### 第八章 存储器系统

#### 8.1 概述

计算机解决问题的能力受它的存储器系统和输入/输出设备的影响，例如显示器、键盘和打印机，这些设备使我们可以操作和查看计算机的计算结果。本章内容探讨这些实际存储器和I/O系统。

计算机的性能依赖于处理器微结构，同时也依赖于存储器系统。第七章假想了一个可以在单时钟周期内访问的理想存储器系统。然而，这种假想只有在非常小的存储器或非常低速的处理器时才会成立。早期的处理器相对较慢，存储器能跟上其速度。但是处理器速度的增长比存储器速度要快。当前，处理器速度是DRAM存储器速度的10倍到100倍。对于处理器和存储器之间不断增大的速度差异，需要借助于巧妙的存储器系统来和处理器的速度匹配。这一章节将研究实际的存储器系统，并考虑速度、容量、成本之间的折衷。

处理器通过存储器接口（memory interface）与存储器系统相连。图8.1为多周期MIPS存储器中使用的简单存储器接口。处理器通过地址总线发送一个地址到存储器系统。对于读操作，MemWrite信号为0，存储器通过读数据总线readdata返回数据。对于写操作，MemWrite为1，处理器通过写数据总线writedata发送数据到存储器。

图8.1 内存接口

Processor: 处理器

Memory: 存储器

存储器系统设计的主要问题可以用图书馆里的书来比喻说明。图书馆的很多书都放在书架上。如果你正在写一篇以梦为主题的学期报告，你可能会去图书馆，取出弗洛伊德的《梦的解释》，然后带到工作室。在浏览之后，你会把它带回图书馆，然后取出荣格的《无意识心理学》。之后你可能因为一篇参考文献又回到图书馆查阅《梦的解释》，随之又来回查阅弗洛伊德的《自我与本我》。一个更聪明的办法是，把所有的书都保存在你的工作室以节省时间，而不是带着这些书来来回回。更进一步，当你取出一本弗洛伊德的书时，还应该在同一个书架取出他编著的其它书。

这个例子说明了6.2.1小节中所介绍的应使常见事件速度更快的原则。将那些最近使用或者最近最可能使用的书保存在工作室，可以减少了来回奔波的时间耗费。这里应用了时间局部性（temporal locality）和空间局部性（spatial locality）的原理。时间局部性意味着如果你最近使用的一本书，很可能很快就再用上它。空间局部性意味着当你使用一本书时，很可能会对同一主题的另外一些书也感兴趣。

图书馆自身也使用局部性原理使得常见事件速度更快。图书馆没有这么多的书架空间和预算来提供世界上的所有书。但是，它把一些不常用的书保存在地下室中。而且，它会与周边的图书馆建立馆际借阅约定，这样它就可以提供比其物理存储量更多的书籍。

总的来说，通过存储层次化可以对最常用的书做到大容量和快速访问。最常用书籍在你的工作室中；更多的书籍放在书架上；而其它更大一部分的可用书籍存储在地下室和其他图书馆。相似地，存储器系统使用存储器层次结构以快速访问最常用的数据，同时也有容量来存储大量的数据。

基于这种层次结构的存储子系统已经在5.5小节中介绍了。计算机内存基本上由动态RAM（DRAM）和静态RAM（SRAM）组成。理想的计算机存储器系统应是快速、大容量和廉价的。但实际上，某种特定内存只能具有这三个属性中的两个：必然会具有速度慢、容量小或者昂贵等三个缺点之一。但是计算机系统可以将一个快速、小容量和廉价的存储器和一个低速、大容量和廉价的存储器组合起来以接近理想的存储器系统。快速的存储器存储最常用的数据和指令，所以平均来看，存储器系统看起来可以快速运行。大容量的存储器存储其余的数据和指令，所以总的容量很大。两个廉价存储器组合在一起比单独使用一个大容量快速存储器要便宜得多。这个原则可以扩展至整个存储器层次结构以降低成本，提高速度。

计算机主存一般由DRAM芯片组成。在2012年，一个典型PC机的DRAM内存容量为4到8 GB，其成本约为10美元/GB。在过去的30年中，DRAM的价格以每年大约25%的速度下降，存储容量以相同的速度增加。所以PC机中存储器的总成本保持大致稳定。不幸的是，DRAM的速度只以每年7%的速度增长，然而处理器的性能则以每年25%到50%的速度增长，如图8.2所示。图中以1980年的存储器和处理器速度作为基线。在1980年，处理器和存储器的速度是一样的。但是性能却从此开始有了差别，存储器速度开始严重落后。

图8.2 逐渐分离的处理和存储器性能

Performance: 性能

Memory: 存储器

（得到Hennessy和Patterson所著《计算机体系结构：量化分析方法》第5版许可）

在20世纪70年代和80年代，DRAM的速度和处理器保持一致，但现在却慢得可怜。DRAM访问时间是处理器周期的一到两个数量级（前者需要几十纳秒，后者则不到1个纳秒）。

为了抵消这个趋势，计算机将最常用的指令和数据存储在更快、但较小的存储器中，这称为高速缓存（cache）。高速缓存通常置与处理器同一芯片上的SRAM中，其速度与处理器相近。这是因为SRAM比DRAM要快，而且片上存储器可以消除片间传输产生的延迟。2012年，片上SRAM的成本大约为10000美元/1GB，但是高速缓存容量较小（千字节到数兆字节），所以总的花费并不很高。高速缓存可以存储指令和数据，但是统称它们的内容为“数据”。

如果处理器需要的数据在高速缓存中可用，它就会被快速地返回。这称为缓存命中（hit）。否则，处理器就需要从主存（DRAM）中获得数据。这称为缓存缺失（miss）。如果大部分情况下缓存命中，那么处理器就基本上不需要等待低速的主存，平均访问时间就会比较短。

存储器层次结构的第三层是硬盘（hard drive）。就象图书馆使用地下室储存没有放在书架上的书籍一样，计算机使用硬盘去存储不在主存的数据。在2012年，一个使用磁性存储器构建的硬盘驱动器（hard disk drive，HDD）价格低于0.1美元/GB，访问时间约为10ms。其价格以每年60%的速度下降，但是访问时间几乎没有提高。使用闪存技术构建的固态硬盘（solid state drive，SSD），日益成为HDD的常见替代。SSD已经在小众市场里使用了超过20年，在2007年首次进入主流市场。SSD克服了HDD的一些机械故障，但价格是HDD的十倍，1美元/GB。

硬盘提供了一个比主存实际容量中更大的存储器空间，称为虚拟存储器（virtual memory）。正如地下室的书，需要花费很长的时间访问虚拟存储器中的数据。主存，也称为物理存储器（physical memory），包含了虚拟存储器的一个子集。因此，主存可以看作硬盘中常用数据的高速缓存。

图8.3总结了后续章节讨论的计算机系统存储器层次结构。处理器首先在容量小，但是速度快的高速缓存内寻找数据。如果数据不在高速缓冲中，处理器会在主存中寻找。如果数据也不在主存，那么处理器就会从容量大，但是速度低的硬盘上的虚拟存储器中获取数据。图8.4说明了存储层次结构中容量和速度的权衡，列举了2012年技术水平下典型的成本和访问时间数据。其中访问时间越短，速度越快。

图8.3 典型的存储体系结构

Processor chip: 处理器芯片

Cache: 高速缓存

Main memory: 主存

Hard disk: 硬盘

图 8.4 2012年时存储层次结构中各组成部分的典型特征

Cache：高速缓存

Main memory: 主存

Virtual memory: 虚拟存储器

Speed：速度

Capacity：容量

Technology：技术

Price/GB: 每GB价格

Assess Time：访问时间

Bandwidth：带宽

8.2节分析了存储器系统的性能。8.3节讨论了几种高速缓存的组织方法。8.4小节研究了虚拟存储器系统。总之，本章主要探究处理器如何采用与访问内存相似的方式来访问输入输出设备（如键盘和监视器）。8.5小节还将讨论存储器映射I/O。8.6小节解决嵌入式系统的I/O问题，8.7小节描述个人电脑的主要I/O标准。

#### 8.2 存储器系统性能分析

设计者（和计算机购买者）需要定量的方法衡量存储器系统的性能，以评估不同选择下成本和收益的平衡点。存储器系统性能的衡量标准为：缺失率（miss rate）或者命中率（hit rate），平均存储器访问时间（average memory access time）。缺失率和命中率计算如下：







（8.1）

例8.1 计算高速缓存性能

假设一个程序有2000条数据访问指令（读和写），其中1250条指令所需要的数据在高速缓存中能找到，其余的750个数据由主存或者硬盘提供。高速缓存的缺失率和命中率是多少？

解： 缺失率为750/2000=0.375=37.5%，命中率为1250/2000=0.625=1-0.375=62.5%

平均存储器访问时间（average memory access time，ATAM）为处理器的一条存储器访问指令执行过程中等待存储器的平均时间。图8.3的典型计算机系统中，处理器首先在高速缓存中查找数据。如果高速缓存找不到，处理器随之在主存中查找。如果主存中也缺失，处理器将访问硬盘上的虚拟存储器。因此，ATAM计算如下：

（8.2）

其中，tcache，tMM和tVM分别为高速缓存、主存和虚拟存储器的访问时间。MRcache和MRMM分别为高速缓存和主存的缺失率。

例8.2 计算平均存储器访问时间

假设某计算机系统拥有存储器层次结构由高速缓存和主存两级构成。根据表8.1给出的缺失率计算平均存储访问时间是多少？

解：平均存储访问时间为 1+0.1（100）=11周期

表8.1 访问时间和缺失率

Memory level: 存储器层次

Cache: 高速缓存

Main memory: 主存

例8.3 改进访问时间

11个周期的平均存储器访问时间意味着处理器对每一个实际需要使用的数据需要等待10个周期。为了将平均存储访问时间减低至1.5个周期，高速缓存缺失率应为多少？使用表8.1的数据。

解：如果缺失率为m，平均访问时间为1+100m。设置这个时间为1.5，解出需要的高速缓存缺失率m为0.5%。

值得注意的是，性能改进并不是像看起来那么好。例如，存储器系统速度提高10倍并不一定意味着计算机程序速度快10倍。如果程序性能的50%归因于存储器访问指令，提高10倍速度的存储器系统只意味着能提高1.82倍的程序性能。这个通用原则称为Amdahl定律（Amdahl’s law），即只在子系统的性能影响占全部性能中大部分的时候，提高子系统性能的努力才会有效果。

#### 8.3 高速缓存

高速缓存中存放着常用的存储器数据，其存放数据字的量为容量（Capacity）C。因为高速缓存的容量比主存要小，所以计算机系统设计者必须选择主存的子集存放在高速缓存中。

当处理器尝试访问数据，它首先检查高速缓存的数据。如果高速缓存命中，那么数据马上就可被使用。如果高速缓存缺失，处理器就会从主访问出数据，然后把它放在高速缓存以便以后使用。为了放置新的数据，高速缓存必须替换（replace）旧数据。本节将研究高速缓存设计中的以下问题：（1）在高速缓存中存放哪些数据？（2）如何在高速缓冲中寻找数据？ （3）当高速缓存满时，如何替换旧数据以放置新数据？

阅读后续章节时，要记住解决这些问题的驱动力在于大部分应用中数据访问存在的固有的时间和空间局部性。高速缓存使用时间和空间局部性预测下一步需要的数据是什么。如果程序以随机方式访问数据，它将不会在高速缓存的策略中得益。

我们将在后续章节中，以容量C，组数S，块容量b，块数B和相联度N来刻画高速缓存。

尽管我们这里主要关注数据高速缓存的读出操作，但是对指令高速缓存也适用于同样原则。数据高速缓存的写入操作也与之相似，并将在8.3.4小节讨论。

##### 8.3.1高速缓存中存放的数据

理想化的高速缓存应能提早预知所有处理器需要的数据，并提前从主存访问得到，所以理想的高速缓存缺失率为零。因为不可能精确地预计将来所需的数据，所以高速缓存必须基于过去存储器访问的方式来猜测将来需要什么数据。特别地，高速缓存利用时间和空间的局部性以达到低缺失率。

时间局部性意味着：如果处理器最近访问过一块数据，那么它很可能再次访问这一份数据。因此，当处理器读出和写入不在高速缓存中的数据时，数据需要从主存拷贝到高速缓存中。后续对此数据的请求将在高速缓存内命中。

空间局部性意味着：当处理器访问一块数据时，那么它很可能也访问此存储位置附近的数据。因此，当高速缓读出内存中一个字的时候，它也可以取出临近的一些字。这样的一组字称为高速缓存块（cache block）。一个高速缓存块中字的数量称为块大小b。容量为C的高速缓存包含了B=C/b个块。

实际程序的时间和空间局部性原理已经为实验所验证。如果在程序中使用了一个变量，那么同一变量就很可能被再次使用，从而产生了时间局部性。如果使用了一个数组的元素，那么同数组的其它元素也很可能被使用，从而产生了空间局部性。

##### 8.3.2 高速缓存中的数据查找

一个高速缓存可以组织成S个组，其中每一组有一个或者多个数据块。主存中数据的地址和高速缓存中数据的位置之间的关系称为映射（mapping）。每一个内存地址可以准确地映射到高速缓存中的一个组。地址中的一些位用于确定哪个高速缓存组包含了数据。如果一个组包含了多个块，那么数据可能包含在组中的任何一块内。

高速缓存按照组中块的数目进行分类。在直接映射（direct mapped）高速缓存内，每一组包含一个块，所以高速缓存包含了S=B个组。因此，一个主存地址映射到高速缓存唯一的块。在N路组相联（N-way associative）高速缓存中，每一组包含N个块。地址依然映射到唯一的组，其中共有S=B/N个组。但是这个地址对应的数据可以映射到组中的任何块。全相联（full associative）高速缓存只有唯一一个组（S=1）。数据可以映射到组内B个块中的任何一块。因此，全相联高速缓存也是B路组相联高速缓存的别名。

为了说明高速缓存的组织方式，我们将考虑32位地址和32位字的MIPS存储器系统。内存按照字节编址，每个字有四个字节，所以内存包含了230个字，并按照字方式对齐。为了简化起见，我们将首先分析容量C为8个字，块大小b为1个字的高速缓存，之后推广到更大的块。

**直接映射高速缓存**

直接映射高速缓存的每个组内有一个块，所以其组数S等于块数B。要理解高速缓存块上内存地址的映射，可以想象内存就像高速缓存那样映射到多个b个字大小的块。内存中第0块的地址映射到高速缓存的第0组。内存中第1块的地址映射到高速缓存的第1组，这样一直到内存中第B-1块的地址映射到高速缓存的第B-1组。此时高速缓存没有更多的块了，所以就开始循环，内存的第B块映射到高速缓存的第0组。

图8.5中用容量为8个字，块大小为1个字的直接映射高速缓存说明了这个映射。高速缓存中有8组，每一组有1块。因为地址是字对齐的，所以最低两位地址恒为00。紧接着的log28=3位说明了存储器地址映射到哪一个组。因此，地址0x00000004, 0x00000024, …, 0xFFFFFFE4的数据全部映射到第1组，以蓝色标注。相似地，地址0x00000010, …, 0xFFFFFFF0的数据全部映射到第4组，如此类推。每一个主存地址都可以映射到唯一的一个高速缓存组上。

图8.5 将主存映射到直接映射高速缓存

Address：地址

Data: 数据

230-word main memory: 230个字的主存

23-word cache: 23个字的高速缓存

Set: 组

例8.4 高速缓存字段

对于图8.5，地址0x00000014的字映射到哪一个高速缓存组？给出另一个映射到相同组上的地址。

解：因为地址是字对齐的，所以地址的最低两位为00。相邻三位为010，所以这个字映射到第5组。地址0x34，0x54，0x74，…,0xFFFFFFF4上的字都映射到这一组。

因为很多地址都映射到同一组上，高速缓存还必须保持实际包含在每一组内数据的地址。地址的最低有效位说明了哪一组包含数据。剩下的高位称为标志（tag），说明了包含在组内的数据是众多可能地址中的哪一个。

在我们先前的例子中，32位地址中的最低两位称为字节偏移（byte offset），说明了字节在字中的位置。紧接着的3位称为组位（set bits），说明了地址映射到哪一组（一般来说，组位的位数为log2S）。剩下的27位标志位说明了存储在特定高速缓存组中数据的存储器地址。图8.6给出了地址0xFFFFFFF4的高速缓存字段。它映射到第一组，且所有标志都为1。

图8.6 当地址0xFFFFFFF4映射到图8.5高速缓存时相应的高速缓存字段

Memory address: 存储器地址

Tag: 标志

Set: 组号

Byte offset: 字节偏移

例 8.5 高速缓存字段

为具有1024（210）个组和块大小为1个字的直接映射高速缓存确定组位和标志位的数目。其中地址长度为32位。

解：一个有210组的高速缓存的组位长度为log2210=10位。地址中的最两位为字节偏移，剩下的32-10-2=20位作为标志。

有时（例如计算机刚启动时），高速缓存组没有包含任何数据。高速缓存的每一组都有一个有效位（valid bit），以说明此组是否包含有意义的数据。如果有效位为0，那么其内容就没有意义。

图8.7为图8.5中直接映射高速缓存的硬件结构。高速缓存由8个表项的SRAM组成。每一个表项（组）包含了一个32位的数据缓存行、27位的标志和1位有效位。高速缓存使用32位地址访问。最低两位的字节偏移位因为字对齐而省略，紧接着的3位（组位）指明了高速缓存中的表项号或组号。读指令从高速缓存中读出特定的表项，检查标志和有效位。如果标志与地址中的最高27位相同，而且有效位为1，则高速缓存命中，数据将返回到处理器。否则，高速缓存发生缺失，存储器系统必须从主存中取出数据。

图8.7 8组的直接映射高速缓存

Memory address: 存储器地址

Tag: 标志

Set: 组号

Byte offset: 字节偏移

Data: 数据

Hit：命中

8-entry×（1＋27＋32）bit SRAM：8个表项的SRAM，其中每个表项包括（1＋27＋32）位。

例 8.6直接映射高速缓存的时间局部性

实际应用中，循环是时间和空间局部性的常见来源。使用图8.7中8表项的高速缓存，给出在执行以下MIPS汇编代码循环后高速缓存中的内容。假设高速缓存开始状态为空。缺失率为多少？

addi $t0, $0, 5

loop: beq $t0, $0, done

lw $t1, 0x4($0)

lw $t2, 0xc($0)

lw $t3 ,0x8($0)

addi $t0 , $t0 ,-1

j loop

done:

解：这个程序包含一个重复5遍的循环。每一次循环涉及3次的内存访问（读取），最后总计产生15次内存访问。第一次循环执行的时候，高速缓存为空。必须分别从主存的0x4,0xC,0x8中获取数据，存放到高速缓冲的第1组，第3组和第2组。然后，以后四次的循环执行，数据在高速缓存中都能被找到。图8.8所示为最后对内存地址0x4请求时的高速缓存内容。因为高27位的地址为0，所以标志全为0。缺失率为3/15=20%.

