Holtek C Compiler V3 FAQ

版本：V2.00　日期：2023-12-22

www.holtek.com

用户须知

1. 所有文档均会过时，本文档也不例外。Holtek 的工具和文档将不断演变以满足客户的需求，因此实际使用中有些对话框和工具说明可能与本文档所述之内容有所不同。请访问我们的网站 (http://www.holtek.com.tw/zh/mcu_tools_users_guide) 获取最新文档。

2. 这里假定读者已具备如下基本素质 :
   - 知道如何编写 C 程序
   - 已经阅读并理解所使用单片机的数据手册

目录

Part I V3 基本介绍

1.1 V3 的版本历程 ?

A：

Release 日期	V3 Compiler 版本号	对应的 IDE 版本号
2012/12	C Compiler V3.10	HT-IDE30007.7
2013/10	C Compiler V3.20	HT-IDE30007.8
2014/03	C Compiler V3.30	HT-IDE30007.82
2014/09	C Compiler V3.31	HT-IDE30007.85
2015/01	C Compiler V3.40	HT-IDE30007.86
2015/11	C Compiler V3.41	HT-IDE30007.89
2016/06	C Compiler V3.42	HT-IDE30007.90
2016/12	C Compiler V3.50	HT-IDE30007.93
2017/05	C Compiler V3.51	HT-IDE30007.94
2017/12	C Compiler V3.52	HT-IDE30007.96
2018/07	C Compiler V3.53	HT-IDE30007.97
2018/11	C Compiler V3.54	HT-IDE30007.98
2020/09	C Compiler V3.59	HT-IDE30008.04
2024/01	C Compiler V3.70	HT-IDE30008.1.6

1.2 新版改进的功能

A：

V3.70 改进的功能

- 优化带乘除法器 (MDU) 之 MCU 的数学运算。
- 优化浮点数运算程序，删除未调用函数。
- 修改 switch/ 函数传参等少数情况出现的已知 bug。

V3.51，V3.52，V3.53，V3.54，V 3.59 改进的功能

- 修改部分 bug

V3.5 改进的功能

- 增加 bit 数据类型 (详见《C Compiler V3 使用手册》2.2.11 节)
- 修改部分 bug

V3.42 改进的功能

- 增加硬件乘除法器功能 (需 MCU 本身有 MDU 功能 , 详见《C Compiler V3 使用手册》2.2.10 节)
- 修改部分 bug

V3.41 改进的功能

- 修改不传优化参数时，部分 MCU 写 EEPROM 失败的问题
- 修改部分 bug

V3.40 改进的功能

- 修改 V3.31 的所有已知 bug
- 优化具扩展指令 MCU 的 RAM 空间分配，详见《C Compiler V3 使用手册》10.1 节

V3.31 改进的功能

- 支持入口函数与 main 函数不在一个档案
- 修改 const array 作为函数参数的 bug
- 增加 GCC_DELAY(n) 内建函数，详见《C Compiler V3 使用手册》2.2.3 节

V3.30 改进的功能

- 新增支持指定程序入口函数，详见《C Compiler V3 使用手册》2.2.9 节
- 修改 startup 程序，避免使用 TABRD 指令
- 当函数参数缺少类型时报错

V3.20 改进的功能

- 支持 float/double 数据类型，以及 C Standard library
- 支持扩展指令架构的 MCU，如 HT66F70A
- 支持 const 变量指定地址，详见《C Compiler V3 使用手册》2.2.7 节
- 支持 function 指定地址，详见《C Compiler V3 使用手册》2.2.8 节
- 兼容 V2 版本的内嵌汇编语法

