FMD 编译器用户手册

© 2020 Fremont Micro Devices Inc.

目录

1 汇编编译器说明

1.1 变量命名规则

必须以字母开头，由字母、数字和下划线组成，不区分大小写，且变量名不能为关键字，关键字包含指令和伪指令。

1.2 数字进制表示

支持二进制、十六进制和十进制，表示方法如下：

- 二进制：

1) 以‘B’结束的 0/1 字符，例如：00010110B
2) 以‘B”开始，以‘”结尾的 0/1 字符，例如：B’00010110’

- 十进制：

1) 不含前后缀的数，就是十进制，例如：16
2) 以‘.‘开始的十进制数，例如：.10

- 十六进制：

1) 以‘H‘结束的十六进制数，例如：1FH
2) 以‘0X‘开始的十六进制数，例如：0x1F
3) 以‘H”开始，以‘”结束的十六进制数，例如：H’1F’

1.3 地址标号

标号名遵循变量命名规则，同一个名字不能重复定义。标号后面加冒号‘：’。

1.4 指令

参考芯片手册。

1.5 伪指令

1.5.1 ORG

- 格式：ORG　ADDR
- 说明：定义 PC 地址，ADDR 不能小于当前 PC，也不能大于最大 PC。

例：

```
ORG    0000H
Goto   START
ORG    0004H；中断入口
JUMP   INTtimer0
```

1.5.2 Include

- 格式：#Include<文件名> 、#include “文件名”
- 说明：<文件名>为系统目录下的文件，“文件名”为工程目录下的文件,文件类型可以是’.H’或’.HIC’的头文件，也可以是’.ASM’的源文件。头文件必须在 PC 地址为 0 之前引入，源文件可以在文件任意位置引入。

  例：

  ```
  #INCLUDE   <FMD6001.inc>
  #INCLUDE   “LED.ASM”
  ```

1.5.3 EQU

- 格式：变量名 EQURAM 地址
- 说明：变量名遵循变量命名规则，同一个变量名不允许对应多个地址，多个变量允许对应同一个地址，注意当变量没有被使用时，不检查 RAM 地址是否有效。

  例：LEDLEVEL　　EQU　　0x40

1.5.4 DB

- 格式：变量名 DB ？
- 说明：变量名遵循变量命名规则，同一个名字不能重复定义。不指定对应 RAM 地址，由编译器自动分配地址。注意当变量没有使用时不分配地址

  例：V1　DB　　？

1.5.5 DE

- 格式：DE Data0，Data1，……Datan
- 说明：Data EEPROM 数据表，数据表的地址必须在 0x4100 以后，DE 表后的数据从 ORG 定义的地址开始顺序排放，数据个数无限制，但是必须在同一行。

  例：ORG　　4110H

  ```
  DE   0x10,0x11,0x12,0x13,0x14,0x15,0x16,0x17
  DE   0x18,0x19,0x1A,0x1B,0x1C,0x1D,0x1E,0x1F
  ```

1.5.6 DBIT

- 格式: 变量名 DBIT ?
- 说明：定义位变量，变量名遵循变量命名规则。不指定对应 RAM 地址，由编译器自动分配地址。注意当变量没有使用时不分配地址。该指令定义的变量只能用于位操作指令。位操作指令包含：BCR、BSR、BTSC、BTSS，BCF、BSF、BTFSC、BTFSS

  例：V1_1 DBIT ?

1.5.7 Define

- 格式：#define 标识符字符串
- 说明：无参数的宏定义，宏定义是用宏名来表示一个字符串，以该字符串取代宏名，这只是一种简单的代换，字符串中可以含任何字符，可以是常数，也可以是表达式，预处理程序对它不作任何检查。如有错误，只能在编译已被宏展开后的源程序时发现

