第1章作业.

1. 1. 在一台运行 Linux 操作系统(页大小为 4KB)的 MIPS 计算机上执行如下 C 语言程序。在该程序执行过程中共发生了多少次例外？说明其过程。

```
void cycle(double * a) {
    int i;
    double b[65536];
    for(i=0;i<3;i++) {
        memcpy(a,b,sizeof(b));
    }
}
```

int i为局部变量，其值放在栈上。假设代码与i所在页已被pin住。

a和b大小均为512KB，不妨设他们从页的开头写入，则各占128页。

声明b时，并没有真正分配空间。

首次访问b时，每2页触发1次TLB Refill，每1页触发1次TLB invalid。

首次访问a也是如此。

因此，如果TLB足够大，并且页表未被换出，则触发128次TLB Refill和256次TLB invalid。

之后的循环中，由于TLB中已有表项，并且也为valid，不再触发例外。

如果TLB有限，那么在TLB表项被置换后，再访问时需触发TLB Refill，以对应的奇偶页为单位。

如果页面被从内存换出，也要再触发TLB invalid，以页为单位。

2. 2. 对于指令 cache 是否有必要考虑 cache 别名问题？

需要.

因为有3虚存，当cache使用虚地址索引时，有可能出现两个不同的cache行，不同的虚地址，却映射到同一个物理地址的情况。

①当遇到self-modifying自修改指令时，别名的Cache行会有数据不一致，此时再取指，当Cache命中时可能会取回旧值，从而导致错误。

比如虚地址0x10000与0x11000可能都映射到0x80011000，而这两个虚地址会被保存到不同的Cache line.

②通常L3-Cache以物理地址索引，若L1-Cache用虚地址索引，则CPU无法维护具有相同物理地址的L1与L3 Cache之间的数据一致性。

因此，通常采用：

第1拍：①虚地址TLB　　②虚地址低位访问Cache，取出Tag和data

第2拍：物理地址和Tag比较，取出对应数据.

这样可消除别名问题.因为虚地址的低位的含义与物理地址低位一致.

比如，0x10000与0x11000现在会被视为同一个Cache line.

3. 3. 假定在某一个 CPU 的 cache 中需要 64 位虚拟地址，8 位的进程标识，而其支持的物理内存最多有 64GB。请问，使用虚拟地址索引比使用物理地址作为索引的 Tag 大多少？这个值是否随着 cache 块大小的变化而发生改变？

虚地址索引：64+8=72位.

物理内存：6+30=36位.

故Tag大36位。

Cache块大小改变，会影响低位的分配，对总的差额并没有影响。

4. 4. 在一个包含 TLB 的当代处理器中，①请阐述 TLB、TLB 失效例外、页表和 Page fault 之间的关系；②如果有这样一个机器设计，对于同样的虚拟地址，TLB 命中和 Page fault 同时发生，这样的设计合理吗？为什么？③现代计算机页表普遍采用层次化的方式，请解释原因。

① TLB：Translation lookaside buffer，旁路转换缓冲。

它位于CPU内，是页表的Cache.

页表：记录虚拟地址到物理地址的映射.它以内存单位(页)记录虚拟地址对应到哪块物理地址.

TLB失效例外：

① TLB中没有记录当前虚地址，触发Refill倒外，TLB去查找页表并返回.

② TLB中有该项，但 V=0，触发invalid倒外，操作系统介入。

Page fault：缺页倒外.即页表中没有记录虚地址VA到物理地址的映射，或者记录的物理地址内容被换入硬盘。

② 不合理.

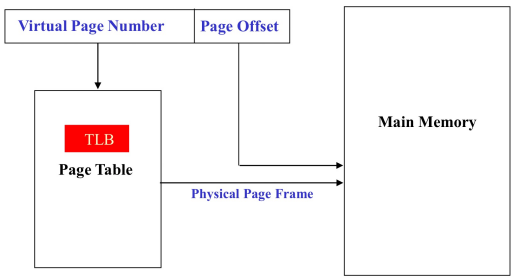
TLB命中 ⇒ TLB有VA对应页表且PA有效 ← 矛盾！

Page fault ⇒ 该VA无映射或PA无效

TLB与主存间一致性被破坏。

③ 层次化：用时再分配，不用不分配，节约空间。

如：单级页表



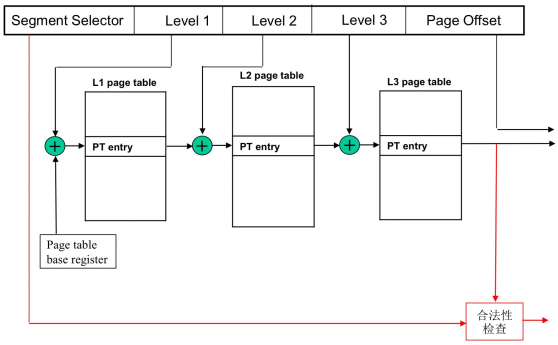
假设32位.页面4kB，12位.

页号 32-12=20位.

因此，2^{20}个页号，每条占4B，共用$2^{22}B = 4MB$空间.

并且，该4MB空间需要一次性连续分配.

多级页表：



若采用3级页表，8+8+4+12
L1大小为1KB，一次性连续分配
L2每页大小1KB，L3每页大小64B.
如果空间全部被分配，则单级与多级页表使用相同大小的空间。
但是L2与L3是随用随分配，故在空间未完全利用时，多级页表使用更少的空间。

上.

5. 已知一台计算机的虚地址为48位，物理地址为40位，页大小为16KB，TLB为64项全相联，TLB的每项包括一个虚页号vpn，一个物理页号pfn，以及一个有效位valid，请根据如下模块接口写出一个TLB的地址查找部分的Verilog代码。

```
module tlb_cam(vpn_in,pfn_out,hit,…);
```

其中，vpn_in为输入的虚页号，pfn_out为输出的物理页号，hit为表示是否找到的输出信号，“…”表示与该TLB输入输出有关的其他信号。重复的代码可以用“…”来简化，如：

```
module tlb_cam(vpn_in,  pfn_out, hit, valid_out);

Input [33:0] vpn_in;          //48位虚地址 – 14位页面 = 34位虚页号
Output [25:0] pfn_out;    //40位物理地址 – 14位页面 = 26位页号
Output hit;                   //命中
Output valid_out;          //页表项有效

Reg[60:0] tlb_content[63:0]; //64项表项；[60:27]vpn; [26:1]pfn; [0:0]valid;
Wire [63:0] tlb_hit;         //tlb命中one-hot

assign tlb_hit[0] = vpn_in == tlb_content[0][60:27] ;
……
assign tlb_hit[63] = vpn_in == tlb_content[63][60:27];

assign hit = | tlb_hit;

assign pfn_out = {26{tlb_hit[0]}} & tlb_content[0][26:1] |   //将tlb-hit扩充26位以选中pfn
                        ……
                        {26{tlb_hit[63]}} & tlb_content[63][26:1] ;

assign valid_out = tlb_hit[0] & tlb_content[0][0] |
                         ……
                         tlb_hit[63] & tlb_content[63][0];

endmodule
```