



华芯微 MCU 汇编语法手册

V1.0



手册修正记录:

版本	修改时间	内容	对应编译器版本
V1.0		初版	V1.0.0



编译器更新记录:

版本	修改时间	修改内容
V1.0.0		初版

说明:

- 1.该手册中的示例程序均采用 HS26P00 系列指令来编写，但所有伪指令、数据定义指令、条件编译指令、错误输出指令等与机器码无关的指令适用于华芯微所有 MCU 芯片。
- 2.由于每种芯片的指令集架构不相同，因此，每种芯片的指令助记符和操作数不相同，编写不同芯片程序时需采用不同的指令格式。



目录

一.文件结构.....	6
二.标识符.....	7
三.LABEL.....	8
四.操作数.....	9
五.数字表达式.....	10
六.算数表达式.....	11
七.注释.....	12
八.伪指令.....	13
1.程序开始.....	14
2.程序结束.....	15
3.变量定义.....	16
(1)EQU (EQU_W、EQU_R 同)	16
(2)ASSIGN(=)	16
(3)TEXTEQU.....	16
(4)CATSTR.....	17
(5)SUBSTR.....	17
(6)SIZESTR	18
(7)INSTR(INISTR).....	18
(8)#define.....	19
(9)REG.....	19
(10)BIT (位定义)	19
(11)DS	19
4.段定义: .CODE、.DATA、.CONST	20
5.设置程序和数据及常量部分的地址: ORG	21
6.设置地址对齐方式: .ALIGN	22
7.程序数据定义.....	23
(1)字节数据定义:	23
(2)字数据定义:	23
(3)双字数据定义:	23
(4)数组定义:	24
(5)位算法函数:	24
(6)输出文件:	24



(7)包含文件	25
(8) MACRO.....	25
(9)REPEAT	26
(10)FOR.....	26
(11)FORC	26
(12)EXITM.....	27
(13)&.....	27
(14)%	27
(15)!	27
(16)宏参数说明及实例	28
8.条件编译.....	30
9.触发错误.....	32



一.文件结构

TITLE		;用户定义文件的名称，编译时忽略该名称
CHIP	HS26P00	;定义用户使用的芯片
INCLUDE	DATADF. H	;引入需要包含的文件
. DATA		;数据定义开始，用于说明以下内容是数据定义
. CODE		;程序开始，用于说明一下内容是程序代码
ORG	0X200	;地址重定义
DW	0X1234	;程序区的数据定义
INCLUDEBIN	BINFILE. BIN	;引用编译好的二进制文件
ENDP		;结束编译

说明：

文件中的所有字符不区分大小写。



二.标识符

1. 标识符必须以以下字符开始：A-Z ， a-z， @ ， _。
2. 除了第一个字符之外的字符可以使用以下字符：A-Z， a-z, @, _， 0-9。
3. 标识符最大长度 64 个字符。



三.LABEL

1. 标号的名称是一个正常的标识符，但标号后面必须是“:”结尾。

2. 同一行可以有多个标号名称，每个标号名称都有效。如：

Label1: label2: label3: MOV A, #55; 一行多个标号

或者

LABLE1:

LABEL2:

LABEL3:

MOV A, #55

3. 标号名称不可与其他已定义的任何名称重复。

4. 为了防止使用如此多不同的标签名称，可以使用以下说明来指示不同的标签名称：

首先，使用符号“@@:”作为暂定标签名称，并使用这个暂定标签来表示它的前标签名称和下一个标签名称。

其次，使用“@B”查找到上一个“@@:”，使用“@F”查找到下一个“@@:”。如下所示：

JMP @F ;跳转到下一个@@位置

...

@@:

...

