九齊科技股份有限公司

Nyquest Technology Co., Ltd.

使用手冊

NYASM

Nyquest MCU Assembler

Version 5.1

Feb. 22, 2024

目　　錄

1 前言

第一章將介紹在使用*NYASM*所需的基本相關知識。

1.1 導覽

1.1.1 大綱

這份檔主要介紹如何使用*NYASM*來開發九齊科技的微控制器應用程式。

用戶指南大綱如下：

2 *NYASM*簡介：定義及說明*NYASM*軟體是如何與其他的開發工具聯結使用。.

3 *NYASM*安裝和開始：*NYASM*安裝說明及操作概述。

4 在視窗作業系統中使用*NYASM*：介紹*NYASM*的操作介面及參數。

5 虛擬指令：介紹*NYASM* 程式語言所包含陳述、運算元、變數和其他的元件。

6 巨集指令：介紹如何使用*NYASM*內建的巨集處理器。

7 表示式的語法與運算：為在*NYASM*源碼檔中使用複雜的運算式提供一個指導方針。

附錄 A - 快速索引：*NYASM*快速參考、MCU列表。

附錄 B - *NYASM*錯誤與警告訊息：包含*NYASM*所產生的錯誤訊息及警告訊息列表。

附錄 C - 詞彙表：專業術語詞彙表。

1.1.2 指南規範

使用手冊時請依照下列的規範：

表格 1-1

型態	意義	範例
Arial 字型	使用者程式或範例程式。	#define BITWIDTH
角括號: <>	使用者定義變數。	<label>, <exp>
大括號與直立線段: { | }	互斥的參數選項。	an OR selection error level { 0 | 1 }
方括號: []	方括號內的內容是可省略的。	[<label>] db <expr>[,<expr>,...,<expr>]
省略號: ...	省略範例中已不需要特別說明的部分。	List “list_option”, ...,“list_option”
0xnn	16進位數值, n 是一個16進位數字。	0xFF, 0x3B

1.1.3 更新

NYASM 和其他九齊科技的開發工具將不斷地依照客戶需求做更新，因此客戶可能會遇到在使用最新版軟體時可能會有一些實際的交談視窗或工具的描述與本份檔有些許的出入。最新版的文件請參照九齊科技的網站。

1.2 建議閱讀

本檔只是一份介紹如何使用*NYASM*的入門指南。使用者在做應用開發時需要再參考所使用微控制器的資料手冊。

- Interface
  粗體字指定一個按鍵 OK、Cancel。
  三角括號內大寫字元：< >定義為特殊鍵 <TAB>，<ESC>。
- Microsoft Windows Manuals
  本使用手冊是假定使用者熟悉微軟的視窗作業系統。

1.3 九齊科技網站

九齊科技透過全球資訊網際網路提供線上支援。在這個網站上客戶將可以很容易的獲得九齊科技所提供的最新檔案及訊息。客戶可以使用Microsoft® Internet Explorer或其他瀏覽器透過網際網路就可以連上九齊科技網站。

- 連線到九齊科技網際網路網站
  九齊科技網址可以使用你慣用的網際網路瀏覽器連接到：**http://www.nyquest.com.tw**
  九齊科技的網站提供多樣的服務，使用者可以下載最新的開發工具、資料手冊、應用手冊、使用者指南和檔案。

其他特別需要注意的訊息：

- 九齊科技最新的新聞稿。
- 產品訊息。

1.4 開發系統的客戶通知服務

九齊科技提供客戶通知服務來幫助我們的客戶花最少的精力去掌握目前九齊科技的產品。每當我們在產品系列或開發工具上有任何的改變、更新、修訂或勘誤，客戶都能收到我們的email通知。

1.5 客戶支援

九齊科技所有產品的使用者將可以透過下列幾種管道獲得所需要的幫助。

- 經銷商或代理商。
- 應用工程師（FAE）。
- 熱線。

客戶若需要支援服務時可以與經銷商、代理商或應用工程師聯絡。

2 *NYASM* 簡介

*NYASM*為一套在PC執行的視窗版應用程式，它是針對九齊科技的微控制器家族所提供的一個組合語言開發平臺。

內容：

2.1 *NYASM* 系統需求

- Pentium 1.3GHz或更高級處理器，Windows 7、8、10、11作業系統。
- 至少1G以上的動態存取記憶體（SDRAM）。
- 至少2G可用硬碟空間。
- 顯示器和顯示卡支援解析度1366x768或更高。
- 需安裝 .Net Framework 4.0。

2.2 *NYASM* 功能性

*NYASM*是九齊科技為8-bit和4-bit微控制器提供一個組合語言開發的通用型解法方案。

*NYASM*主要的特色包含如下：

- 支援所有MCU的指令集。
- 視窗化的操作化介面。
- 完整的虛擬指令。

2.3 相容性

*NYASM*相容於目前九齊科技所有的產品開發系統，它包括了*Q-Code*、*NYIDE*等。*NYASM* 支援一個明確且一致的方法（章節4）。*NYASM* 也支援一些較舊的語法，所以使用者可以安心的使用本檔所描述的方法去開發新的程式。

3 *NYASM* 安裝和開始

本章將介紹如何安裝*NYASM* 到你的作業系統中，且將對*NYASM* 這個組譯器做概略介紹。

內容：

3.1 安裝

目前*NYASM*為支援於視窗XP/WIN7/WIN8的視窗版，*NYASM*.EXE有一個視窗圖型化操作介面。*NYASM*.EXE可以安裝在視窗XP/WIN7/WIN8上且被正常的執行。使用者可以從九齊科技網站上下載*NYASM*。*NYASM*下載時的檔案格式為ZIP壓縮檔。

安裝方法：

- 在你所要存放檔案的地方建立一個目錄。
- 使用WinZip對*NYASM*.zip做解壓縮。

3.2 組譯器簡介

*NYASM*所產生的完整程式碼可以直接被微控制器所執行。*NYASM*預設的條件即可以產生完整的程式碼。如圖3.1所示的程式，當一個源碼檔在這種方式下被組譯時，程式中所使用的值都必須在源碼檔中被定義，或者是在被引入的含括檔中定義。假設整個組譯的程式都沒有出現任何的錯誤時，最後將會產生一個BIN檔案。BIN檔為源碼檔經過組譯後可被執行的機械碼，使用燒錄器將這BIN檔讀入即可對展示版本做燒錄的動作，並驗證功能。

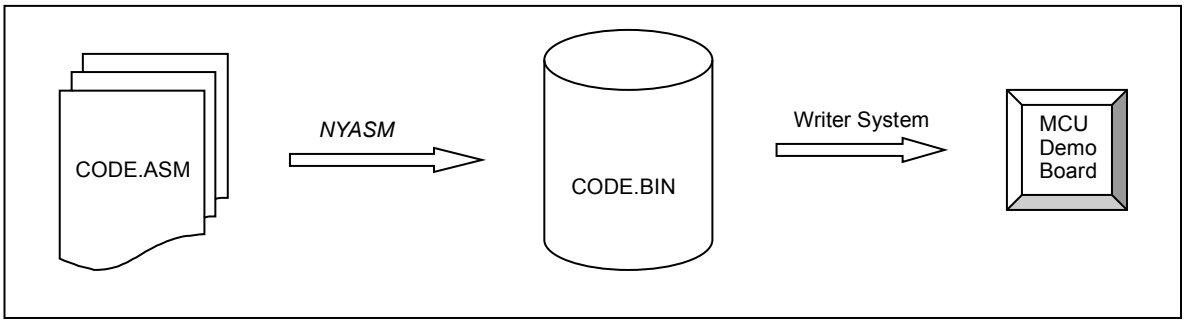

圖 3-1 產生一個完整的程式碼並驗證功能

3.3 組譯器輸入/輸出檔案

*NYASM*有些預設的副檔名格式，其各代表著相關不同的功能。

表格 3-1預設副檔名

副檔名	用途
.ASM	輸入到*NYASM* 的源碼檔。 <source_name>.ASM
.LST	組譯完成後所產生的列表檔。 <source_name>.LST
.ERR	*NYASM* 輸出檔案，內容為記錄組譯時所找到的警告與錯誤。 <source_name>.ERR
.BIN	組譯完成後所產生的二進制格式應用程式機械碼。 <source_name>.BIN
.HEX	組譯完成後所產生用來取代二進制格式的十六進制格式應用程式碼。 <source_name>.HEX
.DBG	*NYASM* 輸出的符號及除錯檔，這個檔案主要提供 *NYIDE* 除錯模式使用。 <source_name>.DBG

3.3.1 源碼格式 (.ASM)

源碼檔可由一般的ASCII文字編輯器所產生。其內容格式必須符合下列基本的方針。源碼檔的每一列可能包含了四種不同的資訊：

- 標記（Labels）
- 指令助憶碼（mnemonics）
- 運算元（operands）
- 註解（comments）

上述的順序及位置是非常重要的。標記必須放在每一列的第一個非空白位置。指令助憶碼若單獨佔用一列，必須放在第一個非空白位置。指令助憶欄若與在標記在同一列，必須在標記的後面並以冒號（：）隔開。運算元是跟隨在指令助憶碼的後面並以空白字元隔開。註解可以是放在標記、指令助憶碼、運算元後面，也可以是在任何列的任意位置。冒號或空格是用來分隔標記和指令助憶碼及指令助憶碼和運算元。複合的運算元必須使用逗號做分隔。範例如下：

範例：

NYASM **程式碼範例 (顯示多個運算元)**

```
; sample NYQUEST assembler source code
    ;
    list    p=ny5c640b, c=off, r=hex
    ORG_OFF         equ     0x30
    ORG_SUBOFF      equ     0x00
    SUBPPTRADDR     equ     ORG_SUBOFF+ORG_OFF
```

```
#include   "2102.h"
org        0x10
mvma       0x20
jmp        start
org        0x30
start:
mvma       0x30
mvat       0x12
end
```

3.3.1.1 標記 (Labels)

一個標記必須放在每一列的第一個非空白位置。一個標記後面必須加冒號（：）。標記的開頭必需為英文字母或是底線（_），標記內容則可由英文字母、底線（_）或是@符號所組成。標記欄的英文字母預設不區分大小寫，但是也可以從*NYASM* 的命令列中選擇開啟。

3.3.1.2 指令助憶碼 (Mnemonics)

組合語言指令助憶碼、組合語言虛擬指令和巨集呼叫可以放在每一列的任何行位置。如果一列同時存在有標記及指令助憶碼，則在兩者之間需用冒號隔開。

3.3.1.3 運算元 (Operands)

運算元必須使用一個以上的空格或tab來和指令助憶碼隔開。多個運算元必需使用逗號來區隔。

3.3.1.4 註解 (Comments)

NYASM 會將分號後面的任何文字、符號視為註解。從分號後面開始到一列的結束中間的所有字元都將被*NYASM* 所忽略。若在字串常數當中是允許包含了一個分號，並不會和註解搞混。

3.3.2 列表檔格式 (.LST)

範例：

```
NYASM列表檔（.LST）實例
Nyquest Technology Co., Ltd.
NYASM 1.00 Copyright(c) Nyquest Technology Co., Ltd. [Build:Dec 20 2007]

 File=E:\MyProjects\Build\asm\NYASM\Sample\NYASMSample.asm
Date=2007/12/20, 06:22:21 pm

ADDR      OPCODE/VALUE   LINE    TAG SOURCE STATEMENT                PAGE:1
                         0-0001        ; sample NYQUEST assembler source cod
```

```
                    0-0002    ;
                    0-0003    list   p=ny5c640b , c=off ,r=hex
        000000030   0-0004    ORG_OFF        equ    0x30
        000000000   0-0005    ORG_SUBOFF     equ    0x00
        000000030   0-0006    SUBPPTRADDR    equ    ORG_SUBOFF+ORG_OFF
                    0-0007    #include   "2102.h"
                    1-0001    ;
        000000010   0-0008    org      0x10
000010  D020        0-0009    mvma     0x20
000011  6030        0-0010    jmp      start
        000000030   0-0011    org      0x30
        000000030   0-0012    start:
000030  D030        0-0013    mvma     0x30
000031  0112        0-0014    mvat     0x12
                    0-0015    end

NYASM 1.00 Copyright(c) Nyquest Technology Co., Ltd. [Build:Dec 20 2007]

File=E:\MyProjects\Build\asm\NYASM\Sample\NYASMSample.asm
Date=2007/12/20, 06:22:21 pm

SYMBOL TABLE        TYPE      VALUE        PAGE:2
__NY5C640B          Constant  00000001
ORG_OFF             Constant  00000030
ORG_SUBOFF          Constant  00000000
Start               Label     00000030
SUBPPTRADDR         Constant  00000030
SOURCE FILE TABLE
000   E:\MyProjects\Build\asm\NYASM\Sample\NYASMSample.asm
001   E:\MyProjects\Build\asm\NYASM\Sample\2102.h

PROCESSOR       = NY5C640B (4 bits)
PROGRAM ROM     = 0x00000000 - 0x000FFFFF
DATA ROM        = 0x00000000 - 0x000FFFFF
SRAM / SFR      = 0x00000000 - 0x000000FF
```

列表檔的格式是由*NYASM*直接產生。產品名稱和版別，組譯的日期和時間，和頁碼將會顯示在每一頁的最上面。第一個欄位的數字代表著程式碼將被放在記憶體內部的位址。第二個欄位表示任何符號經由EQU、VARIABLE、CONSTANT或CBLOCK等虛指令所產生的32-bit值。第三個欄位元被保留給機械指令用，這些指令將被九齊科技的MCU所執行。第四個欄位列出程式源碼檔中對應的列數。剩下的欄位保

留給產生機械碼的原始程式碼。錯誤、警告和訊息將被嵌入於來源列之間且是位於相關聯的來源列的下面。符號表將列出所有被程式所定義的符號。

3.3.3 錯誤訊息的檔案格式 (.ERR)

*NYASM*預設值將會產生一個錯誤檔案，這個檔案在對程式碼做除錯時非常的實用。

錯誤檔案中的訊息格式如下：

[<type>] <file> (<line>) <number> <description>

範例：

[Error] C:\PROG.ASM 7 (133) w001: Symbol not previously defined (start)

附錄B為*NYASM*所產生的錯誤訊息說明。

3.3.4 十六進制檔格式 (.HEX)

*NYASM*能夠產生不同的hex檔案格式。

3.3.5 符號和除錯檔格式 (.DBG)

當*NYASM*被*NYIDE*所呼叫時，將會產生一個DBG檔案給ICE做程式除錯使用。

4 在視窗作業系統中使用 *NYASM*

這一章將用來介紹視窗版的*NYASM*。視窗版的*NYASM*可以被單獨執行的視窗程式或是被九齊科技其他的開發工具所連結。舉例像是*Q-Code*和*NYIDE*。

