GUÍA BÁSICA DE USUARIO PARA EL DESARROLLO DE PROGRAMAS BAJO EL COMPILADOR CRUZADO SBAS8

Por: Antonio Salvá Calleja

Profesor de carrera

Departamento de Control y Robótica

Facultad de Ingeniería

UNAM

Agosto de 2016

Índice

Para una adecuada comprensión de los conceptos expuestos en esta guía, se requiere que el lector tenga experiencia de programación en lenguaje ensamblador de microcontroladores de la familia hcs08, y BASIC soportado por el compilador cruzado MINIBAS8A.

Los ejemplos de apoyo ilustrativo suponen que el MCU destino es el chip mc9s08sh32, habilitado como dispositivo CHIPBAS8SH32; si esto no fuera el caso no es difícil hacer las adecuaciones requeridas acorde con algún otro MCU de la familia hcs08.

1 Paradigma de programación contemplado por el compilador cruzado SBAS8

El compilador cruzado SBAS8 es una evolución del compilador cruzado MINIBAS8A desarrollado previamente por el autor de esta guía. MINIBAS8A admite código tipo QuickBasic, para ser ejecutado en microcontroladores de la familia HCS08 de NXP (FREESCALE), aspectos acerca de la programación bajo MINIBAS8A pueden verse en el capítulo 7 de [1]. Por otra parte, el paradigma de programación para SBAS8, contempla la inclusión de procedimientos denominados como: **sub, fun y subint;** los dos primeros pueden o no tener variables argumento, el último no contempla tener variables argumento. Los argumentos pueden ser expresiones de tipo byte, integer, long, single y string. Los procedimientos de tipo fun, pueden retornar variables de tipo numérico o string. Los procedimientos de tipo sub y subint no retornan variables. El código fuente dentro de cada uno de los procedimientos que integren un programa, deberá seguir los lineamientos de programación de MINIBAS8A y podrá contener invocaciones a procedimientos de tipo **sub** y **fun**, que sean parte del mismo programa. Una excepción a esto último es el hecho de que si el nombre de un procedimiento es 'main', éste no es invocable; **además, un procedimiento no podrá ser invocado desde dentro de él mismo.**

A continuación en las secciones 1.1, 1.2 y 1.3; se explica como deben ser declarados los procedimientos de tipo 'sub', 'fun' y 'subint'. En la sección 1.4 se esboza la estructura que deben tener los argumentos de un determinado procedimiento, al ser éste invocado en alguna determinada posición en el programa.

1.1 Formato para la declaración de un procedimiento de tipo 'sub'

Los procedimientos de tipo 'sub' no retornan valores, pero pueden tener o no argumentos el formato para declararlos es el siguiente:

def_sub *nombre del procedimiento*(lista de argumentos separados por comas)

Declaración, si esto es requerido, de variables especiales locales al procedimiento.

Declaración, si esto es requerido, de variables locales al procedimiento,
éstas pueden ser de tipo numérico; o bien, de tipo string.

Código fuente bajo los lineamientos de MINIBAS8A.
De ser necesario, podrá haber invocaciones a procedimientos de tipo sub o fun.

end_sub 'Cierre del procedimiento.

El nombre del procedimiento podrá ser una cadena de hasta 16 caracteres, siendo el primero no número y los subsecuentes podrán ser: letras o números o vocales acentuadas o el guión bajo. Por ejemplo, la cadena: **sí_fue_entero** puede ser el nombre de un procedimiento. Cabe señalar que lo especificado aquí para la cadena que denota el nombre de un procedimiento, es aplicable también para procedimientos de tipo 'fun' y 'subint'.

En la lista de argumentos se deberá explicitar para cada variable implicada el tipo de ésta, esto se explicita en esta guía mediante ejemplos ilustrativos.
Si por razones del flujo de ejecución del procedimiento, se requiere que este finalice antes de llegar a la sentencia **end_sub**, bastará con poner la palabra reservada **'exit_sub'** en la posición apropiada; esta sentencia podrá usarse tantas veces como sea necesario en el procedimiento.

Ejemplo 1. Declaración de procedimiento de tipo 'sub'

A continuación se muestra la declaración de un procedimiento de tipo 'sub', éste genera una espera de un número de milisegundos determinado por una variable argumento de tipo integer, denotada como 'n'. Si la variable argumento es menor que uno; o bien, si ésta es mayor que

32766 se debe retornar sin que se genere un retardo. El código asociado sigue los lineamientos de MINIBAS8A, véase el capítulo 7 de [1].

```
def_sub retms(n as integer)
  'Este procedimiento genera una espera de 'n' milisegundos,
  'se supone que la frecuencia del reloj de bus del MCU
  'es 20 MHz.

    if n < 1 or n > 32766 then
    'Si 'n' no está en el rango apropiado
    'retorna sin generar un retardo.
     exit_sub
    endif

        for i%=1 to n
        gosub ret1ms
        next i%

        exit_sub

ret1ms:
        iniens
        pshh
        pshx
        ldhx #$07cd
vuelta: nop
        nop
        aix #$ff
        cphx #$0000
        bne vuelta
        pulx
        pulh
        rts
        finens

end_sub
```

Nótese como se declara el hecho de que la variable argumento 'n' es de tipo integer.

1.2 Formato para la declaración de un procedimiento de tipo 'fun'

Los procedimientos de tipo 'fun' pueden tener o no argumentos; además, pueden retornar ya sea un string o bien un valor numérico, asignable a una variable de tipo apropiado. A este tipo de procedimientos también suele llamárseles como *funciones*. El formato para la declaración de este tipo de procedimientos es:

def_fun *nombre del procedimiento*(lista de argumentos separados por comas) as *tipovaret*

Declaración, si esto es requerido, de variables especiales locales al procedimiento.

Declaración, si esto es requerido, de variables locales al procedimiento,
éstas pueden ser de tipo numérico; o bien, de tipo string.

Código fuente bajo los lineamientos de MINIBAS8A.
De ser necesario, podrá haber invocaciones a procedimientos de tipo sub o fun.

' Asignación de variable a retornar
nombre del procedimiento = expresión que calcule el valor a retornar
end_fun 'Cierre del procedimiento.

La cadena denotada como *tipovaret* podrá ser una de las siguientes palabras reservadas: byte, integer, long, single, string; esto acorde con la entidad numérica o de tipo string que retorne el procedimiento.

Al igual que lo propio para la declaración de procedimientos de tipo 'sub', el nombre del procedimiento podrá ser una cadena de hasta 16 caracteres, siendo el primero no número y los subsecuentes podrán ser: letras o números o vocales acentuadas o el guión bajo. Además, en la lista de argumentos se deberá explicitar para cada variable implicada el tipo de ésta, esto se explicita en esta guía en diversos ejemplos ilustrativos.

Si por razones del flujo de ejecución del procedimiento, se requiere que este finalice antes de llegar a la sentencia **end_fun**, bastará con poner la palabra reservada **'exit_fun'** en la posición apropiada; esta sentencia podrá usarse tantas veces como sea necesario en el procedimiento.

Nótese que para la asignación de la variable a retornar se usa una sentencia de la siguiente forma:

***nombre del procedimiento* = expresión que calcule el valor a retornar**

esto podrá ser usado las veces que fuere necesario dentro del procedimiento. En caso de que la sentencia antes citada no esté presente; si la entidad a retornar es de tipo numérico ésta retornara con el valor cero; por otra parte si lo retornado es de tipo string; éste retornará como string nulo. **Cabe señalar que, no debe haber etiqueta, en el renglón donde esté la asignación de la entidad a retornar, si esto fuera el caso se genera un error al compilarse el programa.**

Ejemplo 2. Declaración de procedimiento de tipo 'fun'
A continuación se muestra la declaración de un procedimiento de tipo 'fun' denotado como **potpos**, cuyos argumentos son una variable de tipo single y una variable de tipo byte denotadas respectivamente como 'x1' y m. La función debe retornar el valor que resulta de elevar a la potencia 'm' la variable x1. El exponente 'm' debe ser un entero mayor o igual a uno y menor o igual que 20; de no ser éste el caso, se deberá detener la ejecución y presentar en la consola el mensaje:

Exponente inválido en la función potpos,
se detendrá la ejecución del programa.

La declaración asociada con esta función es:

```
def_fun potpos(x1 as single, m as byte) as single
      dim mipot as single

      if m < 1 or m > 20. then
       print "Exponente inválido en la función potpos,"
       print "se detendrá la ejecución."
       iniens
       sei    'Se deshabilitan globalmente las interrupciones.
       finens
       end
      endif

      mipot=1.
      for i~=1 to m
      mipot=mipot*x1
      next i~

      potpos=mipot 'Asignación de valor a retornar.
end_fun
```

Para fines ilustrativos, se presenta el siguiente tramo de código que, desde luego podría estar en cualquier otro procedimiento del programa que no sea el propio 'potpos' de este ejemplo.

```
mibte = 4
base = 6.5
poten = potpos(base, mibte)
```

Se supone que mibte es un variable de tipo byte y las variables base y poten son de tipo single.
Al ejecutarse el código anterior el valor numérico 1785.0625, estará contenido en la variable poten.

1.3 Formato para la declaración de un procedimiento de tipo 'subint'

Los procedimientos de tipo 'subint' no retornan valores, y no pueden tener argumentos **su uso en un programa es para validar rutinas de servicio de interrupción,** el formato para declararlos es el siguiente:

def_subint *nombre del procedimiento*( )

Declaración, si esto es requerido, de variables especiales locales al procedimiento.

Declaración, si esto es requerido, de variables locales al procedimiento,
éstas pueden ser de tipo numérico; o bien, de tipo string.

