APÉNDICES

APÉNDICE A

INICIACIÓN DE UN PROYECTO CON ESTRUCTURAS DE CÓDIGO REUSABLES EN LENGUAJES ENSAMBLADOR Ó C

A.1 LA NECESIDAD DE CÓDIGOS REUSABLES

Durante el transcurso de mi tesis, me he dado cuenta que en cada ocasión que necesite hacer un proyecto, hay cosas comunes que uso frecuentemente (hablando por el momento de inicializar los módulos) p.e., generar un temporizado y luego de vencido el tiempo activar un evento, leer una variable física (p.e. temperatura), saber el estado de las entradas, etc.

Los grandes programadores llegan rápidamente a la solución de su problema en corto tiempo pues probablemente tengan una colección de ***"rutinas utilitarias"*** de uso frecuente que les simplifique su desarrollo. Del mismo modo, el concepto aplica para industrias. Imagínese que cada vez que una compañía automotriz quiera lanzar al mercado un nuevo modelo o mejorado lo hiciese desde el principio.

Por ende, ¿qué es más importante? ¿reinventar lo que ya está hecho? ó ¿llegar más rápido a hacer que las cosas funcionen?. El propósito de los siguientes apéndices es explicar el uso de las rutinas utilitarias de microcontroladores (en especial JK3/JL3/QT y GP32) y así ahorrarse tiempo de desarrollo, llegando más rápido a la aplicación.

Más importante aún, nada de esto sería posible si no se ***"estandarizara***" el código. Ejemplo claro es el relato típico de la Torre de Babel; esta nunca se pudo llegar a terminar pues las personas colaboradoras hablaban diferentes idiomas lo que trajo una confusión de ideas y el caos.

La estandarización es importante si se trabaja en equipo, pues facilita la comprensión del proyecto y del mismo modo agiliza el tiempo de desarrollo. Los códigos presentados, están en un 95% realizados en ambos lenguajes C y ensamblador para microcontroladores de la familia HC08.

Nota: Si en las siguientes secciones A.1.1 a A.1.5 no se está claro en el procedimiento de iniciar un proyecto en lenguajes C o ASM con rutinas reusables, revisar el archivo ***NT0026_-_Modules_11_09_04.zip*** **el cual contiene los archivos:**

ComoDebeVerseUnProyectoDeWinIDEconJL3.zip

ComoDebeVerseUnProyectoDeCW3.xconJL3.zip

EjemploDeProyectoCW3.xconGP32.zip

A.1.1 División del Archivo de Módulos

Los módulos están divididos en el archivo comprimido ***NT0026_-_Modules_13_12_04.zip*** para cada microcontrolador correspondiente y para cada lenguaje (C y Ensamblador). La manera de cómo están divididos se muestra en la figura A.1. *Para las últimas actualizaciones, ver el archivo "zip"*.

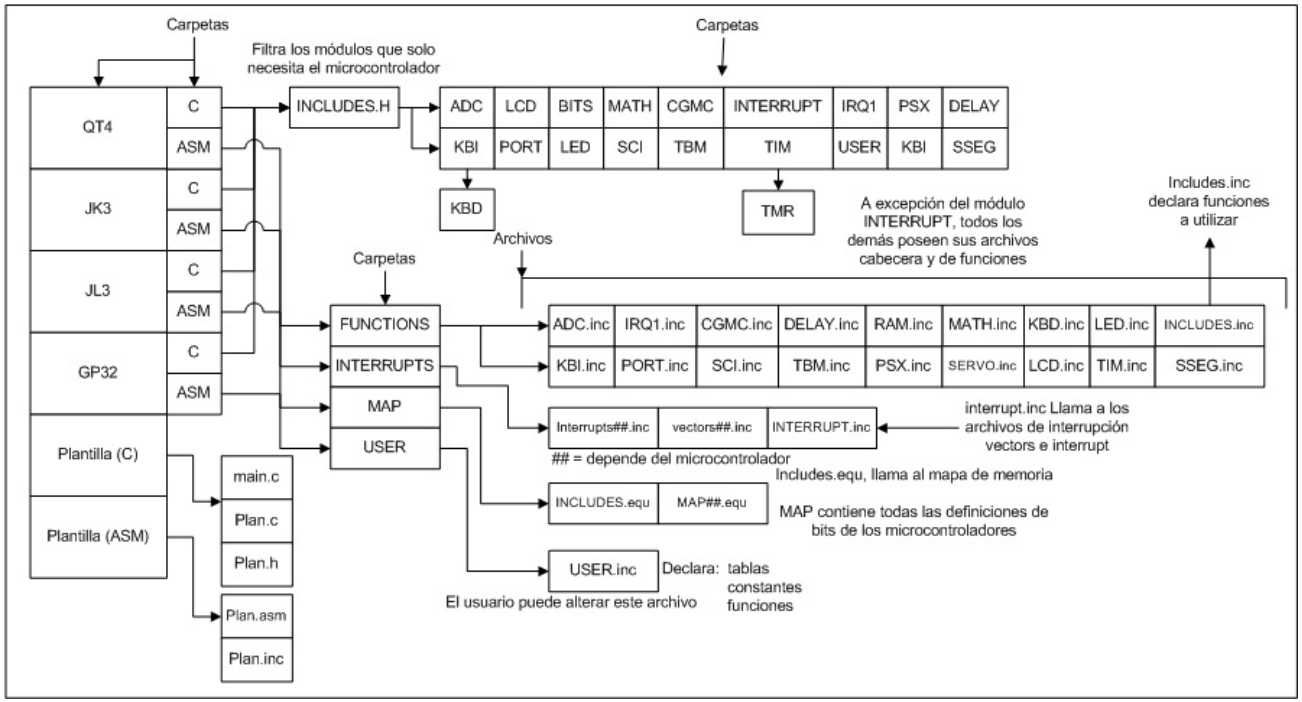

Figura A.1. Descripción del Archivo "zip" con las Rutinas Reusables

A.1.2 Plantillas

Para iniciar un proyecto en lenguaje C refiérase a la NT1003 – “CodeWarrior”; y en ensamblador, ver el documento NT0006 – “Software”.

Lenguaje C

Arroje *todo el contenido del microcontrolador correspondiente* a su carpeta de proyectos (SOURCES) y reemplace el contenido de su archivo *main.c* por el contenido de *main.c* que se encuentra en el archivo zip NT0026/Plantilla C/

```
/*
**************************************************************************
*            UNIVERSIDAD TECNOLÓGICA DE PANAMÁ
*               PANAMÁ, REPÚBLICA DE PANAMÁ
*
*  Archivo             : main.c
*  Programador(es) :
*  Lenguaje            : ANSI-C
*  Descripción         :
*  ---------------------------------------------------------------------------
*  HISTORIAL
*  DD MM AA
*  ##  ##   ##          Creado.
*  ##  ##   ##          Modificado.
**************************************************************************
*/
/*
**************************************************************************
*                        ARCHIVOS CABECERA
**************************************************************************
*/
#include "includes.h"   /* Incluye archivos cabecera a usar */
/*
**************************************************************************
*                        PROGRAMA PRINCIPAL
**************************************************************************
*/
void main(void)

{                           /* Programa Principal                  */
/* El código aquí */
}
```

Lenguaje Ensamblador

Arroje todo el contenido del microcontrolador correspondiente a su carpeta de proyectos y copie el encabezado del archivo por el contenido de *Plan.asm* que se encuentra en el archivo zip NT0026/Plantilla ASM/

```
;====================================
; ARCHIVO          :
; PROPÓSITO        :
; NOTA             :
;
; REFERENCIA       :
;
; LENGUAJE         :  IN-LINE ASSEMBLER
; ---------------------------------
; HISTORIAL
;   DD MM AA
;   ## ## ##          Creado.
;   ## ## ##          Modificado.
;====================================

;====================================
;   Cabecera de Macros, Const. y Memoria
;====================================
$include '\MAP\includes.equ'

;====================================
; OBJETIVO    :  Inicio de Codif. del Ensam-
;                blador en Memoria FLASH.
;====================================
        org FLASH_START

;==================================
; OBJETIVO    :
;==================================
START
            ; El código aquí

;==================================
   Declaración y definición de funciones
;==================================
$include '\FUNCTIONS\includes.inc'

;==================================
;Declaración y Definición de interrupciones
;==================================
$include '\INTERRUPTS\interrupt.inc'
```

Listado A.1 (izquierda) y A.2 (superior). Apariencia Principal de las Plantillas en C y en Ensamblador. Si está utilizando una plantilla en lenguaje C puede empezar a escribir código directamente. Si desea ver ejemplos de utilidad, ver los contenidos de las notas técnicas NT0008 o superiores. Para una plantilla en ensamblador requiere las inicializaciones de los listados A.1.3, A.1.4 y A.1.5.

A.1.3 Mapa de Memoria y Bits

Exceptuando el lenguaje C, personalmente, el autor se encargó de declarar el mapa de memoria del microcontrolador correspondiente y sus grupos de bits para cada microcontrolador. Para incluir archivos en su proyecto, lenguaje C ver NT1003 – “CodeWarrior”.

Lenguaje C	Lenguaje Ensamblador
No se necesitan declarar los registros, existe un archivo tipo estructura *MC68HC908##.h* y *MC68HC908##.c* que se encargan de definir los registros. Adicionalmente la carpeta BITS del archivo “zip” NT0000 - Modules, contiene los bits en estado ON de todos los microcontroladores.	Si descargó el contenido de la carpeta del microcontrolador correspondiente, no habrán problemas al momento de compilar. Adicionalmente la carpeta MAP contiene los archivos “equates” que proveen los bits ON de todos los microcontroladores.

A.1.4 Inclusión de Funciones

Lenguaje C

Para incluir los archivos de las correspondientes funciones siga los pasos de la NT1003 – “CodeWarrior”.

Lenguaje Ensamblador

Abrir la carpeta FUNCTIONS e incluir en el archivo INCLUDES.inc las funciones de usuario, del sistema y rutinas utilitarias a usar. El listado A.3 muestra un ejemplo.

Listado A.3. Inclusión de Funciones. Incluir las funciones de usuario, sistema y rutinas de uso frecuente, describiendo primero el fólder en el cual se encuentran y luego el nombre con su respectiva extensión.

```
;===========================================
;                Funciones de Usuario
;===========================================
$include '\USER\USER.inc'

;===========================================
;        Funciones de Inicialización del Sistema
;===========================================
$include '\FUNCTIONS\ADC.inc'

;===========================================
;                Rutinas Utilitarias
;===========================================
$include '\FUNCTIONS\DELAY.inc'
```

A.1.5 Interrupciones

Del mismo modo que los grupos de bits, el autor también se encargó de realizar la lista de interrupciones para que el usuario solamente se especialice en escribir código y se facilite la aplicación.

Lenguaje C

Para escribir una interrupción abrir la carpeta INTERRUPT y seguido el archivo *INTERRUPT.c*, para luego remover los comentarios y escribir el código de las interrupciones.

Lenguaje Ensamblador

Para escribir una interrupción abrir la carpeta INTERRUPTS y los archivos *vectors.inc* e *interrupts.inc*; inhabilitar los comentarios de la interrupción (*vectors.inc*) correspondiente y escribir el código de las interrupción (*interrupts.inc*).

```
interrupt 16 void ADCH(void){
// El código aquí
}


/*
interrupt 15 void KBIH(void){
// El código aquí
}
*/

/*
interrupt 14 void SCITXH(void){
// El código aquí
}
*/
```

(a)

```
;===================================
;  Interrupción de Convertidor Análogo Digital
;===================================
ADCINTL         ; Conversión Completa (Bajo);
"El código aquí"
        rti     ; Retorna

;===================================
;       Interrupción del Módulo de Teclado
;===================================
;KBINTL          ; Teclado (Bajo)
; "El código aquí"
;       rti     ; Retorna

;===================================
;    Interrupción de Transmisión del SCI
;===================================
;SCITXL          ; Transmisión del SCI (Bajo)
; "El código aquí"
;       rti     ; Retorna
```

(b)

```
;================================
;    Vector de Convertidor Análogo Digital
;================================
        org ADCINTH
        dw ADCINTL

;================================
;       Vector del Módulo de Teclado
;================================
;       org KBINTH
;       dw KBINTL

;================================
;       Vector de Transmisión del SCI
;================================
;       org SCITXH
;       dw SCITXL
```

Listado A.4 (izquierda), A.5 (derecha). Declaración y definición de Interrupciones. El listado A.4 muestra como habilitar interrupciones en C solamente removiendo los comentarios. Por otro lado, el listado A.5(a), muestra en ensamblador como definir el código de la interrupción solo removiendo los comentarios, y el listado A.5(b), es necesario para definir donde está ubicada la interrupción, igualmente se deben remover los comentarios.

A.2 Introducción a Código Reusable

Lo siguiente apéndices son códigos reciclables, o más bien, el llamado de los códigos reciclables dispuestos por el autor para facilitar la inicialización de los módulos del microcontrolador, estos códigos están hechos para funcionar tanto en ensamblador como en lenguaje C, en otras palabras, cada microcontrolador tiene desarrollado el mismo código en ambos lenguajes para cualquier usuario que programe en cualquier plataforma de desarrollo (WinIDE o “CodeWarrior”).

Cada código función de código reciclable inicia con su prefijo del módulo, por ejemplo ADCInit() inicializará el módulo ADC, inicialmente se explicará su utilidad en lenguaje C en su columna izquierda y luego su equivalente en ensamblador en la columna derecha.

Si los códigos correspondientes no aparecen listados en la plataforma de su agrado, ya sea C o ensamblador, simboliza que estos no han sido realizados aún, pero no significa que no se pueda realizar. Los códigos reusables muestran una manera eficiente de estructurar un programa, no necesariamente es el código óptimo de programación, significa que estos se pueden mejorar.

Se ha estudiado solo la posibilidad de código en lenguaje C, pues es mucho más sencillo de entender al lenguaje C++ y también, porque la orientación de los cursos de programación en las escuelas y universidades se orienta hacia lenguaje C.

A.3 Referencia Rápida de Programación en C

En los siguientes puntos de referencia rápida se dará un pequeño pantallazo a la programación en C. La gran versatilidad que posee C sobre la memoria, como el manejo de punteros, hace que este sea el lenguaje preferido de los programadores; además, el aumento de memoria en pastilla de los microcontroladores hoy en día, hace que no sea tan traumático implementar una aplicación en dicho lenguaje sin necesidad de mal gasto de la memoria.

Mucha es la mística sobre este lenguaje, lo difícil que resulta tener que programar en alto nivel en la PC; por extensas que sean las librerías de funciones que contenga el programa C, son poco utilizadas para microcontroladores, a raras excepciones, dependiendo de la complejidad del proyecto. Para mayores referencias al respecto, consultar libros como:

Horton, Ivon. *Beginning Visual C++ 6*. Indianapolis, Indiana: Wiley Publishing Inc., 2003.

Labrosse, Jean J. *Embedded Systems Building Blocks*, Complete and Ready-to-Use Modules in C. 600 Harrison Street, San Francisco: CMP Books, 2002.

Dichos libros explican con ejemplos orientados a lenguaje C a nivel de PC y microprocesadores, pero son una gran mina de referencia de información a estandarización de código.

A.3.1 Estructura de un Programa

Un programa en lenguaje C, para microcontroladores, puede adquirir la siguiente composición:

```
1     #include “includes.h”
2     #include <MC68HC908JL3.h>

3     char c;

4     void main(void) { /* Inicia el programa */

5        DDRA = 0xFF;
6        PTA = 0x00;
7        PTA = 0xFF;

8     }
```

Listado A.6. Estructura de un Programa en C

Las líneas [1] y [2] llaman a los archivos cabecera o librerías del programa, la diferencia entre comillas es que llama a la librería desde la carpeta del proyecto, que por lo general, para “CodeWarrior” es la carpeta “Sources”; mientras tanto, la línea 2 llama desde los archivos librerías del IDE. Las líneas [4] y [8] representan la función principal donde se escribirá el código del programa, nótese los corchetes inicial y final de la función. La línea [3] representa una variable, de tipo global, pues está fuera de la sección de la líneas [4] y [8], de lo contrario, si se encuentran dentro, representan una variable local.

Las líneas [5] a [7] corresponden al código del programa.
Como se habrá notado, ciertas líneas poseen un “`;`” al final de su sentencia y se debe a que el lenguaje, cuando no es una palabra reservada, inclusión o cierre de llaves, necesita de ella para reconocer entre la siguiente sentencia de código.
Al final de la línea [4] se lleva parte de ella entre “`/* */`”, cualquier texto escrito entre ellas es un comentario y no forma parte del código de programa, solo se utiliza para esclarecer el código generado.

A.3.2 Declaración de Registros

Las líneas [5] a [7], extremo izquierdo, del listado A.6, corresponden al puerto del microcontrolador, pero la misma está declarada implícitamente en la librería [2].

```
1    #define PTA (*(volatile unsigned char *) 0x0000
2    #define volatile unsigned char DDRA @0x0004
```

Listado A.7. Definición de Registros en C

Las líneas [1] y [2] hacen lo mismo, definen los registros para el microcontrolador, son distintas formas de llamar a un registro, la única diferencia es la carencia de portabilidad de la línea [2] hacia otro compilador, es decir, la línea [2] no es código C puro.

A.3.3 Tipo de Variables

La línea [3] del listado A.6, representa una variable, pero igualmente, los registros son variables. Debido a que obviamente el sistema es de 8 bits, se trabaja con variables char aunque también soporta mayor cantidad de bits, pero la operación de proceso del número es más lenta.

Tabla A.1. Tipo de Variables y Precisión de Bits

Tipo	Rango en Decimal	Presición en Bits
unsigned char	0 a 255	8
char	-127 a 127	8
unsigned int	0 a 65535	16
int	-32767 a 32767	16
unsigned short	0 a 65535	16
short	-32767 a 32767	16
unsigned long	0 a 4294967295	32
long	-2147483647 a 2147483647	32

A.3.4 Números, Caracteres y Cadena de Caracteres

Todo valor que pueda manejar el C se le llama literal. Un literal consiste en números, caracteres y cadenas de caracteres.

Nota sobre las variables: El lenguaje C es sensitivo a como se nombran de las variables, p.e. una variable `Var`, será diferente a otra llamada `var`.

Los números se pueden representar, según la sección 1.2, en binario, octal decimal y hexadecimal. Como el sistema octal no es muy utilizado, pasaremos por alto su nomenclatura.

```
1     void main(void) {
2        unsigned char caracter;    /* variable 1 */
3        unsigned char cadena[7];   /* variable 2 */
4        DDRA = 0xFF;          /* Hexadecimal    */
5        PTA  = 0b00000000;    /* Binario        */
6        PTA  = 255;           /* Decimal        */
7        caracter = '9';       /* Caracter       */
8        cadena   = "Isaias"   /* Cadena         */
9     }
```

Listado A.8. Asignación de Literales a Variables

Las líneas [2] y [3] declaran un caracter y una cadena, por el momento obvie los paréntesis al final de la palabra cadena.

Líneas [4] a [6]. Para los números hexadecimal, se utiliza el prefijo `0x`, binario `0b` y el decimal carece de prefijo. Para ver la nomenclatura de sistemas numéricos, referirse a la sección 1.2.

Líneas [7] y [8]. Un carácter es un literal entre apóstrofes, y no es lo mismo `'9'`, que `9`. El apóstrofe representa que es un carácter ASCII mientras que sin paréntesis es un número decimal. Ver http://www.asciitable.com/. Una cadena es un conjunto de caracteres, pero de terminación nula, es decir: `'I'`, `'s'`, `'a'`, `'i'`, `'a'`, `'s'`, `0`.

A.3.5 Secuencias de Escape

A veces, es necesario hacer una interfase con la PC y se necesitan saber ciertas condiciones especiales como las secuencias de escape como el salto de línea, un beep o escribir una diagonal entre una cadena. Estas se diferencian porque están escritas entre diagonales.

Tabla A.2. Secuencias de Escape Comunes

Secuencia de Escape	Acción
\a	Sonido "beep"
\n	Salto de Línea
\'	Imprime Apóstrofe
\\	Imprime Backslash
\b	Retorna una posición
\"	Imprime comillas
\?	Imprime Caracter de Interrogación
\r	Retorno de Carro
\v	Espaciado Vertical
\t	Tabulado

A.3.6 Constantes y Variables

Una *constante* es un elemento que reside en la memoria, generalmente en la memoria ROM, nuestro caso particular es la FLASH, esto significa que no podrá ser modificada en tiempo de ejecución y solamente puede ser leída. Mientras tanto, una *variable*, se almacena en la memoria RAM y puede ser leída o escrita.

Utilizando la tabla A.1, observamos que existen diferentes tipos de variables. En la siguiente página se muestra como declarar una variable y asignarle un valor, al igual que a una constante.

A.3.6.1 Asignación de Valores Iniciales de Variables

Cuando declara una variable, esta puede tener valores iniciales, y las acepta siempre y cuando esté en el rango de las variables a utilizar:

```
unsigned char A = 20;
char   B = -3;
```

En este caso, cada variable toma un valor en específico, inicial, especificado por el programador.

A.3.6.2 Definición de su Propio Tipo de Variables

Resulta bastante tedioso escribir siempre el mismo tipo de variables, lo mejor es tener un especificador que represente que tipo de variable se está trabajando. La sentencia a utilizar es "`typedef`".

```
1     typedef unsigned char INT8U;
2     typedef          char INT8S;

3     void main(void) {
4         INT8S ConSigno;
5         INT8U SinSigno;

7         ConSigno = -127;
8         SinSigno =  221;
11    }
```

Listado A.9. Declaración de Variables y Constantes

Las líneas [1] y [2] definen nuestro propio tipo de datos, de esta forma, no tendremos que especificar siempre `unsigned char` o `char`, llamaremos solo al tipo de dato que represente nuestra variable o constante.

A.3.6.3 Declaración de Variables y Constantes

Para declarar variables ayúdese del siguiente código:

```
1     unsigned char A;
2     int D;
3     const unsigned int Offset = 900;

4     void main(void) {
5         char B;
6         unsigned int C;

7         A = 20;
8         B = -3;
9         C = 160*A;
10        D = 300*B - Offset;
11    }
```

Listado A.10. Declaración de Variables y Constantes

Las variables de las líneas [1] y 2, son variables globales, mientras que las lineas [5] y [6] son declaración de variables locales. Las líneas [7] a [10] son asignaciones de las variables, como observa, las variables tipo "`unsigned`" solo admiten números positivos o iguales a `0`.

La línea [3] ejemplifica como declarar una constante en memoria (FLASH), es decir que esta se aloja y no puede ser alterada, solo leída.

Sugerencias para trabajo con variables y constantes en el microcontrolador:

- Seleccione variables de tipo "`char`" para trabajar para agilizar el procesamiento.
- Trabaje variables tipo "`int`" solo si el registro es de 16 bits
- No ejecute operaciones de punto flotante, esto hace que el procesado matemático sea ineficiente, busque siempre trabajar con enteros.

