十速科技股份有限公司
tenx technology inc.

Advance Information

TM89 series

cross assembler

使用手冊

Tenx reserves the right to change or discontinue this product without notice.

tenx technology inc.

INDEX

一、特性：

1. 運算元 (OPRAND) 可以定義成常數，且此常數可做四則運算。
2. 具有 Macro (巨集) 的功能，常用的程式可以寫成 Macro，供其它程式呼叫。
3. 具有compiler和link多個原始碼檔案之功能。(請參考介面使用手冊)
4. 具有每個專案都可以設定自己的library path。(請參考介面使用手冊)

二、限制：

1. **命名：**
   變數，常數及 label 的字串中只可以使用以下之文字符號：
   0 ~ 9, a ~ z, A ~ Z, "_"，但是不能以 0 ~ 9 的數字作爲開頭。名稱長度沒有限制。

2. **大小寫區分：**
   組譯label的名稱跟巨集的名稱時，可以設定是否要區分英文大小寫。(預設值是區分英文大小寫，請參考 .CASE)

3. **符號 (symbols) 跟標籤 (labels)：**
   (1). 數值常數 (Numeric constants)：
   數值常數是用等於符號 ('=') 來定義。如下：

   two = 2

   可以把常數 "two" 用在程式的任何地方，而且它的內容就是 2。
   <範例>：four = two * two

   (2). 一般標籤 (Standard labels)：
   標籤的使用是在每一列最前頭的位置定義一個標籤名稱，並且後面緊接著 一個冒號。

   (3). 區域標籤跟符號 (Local labels and symbols)：
   使用 .PROC指令，它能建立一段程式節區，在節區內宣告的labels跟symbols是區域性的。這些labels跟 symbols它們在此節區以外是不爲人知的且不能被存取的。

   (4). 使用巨集定義 labels 和常數(Using macros to define labels and constants)：

使用這方式也有它不便利的時候，但它可能在一些情形中是便利的。如　使用 .DEFINE指令，可以定義symbols或者其它地方想使用的常數。巨集的使用基本上是沒有設限的，在很低階的操作或運算上也是可以的。在其它的地方，你也可以用這樣的方式來定義字串常數。(其它型態的符號是不能這樣做的)

<範例>：

```
.DEFINE two     2
.DEFINE version "SOS V2.3"

 four = two * two                ; Ok
.byte   version                  ;  Ok

.PROC                            ;  開始區域範疇 (scope)
 two = 3                         ;  "two = 3",two 是區域常數
.ENDPROC                         ;  結束區域範疇 (scope)
```

三、數值系統：

1. 表示法：

(1). 二進位：以 “B” 做爲識別，不區分大小寫，或者以 “%” 作爲開頭。

<範例 1>：1000B, 10000100B, 1011b

<範例 2>：%1000, %10000100

(2). 十進位：不需加上任何識別字。

<範例>：20

(3). 十六進位：

(3-1). 以 “H” 做爲識別，不區分大小寫。

<範例>：8H, 0FFH, 0fh

(3-2). 以 “$” 作爲開頭。

<範例>：$9A, $FD

(建議以 $ 作爲開頭，避免與 .EQU, .DN 定義名稱混淆)

(3-3). 以 “0x” 作爲開頭，不區分大小寫。

<範例>：0x8, 0X0FF

(4). 常數或位址：以“.EQU”做爲識別，不用區分大小寫。或是在名稱後面接著“=”的符號。

<範例 1>：addr .EQU 4

<範例 2>：addr = 4

2. 四則運算：

在原始碼中可以針對常數做四則運算，運算時會遵循四則運算之先乘除後加減之特性。運算符號如下：

" + "：加法符號

" - "：減法符號

" * "：乘法符號

" / "：除法符號

" % ", ".MOD "：modular 符號

```
<範例>
addr .equ 4
lda addr%3                  ; lda 1
lda addr .mod 4             ; lda 0
```

