# 专题11-内存管理单元（MMU）

## 一、MMU功能解析

### 1.1、MMU作用

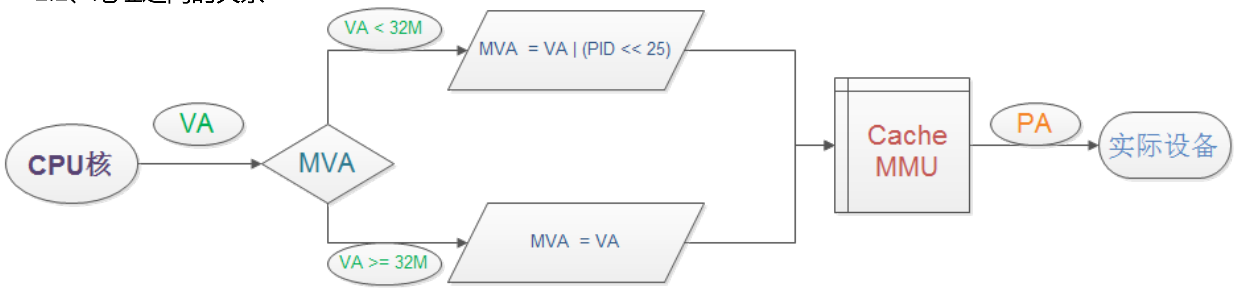
MMU负责虚拟地址到物理地址的映射，并提供硬件机制的内存访问权限检查。

## 二、深入剖析地址转化

### 2.1、地址概念

虚拟地址（VA，Virtual Address）　变换后的虚拟地址（MVA，Modified Virtual Address）　物理地址（PA，Physical Address）

### 2.2、地址之间的关系



利用PID生成MVA的目的是为了减少切换进程时的代价。后面提及的虚拟地址，一般是指MVA。

### 2.3、虚拟地址到物理地址的转换过程

**Backwards-compatible page table format**

Figure 6-4 shows a backwards-compatible format first-level descriptor.

| | 31–24 | 23–20 | 19 | 18 | 17–15 | 14–12 | 11–10 | 9 | 8–5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Translation fault | Ignored | | | | | | | | | | | | 0 | 0 |
| Coarse page table | Coarse page table base address | | | | | | | P | Domain | SBZ | NS | SBZ | 0 | 1 |
| Section (1MB) | Section base address | | NS | 0 | SBZ | TEX | AP | P | Domain | 0 | C | B | 1 | 0 |
| Supersection (16MB) | Supersection base address | SBZ | NS | 1 | SBZ | TEX | AP | P | Ignored | 0 | C | B | 1 | 0 |
| Reserved | | | | | | | | | | | | | 1 | 1 |

**Figure 6-4 Backwards-compatible first-level descriptor format**

BIT[1~0] 用于确认转换方式：转换错误，页式转换，段式转换，超级段式转换。

| | 31–16 | 15–12 | 11 | 10 | 9 | 8 | 7 | 6 | 5–4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Translation fault | Ignored | | | | | | | | | | | 0 | 0 |
| Large page (64KB) | Large page base address | TEX | AP3 | | AP2 | | AP1 | | AP0 | C | B | 0 | 1 |
| Small page (4KB) | Small page base address | | AP3 | | AP2 | | AP1 | | AP0 | C | B | 1 | 0 |
| Extended small page (4KB) | Extended small page base address | | SBZ | | | TEX | | | AP | C | B | 1 | 1 |

