```
.INCLUDE "M8515DEF.INC"

#define      MI_PIN             PINA2
#define      CONTACT            R17
#define      REED_CONTACT       DDRA
#define      REED_CONTACT_OUT   PORTA
```

```
.def        MI_REGISTRO_1      =R18
.def        MI_REGISTRO_2      =R19
```

Estas dos líneas se explicarán en **.DEF**

```
.CSEG
.ORG 0

;AQUÍ VA EL STACK POINTER

LDI CONTACT,50
LDI R16,0b0000_0001
OUT REED_CONTACT,R16
OUT REED_CONTACT_OUT,R16

LDI MI_REGISTRO_1,20
LDI MI_REGISTRO_2,100

CICLO:
IN CONTACT,MI_PIN
RJMP CICLO
```

```
.def  XL    =R26
.def  XH    =R27
```

Es posible monitorear el estado de un registro personalizado con .DEF en la ventana de WATCH del AVR Studio

**Sintaxis:**

```
.DEF Simbolo=Registro
```

**Ejemplo:**

```
.INCLUDE "M8515DEF.INC"
.DEF   SALIDA_1=R0

;.def MI_PIN =PORTA

.CSEG

.def   SALIDA_2=R20
.def   SALIDA_3=R18

.ORG 0
LDI R16,LOW(RAMEND)
OUT SPL,R16
LDI R16,HIGH(RAMEND)
OUT SPH,R16

.def   SALIDA_4=R19

CICLO:RJMP CICLO
```

Está comentado ya que marcaría error

```
.INCLUDE "M8515DEF.INC"
.INCLUDE "mis_macros.INC"

.CSEG
.ORG 0
```

```
STACK       ;MACRO

;MAPEO DE REGISTROS:

;R0  - para leer programa LPM
;R1  - para leer programa LPM
;R2  - para delay 1 segundo
;R3  - LIBRE
;R4  - LIBRE
;R5  - para delay 1 segundo
;R6  - LIBRE
;R7  - para copiar contenido de PINA
;R8  - LIBRE...ETC
;R10 - para almacenar dato de proceso 1
;R11 - para almacenar dato de proceso 1
;R16 - para almacenar dato de proceso 1
;R18 - para almacenar dato de proceso 2
;R19 - para almacenar dato de proceso 2
;R20 - para almacenar dato de proceso 3
;R21 - para saber donde se quedó delay_media hora
;R22 - para saber donde se quedó delay_media hora
;R23 - para saber donde se quedó delay_media hora
;R24 - para memorizar contador 0
;R25 - para memorizar contador 0
;R29 - para memorizar contador 1
;R30 - para memorizar contador 1
;R31 - para reset de todas las funciones

.
.
.
;Y LO QUE SIGUE ES UN CÓDIGO DE MÁS LÍNEAS
```

Éste es un mapeo de registros

```
.INCLUDE "M8515DEF.INC"
.INCLUDE "mis_macros.INC"
```

```
.def   USER_LMP1    =R0    ;para leer programa LPM
.def   USER_LMP2    =R1    ;para leer programa LPM
.def   USER_LMP3    =R2    ;para leer programa LPM
.def   USER_DELAY1  =R5    ;para delay 1 segundo
.def   USER_PINA    =R7    ;para copiar contenido
                           ;de PINA


.                   .
.                   .
.                   .
.def   USER_CONT1   =R30   ;para memorizar contador 1
.def   USER_RST     =R31   ;para reset de todas las
                           ;funciones

.CSEG
.ORG 0

STACK

;Y LO QUE SIGUE ES UN CÓDIGO DE MÁS LÍNEAS
```

Ésta es la sección de definiciones **.DEF**

Aquí sólo se tendrían que modificar los registros y ya no en todo el programa

**Sintaxis:**

```
.UNDEF símbolo
```

**Ejemplo:**

```
.INCLUDE "M8515DEF.INC"

.DEF   SALIDA_1=R0

.CSEG
```

```
.def  SALIDA_2=R20
.def  SALIDA_3=R18

.ORG 0

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

.def  SALIDA_4 = R19

;se usan SALIDA_1, SALIDA_2, SALIDA_3, SALIDA_4 en el
;programa
.
.
.
;ahora nos interesa renombrar o quitarle la etiqueta a R20,
;entonces

.UNDEF SALIDA_2          ;DESNOMBRAMOS aquí

.def  SALIDA_20 = R20    ;y volvemos a NOMBRAR acá a R20 con
                         ;la etiqueta
                         ;"SALIDA"
CICLO: RJMP CICLO
```

