Ejemplo:

```
;ANTES DE ESCRIBIR LAS LÍNEAS DE CÓDIGO SE PUEDE "COMENTAR"
;EN ESTA SECCIÓN QUÉ ES LO QUE HACE ESTE PROGRAMA.
//ESTA VERSIÓN ES LA VER. 1.0 DESARROLLADO POR EL AUTOR.
//CON ESTE LIBRO APRENDERÁN A PROGRAMAR AVR CON FACILIDAD
;INICIAMOS...

.INCLUDE "TN2313DEF.INC"
```

Todos los comentarios se pondrán en color verde de forma automática en el AVR Studio

Estructura 1:

```
.INCLUDE "M8515DEF.INC"
.CSEG
```

```
.ORG 0x000  ;se inicia en la dirección 0x000

.
.
.
;lo_que_siga
```

Estructura 2 (cuando se desea incluir algún otro archivo dentro del código del programa):

```
.INCLUDE "M8515DEF.INC"
.INCLUDE "OTRO_PROGRAMA.ASM"
.CSEG
.ORG 0

.
.
.
;lo_que_siga
```

Estructura 3 (cuando se desea usar la directiva `.DEF/.EQU`):

```
.INCLUDE "M8515DEF.INC"
.INCLUDE "OTRO_PROGRAMA.ASM"

.DEF TEMPORAL = R17
.CSEG
.ORG 0

.
.
.
;lo_que_siga
```

Estructura 4 (cuando se desea escribir múltiples .ORG en forma consecutiva en el programa):

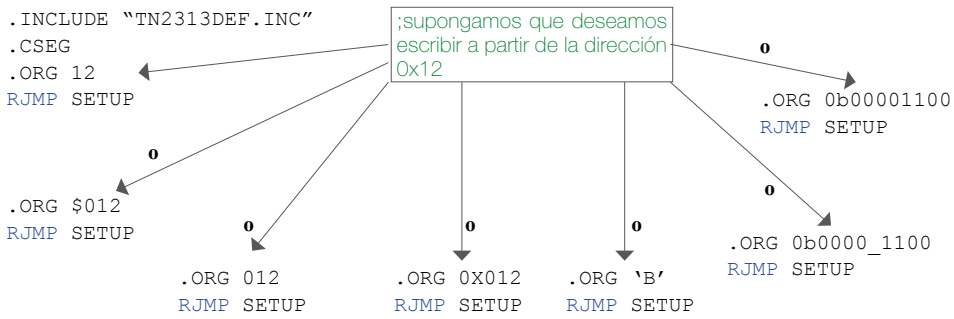
```
.INCLUDE "TN2313DEF.INC"
.CSEG
.ORG 0
;AQUÍ VA CÓDIGO

.ORG $003
;AQUÍ VA CÓDIGO

.ORG 005
;AQUÍ VA CÓDIGO

.ORG 0X00A
;AQUÍ VA CÓDIGO
```

Resumen de nomenclaturas (constantes enteras)
Los datos en AVR pueden ser escritos en:

```
LDI R16,0xFF              ;HEXADECIMAL 1
LDI R16,$FF               ;HEXADECIMAL 2
LDI R16,255               ;DECIMAL 1
LDI R16,8                 ;DECIMAL 2
LDI R16,0377              ;OCTAL
LDI R16,0b11111111        ;BINARIO 1
LDI R16,0b1111_1111       ;BINARIO 2
LDI R16,-128              ;NÚMEROS SIGNADOS
LDI R16,'A'               ;CARACTERES ASCII
```

Nota:

La sintaxis de la dirección usando **.ORG** puede ser escrita en las diferentes nomenclaturas, es decir, en decimal, hexadecimal (en sus dos formatos), octal, ASCII o binario. Lo más común es usar nomenclatura decimal o hexadecimal con la directiva .ORG (dependiendo del propósito del programador):



```
INCLUDE "8515DEF.INC"
.CSEG
.ORG 0X00

LDI R16,LOW(RAMEND)
OUT SPL,R16
LDI R16,HIGH(RAMEND)
OUT SPH,R16
```

ES POSIBLE ESCRIBIR:

```
.ORG 0
.ORG $0
.ORG 0X00
.ORG 0X000
.ORG 0X0000
```

En esta sintaxis la directiva `.CSEG` indica que todo lo escrito debajo será fijado en el código del programa

Nótese que se escribieron las instrucciones en color azul, sin embargo en lo futuro se escribirán en color negro por cuestiones prácticas en la edición de este libro

```
.INCLUDE "M8515DEF.INC"
.CSEG
.DB "VER. 1 ELABORADO POR ING. FULANITO"

LDI R16,LOW(RAMEND)
OUT SPL,R16
LDI R16,HIGH(RAMEND)
OUT SPH,R16

LDI R16,$40
.
.
.
;LO_QUE_SIGA
```