A.3.7 Operadores

Los operadores, como lo indica su nombre, se utilizan para realizar operaciones entre variables para desempeñar un cálculo que representará una parte del proceso del programa o un resultado. Los operadores más comunes son los operadores aritméticos binarios como adición, sustracción, multiplicación y división.

Tabla A.3. Operadores Binarios Aritméticos

Operadores	Descripción
+	Suma
-	Resta
*	Multiplicación
/	División
%	Resto/Residuo
<<	Corrimiento a Izquierda
>>	Corrimiento a la Derecha

Siempre es necesario modificar variables, el siguiente ejemplo muestra lo versátil en código que es C al realizar una operación aritmética:

```
1     cuenta   = cuenta + 5;
2     cuenta  += 5;
3     B        = B % C;
4     B       %= C;
5     B        = A << 5;
```

Ambas líneas [1] y [2]; [3] y [4] describen lo mismo, pero las líneas pares demuestran una forma más elegante y fácil de entender. En ciertas ocasiones es necesario buscar el residuo de una operación, las líneas [3] y [4] muestran como se puede realizar dicha operación.

La línea [5] muestra como se debe acuñar una operación de multiplicación binaria, si `A = 3`, esta es lo mismo que 2^5 `x 3 = 96`.

Los operadores unarios son aquellos que toman una simple entrada y nos dan una simple salida. La tabla A.4 muestra algunos de ellos.

Tabla A.4. Operadores Unarios

Operadores	Descripción
~	Complemento (1's por 0's)
!	1 por 0 y otros números por 0
&	Dirección de
-	Negativo
+	Positivo
++	Incremento
--	Decremento
*	Referencia

Despreocupándonos de ciertos operadores avanzados por el momento, concentrémonos en el siguiente ejemplo:

```
1     void main(void) {
2         char a, b, c;
3         char d = 3;

4         a = 5;
5         c = 0;
6         c++;
7         b = 1 + a++;
8         c = --a;
9     }
```

Listado A.11. Operadores Unarios

En la línea [6], `c = 1`; mientras que `b = 6` en la línea [7] y a finaliza con un valor de 6, debido a que primero se ejecuta la operación y luego se incrementa el valor (postincremento), finalmente, en la línea [8] a vuelve a retomar el valor de 5 (predecremento).

Otros operadores binarios aritméticos muy utilizados en microcontroladores, son los operadores binarios lógicos, que realizan

Tabla A.5. Operadores de Bits y Booleanos

Operadores	Descripción
De Bits	Dos valores binarios dan un resultado
&	AND, si el bit de ambas variables es 1 resulta un bit 1 en esa posición binaria
|	OR, si cualquier bit en ambas variables es 1 resulta un 1 en esa posición binaria
^	XOR, si en la posición del bit de ambas variables coincide en 1, resulta un 0
Booleanos	Do valores booleanos dan un resultado
&&	Condición 1 si ambas son 1
||	Cualquier condición es 1 si alguna es 1

Despreocupándonos de los operadores booleanos por el momento, concentrémonos en el siguiente ejemplo:

```
1     void main(void) {
2         char a, b, c;

3         a = 0b11110001
4         b = 0b10001111
5         c = a & b;
6         c = a | b;
7         c = a ^ b

8     }
```

Listado A.12. Trabajando con Operadores de Bits

El resultado de la línea [5] es 0b10000001, solo los bits extremos son 1, por otro lado, la línea [6] es 0b11111111, pues alguno es 1 en la posición, finalmente, en la línea [7] 0b01111110, los bits que difieren entre a y b son iguales a 1.

Algunas operaciones a nivel de bits muy comunes en lenguaje C se traducen a una simple línea en ensamblador, activar un bit, apagar un bit, conmutar un bit. Concentrémonos en el siguiente ejemplo:

```
1     void main(void) {

2         DDRA = 0xFF;  /* Todos salidas       */
3         PTA  = 0;     /* Nivel Bajo          */
4         PTA |= 0xF0;  /* Bit[4:7] encendidos */
5         PTA &= ~0xF0; /* Bit[4:7] apagados   */
6         PTA ^= 0x0F;  /* Bit[0:3] conmutados */
7         PTA ^= 0x0F;  /* Bit[0:3] conmutados */

8     }
```

Listado A.13. Realizando Operaciones de Manejo de Bits

La línea [2] y [3] inicializan el puerto, siguiendo la línea [4] enciende los bits del 4 al 7, por otro lado la línea [5] apaga los bits que se encendieron y finalmente las líneas [6] y [7] cambias de estado el puerto, bits 0 a 3, respectivamente entre estados de 1 y 0 cada bit.

Los operadores tienen cierto orden de precedencia, esta jerarquía es obedecida por el compilador, la precedencia se basa en la asociatividad de las variables, siendo la de mayor precedencia la que se ejecutará prioritariamente.

Para verificar la precedencia refiérase a la tabla A.6 de la siguiente página en donde muestra los operadores más utilizados del lenguaje y de un ejemplo del uso de precedencia para calcular una operación aritmética frecuente con su descripción.

Tabla A.6. Jerarquía de los Operadores

Precedencia	Operador	Orden Asociativo
La más alta	() [] -> .	Izquierdo
	! ~ +(unario) –(unario) ++ -- &(unario)	Derecho
	*(unario)	Izquierdo
	* / %	
	+ -	
	<< >>	
	< <= > >=	
	== !=	
	&	
	^	
	|	
	&&	
	||	
	? :	Derecho
	= += -= *= /= %= <<= >>= |= &= ^=	
La más baja	,	Izquierdo

Tratando de escribir una operación que transforme de pies a metros, se escribiría de la siguiente manera:

```
m = 100*pies/328;
```

Olvidándonos un poco de la precedencia, resulta engañosa la apreciación de la ecuación si no sabemos quien precede a quien. Como estamos trabajando con enteros, lo recomendable es multiplicar primero, porque al dividir `pies/328`, solo será divisible por sus múltiplos y truncará el resultado. Lo propio sería mejor multiplicar y luego dividir, el problema se soluciona siempre añadiendo paréntesis en las secciones inentendibles

```
m = (100*pies)/328;
```

Ahora, la ecuación está más acorde y legible al resultado que esperamos y arrojará el resultado correcto.

A.3.8 Condiciones de Control de Flujo

Las estructuras de control, son aquellas que nos permiten tomar decisiones de las partes de un programa, nos hacen desviar la secuencia de un programa y dar la sensación que la aplicación responde al ambiente de forma inteligente.

A.3.8.1 Comparación de Valores

La única manera de que un programa tome decisiones es comparar cosas, las comparaciones numéricas son las que al final se realizan en un programa. La tabla A.7 Muestra los operadores de comparación:

Tabla A.7. Operadores de Comparación

Operador	Descripción	Operador	Descripción
<	Menor o Igual	<=	Menor o Igual a
>	Mayor o Igual	>=	Mayor o Igual a
==	Igual a	!=	Diferente a

Los operadores de comparación arrojan un valor cierto o `true` o `1`, si la comparación es verdadera, de lo contrario, arrojan un resultado `false` ó `0` si la comparación es falsa. Vea el siguiente ejemplo:

```
char a = 1, b = 3, c = -5

a < b       b == (a+2)  c > (a + b)
```

La primera y segunda comparaciones son ciertas, pero la última es falsa, con esto podremos realizar comparaciones y hacer que con sentencias de control, el flujo del programa se desvíe.

A.3.8.2 Sentencia if

La sentencia ”if” solo se ejecutará si la condición es cierta, la forma común de representar a esta sentencia es:

```
if (condición) {
    /* Hacer algo */
}
    /* Seguir     */
```

También podemos anidar sentencias de este tipo, es decir, ”`if`” dentro de ”`if`”, pero es recomendable no tener excesivas profundidades de anidamiento, pues el programa se torna difícil de entender. Veamos el siguiente ejemplo de anidamiento y uso de la sentencia.

```
1     void main(void) {
2         char a = 1, b = 2, c = -5;

3         if (a < b) {
4             if (a > c) {
5                 a++;
6                 b = c + 9;
7             }
8             if (a == b)
9                 a = 0;
10        }

11    }
```

Listado A.14. Trabajando con la Sentencia if

La línea [3] y [10] corresponden a la primera sentencia, mientras que las líneas [4] y [7] corresponden a el contenido de la segunda, y la línea [8] y [9] corresponden al tercero; como vemos, se cumple la condición de las líneas [3], el programa pasa a la línea [4] y la condición se cumple nuevamente, así, pasa a la línea [5], [6], [8], evalúa la expresión, pero vemos que no se cumple, así que pasa por alto la línea [9].

A.3.8.3 Sentencia if Extendida

La sentencia ”if” posee otra variante muy útil, la cual se representa como sigue:

```
if (condición) {
    /* Hacer algo      */
} else {
    /* Hacer otra cosa  */
}
    /* Seguir     */
```

Como en la sub sección anterior, la sentencia ”if” se ejecuta si la condición es cierta, pero si resulta falsa, pasará a evaluar el contenido de la sentencia else. Igualmente, se pueden anidar sentencias ”if”

```
1     void main(void) {
2         char a = 1, b = 2, c = -5;

3         if (a > b) {
4             c = 0;
5         } else {
6             c = 1;
7         }

8     }
```

Listado A.15. Trabajando con la Sentencia if else

Como la línea [3] no se llega a cumplir, no ejecuta el contenido de la sentencia ”if”, sino que pasa directamente al contenido de la sentencia ”else”, asignando así un nuevo valor a la variable ”c”.

A.3.8.4 Sentencia switch

La sentencia ”switch” se utiliza para casos en donde la solución sea de respuesta múltiple, es igual que tener múltiples sentencias ”if” evaluando la misma condición:

```
switch(condición) {
    case 0:
        /* Hacer algo   */
        break;
    .
    .
    .
    case n:
        /* Hacer algo   */
        break;
    default:                    /* Default        */
        /* Hacer algo   */     /* es una parte   */
        break;                  /* opcional,      */
}                               /* no es necesaria*/
        /* Seguir       */     /* incluirla      */
```

Si la condición en la sentencia es cierta, pasará a decidir que caso evaluar, pero si el caso no está en el rango ejecutará la opción default, la cual es opcional y puede o no incluirse en la sentencia ”switch”.

```
1     void main(void) {
2         char valor= ’2’;

3         switch(valor) {
4             case ’0’:
5                 break;
6             case ’1’:
7             case ’2’:
8                 valor += 2;
9                 break;

10        }
11    }
```

Listado A.16. Trabajando con la Sentencia switch

Del listado A.14, la condición a ejecutar es el caso dos, el cual le suma un valor de dos, dando como resultado un carácter ’4’ y sale de la sentencia.

A.3.8.5 Sentencia while

La sentencia ”while” es un simple lazo cerrado, la misma se ejecuta y seguirá ejecutándose solo si la condición es cierta la forma general es:

```
while (condición) {
    /* Hacer algo */
}
    /* Seguir     */
```

El siguiente programa ejemplifica el uso de la sentencia while.

```
1     void main(void) {
2         char timer = 0;

3         while (1) {
4             while (timer < 10)
5                 timer++;
6             timer = 0;
7         }

8     }
```

Listado A.17. Trabajando con la Sentencia while

La línea [3] corresponde a un lazo infinito, así, todo lo que esté dentro de este lazo se repetirá para siempre. Mientras tanto, la línea [4] se evalúa y como es menor a 10 se ejecutará la línea [5], luego, saltará a la línea [4] y se ejecutará nuevamente el ciclo while si la condición es cierta. Todo el ciclo se repite hasta que ”timer = 10”, es aquí donde sale del ciclo y pasa a la línea [6], reiniciando el temporizador, finalmente, salta a la línea [3] y empezará el ciclo nuevamente.

La sentencia ”while” también posee variaciones que no se estudiarán en este resumen, para mayor información consulte un libro de programación en C.

A.3.8.6 Sentencia for

La sentencia ”for” se utiliza primeramente, para un lazo en donde se deban iniciar las variables, definir si la condición debe continuar y por último para incrementar o decrementar contadores. La forma general se observa a continuación:

```
For (valores iniciales, condición, incremento) {
    /* Hacer algo */
}
    /* Seguir     */
```

Los ciclos for se pueden anidar, y comúnmente se utiliza esta anidación para operaciones con matrices en donde se busca o rellena con un valor en específico.

```
1     void main(void) {
2         char timer;

3         for(;;) {
4             for (timer = 0; timer < 10; timer++)
5                 ;
6         }

7}
```

Listado A.18. Trabajando con la Sentencia for

El programa ejemplifica el mismo resultado del listado A.15, pero observe que es mucho más entendible el programa, la línea [3] es igualmente un ciclo infinito y la línea [4] inicializa el contador, compara, realiza un incremento y vuelve a evaluar la condición.

A.3.8.7 Sentencia break y continue

Al ejecutar la sentencia continue dentro de un lazo, obliga al programa a seguir en la siguiente iteración, mientras que la sentencia ”break”, como su nombre lo dice, rompe el lazo y sigue en el programa .

```
1     void main(void) {
2         char timer;
3         char letra = ‘a’;

3         for(;;) {
4             for (timer = 0; timer < 10; timer++) {
5                 letra++;
6                 if (letra > ‘a’ && letra < ‘z’) {
7                     continue;
8                 }
9                 if (letra == ‘Z’) {
10                    letra = ‘a’;
11                    break;
12                }
13            }
14        }

15    }
```

Listado A.19. Trabajando con las Sentencias break y continue

El ciclo es infinito, si el temporizado es menor a 10, ejecuta el contenido del paréntesis. En la línea [5] ‘a’ cambia a ‘b’, siguiendo la línea [6] es verificada la condición, si la letra está entre ‘b’ y ‘y’ entonces pasará a la línea [7], esta sentencia obliga a ir a la línea [4] y se repetirá el ciclo [4] a [7] hasta que letra sea ‘z’, luego pasará a la línea [9], cuando la letra sea ‘Z’, entonces dará un reinicio al caracter, pero en la línea [11], saltará a la línea [3], es decir, sale del ciclo y salta hacia la raíz.

A.3.9 Arreglos o Matrices

Un arreglo o matriz es un grupo de variables distribuidas equitativamente en lugares consecutivos de memoria, en estas se puede almacenar información.

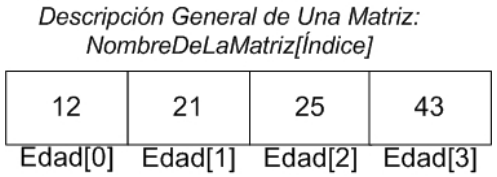

Figura A.2. Gráfico de un Arreglo Unidimensional

A.3.9.1 Declaración de una Matriz

Se declaran igual que una variable, la única diferencia es el especificar cuantos elementos tendrá la matriz.

```
char edad[4];
int  tiempo[3];
```

A.3.9.2 Inicialización de una Matriz

Se inicializa una matriz con el contenido deseado de miembros, seguido del igual y entre llaves separados por comas.

```
char edad[4] = {12, 21, 35, 43};
int  tiempo[3] = {300, 500, 1000};
```

A.3.9.3 Arreglos de Caracteres

En sub secciones anteriores se habló de cadenas y se expresó que era un conjunto de caracteres con terminación nula. A un arreglo de caracteres se les conoce como cadena de caracteres.

```
unsigned char[7] = “Isaias”;
unsigned char[ ] = “Isaias”;
```

El arreglo de caracteres posee terminación nula, por eso los siete elementos en vez de seis. Si como en este caso, el arreglo va a ser inicializado, se prefiere la última asignación pues es más cómoda y no se debe especificar cada vez una nueva cantidad de valores.

A.3.9.4 Arreglos Multidimensionales

Los arreglos anteriores eran del tipo unidimensional porque el índice era solo uno, mientras que un arreglo multidimensional puede contener dos o más índices.

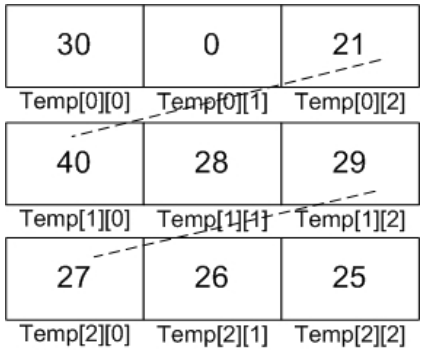

Figura A.3. Gráfico de un Arreglo Multidimensional

A.3.9.5 Declaración de un Arreglo Multidimensional

Una matriz multidimensional consta de más de un índice. Por lo general, cuando es de dos posiciones se habla de que tiene una fila y una columna, ver el ejemplo siguiente:

```
unsigned char temp[3][3];
unsigned short CoordEspaciales[20][20][20];
```

Si nos tratamos de mover por la anteriormente declarada temp, comenzamos desde el índice 00, 01, 02 y luego comenzará en la siguiente fila 10, 11, 12 y así sucesivamente, ver figura A.3.

A.3.9.6 Inicialización de una Matriz Multidimensional

El procedimiento es similar a una matriz unidimensional, ahora solo posee un índice más que hay que añadir.

```
unsigned char temp[3][3] =  {
                                {30,  0, 21},
                                {40, 28, 29},
                                {27, 26, 25}
                            };
```

A.3.9.7 Arreglos de Caracteres Multidimensionales

Ver sección A.3.9.3.

```
Unsigned char nombres[2][20] = {
                                   “Rangel Alvarado”,
                                   “Ivan Alvarado”
                               };
```

A.3.9.8 Uso de Matrices

El siguiente programa demuestra el uso de matrices multidimensionales y ciclos.

```
1     void main(void) {
2         char i,j,temp[3][3];

3         for(;;) {
4             for (i = 0; i < 3; i++) {
5                 for (j = 0; j < 3; j++)
6                     temp[i][j] = 0;
7             }
8             temp[2][0] = 5;
9             temp[1][2] = 21;
10            i = temp[2][0];
11        }

12    }
```

Listado A.20. Trabajando con Matrices Multidimensionales

Las líneas [4] y [5] realizan la función de anidar los ciclos for. La línea [6] llena la matriz en la posición fila,columna con el valor de cero, es decir, inicializa la matriz con valores. La línea [8] rellena la posición `2,0` de la matriz con un `5` e igualmente el elemento `1,2` con `21` dado en la línea [9], como no estamos utilizando nuevamente la variable `i`, tomamos el valor de la matriz en la posición 2,0 y lo almacenamos en la variable temporal `i`, que es como resultado, un `5`.

A.3.10 Punteros

La versatilidad de C radica en el uso de punteros. Un puntero no es más que una variable la cual nos permite movernos por la memoria, es decir, contiene la dirección de la memoria a la cual vamos a afectar. Como se verá un poco más adelante, la matriz es un estilo de puntero, pues almacenamos datos y nos movemos sobre ella buscando o insertando información.

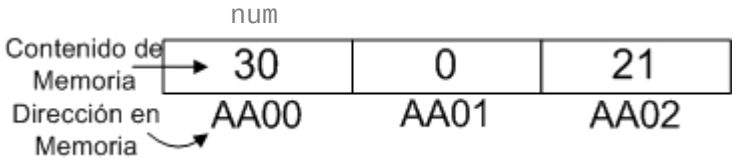

Figura A.4. Grafico Representativo de un Puntero

A.3.10.1 Declarando un Puntero

La declaración de un puntero es igual a la de una variable a excepción que lleva un asterisco como prefijo.

```
unsigned char *pnum;
```

También se pueden mezclar declaraciones con variables, como se dijo anteriormente, un puntero también es una variable.

```
unsigned char *pnum, num = 30;
```

Una convención muy utilizada en lenguaje C es que la variable tipo puntero siempre empiece con la letra p, que simboliza un puntero.

A.3.10.2 El Operador de Referencia

Hasta ahora hemos declarado el puntero, pero no le hemos asignado a quien verá en la memoria, es decir, estará flotando, si tratamos de usarlo puede apuntar a cualquier lugar y no queremos que suceda esta anomalía, sino que apunte a un lugar deseado.

Utilizando el operador de Referencia (`&`, ver tabla A.4) nos devuelve la *dirección de* la variable, registro al cual deseamos alterar o leer. Algo muy importante es que *el puntero debe de ser del mismo tipo de la variable*.

```
pnum = &num;
```

Esto es lo mismo a decir: “Dame la dirección de `num`”.

A.3.10.3 Utilizando el Puntero

Ahora que tenemos la dirección de la variable ¿Cómo alteramos el contenido de un puntero?. Simplemente utilizando el operador `*`. Dicho operador es el operador de indirección, se le llama así pues “indirectamente”, modificando el puntero, modificamos la variable

```
*pnum = 22;
```

Esto es lo mismo a decir, “Modifica el contenido de `pnum` a“. Pero como `pnum` tiene la dirección de `num`, es lo mismo que si escribiéramos directamente en `num`.

¿Porqué utilizar punteros si se puede hacer de una manera mucho más fácil?. Simple, utilizar punteros en vez de arreglos agiliza la operación de la máquina, es más eficiente; más adelante veremos que es mucho más fácil llevarse por punteros cualquier tipo de datos, en vez de especificar un grupo cualquiera y una muy importante característica no explorada en este resumen, es el alojamiento de memoria dinámica.

A.3.10.4 Inicializando Punteros

Anteriormente se aclaró que un puntero no apunta inicialmente a ningún lugar de la memoria. Inicializar el puntero se realiza como si fuese una variable común, a excepción que el valor dispuesto ahora es una posición de memoria.

```
unsigned char *pnombre = NULL;
```

Con esta declaración nos aseguramos de que sea iniciada, pero el valor apuntado es nulo o 0.

```
1     void main(void) {
2         unsigned char i = 9;
3         unsigned char j = 127;
4         unsigned char *pvar = NULL;

5         pvar  = &i;
6         *pvar = *pvar + 1;
7         pvar  = &j;
8         *pvar = *pvar – 9;
9         if (i > *pvar)
10           pvar = &i;

11    }
```

Listado A.21. Trabajando con Punteros

La línea [5] hace al puntero apunte hacia la variable `i`, la siguiente línea, incrementa el contenido del puntero (9) en `1`, resultado un `10`, seguidamente, la línea [7] cambia el puntero a la variable `j`.

Seguidamente la línea [8] decrementa el contenido del puntero, cambiando a `118`. Luego comparamos `i` con `pvar`, pero no la dirección, sino el contenido de `pvar`, que es lo mismo a `i > j`, si es cierto, ejecuta la línea [10] apuntando nuevamente a la variable `i`.

A.3.10.5 Puntero a Caracter

Un puntero a caracter puede ser inicializado como una cadena, pero como la cadena es un conjunto de caracteres, el puntero solo almacenará una variable a su vez, apuntando al inicio de la cadena.

```
unsigned char *pname = “Isaias”; /* Apunto a I */
```

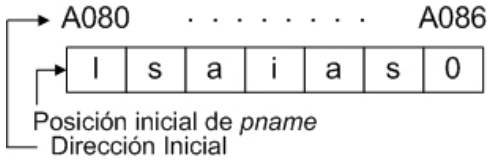

Figura A.5. Puntero a Caracter

```
1     void main(void) {
2         unsigned char *pname = “Isaias”;
3         unsigned char i, letra;

4         for (i = 0; i != 0; i++) {
5             letra = *pname++;
6             /* Imprime la letra   */
7         }

8     }
```

Listado A.22. Trabajando con Punteros a Caracter

La línea [2], declara el puntero a caracter, apuntando a la letra `'I'` o el inicio de la cadena, luego, la variable `letra` almacena el carácter, y siguiendo el orden de precedencia, se incrementa la posición del puntero. Asumiendo que la línea [6] imprime la tecla en una pantalla, salta luego a ejecutar la línea [4], el ciclo se repetirá e imprimirá toda la cadena hasta que se encuentre el carácter de fin de cadena `0` ó `\0`.

