



UNIVERSIDAD DE VALLADOLID

**ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN**

**PROYECTO FIN DE CARRERA
INGENIERO EN ELECTRÓNICA**

**Diseño de un Ensamblador – Simulador
para el Microcontrolador i8051/8052**

AUTOR: César Junquera Luppi

TUTOR: Jesús Arias Alvarez

Febrero 2000

TITULO: **Diseño de un Ensamblador – Simulador para el
Microcontrolador i8051/8052**

AUTOR: César Junquera Luppi

TUTOR: Jesús Arias Alvarez

DEPARTAMENTO: Electricidad y Electrónica

Miembros del Tribunal

PRESIDENTE: J. Emiliano Rubio García

VOCAL: Jesús Arias Alvarez

SECRETARIO: Jesús Hernández Mangas

FECHA DE LECTURA:

CALIFICACIÓN:

RESUMEN DEL PROYECTO

El presente proyecto consta de dos partes diferenciadas. En la primera se desarrolla un ensamblador de código máquina para los microcontroladores de Intel 8051/8051 partiendo del ensamblador XASM y realizando las modificaciones necesarias para que sea capaz de procesar código de estos microcontroladores. En la segunda se crea un entorno de simulación para el sistema operativo Windows que contempla el funcionamiento de todas las partes integrantes de estos microcontroladores.

ABSTRACT

This project has two independent parts. The first of them develops an assembler for the Intel 8051/8051 microcontrollers, starting from XASM assembler and modifying it to be able to process these microcontrollers code. In the second part a simulation environment is created for the Windows operating system that implements the operation of the whole microcontroller components.

PALABRAS CLAVE

Microcontrolador, Intel, Simulador, Ensamblador, 8051, 8052

ÍNDICE

1 INTRODUCCIÓN	8
1.1 OBJETIVOS.....	8
1.2 JUSTIFICACIÓN.....	8
1.3 DESCRIPCIÓN DEL MICROCONTROLADOR	9
1.3.1 Sistema de memoria	9
1.3.2 Puertos de Entrada/Salida.....	10
1.3.3 Temporizadores/Contadores.....	10
1.3.4 Puerto Serie	11
1.3.5 Interrupciones	12
2 ENSAMBLADOR - ENLAZADOR.....	14
2.1 XASM. JUSTIFICACIÓN.....	14
2.2 ENSAMBLADOR	15
2.2.1 Formato del fichero	15
2.2.1.1 Formato de las entradas	15
2.2.1.2 Símbolos y Expresiones	16
2.2.1.3 Directivas del Ensamblador	20
2.2.1.4 Uso de la Línea de Comando	25
2.2.1.5 Errores	26
2.2.1.6 Ficheros de Salida	26
2.3 ENLAZADOR	28
2.3.1 Formato de Entrada	28
2.3.2 Formato del módulo objeto	29
2.3.2.1 Cabecera	29
2.3.2.2 Línea de Módulo	29
2.3.2.3 Línea de Símbolo	29

2.3.2.4	Línea de Área	30
2.3.2.5	Línea T	30
2.3.2.6	Línea R.....	30
2.3.2.7	Línea P.....	31
2.3.3	Ejecución del Enlazador	31
2.3.4	Formato del Fichero de Salida	32
2.3.4.1	Formato Motorola S19	32
2.3.3.2	Formato Intel Hex .ihx	33
2.4	CARACTERISTICAS ESPECIFICAS DEL ENSAMBLADOR.	33
2.4.1	Registros del Microcontrolador.....	33
2.4.2	Modos de Direccionamiento	37
2.4.3	Mnemónicos Reconocidos y Equivalencia con los Opcodes.	38
2.4.3.1	Ordenados por código	38
2.4.3.2	Ordenado por Mnemónico	48
2.5	PROGRAMACIÓN	58
2.5.1	8051pst	59
2.5.2	8051ext.c	60
2.5.3	8051.h.....	60
2.5.4	8051adr.c.....	60
2.5.5	8051mch.c	62
3	SIMULADOR.....	64
3.1	MANUAL DE USUARIO	64
3.1.1	Ejecución del Programa	64
3.1.2	Ventana Principal.....	64
3.1.1.1	Barra de Menús	65
3.1.1.2	Memoria de Programa.....	65
3.1.1.3	Memoria Interna y Externa	65

3.1.1.4 Ventana de Special Frame Registers	66
3.1.2. Barra de Herramientas.....	68
3.1.3 Barra de Menú	69
3.1.3.1 Menú Archivo	70
3.1.3.2 Menú Edición.....	70
3.1.3.3 Menú Ver.....	72
3.1.3.4 Menú Ejecutar	73
3.1.3.5 Menú Depurar	74
3.1.3.6 Menú Opciones	78
3.1.3.7.- Menú Ventana	80
3.2 PROGRAMACIÓN	80
3.2.1 Variables globales utilizadas	80
3.2.2 Inicio de la Aplicación	89
3.2.3 Configuración de Hardware	91
3.2.4 Ejecución del programa simulado.....	93
3.2.5 Añadiendo Variables.....	96
3.2.6 Definiendo Breakpoints.....	96
3.2.7 Abriendo Ficheros de Código	97
3.2.8 Abriendo ficheros de memoria de datos.....	99
3.2.9 Búsqueda de Datos en Memoria	99
3.2.10 Otras Opciones del Menú	101
3.2.11 Tratamiento de la Barra de Herramientas	102
3.2.12 Cuadro de Diálogo GenIntForm.....	102
3.2.13 Cuadro de Diálogo BuscarForm.....	103
3.2.14 Cuadro de Diálogo DefBreakForm	105
3.2.15 Cuadro de Diálogo AddVarForm.....	107
3.2.16 Cuadro de Diálogo IBo xForm.....	108

3.2.17	Ventana de Configuración de Hardware ConfHwForm.....	108
3.2.18	Ventana de Memoria Interna IntMemForm.....	110
3.2.19	Ventana de Memoria Externa ExtMemForm.....	114
3.2.20	Ventana de Código CodeMemForm	114
3.2.21	Cuadro de Diálogo SFRBitsForm.....	114
3.2.22	Ventana de SFR´s SFRsForm	116
3.2.23	Ventana de Variables VarForm.....	118
3.2.24	Ventana PresentaForm	119
3.2.25	Ventana Terminalform.....	119
3.2.26	Funciones de Lectura de Memoria	119
3.2.27	Funciones de Escritura en Memoria	121
3.2.28	Otras Funciones	122
3.2.29	Listado de Funciones.....	127
3.2.29.1	AddVarForm	127
3.2.29.2	BuscarForm.....	127
3.2.29.2	CodeMemForm	127
3.2.29.3	ConfHwForm	127
3.2.29.4	DefBreakForm.....	128
3.2.29.5	ExtMemForm	128
3.2.29.6	GenIntForm.....	128
3.2.29.7	IBox.....	129
3.2.29.8	IntMemForm	129
3.2.29.9	SfrBitsForm.....	129
3.2.29.10	SfrsForm.....	129
3.2.29.11	VarForm	130
3.2.29.12	MdiForm.....	130
3.2.29.13	Modulo1	131

4 PROGRAMAS EJEMPLO	133
4.1 8051.ASM	133
4.2 INTERRUP.ASM.....	133
4.3 COMPCAD.ASM	134
5 BIBLIOGRAFÍA.....	135

1 INTRODUCCIÓN

1.1 OBJETIVOS

Los objetivos del proyecto son dos:

Por una parte se pretende construir un compilador de lenguaje ensamblador para el microcontrolador. Para ello se partirá de la base de los ensambladores XASM, que permiten construir nuevos ensambladores modificando una pequeña parte del código. Este compilador estará escrito en C estándar para ser portable a diferentes sistemas operativos.

El segundo objetivo es la realización de un simulador - depurador de funcionamiento del microcontrolador. Como simulador será capaz de reflejar el funcionamiento de todos los dispositivos y características propias del microcontrolador (puerto serie, temporizadores, registros, interrupciones,...). Como depurador poseerá las opciones típicas en estas utilidades: ejecución paso a paso, breakpoints, visor de variables, control del estado de la memoria y registros,...

El programa se realizará en Visual Basic v5.0 lo que garantiza una buena integración con el entorno operativo Windows y un manejo bastante intuitivo.

El proyecto se acompañará de varios ficheros de ejemplo para comprobar el correcto funcionamiento de ambos.

1.2 JUSTIFICACIÓN

Desde el principio, los microcontroladores han sido componentes muy valorados por los desarrolladores de hardware, ya que unen la flexibilidad del microprocesador a la hora de determinar la función que se debe realizar, con la comodidad de tener incluidos periféricos como temporizadores, convertidores A/D, puertos serie,... haciendo mucho más sencillo el diseño de los sistemas basados en ellos.

Es esta flexibilidad y la complejidad de las tareas que se les asigna, la que hace que cometer un error en el algoritmo programado sea frecuente. Dado que además suelen ser el núcleo funcional del dispositivo desarrollado, se hace necesaria una forma de asegurarse de que el microcontrolador va a funcionar como se espera. En este punto los simuladores se presentan como una opción sencilla y eficaz de comprobar la programación.

Entre la gran variedad de microcontroladores existentes en el mercado, se encuentra el Intel 8051/8052, un microcontrolador de 8 bit que si bien ha sido superado en prestaciones por desarrollos más modernos, goza de una popularidad muy elevada y establece las características básicas de toda una familia de microcontroladores. Es por

esto que considero interesante desarrollar un ensamblador – simulador para este microcontrolador.

1.3 DESCRIPCIÓN DEL MICROCONTROLADOR

En este apartado se describe someramente la arquitectura del microcontrolador. Para una descripción más detallada ver el apéndice.

La características principales de este microcontrolador son:

- CPU de 8 bit optimizada para aplicaciones de control.
- Memoria de Programa de 64k.
- Memoria de datos de 64k.
- 4k (8051) o 8k (8052) de memoria de programa en el propio chip (según versiones).
- 128 bytes (8051) o 256 bytes (8052) de memoria de datos en el propio chip.
- 32 líneas bidireccionales de entrada/salida direccionables individualmente.
- 2 (8051) o 3 (8052) contadores/temporizadores.
- UART full duplex
- 6 (8051) o 8 (8052) interrupciones vectorizadas con dos niveles de prioridad.

1.3.1 Sistema de memoria

Este microcontrolador tiene espacios de memoria separados para datos y programa. Ambos espacios son de un máximo de 64K.

La memoria de programa es de sólo lectura y puede tener integrada en el propio chip 4k en el 8051 o 8k en el caso del 8052. Después de un reset el contador de programa apunta a la dirección 0. Los dos primeros bytes de la memoria están reservados para una instrucción de salto al inicio del programa. A continuación se sitúan los 5 espacios reservados para el servicio a las interrupciones de 8 bytes cada uno. A partir de la posición 33h está disponible para almacenar el programa.

La memoria de datos tiene siempre las direcciones bajas en el chip. En el caso del 8051 el tamaño es de 128 bytes. A partir de esta dirección y hasta los 256 se encuentran los registros especiales. Para referirnos a ellos basta un direccionamiento directo a estas posiciones.

En el caso del 8052 el tamaño de esta memoria es de 256 bytes. Para diferenciar la memoria de los registros especiales se hace que los direccionamientos indirectos hagan

referencia a la memoria de datos interna, mientras que los directos se refieren a los registros especiales.

En ambos casos las 8 primeras posiciones están reservadas a espacio para la pila, ya que después de un reset el puntero de pila apunta a la dirección 7h y se utiliza un esquema de predecremento. De la dirección 8 a la 1Fh se encuentran 4 bancos de 8 registros. Estos son los registros de propósito general y sólo un banco podrá ser utilizado al mismo tiempo.

Las siguientes posiciones hasta la 2Fh tienen la propiedad de poder referenciarse bit a bit. Así se definen las direcciones de bit 0 a 7Fh.

El resto de la memoria es memoria de datos genérica.

Cuando se realiza un acceso externo a cualquiera de las dos memorias la dirección es de 16 bits. Para el bus de direcciones se utilizan los puertos P0 y P2, por lo que no deben ser utilizados para entrada/salida cuando tenemos memoria externa.

1.3.2 Puertos de Entrada/Salida

El microcontrolador posee 4 puertos de entrada/salida bidireccionales. Cada uno de ellos posee un buffer conectado con el pin de salida a través de cierta lógica intermedia.

La salida se realiza escribiendo en el buffer de puerto correspondiente. El valor escrito se refleja inmediatamente en los pines. Para configurar un puerto (o pin) como entrada debe estar escrito un 1 en el bit correspondiente del buffer. El valor de entrada no se graba en el registro.

Cuando el puerto 3 se usa como entrada cada bit tiene un significado concreto. Son utilizados como peticiones de interrupción externa, recepción y transmisión serie y disparos externos de los contadores. En el 8051 además los bit 0 y 1 del puerto 1 son usados como disparo externo y señal de captura del temporizador 2

Los puertos 0 y 2 son utilizados además en el direccionamiento de la memoria externa. Estos contienen la dirección, mientras que dos pines del puerto 3 contienen la señal de lectura y escritura. Los contenidos del puerto 2 no son modificados y reaparecen cuando termina el acceso. El puerto 1 queda con el valor FFh escrito. Además este puerto contiene la parte baja de la dirección multiplexada con el dato a leer o escribir en la segunda parte del acceso.

1.3.3 Temporizadores/Contadores

En el 8051 hay dos temporizadores, mientras que en el 8052 hay uno más.

Pueden actuar como temporizadores, incrementándose su valor a cada ciclo máquina o bien contadores, incrementando el valor cuando hay una transición de 1 a 0 en la patilla de disparo externo correspondiente.

En el caso de los temporizadores 0 y 1 hay 4 modos de funcionamiento:

- Modo 0: Es un contador de 8 bit con una preescala de división por 32 (para incrementar 1 necesito 32 impulsos de cuenta o 32 ciclos máquina.) Cuando se produce un overflow se provoca una interrupción. Existe además un control de habilitación externo que permite activar la cuenta sólo por software o bien requiere también la activación de la patilla externa INT0 o INT1 según el temporizador.
- Modo 1: El funcionamiento es similar al modo 0 pero el contador es de 16 bits
- Modo 2: Es un contador de 8 bit con precarga. Al producirse un overflow se pide interrupción y el byte bajo del contador se carga con el valor almacenado en el byte alto.
- Modo 3: Si es el temporizador 1 mantiene la cuenta, lo que equivale a deshabilitarlo. En el caso del temporizador 0 se dividen los bytes alto y bajo como dos contadores de 8 bit independientes. El byte bajo funciona como un contador/temporizador de 8 bit utilizando todos los controles e interrupciones del temporizador 0. La parte alta es en todo caso temporizador y utiliza el control de habilitación y la petición de interrupción del temporizador 1, lo que implica que cuando el temporizador 0 esté funcionando en este modo, el temporizador 1 estará siempre habilitado (A no ser que se configure en modo 3) y que no va a pedir interrupción cuando se produzca un rebosamiento.

En el temporizador 2 también se pueden definir 3 modos de funcionamiento.

- Modo 16 bit con Captura: En este modo se configura como un contador/temporizador de 16 bit. Cuando hay rebosamiento se pide una interrupción. Además se puede configurar para que una transición de 1 a 0 en la patilla externa EXEX2 (Puerto 1 bit 1) provoque una captura del valor del contador en un registro especial, provocándose además una interrupción.
- Modo 16 bit con autorecarga: Es un contador de 16 bit. Cuando hay desbordamiento el contador se carga con el valor previamente almacenado en un registro especial. Esta recarga puede también activarse con la línea externa EXEN2 provocándose una interrupción.
- Generador de tasa de bit: Se utiliza junto con el puerto serie para generar velocidades binarias arbitrarias.

1.3.4 Puerto Serie

El puerto serie que posee el microcontrolador es full duplex, pudiendo transmitir y recibir al mismo tiempo. Posee además buffers de entrada y salida que son accedidos en la misma dirección distinguiendo entre ambos según el acceso sea de lectura o escritura.

La transmisión se inicia escribiendo un dato en el buffer de salida. La recepción se habilita por software activando el bit REN.

Tiene 4 modos de funcionamiento:

- Modo 0: Configuración half duplex. Por RXD entran y salen los bit. Por TXD sale el reloj. Se transmiten 8 bit a una velocidad 1/12 la del reloj del sistema.
- Modo 1: Configuración duplex. Se transmiten 8 bit más uno de inicio y otro de fin. La velocidad binaria es variable. El bit de parada se almacena en RB8.
- Modo 2: Se transmiten 8 bit más uno programable en TB8 además de los de principio y fin. La velocidad se puede escoger entre 1/32 o 1/64 la frecuencia del reloj. El noveno bit se almacena en RB8.
- Modo 3: Este modo es idéntico al modo 2 pero la velocidad binaria es variable.

En los casos en los que la velocidad es variable la podemos regular de dos modos. El primero de ellos es con el temporizador 1: El desbordamiento de este temporizador es el que marca la salida del bit. En el caso de tratarse del 8052 también puedo utilizar el desbordamiento de este contador para marcar el inicio de los bit.

1.3.5 Interrupciones

El 8051 posee 6 interrupciones con 5 vectores mientras que el 8052 tiene 8 con 6 vectores.

Las interrupciones posibles con sus vectores de interrupción son:

<i>Origen</i>	<i>Vector</i>	<i>Descripción</i>
IE0	03H	Interrupción externa 0. Puede ser activada por transición de 1 a 0 o por nivel en la patilla INT0.
TF0	0BH	Desbordamiento del temporizador 0
IE1	13H	Interrupción externa 1. Puede ser activada por transición de 1 a 0 o por nivel en la patilla INT1.
TF1	1BH	Desbordamiento del temporizador 1
RI	23H	Dato recibido por el puerto serie
TI	23H	Dato transmitido por el puerto serie
TF2	2BH	Desbordamiento del temporizador 2
EXTF2	2BH	Recarga/Captura del temporizador 2

Todas estas interrupciones pueden ser habilitadas o deshabilitadas de forma conjunta o individualmente.

Se definen además dos niveles de prioridad. Una vez que se está atendiendo una interrupción no se atenderá otra a menos que la que se está atendiendo sea de prioridad baja y la nueva interrupción sea de prioridad alta. Con éste método el nivel de anidamiento de las interrupciones se ve limitado a dos. La prioridad de cada interrupción se puede definir individualmente. Además, cuando dos interrupciones son pedidas al mismo tiempo se establece un orden de preferencias siendo la de mayor prioridad la que tiene el vector de interrupción más bajo.

El proceso de las interrupciones es el siguiente: Cuando se pide una interrupción, se pone a uno un flag, de modo que cada vez que el microcontrolador termina de ejecutar una instrucción comprueba estos flag y si está activo ejecuta una instrucción LCALL al vector de interrupción correspondiente. Estos flags son borrados por hardware una vez ejecutada la llamada excepto los que se refieren al puerto serie y al temporizador 2, que deberán ser borrados por software.

2 ENSAMBLADOR - ENLAZADOR

2.1 XASM. JUSTIFICACIÓN

El paquete XASM es un conjunto de ensambladores escritos en C con dos partes:

Una parte general común a todos los ensambladores que le confieren al programa las siguientes características:

- Módulos objeto reubicables
- Símbolos globales para enlazar módulos objeto
- Directivas de ensamblado condicional
- Listado de símbolos con formato y en orden alfabético

Una parte dependiente del dispositivo que se encarga de:

- Descripción del dispositivo, orden de los byte e información sobre las extensiones de los archivos
- Directivas propias de cada dispositivo, mnemónicos y opcodes asociados
- Código para procesar los mnemónicos, modos de direccionamiento y directivas especiales

A los ensambladores les acompaña un enlazador común que realiza las siguientes funciones:

- Une las diferentes módulos objeto en una sola imagen de memoria
- Resuelve las referencias mediante símbolos entre diferentes módulos
- Procesa los atributos absoluto, relativo, concatenado,... de los segmentos de programa
- Realiza los cálculos de los direccionamientos relativos a PC
- Asocia el valor absoluto definitivo a cada símbolo.
- Produce mapas de memoria en formato Intel Hex o Motorola S19

Visto esto para crear un nuevo ensamblador - enlazador lo único que hace falta es programar la parte específica del ensamblador. Es por esta forma tan eficaz de aprovechar el código ya escrito y depurado por lo que se utiliza este paquete para la programación del compilador.

2.2 ENSAMBLADOR

2.2.1 Formato del fichero

En este apartado se describen las reglas que debe respetar el fichero de entrada

2.2.1.1 Formato de las entradas

Todo programa fuente está compuesto por una sucesión de entradas. Cada entrada debe ocupar una línea de un máximo de 128 caracteres.

Una entrada tiene los siguientes campos:

[Etiqueta:] Operador operando [;Comentarios]

Los campos de Etiqueta y Comentarios son opcionales

ETIQUETA

Una etiqueta es un símbolo definido por el usuario para referenciar una determinada posición en el programa. Este símbolo pasará a formar parte de la tabla de símbolos. La dirección de memoria a la que referencia será determinada dependiendo de si el segmento es absoluto o relativo

Toda etiqueta terminará con dos puntos (:) y sólo podrá contener los siguientes caracteres:

- Letras de la A a la Z, mayúsculas y minúsculas
- Números del 0 al 9
- Caracteres . \$ _

Sólo son significativos los 8 primeros caracteres y el primero nunca será un número

OPERADOR

Es el que indica la operación a realizar. Es un mnemónico del procesador o bien una directiva del ensamblador.

OPERANDO

Define los elementos sobre los que se ejecuta la acción del Operador.

Pueden ser expresiones o símbolos (sólo si el operador no es un mnemónico) que serán separados por comas si hay más de uno.

COMENTARIOS

Comienzan con un punto y coma (;) y son ignorados por el ensamblador

2.2.1.2 Símbolos y Expresiones

Aquí se describe los componentes genéricos del ensamblador, símbolos, convenciones,...

SET DE CARACTERES

Se permiten los siguientes caracteres

- Letras mayúsculas y minúsculas de la A a la Z
- Números del 0 al 9
- Caracteres . \$ _
- Los descritos en las siguientes tablas:

Tabla 1 Delimitadores de etiqueta y operadores de asignación

:	Delimitador de etiqueta
::	Delimitador de etiqueta global
=	Asignación directa

==	Asignación directa global
----	---------------------------

Tabla 2 Delimitadores de campos y separadores de operandos

Tab	Delimitador de campo
Espacio	Delimitador de campo
,	Separador de operandos
;	Indicador de comentario

Tabla 3 Operadores de ensamblador

#	Indicador de expresión inmediata
.	Contador de posición actual
(Delimitador de expresión
)	Delimitador de expresión.

Tabla 4 Operadores unitarios

<	<FEDC	Extrae el byte inferior de la expresión (DC)
>	>FEDC	Extrae el byte superior de la expresión (FE)
+	+A	Valor positivo de A
-	-A	Complemento a dos de A
~	~A	Complemento a uno de A

'	'A	Valor del carácter A
"	"AB	Devuelve el valor de tamaño doble byte de AB
\	\n \001	Caracteres tipo UNIX o valor octal

Tabla 5 Operadores Binarios

<<	0800<<<4	Rota 4 bit a la izquierda (8000)
>>	0800>>>4	Rota 4 bit a la derecha (0080)
+	A+B	Suma aritmética
-	A-B	Resta Aritmética
*	A*B	Multiplicación con signo (16 bit)
/	A/B	División con signo
&	A&B	Y Lógica
	A B	O Lógica
%	A%B	Modulo (16 Bit)
^	A^B	XOR Lógica

Tabla 6 Operadores de base temporal

0b, 0B	Base Binaria.
0@, 0o, 0O, 0q, 0Q	Base Octal.
0d, 0D	Base Decimal.
0h, 0H, 0x, 0X	Base Hexadecimal.

