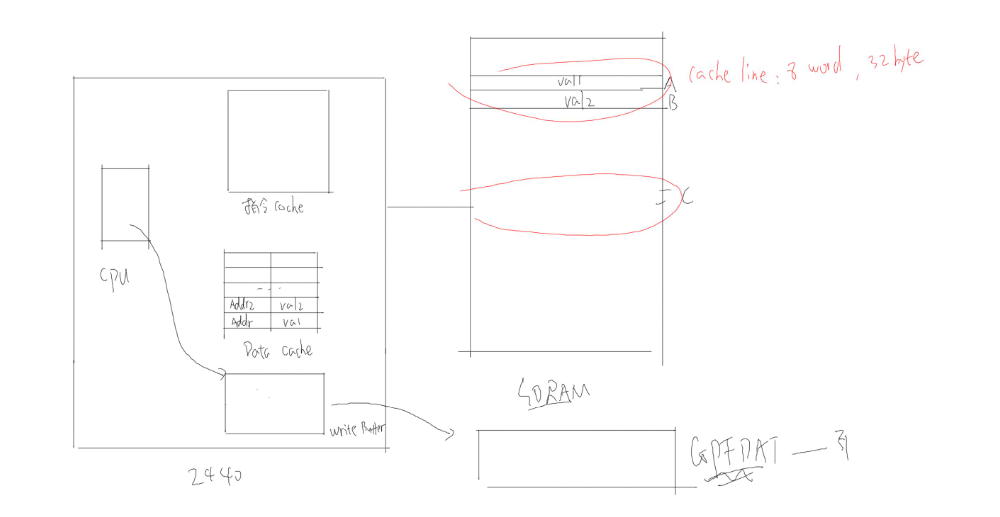
程序的时间局部性：在同一段时间里有极大的可能访问同一地址的数据或指令

在同一段时间里，

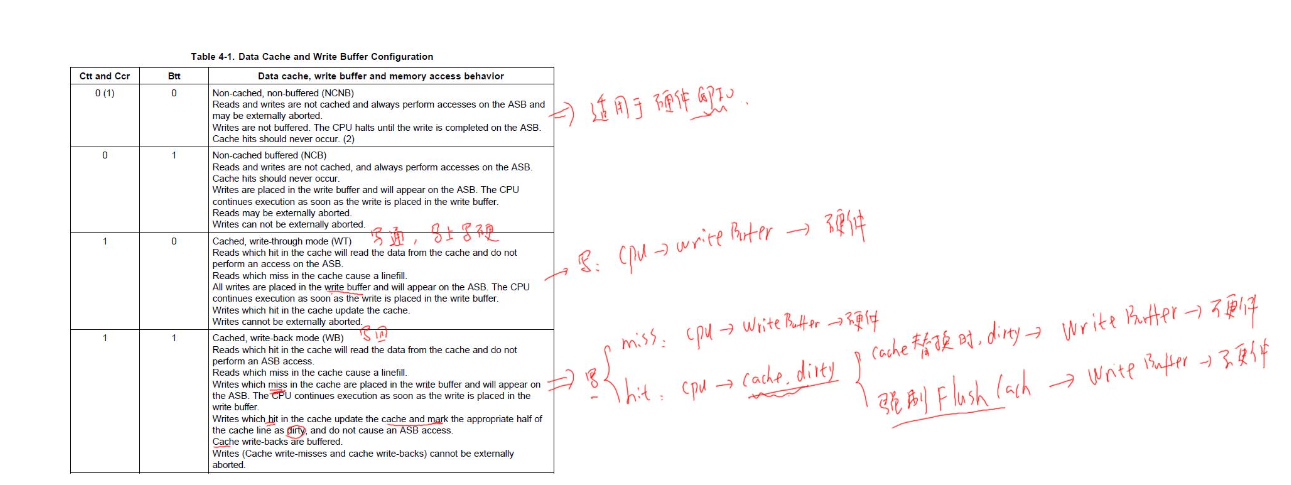
空间局部性：程序执行的过程中有极大可能和概率访问相邻空间的指令和数据

所以引入cache将内存的数据或者指令读到cache中。

但是cache 一般比内存要小：



s3c2410:



