**1、CPU性能公式及提高CPU性能的主要方法。**

CPU时间=指令数×CPI×时钟周期时间

CPI：每条指令的时钟周期数，执行某个程序、程序片段时每条指令所需的时钟周期平均数。

提高CPU速度的方法：减少指令数，降低CPI，增加时钟频率。提高CPU的频率，缩短时钟周期时间；采用较科学的编程方法，减少指令数。

具体来说，可以：

1. 优化算法，降低指令数和CPI

（1）算法决定源程序执行指令的数目，从而也决定了CPU执行指令的数目；（2）算法也可能通过使用较快或较慢的指令影响CPI。

1. 改变编程语言，降低指令数和CPI

（1）编程语言的语句必须翻译为指令，从而决定了指令数；

（2）编程语言也影响CPI。

1. 优化编译程序，降低指令数和CPI，编译程序的效率，既影响指令数，又影响CPI。因为编译程序决定了最后生成的计算机指令，所以会以复杂的方式影响CPI。

**影响Cache性能主要的因素有哪些。**

Cache的容量；

Cache的控制算法；

Cache的结构

cache相联度；

cache块大小；

不同的替换算法：LRU和随机法FIFO等。

**2、Cache失效主要原因及解决方法。**

Cache失效主要原因：

1. 强制性失效 (Compulsory)

第一次访问某一块，只能从下一级Load，也称为冷启动或首次访问失效

1. 容量失效（Capacity）

如果程序执行时，所需块由于容量不足，不能全部调入Cache， 则当某些块被替换后，若又重新被访问，就会发生失效。可能会发生“抖动”现象

1. 冲突失效（Conflict (collision)）

组相联和直接相联的副作用，若太多的块映像到同一组（块）中，则会出现该组中某个块被别的块替换（即使别的组或块有空闲位置），然后又被重新访问的情况，这就属于冲突失效

解决方法：

**降低失效率**：1、增加Cache块的大小 2、增大Cache容量 3、提高相联度

增大Cache容量：对冲突和容量失效的减少有利

增大块：减缓强制性失效、可能会增加冲突失效（因为在容量不变的情况下，块的数目减少了）

通过预取可帮助减少强制性失效：必须小心不要把你需要的东西换出去、需要预测比较准确（对数据较困难，对指令相对容易）

提高相联度，会增加命中时间

**减少失效开销**：4、多级Cache 5、使读失效优先于写失效

多级cache的优点：减少失效开销、缩短平均访存时间（AMAT）

由于读操作为大概率事件，需要读失效优先，以提高性能

**缩短命中时间**： 6、避免在索引缓存期间进行地址转换

**3、Tomasulo算法基本思想及如何解决流水线中的存在的数据冲突。**

Tomasulo算法是将寄存器数据提取到保留站中，如果原来寄存器中的操作数修改了，就将修改后的值提取到另外一个保留站中，以此类推。在命令发射的时候，会将待用操作数的寄存器说明符更名为它对应保留站的名字。其核心思想是：记录和检测指令相关，操作数一旦就绪就立即执行，把发生RAW（写后读）冲突的可能性减少到最少。通过寄存器换名来消除WAR（读后写）和WAW（写后写）冲突。

主要解决的问题：【通过寄存器换名消除WAR和WAW冲突】如果此时需要的源数据已经准备好则直接从相应寄存器送到保留站对应位置，等待功能单元空闲即运行，如果源数据没准备好，则在保留站对应区域设置指向该数据寄存器的指针，等数据准备好，则直接通过CDB总线送到保留站，而不需等待数据写回寄存器。

**解决数据冲突：**WAW和WAR可以通过寄存器重命名的方法将相关性去除，从而无需担心其执行顺序。寄存器重命名技术在Tomasulo算法中通过保留站和ROB（Re-Order Buffer）完成，或者采用纯物理寄存器（而不用ROB）的方式完成。