  例：#define Defname 1+5

```
ORG     0000H
……
LDWI    Defname
```

1.5.8 MARCO

- 格式：宏名 MARCO par1……parn

```
……；宏内容
ENDM
```

- 说明：带参数的宏，宏名和参数名遵循变量命名规则，以 ENDM 结束宏模块。

  例：

```
Delayms macro a1,a2,a3
LDWIa1
STRDELAYCNT1
LDWIa2
STRDELAYCNT2
LDWIa3
STRDELAYCNT3
CALLDELAYLOOP
Endm
ORG0000H
……
```

Delayms 0Xf0,0x49,0x30

1.5.9 Ifdef

- 格式：ifdef 条件宏

  ……　;程序段 1

  Else

  ……　;程序段 2

  endif

- 说明：如果条件宏不等于 0，执行程序段 1，否则执行程序段 2。注意该指令不能嵌套使用。

  例：

```
ifdef       defname
LJUMP       RESTART_WDT
DECRSZ DELAYCNT1,F
LJUMP       POWERDOWN_2SLOOP
else
DECRSZ DELAYCNT2,F
LJUMP       POWERDOWN_2SLOOP
DECRSZ DELAYCNT3,F
LJUMP       POWERDOWN_2SLOOP
Endif
```

2 C 编译器说明

2.1 变量类型

支持 1-4 字节的基本类型，遵循 Little-endian 标准，多字节变量的低字节放在存储空间的低地址，高字节放在高地址。表 1-1 列出了所有数据类型集它们所占空间大小。

类型	位数（bit）	数字类型	取值范围
bit	1	逻辑类型	0 或 1
char	8	有符号字符型	-128 ~ +127
unsigned char	8	无符号字符型	0 ~ 255
short	16	有符号短整型	-32768 ~ +32767
unsigned short	16	无符号短整型	0 ~ 65535
int	16	有符号整数型	-32768 ~ +32767
unsigned int	16	无符号整数型	0 ~ 65535
long	32	有符号长整型	-2147483648 ~ +2147483647
unsigned long	32	无符号长整型	0 ~ 4294967295
float	24	浮点型	
double	24	双精度浮点型	

表 1-1 数据类型

2.2 浮点数

浮点数是以 IEEE-754 标准格式实现的。此标准下定义的浮点数为 32 位长， 在单片机中要用 4 个字节存储。为了节约单片机的数据空间和程序空间，专门提供了一种长度为 24 位的截短型浮点数，它损失了浮点数的一点精度，但浮点运算的效率得以提高。一般控制系统中关心的是单片机的运行效率，在程序中定义的 float 型标准浮点数的长度固定为 24 位，双精度 double 型浮点数也是 24 位长。

2.3 变量的绝对定位

首先必须强调，在用 C 语言写程序时变量一般由编译器和连接器最后定位，在写程序之时无需知道所定义的变量具体被放在哪个地址。

变量绝对定位如下：

unsigned char Data @ 0x20; //Data 定位在地址 0x20

对绝对定位的变量不保留地址空间。上面变量 Data 的地址是 0x20，但 0x20 处完全有可能又被分配给了其它变量使用，这样就发生了地址冲突。因此针对变量的绝对定位要特别小心。

© 2020 Fremont Micro Devices Inc.

从应用经验看，在一般的程序设计中用户自定义的变量没有绝对定位的必要。

位变量也可以绝对定位。但必须遵循上面介绍的位变量编址的方式。如果一个普通变量已经被绝对定位，那么此变量中的每个数据位就可以用下面的方式实现位变量指派：

```
unsigned char Data @ 0x20; //Data 定位在地址 0x20
bit Bit0 @((unsigned)& Data *8)+0; //Bit0 对应于 Data 第 0 位
bit Bit1 @((unsigned)& Data *8)+1; //Bit0 对应于 Data 第 1 位
bit Bit2 @((unsigned)& Data *8)+2; //Bit0 对应于 Data 第 2 位
```

如果 Data 事先没有被绝对定位，那就不能用上面的位变量定位方式。

2.4 其他变量修饰关键词

2.4.1 extern 外部变量声明

引用外部变量时，变量声明为“extern”外部类型。例如程序文件 File1.c 中有如下定义：

```
unsigned char var1, var2; //定义了两个变量
```

