

AIDA08
AMBIENTE INTEGRADO PARA
DESARROLLO Y APRENDIZAJE CON
MICROCONTROLADORES DE LA FAMILIA
68HC908 DE FREESCALE

(Manual de usuario básico)

Por: Antonio Salvá Calleja

Profesor de carrera
Departamento de Ingeniería de Control
División de Ingeniería Eléctrica
Facultad de Ingeniería
UNAM

Enero de 2011

ÍNDICE

1 DESCRIPCIÓN GENERAL DE AIDA08	4
2 TARJETA DESTINO MINICON_08A	6
2.1 Modos de funcionamiento de la tarjeta destino.	6
2.2 Aspectos básicos de la modalidad A de funcionamiento	6
2.3 Aspectos básicos de la modalidad B de funcionamiento.....	6
2.4 Aspectos básicos de la modalidad C de funcionamiento.....	7
2.5 Acciones al RESET de la tarjeta MINICON_08A operando bajo los modos B y C.....	8
2.6 Vectores de usuario y zona de FLASH protegida contra borrado	9
2.7 Ubicación y función de los jumpers sencillos de la tarjeta MINICON_08A	10
2.8 Ubicación y función de los jumpers excluyentes de la tarjeta MINICON_08A	11
2.9 Ubicación y funcionalidad de los conectores de la tarjeta MINICON_08A	11
2.10 Ubicación y funcionalidad de los postes de prueba de la tarjeta MINICON_08A.....	12
2.11 Eliminador de batería recomendado	12
2.12 Inicialización del sistema AIDA08 para uso de éste en una PC determinada.....	12
3 SOFTWARE MANEJADOR PUMMA_08+.....	15
3.1 Aspecto del ambiente de manejo hexadecimal (AMH).....	15
3.2 Menú “Archivo” del AMH.	15
3.3 Menú “Memoria” del AMH.	17
3.4 Menú “Ejecutar” del AMH.....	21
3.5 Menú “Manejo de FEEP” del AMH.....	24
3.6 Menú “Acerca de” del AMH	30
3.7 Descripción básica de las facilidades presentes en el editor de PUMMA_08+	30
3.8 Menú “Archivo” del editor de PUMMA_08+	32
3.9 Menú “Editar” del editor de PUMMA_08+	35
3.10 Menú “Buscar” del editor de PUMMA_08+.....	35
3.11 Menú “Opciones” del editor de PUMMA_08+, (Configuración de procesador destino) ..	39
3.12 Menú “Ayuda” del editor de PUMMA_08+	44
4 INSTALACIÓN E INICIALIZACIÓN DE PUMMA_08+.....	47
4.1 Requerimientos de PUMMA_08+.....	47
4.2 Instalando PUMMA_08+	47
4.3 Inicialización de PUMMA_08+	49
5 ASPECTOS ACERCA DEL ENSAMBLADOR ENS08 PRESENTE EN PUMMA_08+.....	50
5.1 Formato de instrucciones en el programa fuente a ser procesado por ENS08	50
5.2 Pseudo-operaciones soportadas por la versión 2.5 de ENS08.....	51
5.3 Manejo de aritmética con etiquetas y cadenas definidas por pseudosinstrucciones EQU ..	56
6 PROGRAMAS EJEMPLO EN LENGUAJE ENSAMBLADOR.....	57
6.1 Estructura de un programa de aplicación	57
6.2 Diseño de una subrutina de retardo	57
6.3 Ejecución autónoma de programas en la TD cuando ésta opera en los modos B o C.....	62
6.4 Ejecución autónoma de programas en la TD cuando ésta opera en el modo A	63
6.5 Fundamentos del convertidor análogo – digital (CAD)	64
6.6 Interfaz para despliegado alfanumérico presente en la tarjeta MINICON_08A	70

7	COMPILADOR CRUZADO MINIBAS8A.....	86
7.1	Constantes, variables, expresiones y operadores.....	88
7.2	Arreglos	93
7.3	Colocación de comentarios.....	94
7.4	Colocación de más de una sentencia en un renglón	94
7.5	Uso de etiquetas.....	94
7.6	Sentencia input	96
7.7	Sentencia print	98
7.8	Incrustación de código en ensamblador.....	99
7.9	Variables especiales de usuario	100
7.10	Colocación en memoria de constantes de tipo numérico	101
7.11	Uso de subrutinas	103
7.12	Rutinas de servicio de interrupción	103
7.13	Subrutina de biblioteca lee#car.....	104
7.14	Subrutina de biblioteca pon#car	105
7.15	Sentencia cls	106
7.16	Sentencia end.....	106
7.17	Estructuras de programación	107
7.18	Funciones propias de BASIC	114
8	PROGRAMAS EJEMPLO EN MINIBAS8A.....	118
	Ejemplo 8.1. Lectura y escritura de puertos.	119
	Ejemplo 8.2. Rutinas de retraso.....	120
	Ejemplo 8.3. Contador Binario de ocho bits.	121
	Ejemplo 8.4. Uso del Emulador de Terminal.	122
	Ejemplo 8.5. Lógica de funcionamiento de un Semáforo.	123
	Ejemplo 8.6. Solución a la ecuación de segundo grado.	125
	Ejemplo 8.7. Uso de la Unidad de Desplegado de Caracteres.	127
	Ejemplo 8.8. Velocidad promedio de un móvil.....	130
	REFERENCIAS	132

1 DESCRIPCIÓN GENERAL DE AIDA08

El sistema AIDA08 es una herramienta integrada de fácil uso, mediante la cual se pueden desarrollar aplicaciones basadas en microcontroladores de la familia 68HC908 de FREESCALE. Cuenta con hardware basado en el MCU 68HC908GP32CP, para prueba inmediata de software ejecutable en el dispositivo mencionado escrito en ensamblador y/o BASIC.

El sistema AIDA08 está integrado por dos componentes funcionales de hardware y software, los cuales son:

A) Tarjeta para desarrollo MINICON_08A ligada vía serie con una computadora de tipo PC donde corre un software manejador.

B) Software manejador denominado PUMMA_08+, el cual cuenta con un ensamblador cruzado denominado ENS08 y un compilador cruzado de BASIC denominado MINIBAS8.

De aquí en adelante a la tarjeta MINICON_08A eventualmente se le podrá denominar como Tarjeta Destino (TD).

En la figura 1.1 se muestra un esquema simplificado de los componentes funcionales del sistema AIDA08.



Figura 1.1 Componentes funcionales del sistema AIDA08.

Las características principales de la Tarjeta Destino (TD) son:

1. Está basada en el MCU 68HC908GP32CP fabricado por FREESCALE.
2. Contiene interfaz de seis hilos compatible con la tarjeta de programación IP_ASC_08B, lo cual permite probar aplicaciones partiendo de microcontroladores sin firmware en memoria FLASH, empleando para ello al manejador AMIGO_08.
3. Capacidad para operar con microcontroladores de la familia HC908 que contengan el firmware NBCP8.
3. Capacidad para operar con microcontroladores de tipo CHIPBAS8 GP32, que contienen el firmware NBCP8_BIBAS8 compatible con el compilador cruzado MINIBAS8A.
4. Contiene postes (headers) que acceden diversos puntos de interés, como pueden ser líneas de entrada y/o salidas de puertos y diversos puntos de prueba.
5. Incluye auxiliares didácticos tales como LEDS testigo para el puerto A.
6. Contiene interfaz para desplegados alfanuméricos populares en la industria.
7. Capacidad para ejecución autónoma de programas previamente cargados en memoria FLASH con facilidades propias del manejador PUMMA_08+.

Nota: NBCP8 son siglas que denotan lo siguiente: Núcleo Básico de Comunicaciones con PUMMA_08+. NBCP8_BIBAS8 denota firmware que contiene las bibliotecas de MINIBAS8A y al NBCP8.

En la figura 1.2 se muestra una fotografía de la tarjeta destino.



Figura 1.2. Tarjeta destino MINICON_08A

Las características principales del software manejador PUMMA_08+ son:

1. Ejecutable bajo WINDOWS (98/M/XP/V).
2. Capacidad para cargar en la TD archivos S19, que pudieran contener datos o programas en lenguaje de máquina.
3. Capacidad para ejecutar un programa previamente cargado en la TD.
4. Capacidad para escribir datos a memoria o puerto en la TD.
5. Capacidad para examinar memoria o puertos en la TD.
6. Capacidad para programar la memoria FLASH contenida en el MCU presente en la TD.
7. Contiene un editor básico para que el usuario pueda escribir y almacenar el o los programas asociados con una determinada aplicación.
8. Contiene un ensamblador cruzado, denominado **ENS08**.
9. Contiene un compilador cruzado de lenguaje **BASIC**, denominado **MINIBAS8**.
10. Capacidad para obtener, a partir del código fuente en ensamblador, presente en la ventana del editor, el archivo S19 que contiene el código de máquina ejecutable en el MCU de la TD.
11. Capacidad para obtener, a partir del código fuente en BASIC, presente en la ventana del editor, el archivo S19 que contiene el código de máquina ejecutable en el MCU de la TD.
12. Capacidad para ejecutar de inmediato el código generado a partir de un determinado programa fuente escrito en lenguaje ensamblador y presente en la ventana del editor.
13. Capacidad para ejecutar de inmediato el código generado a partir de un determinado programa fuente escrito en lenguaje BASIC y presente en la ventana del editor.
14. Contiene un emulador de terminal básico mediante el cual se realiza la consola de interfaz con el usuario para fines de la ejecución de programas en BASIC.

2 TARJETA DESTINO MINICON_08A

2.1 Modos de funcionamiento de la tarjeta destino.

La tarjeta destino puede utilizarse de tres formas diferentes, esto depende del tipo de firmware residente en la memoria FLASH del microcontrolador presente en ésta. Si se parte de un chip 68HC908GP32CP nuevo tal como éste es surtido por FREESCALE, la tarjeta operará bajo la modalidad denominada MODO A. En caso de que en la memoria FLASH se encuentre el firmware NBCP8 la tarjeta operará bajo la modalidad denominada MODO B. Por otro lado si el firmware residente en la memoria FLASH es el denominado como NBCP8_BIBAS8 la modalidad de funcionamiento se denomina MODO C. A continuación se describen aspectos básicos del funcionamiento de la tarjeta bajo cada una de las modalidades de funcionamiento aquí mencionadas.

2.2 Aspectos básicos de la modalidad A de funcionamiento

Bajo este modo, la memoria FLASH disponible para el usuario sería toda la disponible en el chip y para probar y depurar aplicaciones se podrá emplear, entre otras herramientas, a la tarjeta de programación IP_ASC_08B conectada mediante seis hilos al conector siete presente en la tarjeta. Para fines de interfaz con el chip se emplearía el software manejador AMIGO_08, detalles sobre la tarjeta de programación y el software manejador aquí mencionado pueden verse en [2] y [3]. Cabe señalar que para la operación correcta de AMIGO_08 para fines de los procesos de interfaz con el chip, deben estar colocados los *jumpers* 17, 18, el inferior de JE6 y el izquierdo de JE1, además de que el pin PTA7 debe conectarse a tierra. En esta modalidad cuando se maneja el chip empleando el software AMIGO_08 el microcontrolador operaría bajo lo que el fabricante denomina modo monitor, detalles acerca de esto pueden verse en [1].

2.3 Aspectos básicos de la modalidad B de funcionamiento

Bajo este modo, no se requiere de hardware adicional para el interfazado con el microcontrolador, pero habrá una zona de la memoria FLASH reservada y protegida donde residirá el NBCP8 que viene siendo un valor agregado al chip, lo que permite que éste pueda ser manejado empleando al manejador PUMMA_08+. En la tabla 2.1 se muestra el mapa de memoria del MCU de la tarjeta destino bajo la modalidad B.

El NBCP8 ocupa un total de 548 localidades de la memoria FLASH del MCU, esto en dos intervalos disjuntos los cuales son:

Intervalo 1 del NBCP8, el cual está comprendido de la dirección FC00 a la dirección FDFF, aquí es donde está el receptor de comandos enviados por el usuario por medio del software manejador PUMMA_08+. Estos podrían ser entre otros: el requerimiento de lectura de una zona de memoria, el borrado de la memoria FLASH disponible para el desarrollo de las aplicaciones, o bien el grabado en memoria FLASH del código asociado con una determinada aplicación, previamente generado con las facilidades de ensamble y/o compilación propias del software PUMMA_08+.

Intervalo 2 del NBCP8, el cual está comprendido de la dirección FFDC a la dirección FFFF, aquí es donde se colocan los vectores de RESET y de interrupción propios del NBCP8.

TABLA 2.1. MAPA DE MEMORIA DEL MCU 68HC908GP32CP COMO COMPONENTE DE LA TARJETA MINICON_08A OPERANDO EN EL MODO B.

INTERVALO DE DIRECCIONES EN EXPRESADAS EN HEXADECIMAL	CONTENIDO
0000 a 003F	Zona 1 de registros de control y operación (RCO) (I/O registers)
0040 a 023F	Memoria RAM (512 bytes)
0240 a 7FFF	32192 localidades no implementadas
8000 a FBDB	31708 localidades de Memoria FLASH de usuario
FBDC a FBFF	Vectores de RESET e Interrupción del usuario. (Memoria tipo FLASH)
FC00 a FDFF	Intervalo 1 del NBCP8. (Memoria tipo FLASH)
FE00 a FE0C	Zona 2 de registros de control y operación
FE0D a FE1F	19 localidades no implementadas
FE20 a FF52	307 localidades del ROM monitor
FF53 a FF7D	43 localidades no implementadas
FF7E	Registro tipo FLASH delimitador de dirección inicial de protección contra borrado de la memoria FLASH (FLBPR)
FF7F a FFDB	93 localidades no implementadas
FFDC a FFFF	Intervalo 2 del NBCP8, vectores del NBCP8. (Memoria tipo FLASH)

2.4 Aspectos básicos de la modalidad C de funcionamiento

Bajo este modo, no se requiere de hardware adicional para el interfazado con el microcontrolador, pero habrá una zona de la memoria FLASH reservada y protegida donde residirá el firmware NBCP8_BIBAS8 que comprende al NBCP8 y a las bibliotecas completas del compilador MINIBAS8A, lo cual constituye un valor agregado al chip, lo que permite que éste pueda ser manejado empleando al manejador PUMMA_08+, además de poder ejecutar programas compilados a partir de código fuente en lenguaje BASIC. A los microcontroladores 68HC908GP32 que contengan como software de base el firmware NBCP8_BIBAS8 se les denomina **Chipbas8 GP32**. En la tabla 2.2 se muestra el mapa de memoria del MCU de la tarjeta destino bajo la modalidad C.

TABLA 2.2. MAPA DE MEMORIA DEL MCU 68HC908GP32CP COMO COMPONENTE DE LA TARJETA MINICON_08A OPERANDO EN EL MODO C.

INTERVALO DE DIRECCIONES EN EXPRESADAS EN HEXADECIMAL	CONTENIDO
0000 a 003F	Zona 1 de registros de control y operación (RCO) (I/O registers)
0040 a 023F	Memoria RAM (512 bytes)
0240 a 7FFF	32192 localidades no implementadas
8000 a D7DB	22492 localidades de Memoria FLASH de usuario
D7DC a D7FF	Vectores de RESET e Interrupción del usuario. (Memoria tipo FLASH)
D800 a FBFF	Bibliotecas de MINIBAS8A
FC00 a FDFE	Intervalo 1 del NBCP8. (Memoria tipo FLASH)
FE00 a FE0C	Zona 2 de registros de control y operación
FE0D a FE1F	19 localidades no implementadas
FE20 a FF52	307 localidades del ROM monitor
FF53 a FF7D	43 localidades no implementadas
FF7E	Registro tipo FLASH delimitador de dirección inicial de protección contra borrado de la memoria FLASH (FLBPR)
FF7F a FFDB	93 localidades no implementadas
FFDC a FFFF	Intervalo 2 del NBCP8, vectores del NBCP8. (Memoria tipo FLASH)

2.5 Acciones al RESET de la tarjeta MINICON_08A operando bajo los modos B y C

Dependiendo del status lógico del bit PTC3 del puerto C del MCU y del contenido de las localidades de almacenaje de una palabra de dos bytes que se denomina *vector de RESET usuario*, al darse un RESET podrán efectuarse una de los siguientes dos acciones:

- Ejecución del receptor de comandos del NBCP8, lo cual es testificado por el parpadeo (blinking) del LED 1 (amarillo) presente en la tarjeta, esto hace que el usuario pueda efectuar diversos comandos desde el manejador PUMMA_08+.
- Ejecución autónoma de un programa precargado en memoria FLASH a partir de la dirección denotada por el contenido de las localidades XXFE y XXFF. A tal dirección se le denomina como *vector de RESET de usuario*. XX es 'FB' para la modalidad B y 'D7' para la modalidad C.

El nivel lógico del bit PTC3 es gobernado por el *jumper* excluyente JE1 presente en la parte central derecha de la tarjeta, véase la figura 2.1. Cuando se coloca un puente del lado con la leyenda PUMMA_08 el nivel lógico de PTC3 será alto, cuando el puente es colocado del lado con la leyenda EA el nivel lógico de PTC3 será bajo. En la tabla 2.3 se detallan las posibles acciones al RESET aquí mencionadas.

TABLA 2.3 ACCIONES AL RESET DE LA TARJETA MINICON_08A BAJO LOS MODOS B Y C

STATUS DE JE1	VECTOR DE RESET DE USUARIO (VRS)	ACCIÓN AL RESET
Jumper colocado del lado EA (PTC3=0)	VRS diferente de FFFF	Ejecución autónoma de programa originado en la dirección VRS
Jumper colocado del lado EA (PTC3=0)	VRS = FFFF	Ejecuta receptor de comandos del NBCP8
Jumper colocado del lado PUMMA_08 (PTC3=1)	Cualquier valor	Ejecuta receptor de comandos del NBCP8

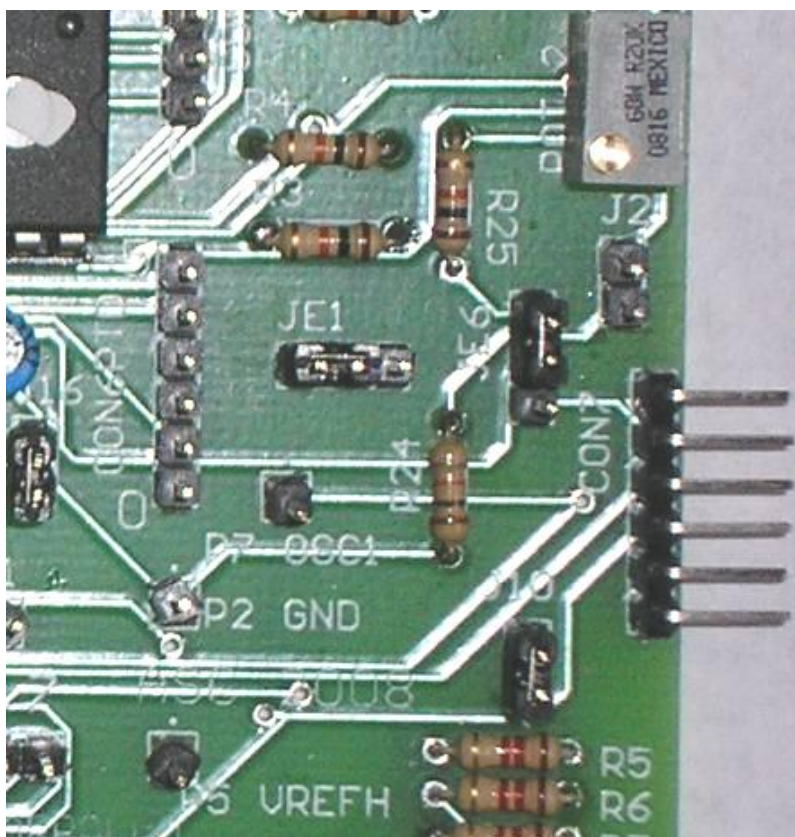


Figura 2.1. Ubicación del jumper excluyente JE1 en la tarjeta MINICON_08A

2.6 Vectores de usuario y zona de FLASH protegida contra borrado

Para la operación correcta de la tarjeta destino, bajo las modalidades B o C dependiendo del caso debe estar siempre presente ya sea el firmware NBCP8 o bien el denominado como NBCP8_BIBAS8. Esto hace que para la modalidad B la memoria FLASH deberá estar protegida contra escritura o borrado a partir de la dirección FC00 y para la modalidad C esta zona de protección deberá iniciar a partir de la dirección D800. Por lo tanto, los vectores de RESET e interrupción propios del MCU no pueden ser modificados por el usuario; de hecho, estos apuntan

a direcciones dentro del NBCP8 donde existe *código de enrutamiento* para cada instancia de interrupción. El código de enrutamiento lee la dirección de inicio de las rutinas de servicio programadas por el usuario en la zona de vectores de usuario (FBDC a FBFF para la modalidad B y D7DC a D7FF para la modalidad C), correspondiendo ahí un par de localidades para cada instancia de interrupción, donde se debe cargar en cada caso la dirección de inicio de la rutina de servicio que corresponda. A las direcciones de origen de las rutinas manejadoras de interrupción se les denomina *vectores de usuario*. Las direcciones de las localidades donde éstas deben ser colocadas pueden obtenerse a partir de la dirección de almacenamiento del vector propio del MCU que corresponda, cambiando en ésta los dos dígitos hexadecimales más significativos por el par ‘FB’ para la modalidad B, para la modalidad C el par a colocar será ‘D7’. Por ejemplo, si el usuario usa la instancia de interrupción cuyo evento asociado es el sobreflujo del contador del temporizador uno del MCU, las direcciones donde se debe colocar el vector de usuario son FBF2 y FBF3 para la modalidad B; para la modalidad C estas direcciones serán D7F2 y D7F3, ya que las direcciones propias de colocación del vector asociado a este evento para el MCU son FFF2 y FFF3. Las direcciones propias de colocación de vectores del MCU pueden verse en la tabla presente en la página 79 de [1].

2.7 Ubicación y función de los jumpers sencillos de la tarjeta MINICON_08A

En la tabla 2.4 se muestra la ubicación y funcionalidad de los *jumpers* sencillos de la tarjeta MINICON_08A.

TABLA 2.4. UBICACIÓN Y FUNCIONALIDAD DE LOS *JUMPERS* SENCILLOS DE LA TARJETA MINICON_08A

Jumper	FUNCIONALIDAD	CUADRANTE DE UBICACIÓN	Status por defecto
J1	Polarización de llave externa de MINIBAS8	Superior izquierdo	No colocado
J2	Conexión de filtro de PLL	Superior derecho	No colocado
J3	Puente sobre resistencia R16	Inferior izquierdo	Colocado
J4	Postes en paralelo con botón de RESET	Inferior izquierdo	No colocado
J5	Conexión de +VDD a pin VDDAD	Inferior derecho	Colocado
J6	Conexión de extremo variable de POT2 a pin PTB7/AN7	Inferior derecho	No colocado
J7	Conexión de +VCC a pin VDD	Superior derecho	Colocado
J8	Conexión de +VCC a pin VDDA	Superior derecho	Colocado
J9	Iluminación de LCD	Superior derecho	No colocado
J10	Retorno a tierra de LEDS conectados al puerto A	Inferior derecho	Colocado
J11	Conexión del LED testigo del receptor de comandos del NBCP8	Inferior Izquierdo	Colocado
J12	Conexión de resistencia de PUP/PDWN a pin PTB3	Inferior Izquierdo	No colocado
J13	Conexión de resistencia de PUP/PDWN a pin PTB2	Inferior Izquierdo	No colocado
J14	Conexión de resistencia de PUP/PDWN a pin PTB1	Inferior Izquierdo	No colocado
J15	Conexión de resistencia de PUP/PDWN a pin PTB0	Inferior Izquierdo	No colocado
J16	Conexión a tierra de pin VSSAD/VREFL	Superior derecho	Colocado
J17	Conexión de pin PTC0 a +VCC	Superior derecho	No colocado
J18	Conexión a tierra de pin PTC1	Superior derecho	No colocado
J20	Habilitador de llave residente de MINIBAS8	Superior derecho	Colocado
J21	Conexión directa de TxD a pin T1IN del CI 2	Superior derecho	No colocado

2.8 Ubicación y función de los jumpers excluyentes de la tarjeta MINICON_08A

Además de los *jumpers* sencillos, la TD cuenta con seis *jumpers* excluyentes que seleccionan una de dos opciones mutuamente exclusivas como podría ser entre otras, el validar que un programa se ejecute de manera autónoma o bien que el mismo sea cargado y ejecutado bajo comandos del manejador PUMMA_08+, lo cual es configurado mediante el *jumper* excluyente denominado JE1. En la tabla 2.5 se muestra la funcionalidad y ubicación de los jumpers excluyentes aquí mencionados.

TABLA 2.5. UBICACIÓN Y FUNCIONALIDAD DE LOS *JUMPERS* EXCLUYENTES DE LA TARJETA MINICON_08A

JUMPER EXCLUYENTE (Ubicación **)	Opción seleccionada con el puente Izquierdo/Inferior colocado	Opción seleccionada con el puente Derecho/Superior colocado.
JE1 (SD)	*Al RESET se ejecuta el receptor de comandos del NBCP8 lo que hace que la TD pueda ser manejada por PUMMA_08+	*Se ejecuta de manera autónoma el programa originado por la dirección dada por el vector de usuario
JE2 (SI)	Las resistencias conectadas a PTBi i=0,1,2,3 son de <i>pull up</i>	Las resistencias conectadas a PTBi i=0,1,2,3 son de <i>pull down</i>
JE3 (SI)	+VCC (5V) se obtiene del regulador (CI 3).	+VCC (5V) se obtiene de fuente externa conectada a CON4
JE4 (II)	El cristal está desconectado del pin OSC1 del MCU	El cristal está conectado al pin OSC1 del MCU
JE5 (II)	El cristal está conectado al pin OSC2 del MCU	El cristal está desconectado del pin OSC2 del MCU
JE6 (SD)	El pin IRQ del MCU está conectado a la Terminal 2 (vtst) del conector 7	El pin IRQ del MCU está conectado a una resistencia de <i>pull up</i>

* Véase además la tabla 2.3.

** Véase la nota 1 al final de la tabla 2.7.

2.9 Ubicación y funcionalidad de los conectores de la tarjeta MINICON_08A

En la tabla 2.6 se muestra la ubicación y funcionalidad de los conectores (headers) presentes en la tarjeta MINICON_08A.

TABLA 2.6. UBICACIÓN Y FUNCIONALIDAD DE LOS CONECTORES PRESENTES EN LA TARJETA MINICON_08A

CONECTOR	FUNCIONALIDAD	UBICACIÓN **
CON1	Alimentación desde eliminador de batería	II
CON2	Terminal DB9 hembra para cable RS232	SI
CON3	Interfaz para desplegado (LCD)	SD
CON4	Fuente externa de 5 V	SI
CON5	Interfaz de teclado	II e ID
CON6PTA	Acceso a pines del puerto A	ID
CON6PTB	Acceso a pines del puerto B	II e ID
CON6PTC	Acceso a pines del puerto C	SD
CON6PTD	Acceso a pines del puerto D	SD
CON7	Interfaz para programar chips nuevos empleando la tarjeta de programación IP_ASC_08B	ID y SD

** Véase la nota 1 al final de la tabla 2.7.

2.10 Ubicación y funcionalidad de los postes de prueba de la tarjeta MINICON_08A

En la tabla 2.7 se muestra la ubicación y funcionalidad de los postes de prueba presentes en la tarjeta MINICON_08A.

TABLA 2.7. UBICACIÓN Y FUNCIONALIDAD DE LOS POSTES DE PRUEBA PRESENTES EN LA TARJETA MINICON_08A

POSTE	FUNCIONALIDAD	UBICACIÓN (Véase la nota 1)
1	Testigo de pin PTB7/AN7	ID
2	Tierra	ID
3	+VCC (+5 V)	SI
4	Testigo de pin OSC2	II
5	Testigo de pin VREFH	ID
6	Testigo de pin VREFL	ID
7	Testigo de pin OSC1	ID
8	Testigo de pin IRQ	ID

Nota 1:

II denota: cuadrante inferior izquierdo.

ID denota: cuadrante inferior derecho.

SI denota: cuadrante superior izquierdo.

SD denota: cuadrante superior derecho.

2.11 Eliminador de batería recomendado

La tarjeta MINICON_08A puede polarizarse ya sea conectando directamente una fuente de cinco volts regulada al conector CON4 o bien mediante un eliminador de batería conectado al conector CON1, véase en la tabla 2.5 lo referente al jumper excluyente JE3. Si se usa el eliminador de batería el mismo deberá proporcionar preferentemente un voltaje comprendido en el rango de 7 a 9 volts con el centro del plug con polaridad positiva. Un dispositivo de este tipo popular en la industria, entre otros, es el denominado como CONVERTIDOR DE VOLTAJE UNIVERSAL CA/CD modelo ELI-30, fabricado y distribuido por la empresa STEREN. Éste cuenta con un selector de voltaje. Para que se logre el rango aquí mencionado se debe posicionar el selector en la posición (6), ya que por lo regular la tarjeta destino consume menos de los 300 mA que puede suplir el eliminador aquí mencionado.

2.12 Inicialización del sistema AIDA08 para uso de éste en una PC determinada.

Para inicializar AIDA08 para su operación empleando una PC determinada, ésta deberá contar con algún puerto serie disponible, si éste no es el caso como en las de tipo *Notebook*, se deberá usar un adaptador USB – SERIE, debiéndose haber instalado el *driver* de éste previo a la primera ejecución de PUMMA_08+. Una vez instalado el *driver*, empleando el administrador de dispositivos de WINDOWS, el usuario deberá tomar nota del número de puerto serie que el adaptador genera, en adelante, al usar el adaptador, preferentemente se deberá conectar éste siempre en el mismo puerto USB empleado al instalar el *driver*, ya que se ha observado en algunos sistemas, que cuando el adaptador se conecta a un puerto USB que no sea el mismo que se usó al instalar el *driver*, el número de puerto serie generado cambia, originando esto confusiones al ejecutar PUMMA_08+. **Siempre que se ejecute PUMMA_08+ el adaptador USB serie deberá estar conectado a la PC, ya que en caso contrario se originará un mal funcionamiento del programa.**

Los pasos a seguir para la inicialización son:

1. Si la PC se encuentra ejecutando alguna aplicación que use el puerto serie que el usuario piensa emplear para fines de AIDA08, ésta deberá ser terminada y cerrada.
2. Energizar la tarjeta MINICON_08A. Después de esto se deberá observar un parpadeo del LED 1, testificándose así que la TD puede ser manejada por el manejador PUMMA_08+.
3. Empleando un cable RS-232C conectar la TD al puerto serie disponible en la PC.
4. **Instalar en la PC el software manejador PUMMA_08+. Antes de instalar, se recomienda ver lo propio respecto a este proceso, en el capítulo 4 de este manual.**
5. Ejecutar PUMMA_08+ y dar el número de puerto serie disponible cuando éste lo pida. En caso de que el puerto serie dado por el usuario no esté disponible, PUMMA_08+ despliega el mensaje: **“Puerto serie ocupado o no existente”**. Si el usuario observa esto aún cuando el puerto serie definido exista, seguramente éste está capturado por otra aplicación, la cual debería haberse cerrado antes de ejecutar PUMMA_08+, (paso 1).
6. Definir 9600 bps como baudaje cuando PUMMA_08+ lo pida. Después de esto PUMMA_08+ confirmará al usuario el baudaje y el número de puerto serie a emplear en el enlace. En la figura 2.2 se muestra esta confirmación para una PC en la que se use el puerto serie COM1 para el enlace con la TD.
7. Después de la confirmación mencionada en el paso 5, PUMMA08+ pedirá al usuario definir el microcontrolador presente en la TD, si ésta es la tarjeta MINICON_08A, el usuario **deberá seleccionar la opción 2 que corresponde al microcontrolador 68HC908GP32**, véase la figura 2.3. Si el usuario selecciona alguna opción que no corresponda con el procesador presente en la TD se producirá un mal funcionamiento del manejador PUMMA_08+.

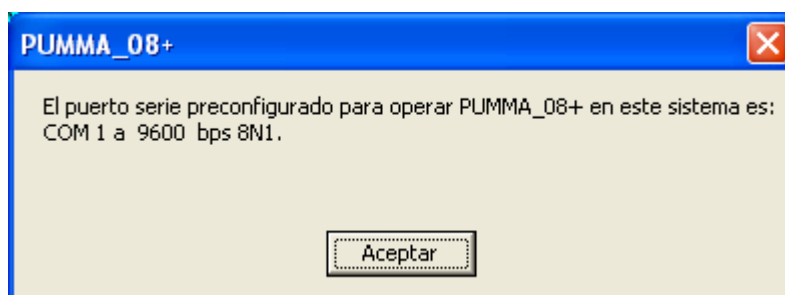


Figura 2.2. Confirmación de PUMMA_08+ acerca del puerto serie a emplear para el enlace.

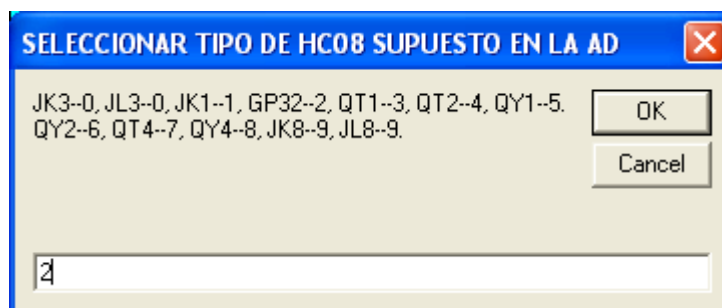


Figura 2.3. Dialogo de PUMMA_08+ para definir el tipo de HC08 presente en la TD.

Después de que el usuario ha definido el CPU presente en la TD, deberá aparecer la ventana del ambiente de edición de PUMMA_08+, el cual tendrá el aspecto mostrado en la figura 2.4.

En estas circunstancias está todo listo para trabajar con el sistema AIDA08 integrado por el binomio **PUMMA_08+ - Tarjeta MINICON_08A**. En adelante al ejecutarse PUMMA_08+ aparecerá en primera instancia el dialogo de confirmación del puerto serie a emplear que fue

predefinido por el usuario en la primera ejecución de PUMMA_08+. Si el usuario desea cambiar el puerto serie predefinido puede usar el submenú “**Configuración de puerto serie**”, presente en el menú archivo de la ventana de edición.

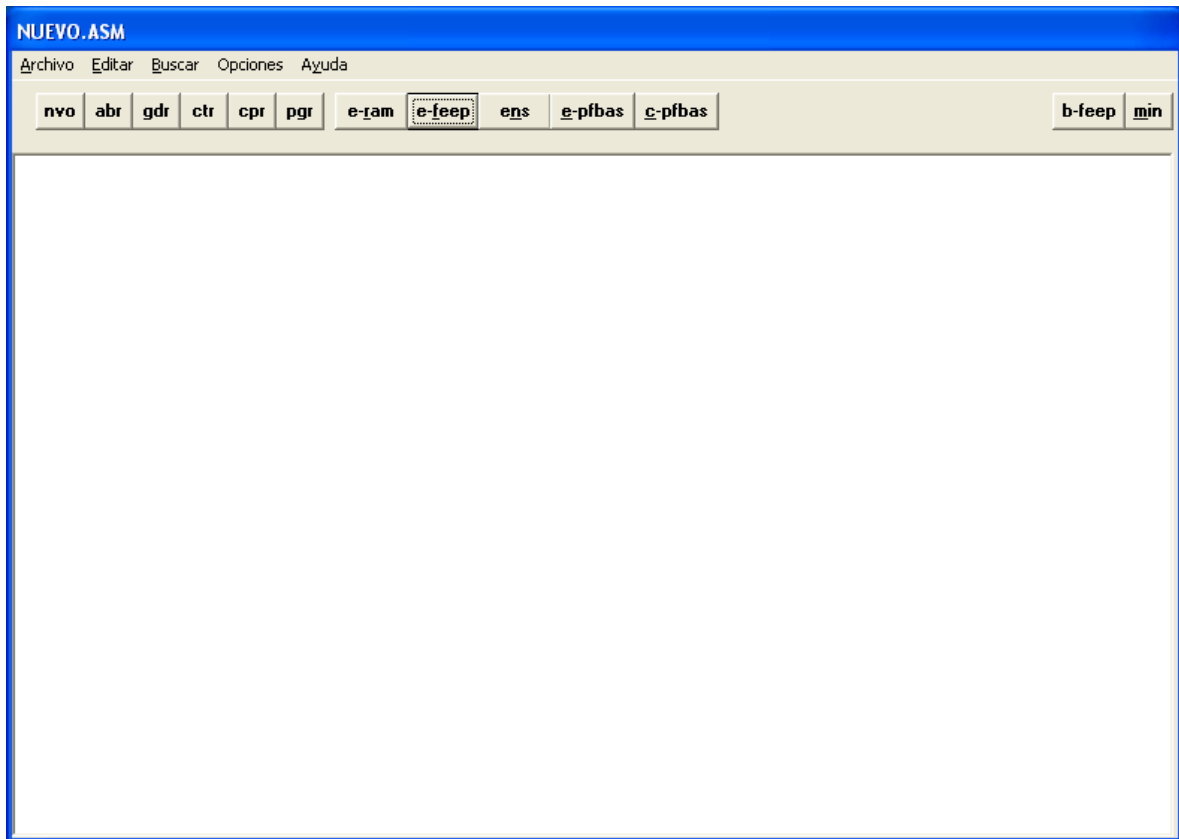


Figura 2.4. Ventana del editor de PUMMA_08+ una vez que éste se ha comunicado con la TD.

Es importante destacar aquí que para cualquier accionamiento a efectuarse sobre la TD bajo comandos de PUMMA_08+ el LED1 debe presentar un continuo proceso de encendido y apagado (*parpadeo*), lo cual testifica que el receptor de comandos presente en el NBCP8 está activo. Al ejecutarse algún programa del usuario, éste toma el control de la TD y el LED1 se apagará. Bajo esta circunstancia la TD no podrá ser manejada por PUMMA_08+; para revertir esto, bastará con oprimir el botón de RESET de la TD.

3 SOFTWARE MANEJADOR PUMMA_08+

PUMMA_08+ está integrado por varias ventanas, las principales son las siguientes:

- Ventana del ambiente de manejo hexadecimal (AMH).
- Ventana de edición

3.1 Aspecto del ambiente de manejo hexadecimal (AMH)

En esta ventana están las facilidades 2 a 6 mencionadas en la lista de características básicas de PUMMA_08+, para fines ilustrativos, en la figura 3.1 se muestra el despliegue de una página de memoria de 256 bytes tal como se ve en el AMH.

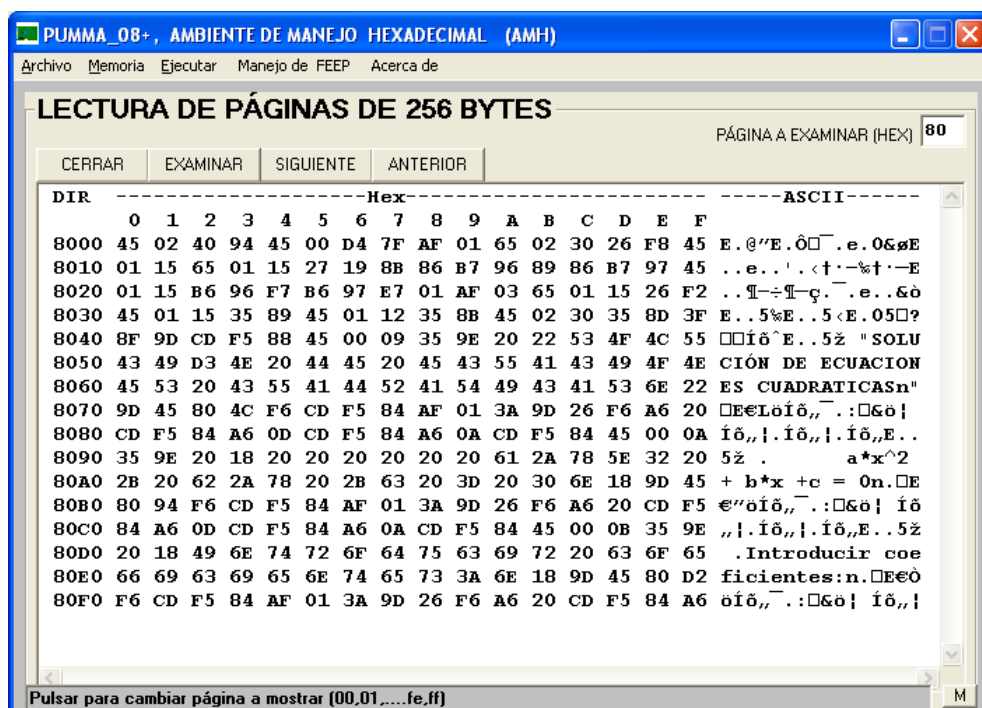


Figura 3.1. Ventana de manejo hexadecimal de PUMMA_08+, mostrando una página de 256 bytes

A continuación se describen los diversos menús y submenús presentes en el AMH de PUMMA_08+.

3.2 Menú “Archivo” del AMH.

El menú archivo del AMH cuenta con los siguientes submenús:

- Invocar editor
- Configuración de puerto serie
- Invocar emulador de Terminal

- Salir

En seguida se describen los accionamientos que se dan al invocarse cada una de las opciones anteriores.

3.2.1 Accionamiento al seleccionarse la opción “Invocar editor”

Esta opción conduce a la apertura y despliegue de la ventana del editor de PUMMA_08+.

3.2.2 Accionamiento al seleccionarse la opción “Configuración de puerto serie”

Al validarse esta opción PUMMA_08+ presenta al usuario un dialogo donde le confirma el puerto serie empleado para el enlace con la TD, éste se muestra en la figura 3.2 para el caso de que el puerto en cuestión sea el COM1.

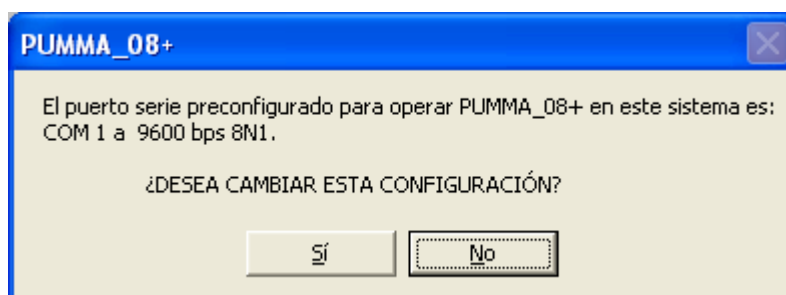


Figura 3.2. Dialogo presentado al usuario al invocarse la opción “Configuración de puerto serie”.

Si el usuario opta por cambiar la configuración del puerto serie, PUMMA_08+ le pedirá en primera instancia el número de puerto serie a emplear; en seguida le solicitará definir el baudaje, el cual deberá ser 9600 bps. En caso de que el usuario defina un número de puerto serie que esté ocupado o no exista, PUMMA_08+ presentará el mensaje mostrado en la figura 3.3.

Si el usuario opto por un puerto válido, éste será el predeterminado a partir de que se proporcione el baudaje como 9600 bps para el nuevo puerto serie a utilizar.

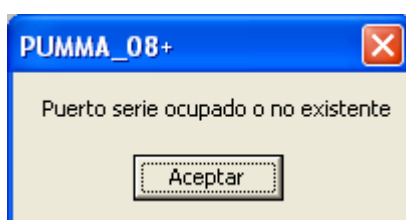


Figura 3.3. Mensaje de PUMMA_08+ indicando que el puerto serie seleccionado está ocupado o no existe.