**4、如何实现Cache一致性？**

a）通过在总线加LOCK#锁的方式：使得只能有一个CPU能使用这个变量的内存。在总线上发出了LCOK#锁的信号，那么只有等待这段代码完全执行完毕之后，其他CPU才能从其内存读取变量，然后进行相应的操作；

b）通过缓存一致性协议：Intel的MESI协议，保证了每个缓存中使用的共享变量的副本是一致的。当CPU写数据时，如果发现操作的变量是共享变量，即在其他CPU中也存在该变量的副本，会发出信号通知其他CPU将该变量的缓存行置为无效状态，因此当其他CPU需要读取这个变量时，发现自己缓存中缓存该变量的缓存行是无效的，那么它就会从内存重新读取。

cache的操作有2种：写回和无效。写回操作是将cache中数据写回到DDR中，无效操作是无效掉cache中原有数据，下次读取cache中数据时，需要从DDR中重新读取。这两种操作其实都是为了保证cache数据一致性。

**5、简述虚拟地址翻译机制及如何提高其性能。**

答：地址翻译指的是DRAM缓存命中时，由虚拟地址找到物理地址的过程。该过程是完全由硬件来完成的。

1)CPU有一个专门的页表基地址寄存器(PTBR)指向当前页表的基地址，快速定位到该进程的页表。

2)根据虚拟页号，找到虚拟地址在页表的值。

3)根据值中的物理页号，找到物理地址。

4)得到实际物理地址后，根据高速缓存的原理，把一个物理地址映射到高速缓存具体的组，行，块中，找到实际存储的数据。

（如果你们觉得这个版本少的话，可以用这个版本：

1)CPU拿到一个虚拟地址，分为两步，先通过页表机制确定该地址所在虚拟页的内容是否从磁盘加载到物理内存页中，然后通过高速缓存机制从该物理地址中取到数据

2)地址翻译硬件要把这个虚拟地址翻译成一个物理地址，从而可以再根据高速缓存的映射关系，把这个物理地址对应的值找到

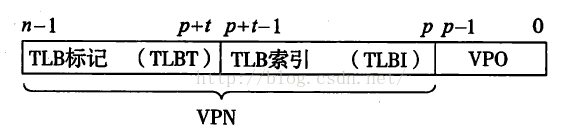
3)地址翻译硬件利用页表数据结构，TLB硬件缓存等技术，目的只是把一个虚拟地址映射到一个物理地址。要记住DRAM缓存是全相联的，所以一个虚拟地址和一个物理地址是动态关联的，不能直接根据虚拟地址推导出物理地址，必须根据DRAM从磁盘把数据缓存到DRAM时存到页表时存的实际物理页才能得到实际的物理地址，用物理页PPN + VPO就能算出实际的物理地址 (VPO = PPO，所以直接用VPO即可)。 PPN的值是存在页表条目PTE中的。地址翻译做了一堆工作，就是为了找到物理页PPN，然后根据VPO页面偏移量，就能定位到实际的物理地址。

4)得到实际物理地址后，根据高速缓存的原理，把一个物理地址映射到高速缓存具体的组，行，块中，找到实际存储的数据。）

**提高其性能**

利用高速缓存及TLB加速地址翻译

1)在地址翻译的过程中，有一个步骤是通过页表条目地址来查找PTE，我们可以通过结合高速缓存来提高其查找速度；

2) TLB是专门用于虚拟寻址的缓存，称为翻译后备缓冲器，由标记（TLBT）和索引（TLBI）组成。

**6、Amdal定理及其应用**

Amdahl定律：系统中对某一部件采用更快执行方式所能获得的系统性能改进程度，取决于这种执行方式被使用的频率，或所占总执行时间的比例。

Amdahl定律定义了穿行系统并行化后加速比的计算公式和理论上限：

加速比 = 优化前的系统耗时/优化后的系统耗时。

Amdahl定律定义了由于采用特殊的方法所能获得的加速比的大小：

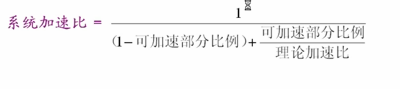
Speedup= 1/[(1-增强比例)+(增强加速比/增强比例)]

应用：

1）苹果手机的单个硬件指标并不是最突出的，拼单个硬件的话跟同价位甚至是低价位的手机都比不过。但是苹果并不追求单个硬件指标的性能突出，而是追求整体性能的均衡，包括硬件和软件完美结合，因此在用户体验方面，苹果手机优于很多其他手机。

