DSP 的存储器的地址范围，CMD 是主要是根据那个来编的。
CMD 它是用来分配 rom 和 ram 空间用的,告诉链接程序怎样计算地址和分配空间.

所以不同的芯片就有不同大小的 rom 和 ram.放用户程序的地方也不尽相同.所以要根据芯片进行修改.分两部分.MEMORY 和 SECTIONS.

```
MEMORY
{ PAGE 0 ..........
PAGE 1.........
}
SECTIONS
{SECTIONS
{
.vectors ..................
.reset ..................
..................
}
```

MEMORY 是用来指定芯片的 rom 和 ram 的大小和划分出几个区间.
PAGE 0 对应 romAGE 1 对应 ram
PAGE 里包含的区间名字与其后面的参数反映了该区间的起始地址和长度.
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。

```
SECTIONS
{
.vectors : { } > VECS PAGE 0
.reset : { } > VECS PAGE 0
.............
.............
..........
}
eg:
MEMORY
{
PAGE 0: VECS: origin = 00000h, length = 00040h
LOW: origin = 00040h, length = 03FC0h
SARAM: origin = 04000h, length = 00800h
B0: origin = 0FF00h, length = 00100h
PAGE 1: B0: origin = 00200h, length = 00100h
B1: origin = 00300h, length = 00100h
B2: origin = 00060h, length = 00020h
SARAM: origin = 08000h, length = 00800h
}
```

```
SECTIONS
{
.text : { } > LOW PAGE 0
.cinit : { } > LOW PAGE 0
.switch : { } > LOW PAGE 0
.const : { } > SARAM PAGE 1
.data : { } > SARAM PAGE 1
.bss : { } > SARAM PAGE 1
.stack : { } > SARAM PAGE 1
.sysmem : { } > SARAM PAGE 1
}
```

由三部分组成：
输入/输出定义：这一部分，可以通过 ccs 的“Build Option.....”菜单设置

- 。obj 链接的目标文件
- 。lib 链接的库文件
- 。map 生成的交叉索引文件
- 。out 生成的可执行代码

MEMORY 命令：描述系统实际的硬件资源
SECTION 命令：描述“段”如何定位
例子
.cmd 文件

```
-c
-o hello.out
-m hello.map
-stack 100
-l rts2xx.lib
MEMORY
{
    PAGE 0: VECT:origin=0x8000,length 0x040
    PAGE 0: PROG:origin=0x8040,length 0x6000
    PAGE 1: DATA:origin=0x8000,length 0x400
}
SECTIONS
{
.vextors >VECT PAGE 0
.text >PROG PAGE 0
.bss >DATA PAGE 1
.const >DATA PAGE 1
}
```

存储模型：c 程序的代码和数据如何定位
系统定义

.cinit 存放程序中的变量初值和常量
.const 存放程序中的字符常量、浮点常量和用 const 声明的常量
.switch 存放程序中 switch 语句的跳转地址表
.text 存放程序代码
.bss 为程序中的全局和静态变量保留存储空间
.far 为程序中用 far 声明的全局和静态变量保留空间
.stack 为程序系统堆栈保留存储空间，用于保存返回地址、函数间的参数传递、存储局部变量和保存中间结果
.sysmem 用于程序中的 malloc 、calloc 、和 realoc 函数动态分配存储空间

CMD 的专业名称叫链接器配置文件，是存放链接器的配置信息的，我们简称为命令文件，其中比较关键的就是 MEMORY 和 SECTIONS 两个伪指令的使用，常常令人困惑，系统出现的问题也经常与它们的不当使用有关。CCS 是 DSP 软件对 DOS 系统继承的开发环境，CCS 的命令文件经过 DOS 命令文件长时间的引申发展，已经变得非常简洁（不知道 TI 文档有没有详细 CMD 配置说明）。我学 CMD 是从 DOS 里的东西开始的，所以也从 DOS 环境下的 CMD 说起：

1 命令文件的组成
命令文件的开头部分是要链接的各个子目标文件的名字，这样链接器就可以根据子目标文件名，将相应的目标文件链接成一个文件；接下来就是链接器的操作指令，这些指令用来配置链接器，接下来就是 MEMORY 和 SECTIONS 两个伪指令的相关语句，必须大写。MEMORY，用来配置目标存储器，SECTIONS 用来指定段的存放位置。结合下面的典型 DOS 环境的命令文件 link.cmd 来做一下说明：

```
file.obj //子目标文件名 1
file2.obj //子目标文件名 2
file3.obj //子目标文件名 3
- o prog.out //连接器操作指令,用来指定输出文件
- m prog.m //用来指定 MAP 文件
MEMORY
{ 略 }
SECTIONS
{ 略 }
otherlink.cmd
```

本命令文件 link.cmd 要调用的 otherlink.cmd 等其他命令文件，则文件的名字要放到本命令文件最后一行，因为放开头的话,链接器是不会从被调用的其他命令文件中返回到本命令文件。