"<<"：左移符號

```
<範例>
addr .equ 2
lda addr<<2                 ; lda 8
```

">>"：右移符號

```
<範例>
addr. equ     8
lda     addr>>2               ; lda 2
```

" | ", ".BITOR"：二進位的 OR 符號

```
<範例>
addr .equ 2
lda addr | 4                  ; lda 6
```

" & ", ".BITAND"：二進位的 AND 符號

<範例 >

```
addr. equ    6
lda addr & 4                    ; lda 4
```

" ~ ", ".BITNOT"：二進位的 NOT 符號

<範例>

```
addr. equ 11110000B
lda ~addr                       ; lda 00001111B
```

" ^ ", ".BITXOR"：二進位的 XOR 符號

<範例>

```
addr. equ $F0
lda addr ^ $FF                  ; lda $0F
```

" (" , ") "：小括號

<範例> 兩層的括弧計算

```
HIGHT   .EQU   ((10+5)*2)/3
```

" = "：比較運算 (等於)

" <> "：比較運算 (不等於)

" < "：比較運算 (小於)

" > "：比較運算 (大於)

" <= "：比較運算 (小於或等於)

" >= "：比較運算 (大於或等於)

" && ", ".AND"：布林 AND 符號

<範例>

```
addr. equ 6
lda addr && 4               ; lda 1
```

" || ", ".OR"：布林 OR 符號

<範例>

```
addr. equ 1
```

```
.define zpaddr 1
lda (!addr) && zpaddr          ; lda 0
```

".XOR"：布林 XOR 符號

<範例>

```
addr. equ 6
lda addr .XOR 4                ; lda 0
```

" ! ", ".NOT"：布林 NOT 符號

<範例>

```
addr. equ 1
lda (!addr) && 1               ; lda 0
```

四、假指令種類：

1. 區域指定指令：

下列節區 (segment) 並無先後次序之分別，但是每個節區都必須以其 KEY WORD 做爲完整之區隔。沒有使用的節區可以不必定義。

(1). ".CODE"：表切換到程式節區，大小寫均可。程式的原始碼在此節區定義。是「.segment "CODE"」的縮寫。

(2). ".RODATA"：表示切換到資料表節區，大小寫均可。用來定義 Table ROM 的內容同時可以使用 labels。

(3). ".RAM" , ".ENDRAM"：RAM 節區，大小寫均可，但須注意一定要"成對存在"。宣告 data RAM 位址的變數或常數之節區。data RAM 位址的變數(DN)只可在此一節區宣告。

(4). ".END"：大小寫均可，只能宣告一次。碰到這個指令組譯會被強迫終止。組譯終止在此，即使結束指令從引入檔案中被讀到。

2. 晶片種類指令：

(1). ".CPU"：大小寫均可，只能宣告一次，可以放在節區外的任意位置 (建議放在檔案開始位置)。宣告方式如下：

.CPU chip_species

chip_species: 晶片代號

<範例>

```
.CPU TM8959    <----  8959 chip
.CODE
……
……
ADD     10H
```

```
ADD     21H
.....
.....
.END
```

3. 其餘指令：

(1). ".ADDR"：

定義 2 個位元組大小的資料。這是 ".WORD"指令的一個別名，而且如果資料是位址值時用".ADDR"會更容易判讀。這個指令需要緊接著一串連續的表示式。(不一定非得是常數值，也可是識別字。)

<範例>

```
.addr   $0D00, $AF13, _Clear
```

(2). ".AUTOIMPORT"：

可跟隨一個加號 (+) 或減號 (-) 的字元。當開啓此功能(用 "+" 號)，不明確的 symbols 會自動地被標示以 import 替代錯誤。當關閉此功能(此爲預設值，組譯不做太多的判斷)，不會發生上述情形，但是會 顯示一個錯誤訊息。當自動引入 Symbols 的功能開啓時，組譯原始碼時對 未知的 Symbols 不會產生錯誤訊息，直到 Link 結束後才會產生 Symbols 沒有定義的錯誤訊息。

<範例>

```
.autoimport     +       ; 開啓自動引入符號的功能
Or
.autoimport     on      ; 開啓自動引入符號的功能
```