1.3 V3 的使用手册有哪些？

A：http://www.holtek.com.tw/zh/mcu_tools_users_guide

《C Compiler V3 使用手册》

《Holtek C Compiler V3 FAQ》

HT-IDE3000 目录下的《标准函数库使用手册》

1.4 V3 不支持的 MCU 有哪些？

A：V3 不支持 MP 少于 8bit 的 MCU，其型号如下列表，具扩展指令的 MCU 只有 V3.20 以上版本支持

MCU name			
HT45F2Y	HT46R46	HT48E30	HT66F03M
HT45R04	HT46R46-H	HT48F06E	HT66F03T3
HT45R0G	HT46R47-H	HT48F10E	HT66F13
HT45R34	HT46R48A	HT48F30E	HT66F20
HT45R35	HT46R51	HT48R002	HT66F23D
HT45R35V	HT46R52	HT48R003	HT66F30
HT45R36	HT46R53	HT48R005	HT66FB30
HT45F39	HT46R54	HT48R006	HT66FU30
HT46C22	HT46R71D	HT48R01A	HT68F002
HT46R22	HT46R71D-1	HT48R02	HT68F003
HT46C46E	HT46R72D-1	HT48R063	HT68F03
HT46R46E	HT46R72D-1A	HT48R063B	HT68F03C
HT46C47	HT46R73D-1	HT48R064	HT68F03M
HT46R47	HT46R73D-1A	HT48R064B	HT68F03T3
HT46C47E	HT46R74D-1	HT48R064D	HT68F13
HT46R47E	HT46RU22	HT48R064G	HT68F20
HT46C48AE	HT46R92	HT48R07A-1	HT68F30
HT46R48AE	HT48C05	HT48R08A-1	HT68FB30
HT46C62	HT48R05A-1	HT48R09A-1	HT68FU30
HT46R62	HT48C06	HT48R52	HT82J97A
HT46F46E	HT48R06A-1	HT48R52A	HT82J97E
HT46F47E	HT48C062	HT48R53	HT82K72A
HT46F48E	HT48R062	HT48RA0-5	HT82M39
HT46R002	HT48C10-1	HT48RA0-6	HT82M39B
HT46R003	HT48R10A-1	HT49C10-1	HT82M72A
HT46R003B	HT48C30-1	HT49C30-1/	HT82M98
HT46R02	HT48R30A-1	HT49R30A-1	HT82M99A
HT46R004	HT48CA0	HT49C30L	HT82M99E
HT46R005	HT48RA0A	HT49CA0	HT82M99AE
HT46R01A	HT48CA0-1	HT49RA0	HT82M99EE
HT46R064	HT48RA0-1	HT49RA0-6	HT83020
HT46R064B	HT48CA0-2	HT49R10A-1	HT83F10
HT46R064D	HT48RA0-2	HT56R22	HT83F20
HT46R064G	HT48CA0-3	HT56R62	HT83F40
HT46R12A	HT48RA0-3	HT66F002	HT83F60
HT46R32	HT48CA6	HT66F003	HT83F80
HT46R321	HT48E06	HT66F03	HT83P00-1
HT46R322	HT48E10	HT66F03C	HT83R00

1.5 什么是扩展指令架构 MCU?

A：具有 LMOV，LSET 等扩展指令 (长度为 2word) 的 MCU，如 HT66F70A，一颗 MCU 是否有扩展指令，可以查看它的 datasheet，是否有 LMOV 等扩展指令，每条扩展指令比对应的普通指令增加 1 个指令周期。

Part II V3 与 V2 的不同

2.1 V3 与 V2 语法有何不同？与 V1, V2，标准 C 的比较？

A：V3 与 V2 主要是绝对地址变量，中断语法，内嵌汇编等语法不同，详见《C Compiler V3 使用手册》第 2.2 节

V3, V2, V1 与标准 C 的比较表见《 C Compiler V3 使用手册》第四章

2.2 V3 对 V2 改进的功能

A：

	V3	V2
全局变量	支持初始化，详见《C Compiler V3 使用手册》2.2.4 节	不支援初始化
const 变数	支持最大 size 64page 支援 extern const 支持 const 变量指定地址，详见《C Compiler V3 使用手册》2.2.7 节	size 超过 1page 有可能出错
数组	支持三维以上数组	只支持二维以下
中断向量	可以调用函数，详见《C Compiler V3 使用手册》2.2.1 节	不能调用函数
函数	支持设定入口函数属性，详见《C Compiler V3 使用手册》2.2.9 节	不支持入口函数

2.3 将 V2 程序改至 V3 常见的错误

2.3.1 ISR warning

e.g.

```
#pragma vector Int_isr @ 0x04
void Int_isr() {}
```

warning: ignoring #pragma vector Int_isr [-Wunknown-pragmas]

解决方式：

使用正确的中断函式语法：void __attribute((interrupt(0x04))) Int_isr() {}

详见《C Compiler V3 使用手册》2.2.1 节

注意：

i：因为是 warning，如果不修改也可以编译通过，只是编译程序会将它当一般函式处理，而不是中断函式

ii：#pragma 的其他关键词 rambank/function 等，也会报此 warning，表示在 V3 版此功能无效。

2.3.2 内嵌汇编 error

e.g.

```
#asm
nop
#endasm
```

error: invalid preprocessing directive #asm
error: invalid preprocessing directive #endasm

解决方式：

选择参数“兼容 Holtek C V2 内嵌汇编语言”

详见语法书《 C Compiler V3 使用手册》2.2.5 节

2.3.3 bit 变数 error

e.g.bit a;

error: unknown type name 'bit'

解决方式：使用 HT-IDE3000 V7.93 以上版本

2.3.4 标志位 error

e.g. _40_1 = 1;

error: '_40_1' undeclared (first use in this function)

解决方式：

1. 用结构体位域定义 bit flag：

```
bit_type bit_var __attribute__ ((at(0x40)));
#define _40_1 bit_var.bit1
```

详见《C Compiler V3 使用手册》2.2.3 节

2. 使用 bit 定义：

```
static volatile bit flag1 __attribute__ ((at(0x40),bitoffset(1)));
```

详见《C Compiler V3 使用手册》2.2.11 节

2.3.5 内部函数 error

e.g. _delay(2);