图8.8 直接映射高速缓存的内容

Memory address: 存储器地址

Tag: 标志

Set: 组号

Byte offset: 字节偏移

Data: 数据

当两个最近访问的地址映射到同一个高速缓存块时，会产生冲突（conflict）。最近访问的地址从块中逐出较前的地址。直接映射高速缓存每一组只有1个块，所以两个映射到同一个组的地址常常会产生冲突。下面的例子来说明冲突。

例8.7 高速缓存块冲突

当在图8.7中的8字直接映射高速缓存中执行以下循环时，缺失率是多少？假设高速缓存初始为空。

addi $t0, $0, 5

loop: beq $t0 , $0, done

lw $t1 , 0x4($0)

lw $t2 , 0x24($0)

addi $t0 , $t0 , -1

j loop

done:

解：内存地址 0x4和0x24都映射到第一组。当初始执行循环的时候，地址0x4中的数据被读入到高速缓存的第一组。接着，地址0x24中的数据被读入到第一组，并逐出地址0x4中的数据。在下一循环执行时，这种访问模式会重复发生。高速缓存必须重新获取地址0x4中的数据，逐出地址0x24的数据。这两个地址产生冲突，缺失率为100%

**多路组相联高速缓存**

N路组相联的高速缓存通过为每组提供N块的方式以减少冲突。每个内存地址依然映射到唯一的组中，但是它可以映射到一个组内N个块的任意一块。因此，直接映射高速缓存也可称为单路组相联高速缓存。其中，N称为高速缓存的相联度（degree of associative）。

图8.9给出了容量C为8个字，相联度N为2的两路组相联高速缓存的硬件。高速缓存现在只有4组，而不是直接映射高速缓存的8组。因此，只需要log24=2位组位来选择组，而不是直接映射高速缓存的的3位。标志从27增加到28位。每一组包括了两路（相联度为2）。每一路由数据块、有效位和标志位组成。高速缓存从选定的组中读取所有两路中的块，检查标志和有效位以确定是否命中。如果其中一路命中，多路选择器从此路选择数据。

图8.9 两路组相联高速缓存

Memory address: 存储器地址

Tag: 标志

Set: 组号

Byte offset: 字节偏移

Data: 数据

Hit：命中

Way 0：第0路

Way 1: 第1路

与同容量的直接映射高速缓存相比，组相联高速缓存的缺失率一般会较低。这是因为它们的冲突更少。然而，因为输出的多路选择器和额外的比较器，组相联高速缓存常常比较慢，成本也比较高。并且还会产生另外一个问题：当两路都满时，选择哪一路进行替换？这个问题将在8.3.3小节中讨论。大部分的商业系统都使用组相联高速缓存。

例8.8 组相联高速缓存的缺失率

重复例8.7的问题，使用图8.9中8字2路组相联高速缓存。

解：两个对0x4和0x24地址的存储器访问都映射到第一组。然而，高速缓存中有两路，所以它能同时为两个地址提供数据空间。在第一个循环期中，空的高速缓存对两个地址访问都产生缺失，然后读入两个字的数据存放到第1组的两路中，如图8.10所示。随后的4次循环中，高速缓存都命中。因此，缺失率为2/10=20%。对比例8.7中同容量大小的直接映射高速缓存缺失率为100%

图8.10 两路组相联高速缓存内容

Way 1: 第1路

Way 0: 第0路

Tag：标志

Data：数据

Set 3：第3组

Set 2：第2组

Set 1：第1组

Set 0：第0组

**全相联高速缓存**

全相联高速缓存只有一个组，其中包含了B路（B为块的数目）。存储器地址可以映射到这些路中的任何一个块。全相联高速缓存也可以称为B路单组组相联高速缓存。

图8.11为包含了8个块的全相联高速缓存SRAM阵列。对于一个数据请求，由于数据可能在任何一块中，所以必须对8个标志进行比较（图中没有表示出）。相似地，当一路命中时，需要使用8：1多路选择器选择合适的数据。在相同容量下，全相联高速缓存一般具有最小的冲突缺失，但是需要更多的硬件用于标志比较。因为需要大量的比较器，它们仅仅适合于较小的高速缓存。

图8.11 8块全相联高速缓存

Way 1: 第1路

Way 0: 第0路

…

Tag：标志

Data：数据

**块大小**

前面的例子中块大小仅为一字，因此只能利用时间局部性。为了利用空间局部性，高速缓存需要使用更大的块来保存若干连续的字。

块大小大于1个字的优势在于，在发生缺失时所需要字会取到高速缓存内，同时在块内临近的字也会从存储器取到高速缓冲中。因为空间局部性的关系，后续的访问就很可能会命中。然而，对于容量固定的高速缓存而言，较大的块大小意味着块的数目较少。这可能会导致更多的冲突，从而增加缺失率。而且，因为要将多于一个字的数据从主存中取出，所以在一次缺失时需要耗费更多时间来读取缺失的高速缓存块。读取缺失块到高速缓存所需的时间称为缺失代价（miss penalty）。如果块中相邻的字在稍后未被访问，那么用于取出它们的工作就会被浪费。然而，大部分实际程序在取较大的块时可以获得更好性能。

图8.12中给出了容量C为8个字，块大小b为4字的直接映射高速缓存硬件。此时，高速缓存只有B=C/b=2块。直接映射高速缓存的每一个组里仅有一块，所以这个高速缓存中两个组，只需要log22=1用于选择组。同时，需要一个多路选择器来选择在一个块里的字。多路选择器由地址中的块内偏移（block offset）位（log24＝2位）控制。最高的27位地址组成标志。整个块只需要一个标志，这是因为块内字的地址是连续的。

图8.12 组数为2，块大小为4字的直接映射高速缓存

Tag：标志

Memory address:存储器地址

Set：组

Block offset:块内偏移

Byte offset: 字节偏移

Data：数据

Hit：命中

图8.13给出了地址0x8000009C映射于此直接映射高速缓存时的cache字段。对于字访问时，字节偏移总是0。后续log2b=2位的块内偏移位指明了此字在块中的位置。后续一位指出组号。剩下的27位为标志位。因此，地址为0x8000009C的字映射到此高速缓存中第1组的第3个字。

图8.13 映射到图8.12的高速缓存时，对于于地址0x8000009C的cache字段

Tag：标志

Memory address:存储器地址

Set：组

Block offset:块内偏移

Byte offset: 字节偏移

例 8.9 直接映射高速缓存中的空间局部性

其中采用容量为8个字、块大小为4个字的直接映射高速缓存重复例子8.6

解：图8.14给出了第一次存储器访问后高速缓存的内容。在第一次迭代循环时，高速缓存在访问存储器地址0x4时产生缺失。这次访问将读取从地址0x0到0xC的数据到高速缓存的块中。所有的后续访问（如地址0xC所示）都将在高速缓存内命中。因此，缺失率为1/15=6.67%。

图8.14 块大小为4字的高速缓存内容

Tag：标志

Memory address:存储器地址

Set：组

Block offset:块内偏移

Byte offset: 字节偏移

**小结**

高速缓存组织为二维阵列的结构。其中行称为组，列称为路。阵列中每个表项包括一个数据块、相应的有效位和标志位。高速缓存的关键参数为：

* 容量 C
* 块大小 b（以及块数量 B=C/b）
* 一组内块的数量（N）

表8.2总结不同类型的高速缓存组织方式。存储器中的每个地址映射到唯一一组，但是它可以存放在此组的任何一路中。

表8.2 高速缓存的组织方式

Organization：组织方式

Direct mapped: 直接映射

Set associative: 组相联

Full associative: 全相联

Number of ways: 组数

Number of ways: 路数

高速缓存的容量、相联度，组大小和块大小一般都为2的整数次幂。这使得高速缓存字段（标志，组号和块内偏移）均为地址位的子集。

增加相联度N可以减少因为冲突引起的缺失。但是高相联度需要更多的标志比较器。增加块的大小b，会从空间局部性获益而减少缺失率。然而，对于固定容量的高速缓存这将减少组数，可能导致更多的冲突。同时，它也会增加了缺失代价。

##### 8.3.3 数据的替换

在直接映射高速缓存中，每个地址映射到唯一的块和组上。如果一个组满的时候，而又必须读取新的数据，那么组里的块就会被替换为新数据。在组相联和全相联的高速缓存中，高速缓存必须在组满时选择哪一个块被替换。时间局部性原则建议最好选择最近最少使用的块，因为它看起来最近最不可能再次用到。因此，大部分相联高速缓存采用最近最少使用（least recently used, LRU）的替换原则。

在两路组相联高速缓存内，用1位使用位（use bit）U，说明组内的哪一路是最近最少使用的。每次使用其中一路，就修改U位来指示另一路为最近最少使用。对于多于两路的组相联高速缓存，跟踪最近最少使用的路将更为复杂。为了简化问题，组里的多路分成两部分（group），U指示那一部分为最近最少使用的。替换时，就从最近最少使用部分里面随机选择一块用于替换。这样的策略称为伪-LRU策略，也易于实现。

例 8.10 最近最少使用替换

写出下述执行代码后，容量为8个字的两路组相联高速缓存的内容。假设采用最近最少使用替换策略，块大小为1个字，初始时高速缓存为空。

lw $t0 ,0x04($0)

lw $t1 ,0x24 ($0)

lw $t2 ,0x54 ($0)

解： 开始的两条指令从存储器地址0x4和0x24中读取数据到高速缓存的第1组中，如图8.15（a）所示。U=0说明在第0路的数据是最近最少使用的。下一次存储器访问的地址为0x54，依然映射到组1，这将替换最近最少使用的第0路数据。如图8.15（b）所示。随后使用位U设置为1，说明第1路的数据是最近最少使用的。

图8.15 LRU替换的两路相联高速缓存

Tag：标志

Data：数据

Set: 组

Way 0： 第0路

Way 1： 第1路

##### 8.3.4 高级高速缓存设计\*

现代微处理器系统使用多层高速缓存以减少内存访问时间。本节将讨论两层高速缓存系统的性能，研究块大小、相联度和高速缓存容量对缺失率的影响。本节还介绍高速缓存如何使用直写或回写策略控制处理存储器写入。

**多层次高速缓存**

大容量高速缓存的效果更好，这是因为它们更有可能保存当前需要使用的数据，因此会有更低的缺失率。然而，大容量高速缓存的速度比小容量高速缓存要低。现代处理器系统常常使用两级高速缓存，如图8.16所示。第一级（L1）高速缓存足够小以保证访问时间为1到2个处理器周期。第二级（L2）高速缓存常常也由SRAM构成，但比L1高速缓存容量更大，因此速度也更慢。处理器首先在L1高速缓存中查找数据。如果在L1高速缓存中缺失，那么处理器将从L2高速缓存中查找。如果L2高速缓存也缺失，处理器将从主存访问数据。因为从主存的访问速度实在太慢，一些现代处理器系统在存储器层次结构中增加了更多级的高速缓存。

图8.16 带两级高速缓存的存储器体系结构

L1 Cache：一级高速缓存

L2 Cache：二级高速缓存

Main memory: 主存

Virtual memory: 虚拟存储器

Speed：速度

Capacity：容量

例 8.11 带L2高速缓存的系统

使用图8.16中的系统，其中L1、L2高速缓存和主存的访问时间分别为1、10和100个周期。假设L1、L2高速缓存的缺失率分别为5%和20%。即5%的访问在L1中缺失，其中的20%在L2中依然缺失。那么平均访问时间(AMAT)是多少？

解：每一次的内存访问都会检查L1高速缓存。当L1高速缓存缺失时（访问中的5%），处理器就检查L2高速缓存。当L2高速缓存缺失时（访问中的20%），处理器就从主存获取数据。使用等式8.2，可计算平均内存访问时间如下：1周期+0.05[10周期+0.2（100周期）]=2.5周期。

L2高速缓存的缺失率高是因为它只接收那些在L1高速缓存缺失的“硬”内存访问。如果所有的访问都直接由L2高速缓存中获得，那么L2的缺失率会是大约1%。

**减少缺失率**

可以通过改变容量、块大小和相联度的方式减少高速缓存的缺失率。减少缺失率的第一步是理解产生缺失的原因。缺失可以分为强制缺失、容量缺失和冲突缺失等三种原因。对于高速缓存块的第一次请求被称为强制缺失（compulsory miss），这是因为无论高速缓存任何设计，块都必须先从内存读取。因高速缓存太小以不能保存所有并发使用的数据时发生容量缺失（capacity miss）。因多个地址映射到同一组而被替换的块依然需要时发生冲突缺失（conflict miss）。

改变高速缓存的参数可以影响一种或更多的高速缓存缺失类型。例如，增加高速缓存容量可以减少冲突和容量缺失，但是不会影响强制性缺失。另一方面，增加块大小可以减少强制性缺失（因为空间局部性），但是可能增加冲突缺失（因为更多的地址可能会被映射到同一组中，这可能会冲突）。

存储器系统十分复杂，衡量它们性能的最佳方法是在不同的高速缓存配置参数下运行基准测试程序。图8.17描述了SPEC2000基准测试程序下高速缓存容量、相联度和缺失率的关系。这一基准测试程序中强制性缺失较少，以靠近x轴的黑色区域表示。正如所期望的，增加高速缓存容量可以减少容量缺失。特别对于小型高速缓存来说，增加相联性可以减少冲突缺失，如顶端的曲线所示。在四路或八路以上再增加相联性只能小幅度地减少缺失率。

图8.17 基准测试程序SPEC2000下高速缓存容量、相联度和缺失率的关系

（得到Hennessy和Patterson所著，Computer Architecture: A Quantitative Approach，第5版许可）

Miss Rate per Type: 每种类型的缺失率

Cache size: 高速缓存容量

Capacity：容量缺失

Compulsory: 强制缺失

1-way: 1路

2-way: 2路

4-way: 4路

8-way: 8路

正如前面提到的，可以用增加块大小的方法利用空间局部性，减少缺失率。但是在固定大小的高速缓存中，随着块大小的增加，组的数量将减少，从而增加冲突的可能性。图8.18描述了对不同容量下高速缓存块大小（以字节为单位）和缺失率之间的关系。对于小型高速缓存（如4KB高速缓存），在大于64字节时增加块大小会因为冲突而增加缺失率。对于大型高速缓存，增加块大小时缺失率并不改变。然而，较大的块大小可能还将增加执行时间，这是因为其缺失时从主存用于获取高速缓存块的缺失代价时间更多。

图8.18基准测试程序SPEC2000下块大小、高速缓存大小和缺失率的关系

（得到Hennessy和Patterson所著，Computer Architecture: A Quantitative Approach，第5版许可）

Miss Rate：缺失率

Block Size：块大小

**写入策略**

前面各节关注了存储器的读出操作。存储器的存储（写入）操作遵循与读出操作相似的过程。在存储器写入前，处理器会检查高速缓存。如果高速缓存缺失，就会将相应的高速缓存块从主存读取到高速缓存，之后将写入高速缓存块中适当的字。如果高速缓存命中，就简单地将字写入到高速缓存块中。

高速缓存可以分为写直达和写回两种方式。在写直达（write-through）高速缓存中，写入到高速缓存块的数据会同时写到主存中。在写回（write-back）高速缓存中，需要增加一个与各高速缓存块关联的脏位（D）。当写入高速缓存块时D设置为1，其余情况为0。只在脏高速缓存块从高速缓存内替换出来时，它们才写回到主存。写直达的高速缓存不需要脏位，但是与写回方式的高速缓存相比需要更多主存写入操作。由于主存访问时间太长，现代的高速缓存往往采用写回方式。

例 8.12 写直达与写回

假设某高速缓存的块大小为4个字。使用直写和写回两种策略，在执行以下代码时主存访问次数分别为多少？

sw $t0 , 0x0($0)

sw $t0 , 0xC($0)

sw $t0 , 0x8($0)

sw $t0 , 0x4($0)

解：所有四个存储指令对同一个高速缓存块写入。在写直达高速缓存中，每一个存储指令对主存写入一个字，需要4次的主存写入。写回策略仅仅在脏高速缓存块被替换时才需要一次主存访问。

##### 8.3.5 MIPS处理器中高速缓存的发展\*

表8.3给出了1985到2010年间MIPS处理器中高速缓存组织结构的发展情况。主要的趋势包括引入多级高速缓存结构，更大高速缓存容量，增加相联度。产生这些趋势的原因在于不断增长的CPU频率和主存速度差异，以及不断下降的晶体管成本。CPU和存储器之间不断增加的速度差异需要不断减少缺失率以克服主存瓶颈，不断下降的晶体管成本则为增加高速缓存容量提供了可能。

表8.3 MIPS处理器中高速缓存的发展

Year: 年份

MHz：主频（MHz）

Direct mapped: 直接映射

Two-way: 两路组相联映射

variable size: 可变大小

\*来自于D. Sweetman所著See MIPS Run，Morgan Kuafmann，1999

#### 8.4 虚拟存储器

大部分现代计算机系统使用硬盘（也被称为硬盘驱动器）作为存储体系的最底层（如图8.4）。与理想化的大容量、快速、廉价存储器相比，硬盘容量大，价格便宜，但是速度却非常慢。硬盘比高成本效益的主存（DRAM）提供了更大容量。然而，如果大部分的存储器访问需要使用硬盘，性能将严重下降。在PC机上一下运行太多程序时，就可能遇到这种情况。

图8.19所示为一个去了盖子的硬盘驱动器，它由磁性存储器构成，也称为硬盘。顾名思义，硬盘包含了一片或者更多的坚硬盘片（platter），每一个盘片的长三角臂末端都有一个读写头（read/write head）。首先移动读写头到盘片的正确位置，当盘片在它下面旋转时以磁方式读写。读写头需要毫秒级的时间完成盘片上的正确寻道，这对于人看来很快，但却比处理器慢百万倍。

图8.19 硬盘

在存储器层次结构中增加硬盘的目标在提供一个虚拟化的廉价大容量存储系统，而且在大部分存储器访问时，依然能提供较快速的存储器访问速度。例如，一个只提供128MB DRAM的计算机，可以用硬盘高效提供2GB的存储。较大的2GB存储空间被称为虚拟存储器（virtual memory）。较小的128MB主存称为物理存储器（physical memory）。本节中，我们将使用物理存储器这个术语来指主存。

程序会访问虚拟存储器内的任一数据，必须使用虚地址（virtual address）指明其在虚拟存储器内的位置。物理存储器内保存了虚拟存储器中大部分最近访问过的子集。从这点来看，物理存储器充当了虚拟存储器的高速缓存。因此，大部分访问将以DRAM的速度命中物理存储器，而程序却可以使用更大容量的虚拟存储器。

针对8.3节中讨论的高速缓存原理，虚拟存储器系统使用了不同的术语。表8.4总结了类似的术语。虚拟存储器被分为虚页（virtual page），大小一般为4KB。物理存储器也类似地划分为物理页，大小也是4KB。一个虚页可能在物理存储器（DRAM）中，也可能在硬盘上。例如，图8.20中给出了一个大于物理存储器的虚拟存储器。长方形表示页。一些虚页当前在物理存储器中，另一些在硬盘上。根据虚地址确定物理地址的过程称为地址转换（address translation）。如果处理器试图访问不在物理存储器里的虚地址，就会产生页面失效（page fault），由操作系统将页从硬盘读取到物理存储器中。

表8.4 高速缓存和虚拟存储器相似的术语

Cache:高速缓存

Virtual memory: 虚拟存储器

Block：块

Block size: 块大小

Block offset: 块偏移

Miss：缺失

Tag：标志

Page：页

Page size: 页大小

Page offset: 页偏移

Page fault: 页面失效

Virtual page number：虚页号

图8.20 虚页和物理页

Virtual addresses: 虚拟地址

Address translation: 地址转换

Physical addresses: 物理地址

Physical memory: 物理存储器

Hard disk: 硬盘

为了防止因冲突而产生的页面失效，任意虚页都可以映射到任意物理页。换一句话说，物理存储器就象针对虚拟存储器的全相联高速缓存。在常规的全相联高速缓存里，每一个高速缓存块都有一个比较器，来比较最高位地址位与标志，以确定请求是否特定块是否命中。在类似的虚拟存储器系统里，每一个物理页也应该需要一个比较器，来比较虚拟存储器地址的最高位和标志，以确定虚页是否映射到物理页上。

现实的虚拟存储器系统中有很多物理页，对每一个页提供一个比较器的成本很高。作为代替，虚拟存储器系统使用页表实现地址转换。每个虚页都在页表都对应一个表项，来说明它在物理存储器中的位置，或在硬盘中的位置。每次读出或者写入指令需要首先访问页表，然后访问物理存储器。页表访问将程序使用的虚地址转换为物理地址。物理地址随后用于实际的读或者写数据。