SÍMBOLOS DEFINIDOS POR EL USUARIO Y LOCALES

Son aquellos símbolos igualados a un valor específico a través de una asignación directa o que aparecen como etiquetas

Admiten todos los caracteres alfanuméricos y `$_`

No deben empezar por número (excepto si son locales) ni tener espacios ni tabulaciones en medio.

Los símbolos locales son símbolos que sólo son válidos dentro de un bloque de símbolos locales. Estos bloques son delimitados por dos etiquetas normales cualquiera. Esto posibilita el poder usar el mismo símbolo en dos bloques diferentes de programa.

Los símbolos locales están formados por un número entero entre 0 y 255 seguidos del carácter `$` (`1$, 25$,...`).

CONTADOR DE POSICIÓN ACTUAL

Se refiere al valor que toma el contador de programa en un punto del código. Su símbolo es el punto (`.`).

así `jmp .+4` saltaría 4 bytes por delante del inicio de la instrucción.

EXPRESIONES

Una expresión es una combinación de términos unidos por operadores binarios. El resultado es de 16 Bit

La jerarquía de evaluación es:

<code>*, /, %</code>
<code>+, -</code>
<code><<, >></code>
<code>^</code>
<code>&</code>
<code> </code>

Esta jerarquía puede modificarse con el uso de paréntesis

Las expresiones pueden ser de tres tipos:

- Reubicable: son aquellas definidas en un área de programa reubicable. Son calculadas con relación a la dirección base del área de programa en cuestión.
- Absoluta: Su valor es fijo. Las expresiones que sólo emplean números y caracteres ASCII son absolutas.
- Externa: Son aquellas que tiene una referencia no definida en el programa actual. Estas referencias sólo se resuelven en el enlazado al juntar todos los módulos fuente.

2.2.1.3 Directivas del Ensamblador

DIRECTIVA `.module`

Formato:

`.module cadena`

Descripción:

Esta directiva hace que la cadena sea incluida en el fichero de salida como un identificador para ese módulo objeto.

DIRECTIVA `.title`

Formato:

`.title cadena`

Descripción:

Hace que la cadena aparezca en la segunda línea de cada página a la hora del listado

DIRECTIVA `.sbttl`

Formato:

`.sbttl`

Descripción:

Hace que la cadena aparezca en la tercera línea de cada página a la hora del listado

DIRECTIVA .page

Formato:

`.page`

Descripción:

Indica un cambio de página en el listado.

DIRECTIVAS .byte y .db

Formato:

`.byte exp ó .db exp`

Descripción:

Almacena el valor exp de 8 bit (truncado del valor de 16 bit de la expresión) en la posición donde aparece. Para almacenar varios valores puedo separarlos por comas.

DIRECTIVAS .word y .dw

Formato:

`.word exp ó .dw exp`

Descripción:

Similar a la anterior pero con valores de 16 bit

DIRECTIVAS .blkb y .blkw

Formato:

`.blkb N, .ds N, .blkw N`

Descripción:

Reserva N posiciones de memoria de 8 bit en los dos primeros casos o de 16 bit en el tercero.

DIRECTIVA .ascii

Formato:

.ascii /cadena/

Descripción:

Escribe en sucesivas posiciones memoria los valores ascii de los caracteres de la cadena. Los // delimitan la cadena. En realidad pueden ser cualquier pareja de caracteres, cuidando que no aparezcan en la propia cadena.

DIRECTIVA .asciz

Formato:

.asciz /cadena/

Descripción:

Similar a la anterior pero al final de la cadena se añade el valor 0.

DIRECTIVA .radix

Formato:

.radix carácter

Descripción:

Establece la base numérica por defecto. El carácter puede ser:

B, b	Binario
O, o, Q, q, @	Octal
D, d, en blanco	Decimal
H, h, X, x	Hexadecimal

DIRECTIVA .even

Formato:

.even

Descripción:

Hace que el contador de posición actual sea par sumándole 1 si es necesario.

DIRECTIVA .odd

Formato:

```
.odd
```

Descripción:

Hace que el contador de posición actual sea impar sumándole 1 si es necesario.

DIRECTIVA .area

Formato:

```
.area nombre [(opciones)]
```

Descripción:

Define secciones de programa y datos independientes.

El nombre debe tener entre 1 y 8 caracteres y puede ser el mismo que el de cualquier símbolo, ya que son independientes.

Las opciones especifican el tipo de área programa o datos.

- ABS: Absoluto.
- REL: Relativo. Su ubicación se podrá modificar a la hora de enlazar el código
- CON: Concatenado. Si existen varias áreas con el mismo nombre se colocará una al final de la otra. No es compatible con ABS
- OVR: Overlay. Si existen varias áreas con el mismo nombre se superpondrán unas con otras
- PAG: paginada. Su tamaño máximo es de 256 byte. El enlazador comprueba que no se exceda ese límite

Las áreas con el mismo nombre deben tener las mismas opciones.

Dos áreas son creadas automáticamente:

- `._ABS.` Contiene todos los símbolos absolutos y su valor
- `_CODE` Es el área de programa y datos por defecto. Es de tipo (REL,CON)

DIRECTIVA .org

Formato:

```
.org exp
```

Descripción:

La expresión absoluta exp se asigna al contador de posición actual. Sólo es válida en áreas absolutas

DIRECTIVA .globl

Formato:

```
.globl sym1, sym2, ...
```

Descripción:

Define símbolos globales

DIRECTIVA .if/.else/.endif

Formato:

```
.if exp /else /endif
```

Descripción:

Directivas de ensamblado condicional. Si la expresión es distinta a cero se ensambla el código entre las directivas .if y .else. Si es cero se ensambla lo comprendido entre .else y .endif

Se permiten 10 niveles de anidamiento.

DIRECTIVA .include

Formato:

```
.include cadena
```

Descripción:

Inserta el fichero de código fuente denotado por 'cadena' en el código actual. Una directiva .page el principio del código insertado y otra al final son añadidas automáticamente.

Se permite hasta 5 niveles de anidamiento.

DIRECTIVA `.setdp`

Formato:

`.setdp`

2.2.1.4 Uso de la Línea de Comando

El formato de la llamada al programa es el siguiente:

`As8051 [-dqxgalosf] fichero1 [fichero2 fichero3 ... fichero6]`

Las opciones quieren decir:

<code>-d</code>	Listado decimal.
<code>-q</code>	Listado octal.
<code>-x</code>	Listado hexadecimal.

Esta base afecta a los ficheros `.lst`, `.rel` y `.sym`

<code>-g</code>	Hace que los símbolos no definidos sean globales.
<code>-a</code>	Hace que todos los símbolos sean Globales.
<code>-l</code>	Crea fichero de listado (<code>.lst</code>)
<code>.o</code>	Crea fichero objeto (<code>.rel</code>)
<code>-s</code>	Crea fichero de símbolos (<code>.sym</code>)
<code>-f</code>	Marca los bytes reubicados con ‘
<code>-f</code>	Marca los byte reubicados según el modo.

El nombre de los ficheros de salida es el del primer fichero de entrada y su contenido es el resultado de ensamblar los ficheros de entrada concatenados.

Si no se especifica la opción `-s`, la tabla de símbolos se añade al listado `.lst`.

2.2.1.5 Errores

El ensamblador puede detectar varios tipos de errores. Estos se reflejarán en pantalla y en el fichero .lst

(.)	Se ha intentado asignar un valor absoluto al contador de posición actual.
(a)	Error en el modo de direccionamiento de un mnemónico.
(b)	Error de límites de página.
(d)	Error de direccionamiento de página.
(i)	Error en directiva <code>.include</code> o estructura <code>.if/.endif</code> .
(m)	Múltiples definiciones de la misma etiqueta, múltiples directivas <code>.module</code> o conflicto de atributos en directiva <code>.area</code>
(o)	Error de directiva o mnemónico o uso de <code>.org</code> en un área reubicable.
(p)	Error de fase: La posición de una etiqueta se ha modificado entre las pasadas 2 y 3 de ensamblado.
(q)	Error de sintaxis.
(r)	Error en reubicación. Generalmente por operaciones lógicas o aritméticas con términos reubicables.
(u)	Símbolo indefinido encontrado.

2.2.1.6 Ficheros de Salida

LISTADO (.lst)

Cada página de este fichero contiene una cabecera de cuatro líneas:

- Nombre del programa ensamblador y número de página.
- Título establecido en la directiva `.title`
- Subtítulo establecido en la directiva `.sbttl`
- Línea en blanco.

Cada línea siguiente contiene cinco campos:

- Campo de error (3 caracteres).
- Contador de posición actual.
- Código generado en formato de byte.
- Número de línea del código fuente.
- Código fuente.

El formato de los números es el escogido en la línea de comandos al invocar el ensamblador.

Hay dos tipos de líneas que no se listan, las que contienen la directiva `.page` y las que contienen `.include` excepto si no se pudo hallar el fichero.

Si se especificó la opción `-f` todos los byte que son reubicados se les antepone el carácter `'`. Si se especificó `-ff` se les antepone con

*	Reubicación de página.
u	Reubicación de byte sin signo.
p	Reubicación relativa al contador de programa de un byte o del byte bajo de una palabra.
q	Reubicación relativa al contador de programa del byte alto de una palabra.
r	Reubicación del byte bajo.
s	Reubicación del byte alto.

FICHERO DE SÍMBOLOS (.sym)

La tabla de símbolos tiene dos partes:

- Lista de símbolos y etiquetas por orden alfabético.
- Lista de las áreas de programa definidas.

La primera contiene

- el número del área de programa.
- el símbolo.
- el valor del símbolo relativo a la base del área o **** si es un símbolo no definido.
- Los caracteres G- global, R- reubicable, X- Externo.

La segunda tiene la correspondencia área - número, el tamaño y los atributos.

FICHERO OBJETO (.rel)

Contiene la información necesaria para que el enlazador una los diferentes módulos. Será explicado con mayor detalle en la sección del enlazador.

2.3 ENLAZADOR

2.3.1 Formato de Entrada

El fichero de entrada es un fichero ASCII conteniendo la información necesaria por el enlazador para unir los múltiples módulos objeto en una imagen de memoria completa.

El módulo objeto contiene los siguientes designadores:

[XDQ]	[HL]
X	Base hexadecimal.
D	Base decimal.
Q	Base octal.
H	Primero el byte más significativo.
L	Primero el byte menos significativo.
H	Cabecera.
M	Módulo.
A	Area.

S	Símbolo.
T	Código objeto.
R	Información de reubicación.
P	Información de paginación.

2.3.2 Formato del módulo objeto

La primera línea de un módulo objeto contiene el especificador de formato [XDQ] [HL] para los siguientes designadores.

2.3.2.1 Cabecera

Formato:

H aa áreas gg símbolos globales

Descripción:

Esta Línea especifica el número de áreas (aa) y el número de símbolos globales (gg) definidos o referenciados en este módulo.

2.3.2.2 Línea de Módulo

Formato:

M nombre

Descripción:

Especifica el nombre del módulo desde el cual este segmento de cabecera fue ensamblado. No aparecerá si la directiva `.module` no fue usada.

2.3.2.3 Línea de Símbolo

Formato:

S cadena Defnnnn

O

S cadena Refnnnn

Descripción:

Define (Def) o referencia (Ref) el símbolo 'cadena' con el valor nnnn. El valor definido es relativo a la dirección base del área actual. Las referencias a símbolos externos tendrán un valor cero. Las referencias a símbolos globales y constantes aparecen antes de la primera definición de área.

2.3.2.4 Línea de Área

Formato:

A etiqueta tamaño ss atributos ff

Descripción:

Define la etiqueta del área, su tamaño (ss) en bytes y sus atributos (ff)

Los atributos se definen:

- OVR/CON bit 2.
- ABS/REL bit 3.
- PAG bit 4.

2.3.2.5 Línea T

Formato:

T xx xx nn nn nn nn ...

Descripción:

Esta línea contiene el código ensamblado siendo xx xx la dirección de offset relativa a la dirección base de la actual área y nn los bytes ensamblados.

2.3.2.6 Línea R

Formato:

R 0 0 nn nn n1 n2 xx xx ...

Descripción:

Esta línea provee información para la reubicación.

El valor nn nn es el índice del área actual.

La información para la reubicación viene dada en grupos de 4 bytes:

1. N1 Modo de reubicación y formato

bit0	byte/word.
bit1	símbolo/area reubicable.
bit2	Reubicación relativa a PC/normal.
bit3	formato para datos tipo byte 2-byte/1-byte.
bit4	Bytes sin signo/con signo.
bit5	Referencia página '0'/normal.
bit6	referencia página 'nnn'.

2. N2 puntero al dato a ser modificado en la línea T precedente.

3. xx xx es el índice del área o símbolo referenciado.

Este grupo se repite por cada elemento que necesite reubicación en la línea T precedente.

2.3.2.7 Línea P

Formato:

P 0 0 nn nn n1 n2 xx xx

Descripción:

Provee la información sobre paginación como se especificó en la directiva .setdp. El formato es idéntico al de la línea R.

La línea T correspondiente tendrá T xx xx aa aa bb bb , donde aa aa es la referencia al área y bb bb la dirección base.

2.3.3 Ejecución del Enlazador

El enlazador se puede ejecutar en modo línea de comando o en modo fichero de comando (.lnk). Las opciones iniciales para el enlazador son:

-c/-f modos comando/fichero.

-p/-n habilito/deshabilito un eco en pantalla del fichero .lnk.

El enlazador se ejecuta como:

ASLINK -(cfpn)

La opciones válidas una vez ejecutado son:

-i/-s	Imagen de salida en formato Intel hex (.ihx)/Motorola(.s19).
-m	Genera un fichero mapa (.map). Contiene un alista de símbolos con direcciones absolutas, tamaños de las áreas enlazadas y otra información.
-xdq	Base del fichero mapa.
ficheros	Ficheros a enlazar. Pueden especificarse varias en una o diferentes líneas.
-b área = expresión	Especifica la dirección base del área. La expresión puede contener constantes y símbolos de los ficheros enlazados.
-g símbolo = expresión	Asigna un valor a un símbolo. . La expresión puede contener constantes y símbolos de los ficheros enlazados.
-e o línea vacía	Finaliza la entrada de opciones..

2.3.4 Formato del Fichero de Salida

2.3.4.1 Formato Motorola S19

El fichero esta formado por varios registros consecutivos ocupando una línea cada uno.

Registro de datos: 'Stnnaaaaddddddddddddddddddddddddddddddd...cc'

S	Indica comienzo del registro.
t	Tipo de registro, '1'=dato, '9'=fin de fichero.
n	Número de bytes en el registro.

a	Dirección de inicio del registro.
d	Bytes con que se llenará la memoria.
c	Código detector de error de $n+a+d$

El código detector se calcula como el complemento a uno de la suma de 8 bit de todos los valores desde nn hasta el final de los datos.

Nn es tres unidades mayor que el número de datos en el registro.

2.3.3.2 Formato Intel Hex .ihx

El fichero esta formado por varios registros consecutivos ocupando una línea cada uno.

Registro de datos: ':nnaaaattdddddddddddddddddddddddddd...cc'

:	Inicio del registro.
n	Número de bytes en el registro.
a	Dirección de inicio del registro.
t	Tipo de registro, '0' dato, '1' fin de fichero.
d	bytes con que se llenará la memoria.
c	código detector de error de $n+a+d$

El código detector se calcula como el complemento a uno de la suma de 8bit de todos los valores desde nn hasta el final de los datos.

El registro de fin de fichero tiene la cuenta de bits puesta a cero.

En ambos formatos los números representados están en formato hexadecimal.

2.4 CARACTERISTICAS ESPECIFICAS DEL ENSAMBLADOR.

En este apartado se describen los modos de direccionamiento, registros, instrucciones,..

2.4.1 Registros del Microcontrolador

Los registros disponibles para el almacenamiento temporal de datos son los siguientes:

- R0 – R7 Registros de propósito general.
- A Acumulador.
- B Registro Auxiliar.

Existen otros registros, llamados SFR (Special Frame Registrers) que controlan el comportamiento del microcontrolador o que tienen una aplicación específica. Son los siguientes:

PSW Registro de estado del programa.

Se compone de los siguientes bit:

C	Acarreo.
AC	Acarreo del tercer dígito al cuarto.
F0	Disponible para el programador.
RS1-RS0	Selector del banco de registros.
OV	Overflow.
-	No usado.
P	Paridad.

SP Registro puntero de la pila (16 Bits).

PCON Registro para el control de potencia.

Se compone de los siguientes bit:

SMOD	Duplicador de la tasa de bits del puerto serie.
-	No usado.
-	No usado.
-	No usado.
GF1	Bit de propósito general.
GF0	Bit de propósito general.

PD	Apagado.
IDL	Suspendido.

DPTR Registro de direccionamiento directo (16 bit). Se divide en DPH y DPL

IE Registro habilitador de Interrupciones.

Se compone de los siguientes bit:

EA	Habilitación de interrupciones.
-	No usado.
ET2	Temporizador 2.
ES	Puerto serie.
ET1	Temporizador 1.
EXT1	Externa 1.
ET0	Temporizador 0.
EXT0	Externa 0.

IP Registro de Prioridad de Interrupciones.

Se compone de los siguientes bit:

-	No usado.
-	No usado.
PS	Puerto serie.
PT1	Temporizador 1.
PXT1	Externa 1.
PT0	Temporizador 0.
PXT0	Externa 0.

SCON Registro de configuración del puerto serie.

Se compone de los siguientes bit:

SM0-SM2	Selector de modo del puerto serie.
REN	Habilitación de recepción.
TB8-RB8	Bit 9º en TX-RX.
TI	Bandera de interrupción de transmisión.
RI	Bandera de interrupción de recepción.

SBUB Registro buffer del puerto serie.

TCON Registro de control de los temporizadores.

Se compone de los siguientes bit:

TF1	Bandera de desbordamiento del temporizador 1.
TR1	Habilitación del temporizador 1.
TF0	Bandera de desbordamiento del temporizador 0.
TR0	Habilitación del temporizador 0.
IE1	Bandera de la interrupción externa 1.
IT1	Habilitación de la interrupción externa 1.
IE0	Bandera de la interrupción externa 0.
IT0	Habilitación de la interrupción externa 1.

TMOD Registro de control de los temporizadores.

Se compone de los siguientes bit:

GATE	control de evento de cuenta del temporizador 1.
C-/T	Función como contador o temporizador .
M1	Selector de modo.
M0	Selector de modo.

Los mismos bit se repiten para el temporizador 0.

T0-T3 Registros temporizadores. Se dividen en THx y TLx.

T2CON Control del temporizador 2.

Se compone de los siguientes bit:

TF2	Bandera de desbordamiento del temporizador 2.
EXF2	Bandera de captura o recarga del temporizador 2.
RCLK	Reloj de recepción del puerto serie.
CLK	Reloj de emisión del puerto serie.
EXEN2	Habilitación de evento externo.
TR2	Habilitación del temporizador.
C-/T2	Contador – temporizador.
CP-/RL2	Captura o Recarga

RCAP2 Registro Captura del temporizador 2 (16 bit). Se divide en RCAP2H y RCAP2L.

P0-P3 Puertos de entrada-salida.

2.4.2 Modos de Direccionamiento

Directo a Registro (Rn)

El dato es el valor que contiene el registro.

Directo (direct)

El dato se transfiere de la posición de memoria indicada en 'direct'.

Indirecto (@Rn)

El dato se transfiere de la posición de memoria que indica Rn.

Inmediato(#dato8,#dato16)

El dato se especifica en la propia instrucción. Es dato es de 8 o 16 bit.

Addr16, addr11

El dato es una dirección de destino. Usados por las instrucciones de salto.

Relativo (reladdr)

Relativo a PC. La dirección final es PC+rel con rel de 8bit con signo.

Indirecto Externo (@DPTR)

El dato proviene de la posición de memoria externa apuntada por DPTR.

Indexados a código (@A+DPTR, @A+PC)

Se suma el acumulador al PC o al DPTR para obtener la dirección final.

Directo a Bit (bit)

Hace referencia a las instrucciones de manejo de bit. Puedo referenciarlos por su posición numérica, por el nombre del bit, o por el nombre del registro seguido de un punto y la posición que ocupa el bit (136,IT0, TCON.0 hacen referencia al mismo bit).

En el caso de direccionamiento directo, si la dirección corresponde con un registro, también puedo referirme a ella mediante el nombre del registro.