JMP @B ;跳转到上一个@@位置



四.操作数

1. 如果操作数是内存单元，则可以定义数字以表示其地址。如果该数字是一个常量值，则该数字应以#开头。

例如： `MOV 0x80, #3` `// RAM[0x80] = #3`

2. \$符号表示当前活动的编译器程序地址：

例如：

```
JMP      $           //在当前地址循环
JMP      $+1         //跳转到下一条指令
```

此外，\$符号可以用于获得标签的高(H)、中间(M)、低(L)中的字节：

例如：

```
BOMOV X, #DATA1$H    // X = 0X12
BOMOV Y, #DATA1$M    // Y = 0X34
BOMOV Z, #DARA1$L    // Z = 0X56
MOVC
```

另外，用\$可以取标签地址的 14-17 位作为数据高位。

例如：

```
BOMOV PFLAG, #Far_Lab$J    ;== BOMOV PFLAG, #30h
JMP Far_lab
...
ORG 0XC000
Far_Lab:
...
```

3. 操作数可以是字符，字符的值等于该字符的ASCII码，字符前后必须使用单引号：

例如：

`MOV A, #'C'` `;A = 67`

4. 字符串类型操作数定义如下：

<string> | “string” | string | (string) | {string} | [string]

例如：

```
STRING1 TEXTEQU <STRING_TEST>
STRING1 TEXTEQU “STRING_TEST”
STRING1 TEXTEQU STRING_TEST
STRING1 TEXTEQU (STRING_TEST)
STRING1 TEXTEQU {STRING_TEST}
STRING1 TEXTEQU [STRING_TEST]
```



五.数字表达式

支持 4 种格式的数字:

255 ;十进制

0Xff ;十六进制

0FFH ;十六进制, 如果第一个字符是 A-F, 前面必须要加 0

11111111B ;二进制



六.算数表达式

用户可以使用+ - * / % & | ^ ~ () 等进行算术逻辑运算

例如:

$$2 + 3 - 4 = 1$$

$$2 + 3 * 4 = 14$$

$$\text{MOV } A, \#((28h+10h)\$1+30/5+(20-1*(2+3)))\$1 + 1000h\$m$$

以下是支持的各种运算符及优先级:

优先级 (由高到低)	运算符	描述
1	()	用于改变优先级或创建子表达式
	+	正
	-	负
	~	按位取反
	!	逻辑取反
2	*	乘
	/	除
	%	取模
3	+	加
	-	减
4	<<	逻辑左移, 右侧补零
	>>	逻辑右移, 左侧补零
5	>	大于
	<	小于
	>=	大于等于
	<=	小于等于
6	==	相等
	!=	不相等
7	&	按位与
8	^	按位异或
9		按位或
10	&&	逻辑与
11		逻辑或

说明:

- 1.逻辑运算的结果只有 2 个值: 0 和 1, 任何不等于 0 的值都为逻辑 1。
- 2.以上运算符可以混合使用, 但会严格按照优先级来运算。
- 3.能使用数字的地方都可以使用一个算数表达式。



七.注释

支持以下三种形式的注释：2 种单行注释和 1 种多行注释

1. 以 “;” 开始的注释。

例如：

```
MOV A, #00AH ;这里是注释
```

2. 以 “//” 开始的注释。

例如：

```
MOV A, #00AH //这里是注释
```

3. 以 /* 开始 */ 结束，表示单行注释功能或多行注释功能。

例如：

```
DW 19      /*AKAI*/  
/*Akai*/ DW 000 001 002  
/* MOV A, #00AH  
MOV A, #00BH  
MOV A, #00CH  
MOV A, #00DH  
*/
```



