la variable su valor pasa automáticamente al límite superior empezando a descontar otra vez a partir de ahí.

Ej: en el caso de un Int8U cuyo valor es 0 (límite inferior) si lo decrementamos:

0 -- = 255.

9.6.24 Función “ADD”:

- La función “add” (suma) realiza una suma de dos variables tipo **Int.** O de una **Int.** con un literal, colocando el resultado en otra variable o en una de ellas.
 - Operandos:
 - Op1: contiene, la dirección del primer operando de la operación “add”.
 - Op2: contiene, el valor (literal en hexadecimal) o dirección del segundo operando de la operación “add”.
 - Resultado: corresponde a la dirección donde se va a colocar el resultado de la operación “add”.



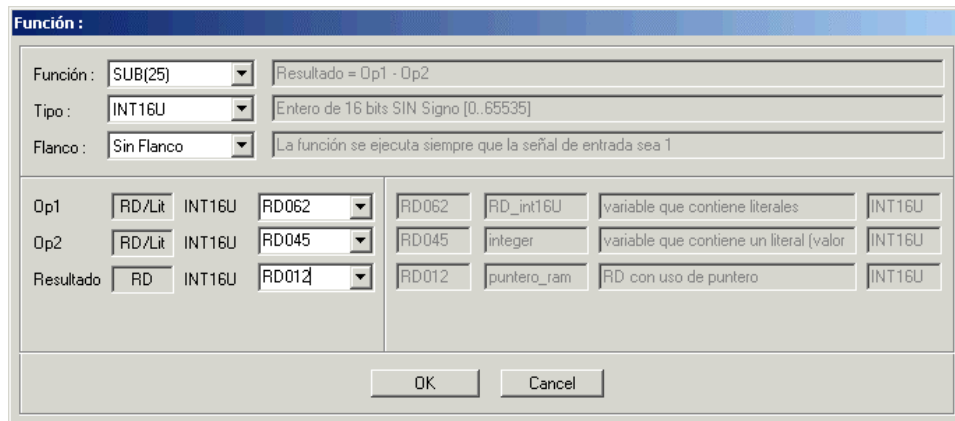
Ejemplo función “add”

- En este ejemplo la función “add” realiza una suma entre los literales de 16bit (sin signo) RD012 y RD045 y coloca el resultado en RD062.
- Un ejemplo podría ser el siguiente.

Op1 RD012	12563
Op2 RD045	10015
Resultado RD062	12578

9.6.25 Función “SUB”:

- La función “sub” (resta) realiza una suma de dos variables tipo **Int.** o de una **Int.** con un literal, colocando el resultado en otra variable o en una de ellas.
 - Operandos:
 - Op1: contiene, la dirección del primer operando de la operación “sub”.
 - Op2: contiene, el valor (literal en hexadecimal) o dirección del segundo operando de la operación “sub”.
 - Resultado: corresponde a la dirección donde se va a colocar el resultado de la operación “sub”.



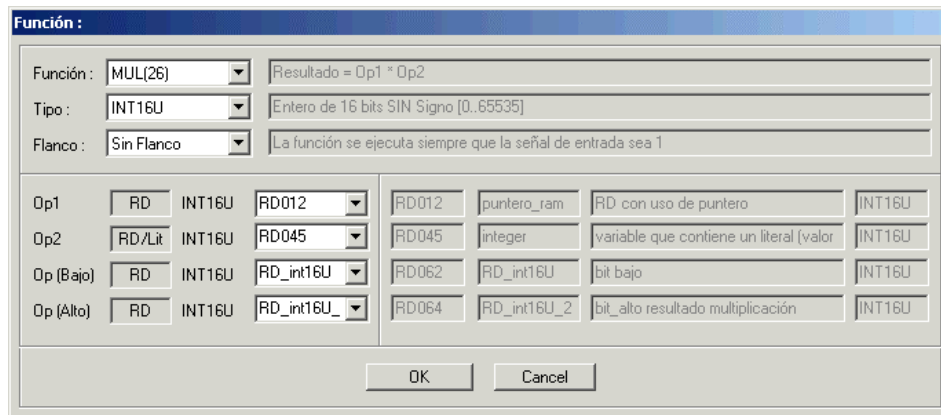
Ejemplo función “sub”

- En este ejemplo la función “sub” realiza una resta entre los literales de 16bit (sin signo) RD062 y RD045 y coloca el resultado en RD012.
- Un ejemplo podría ser el siguiente.

Op1 RD062	12563
Op2 RD045	10015
Resultado RD012	2528

9.6.26 Función “MUL”:

- La función “mul” (multiplicar) realiza el producto de dos variables tipo **Int.** o de una **Int.** Con un literal, colocando el resultado en otra/s variable/s o en una de ellas, dependiendo si el producto sobrepasa la capacidad del tipo de integer utilizado.
 - Operandos:
 - Op1: contiene, la dirección del primer factor de la operación “mul”.
 - Op2: contiene, el valor (literal en hexadecimal) o dirección del segundo operando de la operación “mul”.
 - Op (bajo): corresponde a la dirección donde se va a colocar el resultado de la operación “mul”.
 - Op (alto): si se produce un “*Overflow*” debido a que el resultado del producto es de un tipo de integer mayor que el de los factores de la multiplicación, se almacenara la parte *alta* del resultado en este operando.



