4.2.6　存储器的校验

在计算机运行过程中，由于种种原因致使数据在存储过程中可能出现差错。为了能及时发现错误并及时纠正错误，通常可将原数据配成汉明编码。

1. 汉明码的组成

汉明码是由 Richard Hanming 于 1950 年提出的，它具有一位纠错能力。

由编码纠错理论得知，任何一种编码是否具有检测能力和纠错能力，都与编码的最小距离有关。所谓编码最小距离，是指在一种编码系统中，任意两组合法代码之间的最少二进制位数的差异。

根据纠错理论得

$$L-1=D+C \qquad \text{且 } D\geqslant C$$

即编码最小距离 L 越大，则其检测错误的位数 D 越大，纠正错误的位数 C 也越大，且纠错能力恒小于或等于检错能力。例如，当编码最小距离 $L=3$ 时，这种编码可视为最多能检错二位，或能检错一位、纠错一位。可见，倘若能在信息编码中增加若干位检测位，增大 L，显然便能提高检错和纠错能力。汉明码就是根据这一理论提出的具有一位纠错能力的编码。

设欲检测的二进制代码为 n 位，为使其具有纠错能力，需增添 k 位检测位，组成 $n+k$ 位的代码。为了能准确对错误定位以及指出代码没错，新增添的检测位数 k 应满足：

$$2^k\geqslant n+k+1$$

由此关系可求得不同代码长度 n 所需检测位的位数 k，如表 4.2 所示。

表 4.2　代码长度与检测位位数的关系

n	k(最小)
1	2
2~4	3
5~11	4
12~26	5
27~57	6
58~120	7

k 的位数确定后，便可由它们所承担的检测任务设定它们在被传送代码中的位置及它们的取值。

设 $n+k$ 位代码自左至右依次编为第 1,2,3,…,$n+k$ 位，而将 k 位检测位记作 C_i($i=1,2,4,8,\cdots$)，分别安插在 $n+k$ 位代码编号的第 1,2,4,8,…,2^{k-1} 位上。这些检测位的位置设置是为了保证它们能分别承担 $n+k$ 位信息中不同数位所组成的“小组”的奇偶检测任务，使检测位和它所负责检测的小组中 1 的个数为奇数或偶数，具体分配如下：

C_1　检测的 g_1 小组包含 1,3,5,7,9,11,…位。

C_2　检测的 g_2 小组包含 2,3,6,7,10,11,14,15,…位。

C_4 检测的 g_3 小组包含 4,5,6,7,12,13,14,15,…位。

C_8 检测的 g_4 小组包含 8,9,10,11,12,13,14,15,24,…位。

⋮

其余检测位的小组所包含的位也可类推。这种小组的划分有如下特点:

① 每个小组 g_i 有一位且仅有一位为它所独占,这一位是其他小组所没有的,即 g_i 小组独占第 2^{i-1} 位($i=1,2,3,\cdots$)。

② 每两个小组 g_i 和 g_j 共同占有一位是其他小组没有的,即每两小组 g_i 和 g_j 共同占有第 $2^{i-1}+2^{j-1}$ 位($i,j=1,2,\cdots$)。

③ 每 3 个小组 g_i、g_j 和 g_l 共同占有第 $2^{i-1}+2^{j-1}+2^{l-1}$ 位,是其他小组所没有的。

依次类推,便可确定每组所包含的各位。

例如,欲传递信息为 $b_4b_3b_2b_1$($n=4$),根据 $2^k \geqslant n+k+1$,可求出配置成汉明码需增添检测位 $k=3$,且它们位置的安排如下:

二进制序号	1	2	3	4	5	6	7
名称	C_1	C_2	b_4	C_4	b_3	b_2	b_1

如果按配偶原则来配置汉明码,则 C_1 应使 1、3、5、7 位中的“1”的个数为偶数;C_2 应使 2、3、6、7 位中的“1”的个数为偶数;C_4 应使 4、5、6、7 位中的“1”的个数为偶数。

故 C_1 应为 3 位⊕5 位⊕7 位,即 $C_1=b_4 \oplus b_3 \oplus b_1$;$C_2$ 应为 3 位⊕6 位⊕7 位,即 $C_2=b_4 \oplus b_2 \oplus b_1$;$C_4$ 应为 5 位⊕6 位⊕7 位,即 $C_4=b_3 \oplus b_2 \oplus b_1$。

令 $b_4b_3b_2b_1=0101$,则

$C_1=b_4 \oplus b_3 \oplus b_1=0 \oplus 1 \oplus 1=0$

$C_2=b_4 \oplus b_2 \oplus b_1=0 \oplus 0 \oplus 1=1$

$C_4=b_3 \oplus b_2 \oplus b_1=1 \oplus 0 \oplus 1=0$

故 0101 的汉明码应为 $C_1C_2b_4C_4b_3b_2b_1$,即 0100101。

2. 汉明码的纠错过程

汉明码的纠错过程实际上是对传送后的汉明码形成新的检测位 P_i($i=1,2,4,8,\cdots$),根据 P_i 的状态,便可直接指出错误的位置。P_i 的状态是由原检测位 C_i 及其所在小组内“1”的个数确定的。倘若按配偶原则配置的汉明码,其传送后形成新的检测位 P_i 应为 0,否则说明传送有错,并且还可直接指出出错的位置。由于 P_i 与 C_i 有对应关系,故 P_i 可由下式确定:

