特别说明

关于 DSP 中 CMD 文件的解释
DSP　CMD 文件的编写 EETOP 专业博客---电子工程师自己的家园 L }#g Z-n W r

1.　　COFF 格式

1>　　通用目标文件格式（Common Object File Format）是一种流行的二进制可执行文件格式，二进制可执行文件包括库文件（lib），目标

文件（obj）最终可执行文件（out）。，现今 PC 机上的 Windows95 和 NT4.0 以后的操作系统的二进制文件格式（PE）就是在 COFF 格式基础上的进

一步扩充。

2>　　COFF 格式：详细的 COFF 文件格式包括段头，可执行代码和初始化数据，可重定位信息，行号入口，符号表，字符串表等，这些属于编写

操作系统和编译器人员关心范畴。而对于 C 只需要了解定义段和给段分配空间就可以了。

3>　　采用 COFF 更有利于模块化编程,程序员可以自由决定愿意把哪些代码归属到哪些段，然后加以不同的处理。

2.　　Section 目标文件中最小单位称为块。一个块就是最终在存储器映象中占据连续空间的一段代码或数据。

1>　　COFF 目标文件包含三个默认的块：

.text 可执行代码

.data 已初始化数据

.bss 为未初始化数据保留的空间

2>　　汇编器对块的处理

未初始化块

.bss　　变量存放空间

.usect　　用户自定义的未初始化段

初始化块

.text　　汇编指令代码

.data　　常数数据（比如对变量的初始化数据）

.sect　　用户自定义的已初始化段

.asect　　通.sect，多了绝对地址定位功能，一般不用

3>C 语言的段

未初始化块（data）

.bss　　存放全局和静态变量

.ebss　　长调用的.bss(超过了 64K 地址限制)

.stack　　存放 C 语言的栈

.sysmem 存放 C 语言的堆

.esysmem　　长调用的.sysmem(超过了 64K 地址限制)

初始化块

.text　　可执行代码和常数(program)

.switch　switch 语句产生的常数表格（program/低 64K 数据空间）

.pinit　　Tables for global constructors (C++)(program)

.cinit　　用来存放对全局和静态变量的初始化常数值(program)

.const　　全局和静态的 const 变量初始化值和字符串常数，（data）

.econst 长.const（可定位到任何地方）（data）

3>　　自定义段（C 语言）

#pragma DATA_SECTION(函数名或全局变量名,"用户自定义在数据空间的段名或者说一输出段的名称");

#pragma CODE_SECTION(函数名或全局变量名,"用户自定义在程序空间的段名");

不能在函数体内声明。

必须在定义和使用前声明

#pragma 可以阻止对未调用的函数的优化

3.　　连接命令文件（CMD）

1>　　MEMORY 指定存储空间

```
MEMORY
{
PAGE 0:
name 0 [attr]    : origin = constant, length = constant
.............
PAGE n:
name n [attr]    : origin = constant, length = constant

}
```

PAGE n:标示存储空间，n<255；PAGE 0 为程序存储空间；PAGE 1 为 data 存储空间

name:存储空间名称

attr:存储空间属性：只读 R，只写 W，可包含可执行代码 X,可以被初始化 I。

orgin:用来定义存储空间的起始地址

Lenth:用来定义存储空间的长度

2>　　SECTIONS 分配段

```
   SECTIONS
{

name                : [property,property,……]
 LedDataRegsFile    : > LED_REG,      PAGE = 1
}
```

name:输出段的名称

SECTIONS：(在程序里添加下面的段名如.vectors.用来指定该段名以下，另一个段名以上的

程序(属于 PAGE0)或数据(属于 PAGE1)放到“>”符号

后的空间名字所在的地方。

property：输出段的属性：

load＝allocation（强制地址或存储空间名称）同>allocation：定义输出段将会被装载到哪里。

run= allocation（强制地址或存储空间名称）同>allocation：定义输出段将会在哪里运行。

注：CMD 文件中只出现一个关键字 load 或 run 时，表示两者的地址时表示两者的地址 shi 重合的。

PAGE = n，段位于那个存储页面空间。

```
例：ramfuncs                : LOAD = FLASHD,
                              RUN = RAML0,
                              LOAD_START(_RamfuncsLoadStart),
                              LOAD_END(_RamfuncsLoadEnd),
                              RUN_START(_RamfuncsRunStart),
                              PAGE = 0
```

