品腾 8 位 MCU
C 语言程序设计指南
V1.3

目录

1. C语言编译系统PTCC简介

PTCC 是湖南品腾电子科技有限公司（品腾科技）面向 8 位 MCU 开发的 C 语言编译系统。

PTCC 编译系统主要由 C 预处理器、C 编译器、汇编器、链接器、库工具等组成。PT-IDE2 集成开发环境集成了 PTCC 编译系统，支持用户用 C 语言开发应用程序。PTCC 输入为符合标准 C 语言规范的 C 语言程序（主要遵从 ANSI C99 规范，并针对 8 位 MCU 体系结构特点扩展了少量语法）。PTCC 的输出为可执行程序（Hex 文件），包含可执行代码的 Hex 文件可以由品腾科技提供的烧录软件和烧录器硬件烧录到品腾科技的 8 位 MCU 中执行。

使用品腾科技提供的集成开发环境 PT-IDE2，用户只需专注于编写 C 程序，对 C 程序的预处理、编译、汇编、链接过程都由 PT-IDE2 自动完成。关于 PT-IDE2 的使用，请参考《PT-IDE2 用户手册》。

1.1 项目编译过程和编译输出的文件

在含有多个 C 源程序文件的项目中，PTCC 编译器首先将每个 C 源程序文件独立编译生成对应的汇编程序文件，然后调用汇编器将每个汇编程序文件编译为对应的可重定位目标文件，再由链接器将项目中所有目标文件链接成 Hex 文件，在链接过程中也会链接预先编译好的其他可重定位目标文件或库，而库程序由库工具将多个可重定位目标文件合并构成。使用 PTCC 编译系统，用户的 C 语言项目编译过程如图 1-1 所示，用户编写的 C 程序文件，通过 C 编译、汇编、链接三个步骤构造出最终的 Hex 文件。对于支持 8 位单片机 C 语言编程的 PTCC 编译系统，程序的调试信息文件由编译、汇编、链接过程创建；若要编译出可调试的程序，需要在 IDE 中重新以 DEBUG 模式编译整个工程项目。PT-IDE2 中用 C 语言开发的项目，编译时缺省采用 DEBUG 模式。

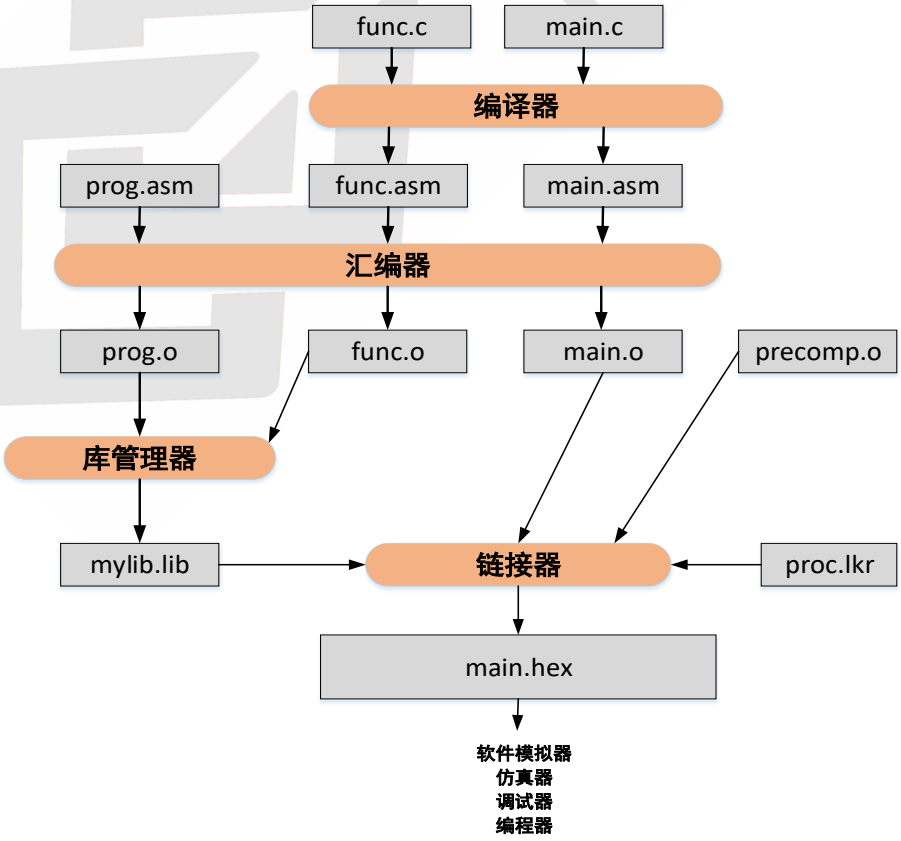

图 1-1 使用 PTCC 编译系统用户 C 语言项目的编译过程

项目编译的最后阶段是将多个.o 目标文件链接成二进制文件(.hex)，此阶段 C 编译器调用链接器，传递相应的链接选项，并提供和具体芯片对应的链接脚本文件（.lkr）。

关于汇编器和链接器的使用说明请参见相关使用手册。

在完成整个编译过程中，PTCC 编译系统会生成如下表 1-1 所示的文件。

表 1-1 PTCC 编译系统的主要输出文件

文件扩展名	文本形式	生成者	说明
.o	二进制	汇编器	汇编器对.asm 文件汇编生成.o 目标文件
.asm	文本	C 编译器	C 编译器编译.c 源程序文件生成对应的.asm 汇编程序文件
.d	文本	汇编器	汇编器汇编.asm 文件时生成，记录参与汇编的所有文件
.map	文本	链接器	链接完成后记录汇编程序符号的存储映射信息的文件
.adb	文本	C 编译器	C 语言程序的调试信息文件
.cdb	文本	链接器	链接器综合 C 程序调试信息和链接时的可重定位信息构建的 C 程序调试信息文件
.hex	二进制	链接器	链接器将多个.o 文件和.a 库函数文件链接生成可烧录到 MCU 中执行的二进制文件