Código fuente para rutinas de interrupción bajo los lineamientos de MINIBAS8A.
Véase la sección 7.12 de [1].

De ser necesario, podrá haber invocaciones a procedimientos de tipo sub o fun.

end_subint 'Cierre del procedimiento.

int#(nnn) *nombre del procedimiento* 'Colocación del vector asociado.

La sentencia que se aprecia después del cierre de procedimientos de tipo 'subint', es para que el vector asociado a la instancia de interrupción que corresponda, sea colocado en el par de direcciones propias para ello; SBAS8 determina esta dirección de carga del vector, acorde con el MCU destino predeterminado en el software manejador PUMMA_EST. La cadena nnn es el número de instancia de interrupción asociado a tres dígitos en decimal, éste puede verse para el MCU mc9s08sh32, en la tabla 5-2 que está en la página 64 de [2]. **Es importante señalar que la sentencia declaratoria de la colocación del vector, debe estar a partir de la primera columna de la pantalla de edición; además, después de la cadena que denota el nombre del procedimiento de tipo sub_int, deberá haber cuando menos un espacio antes del fin del renglón implicado, de no ser así, SBAS8 reportará un error con la siguiente leyenda asociada con éste.**

Sentencia de código colocada fuera de intervalo delimitado por
declaraciones de apertura y cierre de un procedimiento

Al igual que lo propio para la declaración de procedimientos de tipo 'sub' y 'fun', el nombre del procedimiento podrá ser una cadena de hasta 16 caracteres, siendo el primero no número y los subsecuentes podrán ser: letras o números o vocales acentuadas o el guión bajo.
Si por razones del flujo de ejecución del procedimiento, se requiere que este finalice antes de llegar a la sentencia **end_subint**, bastará con poner la palabra reservada **'exit_subint'** en la posición apropiada; esta sentencia podrá usarse tantas veces como sea necesario en el procedimiento.

Ejemplo 3. Declaración de procedimiento de tipo 'subint'
Para fines ilustrativos en este ejemplo se muestra la declaración de un procedimiento de tipo 'subint', en éste simplemente se incrementa una variable global de tipo integer denominada contms. Se supone que la instancia de interrupción implicada es el *overflow* del contador del temporizador 1. **La razón para el uso de las palabras reservadas 'glip' y 'relip' se explica en la sección 7.12 de [1]**. En la tabla 5-2 que está en la página 64 de [2], se aprecia que el número de instancia de interrupción para el evento de *overflow* del temporizador 1 es once. Al procedimiento aquí ejemplificado se le nombro como 'servovft1'. La declaración de éste es:

```
def_subint servovft1()
  dim byaux as byte

  'Código para regresar a cero la bandera TOF
   byaux = tpm1sc
   tpm1sc = tpm1sc and &h7f 'TOF <-- 0
  '.....................................
   glip
   contms = contms + 1 'Incrementa contms
   relip

end_subint

int#(011) servovft1 'Colocación del vector asociado.
```

Más adelante en esta guía se muestra un ejemplo de un programa completo compilable por SBAS8. En éste se usa una interrupción por *overflow* del temporizador 1, siendo el procedimiento tipo subint asociado el detallado en este ejemplo con un agregado de código requerido por la aplicación.

1.4 Formato de argumentos al invocarse procedimientos de tipo 'sub' y 'fun'
Para invocar un procedimiento bastará con escribir la sentencia apropiada dentro del procedimiento que sea necesario llamarlo; la forma que deben presentar los argumentos de éste al ser invocado, puede ser desde un simple número o string explícitos según sea el caso; hasta una expresión cuyo resultado sea un valor numérico o un string; esto acorde con el tipo de variable que se haya declarado para fines del argumento en cuestión. Para fines ilustrativos se ejemplificará este concepto auxiliándose del procedimiento de tipo 'sub' denominado como retms para el cual, en el ejemplo 1, se ha mostrado su código.

Así por ejemplo, para generar una espera de un segundo, bastará escribir la sentencia:

```
retms(1000)
```

En la ejemplificación anterior, el argumento que se pasa es un número explícito; sin embargo, en lo general el argumento podrá ser una expresión que tenga como resultado un valor; si éste no es asimilable a un entero de tipo integer, SBAS8 lo convierte a este tipo. Por ejemplo, al ejecutarse el siguiente tramo de código se generará una espera de 1.5 segundos.

```
a1 = 3001.5
a2 = 2.
retms (a1/a2)
```

ya que el resultado de dividir las dos variables reales a1 y a2 es el real 1500.75, que para fines del argumento implicado, SBAS8 lo convierte a tipo integer que es 1500.
Es importante señalar que para las expresiones que denotan argumentos de un procedimiento al ser éstos invocados, no debe haber operadores de tipo booleano, si este

llegara a ser el caso se generan errores al compilar. Por ejemplo, al compilarse con SBAS8 un programa que contenga la siguiente línea:

```
retms(a% and b%)
```

se generan errores y se aborta la compilación. Para solventar el problema bastará con efectuar la operación booleana antes de la invocación del procedimiento implicado, así el siguiente tramo de código resuelve la problemática aquí descrita.

```
z% = a% and b%
retms (z%)
```

Otra restricción para expresiones asociadas con argumentos de procedimientos, es el hecho de que en éstas no se permite la presencia de invocaciones a procedimientos de tipo 'fun'. Por ejemplo, se produce un error al compilarse un programa que contenga la siguiente línea de código:

```
retms (1000*potpos(2.5, 2)
```

donde en el argumento implicado, está presente la función potpos cuya funcionalidad y declaración se describió en el ejemplo 2. Para solventar este problema podría emplearse el siguiente tramo de código:

```
mipoten = 1000*potpos(2.5, 2)
retms (mipoten)
```

2. Estructura de un programa compilable con SBAS8

En lo general, un programa fuente compilable por SBAS8, podrá contener hasta cuatro bloques de código, los cuales se describen a continuación:

2.1 Formato del Bloque 1 de un programa compilable con SBAS8

Este bloque contendrá en el siguiente orden lo siguiente:

1. Declaraciones asociadas con las variables numéricas especiales de usuario que la aplicación requiera, véase lo propio acerca de este tema en la sección 7.9 de [1]. Estas variables serán globales; por lo tanto, serán visibles en todos los procedimientos del programa, por lo regular se emplean éstas para asociar variables de tipo byte con identificadores que coinciden con nombres de registros del MCU y haciendo que la dirección asociada en cada caso sea la misma que es propia del registro en cuestión. O bien para la definición de constantes numéricas, véase más adelante la sección 3.

2. Declaraciones asociadas con las variables y arreglos globales de tipo numérico, si estas son necesarias en el programa. **Cabe señalar que SBAS8 no permite que haya variables globales de tipo string bajo el formato del lenguaje BASIC contemplado por MINIBAS8A**, aunque si se podría hacer uso de arreglos globales de tipo byte, que podrían asociarse con cadenas de caracteres que desde luego serían visibles en todos los procedimientos.

3. Código asociado con la inicialización de recursos del MCU que son requeridos por la aplicación que el programa valide. Esto podría ser entre otros casos: la inicialización como salida de algún puerto del MCU, la configuración de registros propios de los periféricos del MCU que use la aplicación, la inicialización de variables globales de tipo numérico, etc.

4. Palabra reservada **end_codini** que delimita el final del código de inicialización.

Si el programa no requiere que haya variables especiales de usuario ni variables globales ni código de inicialización, el bloque 1 aquí descrito contendrá únicamente a la palabra reservada 'end_codini'. **Al detectar SBAS8 el final del código de inicialización genera un salto a un procedimiento de tipo sub que deberá tener el nombre 'main' y siempre deberá estar colocado inmediatamente después de la sentencia end_codini.**

Ejemplo 4. Muestra de código de inicialización
Para fines ilustrativos, a continuación se muestra la forma que tendría el código de inicialización para un programa que usa dos registros del MCU asociados con el puerto A, de éste; además, se usarán dos variables globales, una de tipo byte y la otra de tipo single, las cuales se inicializan, la primera con 4 y la segunda con 4.5, el código en cuestión puede ser el mostrado a continuación:

```
'Ejemplo de código de inicialización.

 'Contempla el uso de:
 'Dos variables especiales asociadas
 'con registros del puerto A del MCU mc9s08sh32.

 'Dos variables globales una de tipo byte y la otra de tipo single,
 'La primera debe inicializarse con 4 y la segunda con 4.5, el código
 'asociado es:

'Declaraciones de variables especiales de usuario.
 defvarbptr ptad &h0
 defvarbptr ptadd &h1
'.............................................

'Declaraciones de variables globales numéricas.
 dim groc as single
 dim mibyte as byte
'.........................................

'Inicialización de variables globales y de recursos del MCU.

 ptad = &hff 'Todas la líneas del puerto A son salidas.

 mibyte = 4
 groc = 4.5

'.........................................

 end_codini 'Fin de código de inicialización
```

2.2 Formato del Bloque 2 de un programa compilable con SBAS8
Este bloque contempla la declaración del procedimiento de entrada al programa, que debe denominarse como 'main', ser de tipo 'sub', no debe tener argumentos, y debe estar colocado inmediatamente después de la sentencia end_codini que delimita el final del código de inicialización. El proceso 'main' se ejecuta inmediatamente después que ha concluido la ejecución del código de inicialización (Bloque 1). **Cabe señalar que SBAS8 potencialmente puede reconfigurarse de modo que el nombre del proceso**

de entrada no sea 'main', sino otro como podrían ser entre otros: 'begin', 'inicio' ,'start', 'arranque, etc.

