Lab2 实验报告

一、思考题

Thinking 2.1

请思考 cache 用虚拟地址来查询的可能性，并且给出这种方式对访存带来的好处和坏处。另外，你能否能根据前一个问题的解答来得出用物理地址来查询的优势？

使用虚拟地址访问 cache

优点：不需要经过 mmu 可以直接访问

缺点：（1）会出现虚地址访问 cache 冲突的情况

（2）如果 cache 直接使用虚拟地址访问，就绕过了页表机制，使用流水线中指令的执行不再存在保护

（3）现代操作系统允许有多个虚拟页面空间映射到同一个物理地址页面空间，存在读写不一致的情况

（4）现在外部 IO 设备的地址映射一般是物理地址映射；而我们有 TLB 把虚拟地址映射为物理地址，但是在读写外部 IO 设备时，我们是没有机制把物理地址映射为虚拟地址的。

物理地址查询的优势：物理地址是单一的，以上缺点都不会存在

Thinking 2.2

请查阅相关资料，针对我们提出的疑问，给出一个上述流程的优化版本，新的版本需要有更快的访存效率。（提示：考虑并行执行某些步骤）

虚实地址结合，虚拟地址访问 cache，物理地址比较 Tag。利用了页表的虚拟地址和物理地址其实在低位是相同的，不同的是高位映射。

而虚实结合的 cache 访问方式，其模式如下：

节拍-1：利用虚拟地址访问 TLB，进行虚实地址转换；同时在这个周期中，利用虚拟地址的低位访问 cache，取出 cache 中存储的 TAG 信息和 data 信息，但注意，这个时候的 TAG 信息其实对应的是物理地址的 TAG；

节拍-2：假设 TLB 和 cache 都命中，则利用上个节拍得到的物理地址进行 TAG 比较，并选择合适的 cache data 数据。

使用物理地址 cache 方式的节拍需要 3 个周期得到需要的指令；这种模式下，仅需要 2 个时钟周期就能得到所需要的 cache 数据。

Thinking 2.3

在我们的实验中，有许多对虚拟地址或者物理地址操作的宏函数（详见 include/mmu.h ），那么我们在调用这些宏的时候需要弄清楚需要操作的地址是物理地址还是虚拟地址，阅读下面的代码，指出 x 是一个物理地址还是虚拟地址。

```
int x;
char *value = return_a_pointer();
*value = 10;
x = (int) value;
```

虚拟地址，经过 mmu 之前所操作的地址均为虚拟地址

Thinking 2.4

我们注意到我们把宏函数的函数体写成了 |do /* ... */while(0)| 的形式，而不是仅仅写成形如 | /* ... */ | 的语句块，这样的写法好处是什么？
使用 do{...}while(0)构造的宏定义不会受到大括号、分号等的干扰

Thinking 2.5
注意，我们定义的 Page 结构体只是一个信息的载体，它只代表了相应物理内存页的信息，它本身并不是物理内存页。 那我们的物理内存页究竟在哪呢？Page 结构体又是通过怎样的方式找到它代表的物理内存页的地址呢？ 请你阅读 include/pmap.h 与 mm/pmap.c 中相关代码，给出你的想法。
物理内存页在主存或磁盘中
通过偏移和移位，确定物理地址

Thinking 2.6
请阅读 include/queue.h 以及 include/pmap.h, 将 Page_list 的结构梳理清楚,选择正确的展开结构(请注意指针)。

C.

```
struct Page_list{
    struct {
        struct {
            struct Page *le_next;
            struct Page **le_prev;
        } pp_link;
        u_short pp_ref;
    }* lh_first;
}
```

Thinking 2.7
在 mmu.h 中定义了|bzero(void *b, size_t)|这样一个函数,请你思考，此处的 b 指针是一个物理地址， 还是一个虚拟地址呢？
虚拟地址

Thinking 2.8
了解了二级页表页目录自映射的原理之后，我们知道，Win2k 内核的虚存管理也是采用了二级页表的形式，其页表所占的 4M 空间对应的虚存起始地址为 0xC0000000，那么，它的页目录的起始地址是多少呢？
0xC0160000

Thinking 2.9
思考一下 tlb_out 汇编函数,结合代码阐述一下跳转到 NOFOUND 的流程？从 MIPS 手册中查找 tlbp 和 tlbwi 指令，明确其用途，并解释为何第 10 行处指令后有 4 条 nop 指令。

```
#include <asm/regdef.h>
#include <asm/cp0regdef.h>
#include <asm/asm.h>

LEAF(tlb_out)
      nop
      mfc0       k1,CP0_ENTRYHI
      mtc0       a0,CP0_ENTRYHI
      nop
      // insert tlbp or tlbwi
      tlbp
      nop
      nop
      nop
      nop
      mfc0       k0,CP0_INDEX
      bltz       k0,NOFOUND
      nop
      mtc0       zero,CP0_ENTRYHI
      mtc0       zero,CP0_ENTRYLO0
      nop
      // insert tlbp or tlbwi
      tlbp
NOFOUND:
      mtc0       k1,CP0_ENTRYHI
      j      ra
      nop
END(tlb_out)
```