$P_1=1 \oplus 3 \oplus 5 \oplus 7$,即 $P_1=C_1 \oplus b_4 \oplus b_3 \oplus b_1$

$P_2=2 \oplus 3 \oplus 6 \oplus 7$,即 $P_2=C_2 \oplus b_4 \oplus b_2 \oplus b_1$

$P_4=4 \oplus 5 \oplus 6 \oplus 7$,即 $P_4=C_4 \oplus b_3 \oplus b_2 \oplus b_1$

设已知传送的正确汉明码(按配偶原则配置)为 0100101,若传送后接收到的汉明码为 0100111,其出错位可按下述步骤确定。

令

二进制序号	1	2	3	4	5	6	7
正确的汉明码	0	1	0	0	1	0	1
接收到的汉明码	0	1	0	0	1	1	1

则新的检测位为

$P_4=4\oplus5\oplus6\oplus7$，即 $P_4=0\oplus1\oplus1\oplus1=1$

$P_2=2\oplus3\oplus6\oplus7$，即 $P_2=1\oplus0\oplus1\oplus1=1$

$P_1=1\oplus3\oplus5\oplus7$，即 $P_1=0\oplus0\oplus1\oplus1=0$

由此可见，传送结果 P_4 和 P_2 均不呈偶数，显然出了差错。那么，错在哪一位呢？仔细分析发现，只有第 6 位出错才会同时使 P_4 和 P_2 不呈偶数。同时，P_4、P_2、P_1 所构成的二进制值恰恰是出错的位置，即 $P_4P_2P_1=110$，表示第 6 位出错。发现错误后，计算机便自动将错误的第 6 位“1”纠正为“0”。

又如，若收到按偶配置的汉明码为 1100101，则经检测得

$P_4=4\oplus5\oplus6\oplus7$，即 $P_4=0\oplus1\oplus0\oplus1=0$

$P_2=2\oplus3\oplus6\oplus7$，即 $P_2=1\oplus0\oplus0\oplus1=0$

$P_1=1\oplus3\oplus5\oplus7$，即 $P_1=1\oplus0\oplus1\oplus1=1$

即 $P_4P_2P_1=001$，表示第 1 位出错。由于第 1 位不是欲传送的信息位，而是检测位，而检测位不参与运算，故在一般情况下可以不予纠正。

以上均以 $n=4$ 为例，其实对任意不同 n 位的信息，均可按上述步骤配置汉明码，即先求出需增加的检测位位数 k，再确定 C_i 的位置，然后，按奇或偶原则配置 C_i 各位的值即可。值得注意的是：按奇配置与按偶配置所求得的 C_i 值正好相反，而新的检测位 P_i 的取值与奇偶配置原则是相对应的，读者可自行分析。

汉明码常常被用在纠错一位的场合，若欲实现检错两位，实用时还得再增添一位检测位。

例 4.4　已知接收到的汉明码为 0110101（按配偶原则配置），试问欲传送的信息是什么？

解：由于要求出欲传送的信息，必须是正确的信息，因此不能简单地从接收到的 7 位汉明码中去掉 C_1、C_2、C_4 这 3 位检测位来求得。首先应该判断收到的信息是否出错。纠错过程如下：

$P_1=1\oplus3\oplus5\oplus7=1$

$P_2=2\oplus3\oplus6\oplus7=1$

$P_4=4\oplus5\oplus6\oplus7=0$

所以，$P_4P_2P_1=011$，第 3 位出错，可纠正为 0100101，故欲传送的信息为 0101。

例 4.5　按配奇原则配置 1100101 的汉明码。

解：根据 1100101，得 $n=7$。根据 $2^k\geqslant n+k+1$，可求出需增添 $k=4$ 位检测位，各位的安排如下：

二进制序号	1	2	3	4	5	6	7	8	9	10	11
汉明码	C_1	C_2	1	C_4	1	0	0	C_8	1	0	1

按配奇原则配置,则

$C_1=\overline{3\oplus5\oplus7\oplus9\oplus11}=1$

$C_2=\overline{3\oplus6\oplus7\oplus10\oplus11}=1$

$C_4=\overline{5\oplus6\oplus7}=0$

$C_8=\overline{9\oplus10\oplus11}=1$

故新配置的汉明码为11101001101。

4.2.7 提高访存速度的措施

随着计算机应用领域的不断扩大,处理的信息量越来越多,对存储器的工作速度和容量要求也越来越高。此外,因CPU的功能不断增强,I/O设备的数量不断增多,致使主存的存取速度已成为计算机系统的瓶颈。可见,提高访存速度也成为迫不及待的任务。为了解决此问题,除了寻找高速元件和采用层次结构以外,调整主存的结构也可提高访存速度。

1. 单体多字系统

由于程序和数据在存储体内是连续存放的,因此CPU访存取出的信息也是连续的,如果可以在一个存取周期内,从同一地址取出4条指令,然后再逐条将指令送至CPU执行,即每隔1/4存取周期,主存向CPU送一条指令,这样显然增大了存储器的带宽,提高了单体存储器的工作速度,如图4.41所示。

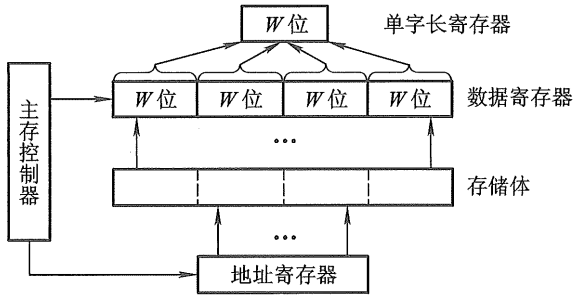

图4.41 单体四字结构存储器

图中示意了一个单体四字结构的存储器,每字W位。按地址在一个存取周期内可读出$4\times W$位的指令或数据,使主存带宽提高到4倍。显然,采用这种办法的前提是:指令和数据在主存内必须是连续存放的,一旦遇到转移指令,或者操作数不能连续存放,这种方法的效果就不明显。