Ejemplo:

```
.EQU TEST   =$40
.EQU NUMERO =23000            ;$59D8 EN HEXADECIMAL

LDI R16,LOW(NUMERO)           ;R16=$D8
LDI R17,HIGH(NUMERO)          ;R17=$59
```

```
.EQU NUM_40BITS = $7F_25_41_DE_22    ;=(546_085_920_290)dec

LDI R16,BYTE1(NUM_40BITS)      ;$22

LDI R17,BYTE2(NUM_40BITS)      ;$DE
LDI R18,BYTE3(NUM_40BITS)      ;$41
LDI R19,BYTE4(NUM_40BITS)      ;$25
LDI R20,BYTE5(NUM_40BITS)      ;$7F
```

Esta línea causará error en el uso de la función BYTE5.

```
   $7F        25         41         DE         22
0111_1111_0010_0101_0100_0001_1101_1110_0010_0010

      ←| 32 – 25 | | 24 – 17 | | 16 – 9 | | 8 – 1 | ← Posiciones
```

La sintaxis queda:

```
LDI R16, NUM_40BITS >> 32
```

Entonces la sintaxis final para cargar todo el número quedará:

```
#Define NUM_40BITS $7F_25_41_DE_22

LDI R16,BYTE1(NUM_40BITS)      ;$22
LDI R17,BYTE2(NUM_40BITS)      ;$DE
LDI R18,BYTE3(NUM_40BITS)      ;$41
LDI R19,BYTE4(NUM_40BITS)      ;$25
LDI R20, NUM_40BITS >> 32      ;$7F
```

Puede usar esta sintaxis con `#Define` para números más grandes.

Ejemplo:

```
.SET TEST = 40
LDI R16,TEST

.SET TEST = 50
LDI R17,TEST
```

Ejemplo:

```
.INCLUDE "TN2313DEF.INC"
.DEVICE ATTINY2313

.CSEG
.ORG 0

LDI R16,LOW(RAMEND)
```

```
OUT SPL,R16
.
.
.
;LO QUE SIGA
```

**Sintaxis:**

```
.CSEGSIZE = 10 | 12 | 14 | 16
```

**Por ejemplo:**

```
.CSEGSIZE = 12      ;Especifica el tamaño de la memoria como 12K x 16
```

Ejemplo:

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

```
.ORG 0

LDI R16,0
CPI R16,5

.IF 0                          ;SI ES DIFERENTE DE CERO FUNCIONA
LDI R16,10

.ELSE
LDI R16,100
.ENDIF

FIN: RJMP FIN
```

El cursor va a saltar hasta `.ELSE` sin pasar por el `.IF 0`

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

LDI R16,0
CPI R16,5

.EQU HOLA=0

.IF HOLA==0          ;SI LA EXPRESIÓN ES IGUAL A CERO FUNCIONA

LDI R16,10

.ELSE
LDI R16,100

.ENDIF
FIN: RJMP FIN
```

El cursor sí va a saltar al `.IF HOLA==0`

Si la expresión fuera diferente de cero, por ejemplo `.IF HOLA==2`, el cursor se iría a la condición `.ELSE`

```
.INCLUDE "TN2313DEF.INC"
.CSEG
.ORG 0

#define NUM_40BITS $7F_25_41_DC_22
.def NUM=R10

.IFDEF NUM
LDI R16,BYTE1(NUM_40BITS);$22
LDI R17,BYTE2(NUM_40BITS);$DC
LDI R18,BYTE3(NUM_40BITS);$41
LDI R19,BYTE4(NUM_40BITS);$25
LDI R20,NUM_40BITS >> 32 ;$7F
.ENDIF

.IFNDEF NUM
LDI R16,BYTE1(NUM_40BITS);$22
LDI R17,BYTE2(NUM_40BITS);$DC
LDI R18,BYTE3(NUM_40BITS);$41
LDI R19,NUM_40BITS >> 31
LDI R20,NUM_40BITS >> 32
.ENDIF

LO_QUE_SIGUE:
.
.
.
FIN: RJMP FIN
```

También funcionaría si escribimos:

```
.DEF NUM=R10
.EQU NUM=10
```

```
.MACRO TEST_IF_ELIF_ELSE
.IF (@0==10)
      LDI R16,20
.ELIF (@0>1)
      LDI R16,40
.ELSE
      LDI R16,'a'
.ENDIF
.ENDMACRO
```

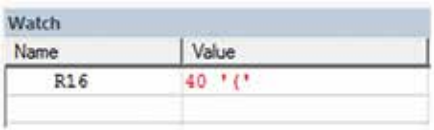
En el simulador escribiremos lo siguiente:

```
.INCLUDE "TN2313DEF.INC"
.INCLUDE "MIS_MACROS.INC"

