第八章 参考答案 转移预测

1. 解：BHT $2^6*9=576b$

PHT $2^8*2^9*2=262144b$

共 262720b

基本原理：BHT 根据地址低 6 位选出一个 9 位向量，和地址低 8 位一起到 PHT 中选取 2 位饱和计数。图略

2. 解： 这个题的题干歧义

S1 中 b[i]读出和上一次循环 S4 中 b[i+1]赋值真相关

S3 中 a[i-1]赋值和上一次循环 S1 中 a[i]赋值输出相关

S3 中 a[i-1]赋值和上一次循环 S1 中 a[i]读出反相关

S3 中 a[i-1]赋值和上一次循环 S2 中 a[i]读出反相关

S3 中 b[i]读出和上一次循环 S4 中 b[i+1]赋值真相关

S4 中 b[i]读出和上一次循环 S4 中 b[i+1]赋值真相关

5. 解： **概念题**

(a)

```
1:beqz t1, 1f
  nop
  ……
1:bnez t1, 1b
  nop
```

假设两条转移指令共享一个历史表项，它们的结果完全相反

(b)

```
For(i=0;i<10;i++) j+=i;
For(i=0;i<10;i++) j+=2*i;
```

假如两个 for 循环的转移指令共享共享一个历史表项，第二个循环会得益于第一个循环的预测，第一次跳转能猜测成功（假设是 2 位饱和计数器）。

(c) 如果使用其他地址，例如高位地址，对某些程序会有好处，但离得很近的多条分支指令就会共享同一个表项，对不同的程序，会对分支正确率产生不同影响。

如果对多位地址进行哈希，效果会更好，会减少两条转移指令共享一个历史表项的概率。

6. 解：写出软流水代码，计算拍数即可。留意指令的延迟

（1）

```
//装入代码
        LDC1        F0,0(R1)
        ADD.D       F2,F0,F1
        LDC1        F0,-8(R1)
        ADDIU       R1,R1,-24
//主体循环
L1:     SDC1        F2,24(R1) //Loop i-2
        ADD.D       F2,F0,F1  //Loop i-1
        LDC1        F0,8(R1)  //Loop i
```

```
        BNE         R1,R2,L1
        ADDIU       R1,R1,-8
//排空代码
        SDC1        F2,24(R1)
        ADD.D       F2,F0,F1
        SDC1        F2,16(R1)
```

总的执行周期数：(4+1) + (5*(n-2)) + (3+2) = 5n

（2）软流水无法降低分支频率，循环展开可以；

软流水代码量增加少，循环展开会带来代码膨胀。

7. 答：通过在处理器维护一个指令的有序队列（如 reorder-buffer, branch-queue）。当发现某转移指令猜测失败时，依据该有序队列的顺序信息，将该转移指令后面的指令取消，同时根据正确的跳转目标重新取指。如果采用的是 reorder-buffer，取消的粒度是指令；如果采用的是 branch-queue，取消的粒度是基本块。 **概念题**

第九章 参考答案 功能部件

1. 解：

a). 补码: 分别表示-0x70104000 和 0
b). 无符号数；分别表示 0x8FEFC000 和 0
c). 单精度浮点数；分别表示$-(1.110111111)_b \times 2^{31-127}$ 和+0.0
d). 一条 MIPS 指令：分别表示 LW R15, 0xC000(R31)以及 NOP 指令 **查阅资料题**

2. 解： 第二问先行进位记得算 pg 和全加器的延迟。

a). 串行进位加法器；
每一级进位传递的延迟为 2T，因此生成 c16 需要 32T；
每一级产生结果的延迟为 3T，因此生成 s15 需要（30T+3T）=33T

b). 先行进位加法器
参照课本图 9.6 的方式搭建：组内并行、组间并行，4 位一组。
延迟共 2T（产生 p、g）+2T（产生每组 P、G）+2T（产生组间进位）+2T（产生组内进位）+ 3T（全加器逻辑）= 11T

c). 先行进位加法器快的原因是它能更快地生成第 i 位的 c 而不需要依赖于第 i-1 位的 c。 **概念题**

3. 解： 部分同学对第二问“加法树”的理解有歧义，题目中想说这是华莱士树

a). 使用多个加法器；

采用先行进位加法器两两相加，需要 2*11T=22T 延迟

b). 使用加法树及加法器。

使用加法树把四个数相加变成两个数相加，需要 2 级全加器延迟（6T），然后再使用先行进位加法器（11T）得到最后结果，因此共 6T+11T=17T 延迟。

4. 证明：

假定带符号数 x, y, x+y 都在 n 位数表示范围之内，由于求补的本质是取模运算，则它们的补码可以如下表示：$[x]_{补} = 2^{n+1} + x$, $[y]补 = 2^{n+1} + y$, $[x+y]补 = 2^{n+1} + x + y$ ，其中第 n+1 位舍弃。于是:

$[x]补 + [y]补 = 2^{n+1} + x + 2^{n+1} + y = 2^{n+1} + (2^{n+1} + x + y) = 2^{n+1} + [x+y]补 = [x+y]补$（第 n+1 位舍弃）

■

5. Verilog 题

```
module add16(a, b, cin, out, cout);
    input    [15:0]   a;
    input    [15:0]   b;
    input             cin;
    output   [15:0]   out;
    output            cout;

    wire [15:0] p = a|b;
    wire [15:0] g = a&b;
    wire [3:0]    P, G;
    wire [15:0]   c;

    assign c[0] = cin;

    C4 C0_3(.p(p[3:0]),.g(g[3:0]),.cin(c[0]),.P(P[0]),.G(G[0]),.cout(c[3:1]));
    C4 C4_7(.p(p[7:4]),.g(g[7:4]),.cin(c[4]),.P(P[1]),.G(G[1]),.cout(c[7:5]));
    C4 C8_11(.p(p[11:8]),.g(g[11:8]),.cin(c[8]),.P(P[2]),.G(G[2]),.cout(c[11:9]));
    C4 C12_15(.p(p[15:12]),.g(g[15:12]),.cin(c[12]),.P(P[3]),.G(G[3]),.cout(c[15:13]));
    C4 C_INTER(.p(P),.G(G),.cin(c[0]),.P(),.G(),.cout({c[12],c[8],c[4]}));

    assign cout = (a[15]&b[15]) | (a[15]&c[15]) | (b[15]&c[15]);
    assign out = (~a&~b&c)|(~a&b&~c)|(a&~b&~c)|(a&b&c);
endmodule
```

```
module C4(p,g,cin,P,G,cout)
    input    [3:0]    p, g;
    input             cin;
    output            P,G;
    output   [2:0]    cout;
```

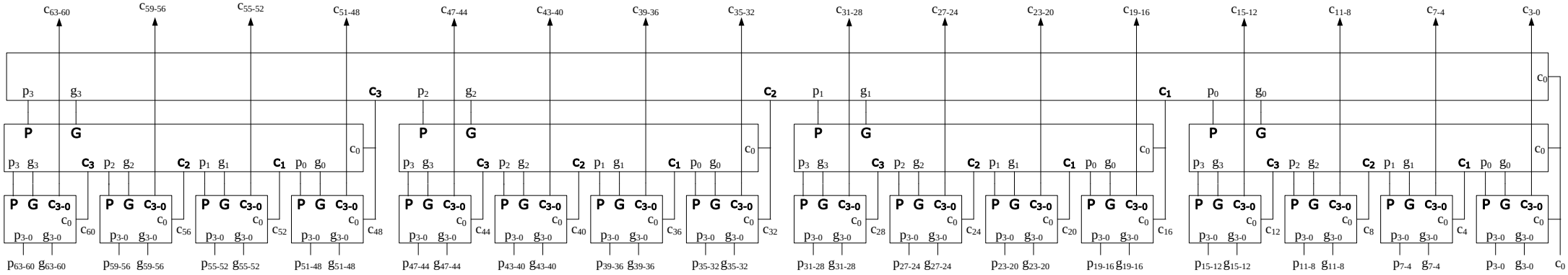
```
    assign P=&p;
    assign G=g[3]|(p[3]&g[2])|(p[3]&p[2]&g[1])|(p[3]&p[2]&p[1]&g[0]);

    assign cout[0]=g[0]|(p[0]&cin);
    assign cout[1]=g[1]|(p[1]&g[0])|(p[1]&p[0]&cin);
    assign cout[2]=g[2]|(p[2]&g[1])|(p[2]&p[1]&g[0])|(p[2]&p[1]&p[0]&cin);
endmodule
```