4.1 視窗介面

視窗版的*NYASM*對於在組譯時的所有選項提供了一個圖型化的介面給使用者去設定組譯時相關的選項。在視窗中點選*NYASM*.EXE檔案執行即可。

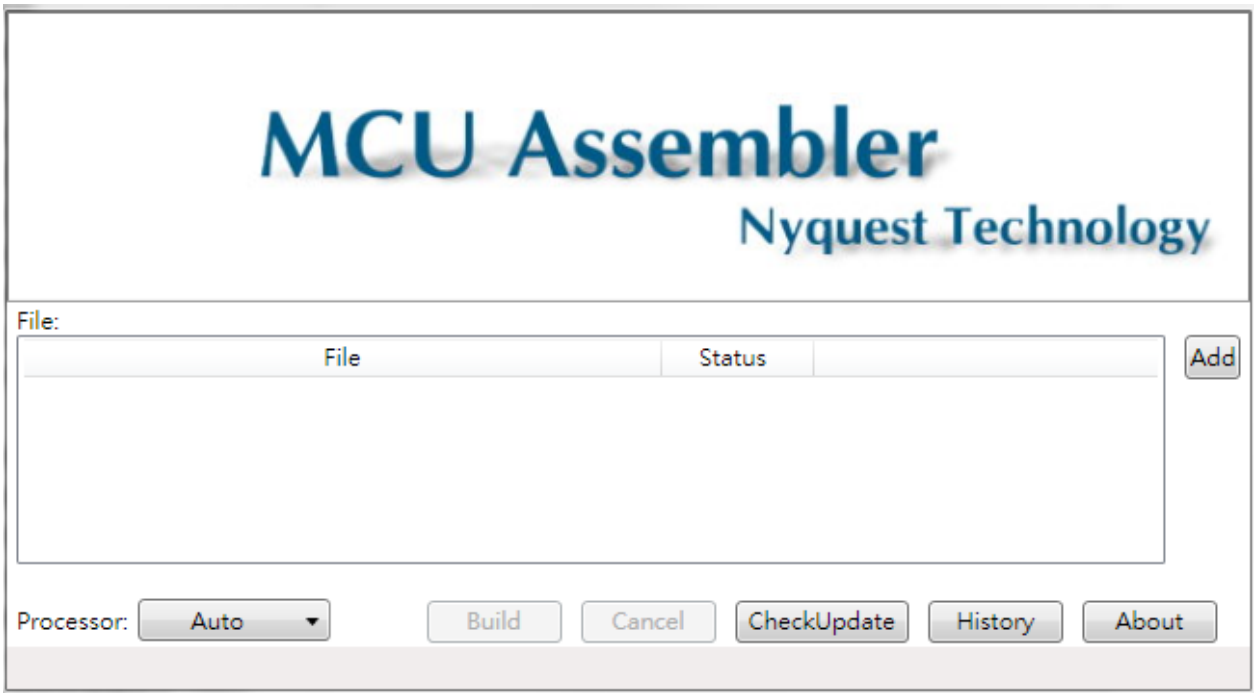

圖 4-1 視窗介面

選擇一個源碼檔的方法有二：將檔案拖拉至視窗內或者是按Add按鈕。設定變數的選項說明如下。最後按下Build對源碼檔做組譯。

注意：當視窗版的 NYASM 被九齊科技其他的開發工具所引用時，則選項畫面將不會出現。設定選項將會由其他的工具，以參數形式傳遞給 NYASM。

表格 4-1 組合語言設定選項

選項	用法
Processor	取代源碼檔的處理器設定。請參考A.2 MCU列表。

4.2 文字介面

*NYASM*除了圖形使用者介面之外，亦提供了文字模式介面。使用者可以在指令稿（script）使用文字介面呼叫*NYASM*，進而將工作自動化。文字介面的執行檔為安裝目錄下的*NYASM*.exe。可使用的參數如下表

表格 4-2 可用選項

選項	用法
/o=<file>	指定輸入的asm檔案
/p=<icbody>	取代源碼檔的處理器設定。請參考A.2 MCU列表。
/f=<file>	指定硬體組態設定檔。
/bypass	不顯示圖形介面，在組譯完成後結束程式。
/unlockrsvmem	允許編輯保留記憶體區域。
/nocfgblk	在組譯階段忽略已存在的硬體組態設定檔。

5 虛擬指令

這一章將介紹*NYASM* 的虛擬指令。虛擬指令是組譯器的命令，它是被加在源碼檔的程式當中，但並不會直接轉換成操作碼。他是被用來控制組譯器的輸入、輸出和資料分配。

5.1 基本型態

NYASM 提供了五種基本型態的虛擬指令：

- 控制型虛擬指令（Control Directives）：控制型虛擬指令允許有條件式的組譯程式區塊。
- 資料型虛擬指令（Data Directives）：資料型虛擬指令主要是控制記憶體配置且經由特殊的命名方式象徵性的提供一種參考資料的方法。
- 列印型虛擬指令（Listing Directives）：列印型虛擬指令是控制*NYASM* 產生清單檔格式的指令。他們允許的規格有標題、頁碼和其他列印的控制。
- 巨集型虛擬指令（Macro Directives）：這些虛擬指令是控制在巨集定義內部的執行與資料配置。

5.2 NY4、NY5、NY7、NY8A、NY9

5.2.1 虛擬指令摘要

表格5-1含括*NYASM*所提供的所有虛擬指令。章節後面將專門的詳細介紹*NYASM*所提供的虛擬指令。

表格 5-1虛擬指令集

虛擬指令	說明	語法
BREAK	從 **FOR**, **WHILE** 或 **REPEAT-UNTIL** 迴圈中跳離，或者從 **SWITCH** 區塊中跳到 **SWITCH** 的最末端。	break [<Boolean expression>]
CASE	屬於 **SWITCH** 區塊的一部分，**CASE** 必須在 **SWITCH** 區塊中使用。	switch <expression> case <expression 1>[,<expression 2>] <statements>
CBLOCK	定義一個常數區塊。	cblock [<expr>]
CONSTANT	宣告符號常數。	constant
CONTINUE	跳至內部含有**CONTINUE**虛擬指令的**FOR**、**WHILE**或**REPEAT-UNTIL**迴圈的起始，在迴圈內**CONTINUE**後面的表示式都將會被忽略。	continue [<Boolean expression>]
DEFAULT	**SWITCH**區塊的一部分，使用**SWITCH**區塊時必須要有 **DEFAULT** 的條件式。**DEFAULT**為表示**SWITCH**判斷式中的預設的組譯區塊。	default <statements>
#DEFINE	定義一個文字替代標記。	#define <name> [<value>] #define <name> [<arg>,...,<arg>]

虛擬指令	說明	語法
DW	宣告一個字元組(word)的資料。	[<label>] dw <expr>[,<expr>,...,<expr>]
ELSE	提供一個相對於**IF**的組譯區塊。*NYASM*在**IF**的組譯區塊與ELSE的組譯區塊二者之間只會選擇一個做組譯。	else <statements>
END	結束程式區段。	end
ENDC	結束一個常數區段。	endc
ENDFOR	結束一個**FOR**迴圈。	endfor
ENDIF	結束條件式組譯區塊。	endif
ENDM	結束一個巨集定義。	endm
ENDS	提供一個方便管理的結束指令，可以使用在**ENDFOR, ENDW, ENDSW, ENDIF**。	ends
ENDSW	結束條件式**SWITCH**組譯區塊。	endsw
ENDW	結束一個**WHILE**迴圈。	endw
EQU	定義一個常數。	<label> equ <expr>
ERROR	發佈一個錯誤訊息。	error "<text_string>"
EXITM	從一個巨集離開。	exitm
EXPAND	巨集列表展開。	expand
FOR	執行**FOR**的迴圈計數。	for <iterator> = <expr1> to <expr2> [step <expr3>]
IF	條件式組譯程式區塊的起始。	if <expr>
IFDEF	假如符號已經有被定義就執行。	ifdef <label>
IFNDEF	假如符號沒有被定義就執行。	ifndef <label>
#INCLUDATA	含括一個二進制資料檔案。	#includata "<data_file>" [,<address>]
#INCLUDE	含括一個附加源碼檔。	#include "<include_file>"
LINES	重新宣告每一頁的列數。	lines <value>
LIST	列表的選項。	list [<list_option>,...,<list_option>]
LOCAL	宣告局部巨集變數。	local <label>[,<label>]
MACRO	宣告巨集定義。	<label> macro [<arg>,...,<arg>]
MAXMACRODEPTH	設定巨集展開的最大層數。	Maxmacrodepth [=] <expr>
MESSG	產生使用者定義的訊息。	messg "<message_text>"
NEWPAGE	重新宣告每一頁的列數。	Newpage <value>
NOEXPAND	關閉巨集的展開。	noexpand
ORG	設定程式的起始點。	[<label>:] org <expr>

虛擬指令	說明	語法
ORGALIGN	對齊程式的起始點位址。	[<label>:] orgalign <expr>, <align>
RADIX	宣告設定數值格式。	radix <default_radix>
REPEAT	至少會執行一次迴圈。	Repeat <statements> until <Boolean expression>
SUBTITLE	指定程式副標題。	subtitle "<sub_text>"
SWITCH	條件式組譯區塊的起始。	switch <expr>
TITLE	指定程式標題。	title "<title_text>"
#UNDEFINE	刪除一個替代的標記。	#undefine <label>
UNTIL	假如條件式成立就結束至少會執行一次的迴圈。	Repeat <statements> until <Boolean expression>
VARIABLE	宣告符號變數。	variable <label>[=<expr>,...,<label>[=<expr>]]
WHILE	**WHILE**所帶的條件若是成立則執行迴圈。	while <expr>
.ALIGN2	對齊程式的起始點位址。	.align2 <expr>, <bit>

5.2.2 BREAK – 跳出目前所在的迴圈

◆ 語法

語法 1：

```
<for|while|repeat – loop begin>
    [<statements>]
    break [<Boolean expr>]
    [<statements>]
<for|while|repeat – loop end>
```

語法 2：

```
switch <expr>
    case <expr1>[,<expr2>]
    [<statements>]
    break [<Boolean expr>]
    [<statements>]
    [<case-statements>]
endsw
```

◆ 說明

從程式中WHILE、FOR或REPEAT-UNIT等迴圈中跳離目前正在執行的流程。Break也應用於switch

區塊當中，其作用是在條件式分支之間做切換。

◆ 範例

範例 1：

```
for i =0 to 4
nop
break i==2
halt
endfor
```

範例 2：

```
a=1
switch a
case 1, 2
nop
break
case 1
halt
endsw
```

◆ 請參閱

FOR, WHILE, REPEAT, SWITCH

5.2.3 CASE – 定義一個 SWITCH 選項

◆ 語法

```
switch <expr>
    case <expr1>[,<expr2>]
    [<statements>]
    :
    :
    default
    [<statements>]
endsw
```

◆ 說明

定義一個被選擇的陳述句。一旦<expr>等於某一個在 case 後面的<exprN>，則目前執行的流程將會接續至 case 後面的項目。Case 是屬於 switch 區塊的一部分，且必須跟隨著 switch 一起使用。

◆ 範例

```
a=1
```

```
switch a
    case 1, 2
    nop
    break
    case 1
    halt
endsw
```

◆ 請參閱

DEFAULT, SWITCH

5.2.4 CBLOCK – 定義一個常數區塊

◆ 語法

```
cblock [<expr>]
[<label>[=<increment>][,<label>[=<increment>]]]
endc
```

◆ 說明

定義一個已被命名的常數列表。每一個標記<label>所被分配到的新數值將會遞增於先前標記<label>的數值。這個虛擬指令的目的是在分配位址偏移量給很多個標記。在遇到 ENDC 這個虛擬指令時結束目前命名的列表。<expr>表示在這區塊中第一個名稱的起始值。假如沒有<expr>，第一個名稱的數值將會接續上一個 CBLOCK 最後一個名稱的值往上加。假如在源碼檔中的第一個 CBLOCK 沒有<expr>，則將從 0 開始分配數值。假如<increment>有被指定，則下一個標記<label>將會分配到比先前標記<label>更高的<increment>的數值。多個命名可以寫在同一列上，但需使用逗號作區隔。在程式及資料記憶體中利用 cblock 定義常數是非常的實用。

◆ 範例

```
cblock 0x20                 ; name_1 will be assigned 20
name_1, name_2              ; name_2 is 21
name_3 =0x30, name_4        ; name_4 is assigned 30,name_4 is assigned 31.
endc
```

◆ 請參閱

ENDC

5.2.5 CONSTANT – 宣告符號常數

◆ 語法

```
constant <label>=<expr> [...,<label>=<expr>]
```

◆ 說明

產生符號給 *NYASM* 表示使用。將符號用 CONSTANT 做宣告與使用。VARIABLE 做宣告的最主要

差異在於使用 Constants 宣告作第一次初始後就不能再被重置並且在表示式<expr>在指定的同時就需要全部被處理。除此之外 constant 和 variables 在表示式中是可以交替使用。

◆ 範例

```
variable RecLength=64                    ; Set Default RecLength
constant BufLength=512                    ; Init BufLength
    :                                    ; RecLength may
:                                        ; be reset later
:                                        ; in RecLength=128
:                                        ;
constant MaxMem=RecLength+BufLength      ;CalcMaxMem
```

◆ 請參閱

VARIABLE

5.2.6 CONTINUE – 忽略後面的表示式並且從下一個迴圈開始

◆ 語法

```
<for|while|repeat – loop begin>
    [<statements>]
    continue [<Boolean expr>]
    [<statements>]
<for|while|repeat – loop end>
```

◆ 說明

在 WHILE、FOR 或 REPEAT-UNITL 迴圈區塊當中使用 continue 設一個邏輯參考點，在這個 continue 邏輯參考點之後的述句將全部被忽略，然後跳至包含 continue 虛擬指令迴圈方塊的起始。

◆ 範例

```
for i =0 to 4
        nop
        continue i==2
        halt
endfor
```

◆ 請參閱

FOR, WHILE, REPEAT

5.2.7 DEFAULT – 定義 SWITCH 中無條件式項目

◆ 符號

```
switch <expr>
    case <expr1>[,<expr2>]
    [<statements>]
```

```
:
:
default
[<statements>]
endsw
```

◆ 說明

定義 SWITCH 中的一個無條件選項。一旦在 switch 後面的<expr>都沒有與 case 後面的項目符合時，執行流程將會進入到 default 這個項目。Default 是 switch 區塊的一部分，它必需與 switch 搭配使用。

◆ 範例

```
a=1
switch a
case 1, 2
        nop
        break
        case 1
        halt
        default
nop
endsw
```

◆ 請參閱

CASE, SWITCH

5.2.8 #DEFINE – 定義一個文字替代標記

◆ 語法

```
#define <name> [<string>]
```

◆ 說明

這個虛擬指令是定義一個文字替代字串。在組合語言程式中的所有<name>都將會被<string>所替代。使用#DEFINE 這個虛擬指令時若沒有填入<string>，則所要定義的<name>將可能會被視為內部的註解並且作為 IFDEF 虛擬指令測試用。

◆ 範例

```
#define length 20
#define control 0x19, 7
:
:
srbr control ; set bit 7 in 0x19
```

◆ 請參閱

#UNDEFINE, IFDEF, IFNDEF

5.2.9 DW – 宣告一個字元組的資料

◆ 語法

[<label>:] dw <expr>[,<expr>,...,<expr>]