2）2008年，《利用Amdahl定律分析和提高画云图的效率》论文中提出，利用Amdahl定律结合一个画云图的程序设计实例，针对其存在的严重效率问题，应用Amdahl定律来分析、定位和改进、提高效率，改进之后程序运行速度比改进之前提高了200多倍。

**举例说明什么是****Amdal定理。**



Amdal定理:系统中对某一部件采用更快的执行方式所获得的性能改进程度，取决于这一方式所被使用的频率，或占总时间的比例。

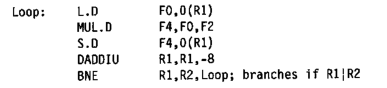
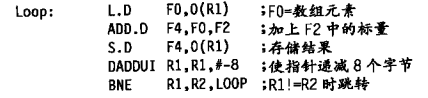
简而言之，通过更快的处理器所获得的加速会被慢的系统组件所限制。

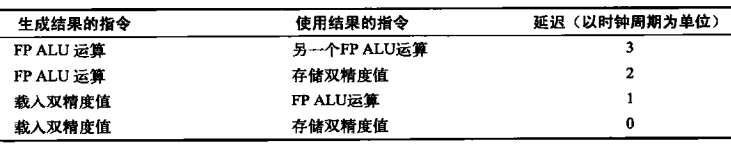
公式:S=1/(1-a+a/n)

a表示系统中并行计算所占的比例，n为并行处理结点个数。当a=1时，系统中全为并行计算，没有串行，此时s=n，系统的加速比最大。a=0时，系统中全为串行计算，没有并行操作，此时s=1，加速比最小。

当n→∞时，极限加速比s→ 1/（1-a），这也就是加速比的上限。

**7、指令静态调度及循环展开**





|  |  |  |  |
| --- | --- | --- | --- |
| LOOP： | L.D | F0,0(R1) | 1 |
|  | L.D | F6,-8(R1) | 2 |
|  | L.D | F10,-16(R1) | 3 |
|  | L.D | F14,-24(R1) | 4 |
|  | MUL.D | F4,F0,F2 | 5 |
|  | MUL.D | F8,F6,F2 | 6 |
|  | MUL.D | F12,F10,F2 | 7 |
|  | MUL.D | F16,F14,F2 | 8 |
|  | S.D | F4,0(R1) | 9 |
|  | S.D | F8,-8(R1) | 10 |
|  | DADDIU | R1,R1,#-32 | 11 |
|  | S.D | F12,16(r1) | 12 |
|  | S.D | F16,8(R1) | 13 |
|  | BNE | R1,R2,Loop | 14 |

展开后循环的执行时间缩减到总共14个时钟周期，每个元素3.5个时钟周期。

指令静态调度：静态调度是指通过改变指令顺序而不改变指令间数据相关来改善指令延迟。在出现数据相关时，为了消除或者减少流水线空转，编译器确定并分离出程序中存在的相关指令，然后进行指令调度，并对代码进行优化。它不是在程序执行的过程中、而是在编译期间进行代码调度和优化的。

循环展开：循环展开是将循环中多个基本块展开成一个基本块从而填充stall间隙的方法，这是一种牺牲程序的尺寸来加快程序的执行速度的优化方法。它可以由程序员完成，也可由编译器自动优化完成。循环展开通过将循环体代码复制多次实现。循环展开能够增大指令调度的空间，减少循环分支指令的开销。循环展开可以更好地实现数据预取技术。但是由于循环展开通过减少循环次数来提高程序性能，如果展开次数太多，反而会造成性能急剧下降。因为展开次数太多，那么运算过程中中间变量会很多，而计算机的寄存器个数是固定的，当变量个数超了寄存器，那么变量只能存到栈中从而导致性能下降。

**8、相变内存（PCM）**

相变内存，通常称为PCM技术或相变RAM技术。之所以被称为相变内存，是因此它利用特殊材料（包括硫、硒或者碲等，目前主要的研究方向是硫属化物玻璃）在晶态和非晶态之间相互转化时所表现出来的导电性差异来存储数据。它是一种非易失性的内存产品。相变内存是下一代内存（闪存）技术。相变内存结合了DRAM内存的高速存取，以及闪存在关闭电源之后保留数据的特性，被业界视为未来闪存和内存的替代品。人们需要一种能够保存数据的DRAM，而相变内存将满足这一需求。相变内存的另外一个优点是，可以在不删除现有数据的情况下写入数据，这比如今的内存更为快捷。根据统计，相变内存的功耗只有现有闪存的一半，但是读写速度可以达到闪存的1000倍。

