19/15期末虚存题解答

2021-1-7

19期末第五题

第五题（10 分）

某 32 位机器有 **28** 位地址空间，采用二级页表，一级页表中有 64 个 PTE，二级页表中有 1024 个 PTE，一级页表 4KB 对齐，PTE 最高一位是有效位，没有 TLB 和 Cache。惠楚思程序员在该机器上执行了如下代码。

```
1. int* a=calloc(10000, sizeof(int));
2. int* b = a+5000;
3. fork();
4. *b=0x80C3F110;
```

（1） 假设编译后 b 的值保存在寄存器中，请问该机器页面大小为①__字节，VPN1 长度为②____，VPN2 长度为③____。

（2） 由于该执行结果不符合预期，惠楚思需要对程序执行过程进行还原。程序执行过程中硬件记录了部分内存访问记录，但访问记录并不完整。目前已知在父进程执行第四行的时候，进行了若干次物理内存的读写操作，其中第一次读内存操作从地址 0x67F0E8 读出 0x80AA32C4，另外有两次将 0x80C3F110 写入内存的操作，写入的地址分别是 0xC3F50D 和 0xAA3AD0，但写入顺序并不清楚。程序结束后变量 b 中保存的值是①_____。在解析“*b”的过程中，一级页表的起始地址为②_____；二级页表的起始地址为③_____；物理页面的起始地址为④_____。（地址均用 16 进制表示）

First Analysis

虚拟地址大小

Background knowledge: 机器支持32位虚拟地址(CPU有32根地址线)，但可以不全用。同样地，一级页表有多少条目根据实际虚拟地址空间大小决定，无需占满整个页。

第五题（10 分）

某 32 位机器有 **28** 位地址空间，采用二级页表，一级页表中有 64 个 PTE，二级页表中有 1024 个 PTE，一级页表 4KB 对齐，PTE 最高一位是有效位，没有 TLB 和 Cache。惠楚思程序员在该机器上执行了如下代码。

```
1. int* a=calloc(10000, sizeof(int));
2. int* b = a+5000;
3. fork();
4. *b=0x80C3F110;
```

（1） 假设编译后 b 的值保存在寄存器中，请问该机器页面大小为①__字节，VPN1 长度为②____，VPN2 长度为③____。

（2） 由于该执行结果不符合预期，惠楚思需要对程序执行过程进行还原。程序执行过程中硬件记录了部分内存访问记录，但访问记录并不完整。目前已知在父进程执行第四行的时候，进行了若干次物理内存的读写操作，其中第一次读内存操作从地址 0x67F0E8 读出 0x80AA32C4，另外有两次将 0x80C3F110 写入内存的操作，写入的地址分别是 0xC3F50D 和 0xAA3AD0，但写入顺序并不清楚。程序结束后变量 b 中保存的值是①_____。在解析“*b”的过程中，一级页表的起始地址为②_____；二级页表的起始地址为③_____；物理页面的起始地址为④_____。（地址均用 16 进制表示）

First Analysis

VPN1长度 = log64 = 6
VPN2长度 = log1024 = 10
页面大小 = 2^(28 – VPN1 – VPN2) = 4096

第五题（10 分）

某 32 位机器有 **28** 位地址空间，采用二级页表，一级页表中有 64 个 PTE，二级页表中有 1024 个 PTE，一级页表 4KB 对齐，PTE 最高一位是有效位，没有 TLB 和 Cache。惠楚思程序员在该机器上执行了如下代码。

```
1. int* a=calloc(10000, sizeof(int));
2. int* b = a+5000;
3. fork();
4. *b=0x80C3F110;
```

27 22 21 12 11 0

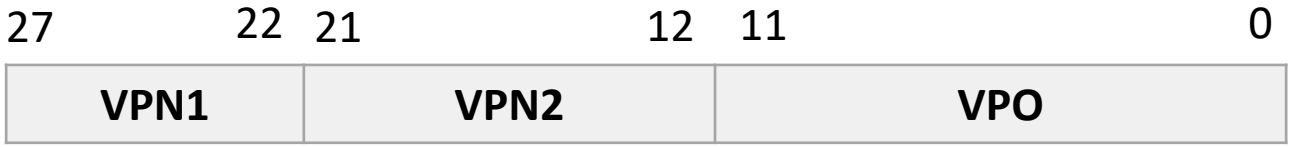
VPN1	VPN2	VPO

（1） 假设编译后 b 的值保存在寄存器中，请问该机器页面大小为①__4096__字节，VPN1 长度为②__6__，VPN2 长度为③__10__。

（2） 由于该执行结果不符合预期，惠楚思需要对程序执行过程进行还原。程序执行过程中硬件记录了部分内存访问记录，但访问记录并不完整。目前已知在父进程执行第四行的时候，进行了若干次物理内存的读写操作，其中第一次读内存操作从地址 0x67F0E8 读出 0x80AA32C4，另外有两次将 0x80C3F110 写入内存的操作，写入的地址分别是 0xC3F50D 和 0xAA3AD0，但写入顺序并不清楚。程序结束后变量 b 中保存的值是①_____。在解析“*b”的过程中，一级页表的起始地址为②_____；二级页表的起始地址为③_____；物理页面的起始地址为④_____。

（地址均用 16 进制表示）

第二问



第五题（10 分）

某 32 位机器有 **28** 位地址空间，采用二级页表，一级页表中有 64 个 PTE，二级页表中有 1024 个 PTE，一级页表 4KB 对齐，PTE 最高一位是有效位，没有 TLB 和 Cache。惠楚思程序员在该机器上执行了如下代码。

```
1. int* a=calloc(10000, sizeof(int));
2. int* b = a+5000;
3. fork();
4. *b=0x80C3F110;
```