A.3.10.6 Punteros y Arreglos

Un puntero y un arreglo son lo mismo, cuando un arreglo es llamado, es transformado a un puntero al primer elemento del arreglo.

```
unsigned char *pname;              /* Puntero   */
unsigned char  name[] = "Isaias";  /* Arreglo   */
pname = name;     /* Inicia el puntero al primer
                     elemento del arreglo       */
```

A.3.10.7 Aritmética de Punteros

Si el puntero es una variable, igualmente se pueden realizar operaciones matemáticas. Recuerde que el puntero apunta siempre a una localidad de memoria y se pueden adquirir la dirección apuntada y el contenido de la dirección apuntada, con esto, se pueden realizar problemas complejos.

```
pname = &name[1]; /* Apunto a la primera 's'    */
pname++;          /* Apunto a la primera 'a'    */
```

La segunda línea anterior también pudo haber sido escrita

pname += 1;

Veamos ahora la siguiente parte de código, reemplaza la letra 'I' en la letra 'i' de la cadena utilizando punteros, si `pname = &name[0]`.

```
*(pname + 2) = *pname; /* Reemplaza 'i' por 'I'*/
```

Esto es equivalente a decir:

```
name[3] = name[0];
```

Para el manejo de punteros con aritmética es necesario el uso de paréntesis, debido a que el operador * es de mayor precedencia a los operadores aritméticos, en este caso, el operador +.

A.3.10.8 Notación de Punteros Para Arreglos

Un arreglo se asemeja a un puntero y se puede tratar de igual forma. Asumiendo tengamos la notación de un arreglo unidimensional, controlada por una variable en algún lugar del programa de flujo:

```
name[i];
```

Sería lo mismo interpretar este tipo de notación `name[i]`, para la posición del arreglo por su puntero

```
*(name + i);
```

Así, de esta forma, podemos tratar un simple arreglo como un puntero. Lo recomendable es no utilizar esta nomenclatura, pues tiende a confundir, por lo general, cuando se escribe código, esta notación se utiliza para punteros puros, no para arreglos en sí.

A.3.10.9 Notación de Punteros para Arreglos Multidimensionales

De la sub sección anterior, se puede aplicar esta regla a arreglos multidimenionales; si tenemos en cuenta que el siguiente pedazo de código corresponde a una sección de programa:

```
name[i][j];
```

También podemos interpretar en alguna parte del código a acceder o escribir a algún miembro del arreglo de la siguiente manera:

```
*(name[i] + j);
*(*(name + i) + j);
```

Realmente este tipo de programación para arreglos es muy inentendible, así que no se recomienda su uso, solo se da ejemplo de esta, por si algún programador que desea no se entienda su código, desea demostrarlo. Recuerde que la programación debe ser clara.

A.3.10.10 Punteros y Arreglos Multidimensionales

Igualmente se expresan los punteros para arreglos multidimensionales, si tenemos un arreglo:

```
typedef unsigned char INT8U;
INT8U data[2][5] = {
                      {1,2,3,4,5},
                      {6,7,8,9,10}
                   };
```

Podemos apuntar a cualquier línea con un puntero.

```
INT8U *pdata = &data[2][5];   /* Apunto a 10    */
pdata = &data[1][3];          /* Apunto a 8     */
pdata = &data[1];             /* Apunto a 6     */
```

Para declarar un arreglo de punteros, necesitamos de la siguiente declaración

```
unsigned char (*pdata)[4]; /*ArregloDe4Punteros*/
```

La declaración anterior puede servir para declarar un arreglo de los vectores de interrupción del microcontrolador, en donde cada puntero sería la dirección a la interrupción a generar.

A.3.11 Funciones

Una función es un pedazo de código que se puede invocar muchas veces dentro de un programa. Una razón de crear funciones es la de hacer un programa más corto rompiéndolo en pequeños pedazos.

Un programa complejo puede especificar muchas funciones que hacen su resolución más simple, los apéndices B hasta O tratan de realizar funciones específicas para los microcontroladores y hacer la vida del programador de proyectos un poco más sencilla.

Así una función tiene un encabezado que declara una función, luego sigue su nombre entre paréntesis y entre llaves la definición de la función. Veamos el siguiente ejemplo de una función que retorna el resultado de la ley de Ohm (V = RI).

```
int volt(unsigned char R, char I) {/* Encabezado */
    int voltaje;        /* Variable Temporal    */

    voltaje = R * I;    /* Ley de Ohm           */
    return voltaje;     /* Retorna el resultado */
}
```

A.3.11.1 Declaración

La estructura general del encabezado de una función es:

VariableDeRetorno NombreDeFunción(ListaDeArgumentos)

```
Int volt(unsigned char R, char I)  /* Encabezado */
```

A.3.11.2 Paso de Argumentos

Los argumentos se encuentran entre paréntesis, existen dos tipos de *paso de argumentos*, *por valor y por referencia*. Los utilizados *en la función anterior son del tipo por valor*, pues no alteran directamente el contenido de la variable, sino que se realiza una copia de la variable sin alterar el contenido.

A.3.11.3 Cuerpo de la Función

El cuerpo realiza la labor específica de la función, en este caso, calcula el voltaje. Inicialmente se declara una variable temporal `voltaje`, las variables declaradas internamente en la función, solo existen dentro de ella, cuando la función cumple su cometido (regrese a la función principal) la variable `voltaje` desaparecerá del sistema.

A.3.11.4 Retorno de la Variable

En el encabezado de la función se especifica la variable que se va a retornar, volviendo a la función `volt()`, esta retorna un tipo entero.

```
int volt(unsigned char R, char I)
```

Si observamos más a fondo el cuerpo de la función, la última línea nos describe que variable va a ser retornada

```
return voltaje;
```

Esta variable (`voltaje`) también fue declarada un tipo entero, sino el compilador dará advertencias parciales sobre la variable a retornar. No necesariamente tuviésemos que haber escrito la función con una variable temporal de cálculo, la misma acción pudo haberse escrito de la siguiente manera:

```
int volt(unsigned char R, char I) {
    return (R*I);
}
```

La función tiene el mismo fin o propósito que la original y dará el mismo resultado.

A.3.11.5 Llamado de la función

La función debe ser llamada especificando el mismo tipo de variable como argumentos a utilizar. Observe esta parte del código:

```
unsigned char a;
char b;
int resultado;
a = 2;
b = 5;
resultado = volt(a,b);  /* Utilizando variables   */
resultado = volt(9,-1); /* Utilizando constantes */
```

Preferiblemente se prefiere escribir la declaración de la función en los archivos `*.h` y la definición en los archivos `*.c`, de tal manera que solo llamáramos la librería correspondiente a todas las funciones y utilizáramos solo las necesitadas, tal es el caso en C de la librería estándar de entrada y salida, `stdio.h`, en donde es invocada y utilizada solo las funciones necesarias de momento.

Imaginemos que existe una librería de funciones eléctricas (`FE.h`)que llama a la función `volt()` para calcular el voltaje y utilicemos la librería `stdio.h` para imprimir en pantalla con printf el voltaje.

```
1     #include <stdio.h> /* Librería para PC     */
2     #include "FE.h"    /* Funciones de usuario */

3     void main(void) {
4       int res, i;

5       printf("Valores de Voltaje a 10 Ohms\n");
6       for(i = 0; i < 255; i++) {
7          res = voltaje(10,i);
8          printf ("voltaje = %i V a %i A\n",res,i);
9       }
10      return 0;

11    }
```

Listado A.23. Trabajando con Funciones

La línea [2] especifica funciones creadas por el usuario, en este caso posee la función `volt()` dentro de la librería (ver archivo zip de módulos para un ejemplo de librerías de funciones). Luego, la línea [5] especifica el encabezado en el monitor de la pantalla. Las líneas [6] a [8] imprimen en la pantalla el resultado del cálculo del voltaje; cuando la iteración sea 50 imprimirá en la pantalla:

```
voltaje = 500 V a 50 A
```

A.3.11.6 Punteros como Argumento de Función

Cuando el puntero es el argumento de la función, toma el argumento pasado por valor, haciendo una copia de la dirección.

```
1     void i_por_I(unsigned char *pname);
2     int mul_por_2(unsigned int *pvar);

3     void main(void) {
4         unsigned char name[] = “Isaias”;
5         int result, a = 5;

6         result = mul_por_2(&a);
7         i_por_I(name);
8         i_por_I(“Indigo”);

9}


10    void i_por_I(unsigned char *pname) {
11        *(pname + 3) = *pname;
12    }

13    int mul_por_2(unsigned int *pvar) {
14        return ((*pvar)*2);
15    }
```

Listado A.24. Trabajando con Punteros como Argumento a Funciones

Las líneas [1] y [2] son la declaración de las funciones. En el caso [1], la palabra `void` por delante de la función, simboliza que no posee valor de retorno. La línea [10] a [15] contienen la definición de cada función.

La línea [4] define e inicializan el puntero tipo arreglo, mientras que la línea [5] dos variables tipo enteras. Finalmente, las líneas [6] a [8] llaman a las funciones. La primera toma una copia de la *dirección de a* y hace una operación de multiplicación por dos, mientras que las últimas dos líneas [7] y [8] son operaciones de punteros con arreglos.

A.3.11.7 Punteros a Funciones

Un puntero a función contiene la dirección de memoria de la función a invocar. Para ejecutar de manera correcta esto, el puntero a función, debe concordar con valores de retorno y argumentos iguales, es decir, debe tener las mismas características de la función.

```
int (*pfunc)(char var1, char var2);
```

Los paréntesis son necesarios para saber si es puntero a función, sin paréntesis, sería una función que retorna un puntero.

```
1     int suma(char sumando1, char sumando2);
2     int resta(char minuendo, char sustraendo);

3     void main(void) {
4         int resultado;
5         int (*pfunc)(char var1, char var2);

6         pfunc = suma;
7         resultado = pfunc(3,-1);
8         pfunc = resta;
9         resultado = pfunc(9,15);
10}

11    int suma(char sumando1, char sumando2) {
12        return (sumando1 + sumando2);
13    }

14    int resta(char minuendo, char sustraendo) {
15        return (minuendo - sustraendo);
16    }
```

Listado A.25. Trabajando con Punteros a Funciones

La línea [5] define el puntero a función, mientras que las líneas [6] y [8] hacen apuntar la función a la correspondiente operación a ejecutar por las líneas [7] y [8].

A.3.11.8 Puntero a Función como Argumento de Funciones

En una función también puede residir un puntero a función. La función llamará a la función apuntada dentro de la definición de la función pasando dicha función como argumento.

```
1   int suma(char sumando1, char sumando2);
2   int mul2(int (*pfunc)(char v1,char v2),char a);

3   void main(void) {
4       int resultado;

5       resultado = mul2(suma, 125);
6   }

7   int suma(char sumando1, char sumando2) {
8     return (sumando1 + sumando2);
9   }

10  int mul2(int (*pfunc)(char v1,char v2),int a) {
11      return pfunc(a, a);
12  }
```

Listado A.26. Trabajando con Punteros a Funciones como Argumento de Función

La línea [2] define una función como argumento; dicha función obtiene un resultado multiplicado por dos. La línea [5] invoca la función que contiene como argumento la función suma multiplicando el argumento 125 por 2, definido por el cuerpo de la función en la línea [11].

A.3.11.9 Arreglo de Punteros a Funciones

```
int (*pfunc[4])(char var1, char var2) =
    {suma, resta, mult, div};
```

La línea anterior sería como definir un arreglo de punteros a función.

Para llamar un arreglo con una función específica definida, ejemplo, la división de dos variables.

```
pfunc[2](24, 2);  /* Llama a la función div() */
```

A.3.12 Estructuras

Una estructura es una colección de variables que comparten el mismo nombre. Las estructuras son un tipo de acción poderosa de C que nos permite ordenar agrupadamente variables.

A.3.12.1 Declaración de una Estructura

Una estructura se define por su nombre principal y sus miembros.

```
struct MOTOR {
    unsigned char n     /* Número              */
    unsigned int rpm    /* Revoluciones        */
    char rot            /* Sentido de Rotación */
};
```

Para conveniencia de nosotros, todos los códigos reusables propuestos en estructuras, serán escritos por la definición `typedef`.

```
struct MTR {
    unsigned char n     /* Número              */
    unsigned int rpm    /* Revoluciones        */
    char rot            /* Sentido de Rotación */
} MOTOR;
```

Todo lo anterior no tiene sentido si no se asigna que tipo de datos será cada motor, las partes de código anteriores equivalen a decir `unsigned char`, `typedef unsigned char INT8U`; no hemos definido la variable que lo representa.

```
MOTOR induccion, dc, iman;
```

A.3.12.2 Inicializando la Estructura

Simple y sencillamente se debe especificar la variable estructural, seguido del igual y luego entre llaves inicializar las variables por comas.

```
MOTOR induccion =
{
      1,
      60,
      -1
};
```

A.3.12.3 Acceso a los Miembros de una Estructura

Para acceder solo debemos invocar la variable seguida de un punto y el elemento interno a la estructura.

```
induccion.n =  2;
dc.n        =  2;
iman.rot    = -1;
```

A.3.12.4 Utilizando Punteros para Acceder Estructuras

Igualmente, se puede utilizar punteros para acceder la estructura anteriormente creada.

```
MOTOR induccion, *pinduc;
pinduc = &induccion;
pinduc->n = 2;
(*pinduc).rpm = 60;
```

Ambas utilicades de punteros acceden a la estructura, pero la más aceptable es `pinduc->` , que es como generalmente se expresa.

```
1     typedef FIG {
2         unsigned int largo;
3         unsigned int ancho;
4         unsigned int alto;
5     } SOLIDO;

6     SOLIDO paralelepipedo =
7     {12, 10, 5};

8     void main(void) {
9     SOLIDO *ptr;
10    unsigned int vol;

11    ptr = &paralelepipedo;
12    vol = (ptr->largo)*(ptr->ancho)*(ptr->alto);
13    }
```

Listado A.27. Trabajando con Estructuras

Las líneas [1] a [7] definen e inicializan la estructura, la línea [9] utiliza un puntero para acceder la estructura y la línea [11] referencia a que estructura se referirá para en la última línea del programa, calcular el volumen del paralelepipedo.

A.3.12.5 Campo de Bits

Una utilidad importante de las estructuras es el manejo de bits, con esto podemos escribir a bits sin necesidad de escribir alterar otros bits del registro, el ejemplo se puede ver en la siguiente página.

```
typedef unsigned char INT8U;
typedef Byte {
      INT8U B0    :1;
      INT8U B1    :1;
      INT8U B2    :1;
      INT8U B3    :1;
      INT8U B4    :1;
      INT8U B5    :1;
      INT8U B6    :1;
      INT8U B7    :1;
} BITS;

BITS PortA;
```

Para acceder al bit 5 del puerto A y asignarle un cero, la secuencia sería la que se da a continuación.

```
PortA.B5 = 0;
```

Igualmente se pueden agrupar más bits dentro de la estructura, no solo pueden ser individuales.

```
typedef unsigned char INT8U;
typedef Byte {
      INT8U B0B3  :4;
      INT8U B4    :1;
      INT8U B5    :1;
      INT8U B6    :1;
      INT8U B7    :1;
} BITS;

BITS PortA;

PortA.B0B3 = 0xF;
```

A.3.13 Uniones

Las uniones son un grupo no muy utilizado de datos, se definen, declaran y usan igual que las estructuras, pero estas, a diferencia de las estructuras, ocupan el mismo lugar de memoria de la variable.

```
1     typedef unsigned int INT16U;
2     typedef unsigned char INT8U;
3     typedef union MiUnion{
4         INT16U Word;
5         INT8U HiByteOfWord;
6     } WORD;

7     WORD Palabra;

8     void main(void) {
9      for(;;) {
10      Palabra.HiByteOfWord = 0xFF; /*Word=0xFF00*/
11      Palabra.Word = 0xABCD; /*Word=0xABCD*/
12      Palabra.HiByteOfWord = 0x0A; /*Word=0x0ACD*/
13     }
14    }
```

Listado A.28. Trabajando con Uniones

Una descripción muy común utilizada para definir puertos es utilizar uniones y estructuras de forma anidada, como de la siguiente manera:

```
typedef union {
  byte Byte;
  struct {
    byte PTA0    :1;
    byte PTA1    :1;
    byte PTA2    :1;
    byte PTA3    :1;
    byte PTA4    :1;
    byte PTA5    :1;
    byte PTA6    :1;
    byte         :1;
  } Bits;
  struct {
    byte PTA     :7;
    byte         :1;
  } MergedBits;
} PTASTR;
```

```
extern volatile PTASTR _PTA @0x00000000;
#define PTA _PTA.Byte
#define PTA_PTA0 _PTA.Bits.PTA0
#define PTA_PTA1 _PTA.Bits.PTA1
#define PTA_PTA2 _PTA.Bits.PTA2
#define PTA_PTA3 _PTA.Bits.PTA3
#define PTA_PTA4 _PTA.Bits.PTA4
#define PTA_PTA5 _PTA.Bits.PTA5
#define PTA_PTA6 _PTA.Bits.PTA6
#define PTA_PTA _PTA.MergedBits.PTA
```

Así, cuando el usuario escribe a puertos, ya existen especificaciones de uniones y estructuras definidas en C para CodeWarrior.

```
PTA = 0xA0;       /* PTA = 0xA0     */
PTA_PTA0 = 1;     /* PTA = 0xA1     */
```

A.3.14 Más Acerca de Variables

Existen palabras reservadas del C que nos permiten no solo escribir un mejor código, sino que realizarlo más robusto, estas condiciones son: `const`, `static`, `volatile`, `extern`.

A.3.14.1 Variables Tipo const

Se puede aplicar a cualquier variable y es aquella que va a residir en la memoria de programa (FLASH).

```
const unsigned char datos[] = {25,32,12,45,0,10};
```

El prefijo `const` puede ser aplicado a estructuras, funciones, tipos de datos, en fin, todo tipo de variables en lenguaje C. Algunos compiladores ubican variables de tipo `const` en la memoria RAM, CodeWarrior automáticamente ubica estas variables en la memoria FLASH.

A.3.14.2 Variables Tipo Static

Se trabaja con variables tipo estáticas cuando se desea retener el valor aún cuando la función haya sido abandonada. Las variables estáticas el compilador se encarga de inicializarlas con un valor inicial de cero. Si la variable es definida dentro de la función o dentro de un módulo (*.c), solo estará accesible por la función o módulo indicado.

```
void main(void) {
      VarStatInc();     /* al entrar var = 0    */
      VarStatInc();     /* al entrar var = 1    */
      var++;            /* Error, no accesible  */
}

void VarStatInc(void) {
    static unsigned char var = 0;
      var++;
}
```

Listado A.29. Trabajando con Variables Tipo Estáticas

A.3.14.3 Variables Tipo volatile

Por lo general, una variable tipo volatile se presenta cuando se quiere que cada vez en el programa se cambie su contenido.

```
volatile unsigned char PORTA @0X0000;
```

Si escribiéramos esta parte siguiente de código sin volatile, el compilador trataría de optimizar el código eliminando una línea de programación

```
PORTA = 0xFC;
PORTA = 0xAB;
```

Así, si queremos que cada vez que escribimos el valor en un puerto, sea nuevamente reconocido, CodeWarrior ha declarado estos puertos estilo volátiles.

A.3.14.4 Variables Tipo extern

Objetos definidos fuera de cualquier módulo, deben ser definidas del tipo `extern`. Solo las variables globales y puertos pueden ser referidas a este tipo. El compilador sabe de que tipo son, pero no donde se encuentran y es el mismo compilador que se encargará después de buscar la dirección y ejecutar operaciones con ella.

```
extern volatile PTASTR _PTA @0x00000000;
```

Si se observa la línea anterior, representa un pedazo e código utilizado en la sección A.3.13.

A.3.15 Directivas del Preprocesador

Las directivas del preprocesador nos permiten realizar programas de una manera que los cambios no sean traumáticos, sin necesidad de cambiar línea por línea el código, realizar nuevas funciones, compilar secciones de manera inteligente.

A.3.15.1 Macros

Utilizamos macros para hacer secciones de código cambiables de una forma rápida y no traumática, esclarecer el código y salvar tiempo en la realización de un programa.

Si tuviésemos que cambiar la cantidad de integrantes en un programa, en donde estuviera “regada” por todo el programa, se definiría un macro.

```
#define MAX 10
.
unsigned char integrantes[MAX];
.
for(i = 0; i < MAX, i++)
```

A.3.15.2 Compilación Condicional

La compilación condicional se utiliza para hacer, ciertamente, secciones de código reconocibles a la hora de compilar.

```
#define COMPILAR
#define A   1

#ifdef COMPILAR
.
.
#else
.
.
#endif

#if A == 1
.
#endif
```

Como fue definida la palabra `COMPILAR`, la sección de código entre `#ifdef` e `#else` será compilada y ejecutada, al igual, al haber sido definido `A` como `1`, esta sección entre `#if` y `#endif` será compilada.

A.3.15.3 Incluyendo Otras Fuentes

Utilizando la directiva de inclusión, podemos incluir funciones, esto lo hacemos a menudo, cada vez que exploramos el uso de la directiva siguiente:

```
#include <stdio.h>
#include “FE.h”
```

Ambas utilizan funciones de librerías, la diferencia reside en que la línea última busca las funciones en la carpeta del proyecto del programa, mientras que la primera busca la librería en la carpeta del compilador.

APÉNDICE B

RUTINAS REUSABLES DEL CONVERTIDOR ANALÓGICO A DIGITAL (ADC)

Cuando se necesite transformar una cantidad física al mundo digital, se necesita de un convertidor analógico digital o ADC, llamados también cuantizadores. Con la ayuda de transductores (sensores), se procesa una señal analógica, la misma es acondicionada, filtrada y llega a nuestro microcontrolador pasando por este módulo, luego de esto, tendremos un valor analógico representado en una escala digital, en este caso de 0 a 255, debido a que nuestro microcontrolador posee módulos ADC de solo 8 bits.

ADCInit( )

void ADCInit(BOOLEAN adco, INT8U divider)

ADCInit() es el código de inicialización del módulo ADC. Usted deberá llamar a ADCInit() antes de cualquier función provista. Esta función es la responsable de inicializar el tipo de conversión y el reloj divisor del ADC.

Argumentos

adco es el tipo de conversión, 0 para una conversión y 1 ó _ADC0 para una conversión continua.

divider es el divisor de reloj de entrada del ADC. ADCDIV1, ADCDIV2, ADCDIV4, ADCDIV8, ADCDIV16 son macros especiales creados para seleccionar el divisor de entrada.