页表常常太大而只能存放在物理存储器中。因此，每一次读出或者写入需要两次物理存储器访问：第一次是访问页表，第二次访问数据。为了加速地址转换，地址转换后备缓冲（translation lookaside buffer，TLB）缓存了最常用的页表表项。

本节的后续部分详细介绍了地址转换、页表和TLB。

##### 8.4.1 地址转换

在包含虚拟存储器的系统中，程序使用虚地址访问大容量存储空间。计算机必须将虚地址转换以找到物理存储器中的地址，或产生一个页面缺失然后从硬盘获得数据。

前面提到虚拟存储器和物理存储器都被分成页。虚地址或者物理地址的高位分别说明了虚页或物理页的页号（page number）。低位说明指明了页内字的位置，也称为页偏移。

图8.21说明包含2GB虚拟存储器，128MB物理存储器，页大小为4KB的虚拟存储器页组织结构。MIPS处理器采用32位地址。对于2GB=231字节虚拟存储器，仅使用虚拟存储器地址的最低31位，第32位总为0。相似地，对于128MB=227字节的物理存储器，仅只用物理地址的最低27位，最高5位总为0。

图8.21 物理页与虚页

Physical page number：物理页号

Physical address：物理地址

Physical memory：物理存储器

virtual page number：虚页号

virtual address：虚地址

virtual memory：虚拟存储器

因为页大小为4KB=212字节，所以有231/212=219个虚页和有227/212=215个物理页。因此，虚页和物理页的页号分别为19位和15位。物理存储器在任何时间只能最多保存1/16的虚页。其余的虚页保存在硬盘上。

图8.21表明虚页5映射到物理页1，虚页0x7FFFC映射到物理页0x7FFE等等。例如，虚地址0x53F8（虚页5内0x3F8的偏移）映射到物理地址0x13F8（物理页1内0x3F8的偏移）。虚地址和物理地址的最低12位是一样的（0x3F8），它指明了虚页和物理页内的页偏移。从虚地址到物理地址的转换过程中，只需要转换页号。

图8.22说明了虚地址到物理地址间的转换。最低12位为页偏移，不需要转换。虚地址的最高19位为虚页页号（virtual page number, VPN），可转换为15位物理页页号（physical page number, PPN）。后面两小节将进一步介绍页表以及如何使用TLB实现地址转换。

图8.22 虚地址到物理地址的转换

Virtual address: 虚地址

Page offset: 页偏移

Physical address: 物理地址

例8.13 虚地址到物理地址的转换

用图8.21的虚拟存储器系统确定虚地址0x247C的物理地址。

解：12位页偏移（0x47C）不需要转换。虚拟存储器地址的其余19位给出了虚页页号，所以虚地址0x247C应在虚页0x2中。图8.21中，虚页0x2映射到物理页0x7FFF。因此，虚地址0x247C映射到物理地址0x7FFF47C。

##### 8.4.2 页表

处理器使用页表（page table）完成虚地址和物理地址间的转换。前面提到页表对每一个虚页都包含了一个表项。表项中包括了物理地址页号和有效位。如果有效位是1，虚页映射到表项指定的物理页。否则，虚页在硬盘中。

因为页表比较大，需要存储在物理存储器中。假设页表存储为连续的数组，如图8.23所示。页表包含图8.21中的存储器系统映射。页表以虚页页号（VPN）作为下标。例如，第5个表项说明了虚页5映射到物理页1。第6个表项无效（V=0），所以虚页6在硬盘中。

图8.23 图8.21的页表

Physical page number:物理页号

Page table: 页表

Virtual page number: 虚拟页号

例 8.14 使用页表实现地址转换

使用图8.23给出的页表找出虚地址0x247C的物理地址。

解：图8.24给出了虚地址0x247C到物理地址的转换。其中12位页偏移不需要转换，虚地址的其余19位为虚页页号0x2，为页表的索引。页表虚页0x2到物理页0x7FFF。所以，虚地址0x247C映射到物理地址0x7FFF47C。其中物理地址和虚地址的最低12位保持不变。

图8.24 使用页表进行地址转换

Virtual address: 虚地址

Virtual page number: 虚页号

Page offset: 页偏移

Physical page number: 物理页号

Page table: 页表

Physical address: 物理地址

Hit: (应该为V)

页表可以存放在物理存储器的任何位置，这由操作系统自由决定。处理器一般使用专用的页表寄存器（page table register）存放页表在物理存储器中的基址。

为了实现读出和写入操作，处理器必须首先进行虚地址到物理地址的转换，之后访问物理地址中的数据。处理器从虚地址提取虚页页号，之后将其和页表寄存器相加，找到页表表项的物理地址。处理器随后从物理存储器读取页表表项，以获取物理页页号。如果表项有效，处理器将物理页页号和页偏移合并，生成物理地址。最后，它根据物理地址读出或者写入数据。因为页表存储在物理存储器中，所以每一次读出或者写入操作需要两次物理存储器访问。

##### 8.4.3 地址转换后备缓冲

如果每一次的读出和写入都需要读取一次页表，将加倍了读出和写入的延迟，对虚拟存储器的性能会产生严重的影响。幸运的是，页表访问有很好的时间局部性。数据访问的时间和空间局部性，以及比较大的页意味着很多连续的读出和写入操作都发生在同一页上。因此，如果处理器能记住它最后读出的页表表项，就很可能可以重用这个转换表项而不需要重读页表。一般来说，处理器可以将最近使用的一些页表表项保存在称为地址转换后备缓冲（translation lookaside buffer, TLB）的小型高速缓存内。处理器在访问物理存储器页表前首先查找TLB内的转换表项。在实际的程序中，绝大多数访问都在TLB中命中，避免了读取物理存储器中页表的时间消耗。

TLB以全相联高速缓存的方式组成，一般有16到512个表项。每一个TLB表项有一个虚页号和它相应的物理页号。TLB使用虚页页号访问。如果TLB命中，它返回相应的物理页号；否则，处理器必须从物理存储器里读页表。TLB设计得足够小使得它的访问时间可以小于一个周期。即使如此，TLB的命中率一般也大于99%。对于大多数读出和存取指令，TLB使内存访问数从2次减少为1次。

例8.15 使用TLB实现地址转换

考虑图8.21的虚拟存储器系统。使用一个2表项TLB完成地址转换，或解释为什么对于虚地址0x247C和0x5FB0到物理地址的转换必须访问页表。假设TLB目前包换有效的虚页0x2和0x7FFFD的转换内容。

解：图8.25为处理虚地址0x247C请求的2个表项的TLB。TLB接收传入虚拟地址的虚页号0x2，将其与每一个表项的虚页页号比较。表项0匹配而且有效，所以请求命中。将匹配表项的物理页页号0x7FFF与虚地址的页偏移拼接形成转换后的物理地址。如常一样，页偏移不需要转换。

对虚地址0x5FB0的请求在TLB中缺失。所以请求需要转向到页表进行转换。

图8.25 使用2表项TLB的地址转换

Virtual address: 虚地址

Virtual page number: 虚页号

Page offset: 页偏移

Entry 0: 表项0

Entry 1: 表项1

Physical page number: 物理页号

Physical address: 物理地址

##### 8.4.4 存储器保护

到目前为止，本节都集中于如何使用虚拟存储器来提供一个快速，廉价和大容量的存储空间。使用虚拟存储器的一个同样重要原因是提供并发程序间的保护。

读者可能已经知道，现代计算机一般在同一时间运行好几个程序或者进程。所有程序在物理存储器内是并存的。在一个设计良好的计算机系统中，程序应当各自独立地保护起来，以避免被某个程序使得其他程序崩溃，或破坏其他程序。更准确地说，在没有得到允许的情况下，某个程序不能访问其他程序的存储空间。这称为存储器保护（memory protection）。

虚拟存储器系统为每个程序提供独立的虚拟存储器地址空间（virtual address space）以提供存储器保护。每一个程序可以任意使用自己虚拟存储器地址空间中的存储器，但在任一时刻只有部分的虚拟存储器地址空间在物理存储器中。每一个程序可以使用它所有的虚地址空间而无需担心其他程序的物理位置。然而，一个程序只能访问已经映射到自身页表里的物理页。这样，程序就不能意外地或者恶意地访问其他程序的物理页，因为它们没有映射到程序的页表中。在某些情况下，多个程序可以访问公共的指令或者数据。操作系统会为每一个页表项加上控制位，以决定哪些程序可以写入共享的物理页。

##### 8.4.5 替换策略\*

虚拟存储器系统使用写回和近似的最近最少使用（LRU）替换策略。每一个对物理存储器的写都产生写硬盘的写直达策略是不实际的。如果采用写直达策略，写入指令将以毫秒级的硬盘速度操作，而不是纳秒级的处理器速度。在写回策略下，当物理页从物理存储器替换出来时，才写回到硬盘。把物理页写回到硬盘，然后从硬盘中读出另外的虚页称为交换（swap）。所以虚拟存储器系统里的硬盘有时称为交换空间（swap space）。处理器在发生页面缺失时换出一个最近最少使用物理页，然后把缺失的虚页替换到被换出的页。为了支持这一替换策略，每一个页表表项需要包含两个额外的状态位：脏位D和使用位U。

在物理页从硬盘读出后，若任何写入指令修改过它的话，脏位就设置为1。当物理页被换出时，只在它的脏位为1时，才需要写回到硬盘。否则，硬盘已经有了这一页的正确副本。

如果物理页最近被访问过，那么使用位U为1。和高速缓存系统一样，精确的LRU替换将会异常复杂。所以操作系统使用近似的LRU替换策略：周期性地重设所有页表中的使用位为0。当一个页被访问，它的使用位设为1。在页面缺失时，操作系统寻找U=0的页换出物理存储器。因此，操作系统未必替换出最近最少使用的页，而只是其中一个最近最少使用页。

##### 8.4.6 多级页表\*

页表会占据大量的物理存储器。例如，前面提到的页面大小为4KB的2GB虚拟存储器将需要219个表项。如果每一表项占用4个字节，页表需要占用219×22字节＝221字节=2MB。

为了节省物理存储器，页表可以被分为多级（一般是两级）。第一级页表常保持在物理存储器中。它指明了小的第二级页表在虚拟存储器中的存放位置。第二级页表包含了一段范围虚页的实际转换内容。如果特定范围的转换内容没有使用到，相应的第二级页表可以被替换到硬盘，而不需要浪费物理存储器。

在两级页表中，虚页页号分为两部分：页表号（page table number）和页表偏移（page table offset），如图8.26所示。页表号对驻留在物理存储器里的第一级页表进行寻址。第一级页表表项给出了第二级页表的基址或者在V为0时表示必须从硬盘获取。页表偏移对第二级页表进行寻址。虚拟存储器地址的余下12位为页偏移，页大小为212=4KB。

图8.26 多级页表

Virtual address: 虚地址

Page table number: 页表号

Page table offset: 页表偏移

Page offset: 页偏移

Page table address: 页表地址

First-level page table: 第一级页表

Second-level page table: 第二级页表

Physical page number：物理页号

512 entries: 512个表项

1K entries: 1K个表项

图8.26中，19位虚页号被分为9位的页表号和10位的页表偏移。因此，第一级页表有29=512个表项。这512个第二级页表均有210=1K个表项。如果每个一级和第二级页表表项占用32位（4个字节），而且只有两个第二级页表同时在物理存储器中，这个层次化页表结构只使用了（512×4字节）+2×（1K×4字节）=10KB的物理存储器空间。两级页表只需要存储全部页表的2MB物理存储器的一小部分。两级页表的缺点是TLB缺失时转换过程将增加一次额外的存储器访问。

例 8.16 使用多级页表完成地址转换

图8.27为图8.26所示的两级页表可能包含的内容。仅给出了一个第二级页表的内容。使用这个两级页表，描述访问虚地址0x003FEFB0时发生了什么情况。

解：如常一样，只有虚页号需要转换。虚地址中最高9位为页表号0x0，这是第一级页表的索引。第一级页表的0x0号表项说明第二级页表在内存中（V=1），其物理地址为0x2375000。

虚地址的后10位（0x3FE）为页表偏移，给出了第二级页表的索引。第二级页表的表项0位于底部，表项0x3FF位于顶部。第二级页表的第0x3FE号表项说明虚页在物理存储器（V=1）中，且物理页号为0x23F1。将物理页号和页偏移拼接起来，形成物理地址0x23F1FB0。

图8.27 两层页表的地址转换

Virtual address: 虚地址

Page table number: 页表号

Page table offset: 页表偏移

Page offset: 页偏移

Page table address: 页表地址

First-level page table: 第一级页表

Second-level page table: 第二级页表

Physical page number：物理页号

#### 8.5 I/O简介

　　I/O系统用于连接计算机与外部设备（peripherals）。在个人计算机里，这些设备一般包括键盘、显示器、打印机和无线网络。在嵌入式系统里，这些设备可能包括吐司机的加热元件、玩偶的声音同步器、引擎的燃料注入器、卫星的太阳能面板定位马达，等等。处理器就像访问内存一样使用地址和数据总线访问I/O设备。

一部分的地址空间用于I/O设备而非内存。例如，设定0xFFFF0000到0xFFFFFFFF范围内的地址用于I/O。回忆6.6.1小节，这些地址为存储器映射的保留区域。每一个I/O设备可以指定到这一范围中的一个或者多个地址上。对于特定地址的写入操作就会给设备发送数据，读出操作就将从设备接收数据，这种I/O设备通讯方式称为内存映射I/O（memory-mapped I/O）。

在内存映射I/O系统里，读出和写入可能访问内存，也可能访问I/O设备。图8.28给出了支持两个内存映射I/O设备所需要的硬件。其中，地址译码器决定处理器与哪个设备进行通讯，根据Address和MemWrite信号来产生对其它硬件的控制信号。ReadData多路选择器在内存和各种I/O设备间选择读出数据。写使能寄存器保存了写入到I/O设备的值。

图8.28 地址映射I/O所需的硬件

Processor：微处理器

Address Decoder：地址译码器

Memory：存储器

I/O device: I/O设备

例 8.17 与I/O设备通讯

假设图8.28中的I/O设备1被指定到了内存地址0xFFFFFFF4。写出把值7写入到I/O设备1和从I/O设备1读出输出值的MIPS汇编语言。

解：以下MIPS汇编代码把值7写入到I/O设备1.

addi $t0 , $0, 7

sw $t0 , 0xFFF4($0)　 # FFF4 is sign-extended to 0xFFFFFFF4

因为地址为0xFFFFFFF4，地址译码器将设置WE1，同时设置MemWrite 为TRUE。WriteData总线的值（7）被写入到与I/O设备1输入管脚相连的寄存器中。

为了从I/O设备读出，处理器应执行以下的MIPS汇编代码。

lw $t1 , 0xFFF4 ($0)

因为地址译码器发现地址位0xFFFFFFF4且Mem Write为FALSE，所以它把RDsel1:0设置为01。I/O设备1的输出通过多路选择器传递到ReadData总线，然后被读入到处理器的$t1中。

与I/O设备通讯的软件称为设备驱动程序（device driver）。读者可能已经下载或者安装过打印机或者其他I/O设备的设备驱动程序。编写一个设备驱动程序需要详细了解I/O设备硬件知识。其他程序调用设备驱动提供的函数来访问设备，而不需知道底层设备的硬件。

与I/O设备相关联的地址通常称为I/O寄存器，因为它们可能与I/O设备的物理寄存器一致，如图8.28中所示。

本章后面小节将提供I/O设备的具体例子。8.6小节在嵌入式系统范畴调查I/O设备，展示了如果使用基于MIPS的微控制器来控制许多物理设备。8.7小节调研PC中使用的主要I/O系统。

#### 8.6 嵌入式I/O系统

嵌入式系统使用一个处理器来控制与物理环境的交互。嵌入式系统一般围绕着微控制器单元（microcontroller unit，MCU）来构造，MCU结合了一个微处理器与一组容易使用的外围设备，例如通用数字和模拟I/O引脚、串行端口、计时器等。微控制器通常是廉价的，并且通过集成大部分必要部件到单一芯片上来使系统成本和尺寸最小化。大多数嵌入式系统比一角硬币更小、更轻，功率只有几毫瓦，成本从几角到几美元不等。微控制器依据处理的数据量大小来进行分类。8位微控制器是最小和最便宜的，而32位微控制器则提供更多内存和更高性能。

为了具体化，本节将在商业微控制器的背景下介绍嵌入式系统I/O。具体而言，我们将重点放在PIC32MX675F512H，这是基于32位MIPS微处理器的Microchip PIC32系列微控制器。 PIC32系列还具有丰富种类的片上外设和存储器，使整个系统的构建能少用外部元件。我们选择这个系列，是因为它价格便宜和易于使用的开发环境，它是基于本书所讲解的MIPS体系结构，而且Microchip是一家领先的微控制器供应商，每年销售大于十亿块芯片。不同制造商生产出来的微控制器I/O系统颇为相似，所以在PIC32系列中所阐述的原理适用于其他微控制器。

本节的其余部分将阐述微控制器如何进行通用的数字、模拟和串行I/O操作。计时器通常用于生成或测量精确的时间间隔。本节最后以其它有趣的外围设备，如显示器、马达、和无线链接来结束。

##### 8.6.1 PIC32MX675F512H微控制器

图8.29展示了PIC32系列微控制器的框图。该系统的心脏是一个32位MIPS处理器。处理器通过32位总线连接存储程序的闪存和存储数据的SRAM。PIC32MX675F512H具有512KB闪存和64KB RAM；其他风格可能包含16至512KB闪存和4到128KB RAM，根据不同价格而变化。高性能外设，如USB和以太网，也直接通过总线矩阵与RAM沟通。低性能外设，包括串行端口、计时器和A/D转换器，共用一个独立的外围总线。该芯片还包含生成时钟信号的时钟产生电路和检测芯片正在通电或者将要断电的电压感测电路。

图8.30显示了微控制器的虚拟内存映射。编程使用的所有地址都是虚拟的。MIPS架构提供32位地址空间来访问多达232 bytes = 4 GB的内存，但只有存储器的一小部分是实际在芯片上实现。从地址0xA0000000-0xBFC02FFF的相关分区包括内存、闪存、以及用来与外围设备进行通信的特殊功能寄存器（special function register，SFR）。需要注意的是额外有一个12 KB的引导闪存（Boot Flash ），通常进行一些初始化，然后跳转到闪存中的主程序。当上电复位时，程序计数器初始化为在地址0xBFC00000的引导闪存起始点。

图8.29 PIC32MX675F512H 框图

图8.30 PIC32内存映射

图8.31展示了微控制器的引脚。引脚包括电源和接地、时钟、复位和多个可被用于通用I/O和/或专用外围设备的I/O引脚。图8.32显示了64引脚薄型四方扁平封装（Thin Quad Flat Pack，TQFP）的微控制器照片，引脚以20密耳（0.02英寸）的间隔分布在封装四边。微控制器也有包含更多I/O引脚的100引脚封装；该版本的零件编号以L而不是H结束。

图8.33展示了连接着最低运行配置的微控制器，包括电源、外部时钟、复位开关和编程连接线插孔。PIC32和外部电路挂载在印刷电路板上；这个电路板可能是实际的产品（例如，一个烤面包机控制器），或者便于在测试过程中轻松访问芯片的开发板。