**9、RISC-V开源处理器**

RISC-V是一个基于精简指令集（RISC）原则的开源指令集架构（ISA）。与大多数指令集相比，RISC-V指令集可以自由地用于任何目的，允许任何人设计、制造和销售RISC-V芯片和软件。虽然这不是第一个开源指令集，但它具有重要意义，因为其设计使其适用于现代计算设备（如仓库规模云计算机、高端移动电话和微小嵌入式系统）。设计者考虑到了这些用途中的性能与功率效率。该指令集还具有众多支持的软件，这解决了新指令集通常的弱点。RISC-V指令集的设计考虑了小型、快速、低功耗的现实情况来实做，但并没有对特定的微架构做过度的设计。

**10、Web服务器普遍采用如Javascript、python、PHP等动态脚本语言。为加速这些语言中的对象属性访问，采用了硬件机制来缓存这些属性，使得访问时间减小为原来的1/10。问：（1）假定对象属性访问指令时间占总时间40%，则加速后系统性能提高了多少？（2）如果要使加速后系统性能提高1倍，则对象属性访问指令时间占总时间为多少？**

（1）升级比例=0.4 升级加速比=10

答：系统性能大约提高了56%

（2）解：设升级比例为x 升级加速比=10

解得x=

这里的加速比应该是10还是11，我不确定

**11、请用VerilogHDL语言给出课内实验2中4指令处理器控制器的状态机实现。**

//状态机设计的例子

module FSM(clk,clr,out,start,step2,step3);  
input clk,clr,start,step2,step3;  
output[2:0] out;  
reg[2:0] out;  
reg[1:0] state,next\_state;  
 parameter   state0=2'b00,state1=2'b01,  
state2=2'b11,state3=2'b10;  
  
always @(posedge clk or posedge clr)      
begin  
   if (clr) state <= state0;  
   else state <= next\_state;

end  
 always @(state or start or step2 or step3)      
begin      
case (state)  
   state0: begin  
     if (start)  next\_state <=state1;  
     else       next\_state <=state0;  
     end  
   state1: begin  
           next\_state <= state2;  
     end  
   state2: begin  
     if (step2)  next\_state <=state3;  
     else       next\_state <=state0;  
     end  
   state3: begin  
     if (step3)  next\_state <=state0;  
     else       next\_state <=state3;  
     end  
   default:    next\_state <=state0;    
 endcase  
end  
always @(state)           
begin  
case(state)  
     state0: out=3'b001;  
     state1: out=3'b010;  
     state2: out=3'b100;  
     state3: out=3'b111;  
     default:out=3'b001;      
endcase  
end

**12、（1）什么是局部性原理。；**

局部性通常有两种不同的形式：时间局部性和空间局部性。在一个具有良好时间局部性的程序中，被引用一次的存储器位置很有可能在接下来时间内再被多次引用。在已给具有良好空间局部性的程序中，若一个存储器位置被引用了一次，那么在接下来时间内将引用附近的一个存储器位置。