(3). ".BYT",".BYTE"：

定義 1 個位元組大小的資料。 這個指令需要緊接著一串連續的表示式或字串。

<範例>

```
.byte   'l', 'i', 'n', 'k'
.byt    'f', 'i', 'l', 'e', $0D, $00
```

(4). ".CASE"：

切換組譯時開啓或關閉識別字大小寫辨識功能，預設值是關閉(指識別字不區分大小寫)。這個命令後面必須跟著一個"+"或"-"號去選擇開啓或關閉。

<範例>

```
.case   -       ; 識別字 (保留字關鍵字或變數名稱) 不分大小寫
Or
.case   on      ; 識別字 (保留字關鍵字或變數名稱) 有分大小寫
```

(5). ".DB"：

定義位元組大小的資料。

<範例>

```
.RODATA
.db '2','3','4', '5'                ; 產生的位元組是 $32 $33 $34 $35
.org 10h
.byte $12, $34, $56, $78, $9a  ; 由 TABLE ROM 位址 10h 開始設定資料
.org 20h
.db $11, $22, $33, $44              ; 由 TABLE ROM 位址 20h 開始設定資料
.db 'A', 'B', 'C', 'D'
```

(6). ".DBYT"：

定義 2 個位元組且互換高低位元組的資料。這個指令需要緊接著一連串以字組的形式排列的資料 (word ranged)。

<範例>

```
.dbyt   $1234, $4512
```

產生的位元組是

$12 $34 $45 $12

將按照這個順序寫入目前的節區

(7). ".DEFINE"：

這個指令須跟隨一個識別字 (巨集名稱)，其後還可以跟隨一連串在括號內的參數。其後括號內還可以跟隨一連串的參數，請參考 .MACRO。

(8). ".DEF",".DEFINED"：

這個指令要有一個定義的參數在括號內。這個參數會被檢查，假設到目前程式位置之前，這個符號已經被定義在某處，函數就傳回真值；否則傳回假值。下面*<範例>*可以取代 .IFDEF 指令的用法：

<範例>

```
.if     .defined(a)
```

(9). ".DN"：

宣告一個變數來取代 data RAM 的位址，並定義此一變數所佔的 data RAM 的 nibble 數。在程式中，此變數可以直接取代 dataRAM 的位址。此一指令只可在 RAM 節區及常數節區宣告。宣告方式如下：

Variable .DN Nibble
Variable：變數名稱
Nibble ：宣告變數所佔的 data RAM 的 nibble 數，但是不可超過 data RAM 的最大位址。此一指令必須與 ORG 指令合併使用，以便定義 data RAM 的起始位址。

<範例>

```
.RAM
      .ORG      40H    ; 定義 data RAM 變數宣告的起始位址
      DISPLAY   .DN  1 ; 將 data RAM 的 40H 位址宣告成爲 DISPLAY
      SPEED     .DN  2 ; 將 data RAM 的 41H 及 42H 位址宣告成爲 SPEED
                       ; SPEED 取代 data RAM 的位址 41H
                       ; SPEED+1 取代 data RAM 的位址 42H，SPEED+1
      CHAR      .DN  1 ; 將 data RAM 的 43H 位址宣告成爲 CHAR
      ORG       70H
      LCD1      .DN  1 ; 將 data RAM 的 70H 位址宣告成爲 LCD1
.ENDRAM

.CODE
……
      ADC  SPEED       ; SPEED 取代 data RAM 的位址 41H
                       ; SPEED+1 取代 data RAM 的位址 42H
……
.END
```

(10). ".DWORD"：

定義 4 個位元組的資料型態，這個指令需要緊接著一串連續的表示式。

<範例>

```
.dword $12344512, $12FA489
```

(11). ".ELSE"：

條件式指令，用以逆轉條件表示式。(請參考 .ERROR)

(12). ".ELSEIF"：

條件式指令，用以逆轉目前的條件表示式並且檢查另一個條件表示式。(請參考 .ERROR)

(13). ".ENDIF"：

條件式指令，結束一個 .IF 或 .ELSE 的敘述。(請參考 .ERROR)