八.伪指令

操作符	描述
CHIP, ENDP	程序开始, 结束
EQU, =, TEXTEQU, REG, BIT, #define, CATSTR, SUBSTR, SIZESTR, INSTR, INISTR	变量定义
.CODE, .DATA .CONST	段定义
ORG, .ALIGN	段地址定义
DS	在.DATA 段中变量定义
DB, DW, DD	在.CODE 段中数据定义
INCLUDE, INCLUDEBIN, INCLUDESTD	包含别的文件
TITLE	用户定义的文件标题
MACRO, EXPAND, ENDM, REPEAT, FOR, FORC, EXITM	宏定义
.LIST, .NOLIST	控制生成的.lst 文件中是否显示内容
IF, IFE, IFB, IFNB, IFDEF, IFNDEF, IFIDN, IFDIF, IFIDNI, IFDIFI, ELSEIF, ELSEIFE, ELSEIFB, LSEIFNB, ELSEIFDEF, ELSEIFNDEF, ELSE, ENDIF	条件语句
.ERR, .ERRE, .ERRNZ, .ERRB, .ERRNB, .ERRDEF, .ERRNDEF, ERROR, ECHO	错误提示
.EXEC	执行外部程序
@BIT, @INT, @FIELD	位数据操作函数



1.程序开始

语法: **CHIP** **chip_name**

描述: 选择芯片。命令应该在任何汇编语言之前定义，并且只能定义一次。

例如:

chip HS26P10



2.程序结束

语法: **ENDP**

描述: 强制结束程序, 在此命令之后编写的程序就不会被编译。

例如: ENDP

语法: TITLE

说明: 标题后写的语句是描述语句。

例如:

TITLE This is a demo code.



3.变量定义

(1)EQU (EQU_W、EQU_R 同)

语法: **VARIABLE EQU VALUE or BIT**

说明: 定义一个无法修改的固定变量。

例如:

```

PFLAG      EQU      86H
FZ          EQU      PFLAG.0
ADM         EQU      0xB1
ADM         EQU      1
NUM1        EQU      0X20+0X3
REDMODEXFLAG EQU      00010001B
IMM_EXPR    EQU      #((1+3)*4)/2+(2+2)
DAT1        EQU      ((12+4)*2+(1+2))*2/2
DAT_b       EQU      (((12+4)*2+(1+2))*2/2) . ((1+1)*2+1-2)

```

(2)ASSIGN(=)

语法: **VARIABLE = VALUE | BIT**

说明: 可修改变量可以修改.

示例:

```

TEMP  =  0
TEMP  =  TEMP + 5 // TEMP = 5
TPIN  =  TEMP.1

```

```

ENUM MACRO LAB:VARARG
    TEMP = 0
    FOR L, <LAB>
        L      EQU      TEMP
        TEMP   =         TEMP + 1
    ENDM
ENDM

```

```

ENUM CON0, CON1, CON2 // CON0 EQU 0; CON1 EQU 1; CON2 EQU 2

```

(3)TEXTEQU

语法:

```

STRING TEXTEQU <STRING>
STRING TEXTEQU TEXT MACRO
STRING TEXTEQU % VARIABLE
STRING TEXTEQU % (ARITHMETIC)

```

描述: 定义个字符串, 程序中的该变量将被字符串替换。

例 1:

```

STRING1 TEXTEQU <STRING_TEST>
CLRA    TEXTEQU <MOV A,#0> ;程序中的 CLRA 将字符串“MOV A,#0”替换

```



例 2:

```
STRING  TEXTEQU  STRING1
CLRALU  TEXTEQU  CLRA
```

例 3:

```
TEMP    =    30
STR1    TEXTEQU  %TEMP          ; TEXTEQU <0x1E>
STR1    TEXTEQU  %0d:TEMP       ; TEXTEQU <30>
STR1    TEXTEQU  %0x:TEMP       ; TEXTEQU <1e>
STR1    TEXTEQU  %0X:TEMP       ; TEXTEQU <1E>
STR1    TEXTEQU  %03d:TEMP      ; TEXTEQU <030>
STR1    TEXTEQU  %03x:TEMP      ; TEXTEQU <01e>
STR1    TEXTEQU  %03X:TEMP      ; TEXTEQU <01E>
STR1    TEXTEQU  %03X: (7+8)*2 ; TEXTEQU <01E>
```