6. 扩展单精度 扩展双精度，网上有不同的材料，确实说法不一样。这个参考答案取课本中表 3.2 的说法。

参数	格式			
	单精度	扩展单精度	双精度	扩展双精度
尾数位宽 P	23	>=31	52	>=63
指数最大值 Emax	127	>=1023	1023	>=16383
指数最小值 Emin	-126	1-Emax	-1022	1-Emax
指数偏移量 Bias	127	Emax	1023	Emax
指数位数	8	>=11	11	>=15
浮点格式宽度	32	>=43	64	>=79

7.


c63-60 c59-56 c55-52 c51-48 c47-44 c43-40 c39-36 c35-32 c31-28 c27-24 c23-20 c19-16 c15-12 c11-8 c7-4 c3-0
p3 g3 c3 p2 g2 c2 p1 g1 c1 p0 g0 c0
P G c3-0
p3-0 g3-0
p63-60 g63-60 p59-56 g59-56 p55-52 g55-52 p51-48 g51-48
p47-44 g47-44 p43-40 g43-40 p39-36 g39-36 p35-32 g35-32
p31-28 g31-28 p27-24 g27-24 p23-20 g23-20 p19-16 g19-16
p15-12 g15-12 p11-8 g11-8 p7-4 g7-4 p3-0 g3-0

正确性证明略。

8. 答： **概念题**

16 个数相加的华莱士树，按照课本上的画法，共有 14 个进位输入；最后的全加器有一个 cin；最后加法器中，C 和 S 是错位相加的，因此 C 的低位还有一个空位。共计 16 个位置。

第十章 参考答案 高速缓存

1、解： 参考课本内容即可

页大小为 4KB=2^{12}B，页内地址为[11:0]位。

cache 容量 64KB=2^{16}B，地址范围为[15:0]；cache 块大小为 32B=2^{5}B，地址范围为[4:0]。

1）直接相联：

cache 索引位数为地址的[15:5]，需要页着色的是地址[15:12]，共 4 位；

2）二路组相联：

cache 索引位数为地址的[14:5]，需要页着色的是地址[14:12]，共 3 位；

3）四路组相联：

cache 索引位数为地址的[13:5]，需要页着色的是地址[13:12]，共 2 位。

2、解： 代价、损耗等词语，一看就是描述额外开销

$MissPenalty_{L1}=HitTime_{L2} + MissPenalty_{L2} \times MissRate_{L2}$

1）直接相联：

$MissPenalty_{L1}=4 + 60 \times 25\% = 19$ 个时钟周期

2）2 路组相联：

$MissPenalty_{L1}=5 + 60 \times 20\% = 17$ 个时钟周期

3）4 路组相联：

$MissPenalty_{L1}=6 + 60 \times 15\% = 15$ 个时钟周期

3、解： 哪个缺失率高，可以直接按 1.5+38 和 40 进行比较；但是如果算各个 cache 的访问缺失率，则需要按下面的方式算：

1）MissRate = CacheMissOps/MemOps

每 1000 条指令中 load/store 指令的条数是 1000×(26%/74%)=351

32KB 指令 cache MissRate 为 0.0015；

32KB 数据 cache MissRate 为 38/351=0.1083

指令 cache 和数据 cache 各 32KB 的 MissRate = (1.5 + 38)/(1000+351) = 0.0292

64KB 一体 cache 的 MissRate = 40/(1000+351) = 0.0296

所以指令 cache 和数据 cache 各 32KB 组织方式缺失率更低

2）AMAT 的定义存在歧义，参考答案提供其中一种算法：实际执行时间 除以 总访问次数

指令 cache 和数据 cache 各 32KB 的 AMAT = (1000+1.5×100+38×100)/(1000+351)=3.66

64KB 一体 cache 的 AMAT = (1000+351+40×100)/(1000+351)=3.96

4、解： 这个题歧义比较严重。 这个题出的不好， 1.内存访问序列本应是字节为单位，但是参考答案按照“字”为单位。 2.区分容量缺失和冲突缺失的概念不合理：容量缺失和冲突缺失的区分，更适合用来在“缺失率”的意义上做，而非针对某次访问到底是什么类缺失。