Error(L2001): Unresolved external symbol '__delay' in file

解决方式：

改成：

```
#include “ht66f50.h”
GCC_DELAY(2);
```

详见《C Compiler V3 使用手册》2.2.3 节

2.3.6 绝对地址变数 error

e.g.

```
unsigned char a @ 0x40;
```

error: stray '@' in program

error: expected '=', ',', ';', 'asm' or '__attribute__' before numeric constant

解决方式：

改成以下语法：

```
volatile static unsigned char var_name __attribute__ ((at(0x40)));
```

详见《C Compiler V3 使用手册》2.2.2 节

2.3.7 函式指标 error

e.g.

```
void FileFunc(){}
void EditFunc(){}
void main()
{
    typedef void (*funcp)(void);
    funcp pfun= FileFunc;
    pfun();
    pfun = EditFunc;
    pfun();
}
```

error: incompatible types when initializing type 'funcp' using type 'void()'

error: incompatible types when assigning to type 'funcp' from type 'void()'

解决方式：目前 V3 尚不支持 function pointer

Part III V3 的特殊语法及用法

3.1 如何将一个变量定义于指定 bank?

A：若是无扩展指令架构的 MCU，只能将变量定义在指定地址，如：

volatile static unsigned char var_name __attribute__ ((at(0x140)));
详见《C Compiler V3 使用手册》2.2.2 节

若是扩展指令架构的 MCU，无需特别指定，linker 会自动分配变量到任意 bank。

3.2 如何将函数定义于指定地址？

A：IDE 7.8 以上版本才支持此功能，语法：

```
char __attribute__((at(0x373))) foo (char parm)
{}
```

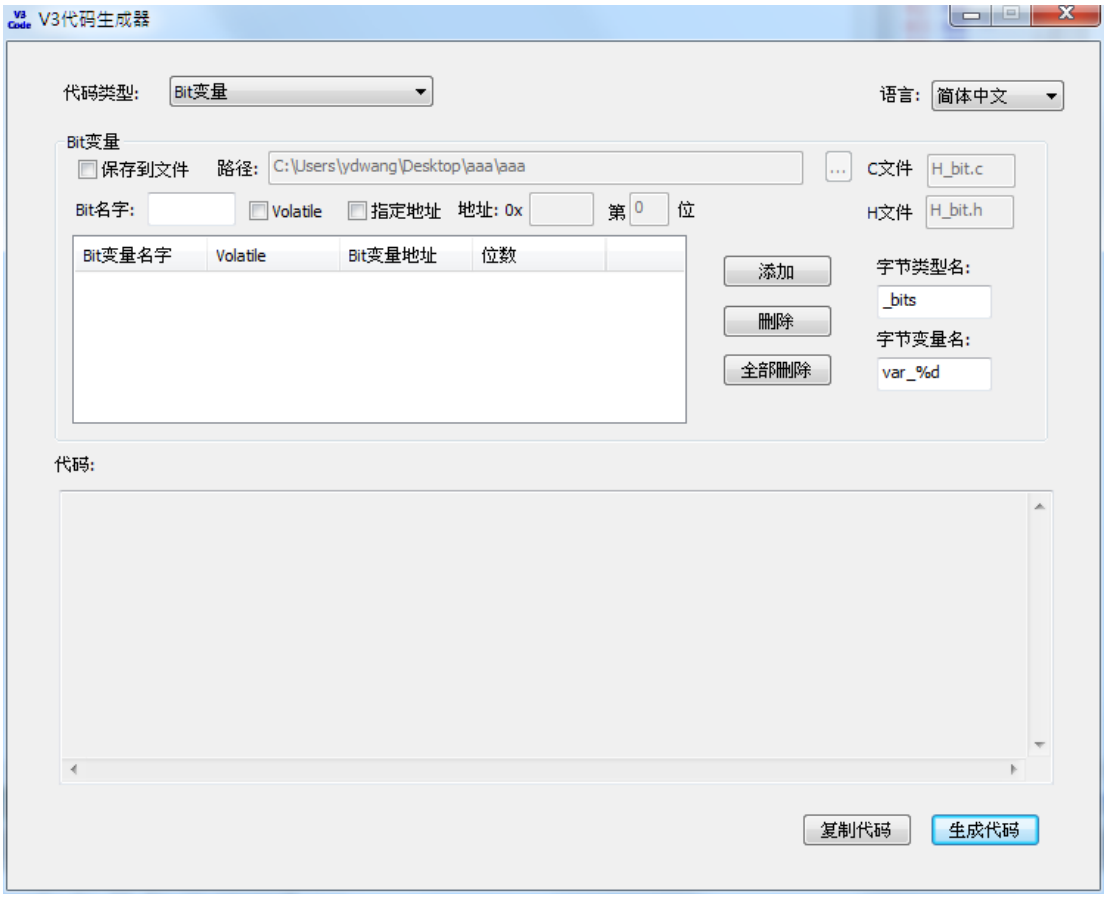
表示将函数 foo 指定在地址 0x373. 详见《C Compiler V3 使用手册》2.2.6 节

3.3 如何在 V3 中使用混合语言 (mixed language)?