Ejemplo función “mul”

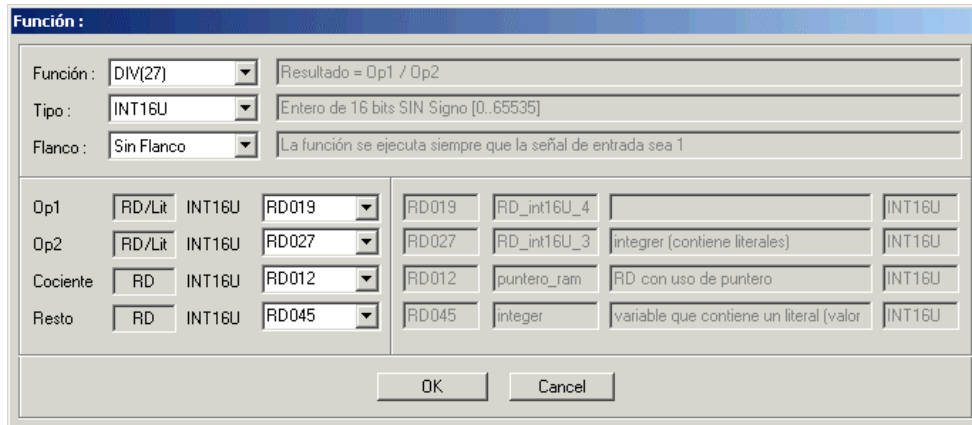
- En este ejemplo la función “mul” realiza una multiplicación entre los integers de 16bit (sin signo) RD012 y RD045 y coloca el resultado en RD062 y RD064 (en este último se almacena la parte alta si se produce Overflow).
- Un ejemplo podría ser el siguiente.

Op1 RD062	5000	(0x1388)
Op2 RD045	50000	(0xC350)
Resultado de la operación	250000000	(0x0EE6 B280)
Resultado bajo RD062		(0xB280)
Resultado alto RD064		(0x0EE6)

NOTA: se puede apreciar como cuando se produce un *overflow* es necesario utilizar otra RD, en este caso la parte alta del producto se almacena en la otra variable adicional RD064 del mismo tipo que las demás.

9.6.27 Función “DIV”:

- La función “mul” (multiplicar) realiza el producto de dos variables tipo **Int.** o de una **Int.** con un literal, colocando el resultado en otra/s variable/s o en una de ellas, dependiendo si el producto sobrepasa la capacidad del tipo de integer utilizado.
 - Operandos:
 - Op1: contiene, la dirección del *Dividendo* de la operación.
 - Op2: contiene, el valor (literal en hexadecimal) o dirección del *Divisor* de la operación.
 - Cociente: corresponde a la dirección donde se va a colocar el cociente de la función “div”.
 - Resto: corresponde a la dirección donde se va a colocar el resto de la función “div”.



Ejemplo función “div”

- En este ejemplo la función “div” realiza una división de los integers de 16bit (sin signo) RD019 (dividendo) y RD027 (divisor) y coloca el cociente en RD062 y el resto en RD064. Un ejemplo podría ser el siguiente.

Op1 RD062	50000	(0xC350)
Op2 RD045	5000	(0x1388)
Resultado de la operación	10	(0x000A)
Cociente RD062	10	(0x000A)
Resto RD064	0	(0x0000)

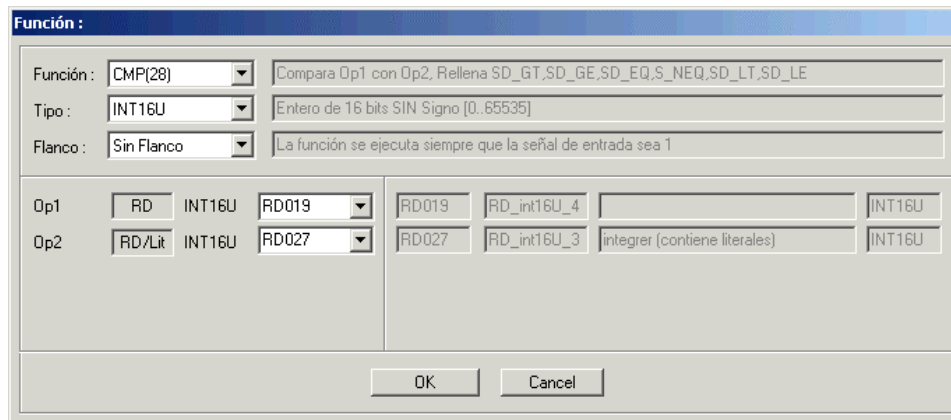
9.6.28 Función “CMP”:

- La función “cmp” (comparación) realiza una comparación de una variable, con una referencia que puede ser otra variable o un literal. Las variables usadas en la función de comparación solo pueden ser del tipo **Integers** (contienen valores numéricos).
- La comparación que se lleva a cabo es $OP1 \geq OP2$
- El resultado de la comparación activa en función del resultado los diferentes flags:

Flag	Denominación
SD-EQ	Igual
SD-NEQ	No Igual
SD-GT	Mayor
SD-GE	Mayor o igual
SD-LT	Menor
SD-LE	Menor o Igual

Estos flags se pueden utilizar como contactos (del tipo *Special Data*):

- Operandos:
- Op1: contiene, la dirección RD a comparar.
- Op2: contiene, el valor (literal en hexadecimal) o dirección que se va utilizar como referencia de la comparación.