En este ejemplo no se escribió la directiva `.ORG`, entonces después del SP lo que se escriba se recorre dentro de la memoria de la Flash. Si se usara la directiva `.ORG` es posible que surja un mensaje de advertencia (warning) de `invalid opcode`

```
.
.
.
DEC R2
DEC R3
NOP
DEC R4
.ORG $FF0
.DB '"HOLA MUNDO CÓMO HAS ESTADO?"
```

En esta aplicación se usará .DB en .CSEG:

```
.INCLUDE "M8515DEF.INC"
.CSEG
```

MENSAJE:

```
.DB 0,1
```

La palabra MENSAJE está escrita una línea arriba de los datos .DB 0,1… pero también puede ser escrito en la misma línea:

```
MENSAJE: .DB 0,1
```

```
.
.
.
;LO QUE SIGA

FIN:
RJMP FIN
```

```
.INCLUDE "M8515DEF.INC"
.CSEG
MENSAJE:.DB 0,1,4
.
.
.
CICLO: RJMP CICLO
```

Nótese que en MENSAJE existen tres valores, es decir un número impar de expresiones, por lo que la memoria del programa al momento de simular agregará el valor 0x00

```
.CSEG
DATOS: .DB 0,255,0b01010101,-128,0xAA, 'b',$55,0334
```

```
;PROGRAMA VERIFICADOR DE TABLA DE DATOS

.INCLUDE "M8515DEF.INC"
.CSEG

MENSAJE: .DB 1,2,3,4

INICIO:
LDI R16, LOW(RAMEND)
OUT SPL,R16
LDI R16, HIGH(RAMEND)
OUT SPH,R16

LDI R16,$FF
OUT DDRB,R16

LDI ZH,HIGH(2*MENSAJE)
LDI ZL,LOW(2*MENSAJE)

CARGAR_BYTE:

LPM
OUT PORTB, R0
ADIW ZL,1
RJMP CARGAR_BYTE

FIN: RJMP FIN
```

```
.CSEG
LISTA_DE_VARIABLES: .DW 0, 0xffff, 0b1001110001010101,
-32768, 65535
```

Para el caso del número binario se puede escribir de estas formas:

```
0b1001110001010101
0b10011100_01010101
0b1001_1100_0101_0101
```

En esta aplicación se usará `.DD` y `.DQ` en `.CSEG`: el uso de `.DD` es empleada para una lista de expresiones de 32-bits cada uno (desde `-2147483648` hasta `4294967295`), y `.DQ` es usada para una lista de expresiones de 64-bits cada uno (desde `-9223372036854775808` hasta `18446744073709551615`).

```
.CSEG
LISTA: .DD 0, 0xABCFEDBC, -2147483648, 4294967295, 1 << 30

.CSEG
LISTA: .DQ 0, 0xABCDEFFEDBCBBACD, -9223372036854775808, 1 <<62
```

```
.INCLUDE "M8515DEF.INC"

.ESEG
MENSAJE_EEPROM: .DB 0,1,2,3,4,5,6,7,8,9,1,0,11

.CSEG
INICIO:

FIN: RJMP FIN
```

```
LDI ZH,HIGH(2*MENSAJE_EEPROM)
LDI ZL,LOW(2*MENSAJE_EEPROM)
CARGAR_BYTE:
LPM
OUT PORTB, R0
ADIW ZL,1
RJMP CARGAR_BYTE
```

Éste es el programa que deberá editar en el AVR Studio:

```
;PROGRAMA VERIFICADOR DE TABLA DE DATOS PARA .ESEG
.INCLUDE "M8515DEF.INC"

.ESEG
MENSAJE_EEPROM: .DB 0,1,2,3,4,5,6,7,8,9,1,0,11

.CSEG

INICIO:
LDI R16, LOW(RAMEND)
OUT SPL,R16
LDI R16, HIGH(RAMEND)
OUT SPH,R16

LDI R16,$FF
OUT DDRB,R16

LDI R16,0          ;para que inicialice la dirección de la
                   ;EEPROM en 0x00

CICLO:
RCALL READ_EEPROM

OUT PORTB, R30     ;R30 contiene el dato para sacar LECTURA

RCALL UN_SEGUNDO
RCALL UN_SEGUNDO
INC R16
RJMP CICLO
```

Estas subrutinas de `1_SEGUNDO` se encuentran explicadas en otro programa más adelante (Programa_5)

```
;*****************************
;ESTA SUBRUTINA DE EEPROM SE ENCUENTRA EN LA SECCIÓN
;DE EEPROM EN ESTE LIBRO
;PÁGINAS MÁS ADELANTE
READ_EEPROM:
SBIC EECR,1         ;SENSO SI ESTÁ LISTO
RJMP READ_EEPROM

OUT EEARL,R16       ;1) PRIMERO LA DIRECCIÓN DESEADA A LEER

LDI R17,$01
OUT EECR,R17        ;2) ACTIVAR EL Bit DE LECTURA EN EL REG
                    ;DE CONTROL DE LA EEPROM

IN R30,EEDR         ;3) LEER EL DATO

NOP                 ;4) 4 CICLOS DE RELOJ DE DELAY
NOP
NOP
NOP
RET
```