.CSEG
.ORG 0

LDI R16,LOW(RAMEND)
OUT SPL,R16

TEST_IF_ELIF_ELSE 2

FIN: RJMP FIN
```

Se incluyó el archivo `MIS_MACROS.INC`

Se escribió el número 2 para comparar:
- No es igual a `10`
- Es mayor que `1 (2>1)` por lo que `R16=40`

En el simulador en VIEW>WATCH, R16=40:



Si cambiamos el valor de la macro a 10:

```
TEST_IF_ELIF_ELSE 10
```

En el simulador en VIEW>WATCH, R16=20

Si cambiamos el valor de la macro a 0:

```
TEST_IF_ELIF_ELSE 0      ;(un valor que no sea igual a 10 ni
                         ;mayor a 1)
```

En el simulador en VIEW>WATCH, R16='a'

```
MACRO USANDO_DIRECTIVA_ERROR
.IF (@0>1)
.ERROR "ES MAYOR QUE 1"
.ENDIF
.ENDMACRO
```

```
.INCLUDE "TN2313DEF.INC"
.INCLUDE "MIS_MACROS.INC"
.CSEG
.ORG 0

USANDO_DIRECTIVA_ERROR 1
FIN: RJMP FIN
```

Pero si ahora le asignamos el número 2 a la macro (que sí es mayor que el número 1):

```
.
.
.
USANDO_DIRECTIVA_ERROR 2
FIN: RJMP FIN
```

```
.MACRO USANDO_DIRECTIVA_WARNING
.IF (@0==1)

.WARNING "TEN CUIDADO!!!"
.ENDIF
.ENDMACRO
```

```
.INCLUDE "TN2313DEF.INC"
.INCLUDE "MIS_MACROS.INC"
.CSEG
.ORG 0
```

```
USANDO_DIRECTIVA_WARNING 1
FIN: RJMP FIN
```

```
.MACRO USANDO_DIRECTIVA_MESSAGE
.MESSAGE "**ESTE ES UN MENSAJE**"
.ENDMACRO
```

En la ventana de edición escribiremos lo siguiente:

```
USANDO_DIRECTIVA_MESSAGE
```

```
.INCLUDE "TN2313DEF.INC"
.CSEG
.ORG 0

LDI R16,LOW(RAMEND)
OUT SPL,R16

LDI R16,2
LDI R17,20
LDI R18,20

.EXIT

LDI R19,1
LDI R20,10
LDI R21,100
LDI R22,30

FIN: RJMP FIN
```

```
.INCLUDE "TN2313DEF.INC"
.CSEG
.ORG 0

LDI R16,LOW(RAMEND)
OUT SPL,R16
.INCLUDE "EXIT_TEST.TXT"

;continúa aquí después del .EXIT
;para abajo o lo que tenga que hacer
LDI R19,1
LDI R20,10
LDI R21,100
LDI R22,30

FIN: RJMP FIN
```

Éste es el archivo incrustado "EXIT_TEST.TXT":

```
EXIT:
LDI R16,2
LDI R17,20
LDI R18,20
.EXIT
```

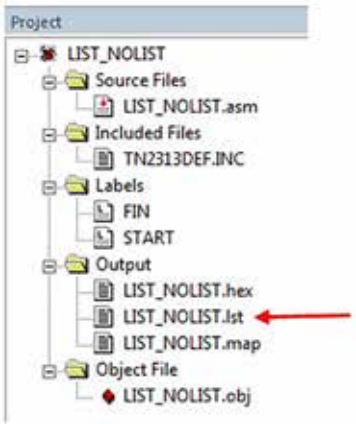
```
.INCLUDE "TN2313DEF.INC"
.CSEG
.ORG 0

INICIO:
LDI R16,LOW(RAMEND)
OUT SPL,R16
```

```
.NOLIST
LDI R16,'N'
LDI R16,'O'
LDI R16,'L'
LDI R16,'I'
LDI R16,'S'
LDI R16,'T'
```

Esta sección corresponde a las direcciones 0x02 al 0x07 que no aparecerán en .lst

```
.LIST
LDI R16,'L'
LDI R16,'I'
LDI R16,'S'
LDI R16,'T'
FIN: RJMP FIN
```

Esta sección sí aparecerá en la lista .lst




```
.
.
.
;*****END OF FILE **********************
```

```
                    .CSEG
                    .ORG 0

                    INICIO:
000000 ed0f         LDI R16,LOW(RAMEND)
000001 bf0d         OUT SPL,R1
```

Observe que las direcciones 0x02 al 0x07 no apareceren en .lst

```
                   .LIST
000008 e40c         LDI R16,'L'
000009 e409         LDI R16,'I'
00000a e503         LDI R16,'S'
00000b e504         LDI R16,'T'

00000c cfff         FIN: RJMP FIN
```

Observe que las direcciones 0x08 a la 0x0b sí apareceren en .lst

```
RESOURCE USE INFORMATION
------------------------
Notice:
The register and instruction counts are symbol table hit
counts,
and hence implicitly used resources are not counted, eg, the
'lpm' instruction without operands implicitly uses r0 and z,
none of which are counted.

x,y,z are separate entities in the symbol table and are
counted separately from r26..r31 here.