**（2）比较Cache块的三种放置方法优缺点。**

（1）全相联映射方式： 灵活，但映射函数复杂，不易实现。

（2）直接映射方式： 映照简单，不需计算，快速，但效率不高，易颠簸;。 （3）组相联映射方式： 组内全相联映射，组间直接映射

**13举例说明流水线中的几种冲突。**

流水线冲突有三种类型：结构冲突、数据冲突、控制冲突。

（1）结构冲突

定义：因硬件资源满足不了指令重叠执行的要求而发生的冲突。

例子：计算资源不足（加法器不够多）以使程序并行（加法）计算 ，造成的结构停顿。

（2）数据冲突

定义：当指令在流水线重叠执行时，因需要用到前面的执行结果而发生的冲突。

分类：写后读冲突、写后写冲突、读后写冲突。

例子：写后读，因为用到了前面操作的结果，造成的RAW停顿，如果读操作发生在写操作之前会错误的获得旧值。

（3）控制冲突

定义：流水线遇到分支指令和其他会改变PC值的指令所引起的冲突。

例子：不能把IF语句中的THEN部分中的一条指令拿出来，移到这个分支之前，因此会造成控制停顿。

**14、简述V-MIPS向量计算机的基本结构。**

向量寄存器（Vector registers）：每个向量寄存器都是一个定长的寄存器组，能够容纳一个向量。VMIPS有 8 个向量寄存器，每个寄存器能够容纳 64 个向量元素，每个元素宽度为64位。每个寄存器至少有两个读端口和一个写端口。这能够保证需要不同向量寄存器的多个向量运算能够互相同步进行[1]（我们并不考虑由于寄存器端口短缺而引起的问题。在实际的机器中，这会导致结构冲突）。总共 16 个读端口和 8 个写端口通过一对 crossbar（交叉交换器） 和功能单元的输入输出相连。

向量功能单元（Vector functional units）：每个单元都是完全流水化的，并且每一个新的时钟周期可以开始对于一个新向量元素的操作。另外还需要一个控制部件来检测 hazard，包括功能单元的结构冲突和寄存器访问的数据冲突。如图所示，VMIPS 有五个功能单元。为了简化讨论，我们将只讨论浮点运算单元。取决于不同的设计，标量运算可能使用向量功能部件，或者有单独的一组功能部件。我们这里假设功能部件是共享的，并且，再一次忽略任何可能的冲突。

向量访存（载入/存储）单元（Vector load-store unit）：这是一个能够 load 和 store一个向量到主存的单元。VMIPS 中的 load 和 store 操作是完全流水化的，因此在一开始的延迟之后，寄存器和内存之间的带宽可以达到每一个时钟周期一个 word。这个单元通常也处理标量访存工作。

标量寄存器集合（scalar registers）：标量寄存器能为标量功能单元提供输入数据[3]，并且也能为内存访问单元提供地址。这些其实就是MIPS中常见的 32 个通用寄存器和 32 个浮点寄存器。标量数据从标量寄存器中读出，然后锁存到向量单元的一个输入之中。

1. **并行处理系统的主要挑战有哪些？**

答案一：

（1）程序中的并行性有限

（2）相对较高的通讯开销

答案二：

1. 芯片（2）可用性（3）系统管理（4）调试（5）性能预测（6）编程模型（7）功耗

你自己看着编吧，实在不行自己找答案。

1. **机器学习中普遍采用GPU来进行计算加速，业界也提出了一些专用的计算架构来处理如深度学习中的大量计算，如Google提出针对TensorFlow计算框架的张量处理器（TPU）。请比较GPU和TPU的异同。**

相同点: 都是可以运行绘图运算工作的[微处理器](https://zh.wikipedia.org/wiki/%E5%BE%AE%E8%99%95%E7%90%86%E5%99%A8)。

差别:TPU与[图形处理器](https://zh.wikipedia.org/wiki/%E5%9C%96%E5%BD%A2%E8%99%95%E7%90%86%E5%99%A8)（GPU）相比，TPU采用低精度（8位）计算，以降低每步操作使用的[晶体管](https://zh.wikipedia.org/wiki/%E6%99%B6%E4%BD%93%E7%AE%A1)数量。降低精度对于深度学习的准确度影响很小，但却可以大幅降低功耗、加快运算速度。同时，TPU使用了[脉动阵列](https://zh.wikipedia.org/w/index.php?title=%E8%84%89%E5%8A%A8%E9%98%B5%E5%88%97&action=edit&redlink=1)的设计，用来优化[矩阵乘法](https://zh.wikipedia.org/wiki/%E7%9F%A9%E9%98%B5%E4%B9%98%E6%B3%95)与[卷积](https://zh.wikipedia.org/wiki/%E5%8D%B7%E7%A7%AF)运算，减少[I/O](https://zh.wikipedia.org/wiki/I/O)操作。此外，TPU还采用了更大的片上内存，以此减少对[DRAM](https://zh.wikipedia.org/wiki/DRAM)的访问，从而更大程度地提升性能。