窗体底端

**s3c6410\_MMU地址映射过程详述**

**1. ARMv6 MMU简述**

1）MMU由协处理器CP15控制；

2）MMU功能：地址映射(VA->PA)，内存访问权限控制；

**一. 与MMU管理有关的寄存器有：**

**C1：某些位 用于配置MMU中的一些操作  
C2：页表基地址，有效的为[31:14]，所以页表地址必须16KB对齐。  
C3：域(domain)的访问控制属性  
C4：保留  
C5：内存访问失效状态指示  
C6：内存访问失效时失效的地址  
C8：控制和清除TLB内容相关的操作  
C10：控制和锁定TLB内容相关的操作**

**二. 禁止/使能MMU**

**C1的0位控制禁止/使能MMU：  
MRC P15，0，R0，C1，0，0 ----> r0=c1**

**ORR R0，#01 -----> r0=r0|1**

**MCR P15（协处理器），0，R0（arm寄存器），C1（协处理器的寄存器），0，0（默认）**

**//R0寄存器的值被传送到p15 c1寄存器中 ------> c1=r0**

MCR指令将ARM处理器的寄存器中的数据传送到协处理器的寄存器中

MRC指令将协处理器的寄存器中数值传送到ARM处理器的寄存器中

**2. 址映射过程详述**

ARMv6的MMU 转换。

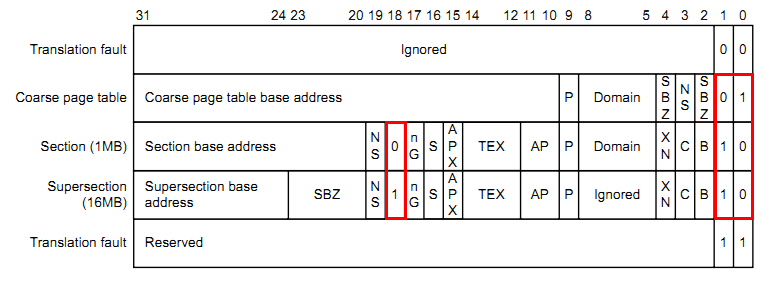
MMU地址映射 映射方式 映射粒度

一级页表 段映射 1M section和16M supersection

 二级页表 页映射 4K small page和64K large page

 硬件在做地址转换时，映射方式以及映射粒度可以从页表的入口描述符中获得

一级页表的入口描述符(first-level descriptor)：



第[1:0]位决定映射方式：

[1:0]=10b时，是段映射，此时只需作一级映射，描述符的最高12或8位存放的是段基址；

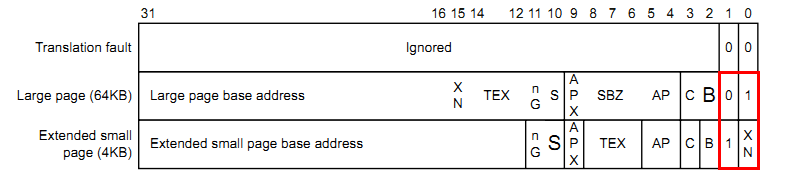
[1:0]=01b时，是页映射，此时虚拟地址转换为物理地址需要经历二级映射，描述符的最高22位存放的是二级页表的物理地址；

