



计算机系统结构系列教材

计算机原理 与设计

王保恒 肖晓强 张春元 文 梅 编

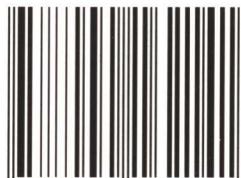


高等教育出版社

计算机系统结构系列教材

- 电子与电路技术基础
- 数字逻辑原理与工程设计
- ◆ 计算机原理与设计
- 汇编语言程序设计
- 计算机体系结构
- 嵌入式系统设计
- 数字系统设计工程

ISBN 7-04-016214-8



9 787040 162141 >

定价 33.20 元

计算机系统结构系列教材

计算机原理与设计

王保恒 肖晓强 张春元 文 梅 编

高等 教育 出 版 社

内容提要

本书从计算机部件及其行为层次角度系统地阐述了电子数字计算机的结构组成、逻辑功能、工作原理和实现方法。全书共7章，内容包括：计算机基本组成和工作原理，指令系统与设计，运算方法与运算器，控制器设计技术，存储器与存储器设计，输入输出控制方式以及计算机互连结构等。

本书取材较新，同时保留了经典计算机组成与设计的相关知识。本书采用实例教学的组织方式，内容由浅入深，相互联系。书中给出了大量的设计实例、例题和习题便于学习。

本书可作为高等院校计算机类、电子类和自动化类等相关专业的教材和参考书，也可供相关专业工程技术人员参考。

图书在版编目 (CIP) 数据

计算机原理与设计/王保恒等编. —北京：高等教育出版社，2005.1

(计算机系统结构系列教材)

ISBN 7-04-016214-8

I . 计... II . 王... III . 电子计算机 - 基础理论 -
高等学校 - 教材 IV . TP301

中国版本图书馆 CIP 数据核字(2005)第 002068 号

策划编辑 刘建元 责任编辑 付 欣 市场策划 陈 振
封面设计 李卫青 责任印制 杨 明

出版发行 高等教育出版社
社 址 北京市西城区德外大街 4 号
邮政编码 100011
总 机 010-58581000

购书热线 010-58581118
免费咨询 800-810-0598
网 址 <http://www.hep.edu.cn>
<http://www.hep.com.cn>
网上订购 <http://www.landraco.com>
<http://www.landraco.com.cn>

经 销 北京蓝色畅想图书发行有限公司
印 刷 北京宏伟双华印刷有限公司

开 本 787×1092 1/16 版 次 2005 年 1 月第 1 版
印 张 26.75 印 次 2005 年 1 月第 1 次印刷
字 数 550 000 定 价 33.20 元

本书如有缺页、倒页、脱页等质量问题，请到所购图书销售部门联系调换。

版权所有 侵权必究

物料号:16214-00

前　　言

计算机原理是计算机学科各专业必修的一门重要的专业基础课,它以计算机单机系统为研究对象,从计算机部件及其行为层次角度阐述电子数字计算机的结构组成、逻辑功能、工作原理和实现方法。

本书作为计算机系统结构系列教材中的一本,全书共7章。第一章重点介绍冯·诺依曼计算机的基本结构、各部件组成和工作流程。第二章阐述计算机的指令系统,包括数据表示、指令类型与指令格式以及寻址方式等内容。第三章讨论计算机的运算器及运算方法,重点介绍定点四则运算的运算方法和运算器以及浮点的运算方法。第四章通过简单的8位实例计算机的设计,介绍组合逻辑和微程序两种控制器的基本组成部件、设计步骤与方法以及设计的实现问题。第五章阐述各种物理存储器的基本组成与工作原理,讨论内存存储器的设计和并行主存技术。第六章简单介绍I/O设备,阐述I/O控制和I/O子系统的基本概念,重点介绍程序查询、程序中断、DMA以及I/O通道四种I/O控制方式的硬件组成、工作原理和工作过程。第七章简介计算机的模块及各种互连结构,重点讨论总线的基本组成、基本工作原理、结构及其设计问题,最后介绍总线标准及实例——PCI总线。

本书具有如下特色:

1. 既注重计算机的基本概念、基本组成和基本工作原理,又兼顾它的逻辑设计。除对计算机的各大基本部件的组成原理及其相互关系进行了详尽的描述外,还给出了控制器、内存存储器设计实例,提供了多种运算方法对应运算器的逻辑结构图。

2. 为适应学科发展,保留经典而又传统计算机的内容及其描述方式,淘汰陈旧内容和烦琐电路,补充了如快擦写存储器、多种DMA传送接口及PCI总线等新技术和新器件,并把重点放在逻辑器件(芯片)的外特性及其使用和逻辑实现上。

3. 书中列举大量的实例和例题以方便读者理解,尤其是第四章实例计算机的设计,将计算机指令系统、运算器、控制器等部件形成了一个有机整体,便于读者从整体上了解计算机和把握计算机的工作过程。每章后都附有相当数量的习题,通过回答和演算习题,可帮助读者理解和掌握所学知识。

4. 本书按照运算器、控制器、存储器、输入和输出设备的顺序分别介绍计算机各大硬件的基本组成部件,但各章之间并不是相互独立的,在介绍每种部件时都注意说明与其他各章的联系。例如第四章的控制器设计中就用到了第二章介绍的指令系统的内容,第三章运算器的组织就利用了第七章总线系统的相关知识等。

本书第一章由王保恒、肖晓强编写,第二章由文梅编写,第三章和第七章由肖晓强编写,

第四章由张春元、肖晓强、文梅编写，第五章和第六章由王保恒编写，全书由王保恒负责统编和定稿。在成书的过程中，国防科学技术大学计算机学院王凤学教授仔细阅读了全书，并提出了许多具体的修改意见，特在此表示感谢。南开大学陆志才教授审阅了本书，提出了宝贵的意见，编者据此进行了修改，并表示诚挚的谢意。本书得到了国防科学技术大学计算机学院、计算机科学与技术系领导和专家的关心和帮助，在此一并致谢。

本书适合作为高等学校本科计算机类、电子类和自动化类等相关专业的教材和参考书，参考授课学时数为 72 学时。本书也可供从事计算机工作的工程技术人员参考。

虽然编者在成书前对书稿进行了多次校正，但由于水平和经验有限，疏忽之处在所难免，恳请专家和读者批评指正。

编 者

2004 年 10 月

目 录

第一章 绪论	(1)
引言	(1)
1.1 计算机的发展史	(1)
1.1.1 计算简史	(1)
1.1.2 计算机发展简史	(3)
1.2 计算机系统组成	(4)
1.2.1 存储程序原理	(4)
1.2.2 计算机系统的组成	(5)
1.2.3 计算机硬件系统组成	(6)
1.3 计算机的工作过程	(11)
1.3.1 使用计算机求解的一个简单例子	(11)
1.3.2 指令执行过程	(13)
1.3.3 计算机工作的过程	(15)
1.4 计算机的性能指标和分类	(16)
1.4.1 计算机的性能指标	(16)
1.4.2 计算机的分类	(18)
1.5 计算机应用与发展	(20)
1.5.1 计算机的应用	(20)
1.5.2 计算机发展前景	(22)
小结	(24)
习题	(24)
第二章 指令系统	(26)
引言	(26)
2.1 数据表示	(27)
2.1.1 数值数据的数据表示	(27)
2.1.2 字符的数据表示	(34)
2.1.3 堆栈数据表示	(36)
2.1.4 向量数据表示	(38)
2.1.5 数据表示小结	(42)
2.2 指令格式	(42)
2.2.1 指令长度	(43)
2.2.2 操作码结构	(44)
2.2.3 地址码结构	(45)
2.3 寻址技术	(47)
2.3.1 基本寻址方式	(47)
2.3.2 复合寻址方式	(53)
2.4 基本指令和指令类型	(53)
2.4.1 数据传送指令	(53)
2.4.2 算术逻辑运算指令	(54)
2.4.3 数据转换指令	(55)
2.4.4 程序控制指令	(55)
2.4.5 输入/输出指令	(59)
2.4.6 系统控制指令	(60)
2.5 指令系统的分类	(60)
2.6 指令系统设计与实例	(62)
2.6.1 指令系统设计的基本要求	(62)
2.6.2 CISC 和 RISC	(63)
2.6.3 指令系统实例	(64)
小结	(74)
习题	(75)
第三章 运算方法与运算器	(77)
引言	(77)
3.1 基本运算	(77)
3.1.1 逻辑运算	(78)
3.1.2 移位运算	(81)
3.1.3 算术运算	(86)
3.2 定点加(减)法运算	(88)
3.2.1 二进制加(减)法运算	(88)
3.2.2 二进制补码加法器	(92)
3.2.3 多功能算术逻辑运算单元 ALU	(96)
3.2.4 十进制加法器	(101)
3.3 定点乘法运算	(104)
3.3.1 原码一位乘法	(104)
3.3.2 补码一位乘法	(109)

3.3.3 快速乘法	(115)	4.5.2 微指令顺序控制	(214)
3.4 定点除法运算	(126)	4.6 微指令时序控制	(219)
3.4.1 原码一位除法	(126)	4.6.1 微指令周期	(220)
3.4.2 补码一位除法	(131)	4.6.2 微指令周期多相控制	(220)
3.4.3 快速除法	(140)	小结	(221)
3.5 浮点运算方法	(144)	习题	(221)
3.5.1 浮点加(减)运算	(144)	第五章 存储器	(223)
3.5.2 浮点乘法运算	(147)	引言	(223)
3.5.3 浮点除法运算	(151)	5.1 存储器概论	(224)
3.6 运算器组织	(153)	5.1.1 存储器的分类	(224)
3.6.1 运算器的基本结构	(153)	5.1.2 内存的主要技术指标	(228)
3.6.2 运算器组成实例	(157)	5.2 内存储器的工作原理	(229)
3.6.3 位片式运算器	(159)	5.2.1 内存储器的基本组成及工作 过程	(230)
3.6.4 浮点运算器	(160)	5.2.2 随机存取存储器 RAM	(233)
小结	(163)	5.2.3 RAM 存储芯片	(237)
习题	(163)	5.2.4 只读存储器 ROM	(247)
第四章 控制器	(166)	5.3 按内容寻址存储器 CAM	(259)
引言	(166)	5.3.1 概述	(259)
4.1 指令结构	(167)	5.3.2 CAM 存储位元电路	(260)
4.1.1 实例计算机的指令系统	(168)	5.3.3 CAM	(261)
4.1.2 实例计算机指令的含义	(169)	5.4 内存储器的设计	(263)
4.2 控制器的基本设计技术	(172)	5.4.1 设计内存储器的一般原则和 方法	(263)
4.2.1 控制器的基本组织	(172)	5.4.2 内存储器的逻辑设计	(264)
4.2.2 基本控制器的设计	(175)	5.4.3 内存储器逻辑设计举例	(267)
4.3 指令流控制和复杂指令的设计	(187)	5.4.4 DRAM 刷新	(275)
4.3.1 计算机指令流的控制	(187)	5.5 磁表面存储器	(279)
4.3.2 局部控制周期技术	(191)	5.5.1 磁记录原理	(279)
4.3.3 微操作控制信号组合逻辑网络的 实现	(195)	5.5.2 磁表面存储器的主要技术 指标	(283)
4.3.4 组合逻辑控制器的设计 过程	(197)	5.5.3 磁盘存储器	(287)
4.4 微程序控制器技术	(200)	5.6 光盘存储器	(297)
4.4.1 微程序控制基本原理	(200)	5.6.1 光盘记录介质	(297)
4.4.2 Wilkes 模型	(201)	5.6.2 光盘分类	(298)
4.4.3 实例计算机的微程序实现	(205)	5.6.3 光盘读写原理	(300)
4.5 微程序的技术问题	(209)	5.7 计算机的存储系统	(303)
4.5.1 微命令控制信号编码与微指令 格式	(210)	5.7.1 并行主存系统	(303)

5.7.2 存储系统及其层次结构	(307)	6.6.2 通道命令字和通道程序	(366)
小结	(310)	6.6.3 通道传送的工作过程	(370)
习题	(310)	小结	(371)
第六章 输入/输出(I/O)控制	(314)	习题	(371)
引言	(314)	第七章 计算机模块结构与互连	(373)
6.1 外围设备简介	(314)	引言	(373)
6.1.1 外设分类	(314)	7.1 模块结构与互连	(373)
6.1.2 外设的地位与作用	(316)	7.1.1 计算机的模块结构	(373)
6.1.3 外设与主机的连接方式	(317)	7.1.2 常见互连结构	(374)
6.2 I/O 控制的有关问题	(317)	7.2 总线系统概述	(375)
6.2.1 I/O 设备与 I/O 操作特点	(318)	7.2.1 总线的基本组成	(376)
6.2.2 四级 I/O 子系统和三级 I/O 子 系统	(319)	7.2.2 总线系统的基本工作原理	(377)
6.2.3 I/O 控制的类型	(320)	7.2.3 总线的分类	(378)
6.2.4 I/O 接口	(321)	7.2.4 总线的特点和性能	(379)
6.2.5 I/O 指令	(325)	7.3 总线系统的结构	(380)
6.3 程序直接控制传送	(327)	7.3.1 总线设备	(380)
6.3.1 无条件传送方式	(327)	7.3.2 总线设备接口	(381)
6.3.2 程序查询传送方式	(327)	7.3.3 总线控制器	(382)
6.4 中断及程序中断控制传送	(330)	7.3.4 总线的连接方式	(383)
6.4.1 中断的有关问题	(330)	7.4 总线设计要素	(386)
6.4.2 中断系统的结构组成	(333)	7.4.1 总线宽度	(386)
6.4.3 程序中断控制传送及其 接口	(342)	7.4.2 总线的复用方式	(387)
6.4.4 可编程中断控制器 8259A 简介	(346)	7.4.3 总线定时方式	(388)
6.5 直接存储器访问 DMA	(349)	7.4.4 总线仲裁	(392)
6.5.1 DMA 概述	(350)	7.4.5 总线数据传送方式	(399)
6.5.2 DMA 接口的基本组成	(352)	7.5 PCI 总线标准	(404)
6.5.3 DMA 的工作过程	(354)	7.5.1 PCI 总线概述	(404)
6.5.4 DMA 传送接口类型	(356)	7.5.2 PCI 总线信号	(406)
6.5.5 DMA 传送举例	(359)	7.5.3 PCI 总线命令	(409)
6.6 I/O 通道	(361)	7.5.4 PCI 总线仲裁	(410)
6.6.1 I/O 通道概述	(361)	7.5.5 PCI 总线数据传送方式	(412)
		小结	(414)
		习题	(415)
		参考文献	(416)

第一章 絮 论

引 言

当今世界,信息化、数字化、电脑化的浪潮汹涌澎湃,人们无时无处不在感受着电脑带来的魅力和风采。电脑就是计算机,计算机是信息加工和处理的工具,它具有运算速度快、运算精度高、记忆功能强、通用性广等特点,能够自动地完成各种复杂的计算。如果说人类制造的其他工具是人类四肢五官的延伸,那么用计算机代替人脑进行信息加工与处理,则可以说是人类大脑的延伸。因此,人们习惯称计算机为电脑,确也恰如其分。

作为计算机专业的学生和热衷于探寻计算机奥秘的读者,自然会有这样的疑问,计算机到底是如何构成和怎样工作的?本书就是要系统地探讨计算机的基本结构、工作原理和设计方法。作为开篇,本章将介绍:计算机发展史、计算机系统组成、计算机的工作过程、计算机的性能指标及分类、以及计算机的应用与发展,使读者对计算机先有一个大体了解,并为后续章节的学习奠定基础。

1.1 计算机的发展史

1.1.1 计算简史

人类社会的发展历史,始终是伴随着计算工具的产生、应用及发展的历史。人类通过劳动和智慧创造了工具,包括机械工具、测试工具和计算工具。人们用机械工具延伸扩展了四肢的功能;用测试工具延伸扩展了五官的功能;用计算工具延伸扩展了大脑的功能。

人类从记数、计数到计算,经历了漫长的历史阶段,而计算则经历了从手工计算阶段、机械计算阶段,一直发展到现今的电子计算阶段。

1. 手工计算阶段

远古时代,人类只能通过穴石、结绳和刻木等简单方法记载发生过的事件。当它们所代表的具体事件无法分辨时,留下的只能是事件多少的记录。因此,穴石、绳结、刻痕只能用于记数,它们是记数工具。

在与自然界的斗争中,人们不能满足于对数的简单记录,迫切需要对数进行比较,即进

行计算。人们发现,十指是最方便、最简单的计算工具。采用十指对数进行度量,产生了十进制计数法,这是一大飞跃。它延伸扩展了大脑的计算功能,以至于现今还被不少人用作计算的工具,而十进制计数法更是今天数学体系的计数制基础。

据史料记载,中国在公元前 5~6 世纪就已经出现了算盘,这是人类应用时间最长、功能最完善的非自然化计算工具。人们不但制作了各种各样精巧美观的算盘,还形成了一整套的运算口诀和操作方法。口诀是针对算盘的结构特点设计的基本操作命令,用现代计算机的术语,它就是算盘的指令系统或珠算语言。对不同的计算操作,可使用该指令系统的不同序列。算盘的发明是人类计算工具史上的一次大飞跃,是中华民族对人类文明的重大贡献之一。算盘先后传至日本、朝鲜、东南亚,后来传至欧洲。迄今为止,它仍是人类使用最多、最有效、价格最低廉的计算工具之一。

1621 年,计算尺问世。这种可滑动的尺子是 20 世纪 50~60 年代的工科大学生所必备的计算工具。它用长度来模拟数值的大小,因此是一种模拟计算工具,它除了可以进行一般的四则运算外,还可进行一些复杂的非四则运算。

从穴石、绳结、刻痕、十指到算盘及计算尺,都是一种手工计算。由于人参与计算过程,故人的技能水平、精力、智力都直接影响计算的速度和正确性。而且,手工计算对许多大型复杂问题的计算往往显得无能为力。

2. 机械计算阶段

1642 年,法国人 Pascal 发明了可做加减法的机械计算器。1673 年,德国人 Leibnitz 改进了 Pascal 的设计,增加了乘除运算。由于生产技术水平的限制,直到 19 世纪,手摇计算机才得以商品化生产。

1812 年,英国人 Babbage 首先提出了整个计算过程自动化的概念,设计了第一台通用自动时序控制机械式计算机。Babbage 在他的后半生花费了大部分精力和财力来制造这台计算机,可惜的是,由于当时技术水平的限制,这台计算机未能制造出来。Babbage 认为自动计算机必须具有输入、输出、处理、存储、控制五大功能。另外,他还提出计算机只有具有记忆功能,能记住数据和要进行的操作步骤,并按这些步骤规定对机器进行自动控制,才能实现自动计算。这些思想的确是对现代电子数字计算机的伟大贡献。因此,人们称 Babbage 为“计算机之父”。

机械式计算工具虽然在这一时期取得了较大的成功,但仍要由人按照一定的步骤进行操作。从提供操作数,到选择操作,得到计算结果,整个过程中频繁的人工干预限制了计算速度的提高。

3. 电子计算阶段

电子计算的理论基础是二进制。中国是二进制理论的最早发明者。古人使用的符号为“爻(yáo)”,爻分阳爻和阴爻,可以说阳爻对应二进制中的 1,阴爻对应二进制中的 0。著名的《易经》将八卦和六十四卦分别用 3 个爻和 6 个爻的组合表示。Leibnitz 曾致信清朝康熙

皇帝说“伏羲在其推演的八卦中使用了二进制算术”。遗憾的是,这种二进制算术在中国未得到重视与发展,更谈不上应用到计算工具中去了。

1854 年,英国数学家 George Boole 发表了《逻辑的数学分析》和《思维规律的研究》两部著作。两部著作的核心,就是现代的《布尔代数》的精髓。它把运算和逻辑理论建立在“0”和“1”两种数值以及“与”、“或”、“非”三种基本逻辑运算的基础上。布尔代数为二进制的数字计算机奠定了理论基础,也是现代所有数字式设备的理论基础。

20 世纪电子技术和电子器件的不断发展为电子计算机的诞生与发展铺平了道路。

1939 年,美国依阿华大学教授 V. Atanasoff 首次试用电子元件按二进制原理制造了一台电子管计算机。1942 年,又在研究生 Clifford Berry 的协助下制造出了一台电子管计算机 ABC(Atanasoff Berry Computer)。

1943 年,美军缴获了德军著名的 Leopold 重炮。为研究这种重炮,美军军械部每天要向阿伯丁试验场提供 6 张火力表。一张火力表约需 3 000 个不同的弹道参数,每一个弹道参数都需用几个不同的微分方程来计算。就单个参数而言,飞行时间为 60 s 的弹道,用台式计算器计算需要 20 小时;即使采用当时新式的微分分析仪计算,也需要 20 分钟。为了提高计算速度,减少计算时间,美军军械部求助于宾夕法尼亚大学的摩尔电机学院,希望由他们研制新的计算机。1945 年春天,世界上第一台电子数字计算机 (Electronic Numerical Integrator and Computer, ENIAC) 开始试运行。用这台机器计算弹道参数,60 s 射程的弹道计算,由原来的 20 分钟缩短为 30 s!

从现在的观点看来,ENIAC 实在是一个庞然大物。它高 8 英尺,宽 3 英尺,长 100 英尺,装有 18 000 个真空管,1 500 个电子继电器,70 000 个电阻,18 000 个电容,总重 30 吨。它的性能也实在不算高:每秒 5 000 次加法运算,每秒 50 次乘法运算,可进行平方、立方、 \sin 和 \cos 函数数值运算等。

除了常规的弹道计算外,ENIAC 的工作还涉及到天气预报、原子核能、风洞实验等诸多领域。著名数学家冯·诺依曼 (Von. Neumann) 邀请了研制原子弹的学者在这台机器上进行了有关原子裂变的能量计算,可以说 ENIAC 为世界上第一颗原子弹的早日问世立下了汗马功劳。1949 年,ENIAC 经过 70 小时的计算,将圆周率计算到小数点后 2 037 位,这是人类第一次用机器计算出来的最精确的圆周率数值。

ENIAC 于 1955 年退休。10 年间它运行了 8 023 小时,它的算术运算量比有史以来人类大脑所有运算的总和还要大得多。更重要的是,ENIAC 是一个划时代的创举,它成为现代电子数字计算机的始祖。1949 年,英国剑桥大学开发的 EDSAC (Electronic Delay Storage Automatic Computer) 是世界上第一台通用电子数字计算机。从此,人类开始进入了电子计算的电子数字计算机时代。

1.1.2 计算机发展简史

从 ENIAC 算起,计算机的发展至今不过半个多世纪。然而,计算机发展之迅猛,普及应

用之广泛,对人类社会影响之深远,是历史上任何学科所无法比拟的。迄今为止,国际公认的计算机发展经历了四代。在推动计算机发展的诸因素中,电子器件是划分时代最重要的标志。

第一代计算机(20世纪40年代中期到50年代末期):以电子管作为逻辑元件,用阴极射线管、声汞延迟线、磁带和磁鼓等作为主存储器,数据主要是定点表示,用机器语言或汇编语言编写程序。除了前述的ENIAC和EDSAC外,具有代表性的机器还有以冯·诺依曼为首研制的存储程序计算机IAS、UNIVAC-1和IBM 704,以及我国自己研制的第一代计算机,主要104机、103机和119机等。

第二代计算机(20世纪50年代中、后期到60年代中期):以晶体管作为逻辑元件,磁芯作为主存储器元件,引入浮点运算硬件;建立了子程序库和批处理的管理程序;配备了FORTRAN、COBOL和ALGOL等高级语言,从而大大简化了程序设计。晶体管计算机体积小,功耗低,速度快,可靠性高。具有代表性的机器有IBM 7040、7070、7090,CDC 1604等。国产晶体管计算机的代表有109机、441B机和108机等。

第三代计算机(20世纪60年代中期到70年代中期):以集成电路作为基础器件,这是微电子技术与计算机技术相结合的一大突破。主要采用小规模集成电路和中规模集成电路,半导体存储器逐渐取代磁芯存储器;引进了多道程序和并行处理等新的技术,操作系统日趋成熟。具有代表性的机器有IBM 360系列、CDC 6600/7600系列和CYBER系列等。国产机器代表有150、151、DJS-2000系列和DJS-1000系列等。

第四代计算机(20世纪70年代中期至今):特点是采用大规模集成电路和超大规模集成电路;并行处理、多机系统、分布式计算机、计算机网络等技术迅速发展;各种高级语言、分布式操作系统、数据库技术竞相争艳。更重要的是微处理器、微型计算机得到迅速发展,各种小型机、超级小型机、大型机、巨型机不断问世。这时期的机器类型和代表真可谓是百花齐放,异彩纷呈。计算机进入了空前繁荣的时代。

1.2 计算机系统组成

1.2.1 存储程序原理

现代电子计算机的组成原理均是依据美籍匈牙利数学家冯·诺依曼等发表的题为《关于电子计算装置逻辑结构初探》报告中所阐述的思想构建的。在此报告中,冯·诺依曼提出了以存储程序为核心的通用电子数字计算机体系结构原理,从而奠定了当代电子计算机体系结构的基础。通常,按照存储程序原理构建的计算机被称为“存储程序计算机”。

存储程序原理的基本思想是:计算机要自动完成解题任务,必须将事先设计好、用以描述计算机解题过程的程序(程序即指令的有序集合),和数据一样,采用二进制形式存储在机

器内部,计算机在工作时自动高速地从机器中逐条取出指令并加以执行。存储程序是计算机能自动工作的关键所在。计算器不同于计算机之处在于,计算器的解题步骤即程序是在执行过程中由人工临时编制和控制执行的。

按照存储程序原理,计算机必须具有五大功能:

(1) 数据传送功能。计算机必须有能力将原始数据和解题程序输入到机器中,而计算的结果和计算过程中出现的情况也能随时输出给用户。这也就是说,计算机必须具有输入和输出功能。

(2) 数据存储功能。计算机应能“记住”输入的原始数据和解题步骤即程序,以及解题过程中产生的一些中间结果,即具备数据存储功能,这是计算机能实现自动运算的关键。

(3) 数据处理功能。计算机应能进行一些最基本的运算,从而组合成人类所需要的一切复杂的运算和操作。这是计算机进行运算、处理和控制的基础。

(4) 操作控制功能。计算机应能保证程序执行的正确性,应能对组成计算机的各部件进行协调和控制。

(5) 操作判断功能。在完成一步操作后,计算机应能从预先无法确定的几种方案中选择一种方案,从而保证解题操作正确完成。

虽然计算机种类繁多,运算处理能力有强有弱,但作为计算机,它们的基本功能却都是类似和相同的。即存储程序计算机都应当具有数据传送、数据存储、数据处理、操作控制和操作判断这五大功能。

经典的“存储程序计算机”,即经典冯·诺依曼结构计算机框图如图 1.1 所示。图中实线为数据线,虚线为控制线和反馈线。

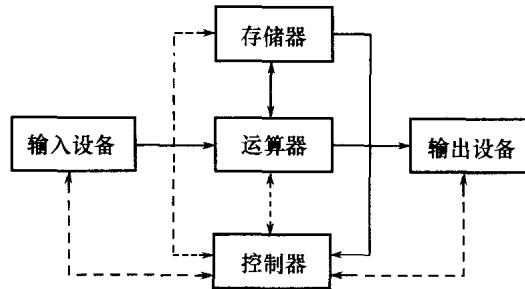


图 1.1 经典冯·诺依曼计算机框图

1.2.2 计算机系统的组成

虽然计算机类型档次多,功能差别大,但它们的基本结构却是类似的。完整的计算机系统均由硬件系统和软件系统两部分构成。

1. 计算机的硬件系统

硬件(Hardware)是组成电子计算机的所有电子器件和机电装置的总称。它是计算机系统中实际存在的物理实体,是看得见摸得着的。从物理构成上看,硬件系统是由电子器件、插件板、电源、机架和各种功能的外部设备组成。从逻辑功能看,硬件系统是由运算器、控制器、存储器、输入设备和输出设备以及它们之间的互连结构组成。

2. 计算机的软件系统

软件(Software)系统是相对于硬件系统而言的,它是看不见摸不着的。要使用计算机,

就必须编制程序,必须有软件。按照国际标准化组织(ISO)的定义,软件是电子计算机程序及运用数据处理系统所必需的手续、规则和文件的总称。因此,一般认为软件是由程序与文档两部分组成。程序(Program)是为了取得一定的结果而编写的计算机指令的有序集合;文档(Document)则是描述程序操作及使用的有关资料。程序可由计算机执行,而文档则不能由计算机执行。程序是计算机软件的主体,故一般说到软件,主要是指程序。

3. 计算机软件系统和硬件系统的关系

软件作为计算机用户与硬件之间的接口界面,在计算机系统中起指挥管理作用。硬件与软件的组合构成了完善实用的计算机系统。硬件是躯体,是物质基础;软件是智慧,是灵魂,是硬件功能的完善与扩充。没有硬件,或者没有良好的硬件,就无从谈起运行软件,也就无法计算、处理某一方面的问题。没有软件,或者没有优秀的软件,计算机就是一个空壳,根本无法工作,或者不能高效率地工作。因此,硬件与软件是相互渗透,相互依存,互相配合,互相促进的关系,二者缺一不可。

应当指出的是,计算机系统的功能由硬件或软件实现,软件和硬件在逻辑功能上是等价的。也就是说,用硬件实现的功能,在原理上可以用软件实现;同样,用软件实现的功能,在原理上也可由硬件完成。例如,完成乘法运算,既可用硬件乘法器实现,也可用乘法子程序实现。软硬件功能究竟怎样分配,这涉及到系统的成本和速度等诸多问题。由硬件或软件实现的计算机系统的成本、效率等是不同的,一般在系统设计时应加以权衡。

计算机系统的软件与硬件可互相转化,随着超大规模集成电路技术的发展,软件硬化或固化已经是提高计算机处理能力的最常用手段。例如,将本来由编译器来识别的指令之间并行性这个功能由硬件完成,就实现了现代计算机的指令动态调度功能。在单片机中,将程序(如 BASIC 解释程序)固化在只读存储器 ROM 中,装入机器,就可以随用随取。这种将程序固化在 ROM 中组成的部件称为固件(Firmware)。固件是一种具有软件特性的硬件,既具有硬件的快速性,又具有软件的灵活性。

1.2.3 计算机硬件系统组成

硬件系统是构成计算机的物质基础,是计算机系统的核心。根据存储程序计算机的五大功能,计算机硬件系统由运算器、控制器、存储器、输入设备和输出设备以及将它们连结为有机整体的互连结构组成。图 1.1 表示的是早期经典的冯·诺依曼计算机结构,它以运算器为中心。由于输入/输出设备速度慢,并且传输操作都需通过运算器,导致计算机效率不高。现代计算机结构已转化为以存储器为中心,如图 1.2 所示。

由于运算器、控制器在逻辑关系上联系紧密,尤其在大规模集成电路出现后,两部件又制作在同一芯片上,故将它们统称为中央处理器(Central Processing Unit, CPU)。存储器用来存储二进制信息,通常粗分为两大类。一类称为内存储器,又称为主存储器,它的存取速度快,存储容量较小,与 CPU 直接打交道,由半导体元器件构成;另一类为外存储器,亦称辅

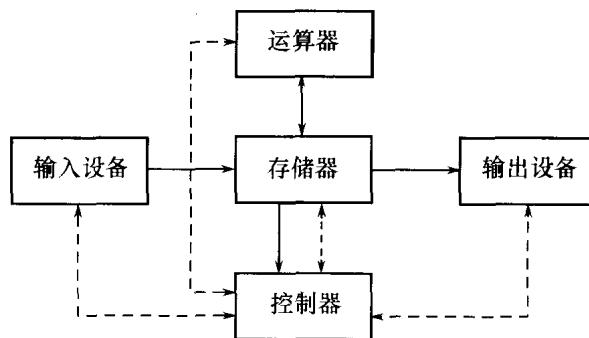


图 1.2 现代计算机结构框图

助存储器,如磁盘、磁带和光盘等,它的存取速度慢,存储容量较大;但不能与 CPU 直接打交道,必须将其中的信息送到内存后才能由 CPU 处理。图 1.2 中的存储器指的是内存储器。CPU、内存储器合起来又称主机。输入/输出设备和外存储器又被统称为外部设备。下面分别介绍这些部件的组成。

1. 运算器

运算器(Arithmetic Unit)是进行数据处理即执行算术运算和逻辑运算的部件。算术运算是按照算术规则进行的运算,如加、减、乘、除及它们的复合运算等。逻辑运算是非算术性运算,如比较、移位、逻辑加、逻辑乘以及异或操作等。带输入锁存器的运算器由算术逻辑单元、内部总线、通用寄存器组、锁存器、标志寄存器和移位器组成,如图 1.3 所示。

(1) 算术逻辑单元(Arithmetic and Logic Unit, ALU): ALU 是具体完成算术逻辑运算的部件,由加法器及逻辑运算器件组成,它是运算器的核心。

(2) 内部总线: ALU、通用寄存器组、锁存器、标志寄存器 Flag 和移位器均通过它进行连接,它是 CPU 内部的数据通路。与系统总线不同,内部总线仅用来传送数据。

(3) 通用寄存器组 $R_0 \sim R_{n-1}$: 用于存放参与运算的操作数,如:加数、减数、乘数、被除数、积或者商等。在连续运算中,还用于存放中间结果和最终结果。通用寄存器中的数据均从存储器中取得,最终结果也要存放到存储器中。

(4) 锁存器: ALU 的两个输入端各有一个锁存器,用来暂存要参与运算的操作数。如要实现 $R_0 \leftarrow (R_0) + (R_2)$, 可通过内部总线先将 R_0 中的数据送入锁存器 1, 再通过内部总线将

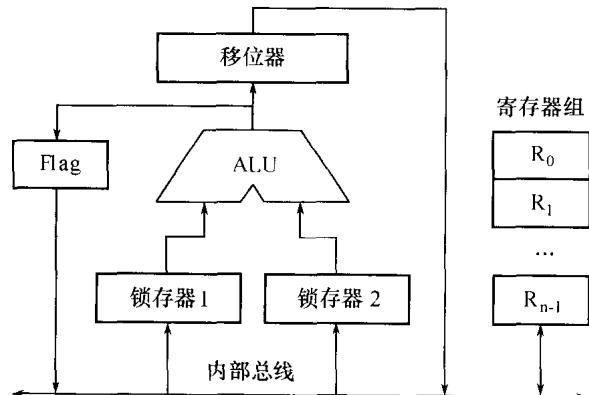


图 1.3 带输入锁存器的运算器示意图

R_2 中的数据送入锁存器 2, 然后再在 ALU 中相加, 最后将结果通过内部总线送入 R_0 。

(5) 标志寄存器 Flag: 寄存运算的状态信息, 例如结果是否为“0”、是正数还是负数、溢出与否等。

(6) 移位器: 是一个多路开关, 可对 ALU 的加工结果再进行辅助操作, 例如可实现左移、右移和直接传送等功能。

除带输入锁存器的运算器外, 还有带多路选择器的运算器, 它们的基本组成相同, 主要区别是通用寄存器与 ALU 的连接方式不同。关于运算器的详细内容将在第三章中介绍。

2. 控制器

控制器 (Control Unit) 是计算机的管理机构和指挥中心, 它协调计算机的各部件自动地工作。控制器完成的工作实质上就是解释程序, 它每次从存储器中读取一条指令, 经过分析译码, 产生一系列的控制信号, 发往各个部件以控制它们的操作。连续不断、有条不紊地继续上述动作, 即所谓执行程序。

计算机有两种类型的控制器, 即组合逻辑控制器和微程序控制器。组合逻辑控制器的结构主要由指令控制部件、地址形成部件、定时部件及微操作控制部件组成, 如图 1.4 所示。

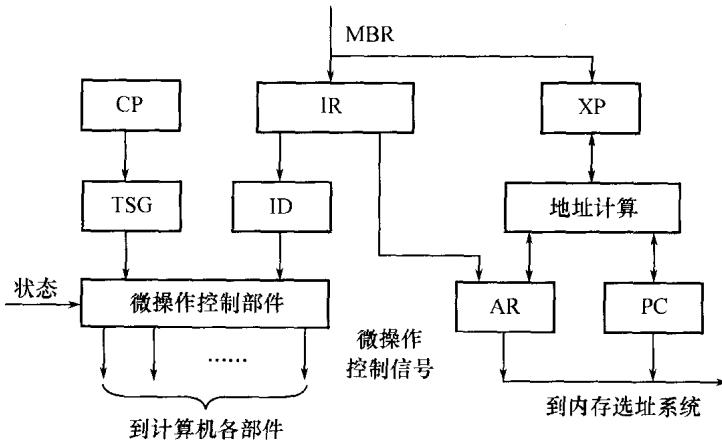


图 1.4 控制器的组成

(1) 指令控制部件: 由程序计数器 (Program Counter, PC)、指令寄存器 (Instruction Register, IR) 和指令译码器 (Instruction Decoder, ID) 组成。

程序计数器 PC 又叫指令计数器。它给出程序中指令在存储器中的单元地址。它兼有指令地址寄存器和计数器的功能。取指令时, PC 指向要取指令的地址。当一条指令执行完毕时, PC 作为指令地址寄存器, 其内容已变成下一条指令的地址。控制器依据 PC 的内容从存储器取出指令送到 IR 后, PC 将自动加 1, 指向下一条指令的地址。注意, 这里假定一个存储单元存储一条指令。若非顺序执行, 只要将 PC 内容作相应改变, 就可按新的序列顺序执行。指令寄存器 IR 保存当前正在执行的指令代码, 在指令执行过程中由它决定指令的操作

性质及参与操作的操作数地址等。ID 也叫操作码译码器,它将指令的操作码转换为相应的控制电位信号,指示各部件做什么操作。

(2) 地址形成部件:包括地址寄存器 (Address Register, AR)、变址寄存器 (Index Register, XR) 和地址计算部件。其功能主要是依据指令的寻址方式和指令的地址码部分生成实际的操作数地址。

(3) 定时部件:亦称时序部件,由时钟 (Clock Pulses, CP) 和时序信号产生器 (Timing Signal Generator, TSG) 组成。CP 是协调计算机各部件进行操作的同步时钟,其工作频率称为计算机的主频。主频的高低直接影响计算机的工作速度。TSG 的功能是按时间顺序,周而复始地发出若干节拍信号和脉冲。节拍即相邻两个时钟脉冲的时间间隔,即主频周期。定时部件就是根据机器的时钟脉冲,发出全机所需的定时节拍信号和脉冲。各部件在不同的节拍信号控制下依次进行工作。

(4) 微操作控制部件:微操作 (Micro-operation) 是指计算机各部件在一个节拍内能完成的基本操作。计算机任意一条指令的执行都需分解成许多微操作来执行。微操作控制部件是控制器的核心部件,也是控制器中最复杂的部件。它根据指令控制部件给出的指令译码电位信号(进行什么操作)、时序部件给出的节拍信号和脉冲(指令执行到哪一步)和运算器 Flag 提供的状态信息,产生计算机指令系统中所有指令所需的各种微操作控制信号。然后再将这些控制信号发送给运算器、存储器、输入/输出设备以及控制器本身。

微程序控制器除了将微操作控制部件用微程序库代替外,其他部件和组合逻辑控制器相似。有关控制器的详细内容将在第四章中介绍。

3. 内存储器

电子计算机的特点之一是具有存储记忆功能,理所当然应该有存储信息的装置——存储器 (memory/storage)。存储器的主要功能是存放数据和程序。程序是计算机指令的有序集合,数据是计算机操作的对象,它们均以二进制的形式表示。为了实现自动运行程序,数据和程序必须存放在内存储器中。内存储器由存储体、选址系统、读写系统和存储时序控制线路构成,如图 1.5 所示。

(1) 存储体:存储体就像一个庞大的仓库,它被分成一个一个的存储单元,每个存储单元存放多位二进制信息。信息的位数通常是计算机的字长,一般字长为字节的整数倍。通常每个单元存放一个数据或一条指令。存储单元的集合称之为存储体。存储单元按顺序编号,每个存储单元对应一个编号,此编号称为存储单元地址,简称地址。只要给定一个地址,就可通过地址译码器译码,找到对应的存储单元,从而可从该单元读取信息,或将信息写入该单元。注意,地址与存储单元是一一对应的,每个存储单元只有一个地址。

(2) 选址系统 (Addressing System):由存储地址寄存器 (Memory Address Register, MAR)、地址译码器和地址驱动器三部分组成。CPU 或 I/O 与内存储器交换信息是以存储单元为单位的。向存储单元存入或从存储单元取出信息,称之为访问存储器。I/O 或 CPU 访

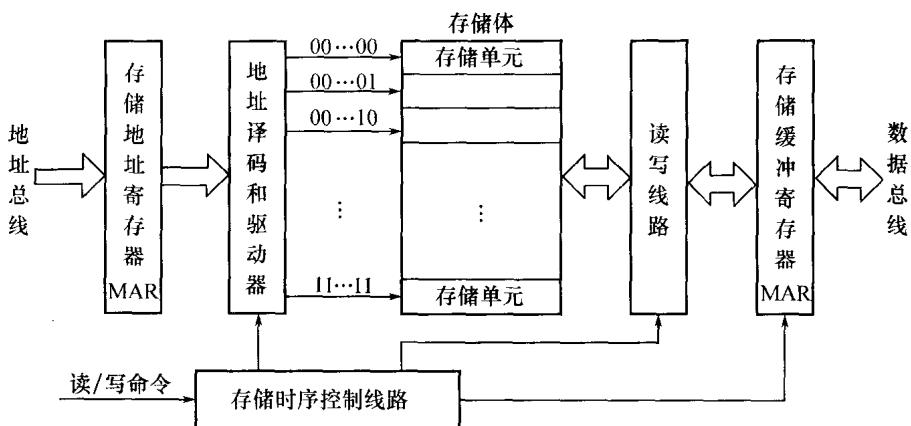


图 1.5 内存储器构成

存时,先将访存地址送 MAR,经地址译码器找到被访问的存储单元,最后由地址驱动器驱动该存储单元以实现读或写。

(3) 读写系统(Read and Write System):包括存储缓冲寄存器(Memory Buffer Register, MBR)和读写线路。读出时,控制器发出读控制信号,借助于读出线路将由选址系统确定的存储单元内容读出送至 MBR,以供 I/O 或 CPU 使用。写入时,先将要写入的数据送至 MBR,控制器发出写控制信号,借助于写入线路,将 MBR 内容写入由选址系统确定的存储单元。

(4) 存储时序控制线路(Memory Sequential Control Circuit):由控制触发器,各种门电路和延迟线路等组成。接收来自 I/O 或 CPU 的启动、读写、清除等命令,产生一系列控制内存完成读写等操作的信号。

关于存储器的详细内容将在第五章中介绍。

4. 输入设备和输出设备

输入设备(Input Equipment/Device):其作用是将数据、程序送入计算机。常见的输入设备有键盘、鼠标、数字化仪和扫描仪和摄像机等。它们多是电子和机电混合的装置,与运算器、控制器和内存储器等纯电子部件相比,速度较慢。因此,一般需通过被称之为接口(Interface)的电子部件与运算器、控制器和存储器相连接。

与输入设备正好相反,输出设备(Output Equipment/Device)是将计算处理的结果转化为人或其他设备所能识别或接收的信息形式的装置。例如,显示器能将信息转化为字符、汉字、图形、图像,并在屏幕上显示;打印机能将文件打印出来;绘图机可将图形绘制出来,等等。和输入设备一样,输出设备也多为机电装置,也需通过接口与运算器、控制器和存储器相连接。

关于输入/输出设备简介及输入/输出接口的详细内容将在第六章中介绍。

5. 互连结构

计算机的五大部件需通过互连结构(Interconnection Structure)连接成一个有机的整体,

现代计算机通常采用总线(BUS)互连结构。总线是连接计算机各部件进行信息传送的一组公共传输线,其类型和结构形式繁多。关于互连结构的详细内容将在第七章中介绍。

1.3 计算机的工作过程

本节通过一个简单例子,用机器语言编制程序,并结合计算机整机结构来说明计算机各组成部件是怎样相互配合实现自动解算问题的,以期使读者建立初步的整机概念,便于以后各章的学习。

1.3.1 使用计算机求解的一个简单例子

采用现代计算机系统求解问题时,一般遵循如下步骤:首先由用户提出任务并建立数学模型,其次是要确定便于计算机实现的算法,然后是选择合适的语言编写程序,最后是上机调试运行。无论采用何种程序设计语言编写源程序,都必须转换为机器语言程序,即目标程序,然后计算机才能执行。机器语言程序是机器指令(简称指令)的有序集合。计算机的工作过程,就是执行指令的过程。

1. 一个简单例子

例 1.1 用计算机求解代数多项式: $y = ax^3 + bx^2 + cx + d$,其中 a, b, c, d, x 为已知数。

对该代数多项式可直接进行四则运算:先求出 x^2 和 x^3 ,然后计算每项乘积 ax^3 、 bx^2 和 cx ,最后再两两相加求得 y 值。此算法需要 5 次乘法和 3 次加法,共 8 次运算。

如果将多项式的幂、乘加形式变换为乘、加迭代形式,即: $y = ((ax + b)x + c)x + d$,其计算步骤如图 1.6 所示。此时算法需要乘、加各 3 次,共 6 次运算。

可见,在确定计算步骤时,仔细分析和简化算法很重要。在计算多项式时,将其化为乘、加迭代的形式通常是比较好的算法。图 1.6 中的算法将 8 次运算减少到 6 次,使运算次数减少了四分之一,效果比较显著。

2. 指令与指令系统简介

有了算法就可编写机器语言程序了。机器语言程序是指令的有序集合,因此有必要先简单介绍一下指令和指令系统。关于指令系统的详细内容将在第二章中介绍。

指令是计算机设计者赋予计算机实现某种基本操作的命令。指令一般包括两个字段。一个称为操作码字段,指出计算机要执行什么类型的操作,即决定了指令的功能;另一个称为地址码字段,指出参与操作的操作数或者操作数的地址,即规定了指令的

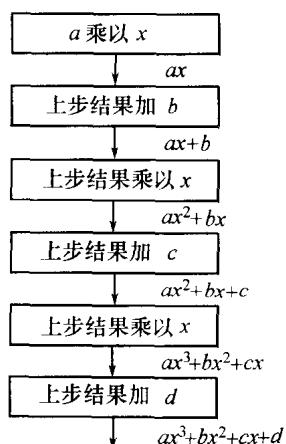
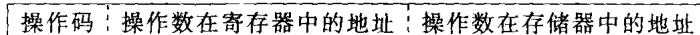


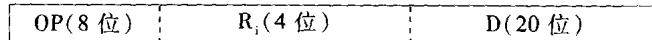
图 1.6 计算流程图

操作对象。地址码字段的操作对象可以是零个到多个。操作码和地址码构成了指令字。例 1.1 中,运算的操作对象有两个,一个为寄存器中的操作数,另一个为存储器中的操作数,能够完成计算任务的指令字格式为:



指令系统包含一台计算机所能执行的所有指令,它表征着计算机的基本功能和使用属性。计算机只能执行自己指令系统中的指令。设计和研制计算机时首先应当确定指令系统并给出指令格式。为保证问题能够求解,指令系统应包括算术运算、数据传送、以及处理机控制等多种类型的指令。

假定在某 32 位字长,设有 16 个通用寄存器的计算机上进行例 1.1 要求的计算,其指令格式如下:



OP:操作码,占 8 位,表示该机指令系统中最多包含 256 条指令。这些指令包括算术运算(例如加、乘)、数据传送(例如存数、取数)、以及处理机控制(例如停机)等指令。

R_i:操作数在寄存器中的地址,指向某通用寄存器,用 4 位表示。16 个通用寄存器的地址(编号)可以分别用 16 进制的 0 ~ F 来表示。

D:操作数在存储器中的地址,指向内存储器某存储单元,用 20 位表示。这表示该计算机最多能够访问 2²⁰ 个存储单元。

R_i 和 D 共同构成了该计算机的地址码。OP、R_i 和 D 构成了计算机的指令字。

为完成例 1.1 的工作,该计算机提供了 5 条指令,它们的操作码和指令功能见表 1.1。

表 1.1 某计算机五种操作的指令操作码及功能

操作	操作码	指令功能符号表示	指令功能说明
加法	02H	R _i ← R _i + (D)	通用寄存器 R _i 中内容与 D 存储单元内容相加之和送 R _i
乘法	08H	R _i ← R _i × (D)	通用寄存器 R _i 中内容与 D 存储单元内容相乘之积送 R _i
取数	10H	R _i ← (D)	将 D 存储单元内容读出送运算器的通用寄存器 R _i
存数	14H	D ← R _i	将运算器的通用寄存器 R _i 内容存入 D 存储单元
停机	FFH	HLT	停机

对同一条指令的表示通常可以使用二进制和十六进制两种表示方法。在计算机中存储的指令为二进制形式,采用十六进制只是方便人们的书写和阅读,它们的含义是相同的。例如有以下指令:

二进制表示	0000 0010 1000 0000 0100 0000 0000 0000
十六进制表示	02 8 04000

根据表 1.1 知,该指令表示将寄存器 R_8 中的数据与地址为 04000H 的存储单元中的数据做加法,结果放入寄存器 R_8 中。

3. 编写求 y 的机器语言程序

编写机器语言程序之前,必需进行存储单元分配,编写求 y 的机器语言程序也不例外。假定将初始数据 x, a, b, c, d 分别存放在地址为 00B00H ~ 00B04H 的存储单元中,计算结果 y 存储到地址为 00B05H 的存储单元。程序存放在以 00A00H 为始地址的存储单元中。

依计算流程图 1.6 和表 1.1 提供的指令编写机器语言程序如表 1.2。表中的程序代码和地址码均为 16 进制数。

表 1.2 计算 y 的机器语言程序

主存地址	指令或数据			说 明	
00A00	10	8	00B01	取数: $R_8 \leftarrow a$	
00A01	08	8	00B00	乘法: $R_8 \leftarrow a \times x$	
00A02	02	8	00B02	加法: $R_8 \leftarrow a \times x + b$	
00A03	08	8	00B00	乘法: $R_8 \leftarrow (a \times x + b) \times x$	
00A04	02	8	00B03	加法: $R_8 \leftarrow (a \times x + b) \times x + c$	
00A05	08	8	00B00	乘法: $R_8 \leftarrow ((a \times x + b) \times x + c) \times x$	
00A06	02	8	00B04	加法: $R_8 \leftarrow (((a \times x + b) \times x + c) \times x + d)$	
00A07	14	8	00B05	存数: $(00B05) \leftarrow R_8$ (y 的值)	
00A08	FF	0	00000	停机	
00B00	x			初始数据区	
00B01	a				
00B02	b				
00B03	c				
00B04	d			结果数据区	
00B05	y				

从表 1.2 中可以看出,指令和数据都是以代码形式存储于同一存储器的,计算机在执行时才加以区分,这是冯·诺依曼存储程序型计算机最基本的原理。

1.3.2 指令执行过程

1. 计算机总体框图

1.2 节已阐述了构成计算机硬件系统的五大部件的功能和组成。为了了解计算机整机

工作过程,现将计算机各组成部件连接成一个整体。计算机总体框图如图 1.7 所示,它在图 1.3 和图 1.4 的基础上增加了输入/输出部件并对内存存储器作了适当简化,各部件功能与组成这里不赘述。图中实线箭头代表数据信号,而虚线箭头代表控制信号。

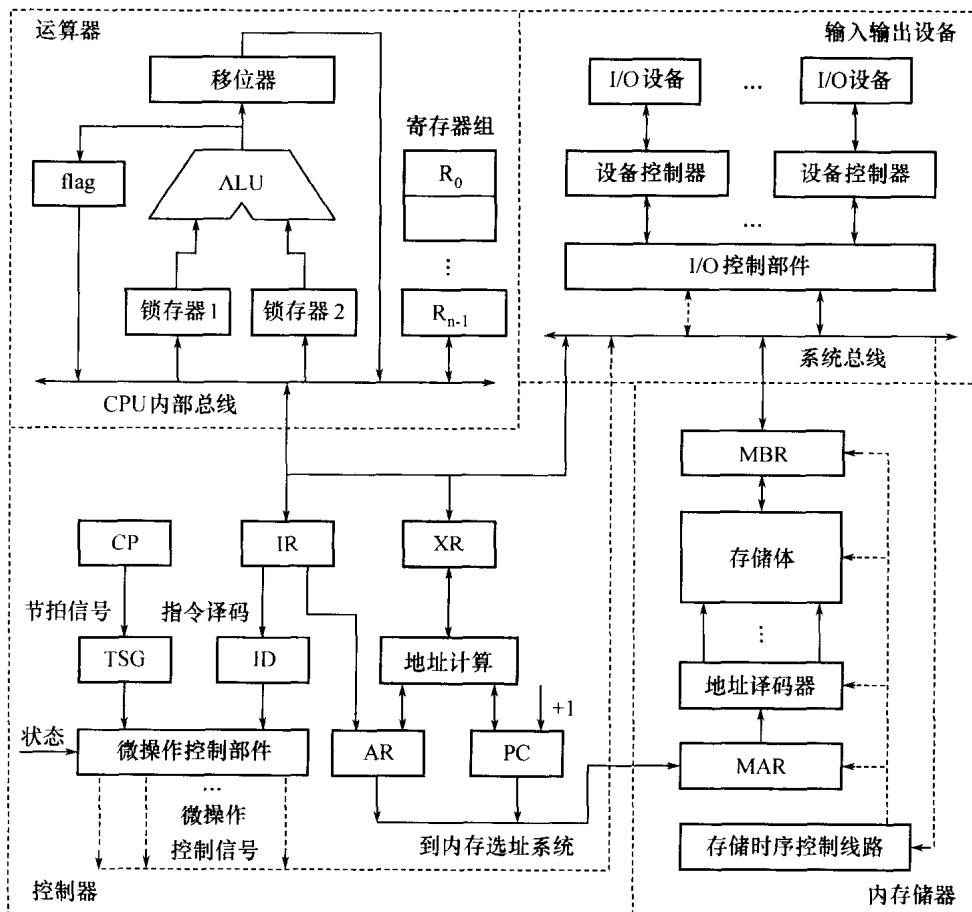


图 1.7 计算机总体框图

2. 指令工作过程简介

计算机工作过程实质上就是执行程序的过程,而执行程序的过程就是逐条执行指令的过程。因此,了解指令执行过程是了解计算机工作过程的基础。通过指令执行过程的讨论,还可以了解到计算机各组成部件具体是如何协调工作以及它们之间的功能联系。

要执行指令,首先要从内存取出指令,然后才能执行。因此,可以把指令执行过程分为两个阶段,取指令阶段和执行指令阶段。

(1) 取指令过程

参照图 1.7,取指令过程可分为如下几个步骤:

- ① 程序计数器 PC 内容送往内存储器选址系统的 MAR, 即 $MAR \leftarrow PC$ 。
 - ② 控制器向内存储器发出读命令, 内存储器把 MAR 所指存储单元的内容读出送 MBR, 即 $MBR \leftarrow M(MAR)$ 。注意, 此时 MBR 存放的是指令。
 - ③ 程序计数器内容加 1, 为取下一条指令准备好地址, 即 $PC \leftarrow PC + 1$ 。
 - ④ 经系统总线把 MBR 中的指令送控制器的指令寄存器 IR, 即 $IR \leftarrow MBR$ 。
- 上述取指令过程与取出指令的内容是无关的, 无论什么指令都需要这些步骤, 因此取指令操作被称为公共操作。

(2) 执行指令过程

指令送到 IR 后, 计算机就进入了执行指令过程。指令操作码经 ID 译码后, 确定执行操作的内容, 然后将操作控制信号送入微操作控制部件。在时序部件和运算状态的配合下, 由微操作控制部件向相关部件发送执行该指令需要的所有微操作控制信号。这些微操作控制信号有的是在同一节拍内产生的, 有些则应按序产生, 它们取决于操作的性质和完成相应微操作所用硬件的速度。指令不同, 微操作控制信号序列也不同。执行不同的指令, 涉及和使用的部件也不相同。有些指令仅涉及和使用控制器部件, 有些指令既涉及和使用控制器也涉及和使用内存储器、运算器和 I/O 等其他部件。但无论执行什么指令, 执行完最后一个微操作后都会返回到取指令的公共操作阶段, 如图 1.8 所示。

下面结合图 1.7 和图 1.8, 以执行加法指令为例说明指令执行过程。当 ID 译码后, 加法控制电位为有效。在 TSG 配合下, 微操作控制部件首先发出控制信号 $MAR \leftarrow D$, 将存储器操作数的地址送至 MAR; 然后读存储器, 并将读出的数据送至 MBR, 即 $MBR \leftarrow M(MAR)$; 接着分别将加数、被加数送往两个锁存器; 最后完成加法操作, 并将结果送回 R_i , 即 $R_i \leftarrow \text{锁存器 } 1 + \text{锁存器 } 2$ 。加法指令执行完成后, 又回到下一条指令的取指令过程。如此周而复始执行程序中的每条指令, 直至程序执行完毕。

1.3.3 计算机工作的过程

计算机能实现自动解题完全是程序员预先安排的结果, 没有程序员设计好的程序, 计算机将一事无成。现以表 1.2 中的机器语言程序为例, 结合图 1.7 说明计算机的工作过程。

表 1.2 中的机器语言程序是计算 $y = ax^3 + bx^2 + cx + d$ 的程序。程序设计好后如何装入内存储器呢? 有多种方法, 其中一种是通过操作系统下的调试程序送到内存储器指定存储区域。同时通过它把机器语言程序的始地址送给程序计数器 PC。例 1.1 中, 应把程序始地址 00A00H 送 PC, 然后可启动计算机工作。

计算机开始执行程序, 即开始了指令执行的过程。从 00A00H 存储单元取出指令 10800B01H 放入 IR, PC 内容加 1 变为 00A01H, IR 的内容经 ID 译码发现是取数指令, 于是在执行指令阶段, 将存储单元 00B01H 中的数 a 读到 R_8 寄存器中; 接着又进入取指令阶段, 从 00A01H 存储单元取出指令 08800B00H 放入 IR, PC 内容加 1 变为 00A02H, IR 的内容经

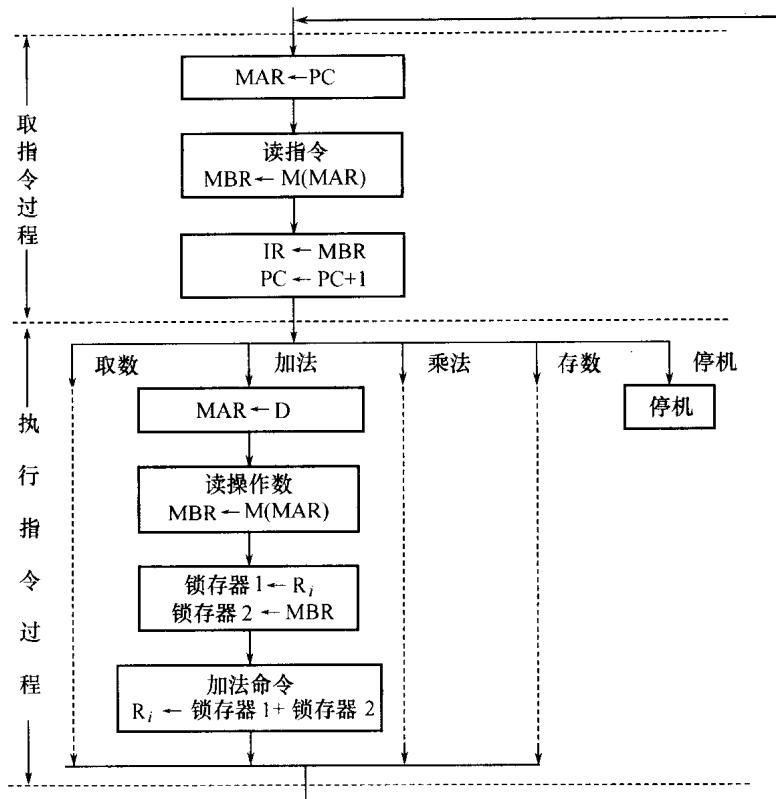


图 1.8 指令执行过程

ID 译码发现是乘法指令,于是在执行指令阶段,从 00B00H 存储单元取出乘数,将它与 R_8 中的被乘数 a 进行乘法运算,乘积存入 R_8 ;接着又从 00A02H 存储单元取出指令 02800B02H 放入 IR,PC 内容加 1 变为 00A03H,IR 的内容经 ID 译码发现是加法指令,执行加法指令;如此执行程序中的每条指令,直至从 00A08H 存储单元取出停机指令并执行。停机指令使 TSG 不再发出节拍信号,计算机也停止了指令执行过程,该程序执行完毕。

1.4 计算机的性能指标和分类

1.4.1 计算机的性能指标

怎样评价一台计算机,这是一个较复杂的问题。一台计算机性能的高低,与它的组成与结构、软硬件配置等因素有关。下面仅介绍与硬件相关的几个性能指标。

1. 主频

主频或时钟周期是计算机的主要性能指标之一,它在很大程度上决定了计算机的运行

速度。CPU 的工作节拍是由时钟 CP 控制的,时钟不断产生固定频率的时钟脉冲,这个时钟的频率就是 CPU 的主频 f 。主频越高,CPU 的工作节拍就越快,运算速度就越高。主频通常用一秒种内处理器所能发出电子脉冲数来表示,单位一般为兆赫兹(MHz)。以 Intel 系列微型计算机为例,最早 8086 的主频为 5 MHz,80386 的主频为 16 MHz,586 的主频可达 266 MHz;奔腾 II 的主频可达 450 MHz;奔腾 III 的主频可达 1.2 GHz,而奔腾 4 的主频现已达 3 GHz 甚至更高。

2. 运算速度

运算速度是计算机工作能力和工作效率的主要表征,它取决于在给定的时间内处理器所能处理的数据量以及处理器的时钟频率。运算速度通常用每秒执行指令的条数来表示,其计量单位为 MIPS(百万条指令每秒)或者 MFLOPS(百万次浮点运算每秒)。MIPS 用来描述计算机的定点运算速度;MFLOPS 则用来描述计算机的浮点运算速度。对于计算机的运算速度,可以采用以下一些方法来计算。

(1) 吉布森混合法

早期评估计算机的运算速度,需要从应用课题程序中统计各类指令所占百分比,然后用指令混合比计算指令的平均执行速度,这种方法称为吉布森混合法(Gibson Mix),也称混合比率计算法。假定第 i 类指令($i=1,2,\dots,n$)在使用过程中出现的概率为 P_i ,其执行时间为 t_i ,则指令的平均执行时间 t_E 为

$$t_E = \sum_{i=1}^n P_i \times t_i$$

t_E 的倒数就是吉布森混合法平均执行速度,其中 P_i 的集合即为混合比。

由于现代计算机的指令系统十分复杂,不少指令的执行时间并不固定,采用固定的比例可能严重偏离实际,故这种方法目前较少采用。

(2) 计算各种指令的执行速度

根据处理器的主频 f (MHz),求出处理器的基本节拍周期 $T = 1/f$ (ns),再根据处理器结构模型和指令操作流程,可推算出执行各种指令的基本节拍数和每秒执行指令的条数。显然,这种方法得到的性能并不十分精确,有时甚至做了一些理想化的假定(例如假定全部功能部件都处在满负荷的最佳工作状态下)。这种在理想状态下得到的最佳性能被称为峰值执行速度,在实际指令执行时是不可能出现的。

(3) 计算典型程序的运算速度

选取应用中具有代表性的课题,如快速傅里叶变换 FFT、图形图像的处理等,计算出这些典型程序的运算速度。

除了上述方法外,对计算机的运算速度还可以采用模型分析与模型模拟,以及实际测量等方法进行。

3. 运算精度

运算精度通常用计算机能直接处理的二进制信息位数来衡量。这个位数一般和 CPU

中存储数据寄存器的位数是相同的,位数越多,精度越高。参与运算操作数的基本位数称为基本字长。因此,基本字长也在一定情况下标志着计算精度。基本字长决定着寄存器、加法器、数据总线等的位数,直接影响着硬件的造价。为了适应不同类型计算的需要,并较好地协调精度与造价的关系,许多计算机允许变字长计算,如半字长、全字长、双字长等等。早期的微型计算机字长多为8位和16位,现在多为32位;大中型计算机多为32位和64位;巨型计算机一般都是64位。由于数据和指令都存放在内存中,所以内存存储单元长度一般等于字长或是字长的整数倍。

4. 主存存储容量

主存储器用来存储数据和程序,直接与CPU交换信息。主存的容量越大,可存储的数据和程序就越多,处理问题的能力也就越强;而且与外存储器的信息交换次数越少,系统的效率就越高。因此,主存容量是衡量计算机的主要性能指标之一。以字为单位的计算机常用字数乘以字长表示主存容量。例如 $4\text{ K} \times 16\text{ 位}$,表示主存有4 096个存储单元,每个单元字长16位。以字节为单位的计算机则以字节数表示主存容量。例如32 KB,表示主存容量为 $32\ 768 \times 8\text{ 位}$ 。现在的高档微机,主存容量从128 MB、256 MB到512 MB不等,有的为2 GB甚至更高。

5. 主存存取周期

将信息存入主存,称为写入;将信息从主存中取出,称为读出;对主存的读写,简称访存。对主存连续两次访存所允许的最短时间间隔,叫存取周期。存取周期既是表征主存性能的基本参数,也是反映计算机整机性能的重要参数。显然,存取周期愈小,表明从主存存取信息的时间愈短,计算机系统性能愈高。现在的计算机主存存取周期一般在几十纳秒,有的甚至可以达到几个纳秒。

一台计算机技术性能的高低好坏,考虑的因素应是多方面的,如系统结构、硬件组成、外设配置、软件种类、吞吐率和响应时间,还有可靠性、可用性、可维性、完整性和安全性等。不能片面强调某一项指标,应综合全面考虑。通常以性能价格比作为综合性指标,性能是指机器的综合性能,包括硬件、软件的各种性能。价格指的是整个系统的价格。显然,性能价格比愈大越好,它是选择和设计计算机的重要依据。

总之,计算机性能评价是比较复杂和细致的工作。为此,已设专门的课程进行探讨研究。

1.4.2 计算机的分类

电子计算机是一种由电子线路构成的设备,它能对信息进行记忆、传送和处理,从而实现计算的功能。计算机最常见的分类方法有两种。

1. 按计算机的用途分类

计算机按其用途即应用特点可分为通用计算机和专用计算机。

专用计算机是针对某一特定应用领域或面向某种算法的计算机,如工业过程中的控制计算机,军事上武器装备的控制指挥仪,卫星图像处理和天气预报用的大型并行处理机等。专用机的结构及软件是专门针对其应用领域设计的,因而对该领域的应用是高效的。专用机若用于其他领域,则效率较低甚至无法运行。这种机器功能单一,结构较简单,成本较低,可靠性较高。

通用计算机就是人们常说的计算机。它通常根据不同的计算机系列型号配备一定的外设。它配备多种系统软件,如操作系统、数据库管理系统及多种工具软件,只要再配备相应 的应用软件,就可应用于各种领域。通用机的特点是通用性强,功能全。

2. 按计算机的规模分类

规模是综合计算机的多方面因素而言的。通常涉及计算机的运算速度、字长、存储容量、外部设备等硬件配置,以及软件、价格等诸多方面。因此,按规模分类实际也即按计算机的性能来分类。由于计算机科学技术的飞速发展,这种规模或性能的概念也是在不断变化的。昔日的超级计算机,其性能可能赶不上现在的微型机。尽管如此,计算机界仍习惯将计算机分为巨型机、大型机、小型机、微型机、单片机等几种类型。

(1) 巨型机

巨型机(Supercomputer)也称为超级计算机。它是一个国家科技水平、经济实力和军事威力的象征。目前,世界上仅有美、英、俄、法、日、中等少数几个国家拥有巨型机。巨型机速度最快,性能最高,功能最强,技术最复杂,具有巨大的数值计算和信息处理能力,是计算机高精尖技术的集中代表。目前巨型计算机一般字长 64 位,每秒平均执行百亿次以上的浮点运算,主存容量 1~4 百万字以上,其高速数据通道每秒可传送数据几千万个字以上,具有丰富的系统软件。例如 IBM 公司的 SP2 系列,速度可达每秒 3 万亿次浮点运算。Intel 公司使用 Pentium Pro 构成的巨型计算机 Paragon,速度达到每秒 1 万亿次。我国自行研制的银河Ⅱ型巨型机运算速度达到每秒 10 亿次,银河Ⅲ型巨型机运算速度达到每秒 130 亿次,曙光巨型机的运算速度已达每秒 3 000 亿次。

(2) 大型计算机

大型计算机(Large Scale Computer / Mainframe)是计算机家族中通用性最强、功能很强的计算机。一般字长 32~64 位,每秒执行数百万到数千万条指令,主存容量几十万到几百万字,有较多的外设和通信接口,有很强的 I/O 处理能力和丰富的系统软件及应用软件,是体现所在时代计算机科技水平的一类高性能大容量计算机。典型机种如 20 世纪 60 年代的 IBM 360 系列、70 年代的 IBM 370 系列、80 年代的 IBM 370-XA 系列和 90 年代的 IBM 390 系列等。

(3) 小型计算机

小型计算机(Minicomputer)是性能较好、价格便宜、应用领域十分广泛的计算机。一般字长 32 位,速度每秒几十万到几百万次,内存容量几万到几十万字,配有一定数量的外设与

通信接口,有汇编语言和多种高级语言,有功能较强的操作系统。

另外,还有一类超级小型计算机(Super Minicomputer),其处理能力、内存容量、操作系统功能等均远远超过一般的小型计算机。历史上,小型计算机和超级小型计算机的典型机种有NOVA系列、PDP-11系列、VAX-11系列、MicroVAX系列和王安VS系列等。

(4) 微型计算机

微型计算机(Microcomputer)简称微型机或微机,它是以微处理器为中央处理器而组成的计算机系统。它是性能价格比最高、应用领域最广的一种计算机。近年来,微型机的发展最快、市场占有率最高,其许多性能达到甚至超过了小型机的水平。微型计算机正不断向微型化、网络化、高性能和多用途方向发展。

微型计算机的分类方法很多。按字长可分为8位、16位、32位微型计算机;按组装形式可分为非便携式和便携式微型计算机。前者如台式机,后者是一种可移动的微型机,如笔记本型和掌上型计算机;按最终是否由用户使用可分为独立式和嵌入式微型计算机,前者可由最终用户直接使用,最常见的是个人计算机(Personal Computer, PC),后者是将其作为一个信息处理部件装入一个应用设备,最终用户不直接使用计算机,而使用该应用设备,如包含计算机的医疗设备、家用电器等。

(5) 单片机

将中央处理器、存储器和输入/输出接口集成在一块芯片上的微型计算机,简称单片机(Single Chip Computer)。由于单片机主要应用于控制系统,故通常又称为微控制器(Micro Control Unit, MCU)。

单片机一般都采用面向控制的系统结构和指令系统。字长从4位、8位到16位,20世纪90年代又推出了32位单片机。多功能的输入/输出结构是单片机的显著特点之一。单片机通常有定时计数器、并行接口、串行接口、数模转换器和模数转换器等。

个人计算机和单片机是目前应用最广、产量最高的两种计算机。单片机多作为MCU装入各种设备,是嵌入式微型计算机最主要的应用形式,通常用在家用电器、一般控制及高速控制等应用领域。在家用电器中,多采用4位单片机或低档8位单片机;在一般控制应用中,多采用高性能8位或16位单片机;在高速控制应用中,则多采用高性能16位或32位单片机。

1.5 计算机应用与发展

1.5.1 计算机的应用

目前,计算机可以说是无所不能,无处不在。按照特点,计算机的应用大体可以概括为科学计算、数据处理、实时控制、辅助设计和智能模拟等几大类。

1. 科学计算

计算机的发明和发展,首先是为了解决科学技术和工程设计中大量的数学计算问题。因此,科学计算是计算机应用的一个重要领域。有效地使用数字计算机来求解数学问题的近似解的理论、方法与过程,已形成专门的学科——数值计算(Numerical Computation)。

许多计算领域的问题,如计算物理、计算力学、计算化学、计算经济学等均可归结为数值计算问题。这类计算往往涉及到较复杂的数学公式,如求解上千阶的微分方程组,几百个线性方程构成的方程组、大型的矩阵运算等。这样的计算任务是其他计算工具根本无法完成的。计算机的快速性与精确性为解决此类科学计算课题提供了不可缺少的重要工具。没有计算机,或者说没有高性能计算机,这些科学研究与工程设计课题或者不能解决,或者即使能解决,其速度和质量也不能达到预期的效果。

计算机应用于科学计算大大促进了科学的研究和国民经济的发展,而科学的研究和国民经济的发展又对计算机不断提出新的要求。计算机在科学计算领域的巨大威力正日益发挥出强大的推动作用。

2. 数据处理

数据处理的主要功能,是将输入设备送来的数据及时记录、整理、分类、加工,以得到所需要的信息,如企业管理、库存管理、帐目计算、情报检索、图像处理等。它们的特点是原始数据量大,算术运算比较简单,有大量的逻辑与判断,处理的结果多以报表或文件形式存储或输出。对于这些工作,人力根本无法完成,而采用计算机则很快可得到结果。

随着软件技术特别是数据库技术的发展,数据处理已成为计算机应用的重要领域。目前,利用数据库系统软件,如 Sybase、Oracle、Access 等开发的各种应用软件系统,如财务管理系统、交通客票预定系统、物资管理系统、人事管理系统、办公自动化系统、数字地图系统、情报分析与检索系统等都已得到广泛应用,使人们从大量繁杂的数据统计与管理事务中解脱出来,大大提高了工作效率与工作质量。利用计算机网络与通信技术,实现资源共享和远程信息传输,更使数据处理这类应用焕发了新的生机。目前世界上用于数据处理的计算机数量已大大超过科学计算所用计算机的数量。

3. 实时控制

实时控制是计算机在过程控制中的重要应用。“实时”就是指计算机的运算和控制时间与被控制过程的实际时间相适应。实时性是以计算机速度为基础的,不同受控对象其物理过程的时间参数是不同的。如生产过程控制时间参数较大,对计算机速度要求不高;对武器发射过程控制时间参数较小,对计算机速度要求高。最初的过程控制主要应用于导弹、卫星、飞船等现代化武器系统和航空航天等领域,而现在过程控制已广泛应用于工业生产过程。过程控制发挥着神经系统的功能,由于高灵敏度、高精确度的控制,使得上述领域的工作者能在人的意志控制下准确无误地完成。

4. 辅助设计

计算机辅助设计(Computer Aided Design,CAD)是利用计算机帮助设计人员进行工程、产品等设计工作的过程和技术。

在航空航天、船舶交通、机械电子、基建和化工等部门,工程设计的工作量极其巨大,不但要进行大量的工程计算,而且要绘制许多的工程图纸。由计算机辅助产生的设计结果通过图形设备与设计人员交互,可及时对设计做出判断和修改,最终完成设计工作。采用 CAD 技术,提高了设计的自动化水平,缩短了设计周期,减轻了设计人员的劳动,也大大提高了设计质量。以飞机设计为例,靠人工设计,从制定方案到画出全套工程图纸,一般需要 3 年左右的时间,而采用 CAD 后,只需 1~2 个月即可完成。

CAD 技术的发展,也带动了计算机辅助制造(CAM)的进步。CAM 是在制造业中,利用计算机辅助各种设备完成产品的加工、装配、检测和包装等的过程和技术。CAM 的应用,显著地提高了企业的生产效率,缩短了工作周期,降低了产品成本,提高了产品质量。现在 CAD 和 CAM 紧密结合在一起,构成所谓 CAD/CAM 系统。

此外,计算机辅助教学(CAI)等也得到愈来愈广泛的应用。CAI 是利用计算机辅助教师对学生进行教学的过程和技术。其最大特点是在计算机与学生之间实现对话,学生可根据个人特点进行学习,变被动为主动,生动直观形象,极大地提高了学习的兴趣与效果。

5. 智能模拟

人工智能是计算机应用研究最前沿的学科。它是采用计算机来研究开发用于模拟、延伸和扩展人的智能,如感知、推理、学习、理解等。人工智能是在质的方面扩充计算机的能力,提高它的智能水平。人工智能的研究与应用已在模式识别、景物分析、自然语言的理解与生成、博弈、自动定理证明、自动程序设计、专家系统、模拟训练系统、智能决策系统以及机器人等领域得到发展。

智能模拟将计算机的应用提高到了一个更高的阶段,为计算机的推广应用开拓出一个全新的领域。怎样把计算机用得更聪明,怎样设计和制造具有高智能水平的计算机应用系统,不仅涉及到软件,也涉及到硬件和其他自动化设备及通信设备。

上述五种应用类型虽覆盖了计算机应用的大多数领域,但并未能包括当今计算机的所有应用领域。随着计算机网络技术和信息高速公路的发展,计算机的应用几乎渗透到人类活动的各个领域,如电子商务、电子邮件、交互式通信、检索查询、电子游戏、远程医疗、远程教育、电子购物等,计算机的广泛应用正改变着人类的生活方式,它标志着人类进入了一个全新的时代——信息化时代。

1.5.2 计算机发展前景

计算机科学技术的发展可谓是日新月异。那么,计算机将怎样发展呢?就其总的的趋势来看,主要是巨型化、微型化、网络化和智能化。

1. 巨型化

由于航空航天、气象预报、地震分析、原子裂变、生物工程等领域中大型科学计算和数据处理的需要,对超大规模、高性能巨型计算机提出了永无止境的性能要求。因此,计算机向巨型化方向发展,将是无可争议的趋势。日本 NEC 公司 2002 年制造的 Earth - Simulator 超级计算机由 5 120 个处理器构成,每秒能完成 35 万亿次的浮点运算,它是目前世界上最快的计算机。从 1993 年开始,每隔 2 年世界上性能最高的 500 台计算机被列于 www.top500.org 网站上,有兴趣的读者可自行访问。

2. 微型化

计算机向微型化、高性能、多用途方向发展是另一个不争的事实。一方面,微处理器芯片性能不断提高,例如奔腾 4 芯片的主频已达 3 GHz 以上。目前微处理器芯片的计算能力已经超过了早期巨型机的计算能力。另一方面,微型机所需的配套器件的性能也在不断提高。如主存芯片的集成度、读写速度和数据传输率,光盘、移动硬盘和硬盘的容量、速度和数据传输率,显示器的综合性能,各种多媒体和大量预装软件等都使得微型机的整体性能不断攀升,使得微型机的性能愈来愈高、用途愈来愈广。

3. 网络化

Internet 的建立和信息高速公路的崛起,使得人类社会处理和传播信息的能力大大增强。目前全球互联网用户近 7 亿,而在我国大陆地区已达 8 000 万。为适应网络化的需要,可移动式计算机及网络配套用的硬件和软件,如调制解调器、声卡、视频卡、电源等装置将不断推出新型的产品。计算机硬件和软件的不断创新,以及以 Internet 和信息高速公路为核心的“网络革命”和“网格计算”,将成为信息时代和知识经济时代强大的助推器。

4. 智能化

计算机科学与控制论、仿生学、心理学等相结合,将使计算机的研究与发展产生一次重大的变革。计算机的智能化一直以来是人类的奋斗目标。用计算机去模拟人类的某些智能行为,如触觉、视觉、嗅觉等感觉功能,对声音、图形、图像及其他模式的识别能力,提供知识、进行推理和自我学习的能力等,都是智能化的努力方向。计算机智能化的不断进展,使得计算机不但能储存、计算和处理数据,而且在模式识别、定理证明、学习研究、联想探索、语言理解等领域都将成为人类的得力助手。

计算机是人类 20 世纪最伟大的创造之一。几十年来,它的发展速度之快、普及应用之广可谓史无前例。从整体上看,其未来发展还具有下述特点:

首先,新型元器件、体系结构及实现技术的发展,将大大提高计算机系统的性能及性能价格比。随着集成电路纳米级微细加工技术的成熟,半导体各种芯片的集成度、速度等将进一步提高。而光电子器件与生物器件一旦成为现实,计算机的运算速度还可提高几个数量级。计算机辅助技术及新兴工艺技术的应用,又将使计算机的整体性能大幅度提高。

其次,计算机与通信技术的融合与渗透,将大大加速人类社会信息化的进程。全球性的计算机联网,将大大促进信息资源的开发与利用。计算机走进寻常百姓家,已成为人类工作、学习与生活的必需品。计算机科学与技术将成为人类必须学习与具备的基础知识。

第三,随着以智能化、集成化、自动化、并行化、开放化和自然化为标志的计算机软件新技术的深入研究、开发与利用,不但将使软件的功能与性能迅速提高,而且将解决软件生产率低下的问题。软件理论与软件工程的发展,将从理论与实践两方面解决计算机系统开发中的软件瓶颈问题。信息产业和软件产业将成为新的经济增长点,成为国民经济的支柱产业。

第四,在 Internet 与信息高速公路迅速发展普及的形势下,为保证信息资源共享,计算机系统与网络的互操作性、开放性和标准化将受到高度重视。由于计算机进入千家万户,使用的简明化、自然化、信息的安全保密、防止计算机犯罪等问题也是计算机科学技术中的重大研究课题。

第五,完全新型的计算机不断问世。前面提到的各种类型计算机均采用存储程序计算机结构,统称为传统计算机。传统计算机的基本工作方式是顺序执行指令的串行工作方式,这将导致传统计算机在并行处理、字符处理、知识处理等方面的低效能。因此,突破存储程序计算机结构的局限性,人们研究开发了一些非传统计算机,例如数据流计算机、归约机、逻辑推理机、神经计算机等。此外,为进一步提高计算机系统的性能,研究开发不采用传统电子器件的计算机也是一个重要方向与课题,例如光计算机、生物计算机以及量子计算机等。

小 结

本章简要回顾了计算史和计算机发展史;重点阐述了冯·诺依曼存储程序计算机五大部件的结构组成、工作原理及各组成部分的功能,并通过一个解题实例,说明计算机是怎样实现自动解算问题的,目的是加深读者对计算机五大部件组成、功能和工作原理的了解和认识,以便初步建立计算机整机的清晰概念,激发读者学习该课程的兴趣;另外,本章还介绍了计算机硬件的几个主要性能参数和计算机的两种分类方法。

本章是开篇之章,对计算机各大部件阐述仅是初步的、简略的和提纲携领式的。读者肯定会有诸多疑问或很想了解更多更深入的问题,有些问题的阐述亦可能存在不确切之处。本书后续各章将分别对计算机的组成部分及其工作原理进行深入细致的探讨。

习 题

1.1 解释下列术语:

硬件 软件 固件 CPU 主机 主频 运算速度 指令

1.2 什么是存储程序原理?按此原理,计算机应具备哪些功能?