2.4.3 Mnemónicos Reconocidos y Equivalencia con los Opcodes.

2.4.3.1 Ordenados por código

<i>Opcode</i>	<i>Bytes</i>	<i>Mnemónico</i>	<i>Direccionamiento</i>
0	1	NOP	
1	2	AJMP	codesddr
2	3	LJMP	codesddr
3	1	RR	A
4	1	INC	A
5	2	INC	dataaddr
6	1	INC	@RO

7	1	INC	@Rl
6	1	INC	RO
0E	1	INC	RI
0A	1	INC	R2
0B	1	INC	R3
0C	1	INC	R4
0D	1	INC	R5
0E	1	INC	R6
0F	1	INC	R7
10	3	JBC	bitaddr,codeaddr
11	2	ACALL	codeaddr
12	3	LCALL	codeaddr
13	1	RRC	A
14	1	DEC	A
15	2	DEC	dataaddr
16	1	DEC	@RO
17	1	DEC	@Rl
16	1	DEC	RO
19	1	DEC	RI
1A	1	DEC	R2
1B	1	DEC	R3
1C	1	DEC	R4
1D	1	DEC	R5
1E	1	DEC	R6
1F	1	DEC	R7
20	3	JB	Bitaddr,codeaddr

21	2	AJMP	Codeaddr
22	1	RET	
23	1	RL	A
24	2	ADD	A,#data
25	2	ADD	A,dataaddr
26	1	ADD	A,@RO
27	1	ADD	A,@R1
28	1	ADD	A,RO
23	1	ADD	A,R1
2A	1	ADD	A,R2
2B	1	ADD	A,R3
2C	1	ADD	A,R4
2D	1	ADD	A,R5
2E	1	ADD	A,R6
2F	1	ADD	A,R7
30	1	JNB	bitaddr,codeaddl
31	2	ACALL	codeaddr
32	1	RETI	
33	1	RLC	A
34	2	ADDC	A,#data
35	2	ADDC	A,dataaddr
36	1	ADDC	A,@RO
37	1	ADDC	A,@R1
36	1	ADDC	A,RO
39	1	ADDC	A,R1
3A	1	ADDC	A,R2

3B	1	ADDC	A,R3
3C	1	ADDC	A,R4
3D	1	ADDC	A,R5
3E	1	ADDC	A,R6
3F	1	ADDC	A,R7
40	2	JC	codeaddr
41	2	AJMP	codeaddr
42	2	ORL	dataaddr,A
43	3	ORL	dataaddr,#data
44	2	ORL	A,#data
45	2	ORL	A,dataaddr
46	1	ORL	A,@RO
47	1	ORL	A,@R1
46	1	ORL	A,RO
49	1	ORL	A,R1
4A	1	ORL	A,R2
4B	1	ORL	A,R3
4C	1	ORL	A,R4
4D	1	ORL	A,R5
4E	1	ORL	A,Re
4F	1	ORL	A,R7
50	2	JNC	codeaddr
51	2	ACALL	codeaddr
52	2	ANL	dataaddr,A
53	3	ANL	dataaddr,#data
54	2	ANL	A,#data

55	2	ANL	A,datsaddr
56	1	ANL	A,@RO
57	1	ANL	A,@R1
5e	1	ANL	A,RO
59	1	ANL	A,R1
5A	1	ANL	A,R2
5B	1	ANL	A,R3
5C	1	ANL	A,R4
5D	1	ANL	A,R5
5E	1	ANL	A,R6
5F	1	ANL	A,R7
60	2	JZ	codeaddr
61	2	AJMP	codeaddr
62	2	XRL	datesddr,A
63	3	XRL	datesddr,#data
64	2	XRL	A,#data
65	2	XRL	A,dataaddr
56	1	XRL	A,@RO
57	1	XRL	A,@R1
56	1	XRL	A,RO
59	1	XRL	A,RI
3A	1	XRL	A,R2
5B	1	XRL	A,R3
5C	1	XRL	A,R4
6D	1	XRL	A,R5
6E	1	XRL	A,R6

6F	1	XRL	A,R7
70	2	JNZ	codeaddr
71	2	ACALL	codeaddr
72	2	ORL	C,bitaddr
73	1	JMP	@A+DPTR
74	2	MOV	A,#data
75	3	MOV	dataaddr,#data
76	2	MOV	@RO,#data
77	2	MOV	@R1,#data
78	2	MOV	RO,#data
79	2	MOV	R1,#data
7A	2	MOV	R2,#data
7B	2	MOV	R3,#data
7C	2	MOV	R4,#data
7D	2	MOV	R5,#data
7E	2	MOV	R6,#data
7F	2	MOV	R7,#data
80	2	SJMP	codeaddr
81	2	AJMP	codeaddr
82	2	ANL	C,bitaddr
83	1	MOVC	A,@A+PC
84	1	DIV	AB
85	3	MOV	dataaddr,dataaddr
86	2	MOV	dataaddr,@RO
87	2	MOV	dataaddr,@R1
88	2	MOV	dataaddr,RO

89	2	MOV	dataaddr,R1
8A	2	MOV	dataaddr,R2
8B	2	MOV	dataaddr,R3
8C	2	MOV	dataaddr,R4
8D	2	MOV	dataaddr,R5
8E	2	MOV	dataaddr,R6
8F	2	MOV	dataaddr,R7
90	3	MOV	DPTR,#data
91	2	ACALL	codeaddr
92	2	MOV	bitsddr,C
93	1	MOVC	A,@A+DPTR
94	2	SUBB	A,#data
95	2	SUBB	A,dataaddr
96	1	SUBB	A,@RO
97	1	SUBB	A,@R1
98	1	SUBB	A,RO
99	1	SUBB	A,R1
9A	1	SUBB	A,R2
9B	1	SUBB	A,R3
9C	1	SUBB	A,R4
9D	1	SUBB	A,R5
9E	1	SUBB	A,R6
9F	1	SUBB	A,R7
AO	2	ORL	C,/bitaddr
A1	2	AJMP	codeaddr
A2	2	MOV	C,bitaddr

A3	1	INC	DPTR
A4	1	MUL	AB
A5		reservado	
A6	2	MOV	@RO,dataaddr
A7	2	MOV	@R1,dataaddr
A8	2	MOV	RO,data addr
A9	2	MOV	R1,dataaddr
AA	2	MOV	R2,dataaddr
AB	2	MOV	R3,dstaaddr
AC	2	MOV	R4,dataaddr
AD	2	MOV	R5,dataaddr
AE	2	MOV	R6,dataaddr
AF	2	MOV	R7,dataaddr
BO	2	ANL	C,/bitaddr
B1	2	ACALL	codeaddr
B2	2	CPL	bitaddr
B3	1	CPL	C
B4	3	CJNE	A,#data,codeaddr
B5	3	CJNE	A,dataaddr,code addr
B6	3	CJNE	@RO,#dats,codaad dr
B7	3	CJNE	@R1,#data,codead dr
B8	3	CJNE	RO,#data,codeaddr
B9	3	CJNE	R1,#datasodeaddr
BA	3	CJNE	R2,#data\$odeaddr

BB	3	CJNE	R3,#daQcodeaddr
BC	3	CJNE	R4,#dats@deaddr
BD	3	CJNE	R5,#data,codeaddr
BE	3	CJNE	R8,#data,codeaddr
BF	3	CJNE	R7,#data,codeaddr
C0	2	PUSH	dataaddr
C1	2	AJMP	codeaddr
C2	2	CLR	bitaddr
C3	1	CLR	C
C4	1	SWAP	A
C5	2	XCH	A,dataaddr
C8	1	XCH	A,@RO
C7	1	XCH	A,@R1
C8	1	XCH	A,RO
C9	1	XCH	A,R1
CA	1	XCH	A,R2
CB	1	XCH	A,R3
CC	1	XCH	A,R4
CD	1	XCH	A,R5
CE	1	XCH	A,R6
CF	1	XCH	A,R7
Do	2	POP	dataaddr
D1	2	ACALL	codaaddr
D2	2	SETB	biladdr
D3	1	SETB	C
D4	1	DA	A

D5	3	DJNZ	dataaddr,codeaddr
D6	1	XCHD	A,@RO
D7	1	XCHD	A,@R1
CM	2	DJNZ	RO,codeaddr
D9	2	DJNZ	R1,codeaddr
DA	2	DJNZ	R2,codeaddr
DB	2	DJNZ	R3,codeaddr
DC	2	DJNZ	R4,codeaddr
DD	2	DJNZ	R5,codeaddr
DE	2	DJNZ	R6,codeaddr
DF	2	DJNZ	R7,codeaddr
EO	1	MOVX	A,@DPTR
E1	2	AJMP	codeaddr
E2	1	MOVX	A,@RO
E3	1	MOVX	A,@R1
E4	1	CLR	A
E5	2	MOV	A,dataaddr
E6	1	MOV	A,@RO
E7	1	MOV	A,@R1
E8	1	MOV	A,R0
E9	1	MOV	A,R1
EA	1	MOV	A,R2
EB	1	MOV	A,R3
EC	1	MOV	A,R4
ED	1	MOV	A,R5
EE	1	MOV	A,R6

EF	1	MOV	A,R7
FO	1	MOVX	@DPTR,A
FI	2	ACALL	codeaddr
F2	1	MOVX	@RO,A
F3	1	MOVX	@RI,A
F4	1	CPL	A
F5	2	MOV	dataaddr,A
F6	1	MOV	@RO,A
F7	1	MOV	@R1,A
F8	1	MOV	RO,A
F9	1	MOV	RI,A
FA	1	MOV	R2,A
FB	1	MOV	R3,A
FC	1	MOV	R4,A
FD	1	MOV	R5,A
FE	1	MOV	R6,A
FF	1	MOV	R7,A

2.4.3.2 Ordenado por Mnemónico

<i>Opcode</i>	<i>Bytes</i>	<i>Mnemónico</i>	<i>Direccionamiento</i>
D1	2	ACALL	Codeaddr
11	2	ACALL	Codeaddr
31	2	ACALL	Codeaddr
51	2	ACALL	Codeaddr
71	2	ACALL	codeaddr

91	2	ACALL	codeaddr
B1	2	ACALL	codeaddr
FI	2	ACALL	codeaddr
24	2	ADD	A,#dats
27	1	ADD	A,@R1
26	1	ADD	A,@RO
25	2	ADD	A,datsaddr
23	1	ADD	A,R1
2A	1	ADD	A,R2
2B	1	ADD	A,R3
2C	1	ADD	A,R4
2D	1	ADD	A,R5
2E	1	ADD	A,R6
2F	1	ADD	A,R7
28	1	ADD	A,RO
34	2	ADDC	A,#data
37	1	ADDC	A,@R1
36	1	ADDC	A,@RO
35	2	ADDC	A,datsaddr
39	1	ADDC	A,R1
3A	1	ADDC	A,R2
3B	1	ADDC	A,R3
3C	1	ADDC	A,R4
3D	1	ADDC	A,R5
3E	1	ADDC	A,R6
3F	1	ADDC	A,R7

36	1	ADDC	A,RO
21	2	AJMP	Codeaddr
41	2	AJMP	codeaddr
61	2	AJMP	codeaddr
81	2	AJMP	codeaddr
A1	2	AJMP	codeaddr
C1	2	AJMP	codeaddr
E1	2	AJMP	codeaddr
1	2	AJMP	codesddr
54	2	ANL	A,#data
57	1	ANL	A,@R1
56	1	ANL	A,@RO
55	2	ANL	A,datsaddr
59	1	ANL	A,R1
5A	1	ANL	A,R2
5B	1	ANL	A,R3
5C	1	ANL	A,R4
5D	1	ANL	A,R5
5E	1	ANL	A,R6
5F	1	ANL	A,R7
5e	1	ANL	A,RO
BO	2	ANL	C,/bitaddr
82	2	ANL	C,bitaddr
53	3	ANL	dataaddr,#data
52	2	ANL	dataaddr,A
B7	3	CJNE	@R1,#data,codeaddr

			dr
B6	3	CJNE	@RO,#dats,codaa ddr
B4	3	CJNE	A,#data,codeaddr
B5	3	CJNE	A,dataaddr,code addr
BA	3	CJNE	R2,#data\$odeaddr
BB	3	CJNE	R3,#daQcodeaddr
BC	3	CJNE	R4,#dats@deaddr
BD	3	CJNE	R5,#data,codeadd r
BF	3	CJNE	R7,#data,codeadd r
BE	3	CJNE	R8,#data,codeadd r
B9	3	CJNE	R1,#datasodeaddr
B8	3	CJNE	RO,#data,codeadd r
C2	2	CLR	bitaddr
C3	1	CLR	C
E4	1	CLR	A
F4	1	CPL	A
B2	2	CPL	bitaddr
B3	1	CPL	C
D4	1	DA	A
17	1	DEC	@RI
16	1	DEC	@RO
14	1	DEC	A
15	2	DEC	dataaddr

1A	1	DEC	R2
1B	1	DEC	R3
1C	1	DEC	R4
1D	1	DEC	R5
1E	1	DEC	R6
1F	1	DEC	R7
19	1	DEC	RI
16	1	DEC	RO
84	1	DIV	AB
D5	3	DJNZ	dataaddr,codeaddr
DA	2	DJNZ	R2,codeaddr
DB	2	DJNZ	R3,codeaddr
DC	2	DJNZ	R4,codeaddr
DD	2	DJNZ	R5,codeaddr
DE	2	DJNZ	R6,codeaddr
DF	2	DJNZ	R7,codeaddr
D9	2	DJNZ	RI,codeaddr
CM	2	DJNZ	RO,codeaddr
7	1	INC	@RI
6	1	INC	@RO
4	1	INC	A
5	2	INC	dataaddr
A3	1	INC	DPTR
OA	1	INC	R2
OB	1	INC	R3
Oc	1	INC	R4

OD	1	INC	R5
OE	1	INC	R6
OF	1	INC	R7
Oe	1	INC	RI
6	1	INC	RO
20	3	JB	Bitaddr,codeaddr
10	3	JBC	bitaddr,codeaddr
40	2	JC	codeaddr
73	1	JMP	@A+DPTR
30	1	JNB	bitaddr,codeaddl
50	2	JNC	codeaddr
70	2	JNZ	codeaddr
60	2	JZ	codeaddr
12	3	LCALL	codeaddr
2	3	LJMP	codesddr
F7	1	MOV	@R1,A
77	2	MOV	@R1,#data
A7	2	MOV	@R1,dataaddr
76	2	MOV	@RO,#data
F6	1	MOV	@RO,A
A6	2	MOV	@RO,dataaddr
74	2	MOV	A,#data
E7	1	MOV	A,@R1
E6	1	MOV	A,@RO
E5	2	MOV	A,dataaddr
E9	1	MOV	A,R1

EA	1	MOV	A,R2
EB	1	MOV	A,R3
EC	1	MOV	A,R4
ED	1	MOV	A,R5
EE	i	MOV	A,R6
EF	1	MOV	A,R7
E8	1	MOV	A,RO
92	2	MOV	bitsddr,C
A2	2	MOV	C,bitaddr
87	2	MOV	dataaddr,@R1
86	2	MOV	dataaddr,@RO
F5	2	MOV	dataaddr,A
85	3	MOV	dataaddr,dataaddr
8A	2	MOV	dataaddr,R2
8B	2	MOV	dataaddr,R3
8C	2	MOV	dataaddr,R4
8D	2	MOV	dataaddr,R5
8E	2	MOV	dataaddr,R6
8F	2	MOV	dataaddr,R7
89	2	MOV	dataaddr,R1
66	2	MOV	dataaddr,RO
75	3	MOV	datsaddr,#data
90	3	MOV	DPTR,#data
7A	2	MOV	R2,#data
FA	1	MOV	R2,A
AA	2	MOV	R2,dataaddr

70	2	MOV	R3,#data
FB	1	MOV	R3,A
AB	2	MOV	R3,dstaaddr
7C	2	MOV	R4,#data
FC	1	MOV	R4,A
AC	2	MOV	R4,dataaddr
7D	2	MOV	R5,#data
FD	1	MOV	R5,A
AD	2	MOV	R5,dataaddr
7E	2	MOV	R6,#data
FE	1	MOV	R6,A
AE	2	MOV	R6,dataaddr
7F	2	MOV	R7,#data
FF	1	MOV	R7,A
AF	2	MOV	R7,dataaddr
F9	1	MOV	RI,A
79	2	MOV	RI,#data
A9	2	MOV	RI,dataaddr
78	2	MOV	RO,#data
F8	1	MOV	RO,A
A8	2	MOV	RO,data addr
93	1	MOVC	A,@A+DPTR
83	1	MOVC	A,@A+PC
FO	1	MOVX	@DPTR,A
F3	1	MOVX	@RI,A
F2	1	MOVX	@RO,A

E0	1	MOVX	A,@DPTR
E3	1	MOVX	A,@R1
E2	1	MOVX	A,@RO
A4	1	MUL	AB
0	1	NOP	
44	2	ORL	A,#data
47	1	ORL	A,@R1
46	1	ORL	A,@RO
45	2	ORL	A,dataaddr
49	1	ORL	A,R1
4A	1	ORL	A,R2
4B	1	ORL	A,R3
4C	1	ORL	A,R4
4D	1	ORL	A,R5
4F	1	ORL	A,R7
4E	1	ORL	A,Re
46	1	ORL	A,RO
AO	2	ORL	C,/bitaddr
72	2	ORL	C,bitaddr
43	3	ORL	dataaddr,#data
42	2	ORL	datsaddr,A
Do	2	POP	dataaddr
C0	2	PUSH	dataaddr
A5		reservado	
22	1	RET	
32	1	RETI	

23	1	RL	A
33	1	RLC	A
3	1	RR	A
13	1	RRC	A
D2	2	SETB	biladdr
D3	1	SETB	C
80	2	SJMP	codeaddr
94	2	SUBB	A,#data
97	1	SUBB	A,@R1
96	1	SUBB	A,@RO
95	2	SUBB	A,dataaddr
99	1	SUBB	A,R1
9A	1	SUBB	A,R2
9B	1	SUBB	A,R3
9C	1	SUBB	A,R4
9D	1	SUBB	A,R5
9E	1	SUBB	A,R6
9F	1	SUBB	A,R7
98	1	SUBB	A,RO
C4	1	SWAP	A
C7	1	XCH	A,@R1
C8	1	XCH	A,@RO
C5	2	XCH	A,dataaddr
C9	1	XCH	A,R1
CA	1	XCH	A,R2
CB	1	XCH	A,R3

CC	1	XCH	A,R4
CD	1	XCH	A,R5
CE	1	XCH	A,R6
CF	1	XCH	A,R7
C8	1	XCH	A,RO
D7	1	XCHD	A,@R1
D6	1	XCHD	A,@RO
64	2	XRL	A,#data
57	1	XRL	A,@R1
56	1	XRL	A,@RO
65	2	XRL	A,dataaddr
3A	1	XRL	A,R2
5B	1	XRL	A,R3
5C	1	XRL	A,R4
6D	1	XRL	A,R5
6E	1	XRL	A,R6
6F	1	XRL	A,R7
59	1	XRL	A,RI
56	1	XRL	A,RO
63	3	XRL	datesddr,#data
62	2	XRL	datesddr,A

2.5 PROGRAMACIÓN

Como ya se ha dicho, el ensamblador posee una parte común y otra específica. La parte común está formada por los siguientes ficheros:

- Asm.h

- Asmain.c
- Aslex.c
- Assym.c
- Assubr.c
- Asexpr.c
- Asdata.c
- Aslist.c
- Asout.c

La parte específica se compone de los siguientes:

- 8051.h – Fichero de cabecera que contiene las definiciones de constantes, variables, estructuras y tipos.
- 8051ext.c – Extensiones de ficheros, orden de los bytes y descripción del dispositivo.
- 8051pst.c – Tabla con las directivas del ensamblador y los mnemónicos con sus correspondientes opcodes.
- 8051mch.c – Código para procesar los opcodes, directivas específicas y modos de direccionamiento.

Veamos estos ficheros en detalle:

2.5.1 8051pst

Este fichero describe las directivas del sistema asociándolas a su opcode. Para ello hace uso de la estructura mne definida en el fichero asm.h con los siguientes campos:

- Puntero a otra estructura tipo mne: No utilizado, debe tener valor NULL.
- Mnemónico: Es una cadena que define cómo debe ser escrito el mnemónico o la directiva del compilador.
- Tipo de mnemónico: Utilizado para distinguir distintos tipos de procesamiento del mnemónico. En general cada mnemónico tiene un procesamiento diferente, pero los hay suficientemente similares como para necesitar una sólo rutina de procesamiento.

- Banderas aplicables al mnemónico: Cada mnemónico o directiva se almacena en un array de estructuras mne. El programa a la hora de decidir si lo que encuentra es un mnemónico o directiva válida va comparando los diferentes elementos del array. Este valor es siempre 0 excepto en el último elemento de la estructura que toma el valor S_END y que le dice al programa que el array termina en ese punto.
- Valor del mnemónico: En el caso de que sea mnemónico almacena el opcode correspondiente (la raíz, luego se modificarán o añadirán valores en el procesamiento).

2.5.2 8051ext.c

Define la CPU como “Intel 8051/8052”

Ordene byte: Primero almaceno el más significativo.

Extensión: “asm”

2.5.3 8051.h

Se asignan constantes y definen estructuras:

Las constantes definidas como direccionamiento almacenan la dirección en memoria del registro o bit referenciado.

A las definidas como registros se les asigna un valor que se usará para calcular el opcode.

El resto se utilizan como etiquetas significativas para evitar usar números.

Se define la estructura adsym, encargada de almacenar los diferentes modos de direccionamiento con los siguientes campos:

- A_str: Almacena el modo de direccionamiento tal y como deberá ser escrito en el mnemónico.
- A_val: Valor asociado al modo de direccionamiento. El valor debe ser valor estratégico para facilitar el cálculo posterior del opcode, aunque no tiene por qué ser así.

2.5.4 8051adr.c

En este archivo se resuelven los modos de direccionamiento.

Aquí se crean las diferentes estructuras de tipo asdym definidas en 8051.h, una por cada tipo de direccionamiento: Directo a registro, indirecto, directo al Acumulador, directo a

bit C, directo a par AB(en multiplicación y división),directo a DPTR, indexados con DPTR y PC, indirecto a memoria externa (@DPTR) directo a bit y directo a SFR.

En realidad los dos últimos no son modos de direccionamiento, ya que en las instrucciones los bit y los SFR se referencian mediante un número, sin embargo se crean para dar cabida a símbolos del tipo P0.0 o TCON y que el programador no tenga que calcular su valor.

El procesamiento se realiza mediante 4 funciones.

Addr(esp)

esp es una estructura de tipo expr definida en asm.h. De los campos que tiene nos interesan dos:

E_mode: Almacena el modo de direccionamiento

E_addr: Almacena la dirección

Esta función extrae el modo de direccionamiento del siguiente término del mnemónico y el valor a_val correspondiente a ese modo. El primero se almacena en el campo e_mode y el segundo en e_addr.

Cuando el direccionamiento es un texto (R1,@DPTR, TCON, ...) el almacenaje se realiza con la ayuda de la función adsym.

Si el modo incluye un número se extrae su valor y se almacena en el campo e_addr con la función expr definida en asm.h. El campo e_mode se rellena manualmente. Estos modos son los que comienzan por #, / o los que no comienzan por esos caracteres y tampoco coinciden con los modos de direccionamiento devueltos por adsym (directo a memoria por ejemplo). En cualquier caso se tiene en cuenta que este número pudiera ser un símbolo llamando a adsym para comprobarlo.

En esta función se detectan también los errores de direccionamiento.

Admode(sp)

Llama a la función srch para extraer el siguiente término del mnemónico. A continuación compara este término con los elementos de la estructura de tipo adsym apuntada por sp. Si coincide con alguno se devuelve el valor a_val correspondiente a dicho elemento de la estructura.

Srch(str)

Utilizando la función any comprueba si en el mnemónico que se está procesando esta presente la cadena str. Aquí es donde se tiene en cuenta si la codificación es sensible a las mayúsculas. En este caso los mnemónicos deben estar en mayúsculas.

Any (c,str)

Devuelve si la cadena str contiene la letra c.

2.5.5 8051mch.c

Esta función realiza el procesamiento de las instrucciones para calcular el opcode completo.

Utiliza las siguientes funciones:

Machine(mp)

Esta es la rutina que procesa el mnemónico. Mp es una estructura de tipo mne definida en 8051pst.h.

El programa ya ha determinado si es directiva o mnemónico. Si es mnemónico, llama esta rutina para que le procesen y se lo pasa en una estructura mne con los campos correspondientes.

Dado que en el campo m_type se identificaba el tipo de procesamiento que se le debe dar, basta una estructura case para dirigir el programa al código adecuado. En caso de que el tipo no sea conocido se produce un error.

Esta rutina utiliza dos funciones importantes: Outab y Outrw que se describe más adelante.

En cada segmento case se utiliza la función addr para extraer los diferentes términos del mnemónico, la función comma para extraer los separadores. Con estructuras if se actúa según el modo de direccionamiento y con la ayuda del valor devuelto en el campo e_addr de la estructura expr se calcula el opcode que se enviará al fichero objeto.

Outab(op)

Envía al fichero objeto el valor op como absoluto .

Outrw(e1,0)

Envía al fichero objeto el valor e1->e_adrr como reubicable, siendo e1 una estructura tipo expr. Es de notar que sólo los saltos y llamadas no absolutas deben utilizar esta función. Los saltos relativos a pc deben ser calculados y enviados como byte absoluto. Para calcular este byte se usa la variable dot->s_adr, definida en asm.h y usada para referirse al contenido del contador de programa.

Comma()

Extrae una coma del mnemónico(separador de términos), dando un error de sintaxis si no la encuentra.

Con estos cuatro ficheros queda completamente definido el ensamblador. Es de destacar la dificultad encontrada para descifrar el significado de las funciones a utilizar así como el flujo del programa, debido a la nula información encontrada al respecto, teniendo que realizar esta tarea a base de leer e interpretar el código fuente y realizando diversas pruebas para averiguar el comportamiento de la rutinas.