(14). ".ENDMAC",".ENDMACRO"：

巨集定義的結束。(請參考 .MACRO)

(15). ".ENDPROC"：

結束局部程式區塊。(請參考 .PROC)

(16). ".EQU"：

這個指令用來定義常數且區分大小寫。定義一個常數(constant)，此指令不能在 RAM 節區宣告。宣告方式如下：

Constant .EQU data
Constant：常數名稱
Data ：常數的內容值

<範例 1>

```
VALUE1  .EQU  10H
```

<範例 2>

```
.RODATA
    AH    .EQU   0H
    BH    .EQU   0H
.CODE
    ADD   AH  ; AH is equal to 0H.
    ADC   BH  ; BH is equal to 0H.
.END
```

(17). ".ERROR"：

警告組譯錯誤。組譯器會輸出一個使用者自訂的錯誤訊息， 因此不會產生 object 檔案。這個指令是用來檢查必須滿足的組譯條件。

<範例 1>

```
.if         foo = 1
……
.elseif     bar = 1
……
.else
.error      "Must define foo or bar!"
.endif
```

<*範例 2*>

```
.if DEBUG=1
   .error "No support in Debug mode"
.endif
```

(18). ".EXITMAC",".EXITMACRO"：

立即跳出巨集。這個指令在巨集的遞迴裡很常使用。(請參考 .MACRO)

(19). ".EXPORT"：

使其它程式碼檔案 (*.asm ,*.c) 也可以連結到目前宣告的 symbols。這個指令必須用逗號來區隔編列的symbols。 (請參考 .AUTOIMPORT)

<*範例*>

```
.export msg, start
.case -
.RODATA                          ; 定義 Table Rom 開始區間
.word $1234
.db '2','3','4'
.word 't','7'
.dword 12345678h,8765432h
msg:
.addr   $0D00, $AF13
.code
Start:
        szrx
        setdat $00
        sta 12
        sta 12 .mod 11
```

(20). ".GLOBAL"：

宣告全域的Symbols。這個指令宣告必須用逗號來區隔Symbols串列。串列裡的Symbols 是被定義在原始碼的某個地方，且被export出來，其它所有要使用到Symbols的地方都要import。另外相同的symbol 是允許同時使用 .IMPORT 或 .EXPORT 指令。

<範例>

```
.global foo, bar
```

(21). ".IF"：

評估一個表示式並且根據表示式的值來決定組譯程式是否進行輸出。這個表示式必須是一個常數運算式，也就是所有的操作元必須是　被定義的常數。表示式的值若爲 0，就爲假值；其它值都視爲真值。 (請參考 .ERROR)

(22). ".IFBLANK"：

條件式指令，測試巨集的某一個參數是否有傳入。如果　條件不成立，則要後面的程式是不會組譯直到碰到 .ELSE或 .ELSEIF或 .ENDIF的其中一個指令。這個指令通常是用來判斷巨集的參數是否有　傳入，若巨集參數沒有傳入表示爲真；反之爲否。

<範例>

```
.macro ADD2 v1,v2,sum
 lda v1
.ifblank   v2
 add sum              ; 如果 v2 沒有傳參數進來
.else
 add    v2            ; 如果 v2 有傳參數進來
.endif
 sta sum
.endmacro

.code
 lda 1
 ADD2 1, , sum        ; 如果沒有傳參數進來，必須以 "," 代替
 sta   2
.endcode
```

(23). ".IFDEF" :

條件式指令，測試symbol是否有被定義。這個指令後面必須 跟隨一個symbol的名稱。條件如果成立 (TRUE) 表示symbol已經被定義，否則就是FALSE。(請參考 .MACRO)

(24). ".IFNDEF" :

條件式指令，測試 symbol 是否有被定義。這個指令後面必須跟隨一個 symbol 的名稱。條件如果成立 (TRUE) 表示 symbol 沒有被定義， 否則就是 FALSE。

(25). ".IMPORT" :