A：参考《C Compiler V3 使用手册》第 7 章节

3.4 V3 代码生成器

为协助用户更好的使用 V3 专用语法，IDE3000 7.83 以上版本增加工具“V3 代码生成器”，位于菜单栏 → 工具 → V3 代码生成器。它可以成生 bit 变量，中断语法，绝对地址变量，内嵌汇编，delay 函数及指定其它 bank 的变量等语法，具体可查看 HT-IDE3000 使用手册第三章“V3 代码生成器”部份。


V3代码生成器
代码类型:
Bit变量
语言:
简体中文
Bit变量
保存到文件
路径:
C:\Users\ydwang\Desktop\aaa\aaa
C文件
H_bit.c
Bit名字:
Volatile
指定地址
地址: 0x
第
0
位
H文件
H_bit.h
Bit变量名字
Volatile
Bit变量地址
位数
添加
删除
全部删除
字节类型名:
_bits
字节变量名:
var_%d
代码:
复制代码
生成代码

Part IV V3 常见的 error, warning 及其解决方式

4.1 error“multi-ram-bank should be equipped with mp1”

A：请确认 MCU 是否为扩展指令架构，若是，请使用 IDE7.8 以上版本

4.2 error“internal compiler error:xxxx”

A：compiler 内部错误，请与 Holtek 公司返馈

4.3 error (L1038)“RAM (bank0) overflow, memory allocation fails for section ….”

A：对于无扩展指令的 MCU，C Compiler 会默认把变量配置到 RAM bank0 (具扩展指令的 MCU 可以自动配置到任意 bank)，当 bank0 满了之后，会报 RAM bank 0 overflow 出现此信息后，做法如下：

- 确认 unsigned int/long 类型的大小 (特别是从 V1 C Compiler 移植过来的程序)
- 若为 multi RAM bank MCU，可手动将全局变量调到其它 bank，参考 3.1

4.4 error (L1038)“ROM/RAM (bank*) overflow, memory allocation fails for section ….”

A：ROM 或 RAM 空间不够出现此信息后，做法如下：

- 检查是否打开优化参数 -Os，参考《C Compiler V3 使用手册》2.1.4 节
- 删减不必要的程序

4.5 warning (L3010) (absolute address: xxh, length: x) is overlay with (absolute address: xxh, length: x)

A：出现此 warning 的情况：

不同变量定义的地址重迭，如下，_b 与 _a 地址重迭，需将 _b 定义在 0x0142

```
DEFINE_SFR(unsigned int _a, 0x0140);
DEFINE_SFR(unsigned char _b, 0x0141);   //error
```

4.6 warning (L3009): Same sub function exists between ISR(04H) CMG and MAIN CMG: _func

A：中断服务程序 (04H) 与主函数共同调用子函数 func，解决方式：

- 避免共同调用

详见《C Compiler V3 使用手册》2.2.1 节

Part V V3 常见的问题及其解决方式

5.1 如何在 V3 使用 bit 变量？

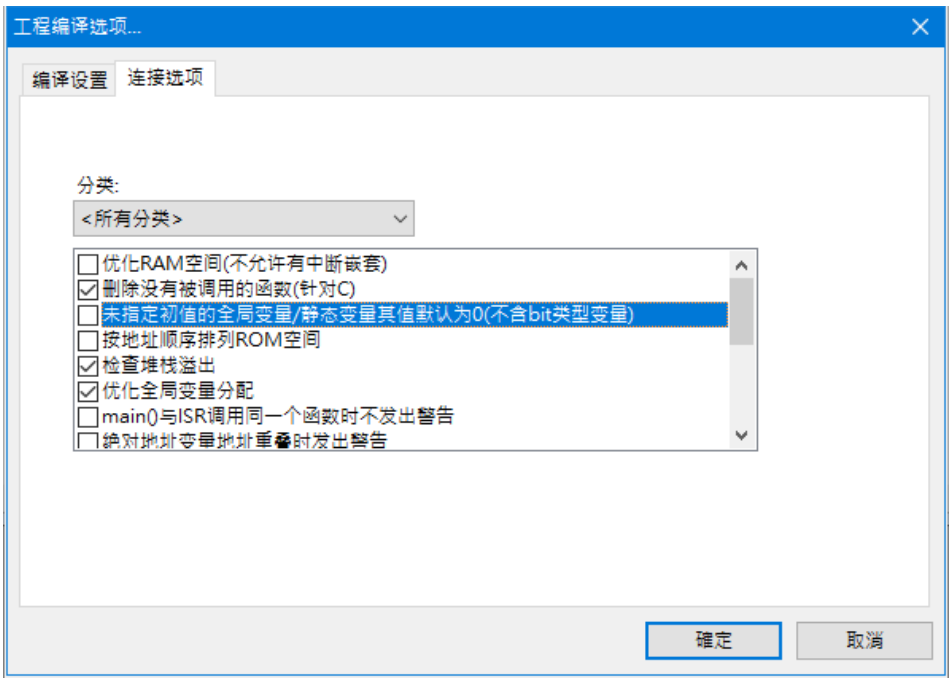
A：bit flag1; 详见《C Compiler V3 使用手册》2.2.11 节。