假如cpu要读地址A的数据: Ldr r0 A

一：

1. 首先以地址A在cache查找数据。一开始cache 无数据，称为cache miss
2. Cpu 把地址传给sdram.并且读入cache line （并将a的地址返回给cpu）cache fill

arm9 的cache line 8 word 32 字节

3程序读取地址A的数据，cpu已地址A在cache查找到，称为cache命中

假如去读地址B的数据根据程序的局部性，程序极大的可能去读相邻地址的数据，在cache中能快速找到。

1. cache(16kb) 假如访问c地址。cache 满了之后会将老的数据置换出来，会将C周围的地址数据读出来

除了有 数据cache 指令cache 还有write buffer.

1可以设置为no cache no buffer,直接去读硬件，因为cache他是老的数据.（适用于硬件的GPIO）

2 Cpu可以将数据写给write buffer,然后让write buffer 去操作硬件

在 1 1 情况下 写操作：

如果cache miss cpu 将数据送到write buffer ---送到硬件

如果cache hit cahe.dirty 将数据标记位脏的 ，在合适的时机(cahce 替换时)将数据写给write buffr -- 送到硬件

2.协处理器指令

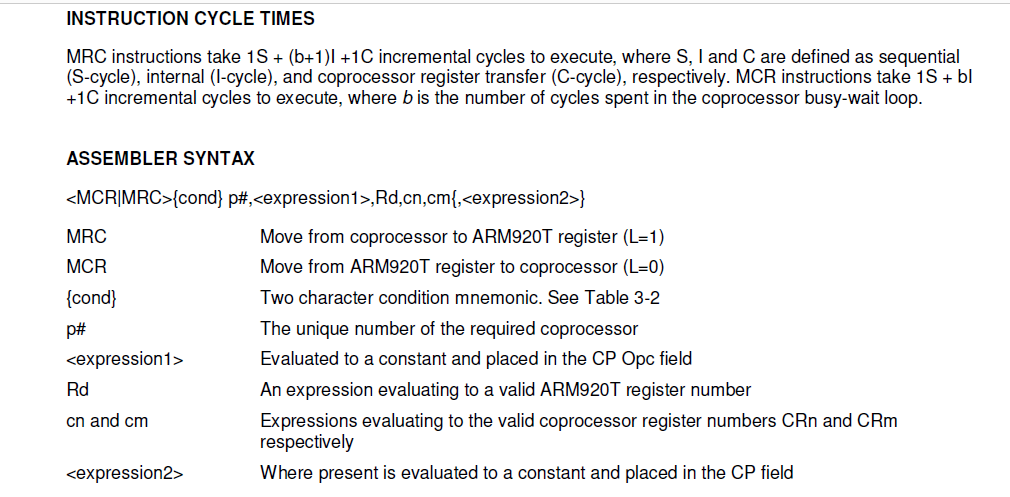
Arm9有cp0 –cp15 16个协处理器。

协处理器里面也有很多的寄存器：

mrc : 也就是mov egister oprocessor 将协处理器的指令放到主cpu

mcr也就是mov oprocessor register 将主cpu 放到 协处理器

协处理器指令格式：



Mcr p15, 0,(expression1) r1, c1,c0,(cm 用不着写c0) 0 (expression2)// 将主寄存器r1写入协处理器的cp15的c1

//c0 0一般用来区分呢一个c1,一般写为c0 0

2.Mrc p15 ,0 r1,c1,c0,0 //(将协处理器的cp15的c1写入主寄存器r1

启用协处理器可以加快cpu的执行效率：(参考2410手册)