Valor de Retorno

Ninguno

Notas/Advertencias

ADCInit() no realiza la conversión, solo configura el ADC.

Ejemplo

```
void main(void)
{
      .
      ADCInit(1,ADCDIV4);  /* continua y reloj/4 */
      .
}
```

ADCInit

ADCInit es el código de inicialización del módulo ADC. Usted deberá llamar a ADCInit antes de cualquier función provista. Esta función es la responsable de inicializar el tipo de conversión y el reloj divisor del ADC.

Argumentos

A = adco es el tipo de conversión, 0 para una conversión y 1 ó ADC0 para una conversión continua.

X = divider es el divisor de reloj de entrada del ADC. ADCDIV1, ADCDIV2, ADCDIV4, ADCDIV8, ADCDIV16 son macros especiales creados para seleccionar el divisor de entrada.

Valor de Retorno

Ninguno

Notas/Advertencias

ADCInit() no realiza la conversión, solo configura el ADC.

Ejemplo

```
START .
      lda #ADC0          ; Conversión Continua
      ldx #ADCDIV1       ; Reloj ADC / 1
      jsr ADCInit        ; Inicializa el ADC
      .
```

ADCIntEn( )

void ADCIntEn(void)

`ADCIntEn()` es utilizado para habilitar las interrupciones del módulo ADC. `ADCIntEn()` inhabilita las interrupciones globales para luego habilitar la interrupción del ADC y seguidamente habilitar las interrupciones globales.

Argumentos
Ninguno.
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
void main(void)
{
      .
      ADCIntEn();  /* Int. ADC ON   */
      .
}
```

ADCIntEn

`ADCIntEn` es utilizado para habilitar las interrupciones del módulo ADC. `ADCIntEn` inhabilita las interrupciones globales para luego habilitar la interrupción del ADC y seguidamente habilitar las interrupciones globales.

Argumentos
Ninguno.
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
START .
      jsr ADCIntEn      ; Habilita Int. ADC
      .
```

ADCIntDis( )

void ADCIntDis(void)

ADCIntDis()es utilizado para inhabilitar las interrupciones del módulo ADC. ADCIntDis() inhabilita las interrupciones globales para luego inhabilitar la interrupción del ADC y seguidamente habilitar las interrupciones globales.

Argumentos
Ninguno.
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
void main(void)
{
        .
        ADCIntEn();  /* Int. ADC ON      */
        .
}
```

ADCIntDis

ADCIntDis es utilizado para habilitar las interrupciones del módulo ADC. ADCIntDis inhabilita las interrupciones globales para luego habilitar la interrupción del ADC y seguidamente habilitar las interrupciones globales.

Argumentos
Ninguno.
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
START .
        jsr ADCIntDis    ; Inhabilita Int. ADC
        .
```

ADCConv( )

void ADCConv(BOOLEAN adco)

ADCConv()es utilizado para cambiar del tipo de conversión. ADCConv( ) cambia de conversión continua a una sola conversión o viceversa.

Argumentos

adco es el tipo de conversión, 0 para una conversión y 1 ó _ADC0 para una conversión continua.

Valor de Retorno

Ninguno

Notas/Advertencias

Ninguna

Ejemplo

```
void main(void)
{
      .
      ADCConv(_ADC0);   /* Conv. Continua */
      .
}
```

ADCConv

ADCConv es utilizado para cambiar del tipo de conversión. ADCConv cambia de conversión continua a una sola conversión o viceversa.

Argumentos

A = adco es el tipo de conversión, 0 para una conversión y 1 ó ADC0 para una conversión continua.

Valor de Retorno

Ninguno

Notas/Advertencias

Ninguna

Ejemplo

```
START .
      lda #0T     ; A realizar 1 conversión
      jsr ADC0    ; Configura conversión
      .
```

ADCOff( )

void ADCOff(void)

`ADCOff()` es utilizado para apagar el ADC. `ADCOff()` Mantiene el tipo de conversión, la interrupción y apaga el convertidor.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Preserva la interrupción y el tipo de conversión
Ejemplo

```
void main(void)
{
      .
      ADCOff();         /* Apaga el ADC   */
      .
}
```

ADCStop

`ADCStop` es utilizado para apagar el ADC. `ADCStop` Mantiene el tipo de conversión, la interrupción y apaga el convertidor.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Preserva la interrupción y el tipo de conversión
Ejemplo

```
START .
      jsr ADCStop ; Apaga el Convertidor
      .
```

ADCMeasure( )

BOOLEAN ADCMeasure(INT8U channel)

`ADCMeasure()` es utilizada para saber el estado del canal del ADC. `ADCMeasure()` devuelve un valor booleano del canal a leerse el cual sirve como bandera y saber en el momento justo cuando se puede leer el ADC.

Argumentos

channel es el canal especificado de conversión. Dados por los macros `ADC0, ADC1, ADC2, … ADC11`.

Valor de Retorno

`TRUE` si el canal está disponible para lectura; `FALSE` si no se ha podido realizar una conversión

Notas/Advertencias

Esta función contiene un contador de tiempo fuera, no utilice interrupciones si llama a esta rutina.

Ejemplo

```
void main(void)
{
      .
      if (ADCMeasure(ADC7)) { /* si se puede leer
                              el canal 7      */
         /* procesar dato del canal             */
      }
}
```

ADCMeasure

`ADCMeasure` es utilizada para saber el estado del canal del ADC. `ADCMeasure` devuelve un valor booleano del canal a leerse el cual sirve como bandera y saber en el momento justo cuando se puede leer el ADC.

Argumentos

A = channel es el canal especificado de conversión. Dados por los macros `ADC0, ADC1, ADC2, … ADC11`.

Valor de Retorno

A = 1 si el canal está disponible para lectura; **A = 0** si no se ha podido realizar una conversión

Notas/Advertencias

Esta función contiene un contador de tiempo fuera, no utilice interrupciones si llama a esta rutina.

Ejemplo

```
START .
      lda #ADC3         ; Canal 3
      jsr ADCMeasure    ; Medir
      beq ADC0000.A     ; No procesar
                        ; Procesar resultado
ADC0000.A               ; Resultado no procesado
      .
```

APÉNDICE C

C.1 RUTINA REUSABLE DEL MÓDULO DE GENERACIÓN DE RELOJ (CGMC)

Si ud. posee la necesidad de generar frecuencias más elevadas a las conseguidas con su cristal conectado, utilice el módulo CGMC. El CGMC genera a partir de la señal del cristal o el dispositivo PLL (lazo enganchado de fase) interno, una señal de sincronismo del sistema, necesaria para la operación del microcontrolador y que así se ejecuten las instrucciones en el tiempo preciso.

Así, de esta forma, se puede generar frecuencias de hasta 8 MHz, que corresponde a la frecuencia máxima de bus con un cristal de base de 32kHz, solo configure la función de iniciación del sistema CGMC que actuará de forma automática para generar este reloj, para esto, necesita de un arreglo de capacitores en la entrada CGMC del microcontrolador GP32.

CGMInit( )

void CGMCInit(INT8U PE, INT16U N, INT8U L, INT8U R)

`CGMCInit()` es utilizada para inicializar el módulo CGMC. `CGMCInit()` configura los registros principales del módulo para que opere de modo automático y genere una frecuencia determinada por el PLL interno.

Argumentos

PE, **N**, **L**, **R** son los diferentes contenidos para los parámetros de un PLL, para obtener los valores de los argumentos, revisar la `NT1001`.

Valor de Retorno

Ninguno

Notas/Advertencias

Una mala configuración de un PLL puede causar destrucción del microcontrolador.

Ejemplo

```
void main(void)
{
      . /* Configuración para XTAL = 4.9152 MHz */
      CGMCInit(0x02,0x34,0xD0,0x08);
        /* CGMC para fbus = 8 MHz              */
      .
      .
}
```

CGMInit

`CGMCInit` es utilizada para inicializar el módulo CGMC. `CGMCInit` configura los registros principales del módulo para que opere de modo automático y genere una frecuencia determinada por el PLL interno.

Argumentos

A (PUSH1) = PE, **H:X = N**, **A (PUSH2) = L**, **A = R** son los diferentes contenidos para los parámetros de un PLL, para obtener los valores de los argumentos, revisar la `NT1001`.

Valor de Retorno

Ninguno

Notas/Advertencias

Una mala configuración de un PLL puede causar destrucción del microcontrolador.

Ejemplo

```
START .
      lda #$02          ; P y E
      psha              ; Empuja P y E
      ldhx #$0034       ; N
      lda #$D0          ; L
      psha              ; Empuja L
      lda #$08          ; R
      jsr CGMCInit      ; XTAL = 4.9152, fbus =
                        ; 8MHz
      ais #2            ; Pila en posición
```

C.2 RUTINA REUSABLE DE MEMORIA RAM (RAM)

Por el momento, solo está disponible la rutina RAMClr, para lenguaje ensamblador, puesto a que el mismo compilador C se encarga de los registros en RAM que son declarados estáticos.

RAMClr

Posiciona el cursor en algún lugar de la pantalla sin salirse de los límites de la misma.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Llamarla después de deshabilitar el watchdog o al inicio del programa.
Ejemplo

```
START .
      bset BIT0,CONFIG1 ; Deshabilita watchdog
      jsr RAMClr        ; Borra RAM
```

APÉNDICE D

RUTINAS REUSABLES DE GENERACIÓN DE RETARDO DE SOFTWARE (DELAY)

En ciertas ocasiones, necesitamos retardar nuestro sistema para que se den los cambios de manera visible en el mundo real. Debido a que el microcontrolador procesa instrucciones a 125 ns, si es a un cristal de 32MHz, la temporización es una de las rutinas básicas que se debe tener a mano a la hora de realizar un programa. Estas rutinas de retardo no necesariamente se ajustan a una fina precisión, pero sirven como referencia rápida para sistemas de "*primer plano/segundo plano*" ("foreground/background systems").

Retarde desde microsegundos, hasta milisegundos con un máximo de 65535 ms, si lo desea. De esta forma se perderá tiempo realizando "nada", y podrá visualizar sus acciones. Para aplicaciones donde la precisión es importante, se recomienda utilizar el módulo TIM, del microcontrolador.

delay( )
void delay(INT16U ms)

`delay()` es una rutina de generación de tiempo basado en ciclos de máquina. `delay()` utiliza uno registros principales del CPU (H:X) para generar un retardo programable de base de 1 ms. El macro #define `DELAY_XTAL` controla el retardo dependiendo del cristal utilizado.

Argumentos
ms son los milisegundos a retardar desde 1 hasta 65535. La rutina tiene una base de tiempo de 1 ms.
Valor de Retorno
Ninguno
Notas/Advertencias
A pesar de que se especifica una base de 1 ms, esta no es precisa. Un valor de 0 no generará retardo. Abir el archivo `DELAY.h` y reemplazar:

```
DELAY_XTAL  en 0  para 4.0000 MHz
DELAY_XTAL  en 1  para 4.9152 MHz
DELAY_XTAL  en 2  para 9.8304 MHz
DELAY_XTAL  en 3  para 12.800 MHz
DELAY_XTAL  en 4  para 32.000 MHz
```

Ejemplo

```
void main(void)
{
      .
      delay(15);  /* Retarda 15 ms  */
      .
}
```

delay

`delay` es una rutina de generación de tiempo basado en ciclos de máquina. `Delay` utiliza uno registros principales del CPU (H:X) para generar un retardo programable de base de 1 ms. Existan cinco (5) equates principales para configurar los retardos

Argumentos
H:X = ms son los milisegundos a retardar desde 1 hasta 65535. La rutina tiene una base de tiempo de 1 ms.
Valor de Retorno
Ninguno
Notas/Advertencias
A pesar de que se especifica una base de 1 ms, esta no es precisa. Un valor de 0 no generará retardo. Abir el archivo `DELAY.inc` y reemplazar la línea de código `DELAY##` por:

```
  DELAY4      ; Constante de Retardo a 4.0000 MHz
  DELAY49152  ; Constante de Retardo a 4.9152 MHz
  DELAY98304  ; Constante de Retardo a 9.8304 MHz
  DELAY128    ; Constante de Retardo a 12.800 MHz
  DELAY32     ; Constante de Retardo a 32.000 MHz
```

Ejemplo

```
START
      .
      ldhx #500T  ; A retardar 0.5 seg
      jsr Delay   ; ejecuta retardo
      .
```

delayNus( )
void Delay(INT8U xtal, INT8U us)

`delayNus()` es una rutina de generación de tiempo, solo dispuesta para aquel que requiere un retardo de software aproximado. El retardo es aproximadamente `xtal` × `us` × `tiempo` de 1 ciclo de reloj.

Argumentos
us son los microsegundos a retardar desde 1 hasta 255. La rutina tiene una base de tiempo dispuesta por la descripción de la rutina en el encabezado del módulo.
xtal es el número en Megahertz del cristal oscilador externo.
Valor de Retorno
Ninguno
Notas/Advertencias
El retardo tratado de implementar no es preciso con la ecuación superior, solo es un estimado que está por encima del retardo original.

Ejemplo
```
void main(void)
{
      .
      delayNus(5, 1);   /* 1 us a 5MHz    */
      .
}
```

delayNus

`delayNus` es una rutina de generación de tiempo, solo dispuesta para aquel que require un retardo de software aproximado. El retardo es aproximadamente `xtal` × `us` × `tiempo` de 1 ciclo de reloj.

Argumentos
A = us son los microsegundos a retardar desde 1 hasta 255. La rutina tiene una base de tiempo dispuesta por la descripción de la rutina en el encabezado del módulo.
X = xtal es el número en Megahertz del cristal oscilador externo.
Valor de Retorno
Ninguno
Notas/Advertencias
El retardo tratado de implementar no es preciso con la ecuación superior, solo es un estimado que está por encima del retardo original.

Ejemplo
```
START
      .
      lda #1T     ; A retardar 1 us
      ldx #5T     ; Cristal externo de 5MHz
      jsr Delay   ; ejecuta retardo
      .
```

APÉNDICE E

RUTINAS REUSABLES DE MÓDULO DE INTERRUPCIÓN EXTERNA (IRQ1)

La familia HC08 de Motorola posee un pin especial en donde podemos realizar captura de eventos de bajada. Este módulo especial, IRQ1, permite detectar transiciones o bajos niveles y es recomendable utilizarlo para rutinas de comunicación serie entre dispositivos como PCs y otros microcontroladores. Otras posibilidades del módulo IRQ1 pueden ser un contador de pulsos, frecuencímetro, cálculo de eventos de sincronía, etc.

IRQ1Init( )
void IRQ1Init(BOOLEAN mode1)

`IRQ1Init()` es una rutina encargada en inicializar el módulo de interrupción externa. `IRQ1Init()` configura el pin para detectar bordes de bajada o bajos niveles o ambos.

Argumentos
mode1 es el tipo de nivel a sensar. 1 o _MODE1 para detectar bordes de bajada y bajos niveles; 0 para generar interrupciones a bordes de bajada.
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
void main(void)
{
      .
      IRQ1Init(_MODE1); /* Bordes y Niveles bajos*/
      .
}
```

IRQ1Init

`IRQ1Init` es una rutina encargada en inicializar el módulo de interrupción externa. `IRQ1Init` configura el pin para detectar bordes de bajada o bajos niveles o ambos.

Argumentos
A = mode1 es el tipo de nivel a sensar. 1 o MODE1 para detectar bordes de bajada y bajos niveles; 0 para generar interrupciones a bordes de bajada.
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
START .
      lda #MODE1        ; Bordes y bajos niveles
      jsr IRQ1Init      ; Inicializa el módulo
      .
```

IRQ1Ack( )
void IRQ1Ack(void)

`IRQ1Ack()` es una rutina encargada de generar una señal de reconocimiento de la interrupción. Esta señal es necesaria si posee una interrupción del módulo.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Preserva el bit `MODE1`
Ejemplo

```
interrupt 2 void IRQ1H(void)
{
      .
      .
      IRQ1Ack();  /* Reconoce la interrupción   */
}
```

IRQ1Ack

`IRQ1Ack` es una rutina encargada de generar una señal de reconocimiento de la interrupción. Esta señal es necesaria si posee una interrupción del módulo.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Preserva el bit `MODE1`
Ejemplo

```
IRQ1L(void)
      .
      jsr IRQ1Ack ; Reconoce la interrupción
      rti         ; Retorna de la interrupción
      .
```

IRQ1Status( )

BOOLEAN IRQ1Status(void)

`IRQ1Status()` retorna un valor booleano que define si existe o no un flanco detectado por el módulo

Argumentos
Ninguno
Valor de Retorno
TRUE si se detecta una transición o flanco y FALSE si no ha ocurrido una transición o flanco.
Notas/Advertencias
Esta rutina tiene un uso relevante si no se habilitaron las interrupciones.
Ejemplo

```
void main(void)
{
      .
      if (IRQ1Status()) /* Si hay int.    */
                        /* procesar       */
}
```

IRQ1Status

`IRQ1Status` retorna un valor booleano que define si existe o no un flanco detectado por el módulo

Argumentos
Ninguno
Valor de Retorno
A = 1 si se detecta una transición o flanco ó **A = 0** si no ha ocurrido una transición o flanco.
Notas/Advertencias
Ninguna
Ejemplo

```
START .
      jsr IRQ1Status    ; Verifica estado
      beq IRQ10000.A    ; Si no hay flanco, salir
      .                 ; Procesar
IRQ10000.A              ; No int., seguir
      .
```

APÉNDICE F

RUTINAS REUSABLES DE MÓDULO DE INTERRUPCIÓN POR TECLADO (KBI)

Un módulo especial y muy útil es el módulo KBI. El módulo permite programar la mayoría de los pines de un puerto del microcontrolador, por lo general, el puerto A, para reconocer interrupciones externas con flanco activo bajo y detección de bajos niveles. A diferencia del módulo IRQ1, el módulo KBI fue generado para conectar un teclado matricial, y pese a que internamente, el microcontrolador posee resistores de pull-ups, es decir, puesta a +5V, no se necesita de “hardware” externo más que el teclado, de esta forma, simplifica el impreso y reduce los costos.

KBIInit( )

void KBIInit(INT8U kbier, BOOLEAN modek)

`KBIInit()` es la rutina encargada de inicializar las teclas del módulo como interruptores de teclado. `KBIInit()` asegura que no hayan rebotes de incialización.

Argumentos
kbier son las teclas que sirven como interruptores de teclado, p.e. `_KBIE7, _KBIE6, … _KBIE0`.
modek es el tipo de detección de teclado 1 o `_MODEK` si desea que aún presionada siga ejecutando interrupciones, 0 solo genera una interrupción aún presionada.
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
void main(void)
{
      KBIInit(_KBIE5|_KBIE0,0);
      /* Teclas 0 y 5 como int. de teclado */
}
```

KBIInit

`KBIInit` es la rutina encargada de inicializar las teclas del módulo como interruptores de teclado. `KBIInit` asegura que no hayan rebotes de incialización.

Argumentos
A = kbier son las teclas que sirven como interruptores de teclado, p.e. `KBIE7, KBIE6, … KBIE0`.
X = modek es el tipo de detección de teclado 1 o MODEK si desea que aún presionada siga ejecutando interrupciones, 0 solo genera una interrupción aún presionada.
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
START .
      lda #{KBIE4|KBIE2}      ; Teclas 4 y 2
      ldx #MODEK              ; Int. continua
      jsr KBIInit             ; Inicializa teclado
      .
```

KBIAck( )
void KBIAck(void)

KBIAck() es la rutina encargada de generar una señal de reconocimiento de la interrupción. Es necesaria si se han activado las interrupciones del módulo KBI.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
interrupt 15 void KBIH(void)
{
      .
      .
      KBIAck();         /* Reconoce la int. */
}
```

KBIAck

KBIAck es la rutina encargada de generar una señal de reconocimiento de la interrupción. Es necesaria si se han activado las interrupciones del módulo KBI.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
KBIL
      .
      jsr KBIAck  ; Reconoce la int.
      rti         ; Retorna
      .
```

KBIStatus( )

BOOLEAN KBIStatus(void)

KBIStatus() verifica el registro encargado del teclado y detecta si se ha presionado una tecla. Devuelve TRUE si hay alguna tecla presionada o FALSE si no existe tecla alguna presionada.

Argumentos
Ninguno
Valor de Retorno
TRUE si se detecta que alguna tecla ha sido presionada. FALSE si no se ha detectado tecla alguna presionada.
Notas/Advertencias
Esta rutina tiene utilidad en situaciones donde las interrupciones fueron inhabilitadas o nunca habilitadas.
Ejemplo

```
void main(void)
{
      .
      while (!KBIStatus());   /* Espera tecla   */
      .                       /* Procesar       */
      .
}
```

KBIStatus

KBIStatus verifica el registro encargado del teclado y detecta si se ha presionado una tecla. Devuelve un valor A = 1 si hay alguna tecla presionada o A = 0 si no existe tecla alguna presionada. Esta rutina tiene uso relevante si no se habilitaron las interrupciones del módulo.

Argumentos
Ninguno
Valor de Retorno
A = 1 si se detecta que alguna tecla ha sido presionada. **A = 0** si no se ha detectado tecla alguna presionada.
Notas/Advertencias
Esta rutina tiene utilidad en situaciones donde las interrupciones fueron inhabilitadas o nunca habilitadas.
Ejemplo

```
START .
KBI0000.A
      jsr KBIStatus     ; Verifica si tecla
      beq KBI0000.A     ; Espera tecla
      .                 ; Procesar
```

APÉNDICE G

RUTINAS REUSABLES DEL MÓDULO DE COMUNICACIONES SERIALES ASÍNCRONAS (SCI)

El módulo SCI es utilizado para generar comunicaciones seriales de tipo asíncrona entre otros periféricos y microcontroladores. El SCI puede ser calibrado para una diferente gama de baudios dependiendo del tipo de cristal para alcanzar velocidades relativamente altas. El dispositivo utiliza la comunicación serie NRZ (no retorno a cero), 8N1, pero puede ser cambiado o configurado en otras ocasiones para modos de 9 bits. También las rutinas ofrecen un almacenador intermediario o buffer mientras esta se esté ejecutando, lo que permite recibir y transmitir grandes cantidades de información de manera inteligente.

SCIInit( )

void SCIInit(INT8U baud, INT8U data, INT8U pty)

`SCIInit()` es una rutina encargada de inicializar las características del puerto serial. `SCIInit()` debe ser llamada antes de cualquier rutina de este módulo para establecer un puerto de comunicaciones.

Argumentos

baud es la rata del baudio, ver archivo SCI.h

data la cantidad de bits de datos ocho (8) o nueve (9), cualquier otro valor configurará el SCI a 8 bits de datos

pty es el tipo de paridad

```
SCI_PTY_ODD       para paridad impar
SCI_PTY_EVEN      para paridad par
SCI_PTY_NONE      para no paridad
```

Valor de Retorno

Ninguno

Notas/Advertencias

Ud. debe de acceder el archivo SCI.h y cambiar el valor de SCI_XTAL para definir el tipo de cristal externo que posee.

```
SCI_XTAL    por   0     para 4.9152 MHz
SCI_XTAL    por   1     para 9.8304 MHz
SCI_XTAL    por   2     para 32.000 MHz
```

Rutina SCIInit no implementada en C para micros sin módulo serial.

