在保护模式下测定内存容量

作者：Robert R. Collins

翻译：coly li

告诫

我在1992年写的这篇文档，从那时起肯定memory size的技术肯定已经变化了，但是基本概念还是一样的。如果你从这篇文档中看到一些过时的名词，或者对于过时的CPU和内存大小的引用，请原谅我。如果你看到一些过时的地方，如果能够将修改稿发给我，那我将不胜感激。

测定内容容量的概念

不说明RAM是如何工作的，就不能清楚的阐述memory sizing的概念。将内存看作2维数组的空间，行和列一样多。数组中的每一个元素代表了内存中的一位（不是一个字节）。为了对数组中的每一个单独的位（bit）寻址，我们为RAM芯片提供了行地址和列地址。

为了尽量减少芯片的引脚，RAM芯片只有一组地址引脚，并将他们公用于列和行地址（看图1）。当行地址通过行地址门（Row Address Strobe（RAS））有效时，或者当列地址通过列地址门（Column Address Strobe（CAS））有效时，内存控制器必须可以提供信号告诉RAM芯片对应的地址类型。

使用一组地址线来传输行地址和列地址，这种方式叫做复用或多路技术（MUX）。作为地址线复用的结果，RAM芯片的大小总是2的指数倍（或者2的偶数倍）。具有8条地址线的芯片可以保存64K bit的信息($(2^8)^2 = 65536$)；9根地址线可以保存256K bit的信息，10根地址线则为1M bit，以此类推。

进一步，我们就可以讨论CPU地址总线和内存地址总线了。286的 CPU具有24根地址引脚386、486和pentium的CPU具有32根地址总线，对于pentium pro则具有36根地址线。RAM芯片的引脚显然要比CPU的少的多，因此为了将CPU地址转换为RAM的RAS和CAS地址，CPU地址总线加入了一个内存控制器。这个内存控制器将CPU地址信号转换为RAS地址、CAS地址和BANK地址（关于BANK稍后有述）。

我们假设我们的机器是一个紧紧支持4个bank内存的286机器（虽然80286已经非常老旧了，但是描述他的内存配置对于举例来说是非常合适的）。这个内存可以由任何的连续的64K、256K、1M、4M的芯片构成，但是每个bank的内存芯片大小必须一致。为了支持4M的内存芯片，我们的内存控制器必须为11个RAS地址和11个CAS地址提供信号。这就会占用CPU24根地址线中的22根，剩余的2根地址线可以被内存控制器解释为BANK的片选信号。图2展示了CPU地址总线和内存地址总线的假想关系。根据此图，任何时候，当CPU 声明A00—A10，RAM芯片都会接收到一个列地址，同样，任何时候当CPU声明A12—A23时，RAM芯片就会接收到一个行地址。CPU地址线的A11和A12用来进行RAM的Bank的片选。由于我们假定计算机只支持4个Bank的内存，当<A12,A11>=00时，内存控制器选择Bank 0；当<A12,A11>=01时，内存控制器选择Bank1；当<A12,A11>=10时，内存控制器选择Bank2；当<A12,A11>=11时，内存控制器选择Bank3。

根据这些信息，我们就可以对计算机中的 RAM 芯片的任何一个 Bank 的任何一行进行写入操作了。在这种方式下，我们可以探测这个计算机中有多少个 Bank 上安装了 RAM 芯片，然后判定每一个 RAM Bank 的大小。要完善关于 RAM sizing 讨论，下面我们就需要应用我们的知识研究一些理论和算法。

在我们假想的计算机中，内存控制器根据 RAM 芯片的大小复用了地址线。图 2 展示了对于 4M 芯片的 CPU 地址总线/内存地址总线之间的关系。当芯片容量小于 4M 的时候，CPU 地址总线和 RAM 地址总线的关系就会改变了。在我们的计算机中，RAM bank 的芯片容量是可以通过程序配置的。因此我们需要测定插槽中的芯片的容量，然后当我们在计算机中正式访问内存之前重新对内存控制器编程来配置芯片大小。我们在一开始时假定每一个 RAM bank 都具有可以支持到 4M 的最大容量的 RAM 芯片。在我们的假想计算机中，内存控制器支持 4M 的芯片。因此，我们将内存控制器配置为支持 4MB 芯片的 4 个 bank。当进行内存控制器的配置时，我们可以探测到是否有一个小一些的内存芯片安装在插槽上。但是在我们检查芯片尺寸之前，我们需要先探测是否插槽上安装了内存条。

通过向 RAM bank 进行写操作的方法来来探测 RAM：首先写入一个特定的序列，然后再读出来，并检查是否是我们写入的那个序列。如果和我们写入的数值一样，那么我就就可以判定这个 bank 的 RAM 是有效的，然后我们就可以检查芯片的容量了。

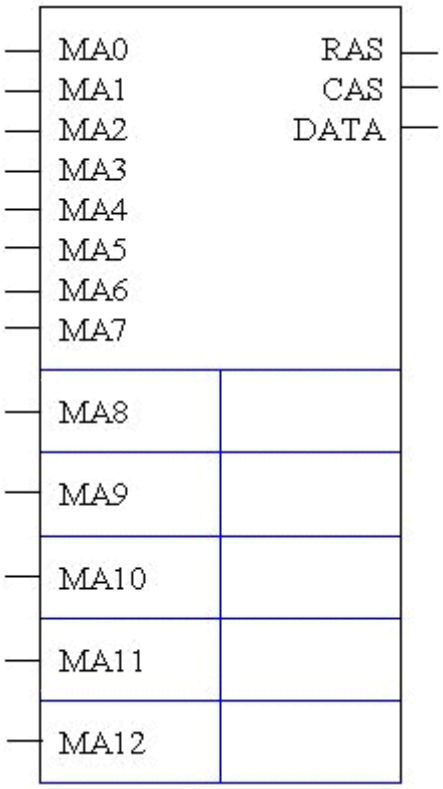

图 1 64K，256K，1M，4M，64M，256M

DRAM 引脚输出

		RAS	CAS
Multiplexed Addresses:	MA00	A13	A00
	MA01	A14	A01
Bank Select: A11, A12	MA02	A15	A02
	MA03	A16	A03
	MA04	A17	A04
	MA05	A18	A05
	MA06	A19	A06
	MA07	A20	A07
	MA08	A21	A08
	MA09	A22	A09
	MA10	A23	A10