◆ 說明

在程式記憶體中保留一段空間給字元組型態的資料使用。數值將被存進連續記憶體位置中且位置計數器是每次加一。表示式<expr>或許是文字字串且被存放方式的說明請參照 DATA 虛擬指令的說明。

◆ 範例

```
dw 39, (d_list*2+d_offset)
dw diagbase-1
```

5.2.10 ELSE – 相對於 IF 的組譯區塊

NYASM 會在 IF 的組譯區塊與 ELSE 的組譯區塊之間擇一。

◆ 語法

else

◆ 說明

ELSE 需與 IF 虛擬指令搭配使用。它是在 IF 的判斷式不成立時提供一個替代的程式區塊。ELSE 在程式區塊或巨集裡是經常被使用到。

◆ 範例

```
speed macro rate
If rate < 50
dw slow
else
dw fast
endif
endm
```

◆ 請參閱

ENDIF, IF

5.2.11 END – 結束程式區段

◆ 語法

end

◆ 說明

表示程式結束。

◆ 範例

```
list p= ny4b095a
:       ; executable code
:       ;
end ; end of instructions
```

5.2.12 ENDC – 結束一個常數區段

◆ 語法

endc

◆ 說明

將 ENDC 放在 CBLOCK 列表的末端做為結束。

◆ 請參閱

CBLOCK

5.2.13 ENDFOR – 結束一個 For 的迴圈

◆ 語法

endfor

◆ 說明

ENDFOR 是做為 FOR 迴圈的結束。只要 FOR 虛擬指令的迴圈計數器被指定，在 FOR 與 ENDFOR 兩個虛擬指令之間的程式碼將會不斷的被展開並且連續的插入到原始程式碼之間，它將會一直執行到條件式所設定的邊界時才會停止。ENDFOR 在程式區塊或巨集裡是經常被使用到。

◆ 請參閱

FOR

5.2.14 ENDIF – 結束條件式組譯區塊

◆ 語法

endif

◆ 說明

這個虛擬指令是做為條件式組譯區塊的結束。ENDIF 在程式區塊或巨集裡是經常被使用到。

◆ 請參閱

ELSE, IF

5.2.15 ENDM – 結束一個巨集定義

◆ 語法

endm

◆ 說明

結束一個由 MACRO 開始的巨集定義。

◆ 範例

```
make_table macro arg1, arg2
    dw arg1, 0     ; null terminate table name
    res arg2  ; reserve storage
endm
```

◆ 請參閱

MACRO, EXITM

5.2.16 ENDS – 通用結束指令

◆ 語法

ends

◆ 說明

使用在 ENDFOR, ENDW, ENDSW, ENDIF

◆ 請參閱

ENDFOR, ENDW, ENDSW, ENDIF

5.2.17 ENDSW – 結束一個 Switch 的區塊

◆ 語法

endsw

◆ 說明

結束一個由 SWITCH 開始的區塊定義。

◆ 範例

參考 SWITCH 的範例。

◆ 請參閱

SWITCH

5.2.18 ENDW – 結束一個 While 的迴圈

◆ 語法

endw

◆ 說明

ENDW 作為 WHILE 迴圈的結束。只要在 WHILE 虛擬指令中所指令的條件式保持成立的狀態，則在 WHILE 與 ENDW 兩個虛擬指令之間的程式碼將會不斷的被展開並且連續的插入到原始程式碼之間。ENDW 在程式區塊或巨集裡是經常被使用到。

◆ 範例

參考 WHILE 的範例。

◆ 請參閱

WHILE

5.2.19 EQU – 定義一個組譯器常數

◆ 語法

<label> equ <expr>

◆ 說明

標記<label>將以<expr>的值取代。

◆ 範例

```
four equ 4 ; assigned the numeric value of 4 to label four
```

5.2.20 ERROR –發佈一個錯誤訊息

◆ 語法

error "<text_string>"

◆ 說明

<text_string>是以相同的格式被列印到任何的 *NYASM* 錯誤訊息。<text_string>可以包含字元數是 1 到 80 個。

◆ 範例

```
error_checking macro arg1
    if arg1 >= 55  ; if arg is out of range
    error "error_checking-01 arg out of range"
    endif
endm
```

◆ 請參閱

MESSG

5.2.21 EXITM – 從一個巨集離開

◆ 語法

exitm

◆ 說明

在組譯的時候立即強制從巨集展開的函式中離開。這個虛擬指令與在組譯時遇到 ENDM 虛擬指令有相同的效果。這個指令只有在 NY5+、NY6 系列可以使用。

◆ 範例

```
test macro filereg
    if filereg == 1 ; check for valid file
    exitm
    else
    error "bad file assignment"
    endif
endm
```

◆ 請參閱

ENDM MACRO

5.2.22 EXPAND – 展開巨集列印

◆ 語法

expand

◆ 說明

在列表檔中展開所有的巨集。這個虛擬指令的功用概略相等在組譯器上”Macro Expansion”的選項設定。只是它能夠在後面遇到 NOEXPAND 時，將展開的功能關閉。

◆ 請參閱

MACRO, NOEXPAND

5.2.23 FOR – 當設定值符合條件式時就執行 For 迴圈

◆ 語法

```
for <iterator>=<expr1> to <expr2> [step <expr3>]
:
:
endfor
```

◆ 說明

只要<iterator>等於<expr1>到<expr2>所設定的範圍時，則在 FOR 與 ENDFOR 之間的所有程式將會被組譯。一個 FOR 迴圈最大的迴圈數是 256 次。

◆ 範例

```
for I=0 to 5
        nop
endfor
```

◆ 請參閱

ENDFOR

5.2.24 IF – 一個有條件式可被組譯的程式碼區塊

◆ 語法

if <expr>

◆ 說明

開始執行一個條件式的組譯區塊。假如 <expr>判定為正確時，跟隨在 IF 後面的程式碼將會立即被組譯。否則在 IF 後面的程式碼將會被組譯器所跳過，一直到碰見 ELSE 或 ENDIF 虛擬指令。若<expr>最後的結果是等於零時則將被視為邏輯性的"正確"，反之若這個<expr>最後的結果是等於其他任意的值時，將被視為邏輯性的"錯誤"。在 IF 及 WHILE 等虛擬指令是將其表示式的結果視為一個邏輯性運算，"正確"表示將會回傳一個非零的值，"錯誤"表示將會回傳一個零值。

◆ 範例

```
if version == 100; check current version
    :      ;executable cod
    :      ;executable cod
else
    :      ;executable cod
    :      ;executable cod
endif
```

◆ 請參閱

ELSE, ENDIF

5.2.25 IFDEF – 假如符號已經被定義就執行

◆ 語法

ifdef <label>

◆ 說明

假如<label>在先前已經被定義過了，則條件式路徑的內容就會被組譯直到組譯遇到 ELSE 或 ENDIF 時才會停止。先前的定義通常是用#DEFINE 虛擬指令來完成或者是在 *NYASM* 命令列上做設定。

◆ 範例

```
#define testing 1 ; set testing "on"
:
:
ifdef testing
<execute test code> ; this path would be executed.
Endif
```

◆ 請參閱

#DEFINE, ELSE, ENDIF, IFNDEF, #UNDEFINE

5.2.26 IFNDEF – 假如符號還沒被定義就執行

◆ 語法

ifndef <label>

◆ 說明

假如<label>先前尚未被定義或者是已經被用虛擬指令#UNDEFINE 改成未定義的型態，則虛擬指令 IFNDEF 後面的程式碼將被組譯。組譯將會被啟動或關閉直到組譯遇到 ELSE 或 ENDIF 時才會停止。

◆ 範例

```
#define testing1 ; set testing on
:
:
#undefine testing1 ; set testing off
ifndef testing ; if not in testing mode
:   ; execute this path
:
endif
end ; end of source
```

◆ 請參閱

#DEFINE, ELSE, ENDIF, IFDEF, #UNDEFINE

5.2.27 #INCLUDATA – 含括一個二進制檔案

◆ 語法

#includata "<binary_data_file>"[, address]

◆ 說明

將一個特殊檔案讀入當成二進制的資料。假如包含的資料檔需要插入到一個特殊的位置時，使用者可以在 address 特別指定所要位置。#includata 必須是在虛擬指令 end 前面的最後一個陳述句。<binary_data_file> 必須置於引號內。假如有特別指定路徑時，則組譯器只會到指定的路徑去搜尋，否則將以源碼檔所在的目錄做搜尋。<binary_data_flie>在組譯後將會變成一個標記。

◆ 範例

#includata "c:\music\s02.sog", 0x2000 ; insert data file at 0x2000

5.2.28 #INCLUDE – 含括額外的源碼檔

◆ 語法

#include "<include_file>"

◆ 說明

將一個特殊檔案讀入當成源碼檔的程式。在插入檔案之後，組譯器將會重新開始組譯原始源碼檔。<include_file> 必須置於引號內。假如有特別指定路徑時，則組譯器只會到指定的路徑去搜尋。否

則將以源碼檔所在的目錄做搜尋。

◆ 範例

#include "c:\sys\sysdefs.inc" ; system defs

#include “regs.h” ; register defs

5.2.29 LINES – 列表檔每一頁的行數

◆ 語法

lines <value>

◆ 說明

設定列表檔中一頁的最大行數。

◆ 請參閱

NEWPAGE

5.2.30 LIST – 列印選項

◆ 語法

list [<list_option>, ..., <list_option>]

◆ 說明

這個 LIST 虛擬指令和開啟清單輸出有相同的作用。此外，位於下表的 List 選項，都有個別控制組譯的程式或輸出列表檔的格式：

表格 5-2 List 虛擬指令選項

選項	預設狀態	說明
c	Off	英文字母大小寫區分開啟/關閉 c=on 開啟 c=off 關閉
p	None	設定處理器的類型： /p=<processor_type> 這裡的<processor_type>是指九齊科技的元件。例如：NY5A005A
unlockrsvmem	Locked	/unlockrsvmem 只適用於4-bit MCU。允許編輯保留記憶體區域。
nocfgblk	Configuration Block required	/nocfgblk 只適用於4-bit MCU。在組譯階段忽略已存在的硬體組態設定。

◆ 範例

list p=ny5c640b, c=off

5.2.31 LOCAL – 宣告區域性的巨集變數

◆ 語法

local <label>[,<label>...]

◆ 說明

在巨集裡宣告內部的標記，且僅在巨集中具效力。在宣告時<label>可能與巨集定義外的另一個標記是完全相同；但二者之間並不會衝突。即使在巢狀式的巨集呼叫中，每個引用將有自己局部的定義。

◆ 範例

```
<main code segment>
:
:
len equ 10 ; global version
size equ 20 ; note that a local variable may now be created and modified
test macro size
    local len, label ; local len and label
    len set size ; modify local len
    label res len ; reserve buffer
    len set len-20 ;
endm ; end macro
```

◆ 請參閱

ENDM, MACRO

5.2.32 MACRO – 宣告一個巨集定義

◆ 語法

<label> macro [<arg>, ..., <arg>]

◆ 說明

一個巨集表示可以用單一個巨集呼叫將一串序列的指令集插入到組合語言程式檔中。這個巨集必須先被定義，且可供後面的來源程式碼做索引。一個巨集可以呼叫另一個巨集或遞迴的呼叫自己本身。

◆ 範例

```
Read macro device, buffer, count
mvma device
mvma buffer
mvma count
endm
:
:
read 1,2,3
```

◆ 請參閱

ELSE, ENDIF, ENDM, EXITM, IF, LOCAL

5.2.33 MAXMACRODEPTH – 定義最大的巨集層數

◆ 語法

maxmacrodepth[=]<expr>

◆ 說明

MAXMACRODEPTH 將巨集的最大有效層數定義為<expr>。<expr>必須小於或等於最大的 256 層。MAXMACRODEPTH 在源碼檔中常被使用到。每次使用時就是重新定義巨集最大的有效層數。

◆ 範例

```
list p=ny5c640b
maxmacrodepth 0x10
:
:
```

5.2.34 MESSG – 產生使用者定義的訊息

◆ 語法

messg "<message_text>"

◆ 說明

依據一個訊息然後顯示在列表檔中。發佈一個 MESSG 虛擬指令不需要設定任何的錯誤回傳碼。

◆ 範例

```
mssg_macro macro
    messg "mssg_macro-001 invoked without argument"
endm
```

◆ 請參閱

ERROR

5.2.35 NEWPAGE – 在列表檔中換頁

◆ 語法

newpage <value>

◆ 說明

在列表檔中換至新的一頁繼續列印。

◆ 請參閱

LINE

5.2.36 NOEXPAND – 關閉巨集展開

◆ 語法

noexpand

◆ 說明

關閉在列表檔中巨集展開的功能

◆ 請參閱

EXPAND

5.2.37 ORG – 設定程式的起始點

◆ 語法

[<label>:] org <expr>

◆ 說明

設定程式的起始點位址在<expr>。假如<label>有被指定，則將會給定這個<label>一個在<expr>所設定的值。假如沒有使用 ORG，則程式碼將會在位址零的位置開始放。

◆ 範例

```
int_1: org 0x20
; Vector 20 code goes here
int_2: org int_1+0x10
; Vector 30 code goes here
```

5.2.38 ORGALIGN – 設定程式的起始點位址排列

◆ 語法

[<label>:] orgalign <expr>,<align>

◆ 說明

設定源碼的起始位置<expr>|<align>。假如<label>是被指定的，則它將被給定一個<expr>|<align>的值。假使沒有使用 ORGALIGN，源碼將在位址零開始被產生。

◆ 範例

```
int_1: orgalign 0x20,0x7
```

◆ 請參閱

.align2

5.2.39 RADIX – 數值格式

◆ 語法

radix <default_radix>

◆ 說明

設定在資料表示式中數值格式的預設值。格式的預設值是 dec(十進制)。格式的有效設定值有：hex(十六進制)、dec(十進制)、oct(八進制)或是 bin(二進制)。

◆ 範例

```
radix dec
```

◆ 請參閱

LIST

5.2.40 REPEAT – 定義一個從 Repeat 至 Until 的迴圈方塊

◆ 語法

```
repeat
:
:
until <expr>
```

◆ 說明

定義一個由 REPEAT 至 UNTIL 的迴圈區塊。

◆ 範例

```
test_mac macro count
   variable i
   i = 0
   repeat
      i += 1
   until i > count
endm
:
:
End
```

◆ 請參閱

WHILE, UNTIL

5.2.41 SUBTITLE – 程式的副標題

◆ 語法

```
subtitle "<sub_text>"
```

◆ 說明

<sub_text> 是一個在雙引號內部的 ASCII 字串。字串的最大長度是小於或等於 60 字。這個虛擬指令是建立第二個程式標題當作列印輸出時的副標題。

◆ 範例

```
subtitle "diagnostic section"
```

◆ 請參閱

TITLE

5.2.42 SWITCH – 條件式的交換組譯區塊

◆ 語法

```
switch <expr>
    case <expr1>[,<expr2>]
    [<statements>]
    case < exprM>[,<exprN>]
    :
    :
    default
    [<statements>]
endsw
```

◆ 說明

開始執行條件式的交換組譯區塊。假如對<expr>判定的結果是與任何在 case 後的數值相符合，則在 case 後的程式將會被組譯。否則隨後的程式碼將都會被組譯器省略跳過，直到碰到 default 或 ENDSW 虛擬指令為止。

◆ 範例

```
a=1
switch a
    case 1, 2
    nop
    break
    case 1
    halt
    default
endsw
```

◆ 請參閱

BREAK, DEFAULT

5.2.43 TITLE – 程式標題

◆ 語法

```
title "<title_text>"
```

◆ 說明

<title_text>是一個置於雙引號內可被列印的 ASCII 字串，且字串數必須小於或等於 60 個字元，這個虛擬指令是將雙引號內的文字置於清單檔中每一頁的最上層。

◆ 範例

```
title "operational code, rev 5.0"
```

◆ 請參閱

SUBTITLE

5.2.44 #UNDEFINE – 刪除一個替代的標記

◆ 語法

#undefine <label>

◆ 說明

<label>是一個先前使用#DEFINE 虛擬指令所定義的識別符號。它必須是一個對 *NYASM* 有效的標記。這個虛擬指令是將已經被命名的符號從符號表中被移除。

◆ 範例

```
#define length 20
:
:
#undefine length
```

◆ 請參閱

#DEFINE, IFDEF, #INCLUDE, IFNDEF

5.2.45 UNTIL – 執行迴圈直到條件式成立

◆ 語法

```
repeat
:
:
until <expr>
```

◆ 說明

只要<expr>的判定結果是”否”，則在 REPEAT 及 UNTIL 之間的程式碼將至少會被組譯過一次以上。一個 REPEAT 迴圈最大的重覆次數為 256 次。

◆ 範例

```
test_mac macro count
    variable i
    i = 0
    repeat
    i += 1
    until i < count
endm
:
:
end
```

◆ 請參閱

WHILE, REPEAT

5.2.46 VARIABLE – 宣告一個符號變數

◆ 語法

variable <label>[=<expr>][,<label>[=<expr>]...]