Ejemplo

```
void main(void)
{
      .
      SCIInit(SCI_BAUD_9600,8,SCI_PTY_NONE);
      /* 9600 baudios, 8 bits de datos, paridad
         ninguna                                  */
      .
}
```

SCIInit

`SCIInit` es una rutina encargada de inicializar las características del puerto serial. `SCIInit` debe ser llamada antes de cualquier rutina de este módulo para establecer un puerto de comunicaciones.

Argumentos

A = baud es la rata del baudio, ver archivo SCI.inc

X (PUSH1) = data la cantidad de bits de datos ocho (8) o nueve (9), cualquier otro valor configurará el SCI a 8 bits de datos

X = pty es el tipo de paridad

```
SCI_PTY_ODD       para paridad impar
SCI_PTY_EVEN      para paridad par
SCI_PTY_NONE      para no paridad
```

Valor de Retorno

Ninguno

Notas/Advertencias

Ud debe de acceder el archivo `SCI.inc` y cambiar el valor de `SCI_XTAL0` por el valor del cristal que ud. posee, p.e.:

```
$SET     SCI_XTAL1            ; XTAL = 9.8304 MHz
$SETNOT  SCI_XTAL2
$SETNOT  SCI_XTAL0
```

Para microcontroladores sin SCI entrar al archive SCI.inc en inicializarlo según la descripción en el encabezado.

Ejemplo

```
START .
      lda #SCI_BAUD_4800      ; 4800 baudios
      ldx #$09                ; 9 bits
      pshx                    ; empuja bits
      ldx #SCI_PTY_NONE       ; Paridad Ninguna
      jsr SCIInit             ; Inicializa serial
      ais #1                  ; Pila en posición
      .
```

SCITxEn( )
void SCITxEn(void)

`SCITxEn()` se encarga de habilitar el transmisor.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
void main(void)
{
      .
      SCITxEn();  /* habilita el transmisor */
      .
}
```

SCITxEn

`SCITxEn` se encarga de habilitar el transmisor.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
START .
      jsr SCITxEn ; Habilita el transmisor
      .
```

SCIRxEn( )
void SCIRxEn(void)

`SCIRxEn()` se encarga de habilitar el receptor.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
void main(void)
{
      .
      SCIRxEn();  /* habilita el receptor */
      .
}
```

SCIRxEn

`SCIRxEn` se encarga de habilitar el receptor.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
START
      .
      jsr SCIRxEn ; Habilita el receptor
      .
```

SCITxDis( )
void SCITxDis(void)

`SCITxDis()` se encarga de inhabilitar el transmisor.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
void main(void)
{
      .
      SCITxDis(); /* inhabilita el transmisor */
      .
}
```

SCITxDis

`SCITxDis` se encarga de inhabilitar el transmisor.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
START .
      jsr SCITxDis      ; Inhabilita el transmisor
      .
```

SCIRxDis( )
void SCIRxDis(void)

`SCIRxDis()` se encarga de habilitar el receptor.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
void main(void)
{
      .
      SCIRxDis(); /* inhabilita el receptor */
      .
}
```

SCIRxDis

`SCIRxDis` se encarga de habilitar el receptor.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo
START

```
      .
      jsr SCIRxDis      ; Inhabilita el receptor
      .
```

SCITxIntEn( )
void SCITxIntEn(void)

`SCITxIntEn()` se encarga de habilitar las interrupciones del transmisor.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Para micros sin SCI esta función no existe.
Ejemplo
```
void main(void)
{
      .
      SCITxIntEn();     /* habilita int. del Tx */
      .
}
```

SCITxIntEn

`SCITxIntEn` se encarga de habilitar las interrupciones del transmisor.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Para micros sin SCI esta función no existe.
Ejemplo
```
START .
      jsr SCITxIntEn    ; Habilita int. del Tx
      .
```

SCIRxIntEn( )
void SCIRxIntEn(void)

`SCIRxIntEn()` se encarga de habilitar las interrupciones del receptor.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Para micros sin SCI esta función no existe.
Ejemplo
```
void main(void)
{
      .
      SCIRxIntEn();     /* habilita int. del Rx */
      .
}
```

SCIRxIntEn

`SCIRxIntEn` se encarga de habilitar las interrupciones del receptor.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Para micros sin SCI esta función no existe.
Ejemplo
```
START
      .
      jsr SCIRxIntEn    ; Habilita el receptor
      .
```

SCITxIntDis( )
void SCITxIntDis(void)

`SCITxIntDis()` se encarga de inhabilitar las interrupciones del transmisor.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Para micros sin SCI esta función no existe.
Ejemplo

```
void main(void)
{
      .
      SCITxIntDis();    /* inhabilita int. del Tx*/
      .
}
```

SCITxIntDis

`SCITxIntDis` se encarga de inhabilitar las interrupciones del transmisor.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Para micros sin SCI, esta función no existe.
Ejemplo

```
START .
      jsr SCITxIntDis   ; Inhabilita int. del Tx
      .
```

SCIRxIntDis( )
void SCIRxIntDis(void)

`SCIRxIntDis()` se encarga de inhabilitar las interrupciones del receptor.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Para micros sin SCI esta función no existe.
Ejemplo

```
void main(void)
{
      .
      SCIRxIntDis();/* inhabilita int. del Rx */
      .
}
```

SCIRxIntDis

`SCIRxIntDis` se encarga de inhabilitar las interrupciones del receptor.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Para micros sin SCI esta función no existe.
Ejemplo

```
START
      .
      jsr SCIRxIntDis   ; Inhabilita el receptor
      .
```

SCITxEmpty( )
BOOLEAN SCITxEmpty(void)

`SCITxEmpty()` permite saber el estado del transmisor para iniciar a transferir un nuevo caracter.

Argumentos
Ninguno
Valor de Retorno
`TRUE` si existe una transmisión en progreso ó FALSE si no existe una transmisión en progreso.
Notas/Advertencias
Para micros sin SCI esta función no existe
Ejemplo

```
void main(void)
{
      .
      while (!SCITxEmpty())   /* Si no hay Trans.*/
      .                       /* Transmitir dato */
}
```

SCITxEmpty

`SCITxEmpty` permite saber el estado del transmisor para iniciar a transferir un nuevo caracter.

Argumentos
Ninguno
Valor de Retorno
A = 1 si existe una transmisión en progreso ó **A = 0** si no existe transmisión en progreso.
Notas/Advertencias
Para micros sin SCI esta función no existe
Ejemplo

```
START .
SCI0000.A
      jsr SCITxEmpty    ; Transmisiones en
progreso?
      bne SCI0000.A     ; Espera a terminar trans.
      .                 ; Transmitir nuevo dato
```

SCIRxFull( )

BOOLEAN SCIRxFull(void)

SCIRxFull() se encarga de verificar si existe una recepción en progreso.

Argumentos
Ninguno
Valor de Retorno
TRUE si existe una recepción en progreso ó FALSE si no existe una recepción en progreso.
Notas/Advertencias
Para micros sin SCI esta función no existe
Ejemplo

```
void main(void)
{     .
      if (!SCIRxFull());      /* Si no hay recep.*/
      .                       /* Recibir dato    */
}
```

SCIRxFull

SCIRxIntDis se encarga de verificar si existe una recepción en progreso.

Argumentos
Ninguno
Valor de Retorno
A = 1 si existe una recepción en progreso ó **A = 0** si no existe una recepción en progreso.
Notas/Advertencias
Para micros sin SCI esta función no existe
Ejemplo

```
START .
      jsr SCIRxFull     ; Inhabilita el receptor
      bne SCI0000.A     ; Salta a seguir
                        ; Recibir dato
SCI0000.A               ; seguir
```

SCITxPutLine( )

void SCITxPutLine(INT8U *pdata)

`SCITxPutLine()` envía una cadena de caracteres por el puerto serie. Esta función utiliza a la llamada anterior `SCITxPut()`.

Argumentos
pdata es el puntero a la posición inicial de la cadena a enviar.
Valor de Retorno
Ninguno
Notas/Advertencias
Debe ser inicializado el módulo serial y habilitar el transmisor. *Para micros sin SCI, esta función no esta implementada en C.*
Ejemplo

```
void main(void)
{     .     /* SCI Inicializado y Tx hábil     */
      SCITxPutLine(“Envío bytes”);  /* Enviar   */
      .
}
```

SCITxPutLine

`SCITxPutLine` envía una cadena de caracteres por el puerto serie. Esta función utiliza a la llamada anterior `SCITxPut`.

Argumentos
Ninguno
Valor de Retorno
H:X = pdata que es el puntero del mensaje e enviar.
Notas/Advertencias
Debe ser inicializado el módulo serial y habilitar el transmisor.
Ejemplo

```
START .                 ; SCI Inicializado y Tx ON
      Ldhx #MENSAJE     ; Puntero del mensaje
      jsr SCITxPutLine  ; Envía mensaje
      .

MENSAJE                 ; Mensaje con fin de
mensaje
      ‘MENSAJE A ENVIAR’,0
```

SCIRxGet( )

INT8U SCIRxGet(void)

`SCIRxGet()` adquiere un caracter por el puerto serie. Sus condiciones son que el SCI esté inicializado y el receptor habilitado.

Argumentos
Ninguno
Valor de Retorno
Retorna un valor de cero (0), no '0' ASCII si la información no fue leída o un caracter diferente de cero si se ha recibido un byte.
Notas/Advertencias
Se debe inicializar el módulo serial y encender el receptor. *Para micros sin SCI, esta función no está implementada en lenguaje C.*
Ejemplo

```
void main(void)
{     .     /* SCI Inicializado y Rx hábil     */
      if (!SCIRxFull)        /* ¿no recepción? */
          data = SCIRxGet();  /* leer car. rec. */
      .
}
```

SCIRxGet

`SCIRxGet` adquiere un caracter por el puerto serie. Sus condiciones son que el `SCI` esté inicializado y el receptor habilitado.

Argumentos
Ninguno
Valor de Retorno
Retorna un valor de cero (0), no '0' ASCII si la información no fue leída o un caracter diferente de cero si se ha recibido un byte.
Notas/Advertencias
Se debe inicializar el módulo serial y encender el receptor. *Para micros sin SCI, `SCI_SCDR` contiene el caracter recibido por serial.*
Ejemplo

```
START .                ; SCI Inicializado y Tx ON
      ; si no existe recepción
      jsr SCIRxGet     ; lee el carac enviado
      beq SCI0000.A    ; Saltar si es 0
      .                ; Procesar resultado
SCI0000.A              ; seguir
```

SCIBufInit( )
void SCIBufInit(void)

`SCIBufInit()` inicializa el buffer anillo en su estado inicial. Usted debe invocar `SCIBufInit()` si piensa utilizar el buffer anillo para adquirir datos por el puerto serie y empaquetarlos.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Entrar al archivo `SCI.h` y adecuar la cantidad de caracteres seriales:

```
SCI_MAX_CHAR        64        /* 64 caracteres para buffer anillo */
```

Función no implementada en lenguaje C para micros sin SCI.
Ejemplo

```
void main(void)
{     .
      SCIBufInit();     /* Buffer inicializado  */
      .
}
```

SCIBufInit

`SCIBufInit` inicializa el buffer anillo en su estado inicial. Usted debe invocar `SCIBufInit` si piensa utilizar el buffer anillo para adquirir datos por el puerto serie y empaquetarlos.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Entrar al archivo SCI.inc y adecuar la cantidad de caracteres seriales:

```
SCI_MAX_CHAR      64    /* 64 caracteres anillo */
```

Para micros sin SCI, el archivo SCI.inc debe adecuarse para el buffer.
Ejemplo

```
START
      .
      jsr SCIBufInit    ; Buffer inicializado
      .
```

SCIBufEmpty( )

BOOLEAN SCIBufEmpty(void)

SCIBufEmpty() señaliza un valor booleano que determina si hay un caracter insertado en el buffer por el puerto serial.

Argumentos
Ninguno
Valor de Retorno
TRUE si el buffer está vacío o sin datos, FALSE si el buffer no está vacío o con datos por extraer.
Notas/Advertencias
Solo implementada para micros con SCI.
Ejemplo

```
void main(void)
{     .
      if (!SCIBufEmpty())     /* Dato en buffer? */
      .                       /* extraer         */
}
```

SCIBufEmpty

SCIBufEmpty señaliza un valor booleano que determina si hay un caracter insertado en el buffer por el puerto serial.

Argumentos
Ninguno
Valor de Retorno
A = 1 si el buffer está vacío o sin datos ó **A = 0** si el buffer no está vacío o con datos por extraer.
Notas/Advertencias
Esta rutina existe en este lenguaje para micros sin SCI.
Ejemplo

```
START
WAIT0 jsr SCIBufEmpty   ; Verifica estado del buf.
      beq WAIT0         ; Si no hay nada esperar
      .
```

SCIBufFull( )
BOOLEAN SCIBufFull(void)

`SCIBufFull()` señaliza un valor booleano que determina si no se pueden insertar más caracteres en el buffer serial.

Argumentos
Ninguno
Valor de Retorno
`TRUE` si el buffer está lleno o no se puede insertar datos en el serial ó `FALSE` si el buffer no está lleno
Notas/Advertencias
Solo implementada para micros con SCI.
Ejemplo

```
void main(void)
{     .
      if (!SCIBufFull());     /* Puedo insertar? */
      .                       /* enviar al buf.  */
}
```

SCIBufFull

`SCIBufFull` señaliza un valor booleano que determina si no se pueden insertar más caracteres en el buffer serial.

Argumentos
Ninguno
Valor de Retorno
A = 1 si el buffer está lleno o no se puede insertar datos en el serial ó **A = 0** si el buffer no está lleno.
Notas/Advertencias
Esta rutina existe en este lenguaje para micros sin SCI.
Ejemplo

```
START
WAIT0 jsr SCIBufFull    ; Verifica estado del buf.
      bne WAIT0         ; Seguir si buffer lleno
      .
```

SCIBufPut( )

void SCIBufPut(INT8U data)

`SCIBufPut()` es una rutina que trata de imponer un caracter en el buffer si y solo si existe espacio en el buffer, de lo contrario el caracter es ignorado.

Argumentos
data es el carácter a enviar hacia el buffer
Valor de Retorno
Ninguno
Notas/Advertencias
Caracter ignorado si el buffer está lleno.
Ejemplo

```
void main(void)
{     .
      SCIBufPut('a');   /* Envía byte al buffer */
      .
}
```

SCIBufPut

`SCIBufPut` es una rutina que trata de imponer un caracter en el buffer si y solo si existe espacio en el buffer, de lo contrario el caracter es ignorado.

Argumentos
A = data es el carácter a enviar hacia el buffer
Valor de Retorno
Ninguno
Notas/Advertencias
Caracter ignorado si el buffer está lleno. *Esta rutina solo existe en este lenguaje para micros sin SCI.*
Ejemplo

```
START
      lda SCI_SCDR      ; Dato captado serialmente
      jsr SCIBufPut     ; Envía al buffer
      .
```

SCIBufGet( )

INT8U SCIBufGet(void)

`SCIBufGet()` es una rutina que recupera un byte del buffer si y solo si el buffer tiene un dato posible a extraer.

Argumentos
Ninguno
Valor de Retorno
Se obtiene el dato extraído del buffer; además, si del buffer no se puede extraer datos envía un cero (0).
Notas/Advertencias
Ninguna
Ejemplo

```
void main(void)
{     .
      data = SCIBufGet();/* recupera del buffer */
      .
}
```

SCIBufGet

`SCIBufGet` es una rutina que recupera un byte del buffer si y solo si el buffer tiene un dato posible a extraer.

Argumentos
Ninguno
Valor de Retorno
Se obtiene el dato extraído del buffer; además, si del buffer no se puede extraer datos envía *A = 0. Esta rutina solo está implementada en este lenguaje para micros sin SCI.*
Notas/Advertencias
Ninguna
Ejemplo

```
START
      .
      jsr SCIBufGet     ; Recoge del buffer
      sta PORTD         ; Almacena en el puerto
```

SCIPutchar( )
void SCIPutchar(INT8U data)

`SCIPutchar()` utiliza el buffer anillo para enviar un caracter serialmente, si el buffer está lleno, el carácter es ignorado.

Argumentos
data es el carácter a enviar por serial mediante el buffer
Valor de Retorno
Ninguno
Notas/Advertencias
Si el caracter es ignorado, el buffer está lleno.
Ejemplo

```
void main(void)
{     .
      SCIPutchar('b');  /* Envía byte por serial */
      .
}
```

SCIGetchar( )

INT8U SCIGetchar(void)

`SCIGetchar()` utiliza el buffer anillo para recibir un caracter serialmente, si el buffer está vacío, el carácter recibido es cero (0).

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Si el caracter retornado es cero (0), el buffer está vacío.
Ejemplo

```
void main(void)
{     .
      if(!SCIBufEmpty())   /* si puedo extraer */
            data = SCIGetchar(); /* recibe     */
      .
}
```

APÉNDICE H

RUTINAS REUSABLES DEL MÓDULO DE BASE DE TIEMPO (TBM)

El módulo de base de tiempo de los microcontroladores HC08 permite generar ratas variable a 8 divisores diferentes. Utilice este módulo cuando necesite, por ejemplo, refrescar un evento a un tiempo específico, realizar un reloj de tiempo real, contar eventos a tiempos específicos, etc. sin necesidad de utilizar el temporizador. Este módulo no es un temporizador en sí, sino una base de tiempo para ajustar eventos en donde se desee utilizar señales de sincronismo sin utilizar el temporizador.

TBMInit( )
void TBMInit(INT8U tbcr)

`TBMInit()` define las condiciones de arranque del módulo de tiempo. Utiliza macros predefinidos para facilitar el uso de argumentos.

Argumentos
tbcr es el divisor del módulo de base de tiempo. Para ver los posibles casos, abrir el archivo TBM.h.
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
void main(void)
{     .
      TBMInit(TBMDIV8); /* XTAL/8 */
      .
}
```

TBMInit

`TBMInit` define las condiciones de arranque del módulo de tiempo. Utiliza macros predefinidos para facilitar el uso de argumentos.

Argumentos
A = tbcr es el divisor del módulo de base de tiempo. Para ver los posibles casos, abrir el archivo `TBM.inc`.
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
START .
      lda #TBMDIV32768  ; carga prescalar
      jsr TBMInit       ; ejecuta XTAL/Divisor
      .
```

TBMOn( )
void TBMOn(void)

`TBMOn( )` inicia el contador del módulo de base de tiempo.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
La rutina de interrupción del módulo debe estar habilitada.
Ejemplo

```
void main(void)
{     .
      TBMOn();    /* TBM inicia a contar */
      .
}
```

TBMOn

`TBMOn` inicia el contador del módulo de base de tiempo.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
START .
      .
      jsr TBMOn         ; Inicia el TBM
      .
```

TBMOff( )
void TBMOff(void)

`TBMOff()` detiene o apaga el contador del módulo de base de tiempo.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
void main(void)
{     .
      TBMOff();   /* TBM detenido              */
      .
}
```

TBMOff

`TBMOff` detiene o apaga el contador del módulo de base de tiempo.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
START .
      .
      jsr TBMOff        ; Detiene el TBM
      .
```

TBMAck( )
void TBMAck(void)

`TBMAck()` reconoce el bit de sobreflujo del módulo de base de tiempo. Si la rutina de interrupción está en uso, una de sus líneas finales debe ser esta.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Llamar al final de la interrupción.
Ejemplo

```
interrupt 17 void TBMH(void)
{     .
      .
      TBMAck();   /* Reconoce interrupción     */
}
```

TBMAck

`TBMAck` reconoce el bit de sobreflujo del módulo de base de tiempo. Si la rutina de interrupción está en uso, una de sus líneas finales debe ser esta.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Llamar al final de la interrupción.
Ejemplo

```
TBML  .
      .
      jsr TBMAck        ; Reconoce la interrupción
      rti
```

TBMReset( )
void TBMReset(void)

`TBMReset()` se encarga de reiniciar el módulo TBM de forma segura. Luego de reiniciar, el primer desborde ocurre a mitad del período especificado.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Primer desborde en la mitad del período especificado.
Ejemplo

```
void main(void)
{     .
      TBMReset(); /* Reinicia el módulo TBM     */
      .
}
```

TBMReset

`TBMReset` se encarga de reiniciar el módulo TBM de forma segura. Luego de reiniciar, el primer desborde ocurre a mitad del período especificado.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Primer desborde en la mitad del período especificado.
Ejemplo

```
START .
      jsr TBMReset      ; Reinicia el módulo TBM
      .
```

TBMSet( )
void TBMSet(INT8U n)

`TBMSet()` impone un nuevo valor de frecuencia en el divisor del módulo. `TBMSet()` reinicia y configura el TBM de forma segura.

Argumentos
n es el número que configura el divisor, ver la página 337 del manual del microcontrolador.
Valor de Retorno
Ninguno
Notas/Advertencias
Un divisor muy bajo puede causar a largo plazo desborde de la pila.
Ejemplo

```
void main(void)
{     .
      TBMSet(2);  /* Reconfigura TBM y div a 2048
*/
      .
}
```

TBMSet

`TBMSet` impone un nuevo valor de frecuencia en el divisor del módulo. `TBMSet` reinicia y configura el TBM de forma segura.

Argumentos
A = n es el número que configura el divisor, ver la página 337 del manual del microcontrolador.
Valor de Retorno
Ninguno
Notas/Advertencias
Un divisor muy bajo puede causar a largo plazo desborde de la pila.
Ejemplo

```
START .
      lda #1T
      jsr TBMSet  ; Reconfigura TBM y divisor 8192
      .
```

TBMStatus( )

BOOLEAN TBMStatus(void)

`TBMStatus()` esta rutina retorna un valor booleano que representa el desborde del contador del módulo `TBM`. `TBMStatus()` cobra relevancia si el módulo posee las interrupciones deshabilitadas.

Argumentos
Ninguno
Valor de Retorno
`TRUE` si ha ocurrido un desborde de conteo ó `FALSE` si no ha ocurrido un desborde de conteo.
Notas/Advertencias
Ninguna
Ejemplo

```
void main(void)
{     .
      if(TBMStatus())   /* si se vence la cuenta */
      .                 /* ejecutar procedimiento*/
}
```

TBMStatus

`TBMStatus` esta rutina retorna un valor booleano que representa el desborde del contador del módulo `TBM`. `TBMStatus` cobra relevancia si el módulo posee las interrupciones deshabilitadas.