图8.29 PIC32MX6xxFxxH引脚分布。黑色引脚可以接受5 V电压。

图8.30 64引脚TQFP封装的PIC32

图8.33 PIC32基本操作电路

LTC1117-3.3稳压器可接受4.5—12V（例如从墙上变压器、电池或外接电源输入），然后使其降到电源引脚需要的稳定3.3V。PIC32有多个VDD和GND引脚，通过提供低阻抗路径以降低电源噪声。各种各样的旁路电容提供充电贮存，以在电流需求突然变化的情况下保持电源供应稳定。一个旁路电容也连接到VCORE引脚，该引脚服务于一个内部1.8 V电压调节器。PIC32通常从电源供应获得1-2mA/MHz 电流。例如，在80MHz时，最大功耗为P = VI =（3.3 V）（120 mA）= 0.396 W。64引脚TQFP的热阻为47℃/W，因此若无散热器或冷却风扇，芯片可能会升温至约19℃。

运行频率高达50MHz的外部振荡器可以连接到时钟引脚。在本实例电路中，振荡器工作在40.000MHz。或者，微控制器可被编程为使用8.00MHz±2％的内部振荡器。这是不太精确的频率，但也足够好。I/O设备（如串行端口、A/D转换器、计时器）的外设总线时钟（PBCLK）通常运行在主系统时钟（SYSCLK）速率的一小部分（例如，一半）。这个时钟方案可以通过在MPLAB开发软件设置配置位，或者将下面代码放在C程序的开头。

#pragma config FPBDIV = DIV\_2 // peripherals operate at half

// sysckfreq (20 MHz)

#pragma config POSCMOD = EC // configure primary oscillator in

// external clock mode

#pragma config FNOSC = PRI // select the primary oscillator

提供复位按钮使芯片进入已知的操作初始状态总是很便利的。复位电路包括一个按钮开关和一个连接到复位引脚（）的电阻。复位引脚是低电平有效，表明当复位引脚为0时，处理器复位。当不按下按钮，开关打开，电阻拉动复位引脚到1，允许处理器正常运行。当按下按钮时，开关关闭，复位引脚被拉下到0，迫使处理器复位。当接通电源时，PIC32自动复位。

微控制器最简单的编程方法就是使用Microchip In Circuit Debugger（ICD）3，亲切地称为冰球（puck）。ICD3，如图8.34所示，允许程序员从PC与PIC32通信，下载代码和调试程序。ICD3连接到PC的USB端口，并连接到PIC32开发板的六针RJ-11标准连接器。RJ-11连接器常用于美国电话插孔。ICD3通过双线电路内串行编程接口（In-Circuit Serial Programming interface）与PIC32进行通信，串行编程接口包含一个时钟和一个双向数据引脚。你可以使用Microchip免费的MPLAB集成开发环境（Integrated Development Environment，IDE）写汇编语言程序或C程序，进行仿真调试，以及通过ICD下载到开发板并进行测试。

PIC32微控制器的大多数引脚默认为通用数字I/O引脚。因为该芯片具有有限数量的引脚，这些相同的引脚共享用于专用I/O功能，例如串行端口、模拟数字转换器输入等，这些引脚当相应的外设启动时有效。程序员有责任使得在任意时间里每个引脚只有一个用途。8.6节余下部分将详细探索微控制器的I/O功能。

微控制器的功能远远超出在本章的有限空间里所能涵盖的内容。请参见制造商的数据手册以了解更多细节。尤其是，Microchip的PIC32MX5XX/6XX/7XX系列数据手册和PIC32系列参考手册，它们具有权威性和可读性。

图8.34 Microchip ICD3

##### 8.6.2 通用数字I/O

通用I/O（general-purpose I/O，GPIO）引脚用于读写数字信号。例如，图8.35展示了连接到12位GPIO端口的八个发光二极管（light-emitting diode，LED）和四个开关。电路图显示端口12个引脚的名称和编号；这告诉了程序员每个引脚的功能，并告诉了硬件设计人员如何设置物理连接。这些LED用1来驱动时则发光，用0来驱动时则灭灯。开关闭合时产生1，打开时则产生0。微控制器可以使用这个端口同时驱动LED和读取开关状态。

PIC32将GPIO组合成同时具备读写功能的端口。PIC32的端口名为RA、RB、RC、RD、RE、RF和RG；简称为A口、B口等等。虽然PIC32没有足够的引脚来为它的所有端口提供信号，每个端口可能有多达16个GPIO引脚。

图8.35连接到12位GPIO端口的发光二极管和开关

每个端口由TRISx和PORTx这两个寄存器控制，其中x是一个字母（A-G）以表示当前端口。TRISx寄存器确定端口的引脚是输入还是输出，而PORTx寄存器表明某个数值是从输入读取或驱动到输出去。每个寄存器的16位低位对应GPIO端口的16个引脚。当TRISx寄存器的某个位为0，该引脚是输出，而当它为1时，该引脚为输入。谨慎的做法是将未使用的GPIO引脚设置为输入（默认状态），以使它们不会不经意驱动为麻烦的数值。

每个寄存器映射到虚拟内存（0xBF80000-BF8FFFF）特殊功能寄存器的一个字。例如，TRISD映射到地址0xBF8860C0和PORTD映射到地址0xBF8860D0。p32xxxx.h头文件声明这些寄存器为32位无符号整数。因此，程序员可以通过名称访问它们，而不必查找对应地址。

例8.18 用于开关和LED的GPIO

写一个C程序读取4个开关信号，使用图8.35所示的硬件点亮底部对应的4个LED。

解：配置TRISD使得引脚RD[7:0]为输出和引脚RD[11:8]为输入。然后通过引脚RD[11:8]读取开关信号，将信号的值写回RD[3:0]，从而点亮正确的LED。

{代码}

例8.18中同时写入整个寄存器，但访问单个寄存器位也是可以的。例如，以下代码把第一个开关的数值复制到第一个LED。

{代码}

每个端口也有对应的SET和CLR寄存器，它们可以使用标明哪些位是置位或复位的掩码来写入。例如：

{代码}

以上代码将PORTD的第一位和第三位置位，将第四位复位。如果PORTD末端4位原是1110， 那么运行上述代码则变为0111。

可用的GPIO引脚数由封装尺寸决定。表8.5总结了各种封装情况下的可用引脚。例如，100引脚TQFP提供A口的引脚RA[15:14]、RA[10:9]和RA[7:0]。注意RG[3:2]只用于输入。RB[15:0]共享为模拟输入引脚，其它引脚也有多种功能。

逻辑电平是LVCMOS兼容的。输入引脚逻辑电平期望值为VIL = 0.15VDD和VIH = 0.8VDD，或0.5 V 和2.6 V （假设VDD 为3.3V）。只要输出电流Iout不超过微小的7 mA，输出引脚生成VOL为0.4 V 和 VOH为2.4 V。

表8.5 PIC32MX5xx/6xx/7xx GPIO引脚

##### 8.6.3 串行I/O

如果微控制器需要发送比GPIO可用引脚数更多的位数，它必须把消息拆分成多个较小的传输。在每个步骤中，它可以发送任意一个比特或几个比特。 前者称为串行I/O（serial I/O），后者称为并行I/O（parallel I/O）。串行I/O的普及是因为它使用少量电线和对很多应用来说足够快。事实上，它的普及程度高到建立了多个串行I/O标准，通过这些标准，PIC32的专用硬件能方便地发送数据。本节描述串行外设接口（Serial Peripheral Interface，SPI）和通用异步接收/发送器（Universal Asynchronous Receiver/Transmitter，UART）标准协议。

其它常用串行标准包括内部集成电路（ Inter-Integrated Circuit，I2C）、通用串行总线（Universal Serial Bus，USB）和以太网。 I2C是包含一个时钟和一个双向数据引脚的双线接口；它的使用方式与SPI类似。USB和以太网较为复杂，高性能标准分别在8.7.1节和8.7.4节描述。

**串行外设接口（SPI）**

SPI是一个简单的同步串行协议，易于使用，而且相对速度较快。物理接口由三个引脚组成：串行时钟（Serial Clock，SCK）、串行数据输出（Serial Data Out，SDO）和串行数据输入（Serial Data In，SDI）。 SPI连接主设备（master device）和从设备（slave device），如图8.36（a）所示。主设备生成时钟信号。它通过SCK发送一系列时钟脉冲来启动通信。如果它想将数据发送给从设备，它把数据从最高有效位开始放在SDO。从设备可能通过将数据放在主设备的SDI同时响应。图8.36（b）展示了8位数据传输的SPI波形。

PIC32有多达四个SPI端口，命名为SPI1—SPI4。每个端口都可以作为主设备或从设备。本节介绍了主模式操作，从模式是相似的。要使用SPI端口，PIC®单片机程序必须首先配置端口。然后，它可以将数据写入寄存器；数据被串行发送给从设备。可以使用另一个寄存器收集从从设备接收的数据。当传输完成时，PIC32可以读取所接收的数据。

每个SPI口与4个32位寄存器相关联：SPIxCON、SPIxSTAT、SPIxBRG和SPIxBUF。例如，SPI1CON是SPI端口1的控制寄存器，用于启动SPI和设置属性，如传输比特数和时钟极性。表8.6列出CON寄存器所有位的名称和功能，复位的默认值全部为0。大部分的功能​​，如成帧、增强缓冲、从设备选择信号和中断未在本小节中使用，但可以在数据手册里找到。STAT是状态寄存器，指示接收寄存器是否已满。这个寄存器的详细内容在PIC32数据手册里有完整的描述。

图8.36 SPI连接和主设备波形

Master: 主设备

Slave: 从设备

表8.6 SPIxCON寄存器字段

Bits: 比特位

Name: 名称

Function: 功能

Enable framing: 成帧使能

Frame sync pulse direction control：帧同步脉冲方向控制

Frame sync polarity (1 = active high)：帧同步极性（1 = 高电平有效）

Enable slave select generation in master mode：主模式下从设备选择信号生成使能

Frame sync pulse width bit：帧同步脉冲宽度位

(1 = 1 word wide, 0 = 1 clock wide)：（1 = 1字宽，0 = 1时钟周期宽）

Frame sync pulse counter：帧同步脉冲计数器

(frequency of sync pulses)：（同步脉冲频率）

Master clock select：主时钟选择

(1 = master clock, 0 = peripheral clock)：（1 = 主时钟，0 = 外设时钟）

unused：不使用

Frame sync pulse edge select：帧同步脉冲边沿选择

Enable enhanced buffering：增强缓冲使能

SPI ON：SPI启动

Stop SPI when CPU is in idle mode：在CPU空闲模式下停止SPI

disable SDO pin：停用SDO引脚

32-bit transfers：32位传输

16-bit transfers：16位传输

Sample phase (see Figure 8.39)：采样相位（参见图8.39）

Clock edge (see Figure 8.39)：时钟边沿（参见图8.39）

Enable slave select：从设备选择使能

Clock polarity (see Figure 8.39)：时钟极性（参见图8.39）

Enable master mode：从模式使能

disable SDI pin：停用SDI引脚

Transmit buffer interrupt mode：发送缓冲中断模式

Receive buffer interrupt mode：接收缓冲中断模式

串行时钟频率可以被配置为外设时钟频率的一半或更少。SPI对数据速率没有理论极限，虽然在使用试验电路板上连线运行在1MHz以上时可能会遇到噪声问题，它可以容易地在印刷电路板上以数十MHz的频率操作。波特率寄存器（baud rate register，BRG）根据以下公式依据外设时钟设置SCK速率：

（8.3）

BUF是数据缓冲器。写入BUF的数据通过SPI的SDO引脚传输，SDI引脚所接收的数据可以在传输完成后通过读取BUF而找到。

为了准备SPI主模式，首先通过将CON寄存器第15位（ON位）清除为0而将其关闭。通过读取BUF寄存器来清除任何可能在接收缓冲区里的数据。通过写BRG寄存器设置所需的波特率。例如，如果外设时钟为20MHz和所需的波特率为1.25MHz，设置BRG为[20/（2×1.25）] - 1 = 7。通过设置CON寄存器第5位（MSTEN）为1而把SPI置于主模式下。设置CON寄存器第8位（CKE）使SDO位于时钟上升沿的中间。最后，通过设置CON寄存器ON位重启SPI。

为了发送数据到从设备，将数据写入到BUF寄存器。数据将被串行传输，从设备将同时发送数据回主设备。等到STAT寄存器第11位（SPIBUSY位）为0，这表明SPI已完成其操作。然后，可以从BUF读取从从设备接收的数据。

PIC32的SPI端口是高度可配置的，这样它可以跟多种串行设备沟通。不幸的是，这导致了可能错误配置端口和获取乱码数据传输。时钟和数据信号的相对时序通过三个CON寄存器位（CKP、CKE和SMP）来配置。缺省情况下，这些位都是0，但图8.36（b）的时序采用CKE = 1。主设备在SCK下降沿改变SDO，因此从设备应当使用上升沿触发的触发器在其上升沿进行数值采样。主设备希望SDI在SCK上升沿保持稳定，所以从设备应在下降沿改变，如时序图所示。CON寄存器的MODE32和MODE16位指定应该发送32位或16位字；这些位的默认值都是0，表示进行8位传输。

例8.19 通过SPI发送和接收字节

设计一个系统以在PIC®主设备与FPGA从设备间通过SPI进行通信。为微控制器编写C程序，发送字符‘A’并接收回一个字符。为FPGA上的SPI从设备编写HDL代码。如果从设备只需要接收数据，如何简化？

解：图8.37展示了使用SPI端口2的设备间连接。引脚编号从器件数据手册（例如，图8.31）获得。注意，图上显示了引脚编号和信号名称以标示其物理和逻辑连接。这些引脚也被GPIO端口RG[8：6]使用。当SPI启用，G端口这些位不能用于GPIO。

图8.37 PIC32和FPGA之间的SPI连接

Master: 主设备

Slave: 从设备

下面的C程序代码将SPI初始化，然后发送和接收字符。

{代码}

FPGA的HDL代码在下面列出，时序图如图8.38所示。FPGA采用移位寄存器来保存从主设备接收的数据位和要发送到主设备的剩余数据位。在复位后SCK第一个上升沿以及往后每8个时钟周期，从d产生的一个新字节加载到移位寄存器。在每个后续周期，一个比特位数据移入FPGA的SDI和一个比特位数据移出FPGA的SDO。SDO一直被延迟到SCK下降沿，以便它被主设备在下一个上升沿采样。经过8个周期后，接收字节可以在q上找到。

{代码}

图8.38 SPI从设备电路和时序

如果从设备仅需要从主设备接收数据，它简化成一个简单的移位寄存器，见下面的HDL代码。

{代码}

有时候，有必要改变配置位以跟不同时序期望值的设备进行通信。当CKP =1，SCK反转。当CKE =0时，相对于数据提前半个周期进行时钟切换。当SAMPLE=1时，主设备延迟半个周期进行SDI采样（从设备应确保它在那个时间段里是稳定的）。这些模式显示在图8.39中。要知道，不同的SPI产品可能为这些选项使用不同的名称和极性；仔细检查设备的波形。使用示波器检查SCK、SDO和SDI，对解决通信困难有帮助。

图8.39 由CKE、CKP和SAMPLE控制的时钟和数据时序

**通用异步收发器（UART）**

UART是两个系统之间不发送时钟信号而进行通信的一种串行I/O外设。但这两个系统必须事先就使用何种数据速率达成一致，每个系统必须生成自己的时钟。虽然这些系统时钟可能有小频率误差和未知的相位关系，但UART能设法坐到可靠的异步通信。UART被用于诸如RS-232和RS-485等协议。例如，计算机串口使用RS-232C标准，这由电子工业协会（Electronic Industries Association）于1969年提出。该标准的最初设想是连接数据终端设备（Data Terminal Equipment，DTE）和数据通信设备（Data Communication Equipment，DCE），例如计算机主机和调制解调器。尽管与SPI相比，UART相对较慢，但该标准已经存在了很久，因而它们现在仍然重要。

图8.40（a）表示一个异步串行链路。 DTE在TX线上发送数据到DCE，在RX线上接收数据。图8.40（b）展示出在上述其中一条传输线上以9600波特的数据速率发送一个字符的过程。链路空闲时，其逻辑为‘1’。每个字符传输包括一个起始位（0）、7-8个数据位、一个可选的奇偶位、以及一个或多个停止位（1）。UART检测从空闲状态到起始状态的下降沿，从而将传输锁定在适当的时间。虽然7个数据位足以发送ASCII字符，但通常使用8个数据位，因为这样可以传递任意一个字节数据。

图8.40 异步串行链路

Idle: 空闲

Start: 起始

Stop: 停止

也可以发送一个附加位（奇偶校验位），使系统能检测在传输过程中是否有数据位遭到损坏。它可以被配置为偶数或奇数；偶校验意味着传输数据和奇偶校验位的集合共有偶数个1；也就是说，奇偶位是对数据位进行异或（XOR）操作的结果。然后接收器可以检查是否收到偶数个1，如果没有则生成错误信号。奇校验是相反的，因此奇偶位是对数据位进行同或（XNOR）操作的结果。

一个常见的​​选择是使用8个数据位、无奇偶校验、1个停止位，总共10个符号来传达一个8位字符信息。因此，信号传输速率单位是波特而不是位/秒（bits/sec）。例如，9600波特表示9600符号/秒（symbols/sec），或960字符/秒（characters/sec），因而960×8 = 7680位/秒的数据速率。两个传输系统都必须配置合适的波特率和数据、奇偶校验和停止位的数量，否则数据会出现乱码。这是一个麻烦，尤其对非技术用户而言，这是为什么通用串行总线（USB）在PC系统中取代UART的原因。

常用的波特率包括300、1200、2400、9600、14400、19200、38400、57600和115200。较低波特率在七八十年代用于调制解调器通过电话线发送一系列音调数据。在现代系统中，9600和115200是两个最常见的波特率；当对速度没有要求时使用9600，115200是最快的标准速率，虽然跟其他现代串行I/O标准相比仍然缓慢。

RS-232标准定义了几个额外的信号。请求发送（Request to Send，RTS）和清除发送（Clear to Send，CTS）信号，可用于硬件握手（hardware handshaking）。它们可以在以下两种模式的任意一种模式下工作。在流控制模式（flow control mode）下，当DTE已准备好从DCE接收数据时，DTE清除RTS为0时。同样，当DCE已准备好从DTE接收数据，DCE清除CTS为0时。一些数据手册使用上划线来表示它们是低电平有效。在较老的单工模式下，当DTE准备好发送数据时，DTE清除RTS为0时。当DCE准备好接收数据时，DCE通过清除CTS回复DTE。

一些系统，特别是那些通过电话线连接的系统，还使用数据终端就绪（Data Terminal Ready，DTR）、数据载波检测（Data Carrier Detect，DCD）、数据集就绪（Data Set Ready，DSR）和环指示符（Ring Indicator，RI）来指示设备何时连接到线路。

最初的标准推荐了大块的25针DB-25连接器，但PC精简成一个9针DE-9公连接器它的引脚图如图8.42（a）所示。电缆线通常如图8.42（b）所示直接连接。但是，如果直接连接两个DTE，可能需要图8.42（c）所示的零调制解调器连线（null modem cable）来交换RX和TX以完成握手。有些连接器是公连接器，而有些是母连接器。总之，可能要使用一大箱电缆和一定量的推测工作才能通过RS-232连接两个系统，再次说明转向USB的必要性。幸运的是，嵌入式系统通常使用一个简化的3-或5-导线设置，由GND、TX、RX组成，可能包括RTS和CTS。

图8.41Cap’n Crunch水手口哨。

图8.42 DE-9公电缆（a）引脚配置，（b）标准布线，（c）零调制解调器连线

RS-232使用3至15 V电压表示0，使用-3至-15 V电压表示1；这就是所谓的双极信号（bipolar signaling）。收发器转换UART的数字逻辑电平为RS-232所期望的正负电平，并且还提供静电放电保护，以在用户插入电缆时保护串口免受损坏。MAX3232E是与3.3V和5 V数字逻辑兼容的收发器。它包含一个与外部电容相结合的电荷泵，从单一低电压电源产生±5 V输出。