图 2 针对 4MB DRAM 芯片的 CPU 地址总线转换

要测定RAM的实际容量，我们就需要探测到底有多少根地址线连接到RAM芯片上了。如果我们的RAM是4M的芯片，那么他的所有11根地址线（MA0—MA10）都是连接着的。如果这个RAM是一个1M的芯片，那就只有10根地址线（MA0—MA9）连接着。以此类推，256K的芯片有9根地址线（MA0—MA8），64K的有8根地址线（MA0—MA7）连接。为了测定是否安装的是一个4M的芯片，我们需要测定是否MA10被连接了。我们向MA10写入，然后从RAM地址0读回信息（MA0—MA10没有被声明）。如果一个4M的芯片被插在插槽上，那么当我们向MA10写入时，数据将显示在MA10，而不是MA0。但是如果这个芯片不是4M的，那么MA10就没有被连接，因此我们写入MA10的数据就会出现在地址0上。

我们的算法会对1M的MA9和256K的MA8重复的使用。如果MA8到MA10都失败了，鉴于我们已经确认在这个插槽上确实插了一个内存芯片，所以我们就可以很安全的认为插在这个插槽上的内存芯片是64K的。

为了探测有多少个Bank的内存安装在计算机上，我们需要探测每一个Bank上是否安装有RAM。这是通过向控制RAM Bank选择的CPU地址线写数据实现的。对于我们的CPU而言，就是地址线A11—A12。（可见，本文中Ax是指CPU总线地址，MA是指内存控制器的复用地址）。

通过对每一个bank上的内存重复测定RAM容量的算法，我们可以判定计算机中总共安装了多大容量的RAM。图3概要的说明了内存地址总线上的复用地址（MA），和CPU地址总线上的CPU地址的关系。图中的CPU地址是通过MA地址表计算出来的。如果我们需要声明

<A23>,<A11>和<A12>，那么我们就可以通过计算 $2^{23} + 2^{11} + 2^{12} = 801800h$ 得到 CPU 地址。

图 2　通过以下位置读/写操作测试 RAM 芯片容量

	Write at CPU Address Lines	Write at CPU Address	Read at CPU Address Lines	Read at CPU Address
Bank 0:				
4 MB	<A23>	800000h	<>	000000h
1 MB	<A22>	400000h	<>	000000h
256 kb	<A21>	200000h	<>	000000h
64 kb	<A20>	100000h	<>	000000h
Bank 1:				
4 MB	<A23, A11>	800800h	<A11>	000800h
1 MB	<A22, A11>	400800h	<A11>	000800h
256 kb	<A21, A11>	200800h	<A11>	000800h
64 kb	<A20, A11>	100800h	<A11>	000800h
Bank 2:				
4 MB	<A23, A12>	801000h	<A12>	001000h
1 MB	<A22, A12>	401000h	<A12>	001000h
256 kb	<A21, A12>	201000h	<A12>	001000h
64 kb	<A20, A20>	101000h	<A12>	001000h
Bank 3:				
4 MB	<A23, A12, A11>	801800h	<A12, A11>	001800h
1 MB	<A22, A12, A11>	401800h	<A12, A11>	001800h
256 kb	<A21, A12, A11>	201800h	<A12, A11>	001800h
64 kb	<A20, A12, A11>	101800h	<A12, A11>	001800h

总结：知道CPU地址、RAM行列地址，和bank选择地址之间的关系是测定RAM容量和数量的基础。RAM bank选择是通过向那些被内存控制器当作bank选择控制的CPU地址线写入信息实现的。

RAM容量是通过向给定容量的RAM最高行地址写入数据，然后从地址0读回的方法实现的。如果这个数据在地址0的位置出现了，你就知道这个RAM的行地址并没有连接到RAM芯片上（换句话说就是容量没有那么大）。接下来，对于每一个Bank上的RAM都依次使用该算法，就可以得到所有的内存容量了。这样，我们就可以根据测定到的内容容量对内存控制器进行适当的配置了。

在程序的控制下测定内存的容量和在上电期间测量内存容量是完全不同的事情。在上电期间，BIOS被赋予了对整个系统的完全控制权。BIOS可以认为cache功能是disable的，并且BIOS的操作不会影响结果。但是在程序控制下，我们在使用写/读如内存的算法来判定内存大小时，不能对硬件或者cache的状态有任何的假设。如果我们重新编程设置了内存控制器，鉴于CPU地址同RAM地址的转换机制发生了变化，我们的程序大多数情况下肯定会失败。

为了在程序控制下判定RAM的容量，我们必须使用不同的方法，一个不需要依靠硬件知识的方法。这意味着我们不能够重新设置内存控制器，或者cache控制器。由于我们不能够重新设置内存控制器（或者对cache控制器的存在与否、状态以及是否可编程进行任何假设），我们必须编写一种可以避免cache RAM而计算系统中存在的RAM容量的算法。因此，这个算法必须可以在检测内存是否存在时，使cache RAM的内容无效。

（译者：本文似乎没有给出在保护模式下解决这个问题的完整方法）