Argumentos
Ninguno
Valor de Retorno
A = 1 si ha ocurrido un desborde de conteo ó **A = 0** si no ha ocurrido un desborde de conteo.
Notas/Advertencias
Ninguna
Ejemplo

```
START .
      jsr TBMStatus     ; verifica estado
      beq TBM0000.A     ; ¿cuenta vencida?, saltar
      .                 ; Ejecutar procedimiento
TBM0000.A
```

TBMIntEn( )
void TBMIntEn(void)

`TBMIntEn()` habilita las interrupciones del módulo de base de tiempo.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
void main(void)
{     .
      TBMIntEn();       /* Hab. Int. del TBM
*/
      .
}
```

TBMIntEn

`TBMIntEn` habilita las interrupciones del módulo de base de tiempo.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
START .
      jsr TBMIntEn      ; Habilita Int. del TBM
      .
```

TBMIntDis( )
void TBMIntDis(void)

`TBMIntDis()` inhabilita las interrupciones del módulo de base de tiempo.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo
```
void main(void)
{     .
      TBMIntDis();      /* Deshabilita int. TBM */
      .
}
```

TBMIntDis

`TBMIntDis` inhabilita las interrupciones del módulo de base de tiempo.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo
```
START .
      jsr TBMIntDis     ; Deshabilita Int. del TBM
      .
```

APÉNDICE I

RUTINAS REUSABLES DEL MÓDULO DE INTERFACE DE TEMPORIZADOR (TIM)

Una unidad importante que poseen los microcontroladores actuales, e inclusive los HC08 es el módulo TIM. Con este módulo se pueden generar eventos temporizados con una muy buena resolución, que depende del cristal en uso. Configurando apropiadamente los registros del módulo temporizador se pueden generar eventos temporizador; “output compares” o interrupciones periódicas que generarán trenes de ancho de pulso con una polaridad, duración y frecuencia configurables, PWM o modulación por ancho de pulso, indispensables para el control de velocidad de motores DC, luminosidad de pantallas, etc. Una configuración especial del TIM también permite capturar flancos, ya sea, ascendentes, descentendes o ambos estilos de flancos, esta opción serviría entonces para realizar un frecuencímetro, contador de pulsos, decodificador de ejes, etc.

TIMInit ( )

void TIMInit(INT8U n, INT8U ps, INT8U tmod)

`TIMInit()` es la rutina encargada de inicializar el módulo temporizador.

Argumentos

n es el número del temporizador correspondiente, n = 1 ó 2.

ps es el macro correspondiente al divisor, el cual puede verse en el archivo `BITS.h`.

`DIV1, DIV2, DIV4… DIV64`.

tmod es el período de desborde del temporizador.

Valor de Retorno

Ninguno

Notas/Advertencias

Si el número del temporizador no está en el rango, el módulo no es inicializado.

Ejemplo

```
void main(void)
{     .
      TIMInit(1, DIV2, 0xFFFF);
      /* TIM1, con divisor 2 y cuenta libre    */
      .
}
```

TIMInit

`TIMInit` es la rutina encargada de inicializar el módulo temporizador.

Argumentos

A = n es el número del temporizador. **A = 1 ó 2**.

X (PUSH) = ps es el equate correspondiente al divisor, el cual puede verse en el archivo BITS.equ.

`DIV1, DIV2, DIV4, … DIV64`

H:X = tmod es el período de desborde del temporizador.

Valor de Retorno

Ninguno

Notas/Advertencias

Si el número del temporizador no está en el rango, el módulo no es inicializado.

No utilizar líneas [1], [2], [3] y [6] del ejemplo para micros con 1 temporizador.

Argumento A, línea [1], es el preescalar divisor para micros con un (1) temporizador.

Ejemplo

```
START .
      lda #2T     ; TIM2                        [1]
      ldx #DIV1   ; PS[2:0] = DIV1              [2]
      pshx        ; Empuja divisor              [3]
      ldhx #$0AB9 ; T2MOD = AB9                 [4]
      jsr TIMInit ; Inicializa el módulo TIM    [5]
      ais #1      ; Posiciona puntero de pila   [6]
      .
```

TIMOption ( )

void TIMInit(INT8U n, INT8U option, INT8U channel)

`TIMOption()` es la rutina encargada de inicializar las diferentes funciones especiales del módulo temporizador. Configura el temporizador para: Salida en estado inicial, captura de entrada o `PWM`.

Argumentos

n es el número del temporizador correspondiente, n = 1 ó 2.

option es el macro correspondiente al tipo de configuración, para ver los diferentes estilos, abrir el archivo `TIM.h`.

channel es el canal correspondiente del temporizador entre 0 y 1.

Valor de Retorno

Ninguno

Notas/Advertencias

Si el número del temporizador o el del canal no están en el rango, el módulo no es inicializado.

Ejemplo

```
void main(void)
{     .
      T1CH0 = 0x7FFF;
      /* configura el ciclo de trabajo    */
      TIMOption(1, TIM_PWM_UN, 0);
      /* TIM1 CH0, PWM Unbuffered         */
      .
}
```

TIMOption

`TIMOption` es la rutina encargada de inicializar las diferentes funciones especiales del módulo temporizador. Configura el temporizador para: Salida en estado inicial, captura de entrada o `PWM`.

Argumentos

A (PUSH) = option es el macro correspondiente al tipo de configuración, para ver los diferentes estilos, abrir el archivo TIM.inc.

A = n es el número del temporizador

X = channel es el canal correspondiente del temporizador entre 0 y 1.

Valor de Retorno

Ninguno

Notas/Advertencias

Si el número del temporizador o del canal no está en el rango, el módulo no es inicializado.

No utilizar líneas [2], [3] y [6] del ejemplo para micros con 1 temporizador.

Ejemplo

```
START .
      lda #TIM_IC_RISE  ; Captura flanco asc.   [1]
      psha              ; empuja opción a pila  [2]
      lda #1T           ; TIM1                  [3]
      ldx #0T           ; Canal 0               [4]
      jsr TIMOption     ; Inicializa Opción TIM [5]
      ais #1            ; Posiciona SP          [6]
      .
```

TIMOv ( )
BOOLEAN TIMOv(INT8U n)

`TIMOv( )` devuelve un valor booleano que representa si ha ocurrido o no un sobreflujo del módulo de conteo del `TIM`. Esta rutina tiene relevancia si se han inhabilitado las interrupciones

Argumentos
n es el número del temporizador correspondiente, n = 1 ó 2.
option es el macro correspondiente al tipo de configuración, para ver
Valor de Retorno
`TRUE` si existe un desborde del módulo `TIM` ó `FALSE` si no existe un desborde del módulo TIM.
Notas/Advertencias
Si el número del temporizador o del canal no está en el rango, el módulo no es inicializado.
Ejemplo

```
void main(void)
{     .
      if (TIMOv(2))     /* ¿sobreflujo de TIM2? */
      .                 /* ejecutar código     */
      .
}
```

TIMOv

`TIMOv` devuelve un valor booleano que representa si ha ocurrido o no un sobreflujo del módulo de conteo del `TIM`. Esta rutina tiene relevancia si se han inhabilitado las interrupciones.

Argumentos
A = n es el número del temporizador.
Valor de Retorno
A = 1 si existe un desborde del módulo TIM ó **A = 0** si no existe un desborde del módulo TIM.
Notas/Advertencias
Si el número del temporizador o del canal no está en el rango, el módulo no es inicializado.
No utilizar la línea [2] para micros con 1 temporizador.
Ejemplo

```
START .
      lda #1T           ; TIM1                    [1]
      jsr TIMOv         ; Sobreflujo del TIM1?    [2]
      beq TIM0000.A     ; Seguir
      .                 ; Procesar
TIM0000.A
      .                 ; seguir
```

TIMOvIntEn ( )
void TIMOvIntEn(INT8U n)

`TIMOvIntEn()` habilita las interrupciones al sobreflujo del módulo del temporizador.

Argumentos
n es el número del temporizador correspondiente, n = 1 ó 2.
Valor de Retorno
Ninguno
Notas/Advertencias
Si el número del temporizador o del canal no está en el rango, el módulo no es inicializado.
Ejemplo

```
void main(void)
{     .
      TIMOvIntEn(1)     /* T1SC_TOIE = 1  */
      .
}
```

TIMOvIntEn

`TIMOvIntEn` habilita las interrupciones al sobreflujo del módulo del temporizador.

Argumentos
A = n es el número del temporizador correspondiente, n = 1 ó 2.
Valor de Retorno
Ninguno
Notas/Advertencias
Si el número del temporizador o del canal no está en el rango, el módulo no es inicializado.
No utilizar la línea [1] para micros con 1 temporizador.
Ejemplo

```
START .
      lda #2T           ; TIM1                  [1]
      jsr TIMOvIntEn    ; Habilita int. TIM2    [2]
      .                 ; seguir
```

TIMOvIntDis ( )
void TIMOvIntDis(INT8U n)

`TIMOvIntEn`() inhabilita las interrupciones al sobreflujo del módulo del temporizador.

Argumentos
n es el número del temporizador correspondiente, n = 1 ó 2.
Valor de Retorno
Ninguno
Notas/Advertencias
Si el número del temporizador o del canal no está en el rango, el módulo no es inicializado.
Ejemplo

```
void main(void)
{     .
      TIMOvIntDis(1)    /* T1SC_TOIE = 0  */
      .
}
```

TIMOvIntDis

`TIMOvIntDis` inhabilita las interrupciones al sobreflujo del módulo del temporizador.

Argumentos
A = n es el número del temporizador correspondiente, n = 1 ó 2.
Valor de Retorno
Ninguno
Notas/Advertencias
Si el número del temporizador o del canal no está en el rango, el módulo no es inicializado.
No utilizar la línea [1] para micros con 1 temporizador.
Ejemplo

```
START .
      lda #2T           ; TIM2                    [1]
      jsr TIMOvIntDis   ; Inhabilita int. TIM2  [2]
      .                 ; seguir
```

TIMStop ( )
void TIMStop(INT8U n)

`TIMStop()` detiene el correspondiente módulo de temporizador.

Argumentos
n es el número del temporizador correspondiente, n = 1 ó 2.
Valor de Retorno
Ninguno
Notas/Advertencias
Si el número del temporizador o del canal no está en el rango, el módulo no es inicializado.
Ejemplo

```
void main(void)
{     .
      TIMStop(1)        /* T1SC_TSTOP = 1 */
      .
}
```

TIMStop

`TIMStop` detiene el correspondiente módulo de temporizador.

Argumentos
A = n es el número del temporizador correspondiente, n = 1 ó 2.
Valor de Retorno
Ninguno
Notas/Advertencias
Si el número del temporizador o del canal no está en el rango, el módulo no es inicializado.
No utilizar la línea [1] para micros con 1 temporizador.
Ejemplo

```
START .
      lda #2T           ; TIM2                  [1]
      jsr TIMStop       ; Detiene TIM2          [2]
      .
```

TIMStart ( )
void TIMStart(INT8U n)

`TIMStart()` inicia el correspondiente módulo de temporizador.

Argumentos
n es el número del temporizador correspondiente, n = 1 ó 2.
Valor de Retorno
Ninguno
Notas/Advertencias
Si el número del temporizador o del canal no está en el rango, el módulo no es inicializado.
Ejemplo

```
void main(void)
{     .
      TIMStart(1)       /* T1SC_TSTOP = 0 */
      .
}
```

TIMStart

`TIMStart` inicia el correspondiente módulo de temporizador.

Argumentos
A = n es el número del temporizador correspondiente, n = 1 ó 2.
Valor de Retorno
Ninguno
Notas/Advertencias
Si el número del temporizador o del canal no está en el rango, el módulo no es inicializado.
No utilizar la línea [1] para micros con 1 temporizador.
Ejemplo

```
START .
      lda #2T           ; TIM2                  [1]
      jsr TIMStart      ; Inicia TIM2           [2]
      .
```

TIMReset( )
void TIMReset(INT8U n)

`TIMReset()` reinicia el correspondiente módulo de temporizador.

Argumentos
n es el número del temporizador correspondiente, n = 1 ó 2.
Valor de Retorno
Ninguno
Notas/Advertencias
Si el número del temporizador o del canal no está en el rango, el módulo no es inicializado.
TIMReset() solo reinicia el temporizador, no lo reactiva.
Ejemplo

```
void main(void)
{     .
      TIMReset(1)       /* TIM1 reiniciado      */
      .
}
```

TIMReset

`TIMReset` reinicia el correspondiente módulo de temporizador.

Argumentos
A = n es el número del temporizador correspondiente, n = 1 ó 2.
Valor de Retorno
Ninguno
Notas/Advertencias
Si el número del temporizador o del canal no está en el rango, el módulo no es inicializado.
No utilizar la línea [1] para micros con 1 temporizador.
TIMReset solo reinicia el temporizador, no lo reactiva.
Ejemplo

```
START .
      lda #2T           ; TIM2                  [1]
      jsr TIMReset      ; Reinicia TIM2         [2]
      .
```

TIMChk( )
INT16U TIMChk(INT8U n)

`TIMChk()` devuelve un valor el cual representa la cuenta del temporizador, si TIMChk() retorna un valor de cero (0) se ha vencido la cuenta, es decir, hay un sobreflujo o se ha configurado erróneamente el canal.

Argumentos
n es el número del temporizador correspondiente, n = 1 ó 2.
Valor de Retorno
Retorna un valor de 16-bits, el cual corresponde al tiempo transcurrido de la cuenta.
Notas/Advertencias
Si el número del temporizador o del canal no está en el rango, el módulo no es inicializado.
Ejemplo

```
void main(void)
{     .
      data = TIMChk(1); /* Verifica cuenta T1CNT */
      .
}
```

TIMChk

`TIMChk` devuelve un valor el cual representa la cuenta del temporizador, si `TIMChk` retorna un valor de cero (0) se ha vencido la cuenta, es decir, hay un sobreflujo o se ha configurado erróneamente el canal.

Argumentos
A = n es el número del temporizador correspondiente, n = 1 ó 2.
Valor de Retorno
Retorna un valor de 16-bits en H:X, el cual corresponde al tiempo transcurrido de la cuenta.
Notas/Advertencias
Si el número del temporizador o del canal no está en el rango, el módulo no es inicializado.
No utilizar la línea [1] para micros con 1 temporizador.
Ejemplo

```
START .
      lda #2T           ; TIM2                  [1]
      jsr TIMChk        ; Busca valor T2CNT     [2]
      .                 ; Procesar reg. H:X     [3]
```

TIMChIntEn( )

void TIMChIntEn(INT8U n, INT8U channel)

`TIMChIntEn()` habilita las interrupciones del correspondiente canal.

Argumentos
n es el número del temporizador correspondiente, n = 1 ó 2.
channel es el canal el cual se piensa examinar el sobreflujo.
Valor de Retorno
Ninguno
Notas/Advertencias
Si el número del temporizador o del canal no está en el rango, el módulo no es inicializado.
Ejemplo

```
void main(void)
{     .
      TIMChIntEn(1, 0); /* Int. del T1SC0 hábil */
      .
}
```

TIMChIntEn

`TIMChIntEn` habilita las interrupciones del correspondiente canal.

Argumentos
A = n es el número del temporizador correspondiente, n = 1 ó 2.
X = channel es el canal el cual se piensa examinar el sobreflujo.
Valor de Retorno
Ninguno
Notas/Advertencias
Si el número del temporizador o del canal no está en el rango, el módulo no es inicializado.
Utilizar solo la línea [1] como argumento de canal para Q/JK/JL.
Ejemplo

```
START .
      lda #2T           ; TIM2                  [1]
      ldx #1T           ; CH1                   [2]
      TIMChIntEn        ; Activa interrupción   [3]
      .
```

TIMChIntDis( )

void TIMChOvIntDis(INT8U n, INT8U channel)

`TIMChIntDis()` inhabilita las interrupciones del correspondiente canal.

Argumentos
n es el número del temporizador correspondiente, n = 1 ó 2.
channel es el canal el cual se piensa examinar el sobreflujo.
Valor de Retorno
Ninguno
Notas/Advertencias
Si el número del temporizador o del canal no está en el rango, el módulo no es inicializado.
Ejemplo

```
void main(void)
{     .
      TIMChIntDis(1, 0);/* Int. del T1SC0 des. */
      .
}
```

TIMChIntDis

`TIMChIntDis` inhabilita las interrupciones del correspondiente canal.

Argumentos
A = n es el número del temporizador correspondiente, n = 1 ó 2.
X = channel es el canal el cual se piensa examinar el sobreflujo.
Valor de Retorno
Ninguno
Notas/Advertencias
Si el número del temporizador o del canal no está en el rango, el módulo no es inicializado.
Solo utilizar la línea [1] como argumentos para micros Q/JK/JL.
Ejemplo

```
START .
      Lda #2T           ; TIM2
      Ldx #1T           ; CH1
      TIMChIntDis       ; Inactiva interrupción
      .
```

TIMChOv( )

BOOLEAN TIMChOv(INT8U n, INT8U channel)

`TIMChOv()` devuelve un valor booleano el cual representa el estado de sobreflujo del correspondiente canal del temporizador.

Argumentos
n es el número del temporizador correspondiente, n = 1 ó 2.
channel es el canal el cual se piensa examinar el sobreflujo.
Valor de Retorno
TRUE si existe sobreflujo en el canal ó FALSE si no existe sobreflujo en el canal.
Notas/Advertencias
Si el número del temporizador o del canal no está en el rango, el módulo no es inicializado. Rutina relevante solo sin interrupciones.
Ejemplo

```
void main(void)
{     .
      if (TIMChOv(1))   /* Verifica cuenta T1CNT */
      .
}
```

TIMChOv

`TIMChOv` devuelve un valor booleano el cual representa el estado de sobreflujo del correspondiente canal del temporizador.

Argumentos
A = n es el número del temporizador correspondiente, n = 1 ó 2.
X = channel es el canal el cual se piensa examinar el sobreflujo.
Valor de Retorno
A = 1 si existe sobreflujo en el canal ó **A = 0** si no existe sobreflujo en el canal.
Notas/Advertencias
Si el número del temporizador o del canal no está en el rango, el módulo no es inicializado.
Solo utilizar el argumento A con el núm. del canal para otros micros.
Ejemplo

```
START .
      lda #2T           ; TIM2                  [1]
      ldx #1T           ; CH1                   [2]
      jsr TIMChOv       ; Verifica sobreflujo   [3]
      .
```

TIMOvAck( )
void TIMOvAck(INT8U n)

`TIMOvAck()` es la señal de reconocimiento de interrupción. Necesariamente `TIMOvAck()` debe ir entre las primeras líneas de la interrupción del temporizador.

Argumentos
n es el número del temporizador correspondiente, n = 1 ó 2.
Valor de Retorno
Ninguno
Notas/Advertencias
Si el número del temporizador no está en el rango, el módulo no reconoce la interrupción.
Ejemplo

```
interrupt 6 void TIM1OFH(void)
{     TIMOvAck(1);      /* Int. TIM1 Reconocida */
      .
      .
}
```

TIMOvAck

`TIMOvAck` es la señal de reconocimiento de interrupción. Necesariamente `TIMOvAck` debe ir entre las primeras líneas de la interrupción del temporizador.

Argumentos
A = n es el número del temporizador correspondiente, n = 1 ó 2.
Valor de Retorno
Ninguno
Notas/Advertencias
Si el número del temporizador no está en el rango, el módulo no reconoce la interrupción.
No utilizar la línea [1] para micros con 1 temporizador.
Ejemplo

```
TIM1OVFL    .
            lda #2T      ; TIM2                 [1]
            jsr TIMOvAck ; reconoce int. TIM2   [2]
            .
```

TIMChAck( )
void TIMOvAck(INT8U n, INT8U channel)

`TIMChAck()` es la señal de reconocimiento de interrupción. Necesariamente `TIMChAck()` debe ir entre las primeras líneas de la interrupción del canal del temporizador.

Argumentos
n es el número del temporizador correspondiente, n = 1 ó 2.
channel es el canal el cual se piensa examinar el sobreflujo.
Valor de Retorno
Ninguno
Notas/Advertencias
Si el número del temporizador o del canal no está en el rango, el módulo no reconoce la interrupción.
Ejemplo

```
interrupt 5 void TIM1CH1H(void)
{     TIMAChAck(1,1);   /* Int. T1CH1 Reconocida */
      .
      .
}
```

TIMChAck

`TIMChAck` es la señal de reconocimiento de interrupción. Necesariamente `TIMChAck` debe ir entre las primeras líneas de la interrupción del canal del temporizador.

Argumentos
A = n es el número del temporizador correspondiente, n = 1 ó 2.
X = channel es el canal el cual se piensa examinar el sobreflujo.
Valor de Retorno
Ninguno
Notas/Advertencias
Si el número del temporizador o del canal no está en el rango, el módulo no reconoce la interrupción.
Utilizar solo la línea [1] como argumento del canal.
Ejemplo

```
TIM1CH1L    .
            lda #2T      ; TIM2                [1]
            ldx #1T      ; CH1                 [2]
            jsr TIMChAck ; Reinicia TIM2       [2]
            .
```

APÉNDICE J

RUTINAS REUSABLES DEL TIM, MÓDULO DE CONTADORES DESCENDENTES (TIM – TMR)

El módulo TIM ofrece grandes ventajas que no poseen otros microcontroladores de compañías diferentes. Motorola, o más bien Freescale, conserva este tipo de módulo para la gran mayoría de los microcontroladores. Pero como el microcontrolador solo posee un módulo TIM, algunos piensan que solo es posible temporizar una señal, a excepción del microcontrolador GP32, que posee dos TIM, TIM1 y TIM2. La idea del módulo TMR es convertir un solo módulo TIM para producir una cadena de contadores de tiempo fuera descendentes, de esta manera, ud. podrá temporizar con sus pines de entrada y salida eventos diferentes, sin importar cuantos módulos TIM ud. posea. Posibles aplicaciones de contadores de tiempo fuera, pudieran ser un On Delay, Off Delay, temporizadores de un relevador y una controladora de motores DC.

Nota: Las rutinas solamente están disponibles en lenguaje C, las rutinas de lenguaje ensamblador son mucho más laboriosas y están pendientes.

TmrFuncCfg( )

void TmrFuncCfg(INT8U n, void (*pfunc)(void *), void *arg)

`TMRFuncCfg()` llama a una función luego de vencido un temporizador.

Argumentos
n es el número del temporizador correspondiente, n es determinado.
pfunc es la dirección de la función a ejecutar luego de vencido el temporizado.
arg es el argumento que adquiere la función si lo necesita.
Valor de Retorno
Ninguno
Notas/Advertencias
La compilación arroja un mensaje preventivo, ud. puede ignorarlo.
Ejemplo

```
void main(void)
{     .
      TmrFuncCfg(0, TmrOut0, NULL);
      /* Leer del temporizador 0, arg. nulo     */
      .
}

void TmrOut0
{
      /* Función de tiempo fuera del temporizado */
}
```

TmrSignal( )

void TmrSignal(void)

`TmrSignal()` envía una señal o semáforo luego de vencido el tiempo cada `TIM_MAX_DSEG`; macro definido en `Tmr.h`. El módulo TIM debe estar inicializado en una base de tiempo de un decisegundo (1 dseg).

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
El módulo `TIM` debe estar inicializado en una base de tiempo de un decisegundo (1 dseg).
Ejemplo

```
interrupt 6 TIMOFH(void)
{
      TmrSignal();      /* Cuenta sobreflujos y
                           al vencerse señaliza
                           una bandera
                        */
      .
}
```

Nota: Definir el macro TIM_MAX_TMR en TIM.h con la cantidad de temporizadores a utiizar (hasta 8).