3.2.3 Accionamiento al seleccionarse la opción “Invocar emulador de Terminal”

La selección de esta opción conduce a la ventana de un emulador de Terminal básico presente en PUMMA_08+, el baudaje por defecto es el propio de PUMMA_08+ para fines del enlace con la TD (9600 bps, formato 8N1). Mediante este emulador el usuario puede validar la consola de interfaz con la ejecución de programas en BASIC compilados por el compilador MINIBAS8 presente en PUMMA_08+.

En la figura 3.4 se muestra el aspecto que presenta la ventana del emulador de Terminal de PUMMA_08+ al invocarse el submenú aquí descrito.

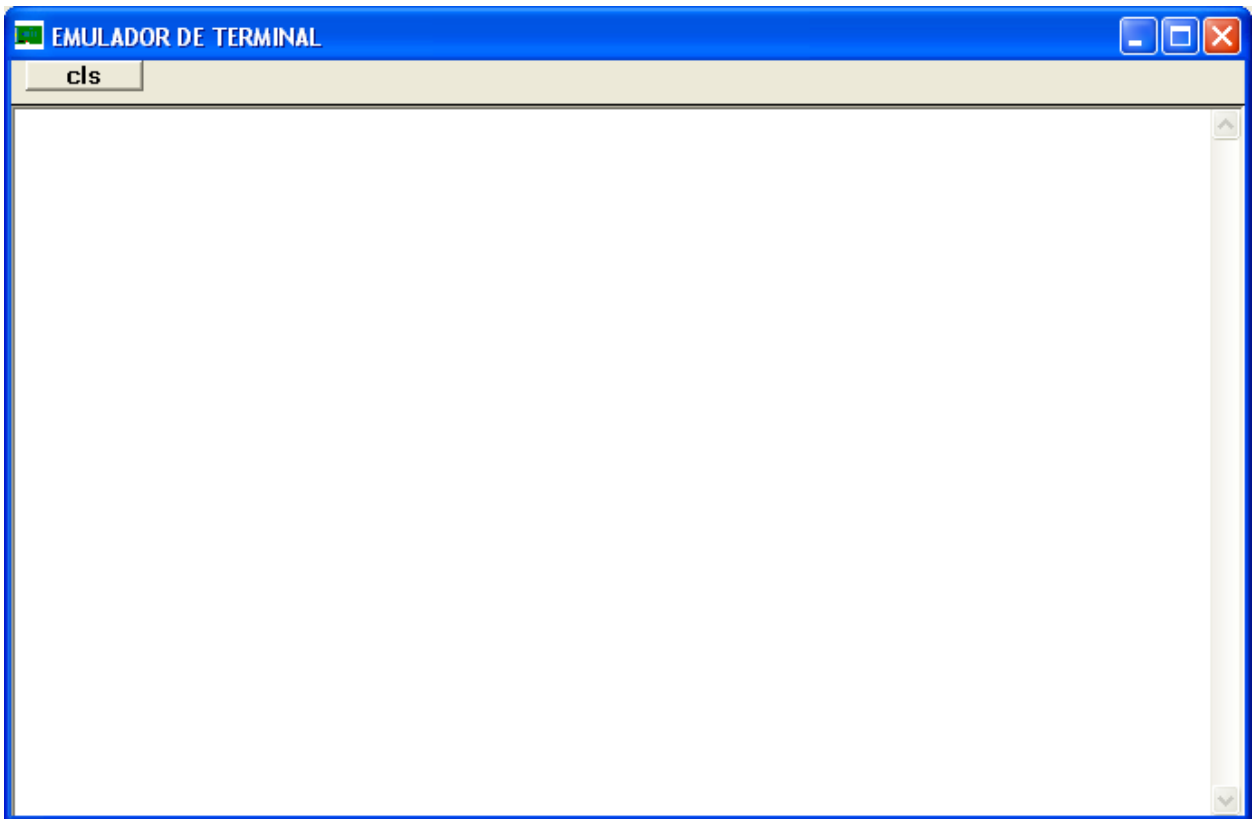


Figura 3.4. Aspecto del emulador de Terminal de PUMMA_08+ al abrirse éste.

3.2.4 Accionamiento al seleccionarse la opción “Salir”

Esta opción simplemente conduce a la terminación de la ejecución de PUMMA_08+.

3.3 Menú “Memoria” del AMH.

El menú archivo del AMH cuenta con los siguientes submenús:

- Examinar memoria: Leer una localidad
- Examinar memoria: Leer páginas de 256 bytes
- Escribir bytes en memoria de AD: Escribir un byte
- Escribir bytes en memoria de AD: Escribir varios bytes
- Cargar en memoria RAM archivo S19

En seguida se describen los accionamientos que se dan al invocarse cada una de las opciones anteriores.

3.3.1 Accionamiento al invocarse la opción “Examinar memoria: Leer una localidad”

Al seleccionar el usuario este submenú, PUMMA_08+ presenta un dialogo donde solicita la dirección de la localidad a leer, debiendo ésta ser especificada siempre con cuatro dígitos y en notación hexadecimal. En la figura 3.5 se muestra el dialogo aquí mencionado una vez que el usuario ha definido que se lea la dirección \$0000.

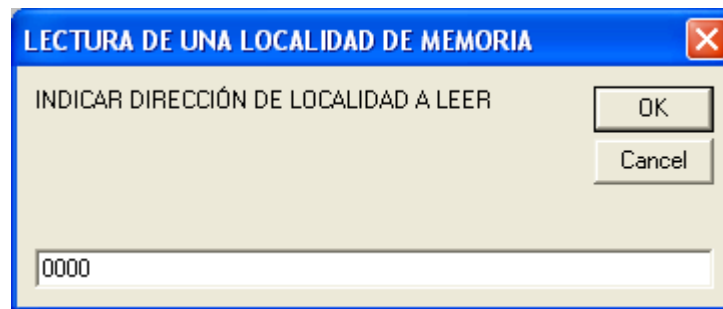


Figura 3.5. Dialogo presentado por PUMMA_08+ para especificar una dirección por leer.

Después de que el usuario oprime el botón “ok” en el dialogo de la figura 3.5, PUMMA_08+ desplegará sobre la ventana del AMH el contenido de la localidad cuya dirección se especificó. Véase la figura 3.6.

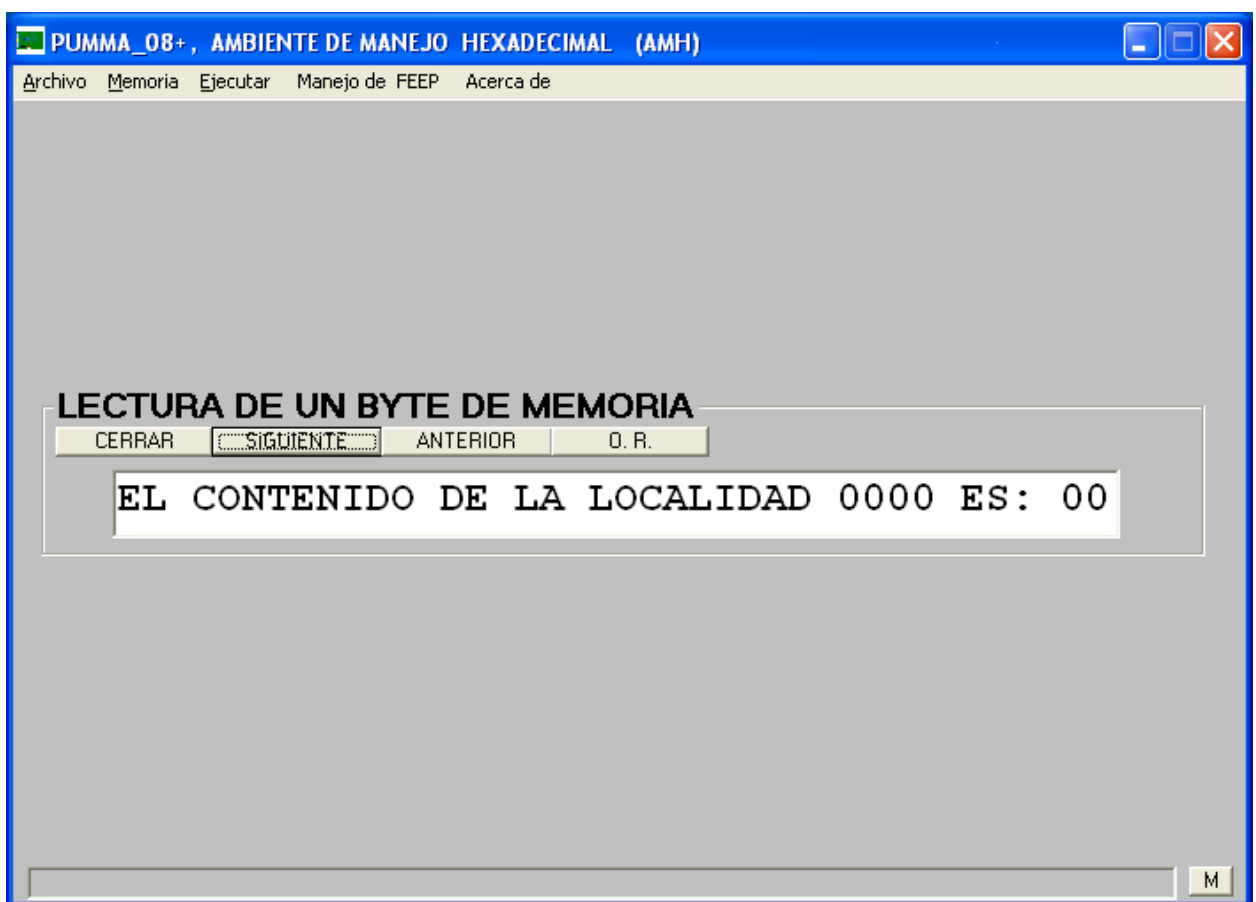


Figura 3.6. Dialogo de PUMMA_08+ al desplegar el contenido de una dirección de memoria.

Si el usuario oprime el botón “SIGUIENTE” se desplegará el contenido de la dirección adyacente hacia arriba. En caso de oprimirse el botón “ANTERIOR” se despliega el contenido de la dirección adyacente hacia abajo. Si se oprime el botón “O. R.” PUMMA_08+ hace lecturas continuas de la dirección especificada, reflejándose de inmediato en tiempo real cualquier cambio que se pudiera presentar en la localidad en cuestión.

Cabe señalar que la observación repetitiva (O. R.) puede interferir con otras facilidades de PUMMA_08+, por lo que ésta deberá cerrarse, oprimiendo el botón “CERRAR”, antes de usar otros comandos de PUMMA_08+.

3.3.2 Accionamiento al invocarse la opción “Examinar memoria: Leer páginas de 256 bytes”

Al invocarse esta opción, PUMMA_08+ desplegará el contenido de una página de 256 localidades de memoria, esto entre las direcciones XX00 y XXFF siendo XX comprendido entre 00 y FF. En la figura 3.1 se muestra el despliegue de la página comprendida entre las direcciones 8000 y 80FF (XX=80). El valor de XX puede cambiarse pulsando el botón izquierdo del *ratón* sobre la caja de texto que especifica en hexadecimal el número de página a desplegar y que está situada a la derecha de la leyenda “PÁGINA A EXAMINAR (HEX)”; después de esto, dicha caja de texto se pone en blanco, debiendo el usuario teclear sobre ella el número de página deseado empleando dos dígitos en notación hexadecimal. Si a continuación se oprime la tecla “RETURN” en el teclado de la computadora PC, se desplegará el contenido de la página definida. Una vez que la página ha sido especificada, oprimiendo el botón “EXAMINAR” se desplegará el contenido de ésta. Oprimiendo los botones “ANTERIOR” y “SIGUIENTE” se desplegarán los contenidos de páginas adyacentes hacia abajo y hacia arriba respectivamente.

3.3.3 Respuesta a la opción “Escribir bytes en memoria de AD: Escribir un byte”

Esta opción permite al usuario escribir sobre una localidad de memoria que puede ser RAM, o bien, algún registro volátil de control y operación (I/O register). Al invocarse esta facilidad, PUMMA_08+ presentará al usuario el dialogo mostrado en la figura 3.7, donde se debe especificar la dirección de la localidad por escribir la cual deberá denotarse en hexadecimal empleando cuatro dígitos. Para fines ilustrativos en la figura 3.7 se especifica la dirección 010A.

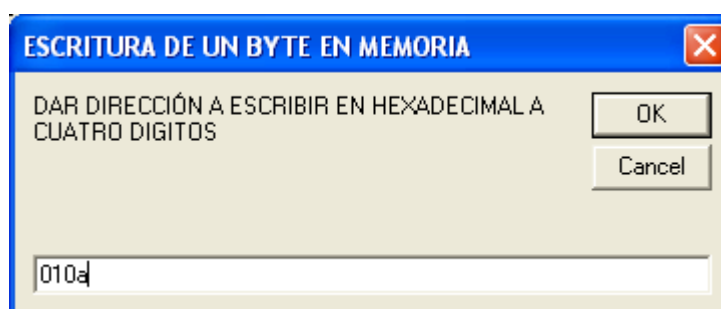


Figura 3.7. Dialogo para especificar dirección sobre cuyo contenido se va a escribir.

Después de que el usuario oprime el botón “ok” PUMMA_08+ le pedirá definir el valor del byte por escribir, lo cual deberá hacerse empleando dos dígitos en notación hexadecimal. El correspondiente dialogo se muestra en la figura 3.8, donde para fines ilustrativos se supone que el usuario definió que en la localidad cuya dirección es 010A se escriba el byte 0A.

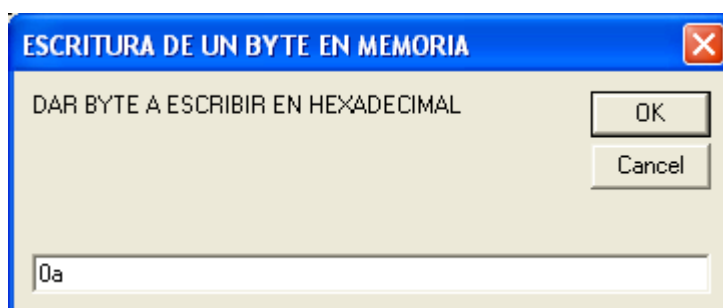


Figura 3.8. Dialogo de PUMMA_08+ para especificar byte a escribir en memoria.

Una vez que el usuario oprime el botón “ok” en el dialogo de la figura 15, PUMMA_08+ efectúa la escritura del byte especificado en la dirección previamente definida en el dialogo de la figura 3.7.

3.3.4 Respuesta a la opción “Escribir bytes en memoria de AD: Escribir varios bytes”

Al invocarse esta opción, PUMMA_08+ presenta al usuario el dialogo mostrado en la figura 3.9 donde se debe definir la dirección inicial a escribir, para fines ilustrativos en la figura 16 se muestra la dirección 018A.

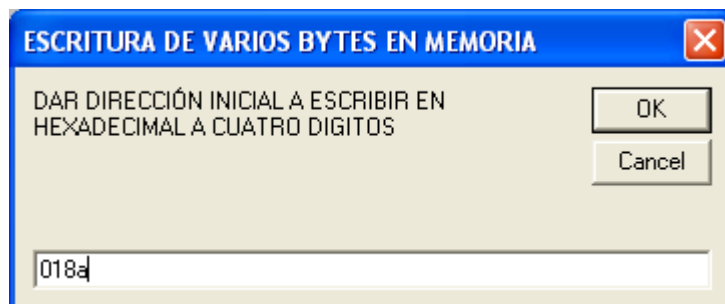


Figura 3.9. Dialogo de PUMMA08+ para definir la dirección inicial a escribir.

Una vez que el usuario ha definido la dirección inicial a escribir, PUMMA_08+ le pide definir los bytes que se van a escribir en direcciones subsecuentes, a partir de la especificada en el dialogo de la figura 3.9. Para ello PUMMA_08+ presentará al usuario sendos diálogos donde éste debe definir los bytes a cargar. Para finalizar el proceso, suponiendo que el usuario va a escribir en ‘n’ localidades, basta con oprimir el botón “CANCEL” en el dialogo que presente PUMMA_08+ para la introducción del byte ‘n+1’. En las figuras 3.10 y 3.11 se muestran los diálogos para la introducción de dos bytes a partir de la dirección \$010A, para fines ilustrativos los bytes a escribir son respectivamente \$0A y \$AB.

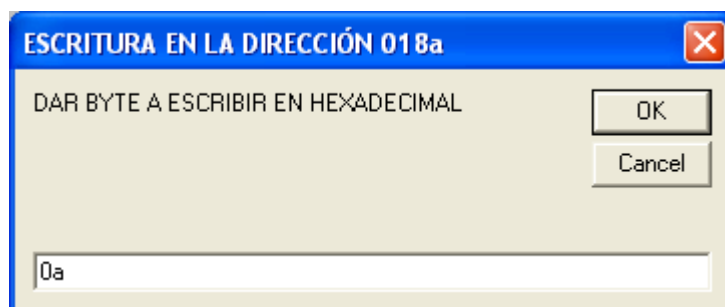


Figura 3.10. Dialogo para introducir un byte en la dirección \$018A.

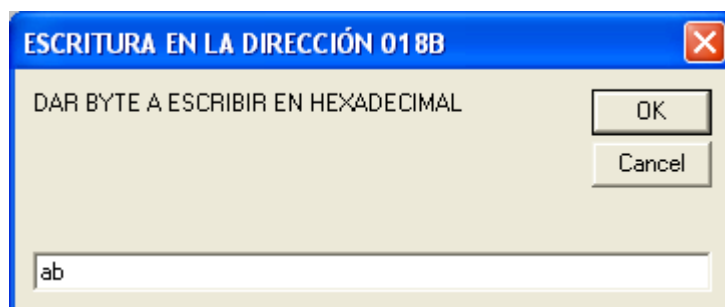


Figura 3.11. Dialogo para introducir un byte en la dirección \$018B.

Para la ejemplificación aquí mostrada, el usuario deberá oprimir el botón “CANCEL” en el dialogo donde se le pide el byte a escribir en la dirección \$018C, ya que sólo se pretende escribir dos bytes.

3.3.5 Accionamiento al invocarse la opción: “Cargar en RAM archivo S19”

Al invocarse esta opción, PUMMA_08+ presenta al usuario el dialogo mostrado en la figura 3.12, para que se seleccione el archivo S19 a cargarse en la memoria RAM. Una vez que el nombre del archivo ha sido definido y el usuario ha oprimido el botón “ABRIR”, PUMMA_08+ procede a leer el archivo y volcar su contenido en la memoria RAM de la TD.

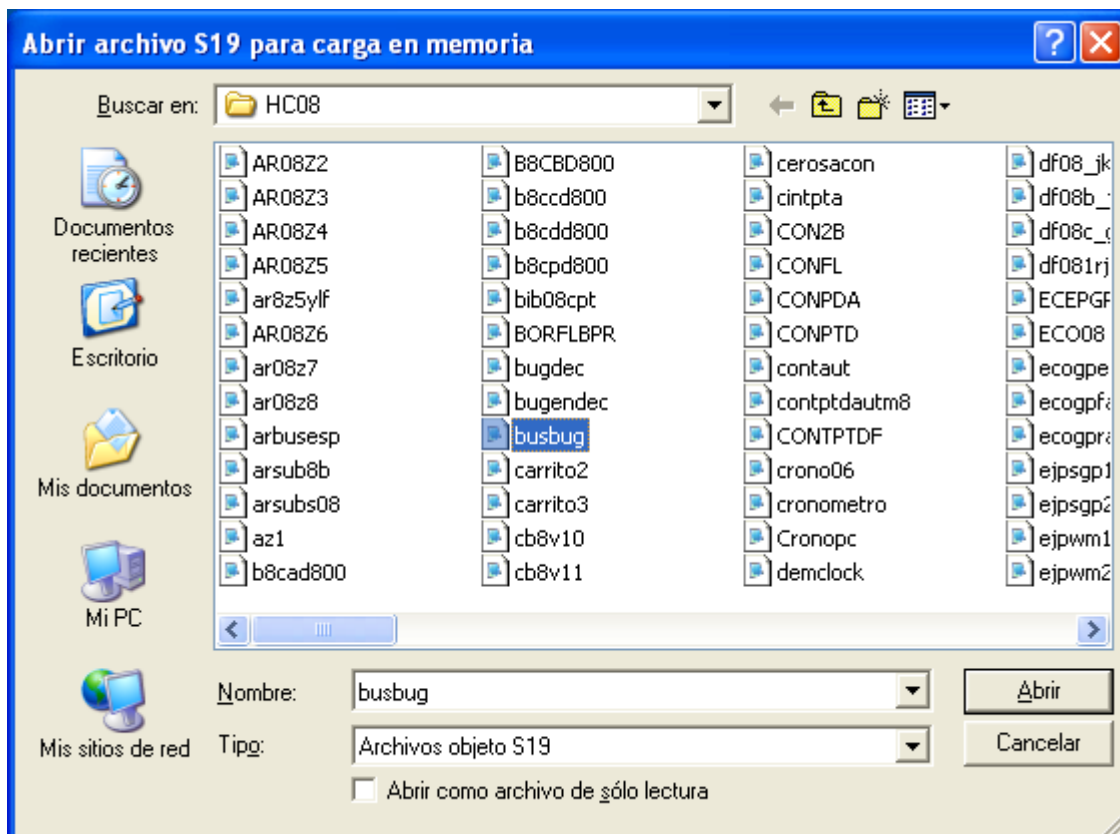


Figura 3.12. Dialogo para abrir y cargar en RAM de la TD un archivo S19.

Es responsabilidad del usuario el que el archivo S19 implicado, en efecto contenga direcciones que correspondan a memoria RAM presente en la TD; en caso contrario, la información del archivo en cuestión no será cargada correctamente.

3.4 Menú “Ejecutar” del AMH.

El menú Ejecutar del AMH cuenta con los siguientes submenús:

- Ejecutar a velocidad plena a partir de dirección dada
- Ejecutar a velocidad plena a partir de archivo S19 en RAM
- Ejecutar a velocidad plena a partir de archivo S19 en FEEPROM

En seguida se describen los accionamientos que se dan al invocarse cada una de las opciones anteriores.

3.4.1 Accionamiento de la opción “Ejecutar a velocidad plena a partir de dirección dada”

Al seleccionarse este submenú, PUMMA_08+ presenta al usuario el dialogo mostrado en la figura 3.13, donde se debe especificar la dirección inicial de un programa que se desea ejecutar y que previamente se cargó en la memoria de la tarjeta destino. La dirección implicada debe especificarse empleando cuatro dígitos en notación hexadecimal; en la figura se muestra la dirección \$8000 para fines ilustrativos.

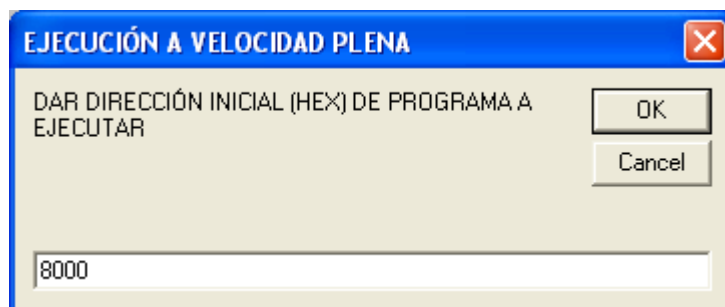


Figura 3.13. Dialogo para especificar dirección inicial de programa a ejecutar.

Cuando el usuario oprime el botón “OK” se inicia la ejecución de código de máquina en la TD a partir de la dirección definida en el dialogo de la figura 3.13. Es responsabilidad del usuario que en efecto exista código coherente a partir de la localidad de memoria especificada, de no ser éste el caso, se originará un mal funcionamiento en la TD que se podrá revertir oprimiendo el botón de RESET en ésta.

3.4.2 Acción de la opción “Ejecutar a velocidad plena a partir de archivo S19 en RAM”

Al seleccionarse esta opción, PUMMA_08+ presenta al usuario el dialogo mostrado en la figura 3.14, donde se debe especificar el nombre del archivo S19 que se desea ejecutar.

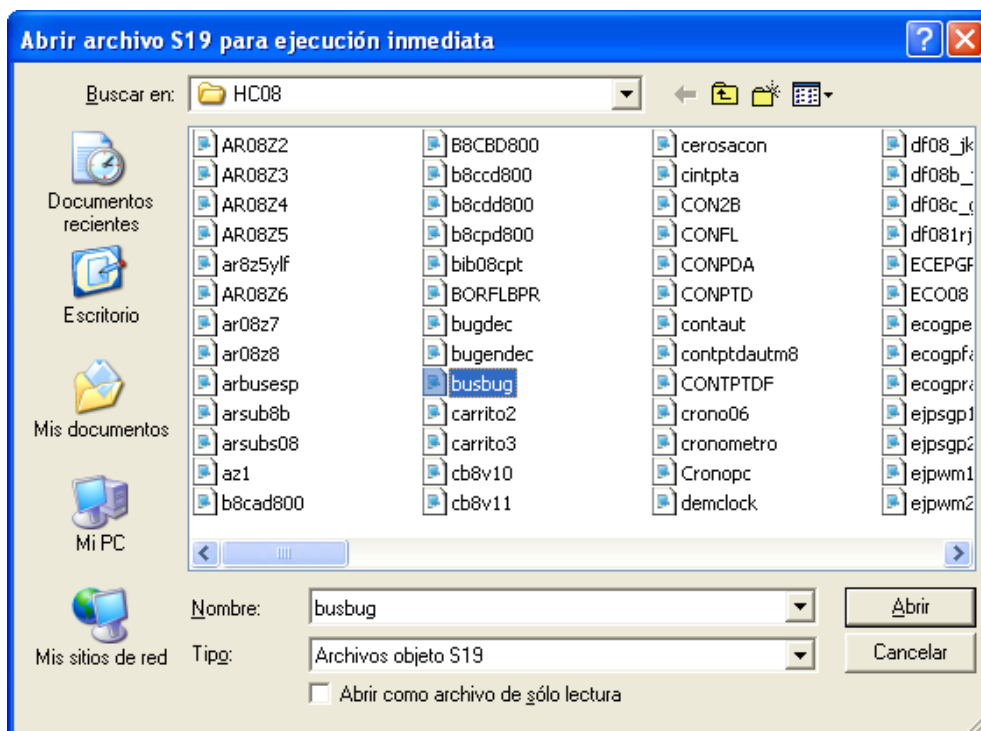


Figura 3.14. Dialogo para especificar nombre de archivo S19 a ejecutar en memoria RAM.

Una vez que el usuario oprime el botón “ABRIR”, PUMMA_08+ carga en memoria RAM el archivo S19 especificado y lo ejecuta de inmediato. Es responsabilidad del usuario que, para las direcciones presentes en el archivo S19, en efecto exista memoria RAM física en la TD, de no ser éste el caso, se originará un mal funcionamiento en la TD.

Es importante señalar que, para el proceso de carga y ejecución aquí descrito, PUMMA_08+ supone que la dirección inicial del programa a ejecutar es la contenida en el primer renglón del archivo S19 implicado; si éste no es el caso, se genera un mal funcionamiento en la TD.

3.4.3 Acción de la opción “Ejecutar a velocidad plena a partir de archivo S19 en FEEP”

Al seleccionarse esta opción, PUMMA_08+ presenta al usuario el dialogo mostrado en la figura 3.15; ahí se debe especificar el nombre del archivo S19 a ejecutar en la memoria FLASH de la TD.

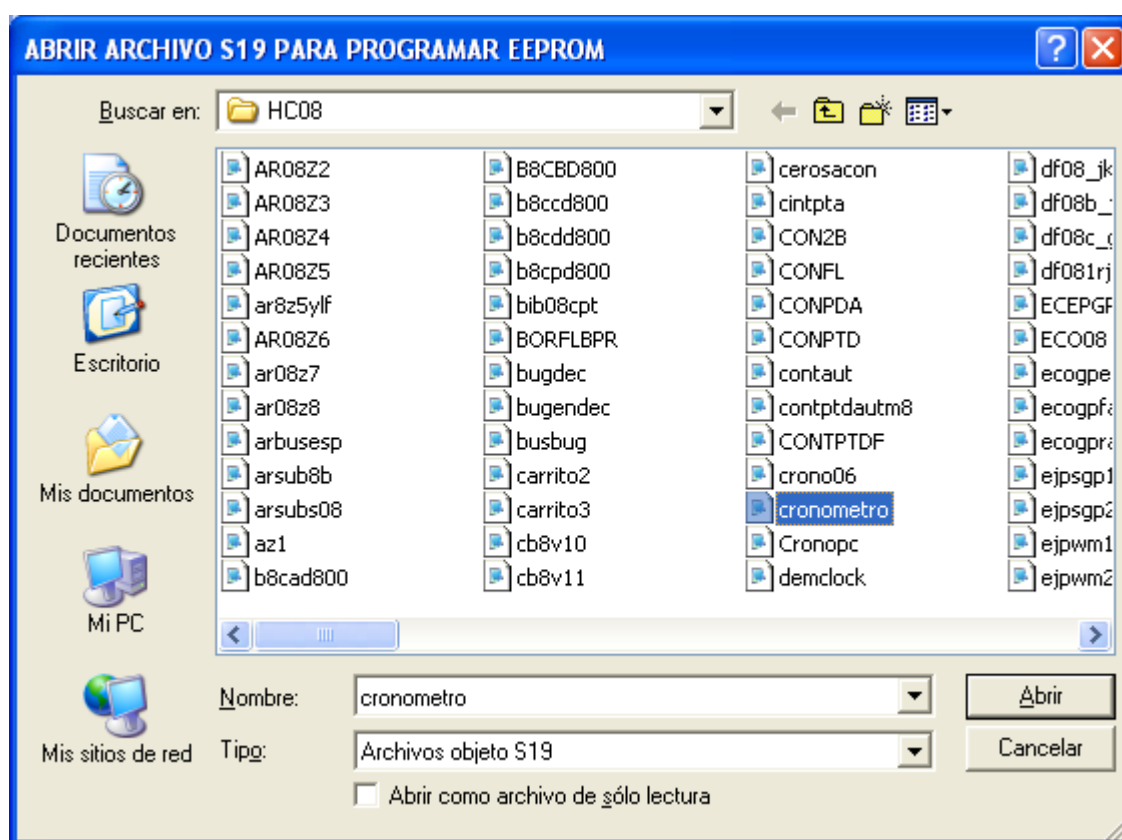


Figura 3.15. Dialogo para especificar archivo S19 a ejecutarse en memoria FLASH.

Después de que el usuario oprime el botón “ABRIR” en el dialogo de la figura 3.15, PUMMA_08+ graba el contenido del archivo en memoria FLASH de la TD para de inmediato proceder con la ejecución de éste. Es responsabilidad del usuario que, para las direcciones presentes en el archivo S19, en efecto exista memoria FLASH física en la TD, de no ser éste el caso, se originará un mal funcionamiento en la TD.

Es importante señalar que, para el proceso de carga y ejecución aquí descrito, PUMMA_08+ supone que la dirección inicial del programa a ejecutar es la contenida en el primer renglón del archivo S19 implicado; si éste no es el caso, se genera un mal funcionamiento en la TD.

3.5 Menú “Manejo de FEEP” del AMH.

El menú Manejo de FEEP del AMH cuenta con los siguientes submenús:

- Verificar que FEEP esté borrada
- Programar FEEP desde teclado como texto
- Programar FEEP desde teclado como bytes (hex)
- Programar FEEP desde archivo S19
- Verificar FEEP contra archivo S19
- Borrar FEEP

En seguida se describen los accionamientos que se dan al invocarse cada una de las opciones anteriores.

3.5.1 Accionamiento propio de la opción “Verificar que FEEP esté borrada”

Al seleccionarse este submenú, PUMMA_08+ presenta al usuario de manera sucesiva los diálogos mostrados en las figuras 3.16 y 3.17. En el de la figura 3.16 el usuario debe especificar la dirección origen de memoria a verificar. Aquí PUMMA_08+ especifica por defecto la dirección inicial de la memoria FLASH de usuario en la TD. En la figura 3.17 el usuario debe especificar la dirección final de memoria a verificar. Aquí PUMMA_08+ especifica por defecto la dirección final de la memoria FLASH de usuario en la TD.

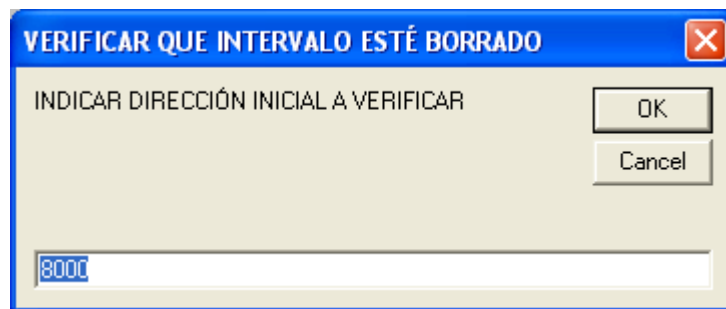


Figura 3.16. Dialogo para especificar dirección inicial a verificar por borrado.

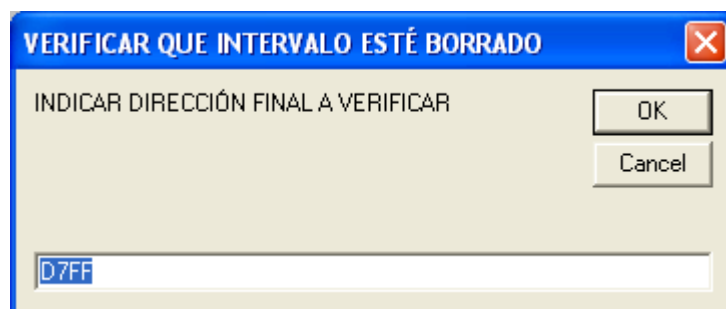


Figura 3.17. Dialogo para especificar dirección final a verificar por borrado.

Una vez que el usuario ha oprimido el botón “OK” en el dialogo de la figura 3.17, PUMMA_08+ procede a contar cuantas localidades de memoria contienen bytes que no sean \$FF y reporta al usuario la cuenta resultante. En caso de que ésta sea cero se muestra al usuario el mensaje ilustrado en la figura 3.18.

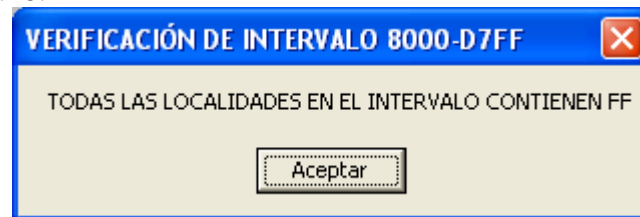


Figura 3.18. Mensaje mostrado al usuario cuando todas las localidades verificadas contienen \$FF

3.5.2 Accionamiento propio de la opción “Programar FEEP desde teclado como texto”

Si el usuario invoca esta opción, PUMMA_08+ le pedirá primero especificar la dirección inicial en la memoria FLASH, a partir de la cual ha de grabarse un texto. Véase la figura 3.19.

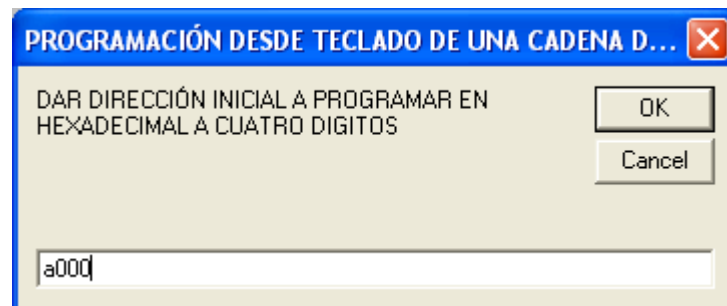


Figura 3.19. Dialogo para especificar la dirección inicial de memoria FLASH a grabar con texto

Para fines ilustrativos, en la figura 3.19 se especifica la dirección de FLASH \$A000. Después de que el usuario oprime el botón “OK”, PUMMA_08+ le pide teclear el texto a grabar; esto mediante el dialogo mostrado en la figura 3.20, donde para fines demostrativos se ha tecleado el texto: “Si funciona el proceso de grabación de texto en memoria FLASH”.

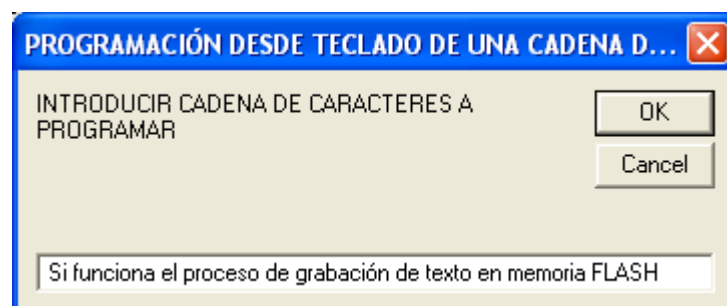


Figura 3.20. Dialogo para especificar texto por grabar en memoria FLASH.

Después de que el usuario oprime el botón “OK” en el dialogo de la figura 3.20, PUMMA_08+ procede a grabar en la memoria FLASH el texto en cuestión a partir de la dirección de memoria definida previamente para tal fin.

Es responsabilidad del usuario que en efecto exista memoria FLASH a partir de la dirección especificada. En la figura 3.21 se muestra un despliegue de la página \$A0, apreciándose ahí el texto definido ya grabado en la memoria FLASH.

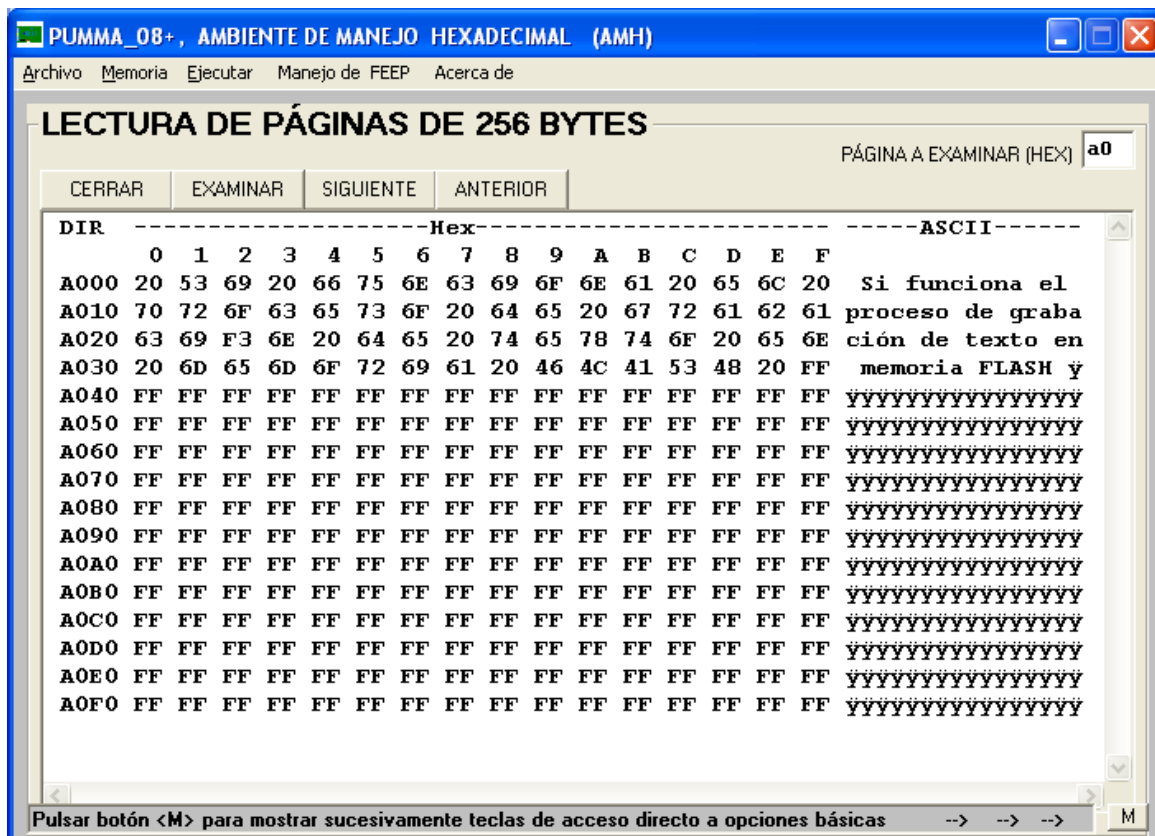


Figura 3.21. Despliegue de memoria de la TD mostrando texto grabado en ésta.

3.5.3 Accionamiento de la opción “Programar FEEP desde teclado como bytes (hex)”

Esta opción permite al usuario grabar en la memoria FLASH una lista de bytes denotados en hexadecimal, esto a partir de una dirección dada, la cual debe expresarse en hexadecimal empleando cuatro dígitos. Al invocarse esta facilidad, PUMMA_08+ presenta al usuario el dialogo mostrado en la figura 3.22; ahí el usuario debe definir la dirección a partir de la cual se grabará la lista de bytes implicada. En la figura 3.22 se define para fines ilustrativos la dirección \$A080.

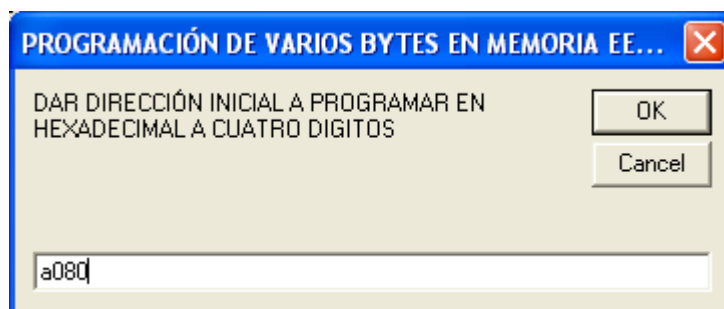


Figura 3.22. Dialogo para definir dirección inicial para grabar bytes en memoria FLASH.

Una vez que el usuario ha definido la dirección inicial a grabar, PUMMA_08+ le pide definir los bytes que se van a grabar en direcciones subsecuentes, a partir de la especificada en el dialogo de la figura 3.22. Para ello PUMMA_08+ presentará al usuario sendos diálogos donde éste debe definir los bytes a grabar. Para finalizar el proceso, suponiendo que el usuario va a escribir en ‘n’ localidades, basta con oprimir el botón “CANCEL” en el dialogo que presente PUMMA_08+ para la grabación del byte ‘n+1’. En las figuras 3.23 y 3.24 se muestran los diálogos para la

grabación de dos bytes a partir de la dirección \$A080, para fines ilustrativos los bytes a grabar son respectivamente \$61 y \$63.

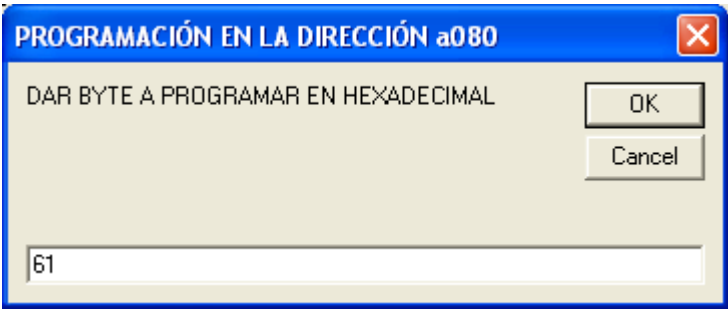


Figura 3.23. Dialogo para grabar un byte en la dirección \$A080.