PIC32有6个UART，名为U1-U6。正如SPI，PIC®单片机程序必须首先配置端口。与SPI不同的是，读取和写入能独立操作，因为两个系统都可能只发送不接收，反之亦然。每个UART与5个32位寄存器相关联：UxMODE、UxSTA（状态）、UxBRG、UxTXREG和UxRXREG。例如，U1MODE是UART1的模式寄存器。模式寄存器用于配置UART，STA寄存器用于检查数据是否可用。 BRG寄存器用于设置波特率。数据通过写入TXREG或读取RXREG来完成发送或接收。

模式寄存器（MODE）默认为8个数据位、1位停止位、无奇偶校验和没有RTS/CTS流控制信号，所以大多数应用程序员只对第15位（ON位，UART使能位）感兴趣。

STA寄存器包含启用发送和接收引脚的比特位，也包含检查发送和接收缓冲器是否已满的比特位。设置UART ON位后，程序员还必须设置STA寄存器的UTXEN和URXEN位（第10位和第12位）来启用这两个引脚。UTXBF（第9位）指示发送缓冲区已满。 URXDA（第0位）表明接收缓冲区具有可用数据。STA寄存器还包含表示奇偶校验和帧错误的比特位；如果起始或停止位没有在预期时间内找到，则发生帧错误。

16位BRG寄存器用于将波特率设置为外围总线时钟的一小部分。

（8.4）

表8.7列出了常用目标波特率前提下的BRG设置，假设一个20 MHz的外部时钟。有时不可能正好达到目标波特率。然而，只要频率误差远低于5％，将能在一个10位帧的持续时间内保持发射器和接收器之间的小相位误差，以便接收到正确的数据。然后，系统将在下一个起始位重新同步。

要传输数据，需等到STA.UTXBF清空，这表明发送缓冲区有可用空间，然后写入字节到TXREG。要接收数据，检查STA.URXDA，看数据是否已经到了，然后从RXREG读取字节。

表8.7 针对20 MHz外部时钟的BRG设置

Target Baud Rate：目标波特率

Actual Baud Rate：实际波特率

Error：误差

例8.20 与PC的串行通信

开发PIC32与PC沟通的电路及C程序，通过包含8个数据位、1个停止位和无奇偶校验的串行端口以115200波特率进行通信。PC运行控制台程序，如PuTTY[[1]](#footnote-1)，通过串行端口进行读写操作。该程序应要求用户输入一个字符串，然后应该告诉用户她输入了什么。

解：图8.43展示了串行链路电路图。因为很少PC仍有物理串行端口，我们使用可插入式USB到RS-232 DB9串行适配器（参见plugable.com），如图8.44所示，提供一个到PC的串行连接。适配器连接到一个DE-9母连接器上，该连接器焊接到馈入收发器的电线上，从而将双极RS-232电平转换为PIC32微控制器的3.3 V电平。在这个例子中，我们选择PIC32上的UART2端口。微控制器和PC都是数据终端设备，所以TX和RX引脚必须是交叉连接的。不使用PIC32 RTS/CTS握手，而DE9连接器的RTS和CTS被连接在一起，使得PC自己完成握手。

要配置PuTTY使用串行链路工作，设置连接类型为串行和速度为115200。设置串行线路到由操作系统分配的COM端口为串口转USB适配器（Serial to USB Adapter）。在Windows操作系统中，这可以在设备管理中找到；例如，它可能是COM3。在连接→串口选项卡，设置流控制为NONE或RTS/CTS。在终端选项卡，设置本地回显为强制开，从而在终端显示你键入的字符。

图8.43 PIC32到PC串行链路

MAX3232E Transceiver：MAX3232E收发器

Female DE-9 connector：DE-9母连接器

Plugable USB to RS-232 serial adapter：可插入式USB到RS-232串行适配器

To PC USB port：至PC USB端口

图8.44 可插入式USB到RS-232 DB9串行适配器

该代码是下面列出。终端程序的输入键在C语言中对应一个回车符称为‘\r’，它的ASCII码是0X0D。 要在打印输出时跳到下一行的开头，同时发送‘\n’和‘\r’（新一行和回车）字符[[2]](#footnote-2)。

初始化和读写串口的函数构成了一个简单的设备驱动程序。设备驱动程序为程序员和硬件之间提供了抽象和模块化层次，以使得程序员使用设备驱动程序时不必了解UART的寄存器集。它还简化了将代码移植到不同微控制器的过程；设备驱动程序必须重写，但调用它的代码可以保持不变。

main函数演示了使用putstrserial和getstrserial函数打印输出到控制台和从控制台读取数据。它还演示了如何使用stdio.h中的printf函数自动通过UART2打印。不幸的是，PIC32库目前不能优雅地支持在UART使用scanf函数，但可以使用getstrserial。

{代码}

在PC上用C程序与串口进行通讯是有点麻烦的，因为串口驱动程序库没有跨操作系统规范。而其他编程环境，如Python、MATLAB或LabVIEW，可顺利进行串行通信。

##### 8.6.4 计时器

嵌入式系统通常需要测量时间。例如，微波炉需要一个计时器来跟踪日期时间，另一个计时器来测量烹饪时间。它还可能使用再一个计时器产生马达脉冲以旋转食物盘，以及第四个计时器通过仅在每秒的一小部分时间里激活微波能量以控制功率设置。

PIC32在板上有5个16位计时器。计时器1被称为A型计时器，可以接受异步外部时钟源，例如一个32KHz钟表晶体。计时器2/3和4/5是B型计时器。 它们与外设时钟同步运行并且可配对（例如，2与3配对）以构成32位计时器，用于测量长时间段。

每个计时器与3个16位寄存器相关联：TxCON、TMRx和PRx。例如，T1CON是计时器1的控制寄存器。CON是控制寄存器，TMR包含当前时间计数，PR是周期寄存器。当计时器达到指定时间，它回到0并设置IFS0中断标志寄存器的TxIF位。程序可通过查询该位来检测溢出。或者，它可以生成一个中断。

缺省情况下，每个计时器相当于一个16位计数器，累加内部外设时钟（在这个例子里是20MHz）的节拍。CON寄存器的第15位（ON位）启动计时器计数。CON寄存器的TCKPS位指定预分频器（prescalar），如表8.8和表8.9所示。用k:1进行预分频使计时器在每k个节拍只计数一次；这可用于产生更长的时间间隔，特别是当外设时钟跑快了。其他CON寄存器位在A类和B类计时器中稍有不同，详见数据手册。

表8.8 A类计时器预分频器

Prescale：预分频

表8.9 B类计时器预分频器

Prescale：预分频

例8.21 延迟产生

编写2个函数，使用计时器1创建指定数量的微秒级和毫秒级延迟。假设外设时钟运行在20MHz。

解：每微秒为20个外设时钟周期。我们根据示波器观察经验，delaymicros函数有大约6微秒的时间开销用于函数调用和计时器初始化。所以，我们设置PR为。因此，该函数在持续时间小于6微秒时是不准确的。在一开始的检查可以防止16位PR的溢出。

delaymillis函数重复调用delaymicros（1000）来创建适当数量的1毫秒延迟。

{代码}

另一个方便的计时器功能是门控时间累加（gated time accumulation），其中，计时器只当外部引脚为高电平才计数。这允许计时器可测量外部脉冲的持续时间。它CON寄存器启用。

##### 8.6.5 中断

定时器通常与中断结合使用，从而使一个程序可以照常运行，并周期性地在计时器产生中断时处理相应任务。6.7.2节将从体系结构的角度来描述MIPS中断。本节将探讨如何在PIC32中使用中断。

中断请求发生硬件事件发生时，例如计时器溢出、UART接收到字符、或某些GPIO引脚切换。每种类型的中断请求都设置中断标志状态（Interrupt Flag Status，IFS）寄存器的特定位。然后处理器检查中断允许控制（Interrupt Enable Control，IEC）寄存器的相应位。如果该位被设置，微控制器应通过调用中断服务程序（interrupt service routine，ISR）来响应中断请求。ISR是带有void参数的函数，它处理中断并在返回前清除IFS的相应位。PIC32中断系统支持单向量和多向量模式。在单向量模式中，所有中断调用相同的ISR，它必须检查CAUSE寄存器以确定中断的原因（如果可能会出现多种类型的中断）并相应地处理它。在多向量模式中，每种类型的中断调用不同的ISR。INTCON寄存器的MVEC位用于判定具体模式。在任何情况下，MIPS中断系统必须在它接受任何中断前使用ei指令启用。

PIC32还允许每个中断源具备可配置的优先级（priority）和子优先级（subpriority）。优先级范围是0-7，7为最高级。较高优先级的中断会抢占正在处理的中断。例如，假设一个UART中断优先级为5，而一个计时器中断优先级为7。如果在程序正常执行时，在UART出现一个字符，那么将产生一个中断使微控制器可以从UART读取数据并处理它。如果在UART ISR有效时计时器溢出，ISR自身将被中断使微控制器可以立即处理计时器溢出。当它完成后，它将在返回到主程序之前完成UART中断。另一方面，如果计时器中断优先级为2，将先完成UART ISR，再调用计时器ISR，最后回到主程序。

子优先级的范围是0-3。如果两个具有相同优先级的事件同时待解决，具有更高子优先级的将被优先处理。但是，子优先级不会引起一个新的中断抢占当前正被服务的相同优先级中断。每个事件的优先级和子优先级在IPC寄存器配置。

每个中断源有个在0-63范围内的向量号。例如，计时器1溢出中断为向量4，UART2 RX中断为向量32，通过RD0引脚变化触发的INT0外部中断为向量3。IFS、IEC和IPC寄存器字段对应向量号在PIC32数据手册中说明。

ISR函数声明由两个特殊\_ \_attribute\_ \_指示符标记，说明优先级和向量号。编译器使用这些属性关联ISR与相应的中断请求。Microchip PIC32 MCU MPLAB® C编译器用户手册里有关于编写中断服务程序的更多信息。

例8.22 周期性中断

使用中断编写一个使LED以1Hz闪烁的程序。

解：我们将设置计时器1每隔0.5秒溢出，在中断处理程序中将LED在ON和OFF之间切换。

下面的代码演示了多向量模式操作，即使只有计时器1中断是实际启用的。blinkISR函数的属性表明它具有优先级7（IPL7），为向量4（计时器1溢出向量）。ISR切换LED和清除IFS0的计时器1中断标志（T1IF）位，然后返回。

initTimer1Interrupt函数设置计时器周期为1/2秒，使用256：1预分频和39063拍。它启用多向量模式。优先级和子优先级分别在IPC1寄存器的4:2位和1:0位指定。计定时器1中断标志（T1IF，IFS0第4位）被清零，计时器中断使能（T1IE，IEC0第4位）被设置为从计时器1接受中断。最后，asm指示符用于生成ei指令以启用中断系统。

main函数在初始化计时器中断后，在一个while循环里等待。但是，它可以做一些更有趣的事情，例如与用户玩游戏，而中断程序仍确保LED以正确速率闪烁。

{代码}

##### 8.6.6 模拟I/O

真实的世界是模拟信号的世界。许多嵌入式系统需要通过模拟输入和输出与外界交互。它们使用模拟—数字转换器（analog-to-digitalconverter，ADC）将模拟信号量化成数字数值，使用数字—模拟转换器（digital-to-analog-converter，DAC）做相反的事情。图8.45展示了这些组件符号。转换器的特征由它们的分辨率、动态范围、采样频率和精确度来决定。例如，一个ADC可能有N = 12位分辨率，范围Vref-—Vref+为0—5V，采样频率为fs =44KHz，精确度为±3最低有效位（LSB）。采样频率单位也可写成样本/秒（samples per second，sps），其中1sps=1Hz。模拟输入电压Vin（t）和数字样本X[n]之间的关系是

（8.5）

图8.45 ADC和DAC符号

例如，2.5V输入电压对应于1000000000002=80016的输出，不确定性高达3个LSB。

同样地，Vref = 2.56V时，DAC可能具有N = 16位分辨率。它产生的输出为

（8.6）

许多微控制器具有内置的中等性能ADC。对于更高性能（例如，16位分辨率或超过1MHz的采样频率），则通常需要使用连接到微控制器的独立ADC。较少微控制器具有内置DAC，所以可能使用独立芯片。然而，微控制器经常使用一种称为脉冲宽度调制（pulse-width modulation，PWM）技术仿真模拟输出。本节在PIC32微控制器背景下描述模拟I/O。

**模拟/数字转换**

PIC32中有一个10位ADC，最高速度为100万个样本/秒（Msps）。ADC可以通过一个模拟多路选择器连接到任意16个模拟输入引脚。模拟输入称为AN0-15，它们与数字I/O端口RB共享引脚。默认情况下，Vref +是模拟VDD引脚和Vref-是模拟GND引脚；在所述的系统里，这就是3.3和0V。程序员必须初始化ADC，指定采样引脚，等待足够长的时间进行电压采样，开始转换，等待完成，读取结果。

ADC是高度可配置的，可以在可编程时间间隔里自动扫描多个模拟输入并在完成后生成中断。本节简单描述如何读取单一模拟输入引脚；其它功能详见PIC32系列参考手册。

ADC由一群寄存器控制：AD1CON1-3、AD1CHS、AD1PCFG、AD1CSSL和ADC1BUF0-F。AD1CON1是主控制寄存器。它有一个ON位用于启用ADC，一个SAMP位用于控制何时进行采样和转换，以及一个DONE位指示转换完成。 AD1CON3具有ADCS[7：0]位用于控制A/D转换速度。AD1CHS是频道选择寄存器用于指定要采样的模拟输入。AD1PCFG是引脚配置寄存器。当一个位为0，相应的引脚作为模拟输入。当这个位为1，该引脚用作数字输入。ADC1BUF0用于存储10位转换结果。而其他寄存器在这个简单的例子不需要。

ADC使用逐次逼近寄存器在每个ADC时钟周期生成一位结果。对于总共12个ADC时钟/转换，两个额外周期是必需的。为保证正确操作，ADC时钟周期TAD必须至少为65ns。它被设置为外设时钟周期TPB的倍数，根据以下关系式使用ADCS位得到：

（8.7）

因此，对于高达30MHz的外设时钟，ADCS位可保留在默认值0。

采样时间是新输入在转换开始前必需的稳定时间。只要所要采样的电阻源小于5KΩ，采样时间可小到132ns，这只是少量时钟周期。

例8.23 模拟输入

编写程序以读取AN11引脚的模拟数值。

解：initadc函数初始化ADC和选择指定通道。它让ADC停留在采样模式。readadc函数结束采样并开始转换。直到转换完成后，重新开始采样并返回转换结果。

{代码}

**数字/模拟转换**

PIC32没有内置DAC，所以本节介绍使用外部DAC进行D/A转换。本节还阐述PIC32通过并行端口和串行端口与其他芯片进行交互的过程。相同的方法可以用于PIC32与更高分辨率或更快的外部ADC间的交互。

一些数模转换器通过具有N条连线的并行接口接收N位数字输入，而另一些则通过串行接口如SPI接收数据。有些DAC要求正负电源电压供应，而其它的则使用单电源供电操作。一些DAC支持弹性的电源电压范围，而另一些则需要特定的电压。输入逻辑电平应当与数字源兼容。有些DAC生成与数字输入成正比的电压输出，而另一些则产生电流输出；可能需要一个运算放大器来将此电流转换为在所需范围内的电压。

在本节中，我们使用Analog Devices AD558 8位并行DAC和Linear Technology LTC1257 12位串行DAC。这两种DAC产生电压输出，共用同一个5-15V电源，使用VIH = 2.4 V，因而与PIC32的3.3 V输出兼容，采用DIP封装使它们易于装在试验电路板上，并且易于使用。AD558产生范围在0-2.56V的输出，功耗75mW，采用16引脚封装，并有1μs稳定时间从而允许1M样本/秒的输出速率。数据手册可从analog.com获得。LTC1257产生范围在0-2.048V的输出，功耗低于2mW，采用8引脚封装，并有6μs稳定时间。它的SPI运行最高频率为1.4MHz。数据手册可从linear.com获得。Texas Instruments是另一家领先的ADC和DAC制造商。

例8.24 结合外部DAC的模拟输出

画出简单信号发生器的电路图和编写相应软件程序，利用PIC32、AD558和LTC1257生成正弦波和三角波。

解：图8.37展示了使用SPI端口2的设备间连接。引脚编号从器件数据手册（例如，图8.31）获得。注意，图上显示了引脚编号和信号名称以标示其物理和逻辑连接。这些引脚也被GPIO端口RG[8：6]使用。当SPI启用，G端口这些位不能用于GPIO。

电路如图8.46所示。AD558通过RD 8位并行端口连接到PIC32。它将Vout Sense和Vout Select连接在到Vout以设置2.56V满量程输出范围。LTC1257通过SPI2连接到PIC32。两个ADC都采用5V电源供电，并有一个0.1μF去耦电容以降低电源噪声。DAC上的低电平有效芯片使能和负载信号指示何时转换下一个数字输入。当加载新的输入时，它们应该保持为高电平。

该程序如下所示。initio初始化串口和并口，并设置计时器周期以产生所需的输出频率。SPI设置为16位模式，运行在1MHz，但LTC1257只关心发送数据的最后12位。initwavetables预计算正弦和三角波的样值数组。正弦波被设定为12位规模，而三角波被设定为8位规模。每个波一个周期有64个采样点；改变这个数值是在用采样精度交换采样频率。genwaves重复通过采样样本。对于每个样本，它禁用了到DAC的CE和LOAD信号，通过并行和串行端口发送新样本，重新启用DAC，然后等待直到计时器指示到了下一个样本的时间。正弦波和三角波的最小频率是5Hz，这由16位计时器1周期寄存器设置，最大频率605Hz（38.7 Ksamples/s）是按genwaves函数发送每个采样点的时间设置，其中SPI传送占了主要部分。

图8.46 DAC到PIC32的并行和串行接口

{代码}

**脉冲宽度调制**

另一种产生模拟输出方的式是脉冲宽度调制（pulse-width modulation ，PWM），其中产生一个周期性脉冲输出，输出一部分为高电平，其余部分为低电平。占空比是脉冲为高电平部分在一个周期里所占比例，如图8.47所示。输出平均值正比于占空比。例如，如果输出在0和3.3V之间摆动，并具有25％的占空比，则电压平均值为0.25×3.3=0.825V。对PWM信号进行低通滤波可消除振荡使信号得到所需平均值。

图8.47 脉冲宽度调制信号

Pulse width：脉冲宽度

Duty cycle：占空比

Period：周期

PIC32包含五个输出比较（output compare）模块OC1-OC5，每个模块与计时器2或3结合，可以生成PWM输出[[3]](#footnote-3)。每个输出比较模块与3个32位寄存器相关联：OCxCON、OCxR和OCxRS。 CON是控制寄存器。CON寄存器的OCM位应设置为1102以激活PWM模式，ON位也应启用。缺省情况下，输出比较模块使用计时器2运行在16位模式，但OCTSEL和OC32位可以用来选择计时器3和/或32位模式。在PWM模式，RS设置占空比，计时器的周期寄存器PR设置周期，OCxR可以忽略。

例8.25 结合PWM的模拟输出

写一个函数，使用PWM和外部RC滤波器以生成模拟输出电压。该函数应接受0（0V输出）到256（全3.3V输出）之间的输入。

解：使用OC1模块在OC1引脚产生78.125KHz信号。图8.48的低通滤波器具有如下转角频率以消除高速振荡并传递平均值。

（8.4）

图8.48 使用PWM和低通滤波器的模拟输出

计时器应在20MHz工作，每周期有256节拍，因为20MHz/256得到预期78.125KHz的PWM频率。占空比输入是输出为高电平的节拍数。如果占空比为0，输出将保持在低水平。如果是256或更高，输出将维持高电平。