◆ 說明

產生一個可以在 *NYASM* 表示式使用的符號。變數和常數在表示式中通常是可以交替的使用。要特別注意的是變數的值在運算時不會被更新，使用者必須以等號(=)賦值，指定變數的改變。

◆ 範例

請參閱虛擬指令 CONSTANT 的範例說明。

◆ 請參閱

CONSTANT

5.2.47 WHILE – 當條件成立時就執行迴圈

◆ 語法

```
while <expr>
:
:
endw
```

◆ 說明

只要<expr>判定為真，在 WHILE 及 ENDW 之間的每一列將會被組譯。若<expr>最後的結果是零時，被視為邏輯性的”錯誤”，反之若<expr>最後的結果是不等於零的任意數時，將被視為邏輯性的”正確”。”正確”的代表將會回傳一個非零的值，”錯誤”表示將會回傳一個零值。一個 WHILE 迴圈內部最多可以包含 100 列指令，迴圈重覆次數最大到 256 次。

◆ 範例

```
test_mac macro count
variable i
i = 0
while i < count
movlw i
i += 1
endw
endm
start
test_mac 5
end
```

◆ 請參閱

ENDW IF

5.2.48 .ALIGN2 – 對齊程式位址

◆ 語法

.align2 <expr>, <bit>

◆ 說明

程式起始位址對齊<bit>個位元，並且位址的低位元為<expr>。

當我們需要位址為 0x41, 0x141, 0x241, 0x341, ……的時候，可以使用.align2 對齊 8 個位元，並設定低位元為 0x41。而 ORGALIGN 指令沒辦法指定位元數，所以無法避免 0xC1 產生。

這個指令只有在 NY5+, NY6 有支援。

◆ 範例

.align2 0x41, 8

◆ 請參閱

ORGALIGN

5.3 NY8L

NY8L所使用的虛擬指令和其餘系列皆不同，獨立於本章節介紹。

5.3.1 虛擬指令摘要

表格 5-3含括*NYASM*所提供的所有虛擬指令。章節後面將會詳細介紹*NYASM*所提供的虛擬指令。

表格 5-3虛擬指令集

虛擬指令	說明	語法
.and	布林 and 運算	<expr> .and <expr>
.bankbyte	取得 bank byte	.bankbyte(<expr>)
.bitand	位元 and 運算	<expr> .bitand <expr>
.bitnot	位元 not 運算	.bitnot <expr>
.bitor	位元 or 運算	<expr> .bitor <expr>
.bitxor	位元 xor 運算	<expr> .bitxor <expr>
.blank	參數是否為空	.blank(<symbol>)
.byte	低位元組	.byte(<expr>)
.ceil	無條件進位	.ceil(<expr>)

虛擬指令	說明	語法
.code	.segment “code”的簡寫	.code
.data	.segment “data”的簡寫	.data
.define	定義	.define <symbol> <expr>
.defined	查詢是否有被定義過	.defined(<symbol>)
.else	條件式組譯否則區塊	.else
.elseif	條件式組譯否則再判斷區塊	.elseif(<expr>)
.endif	結束條件式組譯區塊	.endif
.endmacro	結束巨集定義區塊	.endmacro
.endrepeat	結束重複區塊	.endrepeat
.endscope	結束變數範圍區塊	.endscope
.endstruct	結束結構區塊	.endstruct
.equ	定義常數	<symbol> .equ <expr>
.error	產生一個組譯錯誤	.error “<text>”
.export	匯出符號	.export <symbol>
.exportzp	匯出零頁符號	.exportzp <symbol>
.extern	宣告全域符號	.extern <symbol>
.externzp	宣告全域零頁符號	.externzp <symbol>
.floor	無條件捨去	.floor(<expr>)
.hibyte	高位元組	.hibyte(<expr>)
.if	條件式編譯	.if(<expr>)
.ifblank	條件組譯若參數為空	.ifblank(<symbol>)
.ifdef	條件式組譯若有定義	.ifdef(<symbol>)
.ifnblank	條件組譯若參數不為空	.ifnblank(<symbol>)
.ifndef	條件式組譯若無定義	.ifndef(<symbol>)
.import	匯入符號	.import <symbol>

虛擬指令	說明	語法
.importzp	匯入零頁符號	.importzp <symbol>
.incbin	插入二進位檔案	.incbin “<file>”
.include	引用檔案	.include “<file>”
.lobyte	低位元組	.lobyte(<expr>)
.local	巨集區域變數	.local <symbol>
.macro	定義巨集	.macro <name> <arg1>, <arg2>, …
.mod	取餘數運算	<expr> .mod <expr>
.not	布林反向運算	.not <expr>
.or	布林或運算	<expr> .or <expr>
.org	設定程式起始點	.org <expr>
.repeat	重複展開	.repeat <expr>
.res	保留空間	.res <expr>, <expr>
.round	四捨五入	.round(<expr>)
.scope	變數範圍區塊	.scope <symbol>
.segment	程式片段	.segment “<symbol>”
.setcpu	設定 CPU	.setcpu <ic_body>
.shl	左移	<expr> .shl <expr>
.shr	右移	<expr> .shr <expr>
.string	取得字串	.string(<symbol>)
.word	字組	.word <expr>
.xor	布林互斥或	<expr> .xor <expr>

5.3.2 .AND – 布林 AND 運算

◆ 語法

<symbol> = <expr1> .and <expr2>

◆ 說明

計算 expr1 & expr2

◆ 範例

```
.if(0 .and 1)
    .error
.endif
```

5.3.3 .BANKBYTE – 取得 bank byte

◆ 語法

.bankbyte(<expr>)

◆ 說明

取得 <expr> 高位元的 bank byte (bit 16~23)

◆ 範例

Label1_bank = .bankbyte(label1)

5.3.4 .BITAND – 位元 AND 運算

◆ 語法

<symbol> = <expr1> .and <expr2>

◆ 說明

計算 expr1 & expr2

◆ 範例

Ans = 1 .bitand 3

; Ans = 1

5.3.5 .BITNOT – 位元 NOT 運算

◆ 語法

<symbol> = .bitnot <expr>

◆ 說明

將 expr 每個 bit 反向，位元寬度是 32bit。若將一個 0 反向，將得到 32 bit 的 1。

◆ 範例

Ans = .bitnot 3

; Ans = 0xFFFFFFFC

5.3.6 .BITOR – 位元 OR 運算

◆ 語法

<symbol> = <expr1> .bitor <expr>

◆ 說明

計算 expr1 | expr2

◆ 範例

Ans = 3 .bitor 6

; Ans = 7

5.3.7 .BITXOR – 位元 XOR 運算

◆ 語法

<symbol> = <expr1> .xor <expr2>

◆ 說明

計算 expr1 互斥或 expr2

◆ 範例

Ans = 1 .xor 3

; Ans = 2

5.3.8 .BLANK – 參數是否為空

◆ 語法

.blank(<symbol>)

◆ 說明

會得到表示參數<symbol>是否為空的布林值。用於巨集之中可檢查呼叫端是否有傳遞指定的參數。

◆ 範例

```
.macro M1 arg1
   .if ( .blank(arg1) )
      .error
   .endif
.endmacro
```

5.3.9 .BYTE – 低位元組

◆ 語法

<symbol> = .byte(<expr>)

◆ 說明

取得 expr 的低位元一個 byte.

◆ 範例

Ans = .byte(0x1234)

; Ans = 0x34

5.3.10 .CEIL – 無條件進位

◆ 語法

<symbol> = .ceil(<expr>)

◆ 說明

如果<expr> 是一個浮點數，則將之無條件進位到整數。

◆ 範例

Ans = .ceil(1.2)

; Ans = 2

5.3.11 .CODE – .segment “code” 的簡寫

◆ 語法

.code

◆ 說明

等同 .segment “code”

◆ 請參閱

.SEGMENT

5.3.12 .DATA – .segment “data” 的簡寫

◆ 語法

.data

◆ 說明

等同 .segment “data”

◆ 請參閱

.SEGMENT

5.3.13 .DEFINE – 定義

◆ 語法

.define <symbol> <expr>

◆ 說明

將運算式定義到符號，之後這個符號就代表著運算式。

◆ 範例

.define AAA 1 + 2

Ans = AAA

;Ans = 3

◆ 請參閱

.DEFINED

5.3.14 .DEFINED – 查詢是否有被定義過

◆ 語法

.defined(<symbol>)

◆ 說明

若<symbol>曾經被定義過，則結果為 true(1)，否則為 false(0)。一般情形判斷符號有無定義，使用 .ifdef <symbol>即可。若要條件為多個符號皆有定義才算成立，勢必使用巢狀.ifdef。而.defined正好可解決此種窘境:

```
.ifdef (symbol1)
    .ifdef(symbol2)
            <statements>
     .endif
.endif
```

可改寫成

```
.if (.defined(symbol1) && .defined(symbol2) )
      <statements>
.endif
```

◆ 範例

```
.if ( .defined(def_name))
      .error
      ; 因為並沒有定義 def_name 符號，此處程式不會被組譯。
.endif
```

◆ 請參閱

.DEFINE

5.3.15 .ELSE – 條件式組譯否則區塊

◆ 語法

```
.if(<expr>)
      <statements>
.else
      <statements>
.endif
```

◆ 說明

若 if 不成立則組譯此區塊

◆ 範例

```
.if( 0 )
    .error
.else
    ; do something
.endif
```

◆ 請參閱

.IF, .ELSEIF

5.3.16 .ELSEIF – 條件式組譯否則再判斷區塊

◆ 語法

```
.if(<expr>)
    <statements>
.elseif (<expr>)
    <statements>
.endif
```

◆ 說明

若先前的 if 或 elseif 皆不成立，則在判斷 <expr>是否成立，成立則組譯此區塊。

◆ 範例

```
TempVar = 1
.if( TempVar < 1 )
    .error
.elseif(TempVar < 2)
    ; do something
.endif
```

◆ 請參閱

.IF, .ELSE

5.3.17 .ENDIF – 結束條件式組譯區塊

◆ 語法

```
.if(<expr>)
    <statements>
.else if (<expr>)
    <statements>
.endif
```

◆ 說明

結束由.if 開始的條件式組譯區塊

◆ 請參閱

.IF, .ELSE, ELSEIF

5.3.18 .ENDMACRO – 結束巨集定義區塊

◆ 語法

```
.macro <symbol> [<arg1> [,<arg2>…] ]
    <statements>
.endmacro
```

◆ 說明

結束由.macro 開始的巨集定義區塊。

◆ 請參閱

.macro

5.3.19 .ENDREPEAT – 結束重複區塊

◆ 語法

```
.repeat <expr>
    <statements>
.endrepeat
```

◆ 說明

結束由.repeat 開始的重複區塊。

◆ 請參閱

.repeat

5.3.20 .ENDSCOPE – 結束變數範圍區塊

◆ 語法

```
.scope <symbol>
    <statements>
.endscope
```

◆ 說明

結束由.repeat 開始的重複區塊。

◆ 請參閱

.scope

5.3.21 .ENDSTRUCT – 結束結構區塊

◆ 語法

.endstruct

◆ 說明

結束由.struct 開始的區塊。

◆ 請參閱

.struct

5.3.22 .EQU – 定義常數

◆ 語法

<symbol> .equ <expr>

◆ 說明

定義一個常數<symbol>並賦值<expr>。<expr>必須是此時此刻能計算的常數值，常數一經定義則不可更動其值。

◆ 範例

MyInteger .equ 1

5.3.23 .ERROR – 產生一個組譯錯誤

◆ 語法

.error [“<message>”]

◆ 說明

產生一個錯誤，若有指定錯誤訊息則使用。錯誤訊息必須為雙引號引住的文字。

◆ 範例

.error “argument out of range”

5.3.24 .EXPORT – 匯出符號

◆ 語法

.export <symbol>

◆ 說明

效果與.extern 相同。此項目僅是為了相容性而建立的別名。

◆ 請參閱

.extern

5.3.25 .EXPORTZP – 匯出零頁符號

◆ 語法

.exportzp <symbol>

◆ 說明

效果與.externzp 相同。此項目僅是為了相容性而建立的別名。

◆ 請參閱

.externzp

5.3.26 .EXTERN – 宣告外部符號

◆ 語法

.extern <symbol>

◆ 說明

宣告 symbol 為一個全域符號(Global symbol)。外部符號可以是本模組中定義的，亦可引用其他模組所定義的符號。多個模組做連結時，不可存在多個相同名稱的全域符號。亦不可存在任何全域符號未曾賦值。

◆ 範例

僅於多模組連結之情況，此指令方有意義。故以下範例將分為三個檔案以便說明。

```
;----- header.h------
.ifndef HEADER_H
.define HEADER_H
.extern GLOBAL_LABEL
.endif

;------ module1.s ------
.include “header.h”
      jmp GLOBAL_LABEL ;jump to module2

;------ module2.s -----
.include “header.h”
GLOBAL_LABEL:
      nop
```

◆ 請參閱

.externzp

5.3.27 .EXTERNZP – 宣告外部零頁符號

◆ 語法

.externzp <symbol>

◆ 說明

宣告 symbol 為一個全域符號(Global symbol)，在指令使用到此符號時，優先使用零頁定址模式。為此符號賦值時，僅可給予零頁範圍內之值(0x00 ~ 0xFF)。賦值超出範圍則組譯器報錯。通常.externzp 用於全域的記憶體位址定義，而.extern 用於全域副程式定義。