第[18]位决定段映射的粒度：

[18]=0b时，映射粒度为1M，描述符的最高12位存放段基址；

[18]=1b时，映射粒度为16M，描述符的最高8位存放段基址；

当映射方式为页映射时，二级页表，二级页表的入口描述符（second-level descriptor）：



一级页表总包含4096（2^12）个转换描述符，通过虚拟地址的31:20(12)位寻址的

页表的首地址就是TTB(Translation Table Base)中存储了段描述符的物理地址

第[1:0]位决定页映射的映射粒度：

[1:0]=10b或11b时，映射粒度为4KB，描述符的最高20位为页基址；

[1:0]=01b时，映射粒度为64KB，描述符的最高16位为页基址；

下面分4种情况对地址映射过程做详细描述：

1）段映射，映射粒度为1M

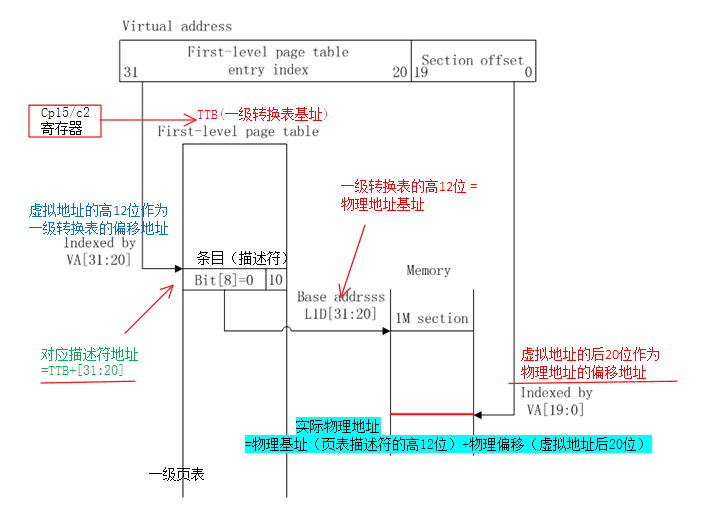
2）段映射，映射粒度为16M

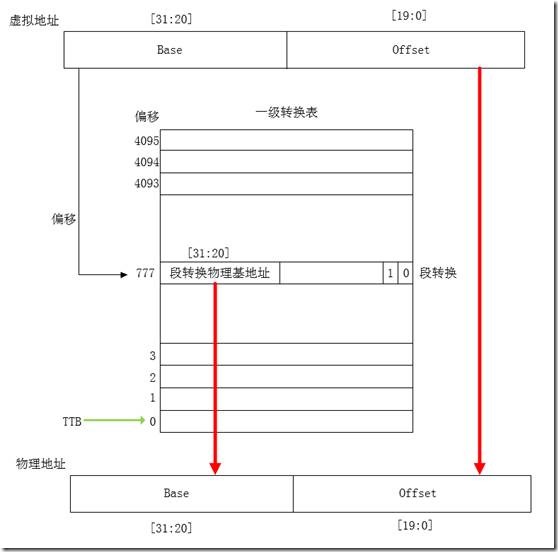
3）页映射，映射粒度为4K

4）页映射，映射粒度为64K

**2.1段映射，映射粒度为1M**

当映射方式为段映射，且映射粒度为1M时，映射图如下：



[](http://files.chinaaet.com/images/blog/2015/04/29/12450062582437.jpg)