1.3 存储程序计算机包含哪几部分?它们的功能各是什么?

- 1.4 控制器的任务是什么？它主要由哪些部件组成？简述各组成部件的功能。
- 1.5 何谓程序计数器？它有何作用？
- 1.6 试述内存储器的组成及各组成部分的功能。
- 1.7 存储器是怎样编址的？128 K 字、512 K 字和 1 M 字容量的存储器其 16 进制编址范围如何表示？
- 1.8 试说明计算机硬件的主要性能参数。
- 1.9 何谓基本字长？它对计算机的哪些部件有何影响？
- 1.10 根据表 1.2 中的程序，结合图 1.7 中的计算机总体框图说明计算机工作过程，并计算执行完该程序共访问多少次内存储器。
- 1.11 计算机一般有哪两种分类方法？各可分成哪些类型的计算机？
- 1.12 计算机可应用于哪些方面？对每一种应用类型各举两个实例加以说明。
- 1.13 就你所知道和了解的当前计算机的热门话题，谈谈你的看法。

第二章 指令系统

引言

指令系统(Instruction Set)是计算机程序员接触到的计算机的所有功能,它是影响计算机全局性的问题,也是计算机硬件和软件的接口和界面。指令系统是表征计算机CPU的重要因素,它不仅直接影响CPU的硬件结构,而且也直接影响系统软件、机器的功能、效率和适用范围。

一台计算机的指令系统包含了所有机器指令。机器指令就是计算机设计者赋予计算机实现某种基本操作的命令。计算机的一切工作都是通过执行由它组成的程序来完成的,这样的程序被称为机器语言程序。如无特别说明,本书中的指令即指机器指令。

指令应包括三个基本要素:

- 操作码(Operation Code),指明进行什么操作,如加减等,常用 opcode 或 OP 标记。
- 源操作数地址(Source Operand Reference),指出操作数在哪,常用 S 标记。
- 目的操作数地址(Result Operand Reference),指出指令操作结果送往何处,常用 R 标记。

在计算机内部,指令表示为一个二进制位串,这个位串被称为指令字。指令字被划分为几个字段,分别对应指令的各要素,图 2.1 给出一个简单的指令字的例子。因为二进制位串可读性差,所以操作码可用助记符(mnemonic)来表示。例如:ADD 表示加法操作,SUB 表示减法操作。

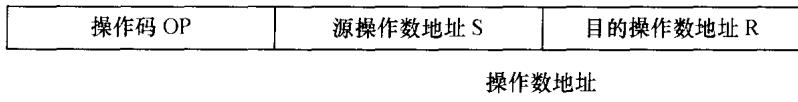


图 2.1 一个简单的指令字

除了包括所有机器指令外,指令系统中还包括数据表示(硬件可以直接识别的数据类型)以及寻址方式(如何获得操作数地址)两部分的内容。本章首先介绍数值、字符、堆栈和向量等数据表示,然后阐述指令系统的分类、常见寻址方式、指令格式以及设计指令时应考虑的各种因素,最后简单介绍复杂指令集计算机(Complex Instruction Set Computer, CISC)和精简指令集计算机(Reduced Instruction Set Computer, RISC),并给出 Pentium 和 PowerPC 这两种典型机型的指令系统做为实例。

2.1 数据表示

操作数在计算机中需要采用一定的形式来表示,以便计算机存储、识别和处理。数据表示(Data Representation)指的是能由计算机硬件直接识别的数据类型,如定点数、浮点数等。由硬件直接识别意味着某种数据类型可用计算机硬件直接表示,并能由计算机指令直接调用该种数据类型。

只有定点数据表示的计算机,称为定点计算机。定点数可由寄存器或存储器单元来表示或识别,机器指令调用的操作数只能是定点数,而其他类型的数据,如浮点数、向量等在定点机中不能由硬件直接表示,也不能由机器指令直接调用。但是,这并不是说,定点计算机不能处理浮点数据或向量数据,可以通过软件的方法将其变换为等效的定点数据类型,然后再加工处理。

不同种类计算机的数据表示是有差别的,因此在不同计算机上存储的信息往往不能够互相正确识别。计算机的数据表示,包括数据的取值范围与精度要求、表示形式等,不但会影响计算机的硬件结构与组成,而且还可能影响算法的选择。研究一台计算机的指令可以对哪些数据类型进行操作就是研究该计算机的数据表示。

本节主要介绍数值和字符的数据表示,另外还将简单介绍堆栈和向量数据表示。

2.1.1 数值数据的数据表示

计算机中普遍使用三种类型的数值数据:定点数、浮点数和十进制数。

1. 定点数

定点数据是各种数据类型中最简单、最基本的一种数据表示,它用于表示二进制形式具有固定比例换算的数据。如图 2.2 所示,由于定点位置的不同,定点数一般分为两类:

- 整数(Integer):小数点固定于最低位右边的数。
- 小数(Fraction):小数点固定于数的左端,在 2^0 与 2^{-1} 之间,又称为分数。

为避免混淆,本书采用如下约定:一个数列各位的下标等于该位权的幂次绝对值。如整数表示成 $D_{n-1} \cdots D_1 D_0$,其中 D_i 位的权为 2^i ;小数表示成 $D_0.D_1 \cdots D_{n-1}$,其中 D_i 位的权为 2^{-i} 。

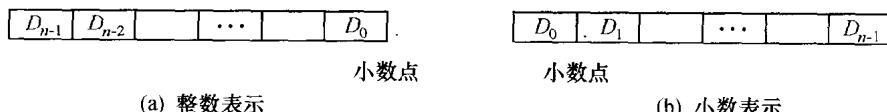


图 2.2 定点数表示

由于有浮点数据表示(在下一节中讨论),现代计算机中大多定点数只具有整数数据表示,而小数则通过浮点数据表示来实现。

整数数据可以分为无符号整数和带符号整数。无符号整数一般表示地址等。表示算术操作数时,应使用带符号整数,一般以左边最高位表示符号位,数据可用原码、补码或反码等来表示。这三种机器数的表示形式中,符号部分的规定是相同的,所不同的仅是数值部分的表示形式。有关定义如下:

设真值 x 为整数,即 $x = \pm x_1 x_2 \cdots x_{n-1}$ ($-2^{n-1} < x < 2^{n-1}$) (补码 ($-2^{n-1} \leq x < 2^{n-1}$))

$$\text{则 } [x]_{\text{原}} = \begin{cases} x & \text{当 } 0 \leq x < 2^{n-1} \\ 2^n - x & \text{当 } -2^{n-1} < x \leq 0 \end{cases}$$

$$[x]_{\text{反}} = \begin{cases} x & \text{当 } 0 \leq x < 2^{n-1} \\ (2^n - 1) + x & \text{当 } -2^{n-1} < x \leq 0 \end{cases}$$

$$[x]_{\text{补}} = \begin{cases} x & \text{当 } 0 \leq x < 2^{n-1} \\ 2^n + x & -2^{n-1} \leq x < 0 \end{cases}$$

设真值 x 为小数,即 $x = \pm 0.x_1 x_2 \cdots x_{n-1}$ ($-1 < x < 1$) (补码 ($-1 \leq x < 1$))

$$\text{则 } [x]_{\text{原}} = \begin{cases} x & \text{当 } 0 \leq x < 1 \\ 1 - x & \text{当 } -1 < x \leq 0 \end{cases}$$

$$[x]_{\text{反}} = \begin{cases} x & \text{当 } 0 \leq x < 1 \\ (2 - 2^{-(n-1)}) + x & \text{当 } -1 < x \leq 0 \end{cases}$$

$$[x]_{\text{补}} = \begin{cases} x & 0 \leq x < 1 \\ 2 + x & -1 \leq x < 0 \end{cases}$$

详细内容参见本系列教材《数字逻辑原理与工程设计》。

早期计算机中整数数据的长度是固定的,一般取机器字长(机器字长指计算机能直接处理的二进制数据的位数);而参与运算的操作数长度也是固定的,一般称为定长运算或定长操作。在实际使用中发现,固定字长存在着明显的缺点。如:某些应用的数据范围和精度要求只需较少的位数就可满足,但在定长数据类型中,多余的位只能填 0 或 1(取决于数的正负和表示的码制),使运算器和存储器设备利用率较低;而另外一些数据范围和精度要求较高的应用场合,单字长又不能满足要求,采用软件方法进行多倍字长运算,又大大降低了机器运算速度和效率。理想的方案是字长随用户需要而变,计算机能实现可变字长操作。但是,这将引起计算机硬件结构设计的复杂性,降低机器的性能价格比。作为折衷的方案,目前计算机中设计并采用了多种定长数据类型,对每种字长的操作仍然是定长操作。

下面以 Intel 的 Pentium 为例,讨论整数数据表示的一些特征。Pentium 能处理的整数数据类型如图 2.3 所示。

(1) 无符号整数

无符号整数有 8 位、16 位、32 位和 64 位四种字长,包括:

- 无符号字节整数(Byte Unsigned Integer);字长 8 位,数值范围:00H ~ OFFH,十进制为 0 ~ 255。通常以 H 结尾表示十六进制数据,并在十六进制 A、B、C、D、E、F 开头的数值前加 0。

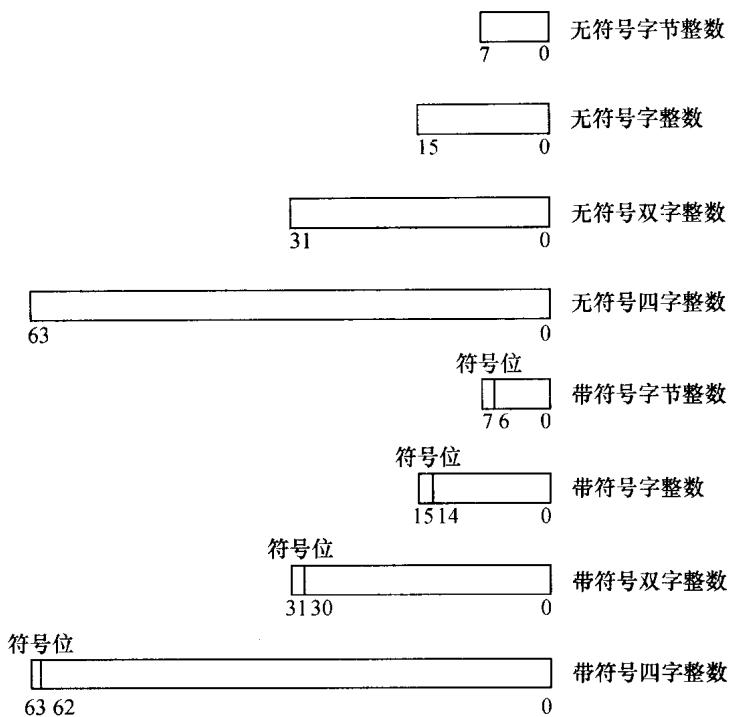


图 2.3 Pentium 整数数据类型

- 无符号字整数 (Word Unsigned Integer); 字长 16 位, 数值范围: 0000H ~ OFFFH, 十进制为 0 ~ 65535。
- 无符号双字整数 (Doubleword Unsigned Integer); 字长 32 位, 数值范围: 00000000H ~ OFFFFFFFH, 十进制为 0 ~ 2^{32} - 1。
- 无符号四字整数 (Quadword Unsigned Integer); 字长 64 位, 数值范围: 0000000000000000H ~ OFFFFFFFFFFFFFFFH, 十进制为 0 ~ 2^{64} - 1。

(2) 带符号整数

在 Pentium 中, 带符号整数都采用补码形式表示, 符号位为 0 则表示该数为正数, 为 1 则表示该数为负数。带符号整数有 8 位、16 位、32 位和 64 位四种字长, 包括:

- 带符号字节整数 (Byte Signed Integer); 字长 8 位, 数值范围: 80H ~ 7FH, 十进制为 -128 ~ +127。
- 带符号字整数 (Word Signed Integer); 字长 16 位, 数值范围: 8000H ~ 7FFFH, 十进制为 -32768 ~ +32767。
- 带符号双字整数 (Doubleword Signed Integer); 字长 32 位, 数值范围: 80000000H ~ 7FFFFFFFH, 十进制为 - 2^{31} ~ +(2^{31} - 1)。
- 带符号四字整数 (Quadword Signed Integer); 字长 64 位, 数值范围: 8000000000000000H ~ 7FFFFFFFH, 十进制为 - 2^{63} ~ +(2^{63} - 1)。

$\sim 7\text{FFFFFFFFFFFFFH}$, 十进制为 $-2^{63} \sim +(2^{63}-1)$ 。

Pentium 指令系统中, 并不是每条指令都可支持所有类型的整数数据, 例如 MUL(整数乘)指令和 DIV(整数除)指令只能处理无符号整数。这两种指令的操作码不仅指出操作的性质, 而且还指出操作数的数据类型。

2. 浮点数

对于定点整数, 可以通过重新假定小数点的位置来表示小数, 但其所能表示的数值范围在许多应用中是不够用的。例如, 电子质量为 9×10^{-28} g, 太阳质量为 2×10^{33} g, 对这类数据如果仍然采用定点数运算是不合适的, 因此在计算机中引入浮点数据表示是必然的。

(1) 浮点数据的表示

浮点数据类型的基本原理来源于十进制数中使用的科学记数法(Scientific Notation), 上面的电子质量和太阳质量就是用科学记数法给出的。一个数 N 的科学记数法形式可写成:

$$N = M \times R^E$$

其中 M (Mantissa)代表尾数, E (Exponent)代表阶码, R (Radix)代表基数。当尾数、阶码采用二进制, 基数 $R=2$ 时, 就是计算机中的浮点数据, 故可以将计算机中的浮点数据类型理解为二进制数中使用的科学记数法。对于计算机中浮点数据的表示, R 还可取为 4、8、16 等。对整个计算机系统, 由于 R 是常数, 故不需要在数码中表示出来。因此, 浮点数只需用一对定点数就可以表示: 一个是尾数 M , 为小数; 另一个是阶码 E , 为整数。浮点数据的一般格式如图 2.4 所示。其中, 尾数 M 的符号就是整个浮点数据的符号, 故通常将其放在整个浮点数据的最高位。

符号位	阶码 E (整数)	尾数 M (小数)
-----	-------------	-------------

图 2.4 浮点数据类型的一般格式

需要强调的是: 浮点数据是实数的一种近似表示。浮点数据格式必须兼顾表示范围和表示精度的要求。下面讨论若干浮点数据表示的几个问题。

① 浮点数规格化形式

浮点数的精度由尾数 M 的位数决定, 数据的表示范围由基数 R 和阶码 E 的位数决定。一个浮点数可有多种表示形式, 如电子质量 9×10^{-28} g, 也可表示成 0.9×10^{-27} g 以及 90×10^{-29} g 等。为了提高运算精度, 使尾数的有效数字尽可能占满已有的位数, 同时也使计算机实现浮点运算时有一个统一固定的标准形式, 可将浮点数据表示为规格化(Normalized)的形式。规格化的浮点数对尾数 M 提出限制要求:

$$1/2 \leq |M| < 1$$

浮点数规格化形式又称为规格化浮点数。将一个浮点数据转化为规格化形式的过程称为浮点数规格化。

现代计算机中, 一般浮点数都是以规格化形式存储和运算的。在浮点运算时, 若尾数的

有效数字超过最高有效位(小数点后第 1 位),为使其规格化需要进行右规。右规时尾数右移 1 位,阶码加 1。若运算结果的尾数有效数字不在最高有效位,为使其规格化需要进行左规,左规时尾数每左移 1 位,阶码将减 1。

② 浮点数据中采用的编码

浮点数据的符号就是尾数的符号,尾数可以采用原码或者补码,阶码可以采用补码或者移码。一个数的移码是该数补码的反符号;由于数 0 的补码是惟一的,故数 0 的移码也是惟一的, $[0]_{\text{移码}} = 100\cdots 0$;当 $[x]_{\text{移}} = 00\cdots 0$ 时, x 为编码所能够表示的最小值。关于移码的运算将在 3.2.1 节中讨论。下面举例说明规格化浮点数据的表示。

例 2.1 假设浮点数据为 32 位,1 位符号位,阶码采用 8 位移码表示(包括一位符号位),尾数采用原码表示,基数为 2,采用规格化浮点数据类型如下:

$$\begin{aligned} 0.1101\ 0001 \times 2^{10110} &= 0\ 10010110\ 11010001\ 00000000\ 00000000 \\ -0.1101\ 0001 \times 2^{10110} &= 1\ 10010110\ 11010001\ 00000000\ 00000000 \\ 0.1101\ 0001 \times 2^{-10110} &= 0\ 01101010\ 11010001\ 00000000\ 00000000 \\ -0.1101\ 0001 \times 2^{-10110} &= 1\ 01101010\ 11010001\ 00000000\ 00000000 \end{aligned}$$

③ 浮点数据中“零”的表示

以下两种情况的浮点数($N = M \times R^E$)被当作“零”处理:

- 当尾数 $M = 0$ 时,对所有 E 值均有 $N = 0 \times R^E = 0$ 。
- 当 $E \leq -2^n$,且 $M \neq 0$ 时,称发生浮点数据下溢,即所需要表示的数据 N 小于机器所能表示的最小数,一般用 $N = 0_m$ 处理,通常称之为“机器零”。

“机器零”的标准格式应是 $M = 0, E = -2^n$,即尾数为 0,阶码为最小值。

如果采用图 2.4 所示的指令格式,当尾数和阶码均采用补码时,“机器零”的形式为:

0	100...00	00...0
	$[-2^n]_{\text{补}}$	$[0]_{\text{补}}$

当尾数采用补码,阶码采用移码时,“机器零”的形式为:

0	000...00	00...0
	$[-2^n]_{\text{移}}$	$[0]_{\text{补}}$

可以看出,当阶码采用移码表示时,浮点数“机器零”的形式为“全零”。为使定点数、浮点数判“全零”一致,多数通用计算机中浮点数的阶码采用移码表示。

(2) 浮点数据类型的特点

采用浮点数据表示,最直观的感觉是数据范围扩大了。除此之外,浮点数据类型还有哪些特点呢?

① 浮点数据的表示范围

对于一个 32 位字,补码可以表示的整数数据范围是 $-2^{31} \sim 2^{31} - 1$,共计 2^{32} 个不同的整数,如图 2.5 所示。

采用前例中的规格化浮点数据表示形式时,对于一个 32 位浮点数据,一位符号位,阶码采用移码表示,尾数采用原码表示,基数为 2,则数据表示范围为:

- 负数在 $-(1 - 2^{-23}) \times 2^{127}$ 到 -0.5×2^{-128} 之间;
- 正数在 0.5×2^{-128} 到 $(1 - 2^{-23}) \times 2^{127}$ 之间。

可见,浮点数的表示范围远远大于相同位数的整数的表示范围。除了表示范围,还有 5 个与浮点数表示能力相关的区域:

- 小于 $-(1 - 2^{-23}) \times 2^{127}$ 的负数区域,称之为负上溢出(Negative overflow);
- 大于 -0.5×2^{-128} 的负数区域,称之为负下溢出(Negative underflow);
- 零;
- 小于 0.5×2^{-128} 的正数区域,称之为正下溢出(Positive underflow);
- 大于 $(1 - 2^{-23}) \times 2^{127}$ 的正数区域,称之为正上溢出(Positive overflow)。

这些区域在数轴上的分布如图 2.6 所示。

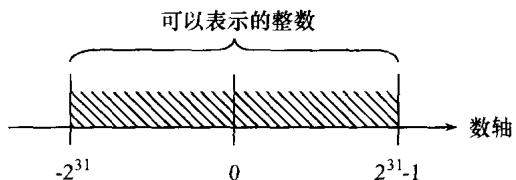


图 2.5 32 位补码整数

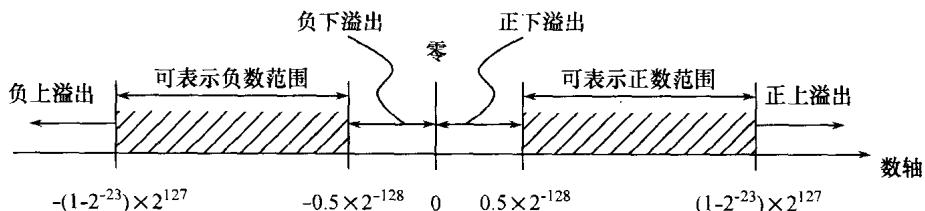


图 2.6 32 位浮点数据类型的数据范围和区域分布

负上溢出和正上溢出统称之为上溢(Overflow),负下溢出和正下溢出统称之为下溢(Underflow),上溢和下溢统称为溢出。从图 2.6 中可以看出,上溢就是数据的绝对值太大,超出了数据类型表示的能力范围;而下溢则是数据的绝对值太小,使得数据无法有效地表示。数据一旦上溢,其后的计算和使用的数据就不可能再正确,计算机必须终止处理后续程序,通知用户。与上溢相比,下溢的问题比较小,数据一旦出现下溢,就可以认为数据为 0。很多计算机系统对下溢不作任何复杂处理,仅仅置成零就可以了。所以,通常说的溢出往往是指上溢。

② 浮点数据的分布密度

浮点数与整数在数轴上的分布上存在很大的不同。

整数在数轴上是均匀分布的,也就是几乎所有连续两个数据之间的差为 1,仅仅在原码或反码表示中,+0 和 -0 的时候数据出现不均匀。

浮点数在数轴上分布则是不均匀的,越是靠近零点,数越密集,越远离零点,数越稀疏。

原因是：浮点数的位数是确定的，则所能表示的数据个数就确定了。对于基数为 2 的情况，阶码绝对值为 $n+1$ 的浮点数值覆盖区域比阶码绝对值为 n 的浮点数值覆盖区域大一倍，但两者在各自区域中所能表示的数据个数是相同的，因此阶码绝对值为 $n+1$ 的浮点数值覆盖区域内的数据密度比阶码绝对值为 n 的浮点数值覆盖区域小一倍。浮点数的数据密度分布如图 2.7 所示。

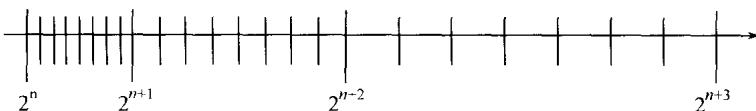


图 2.7 浮点数据类型的数据密度分布示意

③ 浮点数据的精度

浮点数据的精度是通过尾数位数来确定的，所以相同位数的整数的数据精度要高于浮点数据，这也是在进行数据处理时不容忽视的问题之一。

为了满足多种应用对精度的需求，浮点数据类型也是从固定字长发展为多种字长数据类型。由于单字长浮点数中阶码占用了若干位，尾数位数小于单字长，影响了数的表示精度。在精度要求高的场合需采用双字长浮点数表示。

(3) IEEE 浮点数标准 (Floating-point number Standard)

IEEE 浮点数标准是由国际电气和电子工程师协会 IEEE 制定的浮点数的表示格式和运算规则。浮点数的运算规则必须尽可能地保持精度，缩小误差。在计算机诞生后的很长一段时间里，由于没有统一的浮点数标准，不同系列的计算机采用各不相同的浮点数表示形式，给数值计算和软件移植带来了很多困难和麻烦。20 世纪 80 年代以后，考虑到微处理器性能的不断提高和计算机应用的进一步普及，IEEE 制定了浮点数标准，即 IEEE 754 标准。它现已成为所有微处理器应该遵循的二进制浮点算术运算标准。标准的内容包括浮点数的表示形式、浮点操作的类型和定义、舍入方式、例外处理方法等。另外还有一个 IEEE 854 标准，它主要针对十进制浮点运算。由于目前计算机主要使用二进制运算，几乎没有使用十进制浮点的计算机，所以 IEEE 854 标准很少被提及。

3. 十进制数

十进制数是人们最习惯、最常用的一种数据类型，而二进制又是计算机最适于运算处理的数据类型。因此，计算机总是要进行十进制数与二进制数之间的转换。

当计算机主要以二进制信息进行运算和处理时，只需在输入与输出处理中进行二——十进制转换，并不需要特别支持十进制数据类型，也就是说不需要设置能够识别和处理十进制数的硬件设备。但是当计算机在某些应用领域，如商业统计计算，其运算简单而数据量很大时，二——十进制转换时间要占用机器总的运行时间很大比例。为提高机器使用效率，设置十进制数据表示，直接对十进制数进行运算和处理是合适的。所以，一般通用性较强的计

算机,都设有十进制数据表示。

《数字逻辑原理与工程设计》课程中讨论过十进制数的各种二进制编码方法,其中应用最广泛的编码是 BCD 码。一位 BCD 十进制数字编码用 4 位二进制表示,即一个字节可以存放两位十进制数。

计算机中十进制数的位数怎样确定呢?由于实际应用中数据范围变化很大,故计算机多采用可变长度位数的方法。例如,Pentium 能处理以下三种十进制整数,如图 2.8 所示。

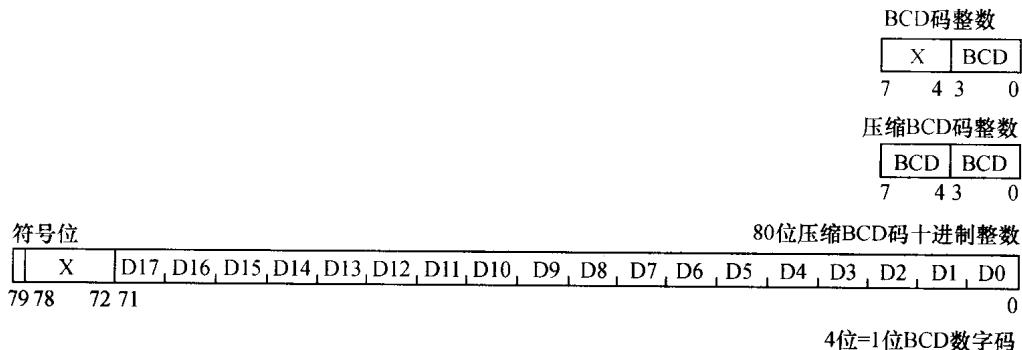


图 2.8 Pentium 的十进制整数类型

图 2.8 包括的所示的十进制整数类型包括:

- BCD 码整数(BCD integers),字长 8 位,表示 1 位无符号十进制数,低 4 位表示数值,而高 4 位无意义,在加或减的时候可以为任意值,但在乘或除时必须是 0,表示范围 0 ~ 9。
- 压缩 BCD 码整数(Packed BCD integers),字长 8 位,表示两位无符号十进制数,0 ~ 3 为低位,4 ~ 7 为高位,表示范围 0 ~ 99。
- 80 位压缩 BCD 码十进制整数(80 - bit Packed BCD Decimal integers),字长 80 位,0 ~ 71 位表示 18 位十进制数,每 4 位表示 1 位十进制数,72 ~ 78 位无意义,79 位为符号位,0 为正,1 为负,表示范围为 $-10^{18} + 1 \sim 10^{18} - 1$ 。该类型数据只存在于主存(memory)中,一旦装载入浮点单元(floating-point unit, 缩写为 FPU)数据寄存器,就自动转换成扩充双精度浮点格式,因此有的资料也将该数据类型归入浮点数。

2.1.2 字符的数据表示

人们常用的信息除了数字外,还大量使用的就是字符,但在数据处理时不便以字符形式存储或发送。因此,人们研制了几种编码方式将字符表示成二进制的位序列。早期人们采用莫斯代码(Morse Code),现在应用最广、最普及的方法是采用 ASCII 码。每个字符用 7 位来表示,可表示 128 个字符,包括:

- 图形字符 96 个:26 个大写和 26 个小写英文字母,10 个数字符号,34 个专用符号。
- 控制字符 32 个。

ASCII 码一般以每个字符 8 位来传递及存储, 其中 7 位表示字符, 1 位可置为 0 或奇偶校验位。值得注意的是: 数字 0 ~ 9 的 ASCII 码的形式是 $011 \times \times \times \times$, 低 4 位 $\times \times \times \times$ 恰好是 0000 ~ 1001, 即 BCD 码。因此 ASCII 码与 BCD 码的十进制整数之间转换是非常方便的。

被处理的字符信息, 通常是一组字符的序列, 称为“字符串”(Character String), 如用字符串表示名称、数据记录、文件等。随着计算机系统软件与应用软件的迅速发展, 字符串的运算大量增加。为了提高计算机性能减少系统软件的开销(Overhead, 运行用户程序外的系统程序所占用的计算机时间和资源), 设置字符串数据表示是很有意义的。按照字符串的存储结构, 字符串有多种表示方法。最常用的是向量表示法和向前串表表示法。

(1) 向量表示法

在字符串向量表示法中, 串的所有元素(字符)在物理上是邻接的。如字符串“IF X > 0 THEN CALL OK”, 其向量表示在存储器中的分配如图 2.9 所示。

A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10	+11	+12	+13	+14	+15	+16	+17	+18	+19			

I	F		X	>	0		T	H	E	N		C	A	L	L		O	K	;		
---	---	--	---	---	---	--	---	---	---	---	--	---	---	---	---	--	---	---	---	--	--

图 2.9 字符串“IF X > 0 THEN CALL OK”的向量表示

图中每个方格代表一个存储单元。没写字母、数字的字节为空字符, “;”为字符串的结束符。整个字符串需 20 个存储字单元。

向量表示法是最简单、最节省存储空间的方法。但是, 当字符串进行删除(Delete)和插入(Insert)运算时, 删除或插入子串后的字符要全部重新分配存储空间。由于目前计算机多具有内存数据成组移动的指令, 所以这种结构的字符串移动时速度较快。向量表示是现代计算机中最常采用的字符串表示法。

(2) 向前串表表示法

向前串表表示法, 简称串表表示法, 在此表示法中, 串的每个字符后有一个链接字, 用以指出下一个字符的地址。因此, 串表表示法不要求串中字符在物理上是相邻的, 原则上讲字符可以任意安排。仍以字符串“IF X > 0 THEN CALL OK”为例, 其存储分配如图 2.10(a)所示。一个存储字包括字符和链接字两个部分, 其中链接字占据一个存储字的大部分空间。

如一个存储字有 32 位, 存放一个字符需 8 位, 链接字占用 24 位。在这种情况下, 存放串的有效信息的利用率只占 25%, 这是串表表示法的最大缺点。但是, 它也有吸引人的优点, 即在字符中进行删除、插入运算时非常方便。

如要将上述字符串修改为“IF SIZE > 0 THEN CALL OK”, 实际只需修改相应字符的链接字即可。具体的修改操作为: 首先把字符 X 修改为 S, 然后插入字符串“IZE”, 接着修改 S 的链接字, 使其指向“IZE”中的“I”, 最后将“IZE”中字符“E”的链接字指向字符“>”的地址。图 2.10(b)中标斜体字的单元涉及到这些修改。

A+0	I	A+1	A+0	I	A+1
A+1	F	A+2	A+1	F	A+2
A+2		A+3	A+2		A+3
A+3	X	A+4	A+3	S	A+21
A+4	>	A+5	A+4	>	A+5
A+5	0	A+6	A+5	0	A+6
A+6		A+7	A+6		A+7
A+7	T	A+8	A+7	T	A+8
⋮			⋮		
A+17	O	A+18	A+17	O	A+18
A+18	K	A+19	A+18	K	A+19
A+19	;	~	A+19	;	~
A+20			A+21	I	A+22
			A+22	Z	A+23
			A+23	E	A+4

(a) 修改前

(b) 修改后

图 2.10 向前串表表示法存储分配及修改操作示例

Pentium 中没有字符串数据表示,但 Pentium 支持一种称为串(string)的数据类型来表示字符,包括:

- 位串(bit string),可从任何位位置开始一个位串,最长可有 $2^{32} - 1$ 位;
- 字节串(byte string),可包含字节、字、双字,最长可有 $2^{32} - 1$ 字节(4 GB)。显然,字符串包含其中。

2.1.3 堆栈数据表示

在计算机系统软件和程序设计技术中较为广泛应用的一种数据结构是堆栈(Stack)。堆栈实际上是一种数据有序表(Ordered Set of Data),某个时刻其中只有一个数据能够被访问,这个数据就是栈顶(Top),通常需要一个专用的寄存器来指明,这个寄存器叫做堆栈指针(Stack Point,SP)。对堆栈数据处理是依据后进先出的原则,所以堆栈也称为后进先出表(Last In First Out,LIFO)。堆栈只有一个数据出入的端口,数据存入堆栈称为压栈(PUSH),数据从堆栈取出称为弹栈(POP)。当向堆栈存入数据时,新数据放在栈顶;当从堆栈取出数据时,从栈顶单元取数据,并使栈顶后的第二个单元数据成为栈顶。可见,数据的存取只能对栈顶进行操作。因而表征堆栈数据的重要参数是栈顶单元地址,一般由堆栈指针 SP 给

出。堆栈数据压栈与弹栈过程如图 2.11 所示。

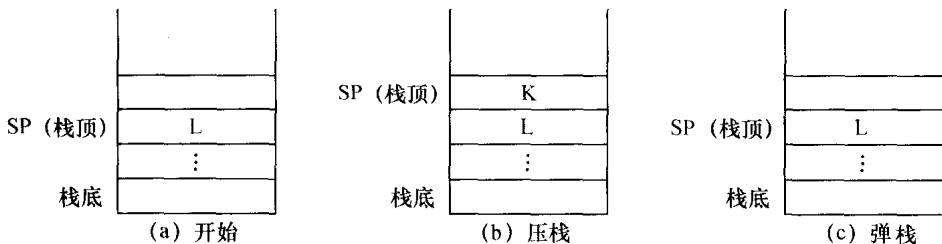


图 2.11 基本堆栈操作——压栈和弹栈

图中开始时堆栈中有数据“L”，然后进行数据“K”的压栈和弹栈。栈顶单元地址 SP 是随数据的压栈与弹栈而改变的。SP 的变化与堆栈的生长方向有关。如果堆栈的栈顶地址编码大于栈底地址编码，堆栈称为向上生长的堆栈；若栈顶地址编码小于栈底地址编码，则称为向下生长的堆栈。对于向上生长的堆栈，压栈时， $SP = SP + 1$ ，弹栈时， $SP = SP - 1$ ；而对于向下生长的堆栈则相反，压栈时， $SP = SP - 1$ ，弹栈时， $SP = SP + 1$ 。

对于堆栈的操作，除了上述压栈和弹栈，往往还包括一些运算类指令。表 2.1 列出了一些面向堆栈的操作指令，其中的一元操作和二元操作是为了与堆栈交换数据而设计的，并没有运算功能。堆栈数据运算的重要特点是指令只需指出进行什么样的操作，而无需指出操作数地址，因为地址就是栈顶，所以堆栈运算指令是零地址指令。

表 2.1 面向堆栈的操作指令

压栈(PUSH)	在栈顶增加一个新元素
弹栈(POP)	从栈顶取走一个元素
一元操作	对栈顶元素进行操作以后，用结果替换栈顶元素
二元操作	对栈顶的两个元素进行操作以后，用结果替换栈顶的两个元素

有关堆栈的操作指令在执行时，总是伴随着堆栈指示器的修改： $SP = SP \pm 1$ ，若以某存储单元作为 SP，对它进行修改操作需要两次访问主存。

为提高堆栈操作速度，硬件应对堆栈提供有力的支持。目前多数计算机的堆栈是在主存中开辟一个堆栈区，堆栈指针及堆栈上下界标志用寄存器实现，如图 2.12(a)所示。其中 SP 为栈顶指针，B(Bottom)寄存器为堆栈下界(栈底)指针，L(Limit)寄存器为堆栈上界(栈顶)指针。设置堆栈上下界寄存器的目的是为避免堆栈区与其他存储区混淆。对于向上生长的堆栈，在压栈时， $SP > L$ 称为堆栈“上溢”，这是不允许的；在弹栈时， $SP < B$ 称为堆栈“下溢”，也是不允许的。因此，堆栈数据操作时，不但 SP 要修改，还需要判界。因此采用软件方法实现速度较慢，若采用硬件方法实现堆栈指针修改 $SP = SP \pm 1$ ，以及判界操作，将会大大

加速堆栈数据操作速度。

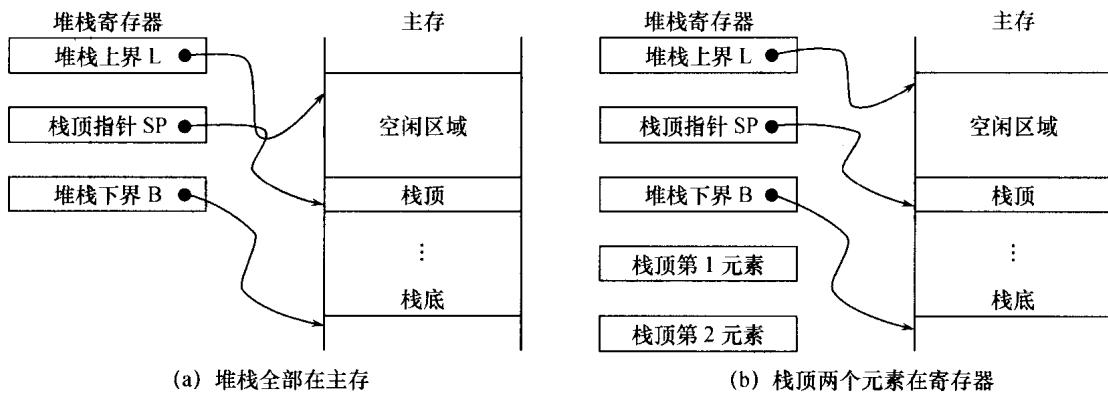


图 2.12 典型的堆栈组织

为加快堆栈操作,还可以将栈顶部的两个元素取到寄存器中,如图 2.12(b)所示。此时栈顶指针 SP 指向第三个栈元素。有的堆栈型机器,如 HP 3000,堆栈的头四个元素存于寄存器中,指令中对栈顶和次栈顶的数据操作可直接对寄存器进行。这种设置栈顶元素寄存器的硬件结构在面向堆栈运算的计算机中是普遍采用的。

综合起来,硬件对堆栈提供的支持有:堆栈指针 SP 及其操作时的修改,堆栈上、下界寄存器及其判界,以及栈顶元素寄存器三种。除了一些专门设计的面向堆栈进行计算的计算机外,目前,堆栈在通用计算机中也普遍使用,主要应用于系统程序设计,如子程序的处理、表达式的计算等,而在用户程序中使用较少。

2.1.4 向量数据表示^{*}

在计算机科学计算中,如多项式求解,地球物理勘探数据处理,空气动力学计算等,广泛使用向量数据结构。20世纪80年代研制的各类巨型计算机,尤其是数组处理机等多以向量数据作为主要的高效数据处理对象,向量在那些巨型计算机上已经成为主要的数据表示方法。在某些大型通用计算机中,为了高效处理向量数据,也专门设置向量处理部件。

在没有向量数据表示的计算机中,处理向量数据运算是通过软件方法实现的。例如,两个向量加法,利用 Fortran 语言的 DO 语句可实现:

```
DO 10 I = 1,50
10C(I) = A(I) + B(I)
```

这段源程序经编译后,机器实际执行的是一段循环程序。若机器有向量数据表示,只需要一条向量加法指令即可实现:

$$C(1,50) = A(1,50) + B(1,50)$$

从上例看,如果使用 DO 语句的循环程序实现向量加法,循环体内的指令就要取 50 次,

而有向量数据表示的机器只需取一条向量加法指令。这样,就节省了大量访存取指令的时间,大大提高了向量运算速度。

1. 基本概念

向量数据是一组元素的有序集合。这些元素具有以下特点:

- 均匀性。所有元素必须具有相同的结构,即有相同的数据类型和相同的字长。
- 有序性。各元素是有序的,元素之间的次序不能改变。

因此,向量数据在存储器中的物理结构应该是各元素以等字长、等间距顺序存储。描述向量数据的参量应有:向量起始地址,即向量第一个元素的存储地址;向量长度,即向量元素的个数;向量间距,即各元素地址的间距。若向量起始地址为 D ,向量长度为 L ,向量间距为 Δ ,则向量数据在存储中的结构如图 2.13 所示。

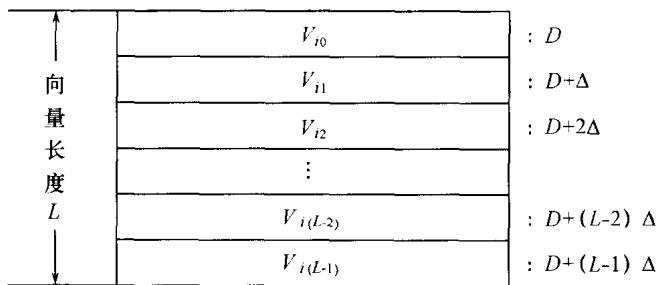


图 2.13 向量数据类型的存储结构

在指令调用向量数据时,根据给出的向量起始地址 D 可以确定向量第一个元素 V_{i0} 的地址,而向量第二元素的地址即为向量起始地址和给出的间距之和 $D + \Delta$ 。同理,向量第 3 个元素的地址为 $D + 2\Delta$,等等。根据给定的向量长度,可以确定向量的元素个数,向量最后一个元素的地址应是 $D + (L - 1)\Delta$ 。

在向量处理机中,为加速向量运算速度,需要集中大量的数据,这就给存储器的流量带来极大的压力。在绝大多数向量计算机中,设置了向量寄存器组。在向量运算之前,先将存储器中的向量数据成组地传送到向量寄存器;在向量运算时,不再与存储器打交道,从而缓解了存储器的压力。向量运算指令直接调用向量寄存器中向量各元素参加运算,并将向量运算结果存于向量寄存器。

图 2.14 是典型的向量寄存器的结构。图中有 8 个向量寄存器 $V_0 \sim V_7$,每个向量寄存器有 64 个元素 $V_{i,0} \sim V_{i,64}$,每个元素字长为 64 位。它们用来作为向量运算的源操作数和目的操作数数寄存器。可见,向量数据类型是一种硬件代价非常高的技术手段。已有的向量巨型计算机表明,向量数据类型对于提高计算机向量处理的能力是非常有效的。

2. 稀疏向量

在向量运算中,常常会遇到零元素个数多于非零元素个数的向量。一般来说,如果数组

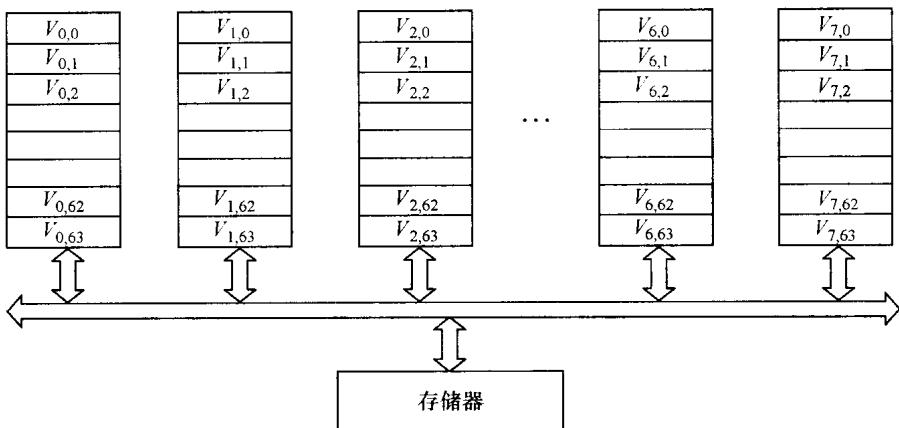
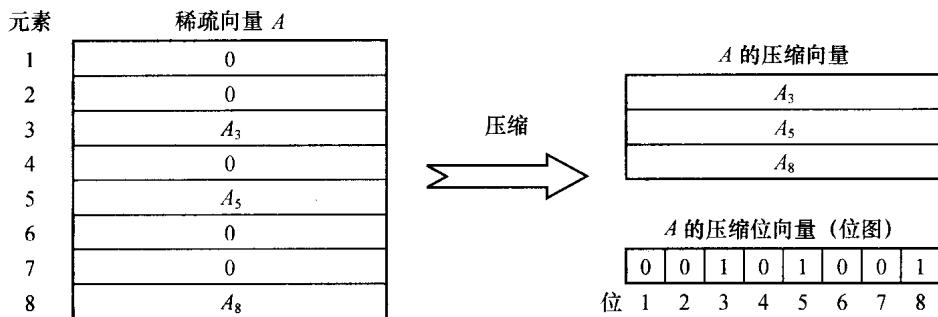


图 2.14 向量寄存器的结构以及与存储器的关系

规模变大时,非零元素为全体元素的百分之几以下也是很寻常的事。若把零元素很多的向量与通常向量作同样处理,一则会使大量零元素占据存储资源,降低了存储器的使用效率;二则对零元素的处理也会浪费机器时间。因此,有必要采取措施来改变这种状况。

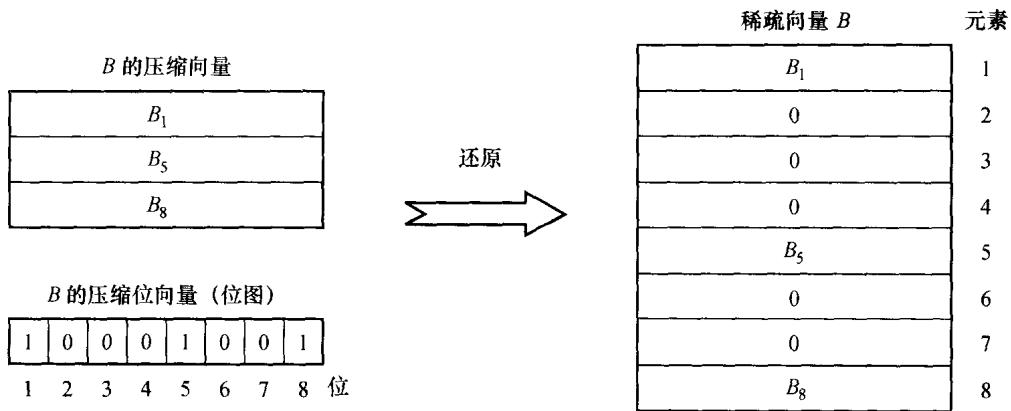
通常把零元素很多,非零元素稀少的向量称为“稀疏向量”。对这种类型的向量,一般利用压缩还原技术把它分为两部分:一部分为非零元素压缩组成的向量——压缩向量,另一部分为表示非零元素在原来向量哪个位置的压缩位向量(位图),如图 2.15 所示。

图 2.15 稀疏向量 A 的压缩

图中,稀疏向量 A 经压缩技术形成 A 的压缩向量和压缩位向量。 A 位向量的位数等于稀疏向量 A 元素个数,从左至右的序号与稀疏向量元素序号一一对应:第 i 个元素是非零元素,则位向量第 i 位为“1”,否则为“0”。压缩向量存于存储器中都是非零元素,压缩位向量只占用一个存储单元。所以,稀疏向量经压缩后可以节省大量存储空间。

图 2.16 给出了上述过程的反过程,即还原过程,描述了稀疏向量 B 的压缩向量和压缩位向量,经过还原技术复原稀疏向量 B 的情况。

有的向量处理机,如 CDC STAR - 100 计算机等,设有稀疏向量运算指令,利用压缩位向

图 2.16 稀疏向量 B 的还原

量作为向量运算的控制向量,快速自动地处理零元素的运算。图 2.17 给出了上述两个稀疏向量 A 与 B 进行相加运算和相乘运算的情况。由图可见,稀疏向量相加时,压缩位向量 A 与 B 作逻辑或运算可得和向量 C 的压缩位向量 C 。两个位向量对应位都是“1”时,需进行元素的加法运算;对应位只有一个“1”时,只进行数据传送;对应位都是“0”时,不进行运算,对应位的和为“0”。将向量的乘法定义为向量对应元素的运算,即两个稀疏向量相乘时,两个压缩位向量作逻辑与运算可得积向量 C 的压缩位向量 C ;两个位向量对应位都是“1”时,才进行对应元素的乘法运算,否则对应位的积为零。

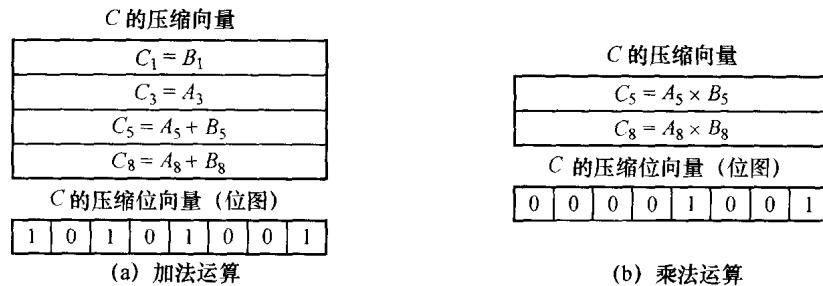


图 2.17 稀疏向量的运算

综上所述,与一般向量相比,稀疏向量运算可以大大提高向量运算的处理速度。因此,稀疏向量一般可作为独立的向量数据表示加以研究和处理。

2.1.5 数据表示小结

数据表示对计算机设计的影响很大,它甚至可以影响一类计算机的使用和市场。本节研究的就是一些常见的数据表示。

数据表示是一个变化和发展的领域。早期计算机,由于硬件成本昂贵,只支持简单的定点数据类型,机器也相应的只有定点运算和逻辑运算指令。定点数的表示范围小,使用很不方便。随着计算机在数值计算领域中应用需求的发展,计算机中很快又增加了浮点数据表示,相应的增加了浮点运算指令,机器也设置了浮点运算部件和相应的控制线路。

随着计算机在商业和事务处理领域获得了广泛应用,增加十进制数据表示,可以使计算机的效率大幅度提高。计算机硬件可以直接识别十进制数,需要增加十进制运算指令,其硬件也要支持十进制数据的各种操作。

随着计算机硬件成本的进一步降低,以及计算机系统软件的发展和应用领域需求的扩大,许多计算机又增设了各种新的数据表示,如字符串、堆栈等,而有些大型机、巨型机还设置了向量。这些数据表示提高了计算机的性能,同时也提高了软件设计和生产的效率。

本节所讲的各种数据类型的表现形式和《数据结构》中的内容虽然是相同的,但一种数据类型可以称之为数据表示,是因为计算机中存在可以直接处理这种类型数据的指令,有相应硬件支持,这也就是硬件可以直接识别这种数据类型。如果一种数据类型不能被计算机直接识别(比如矩阵),而在程序中又要使用它,那么就需要在计算机现有数据表示的基础上,运用数据结构的知识把它构造出来,供程序使用。这也就是数据表示和数据结构的区别。

计算机如何确定其数据表示,是一个很复杂的系统设计问题。它取决于面向问题的算法、机器硬件功能与软件功能的分配以及机器性能价格比等诸多因素。如何权衡这些因素,是“计算机系统结构”课程中要研究的内容。

2.2 指令格式

在计算机内部,指令表示为一个二进制位串,称为指令字。图 2.1 给出了一个简单指令的例子。对应于指令的组成要素——操作码和操作数地址,指令字划分为若干个字段。下面详细讨论如何对指令字中各字段进行分配,即指令格式。

一条指令中,操作码字段不能少于一个,而地址码字段则可以是零个、一个或多个,实际计算机指令的操作数地址码也是具有多样性的。根据地址码的数量,通常称没有操作数地址码的指令为零地址指令,相应也就有 1 地址指令、2 地址指令和 3 地址指令等。图 2.18 给出了一些指令格式的实例。

设计指令格式,一般涉及 3 个问题:指令长度、操作码结构和地址码结构。

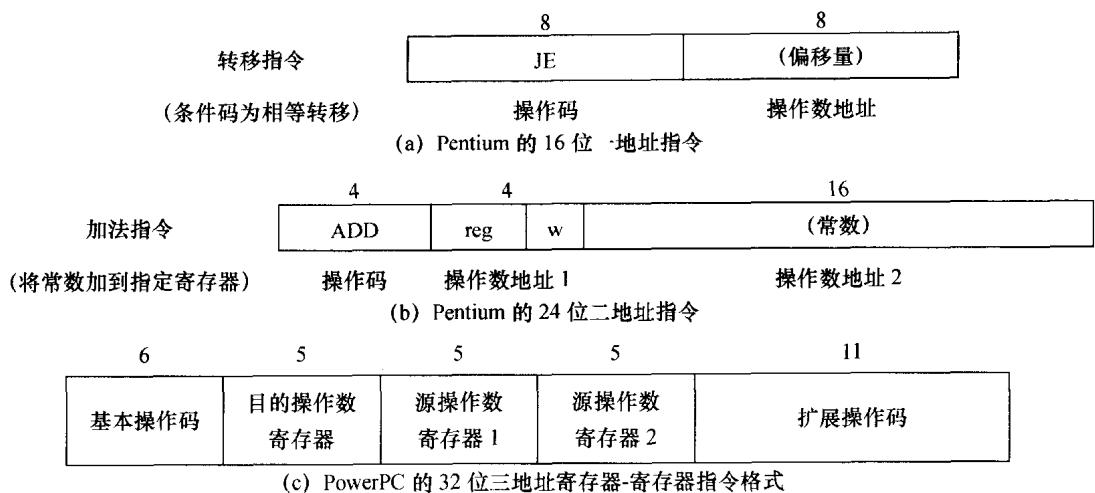


图 2.18 指令格式的一些实例

2.2.1 指令长度

指令长度即一条指令中包含的二进制代码的位数。

机器字长是计算机能直接处理的二进制数据的位数。它表示计算机内部数据通路和数据寄存器的宽度,是各种因素综合表现的属性。由于指令一般存放在主存中,被 CPU 执行时存放在指令寄存器(IR)中,因而指令长度要受到机器结构的限制。

通常指令长度与机器字长有简单的倍数关系,有的等于机器字长,有的大于或小于机器字长。指令字长度等于机器字长度的指令,称为单字长指令;指令字长度等于半个机器字长度的指令,称为半字长指令;指令字长度等于两个机器字长度的指令,称为双字长指令。为了便于处理字符数据,尽可能充分利用存储空间,减少指令和数据的存储时间,现代计算机的机器字长、存储器宽度和 I/O 传输宽度几乎都是字节的整数倍(如 1、2、4 或 8 倍),这个限制同样适合于指令长度。

从访问存储器和指令复杂性的角度来看,短指令比长指令好。短指令能够节省存储空间,减少访问存储器次数,具有快的执行速度。但短指令也有其不可克服的局限性。由于指令中包括的信息少,因而指令功能较弱。现代计算机仍然希望发展长指令,长指令的功能比较强,便于程序设计,但长指令又可能造成利用率不高的浪费。为了合理地安排存储空间,并使指令能表达较丰富的含义,同一台计算机的指令系统通常采用变长措施,即有的与机器字长相等,有的是机器字长的若干倍。例如,IBM 370 机,它的指令长度有 16 位(半字)的,有 32 位(单字)的,有 48 位(一个半字)的。Pentium 也是变长指令格式结构,包括从 8 位(1/4 个字)到 128 位(4 个字)的多种形式。变长指令格式结构灵活,能充分利用指令长度,但指令的控制复杂。当采用变长指令格式时,通常把最常用的指令设计成短指令,以便节省存储

空间和提高指令的执行速度。

在一个指令系统中,如果所有指令字长度是相等的,则称为等长指令字结构。等长指令字结构具有结构简单,便于实现等优点。

目前还有一种非常适合超大规模集成电路实现的计算机指令结构,称为超长指令字(Very Long Instruction Word, VLIW)。这种结构的指令字长度在 100 位以上,在一个指令字中包含一组多种类型并可以同时执行的指令,借助于集成电路技术的支持,在 CPU 中设计大量的功能部件同时执行这一组指令,使系统达到很高的性能。大量研究已经表明,通过目前的计算机系统设计和软件设计技术,采用这种指令字可以非常有效地提高处理器的性能。俄罗斯的 E2K 芯片、Transmeta 的 Crusoe 芯片、Intel 的 Iantum 芯片都采用了该技术或在此基础上改进的技术。

2.2.2 操作码结构

操作码是指令中表示操作性质的部分,若所有指令的操作码长度相同,该长度决定了指令系统中完成不同操作的指令条数。例如某机器的操作码长度为 k 位,则它最多只能有 2^k 条不同指令。操作码位数越多,所能表示的操作种类就越多。所以必须从指令中划出足够的位数表示全部操作功能。但是,当指令长度一定时,地址码与操作码又是相互制约的,越长的操作码字段意味着地址码字段越短,这将影响寻址能力。目前在指令操作码设计上主要可分为固定长度操作码和可变长度操作码两种情况。

(1) 固定长度操作码

操作码的长度固定,且集中放在指令字的一个字段中,其余部分全部用于地址码。例如 IBM 370SM 和 VAX - 11,操作码的长度均为 8 位,可表示 256 种操作。这种格式的优点是规则,有利于简化硬件译码逻辑,减少指令译码时间。在字长较长的大中型机以及超级小型机上广泛采用。

(2) 可变长度操作码

又称为扩展操作码,操作码的长度允许有几种不同的选择,对地址数少的指令允许操作码长些,对地址数多的指令,则操作码就短些。

假设某机器的指令长度为 16 位,包括基本操作码 4 位和三个地址段 A_1 、 A_2 、 A_3 ,每个地址段长 4 位,其格式如图 2.19 所示:



图 2.19 一种 16 位字长的三地址指令格式

4 位基本操作码有 16 种组合,若全部用于表示三地址指令,则最多有 16 条。但是,如果三地址指令仅需 15 条,则剩下的编码可以用于表示二地址指令,操作码可扩展为 8 位。根

据上述思想,可以继续向下扩展。如果三地址指令需要 15 条,二地址指令需要 15 条,一地址指令需 15 条,零地址指令需 16 条,共 61 条指令,则扩展的方法如图 2.20 所示。

	0000	xxxx	xxxx	xxxx
	0001	xxxx	xxxx	xxxx
	⋮	⋮	⋮	⋮
	1101	xxxx	xxxx	xxxx
	1110	xxxx	xxxx	xxxx
	1111	0000	xxxx	xxxx
	1111	0001	xxxx	xxxx
	⋮	⋮	⋮	⋮
	1111	1101	xxxx	xxxx
	1111	1110	xxxx	xxxx
	1111	1111	0000	xxxx
	1111	1111	0001	xxxx
	⋮	⋮	⋮	⋮
	1111	1111	1101	xxxx
	1111	1111	1110	xxxx
	1111	1111	1111	0000
	1111	1111	1111	0001
	⋮	⋮	⋮	⋮
	1111	1111	1111	1110
	1111	1111	1111	1111

图 2.20 指令的扩展

除上述扩展方法以外,还有其他多种扩展方法。如可以形成 15 条三地址指令,14 条二地址指令,31 条一地址指令和 16 条零地址指令,共 76 条指令。实际机器可采用多种灵活的扩展方式,如 PDP-11 的指令格式中,操作码的长度有 4、8、10、13 和 16 位五种选择。

使用扩展操作码技术的一个重要原则是使用频度高的指令应分配短的操作码,使用频度低的指令相应地分配较长的操作码。这样不仅可以有效地缩短操作码在程序中的总位数,节省存储空间,而且缩短了常用指令的译码时间,因而有利于提高程序的运行速度。

扩展操作码技术是一种重要的指令优化技术。它可以缩短指令的长度,减少程序的总位数以及增加指令字所能表示的操作信息。当然,扩展操作码译码复杂,在硬件设计和实现上要困难些。图 2.18(c)所示的 PowerPC 指令就是一种采用扩展码的形式。

2.2.3 地址码结构

地址码的结构涉及到地址个数和寻址方式等因素,这里只讨论地址个数的问题,寻址方式将在 2.3 节中讨论。

指令中操作数地址的个数是指令系统的一个重要特征。在目前的计算机中,一般不超

过3个。表2.2列出了地址数为0~3的各指令的操作情况,其中假设所有指令的操作都是二元操作,需要3个操作数,即两个源操作数和一个结果操作数。

在三地址指令中,结果操作数地址A独立给出,因此,这种指令不会破坏源操作数。在二地址指令中,有一个操作数地址A的作用是双重的,既是源操作数地址,又是结果操作数地址。因此,这种指令会破坏A中的源操作数。若需保存源操作数,需在该指令前先用传送指令将其保存。在一地址指令中,这个地址指出了一个源操作数,而另一个源操作数地址及结果操作数地址则是隐含的,即累加器AC。零地址指令运用于堆栈的情况,其操作数地址都是隐含的。它对堆栈最顶部的两个数进行操作,结果放回栈顶。

表2.3、表2.4和表2.5分别给出了在三地址、二地址和一地址指令的不同情况下,计算表达式: $Y = (A - B) \div (C + D \times E)$ 。所用的典型指令序列,其指令条数分别为4、6和8。

表2.2 不同地址数的操作

地址个数	指 令	操 作
3	OP A,B,C	$A \leftarrow (B) OP(C)$
2	OP A,B	$A \leftarrow (A) OP(B)$
1	OP A	$AC \leftarrow (AC) OP(A)$
0	OP	$SP \leftarrow (SP) OP(SP-1)$

注: AC——累加器, SP——栈顶指针

表2.3 三地址指令

指 令	操 作
SUB Y,A,B	$Y \leftarrow (A) - (B)$
MUL T,D,E	$T \leftarrow (D) \times (E)$
ADD T,T,C	$T \leftarrow (T) + (C)$
DIV Y,Y,T	$Y \leftarrow (Y) \div (T)$

表2.4 二地址指令

指 令	操 作
MOVE Y,A	$Y \leftarrow (A)$
SUB Y,B	$Y \leftarrow (Y) - (B)$
MOVE T,D	$T \leftarrow (D)$
MUL T,E	$T \leftarrow (T) \times (E)$
ADD T,C	$T \leftarrow (T) + (C)$
DIV Y,T	$Y \leftarrow (Y) \div (T)$

表2.5 一地址指令

指 令	操 作
LOAD D	$AC \leftarrow (D)$
MUL E	$AC \leftarrow (AC) \times (E)$
ADD C	$AC \leftarrow (AC) + (C)$
STOR Y	$Y \leftarrow (AC)$
LOAD A	$AC \leftarrow (A)$
SUB B	$AC \leftarrow (AC) - (B)$
DIV Y	$AC \leftarrow (AC) \div (Y)$
STORE Y	$Y \leftarrow (AC)$

指令中地址的个数是指令系统设计中的一个基本问题。地址数越少,指令的功能就越基本、越简单,CPU的复杂性也就越低,而且指令的长度也越短。但另一方面,程序中的指令条数也就越多,这通常会增加程序的执行时间和复杂度。实际上,地址个数的选择在很大程度上依赖于指令系统的结构。一般来说,堆栈结构的指令系统主要采用零地址指令,累加器结构的指令系统主要采用一地址指令。当代的计算机大多既具有二地址指令又具有三地址指令。

地址个数的选择还与地址的类型有关。对于固定的字长来说,地址越短,其个数就可以越多。寄存器的地址比存储器的地址短得多,所以,操作数在寄存器中的指令易采用三地址结构。三地址指令因不会破坏源操作数,使用比较简单,便于代码生成。目前的计算机中已普遍采用了操作数在寄存器中的三地址指令。

2.3 寻址技术

寻址技术(Addressing),又称为寻址方式(Addressing Mode),是确定操作数地址的技术,它是计算机中硬件对软件最早提供支持的技术之一。

寻址技术出现的目的有两个:其一,希望能够灵活地访问到大范围的存储器区域,这就需要提供较长的地址,但是指令字中的地址域的位数却又是非常有限的,对于虚拟存储器这个问题更加突出。其二,高级语言和很多种数据结构,如循环语句和数组结构需要有效的地址变换以提高访问数据的灵活性和有效性。因此,必须采用寻址技术来解决这些问题。

在寻址时,通常将指令字中给出的地址称为形式地址(Formal Address),又称为逻辑地址(Logic Address)。形式地址按一定的规则变换后,得到能直接访问操作数的地址,这个地址称为有效地址(Effective Address)。因此,寻址技术也就是指从形式地址产生有效地址的方法。指令系统具有哪几种寻址方式,能否为用户编程提供较强的寻址能力,是指令系统设计的关键问题之一。寻址能力主要反映在寻址方式的多样性、灵活性以及访问地址范围的大小上。计算机系统对所支持的寻址方式选择,一方面需考虑寻址能力,另一方面则还要考虑地址计算的复杂度。因为后者涉及硬件实现的复杂度和计算机的速度。

2.3.1 基本寻址方式

本节将详细讨论7种基本寻址方式:立即数寻址(Immediate)、直接寻址(Direct)、间接寻址(Indirect)、寄存器寻址(Register)、寄存器间接寻址(Register Indirect)、偏移寻址(Displacement)和堆栈寻址(Stack)。通过对这些基本寻址技术的讨论,了解寻址技术的工作原理。除了堆栈方式,前6种寻址方式的地址转换过程在图2.21中给出了示意说明。

在图2.21以及下面的讨论中,将使用如下约定:

A——形式地址。指令地址字段中访问存储器的地址

R——形式地址。指令地址字段中访问寄存器的地址

EA——有效地址。包含有被访问操作数的实际单元

(X)——表示单元X中的内容。

1. 立即数寻址

立即数寻址如图2.21(a)所示。它是最简单的寻址方式,其指令的地址字段给出的不是操作数的地址,而是操作数本身,即:

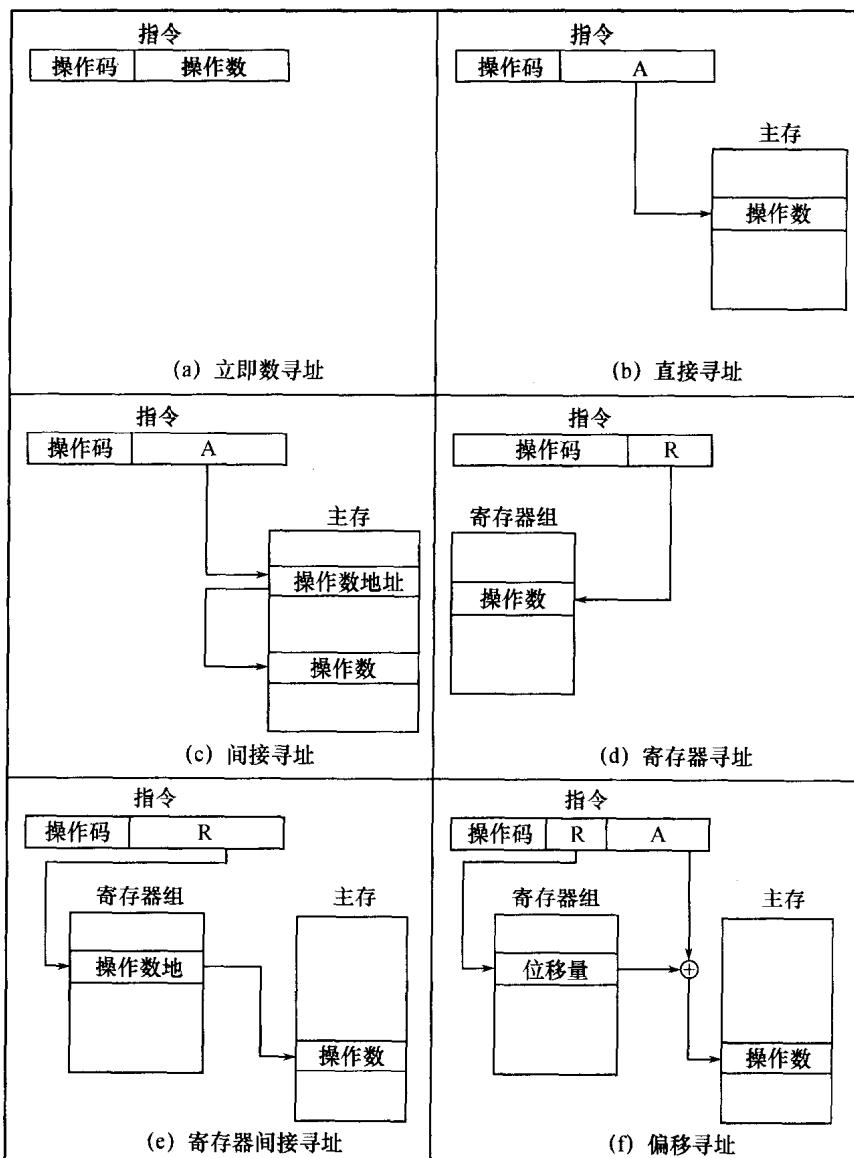


图 2.21 六种寻址方式的地址转换过程

操作数 = A

立即数寻址方式的特点是指令执行时间很短,因为它不需要访问内存去取操作数,从而节省了访存时间,但是操作数的大小受地址字段长度的限制。

这种方式通常用于提供一个常数或给某个变量赋初值。典型的立即数是采用补码的形式存储的,有一个符号位。由于立即数一般较短,在装入寄存器时,往往需要根据寄存器进行数据符号扩展。

立即数寻址方式在目前的计算机中还在广泛使用。

2. 直接寻址

直接寻址方式的工作过程如图 2.21(b) 所示。由于操作数的地址直接给出，不需要经过变换或运算，所以称之为直接寻址方式。它也是一种非常简单的寻址方式，同时也是最古老的寻址方式。在这种寻址方式中，指令的地址码部分就是操作数的有效地址，即：

$$EA = A$$

直接寻址优点是处理简单、直接，不足之处是寻址空间受到指令的地址字段长度的限制。

直接寻址方式在早期的计算机中普遍采用，现在只有一些规模较小的计算机中使用，例如 8 位计算机和一些 16 位计算机。

3. 间接寻址

由于指令中地址字段的长度较小，限制了直接寻址的空间范围。解决这个问题的方法之一就是找一些特定的存储单元保存足够长的地址，而将指令中的地址字段指向保存地址的存储单元，这就是间接寻址，其工作过程如图 2.21(c) 所示。可见，间接寻址是相对于直接寻址而言的。间接寻址时，形式地址 A 不是操作数的真正地址，而是操作数地址所在的存储单元的指示器，或者说 A 单元的内容才是操作数的有效地址。即：

$$EA = (A)$$

由于(A)单元的地址长度比指令中 A 字段的地址长度更容易调整，所以间接寻址的存储器访问空间可以根据需要，比较容易扩大。

在实现寻址方式时，通常可以使用寻址特征位来标志寻址方式的种类。例如若指令系统只具有直接寻址和间接寻址，则实际的指令格式如图 2.22 所示。



图 2.22 一种支持直接寻址和间接寻址的指令格式

若寻址特征位 $I = 0$ ，表示直接寻址；若 $I = 1$ ，则表示间接寻址。

间接寻址又分为一次间接和多次间接。一次间接是指形式地址 A 是操作数地址的地址，而多次间接是指这种间接变换在两次或两次以上：

二次间接寻址： $EA = ((A))$

多次间接寻址： $EA = (\dots(A)\dots)$

间接寻址的优点是寻址空间大，而且灵活，便于编程。其缺点是至少需要两次访存才能取得操作数，执行速度慢。

间接寻址方式也是目前使用比较多的寻址方式。

4. 寄存器寻址

当操作数不放在内存中，而是放在 CPU 的通用寄存器中时，可采用寄存器寻址方式。

寄存器寻址过程如图 2.21(d) 所示。这种寻址方式类似于直接寻址,惟一的不同之处是此时指令中给出的操作数地址不是内存的地址单元号,而是通用寄存器的编号。

$$EA = R$$

寄存器寻址优点是只需要很短的地址字段,无需访问存储器。对于典型的寄存器寻址,其地址字段长度往往只有 3~8 位,可以访问 8~256 个通用寄存器。因为访问 CPU 内寄存器的时间比访问主存短得多,所以采用这种寻址方式的指令执行速度较快。寄存器寻址的缺点是寄存器的地址空间很有限,这主要是因为一般机器中只设置几个到几百个可以编程使用的通用寄存器。例如,Pentium 为应用编程提供了 24 个寄存器,包括 8 个 32 位的通用寄存器。

由于寄存器寻址在速度上的优势,因此通过使用寄存器来提高应用程序的性能是目前最常用的技术手段。但是,只有当存在寄存器中的数据能多次被使用才能有效地利用寄存器,提高机器的效率。程序员可以决定哪些值应存于主存中,哪些值应存于寄存器中,但是这样会给程序员带来许多麻烦。对于高级语言,目前这项工作主要是由编译器优化技术完成。如何使用好寄存器,一直是计算机编译技术研究的重点之一。

寄存器寻址方式一直是使用最多的寻址方式之一。

5. 寄存器间接寻址

寄存器间接寻址的过程如图 2.21(e) 所示。这种寻址方式与间接寻址类似,所不同的是,指令中给出的是寄存器地址,而该寄存器的内容才是所需操作数的地址即:

$$EA = (R)$$

寄存器间接寻址的特点与一次间接寻址类似,可以具有较大的寻址空间,但它比一次间接寻址少访问存储器一次。

寄存器间接寻址方式的使用目前也比较普遍。

6. 偏移寻址

偏移寻址组合了直接寻址和寄存器间接寻址两种方式,因此功能强大。在指令不作任何修改的情况下,它可以通过修改寄存器的内容来改变访问单元地址的位移,因此称之为偏移寻址。偏移寻址的寻址过程如图 2.21(f) 所示,其有效地址计算为:

$$EA = (R) + A$$

实际上,根据寄存器间接寻址所使用的寄存器的不同,偏移寻址的方式很多。下面讨论较为常见的三种偏移寻址:相对寻址 (Relative Addressing)、基址寻址 (Base - Register Addressing) 和变址寻址 (Index Addressing)。

(1) 相对寻址

相对寻址,是把程序计数器 PC 作为参照寄存器,将当前 PC 的值加上该指令地址字段的值而形成操作数的有效地址。即:

$$EA = (PC) + A$$

因此，“相对”寻址就是相对于当前指令的地址。采用相对寻址，形式地址 A 通常称为偏移量(Offset)，操作数的有效地址在相对当前指令地址的一个上下范围内浮动。

如果大多数操作数存储在当前执行的指令附近，使用相对寻址就可以节省指令中地址的位数，无需用绝对地址。这里利用了程序访问的局部性的概念，简单说就是，程序执行时涉及的指令和数据的地址不是随机分布，而是相对簇集成块。关于这一点，将在“计算机系统结构”课程中详细说明。

(2) 基址寻址

基址寻址，实际上是相对于基址寄存器(Base Register, BR)的偏移寻址。基址寄存器中存放基址。基地址(Base Address)就是当前程序访存操作的一个参考基准地址。指令中的地址字段给出的是相对于基地址的偏移量 A，操作数的有效地址是把基址寄存器的内容加上指令字中的形式地址得到。即：

$$EA = (BR) + A$$

不同的 A 值，指出了以(BR)为基地址的一个数据块中的不同单元。

采用基址寻址方式，程序员可以不用考虑程序加载到主存中的位置。利用这种寻址方式，可以方便地实现段式存储管理，例如 Pentium 的段式访存。关于存储管理，“操作系统”课程将会详细研究。

除此以外，基地址的访存结构还是对存储器区域进行保护的基础。这种存储器在受到保护的情况下，一般用户程序不能修改基址寄存器，只有具有相应权限的系统程序才能够设置和修改基址寄存器。

(3) 变址寻址

变址寻址是相对于变址寄存器 X(Index Register)的偏移寻址。其有效地址是把变址寄存器的内容加上指令字中的形式地址得到。即：

$$EA = (X) + A$$

程序是通过改变变址寄存器 X 的内容来访问主存的。变址寻址的一个重要用途就是，通过有规律地改变 X 的内容，遍历从基准地址 A 开始的一片单元，从而可以通过循环执行同一条指令实现对一批数据的处理。例如要访问从 A 开始的一片单元，A、A+1、A+2、……。只要 X 的初值置为 0，然后循环，每循环一次 X 的内容加 1 即可。由于变址寻址在循环中使用的非常多，而且往往是每次使用后需要将变址寄存器的内容加 1。为了适应这种情况，很多计算机在变址寻址的基础上增加了一种自变址寻址(Autoindexing)，在每次变址寻址后增加对 X 的增量操作 $X = X + 1$ 。

在形式上以及计算操作数的有效地址的方法上，变址寻址和基址寻址是相似的，但两者有着不同的特点和用途。基址寻址方式中，基址寄存器的内容是不变的，形式地址 A 是相对该基准地址的偏移量。由于操作码的限制，基址寻址中的偏移量位数较短。基址寻址主要是解决程序逻辑空间与存储器物理空间的无关性问题。对于变址寻址来说，形式地址 A 给

出的是一个存储器地址基准,变址寄存器 X 中存放的是相对于该基准地址的偏移量,变址寻址中的偏移量位数相对较大。变址寻址主要是为了可以编写出高效地访问一片存储空间的程序。

基址寻址和变址寻址的优点是提高了寻址的灵活性,扩大了寻址范围,但同时增加了实现的硬件复杂性:除了指令译码的复杂性增加,往往还需要专门硬件支持,例如专用寄存器、地址加法器等。

如果机器中的基址寄存器和变址寄存器是专用的寄存器,则在指令中其地址是隐含的,无需明确指出。如果用通用寄存器作为基址寄存器或变址寄存器,就需在指令中明确给出其寄存器地址。

作为对系统软件和计算机语言结构的重要支持,偏移寻址正在被广泛地使用。

7. 堆栈寻址

有关堆栈的概念,在 2.1.3 节中已经讨论过了。机器中通常使用专用的寄存器存放栈顶,称为堆栈指针 SP。指令通过 SP 访问堆栈,因此访问堆栈实质上就是一种寄存器间接寻址。

以堆栈寻址方式工作的指令,一般都不是明显地给出操作数的地址,而是隐含着操作数的地址,这个地址就是栈顶。将这种隐含操作数地址的寻址方式称为隐含寻址(Implied Addressing)。除了堆栈寻址外,在其他一些寻址过程中也存在着隐含寻址。例如累加寄存器就是采用隐含寻址来使用的。

堆栈寻址在早期的计算机中使用得并不多,但现在几乎所有的计算机都支持堆栈寻址,因为堆栈寻址可以很好地支持子程序、递归、嵌套等程序结构。

8. 基本寻址方式的小结

上面讨论了一些基本的寻址方式,通过表 2.6,对各种基本寻址方式进行以下小结。

表 2.6 基本寻址方式的比较

寻址方式	规 则	主要优点	主要缺点
立即数寻址	$\text{操作数} = A$	无需访问存储器	操作数范围受限
直接寻址	$EA = A$	简单	寻址空间受限
间接寻址	$EA = (A)$	寻址空间大	多次访问主存
寄存器寻址	$EA = R$	无需访问存储器	寻址空间受限
寄存器间接寻址	$EA = (R)$	寻址空间大	多访问主存一次
偏移寻址	$EA = (R) + A$	灵活	复杂
堆栈寻址	$EA = (SP)$ 地址隐含	缩短指令字长	应用范围有限