3 SIMULADOR

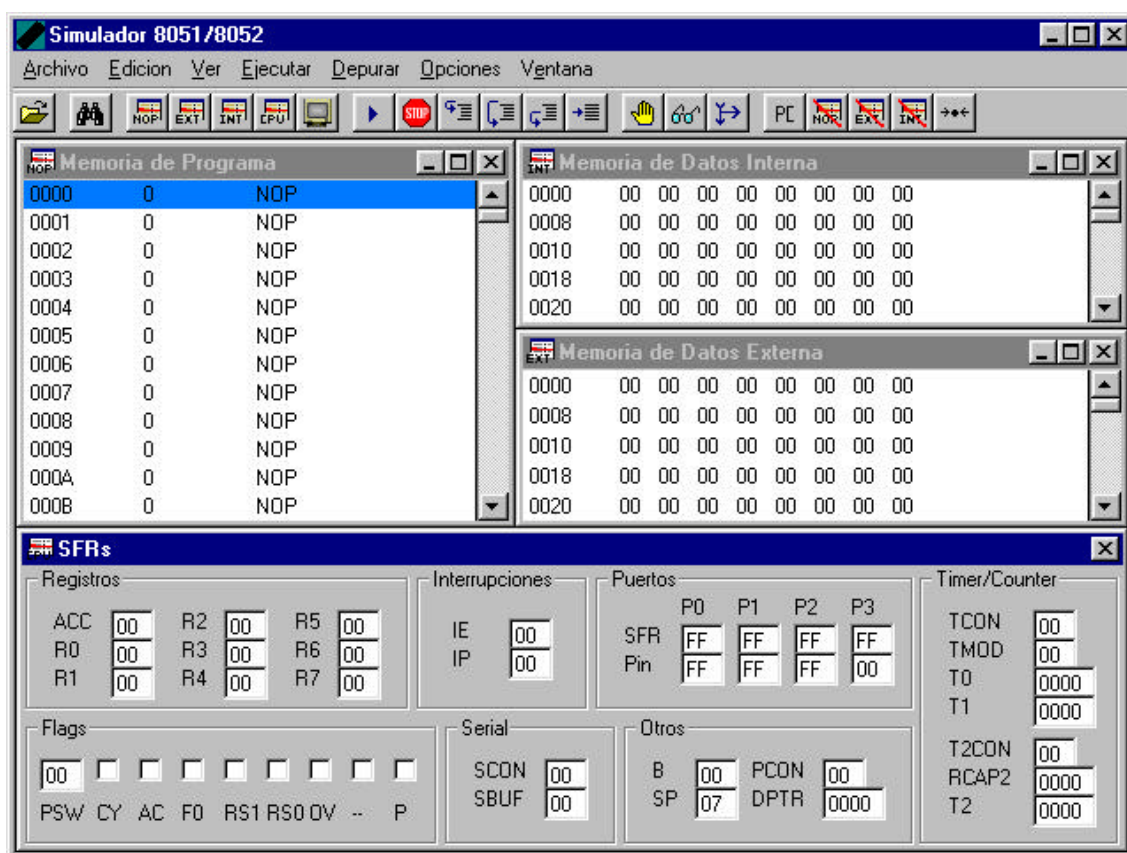
3.1 MANUAL DE USUARIO

3.1.1 Ejecución del Programa

Para comenzar a ejecutar el programa basta con pinchar en el icono del menú Inicio correspondiente a esta aplicación. Inmediatamente se muestra una pantalla de presentación mientras se cargan la aplicación que se cerrará dando paso a la ventana principal de la aplicación.

3.1.2 Ventana Principal

El simulador nada más arrancarse presenta la siguiente ventana:



En ella se pueden distinguir varias partes que describiremos a continuación.

3.1.1.1 Barra de Menús

Será descrita en la siguiente sección por lo que no se comenta más en este punto.

3.1.1.2 Memoria de Programa

En esta ventana se puede ver el programa que contiene dicha memoria.

Posee tres columnas, la primera de ellas indica la dirección de memoria que se está visualizando, la segunda el código en hexadecimal del opcode que hay en dicha posición de memoria y la tercera en la que se presenta el mnemónico correspondiente.

Esta ventana se puede manejar con la barra de desplazamiento o con las flechas de cursor para mostrar las sucesivas posiciones de la memoria.

Para ir a una posición en concreto, basta con pinchar dos veces en la primera columna o pulsar Enter y se abrirá un cuadro de diálogo donde se debe introducir la dirección a visualizar en formato hexadecimal.

Se utiliza un código de colores para resaltar determinadas zonas de memoria:

Blanco	Sin resaltar.
Azul Oscuro	Cursor.
Azul Claro	Posición a la que apunta el contador de programa.
Rojo	Breakpoint en esa posición.
Rosa	Breakpoint en n posiciones a partir de esta.

3.1.1.3 Memoria Interna y Externa

Muestra el contenido de la memoria de datos interna o externa del microcontrolador.

Esta ventana se puede manejar con la barra de desplazamiento o con las flechas de cursor para mostrar las sucesivas posiciones de la memoria.

Para ir a una posición en concreto basta con pinchar dos veces en la primera columna o situar el cursor en dicha columna y pulsar Enter. Hecho esto se abrirá un cuadro de diálogo donde se introducirá la dirección a visualizar en formato hexadecimal.

Para cambiar el contenido de una celda de memoria hay que situar en esa celda el cursor y pulsar Enter o bien pinchar en ella dos veces. Aparecerá un cuadro de diálogo donde se introducirá el dato a almacenar en hexadecimal. En caso de introducir más de dos

caracteres, el resto será tomado como una nueva entrada para cambiar la posición de memoria inmediatamente posterior a la seleccionada, de este modo se pueden cambiar varias posiciones de una sola vez.

El cambio del contenido se puede realizar también en modo texto seleccionando la última columna. En este caso El texto introducido en el cuadro de diálogo correspondiente se divide carácter a carácter y se convierte su valor a ASCII, que será el que se almacene en la posición de memoria correspondiente. Se pueden variar varias posiciones de memoria de una vez.

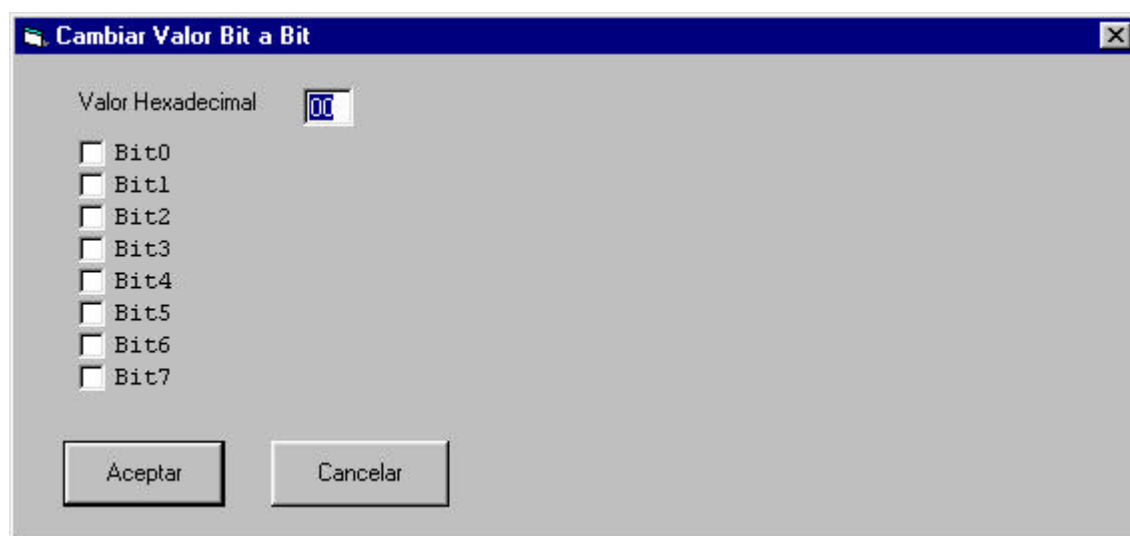
3.1.1.4 Ventana de Special Frame Registers

Muestran el contenido de los registros especiales y están agrupados del siguiente modo:

REGISTROS DE PROPÓSITO GENERAL:

Incluyen el Acumulador y los registros R0-R7. El banco de registros que se muestra depende de los flags RS0 - RS1 del PSW.

Para modificar su contenido basta con pinchar en el texto correspondiente, abriéndose la siguiente ventana:



En ella se puede introducir el dato de forma hexadecimal o bien bit a bit. Esta ventana es compartida por la mayoría de los SFR's, variando únicamente la descripción del significado de los bits.

FLAGS

Muestra el contenido del Program Status Word y desglosa su valor en los siguientes flags.

C	Carry. Usado en aritmética sin signo y operaciones lógicas.
AC	Acarreo del tercer al cuarto bit del resultado en operaciones aritméticas.
F0	Registro a disposición del programador.
RS1 y RS0	Seleccionan el banco de registros R0-R7 que se están utilizando.
OV	Flag de Overflow, usado en aritmética con signo.
-	No usado.
P	Bit de paridad. Varía según el número de unos del Acumulador.

El valor del PSW se puede cambiar escribiendo directamente en la ventana de texto o bien variando los flags.

INTERRUPCIONES

- IE Registro habilitador de interrupciones. Habilita interrupciones externas y de los temporizadores 0 y 1 y contiene los flags que indican que se ha producido una interrupción.
- IP Prioridad de las interrupciones. Utilizado para determinar qué interrupción se sirve primero en los casos en que se producen dos interrupciones a la vez.

COMUNICACIÓN SERIE

Agrupar los registros que afectan al puerto serie:

- SCON Determina el funcionamiento del puerto serie.
- SBUF Registro de recepción-emisión del puerto serie.

PUERTOS

Son los puertos de entrada/salida del procesador. Se hace distinción entre SFR y Pin porque no tienen necesariamente el mismo valor y hay instrucciones que lee uno u otro. En general todo lo que escriba en el registro se refleja en el pin. Lo que se escriba en el pin se considera entrada si en el bit correspondiente del registro hay un 1.

En el caso del puerto 3 (y el 1 en el 8052) los pines sirven como entradas para diferentes eventos, como petición de interrupción externa, y disparo de los contadores/temporizadores.

TEMPORIZADORES/CONTADORES

TCON	Control de habilitación e interrupciones de los temporizador/contador 0 y 1.
TMOD	Control de modo de funcionamiento de los temporizador/contador 0 y 1.
T0 y T1	Valor de los contadores.
T2CON	Control de habilitación, modo e interrupciones del temporizador/contador 2. (Sólo en 8052).
RCAP2	Precarga o buffer de captura del temporizador/contador 2. (Sólo en 8052).
T2	Valor del temporizador/contador 2. (Sólo en 8052).

OTROS

B	Registro auxiliar.
SP	Puntero de la pila.
PCON	Control de los modos de ahorro de energía del microcontrolador. Estos modos no afectan a la simulación.
DPTR	Usado para trabajar con accesos externos de 16 bit.

Todos los valores se pueden cambiar pinchando sobre la casilla correspondiente.

3.1.2 Barra de Herramientas

Todo lo que se hace a través de esta barra se puede realizar a través de menús, por lo que su explicación detallada se realizará en el siguiente apartado.



Abrir Fichero.



Nueva Búsqueda en Memoria.



















Muestra la Ventana de Código.



Nueva Ventana de Memoria Externa.



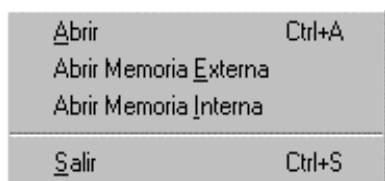
Nueva Ventana de Memoria Interna.

	Muestra la Ventana de SFR's.
	Muestra la Ventana del Terminal.
	Ejecutar.
	Detiene la Ejecución.
	Paso de Ejecución.
	Paso de Ejecución sin entrar en Subrutina.
	Ejecutar hasta Salir de Subrutina.
	Ir A.
	Breakpoint en Cursor.
	Añade Variable.
	Generar Interrupción.
	Modifica el valor del PC.
	Borra Memoria de Código.
	Borra Memoria Externa.
	Borra Memoria Interna.
	Reset.

3.1.3 Barra de Menú

La Barra de Menú tiene siete submenús con el siguiente contenido:

3.1.3.1 Menú Archivo



ABRIR

Abre un nuevo fichero de código. Los formatos admitidos son el estándar Intel (ihx) y el estándar Motorola (s19). Cuando se carga el programa se borra la memoria de código anterior.

ABRIR EN MEMORIA INTERNA

Abre un fichero en formato Intel o Motorola pero el contenido lo almacena en la memoria interna. Antes de cargar la memoria se borra el contenido anterior.

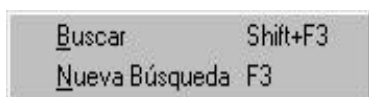
ABRIR EN MEMORIA EXTERNA

Abre un fichero en formato Intel o Motorola pero el contenido lo almacena en la memoria externa. Antes de cargar la memoria se borra el contenido anterior.

SALIR

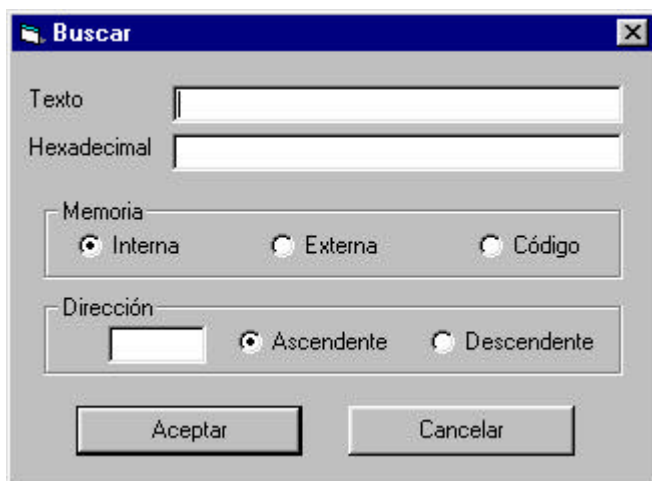
Cierra el simulador.

3.1.3.2 Menú Edición



BUSCAR

Busca una cadena en la memoria. Al pinchar aparece el siguiente cuadro de diálogo:



La cadena a buscar se puede introducir como texto ASCII o bien como hexadecimal. Cualquier cambio que se realice en una de las casillas repercutirá en la otra. Cuando se escribe en la casilla hexadecimal, el número de dígitos ha de ser impar, de lo contrario el último dígito se ignora.

Con el recuadro Memoria se indica el tipo de memoria en la que quiero buscar.

En el recuadro dirección se puede rellenar la casilla de texto con la dirección de inicio de la búsqueda. Si se deja en blanco se entiende que es cero. Se puede especificar además el sentido de la búsqueda seleccionando uno de los dos cuadros de opción.

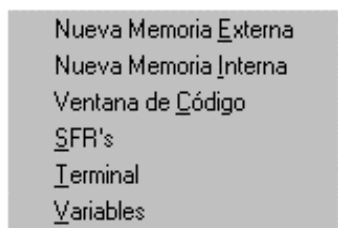
Si se encuentra la cadena buscada, sale un cuadro de diálogo diciendo la dirección donde se encontró. Además, si la búsqueda es en la memoria cuya ventana teníamos seleccionada en el momento de pinchar en el menú, se lleva el cursor a la dirección donde se encontró la cadena. Si la ventana seleccionada no coincide con la memoria en la que se busca, se abrirá una nueva ventana de memoria (sólo si es de datos) y se resaltará la dirección igualmente.

Cuando la búsqueda llega al final (o principio) de la memoria sin encontrar la cadena buscada se emite un mensaje informando de ello.

NUEVA BÚSQUEDA

Se repite la búsqueda de la cadena anteriormente introducida en el menú Buscar, en el mismo sentido pero a partir de la dirección de la última cadena encontrada.

3.1.3.3 Menú Ver



NUEVA MEMORIA EXTERNA

Abre una nueva ventana de memoria de datos externa. El máximo número de ventanas abiertas al mismo tiempo es de 10.

NUEVA MEMORIA INTERNA

Abre una nueva ventana de memoria de datos interna. El máximo número de ventanas abiertas al mismo tiempo es de 10.

VENTANA DE CÓDIGO

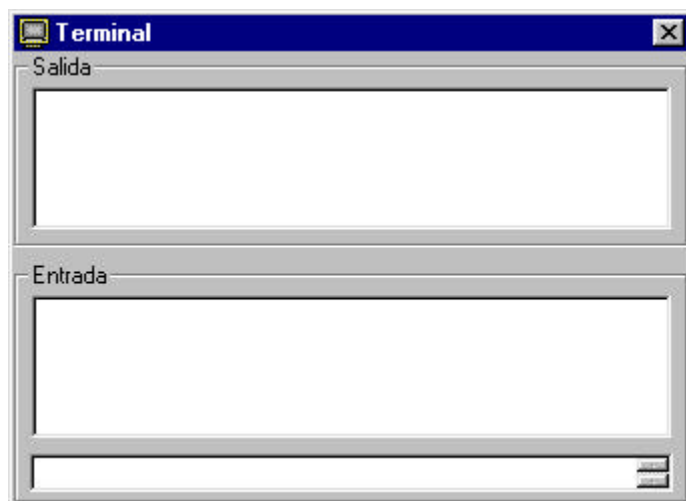
Abre la ventana de memoria de código en caso de que estuviera cerrada. En caso contrario no hace nada.

SFR's

Abre la ventana de SFR's en caso de que estuviera cerrada. En caso contrario no hace nada.

TERMINAL

Muestra la ventana del terminal utilizado para ayudar a la simulación del puerto serie.



En esta ventana se distinguen dos partes. La primera de ellas es la marcada con la palabra salida. En este recuadro se imprimen los valores que salen por el puerto serie una vez que se han traducido a caracteres ASCII.

En la segunda parte se produce la entrada del puerto serie. La escritura se realiza en el recuadro inferior y a medida que el microcontrolador recoge los caracteres, se van imprimiendo en el recuadro superior.

En los recuadros de entrada y salida el usuario puede introducir texto o avances de línea, pero sólo servirán para ordenar los caracteres allí mostrados y nunca afectarán el comportamiento del puerto serie.

VARIABLES

Muestra la ventana de variables definidas que será descrita posteriormente.

3.1.3.4 Menú Ejecutar

Ejecutar	F5
Parar	Shift+F5
Paso	F8
Paso saltando	Shift+F8
Ir a cursor	Ctrl+F8
Salir de Subrutina	Shift+Ctrl+F8

EJECUTAR

Comienza la ejecución del programa simulado a partir de la posición del Contador de Programa. Las vistas no son actualizadas durante la ejecución a menos que se

especifique en el menú Depurar. Se añade a la barra de título la palabra 'Ejecutando' para denotar este estado.

PARAR

Detiene la ejecución y actualiza las ventanas con el nuevo estado del microcontrolador.

PASO

Avanza la ejecución del programa en una instrucción.

PASO SALTANDO

Igual que el anterior pero si encuentra una instrucción de tipo CALL sigue ejecutando código hasta encontrar una instrucción RET.

IR A CURSOR:

Ejecuta instrucciones hasta que el PC apunte a la dirección donde está situado el cursor en la ventana de código.

SALIR DE SUBROUTINA

Ejecuta instrucciones hasta que encuentra una instrucción RET o RETI.

En general, una vez que el simulador alcanza el final de la memoria se emite un mensaje y se continúa la ejecución por la dirección 0H.

3.1.3.5 Menú Depurar

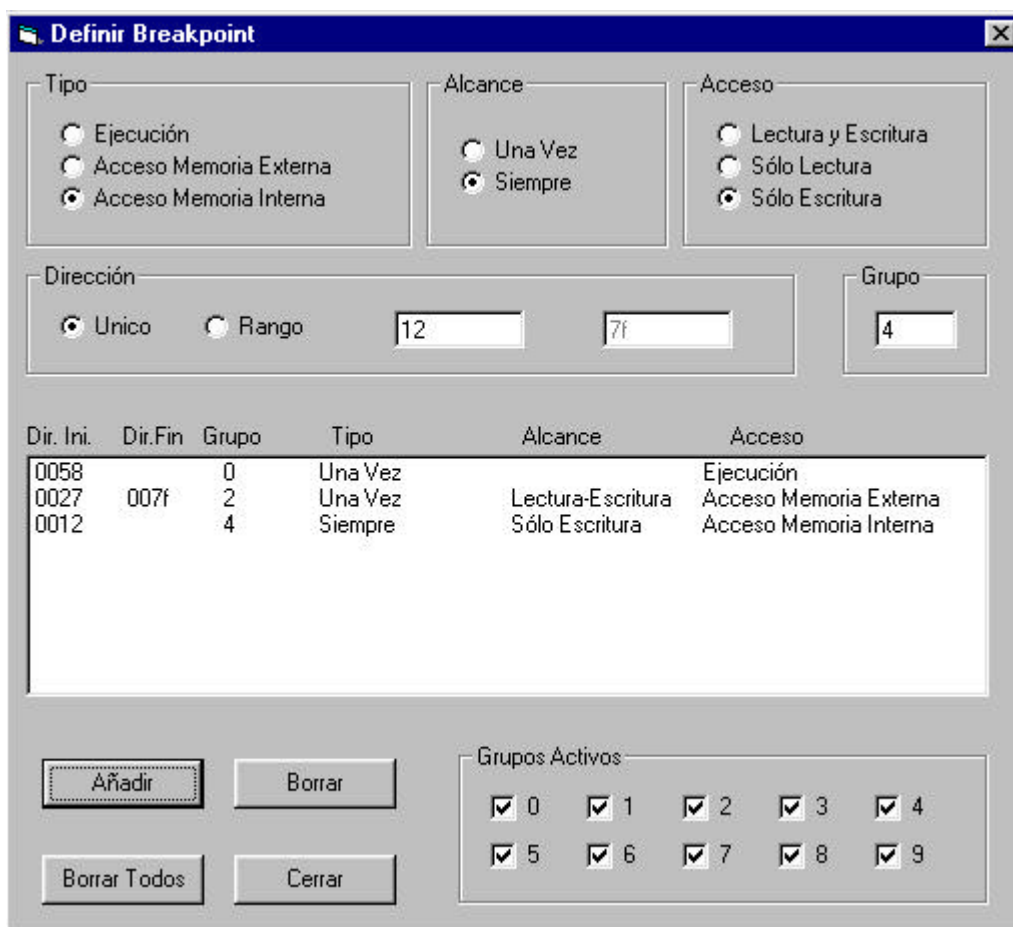
Modificar PC	
Definir Breakpoint	Shift+F11
Breakpoint en Cursor	F11
Añadir Variable	F12
Generar Interrupción	

MODIFICAR PC

Modifica el valor del contador de programa. El valor introducido debe ser de una posición de memoria existente y apuntar al inicio de una instrucción, de lo contrario saldrá un mensaje de error.

DEFINIR BREAKPOINT

Abre la ventana de gestión de breakpoints.



El uso de esta ventana es como sigue:

- Tipo: Selecciona si es un breakpoint de ejecución o al acceder a memoria externa o interna.
- Alcance: Si es ‘una vez’, el breakpoint se borrará una vez alcanzado. Si es ‘siempre’ el breakpoint permanece hasta que se borre manualmente.
- Acceso: Se escoje si parará por una lectura, por una escritura o por ambas. Sólo está activo si se define como breakpoint de acceso.

- Dirección:** Si es de tipo 'único' sólo habrá una casilla de texto activa donde se pondrá la dirección en la que estará el breakpoint. Si es de tipo 'rango' se especificará dirección de inicio y dirección final. Cualquier acceso (o ejecución) en ese rango de direcciones detendrá la ejecución.
- Ventana resumen:** Muestra todos los breakpoints definidos y sus características.
- Grupo:** Indica el grupo al que va a pertenecer el breakpoint. Un grupo es un conjunto de breakpoints que pueden ser activados o desactivados de forma conjunta. Si el grupo está inactivo no se detendrá la ejecución aunque se alcance el breakpoint. El número de grupos es 10, siendo el grupo 0 el que se toma por defecto.
- Añadir:** Añade a la lista el breakpoint que hemos definido anteriormente.
- Borrar:** Borra el breakpoint que hayamos seleccionado antes en la ventana resumen.
- Borrar todos:** Borra todos los breakpoint de una vez.
- Cerrar:** Cierra la ventana.
- Grupos activos:** Define qué grupos se tendrán en cuenta a la hora de detener la ejecución. La casilla marcada significa grupo activo.

BREAKPOINT EN CURSOR

Crea un breakpoint en la posición de memoria de código donde se encuentra el cursor. Es un breakpoint único en ejecución, de alcance permanente y perteneciente al grupo 0.

Si en la posición del cursor ya había marcado un breakpoint **de dirección única**, lo borra de la lista.