Ejemplo 5. Muestra de procedimiento 'main'
En este ejemplo se muestra la declaración de un procedimiento 'main', al ejecutarse un programa que lo contenga, después de abrir la terminal que validaría la consola de interfaz y oprimir cualquier tecla en ésta, deberá aparecer en la pantalla el siguiente mensaje:

Hola desde el MCU MC9S08SH,
habilitado como dispositivo CHIPBAS8SH.

Después de esto se pasará a un lazo que contendrá código que hará que se efectúen las siguientes acciones:

1. Se pide al usuario especificar los valores de dos variables numéricas locales al procedimiento. Una de tipo byte denotada como mibte y la otra de tipo single denotada como base.

2. Se efectúan cálculos que implican la invocación de la función potpos descrita en el ejemplo 2.

3. Se despliega en la consola el resultado de los cálculos hechos en el paso 2.

4. Se repite la secuencia regresando al paso 1.

Desde luego que en el programa deberá estar presente la declaración de la función potpos que se invoca desde el proceso de entrada 'main'.

Una posible declaración de este proceso 'main' es:

```
def_sub main() 'Encabezado de procedimiento 'main'
    dim mibte as byte
    dim base,poten,z1 as single

    iniens
    jsr lee#car 'Esto equivale a una sentencia getchar(); de C
    finens

    print "Hola desde el MCU MC9S08SH,"
    print "habilitado como dispositivo CHIPBAS8SH."
    print

    while 1
    input "mibte=",mibte
    input "base=",base
    poten=potpos(base,mibte)
    z1 = 4.*potpos(base, mibte)
    print "poten=";poten
    print "z1=";z1
    print

    wend

end_sub 'Fin de procedimiento 'main'
```

En la figura 1 se muestra el aspecto de la pantalla de la consola de interfaz cuando se ejecuta un programa cuyo proceso 'main' sea el detallado en este ejemplo. Nótese que tal como lo estipula el código de la función potpos, véase el ejemplo 2, al dar un valor inválido para el exponente implicado, se avisa esto en la consola y se detiene la ejecución.

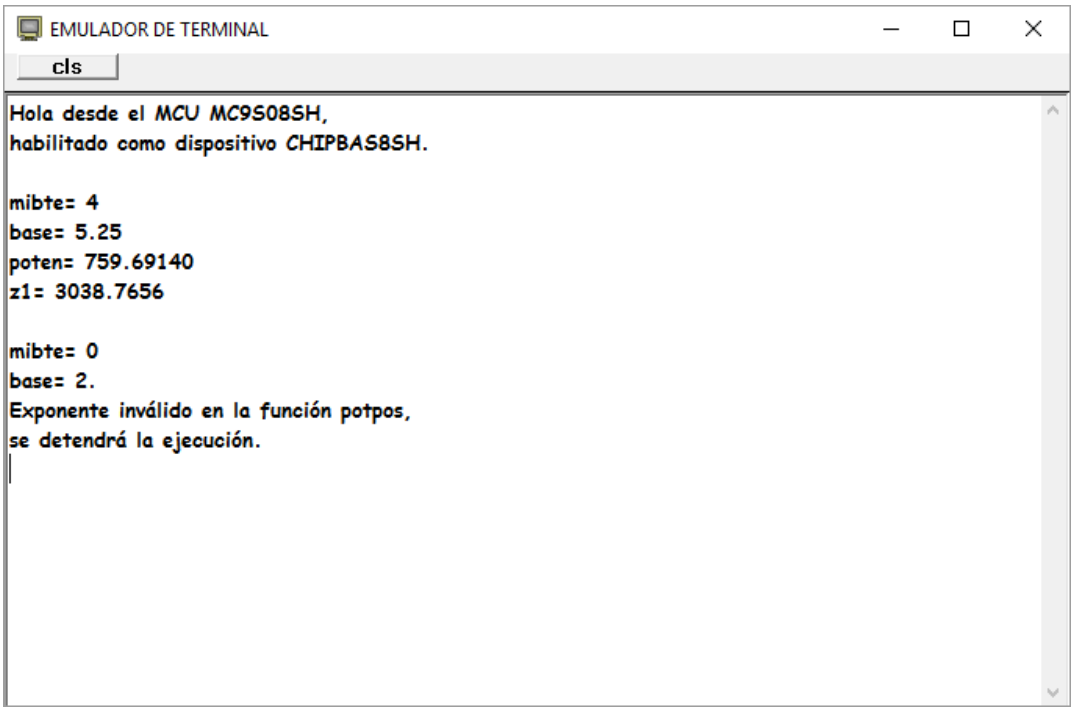

Fig 1. Pantalla de la consola de interfaz al ejecutarse un programa cuyo proceso 'main' sea mostrado en el ejemplo 5.

2.3 Formato del Bloque 3 de un programa compilable con SBAS8

Este bloque debe contener las declaraciones de todos los procedimientos adicionales al proceso 'main', que pudiera contener el programa que se esté desarrollando en un momento dado. El orden en que estas declaraciones de procesos sean colocadas es indistinto.

Para fines ilustrativos aquí se muestra una posible estructura del bloque 3, para un programa que contenga los procedimientos descritos en los ejemplos 1, 2 y 3.

Este bloque 3 podría ser:

```
    def_subint servovft1()
      dim byaux as byte
      'Código para regresar a cero la bandera TOF
      byaux = tpm1sc
      tpm1sc = tpm1sc and &h7f 'TOF <-- 0
    glip
    contms = contms + 1 'Incrementa contms
    relip

    end_subint
int#(011) servovft1 'Colocación de vector asociado
```

```
      def_fun potpos(x1 as single,m as byte) as single

       dim mipot as single

       if  m < 1 or m > 20. then

        print "Exponente inválido en la función potpos,"
        print "se detendrá la ejecución."

        iniens
        sei
        finens

        end

       endif

       mipot=1.

       for i~=1 to m
       mipot=mipot*x1
       next i~

       potpos=mipot 'Asignación de valor a retornar.

     end_fun

      def_sub retms(n as integer)

  'Este procedimiento genera una espera de n milisegundos,
  'se supone que la frecuencia del reloj de bus del MCU
  'es 20 MHz.

    if n < 1 or n > 32767 then
    'Si 'n' no está en el rango apropiado
    'retorna sin generar un retardo.
     exit_sub
    endif

        for i%=1 to n
        gosub ret1ms
        next i%

        exit_sub

ret1ms:
        iniens
        pshh
        pshx
        ldhx #$07cd
vuelta: nop
        nop
        aix #$ff
        cphx #$0000
        bne vuelta
        pulx
        pulh
        rts
        finens

        end_sub
```

2.4 Formato del Bloque 4 de un programa compilable con SBAS8

Este bloque estaría integrado por las declaraciones para la colocación en memoria de constantes de tipo numérico, esto por lo regular se hace en memoria no volátil. Las sentencias para efectuar esto iniciarían con una de las siguientes palabras reservadas: datab, dataw, datal y datas, esto respectivamente para constantes de tipo byte, integer, long y single. **En caso de que el lector no esté familiarizado con la sintaxis propia de este tipo de declaraciones, antes de seguir leyendo esta sección, véase la sección 7.10 de [1].**

Ejemplo 6. Colocación de constantes numéricas en memoria no volátil

En este ejemplo se muestra la forma que tendrían las declaraciones asociadas con la colocación en memoria no volátil del MCU de diversas constantes numéricas. Los valores y dirección de colocación indicados en cada caso solo se escogieron para fines ilustrativos. Las especificaciones de las constantes son:

- Dos constantes de tipo byte cuyos valores son: -1 y 100; siendo &ha000 la dirección inicial de colocación de éstas.
- Tres constantes de tipo integer cuyos valores son: -10, 1500 y 10000; siendo &ha080 la dirección inicial de colocación de éstas.
- Una constante de tipo long cuyo valor es: 1450000; siendo &ha0a0 la dirección inicial de colocación de ésta.
- Dos constantes de tipo single cuyos valores son: -1.5e9 y 456.25; siendo &ha0b0 la dirección inicial de colocación de éstas.

Acorde con lo explicado en la sección 7.10 de [1], las declaraciones requeridas podrían ser las siguientes:

```
datab &ha000 -1,100
dataw &ha080 -10,1500,10000
datal &ha0a0 1450000
datas &ha0b0 -1.5e9,456.25
```

Es importante destacar que las sentencias declaratorias de constantes numéricas deben colocarse a partir de la primera columna del editor, de no ser este el caso, se generaría un error y SBAS8 indicaría esto al usuario desplegando una leyenda alusiva. Por ejemplo, si en el ejemplo anterior la sentencia que inicia con la palabra reservada datab, no se coloca a partir de la primera columna del editor, SBAS8 mostraría la siguiente leyenda:

Sentencia de código colocada fuera de intervalo delimitado por
declaraciones de apertura y cierre de un procedimiento --> datab &ha000 -1,100

Es responsabilidad del programador el que no haya intersecciones entre las direcciones de colocación de las constantes numéricas, y que estás queden fuera de las direcciones donde estaría colocado el código del programa. Para averiguar el intervalo de colocación del código asociado con un determinado programa bastaría compilarlo sin la opción de ejecución, si no hubiere errores, SBAS8 podría mostrar al usuario un dialogo como el mostrado en la figura 2. Ahí se aprecia que el intervalo de

colocación del código del programa es [&h8000, &h86DE]. Por lo tanto, si en el programa en cuestión se colocaran en memoria las constantes numéricas del ejemplo 6, no habría ningún problema. En la sección 4.1 de esta guía se explica como compilar un programa bajo SBAS8 sin la opción de ejecución.