2. 多体并行系统

多体并行系统就是采用多体模块组成的存储器。每个模块有相同的容量和存取速度，各模块各自都有独立的地址寄存器（MAR）、数据寄存器（MDR）、地址译码、驱动电路和读/写电路，它们能并行工作，又能交叉工作。

并行工作即同时访问 N 个模块，同时启动，同时读出，完全并行地工作（不过，同时读出的 N 个字在总线上需分时传送）。图 4.42 是适合于并行工作的高位交叉编址的多体存储器结构示意图，图中程序因按体内地址顺序存放（一个体存满后，再存入下一个体），故又有顺序存储之称。显然，高位地址可表示体号，低位地址为体内地址。按这种编址方式，只要合理调动，使不同的请求源同时访问不同的体，便可实现并行工作。例如，当一个体正与 CPU 交换信息时，另一个体可同时与外部设备进行直接存储器访问，实现两个体并行工作。这种编址方式由于一个体内的地址是连续的，有利于存储器的扩充。

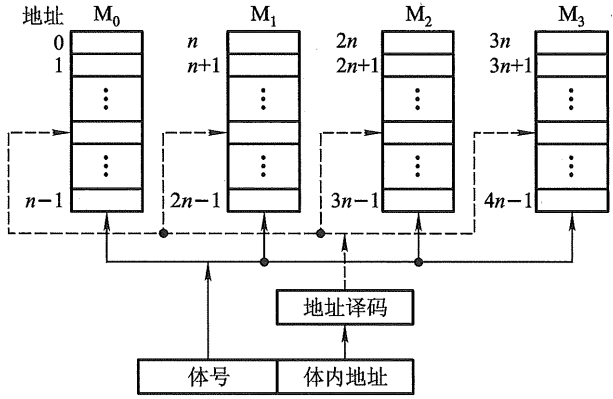

图 4.42　高位交叉编址的多体存储器

图 4.43 是按低位交叉编址的多体模块结构示意图。由于程序连续存放在相邻体中，故又有交叉存储之称。显然低位地址用来表示体号，高位地址为体内地址。这种编址方法又称为模 M 编址（M 等于模块数），表 4.3 列出了模 4 交叉编址的地址号。一般模块数 M 取 2 的方幂，使硬件电路比较简单。有的机器为了减少存储器冲突，采用质数个模块，例如，我国银河机的 M 为 31，其硬件实现比较复杂。

表 4.3　模 4 交叉编址地址表

体号	体内地址序号	最低两位地址
M_0	0,4,8,12,…,4i+0	00
M_1	1,5,9,13,…,4i+1	01
M_2	2,6,10,14,…,4i+2	10
M_3	3,7,11,15,…,4i+3	11

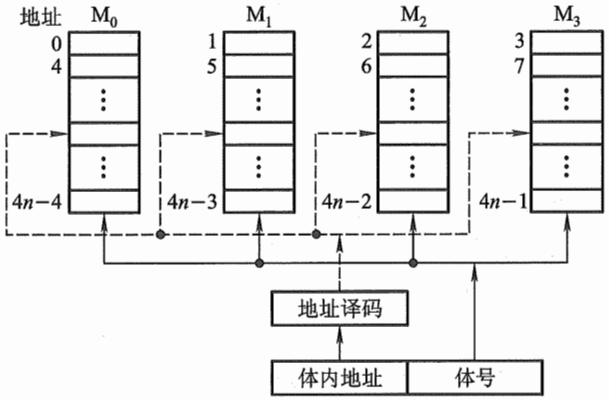

图 4.43 低位交叉编址的多体存储器

多体模块结构的存储器采用交叉编址后,可以在不改变每个模块存取周期的前提下,提高存储器的带宽。图 4.44 示意了 CPU 交叉访问 4 个存储体的时间关系,负脉冲为启动每个体的工作信号。虽然对每个体而言,存取周期均未缩短,但由于 CPU 交叉访问各体,使 4 个存储体的读/写过程重叠进行,最终在一个存取周期的时间内,存储器实际上向 CPU 提供了 4 个存储字。如果每个模块存储字长为 32 位,则在一个存取周期内(除第一个存取周期外),存储器向 CPU 提供了 32×4=128 位二进制代码,大大增加了存储器的带宽。

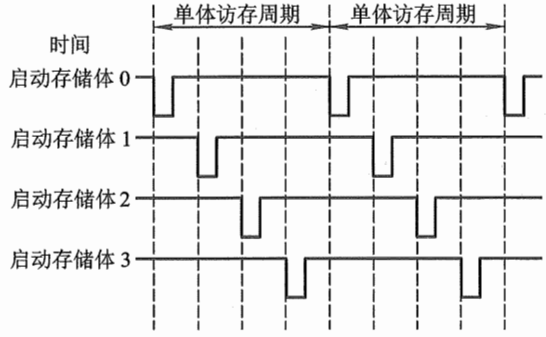

图 4.44 4 个存储体交叉访问的时间关系

假设每个体的存储字长和数据总线的宽度一致,并假设低位交叉的存储器模块数为 n,存取周期为 T,总线传输周期为 τ,那么当采用流水线方式(如图 4.44 所示)存取时,应满足 $T=n\tau$。为了保证启动某体后,经 $n\tau$ 时间再次启动该体时,它的上次存取操作已完成,要求低位交叉存储器的模块数大于或等于 n。以四体低位交叉编址的存储器为例,采用流水方式存取的示意图如图 4.45 所示。

可见，对于低位交叉的存储器，连续读取 n 个字所需的时间 t_1 为

$$t_1 = T + (n-1)\tau$$

若采用高位交叉编址，则连续读取 n 个字所需的时间 t_2 为

$$t_2 = nT$$

例 4.6　设有 4 个模块组成的四体存储器结构，每个体的存储字长为 32 位，存取周期为 200 ns。假设数据总线宽度为 32 位，总线传输周期为 50 ns，试求顺序存储和交叉存储的存储器带宽。