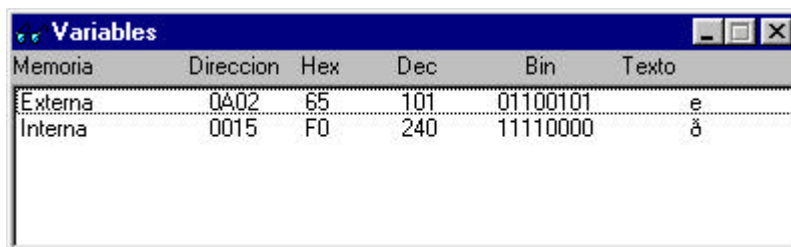
AÑADIR VARIABLE

Incluye una nueva posición de memoria a visualizar en la ventana de variables. El cuadro de diálogo mostrado es el siguiente:



La variable se puede definir en la memoria interna o la externa.

Al definir la primera variable se abre la ventana donde se muestran las variables:

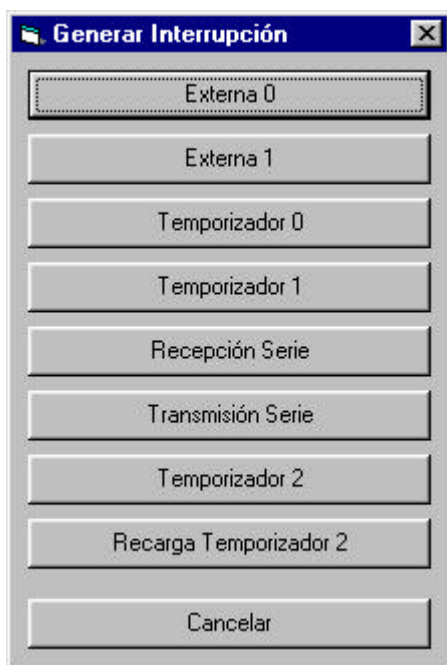


Memoria	Direccion	Hex	Dec	Bin	Texto
Externa	0402	65	101	01100101	e
Interna	0015	F0	240	11110000	ð

En ella se distinguen varias columnas. La primera de ellas indica la posición de memoria que se está visualizando. La segunda el tipo de memoria de que se trata. El resto visualiza el dato en formato hexadecimal, decimal, ASCII y binario.

Para borrar una variable, basta con seleccionar la línea correspondiente en la ventana de variables y pulsar suprimir o retroceso.

GENERAR INTERRUPCIÓN



Desde esta ventana se puede provocar una interrupción de forma artificial. Las referentes al temporizador 2 no aparecerán si se trata de un 8051. Se tienen en cuenta la

habilitación de interrupciones y prioridades entre estas y demás requisitos para que se atienda la interrupción.

3.1.3.6 Menú Opciones

Cambiar Fuente	
Definir Hardware	Ctrl+H
Refrescar Continualmente	
Simular Temporizadores	
Simular Puerto Serie	
Reset	Ctrl+R
Borrar Memoria Interna	Ctrl+I
Borrar Memoria Externa	Ctrl+E
Borrar Código	Ctrl+C

CAMBIAR FUENTE

Permite cambiar el tipo de letra que se usará en las ventanas de memoria de código y datos . No afecta a las ventanas de SFR's ni otras.

REFRESCAR CONTÍNUAMENTE

Puede seleccionarse o no y permite que los cambios que se produzcan en la memoria y los SFR's durante la ejecución de un programa sean mostrados a cada instrucción o bien que se oculten hasta finalizar la simulación, de modo que aumente la velocidad de ejecución.

SIMULAR TEMPORIZADORES

Es una opción seleccionable y provoca que el simulador ignore la existencia de los temporizadores o no. El no simularlos acelera la ejecución.

SIMULAR PUERTO SERIE

Al igual que la anterior permite acelerar la simulación ignorando el puerto serie.

RESET

Reset del procesador. Los SFR's toman sus valores iniciales, la ejecución se detiene y las interrupciones pendientes dejan de servirse. El contenido de las memorias se deja como está, así como los breakpoints y las variables definidas.

BORRAR MEMORIA INTERNA

Pone el contenido de la memoria interna a cero.

BORRAR MEMORIA EXTERNA

Pone el contenido de la memoria externa a cero.

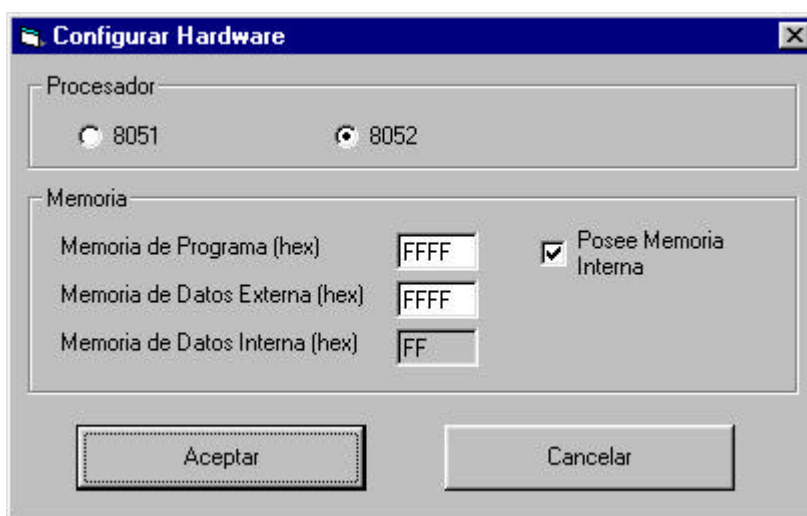
BORRAR CÓDIGO

Pone el contenido de la memoria de programa a cero (Instrucciones NOP en todas las posiciones de memoria).

DEFINIR HARDWARE

Con este cuadro de diálogo podemos definir el tipo de hardware que tenemos instalado.

Configura el tipo de dispositivo que estamos usando:

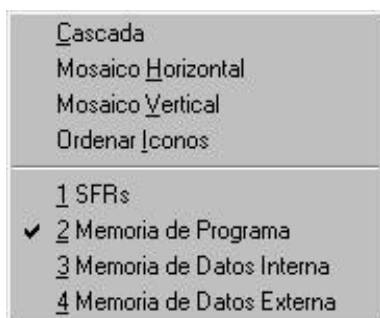


En esta ventana se puede seleccionar el tipo de procesador que será simulado, así como el tamaño de la memoria de código y de datos externa. La memoria de datos interna es fija y depende del modelo de microcontrolador instalado. En el caso del 8051 esta memoria estará fijada a 7Fh y con el 8052 a FFh..

La casilla 'Posee Memoria Interna' hace referencia a que el controlador tiene memoria de datos incluida.

Cuando se modifica el hardware se hace un reinicio del procesador: Se borran todas las memorias, se hace un reset al microcontrolador y todos los breakpoints y las variables son eliminadas.

3.1.3.7.- Menú Ventana



En esta ventana se encuentran las opciones más comunes para colocar las ventanas, además de un listado con las que se encuentran abiertas, pudiéndolas traer a primer plano pinchando en la línea correspondiente.

3.2 PROGRAMACIÓN

3.2.1 Variables globales utilizadas

Public Type Instru

Direccion As Long

Opcode As Long

Mnemonic As String

End Type

La siguiente estructura almacena diversos datos de las instrucciones máquina. Posteriormente se definirá un array de variables de este tipo, una por cada posición de la memoria de código. El subíndice de dicho array identificará la posición de memoria a la que se refieren estos datos almacenados. Cada posición de memoria tendrá asociada una estructura de este tipo incluso aunque la memoria no esté definida como existente. En aquellas posiciones de memoria que no comience una instrucción los valores que almacena no son significativos.

- **Direccion:** Indica la dirección de comienzo de la instrucción. En las direcciones donde comienza una instrucción coincide con el valor del subíndice del array de estructuras antes mencionado. Si en la posición de memoria indicada por el subíndice no comienza la instrucción se le asigna el valor -1.
- **Opcode:** Almacena el opcode de la instrucción. Si la posición de memoria a la que hace referencia es comienzo de instrucción se almacena el opcode completo. Si no

es comienzo de instrucción se almacena el byte del opcode correspondiente a esa posición de memoria.

- Mnemonic: Es una cadena de texto representando el mnemónico. Si la posición de memoria es comienzo de instrucción se almacena el mnemónico. Si no es comienzo de instrucción no se almacena ningún valor.

Public Instruccion(0 To 65535) As Instru

Este es el array que almacena las estructuras Instru. La dimensión es fija independientemente del tamaño de la memoria definida.

Public Type Variable

Direccion As Long

Tipo As Integer

End Type

Esta estructura almacena los datos de las variables que definirá el usuario en tiempo de ejecución. Del mismo modo que con la estructura anterior, se crea un array de variables de este tipo para cada variable que se definan.

- Direccion: Almacena la dirección donde se ha definido la variable.
- Tipo: Almacena el tipo de memoria en la que se define la variable. Si tiene el valor 0 indica que se refiere a memoria interna. Si es 1 se refiere a memoria externa.

Public Var() As Variable

Este es un array de estructuras tipo variable que almacena los datos necesarios de las variables que se definan en la simulación. Al iniciarse el programa tiene dimensión cero e irá creciendo o disminuyendo a medida que se creen o borren variables.

Public Type Break

DireccionIni As Long

DireccionFin As Long

Tipo As Integer

Alcance As Integer

Grupo As Integer

Rango As Integer

Acceso As Integer

End Type

Esta estructura almacena los datos requeridos para el uso de breakpoints. Se crea una estructura de este tipo por cada breakpoint que defina el usuario al realizar la simulación.

- **DireccionIni:** Indica la dirección donde se ha definido el breakpoint o bien la dirección de inicio si es un breakpoint que afecta a un grupo de direcciones.
- **DireccionFin:** Sólo es significativo si el breakpoint se refiere a un grupo de direcciones, en cuyo caso almacena la dirección final del grupo de direcciones a tener en cuenta. En caso contrario este campo no tiene asignado ningún valor.
- **Tipo:** Indica si el breakpoint se define en la ejecución del programa, en cuyo caso las direcciones anteriores se referirán a la memoria de código, o bien si es en un acceso a la memoria, haciendo referencia las direcciones anteriores a las memorias interna o externa. Si el valor almacenado es 0 el breakpoint se define en ejecución. Si es 1 se define en acceso a memoria externa y si es 2 se refiere a un acceso a memoria interna.
- **Alcance:** Especifica si una vez que el breakpoint se ha alcanzado, este se borra o permanece definido. Si el valor es 0 el breakpoint se tendrá en cuenta una sola vez. Si es 1 el breakpoint permanecerá hasta que se borre manualmente.
- **Grupo:** Indica el grupo al que pertenece el breakpoint para poder ser activado o desactivado a la vez con otros de su mismo grupo.
- **Rango:** Si el valor es 0 el breakpoint se define únicamente para la dirección en cuestión. Si es 1 el breakpoint se activará en cualquiera de las direcciones entre DireccionIni y DireccionFin.
- **Acceso:** Este valor sólo es significativo si el breakpoint se define para acceso a memoria externa o interna. En el caso de valer 0 el breakpoint se activa si hay lectura o escritura en la dirección definida. Si es 1 se activa si hay una lectura y si es 2 si hay una escritura.

Public Breakpoints() As Break

Este es un array de estructuras de tipo Break. La dimensión es variable y se modifica a medida que se crean o destruyen breakpoints.

Public Type SfrBits

Bit0 As String

Bit1 As String

Bit2 As String

Bit3 As String

Bit4 As String

Bit5 As String

Bit6 As String

Bit7 As String

Hex As String

End Type

Al cambiar el valor de los SFR's aparece un texto descriptivo de qué significa cada bit. En esta estructura se almacenan esos textos, uno por bit.

Public SfrBit As SfrBits

Esta es la variable que hace referencia a la estructura anterior. Se pasa al formulario SFRBitsForm con el texto que debe presentar.

Public Ejecutando As Boolean

Indica que el simulador está ejecutando un programa de forma continua. Se emplea para permitir o no el refresco de ciertas ventanas de modo que se acelere la simulación.

Public InstruccionEjecutada As Long

Almacena el primer byte del opcode de la última instrucción ejecutada. Con este valor ya se puede identificar de qué tipo de instrucción se trata y se utiliza para detectar la instrucción RETI en el tratamiento de las interrupciones, ya que estas no se sirven si esa fue la instrucción que se ejecutó en último lugar.

Public SFRMem(0 To 255) As Long

Este array almacena los valores de la memoria de SFR's

Public Memex(0 To 10) As ExtMemForm

Public Memin(0 To 10) As IntMemForm

Public Memcod As CodeMemForm

Estas tres últimas variables almacenan objetos de los tipos especificados. En realidad son punteros a las ventanas que muestran los contenidos de las memorias externa, interna y de código. En el caso de la memoria externa y la interna se pueden abrir varias vistas, por lo que se crea un array de variables. Dada su dimensión el número máximo de ventanas que pueden ser abiertas es de 11.

Public DirtoRow(0 To 65535) As Long

Public RowtoDir(0 To 65535) As Long

Dado que una instrucción ocupa uno o más bytes, el número de instrucciones será menor o igual que el número de posiciones de memoria. El primer array relaciona la dirección de memoria con el número de la instrucción que contiene. El segundo array devuelve la posición de memoria en la que comienza una determinada instrucción.

Son utilizadas en el manejo de la ventana de código. En esta ventana cada línea es una instrucción, y la barra de scroll cuenta instrucciones. DirtoRow se emplea para transformar el valor de la dirección en los saltos, tratamiento con el PC,... RowtoDir a la hora de rellenar los campos de dirección y para referirnos a la estructura Instru correcta cuando señalamos una instrucción particular.

Public MaxCodeMem As Long

Almacena el tamaño de la memoria de programa. Se modifica al definir el hardware del sistema y se utiliza en la comprobación de límites.

Public MaxRowCode As Long

Indica el Número de instrucciones que contiene la memoria. Se modifica cada vez que abrimos un programa y siempre será menor que MaxCodeMem.

Public MaxExtMem As Long

Public MaxIntMem As Long

En estas variables se almacena el tamaño de las memorias externa e interna respectivamente. Su valor se define al definir el hardware simulado.

Public ExtMem(0 To 65535) As Long

Public IntMem(0 To 65535) As Long

Estos arrays almacenan los valores que contienen las memorias externa e interna respectivamente. Su tamaño es fijo independientemente de la memoria definida.

Public NumVentMemExt As Long

Public NumVentMemInt As Long

Dado que se pueden abrir varias ventanas de memoria necesito saber el número de ellas para efectuar tareas sobre ellas como es refrescar los valores mostrados. Estas variables almacenan ese número para las ventanas de memoria externa e interna respectivamente.

Public Fuente As New StdFont

Las ventanas de memoria pueden cambiar el tipo de fuente empleado. En esta variable almaceno ese tipo para asignárselo a las nuevas ventanas que se vayan abriendo, ya que al crearse adoptan un tipo de fuente por defecto que no tiene por qué coincidir con el definido.

Public Acc As Long

Public B As Long

Public Psw As Long

Public SP As Long

Public DPTR As Long

Public DPL As Long

Public DPH As Long

Public P0 As Long

Public P1 As Long

Public PCON As Long

Public TCON As Long

Public TMOD As Long

Public T0 As Long

Public T0L As Long

Public T0H As Long

Public T1 As Long

Public T1L As Long

Public T1H As Long

Public SCON As Long

Public SBUF As Long

Public P2 As Long

Public IE As Long

Public P3 As Long

Public IP As Long

Public T2CON As Long

Public RCAP2 As Long

Public RCAP2L As Long

Public RCAP2H As Long

Public T2 As Long

Public T2L As Long

Public T2H As Long

Todos los SFR's tienen un nombre definido. Para facilitar la programación se definen estas variables con el nombre del SFR y cuyo valor corresponde con el valor de la dirección de memoria donde se encuentra.

Public SBUF2 As Long

En el puerto serie el dato de salida se se escribe en SBUF. Cuando el puerto recibe un dato basta con leer el registro SBUF, pero el microcontrolador desvía esta lectura a un registro diferente del de escritura. SBUF2 representa ese registro.

Public PinP0 As Long

Public PinP1 As Long

Public PinP2 As Long

Public PinP3 As Long

En el simulador se hace una diferencia entre los SFR's que son buffer de los puertos y los pines físicos de estos puertos, ya que según la instrucción ejecutada se lee uno u otro. Estas cuatro variables almacenan el valor de los pines de cada puerto.

Public C As Long

Public AC As Long

Public F0 As Long

Public RS1 As Long

Public RS0 As Long

Public OV As Long

Public P As Long

Del mismo modo que se definen variables para almacenar la posición de los SFR's definimos estas para asociar el nombre de cada uno de los flags del PSW con dirección de bit.

Public RefreshEnable As Boolean

Se emplea para permitir o no que las ventanas refresquen los valores que presentan.

Public Resaltando As Boolean

Indica que estoy coloreando las líneas de la ventana de memoria de programa para resaltar los breakpoints definidos y el PC.

Evita que CodeMemForm detecte el cambio de fila o columna y se repita el código previsto para cuando ocurra este evento.

Public NumBreak As Integer

Indica el número de breakpoints definidos. Usado cuando efectuamos operaciones con los breakpoints para evitar hacer una referencia a un elemento del array no definido.

Public BreakMask As Integer

Máscara de grupos de breakpoints. Si el bit correspondiente está a uno indica que el grupo de breakpoints correspondiente está activo. Se utiliza en la comprobación que determina si el breakpoint detendrá la ejecución o no.

Public NumVar As Integer

Número de variables definidas.

Public IEIPModificado As Integer

Indica si se modificó IE o IP en la última instrucción ejecutada. Se usa en la atención a las interrupciones, ya que si se da el caso, la interrupción no debe ser atendida.

Public EjecutandoInt As Integer

Indica que la el código en ejecución pertenece al servicio de una interrupción. La ejecución de una interrupción puede ser interrumpida por otra, pero esta segunda interrupción no se interrumpirá en ningún caso. El máximo anidamiento es uno.

Public IntPrio As Integer

Almacena la prioridad de la interrupción que se está sirviendo. Necesaria porque una segunda interrupción anidada sólo se atiende si es de prioridad superior.

Public MemCodeInt As Integer

Si su valor es cero el microcontrolador no tiene memoria de programa interna.

Public Procesador As Integer

Almacena el tipo de procesador que se está simulando. Si su valor es cero se utiliza el 8051. Si es 1 el 8052.

Public PC As Long

Almacena el valor del contador de programa.

Public MaxCodeMemInt As Long

Indica el tamaño de la memoria de programa interna si es que se define que el microcontrolador la posee. En caso contrario este valor no es significativo.

Public BreakDetectado As Boolean

Cuando se detecta un breakpoint en ejecución la instrucción no se ejecuta. Si el breakpoint no se borra, cuando comenzase de nuevo la ejecución el breakpoint volvería a detectarse. Esta variable evita que esto suceda.

Public RefrescoCont as Boolean

Esta es la variable que indica si se ha seleccionado la opción de mostrar los cambios realizados a cada instrucción durante la ejecución continua.

3.2.2 Inicio de la Aplicación

La aplicación comienza a ejecutarse creándose un objeto de tipo MDIForm1, que es la ventana principal y que contendrá las ventanas de memoria, código,... En ella se presentan la barra de menús y la barra de herramientas.

MDIForm.Load

Argumentos: Ninguno.

Descripción: Esta es la primera función que se ejecuta. Lo primero que hace es cargar una ventana de presentación mientras se inicia la aplicación que será descargada al finalizar la subrutina. Se define entonces el tamaño de la ventana, que se ajustará a 640x480 pixel.

Se llama entonces a las rutinas de iniciación InitVar, InitSFR, InitCodeMem, InitIntMem, InitExtMem, InitRowDir.

Se crea una ventana de código, se muestra la ventana de SFR's y se llama a NMemInt_Click y NMemExt_Click para crear las ventanas que muestran las memorias interna y externa.

InitVar

Argumentos: No posee

Descripción: Da valor inicial a las diferentes variables globales utilizadas. Entre ellas están las que definen el tamaño de las memorias y el tipo de microcontrolador. Se asocia el nombre de los SFR's y los flags con su

dirección en memoria. Se inician las variables que controlan los breakpoints, las variables y las interrupciones.

InitSFR

Argumentos: No posee.

Descripción: Define el valor de los SFR's después de un Reset. Se llama al iniciar el simulador, al realizar un reset o cambiar la definición del hardware simulado.

InitCodeMem

Argumentos: No posee.

Descripción: Inicia la memoria de código con instrucciones NOP rellenando los campos de la estructura Instru asociada a cada posición de memoria con los valores adecuados.

InitRowDir

Argumentos: No posee.

Descripción: Inicia los arrays que relacionan número de instrucción con posición de memoria y viceversa. En un principio estos dos valores son coincidentes (NOP ocupa un byte).

InitIntMem

Argumentos: Ninguno.

Descripción: Inicia la memoria interna con ceros en todas sus posiciones mediante un bucle que recorre las direcciones desde cero hasta la máxima definida.

InitExtMem

Argumentos: Ninguno.

Descripción: Inicia la memoria externa con ceros en todas sus posiciones mediante un bucle que recorre las direcciones desde cero hasta la máxima definida.

VentCode_Click

Argumentos: Ninguno.

Descripción: Corresponde a la opción de menú Ver-Ventana de Código. Muestra la ventana de código descargando previamente la que pudiera estar mostrándose en ese momento, ya que de otro modo se mostrarían dos ventanas.

VerSFR_Click

Argumentos: Ninguno.

Descripción: Opción de menú Ver-SFR's. Muestra la ventana de SFR's.

NMemInt_Click

Argumentos: Ninguno.

Descripción: Opción de menú Ver-Nueva Memoria Interna. Abre una nueva ventana de memoria interna. Crea una nueva instancia de IntMemFormy la muestra. Actualiza la variable NumVentMemInt.

NMemExt_Click

Argumentos: Ninguno.

Descripción: Opción de menú Ver-Nueva Memoria Externa. Abre una nueva ventana de memoria externa. Crea una nueva instancia de ExtMemForm y la muestra. Actualiza la variable NumVentMemExt.

3.2.3 Configuración de Hardware

Con esta parte del código especificamos el microcontrolador que simulamos y la configuración de memoria que deseamos.

DefHard_Click

Argumentos: Ninguno.

Descripción: Esta es la función que se llama cuando se selecciona la opción de menú Opciones – Definir Hardware. Esta función se limita a crear la ventana ConfHwForm descrita posteriormente. Esta realiza la configuración y luego reinicia el simulador mediante la función ReInit.

Reinit

Argumentos: Ninguno

Descripción: Reinicia el simulador con los siguientes pasos:

- Muestra de nuevo las ventanas de memoria externa e interna y código.
- Borra las memorias con las funciones `BorrarMemInt_Click`, `BorrarMemExt_Click`, `BorrarCodeMem_Click`.
- Resetea el microprocesador con la función `Reset_Click`.
- Refresca la presentación de los nuevos valores con la subrutina `RefrescaVentanas`.
- Borra los breakpoints llamando a `DefBreakForm.Command3_Click` (descrito posteriormente).
- Borra las variables redimensionando al array `Var(x)` a cero y refresca la ventana de variables.

BorrarMemInt_Click

Argumentos: Ninguno.

Descripción: Opción de menú Opciones - Borrar Memoria Interna. Para ello llama a `InitIntMem` y a `RefrescaVentanas`.

BorrarMemExt_Click

Argumentos: Ninguno.

Descripción: Opción de menú Opciones-Borrar Memoria Externa. Llama a `InitExtMem` para dejar los valores de la memoria a cero y refresca las ventanas mostradas.

BorrarCodeMem_Click

Argumentos: Ninguno.