Ejemplo función “Cmp”

- En este ejemplo la función “div” realiza la comparación de los integers de 16bit (sin signo) RD019 \geq RD027 dependiendo del resultado de la comparación activa o desactiva los Flags especificados arriba, aquí tenemos 2 ejemplos:

Op1: RD019	2700	2700
Op2: RD027	150	2700
Resultado de la operación	Op1 > Op2	Op1 = Op2
SD-EQ Igual	Off	On
SD-NEQ No Igual	On	Off
SD-GT Mayor	On	Off
SD-GE Mayor o igual	On	On
SD-LT Menor	Off	Off
SD-LE Menor o Igual	Off	On

9.6.29 Función “EQUAL”:

- La función “equal” (igualdad) realiza una comparación de una variable, con una referencia que puede ser otra variable o un literal. A diferencia de la función “**cmp**” esta función no va orientada a las variables del tipo Integers, esta función solo acepta variables de tipo byte, word etc...(conjuntos de bits, no valores numéricos).
- La comparación que se lleva a cabo es $OP1 = OP2$
- El resultado de la comparación activa en función del resultado los diferentes flags:

Flag	denominación
SD-EQ	Igual
SD-NEQ	No Igual

Estos flags se pueden utilizar como contactos (del tipo *Special Data*):

- Operandos:
- Op1: contiene, la dirección RD a comparar.
- Op2: contiene, el valor (literal en hexadecimal) o dirección que se va utilizar como referencia de la comparación.

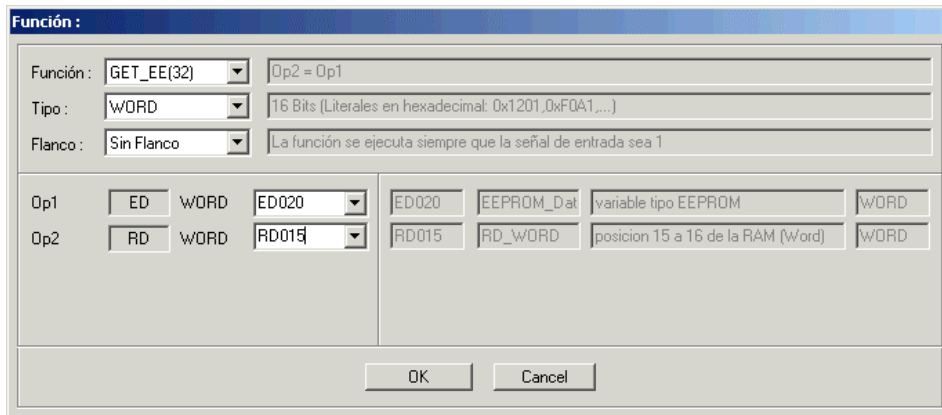
Ejemplo función “equal”

- En este ejemplo la función “equal” realiza la comparación las RD tipo *WORD*: $RD015 = RD017$ dependiendo del resultado de la comparación activa o desactiva los Flags especificados arriba, aquí tenemos 2 ejemplos:

Op1: RD015	0x1256	0x1256
Op2: RD017	0x1258	0x1256
Resultado de la operación	$Op1 \neq Op2$	$Op1 = Op2$
SD-EQ Igual	Off	On
SD-NEQ No Igual	On	Off

9.6.30 Función “GET_EE”:

- La función “get_ee” (leer EEPROM) lee una posición especificada de la EEPROM (direcciones ED) y copia el valor leído a una posición RD especificada.
 - Operandos:
 - Op1: contiene la ED que va a ser leída.
 - Op2: contiene la RD donde se va a almacenar el valor leído.