例 4:

```
TEMP    =    20
STR1    TEXTEQU  %(TEMP+6)      ; TEXTEQU <0x1A>
```

例 5:

```
MOVT2 TEXTEQU  MOV
MOVT2 A,#33    ;相当于 MOV A,#33
```

(4)CATSTR

语法:

```
STRING  CATSTR  <STRING>,<STRING>
STRING  CATSTR  TEXT MACRO, TEXT MACRO
STRING  CATSTR  % VARIABLE, % VARIABLE
STRING  CATSTR  % (ARITHMETIC), % (ARITHMETIC)
```

说明: 通过连接 2 个字符串组成一个新的字符串

例 1:

```
S1  TEXTEQU  <12>
S2  CATSTR   <0x>, S1    // S2 = "0x12"
```

例 2:

```
C_MOV TEXTEQU  <MOV>
C_A   TEXTEQU  < A,#1>
C_R   CATSTR   C_MOV, C_A ; C_R = "MOV A,#1"
```

例 3:

```
TEMP    =    3
TEMP1    =    2
TEMP2 CATSTR  %0d:TEMP,%0d:TEMP1 ; TEMP2 = "32"
```

类型 4:

```
TEMP    =    2
TEMP1 CATSTR  %(TEMP+6), %(TEMP+5)
```

(5)SUBSTR

语法:

```
STRING  SUBSTR  <STRING>,start,[length]
STRING  SUBSTR  TEXT MACRO,start,[length]
```



STRING SUBSTR % VARIABLE,start,[length]
STRING SUBSTR % (ARITHMETIC),start,[length]

描述：截取从 start 位置开始的 length 个字符构成一个新的字符串，如果 length 省略，则截取从 start 位置开始到 STRING 结束的字符构成哪一个新的字符串。

例 1:

```
S1 TEXTEQU <123456>
S2 SUBSTR S1, 4, 2 // s2 等于 “45”
S3 SUBSTR S1, 3 // s3 等于 “3456”
```

例 2:

```
CLRA TEXTEQU <MOV A,#25>
CLRA2 SUBSTR CLRA, 7, 2
CLRA3 SUBSTR CLRA, 7, 3
```

例 3:

```
TEMP = 30
CLRA4 SUBSTR %TEMP, 1, 4
```

例 4:

```
TEMP = 30
CLRA4 SUBSTR %(TEMP+6), 1, 2
MOV A, CLRA4
```

(6)SIZESTR

语法:

```
VALUE SIZESTR <STRING>
VALUE SIZESTR TEXT MACRO
VALUE SIZESTR % VARIABLE
VALUE SIZESTR % (ARITHMETIC)
```

描述：获取字符串的长度

例 1:

```
V1 SIZESTR <123456> // v1 = 6
```

例 2:

```
CLRA TEXTEQU <MOV A,#25>
V2 SIZESTR CLRA // v2 = 9
```

例 3:

```
TEMP = 30
V3 SIZESTR %TEMP // v3 = 4
```

例 4:

```
TEMP = 30
V4 SIZESTR %(TEMP+6) // v4 = 2
```

(7)INSTR(INISTR)

语法:

```
VALUE INSTR START, <STRING>, <SUBSTRING>
VALUE INSTR START, TEXT MACRO, <SUBSTRING>
```



VALUE	INSTR	START, % VARIABLE, <SUBSTRING>
VALUE	INSTR	START, % (ARITHMETIC), <SUBSTRING>

描述：从 start 开始位置开始的字符串中查找子字符串，其中第一个字符位于位置 1。如果找不到子字符串，则值为 0。

INSTR：查找时区分大小写字符。

INISTR：查找不区分大小写字符。

例 1:

```
S1  TEXTEQU    <12,34,56>
V1  INSTR      1, S1, <,>      // v1 = 3
V2  INSTR      V1+1, S1, <,>   // v2 = 6
```