1.2 PTCC的C编译器选项

用户使用 PT-IDE2 进行应用开发的时候，一般不必关心 IDE 调用 PTCC 进行项目编译的编译选项。PTCC 的 C 编译器主要编译选项如表 1-2 所示。

表 1-2 PTCC 的 C 编译选项

选项	英文说明	中文说明
-h	Display this help.	显示帮助信息。
-v	Display ptcc's version.	显示 ptcc 版本信息。
-V	Execute verbosely. Show sub commands as they are run.	显示 ptcc 编译过程中各个步骤执行的命令和命令行参数。
-d	Output list of macro definitions in effect. Use with -E.	输出有效的宏定义内容，需和-E 选项一起使用。
-D	Define macro as in –Dmacro	在调用 ptcc 的命令中定义宏。
-I	Add to the include (*.h) path, as in –Ipath	增加 include 目录。
-U	Undefine macro as in –Umacro	取消宏定义。

-W --include	Pass through options to the pre-processor (p), assembler (a) or linker. Pre-include a file during pre-processing.	在预处理时指定预先包含的文件。 -W 选项可以给预处理器、汇编器或链接器传递参数。
-S	Compile only; do not assemble or link.	仅将.c 程序编译成.asm 汇编程序，不进行汇编和链接。
-c	Compile and assemble, but do not link.	进行编译和汇编，生成.o 目标文件，不链接。
-E	Preprocess only, do not compile.	仅进行预处理，不编译。
-o	Place the output into the given path resp. file.	指定编译输出文件的名称。
--Werror	Treat the warnings as errors.	把 Warning 警告信息视作 Error 错误信息。
--debug	Enable debugging symbol output.	输出调试符号信息。
-m	Set the port to use e.g. -mpt2000.	指定处理器类（指令集）。
-p	Select port specific processor.	指定特定的处理器型号。
-l	Include the given library in the link.	包含链接器命令中指定的库。
-L	Add the next field to the library search path.	添加库文件路径。
--rom-size	Total size of ROM, Default (2000) is 8192.	指定 ROM 大小，缺省是 8192 个指令字。
--ram-size	Total size of RAM, Default (2000) is 512.	指定 RAM 大小，缺省是 512 个字节。
--ram-start	Start address of RAM.	用户 RAM 的起始地址。
--ram-end	End address of RAM.	用户 RAM 的结束地址。

2. PTCC的C语言特性

PTCC 支持的 C 语言基本符合《ISO/IEC 9899:1999》语言标准（C99 标准），针对 8 位 MCU 的结构特点引入了一些新的语言元素并对一些 C 语法进行了限制。

2.1 字符集

PTCC 的 C 语言字符集包括如下几部分：

C99 标准规定的字符集：52 个拉丁字母、10 个数字、空格、水平制表符、垂直制表符、换

页符、29 个特殊字符，如表 2-1 所示。

表 2-1 PTCC 支持的 29 个特殊字符

字符	名称	字符	名称	字符	名称
!	感叹号	+	加号	“	双引号
#	序号字符	=	等号	{	左花括号
%	百分号	~	波浪号	}	右花括号
^	折音符	[	左方括号	,	逗号
&	与号	]	右方括号	.	句号
*	星号	‘	单引号	<	小于号
（	左括号	|	竖杠	>	大于号
）	右括号	\	反斜杠	/	正斜杠
_	下划线	：	冒号	？	问号
-	减号（连字符）	；	分号		

(1) 汉字。汉字仅可以出现在注释中，否则视为非法字符。

(2) 格式符。包括退格符、回车符、空格。这些字符在非注释语句中，都被视作空格处理。

(3) 注意，PTCC 不支持@号。变量或函数声明和定义时指定地址，使用__at 关键字。

2.2 词汇

- 标识符

在程序中使用的变量名、函数名、标号等统称为标识符。除库函数的函数名由系统定义外，其余都由用户自定义。标识符只能是字母(A～Z，a～z)、数字(0～9)、下划线(_)组成的字符串，并且其第一个字符必须是字母或下划线。遵从 C 语言规范，标识符中的字符区分大小写。

- 关键字

在 C 语言中，不允许用户将关键字作为普通标识符使用。

PTCC 支持的 C 语言关键字在遵从 C 语言规范的基础上有所增减，具体如表 2-2 所示，其中以双下划线开头的关键字是新增的关键字。当前 PTCC 不支持浮点类型数据的操作，因此和标准 C 语言相比，不支持 short、long、float 和 double 关键字，但是，仍不允许用户将这些关键字用作普通标识符。

表 2-2 关键字

sizeof	return	char	enum	case
switch	int	continue	typedef	unsigned
signed	default	while	void	do
struct	extern	break	union	else
volatile	register	auto	for	if
goto	const	static	inline	
__asm	__endasm	asm	__at	__sfr
__interrupt	bit	__naked		

1、__at 关键字

__at 关键字用于在 RAM 变量声明的时候，指定变量的地址值。使用方法如下：

a) 变量声明

<声明限定符> <基类型> __at (Addr) <变量名>

例如：

unsigned int __at (0x64) YoInt；

PTCC 编译器将把无符号整形变量 YoInt 放到 RAM 空间的地址 0x64 处。如果声明限定符指定为 const，那么变量将放到 ROM 空间中。编译器会判断用户指定的 RAM 地址是否在芯片硬件设计允许的范围内，若超过芯片的 RAM 或 ROM 大小，则编译器报错；若是指定的 ROM 地址和芯片的中断入口地址相同，编译器会报错，提示用户修改指定的地址。

b) 函数声明

<函数返回值> __at (ROMAddr) <函数名>（函数参数表）

PTCC 编译器会将此函数入口地址分派到 ROM 地址 ROMaddr 处。如果用户指定的地址和其他代码段的地址有冲突，链接阶段会报错，需要用户修改程序。

2、asm、__asm、__endasm 关键字

用于在 C 程序中嵌入汇编代码，详见本文档“C 和汇编混合编程”部分。

3、bit 关键字

用于声明位变量。

4、__interrupt 关键字

用于指定函数为中断函数。

5、__sfr 关键字

用于指明变量为 sfr（特殊功能寄存器）。

6、__naked 关键字

用于修饰函数。

- 运算符

PTCC 支持所有 C 语言运算符，具体如表 2-3 所示。

表 2-3 运算符

运算符	含义	运算符	含义	运算符	含义
!	逻辑非	<=	小于等于	{	左花括号
%	取模	==	逻辑等	}	右花括号
&	位与/取地址	!=	不等于	|	位或
++	自增	>=	大于等于	~	按位取反
（	左括号	>>	右移	<<=	左移赋值
）	右括号	<<	左移	>>=	右移赋值
*	乘/间接访问	:	冒号	-=	减赋值
+	加	;	分号	|=	位或赋值
,	逗号	<	小于	&=	位与赋值
-	减	=	赋值	+=	加赋值
.	直接成员选择	>	大于	/=	除赋值
/	除	?:	条件表达式	%=	取模赋值
--	自减	[	取下标左方括号	^=	异或赋值