從其它模組引入一個符號，這個指令的用法是在指令的後面緊接著一串以逗號分隔的引入符號。

<範例>

```
.import foo, bar
```

(26). ".INCLUDE" :

引入另一個檔案，巢狀引入檔案的深度最多 16 層。

<範例>

```
.include       "subs.inc"
```

(27). ".LOCAL" :

宣告區域性的 label 名稱，只能在巨集裡使用，巨集之外的 程式則不能使用。使用 local 宣告 label 的目的，是在避免巨集展開時，label 重覆使用的問題。若在巨集外使用區域性的 label，組譯時會產生錯誤訊息。

<範例>

```
.macro ADD1 v1,v2,sum
.local L1                    ; L1 定義成 ADD1 巨集裡的 label，
                             ; 巨集之外的程式則不能參照它
      lda       v1
      add       v2
      jmp       L1
      lda       v2
L1:   sta sum
.endmacro
```

(28). ".MAC",".MACRO"：

開始一個巨集的定義，這個指令必需跟隨一個識別字(巨集的名稱)並且可選用逗號來區隔巨集參數的識別字。

<範例>

```
.CODE
.macro  foo     arg1, arg2, arg3
   .if arg3 >0
   .define sum 123
   .endif

.if     .paramcount < 3       ; 判斷傳入巨集的參數是否小於 3
.error  "Too few parameters for macro foo" ;輸出使用者自訂的錯誤
.endif

.if     .paramcount > 3       ; 判斷傳入巨集的參數是否大於 3
.error  "Too many parameters for macro foo" ; 輸出使用者自訂錯誤
.exitmacro
.endif

lds 0x10,arg1
lda 0x10
lds 0x11,arg2
.ifdef sum                    ; 做加法
adc 0x11
.exitmacro
.endif
sbc 0x11                      ; 做減法
.endmacro

start :
   foo 5,9,
   foo 5,9,1
   jmp start
.END
```

(29). "_main" ：

定義 ASM 檔案的程式進入點，如同 C 程式的 void main()以利 Link 程式設定 PC 值 ,專案中有一個以上的*.ASM 或是混合專案中建議使用。

(30). ".ORG"：

定義一個起始位址，以便後續程式或是變數宣告之使用，在同一 節區內可以多次定義。若使用在資料表節區(.RODATA)時，C與ASM混和的專案不適合使用，建議使用者只在純ASM專案中使用，可搭配 .RELOC指令來避免 Overlap 而不自知的情形發生。這個指令不需區分大小寫，其宣告方式如下：

```
.ORG    Setting_addr
Setting_addr: 程式、RAM 或是 Table ROM 的位址
```

在程式節區中使用 ORG 指令時，可以直接定義下一個有效原始碼在程式中的位址。但是 label 名稱後面不能馬上跟著".ORG"。

<範例>

```
.code
……
        jb1   30H
        jb2   40H
        jb3   50H
        .org   30H     ; 程式 PC 位址爲 30H
        lda   1H
……
        .org   40H     ; 程式 PC 位址爲 40H
        lda   2H
……
        .org   50H     ; 程式 PC 位址爲 50H
        lda   3H
……
.end
```

在 RAM 節區中使用 ORG 指令時，可以直接定義下一個"DN"指令所定義的 data RAM 變數所代表的 data RAM 位址。

(31). ".PARAMCOUNT"：

用在 macro 裡，判斷傳入巨集的參數的個數。(請參考 .MACRO)

(32). ".PROC"：

使用.PROC指令會進入字彙宣告層的功能，所有在此以後新定義的符號都只存在這個局部的宣告區塊中，從外部是無法存取的。定義在宣告層外的符

號只要不被局部字彙重新定義，就可以存取得到。在別的宣告層字彙裡符號並不會引起命名衝突，因此，可以使用相同名稱來宣告變數。當碰到.ENDPROC指令時，字彙宣告層的功能便會告一段落。字彙宣告層最多可以有 16 層，指令也可以緊接著一個變數名稱，而這個變數名稱是一個定義在外層字彙的標籤，其值會是這個宣告層開始位址的程式計數器值。請注意，巨集名稱一定是在全域宣告中而且是存在另一個命名空間中。