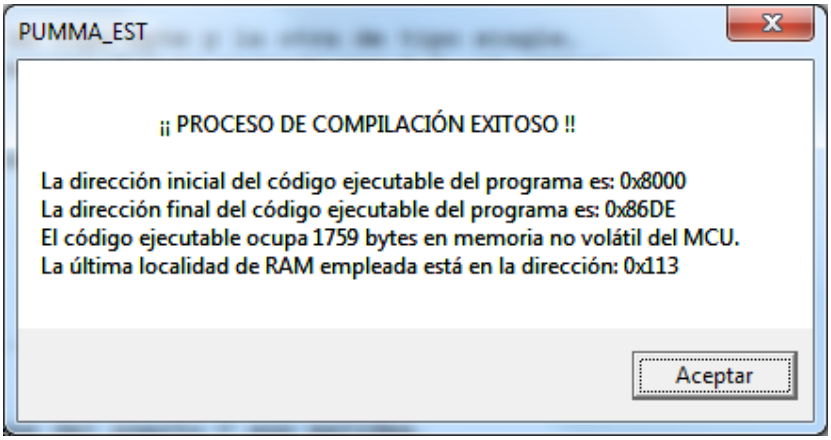

Fig 2. Dialogo presentado por SBAS8 después de que se ha compilado un programa para el cual no hay errores.

3. Definición de constantes globales numéricas de solo lectura (CGN)
SBAS8 contempla la posibilidad de poder definir constantes numéricas que residirán en memoria, por lo regular no volátil. Para habilitar éstas en un programa determinado, se deberá hacer lo siguiente:

- En el bloque 1 del programa se deberá definir el nombre de la constante y la dirección de memoria donde estará colocada, para ello simplemente se define ésta como si fuera una variable global especial, véase la sección 2.1 de esta guía.

- En el bloque 4 del programa, se hace lo propio para que el valor de las constantes implicadas queden colocadas en memoria en direcciones que estén fuera del intervalo de colocación del código del programa en cuestión, véase la sección 2.4. Estas direcciones deben ser las que se usen en la definición que se haga para el nombre y dirección de cada constante en el bloque 1.

Ejemplo 7. Definición de constantes globales bajo SBAS8
Aquí se ilustra la definición de dos constantes globales que van a usarse en un determinado programa. Se muestran las sentencias que han de ponerse en los bloques 1 y 4, para los fines de este ejemplo. Las constantes se denotarán como pi y gt, cuyos valores se desea sean respectivamente 3.141592 y 9.80665. Se desea que estas constantes queden colocadas en la memoria a partir de la dirección &hd700, **desde luego que, se deberá checar que no haya intersección entre el intervalo de colocación en memoria del código del programa en cuestión, y las direcciones de colocación de las constantes,** véase el ejemplo 6.

Las sentencias a colocar al inicio del bloque 1 del programa deberán ser:

```
defvarsptr pi &hd700
defvarsptr gt &hd704
```

Las sentencias a colocar en el bloque 4 del programa deberán ser:

```
datas &hd700 3.141592
datas &hd704 9.80665
```

O bien, considerando el hecho de que las constantes estarán en direcciones subsecuentes de memoria, podría usarse una sola sentencia en el bloque 4, esta sería:

```
datas &hd700 3.141592,9.80665
```

4. Ejemplos de programación en BASIC bajo SBAS8

Aquí se muestran ejemplos ilustrativos de programas completos compilables con SBAS8. En esta sección se supone que el MCU destino, donde se ejecutarán los programas, es el chip MC9S08SH32 habilitado como dispositivo CHIPBAS8SH y presente en la tarjeta para desarrollo MINICON_08SH. El sistema para desarrollo empleado para fines de la compilación y carga de los programas en el MCU destino es el denominado como AIDA08SH y descrito en la referencia [3]. En la figura 3 se muestra un esquema del sistema AIDA08SH.

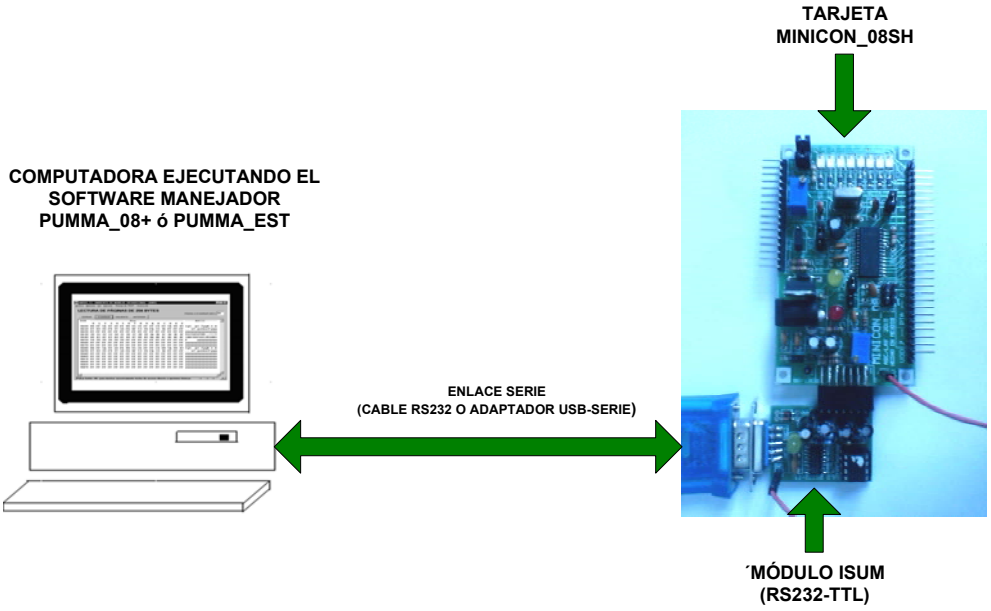

Fig 3. Sistema para desarrollo y aprendizaje AIDA08SH

Antes de operar el sistema AIDA08SH para fines de los ejemplos aquí presentados, se requiere preconfigurar en el software manejador PUMMA_EST lo siguiente:

- El MCU destino debe ser el chip MC9S08SH32 con firmware de base que lo habilita como dispositivo CHIPBAS8SH. Esto está explicado en las secciones 10 y 4 de [3].

- Habilitar el hecho de que el perfil de memoria del MCU destino es el propio de un dispositivo CHIPBAS8SH. Esto se explica en la sección 9.1 de [3].

Ejemplo 8. Programa 'HOLA MUNDO'
Este programa despliega el texto 'HOLA MUNDO' en la pantalla del emulador de terminal que habilite la consola de interfaz, en un momento dado. Para fines de la prueba del mismo se podría usar el emulador de terminal presente en el software manejador PUMMA_EST. El texto asociado con el programa se puede capturar en el editor de PUMMA_EST; o bien, también podría hacerse lo propio empleando el NotePad de Windows y guardando el texto con la extensión '.b', para después ser abierto con el editor de PUMMA_EST.

El código fuente del programa de este ejemplo podría ser:

```
end_codini 'Fin de código de inicialización.

def_sub main()

  iniens
  jsr lee#car 'Esto equivale a la sentencia getchar(); en C.
  finens

  print "HOLA MUNDO"

end_sub
```

Nótese que debido a que el programa no contempla variables globales, y no hay recursos del MCU que requieran ser inicializados, el bloque 1 contiene únicamente a la sentencia 'end_codini'. En la figura 4 se muestra la ventana del editor de PUMMA_EST, una vez que el programa de este ejemplo ha sido capturado y guardado en el archivo hola_mundo.b.

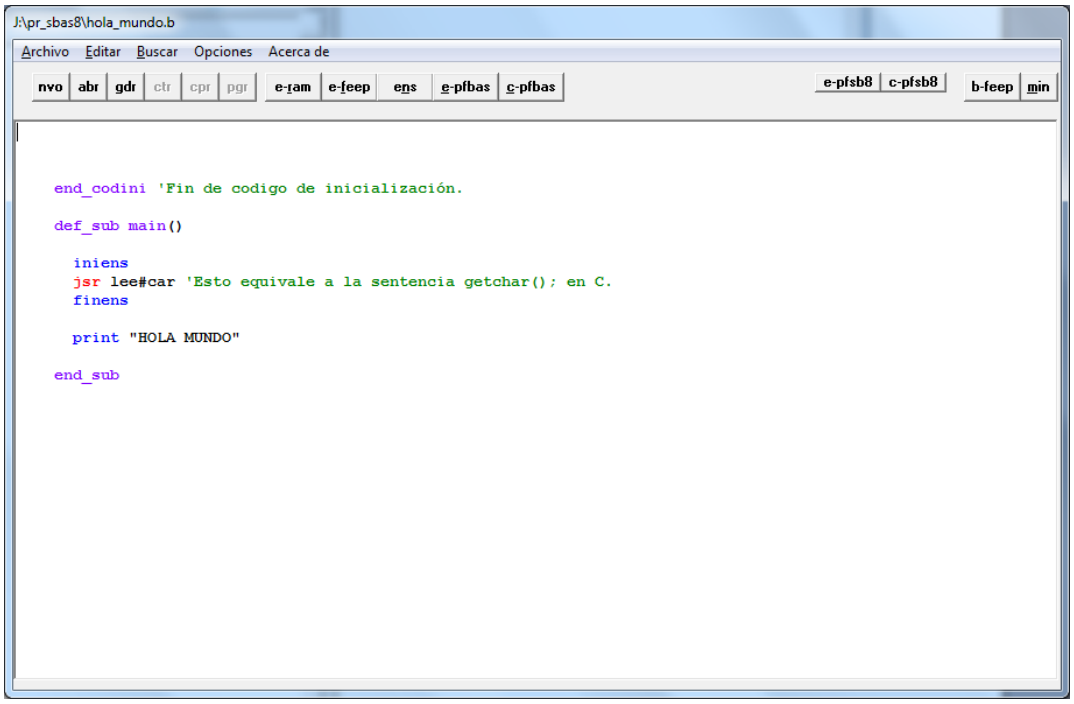

Fig 4. Programa hola_mundo presente en la ventana del editor de PUMMA_EST

El programa de este ejemplo se almacenó en el archivo hola_mundo.b, el cual está presente en la carpeta de ejemplos que forma parte de la información básica acerca del sistema AIDA08SH, véase el archivo Info_AIDA08SH_2.rar, descargable desde dctrl.fi-b.unam.mx/~salva.