映射流程：

当CPU从C2中读取TTB地址之后，如上图所示，将通过virtual address的[31:20]进行索引，这个索引所需要的地址为TTB+虚拟地址的偏移

  通过虚拟地址的高12位偏移，找到页表中的对应描述符，描述符其实就是一个存在ram中的32位的一个数据

一级页表基址（TTB）+ VA[31:20] （虚拟地址偏移）= 该虚拟地址对应的页表描述符的入口地址；（页表描述符的映射粒度由该描述符下第18we决定）

判断描述符的最后两位，确定是什么转换。10的话就表示是段转换。

  判断是段转换后，将描述符的高12位取出，这个就是物理的基地址，在和虚拟地址的后20位拼接，就得到对应的物理地址。

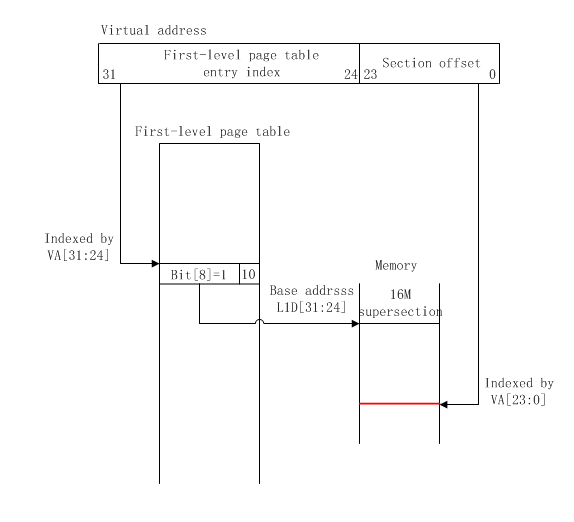
页表描述符的[31:20]位为该虚拟地址对应的物理段基址；

物理段基址+ VA[19:0]段偏移= 物理地址

由映射图可知，一个虚拟地址可以索引2^12个一级页表入口，每个入口映射2^20大小的内存，故虚拟地址可以映射的最大物理内存为：2^12 \* 2^20，即4G。

**2.2 段映射，映射粒度为16M**

当映射方式为段映射，且映射粒度为16M时，映射图如下：



虚拟地址到物理地址的映射过程如下：

虚拟地址的[31:24]位存放一级页表的入口index，[23:0]位存放段偏移；

从TTBR（translation table base register，协处理器CP15中的一个寄存器，用于存放一级页表的基址）寄存器中获取一级页表的基址；

一级页表基址+ VA[31:24] = 该虚拟地址对应的页表描述符的入口地址；

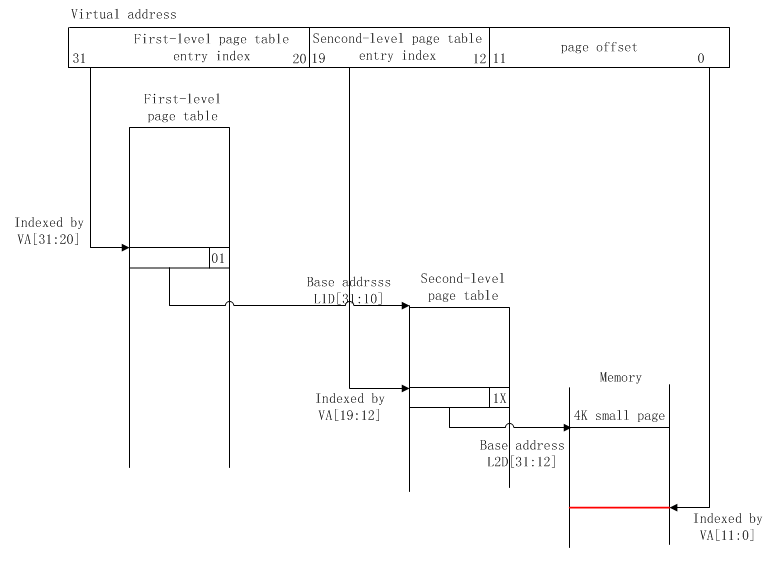
页表描述符的[31:24]位为该虚拟地址对应的物理段基址；

物理段基址+ VA[23:0]段偏移= 物理地址

由映射图可知，一个虚拟地址可以索引2^8个一级页表入口，每个入口映射2^24大小的内存，故虚拟地址可以映射的最大物理内存为：2^8 \* 2^24，即4G。

**2.3 页映射，映射粒度为4K**

当映射方式为页映射，且映射粒度为4K时，映射图如下：



虚拟地址到物理地址的映射过程如下：

虚拟地址的[31:20]位存放一级页表的入口index，[19:12]位存放二级页表的入口index，[11:0]位存放页偏移；

从TTBR（translation table base register，协处理器CP15中的一个寄存器，用于存放一级页表的基址）寄存器中获取一级页表的基址；

一级页表基址+ VA[31:20] = 一级页表描述符的入口地址；

一级页表描述符的[31:10]位存放二级页表的基址；

二级页表基址+ VA[19:12] = 二级页表描述符的入口地址；

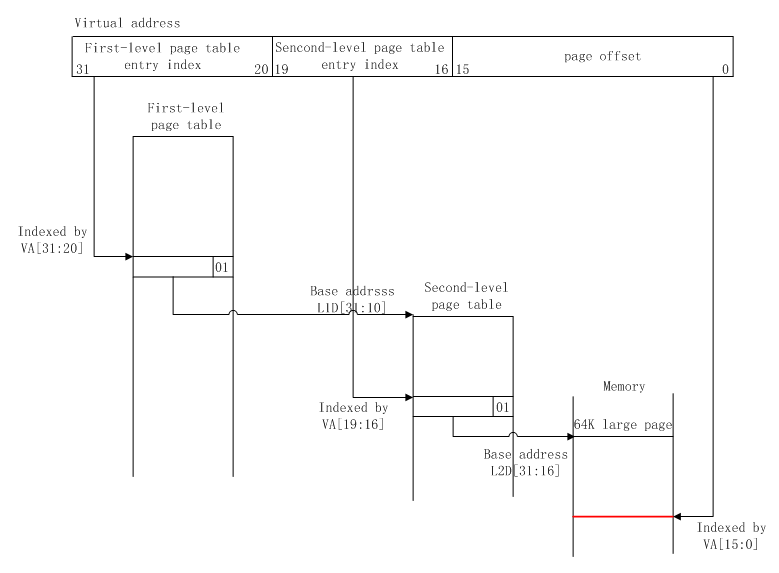
二级页表描述符的[31:12]位存放该虚拟地址在内存中的物理页基址；

物理页基址+ VA[11:0]页偏移= 物理地址

由映射图可知，一个虚拟地址可以索引2^12个一级页表入口，每个一级页表入口指向的二级页表最大可以有2^8个二级页表入口，每个二级页表入口映射2^12大小的内存，故虚拟地址可以映射的最大物理内存为：2^12 \* 2^8 \* 2^12 ，即4G。