例 2:

```
CLRA  TEXTEQU    <MOV A,#25>
V3    INSTR      1,CLRA,<,>    //V3 = 6
```

例 3:

```
TEMP1 = 30
V4     INSTR      1,%TEMP1,E    //TEMP1=0X1E,V4=4
```

例 4:

```
TEMP = 30
V5    INSTR      1,%(TEMP+6),A  //%(TEMP+6)=0X1A, V5=4
```

(8)#define

例如:

```
#define FREG1      3CH
#define F_05HZ_25_V0  FREG1.0
#define HHHHHH     MOV A,#09
#define FREG20
#define Temp11      #((28h+10h)$1+30/5+(20-1*(2+3)))$1 + 123$1
```

(9)REG

例如:

```
TM_20MS_1    REG      02H
ADM111       REG      0xB1
ADM222       REG      1
ADM333       REG      0X20+0X3
```

(10)BIT （位定义）

例如:

```
TM_20MS_B    BIT      02H.5
```

(11)DS

例如:

```
BUF0  DS  2 // equivalent to buf0 equ 2
BUF1  DS  1 // equivalent to buf1 equ 4
```



4.段定义：.CODE、.DATA、.CONST

描述：将当前地址设置为代码部分(.code)、数据部分(.Data)或常量部分(.const)。代码部分的大小取决于 ROM 大小的空间。数据部分的大小取决于内存大小的空间，在常量段中不存在大小问题。每个部分可以相互替换。系统默认该节为编程部分(.code)，其起始地址为芯片的复位地址。

例如：

.DATA

```
DAT1    EQU    01H
#deifne DEF_D  02H
DAT2    EQU    03H.2
DAT3    REG    06H
BIT1    BIT     06H.1
```

.CONST

DDEEE = 04H ;应用于=(等于赋值)时，定义的值不能再重复定义

.CODE

```
ORG     00H
JMP     INIT
ORG     08H
JMP     INT_SUB
```

INIT:

...

INT_SUB:

...

MAIN:

```
...
JMP     MAIN
```

ENDP



5.设置程序和数据及常量部分的地址: **ORG**

语法: **ORG NEW ADDRESS**

描述: 用户可以重新设置用于中断编程地址位置的编程地址, 如果在程序开始时没有设置特定的编程地址, 则系统默认为 0。

```
ORG    0DAAH
ORG_TMP    =    $           ;$表示当前地址值
ORG    ($+15) & 0X3FF0
ORG    ORG_TMP + 0X100
```



6.设置地址对齐方式: **.ALIGN**

语法: **.ALIGN VALUE**

说明: 该指令可以编译, 但不会产生任何效果。



7.程序数据定义

(1)字节数据定义:

语法:

```
[LABEL]    DB      D1      [, D2 , ...]
[LABEL]    DB      "STRING"  [, "STRING", ... ]
```

描述: 在 ROM 空间中定义数据, 数据大小必须位于 0~0xff 之间, 或者是被双引号括起来的一个字符串, 该字符串中的每个字符相当于一个字节数据, 两个字节组成一个 16 位宽的数据。第一个字节是低字节, 第二个字节是高字节。如果两个字节数据不能构成一个 16 位宽的数据, 则将高字节设置为 0, 只能在.CODE 段中使用。

例如:

```
DB 12, 34, 30, "ABCD"
```

相当于

```
DW 0X220C, 0X411E, 0X4342, 0X0044
```

说明: HS2300 系列和 HS2303-P 系列不支持该指令, 如果程序中有该指令, 运行时可能出错。

(2)字数据定义:

语法:

```
[LABEL]    DW      D1      [, D2 , ...]
[LABEL]    DW      "STRING" [, "STRING", ... ]
```

描述: 在 ROM 空间中定义 16 位宽的数据, 每个数据大小必须位于 0~0xffff 之间, 或者是由双引号括起来的一个字符串, 每两个字符变成一个 16 位宽的数据。两个字节构成一个字。第一个字节是低字节, 第二个字节是高字节。如果数据不能构成一个 16 位宽的数据, 则将高位字节设置为 0, 只能在.CODE 段中使用。