上述的基本寻址方式是目前最常用的一些寻址方式,但不是全部。很多计算机可以对已有的基本寻址方式进行组合,构造更加复杂的复合寻址方式,下面简单地讨论复合寻址方式。

2.3.2 复合寻址方式

两种或者两种以上的寻址方式相结合而形成的寻址方式称为复合寻址方式,如把间接寻址方式同相对寻址方式或变址方式相结合而形成的寻址方式。它分为先间接方式与后间接方式两种。

1. 变址间接方式

这种寻址方式是先把变址寄存器 X 的内容和形式地址 A 相加得 $(X) + A$,然后间接寻址,求得操作数的有效地址,即:

$$EA = ((X) + A)$$

2. 间接变址方式

这种寻址方式是先将形式地址取间接变换 $(A) = N$,然后把 N 和变址寄存器的内容(X)相加,即得操作数的有效地址。即:

$$EA = (X) + N = (X) + (A)$$

还有相对间接方式、间接相对方式等寻址方式,与上述间接变址方式相类似,读者亦可自己得出结论。

作为找到操作数的方法,寻址方式的种类有很多。由于寻址方式的实现需要硬件的有效支持,故一台计算机通常不会支持所有这些寻址方式,需要在寻址的灵活性和硬件复杂性之间进行权衡。到底选择支持哪些寻址方式将是“计算机体系结构”课程中学习的内容。

2.4 基本指令和指令类型

指令系统一般由几十到几百条不同类型的指令构成。据第一章知,存储程序计算机都应当具有数据传送、数据存储、数据处理、操作控制和操作判断这 5 大功能。因此基本的指令系统中应包含以下几大类型的指令:数据传送、算术逻辑运算、数据转换、I/O、程序控制和系统控制类等。

2.4.1 数据传送指令

数据传送指令可实现寄存器与寄存器、寄存器与存储单元以及存储单元之间的数据传送。在指令系统中,这是一类最基本最常用的指令。

数据传送指令可以一次传送一个数据,也可以一次传送一批数据。指令中应该给出以下 3 个参数:源操作数地址,目的操作数地址,传送数据的长度。

其中操作数地址可以是存储器地址、寄存器地址或栈顶。有的指令系统,如 IBM S/370,在操作码中指出操作数的位置(存储器或寄存器),如表 2.7 所示。有的指令系统,如 VAX,则是在操作数地址中指出操作数的位置。

表 2.7 IBM S/370 数据传送指令

助记符	操作名称	传送的字节数	传送说明
L	取数	4	存储器传送到寄存器
LH	取半字	2	存储器传送到寄存器
LR	取数	4	寄存器传送到寄存器
LER	取数(短)	4	浮点寄存器传送到浮点寄存器
LE	取数(短)	4	存储器传送到浮点寄存器
LDR	取数(长)	8	浮点寄存器传送到浮点寄存器
LD	取数(长)	8	存储器传送到浮点寄存器
ST	存数	4	寄存器传送到存储器
STH	存半字	2	寄存器传送到存储器
STC	存字节	1	寄存器传送到存储器
STE	存数(短)	4	浮点寄存器传送到存储器
STD	存数(长)	8	浮点寄存器传送到存储器

一般来说,对于不同长度的单个数据,如 1 字节、2 字节、4 字节或 8 字节,指令系统中都分别设有不同的指令,即在指令码中指出传送的数据长度,IBM S/370 就是一个例子。但是当指令需要传送一批数据时,通常要在指令中使用一个操作数来指出传送数据的长度。

由于数据类型与硬件结构的不同,各种机器的传送指令有很大不同。除了单纯的数据传送指令,有些机器还有清零传送指令,传送算术负值,传送逻辑反码等。另外,有些机器还设置了数据交换指令,完成源操作数与目的操作数的互换,它可以看成是双向数据传送。

堆栈操作指令 PUSH(压栈)和 POP(弹栈)也可以看成是数据传送指令。对于 PUSH 指令来说,其目的操作数地址是隐含的,即栈顶。指令中只需给出源操作数地址。对于 POP 来说则相反,指令中只需给出目的操作数地址。

从 CPU 操作的角度来看,数据传送类指令的处理相对比较简单。对于源和目的操作数都是寄存器的情况,CPU 只需要产生传送信号就可以了,这个操作在 CPU 内部就能完成。如果操作数在存储器中,则需要经过寻址机构访问存储器获得。

2.4.2 算术逻辑运算指令

计算机运算处理可分为算术运算和逻辑运算两大类,因此,现代计算机一般都设有算术运算指令和逻辑运算指令。

1. 算术运算指令

虽然这一类指令对不同的机器差别比较大,但大多数机器都提供相同的基本操作,至少提供二进制定点数的加、减、乘、除等最基本的运算指令。其他的指令还包括一些单操作数运算指令,如:求绝对值、求反、加 1 和减 1 等。

算术运算指令的操作过程一般都包括一些数据传送的过程,首先是将源操作数传送到算术逻辑部件 ALU 的输入端,然后进行运算,最后将 ALU 的运算结果传送到目的操作数地址单元。

2. 逻辑运算指令

逻辑运算是—类按位进行的运算,它可以针对任何一种可寻址单元中的位进行,故在有的机器中称之为位操作指令。逻辑运算的基础是布尔运算。

两个变量的逻辑运算可以有多种,但很少在指令系统中全部设置。在计算机中设置的逻辑运算指令一般包括与、或、非、异或以及相等操作等。

对于非运算的位操作,如位测试(测试某位的值)、位清除(将某位清零)、位设置(将某位置 1)、位求反(将某位求反)等,有些机器设置有专门的位操作指令,有些机器则通过逻辑运算指令去实现。

还有一类按位操作的逻辑运算,就是移位操作。移位的种类主要有算术移位、逻辑移位和循环移位。

有关算术逻辑运算的细节可以参见本书第三章的相关内容。

2.4.3 数据转换指令

数据转换指令用于改变数据的格式。这类指令包括数制转换指令和数据类型转换指令。数制转换指令主要指将十进制数转换为二进制数或相反。例如 Pentium 指令系统中的 FBLD (load packed decimal)^①指令转换内存的压缩 BCD 数成双扩展精度浮点数,并将数据压入栈顶。还有一类数据类型转换指令可以完成定点数和浮点数之间的转换。例如 Pentium 指令系统中的 FILD (load integer) 指令转换内存的整数操作数成双扩展精度浮点数,并将数据压入栈顶。复杂的数据转换可以通过某一个对照表进行,在 Pentium 指令系统中的 XLAT 指令,它根据用户定义的转换表,将一个数据转换成另一个数据,这种数据转换指令对于采用非线性索引的数据转换非常有效。

2.4.4 程序控制指令

程序控制指令主要用来控制指令执行的顺序,即控制程序的流程,是指令系统中一组非常重要的指令。

计算机在执行程序时,通常情况下按指令计数器的现行地址顺序取指令。每执行一条指令后,指令计数器 PC 自动加 1,指向顺序的下一条指令。但是,在许多情况下,需要改变程序原来的执行顺序。它是由程序控制指令来实现的。程序中需要改变执行顺序的原因很多,主要有:

^① FBLD 指令和 FILD 指令都不仅仅是数据转换指令,还有 load 功能。

- 在计算机的实际应用中,有时需要反复执行一段程序,这要靠循环来实现;
- 几乎所有的程序都涉及决策问题,即在不同条件的情况下做不同的事情;
- 编写和调试大程序是一个非常困难复杂的任务,往往需要将之分解为若干个较小的模块来分别编写。执行时要靠程序控制指令来控制这些模块的执行顺序。

程序控制指令主要包括转移、跳越、子程序调用与返回、以及循环控制等几种。

1. 转移指令

转移(Transfer),又称为分支(Branch)、跳转(Jump),是指CPU执行的指令顺序发生变动,即让CPU到新的地址去取指令执行。转移指令主要有无条件转移和条件转移两种。

无条件转移不受任何条件约束,直接将控制转移到指令中所指定的目标,从那里开始执行。CPU在执行无条件转移指令时,只需将转移目标地址送入程序计数器PC就可以完成。在转移指令中给出转移目标地址的最常用的方法是采用相对寻址,即将偏移量与程序计数器PC相加,就形成实际的转移目标地址。

条件转移则是指根据所指定的条件是否满足来决定是否转移。当条件满足时,才发生转移,否则仍顺序执行下一条指令。转移的条件,一般是上一条指令或更前面指令运算结果的某些特征。经常使用的有下列几种:结果=0,结果 $\neq 0$,结果>0,结果<0,有进位,结果溢出等。

实现条件转移实际上涉及两部分工作,即条件的产生和根据条件进行转移。在顺序上,条件产生在前,转移在后。实现条件转移的方法主要有采用条件码、使用条件寄存器以及采用比较与转移指令三种。

(1) 采用条件码(Condition Code, CC)

在机器中设置条件码寄存器,用以记录最近操作的执行结果标志。主要包含:进位标志,结果为0标志,结果为负标志,结果溢出标志等。许多指令对这些标志有影响,即在执行这些指令后,硬件将自动根据运算结果设置上述标志位。通常将由这些标志组成的寄存器称为条件码寄存器。

具有条件码寄存器,在条件转移时就可以不必对运算结果进行测试,若条件码寄存器的内容满足指令中指定的转移条件,则发生转移。在指令中指定转移条件,既可以在指令操作码中指出(即对于不同的转移条件有不同的指令操作码),也可以专门设置一个转移条件字段来确定。这种方法的优点是条件码是由硬件自动产生,在许多情况下不必专门用指令来判断结果的状态。其缺点是对指令的顺序要求严格,经常是要求产生条件码的指令紧挨着或靠近转移指令,以免条件码被其他指令破坏。这对于指令的并行处理有较大影响。

(2) 采用条件寄存器

采用这种方法时,先用比较指令对运算结果进行测试,并将测试结果(真或假)存放于一个通用的寄存器(称为条件寄存器)中,然后条件转移指令根据该寄存器的内容进行转移。这种方法实际上是将产生转移条件的工作交给程序员,由程序员对条件码的产生、存放和测

试进行安排。

这种方法的优点是简单,且对指令的顺序要求较低。只要保证不对条件寄存器进行写操作,产生条件的指令与条件转移指令可以相距任意远,便于指令的并行处理。其缺点是需要使用一个通用寄存器,而且要专门用指令来产生条件码,并且实现转移需要两条指令。

(3) 采用比较与转移指令

比较与转移指令就是在一条指令中完成产生条件和转移两部分工作。

这种方法的优点是只用一条指令即可。但这一条指令与其他指令相比,功能比较复杂,不利于实现,而且其产生条件的操作也往往局限于部分比较功能。

2. 跳越指令

另一种常用的程序控制指令是跳越指令。通常它只跳越过一条指令。该指令中隐含了一个地址,即下一条指令的地址。

由于跳越指令功能简单且不需要目标地址字段,所以可以让它顺带完成其他一些功能,“加 1 - 判 0 - 跳越 (ISZ)”指令就是一个典型的例子。这实际上是一条条件跳越指令。它可用来实现迭代循环,如下所示:

```
301          ;循环开始  
...  
309      ISZ      R1      ;加 1 - 判 0 - 跳越,结束循环  
310      BR       301      ;继续循环  
311
```

其中 R_1 的初值为循环次数的负值,在循环的末尾,由 ISZ 指令把 R_1 加 1,并判是否为 0。若不为 0,则执行 BR 指令,转移到循环的开始。否则,跳过 BR 指令,退出循环。

3. 子程序调用和返回指令

子程序的概念可能是程序设计中最重要的创新之一,它也是模块化程序设计的基础。子程序可以随时由主程序调用,这样编程人员就不必多次重复编写,既简化了程序设计,提高了编程效率,又节省了存储空间。另外,现代计算机系统往往也提供了大量的通用子程序,如求解初等函数、线性方程组等计算子程序以及其他标准子程序。将这些子程序放在一起,就构成子程序库,极大地方便了用户编写应用程序。

子程序调用机制涉及两条基本指令:子程序调用指令和返回指令。子程序调用指令使 CPU 从调用指令处转移到子程序去执行。返回指令则是使 CPU 从子程序中返回到调用指令的下一条指令继续执行。它们都是某种形式的转移指令。需要说明的是,如果子程序中又出现了子程序调用,这种现象称之为嵌套;如果子程序调用的是自己,这种情况称为递归调用。

在实现子程序调用机制时,一个重要问题是正确地实现从子程序返回主程序,由于调用程序可以在任何一个位置调用子程序,所以如何确定返回地址就成了问题的关键。

解决这个问题的一般方法,是在调用子程序时,将返回地址存放在某个约定的地方;而返回时,就根据这个约定地方中存放的地址返回。常用的存放返回地址的地方有3个:寄存器、子程序的起始位置或者栈顶。

考虑子程序调用指令 $\text{CALL } X$, X 为子程序的首地址。若采用寄存器存放返回地址,则该指令的操作为:

$$\begin{aligned} \text{RN} &\leftarrow \text{PC} + \Delta \\ \text{PC} &\leftarrow X \end{aligned}$$

其中 RN 为专用于保存返回地址的寄存器, Δ 为子程序调用指令的长度。若将返回地址存于子程序的起始位置,则调用指令的操作为:

$$\begin{aligned} \text{X} &\leftarrow \text{PC} + \Delta \\ \text{PC} &\leftarrow \text{X} + 1 \end{aligned}$$

X 为子程序的首地址,其中存放了返回地址,而 $\text{X} + 1$ 处为子程序的第一条指令。

以上两种方法都在实际机器中得到了应用。但第一种方法不支持子程序嵌套和递归调用,而第二种方法不支持子程序递归调用。两种方法返回调用程序均可使用无条件转移指令代替。采用寄存器间接寻址的无条件转移指令,如 $\text{BR } @ \text{RN}$;采用存储器间接寻址的无条件转移指令,如 $\text{BR } @ \text{X}$ 。指令中“@”表示间接寻址方式。

为解决子程序嵌套和递归调用问题,计算机通常采用堆栈来保存返回地址。调用子程序时,首先将调用指令的下一条指令的地址压入堆栈保存(保存断点),然后转入所调用的子程序执行。子程序执行完毕,由返回指令把返回地址从堆栈中弹出,返回调用程序。由于堆栈具有后进先出的性质,因而用堆栈保存返回地址可实现子程序嵌套和递归调用。

例如,假设有3个程序 M 、 A 、 B ,它们的调用关系是 M 调用 A , A 又调用 B ,如图 2.23 所示。

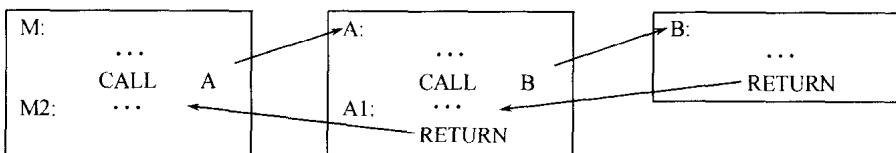


图 2.23 M 、 A 和 B 的调用关系

当程序 M 执行到调用子程序指令 $\text{CALL } A$ 时,首先将该指令的下条指令的地址(返回地址) $M2$ 压入堆栈保存,然后执行 A 。当子程序 A 执行到指令 $\text{CALL } B$ 时,又将其下条指令的地址 $A1$ 压入堆栈,并转入执行子程序 B 。子程序 B 执行完毕,由返回指令从堆栈中弹出 $A1$ 作为返回地址,返回到子程序 A 继续执行。子程序 A 执行完毕,又从堆栈中弹出 $M2$,返回到程序 M 继续执行。图 2.24 就说明了这个过程中堆栈的使用情况。

除了返回地址问题外,子程序调用时,还要考虑参数传递问题,即在调用子程序时,经常需要将参数从调用程序传给被调用子程序。虽然可以用寄存器或存储单元来传递,但与保存返回地址的情况一样,它们也不支持子程序的可再入性。更好更灵活的方法也是使用堆

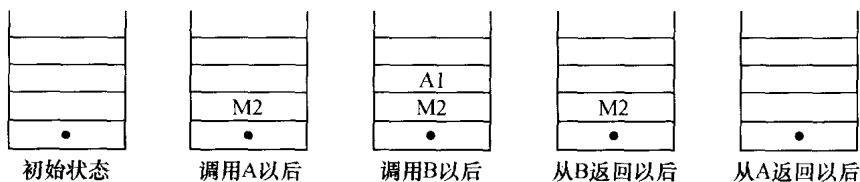


图 2.24 使用堆栈实现 M、A 和 B 中的子程序嵌套

栈。因此现代计算机不仅用堆栈来保存返回地址,而且用堆栈来传递参数。

子程序调用指令和转移指令都可以改变程序的执行顺序,这一点它们是相似的。但两者又有所不同。子程序调用指令在转去执行一段子程序之前,保存了返回地址,它可以实现程序与程序之间的转移,而且能从子程序返回到原来的调用程序继续执行。子程序还可以自己调用自己,实现递归调用。而转移指令则转移到指令给出的转移地址处去执行指令,不存在返回要求,一般用于实现同一程序内的转移。

4. 循环控制指令

有了条件转移指令就可以实现循环程序设计。为了对循环提供更多的硬件支持,很多机器在指令系统中还专门的设置了循环控制指令。这种指令的功能包括对循环控制变量的修改和结束循环条件的判断,集运算(通常为加、减等)、测试和条件转移于一体,是一种具有复合功能的指令。例如:Pentium 指令系统中的 LOOPZ 指令,其功能为循环,同时循环变量(存于通用寄存器 ECX 中)自增或自减,直到循环变量值为 0,循环结束。

2.4.5 输入/输出指令

输入/输出指令用来实现主机与外部设备之间的信息交换。许多机器一般只提供基本的几条 I/O 指令。典型的指令有输入、输出、启动 I/O、测试 I/O 等。输入/输出指令的设置方法有以下两种。

1. 设置专用的 I/O 指令

用专门的 I/O 指令进行输入/输出操作。这种 I/O 指令通常包含两个操作数。一个指出存放输入或输出数据的单元的地址,另一个指出外部设备的地址。输入指令从外部设备读入数据,存放到指定的单元中,输出指令则将数据送到指定的外部设备中。

2. 用通用的数据传送指令实现 I/O 操作

用通用的数据传送指令实现 I/O 操作,又称之为存储器映像的 I/O 操作(Memory-mapped I/O)。在外部设备的寄存器与主存单元统一编址的机器中,因为外部设备接口中的寄存器和存储器单元一样同等对待,所有访问存储器单元的指令均可以访问外部设备的寄存器。这样就可以用传送类指令去访问外部设备接口中的寄存器,来实现主机与外部设备之间的信息交换,而不必专门设置 I/O 指令。

关于输入/输出指令的详细内容可参见本书第六章。

2.4.6 系统控制指令

系统控制指令是指那些通常只能由操作系统执行,而不直接提供给用户使用的指令,故又称为特权指令。CPU 只有处于特权状态时,才能执行这些指令。系统控制指令主要用于实现对控制寄存器进行操作,检测或修改访问权限,改变系统的工作方式,访问进程控制块等。在多用户、多任务环境下使用较多。

当一般用户需要计算机系统的特权服务时,必须通过系统控制指令向操作系统发出请求,由操作系统处理,并将处理的结果返回给用户。

在某些多用户计算机系统中,输入/输出指令也被作为特权指令,不允许用户直接使用。需要使用输入/输出指令时,必须通过系统控制指令,由操作系统完成,这样操作系统就可以统一管理所有的外部设备。

2.5 指令系统的分类

不同类型的计算机都有各具特色的指令系统。指令系统可以按多个方面进行分类,例如:按 CPU 中操作数如何存储,按指令中的操作数个数,按操作类型等。其中按 CPU 中操作数如何存储来分类是最基本的,对计算机系统设计的影响也最大,本节仅讨论这种分类方法。

按照 CPU 中操作数的存储位置,指令系统可分为堆栈型、累加器型和通用寄存器型三类,其相应的机器分别称为堆栈型机器、累加器型机器和通用寄存器型机器。在堆栈型指令中,操作数是隐含的,即在栈顶;在累加器型指令中有一个操作数是隐含的,即累加器;在通用寄存器型指令中,操作数全都是显式的,或者为寄存器地址,或者为主存地址。显式操作数有的可以直接从主存中存取,有的需先装入暂时存储器,依不同的结构和具体的指令而定。三种不同的指令系统类型列于表 2.8。

表 2.8 三种类型的指令系统

指令系统类型	暂时存储器	机器实例	运算指令中显式操作数	结果存放位置
堆栈型	堆栈	B5500, HP3000/70	0 个	堆栈
累加器型	累加器	PDP-8, Motorola 6809	1 个	累加器
通用寄存器型	寄存器组	X86 系列, 所有 RISC	2 或 3 个	寄存器或存储器

为进一步了解上述三种结构,给出在这三种指令系统的情况下,求解 $C = A + B$ 的典型指令序列,如表 2.9 所示。

表 2.9 三种不同类型的指令系统求解 $C = A + B$ 的指令序列

堆 栈 型		累 加 型		通 用 寄 存 器 型	
PUSH	A	LOAD	A	LOAD	R1, A
PUSH	B	ADD	B	ADD	R1, B
ADD		STORE	C	STORE	C, R1
POP	C				

注：假设 A、B、C 均在存储器中，且 A 和 B 不能被破坏。

堆栈型机器的主要优点是表达式计算简单，指令短。其缺点有两点，一是堆栈不能被随机访问，这使得难以生成有效的代码；二是难以快速运算，因为堆栈会成为瓶颈。

累加器型机器的主要优点是使机器的内部状态简单，指令短，实现容易；其缺点是访问累加器的操作最频繁，因为运算指令中只有累加器作为目的地址。

通用寄存器型机器的主要优点在于它是编译器代码生成的最通用的模型，编译器可以有效地利用寄存器来计算表达式的值；利用寄存器来存放变量，可以减少访存次数，大幅度地提高程序执行速度。其缺点是指令中需显式指出所有的操作数，致使指令字较长。

早期的机器大多数都用采用累加器风格的结构，采用堆栈风格的计算机一直数量不大，目前最具有生命力的机器几乎都采用通用寄存器结构。通用寄存器型机器主宰着当今计算机的指令系统结构，并且似乎在将来也是如此。通用寄存器型机器出现的原因有两点，第一，与其他形式的 CPU 内部存储器一样，由于采用与 CPU 相同的工艺一起生产，寄存器可以与 ALU 同步工作，其速度远比主存快；第二，寄存器更易于被编译器使用，而且能比其他任何形式的内部存储器更有效地使用。随着集成电路技术的发展，单片芯片中的电路数量大幅度增加，计算机设计师都使用了大量的电路制造寄存器，目前高性能微处理器中的寄存器都达到了几百个的数量级，而某些专用寄存器的数量更是多达几千个。大量使用寄存器是现代计算机提高性能最重要的手段之一。

根据运算指令中存储器型操作数的个数，通用寄存器型机器可以进一步分为三种类型：

(1) 寄存器 - 寄存器 (R - R) 型(也称为 LOAD/STORE 型)，寄存器 - 寄存器型机器的运算指令中不含存储器型操作数。机器在执行这类指令的过程中，只对寄存器中的操作数进行操作，从寄存器中取操作数，结果也放到寄存器中，不需要访存。因此其执行速度很快。

(2) 寄存器 - 存储器 (R - M) 型，寄存器 - 存储器型机器的运算指令操作数既有寄存器型的，又有存储器型的。执行这类指令时，既要访问寄存器，又要访问存储器。

(3) 存储器 - 存储器 (M - M) 型，存储器 - 存储器型机器中，部分运算指令的操作数都存放在存储器中。执行这种指令时，需从存储器中取操作数，操作结果也放至存储器中，因此，需要多次访问存储器，执行速度慢。

2.6 指令系统设计与实例

指令系统的设计首先是根据需求,确定指令系统基本目标和结构,然后进行具体设计。也就是先确定设计的指导思想,再进行功能实现。具体设计主要包括指令基本功能、操作数类型、寻址方式和指令格式的设计等内容,这些内容前面已经讨论过了。

本节主要讨论指令系统设计的基本要求,然后简单介绍 CISC 和 RISC 的设计思想,最后介绍 Pentium 和 PowerPC 这两种典型的指令系统的基本特征。

2.6.1 指令系统设计的基本要求

指令系统的性能如何,决定了计算机的基本功能,因而指令系统的设计是计算机系统设计中的一个核心问题。它不仅与计算机的硬件结构紧密相关,而且直接关系到用户的使用需求。不同类型计算机都有各自特色的指令系统。由于计算机性能、机器结构和特点、使用环境等要求不同,指令系统间的差异也是很大的。同时,随着计算机迅速发展,对计算机性能要求愈来愈高,硬件价格迅速下降,机器指令系统也日趋功能强大、高效率、复杂化。因此,企图给计算机指令系统确定一个统一的衡量标准是很困难的。只能讨论在一般情况下,一个完善的指令系统应满足的一些基本要求。

1. 完备性

指令系统的完备性是指在一个有限可用的存储空间,对于任何可解的问题,编制计算程序时,指令系统所提供的指令足够使用。这是一个原则性要求,很难确定一个完备性的标准。完备性要求指令系统丰富、功能齐全、使用方便。

一台计算机中最基本、必不可少的指令是不多的。许多指令可用最基本的指令编程来实现。例如,乘除运算指令、浮点运算指令可直接用硬件来实现,也可用基本指令编写的程序(软件)来实现。采用硬件指令的目的是提高程序执行速度,便于用户编写程序。

一般来说,为了程序高效运行和便于硬件实现,实际计算机指令系统中实现的指令远远超过了基本完备性的要求,成为一个功能完备的指令系统。一个功能完备的指令系统应包括的指令在 2.4 节中已经作了比较全面的论述。

2. 有效性

有效性是指利用该指令系统所编写的程序能够高效率地运行。高效率主要表现在程序占据存储空间小、执行速度快。有效性是针对整个指令系统而言,是一个很复杂的问题,也很难确定一个统一标准。它反映了指令的功能要求,也反映了指令系统的完备性要求。一个更完备的指令系统就会有更好的有效性。如指令系统中有多种移位指令和字符处理指令,对于数据处理就会有较高的有效性;而设计一些功能较强的指令也会增加指令系统的有效性。一般来说,一个功能更强、更完善的指令系统,必定有更好的有效性。

强调有效性,一直是计算机系统实际的重要原则之一,也是传统的 CISC 的出发点。

3. 规整性

规整性包括指令系统的对称性、匀齐性、指令格式和数据格式的一致性。

对称性:在指令系统中所有的寄存器和存储器单元都可同等对待,所有的指令都可使用各种寻址方式。这种操作的对称性对于提高软件效率和使用方便是很有利的。如传送指令既有 $A \leftarrow B$,也有 $B \leftarrow A$;加法指令既有 $A \leftarrow (A + B)$,也有 $B \leftarrow (A + B)$,等等。但是,在许多机器上还不能很好地实现这一对称性。VAX - 11 机的指令系统操作有较好的对称性。

匀齐性:一种操作性质的指令可以支持各种数据类型,如算术运算指令可支持字节、字、双字整数的运算,十进制数运算和单、双精度浮点数运算等。操作的匀齐性可使汇编程序设计与编译程序无须依赖数据类型而选用指令,缩短程序空间和加快程序执行速度。指令操作匀齐性在目前一些机器中还不能很好地实现。究其原因,主要还是出于机器性能价格比的考虑,另外,过分地追求匀齐性,还会出现一些意义不大的指令,造成指令系统过分庞杂。

指令格式和数据格式的一致性:指令长度和数据长度有一定的关系,以方便处理和存取。例如指令长度和数据长度通常是字节长度的整数倍。如机器基本字长为 32 位,长指令选 32 位,短指令选 16 位,在这个字长条件下,来选择指令的格式,指令的存放就比较规整。

4. 兼容性

系列计算机各机种之间具有相同的基本结构和共同的基本指令系统,所以各机种上软件可以通用,因而系列计算机的指令系统是兼容的。但由于不同机种推出的时间不同,在结构和性能上有差异,做到所有软件任意互换都完全兼容是不可能的。但是对于系列计算机来说,低档计算机上的软件可以在高档机器运行(向上兼容),这是系列计算机的本质特征之一。

2.6.2 CISC 和 RISC

指令系统的设计与计算机制造技术的发展是密切相关的。20世纪 50~60 年代,计算机大多数采用分立元件的晶体管或电子管组成。由于受器件的限制,计算机所支持的指令系统只有定点加减、逻辑运算、数据传送、转移等十几至几十条指令。20世纪 60 年代后期,随着集成电路的出现,指令系统越来越丰富,除以上基本指令外,还设置了乘除运算、浮点运算等指令,指令数目多达一、二百条。

20世纪在 70 年代,高级语言已成为大、中、小型机的主要程序设计语言。为对高级语言提供更多的支持,计算机设计者们利用当时已经成熟的微程序技术和飞速发展的 VLSI 技术,增设各种各样的复杂的、面向高级语言的指令,使指令系统越来越庞大,达到了包含多达数百条指令。这是几十年来人们在设计计算机时,保证和提高指令系统有效性方面传统的想法和作法。按这种传统方法设计的计算机系统称为复杂指令集计算机 CISC。研究结果表明,CISC 由于其指令系统的复杂性,存在以下主要问题:

- 复杂指令并不能有效地得到利用;

- 复杂指令系统会降低整个机器的执行速度；
- 复杂指令带来了计算机组成及实现上的复杂性，不便于用 VLSI 实现；
- 指令系统设计时间长，且由于系统复杂，可能包含更多的设计错误。

CISC 的设计思想是把硬件资源主要用于提高指令系统的功能和规模，使指令系统设计得尽可能接近高级语言，而没有最大限度地利用这些资源来提高性能。为了解决这个问题，人们又提出了便于 VLSI 技术实现的精简指令集计算机 RISC。

RISC 是由美国加州大学伯克利分校 David Paterson 教授在 20 世纪 70 年代所取的名字，一直使用至今。RISC 最主要的特点反映在其指令系统上，以尽可能地提高处理机的有效速度为依据来确定指令系统，这种指令系统简单、紧凑，硬件实现容易，而且能有效地支持优化编译。另外，由于 RISC 机器的研制周期短，耗资少，性能价格比高，并且没有软件兼容性的问题，所以迅速受到计算机界的重视。具体来说，RISC 指令系统具有以下主要特点：

- 指令功能简单，指令条数少；
- 采用定长、简单的指令格式，典型的为 4 个字节；
- 寻址方式简单，数量少，不采用存储器间接寻址技术；
- 只有 Load/Store 指令能够访问主存，在一条指令中，操作数访存寻址不会超过一次；
- 运算类指令多采用三地址寄存器寻址方式，不直接访存；
- 设置大量的寄存器，指令操作大多都在寄存器之间进行；
- 对于有浮点处理部件的计算机，使用大量的浮点寄存器；
- 一个周期出现一个指令执行结果，但是其性能的发挥极度依赖于编译器的优化。

RISC 机型拥有一个有限的指令集合，并且能够极快地执行每一条指令。但是由于缺乏复杂指令，使用精简指令所编写出来的代码会更长，所以执行任务所需的内存也就更大。而 CISC 机型有着丰富的指令集合，使计算机能够利用一两条指令就能执行非常复杂的操作，最终所产生的代码长度较小。CISC 和 RISC 各有优点，因此，融合 CISC 和 RISC 系统设计优点的计算机系统已经成为新型计算机设计的指导思想。

2.6.3 指令系统实例

本节主要介绍 Pentium 和 PowerPC 这两种典型机型的指令系统。指令系统的设计要素主要包括以下四方面：数据表示、指令格式、寻址方式和操作类型。两种机型的数据表示基本相同，包括字符、多种类型的整数、浮点数、十进制数等。因此这里着重讨论它们的指令格式、寻址方式和操作类型。

1. Pentium 指令系统

(1) 指令格式

Pentium 配备了较复杂的各种指令格式，如图 2.25 所示。指令由以下几个部分组成：

- 前缀字段：0 ~ 4 个字节可选；

- 操作码字段:1 或 2 个字节;
- MOD/RM 字段:0 或 1 个字节;
- SIB 字段:0 或 1 个字节;
- 偏移量字段:0、1、2、4 个字节可选;
- 立即数字段:0、1、2、4 个字节可选。

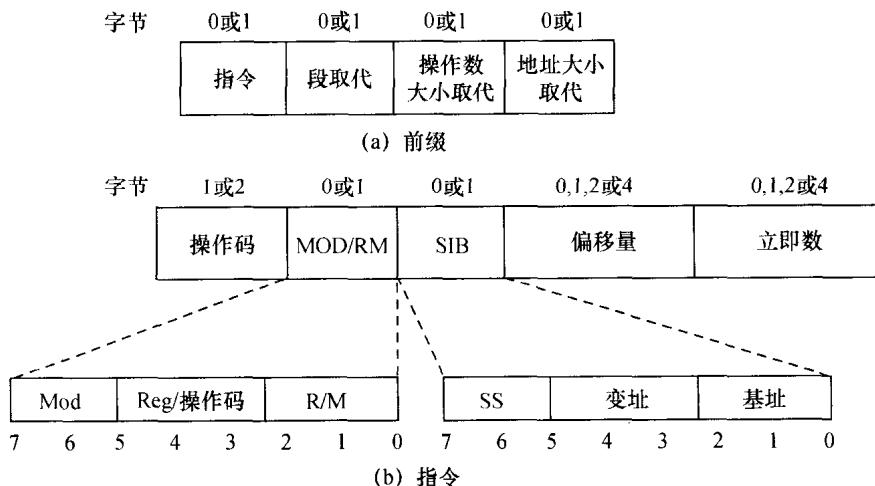


图 2.25 Pentium 指令格式

除操作码字段是必要的,其他都是可选的。下面具体说明每个字段含义。

① 前缀字段包括:

- 指令前缀(Instruction Prefix):由 LOCK(锁定)前缀或重复前缀所组成。LOCK 前缀用于在多处理机环境中,保证对共享存储器的独占性访问。重复前缀指定串的重复操作,这就使 Pentium 处理串要比正规的软件循环快得多。有 5 种重复前缀:REP, REPE, REPZ, REPNE 和 REPNZ。当无条件的 REP 前缀出现时,指令中指定的操作对串的连续元素重复执行,重复的次数由 CX 寄存器指定。条件 REP 前缀出现时,引起指令重复执行直到 CX 变成 0 或指定的条件被满足。

- 段取代(Segment Override):指定这条指令应使用哪个段寄存器。
- 地址长度(Address Size):处理器能使用 16 位或 32 位地址来寻址存储器。该前缀确定了指令格式中偏移量和在有效地址计算中生成的位移量的位数。指令缺省的地址长度是 16 位或 32 位,地址长度前缀用于 32 位和 16 位地址生成之间的切换。
- 操作数长度(Operand Size):指令缺省的操作数长度是 16 位或 32 位,操作数长度前缀用于 32 位和 16 位操作数之间的切换。

② 操作码:操作码亦包括一些位指定:数据是字节还是全尺寸(16 位或 32 位,取决于上下文),数据操作方向(写入存储器或从存储器读出),以及一个立即数字段是否是符号扩展的。

③ MOD/RM:这个字段和 SIB 字段联合提供寻址信息,合称为地址指定器。MOD/RM 字节指明寄存器操作数、存储器操作数,若是存储器操作数,则说明采用的寻址方式。MOD / RM 字节由三个字段组成:

- MOD 字段,2 位,与 RM 字段组合可构成 32 种寻址方式;
- Reg/操作码字段,3 位,指定一个寄存器号或者用作扩展操作码;
- RM 字段,3 位,指定一个寄存器作为一个操作数的位置,或者与 MOD 字段组合起来编码,构成寻址方式的一部分。

④ SIB:MOD / RM 字节的某些编码要求与 SIB 字节联合来完成寻址方式的指定。SIB 字节由三个字段组成:

- SS 字段,2 位,指定用于比例变址的比例因子;
- 变址字段,3 位,指定变址寄存器;
- 基址字段,3 位,指定基址寄存器。

⑤ 偏移量:当地址指定器指出使用一个偏移量时,该字段指定一个 8 位、16 位或 32 位有符号整数的偏移量。

⑥ 立即数:在指令中提供一个 8 位、16 位或 32 位的操作数值。

在 Pentium 指令格式中,只能有一个存储器操作数参与操作。Pentium 指令格式允许变址使用 1、2 或 4 字节的位移。虽然使用较长的变址位移会导致指令更长,但这个特点能提供所需的灵活性。例如,在寻址大的数组时它就很有用。

Pentium 的指令格式是很复杂的。其部分原因是与 80X86 兼容的需要,部分原因是设计者打算为编译器编制者提供尽可能多的支援以产生更有效的代码。

(2) 寻址方式

由于 Pentium 存储器的分段结构,使得存储器操作数的物理地址由段基址和偏移地址两部分构成。段基址由 CPU 的工作方式、指令执行的上下文和寻址时所使用的寄存器决定。偏移地址相当于前面谈及的有效地址,仅与寻址方式有关。不同的寻址方式,偏移地址的形成方法不同。表 2.10 列出了 8 种寻址方式。

表 2.10 Pentium 寻址方式

方式	算法	方式	算法
立即数	操作数 = A	变址	$EA = (I) + A$
寄存器	$EA = R$	比例变址	$EA = (I) \times S + A$
直接	$EA = A$	基址比例变址	$EA = (B) + (I) \times S + A$
寄存器间接	$EA = (R)$	相对	$EA = (PC) + A$

注: EA = 有效地址; (X) = X 的内容; A = 指令中地址字段的内容;

R = 寄存器; B = 基址寄存器; I = 变址寄存器; S = 比例因子;

① 立即数方式,操作数包含在指令中。操作数可以是字节、字或双字。

② 寄存器寻址方式,操作数位于一寄存器中。对于数据传送、算术和逻辑指令这样的通常指令,操作数可以是一个 32 位通用寄存器(EAX,EBX,ECX,EDX,ESI,EDI,ESP,EBP)、一个 16 位通用寄存器(AX,BX,CX,DX,SI,DI,SP,BP),或是一个 8 位通用寄存器(AH,BH,CH,DH,AL,BL,CL,DL)。对于浮点操作,使用两个 32 位寄存器作为一对来构成一个 64 位操作数。Pentium 指令系统中亦提供访问段寄存器(CS,DS,ES,SS,FS,GS)的一些指令。

③ 直接寻址方式,指令中的 16 位或 32 位形式地址就是操作数距段起点的位移,可访问全局变量。在 Pentium 机中,偏移量可长达 32 位。

④ 寄存器间接寻址方式,指定一个 16 位或 32 位的寄存器,该寄存器中含有效地址。

⑤ 基址寻址方式,指令包括一个将被加到一个基址寄存器的形式地址,基址寄存器可是任意一个通用寄存器。这种方式可用于访问记录中的字段,基址寄存器指向记录的起点,而形式地址是到此字段的位移。

⑥ 变址寻址方式,指令包括一个将加到变址寄存器的形式地址。除 ESP 寄存器外的任何 32 位寄存器和 SI、DI 两个 16 位通用寄存器作为变址寄存器。有效地址为变址寄存器内容与形式地址之和。

⑦ 比例变址寻址方式,指令包括一个将加到变址寄存器的形式地址。除 ESP 外的任何 32 位通用寄存器都可以作为变址寄存器。计算有效地址时,变址寄存器的内容乘以 1、2、4,或 8 的比例因子,然后加上形式地址。对于访问一个数组,这种方式是很方便的。比例因子为 2 可用于一个 16 位整数数组;比例因子为 4 可用于 32 位整数或浮点数;比例因子为 8 可用于一个双精度浮点数的数组。

⑧ 基址比例变址寻址方式,将变址寄存器内容乘以比例因子、基址寄存器内容和形式地址三者取和。如一个数组存于堆栈中,这种寻址方式是很有用的,此时数组元素可以是 2、4、或 8 字节长。这种方式亦能对数组元素是 2、4 或 8 字节长的二维数组提供有效的访问。

⑨ 相对寻址方式,形式地址与程序计数器的值相加。此时,形式地址被看作是一个有符号的字节、字或双字,只用于程序控制指令。

每条指令的操作数寻址方式由指令中的地址指定器字段来描述。

(3) 操作类型

Pentium 提供了一系列复杂的操作类型,如表 2.11 所示。这些指令大多数是在其他机器指令系统中也能找到的常规指令,少部分是 Pentium 特有的。如 enter 指令(4 字节)是为了给编译程序提供直接的支持,也用来支持一些高级语言(如:PASCAL、COBOL、ADA)的嵌套。虽然使用 enter 指令节省了存储器的几个字节(相比 push、mov、sub 指令序列,6 字节),但实际执行它需要更长的时间。所以它使得处理器变得更复杂,而带来很少甚至没有好处。这就是 CISC 相对 RISC 的缺点所在。

表 2.11 Pentium 操作类型

指 令	描 述
	数据传送
MOV	在寄存器之间或寄存器与存储器之间传送操作数
PUSH	将操作数压入堆栈
PUSHA	将所有寄存器的内容压入堆栈
MOVSX	传送字节、字、双字；符号被扩展。字节传送到字或字传送到双字，以 2 的补码符号扩展方式进行
LEA	装入有效地址。将源操作数的偏移，而不是它的值，装到目标操作数
XLAT	表查找翻造。以用户编制的翻造表的一字节替代 AL 中的字节。当 XLAT 执行时，AL 中应有表的无符号数的索引值。XLAT 以被索引项的内容替代 AL 的内容
IN, OUT	由 I/O 空间输入，或输出到 I/O 空间
	算术
ADD	加操作数
SUB	减操作数
MUL	乘，双/单精度
IDIV	除
	逻辑
AND	AND 操作数
BTS	位测试和置位。对一位域操作数进行操作。指令将一位当前值拷贝到 CF 标志，并设置原来位为 1
BSF	位向前扫描。一位，扫描一个字或双字并将第 1 个匹配位的位号存于一个寄存器
SHL/SHR	左或右的逻辑移位
SAL/SAR	左或右的算术移位
ROL/ROR	左或右的循环移位
SETcc	根据状态标志定义的 16 个条件的某一个，设置一字节为 0 或 1
	控制传递
JMP	无条件转移
CALL	传递控制到另一位置。在传递之前，此 CALL 指令之后的指令地址压入堆栈
JE/JZ	若相等/为 0 则转移
LOOPE/LOOPZ	若相等/为 0 则循环。这是一个使用 ECX 中值的条件转移。在为转移条件测试 ECX 之前，指令先减量 ECX
INT/INTO	中断/若溢出则中断。传递控制到一个中断服务子程序
	串操作
MOVS	传送字节、字或双字的串。它对由 ESI 和 EDI 索引的串元素进行操作，每次串操作之后，这两个寄存器自动被增量或减量以指向下一个串元素
LODS	装入字节、字或双字的串
	高级语言支持
ENTER	生成一个能用来实现块结构高级语言规则的栈帧

(续表)

指 令	描 述
LEAVE	上述 ENTER 动作的逆动作
BOUND	检查阵列边界。验证操作数 1 是否在一个上下限范围之内,此上下限值存于被操作数 2 所参照的两个相邻存储器位置中。若值不在边界内,出现中断。此指令用来检查阵列索引
	标志控制
STC	置位进位标志
LAHF	将标志装入 AH 寄存器。拷贝 SF, ZF, AF, PF 和 CF 位到 AH 寄存器
	段寄存器
LDS	装入指针到 D 段寄存器
	系统控制
HLT	保持
LOCK	对共享存储器提出保持要求,这样在立即跟随 LOCK 之后的那条指令期间,Pentium 可排他性地使用存储器
ESC	处理器扩展换码。此换码指示后面的指令,将在支持高精度整数和浮点数计算的数据协处理器上执行
WAIT	等待,直到 BUSY 信号撤除。挂起 Pentium 程序的执行,直到处理器已测出 BUSY 引脚信号无效。不忙则表示数据协处理器已执行结束
	保护
SGDT	保存全局描述符表
LSI	装入段限。将段限装入一个用户指定的寄存器中
VERR/VERW	读/写证实段
	高速缓存管理
INVD	清洗内部 cache
WBINVD	在将脏行写回存储器之后,清洗内部 cache
INVLPG	使一个转换后援缓冲器(TLB)项无效

2. PowerPC 指令系统

PowerPC 充分考虑到计算机技术的发展,它的指令系统既可以用于 32 位微处理器系统,又可以用于 64 位系统,但是其中有些指令只能用于 64 位系统。这里主要讨论 32 位系统。

(1) 指令格式

PowerPC 的所有指令都是 32 位长的规整的指令格式,规整的指令格式为控制器的实现提供了方便。图 2.26 分类给出了 PowerPC 的所有指令的格式,图中的阴影部分为操作码,大部分操作码为 6 位,也可以扩展。

图中 A 为绝对或自相对标志(64 位系统有效);L 为指示或子程序标志;O 表示使用纪录溢出;R 表示使用纪录条件;XO 为操作码扩展;S 为移位量的一部分。

6位	5位	5位	16位		
分支	24位长立即数(地址)			A	L
条件分支	选择	条件位	分支偏移量		
条件分支	选择	条件位	间接分支指示		

(a) 分支指令

条件寄存器	目的数据位	源数据位	源数据位	运算(And、Or、Xor等)	/
-------	-------	------	------	-----------------	---

(b) 条件寄存器逻辑指令

Load/Store	目的寄存器	基址寄存器	偏移量		
Load/Store	目的寄存器	基址寄存器	变址寄存器	大小、符号、更新等	
Load/Store	目的寄存器	基址寄存器	偏移量		

(c) Load/Store 指令

算术	目的寄存器	源寄存器	源寄存器	O	加、减等	R
加、减等	目的寄存器	源寄存器	有符号立即数			
逻辑	源寄存器	目的寄存器	源寄存器	与、或、异或等		
与、或等	源寄存器	目的寄存器	无符号立即数			
循环	源寄存器	目的寄存器	移位量	屏蔽首	屏蔽尾	R
循环、移位	源寄存器	目的寄存器	源寄存器	移位类型或屏蔽		
循环	源寄存器	目的寄存器	移位量	屏蔽	XO	S R
循环	源寄存器	目的寄存器	源寄存器	屏蔽	XO	R
移位	源寄存器	目的寄存器	移位量	移位类型或屏蔽		S R

(d) 整数操作指令

单精度/双精度	目的寄存器	源寄存器	源寄存器	源寄存器	浮点加法等	R
---------	-------	------	------	------	-------	---

(e) 浮点算术指令

图 2.26 PowerPC 的指令格式

条件分支指令中的条件位,是用来定义本指令所测试的条件位究竟是条件寄存器中的哪一位。根据不同的条件位,可以产生以下一些条件分支指令:无条件分支、计数器 0、计数器 = 0、条件成立、条件不成立、计数器 0,条件成立、计数器 0,条件不成立、计数器 = 0,条件成立、计数器 = 0,条件不成立。绝大多数运算指令会对条件寄存器的设置产生影响。通过这些条件值,就可以使用上述分支指令。

Load/Store 指令,操作码后面有两个 5 位的寄存器域,它指明要使用和访问的寄存器。绝大多数浮点指令使用两个源寄存器,复合运算指令会用到 3 个源寄存器,在向量和矩阵运算中经常遇到,如求点积。

(2) 寻址方式

PowerPC 代表着一类计算机对寻址方式处理的技术观点:简单、直观、高效。

① 寻址方式分类

PowerPC 的寻址方式是按照指令的类型进行分类的,表 2.12 给出了它的寻址方式。

表 2.12 PowerPC 的寻址方式

指令	寻址方式	地址计算
Load/Store	基址寻址	$EA = (BR) + D$
	基址变址寻址	$EA = (BR) + (IR)$
分支指令	直接寻址	$EA = I$
	相对寻址	$EA = (PC) + I$
	间接寻址	$EA = (L/CR)$
定点运算指令	寄存器寻址	$EA = GPR$
	立即数寻址	操作数 = I
浮点运算指令	寄存器寻址	$EA = FPR$

符号说明:

EA:有效地址

(X):X 的内容

BR:基址寄存器

IR:变址寄存器

PC:程序计数器

L/CR:分支目标寄存器或者记录寄存器

GPR:通用寄存器

FPR:浮点寄存器

D:位移量

I:立即数

② Load/Store 指令的寻址方式

PowerPC 访问主存是通过专门的访存指令 Load/Store 指令来完成的。Load/Store 指令具有两种寻址方式:

- 基址寻址:指令的地址字段中包括一个 16 位的位移量,它与一个基址寄存器相加得到有效逻辑地址,该逻辑地址可以回送到基址寄存器,任何一个通用寄存器都可作为基址寄存器。图 2.27(a)给出了基址寻址访问主存的地址形成过程。

- 基址变址寻址:指令的有效逻辑地址为基址寄存器和变址寄存器这两个寄存器的内容之和,任一通用寄存器均可用来作为基址寄存器或变址寄存器。图 2.27(b)为基址变址寻址访问主存的地址形成过程。

③ 分支指令的寻址方式

分支指令提供三种寻址方式:直接寻址、相对寻址和间接寻址。

- 直接寻址:由指令的地址字段给出分支目标。但是,不同的分支指令,直接寻址是有差别的。如果是无条件转移指令,绝对地址中有 24 位由指令给出。对于这 24 位地址,在它的后面补 2 个“0”(因为 RISC 指令都是 32 位的,所以指令的起始字节地址的最后两位一定为 0),前面还有 6 位进行符号位扩展,从而构成 32 位地址。如果是有条件转移指令,指令中只能给出绝对地址中的 16 位,然后在尾部加 2 个“0”,高 14 位是绝对地址的符号扩展,从而构成 32 位地址。

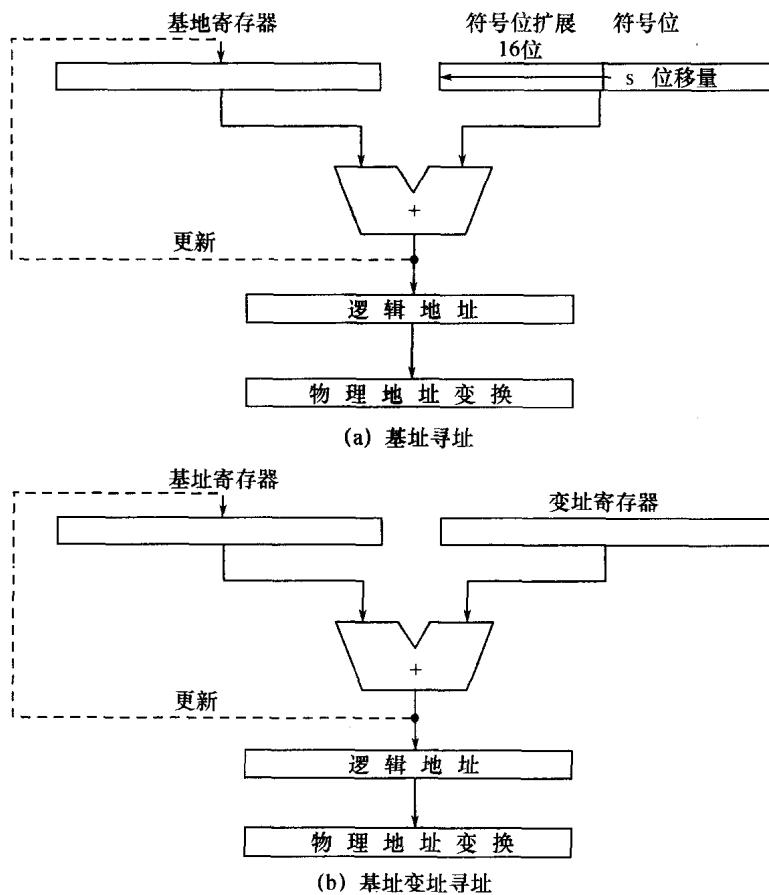


图 2.27 PowerPC 存储器操作数寻址方式

- 相对寻址: 是相对于程序计数器 PC 而言的。其分支的偏移量由指令的地址字段给出, 目标地址为程序计数器的值加上指令中给出的偏移量。如果是无条件转移指令, 指令中将给出偏移量的 24 位; 如果是有条件转移指令, 指令中只能给出偏移量中的 14 位。
- 间接寻址: 在加速分支处理的寄存器中获得分支目标。
- ④ 运算指令的寻址方式
运算指令包括定点计算和浮点计算。
 - 寄存器寻址: 是指操作数存放在通用寄存器中, 指令中包含寄存器的地址。特别要指出的, 寄存器寻址是浮点运算指令唯一支持的寻址方式。
 - 立即数寻址: 是指在指令中包含一个 16 位的有符号的数据量, 它只能是整数。是定点运算指令独有的寻址方式。

可以看出, PowerPC 的运算指令不允许访问主存取操作数, 这也是 RISC 与 CISC 的重要区别之一, 采用这种策略的目的是简化运算指令的功能, 缩短执行周期。

(3) 操作类型

PowerPC 提供了大量的操作类型,如表 2.13 所示。从数量上看,比 Pentium 少了许多,而且只有 load/store 指令可以访问存储器,算术逻辑指令只能对寄存器进行操作,这些都是 RISC 设计的特征。

表 2.13 PowerPC 操作类型

指 令	描 述
	转移有关的
b	无条件转移
bl	转移到目标地址,并将转移指令之后的指令的有效地址放入 Link 寄存器
bc	依据计数寄存器和/或条件寄存器中的位的条件转移指令
sc	要求一个操作系统服务的系统调用
trap	比较两个操作数,并且若指定的条件满足,则调用系统自陷处理程序
	装入/存储
lwzu	装入字并以 0 扩展左边;u 意味着使用修改(update)形式
ld	装入双字
lmw	装入多个字;连续的字装入到由目标寄存器到通用寄存器 31 的相邻各寄存器中
lswx	装字节串到以目标寄存器开始的各寄存器中,每寄存器 4 字节,寄存器 31 环绕道到寄存器 0
	整数算术
add	两个寄存器的内容相加并放入第三个寄存器内
subf	两个寄存器的内容相减并放入第三个寄存器内
mullw	两个寄存器的低序 32 位相乘并将 64 位积放入第三个寄存器内
divd	两个寄存器的 64 位内容相除并将商放入第三个寄存器内
	逻辑和移位
cmp	比较两个操作数并在指定的条件寄存器字段设置 4 个条件位
crand	条件寄存器 AND:条件寄存器两位相“与”,并将结果放入两位之一
and	两个寄存器相“与”并放入第三个寄存器内
cntlzd	统计源寄存器中由位 0 开始的连续个 0 的位数并将计数放入目标寄存器
rldic	旋转左移双字寄存器,“与”上屏蔽,再将结果存入目标寄存器
sld	左移源寄存器的位并存入目标寄存器
	浮点
lfs	由存储器取一 32 位浮点数,转换成 64 位格式,存入浮点寄存器
fadd	两个浮点寄存器内容相加并存入第三个寄存器内
fmadd	两个寄存器内容相乘,加上第三个寄存器内容,存入第四个寄存器内
fcmpu	比较两个浮点操作数并设置条件位
	Cache 管理
dcbf	数据 cache 块清除;完成对指定目标地址在 cache 中的查找并完成清除操作
icbi	指令 cache 块无效

3. 扩展指令系统

指令系统是一个发展的系统,各 CPU 芯片公司为了保持产品市场占有率,不断升级产品,其中的一个重要举措就是扩展指令系统(为保持兼容性,一般不能删减),下面以 Intel 的 MMX 指令扩展为例说明。

MMX 是 Multi - Media eXtension 的缩写,专门指明是由 Intel 公司在 x86 系列微处理器上开发的一组多媒体扩展指令系统。第一个使用 MMX 指令系统的微处理器为 Pentium MMX。这次扩展包括以下主要内容:增加了 8 个新的寄存器;增加 57 条新的指令;以及针对视频、图形、声音等的多媒体优化。8 个新的寄存器是在原有的浮点寄存器的基础上扩展的。x86 浮点处理部件中有一组 80 位的寄存器,包括 16 位指数和 64 位尾数,为了不增加复杂性,MMX 就是用的其中的 64 位。所以,在 Pentium MMX 中,MMX 部件和浮点功能部件不能同时工作,必须进行硬件现场切换,从而导致性能的损失。这一点在 Pentium II 以后的微处理器中得到改善:采用大量寄存器构成的寄存器堆供所有部件使用。

MMX 指令主要包括以下几种类型:

- 算术类,对位向量的并行加、减、乘、乘加,操作数包括压缩的字节、半字、字等;
- 比较类,可以进行压缩数据的比较;
- 转换类,可以进行各种压缩类数据之间的转换和位置调整;
- 移位类,可以对各种压缩类数据进行移位操作;
- 逻辑指令类,进行 64 位逻辑运算;
- 传送类,在 MMX 寄存器和通用寄存器之间进行数据传送;
- 机器状态转换类,在浮点功能和 MMX 功能之间转换。

MMX 指令系统的成功使计算机系统结构设计出现了新的变化。简单地看,MMX 采用了 CISC 指令系统的设计思想。实际上,MMX 是结合 CISC 和 RISC 系统设计的优点而产生的。例如,发挥了 MMX 优势的 Pentium II,它采用了一个 RISC 处理器的内核,使用了大量的 RISC 的特性,使 x86 系列微处理器的性能登上了一个崭新的台阶。

MMX 主要用于增强 CPU 对多媒体信息的处理,提高 CPU 处理 3D 图形、视频和音频信息能力。但由于只对整数运算进行了优化而没有加强浮点方面的运算能力,所以在 3D 图形日趋广泛,因特网 3D 网页应用日趋增多的情况下,MMX 也已心有余而力不足了。针对 MMX 指令系统的弱点和用户的新要求,在 MMX 之后,又出现了新的扩展指令系统,如 AMD 的 3D Now!,Pentium III 中的 SSE 等。

小 结

计算机指令系统的主要内容包括:数据表示、指令类型与指令格式以及寻址方式。

程序员无需知道计算机是采用什么工艺、什么器件、什么结构等,只需了解指令系统,通

过指令系统来和计算机打交道,完成问题的求解。因此各种 CPU 芯片都有其指令系统的相关文档发布,以便程序员编程。指令系统的生存能力,直接影响了计算机系统的生命力。通过对计算机指令系统基本要素的了解,可以进一步深刻理解计算机系统的分析、设计和实现。

在本章讨论的实例为 Pentium 和 PowerPC 微处理器的指令系统。Pentium 代表了在 CISC 几十年努力设计的结晶,它采用了一度只有大型机和超级计算机才使用的设计原则,是 CISC 设计的优秀范例。PowerPC 是第一个 RISC 系统,它是市场上基于 RISC 的功能最强大、设计最好的系统之一。这两种微处理器不但具有代表性,而且它们的后代微处理器在目前占有很大的市场。

习 题

2.1 解释下列术语:

机器指令	寻址方式	定点数据类型	规格化浮点数
机器零	上溢	下溢	字符串
堆栈	向量	堆栈型指令	累加器型指令
通用寄存器型指令	有效地址	形式地址	RISC CISC

2.2 什么是数据表示?试分析数据表示与数据结构研究内容的区别。

2.3 试述计算机发展中数据表示的演变。

2.4 将下列各数表示成 16 位二进制补码定点整数:

378, -2, 2045, 16381

2.5 根据下列给定的数码寄存器位数,确定表示定点整数的数值范围:

8 位, 12 位, 16 位, 24 位, 32 位, 64 位

2.6 20 位数码寄存器(一位符号位)能表示二进制定点整数的数值范围多大?若用 BCD 码表示十进制定点整数,其数值范围多大?

2.7 32 位浮点二进制数,8 位(含一位符号位)为用补码表示的阶码,24 位(含一位符号位)为补码表示的规格化尾数,试指出它所表示的最大正数与最小正数的数据格式。

2.8 最常用的指令格式有哪几种?

2.9 根据给定的真值,求其移码:

+011011011, -11001101, -00010001, +00011101

2.10 如果向量寄存器由 128 个单元组成,要处理向量长度为 L ,怎样分段处理该向量数据?

2.11 根据给定的压缩向量及压缩位向量,还原稀疏向量 A :

A 的压缩向量				A 的压缩位向量							
A_1											
A_3											
A_5											
A_8								1	0	1	0

- 2.12 计算机指令应包含哪些最基本的成分？各有何作用？
- 2.13 试比较寄存器-寄存器型指令与存储器-存储器型指令的优缺点。
- 2.14 选择指令格式应考虑哪些因素？简述其理由。
- 2.15 利用扩展操作码法，构成 40 条指令系统。要求为：
 $OP = 4, 13$ 条； $OP = 7, 6$ 条； $OP = 9, 7$ 条； $OP = 12, 14$ 条。
- 2.16 试述各种寻址方式的特点。
- 2.17 如何利用变址寻址方式，进行 1000 个数在内存中的移动？
- 2.18 试比较变址寻址与基址寻址方式异同点。它们有什么优缺点？
- 2.19 试述常用指令类型有哪些？
- 2.20 完善的指令系统应满足哪些基本要求？
- 2.21 RISC 和 CISC 指令系统各有何特点？

第三章 运算方法与运算器

引　　言

经典冯·诺依曼结构计算机的特点是以运算器为核心。按照存储程序原理，计算机必须具有的五大功能之一是进行数据处理，而运算器就是进行数据处理即执行算术运算和逻辑运算的部件。那么运算器是怎样构成的？它又是怎样实现运算的？这就要涉及到运算方法和运算器的结构。

运算方法讨论的是各种算术运算和逻辑运算的运算规则。计算机的运算最终均可分解成算术(四则)运算与逻辑运算。四则运算的核心是加法运算，减法运算可以通过补码的加法运算实现，乘、除法运算可通过加法运算与移位操作实现，而浮点运算则可以通过分别对阶码和尾数的运算组合实现。

运算器是实现运算方法的硬件。运算器的核心是一个全加器，运算器的逻辑结构取决于机器的指令系统、运算方法和选用的电路等。运算器的硬件组织，一般按下述层次安排：

- 根据一位二进制加法规则产生逻辑表达式，采用基本门电路构成一位全加器；
- n 位全加器连同进位逻辑，构成 n 位并行加法器；
- 采用多路选择逻辑实现多种输入组合，在加法器的基础上扩展逻辑运算功能；
- 加法器与移位寄存器组合构成乘法器与除法器；
- 定点整数的阶码运算器和定点小数的尾数运算器组合构成浮点运算器；
- 再加上寄存器组成运算器 ALU。

由于软件和硬件在逻辑上的等效性，运算方法的实现有许多选择方式，有些功能可以由硬件实现，有些功能可以由软件实现。选择的标准在于如何提高运算的性能而又要使运算器的结构简单，既要使运算具有一定的速度和精度又要节省器件。本章中只讨论运算器的硬件实现，首先介绍基本的算术逻辑运算，然后再讨论运算方法与运算器。

3.1 基本运算

运算器完成的基本操作是算术运算和逻辑运算。本节介绍的是基本的算术运算和逻辑运算，它们具有如下特点：

- 它们都是对寄存器里的数据进行的运算；
- 它们都在一个时钟周期内完成，是计算机中最基本的运算，是最小的具有独立意义的运算；
- 它们都需要控制信号以决定什么时候进行这些运算（关于控制信号和控制信号的形成将在下一章介绍）。

基本运算可以采用如下方法进行表征：

$$\text{目的寄存器 } D \leftarrow \text{op 源寄存器 } S \quad (3.1)$$

式(3.1)表示对源寄存器 S 进行 op 代表的运算后，将结果传送给目的寄存器 D。其中“ \leftarrow ”表示信息的传送方向。

计算机的所有运算均可分解成基本的算术运算和逻辑运算的组合。运算器的设计就是以这些基本运算为基础的。

逻辑运算的对象是逻辑数。逻辑运算的主要特点是：寄存器中的数据按位进行操作，即各位同时进行指定的操作，每位均按二值布尔规则运算，各位之间无进位和溢出，因而比算术运算简单。

算术运算的运算方法比较复杂，每一位的运算结果与其余各位都有关联。同时，机器采用的数据表示和码制不同，其运算规则和方法亦不同。

某些移位运算既有算术运算又有逻辑运算的功能，所以它既是算术运算又是逻辑运算。

3.1.1 逻辑运算

逻辑运算对逻辑数的每位同时进行操作，各位之间没有关系，运算比较简单。

1. 逻辑运算的种类

两个变量 X 和 Y 的逻辑运算可以有 4 种组合： $XY, \bar{X}Y, X\bar{Y}, \bar{X}\bar{Y}$ ，它们在二值空间可组成 16 种逻辑函数。如用 F 表示函数，则：

$$F = \alpha_0 \bar{X}\bar{Y} + \alpha_1 \bar{X}Y + \alpha_2 X\bar{Y} + \alpha_3 XY \quad (3.2)$$

式中，系数 $\alpha_0 \sim \alpha_3$ 为特征值，共有 16 种组合，将其代入上式，即可得 16 种逻辑函数，如表 3.1 所示。表中函数列为函数表达式，运算列为相应的逻辑运算的表征。其中 A 寄存器存放变量 X，B 寄存器存放变量 Y，逻辑运算的结果也被存放于寄存器 A。

表 3.1 两个变量组成的 16 个逻辑函数运算

α_0	α_1	α_2	α_3	函数	名称	运算	α_0	α_1	α_2	α_3	函数	名称	运算
0	0	0	0	$F_0 = 0$	清除	$A \leftarrow 0$	1	0	0	0	$F_8 = \bar{X} + Y$	或非	$A \leftarrow \bar{A} \vee B$
0	0	0	1	$F_1 = XY$	与	$A \leftarrow A \wedge B$	1	0	0	1	$F_9 = \bar{X} \oplus Y$	同或	$A \leftarrow \bar{A} \oplus B$
0	0	1	0	$F_2 = \bar{X}Y$	与非 Y	$A \leftarrow A \wedge \bar{B}$	1	0	1	0	$F_{10} = \bar{Y}$	Y 求反	$A \leftarrow \bar{B}$
0	0	1	1	$F_3 = X$	传送 X	$A \leftarrow A$	1	0	1	1	$F_{11} = X + \bar{Y}$	或非 Y	$A \leftarrow \bar{A} \vee B$

(续表)

$\alpha_0 \ \alpha_1 \ \alpha_2 \ \alpha_3$	函数	名称	运算	$\alpha_0 \ \alpha_1 \ \alpha_2 \ \alpha_3$	函数	名称	运算
0 1 0 0	$F_4 = \bar{X}Y$	与非 X	$A \leftarrow \bar{A} \wedge B$	1 1 0 0	$F_{12} = \bar{X}$	X 求反	$A \leftarrow \bar{A}$
0 1 0 1	$F_5 = Y$	传送 Y	$A \leftarrow B$	1 1 0 1	$F_{13} = \bar{X} + Y$	或非 X	$A \leftarrow \bar{A} \vee B$
0 1 1 0	$F_6 = X \oplus Y$	异或	$A \leftarrow A \oplus B$	1 1 1 0	$F_{14} = \bar{X}Y$	与非	$A \leftarrow \bar{A} \wedge B$
0 1 1 1	$F_7 = X + Y$	或	$A \leftarrow A \vee B$	1 1 1 1	$F_{15} = 1$	置 1	$A \leftarrow 1$

2. 基本逻辑运算及其应用

逻辑运算多用于按位或字段的处理。如用来改变某些指定位的状态;在一个字中取出一部分字段,或插入一部分新的数值;按照另外一个寄存器的内容改现有数据等。

16 种逻辑运算中,清除(置 0)、置 1、与、或、异或、同或等是最基本、最常用的运算,有关其描述及应用分述如下:

(1) 清除运算

清除运算的描述语句为 $A \leftarrow 0$,表示 A 寄存器所有位清 0。这是计算机最常见最基本的运算,可用于建立机器的启动初始状态;清除状态位信息;实现机器工作状态转换;外设工作状态的撤消;清除寄存器内容等。

(2) 置 1 运算

置 1 运算的描述语句为 $A \leftarrow 1$,表示 A 寄存器所有位置 1。也可以根据要求将某些位置 1,例如: $A_i \leftarrow 1, i = 1, 4, 6$,表示将 A 寄存器的第 1 位 A_1 、第 4 位 A_4 以及第 6 位 A_6 置 1。它也是最常用的运算,可用来置寄存器或计数器的初值;置工作方式位以启动机器;实现机器工作状态转换;外设工作状态的建立等。

(3) 逻辑与运算

逻辑与运算的描述语句为 $A \leftarrow A \wedge B$,表示 A、B 寄存器对应位实现逻辑与运算,结果送 A 寄存器。逻辑与运算主要用于数据处理中的“屏蔽”(Mask)或“删除”运算。

例如,若将 A 寄存器后半部分的内容变为 0,则可设置 B 寄存器的前半部分为全 1,后半部分为全 0,通过逻辑与运算实现。如下式:

$$\begin{array}{r}
 11011101 & \text{A 的始值} \\
 \wedge \quad 11110000 & \text{B 值} \\
 \hline
 11010000 & \text{A 的终值: } A \leftarrow A \wedge B
 \end{array}$$

可见,A 寄存器前半部分保持不变,后半部分被清 0。同理,如果要将 A 中的某些位删去,只须将 B 中对应位置 0,其余位置 1,再使用逻辑与运算即可实现。

(4) 逻辑或运算

逻辑或运算的描述语句为 $A \leftarrow A \vee B$,表示 A、B 寄存器对应位实现逻辑或运算,结果送 A 寄存器。逻辑或运算主要用于数据处理中的“选位置 1”(Selective Set)、“插入”(Insert)和

“拼组”(Packing)等运算。

① “选位置1”是指寄存器某些位要求置1时的运算。

例如,要求A寄存器中的 A_3 、 A_5 和 A_8 位被置1,则只要在B寄存器中与之对应的那些位置1,其余位置0,通过执行逻辑或运算实现。如下式:

10011100	A 的始值
\vee	B 值
<hr/>	A 的终值: $A \leftarrow A \vee B$
00101001	
10111101	

可见,B寄存器中是0的那些位对应的A寄存器中各位内容不变,而与B中是1的那些位对应的A中各位内容被置1。

② “插入”运算是指寄存器中某些位的内容用新的数值取代。

例如,要求在A寄存器前4位插入新的数值(1101),需要首先使用逻辑与运算将A的前4位删除,然后再将A的前4位与要求插入的数值作逻辑或运算。如下式:

10010101	A 的始值
\wedge	B 值(删除)
<hr/>	A $\leftarrow A \wedge B$
00000111	
00000101	A 的始值
\vee	B 值(插入)
11010000	
11010101	A $\leftarrow A \vee B$

可见,插入运算时逻辑与和逻辑或两个运算结合使用才能完成的运算。

③ “拼组”运算是指将两个或多个二进制编码信息(如字母)组合成一个字的运算。

例如,将ASCII码中三个字母拼组成一个21位的字,需在21位寄存器A中拼组。设要拼组的字母为“Y”、“O”和“U”,则须从存储器依次读到三个寄存器 R_1 、 R_2 和 R_3 中,前14位为0,后7位放相应的ASCII码,并作如下逻辑运算:

$A \leftarrow 0$	00000000000000000000000000000000
R_1 的值	0000000000000001011001
$A \leftarrow A \vee R_1$	0000000000000001011001
A 左移7位	0000000101100100000000
R_2 的值	0000000000000001001111
$A \leftarrow A \vee R_2$	000000010110011001111
A 左移7位	1011001100111100000000
R_3 的值	0000000000000001010101
$A \leftarrow A \vee R_3$	101100110011111010101

最后在A寄存器中拼组成“YOU”字。拼组运算时逻辑或和逻辑移位(见下节)两种运

算结合使用才能完成的运算。

(5) 异或运算

异或运算的描述语句为: $A \leftarrow A \oplus B$, 表示 A、B 寄存器对应位实现异或运算, 结果送 A 寄存器。异或运算是计算机中应用较广的一种运算, 在逻辑运算中主要用于数据处理中“选位置反”(Selective Complement)以及“比较”(Compare)等运算。

① “选位置反”是指寄存器的某些位要求置反的运算。

例如, 要求 A 寄存器的 A_1 、 A_2 、 A_5 和 A_6 位的内容置反, 则在 B 寄存器与之相对应的位置 1, 其余位置 0, 通过异或运算 $A \leftarrow A \oplus B$ 实现。如下式:

$$\begin{array}{rcl} & 10100101 & \text{A 的始值} \\ \oplus & 11001100 & \text{B 值} \\ \hline & 01101001 & \text{A 的终值: } A \leftarrow A \oplus B \end{array}$$

可见, 经异或运算后, B 中是 1 的那些位相对应的 A 中各对应位内容置反; B 中是 0 的那些位相对应的 A 中各对应位内容保持不变。如果 B 中只有第一位置 1, 其余位置 0, 经异或运算, 只改变 A 中的第一位的内容, 这实际是一种变符号运算。

② “比较”运算是指两个寄存器内容逐位加以比较, 检查内容是否相同的运算, 如两个寄存器内容相同, 经异或运算后, 目的寄存器内容为全 0; 两个寄存器内容不同时, 经异或运算后, 目的寄存器内容不为全 0。

例如, A、B 寄存器内容进行比较, 如下式:

$$\begin{array}{rcl} & 10101010 & \text{A 的始值} \\ \oplus & 10101010 & \text{B 值} \\ \hline & 00000000 & \text{A 的终值: } A \leftarrow A \oplus B \end{array}$$

比较运算后, 还须对目的寄存器进行判 0 操作。可以用软件方法逐位判 0。但是, 一般机器都设有硬件判全 0 或判全 1 的逻辑线路, 直接由硬件实现判 0 将更加迅速。如果有硬件判全 1 线路时, 比较运算也可以通过“同或”运算“ $A \leftarrow A \odot B$ ”实现, 如下式:

$$\begin{array}{rcl} & 10101010 & \text{A 的始值} \\ \odot & 10101010 & \text{B 值} \\ \hline & 11111111 & \text{A 的终值: } A \leftarrow A \odot B \end{array}$$

当 A、B 寄存器内容不同时, 经同或运算后, A 寄存器内容不会为全 1。仅当两个寄存器内容完全相同时, 经同或运算后, A 寄存器内容才为全 1。

3.1.2 移位运算

移位运算既属于算术运算又属于逻辑运算。如果被移位的数据是逻辑数, 则移位运算就是逻辑运算; 另一方面, 移位相当于对数据进行乘除基数的操作(右移一位相当于除以 2,

左移一位相当于乘以 2), 此时的移位运算就是算术运算。移位, 作为一种指令, 它可以独立地被执行; 作为一种基本运算, 它又可以伴生于其他运算型指令之中被执行。这里把移位作为一种基本运算来研究, 这种基本运算不管是独立地形成指令, 还是伴生于其他指令之中, 其功能和原理是一样的。

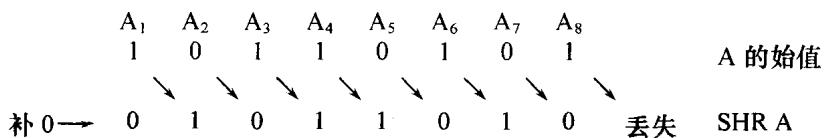
利用移位运算可以进行寄存器间信息的串行传送。在移位时, 寄存器“末端”触发器的状态决定于移位类型。“末端”触发器, 在右移时是指参加移位的数据最左边那位对应的触发器; 在左移时是指参加移位的数据最右边那位对应的触发器。

移位的类型按移位性质可分为逻辑移位、循环移位和算术移位; 按被移位数据长度可分为字节移位、半字长移位、全字长移位、双倍或多倍字长移位; 按每次移位的位数可分为移一位和移多位。

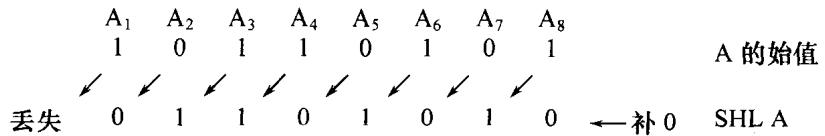
计算机中移位指令应指明移位性质、被移位数据长度和一次移位的位数。被移位的数据长度和一次移位的位数取决于机器的指令系统, 也比较好理解。这里将重点讨论按移位性质来划分的 3 种类型移位运算。

1. 逻辑移位

逻辑移位 (Logic Shift) 是指对寄存器中整组数据进行的移位。逻辑移位按照移位的方向又分为逻辑右移和逻辑左移, 分别用语句 $\text{SHR } A$ 和 $\text{SHL } A$ 表示, 移位时寄存器末端触发器补入 0。需要说明的是, 移位运算的符号可以具有数字下标, 代表移动的位数。例如 $\text{SHR}_2 A$ 表示对 A 寄存器中的数据右移两位, 而没有数字下标一般表示移一位。例如, 对 A 寄存器进行逻辑右移, 如下式:



结果 A 寄存器的内容逻辑右移一位, A_8 的内容右移后被丢失, 末端触发器 A_1 补入 0。再如, 对 A 寄存器进行逻辑左移, 如下式:

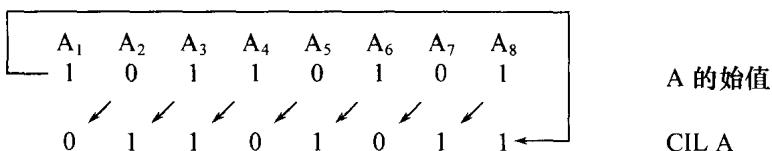


结果 A 寄存器的内容逻辑左移一位, A_1 的内容左移后被丢失, 末端触发器 A_8 补入 0。

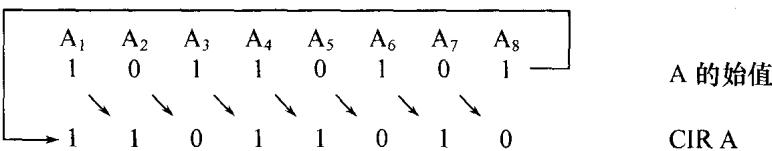
逻辑移位运算主要用于数据处理中字的装配、拼组与拆卸等操作, 也可用于程序控制中的状态位和特殊信息的调用。当移位寄存器的末端触发器与其他寄存器有移位通路时, 逻辑移位运算还可实现信息的串行传送。

2. 循环移位

循环移位(Circular Shift)是指寄存器两端触发器有移位通路,形成闭合的移位环路。在移位时,寄存器内信息便在这个环路内循环流动。循环移位语句为 CIR A 或 CIL A,它们分别表示 A 寄存器的内容循环右移或循环左移。例如,对 A 寄存器进行循环左移,如下式:



结果 A 寄存器的内容循环左移一位,A₁ 的内容被移到 A₈。再如,对 A 寄存器进行循环右移,如下式:



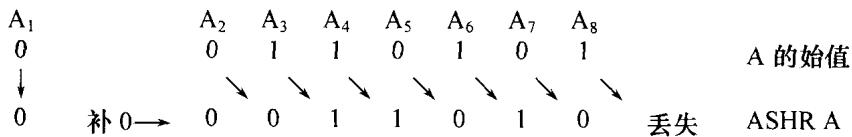
结果 A 寄存器的内容循环右移一位,A₈ 的内容被移到 A₁。

循环移位主要用于寄存器移位时信息仍须保留的情况。例如,当移位寄存器 B 的末端触发器与寄存器 A 有移位通路时,要将 B 寄存器的内容传送到 A 寄存器且 B 寄存器内容仍保持不变,则可通过对 B 寄存器与 A 寄存器执行循环移位运算实现。

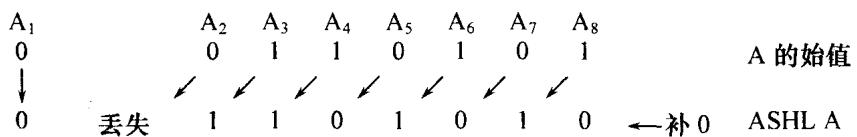
3. 算术移位

算术移位(Arithmetic Shift)是指寄存器带符号数的移位。执行算术移位时,寄存器中的数是带符号的算术运算数,可以是定点数也可以是浮点数的尾数或者阶码。移位后,数的符号位保持不变。算术移位语句为 ASHR A 或 ASHL A,它们分别表示 A 寄存器的内容算术右移或算术左移。

算术运算中,数的移位运算通常会引起数值的变化:右移一位,相当于带符号的数除以 2(乘以 1/2);左移一位,相当于带符号的数乘以 2。例如,对 A 寄存器进行算术右移,如下式:



结果 A 寄存器的内容算术右移一位,符号位 A₁ 保持不变,末端触发器 A₂ 的内容补 0。右移一位相当于对数据进行除以 2 的运算。寄存器 A 的始值($+0110101_2 = (+53)_{10}$),右移一位后将结果取整: $\lfloor +53/2 \rfloor = +26$, ($+26)_{10} = (+0011010)_2$ 。再如,对 A 寄存器进行算术左移,如下式:



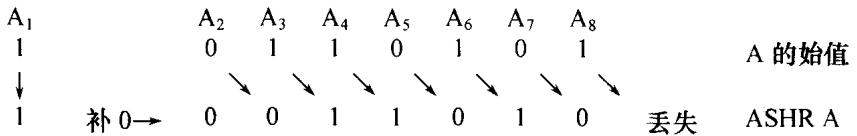
结果 A 寄存器的内容算术左移一位, 符号位 A_1 保持不变, 末端触发器 A_8 的内容补 0。左移一位相当于对数据进行乘以 2 的运算。寄存器 A 的始值 $(+0110101)_2 = (+53)_{10}$, 左移一位后: $+53 \times 2 = +106$, $(+106)_{10} = (+1101010)_2$ 。因此, 算术移位运算是使被移位的数增大 1 倍或减小 1 倍的运算, 但数的增大或减小并不影响数的符号。算术移位运算的符号保持不变, 这一点是与其他类型移位的主要区别所在。

带符号的数有三种常见表示法: 原码、补码和反码。对于正数, 符号位为 0, 三种码制的表示结果相同, 算术移位时末端触发器均补入 0, 前述算术右移与算术左移的举例即为正数移位情况。对于负数, 符号位为 1, 在算术移位时其末端触发器补入的数将随码制的不同而不同。

(1) 原码的算术移位

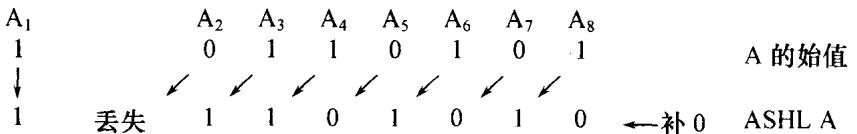
原码表示的数是由符号位与绝对值组成。执行算术移位运算时, 符号位不参与移位, 只是绝对值移位, 其末端触发器补入 0。无论正数还是负数, 算术运算中的移位运算通常会引起数值的变化。

例如, A 寄存器中为一个原码表示的负数, 若进行算术右移运算, 其移位过程如下式:



寄存器 A 的始值 $10110101 = [-0110101]_{原}, (-0110101)_2 = (-53)_{10}$, 右移一位后将结果取整: $\lfloor -53/2 \rfloor = -26$, $(-26)_{10} = (-0011010)_2, [-0011010]_{原} = 10011010$ 。结果表明: A_1 的内容为符号位, 不参与移位; A_8 的内容被移出后丢失; 末端触发器 A_2 补入 0。

再如, 若对 A 寄存器内容进行算术左移运算, 其移位过程如下式:



寄存器 A 的始值 $10110101 = [-0110101]_{原}, (-0110101)_2 = (-53)_{10}$, 左移一位后: $-53 \times 2 = -106$, $(-106)_{10} = (-1101010)_2, [-1101010]_{原} = 11101010$ 。结果表明: A_1 的内容为符号位, 不参与移位; A_2 的内容被移出后丢失; 末端触发器 A_8 补入 0。

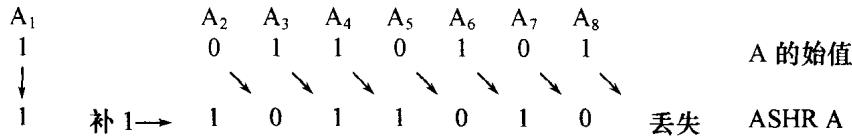
(2) 补码的算术移位

补码表示的数进行算术运算时,符号位要参与运算,即补码的符号位可以作为数的一部分一起参加运算。对于补码表示的负数,问题在于算术移位运算后怎样才能保持正确的符号和正确的移位值。

先考虑右移情况。假设数 $X = -1011000$, 其补码 $[X]_{\text{补}} = 10101000$, 第一位为符号位。现将其真值 X 右移,再求出其对应的补码,列式如下:

不移位	$X = -1011000$	$[X]_{\text{补}} = 10101000$
右移一位	$\frac{1}{2}X = -0101100$	$[\frac{1}{2}X]_{\text{补}} = 11010100$
右移二位	$\frac{1}{4}X = -0010110$	$[\frac{1}{4}X]_{\text{补}} = 11101010$

比较真值移位后对应的补码,可以发现,负数的补码右移时,只要将其符号位也随之右移且符号位不变即可。例如,A 寄存器中为一个补码表示的负数,现进行算术右移运算,其移位过程如下式:

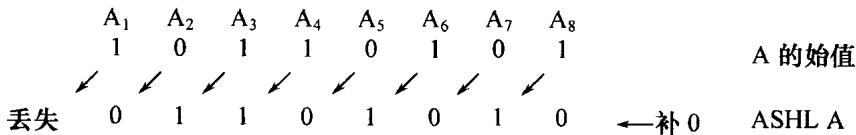


寄存器 A 的始值 $10110101 = [-1001011]_{\text{补}}, (-1001011)_2 = (-75)_{10}$, 右移一位后将结果取整: $\lceil -75/2 \rceil = -38, (-38)_{10} = (-0100110)_2, [-0100110]_{\text{补}} = 11011010$ 。

再考虑左移情况。假设数 $Y = 0010110$, 其补码为 $[Y]_{\text{补}} = 11101010$, 第一位为符号位。现将其真值 Y 左移,再求出其对应的补码,列式如下:

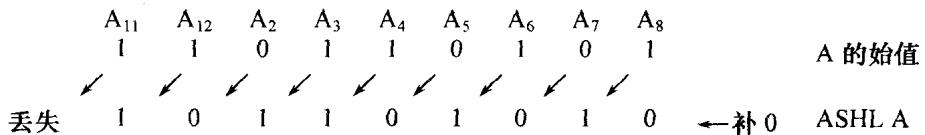
不移位	$Y = -0010110$	$[Y]_{\text{补}} = 11101010$
左移一位	$2Y = -0101100$	$[2Y]_{\text{补}} = 11010100$
左移二位	$4Y = -1011000$	$[4Y]_{\text{补}} = 10101000$

比较真值移位后对应的补码,可以发现,负数的补码左移时,其末端触发器补入 0,在该例中数值高位移入符号位仍会得到正确的符号。但是,如果负数符号位的后一位是 0,在左移时就会出现溢出即超出数的表示范围,这样将破坏正确的符号位。例如:



寄存器 A 的始值 $10110101 = [-1001011]_{\text{补}}, (-1001011)_2 = (-75)_{10}$, 左移一位后: $-75 \times 2 = -150$, 而 $|-150| > |-128|$ 。结果表明,若 A 中为绝对值较大的负数,左移扩大 1 倍后超出了数的表示范围,因而产生溢出,符号位改变成 0(表示正数),这是一个错误的结

果。为了保持正确的符号,往往采用补码变形码,使用两位或多位符号位,可以克服改变符号的错误。例如:



此例中,补码变形码采用两位符号位 A_{11} 和 A_{12} , A_{11} 代表数的正确符号,若 A_{12} 为 0 则表示其绝对值是大于 2^7 的一个负数。补码算术移位后两个符号位出现 10 情况则表示负数溢出。当然,溢出也可能发生在对正数进行左移的时候,当补码算术移位后两个符号位出现 01 情况则表示正数溢出。

综上所述,对于补码表示数的算术移位,按上述规则操作:

- 符号位参与操作;
- 右移时,其末端触发器补入符号位;
- 左移时,其末端触发器补入 0。

(3) 负数反码的算术移位

反码表示的数进行算术运算时,符号位也要参与运算,即反码的符号位亦可以作为数的一部分一起参加运算。反码表示数的算术移位,可作与补码数算术移位类似的分析,得到如下结论:

- 符号位参与操作;
- 对正数,移位时其末端触发器补入 0;
- 对负数,移位时其末端触发器补入 1。

读者可自行推导证明。

3.1.3 算术运算

在计算机的各种运算或控制当中,还会遇到多种算术运算。本小节只讨论几种简单的算术运算的功能和描述语句,对于求反和求补运算,还略述其逻辑实现。

1. 加运算

加运算的描述语句为 $A \leftarrow A + B$, 表示 A 寄存器内容与 B 寄存器内容作加法运算,其和存入 A 寄存器。从这个语句中,除了要理解其操作意义,还需理解其硬件实现。它不仅需要两个寄存器(其中一个既是源寄存器也是目的寄存器),还需要有实现加运算的逻辑电路——加法器和输入开关等。如果有三个寄存器参与加运算,其描述语句为: $C \leftarrow A + B$, 表示 A 和 B 寄存器的内容相加,其结果存入 C 寄存器。

加法语句主要是描述并行加法运算的,用它也可以描述串行加法运算,但串行加法的运算时间较长,所以加法的控制信号的持续时间比较长。关于控制信号和控制信号的形成将

在下一章中介绍。

加法运算可以形成独立的指令。

2. 求反、求补运算

求反运算的描述语句为 $A \leftarrow \overline{A}$, 表示 A 寄存器各位的内容取反后仍存于 A 寄存器。实现此运算, 寄存器需要有求反线路。

求补运算的描述语句为 $A \leftarrow \overline{A} + 1$, 表示 A 寄存器各位内容取反, 最低位加 1 后仍存于 A 寄存器。实现此运算, 寄存器需要求补线路。为了在一个时钟周期内完成求补运算, 可采用图 3.1 所示的逻辑线路。

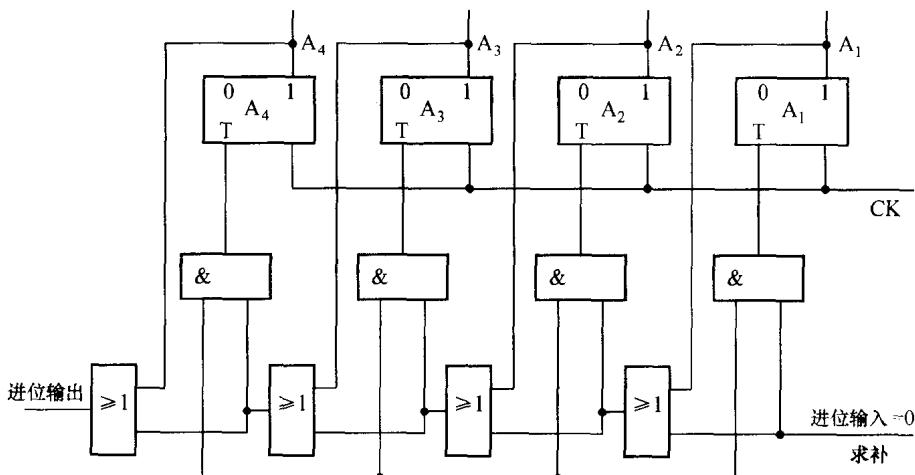


图 3.1 具有求补功能的四位寄存器

图 3.1 中的寄存器由 4 个 T 触发器构成, 当 Ck 脉冲到达时, 如果 T 触发器输入端为 1, A_i 的状态就发生翻转。该逻辑线路是根据负数的简便求补规则导出的, 即二进制数据中最低(最右边)的 1 及其右边的 0 保持不变, 而将所有其他各位置反。这样, 最低位 A_1 不论是 1 还是 0 总保持不变, 其他各位只要求补控制信号有效且它的右边有一位是 1 就置反。 A_4 的进位输出可以连到下一级进位输入端, 就可以将四位寄存器的求补线路扩展成更多位寄存器的求补线路。

求补、求反运算可以形成独立的指令。

3. 减运算

减运算的描述语句为 $A \leftarrow A - B$, 表示 A 寄存器内容与 B 寄存器内容作减法运算, 其差存入 A 寄存器。实现这个语句需要两个寄存器和实现减法运算的逻辑电路——减法器及输入开关等。

一般减法运算均是通过补码加法来实现, 故其描述语句可改为 $A \leftarrow A + \overline{B} + 1$, 表示 A 寄

存器内容加上 B 寄存器内容的求补结果, 其和存入 A 寄存器, 它等效于 $A \leftarrow A - B$ 运算。

与加法运算一样, 减法运算也可以形成独立的指令。

4. 递增和递减运算

递增运算的描述语句为 $A \leftarrow A + 1$, 表示 A 寄存器内容加 1 后再存入 A 寄存器。实现此运算, 要求 A 寄存器具有正向计数功能。

递减运算的描述语句为 $A \leftarrow A - 1$, 表示 A 寄存器内容减 1 后再存入 A 寄存器。实现此运算, 要求 A 寄存器具有逆向计数功能。

当然, 实现递增与递减运算也可以通过加法线路来实现。用于控制的各种计数器, 多用到此两种运算, 其递增或递减量也可为大于 1 的其他整数。

3.2 定点加(减)法运算

算术运算和逻辑运算是计算机的两大基本运算功能, 前者是数值运算的基础, 后者则是非数值运算的基础。而定点加(减)法运算又是算术运算的基础。

运算方法与数据表示和编码相关, 本章在介绍定点四则运算方法时, 主要介绍原码和补码的运算方法。

本节首先分别讨论原码、补码和移码的加(减)法运算, 然后介绍二进制加法器、多功能算术逻辑部件 ALU, 最后介绍十进制加法器。

3.2.1 二进制加(减)法运算

定点加(减)法运算既可采用原码也可采用补码和移码进行, 不同码制下运算方法的特点不同。原码加(减)法比较复杂, 因为当两操作数异号时, 加法实际变成了减法, 减法实际又变成了加法。另外, 在进行减法处理时, 还需要使用求补运算, 这样在原码运算时还需要使用补码, 导致计算机系统将同时使用两种码制进行运算。目前的计算机系统中, 应用最普遍的是补码表示, 即以补码形式存储、传送和加工数据。浮点数据的阶码多采用移码表示, 移码加(减)法对于浮点数据的阶码运算十分重要。

需要说明的是, 定点数据表示分为定点整数和定点小数两种数据形式。由于浮点数据的尾数表示为定点小数, 阶码表示为定点整数, 故本小节以定点小数为例讨论原码和补码的加(减)运算, 而以定点整数讨论移码的加(减)运算。

1. 二进制原码加(减)法

在进行原码加(减)法时, 操作数与运算结果均用原码表示, 运算时尾数进行加(减), 符号位则单独处理, 不能参加运算。

设参加运算的为两个 $n+1$ 位定点小数。在本章中, 如果不特别指出, $n+1$ 位的定点小数都由 1 位符号位和 n 位尾数构成。如果使用原码完成 $C = A \pm B$ 的计算, 记:

被加(减)数 A 的原码: $[A]_{\text{原}} = A_0.A_1A_2 \cdots A_n$;

加(减)数 B 的原码: $[B]_{\text{原}} = B_0.B_1B_2 \cdots B_n$;

和(差)C 的原码: $[C]_{\text{原}} = C_0.C_1C_2 \cdots C_n$ 。记 $[C]_{\text{原}}$ 的尾数部分为 $C_{\text{尾}}$ 。

设机器操作后的结果为 $C' = C'_0.C'_1C'_2 \cdots C'_n$ 。记 C' 的尾数部分为 $C'_{\text{尾}}$ 。

原码数据数进行加(减)运算时, 必须考虑符号位。原码加(减)法的规则列于表 3.2。

表 3.2 原码加(减)法规则

指令	A_0	B_0	机器操作	尾数	符号	溢出
+ +	0 1	0 1	$ A + B $	$C_{\text{尾}} = C'_{\text{尾}}$	$C_0 = A_0 = 0$	$C'_0 = 1$ 正溢出
+ +	1 0	1 0			$C_0 = A_0 = 1$	$C'_0 = 1$ 负溢出
+ -	0 1	1 0	$ A + [- B]_{\text{补}}$	若 $C'_0 = 0$ 则 $C_{\text{尾}} = C'_{\text{尾}}$ 若 $C'_0 = 1$ 则 $C_{\text{尾}} = [C'_{\text{尾}}]_{\text{补}}$	$C_0 = A_0 (C'_0 = 0)$	无溢出
- -	0 1	0 1			$C_0 = B_0 (C'_0 = 1)$	
- -	0 1	1 0	$ A + B $	$C_{\text{尾}} = C'_{\text{尾}}$	$C_0 = A_0 (C'_0 = 0)$	
- -	1 0	0 1			$C_0 = \bar{B}_0 (C'_0 = 1)$	
- -	0 1	1 0			$C_0 = A_0 = 0$	$C'_0 = 1$ 正溢出
- -	1 0	0 1			$C_0 = A_0 = 1$	$C'_0 = 1$ 负溢出

由表 3.2 易得到如下结论:

(1) 相加操作条件

“指令加 $\cdot (A_0 = B_0)$ + 指令减 $\cdot (A_0 \neq B_0)$ ”成立时, 机器完成操作: $C' = |A| + |B|$

和(差)的结果: $[C]_{\text{原}} = A_0 + C'_{\text{尾}} = A_0.C'_1C'_2 \cdots C'_n$

(2) 相减操作条件

“指令加 $\cdot (A_0 \neq B_0)$ + 指令减 $\cdot (A_0 = B_0)$ ”成立时, 机器完成操作: $C' = |A| + [-|B|]_{\text{补}}$

当 $C'_0 = 0$ 时, 和(差)的结果: $[C]_{\text{原}} = A_0 + C'_{\text{尾}} = A_0.C'_1C'_2 \cdots C'_n$

当 $C'_0 = 1$ 时, 和的结果: $[C]_{\text{原}} = B_0 + [C'_{\text{尾}}]_{\text{补}} = B_0.\bar{C}'_1\bar{C}'_2 \cdots \bar{C}'_n + 2^{-n}$

当 $C'_0 = 1$ 时, 差的结果: $[C]_{\text{原}} = \bar{B}_0 + [C'_{\text{尾}}]_{\text{补}} = \bar{B}_0.\bar{C}'_1\bar{C}'_2 \cdots \bar{C}'_n + 2^{-n}$

(3) 溢出判断

无论是指令加或指令减, 只有做 $|A| + |B|$ 操作时机器才可能产生溢出。而做 $|A| + [-|B|]_{\text{补}}$ 操作时, 相当于两异号数相加, 永远不会产生溢出。因此, 溢出条件是:

$$\text{正溢出} = (|A| + |B|) \cdot (C'_0 = 1) \cdot (A_0 = 0)$$

$$\text{负溢出} = (|A| + |B|) \cdot (C'_0 = 1) \cdot (A_0 = 1)$$

(4) 符号位

原码加(减)法, 因符号位无需参加运算和参与判断溢出, 故只需一位符号位。

从原码加(减)法的运算规则可以看出, 原码运算规则比较复杂, 其对应的运算器势必也

会较复杂。

2. 二进制补码加(减)法

补码加(减)法运算有以下规则:操作数用补码表示,结果也用补码表示;两数的符号位参加运算。运算依据的关系是:

$$[A + B]_{\text{补}} = [A]_{\text{补}} + [B]_{\text{补}} \pmod{2} \quad (3.3)$$

$$[A - B]_{\text{补}} = [A]_{\text{补}} + [-B]_{\text{补}} \pmod{2} \quad (3.4)$$

式(3.3)表明:两数作补码加法运算时,可直接将补码表示的两操作数相加,无须考虑它们的数符是正或负,所得结果即为补码表示的和,即两数和的补码等于两数补码之和。

式(3.4)表明:为使补码减法运算变成补码加法运算,可将减去减数转换为与减数的机器负数相加。这里 $[-B]_{\text{补}}$ 就是 $[B]_{\text{补}}$ 的机器负数,通过对补码数连同其符号位求反加1而得。因此,两数差的补码等于被减数的补码加上减数的机器负数。

为便于判断溢出,补码采用两位符号位,即采用变形补码。这样,加(减)法运算为:

$$[A + B]_{\text{补}} = [A]_{\text{补}} + [B]_{\text{补}} \pmod{4} \quad (3.5)$$

$$[A - B]_{\text{补}} = [A]_{\text{补}} + [-B]_{\text{补}} \pmod{4} \quad (3.6)$$

式(3.5)和式(3.6)中使用的变形码只是在运算过程中采用。在正常情况下,由于两符号位是相同的,故在传送和存储时仍只保留一个符号位。

因此,补码加(减)运算规则可归纳如下:

- 参与运算的操作数用补码表示;
- 采用两位符号位,即用变形补码表示;
- 符号位作为数的一部分参与运算;
- 运算结果以补码表示,若两符号位相同,结果正常;若符号位为01,表示正溢出;若符号为10,表示负溢出。

例 3.1 已知 $A = 0.1011, B = -0.1110$, 求 $[A + B]_{\text{补}} = ?$

解: $[A]_{\text{补}} = 00.1011, [B]_{\text{补}} = 11.0010$, 其演算过程及结果如算式 3.1 所示。

例 3.2 已知 $A = 0.1011, B = -0.0010$, 求 $[A - B]_{\text{补}} = ?$

解: $[A]_{\text{补}} = 00.1011, [-B]_{\text{补}} = 00.0010$, 其演算过程及结果如算式 3.2 所示。

$\begin{array}{r} 00.1011 \\ +) \quad 11.0010 \\ \hline 11.1101 \end{array}$	$\begin{array}{r} 00.1011 \\ +) \quad 00.0010 \\ \hline 00.1101 \end{array}$
$[A + B]_{\text{补}} = 11.1101$	$[A - B]_{\text{补}} = 00.1101$
算式 3.1	算式 3.2

例 3.3 已知 $A = -0.1101, B = -0.1010$, 求 $[A + B]_{\text{补}} = ?$

解: $[A]_{\text{补}} = 11.0011, [B]_{\text{补}} = 11.0110$, 其演算过程及结果如算式 3.3 所示。

例 3.4 已知 $A = 0.1101, B = -0.1010$, 求 $[A - B]_{\text{补}} = ?$

解: $[A]_{\text{补}} = 00.1101, [-B]_{\text{补}} = 00.1010$, 其演算过程及结果如算式 3.4 所示。

$$\begin{array}{r} 11.0011 \\ +) \quad 11.0110 \\ \hline 10.1001 \end{array}$$

结果为负溢出

算式 3.3

$$\begin{array}{r} 00.1101 \\ +) \quad 00.1010 \\ \hline 01.0111 \end{array}$$

结果为正溢出

算式 3.4

由于数据以补码形式存储, 各位求反加 1 操作也易于实现, 而且符号位既不单独处理, 也无须按参加运算数的符号决定怎样进行运算, 运算十分方便, 故在大多数计算机中均采用补码加(减)法运算。

3. 二进制移码加(减)法

两个 $n+1$ 位定点整数(包含 1 位符号位)的移码加(减)法运算有以下规则: 操作数用移码表示, 结果也用移码表示; 两数的符号位参加运算。由于移码是在补码的基础上反符号位得到的, 故 $n+1$ 位定点整数 A (包括 1 位符号位)的移码为:

$$[A]_{\text{移}} = [A]_{\text{补}} + 2^n \quad (\bmod 2^{n+1}) \quad (3.7)$$

因此, 移码加(减)运算可以依据如下公式:

$$\begin{aligned} [A + B]_{\text{移}} &= [A + B]_{\text{补}} + 2^n \\ &= [A]_{\text{补}} + [B]_{\text{补}} + 2^n \quad (\bmod 2^{n+1}) \end{aligned} \quad (3.8)$$

$$\begin{aligned} &= [A]_{\text{移}} + [B]_{\text{补}} \\ &= [A]_{\text{移}} + [B]_{\text{移}} + 2^n \end{aligned}$$

$$\begin{aligned} [A - B]_{\text{移}} &= [A - B]_{\text{补}} + 2^n \\ &= [A]_{\text{补}} + [-B]_{\text{补}} + 2^n \quad (\bmod 2^{n+1}) \end{aligned} \quad (3.9)$$

$$\begin{aligned} &= [A]_{\text{移}} + [-B]_{\text{补}} \\ &= [A]_{\text{移}} + [-B]_{\text{移}} + 2^n \end{aligned}$$

在使用移码进行加(减)运算时, 通常使用如下公式:

$$[A + B]_{\text{移}} = [A]_{\text{移}} + [B]_{\text{补}} \quad (\bmod 2^{n+1}) \quad (3.10)$$

$$[A - B]_{\text{移}} = [A]_{\text{移}} + [-B]_{\text{补}} \quad (\bmod 2^{n+1}) \quad (3.11)$$

为便于判断溢出, 移码采用两位符号位, 即采用变形移码: 数据的第 1 位符号为 0, 而第 2 位符号代表数据的正负。即当 A 为正数时, $[A]_{\text{移}}$ 的符号为 01; 而当 A 为负数时, $[A]_{\text{移}}$ 的符号为 00。这样, 加(减)法运算可写作为:

$$[A + B]_{\text{移}} = [A]_{\text{移}} + [B]_{\text{补}} \quad (\bmod 2^{n+2}) \quad (3.12)$$

$$[A - B]_{\text{移}} = [A]_{\text{移}} + [-B]_{\text{补}} \quad (\bmod 2^{n+2}) \quad (3.13)$$

式(3.12)和式(3.13)中使用的变形码只是在运算过程中采用。在正常情况下, 只有第 2 位

符号代表数据的正负,故在传送和存储时仍只保留一位符号位。

因此,移码加(减)运算规则可归纳如下:

- 参与运算的操作数用移码表示;
- 采用两位符号位,即用变形移码表示;
- 符号位作为数的一部分一起参与运算;
- 运算结果以移码表示,若第1位符号为0,结果正常;若第1位符号为1则表示溢出:符号位为10表示正溢出,符号位为11时表示负溢出。

当然,也可以将变形移码的第一位符号位设置为1,此时的运算规则读者可以自行推导。

例 3.5 已知 $A = 1011, B = -1110$, 求 $[A + B]_{\text{移}} = ?$

解: $[A]_{\text{移}} = 011011, [B]_{\text{补}} = 110010$, 其演算过程及结果如算式 3.5 所示。

例 3.6 已知 $A = 1011, B = -0010$, 求 $[A - B]_{\text{移}} = ?$

解: $[A]_{\text{移}} = 011011, [-B]_{\text{补}} = 000010$, 其演算过程及结果如算式 3.6 所示。

$\begin{array}{r} 011011 \\ +) \quad 110010 \\ \hline 001101 \end{array}$	$\begin{array}{r} 011011 \\ +) \quad 000010 \\ \hline 011101 \end{array}$
$[A + B]_{\text{移}} = 001101$ 算式 3.5	$[A - B]_{\text{移}} = 011101$ 算式 3.6

例 3.7 已知 $A = -1101, B = -1010$, 求 $[A + B]_{\text{移}} = ?$

解: $[A]_{\text{移}} = 000011, [B]_{\text{补}} = 110110$, 其演算过程及结果如算式 3.7 所示。

例 3.8 已知 $A = 1101, B = -1010$, 求 $[A - B]_{\text{移}} = ?$

解: $[A]_{\text{移}} = 011101, [-B]_{\text{补}} = 001010$, 其演算过程及结果如算式 3.8 所示。

$\begin{array}{r} 000011 \\ +) \quad 110110 \\ \hline 111001 \end{array}$	$\begin{array}{r} 011101 \\ +) \quad 001010 \\ \hline 100111 \end{array}$
结果为负溢出 算式 3.7	结果为正溢出 算式 3.8

同补码一样,移码符号位也参加运算,运算规则和判溢出条件都比较简单。在移码加(减)运算中, $[B]_{\text{补}}$ 或 $[-B]_{\text{补}}$ 可以通过对 $[B]_{\text{移}}$ 或 $[-B]_{\text{移}}$ 的符号位取反得到,因此增加的电路不多。在浮点数据表示时,阶码通常使用移码来表示。

3.2.2 二进制补码加法器

一位二进制加法单元有3个输入量:操作数 A_i 和 B_i ,低位来的进位信号 C_{i-1} 。运算后产生本位和 S_i 及向高位的进位信号 C_i 。具有全部3个输入的加法单元被称为全加器(Full Adder, FA)。如果只考虑2个输入相加,不考虑低位进位信号,则叫做半加器(Half Adder, HA)。

如果 n 位二进制数据相加,分成 n 步实现,即每步只求一位和和该位的进位,则这种加法器叫串行加法器(Serial Adder)。串行加法器成本低,但速度慢,现代计算机中几乎不予采用。如果 n 位二进制数据同时相加,即用 n 位全加器一步实现 n 位数据相加,这种加法器称为并行加法器(Parallel Adder)。

并行加法器所用全加器的位数与操作数位数相同。虽然操作数的各位是同时提供,但低位运算所产生的进位将会影响高位的运算结果,故存在进位信号的传递问题。由于进位传递所经过的门电路级数一般超过各位全加器的门电路级数,即进位传递延迟大于全加器本身的延迟,因此并行加法器的运算速度不但与全加器本身有关,更与进位传递的速度有关。

本节首先简介全加器的结构与工作原理,然后讨论并行加法器进位传递及其加速问题。

1. 全加器

从“数字逻辑原理与工程设计”课程中可知,只要将一位求和逻辑用真值表描述,用卡诺图化简,就可得到采用门电路构成的全加器。如果采用异或逻辑实现半加,用两次半加实现一位全加,则构成全加器的逻辑结构非常简单,也有利于快速进位传递。

图 3.2 是分别采用原变量和反变量输入的全加器。

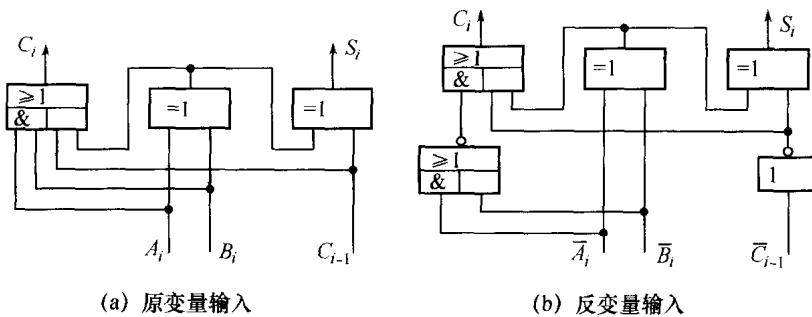


图 3.2 用半加器构成的全加器

当采用原变量输入时,其本位和 S_i 及向高位的进位 C_i 分别为:

$$S_i = A_i \oplus B_i \oplus C_{i-1} \quad (3.14)$$

$$C_i = A_i B_i + (A_i \oplus B_i) C_{i-1} \quad (3.15)$$

当采用反变量输入时, S_i 和 C_i 分别为:

$$S_i = (\bar{A}_i \oplus \bar{B}_i) \oplus \bar{\bar{C}}_{i-1} \quad (3.16)$$

$$C_i = \bar{\bar{A}}_i + \bar{\bar{B}}_i + (\bar{A}_i \oplus \bar{B}_i) \bar{\bar{C}}_{i-1} \quad (3.17)$$

式(3.14)表明:第一次半加只考虑了本位的两个输入 A_i 和 B_i ,第二次半加则考虑了低位进位 C_{i-1} ;如三个输入中有奇数个 1,则本位和 S_i 为 1。式(3.15)表明,产生进位的条件有两项:当本位的两个输入 A_i, B_i 均为 1 时,不管低位有无进位 C_{i-1} ,都会产生进位;若 C_{i-1} 为 1,则只要 A_i, B_i 中有一个为 1,也会产生进位。式(3.16)和式(3.17)为采用反变量时的情况。

显而易见,对于一位全加器,采用原变量或反变量输入并无实质上的差异。但在构成 n 位加法器时,则需考虑低位来的进位怎样与高一位全加器的极性相配合。在构成完整的 ALU 时,还需考虑输入操作数与全加器输入之间的极性配合。这样做的目的是使进位传递延时尽可能的小。

2. 并行加法器的进位传递

由于进位是由低位向高位逐级传递,进位的逻辑结构形似链条,故常称进位传递逻辑为进位链(Carry Link)。并行加法器的结构可分为全加器与进位链两部分。

对式(3.15)进行推导有:

$$C_i = A_i B_i + (A_i \oplus B_i) C_{i-1} = A_i B_i + (A_i + B_i) C_{i-1} = G_i + P_i C_{i-1} \quad (3.18)$$

其中 $G_i = A_i B_i$, 称为第 i 位的进位产生函数(Carry Generate Function)或进位生成条件,或称为本地进位或绝对进位。它表明:若本位的两个输入量均为 1, 必产生进位, 它是不受进位传递影响的分量。 $P_i = A_i + B_i$, 称为第 i 位的进位传递函数(Carry Propagate Function)或进位传递条件, 而 $P_i C_{i-1}$ 则称为传递进位或条件进位。它表明:若本位的两个输入中至少有一个为 1, 且低位有进位传来时, 本位将产生进位。换言之, 当 $P_i = 1$ 时, 低位来的进位 C_{i-1} 将通过本位向高位传递。式(3.18)说明:进位由本地进位和传递进位两部分构成, 它是构成各种进位链结构的基本逻辑表达式。

(1) 串行进位

所谓串行进位就是逐级地形成各位进位, 每一级进位直接依赖于前一级进位, 通常称行波进位(Ripple Carry)。设第一位为最低位, 第 n 位为最高位, 则采用串行进位的 n 位并行加法器各进位信号的表达式为:

$$\left. \begin{aligned} C_1 &= G_1 + P_1 C_0 = A_1 B_1 + (A_1 + B_1) C_0 \\ C_2 &= G_2 + P_2 C_1 = A_2 B_2 + (A_2 + B_2) C_1 \\ &\dots \\ C_n &= G_n + P_n C_{n-1} = A_n B_n + (A_n + B_n) C_{n-1} \end{aligned} \right\} \quad (3.19)$$

采用串行进位的并行加法器如图 3.3 所示。

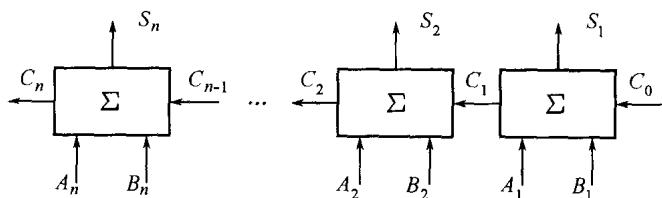


图 3.3 采用串行进位的并行加法器

图 3.3 所示的并行加法器, 在 n 位全加器之间, 进位信号采用串行传递。这种结构所用元件较少, 但进位传递时间较长。显而易见, 当各位全加器的两个输入中都只有一个 1, 而初

始进位 C_0 又为 1 时, 进位信号需逐级传递。采用串行进位的并行加法器的运算时间较长, 且与位数 n 有关。如果能够同时形成每位的低位进位 $C_1 \sim C_{n-1}$, 就可以同时进行 n 位加法, 这就是采用并行进位的并行加法器。

(2) 并行进位

显然, 串行进位的运算速度慢, 将严重制约整个加法器的速度。为了提高运算速度, 目前的加法器多采用并行进位结构, 即并行地形成各级进位。将式(3.19)以 G 和 P 逐项替代, 可得并行进位的进位逻辑表达式:

$$\left. \begin{aligned} C_1 &= G_1 + P_1 C_0 \\ C_2 &= G_2 + P_2 G_1 + P_2 P_1 C_0 \\ C_3 &= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_0 \\ &\dots \\ C_n &= G_n + P_n G_{n-1} + \dots + P_n P_{n-1} \dots P_1 C_0 \end{aligned} \right\} \quad (3.20)$$

对比式(3.19)和式(3.20)可知: 串行进位各进位信号均依赖于前一级, 而并行进位各进位信号都是独自形成。当加法运算的 A_i 、 B_i 和 C_0 稳定后, 各位可同时形成自己的 G_i 与 P_i , 从而同时形成各自的进位信号 C_i , 这样便大大提高了整体运算速度。

并行进位又叫做同时进位或先行进位(Carry Look Ahead)。但这种先行进位在实现时也存在一定困难, 即高位的进位形成逻辑涉及输入变量过多, 将受到器件扇入系数的限制。因此在位数较多的加法器中, 常采用分级、分组的进位链结构。即将所有参与运算的位分成适当位数的组, 组内并行, 组间或串行或并行。典型的结构是 4 位一组, 进位机构连同 4 位全加器均集成在一块芯片内; 如组间也采用并行, 则可将组间并行进位链集成在专用芯片中; 如加法器位数较长, 还可分级构成并行进位逻辑。

例如, 若加法器字长 16 位, 每 4 位分成一组, 组内并行, 组间并行, 则可将进位链分为两级。第一级为小组内并行进位链, 第二级为组间并行进位链。

① 第一级: 小组内并行进位链

第一小组内的进位逻辑为:

$$\begin{aligned} C_1 &= G_1 + P_1 C_0 \\ C_2 &= G_2 + P_2 G_1 + P_2 P_1 C_0 \\ C_3 &= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_0 \\ C_4 &= G_4 + P_4 G_3 + P_4 P_3 G_2 + P_4 P_3 P_2 G_1 + P_4 P_3 P_2 P_1 C_0 = G_I + P_I C_0 \end{aligned} \quad (3.21)$$

其中, $G_I = G_4 + P_4 G_3 + P_4 P_3 G_2 + P_4 P_3 P_2 G_1$, 是第一小组的进位产生函数; $P_I = P_4 P_3 P_2 P_1$, 是第一小组的进位传递函数。

第二小组内的进位逻辑为:

$$\begin{aligned} C_5 &= G_5 + P_5 C_I \\ C_6 &= G_6 + P_6 G_5 + P_6 P_5 C_I \end{aligned}$$

$$C_7 = G_7 + P_7G_6 + P_7P_6G_5 + P_7P_6P_5C_I \quad (3.22)$$

$$C_8 = G_8 + P_8G_7 + P_8P_7G_6 + P_8P_7P_6G_5 + P_8P_7P_6P_5C_I = G_H + P_HC_I$$

这里 C_I 就是第一组产生的组间进位 C_4 。其余各组内的进位逻辑可依此类推。

② 第二级：小组间并行进位链

小组间的并行进位链表达式为：

$$C_I = G_I + P_I C_0$$

$$C_H = G_H + P_HG_I + P_HP_IC_0$$

$$C_{III} = G_{III} + P_{III}G_H + P_{III}P_HG_I + P_{III}P_HP_IC_0 \quad (3.23)$$

$$C_{IV} = G_{IV} + P_{IV}G_{III} + P_{IV}P_{III}G_H + P_{IV}P_{III}P_HG_I + P_{IV}P_{III}P_HP_IC_0$$

不难看出，两级并行进位链的逻辑形态完全相同。

3.2.3 多功能算术逻辑运算单元 ALU

将若干位全加器、并行进位链及输入选择门集成于一块芯片上，即构成多功能算术逻辑运算单元 ALU。ALU 因可产生多种输出函数，故又可称之为通用函数发生器。例如 SN74181 就是将 4 位全加器与组内并行进位链集成于一块芯片，构成 4 位片，即一片能实现 4 位运算。此外，还有 8 位片、16 位片的 ALU 器件。SN74182 则是组间并行进位链的专用芯片。用数片 SN74181 和 SN74182 一起可构成多位的 ALU 部件。

1. 一位 ALU 单元

一位 ALU 基本单元逻辑如图 3.4 所示，它可分成三部分：

(1) 全加器

由两个半加器构成一位全加器。

(2) 选择控制门

用来对算术运算和逻辑运算进行选择控制，当 $M = 0$ 时，开门接收低位来的进位信号 C_{i-1} ，执行算术运算；当 $M = 1$ 时，关门不接收 C_{i-1} ，执行与进位无关的逻辑运算。

(3) 输入选择逻辑

由与或非门构成的选择逻辑通过 4 个控制信号 S_3, S_2, S_1 和 S_0 对输入数据 A_i, B_i 进行不同的逻辑组合。巧妙的是，输入选择逻辑根据构造并行进位链的需要，让 X_i 输出中包含进位传递函数 $P_i =$

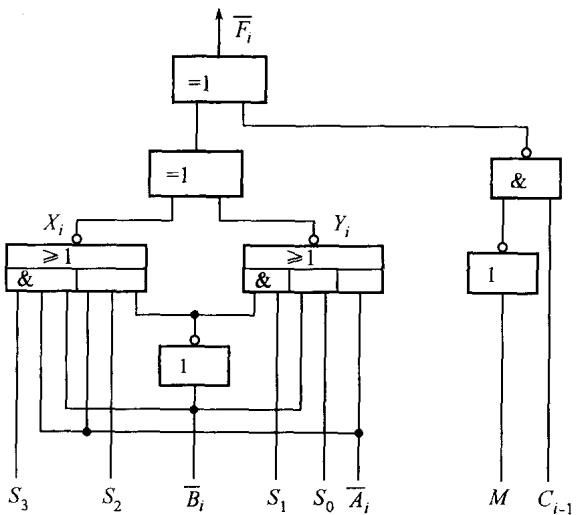


图 3.4 一位 ALU 单元

$A_i + B_i$, Y_i 输出中包含进位产生函数 $G_i = A_i B_i$ 。 S_3 与 S_2 控制选择左边的一个与或非门的输出 X_i , S_1 与 S_0 则控制选择右边一个与或非门的输出 Y_i 。如表 3.3 所示, 只要选择不同的控制信号 $S_3 S_2 S_1 S_0$, 就可获得不同的输出 F_i , 从而实现不同的运算功能。

表 3.3 S_i 与 X_i 、 Y_i 的逻辑关系

S_3	S_2	X_i	S_1	S_0	Y_i
0	0	1	0	0	A_i
0	1	$A_i + \bar{B}_i$	0	1	$A_i B_i$
1	0	$A_i + B_i$	1	0	$A_i \bar{B}_i$
1	1	A_i	1	1	0

2. 4 位片 SN74181

SN74181 可以使用两种极性, 对应的引脚分布如图 3.5 所示; 两种极性关系的比较列于表 3.4 中; 其逻辑构成如图 3.6 所示。

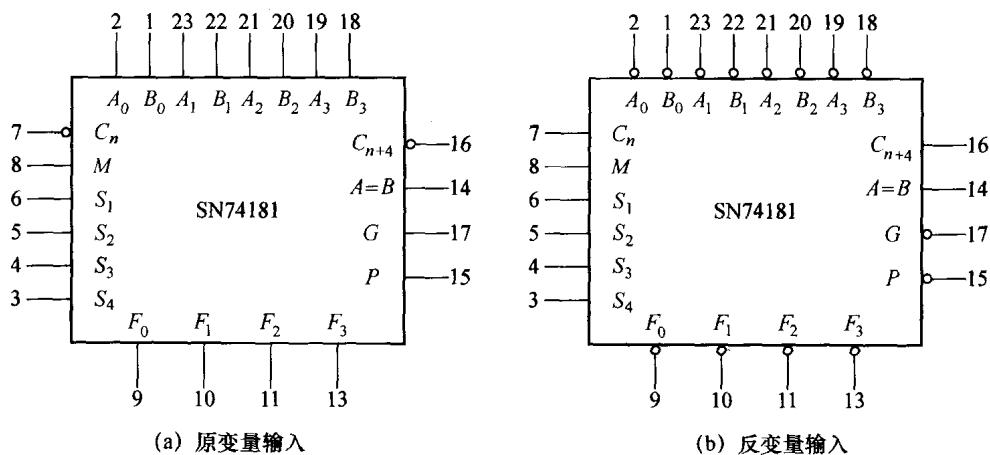


图 3.5 SN74181 引脚分布

表 3.4 SN74181 两种使用方式

	输入原变量、输出原变量	输入反变量、输出反变量
输入	$A_3 \sim A_0, B_3 \sim B_0, \bar{C}_n$	$\bar{A}_3 \sim \bar{A}_0, \bar{B}_3 \sim \bar{B}_0, C_n$
输出	$F_3 \sim F_0, C_{n+4}, G, P, A = B$	$F_3 \sim F_0, C_{n+4}, G, P, A = B$
控制信号	M, S_3, S_2, S_1, S_0	M, S_3, S_2, S_1, S_0

(1) 4 位 ALU

4 位全加器位于图 3.6 的上半部, 每一位的逻辑构成均与一位 ALU 单元相同。但 4 位

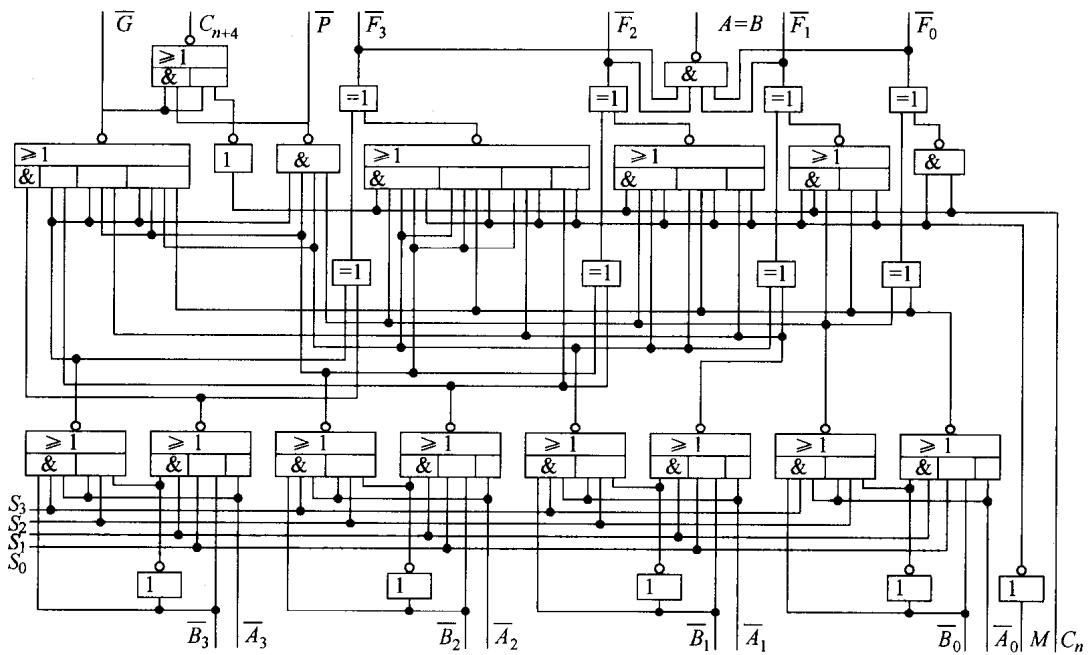


图 3.6 SN74181 内部逻辑构成

共用一个控制门,位于图的右下角由 M 控制选择算术运算或逻辑运算。4 位的输入选择门均位于图的下半部,选择控制输入数据 A_i, B_i ,以实现不同的运算功能。

(2) 组内并行进位链

片内 4 位全加器为一组,构成并行进位结构。初始进位输入为 C_n , 小组进位信号为 C_{n+4} , \bar{G} 和 \bar{P} 分别为小组进位产生和进位传递函数。利用 C_{n+4} 可构成组间串行进位; 利用 \bar{G} 和 \bar{P} 送往 SN74182 可构成组间并行进位。逻辑表达式为:

$$\begin{aligned}
 C_{n+1} &= G_0 + P_0 C_n \\
 C_{n+2} &= G_1 + P_1 G_0 + P_1 P_0 C_n \\
 C_{n+3} &= G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_n \\
 C_{n+4} &= G + P C_n = \overline{\bar{G} \cdot \bar{P} + \bar{G} \cdot C_n}
 \end{aligned} \tag{3.24}$$

其中, $G = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0$, $P = P_3 P_2 P_1 P_0$ 。

(3) 符合比较“ $A = B$ ”

工作方式选择 SN74181 可执行异或运算,输出 $A \oplus B$ 或 $\overline{A \oplus B}$,通过位于图上端的输入门“ $A = B$ ”可获得比较结果。

SN74181 可能实现的算术运算与逻辑运算功能如表 3.5 所示。一旦确定 ALU 所需执行的运算功能,即可根据该表选择控制信号的相应组合。表中的运算符“+”是逻辑加(或),而算术加则用“加”表示。表中的算术运算采用补码数据表示。

由于输入变量较多(除 A_i, B_i, C_n 外, 还有 $M, S_3 \sim S_0$) , SN74181 在逻辑设计时为求简捷, 不采用通常的真值表→卡诺图→表达式→逻辑图方法。SN74181 首先选取恰当的输入选择组合即 X_i, Y_i 的逻辑关系, 再导出可能的 16 种算术运算与 16 种逻辑运算功能。表中有些功能无多大实用价值, 但包含了通常需要的基本运算功能。例如 A 加 B 、 A 减 B (利用 A 加 \bar{B})、 A 加 1 ($M = 0, S_3 \sim S_0 = 1111, C_n = 1$)、 A 减 1、逻辑与 $A B$ 、逻辑或 $A + B$ 、求反 \bar{A} 或 \bar{B} 、异或 $A \oplus B$ 、传送 A (即输出 $A, M = 1, S_3 \sim S_0 = 1111$)、传送 B ($M = 1, S_3 \sim S_0 = 1010$)、输出 0、输出 1 等。

表 3.5 SN74181 的功能

工作方式选择				原变量输入输出		反变量输入输出	
S_3	S_2	S_1	S_0	逻辑运算 ($M = 1$)	算术运算 ($M = 0, C_n = 1$)	逻辑运算 ($M = 1$)	算术运算 ($M = 0, C_n = 0$)
0	0	0	0	\bar{A}	A	\bar{A}	A 减 1
0	0	0	1	$\bar{A} + \bar{B}$	$A + B$	$\bar{A}\bar{B}$	AB 减 1
0	0	1	0	$\bar{A}\bar{B}$	$A + \bar{B}$	$\bar{A} + B$	$A\bar{B}$ 减 1
0	0	1	1	逻辑 0	~ 1	逻辑 1	~ 1
0	1	0	0	$\bar{A}\bar{B}$	A 加 $A\bar{B}$	$\bar{A} + \bar{B}$	A 加 $(A + \bar{B})$
0	1	0	1	\bar{B}	$(A + B)$ 加 $A\bar{B}$	\bar{B}	AB 加 $(A + \bar{B})$
0	1	1	0	$A \oplus B$	A 减 B 减 1	$\bar{A} \oplus \bar{B}$	A 减 B 减 1
0	1	1	1	$\bar{A}\bar{B}$	AB 减 1	$\bar{A} + \bar{B}$	$A + \bar{B}$
1	0	0	0	$\bar{A} + B$	A 加 AB	$\bar{A}\bar{B}$	A 加 $(A + B)$
1	0	0	1	$\bar{A} \oplus \bar{B}$	A 加 B	$A \oplus B$	A 加 B
1	0	1	0	B	$(A + \bar{B})$ 加 AB	B	$A\bar{B}$ 加 $(A + B)$
1	0	1	1	AB	AB 减 1	$A + B$	$A + B$
1	1	0	0	逻辑 1	A 加 A	逻辑 0	A 加 A
1	1	0	1	$A + \bar{B}$	$(A + B)$ 加 A	$\bar{A}\bar{B}$	AB 加 A
1	1	1	0	$A + B$	$(A + \bar{B})$ 加 A	AB	$A\bar{B}$ 加 A
1	1	1	1	A	A 减 1	A	A

3. SN74182

SN74182 是并行进位链芯片, 其对应的引脚分布如图 3.7 所示其逻辑构成如图 3.8 所示。

SN74182 的输入为 $P_0 \sim P_3, G_0 \sim G_3$ 以及 C_n , 输出为:

$$\begin{aligned}
 C_{n+x} &= G_0 + P_0 C_n \\
 C_{n+y} &= G_1 + P_1 G_0 + P_1 P_0 C_n \\
 C_{n+z} &= G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_n \\
 \bar{G} &= \overline{G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0} \\
 \bar{P} &= \overline{P_3 P_2 P_1 P_0}
 \end{aligned} \tag{3.25}$$

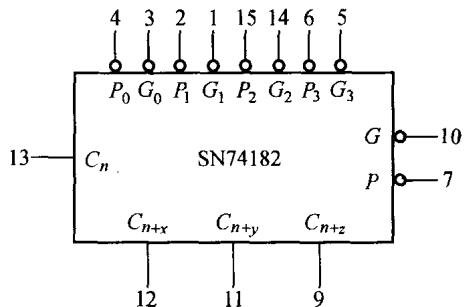


图 3.7 SN74182 引脚分布

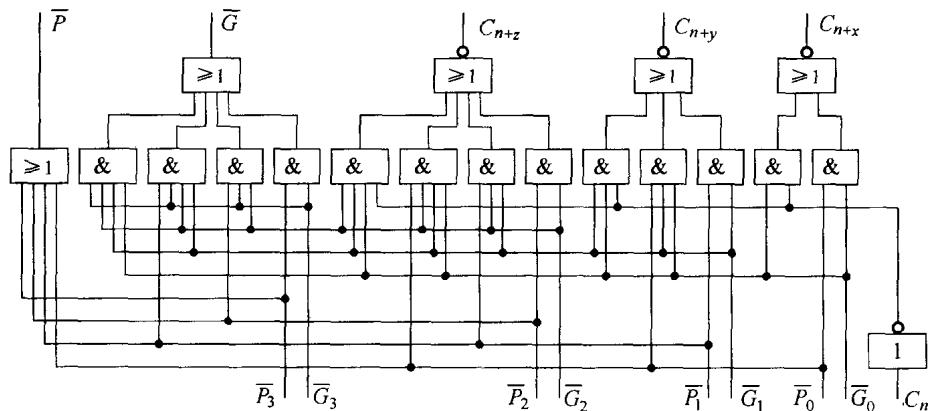


图 3.8 SN74182 内部逻辑构成

对比式(3.23)和式(3.25)不难发现, C_{n+x} 、 C_{n+y} 和 C_{n+z} 与 C_I 、 C_H 和 C_M 的逻辑形态完全相同。这样, SN74182 就可以和多片 SN74181 一起构成多位 ALU 部件。SN74181 一次完成 4 位数据的并行加法, 在组内实现并行进位。SN74182 可以在多个 SN74181 间实现组间并行进位。

4. 多位 ALU 部件

用若干片 SN74181 可方便地构成多位 ALU 部件。由于片内已实现组内并行进位, 若需要采用组间串行进位, 则只需将几片 SN74181 简单地级连, 即将各片的进位输出 C_{n+4} 与高位芯片的进位输入 C_n 相连即可。

若要采用组间并行进位, 则需使用并行进位链芯片 SN74182。一个 16 位组间并行进位的 ALU 的连接如图 3.9 所示。其中, SN74181 输出的小组进位产生函数 \bar{G} 与进位传递函数 \bar{P} , 作为 SN74182 的输入, 而 SN74182 则向各片 SN74181 提供组间进位信号。给定 $\bar{A}_{16} \sim \bar{A}_1$ 、 $\bar{B}_{16} \sim \bar{B}_1$ 和 C_0 后, 4 个 SN74181 首先计算出各自的 \bar{G} 和 \bar{P} , 其中 $\bar{G}_M \sim \bar{G}_1$ 与 $\bar{P}_M \sim \bar{P}_1$ 被分别传递给 SN74182 的输入 $\bar{G}_2 \sim \bar{G}_0$ 与 $\bar{P}_2 \sim \bar{P}_0$; 然后 SN74182 将根据 $\bar{G}_2 \sim \bar{G}_0$ 、 $\bar{P}_2 \sim \bar{P}_0$ 和 C_0 产生 C_{n+z} 、 C_{n+y} 和 C_{n+x} , 对应于组间进位 $C_M \sim C_1$; 最后各个 SN74181 根据各自的 A 、 B 和进位输入 C 得到最终结果 \sum_{16-1} 。

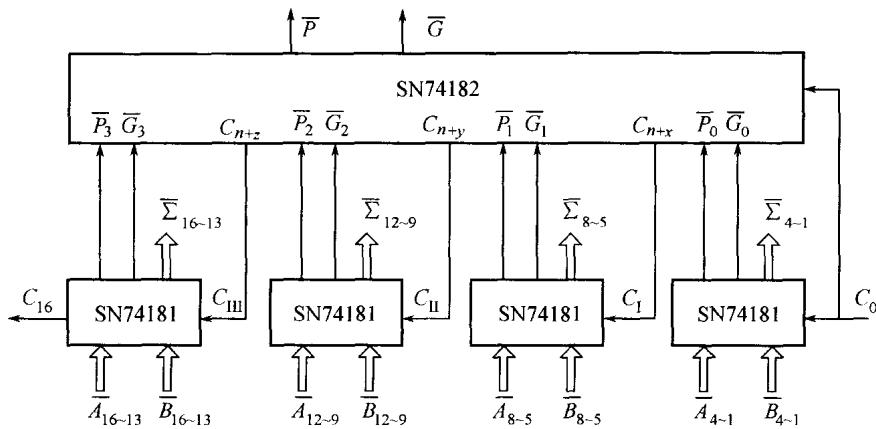


图 3.9 16 位组间并行进位 ALU

SN74182 还可以形成 16 位数据的进位产生函数以及进位传递函数,以便构造更多位数的并行进位链。

3.2.4 十进制加法器

利用多功能算术逻辑部件可完成二进制的加法或减法运算,如果计算机采用十进制数据表示,数据怎样进行运算呢?

处理十进制数通常有两种方法。一种方法是先将十进制数转换为二进制数,然后按二进制运算,最后将结果转换为十进制数。另一种方法是采用二 - 十进制数,直接进行十进制运算,即将每组 4 位先当成二进制数运算,再按十进制运算的进位规律进行校正。第一种方法适用于数据量不太多而计算量较大的情况。第二种是使用较多的方法,它适用于数据量较多但计算较简单的场合。

4 位余 3 码和 8421 码是最常用的二 - 十进制数,下面分别介绍其加法器的构成及工作原理。

1. 余 3 码十进制加法器

两个一位十进制数相加,其和不会超过 18,考虑低位来的进位,其和最大值是 19。两个余 3 码数相加,如无进位,其和应该余 6,所以要得到正确的余 3 码表示的和,应该从和中减去 3 (+1101);如有进位,本位余的 6 已经随着进位进到高位上去了,所以本位和中少 3,应该在和中加上 3 (+0011) 才是本位的余 3 码。

表 3.6 清楚地表明了上述关系。表中的第一栏表示十进制数,第二栏表示与十进制数对应的余 3 码,第三栏表示两个余 3 码按二进制规律相加得到的未加修正的和,第四栏表示对第三栏所得和应采取的修正方法。

表 3.6 余 3 码加法

十进制	余 3 码	未修正的和 $C_i, S'_{i1}S'_{i2}S'_{i3}S'_{i4}$	修正法	十进制	余 3 码	未修正的和 $C_i, S'_{i1}S'_{i2}S'_{i3}S'_{i4}$	修正法
0	0 0 1 1	0, 0 1 1 0		10	1, 0 0 1 1	1, 0 0 0 0	
1	0 1 0 0	0, 0 1 1 1		11	1, 0 1 0 0	1, 0 0 0 1	
2	0 1 0 1	0, 1 0 0 0		12	1, 0 1 0 1	1, 0 0 1 0	
3	0 1 1 0	0, 1 0 0 1	无进位	13	1, 0 1 1 0	1, 0 0 1 1	有进位
4	0 1 1 1	0, 1 0 1 0	$C_i = 0:$	14	1, 0 1 1 1	1, 0 1 0 0	$C_i = 1:$
5	1 0 0 0	0, 1 0 1 1	需(3 修正 (+1101))	15	1, 1 0 0 0	1, 0 1 0 1	需 +3 修正 (+0011)
6	1 0 0 1	0, 1 1 0 0		16	1, 1 0 0 1	1, 0 1 1 0	
7	1 0 1 0	0, 1 1 0 1		17	1, 1 0 1 0	1, 0 1 1 1	
8	1 0 1 1	0, 1 1 1 0		18	1, 1 0 1 1	1, 1 0 0 0	
9	1 1 0 0	0, 1 1 1 1		19	1, 1 1 0 0	1, 1 0 0 1	

从表 3.6 中不难发现,无论有无进位,未修正的和的最低位 S'_{i4} 都要加 1,所以修正最低位的全加器可简化成一个倒相器。图 3.10 给出了余 3 码十进制加法器的逻辑结构。图中下一排 4 位全加器 FA 组成的加法器执行二进制运算产生未修正的和 $S'_{i1}S'_{i2}S'_{i3}S'_{i4}$;上一排 3 个全加器和一个非门组成的倒相器得到修正后的和 $S_{i1}S_{i2}S_{i3}S_{i4}$ 。而采用 +3 或者 -3 修正则通过 C_i 来决定。

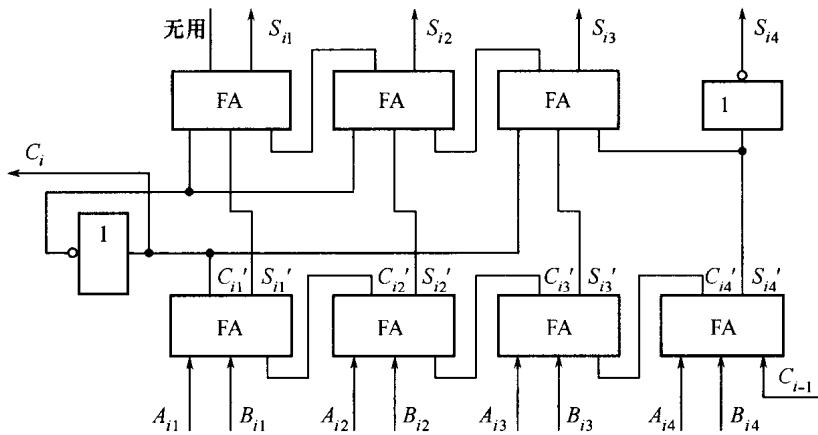


图 3.10 余 3 码表示的十进制加法器

2. 8421 码十进制加法器

两个一位十进制数相加,其和大于等于 10 时就要进位。因此,用 8421 码表示的两个一位十进制数相加,当和大于等于 1010 时,必须加上 6(+0110)形成进位。

表 3.7 给出了 8421 码十进制加法运算和的修正规律。从该表右半部的第三栏可以看

出,若设 $S'_{i1}S'_{i2}S'_{i3}S'_{i4}$ 代表相加后未修正的和, C'_{ii} 代表最高位的进位,则修正条件为:

$$S'_{i1}S'_{i3} + S'_{i1}S'_{i2} + C'_{ii} = C_i$$

可见,修正条件也就是有进位($C_i = 1$)的条件。

表 3.7 8421 码十进制加法运算

十进制	8421 码	未修正的和 $C'_{ii}, S'_{i1}S'_{i2}S'_{i3}S'_{i4}$	修正法	十进制	8421 码	未修正的和 $C'_{ii}, S'_{i1}S'_{i2}S'_{i3}S'_{i4}$	判断位	修正法
0	0000	0,0000	不修正	10	1.0000	0,1010	$S'_{i1}S'_{i3}$	+6 修正 (+0110)
1	0001	0,0001		11	1.0001	0,1011		
2	0010	0,0010		12	1.0010	0,1100		
3	0011	0,0011		13	1.0011	0,1101		
4	0100	0,0100		14	1.0100	0,1110		
5	0101	0,0101		15	1.0101	0,1111		
6	0110	0,0110		16	1.0110	1,0000	C'_{ii}	
7	0111	0,0111		17	1.0111	1,0001		
8	1000	0,1000		18	1.1000	1,0010		
9	1001	0,1001		19	1.1001	1,0011		

8421 码十进制加法器的逻辑结构如图 3.11 所示。它包括一级二进制加法器和一级“加 6 修正”逻辑,左端的与或门即修正条件逻辑。当两数之和大于 9 时, $C_i = 1$,以连线方式提供(0110)与尚未修正的和 $S'_{i1}S'_{i2}S'_{i3}S'_{i4}$ 相加,从而得到修正后的结果 $S_{i1}S_{i2}S_{i3}S_{i4}$ 。当两数之和小于等于 9 时, $C_i = 0$,不需修正,图中表示为加 0(+0000)修正。

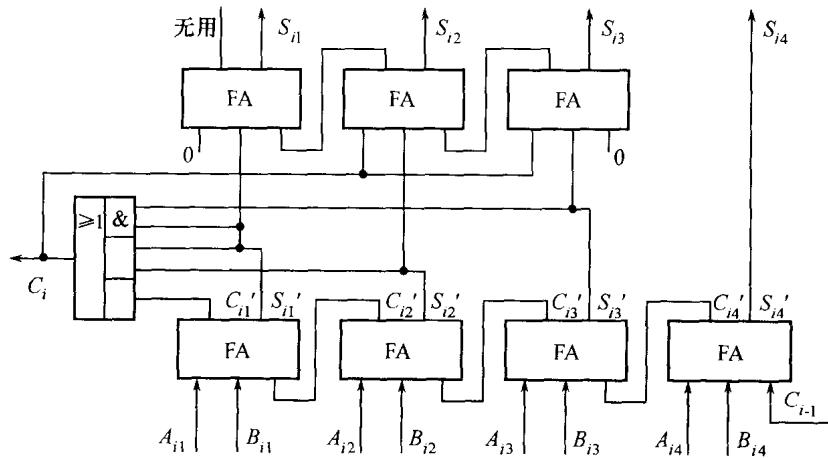


图 3.11 8421 码十进制加法器

3.3 定点乘法运算

乘法和除法都是计算机具有的基本算术运算,但其运算比较复杂,计算过程也较长,其执行速度对计算机的运算速度影响很大。早期的小型计算机和微型计算机的乘法或除法一般通过执行程序来完成。现在几乎所有的计算机都设有专门的乘法和除法指令,即设有专门的乘除运算部件。

乘法运算方法也与码制密切相关。比较而言,乘法采用原码较为简单:乘积的符号为被乘数和乘数符号位的异或,乘积则按两数绝对值相乘即可。原码乘法虽然简单,但由于计算机通常采用补码数据表示方法,若使用原码进行乘法,还需将补码变换成为原码,使得运算过程复杂并增加了处理时间。因此,现在计算机中多直接采用补码进行乘法。

定点数据表示分为定点整数和定点小数两种数据形式,它们的乘法运算过程类似,下面以定点小数为例进行讨论。

本节首先讨论原码和补码一位乘法,然后介绍快速乘法,主要是两位乘法。在介绍算法的同时,讨论乘法器的构成与工作原理。

3.3.1 原码一位乘法

两原码数相乘时,乘积的符号单独对待,即按同号相乘为正,异号相乘为负的方法处理;数值部分为两数绝对值之积。

若参加运算的为两个 $n+1$ 位定点小数,则结果为 $2n+1$ 位。如果使用原码完成 $C = A \times B$ 的计算,记:

被乘数 A 的原码: $[A]_{\text{原}} = A_0.A_1A_2\cdots A_n$

乘数 B 的原码: $[B]_{\text{原}} = B_0.B_1B_2\cdots B_n$

乘积 C 的原码: $[C]_{\text{原}} = C_0.C_1C_2\cdots C_{2n}$

故乘积的符号位为: $C_0 = A_0 \oplus B_0$

乘积的结果为:

$$[C]_{\text{原}} = C_0 + |A| \times |B| = (A_0 \oplus B_0) + (0.A_1A_2\cdots A_n) \times (0.B_1B_2\cdots B_n)$$

式中 $A_0 \oplus B_0$ 的结果被作为数值参加加法运算。由于原码乘法的符号位单独处理,而乘积是取绝对值后两个正数相乘,故为简便起见,通常把下标上的“原”字去掉。

下面举例说明原码乘法采用一般笔算与机器运算的区别。

例 3.9 已知被乘数 $A = 0.1101$, 乘数 $B = 0.1011$, 求 $A \times B = ?$

(1) 一般笔算法

一般笔算方法的过程如算式 3.9 所示。

$$\begin{array}{r}
 & 0. \quad 1 \quad 1 \quad 0 \quad 1 \\
 \times) & 0. \quad 1 \quad 0 \quad 1 \quad 1 \\
 \hline
 M_1 = A \times B_4 & 1 \quad 1 \quad 0 \quad 1 \\
 M_2 = A \times B_3 & 1 \quad 1 \quad 0 \quad 1 \\
 M_3 = A \times B_2 & 0 \quad 0 \quad 0 \quad 0 \\
 +) \quad M_4 = A \times B_1 & 1 \quad 1 \quad 0 \quad 1 \\
 \hline
 A \times B = \sum M_i & 0. \quad 1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1
 \end{array}$$

算式 3.9

需要注意的是,这是从乘数的低位开始乘得到第 1 个位积 M_1 (即 $M_1 = A \times B_4$),也可以从乘数的最高位开始乘,得到第 1 个位积 M_1 (即 $M_1 = A \times B_1$)。这种笔算方法的过程如算式 3.10 所示。

$$\begin{array}{r}
 & 0. \quad 1 \quad 1 \quad 0 \quad 1 \\
 \times) & 0. \quad 1 \quad 0 \quad 1 \quad 1 \\
 \hline
 M_1 = A \times B_1 & 1 \quad 1 \quad 0 \quad 1 \\
 M_2 = A \times B_2 & 0 \quad 0 \quad 0 \quad 0 \\
 M_3 = A \times B_3 & 1 \quad 1 \quad 0 \quad 1 \\
 +) \quad M_4 = A \times B_4 & 1 \quad 1 \quad 0 \quad 1 \\
 \hline
 A \times B = \sum M_i & 0. \quad 1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1
 \end{array}$$

算式 3.10

在算式 3.9 和算式 3.10 中,称每一行的乘积 $M_i = A \times B_i$ 为位积。两个 $n+1$ 位(包含 1 位符号位)定点小数相乘,将产生 n 个位积,乘积为 $2n+1$ 位(包含 1 位符号位)。如果将这 n 个位积直接相加,需要 $2n+1$ 位加法器,但由于每个位积中有 $n+1$ 位为 0,位积相加时不可能全部利用 $2n+1$ 位全加器操作。因此,这种方法不适用于机器运算。

观察算式 3.9 中相邻的位积 M_1 与 M_2 ,在做加法时需要将 M_1 右移一位对齐。这样,如果使用 $n+1$ 位加法器,可以将 M_1 右移一位再与 M_2 相加;得到 M_1 与 M_2 之和后,可以将和右移一位再与 M_3 相加得到三个位积之和……。这样,通过多次右移就可以使用 $n+1$ 位加法器完成这 n 个位积的加法。同样地,对于算式 3.10,使用 $n+1$ 位加法器,通过多次左移也可以完成这 n 个位积的加法。不失一般性,本书在以后对乘法的讨论中,均通过右移来完成位积的加法。

(2) 机器运算方法

乘法实际上就是多次加法的结果(累加)。为了适用于一般加法器完成两数相加的特点,可将 $A \times B$ 改写成如下形式:

$$A \times B = A \times (0.1011) = 0.1 \times A + 0.00 \times A + 0.001 \times A + 0.0001 \times A$$

$$\begin{aligned}
 &= 2^{-1} \{ A + 2^{-1} [0 + 2^{-1} (A + 2^{-1} (A + 0))] \} \\
 &\quad \underbrace{\qquad\qquad\qquad}_{P_0} \\
 &\quad \underbrace{\qquad\qquad\qquad}_{P_1} \\
 &\quad \underbrace{\qquad\qquad\qquad}_{P_2} \\
 &\quad \underbrace{\qquad\qquad\qquad}_{P_3} \\
 &\quad P_4
 \end{aligned}$$

所以 $A \times B = P_4$, 且有:

$$\begin{aligned}
 P_1 &= 2^{-1}(A + 0) \\
 P_2 &= 2^{-1}(A + P_1) \\
 P_3 &= 2^{-1}(0 + P_2) \\
 P_4 &= 2^{-1}(A + P_3)
 \end{aligned}$$

这就是说,机器运算乘法的数值部分是通过多次累加完成的,每累加一次后结果右移一位($\times 2^{-1}$),这样所得的结果称为部分积 P_i 。最后一次部分积即为乘积的数值部分,而把乘法开始时的 $P_0 = 0$ 称为初始部分积。因此,对于例 3.9,可通过 4 次加法,每次加法后右移一位来完成,如算式 3.11 所示。

	部 分 积	乘 数
P_0	0.0000	1 0 1 1
$+A)$	0.1101	
	<hr/>	
	0.1101	
$P_1 \rightarrow 1$	0.0110	1 1 0 1
$+A)$	0.1101	
	<hr/>	
	1.0011	
$P_2 \rightarrow 1$	0.1001	1 1 1 0
$P_3 \rightarrow 1$	0.0100	1 1 1 1
$+A)$	0.1101	
	<hr/>	
	1.0001	
$P_4 \rightarrow 1$	0.1000	1 1 1 1

算式 3.11

因此,乘法的结果为 $C = (0 \oplus 0) + 0.10001111 = 0.10001111$ 。

根据例 3.9 推而广之,可以得到一般原码乘法的递推公式:

$$P_0 = 0$$

$$\begin{aligned}
 P_1 &= 2^{-1}(P_0 + B_n \times |A|) \\
 P_2 &= 2^{-1}(P_1 + B_{n-1} \times |A|) \\
 &\dots \\
 P_i &= 2^{-1}(P_{i-1} + B_{n-i+1} \times |A|) \\
 &\dots \\
 P_n &= 2^{-1}(P_{n-1} + B_1 \times |A|) = |A| \times |B|
 \end{aligned}$$

于是,原码一位乘法的规则可简单表示为:

$$C = A \times B = (A_0 \oplus B_0) + P_n$$

为实现原码一位乘法,乘法器需要由被乘数寄存器、乘数寄存器、乘积寄存器、加法器及其输入选择开关和控制电路等组成,如图 3.12 所示。图中实线代表数据传输线,虚线代表控制信号。

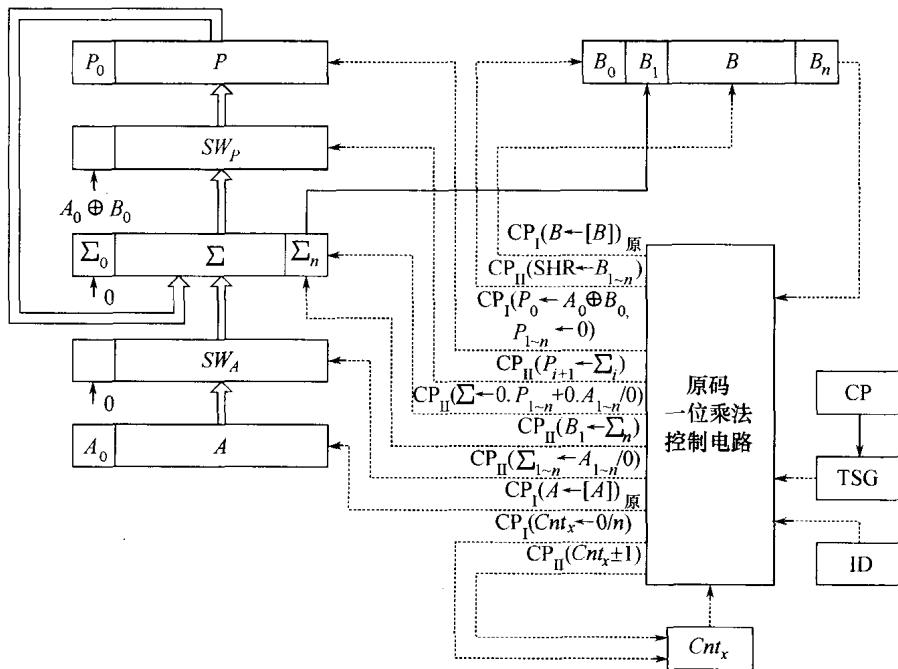


图 3.12 原码一位乘法器逻辑结构

图 3.12 中,当指令译码器 ID 发现是一条乘法指令时,由原码一位乘法控制电路完成乘法。原码一位乘法器各部分的功能如下:

P 寄存器:乘积寄存器, $n + 1$ 位。初态为 0, 乘法完成时 P_0 存放积的符号, $P_1 \sim P_n$ 存放积的高 n 位。

SW_P : P 的输入开关, $n + 1$ 位。第一位将符号位运算的结果 $A_0 \oplus B_0$ 送往 P_0 (注意, 该操作在整个乘法过程中仅做一次); 后 n 位完成传送操作 $P_{i+1} \leftarrow \Sigma_i$ 。

Σ :加法器, $n+1$ 位。被加数: $\Sigma_0 \leftarrow 0, \Sigma_{1-n} \leftarrow P_{1-n}$; 加数: $\Sigma_0 \leftarrow 0, \Sigma_{1-n} \leftarrow A_{1-n}$ 或 $\Sigma_{1-n} \leftarrow 0$ 。

A 寄存器: 被乘数寄存器, $n+1$ 位。存放被乘数 A 的原码, 乘法过程中保持不变。

SW_A : Σ 的加数输入开关, $n+1$ 位。完成 $\Sigma_{1-n} \leftarrow A_{1-n}$ 或 $\Sigma_{1-n} \leftarrow 0$ 的转换, 由 B_n 进行控制。 $SW_{A0} \leftarrow 0$, 表示 A 的符号位单独处理, 不参加运算。

B 寄存器: 乘数寄存器, $n+1$ 位。初值为乘数 B 的原码, 乘法完成时 B_{1-n} 为乘积的低 n 位。它具有右移一位的功能, 右移时 $B_1 \leftarrow \Sigma_n$ (B_0 不参加移位)。

Cnt_x : 乘法计数器。初值为 0 或者 n , 每步乘法后加 1 或减 1, 以控制乘法运算步骤。

需要注意的是, 所有寄存器均采用 D 触发器, 送和数与右移均可在一个时钟周期内同时完成。

当然, 乘法步骤是有一定的先后次序的, 各个操作的执行时机由时钟 CP 和节拍脉冲发生器 TSG 控制。TSG 将产生一系列的 CP_i , 与乘法控制电路配合控制各操作的执行顺序。图 3.12 中, TSG 产生两种节拍 CP_I 和 CP_{II} , 用以表示执行时间的先后次序。受 CP_{II} 控制的操作将在受 CP_I 控制的操作完成之后才能进行。而对于 CP_{II} , 在乘法的每一步中, CP_{II} 代表的节拍脉冲也是不同的。关于对指令的译码, 节拍脉冲的形成将在第四章中讨论。

原码一位乘法的执行过程如下:

(1) 乘法开始时, 各寄存器被置初值, 这些操作受 CP_I 控制。

- $CP_I (A \leftarrow [A]_{原})$ 表示当 CP_I 有效且 “ $A \leftarrow [A]_{原}$ ” 控制信号有效时, 被乘数寄存器 A 的内容被赋值为 $[A]_{原}$ 的值。
- $CP_I (B \leftarrow [B]_{原})$ 控制将乘数寄存器 B 的内容赋值为 $[B]_{原}$ 。
- $CP_I (P_0 \leftarrow A_0 \oplus B_0, P_{1-n} \leftarrow 0)$ 控制形成乘积的符号 P_0 , 并将部分积寄存器 P_{1-n} 的初值赋值为 0。
- $CP_I (Cnt_x \leftarrow 0/n)$ 控制将乘法计数器 Cnt_x 的初值赋值为 0 或者 n 。

(2) 进行一步乘法, 这些操作受 CP_{II} 控制。

- $CP_{II} (\Sigma_{1-n} \leftarrow A_{1-n}/0)$ 表示当 CP_{II} 有效时根据 B_n 控制加法器 Σ 的加数。当 B_n 为 0 时, Σ 的加数为 0, 即 $\Sigma_{1-n} \leftarrow 0$; 当 B_n 为 1 时, Σ 的加数为 A_{1-n} , 即 $\Sigma_{1-n} \leftarrow A_{1-n}$ 。

• $CP_{II} (\Sigma \leftarrow 0, P_{1-n} + 0, A_{1-n}/0)$ 控制加法器 Σ 完成加法。被加数: $\Sigma_0 \leftarrow 0, \Sigma_{1-n} \leftarrow P_{1-n}$; 加数: $\Sigma_0 \leftarrow 0, \Sigma_{1-n} \leftarrow A_{1-n}$ 或 $\Sigma_{1-n} \leftarrow 0$ 。

- $CP_{II} (P_{i+1} \leftarrow \Sigma_i)$ 控制加法的结果右移 1 位送部分积寄存器 P 。
- $CP_{II} (SHR B_{1-n})$ 控制乘数寄存器 B 除符号位外右移 1 位。
- $CP_{II} (B_1 \leftarrow \Sigma_n)$ 控制加法器右移时将丢失的一位被保存至 B_1 。
- $CP_{II} (Cnt_x \pm 1)$ 控制乘法计数器计数。