.dseg memory usage only counts static data declared with
.byte

ATtiny2313 register use summary:
r0 :  0 r1 :  0 r2 :  0 r3 :  0 r4 :  0 r5 :  0 r6 :  0 r7 :  0
r8 :  0 r9 :  0 r10:  0 r11:  0 r12:  0 r13:  0 r14:  0 r15:  0
r16: 12 r17:  0 r18:  0 r19:  0 r20:  0 r21:  0 r22:  0 r23:  0
r24:  0 r25:  0 r26:  0 r27:  0 r28:  0 r29:  0 r30:  0 r31:  0
x  :  0 y  :  0 z  :  0
Registers used: 1 out of 35 (2.9%)

ATtiny2313 instruction use summary:
.lds  :  0 .sts  :  0 adc   :  0 add   :  0 adiw  :  0 and   :  0
andi  :  0 asr   :  0 bclr  :  0 bld   :  0 brbc  :  0 brbs  :  0
brcc  :  0 brcs  :  0 break :  0 breq  :  0 brge  :  0 brhc  :  0
brhs  :  0 brid  :  0 brie  :  0 brlo  :  0 brlt  :  0 brmi  :  0
brne  :  0 brpl  :  0 brsh  :  0 brtc  :  0 brts  :  0 brvc  :  0
brvs  :  0 bset  :  0 bst   :  0 cbi   :  0 cbr   :  0 clc   :  0
clh   :  0 cli   :  0 cln   :  0 clr   :  0 cls   :  0 clt   :  0
clv   :  0 clz   :  0 com   :  0 cp    :  0 cpc   :  0 cpi   :  0
cpse  :  0 dec   :  0 eor   :  0 icall :  0 ijmp  :  0 in    :  0
```

```
inc   :  0 ld    :  0 ldd  :  0 ldi   : 11 lds   :  0 lpm   : 0
lsl   :  0 lsr   :  0 mov  :  0 movw  :  0 neg   :  0 nop   : 0
or    :  0 ori   :  0 out  :  1 pop   :  0 push  :  0 rcall : 0
ret   :  0 reti  :  0 rjmp :  1 rol   :  0 ror   :  0 sbc   : 0
sbci  :  0 sbi   :  0 sbic :  0 sbis  :  0 sbiw  :  0 sbr   : 0
sbrc  :  0 sbrs  :  0 sec  :  0 seh   :  0 sei   :  0 sen   : 0
ser   :  0 ses   :  0 set  :  0 sev   :  0 sez   :  0 sleep : 0
spm   :  0 st    :  0 std  :  0 sts   :  0 sub   :  0 subi  : 0
swap  :  0 tst   :  0 wdr  :  0
Instructions used: 3 out of 105 (2.9%)

ATtiny2313 memory use summary [bytes]:
Segment   Begin    End      Code   Data   Used    Size   Use%
----------------------------------------------------------------
--
[.cseg] 0x000000 0x00001a     26      0     26    2048   1.3%
[.dseg] 0x000060 0x000060      0      0      0     128   0.0%
[.eseg] 0x000000 0x000000      0      0      0     128   0.0%

Assembly complete, 0 errors, 0 warnings
```

Ejemplo:

Vamos a generar una macro llamada `LISTMAC_TEST` con el siguiente contenido, y salvemos esta macro en un archivo para ser incluido en el programa principal:

```
.MACRO LISTMAC_TEST
       LDI R16,5
       ADD R16,@0
.ENDMACRO
```

```
.INCLUDE "TN2313DEF.INC"
.INCLUDE "TEST.INC"
.CSEG
.ORG 0

LDI R18,10
LISTMAC_TEST R18

.LISTMAC
LISTMAC_TEST R0

FIN: RJMP FIN
```

```
.
.
.
                ;*****END OF FILE*************************
                .INCLUDE "TEST.INC"

                LDI R16,5
                ADD  R16,@0
                .ENDMACRO
                .CSEG
                .ORG 0

000000 e02a     LDI R18,10
000001 e005
000002 0f02     LISTMAC_TEST R18

                .LISTMAC
               +
000003 e005    +LDI R16 , 5
000004 0d00    +add R16 , R0
                LISTMAC_TEST R0

000005 cfff     FIN: RJMP FIN


RESOURCE USE INFORMATION
------------------------
.
.
.
```

Con `.LISTMAC` se expandió el contenido de la `MACRO` en el archivo `.lst`