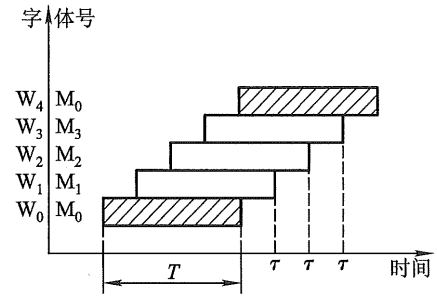

图 4.45　四体低位交叉编址存储器流水线工作方式示意图

解：顺序存储（高位交叉编址）和交叉存储（低位交叉编址）连续读出 4 个字的信息量是 32×4=128 位。

顺序存储存储器连续读出 4 个字的时间是

$$200\ \text{ns} \times 4 = 800\ \text{ns} = 8 \times 10^{-7}\ \text{s}$$

交叉存储存储器连续读出 4 个字的时间是

$$200\ \text{ns} + 50\ \text{ns} \times (4-1) = 350\ \text{ns} = 3.5 \times 10^{-7}\ \text{s}$$

顺序存储器的带宽是

$$128/(8 \times 10^{-7}) = 16 \times 10^{7}\ \text{bps}$$

交叉存储器的带宽是

$$128/(3.5 \times 10^{-7}) = 37 \times 10^{7}\ \text{bps}$$

多体模块存储器不仅要与 CPU 交换信息，还要与辅存、I/O 设备，乃至 I/O 处理机交换信息。因此，在某一时刻，决定主存究竟与哪个部件交换信息必须由存储器控制部件（简称存控）来承担。存控具有合理安排各部件请求访问的顺序以及控制主存读/写操作的功能。图 4.46 是一个存控基本结构框图，它由排队器、控制线路、节拍发生器及标记触发器等组成。

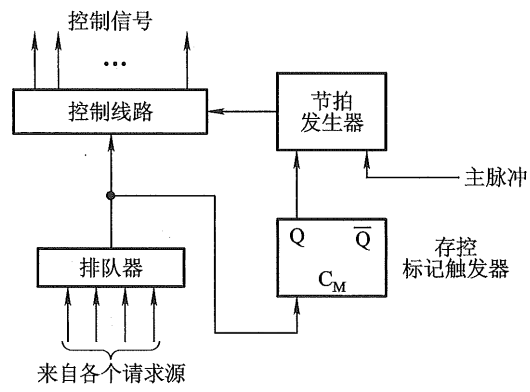

图 4.46　存控基本结构框图

（1）排队器

由于要求访存的请求源很多，而且访问都是随机的，这样有可能在同一时刻出现多个请求源请求访问同一个存储体。为了防止发生两个以上的请求源同时占用同一存储体，并防止将代码错送到另一个请求源等各种错误的发生，在存控内需设置一个排队器，由它来确定请求源的优先级别。其确定原则如下：

① 对易发生代码丢失的请求源，应列为最高优先级，例如，外设信息最易丢失，故它的级别最高。

② 对严重影响 CPU 工作的请求源,给予次高优先级,否则会导致 CPU 工作失常。

例如,写数请求高于读数,读数请求高于读指令。若运算部件不能尽快送走已算出的结果,会严重影响后续指令的执行,因此,发生这种情况时,写数的优先级比读数、读指令都高。若没有操作数参与运算,取出更多的指令也无济于事,故读数的优先级又应比读指令高。

(2) 存控标记触发器 C_M

它用来接受排队器的输出信号,一旦响应某请求源的请求,C_M 被置"1",以便启动节拍发生器工作。

(3) 节拍发生器

它用来产生固定节拍,与机器主脉冲同步,使控制线路按一定时序发出信号。

(4) 控制线路

由它将排队器给出的信号与节拍发生器提供的节拍信号配合,向存储器各部件发出各种控制信号,用以实现对总线控制及完成存储器读/写操作,并向请求源发出回答信号,表示存储器已响应了请求等。

3. 高性能存储芯片

采用高性能存储芯片也是提高主存速度的措施之一。DRAM 集成度高,价格便宜,广泛应用于主存。其发展速度很快,几乎每隔 3 年存储芯片的容量就翻两番。为了进一步提高 DRAM 的性能,人们开发了许多对基本 DRAM 结构的增强功能,出现了 SDRAM、RDRAM 和 CDRAM。

(1) SDRAM (Synchronous DRAM,同步 DRAM)

SDRAM 与常用的异步 DRAM 不同,它与处理器的数据交换同步于系统的时钟信号,并且以处理器-存储器总线的最高速度运行,而不需要插入等待状态。典型的 DRAM 中,处理器将地址和控制信号送至存储器后,需经过一段延时,供 DRAM 执行各种内部操作(如输入地址、读出数据等),才能将数据从存储器中读出或将数据写入存储器中。此时,如果 CPU 的速度与 DRAM 匹配,那么这个延时不会影响 CPU 的工作速度;如果 CPU 的速度更高,那么在这段时间内,CPU 只能"等待",降低了 CPU 的执行速度。而 SDRAM 能在系统时钟的控制下进行数据的读出和写入,CPU 给出的地址和控制信号会被 SDRAM 锁存,直到指定的时钟周期数后再响应。此时 CPU 可执行其他任务,无须"等待"。例如,系统的时钟周期为 10 ns,存储器接到地址后需 50 ns 读出数据。对于异步工作的 DRAM,CPU 要"等待"50 ns 获得数据,而对同步工作的 SDRAM 而言,CPU 只需把地址放入锁存器中,在存储器进行读操作期间去完成其他操作。当 CPU 计时到 5 个时钟周期后,便可获得从存储器读出的数据。