->	间接成员选择	]	取下标右方括号		
&&	逻辑与	^	异或		
||	逻辑或				

- 常量

在程序设计语言中，通常存在两类常量，即字面常量与符号常量。词法分析器所关注的常量是字面常量，例如，3.2、'c'等。而枚举常量、值不变的对象等符号常量并不属于词法意义上的常量。根据 C89 标准，主要有如下几类常量：整型常量、浮点型常量、字符常量、字符串常量。

单片机最常用的是整型常量，整型常量写法如下：

十进制：平常的写法，不用加前缀，后缀。

八进制：前面加数字 0。

十六进制：前面加 0x。

无符号整型常量：末尾加 U 或 u，如 unsigned char num = 10u。

注意，C 规范没有明确定义二进制常量的写法，PTCC 支持的二进制常量写法为：0b11011101 或 0B11011101。

- 注释

PTCC 支持的 C 语言中的注释符是以"/*"开头并以"*/"结尾的串。在"/*"和"*/"之间的即为注释，对于单行注释，PTCC 也支持 C99 标准中的"//"单行注释符。在程序编译时，不对注释作任何处理。注释可出现在程序中的任何位置。

2.3 声明

- 声明限定符

在 C 语言标准中，对于声明限定符有非常详尽的描述与分类，PTCC 遵从 C 语言规范，支持的声明限定符如表 2-4 所示。针对 8 位 MCU 的特点，PTCC 引入了新的声明限定符"__sfr"。

表 2-4 PTCC 支持的 C 语言声明限定符

限定符	说明
extern	可以出现在外部函数、变量的声明中，声明可以位于顶层或者位于代码块内部，它表示对象声明具有外部范围。用户可以通过声明外部函数或变量的方式，引用项目中其他 C 程序的全局函数或变量。外部对象的名称与定位将由链接器管理。如果 C 程序引用了项目内未定义的外部对象名称，那么链接器将报告错误。
const	PTCC 的 const 限定符与标准 C 语言有差异。在标准 C 中，const 只是限定左值对象是只读的。而在 PTCC 中，const 是用于指定对象的存储类别，const 限定的对象必须是分配在 ROM 区域中的。
volatile	volatile 限定的变量表示该变量可能被某些编译器未知的因素改变。编译器对访问该变量的代码不再进行优化。在 PTCC 应用中，如果需要准确跟踪某一局部变量的值，可以使用 volatile 限定该局部变量，使之不参与优化。
static	可以出现在函数或变量的声明中。不过，PTCC 并不支持全局静态函数定义，因此，对于函数定义而言，static 指定符将被忽略。在数据声明中，它用于修饰一些必须分配在静态区域内的局部或全局对象。

register	兼容 C 语言规范，对于 PTCC 而言无实际作用。
typedef	此限定符用于为一种数据类型定义新的类型名称。
__sfr	声明一个 8 位的变量，且此变量映射到一个 8 位的特殊功能寄存器。通常和__at 关键字一起使用。 例如： __at(0x66) __sfr P0； //声明了一个位于地址 0x66 处的特殊功能寄存器 P0。

2.4 数据类型

PTCC 的 C 编译器当前针对 8 位 MCU，支持的数据类型如表 2-5 所示。

表 2-5 类型总览

类型	类型分类	说明	备注
bit	位类型	在用户 RAM 空间声明位类型变量。	
enum （unsigned）char signed char unsigned int （signed）int	整形	仅支持 char 和 int 类型，且 int 类型是 16 位（两个字节），未指定符号属性的 char 缺省为 unsigned char 类型；而未指定符号属性的 int 类型缺省为 signed。	
T*	指针类型		
T[...]	数组类型		聚合类型
struct {...}	结构类型		
union {...}	联合类型		
void	void 类型		

- 算术类型

对于基本算术类型，PTCC 仅支持单字节数据类型和双字节数据类型；单字节数据类型是 char 类型，若未指定是 signed（有符号）还是 unsigned（无符号），则默认为 unsigned（无符号）；双字节数据类型为 int，若未指定是 signed（有符号）还是 unsigned（无符号），则默认为 signed（有符号）。当基本数据类型声明存在溢出的情况，如：signed char = 0x86，则编译器会报出“warning 158: Very dangerous overflow in implicit constant conversion”警告。类型表示的数值范围如表 2-6 所示。

表 2-6 算术类型

类型	长度（bit）	取值范围	说明
enum	8	-128~127	枚举
unsigned char	8	0~255	无符号字符
signed char	8	-128~127	有符号字符

unsigned int	16	0~65536	无符号整数
signed int	16	-32768~32767	有符号整数

根据 C 语言规范，默认情况下“signed”限定符可以省略，默认是有符号的。**但是，PTCC 目前支持的 8 位 MCU，char 类型默认是无符号的。**

- 位（bit）类型

PTCC 在标准 C 语法基础上扩展“bit”关键字，支持位类型，位类型变量只有一个数据位，其值为 0 或 1。但是由于 PT 处理器硬件并没有独立编址的位区域，声明的位类型变量必然只能是存储在某个字节的某一位中。位类型变量有以下两种声明方法：

1）声明时不指定位地址，如：bit　bitvar;