1. Mmu地址映射：

S3c2440 sdram 64M

假如有多个app同时运行。同时保存在SDRAM。

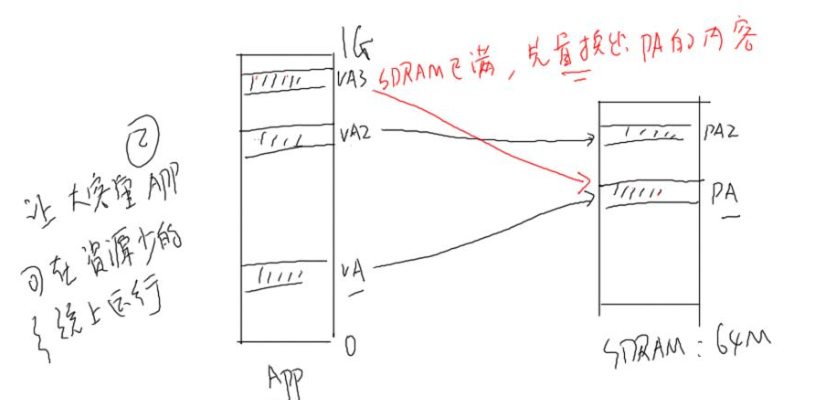
链接地址：程序运行时所处的地址，有多个app 他们所处的地址不同，所以编译某个app时需要单独指定链接地址。这几乎不可能

在多个App的时候不能从西编译，也不可能指定地址

裸机的编译： arm –linux-gcc -o 文件名 .c -static

arm –linux-objdump -D 文件名 > 文件.dis

假如有两个.c 生成的反汇编代码可以看出他执行代码的地址相同。其实是同一个地址然后根据mmu的映射去sdram不同的地址查找对应的地址。



2.假如一个代码需要内存1G,但是sdram 64M,因为代码执行肯定有跳转。所以执行到哪里就去映射地址到sdram上，如果满了置换出以前的空间使的大app可以在资源小的系统上运行。

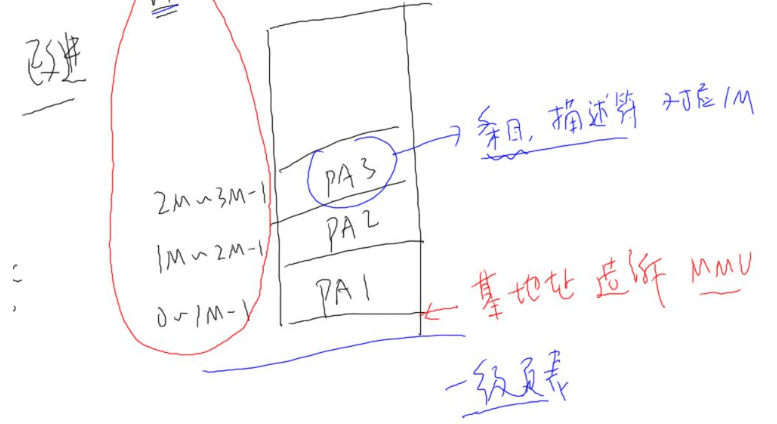
所以 cpu –>mmu –>内存控制器—( sdram gpio lcd)

3.权限管理

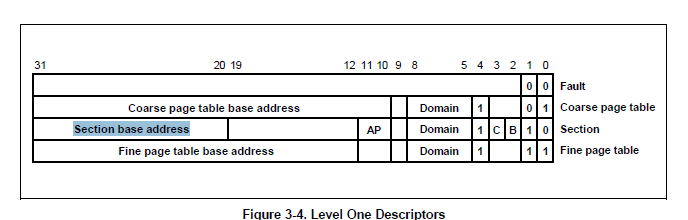
禁止app访问其他空间

Cpu ->虚拟地址到->mmu -> 到物理地址.