5.2 如何在 V3 使用外部定义 bit 变量？

A：extern bit flag1;

5.3 程序 reset 后，变量被清 0 的解决方式？

A：IDE7.8 版本将提供选择不初始化的方式：不选择下面选项“将未初始化的…”即可。



5.4 指定地址的变量，其他文件如何引用？

A：因指定地址的变量 (非 const) 需定义成 static，其它作用域只在本文件，所以可以将其定义在头文件里面，若有其它文件需要用到，直接 include 这个头文件即可，例：

```
//Define_var.h
static volatile unsigned  var1  __attribute__ ((at(0x180)));
//test1.c
#include "Define_var.h"
void foo1()
{
          var1 = 1;
}
//test2.c
#include "Define_var.h"
void foo2()
{
          var1 = 2;
}
```

注：若是 const 变量，不需要定义成 static，用 extern 即可，例：

```
//test1.c
const int __attribute__((at(0x3400))) bb[3]={1,2,3};
//test2.c
extern const int  bb[3];
int b;
void fun()
{
   b=bb[2];
}
```

5.5 对于写 EEPROM 有限制的 MCU (需连续 set wren, wr, flag)，如何使用 V3 写 EEPROM?

A：

i：在 V3 中，因为 _rden 与 _rd 是在 bank1，使用扩展指令，与 datasheet 所规定的指令不同。

ii：这种对指令有严格要求的功能，建议使用内嵌汇编达成，因为 C 不敢保证按特定的方式翻译。

iii：若要产出与 datasheet 中的连续指令，V3 需改成如下写法：

```
typedef struct {
    unsigned char bit0 : 1;
    unsigned char bit1 : 1;
    unsigned char bit2 : 1;
    unsigned char bit3 : 1;
    unsigned char bit4: 1;
    unsigned char bit5 : 1;
    unsigned char bit6 : 1;
    unsigned char bit7: 1;
}iar_bits;
DEFINE_SFR(iar_bits, iar1, 0x02);
#define     iar1_3          iar1.bit3
#define     iar1_2          iar1.bit2
#define     iar1_1          iar1.bit1
#define     iar1_0          iar1.bit0  //rd
```

```
unsigned short EERead(void)
{   unsigned short Backup;
    _mp1=0x40;
    Backup=_bp; _bp=1;
    iar1_1=1; iar1_0=1;
    while(iar1_0);
    _iar1=0;
    _bp=Backup;
    return _eed;
}

void EEWrite(unsigned short EEdata)
{   unsigned short Backup;
    _eed=EEdata;
    _mp1=0x40;
    Backup=_bp; _bp=1;
    _emi=0;
    iar1_3=1; iar1_2=1;
    _emi=1;
    while(iar1_2);
    _iar1=0;
    _bp=Backup;
}
```

5.6 使用 V3 给 bit flag 赋值变量的注意事项

范例：

```
unsigned char flag;
_pa2=flag;
```

编译程序翻译出的 asm：	而不是：
CLR PA2 SZ _flag SET PA2	SZ _flag JMP L1 CLR PA2 JMP L2 L1: SET PA2 L2:

说明：

编译程序只关心计算的结果，而不管计算过程，为减少 CODE 的指令输出，编译程序会翻译左边的指令。

C 语言与汇编不同的是，一条语句不只翻译出一条指令，所以在语句执行结束前，计算未完成。

影响：

不管 flag 值为多少，PA2 都会被先 CLR，如果在此时进入中断，而中断中有用到 PA，则会影响到执行结果。

解决方式：

- 在 bit 赋值前先把中断关掉，等计算完后再打开

总结：

多字节变量的计算，如果中断中有判断此变量，在计算未完成前，都不可进入中断。

5.7 在 V3 使用 ROM BP 的注意事项

对于多 ROM BANK MCU，若整个工程都用 C 语言写，那么，任何时候，用户都不需要设定 ROM BP，Linker 会自动设置 ROM BP，特别注意，若用户在程序中改动到 ROM BP，那么程序将有可能跑飞，用户需特别小心，在设置 RAM BP 时，也应注意不要用到 ROM BP。若工程使用混合语言编程，用 C 函数 call 汇编 section 时，应使用 C 语言或 fcall，在汇编 section call C 函数时，需在 CALL 前设 BP，前在 CALL 之后恢复 BP。

范例：

```
;;Test1.asm
extern _fun2:near
public _fun1
_fun1 .section  'code'
_fun1 proc
mov a, bank _fun2
mov [04H],a  ;; 假设 ROM BP 在 04h
call _fun2
mov a,bank_fun1
mov [04H],a
_fun1 endp
//Test2.c
extern void FUN1();
// 或者 asm("extern _FUN1:near");
void main()
{
   FUN1();
// 或者 asm("fcall _FUN1");
}
void fun2()
{}
```