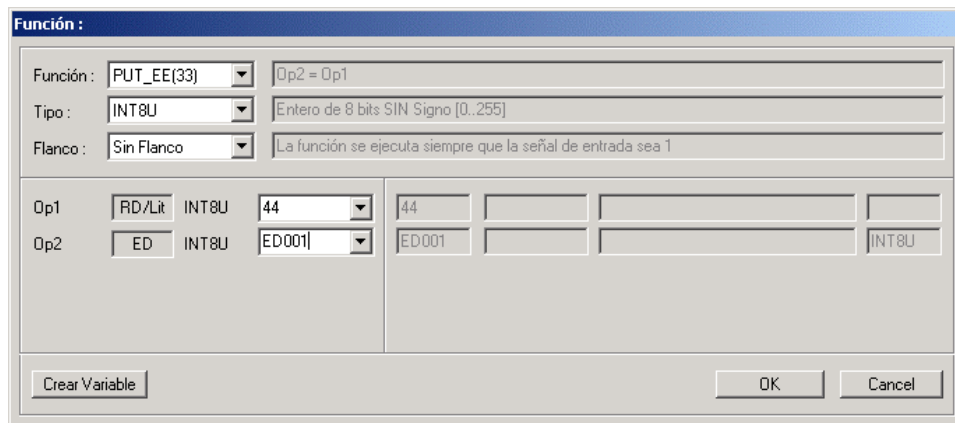
Función:	GET_EE(32)	Op2 = Op1
Tipo:	WORD	16 Bits [Literales en hexadecimal: 0x1201,0xF0A1,...]
Flanco:	Sin Flanco	La función se ejecuta siempre que la señal de entrada sea 1
Op1	ED WORD ED020	ED020 EEPROM_Dat variable tipo EEPROM WORD
Op2	RD WORD RD015	RD015 RD_WORD posicion 15 a 16 de la RAM (Word) WORD

Ejemplo función “get_ee”

- En este ejemplo la función “get_ee” realiza una lectura de la ED020 y copia su valor a la posición RD015.

9.6.31 Función “PUT_EE”:

- La función “put_ee” (colocar en EEPROM) lee una posición especificada RD o un valor literal y copia dicho valor a una posición EEPROM (direcciones ED) especificada.
 - Operandos:
 - Op1: contiene la RD que va a ser leída o valor literal.
 - Op2: contiene la ED donde se va a almacenar el valor del Op1.



Función:	PUT_EE(33)	Op2 = Op1			
Tipo:	INT8U	Entero de 8 bits SIN Signo [0..255]			
Flanco:	Sin Flanco	La función se ejecuta siempre que la señal de entrada sea 1			
Op1	RD/Lit INT8U	44	44		
Op2	ED INT8U	ED001	ED001		INT8U

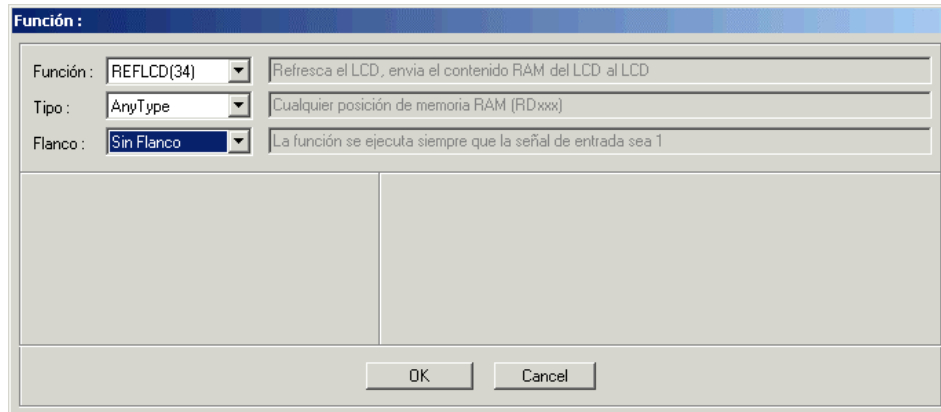
Crear Variable OK Cancel

Ejemplo función “put_ee”

- En este ejemplo la función “put_ee” copia el valor literal “44” a la posición ED020 (EEPROM).

9.6.32 Función “REFLCD”:

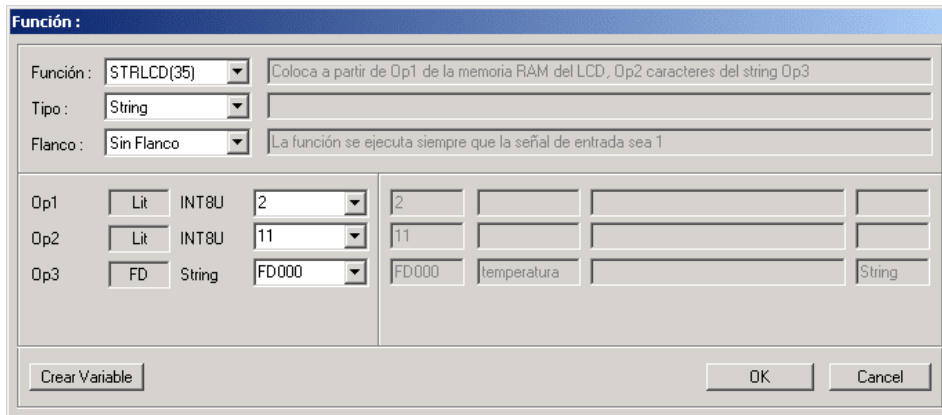
- La función “reflcd” (refrescar lcd) envía al LCD los datos almacenados en la memoria RAM reservada al lcd.