地址访问序列（单位：字）	访问类型	
	直接相联	2 路组相联
0	强制缺失	强制缺失
1	命中	命中
2	命中	强制缺失
3	命中	命中
4	强制缺失	强制缺失
15	强制缺失	强制缺失
14	命中	命中
13	命中	强制缺失
12	命中	命中
11	强制缺失	强制缺失
10	命中	命中
9	命中	强制缺失
0	命中	命中
1	命中	命中
2	命中	命中
3	命中	命中
4	命中	命中
56	强制缺失	强制缺失
28	强制缺失	强制缺失
32	强制缺失	强制缺失
15	容量缺失	命中
14	命中	命中
13	命中	冲突缺失
12	命中	命中
0	容量缺失	冲突缺失
1	命中	命中
2	命中	命中
3	命中	命中

5、答： **概念题**

包含式 cache 关系中一级 cache 的内容是二级 cache 内容的真子集；非包含式 cache 关系中一级 cache 的内容与二级 cache 内容无交集。

1）一级 cache 缺失后查找二级 cache：

包含式 cache 关系中，若二级 cache 命中则直接返回，若二级 cache 也缺失，则从更低层次的存储中取回所需数据，填入二级 cache 并返回至一级 cache。在此过程中，二级 cache 无论命中与否均存在一次读访问过程，若不命中，则存在一次替换操作（一次读访问过程读出被替换 cache 块，一次写访问过程写入回填 cache 块）。

非包含式 cache 关系中，若二级 cache 命中则在将命中的 cache 块返回给一级 cache 的同时将该 cache 块从二级 cache 中无效，若二级 cache 也缺失，则从更低层次的存储中取回所需数据，直接返回至一级 cache。在此过程中，二级 cache 无论命中与否均存在一次读访问过程，二级 cache 命中时存在一次写访问过程，不命中时则无其它访问过程。

2）一级 cache 替换出的 cache 块：

包含式 cache 关系中，替换出的 cache 块若不是脏块，则数据不用写回二级 cache，若是脏块，则直接写回二级 cache 的对应位置。在此过程中，若替换出的不是脏块，则一级 cache 只需要将替换出的地址通知一致性管理部件即可，一级 cache 与二级 cache 间无数据通信发生。

非包含式 cache 关系中，替换出的 cache 块无论是否是脏块，都要写入二级 cache。在将此 cache 块写入二级 cache 时，可能先要替换出一个 cache 块。在此过程中，一级 cache 与二级 cache 间有数据通信发生。

3）硬件实现复杂性方面，略。

4）多核之间一致性维护：（此处考虑二级 cache 被多个处理器核共享的情况）

如果采用基于目录的协议：

包含式 cache 关系中，因所有一级 cache 的内容均在二级 cache 中有备份（可能数据是旧值），所以二级 cache 处可以获得足够的信息来维护多核间的 cache 一致性。而在非包含式 cache 关系中，一级和二级 cache 上发生的所有 cache 块操作的事件的消息都要传递给目录进行维护。

如果采用基于侦听的协议：

包含式 cache 关系中，每个核向外广播的无效请求或更新请求也要传递给二级 cache。而在非包含式 cache 关系中，每个核向外广播的无效请求或更新请求则不必传递给二级 cache。

6、答： **概念题**

1）几乎没有空间局部性和时间局部性：数据库查询操作中被查询的数据

2）较好的空间局部性，几乎没有时间局部性：音视频编解码等流式应用被处理的数据

3）较好的时间局部性，很少的空间局部性：以指针链表、树、图等数据结构开发的应用程序。

4）较好的空间和时间局部性的应用：针对 cache 优化后的矩阵乘法、大量的科学计算、压缩解压缩、加解密、文字处理

7、答： **概念题**

降低 cache 缺失代价的技术：1）读优先；2）关键字优先；3）写合并；4）Victim Cache；5）多级 cache。（降低原因参考教材 P191~192，略）

降低 cache 缺失率的技术：1）增加 cache 容量；2）增加 cache 块大小；3）提高 cache 相联度；4）软件优化。（降低原因参考教材 P188~189，略）