2 MEMORY 伪指令
MEMORY 用来建立目标存储器的模型，SECTIONS 指令就可以根据这个模型来安排各个段的位置，MEMORY 指令可以定义目标系统的各种类型的存储器及容量。
MEMORY 的语法如下：

```
MEMORY
{
```

```
PAGE 0 : name1[(attr)] : origin = constant,length = constant
         name1n[(attr)] : origin = constant,length = constant
PAGE 1 : name2[(attr)] : origin = constant,length = constant
         name2n[(attr)] : origin = constant,length = constant
PAGE n : namen[(attr)] : origin = constant,length = constant
         namenn[(attr)] : origin = constant,length = constant
}
```

PAGE 关键词对独立的存储空间进行标记，页号 n 的最大值为 255，实际应用中一般分为两页,PAGE0 程序存储器和 PAGE1 数据存储器。

name 存储区间的名字，不超过 8 个字符，不同的 PAGE 上可以出现相同的名字（最好不用，免的搞混），一个 PAGE 内不许有相同的 name。

attr 的属性标识，为 R 表示可读；W 可写 X 表示区间可以装入可执行代码；I 表示存储器可以进行初始话，什么属性代码也不写，表示存储区间具有上述的四种属性，基本上我们都选择这种写法。

origin:略。

length:略。

下面是经常用的 2407 的简单写法大家参考,程序从 0x060 开始，要避开加密位,不从 0x0044 开始更可靠一点，此例中的同名的页可以只写第一个，其后省略，但写上至少安全一点：

```
MEMORY
{
PAGE 0: VECS: origin = 0x0000, length 0x40
PAGE 0: PROG: origin = 0x0060, length 0x6000
PAGE 1: B0 : origin = 0x200, length 0x100
PAGE 1: B1 : origin = 0x300, length 0x100
PAGE 1: DATA: origin = 0x0860, length 0x0780
}
```

3 SECTIONS 伪指令

SECTIONS 指令的语法如下：

```
SECTIONS
{
.text: {所有.text 输入段名} load＝加载地址 run =运行地址
.data: {所有.data 输入段名} load＝加载地址 run =运行地址
.bss: {所有.bss 输入段名} load＝加载地址 run =运行地址
.other: {所有.other 输入段名} load＝加载地址 run =运行地址
}
```

SECTIONS 必须用大写字母，其后的大括号里是输出段的说明性语句，每一个输出段的说明都是从段名开始,段名之后是如何对输入段进行组织和给段分配存储器的参数说明：

以.text 段的属性语句为例，“{所有.text 输入段名}”这段内容用来说明连接器输出段的.text 段由哪些子目标文件的段组成，举例如下

```
SECTIONS
```

```
{
.text:{ file1.obj(.text) file2(.text) file3(.text,cinit)}略
}
```

指明输出段.text 要链接 file1.obj 的.text 和 file2 的.text 还有 file3 的.text 和.cinit。在 CCS 的 SECTIONS 里通常只写一个中间没有内容的“{ }”就表示所有的目标文件的相应段

接下来说明“load＝加载地址 run =运行地址”链接器为每个输出段都在目标存储器里分配两个地址：一个是加载地址，一个是运行地址。通常情况下两个地址是相同的，可以认为输出段只有一个地址，这时就可以不加“run =运行地址”这条语句了；但有时需要将两个地址分开，比如将程序加载到 FLASH，然后放到 RAM 中高速运行，这就用到了运行地址和加载地址的分别配置了，如下例所示：

```
.const :{略} load = PROG run = 0x0800
```

常量加载在程序存储区，配置为在 RAM 里调用。

“load＝加载地址”的几种写法需要说明一下，首先“load”关键字可以省略，“＝”可以写成“>”，“加载地址”可以是：地址值、存储区间的名字、PAGE 关键词等，所以大家见到“.text:{ } > 0x0080”这样的语句可千万不要奇怪。“run =运行地址”中的“ = ”可以用“>”，其它的简化写法就没有了。大家不要乱用。

4 CCS 中的案例

在 CCS 中的命令文件好像简化了不少，少了很多东西，语句也精简了好多，首先不用指定输入链接器的目标文件，CCS 会自动默认处理，其次链接器的配置命令也和 DOS 的环境不同，需要了解的请找 TI 文档吧！下面是刘和平书中的例子，大家来看看是不是可以很精确的理解了呢！

```
-stack 40



MEMORY
{
PAGE 0 : VECS : origin = 0h , length = 40h
         PVECS : origin = 40h , length = 70h
         PROG : origin = 0b0h , length = 7F50h
PAGE 1 : MMRS : origin = 0h , length = 05Fh
       B2 : origin = 0060h , length = 020h
       B0 : origin = 0200h , length = 100h
       B1 : origin = 0300h , length = 100h
SARAM : origin = 0800h , length = 0800h
EXT : origin = 8000h , length = 8000h
}
```

```
SECTIONS
{
    .reset : { } > VECS PAGE 0
    .vectors : { } > VECS PAGE 0
    .pvecs : { } > PVECS PAGE 0
    .text : { } > PROG PAGE 0
    .cinit : { } > PROG PAGE 0
    .bss : { } > SARAM PAGE 1
    .const : { } > SARAM PAGE 1
    .stack : { } > B1 PAGE 1
}
```

第二章 CMD 文件的编写

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");

#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 SECTIONS 分配段

```
    SECTIONS

{
```

name : [property,property,……]

}

name:输出段的名称

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时重合的。

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。