Mmu如何去查找虚拟地址到对应的物理地址?如下图：



1. 在内存中创建页表（如上图的一级页表） 页表中对应相应的条目（对应1m）如果向对应更小的页表就引入二级页表
2. 把页表的基地址告诉给mmu
3. 设置cp15,启动mmu



Section base address PA(物理地址)

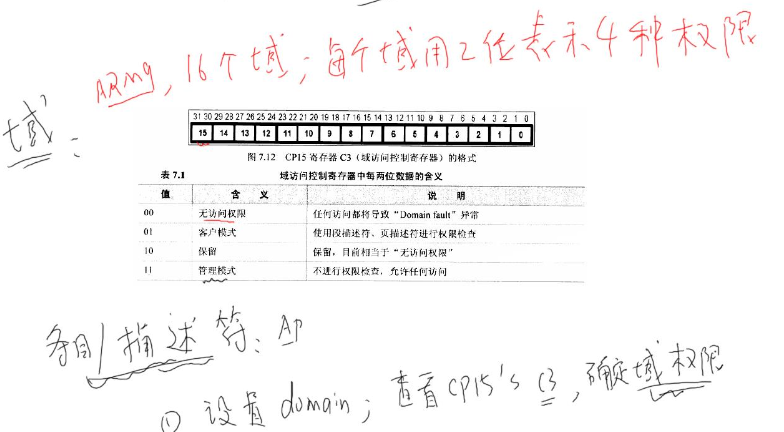
Ap Domain C B 都是用来进行权限管理的。

权限管理：

1 完全不允许访问

2 允许系统模式访问 ，不允许用户模式访问

3允许用户模式下根据描述符（条目）中的ap决定。

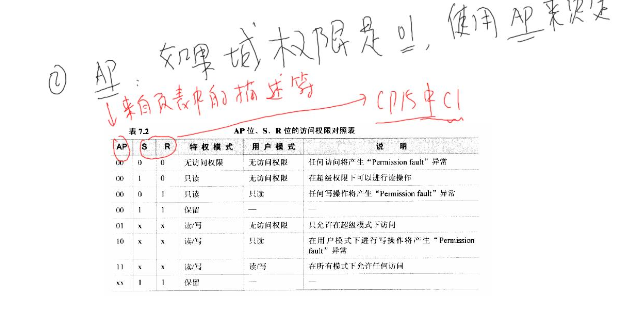


Domain:

假如域15 是00 这块地址就不能访问

1设置domain 查看cp15 c3 确定域权限

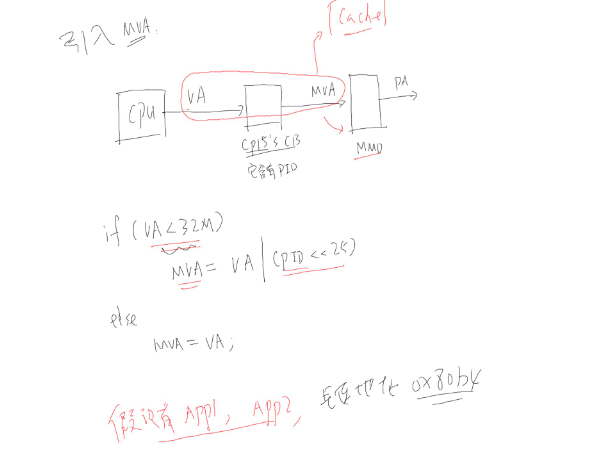
2.ap假如域权限是01 使用ap来决定



假如进行进程的切换时，需要将虚拟地址从新映射，也就是要从新修改页表。使得开销非常大。

引入mva（修改后的虚拟地址modify ）:

如下图cpu 发出虚拟地址(VA) 到协处理器cp15 它含有进程pid,经过mva 到mmu去查找pa(物理地址)



假如有app1 ,pid 1 。 app2 ,pid2 ,链接地址都是0x80B4

运行app1时发出VA MUA = VA | (1 << 25), 对应的页表项app1对应的内存

运行app2时发出VA MUA = VA | (2 << 25) 对应的页表项 app2对应的内存

进程在切换的时候，根据pid的不同对应不同地址不用进行创建页表