4.1 Compilación y ejecución de programas bajo SBAS8

A continuación se detallan los pasos a seguir, para compilar y ejecutar en el MCU un programa en BASIC, bajo el paradigma de programación soportado por el compilador SBAS8. Para apoyo de las explicaciones de esto se usará el programa del ejemplo 8.

4.1.1 Compilación sin opción de ejecución

Para compilar un programa, sin que éste se cargue y ejecute en el MCU destino, se requiere que su texto esté presente en la ventana del editor, y que éste ya haya sido guardado en un archivo denominado como 'np.b'; donde np es el nombre que el programador le asigne al programa.
Al oprimirse el botón **<c-pfsb8>**, presente en la ventana del editor, se inicia el proceso de compilación, si no hay errores, al terminar la compilación se mostrará un dialogo indicando que la compilación fue exitosa; ahí se indica además la siguiente información:

- Direcciones inicial y final donde se colocará el código del programa en la memoria no volátil del MCU.
- El tamaño en bytes del código del programa.
- La dirección de la última localidad de memoria RAM utilizada por el programa.

La dirección inicial de RAM utilizada puede verse cuando se efectúa el perfilamiento de la memoria del chip destino. Véase la sección 9.1 de [3]. Para un dispositivo CHIPBAS8SH la dirección inicial de RAM utilizada por SBAS8 es &h0087.
En caso de que se presenten errores, éstos son detallados al usuario para que sean corregidos.

Por ejemplo, si en la ventana del editor está presente el programa del ejemplo 8, después de oprimir el botón **<c-pfsb8>** transcurrirán algunos segundos después de lo cual aparecerá el diálogo mostrado en la figura 5.

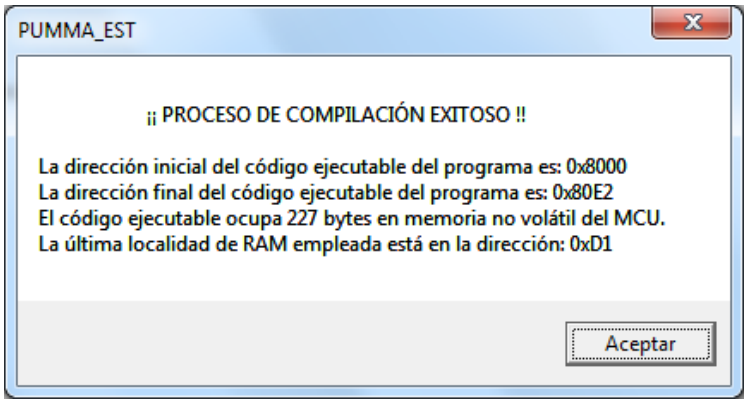

Fig 5. Dialogo presentado al usuario después que se ha compilado el programa del ejemplo 8.

El programa del ejemplo 8 se almacenó en el archivo hola_mundo.b, el cual está presente en la carpeta de ejemplos que forma parte de la información básica acerca del sistema AIDA08SH, véase el archivo Info_AIDA08SH_2.rar, descargable desde dctrl.fi-b.unam.mx/~salva.

4.1.2 Compilación con opción de ejecución

Para compilar un programa, y si no hubo errores ejecutar éste en el MCU destino, se requiere que su texto esté presente en la ventana del editor, y que éste ya haya sido guardado en un archivo denominado como 'np.b'; donde np es el nombre que el programador le asigne al programa.
Al oprimirse el botón **<e-pfsb8>**, presente en la ventana del editor, se inicia el proceso de compilación, si no hay errores, al terminar la compilación PUMMA_EST cargará el código de máquina asociado en el MCU destino para ejecutar de inmediato el programa. Una vez que se haya iniciado la ejecución, PUMMA_EST indicará esto al usuario mediante el diálogo mostrado en la figura 6.

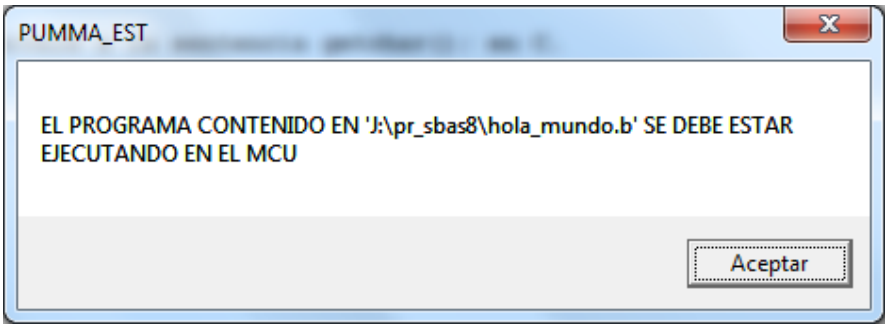

Fig 6. Indicación al usuario acerca de que el programa recién compilado y cargado se está ejecutando en el MCU destino.

Si el programa presente en la ventana del editor es el del ejemplo 8, para verificar su funcionalidad, después de oprimir el botón aceptar en el dialogo de la figura 6, se deberá abrir el emulador de terminal de PUMMA_EST, para ello simplemente se pulsa el submenú *invocar emulador de Terminal*, presente en el menú archivo de la ventana del editor, esto hará que se muestre al usuario la ventana propia del emulador de Terminal, véase la figura 7.

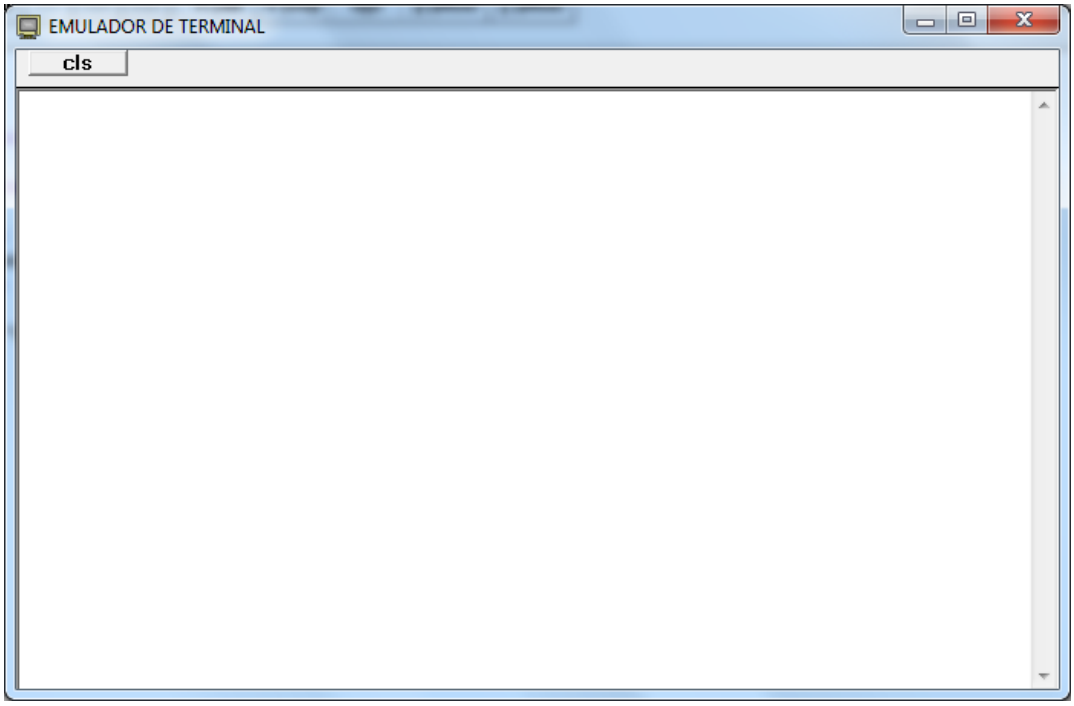

Fig 7. Ventana del emulador de terminal al abrirse éste.

Después de que se ha abierto el emulador de Terminal, se deberá oprimir una tecla en la PC donde corre el software manejador, después de esto en la pantalla del emulador de terminal deberá mostrarse el mensaje 'HOLA MUNDO', véase la figura 8.