8、答： **概念题**

1） 直接相联：每个内存块只能映射到 cache 的唯一位置；

全相联：每个内存块可以映射到 cache 的任一位置；

组相联：每个内存块对应 cache 的唯一的一个组，但可以映射到组内的任一位置。

2） 直接相联：根据访问地址将唯一映射位置上的 cache 块读出，将地址高位与 tag 比较看是否相等；

全相联：将 cache 中所有 cache 块读出，将地址高位与每一项的 tag 比较，看哪一项相

等；

组相联：根据访问地址将唯一映射的组内的所有 cache 块读出，将地址高位与每一项的 tag 比较，看哪一项相等。

3）LRU、Random、FIFO。

4）写穿透策略和写回策略；写分配策略和写不分配策略。

第十一章 参考答案 存储管理

1. 在如下一段 C 语言程序的 for 循环中，

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

程序memcpy函数执行过程中可能发生哪些例外，各多少次。

答： 题目中没有给TLB项数和替换算法，参考答案按项数足够计算，但各位同学需要掌握项数不足情况的计算公式。题目中没有考虑modify例外和a数组已在父函数进行过访问的情况，题干不是很严谨。

【说明：此题中试图问的是与页处理相关的例外（忽略实际系统中可能发生的其他内部外部中断），以MIPS体系结构为例，包括三类共四种例外：

(1) TLB refill 例外；

(2) TLB invalid Load例外 / TLB invalid Store例外；

(3) TLB modify 例外。

其中(2)和(3)两类例外仅与程序对于页的访问需求相关，因此回答次数时需要假设操作系统中页的大小；而(1)还与处理器中TLB的容量相关，因此还需要假设处理器中TLB的项数和TLB的替换策略】。

下面给出一种常用假设条件下的回答：

操作系统页大小为4KB，同时系统的物理内存足够大，即页表分配的物理内存在程序执行过程中不会被交换到swap区上。并且在后续的分析中忽略代码和局部变量i所在的页的影响。a、b两数组大小均为65536×8=512KB。再假设a、b两数组的起始地址恰为一页的起始地址。

那么a、b两数组各自均需要512KB/4KB=128个页表项。

所以a、b的访问造成的TLB invalid 例外次数为128+128=256次；

假设TLB表项为128项，每项映射连续的偶数奇数两个页，采用LRU算法。那么第一次次循环中，a、b两数组每访问相邻偶数奇数两页的首地址时，均会触发一次TLB refill例外。后续各次循环TLB中均命中。那么共触发的TLB refill例外次数为128/2 + 128/2 = 128次。

2. 对于指令 Cache 是否有必要考虑 Cache 别名问题？

答： **概念题**

有必要。

1、当程序中有自修改代码时，可能会由于 cache 别名导致取错指令。

2、由于 Last Level Cache 通常是用物理地址索引的，那么当一级指令 cache 存在别名问题时，整个处理器无法维护一级指令 cache 和 Last Level Cache 之间相同物理地址的数据备份之间的关系，导致处理器进入混乱状态，最终导致程序执行错误甚至死机。

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？

答：参考课本内容即可得

1、由于虚拟地址有 64 位，进程标识为 8 位；而物理地址是 64GB，即需要 36 位物理地址。这样，使用虚拟地址比使用物理地址总共增加的位数为 8＋（64－36）＝8＋28＝36 位。
2、改变块的大小，亦或改变 cache 的其它参数，cache 占据的地址的低位的长度对于虚地址索引和物理地址索引都是一样的，所以对于 cache 的 tag 而言，两中索引方式相差位数保持不变。

4. 考虑一个包含 TLB 的当代处理器。（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，这样的设计合理么？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。

答：概念题。参考答案给出一个合理解答：（在另一种解答中，TLB 命中指的是硬件上 TLB 查询 VPN 命中。而 TLB refill 填回一个 V=0 的非法页，然后才会触发 page fault，因此此时 TLB 命中（但是 V 为 0）且触发 page fault。第二问如何回答，与“page fault”、“TLB 命中”的概念定义有关。）

1） **TLB**：Translation lookaside buffer，即旁路转换缓冲，或称为页表缓冲；里面存放的是一些页表文件（虚拟地址到物理地址的转换表）的一个子集。

TLB 失效例外：假如某个虚地址 VA，在 TLB 中没有找到对应的页表项，则发生 TLB 失效

例外(TLB miss exception)。

页表：简单的说，页表是内存块的目录文件，作用是实现从逻辑页号到物理块号的地址映射。即用指定大小（称之为页）来划分程序逻辑地址空间和处理机的物理内存空间，并建立起从逻辑页到物理页的**页面映射**关系，这些页面映射就是一张张页表（Page Table）。