<範例>

```
.proc Clear          ; 宣告 Clear 局部程式，開始一個新的宣告層。
lds HOUR1,5
Clear_all :          ; Clear_all 是區域性地如果在其它地方使用
                     ; 不會造成相同 symbol 的錯誤產生。
dec* HOUR1
jac 0
setdat Return
jmp Clear_all
Return: rts
.endproc             ; 結束這個宣告層
```

(33). ".RELOC" :

中斷".ORG"的指令,讓 LINK 重新分配 PC 位址。(搭配 .ORG 指令使用)

<範例>

```
.RODATA                  ; 宣告 TABLE ROM 區間。
    .org    00h
    .db  03fh,0f3h
    .db  00fh,0f0h
    .db  0cfh,0fch
    .org    20h
    .db  0f1h,00fh
    .db  0f0h,00fh
    .db  0f8h,08fh

.CODE
.RELOC       ;中斷.RODATA 裡所設定的.org,讓 link 程式自行分配 PC 位址
    LCD_clear:
    SHLX
    SETDAT $0080
    LDS8# @HL,$00
    LDS8# @HL,$00
                …
```

(34). ".SEGMENT"：

切換到另一個節區的指令。CODE 跟 RODATA 固定輸出到它們的節區，而所謂的節區是一種被命名的資料區塊，預設的節區是程式 節區。一個 object 檔案最大可以有 254 個不同的節區 (一個可執行檔最多則可以有 65534 個節區)。CODE 和 RODATA 是兩個最常使用來宣告節區的 指令。節區宣告的指令後緊接著自訂的節區名稱 (命名上有些限制，建議只使用符合變數命名規範的節區名稱)。

<範例>

```
.segment "RODATA"          ; 切換到資料表節區
INT_Counter:
.db   'I','N','T',' ','C'
.db   'o','u','n','t','e'
.db   'r',00

.segment "CODE"            ; 切換到程式節區
IntCounterMode:
    fast
    lds    Mode,DeadLoopMode
    call   ClearLcd
    lds    Row_Pixel,00
    lds    Column_Pixel+0,00
    lds    Column_Pixel+1,00
    lds    StringTableAddress+0,INT_Counter & 0fh
    lds    StringTableAddress+1,INT_Counter>>4&0fh
    lds    StringTableAddress+2,INT_Counter>>8
    lds    StringTableAddress+3,INT_Counter>>12
    call   DisplayString
                :
                :
```

(35). ".WORD"：

定義 2 個位元組的資料型態，這個指令需要緊接著一串連續的表示式 (word ranged，不一定非得是常數值)。

<範例>

```
.word   $0D00, $AF13
```

五、錯誤訊息：

1. "Command/operation not implemented"

2. "Cannot open include file"
   請確認檔案是否存在。

3. "Cannot read from include file"
   請確認檔案是否存在。

4. "Include nesting too deep"
   引入檔案的深度不可以超過 16 層。

5. "Invalid input character: "

6. "Hex digit expected"
   請參考 數值系統 (Value Systems) 的十六進制表示法。
   <範例>
   MRW %01, $%10　　正確是：MRW %01, $10

7. "Digit expected"
   請參考 數值系統 (Value Systems) 的十進制表示法。

8. "`0' or `1' expected"
   請參考 數值系統 (Value Systems) 的二進制表示法。

9. "Numerical overflow"
   整數太大，必須小於等於 $FFFFFFFF。

10. "Control statement expected"
    <範例>
    .INCLUDE a.asm　　正確是：.INCLUDE "a.asm"

11. "Too many characters"

12. "`:' expected"
    Label 定義缺少一個冒號。

13. "`(' expected"
算術運算缺少一個右括弧。

14. "`)' expected"
算術運算缺少一個左括弧。

15. ***保留***

16. "`,' expected"
<範例>
MRW div/value,0fh,12　　正確是：MRW div/value,0fh

17. "Boolean switch value expected (on/off/+/-)"