◆ 請參閱

.extern

5.3.28 .FLOOR –無條件捨去

◆ 語法

<symbol> = .floor(<expr>)

◆ 說明

如果<expr> 是一個浮點數，則將之無條件捨去到整數。

◆ 範例

```
Ans = .floor(1.2)
; Ans == 1
```

5.3.29 .HIBYTE – 高位元組

◆ 語法

<symbol> = .hibyte(<expr>)

◆ 說明

取得 expr 的高位元一個 byte. (bit 8~15)

◆ 範例

```
Ans = .hibyte(0x1234)
; Ans == 0x12
```

5.3.30 .IF – 條件式組譯

◆ 語法

```
.if(<expr>)
    <statements>
.endif
```

◆ 說明

如果<expr>成立則組譯此區塊。

<expr>通常為布林型態運算式，例如a==b。

◆ 範例

```
Tmp = 1 + 2 * 3
.if(Tmp != 7)
    .error
.endif
```

5.3.31 .IFBLANK – 條件組譯若參數為空

◆ 語法

```
.ifblank(<symbol>)
    <statements>
.endif
```

◆ 說明

此指令為 .if(.blank(<symbol>))的簡寫。

◆ 請參閱

.blank

5.3.32 .IFDEF – 條件式組譯若有定義

◆ 語法

```
.ifdef(<symbol>)
    <statements>
.endif
```

◆ 說明

若<symbol>已定義則組譯此區塊。

◆ 範例

```
.ifdef(UNDEFINE_SYM)
    .error
.endif
```

◆ 請參閱

.if, .defined

5.3.33 .IFNBLANK – 條件式組譯若參數不為空

◆ 語法

```
.ifnblank(<symbol>)
    <statements>
.endif
```

◆ 說明

此指令為 .if(！.blank(<symbol>))之簡寫。

◆ 請參閱

.blank

5.3.34 .IFNDEF – 條件式組譯若無定義

◆ 語法

```
.ifndef(<symbol>)
    <statements>
.endif
```

◆ 說明

若<symbol>未曾定義則組譯此區塊。

◆ 範例

```
.ifdef(UNDEFINE_SYM)
    .error
.endif
```

◆ 請參閱

.if, .defined

5.3.35 .IMPORT – 匯入符號

◆ 語法

```
.import <symbol>
```

◆ 說明

效果與.extern 相同。此項目僅是為了相容性而建立的別名。

◆ 請參閱

.extern

5.3.36 .IMPORTZP – 匯出零頁符號

◆ 語法

```
.importzp <symbol>
```

◆ 說明

效果與.externzp 相同。此項目僅是為了相容性而建立的別名。

◆ 請參閱

.externzp

5.3.37 .INCBIN – 插入二進位檔案

◆ 語法

```
.incbin “<file>”
```

◆ 說明

以二進位元的方式將插入檔案<file>之內容。與.include 的差別是，.incbin 直接使用目標檔案內容，而.include 會以文字的方式解譯目標檔案內的指令。.incbin 通常使用於音效、圖形等二進位檔案。

◆ 範例

```
L_RES_Voice1:
.incbin “d:\abc\voice1.v8lx”
```

5.3.38 .INCLUDE – 引用檔案

◆ 語法

```
.include “<file>”
```

◆ 說明

<file>必須為另一個組譯原始檔，組譯器將會暫停目前檔案的組譯，開始組譯<file>檔案，並在<file>結束後回到目前位置。

◆ 範例

```
---- a1.h ----
.ifndef A1_H
.define A1_H
; content
.extern G_Func1

.endif

---- a1.s ----
.include “a1.h”
G_Func1:
ret
```

5.3.39 .LOBYTE – 低位元組

◆ 語法

```
<symbol> = .lobyte( <expr> )
```

◆ 說明

取得 expr 的低位元一個 byte. (bit 0~7)

◆ 範例

```
Ans = .lobyte(0x1234)
; Ans == 0x34
```

5.3.40 .LOCAL – 巨集區域變數

◆ 語法

.local <symbol>

◆ 說明

在巨集裡宣告內部的標記，且僅在巨集中具效力。在宣告時<label>可能與巨集定義外的另一個標記完全相同；但二者之間並不會衝突。即使在巢狀式的巨集呼叫中，每個引用將有它自己局部的定義。

◆ 範例

```
.macro M_x1
    .local LL_exit
    jmp LL_exit
LL_exit:
.endmacro
```

5.3.41 .MACRO – 定義巨集

◆ 語法

```
.macro <symbol> [<arg1>, <arg2>, …]
    <statement>
.endmacro
```

◆ 說明

定義<symbol>為巨集，爾後使用此符號則展開這段巨集。巨集可以有參數，呼叫巨集所使用的參數數目必須符合巨集定義之數目。

◆ 範例

```
.macro M_LDXY arg_value_x, arg_value_y
    LDX #arg_value_x
    LDY #arg_value_y
.endmacro
```

5.3.42 .MOD – 取餘數運算

◆ 語法

<symbol> = <expr1> .mod <expr2>

◆ 說明

計算 expr1 / expr2 的餘數

◆ 範例

```
ans = 5 .mod 3
; ans == 2
```

5.3.43 .NOT – 布林反向運算

◆ 語法

<symbol> = .not <expr1>

◆ 說明

計算 expr1 的反向

◆ 範例

```
ans = .not 1
; ans == 0
```

5.3.44 .OR – 布林或運算

◆ 語法

<symbol> = <expr1> .or <expr2>

◆ 說明

計算 expr1 || expr2

◆ 範例

```
ans = 0 .or 1
; ans == 1
```

5.3.45 .ORG – 設定程式起始點

◆ 語法

.org <expr>

◆ 說明

開始一個新的 segment，並且定位到<expr>計算出來的位址。

◆ 範例

```
.org 0x7e0
    .word L_TM2_INT
.code
L_TM2_INT:
    RTI
```

5.3.46 .REPEAT – 重複展開

◆ 語法

.repeat <expr>

```
        <statement>
    .endrepeat
```

◆ 說明

重複組譯<statement>多次，次數由<expr>指定。

◆ 範例

```
.org 0x7e0
    .word L_TM2_INT
.code
L_TM2_INT:
    RTI
```

5.3.47 .RES – 保留空間

◆ 語法

```
.res <expr1>, <expr2>
```

◆ 說明

保留<expr1>指定大小的空間，並填入<expr2>所指定的值。

◆ 範例

```
; Reserve 12 bytes of memory with value $AA
.res 12, $AA
```

5.3.48 .ROUND – 四捨五入

◆ 語法

```
.round(<expr>)
```

◆ 說明

取得<expr>四捨五入到整數的值。

5.3.49 .SCOPE – 變數範圍區塊

◆ 語法

```
.scope <symbol>
    <statements>
.endscope
```

◆ 說明

開始一個指定名稱的程式區域。在.scope 到.endscope 所包含的範圍內，新定義的符號可以直接在內部存取。唯有在外部使用符號，須加上前綴字。

scope 的名稱不可和其他符號衝突。

◆ 範例

```
.scope Error       ; Start new scope named Error
    None = 0
    File = 1
    Parse = 2
.endscope          ; close scope
LDA #Error::File   ; use symbol from scope Error
```

◆ 請參閱

.endscope

5.3.50 .SEGMENT – 程式片段

◆ 語法

.segment “<symbol>”

◆ 說明

切換到另一個程式片段。.segment 指令必須給一個字串參數為 segment 名稱。可用的名稱與所選 IC 相關，請查閱指定 IC 檔。

◆ 範例

```
.segment “tm0_int”
    .word L_tm0_int
.code
L_tm0_int:
```

◆ 請參閱

.code

5.3.51 .SETCPU – 指定 CPU

◆ 語法

.setcpu <symbol>

◆ 說明

可以在檔案的最前端標明 IC Body，不可以多次切換 IC Body。

◆ 範例

.setcpu NY8L030A

5.3.52 .SHL – 左移

◆ 語法

<expr1> .shl <expr2>

◆ 說明

計算<expr1> 左移 <expr2>的結果。

◆ 範例

```
Result = 2 .shl 1
; Result = 4
```

5.3.53 .SHR – 右移

◆ 語法

<expr1> .shr <expr2>

◆ 說明

計算<expr1> 右移 <expr2>的結果。

◆ 範例

```
Result = 2 .shr 1
; Result = 1
```

5.3.54 .STRING – 取得字串

◆ 語法

.string(<symbol>)

◆ 說明

取得<symbol>符號所定義的字串。

通常用於在 macro，可得到參數帶入的字串。

◆ 範例

```
.macro M_inc_v8lx name
    .incbin .string(name)
.endmacro
```

5.3.55 .WORD – 字組

◆ 語法

.word <expr1>

◆ 說明

在目前位置寫入一個字組(2 位元組)的資料，內容為 <expr1>。

◆ 範例

```
.word 0x12EF
```

5.3.56 .XOR – 布林互斥或

◆ 語法

<expr1> .xor <expr2>

◆ 說明

計算<expr1> 互斥或 <expr2>的結果。

◆ 範例

Result = 0 .xor 1

; Result = 1

6 巨集指令

巨集內容為使用者定義的程式碼，在組譯器編譯時會用程式碼會替換掉巨集名稱。

巨集定義的本身還可以包含一個連續的組合語言及虛擬指令，使用巨集在組合語言程式編碼上具有較佳的可閱讀性及彈性。使用巨集的優點如下：

- 具高階語言的概念，可增加程式的可閱讀性及可靠性。
- 對於被使用性較高的功能有統一固定的解決方案。
- 較易修正。
- 增加可測試性。

應用的範圍可能包含了用來產生一個複雜的表、經常被使用的程式碼、和較複雜的運算。

6.1 巨集指令 for NY4、NY5、NY7、NY8A、NY9

6.1.1 巨集語法

*NYASM*的巨集是依照下列的語法做定義：

<label> macro [<arg1>,<arg2> ..., <argn>]

:

:

endm

這裡的<label>是一個在組譯時有效的標記名稱而且<arg>對於巨集來說是一個可為任意個數的選項。在巨集被展開的同時將會依相對應的參數位置指定這些參數的值。巨集本身可以包含*NYASM*虛擬指令或是*NYASM*巨集虛擬指令（例如LOCAL）。參考第5.2章，*NYASM*將會對巨集本身做展開…等處理，直到*NYASM*碰到EXITM或ENDM這兩個虛擬指令。

注意：在巨集當中是不允許向前引用。

6.1.2 巨集虛擬指令

這些虛擬指令只適於在巨集本身內部使用。（請參照第5.2.1章，有關於這些指令的詳細說明）：

- MACRO
- LOCAL
- EXITM
- ENDM

*NYASM*巨集本身支援上述的虛擬指令及其他的虛擬指令。

6.1.3 文字替代

在巨集本身的內部可以有字串的取代及運算式的判定。巨集所帶的參數可以在巨集本身內部的任何一個地方引用。

命令	說明
<arg>	參數的文字將在巨集展開時做替代。

```
define_table macro num_of_entry
        local a = 0
        while a < num_of_entry
        dw 0
        a += 1
        endw
endm
```

when invoked, would generate:

```
dw 0    ; 1st
dw 0    ; 2nd
  :
  :
dw 0    ; (num_of_entry-1)-th
dw 0    ; (num_of_entry)-th
```

6.1.4 巨集的用法

一個巨集已經被事先定義過後，即可在程式來源模組當中的任何一個位置使用巨集呼叫來引用，如下所述：

<macro_name> [<arg>, ..., <arg>]

這裡的<macro_name>是一個先前已經被定義過的巨集名稱。而參數則是視實際需求再加入，參數之間彼此以逗號分隔。巨集呼叫自己將不會佔用到任何的記憶體位置。然而巨集將會從目前的記憶體位置開始展開。EXITM與ENDM這兩個虛擬指令是被用來終止目前巨集展開，(參考第5章)。在巨集展開的時候，虛擬指令EXITM將會暫停目前巨集展開的動作，而在虛擬指令EXITM與ENDM之間的所有程式碼將會被忽略。假如是巢狀式的巨集，則EXITM將會終止目前這一層巨集的展開，回到上一層巨集展開。

6.2 NY8L

6.2.1 巨集語法

NY8L的巨集是依照下列的語法做定義：

```
.macro <label> [<arg1>,<arg2> ..., <argn>]
:
:
.endmacro
```

這裡的<label>是一個在組譯時有效的標記名稱而且<arg>對於巨集來說是一個可為任意個數的選項。在

巨集被展開的同時將會依相對應的參數位置指定這些參數的值。巨集本身可以包含*NYASM* 虛擬指令或是*NYASM* 巨集虛擬指令(例如.LOCAL)。參考第5.3章，*NYASM* 將會對巨集本身做展開…等處理，直到*NYASM* 碰到.ENDMACRO虛擬指令。

注意：在巨集當中是不允許向前引用。

6.2.2 巨集虛擬指令

這些虛擬指令只適於在巨集本身內部使用。(請參照第5.3章，有關於這些指令的詳細說明)：

.MACRO – 定義巨集

.ENDMACRO – 結束巨集定義區塊

.LOCAL – 巨集區域變數

.IFBLANK – 條件組譯若參數為空

.IFNBLANK – 條件式組譯若參數不為空

.BLANK – 參數是否為空

NYASM 巨集本身支援上述的虛擬指令及其他的虛擬指令。

6.2.3 文字替代

在巨集本身的內部可以有字串的取代及運算式的判定。巨集所帶的參數可以在巨集本身內部的任何一個地方引用。

命令	說明
<arg>	參數的文字將在巨集展開時做替代。

◆ 範例

```
.macro CAJE value, label
        cmp     #value
        jz  label
.endmacro
```

6.2.4 巨集的用法

一個巨集已經被事先定義過後，即可在程式來源模組當中的任何一個位置使用巨集呼叫來引用，如下所述：

<macro_name> [<arg>, ..., <arg>]

這裡的<macro_name>是一個先前已經被定義過的巨集名稱，而參數則是視實際需求再加入，參數以逗號分隔。巨集呼叫將不會佔用到任何的記憶體位置。然而巨集將會從目前的記憶體位置開始展開。巨集呼叫時使用的參數可以少於定義時的參數，並可藉由.blank 檢查呼叫端是否有傳遞指定的參數。

7 表示式的語法與運算

這一章將說明在 *NYASM* 中使用不同的表示式格式、語法及運算。

7.1 NY4、NY5、NY7、NY8A、NY9

內容：

7.1.1 數字常數和格式

NYASM 支援下列的數字格式：十六進制、十進制、八進制和二進制。數字系統在沒有特別去變更時將會使用預設值做為目前數字系統，預設為十進制格式。數字系統將會決定機械碼檔上常數值。*NYASM* 只支援無號數。

以下表格提供不同種類數值型態表示：

表格 7-1數值格式

類型	語法	範例
十進位	D’<digits>’	D’100’
十六進位	H’<hex_digits>’ 0x<hex_digits> <hex_digits>h	H’9f’ 0x9f 9fh
八進位	O’<octal_digits>’	O’777’
二進位	B’<binary_digits>’ <binary_digits>b	B’00111001’ 00111001b