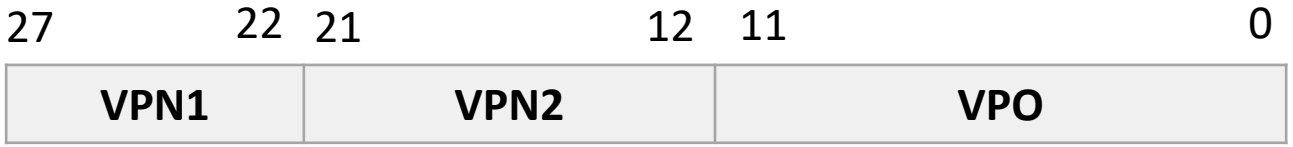
物理地址，因为内存通过物理地址访问

物理地址 24 位

（1） 假设编译后 b 的值保存在寄存器中，请问该机器页面大小为①__字节，VPN1 长度为②____，VPN2 长度为③____。

（2） 由于该执行结果不符合预期，惠楚思需要对程序执行过程进行还原。程序执行过程中硬件记录了部分内存访问记录，但访问记录并不完整。目前已知在父进程执行第四行的时候，进行了若干次物理内存的读写操作，其中第一次读内存操作从地址 0x67F0E8 读出 0x80AA32C4，另外有两次将 0x80C3F110 写入内存的操作，写入的地址分别是 0xC3F50D 和 0xAA3AD0，但写入顺序并不清楚。程序结束后变量 b 中保存的值是①_____。在解析“*b”的过程中，一级页表的起始地址为②_____；二级页表的起始地址为③_____；物理页面的起始地址为④_____。（地址均用 16 进制表示）

第二问



第五题（10 分）

某 32 位机器有 **28** 位地址空间，采用二级页表，一级页表中有 64 个 PTE，二级页表中有 1024 个 PTE，一级页表 4KB 对齐，PTE 最高一位是有效位，没有 TLB 和 Cache。惠楚思程序员在该机器上执行了如下代码。

```
1. int* a=calloc(10000, sizeof(int));
2. int* b = a+5000;
3. fork();
4. *b=0x80C3F110;
```

这里的描述不太好，但是意思是说后面提到的“一读两写”都是父进程执行第四行时发生的

（1） 假设编译后 b 的值保存在寄存器中，请问该机器页面大小为①__字节，VPN1 长度为②____，VPN2 长度为③____。

（2） 由于该执行结果不符合预期，惠楚思需要对程序执行过程进行还原。程序执行过程中硬件记录了部分内存访问记录，但访问记录并不完整。目前已知在父进程执行第四行的时候，进行了若干次物理内存的读写操作，其中第一次读内存操作从地址 0x67F0E8 读出 0x80AA32C4，另外有两次将 0x80C3F110 写入内存的操作，写入的地址分别是 0xC3F50D 和 0xAA3AD0，但写入顺序并不清楚。程序结束后变量 b 中保存的值是①_____。在解析“*b”的过程中，一级页表的起始地址为②_____；二级页表的起始地址为③_____；物理页面的起始地址为④_____。（地址均用 16 进制表示）

第二问

```
1. int* a=calloc(10000, sizeof(int));
2. int* b = a+5000;
3. fork();
4. *b=0x80C3F110;
```

明明只有一写，为什么会“一读两写”？

第五题（10 分）

某 32 位机器有 24 位地址空间，采用二级页表，一级页表中有 64 个 PTE，二级页表中有 1024 个 PTE，一级页表 4KB 对齐，PTE 最高一位是有效位，没有 TLB 和 Cache。惠楚思程序员在该机器上执行了如下代码。

原因是翻译虚拟地址必须从内存中**读两次页表**(二级地址翻译)！
那么为什么只有一读？-> 刚才标记过题目中的一个强调：记录不完整。
所以第一个地址要么是一级页表中的一个条目，要么是二级页表中的对应条目。

两写？ -> 注意到第三行fork了子进程，而第四行父进程试图写一个私有的COW页。因此会Page fault，并处理COW。**（注意PF并不总是因为COW，不理解的同学私下问助教。）**
COW需要干啥？ -> 拷贝新的页，**修改相应的PTE中的权限位和指向的物理地址**。

第二问

```
1. int* a=calloc(10000, sizeof(int));
2. int* b = a+5000;
3. fork();
4. *b=0x80C3F110;
```

（2） 由于该执行结果不符合预期，惠楚思需要对程序执行过程进行还原。程序执行过程中硬件记录了部分内存访问记录，但访问记录并不完整。目前已知在父进程执行第四行的时候，进行了若干次物理内存的读写操作，其中第一次读内存操作从地址 0x67F0E8 读出 0x80AA32C4，另外有两次将 0x80C3F110 写入内存的操作，写入的地址分别是 0xC3F50D 和 0xAA3AD0，但写入顺序并不清楚。程序结束后变量 b 中保存的值是①_____。在解析“*b”的过程中，一级页表的起始地址为②_____；二级页表的起始地址为③_____；物理页面的起始地址为④_____。（地址均用 16 进制表示）

但明明写了两次 *b 的值啊？

-> 巨坑的地方来了。不觉得 *b 的值很刻意吗？**(我们称为magic number，而且这还是magic number中性质特别恶劣的那一种。因此以后你的代码中要注意这一点。)**

第二问

```
1. int* a=calloc(10000, sizeof(int));
2. int* b = a+5000;
3. fork();
4. *b=0x80C3F110;
```

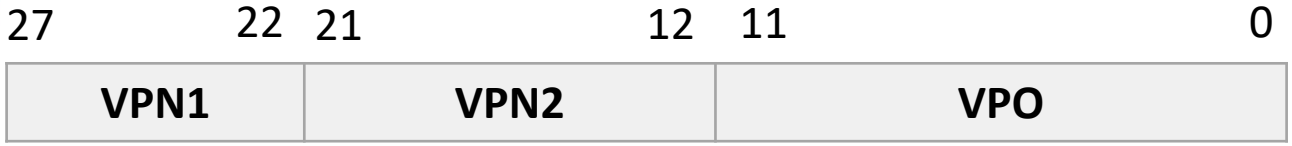
（2） 由于该执行结果不符合预期，惠楚思需要对程序执行过程进行还原。程序执行过程中硬件记录了部分内存访问记录，但访问记录并不完整。目前已知在父进程执行第四行的时候，进行了若干次物理内存的读写操作，其中第一次读内存操作从地址 0x67F0E8 读出 0x80AA32C4，另外有两次将 0x80C3F110 写入内存的操作，写入的地址分别是 0xC3F50D 和 0xAA3AD0，但写入顺序并不清楚。程序结束后变量 b 中保存的值是① _____。在解析“*b”的过程中，一级页表的起始地址为② _____；二级页表的起始地址为③ _____；物理页面的起始地址为④ _____。（地址均用 16 进制表示）