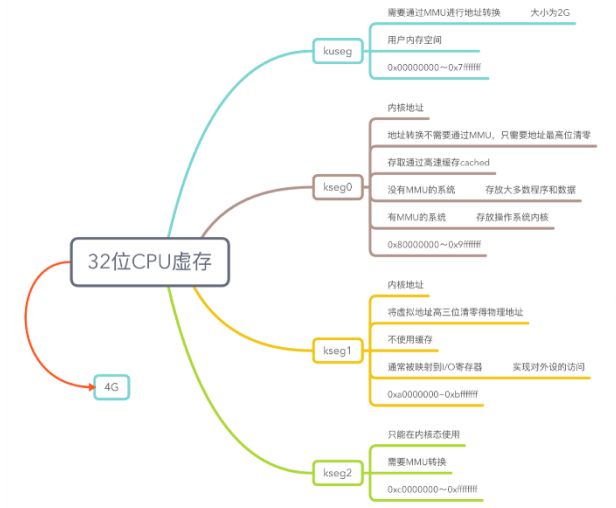
流程：

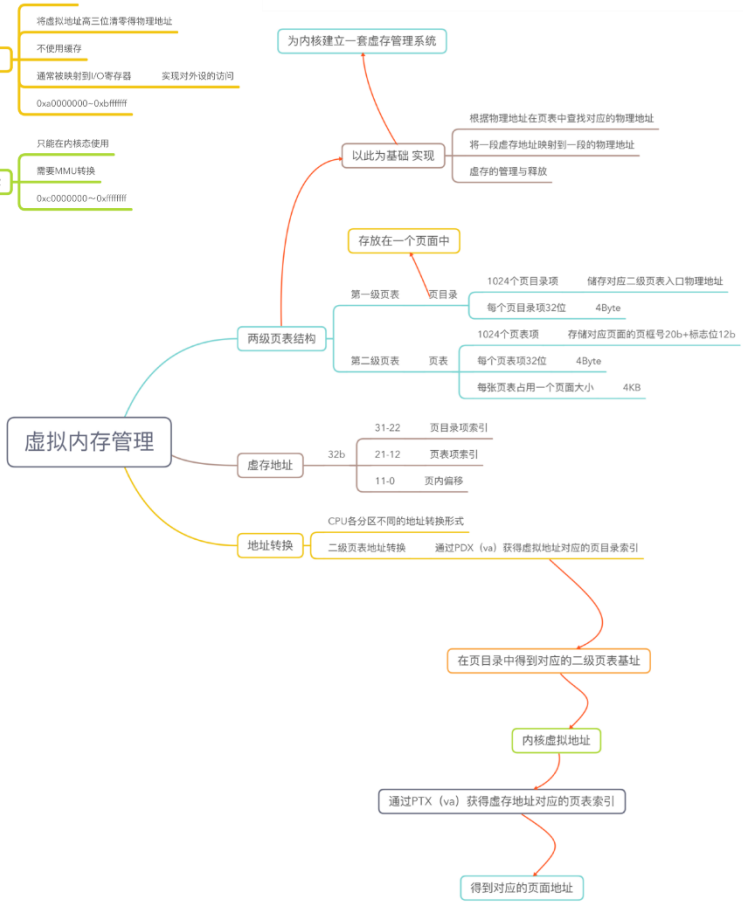
- set $k1 to the value stored in CP0_ENTRYHI
- set CP0_ENTRYHI to the value stored in $a0
- look up in TLB for an entry whose virtual page number and ASID matches those currently in CP0_ENTRYHI. If **not found** , 31 bit of Index register is set as the value is negative.
- set $k0 to the value stored in CP0_INDEX
- if $k0 < 0, go to the label NOFOUND
- move zero to CP0_ENTRYHI and CP0_ENTRYLO0
- write a TLB entry indexed by the Index register.
- move $k1 to CP0_ENTRYHI and restore it.
- jump to the return address for subroutine.

tlbp: to find a matching entry in the TLB
tlbwi: to write a TLB entry indexed by the Index register

tlbp 执行周期长，暂停流水线使之正常运行

二、 知识点与难点





三、 体会与感想

对于存取机制有了一定的了解，但不明白的地方还有很多（比如说在分配内存、建立页表等操作时的细节）

四、 残留难点

二级页表的自映射
TLB 部分等