TmrTask( )
void TmrTask(void)

`TmrTask()` verifica que temporizador se ha vencido y ejecuta la función configurada por `TmrFuncCfg()`. Al ejecutarse la función, no volverá a ejecutarse hasta iniciarse de nuevo.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo
```
void main(void)
{     .
      for (;;) {        /* Ejecuta infinitamente */
            TmrTask();  /* Ejecuta multitarea    */

}
```

TmrInit( )
void TmrInit(INT8U n)

`TmrInit()` Inicializa el temporizador correspondiente, contador en 0 y función y argumento nulo de `TmrFuncCfg()`. Esto previene a no ejecutar alguna rutina por `TmrTask()`. El límite de temporizadores lo dispone el macro `TIM_MAX_TMR` definido por `Tmr.h`.

Argumentos
n es el número del temporizador correspondiente. 8 > n > 0.
Valor de Retorno
Ninguno
Notas/Advertencias
`TmrInit()` debe llamarse antes cualquier temporizador que se utilice.
Ejemplo
```
void main(void)
{     .
      TmrInit(0); /* Inicializa el Timer 0     */
      .
}
```

TmrStop( )
void TmrStop(INT8U n)

`TmrStop()` detiene el temporizador correspondiente.

Argumentos
n es el número del temporizador correspondiente. 8 > n > 0.
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
void main(void)
{     .
      TmrStop(0); /* Detiene el Timer 0   */
      .
}
```

TmrStart( )
void TmrStart(INT8U n)

`TmrStart()` inicia la cuenta del temporizador. Para esto se debe tener inicializado y contando el módulo `TIM`.

Argumentos
n es el número del temporizador correspondiente. 8 > n > 0.
Valor de Retorno
Ninguno
Notas/Advertencias
Ejemplo

```
void main(void)
{     .
      TmrStart(0);/* Inicia cuenta del Timer 0 */
      .
}
```

TmrReset( )
void TmrReset(INT8U n)

`TmrReset()` reinicia la cuenta del temporizador.

Argumentos
n es el número del temporizador correspondiente. 8 > n > 0.
Valor de Retorno
Ninguno
Notas/Advertencias
`TmrReset()` solo reinicia el temporizador, no inicia a temporizar.
Ejemplo

```
void main(void)
{     .
      TmrReset(0);      /* Reinicia el Timer 0  */
      .
}
```

TmrChk( )
INT16U TmrChk(INT8U n)

`TIMChk()` verifica la cuenta del respectivo temporizador.

Argumentos
n es el número del temporizador correspondiente. 8 > n > 0.
Valor de Retorno
El contador del respectivo temporizador en un valor de 16-bits.
Notas/Advertencias
Si el tiempo fue vencido, retorna un valor de cero (0).
Ejemplo

```
void main(void)
{     .
      tiempo = TmrChk(0);/* Tiempo del Tmr 0 */
      .
}
```

TmrSetT( )

void TmrSetT(INT8U n, INT8U dseg)

`TmrSetT()` inicia el temporizador “n” con un nuevo valor en decisegundos.

Argumentos

n es el número del temporizador correspondiente. 8 > n > 0.

dseg es la cantidad de decisegundos de cuenta fuera de temporizado.

Valor de Retorno

Ninguno

Notas/Advertencias

`TmrSetT()` no inicia la cuenta de temporizado.

Ejemplo

```
void main(void)
{     .
      TmrSetT(0, 30);   /* Tmr 0 = 30 dseg
*/
      .
}
```

TmrSetST( )

void TmrSetT(INT8U n, INT8U seg, INT8U dseg)

`TmrSetMT()` inicia el temporizador “n” con los segundos y decisegundos correspondientes.

Argumentos

n es el número del temporizador correspondiente. 8 > n > 0.

seg es la cantidad de segundos de cuenta fuera de temporizado.

dseg es la cantidad de decisegundos de cuenta fuera de temporizado.

Valor de Retorno

Ninguno

Notas/Advertencias

`TmrSetST()` no inicia la cuenta de temporizado.

Ejemplo

```
void main(void)
{     .
      TmrSetMT(0, 1, 30);     /* Tmr 0 = 1.3 seg */
      .
}
```

TmrSetMST( )

void TmrSetT(INT8U n, INT8U min, INT8U seg, INT8U dseg)

`TmrSetMST()` inicia el temporizador “n” con un nuevo valor en minutos, segundos y decisegundos.

Argumentos

n es el número del temporizador correspondiente. 8 > n > 0.
min es la cantidad de minutos de cuenta fuera de temporizado.
seg es la cantidad de segundos de cuenta fuera de temporizado.
dseg es la cantidad de decisegundos de cuenta fuera de temporizado.

Valor de Retorno

Ninguno

Notas/Advertencias

`TmrSetMST()` no inicia la cuenta de temporizado.

Ejemplo

```
void main(void)
{     .
      TmrSetT(2, 50, 30); /* Tmr 0 = 170.3 seg */
      .
}
```

APÉNDICE K

RUTINAS REUSABLES DE PANTALLA DE CRISTAL LÍQUIDO (LCD)

Una pantalla de cristal líquido o LCD, de caracteres, servirá entonces para desplegar información al usuario de las condiciones que se dan, sensan o podrán suceder. Dichas rutinas permiten configurar la pantalla en modos de 8 y 4 bits, minimizando así la cantidad de pines utilizados y así simplificar el diseño. También, dichas rutinas permiten desplegar en una posición fija de la pantalla un caracter, una cadena de caracteres, borrar la pantalla, en fin, toda una unidad especial a las características básicas importantes para cualquier LCD de caracteres que simplifica el diseño.

LCDInit( )
void LCDInit(void)

`LCDInit()` es una rutina que inicializa la pantalla en modo de 8 bits, 2 líneas, matriz de 5 × 11 puntos (si lo posee) y modo de incremento de caracteres. Usted debe llamar `LCDInit()` antes de utilizar la LCD.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Para cambiar los diferentes modos de 1 línea, sin parpadeo, sin cursor, el programador debe entrar a la rutina y remover o agregar macros correspondientes a las acciones.
Ejemplo

```
void main(void)
{     .
      LCDInit();  /* Inicializa la pantalla */
      .
}
```

LCDInit

`LCDInit` es una rutina que inicializa la pantalla en modo de 8 bits, 2 líneas, matriz de 5 × 11 puntos (si lo posee) y modo de incremento de caracteres. Usted debe llamar `LCDInit` antes de utilizar la LCD.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Para cambiar los diferentes modos de 1 línea, sin parpadeo, sin cursor, el programador debe entrar a la rutina y remover o agregar macros correspondientes a las acciones.
Ejemplo

```
START
      .
      jsr LCDInit       ; Inicializa la pantalla
      .
      .
```

Nota: Para cambiar a modo de 8 bits a 4 bits alterar en su respectivo archivo el macro LCD_8BIT_MODE_EN

LCDClr( )
void LCDClr(void)

`LCDClr()` borra la pantalla de cristal líquido y retorna a la posición inicial extrema izquierda el cursor.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
void main(void)
{     LCDInit();        /* Inicializa la pantalla*/
      for(;;) {
            LCDClr();   /* Borra la pantalla    */
            .
      }
}
```

LCDClr

`LCDClr` borra la pantalla de cristal líquido y retorna a la posición inicial extrema izquierda el cursor.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
START
      .
      jsr LCDInit       ; Inicializa la pantalla
      .
      jsr LCDClr        ; Borra la pantalla
      .
```

LCDClrRow( )
void LCDClr(INT8U row)

`LCDClrRow()` sitúa el cursor en la línea y rellena del carácter ASCII 0x20 que representa un espacio en blanco, luego, sitúa el cursor al inicio de la línea.

Argumentos
row es la fila a ser borrada
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
void main(void)
{     LCDInit();  /* Inicializa la pantalla    */
      for(;;) {
            LCDClrRow(1);     /* Borra lín. 1   */
            .
      }
}
```

LCDClrRow

`LCDClrRow` sitúa el cursor en la línea y rellena del carácter ASCII $20 que representa un espacio en blanco, luego, sitúa el cursor al inicio de la línea.

Argumentos
A = row es la fila a ser borrada
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
START
      .
      jsr LCDInit       ; Inicializa la pantalla
      lda #3            ; Fila 3
      jsr LCDClrRow     ; Borra fila
      .
```

LCDWrChar( )

void LCDWrChar(INT8U data)

`LCDWrChar()`, imprime un caracter en la pantalla independientemente de los límites o de donde se esté posicionado.

Argumentos
data es el carácter a imprimir en la posición actual de la pantalla.
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
void main(void)
{     LCDInit();  /* Inicializa la pantalla     */
      for(;;) {
            LCDWrChar('a');   /* Imprime carácter*/
            .
      }
}
```

LCDWrChar

`LCDCWrChar` imprime un caracter en la pantalla independientemente de los límites o de donde se esté posicionado.

Argumentos
A = data es el carácter a imprimir en la posición actual de la pantalla.
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
START
      .
      jsr LCDInit       ; Inicializa la pantalla
      lda #'B'          ; Carga caracter a imprimir
      jsr LCDWrChar     ; Borra la pantalla
      .
```

LCDWrMsg( )

void LCDWrMsg(INT8U *pdata)

Para escribir un mensaje en la posición actual, usted debe llamar a la función `LCDWrMsg()`. `LCDWrMsg()` imprime una cadena.

Argumentos
pdata es el puntero del mensaje a enviar.
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
void main(void)
{     LCDInit();  /* Inicializa la pantalla     */
      LCDWrMsg("HOLA"); /* Imprime cadena       */
      for(;;) {
            .
      }
}
```

LCDWrMsg

Para escribir un mensaje en la posición actual, usted debe llamar a la función `LCDWrMsg`. `LCDWrMsg` imprime una cadena.

Argumentos
H:X = pdata es el puntero del mensaje a enviar a la pantalla.
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
START
      .
      jsr LCDInit         ; Inicializa la pantalla
      ldhx #MENSAJE       ; Carga caracter a
imprimir
      jsr LCDWrChar       ; Borra la pantalla
      .

MENSAJE
      db    'ALBATROS',0 ; Cadena de caracteres
```

LCDWrXY( )

void LCDWrXY(INT8U row, INT8U col, INT8U data)

Si usted desea escribir un solo caracter en la posición actual de la pantalla, en alguna posición específica, utilice la función `LCDWrXY()`.

Argumentos
row es la fila en la cual se desea ubicar el cursor.
col es la columna en la cual se desea ubicar el cursor.
data es el caracter a escribir en pantalla.
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
void main(void)
{     LCDInit();  /* Inicializa la pantalla     */
      LCDWrXY(2, 3, 'A');/* (2, 3) Imp. 'A'     */
      for(;;) {
            .
      }
}
```

LCDWrXY

Si usted desea escribir un solo caracter en la posición actual de la pantalla, en alguna posición específica, utilice la función `LCDWrXY`.

Argumentos
A (push) = data es el carácter a escribir en pantalla.
A = row es la fila en la cual se desea ubicar el cursor.
X = col es la columna en la cual se desea ubicar el cursor.
Valor de Retorno
Ninguno
Notas/Advertencias
Luego de la rutina usted debe reposicionar la pila.
Ejemplo

```
START .
      jsr LCDInit       ; Inicializa pantalla
      lda #'@'          ; Carga carácter a imprimir
      psha              ; Empuja a pila
      lda #2T           ; Fila 2
      ldx #3T           ; Column 3
      jsr LCDWrXY       ; Escribe el carácter en XY
      ais #1            ; Reposiciona pila
```

LCDWrMsgXY( )

void LCDWrMsgXY(INT8U row, INT8U col, INT8U *pdata)

Si usted desea escribir una cadena de caracteres sobre una posición en específica, ud. debe utilizar la rutina `LCDWrMsgXY()`.

Argumentos
row es la fila en la cual se desea ubicar el cursor.
col es la columna en la cual se desea ubicar el cursor.
pdata es la posición o puntero de la cadena del mensaje a enviar.
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
void main(void)
{     LCDInit();  /* Inicializa la pantalla     */
      LCDWrMsgXY(0, 8, 'Beta X');
      .           /* Fila 0 columna 8, imp. Cad. */
}
```

LCDWrMsgXY

Si usted desea escribir una cadena de caracteres sobre una posición en específica, ud. debe utilizar la rutina `LCDWrMsgXY`.
Argumentos
A = row es la fila en la cual se desea ubicar el cursor.
X (PUSH) = col es la columna en la cual se desea ubicar el cursor.
H:X = pdata es el puntero del mensaje a escribir.
Valor de Retorno
Ninguno
Notas/Advertencias
Luego de la rutina usted debe reposicionar la pila.
Ejemplo

```
START jsr LCDInit       ; Inicializa la pantalla
      lda #2T           ; Fila 2
      ldx #0T           ; Columna 0
      pshx              ; Empuja columna
      ldhx #MENSAJE     ; Apunta al mensaje
      jsr LCDWrMsgXY    ; Escribe el mensaje
      ais #1            ; Reposiciona la pila
```

LCDSel( )
void LCDSel(BOOLEAN cmd)

`LCDSel()` selecciona entre el modo de escritura y el modo de comandos, levanta la línea de control RS para modo de escritura y baja la línea RS para modo de comandos

Argumentos
cmd es el valor binario entre escritura `LCD_RS_HIGH` ó 1 y modo de comandos `LCD_RS_LOW` ó 0.
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
void main(void)
{     LCDInit();              /* Inic. Pant.    */
      LCDSel(LCD_RS_HIGH)     /* Modo escritura */
      .
}
```

LCDSel

`LCDSel` selecciona entre el modo de escritura y el modo de comandos, levanta la línea de control RS para modo de escritura y baja la línea RS para modo de comandos.

Argumentos
A = cmd es el valor binario entre escritura `LCD_RS_HIGH` ó 1 y modo de comandos `LCD_RS_LOW` ó 0.
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
START jsr LCDInit       ; Inicializa la pantalla
      .
      clra              ; LCD_RS_LOW (0)
      jsr LCDSel        ; Reposiciona la pila
```

LCDWr( )
void LCDSel(INT8U data)

Ejecuta un comando o escribe el carácter en la pantalla. Todo depende del estado del pin RS dado por la función `LCDSel()`.

Argumentos
data es el dato o carácter a depositar en el bus de datos D[0:7] de la pantalla LCD para luego ejecutarse.
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
void main(void)
{     LCDInit();  /* Inic. Pantalla       */
      LCDWr('A'); /* Escribe              */
      .
}
```

LCDWr

Ejecuta un comando o escribe el carácter en la pantalla. Todo depende del estado del pin RS dado por la función `LCDSel()`.

Argumentos
A = data es el dato o carácter a depositar en el bus de datos D[0:7] de la pantalla LCD para luego ejecutarse.
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
START jsr LCDInit       ; Inicializa la pantalla
      .
      lda #LCD_8_BIT    ; Comando para 8 bits
      jsr LCDWr         ; Selecciona comando
```

LCDSetXY( )

void LCDSetXY(INT8U row, INT8U col)

Posiciona el cursor en algún lugar de la pantalla sin salirse de los límites de la misma.

Argumentos
row es la fila en la cual se desea ubicar el cursor.
col es la columna en la cual se desea ubicar el cursor.
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
void main(void)
{     LCDInit();        /* Inic. Pantalla      */
      LCDSetXY(1,10);   /* Posiciona en X-Y    */
      .
}
```

LCDSetXY

Posiciona el cursor en algún lugar de la pantalla sin salirse de los límites de la misma.

Argumentos
A = row es la fila en la cual se desea ubicar el cursor.
X = col es la columna en la cual se desea ubicar el cursor.
Valor de Retorno
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
START jsr LCDInit       ; Inicializa la pantalla
      lda #3T           ; Fila 3
      ldx #1T           ; Columna 1
      jsr LCDSetXY      ; Posiciona
```

APÉNDICE L

RUTINAS REUSABLES DE TECLADO MATRICIAL DE “M” X “N” TECLAS (KBI – KBD)

Si en su sistema necesita captar datos de entrada, probablemente necesite conectarle al dispositivo un teclado matricial de teclas. Recomendable sería que el teclado que se conecte, fuese mayor a 2 x 2, es decir, 2 filas y 2 columnas, pues, no se tendría sentido tener una rutina tan complicada para detectar el estado de solo cuatro teclas. La rutina fue configurada especialmente para minimizar el consumo de corriente que consumen los resistores de pull-up internos del microcontrolador, así, se podrán utilizar para aplicaciones de baterías.

Por el momento, se provee esta rutina para microcontroladores con más de un puerto KBI, el cual es utilizado como base para estas rutinas, y debido a que el microcontrolador JK3 utiliza su puerto KBI comprometido con el oscilador del sistema, este no podrá ser utilizado con estas rutinas por el momento. Futuras alteraciones de esta rutina incluirán el código para aquellas que no poseen o tengan comprometido el módulo.

KbdInit( )

void KbdInit(BOOLEAN mode)

Inicializa el teclado para decodificación. Esta rutina debe ser llamada antes de empezar a utilizar el teclado. Ud. Puede elegir entre el modo de una sola tecla (0) o autorepeticiones (1) También la misma inicializa el buffer anillo de teclado estilo FIFO.

Argumentos

mode es el valor booleano que elige el tipo de detección de teclas, cero (0) para una sola tecla o uno (1) para modo continuo.

Valor de Retorno

Ninguno

Notas/Advertencias

Inicializa el buffer de teclas. Para configurar que cantidad de teclas detectar, modifique los macros:

```
KBD_ROW_PORT      /* Puerto de Filas */
KBD_ROW_DDR       /* Reg. de Filas   */
KBD_MAX_ROW       /* Máx. de Filas   */

KBD_COL_PORT      /* Puerto de Col.  */
KBD_COL_DDR       /* Reg. de Col.    */
KBD_MAX_COL       /* Máx. de Col.    */
```

Ejemplo

```
void main(void)
{     .
      KbdInit(0); /* Teclado iniciado, 1 a vez  */
      .
}
```

KbdInit

Inicializa el teclado para decodificación. Esta rutina debe ser llamada antes de empezar a utilizar el teclado. Ud. Puede elegir entre el modo de una sola tecla (0) o autorepeticiones (1) También la misma inicializa el buffer anillo de teclado estilo FIFO.

Argumentos

X = mode es el valor booleano que elige el tipo de detección de teclas, cero (0) para una sola tecla o uno (1) para modo continuo.

Valor de Retorno

Ninguno

Notas/Advertencias

Inicializa el buffer de teclas. Para configurar la cantidad de teclas a detectar, modifique los macros:

```
KBD_ROW_PORT      ; Puerto de Filas
KBD_ROW_DDR       ; Registro de Filas
KBD_MAX_ROW       ; Máximo de Filas

KBD_COL_PORT      ; Puerto de Columnas
KBD_COL_DDR       ; Registro de
Columnas
KBD_MAX_COL       ; Máximo de Columnas
```

Ejemplo

```
START .
      clrx              ; 1 tecla a la vez
      jsr KbdInit       ; Inicializa teclado
      .
```

Nota: Para definir la cantidad de caracteres a almacenar en el buffer de teclado, modifique el macro: KBD_MAX_KEY

KbdScan( )
void KbdScan(void)

La subrutina `KbdScan( )` utiliza dos decodificaciones para verificar si una tecla fue detectada y si existe, la almacena la última tecla un buffer de teclado. Por el momento, ud. debe utilizar esta subrutina en la interrupción de teclado.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
La rutina almacena en la variable `KbdLastKey` la última tecla aún si en el buffer no existiese espacio. Reconoce automáticamente la interrupción de teclado.
Ejemplo

```
interrupt 15 void KBIH(void)
{
      KbdScan();        /* Decodifica y retorna */
}
```

KbdScan

La subrutina `KbdScan` utiliza dos decodificaciones para verificar si una tecla fue detectada y si existe, la almacena la última tecla un buffer de teclado. Por el momento, ud. debe utilizar esta subrutina en la interrupción de teclado.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
La rutina almacena en la variable `KbdLastKey` la última tecla aún si en el buffer no existiese espacio. Reconoce automáticamente la interrupción de teclado.
Ejemplo

```
KBIH
      jsr KbdScan       ; Decodifica y retorna
      rti               ; Retorna de la subrutina
```

KbdChkRow( )

INT8U KbdChkRow(void)

Verifica el estado de las filas y devuelve la fila presionada. Si no existe ninguna, devuelve un caracter de error.

Argumentos
Ninguno
Valor de Retorno
Retorna el índice de la fila la cual fue detectada un estado bajo. De lo contrario, devuelve un caracter de error (255_{10}).
Notas/Advertencias
Ninguna
Ejemplo

```
void main (void)
{     INT8U row;        /* Var. Temp. Fila      */

      row = KbdChkRow();/* retorna la fila      */
}
```

KbdChkRow

Verifica el estado de las filas y devuelve la fila presionada. Si no existe ninguna, devuelve un caracter de error.

Argumentos
Ninguno
Valor de Retorno
Retorna al acumulador el índice de la fila la cual fue detectada un estado bajo. De lo contrario, devuelve un caracter de error (255_{10}).
Notas/Advertencias
Ninguna
Ejemplo

```
START .
      .
      jsr KbdChkRow     ; Busca la fila presionada
      .
```

KbdChkCol( )

INT8U KbdChkRow(void)

Verifica el estado de las columnas, dependiendo del estado de las filas y devuelve la columna presionada. Si no existe ninguna, devuelve un caracter de error.

Argumentos
Ninguno
Valor de Retorno
Retorna el índice de la columna la cual fue detectada por una fila en un estado bajo. De lo contrario, devuelve un caracter de error (255_{10}).
Notas/Advertencias
Ninguna
Ejemplo

```
void main (void)
{     INT8U col;        /* Var. Temp. Col.      */

      col = KbdChkCol();/* retorna la col.      */
}
```

KbdChkCol

Verifica el estado de las columnas, dependiendo del estado de las filas y devuelve la columna presionada. Si no existe ninguna, devuelve un caracter de error.

Argumentos
Ninguno
Valor de Retorno
Retorna al registro X la columna la cual fue detectada por una fila en un estado bajo. De lo contrario, devuelve un caracter de error (255_{10}).
Notas/Advertencias
Ninguna
Ejemplo

```
START .
      .
      jsr KbdChkCol     ; Busca la columna pres.
      .
```

KbdBufClr( )
void KbdBufClr(void)

Vacía el buffer de teclado, recoge los caracteres y empieza como un buffer vacío.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
void main (void)
{     .
      .
      KbdBufClr();      /* Vacía el buffer      */
}
```

KbdBufClr

Vacía el buffer de teclado, recoge los caracteres y empieza como un buffer vacío.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
START .
      .
      jsr KbdBufClr     ; Vacía el buffer
      .
```

KbdBufGet( )

INT8U KbdBufGet(void)

Recoge un carácter del buffer. `KbdBufGet()`, adquiere un carácter, si y solo si existe una tecla en el buffer, de lo contrario, retorna un 0.

Argumentos
Ninguno
Valor de Retorno
Adquiere el primer caracter tecleado del buffer, si existe.
Notas/Advertencias
Retorna un valor nulo (0) si el buffer está vacío.
Ejemplo