既然都是父进程写的，那只能理解为新的页表项也恰好是这个值了！
于是这两次写一次是更新页表，一次是写 *b 的数据。

同时得知，一个 PTE 占 4 个字节。（或者也能从一页4KB，二级页表有1024个条目得知）

第二问 # 已弄清场景，开始解题

```
1. int* a=calloc(10000, sizeof(int));
2. int* b = a+5000;
3. fork();
4. *b=0x80C3F110;
```



（2） 由于该执行结果不符合预期，惠楚思需要对程序执行过程进行还原。程序执行过程中硬件记录了部分内存访问记录，但访问记录并不完整。目前已知在父进程执行第四行的时候，进行了若干次物理内存的读写操作，其中第一次读内存操作从地址 0x67F0E8 读出 0x80AA32C4，另外有两次将 0x80C3F110 写入内存的操作，写入的地址分别是 0xC3F50D 和 0xAA3AD0，但写入顺序并不清楚。程序结束后变量 b 中保存的值是① _____。在解析“*b”的过程中，一级页表的起始地址为② _____；二级页表的起始地址为③ _____；物理页面的起始地址为④ _____。（地址均用 16 进制表示）

这个地址有一段是下一页的起始物理地址（注意页一般是按大小对齐的，尽管题目上只强调了一级页表的对齐）

要么是一级页表的 PTE 的地址，要么是二级页表的 PTE 的地址。

一个是 COW 后父进程 *b 对应的物理地址（注意 b 的值是虚拟地址，不受影响！），一个是新的页表项地址。

第二问

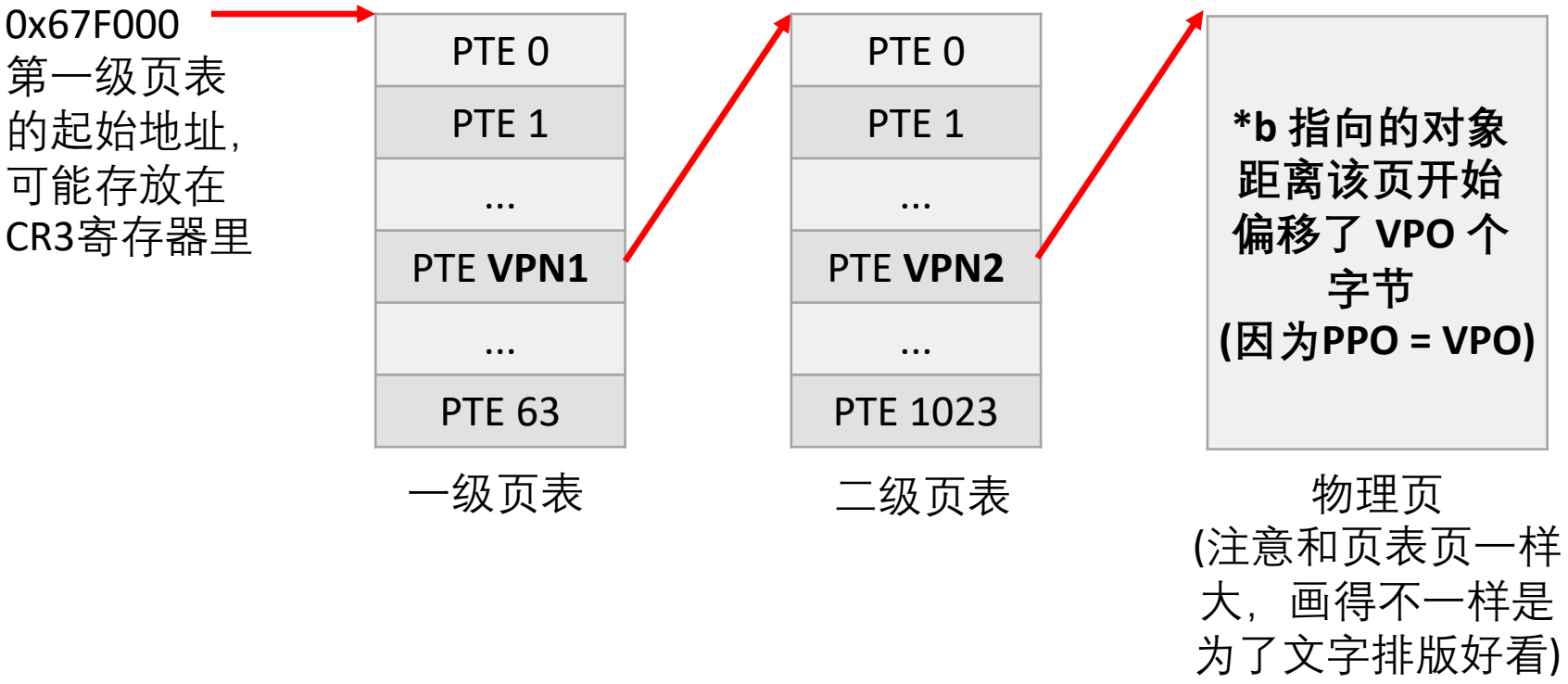
```
1. int* a=calloc(10000, sizeof(int));
2. int* b = a+5000;
3. fork();
4. *b=0x80C3F110;
```

（2） 由于该执行结果不符合预期，惠楚思需要对程序执行过程进行还原。程序执行过程中硬件记录了部分内存访问记录，但访问记录并不完整。目前已知在父进程执行第四行的时候，进行了若干次物理内存的读写操作，其中第一次读内存操作从地址 0x67F0E8 读出 0x80AA32C4，另外有两次将 0x80C3F110 写入内存的操作，写入的地址分别是 0xC3F50D 和 0xAA3AD0，但写入顺序并不清楚。程序结束后变量 b 中保存的值是①_____。在解析“*b”的过程中，一级页表的起始地址为②_____；二级页表的起始地址为③_____；物理页面的起始地址为④_____。（地址均用 16 进制表示）