例如:

```
TEMP44      =      45
DW          0X1234, 5678H, TEMP44+3, 2*5
DW          "ABCDEFGH", 23H
HERE        DW      "HERE", "SONIX"

B0MOV X, #HERE$H
B0MOV Y, #HERE$M
B0MOV Z, #HERE$L
MOVC                               ; ACC = 'H', R = 'E'
```

说明: HS2300 系列和 HS2303-P 系列不支持该指令, 如果程序中有该指令, 运行时可能出错。

(3)双字数据定义:

语法:

```
[LABEL]    DD      D1      [, D2 , ...]
[LABEL]    DD      "STRING" [, "STRING", ... ]
```

描述: 在 ROM 中定义 32 位宽的数据, 每个数据大小必须位于 0~0xffffffff 之间, 或者是由双引号括起来的一个字符串, 四个字符组成一个 32 位宽的数据, 通常用于保存标签, 因为标签的数据通常超过 64k, 只能在.CODE 段中使用。

例如:



```
TABLE DD L1, L2, L3
L1 DW "HELLO"
L2 DW "GOOD"
L3 DW "SONIX"
TEMP2 DD "ABCDEFGH","IJKLMNOP","QRSTUVWXYZ"
```

说明：HS2300 系列和 HS2303-P 系列不支持该指令，如果程序中有该指令，运行时可能出错。

(4)数组定义：

语法：[LABEL] DS SIZE

描述：保留大小为 SIZE 的连续内存单元，只能在.DATA 段中使用。

例如：

```
MEM1 DS 1
ORG 20H
BUFFER1 DS 04 //保留 4 个字节的内存单元，起始地址为 20H
BUFFER2 DS 8 //保留 8 个字节的内存单元，起始地址为 24H
B0MOV Y, #BUFFER1$M
B0MOV Z, #BUFFER1$L
MOV A, BUFFER1[1]
MOV A, BUFFER2[1]
```

(5)位算法函数：

语法：@BIT(parameter), @INT(parameter), @FIELD(parameter)

说明：通过@BIT 可以获取位定义的索引，通过@INT 获取位定义的 RAM 地址，通过@FIELD 获取位列。

例如：

```
P_XOR EQU 12H.4
B0BSET @INT(P_XOR) -0X10.@BIT(P_XOR)
B0BSET 5.@BIT(P_XOR) ;B0BSET 5.4
MOV A, #@FIELD(P_XOR) ;MOV A, #10H
XOR @INT(P_XOR), A ;XOR 12H, A
```

(6)输出文件：

.LST：列表文件(.lst)

.CCS：烧录文件，可直接使用 HSunWirterII 下载到烧录器里面。

.ERR：编译提示文件



(7)包含文件

语法:

```
INCLUDE      FILE
INCLUDESTD   FILE
INCLUDEBIN   FILE
```

说明: 该指令后面跟着一个程序源文件或编译好的二进制文件。文件的搜索顺序如下:

- A.当前的项目路径。
- B.软件的 user_inc 目录。
- C.通过命令行传入的 include 路径。
- D.软件当前工作路径。

如果在以上路径中没有找到该文件, 则会编译报错。

例如:

```
include      "23E53.ASH"
include      MemoryAllocation.asm
includestd    HS26P10.inc
include      C:\MemoryAllocation.asm
INCLUDE      D:\Macor.H
Includebin    "code.bin"
```

(8) MACRO

语法:

```
NAME MACRO [PARA1 [, PARA2,...]] ]
...
ENDM
```

说明: 程序编写可以通过编写宏来简化。通过使用子程序和宏, 可以缩短程序的长度。利用宏程序和子程序的优点, 可以进行最优的程序设计。宏的参数限制为最多 255 个, 宏的长度不受限制, 但是宏必须由 A-Z, a-z, 0-9, @ # _.\$ 字符组成。如果参数必须包含特殊字符, 则该特殊字符必须放在 <> 中。此外, 宏也可以包括宏。宏包含多少层嵌套没有限制。