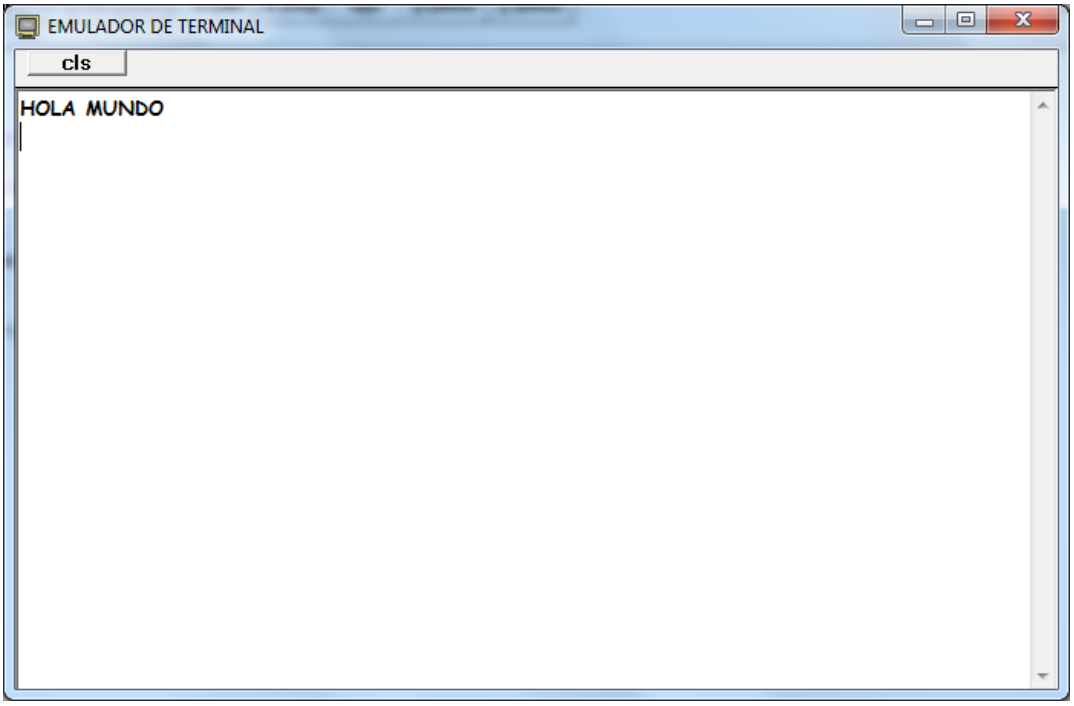

Fig 8. Pantalla del emulador de Terminal al ejecutarse el programa hola_mundo, después que se ha oprimido una tecla

Ejemplo 9. Semáforo simple de seis luces

Se desea validar con la tarjeta MINICON_08SH el control de un semáforo de seis luces dividido en dos unidades denominadas U1 y U2. El MCU generaría las señales lógicas de control para cada uno de los dispositivos de iluminación, desde luego que se requeriría una interfaz de potencia entre las líneas de puerto del MCU asociadas con cada lámpara y el hardware propio de éstas. A las señales de control propias de la U1 se les denota como R1, A1 y V1 y se validan respectivamente con las líneas de puerto: ptc2, ptc1 y ptc0. Por otra parte, a las señales de control propias de la U2 se les denota como R2, A2 y V2 y se validan respectivamente con las líneas de puerto ptc5, ptc4 y ptc3. Se supone que la lógica de interfaz de potencia hace que cada lámpara encienda cuando su señal de control asociada es uno lógico y apague en el otro caso.

Se requiere que en cada unidad el "siga" y el "alto" duren un minuto, y que la luz "preventiva" dure cinco segundos. Es fácil ver que con una secuenciación repetible de cuatro estados de seis bits se realizaría el ciclo básico de funcionamiento del semáforo. En la tabla 1 se muestran los cuatro estados implicados. Se supone que al inicio del ciclo la unidad uno debe marcar alto.

Tabla1 Secuencia de estados para el semáforo del ejemplo 9

Estado	Ptc7 *	Ptc6 *	Ptc5 (R2)	Ptc4 (A2)	Ptc3 (V2)	Ptc2 (R1)	Ptc1 (A1)	Ptc0 (V1)	HEX	Tiempo
1	0	0	0	0	1	1	0	0	0C	60 seg.
2	0	0	0	1	0	1	0	0	14	5 seg.
3	0	0	1	0	0	0	0	1	21	60 seg.
4	0	0	1	0	0	0	1	0	22	5 seg.

Un posible programa que realiza el control del semáforo de este ejemplo se muestra a continuación:

```
'Programa semaforo.b
'Valida un semáforo de seis luces,
'usa el procedimiento tipo sub retms(n)

defvarbptr portsem &h4 'Puerto de control es ptcd
defvarbptr ptcdd &h5

   ptcdd = &h3f 'pta5 a pta0 son salidas

  end_codini 'Fin del código de inicialización.

  def_sub main()

    while 1
    portsem=&h0c
    retms(30000)
    retms(30000)
    portsem=&h14
    retms(5000)
    portsem=&h21
    retms(30000)
    retms(30000)
    portsem=&h22
    retms(5000)
    wend

  end_sub
```

```
  def_sub retms(n as integer)

  'Este procedimiento genera una espera de n milisegundos,
  'se supone que la frecuencia del reloj de bus del MCU
  'es 20 MHz.

    if n < 1 or n > 32766 then
    'Si 'n' no está en el rango apropiado
    'retorna sin generar un retardo.
     exit_sub
    endif

        for i%=1 to n
        gosub ret1ms
        next i%

        exit_sub

ret1ms:
        iniens
        pshh
        pshx
        ldhx #$07cd
vuelta: nop
        nop
        aix #$ff
        cphx #$0000
        bne vuelta
        pulx
        pulh
        rts
        finens

   end_sub
```

Nótese el uso del procedimiento tipo 'sub' nombrado como retms, cuya funcionalidad se explicó en el ejemplo 1 de esta guía. Además es destacable el hecho de que el procedimiento aquí usado para generar los retardos requeridos, puede usarse en cualquier otro programa que lo pudiera requerir.
El programa de este ejemplo se almacenó en el archivo semaforo.b, el cual está presente en la carpeta de ejemplos que forma parte de la información básica acerca del sistema AIDA08SH, véase el archivo Info_AIDA08SH_2.rar, descargable desde dctrl.fi-b.unam.mx/~salva.

Ejemplo 10. Medición del tiempo de duración de un evento
Se desea medir la duración de un evento; el inicio de éste se testifica oprimiendo y soltando un botón normalmente abierto conectado al bit pta0 del MCU, el final del evento se testifica oprimiendo y soltando otro botón que está ligado al bit pta1 del MCU. Una vez efectuada la medición, el resultado de ésta se deberá desplegar en el emulador de Terminal de PUMMA_EST en términos de segundos y milésimas; después de esto se pasará a estar en espera por la medición de la duración de otro evento, repitiéndose este ciclo indefinidamente. En la figura 9 se muestra el diagrama de tiempos asociado con las señales en los bits pta0 y pta1 del MCU, 'Te' es la duración a medir. Si el tiempo 'Te' llegara a rebasar los 30 segundos se debe notificar esto al usuario. En la figura 10 se muestra el conexionado a la tarjeta MINICON_08SH, como

parte del sistema para desarrollo AIDA08SH, para probar el programa que se diseñe para la realización de esta aplicación demostrativa.

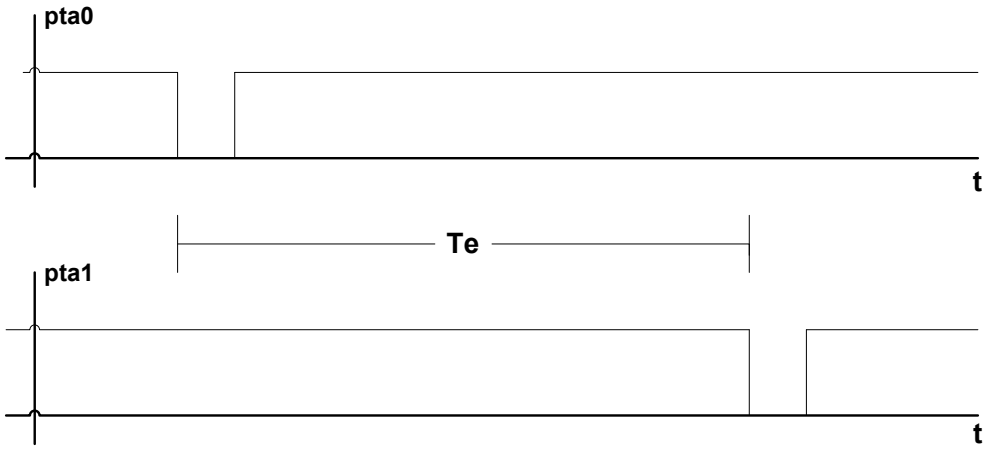

Fig 9. Señales en los bits pta0 y pta1 al efectuarse una medición del tiempo Te

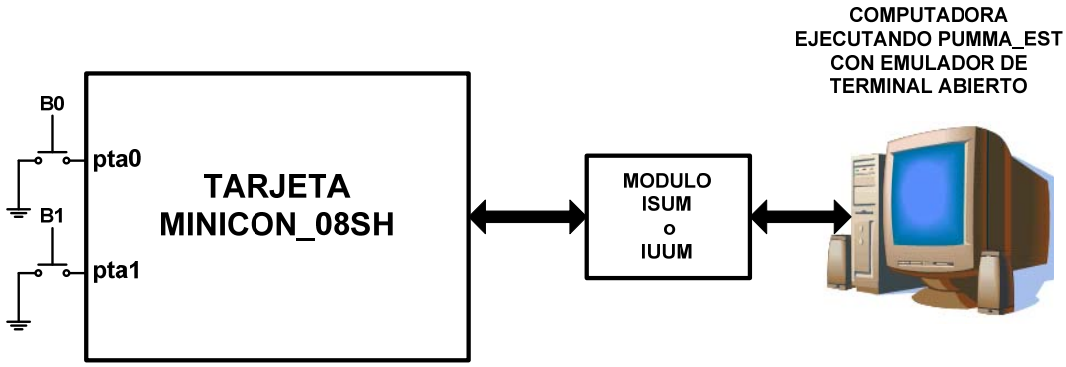

Fig 10. Conexionado requerido para verificar la funcionalidad del programa asociado con la aplicación del ejemplo 10.