注意到红框中的对应（幸亏这里的对应位正好是4bit的位数，不然就难观察了），断言：**0xC3F50D 是 COW 后 *b 对应的物理地址，而 0XAA3AD0 是二级页表 PTE 的地址。顺便得知，0x67F0E8 是一级页表 PTE 的地址。**

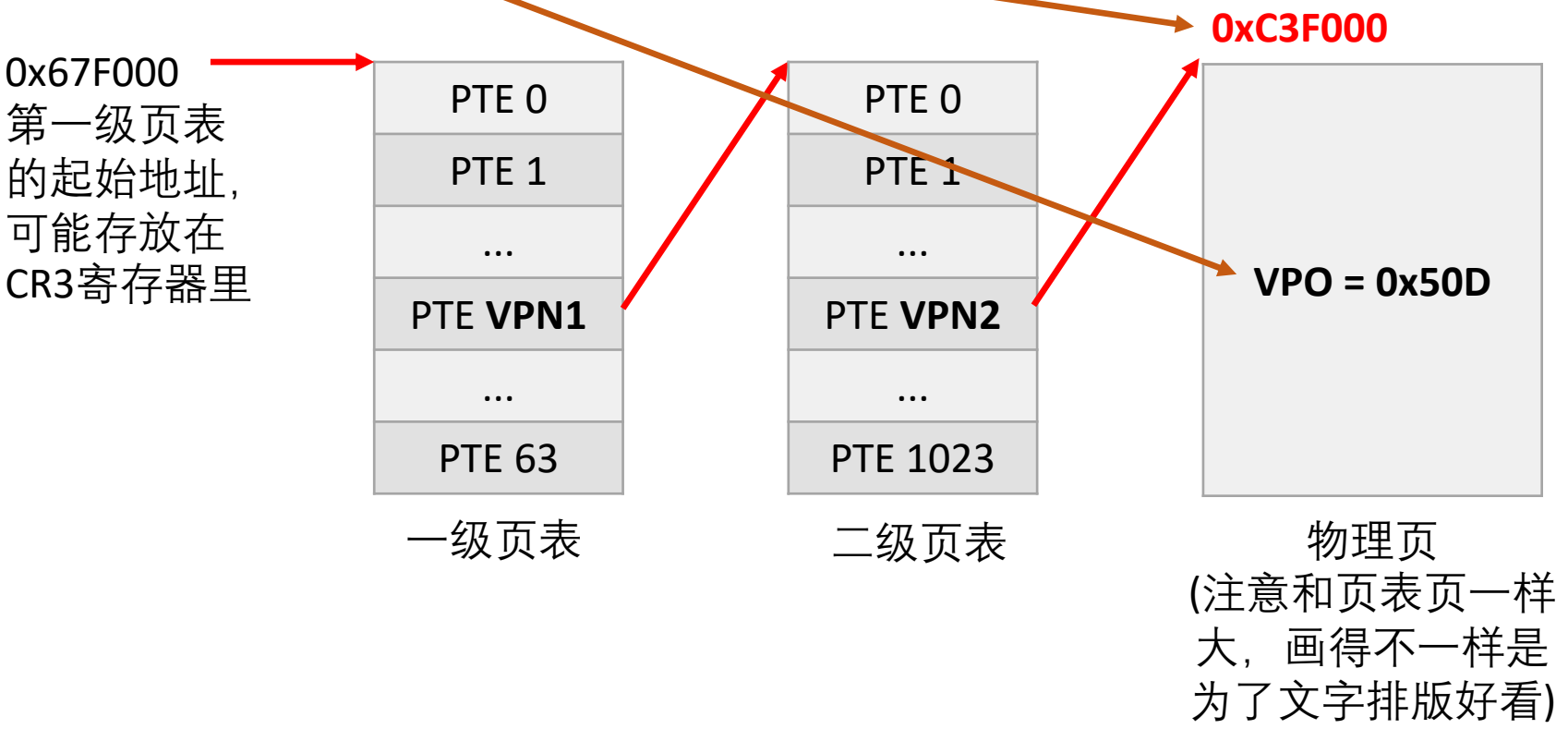
0xC3F50D 是 COW 后 *b 对应的物理地址，而0XAA3AD0 是二级页表 PTE 的地址。顺便得知，0x67F0E8 是一级页表 PTE 的地址。