**2.4 页映射，映射粒度为64K**

当映射方式为页映射，且映射粒度为64K时，映射图如下：



虚拟地址到物理地址的映射过程如下：

虚拟地址的[31:20]位存放一级页表的入口index，[19:16]位存放二级页表的入口index，[15:0]位存放页偏移；

从TTBR（translation table base register，协处理器CP15中的一个寄存器，用于存放一级页表的基址）寄存器中获取一级页表的基址；

一级页表基址+ VA[31:20] = 一级页表描述符的入口地址；

一级页表描述符的[31:10]位存放二级页表的基址；

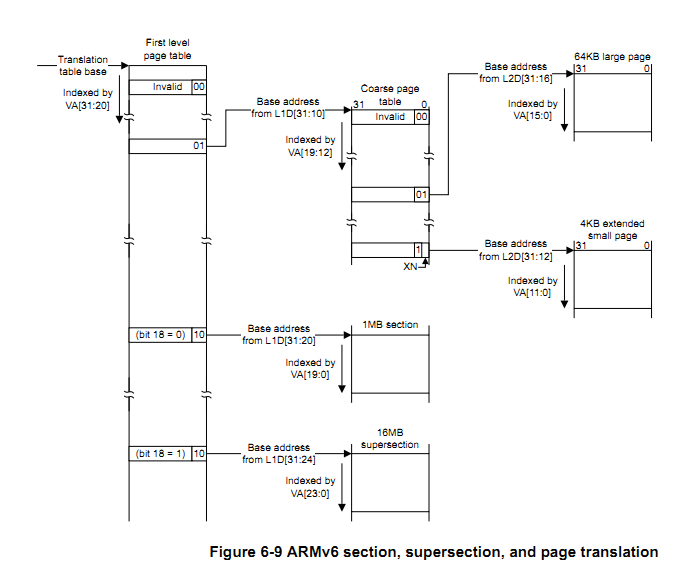
二级页表基址+ VA[19:16] = 二级页表描述符的入口地址；

二级页表描述符的[31:16]位存放该虚拟地址在内存中的物理页基址；

物理页基址+ VA[15:0]页偏移= 物理地址

由映射图可知，一个虚拟地址可以索引2^12个一级页表入口，每个一级页表入口指向的二级页表最大可以有2^4个二级页表入口，每个二级页表入口映射2^16大小的内存，故虚拟地址可以映射的最大物理内存为：2^12 \* 2^4 \* 2^16 ，即4G。

**2.5 地址映射总图**：



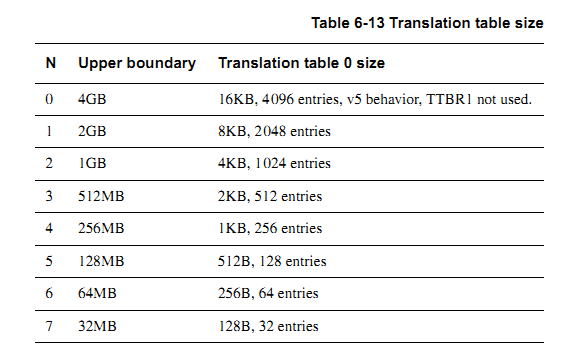
**3. 关于一级页表基址**

参考《ARM1176 JZF-S Technical Reference Manual》6.12 MMU descriptors

ARMv6中有两个协处理器寄存器用来存放一级页表基地址，TTBR0和TTBR1。操作系统把虚拟内存划分为内核空间和用户空间，TTBR0存放用户空间的一级页表基址，TTBR1存放内核空间的一级页表基址。