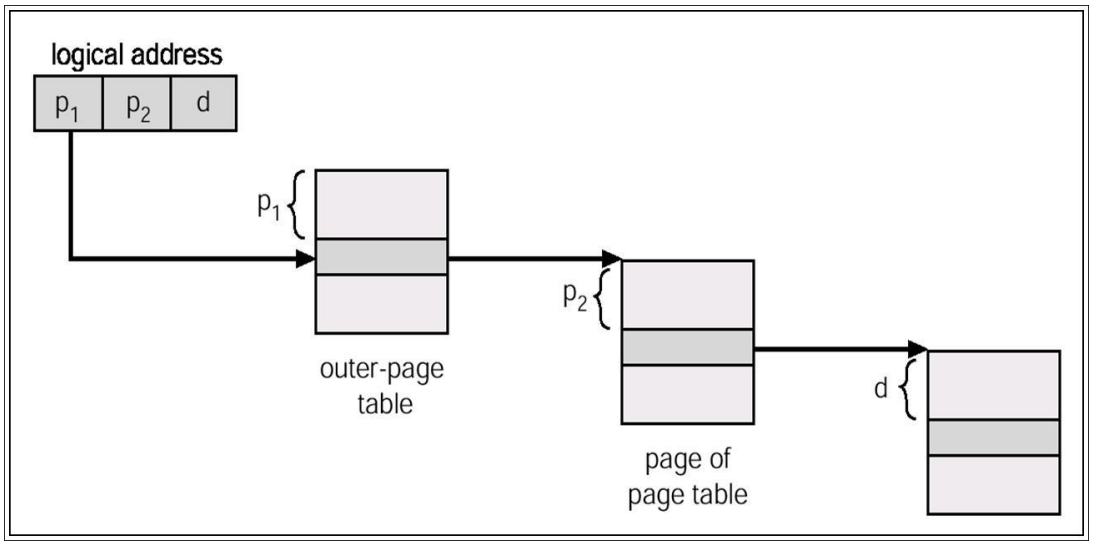
Page Fault：缺页，就是 TLB 中没有虚地址 VA 对应的页表，主存中也没有 VA 对应的物理地址。这时候发生 Page Fault Exception（缺页例外），需要交给操作系统处理，通常需要消耗很多的时间来处理该例外。

2) 不合理，因为 Page Fault 发生时意味着主存中没有虚地址 VA 对应的页表，这时 TLB 中也应该没有，因为 TLB 里的页表是主存中页表的子集，应该发生 TLB miss。倘若没有发生，说明 TLB 不是主存页表的子集，这会导致存储不一致，也违背了层次化设计的基本要求。

3) 多级页表可以减少页表所占用的存储空间。比如：

一个 32 位逻辑地址空间的计算机系统，页大小为 4KB，其虚地址可以被分为：20 位的页号 + 12 位的偏移。那么单级页表的条目有 2^{20} 条。假设每个条目占 4B，则需要 4MB 连续物理地址空间来存储页表本身。

如果采用两级页表，假设将 20 位的页号分解为：10 位的外部页表号 + 10 位的外部页表偏移。如下图所示，逻辑地址中：p1 用来访问外部页表的索引，p2 是外部页表的页偏移。其中第一级页表需要 4KB 连续物理地址空间，第二级页表每张表也需要 4KB 连续物理地址空间，由于通常情况下第一级页表中并非每一项都需要相应的第二级页表，所以总的存储空间要少于单级页表。



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。

答： Verilog 题

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

input [33:0] vpn_in;
output [25:0] pfn_out;
output hit;
output valid_out;

reg[60:0] cam_content [63:0];//[60:60] valid [59:34] pfn [33:0] vpn
wire [63:0] entry_hit;

assign entry_hit[0] = (vpn_in==cam_content[0][33:0]);
assign entry_hit[1] = (vpn_in==cam_content[1][33:0]);
……
assign entry_hit[63] = (vpn_in==cam_content[63][33:0]);

assign hit = |entry_hit;

assign pfn_out =  {26{entry_hit[ 0]}} & cam_content[0][59:34] |
                  {26{entry_hit[ 1]}} & cam_content[1][59:34] |
……
                  {26{entry_hit[63]}} & cam_content[63][59:34];

assign valid_out = entry_hit[ 0] && cam_content[ 0][60] ||
                   entry_hit[ 1] && cam_content[ 1][60] ||
                   …
                   entry_hit[63] && cam_content[63][60];
endmodule
```

第十二章 参考答案 多处理器

1、答： **用 C 写的算法：（不在本课考察范围内）**

（1） Filter Lock 算法：