5.8 混合语言使用 ROM BP 的注意事项

参考 5.7 节

5.9 如何使用 CMD 命令编译 C 工程？

compiler，assembler，linker 的参数详见《V3 C Compiler 使用手册》附录 C

例：

a. 设置环境变量：

```
set HTCFG=C:\Program Files\Holtek MCU Development Tools\HT-IDE3000V7.x\MCU
set HTBIN=C:\Program Files\Holtek MCU Development Tools\HT-IDE3000V7.x\BIN
set HTINCLUDE=C:\Program Files\Holtek MCU Development Tools\HT-IDE3000V7.
x\INCLUDE_V3
set HTLIB=C:\Program Files\Holtek MCU Development Tools\HT-IDE3000V7.x\LIB
```

b. 编译 .c 文件

```
…\hgcc32.exe  t1.c -g -Os -I "%HTINCLUDE%" -o t1.asm
…\hgcc32.exe  t2.c -g -Os -I "%HTINCLUDE%" -o t2.asm
```

c. 编译 .asm 文件

```
…\hasmgcc32.exe /hide=12345678 /chip=HT66F50 /case /z "t1.asm"
…\hasmgcc32.exe /hide=12345678 /chip=HT66F50 /case /z "t2.asm"
```

d. link 所有 obj,lib 产生 .tsk 文件

```
…\hlinkw32.exe /MCU=HT66F50 @ "C:\link-test.bat"
```

link-test.bat 内容：

```
"t1.obj"+
"t2.obj",
"test.tsk",
"test.map",
"test.dbg",
"libholtekgcc.lib";
```

5.10 混合语言工程在 asm 文件中使用 table read 的注意事项

如果一个工程即有 .c 又有 .asm 文件，那么在 .asm 文件中使用 table read 应注意，先将中断关掉，执行完后再打开。比如：

```
clr emi
tabrd r0
inc tblp
mov a,tblh
…
set emi
```

5.11 中断函数中使用 inline assembly 注意事项

在中断及其调用的函数中如果有使用 inline assembly，且 inline assembly 有使用到特殊寄存器 (如 MP, TBLP, TBHP, TBLH 等)，则用户需要自己保存这些寄存器，如下语句：

```
DEFINE_ISR(isr04,0x04)
{
  asm("mov a,[01h]"); // mp0 = [01h]
  asm("mov temp_mp0,a");
  asm("mov a,80h");
  asm("mov [01h],a ");
  asm("mov a,[00h]");
  asm("mov a,temp_mp0");
  asm("mov [01h],a");
}
```

5.12 用其它方式修改 const 变量值 (比如烧录时修改 ROM)，compiler 执行结果不变，如何解决 ?

比如：定义

```
__attribut__((at(0x400)))
const unsigned char array[] = {0,1,2,3,4,5,6,7};
```

烧录时把 ROM 的 400H~410H 清 0，执行 temp = array[7]; 时，temp 值仍为 7。

解决方式：

把 array[] 定义在与 temp = array[7]; 不同的 C 文件。

5.13 使用内嵌汇编语言的注意事项

汇编语言中使用到的变量、函数名、寄存器、标志位应遵循汇编语言的定义。

1. 如果全局变量 / 函数本文件只在内嵌汇编语言中使用，应增加汇编语言声明，比如：

```
asm("extern _a:byte");
asm("extern _func:near");
void main()
{
  asm("clr _a");
  asm("call _func");
}
```

2. 寄存器 / 标志位应先定义后使用，可 include INC 文件，比如：

```
asm("#include HT66F60.INC")
void main()
{
  asm("CLR ACC");
  asm("MOV TBHP,A");
  asm("CLR C");
}
```

3. 汇编语言中的名字要区分大小写。

5.14 绝对地址变量的地址被其它变量占用

如果绝对地址变量在程序中没有用到，那么连接器会将其它变量分配到此地址。

5.15 函数的局部变量地址如何分配

参考《C Compiler V3 使用手册 》1.2.3 节。

Part VI V3 常见的优化问题

6.1 使用 V3 优化参数后，在 watch window 上看不到某些变数的 debug 信息？

A：因启用优化参数，变量有可能被优化删除，故没有 debug 信息，若要在 debug 时观察变量值，可以将此变量暂时定义为 volatile，debug 结束后再删去，比如：

```
volatile int i, j, k;
```

6.2 中断与一般函数访问同一个全局变量，此全局变量的相关语句被优化掉？

A：一般函数与中断没有调用关系，compiler 不知中断何时发生，会影响到一般函数中的变量，因此，建议将此变量用 volatile 修饰，比如：

flag 在中断 ISR_INT0 中使用到，又在 main 函数中用到，则应定义成 volatile：

```
volatile unsigned char flag;
```

```
    do
     {
     _emi=1;
     }while (flag==0);

//   _nop();
//   _nop();
   GCC_NOP();
   while(1);
}


void ISR_INT0()
{
  flag=1;
  _pb=0xff;          //0b00000001;
  _int0f=0;                 // clr interrput
}
```

说明：volatile：一个类型修饰符 (type specifier)。它是被设计用来修饰被不同函数访问和修改的变数，使用 volatile 修饰的变量，不会因编译程序的优化而被省去。