-> 由于一级页表起始地址是4KB对齐的。那么必然为 0x67F000 ！有一个空可以填了。



0xC3F50D 是 COW 后 *b 对应的物理地址，而0XAA3AD0 是二级页表 PTE 的地址。顺便得知，0x67F0E8 是一级页表 PTE 的地址。

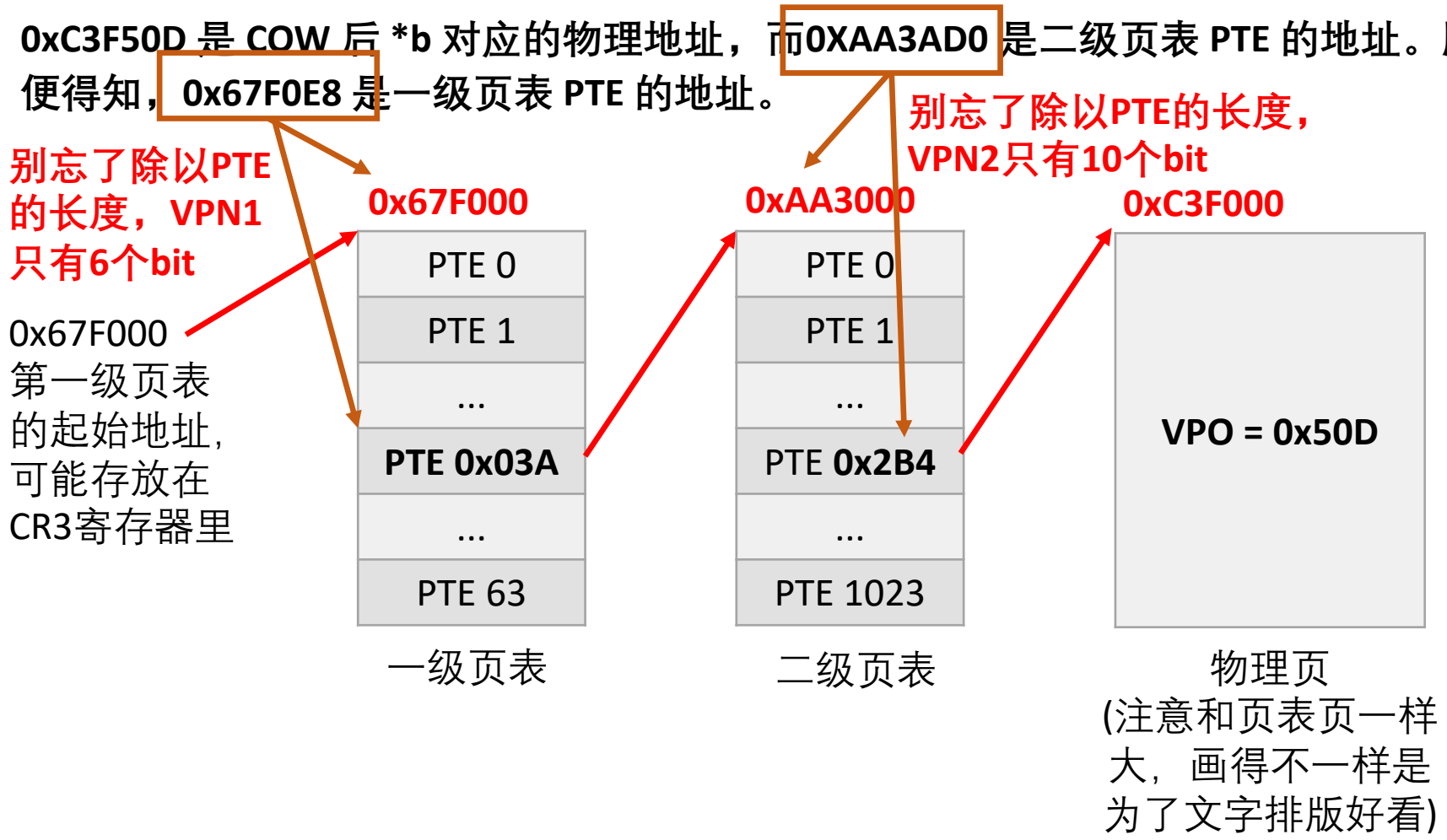
-> 由于一级页表起始地址是4KB对齐的。那么必然为 0x67F000 ！有一个空可以填了。



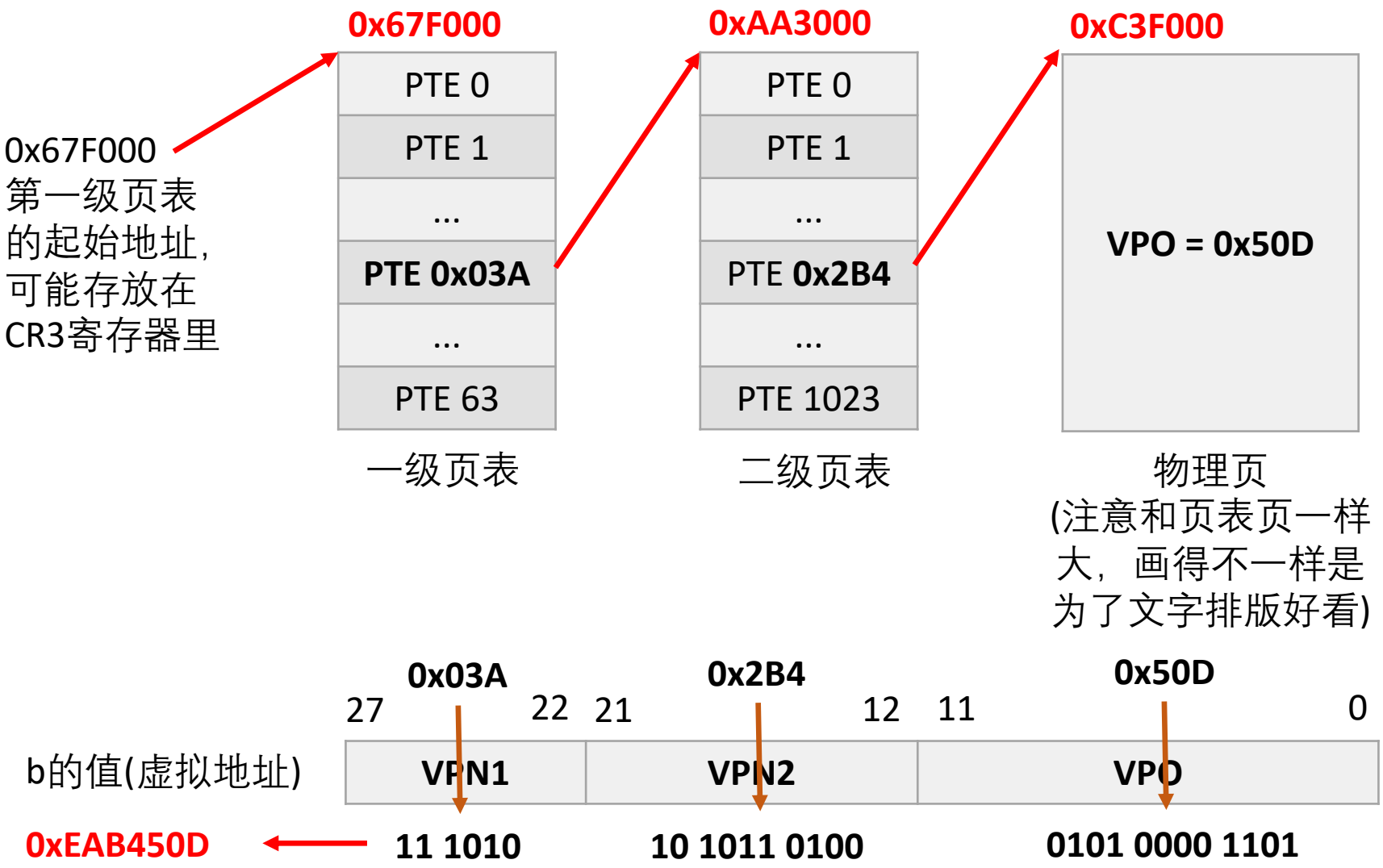
0xC3F50D 是 COW 后 *b 对应的物理地址，而0XAA3AD0 是二级页表 PTE 的地址。顺便得知，0x67F0E8 是一级页表 PTE 的地址。