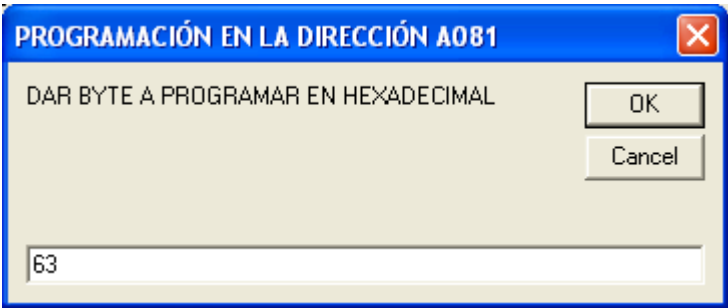


Figura 3.24. Dialogo para introducir un byte en la dirección \$A081.

Para la ejemplificación aquí mostrada, el usuario deberá oprimir el botón “CANCEL” en el dialogo donde se le pide el byte a escribir en la dirección \$A082, ya que sólo se pretende grabar dos bytes.

En la figura 3.25 se muestra un desplegado de memoria de la TD mostrándose ahí la página \$A0, apreciándose el resultado de los dos procesos de grabación de memoria FLASH desde teclado anteriormente descritos.

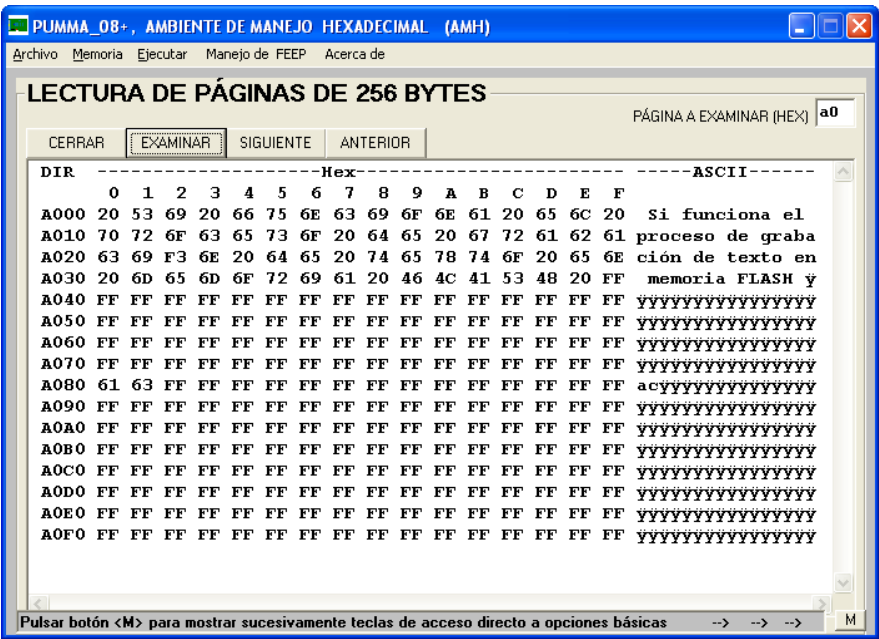


Figura 3.25. Despliegue de memoria de la TD después de grabar bytes desde teclado como texto y como lista de bytes.

3.5.4 Accionamiento propio de la opción “Programar FEEP desde archivo S19”

Mediante esta facilidad, el usuario puede grabar el contenido de un archivo S19 en la memoria FLASH, es responsabilidad de éste el hecho de que todas las direcciones especificadas en el archivo a grabar en efecto estén dentro del intervalo de memoria FLASH; de no ser el caso, se producirán errores diversos y un mal funcionamiento de la TD.

Al invocarse esta opción PUMMA_08+ presenta al usuario el dialogo mostrado en la figura 3.26, una vez que en éste se ha especificado el nombre del archivo S19 a grabar y se ha oprimido el botón “ABRIR”, PUMMA_08+ procede a grabar en memoria FLASH el contenido del archivo en cuestión, esto acorde con las direcciones presentes en el mismo.

Es importante destacar que antes de garbar la memoria FLASH, ésta deberá contener el BYTE \$FF en todas las localidades implicadas por el archivo a grabar; en caso contrario, PUMMA_08+ notifica esto al usuario y cancela el proceso de grabación. Para corregir esto bastará con borrar la memoria FLASH de usuario (véase más adelante el submenú “Borrar FEEP”) y repetir el proceso.

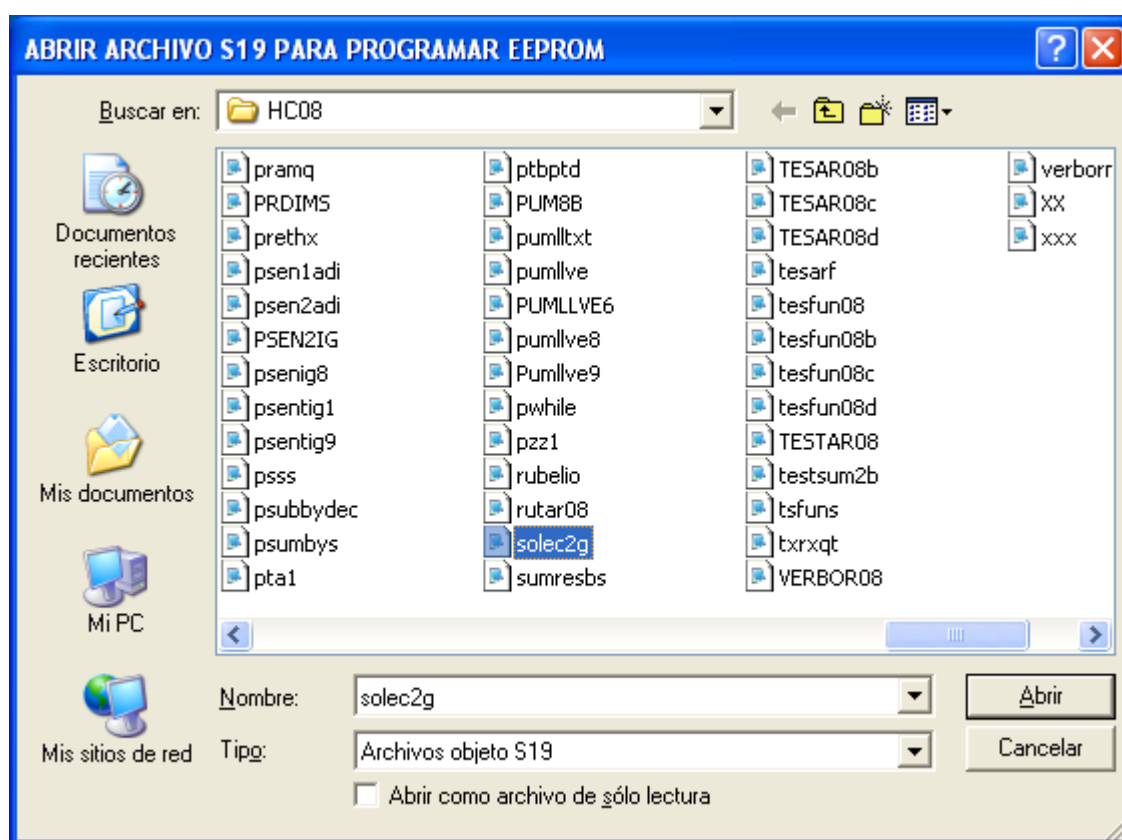


Figura 3.26. Dialogo para especificar archivo S19 a grabar en memoria FLASH.

3.5.5 Accionamiento propio de la opción “Verificar FEEP contra archivo S19”

Mediante esta opción, el usuario puede verificar el contenido de memoria contra la información presente en un archivo S19 estándar. Al invocarse, PUMMA_08+ presenta el dialogo mostrado en la figura 3.27; una vez que en éste se ha especificado el nombre del archivo a comparar con memoria y se ha oprimido el botón “ABRIR”, PUMMA_08+ procede a efectuar la comparación. Si los contenidos en el archivo y memoria son idénticos, esto se le notifica al usuario con un mensaje como el apreciado en la figura 3.28; en otro caso, se despliega un mensaje como el mostrado en la figura 3.29.

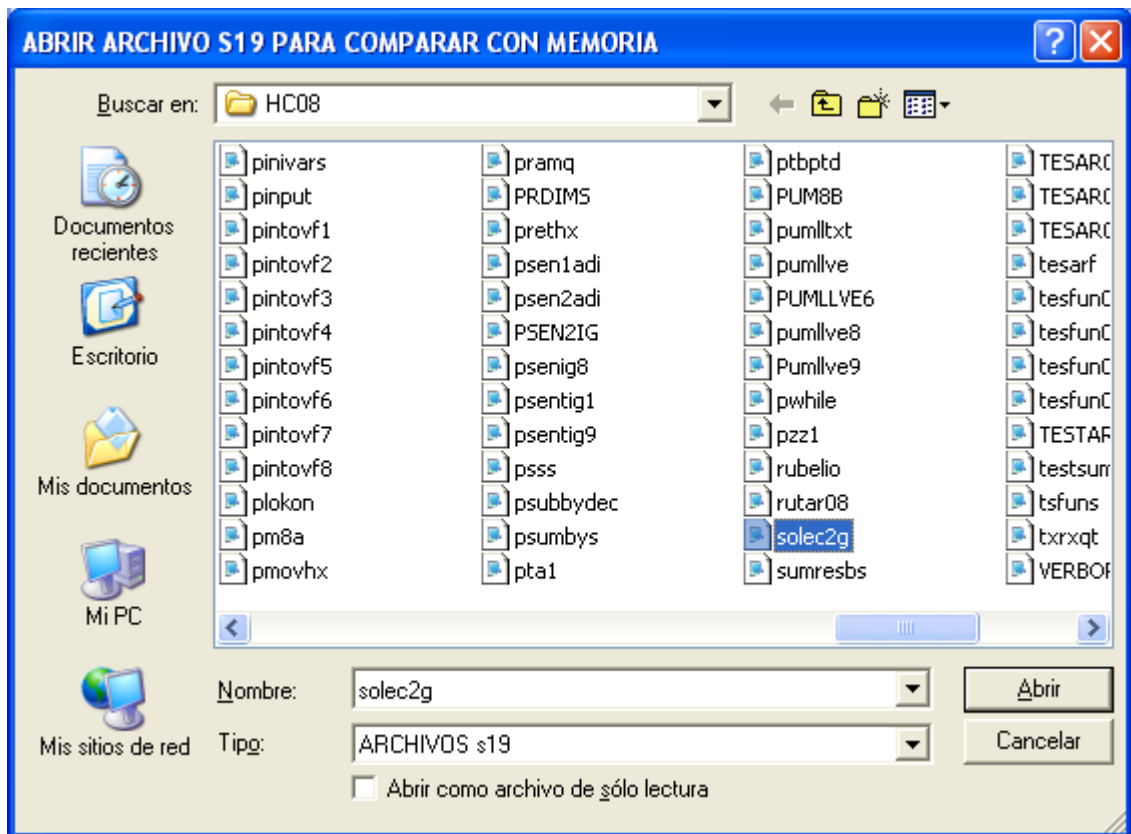


Figura 3.27. Dialogo para especificar archivo S19 a comparar con memoria.

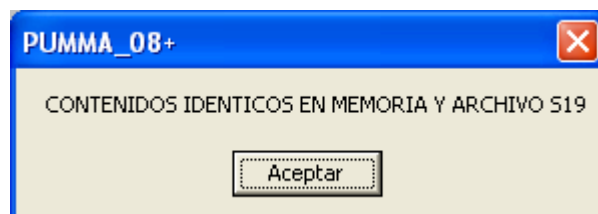


Figura 3.28. Mensaje indicando que los contenidos de memoria y archivo S19 son idénticos.

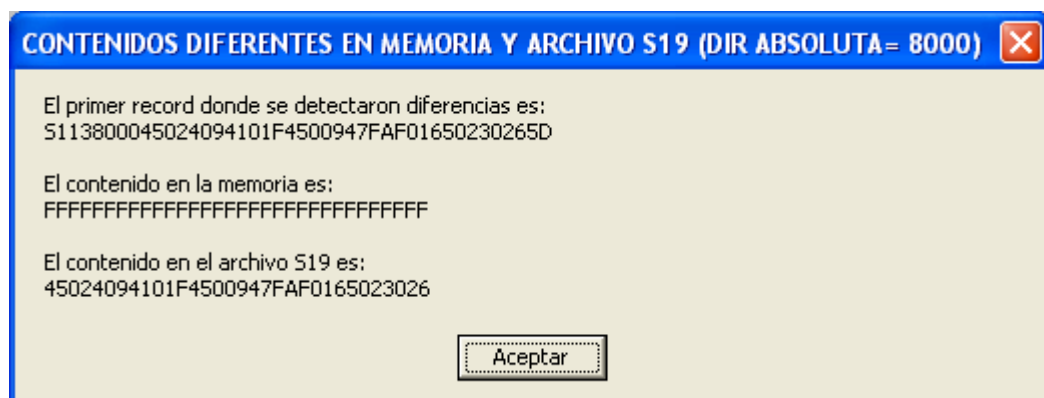


Figura 3.29. Forma de mensaje indicador de discrepancias entre memoria y archivo S19 especificado.

3.5.6 Accionamiento propio de la opción “Borrar FEEP”

Al invocarse esta opción, PUMMA_08+ procede a borrar la memoria FLASH de usuario presente en el MCU de la TD. El proceso toma alrededor de 15 segundos, al concluir éste, todas las localidades implicadas contendrán el byte \$FF.

3.6 Menú “Acerca de” del AMH

Al invocarse este menú se despliega una ventana que contiene información básica y autoral acerca de PUMMA_08+. Véase la figura 3.30.

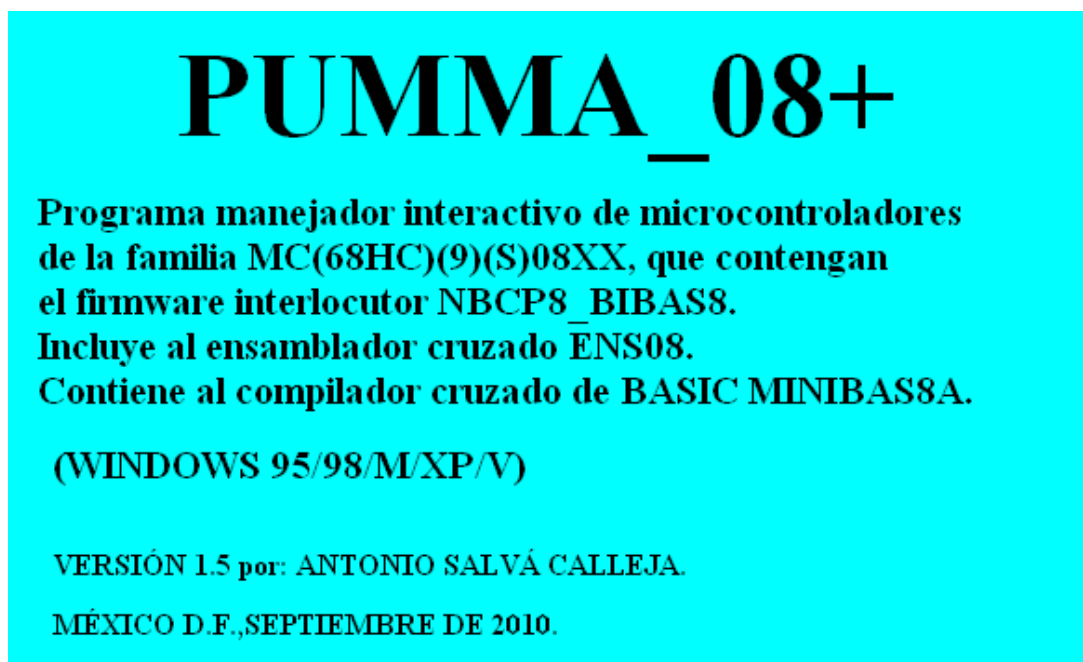


Figura 3.30. Ventana desplegada al invocarse el menú “Acerca de” del AMH.

3.7 Descripción básica de las facilidades presentes en el editor de PUMMA_08+

En la figura 3.31 se muestra la ventana de edición de PUMMA_08+, donde se aprecia un programa fuente en ensamblador. Ahí se pueden ver varios botones que disparan la ejecución de diversas acciones que se efectúan al oprimirlos el usuario. Los accionamientos ligados con los seis botones presentes a la izquierda son los básicos asociados con el funcionamiento del editor como pueden ser: guardar el archivo presente, cortar y pegar entre otros. Para el botón ubicado en el extremo derecho el accionamiento es simplemente minimizar la ventana.

Además de los botones mencionados en el párrafo anterior, existen otros seis, cinco situados en la parte superior central de la ventana del editor, las funciones asociadas con estos están vinculadas con procesos de compilación y/o ensamble del código fuente presente, el otro botón de los seis mencionados, está asociado con el proceso de borrar la memoria FLASH de usuario.

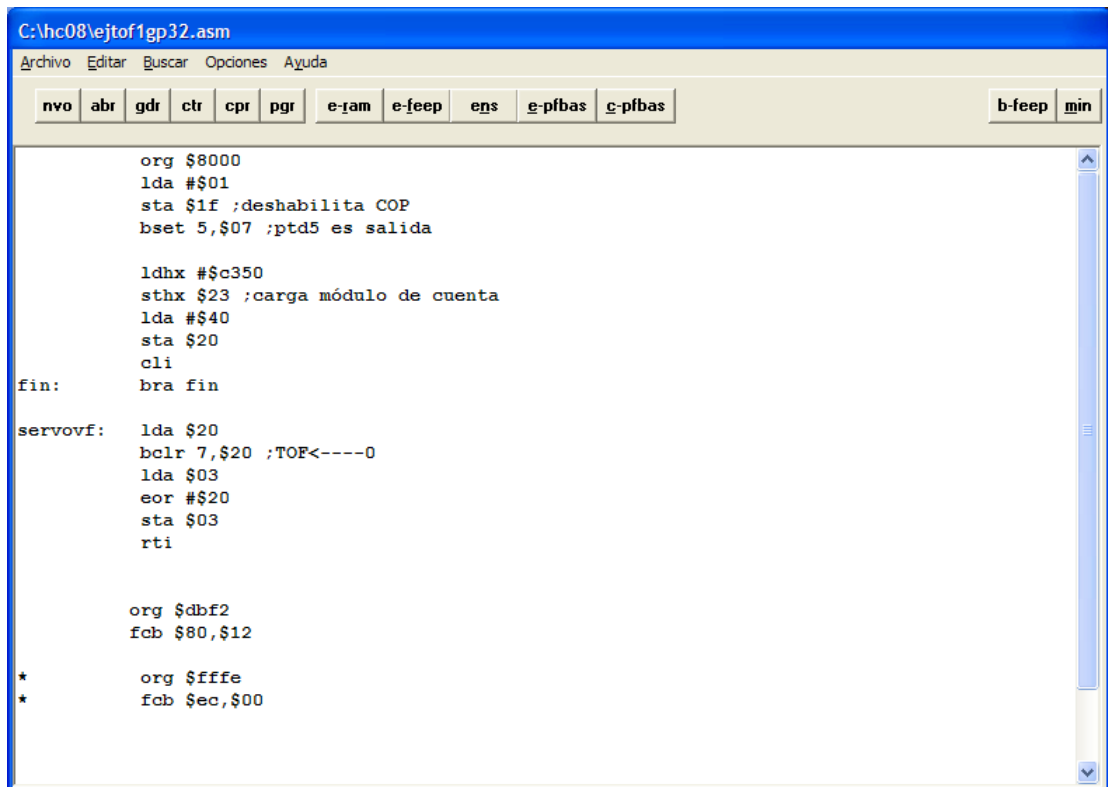


Figura 3.31. Ventana del editor de PUMMA_08+, conteniendo un programa fuente en ensamblador.

A continuación se describen los accionamientos asociados con los cinco botones centrales presentes en la ventana de edición:

3.7.1 Botón con la leyenda “e-ram”, (ejecución inmediata en memoria RAM)

Al oprimirse este botón, se ensambla el programa presente en la ventana del editor y si no hay errores de sintaxis, éste se carga y ejecuta de inmediato en la TD. En caso de haber errores, al primero de estos se detiene el proceso de ensamble y se reporta al usuario éste.

El programa fuente debe estar en lenguaje ensamblador, si no es este el caso se producirán errores diversos. Es responsabilidad del usuario el que para las direcciones de carga que correspondan al código objeto, en efecto exista memoria RAM en la TD, si éste no es el caso se tendrá seguramente un mal funcionamiento en la TD.

3.7.2 Botón con la leyenda “e-feep”, (ejecución inmediata en memoria FLASH)

Al oprimirse este botón se ensambla el programa presente en la ventana del editor y si no hay errores de sintaxis, éste se carga y ejecuta de inmediato en la TD. En caso de haber errores, al primero de estos se detiene el proceso de ensamble y se reporta al usuario éste.

El programa fuente debe estar en lenguaje ensamblador, si no es éste el caso se producirán errores diversos. Es responsabilidad del usuario el que para las direcciones de carga que correspondan al código objeto, en efecto exista memoria FLASH en la TD, si éste no es el caso se tendrá seguramente un mal funcionamiento en la TD.

3.7.3 Botón con la leyenda “ens”

Al oprimirse este botón se ensambla el programa presente en la ventana del editor. En caso de haber errores, al primero de estos se detiene el proceso de ensamble y se reporta al usuario éste.

El programa fuente debe estar en lenguaje ensamblador, si no es éste el caso se producirán errores diversos. Si no hay errores se generan los archivos de salida np.s19 y np.lst,

convencionales. Se supone que el programa fuente original está contenido en el archivo denominado np.asm, siendo “np” el nombre que el usuario le haya dado a éste.

Cabe señalar que después de haber generado los archivos de salida mencionados en el párrafo anterior, no se genera ningún accionamiento que implique la carga en la TD del código de máquina generado presente en el archivo np.s19.

3.7.4 Botón con la leyenda “e-pfbas”, (compilación y ejecución inmediata)

Al oprimirse este botón se compila el programa presente en la ventana del editor y si no hay errores léxicos, sintácticos o semánticos, éste se carga y ejecuta de inmediato en la TD. En caso de haber errores, se reportan estos al usuario en una ventana especial para tal fin.

El programa fuente debe estar en lenguaje BASIC manejable por el compilador MINIBAS8 presente en PUMMA_08+; si no es el caso se producirán errores diversos. Es responsabilidad del usuario el que para las direcciones de carga que correspondan al código objeto, en efecto exista memoria FLASH en la TD, si éste no es el caso se tendrá seguramente un mal funcionamiento en la TD.

3.7.5 Botón con la leyenda “c-pfbas”

Al oprimirse este botón se compila el programa presente en la ventana del editor. En caso de haber errores léxicos, sintácticos o semánticos, se reportan estos al usuario en una ventana especial para tal fin.

El programa fuente debe estar en lenguaje BASIC manejable por el compilador MINIBAS8 presente en PUMMA_08+; si no es el caso se producirán errores diversos. Si no hay errores se generan diversos archivos de salida, entre los que destaca el denominado como np.s19, que contiene el código de máquina ejecutable en un MCU de la familia HC08. Se supone que el programa fuente original está contenido en el archivo denominado np.b, siendo “np” el nombre que el usuario le haya dado a éste.

Cabe señalar que después de haber generado los archivos de salida mencionados en el párrafo anterior, no se genera ningún accionamiento que implique la carga en la TD del código de máquina generado presente en el archivo np.s19.

A continuación se describen los diversos menús y submenús presentes en la ventana del editor de PUMMA_08+.

3.8 Menú “Archivo” del editor de PUMMA_08+

El menú archivo del editor de PUMMA_08+ cuenta con los siguientes submenús:

- Nuevo
- Abrir
- Guardar
- Guardar como
- Ensamblar para ejecutar de inmediato en RAM
- Ensamblar para ejecutar de inmediato en FEED
- Ensamblar para generar archivos s19 y lst
- Manejador Hexadecimal (AMH)
- Invocar emulador de Terminal
- Configuración de puerto serie
- Salir

En seguida se describen los accionamientos que se dan al invocarse cada una de las opciones anteriores.

3.8.1 Accionamiento propio de la opción “Nuevo”

Al invocarse esta opción, se limpia la ventana del editor y se asigna por defecto el nombre NUEVO.ASM al programa que se introduzca a partir de la pantalla en blanco. **Es importante destacar que al iniciar la introducción de un programa nuevo, siempre se le debe dar un nombre a éste que no sea la cadena “NUEVO”.** Para ello bastará con invocar la opción “guardar” descrita más adelante. Si el nombre del programa es NUEVO.ASM, PUMMA_08+ presentará al usuario el dialogo asociado con la opción “Guardar como”; ahí el usuario podrá darle un nombre al programa con el que trabaje en un momento dado.

3.8.2 Accionamiento propio de la opción “Abrir”

Al seleccionarse este submenú, PUMMA_08+ presenta al usuario un dialogo donde éste puede definir el nombre y carpeta asociados con el archivo que se pretenda abrir en un momento dado. Una vez que el usuario ha oprimido el botón “Abrir”, el contenido del archivo seleccionado aparece en el área de texto de la ventana del editor y el nombre de éste es desplegado en la parte superior de ésta. En la figura 3.32 se muestra el dialogo aquí mencionado. Para fines ilustrativos, ahí se ha seleccionado abrir el archivo demclock.asm presente en la carpeta “hc08”.

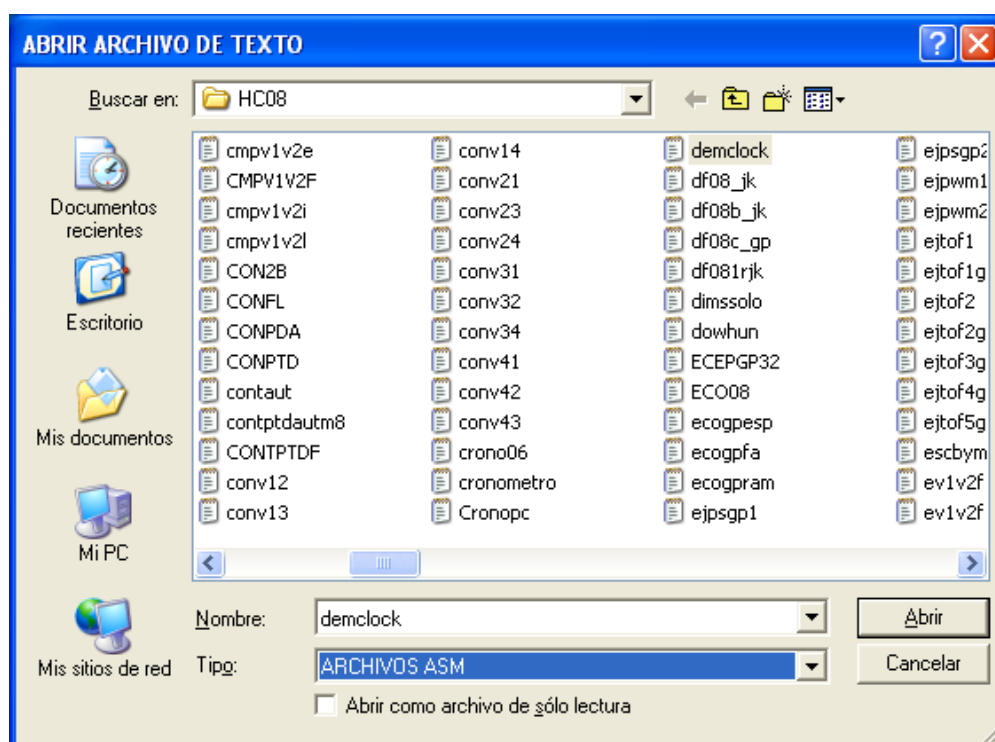


Figura 3.32. Dialogo para abrir un archivo a colocar en el editor de PUMMA_08+

3.8.3 Accionamiento propio de la opción “Guardar”

Esta opción conduce simplemente a guardar el archivo presente en el editor, esto bajo el nombre que éste tenga testificado en el cintillo superior de la ventana de edición.

3.8.4 Accionamiento propio de la opción “Guardar como”

Al invocarse esta opción, se da al usuario la oportunidad para guardar el contenido del editor bajo otro nombre y/o carpeta. El dialogo que presenta PUMMA_08+ se muestra en la figura 3.33, donde se asume que el usuario definió que la información en el editor se guarde bajo el nombre archnvo.asm en la carpeta “HC08”.

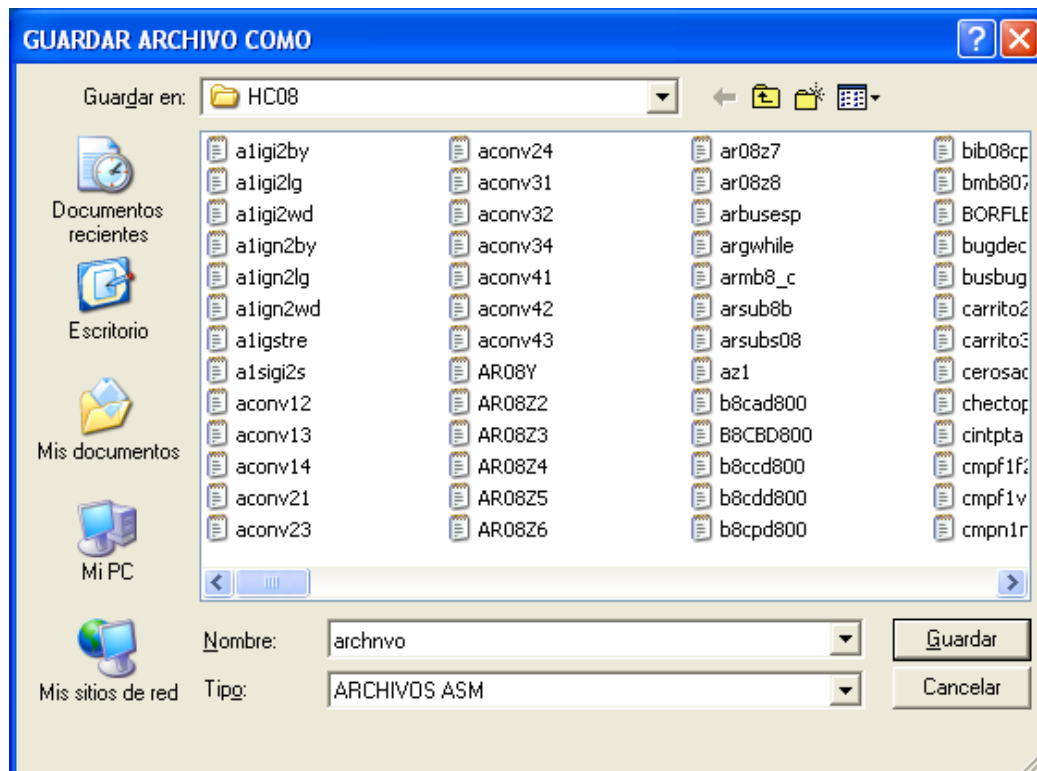


Figura 3.33. Dialogo para guardar un archivo bajo otro nombre.

Una vez que el usuario ha oprimido el botón “Guardar”, la información en el editor es guardada bajo el nombre y carpeta definidos en el dialogo propio de esta opción mostrado en la figura 3.33.

3.8.5 Accionamiento propio de la opción “Ensamblar para ejecutar de inmediato en RAM”

Esta opción conduce al mismo accionamiento que se dispara al oprimirse el botón “e-ram”. Véase la descripción que sobre este particular se ha hecho anteriormente en este manual, bajo el título: Botón con la leyenda “e-ram”, (ejecución inmediata en memoria RAM).

3.8.6 Accionamiento propio de la opción “Ensamblar para ejecutar de inmediato en FEEP”

Esta opción conduce al mismo accionamiento que se dispara al oprimirse el botón “e-feep”. Véase la descripción que sobre este particular se ha hecho anteriormente en este manual, bajo el título: Botón con la leyenda “e-feep”, (ejecución inmediata en memoria FEEP).

3.8.7 Accionamiento propio de la opción “Ensamblar para generar archivos s19 y lst”

Esta opción conduce al mismo accionamiento que se dispara al oprimirse el botón “ens”. Véase la descripción que sobre este particular se ha hecho anteriormente en este manual, bajo el título: Botón con la leyenda “ens”.

3.8.8 Accionamiento propio de la opción “Manejador Hexadecimal (AMH)”

Este submenú conduce simplemente a la apertura y presentación al usuario de la ventana del Ambiente de Manejo Hexadecimal (AMH).

3.8.9 Accionamiento propio de la opción “Invocar emulador de Terminal”

Esta opción conduce a la apertura del emulador de terminal presente en PUMMA_08+. Su funcionalidad es similar a la opción del mismo nombre presente en el menú “Archivo” de la ventana de manejo hexadecimal (AMH).

3.8.10 Accionamiento propio de la opción “Configuración de puerto serie”

Esta opción conduce a predeterminar el puerto serie a emplear en el enlace con la TD. Su funcionalidad es similar a la opción del mismo nombre presente en el menú “Archivo” de la ventana de manejo hexadecimal (AMH).

3.8.11 Accionamiento propio de la opción “Salir”

Esta opción conduce al cierre de la aplicación. En caso de que el usuario haya hecho alguna modificación del archivo presente en el área de edición, PUMMA_08+ le advierte que el archivo no ha sido guardado, dándole la opción para hacer esto antes de terminar la aplicación.

3.9 Menú “Editar” del editor de PUMMA_08+

El menú “editar” del ambiente de edición de PUMMA_08+ cuenta con los siguientes submenús:

- Cortar
- Copiar
- Pegar

Los accionamientos que se dan al invocarse cada una de las opciones anteriores, son los propios con ese nombre, existentes en un ambiente de edición de texto bajo WINDOWS. Los accionamientos que se dan al seleccionar estas opciones haciendo *click* en el submenú correspondiente requieren ajustes finos, por lo que se recomienda al usuario invocar estas opciones empleando sus *shortcuts* asociados los cuales son: ctrl.+X para copiar, ctrl.+C para copiar y ctrl.+V para pegar.

3.10 Menú “Buscar” del editor de PUMMA_08+

El menú “Buscar” de la ventana del editor de texto del ensamblador ENS08, contiene las tres opciones de búsqueda de texto, más comunes: *Buscar*, *Repetir última búsqueda* y *Cambiar*. Estos comandos permiten al editor de texto del ensamblador ENS08 de PUMMA_08+, realizar las tareas básicas de búsqueda y reemplazo de texto dentro de un listado fuente de programa. Este menú cuenta con los siguientes submenús:

- Buscar
- Repetir última búsqueda
- Cambiar

En seguida se describen los accionamientos que se dan al invocarse cada una de las opciones anteriores.

3.10.1 Accionamiento propio de la opción “Buscar”

Al invocarse esta opción, Este comando abre un dialogo como el mostrado en la figura 3.34 y permite al usuario realizar una búsqueda de texto dentro del listado fuente del programa presente en el área de texto del editor de PUMMA_08+. Para realizar una búsqueda, basta con teclear la cadena deseada a localizar, en el campo “*Buscar:*” y luego pulsar en el botón “ACEPTAR”. La búsqueda de la cadena de texto se realizará de arriba abajo y de izquierda a derecha.

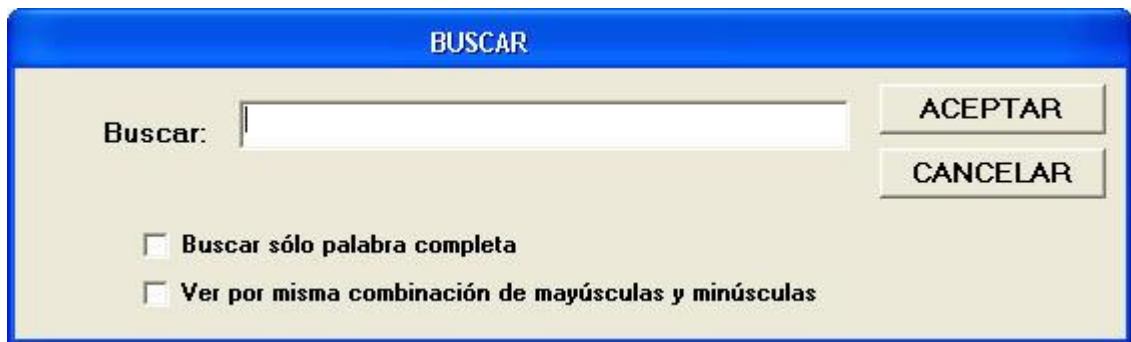


Figura 3.34. Ventana “*Buscar*” que permite introducir la cadena de texto a buscar dentro de un listado fuente de programa presente en el área de texto del editor del ensamblador ENS08

Si la búsqueda es exitosa y existen coincidencias dentro del texto del listado fuente del programa presente en el área de texto del editor, el comando “*Buscar*” resaltará la primera coincidencia que se ha encontrado.

Si por alguna razón, el comando “*Buscar*” no encuentra coincidencias de cadenas de texto dentro del listado fuente de programa presente en la ventana de edición de texto, enviará como resultado el mensaje avisando que no se han encontrado coincidencias de texto.

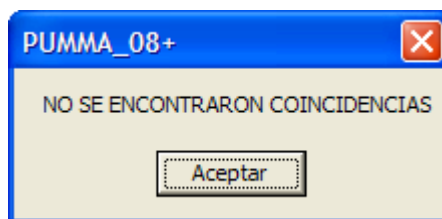


Figura 3.35. Mensaje que avisa al usuario que el comando “*Buscar*” no ha encontrado coincidencias de texto

Al seleccionar la opción “Buscar sólo palabra completa” de la ventana “*Buscar*”, podrá buscar sólo palabras completas dentro del texto del listado fuente de programa y no aquellas palabras que además, contengan la cadena que desea buscar. La opción “Ver por misma combinación de mayúsculas y minúsculas” de la ventana “*Buscar*”, le permitirá buscar sólo aquellas palabras que coincidan en mayúsculas y minúsculas con la cadena de texto que ha tecleado para su búsqueda. Existe la posibilidad de utilizar ambas opciones a la vez, y con la utilización de estas dos opciones, podrá buscar palabras completas que coincidan tanto en mayúsculas y minúsculas y, con esto, aumentará la precisión y efectividad de las búsquedas que desee realizar.

Una vez que se ha tecleado la cadena que desea buscar en la ventana “*Buscar*” y que se han seleccionado las opciones que al usuario le sean útiles de esta ventana, se debe pulsar el botón “ACEPTAR” de la ventana para iniciar la búsqueda de la cadena de texto. Si se desea abortar la búsqueda, basta con pulsar sobre el botón “CANCELAR”.

3.10.2 Accionamiento propio de la opción “Repetir última búsqueda”

Este comando es muy útil cuando se desea repetir una búsqueda de texto (previamente realizada con el comando “*Buscar*”) dentro del listado fuente de programa. Esta opción sólo se activa si se ha realizado por lo menos una búsqueda previa de texto, de otra manera, esta opción se encontrará desactivada.

3.10.3 Accionamiento propio de la opción “Cambiar”

Al invocar este comando aparecerá el dialogo mostrado en la figura 3.36. Esta opción, permite al usuario buscar una cadena de texto (introducida en el campo “*Buscar:*”) dentro del listado fuente de programa y si se han encontrado coincidencias, dicha cadena de texto es reemplazada completamente por la cadena de texto introducida en el campo “*Cambiar por:*”. El funcionamiento de esta ventana es similar al de la ventana “*Buscar*” mencionada con anterioridad.

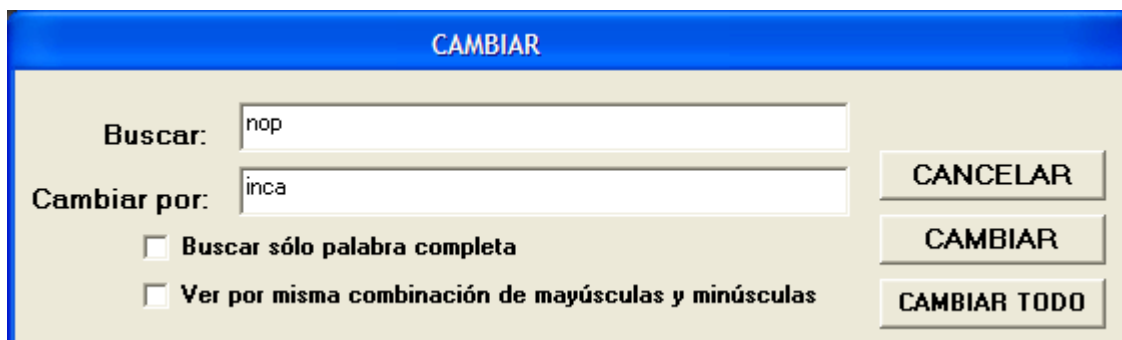


Figura 3.36 Ventana “*Cambiar*” que permite introducir la cadena de texto a buscar y la cadena de texto a reemplazar dentro de un listado fuente de programa presente en el área de texto del editor de PUMMA_08+

Para realizar un reemplazo de texto dentro del listado fuente de programa presente en la ventana de edición de PUMMA_08+, primeramente, es necesario introducir la cadena de texto a buscar en el campo marcado como “*Buscar:*”, en segundo lugar, se debe teclear en el campo “*Cambiar por:*”, la cadena de texto que sustituirá a las cadenas de texto que coincidan durante la búsqueda. Una vez hecho esto, se debe pulsar en el botón “CAMBIAR” para reemplazar una a una las coincidencias de texto que se considere pertinentes o bien, se puede pulsar en el botón “CAMBIAR TODO” para reemplazar todas las coincidencias encontradas sin que PUMMA_08+ pregunte si se deben o no reemplazar. Es importante que utilice con cuidado este botón para evitar cambios indeseados en el listado fuente de programa.

La búsqueda de la cadena de texto se realizará de arriba abajo y de izquierda a derecha. En caso de que se desee cancelar la acción de cambiar la cadena de texto, basta con pulsar el botón “CANCELAR”.

Si se decide pulsar el botón “CAMBIAR”, al encontrar coincidencias de la cadena de texto buscada dentro del listado fuente del programa, se resaltarán y se preguntará al usuario en cada caso si desea o no reemplazarla, véase la figuras 3.37.

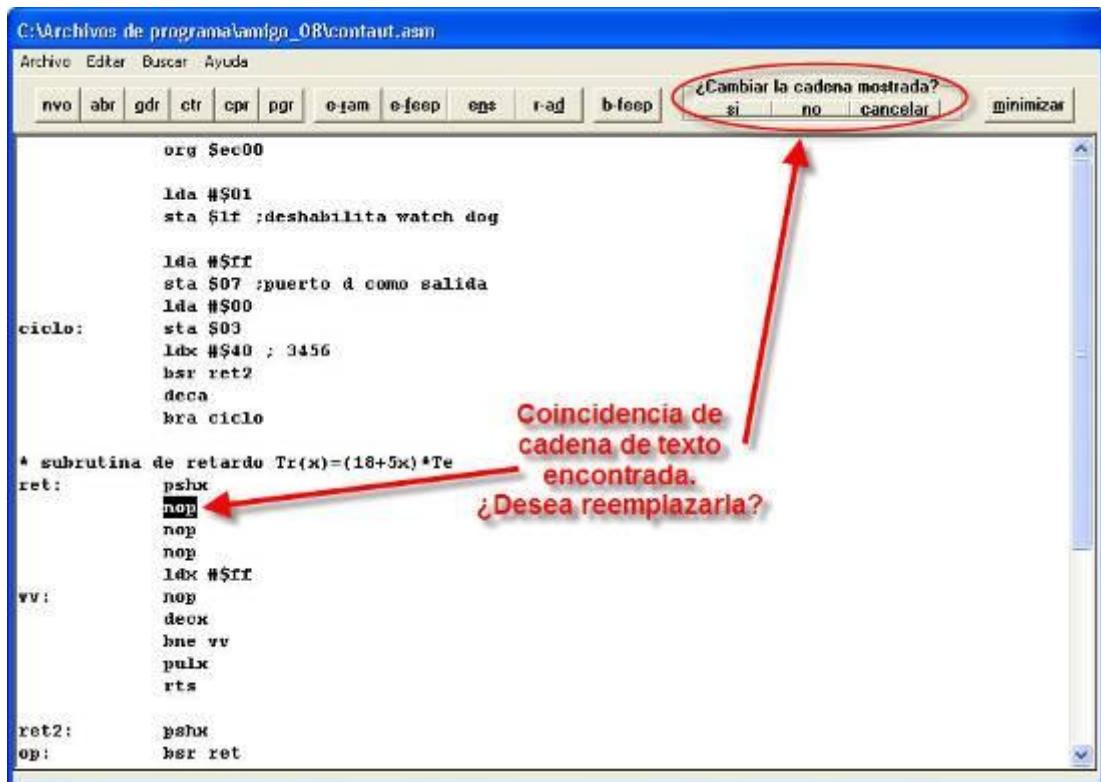


Figura 3.37. Coincidencia de cadena de texto encontrada por el comando “Cambiar”

Si en el momento en que se ha encontrado una coincidencia, el usuario pulsa el botón “si”, entonces la cadena encontrada y seleccionada será reemplazada por la cadena de texto que insertó, previamente, en el campo “Cambiar por:”. De otra manera, si decide hacer click en el botón “no”, entonces se seguirán buscando otras cadenas que coincidan con el criterio de búsqueda, en caso de encontrarlas, se seleccionarán y se le volverá a preguntar si desea o no reemplazarlas. Si se desea abortar la búsqueda y el reemplazo de cadenas de texto, bastará con pulsar el botón “cancelar”. En el momento que ya no existan coincidencias de cadenas de texto, se desplegará el mensaje mostrado en la figura 3.38.