**Figure 6-5 Backwards-compatible second-level descriptor format**

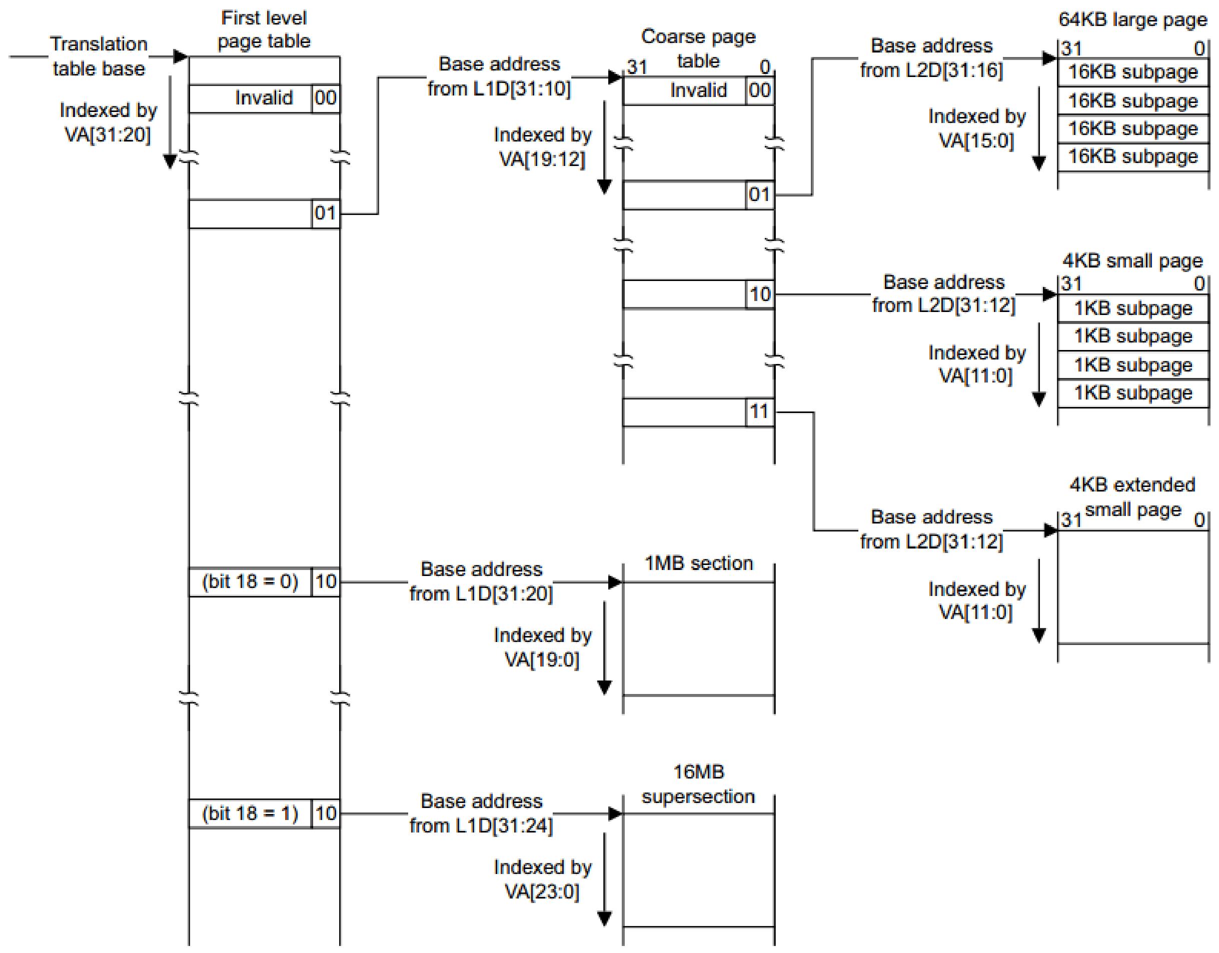


**Figure 6-6 Backwards-compatible section, supersection, and page translation**

TTB base（Translation table base）：代表一级页表的地址，需要将它写入协处理器CP15的寄存器C2（称为页表基址寄存器）。

TLB（Translation lookaside Buffer）：即转换旁路缓存，TLB是MMU的核心部件。它缓存少量的虚拟地址和物理地址的转换关系，是转换表的Cache，因此常成为“快表”。

TTW（Translation Table walk）：即转换表漫游，当TLB中没有缓冲对应的地址转换关系时，需要通过对内存中转换表的访问来获得虚拟地址和物理地址的对应关系。TTW成功后，结果应写入TLB。

MMU实际上就是根据建立好的虚拟地址和物理地址的映射表，然后将TTB写入到CP15寄存器。MMU的配置关键在于映射表的建立。各级表的条目内容配置。最好是使用U-BOOT提供的实现方式。

## 三、MMU配置与使用

使用最简单的段式转换来配置MMU，段式转换每一段的大小为1MB（0x100000），外设操作的0x7F000000映射到0xA0000000，内存映射到原本的位置。

### 3.1、建立一级页表

### 3.2、写入TTB

```
#define GPMCON ((volatile unsigned long *) 0xA0008820)
#define GPMDAT ((volatile unsigned long *) 0xA0008824)
#define GPMPUD ((volatile unsigned long *) 0x7f008828)

#define MMU_SECTION      (2 << 0)
#define MMU_CACHEABLE    (1 << 3)
#define MMU_BUFFERABLE   (1 << 2)
#define MMU_SPECIAL      (1 << 4)
#define MMU_DOMAIN       (0 << 5)
#define MMU_MULL_ACCESS (3 << 10)
#define MMU_SECDESC      (MMU_MULL_ACCESS | MMU_DOMAIN | MMU_SPECIAL | MMU_SECTION)
#define MMU_SECDESC_WB   (MMU_MULL_ACCESS | MMU_DOMAIN | MMU_SPECIAL | MMU_CACHEABLE 
| MMU_BUFFERABLE | MMU_SECTION)


void creat_page_table(void)
{
        unsigned long *ttb = (unsigned long *)0x50000000;

        unsigned long vaddr, paddr;

        vaddr = 0xA0000000;
        paddr = 0x7F000000;

        *(ttb + (vaddr >> 20)) = (paddr & 0xfff00000) | MMU_SECDESC;

        vaddr = 0x50000000;
        paddr = 0x50000000;

        while (vaddr < 0x54000000) {
                *(ttb + (vaddr >> 20)) = (paddr & 0xfff00000) | MMU_SECDESC_WB;
                vaddr +=0x100000;
                paddr +=0x100000;
        }


}

```

3.3、打开MMU

```
void mmu_init(void)
{
        __asm__ volatile (
                "ldr r0, =0x50000000\n"
                "mcr p15, 0, r0, c2, c0, 0\n"

                "mvn r0, #0\n"
                "mcr p15, 0, r0, c3, c0, 0\n"

                "mrc p15, 0, r0, c1, c0, 0\n"
                "orr r0, r0, #0x0001\n"
                "mcr p15, 0, r0, c1, c0, 0\n"
                :
                :

        );
}

int uboot_main()
{
        //1.build table

        //2.write TTB
        creat_page_table();

        //3.enable MMU
        mmu_init();


        *(GPMCON) = 0x00001111;
        *(GPMDAT) = 0b1000;

        return 0;
}
```