N的大小由TTBCR寄存器决定。0x0 -> 1<<(32-N)为用户空间，由TTBR0控制，1<<(32-N) -> 4GB为内核空间，由TTBR1控制。

N的大小与一级页表大小的关系图如下：



操作系统为用户空间的每个进程分配各自的页表，即每个进程的一级页表基址是不一样的，故当发生进程上下文切换时，TTBR0需要被存放当前进程的一级页表基址；TTBR1中存放的是内核空间的一级页表基址，内核空间的一级页表基址是固定的，故TTBR1中的基址值不需要改变。

**4.  u-boot中MMU初始化代码分析**

u-boot中的MMU地址映射方式为段映射，映射粒度为1M，只用到一级页表。

start.S中的MMU初始化代码如下：

[view source](http://www.it165.net/embed/html/201408/2633.html#viewSource)[print](http://www.it165.net/embed/html/201408/2633.html#printSource)[?](http://www.it165.net/embed/html/201408/2633.html#about)

01.#ifdef CONFIG\_ENABLE\_MMU

02.enable\_mmu:

03./\* enable domain access \*/

04.ldr    r5, =0x0000ffff

05.mcr    p15, 0, r5, c3, c0, 0        @ load domain access register

06.

07./\* Set the TTB register \*/

08.ldr    r0, \_mmu\_table\_base

09.ldr    r1, =CFG\_PHY\_UBOOT\_BASE

10.ldr    r2, =0xfff00000

11.bic    r0, r0, r2

12.orr    r1, r0, r1

13.mcr    p15, 0, r1, c2, c0, 0

14.

15./\* Enable the MMU \*/

16.mmu\_on:

17.mrc    p15, 0, r0, c1, c0, 0

18.orr    r0, r0, #1            /\* Set CR\_M to enable MMU \*/

19.mcr    p15, 0, r0, c1, c0, 0

20.nop

21.nop

22.nop

23.nop

24.#endif

25.…

26.…

27.…

28.#ifdef CONFIG\_ENABLE\_MMU

29.\_mmu\_table\_base:

30..<a href="<http://www.it165.net/edu/ebg/>" target="\_blank" class="keylink">word</a> mmu\_table

31.#endif

对协处理器的寄存器操作参考：

《ARM1176 JZF-S Technical Reference Manual》Chapter 3 System Control Coprocessor

MMU初始化过程中有一步是将页表基址（CFG\_PHY\_UBOOT\_BASE + mmu\_table）存入TTBR0中，在lowlevel\_init.S中可以看到对页表的初始化：

[view source](http://www.it165.net/embed/html/201408/2633.html#viewSource)[print](http://www.it165.net/embed/html/201408/2633.html#printSource)[?](http://www.it165.net/embed/html/201408/2633.html#about)

01.#ifdef CONFIG\_ENABLE\_MMU

02.

03./\*

04.\* MMU Table for SMDK6400

05.\*/

06.

07./\* form a first-level section entry \*/

08..macro FL\_SECTION\_ENTRY base,ap,d,c,b

09..<a href="<http://www.it165.net/edu/ebg/>" target="\_blank" class="keylink">word</a> (ase << 20) | (ap << 10) |

10.(d << 5) | (1<<4) | (c << 3) | (<< 2) | (1<<1)

11..endm

12..section .mmudata, 'a'

13..align 14

14.// the following alignment creates the mmu table at address 0x4000.

15..globl mmu\_table

16.mmu\_table:

17..set \_\_base,0

18.// 1:1 mapping for debugging

19..rept 0xA00

20.FL\_SECTION\_ENTRY \_\_base,3,0,0,0

21..set \_\_base,\_\_base+1

22..endr

23.

24.// access is not allowed.

25..rept 0xC00 - 0xA00

26..word 0x00000000

27..endr

28.

29.// 128MB for SDRAM 0xC0000000 -> 0x50000000

30..set \_\_base, 0x500

31..rept 0xC80 - 0xC00

32.FL\_SECTION\_ENTRY \_\_base,3,0,1,1

33..set \_\_base,\_\_base+1

34..endr

35.

36.// access is not allowed.

37..rept 0x1000 - 0xc80

38..word 0x00000000

39..endr

40.