表格 7-2算術運算元及優先順序

運算元		範例
$	取得程式計數器	goto $ + 3
(	左括號	1 + (d * 4)
)	右括號	(Length + 1) * 256
!	反邏輯（邏輯補數)	if ! (a == b)
-	負（二的補數)	-1 * Length
~	一的補數	flags = ~flags
high	取得24位元值的最高位元值	mvma high(0x121314) ;accumulator will contain 0x12
mid	取得24位元值的中間位元值	mvma mid(0x121314) ;accumulator will contain 0x13

運算元		範例
low	取得24位元值的最低位元值	mvma low(0x121314) ;accumulator will contain 0x14
high0	取得24位元值的最高位元的低位4位元值	mvma high0(0x123456) ;accumulator will contain 0x2
high1	取得24位元值的最高位元的高位4位元值	mvma high1(0x123456) ;accumulator will contain 0x1
mid0	取得24位元值的中間位元的低位4位元值	mvma mid0(0x123456) ;accumulator will contain 0x4
mid1	取得24位元值的中間位元的高位4位元值	mvma mid1(0x123456) ;accumulator will contain 0x3
low0	取得24位元值的最低位元的低位4位元值	mvma low0(0x123456) ;accumulator will contain 0x6
low1	取得24位元值的最低位元的高位4位元值	mvma low1(0x123456) ;accumulator will contain 0x5
*	算術乘	a = b * c
/	算術除	a = b / c
%	算術餘	entry_len = tot_len % 16
+	算術加	tot_len = entry_len * 8 + 1
-	算術減	entry_len = (tot - 1) / 8
<<	位元左移	flags = flags << 1
>>	位元右移	flags = flags >> 1
>=	大或等於	if entry_idx >= num_entries
>	大於	if entry_idx > num_entries
<	小於	if entry_idx < num_entries
<=	小或等於	if entry_idx <= num_entries
==	等於	if entry_idx == num_entries
!=	不等於	if entry_idx != num_entries
&	位元與	flags = flags & ERROR_BIT
^	位元互斥或	flags = flags ^ ERROR_BIT
|	位元或	flags = flags | ERROR_BIT
&&	邏輯與	if (len == 512) && (b == c)
||	邏輯或	if (len == 512) || (b == c)
=	設值	entry_index = 0
+=	加並設值	entry_index += 1
-=	減並設值	entry_index -= 1
*=	乘並設值	entry_index *= entry_length
/=	除並設值	entry_total /= entry_length

運算元		範例
%=	求餘並設值	entry_index %= 8
<<=	位元左移並設值	flags <<= 3
>>=	位元右移並設值	flags >>= 3
&=	位元與並設值	flags &= ERROR_FLAG
|=	位元或並設值	flags |= ERROR_FLAG
^=	位元互斥或並設值	flags ^= ERROR_FLAG
++	遞增量為1	i ++
--	遞減量為1	i --

7.1.2 高位/中位元/低位

◆ 語法

high <operand>

mid <operand>

low <operand>

◆ 說明

這些運算元是被用來對一個多位元的標記值取其中一個位元做回傳。這是被用來做動態指標的計算，也可能被用來當對一個目錄的讀取和寫入指令。

7.1.3 增量/減量 (++/--)

◆ 語法

<variable>++

<variable>--

◆ 說明

為變數的值做增加或減少。這些運算元是自己獨立一行且是以自己當作運算元。他們不可以再被加入到其他的運算式當中。

◆ 範例

```
LoopCount = 4
while LoopCount > 0
   nop
   LoopCount --
Endw
```

7.2 NY8L

內容：

7.2.1 數字常數和格式

NYASM 支援下列的數字格式：十六進制、十進制、八進制和二進制。數字系統在沒有特別去變更時將會使用預設值做為目前數字系統，預設為十進制格式。數字系統將會決定機械碼檔上常數值。*NYASM* 只支援無號數。

以下表格提供不同種類數值型態表示：

表格 7-3數值格式

類型	語法	範例
十進位	<digits>	100
十六進位	$<hex_digits>’ 0x<hex_digits>	$9f 0x9f
二進位	%<binary_digits>’	%00111001

表格 7-4算術運算元及優先順序

運算元		範例
(	左括號	1 + (d * 4)
)	右括號	(Length + 1) * 256
!	反邏輯（邏輯補數)	if ! (a == b)
-	負（二的補數)	-1 * Length
~ .bitnot	一的補數	flags = ~flags
.bankbyte	取得24位元值的最高位元值	mvma high(0x121314) ;accumulator will contain 0x12
.hibyte	取得24位元值的中間位元值	mvma mid(0x121314) ;accumulator will contain 0x13
.lobyte	取得24位元值的最低位元值	mvma low(0x121314) ;accumulator will contain 0x14
*	算術乘	a = b * c
/	算術除	a = b / c
%	算術餘	entry_len = tot_len % 16
+	算術加	tot_len = entry_len * 8 + 1
-	算術減	entry_len = (tot - 1) / 8

運算元		範例
<<	位元左移	flags = flags << 1
>>	位元右移	flags = flags >> 1
>=	大或等於	if entry_idx >= num_entries
>	大於	if entry_idx > num_entries
<	小於	if entry_idx < num_entries
<=	小或等於	if entry_idx <= num_entries
==	等於	if entry_idx == num_entries
!=	不等於	if entry_idx != num_entries
& .bitand	位元與	flags = flags & ERROR_BIT
^	位元互斥或	flags = flags ^ ERROR_BIT
| .bitor	位元或	flags = flags | ERROR_BIT
&& .and	邏輯與	if (len == 512) && (b == c)
|| .or	邏輯或	if (len == 512) || (b == c)
.round	四捨五入	.round(2.345)
.ceil	無條件進位	.ceil(2.345)
.floor	無條件捨去	.floor(2.345)

7.2.2 高位/中位元/低位

◆ 語法

.bankbyte <operand>

.hibyte <operand>

.lobyte <operand>

◆ 說明

這些運算元是被用來對一個多位元的標記值取其中一個位元做回傳。這是被用來做動態指標的計算，也可能被用來當對一個目錄的讀取和寫入指令。

8 改版記錄

版本	日期	內 容 描 述	修正頁
1.00	2007/12/20	新發佈。	-
1.01	2009/10/12	改版。	-
1.1	2010/01/11	新增 NY4 系列 MCU。	58
1.2	2010/07/20	1. 新增 NY4/NY5 系列新母體。 2. 新增中文警告/錯誤訊息。	59 63
1.3	2010/08/17	相容於 Windows 7。	12
1.4	2012/02/29	修改 NY5B/5C 系列 MCU。	59
1.5	2013/06/25	1. 使用 *NYASM* 需搭配 Windows XP 以上作業系統。 2. 新增 NY4(B)與 NY7 IC 系列 MCU 清單。	12 59
1.6	2013/08/16	修改 NY7 系列 MCU 清單。	61
1.7	2014/02/24	1. 修改 MACRO 程式範例。 2. 修改預設數值系統為十進位。 3. 修改警告與錯誤訊息。 4. 詞彙表新增 Forward Reference 向前參考解釋。	40 42, 50, 70 62 69
1.8	2014/05/16	新增 NY8 系列 MCU。	60
1.9	2014/11/17	變更 MCU 列表 IC 母體: NY4B018C、NY4B038C、NY4B058C。NY5C158C、NY5C185C、NY5C345C。	58
2.0	2015/01/29	1. 新增二進位數字標記法。 2. 修改位元運算範例。	50 50
2.1	2015/07/27	修改 UI 介紹。	18
2.2	2015/11/20	增加 NY9UB 系列 MCU。	61
2.3	2016/02/03	1. 移除 NY4xxxxA/NY5xxxxA 系列 MCU，僅保留 NY5AxxxA 系列。 2. 增加 NY8 系列 MCU。	- 63

版本	日期	內 容 描 述	修正頁
2.4	2016/05/20	1. 增加 NY6 系列 MCU。 2. 增加 NY8A051C/51D MCU。	60 63
2.5	2016/08/22	增加 NY8A053D MCU。	63
2.6	2016/11/18	1. 移除 NY5AA 系列。 2. 增加 NY9UP01A。	- 79
2.7	2017/05/23	1. 支援 NY8L 系列。 2. 新增 NY8A054A MCU。 3. 新增 NY8L MCU。	42, 62, 67 78 78
2.8	2017/08/03	新增 NY8A054D MCU。	78
2.9	2017/11/17	1. 修正 List 指令說明。 2. 增加 NY8A051E MCU。	34, 72 79
3.0	2018/02/08	新增 NY8B062D MCU。	79
3.1	2018/08/27	1. 刪除 DT 指令。 2. 移除 NY8A057A、NY8B073A、NY8B074A MCU。 3. 移除 NY6C450A ~ NY6C720A MCU。	- - -
3.2	2018/11/21	新增 NY8B062A MCU。	78
3.3	2019/02/19	新增 NY8A051F、NY9UP08A MCU。	77, 78
3.4	2019/05/28	新增 NY5P025J、NY5P055J、NY5P085J、NY5B035C、NY5B045C、NY8A050D、NY8AE51D、NY8B062B MCU。	75
3.5	2019/08/22	移除 NY8L005A、NY8L010A，並新增 NY8LP10A、NY8LP11A。	78
3.6	2019/11/14	新增 NY5P 系列、NY5A018C、NY5A025C、NY8BM72A MCU。	76
3.7	2020/03/16	新增 NY5AC 系列、NY5BC 系列 MCU。	75
3.8	2020/08/18	1. 增加.word、.bitor 指令說明。 2. 增加 NY6P 系列、NY8A054E、NY8B061D。	41, 57 75

3.9	2020/11/12	1. 移除 NY4B018B / NY5AxxxB / NY5BxxxB / NY5C112B / 132B / 158B / 185B/ 225B / 265B / 305B / 345B。 2. 新增 NY8A053E / NY9UP02A。	- 79
4.0	2021/01/27	1. 移除 NY6A003A / NY6A005A / NY8L050A。 2. 新增 NY8B062E / NY8TM52D。	- 80
4.1	2021/05/18	新增 NY5QxxxA / NY8B060E / NY8BE62D。	77
4.2	2021/09/10	1. 刪除 DN、DB、DATA 指令。 2. 新增 NY5Q020A。	- 76
4.3	2021/11/11	新增 NY8TE64A。	79
4.4	2022/02/22	新增 NY5Q026A、NY5Q046A、NY5Q080A、NY5Q160A、NY8A051H、NY8AE51F。	76
4.5	2022/05/19	1. 更新系統需求，增加 win11。 2. 新增 NY8B060D。	8 79
4.6	2022/08/24	新增 NY8B061E。	79
4.7	2022/11/28	新增 NY4P045C、NY8B062F。	74
4.8	2023/02/15	新增 NY4P018C、NY4P065C、NY4P085C、NY4P105C、NY8A050E.。	74
4.9	2023/05/15	修正說明錯誤。	-
5.0	2023/08/21	增加 NY8A052E。	78
5.1	2024/02/22	1. 增加.align2 指令說明。 2. 增加 NY8BM61D、NY8BM62D。 3. 移除 NY8L020A、NY8L030A。	38 80 -

附錄A - 快速索引

在使用*NYASM*開發系統時可以利用本附錄所提供的簡略資料做參考。

內容：

A.1 *NYASM* 快速參考

下列的快速參考指南包含了*NYASM*組譯器所有的指令、虛擬指令和命令列選項。

表格 A-1 虛擬指令集

虛擬指令	說明	語法
控制型虛擬指令		
CONSTANT	宣告符號常數。	constant <label>[=<expr>,...,<label>[=<expr>]]
#DEFINE	定義一個文字替代標記。	#define <name> [<value>] #define <name> [<arg>,...,<arg>]
END	結束程式區段。	end
EQU	定義一個組譯常數。	<label> equ <expr>
ERROR	發佈一個錯誤訊息。	error "<text_string>"
#INCLUDATA	含括一個二進制資料檔。	#includata "<data_file>" [,<address>]
#INCLUDE	含括一個附加的源碼檔。	#include "<include_file>"
LIST	列印的選項。	list [<list_option>,...,<list_option>]
MESSG	產生使用者定義的訊息。	messg "<message_text>"
ORG	設定程式的起始點。	[<label>:] org <expr>
LINES	重新宣告每一頁的列數。	lines <value>
NEWPAGE	可在產生的List File中產生新的一頁。	Newpage <value>
RADIX	宣告數字格式。	radix <default_radix>
SUBTITLE	程式副標題。	subtitle "<sub_text>"
TITLE	程式標題。	title "<title_text>"
#UNDEFINE	刪除一個替代標記。	#undefine <label>
VARIABLE	宣告符號變數。	variable <label>[=<expr>,...,<label>[=<expr>]]
條件式組譯指令		
BREAK	從 **FOR**, **WHILE** 或 **REPEAT-UNTIL** 迴圈中跳離，或者從 **SWITCH** 區塊中跳到最 **SWITCH** 的最末端。	break [<Boolean expression>]

虛擬指令	說明	語法
CASE	屬於 **SWITCH** 區塊的一部分，**CASE** 必須在 **SWITCH** 區塊中使用。	switch <expression> case <expression 1>[,<expression 2>] <statements>
CONTINUE	跳至內部含有**CONTINUE**虛擬指令的**FOR**、**WHILE**或**REPEAT-UNTIL**迴圈的起始位置。 在迴圈內**CONTINUE**後面的表示式都將會被忽略。	continue [<Boolean expression>]
DEFAULT	**SWITCH**區塊的一部分，使用**SWITCH**區塊時必須要有**DEFAULT**的條件式。**DEFAULT**為表示**SWITCH**判斷式中的預設的組譯區塊。	default <statements>
ELSE	提供一個相對於**IF**的組譯區塊。*NYASM* 在**IF**的組譯區塊與**ELSE**的組譯區塊二者之間只會選擇一個做組譯。	else <statements>
ENDFOR	結束一個**FOR**迴圈。	endfor
ENDIF	結束條件式組譯區塊。	endif
ENDS	提供一個方便管理的結束指令，可以使用在**ENDFOR, ENDW, ENDSW, ENDIF**。	ends
ENDSW	結束條件式**SWITCH**組譯區塊。	endsw
ENDW	結束一個**WHILE**迴圈。	endw
FOR	執行FOR的迴圈計數。	for <iterator> = <expr1> to <expr2> [step <expr3>]
IF	條件式組譯程式區塊的起始。	if <expr>
IFDEF	假如符號已經有被定義過就執行。	ifdef <label>
IFNDEF	假如符號尚未被定義就執行。	ifndef <label>
REPEAT	至少會執行一次的迴圈。	Repeat <statements> until <Boolean expression>
SWITCH	條件式交換組譯區塊的起始。	switch <expr>
UNTIL	假如條件式成立就結束至少會執行一次的迴圈。	Repeat <statements> until <Boolean expression>
WHILE	**WHILE**所帶的條件若是成立則執行迴圈。	while <expr>
資料		
CBLOCK	定義一個常數區塊。	cblock [<expr>]
DW	宣告一個字元組(word)的資料。	[<label>] dw <expr>[,<expr>,...,<expr>]
ENDC	結束一個常數區段。	endc