Descripción: Opción de menú Opciones-Borrar Código. Llama a `InitRowDir`, `InitCodeMem` con lo que pone toda la memoria con instrucciones NOP y ajusta los arrays `RowtoDir` y `DirtoRow`. Por último refresca las ventanas.

Reset_Click

Argumentos: Ninguno.

Descripción: Opción de menú Opciones - Reset. devuelve los SFR's a sus valores iniciales. Las memorias, variables y breakpoints los deja como estaban. Efectúa una llamada a la subrutina que para el programa en ejecución por si estuviéramos simulando algo en el momento del reset e inicia algunas variables para definir que no está ejecutando ninguna interrupción.

RefrescaVentanas

Argumentos: Ninguno.

Descripción: Llama a las funciones de refresco que están implementadas en la definición de las ventanas de memoria, código y SFR's.

3.2.4 Ejecución del programa simulado.

El control de ejecución se realiza con las siguientes rutinas que son opciones del menú ejecutar

Ejecutar_Click

Argumentos: Ninguno.

Descripción: Inicia la ejecución continua del programa. Lo primero que hace es cambiar el título de la ventana para mostrar la palabra ejecutando y deshabilitar el refresco de las ventanas para mejorar la velocidad de ejecución.

Según la opción del menú Opciones – Refrescar Continuasmente esté activada o no, así asignará el valor a la variable RefrescoCont. Esta controla si cuando se produce un cambio en los registros o en la memoria se deben reflejar a cada instrucción o sólo cuando se detiene la ejecución.

Si las interrupciones están habilitadas llama la función CompruebaInterrupción que se encarga de realizar las operaciones necesarias para decidir si hay alguna pedida y si debo atenderla.

A continuación se determina si hay algún breakpoint definido. Si no es así se inicia un bucle Do para ejecutar las instrucciones del programa sin comprobar si se alcanza algún breakpoint.

En caso de haber algún breakpoint definido, hay una estructura if-else. Esta estructura está dentro de un bucle do que permite la ejecución de sucesivas instrucciones mientras la variable Ejecutando sea verdadera. En el bloque else se comprueba si hay definido algún breakpoint con la

función CompruebaBreakpoints. Si no los hay se ejecuta una instrucción con la rutina EjecutaInstrucción y se comprueba la petición de interrupciones. En el caso de que hubiera un breakpoint se activa la variable BreakDetectado. Esto permite que a la siguiente instrucción se ejecute la parte de código correspondiente al if, similar a la descrita pero sin comprobación de breakpoints, ya que de lo contrario se volvería a parar en el breakpoint que se acaba de detectar.

Cada 8 instrucciones se llama a DoEvents para permitir entradas del usuario. Si esa entrada es una llamada a Parar_Click, cambiará el valor de Ejeutando y la ejecución se detendrá.

La ejecución de instrucciones prosigue mientras la variable ejecutando sea verdadera.

Parar_Click

Argumentos: Ninguno.

Descripción: Detiene la ejecución. Habilita el refresco de ventanas, Muestra los cambios acontecidos durante la ejecución, las refresca cambia el título de la ventana para eliminar la palabra ejecutando y da el valor falso a la variable Ejeutando.

IraCursor_Click

Argumentos: Ninguno.

Descripción: Realiza la ejecución hasta que el PC se sitúe en la dirección que marca el cursor.

El proceso es el mismo que en la subrutina Ejecutar_Click, pero para que el bucle continúe además de ser Ejeutando verdadero, el PC debe ser distinto a la dirección donde está el cursor. Esta dirección se conoce leyendo la columna de dirección de la ventana de código.

Paso_Click

Argumentos: Ninguno.

Descripción: Ejecuta una instrucción. Deshabilita el refresco de ventanas, comprueba las interrupciones y ejecuta una instrucción, tras lo que llama a Parar_Click. No se tienen en cuenta los breakpoints definidos.

PasoSaltando_Click

Argumentos: Ninguno.

Descripción: Ejecuta una instrucción pero si es del tipo CALL sigue ejecutando instrucciones hasta que retorna de la subrutina.

La primera es ejecutada como en Paso_Click. Con una estructura if y con la variable InstruccionEjecutada compruebo si es una de esas instrucciones, en cuyo caso se inicia un bucle donde se ejecutan instrucciones teniendo en cuenta los breakpoints e interrupciones, y del que se saldrá si Ejecutando es falso (Llamada a Parar_Click) o si la instrucción ejecutada es RET. Durante este periodo de ejecución continua se asigna un valor a la variable RefrescoCont según haya que mostrar los cambios a cada instrucción.

SaliSub_Click

Argumentos: Ninguno.

Descripción: Ejecuta instrucciones hasta que encuentra RET o RETI. La programación es la misma que en Ejecuta_Click pero a la condición del bucle se le añade que la última instrucción ejecutada no sea RET ni RETI en cuyo caso se detiene la ejecución.

EjecutaInstrucción

Argumentos: Ninguno.

Descripción: Esta función está definida en el fichero Modulo1 y se encarga de ejecutar las diferentes instrucciones máquina.

Comienza recogiendo el opcode donde apunta el PC. Se comprueba que el PC apunte al inicio de una instrucción, si no es así ejecuta NOP. Del opcode extrae el primer byte, que es el identificativo de la instrucción.

Si el acceso a memoria de código es externo pone el puerto cero a HFF.

La siguiente estructura case es la que desvía el programa a la ejecución de la instrucción adecuada. En todas las instrucciones se actualiza la posición del PC y se establece el número de ciclos máquina que tarda en ejecutarse (Ciclos), valor que se usará en los temporizadores. Para lecturas y escrituras en las diferentes memorias se usan funciones como ReadExtMem, WriteBit, ... que serán descritas más tarde. Los datos con los que opera la instrucción si no fueran implícitos, se sacan también en la porción del opcode correspondiente de la estructura Instru a la que se hace referencia en la presente instrucción. En las instrucciones en las que se

modifican los Flags como resultado de una operación, se llama a la subrutina `CalculaFlags` para actualizar su valor.

Una vez ejecutada la instrucción se calcula el flag P, ya que se modifica con cada instrucción. Se llama también a la función `CompruebaTimers` para actualizar el tiempo transcurrido y actuar en consecuencia, y a la rutina `Comprueba serie` para decidir si debo enviar un bit o recibirlo por el puerto serie.

Por último comprueba que el PC marca una posición válida y si no es así se pone a cero.

3.2.5 Añadiendo Variables

`AddVariable_Click`

Argumentos: Ninguno.

Descripción: Con esta función se añade una nueva variable. Es una opción del menú Depurar - Añadir Variables. Esta subrutina se limita a llamar al cuadro de diálogo `AddVarForm` y a mostrar la ventana de variables si hay alguna definida.

3.2.6 Definiendo Breakpoints

A la hora de Definir Breakpoints tenemos dos opciones, una de ellas es utilizar el cuadro de diálogo diseñado para ello y otra añadirlo de forma inmediata sobre el cursor.

`DefBreak_Click`

Argumentos: Ninguno.

Descripción: Opción de menú Depurar - Definir Breakpoint. Abre el cuadro `DefBreakForm` y éste se encarga de todo.

`CurBreak_Click`

Argumentos: Ninguno.

Descripción: Opción de menú Depurar - Breakpoint en Cursor. Define un breakpoint en la posición que marca el cursor o lo borra si ya existía. Mediante una condición if-else distingo si tengo o no breakpoints definidos. Si los hay definidos comprueba para cada uno de ellos si su dirección de inicio coincide con la que marca el cursor. Si encuentra alguno definido de

dirección única lo borra moviendo los datos de cada breakpoint una posición antes dentro del array de breakpoints definidos y redimensionando este array con una posición menos. Si no ha coincidido ninguna dirección entonces defino un breakpoint aumentando la dimensión del array y rellenando la nueva estructura con los datos necesarios. El breakpoint así definido es de ejecución, alcance permanente, en dirección única y perteneciente al grupo 0.

3.2.7 Abriendo Ficheros de Código

Abrir_Click

Argumentos: Ninguno

Descripción: Opción de menú Archivo - Abrir. Abre un cuadro de diálogo estándar para la selección del fichero a abrir. Se definen dos opciones en el filtro de ficheros, una *.ihx para los ficheros Intel y otra *.s19 para los Motorola. Según la opción escogida al pulsar aceptar llama a Abrir_Ihx o Abrir_s19. Antes de abrir el fichero se borra la memoria de código.

Abrir_Ihx

Argumentos: Filename As String. Entrega la cadena que describe el fichero que deberá ser abierto por la subrutina.

Descripción: Carga en la memoria de código un programa en formato Intel.

Comienza abriendo el fichero definido e inicia un bucle que va a procesar los diferentes registros del fichero mientras la variable Fin sea cero. Para el control de errores se calcula un valor que será almacenado en la variable Suma. Mediante la función Input se van leyendo los diferentes campos del registro. Primero el principio de línea indicado por el carácter ':' En segundo lugar el número de bytes que tiene el registro. La variable Suma se actualiza cada vez que se lee un nuevo byte para así calcular el código protector de errores.

El tercer campo es la dirección de 4 bytes, aunque se lee de 2 en 2 para actualizar la variable Suma. Luego se lee el tipo de registro, normalmente a cero, pero si es el último de 1 fichero su valor es 1 y se pone la variable Fin a uno para salir del bucle principal.

El resto de bytes menos el último son los opcodes de las instrucciones. Se crea un bucle para leer los bytes que quedan uno a uno y en cada uno de ellos se llama a la función optomne definida en Modulo1 que es la que se encarga de formar el mnemónico y decidir cuándo finaliza una instrucción y comienza otra (variable FinOp). Con ayuda de la variable FinOp se hace

que el campo de dirección de la estructura instrucción sea -1 en aquellas posiciones que no sean principio de instrucción.

Una vez finalizado el bucle se lee el código de protección y se compara con la variable suma, emitiendo un error si no coinciden.

En este punto se realiza la asignación dirección de memoria – número de instrucción actualizado las estructuras RowtoDir y DirtoRow. A continuación se comprueba que el tamaño del programa no supere el tamaño de la memoria de código que hemos definido (el array codemem es siempre de 64k aunque no se permita utilizar más que determinadas posiciones).

Por último se reflejan los cambios refrescando la ventana de código.

Abrir_s19

Argumentos: Filename As String. Entrega la cadena que describe el fichero que deberá ser abierto por la subrutina.

Descripción: La estructura de esta rutina es igual a la anterior pero adaptándose a los campos definidos por el formato S19 de Motorola.

CalculaSimbolo

Argumentos: Dato As Integer. Entrega la dirección hexadecimal que será tratada.

Descripción: Esta rutina convierte la dirección hexadecimal de cualquier SFR en el símbolo que lo representa, de modo que el código sea más legible. Esto se consigue mediante una estructura Case, devolviendo en cada caso el símbolo asociado.

CalculaSimboloBit

Argumentos: Dato As Integer. Entrega la dirección hexadecimal que será tratada.

Descripción: Similar a la anterior, convierte la dirección de los bits en el símbolo correspondiente, si es que lo tiene.

3.2.8 Abriendo ficheros de memoria de datos.

AbrirMemExt_Click y AbrirMemInt_Click

Argumentos: Filename As String. Entrega la cadena que describe el fichero que deberá ser abierto por la subrutina.

Descripción: La estructura de estas funciones es similar a Abrir_Click pero llaman a las funciones que cargan el código en memoria externa o interna. Antes de abrir los ficheros se llama a las funciones de borrado de memorias.

AbrirMemExt_ihx, AbrirMemInt_ihx, AbrirMemExt_s19 y AbrirMemInt_s19

Argumentos: Filename As String. Entrega la cadena que describe el fichero que deberá ser abierto por la subrutina.

Descripción: Estas rutinas son las encargadas de abrir los ficheros de memoria (bien en formato Intel o bien en Formato Motorola) y almacenarlos en la memoria correspondiente.

La estructura es similar a Abrir_ihx y Abrir_s19, pero en el bucle de procesado de los bytes de datos, en vez de calcular los mnemónicos, escriben directamente el byte en la memoria correspondiente.

3.2.9 Búsqueda de Datos en Memoria

Buscar_Click

Argumentos: Ninguno.

Descripción: Corresponde a la opción de menú Edición-Buscar. Inicia la búsqueda de una cadena. Para ello llama a la función búsqueda con la opción false.

Nbúsqueda_Click

Argumentos: Ninguno.

Descripción: Opción de menú Edición - Nueva Búsqueda. Busca de nuevo la cadena ya definida por Busqueda_Click. Llama a la subrutina búsqueda con la opción true.

Búsqueda

Argumentos: Repite As Boolean. Indica si la búsqueda es inicial o repite la búsqueda anterior.

Descripción: Dos primeras instrucciones if comprueban que si es nueva búsqueda y he llegado a cero o Limite (final de la memoria en cuestión) se vuelva a empezar la búsqueda por la dirección cero o por el final de la memoria, según el sentido de búsqueda ascendente o descendente.

Si la función la llamamos para una primera búsqueda (Repite = False) se abre el cuadro de diálogo que recoge la cadena a buscar y demás opciones. Este cuadro no se destruye al esconderse, por lo que se pueden leer sus valores una vez que se ha cerrado. Así en Opcion almaceno el tipo de memoria donde buscar.

Si la cadena a buscar no es nula continuo y si es primera búsqueda ajusto el incremento (a sumar a la dirección de búsqueda) y el límite (0 o el máximo de memoria definida) según la dirección de búsqueda que hayamos escogido.

A continuación inicio un bucle donde se va comparando el primer bit de la cadena a buscar con la primera posición de memoria. Si coincide sigo comparando los sucesivos caracteres de la cadena con las siguientes posiciones de memoria. Si toda la cadena coincide se termina el bucle asignando el valor True a la variable Fin y resalto la posición donde comienza la cadena en la memoria. Si en algún punto no coincide un carácter, se vuelve a comparar desde el primer carácter de la cadena. Si llego al límite y no he encontrado la cadena se emite un mensaje notificándolo.

Por último de descarga la ventana de petición de datos.

MuestraCadena

Argumentos: Direccion As Long. Contienen el valor de la dirección a resaltar.

Opcion As Integer. Define el tipo de memoria que se resaltará, si es 0 la memoria interna, si es 1 la memoria externa y si es 2 la memoria de código.

Descripción: Es la rutina que se encarga de resaltar la dirección encontrada. Con una estructura Case se distingue entre memoria de datos y de código. Si es memoria interna o externa se detecta cuál de todas las que están abiertas es la activa, comprando el manejador de la ventana activa que devuelva la ventana principal con el de las ventanas de memoria abiertas. Si ninguna está activa se creará una nueva (que estará activa nada más crearse). En cualquier caso se llama a la función ResaltaCadena de la correspondiente ventana que es la que señala la dirección escogida.

Si se trata de memoria de código se llama directamente a la función ResaltaCadena porque sólo puede haber una abierta a la vez.

3.2.10 Otras Opciones del Menú

Salir_Click

Argumentos: Ninguno.

Descripción: Opción del menú Archivo - Salir. Termina la aplicación.

VerVar_Click

Argumentos: Ninguno.

Descripción: Opción Edición - Ver Variables. Muestra la ventana de variables y refresca los valores presentados.

VerTerminal_Click

Argumentos: Ninguno.

Descripción: Opción Edición - Ver Terminal. Muestra la ventana de terminal.

ModificarPC_Click

Argumentos: Ninguno.

Descripción: Modifica el valor del contador de programa. Llama a la subrutina IBox para recoger el nuevo valor. Si apunta a una dirección donde no comienza en una instrucción emite un mensaje de error. Si el valor además de existir está dentro de los límites de la memoria definida se le asigna ese valor.

CambiaFuente_Click

Argumentos: Ninguno.

Descripción: Esta función abre un cuadro de diálogo estándar para determinar el tipo de fuente que se quiere aplicar. Una vez extraídos estos datos se modifican las propiedades de las ventanas de memoria interna, de código y externa y se redimensiona el ancho de las columnas de los controles MSFlexGrid para dar cabida a la nueva fuente.

Cascada_Click, MosaicoHorizontal_Click, MosaicoVertical_Click,
OrdenarIconos_Click

Argumentos: Ninguno.

Descripción: Son las opciones del menú ventana y todas ellas llaman a la función que realiza la acción correspondiente.

GenInt_Click

Argumentos: Ninguno.

Descripción: Opción del menú Depurar - Generar Interrupción. Llama al cuadro de diálogo GenIntForm que es el que se encarga de todo.

3.2.11 Tratamiento de la Barra de Herramientas

Toolbar1_ButtonClick

Argumentos: ByVal Button As ComctlLib.Button. Identifica el nBotón que se ha pulsado mediante su clave.

Descripción: Las opciones que se presentan son idénticas que las de algunos elementos del menú, por lo que mediante una estructura Case, según el icono que se haya pulsado se llamará a la función que se llama cuando se pica en el elemento del menú correspondiente.

3.2.12 Cuadro de Diálogo GenIntForm

Con este apartado se termina la descripción de las subrutinas del fichero MDIForm1 y de todas aquellas que tienen que ver con los menús y la ventana principal. A partir de ahora se describirá el funcionamiento de los cuadros de diálogo y otros objetos a los que se ha hecho referencian anteriormente, comenzando por el cuadro que controla la generación de interrupciones.

Form_load

Argumentos: Ninguno.

Descripción: Se ejecuta al cargarse la ventana y según el procesador que está simulando deshabilita o no las interrupciones que genera el temporizador 2.

Commandx_Click

Argumentos: Ninguno.

Descripción: El cuadro de diálogo tiene nueve botones. De ellos el noveno es el de cancelar que descarga la ventana al pulsarlo sin realizar ninguna otra operación.

Los otros ocho botones corresponden a uno por cada interrupción que se puede pedir. El tratamiento es el siguiente:

Mediante estructuras If se comprueba que no se haya modificado IE ni IP en la instrucción anterior, que las interrupciones en general y esa en particular están habilitadas y que no hay otra interrupción de mayor o igual prioridad ejecutándose.

Si esto se cumple se realiza una llamada LCALL a la posición de memoria predefinida que atiende la interrupción. Se informa al sistema de que se está ejecutando una interrupción mediante la variable EjecutandoInt y se activa el flag de petición de interrupción correspondiente excepto si el hardware desactiva automáticamente este flag para la interrupción correspondiente.

3.2.13 Cuadro de Diálogo BuscarForm

Text1_Change

Argumentos: Ninguno.

Descripción: Contiene los datos a buscar representados en forma de cadena de texto. La cadena de texto se convierte en hexadecimal para representarlo en text2. Para ello emplea un bucle que separa los caracteres uno a uno, calcula su valor ASCII y lo añade como una cadena a la variable Texto que luego se asignará al control Texto2.

Text2_Change

Argumentos: Ninguno.

Descripción: Contiene la representación de los datos buscados en forma hexadecimal. Tiene una primera parte para asegurar que el carácter pulsado sea hexadecimal. Esto se realiza separando los caracteres uno a uno. Si es un valor hexadecimal se deja en la cadena, pero si no lo es se elimina de esta.

La segunda parte espera a que el número de caracteres sea par y en ese caso los junta dos a dos calculando luego el carácter correspondiente a ese

valor interpretado como ASCII. Este carácter se añade a la variable Texto que luego se asignará al control Texto1.

Text3_Change

Argumentos: Ninguno.

Descripción: Representa la casilla donde se introduce la dirección a partir de la cual comienza la búsqueda. Esta es una rutina para asegurar que el carácter pulsado sea hexadecimal. El método empleado es el mismo que en Text2_Change.

Command1_Click

Argumentos: Ninguno.

Descripción: Se ejecuta cuando se pulsa el botón Aceptar. Comprueba que el valor introducido en Texto3 que corresponde a la casilla de dirección señala una posición de memoria existente, determinando el máximo definido mediante la función LeeLimite. Si el valor es incorrecto emite un mensaje notificándolo y retorna al cuadro de diálogo. Si el valor es adecuado esconde la ventana pero no la descarga para que la rutina que la llamó pueda leer los valores de los diferentes controles.

Command2_Click

Argumentos: Ninguno.

Descripción: Corresponde al botón Cancelar. Pone las cadenas de texto a nulo lo que evita que la subrutina Busqueda efectúe la búsqueda, y esconde la ventana.

LeeLimite

Argumentos: Ninguno.

Descripción: Según el tipo de memoria seleccionada, determinado por el array de controles Option1, devuelve cuál es el máximo valor definido.

LeeMemoria

Argumentos: Direccion As Long. Dirección de memoria a leer.

Opcion As Integer. Determina la memoria donde se realizará la lectura. Si es 0 en memoria interna. Si es 1 en memoria externa y si es otro valor en memoria de código.

Descripción: Lee un dato de la memoria interna, externa o de código según la opción que haya seleccionado. En el caso de la memoria de código el campo opcode de la estructura Instru almacena el opcode completo si es dirección es principio de instrucción, por lo que se crea un bucle do para extraer sólo el byte más significativo.

3.2.14 Cuadro de Diálogo DefBreakForm

El cuadro se carga e inmediatamente se ejecuta la rutina Form_Load.

Form_Load

Argumentos: Ninguno.

Descripción: Define los valores de los controles al abrirse la ventana. Recoge la máscara de breakpoints y actualiza los checkbox con el valor correspondiente. Para ello convierte el valor de la máscara a binario y modifica el valor del checkbox correspondiente a cada bit.

Check1_Click

Argumentos: Index As Integer. Contiene el índice del control dentro del array de controles que se ha pulsado.

Descripción: Actualiza la máscara de breakpoints cada vez que cambio el valor de un elemento del array de checkbox. Comprueba uno a uno el valor de los elementos y según su índice se le asigna un peso y se le suma a los anteriores. El resultado es la máscara.

Option1_Click

Argumentos: Index As Integer. Contiene el índice del control dentro del array de controles que se ha pulsado.

Descripción: Option1 es un array de tres elementos que define la memoria sobre la que se aplica el breakpoint. Según active o desactive el indicador de memoria de código deshabilito o no la posibilidad de escoger el tipo de acceso (opción(4)).

Option3_Click

Argumentos: Index As Integer. Contiene el índice del control dentro del array de controles que se ha pulsado.

Descripción: Según seleccione dirección única o un rango se deshabilita o no el recuadro de texto que recoge la dirección final.

Text1_Change y Text2_Change

Argumentos: Ninguno.

Descripción: Se encargan de que cada vez que añadido un carácter este sea hexadecimal o ignora la pulsación.

Refrescar

Argumentos: Ninguno.

Descripción: Muestra en la ventana del cuadro de diálogo los breakpoints que están definidos. Para ello borra en primer lugar el control list y para cada breakpoint definido crea una línea de texto con la dirección, memoria, acceso,... Para saber estos datos se leen los controles Option y Text correspondientes de esta misma ventana. Una vez creada la línea se añade a la control List.

Command4_Click

Argumentos: Ninguno.

Descripción: Corresponde al botón cerrar. Descarga la ventana.

Command3_Click

Argumentos: Ninguno.

Descripción: Borra todos los breakpoints redimensionado el array de estructuras Breakpoint a cero. Refresca el control List para borrar los breakpoints mostrados y la ventana de código para borrar los colores que hubieran definidos.

Command2_Click

Argumentos: Ninguno.

Descripción: Borra el breakpoint que está sobre el cursor del control List. Para ello lee el índice del elemento seleccionado en la lista que muestra los datos del breakpoint del mismo índice en el array. El borrado lo realiza moviendo los datos de los breakpoints de índice superior al que borro una posición antes en el array y reduciendo las dimensiones del array en uno. Este proceso no se realiza si la lista está vacía.

Command1_Click

Argumentos: Ninguno.

Descripción: Botón añadir breakpoint. En una primera parte del código se comprueba que los datos introducidos son correctos: las direcciones no están fuera de rango, la dirección final es mayor que la inicial y que esté definido un grupo para el breakpoint. En caso de haber algún error emite un mensaje.