PWM代码使用OC1和计时器2。周期寄存器被设定为255，每周期有256节拍。OC1RS设置为所需的占空比。OC1配置为PWM模式，计时器和输出比较模块被接通。该程序可能会转移到其他任务，而输出比较模块则继续运行。OC1引脚将持续产生PWM信号，直到它被明确关闭。

{代码}

##### 8.6.7 其他微控制器外设

微控制器经常与其他外围设备交互。这小节描述各种常见例子，包括字符模式液晶显示器（liquid crystal display，LCD）、VGA显示器、蓝牙无线链路和电动机控制。标准通讯接口，包括USB和以太网，它们将在8.7.1节和8.7.4节中描述。

**字符LCD**

字符LCD是一个小的液晶显示器，它能够显示一行或几行文字。它们通常用在设备前面板，例如收银机、激光打印机和传真机，这些设备需要显示有限的信息量。它们很容易通过并口、RS-232或SPI接口与微控制器沟通。Crystalfontz America销售各种各样的字符LCD，从8列×1行至40列×4行，可选择颜色、背光源、3.3/5 V操作和日光能见度。数量不多时，他们的液晶显示器需花费20美元或更多，但大批量时价格回落到低于5美元。

本节提供了一个通过8位并行接口连接PIC32单片机和字符LCD的例子。该接口兼容于行业标准HD44780 LCD控制器（最初由Hitachi开发）。图8.49显示了Crystalfontz CFAH2002ATMI-JT 20×2并行液晶显示器。

图8.50展示了字符LCD通过8位并行接口连接到PIC32。逻辑可工作在5V，但与PIC32的3.3V输入兼容。 LCD的对比度是由另一个用电位计产生的电压设定；通常可读性最强是设置为4.2-4.8V。LCD接收三个控制信号：RS（1代表字符，0代表指令）、R/W（1表示从显示屏读取，0表示写入显示屏）和E（高电平表示在下一个字节准备好之前需要至少250ns时间来启用液晶显示器）。当指令被读出，第7位返回忙标志，忙时标记为1，当LCD已准备好接受另一条指令时标记为0。然而，某些初始化步骤和清除指令需要指定的延迟，而不是检查忙标志。

为了初始化LCD，PIC32必须写一系列指令到LCD中，如下所示：

▶VDD加载后，等待>15000微秒

▶写入0x30以设置8位模式

▶等待>4100微秒

▶写入0x30重新设​​置8位模式

▶等待>100微秒

▶写入0x30再次设置8位模式

▶等待直到忙标志清除

▶写入0x3C以设置2条线和5×8点阵字体

▶等待直到忙标志清除

▶写入0x08以关闭显示屏

▶等待直到忙标志清除

▶写入0x01以清除显示屏

▶等待>1530微秒

▶写入0x06以设置输入模式、在每个字符后增加光标

▶等待直到忙标志清除

▶写入0x0C以打开没有光标的显示屏

然后，为了写入文本到LCD，微控制器可以发送ASCII字符序列。它也可以发送指令为0x01以清除显示器或0x02以返回在左上方的起始位置。

图8.49 Crystalfontz CFAH2002A-TMI 20×2 字符LCD

图8.50 并行LCD接口

例8.26 LCD控制

写一个程序把“I love LCDs”写入到字符显示器中。

解：以下程序把“I love LCDs”写入到显示器中。它需要使用例8.21中的delaymicros函数。

{代码}

**VGA显示器**

更灵活的显示选项是驱动计算机显示器。视频图形阵列（Video Graphics Array，VGA）显示器标准于1987年提出，最先用于IBM PS/2计算机，在阴极射线管（cathode ray tube，CRT）具有640×480像素分辨率，使用一个15针连接器通过模拟电压传输色彩信息。现代液晶显示器具有更高的分辨率，但保持与VGA标准的向后兼容。

在阴极射线管中，电子枪从左到右扫描屏幕发射出荧光材料来显示图像。彩色阴极射线管使用三种不同的荧光体，红、绿、蓝，和三个电子波束。每个波束的强度决定像素中每种颜色的强度。在每条扫描线的末端，电子枪必须关闭一段时间，称为水平消隐间隔（ orizontal blanking interval），从而返回到下一条扫描线的开头。所有扫描线完成后，电子枪必须再次关闭一段时间，称为垂直消隐间隔（vertical blanking interval）从而返回到左上角。该过程每秒重复大约60-75次，得到一个稳定的视觉图像。

640×480像素的VGA显示器刷新频率为59.94 Hz，像素时钟为25.175MHz，因此每个像素宽度为39.72ns。全屏幕可以被看作是525条水平扫描线，每条扫描线800像素，但只有480条扫描线和每条扫描线640像素实际用于传达图像，而其余的都是黑色。扫描线始于后廊（back porch），屏幕左边缘的空白区域。它包含640个像素，紧接着是空白左廊（front porch）在屏幕的右边缘，然后是一个水平同步（hsync）脉冲将电子枪快速移动回左边缘。图8.51（a）显示了上述每个扫描线部分的时序，从有效像素开始。整个扫描线是31.778微秒长。在垂直方向上，屏幕始于在顶部的后廊，紧跟着480条有效扫描线，随后是在底部的前廊和一个垂直同步（vsync）脉冲以返回到顶部开始下一帧。新帧绘制频率为每秒60次。图8.51（b）展示了垂直方向时序；注意，此时时间单位是扫描线，而不是像素时钟。更高的分辨率使用更快的像素时钟，388×1536 @ 85 Hz时高达2048 MHz。例如，1024×768 @ 60Hz可以使用65MHz像素时钟来实现。水平时序包括16个时钟的前廊，96个时钟的同步脉冲，以及48个时钟的后廊。垂直时序包括11条扫描线的前廊，2条扫描线的垂直同步脉冲和32条扫描线的后廊。

Figure 8.51 VGA 时序:(a) 水平, (b) 垂直

Color：颜色

Active：有效

pixels：像素

Scan line：扫描线

pixel clock：像素时钟

Frame：帧

Front porch：前廊

Back porch：后廊

图8.52显示了视频源的母连接器引脚图。像素信息使用三个模拟电压传递，分别代表红色、绿色和蓝色。每个电压范围是0-0.7V，电压越大则表示亮度越大。该电压在前廊和后廊时应为0。该电缆还提供I2C串行链路以配置显示器。

图8.52 VGA连接器引脚图

必须实时高速生成的视频信号，在微控制器上实现有困难，但在FPGA上却容易实现。简单的黑白显示可以通过使用0或0.7V来驱动所有三色引脚而产生，这要采用连接到数字输出引脚的分压器。另一方面，彩色显示器使用了具有三个独立D/A转换器的视频DAC（video DAC）以独立地驱动三色引脚。图8.53显示了一个FPGA通过ADV7125三重8位视频DAC来驱动VGA显示器。DAC接收来自FPGA的8位R、G和B信号。它还接收SYNC\_b信号，该信号当HSYNC或VSYNC有效时被驱动为低电平有效。视频DAC产生三个输出电流以驱动红、绿、蓝模拟线路，这通常是平行直连视频DAC和显示器的75Ω传输线。RSET电阻设定输出电流，以达到满量程色彩范围。时钟速率取决于分辨率和刷新率；使用快速级ADV7125JSTZ330模型DAC时可能高达330MHz。

图8.53 FPGA通过视频DAC驱动VGA线

例8.27 VGA显示器

使用图8.53中电路编写HDL代码以在VGA显示器上显示文字和一个绿色框。

解：该代码假定40MHz的系统时钟频率，并在FPGA上使用一个锁相环（phaselocked loop，PLL）生成25.175 MHz的VGA时钟。PLL不同FPGA中的配置不同；对于Cyclone III，频率通过Altera的宏功能向导指定。或者，VGA时钟可以直接由信号发生器提供。

VGA控制器通过屏幕上的列和行进行计数，在适当时候生成hsync和vsync信号。此外，它还生成blank\_b信号，当坐标在640×480有效区域外时，此信号被设置为低电平以绘制黑色。

视频发生器基于当前（x，y）像素位置产生红、绿、蓝颜色数值。（0，0）表示左上角。视频发生器在屏幕上绘制一组字符，以及一个绿色矩形。字符生成器绘制一个8×8像素字符，屏幕尺寸为80×60字符。它从ROM上查找字符，在ROM上采用二进制编码为8行6列字符。另外两列都是空白。SystemVerilog代码中的位顺序是反转的，因为ROM文件的最左列是最高有效位，而它应绘制在最低有效x位置。

图8.54显示了正在运行此程序的VGA显示器照片。字母行交替为红色和蓝色，绿色框覆盖一部分图片。

图8.54 VGA输出

{代码}

**蓝牙无线通信**

现在可用于无线通信的标准有许多，包括Wi-Fi、ZigBee和蓝牙。这些标准是详尽复杂，并需要复杂的集成电路，但是不断增加的模块种类使复杂性被抽象掉，为用户提供无线通信的简单界面。BlueSMiRF是这些模块之一，一个易于使用的蓝牙无线接口，可用来取代串行电缆。

蓝牙是由Ericsson在1994年开发出来的无线通信标准，用于低功率、中速、5-100米距离的通信，这取决于发射器功率级别。它通常用于连接手机听筒或计算机键盘。不同于红外线通信链路，它不要求设备之间进行直接视线连接。

蓝牙工作在2.4 GHz的工业-科学-医疗（industrial-scientificmedical，ISM）频段。它从2402MHz开始，采用1 MHz间隔定义了79条无线信道。它在这些信道之间以伪随机模型跳转更换，以避免与在相同频段工作的其它设备如无线路由器之间的连续干扰。如表8.10所示，蓝牙发射器分成三类功率水平，显示了传输范围和功耗。在基本速率模式下，它采用高斯频移键控（frequency shift keying，FSK）工作在1Mbit/s。在普通FSK中，每个位通过发送fc±fd的频率来传递，其中fc是信道中心频率，fd是偏移量，至少为115kHz。比特位间的频率突然转变占用额外的带宽。在高斯FSK中，对频率变化进行平滑处理，以更好地利用频谱。图8.55显示2402 MHz信道中传输0和1序列的频率，分别使用FSK和GFSK。

表8.10 蓝牙类别

类别

Transmitter Power：发送器功率

Range：范围

图8.55 FSK和GFSK波形

BlueSMiRF Silver模块，如图8.56所示，在一张带有串行接口的小卡片上包含2类蓝牙无线、调制解调器和接口电路。它与其它蓝牙设备如连接到PC的蓝牙USB适配器进行通信。因此，它可以在PIC32和PC之间提供一个无线串行链路，与图8.43的链路类似但没有电缆线。图8.57展示了这种链路的示意图。BlueSMiRF的TX引脚连接到PIC32的RX引脚，反之亦然。RTS和CTS引脚连接，使得BlueSMiRF能自己完成握手。

BlueSMiRF默认为115.2K波特率，8个数据位、1个停止位、无奇偶校验或流控制。它工作在3.3 V数字逻辑电平，所以RS-232收发器不需要与另一个3.3 V设备连接。

要使用该接口，就将USB蓝牙适配器插入电脑。启动PIC32和BlueSMiRF。BlueSMiRF的红色STAT指示灯将闪烁以表明它正在等待建立连接。打开PC系统中的蓝牙图标，使用添加蓝牙设备向导来配对适配器和BlueSMiRF。BlueSMiRF的默认配对码为1234。留意到哪个COM端口分配给适配器。然后，就像使用串行电缆一样进行通信。需要注意的是适配器通常工作在9600波特，PuTTY必须相应地进行配置。

图8.56 BlueSMiRF模块USB适配器

图8.57 蓝牙PIC32到PC链路

Wireless link：无线链路

USB bluetooth dongle：USB蓝牙适配器

USB port：USB端口

**电动机控制**

微控制器的另一个主要应用是驱动致动器，例如电动机。本小节描述三种类型电动机：直流电动机（DC motor），伺服电动机（servo motor）和步进电动机（stepper motor）。直流电动机需要高驱动电流，所以强大的驱动如H桥（H-bridge）必须连接微控制器和电动机。它们还需要一个单独的轴角编码器（shaft encoder），如果用户想要知道电机的当前位置。伺服电动机接收一个脉冲宽度调制信号以在有限的角度范围内指定位置。它们很容易连接，但不那么强大，也不适合连续旋转。步进电动机接收脉冲序列，每个脉冲信号将电动机转动一固定角度，称为一步。它们更昂贵，也需要一个H桥来驱动大电流，但位置可以精确地控制。

电动机引入大量电流，并可能在电源供应处引入毛刺而干扰数字逻辑。减轻这个问题的一种方法是对电动机和数字逻辑使用不同电源或电池。

**直流电动机**

图8.58展示了一个带刷直流电动机的操作。电动机是双终端装置。它包含永久固定磁铁称为定子和连接在轴上的旋转电磁铁称为转子（rotor）或电枢（armature）。转子的前端连接到一个裂开的金属环被称为换向器（commutator）。连接到电源接头（输入终端）的金属刷摩擦换向器，为转子电磁体提供电流。这在转子引入磁场使其旋转对准定子磁场。一旦转子接近对准定子，电刷摩擦换向器的反侧，反转电流和磁场，使其继续无限旋转下去。

图8.58 直流电动机

power lugs:电源接头

shaft:轴

stator north pole:定子北极

stator south pole:定子南极

rotor electromagnet:转子电磁铁

commutator:换向器

brushes:刷

直流电动机倾向于用非常低的扭矩旋转在每分钟几千转（rotations per minute，RPM）的速度。大多数系统添加齿轮组来降低速度到较合理水平，同时增加扭矩。寻找能紧密配合你的电动机的齿轮组。Pittman生产种类繁多的高品质直流电动机及配件，而价格低廉的玩具电动机很受发烧友欢迎。

直流电动机需要大量的电流和电压来提供显著的负载功率。如果电动机能双向旋转，电流应是可逆的。大多数微控制器不能产生足够的电流直接驱动直流电动机。相反，他们使用一个H桥，其概念上包含四个电控制开关，如图8.59（a）所示。如果开关A和D闭合，电流由左到右流过电机，并向一个方向旋转。如果B和C闭合，电流从右到左流过电机，并向另一个方向旋转。如果A和C或B和D闭合，电机两端电压强制为0，使电机主动制动。如果没有一个开关闭合，电机会自然停止。H桥的开关是功率晶体管。H桥还包含一些数字逻辑来方便地控制开关。当电动机的电流突然改变，电机电磁体的电感将引起一个大电压，可能超过电源大小而损坏功率晶体管。

因此，许多H桥还具有平行于开关的保护二极管，如图8.59（b）所示。如果感应冲击驱动电机的某个终端高于Vmotor或低于地线，二极管将转为ON以将电压钳位在一个安全水平。H桥可以消耗大量功率，因而可能需要散热器以保持冷却。

图8.59 H桥

例8.28自主小车

设计由PIC32控制两个驱动电机的机器人汽车系统。写一个函数库来初始化电机驱动器，使车向前向后行驶、左转或右转、停止。使用PWM来控制电机速度。

解：图8.60展示了一对受控于PIC32的直流电动机，采用Texas Instruments SN754410双H桥。H桥需要5 V逻辑电源VCC1和4.5-36V 电机电源VCC2；它的VIH=2V，因此与PIC32的3.3 V I/O兼容。它可以为每个电机提供高达1A 的电流。表8.11描述了每个H桥的输入如何控制电机。微控制器结合PWM信号驱动使能信号来控制电动机速度。它推动了其他四个引脚以控制每个电机的方向。

将PWM配置在约781Hz频率工作，其占空比范围从0至100％。

图8.60电机控制与双H桥

H-Bridge:H桥

Left:左

Right:右

表8.11 H桥控制

Motor:电动机

Coast:滑行

Brake:制动

Reverse:反向

Forward:正向

Brake:制动

{代码}

在前面的例子中，没有办法测量每个电机的位置。两个电机不可能精确匹配，所以一个可能转得比另一个稍快一些，当然会使机器人小车逐渐转向。为了解决这个问题，有些系统添加了轴角编码器。图8.61（a）展示一个简单的轴角编码器，由安装在电动机轴上的分槽圆盘组成。一个LED被放置在一侧上而一个光传感器被置于另一侧。每次槽间隙旋转经过LED时，轴角编码器生成一个脉冲。微控制器通过计算这些脉冲数目以测量转轴所转过的总角度。通过使用两对LED/传感器对，相隔半个槽宽度放置，改进后的轴角编码器可以产生图8.61（b）所示的正交输出，这能看出轴转动方向以及转过的角度。有时，轴角编码器增加一个孔以表示转轴处于索引位置。

图8.61 轴角编码器（a）圆盘，（b）正交输出

**伺服电动机**

伺服电动机是直流电动机集成了齿轮组、轴角编码器和一些控制逻辑，因而更容易使用。它们旋转角度有限，通常为180°。图8.62展示了除盖后的伺服电动机。伺服电机使用3针接口：电源（通常为5V）、接地以及控制输入端。控制输入​​通常是50Hz脉冲宽度调制信号。伺服电机的控制逻辑驱动转轴转动至由控制输入的占空比所确定的位置。伺服电机的轴角编码器是典型旋转式电位器，依赖于轴位置产生电压。

在一个180度旋转的典型伺服电机中，0.5ms脉冲宽度驱动轴转至0°，1.5ms脉冲宽度驱动轴转至90°，而2.5ms脉冲宽度驱动轴转至180°。例如，图8.63展示了1.5ms脉冲宽度的控制信号。过度驱动伺服电机可能导致其撞击机械制动和损坏。伺服的电源来自电源引脚，而不是控制引脚，因此控制信号可以直接连接到微控制器，而不使用H桥。伺服电机通常用于远程控制模型飞机和小机器人，因为它们小、轻和方便。而找到具有充足数据手册的电机可能是困难的。带红线的中心引脚通常是电源，而黑色或棕色线则通常接地。

图8.62 SG90伺服电动机

DC Motor:直流电动机

Gears:齿轮

3 wire interface:3线接口

shaft encoder:轴角编码器

图8.63 伺服控制波形

pulse width: 脉冲宽度

period: 周期

例8.29 伺服电动机

设计一个系统使用PIC32微控制器驱动一个伺服电动机转至所需角度。

解：图8.64展示了连接到SG90伺服电动机的示意图。伺服电机使用4.0-7.2 V电源工作。只需使用一根连线来传递PWM信号，可运作在5或3.3V的逻辑电平。代码使用输出比较模块1来配置PWM生成，并为所需角度设置相应占空比。

图8.64 伺服电动机控制

control: 控制

{代码}

另外，也可以将普通伺服电动机转换成连续伺服电动机（continuous rotation servo）：仔细拆卸它，去除机械制动，并用固定分压器代替电位计。许多网站展示了改造特定伺服电机的详细指导。PWM将控制速度而不是位置，1.5ms表示停止，2.5ms表示全速前进，而0.5ms则表示全速后退。连续旋转伺服电机可能会比结合H桥和齿轮组的简单直流电机更方便和更便宜。

**步进电动机**

步进电动机以离散步伐前进，因为脉冲施加到交替的输入上。步长通常是几度，允许精确定位和连续旋转。小型步进电机一般都有两组线圈，称为相位（phase），有双极性和单极性两种方式。双极电动机功能更强大、相对更便宜，但需要一个H桥驱动器，而单极电机可使用晶体管作为开关来驱动。本节的重点是更高效的双极步进电机。

图8.65（a）展示一个简化的两相双极电机，步长90°。转子是有一个北极和一个南极的永久磁体。定子是包含两对线圈的电磁铁，这两对线圈构成两个相位。两相双极电机因而有四个终端。8.65（b）显示了步进电机符号，将两个线圈建模成电感器。

