实验思考题

2.1

- 总体来说，cache与TLB的功能定位相仿，都是利用局部性原理来达到**尽可能不访问大容量目标**而从一个**缓存区直接得到目标**这个目标的工具。cache中存放的是物理存储器中真实的数据，因此cache才必须使用物理地址进行查询。
- 假设cache使用虚存地址，也就是说，cache接受虚拟地址然后给出一个真实数据，那么cache中存放的应是虚拟地址和真实数据。cache命中缺失的时候，则通过这个虚存地址经由MMU直接访问真实的主存，遂获取真实数据。宏观来看，CPU一次访存的经历由**TLB-(pagetable-(exception)*)*-cache-(MEM)***变成了**cache-(TLB-(pagetable-(exception)*)*-MEM)***。
- 相比于物理地址的方法，这个新方法在cache命中时**仅需要**访问一次cache，优于原方法的最佳情况。但是，一般来说虚拟地址数远大于物理地址数(比如实验课中的32位-64M和现在主流的64位-8G)。如果64位操作系统cache使用虚拟地址，那cache的负担远远大于全命中时带来的一点收益。如果再仿照MMU处理64位的方法(多级)而设置5级甚至6级cache，那功耗将更加可怕。
- 更重要的一点是，cache不仅仅向页表那样一旦建立就几乎是只读的，他要经常写数据。设想这两个例子：
    - 假如两个虚拟地址映射到同一个物理地址，都在cache中，都被写过，那么回填主存时会引起冲突。
    - 假如A程序写了一个虚存地址，B程序需要这个修改后的值，但A和B对于这一块内存的虚拟地址被各自的页表映射到了不同的虚拟地址上

  为了解决冲突只能加入类似脏位的修饰，但修饰了一个虚拟地址之后，还要遍历所有同映射一个物理地址的虚拟地址，并在cache中都标记，这样的过程非常低效。

2.2

- 首先，注意到两个事实：
    - 虚实转换以页为单位，后面的offset不变
    - 访问cache时在一个块内也需要查找offset(尤其是组相联时组数很多)
- 因此，可以并行的执行以上两个步骤
- 此外，仿照乘除槽的特性，TLBmiss时直接按照offset去cache中寻找数据，同时查询页表。将存在冒险的指令拦截，而让其他指令照常执行

2.3

- 虚拟地址
- 理由：在程序中打印某个变量的地址，出现的是32位数，而不是与64M对应的26位数

2.4

- 编译器可以很有效的识别do{}while();语句，因此，do与while紧紧地把中间的代码段框了起来，不与外界产生可能的语义混淆，如果外面语句有语法错误，编译器可以很直接的识别出来，而不会将其与这个宏一起识别，导致崩溃，例如：

```
while(3)
LIST_INSERT_BEFORE();
```

这样的语句表面来看显然是非法(因为while后面没有写括号)的，但如果宏函数仅写了代码段，就会恰好变成合法语句，鲁棒性较差

2.5

- 物理内存在哪里：
  - 对于真实的MIPS系统，物理内存就是内存条，用接口与计算机进行交互
  - 对于实验，物理内存由gxemul所模拟，是linux系统中的一块虚拟内存
  - 通过阅读代码，物理内存直接对应到了逻辑地址的0x8000_0000 ~ 0x8400_0000;
- 如何获取：
  - 通过page2pa函数，找到page对应的物理内存
  - 通过pa2page函数，找到物理内存对应的page

2.6

- C，因为pp_link不是指针，但lh_first是指针

2.7

- b是一个虚拟地址，因为这里是分配一页内存时进行的清零操作，bzero函数是连续清零的操作，每一页的虚拟地址是连续的，物理地址不连续，因此b是虚拟地址

2.8

- (C0000000 >> 12)*4 = 00300000,所以起始目录为0xC0300000

2.9

- tlb_p是寻找相应项的索引，tlb_wi是根据p指令写下的index来写tlb的项，如果index小于0，说明tlb_p没有找到，则miss
- 因为MIPS是我阶段流水线，tlb_p不希望打扰别人，所以后面跟了nop伪造单周期效果

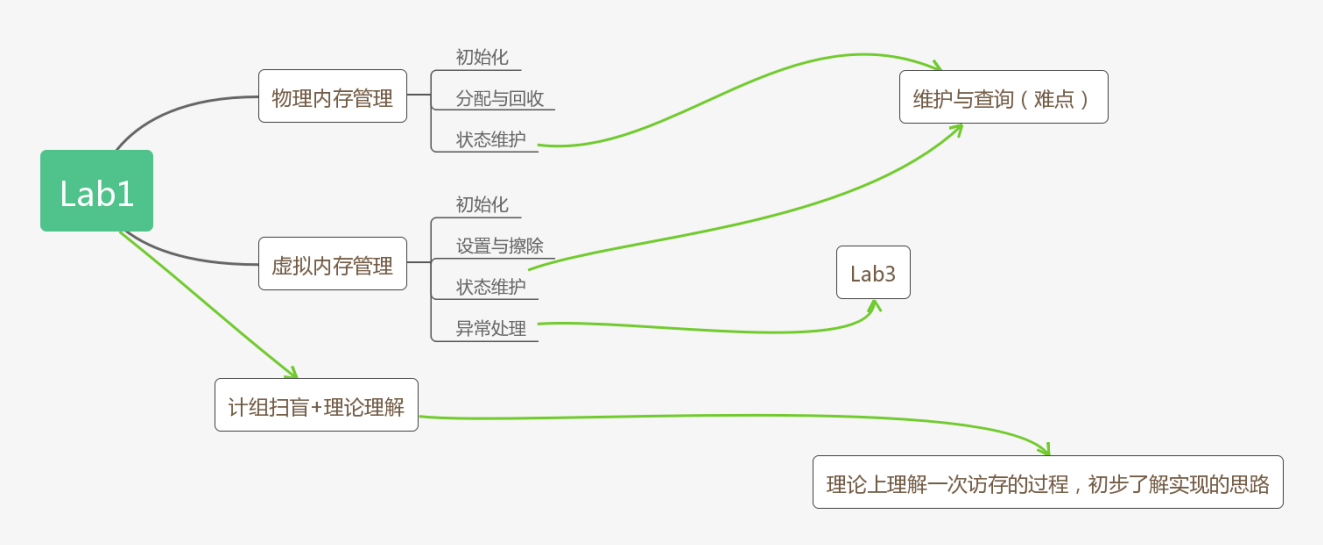
2.10

- pa是真实物理地址，但不在kseg0/1，不能直接+Ulim访问，需要走MMU

2.11

- PSE为0时页面大小为4KB，PSE为1时页面为4MB，这是最直观的区别。
- 可以直接从页目录（换句话说，一级页表）找到对应页面内，节约了TLB
- 总体来看，PSE为1时类似一个一级页表

难点



体会与感想

难度评价

- 总体难度较高，但理解通顺后还可以
- 相比于理论，实践操作内存管理更加困难。对于某些题目，在刚刚读完指导书后，根本无从下手。只有参阅了网络上的代码和阅读了相关资料后，才能实现

花费时间

写代码大概2小时，理解1周+

指导书反馈

无

残留难点

与OO或计组的**轻易可实践**不同，OS在自行探索方面不仅环境不友好（怕误操作），而且debug方式单一（断点+输出），缺少**刷题**的机会。难度与计组的流水线相仿，是关键但不困难的一步。所谓不困难，是因为计组课下可以开脑洞给自己出几十道题练手，而OS并没有这样的机会。