3>　　直接写编译命令

```
-l rts2800_ml.lib       连接系统文件 rts2800_ml.lib
-o filename.out                 最终生成的二进制文件命名为 filename.out
-m filename.map         生成映射文件 filename.map
-stack 0x200                    堆栈为 512 字
```

4. .const 段：

由关键字 const 限定的全局变量（const 限定的局部变量不产生）初始化值，和出现

在表达式（做指针使用，而用来初始化字符串数组变

量不产生）中的字符串常数，另外数组和结构体是局部变量时，其初始值会产生.const 段，而全局时不产生。

例子
在结合硬件的编程中,有些变量需要特定地址.

一般可以采用指针变量.
例如:

```
unsigned int *LedReg   = (unsigned int *)0x5f00;//发光二极管地址
*LedReg = 0xFF;
```

但有些时候感觉不是太好时,可以像 Keil 那样

```
unsigned int LedReg at 0x5f00;//发光二极管地址
LedReg = 0xFF;
```

这样看着"比较顺眼"~~~
但在 TIDSP 的 CCS 环境下,不能采用 at 或__at 等来定位地址.
不过 CCS 也提供了类似的手段,不过有些"麻烦".

具体实现:

1.在 CMD 文件中

```
MEMORY
{
 PAGE 0:    /* Program Memory */

 PAGE 1:    /* Data Memory */

    LED_REG     : origin = 0x005f00, length = 0x00001 /*发光二极管地址*/
    LCD_ComL    : origin = 0x005100, length = 0x00001
    LCD_ComH    : origin = 0x005200, length = 0x00001
    LCD_DatL    : origin = 0x005300, length = 0x00001
    LCD_DatH    : origin = 0x005400, length = 0x00001
    EINT_REG    : origin = 0x005c00, length = 0x00001
    KEY_REG     : origin = 0x005b00, length = 0x00001
    RCLK_REG    : origin = 0x005500, length = 0x00001 /*数码管锁存地址*/
}


SECTIONS
```

```
{
    LedDataRegsFile    : > LED_REG,      PAGE = 1
    LcdComlRegsFile    : > LCD_ComL,     PAGE = 1
    LcdComhRegsFile    : > LCD_ComH,     PAGE = 1
    LcdDatlRegsFile    : > LCD_DatL,     PAGE = 1
    LcdDathRegsFile    : > LCD_DatH,     PAGE = 1
    KeyRegsFile        : > KEY_REG,      PAGE = 1
    EintRegsFile       : > EINT_REG,     PAGE = 1
    RclkRegsFile       : > RCLK_REG,     PAGE = 1
}
```

2.在 C/C++文件中

```
#ifdef __cplusplus
#pragma DATA_SECTION("LedDataRegsFile") /*C++语言格式*/
#else
#pragma DATA_SECTION(LedRegs,"LedDataRegsFile");/*C 语言格式*/
#endif
volatile unsigned int LedRegs;

/*以下为 C++语言格式*/

#pragma DATA_SECTION("LcdComlRegsFile")
volatile unsigned int LcdComL;
#pragma DATA_SECTION("LcdComhRegsFile")
volatile unsigned int LcdComH;
#pragma DATA_SECTION("LcdDatlRegsFile")
volatile unsigned int LcdDatL;
#pragma DATA_SECTION("LcdDathRegsFile")
volatile unsigned int LcdDatH;

#pragma DATA_SECTION("EintRegsFile")
volatile unsigned int EintRegs;

#pragma DATA_SECTION("KeyRegsFile")
volatile unsigned char KeyRegs;

#pragma DATA_SECTION("RclkRegsFile")
volatile bool SpiRclkRegs;
```

3.在应用程序中

```
extern unsigned int LedRegs;
extern volatile unsigned int EintRegs;
extern volatile unsigned char KeyRegs;
extern volatile bool SpiRclkRegs;
//.............
LedRegs = 0xff;//这样就不需要指针变量的写法了
//..................
```