```
int level[p];
int victim[p]; 11  g      /
for(int i = 0; i < p; ++i)
```

```
{
    level[i] = 0;
}

void lock(int tid)
{
    for (int i=1; i<p; i++)
    {
        level[tid] = i;
        victim[i] = tid;
        while (same_or_higher(tid, i) && victim[i]==tid);
    }
}

void unlock(int tid)
{
    level[tid] = 0;
}

bool same_or_higher(int tid, int i)
{
    for (int k=0; k<p; k++)
        if (k!=tid && level[k] >= i) return true;
    return false;
}
```

（2）Lamport's bakery 算法：

```
int number[p];
bool entering[p];
for(int i = 0; i < p; ++i)
{
    number[i] = 0;
    entering[i] = false;
}

void lock(int tid)
{
    entering[tid] = true;
    number[tid] = 1 + array_max(number); 11h  f  c       g c  c
    entering[tid] = false;
    for (int i=0; i<p; i++)
    {
        if (i != tid)
        {
```

```
            11  c             gcf    geg  g          dg
            while (entering[i]);
            11  c          c      gcf          c  g     dg
            11            g  c g    dg . d          i g          .
            11 h         g
            while ((number[i]!=0) && (number[tid]>number[i] ||
                    (number[tid]==number[i] && tid>i)));
        }
    }
}

void unlock(int tid)
{
    number[tid] = 0;
}
```

（3）最少需要访问 p 个内存地址。假设只需要 i 个地址，存在某个时刻，p 个进程都要向这 i 个地址写以表明自己想获得锁。如果 i<p, 必然有某个进程 p_j 希望获得锁的意图会被其它进程的写覆盖掉，从而其它进程不知道 p_j 希望获得锁。这样就会产生不公平。

我用 LLSC 汇编写的（有很多种参考答案，以下只是一种）:（请掌握其原理）

(1) 简单自旋锁，设 a0 为锁的地址

```
Lock:
  1:  LL      t0, 0x0(a0)    //取回锁值，0 表示未被锁，1 表示有人持有锁
      BNEZ    t0, 1b         //如果已经被锁，则自旋等待
      ADDIU t0, t0, 0x1      //将 t0 改为已持有
      SC      t0, 0x0(a0)    //存回
      BEQZ    t0, 1b         //检查是否成功，否则重做
      NOP

Unlock:
      //mem fence
      SW    $0, 0x0(a0)   //解锁时写 0
      //mem fence
```

(2) 简单自旋排队锁　设 a0 为锁的地址。按请求顺序来 service。

```
Lock:  (锁值为高低各 2 字节组成，高位为 tickets, 低位为 serve)
   1: LL      v0, 0x0(a0)     //同时取回高低位
      LI      t0, 0x10000     //将高位 tickets 加一
      ADDU    t0, t0, v0
      SC      t0, 0x0(a0)     //存回去
      BEQZ    t0, 1b          //检查是否 LLSC 成功，失败则重试
      NOP
```

```
    SRL     v0, v0, 16      //取出此时已经确定拿到的 tickets，放到低位
2:  LHU     t0, 0x0(a0)     //检查当前 serve 值
    BNE     t0, v0, 2b      //如果当前 serve 不是自己，则循环等待
    NOP