这种声明的位变量，位的值实际上存储在字节变量最低位，这种声明方法每个位变量占用了一个字节。

2）声明时用__at 关键字指定位地址，如: bit __at(0x481)　bitvar;

位地址的编址规则是：（字节地址 X 8 + 字节内位偏移）。PTCC 支持对整个数据存储空间进行位编址，字节地址为 0x80 的存储单元的第 0 位的位地址 0x400，第 1 位的位地址为 0x401，依次类推，每个字节 8 个位地址。

如果声明：bit　__at(0x482) flag;

那么位变量 flag 实际存储空间位于:

字节地址＝0x482/8 = 0x90

位偏移＝0x482%8 = 2

即 flag 位变量位于地址为 0x90 的字节的第 2 位。在程序调试时如果要观察 flag 的变化，必须观察地址为 0x90 的字节。

位类型不是 C 规范中明确定义的语法，关于位变量赋值和运算规则，不同的 C 编译系统有不同的定义。

PTCC 编译器对 C 编程中使用位变量，规定如下规则：

1、位类型变量必须是全局变量或静态变量。

2、位类型变量的赋值规则：

bit　b = a;

如果 a 的最低位是 0，则 b 为 0; 如果 a 的最低位是 1，则 b 为 1。

3、编译器不支持的位变量运算：

移位：右移（>>），左移（<<）;

自增和自减：自增（++），自减（--）;

只有 1 位有效值的位变量，其移位运算编译器会报错。对于位变量，移位后的值规定为 0，需要用户改正程序将位变量的移位修改为“将 0 赋值给位变量”。

位变量的++、--运算编译器也会报错，需要用户修改程序；编译器这样处理也限制了用户将位变量用作计数循环的循环控制变量。

4、位变量可以作为函数返回值，编译器将通过处理器状态寄存器中的 C 位返回该位变量值。

5、位变量的加减运算、乘法运算和通常的加减运算、乘法运算的语义相同。

6、位变量的除法、取模定值规则：

除法：两个位变量都为 1 时结果为 1，有一个位变量为 0，则结果为 0;

取模：结果为 0。

- 指针类型

PTCC 支持的指针类型分为两类：ROM 指针和 RAM 指针。ROM 指针指向 ROM 区域，RAM 指针指向 RAM 区域，指针长度都为 16 位（两个字节），但不支持两类指针之间类型转换。

ROM 指针的声明形式：

const <基类型>* <指针变量名>;

RAM 指针的声明形式：

<基类型>* <指针变量名>;

- 数组类型

PTCC 支持的数组可分为两类：存储在 ROM 中的数组（简称“ROM 数组”）、存储在 RAM 中的数组（简称“RAM 数组”）。与 ANSI C 相同，数组名也可用于表示该数组的首地址。因此，在 PTCC 中，试图令一个 RAM 指针指向一个 ROM 数组的元素是无效的，但编译器并不会对此进行静态检查。

ANSI C 的规定数组元素必须是连续存放的，在 PTCC 中，数组的长度通常会受到芯片 ROM 和 RAM 可用空间大小的限制。在实际应用中，必须特别注意数组的长度定义。

- 结构类型与联合类型

PTCC 支持的结构类型（联合类型）基本与标准 C 兼容，相关的语法不作详细介绍。但是，由于 PTCC 主要是针对嵌入式系统开发应用的，因此，有些语言特性必须兼顾芯片架构。主要包括以下几点：

1、不能使用声明限定符修饰结构或联合的成员。

2、结构类型变量不能作为函数参数传递，函数返回值也不能是结构变量。

例如，PTCC 不支持下述遵从 C89 规范的 C 结构体声明和使用方法：

```
struct s { … };
struct s foo (struct s par)
{
    struct s ret;
        …
    return rets;
}
```

不支持函数、void 类型的成员；

3、结构/联合的数组初始化时，必须为每个结构/联合元素加{}。

例如：

```
struct s { char x } a[] = {{1},{2}};
```

- void 类型

PTCC 关于 void 类型的使用与 ANSI C 一致，大致可以分为几下几类：

1、作为函数的返回类型，表示这个函数不返回任何值；

2、形成 void *类型，它是一种“通用的”数据指针；

3、在函数声明中代替形参列表，表示这个函数不接受任何参数。

- 声明一致性

根据 C 语言标准，一个符号多次重复声明是合法的，但用户必须保证所有声明的类型是兼容的，否则编译或链接阶段会报错。

2.5 表达式

- 运算符、表达式、语句

PTCC 支持 C 语言运算符和标准 C 完全兼容，如表 2-7 所示。

表 2-7 PTCC 支持 C 语言运算符优先级

操作符标记	说明	类型	优先级	结合性
变量名、字面值	简单标记	基本	16	无
a[k]	下标	后缀	16	从左到右
f(...)	函数调用	后缀	16	从左到右
.	直接选择	后缀	16	从左到右
->	间接选择	后缀	16	从左到右
++ --	自增、自减	前缀	16	从左到右
++ --	自增、自减	后缀	15	从右到左
sizeof	长度	单目	5	从右到左
~	位非	单目	15	从右到左
!	逻辑非	单目	15	从右到左
+ -	算术正负号	单目	15	从右到左
&	取地址	单目	15	从右到左
*	间接访问	单目	15	从右到左
（类型名）	类型转换	单目	14	从右到左
* / %	乘、除、取模	双目	13	从左到右
+ -	加、减	双目	12	从左到右
<< >>	左移、右移	双目	11	从左到右
< > <= >=	关系	双目	10	从左到右
== !=	相等/不相等	双目	9	从左到右
&	位与	双目	8	从左到右
^	位异或	双目	7	从左到右
|	位或	双目	6	从左到右
&&	逻辑与	双目	5	从左到右
||	逻辑或	双目	4	从左到右
?:	条件	三目	3	从右到左
= += -= *= /= <<= >>= &= ^= |=	赋值	双目	2	从右到左
,	逗号	双目	1	从右到左