Para realizar esta aplicación se usará la interrupción por sobreflujo del temporizador 1 del MCU, inicializando lo propio para que se de una interrupción de este tipo cada milisegundo, en la rutina de servicio simplemente se incrementa una variable global, denominada como contms, que cuenta los milisegundos transcurridos a partir de que se oprimió el botón de arranque (B0). Si el resultado del incremento es mayor que 30000 (30 segundos), se testifica esto asignando '-1' a la variable global que cuenta los milisegundos, y se deshabilita la instancia de interrupción empleada. Además del código de inicialización y el procedimiento 'main' el programa propuesto cuenta con un procedimiento de tipo 'subint' que valida la rutina de servicio de interrupción requerida. En la tabla 5-2 de la página 64 de [2], se aprecia que once es el número de la instancia de interrupción usada en esta aplicación.

Para una mejor comprensión del código en la rutina de servicio de interrupción se sugiere ver la sección 7.12 de [1], aclarando aquí que bajo el paradigma de programación soportado por el compilador SBAS8, no es debe colocar en un procedimiento de tipo 'subint' las sentencias 'retint'o rti, ya que éstas se generan al procesarse las sentencias exit_subint y end_subint. Si se llegarán a poner las

senetencia ‘retint’ o ‘rti’ en una rutina de servicio de interrupción se desincroniza la pila (stack), lo que origina un fatal mal funcionamiento del programa.

En el procedimiento ‘main’ del programa en síntesis se efectúan cíclicamente las siguientes acciones:

1. Se inicializa con cero la variable global ‘contms’ que cuenta milisegundos.

2. Se pasa a un lazo de espera por la opresión del botón de arranque (B0). Una vez que se detecta esto se habilita la interrupción empleada y se avisa al usuario que inició el evento.

3. Se pasa a un lazo de espera por la opresión del botón de paro (B1). Una vez que se detecta esto se deshabilita la interrupción empleada y se avisa al usuario que finalizó el evento.

4. Se realiza la aritmética requerida para desplegar el tiempo transcurrido, en el formato requerido por la aplicación.

5. Se pasa al paso uno.

El programa asociado con este ejemplo se muestra a continuación:

```
''********** Programa med_te_evento *************
'Mide el tiempo que transcurre entre la opresión
'de dos botones denominados B0 y B1 ligados
'respectivamente con los bits pta0 y pta1 del MCU.
'B0 es el botón de arranque del contador de tiempo.
'B1 es el botón de paro del contador de tiempo.
'Se usa la interupción de sobreflujo del temporizador 1
'del MCU de modo que el requerimiento de interrupción
'asociado se de cada milisegundo.

'Por: Antonio Salvá Calleja
'Fecha: 22/abril/2016

' ***** CÓDIGO DE INICIALIZACIÓN *****

'Declaraciones de variables globales especiales de usuario.
defvarbptr ptad &h0
defvarbptr ptape &h1840
defvarbptr tpm1sc &h20
defvarbptr tpm1modh &h23
defvarbptr tpm1modl &h24
'.............................................
'Declaración de la variable global contms que
'será el contador de milisegundos.
dim contms as integer
'.............................................

 ptape=&h3 'pta0 y pta1 tienen 'pull-up'

 tpm1sc=&h08 'pe=1,toie=0
 tpm1modh=&h4e 'Tovf = 1 ms
 tpm1modl=&h1f
```

```
      iniens
      cli     'Se habilitan globalmente interrupciones
      finens

   end_codini 'Fin de código de inicialización

   def_sub main()

      iniens
ptad@ equ $00
tpm1sc@ equ $20
       jsr lee#car
      finens


      while 1
      contms=0

      print "En espera de inicio de evento"
      print

      iniens
espb0: brset 0,ptad@,espb0
        bset 6,tpm1sc@ 'toie <-- 1
      finens

      print "Inició evento"

      iniens
espb1: brset 1,ptad@,espb1
        bclr 6,tpm1sc@ 'toie <-- 0
      finens

      print "Finalizó evento"

      if contms<0 then
      'Si hubo saturación en el tiempo,
      'se notifica esto al usuario.
      print "La duración del evento rebasó los 30 segundos"
      print
      else
      segs%= contms/1000
      milisegs%= contms mod 1000
      print "Te = ";segs%;" segundos y ";milisegs%;" milésimas"
      print
      endif

      print "Oprimir una tecla para efectuar otra medición"

      iniens
      jsr lee#car
      finens

     wend

   end_sub
```

```
def_subint servovft1()
      dim byaux as byte

       glip 'Se requiere porque hay código BASIC
            'en la subrutina de servicio.

      'Código para regresar a cero la bandera TOF
      byaux = tpm1sc
      tpm1sc = tpm1sc and &h7f 'TOF <-- 0
      '.....................................
      contms=contms+1 'Incrementa contador de milisegundos

      if contms>30000 then
       'Si el tiempo es mayor que 30 segundos,
       'testifica esto asignando un valor negativo,
       'y deshabilita localmente la interrupción.
       contms=-1
       tpm1sc = tpm1sc and &hbf 'toie <-- 0
      endif

      relip 'Se requiere porque hay código BASIC
            'en la subrutina de servicio.

    end_subint

int#(011) servovft1 'Colocación de vector asociado
```

En la figura 11 se muestra la ventana del emulador de terminal al ejecutarse el programa de este ejemplo, ahí se aprecian dos mediciones, para la segunda hubo saturación y se indicó esto al usuario.

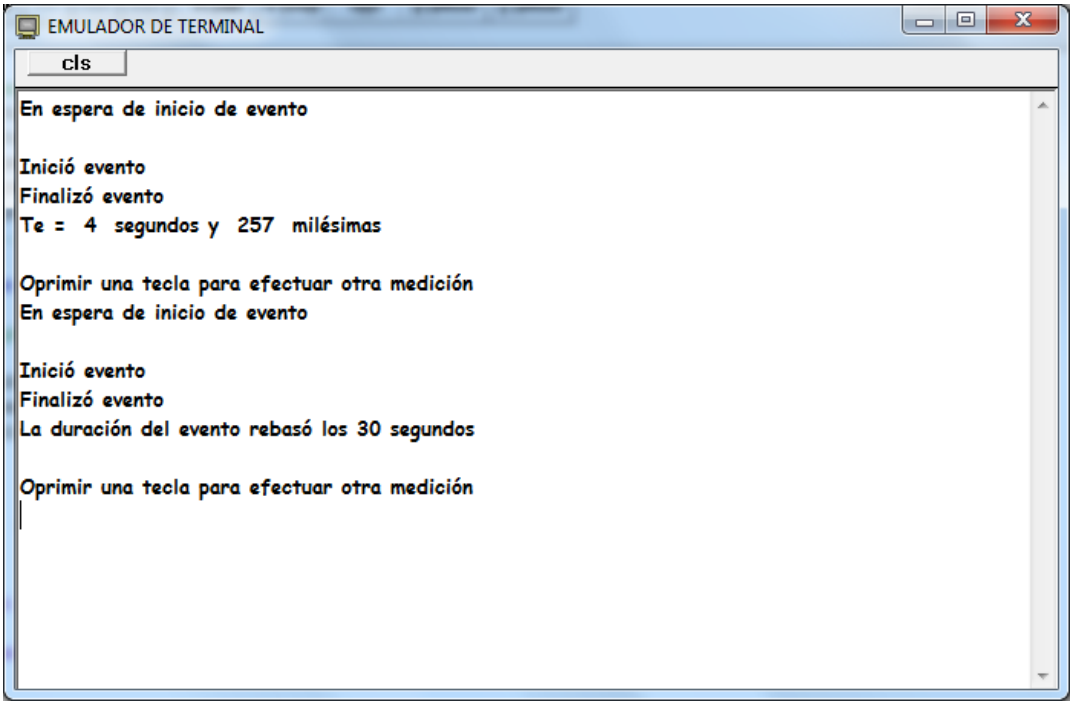

Fig 11. Ventana del emulador de terminal al ejecutarse el programa del ejemplo 10

El programa de este ejemplo se almacenó en el archivo **med_te_evento**.b, el cual está presente en la carpeta de ejemplos que forma parte de la información básica acerca del sistema AIDA08SH, véase el archivo Info_AIDA08SH_2.rar, descargable desde dctrl.fi-b.unam.mx/~salva.

5. Ejecución autónoma en dispositivos CHIPBAS8 bajo SBAS8

Los lineamientos a seguir para ejecutar en forma autónoma un programa en un dispositivo CHIPBAS8 en general son los mismos para las diversas versiones de éstos. Aquí se ejemplifica este concepto para dispositivos CHIPBAS8SH como es el caso del MCU presente en la tarjeta MINICON_08SH. Estos lineamientos pueden verse en la sección 8 de [1]; sin embargo, considerando el paradigma de programación de SBAS8 un programa ejecutable en forma autónoma deberá contemplar lo siguiente:

- Como primera sentencia del código de inicialización se deberá escribir lo propio en el registro 'sopt1' del MCU, esto acorde con el hecho de que se use o no el COP (watch dog) y la inicialización de otras funcionalidades básicas. Véase lo que se describe como bloque 1 en la sección 8 de [1]; si se usa el COP, véase lo propio acerca de éste y del registro 'sopt2' en el capítulo 5 de [2]. **Si la aplicación no usa el COP, como es frecuente el caso, simplemente bastará con escribir al registro sopt1 el byte &h21 si se desea que el pin de reset funcione, en otro caso el byte a escribir es &h20.**

En lo que toca a lo que se describe como bloque 3 en la sección 8 de [1], ya no es necesario poner al final del programa el código ahí descrito, esto debido a que SBAS8 por defecto siempre coloca el vector de reset de usuario (VRA), en las direcciones de memoria que correspondan al dispositivo CHIPBAS8 que se use en un momento dado.