Si los datos son correctos se crea un nuevo elemento en el array breakpoints y va leyendo los valores de los controles para ir rellenando la estructura breakpoints correspondiente con los datos adecuados. Al mismo tiempo se va creando una línea de texto que luego se añadirá a la lista donde se muestran los breakpoints definidos.

Por último se refresca la ventana de código por si hay que colorear alguna línea.

3.2.15 Cuadro de Diálogo AddVarForm

Se encarga de tomar los datos para añadir una variable.

Text1_Change

Argumentos: Ninguno.

Descripción: Cuando se cambia el texto se comprueba que el número sea hexadecimal. Si no lo es se ignora la última pulsación.

Command2_Click

Argumentos: Ninguno.

Descripción: Botón cancelar. Descarga la ventana.

Command1_Click

Argumentos: Ninguno.

Descripción: Botón aceptar. Comprueba que la dirección se encuentra dentro de los márgenes permitidos según el tipo de memoria seleccionada, mostrando un mensaje de error si no lo está. En caso contrario almacena en la estructura Var la dirección y tipo de variable y actualiza la variable NumVar. Por último refresca la ventana de variables para mostrar la nueva definición.

3.2.16 Cuadro de Diálogo IBoxForm

Este cuadro se encarga de pedir un valor comprobando que está dentro de unos límites.

Text1_change

Argumentos: Ninguno.

Descripción: Comprueba que el carácter pulsado es hexadecimal. Si no lo es lo ignora.

Command1_Click

Argumentos: Ninguno.

Descripción: Botón aceptar. Esconde la ventana pero no la descarga para que la subrutina que la llamó pueda leer el valor de los controles.

Command2_Click

Argumentos: Ninguno.

Descripción: Botón cancelar. Escribe en la ventana de texto una cadena vacía y esconde la ventana.

3.2.17 Ventana de Configuración de Hardware ConfHwForm

Es un cuadro de diálogo desde donde le decimos al programa qué hardware estamos simulando.

Form_Load

Argumentos: Ninguno.

Descripción: Es la primera que se ejecuta al llamar a la ventana. Presenta en las casillas correspondientes los valores de memoria actuales, el tipo de microcontrolador y si posee memoria de código interna.

Option1_Click

Argumentos: Ninguno.

Descripción: Si se señala esta opción es que se ha seleccionado el 8051. La memoria interna se ajusta a H7F. Si además se indica que posee memoria de código interna comprueba que la definida sea mayor de 4k. Si no es así escribe el mínimo en la casilla correspondiente.

Option2_Click

Argumentos: Ninguno.

Descripción: Se ha seleccionado el 8052. La memoria interna se ajusta a HFF. En caso de tener memoria de código interna se exige que la memoria de código sea mayor o igual que 8k.

Tex1_change

Argumentos: Ninguno.

Descripción: Comprueba que el texto introducido es un número hexadecimal. Si no lo es, se ignora el carácter no hexadecimal.

Text2_change

Argumentos: Ninguno.

Descripción: Igual comportamiento que la anterior.

Text1_LostFocus

Argumentos: Ninguno.

Descripción: Cuando se deja el texto que indica el tamaño de la memoria de código se comprueba que el mínimo de memoria se cumple. Si no es así se ajusta al mínimo.

Check1_Click

Argumentos: Ninguno.

Descripción: Si cambia a verdadero indica que el microcontrolador posee memoria de código interna y se comprueba el valor mínimo de ésta memoria.

Command2_Click

Argumentos: Ninguno.

Descripción: Botón cancelar. Se descarga la ventana.

Command1_Click

Argumentos: Ninguno.

Descripción: Botón aceptar. Comprueba que los datos del tamaño de las memorias están dentro de los límites y redefine MaxExtMem, MaxIntMem, MaxCodeMem y MaxRowCode.

Actualiza la variable Procesador y MaxCodeMemInt.

Con estos cambios de variables sólo queda descargar la ventana y llamar a la función ReInit para que el simulador se inicie con los nuevos cambios.

3.2.18 Ventana de Memoria Interna IntMemForm

Controla el comportamiento de la ventana que muestra los contenidos de la memoria de datos interna.

Posee un control MSFlexGrid para mostrar los resultados y un control VScroll para controlar el desplazamiento de las direcciones visualizadas.

Se define una variable local NumLineas que establece el número de filas que tendrá el control MSFlexGrid que dependerá del tamaño de la ventana pero que siempre será el número de líneas vistas más cuatro. Esto es así para poder determinar el desplazamiento en la memoria cuando se actúa con las teclas de cursor y de avance y retroceso de página.

Utiliza las siguientes funciones:

Form_load

Argumentos: Ninguno.

Descripción: Se ejecuta el al cargar la ventana y se encarga de llamar a las dos funciones que definen las características de los controles MSFlexGrid y Vscroll .Define además el tamaño inicial de la ventana y llama a la función Refrescar para mostrar los valores de memoria adecuados.

InitMSFlexGrid1

Argumentos: Ninguno.

Descripción: Define las características del control: Fuente usada, número de filas calculadas teniendo en cuenta la altura del control y la altura del tipo de fuente escogida, número de columnas y su tamaño según la fuente y el alineamiento del texto en las celdas que será centrado.

InitVscroll1

Argumentos: Ninguno.

Descripción: Define el rango de valores entre los que variará la propiedad Value del control al desplazar el cursor y que será acorde con el tamaño de memoria interna definida.

Refrescar

Argumentos: Ninguno.

Descripción: Se encarga de actualizar la presentación de la ventana con la nueva geometría y los nuevos valore de la memoria llamado a la función Form_Resize. El refresco se realiza sólo si está habilitado con la variable RefreshEnable.

Form_Resize

Argumentos: Ninguno.

Descripción: Se llama para refrescar la ventana y cuando sufre un cambio de tamaño.

En primer lugar se comprueba que el formulario no esté maximizado ni minimizado para que no se producía un error. Si el estado de la ventana es normal se definen el tamaño del los controles MSFlexGrid y Vscoll para adecuarse al tamaño de la ventana. Se calcula de nuevo la cantidad de líneas que caben en el nuevo tamaño y se vuelve a variar el tamaño de la ventana para que el número de líneas mostradas sea exacto.

A continuación se modifica el incremento que sufre el valor del control Vscroll para ajustarse a las líneas mostradas y por último se llama a la

función `Vscroll_Change` para mostrar los valores de la memoria. La variable `Dimensionando` evita que esta rutina se ejecute de forma anidada cuando se redefine el tamaño de la ventana.

Vscroll1_Change

Argumentos: Ninguno.

Descripción: Se llama desde `Form_Resize` y cuando se desliza el cursor del scroll. El valor que toma el control corresponde con la dirección de memoria que se mostrará en la celda (1,1).

En primer lugar se comprueban unos límites para que todo el control `MSFlexGrid` esté lleno, es decir, si con el scroll selecciono la última posición de memoria se redefine este valor para que en la casilla (1,1) se muestre las suficientes posiciones memoria anteriores para que el final de la memoria se encuentre en la última fila vista del control.

En un siguiente paso va rellenando las celdas mediante dos bucles anidados, el externo que recorre las filas y el interno que recorre las columnas, siendo la primera la dirección y la última el contenido de la memoria en texto.

MSFlexGrid_DblClick

Argumentos: Ninguno.

Descripción: Gestiona la doble pulsación del ratón.

Si es en la primera columna se pide una nueva dirección mediante el cuadro de diálogo `IBox`. El dato retornado se comprueba para que no sea una cadena vacía lo que indicaría que se ha pulsado el botón cancelar al pedir el dato. El dato retornado será la dirección de memoria que se muestre modificando el valor del control `Vscroll` y llamando indirectamente a la función `VScroll_Change` para mostrar los nuevos datos.

Si es en las columnas de datos se pide un nuevo valor con la misma función `IBox` y se escribe en esa posición de memoria usando la función externa `WriteMemIntInd`. Se refresca la ventana para reflejar el nuevo cambio.

Si es en la última columna se varía la memoria por texto. Se pide un nuevo texto y mediante un bucle se van separando los caracteres uno a uno, se transforman a valor ASCII y se almacena en la posición de memoria correspondiente. Se refresca la ventana para reflejar el nuevo cambio.

Form_KeyPress

Argumentos: KeyAscii As Integer. Devuelve la tecla pulsada.

Descripción: Detecta la pulsación de la tecla Return y llama a la función MSFlexGrid_Db1Click. De este modo la tecla tiene el mismo efecto que la doble pulsación y se puede cambiar la memoria y la dirección visualizada desde el teclado.

MSFlexGrid1_RowColChange

Argumentos: Ninguno.

Descripción: Se llama cuando el cursor cambia de celda. Controla el scroll de la memoria con las teclas de cursor y avance y retroceso página. El control tiene dos líneas ocultas por encima y por debajo de las visibles. Si el cursor llega a la fila 1, es que está en la zona no visible porque he pulsado flecha arriba. Se retrocede entonces el área de memoria vista en una posición cambiando el valor del control Vscroll1. Si el valor mostrado ya es la posición de memoria 0 la pulsación se ignora.

Si se sitúa en la fila 0 es porque he pulsado retroceso página, retrocediendo la memoria vista en tantas posiciones como filas sean visibles. Un método similar se sigue para controlar el desplazamiento hacia abajo del cursor. Las nuevas posiciones se actualizan cambiando el valor del control VScroll, lo que provoca una llamada a VScroll1_change.

La variable RowAnterior se utiliza para devolver el cursor a la fila en la que estaba antes de realizar la pulsación de las teclas de desplazamiento.

ResaltaCadena

Argumentos: Direccion As Long. Contiene la dirección de memoria a resaltar.

Descripción: Se utiliza en la búsqueda de una cadena. Dada una dirección desplaza la memoria a esa posición y la resalta seleccionándola con el cursor.

ActualizaValor

Argumentos: Direccion As Long. Contiene la posición de memoria a actualizar.

Descripción: Esta subrutina cambia el texto mostrado en el control MSFlexGrid1 en la posición de memoria indicada por dirección por el valor actual de esa posición de memoria. Es una actualización de la ventana selectiva.

En primer lugar se realizan unos cálculos para determinar si la posición de memoria que se quiere refrescar es una posición vista. Si no lo es se termina la rutina. Si

lo es se calcula en qué fila y columna se encuentra la posición de memoria escogida y se cambia el texto de esa celda por el valor de memoria leído.

3.2.19 Ventana de Memoria Externa ExtMemForm

Esta ventana tiene las mismas funciones que la ventana de memoria interna y su funcionamiento es similar con la única diferencia de cambiar todas las referencias a la memoria interna por referencias a la memoria externa.

3.2.20 Ventana de Código CodeMemForm

Las funciones empleadas son las mismas que en las ventanas de memoria externa e interna y el funcionamiento similar pero con los siguientes matices:

No se representan todas las posiciones de memoria, sólo aquellas que son inicio de instrucción. De este modo siempre habrá más direcciones que líneas de código. Para relacionar la posición de la línea de código con la dirección de memoria que le corresponde se usan las variables RowtoDir y DirtoRow.

El valor de VScroll no representa posiciones de memoria, sino líneas de código.

En MSFlexGrid1_DblClick sólo se contempla la posibilidad de cambiar la posición vista.

En Vscroll_Change se muestra dirección, opcode y mnemónico.

No existe una versión de la subrutina ActualizaValor.

Se incluye la siguiente subrutina:

ComprobarResaltado

Argumentos: Ninguno.

Descripción: Esta subrutina se encarga de colorear las filas según esté situado el PC o algún breakpoint en ejecución. Si el PC y un breakpoint coinciden se colorea el PC (azul). En caso de ser breakpoint se diferencian dos colores según sea en dirección única o sea un rango de direcciones. Si no es ni uno ni otro se colorea de blanco (ningún color).

3.2.21 Cuadro de Diálogo SFRBitsForm

Se encarga de cambiar bit a bit el valor de un SFR.

Form_Load

Argumentos: Ninguno.

Descripción: Es la primera función que se llama. Se cambia el texto que refleja el significado de cada bit con los valores pasados en la estructura SfrBit que fueron determinados anteriormente por la rutina que llama al cuadro de diálogo. Además se muestra el contenido hexadecimal del SFR que estamos cambiando y tras traducirlo a binario se activan los bits correspondientes.

Text1_Change

Argumentos: Ninguno.

Descripción: Se llama cuando se cambia el valor hexadecimal del registro. Traduce el valor a binario y activa los bits correspondientes.

Text1_KeyPress

Argumentos: KeyAscii As Integer. Valor ASCII de la tecla pulsada.

Descripción: Detecta la pulsación de Enter y si es así llama a Command1_Click (Aceptar). Si es otra tecla y el carácter asociado no es un carácter hexadecimal cambia el valor de la tecla pulsada por el de la tecla shift, de este modo en el recuadro de texto nunca llega a aparecer un carácter no numérico.

Bit_Click

Argumentos: Index As Integer. Devuelve el índice del control sobre el que se ha pulsado.

Descripción: Se llama cuando se cambia el valor de algún bit. Calcula el nuevo valor hexadecimal asignando a cada bit el peso que le corresponde y sumando los valores parciales y lo muestra en el control Text1.

Command2_Click

Argumentos: Ninguno.

Descripción: Botón cancelar. Descarga la ventana.

Command1_Click

Argumentos: Ninguno.

Descripción: Botón aceptar. Descarga la ventana pero antes se almacena el nuevo valor en el campo Hex de la estructura SfrBits.

3.2.22 Ventana de SFR's SFRsForm

Como se puede observar la ventana de SFR's está dividida en varios grupos. Todos los SFR (ventanas de texto) contenidas en un grupo son en realidad un array de controles. Por eso todas las subrutinas de eventos son la misma para cualquiera, debiendo distinguir la propia subrutina mediante una estructura CASE y con ayuda del índice del control de qué elemento del array se trata.

Form_Load

Argumentos: Ninguno.

Descripción: Es la función con la que se carga la ventana. Según el tipo de microcontrolador que tengamos configurado, se habilitarán o deshabilitarán los registros referentes al temporizador 2.

Registro_Click

Argumentos: Index As Integer. Contiene el índice del control Text que se ha pulsado.

Descripción: Esta rutina es similar para todos los registros.

En general cuando el registro es de 8 bit , el valor se cambia mediante la ventana SfrBitsForm. El proceso es definir primero los comentarios que aparecerán para cada bit y el valor del SFR por defecto que se determinará leyendo de la memoria de SFR's. Esto se almacena en la estructura SFRBit y se llama a la ventana SfrBitsForm. A la vuelta se almacena en la memoria de SFR's el nuevo valor y se actualiza el texto mostrado en la ventana SFRsForm.

Si el registro es de 16 bit no hay definición bit a bit, se llama a la ventana Ibox que nos devuelve un valor hexadecimal. Este se almacena en la memoria de SFR's y se actualiza el texto mostrado en la ventana SFRsForm.

En el caso de los puertos hay algún añadido:

PuertoLatch_Change

Argumentos: Index As Integer. Contiene el índice del control Text que se ha pulsado.

Descripción: Hace que todo cambio realizado en el registro se vea reflejado en los pines (salida de datos). Se utiliza cuando el programa simulado escribe en memoria de SFR's en algún puerto.

- En PuertoLatch_Click se añade una línea de código para que después de cambiar el valor se refleje en los pines correspondientes.
- En PuertoPin_Click: El valor de cada uno de los pines sólo se cambia si ese pin está configurado como entrada (valor 1 en el latch). Además en algún pin se detectan transiciones de 0 a 1 que según el pin afectan a los temporizadores (llamando a la función CompruebaTimers para actuar al respecto), o a las interrupciones externas (activando el flag de petición en ese caso).

Para el tratamiento del registro PSW se utilizan las siguientes subrutinas:

Flag_Click

Argumentos: Index As Integer. Contiene el índice del control Text que se ha pulsado.

Descripción: Se llama cuando pulsamos en alguno de los flags. Se calcula el nuevo valor hexadecimal de PSW y se refleja en el texto además de escribirse en la memoria de SFR's.

Pswtext_KeyPress

Argumentos: KeyAscii As Integer. Valor ASCII de la tecla pulsada.

Descripción: Detecta la pulsación de Enter y si es así llama a Pswtext_Change. Si es otra tecla y el carácter asociado no es un carácter hexadecimal cambia el valor de la tecla pulsada por el de la tecla shift, de este modo en el recuadro de texto nunca llega a aparecer un carácter no numérico.

Pswtext_Change

Argumentos: Ninguno.

Descripción: Llamada cuando cambia el texto de PSW. Calcula el valor de los flag y los actualiza. Almacena el nuevo valor en la memoria correspondiente y vuelve a mostrar los registros Ri por si ha cambiado el banco de registros seleccionado.

Existen además otras dos subrutinas.

Refrescar_SFRs

Argumentos: Ninguno.

Descripción: Se encarga de leer todos los valores de la memoria y representarlos en las diferentes casillas siempre que la variable RefreshEnable se verdadera.

ActualizaValor

Argumentos: Direccion as Long. Dirección del SFR a actualizar.

Descripción: Mediante una estructura Case selecciona el registro que debe ser actualizado y escribe su valor en la casilla de texto correspondiente. Es una actualización selectiva.

3.2.23 Ventana de Variables VarForm

Posee un control List y utiliza dos funciones:

Refrescar

Argumentos: Ninguno.

Descripción: Renueva el contenido de la ventana. Comienza borrando todos los elementos de la lista y utiliza la estructura apuntada por Var(x) para extraer la dirección y el tipo de variable. Con esto leo el valor de la memoria correspondiente y creo una línea de texto con la dirección, el tipo y el valor de la variable en diferentes bases. Esta línea la añado a la lista. El proceso se repite para cada variable.

List1_KeyDown

Argumentos: KeyCode As Integer. Contiene el código de la tecla pulsada.

Shift As Integer. Contiene el estado de las teclas shift y control al pulsar la tecla.

Descripción: Borra la variable sobre la que se halla el cursor. Por la posición en la lista se conoce el índice de Var(x) que se ha de borrar. Para ello se desplaza el contenido $\text{Var}(x)=\text{Var}(x+1)$ desde la variable que he de borrar hasta el final del array de variables indicado por NumVar. A continuación se redimensiona el array Var(x) y se disminuye la variable NumVar para adecuarla al nuevo tamaño.

3.2.24 Ventana PresentaForm

Esta ventana no tiene código, es un bitmap de presentación mientras se carga la aplicación.

3.2.25 Ventana Terminalform

Esta ventana no tiene código, se leen y escriben datos en ella desde la función CompruebaSerie definida en Modulo1.

3.2.26 Funciones de Lectura de Memoria

ReadRi

Argumentos: Reg As Integer. Indica el registro que será leído.

Descripción: Lee el valor del registro Ri. Utilizando los bit RS0-RS1 del PSW para calcular el banco de registros y el índice del registro Ri, calcula la dirección de memoria interna que hay que leer. El valor almacenado en esa posición es que se devuelve.

ReadRiBreak

Argumentos: Reg As Integer. Indica el registro que será leído.

Descripción: Esta función es igual a la anterior, pero se comprueba si hay algún breakpoint definido para ese acceso. Se utiliza al leer un registro Ri desde la subrutina EjecutaInstrucción.

ReadSfrMem

Argumentos: Direccion As Long. Indica la dirección del SFR que será leído.

Descripción: Se limita a leer la memoria de SFR si el registro es de 8 bit. En registros de 16 bits lee el byte más significativo, lo pondera con su peso y lo suma al byte menos significativo. Este valor total es el que se devuelve.

ReadIntMem

Argumentos: Direccion As Long. Dirección de la memoria a ser leída.

Descripción: Lee la memoria interna con direccionamiento directo. Si la posición es mayor que H7F lee los SFRs con ReadSfrMem. Si es menor lee el valor de

la memoria interna y comprueba si hay algún breakpoint definido en ese acceso.

ReadIntMemInd

Argumentos: Direccion As Long. Dirección de la memoria a ser leída.

Descripción: Lee la memoria interna con direccionamiento indirecto. Si está dentro de los límites lee la dirección y comprueba si hay algún breakpoint definido en ese acceso.

ReadIntMemPin

Argumentos: Direccion As Long. Dirección de la memoria a ser leída.

Descripción: La función es similar a ReadIntMem, pero si la dirección leída es un puerto lee el pin en vez del registro.

ReadExtMem

Argumentos: Direccion As Long. Dirección de la memoria a ser leída.

Descripción: Lee la memoria externa comprobando si hay breakpoints definidos en esa dirección de memoria.

ReadBit

Argumentos: ByVal Direccion As Long. Contiene la dirección de bit a ser leído.

Descripción: Lee un bit. Para ello transforma en primer lugar la dirección dada en la dirección de memoria donde se encuentra ese bit. Después mediante una instrucción AND se selecciona el bit adecuado de la dirección de memoria calculada y comprueba su valor. La memoria de la que se lee es interna si la dirección del bit es menor de 128 o de SFR's en caso contrario.

ReadBitPin

Argumentos: ByVal Direccion As Long. Contiene la dirección de bit a ser leído.

Descripción: Es una función similar a la anterior pero si la dirección es del bit de un puerto se lee el pin en vez del latch.

3.2.27 Funciones de Escritura en Memoria

WriteRi

Argumentos: Reg As Integer. Indica el registro que será leído.

Valor as Long. Contiene el dato a ser escrito.

Descripción: Escribe el valor en el registro Ri. Utilizando los bit RS0-RS1 del PSW para calcular el banco de registros y el índice del registro Ri, calcula la dirección de memoria interna que hay que escribir. Se refrescan los valores que se muestran en la ventana de variables y el resto de ventanas abiertas.

WriteRiBreak

Argumentos: Reg As Integer. Indica el registro que será leído.

Valor as Long. Contiene el dato a ser escrito.

Descripción: Este Subrutina es igual que la subrutina anterior pero se comprueba si hay definido algún breakpoint para ese acceso.

WriteSfrMem

Argumentos: Direccion As Long. Indica la dirección del SFR que será leído.

Valor as Long. Contiene el dato a ser escrito.

Descripción: Si es un registro de 8 bit escribe directamente en la memoria de SFR's. Si es de 16 extrae el valor de los bytes alto y bajo y los almacena de forma independiente. Por último refresca el valor del SFR modificado.

WriteIntMem

Argumentos: Direccion As Long. Dirección de la memoria a ser leída.

Valor as Long. Contiene el dato a ser escrito.

Descripción: Escritura en memoria interna con direccionamiento directo. Si la dirección es mayor que 128 escribe en los SFRs llamando a WriteSfrMem. Si es menor, en primer lugar comprueba si hay definido algún breakpoint para ese acceso. A continuación escribe el valor en la memoria y actualiza la ventana de variables. Si la dirección es menor que 32 es posible que haya escrito en un Ri. Para comprobarlo calcula el banco activo y en caso de que coincida se llama a WriteRi y se refresca la ventana de SFR's. Por último se refrescan las ventanas de memoria interna.

WriteIntMemInd

Argumentos: Direccion As Long. Dirección de la memoria a ser leída.

Valor as Long. Contiene el dato a ser escrito.

Descripción: Escritura en memoria interna con direccionamiento indirecto. La estructura es similar a WriteIntMem sin tener en cuenta los SFR's. Se escribe en memoria y se procede a refrescar la ventana de variables. Se comprueba si ha sido escritura en un registro Ri y se refrescan las ventanas de memoria interna.

WriteExtMem

Argumentos: Direccion As Long. Dirección de la memoria a ser leída.

Valor as Long. Contiene el dato a ser escrito.

Descripción: Si la dirección se encuentra dentro de la memoria definida procede a la escritura, comprobando antes si hay algún breakpoint definido y refrescando las ventanas de variables y memoria posteriormente.

WriteBit

Argumentos: ByVal Direccion As Long. Contiene la dirección de bit a ser leído.