PTCC 支持的表达式如算术表达式、赋值表达式、逗号表达式等，也和标准 C 语言完全一致。

类型转换的规则也完全遵从标准 C 的约定。

PTCC 支持的 C 语言语句和标准 C 语言完全兼容，这里不做说明。

需要进一步了解 C 语言表达式和语句的用户，请参阅介绍标准 C 语言的资料。

- 函数声明和调用

PTCC 的函数声明和调用遵从 C 规范中的定义，但是 PTCC 对于函数声明和调用增加了一些限制。

在函数声明时，注意以下几点：

1、函数必须先声明再定义和使用。

2、不支持可变参数函数的声明和使用。

3、不支持 K&R 风格的函数声明，如：

```
int foo ( i, j);
int i, j;
{
    ...
}
```

在调用函数时，注意以下几点：

1、 实参表达式的类型隐式转换到形参类型是允许的，否则编译器将报错。

2、 实参表达式的个数必须与形参个数匹配。

3、 实参表达式的求值顺序。C 语言标准并没有严格定义实参表达式的求值顺序与传参顺序，通常由编译器定义。PTCC 采用的是自左向右的求值顺序，因此，在使用有副作用的实参表达式时应特别注意。

2.6 动态存储分配

PTCC 不支持程序运行时动态分配内存。

3. 嵌入式应用相关特性

3.1 用命名位域实现1位或多位操作

在单片机领域，位操作很常用。C 语言规范支持的最小数据类型为字节（8 位）。品腾 8 位 MCU 处理器提供了面向 8 位字节数据的位操作汇编指令形如“OP　R,b”，其中 R 为 8 位数据的 RAM 地址，b 为字节中的位（0<=b<=7）。但是位域不能跨字节存储，一个位域的宽度必须在 1~8 之间；而如果位域的宽度是 8 则建议不要使用位域，直接声明为 char 类型。

为了支持位操作，PTCC 使用命名位域来支持 C 编程中对位的操作，并能充分利用 CPU 提供的位操作指令。

例如：

1、首先定义位域类型如下：

```
typedef struct
{
   unsigned _T0IE                  : 1;
   unsigned _T1IE                  : 1;
   unsigned _T2IE                  : 1;
   unsigned _INT0IE                : 1;
   unsigned _INT1IE                : 1;
   unsigned _INT2IE                : 1;
   unsigned _GIEH                  : 1;
   unsigned _GIE                   : 1;
} __IE0bits_t;
```

2、然后可以定义位域类型的变量，其他源程序文件若使用此位域变量，就用 extern 声明。可以为位域变量的每个位定义一个名字，方便程序中引用，如下：

```
__at(addr_IE0) volatile   __IE0bits_t   IE0bits;
#define    T0IE              IE0bits._T0IE
#define    T1IE              IE0bits._T1IE
#define    T2IE              IE0bits._T2IE
#define    INT0IE            IE0bits._INT0IE
#define    INT1IE            IE0bits._INT1IE
#define    INT2IE            IE0bits._INT2IE
#define    GIEH               IE0bits._GIEH
#define    GIE               IE0bits._GIE
#define    GIEL              IE0bits._GIE
```

3、可以在同一地址处定义一个 unsigned char 变量，如：

```
__at(addr_IE0) unsigned char _byteIE;
```

可以用对字节变量_byteIE 的赋值来一次修改这个字节中的多个位。

通过采用上述定义模式的 C 代码，用户可以直接对命名位变量（如 T0IE、T1IE 等）进行操作。这种定义方法不止可以操作长度为 1 的位变量，也可以操作长度为若干位（通常小于 1 个字节的位宽）的变量。

SFR 中的位和位域，PTCC 已经都按此编程方法定义到了头文件中，用户可以直接使用。如果用户在应用编程中仍需要使用 1 位或多位的操作，要求用户也参照 PTCC 头文件中的命名位域定义方法来编程，并尽可能将多个位域定义到同一个字节中以充分利用 RAM 空间。

例如：

```
typedef struct
{
   unsigned P0_1                : 1;
   unsigned P0_2                : 1;
   unsigned P0_up_value         : 3;
   unsigned P0_low_value        : 3;
} __P0inout_t;
__at(addr_P0)   volatile   __P0inout_t   P0inout;
#define    P0EN              P0inout.P0_1
#define    P0PR              P0inout.P0_2
#define    P0INV             P0inout.P0_up_value
#define    P0OUTV            P0inout.P0_low_value
```

这样定义以后，既可以对单个位如 P0EN、P0PR 进行读取或赋值等操作，又可以对多个位如 P0INV、P0OUTV 进行读取或赋值等操作；并且可以把程序中使用位域的变量尽可能集中紧密地排布到较小的 RAM 空间内。

例如：P0INV = 0x7; P0EN = 1; PTCC 编译系统会对单个 1 位的操作生成位操作指令；对多位的操作用单字节操作，可以保证汇编代码生成数量最少。

3.2 bit变量使用的注意事项

- bit 变量只能是全局变量或静态变量
- 位类型变量的赋值规则：

如果最低位是 0，则转换成位变量 0；如果最低位是 1，则转换成位变量 1。

- 位变量的移位运算编译器会报错：

只有一位有效值的位变量，其移位操作编译器会报错，需要用户改正程序。位变量移位后，无论原来位变量的值是什么，移位后都是 0。

- 声明 bit 变量时指定位地址，可以将多个位变量存放在一个字节中，节省存储空间

例如：

```
unsigned char __at(0x84)   flagchar;
bit __at(0x420) flag0;
bit __at(0x421) flag1;
bit __at(0x422) flag2;
bit __at(0x423) flag3;
bit __at(0x424) flag4;
bit __at(0x425) flag5;
bit __at(0x426) flag6;
bit __at(0x427) flag7;
```

这种指定地址的声明，flag0 到 flag7 存放在字节地址为 0x84 的字节变量 flagchar 的 0-7 位中。这样定义，既可以单独操作各个 bit 变量，也可以操作字节变量 flagchar 来同时操作 8 个位变量。

定义在一个.C 源文件中指定了位地址的位变量，另外一个.C 源文件可以通过声明为外部变量来访问，例如：extern bit __at(0x421) flag1；注意，必须和位变量定义时一样，指定位地址。