SDRAM 还支持猝发访问模式,即 CPU 发出一个地址就可以连续访问一个数据块(通常为 32 字节)。SDRAM 芯片内还可以包含多个存储体,这些体可以轮流工作,提高访问速度。现在又出现了双数据速率的 SDRAM (Double Data Rate SDRAM,DDR-SDRAM),它是 SDRAM 的增强型版本,可以每周期两次向处理器送出数据。

(2) RDRAM (Rambus DRAM)

由 Rambus 开发的 RDRAM 采用专门的 DRAM 和高性能的芯片接口取代现有的存储器接口。它主要解决存储器带宽的问题,通过高速总线获得存储器请求(包括操作时所需的地址、操

作类型和字节数),总线最多可寻址 320 块 RDRAM 芯片,传输率可达 1.6 GBps。它不像传统的 DRAM 采用 $\overline{RAS}$、$\overline{CAS}$ 和 $\overline{WE}$ 信号来控制,而是采用异步的面向块的传输协议传送地址信息和数据信息。一个 RDRAM 芯片就像一个存储系统,通过一种新的互连电路 RamLink,将各个 RDRAM 芯片连接成一个环,数据通信在主存控制器的控制下进行,数据交换以包为单位。图 4.47示意了 RamLink 体系结构。

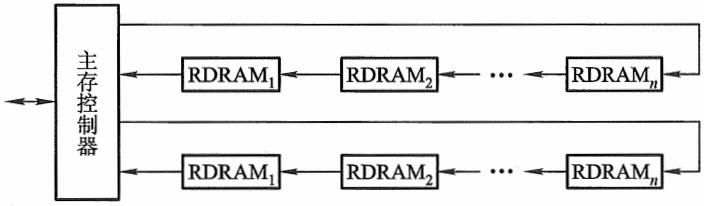

图 4.47　RamLink 体系结构

(3) 带 Cache 的 DRAM (CDRAM)

带 Cache 的 DRAM 是在通常的 DRAM 芯片内又集成了一个小的 SRAM,又称增强型的 DRAM(EDRAM)。图 4.48 是 1 M×4 位的 CDRAM,其中 SRAM 为 512×4 位,DRAM 排列成 2048×512×4 位的阵列。

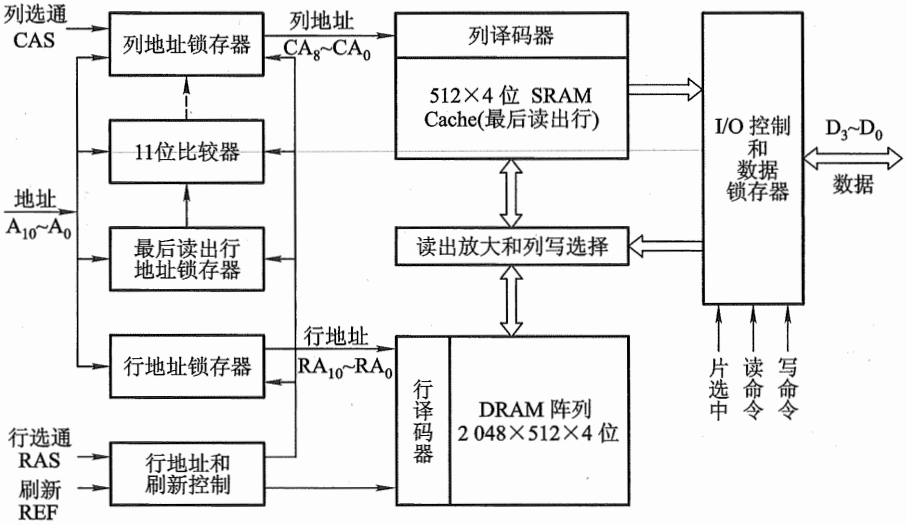

图 4.48　1 M×4 位 CDRAM 芯片结构框图

由图中可见,地址引脚线只有 11 根(A_{10} ~ A_0),而 1 M×4 位的存储芯片对应 20 位地址,此 20 位地址需分时送入芯片内部。首先在行选通信号作用下,高 11 位地址经地址引脚线输入,分别保存在行地址锁存器中和最后读出行地址锁存器中。在 DRAM 的 2048 行中,此指定行地址

的全部数据 512×4 位被读到 SRAM 中暂存。然后在列选通信号作用下,低 9 位地址经地址引脚线输入,保存到列地址锁存器中。在读命令有效时,512 个 4 位组的 SRAM 中某一 4 位组被这个列地址选中,经数据线 $D_3 \sim D_0$ 从芯片输出。

下一次读取时,输入的行地址立即与最后读出行锁存器的内容进行 11 位比较。若比较相符,说明该数据在 SRAM 中,再由输入列地址选择某一 4 位组输出;若比较不相符,则需驱动 DRAM 阵列更新 SRAM 和最后读出行地址锁存器中的内容,并送出指定的 4 位组。

由此可见,以 SRAM 保存一行内容的方法,当对连续高 11 位地址相同(属于同一行地址)的数据进行读取时,只需连续变动 9 位列地址就可使相应的 4 位组连续读出,这被称为猝发式读取,对成块传送十分有利。

从图 4.48 所示的结构可见,芯片内的数据输出路径(由 SRAM 到 I/O)与数据输入路径(由 I/O 到读放大器和列写选择)是分开的,这就允许在写操作完成的同时启动同一行的读操作。此外,在 SRAM 读出期间可同时对 DRAM 阵列进行刷新。