```
void main (void)
{     INT8U key;        /* Var. Temp. Tecla     */
      .
      key = KbdBufGet();/* Adquiere tecla       */
}
```

KbdBufGet

Recoge un carácter del buffer. KbdBufGet, adquiere un carácter, si y solo si existe una tecla en el buffer, de lo contrario, retorna un 0.

Argumentos
Ninguno
Valor de Retorno
Adquiere el primer caracter tecleado del buffer, y lo deposita en el acumulador A si existe.
Notas/Advertencias
Retorna un valor nulo (0) al acumulador si el buffer está vacío.
Ejemplo

```
START .
      .
      jsr KbdBufGet     ; Recoge carácter del
                        ; buffer
      .
```

APÉNDICE M

M.1 RUTINAS REUSABLES DE MULTIPLEXIÓN DE LEDS (LED)

Los dispositivos de salida más utilizados en sistemas embebidos son los LEDs o diodos emisores de luz. Una de sus variantes de presentación se llaman siete segmentos, que contienen siete LEDs en un dispositivo, pero ordenados de tal forma que se pueden desplegar números. Las rutinas presentadas necesitan ciertas inicializaciones del sistema, las cuales están especificadas dentro del archivo. Utilizando la multiplexión, reducimos notablemente el consumo total y mejoramos el tiempo de vida de nuestra aplicación.

LEDDispInit( )

void LEDDispInit(void)

Inicializa el módulo de multiplexión de siete segmentos. `LEDDispInit()` debe ser invocado antes de llamar cualquier rutina de este módulo, antes de usar el módulo deben iniciarse los puertos a utilizar, ver advertencias.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Se debe tomar en consideración inicializar los macros del módulo para una operación exitosa:

```
      LED_MUX_PORT      /* Puerto del Multiplexor*/
      LED_MUX_DDR       /* Registro del Puerto del
                           Multiplexor          */
      LED_MAX_DISP      /* Máximo de caracteres a
                           multiplexar          */
      LED_SSEG_PORT     /* Puerto Decodificador  */
      LED_SSEG_DDR      /* Registro del Puerto del
                           Decodificador        */
      LedMuxCtr         /* Variable de control del
                           Multiplexor          */
```

Ejemplo

```
void main(void)
{     .
      LEDDispInit();/* Inicializa pantallas    */
      .
}
```

LEDDispInit

Inicializa el módulo de multiplexión de siete segmentos. `LEDDispInit` debe ser invocado antes de llamar cualquier rutina de este módulo, antes de usar el módulo deben iniciarse los puertos a utilizar, ver advertencias.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Se debe tomar en consideración inicializar los macros del módulo para una operación exitosa:

```
      LED_MUX_PORT      ; Puerto del Multiplexor
      LED_MUX_DDR       ; Registro del Puerto del
                          Multiplexor
      LED_MAX_DISP      ; Máximo de caracteres a
                          multiplexar
      LED_SSEG_PORT     ; Puerto del Decodificador
      LED_SSEG_DDR      ; Registro del Puerto del
                        ; Decodificador
      LedMuxCtr         ; Variable de control del
                          Multiplexor
```

Ejemplo

```
START .
                        ; Otras instrucciones
      jsr LEDDispInit   ; Inicializa pantallas
      .                 ; seguir
```

Nota: El máximo de pantallas debe llegar a ser menor o igual a ocho pantallas. Configurar el nibble a utilizar (alto o bajo).

LEDDispOff( )
void LEDDispOff(void)

`LEDDispOff()` es la rutina encargada a apagar el siete segmentos. Envía un uno lógico a todos los caracteres para apagar las pantallas, de esta forma no encenderán. Esta función es utilizada para evitar el efecto "fantasma" entre escritura de una pantalla y otra.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
void main(void)
{     .
      LEDDispInit(); /* Inicializa pantallas    */
      LEDDispOff();  /* Apaga las pantallas     */
      .
}
```

LEDDispOff

`LEDDispOff` es la rutina encargada a apagar el siete segmentos. Envía un uno lógico a todos los caracteres para apagar las pantallas, de esta forma no encenderán. Esta función es utilizada para evitar el efecto "fantasma" entre escritura de una pantalla y otra.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
START .
      jsr LEDDispInit   ; Inicializa pantallas
      jsr LEDDispOff    ; Apaga las pantallas
      .                 ; seguir
```

LEDDispMuxHandler( )
void LEDDispMuxHandler(void)

Esta función es la encargada de multiplexar las pantallas, seleccionando el siguiente dígito a encender. `DispMuxHandler()` debe ser ubicado en la subrutina de interrupción del temporizador y dicho temporizador debe estar configurado a una rata de N $\times$ 60 Hz, en donde N representa la cantidad de pantallas a multiplexar, N $\leq$ 8.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
interrupt 6 void TIM1OFH(void)
{
      LEDDispMuxHandler(); /*  Multiplexa y
                              reconoce la int. */
      .
}
```

LEDDispMuxHandler

Esta función es la encargada de multiplexar las pantallas, seleccionando el siguiente dígito a encender. `DispMuxHandler` debe ser ubicado en la subrutina de interrupción del temporizador y dicho temporizador debe estar configurado a una rata de N $\times$ 60 Hz, en donde N representa la cantidad de pantallas a multiplexar, N $\leq$ 8.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
TIM1OFL

      jsr LEDDispMuxHandler   ; Multiplexa y
                              ; reconoce la int.
      rti                     ; Retorna
```

LEDDispStr( )

void LEDDispStr(INT8U dig, INT8U *pdata)

Cuando se invoca la función `LEDDispStr()`, se imprime desde el dígito seleccionado, una cadena ASCII definida en una ubicación de la memoria. Debe tener especial cuidado con estos caracteres a desplegar, pues no todos tienen su equivalente de conversión a ASCII.

Argumentos

dig es el dígito desde donde se desplegará la información en la pantalla.

pdata es el puntero de la cadena a desplegar.

Valor de Retorno

Ninguno

Notas/Advertencias

La información a desplegar tiene que ser ASCII obligatoriamente y no todos los caracteres ASCII pueden ser desplegados en la pantalla.

Ejemplo

```
void main(void)
{     .
      LEDDispInit();/* Inicializa pantallas    */
      LEDDispStr(2, "HOLA");  /* Desde tercer
                                 dígito, imprime
                                 Hola           */
      .
}
```

LEDDispStr

Cuando se invoca la función `LEDDispStr`, se imprime desde el dígito seleccionado, una cadena ASCII definida en una ubicación de la memoria. Debe tener especial cuidado con estos caracteres a desplegar, pues no todos tienen su equivalente de conversión a ASCII.

Argumentos

A = dig es el dígito desde donde se desplegará la información en la pantalla.

H:X = pdata es el puntero de la cadena a desplegar.

Valor de Retorno

Ninguno

Notas/Advertencias

La información a desplegar tiene que ser ASCII obligatoriamente y no todos los caracteres ASCII pueden ser desplegados en la pantalla.

Ejemplo

```
START .
      jsr LEDDispInit   ; Inicializa pantallas
      clra              ; Dígito 0
      ldhx #MENSAJE     ; Puntero del mensaje
      jsr LEDDispStr    ; Despliega desde dígito 0
                        ; la palabra HOLA

MENSAJE
db 'HOLA',0 ; Mensaje a desplegar
```

M.2 RUTINAS REUSABLES DE MULTIPLEXIÓN DE SIETE SEGMENTOS (SSEG)

Si desea una rutina más conveniente, sencilla de utilizar y menos problemática que las anteriores, se recomienda utilizar las rutinas de siete segmentos. Esta rutina utiliza hardware específico, un multiplexor 74LS138 y un decodificador de BCD a 7 segmentos 74LS47. De esta forma solo se estará limitado a desplegar números naturales entre 0 a 9, con la misma capacidad de las rutinas anteriores de desplegar hasta 8 dígitos. Recomendaciones a esta rutina están utilizar los módulos matemáticos de la siguiente sección del apéndice, apéndice N, para tener mayor compatibilidad a desplegar, por ejemplo, la información del ADC convertida a un número natural.

SSegDispInit( )
void SSegDispInit(void)

Inicializa el módulo de multiplexión de siete segmentos. `SSegDispInit()` debe ser invocado antes de llamar cualquier rutina de este módulo, antes de usar el módulo deben iniciarse los puertos a utilizar, ver advertencias.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Se debe tomar en consideración inicializar los macros del módulo para una operación exitosa:

```
SSEG_MUX_PORT     /* Puerto del Multiplexor     */
SSEG_MUX_DDR      /* Registro del Puerto del
                     multiplexor                */
SSEG_MAX_DISP     /* Máximo de caracteres a
                     multiplexar                */
SSEG_DEC_PORT     /* Puerto del Decodificador   */
SSEG_DEC_DDR      /* Registro del Puerto del
                     decodificador              */
SsegMuxCtr        /* Variable de control del
                     multiplexor                */
```

Ejemplo

```
void main(void)
{     .
      SSegDispInit();/* Inicializa pantallas    */
      .
}
```

SSegDispInit

Inicializa el módulo de multiplexión de siete segmentos. SSegDispInit debe ser invocado antes de llamar cualquier rutina de este módulo, antes de usar el módulo deben iniciarse los puertos a utilizar, ver advertencias.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Se debe tomar en consideración inicializar los macros del módulo para una operación exitosa:

```
SSEG_MUX_PORT     ; Puerto del Multiplexor
SSEG_MUX_DDR      ; Registro del Puerto del
                  ; Multiplexor
SSEG_MAX_DISP     ; Máximo de caracteres a
                  ; multiplexar
SSEG_DEC_PORT     ; Puerto del Decodificador
SSEG_DEC_DDR      ; Registro del Puerto del
                  ; Decodificador
SsegMuxCtr        ; Variable de control del
                  ; Multiplexor
```

Ejemplo

```
START .
                        ; Otras instrucciones
      jsr SSegDispInit  ; Inicializa pantallas
      .                 ; seguir
```

Nota: El máximo de pantallas debe llegar a ser menor o igual a ocho pantallas. Configurar el nibble a utilizar (alto o bajo).

SSegDispOff( )
void SSegDispOff(void)

`SSegDispOff()` es la rutina encargada a apagar el siete segmentos. Envía un uno lógico a todos las entradas de conteo del 74LS47 para apagar las pantallas, de esta forma no encenderán. Esta función es utilizada para evitar el efecto "fantasma" entre escritura de una pantalla y otra.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
void main(void)
{     .
      SSegDispInit(); /* Inicializa pantallas  */
      SSegDispOff();  /* Apaga las pantallas   */
      .
}
```

SSegDispOff

`SSegDispOff` es la rutina encargada a apagar el siete segmentos. Envía un uno lógico a todas las entradas de conteo del 74LS47 para apagar las pantallas, de esta forma no encenderán. Esta función es utilizada para evitar el efecto "fantasma" entre escritura de una pantalla y otra.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
START .
      jsr SSegDispInit  ; Inicializa pantallas
      jsr SSegDispOff   ; Apaga las pantallas
      .                 ; seguir
```

SSegDispMuxHandler( )
void SSegDispMuxHandler(void)

Esta función es la encargada de multiplexar las pantallas, seleccionando el siguiente dígito a encender. `SSegMuxHandler( )` debe ser ubicado en la subrutina de interrupción del temporizador y dicho temporizador debe estar configurado a una rata de N $\times$ 60 Hz, en donde N representa la cantidad de pantallas a multiplexar, N $\leq$ 8.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
interrupt 6 void TIM1OFH(void)
{
      SSegDispMuxHandler(); /*  Multiplexa y
                              reconoce la int. */
      .
}
```

SSegDispMuxHandler

Esta función es la encargada de multiplexar las pantallas, seleccionando el siguiente dígito a encender. SSegMuxHandler debe ser ubicado en la subrutina de interrupción del temporizador y dicho temporizador debe estar configurado a una rata de N $\times$ 60 Hz, en donde N representa la cantidad de pantallas a multiplexar, N $\leq$ 8.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
TIM1OFL

      jsr SSegDispMuxHandler  ; Multiplexa y
                              ; reconoce la int.
      rti                     ; Retorna
```

APÉNDICE N

RUTINAS REUSABLES MATEMÁTICAS (MATH)

Ciertas ocasiones nos preguntamos como realizar el despliegue de información, externamente, hacia un dispositivo de salida no complejo, como lo son los siete segmentos. Tenemos que convertir un número hexadecimal a un número decimal y desplegarlo, convertir un conjunto de cuatro bits y convertirlos a ASCII, convertir u número hexadecimal a una cadena de caracteres y todo esto resulta bastante tedioso en realizar si a lo que se quiere llegar es a la solución del problema. Las rutinas matemáticas predispuestas son una solución parcial al problema, no son los códigos fuentes más eficientes en cuanto a nivel de programación, pero resuelven el problema parcial de estar implementando matemática, especialmente en lenguaje ensamblador.

MathHex2Dec( )

void MathHex2Dec(INT8U val, INT8U *pdata)

Rutina matemática que convierte un caracter hexadecimal de 8 bits a su equivalente decimal de tres posiciones. Convierte un número a su equivalente entre 0 a 255.

Argumentos
val es el número entero de ocho bits a convertir a decimal.
pdata es el puntero en donde se desea almacenar la información.
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
void main(void)
{
      INT8U dec[3];           /* Número Decimal */
      .
      MathHex2Dec(0x7F, dec); /* A convertir a
                                 decimal       */
      .
}
```

MathHex2Dec

Rutina matemática que convierte un caracter hexadecimal de 8 bits a su equivalente decimal de tres posiciones. Convierte un número a su equivalente entre 0 a 255.

Argumentos
A = val es el número entero de ocho bits a convertir a decimal.
H:X = pdata es el puntero en donde se desea almacenar la información.
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
START .
      lda #$AB          ; Número a convertir
      ldhx #$0081       ; Posición del decimal
      jsr MathHex2Dec   ; Convierte a decimal
      .                 ; Seguir
```

MathNibble2Ascii( )

INT8U MathNibble2Ascii(INT8U nibble)

Rutina matemática que convierte un caracter hexadecimal, parte baja, a su correspondiente número ASCII equivalente y retorna un número entero de 8 bits.

Argumentos
nibble es entero a transformar a ASCII.
Valor de Retorno
Ninguno
Notas/Advertencias
Solamente se transforma la parte baja del número a su correspondiente número ASCII.
Ejemplo

```
void main(void)
{
      INT8U ascii;      /* Número ASCII         */
      .
      ascii = MathNibble2Ascii(0x7F);
                        /* A convertir a ASCII  */
      .
}
```

MathNibble2Ascii

Rutina matemática que convierte un caracter hexadecimal, parte baja, a su correspondiente número ASCII equivalente y retorna un número entero de 8 bits.

Argumentos
nibble es entero a transformar a ASCII.
Valor de Retorno
Ninguno
Notas/Advertencias
Solamente se transforma la parte baja del número a su correspondiente número ASCII.
Ejemplo

```
START .
      lda #$20                ; Número a convertir
      jsr MathNibble2Ascii    ; Convierte a ASCII
      .                       ; A = #$30
```

MathHex2Str ( )

void MathHex2Str(INT8U val, INT8U *pdata)

Rutina matemática que convierte un caracter hexadecimal de 8 bits a su equivalente en cadena ASCII.

Argumentos
val es entero a transformar a ASCII.
pdata es el puntero en donde se almacena el carácter ASCII
Valor de Retorno
Ninguno
Notas/Advertencias
Adjuntado un caracter de fin de cadena (0).
Ejemplo

```
void main(void)
{
      INT8U str[4];     /* Cadena ASCII         */
      .
      ascii = MathHex2Str(0xC0, str);
                        /* A convertir a cadena
                  ASCII */
      .
}
```

MathHex2Str

Rutina matemática que convierte un caracter hexadecimal de 8 bits a su equivalente en cadena ASCII.

Argumentos
A = val es entero a transformar a ASCII.
H:X = pdata es el puntero en donde se almacena el carácter ASCII
Valor de Retorno
Ninguno
Notas/Advertencias
Adjuntado un caracter de fin de cadena.
Ejemplo

```
START .
      lda #$20                ; Número a convertir
      ldhx #$009B             ; Posición en donde
                              ; almacenar
      jsr MathHex2Str         ; Convierte a cadena
                              ; ASCII
      .                       ; Seguir
```

APÉNDICE O

RUTINAS REUSABLES PARA “PAD” DE “PLAYSTATION” (PSX)

Como en algunas ocasiones no es de difícil acceso conseguir un teclado, se propuso utilizar un control de “playstation” el cual provee una gran cantidad de botones y dos pares de “joystick” analógicos, con él, y ciertos aditivos adicionales, se puede hacer un pequeño control remoto para un robot móvil de tipo casero, también, si se conoce algo de audio y video, se pudiese hacer un pequeño simulador de PONG, un viejo juego arcaico de eras anteriores, este fue uno de los primeros juegos a PC que existieron y era disponible solo en “mainframe computers”, tiempo atrás.

Las rutinas reusables del “Pad”, permiten inicializar el “pad”, adquirir un caracter del “buffer”, limpiar el “buffer” y adquirir una trama del “pad”, todo esto es posible gracias a este módulo que permite la compatibilidad inmediata entre los microcontroladores, el único requisito indispensable es poseer al menos un pin KBI del módulo libre para utilizarlo como teclado, requerimiento del “software”.

PSXPadInit( )
void PSXPadInit(BOOLEAN mode)

Inicializa el microcontrolador para adquirir las respectivas señales del pad. Esta rutina debe ser llamada antes de empezar a utilizar el módulo de pad de playstation. Ud. Puede elegir entre el modo de una sola tecla (0) o autorepeticiones (1) También la misma inicializa el buffer anillo de teclado estilo FIFO.

Argumentos
mode es el valor booleano que elige el tipo de detección de teclas, cero (0) para una sola tecla o uno (1) para modo continuo.
Valor de Retorno
Ninguno
Notas/Advertencias
Inicializa el buffer del pad. Para configurar el pad, modifique los macros internos del módulo, estos son algunos de ellos:

```
PSX_PAD_GPIO_PORT /* Puerto General de E/S Pad   */
PSX_PAD_GPIO_DDR  /* Registro General de E/S Pad */
PSX_PAD_CMD_BIT   /* GPIO.Bit0, CMD              */
PSX_PAD_SEL_BIT   /* GPIO.Bit1, SEL              */
PSX_PAD_CLK_BIT   /* GPIO.Bit2, CLK              */
PSX_PAD_ACK_BIT   /* Bit del Puerto de señal de
                     reconoc.                    */
```

Ejemplo

```
void main(void)
{     .
      PSXPadInit(0);    /* Pad iniciado        */
      .
}
```

PSXPadInit

Inicializa el microcontrolador para adquirir las respectivas señales del pad. Esta rutina debe ser llamada antes de empezar a utilizar el módulo de pad de playstation. Ud. Puede elegir entre el modo de una sola tecla (0) o autorepeticiones (1) También la misma inicializa el buffer anillo de teclado estilo FIFO.

Argumentos
A = mode es el valor booleano que elige el tipo de detección de teclas, cero (0) para una sola tecla o uno (1) para modo continuo.
Valor de Retorno
Ninguno
Notas/Advertencias
Inicializa el buffer del pad. Para configurar el pad, modifique los macros internos del módulo, estos son algunos de ellos:

```
PSX_PAD_GPIO_PORT; Puerto General de E/S para el Pad
PSX_PAD_GPIO_DDR  ; Registro General de E/S para el Pad
PSX_PAD_CMD_BIT   ; GPIO.Bit0, CMD
PSX_PAD_SEL_BIT   ; GPIO.Bit1, SEL
PSX_PAD_CLK_BIT   ; GPIO.Bit2, CLK
PSX_PAD_ACK_BIT   ; Bit del Puerto de señal de reconoc.
```

Ejemplo

```
START
      .
      lda #1T           ; autorepeticiones
      jsr PSXPadInit    ; Pad iniciado
      .
```

PSXPadFrame( )
void PSXPadFrame(void)

`PSXPadFrame()` adquiere la trama digital de un control estándar y analógica si es un control con joystick análogos y su opción analógica es activada.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
void main(void)
{     .
      PSXPadInit();     /* Inicia señales       */
      PSXPadFrame();    /* Recibe trama         */
      .
}
```

PSXPadFrame

`PSXPadFrame` adquiere la trama digital de un control estándar y analógica si es un control con joystick análogos y su opción analógica es activada.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
START
      .
      jsr PSXPadInit    ; Inicia señales
      jsr PSXPadFrame   ; Recibe trama
      .
```

PSXPadBufClr( )
void PSXPadBufClr(void)

Vacía el buffer de caracteres adquiridos por el pad, recoge los caracteres y empieza como un buffer vacío.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
void main (void)
{     .
      .
      PSXPadBufClr();   /* Vacía el buffer      */
}
```

PSXPadBufClr

Vacía el buffer de caracteres adquiridos por el pad, recoge los caracteres y empieza como un buffer vacío.

Argumentos
Ninguno
Valor de Retorno
Ninguno
Notas/Advertencias
Ninguna
Ejemplo

```
START
      .
      .
jsr PSXPadBufClr  ; Vacía el buffer
```

PSXPadBufGet( )
INT8U PSXPadBufGet(void)

PSXPadBufGet() adquiere un caracter recibido del pad de playstation. Sus condiciones prescinden en que la rutina PSXPadInit() y PSXPadFrame() se llamen antes de adquirir cualquier caracter.

Argumentos
Ninguno
Valor de Retorno
Retorna un valor de cero (0), no '0' ASCII, si la información no fue leída o un caracter diferente de cero si se ha recibido un byte.
Notas/Advertencias
Ninguna.
Ejemplo

```
INT8U ID    /* Identificador                    */
void main(void)
{     .
      PSXPadInit();     /* Inicia señales       */
      PSXPadFrame();    /* Recibe trama         */
      ID = PSXPadBufGet();    /* Recoge ID      */
      .
}
```

PSXPadBufGet

PSXPadBufGet adquiere un caracter recibido del pad de playstation. Sus condición prescinde en que la rutina PSXPadInit y PSXPadFrame se llamen antes de adquirir cualquier caracter.

Argumentos
Ninguno
Valor de Retorno
A contiene el carácter de retorno. Retorna un valor de cero (0), no '0' ASCII, si la información no fue leída o un caracter diferente de cero si se ha recibido un byte.
X es la condición del caracter recuperado. **X = 1** si se extrajo un caracter válido del buffer. **X = 0** si se extrajo un caracter inválido o ninguno del buffer.
Notas/Advertencias
Ninguna
Ejemplo

```
START .
      jsr PSXPadInit    ; Pad Inicializado
PADFRAME
      jsr PSXPadFrame   ; Recibe trama
      jsr PSXPadBufGet  ; Convierte a decimal
      cbeqx #0,PADFRAME ; Inválido, recibir otra
                        ; vez
      .                 ; Seguir
```