图8.65（a）简化的双极步进电动机；（b）步进电动机符号

Stator：定子

rotor：转子

图8.66展示了用于两相双极电机的三个常见驱动序列。图8.66（a）阐述了波驱动（wave drive），线圈通电顺序为AB - CD - BA - DC。需要注意的是BA表示绕组AB用反向电流通电；这是双极性（bipolar）名称的由来。转子每一步转动90度。图8.66（b）表示两相驱动，以如下序列驱动：（AB，CD） - （BA，CD） - （BA，DC） - （AB，DC）。（AB，CD）表示两个线圈AB和CD同时通电。转子同样每一步转动90度，但将自己对准两极位置之间。这能得到最高转矩运行，因为两个线圈同时通电。图8.66（c）演示了半步驱动，以如下序列驱动：（AB，CD） - CD - （BA，CD） - BA -（BA，DC） - DC - （AB，DC） - AB。转子每半步转动45度。序列推进速度决定了电机速度。要反转电机方向，则将相同的驱动序列倒序执行。

图8.66 双极电动机驱动

wave drive:波驱动

two-phase on drive:两相驱动

half-step drive:半步驱动

在真实的电动机里，转子具有许多磁极以使步伐之间的角度小得多。例如，图8.67展示的AIRPAX LB82773-M1双极步进电机，步长为7.5度。电机运行在5 V，每个线圈电流为0.8 A。

图8.67 AIRPAX LB82773-M1双极步进电动机

电动机的转矩与线圈电流成正比。电流由施加的电压以及线圈电感L和电阻线圈R来确定。最简单的操作模式称为直流电压驱动（direct voltage drive）或L/R驱动（L/R drive），其中电压V直接施加到线圈。电流逐步上升到I = V/R，上升时间常数由L/R设定，如图8.68（a）所示。这非常适用于慢速操作。然而，在速度较高的情况下，电流没有足够时间逐步上升到满级，如图8.68（b）所示，转矩随之下降。

更有效的驱动步进电机的方法是通过脉冲宽度调制较高电压。高电压使电流更快上升到满级，然后关闭PWM以避免电机过载。然后电压被调制或斩波（chop）以使电流保持在邻近所需水​​平。这就是所谓的斩波恒流驱动（chopper constant current drive），如图8.68（c）所示。该控制器使用一个小电阻串联电动机通过测量电压下降值以感测施加电流值，当电流达到所需水平则施加使能信号到H桥以关闭驱动。原则上，微控制器可以产生正确的波形，但使用步进电机控制器更容易。ST Microelectronics的L297控制器是一个方便的选择，尤其再加上带有电流检测引脚和2A峰值功率能力的L298双H桥。不幸的是，L298没有可用的DIP封装，因此难以安装到试验电路板上。ST Microelectronics的应用笔记AN460和AN470对步进电机设计人员来说是有价值的参考。

图8.68 双极步进电动机直接驱动电流：（a）缓慢转动，（b）快速旋转，（c）结合斩波驱动的快速旋转

current:电流

例8.30双极步进电动机直接波驱动

设计一个系统，其中一个PIC32微控制器采用直接驱动方式以指定速度和方向驱动AIRPAX双极步进电机。

解：图8.69显示了一个双极步进电机被一个由PIC32控制的H桥直接驱动。

图8.69 由H桥直接驱动的双极步进电动机

H-Bridge：H桥

Bipolar stepper motor：双极步进电动机

From PIC：来自PIC

spinstepper函数按照直接驱动序列初始化序列阵列中应用于RD[4：0]的模式。应用了序列中的下一个模式后，等待足够时间以每分钟所需的转数（RPM）旋转。使用20 MHz时钟、7.5°步长、16位计时器和256：1预分频器，可用速度范围是2-230rpm，其最小值受限于计时器分辨率，而最大值受限于LB82773-M1电机的动力。

{代码}

#### 8.7 PC I/O系统

个人计算机（PC）使用各种I/O协议，包括存储器、磁盘、联网、内扩展卡和外部设备。这些I/O标准已经发展到可提供非常高的性能，很容易让用户添加设备。这些属性以I/O协议的复杂度为代价。本节探索在PC上使用的主要I/O标准和考察将PC连接到定制的数字逻辑或其他外部硬件的选项。

图8.70显示了包含Core i5或i7处理器的PC主板。该处理器被打包在一个栅格阵列（land grid array）里，内含1156镀金垫以给处理器提供电源和接地，并连接处理器到存储器和I/O设备。主板包含DRAM内存模块插槽、各种I/O设备接口、以及电源连接器、稳压器和电容。一对DRAM模块连接在一个DDR3接口上。外围设备，如键盘或摄像头，通过USB连接。高性能扩展卡，如显卡，通过PCI Express x16插槽连接，而低性能卡则可以使用PCI Express x1插槽或早期的PCI插槽。PC使用以太网插孔连接到网络。硬盘连接到一个SATA端口。本节剩余内容将讲述这些I/O标准的操作。

图8.70 Gigabyte GA-H55MS2V主板

CPU socket 1156：CPU插座1156

PS/2 keyboard and mouse connectors：PS/2键盘和鼠标连接器

Graphics connectors：显卡接口

USB connectors：USB连接器

Ethernet jack：以太网插孔

Audio jack：音频插孔

PCI express x16 slot：PCI Express x16插槽

PCI slots：PCI插槽

PCI express x1 slot：PCI Express x1插槽

DDR3 memory sockets：DDR3内存插槽

Power supply connector：电源连接器

SATA connectors：SATA连接器

PC I/O标准的一个重大进展是开发了高速串行链路。至今，大多数I/O是围绕并行链路构建的，它由一个宽数据总线和一个时钟信号组成。随着数据速率的增加，总线内线路间的延迟差别限制了总线运行速度。此外，连接到多个设备的总线遭受传输线路问题，如信号反射和不同负载的不同飞越时间。噪声也会损坏数据。点到点串行链路消除许多这些问题。数据通常使用电线差动对发送，因而影响配对的两根线路的外部噪音是不重要的。传输线路很容易正确地终止，因此信号反射小（见A.8节传输线路）。没有发送明确时钟；相反，时钟由接收器通过观察数据传输时序来进行恢复。高速串行链路设计是一个专门课题，良好的接口可以在铜导线上运行超过10 Gb/s，沿光纤传输甚至更快。

##### 8.7.1 USB

直到九十年代中期，将外设添加到PC需要一些技术知识。添加扩展卡需要打开机箱，设置跳线到正确的位置，并且手动安装设备驱动程序。添加一个RS-232设备需要选择合适的电缆并正确配置波特率、数据、奇偶校验位和停止位。通用串行总线（Universal Serial Bus，USB），由Intel、IBM、Microsoft等等一同开发，通过将电缆和软件配置过程标准化而大大简化了添加外围设备流程。现在每年售出数十亿个USB外设。

USB 1.0发布于1996年，采用四线简单连接线：5V、GND和一对电线差动对以携带数据。电缆是不可能反向或倒置插入的。它的工作频率高达12 Mb/s。一个设备可以从USB端口引出高达500 mA电流，所以键盘、鼠标和其他外围设备可以从端口获得所需电源供应，而不必通过电池或独立电源线。

USB 2.0，发布于2000年，通过大幅提高差动导线运行速度而使自身速度升级到480 MB/s。有了更快的​​链路，USB实际用于连接摄像头和外置硬盘。带USB接口的闪存也取代软盘作为在计算机之间传输文件的手段。

USB 3.0，发布于2008年，进一步提高速度到5 Gb/s。它使用相同形状的连接器，但电缆具有多根导线使其在非常高的速度下运作。它更适合于连接高性能硬盘。同时，USB增加了一个电池充电规范，以增强由端口提供的电源，从而加快移动设备的充电。

对于用户而言的简单性，以更复杂的硬件和软件实现为代价。从头开始建立一个USB接口是一个大工程。即使写一个简单的设备驱动程序也是较为复杂的。PIC32带有一个内置的USB控制器。然而，Microchip连接鼠标和PIC32的设备驱动程序（见microchip.com）超过500行代码而且超出本章范围。

##### 8.7.2 PCI和PCI Express

外设组件互连（Peripheral Component Interconnect，PCI）总线是由Intel开发的扩展总线标准，在1994年左右普及。这是用来增加扩展卡如额外串行或USB端口、网络接口、声卡、调制解调器、磁盘控制器、或视频卡。32位并行总线运行频率为33 MHz，具有133 MB/s的带宽。

PCI扩展卡的需求已经稳步下降。更多现代标准接口，如以太网和SATA被集成到主板上。许多曾经需要扩展卡的设备现在可以通过快速USB 2.0或3.0链路连接。而现在视频卡则需要远超过PCI可以提供的带宽。

现代的主板往往还是有少数PCI插槽，但是高速设备，如视频卡，现在通过PCI Express（PCIe）连接。PCI Express插槽提供了一个或多个高速串行链路通道。在PCIe 3.0中，每个通道的运行速度高达8 Gb/s。大多数主板提供x16插槽，16通道为大流量数据设备，如视频卡，提供总共16 GB/s的带宽。

##### 8.7.3 DDR3内存

DRAM通过并行总线连接到微处理器。在2012年，当前标准是DDR3，第三代双倍数据速率存储器总线工作在1.5 V电压。典型主板现在配备了2条DDR3通道，使他们可以同时访问两组内存模块。

图8.71显示了一个4 GB的DDR3双列直插内存模块（DIMM）。该模块每边具有120个触点，总共240个连接，包括一个64位数据总线、一个16位时分多路复用地址总线、控制信号、以及众多电源和接地引脚。在2012年，典型的DIMM装载1-16 GB的DRAM。内存容量每2-3年增加一倍左右。

DRAM当前工作在100-266 MHz时钟速率。DDR3以四倍于DRAM的时钟速率操作存储器总线。而且，它在时钟上升沿和下降沿都传送数据。因此，它在每个存储器时钟发送8个字的数据。如果64位/字，这就对应6.4-17 GB/s 带宽。例如，DDR3-1600使用一个200MHz内存时钟和800MHz I/O时钟来发送16亿字/秒，或12800 MB/s。因此，这模块也称为PC3-12800。不幸的是，DRAM的反应时间依然很长，从一个读请求到数据的第一个字到来有大约50ns的滞后。

图8.71 DDR3内存模块

##### 8.7.4 网络

计算机通过网络接口连接到Internet，这个接口运行传输控制协议和网际协议（Transmission Control Protocol and Internet Protocol，TCP/IP）。物理连接可以是以太网电缆或无线Wi-Fi链路。

以太网是由IEEE 802.3标准定义的。它于1974年在Xerox Palo Alto研究中心（PARC）研发。它最初运作在10 Mb/s（所谓的10兆以太网），但是现在通常发现运行在100兆（MB/s）和1千兆（GB/s），通常使用含有四对双绞线的5类电缆。在光纤上运行的10千兆以太网在服务器和其它高性能计算中日益普及，而100千兆以太网正在形成。

无线网络（Wi-Fi）是IEEE 802.11无线网络标准的普遍称呼。它工作在2.4GHz和5GHz非授权无线频段，意味着该用户不需要一个无线电操作员执照就可以在这些频段以低功率发送数据。表8.12总结了三代无线网络的能力；新兴的802.11ac标准承诺使用超过1 Gb/s的传输速率推送无线数据。日益增加的性能来自推进调制和信号处理、多重天线、和更广的信号带宽。

表8.12 802.11 Wi-Fi协议

Protocol：协议

Release：发布年份

Frequency Band：频段

Data Rate：数据速率

Range：范围

##### 8.7.5 SATA

内置硬盘需要一个连接到PC机的快速接口。1986年，Western Digital推出了集成驱动电子（Integrated Drive Electronics，IDE）接口，后来演变成AT附件（AT Attachment，ATA）标准。该标准使用一个笨重的40或80线带状电缆，最大长度为18”，以16-133 MB/s速率来发送数据。

ATA已被串行ATA（Serial ATA，SATA）所取代，它使用高速串行链路通过更方便的7芯电缆以1.5、3或6 Gb/s速率运行，如图8.72所示。2012年度最快的固态硬盘可接近500 MB/s带宽，充分利用了SATA的优势。

一个相关标准是串行连接SCSI（Serial Attached SCSI，SAS），它是并行SCSI（Small Computer System Interface）接口的演化。SAS提供了可媲美SATA的性能，并支持更长的电缆；它在服务器计算机里很常见。

图8.72 SATA电缆

##### 8.7.6 连接到PC

到目前为止所描述的所有PC I/O标准都为了高性能和易于连接而优化，但很难在硬件中实现。工程师和科学家们经常需要某种方法将PC连接到外部电路，诸如传感器、致动器、微控制器或FPGA等。8.6.3.2节所述的串行连接足以负荷连接到微控制器UART上的低速连接。本节将介绍另外两种方法：数据采集系统，和USB链路。

**数据采集系统**

数据采集系统（Data Acquisition System，DAQ）使用多个模拟和/或数字I/O通道将计算机与现实世界连接起来。DAQ现在通常作为USB设备使用，这使它们易于安装。National Instruments（NI）是一家领先的DAQ制造商。

高性能DAQ的价格往往会高达数千美元，主要是因为市场小而限制了竞争。幸运的是，2012年，NI以200美元的学生折扣价出售便于使用的myDAQ系统，包括了LabVIEW软件。图8.73展示了myDAQ。它包含了输入和输出两个模拟通道，200 ksamples/s采样频率、16位分辨率和±10V动态范围。这些通道可被配置成像示波器和信号发生器那样工作。它还有8条数字输入和输出线，可与3.3和5 V系统兼容。此外，它产生+5、+15和-15 V电源输出，包括能够测量电压、电流和电阻的数字万用表。因此，myDAQ可取代整套试验和测量设备，它能同时自动记录数据。

大多数NI DAQ使用LabVIEW控制，它是NI的图形语言，用于设计测量和控制系统。一些DAQ也可以在LabWindows环境使用​​C程序控制，在Measurement Studio环境使用Microsoft .NET应用程序控制，或者在Data Acquisition Toolbox使用Matlab控制。

图8.73 NI myDAQ

**USB链路**

越来越多种类产品通过USB为PC和外部硬件之间提供简单、价格低廉的数字链路。这些产品含有预开发的驱动程序和库，允许用户轻松地在PC上编写程序从FPGA或微控制器读写数据。

FTDI是此类系统的领先供应商。例如，图8.74所示的FTDI C232HM-DDHSL USB到多协议同步串行引擎（MPSSE）电缆，电缆一端是USB插孔，而另一端是一个工作频率高达30 Mb/s的SPI接口，以及3.3 V电源和四个通用I/O引脚。图8.75展示了使用电缆将PC连接到FPGA的例子。电缆可以选择提供3.3 V电源给FPGA。这三个SPI引脚像例8.19那样连接到FPGA从设备上。该图还显示了用于驱动LED的一个GPIO引脚。

PC要求安装D2XX动态链接库驱动程序。然后，你就可以使用该库编写一个C程序在电缆上发送数据。

如果需要更快的连接，图8.76所示的FTDI UM232H模块连接PC的USB端口和一个8位同步并行接口，它的工作频率高达40 MB/s。

图8.74 FTDI USB到MPSSE电缆

图8.75 C232HM-DDHSL USB到MPSESE接口（从PC到FPGA）

From PC USB port：来自PC USB端口

图8.76 FTDI UM232H模块

#### 8.8 现实世界透视：x86内存和I/O系统\*

处理器速度越来越快，需要更复杂的存储器层次结构以保持提供稳定的数据流和指令流。本节介绍了x86处理器的存储器系统，以说明这个发展过程。7.9小节中的处理器照片突出显示了片上高速缓存。x86还使用了一个与大部分常见内存映射I/O不同的直接I/O编程方式。

##### 8.8.1 x86高速缓存系统

1985年开始生产的80386以16MHZ的频率工作。它没有高速缓存，直接访问主存以获取所有指令和数据。由于存储器速度的差异，处理器可能立即得到响应，也可能会暂停一到两个周期等待内存响应。这些等待的周期称为等待状态（wait states），将会增加处理器的CPI。从那时候开始，微处理器的频率每年都提高至少25%，然而内存的延迟并没有降低。从处理器向主存发出地址到主存返回数据的延迟目前已经超过100个时钟周期。因此，低缺失率的高速缓存是提高性能所必需的。表8.13总结了英特尔x86处理器高速缓存系统的发展。

表8.13 Intel x86微处理器存储器系统的发展

Processor：微处理器

Year：年代

Frequency：主频

Level 1 Data Cache: 第一级数据高速缓存

Level 1 Instruction Cache: 第一级指令高速缓存

Level 2 Cache: 第二级高速缓存

8KB unified: 8KB统一高速缓存

32KB/core: 每核32KB

12KB op trace cache: 12KB微操作trace cache

None on chip: 不在片上

256KB—1MB on MCM：256KB—1MB（多芯片模块）

256—512KB on cartridge：256—512 KB（盒装）

256—512KB on chip：片上256—512 KB

256KB—2MB on chip: 片上256KB—2MB

1—2MB on chip：片上1—2MB

2MB shared on chip: 片上2MB共享

256KB/core + 4—15 MB L3: 每核256KB + 4—15 MB第三级缓存

80486引入了一个统一的写直达高速缓存来保存指令和数据。大部分高性能的计算机系统使用速度大大高于主存的商业SRAM芯片在主板上提供了更大的二级高速缓存。

Pentium处理器引入了分离的指令和数据高速缓存，以防止同时请求指令和数据时所产生的竞争。高速缓存使用写回策略以减少与主存的通讯。同样地，在主板上提供了一个更大的第二级高速缓存（一般为256-512KB）。

P6系列处理器（Pentium Pro，Pentium II，和Pentium III）的设计可以支持更高的时钟频率。主板上的第二级高速缓存没有继续保留，而被移到与处理器更接近的位置，以改进其延迟和吞吐量。Pentium Pro的封装采用了多芯片模块（Multichip Module, MCM），其中包含了处理器芯片和第二级高速缓存芯片，如图8.77所示。和奔腾相似，它采用分离的8KB第一级指令和数据高速缓存。然而，这些高速缓存是非阻塞的（nonblocking）：即使特定访问在高速缓存中缺失而必须从主存获取数据时，乱序执行的处理器仍然可以继续后续的高速缓存访问。第二级高速缓存容量为256KB，512KB，或者1MB大小，并可以处理器相同的速度运行。不幸的是，对于大批量的制造而言，多芯片模块封装过于昂贵。因此Pentium II以包含处理器和第二级高速缓存的低成本盒装发售。第一级高速缓存容量增加了一倍，以弥补第二级高速缓存以处理器一半速度运行的不足。Pentium III在处理器的同一颗芯片上直接集成了一个全速第二层高速缓存。同一芯片上的高速缓存可以以更佳的延迟和吞吐量运行，它比片外的同等容量高速缓存更有效率。

Pentium 4处理器提供一个非阻塞的第一级数据高速缓存。在指令被译码为微操作后，它变为trace cache来存储指令，从而避免指令每次从高速缓取出后而进行重复译码的延迟。

Pentium M处理器的设计来源于Pentium III处理器。它大幅度增加第一级高速缓存容量到32KB，配置一个1到2MB的第二级高速缓存。Core Duo包含了两个改进的Pentium M处理器和一个共享的2MB片上高速缓存。共享的高速缓存可用于处理器间的通讯：一个处理器写入高速缓存，另一个处理器读出。

Nehalem（Core i3-i7）处理器设计添加了第三级缓存，由片上所有处理器内核共享，从而便利内核间的信息共享。每个内核拥有自己的64KB第一级缓存和256KB第二级缓存，共享4—8MB（或更多）第三级缓存。

图 8.77 PGA封装中的Pentium Pro 多芯片模块，左边为处理器，右边为256KB高速缓存（得到Intel公司许可）

##### 8.8.2 x86虚拟存储器

x86处理器在实模式或者保护模式下运行。实模式（real mode）向后兼容原始的8086。它只使用20位地址，内存容量限制为1MB，也不支持虚拟存储器。

80286中引入了保护模式（protected mode），80386将其被扩展到32位地址。它支持4KB页的虚拟存储器。它还提供内存保护，一个程序不能访问属于其他程序的页。因此，一个有漏洞或者恶意的程序就不能破坏或者影响其他程序了。所有现代的操作系统都使用保护模式。