Caso 1:

```
.INCLUDE "M8515DEF.INC"
.CSEG

LDI R20,'A'
STS 0X70,R20        ;el registro R20 almacena el dato a ser
                    ;escrito en la dirección 0x70 (VER FIGURA
                    ;3.12)
.
.
.

;LO_QUE_SIGA:
```

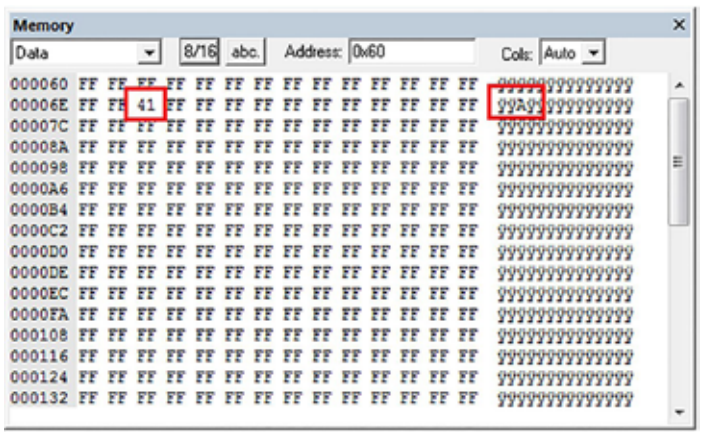


Figura 3.12 Éste es el dato en la dirección 0x70

Caso 2:

En este caso el origen de escritura para la memoria de datos se hará en la dirección `0x100` (la dirección inicial de fábrica es `0x60`), y se usarán las etiquetas `VALORES1`, `VALORES2`, y `VALORES3` que reservarán `1 Byte` respectivamente en la memoria SRAM (otro caso es cuando se desea reservar espacio de memoria usando etiquetas en lugar de constantes):

```
.INCLUDE "M8515DEF.INC"
.DSEG

.ORG 0X100

RESERVA1: .BYTE 1 ;reserva 1 BYTE en la SRAM
RESERVA2: .BYTE 1 ;reserva 1 BYTE en la SRAM
RESERVA3: .BYTE 1 ;reserva 1 BYTE en la SRAM


.DB 1,2,3,4,5,6,7,8
.CSEG

LDI R20,'A'
STS RESERVA1,R20

LDI R20,'B'
STS RESERVA2,R20

LDI R20,'C'
STS RESERVA3,R20

LDI R16,4
.
.
.
LO_QUE_SIGUE:
```

Nótese que la directiva `.DSEG` es usada después del `.INCLUDE`, y una vez que se han reservado los bytes se usa la directiva `.CSEG` para la escritura en el segmento de código

Es muy importante reservar con directivas `.BYTE`

```
.INCLUDE "M8515DEF.INC"

.DSEG                   ;DATA SEGMENT
.ORG 0x59               ;dirección del SRAM con warning de "UNDER"

variable: .BYTE 1       ;se reserva un BYTE en la SRAM

.CSEG
.ORG 0

LDI R16,LOW(RAMEND)
OUT SPL,R16
LDI R16,HIGH(RAMEND)
OUT SPH,R16

CICLO:
RJMP CICLO
```

La subrutina debiera quedar de esta forma:

```
.INCLUDE "M8515DEF.INC"

.DSEG                ;DATA SEGMENT
.ORG 0x60            ;dirección del SRAM sin warning de "UNDER"

RESERVA: .BYTE 1     ;se reserva 1 BYTE en la SRAM

.CSEG
.ORG 0
.
.
.
CICLO:
RJMP CICLO
```

Es posible usar más de una vez la directiva `.CSEG`, lo importante es regresar al uso de la directiva `.CSEG` para inicializar la escritura en la memoria Flash. En el siguiente ejemplo se muestra la configuración del SP seguido de la reserva de `1 Byte` en la SRAM usando `.DSEG,` y posteriormente volver a usar `.CSEG` para el inicio del código de programa:

```
.INCLUDE "M8515DEF.INC"

.CSEG
.ORG 0

LDI R16,LOW(RAMEND)
```

```
OUT SPL,R16
LDI R16,HIGH(RAMEND)
OUT SPH,R16

.DSEG
.ORG 0x60
variable: .BYTE 1

.CSEG
LDI R17,$20
.
.
.
LO_QUE_SIGA:
```

Recuerde

Las directivas del ensamblador pueden cambiar la forma en que el ensamblador trabaja con el código, se puede cambiar la localidad del código en la memoria del programa asignando *etiquetas* a las direcciones de la SRAM o definiendo *constantes*. Se puede intercalar el uso de las directivas `.CSEG`, `.DSEG` y `.ESEG`, usando siempre al final de las directivas la `.CSEG` para regresar al código del programa.