若不满足乘法结束条件时再次进行第(2)步乘法, 否则转第(3)步。

(3) 当满足结束条件(乘法计数器 $Cnt_x = n$ 或者 0)时, 乘法结束。

从乘法器的工作过程可见, 原码一位乘法需要 $n+1$ 步(1 个 CP_I 和 n 个 CP_{II})完成。需

要注意的是,受 CP_I(或 CP_{II})控制的操作可以安排在一个时钟周期内完成,也可以安排在不同的时钟周期内。如果这些操作安排在一个时钟周期内,原码一位乘法需要 $n+1$ 个时钟周期完成;如果这些操作安排在多个时钟周期内,这 $n+1$ 步乘法就需要多于 $n+1$ 个时钟周期来完成。

3.3.2 补码一位乘法

补码一位乘法是在原码一位乘法的基础上发展而来的,目前使用最广泛的是补码比较乘法。它是由英国的 Booth 夫妇首先提出的,故又称为 Booth 乘法。

若参加运算的为两个 $n+1$ 位定点小数,则结果为 $2n+1$ 位。如果使用补码完成 $C = A \times B$ 的计算,记:

$$\text{被乘数 } A \text{ 的补码: } [A]_{\text{补}} = A_{01}A_{02} \cdot A_1A_2 \cdots A_n$$

$$\text{乘数 } B \text{ 的补码: } [B]_{\text{补}} = B_0 \cdot B_1B_2 \cdots B_n$$

$$\text{乘积 } C \text{ 的补码: } [C]_{\text{补}} = C_0 \cdot C_1C_2 \cdots C_{2n}$$

与原码乘法不同的是,补码乘法中乘数和被乘数的符号位需要参加运算。在乘法过程中,被乘数 A 需要采用两位符号位的变形补码,即使用模 4 补码, A_{01} 和 A_{02} 均为 $[A]_{\text{补}}$ 的符号位。由于 $A = -1$ 时, A 的补码是具有意义的, $[A]_{\text{补}} = 11.00 \cdots 0$, 而 $[-A]_{\text{补}} = 01.00 \cdots 0$ 。然而,如果使用一位符号位的模 2 补码, $[-A]_{\text{补}}$ 将无法表示。同样,为保证在进行加法时部分积的符号位不被破坏,部分积也要采用两位符号位。

首先,分乘数非负 ($B \geq 0$) 和乘数为负 ($B < 0$) 两种情况来讨论。

(1) A 符号任意, B 为非负数 ($B_0 = 0$, $[B]_{\text{补}} = B$)

① 当被乘数 $A \geq 0$ 时, $[A]_{\text{补}} = [A]_{\text{原}}$, $[B]_{\text{补}} = [B]_{\text{原}}$, 乘法过程与原码乘法相同,但补码的符号位参加运算。

② 当被乘数 $A < 0$ 时,需要使用模 4 补码。根据补码的定义有:

$$[A]_{\text{补}} = 2^2 + A = 2^{n+2} + A \pmod{4}$$

此时, $[B]_{\text{补}} = 0 \cdot B_1B_2 \cdots B_n = \sum_{i=1}^n B_i 2^{-i}$ 。于是:

$$\begin{aligned} [A]_{\text{补}} \times [B]_{\text{补}} &= (2^{n+2} + A) \times B \\ &= 2^{n+2} \times \sum_{i=1}^n B_i \times 2^{-i} + A \times B \\ &= 2^2 \times \sum_{i=1}^n B_i \times 2^{n-i} + A \times B \\ &= 2^2 + A \times B \pmod{4} \\ &= [A \times B]_{\text{补}} \end{aligned} \tag{3.26}$$

上式中,因为 $\sum_{i=1}^n B_i \times 2^{n-i}$ 是个非负整数,故有:

$$2^2 \times \sum_{i=1}^n B_i \times 2^{n-i} + A \times B = 2^2 + A \times B \pmod{4}$$

综合①和②,当A符号任意,B为正数时,有:

$$[A \times B]_{\text{补}} = [A]_{\text{补}} \times [B]_{\text{补}}$$

此时,补码乘法运算的递推公式为:

$$[P_0]_{\text{补}} \equiv 0$$

$$[P_1]_{\text{补}} = 2^{-1}([P_0]_{\text{补}} + B_n \times [A]_{\text{补}})$$

$$[P_2]_{\text{补}} = 2^{-1}([P_1]_{\text{补}} + B_{n-1} \times [A]_{\text{补}})$$

.....

$$[P_i]_{\text{补}} = 2^{-1}([P_{i-1}]_{\text{补}} + B_{n-i+1} \times [A]_{\text{补}})$$

.....

$$[P_n]_{\text{补}} = 2^{-1}([P_{n-1}]_{\text{补}} + B_1 \times [A]_{\text{补}}) \quad (3.28)$$

这与原码乘法的递推公式类似,只是相加时A的符号位按补码规则参加运算,移位时按补码规则移位即可。

(2) A符号任意,B为负数($B_0 = 1$)

当 $B < 0$ 时,根据补码的定义有:

$$[B]_{\text{补}} = 2 + B \pmod{2}$$

此时, $[B]_{\text{补}} = 1.B_1B_2\dots B_n$,则 $B = 0.B_1B_2\dots B_n - 1$ 。注意,式中的 B_i 已经是 $[B]_{\text{补}}$ 第*i*位的值。故

$$\begin{aligned} [A \times B]_{\text{补}} &= [A \times (0.B_1B_2\dots B_n) - A]_{\text{补}} \\ &= [A \times (0.B_1B_2\dots B_n)]_{\text{补}} + [-A]_{\text{补}} \\ &= [A]_{\text{补}} \times (0.B_1B_2\dots B_n) + [-A]_{\text{补}} \end{aligned} \quad (3.29)$$

于是,可写出乘数为负时乘法运算的递推公式:

$$[P_0]_{\text{补}} \equiv 0$$

$$[P_1]_{\text{补}} = 2^{-1}([P_0]_{\text{补}} + B_n \times [A]_{\text{补}})$$

$$[P_2]_{\text{补}} = 2^{-1}([P_1]_{\text{补}} + B_{n-1} \times [A]_{\text{补}})$$

.....

$$[P_i]_{\text{补}} = 2^{-1}([P_{i-1}]_{\text{补}} + B_{n-i+1} \times [A]_{\text{补}})$$

.....

$$[P_n]_{\text{补}} = 2^{-1}([P_{n-1}]_{\text{补}} + B_1 \times [A]_{\text{补}})$$

$$[A \times B]_{\text{补}} = [P_n]_{\text{补}} + [-A]_{\text{补}}$$

与B为正时比较,B为负数时,求得 $[P_n]_{\text{补}}$ 后需再作一次加法,即加上一个校正量 $[-A]_{\text{补}}$ 。

将上述(1)、(2)两种情况综合起来,可得补码一位乘法的统一公式:

$$[C]_{\text{补}} = [A \times B]_{\text{补}} = [A]_{\text{补}} \times (0.B_1 B_2 \cdots B_n) + [-A]_{\text{补}} \times B_0$$

改写一下统一公式：

$$\begin{aligned}[C]_{\text{补}} &= [A]_{\text{补}} \times (B_1 \times 2^{-1} + B_2 \times 2^{-2} + \cdots + B_n \times 2^{-n}) - [A]_{\text{补}} \times B_0 \\&= [A]_{\text{补}} \times [-B_0 + (B_1 - B_1 \times 2^{-1}) + (B_2 \times 2^{-1} - B_2 \times 2^{-2}) + \cdots + \\&\quad (B_n \times 2^{-(n-1)} - B_n \times 2^{-n})] \\&= [A]_{\text{补}} ([B_1 - B_0] \times 2^0 + [B_2 - B_1] \times 2^{-1} + \cdots + [B_{n+1} - B_n] \times 2^{-n}]\end{aligned}$$

式中, $B_{n+1} \equiv 0$ 。将上式改写成：

$$\begin{aligned}[C]_{\text{补}} &= [A]_{\text{补}} (B_1 - B_0) + 2^{-1} \{ [A]_{\text{补}} (B_2 - B_1) + \\&\quad 2^{-1} \{ \cdots + 2^{-1} \{ [A]_{\text{补}} (B_{n+1} - B_n) + 0 \} \cdots \}\}\end{aligned}$$

其递推公式可写出：

$$\begin{aligned}[P_0]_{\text{补}} &\equiv 0 \\[P_1]_{\text{补}} &= 2^{-1} \{ [P_0]_{\text{补}} + (B_{n+1} - B_n) \times [A]_{\text{补}} \} \\[P_2]_{\text{补}} &= 2^{-1} \{ [P_1]_{\text{补}} + (B_n - B_{n-1}) \times [A]_{\text{补}} \} \\&\dots \\[P_i]_{\text{补}} &= 2^{-1} \{ [P_{i-1}]_{\text{补}} + (B_{n-i+2} - B_{n-i+1}) \times [A]_{\text{补}} \} \\&\dots \\[P_n]_{\text{补}} &= 2^{-1} \{ [P_{n-1}]_{\text{补}} + (B_2 - B_1) \times [A]_{\text{补}} \}\end{aligned}$$

所以

$$[C]_{\text{补}} = [P_{n+1}]_{\text{补}} = [P_n]_{\text{补}} + (B_1 - B_0) \times [A]_{\text{补}}$$

由递推公式可以发现,新的部分积 $[P_i]_{\text{补}}$ 的计算决定于两个比较位 B_{n-i+2} 和 B_{n-i+1} 的数值。与原码乘法类似,补码乘法中乘数寄存器在每计算出一个新的部分积后都要右移一位。这样,判断位 B_{n-i+2} 和 B_{n-i+1} 就是每次移位后乘数寄存器的最后两位 B_n 和 B_{n+1} 。根据递推公式,可得表3.8所示的乘法规则。这种根据乘数的相邻两位比较结果而决定运算的方法称之为比较法,也称为Booth乘法。

表3.8 Booth乘法表

判断位 B_n B_{n+1}	新部分积	操作	说 明
0 0	$[P_{i+1}]_{\text{补}} = 2^{-1} [P_i]_{\text{补}}$	$\rightarrow 1$	上次部分积右移一位
0 1	$[P_{i+1}]_{\text{补}} = 2^{-1} \{ [P_i]_{\text{补}} + [A]_{\text{补}} \}$	$+ [A]_{\text{补}}, \rightarrow 1$	上次部分积加 $[A]_{\text{补}}$ 后再右移一位
1 0	$[P_{i+1}]_{\text{补}} = 2^{-1} \{ [P_i]_{\text{补}} + [-A]_{\text{补}} \}$	$+ [-A]_{\text{补}}, \rightarrow 1$	上次部分积加 $[-A]_{\text{补}}$ 后再右移一位
1 1	$[P_{i+1}]_{\text{补}} = 2^{-1} [P_i]_{\text{补}}$	$\rightarrow 1$	上次部分积右移一位

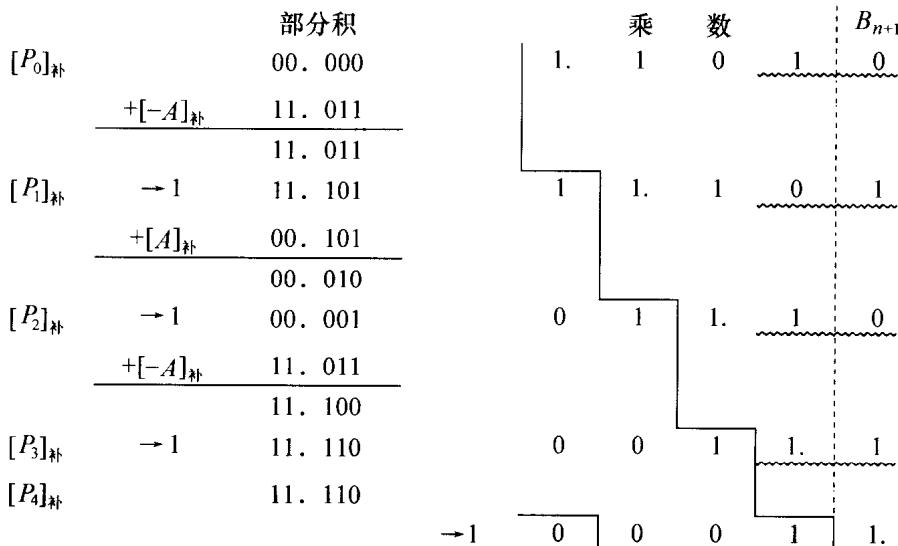
于是,补码一位比较乘法即Booth乘法可归纳为如下规则:

- (1) 被乘数和部分积取两位符号位, 乘数取一位符号位, 并参与运算;
- (2) 乘数末尾增设附加位 B_{n+1} , 其初始值为 0;
- (3) B_n 和 B_{n+1} 构成各步运算的乘数判断位, 按表 3.8 所示方法进行操作;
- (4) 按补码移位规则, 部分积右移时末端触发器补入第一位符号位;
- (5) 按表 3.8 算法进行到第 $n+1$ 步, 但第 $n+1$ 步的部分积不再移位。

值得注意的是, 由于乘数 $[B]_b$ 只使用一位符号位, Booth 乘法只需要作 $n+1$ 步乘法, 并不因为部分积使用了两位符号位而需要作 $n+2$ 步乘法。

例 3.10 已知 $A = 0.101, B = -0.011$, 求 $[A \times B]_b = ?$

解: $[A]_b = 00.101, [B]_b = 1.101, [-A]_b = 11.011$, 运算过程如算式 3.12 所示。



算式 3.12

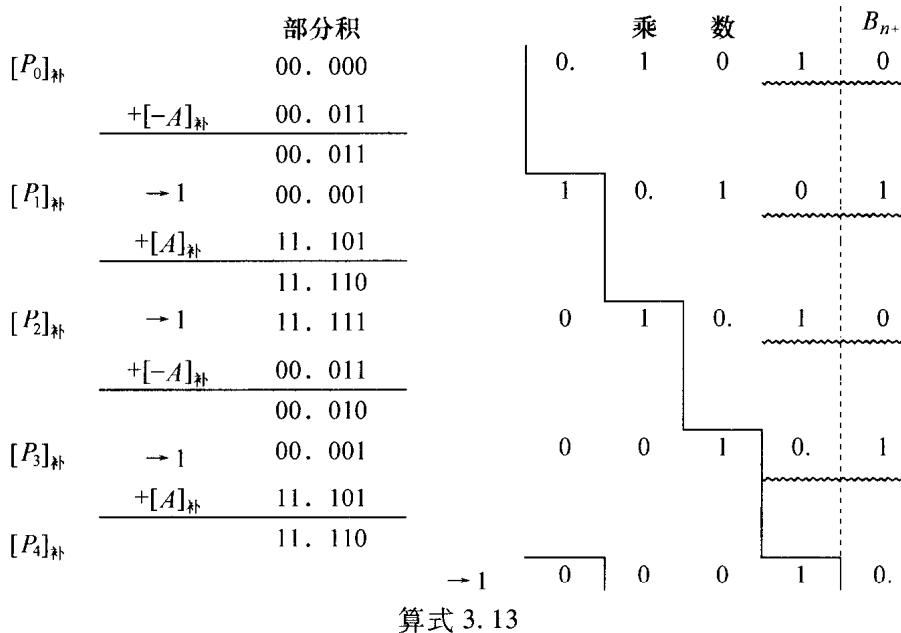
$$\text{故 } [C]_b = [A \times B]_b = 1.110001$$

从例 3.10 可以看出, Booth 乘法最后一步部分积不移位。此时, 乘数寄存器 B 中已经形成了正确的乘积低 n 位 ($B_0 \sim B_{n-1}$), 为什么还要右移一位呢?

在运算器的实现中, 乘法中乘数寄存器 B 与除法中商寄存器 C 使用的是一个寄存器, 故又被称为乘商寄存器 MQ, MQ_0 代表符号位。为了使硬件电路简单, 乘法直接从 $MQ_1 \sim MQ_n$ 中获得部分积的低位, 除法将直接从 $MQ_1 \sim MQ_n$ 中获得商的尾数部分。因此, 乘数寄存器 B 右移一位的目的是使乘积的低位正确存放到乘数寄存器的尾数中。乘法结束时, 寄存器的符号位无用, $B_1 \sim B_n$ 为乘积的低 n 位, 乘积的符号为部分积 $[P_n]_b$ 的符号。

例 3.11 已知 $A = -0.011, B = 0.101$, 求 $[A \times B]_b = ?$

解: $[A]_b = 11.101, [B]_b = 0.101, [-A]_b = 00.011$ 。运算过程如算式 3.13 所示。



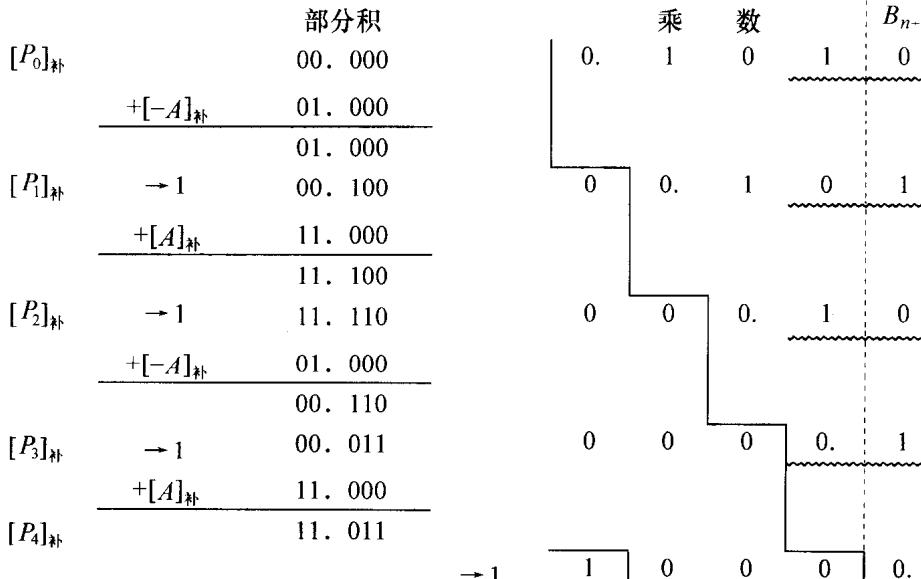
算式 3.13

故 $[C]_{\text{补}} = [A \times B]_{\text{补}} = 1.110001$ 。

例 3.10 和例 3.11 说明: $[A \times B]_{\text{补}} = [B \times A]_{\text{补}}$ 。

例 3.12 已知 $A = -1, B = 0.101$, 求 $[A \times B]_{\text{补}} = ?$

解: $[A]_{\text{补}} = 11.000, [B]_{\text{补}} = 0.101, [-A]_{\text{补}} = 01.000$ 。运算过程如算式 3.14 所示。



算式 3.14

故 $[C]_{\text{补}} = [A \times B]_{\text{补}} = 1.011000$ 。

注意 $[P_3]_{\text{补}}$ 的形成, 说明被乘数和部分积设置两位符号位是必要的, 否则移位的结果将导致乘法结果错误。

例 3.13 已知 $A = -1, B = -1$, 求 $[A \times B]_{\text{补}} = ?$

解: 假定小数点后采用 3 位表示, 则 $[A]_{\text{补}} = 11.000, [B]_{\text{补}} = 1.000, [-A]_{\text{补}} = 01.000$ 。运算过程如算式 3.15 所示。

	部分积	乘数	B_{n+1}
$[P_0]_{\text{补}}$	00. 000	1. 0 0	0 0
$[P_1]_{\text{补}}$	→ 1 00. 000	0 1. 0 0	0 0
$[P_2]_{\text{补}}$	→ 1 00. 000	0 0 1. 0 0	0 0
$[P_3]_{\text{补}}$	→ 1 00. 000	0 0 0 1. 0 0	0 0
	$+[-A]_{\text{补}}$ 01. 000		
$[P_4]_{\text{补}}$	01. 000		
		→ 1 0 0 0 0	1.

算式 3.15

故 $[C]_{\text{补}} = [A \times B]_{\text{补}} = 01.000000$ 。

此结果为 +1 表示溢出, 无意义。 $(-1) \times (-1)$ 是补码乘法惟一溢出情况。

补码比较法乘法器逻辑结构如图 3.13 所示。图中实线代表数据传输线, 虚线代表控制信号。TSG 产生两种节拍 CP_I 和 CP_{II} , 用以表示执行时间的先后次序。它们的作用同原码一位乘法器。

补码 Booth 一位乘法器与原码一位乘法器比较, 在结构上有许多相似之处。不同的是, 每一步乘法除了能作不加、加 $[A]_{\text{补}}$ 和右移一位外, 还能作减法操作即加 $[-A]_{\text{补}}$ 。操作的选择, 由 B_n 和 B_{n+1} 比较译码决定。图 3.13 中, 当指令译码器 ID 发现是一条乘法指令时, 由补码一位 Booth 乘法控制电路完成乘法。补码一位 Booth 乘法器各部分功能如下:

P 寄存器: 乘积寄存器, $n+2$ 位。初态为 0, 乘法完成时 P_{01} 和 P_{02} 存放积的符号, P_{1-n} 存放积的高 n 位。

SW_P : P 的输入开关, $n+2$ 位。乘法的前 n 步完成传送操作 $P_{i+1} \leftarrow \Sigma_i, P_0 \leftarrow \Sigma_{01}, P_1 \leftarrow \Sigma_{02}$; 乘法的最后一步完成操作 $P_i \leftarrow \Sigma_i$ 。

Σ : 加法器, $n+2$ 位。被加数: $\Sigma \leftarrow P$; 加数: $\Sigma \leftarrow A, \Sigma \leftarrow 0$ 或 $\Sigma \leftarrow \bar{A} + 2^{-n}$ 。 P, SW_P 和 Σ 均设两位符号位。

A 寄存器: 被乘数寄存器, $n+1$ 位。存放被乘数 A 的补码, 乘法过程中保持不变。

SW_A : Σ 的加数输入开关, $n+1$ 位。完成 $\Sigma \leftarrow A, \Sigma \leftarrow 0$ 或 $\Sigma \leftarrow \bar{A} + 2^{-n}$ 的转换, 该转换由 B_n 和 B_{n+1} 比较译码决定。 A 和 SW_A 只需一位符号位。

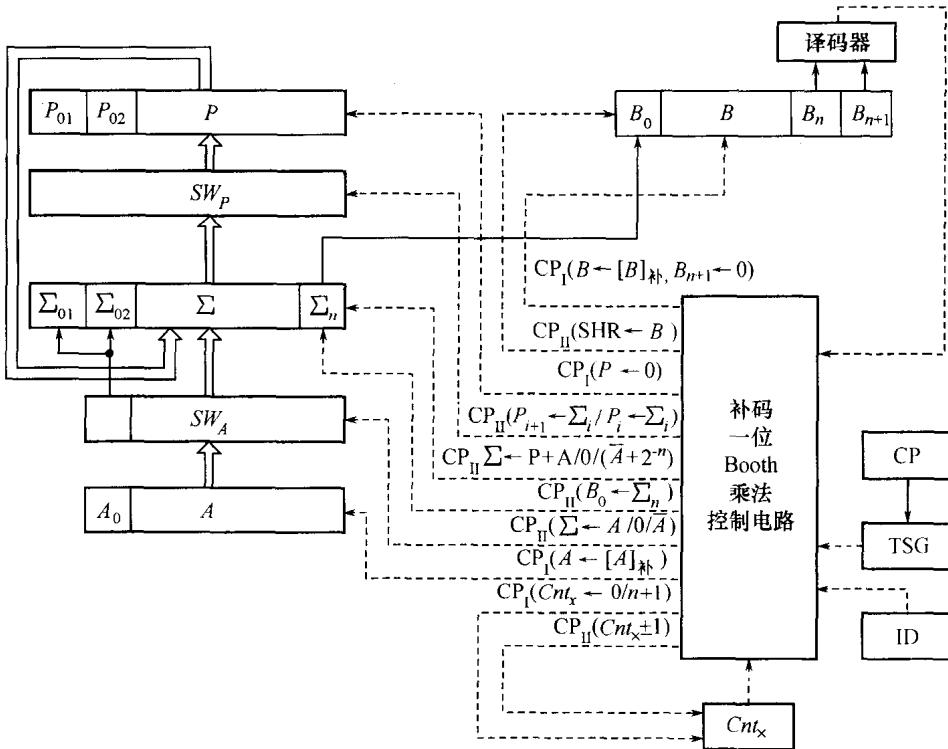


图 3.13 补码比较法乘法器

B 寄存器: 乘数寄存器, 由于增加了附加位 B_{n+1} , 故共 $n+2$ 位。初值为乘数 B 的补码, 乘法完成时 B_{1-n} 为乘积的低 n 位。它具有右移一位的功能, 右移时 $B_0 \leftarrow \Sigma_n$ 。

Cntr_x: 乘法计数器, 用于标志 $n+1$ 步乘法, 每进行一步乘法, Cntr_x 加 1 或减 1。

补码一位 Booth 乘法过程可概括如下:

- 初始状态是: $A \leftarrow [A]_{\text{补}}, B \leftarrow [B]_{\text{补}}, P \leftarrow 0, Cntr_x \leftarrow 0$ (或 $n+1$), $B_{n+1} = 0$;
- 乘法每一步: $P_{i+1} \leftarrow \Sigma_i$ (最后一步 $P_i \leftarrow \Sigma_i$); $\text{SHR } B, B_0 \leftarrow \Sigma_n; Cntr_x + 1$ (或 -1);
- $Cntr_x$ 计满 $n+1$ 步乘法时, 控制乘法完成。

3.3.3 快速乘法

乘法是计算机中使用较多的一类运算, 提高乘法的运算速度、设置快速乘法部件, 是提高机器速度的重要措施。那么怎样才能提高乘法速度呢?

两个 $n+1$ 位的定点小数相乘, 如果使用一位 Booth 乘法, 经过运算和移位, 产生的第 $n+1$ 个部分积就是结果。为加快结果的产生, 可以考虑减少产生部分积的个数, 即减少产生结果的步数。因此, 第一种提高乘法速度的方法是将每步处理一位乘数变为每步处理两位乘数。根据两位乘数的组合决定应进行什么操作, 一步求得与两位乘数相对应的部分积。

这种一步处理两位乘数的方法叫两位乘法,其速度比一位乘法提高近一倍。在两位乘法的基础上,根据同样的设计思想,可得到多位乘法的算法。

例 3.9 表明,两个 $n+1$ 位的定点小数相乘,如果使用笔算法,将产生 $n+1$ 个位积。传统的加法器一次只能处理两个操作数,所以产生最后的结果还需要使用 $n-1$ 次加法。为加快结果的产生,可以考虑加速加法的执行,例如本小节将介绍的伪加器可以一次求出三个位积之和。因此第二种方法是,采用快速硬件,一次实现多个位积的快速相加。例如,采用伪加器构成的柱形和树形乘法器。此外,利用实现多位乘的乘法器集成芯片,可构成阵列乘法器。

下面首先讨论补码两位乘法,然后简介其他快速乘法。

1. 补码两位乘法

补码两位乘法,是在 Booth 乘法的基础上推导出来的。它保留了 Booth 一位乘法的优点,因此被广泛采用。补码两位乘法作为一种单独的运算方法,可使运算速度提高近一倍;另外它也是构成多位乘法的基础。

补码两位乘法是将 Booth 一位乘法两步合并成一步而导出的。一位乘法是将本次乘数位 B_n 与上一步乘数位 B_{n-1} 进行比较,从而决定下一步进行什么运算。

设某一步的部分积为 $[P_i]_b$,据递推公式可得下一个部分积 $[P_{i+1}]_b$:

$$[P_{i+1}]_b = 2^{-1} \{ [P_i]_b + (B_{n-i+1} - B_{n-i})[A]_b \} \quad (3.32)$$

有了 $[P_{i+1}]_b$,又可以根据递推公式得下下一个部分积为 $[P_{i+2}]_b$:

$$[P_{i+2}]_b = 2^{-1} \{ [P_{i+1}]_b + (B_{n-i} - B_{n-i-1})[A]_b \} \quad (3.33)$$

将式(3.32)和式(3.33)的两步乘法合并成一步,即可从 $[P_i]_b$ 一步求得 $[P_{i+2}]_b$:

$$\begin{aligned} [P_{i+2}]_b &= 2^{-1} \{ [P_{i+1}]_b + (B_{n-i} - B_{n-i-1})[A]_b \} \\ &= 2^{-2} \{ [P_i]_b + (B_{n-i+1} - B_{n-i})[A]_b + 2(B_{n-i} - B_{n-i-1})[A]_b \} \\ &= 2^{-2} \{ [P_i]_b + (B_{n-i+1} - B_{n-i} + 2B_{n-i} - 2B_{n-i-1})[A]_b \} \\ &= 2^{-2} \{ [P_i]_b + (B_{n-i+1} + B_{n-i} - 2B_{n-i-1})[A]_b \} \end{aligned} \quad (3.34)$$

式(3.34)表明,欲求 $[P_{i+2}]_b$,可先求 $[P_i]_b + (B_{n-i+1} + B_{n-i} - 2B_{n-i-1})[A]_b$,再右移两位即可。与一位补码乘法不同的是,两位补码乘法的乘数寄存器在每计算出一个新的部分积后需要右移两位。这样,判断位 B_{n-i+1} 、 B_{n-i} 和 B_{n-i-1} 就是每次移位后乘数寄存器的最后三位 B_{n+1} 、 B_n 和 B_{n-1} 。 B_{n-1} 、 B_n 和 B_{n+1} 有 8 种可能的值,将其代入式(3.34),即得到补码两位比较乘法的规则,如表 3.9 所示。

对于补码一位比较乘法,需要逐位地判断乘数位,因此完成一次乘法,需要作 $n+1$ 步操作,最后一步部分积不移位。与一位补码比较乘法一样,两位补码比较乘法的符号位也要参与运算。同时,两位补码比较乘法的乘数也需要增加一个附加判断位 B_{n+1} ,其初值为 0。但两位比较乘法最后一步是否移位,决定于乘数符号位的位数。

当 n 为偶数时,1 位 Booth 乘法和 2 位快速乘法递推公式的对比列于表 3.10。

表 3.9 补码两位比较乘法规则

判 断 位			新 部 分 积	操 作
B_{n-1}	B_n	B_{n+1}		
0	0	0	$[P_{i+2}]_b = 2^{-2}[P_i]_b$	$\rightarrow 2$
0	0	1	$[P_{i+2}]_b = 2^{-2}\{[P_i]_b + [A]_b\}$	$+ [A]_b, \rightarrow 2$
0	1	0	$[P_{i+2}]_b = 2^{-2}\{[P_i]_b + [A]_b\}$	$+ [A]_b, \rightarrow 2$
0	1	1	$[P_{i+2}]_b = 2^{-2}\{[P_i]_b + [2A]_b\}$	$+ [2A]_b, \rightarrow 2$
1	0	0	$[P_{i+2}]_b = 2^{-2}\{[P_i]_b + [-2A]_b\}$	$+ [-2A]_b, \rightarrow 2$
1	0	1	$[P_{i+2}]_b = 2^{-2}\{[P_i]_b + [-A]_b\}$	$+ [-A]_b, \rightarrow 2$
1	1	0	$[P_{i+2}]_b = 2^{-2}\{[P_i]_b + [-A]_b\}$	$+ [-A]_b, \rightarrow 2$
1	1	1	$[P_{i+2}]_b = 2^{-2}[P_i]_b$	$\rightarrow 2$

表 3.10 n 为偶数时递推公式的比较

1 位 Booth 乘法	2 位快速乘法
$[P_0]_b \equiv 0$	$[P_0]_b \equiv 0$
$[P_1]_b = 2^{-1}\{[P_0]_b + (B_{n+1} - B_n) \times [A]_b\}$	$[P_1]_b = 2^{-2}\{[P_0]_b + (B_{n+1} + B_n - 2B_{n-1}) \times [A]_b\}$
$[P_2]_b = 2^{-1}\{[P_1]_b + (B_n - B_{n-1}) \times [A]_b\}$	\vdots
$[P_{i-1}]_b = 2^{-1}\{[P_{i-2}]_b + (B_{n-i+3} - B_{n-i+2}) \times [A]_b\}$	$[P_{i-1}]_b = 2^{-1}\{[P_{i-1}]_b + (B_{n-i+3} + B_{n-i+2} - 2B_{n-i+1}) \times [A]_b\}$
$[P_i]_b = 2^{-1}\{[P_{i-1}]_b + (B_{n-i+2} - B_{n-i+1}) \times [A]_b\}$	\vdots
$[P_{n-1}]_b = 2^{-1}\{[P_{n-2}]_b + (B_3 - B_2) \times [A]_b\}$	$[P_{n-1}]_b = 2^{-1}\{[P_{n-1}]_b + (B_3 + B_2 - 2B_1) \times [A]_b\}$
$[P_n]_b = 2^{-1}\{[P_{n-1}]_b + (B_2 - B_1) \times [A]_b\}$	$[C]_b = [P_{n/2+1}]_b = [P_{n/2}]_b + (B_1 - B_0) \times [A]_b$
$[C]_b = [P_{n+1}]_b = [P_n]_b + (B_1 - B_0) \times [A]_b$	$= [P_{n/2}]_b + (B_1 + B_{02} - 2B_{01}) \times [A]_b$

当 n 为奇数时, 1 位 Booth 乘法和 2 位快速乘法递推公式的对比列于表 3.11。

表 3.11 n 为奇数时递推公式的比较

1 位 Booth 乘法	2 位快速乘法
$[P_0]_b \equiv 0$	$[P_0]_b \equiv 0$
$[P_1]_b = 2^{-1}\{[P_0]_b + (B_{n+1} - B_n) \times [A]_b\}$	

(续表)

1位 Booth 乘法	2位快速乘法
$[P_2]_{\text{补}} = 2^{-1} \{ [P_1]_{\text{补}} + (B_n - B_{n-1}) \times [A]_{\text{补}} \}$	$[P_1]_{\text{补}} = 2^{-2} \{ [P_0]_{\text{补}} + (B_{n+1} + B_n - 2B_{n-1}) \times [A]_{\text{补}} \}$
\vdots	\vdots
$[P_{i-1}]_{\text{补}} = 2^{-1} \{ [P_{i-2}]_{\text{补}} + (B_{n-i+3} - B_{n-i+2}) \times [A]_{\text{补}} \}$	$[P_i]_{\text{补}} = 2^{-2} \{ [P_{i-1}]_{\text{补}} + (B_{n-i+3} + B_{n-i+2} - 2B_{n-i+1}) \times [A]_{\text{补}} \}$
\vdots	\vdots
$[P_{n-2}]_{\text{补}} = 2^{-1} \{ [P_{n-3}]_{\text{补}} + (B_4 - B_3) \times [A]_{\text{补}} \}$	
$[P_{n-1}]_{\text{补}} = 2^{-1} \{ [P_{n-2}]_{\text{补}} + (B_3 - B_2) \times [A]_{\text{补}} \}$	$[P_{(n-1)/2}]_{\text{补}} = 2^{-2} \{ [P_{(n-1)/2-1}]_{\text{补}} + (B_4 + B_3 - 2B_2) \times [A]_{\text{补}} \}$
$[P_n]_{\text{补}} = 2^{-1} \{ [P_{n-1}]_{\text{补}} + (B_2 - B_1) \times [A]_{\text{补}} \}$	
$[C]_{\text{补}} = [P_{(n+1)/2}]_{\text{补}}$ $= 2^{-1} \{ [P_{(n-1)/2}]_{\text{补}} + (B_2 - B_1) \times [A]_{\text{补}} \} + (B_1 - B_0) \times [A]_{\text{补}}$ $= 2^{-1} \{ [P_{(n-1)/2}]_{\text{补}} + (B_2 + B_1 - 2B_0) \times [A]_{\text{补}} \}$	

表 3.10 和表 3.11 中, 对应行的部分积是相等的。从表 3.10 中可以看出, 对于补码两位乘法, 当乘数尾数的位数 n 为偶数时, 乘数需要两位符号位 B_{01} 和 B_{02} , 共需作 $n/2 + 1$ 步操作, 最后一步部分积不移位; 从表 3.11 中可以看出, 当乘数尾数的位数 n 为奇数时, 乘数只需要一个符号位 B_0 , 共需作 $(n+1)/2$ 步操作, 最后一步部分积右移一位。

需要说明的是, 对于补码两位乘法, 被乘数 A 需要使用 3 位符号位。由于 $A = -1$ 时, A 的补码是有意义的, $[2A]_{\text{补}} = 110.00\cdots 0$, 而 $[-2A]_{\text{补}} = 010.00\cdots 0$ 。如果使用两位符号位, $[-2A]_{\text{补}}$ 将无法表示。同样, 为保证在进行加法时部分积的符号位不被破坏, 部分积也要采用三位符号位。

在每一步乘法的操作中, $[\pm A]_{\text{补}}$ 和 $[\pm 2A]_{\text{补}}$ 是如何实现的呢? 它是通过被乘数的子倍数选择开关来实现的, 子倍数选择开关只要使用四与或门即可。图 3.14 给出了补码两位乘法第 i 位的被乘数子倍数选择开关, 把 ± 1 、 ± 2 叫做被乘数 $[A]_{\text{补}}$ 的子倍数。

前面讲过, 对寄存器左移一位等效于对寄存器内容增大一倍的运算, $[\pm 2A]_{\text{补}}$ 的形成正是利用了这一结论。如果需要 $+[2A]_{\text{补}}$, 则将 $[A]_{\text{补}}$ 左移一位送给加法器。如果需要 $-[A]_{\text{补}}$ 或者 $-[2A]_{\text{补}}$, 则将 $[A]_{\text{补}}$ 或者将 $[A]_{\text{补}}$ 左移一位后按位取反送到加法器, 然后在最后

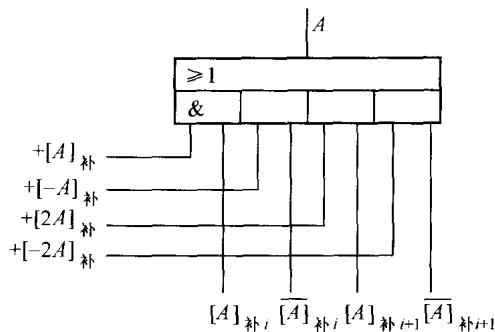
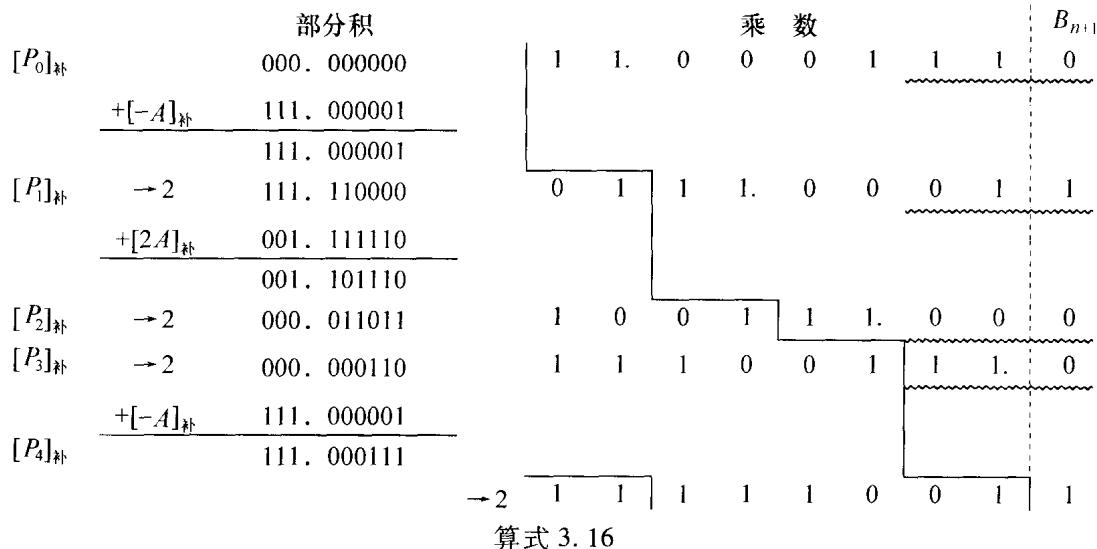


图 3.14 子倍数选择开关

一位加 1 修正。这样,为形成新的部分积,只要把子倍数选择开关第 i 位的输出 A_i 送到加法器与上次部分积的第 i 位相加,就达到了 $[P_i]_{\text{补}} \pm [A]_{\text{补}}$ 或 $[P_i]_{\text{补}} \pm [2A]_{\text{补}}$ 的目的。可见,两位乘法增加的设备并不多,但却可以使乘法速度提高一倍。

例 3.14 已知 $A = 0.111111$, $B = -0.111001$, 求 $[A \times B]_{\text{补}} = ?$

解: $[A]_{\text{补}} = 000.111111$, $[-A]_{\text{补}} = 111.000001$, $[2A]_{\text{补}} = 001.11110$; 因 $n=6$ 为偶数, 所以 $[B]_{\text{补}}$ 需两位符号位, $[B]_{\text{补}} = 11.000111$ 。运算过程如算式 3.16 所示。



算式 3.16

例 3.14 中, 每步乘法乘数都右移两位, 最后一步乘数右移两位是使乘积的低 n 位正确存放到乘数寄存器的尾数中。因 $n = 6$ 为偶数, 所以最后一步乘法部分积不移位, 需作 $n/2 + 1 = 4$ 步乘法。

故: $[C]_{\text{补}} = [A \times B]_{\text{补}} = 1.00011111001$ 。

例 3.15 已知 $A = -1, B = -0.11001$, 求 $[A \times B]_{\text{补}} = ?$

解: $[A]_{\text{补}} = 111.00000, [-A]_{\text{补}} = 001.00000, [2A]_{\text{补}} = 110.00000, [-2A]_{\text{补}} = 010.00000$; 因 $n=5$ 为奇数, 所以 $[B]_{\text{补}}$ 需一位符号位, $[B]_{\text{补}} = 1.00111$, 运算过程如算式 3.17 所示。

		部分积				乘 数			B_{n+1}
$[P_0]_{\text{补}}$		000.	00000			1.	0	0	
	$+[-A]_{\text{补}}$	001.	00000			1	1	1	0
		001.	00000						
$[P_1]_{\text{补}}$	$\rightarrow 2$	000.	01000			0	0	1	1
	$+[2A]_{\text{补}}$	110.	00000						
		110.	01000						
$[P_2]_{\text{补}}$	$\rightarrow 2$	111.	10010			0	0	0	0
	$+[-2A]_{\text{补}}$	010.	00000						
		001.	10010						
$[P_3]_{\text{补}}$	$\rightarrow 1$	000.	11001	$\rightarrow 2$	1	0	0	0	1.
						0	0	0	

算式 3.17

同例 3.14 一样, 每步乘法乘数都右移两位, 最后一步乘数右移两位是使乘积的低 n 位正确存放到乘数寄存器的尾数中。因为 $n=5$ 为奇数, 所以最后一步乘法部分积右移一位, 共需作 $(n+1)/2=3$ 步乘法。

故: $[C]_{\text{补}} = [A \times B]_{\text{补}} = 0.1100100000$ 。

例 3.16 已知 $A = -1, B = -1$, 求 $[A \times B]_{\text{补}} = ?$

解: 假定尾数部分位数 $n=3$, 则: $[A]_{\text{补}} = 111.000, [-A]_{\text{补}} = 001.000, [-2A]_{\text{补}} = 010.000$; 因 $n=3$ 为奇数, 所以 $[B]_{\text{补}}$ 需一位符号位, $[B]_{\text{补}} = 1.000$ 。运算过程如算式 3.18 所示。

		部分积				乘 数			B_{n+1}
$[P_0]_{\text{补}}$		000.	000			1.	0	0	
	$+[-A]_{\text{补}}$	000.	000			0	0	1.	0
		010.	000						
$[P_1]_{\text{补}}$	$\rightarrow 2$	001.	000						
	$+[-2A]_{\text{补}}$	010.	000						
		010.	000						
$[P_2]_{\text{补}}$	$\rightarrow 1$	001.	000	$\rightarrow 2$	0	0	0	0	1.
						0	0	0	

算式 3.18

每步乘法乘数都右移两位, 最后一步乘数右移两位是使乘积的低 n 位正确存放到乘数

寄存器的尾数中。因为 $n=3$ 为奇数, 所以最后一步乘法部分积右移一位, 共需作 $(n+1)/2 = 2$ 步乘法。

结果 $[P_2]_{\text{补}} = 001.000$, 代表 +1, 表明发生过溢出。当 $A=B=-1$ 时, 乘法结果为 +1, 发生溢出, 这是补码乘法溢出的惟一一种情况。

从上面的例子都可看出, 乘数寄存器在乘法完成后, 寄存的是 $2n+1$ 位乘积的低 n 位。但应注意, 最后乘数寄存器的符号位内容无意义, 有意义的只是尾数。

显然, 每次作乘法前, 必须置 0 增设的乘数附加位 B_{n+1} 。因为作完整个乘法后, B_{n+1} 的内容可能任意(1 或 0), 如果残存的是 1 而不置 0 的话, 则下一次乘法的第一步就必定出错。

一般情况下, 机器字长均为 2 的幂次方, 如 16、32、64 等。操作数在存储和传送时通常带有一位符号位, 即操作数的尾数 n 通常为奇数。根据前面的分析, 当乘数尾数的位数 n 为奇数时, 乘数只需要一个符号位, 共需作 $(n+1)/2$ 步操作, 最后一步部分积右移一位。

如图 3.15 所示为 n 为奇数时的补码两位比较乘法器的原理图。图中实线代表数据传输线, 虚线代表控制信号。TSG 产生两种节拍 CP_I 和 CP_{II} , 用以表示执行时间的先后次序。它们的作用同补码一位比较乘法器。

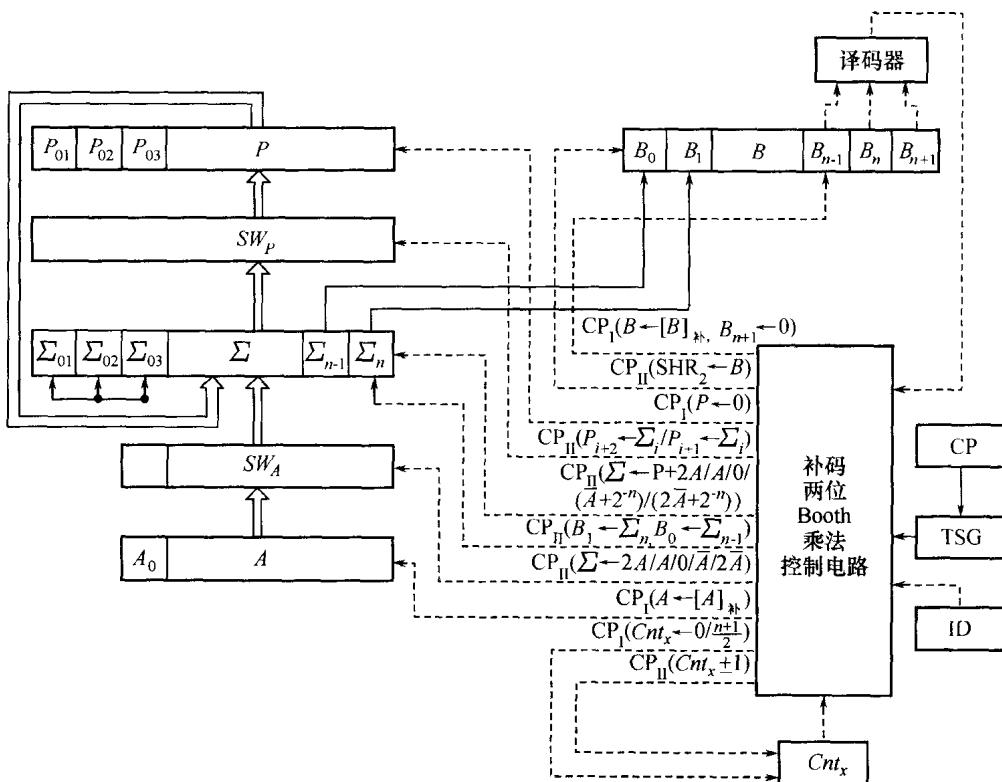


图 3.15 n 为奇数的两位比较乘法器

图 3.15 所示的两位比较乘法器和图 3.13 所示的一位比较乘法器很相似, 并未增加更多的设备。所不同处, 只是比较判断的两位译码变成了三位译码; SW_A 由三选一开关变成了五选一开关; P 、 Σ 和 SW_p 各增加一个符号位, Σ 右斜送一位和直送, 变成了右斜送两位和右斜送一位; 计数器的初值减小(因为乘法步数变少)。

补码两位 Booth 乘法过程可概括如下:

- 初始状态是: $A \leftarrow [A]_b$, $B \leftarrow [B]_b$, $P \leftarrow 0$, $Cnt_x \leftarrow 0$ (或 $(n+1)/2$), $B_{n+1} = 0$;
- 乘法每一步: $P_{i+2} \leftarrow \Sigma_i$ (最后一步 $P_{i+1} \leftarrow \Sigma_i$); $SHR_2 B$, $B_0 \leftarrow \Sigma_{n-1}$, $B_1 \leftarrow \Sigma_n$; $Cnt_x + 1$ (或 -1);
- Cnt_x 计满 $(n+1)/2$ 步乘法时, 控制乘法完成。

2. 多位乘法

根据一位乘法的递推公式, 将多个部分积的产生合并为一步, 就形成了多位乘法。两位以上的乘法, 都可算作多位乘法。例如, 将补码一位乘法比较法三步合成一步, 就构成了补码三位比较乘法。

如果按照同样的设计思想, 去推导更多位数的乘法, 将出现新的问题。因为当一次比较的位数增加时, 被乘数 A 的子倍数将成倍增加。例如: 当比较位数为 3 位时, 其子倍数为 8 (包括 ± 1 、 ± 2 、 ± 3 和 ± 4 , 有兴趣的读者可以自行推导); 当比较位数为 4 位时, 其子倍数为 16。随着子倍数选择开关的成倍增加, 不但会使子倍数选择开关变得越来越复杂, 而且还会出现若干难于处理的操作。例如, $[3A]_b$ 就不可以直接通过 $[A]_b$ 移位实现。如果将多位乘法, 采取类似于两位、三位乘法一样, 一位一位地简单堆砌, 在设计实践上并没什么实际意义。因此, 目前的多位乘法多采取以一位乘或两位乘为基础, 同时形成与整个乘数对应的多项位积, 然后用多操作数加法网络相加, 快速求出乘积结果。若采用实现多位乘的专用集成芯片, 可组成更多位的乘法器——阵列乘法器。

(1) 多操作数加法网络——柱形乘法器

为解决多操作数相加时的快速问题, 通常使用存储进位加法器 (Carry Save Adders, CSA)。它的基本思想是在同一级加法器中将进位信息暂时保留, 留待下一级加法器或以后级处理。与常规的带进位传播加法器 (Carry Propagate Adders, CPA) 相比, 计算的总效果一致, 但 CSA 不需要考虑同级 CSA 间的进位问题, 打断了同级进位链, 故可以实现快速加法。从结构和逻辑上讲, 一个一位 CSA, 就是一个一位全加器 FA。在外部功能上看, 加法器既可以对两个操作数与低位来的进位信号进行全加, 也可以对三个操作数 (无进位) 进行伪加。为与带进位的常规全加器有所区别, 通常称 CSA 为伪加器, 其所得之和叫伪加和。例如, 三个数 a 、 b 和 c 的第 i 位相加, 得到的伪加和 S_{pi} 与伪加进位 C_{pi} 的真值表列于表 3.12。

但是, 要得到这三个 n 位数的真正的和, 还要把伪加和 S_{pi} 与左移一位后的伪加进位 C_{pi} 在常规加法器 (为加快速度, 通常使用 CPA) 上相加才能得到。即:

$$S = \sum_{i=1}^n (S_{pi} + 2C_{pi}) = a + b + c$$

表 3.12 伪加器真值表

操作数			伪加进位 C_{pi}	伪加和 S_{pi}
a_i	b_i	c_i		
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

由此可见,三个数相加,需要通过一次 CSA 以及一次 CPA 求得真正的和。以此类推,若四个数 a, b, c 和 d 相加,则可以通过两次 CSA 以及一次 CPA 完成。第一次 CSA 完成操作:

$$a + b + c = (S_{pi}, C_{pi})$$

第二次 CSA 完成操作:

$$S_{pi} + 2C_{pi} + d = (S_{pii}, C_{pii})$$

第三次 CPA 完成操作:

$$\sum_{ii=1}^n (S_{pii} + 2C_{pii}) = S$$

这样构造的多操作数加法网络,分为若干级,前面几级采用不考虑进位的 CSA,最后一级采用考虑进位的 CPA,而 CPA 采用高速进位链。采用多级 CSA 和一级 CPA 构成的柱形乘法器如图 3.16 所示。

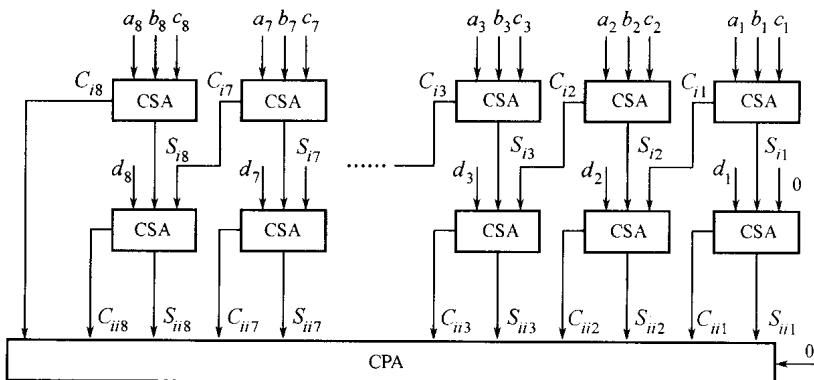


图 3.16 柱形乘法器

图 3.16 中的柱形乘法器只采用了两级 CSA,其中第一级产生 (S_i, C_i) ,第二级产生 (S_{ii}, C_{ii}) 。第一级三操作数相加,对应三项位积;以后每一级加入一个操作数即一项位积。每一

级产生的伪加和直接送往下一级对应的同一位置，伪加进位则左移一位传到下一级。因为采用 CSA，使进位留待下一级进行处理，故同级各位之间无进位关系。第二级的输入，一是第一级来的伪加和，二是新增加的位积输入，三是第一级来的伪加进位。在这一级中，各位之间仍无进位关系，依旧采用 CSA。最后一级采用带进位传播加法器 CPA，每位全加器接收 3 个输入：一是上一级直传来的伪加和；二是上一级左移一位传送来的伪加进位；三是本级 CPA 中低位来的进位信号。这一级 CPA 采用并行进位链。

由于 CSA 部分信号是沿纵向传递，故送往 CPA 的输入信号延迟取决于柱形网络的级数和 CSA 本身的延迟。因 CPA 采用并行进位链，故进位传递很快。从柱形乘法器的工作原理可知，如果 m 个数据相加，则需要 $m - 2$ 级 CSA 和一级 CPA。这种柱形网络为多位乘提供了途径；设乘数尾数为 n 位，可按一位乘算法同时形成 n 项位积，或按两位乘算法同时形成 $n/2$ 项两位积，然后用若干级 CSA 与一级 CPA 一步求得 n 位乘的乘积。但是，如果操作数的个数较多，需要的 CSA 的级数也就较多，而级数的多少将影响到乘法器的速度。为减少 CSA 的级数，可以采用树形乘法器。

(2) 多操作数加法网络——树形乘法器

树形乘法器的基本思想是：在每一级尽可能多地使用 CSA，以处理该级可相加的数位，从而减少整个网络的级数，提高整个乘法器的速度。

假定乘法的操作数为 6 位，则将产生 6 项位积 $a \sim f$ 。使用多操作数加法网络构成的树形乘法器如图 3.17 所示。

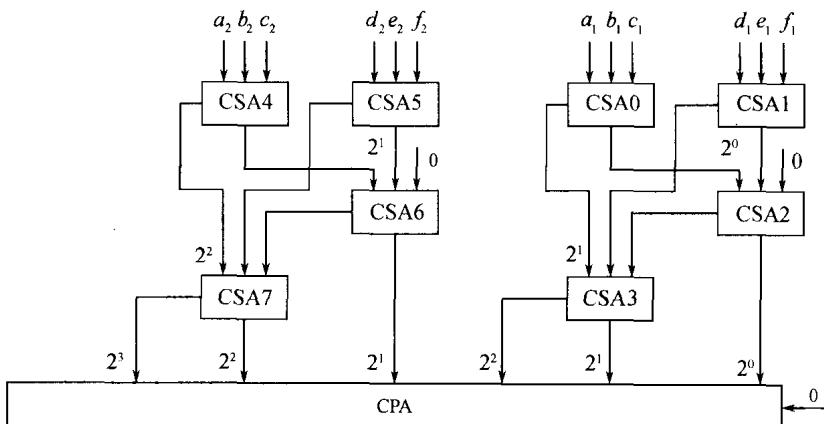


图 3.17 树形乘法器

图 3.17 中只画出 6 个操作数的最低两位部分逻辑，CSA0 与 CSA1 处理操作数（位积）中的 2^0 位，每个 CSA 将三个 2^0 位相加，输出的伪加和 S 权重为 2^0 ，伪加进位 C 权重为 2^1 。CSA2 处理来自 CSA0 和 CSA1 的伪加和，产生的伪加和，权重为 2^0 。CSA3 处理来自 CSA0、CSA1 和 CSA2 产生的伪加进位，产生的两个输出结果的权重分别为 2^1 和 2^2 。同样，CSA4 与

CSA5 处理位积中的 2^1 位, 输出 S 为 2^1 位, C 为 2^2 位。经 CSA6 和 CSA7 形成 3 位输出, 权分别为 2^1 、 2^2 和 2^3 。最后一级 CPA 求得乘积结果。

比较而言, 树形乘法器处理 6 个位积的时候需要 3 级 CSA, 而柱形乘法器需要 4 级 CSA。操作数位数越多, 这种差异越大。

(3) 阵列乘法器

运用多操作数加法网络思想, 可实现多位乘法, 并制成集成化阵列乘法器芯片。用若干块这种芯片可构造更多位数的快速乘法器。阵列乘法器芯片包含两部分: 一是用若干与门产生和操作数数位对应的多个位积数位; 二是用多操作数加法网络求乘积。

例 3.17 已知被乘数 $A = a_5 \ a_4 \ a_3 \ a_2 \ a_1$, 乘数 $B = b_5 \ b_4 \ b_3 \ b_2 \ b_1$, 求 $A \times B = ?$

解: 被乘数和乘数均为 5 位, 其运算过程如算式 3.19 所示。

\times)	$A =$	a_5	a_4	a_3	a_2	a_1			
	$B =$	b_5	b_4	b_3	b_2	b_1			
		$a_5 b_1$	$a_4 b_1$	$a_3 b_1$	$a_2 b_1$	$a_1 b_1$			
		$a_5 b_2$	$a_4 b_2$	$a_3 b_2$	$a_2 b_2$	$a_1 b_2$			
		$a_5 b_3$	$a_4 b_3$	$a_3 b_3$	$a_2 b_3$	$a_1 b_3$			
		$a_5 b_4$	$a_4 b_4$	$a_3 b_4$	$a_2 b_4$	$a_1 b_4$			
+)	$a_5 b_5$	$a_4 b_5$	$a_3 b_5$	$a_2 b_5$	$a_1 b_5$				
P_{10}	P_9	P_8	P_7	P_6	P_5	P_4	P_3	P_2	P_1

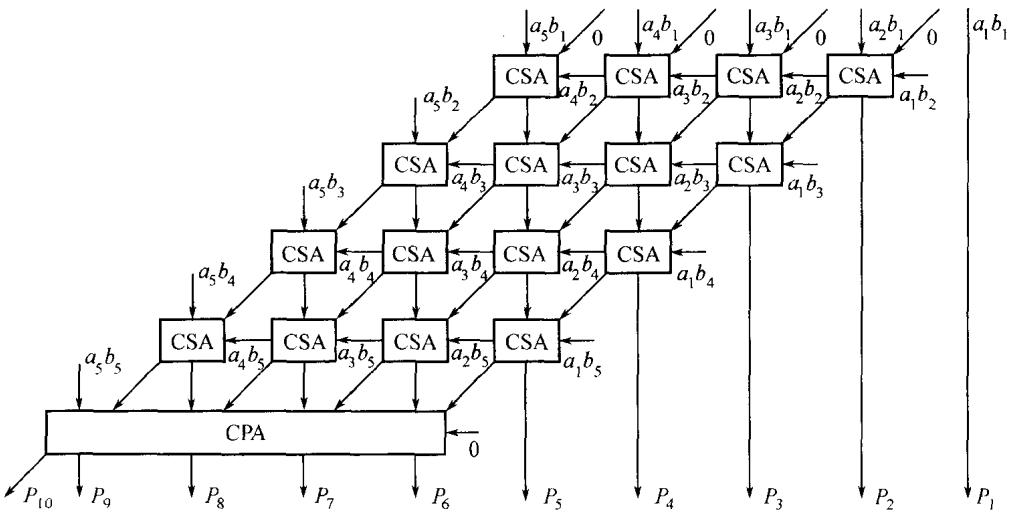
算式 3.19

算式 3.19 中, 乘积 P 由 5 项位积相加而得, 每一项位积有 5 个数位, 这些数位 $a_i b_j$ 可同时用与门逻辑产生。一旦向芯片输入操作数 A 和 B , 通过与门逻辑马上可得到这些数位信息。算式 3.19 中的 5×5 位乘法所对应的 5×5 乘法阵列逻辑如图 3.18 所示。

图 3.18 中前 4 行采用 CSA, 最后一行采用 CPA, 其进位结构可采用并行进位链。

在该阵列乘法器中, 第一级 CSA 将两项位积 Ab_1 和 Ab_2 相加, 以后每一级加入一项位积, 同一级中无进位连接, 且保存左移一位传至下一级的高位。图中乘法器没考虑符号位, 适用于原码乘法器。若用于补码乘法, 需增 A 和 B 的求补逻辑。

以上介绍了两种提高乘法运算速度的方法: 一种是每次用乘数的多位与被乘数相乘以减少位积的个数和加法的次数; 另一种是利用伪加器对位积进行快速加法。当然, 提高乘法运算速度的方法还有很多。例如将上面这两种方法结合在一起, 既可以减少位积的个数, 又可以提高加法的速度。再比如, 也可以将被乘数与乘数的积用表格的方法实现, 即直接通过被乘数和乘数进行译码查表得到它们的积。当然这种方法的硬件代价很高, 尤其是在乘法位数比较多的情况下。这些方法的有关具体实现还有很多技术细节, 有兴趣的读者可查阅有关文献或自行思考。

图 3.18 5×5 阵列乘法器

3.4 定点除法运算

除法,理论上是乘法的逆运算,在算法上本质是一种试探法。对十进制除法,试探被除数或余数有几倍除数,这种试探过程可以通过够不够减来判断,结果得到一位十进制的商和一个新余数。二进制除法可模仿十进制除法运算,它试探被除数或者余数是大于等于还是小于除数,结果得到一位二进制的商和一个新余数。

与乘法的处理思想类似,除法运算的常规算法是将其转换为若干次的加减和移位操作。本节首先介绍常规的除法算法:原码一位加减交替法和补码一位比较除法。但与乘法一样,目前机器中广泛采用的还是补码除法。在讨论算法的同时,还要介绍原码除法和补码除法对应除法器的构成。最后简介多位除法的快速算法。

定点数据表示分为定点整数和定点小数两种数据形式,它们的除法运算过程类似,下面以定点小数为例进行讨论。

3.4.1 原码一位除法

在讨论原码一位除法前,先看一看二进制笔算除法的过程,以便于分析机器二进制除法运算的原理。

首先确定商的符号。与乘法类似,只要被除数和除数异号,商的符号就为负;而被除数和除数同号,商的符号就为正。因而只要将被除数符号与除数符号异或即可得商的符号。

符号位单独处理,那么商的数值部分怎样得到呢?理论上,除法是乘法的逆运算;算法

上,除法进行的是一种试探法或比较法。对于十进制数除法,可试探被除数中有几倍除数,从而得到一位十进制商。类似的,对于二进制除法,则是试探或比较被除数或余数是大于还是小于除数来进行上商,商非1即0。

例 3.18 已知被除数 $A = 0.1001$, 除数 $B = 0.1011$, 求商 $C = ?$ 余数 $R = ?$

解:根据上述二进制除法的笔算试探方法,其除法过程如算式 3.20 所示。

0.1011	0. 1 0 0 1 0	C
-) 0.	0 1 0 1 1	$-2^{-1}B$
-) 0.	0 0 1 1 1 0	R_1
-) 0.	0 0 1 0 1 1	$-2^{-2}B$
-) 0.	0 0 0 0 1 1 0 0	R_2, R_3
-) 0.	0 0 0 0 0 1 0 1 1	$-2^{-4}B$
	0. 0 0 0 0 0 0 0 1	R_4

算式 3.20

故: $C = 0.1101, R = R_4 = 0.00000001$ 。

据此,可将二进制小数除法笔算法的过程归纳于表 3.13。

表 3.13 二进制小数除法过程例

步骤	比 较	C_i	R_i	备 注
1	$A < B$	$C_0 = 0$	$R_0 = A - 0 = A$	若 $A \geq B$ 为溢出
2	$A > 2^{-1}B$	$C_1 = 1$	$R_1 = R_0 - 2^{-1}B$	
3	$R_1 > 2^{-2}B$	$C_2 = 1$	$R_2 = R_1 - 2^{-2}B$	
4	$R_2 < 2^{-3}B$	$C_3 = 0$	$R_3 = R_2$	
5	$R_3 > 2^{-4}B$	$C_4 = 1$	$R_4 = R_3 - 2^{-4}B$	

从上面的运算过程,可以得到以下结论:

(1) 被除数的绝对值必须小于除数的绝对值。

为使除法不溢出,必须满足 $|A| < |B|$, 即商的整数部分必为0,亦即 $|C| < 1$ 。

(2) 比较上商。

判断被除数(或余数)和除数的大小,若除数小于或等于被除数(或余数),则该位商上1,且从被除数(或余数)中减去除数,得到新的余数;若除数大于被除数(或余数),则该位商上0,且被除数(或余数)不变。然后,将余数的最低位补0,再与除数比较,直到除尽或得到的商满足要求的位数为止。

(3) 余数是逐渐减少的。

从例 3.18 可以看出真余数的变化情况:

$$R_1 = 0.0111 \times 2^{-1} < 2^{-1}B$$

$$R_2 = 0.0011 \times 2^{-2} < 2^{-2}B$$

$$R_3 = 0.0110 \times 2^{-3} < 2^{-3}B$$

$$R_4 = 0.0001 \times 2^{-4} < 2^{-4}B$$

可见余数是逐步减小,而且满足: $A = B \times C + R_4$, $0 \leq R_4 < 2^{-4}B$ 。一般地, $A = B \times C + R_n$, $0 \leq R_n < 2^{-n}B$ 。

下面讨论机器除法的过程,被除数(或余数)与除数的比较过程通过减法实现。

若参加运算的为两个 $n+1$ 位定点小数。如果使用原码完成 $C = A/B$ 的计算,记:

被除数 A 的原码: $[A]_{\text{原}} = A_0, A_1 A_2 \dots A_n$ 。

除数 B 的原码: $[B]_{\text{原}} = B_0, B_1 B_2 \dots B_n$ 。

商 C 的原码: $[C]_{\text{原}} = C_0, C_1 C_2 \dots C_n$ 。

余数 R 的原码: $[R]_{\text{原}} = r_0, r_1 r_2 \dots r_n$ 。

于是有: $[A]_{\text{原}} = [B]_{\text{原}} \times [C]_{\text{原}} + [R]_{\text{原}} \times 2^{-n}$ 。

原码除法的符号位需要单独处理,数值部分为绝对值相除。被除数(或余数)减去除数后,若新余数为正,则表示“够减”,该位商上 1,得到的余数就是正确的余数;若新余数为负,表示“不够减”,该位商上 0,得到的余数为错误的余数(假余数),为保证下一步除法的正确执行,需要恢复成该步除法原来的被除数(或余数)。除法的运算规则描述为:

(1) 符号位单独处理: $C_0 = A_0 \oplus B_0$ 。

(2) 数值部分为两正数相除,第一步除法通过 $R_1 = |A| - |B|$ 实现,其余每一步除法均通过 $R_{i+1} = 2R_i - |B|$ 来实现,其中 $i = 1, \dots, n$ 。若 $R_{i+1} \geq 0$, 即新余数为正, 则商上 1; 若 $R_{i+1} < 0$, 即新余数为负, 商上 0, 此时应再加上 $|B|$, 以恢复成原来的余数。

使用这种运算规则,在除法的运算过程中,恢复余数的过程势必影响除法速度,而且除法的步数不固定,控制上也就较复杂。能不能去掉恢复余数的过程呢?回答是肯定的。设除法运算过程中,若某步得到的余数 R_i 是假余数,即 $R_i < 0$ (符号位 $r_0 = 1$)。此时要得到下步除法的新余数 R_{i+1} ,需在恢复余数后左移一位再减 $|B|$ 即得到新余数 R_{i+1} :

即: $R_{i+1} = 2(R_i + |B|) - |B|$

将上式进行变换可得:

$$R_{i+1} = 2R_i + 2|B| - |B| = 2R_i + |B|$$

这就是说,当某步除法得到 $R_i < 0$ (假余数)时,不必恢复余数,只要将 R_i 视为真余数,左移 1 位,再加上 $|B|$ 就可得到新余数 R_{i+1} 。即本次余数为正,下步除法作减法;本次余数为负,下步除法改作加法。鉴于上述的操作特点,这种方法称为加减交替法。

于是,原码加减交替除法的规则可归结如下:

(1) 商的符号位单独处理, $C_0 = A_0 \oplus B_0$ 。

(2) 取绝对值(原码尾数)相除,为不使商溢出,要求被除数绝对值小于除数绝对值,即

$|A| < |B|$, 除法第一步做减法。

(3) 由于加减法操作是由余数的符号位控制,且要等余数左移一位后再进行操作,这时余数的符号位可能被破坏。因此,运算中被除数 $|A|$ 、除数 $|B|$ 和余数应设两位符号位。

(4) 当余数 R_i 为正,商 C_i 上1,余数左移一位,减去 $|B|$ 得新余数 R_{i+1} ; R_i 为负,商 C_i 上0,余数左移一位,加上 $|B|$ 得新余数 R_{i+1} 。用公式可统一描述为:

$$R_{i+1} = 2R_i + (1 - 2C_i) \lceil B \rceil \quad (3.35)$$

式(3.35)表明:

- R_i 为正,表示“够减”,商 $C_i = 1$,第 $i + 1$ 步做 $2R_i - |B|$ 。
- R_i 为负,表示“不够减”,商 $C_i = 0$,第 $i + 1$ 步做 $2R_i + |B|$ 。

(5) 上述操作共作 $n + 1$ 步,得商的绝对值,共 $n + 1$ 位(最高位为符号位,且仅起占位作用)。

(6) 如最后一步除法得到的 R_n 是假余数,而又需保留余数时,则应恢复余数,即需加 $|B|$ 得正确余数。

(7) 最后: $[C]_{原} = (A_0 \oplus B_0) + \lceil [C]_{原} \rceil$; $[R_n]_{原真} = 2^{-n} \lceil [R_n]_{原} \rceil + A_0.00\ldots0$ 。

例 3.19 已知 $[A]_{原} = 1.1001$, $[B]_{原} = 0.1011$,求 $[A/B]_{原} = ?$

解: $\lceil [A]_{原} \rceil = 00.1001$, $\lceil [B]_{原} \rceil = 00.1011$, $\lceil -|B| \rceil_{补} = 11.0101$,其运算过程如算式 3.21 所示。

被除数和余数		商					
00.	1001	0.	0	0	0	0	
+ $\lceil - B \rceil_{补}$	11.0101	$\leftarrow 1$	0.	0	0	0	0
	11.1110	R_0					
$\leftarrow 1$	11.1100	$\leftarrow 1$	0.	0	0	0	0
+ $ B $	00.1011	R_1					
	00.0111						
$\leftarrow 1$	00.1110	$\leftarrow 1$	0.	0	0	0	1
+ $\lceil - B \rceil_{补}$	11.0101	R_2					
	00.0011						
$\leftarrow 1$	00.0110	$\leftarrow 1$	0.	0	0	1	1
+ $\lceil - B \rceil_{补}$	11.0101	R_3					
	11.1011						
$\leftarrow 1$	11.0110	R_4	$\leftarrow 1$	0.	1	1	0
+ $ B $	00.1011						
	00.0001						

算式 3.21

故: $[C]_{原} = (A_0 \oplus B_0) + |[C]_{原}| = 1.1101$ 。

而 $|[R_4]_{原}| = 0.0001$, 故: $[R_4]_{原} = A_0 + |[R_4]_{原}| = 1.0001$, $[R_4]_{原真} = 1.00000001$ 。

加减交替法算法步骤规整,省去了恢复余数的过程,加快了除法的速度。因此,原码一位除法一般均采用加减交替法。原码加减交替法的除法器结构如图 3.19 所示。图中实线代表数据传输线,虚线代表控制信号。TSG 产生两种节拍 CP_I 和 CP_{II} , 用以表示执行时间的先后次序,它们的作用与补码一位比较乘法器中相同。

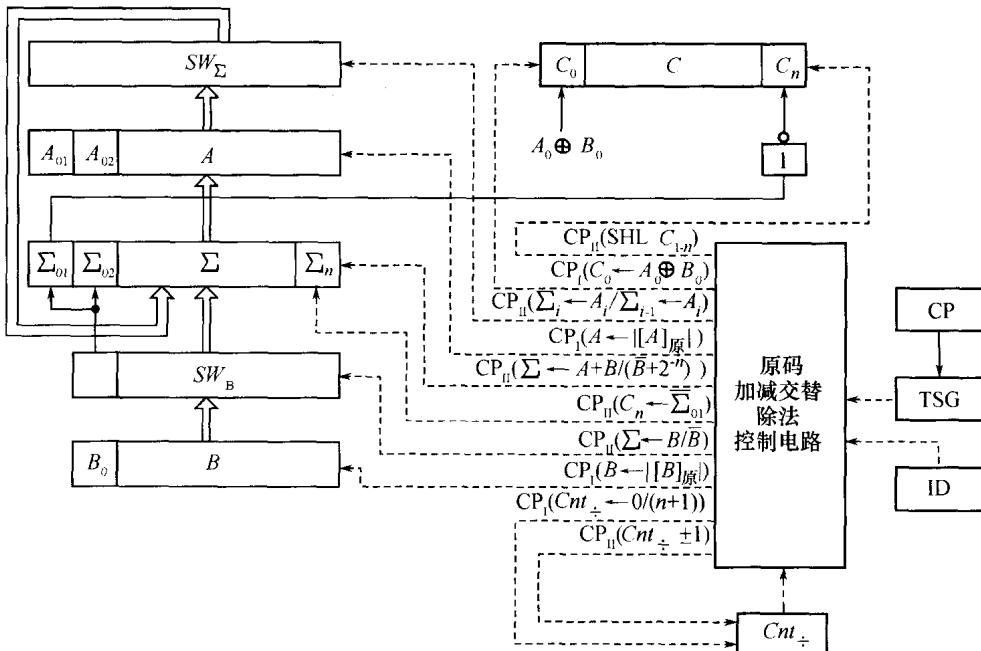


图 3.19 原码加减交替法除法器

图 3.19 中,当指令译码器 ID 发现是一条除法指令时,由原码加减交替除法控制电路完成除法。原码加减交替法除法器各部分功能如下:

A 寄存器: 被除数和余数寄存器,由于被除数设两符号位,故为 $n+2$ 位。初值为被除数 A 的绝对值,即 $A_{01} = A_{02} = 0$ (注意 A_{01}, A_{02} 不一定和被除数 A 的符号位 A_0 相同),除法完成时存放最后的余数。

SW_{Σ} : Σ 的被加数选择开关, $n+2$ 位。除法的第一步完成传送操作: $\Sigma_i \leftarrow A_i$; 其余步完成传送操作 $\Sigma_{i-1} \leftarrow A_i$ 。

Σ: 加法器,因被除数寄存器 A 设两位符号位,故也为 $n+2$ 位。被加数:第一步 $\Sigma_i \leftarrow A_i$, 其余步 $\Sigma_{i-1} \leftarrow A_i$; 作加法时加数: $\Sigma \leftarrow B$; 作减法时加数: $\Sigma \leftarrow \bar{B}$, 同时 Σ 的最低进位输入端加 1(即 $+2^{-n}$)。

B 寄存器: 除数寄存器, $n+1$ 位。存放除数 B 的绝对值,除法过程中保持不变。

SW_B : Σ 的加数选择开关, $n+1$ 位。完成 $\Sigma \leftarrow B$ 或 $\Sigma \leftarrow \bar{B}$ 的转换, 由 C_n 控制。需要注意的是, 除法器中 B 和 SW_B 只设置了一位符号位, B_0 或 \bar{B}_0 被同时送往 Σ_{01} 和 Σ_{02} 。

C 寄存器: 商寄存器, 即 MQ 寄存器, $n+1$ 位。具有左移一位的功能, 左移时, 最后一位移入商(即 $C_n \leftarrow \bar{\Sigma}_{01}$), 但 C_1 不移入 C_0 。商的符号 $C_0 = A_0 \oplus B_0$ (A_0 、 B_0 为被除数和除数的符号位)。

Cnt_z : 除法计数器, 用于标志 $n+1$ 步除法, 每进行一步除法, Cnt_z 加 1 或减 1。

图 3.19 中各寄存器与乘法逻辑中各寄存器通常是公用的, 如 A 寄存器对应乘积寄存器; B 寄存器对应被乘数寄存器; C 寄存器对应乘数寄存器等。 Cnt_z 实际上与 Cnt_x 也是可以公用的, 统称为操作计数器。

3.4.2 补码一位除法

补码除法是指被除数、除数、所求得的商和余数均用补码表示, 连同符号位一起参加运算。两个二进制补码数相除, 如何去比较, 上商和得到新余数呢?

1. 比较

原码一位除法表明, 被除数(或余数)减去除数得到新余数, 如果新余数为正数表示“够减”, 否则表示“不够减”。由于参加补码一位除法运算的是两个带符号的数, 如果两数符号任意, 所谓“够减”指的是被除数(或余数)的绝对值大于等于除数的绝对值, 而“不够减”指的是被除数(或余数)的绝对值小于除数的绝对值。所以, 判断两数大小而进行的比较, 两数同号时应作减法, 两数异号时应作加法。于是, 比较的方法和结果如下:

- 当被除数与除数同号时, 应该用减法去比较二者的大小, 如果得到的新余数与除数同号, 表示“够减”, 否则为“不够减”, 商为正数;
- 当被除数与除数异号时, 应该用加法去比较二者的大小, 如果得到的新余数与除数异号, 表示“够减”, 否则为“不够减”, 商为负数。

为什么使用余数和除数的符号比较, 而不用余数与被除数的符号来判断? 与原码一位除法器的除法过程类似, 每步除法中, 被除数寄存器 A 保存的是新产生的余数, 在整个除法过程中其符号位都可能被修改, 而除数寄存器 B 在整个运算过程中保持不变。因此, 用余数与除数的符号比较来判断是否“够减”是合理可行的。

设 A_{01} 为被除数或余数左移后的符号, B_0 为除数符号, 则上述规则用符号可表示为:

- (1) 当 $A_{01} = B_0$, 计算余数为 $[R_i]_{\text{补}} = \Sigma$:
 - 若 $\Sigma_{01} = B_0$, 表示“够减”(式中 Σ_{01} 可用 Σ_{02} 代替, 因为在没移位之前 $\Sigma_{01} = \Sigma_{02}$)。(下同)
 - 若 $\Sigma_{01} \neq B_0$, 表示“不够减”。
- (2) 当 $A_{01} \neq B_0$, 计算余数为 $[R_i]_{\text{补}} = \Sigma$:
 - 若 $\Sigma_{01} \neq B_0$, 表示“够减”;

- 若 $\Sigma_{01} = B_0$, 表示“不够减”。

2. 上商

有了比较的结果, 即可以上商了。补码除法的商, 自然也应是补码。问题是, 这个商又如何上呢?