Valor as Long. Contiene el dato a ser escrito.

Descripción: Al igual que en ReadBit transforma la dirección del bit en la posición de memoria correspondiente. Luego lee esa posición y activa o desactiva el bit escribiéndolo de nuevo en esa posición. La memoria de la que se lee es interna si la dirección del bit es menor de 128 o de SFR's en caso contrario.

3.2.28 Otras Funciones

Optomne

Argumentos: Dato As Integer. Entrega el byte a tratar como parte de un opcode.

Direccion As Long. Informa de la dirección en la que está el dato.

FinOp As Boolean. La subrutina le asigna el valor verdadero cuando termina de procesar un opcode.

Descripción Esta función es la encargada de ir construyendo el mnemónico a partir de los bytes de opcode que le van pasando. El flujo se inicia determinando mediante la variable `More` si es principio de instrucción. En caso de que lo sea se inicia una estructura `case` donde según el primer byte del opcode se añade al campo mnemónico de la estructura `Instrucción` el primer término de éste. Se establece también en la variable `More` cuántos bytes quedan para que finalice la instrucción.

Si `More` determina que no es principio de instrucción, en primer lugar se almacena en el campo `Opcode` de la estructura `Instrucción` correspondiente a la posición de memoria en la que no comienza la instrucción, el valor que la memoria de código tiene en esa dirección. A continuación se establecen varias condiciones. La primera de ellas es si queda un solo byte de instrucción (un término en el mnemónico) y el mnemónico tiene algún separador antes (arg almacena este separador, que suele ser coma). En este caso se añade al campo mnemónico éste separador.

En caso de ser instrucción `ACALL` o `AJMP` la dirección se leyó en la estructura `case` y se almacenó en la variable `Dr` modificada para adecuarse a las reglas que rigen cómo se calcula la dirección de salto. Se cumple entonces la siguiente condición y tras formar la dirección final mediante un `OR` la añade al mnemónico.

Si es un salto o llamada relativa, indicado por la variable `REL`, se calcula la dirección final teniendo en cuenta el `PC` y se añade al mnemónico.

En otro caso se añade el byte tal como nos lo dan. Por último si hay alguna etiqueta que deba ir detrás del término, `arg2` no será cadena vacía y añadiré al mnemónico ésta etiqueta.

Ibox

Argumentos: `LimInf As Long`. Límite inferior del número que se admite.

`LimSup As Long`. Límite superior del número que se admite.

`Prompt As String`. Mensaje que acompaña a la casilla de entrada de datos.

`Optional Title As String`. Título del cuadro de diálogo que pide el dato.

`Optional Default As String`. Valor por defecto que devuelve.

`Optional Xpos As Integer`. Posición horizontal del cuadro de diálogo.

`Optional Ypos As Integer`. Posición vertical del cuadro de diálogo.

Descripción: Abre un cuadro de diálogo para pedir un dato. Utiliza el cuadro de diálogo `IBoxForm` y comienza definiendo sus características (título, etiqueta,...). Se llama a `IBoxForm` para que el usuario introduzca el dato. Cuando

retorna si no se ha introducido un dato se deja el que estaba por defecto. Luego comprueba que el dato está dentro de los límites. Si no lo está emite un error y vuelve a mostrar el cuadro de diálogo repitiéndose el proceso. Por último descarga IBoxForm y devuelve el valor leído.

CalculaFlags

Argumentos: Medio As Long. Resultado de la operación XOR sobre los que se aplica la subrutina.

Suma As Long. Suma (o resta) de los datos sobre los que se aplica la subrutina.

Descripción: Llamada desde EjecutaInstrucción en las instrucciones que modifican los flags OV, C y AC. Esto sucede en instrucciones de suma y resta.

La subrutina por medio de estructuras If refleja las reglas de activación de los flags:

- C se activa si hay acarreo del bit 7 al 8.
- AC se activa si hay acarreo del bit 3 al 4.
- OV se activa si hay acarreo del bit 7 al 8 pero no del 6 al 7 o si no hay acarreo del 7 al 8 pero sí del 6 al 7.

CompruebaInterrupcion

Argumentos: Ninguno.

Descripción: Se inicia una estructura If donde se van comprobando que los flags de petición de interrupción correspondientes están activados. Si es así se llama a la función que genera la interrupción correspondiente. En el caso de las interrupciones externas se comprueba si es activa por flanco o nivel y se detecta si se está pidiendo según el caso. El orden de la comprobación determina las prioridades a la hora de ser atendidas si se producen dos a la vez.

CompruebaBreakpoints

Argumentos: Direccion As Long. Dirección a la que se accede o que se ejecuta.

Tipo As Integer. Tipo del breakpoint a comprobar:

- 0 = Ejecución.
- 1 = Lectura en memoria interna.

- 2 = Escritura en memoria interna.
- 3 = Lectura en memoria externa.
- 4 = Escritura en memoria externa.

Descripción: Según el tipo de breakpoint que queramos detectar (que depende del argumento Tipo) así se ejecuta una parte de código u otra seleccionada por una estructura CASE. En todos los casos se inicia un bucle para ir recorriendo todos los breakpoints definidos. Para cada breakpoint se comprueba que la dirección concuerde con la de inicio o que esté dentro del rango, según el caso. Se comprueba si pertenece a un grupo activo y el tipo de breakpoint que es. Si todo coincide se llama a la rutina Parar_Click definida en MDIForm1 que detiene la ejecución, se emite un mensaje informando de ello y se devuelve el valor verdadero para indicar que se encontraron breakpoints. A continuación se comprueba el alcance, que si es de una vez borra el breakpoint y se refresca la ventana de código para borrar la línea de color sobre la dirección donde se detectó el breakpoint.

CompruebaTimers

Argumentos: ByVal Ciclos As Integer. Ciclos máquina que han transcurrido desde la última vez que se llamó a esta subrutina.

Tran As Integer: En modo contador determina si se ha producido alguna transición en una patilla externa:

- -1 = No hay transición.
- 0 = Transición en patilla T0.
- 1 = Transición en T1.
- 2 = Transición en T2.
- 3 = Transición en T2EXT.

Descripción: Esta función se encarga de actualizar el estado de los contadores - temporizadores cada vez que se ejecuta una instrucción o se habilita el disparo externo de la cuenta por las patillas del microcontrolador.

En el proceso se distinguen varios pasos. El primero de ellos actualiza el temporizador1 cuando el temporizador 0 no está en modo 3. Comienza haciendo una serie de comprobaciones para determinar que el temporizador debe actualizar la cuenta. En caso afirmativo con una estructura CASE separa los modos de funcionamiento, actuando en consecuencia. Si hay un overflow se activa el flag de petición de interrupción correspondiente.

El segundo paso actualiza el temporizador 0 si no está en modo 3. La estructura es la misma que para el paso anterior.

Si el temporizador 0 está en modo 3 se ejecuta la siguiente parte del código. En este modo las interrupciones que pedía el temporizador 1 son ahora manejadas por el temporizador 0. Al cambiar el comportamiento del temporizador 1 se reescribe el código que lo maneja, pero esta vez sin tener en cuenta las interrupciones ni si está habilitado, ya que siempre lo está. Una vez tratado el temporizador 1 se simula el comportamiento del temporizador 0.

La última parte del código se encarga del temporizador 2, y la estructura es la misma, sólo que también se comprueba que el procesador que estamos simulando sea el 8052.

CompruebaSerie

Argumentos: Ciclos As Integer. Ciclos máquina que han transcurrido desde la última vez que se llamó a esta subrutina.

Salida As Boolean. Si es verdadero determina que debe comenzar a salir un dato.

Origen As Integer. Determina desde dónde se llama a la función.

- 0 = Se llama desde la rutina EjecutaInstrucción.
- 1 = Se llama por overflow en el Temporizador 1.
- 2 = Se llama por overflow en el temporizador 2.

Descripción: Es la subrutina que se encarga de simular el puerto serie. Este tiene 4 modos de funcionamiento y así la rutina tiene 4 partes de código.

En todas ellas comienza por comprobar si hay un byte que esta saliendo ayudado de la variable Sal, que será verdadera. Si es así y se determina que debe salir un bit más se tiene en cuenta para que una vez completos todos los bits se escriba el carácter en la ventana del terminal de salida. La variable Sal se activa cuando el argumento de entrada Salida es verdadero y sólo se desactiva cuando termina de salir el byte. El que un byte comience a salir se determina por una escritura en SBUF.

A continuación se determina si se produce la entrada de un bit comprobando que el recuadro de entrada del terminal posee algún carácter. Si es así, cuando se termina de recibir el byte, se almacena su valor en SBUF2 y se hace un eco en la ventana de entrada del terminal.

3.2.29 Listado de Funciones

3.2.29.1 AddVarForm

```
Private Sub Command2_Click()  
Private Sub Command1_Click()  
Private Sub Text1_Change()
```

3.2.29.2 BuscarForm

```
Public Function LeeLimite() As Long  
Private Sub Command1_Click()  
Public Function LeeMemoria(Direccion As Long, Opcion As Integer) As Long  
Private Sub Command2_Click()  
Private Sub Text3_Change()  
Private Sub Text1_Change()  
Private Sub Text2_Change()
```

3.2.29.2 CodeMemForm

```
Public Sub ResaltaCadena(Direccion As Long)  
Private Numlineas As Integer  
Private Sub Form_KeyPress(KeyAscii As Integer)  
Public Sub Form_Load()  
Public Sub InitMSFlexGrid1()  
Public Sub Form_Resize()  
Private Sub MSFlexGrid1_DblClick()  
Private Sub MSFlexGrid1_RowColChange()  
Private Sub MSFlexGrid1_SelChange()  
Public Sub VScroll1_Change()  
Public Sub InitScroll1()  
Public Sub Refrescar()  
Private Sub ComprobarResultado(Direccion As Long, Fila As Integer)
```

3.2.29.3 ConfHwForm

```
Private Sub Check1_Click()  
Private Sub Text2_Change()  
Private Sub Command1_Click()  
Private Sub Command2_Click()  
Private Sub Text1_Change()  
Private Sub Text1_LostFocus()  
Private Sub Form_Load()
```

```
Private Sub Option1_Click()  
Private Sub Option2_Click()
```

3.2.29.4 DefBreakForm

```
Private Sub Command1_Click()  
Public Sub Command3_Click()  
Private Sub Command2_Click()  
Private Sub Command4_Click()  
Private Sub Check1_Click(Index As Integer)  
Private Sub Option1_Click(Index As Integer)  
Private Sub Form_Load()  
Private Sub Refrescar()  
Private Sub Text1_Change()  
Private Sub Option3_Click(Index As Integer)  
Private Sub Text2_Change()
```

3.2.29.5 ExtMemForm

```
Public Sub ResaltaCadena(Direccion As Long)  
Public Sub VScroll1_Change()  
Public Sub Form_Load()  
Private Numlineas As Integer  
Public Sub InitMSFlexGrid1()  
Private Sub MSFlexGrid1_DbClick()  
Private Sub MSFlexGrid1_RowColChange()  
Private Sub ComprobarResultado(Direccion As Long, Fila As Integer)  
Public Sub InitVScroll1()  
Public Sub Refrescar()  
Public Sub Form_Resize()  
Public Sub ActualizaValor(Direccion As Long)
```

3.2.29.6 GenIntForm

```
Public Sub Command1_Click()  
Public Sub Command2_Click()  
Public Sub Command3_Click()  
Public Sub Command4_Click()  
Public Sub Command5_Click()  
Public Sub Command6_Click()  
Public Sub Command7_Click()  
Public Sub Command8_Click()  
Public Sub Command9_Click()
```



```
Private Sub Form_Load()
```

3.2.29.7 IBox

```
Private Sub Command1_Click()
```

```
Private Sub Command2_Click()
```

```
Private Sub Text1_Change()
```

3.2.29.8 IntMemForm

```
Private Numlineas As Integer
```

```
Public Sub ResaltaCadena(Direccion As Long)
```

```
Public Sub Form_Load()
```

```
Public Sub InitMSFlexGrid1()
```

```
Public Sub Form_Resize()
```

```
Private Sub MSFlexGrid1_Db1Click()
```

```
Private Sub MSFlexGrid1_RowColChange()
```

```
Public Sub InitVScroll1()
```

```
Public Sub Refrescar()
```

```
Public Sub ActualizaValor(Direccion As Long)
```

```
Public Sub VScroll1_Change()
```

```
Private Sub ComprobarResultado(Direccion As Long, Fila As Integer)
```

3.2.29.9 SfrBitsForm

```
Private Sub Command2_Click()
```

```
Private Sub Text1_KeyPress(KeyAscii As Integer)
```

```
Private Sub Command1_Click()
```

```
Private Sub Bit_Click(Index As Integer)
```

```
Private Sub Form_Load()
```

```
Private Sub Form_Paint()
```

```
Private Sub Text1_Change()
```

3.2.29.10 SfrsForm

```
Private Sub Flag_Click(Index As Integer)
```

```
Private Sub Interrupt_Click(Index As Integer)
```

```
Private Sub Otros_Click(Index As Integer)
```

```
Private Sub Pswtext_GotFocus()
```

```
Private Sub Pswtext_KeyPress(KeyAscii As Integer)
```

```
Private Sub Pswtext_LostFocus()
```

```
Private Sub PuertoLatch_Change(Index As Integer)
```

```
Public Sub Refrescar_SFRs()
```

```
Private Sub PuertoPin_Click(Index As Integer)
Private Sub Reg_Click(Index As Integer)
Private Sub PuertoLatch_Click(Index As Integer)
Private Sub Form_Load()
Private Sub Serial_Click(Index As Integer)
Private Sub Pswtext_Change()
Public Sub ActualizaValor(Direccion As Long)
Private Sub Timer_Click(Index As Integer)
```

3.2.29.11 VarForm

```
Private Sub List1_KeyDown(KeyCode As Integer, Shift As Integer)
Public Sub Refrescar()
```

3.2.29.12 MdiForm

```
Private Sub AbrirMemExt_Click()
Private Sub AbrirMemInt_Click()
Private Sub Busqueda(Repite As Boolean)
Private Sub AddVariable_Click()
Private Sub BorrarCodeMem_Click()
Private Sub BorrarMemExt_Click()
Private Sub BorrarMemInt_Click()
Private Sub CambiaFuente_Click()
Private Sub buscar_Click()
Private Sub cascada_Click()
Private Sub CurBreak_Click()
Private Sub DefBreak_Click()
Private Sub DefHard_Click()
Public Sub ReInit()
Private Sub Ejecutar_Click()
Private Sub GenInt_Click()
Private Sub IraCursor_Click()
Private Sub InitRowDir()
Private Sub MDIForm_Load()
Private Sub InitVar()
Private Sub InitCodeMem()
Private Sub InitExtMem()
Private Sub InitIntMem()
Private Sub abrir_Click()
Private Sub InitSFR()
Public Sub NMemExt_Click()
Private Sub Abrir_s19(Filename)
```

```
Private Sub Abrir_ihx(Filename)
Private Sub AbrirMemInt_ihx(Filename)
Private Sub AbrirMemExt_ihx(Filename)
Private Sub MDIForm_Unload(Cancel As Integer)
Private Sub ModificarPC_Click()
Private Sub mosaicovertical_Click()
Private Sub mosaicohorizontal_Click()
Private Sub NBusqueda_Click()
Private Sub ordenariconos_Click()
Private Sub Paso_Click()
Public Sub Parar_Click()
Public Sub NMemInt_Click()
Private Sub PasoSaltando_Click()
Public Sub RefrescaVentanas()
Public Sub MuestraCadena(Direccion As Long, Opcion As Integer)
Private Sub Reset_Click()
Private Sub salir_Click()
Private Sub SalirSub_Click()
Private Sub Toolbar1_ButtonClick(ByVal Button As ComctlLib.Button)
Private Sub VentCode_Click()
Private Sub VerSFR_Click()
Private Sub VerVar_Click()
Private Sub VerTerminal_Click()
Public Sub RefrescaVentanas()
```

3.2.29.13 Modulo1

```
Public BreakDetectado As Boolean
Public Function EjecutaInstrucción() As Long
Public Sub CompruebaSerie(Ciclos As Integer, Salida As Boolean, Origen As Integer)
Public Sub OptoMne(Dato As Integer, Direccion As Long, FinOp As Boolean)
Public Function IBox(LimInf As Long, LimSup As Long, Prompt As String, Optional
    Title As String, Optional Default As String, Optional Xpos As Integer, Optional
    Ypos As Integer) As String
Public Function Ibox(LimInf As Long, LimSup As Long, Prompt As String, Optional
    Title As String, Optional Default As String, Optional Xpos As Integer, Optional
    Ypos As Integer) As String
Public Function ReadRi(Reg As Integer) As Long
Public Function ReadRiBreak(Reg As Integer) As Long
Public Sub WriteRi(Reg As Integer, Valor As Long)
Public Sub WriteRiBreak(Reg As Integer, Valor As Long)
Public Sub WriteIntMemInd(Direccion As Long, Valor As Long)
```

```
Public Function ReadIntMemInd(Direccion As Long) As Long
Public Function ReadSfrMem(Direccion As Long) As Long
Public Sub WriteSFRMem(Direccion As Long, Valor As Long)
Public Function ReadIntMem(Direccion As Long) As Long
Public Function ReadIntMemPin(Direccion As Long)
Public Sub WriteIntMem(Direccion As Long, Valor As Long)
Public Function ReadExtMem(Direccion As Long) As Long
Public Sub WriteExtMem(Direccion As Long, Valor As Long)
Public Function ReadBit(ByVal Direccion As Long) As Boolean
Public Function ReadBitPin(ByVal Direccion As Long) As Boolean
Public Function WriteBit(ByVal Direccion As Long, Valor As Boolean)
Public Sub CalculaFlags(Medio As Long, Suma As Long)
Public Function CompruebaBreakpoints(Direccion As Long, Tipo As Integer) As
    Boolean
Public Sub CompruebaInterrupcion()
Public Sub CompruebaTimers(Ciclos As Integer, Tran As Integer)
Public Sub CompruebaSerie(Ciclos As Integer, Salida As Boolean, Origen As Integer)
Private Function CalculaSimbolo(Dato As Integer) As String
Private Function CalculaSimboloBit(Dato As Integer) As String
```

4 PROGRAMAS EJEMPLO

Para comprobar el correcto funcionamiento del ensamblador y del simulador se han desarrollado varios programas de ejemplo descrito a continuación.

4.1 8051.ASM

Este programa se utiliza para comprobar el buen funcionamiento del ensamblador. La simulación no realiza nada.

En el fichero fuente se recoge una muestra de todas las instrucciones que puede procesar el microcontrolador. En la línea de comentario se muestra el código que debe generar el ensamblador.

La compilación debe realizarse con el flag `-l`, de este modo se crea un fichero `.lst` donde se relaciona en la misma línea el código fuente y el código generado por el compilador para esta línea. De esta forma es sencillo comparar el código teórico con el que realmente se ha generado.

4.2 INTERRUPT.ASM

Este programa comprueba el funcionamiento del sistema de interrupciones del microcontrolador 8052 así como sus periféricos y puertos.

El programa principal se limita a iniciar los SFR's y temporizadores para una correcta ejecución del programa, después de lo cual se introduce en un bucle infinito.

Las interrupciones externas se disponen activadas por nivel. El servicio a estas dos interrupciones consiste en muestrear el pin que las genera hasta que lee un valor 1 en la patilla correspondiente. En ese momento se incrementa el valor del registro R0 o R1 según sea interrupción externa 0 o interrupción externa 1 y se devuelve el control al programa principal.

Los temporizadores 0 y 1 se programan en modo 1, es decir temporizadores de 16 bit. En el servicio a la interrupción que se produce cuando se rebasa la capacidad del temporizador, se desactiva la cuenta en primer lugar, y luego se realiza una precarga del temporizador 0 con un valor FF00H y del temporizador 1 con el valor FF80H. Además en las posiciones de memoria interna 30h y 31h se guardan las veces que se ha producido esta interrupción y se envía este valor al puerto 0 en el caso del temporizador 0 y al puerto 2 con el temporizador 1. Acto seguido se activa de nuevo el temporizador.

En el caso del temporizador 2, se programa como contador de 16 bit con captura. Al enviar un flanco descendente en la patilla EXT2 se produce una captura del valor del

temporizador en el registro RCAP2. El servicio a la interrupción es simular al de los temporizadores 1 y 2, pero la precarga se realiza con el valor FE58H.

La prioridad entre las interrupciones se comprueba fácilmente provocando una interrupción externa y manteniendo el pin con un valor bajo durante el tiempo suficiente para que alguno de los temporizadores se desborde. La recarga no se producirá hasta que no se retorne del servicio a la interrupción externa. Del mismo modo si se provocan las dos interrupciones externas al mismo tiempo, el contador de la interrupción externa 1 no aumenta aunque se haya puesto el pin con valor bajo, ya que la interrupción 0 tiene preferencia sobre aquella y no se sirve hasta que no se sirva ésta. Esta situación cambia si se activa el flag de prioridad de la interrupción 1.

La simulación se realizará teniendo cuidado de activar el refresco continuo de los registros y la simulación de los temporizadores. No hace falta activar la simulación del puerto serie ya que no se utiliza.

4.3 COMPCAD.ASM

Este programa verifica el funcionamiento del puerto serie. El algoritmo busca en un texto previamente almacenado en memoria la cadena que se introduzca por el terminal. Una vez concluida la búsqueda el programa emite un mensaje en el terminal informando de si se ha encontrado o no la cadena buscada.

El programa principal habilita las interrupciones del puerto serie y la línea de recepción. Carga los registros con los valores adecuados y comienza un bucle infinito en espera de una interrupción.

El servicio a la interrupción se llama en recepción y en transmisión de un dato. En primer lugar se comprueba que la llamada ha sido por la recepción de un byte, saliendo del servicio en caso contrario. Si se recibe un byte se va almacenando con los anteriores recibidos en la memoria interna hasta que el carácter introducido sea Enter. En ese momento comienza el algoritmo de búsqueda de la cadena introducida en el texto previamente almacenado. Según se haya encontrado o no la cadena, se llama a una subrutina u otra con funcionamiento similar y encargadas de emitir un mensaje informativo por el puerto serie. La escritura de los caracteres del mensaje se separan en el tiempo mediante un bucle para que de tiempo a que los caracteres se emitan correctamente.

El programa asociado COMPCMI.ASM es el mapa de memoria interna que deberá ser cargado antes de la ejecución del programa. En él se almacenan los mensajes informativos de salida y el texto donde se realizará la búsqueda de la cadena. Este programa deberá ser ensamblado y lincado del mismo modo que COMPCAD.ASM pero cargado en el simulador mediante la opción Abrir Memoria Interna.

Para la simulación, únicamente tendrá que estar activa la opción Simular Puerto Serie. El texto a buscar no deberá ser superior a 32 caracteres, ya que de lo contrario se ocuparían posiciones de memoria reservadas a los mensajes de salida corrompiéndolos.

5 BIBLIOGRAFÍA

La mayor parte de la información necesaria se ha consultado en Internet, en las siguientes direcciones:

<http://www.intel.com>

<http://www.8052.com>

<http://cegt201.bradley.edu/~mrr2ro/micrcontrollers.html>

<http://www.tscontrols.com/embedded.html>

http://www.lurpa.ens-cachan.fr/personnels/Denis/8051_faq/8051_faq_1.html

<http://www.ustr.net>

<http://www.pjrc.com/tech/8051/index.html>

Asimismo se han consultado los siguientes libros sobre el tema:

Título: The 8051 microcontroller: Architecture, programming and applications

Autor: Ayala K. J.

Editorial: Ed West Publishin Company, Minneapolis, 1997

Título: Visual Basic 5

Autor: Antonia González Mangas

Editorial: Paraninfo, Madrid, 1998