虛擬指令	說明	語法
巨集		
ENDM	結束一個巨集定義。	endm
EXITM	從一個巨集離開。	exitm
EXPAND	巨集列表展開。	expand
LOCAL	宣告局部巨集變數。	local <label>[,<label>]
MACRO	宣告巨集定義。	<label> macro [<arg>,...,<arg>]
MAXMACRODEPTH	設定巨集展開的最大深度。	Maxmacrodepth [=] <expr>
NOEXPAND	關閉巨集的展開。	noexpand

表格 A-2 組譯器選項

選項	預設狀態	說明
c	Off	英文字母大小寫區分開啟/關閉 c=on 開啟 c=off 關閉
p	None	設定處理器的類型： /p=<processor_type> 這裡的<processor_type> 是指九齊科技的元件。 例如, NY5A005A。
unlockrsvmem	Locked	/unlockrsvmem 只適用於4-bit MCU。允許編輯保留記憶體區域。
nocfgblk	Configuration block required	/nocfgblk 只適用於4-bit MCU。在組譯階段忽略已存在的硬體組態設定。

表格 A-3 所支援的數字系統基底型態

型態	語法	舉例
十進位	D’<digits>’	D’100’
十六進位	H’<hex_digits>’ 0x<hex_digits> <hex_digits>h	H’9f’ 0x9f 9fh
八進位	O’<octal_digits>’	O’777’
二進位	B’<binary_digits>’	B’00111001’

表格 A-4 *NYASM* 算術運算元

運算元		舉例
$	取得程式計數器	goto $ + 3
(	左括號	1 + (d * 4)
)	右括號	(Length + 1) * 256
!	反邏輯（邏輯補數)	if ! (a == b)
-	負（二的補數)	-1 * Length
~	一的補數	flags = ~flags
high	取得24位元值的最高位元值	mvma high 0x121314 ;accumulator will contain 0x12
mid	取得24位元值的中間位元值	mvma mid 0x121314 ;accumulator will contain 0x13
low	取得24位元值的最低位元值	mvma low 0x121314 ;accumulator will contain 0x14
high0	取得24位元值的最高位元的低位4位元值	mvma high0 0x123456 ;accumulator will contain 0x2
high1	取得24位元值的最高位元的高位4位元值	mvma high1 0x123456 ;accumulator will contain 0x1
mid0	取得24位元值的中間位元的低位4位元值	mvma mid0 0x123456 ;accumulator will contain 0x4
mid1	取得24位元值的中間位元的高位4位元值	mvma mid1 0x123456 ;accumulator will contain 0x3
low0	取得24位元值的最低位元的低位4位元值	mvma low0 0x123456 ;accumulator will contain 0x6
low1	取得24位元值的最低位元的高位4位元值	mvma low1 0x123456 ;accumulator will contain 0x5
*	算術乘	a = b * c
/	算術除	a = b / c
%	算術餘	entry_len = tot_len % 16
+	算術加	tot_len = entry_len * 8 + 1
-	算術減	entry_len = (tot - 1) / 8
<<	位元左移	flags = flags << 1
>>	位元右移	flags = flags >> 1
>=	大或等於	if entry_idx >= num_entries
>	大於	if entry_idx > num_entries
<	小於	if entry_idx < num_entries
<=	小或等於	if entry_idx <= num_entries

運算元		舉例
==	等於	if entry_idx == num_entries
!=	不等於	if entry_idx != num_entries
&	位元與	flags = flags & ERROR_BIT
^	位元互斥或	flags = flags ^ ERROR_BIT
|	位元或	flags = flags | ERROR_BIT
&&	邏輯與	if (len == 512) && (b == c)
||	邏輯或	if (len == 512) || (b == c)
=	設值	entry_index = 0
+=	加並設值	entry_index += 1
-=	減並設值	entry_index -= 1
*=	乘並設值	entry_index *= entry_length
/=	除並設值	entry_total /= entry_length
%=	求餘並設值	entry_index %= 8
<<=	位元左移並設值	flags <<= 3
>>=	位元右移並設值	flags >>= 3
&=	位元與並設值	flags &= ERROR_FLAG
|=	位元或並設值	flags |= ERROR_FLAG
^=	位元互斥或並設值	flags ^= ERROR_FLAG
++	遞增量為1	i ++
--	遞減量為1	i --

A.2 MCU 列表

表格 A-5 MCU 清單

No.	IC 母體	PROG ROM 空間	DATA ROM 空間	ROM 保留地址	I/O 腳個數
1	NY4P018C	16K x 10	48K x 10	0x001F--0x07FF	8 I/O
2	NY4P045C	16K x 10	112K x 10	0x001F--0x07FF	8 I/O
3	NY4P065C	16K x 10	160K x 10	0x001F--0x07FF	8 I/O
4	NY4P085C	16K x 10	208K x 10	0x001F--0x07FF	8 I/O
5	NY4P105C	16K x 10	256K x 10	0x001F--0x07FF	8 I/O
6	NY4A003B	12K x 10	12K x 10	0x001F--0x07FF	4 I/O
7	NY4A005B	16K x 10	16K x 10	0x001F--0x07FF	4 I/O
8	NY4A008B	16K x 10	24K x 10	0x001F--0x07FF	4 I/O
9	NY4A011B	16K x 10	32K x 10	0x001F--0x07FF	4 I/O
10	NY4B003B	12K x 10	12K x 10	0x001F--0x07FF	8 I/O
11	NY4B005B	16K x 10	16K x 10	0x001F--0x07FF	8 I/O
12	NY4B008B	16K x 10	24K x 10	0x001F--0x07FF	8 I/O
13	NY4B011B	16K x 10	32K x 10	0x001F--0x07FF	8 I/O
14	NY4B018C	16K x 10	48K x 10	0x001F--0x07FF	8 I/O
15	NY4B025B	16K x 10	64K x 10	0x001F--0x07FF	8 I/O
16	NY4B038C	16K x 10	96K x 10	0x001F--0x07FF	8 I/O
17	NY4B045B	16K x 10	112K x 10	0x001F--0x07FF	8 I/O
18	NY4B058C	16K x 10	144K x 10	0x001F--0x07FF	8 I/O
19	NY4B065B	16K x 10	160K x 10	0x001F--0x07FF	8 I/O
20	NY4B075B	16K x 10	184K x 10	0x001F--0x07FF	8 I/O
21	NY4B085B	16K x 10	208K x 10	0x001F--0x07FF	8 I/O
22	NY4B095B	16K x 10	232K x 10	0x001F--0x07FF	8 I/O
23	NY4B105B	16K x 10	256K x 10	0x001F--0x07FF	8 I/O
24	NY4B115B	16K x 10	280K x 10	0x001F--0x07FF	8 I/O
25	NY4B125B	16K x 10	304K x 10	0x001F--0x07FF	8 I/O
26	NY4B145B	16K x 10	352K x 10	0x001F--0x07FF	8 I/O
27	NY4B165B	16K x 10	400K x 10	0x001F--0x07FF	8 I/O
28	NY5P025B	16K x 10	64K x 10	0x001F--0x0BFF	16 I/O
29	NY5P055B	16K x 10	136K x 10	0x001F--0x0BFF	16 I/O
30	NY5P085B	16K x 10	208K x 10	0x001F--0x0BFF	16 I/O
31	NY5P185B	16K x 10	448K x 10	0x001F--0x0BFF	16 I/O
32	NY5P025J	16K x 10	64K x 10	0x001F--0x0BFF	16 I/O
33	NY5P055J	16K x 10	136K x 10	0x001F--0x0BFF	16 I/O

No.	IC 母體	PROG ROM 空間	DATA ROM 空間	ROM 保留地址	I/O 腳個數
34	NY5P085J	16K x 10	208K x 10	0x001F--0x0BFF	16 I/O
35	NY5P185J	16K x 10	448K x 10	0x001F--0x0BFF	16 I/O
36	NY5P345J	16K x 10	832K x 10	0x001F--0x0BFF	16 I/O
37	NY5P520J	16K x 10	1248K x 10	0x001F--0x0BFF	24 I/O
38	NY5P720J	16K x 10	1728K x 10	0x001F--0x0BFF	24 I/O
39	NY5P1K0J	16K x 10	2448K x 10	0x001F--0x0BFF	32 I/O
40	NY5P1K2J	16K x 10	3024K x 10	0x001F--0x0BFF	32 I/O
41	NY5A003C	12K x 10	12K x 10	0x001F--0x0BFF	7+1 I/O
42	NY5A005C	16K x 10	16K x 10	0x001F--0x0BFF	7+1 I/O
43	NY5A008C	16K x 10	24K x 10	0x001F--0x0BFF	7+1 I/O
44	NY5A011C	16K x 10	32K x 10	0x001F--0x0BFF	7+1 I/O
45	NY5A018C	16K x 10	48K x 10	0x001F--0x0BFF	7+1 I/O
46	NY5A025C	16K x 10	64K x 10	0x001F--0x0BFF	7+1 I/O
47	NY5A035C	16K x 10	88K x 10	0x001F--0x0BFF	7+1 I/O
48	NY5A045C	16K x 10	112K x 10	0x001F--0x0BFF	7+1 I/O
49	NY5A055C	16K x 10	136K x 10	0x001F--0x0BFF	7+1 I/O
50	NY5A065C	16K x 10	160K x 10	0x001F--0x0BFF	7+1 I/O
51	NY5B005C	16K x 10	16K x 10	0x001F--0x0BFF	14+1 I/O
52	NY5B008C	16K x 10	24K x 10	0x001F--0x0BFF	14+1 I/O
53	NY5B011C	16K x 10	32K x 10	0x001F--0x0BFF	14+1 I/O
54	NY5B018C	16K x 10	48K x 10	0x001F--0x0BFF	14+1 I/O
55	NY5B025C	16K x 10	64K x 10	0x001F--0x0BFF	14+1 I/O
56	NY5B035C	16K x 10	88K x 10	0x001F--0x0BFF	14+1 I/O
57	NY5B046C	16K x 10	112K x 10	0x001F--0x0BFF	14+1 I/O
58	NY5B055C	16K x 10	136K x 10	0x001F--0x0BFF	14+1 I/O
59	NY5B065C	16K x 10	160K x 10	0x001F--0x0BFF	14+1 I/O
60	NY5B075C	16K x 10	184K x 10	0x001F--0x0BFF	14+1 I/O
61	NY5B085C	16K x 10	208K x 10	0x001F--0x0BFF	14+1 I/O
62	NY5B112C	16K x 10	272K x 10	0x001F--0x0BFF	14+1 I/O
63	NY5B132C	16K x 10	320K x 10	0x001F--0x0BFF	14+1 I/O
64	NY5B158C	16K x 10	384K x 10	0x001F--0x0BFF	14+1 I/O
65	NY5B185C	16K x 10	448K x 10	0x001F--0x0BFF	14+1 I/O
66	NY5C112C	16K x 10	272K x 10	0x001F--0x0BFF	19+1 I/O
67	NY5C132C	16K x 10	320K x 10	0x001F--0x0BFF	19+1 I/O
68	NY5C158C	16K x 10	384K x 10	0x001F--0x0BFF	19+1 I/O
69	NY5C185C	16K x 10	448K x 10	0x001F--0x0BFF	19+1 I/O

No.	IC 母體	PROG ROM 空間	DATA ROM 空間	ROM 保留地址	I/O 腳個數
70	NY5C225C	16K x 10	544K x 10	0x001F--0x0BFF	19+1 I/O
71	NY5C265C	16K x 10	640K x 10	0x001F--0x0BFF	19+1 I/O
72	NY5C305C	16K x 10	736K x 10	0x001F--0x0BFF	19+1 I/O
73	NY5C345C	16K x 10	832K x 10	0x001F--0x0BFF	19+1 I/O
74	NY5C450B	16K x 10	1088K x 10	0x001F--0x0BFF	23+1 I/O
75	NY5C520B	16K x 10	1248K x 10	0x001F--0x0BFF	23+1 I/O
76	NY5C640B	16K x 10	1536K x 10	0x001F--0x0BFF	23+1 I/O
77	NY5C720B	16K x 10	1728K x 10	0x001F--0x0BFF	23+1 I/O
78	NY5Q020A	48K x 10	48K x 10	0x001F—0x07FF	8 I/O
79	NY5Q026A	64K x 10	64K x 10	0x001F—0x07FF	4 I/O
80	NY5Q040A	64K x 10	96K x 10	0x001F—0x07FF	8 I/O
81	NY5Q046A	64K x 10	112K x 10	0x001F—0x07FF	12 I/O
82	NY5Q060A	64K x 10	144K x 10	0x001F—0x07FF	16 I/O
83	NY5Q080A	64K x 10	192K x 10	0x001F—0x07FF	12 I/O
84	NY5Q092A	64K x 10	224K x 10	0x001F—0x07FF	16 I/O
85	NY5Q160A	64K x 10	384K x 10	0x001F—0x07FF	12 I/O
86	NY5Q172A	64K x 10	416K x 10	0x001F—0x07FF	16 I/O
87	NY5Q342A	64K x 10	832K x 10	0x001F—0x07FF	20 I/O
88	NY6P025A	64K x 10	64K x 10	0x001E—0x03FF	9 I/O
89	NY6P025J	64K x 10	64K x 10	0x001E—0x03FF	16 I/O
90	NY6P055J	136K x 10	136K x 10	0x001E—0x03FF	16 I/O
91	NY6P085J	208K x 10	208K x 10	0x001E—0x03FF	16 I/O
92	NY6P185J	448K x 10	448K x 10	0x001E—0x03FF	24 I/O
93	NY6P345J	832K x 10	832K x 10	0x001E—0x03FF	24 I/O
94	NY6A008A	24K x 10	24K x 10	0x001E—0x03FF	8 I/O
95	NY6A011A	32K x 10	32K x 10	0x001E—0x03FF	8 I/O
96	NY6A018A	48K x 10	48K x 10	0x001E—0x03FF	8 I/O
97	NY6A025A	64K x 10	64K x 10	0x001E—0x03FF	8 I/O
98	NY6A035A	88K x 10	88K x 10	0x001E—0x03FF	8 I/O
99	NY6A045A	112K x 10	112K x 10	0x001E—0x03FF	8 I/O
100	NY6A055A	136K x 10	136K x 10	0x001E—0x03FF	8 I/O
101	NY6A065A	160K x 10	160K x 10	0x001E—0x03FF	8 I/O
102	NY6B005A	16K x 10	16K x 10	0x001E—0x03FF	16 I/O
103	NY6B008A	24K x 10	24K x 10	0x001E—0x03FF	16 I/O
104	NY6B011A	32K x 10	32K x 10	0x001E—0x03FF	16 I/O