Ejemplo función “reflcd”

9.6.33 Función “STRLCD”:

- La función “strlcd” (string lcd) coloca una variable FD (cadena de caracteres) en la memoria RAM reservada al LCD. La función 34 “reflcd” se encarga de enviar esta parte de la memoria al lcd.
 - Operandos:
 - Op1: se especifica a partir de qué posición del LCD (32 posiciones posibles en caso de un LCD de 2x16 y 16 posiciones posibles en el caso de un LCD de 2x8) se va a mostrar el mensaje, el valor se introduce a través de un literal.
 - Op2: determina el número máximo de caracteres de la cadena que se van a mostrar en LCD. Este valor también se define a través de un literal.
 - Op3: contiene, la dirección FD que contiene el “string” o cadena de caracteres que queremos colocar en la memoria RAM reservada al LCD.



Ejemplo función “strlcd”

- En este ejemplo la función “strlcd” envía a la memoria reservada del LCD la cadena de caracteres contenida en la FD000 (temperatura). La cadena de caracteres se mostrará a partir de la posición del LCD especificada en el Op1, en este caso se mostrará a partir de la 2ª celda del LCD (el LCD está formado por las celdas de la 0 a la 31 en el caso de los LCD 2x16 y de 0 a 15 en el caso de los LCD 2x8).
- Se mostrarán tantos caracteres en el LCD como especifique Op2, en este caso 11 caracteres elegidos por ser el mismo número de caracteres que componen FD000 (“temperatura”), si se escoge un valor mayor, las celdas que no contengan texto se mostrarán en blanco.



Microladder ha diseñado las funciones del LCD orientadas a un LCD de 32 caracteres distribuidos en dos filas (2x16) y a un LCD de 16 caracteres distribuidos en dos filas (2x8).

Posiciones de memoria correspondientes al LCD 2x16:

- Cada una de las 32 posiciones de memoria reservadas del LCD corresponden a cada una de las 32 posiciones del LCD, cada posición del LCD puede contener un carácter.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Posiciones LCD 2x16

Posiciones de memoria correspondientes al LCD 2x8:

- Cada una de las 16 posiciones de memoria reservadas del LCD corresponden a cada una de las 16 posiciones del LCD, cada posición del LCD puede contener un carácter.

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15

Posiciones LCD 2x8

Supongamos el siguiente ejemplo en un LCD de 2x16:

Operando	Contenido	Descripción
Op1 (posición)	2	Posición inicial de la cadena en el Lcd
Op2 (nº caracteres)	11	Numero Max. de caracteres a representar en el LCD
Op3 FD000	Temperature	String

De esta forma la función string enviará a la memoria reservada al LCD los datos del string de tal manera que a través de la función “reflcd” serán mostrados según muestra a continuación:

0	1	t	e	m	p	e	r	a	t	u	r	a	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Visualización en el LCD (según los parámetros especificados en “strlcd”).

Se puede apreciar como el string comienza en la posición especificada (2), y solo muestra el número de caracteres especificado (11).

De manera análoga se puede proceder para un LCD de 2x8.

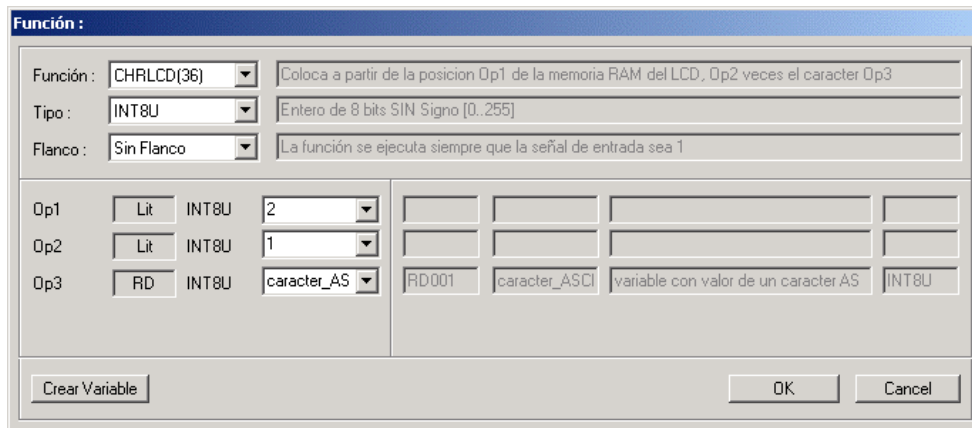
***En la versión de NANO LADDER V1 se dispone de un LCD de 2x8.**

9.6.34 Función “CHRLCD”:

- Esta función está diseñada para enviar a la memoria RAM reservada al LCD caracteres mediante **código ASCII**.
- La función “chrld” (carácter lcd) coloca el valor contenido de una RD o un valor literal, en la memoria RAM reservada al LCD. La función permite repetir ese carácter un número determinado de veces a partir de cualquiera de las 32 posiciones posibles del LCD de 2x16 y de las 16 posiciones posibles del LCD 2x8.