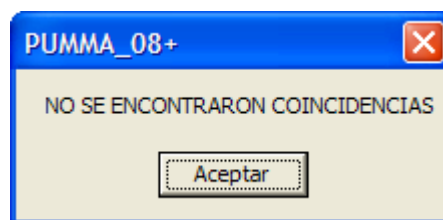


Figura 3.38 Menaje indicando que comando “Cambiar” y “Cambiar todo” ha terminado de encontrar coincidencias

Por otro lado, si decide hacer click en el botón “CAMBIAR TODO”, automáticamente se reemplazarán todas las coincidencias que se encuentren en el listado fuente del programa y una vez que se haya terminado de realizar los reemplazos, aparecerá un mensaje similar al mostrado en la figura 3.38.

Al igual que la ventana “Buscar”, la ventana “Cambiar” también tiene la opción “Buscar sólo palabra completa”, la cual si se selecciona, podrá buscar sólo palabras completas dentro del texto del listado fuente de programa y no aquellas palabras que además, contengan la cadena que desea buscar.

La opción “Ver por misma combinación de mayúsculas y minúsculas” de la ventana “*Cambiar*”, permitirá al usuario buscar sólo aquellas palabras que coincidan en mayúsculas y minúsculas con la cadena de texto que ha tecleado para su búsqueda.

3.11 Menú “Opciones” del editor de PUMMA_08+, (Configuración de procesador destino)

Este menú contiene opciones de configuración de aspectos tales como: el perfil de memoria en el chip destino, para fines de la ejecución de un programa en BASIC, cuyo código ejecutable sea generado por el compilador MINIBAS8; o bien, la habilitación o deshabilitación de la testificación de el número de línea en el programa fuente, para el cual se produzca un error en tiempo de ejecución.

Este menú cuenta con los siguientes submenús:

- Memoria en chip destino
- Testificación de renglón de error al ejecutar

En seguida se describen los accionamientos que se dan al invocarse cada una de las opciones anteriores.

3.11.1 Accionamiento propio de la opción “Memoria en chip destino”

Este submenú permite al usuario predefinir el perfil de memoria asociado con el chip destino, donde correrá el código ejecutable asociado con el programa fuente en BASIC, que se compile en un momento dado. Al invocarse esta opción, PUMMA_08+ presenta al usuario el dialogo mostrado en la figura 3.39. En éste el usuario debe elegir el chip destino para después oprimir el botón aplicar. En la figura se muestra el aspecto del dialogo mencionado cuando no se ha predeterminado ningún perfil de memoria para el procesador destino.

GENERADOR DE CÓDGO EN ENSAMBLADOR HC08

SP INICIAL

DIRECCIÓN INICIAL DE CÓDIGO (DIC)

DIRECCIÓN INICIAL DE DATOS (DID)

DIRECCIÓN FINAL DE DATOS (DFD)

SELECCIONAR TIPO DE MCU DESTINO

CHIP DESTINO

☐ SP INICIAL Y DIC MODIFICABLES

☐ DFD ACORDE CON INTERRUPCIONES EN BASIC

APLICAR ACEPTAR

Figura 3.39. Dialogo para predeterminar perfil de memoria del chip destino.

Una vez que el usuario ha seleccionado el chip destino y ha oprimido el botón aplicar, PUMMA_08+ presenta el dialogo mostrado en la figura 3.40. Esto si el usuario seleccionó como chip destino al CHIPBAS8 GP32.

GENERADOR DE CÓDIGO EN ENSAMBLADOR HC08

CONFIGURACIÓN BÁSICA DE MEMORIA PARA EL MCU DESTINO

SP INICIAL: &H023F

SELECCIONAR TIPO DE MCU DESTINO: CHIPBAS8_GP32

DIRECCIÓN INICIAL DE CÓDIGO (DIC): &H8000

DIRECCIÓN INICIAL DE DATOS (DID): &H0047

DIRECCIÓN FINAL DE DATOS (DFD): &H22F

☐ SP INICIAL Y DIC MODIFICABLES

☐ DFD ACORDE CON INTERRUPCIONES EN BASIC

APLICAR ACEPTAR

Figura 3.40. Confirmación a usuario de predeterminación de perfil de memoria asociado con el procesador destino CHIPBAS8 GP32.

Los parámetros de memoria que se predeterminan son:

- Dirección de inicialización del apuntador de pila (Stackpointer), denotada como “SP INICIAL”
- Localidad inicial de memoria no volátil, a partir de la cual se cargará el código ejecutable en el chip destino. Esta localidad se denota como “DIRECCIÓN INICIAL DE CÓDIGO” y se abrevia “DIC”.
- Localidad inicial de memoria RAM donde se han de colocar las variables de usuario e internas que genere el compilador, además de las localidades de intercambio que emplean las rutinas de biblioteca propias de MINIBAS8. Esta localidad se define como “DIRECCIÓN INICIAL DE DATOS” y se abrevia “DID”.
- Dirección tope en memoria RAM para la colocación de variables de programa. A ésta se le denomina como “DIRECCIÓN FINAL DE DATOS” y se abrevia “DFD”.

En caso de que se intente compilar un programa fuente y no se haya predefinido el perfil de memoria, PUMMA_08+ presentará al usuario el mensaje mostrado en la figura 3.41.

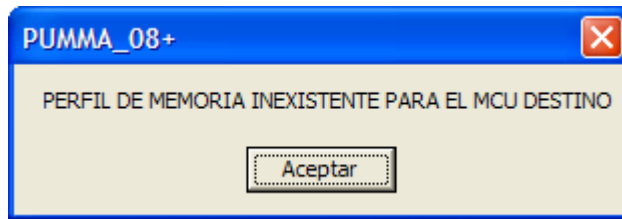


Figura 3.41. Mensaje indicando que no se ha predeterminado perfil de memoria.

Si el usuario lo deseara, podrá predeterminar valores diferentes para el valor inicial del *stackpointer* y para la dirección inicial de colocación de código. Para esto deberá efectuar lo siguiente:

- En el dialogo de predeterminación de perfil de memoria, seleccionar la caja con la leyenda “SP INICIAL Y DIC MODIFICABLES”.
- Oprimir el botón aplicar.
- Modificar los parámetros aquí mencionados.
- Volver a oprimir el botón aplicar.

En la figura 3.42 se muestra la confirmación del perfil de memoria cuando el usuario ha hecho que el valor inicial del stackpointer sea \$0230 y la dirección inicial de colocación de código sea \$A000.

Figura 3.42. Confirmación de perfil de memoria donde el usuario ha definido valores especiales para los parámetros “SP INICIAL” y “DIC”.

3.11.2 Predeterminación de parámetro DFD acorde con interrupciones en BASIC

Al diseñar programas en BASIC que contengan rutinas de servicio de interrupción escritas en lenguaje BASIC, éstas podrían llegar a requerir una pila de mayor tamaño a la que se define por defecto (16 bytes). Esto se debe a que, bajo estas circunstancias, en la rutina de servicio se

deberán preservar en la pila (stack) el contenido de las localidades de intercambio de las bibliotecas de MINIBAS8 lo cual hace que la pila requiera de 73 localidades adicionales a las 16 mencionadas anteriormente. Esto podría generar colisiones de la pila con la zona de variables del programa; para evitar esto, la dirección final de colocación de datos debe estar 73 localidades más abajo. Para ello, previamente a la opresión del botón “APLICAR”, se debe “chechar” la caja de opción que contiene la leyenda “DFD ACORDE CON INTERRUPCIONES EN BASIC”; esto hará que el parámetro DFD esté 89 localidades por debajo del valor inicial del apuntador de pila y no solo 16, véase la figura 3.43.

The screenshot shows a software window titled "GENERADOR DE CÓDGO EN ENSAMBLADOR HC08". It contains a section titled "CONFIGURACIÓN BÁSICA DE MEMORIA PARA EL MCU DESTINO". Within this section, there are several input fields and checkboxes:

- SP INICIAL:** A text box containing the hexadecimal value "&H023F".
- SELECCIONAR TIPO DE MCU DESTINO:** A dropdown menu currently showing "CHIPBAS8_GP32".
- DIRECCIÓN INICIAL DE CÓDIGO (DIC):** A text box containing the hexadecimal value "&H8000".
- DIRECCIÓN INICIAL DE DATOS (DID):** A text box containing the hexadecimal value "&H0047".
- DIRECCIÓN FINAL DE DATOS (DFD):** A text box containing the hexadecimal value "&H1E6".
- SP INICIAL Y DIC MODIFICABLES:** An unchecked checkbox.
- DFD ACORDE CON INTERRUPCIONES EN BASIC:** A checked checkbox.

At the bottom right of the configuration area, there are two buttons: "APLICAR" and "ACEPTAR".

Figura 3.43. Perfil de memoria acorde con rutinas de interrupción escritas en BASIC.

Cabe señalar que para programas en BASIC que usen pocas variables, el ajuste mencionado en el párrafo anterior pudiera ser no necesario, aún cuando se tengan rutinas de servicio de interrupción escritas en BASIC. Un criterio para determinar si es necesario el ajuste del parámetro DFD es checar si la dirección inicial de colocación de variables de tipo string (DICVS), tiene un valor que sea sensiblemente menor al valor del parámetro DFD sin hacer el ajuste (pila de 16 bytes).

El valor del parámetro DICVS, para un determinado programa en BASIC, que se compile después de la opresión del botón “c-pfbas”, es desplegado por MINIBAS8 como parte del mensaje de que el proceso de compilación y ensamble final han sido exitosos. Véase la figura 3.44 donde se aprecia que para el programa que recién se ha compilado dicho parámetro es el valor &HD5 expresado en notación hexadecimal (213 decimal).

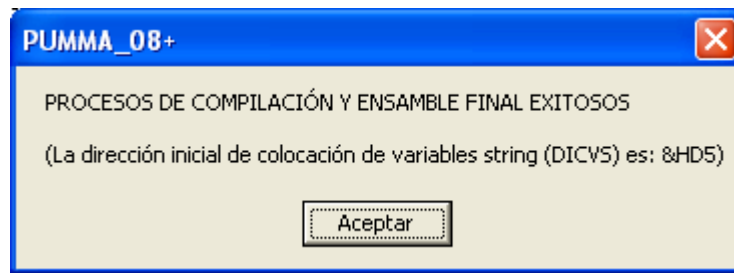


Figura 3.44. Mensaje de MINIBAS8 al haberse efectuado con éxito los procesos de compilación y ensamble final asociados con un programa en BASIC.

Dado que para el valor del parámetro DICVS mostrado en la figura 3.44, la diferencia entre éste y el parámetro DFD usando una pila de 16 bytes, es un valor para el cual el número 73 cabe varias veces, en este caso es improbable que se produzca una colisión de la pila con la zona de variables del programa. Por lo tanto, podrá definirse el perfil de memoria sin la habilitación de la opción “DFD ACORDE CON INTERRUPCIONES EN BASIC”; esto sin importar que el programa use interrupciones escritas en BASIC.

3.11.3 Accionamiento propio de la opción “Testificación de renglón de error al ejecutar”

La invocación de este submenú habilita o deshabilita la testificación en la consola del número de renglón en el programa fuente, asociado con un error en tiempo de ejecución, como podría ser entre otros una división entre cero.

Para habilitar la testificación bastará con invocar el submenú cuando ésta está deshabilitada. Si se invoca el submenú cuando la testificación está habilitada ésta se deshabilita.

Cuando esta opción está deshabilitada aparece sin “chequeo” el texto dentro de la barra de submenús asociada, en otro caso ésta aparece “checada.” Véanse las figuras 3.45 y 3.46.

Si la opción está deshabilitada, al presentarse un error en tiempo de ejecución, éste se reporta al usuario en la consola denotando como “cero” el número de renglón en el programa fuente donde se produce el error. En caso de que la testificación esté habilitada, se reporta el error y el número de renglón en el programa fuente donde éste se origina. Véanse las figuras 3.47 y 3.48.

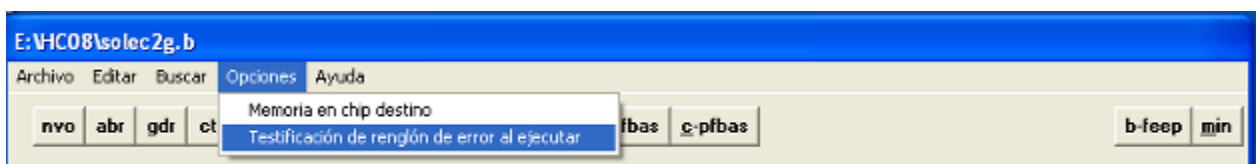


Figura 3.45. Aspecto del submenú “Testificación de renglón de error al ejecutar” cuando esta opción está deshabilitada.

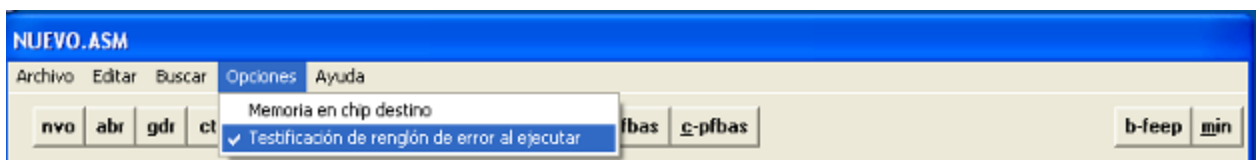
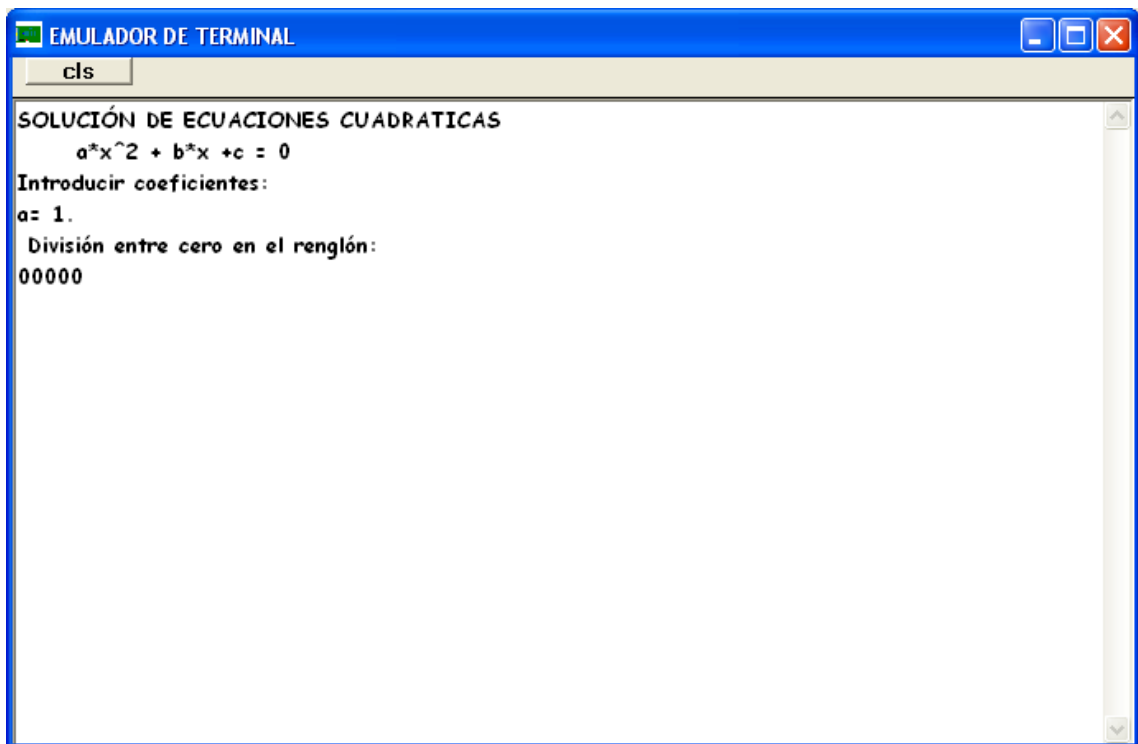


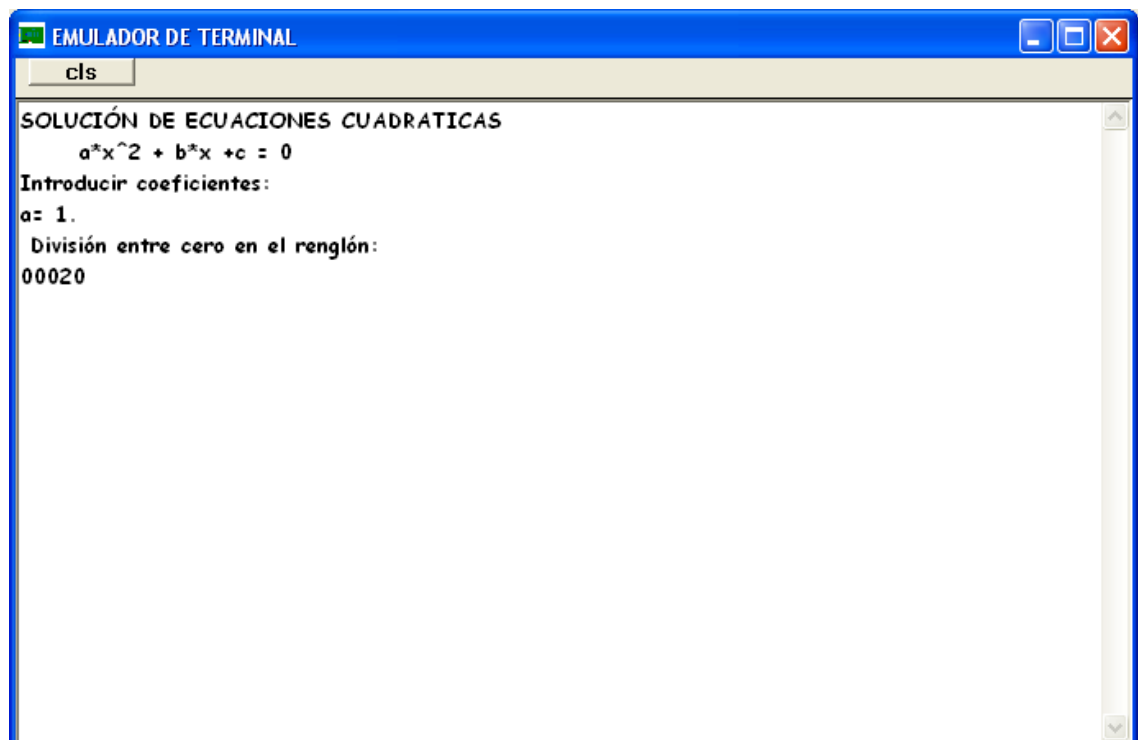
Figura 3.46. Aspecto del submenú “Testificación de renglón de error al ejecutar” cuando esta opción está habilitada.



```
EMULADOR DE TERMINAL
cls

SOLUCIÓN DE ECUACIONES CUADRATICAS
  a*x^2 + b*x +c = 0
Introducir coeficientes:
a= 1.
División entre cero en el renglón:
00000
```

Figura 3.47. Reporte en consola de error en tiempo de ejecución, con la testificación del número de renglón asociado deshabilitada.



```
EMULADOR DE TERMINAL
cls

SOLUCIÓN DE ECUACIONES CUADRATICAS
  a*x^2 + b*x +c = 0
Introducir coeficientes:
a= 1.
División entre cero en el renglón:
00020
```

Figura 3.48. Reporte en consola de error en tiempo de ejecución, con la testificación del número de renglón asociado habilitada.

3.12 Menú “Ayuda” del editor de PUMMA_08+

Este menú contiene invocaciones a ventanas de testificación autoral y versiones, acerca del compilador MNIBAS8A, el ensamblador ENS08 y el programa manejador PUMMA_08+.

Además de lo anterior contiene el acceso a una descripción básica de la funcionalidad y particularidades del ensamblador ENS08.

Este menú cuenta con los siguientes submenús:

- Acerca de MINIBAS8A
- Acerca de ENS08
- Acerca de PUMMA_08+
- Instrucciones básicas de ENS08

Al invocarse cada una de las primeras tres opciones, se despliegan sendas ventanas testificadoras como podrían ser las mostradas en las figuras 3.49, 3.50 y 3.51.

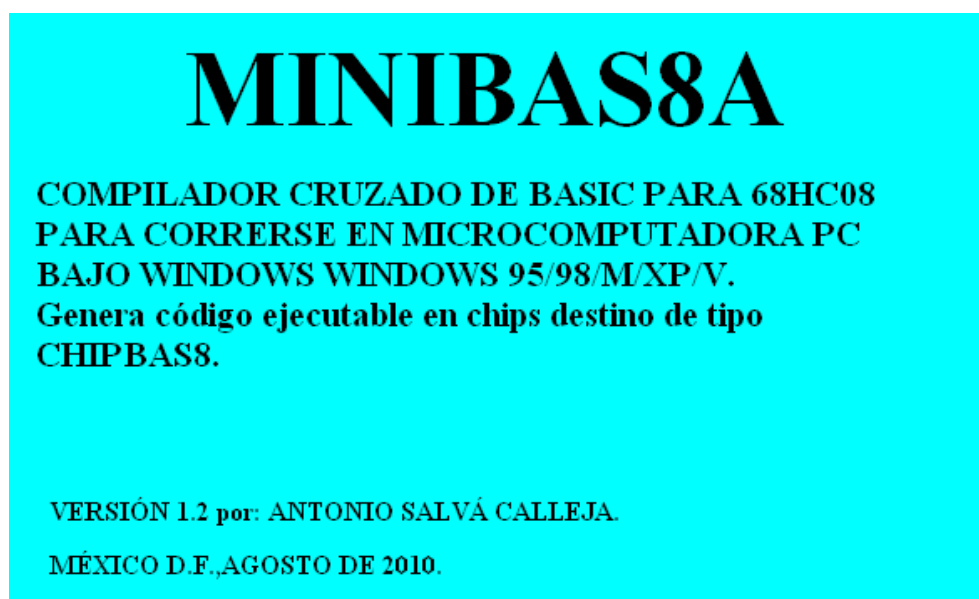


Figura 3.49. Testificación de autoría y versión del compilador MINIBAS8A.

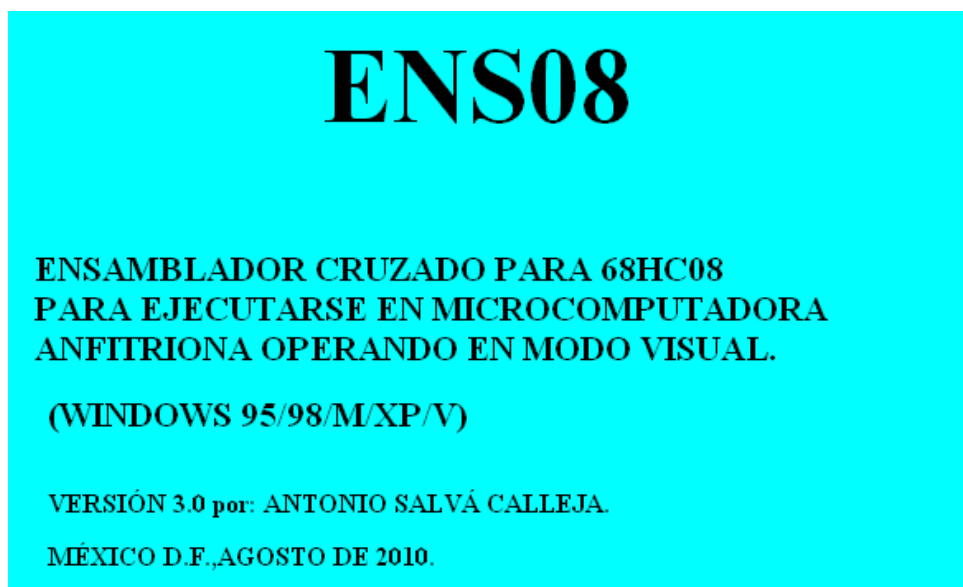


Figura 3.50. Testificación de autoría y versión del ensamblador ENS08.

PUMMA_08+

Programa manejador interactivo de microcontroladores de la familia MC(68HC)(9)(S)08XX, que contengan el firmware interlocutor NBCP8_BIBAS8. Incluye al ensamblador cruzado ENS08. Contiene al compilador cruzado de BASIC MINIBAS8A.

(WINDOWS 95/98/M/XP/V)

VERSIÓN 1.5 por: ANTONIO SALVÁ CALLEJA.

MÉXICO D.F., SEPTIEMBRE DE 2010.

Figura 3.51. Testificación de autoría y versión del manejador PUMMA_08+.

Si se invoca la opción “Instrucciones básicas de ENS08”, aparecerá una ventana donde se describen diversas funcionalidades y particularidades del ensamblador ENS08; véase la figura 3.52. Estos aspectos se describen también más adelante. Véase el capítulo cinco de este manual.

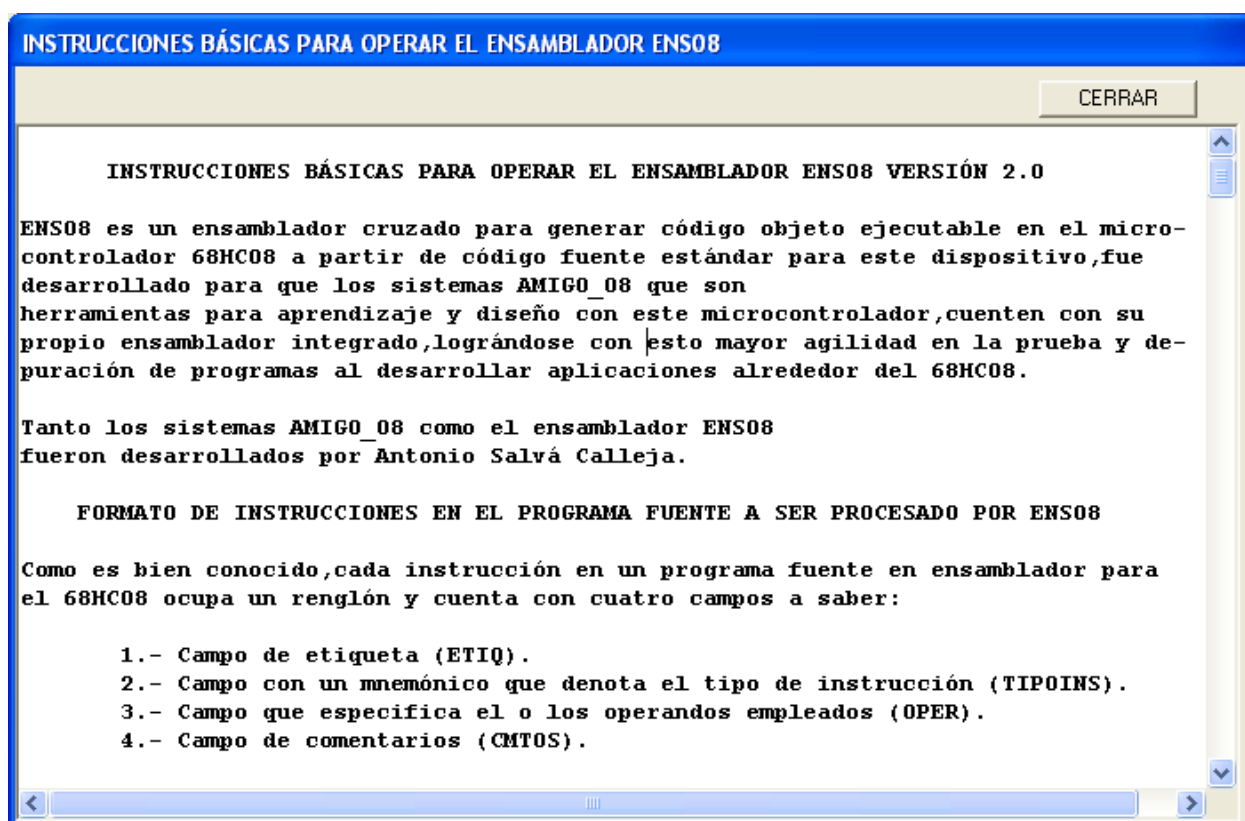


Figura 3.52. Ventana donde se muestran aspectos acerca de la funcionalidad y particularidades del ensamblador ENS08.

4 INSTALACIÓN E INICIALIZACIÓN DE PUMMA_08+

4.1 Requerimientos de PUMMA_08+

Estos son los requerimientos mínimos que PUMMA_08+ necesita para su apropiada instalación y ejecución:

- Microcontrolador deseable: Pentium® en adelante.
- Sistema operativo Microsoft® Windows® 95, Windows 98, Windows 98 Second Edition, Windows Me, Windows 2000 Professional, Windows XP Professional o Home Edition, Windows Vista.
- 64 (MB) de RAM [recomendados 128 (MB)]
- 10 (MB) de espacio disponible en su unidad de disco duro.
- Al menos un puerto serial disponible con conexión DB9 macho. En caso de no disponer con un puerto serial físico, es posible utilizar un adaptador USB – SER, como podría ser el denominado como “USB – SERIAL ADAPTOR” distribuido por la empresa STEREN.

4.2 Instalando PUMMA_08+

Para instalar PUMMA_08+, desde el CD de distribución ejecutar el archivo setup.exe, habiendo cerrado previamente la o las aplicaciones que se encuentren activas. La instalación sigue una secuencia de pasos comunes a accionamientos de este tipo para el sistema operativo Windows. A continuación se describen los pasos más relevantes.

Al iniciar la instalación aparecerá un dialogo como el mostrado en la figura 4.1.

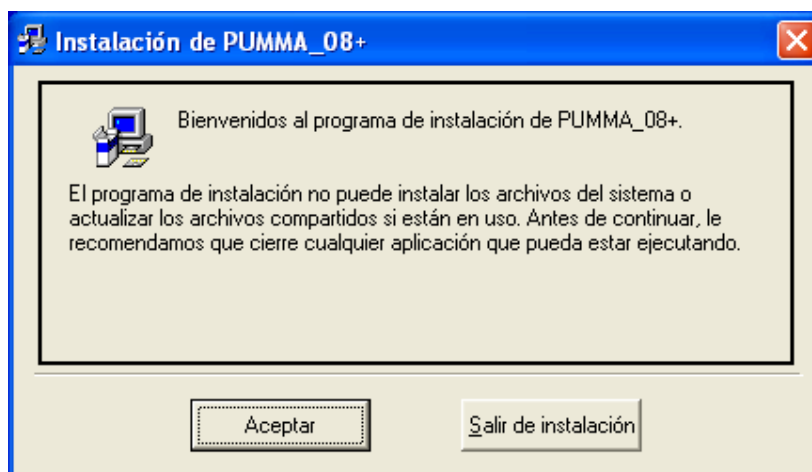



Figura 4.1 Dialogo presentado al usuario al iniciarse la instalación de PUMMA_08+.

Una vez que el usuario ha oprimido el botón “Aceptar” en el dialogo de la figura 4.1 el software de instalación desplegará el dialogo mostrado en la figura 4.2. Ahí se da opción al usuario para cambiar la carpeta de instalación del programa.



Si el usuario oprime el botón  sin haber utilizado previamente la opción “Cambiar directorio”, se iniciará la instalación; siendo “C:\Archivos de programa\PUMMA_08+” la carpeta de trabajo del programa en el sistema.

Cabe señalar aquí que es altamente recomendable utilizar la opción “Cambiar directorio”, para ubicar el directorio destino de instalación, cuyo nombre define el usuario, en la raíz de

alguna de las unidades presentes en el disco duro del sistema, como podría ser la “C”. Para el caso de **WINDOWS VISTA**, es forzoso el cambio de directorio de instalación a la raíz de alguna unidad presente en el disco duro del sistema, ya que se ha observado que para esta versión de **WINDOWS**, si no se hace el cambio mencionado, **PUMMA_08+** no funcionará correctamente.

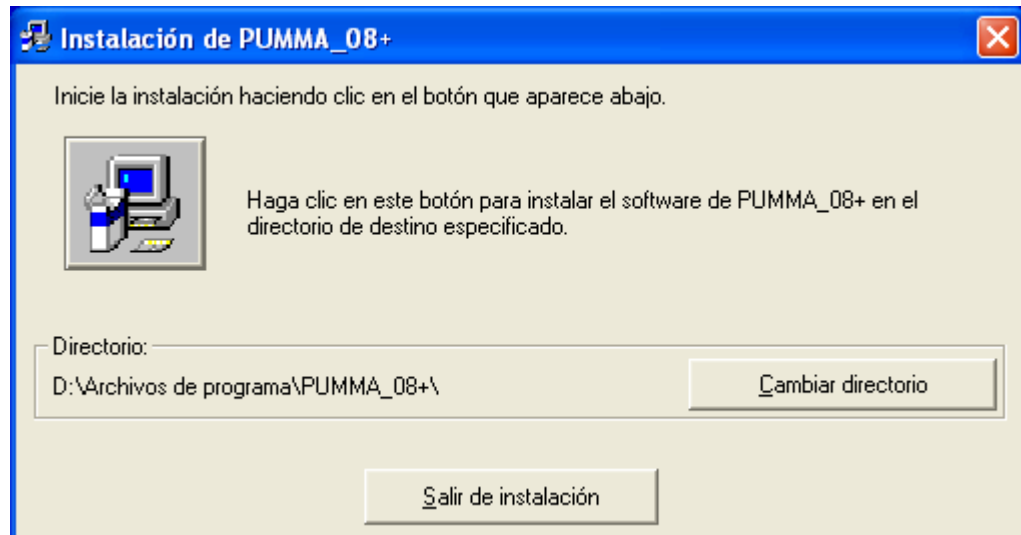


Figura 4.2. Dialogo para iniciar la instalación de PUMMA_08+.

Para cambiar la carpeta de trabajo simplemente se oprime el botón “Cambiar directorio”, luego de lo cual el instalador presenta un dialogo como el mostrado en la figura 4.3, donde el usuario ha definido que la carpeta de instalación sea “c:\p8pv12”.

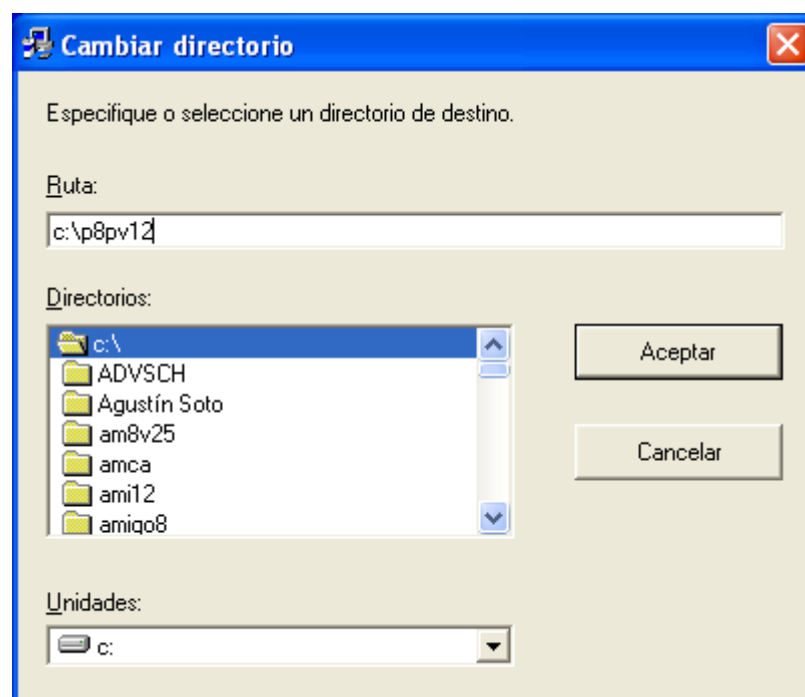


Figura 4.3. Dialogo donde el usuario ha definido que el directorio de trabajo sea “c:\p8pv12”.

Una vez que el usuario oprime el botón “Aceptar” en el dialogo de la figura 4.3, en caso de que el directorio no exista, el instalador pregunta al usuario si desea crearlo, a lo cual el usuario



deberá responder que sí. Después de esto, se deberá oprimir el botón lo que hace que se inicie el proceso de instalación.

Al iniciarse la instalación, se determina si hay suficiente espacio en el disco duro antes de proceder. Existen casos en los que en el reporte de “*Espacio disponible*” en disco duro, se despliega un número negativo. Este error se debe a una incompatibilidad entre la versión del instalador de PUMMA_08+, en cuanto a el tamaño de las variables enteras usadas para denotar la capacidad del disco duro del sistema, de acuerdo con cánones actuales. La figura 4.4 ejemplifica este caso. En caso de que al momento de instalar PUMMA_08+, se presente este error, éste se debe ignorar y continuar con la instalación pulsando para ello al botón “Instalar ahora”.

No hay suficiente espacio en disco en una o varias unidades.

Unidad	Espacio requerido	Espacio disponible	Espacio necesario
C:	614 K	-1471704 K	1472318 K

Figura 4.4 Indicación falsa de error por espacio insuficiente por parte del instalador.

4.3 Inicialización de PUMMA_08+

Una vez que ha concluido la instalación se recomienda proceder a la configuración del puerto serie a emplear y baudaje. Cabe señalar que, si en la TD el MCU es el 68HC908GP32 caracterizado como Chipbas8 GP32, el baudaje requerido es 9600 bps.

Los parámetros mencionados en el párrafo anterior, son solicitados al usuario la primera vez que se ejecuta PUMMA_08+; una vez que éste los define, estos quedan predeterminados y no se le vuelven a pedir al usuario, a menos que éste use la facilidad presente en PUMMA_08+ para cambiarlos.

Es importante señalar que para computadoras que no contengan puerto serie físico, para poder trabajar con PUMMA_08+ se requerirá un adaptador USB-SERIE, para el cual previamente ya se debe haber instalado el driver correspondiente y haber tomado nota del número de puerto serie asociado con dicho adaptador. Esta información puede verse empleando la facilidad de WINDOWS “Administrador de hardware”.

En caso de que se use el adaptador USB-SERIE, éste deberá estar conectado siempre que se ejecute PUMMA_08+, de lo contrario, se puede presentar un funcionamiento errático del programa, que en ocasiones podrá hacer requerir su desinstalación e instalación renovada.

Para más detalles acerca de la inicialización de PUMMA_08+, véase el punto **“Inicialización del sistema AIDA08 para uso de éste en una PC determinada”**, presente en el capítulo 2 de este manual.

5 ASPECTOS ACERCA DEL ENSAMBLADOR ENS08 PRESENTE EN PUMMA_08+

ENS08 es un ensamblador cruzado para generar código objeto ejecutable en microcontroladores de la familia HC08 de Freescale, a partir de código fuente estándar para estos dispositivos, fue desarrollado para que los manejadores AMIGO_08, PUMMA_08 y PUMMA_08+ que son herramientas para aprendizaje y diseño con estos microcontroladores; cuenten con su propio ensamblador integrado, lográndose con esto mayor agilidad en la prueba y depuración de programas al desarrollar aplicaciones alrededor de la familia de microcontroladores aquí mencionada.

5.1 Formato de instrucciones en el programa fuente a ser procesado por ENS08

Como es bien conocido, cada instrucción en un programa fuente en ensamblador para el 68HC08 ocupa un renglón y cuenta con cuatro campos a saber:

- 1.- Campo de etiqueta (ETIQ).
- 2.- Campo con un mnemónico que denota el tipo de instrucción (TIPOINS).
- 3.- Campo que especifica el o los operandos empleados (OPER).
- 4.- Campo de comentarios (CMTOS).

De esta manera la forma genérica para una instrucción es:

ETIQ TIPOINS OPER CMTOS

La etiqueta ETIQ es una cadena de un máximo de veinte caracteres, debiendo el primero estar en la primera columna y ser letra forzosamente, si el usuario lo desea podrá usar ":" como último carácter en una etiqueta, de esta forma, para ENS08 las etiquetas "INICIO" e "INICIO:" serían lo mismo; cabe señalar que entre la etiqueta y el campo que denota el tipo de instrucción debe haber cuando menos un espacio, esto aún cuando el último carácter de la etiqueta sea ":"; así, la instrucción:

OTRO:LDA \$03,X

genera un error al ser procesada por ENS08, para corregir esto la misma podría ponerse:

OTRO: LDA \$03,X

o bien:

OTRO LDA \$03,X

En el campo TIPOINS se especificará el mnemónico asociado con la instrucción de que se trate, deberá haber cuando menos un espacio entre éste y el campo de operandos; además para ENS08 cualquier operando que se especifique como un número debe estar expresado en hexadecimal con el carácter "\$" precediéndolo; así, la instrucción:

LDA 0A,X

generará un error de sintaxis al ser procesada por ENS08, para corregir el error la misma debe ponerse como:

LDA \$0A,X

Cabe señalar aquí, que cuando en el campo de operandos (OPER) se especifiquen números de un byte, los mismos deberán escribirse siempre con dos dígitos, en el caso de números de dos bytes, deberán emplearse siempre cuatro dígitos para denotarlos, utilizando para ambos casos notación hexadecimal.

Para especificar que se desea, para un caso dado, emplear el modo de direccionamiento inmediato se deberá, como es bien sabido, preceder el operando por el caracter "#".

En el campo OPER se especifica la forma en que la instrucción accesa el operando único, o bien uno de los operandos que la misma pudiera emplear, esto se deberá hacer siguiendo la sintaxis estándar asociada con el modo de direccionamiento, que se requiera emplear para la instrucción.

Para separar los campos de operandos (OPER) y de comentarios (CMTOS) se debe usar el delimitador ";" para especificar el inicio del campo de comentarios, debiendo haber cuando menos un espacio entre el final del campo de operandos y el delimitador ";" empleado para marcar el inicio de un comentario.

ENS08 ignorará cualquier texto que se encuentre a la derecha del punto y coma; si se requiere colocar un comentario que ocupe un renglón completo, se puede hacer esto ya sea colocando el caracter ";" o bien el caracter "*" en la primera columna del renglón donde se pondrá el comentario.

5.2 Pseudo-operaciones soportadas por la versión 2.5 de ENS08

ENS08 V2.5 soporta las pseudo-operaciones siguientes:

5.2.1 Pseudo-operación EQU

Esta pseudo-operación permite asignar un valor numérico expresado en hexadecimal, a una cadena de caracteres definida por el usuario, la sintaxis asociada es la siguiente:

NOMBRE EQU valnum

Donde la cadena "NOMBRE" debe ser definida por el usuario, debiendo estar el primer caracter de la misma en la primera columna. Por otro lado la cadena "valnum" representará un valor numerico explícito frecuentemente expresado en hexadecimal.

