# 文档

1. **entry.asm汇编代码分析**

lui t0, %hi(boot\_page\_table) //加载高20位?

li t1, 0xffffffff00000000 //加载立即数

sub t0, t0, t1 //t0-t1 目的是减去高位的ffffffff

srli t0, t0, 12 //将t0左移12位

li t1, (8 << 60) //t1的Mode位置为8 也就是Sv39

or t0, t0, t1 //改变t0的Mode

csrw satp, t0 //将t0写入stap

sfence.vma //更新TLB

boot\_page\_table:

.quad 0

.quad 0

# 第 2 项：0x8000\_0000 -> 0x8000\_0000，0xcf 表示 VRWXAD 均为 1

.quad (0x80000 << 10) | 0xcf //0xcf=>11001111

.zero 507 \* 8

# 第 510 项：0xffff\_ffff\_8000\_0000 -> 0x8000\_0000，0xcf 表示 VRWXAD 均为 1

.quad (0x80000 << 10) | 0xcf

.quad 0

**这个根页表的建立到底是在干什么?看不明白**

可以理解为分配空间

[0][0][x][…][x][0] 这个页表的大致结构是这样

在这两个页表项中 rwx均为1 也就是说明这两个页表项各自代表一个大页

在linker.ld中

BASE\_ADDRESS = 0xffffffff80200000; /\* 修改为虚拟地址 \*/

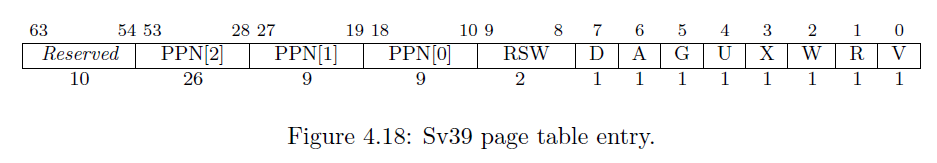
但是opensbi默认还是从0x80200000开始访问

所以我们需要将访问的地址映射过去

为什么510项是vpn3? 还是没有理解

1. **实现页表**

**首先我们需要建立页表项**



fn new=>{

如果物理地址合法 则页表项V置为1

Self的定义=>这里的\*0我没有弄懂它在表示什么，假定为设置页表项地址

此时的页表项映射到传递过来的页面

}

update\_page\_number与new类似 用来更新页号

**建立页表**

在建立页表之前，我们需要先建立一个指向页表的指针

PageTableTracker-> new

建立一个空的页表 我们已经在address.rs实现虚实地址的解引用，并实现了PageTableTracker->PageTable的转变

在这里我们形成了一个简单的映射 可以为后面的做铺垫作用

1. **内核重映射**

我们的目的是要讲映射精细化，使得所有内存段的权限是正确的

**?**操作符是为了省去match

root\_ppn=初始物理地址 self.start\_ppn + offset 此时offset=0

find\_entry

首先将root\_ppn物理地址转换为虚拟地址

获取entry 找到二级页表的物理页号

利用一个循环去寻找对应的物理页表 然后到一级页表偏移

最终找到对应的物理页

map\_one

匹配是虚拟页面与物理页面是否存在关系，不存在则创建页面

map

如果映射片段类型是Linear 则将对应片段内已有的虚拟页面与物理页面建立关系，并且将数据保存下。

如果类型是Framed，则判断数据是否初始化，如果未初始化，则返回初始化数据。如果已有初始化，拷贝值时需要考虑区间对齐，如果仅有部分重合,则需要考虑在开始区间，结束区间将数据对齐，以便于后面的页面连续，将数据拷贝。

利用汇编更换stap的值,并刷新TLB

最后将内核段写入Mapping，根据各个段分配属性

# 实验题

实验三中的汇编代码有些不理解，修改内核的思考题也是一头雾水,汇编值得认真学习一下

**实验题一**

boot\_page\_table 的意义是，将之间内核启动时的Bare模式修改为sv39模式，从而开启页表机制，通过t0找到了rust\_main

**实验题二**

page\_tables 是保存了访问到的页表，而mapped\_pairs 是为了让进程能够看到自己的地址，所以里面保存的是进程的页表。

**实验题三**

不会访问到B。访问B是可以的，人为的将地址指向这个这个物理地址。