- Operandos:

- Op1: se especifica a partir de qué posición del LCD (32 posiciones posibles LCD 2x16 y 16 posiciones posibles LCD 2x8) se va a mostrar el mensaje, el valor puede ser introducido a través de un literal o del contenido de una RD especificada.
- Op2: determina el número de veces que queremos que se repita el carácter, a partir de la posición inicial especificada en Op1.
- Op3: contiene el literal o la dirección RD que contiene el valor ASCII que queremos colocar en la memoria RAM reservada al LCD.



Función:	CHRLCD(36)	Coloca a partir de la posición Op1 de la memoria RAM del LCD, Op2 veces el carácter Op3		
Tipo:	INT8U	Entero de 8 bits SIN Signo [0..255]		
Flanco:	Sin Flanco	La función se ejecuta siempre que la señal de entrada sea 1		
Op1:	Lit INT8U	2		
Op2:	Lit INT8U	1		
Op3:	RD INT8U	caracter_AS	RD001	caracter_ASCII variable con valor de un caracter AS INT8U

Crear Variable OK Cancel

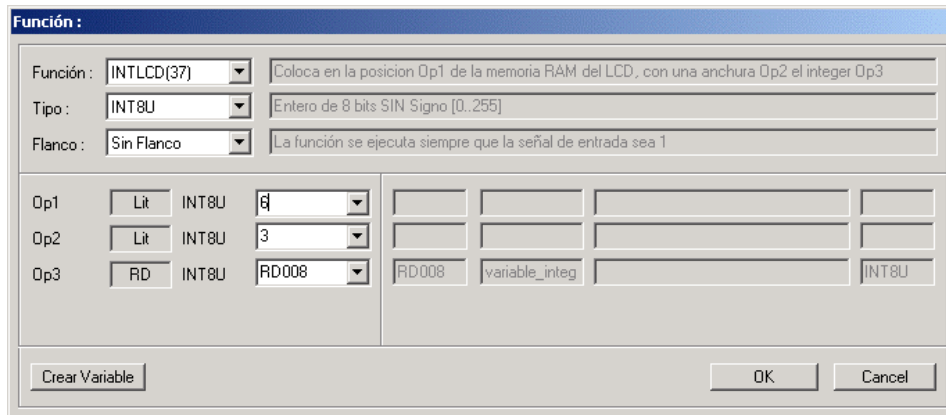
Ejemplo función “chrld”

- En este ejemplo la función “chrld” envía a la memoria reservada del LCD el valor contenido en el Op3 (valor contenido en RD001) a partir de la posición del LCD especificada e Op2 (valor contenido en RD073) y se mostrara tantas veces como especifique Op3 en este caso 1 vez.

***En la versión de NANO LADDER V1 se dispone de un LCD de 2x8.**

9.6.35 Función “INTLCD”:

- Esta función está orientada a enviar a la memoria RAM reservada al LCD valores numéricos ya sean fijos (literales) o contenidos en variables (integers), para una posterior visualización de los mismos.
 - Operandos:
 - Op1: se especifica a partir de qué posición del LCD (0 - 31 posiciones posibles LCD 2x16 y 0-15 posiciones posibles LCD 2x8) se va a mostrar el mensaje, el valor puede ser introducido a través de un literal.
 - Op2: determina el número máximo de caracteres de la cadena que se van a mostrar en LCD. Este valor también se define a través de un literal.
 - Op3: determina el número de veces que queremos que se repita el carácter. la dirección RD que contiene el valor numérico que queremos colocar en la memoria RAM reservada al LCD.



Función :
Función : INTLCD(37) Coloca en la posición Op1 de la memoria RAM del LCD, con una anchura Op2 el integer Op3
Tipo : INT8U Entero de 8 bits SIN Signo [0..255]
Flanco : Sin Flanco La función se ejecuta siempre que la señal de entrada sea 1

Op1 Lit INT8U 6
Op2 Lit INT8U 3
Op3 RD INT8U RD008 RD008 variable_integ INT8U

Crear Variable OK Cancel

Ejemplo función “intlcd”

- En este ejemplo la función “intlcd” envía a la memoria reservada del LCD el valor contenido en la RD008 (Op3) que situará a partir de la posición del LCD especificada e Op1 en este caso la celda 6 del LCD y con la longitud o número de caracteres especificada en Op2, en este ejemplo 3 caracteres como máximo.