4.3 高速缓冲存储器

4.3.1 概述

1. 问题的提出

在多体并行存储系统中,由于 I/O 设备向主存请求的级别高于 CPU 访存,这就出现了 CPU 等待 I/O 设备访存的现象,致使 CPU 空等一段时间,甚至可能等待几个主存周期,从而降低了 CPU 的工作效率。为了避免 CPU 与 I/O 设备争抢访存,可在 CPU 与主存之间加一级缓存(参见图 4.3),这样,主存可将 CPU 要取的信息提前送至缓存,一旦主存在与 I/O 设备交换时,CPU 可直接从缓存中读取所需信息,不必空等而影响效率。

从另一角度来看,主存速度的提高始终跟不上 CPU 的发展。据统计,CPU 的速度平均每年改进 60%,而组成主存的动态 RAM 速度平均每年只改进 7%,结果是 CPU 和动态 RAM 之间的速度间隙平均每年增大 50%。例如,100 MHz 的 Pentium 处理器平均每 10 ns 就执行一条指令,而动态 RAM 的典型访问时间为 60~120 ns。这也希望由高速缓存 Cache 来解决主存与 CPU 速度的不匹配问题。

Cache 的出现使 CPU 可以不直接访问主存,而与高速 Cache 交换信息。那么,这是否可能呢? 通过大量典型程序的分析,发现 CPU 从主存取指令或取数据,在一定时间内,只是对主存局部地址区域的访问。这是由于指令和数据在主存内都是连续存放的,并且有些指令和数据往往会被多次调用(如子程序、循环程序和一些常数),即指令和数据在主存的地址分布不是随机的,而是相对的簇聚,使得 CPU 在执行程序时,访存具有相对的局部性,这就称为程序访问的局部性

原理。根据这一原理，很容易设想，只要将 CPU 近期要用到的程序和数据提前从主存送到 Cache，那么就可以做到 CPU 在一定时间内只访问 Cache。一般 Cache 采用高速的 SRAM 制作，其价格比主存贵，但因其容量远小于主存，因此能很好地解决速度和成本的矛盾。

2. Cache 的工作原理

图 4.49 是 Cache-主存存储空间的基本结构示意图。

主存由 2^n 个可编址的字组成，每个字有唯一的 n 位地址。为了与 Cache 映射，将主存与缓存都分成若干块，每块内又包含若干个字，并使它们的块大小相同（即块内的字数相同）。这就将主存的地址分成两段：高 m 位表示主存的块地址，低 b 位表示块内地址，则 $2^m = M$ 表示主存的块数。同样，缓存的地址也分为两段：高 c 位表示缓存的块号，低 b 位表示块内地址，则 $2^c = C$ 表示缓存块数，且 C 远小于 M。主存与缓存地址中都用 b 位表示其块内字数，即 $B = 2^b$ 反映了块的大小，称 B 为块长。

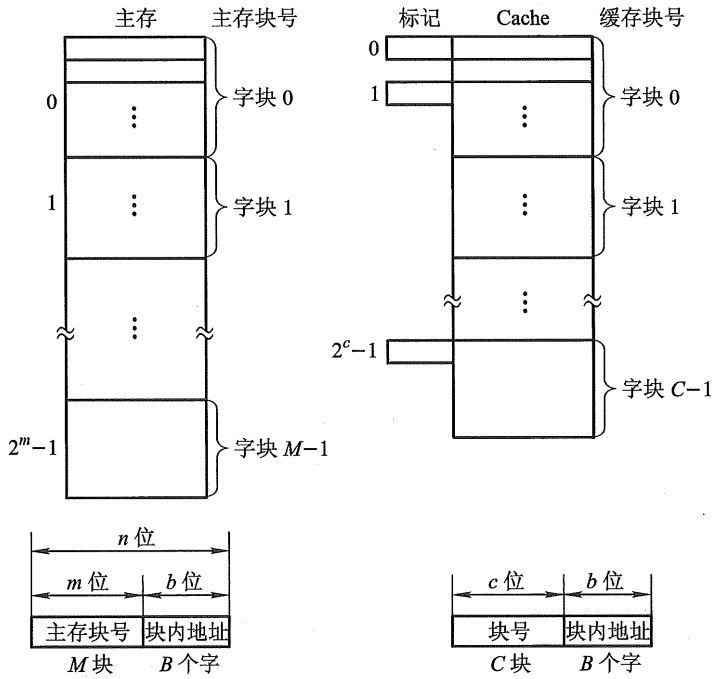

图 4.49　Cache-主存存储空间的基本结构

任何时刻都有一些主存块处在缓存块中。CPU 欲读取主存某字时，有两种可能：一种是所需要的字已在缓存中，即可直接访问 Cache（CPU 与 Cache 之间通常一次传送一个字）；另一种是所需的字不在 Cache 内，此时需将该字所在的主存整个字块一次调入 Cache 中（Cache 与主存之间是字块传送）。如果主存块已调入缓存块，则称该主存块与缓存块建立了对应关系。

上述第一种情况为CPU访问Cache命中,第二种情况为CPU访问Cache不命中。由于缓存的块数 C 远小于主存的块数 M,因此,一个缓存块不能唯一地、永久地只对应一个主存块,故每个缓存块需设一个标记(参见图4.49),用来表示当前存放的是哪一个主存块,该标记的内容相当于主存块的编号。CPU读信息时,要将主存地址的高 m 位(或 m 位中的一部分)与缓存块的标记进行比较,以判断所读的信息是否已在缓存中(参见图4.54)。