No.	IC 母體	PROG ROM 空間	DATA ROM 空間	ROM 保留地址	I/O 腳個數
105	NY6B018A	48K x 10	48K x 10	0x001E—0x03FF	16 I/O
106	NY6B025A	64K x 10	64K x 10	0x001E—0x03FF	16 I/O
107	NY6B035A	88K x 10	88K x 10	0x001E—0x03FF	16 I/O
108	NY6B045A	112K x 10	112K x 10	0x001E—0x03FF	16 I/O
109	NY6B055A	136K x 10	136K x 10	0x001E—0x03FF	16 I/O
110	NY6B065A	160K x 10	160K x 10	0x001E—0x03FF	16 I/O
111	NY6B075A	184K x 10	184K x 10	0x001E—0x03FF	16 I/O
112	NY6B085A	208K x 10	208K x 10	0x001E—0x03FF	16 I/O
113	NY6C112A	272K x 10	272K x 10	0x001E—0x03FF	24 I/O
114	NY6C132A	320K x 10	320K x 10	0x001E—0x03FF	24 I/O
115	NY6C158A	384K x 10	384K x 10	0x001E—0x03FF	24 I/O
116	NY6C185A	448K x 10	448K x 10	0x001E—0x03FF	24 I/O
117	NY6C225A	544K x 10	544K x 10	0x001E—0x03FF	24 I/O
118	NY6C265A	640K x 10	640K x 10	0x001E—0x03FF	24 I/O
119	NY6C305A	736K x 10	736K x 10	0x001E—0x03FF	24 I/O
120	NY6C345A	832K x 10	832K x 10	0x001E—0x03FF	24 I/O
121	NY7A004A	16K x 12	16K x 12	0x0010 – 0x03FF	8 I/O
122	NY7A007A	24K x 12	24K x 12	0x0010 – 0x03FF	8 I/O
123	NY7A010A	32K x 12	32K x 12	0x0010 – 0x03FF	8 I/O
124	NY7A016A	48K x 12	48K x 12	0x0010 – 0x03FF	8 I/O
125	NY7A021A	64K x 12	64K x 12	0x0010 – 0x03FF	8 I/O
126	NY7A032A	96K x 12	96K x 12	0x0010 – 0x03FF	8 I/O
127	NY7A043A	128K x 12	128K x 12	0x0010 – 0x03FF	8 I/O
128	NY7A054A	160K x 12	160K x 12	0x0010 – 0x03FF	8 I/O
129	NY7A065A	192K x 12	192K x 12	0x0010 – 0x03FF	8 I/O
130	NY7B007A	24K x 12	24K x 12	0x0010 – 0x03FF	16 I/O
131	NY7B010A	32K x 12	32K x 12	0x0010 – 0x03FF	16 I/O
132	NY7B016A	48K x 12	48K x 12	0x0010 – 0x03FF	16 I/O
133	NY7B021A	64K x 12	64K x 12	0x0010 – 0x03FF	16 I/O
134	NY7B032A	96K x 12	96K x 12	0x0010 – 0x03FF	16 I/O
135	NY7B043A	128K x 12	128K x 12	0x0010 – 0x03FF	16 I/O
136	NY7B054A	160K x 12	160K x 12	0x0010 – 0x03FF	16 I/O
137	NY7B065A	192K x 12	192K x 12	0x0010 – 0x03FF	16 I/O
138	NY7B076A	224K x 12	224K x 12	0x0010 – 0x03FF	16 I/O
139	NY7B087A	256K x 12	256K x 12	0x0010 – 0x03FF	16 I/O

No.	IC 母體	PROG ROM 空間	DATA ROM 空間	ROM 保留地址	I/O 腳個數
140	NY7C010A	32K x 12	32K x 12	0x0010 – 0x03FF	24 I/O
141	NY7C016A	48K x 12	48K x 12	0x0010 – 0x03FF	24 I/O
142	NY7C021A	64K x 12	64K x 12	0x0010 – 0x03FF	24 I/O
143	NY7C032A	96K x 12	96K x 12	0x0010 – 0x03FF	24 I/O
144	NY7C043A	128K x 12	128K x 12	0x0010 – 0x03FF	24 I/O
145	NY7C054A	160K x 12	160K x 12	0x0010 – 0x03FF	24 I/O
146	NY7C065A	192K x 12	192K x 12	0x0010 – 0x03FF	24 I/O
147	NY7C076A	224K x 12	224K x 12	0x0010 – 0x03FF	24 I/O
148	NY7C087A	256K x 12	256K x 12	0x0010 – 0x03FF	24 I/O
149	NY7C110A	328K x 12	328K x 12	0x0010 – 0x03FF	24 I/O
150	NY7C130A	384K x 12	384K x 12	0x0010 – 0x03FF	24 I/O
151	NY7C150A	448K x 12	448K x 12	0x0010 – 0x03FF	24 I/O
152	NY7C170A	512K x 12	512K x 12	0x0010 – 0x03FF	24 I/O
153	NY7C220A	656K x 12	656K x 12	0x0010 – 0x03FF	24 I/O
154	NY7C260A	768K x 12	768K x 12	0x0010 – 0x03FF	24 I/O
155	NY7C305A	896K x 12	896K x 12	0x0010 – 0x03FF	24 I/O
156	NY7C345A	1024K x 12	1024K x 12	0x0010 – 0x03FF	24 I/O
157	NY7C450A	1344K x 12	1344K x 12	0x0010 – 0x03FF	24 I/O
158	NY7C520A	1536K x 12	1536K x 12	0x0010 – 0x03FF	24 I/O
159	NY8A050D	512 x 14	512 x 14	-	6 I/O
160	NY8A050E	512 x 14	512 x 14	-	6 I/O
161	NY8A051A	1K x 14	1K x 14	-	6 I/O
162	NY8A051B	1K x 14	1K x 14	-	6 I/O
163	NY8A051C	1K x 14	1K x 14	-	6 I/O
164	NY8A051D	1K x 14	1K x 14	-	6 I/O
165	NY8A051E	1K x 14	1K x 14	-	6 I/O
166	NY8A051F	1K x 14	1K x 14	-	6 I/O
167	NY8A051G	1K x 14	1K x 14	-	6 I/O
168	NY8A051H	1K x 14	1K x 14	-	6 I/O
169	NY8A052E	1.5K x 14	1.5K x 14	-	14 I/O
170	NY8A053A	1K x 14	1K x 14	-	12 I/O
171	NY8A053B	1K x 14	1K x 14	-	12 I/O
172	NY8A053D	1K x 14	1K x 14	-	12 I/O
173	NY8A053E	1K x 14	1K x 14	-	12 I/O
174	NY8A054A	2K x 14	2K x 14	-	14 I/O

No.	IC 母體	PROG ROM 空間	DATA ROM 空間	ROM 保留地址	I/O 腳個數
175	NY8A054D	2K x 14	2K x 14	-	14 I/O
176	NY8A054E	2K x 14	2K x 14	-	14 I/O
177	NY8A056A	1K x 14	1K x 14	-	16 I/O
178	NY8AE51D	1K x 14	1K x 14	-	6 I/O
179	NY8AE51F	1K x 14	1K x 14	-	6 I/O
180	NY8B060D	1K x 14	1K x 14	-	6 I/O
181	NY8B060E	1K x 14	1K x 14	-	6 I/O
182	NY8B061D	1.5K x 14	1.5K x 14	-	14 I/O
183	NY8B061E	1.25K x 14	1.25K x 14	-	14 I/O
184	NY8B062A	2K x 14	2K x 14	-	14 I/O
185	NY8B062B	2K x 14	2K x 14	-	14 I/O
186	NY8B062D	2K x 14	2K x 14	-	14 I/O
187	NY8B062E	2K x 14	2K x 14	-	14 I/O
188	NY8B062F	2K x 14	2K x 14	-	14 I/O
189	NY8B071A	1K x 14	1K x 14	-	6 I/O
190	NY8B072A	2K x 14	2K x 14	-	18 I/O
191	NY8BE62D	2K x 14	2K x 14	-	14 I/O
192	NY8BM61D	2K x 14	2K x 14	-	14 I/O
193	NY8BM62D	2K x 14	2K x 14	-	14 I/O
194	NY8BM72A	2K x 14	2K x 14	-	18 I/O
195	NY8TE64A	4K x 14	4K x 14	-	18 I/O
196	NY8TM52D	2K x 14	2K x 14	-	6 I/O
197	NY8LP05A	5K x 8	5K x 8	0x0000~0x07FF	16 I/O
198	NY8LP10A	17K x 8	17K x 8	0x0000~0x07FF	16 I/O
199	NY8LP11A	17K x 8	17K x 8	0x0000~0x07FF	16 I/O
200	NY9T001A	4K x 10	4K x 10	0x001F – 0x01FF	4 I/O
201	NY9T004A	8K x 10	8K x 10	0x001F – 0x01FF	8 I/O
202	NY9T008A	12K x 10	12K x 10	0x001F – 0x01FF	16 I/O
203	NY9T016A	16K x 10	16K x 10	0x001F – 0x01FF	24 I/O
204	NY9UP01A	768 x 10	768 x 10	0x0040 – 0x004F	13 I/O
205	NY9UP02A	1280 x 10	1280 x 10	0x0020 – 0x004F	13 I/O
206	NY9UP08A	8K x 10	8K x 10	0x0020 – 0x004F	13 I/O
207	NY9U032B	32K x 10	32K x 10	0x001F – 0x03FF	16 I/O
208	NY9U064B	64K x 10	64K x 10	0x001F – 0x03FF	16 I/O

附錄B - 詞彙表

這個詞彙表定義了關於本文所用的相關術語，可作為一個共同的對照表。這個詞彙表包含了用於*NYASM*中的術語定義。

B.1 專門術語

Nyquest MCU (九齊科技微控制器)

九齊科技MCU參照NY4/NY5/NY7微控制器系列。

Application (應用)

一個由使用者所開發的軟體和硬體套件，通常被設計成九齊科技微控制器的相關應用產品。.

Assemble (組譯)

NYASM 巨集組譯器執行將來源碼翻成機械碼的動作。

Binary File

由*NYASM* 所輸出且可以個別被執行。

Build

根據需求重新編譯所有源碼檔的功能。

Control Directives (控制型虛擬指令)

控制型虛擬指令允許有條件式的組譯部分代碼。

Data Directives (資料型虛擬指令)

資料型虛擬指令是用來控制記憶體的配置和將所要參照的資料做有意義的命名。

Data RAM (資料記憶體)

MCU上的記憶體通用暫存器，它是屬於RAM的一部分。

Directives (虛擬指令)

虛擬指令是被用來控制組譯器的操作，它告訴*NYASM* 如何處理助憶符、定義資料和列表檔的格式。虛擬指令可以使程式的編碼變的更容易且依照實際的需求提供訂制化的輸出。

Expressions (運算式)

運算式是用於來源列的算術操作，運算欄可能包含了常數、符號或是任何個別常數與符號的組合。

Forward reference (向前參考)

在定義之前使用變數或函數。在*NYASM*裡Forward reference 是不允許的。例如：尚未定義Macro前不可使用Macro、尚未定義常數之前不可使用常數。例如格式，主要是當組譯器在估算表示式的時候使用。預設的格式是十六

Identifier (識別符)

一個功能或變數的名稱。

Initialized Data (資料的初始化)

在定義資料的同時亦定意一個初始值。像C語言，int myVar = 5;定義一個變數並同時定義初始值。

Listing Directives (清單虛擬指令)

清單虛擬指令主要是用來控制*NYASM* 列表檔的格式。他們所允許的規格有數字系統的基底，保留區記憶體的存取和其他列表的控制。

Listing File (列表檔)

清單檔是一個ASCII的文字檔，主要用來顯示在源碼檔中的每一個組合語言所產生的機械碼，*NYASM* 的虛擬指令，或是所遇到的巨集。

Local Label (區域標記)

使用LOCAL虛擬指令將一個標記定義成區域性標記。這些標記是在每一個巨集實例中都具有個別性。換句話說，在巨集當中將符號或標記被宣成local時，則在遇到ENDM時，會將這些符號標記從符號表中清除。

Macro (巨集)

巨集是一串組合語言指令的匯集。當在來源當中下達一個巨集的名稱時，則巨集將會被含括到組合語言程式碼當中。巨集必須在使用前先被定義；並不允許被向前引用。在MACRO虛擬指令後的所有敘述皆屬於巨集定義的一部分。在巨集內部使用標記必須將其宣告成local，如此這個巨集將可以不斷的重復被呼叫。

Macro Directives (巨集虛擬指令)

這些虛擬指令是在控制在巨集定義內部的執行和資料配置。

Mnemonics (助憶符號)

助憶符號將會直接被翻譯成機械碼。這些是被用來將微控制器的程式或資料記憶體內的資料做算術和邏輯運算。它不但可以將暫存器和記憶體的資料的做移進移出且可以改變程式執行的流程。也被稱為運算碼。

***NYASM* (九齊科技組譯器)**

九齊科技股份有限公司的MCU組譯器。

Nesting Depth (巢狀深度)

巨集可以被相互套疊到16層的深度(預設值)。最大的深度是256。

Operators (運算元)

運算元是算術上的符號，像加就寫'+'而減就寫成'-'，主要被使用構成一個明確的運算式子。每一個運算元都有一個被分配的優先次序。

PC (個人電腦)

任何IBM PC或相容的電腦。

PC Host (個人電腦主機)

正在執行Windows XP/7/8的電腦。

Precedence (優先次序)

優先次序是一個觀念。在運算式上的一些元素可以比其他的能更先獲得運算。優先次序相同的運算元在一起時會由左到右做運算。

Program Memory (程式記憶體)

模擬器或模擬器的記憶體也包含被下載到應用端的韌體。

Project (專案)

以一組源碼檔及指令建立的應用代碼。

Radix (數值格式)

Radix是設定基本的數值格式，主要是當組譯器在估算表示式的時候使用。預設的格式是十進制。你可以來改變預設的格式及取消原本設定的格式。

RAM (隨機存取記憶體)

隨機存取記憶體。

Raw Data (原始資料)

二進制格式的代碼或資料的。

Recursion (遞迴)

這是一個巨集的概念，已經被定義過，可以自己呼叫自己。當寫遞歸的巨集必須特別注意，若沒有從遞回離開的話，它很容易就鎖死在無限迴圈內。

ROM (唯讀記憶體)

唯讀記憶體。

Source Code (源碼)

源碼是由九齊科技MCU的指令和*NYASM*的虛擬指令及巨集所構成且將會被翻譯成機械代碼。這個代碼將適用於合九齊科技的開發系統，如*NYIDE*使用。

Source File (源碼檔)

一個ASCII文字檔，內容可以是九齊科技MCU的指令和*NYASM* 的虛擬指令及巨集(原始代碼)，且最後將會被翻譯成機械碼。ASCII文字檔可以由任何的ASCII文字編輯器所產生和編輯。

Stack (堆疊)

在資料記憶體中的一塊區域被用來儲存函數的參數、回傳值、局部變數和返回位址。

Symbol (符號)

符號是一個通用的機制，用來描述包括一個程式裡各式各樣的片段。這些片段包含有功能名稱、變數名稱、檔案名稱、巨集名稱…等等。

Un-initialized Data (未初始化的資料)

一個被定義的資料，但沒有被設定初始值。像是C語言中的 Int myVar。