32位的地址允许最多4GB的存储器空间。从Pentium Pro开始，处理器就已经使用物理地址扩展（physical address extension）技术扩充存储器容量到64GB。每一个进程使用32位地址，虚拟存储器系统将这些地址映射到更大的36位虚拟存储器空间。它对每个进程使用不同的页表，所以每一个进程可以它自己的4GB地址空间。

为了更优雅地避开存储器瓶颈问题，x86升级至x86-64，能提供64位虚拟地址和通用寄存器。目前，虚拟地址只使用了48位，提供了256TB的虚拟地址空间。当存储器扩展时，这个极限可能会扩大至全64位，提供16EB地址空间容量。

##### 8.8.3 x86可编程I/O

大部分体系结构使用8.5小节描述的内存映射I/O机制，其中程序以读和写内存位置的方式访问I/O设备。x86的可编程I/O采用特别的IN和OUT指令实现读和写I/O设备。x86定义216个I/O端口。IN指令从DX指定的端口读取1，2或者4字节到AL、AX或EAX寄存器。OUT指令与之相似，但是写入端口。

把外围设备连接到可编程I/O跟把它连接到内存映射系统很相似。当访问一个I/O端口时，处理器发送端口号（而非内存地址）到地址总线的最低16位。设备从数据总线读或者写数据。最大的不同在于处理器还产生一个信号。当=1时，表示处理器正在访问内存。当它为0时，进程正在访问一个I/O设备。地址译码器也将检查信号以产生合适的主存和I/O设备的使能信号。I/O设备同样可以向处理器发送中断来表示它们准备好通讯。

#### 8.9 总结

存储器系统的组织是决定计算机性能的主要因素。DRAM、SRAM和硬盘等不同的存储技术在容量，速度和消耗等三方面提供了不同的折衷。本章介绍了基于高速缓存和虚拟存储器的组织结构，它们使用存储器层次结构方法提供了接近理想的大容量、快速、廉价的存储器系统。主存一般用DRAM构成，其速度明显比处理器慢。高速缓存把常用数据保存在快速SRAM中以减少访问时间。虚拟存储器用硬盘存储暂时不需要在主存的数据以增加内存容量。高速缓存和虚拟存储器增加了计算机系统的复杂度和硬件，但是益处往往超过成本。所有的现代个人计算机都使用高速缓存和虚拟存储器。大部分处理器还使用内存接口与I/O设备通讯。这被称为内存映射I/O。程序使用读出和写入操作访问I/O设备。

#### 结语

这一章把我们带到了数字系统世界旅程的终点。我们希望本书不仅让读者学习到工程技术知识，也能你感受到美妙和令人神往的数字电路设计艺术。读者学习了如何使用原理图和硬件描述语言设计组合和时序逻辑，熟悉了多路选择器、ALU、存储器等较大的数字电路模块。计算机是最吸引人的数字系统应用之一。读者已经学习了如何用汇编语言对MIPS处理器编程和如何使用数字电路模块构造微处理器和存储器系统。读者可以发现了抽象、规范、层次化、模块化和规整化等原则贯穿了全书。通过这些技术原则，我们可以完成微处理器内部运行这个拼图难题。从移动电话到数字电视再到火星探测器和医学影像系统，我们的世界日益数字化。

试想像，在一个半世纪之前Charles Babbage在与魔鬼的交易中也有相似的经历？？。他只不过渴望以机械精度来计算数学用表。今天的数字系统是昨天的科幻小说。Dick Tracy（三十年代美国连环漫画人物）曾在电话里听说过iTunes吗？Jules Verne（十九世纪法国科幻作家）会发射全球定位卫星星座到太空吗？Hippocrates（古希腊物理学家和医学家）能用过高分辨率的脑部数字照片治疗疾病吗？但是同时，George Orwell噩梦中无所不在的政府监视正一天天地走向现实。而且罪犯声称可以用先进的膝上电脑开发核武器，其计算能力比冷战时期用于模拟炸弹实验的房间大小的超级电脑还强。微处理器的发展和进步仍在加速。未来10年的变化将会超过以往。读者现在已经有工具去设计和建造那些可以改造我们未来的新系统。更高的能力带来更多的责任。我们希望读者不仅仅为了娱乐或金钱来利用它，而应为了人类的利益。

#### 习题

习题8.1用简短的语言描述4个日常活动来说明时间局部性和空间局部性。每一种局部性说出两个例子，并加以解释。

习题8.2 用一个段话描述两个短的可以利用时间局部性和（或）空间局部性的计算机应用。说明原理。

习题8.3给出一个地址序列，针对此地址序列，容量为16个字，块大小为4个字直接映射的高速缓存性能将优于具有同样容量和块大小、采用LRU替换策略的全相联高速缓存。

习题8.4 重做习题8.3的例子，这时全相联高速缓存优于直接映射高速缓存。

习题8.5 在下述高速缓存参数中，增加其中一项而保持其他参数不变时，描述所产生的性能变化。

1. 块大小
2. 相联性
3. 高速缓存大小

习题8.6 两路组相联高速缓存的性能一定比同样容量和块大小的直接映射高速缓存好吗？请解释。

习题8.7 以下是关于高速缓存缺失率的说法。标志每句话是对还是错。简单解释你的原因，当说法是错的时候，给出一个反例。

1. 一个两路组相联高速缓存比有同样容量和块大小的直接映射高速缓存有更低的缺失率
2. 一个16KB大小的直接映射高速缓存比有同样块大小的8KB直接映射高速缓存有更低的缺失率
3. 块大小为32字节的指令高速缓存一般比一个8字节块大小，有同样的相联度和总容量的指令高速缓存有更低的缺失率

习题 8.8高速缓存有以下的参数：块大小b（以字为单位），组数S，路数N，地址位数A。

1. 以给出的参数表示，高速缓存容量C是多少？
2. 以给出的参数表示，需要多少位来存放标志？
3. 全相联高速缓存的容量是C，块大小是B，这是S和N是多少？
4. 直接映射高速缓存的容量为C，块大小为b，S为多少？

习题8.9 16字高速缓存的参数如习题8.8给出。考虑以下重复的lw地址序列（以16进制给出）：

40 44 48 4C 70 74 78 7C 80 84 88 8C 90 94 98 9C 0 4 8 C 10 14 18 1C 20

假设对相联高速缓存采用最近最少使用(LRU)替换策略，如果这个地址序列输入到以下高速缓存，忽略开始的影响（也就是强制缺失），计算有效缺失率。

1. 直接映射高速缓存，b=1字
2. 全相联高速缓存，b=1字
3. 两路组相联高速缓存，b=1字
4. 直接映射高速缓存，b=2字

习题8.10 重复习题8.9。考虑以下重复的lw地址序列（以16进制给出）和缓存配置。缓存容量仍为16字。

74 A0 78 38C AC 84 88 8C 7C 34 38 13C 388 18C

1. 直接映射高速缓存，b=1字
2. 全相联高速缓存，b=2字
3. 两路组相联高速缓存，b=2字
4. 直接映射高速缓存，b=4字

习题8.11假设以以下数据访问模式运行程序。这个模式仅只运行一次。

0x0 , 0x8 , 0x10 , 0x18 , 0x20 , 0x28

1. 如果使用直接映射高速缓存，容量为1KB，块大小为8个字节（2字），高速缓存内有多少组？
2. 针对（a）中的高速缓存，在此直接映射高速缓存中，针对给出的内存访问模式，缺失率是多少？
3. 针对给出的内存访问模式，以下哪一个方法最能减少缺失率？（高速缓存容量保持不变）。圈出你的答案。
4. 增加相联度为2
5. 增加块大小到16字节
6. （i）和（ii）都可以
7. （i）和（ii）都不可以

习题8.12 你正在为MIPS处理器设计一个指令高速缓存。它的总容量为4C=2c+2字节，采用N=2n路组相联（N8），块大小为b=2b’个字节（b8）。以已知参数的形式给出以下问题的答案。

1. 地址的哪些位用于选择块里的字？
2. 地址的哪些位用于选择高速缓存里的组？
3. 每一个标志有多少位？
4. 整个高速缓存中有多少标志位？

习题8.13 考虑以下参数的高速缓存：N（相联度）=2，b（块大小）=2个字，W（字大小）=32位，C（高速缓存大小）=32K字，A（地址大小）=32位。你只需要考虑字地址。

1. 给出地址中的标志、组、块偏移和字节偏移位，每个字段需要多少位。
2. 高速缓存中所有标志占多少位
3. 假设每个高速缓存块还有一位有效位(V)和一位脏位（D）。每一个高速缓存组（包括数据，标志和状态位）需要有多少位？
4. 使用图8.78中的模块和少量的2输入逻辑门电路设计高速缓存。高速缓存的设计必须包括标志存储、数据存储、地址比较、数据输出选择和任何你认为需要的部件。注意多路选择器和比较器块可以为任何大小（分别为n或者p位宽），但是SRAM块必须为16K×4位。请给出包含简明标志的电路模块图。只需设计实现读取功能的缓存。

图8.78 电路模块

习题8.14 你参加了一个热门的新互联网创业，用内嵌传呼机和网络浏览器开发腕表。它使用的嵌入式处理器中采用了图8.79所述的多级高速缓存方案。处理器包括一个小型的片上高速缓存和一个大型的片外第二级高速缓存（对，这个手表重3磅，但是你可以用它上网）。

假设处理器使用32位物理存储器地址但是只以字边界访问数据。表8.14给出了高速缓存参数。DRAM的访问时间为tm，大小为512MB。

图8.79 计算机系统

Processor chip: 处理器芯片

Level 1 Cache：第1级高速缓存

Level 2 Cache：第2级高速缓存

Main Memory：主存

表8.14 存储器特性

|  |  |  |
| --- | --- | --- |
| 特征 | 片上高速缓存 | 片外高速缓存 |
| 组织方式 | 四路组相联 | 直接映射 |
| 命中率 | A | B |
| 访问时间 | ta | tb |
| 块大小 | 16字节 | 16字节 |
| 块数目 | 512 | 256K |

1. 对于存储器中给定的字，在片上高速缓存和第二级高速缓存中总共有多少个可能的位置能找到它?
2. 片上高速缓存和第二层高速缓存的每个标志各需要多少位?
3. 给出内存平均访问时间的表达式。两级高速缓存按顺序连续访问。
4. 对于某一特定问题，测量发现片上高速缓存命中率为85%，第二层高速缓存命中率为90%。然而，当屏蔽片上高速缓存时，第二层高速缓存的命中率提高到98.5%。请解释这个现象。

习题8.15本章描述了最近最少使用（LRU）的多路相联高速缓存替换策略。还有一些不太常见的策略，如先入先出（FIFO）策略和随机策略。FIFO策略替换出存在最长时间的块，而不考虑它是否最近被访问过。随机策略则随机选择一个块作替换。

讨论这些替换策略的优缺点

描述一个FIFO会比LRU性能更好的访问模式

习题8.16 你正在设计的计算机存储体系结构为分离的指令和数据高速缓存，并使用图7.41中的MIPS多周期处理器，主频为1GHz。

1. 假设指令高速缓存已经完美（总是命中），但是数据高速缓存有5%的缺失率。在高速缓存缺失时，处理器暂停60ns访问主存，之后恢复正常操作。考虑高速缓存缺失的情况，平均内存访问时间为多少？
2. 考虑到非理想的存储器系统，平均每一条读出和写入指令需要多少时钟周期？
3. 考虑例子7.7中的基准测试程序，其中有25%的读出指令，10%的存储指令，11%的分支判断指令，2%的跳转指令和52%的R类指令。对非理想化的存储器系统考虑，这个基准测试程序的平均CPI是多少？
4. 现在假设指令高速缓存也是非理想化的，缺失率为7%，那么（c）部分基准测试程序的平均CPI是多少？把指令和数据高速缓存缺失都考虑在内。

习题8.17 参照以下参数重复习题8.16。

1. 假设指令高速缓存已经完美（总是命中），但是数据高速缓存有15%的缺失率。在高速缓存缺失时，处理器暂停200ns访问主存，之后恢复正常操作。考虑高速缓存缺失的情况，平均内存访问时间为多少？
2. 考虑到非理想的存储器系统，平均每一条读出和写入指令需要多少时钟周期？
3. 考虑例子7.7中的基准测试程序，其中有25%的读出指令，10%的存储指令，11%的分支判断指令，2%的跳转指令和52%的R类指令。对非理想化的存储器系统考虑，这个基准测试程序的平均CPI是多少？
4. 现在假设指令高速缓存也是非理想化的，缺失率为10%，那么（c）部分基准测试程序的平均CPI是多少？把指令和数据高速缓存缺失都考虑在内。

习题8.18 如果计算机使用64位虚地址，那么可以访问多少虚拟存储器。注意240字节=1terabyte，250字节=1petabyte，260=1exabyte。

习题8.19一个超级计算机的设计者花费1百万美元在DRAM上，同时花费同样多的钱在硬盘上以作为虚拟存储器。根据图8.4的价格，这台计算机可以拥有多大的物理存储器和虚拟存储器。需要多少位的物理和虚地址来访问这个存储器系统？

习题8.20考虑一个可以寻址全部232字节的虚拟存储器系统。你有无限的硬盘空间，但是只有有限的8MB物理存储器。假设虚页和物理页都是4KB大小。

1. 物理地址为多少位？
2. 系统中最大的虚页号是多少？
3. 系统中有多少物理页？
4. 虚页号和物理页号占多少位？
5. 假设你设计了一个把虚拟存储器映射到物理存储器上的直接映射方案。该映射使用虚页号的若干最低有效位来确定物理页号。每一个物理页上可以映射多少虚页？为什么这里的直接映射不是一个好的方案？
6. 明显地，需要一个比（d）部分更有灵活性和动态性的虚拟存储器地址到物理地址转换方案。假设你使用一个页表存储映射（从虚页号到物理页号的转换）。页表将需要包含多少个页表表项？
7. 除了物理页号以外，每个页表表项还要包括一些状态信息，例如有效位（V）和脏位（D）。每一个页表表项需要占用多少字节？（按照整数字节向上取整）
8. 给出页表的布局图。页表的大小是多少字节？

习题8.21考虑一个可以寻址全部250字节的虚拟存储器系统。你有无限的硬盘空间，但是只有有限的2GB物理存储器。假设虚页和物理页都是4KB大小。

1. 物理地址为多少位？
2. 系统中最大的虚页号是多少？
3. 系统中有多少物理页？
4. 虚页号和物理页号占多少位？
5. 页表将需要包含多少个页表表项？
6. 除了物理页号以外，每个页表表项还要包括一些状态信息，例如有效位（V）和脏位（D）。每一个页表表项需要占用多少字节？（按照整数字节向上取整）
7. 给出页表的布局图。页表的大小是多少字节？

习题8.22 你决定使用地址转换后备缓冲（TLB）为习题8.20的虚拟存储器系统加速。假设内存系统的参数如表8.15所示。TLB和高速缓存的缺失率表示所请求的内容找不到的概率。主存缺失率表示页面缺失的概率。

表8.15 存储器特性

Memory Unit：存储器部件

Access Time：访问时间

Miss Rate：缺失率

Cache: 高速缓存

Main memory: 主存

Hard drive: 硬盘

1. 在加上TLB前虚拟存储器系统的平均内存访问时间是多少？假设页表常驻在物理存储器，而不会保存在数据高速缓存内。
2. 如果TLB有64个表项， TLB大小为多少位？每个表项中包括以下字段：数据（物理页号），标志（虚页号）和有效位。给出各个字段所占用的位数。
3. 画出TLB的草图，清楚标志所有字段和尺寸。
4. 需要多大容量的SRAM来构(c)部分描述的TLB？以深度×宽度的形式给出答案。

习题8.23 你决定采用128个表项的地址转换后备缓冲（TLB）加速习题8.21中的虚拟存储系统。

1. TLB大小为多少位？每个表项中包括以下字段：数据（物理页号），标志（虚页号）和有效位。给出各个字段所占用的位数。
2. 画出TLB的草图，清楚标志所有字段和尺寸。
3. 需要多大容量的SRAM来构(b)部分描述的TLB？以深度×宽度的形式给出答案。

习题8.24 假设7.4小节描述的MIPS多周期处理器使用虚拟存储器系统。

1. 在多周期处理器原理图内画出TLB的位置。
2. 描述加入TLB候如何影响处理器的性能。

习题8.25 你正在设计的虚拟存储器系统使用一个以专用硬件（SRAM和相关逻辑）构成的单层页表。它支持25位虚地址，22位物理地址和216字节（64KB）的页。每一页表表项包含一个物理页号，可用位（V）和脏位（D）.

1. 页表的总大小为多少位？
2. 操作系统组建议将页大小从从64KB减少到16KB。但是你们小组的硬件工程师坚决反对，认为这将增加硬件开销。说出他们的理由。
3. 页表将与片上高速缓存一起集成在处理器芯片上。片上高速缓存只对物理地址（不对虚拟地址）操作。对于给定的内存访问，可以同时访问片上高速缓存合适的组和页表吗？简要解释同时访问高速缓存组和页表表项的必要条件。
4. 对于给定的内存访问，可以同时执行片上高速缓存标志比较和访问页表吗？简要解释原因。

习题 8.26 描述虚拟存储器系统可能影响应用写入的方案。必须讨论页大小和物理存储器大小如何影响程序的性能。

习题8.27假设你的个人电脑使用32位虚地址。

1. 每一个程序可以用最多多少的虚拟存储器空间？
2. PC硬盘的大小如何影响性能？
3. PC的物理存储器大小如何影响性能？

习题8.28 使用MIPS内存映射I/O与用户交互。每一次用户按下按钮，所选择的模式就在五个发光二极管（LED）上显示。假设输入按钮映射到地址0xFFFFFF10，LED映射到地址0xFFFFFF4。当按钮按下时，输出为1，否则为0。

1. 写出实现这个功能的MIPS代码。
2. 为这个内存映射I/O系统画出和图8.30相似的原理图。
3. 写出HDL代码以实现内存映射I/O系统的地址译码器。

习题8.29第三章中设计的有限状态机（FSM），也可以用软件实现。

1. 使用MIPS汇编语言实现图3.25的交通灯有限状态机。输入（TA和TB）映射到地址0xFFFFF000的第1位和第0位。两个3位输出（LA和LB）分别映射到地址0xFFFFF004的第0～2位和第3-5位。假设每一个灯的输出LA和LB采用独热编码：红100，黄010，绿001。
2. 为这个内存映射I/O系统画出和图8.30相似的原理图。
3. 写出HDL代码以实现内存映射I/O系统的地址译码器。

习题8.30 针对图3.30(a)中的FSM，重复习题8.29。地址0xFFFFF040的输入A和输出Y分别由存储映像为0和1。

#### 面试问题

以下的练习是曾经在面试中被问过的问题。

问题8.1 解释直接映射，组相联和全相联高速缓存的不同。对于每一种高速缓存类型，给出一个程序，其性能要好于其它两种高速缓存。

问题8.2 解释虚拟存储器系统是如何工作的。

问题8.3 解释使用虚拟存储器系统的优点和缺点。

问题8.4 解释存储器系统的虚页大小如何影响高速缓存的性能。

问题8.5 用于内存映射I/O的地址可以被高速缓存吗？解释为什么。

1. PuTTY 可在[www.putty.org](http://www.putty.org)免费下载。 [↑](#footnote-ref-1)
2. 当‘\r’被省略时，PuTTY也可以正确打印输出。 [↑](#footnote-ref-2)
3. 输出比较模块还可以被配置成基于计时器生成单一脉冲。 [↑](#footnote-ref-3)