Unlock：
    //mem fence
    LHU     t0, 0x0(a0)   //取出当前 serve
    ADDIU   t0, t0, 0x1
    SH      t0, 0x0(a0)   //加 1 存回去
    //mem fence
```

(3) 需要两个内存地址，一个记录当前服务者，另一个记录当前排队最晚号码；同时，每个进程需要自己使用寄存器或内存地址，记录自己拿到的号码。

2、答： **概念题+MIPS 汇编代码题**

（1）Compare_And_Swap(CAS)和 Load-Linked and Store-Conditional(LL/SC)是两种比较常见的硬件同步原语。例如，Intel、AMD 的 x86 指令集和 Oracle/SUN 的 Sparc 指令集实现了 CAS 指令；Alpha、PowerPC、MIPS、ARM 均实现了 LL/SC 指令。

（2）CAS 指令硬件实现比 LL/SC 的硬件实现复杂。使用 CAS 指令会碰到 ABA 问题，但是 LL/SC 指令不会碰到该问题。LL/SC 指令中由于 SC 是尝试去写，因此在某些情况下，SC 执行成功率很低，导致用 LL/SC 实现的锁执行开销变得很大。

（3）下面仅给出用 CAS 和 LL/SC 实现普通自旋锁的例子。公平的自旋锁可利用这些普通的自旋锁实现。

CAS：指令定义 cas r1, r2, Mem；r1 存放期望值，r2 存放更新值，

获得锁：//锁初始化为 0

```
    la      t0, sem
TryAgain:
    li      t1, 0
    li      t2, 1
    cas     t1, t2, 0x0(t0)
    bnez    t1, TryAgain
    nop
```

释放锁：

```
    la      t0, sem
    li      t1, 0
    sw      t1, 0x0(t0)
```

LL/SC：

获得锁：//锁初始化为 0

```
    la      t0, sem
TryAgain:
    ll      t1, 0x0(t0)
    bnez    t1, TryAgain
    li      t1, 1
```

```
    sc      t1, 0x0(t0)
    beqz    t1, TryAgain
    nop
```

释放锁：

```
    la      t0, sem
    sw      zero, 0x0(t0)
```

（4）LL/SC 需要 n 个地址才能实现公平。CAS 只需要两个地址。最主要的是保证每个进程希望获得锁的信息不会被覆盖掉。CAS 可以保证，而 LL/SC 可能会失败。

3、答： **概念题**

（1）n 个共享存储处理器，每个有 m 内存。进行 O(nm)个变量排序就可能出现超线性加速比。一些随机算法也可能出现。很多搜索算法也会有超线性加速比，例如通过剪枝使得总的被搜索数据量少于顺序搜索的情况，此时并行执行将完成较少的工作。

（2）Amdahl 定律定义是：使用某种较快执行方式获得的性能提高，受到可受这种较快执行方式的时间所占比例的限制。Speedup = 1 / ((1-p)+k*p)。加速因子 k 可以比 1/n 增长快，即所谓超线性。因此并不矛盾。

4、答：**拓展题**

避免多个处理器同时写的不同变量处于同一 cache 行上。

5、答： **概念题**

（1）可能 P1 已经把 A 替换出去了，随后 P1 又希望访问 A，因此向目录发出了 A 的 miss 请求。如果网络不能保证 P1 发出的替换 A 消息和 miss 访问 A 消息达到目录的顺序，后发出的 A 的 miss 请求越过先发出的 A 的替换请求，先到达目录，就会产生上述现象。

（2）两种可行的处理方式：a)网络保证点到点消息的顺序性；b)目录发现不一致时，暂缓 miss 请求的处理，等待替换消息到达后，目录状态正确后，再返回 miss 请求的响应。

6、答： **请掌握 MSI 一致性协议。**

事件	A 状态	B 状态
初始	I	I
CPU A 读	S	I
CPU A 写	M	I
CPU B 写	I	M
CPU A 读	S	S

7、答： **概念题**

（1）P1 对 A 的写被网络阻塞，一直没有传播到 P2 和 P3。

（2）无须施加限制。

（3）

```
A=2000          while (B!=1) {;}        while (C!=1) {;}
B=1             C=1;                    D=A
```

8、答： **概念题**

（1）顺序一致性：

i.a=1 先于 print a 执行，b=1 先于 print b 执行。最终结果：a=1, b=1

ii.a=1 先于 print a 执行，print b 先于 b=1 执行。最终结果：a=1, b=0

iii.print a 先于 a=1 执行，b=1 先于 print b 执行。最终结果：a=0, b=1

（2）弱一致性：

除顺序一致性中的三种执行序外，还可以有第 iv 中执行序：

print a 先于 a=1 执行，print b 先于 b=1 执行。最终结果：a=0, b=0