别忘了除以PTE的长度，VPN1只有6个bit

别忘了除以PTE的长度，VPN2只有10个bit



0xC3F50D 是 COW 后 *b 对应的物理地址，而0XAA3AD0 是二级页表 PTE 的地址。顺便得知，0x67F0E8 是一级页表 PTE 的地址。



Case Closed

（2） 由于该执行结果不符合预期，惠楚思需要对程序执行过程进行还原。程序执行过程中硬件记录了部分内存访问记录，但访问记录并不完整。目前已知在父进程执行第四行的时候，进行了若干次物理内存的读写操作，其中第一次读内存操作从地址 0x67F0E8 读出 0x80AA32C4，另外有两次将 0x80C3F110 写入内存的操作，写入的地址分别是 0xC3F50D 和 0xAA3AD0，但写入顺序并不清楚。程序结束后变量 b 中保存的值是① 0xEAB450D 。在解析“*b”的过程中，一级页表的起始地址为② 0x67F000 ；二级页表的起始地址为③ 0xAA3000 ；物理页面的起始地址为④ 0xC3F000 。（地址均用 16 进制表示）

其实最好写成 0x0EAB450D。

为了方便，我们一直省略了32位地址最前面 4 位的0。注意到 b 总是保存一个 32 位值。只是由于操作系统设计的虚址空间只有 28 位，最高的 4 位总是 0。

Reflection

（2） 由于该执行结果不符合预期，惠楚思需要对程序执行过程进行还原。程序执行过程中硬件记录了部分内存访问记录，但访问记录并不完整。目前已知在父进程执行第四行的时候，进行了若干次物理内存的读写操作，其中第一次读内存操作从地址 0x67F0E8 读出 0x80AA32C4，另外有两次将 0x80C3F110 写入内存的操作，写入的地址分别是 0xC3F50D 和 0xAA3AD0，但写入顺序并不清楚。程序结束后变量 b 中保存的值是① **0x0EAB450D** 。在解析“*b”的过程中，一级页表的起始地址为② **0x67F000** ；二级页表的起始地址为③ **0xAA3000** ；物理页面的起始地址为④ **0xC3F000** 。（地址均用 16 进制表示）

1. b 的值居然没有按 4 字节对齐？ -> 出题的 bug
2. 做虚存题最忌讳盲目。但是**也没有必要**像本讲义中的那样规范严谨。完全应该根据自己的感觉和判断来进展。这里的示例只是提供严谨的思考工具！**现实中谁会一下就能这样清晰理性地思考呢？（好了好了，知道那些大神我无法理解了qwq）**
3. **如果**一个magic number在题目中显得有存在感（就是有用），那么它一般有特殊含义。

15期末第六题 # 慢慢读题

常识：
PDE 一般指一级页表中的 PTE。目录页即是第一级页表 (根页表)。
想想这个命名方法的恰当之处何在？这样你就不会忘了。

得分

第六题（15 分）虚拟内存地址转换

为了提升虚拟内存地址的转换效率，降低遍历两级页表结构所带来的地址转换开销，英特尔处理器中引入了大页 TLB，即一个 TLB 项可以涵盖整个 4MB 对齐的地址空间（针对 32 位模式）。只要设置页目录页中页目录项（PDE）的大页标志位，即可让 MMU 识别这是一个大页 PDE，并加载到大页 TLB 项中。大页 PDE 中记录的物理内存页面号必须是 4MB 对齐的，并且整个连续的 4MB 内存均可统一通过该大页 PDE 进行地址转换。

在 32 位的 Linux 系统中，为了方便访问物理内存，内核将地址 0~768MB 间的物理内存映射到虚拟内存地址 3GB~3GB+768MB 上，并通过大页 PDE 进行进行该区间的地址转换。任何 0~768MB 的物理内存地址可以直接通过加 3G（0xC0000000）的方式得到其虚拟内存地址。在内核中，除了该区间的内存外，其他地址的内存通常都通过普通的两级页表结构来进行地址转换。

15期末第六题 # 慢慢读题

得分

第一段话读完：形成自己的理解。
（如果不能立刻形成，也可以尝试在做第一小题的过程中边尝试边理解）

第六题（15 分）虚拟内存地址转换

为了提升虚拟内存地址的转换效率，降低遍历两级页表结构所带来的地址转换开销，英特尔处理器中引入了大页 TLB，即一个 TLB 项可以涵盖整个 4MB 对齐的地址空间（针对 32 位模式）。只要设置页目录页中页目录项（PDE）的大页标志位，即可让 MMU 识别这是一个大页 PDE，并加载到大页 TLB 项中。大页 PDE 中记录的物理内存页面号必须是 4MB 对齐的，并且整个连续的 4MB 内存均可统一通过该大页 PDE 进行地址转换。

在 32 位的 Linux 系统中，为了方便访问物理内存，内核将地址 0~768MB 间的物理内存映射到虚拟内存地址 3GB~3GB+768MB 上，并通过大页 PDE 进行进行该区间的地址转换。任何 0~768MB 的物理内存地址可以直接通过加 3G（0xC0000000）的方式得到其虚拟内存地址。在内核中，除了该区间的内存外，其他地址的内存通常都通过普通的两级页表结构来进行地址转换。

15期末第六题 # 慢慢读题

得分

第六题（15 分）虚拟内存地址转换

为了提升虚拟内存地址的转换效率，降低遍历两级页表结构所带来的地址转换开销，英特尔处理器中引入了大页 TLB，即一个 TLB 项可以涵盖整个 4MB 对齐的地址空间（针对 32 位模式）。只要设置页目录页中页目录项（PDE）的大页标志位，即可让 MMU 识别这是一个大页 PDE，并加载到大页 TLB 项中。大页 PDE 中记录的物理内存页面号必须是 4MB 对齐的，并且整个连续的 4MB 内存均可统一通过该大页 PDE 进行地址转换。