***En la versión de NANO LADDER V1 se dispone de un LCD de 2x8.**

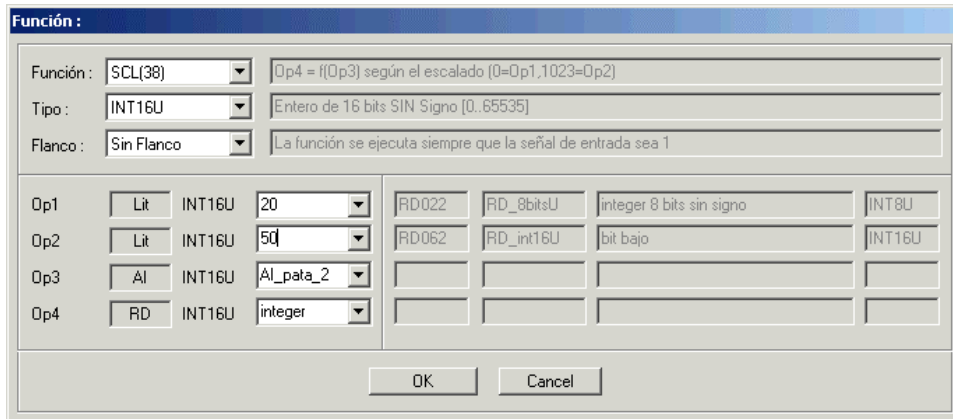
9.6.36 Función “SCL”:

- La función “scl” (escalar) escala los fondos de escala de la entrada analógica seleccionada (0 – 1023) a unos nuevos fondos de escala especificados por el programador.

- Operandos:
- Op1: contiene el nuevo valor asignado al 0 a la entrada analógica.
- Op2: contiene el nuevo valor al 1023 de la entrada analógica.
- Op3: contiene la entrada analógica (AI2, AI3... etc.) a la que se le va a aplicar el escalado.

Nota: la pata correspondiente del micro debe estar definida en el área de trabajo “Configuración” como entrada analógica para el correcto funcionamiento de esta función.

- Op4: almacena el valor de la entrada analógica ya escalado en la RD especificada en este operando.

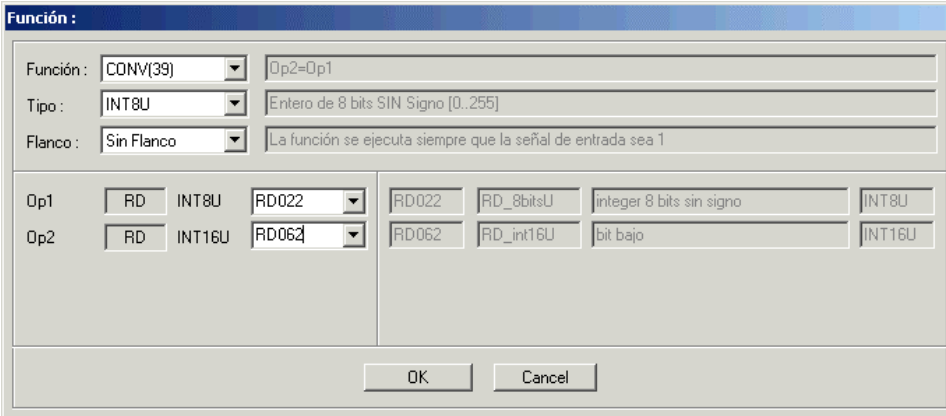


Ejemplo función “scl”

- En este ejemplo la función “scl” escala el valor leído de la entrada analógica de la pata 2 (AI002) ajustando los fondos de escala de (0 – 1023) a (20 – 50) de esta manera, cuando la entrada analógica lea un “0” almacenara en la RD que hemos llamado “integer” un “20” y cuando lea 1023 almacenará un “50”. El escalado también se aplica a los valores comprendidos entre los fondos de escala.
- Un ejemplo práctico de escalado se podría aplicar a un sensor de temperatura cuya lectura a la entrada analógica del micro es “0” a 20°C y “1023” a 80°C, de esta manera, a través del escalado podremos monitorizar directamente el valor de la temperatura que el sensor está midiendo.

9.6.37 Función “CONV”:

- La función “conv” (conversión) convierte un integer de 8 bit a un integer de 16bit.
 - Operandos:
 - Op1: contiene el integer de 8 bit que va a ser convertido.
 - Op2: contiene el nuevo integer de 16 bit resultado de la conversión.
- Nota:** esta función solo trabaja con integers *sin signo*.



Ejemplo función “mov”

- En este ejemplo la función “mov” convierte la RD022 del tipo Int8u al una variable tipo Int16U, en este caso, la RD062.

9.6.38 Función “NOP”:

- La función no realiza ninguna acción.

9.6.39 Función “STOP”:

- Esta función está orientada a utilizarse en caso de un mal funcionamiento del programa, esta función pasa el micro a modo “stop” desactivando las salidas del mismo y dejando de ejecutar el programa de usuario, por otra parte sigue monitorizando las entradas del micro, y las marcas de temporización tipo SD siguen funcionando.
- Para volver a modo “run”, deberemos pulsar el botón dedicado a tal efecto desde el entorno ML-PC, o resetear manualmente el micro.

9.6.40 Función “END”:

- Especifica la línea donde termina el programa de usuario. Si no se pone esta función, Microladder considera la última línea del programa ladder como fin de programa.

9.7 Verificación de Bloques.

Microladder no permite guardar ni transferir programas cuyos bloques no estén bien contruidos, la función de verificación de bloques, permite comprobar si un bloque esta bien o mal construido.

La compilación se produce automática al pasar de un bloque a otro utilizando los cursores, aunque se puede realizar manualmente a través del botón de la barra de herramientas o con la opción “verificar bloque” del menú “CHIP”.