例如:

```
#define Temp11 #((28h+10h)$I+30/5+(20-1*(2+3)))$I+123$I

MOV_ MACRO ADDRESS, VALUE
MOV A, VALUE
MOV ADDRESS, A
ENDM

MOV_ 10H, Temp11
MOV_ 20H, <#6 * 8 + 1>
```



(9)REPEAT

语法:

```
REPEAT      COUNT    //重复次数
ENDM
```

描述: 重复执行 count 次。

例如:

```
RRCM_1      MACRO MEMORY,VALUE
  REPEAT     VALUE
    RRCM      MEMORY
  ENDM
ENDM

RRCM_1      0,4 //RAM[0]右移四位
```

(10)FOR

语法:

```
FOR    PARAMETER ,<PARAMETER 1, PARAMETER 2 ...>
      //the command is to be repeated
ENDM
```

描述: 使用 FOR 重复执行一段命令, 重复次数由参数的个数决定。使用逗号分隔参数序列, 并且一次用一个参数替换参数值。

例如:

```
TEMP  =  0
FOR I, <M0, M1, M2, M3, M4, M5>
  I    EQU    TEMP      ;M0\1\2\3\4\5 EQU    0\1\2\3\4\5
  TEMP =      TEMP + 1
ENDM
```

(11)FORC

```
FORC    PARAMETER,<WORD SERIAL>
      //the command is to be repeated.
ENDM
```

说明: 使用 FORC 重复执行部分命令。重复的次数由参数字符序列的长度决定。根据字序列中字符的顺序, 一次用一个字符替换参数值。

例如:

```
FORC I, <012345>
  M&I EQU I      ;M0\1\2\3\4\5 EQU    0\1\2\3\4\5
ENDM
```



(12)EXITM

说明：使用 EXITM 命令提前结束宏。

例如：

```
MEM    MACRO  value
    IF value >= 10
        ERROR  value must < 10
    EXITM
    ENDIF

    M&value EQU value
ENDM

MEM  9
```

(13)&

语法：&PARAME

描述：将相应的连续词转换为参数。

例如：

```
FORC I, <012345>
    M&I EQU I           ;M0/1/2/3/4/5/    EQU    0/1/2/3/4/5
ENDM
```

(14)%

语法：%PARAME

描述：扩展参数的值。

例如：

```
ERROR_    MACRO    E1 E2 E3
    ERROR    E1 E2 E3
ENDM

TEMP = 10
ERROR_    TEMP IS %TEMP    ;编译时输出错误信息：TEMP IS 0X0A
```

(15)!

语法：!

说明：使下一个字符排除任何特殊意义

例如：



```
ENUM MACRO CH
    FORC I, <012345>
        CH!&I EQU I
    ENDM
ENDM
ENUM M ;CH&0\1\2\3\4\5 EQU 0\1\2\3\4\5
```

(16)宏参数说明及实例

考虑以下宏，由于缺少一个参数，因此没有发生任何错误。

例 1:

```
ADD2 MACRO A1 , B1
    DW A1+B1
ENDM

ADD2 ,4 ;DW +4
```

如果省略了第二个参数，请使用预设置值重写程序。

例 2:

```
ADD2 MACRO A1:REQ , B1 :=0
    DW A1+B1
ENDM

ADD2 4 ;DW=4+0
```

如果参数是变量参数，则按以下方式重写程序。

例 3:

```
ADDN MACRO parmas:VARARG
    TEMP4 = 0
    FOR I, <parmas>
        TEMP4 = TEMP4 + I
    ENDM
    DW TEMP4
ENDM
```

ADDN 1, 2, 3, 4 ;此处的 1,2,3,4 会全部传递给 parmas, parmas = 1,2,3,4
ADDN 1, 2 ;parmas = 1,2