在 32 位的 Linux 系统中，为了方便访问物理内存，内核将地址 0~768MB 间的物理内存映射到虚拟内存地址 3GB~3GB+768MB 上，并通过大页 PDE 进行进行该区间的地址转换。任何 0~768MB 的物理内存地址可以直接通过加 3G（0xC0000000）的方式得到其虚拟内存地址。在内核中，除了该区间的内存外，其他地址的内存通常都通过普通的两级页表结构来进行地址转换。

MMU 翻译地址时总要先去目录页中找到 PDE，如果有大页标记，那么立刻就能从该 PDE 中得到物理地址(VPO = log 4M = 22位)。奇怪的是还总要加载到大页的 TLB 中。（幸亏TLB里的条目都是有效的，至于为什么一定要加载到 TLB，你自己找个解释就好。后续的题与此关系不大。）

15期末第六题 # 慢慢读题

得分

第六题（15 分）虚拟内存地址转换

为了提升虚拟内存地址的转换效率，降低遍历两级页表结构所带来的地址转换开销，英特尔处理器中引入了大页 TLB，即一个 TLB 项可以涵盖整个 4MB 对齐的地址空间（针对 32 位模式）。只要设置页目录页中页目录项（PDE）的大页标志位，即可让 MMU 识别这是一个大页 PDE，并加载到大页 TLB 项中。大页 PDE 中记录的物理内存页面号必须是 4MB 对齐的，并且整个连续的 4MB 内存均可统一通过该大页 PDE 进行地址转换。

在 32 位的 Linux 系统中，为了方便访问物理内存，内核将地址 0~768MB 间的物理内存映射到虚拟内存地址 3GB~3GB+768MB 上，并通过大页 PDE 进行进行该区间的地址转换。任何 0~768MB 的物理内存地址可以直接通过加 3G（0xC0000000）的方式得到其虚拟内存地址。在内核中，除了该区间的内存外，其他地址的内存通常都通过普通的两级页表结构来进行地址转换。

0x00000000-0x30000000 的物理地址转换到虚拟地址的方法，先记着加 0xC0000000，用到可以再回头看确认一下。

蛤？大页不是4MB对齐的吗？
仔细一看，题目说是 PDE 里，不是TLB， 先不管了。(其实你可以这样认为，大页 TLB 里存放的不是大页的PTE，而是这个大页划分成普通页时的部分PTE，这样能解释过去这组数组)

假设在我们使用的处理器中有 2 个大页 TLB 项，其当前状态如下：

索引号	TLB 标记	页面号	有效位
0	0xC4812	0x04812	1
1	0xC9C33	0x09C33	1

有 4 个普通 TLB 项，当前的状态如下：

索引号	TLB 标记	页面号	有效位
0	0xF8034	0x04812	1
1	0xF8033	0x09812	1
2	0xF4427	0x12137	1
3	0xF44AE	0x17343	1

当前页活跃的目录页（PD）中的部分 PDE 的内容如下：

PDE 索引	页面号	其他标志	大页位	存在位
786	0x04800	…	1	1
807	0x09C00	…	1	1
977	0x09C33	…	0	1
992	0x09078	…	0	1

注：普通页面大小为 4KB，并且 4KB 对齐。每个页面的页面号为其页面起始物理地址除以 4096 得到。大页由连续 1024 个 4KB 小页组成，且 4MB 对齐。

这个我们熟悉！OK，过。

假设在我们使用的处理器中有 2 个大页 TLB 项，其当前状态如下：

索引号	TLB 标记	页面号	有效位
0	0xC4812	0x04812	1
1	0xC9C33	0x09C33	1

有 4 个普通 TLB 项，当前的状态如下：

索引号	TLB 标记	页面号	有效位
0	0xF8034	0x04812	1
1	0xF8033	0x09812	1
2	0xF4427	0x12137	1
3	0xF44AE	0x17343	1

当前页活跃的目录页（PD）中的部分 PDE 的内容如下：

PDE 索引	页面号	其他标志	大页位	存在位
786	0x04800	…	1	1
807	0x09C00	…	1	1
977	0x09C33	…	0	1
992	0x09078	…	0	1

注：普通页面大小为 4KB，并且 4KB 对齐。每个页面的页面号为其页面起始物理地址除以 4096 得到。大页由连续 1024 个 4KB 小页组成，且 4MB 对齐。

这个就是每次MMU都要检查是不是大页用到的一级页表了。

假设在我们使用的处理器中有 2 个大页 TLB 项，其当前状态如下：

索引号	TLB 标记	页面号	有效位
0	0xC4812	0x04812	1
1	0xC9C33	0x09C33	1

有 4 个普通 TLB 项，当前的状态如下：

索引号	TLB 标记	页面号	有效位
0	0xF8034	0x04812	1
1	0xF8033	0x09812	1
2	0xF4427	0x12137	1
3	0xF44AE	0x17343	1

当前页活跃的目录页（PD）中的部分 PDE 的内容如下：

PDE 索引	页面号	其他标志	大页位	存在位
786	0x04800	…	1	1
807	0x09C00	…	1	1
977	0x09C33	…	0	1
992	0x09078	…	0	1

注：普通页面大小为 4KB，并且 4KB 对齐。每个页面的页面号为其页面起始物理地址除以 4096 得到。大页由连续 1024 个 4KB 小页组成，且 4MB 对齐。

千万别忘了读 注 。也许你 get 不到啥新的信息，但读一下以防万一。

假设在我们使用的处理器中有 2 个大页 TLB 项，其当前状态如下：

索引号	TLB 标记	页面号	有效位
0	0xC4812	0x04812	1
1	0xC9C33	0x09C33	1

有 4 个普通 TLB 项，当前的状态如下：

索引号	TLB 标记	页面号	有效位
0	0xF8034	0x04812	1
1	0xF8033	0x09812	1
2	0xF4427	0x12137	1
3	0xF44AE	0x17343	1

当前页活跃的目录页（PD）中的部分 PDE 的内容如下：

PDE 索引	页面号	其他标志	大页位	存在位
786	0x04800	…	1	1
807	0x09C00	…	1	1
977	0x09C33	…	0	1
992	0x09078	…	0	1