Cache的容量与块长是影响Cache效率的重要因素,通常用"命中率"来衡量Cache的效率。命中率是指CPU要访问的信息已在Cache内的比率。

在一个程序执行期间,设 N_c 为访问Cache的总命中次数,N_m 为访问主存的总次数,则命中率 h 为

$$h=\frac{N_c}{N_c+N_m}$$

设 t_c 为命中时的Cache访问时间,t_m 为未命中时的主存访问时间,$1-h$ 表示未命中率,则Cache-主存系统的平均访问时间 t_a 为

$$t_a=ht_c+(1-h)t_m$$

当然,以较小的硬件代价使Cache-主存系统的平均访问时间 t_a 越接近于 t_c 越好。用 e 表示访问效率,则有

$$e=\frac{t_c}{t_a}\times 100\%=\frac{t_c}{ht_c+(1-h)t_m}\times 100\%$$

可见,为提高访问效率,命中率 h 越接近1越好。

例4.7 假设CPU执行某段程序时,共访问Cache命中2000次,访问主存50次。已知Cache的存取周期为50 ns,主存的存取周期为200 ns。求Cache-主存系统的命中率、效率和平均访问时间。

解:(1) Cache的命中率为

$$2000/(2000+50)=0.97$$

(2) 由题可知,访问主存的时间是访问Cache时间的4倍(200/50=4)。

设访问Cache的时间为 t,访问主存的时间为 $4t$,Cache-主存系统的访问效率为 e,则

$$e=\frac{\text{访问 Cache 的时间}}{\text{平均访问时间}}\times 100\%$$

$$=\frac{t}{0.97\times t+(1-0.97)\times 4t}\times 100\%=91.7\%$$

(3) 平均访问时间为

$$50\ \text{ns}\times 0.97+200\ \text{ns}\times(1-0.97)=54.5\ \text{ns}$$

一般而言,Cache容量越大,其CPU的命中率就越高。当然容量也没必要太大,太大会增加成本,而且当Cache容量达到一定值时,命中率已不因容量的增大而有明显的提高。因此,Cache容量是总成本价与命中率的折中值。例如,80386的主存最大容量为4 GB,与其配套的Cache容

量为 16 KB 或 32 KB,其命中率可达 95%以上。

块长与命中率之间的关系更为复杂,它取决于各程序的局部特性。当块由小到大增长时,起初会因局部性原理使命中率有所提高。由局部性原理指出,在已被访问字的附近,近期也可能被访问,因此,增大块长,可将更多有用字存入缓存,提高其命中率。可是,倘若继续增大块长,命中率很可能下降,这是因为所装入缓存的有用数据反而少于被替换掉的有用数据。由于块长的增大,导致缓存中块数的减少,而新装入的块要覆盖旧块,很可能出现少数块刚刚装入就被覆盖,因此命中率反而下降。再者,块增大后,追加上的字距离已被访问的字更远,故近期被访问的可能性会更小。块长的最优值是很难确定的,一般每块取 4 至 8 个可编址单位(字或字节)较好,也可取一个主存周期所能调出主存的信息长度。例如,CRAY-1 的主存是 16 体交叉,每个体为单字宽,其存放指令的 Cache 块长为 16 个字。又如,IBM 370/168 机主存是 4 体交叉,每个体宽为 64 位(8 个字节),其 Cache 块长为 32 个字节。

3. Cache 的基本结构

Cache 的基本结构原理框图如图 4.50 所示。

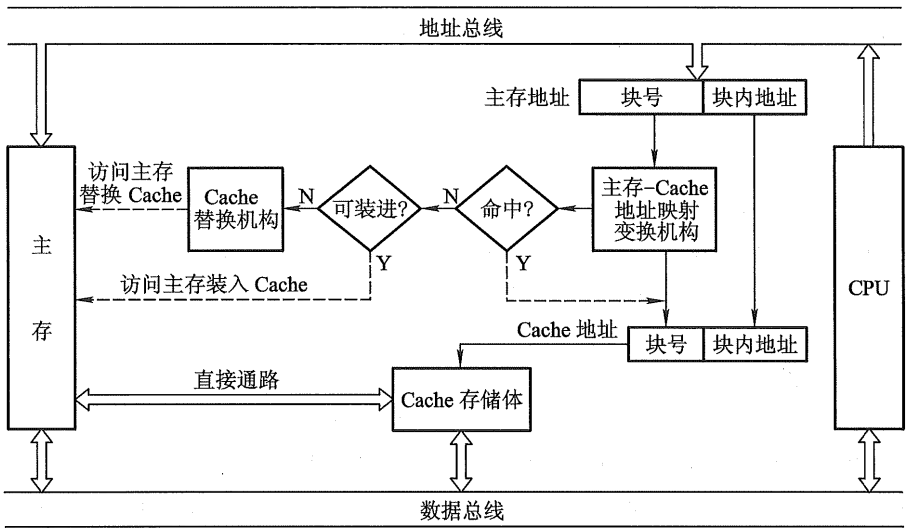

图 4.50　Cache 的基本结构原理框图

它主要由 Cache 存储体、地址映射变换机构、Cache 替换机构几大模块组成。

(1) Cache 存储体

Cache 存储体以块为单位与主存交换信息,为加速 Cache 与主存之间的调动,主存大多采用多体结构,且 Cache 访存的优先级最高。

(2) 地址映射变换机构

地址映射变换机构是将 CPU 送来的主存地址转换为 Cache 地址。由于主存和 Cache 的块大小相同,块内地址都是相对于块的起始地址的偏移量(即低位地址相同),因此地址变换主要

是主存的块号(高位地址)与 Cache 块号间的转换。而地址变换又与主存地址以什么样的函数关系映射到 Cache 中(称为地址映射)有关,这些内容可详见 4.3.2 节。

如果转换后的 Cache 块已与 CPU 欲访问的主存块建立了对应关系,即已命中,则 CPU 可直接访问 Cache 存储体。如果转换后的 Cache 块与 CPU 欲访问的主存块未建立对应关系,即不命中,此刻 CPU 在访问主存时,不仅将该字从主存取出,同时将它所在的主存块一并调入 Cache,供 CPU 使用。当然,此刻能将主存块调入 Cache 内,也是由于 Cache 原来处于未被装满的状态。反之,倘若 Cache 原来已被装满,即已无法将主存块调入 Cache 内时,就得采用替换策略。