Por ejemplo, supóngase que se desea asignar a la cadena PUERTO_A el valor hexadecimal 0000, la declaración correspondiente sería:

PORT_A EQU \$0000

Por lo regular todas las asignaciones numéricas asociadas con cadenas de fácil asimilación para el programador, se colocan al inicio de un programa fuente empleando para ello sendas pseudo-operaciones EQU.

5.2.2 Pseudo-operación ORG

Esta pseudoinstrucción se emplea para especificar la dirección de memoria, donde se desea colocar el código objeto correspondiente al código fuente inmediato posterior a la misma; la sintaxis correspondiente es:

ORG INICIO

Donde la cadena "INICIO" podrá ser un número expresado en notación hexadecimal, o bien una cadena a la cual se le haya previamente asignado un valor hexadecimal, empleando para ello una pseudo-operación EQU.

Por ejemplo, si se desea que el código objeto correspondiente al siguiente código fuente:

```
LDA #$4A
STA $0000
JMP $FC00
```

sea colocado a partir de la dirección \$0100; para ello se ha de emplear la directiva ORG aquí descrita, dos formas correctas para lograr esto serían las siguientes:

```
ORG $0100
LDA #$4A
STA $0000
JMP $FC00
```

o bien:

ARRANQUE EQU \$0100

```
ORG ARRANQUE
LDA #$4A
STA $0000
JMP $FC00
```

5.2.3 Pseudo-operación FCB

Esta pseudoinstrucción se usa cuando se desea que a partir de una dirección sean colocados bytes, que representen datos a emplear por el programa que se esté desarrollando en un momento dado; la sintaxis correspondiente es:

DATOS FCB *lista de bytes*

Donde la cadena "lista de bytes" representa la serie de bytes que se desea queden colocados a partir de la dirección que corresponda a la etiqueta "DATOS", los mismos deben especificarse en hexadecimal y estar separados por comas; cabe señalar que si la lista de bytes es larga, pueden usarse varios renglones con pseudo-operaciones FCB, requiriéndose etiqueta indicativa únicamente en el primero.

Por ejemplo, supóngase que se requiere contar con una tabla con los valores ascii de los dígitos del cero al nueve, si a la etiqueta indicativa se le denomina

ASC_09 una forma para declarar esto sería:

ASC_09 FCB \$30,\$31,\$32,\$33,\$34,\$35,\$36,\$37,\$38,\$39

otra alternativa que logra el mismo fin podría ser:

```
ASC_09 FCB $30,$31,$32,$33
        FCB $34,$35,$36
        FCB $37,$38,$39
```

Cabe señalar que los componentes separados por comas en la cadena "lista de bytes", podrían ser bytes con el prefijo "\$", que se hayan predefinido mediante pseudo-operaciones "equ".

5.2.4 Pseudo-operación DB

Esta pseudo-operación es un alias de la anterior (FCB).

5.2.5 Pseudo-operación FDB

Esta pseudoinstrucción se usa cuando se desea que a partir de una dirección sean colocados palabras de dos bytes, que representen datos a emplear por el programa que se esté desarrollando en un momento dado; la sintaxis correspondiente es:

```
DATOSW FCB lista de palabras
```

Donde la cadena "lista de palabras" representa la serie de pares de bytes que se desea queden colocados a partir de la dirección que corresponda a la etiqueta "DATOSW", los mismos deben especificarse en hexadecimal y estar separados por comas; cabe señalar que si la lista de palabras es larga, pueden usarse varios renglones con pseudo-operaciones FDB, requiriéndose etiqueta indicativa únicamente en el primero.

Por ejemplo, supóngase que se requiere contar con una tabla con los valores ascii de los dígitos del cero al nueve, si a la etiqueta indicativa se le denomina ASC_W una forma para declarar esto sería:

```
ASC_W FDB $3031, $3233, $3435, $3637, $3839
```

otra alternativa que logra el mismo fin podría ser:

```
ASC_W  FDB $3031,$3233
        FDB $3435,$3637
        FDB $3839
```

Cabe señalar que los componentes separados por comas en la cadena "lista de palabras", podrían ser etiquetas del programa o bien palabras de dos bytes con el prefijo "\$", que se hayan predefinido mediante pseudo-operaciones "equ".

5.2.6 Pseudo-operación DW

Esta pseudo-operación es un alias de la anterior (FDB).

5.2.7 Pseudo-operación FCC

Esta pseudoinstrucción se emplea, cuando se desea que a partir de una dirección, queden colocados los bytes que representen los códigos ascii correspondientes a un texto; la sintaxis asociada es la siguiente:

```
TEXTO FCC  "TEXTO A COLOCAR"
```

donde "TEXTO" es la denominación de la etiqueta a partir de la cual se desea queden colocados los códigos ascii asociados con cada uno de los caracteres que integran la cadena que representa

el texto a colocar, es importante hacer notar que el texto debe estar delimitado por comillas como se muestra en la sintaxis genérica.

Por ejemplo, supóngase que se desea que los códigos ascii correspondientes al texto "Este es un demo de como poner una leyenda usada en un programa", queden colocados a partir de la dirección correspondiente con la etiqueta "MI_TEXTO"; al igual que en el caso de la pseudo-operación FCB, pueden emplearse uno o varios renglones para hacer la declaración correspondiente, de este modo, para el texto de este ejemplo la sintaxis podría ser:

```
MI_TEXTO FCC "Este es un demo de como poner una leyenda usada en un programa"
```

o bien:

```
MI_TEXTO   FCC "Este es un demo de como"  
           FCC " poner una leyenda usada "  
           FCC "en un programa"
```

5.2.8 Pseudo_operaciones para reservar bytes y palabras

ENS08 V2.5 maneja pseudo-operaciones para reservar en memoria bytes o palabras de 16 bits, la sintaxis para reservar bytes es la siguiente:

```
rmb n
```

Donde n deberá ser un número en decimal que denota el número de bytes a reservar.

En lo que toca a la reserva de palabras de 16 bits, la sintaxis es la siguiente:

```
rmw n
```

Donde n debera ser un número en decimal que denota el número de palabras a reservar.

5.2.9 Directiva *include*

ENS08 V2.5 soporta la directiva INCLUDE. La sintaxis para ésta es:

```
$INCLUDE "archivoinc"
```

Donde archivoinc es una cadena que especifica el nombre de un archivo fuente que contiene código que se desea quede incluido para fines de la generación del código objeto correspondiente, si no se determina trayectoria (path) se usará la asociada con el subdirectorío (carpeta) activa al momento de efectuar el ensamblado, para evitar confusiones se recomienda al usuario especificar la trayectoria donde esté el archivo a incluir.

Preferentemente el caracter "\$" deberá estar en la primera columna de la pantalla. A continuación se describe un ejemplo de uso de esta directiva.

Supóngase que el contenido del archivo equ1.asm es el siguiente:

```
porta equ $0000  
ddra  equ $0004  
dato  equ $1a  
inicio equ $0100  
pum   equ $FC00
```

y que el mismo se encuentra en la carpeta "c:\mihc08\"; además se tiene el archivo prog.asm cuyo contenido es:
\$include "c:\mihc08\equ1.asm"

```
org inicio
```

```
lda #$ff  
sta ddra  
lda #dato  
sta porta  
jmp pum
```

al ensamblar el programa fuente contenido en el archivo prog.asm, ENS08 genera un archivo prog.tot, en donde se han incluido el contenido de los archivos asociados en todas las sentencias include contenidas en el archivo prog.asm; así, para el ejemplo aquí comentado el archivo prog.tot contendría lo siguiente:

```
* C:\MIHC11\EQU1.ASM
```

```
PORTA EQU $0000  
DDRA EQU $0004  
DATO EQU $1A  
INICIO EQU $0100  
PUM EQU $FC00
```

```
**** FIN DE C:\MIHC11\EQU1.ASM
```

```
ORG INICIO
```

```
LDA #$FF  
STA DDRA  
LDA #DATO  
STA PORTA  
JMP PUM
```

una vez que ENS08 ha generado el archivo prog.tot se efectúa el ensamblado del mismo generándose los archivos prog.s19 y prog.lst convencionales.

Cabe señalar aquí, que los pasos antes mencionados se realizan sin la intervención del usuario, una vez que el mismo invoca un comando que implique ensamblar un programa que contenga sentencias *include*.

Para hacer que una sentencia *include* no se tome en cuenta, se deberá anteponer la misma con el caracter "*", debiendo éste quedar en la primera columna. Así, para eliminar la sentencia *include* en el programa del ejemplo anterior, la sintaxis sería:

```
*$include "c:\miHC08\equ1.asm"
```

5.3 Manejo de aritmética con etiquetas y cadenas definidas por pseudoinstrucciones EQU

La versión 2.5 de ENS08 maneja operaciones aritméticas de suma y resta con las etiquetas y cadenas predefinidas por pseudos instrucciones “EQU”. Debiendo ser el operando izquierdo una etiqueta; o bien, una cadena predefinida en alguna sentencia “EQU”, y el otro un número expresado en decimal. Por ejemplo, después de ejecutarse el siguiente código fuente:

```
bydat equ $F9
      org $0100
inicio: lda #bydat+1
      sta $0180
      lda datos+1
      ldhx #inicio-2
fin:   bra fin

datos: fcb $30,$31,$32
```

en la localidad con dirección \$0180 quedaría almacenado el byte \$FA, el acumulador A quedaría cargado con el byte \$31 y el registro H:X contendría la palabra \$00FE.

Es importante señalar, que entre el operador “+” ó “-” implicado, no debe haber espacios respecto a los dos operadores. En otro caso ENS08 reportaría un error de sintaxis.

6 PROGRAMAS EJEMPLO EN LENGUAJE ENSAMBLADOR

En este capítulo se muestran programas, que ilustran aspectos diversos acerca de la programación y funcionamiento de microcontroladores de la familia HC08. Además, por medio de éstos, se ejemplifica el uso de diversas facilidades propias del manejador PUMMA_08+ y de la tarjeta MINICON_08A.

6.1 Estructura de un programa de aplicación

El software asociado con una determinada aplicación de Control y/o Instrumentación basada en un microcontrolador, frecuentemente estará integrado por los siguientes bloques funcionales:

1. **Bloque de inicialización.** El cual contendrá código donde los diversos periféricos usados en la aplicación son configurados.
2. **Bloque de accionamiento principal.** El cual por lo regular será un lazo que contendrá código que arbitrará hardware de interfaz propio de la aplicación.
3. **Bloque de subrutinas.** El cual podrá estar integrado por una o varias subrutinas, que por lo regular son llamadas desde el bloque principal, desde otras subrutinas contenidas en este mismo bloque, o bien, desde rutinas de servicio de interrupción.
4. **Bloque de rutinas de servicio de interrupción.** El cual estará integrado por las diversas rutinas que han de invocarse por el MCU, al darse cada uno de los eventos de hardware, propios de cada una de las instancias de interrupción que use la aplicación.
5. **Bloque de datos.** El cual contendrá declaraciones que validan la presencia de datos que pudieran ser requeridos por la aplicación o son necesarios para el funcionamiento del programa asociado. Ejemplos de contenidos en este bloque podrían ser entre otros: Listas de bytes cuyo valor denota el código ASCII asociado con textos que a desplegar; o bien, declaraciones para colocar vectores asociados con las instancias de interrupción que use la aplicación.

El orden en que se presentan los bloques, es en la mayoría de los casos, el enumerado anteriormente. Además pueden existir aplicaciones que no contengan todos los bloques aquí descritos.

Antes de pasar a la descripción de programas didácticos en ensamblador, se describe a continuación una subrutina de retardo; la cual será empleada en varios de los ejemplos ilustrativos presentados en este manual.

6.2 Diseño de una subrutina de retardo

Al desarrollar aplicaciones basadas en microcontroladores, es frecuente la necesidad de contar con rutinas que generen un tiempo de espera configurable, esto se puede lograr simplemente haciendo que el procesador ejecute un conjunto de instrucciones un número determinado de veces. A continuación se muestra una rutina de retardo, donde la cuenta del número de veces que se va ejecutar el lazo propio de ésta, se lleva en los registros “H” y “X” del MCU. El número de veces que ha de ejecutarse el lazo se precarga en el par H:X al inicio de la subrutina. A continuación se muestra ésta:

RET:	PSHH	(2)
	PSHX	(2)
	LDHX #\$XXXX	(3)
VUELTA:	NOP	(1)
	NOP	(1)
	AIX #\$FF	(2)
	CPHX #\$0000	(3)
	BNE VUELTA	(3)
	PULX	(2)
	PULH	(2)
	RTS	(4)

Donde XXXX deberá expresarse en notación hexadecimal, y representa el valor del número de veces que se va a ejecutar el lazo que inicia en la instrucción con la etiqueta “VUELTA” y que además está resaltado en negritas. Para fines ilustrativos, a la derecha de cada instrucción se ha colocado el número de ciclos del *reloj de bus* que se lleva la ejecución de éstas.

Sabiendo que la frecuencia del reloj de bus para microcontroladores HC08, cuya señal básica de reloj se genere a partir de un cristal conectado a los pines OSC1 y OSC2, es la cuarta parte de la frecuencia propia del cristal ahí conectado, se infiere que: **Para la tarjeta MINICON_08A, la frecuencia del reloj de bus es 2 MHz, y el periodo de la señal de reloj de bus es 0.5 µs. Ya que la frecuencia asociada al cristal usado en ésta es 8 MHz.**

De esta forma, el tiempo de espera “Tr”, generado al invocarse la subrutina “RET”, mediante una instrucción “JSR” empleando el modo de direccionamiento extendido será:

$$Tr = (20 + 10Xr)Tbus \quad (6.1)$$

Donde: Xr representa el número de veces que ha de ejecutarse el lazo implicado y Tbus representa el periodo del reloj de bus.

Considerando los parámetros de frecuencia de la tarjeta MINICON_08A, se deduce fácilmente a partir de la ecuación 6.1 que, los valores mínimo y máximo para el retardo generado al invocarse la rutina “RET” son respectivamente 15 microsegundos y 327.69 milisegundos.

Despejando Xr en la ecuación 6.1 se obtiene:

$$Xr = \frac{(Tr/Tbus) - 20}{10} \quad (6.2)$$

Considerando además, que el valor del número de vueltas debe ser un entero, si éste se denota como Xre, se tendrá:

$$Xre = I\left\{\frac{(Tr/Tbus) - 20}{10}\right\} \quad (6.3)$$

Donde la notación $I\{.\}$, representa la parte entera del argumento implicado.

Por ejemplo, supóngase que se desea que la subrutina “RET” genere un retardo de 1.933 mS, cuando $Tbus = 0.5 \mu s$. A partir de la ecuación 6.2 se obtiene que Xr debe ser 384.6. Por lo tanto, Xre debe ser 384; lo que hace que el valor explícito para la cadena genérica XXXX mostrada en la rutina RET deberá ser 0180.

Nótese que al tomarse la parte entera de X_r para obtener X_{re} , se tendrá que el retardo generado será un poco menor que el especificado (1.930 mS). Sin embargo, para muchos casos prácticos, el error debido al truncamiento de la parte fraccionaria de X_r es tolerable. Además, puede verse que éste disminuye a medida que aumenta el tiempo de espera que se desee generar en un momento dado.

Debido a que el número de ciclos implicado en la ejecución de la instrucción “JSR” del MCU, varía para los diversos modos de direccionamiento que ésta contempla. En caso de que para invocar la subrutina “RET”, se use algún otro modo de direccionamiento, las expresiones para el cálculo del retardo implicado y el valor de X_{re} , serán un poco diferentes respecto a lo expresado en las ecuaciones 6.1 y 6.3. Lo mismo aplica si la rutina se invoca con la instrucción “BSR”. En la tabla 6.1 se muestran las expresiones para el retardo y el valor de X_{re} para los diversos modos de direccionamiento asociados con la instrucción “JSR” y la invocación hecha con la instrucción “BSR”.

TABLA 6.1. EXPRESIONES PARA T_r y X_{re} ASOCIADAS CON LA INVOCACIÓN DE LA SUBROUTINA RET CON LAS INSTRUCCIONES “JSR” Y “BSR”

TIPO DE INSTRUCCIÓN DE SALTO A SUBROUTINA	MODO DE DIRECCIONAMIENTO	T_r	X_{re}
JSR	DIRECTO	$T_r = (19 + 10X_r)T_{bus}$	$X_{re} = I\left\{\frac{(T_r/T_{bus}) - 19}{10}\right\}$
JSR	EXTENDIDO	$T_r = (20 + 10X_r)T_{bus}$	$X_{re} = I\left\{\frac{(T_r/T_{bus}) - 20}{10}\right\}$
JSR	IX2	$T_r = (21 + 10X_r)T_{bus}$	$X_{re} = I\left\{\frac{(T_r/T_{bus}) - 21}{10}\right\}$
JSR	IX1	$T_r = (20 + 10X_r)T_{bus}$	$X_{re} = I\left\{\frac{(T_r/T_{bus}) - 20}{10}\right\}$
JSR	IX	$T_r = (19 + 10X_r)T_{bus}$	$X_{re} = I\left\{\frac{(T_r/T_{bus}) - 19}{10}\right\}$
JSR	RELATIVO	$T_r = (19 + 10X_r)T_{bus}$	$X_{re} = I\left\{\frac{(T_r/T_{bus}) - 19}{10}\right\}$

Detalles acerca de los modos de direccionamiento implicados en la tabla 6.1 pueden verse en la sección 10 de la referencia [1].

La subrutina de retardo aquí explicada será usada en varios de los ejemplos ilustrativos presentados en este capítulo.

Ejemplo1. Contador ascendente ejecutable desde RAM

En este ejemplo se muestra un programa que al ejecutarse despliega en el puerto A del MCU un contador binario ascendente con una cadencia de 0.25 S entre cuentas. Al ejecutarse el código asociado en la tarjeta MINICON_08A, la cuenta será visible en los LEDS conectados al puerto A, siempre y cuando el *jumper sencillo* J10 esté colocado. Véase la tabla 2.4 presente en el capítulo 2 de este manual.

Para generar el tiempo de espera entre cuentas se usará la rutina RET explicada anteriormente. Dado que el retardo requerido es 0.25 S, a partir de la ecuación 6.3 se obtiene que X_{re} debe ser 49998, que en notación hexadecimal es C34E.

El programa asociado con este ejemplo se muestra a continuación:

```
* EJEMPLO 1
* CONTADOR ASCENDENTE DESPLEGADO EN EL PUERTO A
* Por: ANTONIO SALVÁ CALLEJA
* Octubre de 2008
* ALMACENADO EN EL ARCHIVO CONTPTA.ASM

PTA      EQU    $00
DDRA     EQU    $04

          ORG    $0100    ;EL PROGRAMA SE ORIGINA EN LA LOCALIDAD $0100 DE RAM

* BLOQUE DE INICIALIZACIÓN
          MOV    #$FF,DDRA ;CONFIGURA PUERTO A COMO SALIDA
          CLRA
* FIN DE BLOQUE DE INICIALIZACIÓN

* BLOQUE DE ACCIONAMIENTO PRINCIPAL
CONT:     STA    PTA
          INCA
          JSR    RET250M
          BRA    CONT
* FIN DE BLOQUE DE ACCIONAMIENTO PRINCIPAL

* BLOQUE DE SUBROUTINAS
RET250M:   PSHH
          PSHX
          LDHX   #$C34E
VUELTA:   NOP
          NOP
          AIX    #$FF
          CPHX   #$0000
          BNE    VUELTA
          PULX
          PULH
          RTS
* FIN DE BLOQUE DE SUBROUTINAS
```

Nótese que la cuenta se lleva en el acumulador A del CPU. Este programa se introdujo en el editor de PUMMA_08+ y se guardó bajo el nombre “**CONTPTA.ASM**”.

Para ejecutarlo basta que al estar éste presente en la ventana del editor, se pulse el botón “e-ram” contenido en ésta.

Ejemplo2. Contador descendente ejecutable desde FLASH

En este ejemplo se muestra un programa que al ejecutarse despliega en el puerto A del MCU un contador binario descendente con una cadencia de 0.5 S entre cuentas. Al ejecutarse el código asociado en la tarjeta MINICON_08A, la cuenta será visible en los LEDS conectados al puerto A, siempre y cuando el *jumper sencillo* J10 esté colocado. Véase la tabla 2.4 presente en el capítulo 2 de este manual.

Para generar el tiempo de espera entre cuentas se usará la rutina RET explicada anteriormente. Dado que el retardo requerido es 0.5 S, y éste es mayor que el retardo máximo que se puede generar con la subrutina de espera aquí empleada. Para generar el tiempo requerido se invocará dos veces una subrutina de retardo configurada para generar un tiempo de espera de 0.25 mS igual a la empleada en el ejemplo 1.

El programa asociado con este ejemplo se muestra a continuación:

```
* EJEMPLO 2
* CONTADOR DESCENDENTE DESPLEGADO EN EL PUERTO A
* Por: ANTONIO SALVÁ CALLEJA
* Octubre de 2008
* ALMACENADO EN EL ARCHIVO CONTPTAF.ASM

PTA      EQU    $00
DDRA     EQU    $04

          ORG    $8000 ;EL PROGRAMA SE ORIGINA EN LA LOCALIDAD $8000 DE
;          MEMORY (FLASH)

* BLOQUE DE INICIALIZACIÓN
          MOV    #$FF,DDRA ;CONFIGURA PUERTO A COMO SALIDA
          CLRA
* FIN DE BLOQUE DE INICIALIZACIÓN

* BLOQUE DE ACCIONAMIENTO PRINCIPAL
CONT:     STA    PTA
          DECA
          BSR    RET250M
          BSR    RET250M
          BRA    CONT
* FIN DE BLOQUE DE ACCIONAMIENTO PRINCIPAL

* BLOQUE DE SUBROUTINAS
RET250M:   PSHH
          PSHX
          LDHX   #$C34E
VUELTA:   NOP
          NOP
          AIX    #$FF
          CPHX   #$0000
          BNE    VUELTA
          PULX
          PULH
          RTS
* FIN DE BLOQUE DE SUBROUTINAS
```

Nótese que la cuenta se lleva en el acumulador A del CPU, y que la invocación de la subrutina de retardo se hace con la instrucción BSR. Esto hace que el retardo sea un poco menor al calculado, de hecho éste es 0.5 μ S más chico para cada invocación de la subrutina RET250M. Por lo tanto, la cadencia de la cuenta será 1.0 μ S más pequeña que la que se tendría si se invoca la rutina de retardo usando la instrucción JSR. Claramente esta diferencia es despreciable. Este programa se introdujo en el editor de PUMMA_08+ y se guardó bajo el nombre “CONTPTAF.ASM”. Para ejecutarlo basta que al estar éste presente en la ventana del editor, se pulse el botón “e-feep” contenido en ésta.

6.3 Ejecución autónoma de programas en la TD cuando ésta opera en los modos B o C

Cuando la TD opera en los modos B o C, para ejecutar de manera autónoma código previamente grabado en la memoria FLASH, el programa deberá contener, además de las instrucciones y declaraciones propias de la aplicación, líneas adicionales de código como a continuación se describe:

1. Si el código a ejecutar no hace uso del sistema COP (Watch dog) del MCU, éste deberá ser deshabilitado al inicio del programa. Para esto podrá usarse entre otras la siguiente secuencia de líneas de ensamblador:

```
CONFIG1    EQU    $1F

            BSET  0,CONFIG1
```

Para más detalles acerca del sistema COP del MCU puede verse la sección 9 de [1].

2. Al final del programa se deberán estar presentes declaraciones asociadas con la colocación del vector de RESET de usuario requerido. Esto podría tener la siguiente forma:

```
ORG $XXFE
DW  $8000
```

Donde se ha supuesto, para fines ilustrativos, que el programa a ejecutar de manera autónoma inicia en la dirección \$8000. En cuanto a la cadena XX, ésta deberá ser “FB” cuando la TD opere en modo B y “D7” cuando el modo de operación sea C. Para más detalles acerca del vector de usuario y sus direcciones de colocación, véase el tema 2.5 de este manual.

6.3.1 Preparación previa a ejecución autónoma

Una vez que el programa a ejecutar ha sido estructurado como se describe en los párrafos anteriores, y éste ha sido grabado en la memoria FLASH de la TD, **el puente asociado con el jumper excluyente 1 (JE1) deberá colocarse del lado derecho en éste,** véase el tema 2.5. Esto hará que el programa implicado se ejecute al energizarse la tarjeta, o bien, después de oprimir en ésta el botón de RESET.

6.3.2 Grabación de la memoria FLASH con el código para ejecución autónoma

Si el código para ejecución autónoma, se genera a partir de un programa en ensamblador presente en la ventana del editor; para colocar en la FLASH el programa, simplemente se oprime el botón “e-feep”, estando presente en la ventana de edición el programa implicado. Esto hará que el programa se grave en la memoria no volátil con la información propia del vector de RESET de usuario colocada en donde le corresponde. Además el programa será ejecutado apreciándose su funcionamiento.

En caso de que el programa se ensamble empleando otro ensamblador existente en el mercado; para grabar la memoria FLASH con el código ejecutable, se debe emplear la facilidad para grabar archivos S19 presente en el menú “Manejo de FEED” presente en la ventana de manejo hexadecimal, aplicado esto al archivo S19 que genere el ensamblador utilizado. Para detalles acerca de esta opción véase el subtema 3.5.4 en este manual.

6.4 Ejecución autónoma de programas en la TD cuando ésta opera en el modo A

Bajo esta modalidad se supone que el MCU presente en la tarjeta no contiene firmware en la FLASH; por lo tanto, para cargar el programa a ejecutar de manera autónoma, se requiere que esto se haga empleando una tarjeta de programación conectada mediante una interfaz de seis hilos al conector 7 de la TD. Dicha tarjeta podría ser la IP_ASC_08B. El software manejador tendría que ser el denominado como AMIGO_08. Para detalles acerca de la tarjeta de programación aquí mencionada y su software manejador, véase el tema 2.2 de este manual y las referencias bibliográficas ahí señaladas.

En cuanto a la estructura del programa a ejecutar de manera autónoma, es fácil ver que ésta sería similar a la descrita en el tema 6.3 con la salvedad de que las direcciones de colocación del vector de RESET asociado, tendrían que ser las propias del MCU para este fin; esto es, XX deberá ser FF. **Además, dado que se parte de un MCU que no contiene al monitor NBSP8, para fines de la ejecución autónoma no se requerirá mover el puente presente en JE1 de la posición izquierda a la derecha.** De hecho, para fines del proceso de grabación del programa el puente sí deberá estar colocado en la posición izquierda de JE1, pero para fines de la ejecución autónoma, el puente de JE1 podrá estar colocado en la posición izquierda o derecha, o bien no estar colocado.

A continuación se describe un ejemplo de ejecución autónoma de un programa en la TD cuando esta opera en el modo C. El programa a ejecutar es el mismo que el descrito en el ejemplo 1.

Ejemplo3. Contador ascendente ejecutable de manera autónoma

De acuerdo con lo explicado en el tema 6.3 el programa en ensamblador asociado podría ser el siguiente:

```
* EJEMPLO 3
* CONTADOR ASCENDENTE DESPLEGADO EN EL PUERTO A
* EJECUTABLE DE MANERA AUTÓNOMA
* Por: ANTONIO SALVÁ CALLEJA
* Octubre de 2008
* ALMACENADO EN EL ARCHIVO CONTPTAA.ASM
PTA      EQU    $00
DDRA     EQU    $04
CONFIG1  EQU    $1F

                ORG    $8000 ;EL PROGRAMA SE ORIGINA EN LA LOCALIDAD $8000 DE FLASH

* BLOQUE DE INICIALIZACIÓN
                BSET    0,CONFIG1 ;DESHABILITA COP
                MOV     #$FF,DDRA ;CONFIGURA PUERTO A COMO SALIDA
                CLRA
* FIN DE BLOQUE DE INICIALIZACIÓN

* BLOQUE DE ACCIONAMIENTO PRINCIPAL
CONT:        STA     PTA
                INCA
                JSR     RET250M
                BRA     CONT
* FIN DE BLOQUE DE ACCIONAMIENTO PRINCIPAL

* BLOQUE DE SUBROUTINAS
RET250M:     PSHH
                PSHX
                LDHX    #$C34E
VUELTA:      NOP
                NOP
                AIX     #$FF
                CPHX    #$0000
```

```

        BNE     VUELTA
        PULX
        PULH
        RTS
* FIN DE BLOQUE DE SUBROUTINAS

* BLOQUE DE DATOS
        ORG $D7FE ;SE COLOCA VECTOR DE RESET DE USUARIO
        DW $8000
* FIN DE BLOQUE DE DATOS

```

Dado que en este ejemplo no se utiliza el COP, éste es deshabilitado al inicio del programa. Para que éste se ejecute de forma autónoma simplemente se deben seguir los pasos descritos en el tema 6.3. Véanse los subtemas 6.3.1 y 6.3.2.

6.5 Fundamentos del convertidor análogo – digital (CAD)

Diversos miembros de la familia HC08 cuentan con un convertidor análogo-digital como bloque funcional interno, para el MCU presente en la tarjeta MINICON_08A (68HC908GP32CP9) éste es de 8 bits y se tienen 8 canales de entrada analógica. Los pines asociados con las entradas análogas están compartidos con las líneas de acceso al puerto B. El algoritmo de conversión empleado es el bien conocido de *aproximaciones sucesivas*.

En la figura 6.1 se muestra un diagrama a bloques simplificado del CAD presente en el MCU contenido en la TD.

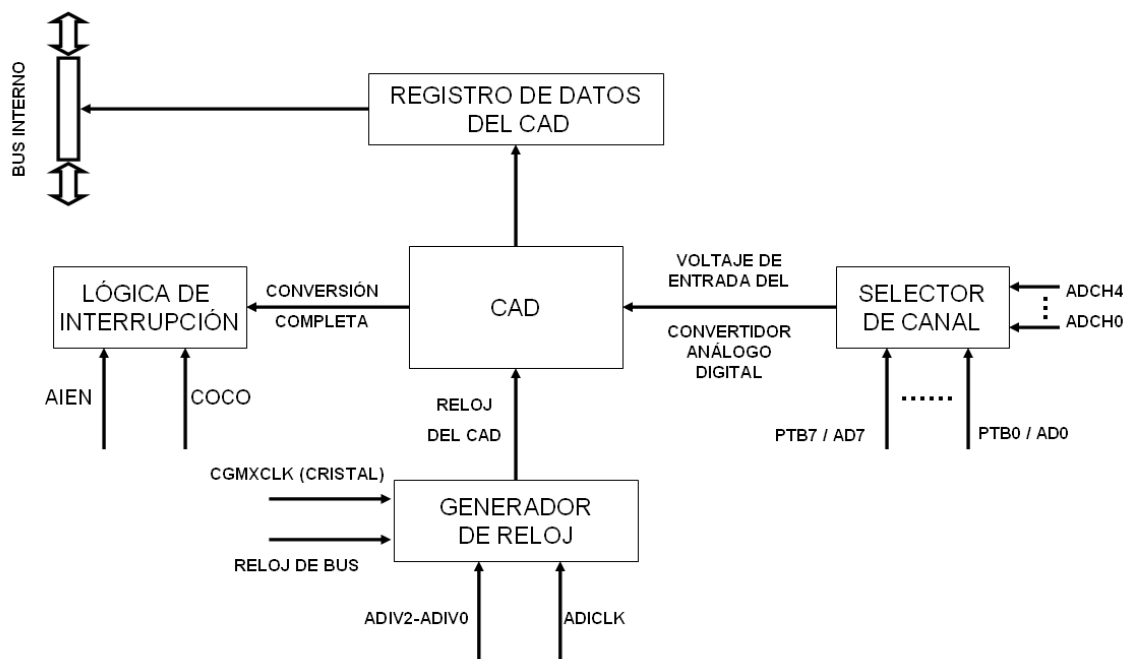


Figura 6.1 Esquema a bloques básico del CAD presente en el MCU 68HC908GP32

El tiempo de conversión es del orden de 16 a 17 periodos del reloj del CAD. **Para un funcionamiento óptimo, el fabricante recomienda que dicha señal de reloj tenga una frecuencia de de 1 MHz.** Esto se logra configurando bits del registro del CAD denominado ADCLK. Más adelante se explicará como hacer esto.

El canal a convertir es seleccionado por los bits ADCH4 a ADCH0 presentes en el registro de status y control del CAD (ADSCR). Después de un RESET estos bits son inicializados con el nivel uno lógico lo cual hace que el CAD esté desenergizado, y todos los pines asociados con el puerto B son líneas de entrada digital. Al seleccionarse un canal de entrada análoga, el pin asociado ya no estará ligado con las instancias de hardware del puerto binario B y cualquier invocación o configuración del bit de puerto asociado no tendrá efecto en dicho pin. En la tabla 6.2 se muestra la selección del canal a convertir en términos de los bits ADCH4 a ADCH0.

TABLA 6.2 SELECCIÓN DE CANAL DEL CAD EN BASE A BITS ADCH4-ADCH0

ADCH4	ADCH3	ADCH2	ADCH1	ADCH0	CANAL SELECCIONADO
0	0	0	0	0	PTB0/AD0
0	0	0	0	1	PTB1/AD1
0	0	0	1	0	PTB2/AD2
0	0	0	1	1	PTB3/AD3
0	0	1	0	0	PTB4/AD4
0	0	1	0	1	PTB5/AD5
0	0	1	1	0	PTB6/AD6
0	0	1	1	1	PTB7/AD7
0	1	0	0	0	RESERVADO
↓	↓	↓	↓	↓	
1	1	1	0	0	
1	1	1	0	1	VREFH
1	1	1	1	0	VREFL
1	1	1	1	1	CAD DESACTIVADO

Los límites inferior y superior dentro de los cuales la señal análoga a convertir debe estar, están fijados respectivamente por los voltajes presentes en los pines VSSAD/VREFL y VDDAD/VREFH del MCU. De esta forma, cuando la entrada análoga a convertir tenga un valor igual al voltaje presente en el pin VREFH el byte entregado por el CAD será \$FF. Por otro lado, si el nivel de la entrada análoga es igual al voltaje presente en el pin VREFL, el byte resultado de la conversión será \$00. Si el valor de la entrada análoga está entre VREFH y VREFL el byte entregado por el CAD estará, en proporción, entre \$FF y \$00. Si se denota como bycad al valor del byte entregado por el CAD después de una conversión, y como Vcade al voltaje análogo de entrada correspondiente, bycad estará dado en forma muy aproximada por la siguiente expresión:

$$bycad \approx I\left\{\frac{255}{VREFH - VREFL}(Vcade - VREFL)\right\} \quad (6.1)$$

Donde el $I\{x\}$ denota “parte entera del argumento ‘x’”.

En la práctica, por lo regular, los pines VREFL y VREFH se conectan respectivamente a tierra (0 Volts) y Vdd (+5 Volts); esto hará que bycad se obtenga usando la siguiente expresión:

$$bycad \approx I\{51(Vcade)\} \quad (6.2)$$

En la tarjeta MINICON_08A los pines VREFL y VREFH están conectados respectivamente a tierra y a VDD (+5 Volts), esto mediante los jumpers J16 y J6. Véase la tabla 2.4 en el capítulo

dos de este manual. Si se quitan los jumpers aquí mencionados, puede conectarse una fuente de precisión de 5Volts, o bien de otro valor conveniente, dedicada únicamente a polarizar el CAD y delimitar el intervalo analógico para la entrada correspondiente; esto hará que se tenga una mejor certidumbre en los niveles de voltaje en los pines VREFL y VREFH, lo cual hará que el valor obtenido para bycad en las ecuaciones 6.1 y 6.2 sea más cercano al valor real correspondiente.

6.5.1 Registros de configuración y operación asociados con el CAD

Para la operación y configuración del CAD el MCU de la tarjeta MINICON_08A cuenta con tres registros denominados **ADSCR**, **ADR** Y **ADCLK**. A continuación se describe la funcionalidad de estos.

6.5.1.1 Registro de estado y control del CAD (ADSCR)

Este registro gobierna diversos aspectos acerca de la configuración y operación del CAD. En la figura 6.2 se esquematiza éste junto con el valor de inicialización que se presenta después de un RESET.

Registro ADSCR. Dirección: \$003C

	COCO	AIEN	ADCO	ADCH4	ADCH3	ADCH2	ADCH1	ADCH0
RESET:	0	0	0	1	1	1	1	1

Figura 6.2. Esquema del registro ADSCR para uso del programador.

La funcionalidad de cada uno de los bits que forman parte de este registro se describe en seguida.

Bit ADCO (habilitador de conversión continua)

Si este bit es uno lógico, el CAD efectuará continuamente conversiones del canal de entrada análoga seleccionado, actualizando con el resultado de ésta el registro **adr** al final de cada conversión. En caso de que el bit ADCO tenga un nivel de cero lógico, una vez que se ha completado una conversión, para realizar otra se requiere que el registro **adscr** sea escrito de nuevo con un byte apropiado. Al RESET este bit se inicializa con cero lógico.

Bit AIEN (habilitador local de interrupción del CAD)

Si este bit es uno lógico, se genera una interrupción cada vez que se completa una conversión. La señal de interrupción es colocada en cero lógico cuando se hace una escritura al registro **adscr**; o bien, cuando se hace una lectura del registro **adr**. Cuando este bit es cero lógico no se genera un requerimiento de interrupción al completarse una conversión. Al RESET este bit se inicializa con cero lógico.

Bit COCO (conversión completada)

Si el bit AIEN es cero lógico, el bit COCO es de sólo lectura y **es colocado en el nivel uno lógico cada vez que se completa un proceso de conversión analógica – digital** y el proceso de conversión ha sido inicializado bajo la modalidad de efectuar una sola conversión (ADCO=0). Si el proceso de conversión se inicializó bajo la modalidad de conversiones continuas (ADCO=1), el bit COCO es colocado en el nivel de uno lógico, únicamente después de la primera conversión implicada. Este bit es colocado en cero lógico al efectuarse una escritura al registro **adscr**; o bien, cada vez que se hace una lectura del registro de datos del CAD (**adr**).

Cuando el bit AIEN es uno lógico, COCO es un bit de lectura/escritura, **el cual debe colocarse en cero lógico mediante código colocado dentro de la rutina de servicio de interrupción asociada.** Después de un RESET el bit COCO es inicializado en cero lógico.

Bits ACH4 a ACH0 (Selección de señal analógica a convertir)

Mediante estos bits se especifica que canal de entrada analógica se va a convertir a su equivalente digital. En La tabla 6.2 se aprecia el detalle de esto. Si la selección en binario contiene cinco unos (31 decimal) el CAD está desactivado; nótese que esta es la situación por defecto después de un RESET.

Para que de inicio el proceso de conversión, basta con que se efectuó una escritura al registro ADSCR con el byte apropiado a las características particulares de ésta. Por ejemplo, si se desea que se efectúe la conversión a digital de la entrada AD6 y se requiere que ésta sea bajo la modalidad de una sola conversión (ADCO=0); además de no requerirse que se genere una interrupción al ser completado el proceso. Para iniciar la conversión se deberá escribir el byte \$06 en el registro ADR.

6.5.1.2 Registro de datos del CAD (ADR)

Este registro se encuentra en la dirección \$003D y cada vez que se completa una conversión, el mismo se actualiza con el resultado de ésta. En la figura 6.3 se esquematiza el registro ADR.

Registro ADR.	Dirección \$003D							
Lectura:	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0
Escritura:	*****	*****	*****	*****	*****	*****	*****	*****
RESET:	0	0	0	0	0	0	0	0

***** No implementado

Figura 6.3. Esquema del registro ADR para uso del programador

6.5.1.2 Registro selector de reloj del CAD (ADCLK)

Mediante este registro se determina la frecuencia de la señal de reloj del CAD (ADC CLOCK). Véase la figura 6.1. **El fabricante recomienda que ésta sea de aproximadamente 1 MHz.** En la figura 6.4 se muestra un esquema de este registro para fines de programación.

Registro ADCLK.	Dirección \$003E							
Lectura:	ADIV2	ADIV1	ADIV0	ADICLK	0	0	0	0
Escritura:					*****	*****	*****	*****
RESET:	0	0	0	0	0	0	0	0

***** No implementado

Figura 6.4. Esquema del registro ADR para uso del programador

La señal de reloj propia del CAD se obtiene mediante un divisor de frecuencia aplicado a una señal que puede escogerse entre dos fuentes; una de las cuales es denominada como CGMXCLK y la otra es la señal de reloj de bus que aquí se denomina como SRBUS Reloj de bus. Si el bit ADICLK es cero lógico, la señal seleccionada es CGMXCLK; en otro caso, ésta será SRBUS. Para el MCU 68HC908GP32 la señal CGMXCLK viene siendo la que se genera a partir del cristal conectado a los pines OSC1 y OSC2; o bien, la que se genere en forma externa y se conecte al pin OSC1.

Por lo que toca al valor de la frecuencia de la señal SRBUS, si no se usa el PLL, ésta será la cuarta parte de la que corresponde a la señal CGMXCLK. Si se usa el PLL la frecuencia de la señal SRBUS será obtenida de acuerdo a como se configure éste.

Cabe señalar aquí que para la tarjeta MINICON_08A el PLL de origen no se usa.

A continuación se describe la funcionalidad de los bits que conforman el registro ADCLK.

Bit ADICLK (selector de reloj de entrada al CAD)

Si este bit es cero lógico el reloj de entrada al CAD es la señal denominada como CGMXCLK. Cuando este bit es uno lógico la señal de entrada al CAD es el reloj de bus (SRBUS).

Bits ADIV2 a ADIV0 (selector de preescalamiento)

Mediante estos tres bits se selecciona el preescalamiento aplicado a la señal de reloj de entrada al CAD para obtener la señal de reloj propia del CAD (ADC CLOCK). En la tabla 6.3 se ilustra esto.

TABLA 6.3. SEÑAL “ADC CLOCK” EN TERMINOS DE LOS BITS ADIV2 - ADIV0

ADIV2	ADIV1	ADIV0	Frecuencia de la señal ADC CLOCK
0	0	0	Frecuencia de señal de entrada al CAD/1
0	0	1	Frecuencia de señal de entrada al CAD/2
0	1	0	Frecuencia de señal de entrada al CAD/4
0	1	1	Frecuencia de señal de entrada al CAD/8
1	X	X	Frecuencia de señal de entrada al CAD/16

X denota indistinto (don't care)

Si la señal de CGMXCLK tiene una frecuencia mayor o igual que 1 MHz, es conveniente usar ésta como señal de entrada al CAD. En otro caso, es conveniente usar la señal SRBUS (reloj de bus) generada convenientemente con el PLL.

Para la tarjeta MINICON_08A la señal CGMXCLK es de 8 MHz; considerando esto y lo explicado acerca de la funcionalidad de los bits del registro ADCLK; una configuración de bits para lograr que la frecuencia de la señal ADC CLOCK sea 1 MHz, podría ser la siguiente:

- **ADICLK=0.** (Señal de entrada al CAD es CGMXCLK con frecuencia de 8 MHz).
- **ADIV2=0, ADIV1=1, ADIV0=1.** (Preescalamiento entre ocho).

Lo cual hace que \$60, sea un valor posible para el byte a escribir en el registro ADCLK previo al uso del CAD. Esto hará que la frecuencia de la señal ADC CLOCK sea 1 MHz tal como lo recomienda el fabricante.

Para más detalles acerca de la configuración y operación del CAD puede verse la sección 5 de la referencia [1].

Para fines ilustrativos, a continuación se muestra un ejemplo de uso del CAD, empleando recursos didácticos presentes en la tarjeta MINICON_08A.