建议定义成 volatile 的变量：特殊寄存器，中断函式使用到的变量，为某些特殊用途的代码定义的变量 (比如 delay 功能)。

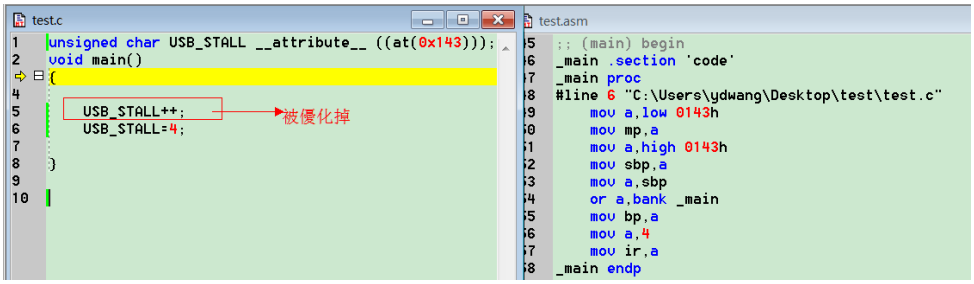
6.3 V3 的优化功能及其对 debug 的影响？

A：详见《C Compiler V3 使用手册》第 3 章

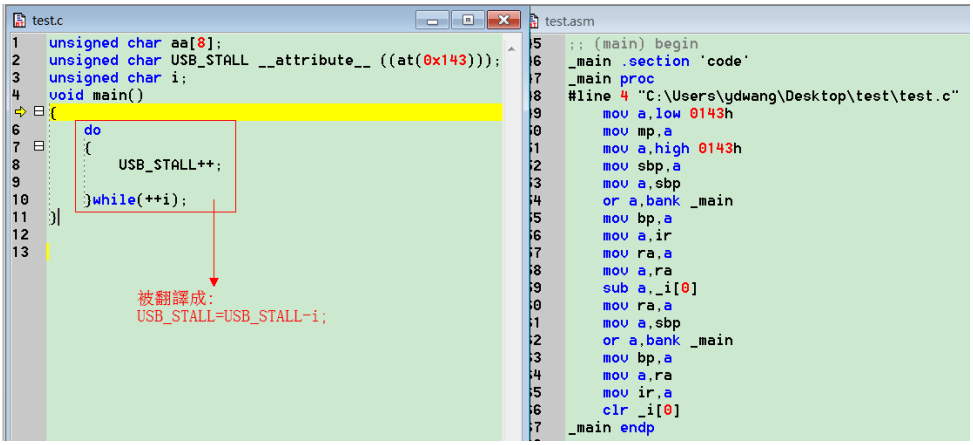
6.4 使用 V3 compiler, debug 时行号出错？

A：可能出现的情况：

a. 某些语句被优化掉而不译出 code，自然也没有 debug 信息。



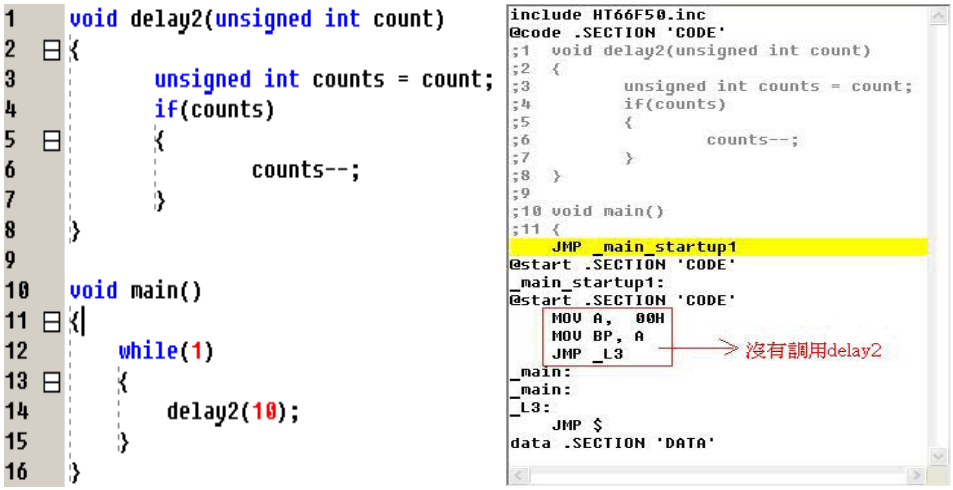
b. 几条语句翻译出同一堆的 code，只显示一条 line number。



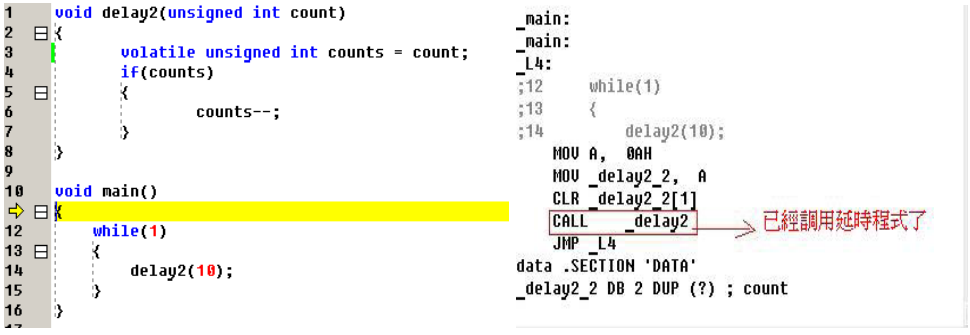
这些情况会可能会影响 debug，但执行结果不会出错，若不是这两种情况，请回报。