除了末位商外, 其余各位商对正商与负商的上商法是不同的。因为在负商的情况下, 任一位商的补码与真值正好是相反的, 故上商必须区别如下:

- (1) 如果商为正(被除数 A 与除数 B 同号):

- 余数与除数同号, 表示“够减”, 商上 1;
- 余数与除数异号, 表示“不够减”, 商上 0。

- (2) 如果商为负(被除数 A 与除数 B 异号):

- 余数与除数异号, 表示“够减”, 商上 0;
- 余数与除数同号, 表示“不够减”, 商上 1。

这样, 负商与正商的上商则可统一起来:

- 余数与除数同号 ($\Sigma_{01} = B_0$), 商上 1;
- 余数与除数异号 ($\Sigma_{01} \neq B_0$), 商上 0。

注意, 该规则实际上是反码上商规则。另外, 上商同时, 商寄存器应左移 1 位。

3. 求新余数

除法进行过程中, 如果某步“够减”, 则产生的余数是真余数; 如果某步“不够减”, 则产生的余数是假余数。如果进行第 i 步除法后, 余数 $[R_i]_b$ 是假余数, 应如何求下步除法的新余数 $[R_{i+1}]_b$ 呢? 下面分两种情况进行分析。

- (1) 除数与被除数(或余数)同号

• 除数与被除数(或余数)都是正数时, 出现假余数也就是余数与除数异号, 即余数为负, 表示“不够减”。类似原码加减交替除法的思想, 此时余数可以不恢复, 将余数左移一位加上除数就可得下步除法的新余数;

• 除数与被除数(或余数)都是负数时, 出现假余数也就是余数与除数异号, 即余数为正, 表示“不够减”。此时余数可以不恢复, 将余数左移一位加上除数也可得下步除法的新余数。

- (2) 除数与被除数(或余数)异号

此时, 出现假余数必是余数与除数同号, 即余数与被除数异号, 表示“不够减”。此时余数可以不恢复, 可将余数左移一位减去除数就得到下步除法的新余数。

综合上面两种情况, 在比较判断“够减”与“不够减”后, 进行如下运算可得新余数:

- 若余数与除数同号, 余数左移一位, 减去除数;
- 若余数与除数异号, 余数左移一位, 加上除数。

如果将求新余数和上商规则统一起来, 可得到表 3.14。

表 3.14 求新余数和上商规则

$[A]_{\text{补}}$ 与 $[B]_{\text{补}}$	$[R_i]_{\text{补}}$ 与 $[B]_{\text{补}}$	上商	下一步操作
同号	同号(够减)	1	$[R_{i+1}]_{\text{补}} = 2[R_i]_{\text{补}} - [B]_{\text{补}}$
	异号(不够减)	0	$[R_{i+1}]_{\text{补}} = 2[R_i]_{\text{补}} + [B]_{\text{补}}$
异号	同号(不够减)	1	$[R_{i+1}]_{\text{补}} = 2[R_i]_{\text{补}} - [B]_{\text{补}}$
	异号(够减)	0	$[R_{i+1}]_{\text{补}} = 2[R_i]_{\text{补}} + [B]_{\text{补}}$

注意:除非 $[R_i]_{\text{补}} = 0$, 必有 $|R_{i+1}| < |B|$, 其中, $i = 0, 1, \dots, n-1$, 即除法一共进行了 $n+1$ 步。

4. 商符的处理

补码除法中, 符号位是要参加运算的, 那么商的符号怎样确定呢? 因是定点除法, 被除数 A 的绝对值必小于除数 B 的绝对值, 否则商就不是定点小数, 商的符号位将被破坏, 发生溢出。

按照一般的方法, 除法第一步的余数 $[R_0]_{\text{补}}$ 是通过被除数 $[A]_{\text{补}}$ 和除数 $[B]_{\text{补}}$ 的比较得到的。由于 A 的绝对值小于 B 的绝对值, 所以:

- 若 $[A]_{\text{补}}$ 与 $[B]_{\text{补}}$ 同号, 形成 $[R_0]_{\text{补}}$ 需要作减法, 商为正;
- 若 $[A]_{\text{补}}$ 与 $[B]_{\text{补}}$ 异号, 形成 $[R_0]_{\text{补}}$ 需要作加法, 商为负。

第一步除法及其上商规则列于表 3.15。

表 3.15 第一步除法及其上商规则

$[A]_{\text{补}}$ 与 $[B]_{\text{补}}$	上商	操作
同号	0	$[R_0]_{\text{补}} = [A]_{\text{补}} - [B]_{\text{补}}$
异号	1	$[R_0]_{\text{补}} = [A]_{\text{补}} + [B]_{\text{补}}$

对比表 3.14 和表 3.15 可以发现, 第一步除法操作对象 ($[A]_{\text{补}}$) 和其后每步除法操作对象 ($2[R_i]_{\text{补}}$) 的形式不一致。按照一般的方法, 除法第一步计算余数的公式为:

$$[R_0]_{\text{补}} = [A]_{\text{补}} + (1 - 2k)[B]_{\text{补}}, \text{ 其中: } k = \begin{cases} 1 & A, B \text{ 同号} \\ 0 & A, B \text{ 异号} \end{cases} \quad (3.36)$$

易知商 $C_0 = 1 - k$, 即同号为正商, 异号为负商。除法第二步计算余数的公式为:

$$[R_1]_{\text{补}} = 2[R_0]_{\text{补}} + (1 - 2C_0)[B]_{\text{补}} = 2[R_0]_{\text{补}} - (1 - 2k)[B]_{\text{补}} \quad (3.37)$$

将式(3.36)代入式(3.37)中, 可得:

$$[R_1]_{\text{补}} = 2[A]_{\text{补}} + (1 - 2k)[B]_{\text{补}}, \text{ 且有 } C_0 = 1 - k \quad (3.38)$$

换个思路, 考虑另一种方法, 如果把 $[A]_{\text{补}}$ 当成是第一步产生的余数, 即: $[R'_0]_{\text{补}} = [A]_{\text{补}}$ 。按照表 3.14 能够得到新余数:

• 当 $[A]_{\text{补}}$ 和 $[B]_{\text{补}}$ 同号, 即 $k=1$ 时, 由于余数与除数同号, 此时的商 $C'_0 = 1 = k$, 需要进行减法得到新余数。即: $[R'_1]_{\text{补}} = 2[R'_0]_{\text{补}} - [B]_{\text{补}} = 2[A]_{\text{补}} + (1-2k)[B]_{\text{补}}$;

• 当 $[A]_{\text{补}}$ 和 $[B]_{\text{补}}$ 异号, 即 $k=0$ 时, 由于余数与除数异号, 此时的商 $C'_0 = 0 = k$, 需要进行加法得到新余数。即: $[R'_1]_{\text{补}} = 2[R'_0]_{\text{补}} + [B]_{\text{补}} = 2[A]_{\text{补}} + (1-2k)[B]_{\text{补}}$;

注意, 这里临时引入 $[R'_0]_{\text{补}}, [R'_1]_{\text{补}}$ 和 C'_0 仅仅为了区别于前一种方法的表示。

可见, 两种情况下新余数的产生公式均为:

$$[R'_1]_{\text{补}} = 2[A]_{\text{补}} + (1-2k)[B]_{\text{补}}, \text{且有 } C'_0 = k \quad (3.39)$$

对比式(3.38)和式(3.39)可知:

$$[R'_1]_{\text{补}} = [R_1]_{\text{补}}, \text{且 } C'_0 = 1 - C_0.$$

这意味着如果把 $[A]_{\text{补}}$ 当成是第一步产生的余数, 按照表 3.14 也能够形成正确的新余数, 这种方法被称为 Booth 除法。Booth 除法中, $[R'_0]_{\text{补}}$ 不需要运算产生, 则除法的运算步数为 n 步。需要说明的是, 从被除数到每一步余数, 均需要和除数进行比较上商, 故商是 $n+1$ 位的。最后一步上商后, 余数不左移。Booth 除法每步除法的操作对象都是一致的, 这样可以降低控制的复杂程度。但是, 由于 $C'_0 = 1 - C_0$, 故除法进行完成后, 需要将第一次上的商取反, 才能得到正确的商符。

5. 恢复余数和修正商

(1) 恢复余数

Booth 除法采用了类似原码加减交替法中的不恢复余数的思想, 最后一步除法得到的余数也有可能是假余数, 需要恢复余数。每步除法得到的余数应与被除数同号, 这是判断新余数是否是真余数的原则。根据这个原则, 当除法除不尽时, 即存在余数 $[R_n]_{\text{补}} \neq 0$ 。

若商为正(被除数与除数同号, $C_0 = 0$), 最后余数与除数异号时, 说明此余数为假余数。显然, 假余数是由于作减法造成的, 为恢复余数应作 $[R_n]_{\text{补}} + [B]_{\text{补}}$ 运算。

若商为负(被除数与除数异号, $C_0 = 1$), 最后余数与除数同号时, 也说明此余数为假余数。显然, 假余数是由于作加法造成的, 为恢复余数应作 $[R_n]_{\text{补}} + [-B]_{\text{补}}$ 运算。

当除法除尽时, 所得到的最后余数应为全 0。所以应设判 0 逻辑线路, 当任何一步除法除尽时(包括最后一步除法除尽)的余数必为全 0, 通过判 0 逻辑判出除尽情况, 除法完成时, 将 A 置成全 0。

综上所述, Booth 除法在作完最后一步除法后, 应按下列规则恢复余数:

- ① 除法除尽时, 判 0 逻辑判出全 0, 将 A 置成全 0;
- ② 除法除不尽时, 判 0 逻辑标志不是全 0;
- ③ 若 $C_0 = 0$, 且余数与除数异号, 应作 $[R_n]_{\text{补}} + [B]_{\text{补}}$, 且将其结果送回 A, 以恢复余数;
- ④ 若 $C_0 = 1$, 且余数与除数同号, 应作 $[R_n]_{\text{补}} + [-B]_{\text{补}}$, 且将其结果送回 A, 以恢复余数。

(2) 修正商

由于上商规则实质上是按反码规则上商的,所以,要得到正确的补码表示的商,就有商的修正问题,简称修商。下面按照除不尽和除尽两种情况来讨论。

① 当除不尽,即 $[R_n]_{\text{补}} \neq 0$ 且任一步 $[R_i]_{\text{补}} \neq 0$ 时,若商为正,商的反码与补码相同,不必修正;若商为负,形成反码商后,应在末位加1,即加 2^{-n} ,才是商的补码。

② 当除尽时,即 $[R_n]_{\text{补}} = 0$ 或任一步 $[R_i]_{\text{补}} = 0$,除尽那步上商,由于除数之正、负不同而不同,需具体分析。

若除数为正($B_0 = 0$),除尽那步除法所得之余数 $[R_i]_{\text{补}} = 0$,其符号位为0,即 $[R_i]_{\text{补}}$ 与除数同号,因此除尽步商为1,下一步除法作减法。

新余数 $[R_{i+1}]_{\text{补}} = 2[R_i]_{\text{补}} + [-B]_{\text{补}} = [-B]_{\text{补}}$ 。 $[R_{i+1}]_{\text{补}}$ 与除数异号,商上0,下一步除法作加法。

新余数 $[R_{i+2}]_{\text{补}} = 2[R_{i+1}]_{\text{补}} + [B]_{\text{补}} = 2[-B]_{\text{补}} + [B]_{\text{补}} = [-B]_{\text{补}}$ 。 $[R_{i+2}]_{\text{补}}$ 仍与除数异号,商仍上0,下一步除法仍需作加法。

以此类推,以后各位商均为0,直到 $[R_n]_{\text{补}} = [-B]_{\text{补}}$ 为止,最后上的商亦为0。可见,除尽且除数为正时,除尽那位商上1,以后各位商上0,商正确,不必修正。

若除数为负($B_0 = 1$),除尽那步除法所得之余数 $[R_i]_{\text{补}} = 0$,其符号位为0,即 $[R_i]_{\text{补}}$ 与除数异号,因此除尽步商为0,此时应修正。应如何修正呢?此时应看除尽步后各位商情况。设 $[R_i]_{\text{补}} = 0$ 为除尽步,那么按规则下一步除法作加法。

新余数 $[R_{i+1}]_{\text{补}} = 2[R_i]_{\text{补}} + [B]_{\text{补}} = [B]_{\text{补}}$ 。 $[R_{i+1}]_{\text{补}}$ 与除数同号,商上1,下一步除法作减法。

新余数 $[R_{i+2}]_{\text{补}} = 2[R_{i+1}]_{\text{补}} + [-B]_{\text{补}} = 2[B]_{\text{补}} + [-B]_{\text{补}} = [B]_{\text{补}}$ 。 $[R_{i+2}]_{\text{补}}$ 仍与除数同号,商仍上1,下一步除法仍需作减法。

以此类推,以后各位商均为1,直到 $[R_n]_{\text{补}} = [B]_{\text{补}}$ 为止,最后上的商亦为1。可见,除尽且除数为负时,不论商为正或负,除尽那位商为0,而后各位商均为1,若加上 2^{-n} ,正好修正成正确的商。

综合上述除不尽和除尽的两种情况,修正商的规则为:

- 若除不尽且商为负时,所得商需加 2^{-n} 修正;
- 若除尽且除数为负时,所得商需加 2^{-n} 修正。

需要指出的是,上述修商规则是比较完善的。有时,为免去判断(是否修商)及修商时的加法,而采用简易修商法。简易修商法是在保证一定商的精度下的近似修商法。常用以下两种方法:

一是“恒置1”法。它是在作除法时,连同符号位,共作 n 步除法,得 n 位商,不作第 $n+1$ 步除法,认为最后一位商恒为1。虽然有时用此法也可能碰巧得到正确商(最后一位商应为1,而又不需修商时),但一般来说得到的是近似商。

二是“0舍1入”法。它是仿照十进制数的四舍五入规则而引入的。具体办法是:采用

多求一位商的办法,即求出 $n+2$ 位商,若 C_{n+1} 为 1, 则 $C_0 \cdot C_1 C_2 \cdots C_n$ 加 2^{-n} 修正; 若 C_{n+1} 为 0, 则 $C_0 \cdot C_1 C_2 \cdots C_n$ 即为所求商。

归纳起来, Booth 除法的详细规则描述为:

(1) 若被除数(或余数)与除数同号, 商为 1, 被除数(或余数)左移一位减去除数得新的余数; 若被除数(或余数)与除数异号, 商为 0, 被除数(或余数)左移一位加上除数得新的余数。

(2) 在新余数基础上, 重复上述操作直至得到所需位数的商($n+1$ 位)为止。最后一步上商后, 余数不左移。

(3) 每步除法上商时, 商寄存器左移一位。

(4) 使商的符号变反。

(5) 最后恢复余数和修正商。

下面举例说明 Booth 除法, 以熟悉其算法规则及书写方法。注意算式中 A_{01} 表示被除数或者余数的符号位。

例 3.20 已知 $A = 0.1001, B = 0.1101$, 求 $[C]_{\text{补}} = ?$, $[2^{-4}R_4]_{\text{补}} = ?$

解: $[A]_{\text{补}} = 00.1001, [B]_{\text{补}} = 00.1101, B_0 = 0, [-B]_{\text{补}} = 11.0011$, 运算过程如算式 3.22 所示。

被除数和余数			商					
$[R_0]_{\text{补}}$	00. 1001	$A_{01} = B_0$	0.	0	0	0	0	0
$\leftarrow 1$	01. 0010		0.	0	0	0	0	1
$+[-B]_{\text{补}}$	11. 0011							
$[R_1]_{\text{补}}$	00. 0101	$A_{01} = B_0$	0.	0	0	0	1	1
$\leftarrow 1$	00. 1010		0.	0	0	1	1	1
$+[-B]_{\text{补}}$	11. 0011							
$[R_2]_{\text{补}}$	11. 1101	$A_{01} \neq B_0$	0.	0	1	1	1	0
$\leftarrow 1$	11. 1010		0.	0	1	1	1	0
$+[B]_{\text{补}}$	00. 1101							
$[R_3]_{\text{补}}$	00. 0111	$A_{01} = B_0$	0.	1	1	0	1	
$\leftarrow 1$	00. 1110		0.	1	1	0	1	
$+[-B]_{\text{补}}$	11. 0011							
$[R_4]_{\text{补}}$	00. 0001	$A_{01} = B_0$	1.	1	0	1	1	
商符变反			0.	1	0	1	1	

算式 3.22

由于未除尽, $C_0 = 0, A_{01} = B_0$, 故不需恢复余数, $[2^{-4}R_4]_{\text{补}} = 0.0001 \times 2^{-4} = 0.00000001$ 。
同时由于未除尽, $C_0 = 0$, 故不需要修正商, $[C]_{\text{补}} = 0.1011$ 。

例 3.21 已知 $A = -0.1001, B = 0.1101$, 求 $[C]_{\text{补}} = ? [2^{-4}R_4]_{\text{补}} = ?$

解: $[A]_{\text{补}} = 11.0111, [B]_{\text{补}} = 00.1101, B_0 = 0, [-B]_{\text{补}} = 11.0011$, 运算过程如算式 3.23 所示。

被除数和余数		商					
$[R_0]_{\text{补}}$	11. 0111	$A_{01} \neq B_0$	0.	0	0	0	0
$\leftarrow 1$	10. 1110		0.	0	0	0	0
$+[B]_{\text{补}}$	00. 1101						
$[R_1]_{\text{补}}$	11. 1011	$A_{01} \neq B_0$					
$\leftarrow 1$	11. 0110		0.	0	0	0	0
$+[B]_{\text{补}}$	00. 1101						
$[R_2]_{\text{补}}$	00. 0011	$A_{01} = B_0$					
$\leftarrow 1$	00. 0110		0.	0	0	0	1
$+[-B]_{\text{补}}$	11. 0011						
$[R_3]_{\text{补}}$	11. 1001	$A_{01} \neq B_0$					
$\leftarrow 1$	11. 0010		0.	0	0	1	0
$+[B]_{\text{补}}$	00. 1101						
$[R_4]_{\text{补}}$	11. 1111	$A_{01} \neq B_0$					
		$\leftarrow 1$	0.	0	1	0	0
		商符变反	1.	0	1	0	0

算式 3.23

由于未除尽, $C_0 = 1, A_{01} \neq B_0$, 所以不需恢复余数, $[2^{-4}R_4]_{\text{补}} = 1.1111 \times 2^{-4} = 1.11111111$ 。

同时由于未除尽, $C_0 = 1$, 故需要修正商, $[C]_{\text{补}} = 1.0100 + 0.0001 = 1.0101$ 。

例 3.20 和例 3.21 表明: 符号位任意的两个补码数相除, 符号位同样参加运算, 结果正确。这是与原码除法的不同之处。

例 3.22 已知 $A = 0.1001, B = -1$, 求 $[C]_{\text{补}} = ? [2^{-4}R_4]_{\text{补}} = ?$

解: $[A]_{\text{补}} = 00.1001, [B]_{\text{补}} = 11.0000, B_0 = 1, [-B]_{\text{补}} = 01.0000$ 。运算过程如算式 3.24 所示。

由于除尽, 故不需要恢复余数, $[2^{-4}R_4]_{\text{补}} = 0.00000000$ 。

同时由于除尽, $B_0 = 1$, 所以需要修正商, $[C]_{\text{补}} = 1.0110 + 0.0001 = 1.0111$ 。

例 3.23 已知 $A = 0.1101, B = 0.1001$, 求 $[C]_{\text{补}} = ? [2^{-4}R_4]_{\text{补}} = ?$

解: $[A]_{\text{补}} = 00.1101, [B]_{\text{补}} = 00.1001, B_0 = 0, [-B]_{\text{补}} = 11.0111$, 运算过程如算式 3.25 所示。此时, $|A| > |B|$, 结果应该溢出。

由于 $\Sigma_{01} \neq \Sigma_{02}$, 表明 $[R_1]_{\text{补}}$ 发生除法溢出, 将溢出标志触发器置 1, 进行相应的中断处理。

被除数和余数		商					
$[R_0]_{\text{补}}$	00. 1001	$A_{01} \neq B_0$	0.	0	0	0	0
$\leftarrow 1$	01. 0010		0.	0	0	0	0
$+[B]_{\text{补}}$	11. 0000						
$[R_1]_{\text{补}}$	00. 0010	$A_{01} \neq B_0$					
$\leftarrow 1$	00. 0100		0.	0	0	0	0
$+[B]_{\text{补}}$	11. 0000						
$[R_2]_{\text{补}}$	11. 0100	$A_{01} = B_0$					
$\leftarrow 1$	10. 1000		0.	0	0	0	1
$+[-B]_{\text{补}}$	01. 0000						
$[R_3]_{\text{补}}$	11. 1000	$A_{01} = B_0$					
$\leftarrow 1$	11. 0000		0.	0	0	1	1
$+[-B]_{\text{补}}$	01. 0000						
$[R_4]_{\text{补}}$	00. 0000	$A_{01} \neq B_0$					
		$\leftarrow 1$	0.	0	1	1	0
		商符变反	1.	0	1	1	0

算式 3.24

被除数和余数		商					
$[R_0]_{\text{补}}$	00. 1101	$A_{01} = B_0$	0.	0	0	0	0
$\leftarrow 1$	01. 1010		0.	0	0	0	1
$+[-B]_{\text{补}}$	11. 0111						
$[R_1]_{\text{补}}$	01. 0001						

算式 3.25

例 3.24 已知 $A = 0.1010, B = 0.1010$, 求 $[C]_{\text{补}} = ? [2^{-4}R_4]_{\text{补}} = ?$

解: $[A]_{\text{补}} = 00.1010, [B]_{\text{补}} = 00.1010, B_0 = 0, [-B]_{\text{补}} = 11.0110$, 运算过程如算式 3.26 所示。此时, $A = B$, 结果应该溢出。

被除数和余数		商					
$[R_0]_{\text{补}}$	00. 1010	$A_{01} = B_0$	0.	0	0	0	0
$\leftarrow 1$	01. 0100		0.	0	0	0	1
$+[-B]_{\text{补}}$	11. 0110						
$[R_1]_{\text{补}}$	00. 1010						

算式 3.26

由于 $[R_1]_{\text{补}} = [B]_{\text{补}}$, 表明 $[R_0]_{\text{补}} = [A]_{\text{补}} = [B]_{\text{补}}$, 此时发生除法溢出, 将溢出标志触发器置 1, 进行相应的中断处理。所以 Booth 除法器的判溢电路的功能还应该包括: 若 $[R_i]_{\text{补}} = [B]_{\text{补}}$, 则除法溢出。当然, 对于例 3.23 和例 3.24 中所示溢出的判断也可以采用其他方法。例如在进行除法运算前, 通过附加逻辑判断条件 “ $|A| < |B|$ ” 是否满足; 若条件不满足, 则置 1 溢出标志触发器, 进行相应的中断处理。

Booth 除法器的逻辑结构如图 3.20 所示。图中实线代表数据传输线, 虚线代表控制信号。TSG 产生三种节拍 CP_1 、 CP_2 和 CP_3 , 用以表示执行时间的先后次序。从结构上而言, 补码比较除法器与原码加减交替法除法器类似, 但决定加减和上商的条件不同。

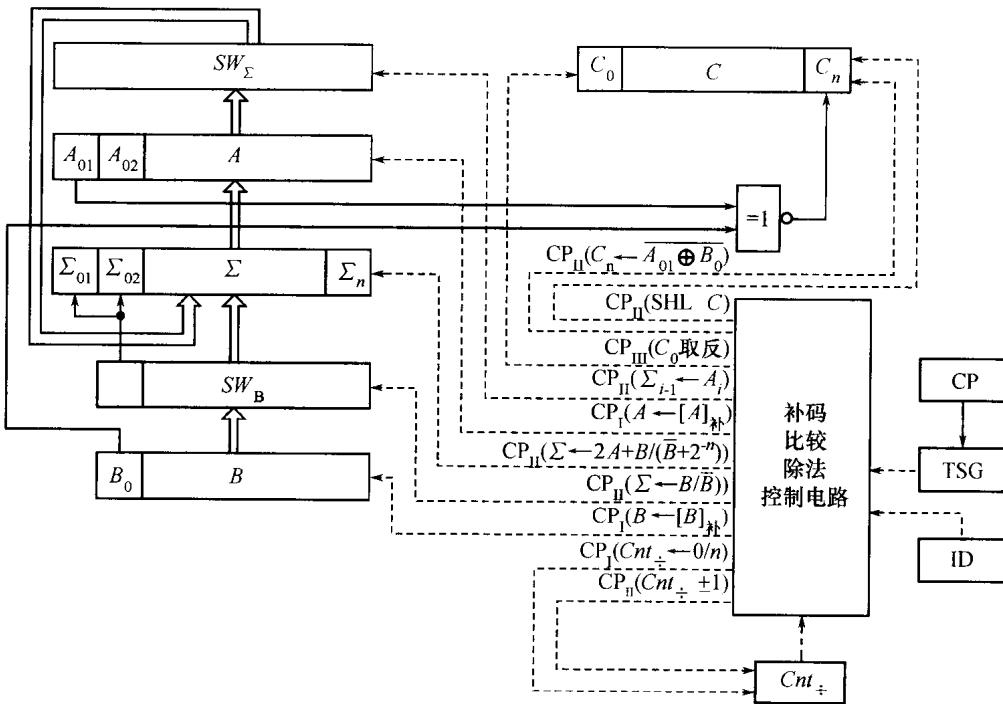


图 3.20 Booth 除法器的逻辑构成

图 3.20 中, 当指令译码器 ID 发现是一条除法指令时, 由补码比较除法控制电路完成除法。Booth 除法器各部分功能如下:

A 寄存器: 被除数和余数寄存器, 由于被除数设两符号位, 故为 $n + 2$ 位。初值为被除数 A 的补码, 除法完成时存放最后的余数。

SW_Σ : Σ 的被加数选择开关, $n + 2$ 位。完成传送操作 $\Sigma_{i-1} \leftarrow A_i$ 。

B 寄存器: 除数寄存器, $n + 1$ 位。存放除数 B 的补码, 除法过程中保持不变。

SW_B : Σ 的加数选择开关, $n + 1$ 位。完成 $\Sigma \leftarrow B$ 或 $\Sigma \leftarrow \bar{B}$ 的转换, 由 C_n 控制。\$B\$ 和 SW_B 只需一位符号位, 同时 SW_B 要送往 Σ_{01} 和 Σ_{02} 。

Σ : 加法器, $n+2$ 位。被加数: $\Sigma_{i-1} \leftarrow A_i$; 加数: $\Sigma \leftarrow B$, 或 $\Sigma \leftarrow \bar{B} + 2^{-n}$ 。

C 寄存器: 商寄存器, 即 MQ 寄存器, $n+1$ 位。具有左移一位的功能, 左移时, 最后一位移入商 ($C_n \leftarrow \overline{A_{01}} \oplus B_0$)。C 左移 $n+1$ 步, 上 $n+1$ 位商。除法最后一步时(由 CP_{III} 控制), 对 C_0 取反。

Cnt_+ : 除法计数器, 用于标志 n 步除法, 每进行一步除法, Cnt_+ 加 1 或减 1。

与图 3.19 类似, 图 3.20 中各寄存器与乘法逻辑中各寄存器通常是公用的, 如 A 寄存器对应乘积寄存器; B 寄存器对应被乘数寄存器; C 寄存器对应乘数寄存器等。 Cnt_+ 实际上与 Cnt_x 也是可以公用的, 统称为操作计数器。

3.4.3 快速除法

与加法、减法及乘法相比, 运算过程中除法指令出现的机会要少得多。因此, 一般机器中仅采用普通的一位除法方案。这样做的好处主要是, 在能够满足机器速度指标的条件下, 可以节省硬件。但在某些高性能计算机(例如巨型机)中对速度的要求更高, 另外机器中已有加减法的快速进位逻辑, 因此如何利用这些硬件提高除法速度, 不但需要, 而且可能。

快速除法通常有三种方法: 一是所谓加(减)除数的跳 1 跳 0 法, 它主要是借助快速加减硬件来完成除法; 二是所谓迭代除法, 它借助快速乘法硬件实现, 因而要将除法转换为乘法; 三是采用阵列除法器, 将逐级递推的各步操作直接在硬件逻辑上级连起来, 在一拍内完成。

加(减)除数的跳 1 跳 0 除法的主要思想是: 当除法某步的余数很小时, 可以连续上多位商, 从而减少除法的步数, 提高除法的速度。有兴趣的读者可参阅参考文献[2], 在此不再赘述。本节中主要介绍迭代除法和阵列除法器两种方法。

1. 迭代除法

迭代除法系以快速乘法为基础的一种快速除法。两数相除, 可以将被除数和除数分别看成一个分数的分子和分母。若分子分母分别乘以某一系列的数 r_i ($i=0, 1, 2, \dots, n$), 使分母迅速收敛于 1, 则分子便以同速度收敛于商, 这样就可使除法转换为乘法。

设被除数为 A , $0 \leq A < B$; 除数为 B , 且 $1/2 \leq B < 1$, 为规格化的数, 则:

$$\frac{A}{B} = \frac{A \times r_0 \times r_1 \times r_2 \times \dots \times r_n}{B \times r_0 \times r_1 \times r_2 \times \dots \times r_n} \quad (3.40)$$

当 $B \times r_0 \times r_1 \times r_2 \times \dots \times r_n \rightarrow 1$ 时, $A \times r_0 \times r_1 \times r_2 \times \dots \times r_n \rightarrow A/B = Q$ 。

其中系列数 $r_0, r_1, r_2, \dots, r_n$ 称之为迭代系数或迭代因子。显然, 问题的关键是如何找到迭代系数 r_i 。很容易想到, 若取初始 r_0 为 B 的近似倒数, 则一个数的近似倒数乘上这个数本身, 其结果必趋近于 1。

如果令:

$$B \times r_0 = 1 - \delta, |\delta| < 2^{-m} \quad (m \text{ 为正整数})$$

则有:

$$r_0 = (1 - \delta) / B, \delta = 1 - B \times r_0.$$

δ 表示第一次迭代后的精度。对其余因子 r_{i+1} , 可以通过 r_i 按照如下方法产生：

$$\left. \begin{array}{l} B_{i+1} = B_i \times r_i \\ A_{i+1} = A_i \times r_i \\ r_{i+1} = 2 - B_{i+1} \end{array} \right\} (i = 0, 1, 2, \dots, n-1) \quad (3.41)$$

式中 $i=0$ 时, $B_0=B, A_0=A$, 于是 B_i, A_i, r_i 可构成如表 3.16 所示数列。不难看出, 经过 n 次迭代, 可以求出 B_n, A_n 。每迭代一次, 分母以平方速度收敛于 1, 分子则以平方速度收敛于商 Q 。

表 3.16 迭代过程及结果

迭代次数	B_i	A_i	r_i	备注
1	$B_1 = B_0 \times r_0 = 1 - \delta$	$A_1 = A_0 \times r_0 = A_0 / B_0 (1 - \delta)$	$r_1 = 2 - B_1 = 1 + \delta$	$i=0, \delta^{2^0} = \delta$
2	$B_2 = B_1 \times r_1 = 1 - \delta^2$	$A_2 = A_1 \times r_1 = A_0 / B_0 (1 - \delta^2)$	$r_2 = 2 - B_2 = 1 + \delta^2$	$i=1, \delta^{2^1} = \delta^2$
3	$B_3 = B_2 \times r_2 = 1 - \delta^4$	$A_3 = A_2 \times r_2 = A_0 / B_0 (1 - \delta^4)$	$r_3 = 2 - B_3 = 1 + \delta^4$	$i=2, \delta^{2^2} = \delta^4$
4	$B_4 = B_3 \times r_3 = 1 - \delta^8$	$A_4 = A_3 \times r_3 = A_0 / B_0 (1 - \delta^8)$	$r_4 = 2 - B_4 = 1 + \delta^8$	$i=3, \delta^{2^3} = \delta^8$
...
n	$B_n = B_{n-1} \times r_{n-1} = 1 - \delta^{2^{n-1}}$	$A_n = A_{n-1} \times r_{n-1} = A_0 / B_0 (1 - \delta^{2^{n-1}})$	$r_n = 2 - B_n = 1 + \delta^{2^{n-1}}$	$i=n-1, \delta^{2^{n-1}}$

因此, 迭代除法的步骤为:

(1) 用求除数 B 的近似倒数法, 可以得到初始迭代系数 r_0 。

(2) 代入公式(3.41), 采用上一次的迭代系数乘以分母并对其进行求补运算的办法, 可以得到以后各次的迭代系数 r_i 。按照此方法可得到所有迭代系数 r_0, r_1, \dots, r_{n-1} 和 r_n 。

(3) n 次迭代系数 r_{n-1} 乘上 A_{n-1} , 即得 n 次迭代除法的近似商 $A_n = A_{n-1} \times r_{n-1}$ 。

(4) 经 n 次迭代后, 迭代精度 $(1 - \delta^{2^{n-1}})$ 已足够达到预期要求。

下面举例说明迭代除法的方法及过程。

例 3.25 设 $m=6$, 根据(3.41)式求各次迭代系数和商。

解: 因为 $m=6$, 故 $\delta=2^{-6}, A_0=A, B_0=B, B$ 的近似倒数为 $r_0=(1-2^{-6})/B$ 。

第一次迭代:

$$A_1 = A_0 \times r_0 = A_0 / B_0 (1 - \delta) = A_0 / B_0 (1 - 2^{-6})$$

$$B_1 = B_0 \times r_0 = 1 - \delta = 1 - 2^{-6}$$

$$r_1 = [-B_1]_{\text{补}} = 1 + \delta = 1 + 2^{-6}$$

第二次迭代:

$$A_2 = A_1 \times r_1 = A_0 / B_0 (1 - 2^{-12})$$

$$\begin{aligned}B_2 &= B_1 \times r_1 = 1 - 2^{-12} \\r_2 &= 1 + 2^{-12}\end{aligned}$$

第三次迭代：

$$\begin{aligned}A_3 &= A_2 \times r_2 = A_0 / B_0 (1 - 2^{-24}) \\B_3 &= B_2 \times r_2 = 1 - 2^{-24} \\r_3 &= 1 + 2^{-24}\end{aligned}$$

第四次迭代：

$$A_4 = A_3 \times r_3 = A_0 / B_0 (1 - 2^{-48})$$

得到商的近似值：

$$Q' = A_4 = A \times r_0 \times r_1 \times r_2 \times r_3$$

因为 $B_4 = B_3 r_3 = 1 - 2^{-48} = B \times r_0 \times r_1 \times r_2 \times r_3$, 故真商为：

$$Q = \frac{A \times r_0 \times r_1 \times r_2 \times r_3}{B \times r_0 \times r_1 \times r_2 \times r_3} = \frac{Q'}{1 - 2^{-48}}$$

故有： $Q' = Q - Q \times 2^{-48}$

显然, 第一次迭代后, 满足初始精度 $1 - 2^{-6}$; 第二次迭代, 误差收敛至 2^{-12} ; 第三次迭代后, 误差收敛到 2^{-24} ; 第四次迭代后, 误差收敛至 2^{-48} , 这已能满足一般的精度要求。

显而易见, 迭代除法问题的关键是如何求得初始迭代系数 r_0 。可以采用造表法, 即构造一个 $B \sim r_0$ 表, 或叫倒数表。对于不同的除数 B , 通过查表得到相应的初始迭代系数 r_0 。 r_0 的精度越高, 则需迭代的次数就越少; 反之, r_0 精度越低, 要达到一定精度的商, 需要迭代的次数就越多。

倒数表可存储在一个只读存储器 ROM 中。若倒数表的容量足够大, 能一次查出倒数 $1/B$, 则 A 乘以 $1/B$ 即可得到商 Q , 称为一次迭代成功。当 B 的位数较长时, ROM 的容量就比较大, 硬件的成本就过高。因此, 一般只设置一个容量有限的 ROM, 第一次只能查得对应有限位数的倒数, 这样当然与 $1/B$ 之间存在一定的误差。

假定倒数表 ROM 的存储容量为 1K 项, 则第一次迭代可查 10 位数字的倒数, 误差 $|\delta| \leq 2^{-10}$ 。第二次迭代后, 误差收敛至 2^{-20} 。第三次迭代后, 误差收敛至 2^{-40} , 显然, 这已能满足一般的要求。

要求得出满足一定精度的商, 怎样在求 r_0 的硬设备和求迭代除本身的硬设备之间综合考虑, 统筹兼顾, 合理安排, 从而充分调动整个运算器的效能, 以获得最佳性能价格比, 这是设计迭代除法的核心问题。

2. 阵列除法器

阵列除法器是仿效阵列乘法器的结构思想, 让各次加减与移位操作以阵列形式在一拍内完成。要做到这一点, 必须解决两个问题; 一是采用可控加减单元; 二是上一级产生的商值能控制下一级的加减操作。

(1) 可控加减单元 CAS

可控加减单元由一个全加单元 Σ 和一个控制加减的异或门组成,如图 3.21 所示。其中, A_i 和 B_i 为本位输入,除数 B_i 还要供给以后各级使用,因此要输往下一级; C_{i-1} 为低位来的进位(或借位)信号; P 为加减控制命令; S_i 为本位和(或差); C_i 为进位信号。

$P=0$ 时, Σ 是加法单元, 实现操作 $A + B$;

$P=1$ 时, Σ 是减法单元, 实现操作 $A - \bar{B}$ 。

(2) 阵列除法器

设: 被除数 $A = 0.A_1 A_2 A_3 A_4 A_5 A_6$; 除数 $B = 0.B_1 B_2 B_3$ 。求商 $Q = 0.Q_1 Q_2 Q_3$ 和余数 $R = 0.r_3 r_4 r_5 r_6$ 。由 CAS 单元构成的阵列除法器如图 3.22 所示。

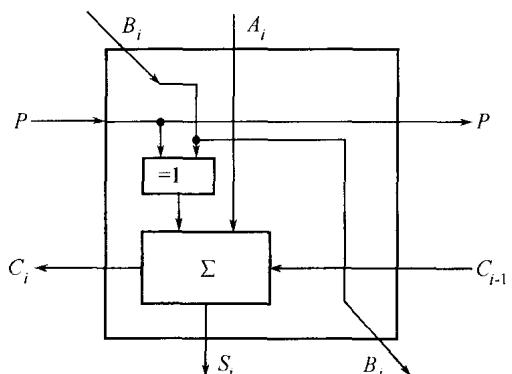


图 3.21 可控加减单元 CAS

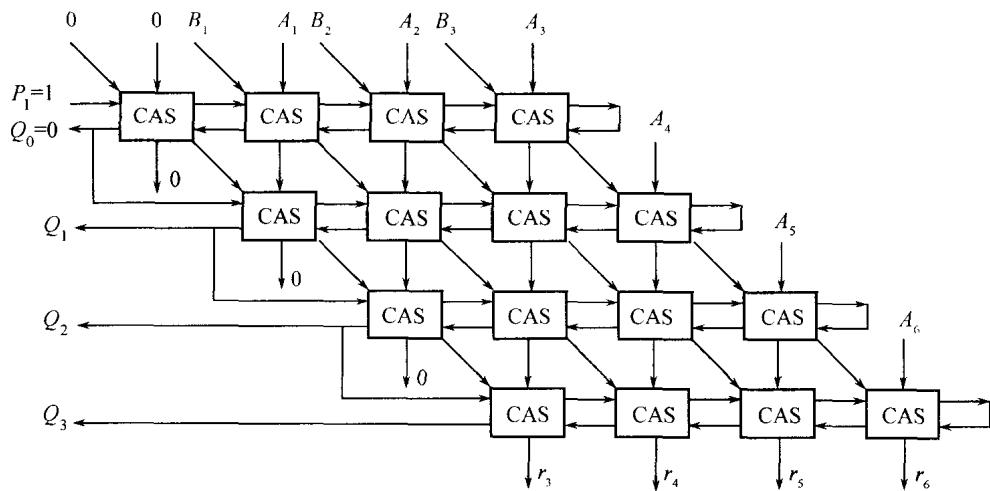


图 3.22 阵列除法器

图 3.22 中阵列除法器有 4 级 CAS, 每一级由控制信号 P_i 选择加或减, P_i 送到这一级的所有 CAS 单元。进位或借位信号则由低位至高位逐级传递, 每一级最后进位即是相应的商值, 同时又作为下一级的控制信号 P_{i+1} 。

由于 A, B 均为正数, 即 A_0 与 B_0 均为 0, 第一行应执行减法操作, 即:

$$0.A_1 A_2 A_3 - 0.B_1 B_2 B_3$$

此时, $P_1=1$, 它同时又是第一行末位的初始进位输入。因 $|A| < |B|$, 故相减后符号位的进位输出为 0, 即商符 $Q_0=0$ 。

第二行 $P_2=0$, 作加法操作, 并补充一位被除数 A_4 。若第二行在高位有进位输出, 相应

商 $Q_1 = 1$, 它又作为下一行的 P , 即 $P_3 = 1$; 若第二行高位无进位输出, 相应 $Q_1 = 0$, 则下一行 $P_3 = 0$ ……依此类推就可以得到商 Q 和余数 R 。

3.5 浮点运算方法

定点数及相应的定点四则运算在科学与工程计算时存在某些困难。例如定点数的表示范围太小, 还有选择、修改和变更比例因子等问题。因此, 在要求可表示的数据范围较大时, 往往采用浮点运算。由于浮点数据表示分为尾数和阶码两部分, 运算时对它们的处理也不相同, 所以浮点运算的硬件成本较高, 完成浮点四则运算时间相比定点运算时间长, 浮点数的格式也较定点数的格式复杂。因此, 从降低代价和复杂度的角度考虑, 计算机的浮点运算功能可以通过软件实现。早期的微型计算机的处理器没有浮点运算功能, 但配有称之为协处理器的浮点部件, 有关浮点运算由协处理器处理, 以提高运算速度。目前, 由于大规模集成电路制造技术的发展, 大部分的计算机的处理器都具有浮点运算功能, 其指令系统中包含有浮点运算指令。

浮点运算可分为非规格化和规格化两类。规格化浮点运算, 只能对规格化浮点数进行操作, 对运算结果要作规格化处理, 而非规格化浮点运算则无上述要求。由于规格化浮点数具有数的表示惟一、编程方便、有效精度高等优点, 故一般都选用规格化的浮点运算。

浮点运算中, 阶码和尾数是分别运算的。由于阶码是定点整数, 尾数是定点纯小数, 因此浮点运算仍可归结为定点运算, 但必须对运算的结果进行规格化。

3.5.1 浮点加(减)运算

规格化浮点加(减)运算可分为五步: 判零, 对阶, 求和, 规格化与舍入。

设两个规格化浮点数分别为:

$$A = M_A \times 2^{E_A}; B = M_B \times 2^{E_B}.$$

其中, M_A 和 M_B 分别为 A 和 B 的尾数, E_A 和 E_B 分别为 A 和 B 的阶码。浮点数的阶码和尾数, 可为原码、补码、反码、移码四种形式之一。

1. 判零

判零即检测操作数是否为零。和定点加(减)运算一样, 当操作数之一为零时, 浮点加(减)运算可以简化操作, 免去不必要的阶码操作。

那么, 怎样判浮点数为零呢? 一种情况是尾数为零, 浮点数为零; 另一种情况是浮点数阶码下溢, 浮点数被作为机器零处理。

2. 对阶

不论阶码和尾数是哪一种码制的机器数形式, 要对两数进行加或减, 必须使小数点对齐后才能进行运算。小数点对齐, 对浮点数来说, 就是使两数的阶码相等。使两数阶码相等的

过程,叫做对阶。对阶的第一步就是求阶差。若阶差用 Δ_E 表示,则应有:

$$\Delta_E = E_A - E_B$$

若 $\Delta_E = 0$,说明两数的阶码相等,不需要对阶;若 $\Delta_E \neq 0$,就需要对阶。对阶时,小阶向大阶看齐。所谓小阶向大阶看齐,就是对阶时将小阶数的尾数右移 Δ_E 位,阶码加 Δ_E 。

例 3.26 设两个浮点数分别为 $A = 0.1101 \times 2^{10}$, $B = -0.1010 \times 2^{11}$, 求此两浮点数之和,要求采用补码数据表示。

解:设浮点数据的格式为:

阶码(补码表示)		尾数(补码表示)	
阶符	阶码	尾符	尾码
2 位	2 位	2 位	4 位

由于采用变形补码,所以阶码和尾数均需要两位符号位。则:

$$[A]_{\text{补}} = 0001,00.1101; [B]_{\text{补}} = 0011,11.0110.$$

故有:

$$[\Delta_E]_{\text{补}} = [E_A]_{\text{补}} - [E_B]_{\text{补}} = 0001 + 1101 = 1110$$

即 $\Delta_E = -2$,表示 E_A 比 E_B 小 2,故对阶时应将 $[A]_{\text{补}}$ 的尾数右移两位,阶码加 2。对阶后:

$$[A]_{\text{补}} = 0011,00.0011$$

此时, $[E_A]_{\text{补}} = [E_B]_{\text{补}}$,即 $[\Delta_E]_{\text{补}} = 0$,对阶完成。

值得注意的是,在此例的对阶过程中, A 的尾数有效位数有所损失,这当然在一定程度上会影响运算的精度。

综上所述,对阶过程可总结为:

- (1) 求阶差;
- (2) 保留大阶作为和的阶;
- (3) 小阶数的尾数右移,同时调整阶差,直到阶差为零为止。

3. 求和

求和就是对尾数进行定点加(减)运算,即:

$$M_A \leftarrow M_A \pm M_B$$

例 3.26 经过对阶后,两数分别变为:

$$[A]_{\text{补}} = 0011,00.0011; [B]_{\text{补}} = 0011,11.0110$$

现将两尾数相加:

$$M_A + M_B = 00.0011 + 11.0110 = 11.1001$$

故 $[A + B]_{\text{补}} = 0011,11.1001$ 。

4. 规格化与判溢出

尾数加减后,若结果不是规格化浮点数,还必须将尾数移位,使之规格化,并相应调整阶

码,这一过程叫做规格化。规格化浮点数的尾数必须满足: $1/2 \leq |M| < 1$ 。

对正的尾数而言,其规格化的定义是使 $1/2 \leq M < 1$ 。即无论采用原码、反码还是补码形式,尾数 M 应是 $00.1 \times \times \dots \times$ 的形式。

对负的尾数而言,若用原码或者反码表示,则 M 应满足: $-1/2 \geq M > -1$ 。即,采用原码时尾数 M 应是 $11.1 \times \times \dots \times$ 的形式,而采用反码时尾数 M 应是 $11.0 \times \times \dots \times$ 的形式。

对负的尾数而言,若用补码表示,则 M 应满足: $-1/2 > M \geq -1$ 。理论上, M 可等于 $-1/2$,但因 $[-1/2]_{\text{补}} = 11.10\dots0$, $[-1]_{\text{补}} = 11.00\dots0$,为便于判断是否规格化,不将 $-1/2$ 列入规格化数,而将 -1 列入规格化数。此时,尾数 M 应是 $11.0 \times \times \dots \times$ 的形式。

综上所述,补码规格化浮点的尾数部分有两种形式: $00.1 \times \times \dots \times$ 和 $11.0 \times \times \dots \times$ 。因此,所谓规格化,对原码数,是使尾数最高有效位为1。对补(反)码数,则依据如下规则:

(1) 左规

若和(差)的两个尾符相同,且与尾数小数点后第一位相同,则需左规,即将和(差)的尾数左移。每左移一位,和(差)的阶码减1,直至尾数小数点后第一位与尾符不同时为止。

工程上,左规的实现,可以每次左移一位,同时将和(差)的阶码减1,直至尾数第一位不同于尾符时结束。也可以通过判断尾数打头1的个数(补码或反码的尾符为1时)或打头0的个数(补码或反码的尾符为0时),控制左移位数,并相应地调整和(差)的阶码。

(2) 右规

若和(差)的两位符号位不等(即为 $01. \times \times \dots \times$ 或 $10. \times \times \dots \times$ 的形式),则需右规,即将和(差)的尾数右移一位,阶码加1。显然,右规最多右移一位。

例 3.26 中,浮点数 A 、 B 的和为:

$$[A + B]_{\text{补}} = 0011,11.1001$$

因和的尾数的两符号位均为1且与尾数的第一位相同,符合左规条件,此时只需左移一位(因尾数前面只有一个打头1),同时阶码减1,于是规格化的浮点数结果为:

$$[A + B]_{\text{补}} = 0010,11.0010$$

值得注意的是:若结果的尾数为0,则不管其阶码为何值,浮点数的值则为0,称此为数量级零。显然,此时不需要规格化。

由于浮点数的表示范围广,浮点加(减)运算溢出的可能性很小。理论上仅有两种极端情况下可能溢出:一是两同号数相加,其中一数的绝对值很大,使正阶码已达到最大值,而相加后又需右规,则尾数右规时阶码增加,有可能上溢,上溢表示数据超出可表示范围需报错处理。二是两异号数相加,两数的绝对值均很小,使负阶码绝对值很大,而相加后又需左规,则尾数左规时阶码减少,有可能下溢,下溢时只需将数据作为零来处理。

5. 舍入

在浮点加(减)法的对阶或右规时,尾数要右移,这样,尾数的末1位或几位可能因超出机器的允许位数而被丢掉,从而造成一定误差。为了减小误差,通常需要进行舍入处理。把

为减小误差而进行的舍入处理，简称为舍入。

舍入处理应注意两个问题。第一是使误差不超过一定的范围。例如尾数为 n 位，要使误差不超过 2^{-n} 。第二是在大量运算过程中，由于随机数不同，要使误差保持平衡。即运算过程中，由于舍入处理，可能使结果增大，也可能使结果减小，增加或减小的机会应力求均等，否则会产生较大的积累误差。

常用的舍入办法有两种：0 舍 1 入法和恒置 1 法。

(1) 0 舍 1 入法

具体做法是类似十进制运算中的“四舍五入”法，在二进制运算中，就是“0 舍 1 入”法。右移时，若丢掉的位数的最高位为 0，则舍去；若丢掉的位数的最高位为 1，则将尾数的末位加 1（相当于舍入）。为此，通过设置专门的寄存器或指定某寄存器来记住要去掉数据的最高位，由它来控制所得尾数最末位是否加 1。

(2) 恒置 1 法

所谓“恒置 1”法，就是只要产生丢失数位的情况，就把尾数的末位置 1，不管末位原来是 1 还是 0。如果原来末位本来为是 1，舍入置 1 后，实际上是把数据丢掉（舍去），出现负误差；如果原来末位是 0，舍入置 1 后，尾数将增大（舍入），出现正误差。如用此种办法，浮点加（减）过程中，凡遇到需对阶或右规时，和（差）的尾数末位均应恒置成 1。

综上所述，浮点加（减）法运算要经过判零、对阶、求和、规格化与判溢出、舍入等步骤，比定点加（减）法要繁琐得多。浮点加（减）法除了应有尾数运算器外，还要有阶码运算器，用来完成求阶差及修改阶码的工作。一般地，阶码运算器只进行加（减）运算。尾数运算器除用以求和外，还应具备左移和右移逻辑，以实现对阶（右移）和规格化（左移或右移）的工作。

3.5.2 浮点乘法运算

浮点乘法是将两个规格化的浮点数相乘。规格化浮点乘法运算可分为四步：判零并置结果数符，阶码相加，尾数相乘，规格化、判溢出与舍入。

设两个规格化浮点数分别为：

$$A = M_A \times 2^{E_A}; B = M_B \times 2^{E_B}.$$

则有：

$$A \times B = (M_A \times M_B) \times 2^{E_A + E_B} \quad (3.42)$$

即乘积的尾数等于两乘数尾数之积，阶码为两乘数阶码之和。

1. 判零并置结果数符

检测操作数是否为零，如操作数中有一个为零，乘积必为零，也就无需做其他操作。只有两个操作数均不为零时，方可进行乘法运算。乘法结果的数符按同号相乘为正，异号相乘为负的规则确定。如果尾数采用补码表示，也可以通过尾数直接运算产生结果的数符。

2. 阶码相加

如果阶码用补码表示,则阶码相加按常规补码加法进行。

如果阶码用移码表示,则因阶码本身已有一个偏移量 2^m (m 为阶码除符号位外的位数),故相加后偏移量将加倍。因此,移码相加后应作修正,即 -2^m 。

阶码相加有可能溢出。同号相加时,若为正阶码,则可能上溢;若为负阶码,则可能下溢。那么,究竟怎样判断阶码的上溢和下溢呢?

(1) 判下溢

产生乘法下溢,有两种可能:一是求乘积的阶码时已下溢,即阶码之和已小于阶的最小值;二是乘积左规时阶码减1而造成下溢。这样,就有一个什么时候判下溢的问题。

那么在求阶码和后判下溢可能会出现什么问题呢?由例3.27说明。

例3.27 已知 $A = -1 \times 2^{-128}$, $B = -1 \times 2^{-1}$,求 $A \times B = ?$

解:在本小节中假定 A 、 B 的阶码与尾数均采用补码表示,阶码取9位(包括两位符号位)。则:

$$[A]_{\text{补}} = 110000000, 11.00\dots0; [B]_{\text{补}} = 111111111, 11.00\dots0.$$

按照运算规则:

$$[A \times B]_{\text{补}} = 101111111, 01.00\dots0.$$

此时,两数阶码之和为101111111(即-129),尾数之积为01.00...0(即+1)。如果在求阶码和后就判溢出,则此时被判为下溢(因为阶码两符号位为10表示下溢)。

但是实际情况如何呢?计算的结果需要右规,即阶码+1,最后阶码为110000000(即-128),尾数为00.10...0。结果没有溢出。

例3.27说明,如果在求阶码和后就判溢出,可能出现本不应为溢出,而被误认为是下溢,错误地扩大了溢出范围。因此正确的做法是应该在规格化时判溢出,才不会扩大溢出范围,但这种作法也有可能造成判断错误,由例3.28说明。

例3.28 已知 $A = 0.5 \times 2^{-128}$, $B = 0.5 \times 2^{-128}$,求 $A \times B = ?$

解:根据题意知:

$$[A]_{\text{补}} = 110000000, 00.10\dots0; [B]_{\text{补}} = 110000000, 00.10\dots0.$$

按照运算规则:

$$[A \times B]_{\text{补}} = 100000000, 00.01\dots0.$$

此时,两数阶码之和为100000000(即-256,阶码两符号位为10,此时应该判断为下溢),尾数之积为00.01...0。计算结果左规后,积的尾数变成00.10...0。阶码需要-1(+111111111):

$$100000000 + 111111111 = 011111111$$

规格化后,由于阶码减去1,阶码两符号位为01,变成了上溢的形式。

例3.28说明,如果在规格化过程中判溢出,就可能把本来是阶下溢错判为阶上溢了。

怎样才能解决这个矛盾呢？一个简单办法是：用求阶码和后的阶码寄存器的最高位（代表阶的符号）来参与控制。

设求阶码和后，阶码寄存器的内容为：

$$R_{E8} R_{E0} R_{E1} R_{E2} R_{E3} R_{E4} R_{E5} R_{E6} R_{E7}$$

左规时，阶码加法器的输出为：

$$\Sigma_{E8} \Sigma_{E0} \Sigma_{E1} \Sigma_{E2} \Sigma_{E3} \Sigma_{E4} \Sigma_{E5} \Sigma_{E6} \Sigma_{E7}$$

当 $R_{E8} = 1$ 且 $\Sigma_{E8} \neq \Sigma_{E0}$ 时下溢，即下溢条件为

$$R_{E8} \cdot (\overline{\Sigma}_{E8} \Sigma_{E0} + \Sigma_{E8} \overline{\Sigma}_{E0}) = R_{E8} \cdot (\Sigma_{E8} \oplus \Sigma_{E0})$$

这样，在规格化时判下溢，既不会扩大溢出范围，也不会错判成上溢了。

判断出下溢后，应相应地将下溢标志触发器置 1，并将下溢中断标志位置 1，以便在规格化之后，将乘法结果清成机器 0。

(2) 判上溢

类似地，产生乘法上溢也有两种可能：一是求乘积的阶码时已上溢，即阶码之和已大于阶的最大值；二是右规时，阶码加 1 而造成上溢。这样，也有一个什么时候判上溢的问题。

如果在求两数阶码之和以后判断，也会使溢出范围扩大。

例 3.29 已知 $A = 0.5 \times 2^{+127}$, $B = 0.5 \times 2^{+1}$, 求 $A \times B = ?$

解：根据题意知：

$$[A]_{\text{补}} = 00111111,00.10\dots0; [B]_{\text{补}} = 00000001,00.10\dots0.$$

按照运算规则：

$$[A \times B]_{\text{补}} = 010000000,00.01\dots0.$$

此时，两数阶码之和为 010000000（即 +128），尾数之积为 0.01...0。如果在求阶码和后就判溢出，则此时被判为上溢（因为阶码两符号位为 01 表示上溢）。

但是实际情况如何呢？计算的结果需要左规，即阶码 -1，最后阶码为 00111111（即 +127），尾数为 00.10...0。结果没有溢出。

例 3.29 说明：如果在求阶码和后就判溢出，就可能出现本不应为溢出，而被误认为是上溢，错误地扩大了溢出范围。

因此，为了不扩大上溢范围，应在规格化后判上溢。此时，并不存在类似下溢时出现的问题，不会把上溢错判为下溢。

例 3.30 已知 $A = -1 \times 2^{+127}$, $B = -1 \times 2^{+127}$, 求 $A \times B = ?$

解：根据题意知：

$$[A]_{\text{补}} = 00111111,11.00\dots0; [B]_{\text{补}} = 00111111,11.00\dots0.$$

按照运算规则：

$$[A \times B]_{\text{补}} = 01111110,01.00\dots0.$$

此时，两数阶码之和为 01111110（即 +254，阶码两符号位为 01，此时判为上溢），尾数

之积为 01.00...0(即 +1)。计算结果满足右规条件,右规后,积的尾数变成 00.10...0。阶码需要 +1,修正为 01111111(即 +255),仍保持上溢的形式,不会错判为下溢。

例 3.30 说明:规格化之后判上溢,并不会有误判,将上溢判为下溢。

但是下溢时已用 $R_{ES} \cdot (\bar{\Sigma}_{ES}\Sigma_{E0} + \Sigma_{ES}\bar{\Sigma}_{E0}) = R_{ES} \cdot (\Sigma_{ES} \oplus \Sigma_{E0})$ 来判断,如果上溢时只用 $\bar{\Sigma}_{ES}\Sigma_{E0}$ 来判断,则类似例 3.28 中出现的下溢就可能误判为上溢。因此只有用类似于判下溢的方法,用求阶码之和后阶码寄存器的最高位 R_{ES} 来参与判上溢,才不会与下溢相混淆,也就不会出错。因此,判上溢的条件也应为:

$$\bar{R}_{ES} \cdot (\bar{\Sigma}_{ES}\Sigma_{E0} + \Sigma_{ES}\bar{\Sigma}_{E0}) = \bar{R}_{ES} \cdot (\Sigma_{ES} \oplus \Sigma_{E0})$$

实际上,只会出现 $\bar{R}_{ES}\bar{\Sigma}_{ES}\Sigma_{E0}$ 的情况,而不会出现 $\bar{R}_{ES}\Sigma_{ES}\bar{\Sigma}_{E0}$ 的情况,只是为使判上溢和判下溢统一用 $\Sigma_{ES} \oplus \Sigma_{E0}$ 逻辑而已。 $\bar{R}_{ES}\Sigma_{ES}\bar{\Sigma}_{E0}$ 不会出现,并不影响逻辑的正确性。

同样,判断出上溢后,也应相应地将上溢标志触发器置 1,并将上溢中断标志位置 1,以便进行上溢中断的处理。

3. 尾数相乘

对不同码制的数,可采用相应码制的定点乘法方案来实现,乘积的尾数等于两个相乘数的尾数之积。

浮点乘法包含两组定点运算,即定点整数的阶码运算和定点小数的尾数运算。两组运算可以分步执行,以便共用一个加法器。通常,浮点运算部件专门设置一个阶码加法器以提高运算速度。

4. 规格化与舍入

浮点乘法也有乘积结果的规格化及相应的舍入等问题。

由于参加运算的数为规格化的数,因而乘积的尾数的绝对值必大于等于 $1/4$,所以乘法的左规,最多只需左规 1 位。由于补码 $[-1]_补$ 是有意义的,所以,当两数尾数都为 $[-1]_补$ 时,尾数相乘后为 01.0...0(即 +1),尾符出现 01 的情况。注意,这是惟一一种尾符出现 01 的情况。因此,浮点补码乘法,还可能有右规的情形。右规时也只能是 1 位。

乘法的舍入,并不产生于补码浮点乘法右规的时候,因此时尾数为 01.00...0,右移 1 位并未丢掉尾数,不需舍入。左规时,向左移位,自然也无舍入问题。那么,乘法的舍入指的是什么呢?乘法的舍入是指当对乘积不取双倍字长的尾数,而取单字长或单字长多几位时,为确保一定乘积精度时的舍入处理。反而言之,乘法如果取双倍字长乘积时,就没有舍入问题。因此,对于定点乘法,如不取双倍字长的乘积,则有舍入问题。通常采用的舍入法,是 0 舍 1 入法。

乘法的 0 舍 1 入法是:当舍掉部分的最高位为 0 时,只截断(乘积的低半部)不舍入;当舍掉部分的最高位为 1 时,则在截断位(即保留位)处加 1。

当乘积取单字长结果时,低半部一个字长全部舍掉,舍掉部分的最高位指低半部字长的首位;截断位指高半部字长的最低位。

下面举例说明浮点乘法的运算过程。

例 3.31 已知 $A = -0.750 \times 2^{-32}$, $B = 0.625 \times 2^{+64}$, 求 $[C]_{\text{补}} = [A \times B]_{\text{补}} = ?$

解: 假定阶码 9 位(包括两位阶符), 尾数采用 5 位(包括两位数符), 据题意知:

$$[A]_{\text{补}} = 111100000, 11.010; [B]_{\text{补}} = 001000000, 00.101。$$

第一步: 判零并置结果数符。

A, B 均不为 0; 由于采用补码表示, 符号位参与运算, 结果数符在尾数相乘时再求得。

第二步: 阶码相加。

$$[E_C]_{\text{补}} = [E_A + E_B]_{\text{补}} = [E_A]_{\text{补}} + [E_B]_{\text{补}} = 000100000。$$

第三步: 尾数相乘。

具体的乘法过程读者可根据前面介绍的定点乘法规则进行, 这里仅给出乘法的结果:

$$[M_C]_{\text{补}} = [M_A \times M_B]_{\text{补}} = 11.100010。$$

第四步: 规格化、判溢出与舍入。

显然, $[M_C]_{\text{补}} = 11.100010$ 不是规格化的形式, 因此需要左规(尾数左移一位, 阶码减 1, $[E_C]_{\text{补}} + [-1]_{\text{补}} = 000011111$)。故:

$$[C]_{\text{补}} = 000011111, 11.000100。$$

本例不存在溢出问题。

如果 C 的尾数也使用 5 位补码表示, 则存在舍入问题。采用 0 舍 1 入法舍入后:

$$[C]_{\text{补}} = 000011111, 11.001。$$

3.5.3 浮点除法运算

浮点除法与浮点乘法类似, 所不同的是用尾数除法代替尾数乘法, 求得商的尾数和得到余数的尾数; 用阶码的减法代替阶码的加法, 求得商的阶码。当符号相反的两数求阶差时, 可能引起阶的上溢或下溢。规格化浮点除法运算可分为四步: 判零并置商符, 尾数调整, 阶码相减, 尾数相除。注意, 除法运算后不需要规格化这一步, 具体原因在“尾数调整”中有详细证明。

设两个规格化浮点数分别为:

$$A = M_A \times 2^{E_A}; B = M_B \times 2^{E_B}.$$

则有:

$$A/B = (M_A / M_B) \times 2^{E_A - E_B} \quad (3.43)$$

即浮点除法运算是尾数相除, 阶码相减。

1. 判零并置商符

检测操作数是否为零, 如果被除数为零而除数不为零, 商和余数均置成零, 除法无需进行。若除数为零, 除法为非法, 应置非法操作标志, 进行中断处理, 除法不应进行。只有两个操作数均不为零时, 除法才能进行下去。除法结果商的数符置位规则与乘法相同。

通常又把判断除数是否为零,被除数是否为零(当除数不为零时)的过程叫做预置。

2. 尾数调整

首先,对尾数来说,当被除数的绝对值大于等于除数的绝对值时(即 $|M_A| \geq |M_B|$),在定点运算中是不允许的,但是在浮点运算中则是允许的。由于两个操作数均为规格化浮点数,故有 $1/2 \leq |M_B| \leq |M_A| < 1 \leq 2|M_B|$,故此时商 M_C 满足: $1 \leq |M_C| < 2$ 。但由于除法规则是在 $|M_A| < |M_B|$ 的前提下推出的,为使定点除法规则在浮点除法尾数相除时也能应用,通常在尾数除法前加上尾数调整的步骤。

所谓尾数调整,是指将被除数尾数调整为小于除数的尾数,即经过调整后被除数的尾数为 M'_A ,应使 $1/2 \leq |M'_A| < |M_B| < 1$ 。以补码为例,按补码除法的比较规则,尾数调整的具体办法是: $M_A \pm M_B = \Sigma$,即同号作减法异号作加法。注意:减法是通过加 $[-M_B]$ 来实现的。

若 $M_{A0} = \Sigma_0$,表示 $|M_A| > |M_B|$,需将 M_A 右移一位得到 M'_A ,且 $E_A + 1$;

若 $M_{A0} \neq \Sigma_0$,表示 $|M_A| < |M_B|$, M_A 保持不变,即 $M_A = M'_A$ 。

注意,作加减后并无 $M_A \leftarrow \Sigma$ 的动作,所以此操作并不改变被除数的尾数。调整以后 $|M'_A| < |M_B|$ 。如 $|M_A| \geq |M_B|$,则 $|M'_A| = |M_A|/2$;如 $|M_A| < |M_B|$,则 $|M'_A| = |M_A|$ 。这样做,不仅使定点除法的规则可以适用于浮点除法的尾数相除,而且所得的商必为规格化的数,即 $1/2 \leq |M'_A|/|M_B| < 1$,因而省去了除法运算后规格化的步骤。

下面分 $|M_A| < |M_B|$ 和 $|M_A| \geq |M_B|$ 两种情况加以证明。

(1) 若 $|M_A| < |M_B|$,则 M_A 不需调整, $|M'_A| = |M_A|$ 。

因为 $|M_A| < |M_B|$,故有: $|M'_A|/|M_B| < 1$ 。

对于 $|M'_A|/|M_B| \geq 1/2$,可以采用反证法证明:

假设 $|M'_A|/|M_B| = m < 1/2$;则有: $|M'_A| = m \times |M_B|$;由于 $|M_B| < 1$,故有: $|M_A| = |M'_A| = m \times |M_B| < m < 1/2$ 。这与 M_A 是规格化数矛盾。所以 $|M'_A|/|M_B| \geq 1/2$ 。

因此, $1/2 \leq |M'_A|/|M_B| < 1$ 。

(2) 若 $|M_A| \geq |M_B|$,则 M_A 需要调整, $|M'_A| = |M_A|/2$ 。此时必有 $|M'_A| < |M_B|$ 。

因为 $|M'_A| < |M_B|$,故有: $|M'_A|/|M_B| < 1$ 。

因为 $|M_A| \geq |M_B|$,故有: $|M_A|/|M_B| \geq 1$;

而 $|M'_A|/|M_B| = (|M_A|/2)/|M_B| = (|M_A|/|M_B|)/2 \geq 1/2$ 。

因此, $1/2 \leq |M'_A|/|M_B| < 1$ 。

综合(1)和(2)两种情况知,所得商必为规格化的数。需要说明的是,这里仅给出了一种证明方法,有兴趣的读者可用其他方法自行证明。

3. 阶码相减

求阶差后,可能产生阶上溢或阶下溢。如阶码用补码表示,则阶符为01表示上溢,应置上溢中断标志;阶符为10表示下溢,使除法结果为机器零。如阶码用变形移码表示,则阶符为10表示上溢,应置上溢中断标志;阶符为11表示下溢,使除法结果为机器零。关于移码

的加减运算可参见 3.2.1 节例 3.8。

余数的阶码与被除数的阶码相同。但在除法过程中,余数左移了 n 位(不含符号位的尾数位数),故真余数的阶码应是被除数的阶码减去 n 。

4. 尾数相除

如果机器设置有专门的阶码运算部件和尾数运算部件,则阶码运算与尾数运算可同时进行,否则需分步执行。尾数相除要用定点小数除法进行。

下面举例说明浮点除法的运算过程。

例 3.32 已知 $A = (5/8) \times 2^{-32}$, $B = (1/2) \times 2^{+64}$, 根据浮点除法规则求 $[C]_{\text{补}}$ 和 $[r]_{\text{补}}$ 。

解:假定 A, B 的阶码与尾数均采用补码表示,阶码取 9 位(包括两位阶符),尾数采用 5 位(包括两位数符),根据题意知:

$$[A]_{\text{补}} = 111100000, 00.101; [B]_{\text{补}} = 001000000, 00.100.$$

第一步:判零并置商符。

A, B 均不为 0;由于采用补码表示,符号位参与运算。

第二步:尾数调整。

由于 $|M_A| \geq |M_B|$,故需要尾数调整, M_A 右移一位, E_A 加 1。调整后:

$$[A]_{\text{补}} = 111100001, 00.010;$$

第三步:阶码相减。

$$[E_C]_{\text{补}} = [E_A]_{\text{补}} - [E_B]_{\text{补}} = 110100001.$$

第四步:尾数相除。具体的除法过程读者可根据前面介绍的定点除法规则进行,这里仅给出除法的结果。

$[M_C]_{\text{补}} = [M_A \div M_B]_{\text{补}} = 00.100$, 商的尾数为规格化的数。所以, $[C]_{\text{补}} = 110100001, 00.100$ 。

由于除尽,故 $[M_r]_{\text{补}} = 00.000$ 。由于尾数的小数部分位数为 3,故 $[E_r]_{\text{补}} = [E_A]_{\text{补}} - [3]_{\text{补}} = 111011110$ 。所以, $[r]_{\text{补}} = 111011110, 00.000$ 。

3.6 运算器组织

运算器在控制器的控制下完成各种操作运算,它不仅可以完成各种算术逻辑运算,还可作为数据的传送通路。那么一般的运算器如何组织呢?运算器的设计,主要是围绕 ALU 与寄存器及数据总线之间怎样传送操作数和运算结果而进行的。

本节首先介绍运算器的基本结构,然后简介位片式运算器和浮点运算器。

3.6.1 运算器的基本结构

一般的运算器主要由以下部件组成:实现基本算术逻辑运算功能的 ALU;提供操作数与

暂存运算结果的寄存器组;有关的判别逻辑(例如判结果是否为0,有无进位,是否溢出等);局部控制电路;以及内部总线等等。

1. 算术/逻辑运算单元 ALU

ALU 的核心是并行加法器及附加的运算函数逻辑,可实现多种算术运算与逻辑运算。在 3.2.3 节中介绍的 SN74181 和 SN74182 就是这种 ALU 的代表。

在图 3.6 所示的 SN74181 中,其控制端 $M, S_0 \sim S_3$ 、末端初始进位等,可以实现 ALU 本身的功能选择。而 ALU 的输出则可经移位器实现,如图 3.23 所示。

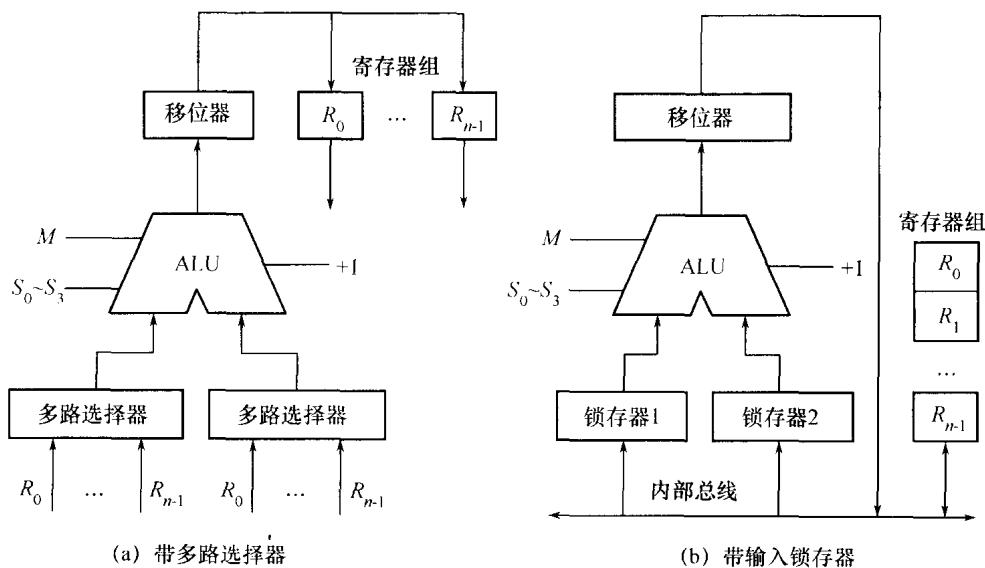


图 3.23 运算器的构成

2. 通用寄存器

通用寄存器主要用来存放操作数和运算的中间结果。在计算机中一般多采用通用寄存器组,以减少对主存的访问次数,提高运算速度,操作也极其灵活方便。采用小容量存储器代替寄存器组也是一种方案,其集成度高,但每次只能有一个输入或输出,若采用双端口存储器,在一定程度上可以弥补其不足。

3. 多路开关/锁存器

ALU 作为数据传送的主要通路,有许多待加工处理或传送的数据都汇集在其入口,因此怎样向 ALU 提供操作数便十分重要。通常有两种提供数据的方法:一是采用多路选择器,在控制信号的作用下,将多个输入信息按需要定时选通,送到 ALU 中去处理;二是在 ALU 输入端使用锁存器,用以暂存数据。

图 3.23(a)所示为带多路选择器的运算器。通过多路选择器既可使 ALU 同时获得多路

数据输入,实现操作数选择,又可通过输入选择实现功能扩展。

带输入锁存器的运算如图 3.23(b) 所示。寄存器组可以是小规模高速存储器结构,每次由控制器发命令可选中其中的某一单元,即相当于选择其中的一个寄存器进行读或写。为了进行双操作数之间的运算,在 ALU 输入端前设置了锁存器,可暂存操作数。例如要实现 $R_0 \leftarrow R_0 + R_1$ 操作,可通过内部总线先将 R_0 的数据送入锁存器 1,再将 R_1 的数据送入锁存器 2,或直接送加法器。然后两者相加,最后经总线将结果送回 R_0 。

4. 移位器

移位器利用输出逻辑对 ALU 的加工结果作进一步的辅助操作,如直传不移位,左移,右移,字节交换等。移位器实质上也是一个多路选择器,如通过斜传可实现移位等。

5. 内部总线

运算器内部各逻辑部件之间广泛采用总线连接。与计算机系统内的系统总线不同,它们只是 CPU 的内部数据通路,故称为内部总线。按内部总线的类型及连接方式,运算器通常有单总线、双总线和三总线三种结构。

(1) 单总线结构运算器

运算器内所有部件均连接到同一总线上,数据可以在任何两个寄存器之间,或在任一个寄存器与 ALU 之间传送。单总线结构的运算器如图 3.24(a) 所示。

单总线结构的运算器在某一时间段内,只能有一个操作数放在总线上。若要将两个操作数输入到 ALU,需分两次将操作数存入 A 和 B 缓冲器。只有两个操作数同时出现在 ALU 的两个输入端,ALU 才能执行相应运算。运算结束后,其结果出现在单总线上,然后执行第三个传送操作,再将结果送入目的寄存器中。显而易见,这种结构的操作速度较慢,但控制比较简单。

(2) 双总线结构运算器

双总线结构运算器如图 3.24(b) 所示。ALU 的两个输入分别由总线 1 和总线 2 供给,故两个操作数可同时送到 ALU 进行运算,只需一次操作控制,立即可得到运算结果。但 ALU 并不能马上将输出结果加到总线上,因为形成操作结果输出时,两条总线均被输入数据所占据。为此,需要在 ALU 的输出端设置缓冲寄存器。操作的控制也要分两步完成:第一步,在 ALU 的两个输入端输入操作数,形成结果并送入缓冲器;第二步,将结果从缓冲器送至目的寄存器。如果在总线 1 与总线 2 和 ALU 之间再各设置一个输入缓冲寄存器,并将两操作数先存放在两个输入缓冲器中,那么 ALU 的输出结果就可以直接送至总线 1 或总线 2 上去。

(3) 三总线结构运算器

ALU 的两个输入端分别由总线 1 和总线 2 供给,而 ALU 的输出直接与总线 3 相连,这就是所谓的三总线结构,如图 3.24(c) 所示。这样,运算可在一步控制中完成。设置总线旁路器的目的是:当某一操作数无需修改,直接由总线 2 送到总线 3,则通过总线旁路器送出,

而不必经过 ALU。当然,如操作数需要修改时,则仍需通过 ALU。三总线结构的操作速度较快,但控制比前两种方式都复杂。

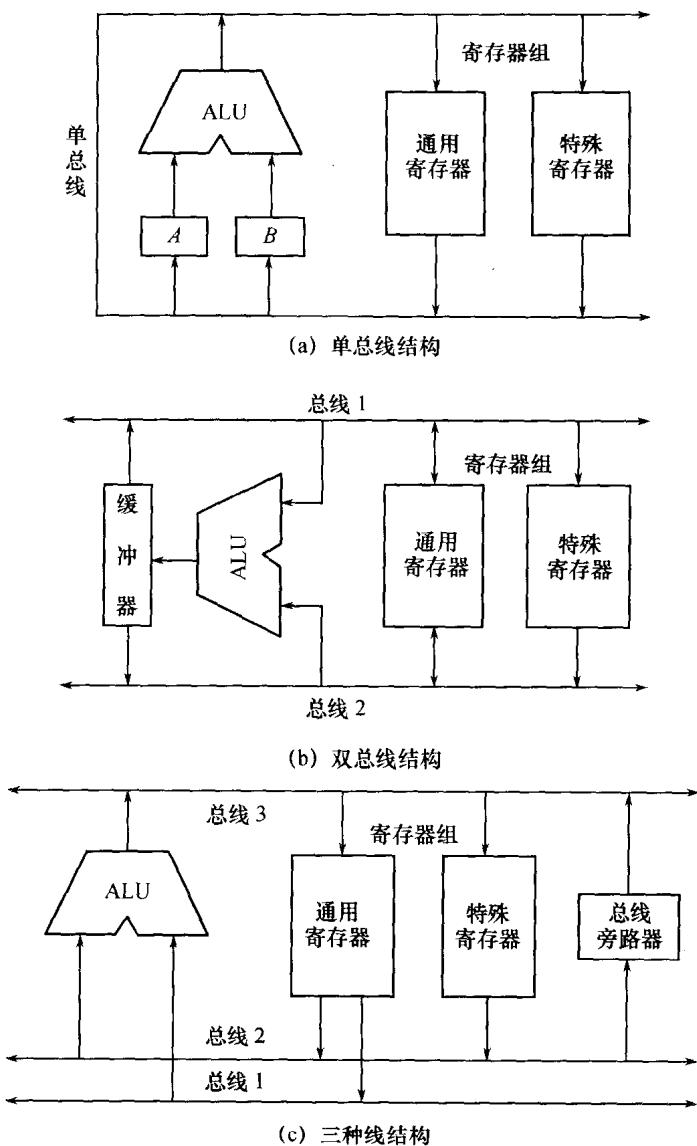


图 3.24 运算器的结构形式

运算器作为 CPU 的主要功能部件,它与计算机的控制器和内存存储器都有着紧密的联系。

运算器与控制器的联系包括:接收控制器发来的各种控制命令,如接收数据命令、运算命令、输出传送命令、通用寄存器读/写命令等等。此外,它还把运算过程中的各种反馈信息送回控制器。

运算器与存储器的信息联系在控制器的控制下完成。例如,向存储器发送写入的数据或接收读出的数据等等。

3.6.2 运算器组成实例

在熟悉运算器的一般构成后,下面以两个实例作进一步的说明。了解这两个运算器的结构组成,还有助于运算器组成实验的操作。

1. 实例 1: 带多路选择器的运算器

一个带多路选择器的运算器实例如图 3.25 所示,它是一个可以实现加减乘除四则运算的并行定点运算器。其执行的基本算术/逻辑运算有: +、-、×、÷、 \wedge 、求补、求反、传送、增 1、加反,还可完成左移、右移、字节交换与结果判零等辅助操作。该运算器主要由 ALU、通用寄存器、多路选择器、移位器、进位寄存器和状态判断逻辑等组成。