Ejemplo 4. Conversión del canal AD7 con despliegue de resultado en el puerto A

En este ejemplo se hace la conversión a digital del canal AD7, desplegándose en tiempo real el byte resultado de ésta en el puerto A. La conversión se hace bajo la modalidad de conversiones continuas deshabilitadas (ADCO=0 en registro ADSCR). La señal de entrada al canal AD7 es suministrada por el circuito mostrado en la figura 6.5, apreciándose ahí que ésta estará

comprendida en el rango de cero a cinco volts. El potenciómetro POT2, el capacitor C12 y el jumper J6 son parte de la tarjeta MINICON_08A.

Al ejecutarse el programa, deberá apreciarse en los ocho LEDS que testifican el estado del puerto A un perfil de unos y ceros, que denotan el byte resultado de la conversión. Al ajustarse el POT2 se observará que el byte resultado cambia.

Para un determinado ajuste del POT2, puede medirse el voltaje presente en el pin VREFH/VDDAD así como la tensión presente en la entrada AD7 (Vcade). Considerando VREFL como cero y tomando como valor de bycad al desplegado por los LEDS conectados al puerto A. Deberá corroborarse que el valor bycad es muy aproximado o igual al que se obtiene mediante la expresión 6.1.

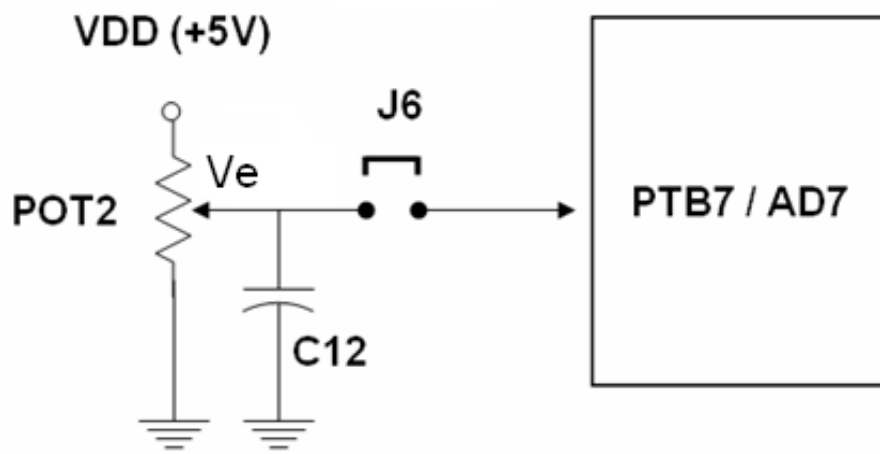


Figura 6.5. Circuito presente en la tarjeta MINICON_08A para suministrar una señal de CD de prueba al canal AD7 del CAD.

El programa asociado se almacenó en el archivo pconv1.asm y se muestra a continuación;

```
* EJEMPLO 4
* CONVERSIÓN DEL CANAL AD7 CON RESULTADO MOSTRADO EN EL PUERTO A
* Por: ANTONIO SALVÁ CALLEJA
* Noviembre de 2008
* ALMACENADO EN EL ARCHIVO PCONV1.ASM
PTA      EQU    $00
DDRA     EQU    $04
ADSCR    EQU    $3C
ADR      EQU    $3D
ADCLK    EQU    $3E

      ORG    $8000 ;EL PROGRAMA SE ORIGINA EN LA LOCALIDAD $8000 DE FLASH

* BLOQUE DE INICIALIZACIÓN
      MOV    #$FF,DDRA ;CONFIGURA PUERTO A COMO SALIDA.
      MOV    #$60,ADCLK ;FRECUENCIA DE SEÑAL ADC CLOCK SERÁ 1 MHz.
* FIN DE BLOQUE DE INICIALIZACIÓN

* BLOQUE DE ACCIONAMIENTO PRINCIPAL
INICONV: MOV    #$07,ADSCR ;INICIA CONVERSIÓN DE CANAL AD7.
CHECOCO: BRCLR 7,ADSCR,CHECOCO ;ESPERA FIN DE CONVERSIÓN.
      MOV    ADR,PTA ;COLOCA RESULTADO EN PUERTO A.
      BRA    INICONV ;PASA A EFECTUAR OTRA CONVERSIÓN.
* FIN DE BLOQUE DE ACCIONAMIENTO PRINCIPAL
```

6.6 Interfaz para desplegado alfanumérico presente en la tarjeta MINICON_08A

La interfaz para LCD presente en la tarjeta MINICON_08A está validada en el conector 3 de ésta. Se ubica en el extremo superior derecho de la tarjeta. Cuenta con 16 pines para conexión de un desplegado alfanumérico. El pin 16 es el que se encuentra en el extremo derecho de éstos. En la figura 6.6 se muestra una fotografía donde se aprecia la interfaz para desplegado de la tarjeta MINICON_08A y en la figura 6.7 se aprecia un posible conexión a ésta de un LCD.



Figura 6.6 Vista de la interfaz para LCD presente en la tarjeta MINICON_08A

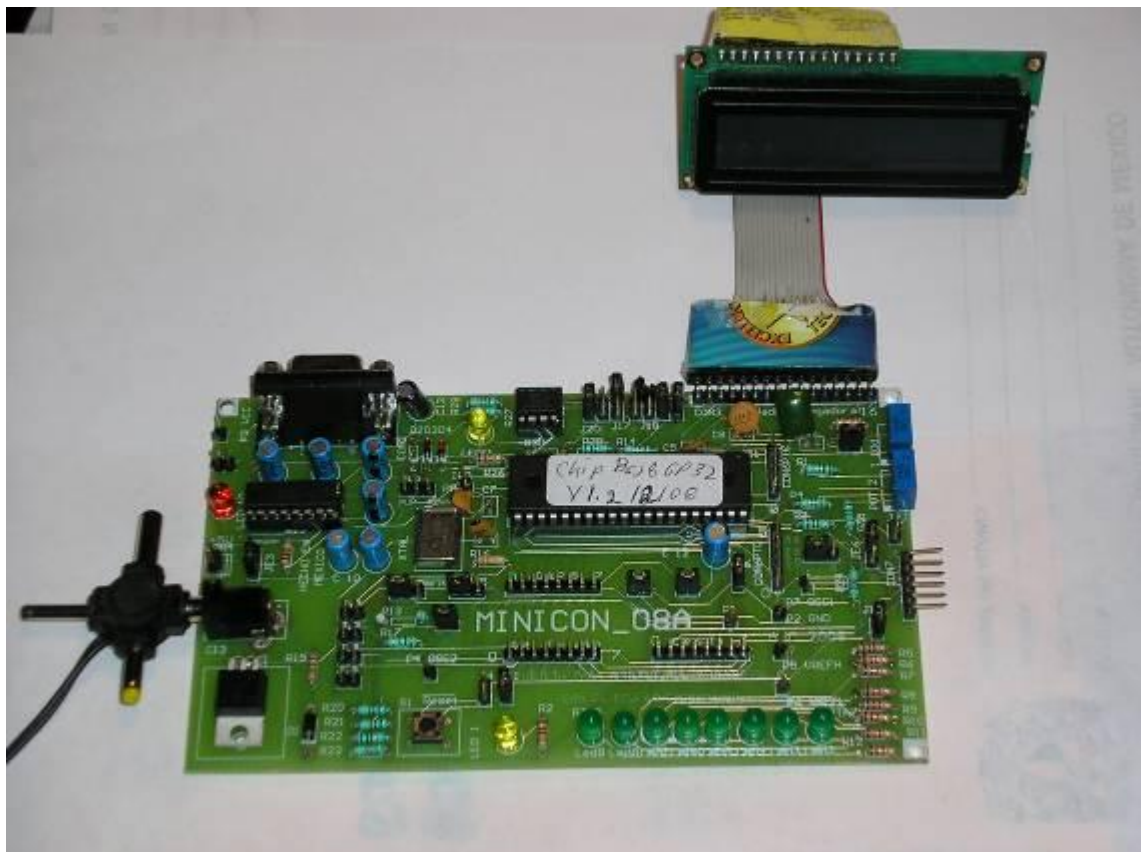


Figura 6.7 Una forma de conexión de un LCD a la tarjeta MINICON_08A

Los desplegados alfanuméricos presentan diversos formatos en cuanto a la organización de la información desplegada en su pantalla; entre otros, estos formatos podrían ser: 1 renglón por 16

columnas, 2 renglones por 16 columnas, o bien, 4 renglones por 20 columnas. Para fines ilustrativos, en las explicaciones y ejemplos asociados en esta sección, se emplea un desplegado de 2 renglones por 16 columnas de aquí en adelante denominado como LCD16x2.

Para fines de la conexión con el dispositivo electrónico que los va a comandar y manejar, los desplegados alfanuméricos cuentan con pines para este propósito. Frecuentemente estos son 16 y están colocados en línea. En la figura 6.8 se muestra una vista posterior de un LCD16x2 donde se aprecia la tira de 16 postes en línea para conexión con el dispositivo que emplearía este LCD. Para el LCD mostrado, el pin 16 es el que se encuentra en la extrema derecha de éstos.

En la figura 6.9 se muestra una vista frontal del LCD cuya vista posterior se aprecia en la figura 6.8.

Los pines de un LCD están asociados con la polarización, y con las señales de control e intercambio de información con el dispositivo manejador.

En la tabla 6.3 se detalla la funcionalidad de cada uno de los pines de un LCD como podría ser el LCD16x2 y los recursos de la tarjeta MINICON_08A empleados cuando ésta usa y maneja un desplegado.

TABLA 6.3. FUNCIONALIDAD DE LOS PINES DE UN LCD ALFANUMÉRICO. Y RECURSOS PRESENTES EN LA TARJETA MINICON_08A EMPLEADOS CUANDO ÉSTA LO MANEJA

PIN	FUNCIONALIDAD	RECURSO DE TARJETA MINICON_08A PRESENTE EN SU INTERFAZ PARA LCD (CON 3)
1	Polarización negativa	Tierra (Vss)
2	Polarización positiva	+ 5 Volts (Vdd)
3	Vc (ajuste de contraste)	Señal entre 0 y 5 Volts generada por el potenciómetro POT1 presente en la tarjeta
4	RS, selector comando/dato Para comando RS=0. Para dato RS=1.	PTA2
5	Lectura/Escritura	Tierra (Vss)
6	E (habilitación)	PTA3
7	DB0	No conexión
8	DB1	No conexión
9	DB2	No conexión
10	DB3	No conexión
11	DB4	PTA4
12	DB5	PTA5
13	DB6	PTA6
14	DB7	PTA7
15	ALI	Conectado a +Vcc por el puente J9
16	CLI	Tierra (Vss)

6.6.1 Elementos del manejo de un LCD16x2

Los desplegados alfanuméricos aquí tratados de hecho son pequeños sistemas embebidos dedicados a la acción desplegada. Para manejarlos se requiere de la escritura a estos de *bytes comando y bytes dato*. Un byte dato por lo general será el código asociado con el carácter a desplegar en un momento dado; para caracteres occidentales este código es el ASCII convencional. Un byte comando por lo general será un valor asociado con alguna configuración de funcionamiento del LCD, como podrían ser entre otros los siguientes:

- Cada una de los bytes comando que han de escribirse al LCD al inicializar éste.
- Comandos para especificar columna y renglón del próximo carácter a desplegar.
- Comando para poner en blanco la pantalla del LCD.
- Comando para habilitar presencia o ausencia de cursor.

Para escribir en el desplegado comandos o datos, se requiere la colocación de la información propia de éstos en la líneas de datos del dispositivo y de la generación de la señalización apropiada en las líneas de control RS, R/W y E. Para una descripción más detallada acerca de la funcionalidad de los bytes de tipo comando y dato y de la señalización requerida para la escritura de éstos puede verse [4].



Figura 6.8 Vista posterior de un LCD16x2 donde se aprecian los 16 postes para conexión con el dispositivo manejador

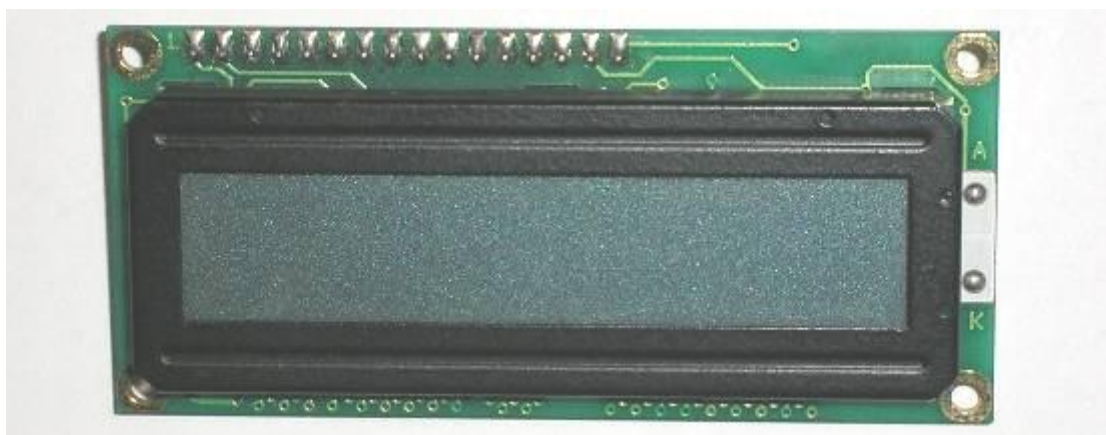


Figura 6.9 Vista anterior del LCD16x2 mostrado en la figura 6.6

6.6.1.1. Bytes comando de posicionamiento para un LCD16x2

Los bytes comando asociados con el posicionamiento del siguiente carácter a desplegar son de mucha importancia; si se desea que el carácter inicial de una cadena de caracteres a mostrar aparezca en una determinada posición (columna y renglón), antes de que se escriba el byte dato apropiado, se debe escribir un byte comando de posicionamiento. En la tabla 6.4 se muestra el valor de los bytes comando de posicionamiento un LCD16x2.

TABLA 6.4. VALOR DE LOS BYTES COMANDO DE POSICIONAMIENTO PARA UN LCD16x2.

POSICIONAMIENTO (REGLÓN/COLUMNA)	VALOR DE BYTE COMANDO (HEX)
1/1	80
1/2	81
1/3	82
1/4	83
1/5	84
1/6	85
1/7	86
1/8	87
1/9	88
1/10	89
1/11	8A
1/12	8B
1/13	8C
1/14	8D
1/15	8E
1/16	8F
2/1	C0
2/2	C1
2/3	C2
2/4	C3
2/5	C4
2/6	C5
2/7	C6
2/8	C7
2/9	C8
2/10	C9
2/11	CA
2/12	CB
2/13	CC
2/14	CD
2/15	CE
2/16	CF

En [4] puede verse el valor de los bytes comando asociados con desplegados de otro tamaño, como podrían ser los de 4 renglones y 16 columnas o bien los de un renglón y 16 columnas. Por ser motivo de confusión de los usuarios de estos dispositivos, en la tabla 6.5 se muestra el valor de los bytes comando de posicionamiento asociados con un LCD16x1.

TABLA 6.5. VALOR DE LOS BYTES COMANDO DE POSICIONAMIENTO PARA UN LCD16x1.

POSICIONAMIENTO (RENGLÓN/COLUMNA)	VALOR DE BYTE COMANDO (HEX)
1/1	80
1/2	81
1/3	82
1/4	83
1/5	84
1/6	85
1/7	86
1/8	87
1/9	C0
1/10	C1
1/11	C2
1/12	C3
1/13	C4
1/14	C5
1/15	C6
1/16	C7

6.6.2. Software para el manejo de un LCD16x2

En lo general, el software requerido para manejar un desplegado intercambiará información con éste. Esto se reducirá simplemente a la escritura de información desde el sistema manejador, como podría ser la tarjeta MINICON_08A, al LCD; o bien, la lectura de información del desplegado por parte del sistema manejador. Por lo general la información leída testifica algo relacionado con el *status* del LCD, como podría ser el verificar que éste ya ha procesado el último byte dato o comando recibido, y está listo para recibir otro dato o comando. Dado que en la interfaz de LCD que aquí nos ocupa la línea R/W está ligada a tierra el software de uso del desplegado consistirá únicamente de escritura de bytes comando y datos apropiados. Después de cada escritura deberá haber tiempos de espera que garanticen que el LCD esté listo para recibir otro byte. El valor de los tiempos requeridos por el LCD para procesar la información escrita a éste puede verse en [4].

Para que un determinado sistema, como podría ser la tarjeta MINICON_08A, pueda manejar un LCD, éste debe ser inicializado previamente mediante la escritura secuencial de diversos bytes comando [4]. Aspectos importantes que quedan configurados después de la fase de inicialización son entre otros los siguientes:

- Tamaño de la interfaz (4 ú 8 bits).
- Tipo de matriz de puntos asociado con el despliegue de los caracteres.
- Presencia o ausencia del cursor

Una vez que el LCD ha sido inicializado, éste podrá desplegar texto requiriéndose para esto simplemente de la escritura secuencial apropiada de sendos bytes dato asociados con el texto a desplegar.

Para facilitar el manejo de desplegados conectados a la interfaz propia para ellos presente en la tarjeta MINICON_08A, el autor desarrolló software acorde con la interfaz de hardware implicada, véase la tabla 6.3. Este software se denomina MDAM8A00 y contiene código que

inicializa el desplegado y subrutinas para escribir bytes comando o dato; o bien, desplegar un texto que comprenda 16 caracteres en uno de los renglones que contempla el desplegado. Adicionalmente a las rutinas antes mencionadas el software MDAM8A00 contiene rutinas auxiliares de retardo para generar los tiempos de espera requeridos mencionados en párrafos anteriores.

El software MDAM8A00 está contenido en un archivo denominado 'mdam8a00.asm' e incluye lo siguiente:

- Código inicializador del desplegado.
- Salto al final del MDAM8A00.
- Subrutina de escritura de comandos al LCD, denominada '**escom4**'.
- Subrutina de escritura de datos al LCD, denominada '**escdat4**'.
- Subrutina de escritura de 16 caracteres a un renglón del LCD, denominada '**copiadis**'.
- Subrutinas auxiliares.

El flujo de ejecución del MDAM8A00 es el siguiente:

1. Se inicializan como salida todas las líneas del puerto A.
2. Se genera una espera del orden de 24 ms ($F_{bus} = 2 \text{ MHz}$).
3. Se escriben los bytes comando requeridos para inicializar el LCD.
4. Se salta al final del MDAM8A00.

La inicialización hecha por el software MDAM8A00, hace que el LCD presente para su manejo las siguientes características:

- El tamaño de la interfaz es de 4 bits.
- La matriz de los caracteres es de 7x5.
- El cursor es invisible.
- La escritura es de izquierda a derecha.

De lo explicado anteriormente se intuye, que un programa de aplicación en el que se use el software MDAM8A00 para fines del manejo del desplegado deberá, en lo general, presentar el siguiente flujo de ejecución:

1. Deshabilitar el COP si esto es necesario.
2. Ejecutar el software MDAM8A00.
3. Ejecutar código del usuario, donde para fines del LCD, se invocarán según sea necesario las subrutinas **escom4**, **escdat4** y **copiadis**.

Para fines del paso dos del flujo de ejecución descrito, se podría usar la directiva '**\$include**'; por ejemplo, esto podría hacerse mediante la línea de código en ensamblador mostrada a continuación:

\$include "e:\hc08\mdam8a00.asm"

Donde se supone que el archivo **mdam8a00.asm** se encuentra en la carpeta '**hc08**', contenida en la unidad de disco '**e**', del sistema desde donde se ejecute el software PUMMA_08+. Si este no es el caso en el sistema del usuario, simplemente se debe colocar lo propio acerca de la trayectoria (path) asociada con la carpeta donde se encuentre el archivo **mdam8a00.asm** en el argumento de la directiva '**\$include**' empleada.

Tal como se ha dicho, el manejo del LCD desde un programa que emplee el software manejador de éste aquí descrito, simplemente se haría con invocaciones a las subrutinas `escdat4`, `escom4` y `copiadis` según lo vaya requiriendo la aplicación.

A continuación se describe la funcionalidad de las tres subrutinas contenidas en el software MDAM8A00, que se usan al manejar un LCD conectado a la interfaz propia de éste, contenida en la tarjeta MINICON_08A (CON3), véase la tabla 6.3.

6.6.2.1. Funcionalidad de la subrutina `escom4`

La subrutina `escom4` escribe un byte comando al LCD. Antes de invocarse el comando a escribir debe precargarse en el acumulador 'A' del CPU, después de esto simplemente se invoca la subrutina. Por ejemplo, supóngase que en un programa de aplicación se requiere, en un tramo de éste, escribir el byte comando \$80 al LCD; el código en ensamblador dentro del programa se destaca en negritas a continuación:

```
$include "e:\hc08\mdam8a00.asm"

        .
        (código adicional del usuario)
        .
        lda #$80
        jsr escom4
        .
        (código adicional del usuario)
        .
fin:      bra fin    ;Fin del programa.
```

6.6.2.2. Funcionalidad de la subrutina `escdat4`

La subrutina `escdat4` escribe un byte dato al LCD. Antes de invocarse el dato a escribir debe precargarse en el acumulador 'A' del CPU, después de esto simplemente se invoca la subrutina. Por ejemplo, supóngase que en un programa de aplicación se requiere, en un tramo de éste, escribir el byte dato \$6A al LCD. El código en ensamblador dentro del programa se destaca en negritas a continuación:

```
$include "e:\hc08\mdam8a00.asm"

        .
        (código adicional del usuario)
        .
        lda #$6a
        jsr escdat4
        .
        (código adicional del usuario)
        .
fin:      bra fin    ;Fin del programa.
```

6.6.2.3. Funcionalidad de la subrutina `copiadis`

Esta subrutina escribe 16 caracteres al LCD en uno de los renglones de éste. Antes de invocarse el acumulador 'A' deberá precargarse con el byte \$80, si se desea que el texto se copie al renglón uno; por otro lado, si se desea que el texto se despliegue en el renglón dos, el byte a precargar en el acumulador 'A' deberá ser \$C0. Además, el par **H:X** deberá precargarse con la dirección a partir de la cual están colocados en memoria los bytes que representan los caracteres asociados con la leyenda a desplegar. Por ejemplo, supóngase que se desea desplegar en el renglón dos del LCD la leyenda **"Si funciona bien"**. El código en ensamblador dentro del programa se destaca en negritas a continuación:

```

$include "e:\hc08\mdam8a00.asm"

        .
        (código adicional del usuario)

        .
        lda #$c0
        ldhx #mitexto
        jsr copiadis
        .
        (código adicional del usuario)

fin:          bra fin ;Fin del programa.

** Inicio de área de datos del programa **
*          1234567890123456
mitexto:    fcc "Si funciona bien"

```

Nótese el uso de la pseudoinstrucción **fcc** para validar el texto a colocar, para detalles acerca de ésta, véase la sección 5.2.7 del capítulo 5.

6.6.2.4. Calibración de contraste previa al uso de un LCD

Antes de poder usar un LCD conectado a un determinado sistema embebido, se debe ajustar el contraste de éste. Para la tarjeta MINICON_08A, esto se hace ajustando el potenciómetro 1 (POT1) de modo que en el renglón uno de la pantalla aparezcan rectángulos oscuros en la posición de cada carácter. Véase la figura 6.10.

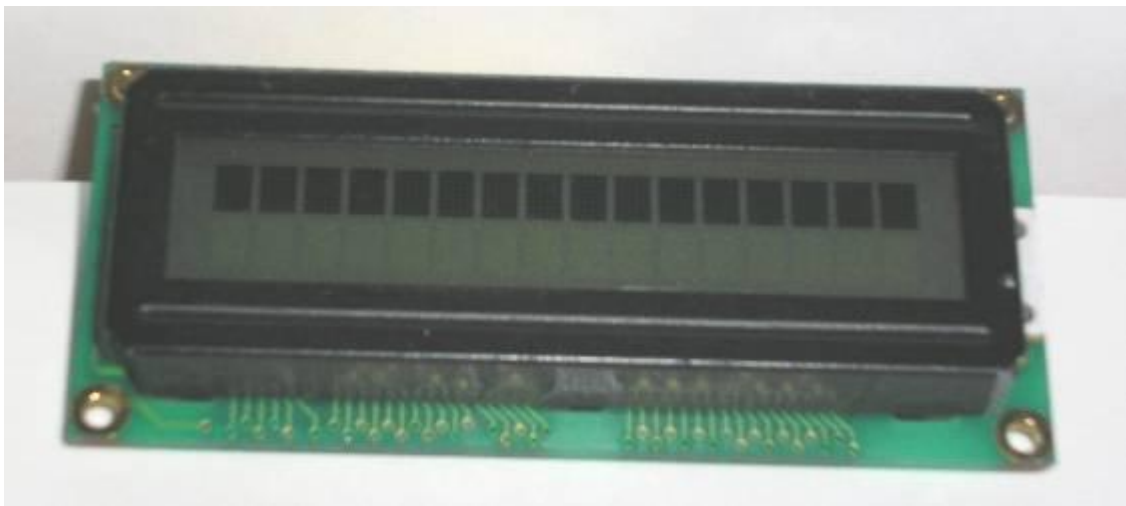


Figura 6.10. Aspecto de la pantalla del LCD una vez que se ha hecho la calibración del contraste.

Es importante señalar que si no se hace esta calibración previa, no será visible en la pantalla del LCD ningún texto enviado al ejecutarse un determinado programa de aplicación.

Ejemplo 5. Despliegue en el LCD de texto y caracteres aislados

En este ejemplo se muestra un programa ejecutable desde RAM que al correr hace que aparezca la cadena 'DB' a partir de la columna tres del renglón uno, y en el renglón dos aparece la leyenda 'Si funciona bien'. El programa se denominó 'LCDGP1' y está contenido en el archivo **lcdgp1.asm**. El código en ensamblador correspondiente se muestra a continuación:

```

* EJEMPLO 5
* DESPLIEGUE EN EL LCD DE TEXTO Y CARACTERES AISLADOS
* Por: ANTONIO SALVÁ CALLEJA
* Noviembre de 2008
* ALMACENADO EN EL ARCHIVO LCDGP1.ASM

```

```

        ORG $0100 ;EL PROGRAMA SE ORIGINA EN LA LOCALIDAD $0100 DE RAM

```

```

* BLOQUE DE INICIALIZACIÓN
$INCLUDE "E:\HC08\MDAM8A00.ASM"
* FIN DE BLOQUE DE INICIALIZACIÓN

```

```

* BLOQUE DE ACCIONAMIENTO PRINCIPAL
        LDA #$82
        JSR ESCOM4 ;POSICIONA EN COLUMNA 3 RENGLÓN 1.
        LDA #$44
        JSR ESCDAT4 ;ESCRIBE ASCII DE 'D' .
        LDA #$42
        JSR ESCDAT4 ;ESCRIBE ASCII DE 'B' .
        LDHX #TEXTO
        LDA #$C0
        JSR COPIADIS ;ESCRIBE LEYENDA
FIN:     BRA FIN
* FIN DE BLOQUE DE ACCIONAMIENTO PRINCIPAL

```

```

* BLOQUE DE DATOS
*             1234567890123456
TEXTO:      FCC "SI FUNCIONA BIEN"

```

En la figura 6.11 se muestra la pantalla del desplegado después de ejecutarse el programa LCDGP1.



Figura 6.11. Despliegue en la pantalla del LCD después de que ejecutarse el programa LCDGP1.

Ejemplo 6. Despliegue en el LCD de el valor del nivel en la entrada AD7

En este ejemplo se despliega en tiempo real en el primer renglón del LCD, el valor de un voltaje denominado 'Ve' aplicado a la entrada analógica AD7 del MCU. Esto al variar Ve entre cero y cinco volts, mediante el potenciómetro POT2 presente en la tarjeta MINICON_08A, véase la figura 6.5.

Se desea que el despliegue sea con un número de cifras decimales especificado como una constante mediante una sentencia 'equ'; para el caso concreto aquí ilustrado, se desplegarán cuatro dígitos para la parte fraccionaria de Ve. Por ejemplo, si Ve fuera 1.9607 Volts, en el LCD deberá aparecer en el primer renglón a partir de la primera columna, la siguiente cadena de caracteres:

$$Ve=1.9607 \text{ V}$$

De lo anteriormente expuesto, se intuye que en el programa implicado para este ejemplo se usará un poco de aritmética y además puede usarse parte del código que corresponde al ejemplo 4, anteriormente descrito en este manual.

Para casos como éste, donde el programa es seguramente más complicado, conviene desdoblar el problema en bloques funcionales de software que se prueban individualmente. Dichos tramos de código por lo regular se diseñan como subrutinas. Para este ejemplo se diseñó primero una subrutina denominada '**convby5v**' que obtiene el valor en decimal asociado con el valor de un byte, como puede ser el entregado por el CAD del MCU, después de efectuar la conversión a digital de una determinada entrada análoga. Al retornar de la subrutina, los bytes que representan al código ASCII correspondientes a cada uno de los dígitos de la representación en decimal de Ve, aparecen colocados a partir de una dirección de memoria RAM denotada como '**pent**'.

A continuación se explica como empleando aritmética se pueden obtener los dígitos de la representación decimal de Ve. Se supone que VREFH es cinco volts y que VREFL es cero volts, considerando esto a partir de la ecuación 6.1 se sabe que Ve expresado en función de el valor del byte entregado por el CAD estaría dado por:

$$Ve \approx \frac{bycad}{51} \quad (6.3)$$

Donde *bycad* representa el valor del byte entregado por el CAD. Desdoblando la división expresada en la ecuación 6.3 se obtiene la siguiente ecuación:

$$Ve = C0 + \frac{Re0}{51} \quad (6.4)$$

Donde C0 denota la parte entera del resultado de la división y Re0 denota al residuo de ésta. Claramente se aprecia que C0 representa el valor de la parte entera de Ve, y que el valor dado por la división Re0/51 es menor que uno y representa a la parte fraccionaria de Ve. Si se quisiera obtener el primer dígito a la derecha del punto decimal simplemente se multiplica la parte fraccionaria de Ve por diez y se obtiene la parte entera de este último resultado, o sea:

$$D1 = I\{10Fr\} \quad (6.5)$$

Donde D1 representa el valor del primer dígito a la derecha del punto en la representación decimal de Ve, y Fr representa la parte fraccionaria de Ve. Dado que Fr es el resultado de la división Re0/51 es fácil ver que D1 será el valor del cociente que se obtiene al efectuarse la división 10Re0/51. Además, si se denota al residuo de esta última división como Re1, el cociente obtenido al efectuar la división 10Re1/51 representará el valor del segundo dígito a la derecha

del punto para la representación decimal de Ve. Por lo tanto, para obtener el valor de un cierto número 'n' de dígitos decimales a la derecha del punto, simplemente habrá que efectuar 'n' divisiones en secuencia, siendo el dividendo el último residuo obtenido multiplicado por diez y el divisor el número 51, de esta forma el dígito decimal i-ésimo 'Di' a la derecha del punto, estaría dado por:

$$Di = I\left\{\frac{10Re_{i-1}}{51}\right\} \quad (6.6)$$

Donde i estará comprendido entre 1 y 'n'. Entonces el software que obtenga la cadena de caracteres que representan al valor decimal de Ve podría estar integrado por los siguientes pasos:

1. Se inicializa un apuntador, denominado apunfrac, con la dirección de colocación del código ASCII que represente el valor del primer dígito decimal a la derecha del punto.
2. Se inicializa con el valor 'n' el contador descendente de dígitos de la parte fraccionaria.
3. Se obtiene el cociente asociado con la división bycad/51
4. Se suma al valor \$30 al valor obtenido en el paso 1, para obtener el código ASCII de la parte entera de Ve.
5. Se almacena este valor en la localidad de RAM que se había definido mediante la cadena 'pent'.
6. Se obtiene el cociente asociando con la división 10Reu/51, donde Reu representa el valor del residuo de la última división efectuada.
7. Se obtiene el código ASCII del dígito decimal actual a la derecha del punto, sumando \$30 al resultado del paso anterior.
8. El resultado del paso anterior se almacena en la dirección especificada por apunfrac.
9. Se incrementa el apuntador apunfrac.
10. Se decrementa el contador de dígitos de la parte fraccionaria a generar.
11. Si el valor obtenido en el paso anterior es cero se coloca el código ASCII del punto decimal en la localidad pent+1, en otro caso se retorna al paso 6.
12. Se concluye retornando de la subrutina.

El código de obtención de la representación decimal del valor de la entrada analógica se validó mediante la una subrutina denominada '**convby5v**' escrita en ensamblador donde se usan localidades de RAM para parámetros tales como el apuntador de colocación del siguiente dígito fraccionario y el número de éstos a desplegar; para declarar las variables mencionadas y las demás implicadas, se emplean directivas de tipo 'equ'; para detalles acerca de ellas puede verse la sección 5.x de este manual. El código fuente de la subrutina **convby5v** se muestra a continuación, precedido por las declaraciones 'equ' requeridas para las variables en RAM a usar.

```
* DECLARACIONES EQU REQUERIDAS POR LA SUBROUTINA CONVBY5V
PENT EQU $E0      ;DIRECCIÓN ORIGEN DE COLOCACIÓN DE CADENA DECIMAL
FR1 EQU $00E2     ;DIRECCIÓN DE COLOCACIÓN DE ASCII DE PRIMER.
*                DÍGITO A LA DERECHA DEL PUNTO.
CONTR EQU $F0     ;DIRECCIÓN DE CONTADOR DE DÍGITOS FRACCIONARIOS.
NUMFR EQU $04     ;PREDETERMINACIÓN DE NÚMERO DE DÍGITOS
*                FRACCIONARIOS.
APUNFRAC EQU $F1  ;DIRECCIÓN DE APUNTADOR DE PRÓXIMO DÍGITO
*                FRACCIONARIO A COLOCAR.
ASCIIIP EQU $2E   ;CÓDIGO ASCII DEL CARÁCTER `.`.
```



```

*****
* SUBROUTINA CONVBY5V
* Por: Antonio Salvá Calleja.
* Enero de 2009.
* A partir del byte generado por el CAD, al completarse una
* conversión a digital, obtiene la representación en decimal
* como cadena de caracteres ASCII, correspondiente al valor Ve
* presente en la entrada analógica que corresponda.
*
* Antes de invocar:
* a<--byte generado por el CAD (bycad).
*
*Al retornar:
* (pent:pent+1::pent+nfr+1)<--representación decimal de Ve.
* Donde nfr representa el número de dígitos fraccionarios
* predeterminado en sentencia equ previa.
* a<--valor del último residuo.
*****

CONVBY5V:      PSHH
                PSHX

                MOV #NUMFR,CONTR ;INICIALIZA CONTR
                LDHX #FR1
                STHX APUNFRAC ;INICIALIZA APUNTADOR DE FRACCIONES

                CLRH
                LDX #$33 ;DIVISOR 51
                DIV
                ADD #$30 ;A<-- ASCII DE PARTE ENTERA
                STA PENT
                PSHH
OTRODIGFRAC:   PULA ;A<---RESIDUO
                LDX #$0A
                MUL ;X:A<--RESIDUO X 10
                PSHX
                PULH ;H:A<---RESIDUO X 10
                LDX #$33
                DIV
                ADD #$30 ;A<--ASCII DE DÍGITO A LA DERECHA DEL
PUNTO
                PSHH ;GUARDA EN STACK RESIDUO DE ÚLTIMA DIVISIÓN.
                LDHX APUNFRAC
                STA ,X ;COLOCA EN BUFFER EL ÚLTIMO DÍGITO
FRACCIONARIO ENCONTRADO.
                AIX #$01
                STHX APUNFRAC ;ACTUALIZA APUNTADOR DE DÍGITOS
FRACCIONARIOS.

                DEC CONTR
                BNE OTRODIGFRAC
                MOV #$2E,PENT+1 ;COLOCA PUNTO DECIMAL.

```

PULA ;A<--ÚLTIMO RESIDUO

PULX

PULH

RTS

Para verificar el correcto funcionamiento de la subrutina convby5v se puede correr un pequeño programa que la invoque para un determinado byte de prueba, dicho programa podría ejecutarse en RAM y en éste simplemente se precarga el acumulador A del cpu con el valor de prueba, y seguido de esto se invoca la subrutina, al retornar de ésta se pasa a un lazo infinito del que se sale oprimiendo el botón de RESET de la tarjeta MINICON_08A. Al programa de prueba se le denominó 'pconvby5v'. En la figura 6.12 se muestra el código de prueba ya presente en la ventana del editor de PUMMA_08+, se aprecia que el valor del byte de prueba empleado fue \$F0 (240d). De acuerdo con la ecuación 6.3, el valor en decimal de 'Ve' asociado con el valor del byte de prueba usado debe ser **4.7058** expresado con cuatro dígitos fraccionarios.

```

C:\hc08\pconvby5v.asm
Archivo  Editar  Buscar  Opciones  Ayuda
nvo  abr  gdr  ctr  cpr  pgr  e-ram  e-feep  ens  e-pfbas  e-pfbas  b-feep  min

pent equ $e0
fr1  equ $00e2
contfr equ $f0
numfr equ $04
apunfrac equ $f1
asciip equ $2e

                org $0100
                lda #$f0
                bsr convby5v
fin:            bra fin

convby5v:       pshh
                pshx
                mov #numfr,contfr ;inicializa contfr
                ldhx #fr1
                sthx apunfrac ;inicializa apuntador de fracciones
                clrh
                ldx #$33 ;divisor 51
                div
                add #$30 ;a<-- ascii de parte entera
                sta pent
                pshh
otrodigfrac:    pula ;a<---residuo
                ldx #$0a
                mul ;x:a<--residuo x 10
                pshx
                pulh ;h:a<---residuo x 10

```

Figura 6.12 Programa para probar la funcionalidad de la subrutina convby5v presente en la ventana del editor de PUMMA_08+.

Una vez que el programa de prueba se ha ejecutado y se ha oprimido el botón de RESET en la tarjeta destino, se podrá examinar la memoria en ésta mediante la opción 'examinar páginas de 256 bytes' presente en la ambiente de manejo hexadecimal (AMH) de PUMMA_08+. En la figura 6.13 se muestra el despliegue correspondiente al contenido de memoria correspondiente a la página cero, se aprecia ahí que a partir de la dirección de RAM asociada con la cadena 'pent' (\$00E0), se han colocado los seis bytes que representan el código ASCII de cada uno de los caracteres que conforman la representación decimal de 'Ve' con cuatro dígitos fraccionarios.

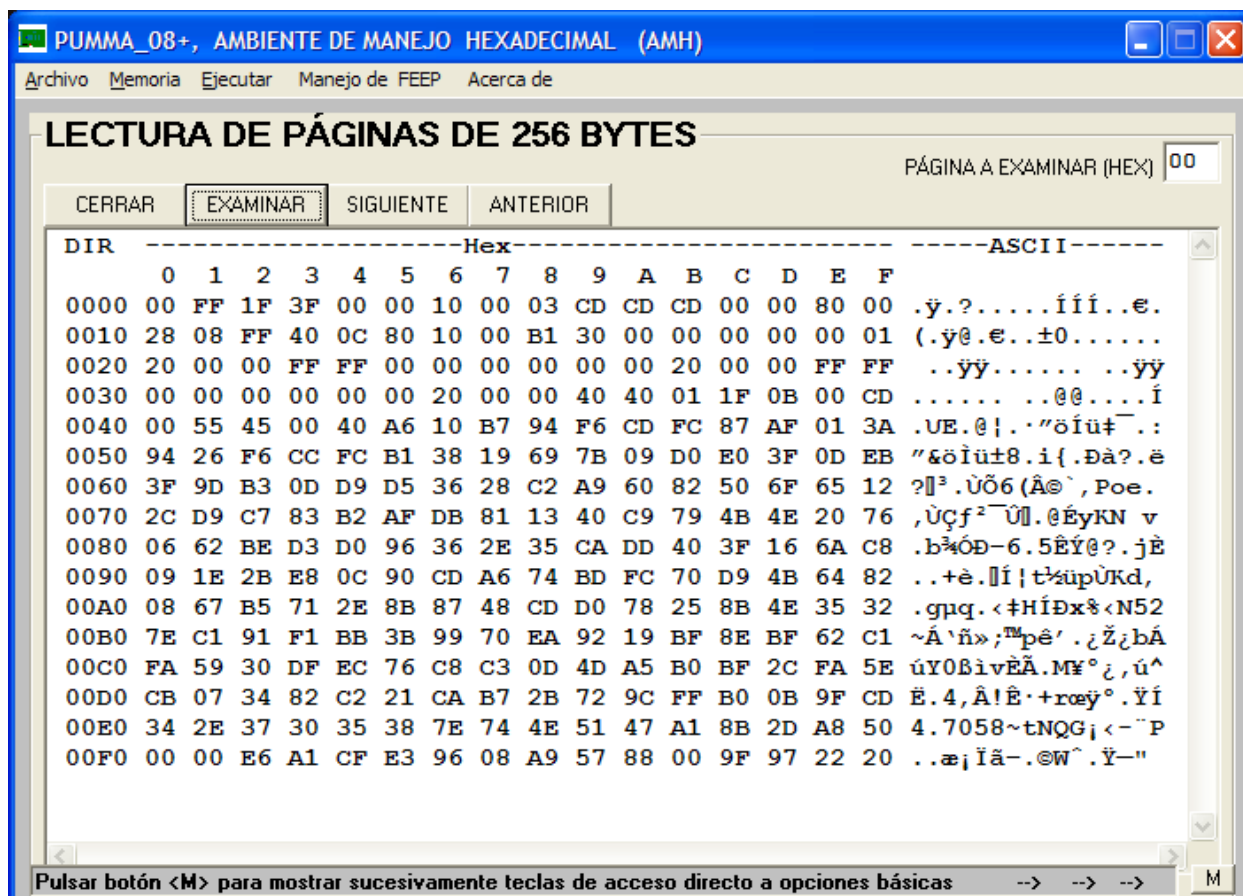


Figura 6.13. Página cero de memoria de la tarjeta MINICON_08A después que se ha ejecutado el programa de prueba pconvby5v.

Cabe señalar que las direcciones de RAM empleadas pudieron haber sido otras, las aquí empleadas se usaron para fines ilustrativos.