例 4:

```
ADDN MACRO te:=9,34,parmas:VARARG
    TEMP9 = 0
    temp1 = te
    FOR I, <parmas>
        TEMP9 = TEMP9 + I
    ENDM
    DW TEMP9
ENDM

ADDN 1, 2, 3, 4,5,6 ; TEMP9=18
```



例 5:

```
forc parame,{123456}
    L1F&parame&L EQU    parame
    forc p,"123456"
        for_equ&parame&p&p&T equ parame&parame&p&p&p
    endm
endm
```

例 6:

```
test_m macro    mp1 mp2 mp3
    test_m_equ&mp1&test equ 1
    test_m_equ&test equ 1

    forc p,<12345>
        mp1&p equ p
        mp2&p equ p&p
        mp3&p equ p&p&p&1+45/(1+1)*2

        in_m_&p macro
            in_m_equ&p equ p
            fkdf equ kf
        endm
    endm

    out_m&p&mp1 equ 1
    out_m&ttt&mp2 equ 2
endm

test_m m t,d
```



8.条件编译

语句	条件	描述
IF	expression	As it isn' t 0, it' s true
IFE	expression	As it is 0, it' s true
IFB	<argue>	As it is blank, it' s true
IFNB	<argue>	As it isn' t blank, it' s true
IFDEF	symbol	As symbol is defined, it' s true
IFNDEF	symbol	As symbol is not defined, it' s true
IFIDN	<str1> ,	As str1 == str2, it' s true
IFDIF	<str1> ,	As str1 <> str2, it' s true
IFIDNI	<str1> ,	As str1 == str2 (either capital or
IFDIFI	<str1> ,	As str1 <> str2 (either capital or
ELSEIF	expression	
ELSEIFE	expression	
ELSEIFB	<argue>	
ELSEIFNB	<argue>	
ELSEIFDEF	symbol	
ELSEIFNDEF	symbol	
ELSEIFIDN	<str1> ,	
ELSEIFIDNI	<str1> ,	
ELSEIFDIF	<str1> ,	
ELSEIFDIFI	<str1> ,	
ELSE		
ENDIF		

例 1:

为了避免参数中出现空参数，可以使用以下程序检测：

```

TEMP = 0
FOR    I, < ZERO, ONE, ,THREE>
    IFNB <I>
        I EQU TEMP
    ENDIF
TEMP = TEMP + 1

ENDM

```

例 2:

为了避免头文件重复加载，可以在头文件中使用以下程序来检测：

```

IFNDEF __TESTFILE__
    __TESTFILE__ EQU 1
    ...
ENDIF

```



例 3:

可以按照以下语法来嵌套使用各种 IF 语句，嵌套的层数没有限制：

```
IF  VAR == 1
...
ELSEIF VAR <= 10
...
    IF VAR > 5
        ...
    ELSE
        ...
    ENDIF
...
ELSEIF VAR <= 100
...
ELSE
...
ENDIF
```



9.触发错误

语句	条件	描述
.ERR		强制产生错误, 没有错误信息
.ERRNZ	expression	表达式的值不等于 0, 产生错误
.ERRE	expression	表达式的值等于 0, 产生错误
.ERRB	<argue>	参数为空, 产生错误
.ERRNB	<argue>	参数不为空, 产生错误
.ERRDEF	symbol	如果符号已经定义过, 产生错误
.ERRNDEF	symbol	如果符号没有定义, 产生错误
.ERRIDN	<str1>, <str2>	如果 str1==str2, 产生错误, 区分大小写
.ERRDIF	<str1>, <str2>	如果 str1!=str2, 产生错误, 区分大小写
.ERRIDNI	<str1>, <str2>	如果 str1==str2, 产生错误, 不区分大小写
.ERRDIFI	<str1>, <str2>	如果 str1!=str2, 产生错误, 不区分大小写
ERROR	string	强制产生错误, 错误信息为 string
ECHO	string	产生一条编译信息 string