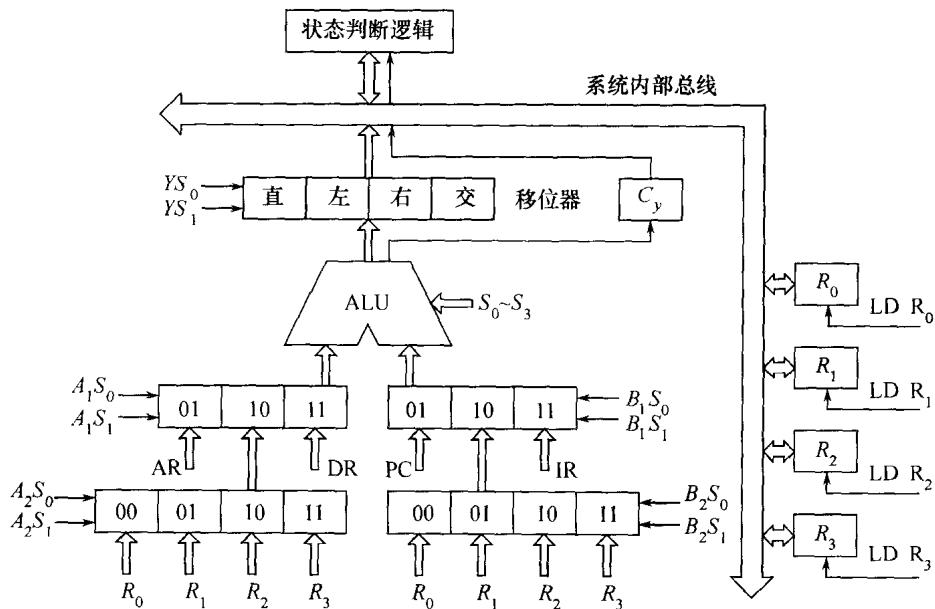


图 3.25 带多路选择器的运算器实例

(1) 算术/逻辑运算单元 ALU

由 4 片 SN74181 和 1 片 SN74182 组成 16 位字长、两级先行进位的 ALU。

(2) 通用寄存器

4 个通用寄存器 \$R_0, R_1, R_2\$ 和 \$R_3\$ 中的任一个既可存放源操作数,也可存放目标操作数。运算的结果可通过系统内部总路线送往任何一个寄存器。4 个寄存器均可作为累加器。\$R_0, R_2\$ 和 \$R_3\$ 仅有寄存功能,而 \$R_1\$ 还具有双向移位功能。因此,\$R_1\$ 在做乘法时可存放乘数,在做除法时可存放商。

(3) 多路选择器

共设 4 个多路选择器: A_1 、 A_2 和 B_1 、 B_2 。 A_1 和 B_1 是 3 选 1 多路选择器。 A_1 可选择来自 $A_2(R_0, R_1, R_2, R_3)$ 、地址寄存器 AR 和数据缓冲寄存器 DR 中的任一路数据,以便送往 ALU 的一个输入端。 B_1 可选择来自于 $B_2(R_0, R_1, R_2, R_3)$ 、指令寄存器 IR 和程序计数器 PC 中的任一路数据,以便送往 ALU 的另一个输入端。AR、IR 和 PC 一般不属于运算器的范畴,但它们的数据可通过 ALU 进行加工处理,故要通过多路选择器被选择,以便将信息输入到 ALU 中。

A_2 和 B_2 是 4 选 1 多路选择器。 A_2 可选择 R_0, R_1, R_2 和 R_3 中的任一个作为源操作数; B_2 则选择它们中的任一个作为目标操作数。通路的选择由 A_2S_0, A_2S_1 或 B_2S_0, B_2S_1 来控制。例如, $A_2S_0 = 0, A_2S_1 = 0$ 时,选择 R_0 ; $A_2S_0 = 1, A_2S_1 = 1$ 时,选择 R_3 等等。

(4) 移位器

它实质上也是 4 选 1 多路选择器。根据 ALU 的 16 位运算结果和 1 位进位值,可实现“循环左移”、“循环右移”、“字节交换”和“直接传送”4 种功能。

(5) 进位寄存器 C_c

它由一个触发器组成,用来寄存每次算术/逻辑运算所形成的最终进位值。在实现双字长运算或乘除法运算中, C_c 起着桥梁作用。

(6) 状态判断逻辑

它可用来判断 ALU 运算结果的状态。例如,用它来判别 ALU 的 16 位运算结果是否为全“0”,以便实现机器指令所规定的操作。

2. 实例 2: 带锁存器的运算器

一个带锁存器的运算器实例如图 3.26 所示。该运算器的基本功能有:加减运算、加 1

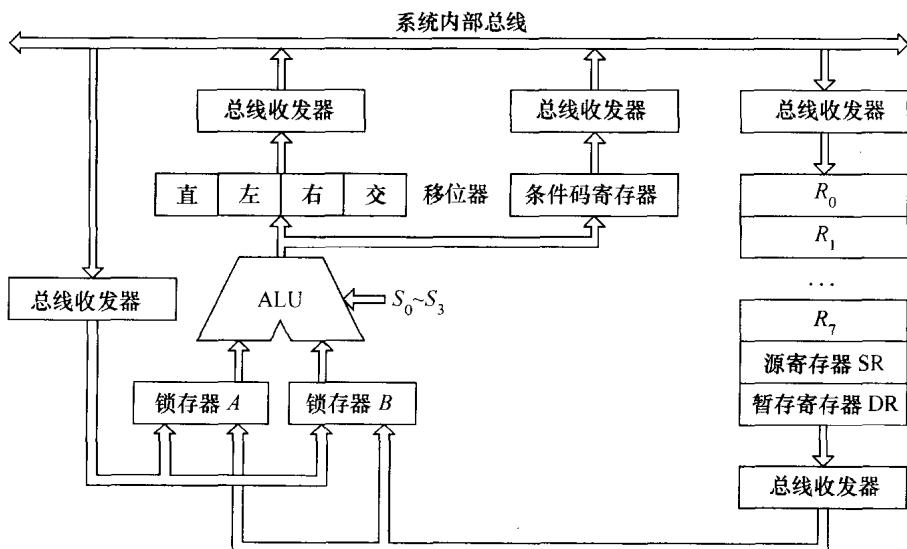


图 3.26 带锁存器的运算器实例

运算、逻辑加、变补、变反传送、以及数的左移、右移、直送和字节交换等操作。

(1) 算术/逻辑运算单元 ALU

它由 4 片 SN74181 和 1 片 SN74182 组成 16 位字长、具有两级先行进位的 ALU。

(2) 锁存器

锁存器 A 和 B 位于 ALU 的两个输入端, 用来暂存来自通用寄存器 $R_0 \sim R_7$ 、源寄存器 SR、暂存寄存器 DR 和外部设备及存储器的数据。数据一旦存入锁存器, 则不论其外部的数据如何变化, ALU 都将依据 A 和 B 中的数据进行运算。

(3) 移位器

它的作用和实例 1 相同, 可将 ALU 的输出进行左移、右移、直接传送和字节交换。因此它也是一个 4 选 1 的多路选择器, 每次只能从 4 路输入中选择 1 路进行输出。

(4) 寄存器组

寄存器组共包括 8 个可由程序编址的通用寄存器 $R_0 \sim R_7$ 和两个程序不能访问的工作寄存器——源寄存器 SR 及暂存寄存器 DR。通用寄存器可作为累加器, 其中的数据经锁存器 A 和 B 送入 ALU, 再经移位器送系统内部总线, 而后经总线收发器送通用寄存器、存储器或外部设备, 从而实现数据的输出操作。反之, 来自存储器或外部设备的数据, 经总线三态接收门送至锁存器, 再经 ALU 和移位器, 送到通用寄存器, 从而实现数据的输入操作。

为什么要设置一个源寄存器 SR 来暂存数据呢? 因该计算机设有双操作数指令, 即一条指令中同时有两个操作数。在执行双操作数指令时, 必须两次计算操作数地址, 两次从存储器取操作数。第一个操作数从总线传来不能马上放在锁存器中, 因为紧接着可能要使用数据通路(当然也包括锁存器)进行第二个操作数地址的寻址计算。因此, 设置一个程序看不见的源寄存器, 用以暂存第一个操作数。待第二个操作数取出来后, 再从源寄存器中读出第一个操作数, 送往锁存器和 ALU 进行运算。

暂存寄存器 DR 用来暂存 ALU 计算出来的结果数据, 以等待总线传出去。计算结果出来后, CPU 申请总线控制权, 再将 DR 中的数据经数据通路和三态发送门, 发送到数据总线上去。

在任一时刻, 寄存器组的 10 个寄存器只有一个可与数据通路发生联系, 究竟哪一个寄存器连接到数据通路, 则由控制器发出的寄存器地址加以确定。

(5) 条件码寄存器

运算器在运算过程中得到的“进位”、“溢出”、“零”和“负”等状态标志可保存在条件码寄存器中, 以供程序判断使用。该寄存器的状态标志信息也可经总线送到存储器加以保存。

3.6.3 位片式运算器

利用大规模集成电路 LSI 技术, 可将 ALU 与寄存器等集成成为一种位片式结构。用几块位片拼接, 可构成位数较多的运算器。代表性的位片有 4 位/片, 如 AMD2900/29000/29300

系列。

AMD2900 系列位片的结构组成如图 3.27 所示。

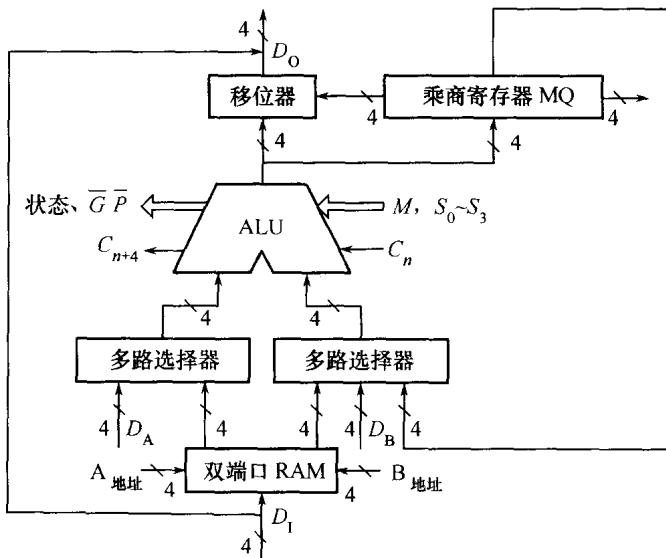


图 3.27 位片式运算器

ALU 的逻辑结构类似于 SN74181,但扩展了功能,可实现乘除运算。其功能控制信号有 $M, S_0 \sim S_3$ 等,有进位输入 C_n ,进位输出 C_{n+4} ,进位辅助函数 \bar{G}, \bar{P} ,此外还有输出的某些状态信息。

多路选择器实现 ALU 的输入选择,它的信息可来源于通用寄存器组、外部直接输入 D_A 和 D_B 以及乘商寄存器 MQ。

RAM 是一个双端口随机存储器。所谓双端口,是可同时向它送入两个地址 A 和 B ,因而可同时选中两个寄存器。RAM 组成一个 16×4 位的通用寄存器组,即相当于有 16 个寄存器,每个寄存器 4 位。同时选中的两个寄存器各自的 4 位数据,经多路选择器送往 ALU 处理。

位片的数据输入端为 D_1 ,移位器的输出即位片的输出端为 D_o 。

3.6.4 浮点运算器

根据浮点运算算法,浮点运算器显然包括阶码运算部件和尾数运算部件两部分,如图 3.28(a)所示,它们是两个松散连结的定点运算部件。

1. 阶码运算部件

阶码运算部件的功能包括阶码大小的比较、阶码相加或相减、阶码调整时的增量与减量。在图 3.28(a)中,操作数的阶码部分放在寄存器的 E_1 和 E_2 中,它们与并行加法器 Σ 相

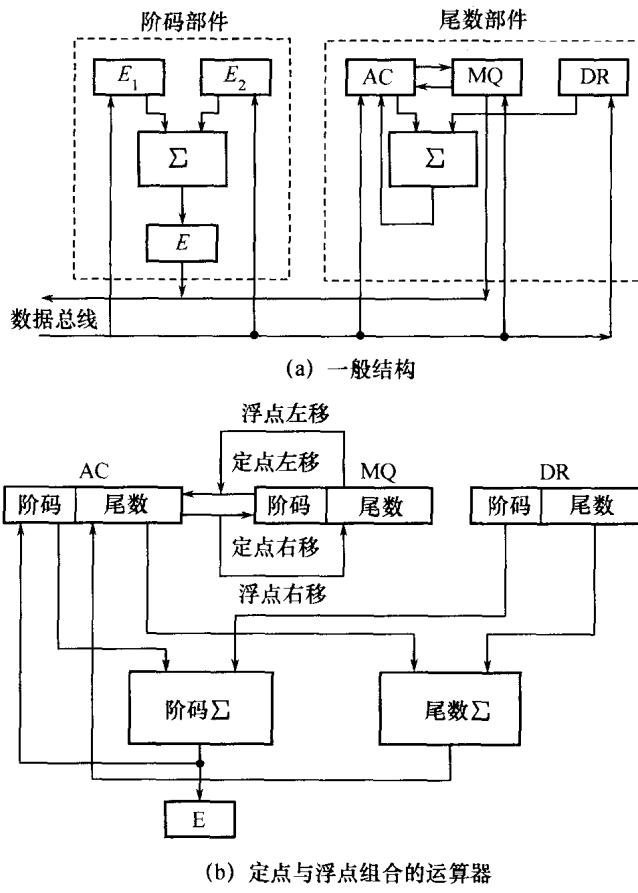


图 3.28 浮点运算器

连,以便计算 $E_1 \pm E_2$ 。浮点加法和减法所需要的阶码比较是通过 $E_1 - E_2$ 来实现的,相减的结果放入计数器 E 中,然后按 E 的符号决定哪一个阶码较大。在尾数相加或相减之前,需要将其中的一个尾数移位,这是由计数器 E 来控制的。 E 每减一次 1,相应的尾数移 1 位,直到使 E 的值减到零为止。一旦尾数调整完毕,就可按通常的定点运算进行处理,运算结果的阶码值仍放到计数器 E 中。

2. 尾数运算部件

尾数运算部件是一个通用的定点小数运算器,它能够实现左移、右移和加、减、乘、除四种基本算术运算。在图 3.28(a)中,设置 3 个单字长的寄存器用来存放操作数,其中有累加器 AC,乘商寄存器 MQ 和数据寄存器 DR。AC 和 MQ 可组成左右移位的双字长寄存器 ACMQ。并行加法器 Σ 用来完成数据的加工处理,其输入分别来自 AC 和 DR,而结果送回 AC。MQ 在乘法时存放乘数,除法时存放商,故称之为乘商寄存器。DR 用来存放被乘数或除数。乘积或商与余数则存放在 ACMQ 中。在四则运算中,使用这些寄存器的典型方法如

表 3.17 所示。

表 3.17 尾数部件寄存器的使用

运 算	寄存器关系
加 法	$AC \leftarrow AC + DR$
减 法	$AC \leftarrow AC - DR$
乘 法	$ACMQ \leftarrow DR \times MQ$
除 法	$ACMQ \leftarrow AC \div DR$

3. 定点和浮点组合的运算器

图 3.28(a)的结构只能用于 n 位浮点运算。然而几乎所有具有浮点运算指令的机器也都具有定点运算指令,因此,一般要求运算器既能执行浮点运算指令,也能执行定点运算指令。图 3.28(b)为定点与浮点组合的运算器结构,它基本上是一个单字长的定点算术运算器。不过,在执行浮点运算时,其寄存器应分为阶码和尾数两部分,加法器亦应分为阶码加法器和尾数加法器两部分。

4. 浮点协处理器

在早期的微机中,常将浮点运算部件与相应控制部件集成为一块芯片,称之为浮点协处理器,成为一种可以增选的部件。例如浮点协处理器 Intel 8087,可配合 Intel 8086/8088 处理器使用;浮点协处理器 Intel 80387,可配合 Intel 80386 处理器使用。而 486 和 586 等高档微机则已将浮点处理单元(Floating Point Unit, FPU)集成于 CPU 处理器芯片中。

8087 的内部结构如图 3.29 所示,它主要分为控制部件 CU 和数值处理部件 NEU 两部分。

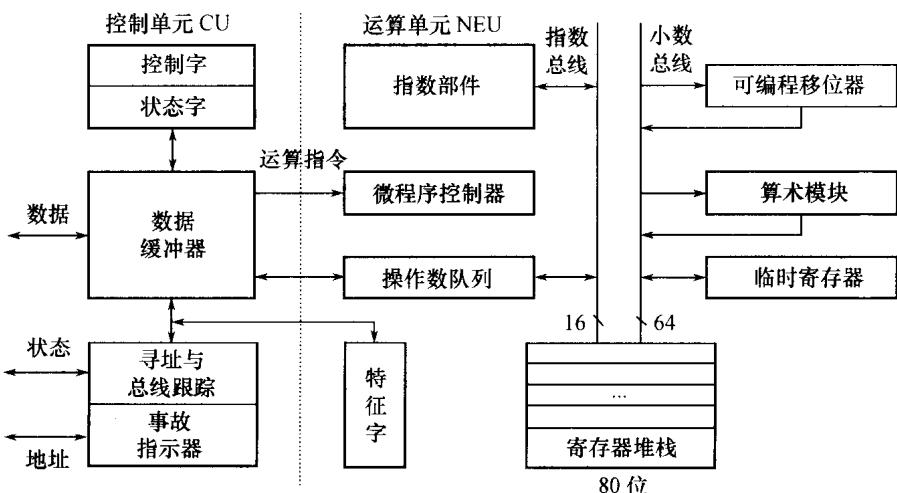


图 3.29 8087 内部结构

NEU 中有 8 个 80 位的寄存器, 构成寄存器堆栈, 其中的数据均用临时实数的形式表示。当各种数据从存储器传送到寄存器栈之前, 自动变换成 80 位的临时实数形式; 反之, 把寄存器栈的值保存到存储器时, 又可以变换成数据的任何一种格式。

8087 本身不能单独使用, 而只能作为 8086 的“协作伙伴”运行。

8087 拥有几十条算术运算指令和函数指令, 通常称之为 ESC 指令。用 8086 和 8087 的指令编写的程序均存放于主存, 它们都由 8086 读取。如读取的是 8086 的指令, 则在 8086 的内部加以处理; 如读取的是 8087 的指令, 则 8086 如同对 I/O 端口输出数据那样, 把这条指令的代码送往 8087。

8087 对 8086 送来的指令进行译码并予以执行。8087 执行指令和 8086 的工作完全是非同步进行的, 因此, 在 8087 执行数值运算期间, 8086 有可能执行下一条指令。

为防止 8087 在执行指令时, 8086 又送来新的 ESC 指令, 通常用 BUSY 信号加以控制。只有 BUSY 是高电平时, 8086 才能将 ESC 指令送给 8087; 如果 BUSY 是低电平, 则在其变为高电平之前, 8086 不能把 ESC 指令代码送给 8087。

8087 作为选配件供给用户。若无 8087 时, 系统中对浮点运算则要由 8086 用软件进行处理。采用了 8087 后, 由于浮点运算无需 8086 处理, 因而大大改善了系统的性能, 特别在大量计算时更是如此。

小 结

本章讨论的是各种算术运算和逻辑运算的运算规则。计算机的运算均可分解成算术(四则)运算与逻辑运算, 四则运算的核心是加法运算。ALU 的实现与运算方法密切相关, 而运算方法与数据的编码有关。本章中分别介绍了定点数据的加减法、一位乘法、一位除法、快速乘除法以及对应的运算部件。在定点数据加工的基础上, 分别对尾数和阶码进行处理就可以实现浮点数据的运算。

实现了基本算术逻辑运算功能的 ALU 以后, 提供操作数与暂存运算结果的寄存器组, 增加有关的判别逻辑, 局部控制电路以及内部总线等就可以构成实际的运算器。

习 题

3.1 解释下列术语:

逻辑运算	算术运算	逻辑移位	算术移位
半加器	全加器	串行进位	并行进位
并行进位加法器	进位产生函数	进位传递函数	通用函数发生器
存储进位加法器	对阶	规格化	左规
右规	舍入		

- 3.2 已知 $A = 1001, B = 0101$, 试计算:
- $C \leftarrow A \wedge B$
 - $C \leftarrow A \vee B$
 - $C \leftarrow A \oplus B$
 - $C \leftarrow A \odot B$
- 3.3 按操作性质, 移位有几种类型? 各有何特点?
- 3.4 已知 A 和 B , 求 $[A + B]_b$ 和 $[A - B]_b$:
- $A = 0.1011, B = -0.1110$
 - $A = -0.1101, B = -0.1010$
 - $A = 0.1101, B = 0.0001$
 - $A = 0.1110, B = 0.0010$
- 3.5 已知 A 和 B , 求 $[A + B]_b$ 和 $[A - B]_b$:
- $A = 0.1011, B = -0.0010$
 - $A = -0.1101, B = -0.1010$
 - $A = -0.1001, B = 0.1101$
 - $A = 0.1101, B = 0.1011$
- 3.6 设计一个 9 位的先行进位加法器, 要求每 3 位为一组, 采用两级先行进位线路。
- 3.7 采用 SN74181 ALU 和 SN74182 器件, 构成一个三级的 64 位先行进位的 ALU。
- 3.8 已知 A 和 B , 用 Booth 乘法求 $[A \times B]_b$ 和 $[B \times A]_b$:
- $A = 0.11011, B = -0.11110$
 - $A = -0.1011, B = -0.0101$
 - $A = -1, B = -1$
 - $A = -01011, B = 11011$
- 3.9 已知 A 和 B , 用补码两位乘法求 $[A \times B]_b$ 和 $[B \times A]_b$:
- $A = 0.10110, B = -0.00011$
 - $A = -0.011010, B = -0.011101$
 - $A = -1, B = -0.11010$
 - $A = 01011, B = -11011$
- 3.10 补码定点乘法是否会溢出? 若溢出, 何时溢出?
- 3.11 证明补码定点一位乘法任一部分积的绝对值小于等于 1。
- 3.12 用流程图分别描述补码一位定点乘法和补码两位定点乘法算法流程。
- 3.13 补码两位定点乘法在工程实践上要注意哪些问题? 与补码一位定点乘法相比要注意什么?
- 3.14 以两位定点乘法为基础, 试推导 3 位补码定点乘法的算法。
- 3.15 已知 A 和 B , 用 Booth 除法求 $[C]_b$ 和 $[2^{-n}R_n]_b$, 并恢复余数和修正商:
- $A = 0.10010, B = -1$
 - $A = 0.10101, B = 0.11011$
 - $A = -0.01011, B = -0.11011$
 - $A = 0.10100, B = -0.10000$

- 3.16 两定点数相除,如何判断商是否溢出?若溢出应注意什么(与不溢出的正常除法比较)?
- 3.17 迭代除法建立在什么基础之上?为何具有生命力?
- 3.18 初始迭代系数 r_0 原则上应如何确定?
- 3.19 迭代除法的执行步骤怎样进行?以4次迭代为例,图示并叙述之。
- 3.20 按补码浮点加(减)法的运算步骤,求 $[A \pm B]_{\text{补}} = ?$
- (1) $A = 2^{-001} \times (0.10100), B = 2^{-010} \times (-0.011110)$
- (2) $A = 2^{-010} \times (-0.1010), B = 2^{-010} \times (-0.011111)$
- (3) $A = 2^{-011} \times (-0.0100), B = 2^{-010} \times (0.010110)$
- 3.21 浮点加(减)法为什么要对阶?为什么要小阶向大阶看齐?何时需要舍入处理?何时判溢出?为什么?
- 3.22 按规格化的补码浮点乘法运算步骤,求 $[A \times B]_{\text{补}}$ (设阶码为3位,尾数为6位,均不包括符号位)。
- (1) $A = 2^{011} \times (0.110100), B = 2^{100} \times (-0.100100)$
- (2) $A = 2^{-011} \times (-0.000110), B = 2^{101} \times (-0.011100)$
- 3.23 浮点乘法应注意哪些问题?何时判下溢和上溢?为什么?
- 3.24 试述补码浮点乘法的运算步骤?若尾数用原码表示,阶码用移码表示,运算步骤上有什么区别?
- 3.25 按规格化补码浮点除法规则,求 $[A \div B]_{\text{补}}$ (设阶码为3位,尾数为6位,均不包括符号位)。
- (1) $A = 2^{-2} \times (13/32), B = 2^3 \times (15/16)$
- (2) $A = 2^3 \times (-11/16), B = 2^5 \times (15/16)$
- (3) $A = 2^{-6} \times (-1), B = 2^{-7} \times (1/2)$
- (4) $A = 2^{-7} \times (-1), B = 2^1 \times (-1)$
- 3.26 试述浮点除法的运算步骤。为何除法进行前要经过预置和尾数调整两个步骤?
- 3.27 用流程图分别描述浮点加减运算、浮点乘法运算和浮点除法运算流程。
- 3.28 乘法和除法浮点运算均可由软件实现,软硬件实现各有何优缺点?
- 3.29 试就数的范围、精度、程序设计、硬件设计等几方面对定点运算和浮点运算进行比较,说明它们的优缺点。
- 3.30 试比较单总线、双总线、三总线结构运算器的优缺点。

第四章 控 制 器

引 言

控制器是存储程序结构计算机的五大组成部分之一,是计算机工作的指挥和控制核心。分析控制器的工作必然涉及到计算机的各个部件,了解控制器的工作过程也就了解了计算机全部工作的过程。因此,本章的学习对于建立整机的概念是至关重要的。

控制器又称为控制电路,是计算机中处理指令的部件。它按序取出并分析程序的每一条指令,然后产生一系列使得运算器和其他部件进行操作的控制信号,计算机各部件就是在控制器的这些控制信号控制下协调工作,一步步地完成指令规定的工作。在各部件间不断流动的指令和数据形成指令流(Instruction Stream)和数据流(Data Stream)。计算机执行的指令序列称之为指令流;根据指令要求依次访问的数据序列称为数据流。指令流和数据流都是程序执行过程中动态的概念。从程序运行的角度来看,控制器的基本功能是:在时间和空间上,对指令流和数据流实施正确控制。空间上,由控制器形成受控部件的控制信号;时间上,控制器控制各种动作的执行顺序。由控制器发出的控制信号序列称为控制流(Control Stream)。

具体来说,从系统设计和实现的角度来看,控制器完成的是以下对指令流的控制任务:

- 控制取指令字(控制指令流出);
- 解释指令字(控制指令分析);
- 组织计算机各个部件动作的信号序列,以完成指令的执行(控制指令执行);
- 确定下一条指令的地址(控制指令流向);
- 执行环境的建立与保护(控制执行环境的维护)。执行环境指的是指令执行时所占用的全部资源的状态,包括功能部件、特征寄存器以及机器工作方式等的状态。

控制器对数据流的控制,主要应包括对数据流出与流入(存、取)的控制;对数据变换、加工等操作的控制。对冯·诺依曼型结构的计算机而言,它的数据流是根据指令流的操作而形成的,也就是说数据流是由指令流来驱动的。

作为冯·诺依曼型计算机的核心控制部件,控制器能够控制存储器、运算器和输入/输出部件,因此控制器需要产生对这些部件的控制信号序列。本章将要讨论的内容就包括这些控制信号产生的时间和方法,这也是控制器设计的主要工作。

组成控制器的基本功能部件,如指令寄存器 IR、指令译码器 ID、程序计数器 PC、变址寄存器 XR 以及地址运算部件等,对于任何类型的控制器都是必不可少的,而且设计方法比较成熟。但是,对于控制信号的产生和控制部件的组成方式存在着两种不同的方法,即组合逻辑控制器和微程序控制器。它们对整个控制器的组成,时标系统的设计等影响很大。

本章首先介绍一个简单的计算机(下面称之为实例计算机)的指令系统,然后通过它来研究控制器的设计方法。这个设计实例将由浅入深,从内部逐渐扩展到外部,对组合逻辑控制器的基本组成、工作原理和设计中的问题进行较全面的讨论。最后将研究微程序控制器设计的有关概念和设计问题。

4.1 指令结构

对于存储程序计算机而言,指令和数据存放在同一个存储器中,只有当程序开始在计算机上执行时,存储器存储的内容是指令还是数据才能确定。假设每个指令字占用一个存储单元,则指令流如图 4.1 所示。图中的停机指令并不是所有计算机都有的。

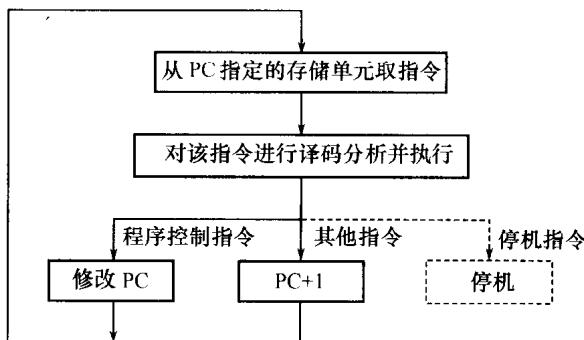


图 4.1 指令流的产生和控制过程

计算机启动后,将从一个规定的存储单元开始取指令并执行,然后顺序执行下去,只有停机(HALT)或者程序控制(转移 BRANCH 或 JUMP、转子 CALL 等)指令才打断或者改变指令执行顺序。如何根据指令流产生控制流是控制器完成的主要工作,而控制信号则是由控制器对指令字进行分析处理后产生的。

正如第二章所述,指令字通常由操作码和地址两个字段组成:操作码字段用于定义指令的操作,地址字段用于指示指令所操作对象的位置。一条指令的地址字段可以包含若干个地址。指令格式还可以分为固定长度指令格式与可变长度指令格式两种。RISC 计算机多采用固定长度指令格式,其指令通常为 4 个字节(32 位)或者 8 个字节(64 位)。CISC 计算机通常不是固定长度指令格式,其指令格式比 RISC 指令格式要复杂。为讨论方便,本章中假定实例计算机采用固定长度指令格式,指令均占用一个存储单元。

4.1.1 实例计算机的指令系统

指令系统包含三个方面的内容:数据表示、指令以及寻址方式。本章将要设计的实例计算机是一个很简单的8位计算机。

计算机的指令涉及指令的功能以及寄存器和存储器的结构。实例计算机设置了一个8位的累加器(Accumulator, AC)和一个8位的栈顶指针(Stack Pointer, SP)。实例计算机的指令为8位单地址指令,采用可变长度操作码,有两种指令格式:基本1地址指令和扩展1地址指令,它们的指令字结构分别如图4.2和图4.3所示。

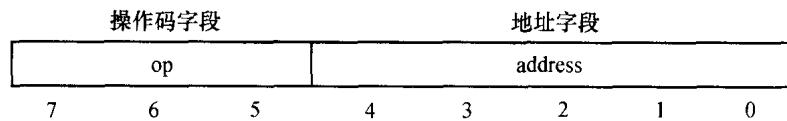


图 4.2 实例计算机基本 1 地址指令的指令字结构

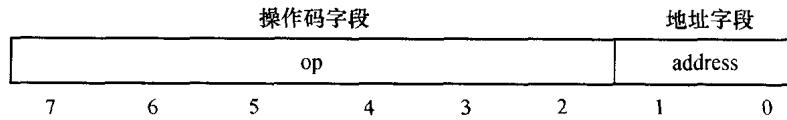


图 4.3 实例计算机扩展 1 地址指令的指令字结构

实例计算机的指令如表4.1所示。表4.1中的指令分为三类:基本1地址指令(例如ADD指令)、扩展0地址指令(例如RETURN指令)和扩展1地址指令(例如SHIFTR指令)。其中,基本1地址指令为存储器访问指令,扩展1指令地址为非存储器访问指令。

表 4.1 实例计算机的指令

指令符	功能	操作码	指令类型	汇编记忆码
ADD	加法	000	1 地址指令	ADD address
SUBTRACT	减法	001	1 地址指令	SUBTRACT address
STORE	存储	010	1 地址指令	STORE address
LOAD	读出	011	1 地址指令	LOAD address
BRANCH	条件转移	100	1 地址指令	BRANCH address
CALL	转子	101	1 地址指令	CALL address
CLEAR	清累加器	111000	0 地址指令	CLEAR
RETURN	子程序返回	111001	0 地址指令	RETURN
SHIFTR	右移	111101	1 地址指令	SHIFTR counter
SHIFTL	左移	111110	1 地址指令	SHIFTL counter

从图 4.2 和表 4.1 可知, 实例计算机的存储器地址为 5 位, 所以其存储器空间为 $2^5 = 32$ 个存储单元。为简单起见, 实例计算机的指令、数据以及堆栈共享该存储空间。不失一般性, 实例计算机将栈顶指针 SP 的内容初始化为 00011111B, 即指向最后一个存储单元。每次压栈前, SP 减 1; 每次弹栈后, SP 加 1。

实例计算机的数据表示只有最简单的二进制 8 位无符号整数形式。

实例计算机的寻址方式主要是直接寻址(基本 1 地址指令)和立即数寻址(扩展 1 地址指令)。实际上, 由于累加器 AC 和栈顶的地址都是隐含的, 因此实例计算机的寻址方式也包含隐含寻址。

4.1.2 实例计算机指令的含义

表 4.1 给出了实例计算机的 10 条指令, 下面分别介绍这些指令的含义。

1. 加法指令(ADD)

加法指令的格式是:

操作码字段			地址字段				
0	0	0	address				
7	6	5	4	3	2	1	0

加法指令是一条基本 1 地址指令, 指令的功能是完成累加器 AC 和存储器指定单元内容的加操作, 其和存放到累加器 AC 中:

$$AC \leftarrow AC + \text{Memory}(address) \bmod 2^8;$$

2. 减法指令(SUBTRACT)

减法指令的格式是:

操作码字段			地址字段				
0	0	1	address				
7	6	5	4	3	2	1	0

减法指令是一条基本 1 地址指令, 指令的功能是完成累加器 AC 中存放的被减数与存储器指定单元内容存放的减数的减操作, 其差存放到累加器 AC 中:

$$AC \leftarrow AC - \text{Memory}(address) \bmod 2^8;$$

3. 存储指令(STORE)

存储指令的格式是:

操作码字段			地址字段				
0	1	0	address				
7	6	5	4	3	2	1	0

存储指令是一条基本 1 地址指令, 指令的功能是完成存储操作, 将累加器 AC 中的数据送入存储器指定单元中:

$\text{Memory}(\text{address}) \leftarrow \text{AC};$

4. 读出指令(LOAD)

读出指令的格式是:

操作码字段			地址字段				
0	1	1	address				
7	6	5	4	3	2	1	0

读出指令是一条基本 1 地址指令, 指令的功能是完成对 AC 的加载操作, 从存储器指定单元中读出数据送入累加器 AC 中:

$\text{AC} \leftarrow \text{Memory}(\text{address});$

5. 条件转移指令(BRANCH)

条件转移指令的格式是:

操作码字段			地址字段				
1	0	0	address				
7	6	5	4	3	2	1	0

条件转移指令是一条基本 1 地址指令, 指令的功能是控制指令流。转移的条件是累加器 AC 的最高位 (AC_7), 若 AC_7 为 '1', 计算机将该指令地址字段中的地址送入程序计数器 PC, 执行指定存储器地址单元中的指令, 否则顺序执行下一条指令:

```
IF ( $AC_7 = '1'$ ) THEN  $PC \leftarrow address;$ 
ELSE  $PC \leftarrow PC + 1;$ 
```

6. 转子指令(CALL)

转子指令的格式是:

操作码字段			地址字段				
1	0	1	address				
7	6	5	4	3	2	1	0

转子指令是一条基本 1 地址指令, 简称转子, 指令的功能是实现转入子程序的操作。实例计算机执行转子指令时, 首先将当前程序计数器 PC 的内容保存到栈顶指针 SP 指示的单元中, 然后将指令地址字段的内容送程序计数器 PC:

```
 $SP \leftarrow SP - 1;$ 
 $\text{Memory}(SP) \leftarrow PC;$ 
 $PC \leftarrow address;$ 
```

7. 清累加器指令 (CLEAR)

清累加器指令的格式是：

操作码字段						地址字段		
1	1	1	0	0	0	x	x	
7	6	5	4	3	2	1	0	

清累加器指令是一条扩展 0 地址指令，地址字段无意义。指令的功能是完成对累加器 AC 的清 0 操作：

AC←0；

8. 子程序返回指令 (RETURN)

子程序返回指令的格式是：

操作码字段						地址字段		
1	1	1	0	0	1	x	x	
7	6	5	4	3	2	1	0	

子程序返回指令是一条扩展 0 地址指令，地址字段无意义。指令的功能是从栈顶指针 SP 指示的存储单元中取出返回地址送程序计数器 PC，完成子程序返回调用程序：

PC←Memory(SP)；

SP←SP + 1；

9. 右移指令 (SHIFTR)

右移指令的格式是：

操作码字段						地址字段		
1	1	1	1	0	1	counter		
7	6	5	4	3	2	1	0	

右移指令是扩展的 1 地址指令，其功能是按指令地址字段的数值，对累加器 AC 右移：

```
again; while not(counter = '00') loop
    AC(0:6)←AC(1:7);
    AC7←0;
    counter←counter - 1;
end loop again;
```

10. 左移指令 (SHIFTL)

左移指令的格式是：

操作码字段						地址字段		
1 1 1 1 1 0						counter		
7	6	5	4	3	2	1	0	

左移指令是扩展的 1 地址指令,其功能是按指令地址字段的数值,对累加器 AC 左移:

```
again:while not( counter = '00' ) loop
    AC(1:7)←AC(0:6);
    AC0←0;
    counter←counter - 1;
end loop again;
```

不难看出,实例计算机的指令系统是可以进一步扩展的。给出了实例计算机的指令系统后,以后各节将通过实例计算机的实现,来讨论计算机控制器的设计技术,逐步完成实例计算机的逻辑设计。

实例计算机指令系统包括 10 条指令,其控制器的设计分阶段进行。首先研究加法、减法、存储、读出以及清累加器这 5 条指令的实现;然后研究如何实现对程序流向的控制,包括条件转移、转子和子程序返回指令的实现;最后针对移位指令这种执行时间不确定的指令,研究如何实现这一类指令的控制。

4.2 控制器的基本设计技术

本节首先通过 5 条指令的实现来讨论控制器的基本设计技术。

4.2.1 控制器的基本组织

程序的执行过程是取指令、分析指令和控制执行指令的过程,换句话说就是控制器从存储器中取指令、访问数据和加工数据的操作序列。这个序列是由指令序列,即指令流来完成的。指令流由控制流控制完成,控制流中包含了控制指令执行的控制信号序列。

1. 指令周期

一条指令的执行时间,即从取指令开始到指令执行完成所需要的时间,称为指令周期。一般情况下,指令周期中包括取指令、分析指令、执行指令等工作。不同的指令,操作性质不同,执行时间的长短不同。所以,不同的指令有不同的指令周期。

根据每个阶段完成工作的不同,指令周期可以分为取指周期(Instruction Fetch, IF)、执行周期(Execution, EX)、间址周期(Indirect)和中断周期(Interruption)等。取指周期完成的主要工作就是根据程序计数器 PC 的值从存储器中取得指令,执行周期的工作就是完成指令规定的操作,所有指令周期均包含取指周期和执行周期。间址周期是使用间接寻址方式时指令周期的组成部分。中断周期则是当有中断请求时,中止当前指令执行,进行中断响应的

时间。

实例计算机中,指令的操作序列是取指周期和执行周期相交替的过程。取指周期中,控制器根据 PC 从存储器中取出指令,然后对指令进行译码,将程序计数器 PC 加 1,为程序顺序执行提供下条指令的地址。执行周期中,控制器根据操作数地址从存储器中取出操作数,并按指令要求完成相应的操作,或者将结果回写(包括送存储器)。实例计算机指令周期的构成以及完成的操作如图 4.4 所示。



图 4.4 指令周期、取指周期、执行周期和对应的操作

需要说明的是,这里所说的存储器指的是内存储器,内存储器的访问时间较短,并且是固定的。

2. 控制寄存器

依第一章可知,控制器主要由指令控制部件、地址形成部件、定时部件及微操作控制部件组成,如图 1.4 所示。这些部件中存在一些必不可少的基本部件,特别是一些控制寄存器。图 4.5 中阴影部分给出了这些寄存器。

下面分别对这些寄存器的功能予以说明:

- 程序计数器 PC,用于指示指令在存储器中的地址。PC 具有清零、加 1 以及接收 IR 地址字段的功能。PC 可以指示指令的执行顺序。实例计算机中,在取指周期结束前,PC 完成“加 1”的功能,指出下一条指令的地址。如果遇到程序控制指令(转移 BRANCH、转子 CALL 指令等),则在执行周期中将 IR 中的地址部分送 PC。当程序启动时,清 PC 为全 0(假定首条指令地址为全 0)。对于其他计算机来说,程序启动时,PC 可能被预置成某个特定值,以便计算机在这个特定的位置启动。
- 指令寄存器 IR,用于保存现行指令。指令从存储器中取出以后,要送入指令寄存器 IR 中,以便进行指令分析。IR 一般与指令字长度相同。
- 存储器地址寄存器 MAR,用于保存访问存储器的地址。
- 存储器缓冲寄存器 MBR,用于暂存从存储器中读出或需要写入存储器的数据。
- 读触发器 R(Read),用于标志对存储器进行读操作。需要从存储器读出数据时,将其置位。
- 写触发器 W(Write),用于标志对存储器进行写操作。当 MBR 中有数据需要写入存

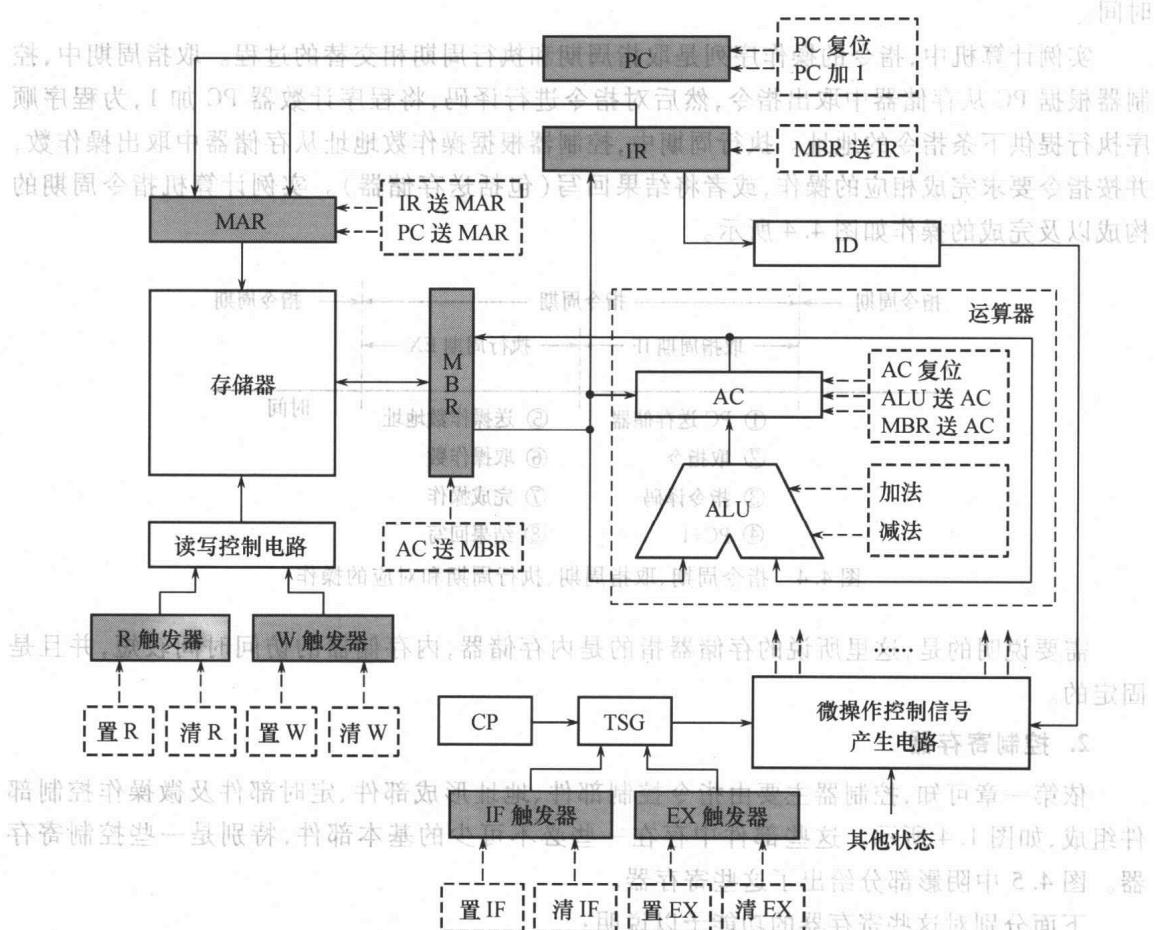


图 4.59 运算器、存储器和处理 5 条指令的控制器

当 CPU 需要访问存储器时，将其置位。

- 取指令触发器 (Instruction Fetch, IF)。IF 置位时表示计算机处于取指周期。由于程序执行时，将首先进入取指周期，故机器启动时，IF 被初始化为 1。

- 执行指令触发器 (Execution, EX)。EX 置位时表示计算机处于执行周期。由于程序执行时，取指周期结束后再进入执行周期，故机器启动时，EX 被初始化为 0。

读触发器 R、写触发器 W、取指令触发器 IF 和执行指令触发器 EX 分别描述存储器访问的工作状态和指令执行不同阶段的状态，因此它们又被称为控制器状态触发器。状态触发器不但用于控制器的控制流生成，还是处理器设计中重要的参考信号标志。复杂的控制器，具有更多状态触发器。例如计算机为实现中断系统、内存直接存取 DMA 以及间接寻址方式就需要设置中断周期触发器、DMA 周期触发器、间址周期触发器以及其他的状态触发器。

输入主要由外部总线 RAM、外部总线数据总线、地址总线、控制总线 (ctrl)、时钟总线 (clock)、复位总线 (reset)、电源总线 (power) 等组成。

3. 对控制寄存器操作的控制信号

对于所有控制寄存器,需要相应的控制信号对它们进行操作。图 4.5 中对于每个控制寄存器画出了相应的一系列控制信号,由虚线箭头表示,这些信号及其功能分别是:

- “PC 复位”:将程序计数器清 0。有些计算机系统在 PC 复位时将它设为某个特定值。需要说明的是,该操作仅在系统复位时使用。

- “PC 加 1”:用于 PC 值加 1 操作。这种操作使得 PC 指向当前指令的下一条指令在存储器中的地址。

- “MBR 送 IR”:将 MBR 的内容送到 IR。即从存储器中取得指令。

- “IR 送 MAR”:将 IR 中地址字段的内容送到 MAR 中去。

- “PC 送 MAR”:将 PC 的值送到 MAR 中。

- “清 R”和“置 R”:完成 R 触发器的置位复位操作,用于对存储器的读操作。对存储器的读操作结束时,发出“清 R”信号;需要读存储器时,需要先发出“置 R”信号。

- “清 W”和“置 W”:完成 W 触发器的置位复位操作,用于对存储器的写操作。它们的功能与“清 R”和“置 R”相似。

- “清 IF”和“置 IF”:完成 IF 触发器的置位复位操作,用于控制取指周期的时间。取指周期开始时,需要发出“置 IF”信号;取指周期结束时,发出“清 IF”信号。

- “清 EX”和“置 EX”:完成 EX 触发器的置位复位操作,用于控制执行周期的时间。它们的功能与“清 IF”和“置 IF”相似。

4. 与运算器有关的控制信号

图 4.5 中还给出了其他一些控制信号,这些控制信号与运算器有关,它们的功能分别是:

- “AC 复位”:将累加器 AC 的内容清为全 0。

- “ALU 送 AC”:将 ALU 的计算结果送 AC。

- “加法”和“减法”:控制 ALU 进行加法和减法运算。

- “AC 送 MBR”:将累加器 AC 的内容送 MBR。

- “MBR 送 AC”:将 MBR 的内容送累加器 AC。

4.2.2 基本控制器的设计

基本控制器的设计问题包括如何形成和连接受控部件的控制信号,以及在何时使这些控制信号有效。本节将通过实例计算机中 5 条指令的实现来说明基本控制器的设计问题。

1. 微操作

计算机的工作过程就是运行程序的过程,也就是在控制器的控制下逐条执行程序中指令的过程。指令的执行,实际上包含一系列的控制过程。在这个控制过程中,由控制器发出的每个控制信号均通过硬连线的方式连接到某个特定部件,在空间上形成受控部件的操作

控制信号。同时,这些特定部件的操作是有时间关系的,控制信号需要在时间上控制各种动作的先后顺序。因此,指令执行过程,就是对多个部件的一系列操作的控制过程。

相对指令完成的功能而言,一个部件能够完成的基本操作,称为微操作(Micro-operation,记为 Micro - OP 或 μ OP)。微操作是组合逻辑控制器设计中一个十分重要的概念,计算机中所有的操作最终均可以分解为不同微操作的序列。从部件的角度看,这些操作不能继续细分。因此,微操作是计算机中最基本的操作,是最小的具有独立意义的操作。

对于 ALU 来说,由于操作数存放在寄存器中,故微操作就是针对寄存器中的数据进行的操作,包括寄存器传输微操作、算术微操作、逻辑微操作以及移位微操作等。这些微操作的特点是通常能够在一个节拍内完成,3.1 节中介绍的基本运算就属于微操作的概念。而对于其他部件来说,操作时间较长的基本操作(例如访存操作)也被认为是微操作。

一条指令的执行过程就是一个微操作序列的操作过程。控制器对指令的控制过程就转换为对指令所需微操作的控制过程。不同的指令对应着不同的微操作序列。控制器对指令流与数据流的控制就是对这些微操作序列进行的控制。当然,每个计算机系统中包含的微操作可以是不同的,但是这些微操作的集合所完成的功能应该是相同或相似的。计算机的功能设计均以微操作为基础,正确、合理地分析和设置微操作,是设计控制器的前提。

2. 控制方式

指令的执行过程要求微操作是有序操作的。在数字系统设计中,对于有序操作的控制通常分为异步控制方式和同步控制方式两种实现方法。

(1) 异步控制方式

采用异步控制方式的系统各个部件之间没有统一的时钟,各部件有自己的时钟,有的异步控制甚至不需要时钟(由元件的延时组成时序逻辑电路)。微操作控制信号采用“命令——微操作——回答”方式进行工作,如图 4.6 所示。

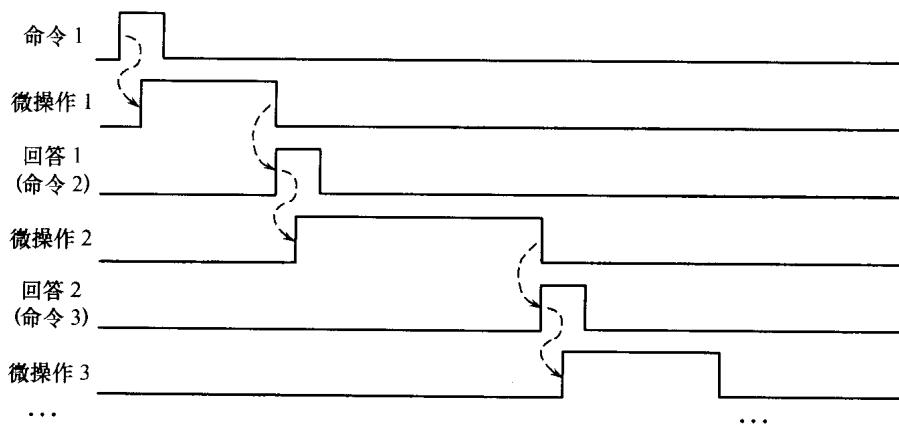


图 4.6 异步控制方式

图中，“命令 1”信号启动“微操作 1”，当“微操作 1”完成后，给出“回答 1”信号；“回答 1”信号同时也是“命令 2”信号，用于启动“微操作 2”；……。

在异步控制方式中，微操作信号的宽度（操作信号的有效时间）是不确定的，每个微操作信号的宽度由对应微操作的需要来确定的。这就意味着异步控制几乎没有时间浪费，这是异步控制方式的优点。异步控制方式的缺点是设计比较复杂，消耗较多的器材，系统调试难度较大，且工作过程中的可靠性不易保证。异步控制方式一般用于速度相差较大的部件或者距离较远的部件之间的信号或者数据交换，如串行接口设备、计算机网络等。部件内部一般很少完全采用异步控制方式进行控制器的设计。

（2）同步控制方式

同步控制方式是指机器有统一的时钟信号，所有的微操作控制信号都与时钟信号同步。如图 4.7 所示。图中 CLOCK 为机器的统一时钟，称为系统时钟；“微操作 1”、“微操作 2”……“微操作 i ”等为各个微操作的控制信号，这些控制信号与系统时钟完全同步。

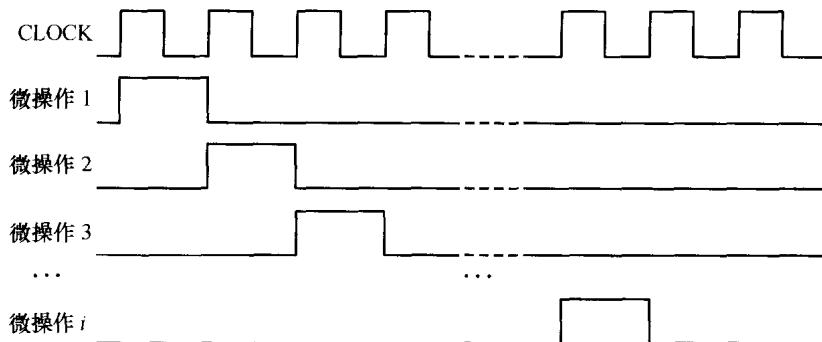


图 4.7 同步控制方式

在同步控制方式下，微操作的操作时间对于时钟的选择有很大影响。由于所有的微操作必须在一个时钟周期内完成，因此系统时钟周期必须不小于最长微操作的工作时间。也就是说，时钟周期的长度必须保证时间最长的微操作可以操作完成。这样，对于其他较快的微操作，必然有时间浪费，这是同步控制方式的缺点。与异步控制方式相比较，同步控制方式设计简单，便于调试，系统较为可靠。同步控制方式是目前控制器设计中采用最多的控制方式。

实例计算机采用同步控制方式，本章研究的重点就是同步控制方式下控制器的设计。在本章以后的讨论中，除非特别声明，均采用同步控制方式进行控制器设计。

3. 时标系统

对于同步控制系统，统一的系统时钟，是协调各个部件有序工作的基本手段。时钟、节拍和节拍电位构成时标系统。合理地设计时标系统，不仅仅可以保证系统工作的协调和正确，也是系统能否高效工作的基础。

(1) 节拍和节拍电位

指令周期内各微操作之间是有顺序要求的。同步方式的组合逻辑控制器中,微操作的顺序通过节拍来控制。在同步控制方式下,根据系统时钟信号,以系统时钟周期为基本单位,将指令周期划分为若干个相等的时间段,每个时间段称为一个节拍。节拍一般用具有一定宽度的电位信号表示,称为节拍电位。在计算机系统中,节拍电位通常是具有周期性的,这个周期称为节拍电位周期。

时钟、指令周期、节拍电位以及节拍电位周期的关系如图 4.8 所示。一般说来,节拍电位的宽度就是机器的系统时钟周期。图中给出了划分成 m 个相等时间段的指令周期 T ,有 m 个节拍电位 $T_0, T_1, T_2, \dots, T_{m-1}$ 与之相对应。从图中可以很清楚地看出,节拍电位 $T_i (m > i \geq 0)$ 经过指令周期 T 时间后,将在下一个指令周期的第 i 个时间段处再出现,所以节拍电位的周期和指令周期是相同的。如果指令周期相同,节拍电位将每隔 T 时间循环出现。

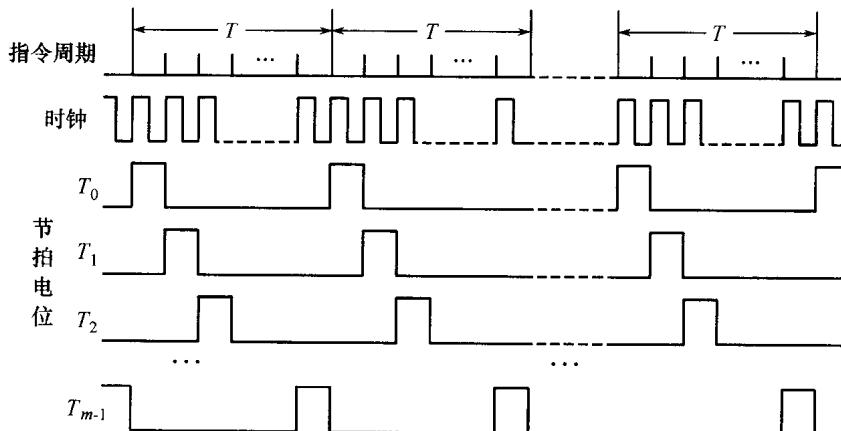


图 4.8 指令周期和节拍电位

指令周期划分为节拍后,将指令的微操作按顺序要求,分配到各个节拍中去执行,所有微操作都与计算机系统时钟信号同步。节拍电位的顺序保证了微操作的顺序。

(2) 节拍电位产生电路

节拍电位产生电路用于产生控制微操作的节拍电位信号。根据指令执行的特点,一般情况下的节拍电位是一种循环信号。因此对于 m 个节拍电位信号,可以采用模 m 计数器和译码器以及其他控制信号来产生。实例计算机具有 8 个节拍,其产生逻辑如图 4.9 所示。

图 4.9 的左半部分是一个模 4 节拍电位产生电路,这 4 个节拍电位分别称为 $T_0 \sim T_3$ 。电路的下半部分为一个模 4 计数器,由两个 JK 触发器构成。上半部分为一个 2/4 译码器, $T_0 \sim T_3$ 就是对触发器的输出进行译码的结果。这 4 个节拍再通过 IF 触发器和 EX 触发器产生 8 个节拍电位: $IF \cdot T_0 \sim IF \cdot T_3, EX \cdot T_0 \sim EX \cdot T_3$ 。需要说明的是,图中 $IF \cdot T_3, EX \cdot \bar{T}_3$ 与 $IF \cdot T_3, EX \cdot T_3$ 的相位正好相反,其作用是用来控制那些需要在 $IF \cdot T_3, EX \cdot T_3$ 节拍下

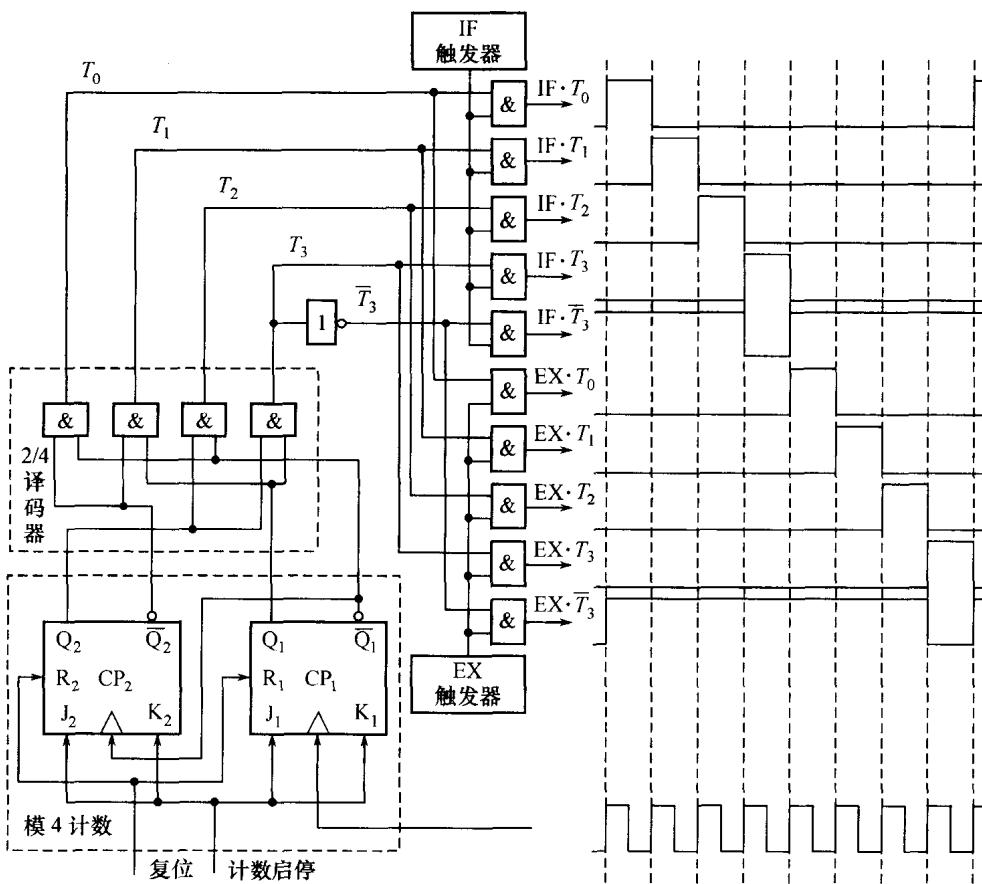


图 4.9 具有 8 个节拍电位的节拍电位产生器

降沿才能进行的操作。

(3) 指令周期和存储周期

存储周期指的是存储器系统的工作周期，即两次存储器访问的最短时间间隔。指令周期通常与存储器周期密切相关。

早期的计算机系统，指令周期通常是存储周期的整数倍，具体的倍数与指令的功能有关。一条没有数据访存的指令，通常其指令周期为一个存储周期，由于指令执行时间相对较短，故这个存储周期主要用于取出指令；一条一地址访存（非间接寻址方式）指令的指令周期需要两个存储周期；如果指令数据需要两次访存，则指令周期等于三倍存储周期，依此类推。图 4.4 所示的指令周期由两个存储周期构成。产生这种倍数关系的主要原因是早期计算机存储器要由系统控制器控制，而且存储器与 CPU 系统在相同的系统时钟下工作。

为提高存储性能，现代计算机系统多采用层次存储结构，按照离 CPU 距离的远近分别设置有 Cache、内存、外存等，由它们共同构成存储层次。现代计算机的指令周期是离 CPU

最近的存储器 Cache 的存储周期的整数倍。当数据不在 Cache 中而必须访问更远的存储器时, 系统通常需要采用加入“等待周期”来完成从存储器取出数据的工作。关于存储层次的知识可以参见本书第五章以及《计算机体系结构》的有关内容。

实例计算机中, 存储器设计和处理器、控制器的设计是相关联的。除移位(SHIFTL 和 SHIFTR)指令外, 其他指令的指令周期均为两个存储周期。这两个存储周期分别称为取指(IF)周期和执行(EX)周期, 每个存储周期又分为 T_0 、 T_1 、 T_2 和 T_3 这 4 个节拍。

(4) 系统时钟周期的确定

对于控制电路而言, 系统时钟周期与节拍宽度是相等的。在控制电路设计时, 最重要的原则是: 除了访存微操作以外, 一般的微操作需要在一个节拍中完成。根据这个原则, 系统时钟频率必须考虑以下因素: 电路的门级延迟 t_d 、电路允许的最大逻辑级数 n 、以及信号时间裕量 Δ (包括电路的线传输延迟时间和电路的离散误差)。

以 T_{cp} 表示节拍宽度, 则下述条件满足时, 电路才能够正常工作:

$$T_{cp} \geq n \times t_d + \Delta$$

一般情况下, 可以取 $\Delta = (1/3 \sim 1/4)n \times t_d$ 。对上式进行变化, 当系统时钟 T_{cp} 确定后, 可以算出系统中微操作允许的最大逻辑级数 n 或者对门级延迟的要求:

$$n \leq \frac{T_{cp} - \Delta}{t_d} \text{ 或 } t_d \leq \frac{T_{cp} - \Delta}{n}$$

根据对门级延迟的要求, 可以用来指导元器件的选择。例如对于 74 系列倒相器电路, 有三种元器件可以选择: 74F04 的门级延迟为 6 ns, 74LS04 的门级延迟为 15 ns, 而 74L04 的门级延迟达 33 ns。

例 4.1 某先行进位加法器运算逻辑为 8.5 级, 加法器输入开关 2 级, 累加器的接收和打入共 6 级, 操作控制信号为 7 级。使用门级延迟为 8 ns 的集成电路设计该加法器, 试确定系统时钟周期。

解: 整个加法器的级数为 $n = 8.5 + 2 + 6 + 7 = 23.5$ (级)

已知 $t_d = 8$ ns, 取 $\Delta = 1/3(n \times t_d) = 62$ ns, 则:

$$T_{cp} \geq n \times t_d + \Delta = 23.5 \times 8 + 62 = 250 \text{ ns}$$

这说明计算机系统时钟周期不能小于 250 ns, 即时钟频率不能够超过 4 MHz。

4. 指令的微操作时间表

有了节拍电位发生器, 就可以安排微操作的顺序了。根据指令的执行过程, 可以将 ADD、SUBTRACT、STORE、LOAD 以及 CLEAR 这 5 条指令分解为微操作序列表, 如表 4.2 至表 4.6 所示。表中各条微操作是以指令和节拍为标志的, 即执行该微操作的时机由指令译码结果以及节拍信号来控制。

实例计算机中, ADD 指令对应的所有微操作被安排在 8 个节拍中完成。在取指周期中完成的工作: 根据指令计数器 PC 的值, 从存储器中读取指令, 将其放入指令寄存器 IR 中。

在此周期中需要形成顺序执行的下一条指令的地址,即 PC 加 1。最后还要改变 IF 和 EX 触发器的状态来结束取指周期和进入执行周期。在执行周期中完成的工作:根据指令中地址码从存储器中取出操作数进行加法。最后还要改变 IF 和 EX 触发器的状态来结束执行周期和进入下一条指令的取指周期。

表 4.2 ADD 的微操作序列表

节 拍	微 操 作	说 明
IF · T ₀	PC 送 MAR 置 R	将程序计数器 PC 的内容送存储器地址寄存器 MAR 从存储器读指令
IF · T ₁	MBR 送 IR 清 R	将读出的指令送指令寄存器 IR 清除存储器读标志 R
IF · T ₂	PC 加 1	程序计数器加 1,准备下一条指令的地址
IF · T̄ ₃	清 IF 置 EX	清除 IF 触发器 置位 EX 触发器,标志计算机从取指周期转入执行周期
EX · T ₀	IR 送 MAR 置 R	将指令寄存器 IR 的地址字段送存储器地址寄存器 MAR 置位 R 触发器,从存储器读加数
EX · T ₁	清 R	清除存储器读标志 R
EX · T ₂	加法	进行加法运算,累加器 AC 的内容与存储缓冲寄存器 MBR 相加
EX · T ₃	ALU 送 AC	加法运算结果从算术逻辑部件 ALU 送累加器 AC
EX · T̄ ₃	清 EX 置 IF	清除 EX 触发器 置位 IF 触发器,标志计算机从执行周期转入取指周期

需要说明的是,微操作序列表中有些微操作是有严格的顺序要求的,有些则不是。例如,对于 ADD 指令来说,“PC 加 1”这个微操作就可以安排在除 IF · T₀ 外的任何一个节拍中。因此,有些微操作的执行时机是可以调整的,也就是说可以通过这种方法来简化控制逻辑。

表 4.2 中第 4 拍为什么使用“IF · T̄₃”而不是“IF · T₃”呢?图 4.9 给出了 8 个节拍信号的产生电路,各节拍电位的产生需要 IF 和 EX 触发器来控制。如果第 4 拍使用“IF · T₃”,则控制器不能正常工作。原因是“IF · T₃”节拍上升沿将执行“清 IF”和“置 EX”微操作,使 IF 触发器变成 0 状态,EX 触发器变成 1 状态。此时节拍信号“IF · T₃”就不能继续维持高电平,故不是一个完整的节拍,安排在第 4 拍中的微操作就可能不能完成而出现错误。另外,由于 EX 变成高电平,将使得“EX · T₃”有效,本来安排在第 8 拍中的微操作就可能提前执行。因此,需要在第 4 拍结束时,使用 IF · T̄₃ 的上升沿(对应 T₃ 的下降沿)完成“清 IF”和“置 EX”微操作。同理,在第 8 拍结束时,使用 EX · T̄₃ 的上升沿(对应 T₃ 的下降沿)完成“清 EX”和“置 IF”微操作。注意,“IF · T̄₃”仍然算作第 4 个节拍,而“EX · T̄₃”仍然算作第 8 个节拍。

表 4.3 中,SUBTRACT 指令取指周期的动作与 ADD 指令完全相同,用“同 ADD”表示。

表 4.3 SUBTRACT 的微操作序列表

节 拍	微 操 作	说 明
IF · T_0	(同 ADD)	(同 ADD)
IF · T_1		
IF · T_2		
IF · \bar{T}_3		
EX · T_0	IR 送 MAR 置 R	将指令寄存器 IR 的地址字段送存储器地址寄存器 MAR 置位 R 触发器, 从存储器读减数
EX · T_1	清 R	清除存储器读标志 R
EX · T_2	减法	进行减法运算, 累加器 AC 的内容与存储缓冲寄存器 MBR 相减
EX · T_3	ALU 送 AC	减法运算结果从算术逻辑部件 ALU 送累加器 AC
EX · \bar{T}_3	清 EX 置 IF	清除 EX 触发器 置位 IF 触发器, 标志计算机从执行周期转入取指周期

表 4.4 STORE 的微操作序列表

节 拍	微 操 作	说 明
IF · T_0	(同 ADD)	(同 ADD)
IF · T_1		
IF · T_2		
IF · \bar{T}_3		
EX · T_0	IR 送 MAR AC 送 MBR 置 W	将指令寄存器 IR 的地址字段送存储器地址寄存器 MAR 将累加器 AC 中的数据送存储缓冲寄存器 MBR 置位 W 触发器, 向存储器中存数
EX · T_1	清 W	清除存储器写标志 W
EX · T_2		(无操作)
EX · \bar{T}_3	清 EX 置 IF	清除 EX 触发器 置位 IF 触发器, 标志计算机从执行周期转入取指周期

表 4.5 LOAD 的微操作序列表

节 拍	微 操 作	说 明
IF · T_0	(同 ADD)	(同 ADD)
IF · T_1		
IF · T_2		
IF · \bar{T}_3		
EX · T_0	IR 送 MAR 置 R	将指令寄存器 IR 的地址字段送存储器地址寄存器 MAR 置位 R 触发器, 从存储器读数
EX · T_1	清 R	清除存储器读标志 R
EX · T_2		(无操作)

(续表)

节拍	微操作	说明
$EX \cdot T_3$	MBR 送 AC	将存储缓冲寄存器 MBR 中的数据送累加器 AC
$EX \cdot \bar{T}_3$	清 EX 置 IF	清除 EX 触发器 置位 IF 触发器, 标志计算机从执行周期转入取指周期

表 4.6 CLEAR 的微操作序列表

节拍	微操作	说明
$IF \cdot T_0$		
$IF \cdot T_1$	(同 ADD)	(同 ADD)
$IF \cdot T_2$		
$IF \cdot \bar{T}_3$		
$EX \cdot T_0$		(无操作)
$EX \cdot T_1$		(无操作)
$EX \cdot T_2$		(无操作)
$EX \cdot T_3$	AC 复位	将 AC 清零
$EX \cdot \bar{T}_3$	清 EX 置 IF	清除 EX 触发器 置位 IF 触发器, 标志计算机从执行周期转入取指周期

5. 指令的识别

设计控制器,首先碰到的设计问题就是对指令如何识别。由表 4.1 可知,不同的指令是通过操作码来区别的。一般情况下,指令的识别是通过指令译码器来完成的。指令译码可以采用如图 4.10 所示的单级全译码或者图 4.11 所示的分级译码两种设计方法。

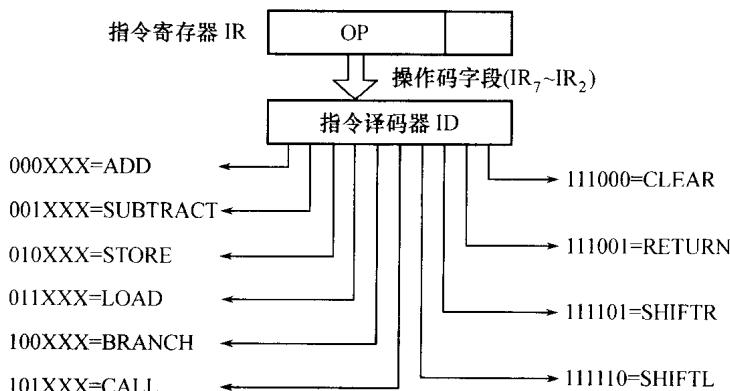


图 4.10 单级全译码

图 4.10 中,指令寄存器的高 6 位 ($IR_7 \sim IR_2$) 参加译码。如果指令操作码均为 6 位,则

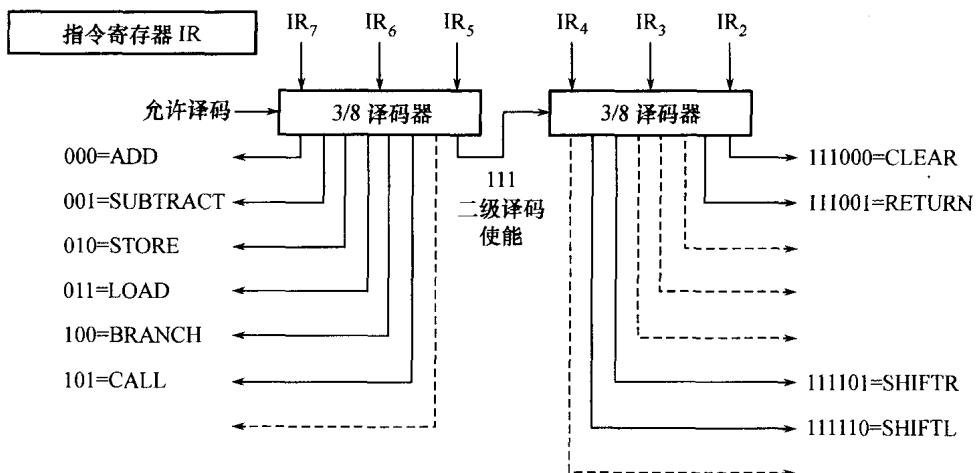


图 4.11 分级译码

指令译码器可以识别 64 条指令,译码电路的复杂度较高。图中“X”代表该位取“0”或者“1”均可。例如“000XXX”表示实例计算机的基本 1 地址指令只有 3 位操作码,余下的 3 位操作码“XXX”不起作用,无论“X”是“0”还是“1”,译码结果均表示加法指令。图 4.10 中,由于其他的译码结果无效,故没有画出来。

图 4.11 中,指令寄存器的 $IR_7 \sim IR_5$ 由一个 3/8 译码器控制构成一级译码电路,而 $IR_4 \sim IR_2$ 由另一个 3/8 译码器控制构成二级译码电路。第一级的译码结果“111”作为第二级译码电路的使能信号。对于实例计算机来说,基本指令使用第一级译码电路即可,而扩展指令则必须使用二级译码电路。图中,虚线代表的是没有用的译码结果,可以用来扩展更多的指令。

对比图 4.10 和图 4.11 可以看出,如果全部指令操作码为等长,并且操作码长度较短时,采用全译码方法比较简单有效,速度较快。如果操作码为变长,或者操作码长度很长时,采用分级译码方法则比较简单有效。目前,大多数计算机(包括 RISC)的指令操作码为变长,所以分级译码方法是目前广泛采用的技术。同样,由于实例计算机指令为变长操作码,并且扩展指令的操作码位数较长(6 位),故采用分级译码方法对指令进行识别是一种较好的选择。

6. 微操作控制信号的生成

获得指令译码的结果以后,根据指令的微操作序列表,就可以获得微操作控制信号生成电路。根据表 4.2 ~ 表 4.6,实例计算机在具有 ADD 等 5 条指令时,应该包括以下这些微操作:

- (1) 置 IF: 设置取指周期标志。
- (2) 清 IF: 清除取指周期标志。
- (3) 置 EX: 设置执行周期标志。

- (4) 清 EX:清除执行周期标志。
 - (5) 置 R:启动存储器读。
 - (6) 清 R:清除 R 触发器。
 - (7) 置 W:启动存储器写。
 - (8) 清 W:清除 W 触发器。
 - (9) PC 加 1:程序计数器加 1,准备下一条指令的地址。
 - (10) PC 送 MAR:将下一条指令地址送存储器地址寄存器 MAR。
 - (11) MBR 送 IR:将读出的指令送指令寄存器。
 - (12) IR 送 MAR:将指令寄存器的地址字段送存储器地址寄存器 MAR(取操作数用)。
 - (13) 加法:ALU 进行加法运算。
 - (14) 减法:ALU 进行减法运算。
 - (15) ALU 送 AC:运算结果从算术逻辑部件 ALU 送累加器 AC。
 - (16) AC 复位:将 AC 清零。
 - (17) AC 送 MBR:将累加器 AC 中的数据送存储缓冲寄存器 MBR。
 - (18) MBR 送 AC:将存储缓冲寄存器 MBR 中的数据送累加器 AC。

为了便于设计微操作控制信号和便于检查其正确性与完整性,需要进一步将微操作进行分类,构成微操作表。在微操作表中需要列出所有微操作的控制信号以及使该控制信号有效的条件。表 4.7 就是实例计算机在具有 ADD、SUBTRACT、STORE、LOAD 以及 CLEAR 这 5 条指令时的微操作表。需要注意的,实例计算机只有 8 个节拍,“EX · T”仍然算作第 8 个节拍。

表 4.7 5 条指令的实例计算机的微操作表

(续表)

节拍 微操作	IF				EX				
	T ₀	T ₁	T ₂	T̄ ₃	T ₀	T ₁	T ₂	T ₃	T̄ ₃
MBR 送 IR	All								
IR 送 MAR					ADD + SUBTRACT + STORE + LOAD				
加法							ADD		
减法							SUBTRACT		
ALU 送 AC								ADD + SUBTRACT	
AC 复位								CLEAR	
AC 送 MBR					STORE				
MBR 送 AC								LOAD	

表 4.7 中, “All”表示对所有的指令均有效。根据该表, 可以得到所有 18 个微操作控制信号的逻辑表达式:

- (1) 置 IF = EX · T̄₃
- (2) 清 IF = IF · T̄₃
- (3) 置 EX = IF · T̄₃
- (4) 清 EX = EX · T̄₃
- (5) 置 R = IF · T₀ + EX · T₀ · (ADD + SUBTRACT + LOAD)
- (6) 清 R = IF · T₁ + EX · T₁ · (ADD + SUBTRACT + LOAD)
- (7) 置 W = EX · T₀ · STORE
- (8) 清 W = EX · T₁ · STORE
- (9) PC 加 1 = IF · T₂
- (10) PC 送 MAR = IF · T₀
- (11) MBR 送 IR = IF · T₁
- (12) IR 送 MAR = EX · T₀ · (ADD + SUBTRACT + STORE + LOAD)
- (13) 加法 = EX · T₂ · ADD
- (14) 减法 = EX · T₂ · SUBTRACT
- (15) ALU 送 AC = EX · T₃ · (ADD + SUBTRACT)
- (16) AC 复位 = EX · T₃ · CLEAR
- (17) AC 送 MBR = EX · T₀ · STORE
- (18) MBR 送 AC = EX · T₃ · LOAD

这些表达式代表着控制器发出的微操作的控制信号, 表达式左边表示的是需要控制的

微操作,右边表示执行该微操作需满足的条件。从控制器设计的角度看,表达式左边代表连接到受控部件的控制信号线,表达式右边代表使该控制信号线有效的条件。例如对于第 15 个微操作“ALU 送 AC”,该控制信号产生的逻辑如图 4.12 所示。