Ahora que se tiene la subrutina convby5v diseñada y probada, se puede proceder a el diseño del programa que nos ocupa en este ejemplo, para ello simplemente podría usarse el código del ejemplo 4 agregándose a éste lo propio para desplegar en el LCD el valor de la entrada analógica Ve aplicada a la entrada AD7 del CAD. El programa fuente en ensamblador es ejecutable desde FLASH y podría ser el siguiente:

```
* EJEMPLO 5
* CONVERSIÓN DEL CANAL AD7 CON RESULTADO MOSTRADO EN EL LCD CON
* CUATRO DÍGITOS FRACCIONARIOS
* Por: ANTONIO SALVÁ CALLEJA
* Enero de 2009
* ALMACENADO EN EL ARCHIVO LCDGP2.ASM

* DECLARACIONES EQU ASOCIADAS CON EL CAD
ADSCR      EQU    $3C
ADR        EQU    $3D
ADCLK      EQU    $3E
*****
* DECLARACIONES EQU ASOCIADAS CON SUBROUTINA CONVBY5V
PENT EQU $E0 ;DIRECCIÓN ORIGEN DE COLOCACIÓN DE CADENA DECIMAL
FR1 EQU $00E2 ;DIRECCIÓN DE COLOCACIÓN DE ASCII DE PRIMER.
*
CONTFR EQU $F0 ;DIRECCIÓN DE CONTADOR DE DÍGITOS FRACCIONARIOS.
```

```

NUMFR EQU $04 ;PREDETERMINACIÓN DE NÚMERO DE DÍGITOS
*
* FRACCIONARIOS.
APUNFRAC EQU $F1 ;DIRECCIÓN DE APUNTADOR DE PRÓXIMO DÍGITO
*
* FRACCIONARIO A COLOCAR.
* DECLARACIONES EQU ASOCIADAS CON VALORES ASCII DE CARACTERES NO NÚMERO
* USADOS.
ASCIIP EQU $2E ;CÓDIGO ASCII DEL CARÁCTER '.'.
ASCIIV EQU $56 ;CÓDIGO ASCII DEL CARÁCTER 'V'.
ASCIEMIN EQU $65 ;CÓDIGO ASCII DEL CARÁCTER 'e'.
ASCIIG EQU $3C ;CÓDIGO ASCII DEL CARÁCTER '='.

ORG $8000 ;EL PROGRAMA SE ORIGINA EN LA LOCALIDAD $8000 DE FLASH

* BLOQUE DE INICIALIZACIÓN
MOV #$60,ADCLK ;FRECUENCIA DE SEÑAL ADC CLOCK SERÁ 1 MHz.
$INCLUDE "E:\HC08\MDAM8A00.ASM" ;INCLUSIÓN DE SOFTWARE MANEJADOR DEL LCD.

*
* 0123456789A
* COLOCA LA CADENA 'Ve= V'.
LDA #$80
JSR ESCOM4
LDA #ASCIIV
JSR ESCDAT4
LDA #ASCIEMIN
JSR ESCDAT4
LDA #ASCIIG
JSR ESCDAT4

* FIN DE BLOQUE DE INICIALIZACIÓN

* BLOQUE DE ACCIONAMIENTO PRINCIPAL

INICONV: MOV #$07,ADSCR ;INICIA CONVERSIÓN DE CANAL AD7.
CHECOCO: BRCLR 7,ADSCR,CHECOCO ;ESPERA FIN DE CONVERSIÓN.
LDA ADR
BSR CONVBY5V ;GENERA REPRESENTACIÓN DECIMAL DE Ve.
LDA #$83
JSR ESCOM4
LDHX #FR1-2
ESCDIGITO: LDA ,X
JSR ESCDAT4
AIX #$01
CPHX #FR1+4
BNE ESCDIGITO
LDA #$20
JSR ESCDAT4 ;ESCRIBE ESPACIO FINAL ANTES DE CARACTER 'V'
BRA INICONV ;PASA A EFECTUAR OTRA CONVERSIÓN.

* FIN DE BLOQUE DE ACCIONAMIENTO PRINCIPAL

* INICIO DE BLOQUE DE SUBROUTINAS
*****
* SUBROUTINA CONVBY5V
* Por: Antonio Salvá Calleja.
* Enero de 2009.
* A partir del byte generado por el CAD, al completarse una
* conversión a digital, obtiene la representación en decimal
* como cadena de caracteres ASCII, correspondiente al valor Ve
* presente en la entrada analógica que corresponda.
*
* Antes de invocar:
* a<--byte generado por el CAD (bycad).
*

```

```

*Al retornar:
* (pent:pent+1:::pent+nfr+1)<--representación decimal de Ve.
* Donde nfr representa el número de dígitos fraccionarios
* predeterminado en sentencia equ previa.
* a<--valor del último residuo.
*****

CONVBY5V:      PSHH
                PSHX

                MOV #NUMFR,CONTFR ;INICIALIZA CONTFR
                LDHX #FR1
                STHX APUNFRAC ;INICIALIZA APUNTADOR DE FRACCIONES

                CLRH
                LDX #$33 ;DIVISOR 51
                DIV
                ADD #$30 ;A<-- ASCII DE PARTE ENTERA
                STA PENT
                PSHH
OTRODIGFRAC:   PULA ;A<---RESIDUO
                LDX #$0A
                MUL ;X:A<--RESIDUO X 10
                PSHX
                PULH ;H:A<---RESIDUO X 10
                LDX #$33
                DIV
                ADD #$30 ;A<--ASCII DE DÍGITO A LA DERECHA DEL PUNTO
                PSHH ;GUARDA EN STACK RESIDUO DE ÚLTIMA DIVISIÓN.
                LDHX APUNFRAC
                STA ,X ;COLOCA EN BUFFER EL ÚLTIMO DÍGITO FRACCIONARIO
;               ENCONTRADO.
                AIX #$01
                STHX APUNFRAC ;ACTUALIZA APUNTADOR DE DÍGITOS FRACCIONARIOS.

                DEC CONTFR
                BNE OTRODIGFRAC
                MOV #$2E,PENT+1 ;COLOCA PUNTO DECIMAL.
                PULA ;A<--ÚLTIMO RESIDUO

                PULX
                PULH
                RTS

```

7 COMPILADOR CRUZADO MINIBAS8A

El compilador cruzado MINIBAS8A genera código ejecutable en microcontroladores HC08 de Freescale, esto a partir de código fuente en lenguaje BASIC estándar. Quien haya programado en BASIC empleando QuickBasic de Microsoft se sentirá en casa. Esta versión del compilador requiere que las bibliotecas en ensamblador del mismo estén presentes en el chip destino donde va a correr el programa. A los dispositivos con el valor agregado constituido por las bibliotecas de MINIBAS8A se les ha denominado genéricamente Chipbas8_xx, donde la cadena ‘xx’ denota el miembro de la familia HC08 empleado. A la fecha de elaboración de este manual, se cuenta con dos tipos de dispositivos chipbas8, los cuales son: el chipbas8_gp32, basado en el MCU 68HC908GP32CP; y el chipbas8_gt60, basado en el MCU MC9S08GT60. Además de las bibliotecas mencionadas, los dispositivos chipbas8 contienen un monitor denominado NBCP8, esto permite su manejo mediante el software PUMMA_08+, cuyas facilidades se describen en el capítulo 3 de este manual. En las figuras 7.1 y 7.2 se muestran los mapas de memoria de los dispositivos chipbas8 aquí mencionados.

MAPA DE MEMORIA DEL CHIPBAS8_GP32	
Dir Hex	Uso
0000 003F	Zona 1 de registros de control y operación (RCO).
0040 023F	Memoria RAM (512 localidades).
8000 D7DB	Memoria Flash de usuario (22,492 localidades).
D7DC D7FF	Vectores de usuario.
D800 FDFF	Firmware NBCP8_BIBAS8.
FE00 FF7D	Zona 2 de registros de control y operación. ROM Monitor.
FFDC FFFF	Vectores del NBCP8.

Figura 7.1. Mapa de memoria del chipbas8_gp32

MAPA DE MEMORIA DEL CHIPBAS8_GT60	
Dir Hex	Uso
0000 007F	Zona 1 de registros de control y operación (RCO).
0080 107F	Memoria RAM (4,096 localidades).
1080 17FF	Zona 1 del firmware NBSP8_BIBAS8.
1800 182B	Zona 2 de registros de control y operación.
182C DFCB	Memoria Flash de usuario (51,104 localidades).
DFCC DFFF	Vectores de usuario.
E000 FFFD	Zona 2 del firmware NBSP8_BIBAS8.
FFFE FFFF	Vector de reset del NBSP8.

Figura 7.2. Mapa de memoria del chipbas8gt60.

En la figura 7.1 se aprecia que, para el chipbas8_gp32, el usuario cuenta con 22,492 localidades para colocar su programa; además, se aprecia una zona de vectores de usuario, ya que los vectores originales del MCU están en una zona protegida contra escritura. Una descripción de la funcionalidad de los vectores de usuario puede verse a partir de la sección 2.4 de este manual.

En la figura 7.2 se aprecia que, para el chipbas8_gt60, el usuario cuenta con 51,104 localidades para colocar su programa, se aprecia ahí que los vectores de usuario están en una zona diferente a la propia para este fin en el chipbas8_gp32; sin embargo, los conceptos asociados con los vectores de usuario explicados en el capítulo 2 son fácilmente extendibles a lo propio sobre estos en el chipbas8_gt60.

Si el usuario desea tener una vista a vuelo de pájaro sobre las facilidades presentes en el compilador MINIBAS8A y cómo está estructurado éste, puede revisar la referencia [5], donde aparece una descripción de las etapas que conforman el compilador, además de una introducción al concepto Chipbas8.

A continuación se describen las facilidades con que cuenta MINIBAS8A.

7.1 Constantes, variables, expresiones y operadores

MINIBAS8A contempla el uso de constantes y variables; además del uso de expresiones que contengan diversos operadores. A continuación se describe lo propio para cada uno de los elementos aquí mencionados.

7.1.1. Constantes

Las constantes pueden ser tanto numéricas como de tipo string, a estas últimas frecuentemente se les denomina como '*string explícito*'. Ejemplos de constantes de tipo string podrían ser:

“PERRO”

“\$345.45”

“El uso de MINIBAS8A”

Las constantes numéricas pueden ser números explícitos tanto positivos como negativos. MINIBAS8A contempla tres tipos de constantes numéricas los cuales se describen a continuación:

Constantes enteras, que son números enteros cuyo valor está comprendido entre -2147483648 y 2147483647. No se usa punto en la escritura de éstas.

Constantes reales, que son números que contienen punto y eventualmente notación exponencial, si no se usa esta última los números podrán contener hasta 7 dígitos. Ejemplos de estas constantes podrían ser:

-3.456

1.51e-2=.0151

23.45e10=23.4500000000

678.

Constantes Hex, que vienen siendo números enteros en notación hexadecimal anteceditas por el prefijo &h. Ejemplos de este tipo de constantes podrían ser:

&hAB=171

&h7a2c=31276

7.1.2. Variables

Las variables son cadenas de caracteres de hasta 19 elementos, el primero debe ser forzosamente una letra y cualquiera de los subsecuentes tendrá que ser letra o bien número. En un programa en BASIC, las variables son usadas para representar valores. El valor de una variable puede ser asignado de manera explícita en el programa, o bien, éste puede ser asignado como el resultado de cálculos que involucren a otras variables y constantes. Antes de que a una variable se le asigne un valor, éste es asumido como cero. MINIBAS8A contempla cinco tipos de variables que a continuación se describen:

Variables enteras signadas de un byte. Éstas se denominan como de *tipo byte*, y podrán tener un valor comprendido entre -128 y 127. MINIBAS8A emplea una localidad de memoria para el almacenaje de este tipo de variables.

Variables enteras signadas de dos bytes. Éstas se denominan como de *tipo integer*, y podrán tener un valor comprendido entre -32768 y 32767. MINIBAS8A emplea dos localidades de memoria para el almacenaje de este tipo de variables.

Variables enteras signadas de cuatro bytes. Éstas se denominan como de *tipo long*, y podrán tener un valor comprendido entre -2147483648 y 2147483647. MINIBAS8A emplea cuatro localidades de memoria para el almacenaje de este tipo de variables.

Variables reales de precisión sencilla. Éstas se denominan como de *tipo single*, y podrán tener un valor comprendido entre -1.7E38 y 1.7E38. MINIBAS8A emplea cuatro localidades de memoria para el almacenaje de este tipo de variables, lo cual hace bajo la norma IEEE 754.

Variables string. Éstas se denominan como de *tipo string*, y están asociadas con cadenas de hasta 32 caracteres. Para cada variable de tipo string MINIBAS8A asocia con el nombre de ésta tres localidades de memoria, donde almacena lo que se denomina como el descriptor de ésta; en las dos primeras localidades se tendrá la dirección donde inicia propiamente la colocación en memoria del string, en la tercera localidad se almacena la longitud de éste.

7.1.2.1. Declaración implícita de tipos asociados con variables mediante el uso de posfijos

Al igual que en el lenguaje BASIC estándar, si una variable no ha sido declarada en alguna forma, MINIBAS8A asume para ésta el tipo single (real de precisión sencilla). Si se desea que una variable sea de algún otro tipo diferente se puede emplear un posfijo al final de la cadena de caracteres que la definen. Los posfijos y tipos asociados son los siguientes:

Posfijo ‘~’, para declaración implícita del tipo **byte**.

Posfijo ‘%’, para declaración implícita del tipo **integer**.

Posfijo ‘&’, para declaración implícita del tipo **long**.

Posfijo ‘\$’, para declaración implícita del tipo **string**.

Por ejemplo, MINIBAS8A asumirá implícitamente que las variables: moly, mybyte~, contador%, contlargo& y minom\$; son respectivamente de tipo single, byte, integer, long y string.

7.1.2.2. Asignación de tipo de acuerdo con la primera letra de la variable

Al igual que en el BASIC clásico, MINIBAS8A contempla la asignación de tipo a una o varias variables usando la primera letra de éstas, como parte de esta definición. Para ello se emplean sentencias de tipo ‘defxxx’, donde la cadena xxx podrá ser ‘bte’, para asignación tipo byte; ‘int’, para asignación tipo integer; ‘lng’, para asignación tipo long; ‘sng’, para asignación tipo single y ‘str’, para asignación tipo string. La sintaxis asociada se describe a continuación:

defxxx <lista de letras > y/o <lista de intervalos de letras >

Una sentencia como la anterior, hará que todas las variables que empiecen con una letra dentro del conjunto definido por las listas que aparecen como argumento de la sentencia, sean del tipo

definido por la cadena xxx, de acuerdo con lo explicado en el párrafo anterior. A continuación se muestran varios ejemplos:

La sentencia:

```
defbte b, c, h, i-k
```

hará que todas las variables que inicien con las letras b, c, h, i, j y k; serán consideradas como de tipo byte.

La sentencia:

```
defsng b-e, q, z
```

hará que todas las variables cuya primera letra esté en el intervalo de la b a la e; o bien sea la letra q ó z; sean asumidas como de tipo single.

7.1.2.3. Asignación explícita de tipos a variables

MINIBAS8A contempla la posibilidad de que el usuario pueda declarar los tipos asociados con las diversas variables que se usen en un programa. La sintaxis asociada se muestra en forma genérica en seguida:

```
dim <lista de variables separadas por comas> as tipo
```

donde tipo podrá ser la palabra byte, integer, long, single, o bien string; esto de acuerdo con el tipo que se desea asociar con las variables que aparecen en la lista. A continuación se muestran algunos ejemplos:

Para declarar que las variables ertq, aps y apl sean de tipo real de precisión sencilla se podría usar la siguiente sentencia:

```
dim ertq, aps, apl as single
```

Para declarar que las variables minom, domi, num y zop sean de tipo string se podría usar la siguiente sentencia:

```
dim minom, domi, num, zop as string
```

7.1.3. Expresiones y operadores aritméticos, relacionales y booleanos

Una expresión es una combinación de variables y constantes relacionadas mediante operadores. Estos pueden ser: aritméticos, relacionales y booleanos.

Los operadores aritméticos manejados por MINIBAS8A son, en orden de precedencia los siguientes:

Tabla 7.1. Orden de precedencia de los operadores aritméticos

Operador	Operación	Ejemplo
\wedge	Potencia	x^y
-	Cambio de signo	-x
*	Multiplicación	$x*y$
/	División	x/y
\	División entera	$x \setminus y$
Mod	Módulo aritmético	$x \bmod y$
+	Adición	$x + y$
-	Resta	$x - y$

Por lo regular el resultado de la evaluación de una expresión se asigna a una variable, o bien, representa un valor usado como argumento de una estructura de programación. Esto último se explicará al detallarse lo propio acerca de las estructuras de programación de MINIBAS8A. A continuación se muestra una expresión aritmética cuyo valor ha de asignarse a la variable 'xr'.

$$xr = plop/polo + aq * plop + polo$$

Si las variables plop, polo y aq tienen respectivamente los valores: 3.5, 10. y 47.5, la valoración de la expresión a la derecha del operador de asignación (=) da como resultado 176.6 siendo este valor numérico el que se asignará a la variable xr.

Cuando se requiera que el orden de precedencia de las operaciones sea diferente al mostrado en la tabla 7.1 se requerirá el uso de paréntesis, por ejemplo, si se desea que la variable aq se multiplique por el resultado de la suma de las variables plop y polo esta adición debe efectuarse antes de la multiplicación siendo la expresión requerida la siguiente:

$$xr = plop/polo + aq * (plop + polo)$$

Si los valores de las variables implicadas son los mismos, entonces 641.6 será el valor que se asignará a la variable xr.

7.1.3.1. Operadores para división entera y modular

MINIBAS8A tiene facilidades para efectuar división entera y división modular, los operadores respectivamente son '\ ' y 'mod'. En el caso de que los operandos no sean enteros, éstos son redondeados a enteros tipo *integer*, si uno o ambos rebasa el intervalo [-32768, 32767], se envía a la consola el mensaje "*sobreflujo al convertir a tipo inferior*" y se detiene la ejecución. A continuación se presentan algunos ejemplos de este tipo de operaciones.

$$15 \setminus 6 = 2$$

$$234.7 \setminus 45.6 = 39$$

$$15 \bmod 6 = 3$$

$$234.7 \bmod 45.6 = 6$$

$$(15/6 = 2 \text{ con un residuo de } 3)$$

$$(235/46 = 5 \text{ con un residuo de } 6)$$

7.1.3.2 Operadores relacionales

Los operadores relacionales se usan para comparar dos valores. El resultado de la comparación es booleano, y puede ser usado para tomar decisiones en cuanto al flujo de ejecución del programa. Por lo regular las expresiones con operadores relacionales forman parte de la validación de condiciones booleanas, que son argumento de estructuras de programación como 'if' y 'while' que se explicaran más adelante.

En la Tabla 7.2 se muestran los operadores relacionales manejados por MINIBAS8A.

Tabla 7.2 Operadores relacionales soportados por MINIBAS8A

Operador	Prueba efectuada	Expresión ejemplo
=	Igualdad	$x = y$
< >	Diferente	$x < > y$
<	Menor que	$x < y$
>	Mayor que	$x > y$
< =	Menor o igual que	$x < = y$
> =	Mayor o igual que	$x > = y$

Cuando operadores aritméticos y relacionales se combinan en una expresión, la aritmética siempre se evalúa primero. Por ejemplo, la expresión:

$$a - b < (z1 + 4.)/c$$

es verdadera si el valor de la resta de a y b es menor que la suma de z1 y 4 dividida entre c.

7.1.3.3 Operadores booleanos

MINIBAS8A contempla operaciones booleanas entre operandos, los cuales pueden ser variables o números explícitos de tipo real o entero (tipos byte, integer y long). Cuando los operandos son enteros, la operación se efectúa bit a bit entre los dos operandos; en caso de que los operandos sean reales, éstos se convierten a entero para después efectuar bit a bit la operación booleana de que se trate en un momento dado, en este último caso los operandos deberán estar en el intervalo cerrado [-32768, 32767], si no es así, se envía a la consola el mensaje: *"Sobreflujo al convertir a tipo inferior"* y se detiene la ejecución. Los operadores booleanos en cuestión son: NOT, AND, OR, XOR, EQV e IMP. En cuanto a la operación NOT, esta se efectúa sobre un operando único para el cual se invierte el nivel lógico de cada uno de los bits que representan al número como entero. Por ejemplo, si la variable i% presente en la siguiente sentencia vale 10, entonces -11. será el valor asignado a la variable za.

$$za = \text{not } i\%$$

En la tabla 7.3 se muestran los resultados de las operaciones booleanas que implican dos operandos.

Tabla 7.3 Resultados de operaciones booleanas que implican dos operandos

X	Y	X AND Y	X OR Y	X XOR Y	X EQV Y	X IMP Y
0	0	0	0	0	1	1
0	1	0	1	1	0	1
1	0	0	1	1	0	0
1	1	1	1	0	1	1

A continuación se muestra un caso específico de una operación booleana de dos operandos.:

Supóngase que en las dos sentencias mostradas enseguida a este párrafo, las variables tipo byte $i\sim$ y $j\sim$ valen respectivamente 68 y 45, entonces 4 y 109 serán los valores que, respectivamente se asignarán a las variables za y zb .

$$Za = i\% \text{ and } j\%$$

$$Zb = i\% \text{ or } j\%$$

A continuación se ilustra bit a bit la operaciones binarias efectuadas, esto primero para el caso de la operación ‘and’.

$i\sim$	0	1	0	0	0	1	0	0	68
and	and	and	and	and	and	and	and	and	and
$j\sim$	0	0	1	0	1	1	0	1	45
za	0	0	0	0	0	1	0	0	4

Para la operación ‘or’ el detallado es el siguiente:

$i\sim$	0	1	0	0	0	1	0	0	68
or	or	or	or	or	or	or	or	or	or
$j\sim$	0	0	1	0	1	1	0	1	45
zb	0	1	1	0	1	1	0	1	109

7.2. Arreglos

MINIBAS8A soporta el uso de arreglos de una, dos y tres dimensiones, los índices implicados se asumen como variables de tipo *integer*, o bien, el resultado de la valoración de una expresión, si éste es un valor de tipo real MINIBAS8A lo transforma a entero. En caso de que bajo la ejecución de un programa, el valor de algún índice de un arreglo quede fuera del rango asociado con éste, se envía a la consola el mensaje “*índice fuera de rango*” y se detiene la ejecución del programa. El rango de valores que puede tomar un determinado índice va desde cero hasta un tope especificado en la declaración del arreglo. Cada variable asociada con los elementos de un arreglo puede ser: del tipo byte, integer, long, single y string.

La siguiente sentencia declara como reales de precisión sencilla a dos arreglos, uno de una dimensión y el otro de dos dimensiones.

dim vec(8), mimat(3,15) as single

La declaración anterior hace que el índice del arreglo vec(.) tenga un rango de valores que van de cero a ocho. En tanto que el primer índice del arreglo mimat(.,.) va de cero a tres y el segundo índice va de cero a 15.

Es fácil ver que el número de localidades de memoria que requieren respectivamente los arreglos vec(.) y mimat(.,.) son 36 y 256.

7.3. Colocación de comentarios

Para colocar comentarios en un programa fuente, éstos deberán estar precedidos por un apóstrofe ('). Los comentarios terminan con el fin del renglón en cuestión. A continuación se muestran algunos casos de colocación de comentarios que se explican por sí mismos.

' Aquí se muestra el uso del apóstrofe para delimitar comentarios

za= sqr (a+b) ' Se asigna a la variable za la raíz cuadrada de la suma de las variables a y b

zlop = 4.56 ' Se inicializa la variable zlop con el valor 4.56

Nótese que para hacer que todo un renglón sea comentario, basta con poner el apóstrofe al inicio de éste. Otra forma de hacer que todo un renglón sea comentario es poner la cadena 'rem' al inicio del renglón en cuestión; esto es parte de la gramática del BASIC tradicional y MINIBAS8A lo contempla. A continuación se muestra un ejemplo.

Rem Este es un renglón que es todo comentario.

7.4. Colocación de más de una sentencia en un renglón

Al igual que en el BASIC clásico, MINIBAS8A contempla la posibilidad de colocar en un renglón varias sentencias de programación; para ello basta con separarlas mediante el carácter ":". A continuación se ilustra esto mediante un ejemplo.

Las siguientes sentencias:

```
for i%=1 to 10
```

```
print "Hola"
```

```
next i%
```

podrían declararse en un solo renglón de la siguiente manera:

```
for i%=1 to 10: print "Hola": next i%
```

7.5 Uso de etiquetas

MINIBAS8A no contempla el manejo de programas escritos en el formato clásico del BASIC, donde cada línea del programa requería un número al inicio de ésta. En contraparte, se utilizan etiquetas para delimitar líneas del programa que pudieran ser referenciadas por el código contenido en otras líneas del programa. Las etiquetas deben ser colocadas en la primera columna de la pantalla de edición y el primer carácter de éstas deberá ser una letra y los siguientes podrán ser letras o números; el final de una etiqueta siempre deberá ser el carácter ":". El tamaño máximo de una etiqueta es 19 caracteres. Cabe señalar que a la cadena que se obtiene al eliminar el posfijo ":" se le conoce como prefijo de la etiqueta en cuestión.

Al igual que muchos lenguajes de alto nivel, el BASIC validado por MINIBAS8A contempla el uso de la transferencia de la ejecución de un programa a una línea de éste delimitada por una determinada etiqueta. Esto se hace empleando la palabra reservada **goto**, seguida por el prefijo de la etiqueta correspondiente. Para ilustrar esto a continuación se muestra un programa que al ejecutarse coloca una onda cuadrada en el bit 1 del puerto A del MCU.

gato ciclo ‘ Se regresa al inicio del lazo
‘ nótese el uso del prefijo de la etiqueta

Nunca deberá usarse la etiqueta completa después de las palabras reservadas goto y gosub (explicada más adelante). Por ejemplo, si la última línea del programa anterior se pone como:

goto ciclo:

Al compilarse el programa se generan errores que abortan la ejecución del compilador

Desde luego que MINIBAS8A contempla diversas estructuras de programación que al emplearse hacen que sea poco frecuente el uso de la sentencia **goto**. Más adelante se explicará el formato de éstas.

7.5.2 Tratamiento de programas con número de línea

En caso de tratar de compilar un programa escrito en el formato clásico de BASIC, el compilador arrojará diversos errores al procesar el código fuente asociado. Sin embargo, es posible hacer modificaciones al programa de modo que éste pueda ser compilado por MINIBAS8A. A continuación se muestra un ejemplo de cómo podría hacerse esto.

Sea el siguiente tramo de código escrito en BASIC tradicional:

```

10 input "app="; app
20 if app=0 then 50
30 print "app es diferente de cero"
40 goto 10
50 end

```

Para que el código anterior pueda ser compilado con MINIBAS8A, éste podría reescribirse de la siguiente forma:

```

e10: input "app="; app
    if app=0 then e50
    print "app es diferente de cero"
    goto e10
e50: end

```

En resumen, puede apreciarse que una forma de hacer la modificación consiste en anteponer un carácter no número en todos los números que estén inmediatamente después de las palabras reservadas: *goto*, *gosub* y *then*. En el ejemplo se usó el carácter “e”. Después de lo anterior, se deberá anteponer en cada uno de los números de línea implicados el mismo carácter, y además colocar al final de estos el delimitador “:”, de este modo se genera una etiqueta válida. Finalmente, es recomendable eliminar todos los números de línea que no estén asociados con las palabras reservadas: *goto*, *gosub* y *then*.

7.6. Sentencia input

En la mayoría de los programas se requiere que el usuario proporcione el valor explícito de diversas variables de uso en estos. Para ello se usa la sentencia *input*. La sintaxis genérica para ésta es la siguiente:

```
input [;] ["mensaje auxiliar";] lista de variables
```

En la sintaxis anterior los argumentos delimitados por corchetes son opcionales. A continuación se explica a detalle las formas de uso de la sentencia *input*.

El **mensaje auxiliar** es un string explícito, que se imprime en la consola antes del *prompt* que indica al usuario que introduzca el valor de la variable que se lee en un momento dado.

La **lista de variables** es un enlistado de las variables implicadas separadas por comas. Para éstas, el valor por asignarles ha de ser leído desde la consola. Dichas variables pueden ser de tipo numéricas, o bien, de tipo string.

Al ejecutarse el código asociado con una sentencia *input*, en la consola aparece un *prompt* precedido por el carácter “?” indicando al usuario que debe introducir el valor por asignar a la variable en cuestión, para terminar esta acción con la opresión de la tecla *return*. La lista de variables puede contener desde una, hasta el número que el usuario considere conveniente. Lo más frecuente y práctico, es que en cada sentencia *input* del programa esté implicada únicamente la lectura y asignación de valor para una sola variable; sin embargo, para la lectura y asignación de valores para más de una variable puede usarse una sola sentencia *input*.

Para suprimir la aparición del carácter “?”, se deberá usar el carácter “,” en lugar del carácter “;” después del string explícito que denota el mensaje auxiliar.

Colocar el carácter “,” inmediatamente después de la cadena *input*, hará que cuando el usuario oprima la tecla *return*, no se devuelvan los caracteres de control *return* y *line feed*. Esto, desde luego hará que no exista un salto de renglón en la consola después de la introducción de la variable.

En caso de que el usuario introduzca un valor por asignar a una variable que no sea coherente con el tipo de ésta, se desplegará en la consola el mensaje: “*Entrada inválida, repetir ésta*”, para inmediatamente después presentar un *prompt*,. para que el usuario repita la acción introduciendo correctamente el valor en cuestión. En el BASIC clásico se emite el mensaje en inglés “*redo from start*”

Cabe señalar aquí que para MINIBAS8A, la consola es simplemente un emulador de terminal serie conectado al MCU mediante el puerto serie de éste, operando a 9600 bps. El manejador PUMMA08+ cuenta con un emulador de terminal básico que puede usarse para validar la consola de interfaz. También puede usarse para este fin, entre otros, el programa hyperterminal contenido en WINDOWS. Si el programa del usuario no contiene sentencias *input* o *print*, la consola no es necesaria.

A continuación se muestran algunos ejemplos.

Al ejecutarse la sentencia:

```
input aq
```

Aparecerá en la consola el *prompt*, para que el usuario introduzca el valor de la variable aq; sin embargo, no se apreciará ninguna indicación acerca de a cual variable se le asignará el valor introducido. Para que se aprecie en la consola qué variable es la implicada, se requiere emplear la cadena auxiliar. Esto podría ser como se muestra a continuación:

```
input “aq=”; aq
```

Al ejecutarse la sentencia anterior, en la consola se desplegará la cadena auxiliar precediendo al *prompt* que solicita al usuario la introducción del valor por asignar a la variable aq. Si se deseara que no apareciera el carácter “?” antes del *prompt*, simplemente se cambia el carácter “;” por el carácter “,” como se muestra a continuación:

```
input “aq=”, aq
```

En los ejemplos anteriores, después de que el usuario oprime la tecla *return*, se genera un salto de renglón en la consola, si por alguna causa no se desea que esto suceda, simplemente se coloca el carácter “;” justo al finalizar la cadena *input*, tal como se explicó en párrafos anteriores. Para ilustrar esta idea a continuación se muestra un ejemplo.

Supóngase que se desea leer desde la consola el valor de dos variables denominadas aq y plop. Se requiere que esto sea en un renglón y usando sendas sentencias *input*. Una forma de hacer esto es mediante las siguientes dos líneas de código BASIC:

```
input; “aq=”; aq
```

```
input “ plop=”; plop
```

Si se deseara que para el ejemplo anterior se empleara solo una sentencia *input*, una manera de hacer esto podría ser mediante la siguiente sentencia:

```
input; “Dar los valores de aq y plop”; aq, plop
```

7.7. Sentencia print

Para desplegar resultados en la consola se usa la sentencia *print*, la sintaxis genérica para ésta se muestra a continuación:

```
print [lista de expresiones] [; ó ,]
```

En la sintaxis anterior los argumentos delimitados por corchetes son opcionales. A continuación se explica a detalle las formas de uso de la sentencia *print*.

La **lista de expresiones** es un enlistado de expresiones de tipo numérico, o bien, de tipo string. Como es sabido, éstas pueden ser desde un número explícito hasta una serie de variables y/o números explícitos afectados por diversos operadores, véase la sección 7.1.3.

Si la lista de expresiones se omite, en la consola se dará un salto de renglón. Si la lista de expresiones se incluye, los valores de éstas se despliegan en la consola. Si al final de la lista de expresiones no se coloca ya sea el carácter “;” o bien, el carácter “,”, después de desplegarse el valor de la última expresión se producirá un salto de renglón; si por el contrario, se coloca al final de la lista de expresiones alguno de los dos caracteres mencionados, no se dará un salto de renglón después de la impresión de la última expresión.

Si se usa el carácter “,” para separar dos ítems de la lista de expresiones, se usarán 14 posiciones para imprimir al primero; siendo el segundo colocado inmediatamente después de la zona de impresión del primero que ocupa 14 posiciones de impresión. Por ejemplo, si el resultado de la primera expresión por desplegar es digamos 27.5 (4 caracteres), se imprimirá este valor seguido por 10 espacios, después de éstos se desplegará el valor que corresponda a la siguiente expresión. Por otra parte, si se emplea el carácter “;” para separar dos ítems de la lista de expresiones, el resultado de la primera se desplegará terminado con únicamente un espacio, desplegándose el valor de la siguiente expresión después del espacio antes mencionado. A continuación se muestra un ejemplo ilustrativo del uso de la sentencia *print*.

Supóngase el siguiente tramo de código BASIC compilable por MINIBAS8A.

```
a=1.
b=2.5
c=4.5

print "a=";a
print "b=";b
print "c=";c

print

print "12345678901234"
print sqr(a+b),a,sqr(b),(a+b+c)/3., "Mi perro querido"

print sqr(a+b);a;sqr(b);(a+b+c)/3.; "Mi perro querido"
```

Al ejecutarse éste, en la consola se desplegará lo mostrado en la figura 7.3

```

EMULADOR DE TERMINAL
cls
a= 1.0
b= 2.5
c= 4.5
12345678901234
1.8708286 1.0 1.5811388 2.6666667 Mi perro querido
1.8708286 1.0 1.5811388 2.6666667 Mi perro querido

```

Figura 7.3. Despliegue en la consola al ejecutarse el tramo de código mostrado arriba

Nótese que cuando se usa como separador de expresiones el carácter “,”, la longitud de cada campo de impresión es de 14 posiciones. Por otro lado si se usa como separador de expresiones el carácter “;”, la longitud de cada campo de impresión es la que proceda, de acuerdo con el resultado de la expresión que corresponda, con un espacio adicional.

7.8. Incrustación de código en ensamblador

MINIBAS8A contempla la posibilidad de incrustar código en lenguaje ensamblador dentro de un programa en BASIC, para ello basta con colocar respectivamente entre las palabras reservadas *iniens* y *finens* el código en lenguaje ensamblador que se requiera incrustar en el programa en un momento dado. **Es importante señalar que en las líneas que contengan las palabras reservadas *iniens* y *finens*, no debe haber etiquetas, si esto sucede, se pueden generar errores diversos en el código en ensamblador generado por el compilador.** A continuación se muestra un ejemplo de un tramo de código fuente que contiene a su vez código en lenguaje ensamblador incrustado.

```

input "by~="; by~

iniens

porta equ $00
ddra equ $04

mov #$ff, ddra 'El puerto A es salida

lda by~
sta porta
finens

```

Al ejecutarse el código anterior, se lee de la consola la variable byte *by~* para luego colocar ésta en el puerto A del MCU.

7.9. Variables especiales de usuario

MINIBAS8A contempla la posibilidad de que el usuario defina variables de tipo numérico, para las cuales la dirección de colocación de éstas en la memoria es propuesta por el usuario. A éstas se les denomina como “*variables especiales de usuario*”. Más adelante se detalla como se deben declarar estas variables para cada uno de los cuatro casos posibles de variables numéricas contempladas por MINIBAS8A.

Es responsabilidad del programador, el que las direcciones que asigne a variables especiales, no coincidan con las propias correspondientes a variables normales que asigna MINIBAS8A sin la intervención del usuario. Es fácil identificar tales direcciones en las sentencias “equ” presentes en el programa en ensamblador asociado con el programa fuente en BASIC original. Tal programa en ensamblador está contenido en el archivo NP.CUS que genera el compilador, siendo NP.B el nombre del archivo que contiene al programa fuente original.

Es importante señalar que las declaraciones asociadas con variables especiales de usuario, siempre deben ser colocadas al inicio del programa que las use, en otro caso pueden generarse errores en el código en ensamblador asociado.

7.9.1 Declaración de variables especiales de usuario de tipo byte

Para declarar variables especiales de tipo byte se deberá usar la palabra reservada **defvarbptr** seguida por la cadena que denota a la variable y la especificación de la dirección asignada a ésta por el usuario. Por ejemplo, si se desea que a la cadena “porta” se le asigne la dirección &h0, que es la propia dirección del puerto A del MCU, la declaración correspondiente podría ser:

```
defvarbptr porta &h0
```

De esta forma, para copiar el contenido del puerto A en una variable de tipo byte, denominada digamos como mibyte~, simplemente podría usarse la siguiente asignación en BASIC:

```
mibyte~ = porta
```

De no contar con el concepto de variables especiales de usuario la asignación anterior sería más complicada, una forma de hacerla sería incrustar el siguiente código en ensamblador:

```
iniens
porta equ $00
lda porta
sta mibyte~
finens
```

En el ejemplo anterior se aprecia la bondad de las variables especiales de usuario. **Cabe señalar que el uso más frecuente de este concepto es el asignar variables especiales de usuario de tipo byte a los registros propios del MCU, asociándose con éstas las direcciones de éstos; facilitándose de esta manera el intercambio de información con dichos registros, lo cual como es sabido, constituye una buena parte de las acciones a efectuarse al realizarse una determinada aplicación validada por el microcontrolador.**

7.9.2 Declaración de variables especiales de usuario de tipo integer

Para declarar variables especiales de tipo integer se deberá usar la palabra reservada **defvariptr** seguida por la cadena que denota a la variable y la especificación de la dirección asignada a ésta

por el usuario. Por ejemplo, si se desea que a la cadena “mientero” se le asigne la dirección &h100, la declaración correspondiente podría ser:

```
defvariptr mientero &h100
```

7.9.3 Declaración de variables especiales de usuario de tipo long

Para declarar variables especiales de tipo long se deberá usar la palabra reservada **defvarlptr** seguida por la cadena que denota a la variable y la especificación de la dirección asignada a ésta por el usuario. Por ejemplo, si se desea que a la cadena “numgrande” se le asigne la dirección &h103, la declaración correspondiente podría ser:

```
defvarlptr numgrande &h103
```

7.9.4 Declaración de variables especiales de usuario de tipo single

Para declarar variables especiales de tipo single (reales de precisión sencilla), se deberá usar la palabra reservada **defvarsptr** seguida por la cadena que denota a la variable y la especificación de la dirección asignada a ésta por el usuario. Por ejemplo, si se desea que a la cadena “mireal” se le asigne la dirección &h106, la declaración correspondiente podría ser:

```
defvarsptr mireal &h106
```

7.10 Colocación en memoria de constantes de tipo numérico

MINIBAS8A contempla la posibilidad de que el usuario coloque en memoria, constantes tanto de tipo entero como de tipo real de precisión sencilla, para declarar esto se deberá, según el tipo de variable, emplear una de las siguientes palabras reservadas: **datab**, **dataw**, **datal** o **datas**; seguida por la dirección de colocación deseada y la lista de constantes separadas por comas.

Es responsabilidad del programador, el que las direcciones de colocación de las constantes no colisionen con las propias de colocación del código del programa en la memoria. Una forma de checar el intervalo de direcciones que ocupará el programa en la memoria es ver los archivos NP.LST, o NP.S19; donde NP.B es el nombre del archivo que contiene el programa fuente original.

A continuación se describe para cada tipo de constante numérica la forma de declarar la colocación de éstas en memoria.

7.10.1 Declaración para colocar en memoria constantes de tipo byte

Para colocar en memoria constantes enteras cuyo valor esté comprendido entre -128 y 127 se emplea la siguiente declaración:

```
datab dir <lista de constantes separadas por comas>
```

Donde dir, representa la dirección expresada en forma explícita. Tanto la dirección como las constantes podrán expresarse en hexadecimal, o bien, en decimal.

Por ejemplo, supóngase que se desea colocar las constantes de tipo byte: -2, 45, y &hf a partir de la dirección \$a000 de memoria; esto podría hacerse mediante la siguiente declaración:

```
datab &ha000 -2, 45, &hf
```

7.10.2 Declaración para colocar en memoria constantes de tipo integer

Para colocar en memoria constantes enteras cuyo valor esté comprendido entre -32768 y 32767 se emplea la siguiente declaración:

```
dataw dir <lista de constantes separadas por comas>
```

Donde dir, representa la dirección expresada en forma explícita. Tanto la dirección como las constantes podrán expresarse en hexadecimal, o bien, en decimal.

Por ejemplo, supóngase que se desea colocar las constantes de tipo integer: -2, 4500, y &hf a partir de la dirección \$a080 de memoria. Esto podría hacerse mediante la siguiente declaración:

```
dataw &ha080 -2, 4500, &hf
```

Además, en la lista de constantes pueden incluirse prefijos de etiqueta, esto es de gran utilidad al declarar la colocación de los vectores de RESET y/o de interrupción que pudiera emplear un determinado programa. Por ejemplo, supóngase que en un determinado programa a ejecutarse en el chipbas8_gp32, se usa una interrupción disparada por el flanco de bajada del pin IRQ del MCU. Sabiendo que las direcciones de colocación del vector de usuario asociado son: &hd7fa y &hd7fb y que el inicio de la rutina de servicio asociada esta denotado con la etiqueta **rutserv**. Una forma de declarar la colocación del vector de usuario, sería emplear la siguiente sentencia al final del programa.

```
dataw &hd7fa rutserv
```

7.10.3 Declaración para colocar en memoria constantes de tipo long

Para colocar en memoria constantes enteras cuyo valor esté comprendido entre -2147483648 y 2147483647 se emplea la siguiente declaración:

```
datal dir <lista de constantes separadas por comas>
```

Donde dir, representa la dirección expresada en forma explícita. Tanto la dirección como las constantes podrán expresarse en hexadecimal, o bien, en decimal.

Por ejemplo, supóngase que se desea colocar las constantes de tipo long: -2, 459876, y &habc89f a partir de la dirección \$a0c0 de memoria; esto podría hacerse mediante la siguiente declaración:

```
datal &ha0c0 -2, 459876, &habc89f
```

7.10.4 Declaración para colocar en memoria constantes de tipo single

Para colocar en memoria constantes reales de precisión sencilla (7 dígitos de resolución) se emplea la siguiente declaración:

```
datas dir <lista de constantes separadas por comas>
```

Donde dir, representa la dirección expresada en forma explícita, tanto en hexadecimal, o bien, en decimal. Las constantes deberán expresarse como números reales en decimal.

Por ejemplo, supóngase que se desea colocar las constantes de tipo single: -2.5, 4.5e-2 y 28.56745 a partir de la dirección \$a0e0 de memoria; esto podría hacerse mediante la siguiente declaración:

```
datas &ha0e0 -2.5, 4.5e-2, 28.56745
```

7.11 Uso de subrutinas

Al igual que el BASIC tradicional, el validado por MINIBAS8A contempla el manejo de subrutinas, las cuales simplemente son tramos de código utilizable en distintas partes dentro de un determinado programa, por lo regular el inicio de éstas se denota con una etiqueta definida por el programador, delimitándose el final de las mismas con la palabra reservada **return**. Para invocar una subrutina basta con colocar la palabra reservada **gosub** seguida por el prefijo de la etiqueta que denota el inicio de la subrutina en cuestión.

Como ejemplo del uso de subrutinas, a continuación se muestra un programa que coloca una onda cuadrada de un segundo de periodo y un ciclo de trabajo del 50% en el bit 7 del puerto A del MCU. Para esto se emplea una subrutina que genera un retardo de 250 ms, ésta está construida incrustando código en ensamblador que genera retardos en función del valor de un valor denominado Xr que se carga en el par h:x, véase la sección 6.2.

```
defvarbptr pta &h0
defvarbptr ddra &h4

        ddra=&h80
ciclo:
        pta = &h80
        gosub ret250ms
        pta=&h0
        gosub ret250ms
        goto ciclo
ret250ms:
        iniens
        pshh
        pshx
        ldhx #$c34e
vuelta:
        nop
        nop
        aix  #$ff
        cphx #$0000
        bne vuelta
        pulx
        finens
        return
```

7.12 Rutinas de servicio de interrupción

Para colocar el código asociado con una rutina de servicio de interrupción, se deberá delimitar el inicio de ésta con una etiqueta definida por el programador, el final de la misma se denotará con la palabra reservada **retint**. Si el código usa sentencias de BASIC es recomendable colocar al inicio de la subrutina la palabra reservada **glip**, lo cual hace que se guarden en la pila las localidades de intercambio que usan las rutinas de biblioteca, al final de la subrutina

deberá colocarse la palabra reservada *relip*, lo cual hace que se recuperen de la pila las localidades de intercambio antes mencionadas. Si el código BASIC hace cálculos aritméticos la colocación de las palabras reservadas antes mencionadas es imprescindible.