Así, para ejecutar en forma autónoma un programa, compilable con SBAS8, en el dispositivo CHIPBAS8SH presente en la tarjeta MINICON_08SH, lo único que se debe añadirle es la escritura al registro 'sopt1' al inicio del código de inicialización y para su ejecución se deben seguir los siguientes pasos:

1.- Con el *jumper* JE1 en su posición default (MON), cargar y ejecutar el programa con el formato aquí descrito; empleando para ello facilidades de PUMMA_EST.

2.- Desenergizar la tarjeta MINICON_08SH.

3.- Pasar el *jumper* JE1 a su posición alterna (AUT).

Al energizar de nuevo la tarjeta se ejecutará de manera autónoma el programa implicado.

Para otro dispositivo CHIPBAS8 presente en otra tarjeta de desarrollo MINICON el proceso sería el mismo, con las variantes propias debidas a variantes en el hardware propio de cada caso específico.

Ejemplo 11. Ejecución autónoma del programa del ejemplo 8

Para fines ilustrativos se presenta aquí el programa del ejemplo 8 con el agregado requerido y descrito en la sección 5. Se supone que la aplicación no usa el COP. El programa es el siguiente:

```
'Programa semaforo_aut
'Valida un semáforo de seis luces,
'usa el procedimiento tipo sub retms(n),
‘configurado para ejecución autónoma.

defvarbptr portsem &h4 'Puerto de control es ptcd
defvarbptr ptcdd &h5
defvarbptr sopt1 &h1802

   sopt1=&h21 'Se deshabilita COP y se habilita pin de reset.
   ptcdd = &h3f 'pta5 a pta0 son salidas

  end_codini 'Fin del código de inicialización.

  def_sub main()

    while 1

    portsem=&h0c
    retms(30000)
    retms(30000)
    portsem=&h14
    retms(5000)
    portsem=&h21
    retms(30000)
    retms(30000)
    portsem=&h22
    retms(5000)

    wend

  end_sub

  def_sub retms(n as integer)

  'Este procedimiento genera una espera de n milisegundos,
  'se supone que la frecuencia del reloj de bus del MCU
  'es 20 MHz.

    if n < 1 or n > 32766 then
    'Si 'n' no está en el rango apropiado
    'retorna sin generar un retardo.
     exit_sub
    endif

        for i%=1 to n
        gosub ret1ms
        next i%

        exit_sub

ret1ms:
        iniens
        pshh
        pshx
        ldhx #$07cd
vuelta: nop
        nop
        aix #$ff
```

```
        cphx #$0000
        bne vuelta
        pulx
        pulh
        rts
        finens

    end_sub
```

Para programas más grandes el proceso aquí descrito para fines de la ejecución autónoma de éstos es el mismo aquí descrito.
El programa de este ejemplo se almacenó en el archivo semáforo_aut.b, el cual está presente en la carpeta de ejemplos que forma parte de la información básica acerca del sistema AIDA08SH, véase el archivo Info_AIDA08SH_2.rar, descargable desde dctrl.fi-b.unam.mx/~salva.

6. Inclusión de código fuente prealmacenado en archivo de texto plano
SBAS8 y MINIBAS8A contemplan la inclusión de código fuente BASIC que haya sido prealmacenado en un archivo de texto plano. La sintaxis para efectuar esto se muestra a continuación:

#include "Denotación de archivo de texto plano que contiene código fuente BASIC"

La palabra reservada '#include' se debe colocar a partir de la primera columna del editor. En la denotación del archivo, preferentemente se debe incluir la ruta que define su ubicación en la computadora donde se trabaje en un momento dado. Por ejemplo, supóngase el archivo dosprints.txt contiene el siguiente texto fuente

```
' Aquí hay dos sentencias print
                print
                print
```

y que éste está en la carpeta 'j:\hcs08\'. Si en un programa fuente, almacenado en un archivo nombrado como 'np.b', se incluye la siguiente sentencia:

#include "j:\hcs08\dosprints.txt"

Al ser procesado el programa fuente que contiene la sentencia de inclusión de código aquí descrita, previo a la compilación el archivo np.b es substituido por otro denominado np.bto, en este último ya se habrá substituido la sentencia de inclusión por el siguiente texto:

```
'*  j:\hcs08\dosprints.txt
' Aquí hay dos senetencias print

          print
          print

'**** fin de j:\hcs08\dosprints.txt
```

Finalmente el archivo np.bto es el que se compila. La sentencia de inclusión de código fuente puede ser usada tantas veces como sea necesario en un programa determinado.
Si el archivo de la sentencia de inclusión no se encuentra, se presentará al usuario el dialogo mostrado en la figura 12 y el proceso de compilación es detenido.

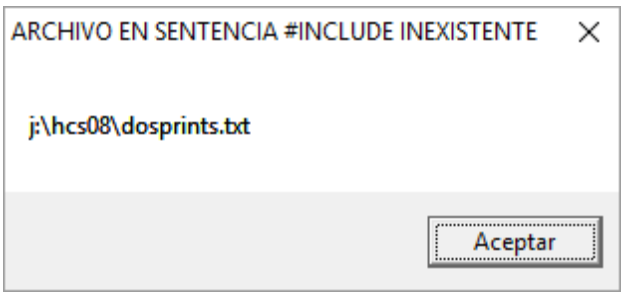

Fig 12. Forma del dialogo que avisa al usuario que el archivo de una sentencia de inclusión no existe.

La inclusión de código desde archivo aquí descrita es de mucha utilidad ya que permite, para el caso de SBAS8, el poder contar con bibliotecas de procedimientos para los cuales se haya verificado previamente su funcionalidad, de modo que éstos puedan ser usados en diversos programas fuente compilables con SBAS8. En lo que toca a MINIBAS8A, las bibliotecas serían de subrutinas previamente probadas, con uso potencial en diversos programas fuente compilables con MINIBAS8A.

Ejemplo 12. Inclusión de procedimiento para un programa
Aquí se ilustra el uso de la inclusión de código desde archivo, esto mediante su uso para simplificar el programa del ejemplo 9, que valida un semáforo de seis luces. Supóngase que en el la carpeta 'j:\hcs08\' existe un archivo denotado como 'mi_retms.bib' cuyo contenido es el código fuente de la declaración del procedimiento tipo 'sub' que genera un retardo de 'n' milisegundos, descrito en el ejemplo 1. Así, el código fuente del programa del ejemplo 9 podrá reducirse a lo siguiente:

```
'Programa semaforo_conbibret.b
'Valida un semáforo de seis luces,
'usa el procedimiento tipo sub retms(n)
defvarbptr portsem &h4 'Puerto de control es ptcd
defvarbptr ptcdd &h5
   ptcdd = &h3f 'pta5 a pta0 son salidas

  end_codini 'Fin del código de inicialización.

  def_sub main()
    while 1
    portsem=&h0c
    retms(30000)
    retms(30000)
    portsem=&h14
    retms(5000)
    portsem=&h21
    retms(30000)
    retms(30000)
    portsem=&h22
    retms(5000)
    wend
  end_sub
#include "j:\hcs08\mi_retms.bib"
```

Si el código fuente anterior se ha almacenado en un archivo denotado como semaforo_conbibret.b; para fines ilustrativos a continuación se muestra el contenido del archivo semaforo_conbibret.bto generado por SBAS8 previo a la compilación.

```
'Programa semaforo_conbibret.b
'Valida un semáforo de seis luces,
'usa el procedimiento tipo sub retms(n)
defvarbptr portsem &h4 'Puerto de control es ptcd
defvarbptr ptcdd &h5

   ptcdd = &h3f 'pta5 a pta0 son salidas

  end_codini 'Fin del código de inicialización.

  def_sub main()

    while 1
    portsem=&h0c
    retms(30000)
    retms(30000)
    portsem=&h14
    retms(5000)
    portsem=&h21
    retms(30000)
    retms(30000)
    portsem=&h22
    retms(5000)
    wend

  end_sub

'*  j:\hcs08\mi_retms.bib

def_sub retms(n as integer)
  'Este procedimiento genera una espera de ‘n’ milisegundos,
  'se supone que la frecuencia del reloj de bus del MCU
  'es 20 MHz.

    if n < 1 or n > 32766 then
    'Si 'n' no está en el rango apropiado
    'retorna sin generar un retardo.
     exit_sub
    endif

        for i%=1 to n
        gosub ret1ms
        next i%
        exit_sub

ret1ms:
        iniens
        pshh
        pshx
        ldhx #$07cd
vuelta: nop
        nop
        aix #$ff
        cphx #$0000
        bne vuelta
        pulx
```

```
        pulh
        rts
        finens

end_sub

'**** fin de j:\hcs08\mi_retms.bib
```

7. Referencias

[1] Salvá, A (2011). *MANUAL DE USUARIO DEL SISTEMA AIDA08*. Archivo muaida08ve2011.pdf, descargable desde http://dctrl.fi-b.unam.mx/~salva

[2] FREESCALE (2014), MC9S08SH32/16. Data sheet contenido en el archivo mc9s08sh32.pdf

[3] Salvá, A (2016). *MANUAL DE USUARIO DEL SISTEMA AIDA08SH*. Archivo guia_aida08sh_cn.pdf, descargable desde http://dctrl.fi-b.unam.mx/~salva