在另外一个程序文件 File2.c 中要对上面定义的变量进行操作，则必须在程序的开头定义：

```
extern unsigned char var1, var2; //声明外部变量
```

2.4.2 volatile 易变型变量声明

volatile 定义的变量，不会被优化。

2.4.3 const 常数型变量声明

常数变量，程序运行过程中不能对其修改。除了位变量，其它所有基本类型的变量将被存放在程序空间（ROM 区）以节约数据存储空间。显然，被定义在 ROM 区的变量是不能再在程序中对其进行赋值修改的，这也是“const”的本来意义。实际上这些数据最终都将以“ret” 的指令形式存放在程序空间，但会自动编译生成相关的附加代码从程序空间读取这些常数。

例如： const unsigned char name[] = ”This is a demo”; //定义一个常量字符串

如果定义了 “const” 类型的位变量，那么这些位变量还是被放置在 RAM 中，但程序不能对其赋值修改，不能修改的位变量没有实际意义。

2.5 指针

指针的基本概念和标准 C 语法没有太多的差别。但是在单片机这一特定的架构上，指针的定义方式有几点需要注意。

2.5.1 指向 RAM 的指针

如果是汇编语言编程，实现指针寻址的方法肯定就是用 FSR 寄存器。为了生成高效的代码，在编译 C 原程序时将指向 RAM 的指针操作最终用 FSR 来实现间接寻址。

例如： unsigned char *ptr0; //定义一个指向无符号字符型的指针

2.5.2 指向 ROM 常数的指针

如果一组变量是已经被定义在 ROM 区的常数，那么指向它的指针可以这样定义：

```
const unsigned char company[]=”Hello”;          /定义 ROM 中的常数
const unsigned char *romPtr;                    //定义指向 ROM 的指针
```

程序中可以对上面的指针变量赋值和实现取数操作：

```
romPtr ＝ company;                              //指针赋初值
data = *romPtr++;                               //取指针指向的一个数，然后指针加 1
```

反过来就是错误的，因为该指针指向的是常数型变量，不能赋值。

```
*romPtr ＝ data;                                //往指针指向的地址写一个数
```

2.6 函数代码长度限制

为了保证同一个函数在同一页面空间，函数代码不能超过 2K 字节。

2.7 函数调用层次的控制

单片机的硬件堆栈深度为 8 级，考虑中断响应需占用一级堆栈，所有函数调用嵌套的最大深度不要超过 7 级。编程员必须自己控制子程序调用时的嵌套深度以符合这一限制要求。

2.8 函数类型声明

建议编写程序代码前先声明所有用到的函数类型。例如：

```
void   Fanction (void);
unsigned char delays( unsigned int timems);
```

类型声明确定了函数的入口参数和返回值类型，这样编译器在编译代码时就能保证生成正确的机器码

2.9 中断函数

实现 C 语言的中断服务程序。中断服务程序的定义方法：

void interrupt ISR(void);

其入口参数和返回参数类型必须是“void”型，亦即没有入口参数和返回参数，且中间必须有一个关键词“interrupt”。中断函数可以被放置在原程序的任意位置。因为已有关键词“interrupt”声明，在最后进行代码连接时会自动将其定位到 0x0004 中断入口处，实现中断服务响应。编译器也会自动实现中断函数的返回指令“retfie”。

2.10 标准库函数

C 提供了较完整的 C 标准库函数支持，其中包括数学运算函数和字符串操作函数。 如果需要用到数学函数，则应在程序前“#include <math.h>” 包含头文件；如果要使用字符串操作函数，就需要包含“#include <string.h>”头文件。在这些头文件中提供了函数类型的声明。通过直接查看这些头文件就可以知道提供了哪些标准库函数。

另：printf/sprintf 是一个非常大的函数，一旦使用，你的程序代码长度就会增加很多,不建议使用。在单片机系统中实现 scanf/printf 没太多意义，如果一定要实现，只要编写好特定的 getch() 和 putch()函数，就可以输入或输出格式化的数据。