A continuación se muestra un programa ejemplo que usa la interrupción IRQ del MCU. Al ejecutarse el programa se mostrará una cuenta ascendente de 1 a 100 en el puerto A, con una cadencia de ¼ de segundo. Al detectarse un flanco de bajada en el pin IRQ se deberá ver en el puerto a la parte entera de la raíz cuadrada del valor de la cuenta en ese momento, esto por un periodo de tiempo de un segundo.

```
defvarbptr pta &h0
defvarbptr ddra &h4
```

ciclo:

```
for i~ = 1 to 100
pta = i~
gosub ret250ms
next i~
goto ciclo
```

servirq:

```
glip ‘guarda en pila localidades de intercambio
pta=sqr(i~)
gosub ret250ms
gosub ret250ms
gosub ret250ms
gosub ret250ms
relip ‘recupera de pila localidades de intercambio
retint
```

ret250ms:

```
iniens
pshh
pshx
ldhx #$c34e
vuelta: nop
nop
aix #$ff
cphx #$0000
bne vuelta
pulx
finens
return
```

```
dataw &hd7fa servirq ‘Coloca vector de usuario para la interrupción IRQ
```

7.13 Subrutina de biblioteca lee#car

lee#car es una rutina en ensamblador propia de la biblioteca de MINIBAS8A cuya funcionalidad es la siguiente:

Al invocarse se estará en un lazo de espera de la llegada de un byte al puerto serie del MCU. Al recibirse éste, se sale del lazo y se retorna con el valor del byte recibido cargado en el

acumulador “A” del MCU. Considerando que la consola es un emulador de terminal, el siguiente tramo de código hará que la ejecución del programa esté en un lazo de espera del cual se saldrá cuando el usuario oprima cualquier tecla de la consola.

```
iniens
jsr lee#car
finens
```

Nótese que para el accionamiento validado por el código anterior, es irrelevante que byte se recibe (que tecla se oprime en la consola). **Cuando se carga y ejecuta un programa en BASIC que contenga sentencias input y/o print, empleando para ello al manejador pumma08+ u otro; es necesario colocar al inicio del programa el código mostrada arriba; esto hace que al iniciarse la ejecución del programa, se pase a una espera de la opresión de una tecla en la consola, esto mientras se abre el software de emulación de terminal empleado para validar la consola, y cualquier acción relacionada con las sentencias input y print presentes no pasará desapercibida. En caso de que el programa se ejecute de manera autónoma la colocación al inicio del programa de la invocación de la rutina lee#car no es necesaria.**

Otro ejemplo de uso de la rutina lee#car es la siguiente subrutina que se denomina como migetchar, su funcionalidad es la siguiente:

Al invocarse se pasa a la espera de la opresión de una tecla en la consola, al efectuarse esto se retorna de la subrutina con el valor del byte recibido en la variable tipo byte micar~. El código de ésta se muestra a continuación:

‘ Subrutina migetchar
‘ Al invocarse se pasa a un lazo de espera de la
‘ opresión de una tecla en la consola.
‘ Al retornar, el código ASCII asociado con la
‘ tecla oprimida estará en la variable micar~
migetchar:

```
micar~ = 0
```

```
iniens
jsr lee#car
sta micar~
finens
```

```
return
```

7.14 Subrutina de biblioteca pon#car

Esta subrutina de la biblioteca de MINIBAS8A coloca en la pantalla de la consola un caracter en la siguiente posición disponible en ésta. Antes de invocarse el ASCII del carácter a desplegar deberá precargarse en el acumulador A del MCU; al retornar, ya se habrá desplegado el carácter en cuestión. Como ejemplo de uso de la subrutina pon#car, a continuación se muestra una subrutina denominada miputchar cuya funcionalidad es la siguiente: Antes de invocar se deberá asignar a la variable miconcar~ el valor del código ASCII asociado con el caracter por desplegar, al retornar, ya se habrá desplegado en la consola el caracter implicado. El código de la subrutina miputchar es el siguiente:

- ‘ Subrutina miputchar
- ‘ Antes de invocar:
- ‘ mponcar~ \leftarrow ASCII de carácter a desplegar
- ‘ Al retornar ya se habrá desplegado el caracter

miputchar:

```

    iniens
    lda mponcar~
    jsr pon3car
    finens
    return

```

7.15 Sentencia cls

Al ejecutarse una sentencia cls, simplemente se limpia la pantalla de la consola. De hecho físicamente el MCU envía el carácter de control *form feed* a ésta. Como ejemplo, a continuación se muestra un programa que al ejecutarse, pedirá al usuario el valor de dos variables reales denominadas a1 y a2, a continuación se desplegará en la consola el valor de la suma de las dos variables leídas; después de esto, se pedirá al usuario oprimir cualquier tecla inmediato a lo cual se limpiará la pantalla de la consola para repetir el proceso.

```

    iniens
    jsr lee#car
    finens

```

otralec:

```

    input " a1=";a1
    input " a2=";a2

    z1= a1+a2
    print "a1+a2=";z1

    print "Oprimir cualquier tecla"

    iniens
    jsr lee#car
    finens
    cls
    goto otralec

```

7.16 Sentencia end

Como en el BASIC tradicional, esta sentencia se usa para finalizar la ejecución de un programa. Físicamente MINIBAS8A genera un salto a la línea final del programa en ensamblador asociado, la cual es la siguiente:

```

fin#codus:    bra fin#codus

```

Cabe señalar que si el programa usa interrupciones, el accionamiento de estas no se detendrá, ya que simplemente éstas interrumpirían el lazo infinito anterior; por lo tanto, si un programa usa

interrupciones, éste se deberá finalizar con una deshabilitación global de éstas seguida por la sentencia end. Esto podría hacerse de la siguiente manera:

```
      iniens
      sei
      finens

      end
```

7.17 Estructuras de programación

Como en todo lenguaje de alto nivel, el BASIC validado por MINIBAS8A cuenta con estructuras de programación que pueden alterar la secuencia de ejecución de un programa. Algunas de éstas son lazos con código que se ejecuta cíclicamente mientras una determinada condición sea válida. Otras seleccionan tramos de código a ejecutarse si una determinada condición es válida; otras son ciclos donde parte del argumento de la estructura contiene información acerca de cuantas veces se ejecutara el código contenido dentro del ciclo. A continuación se describen cada una de las estructuras de programación con las que cuenta el BASIC validado por MINIBAS8A.

7.17.1 Estructura “if” básica

En su forma más simple esta estructura ejecuta o no un tramo de código presente dentro de ésta, dependiendo del valor booleano de una condición. Su formato es el siguiente:

```
      if condición then
      código a ejecutarse si
      la condición es válida.
      endif
```

En general, una condición podrá expresarse en alguna de las siguientes maneras:

- Número explícito. La condición es inválida (false) si dicho número es cero, en otro caso ésta será válida (true).
- Variable numérica. La condición es inválida (false) si el valor de la variable es cero, en otro caso, ésta será válida (true).
- Variable de tipo string. La condición será inválida (false) si el string contenido en la variable es nulo, en otro caso ésta será válida (true).
- Expresión aritmética. La condición será inválida (false) si la valoración de la expresión es cero, en otro caso ésta será válida.
- Expresión relacional. En este caso la condición incluye un operador relacional (=, <>, >, <, >=, <=) flanqueado por expresiones aritméticas que no contengan operadores booleanos. La condición será inválida si los valores de las expresiones aritméticas presentes a la izquierda y derecha del operador relacional implicado contradicen lo expresado por éste, en otro caso la condición será válida.
- Combinación booleana de expresiones aritméticas y/o relacionales. En este caso la validez o invalidez de la condición estará dada por el resultado de la operación booleana implicada.

En seguida se muestra un ejemplo:

```
if a1 > 0. then  
  print "La variable a1 es positiva"  
endif
```

7.17.2 Estructura “if” con alternativa else

Una variante de la estructura “if” se obtiene cuando dentro de ésta se usa la palabra reservada *else* para delimitar el código que se desea ejecutar en caso de que la condición especificada sea inválida, su formato genérico es el siguiente;

```
if condición then  
  código a ejecutarse si  
  la condición es válida.  
else  
  código a ejecutarse si  
  la condición es inválida.  
endif
```

A continuación se muestra un ejemplo de uso de una estructura if con sentencia else presente en ésta.

```
if a1 > 0. then  
  print "La variable a1 es positiva"  
else  
  print "La variable a1 es negativa o cero"  
endif
```

7.17.3 Estructura “if” con alternativas elseif

Otra forma de la estructura “if” emplea la palabra reservada *elseif* para definir más tramos de código a ejecutarse cuando sendas condiciones sean válidas, su forma genérica es como sigue:

```
if condición then  
  código a ejecutarse si  
  la condición es válida.  
  
elseif condición 1 then  
  código a ejecutarse si  
  la condición1 es válida.  
  
elseif condición 2 then  
  código a ejecutarse si  
  la condición2 es válida.  
  .  
  .  
elseif condición m then  
  código a ejecutarse si  
  la condición m es válida.  
  
endif
```

A continuación se muestra un ejemplo:

```
if  $x1 < -3$ . and  $x1 < 3$ . then  
  print “x1 está en el intervalo abierto I0”  
  
elseif  $x1 < 5.5$  and  $x1 > 4$ . then  
  print “x1 está en el intervalo abierto I1”  
  
elseif  $x1 < 25.67$  and  $x1 > 10$ . then  
  print “x1 está en el intervalo abierto I2”  
  
endif
```

7.17.4 Estructura “if” con alternativas elseif y else

Otra forma de la estructura “if” se obtiene al agregar a la estructura expuesta en la sección 7.17.3 la alternativa else; su forma genérica se muestra a continuación:

```
if condición then  
  código a ejecutarse si  
  la condición es válida.  
  
elseif condición 1 then  
  código a ejecutarse si  
  la condición1 es válida.  
  
elseif condición 2 then  
  código a ejecutarse si  
  la condición2 es válida.  
  
  .  
  .  
  
elseif condición m then  
  código a ejecutarse si  
  la condición m es válida.  
  
else  
  código a ejecutarse si ninguna  
  de las condiciones presentes es válida.  
  
endif
```

A continuación se muestra un ejemplo:

```
if x1<-3. and x1 < 3. then
print "x1 está en el intervalo abierto I0"

elseif x1<5.5 and x1 > 4. then
print "x1 está en el intervalo abierto I1"

elseif x1<25.67 and x1 > 10.. then
print "x1 está en el intervalo abierto I2"

else
print " x1 no pertenece a los intervalos I0, I1 e I2"

endif
```

7.17.5 Estructura "if" de una sola línea

MINIBAS8A contempla una estructura "if" que se escribe empleando una sola línea en el programa fuente, su formato es el siguiente:

```
if condición then sentencia de BASIC
```

Si la condición es válida, se ejecutaría la sentencia de BASIC colocada a la derecha de la palabra reservada **then**, en otro caso ésta no se ejecutaría. La sentencia de BASIC podrá tener alguna de las siguientes formas:

- Prefijo de etiqueta. Lo cual originará, en el caso de que la condición sea válida, que se de un salto a la etiqueta correspondiente.
- Goto prefijo de etiqueta. Lo cual originará, en el caso de que la condición sea válida, que se de un salto a la etiqueta correspondiente.
- Gosub prefijo de etiqueta. Lo cual originará, en el caso de que la condición sea válida, que se invoque la subrutina asociada que está delimitada por la etiqueta asociada.
- Asignación a una variable del resultado de una expresión aritmética.
- La sentencia cls.
- La sentencia end

A continuación se muestra un tramo de código BASIC que usa el "if" en diversas oportunidades.

leervara:

```
input "a=";a
if a=0. then leervara

input "b="; b
if b<=0. then end

if b>0. then z1=a+b/(a-b)
```

7.17.6 Estructura while

Esta estructura contempla la ejecución de código delimitado en su inicio y fin respectivamente por las palabras reservadas *while* y *wend*, mientras sea válida una condición, que es argumento de la estructura, se repetirá la ejecución del código mencionado; una vez que la condición no sea válida, se sale del ciclo y se pasa a la posición inmediata siguiente a la sentencia *wend*.

La forma genérica de esta estructura es la siguiente:

```
While condición  
  
    código a ejecutarse  
    repetitivamente mientras  
    la condición sea válida.  
  
wend
```

A continuación se muestran ejemplos de uso de la estructura while.

7.17.6.1 Ejemplo 1 de estructura while

Se muestra aquí un programa que coloca una onda cuadrada de .5 segs de periodo en b1t 1 del puerto A del MCU, esto de manera indefinida.

```
defvarbptr pta 0  
defvarbptr ddra 4  
  
ddra = 2 ' pta1 es salida  
  
while 1  
  
    pta = 0  
    gosub ret250ms  
    pta = 2  
    gosub ret250ms  
  
wend  
  
ret250ms:  
    iniens  
    pshh  
    pshx  
    ldhx #$c34e  
vuelta:  
    nop  
    nop  
    aix #$ff  
    cphx #$0000  
    bne vuelta  
    pulx  
    finens  
return
```

7.17.6.1 Ejemplo 2 de estructura while

En este ejemplo se presenta un programa que, al igual que el presente en el ejemplo anterior, coloca una onda cuadrada de .5 segs de periodo en el bit 1 del puerto A del MCU. La variante en este caso es que dicha señal deberá presentarse solamente mientras el bit 7 del mismo puerto tenga un nivel de uno lógico.

```
defvarbptr pta 0
defvarbptr ddra 4
```

```
ddra = 2 ' pta1 es salida
```

```
while 1
```

```
while pta and &h80
```

```
pta = 0
```

```
gosub ret250ms
```

```
pta = 2
```

```
gosub ret250ms
```

```
wend
```

```
wend
```

```
ret250ms:
```

```
iniens
```

```
pshh
```

```
pshx
```

```
ldhx #$c34e
```

```
vuelta: nop
```

```
nop
```

```
aix #$ff
```

```
cphx #$0000
```

```
bne vuelta
```

```
pulx
```

```
finens
```

```
return
```

7.17.7 Estructura for

Al igual que muchos lenguajes de alto nivel, el BASIC validado por MINIBAS8A contempla la estructura "for". Ésta es de muy frecuente uso en muy diversas situaciones de programación. La forma genérica de ésta es la siguiente:

```
for varcont = exprvalini to exprvalfin step exprstep
```

código a ejecutarse para cada uno de los valores que
tomará la variable varcont

```
next varcont
```

En seguida se describen los elementos presentes en la forma genérica de la estructura "for".

Varcont. Es una variable que lleva la cuenta del número de veces que ha de ejecutarse el código presente en la estructura. **Cabe señalar que para MINIBAS8A varcont debe ser una variable de tipo byte, integer o long; en caso de que esto no sea así se reportará un error semántico.**

Exprvalini. Expresión cuyo resultado define el valor inicial que tomará la variable varcont. Si esta expresión arroja un resultado real, MINIBAS8A genera código que transforma este resultado al tipo que tenga la variable varcont

Exprvalfin. Expresión cuyo resultado define el valor final que tomará la variable varcont. Si esta expresión arroja un resultado real, MINIBAS8A genera código que transforma este resultado al tipo que tenga la variable varcont

Exprstep. Expresión cuyo resultado define el valor que se sumará a la variable contadora en cada paso del ciclo for, este valor podrá ser positivo o negativo. Si esta expresión arroja un resultado real, MINIBAS8A genera código que transforma este resultado al tipo que tenga la variable varcont. **Si en una estructura “for” se omite la palabra reservada step y su expresión asociada, el incremento de varcont en cada paso será uno.**

Next varcont. Delimita el final del ciclo for. En caso de que la variable varcont se actualice a un valor fuera del rango especificado en el encabezado de la estructura, el lazo se dará por terminado y se continuará con la ejecución en la instrucción inmediata posterior a la sentencia next.

A continuación se muestran ejemplos de uso de la estructura “for”

7.17.7.1 Ejemplo 1 de uso de la estructura “for”

Al ejecutarse el código mostrado a continuación se imprimirá en la consola la palabra “HOLA” cinco veces en sendos renglones.

```
for i~ = 1 to 5
print “HOLA”
next i~
```

7.17.7.2 Ejemplo 2 de uso de la estructura “for”

En este ejemplo se muestra un programa que despliega en el puerto A del MCU una cuenta descendente de 20 a 1 con una cadencia de 0.5 segs.

```
defvarbptr pta &h0
defvarbptr ddra &h4
```

```
ddra = &hff

while 1
for i~ = 20 to 1 step -1
pta = i~
gosub ret250ms
gosub ret250ms
next i~
```

```

                                wend
ret250ms:
    iniens
    pshh
    pshx
    ldhx #c34e
vuelta:
    nop
    nop
    aix #ff
    cphx #0000
    bne vuelta
    pulx
    finens
    return

```

7.18 Funciones propias de BASIC

Al igual que el BASIC tradicional, el validado por MINIBAS8A contiene diversas funciones de mucha utilidad en el desarrollo de aplicaciones. En la tabla 7.4 se muestra la información referente a éstas, detallándose ahí que tipo debe tener el argumento y en que tipo la función retorna el resultado.

Tabla 7.4 Funciones contempladas por MINIBAS8A

Denotación de la función	Forma del argumento X	Tipo en el que retorna el resultado	Retorna	Comentarios
abs(X)	Expresión real	single	Valor absoluto del argumento	
asc(X)	Expresión de tipo string	integer	Código ASCII del primer carácter del string del argumento	Si X es un string vacío despliega en la consola: "Argumento inválido" y para la ejecución
atn(X)	Expresión real	single	Angulo en radianes cuya tangente es el argumento	
chr\$(X)	Expresión real	string	Carácter de cuyo código ASCII es el argumanto	Si $X < 0$ ó $X > 255$ Despliega en la consola: "Argumento inválido" y para la ejecución
cint(X)	Expresión real	integer	Convierte X a tipo integer	Si $-32768 < X < 32767$ despliega en la consola: "Desbordamiento al convertir a tipo inferior" y para la ejecución
csgn(X)	Expresión de tipo integer	single	Convierte X a tipo single	

Tabla 7.4 Funciones contempladas por MINIBAS8A (continuación)

Denotación de la función	Forma del argumento X	Tipo en el que retorna el resultado	Retorna	Comentarios
cos(X) <i>X debe estar en radianes</i>	Expresión real	single	Coseno del argumento X	Si $ X > 12868$ despliega en la consola: “Argumento demasiado grande en función trigonométrica” y para la ejecución
exp(X)	Expresión real	single	e^x	Si $X > 88$ despliega en la consola: “Argumento inválido” y para la ejecución
fix(X)	Expresión real	single	Parte entera de X	
hex\$(X)	Expresión real	string	String que representa a X en hexadecimal	Si $-32768 < X < 32767$ despliega en la consola: “Desbordamiento al convertir a tipo inferior” y para la ejecución
int(X)	Expresión real	single	Entero más cercano a la izquierda de X	
len(X)	Expresión de tipo string	integer	Longitud del string X	
log(X)	Expresión real	single	Logaritmo natural de X	Si $X \leq 0$ despliega en la consola: “Argumento inválido” y para la ejecución
peek(X)	Expresión de tipo integer	integer	Valor del byte contenido en la localidad de memoria con dirección X	

Tabla 7.4 Funciones contempladas por MINIBAS8A (continuación)

Denotación de la función	Forma del argumento X	Tipo en el que retorna el resultado	Retorna	Comentarios
peekw(X)	Expresión de tipo integer	integer	Valor de la palabra de dos bytes contenida en las localidades de memoria con dirección X y X+1	
sin(X) <i>X debe estar en radianes</i>	Expresión real	single	Seno del argumento X	Si $ X > 12868$ despliega en la consola: “Argumento demasiado grande en función trigonométrica” y para la ejecución
sgn(X)	Expresión real	single	Si $X < 0$, retorna -1 Si $X = 0$, retorna 0 Si $X > 0$, retorna 1	
sqr(X)	Expresión real	single	\sqrt{X}	Si $X < 0$ despliega en la consola: “Argumento inválido” y para la ejecución
str\$(X)	Expresión real	string	String que representa en decimal el valor de X	
tan(X) <i>X debe estar en radianes</i>	Expresión real	single	Tangente del argumento X	Si $ X > 12868$ despliega en la consola: “Argumento demasiado grande en función trigonométrica” y para la ejecución

Tabla 7.4 Funciones contempladas por MINIBAS8A (continuación)

Denotación de la función	Forma del argumento X	Tipo en el que retorna el resultado	Retorna	Comentarios
val(X)	Expresión de tipo string	single	Valor numérico representado por el string X.	Retorna cero si X no representa un número válido en decimal
varptr(X)	Nombre de variable	integer	Dirección de memoria asignada a la variable denotada en el argumento X	
log10(X)	Expresión real	single	Logaritmo en base 10 de X	Si $X \leq 0$ despliega en la consola: “Argumento inválido” y para la ejecución

8 PROGRAMAS EJEMPLO EN MINIBAS8A

Una de las características más destacadas del ambiente AIDA08 es el compilador cruzado de BASIC MINIBAS8A. Esta herramienta permite desarrollar en alto nivel y con una sintaxis de basic, programas que después de ser compilados y ensamblados, pueden ser descargados y ejecutados en la tarjeta MINICON_08A.

En este capítulo se muestran nueve programas de ejemplo donde se hace uso de la sintaxis de MINIBAS8A y se accede a varios recursos del microcontrolador. La tabla 8.1 lista los ejemplos que se presentarán.

TABLA 8.1. EJEMPLOS PRESENTADOS EN EL CAPÍTULO.

EJEMPLO	NOMBRE
8.1	Lectura y Escritura de Puertos
8.2	Rutinas de retraso
8.3	Contador binario de ocho bits
8.4	Uso del Emulador de Terminal
8.5	Lógica de funcionamiento de un semáforo
8.6	Solución a la ecuación de segundo grado
8.7	Uso de una unidad de desplegado de caracteres
8.8	Velocidad promedio de un móvil

Ejemplo 8.1. Lectura y escritura de puertos.

Recursos MINIBAS8A:	de	Variables, subrutinas, ensamblador.
Recursos MINICON_08A:	de	Comunicación serial. Puerto A, Puerto B.

Planteamiento:

Se requiere hacer un programa que copie bit a bit el contenido del puerto B, previamente configurado como entrada, a los leds asociados al puerto A, previamente configurado como salida. Esto deberá realizarse de manera continua.

Solución:

Para dar solución a este problema se hace uso del registro DDRA, asociado con el puerto A y DDRB, asociado con el puerto B. Para ambos registros de ocho bits, si se configura un uno lógico, el bit asociado será puerto de salida, y si se escribe un cero, el bit asociado constituirá una entrada.

A continuación se lista un programa que efectúa la lógica requerida.

```
defvarbptr pta &h0      'Asociación de variable pta con Puerto A
defvarbptr ddra &h4      'Asociación de variable ddra con registro del Puerto A
defvarbptr ptb &h1      'Asociación de variable ptb con Puerto B
defvarbptr ddrb &h5      'Asociación de variable ddrb con registro del Puerto B
```

```
ddra=&FF      'Configura el Puerto A como Salidas.
ddrb=&00      'Configura el Puerto B como Entradas.
```

```
ciclo:        'Etiqueta de inicio de ciclo principal
```

```
    pta=ptb    'Copia el estado del Puerto B al A
```

```
    goto ciclo 'Regreso a etiqueta de inicio de ciclo
```

Ejemplo 8.2. Rutinas de retraso.

Recursos MINIBAS8A:	de	Variables, subrutinas, ensamblador.
Recursos MINICON_08A:	de	Comunicación serial , Puerto A.

Planteamiento:

Se requiere encender y apagar intermitentemente el Led0 del puerto A. Tanto el tiempo de encendido como el de apagado será de 250 milisegundos.

Solución:

Para abordar este ejemplo, primeramente se va configurar el puerto A como salida con la ayuda del DDRA. También se construirá en ensamblador una subrutina base con un período de 250 milisegundos que será llamada desde una rutina principal infinita.

A continuación se lista un programa que efectúa la lógica requerida.

```
defvarbptr pta &h0      'Asociación de variable pta con Puerto A
defvarbptr ddra &h4      'Asociación de variable ddra con el registro del Puerto A

      ddra=&FF            'Configura el Puerto A como salidas.

ciclo:      'Etiqueta de inicio de ciclo principal
      pta=&h00            'Apaga todos los leds del Puerto A
      gosub ret250ms      'Llamado a la subrutina ret250ms
      pta=&h01            'Enciende el led 0 del Puerto A
      gosub ret250ms      'Llamado a la subrutina ret250ms
      goto ciclo         'Regreso a etiqueta de inicio de ciclo

ret250ms:   'Inicio de subrutina ret250ms
      iniens             'Delimita inicio de código ensamblador
      pshh               'Guarda en stack el valor de H
      pshx               'Guarda en stack el valor de X
      ldhx #$C34E        'Carga HX con #$C34E (Ver sección 6.2)
vuelta:    nop           'No realiza ninguna operación
      nop               'No realiza ninguna operación
      aix #$FF           'Decrementa HX usando suma signada
      cphx #$0000        'Compara HX con #$0000
      bne vuelta         'Salta a Vuelta si no es igual
      pulx               'Lee a X del stack
      pulh               'Lee a H del stack
      finens             'Delimita fin de código ensamblador
      return            'Retorno de la subrutina
```

En este ejemplo se ha preferido el uso de lenguaje de ensamblador para realizar la rutina de retraso porque se conoce exactamente el número de ciclos de reloj que ocupa cada operación. Para ahondar en el uso de las subrutinas de retraso, se recomienda leer la sección 6.2 donde se explica por qué se carga el número \$C34E en la subrutina de retraso de 250 [ms].

Ejemplo 8.3. Contador Binario de ocho bits.

Recursos MINIBAS8A:	de	Variables, subrutinas, ensamblador.
Recursos MINICON_08A:	de	Comunicación serial , Puerto A.

Planteamiento:

Se busca realizar un contador binario de ocho bits que use los leds del puerto A para mostrar la cuenta. El valor será mostrado durante 500 milisegundos.

Solución:

A continuación se lista un programa que efectúa la lógica requerida.

```
dim i as integer      'Definicion de variable contadora i
defvarbptr pta &h0    'Asociación de variable pta con Puerto A
defvarbptr ddra &h4    'Asociación de variable ddra con el registro del Puerto A

        ddra=&FF        'Configura el Puerto A como salidas.

ciclo:                'Etiqueta de inicio de ciclo principal

        for i=0 to 255    'Inicio de ciclo for
            pta=i
                gosub ret250ms    'Llamado a la subrutina ret250ms
                gosub ret250ms    'Llamado a la subrutina ret250ms
        next i            'Fin de ciclo for

        goto ciclo        'Regreso a etiqueta de inicio de ciclo

ret250ms:              'Inicio de subrutina ret250ms
    iniens              'Delimita inicio de código ensamblador
    pshh                'Guarda en stack el valor de H
    pshx                'Guarda en stack el valor de X
    ldhx #$C34E          'Carga HX con #$C34E (Ver sección 6.2)
vuelta:  nop            'No realiza ninguna operación
        nop            'No realiza ninguna operación
        aix #$FF        'Decrementa HX usando suma signada
        cphx #$0000     'Compara HX con #$0000
        bne vuelta      'Salta a Vuelta si no es igual
        pulx            'Lee a X del stack
        pulh            'Lee a H del stack
        finens          'Delimita fin de código ensamblador
        return          'Retorno de la subrutina
```

Observe que para cambiar el valor que muestra el puerto A, es suficiente ir asignando en cada ciclo el valor de la variable contadora directamente a la variable especial de usuario asociada con dicho puerto.

Ejemplo 8.4. Uso del Emulador de Terminal.

Recursos MINIBAS8A:	de	Variables, subrutinas, ensamblador, Emulador de terminal
Recursos MINICON_08A:	de	Comunicación serial.

Planteamiento:

Se requiere evaluar la expresión del área de un triángulo dada su base y su altura. La introducción de las variables y la respuesta la debe realizar el microcontrolador a través de un emulador de terminal.

Solución:

Para resolver este ejemplo, se hará uso del emulador de terminal incluido en el ambiente AIDA08. En caso de requerir mayor información, puede revisar la sección 3.2.3. A continuación se lista un programa que efectúa la lógica requerida.

```
iniens      'Delimita inicio de código ensamblador
jsr lee#car  'Salto a subrutina de espera de recepción de caracteres
finens      'Delimita fin de código ensamblador

ciclo:      'Etiqueta de inicio de ciclo principal
CLS         'Borra el contenido de la pantalla del emulador
PRINT "CÁLCULO DEL ÁREA DE UN TRIÁNGULO"  'Despliegue en el emulador
PRINT "INTRODUCIR VARIABLES:"            'Despliegue en el emulador

INPUT "Base b=", VarBase                  'Recibe y asigna a una variable
INPUT "Altura h=", VarAltura               'Recibe y asigna a una variable

PRINT "....."                          'Despliegue en el emulador
PRINT "El area del tiángulo de base b="; VarBase;
      "y de altura h="; VarAltura; "es: A=",(VarBase*VarAltura)/2

INPUT VarBase                             'Queda en espera para reiniciar el ciclo.
goto ciclo                                'Regreso a etiqueta de inicio de ciclo
```

Para conocer más acerca del uso de las sentencias Input y Print, diríjase a las secciones 7.6 y 7.7 respectivamente.

Ejemplo 8.5. Lógica de funcionamiento de un Semáforo.

Recursos MINIBAS8A:	de	Variables, subrutinas, ensamblador.
Recursos MINICON_08A:	de	Comunicación serial , Puerto A.

Planteamiento:

Se requiere implementar lógica de control de luces de un semáforo para un cruce de dos vías. El tiempo que deben durar las señales de avance (Luz verde) es de un minuto, mientras que el tiempo de preventiva (Luz ámbar) debe ser de cinco segundos.

Solución:

Para este ejemplo se ha decidido usar los leds de la tarjeta MINICON_08A que están conectados al puerto A como indicadores de las luces de los dos semáforos. La siguiente tabla muestra la definición de los estados de las luces. Adicionalmente la columna “HEX” muestra el valor de puerto A en hexadecimal, y la columna “TIEMPO” muestra la duración del estado.

	R1	A1	V1	R2	A2	V2				
ESTADO	PTA7	PTA6	PTA5	PTA4	PTA3	PTA2	PTA1	PTA0	HEX	TIEMPO
1	1	0	0	0	0	1	0	0	\$84	1 min
2	1	0	0	0	1	0	0	0	\$80	5 seg
3	0	0	1	1	0	0	0	0	\$30	1 min
4	0	1	0	1	0	0	0	0	\$50	5 seg

A continuación se lista un programa que efectúa la lógica requerida.

```
dim i as integer      'Definicion de variable contadora i
dim j as integer      'Definicion de variable contadora j
defvarbptr pta &h0    'Asociación de variable pta con Puerto A
defvarbptr ddra &h4    'Asociación de variable ddra con el registro del Puerto A
```

```
    ddra=&fc           'pta7 a pta2 son salidas
```

```
ciclo:                'Etiqueta de inicio de ciclo principal
    pta=&h84            'Establece estado 1 en el puerto A
    gosub ret1min       'Llamado a la subrutina ret1min
    pta=&h88            'Establece estado 2 en el puerto A
    gosub ret5seg       'Llamado a la subrutina ret5seg
    pta=&h30            'Establece estado 3 en el puerto A
    gosub ret1min       'Llamado a la subrutina ret1min
    pta=&h50            'Establece estado 4 en el puerto A
    gosub ret5seg       'Llamado a la subrutina ret5seg
    goto ciclo         'Regreso a etiqueta de inicio de ciclo
```

ret250ms:	'Inicio de subrutina ret250ms
iniens	'Delimita inicio de código ensamblador
pshh	'Guarda en stack el valor de H
pshx	'Guarda en stack el valor de X
ldhx #\$C34E	'Carga HX con #\$C34E (Ver sección 6.2)
vuelta: nop	'No realiza ninguna operación
nop	'No realiza ninguna operación
aix #\$FF	'Decrementa HX usando suma signada
cphx #\$000	'Compara HX con #\$0000
bne vuelta	'Salta a Vuelta si no es igual
pulx	'Lee a X del stack
pulh	'Lee a H del stack
finens	'Delimita fin de código ensamblador
return	'Retorno de la subrutina
ret5seg:	'Inicio de subrutina ret5seg
for i=1 to 20	'Inicio de ciclo for
gosub ret250ms	'Llamado a la subrutina ret250ms
next i	'Fin de ciclo for
return	'Retorno de la subrutina
ret1min:	'Inicio de subrutina ret1min
for j=1 to 20	'Inicio de ciclo for
gosub ret5seg	'Llamado a la subrutina ret5seg
next j	'Fin de ciclo for
return	'Retorno de la subrutina

Como se puede observar en el ciclo principal del programa, la configuración, lectura y escritura del puerto A del microcontrolador se realiza en forma muy sencilla una vez que se han declarado las variables ddra y pta.

La subrutina de retraso básica es la misma que la del ejemplo 8.1. Observe que esta rutina es llamada veinte veces para tener un retraso de cinco segundos y ésta a su vez, es llamada otras veinte veces para integrar el retraso de un minuto.

Ejemplo 8.6. Solución a la ecuación de segundo grado.

Recursos MINIBAS8A:	de	Variables, subrutinas, ensamblador, Emulador de terminal
Recursos MINICON_08A:	de	Comunicación serial.

Planteamiento:

Se requiere calcular las raíces de una ecuación de segundo grado discriminando los casos en que las raíces sean complejas, reales repetidas y reales diferentes. La introducción de las variables y la respuesta la debe realizar el microcontrolador a través de un emulador de terminal.

Solución:

Al igual que el ejemplo 8.3, para resolver este ejemplo, se hará uso del emulador de terminal incluido en el ambiente AIDA08. Observe el uso intensivo de las funciones INPUT y PRINT. A continuación se lista un programa que efectúa la lógica requerida.

iniens 'Delimita inicio de código ensamblador
jsr lee#car 'Salto a subrutina de espera de recepción de caracteres
finens 'Delimita fin de código ensamblador

otrom: 'Etiqueta de inicio de ciclo principal

PRINT "SOLUCIÓN DE ECUACIONES CUADRATICAS" 'Despliegue en el emulador
PRINT " $a*x^2 + b*x + c = 0$ 'Despliegue en el emulador
PRINT "Introducir coeficientes:" 'Despliegue en el emulador

dara: INPUT "a=", a 'Recibe y asigna a una variable
IF a = 0. THEN 'Verifica el valor de a
PRINT "'a' debe ser diferente de cero"
GOTO dara 'Solicita de nuevo la variable a
ENDIF

INPUT "b=", b 'Recibe y asigna a una variable
INPUT "c=", c 'Recibe y asigna a una variable

dis = b * b - 4. * a * c 'Calcula el discriminante
PRINT "dis="; dis

IF dis < 0. THEN 'Calcula el caso de raices complejas
PRINT "RAICES COMPLEJAS"

pim = SQR(-dis) / (2. * a)
pre = -b / (2 * a)
PRINT "r1="; pre; "+ j"; pim 'Despliega raices complejas
PRINT "r2="; pre; "- j"; pim 'Despliega raices complejas

```

ELSEIF dis = 0. THEN      'Calcula el caso de raices reales repetidas
    PRINT "RAICES REALES REPETIDAS"
    rep = -b / (2 * a)
    PRINT "r1=r2="; rep    'Despliega raices reales repetidas

ELSEIF dis > 0. THEN      'Calcula el caso de raices reales diferentes

    PRINT "RAICES REALES DIFERENTES"
    radi = SQR(dis) / (2 * a)
    rep = -b / (2 * a)
    r1 = rep + radi
    r2 = rep - radi
    PRINT "r1="; r1        'Despliega raices reales diferentes
    PRINT "r2="; r2        'Despliega raices reales diferentes
ENDIF

INPUT d
CLS
GOTO otrom

```

Ejemplo 8.7. Uso de la Unidad de Desplegado de Caracteres.

Recursos MINIBAS8A:	de	Variables, subrutinas, ensamblador, Sentencia Include, Emulador de terminal.
Recursos MINICON_08A:	de	Comunicación serial, Unidad Desplegadora.

Planteamiento:

Al igual que en el ejemplo 8.4, se busca calcular el área de un triángulo, introduciendo los parámetros desde el emulador de terminal, pero en este caso, la respuesta deberá mostrarse en la unidad de despliegue de caracteres de dos renglones y 20 columnas.

Solución:

Utilizando los recursos de despliegue de caracteres mostrados en la sección 6.6. A continuación se muestran las modificaciones al ejemplo 8.4, que realizan la lógica requerida.

```
iniens                                'Delimita inicio de código ensamblador
$include "e:\hc08\mdam8a00.asm "      'Llamado a la biblioteca mdam8a00.asm
'La línea anterior debe ser alterada según el directorio donde se encuentre la biblioteca
mdam8a00.asm
jsr lee#car                          'Salto a subrutina de espera de recepción de caracteres
finens                               'Delimita fin de código ensamblador
```

```
ciclo:                               'Etiqueta de inicio de ciclo principal
CLS                                 'Borra el contenido de la pantalla del emulador
PRINT "CÁLCULO DEL ÁREA DE UN TRIÁNGULO" 'Despliegue en el emulador
PRINT "INTRODUCIR VARIABLES:"        'Despliegue en el emulador

INPUT "Base b=", VarBase              'Recibe y asigna a una variable
INPUT "Altura h=", VarAltura          'Recibe y asigna a una variable
```

```
num$="El area es:"                  'Cadena de caracteres a ser desplegada
mipos~=$80                          'Numero de renglón donde se mostrará
gosub despstr                        'Llamado a rutina de despliegue
num$= "A=" + cstr((VarBase*VarAltura)/2)
mipos~=$C0
gosub despstr
```

```
INPUT VarBase                       'Queda en espera para reiniciar el ciclo.
goto ciclo                          'Regreso a etiqueta de inicio de ciclo
```

‘*****

‘Subrutina ‘despstr’

‘*****

‘Despliega en el LCD el string num\$, esto a partir de la posición dada

‘por el byte comando contenido en la variable mips~.

‘Antes de invocar:

‘num\$<---string a desplegar

‘mips~ <-- comando de posicionamiento asociado.

‘Al retornar ya se habrá desplegado el string num\$ a partir de la posición

‘dada por el byte contenido en mips~.

‘Se emplea además la variable byte lstr~, esta retorna con un valor nulo

despstr:

lstr~=len(num\$)

gosub borraren

iniens

lda num\$

psha

pulh

ldx num\$+1

lda mips~

jsr escom4

yuopp: lda ,x

jsr escdat4

aix #\$01

lda lstr~

deca

sta lstr~

bne yuopp

finens

return

‘*****

‘Subrutina ‘borraren’

‘*****

‘Coloca caracteres espacio en un renglón del LCD.

‘Supone renglones de 20 espacios.

‘Antes de invocar:

‘mips~<---comando de posicionamiento de inicio de renglón.

borraren:


```
    iniens
    psha
    pshx
    lda mipo~
    jsr escom4
    ldx #$14
otromas:  lda #$20
    jsr escdat4
    decx
    bne otromas
    pulx
    pula
    finens

    return
```

Ejemplo 8.8. Velocidad promedio de un móvil.

Recursos MINIBAS8A:	de	Variables, subrutinas, ensamblador, Sentencia Include, Emulador de terminal.
Recursos MINICON_08A:	de	Comunicación serial, Unidad Desplegadora.

Planteamiento:

Se requiere medir la velocidad promedio de vehículos que pasan entre dos puntos denominados 'A' y 'B' separados por una distancia denominada 'sd'. En cada punto existen sendos sensores denominados 'SA' y 'SB', los cuales generan cada que pasa un vehículo, sendos pulsos verificados en bajo de un milisegundo. Por cada grupo de 'n' vehículos que hayan pasado se debe desplegar la media de la velocidad promedio de todos ellos.

Solución:

Para solucionar este problema se asume que solo puede pasar un móvil a la vez. También se considera que el sensor SA se encuentra conectado al PTB0 mientras que el sensor SB se conecta al PTB3. Para medir el intervalo de tiempo asociado con el paso de cada uno de los vehículos se emplea la instancia de interrupción por sobreflujo del temporizador 1 del dispositivo, configurada ésta de modo que el sobreflujo se dé cada 10ms. En la rutina de servicio asociada se incrementa en uno una variable entera que contendrá la cuenta de centésimas de segundo que han transcurrido al pasar cada móvil entre los puntos 'A' y 'B'. A continuación se lista un programa que efectúa la lógica requerida.

‘Definición de variables especiales de usuario tipo byte

```
defvarbptr ptb &h1
defvarbptr config1 &h1f
defvarbptr t1sc &h20
defvarbptr t1modh &h23
defvarbptr t1modl &h24
```

‘Definición de variables especiales de usuario tipo integer

```
defvariptr t1mod &h23
```

```
config1=1 ‘Deshabilitación del COP
```

```
iniens      ‘Delimita inicio de código ensamblador
jsr lee#car  ‘Salto a subrutina de espera de recepción de caracteres
finens      ‘Delimita fin de código ensamblador
```

‘t1mod para que el intervalo entre sobreflujos de 10 ms @Fbus =2 Mhz.

```
t1mod=&h4e1f
```

‘Código ensamblador para habilitar interrupciones globalmente.

```
iniens
cli
finens
```

```
lecsd: cls
```

```

input "Dar 'sd' en metros,sd=",sd
if sd<=0 then lecsd

while 1
nv%=1
sumvelpr=0.
input"n=",ntop%
if ntop%<=0 then
print "Fin de la sesión,para reiniciar"
print "oprimir el botón de RESET"
end
endif
while nv%<=ntop%
contcent%=0
iniens
hhhh:    jsr lee#car
        cmp #$31
        bne hhhh
        finens
        t1sc=&h40 'pstop<--0,toie<--1
        print "El vehículo ";nv%;" cruzó el punto 'A'"
        iniens
qqqq:    jsr lee#car
        cmp #$61
        bne qqqq
        finens
        t1sc=&h0 'pstop<---0,toie<--0
        print "El vehículo ";nv%;" cruzó el punto 'B'"
        velpr=360.*sd/contcent%
        print"Vp para el vehículo ";nv%;" es";velpr;" Km/h"
        print
        sumvelpr=sumvelpr+velpr
        nv%=nv%+1
        wend
medv=sumvelpr/ntop%
print"La media de las velocidades es";medv;" Km/h"
print
wend
servovf:
glip
varaux~=t1sc
t1sc=t1sc and &h7f 'tof<---0
contcent%=contcent%+1
relip
retint

'Declaración de vector de usuario para el evento de sobreflujo del TIM1.
dataw &hd7f2 servovf
'Declaración de vector de RESET de usuario.
dataw &hd7fe &h8000

```

REFERENCIAS

- [1] FREESCALE, M68HC08 Microcontrollers. Technical Data, 2002.
Archivo mc68hc908gp32.pdf descargable desde la página de FREESCALE. www.freescale.com
- [2] A. Salvá, “SD_908, SISTEMA PARA DESARROLLO CON MICROCONTROLADORES DE LA FAMILIA 68HC908”, México D. F., Memoria de SOMI18, octubre de 2003.
Archivo textsomi18.pdf, descargable desde la página: <http://dctrl.fi-b.unam.mx/~salva>
- [3] A. Salvá, “MANUAL DE USUARIO DE AMIGO_08”, México D. F., abril de 2005.
Archivo a8mu.pdf, descargable desde la página: <http://dctrl.fi-b.unam.mx/~salva>
- [4] Archivo AlphanumericAppNotes.PDF, que contiene las notas de aplicación del display.
Descargable de: <http://www.purdyelectronics.com/PDF/AlphanumericAppNotes.PDF>
- [5] A. Salvá, “DISPOSITIVOS CHIPBAS8, MICROCONTROLADORES HC08 PROGRAMABLES EN LENGUAJE BASIC”. Memoria del Simposio Anual de Automatización, Electrónica e Instrumentación.(SAAEI 2009) Celebrado en julio de 2009 en la Universidad Carlos III en Madrid España.