6.5 使用 V3 compiler，用于延时的循环代码被优化，怎么解决？

A：如下：



解决方法：将变量定义为 volatile，如下：

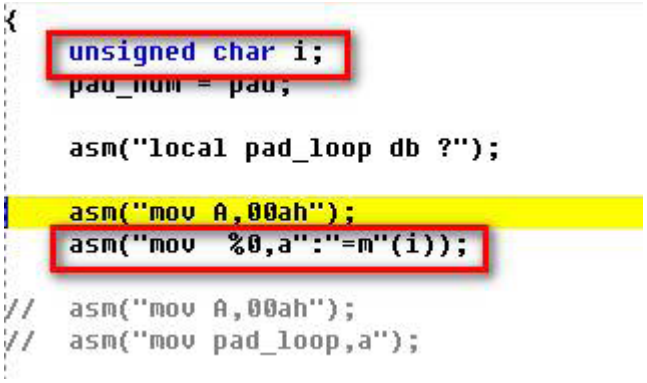


6.6 使用内嵌汇编，被优化掉？

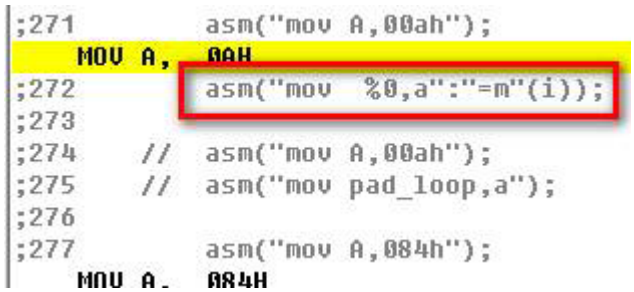
如下：

```
asm("mov %0,a":"=m"(i));// 表示把 ACC 的值赋给 i,
```

因为之后变量 i 没有被用到，所以这条语句被编译程序优化掉，



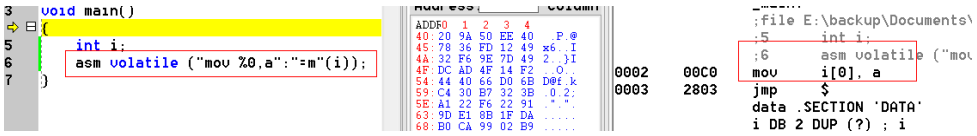
编译后



如果要保留，可以改成：

```
asm volatile ("mov %0,a":"=m"(i));
```

防止被优化



6.7 选择优化参数后，延时函数时间发生变化。

A：延时函数的延时时间通常依赖于执行的指令数，选择优化参数后，compiler 编译出的指令更精简，从而影响延时时间，程序开发者应注意这点，在选择优化参数后，调整延时函数或使用 compiler 自带的延时函数 GCC_DELAY(n)。

Part VII 修正中的 C Compiler 已知问题

7.1 少数情况下，使用 global bit，有可能产出错误的汇编语法

问题描述

在少数情况下，会出现错误的汇编语法，比如如下程序，输出的汇编变量 _mode_oper_2[-1] 会导致 Syntax error。

比如：

```
unsigned char cnt_time_enable;
bit   way4_IO;
bit   way4_on;
extern volatile unsigned char cnt_state;
extern unsigned char cnt_time_flag;
extern unsigned char djrflag;
unsigned char float_uintcmp()
{
   return 0;
}
void mode_oper(void )
{
   way4_IO=way4_on;
   if(float_uintcmp())
   {
           way4_IO=~(way4_on);
   }
   switch(cnt_state)
   {
           case 0x10:
              cnt_time_flag=0;
              break;
           case 0x11:
              cnt_time_enable=1;
              break;
   }
}
void djr_oper(void)
{
           if(float_uintcmp())     djrflag=1;
}
```

如何避开

a. 将 switch 改成 if/else, 或者

b. 将 way4_IO 与 way4_on 定义成 struct bit。

7.2 函数参数带乘除法运算执行错误

问题描述

未使用 MDU (硬件乘除法器) 的 MCU，当函数参数带有乘除法运算时，可能计算错误。

比如：

a. func(a/b,1);

b. unsinged int temp = a * b; func(temp,1);

c. func(3,a%b,1);

如何避开

定义一个 volatile temp 变量，先将表达式计算出来，比如：

volatile unsigned int temp = a * b;

func(temp,1);