18. ***保留***

19. ***保留***

20. "Integer constant expected"

21. "String constant expected"
<範例>
.include a.asm　　正確是：.include "a.asm"

22. "Character constant expected"

23. "Constant expression expected"
常數名稱未定義。

24. "Identifier expected"
變數，常數及 label 的字串中只可以使用以下之文字符號：'0' ~ '9', 'a' ~ 'z', 'A' ~ 'Z','_'，但是不能以 0 ~ 9 的數字作爲開頭。

25. "`.ENDMACRO' expected"
'.ENDMACRO' 指令必須搭配 '.MACRO' 使用。

26. "Option key expected"

27. "`=' expected"

28. ***保留***

29. "User error:"
使用者自訂的錯誤訊息。

30. "String constant too long"
字串常數最大可以容許 255 個字元。

31. "Newline in string constant"
字串常數必須在同一行

32. "Illegal character constant"
<範例>
.BYTE 'c1','2'　　正確是：.BYTE 'c','2'

33. "Illegal addressing mode"

34. "Illegal character to start local symbols"

35. "Illegal use of local symbol"

36. "Illegal segment name"

37. "Illegal segment attribute"

38. "Illegal macro package name"

39. "Illegal emulation feature"

40. "Illegal scope specify"

41. "Syntax error"
<範例>
LDS value#-$1, $1 正確是：LDS value -$1, $1

42. "Symbol is already defined"
重複定義 Symbol。

43. "Undefined symbol"
如果符號的名稱是在另一個來源檔，請用 '.AUTOIMPORT ON'。

44. "Symbol is already marked as import"

45. "Symbol is already marked as export"

46. "Exported symbol is undefined"
Symbol 名稱沒有定義。

47. ***保留***

48. "Unexpected end of file"

49. "Unexpected end of line"
<範例>
.BYTE 'c','2', 正確是：.BYTE 'c','2'

50. ***保留***

51. "Division by zero"
<範例>
```
.RODATA
   div = 0
.CODE
   MRW value/div, 1
```
正確是：div = 10

52. "Modulo operation with zero"
<範例>
MRW 08%,$10 正確是：MRW 08%value,$10

53. "Range error"
整數常數的 Size 大於資料型態的 Size。
<範例>
.BYTE $1234 正確是：.BYTE $12

54. "Too many macro parameters"
巨集傳入太多參數。

55. "Macro parameter expected"

56. "Circular reference in symbol definition"

57. "Symbol redeclaration mismatch"

58. "Alignment value must be a power of 2"

59. "Duplicate `.ELSE'"
'.ELSE' 在 '.ENDIF' 之前被重複使用。

60. "Conditional assembly branch was never closed"
缺少 '.ENDIF' 來結束 '.IF' 或 '.ELSE' 的條件測試分支。

61. "Lexical level was not terminated correctly"

62. "No open lexical level"

63. "Segment attribute mismatch"

64. "Segment stack overflow"

65. "Segment stack is empty"

66. "Segment stack is not empty at end of assembly"

67. ***保留***

68. "Counter underflow"

69. ***保留***

70. ***保留***

71. "File name `%s' not found in file table"

72. "'.DN' must define in '.RAM' segment"
'.DN' 必須在 '.RAM' 區間使用。

73. "`.ENDRAM' expected"
'.RAM' 和 '.ENDRAM' 必須成對出現。

74. "`.DN' expected"
'.DN' 必須在 '.RAM' 區間使用。(請參考 假指令種類的其餘指令)

75. "Illegal data"

76. "Oprand error"
指令的運算元個數錯誤
<範例>
lda src<<1, 1 ; 正確是：lda src <<1

77. "Cannot open COE file"
請確認 COE 檔案是否存在。

78. ***保留***

79. "Program ROM (XXXXH) out of range (YYYYH)"
Program ROM 的最大位址是 YYYYH，XXXXH 已經超出範圍。

80. "Table ROM (XXXXH) out of range (YYYYH)"
Table ROM 的最大位址是 YYYYH，XXXXH 已經超出範圍。