41.#endif

下面对页表的初始化代码作详细解释：

[view source](http://www.it165.net/embed/html/201408/2633.html#viewSource)[print](http://www.it165.net/embed/html/201408/2633.html#printSource)[?](http://www.it165.net/embed/html/201408/2633.html#about)

1./\* form a first-level section entry \*/

2..macro FL\_SECTION\_ENTRY base,ap,d,c,b

3..word (ase << 20) | (ap << 10) |

4.(d << 5) | (1<<4) | (c << 3) | (<< 2) | (1<<1)

5..endm

定义一个宏FL\_SECTION\_ENTRY用来设置页表入口描述符，base即物理基址，ap即access permission，d即domain，c即cacheable，b即bufferable。

内存访问控制和段属性相关描述请参考：

《ARM1176 JZF-S Technical Reference Manual》6.6 Memory access control和6.7 Memory region attributes。

[view source](http://www.it165.net/embed/html/201408/2633.html#viewSource)[print](http://www.it165.net/embed/html/201408/2633.html#printSource)[?](http://www.it165.net/embed/html/201408/2633.html#about)

1..section .mmudata, 'a'

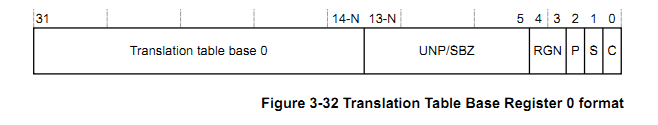
2..align 14

3.// the following alignment creates the mmu table at address 0x4000.

定义一个名为mmudata的段，段属性为“a”，allowable，该段16K对齐。从u-boot.lds中可以看到，u-boot的各个段在内存中的分布依次为：.text，.rodata，.data，.got，.u\_boot\_cmd，.mmudata，.bss。

为什么页表是16K对齐呢？

在上一节我们讲过：有两个寄存器TTBR0和TTBR1用来存放一级页表基址，操作系统把虚拟地址空间划分为用户空间和内核空间，0x0 -> 1<<(32-N)为用户空间，由TTBR0控制，1<<(32-N) -> 4GB为内核空间，由TTBR1控制，N的大小由TTBCR寄存器决定。由于u-boot主要作用是硬件初始化和引导操作系统，所以没有必要对虚拟地址空间进行划分，即N=0，整个虚拟地址空间由TTBR0控制，TTBR0的格式如下：



N=0时，[31:14]存放页表基址，即一级页表的基址为([31:14]<<14)，2^14为16K。

[view source](http://www.it165.net/embed/html/201408/2633.html#viewSource)[print](http://www.it165.net/embed/html/201408/2633.html#printSource)[?](http://www.it165.net/embed/html/201408/2633.html#about)

1..set \_\_base,0

2.// 1:1 mapping for debugging

3..rept 0xA00

4.FL\_SECTION\_ENTRY \_\_base,3,0,0,0

5..set \_\_base,\_\_base+1

6..endr

对虚拟地址0x0-0xA0000000作平行映射（flat mapping），即把虚拟地址0x0-0xA0000000映射到物理地址0x0-0xA0000000。

[view source](http://www.it165.net/embed/html/201408/2633.html#viewSource)[print](http://www.it165.net/embed/html/201408/2633.html#printSource)[?](http://www.it165.net/embed/html/201408/2633.html#about)

1.// access is not allowed.

2..rept 0xC00 - 0xA00

3..word 0x00000000

4..endr

不对虚拟地址空间0xA0000000-0xC0000000作映射，即禁止访问虚拟地址空间0xA0000000-0xC0000000。

[view source](http://www.it165.net/embed/html/201408/2633.html#viewSource)[print](http://www.it165.net/embed/html/201408/2633.html#printSource)[?](http://www.it165.net/embed/html/201408/2633.html#about)

1.// 128MB for SDRAM 0xC0000000 -> 0x50000000

2..set \_\_base, 0x500

3..rept 0xC80 - 0xC00

4.FL\_SECTION\_ENTRY \_\_base,3,0,1,1

5..set \_\_base,\_\_base+1

6..endr

把虚拟地址空间0xC0000000-0xC8000000映射到物理地址空间0x50000000-0x58000000，0x50000000-0x58000000为sdram的地址空间，此时sdram有128M。

[view source](http://www.it165.net/embed/html/201408/2633.html#viewSource)[print](http://www.it165.net/embed/html/201408/2633.html#printSource)[?](http://www.it165.net/embed/html/201408/2633.html#about)

1.// access is not allowed.

2..rept 0x1000 - 0xc80

3..word 0x00000000

4..endr

不对虚拟地址空间0xc8000000-0xffffffff作映射，即禁止访问虚拟地址空间0xc8000000-0xffffffff。