注：普通页面大小为 4KB，并且 4KB 对齐。每个页面的页面号为其页面起始物理地址除以 4096 得到。大页由连续 1024 个 4KB 小页组成，且 4MB 对齐。

```
movl $0xC4812024, %ebx
movl $128, (%ebx)
movl $0xF8034000, %ecx
movl $36(%ecx), %eax
```

请问，执行完上述指令后，eax 寄存器中的内容是（）；在执行上述指令过程中，共发生了（）次 TLB miss？同时会发生（）次 page fault？

注：不能确定时填写“--”。

翻译虚拟地址 0xC4812024 ：

根据描述，先用高 10 位去 PD 里看一下是不是大页。

1100 0100 1000 0001 0010 0000 0010 0100

What’s the fk？这索引还要翻译成 10 进制？？？**

当前页活跃的目录页（PD）中的部分 PDE 的内容如下：

	PDE 索引	页面号	其他标志	大页位	存在位
11 0001 0010	786	0x04800	…	1	1
11 0010 0111	807	0x09C00	…	1	1
11 1101 0001	977	0x09C33	…	0	1
11 1110 0000	992	0x09078	…	0	1

```
movl $0xC4812024, %ebx
movl $128, (%ebx)
movl $0xF8034000, %ecx
movl $36(%ecx), %eax
```

请问，执行完上述指令后，eax 寄存器中的内容是（）；在执行上述指令过程中，共发生了（）次 TLB miss？同时会发生（）次 page fault？

注：不能确定时填写“--”。

是大页！条目有效。 -> 根据描述直接算出物理地址是

0x048 占 10 位， 拼上虚拟地址的低 22 位。 -> 0x04812024

翻译虚拟地址 0xF8034000：
根据描述，先用高 10 位去 PD 里看一下是不是大页。

当前页活跃的目录页（PD）中的部分 PDE 的内容如下：

	PDE 索引	页面号	其他标志	大页位	存在位
0x 11 0001 0010	786	0x04800	…	1	1
0x 11 0010 0111	807	0x09C00	…	1	1
0x 11 1101 0001	977	0x09C33	…	0	1
0x 11 1110 0000	992	0x09078	…	0	1

```
movl $0xC4812024, %ebx
movl $128, (%ebx)
movl $0xF8034000, %ecx
movl $36(%ecx), %eax
```

请问，执行完上述指令后，eax 寄存器中的内容是（）；在执行上述指令过程中，共发生了（）次 TLB miss？同时会发生（）次 page fault？

注：不能确定时填写“--”。

不是大页！条目有效。 -> 得知第二级页表起始物理地址是 0x09078000。

页表项 4 字节，VPN2 = 0x034。二级页表项的相应 PTE 所在地址：0x090780d0

等一下等一下，别忘了要去查 TLB 有无命中！没命中再看。

坑爹的来了，照理说 Tag 位不应该包含 Index 位。注意 TLB 有 4 组， TLBI = log4 = 2，它却写了上去。不过倒是不用再做进制转换了。

得到 0x04812000 + (36)10 = 0x04812024

有 4 个普通 TLB 项，当前的状态如下：

索引号	TLB 标记	页面号	有效位
0	0xF8034	0x04812	1
1	0xF8033	0x09812	1
2	0xF4427	0x12137	1
3	0xF44AE	0x17343	1

```
movl $0xC4812024, %ebx
movl $128, (%ebx)
movl $0xF8034000, %ecx
movl $36(%ecx), %eax
```

请问，执行完上述指令后，eax 寄存器中的内容是 128；在执行上述指令过程中，共发生了 0 次 TLB miss？同时会发生 0 次 page fault？

注：不能确定时填写“--”。

你可能注意到了后面 3 个组的 TLBT 的结尾两位和索引号根本不对应！
（已经强调了正常 TLBT 是不含 TLBI 的，但谁让这题这样出）
确实，这是题目的一个bug。先硬着头皮做吧。

有 4 个普通 TLB 项，当前的状态如下：

索引号	TLB 标记	页面号	有效位
0	0xF8034	0x04812	1
1	0xF8033	0x09812	1
2	0xF4427	0x12137	1
3	0xF44AE	0x17343	1

第二小题相对简单，可参考答案的解释。
有需要私戳我。下面是第三小题的解答。

3．下列虚拟地址中哪一个对应着够将虚拟内存地址 0xF4427048 映射到物理内存地址 0x14321048 的页表项（）？

（A）0x09C33027　　（B）0xC9C3309C
（C）0xC9C33027　　（D）0x09C3309C

重复步骤。
高 10 位，1111 0100 01，不是大页！
TLB 命中！物理地址 0x12137048。

等等，等等！我们要干嘛？ -> 要修改 PTE！
回去 PD 找二级页表起始地址 0x09C33000。
所以该虚拟地址二级页表项地址为 0x09C33000 + 0x027 * 4 = 0x09C3309C
注意这是一个物理地址。为了修改，我们需要提供虚拟地址。
还记得题目开头说的转换方式吗？ -> 虚拟地址 0xC9C3309C，选 B。

Move On

3．下列虚拟地址中哪一个对应着够将虚拟内存地址 0xF4427048 映射到物理内存地址 0x14321048 的页表项（）？

（A）0x09C33027　　（B）0xC9C3309C

（C）0xC9C33027　　（D）0x09C3309C

通过上述虚拟地址，利用 movl 指令修改对应的页表项，完成上述映射，在此过程中，是否会产生 TLB miss？（）（回答：会/不会/不确定）

只好再翻译一下 0xC9C3309C 了。
先去 PD 看看是不是大页。-> 是！
大页 TLB 命中了吗？-> 命中了！
填 不会。

修改页表项后，是否可以立即直接使用下面的指令序列将物理内存地址 0x14321048 开始的一个 32 位整数清零？为什么？

```
movl $0xF4427048, %ebx
movl $0, (%ebx)
```

答：

你注意这时候 PD 显示是一个普通页，普通页的该 TLB 项指向的物理页面还没更新！必须先标记它为失效（这里不用care怎么标记），然后重新设置有效的 PTE 反映这个新的映射关系。 因此 不行。

Case Closed