- 如果用户应用中需要用一组位变量，建议用 3.1 节中定义位域的方法实现，更有利于用户将位变量编排在字节中，节省 RAM 空间的使用。
- PTCC 编译器仅支持在用户 RAM 空间声明位变量

用户 RAM 空间起始地址需查阅具体芯片的用户手册。如果用户定义的位变量的字节地址不在用户 RAM 空间，编译器会报错提示用户。

3.3 函数定义与调用

PTCC 支持的 C 语言总体上遵从 C 语言规范，建议用户先声明再使用，否则编译器会报警告信息。此外，还有下面一些限制。

- 函数代码长度的限制

品腾 8 位 MCU 的 ROM 页面最大为 8K 指令字，那么函数的最大代码长度也必须在此范围内；针对具体芯片的 ROM 空间限制请参阅相应芯片的规格书。使用集成开发环境 PT-IDE2 面向具体芯片进行 C 程序开发的过程中，IDE 调用编译系统的时候，如果最终生成的可执行程序占用的 ROM 空间超过对应芯片的 ROM 空间大小，链接器会报出相应的错误信息呈现在 IDE 界面中。

- 暂不支持的语法

PTCC 编译器不支持如下函数相关的语法：

(1) 函数指针。
(2) 可变参数函数。
(3) 结构体、联合作为函数的返回值或者形参。
(4) C 规范定义的标准库函数。当前，由于品腾 8 位 MCU 的 ROM 空间和 RAM 空间的限制，PTCC 编译系统不支持 C 语言的标准库函数。

- 函数调用层次的限制

在处理函数调用时，品腾 8 位 MCU 依赖于硬件堆栈存储函数返回地址。因此，函数调用的深度就受限于硬件堆栈的容量。品腾 8 位 MCU 的硬件堆栈为 8 级，因此，程序员必须严格控制函数嵌套调用的深度。在程序动态运行过程的任意时刻，函数调用的深度都不能突破硬件堆栈的级数；否则会发生程序执行错误，这类错误难于的调试，并且目前编译器编译过程中无法给出错误报告。

同样，由于受限于硬件堆栈级数，PTCC 也不支持递归函数调用。

- 用 inline 关键字将函数内联

程序中的函数调用在函数调用、参数传递、函数返回方面会产生开销。PTCC 编译器支持使用“inline”关键字指示编译器在编译阶段将函数内联。函数内联在减少函数调用开销的同时，也可以扩大编译阶段进行编译优化的程序范围。inline 关键字的用法示例如下：

```
inline char MyFunc(char a, char b)
{
    return (a+b);
}
char main(void)
{
    char P0 = 0x64;
    P0 = MyFunc(1,9) + P0;
    return P0;
}
```

将被多次调用的函数调试正确后，用 inline 关键字指示编译器在编译时进行内联，可以大幅度提升程序效率。需要注意的是，函数被内联以后，函数将不能被单独调试，即调试阶段将不能在被内联的函数内设置断点、查看变量值。

特别注意：

被 inline 的函数必须和调用它的函数写在同一个文件中，并且定义在调用它的函数的前面。

- 中断函数

PTCC 对函数定义的语法形式进行了扩展，增加了“__interrupt”关键字。

中断函数定义的一般形式：

```
void   中断函数名(void)   __interrupt (number)
{
   [语句列表]
}
```

在使用中断函数时，值得注意以下几点：

1、中断函数形参列表和返回类型都必须为 void。中断函数是由 MCU 在产生中断时自动调用的，不允许被其他用户程序调用。

2、__interrupt 关键字必须出现在中断函数名与函数体之间。

3、__interrupt 关键字后的括号和其中的数字（number）表示此中断函数服务的中断号。

品腾 8 位 MCU：

中断号 1 表示低优先级中断，中断入口地址为 0x0008；

中断号 2 表示高优先级中断，中断入口地址为 0x0018。

4、如果中断处理函数改变了可被其他函数访问的变量，那么这些变量必须声明为 volatile。

5、可以用__at(addr)为中断处理函数指定地址，但是此地址不能和芯片固定的中断入口地址（0x0008 和 0x0018）重叠。编译器会根据“__interrupt（中断号）”语法，生成中断向量表，即在中断入口处生成跳转到具体中断处理函数的跳转指令。

6、可以用__naked 关键字修饰中断处理函数。

7、如果程序要求访问某个多字节变量是原子操作，那么在访问期间一定要关闭中断。这一点要特别注意，因为对于 8 位 CPU，访问 16 位或更长位数的变量都不是原子的。由于中断发生以及变量访问的非原子性带来的程序错误，非常难以重现和定位。

8、要求用户保存和恢复除 PCL、ACC、STATUS 之外的 MPH0、MPL0、EADRH、EADRL、EDATH、GPCR0~GPCR3。这些都可能在中断程序中被使用（被污染），用户可以根据自己中断程序中使用上述寄存器的情况来决定保存和恢复哪些寄存器。

对中断函数中内核寄存器保存和恢复的要求：

1）GPCR0、GPCR1、GPCR2、GPCR3 必须保存和恢复。这 4 个 8 位寄存器用于编译器代码生成或用于函数返回值。即使用户没有显示地调用带返回值的函数，程序编译过程中有可能编译出调用库函数的代码（该库函数有返回值）。

2）中断函数中如果以间接寻址的方式对 RAM/ROM 进行了访问，MPH0、MPL0、EADRH、EADRL、EDATH 必须保存和恢复。

3）不同优先级的中断现场，需要保存在 RAM 中不同的位置。

9、必须保证含主函数 main 的 C 源代码文件知道有中断处理函数存在。

方法：

1）把 main 主函数和中断处理函数编写在一个.c 源程序文件中；

2）如果中断函数和 main 主函数不在一个.c 源文件中，必须在 main 函数所在的.c 源文件中包含中断函数的完整声明。

- 用__naked 关键字修饰函数，可以减少函数编译后占用的 ROM 空间

定义函数的时候，可以用__naked 关键字修饰函数，编译器将不会为此函数生成入口时处理器状态保存的指令也不生成在函数结束时恢复处理器状态的指令，甚至也不生成函数调用返回指令，必须由程序编写者完全负责这些操作。