(3) 替换机构

当 Cache 内容已满,无法接受来自主存块的信息时,就由 Cache 内的替换机构按一定的替换算法来确定应从 Cache 内移出哪个块返回主存,而把新的主存块调入 Cache。有关替换算法详见 4.3.3 节。

特别需指出的是,Cache 对用户是透明的,即用户编程时所用到的地址是主存地址,用户根本不知道这些主存块是否已调入 Cache 内。因为,将主存块调入 Cache 的任务全由机器硬件自动完成。

(4) Cache 的读写操作

读操作的过程可用流程图 4.51 来描述。当 CPU 发出主存地址后,首先判断该存储字是否在 Cache 中。若命中,直接访问 Cache,将该字送至 CPU;若未命中,一方面要访问主存,将该字传送给 CPU,与此同时,要将该字所在的主存块装入 Cache,如果此时 Cache 已装满,就要执行替换算法,腾出空位才能将新的主存块调入。

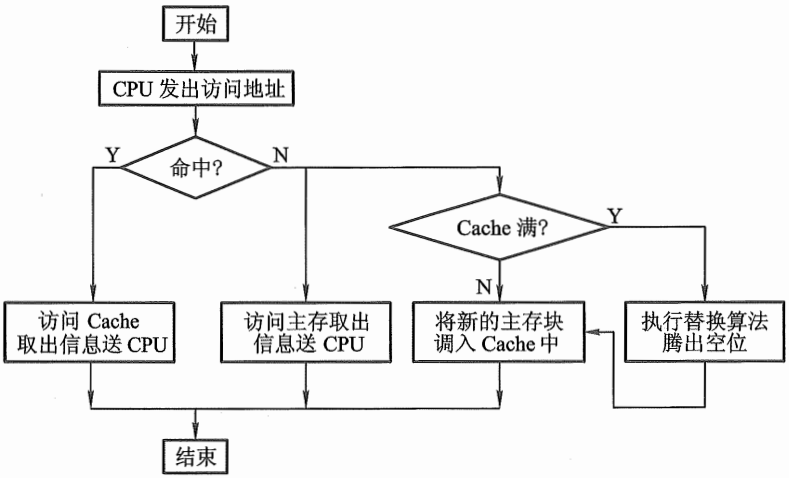

图 4.51 Cache 的读数操作流程

写操作比较复杂,因为对 Cache 块内写入的信息,必须与被映射的主存块内的信息完全一致。当程序运行过程中需对某个单元进行写操作时,会出现如何使 Cache 与主存内容保持一致的问题。目前主要采用以下几种方法。

① 写直达法(Write-through),又称为存直达法(Store-through),即写操作时数据既写入 Cache 又写入主存。它能随时保证主存和 Cache 的数据始终一致,但增加了访存次数。

② 写回法(Write-back),又称为拷回法(Copy-back),即写操作时只把数据写入 Cache 而不写入主存,但当 Cache 数据被替换出去时才写回主存。可见写回法 Cache 中的数据会与主存中的不一致。为了识别 Cache 中的数据是否与主存一致,Cache 中的每一块要增设一个标志位,该位有两个状态:"清"(表示未修改过,与主存一致)和"浊"(表示修改过,与主存不一致)。在 Cache 替换时,"清"的 Cache 块不必写回主存,因为此时主存中相应块的内容与 Cache 块是一致的。在写 Cache 时,要将该标志位设置为"浊",替换时此 Cache 块要写回主存,同时要使标志位为"清"。

写回法和写直达法各有特色。在写直达法中,由于 Cache 中的数据始终和主存保持一致,在读操作 Cache 失效时,只需选择一个替换的块(主存块)调入 Cache,被替换的块(Cache 块)不必写回主存。可见读操作不涉及对主存的写操作。因此这种方法更新策略比较容易实现。但是在写操作时,既要写入 Cache 又要写入主存,因此写直达法的"写"操作时间就是访问主存的时间。

在写回法中,写操作时只写入 Cache,故"写"操作时间就是访问 Cache 的时间,因此速度快。这种方法对主存的写操作只发生在块替换时,而且对 Cache 中一个数据块的多次写操作只需一次写入主存,因此可减少主存的写操作次数。但在读操作 Cache 失效时要发生数据替换,引起被替换的块写回主存的操作,增加了 Cache 的复杂性。

对于有多个处理器的系统,各自都有独立的 Cache,且都共享主存,这样又出现了新问题。即当一个缓存中数据被修改时,不仅主存中相对应的字无效,连同其他缓存中相对应的字也无效(当然恰好其他缓存也有相应的字)。即使通过写直达法改变了主存的相应字,而其他缓存中数据仍然无效。显然,解决系统中 Cache 一致性的问题很重要。当今研究 Cache 一致性问题非常活跃,想进一步了解可查阅有关资料。

4. Cache 的改进

Cache 刚出现时,典型系统只有一个缓存,近年来普遍采用多个 Cache。其含义有两方面:一是增加 Cache 的级数;二是将统一的 Cache 变成分立的 Cache。

(1) 单一缓存和两级缓存

所谓单一缓存,是指在 CPU 和主存之间只设一个缓存。随着集成电路逻辑密度的提高,又把这个缓存直接与 CPU 制作在同一个芯片内,故又称为片内缓存(片载缓存)。片内缓存可以提高外部总线的利用率,因为将 Cache 制作在芯片内,CPU 直接访问 Cache 不必占用芯片外的总线(系统总线),而且片内缓存与 CPU 之间的数据通路很短,大大提高了存取速度,外部总线又可更多地支持 I/O 设备与主存的信息传输,增强了系统的整体效率。例如,Intel 80486 CPU 芯片内就