“Verificar Bloque”

En el momento en que se edite un bloque el margen pasa automáticamente a rojo hasta que se verifique correctamente, entonces pasa a ser verde, si después de verificar el bloque el margen continua rojo, es porque la estructura del bloque contiene errores.

9.8 Compilación de Programa.

La compilación incluye funciones de detección de errores relativos a la configuración de las patas del micro, tipos erróneos de variables contenidas en funciones, detección de salidas y timers duplicados, timers no activados por una función y usados como contactos, variables no definidas... además detecta si un bloque no ha sido verificado. Es el último paso antes de transferir el programa al micro. Se puede acceder a esta opción a través de “Compilar Programa” en el menú “CHIP” o con el botón dedicado a este efecto de la barra de tareas:



“Compilar programa”

Para guardar los programas en *Ladder* solo es necesario que estén verificados correctamente, sin embargo para transferirlos al chip, es necesario compilarlos y que dicha compilación este libre de errores.

10 Monitorización en Microladder.

10.1 Introducción.

Una de las características principales que incorpora Microladder es la Monitorización. Desde Microladder-PC se puede monitorizar el estado de cada una de las entradas y salidas así como el valor instantáneo de las variables y funciones utilizadas en el programa.

10.2 La monitorización.

Para activar la monitorización es necesario haber conectado ML-PC a ML-chip a través de *Conectar*. Una vez conectado, para monitorizar se debe presionar el botón de activar monitorización o A través de la opción Monitorizar desde el menú *CHIP*.



“Activar Monitorización”

La función de monitorización se puede llevar a cabo desde las áreas de trabajo *Ladder* y *Configuración*.

Desde el área de trabajo *Ladder*, se puede monitorizar:

- El estado de los contactos.
- El estado de las salidas digitales.
- El estado de las entradas analógicas y digitales.
- Los valores que adoptan las funciones.
- Los valores de las variables que forman el programa.

Desde el área de trabajo *Configuración*, además de comprobar la correcta configuración de las patas, se puede monitorizar el estado o valor de cada una de ellas.

10.3 Cambio de valores durante la monitorización.

Durante la monitorización se puede cambiar desde el área de trabajo *Ladder*, el valor de los contactos asociados a Variables RD de *Bit*. Para ello se debe hacer clic derecho sobre el contacto deseado y elegir entre las dos opciones posibles (*Poner a “0”* o *Poner a “1”*). No se puede sin embargo cambiar el valor del resto de contactos (asociados a entradas digitales, Timers etc...).

10.4 Monitorización en modo *Run* y modo *Stop*.

En modo *Run* se monitorizan el estado de los contactos y de las entradas y salidas del micro, así como el valor instantáneo de las diferentes variables y funciones que forman el programa.



Botón “modo Run”

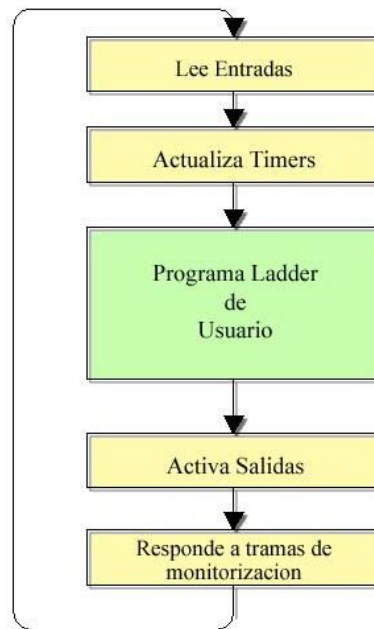
En modo *Stop* el microcontrolador detiene la ejecución del programa, por lo tanto, se desactiva el control sobre las salidas y detiene la ejecución de las funciones, por lo que en la monitorización de estas no produce, sin embargo se sigue monitorizando el estado de los contactos y el de las entradas del micro tanto analógicas como digitales así como los SD de tiempos (SD_0_1seg, SD_0_5seg, SD_1seg).



Botón "modo Stop"

10.5 Monitorización de las Variables SD asociadas a la comparación.

Tal y como se aprecia en el siguiente gráfico Microladder responde a las tramas de monitorización al final de cada ciclo de scan:



Ciclo De Scan

Cada comparación realizada en el programa de usuario sobrescribe el estado los SD de la comparación anterior.

De esta manera, debido a que las respuestas a las tramas de monitorización se produce después del programa de usuario, a la hora de monitorizar solo se visualiza el estado de los SD correspondientes a la última comparación realizada en dicho programa. El hecho de que solo monitorice la última comparación no significa que afecte a la funcionalidad del programa.



10.6 Agradecimientos

Para Finalizar este manual se deben dar los debidos Agradecimientos:

Sin dudas agradecer a quien fuel el creador de este potente Software y a quien considero un Colega de gran Actitud y con un gran Conocimiento en la Materia y en la Educación

ING. PACO TORTOSA

Quien ha permitido que esto pudiera seguir adelante y podamos usar MICROLADDER.