__naked 关键字特别适用于函数内部采用嵌入汇编指令的函数，可以保证这类函数生成的代码和汇编代码一致。

用__naked 关键字也可以修饰中断函数。当中断函数完全由嵌入汇编编写的时候，特别适合使用__naked 关键字，这样用户可以掌握整个中断处理函数使用了哪些汇编指令，可以准确评估中断处理函数完成一次中断处理所用的时间。例如：

```
void nakedInterrupt(void) __interrupt (2) __naked
{
    __asm
    inc   ptc_counter ;
    reti ;
    __endasm;
}
```

注意，用__naked 关键字修饰的普通函数，必须在函数最后加入函数返回汇编指令：asm(“ret”)；否则，函数不会返回就会造成程序执行流混乱。

- 函数定位到绝对地址

通常，函数代码段的存储分配是由编译系统中的链接器在链接多个目标文件时完成。不过，鉴于一些嵌入式应用的特殊需求，PTCC 支持用户为函数代码段指定绝对地址。在存储分配阶段，编译器将优先分配被绝对定位的函数代码段。编译器可以保证非绝对定位的函数代码段不与绝对定位的函数代码段重叠或部分重叠，但是，编译器并不保证绝对定位的函数代码段之间是不存在重叠或部分重叠的。也就是说，一个项目中存在多个绝对定位的函数时，编译器并不关注它们的代码段是否存在重叠或覆盖。从语言实现的角度来说，绝对定位的函数（或者变量）的存储分配是完全由用户指派的，其责任也将由用户承担，由于重叠和覆盖可能造成错误，用户编程要特别注意。

函数定位到绝对地址的一般形式：

```
<返回类型> __at（<绝对地址>）<函数名> ( [形参列表] )
{
[语句列表]
}
```

值得注意的是，这里的“绝对地址”必须是一个整型常量表达式，即该表达式的值是编译时可计算的。编译器将根据该表达式的值，将当前函数的代码段分配到相应的 ROM 存储区中。绝对地址说明仅在函数定义中有效，在函数声明中的绝对地址说明将被忽略。

3.4 C和汇编混合编程

- C 程序中内嵌汇编

PTCC 的 C 编译器支持在 C 程序中嵌入汇编程序，提供两种嵌入汇编的方式。

1、__asm 和__endasm

例如：

```
__asm
    ;汇编程序注释
    label:
      nop
      addk   0x47
__endasm;
```

在__asm 和__endasm 之间，每行编写 1 条汇编指令，可以添加汇编程序要求注释。__endasm 之后一定要加分号（;）。

2、asm（“汇编指令”）

例如：

asm（“;汇编程序注释\n label: \n\t nop \n\t addk 0x47”）;

此种方式下，括号中的引号里，可以写多行汇编指令，每条指令后加入换行符、制表符即可。

在嵌入式系统编程中，C 编程中嵌入汇编非常普遍，在关键程序段嵌入汇编，可能会提高程序执行效率。

3、嵌入汇编中引用 C 程序中的全局变量

在嵌入汇编中引用 C 程序的全局变量时，必须在变量名前加上“ptc_”前缀。在编译完成后，整个编译、链接过程中会自动将其替换为所引用变量的实际地址。内嵌汇编只可以引用 C 程序中定义的全局变量，定义在处理器头文件中的 SFR（特殊功能寄存器）可以在嵌入汇编中直接引用，不需要加前缀。

例如：

```
char i;
char j;
main()
{
        __asm
            MOVK    0x01
            MOV     A    ptc_i        ;访问全局变量i
            MOVK     0x02
            MOV     A     ptc_j       ;访问全局变量j
        __endasm;
}
```

- 在 C 函数中嵌入汇编代码的限制：

1、PTCC 对嵌入汇编程序段中标号的限制：

在嵌入汇编程序段中使用的标号，不能和 C 编译器编译 C 程序生成的汇编文件中的标号相同，

C 编译器编译 C 程序生成的汇编程序中的标号为“t_nnnnn”，其中 nnnnn 为数字。另外，用户在 C 程序中编写的嵌入汇编使用的标号也不能以数字开头。

例如：

```
__asm
        MOVK    10
  A_0061:
        JMP         A_0061
__endasm;
```

其中的 A_0061 是由用户定义的仅仅用于嵌入汇编段的标号。

特别注意：

1）在__endasm 后面一定要有“;”号。

2）嵌入的汇编程序段中不能引用 C 程序中的标号。C 程序段也不可以引用汇编程序段中定义的标号。

2、嵌入汇编程序段中不能调用其他函数

在嵌入汇编程序段中，不支持对外部函数的调用，即嵌入的汇编指令中不能有“CALL”指令。

3、嵌入汇编程序段中不能使用代码段或数据段操作伪指令

嵌入的汇编程序段不要使用汇编编程伪指令，如 ORG、data、da、db 等等。这些伪指令的使用，编译器和汇编器不会报错，但是会造成程序逻辑混乱、难以调试，甚至出现不可预知的行为。

品腾 8 位 MCU 的汇编器支持了如下伪指令：

access_ovr	__badram	__badrom	bankisel	cblock
code	code_pack	__config	config	constant
da	data	db	de	#define
dt	dtm	dw	else	end
endc	endif	endm	endw	equ
error	errorlevel	exitm	expand	extern
fill	global	idata	idata_acs	__idlocs
if	ifdef	ifndef	#include	list
local	macro	__maxram	__maxrom	messg
noexpand	nolist	org	page	pagesel
pageselw	processor	radix	res	set
space	subtitle	title	udata	udata_acs
udata_ovr	udata_shr	#undefine	variable	while

特别注意，嵌入 C 程序中的汇编程序段中：

1）不能用上述表格中的汇编伪指令作为嵌入汇编程序段中的变量名和标号。

2）CPU 的汇编指令助记符也不能作为嵌入汇编程序段中的变量名和标号。

3）嵌入汇编程序段中不能含有汇编编程中使用的汇编语言宏定义。

4）汇编器区分字符的大小写，汇编器可以支持全大写和全小写的汇编指令，汇编语句中引用的寄存器和标志位必须用全大写字母，和 PTCC 的 C 编译器后端与汇编器的“XXX.inc”接口文件中用大写字母定义的寄存器和标志位保持一致。

总之，嵌入汇编程序段中的汇编指令，建议只使用具体的机器指令的汇编指令形式，要特别注意不能将汇编编程中的程序段直接嵌入 C 语言程序中使用，会造成一些编译、汇编错误，也可能虽然编译正确但目标程序运行时结果不可预料。

3.5 与可重定位目标文件的链接

将汇编文件用汇编器编译成可重定位目标文件（*.o 文件），只需将该可重定位目标文件加入项目参与编译即可。由于可重定位目标文件是二进制形式，在一些对源代码安全要求较高的项目中，这种编译方式可以非常有效的。通常，用户开发的稳定的代码，可以编译为多个.o 文件，然后用库工具（pt2000lib.exe）压缩为一个库程序文件，可以供多个项目使用，项目 C 代码可以直接调用库中的函数。

3.6 硬件支持除法指令时使用除法指令要及时查看溢出标志位

支持乘除法指令的 8 位 MCU，为了防止由于除以 0 造成溢出，MCU 在处理器状态标志寄存器中定义了 OV 位，当除法运算中除数为 0 则 OV 位为 1，反之则 OV 位为 0。

编程使用除法指令时，无论是 C 程序中编写除法表达式还是汇编程序中直接编写汇编除法指令，都要特别注意除数是否为 0。如果用户能确定在调用除法运算的时候除数肯定不是 0，那么不必做特殊处理。但是，如果用户不能确定调用除法时除数是否为 0，用户编程要在调用除法后立即判断 OV 位的值来决定后续的处理，如果 OV 为 1 进入死循环状态，否则进行用户程序的正常逻辑。这样可以防止出现由于除法结果溢出造成程序错误难以定位的情况。

3.7 其他编程注意事项

(1) const、volatile 等关键字不能用于修饰结构（联合）的成员、形参、返回类型；
(2) 不支持 ROM 指针与 RAM 指针转换；
(3) 不支持动态内存分配，如 malloc、free 函数等。

4. 系统库函数

4.1 整数库libptcc.a

PTCC 编译系统针对 8 位整数 MCU，提供了如下整数函数，这些函数的目标码在 libptcc.a 库中。这些函数实现了整数的乘法、除法、取模运算。

```
signed int _mulschar (signed char x, signed char y);
signed int _muluschar (unsigned char x, unsigned char y);
unsigned int _mulsuchar (signed char x, signed char y);
unsigned int _muluchar (unsigned char x, unsigned char y);
int _mulint (int a, int b);
signed int _divschar (signed char x, signed char y);
signed int _divuschar (unsigned char x, unsigned char y);
unsigned int _divsuchar (signed char x, signed char y);
int _divsint (int x, int y);
unsigned int _divuchar (unsigned char x, unsigned char y);
unsigned int _divuint (unsigned int x, unsigned int y);
signed int _modschar (signed char x, signed char y);
signed int _moduschar (unsigned char x, unsigned char y);
unsigned int _modsuchar (signed char x, signed char y);
int _modsint (int a, int b);
unsigned int _moduchar (unsigned char x, unsigned char y);
unsigned int _moduint (unsigned int a, unsigned int b);
```

- 编译器自动生成对函数的调用

当用户用 PTCC 支持的 C 语言编写程序的时候，编译器会将编译阶段不能计算出结果值的整数乘法、除法、取余操作自动编译生成对库 libptcc.a 中相应函数的调用，在编译生成的对应的汇编文件（.asm 文件）中，可以看到编译结果。如果编译阶段就可以计算出乘法、除法、取余操作的值，则编译器会计算出结果数值，在生成的汇编程序中直接使用结果值，不会调用库函数。

例如：

```
unsigned char IncUchar(unsigned char c)
{
    return c+1;
}
int main(void)
{
      unsigned char a = 0x86;
      unsigned char b = 0x42;
      unsigned int res;
res = a * IncUchar(b);
return res;
}
```

- 用户编程显式调用库函数

如果用户编写 C 程序的时候，想使用 libptcc.a 库中的函数，那么需要在 C 程序文件中加入形如“extern int _divsint (int x, int y);”的库函数的声明。

4.2 整数库libptcc.a的使用注意事项

生成调用 libptcc.a 库来进行乘法、除法、取余的运算时，会增加对 RAM 空间的使用。如果不是特别必要，建议用户减少调用库来实现这些运算，可以考虑使用移位等运算完成同样的运算功能。

如果调用这些库函数，需要明确指定参与乘法/除法/取余运算的操作数的类型，才能调用对应的函数；根据不同的参数类型调用不同的函数，可以优化 ROM/RAM 空间的使用。

4.3 芯片硬件支持乘除法指令时使用整数库libptcc_muldiv.a

有些 8 位 MCU 硬件实现了对 8 位数据的乘/除法指令，具体请参加芯片对应的规格书。如果 8 位 MCU 的硬件支持 8 位数据的乘/除法指令，那么 C 程序中单字节的数据和变量的乘/除运算，C 编译器会直接生成乘/除法汇编指令，对于超过 8 位的变量的乘除法等运算，C 编译器会继续生成函数调用。但是，此时被调用的函数内部可以使用 8 位的乘/除法指令来减少指令数。因此，对于支持 8 位乘法、除法运算的 MCU，编译系统会在链接时链接 libptcc_muldiv.a 库。

这些库都在集成开发环境 PT-IDE 软件包中，使用 IDE 时，在编译过程中根据芯片型号自动链接对应的库，不用用户手动设置。

5. 历史记录

版本号	修改记录	发布日期
V1.0	初版	2024-01-19
V1.1	将“编写高效C程序的注意事项”单独编写了一个文档	2024-03-18
V1.2	修订整个文档	2024-09-26
V1.3	增加了硬件支持8位乘/除法时编译器调用整数库的说明	2025-03-03