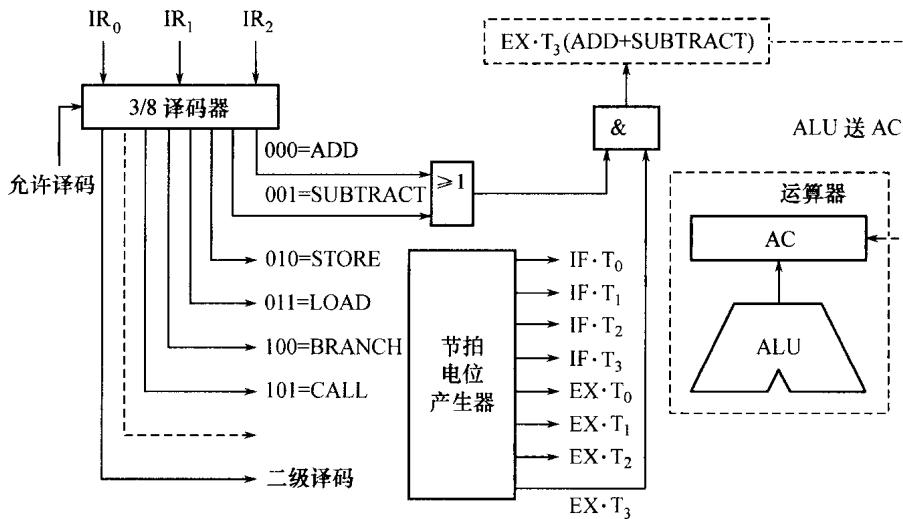


图 4.12 控制信号“ALU 送 AC”的产生电路

有了上述表达式及其对应的控制信号产生电路,就可以实现具有 ADD、SUBTRACT、STORE、LOAD 以及 CLEAR 这 5 条指令控制器的设计了。配合运算器和存储器系统,可以完成构造具有 5 条指令的实例计算机。

本节讨论的是组合逻辑控制器设计技术的一些基本概念和方法,包括控制器的基本结构,指令的分解、指令周期、取指周期、执行周期的关系,微操作的概念,异步控制方式和同步控制方式,时标系统、时钟和节拍信号产生电路,简单指令的实现等问题。通过这些概念和技术的运用,就可以进行简单控制器的设计了。

4.3 指令流控制和复杂指令的设计

本节首先通过实例计算机中条件转移(BRANCH)、转子(CALL)和子程序返回(RETURN)指令的实现来讨论怎样对计算机指令流进行控制;然后通过右移(SHIFTR)和左移 SHIFTL 指令的实现来讨论怎样对非固定节拍指令周期的指令操作进行控制。

4.3.1 计算机指令流的控制

指令流的控制就是改变计算机执行指令的流向,使计算机不再按照“PC + 1”的流程执行程序。所有现代计算机都具备对指令流进行控制的能力,这种控制能力使得计算机程序

具有更大的灵活性和适应性,大大提高了程序设计的效率。下面,对实例计算机的 BRANCH、CALL 和 RETURN 三条指令流控制指令的操作予以分析说明。

1. 条件转移指令(BRANCH)

计算机中,各种各样的转移指令是最典型的也是最基本的指令流控制指令。转移指令一般可分为两类:条件转移和无条件转移。实例计算机设置有条件转移指令 BRANCH,它通过对累加器 AC 最高位 AC₇ 进行判断,决定是否转移。如果 AC₇ = 1,则按指令的地址部分进行转移,就是将指令寄存器 IR₄ ~ IR₀ 的内容送程序计数器 PC;如果 AC₇ = 0,则程序计数器 PC 内容加 1,程序顺序执行。

如果认为 AC₇ 是累加器 AC 的符号位,且采用带符号的数据表示,则这条转移指令就是负数转移指令(Branch on Minus),即当 AC 中为负数(符号位为“1”)时转移,AC 中为正数(符号位为“0”)时顺序执行。

BRANCH 指令的微操作序列表如表 4.8 所示。

表 4.8 BRANCH 的微操作序列表

节 拍	微 操 作	说 明
IF • T ₀	(同 ADD)	(同 ADD)
IF • T ₁		
IF • T ₂		
IF • T̄ ₃		
EX • T ₀		(无操作)
EX • T ₁		(无操作)
EX • T ₂ • (AC ₇ = 0)		若 AC ₇ = 0,不修改 PC,顺序执行
EX • T ₂ • (AC ₇ = 1)	IR 送 PC	若 AC ₇ = 1,将指令地址字段内容(转移目标地址)送程序计数器
EX • T̄ ₃	清 EX	清除 EX 触发器
	置 IF	置位 IF 触发器,标志计算机从执行周期转入取指周期

BRANCH 指令在取指周期与 ADD 指令完成的操作是相同的。在执行周期中,T₀、T₁ 节拍没有操作,T₂ 节拍将根据累加器 AC₇ 的值对程序计数器 PC 进行修改。若转移条件满足,则将指令地址字段内容(转移目标地址)送程序计数器。微操作“IR 送 PC”是原来没有的微操作,必须单独产生。可以在 4.2 节中的控制电路的基础上进行扩充。因此,“IR 送 PC”是第 19 个控制信号,它的信号表达式是:

$$(19) \text{ IR 送 PC} = \text{EX} \cdot T_2 \cdot \text{BRANCH} \cdot (AC_7 = 1)$$

2. 转子指令(CALL)和子程序返回指令(RETURN)

转子指令 CALL 和子程序返回指令 RETURN 与条件转移指令 BRANCH 的相同之处在于都可以改变指令的执行顺序,即指令执行后程序计数器 PC 内容不是加 1,而是改变为目

标指令的地址。

转子指令与条件转移指令的区别有两点,其一是转子指令在执行周期中,不做任何条件判断,直接改变程序计数器 PC 的值,相当于无条件转移指令;其二是转子指令还需要将当前的 PC 保存到由栈顶指针 SP 指示的存储单元中作为返回地址。实例计算机的 SP 被初始化为 00011111B,执行转子指令时需先进行压栈操作,首先将 PC 进栈,而后再将指令地址字段中的子程序入口地址送入 PC,作为下一条指令的地址。

子程序返回指令也相当于一条无条件转移指令,但它的转移目标地址不在指令中,而是存放在由栈顶指针 SP 指示的存储单元中。所以,在执行周期中,子程序返回指令首先需要根据 SP 的内容完成存储器读操作,然后再将取出的返回地址送程序计数器 PC,作为下一条指令的地址。对于堆栈来说,子程序返回指令对应弹栈操作。

根据上述分析,可以形成 CALL 指令和 RETURN 指令的微操作序列表,如表 4.9 和表 4.10 所示。

表 4.9 CALL 的微操作序列表

节拍	微操作	说明
IF · T ₀		
IF · T ₁	(同 ADD)	(同 ADD)
IF · T ₂		
IF · T̄ ₃		
IF · T ₃	SP 减 1	压栈前,将栈顶指针 SP 减 1,指向新栈顶
EX · T ₀	SP 送 MAR	将栈顶指针 SP 的内容送存储器地址寄存器 MAR
	PC 送 MBR	将程序计数器 PC 的内容送存储器缓冲寄存器 MBR
	置 W	置位 W 触发器,向存储器中存数,保存返回地址
EX · T ₁	清 W	清除存储器写标志 W
EX · T ₂	IR 送 PC	将指令地址字段内容即子程序入口地址送程序计数器
EX · T̄ ₃	清 EX	清除 EX 触发器
	置 IF	置位 IF 触发器,标志计算机从执行周期转入取指周期

表 4.10 RETURN 的微操作序列表

节拍	微操作	说明
IF · T ₀		
IF · T ₁	(同 ADD)	(同 ADD)
IF · T ₂		
IF · T̄ ₃		
EX · T ₀	SP 送 MAR	将栈顶指针 SP 的内容送存储器地址寄存器 MAR
	置 R	置位 R 触发器,从存储器读数,取出返回地址

(续表)

节 拍	微 操 作	说 明
EX · T ₁	SP 加 1 清 R	弹栈后, 栈顶指针 SP 加 1, 指向新栈顶 清除存储器读标志 R
EX · T ₂	MBR 送 PC	将存储器缓冲寄存器 MBR 的内容送程序计数器 PC
EX · T ₃	清 EX 置 IF	清除 EX 触发器 置位 IF 触发器, 标志计算机从执行周期转入取指周期

从表 4.9 和表 4.10 可以看出, CALL 指令和 RETURN 指令在执行周期需要使用“SP 送 MAR”, “SP 加 1”, “SP 减 1”, “PC 送 MBR”和“MBR 送 PC”这几个新的微操作, 它们的表达式分别为:

$$(20) \text{ SP 送 MAR} = \text{EX} \cdot \text{T}_0 \cdot (\text{CALL} + \text{RETURN})$$

$$(21) \text{ SP 加 1} = \text{EX} \cdot \text{T}_1 \cdot \text{RETURN}$$

$$(22) \text{ SP 减 1} = \text{IF} \cdot \text{T}_3 \cdot \text{CALL}$$

$$(23) \text{ PC 送 MBR} = \text{EX} \cdot \text{T}_0 \cdot \text{CALL}$$

$$(24) \text{ MBR 送 PC} = \text{EX} \cdot \text{T}_2 \cdot \text{RETURN}$$

另外, 还需要对以下几个前面已经讲过的微操作进行调整:

$$(5) \text{ 置 R} = \text{IF} \cdot \text{T}_0 + \text{EX} \cdot \text{T}_0 \cdot (\text{ADD} + \text{SUBTRACT} + \text{LOAD})$$

改为: 置 R = IF · T₀ + EX · T₀ · (ADD + SUBTRACT + LOAD + RETURN)

$$(6) \text{ 清 R} = \text{IF} \cdot \text{T}_1 + \text{EX} \cdot \text{T}_1 \cdot (\text{ADD} + \text{SUBTRACT} + \text{LOAD})$$

改为: 清 R = IF · T₁ + EX · T₁ · (ADD + SUBTRACT + LOAD + RETURN)

$$(7) \text{ 置 W} = \text{EX} \cdot \text{T}_0 \cdot \text{STORE}$$

改为: 置 W = EX · T₀ · (STORE + CALL)

$$(8) \text{ 清 W} = \text{EX} \cdot \text{T}_1 \cdot \text{STORE}$$

改为: 清 W = EX · T₁ · (STORE + CALL)

$$(19) \text{ IR 送 PC} = \text{EX} \cdot \text{T}_2 \cdot \text{BRANCH} \cdot (\text{AC}_7 = 1)$$

改为: IR 送 PC = EX · T₂ · (BRANCH · (AC₇ = 1) + CALL)

通过本小节的讨论可以知道如何实现程序指令流的控制。在实际计算机中对指令流的控制手段还有很多。对于转移而言, 实际计算机往往有许多条件可以判断, 如大于、小于、等于、溢出、进位、负数等。这些条件通常可以使用一个或者多个标志寄存器, 或者条件码寄存器来保存, 从而形成多种多样的条件转移指令。

实际计算机中, 转子指令和子程序返回指令的处理情况还要复杂一些, 由于种种技术原因, 子程序的返回地址往往不是存放在内存的固定位置, 它在内存中可以浮动。实际计算机中采用堆栈技术较好地解决了这个问题。有关堆栈技术的细节可以参考“数据结构”等课程中的相关内容。

4.3.2 局部控制周期技术

以上研究的 8 条指令都是在两个存储周期(8 个节拍)内完成的。从微操作序列中可以看出,并不是每条指令在每一个节拍中都有微操作,只是为了便于工程实现,在没有微操作的节拍中,让机器空闲,保证了这些指令的周期是相同的。

实际计算机中有一些指令需要更多的时间,如乘法和除法运算;还有些指令的执行时间是不确定的,如左移和右移指令。与前面的指令相比,这些指令同样具有取指令、分析指令、执行指令的过程,主要区别是不能在固定的时间(例如,两个存储周期)内完成对数据的操作。因此,需要在指令执行时间上对控制器进行功能扩展。

在控制器设计时,可以采用中央控制、局部控制和混合控制这三种方式来实现对指令执行时间的控制。

1. 中央控制

中央控制就是要有一种适合于计算机中所有指令的处理方式或时序分配方式,使所有指令都在统一的时序下进行处理,所有指令的指令周期都是相同的。这种方法的优点是控制器的逻辑相对简单,时间便于控制;不足之处是要求所有的指令都以相同的方式进行处理,在效率和功能上给控制器设计带来一些矛盾。如果要侧重效率,就必须让所有指令的处理时间相同或相近,因而限制了采用一些功能较复杂、处理时间很长的指令;反之,如果采用了功能复杂、处理时间长的指令,控制器处理指令的时序就要加长,那么其他简单功能的指令就存在时间浪费、处理效率不高等问题。速度要求不高、功能简单的控制器通常采用这种控制方法。

2. 局部控制

局部控制就是每条指令都有独立的处理方式或时序分配方式,由指令启动各自的时序进行处理,每一条指令的指令周期取决于它的微操作序列长度。这种方法的优点与不足和中央控制方式正好相反,即处理效率高,控制器逻辑复杂。

3. 混合控制

通常情况下,计算机中所有指令的大部分处理过程是相同的。只有某些指令,如乘法、除法、移位等指令所需的运算时间较长。混合控制就是在中央控制的基础上,对个别指令进行局部控制的一种设计方式。这种方式下,控制器对大部分的指令采用中央控制,即大多数指令都采用相同的指令周期,称之为指令基本周期。对于中央控制方式难以处理的指令,将其中处理时间较长的微操作采用局部控制方式,这段时间称之为局部控制周期。当指令的某个操作步需要较长时间时,中央控制时序被暂停,进入该指令的局部控制周期。该局部控制周期由局部控制时序电路提供,一直到该操作步结束。在局部控制周期结束时,再次进入中央控制时序,完成指令处理的所有操作步骤。混合控制方式的处理过程如图 4.13 所示。

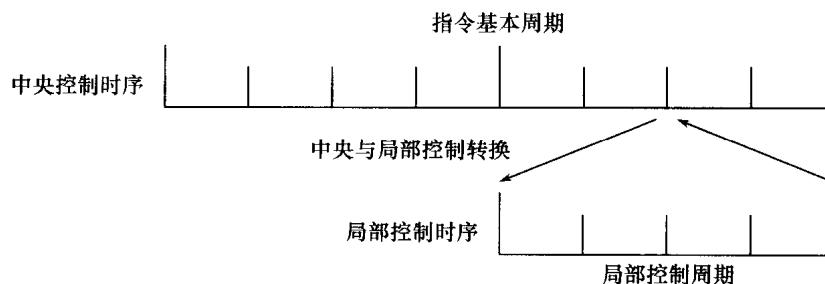


图 4.13 混合控制、指令基本周期与局部控制周期

图 4.13 中,局部控制周期的长度可以根据指令对应操作步的具体需要而定。

到目前为止,对于实例计算机的讨论,控制器采用的都是中央控制方式。下面将在中央控制的基础上,以移位指令的实现为例,来阐述混合控制方式是如何实现的。

实例计算机的移位指令包括右移指令(SHIFTR)和左移指令 SHIFT(L)。右移指令和左移指令的指令字由两部分构成:6 位操作码,指明本指令是左移还是右移指令;2 位地址码,指出移位的位数。因为地址码是 2 位,所以最大的移位值为 3。

由第三章的介绍可知,移位操作是需要运算器支持的。假设无论左移还是右移,运算器每个时钟可以完成 1 位移位。移位的位数越多,指令执行所需的时钟数就越多,故移位指令的指令周期是不固定的。实例计算机对移位指令实施混合控制,在指令基本周期的基础上,增加长度不固定的局部控制周期。

为了完成移位计数,需要设置一个移位计数器(Shift Counter,SC),这个计数器为减 1 计数,其初值为移位指令的地址码部份(移位量)。先判 0,再移位,后减 1。SC 从移位量计数到 0,则移位结束。SC 需要使用时钟 Clock 进行计数,对 SC 赋初值需要“IR 送 SC”微操作来控制。在混合控制中,为实现中央控制和局部控制的转换,需要设置一个移位触发器 SR(Shift Register,SR)作为标志。SR 被置位,表示指令的执行进入局部控制时序。当指令字中 IR₁ 与 IR₀ 均为 0,或者移位计数器 SC 计数结束时,SR 被复位,表示指令的执行从局部控制时序回到中央控制时序。中央控制与局部控制转换的原理如图 4.14 所示。

图 4.14 中,中央微操作控制部件和局部微操作控制部件都是复杂的时序逻辑网络。CTSG 为中央节拍信号产生器,其逻辑结构如图 4.9 所示;LTSG 为局部节拍信号产生器,移位指令计数结束的控制逻辑如图 4.15 所示。CTSG 和 LTSG 都是由计数器、译码分配线路组成的时序网络。需要说明的是,CTSG 和 LTSG 可以使用相同的时钟信号控制,也可以使用不同的时钟信号,即基本指令周期和局部控制周期中的节拍电位宽度可以不同。

图 4.14 中,CC 为中央控制工作触发器,CC 为“1”状态时表示为中央控制周期;LC 为局部控制工作触发器,LC 为“1”状态时表示为局部控制周期。CC 和 LC 不能同时为“1”或者“0”状态。实例计算机中,大部分时间是处于中央控制下工作的。此时,CC 为“1”状态,由 CTSG 给出固定的节拍信号(从 IF · T₀ 到 EX · T₃),中央微操作控制部件在指令译码器 ID

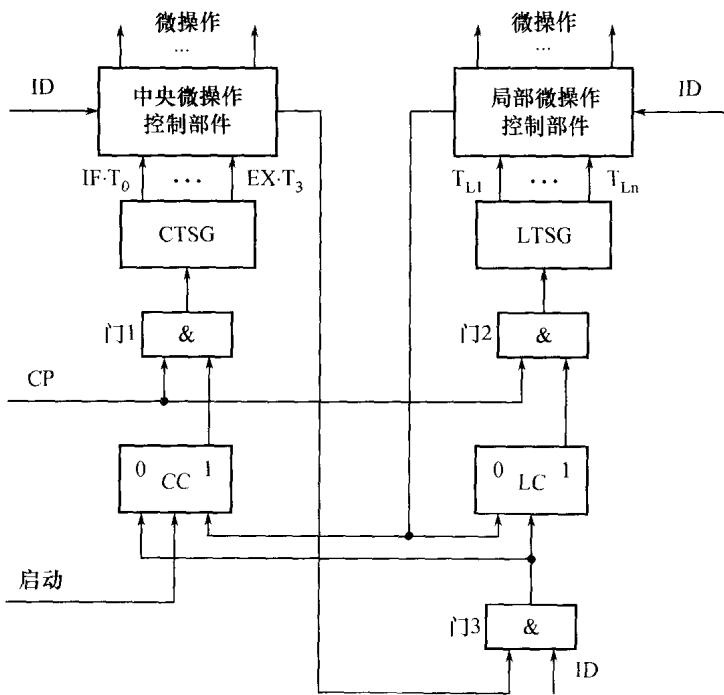


图 4.14 中央控制与局部控制的转换

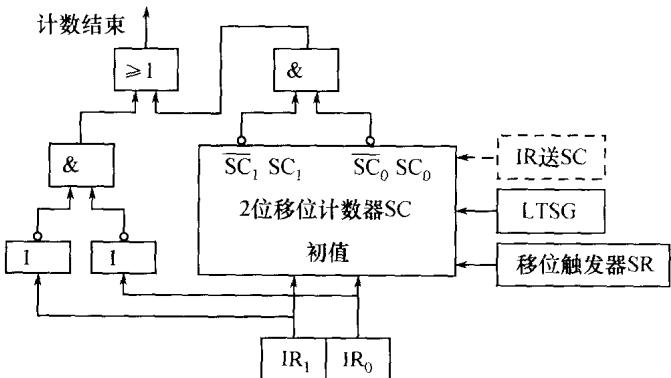


图 4.15 移位指令局部控制电路的逻辑结构

的输出与 CTSG 节拍信号的控制下,形成指令所需要的微操作控制信号序列,控制机器各部件完成指令所规定的操作。而对于移位指令时,由于指令操作需要的时间较长且不固定,中央控制已不能完成该指令微操作序列,需要启动局部控制。执行移位指令时,中央控制完成取指周期,已经为移位指令的执行做好准备,就要启动局部控制。需要说明的是,局部控制可以发生在中央控制的任意两个节拍之间,何时转向局部控制决定于指令流程的安排。

执行移位指令时,中央控制与局部控制之间是如何相互转换的呢?从图 4.14 可见,当

需要进入局部控制时,由中央微操作控制部件发出控制信号加到门3的输入端, ID 给出移位指令的译码信号也加到门3的输入端,门3的输出置0触发器 CC,停止中央控制;置1触发器 LC,启动局部控制。此时,时钟信号停止送往中央控制线路并转向送到局部控制线路 LTSG。由 LTSG 发出局部节拍信号 T_{Li} ,其产生方法与 T_i 类似,但不具有周期性。 T_{Li} 送到局部微操作控制部件,在 ID 控制下产生移位信号。当移位完成时,由局部微操作控制部件发出控制信号,置0触发器 LC,停止局部控制;置1触发器 CC,重新启动中央控制。右移和左移两条指令的微操作序列如表 4.11 和表 4.12 所示。

表 4.11 SHIFTR 的微操作序列表

节 拍	微 操 作	控 制 方 式	说 明
IF • T_0	(同其他指令)	中央	(同其他指令)
IF • T_1			
IF • T_2			
IF • T_3	IR 送 SC	中央	指令寄存器的内容(移位量)送移位计数器
IF • \bar{T}_3	清 IF 置 SR	中央	清除 IF 触发器 置位 SR 触发器,标志计算机从中央控制转入局部控制
SR • $T_{Li} \cdot (SC \neq 00)$	右移	局部	对 AC 的内容进行右移
SR • $\bar{T}_{Li} \cdot (SC = 00)$	清 SR 置 EX	局部	清除 SR 触发器,标志计算机从局部控制转入中央控制 置位 EX 触发器,标志计算机转入执行周期
EX • T_0	(无操作)	中央	(无操作)
EX • T_1			
EX • T_2			
EX • \bar{T}_3	清 EX 置 IF	中央	清除 EX 触发器 置位 IF 触发器,标志计算机从执行周期转入取指周期

表 4.12 SHIFTL 的微操作序列表

节 拍	微 操 作	控 制 方 式	说 明
IF • T_0	(同右移)	中央	(同右移)
IF • T_1			
IF • T_2			
IF • $T_3/IF \cdot \bar{T}_3$			
SR • $T_{Li} \cdot (SC \neq 00)$	左移	局部	对 AC 的内容进行左移
SR • $\bar{T}_{Li} \cdot (SC = 00)$	(同右移)	局部	(同右移)
EX • T_0	(同右移)	中央	(同右移)
EX • T_1			
EX • T_2			
EX • \bar{T}_3			

从表 4.11 和表 4.12 中不难看出,两条指令的操作过程几乎是一致的,只是在局部控制周期中的运算器功能不同:一个是右移,一个是左移。使用的 \bar{T}_{Li} 原因与使用 $IF \cdot \bar{T}_3$ 的原因相同,目的是在 \bar{T}_{Li} 的下降沿完成清除 SR 触发器的工作。对移位计数器 SC 的赋值、对移位触发器 SR 的置位与复位、以及右移与左移是新的微操作,它们的表达式分别为:

- (25) IR 送 SC = IF $\cdot T_3 \cdot (SHIFTR + SHIFTL)$
- (26) 置 SR = IF $\cdot \bar{T}_3 \cdot (SHIFTR + SHIFTL)$
- (27) 清 SR = SR $\cdot \bar{T}_{Li} \cdot (SC = 00) \cdot (SHIFTR + SHIFTL)$
- (28) 右移 = SR $\cdot T_{Li} \cdot (SC \neq 00) \cdot SHIFTR$
- (29) 左移 = SR $\cdot T_{Li} \cdot (SC \neq 00) \cdot SHIFTL$

另外,还需要对一个微操作进行调整:

- (3) 置 EX = IF $\cdot \bar{T}_3$ 改为: 置 EX = IF $\cdot \bar{T}_3 \cdot (ADD + SUBTRACT + STORE + LOAD + CLEAR + BRANCH + CALL + RETURN) + SR \cdot \bar{T}_{Li} \cdot (SC = 00) \cdot (SHIFTR + SHIFTL)$

图 4.16 为能够对实例计算机的 10 条指令进行控制的控制器逻辑图。

需要说明的是,图 4.16 中“SP 复位”微操作仅在系统复位时使用。图中 TSG 既代表了中央控制节拍信号发生器 CTSG,也代表了局部控制节拍信号发生器 LTSG。该控制器除了能够完成图 4.5 所示的基本控制器的所有控制外,还实现了对指令流的控制以及混合控制。

4.3.3 微操作控制信号组合逻辑网络的实现

有了微操作对应的逻辑表达式,就可以采用基本门电路来实现微操作控制信号的组合逻辑网络。但是,用这种方法构建实际计算机的微操作控制部件,逻辑电路十分复杂,不利于计算机设计的自动化和 VLSI 实现。

随着半导体集成电路的不断发展,可编程逻辑器件(Programmable Logic Device, PLD)在许多数字系统的设计中得到广泛应用。PLD 最大的特点是生产出来后,还可以对内部的连接进行编程,通过改变内部逻辑单元的连接方式达到实现所需功能的目的。常见的 PLD 器件主要有:可编程只读存储器(Programmable Read Only Memory, PROM),可编程逻辑阵列(Programmable Logic Array, PLA),可编程阵列逻辑(Programmable Array of Logic, PAL),通用阵列逻辑(Generic Array Logic, GAL)。关于这些逻辑器件的区别和 PLD 的有关详细内容请参阅本系列教材《数字逻辑原理与工程设计》。微操作控制信号组合逻辑网络可以使用不同的 PLD 器件实现,下面仅介绍使用 PLA 的实现方法。

PLA 的主要结构由与、或阵列组成,但 PLA 的与阵列和或阵列都是可编程的,可以用来实现任何组合逻辑电路的功能。PLA 的基本原理是基于布尔函数 $F = \sum m_i$ (m_i 为最小项) 简化为 $F = \sum P_i$ (P_i 为质蕴涵项) 后,在 PLA 与阵列中产生所需的 P_i 项。所谓“可编程”指的就是需要什么与项 P_i ,就可以在与阵列中产生该与项。图 4.17 给出了一个 PLA 的基本结构的示意图。

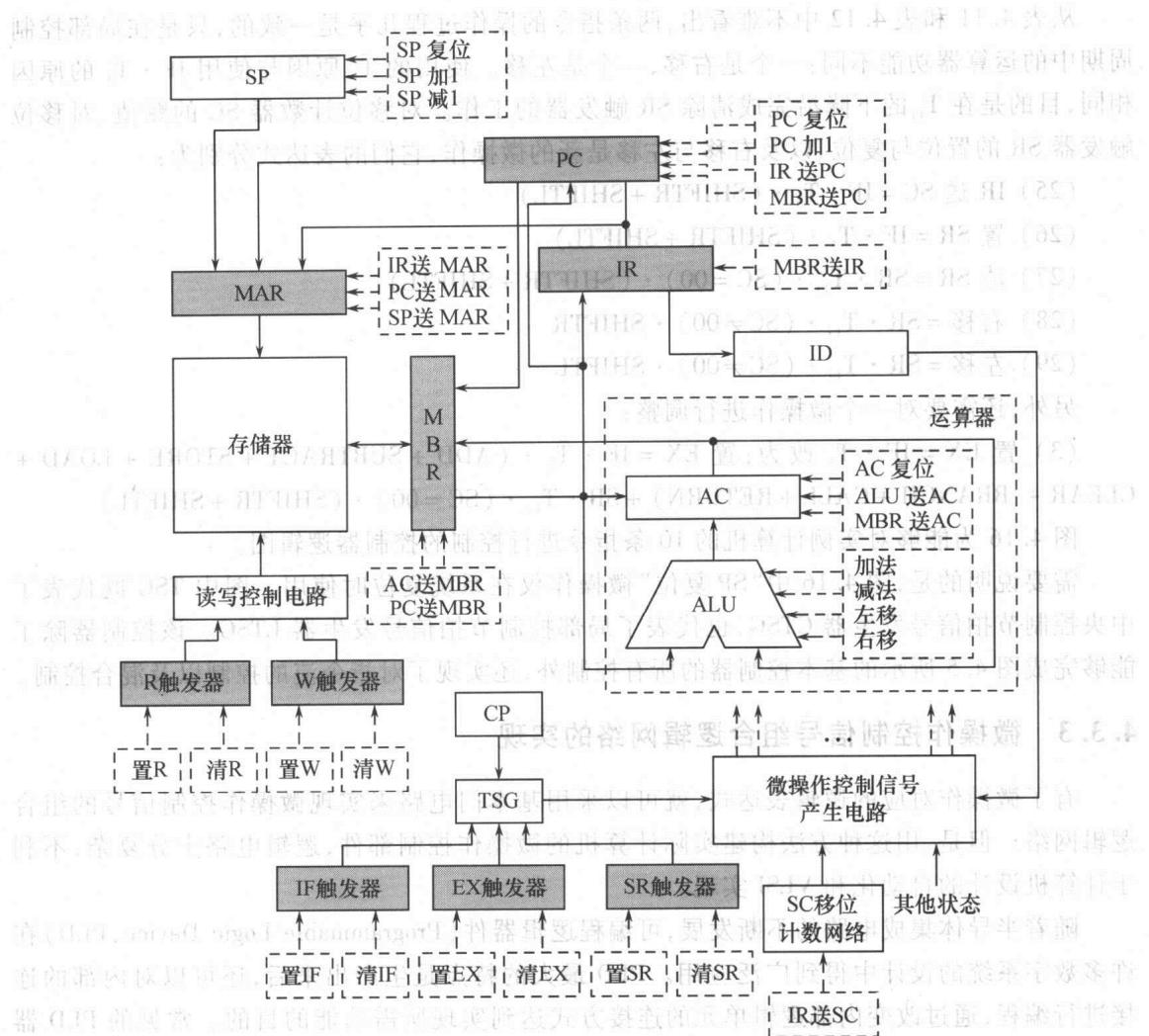


图 4.16 运算器、存储器和处理 10 条指令的控制器

图 4.17 所示的 PLA 具有 6 个输入信号 $I_5 \sim I_0$, 可以提供 6 个不同的最小项 $P_0 \sim P_5$, 可以实现 5 个输出函数 $F_4 \sim F_0$ 。其与阵列的容量为 12×6 位, 或阵列的容量为 5×6 位。如果将 I_0 与 CTSG 的节拍信号 $IF \cdot \bar{T}$ 相连(用 $I_0 = IF \cdot \bar{T}$ 表示), 将 I_1 与 LTSG 的节拍信号 T_{Li} 相连(用 $I_1 = T_{Li}$ 表示), 将 I_2 与 SC 的状态相连(当 $SC = 0$ 时, I_2 为高电平), 将 I_3 与移位触发器 SR 相连, 将 I_4 和 I_5 分别与 ID 的译码结果相连(译码为 SHIFTR 指令时, I_4 有效; 译码为 SHIFTL 指令时, I_5 有效), 此时使用该 PLA 就可以实现实例计算机的下列 4 个逻辑函数:

$$(26) \text{ 置 } SR = F_0 = P_0 + P_1 = I_0 \cdot I_5 + I_0 \cdot I_4 = IF \cdot \bar{T}_3 \cdot (SHIFTR + SHIFTL)$$

$$(27) \text{ 清 } SR = F_1 = P_2 + P_3 = \bar{I}_1 \cdot I_2 \cdot I_3 \cdot I_5 + \bar{I}_1 \cdot I_2 \cdot I_3 \cdot I_4 = SR \cdot \bar{T}_{Li} \cdot (SC = 00) \cdot (SHIFTR + SHIFTL)$$

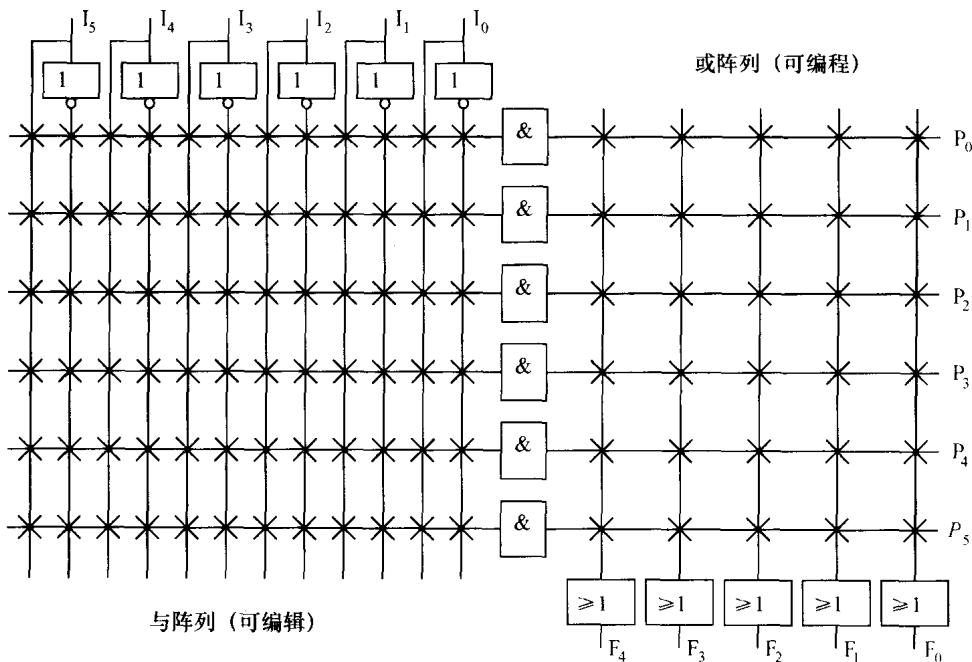


图 4.17 PLA 基本结构示意图

$$(28) \text{ 右移 } = F_2 = P_4 = I_1 \cdot \bar{I}_2 \cdot I_3 \cdot I_5 = SR \cdot T_{Li} \cdot (SC \neq 00) \cdot SHIFTR$$

$$(29) \text{ 左移 } = F_3 = P_5 = I_1 \cdot \bar{I}_2 \cdot I_3 \cdot I_4 = SR \cdot T_{Li} \cdot (SC \neq 00) \cdot SHIFTL$$

编程后的 PLA 电路连接如图 4.18 所示。

PLA 也是一种集成电路芯片,每种 PLA 芯片的容量及输入输出位数总是有限的。在实现微操作控制信号组合逻辑网络时,单个 PLA 芯片的容量及输入输出位数往往不能满足要求,需要使用多个 PLA 芯片来实现。

4.3.4 组合逻辑控制器的设计过程

具有简单指令系统的实例计算机,为实现对 10 条指令的控制,需要使用 31 个微操作。每种微操作的控制信号均直接连接到对应受控部件上,信号有效的时间由微操作逻辑表达式,即节拍、指令译码结果、以及其他一些状态决定。控制器正是通过这种方法在时间上和空间上完成对指令流和数据流的正确控制。

控制器按其组成和实现方式可以分为组合逻辑控制器和微程序控制器两种。从控制器的基本组成来看,两种控制器有许多相同的部件,如指令计数器、指令寄存器、指令译码器、时钟及启停控制线路、信息传送通路等;两种控制器的主要差别是微操作控制部件,它反映了不同的设计方法和设计原理。

一般来说,组合逻辑控制器的设计过程可以分为以下步骤:

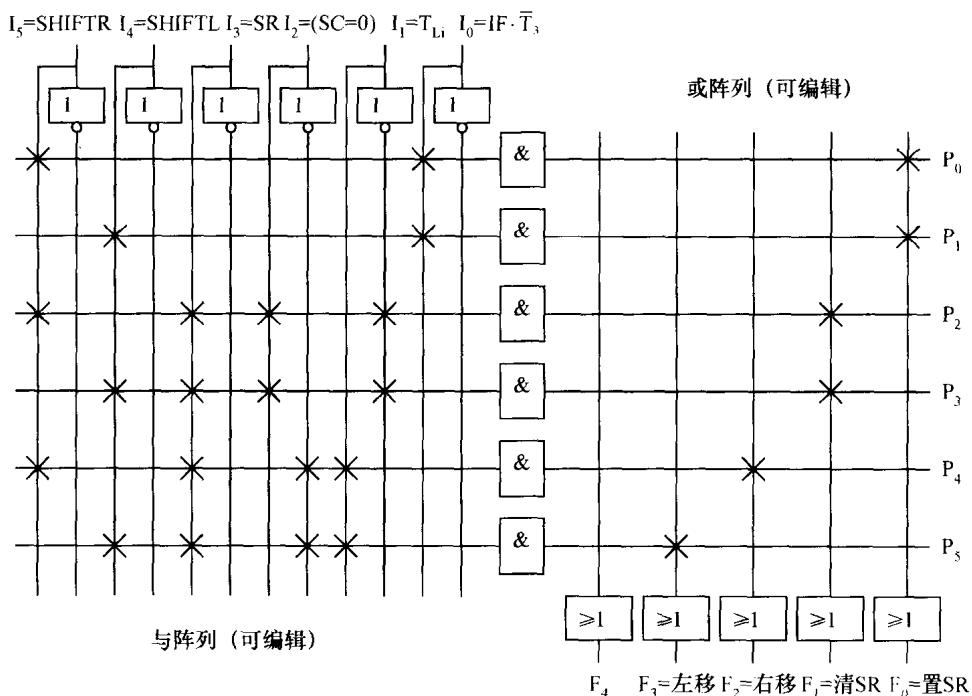


图 4.18 编程后的 PLA

1. 明确设计环境

在进行控制器设计之前,首先需要明确设计环境,包括指令系统、主存的存储周期、时钟周期、部件的结构等。对于指令系统,应该明确所设计计算机的数据表示种类、指令的数量和格式、寄存器的作用及数量、是否设置各种寻址方式等问题。明确主存的存储周期,以便选择时钟周期以及决定每种动作所需节拍的数量。而部件的结构,例如运算器是采用哪种内部总线结构,采用锁存器还是多路选择器向 ALU 提供数据,这些问题都必将影响到指令执行的流程。

2. 分析指令流程

由于指令的功能在一个节拍中不能完成,故需要对指令的流程进行分析。指令的执行过程就是多个部件在控制器控制下按序执行微操作的过程。从部件的角度看,那些最基本的,最小的具有独立意义的操作被称为微操作。指令流程分析的目的是根据指令功能要求和各部件间的数据通路,为各种指令的微操作序列选定适当的节拍信号。

3. 编制指令操作时间表

根据指令流程,编制指令操作时间表是一项非常细致、繁琐的工作。在编制过程中,微操作的时序安排与调整,控制器的结构和数据通路设置往往需要交叉与反复,不断地进行修

改与完善。在编制微操作时间表时,应注意以下几个问题:

(1) 为确保指令的正确性,在基本指令周期内要尽量安排完所有指令的微操作。如果安排有困难,一般需要增加信息传送通路,以提高微操作的并行性;或者增加附加操作周期,如间址周期;或者采用局部控制的方法来解决。

(2) 安排微操作时,应避免引起功能部件或数据通路的冲突。如一个存储周期尚未结束,又发出存储器读写命令,这是不允许的。另外,安排微操作不应过分集中在某些节拍中,以免引起冲突,否则就需要增加设备来解决冲突问题。

(3) 安排微操作对功能部件的要求要合理,不应造成对某些部件速度要求过高或者使某些设备的负载过重。例如并行进位加法操作,由于存在进位问题,要求在一个节拍内完成加法和结果数据传送这两个操作,时间上可能比较紧张,如果将这两个操作安排不同节拍完成可能是个比较好的方案。

(4) 精心设计公操作。公操作是指各条指令都有的微操作,例如“PC + 1”等微操作。精心设计公操作,有利于简化微操作控制部件设计,减少逻辑级延迟时间和节省硬件设备,同时也便于突出各条指令中有特殊性的一些微操作,不易出错。公操作的操作时间表安排还可以考虑有关控制器全局的一些问题,例如中断周期、间址周期的微操作安排与控制等。

4. 形成微操作表

本来有了指令的微操作时间表,就可以进行各种微操作信号的逻辑综合了。但是实际计算机的指令操作时间表是很庞大、复杂的,一个微操作信号也可以出现在不同的指令的不同节拍中,几百个微操作信号直接在操作时间表上进行逻辑综合,容易造成遗漏和错误,同时也便检查。为此,采用微操作表的方法可以清晰、有条理地进行微操作信号的综合。微操作表的编制是以微操作为条目,表示在哪个节拍有哪些指令要执行这个微操作。

5. 进行微操作的逻辑综合

根据微操作表,在节拍信号和指令译码信号的作用下,可以产生各种微操作控制信号,形成每个微操作对应的逻辑表达式。将这些表达式变换为适于所采用门电路特点的表达式,就可以画出微操作控制部件的逻辑图。

6. 产生各个微操作的逻辑线路

组合逻辑也称为硬件控制逻辑。如果使用标准的门电路,每个微操作控制信号都由逻辑门电路产生,那么组合逻辑控制器将是由大量逻辑门和触发器电路构成的非常复杂而庞大的时序逻辑网络,其状态的数目和输入组合数将非常庞大,以至于难以分析、调试、维修和生产,不利于计算机设计自动化和采用 VLSI 设计。为此,需要使用 PLD 技术来解决微操作控制部件设计的规整化问题。PLD 最大的特点是可编程,即通过改变内部逻辑单元的连接方式达到实现所需功能的目的,这样就解决了组合逻辑控制器更改与调试困难的问题。另外,目前流行的复杂可编程逻辑器件(Complex Programmable Logic Device, CPLD)、现场可编

程门阵列(Field Programmable Gate Array, FPGA)可以通过输入原理图或者使用硬件描述语言进行数字系统的设计;通过软件仿真,还可以验证设计的正确性。在印制电路板(Printed Circuit Board, PCB)上实现硬件电路后,还可以通过PLD的在线修改能力修改设计,而不必改动硬件电路。

相比微程序控制器,组合逻辑控制器设计、调试与维修都比较困难,由于设计条件不规整也较难以实现设计自动化。一旦机器研制定型后,想要修改或者扩展某些指令功能比较困难。组合逻辑控制器的优点是可以利用逻辑设计来控制逻辑级数以满足机器快速性的要求。

下面将介绍微程序控制器的基本组成、工作原理和设计中要考虑的问题。

4.4 微程序控制器技术

采用组合逻辑电路设计控制器的技术,不但在计算机系统设计中使用,在一些逻辑控制电路中同样也使用。这种方法在指令系统比较复杂时,设计复杂性也随之增大。另外,复杂控制器的设计方法难以形式化,设计效率较低。为了使计算机的控制器的设计可以和程序设计一样规范,人们发明了控制器的微程序设计方法。

4.4.1 微程序控制基本原理

微程序控制的基本思想是比较简单的,实质是将指令执行过程中的各个微操作用微指令表示,固化于存储单元,然后编制对指令机器进行取指、译码和执行的微指令序列(即微程序),执行该微程序,就完成该机器指令的执行。

如果一台计算机的指令系统共有 n 个微操作,每一个微操作可用一个二进制位控制。当该位为1时,则执行该位所表示的微操作,为0时,就不执行这个微操作。这 n 个控制位在一起就构成了一条微指令。例如,第0位表示PC复位(清0)、第1位表示PC加“1”、第2位表示将IR的内容送给PC等等,这种每一位代表一个微操作的微指令称为直接控制编码微指令,如图4.19所示。



图 4.19 具有 n 个微操作的直接控制编码微指令

在微指令的基础上,对于构成一条指令的一组微操作,就可以用一组微指令来描述,这一组微指令,就构成了一段微程序。微程序和程序一样,放在存储器中,这个存放微程序的存储器称为控制存储器,简称控存(Control Memory, CM)。控制器处理一条指令的工作过程,就是启动这条指令在控制存储器中所对应的微程序,一条一条地执行微指令的过程。这个过程可以通过图4.20说明,图中的微程序顺序控制逻辑用以产生微程序中的地址序列,相当于程序执行时程序计数器PC的功能。采用这种方法设计的控制器称之为微程序控制

的控制器,通常称为微程序控制器。

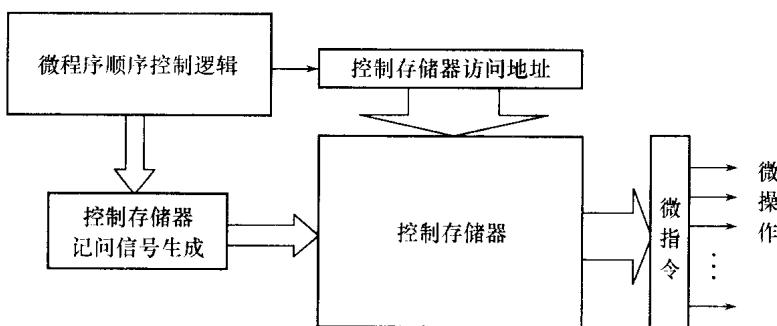


图 4.20 微程序控制器的工作原理

同组合逻辑控制相比,微程序控制具有规整性、灵活性、可维性和便于形式化描述等优点,它用规整的存储控制逻辑代替了复杂,不规整的组合控制逻辑,简化了控制部件的设计和实现,并使控制部件更易于调试和修改。微程序设计技术是用软件的程序设计方法来设计硬件。硬件的功能和操作全由微程序确定。微程序有时可称为“固件(Firmware)”,它兼有硬件和软件的某些特征。微程序控制器控制执行的速度远不及组合逻辑控制器快。

4.4.2 Wilkes 模型

微程序设计的概念和原理是由英国剑桥大学的 M. V. Wilkes 教授于 1951 年首先提出来的。1953 年他又给出了微程序工作模型和一些详细研究的结果,这个模型称之为 Wilkes 模型。Wilkes 模型不但给出了现在使用的微程序控制器的基本概念,而且它们的工作原理是相似的。图 4.21 是 Wilkes 模型的结构图。

Wilkes 模型设计的初衷是提出一种控制器设计的系统化的方法。它的输入是指令寄存器 IR 中的操作码和机器状态标志,其输出为控制信号。即它根据不同指令和机器的状态,产生相应的控制信号序列。微程序控制器由以下几个主要部分构成:

(1) 控制存储器 CM

控制存储器用来存放实现整个指令系统的所有微程序。由于该存储器主要存放控制命令与下一条要执行的微指令地址,所以被称为控制存储器 CM。一般计算机的指令系统是固定的,所以实现指令系统的微程序也是固定的,于是控制存储器可以用只读存储器实现。由于机器内控制信号数量比较多,再加上决定下条微指令地址的地址码有一定宽度,所以一般控制存储器的字长比机器字长要长得多。控制存储器的容量视机器指令系统而定,即取决于微程序的数量,一般为几 K ~ 几 M 微指令字。对控制存储器的要求是速度快。

(2) 微指令寄存器 μIR

微指令寄存器用来存放从控制存储器中读出的当前微指令。微指令中包含两个字段;微地址码字段和控制信号字段。控制信号字段将操作控制信号送到控制信号线上,微地址

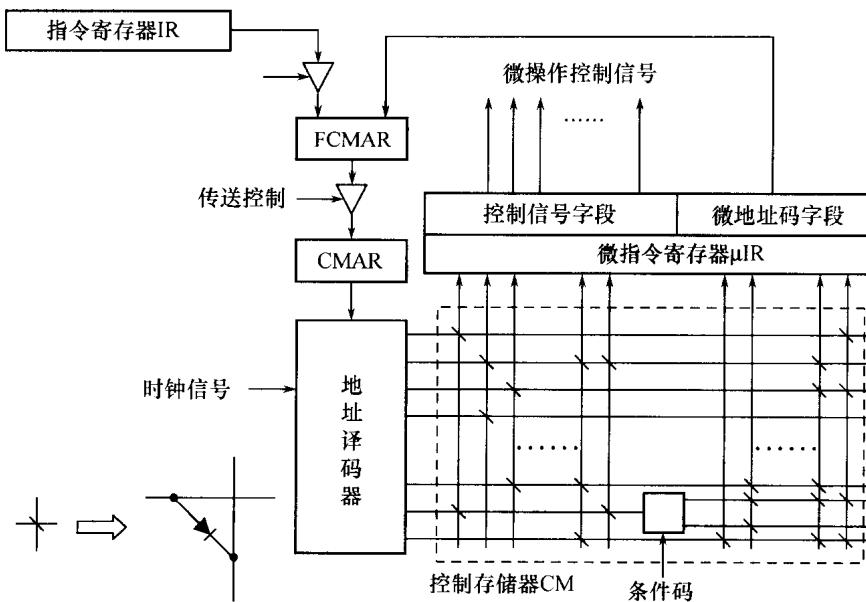


图 4.21 Wilkes 微程序控制模型

码字段用于控制下一条微指令地址的产生。

(3) 控制存储器地址寄存器 CMAR

控制存储器地址寄存器又称为微地址寄存器 μ AR, 用来存放将要访问的下条微指令的地址。

(4) 控存地址缓冲寄存器 FCMAR

控存地址缓冲寄存器又称为微地址形成逻辑 AF。它根据微指令中的微地址码字段信息以及机器的状态标志产生下一条微指令的地址。

FCMAR 还有一个非常重要的功能, 就是进行地址变换。地址变换逻辑根据机器指令的操作码变换产生对应于该指令的微程序的人口地址。

(5) 条件码接收电路 S

条件码接收电路 S 用来接受外部信号, 控制微程序的流向。

(6) 地址译码器

地址译码器的功能与一般存储器的地址译码器相同。它将 CMAR 中的下条微指令地址译码, 以选择 CM 对应的存储单元, 将其中的微指令读出送微指令寄存器 μ IR。

下面通过指令的执行过程来讨论 Wilkes 模型的工作过程。

(1) 在采用微程序控制的计算机中, 每条机器指令对应于一段微程序。这些微程序编好后放入 CM 中。

(2) 根据从指令寄存器 IR 来的指令操作码, 经过一定的变换, 就得到这条指令的微程

序入口的微地址码,以此作为 FCMAR 的初值。

(3) 由传送控制将 FCMAR 的内容送到 CMAR 中,经过地址译码器译码,CM 矩阵中的某一行被启动,这一行就是一条微指令,它由控制信号字段和微地址码字段两部分组成。在 CM 中微指令的控制信号部分,有连接的点就产生了相应的控制信号,这个信号就是要执行的微命令。在 CM 中微地址码部分,由连接和非连接的点形成编码,就是后续微指令的地址,这个编码作为下一条微指令在 CM 中的地址被送到 FCMAR。

(4) 重复步骤(3),直到一条指令对应的微程序执行完毕。当一条指令的微程序执行完毕,接下来首先要用一段微程序取下一条指令存放在指令寄存器 IR 中,然后根据该指令的操作码转移到相应的微程序去执行。实际上在微指令的控制信号中,包括了控制 FCMAR 选择的信号:即是接收由指令寄存器 IR 中的指令操作码通过某些变换得到的微地址码,还是从 μIR 的微地址码部分接收地址。

(5) Wilkes 模型为了具有适应外部信号能力,设计了条件码接收电路 S。这个电路实际上实现了微程序的条件转移功能,它可以根据外部信号的状态来选择后续微指令。

微程序控制器就是通过逐条取出微指令,发出各种微操作,实现将机器指令从主存中取出、分析并执行。

这种经典的 Wilkes 模型存在的主要技术问题是矩阵规模问题。当微命令数量很大时,产生控制信号的存储矩阵过于庞大;同时,微指令中的微地址码部分也比较长。这些问题在后来的实际应用中都得到了改进。

下面讨论一些有关微程序的基本属性和基本概念。

(1) 微命令和微操作

一台计算机基本上可以划分为两大部分:控制部件和执行部件。控制器就是控制部件,而运算器、存储器、外部设备相对控制器来讲,就是执行部件。

微命令是构成控制信号序列的最小单位。例如,打开或关闭某个控制门的电位信号,某个寄存器的打入脉冲等。微命令由控制部件通过控制线向执行部件发出。微操作是执行部件接受微命令后所进行的最基本的操作。微命令是微操作的控制信号,而微操作是微命令控制的操作过程。对控制部件体现为微命令,对执行部件体现为微操作,实质上是同一信号在两个部件上的不同称谓。

微操作是执行部件中最基本的操作。由于数据通路的结构关系,微操作可分为相容的和互斥的两种。所谓相容的微操作,是指在同时或同一个 CPU 时钟周期内可并行执行的微操作。所谓互斥的微操作,是指不能在同时或不能在同一个 CPU 时钟周期内并行执行的微操作。例如,实例计算机中“清除 R”和“MBR 送 IR”两个微操作就是相容的;而“置 IF”和“置 EX”就是互斥的。

(2) 微指令和微指令周期

微指令是一组实现一定操作功能控制的微命令的组合,通常用二进制编码表示。如前

所述,控制器的实质工作,就是按时间顺序产生控制信号序列。从控制器发出的控制信号或者为开(即表示执行相应的微操作)或者为关,这可以用一位二进制来表示。当该位为“1”时,就表示执行相应的微操作。将这些二进制位组织起来,就构成了微指令。

一条微指令通常包含:

- **微操作码字段:**又称为操作控制信号字段,用以指定在一条微指令中同时产生的各种操作控制信号。
- **微地址码字段:**又称为顺序控制字段,用于控制或决定下一条要执行的微指令地址。
- **其他有关信息:**如常数字段,标志字段等。常数字段用于提供操作需要的数据,标志字段用于提供某些特征信息。

和指令一样,每条微指令的操作周期是微程序控制器的基础。微指令周期通常是指从控制存储器读一条微指令并执行完相应的微操作所需的时间。微指令周期和指令周期之间的差别是,微指令周期的时间一般是固定的,指令周期往往是可以变化的。微指令周期是微程序控制的重要指标之一。

(3) 微程序

微程序是微指令的有序集合。微程序设计就是将传统的程序设计方法运用到控制逻辑的设计中。控制逻辑的本质是控制计算机内部的信息传送以及它们之间的相互联系。因此,微程序设计就是用类似程序设计的方法,组织和控制计算机内部信息的传送和互相的联系。

最后,再讨论机器指令与微指令,程序与微程序,主存储器与控制存储器的关系。

机器指令是提供给用户编程的基本单位,微指令则是为实现机器指令操作的一系列微命令的组合。一条机器指令对应由若干条微指令组成的微程序,机器指令由微指令解释执行。

程序是由机器指令构成的,对用户程序而言,是为某项任务编制并放在主存储器中,允许修改。微程序由微指令构成,一条机器指令对应一段微程序,所以微程序是用于描述机器指令的。微程序是在设计计算机时将它预先编制好,并存入控制存储器中,通常不允许用户修改。

主存储器中存放的是系统程序和用户程序,容量很大。控制存储器中存放的是对应于机器指令系统的微程序,实现机器的指令系统,容量有限。

虽然计算机出现不久,微程序技术就产生了,但由于当时硬件条件的限制,一时找不到合适的控制存储器器件,因而没能得到广泛的应用。1964年,IBM公司推出的IBM 360系列计算机才成功地采用了微程序设计技术,开拓了微程序设计技术普及发展的新局面。此后,随着半导体技术的飞速发展;廉价、快速、可靠的半导体只读型存储器的出现又大大促进了微程序设计技术的发展和应用,计算机越来越多地采用了微程序设计技术。

当然,微程序控制也有其不足之处,就是速度比组合逻辑控制慢。典型的RISC一般不采用微程序控制。而目前的CISC处理器,如Intel的x86系列微处理器,自Pentium Pro以来,均采用两种控制逻辑,即常用的简单指令采用组合逻辑控制实现,如寄存器之间的数据

传送指令等,而对于不常用的和复杂的指令则采用微程序控制实现。

4.4.3 实例计算机的微程序实现

在本小节中,将结合实例计算机的设计,比较全面地讨论微程序控制的概念、方法、技术等问题。

根据前面对实例计算机的讨论,为了简化过程,仅仅将 ADD、CLEAR、SUBTRACT、STORE、BRANCH(特指无条件转移)这五条指令使用微程序技术实现。

1. 确定微指令的结构

首先根据前面章节描述的指令功能和微操作序列列出这五条指令的控制流如表 4.13 所示。注意,这里将微指令周期与组合逻辑控制器的节拍取得一致,只是引用组合逻辑控制器中同步控制的思想,便于两者的对比和读者的理解。微程序控制器中无节拍问题,只有微指令的执行顺序问题。表 4.13 给出了这五条指令控制流的描述。

表 4.13 五条指令的控制流

指 令 节 拍	ADD	CLEAR	SUBTRACT	STORE	BRANCH
IF · T ₀		MAR←PC,	R←1		
IF · T ₁		IR←MBR,	R←0		
IF · T ₂		PC←PC + 1			
IF · T ₃			IF←0, EX←1		
EX · T ₀	MAR←IR, R←1		MAR←IR, R←1	MAR←IR, W←1, MBR←AC	
EX · T ₁	R←0	R←0	W←0		
EX · T ₂	ALU←AC + MBR		ALU←AC - MBR		PC←IR
EX · T ₃	AC←ALU	AC←0	AC←ALU		
EX · T ₃			EX←0, IF←1		

根据表 4.13 中的微操作,就得到表 4.14 所示的微命令清单,同时为每个微命令分配一个控制信号位。

完成五条指令需产生 18 个微命令。此外,微程序控制器本身还需一些微命令。这些微命令主要目的是为了控制产生微指令的地址,表 4.15 列出了这些微命令。表中的 MAP (IR) 操作是根据指令操作码,产生一个偏移量,作为每条指令所对应的微程序的入口。在进行微程序编码以后,就可以确定每一条指令的具体偏移量。对应实例计算机这五条指令,共需 21 个控制信号位。

表 4.14 五条指令的微命令清单

微 命 令	控制信号位	含 义
AC←0	1	累加器清零
AC←ALU	2	运算器结果送累加器
ALU←AC + MBR	3	将累加器和存储缓冲寄存器送运算器,启动加法
ALU←AC - MBR	4	将累加器和存储缓冲寄存器送运算器,启动减法
EX←0	5	清除执行标志
EX←1	6	设置执行标志
IF←0	7	清除取指令标志
IF←1	8	设置取指令标志
IR←MBR	9	存储缓冲寄存器送指令寄存器
MAR←IR	10	指令寄存器送存储地址寄存器
MAR←PC	11	程序计数器送存储地址寄存器
MBR←AC	12	累加器送存储缓冲寄存器
PC←IR	13	指令寄存器送程序计数器
PC←PC + 1	14	程序计数器加 1
R←0	15	清除读存储器标志
R←1	16	设置读存储器标志
W←0	17	清除写存储器标志
W←1	18	设置写存储器标志

表 4.15 用于微程序控制器的一些微命令

微 命 令	控制信号位	含 义
FCMAR←0	19	FCMAR 清零, 取下一条指令
FCMAR←CM()	20	微地址码送 FCMAR
FCMAR←FCMAR + MAP(IR)	21	根据指令对 FCMAR 进行微地址码变换

为了其他指令中可能产生的新的微操作, 在微指令中一般留有一些空余的控制信号位。现在规定控制信号位为 26 位, 用去 21 个控制信号位, 余下的 22 到 26 位目前空闲。

下面分析一下控制存储矩阵中的单元数, 由此确定微指令的地址位数。现在分析五条指令, 每条指令最多由 8 条微指令构成, 共计不超过 $5 \times 8 = 40$ 个单元。保留一部分空单元, 假设共有 64 个单元, 这样微地址码字段 SCF 为 6 位。

结合上面的控制信号的位数, 微指令宽度可以设计为 32 位, 格式如图 4.22 所示。

2. 微程序设计和微代码

设计好微指令以后, 就可以编制指令系统中每一条指令的微程序。

根据表 4.13 可以知道, 五条指令在取指令阶段的工作是完全相同的, 每条指令在执行

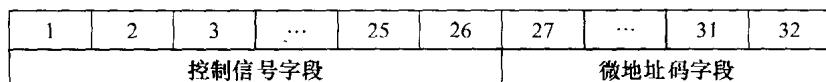


图 4.22 实例计算机的微指令格式

阶段的工作是有差别的。因此可以编写一个取指令的公共部分,然后根据不同的指令进行转移。表 4.16 给出了这五条指令的微程序。

表 4.16 实例计算机中五条指令的微程序

地址	控制信号(微命令)	SCF	说明
0	MAR←PC, R←1, FCMAR←CM(27..32)	1	
1	IR←MBR, R←0, FCMAR←CM(27..32)	2	
2	PC←PC + 1, FCMAR←CM(27..32)	3	
3	IF←0, EX←1, FCMAR←FCMAR + MAP(IR)		按 MAP(IR), 转 4~9 行
4	FCMAR←CM(27..32)	15	转向 ADD
5	FCMAR←CM(27..32)	19	转向 CLEAR
6	FCMAR←CM(27..32)	20	转向 SUBTRACT
7	FCMAR←CM(27..32)	24	转向 STORE
8	FCMAR←CM(27..32)	27	转向 BRANCH
9	(从此到 14 号单元空闲, 备新指令使用)		
10			
11			
12			
13			
14			
15	MAR←IR, R←1, FCMAR←CM(27..32)	16	ADD 指令
16	R←0, FCMAR←CM(27..32)	17	
17	ALU←AC + MBR, FCMAR←CM(27..32)	18	
18	EX←0, IF←1, AC←ALU, FCMAR←0	0	返回
19	EX←0, IF←1, AC←0, FCMAR←0	0	CLEAR 指令, 返回
20	MAR←IR, R←1, FCMAR←CM(27..32)	21	SUBTRACT 指令
21	R←0, FCMAR←CM(27..32)	22	
22	ALU←AC - MBR, FCMAR←CM(27..32)	23	
23	EX←0, IF←1, AC←ALU, FCMAR←0	0	返回
24	MAR←IR, W←1, MBR←AC, FCMAR←CM(27..32)	25	STORE 指令
25	W←0, FCMAR←CM(27..32)	26	
26	EX←0, IF←1, FCMAR←0	0	返回
27	PC←IR, FCMAR←CM(27..32)	28	BRANCH 指令
28	EX←0, IF←1, FCMAR←0	0	返回

(续表)

地址	控制信号(微命令)	SCF	说明
29	(以下目前空闲)		
30			

有了微程序,就可以非常容易地写出控存中的微指令代码。图 4.23 就是现在实现的代码点。其中只有有控制信号的地方填了“1”,所有的“0”都没有填写。

控制信号矩阵												S	C	F										
控 存	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	
地 址	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2		
0 0											1			1			1						1	
1 0												1			1			1						1
2 0															1			1						1
3 0															1				1					1
4 0																	1							1
5 0																1								1
6 0																	1							1
7 0																		1						1
8 0																			1					1
9 0																								0
1 0																								0
2 0																								0
3 0																								0
4 0																								0
5 0																								0
6 0																								0
7 0																								0
8 0																								0
9 0																								0
1 1																								0
2 1																								0
3 1																								0
4 1																								0
5 1																								0
6 1																								0
7 1																								0
8 1																								0
9 1																								0
1 2																								0
2 2																								0
3 2																								0
4 2																								0
5 2																								0
6 2																								0
7 2																								0
8 2																								0
9 2																								0

图 4.23 五条指令的微程序码点

3. 微程序控制器的实现

微程序控制器的控制信号由控制存储器发出,而不是由组合逻辑电路产生。

图 4.24 给出了实例计算机的控制系统方框图。其中与组合逻辑控制器相比,增加了 3 个控制 FCMAR 操作的信号。MAP(IR) 电路完成指令操作码的顺序编号,对于实例计算机的 5 条指令,它们对应的映射关系如表 4.17 所示。

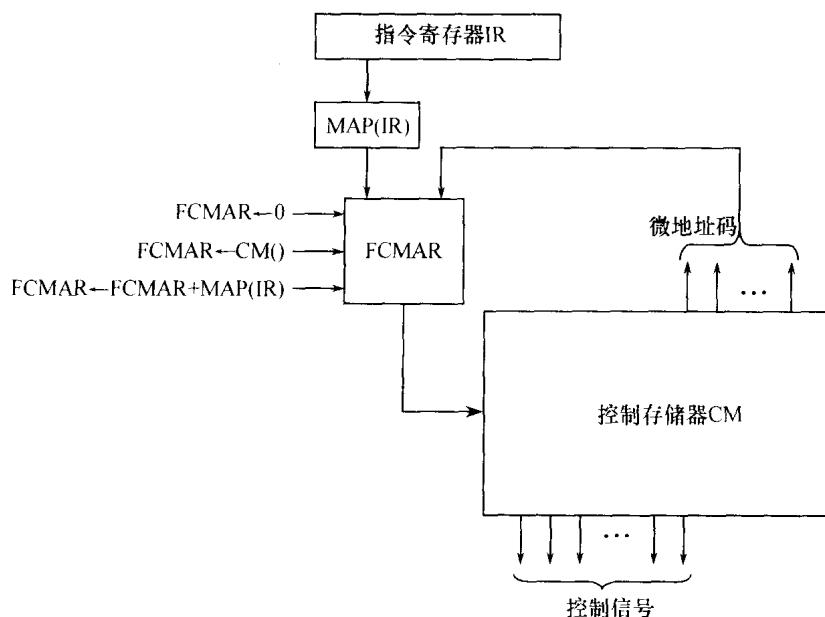


图 4.24 采用微程序控制的实例计算机的控制系统方框图

表 4.17 实例计算机的微程序控制其中 MAP(IR) 映射函数

IR	ADD	CLEAR	SUBTRACT	STORE	BRANCH
MAP(IR)	1	2	3	4	5

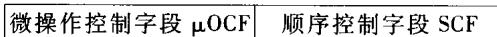
本小节讨论实现的微程序控制是最直接的方法,它的优点是简单直观,不足之处是CM的利用率非常低,对于一些比较复杂的控制信号处理起来比较麻烦,没有条件分支处理,对于微地址码的处理效率也不高,等等。因此,在下一节,将在现有知识的基础上,进一步讨论关于微程序设计的一些技术问题。

4.5 微程序的技术问题

微指令的结构格式、微程序的顺序控制方法及微指令的执行方式直接影响微程序控制器的结构和控制过程,它们都是微程序控制器设计要解决的关键技术问题。

4.5.1 微命令控制信号编码与微指令格式

一条微指令的功能有两个:给出微命令控制信号和后续微指令地址。因此微指令的一般格式中包括两部分的信息:微命令控制信号信息和后续微指令地址信息,分别称为微操作控制字段 μOCF 和顺序控制字段 SCF,格式为:



设计微指令结构时,所追求的目标是:缩短微指令字长度、减少微程序长度、提高微程序的执行速度、以及保证微程序设计的灵活性。

这里首先讨论微指令控制信号字段的编码方法以及微指令格式。

1. 微命令的控制信号编码

微命令的控制信号编码方法有直接控制编码、最短字长编码、分段直接编码、以及分段间接编码 4 种。

(1) 直接控制编码

所谓直接控制编码,就是在微指令的微操作控制字段 μOCF 中,每一个微命令都用一位信息表示,直接对应于一种微操作。设计微指令时,选用或不选用某个微命令,只要将表示该微命令的相应位置成“1”或“0”就可以了。因此,微命令的产生不必经过译码,可从微操作控制字段直接得到,故又称为不译码法,如图 4.25 所示。前面的实例计算机就是采用这种编码方法实现的。

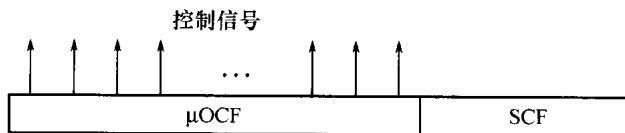


图 4.25 直接控制编码

这种编码法的优点是控制简单、直观,操作并行性最好,从而可以提高速度。其缺点是微指令字太长,控制存储器的容量过大且微指令字利用效率很低,这种编码方法只适用于结构简单(微命令少)或速度要求很高的高速数字控制部件。

(2) 最短字长编码

上述直接控制编码法是所有 4 种编码方法中的一个极端,最短字长编码则是另一个极端。这种编码法是将所有的微命令进行统一的二进制编码,用不同的码点组合去表示不同的微命令,通过译码器产生微操作控制信号,如图 4.26 所示。

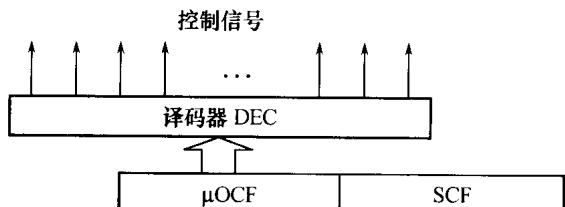


图 4.26 最短字长编码

一般情况下,在没有微操作信号时,应规定一空操作。因此微操作控制字段 μOCF 的长度 L 与微命令个数 N 的关系为:

$$L = \lceil \log_2(N+1) \rceil$$

最短字长编码的优点是微指令字长很短,但是因为它每次只能产生一个微命令,所有微命令不能够并行,难以提高微指令的执行效率,故在实际中很少采用。

(3) 分段直接编码

这是一种介于直接控制编码和最短字长编码之间的编码方法。它将微操作控制字段 μOCF 划分为若干个小字段,每个小字段单独编码,每个码点表示一种微命令。执行采用这种编码的微指令时,每个小字段独立译码,产生一个微操作控制信号,如图 4.27 所示。显然,在这种编码方法中,各小字段之间的微命令可以同时进行,但一个小字段中所表示的各微命令则是每次只能执行一个。

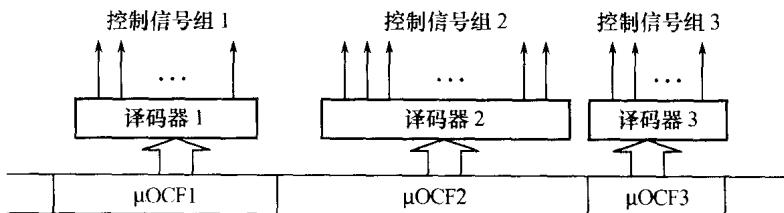


图 4.27 分段直接编码

微指令字的分段可以按功能分段,也可以按资源分段。按功能分段,是对于机器中的每一种功能类型,分配一个字段。按资源分段,是把机器看成是由一组独立资源或部件(如 I/O, 存储器, ALU)构成,给每一种资源部件分配一个字段。实际使用中,可把两者结合起来使用。

分段时,一般把互斥的微命令分在同一字段内,把相容性微操作分在不同的段内。按功能部件或公用数据传送通路来分段往往能较容易地做到这一点。因为对于同一功能部件或公用数据通路的各种微操作往往是互斥的,不会在一个微指令周期内有两个或两个以上的微操作同时进行。

分段直接编码法吸收了直接控制编码和最短字长编码两种方法的优点,既能缩短微指令字长,又有较高的并行性,执行速度比较快,因此,得到了广泛的应用,IBM 370 系列机和 VAX - 11 系列机都采用了此编码法。

直接控制编码和最短字长编码可看成是分段直接编码的特例。前者相当于每位为一段,微操作控制位有几位就有几段;后者则相当于总共只有一段的情况。

(4) 分段间接编码法

分段间接编码是在分段直接编码基础上,进一步缩短微指令字长的一种编码方法。在这种编码方法中,某些参与编码的微命令不能由一个控制字段直接定义,而需要两个或两个以上的控制字段来定义。或者说,一个控制字段的某些微操作需要另外一个控制字段来解

释才能确定,这就是分段间接编码法。

如有一分段间接编码指令如下:

S_0	...	S_i	...	S_j
-------	-----	-------	-----	-------

其控制字段 S_i, S_j 中编码的码点定义何种微操作,还要看解释字段 S_0 的码点内容而定。这样, S_i, S_j 中一个码点可表示多种微操作,从而可以缩短指令长度。当然,一个解释字段要同时解释多个控制字段,才能有效的缩短字长。解释字段应具有某些微操作分类的特征,如二进制运算微操作与十进制运算微操作;控制 CPU 的微操作与控制 I/O 通道的微操作等等。

各种间接编码方案的共同的特点是在并行性、速度和微指令字长等因素之间进行权衡。好的编码方法在影响速度很小情况下,能明显缩短微指令字长。一般情况下,间接编码只作为直接编码的辅助手段。

(5) 常数源字段 E

在微指令中,一般设有常数源字段 E。它的基本用途如同指令中的直接操作数一样,是供设计者在填写微指令时作为要使用的常数;它的另一种用途是参与其他控制字段的间接编码,以减少微指令字长和增加微指令的灵活性。

带有常数源字段 E 的微指令格式一般为:

μ OCF	E	SCF
-----------	---	-----

在执行微程序过程中,往往需要用到各种形式的常数。如用来建立状态触发器、计数器的初值,以适应微程序转移的需要;用来作为主存或其他存储器的部分地址;用作常数参与某些运算等。总之,设置 E 字段,可以节省微操作的数量,减少硬件,增加微程序设计灵活性。

2. 微指令格式

微指令格式的设计是微程序设计的主要部分,它直接影响微程序控制器的结构和微程序的编制,也直接影响计算机的速度和控制存储器的容量。微指令格式的设计除了要实现计算机的整个指令系统之外,还要考虑具体的数据通路结构、控制存储器的速度以及微程序的编制等因素。不同机器有不同的微指令格式,就其共性来说,大致可归纳为两大类,即水平型微指令和垂直型微指令。微命令的编码方法与微指令格式密切相关。

(1) 水平型微指令

水平型微指令是一种广义的说法,并没有统一的精确的定义,一般来说它具有如下特征。

- 微指令字较长。一般为几十位到 100 位左右,有的长达 200 多位。一般来说,机器规模越大,速度越快,采用的微指令字就越长。
- 微指令中的微操作有高度的并行性,即在一个微指令周期中,一次能并行执行多个微命令。因而能充分发挥数据通路并行结构的并行操作能力。
- 微指令译码简单。一般采用直接控制编码和分段直接编码,微命令与数据通路各控

制点之间有较直接的对应关系。

采用水平型微指令编制微程序称为水平微程序设计。这种设计由于微指令的并行操作能力强,效率高,编制的微程序比较短,因此微程序的执行速度比较快,控制存储器的纵向容量小,灵活性强。其缺点是微指令字比较长,明显地增加了控制存储的横向容量。而且,水平微指令与机器指令差别很大,一般要熟悉机器结构、数据通路、时序系统以及指令的执行过程的人才能进行微程序设计。将用微程序设计语言描述的源微程序编译成水平微指令的编码比较困难、复杂,不易实现设计的自动化。

(2) 垂直型微指令

在设置微操作码字段时,一次只能将控制信息从某个源部件传送到某个目标部件处理的微指令称为垂直型微指令。其特征是:

- 微指令字短,一般为 10~20 位左右。
- 微指令的并行微操作能力有限,一条微指令只能控制数据通路的一两种信息传送。
- 微指令译码比较复杂。全部微命令用一个微操作码字段进行编码。执行时,需进行完全译码。微指令的各个二进制位与数据通路的各个控制点之间完全不存在直接对应关系。

采用垂直型微指令编制微程序称为垂直微程序设计。这种设计编制微程序较为容易,只需注意微指令的功能,而对微命令的选择和数据通路的结构则不用过多地考虑。编制的微程序规整、直观,可引用现有程序设计语言和编译程序的结果,便于实现设计的自动化。垂直微指令字较短,使控制存储器的横向容量少。其缺点是,用垂直微指令编制微程序要使用较多的微指令,微程序较长。要求控制存储器的纵向容量大。而且,垂直微指令产生微命令要经过译码,微程序执行速度慢。此外,由于垂直微指令不要求数据通路具有太多的并行能力,如果数据通路具有多种并行操作能力,则不能充分利用。

垂直型微指令与机器指令很相似,它也有操作码字段,源地址和目的地址及某些扩展操作码字段,只是比机器指令更具体些。它也有多种微指令格式,按功能分成几类,如寄存器-寄存器传送型微指令、运算控制型微指令,主存控制型微指令、移位控制型微指令,无条件转移型微指令、条件转移型微指令等,它们一起构成一个微指令系统。例如,寄存器-寄存器传送型指令的格式为

0 1 2 3	7 8	12 13 15
0 0 0	源寄存器编址	目的寄存器编址 其他

这类微指令用来把源寄存器的内容传送到目的寄存器。其中第 0~2 位为微操作码,三位“000”表示 RR 传送型微指令;3~7 位为源寄存器编址,可指定 31 个寄存器之一做为源寄存器(“00000”表示不指定寄存器);8~12 位为目的寄存器编址,可指定 31 个寄存器之一作为目的寄存器;13~15 位是“其他”字段可协助本条微指令完成其他控制功能。

又如移位控制型微指令的格式为:

0	1	2	3	6	7	10	11	12	13	14	15
0	1	1	源寄存器编址		目的寄存器编址	L	AR	CI	其他		

这类微指令将源寄存器中的数据按指定的移位方式进行移位, 移位结果送入目的寄存器。其中 0~2 位为操作码“011”; 11、12、13 位为移位方式, 可表明循环左移, 循环右移、逻辑左移、逻辑右移、算术左移、算术右移等。14、15 位可以进一步定义一些细节, 如每次移位的位数等。

从上面的讨论可以看出, 水平型微指令是面向处理机内部控制逻辑的描述, 而垂直型微指令则是面向算法的描述, 两者各有其优缺点。实际使用中, 常常兼顾两者的优点, 设计出一种混合型微指令, 采用不太长的字长又具有一定的并行控制能力。例如 IBM 370/125 机即采用这种混合微指令格式。该机采用字段编码法将 19 位的微指令字分成数目不等和功能不同的字段, 微指令只有 13 条分为装入类、ALU 类、测试和条件转移类及其他类等 4 种。

4.5.2 微指令顺序控制

微程序控制器的两个基本工作是执行微指令和微指令顺序控制, 前面已经讨论了执行部分, 也就是控制信号的编码, 下面将研究微指令顺序控制问题, 即如何从控制存储器中取得下一条微指令, 以便微程序能连续执行下去。这个问题的关键是如何确定下一条微指令的地址(也称为后继微地址), 后继微地址的确定与微程序的基本流程密切相关。由于在微程序设计中充分运用了软件的程序设计技术, 因而微程序流程中有微程序分支、微程序循环、微子程序等, 现分别讨论。

1. 微程序流程

(1) 无分支流程

无分支流程是微程序流程的一种基本形式, 其特征为一条微指令 a 执行完后, 接着必定执行下一条微指令 b, 后继微指令地址惟一确定。微地址可以是连续的, 也可以不连续。

(2) 有分支流程

有分支流程是微程序流程的一种常见形式, 如何处理好分支流程问题是微指令格式和微程序控制器设计中很关键的问题。分支流程是指一条微指令 a 执行完后, 要根据某些测试条件 $c_1 \sim c_k$ 来确定下一条要执行的微指令是 b_1, b_2, \dots 或 b_n , 如图 4.28 所示。当只有一个测试条件时, 称为两分支流程; 当有多个测试条件时, 称为多分支流程。微程序分支实际上就是微程序转移, 因此分支流程的后继微地

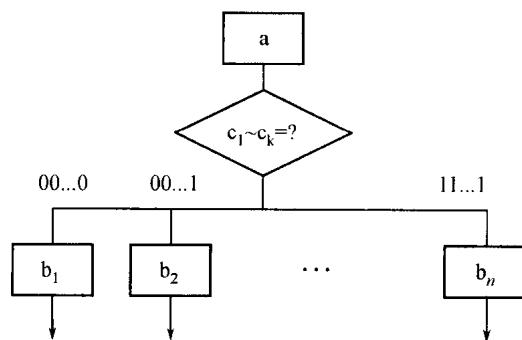


图 4.28 微程序分支流程

址不是惟一的,要根据转移条件来确定。

与程序中出现分支的情况相比,微程序中出现分支的情况较多,这是因为除转移微操作引起分支外,还有些微操作的控制条件或特征位的状态也能引起分支。例如,Booth 乘法中乘数的状态决定部分积是否加被乘数,这就会引起分支。又如除法上商的不同也会引起微程序分支。所以,微程序分支是较常见的一种流程形式。分支越多,后继微地址的形成越复杂。

(3) 微程序循环

在微程序设计中引用程序循环技术,使得某段微程序多次重复地执行,该段微程序称为微程序循环,它是简化微程序设计、缩短微程序长度和减少控存空间的有效方法。程序循环是用条件转移指令、计数转移指令等来实现,而微程序循环是通过微指令地址转移测试来实现的。

(4) 微子程序和公用微程序

在微程序设计中,借用程序设计技术引用微子程序和公用微程序也是简化微程序设计、缩短微程序长度和减少控存空间的有效方法。

微子程序是指被多个微程序调用的一段微程序,调用后返回被调用处继续执行原来的微程序。例如:浮点加减法等微程序需要调用对阶微子程序。调用微子程序的那些微程序称为微主程序。微子程序和微主程序关系类似于子程序与主程序关系,需要解决怎样从微主程序转入微子程序以及从微子程序返回微主程序问题。可设专门寄存器或堆栈来保存返回地址。

公用微程序是指被多个微程序调用的一段微程序,但是调用后并不返回被调用处执行原来的微程序。显然调用公用微程序只须解决从微程序转入公用微程序问题,不需保存返回地址。

2. 后继微地址形成

下面从两个方面讨论后继微地址的形成方法。一是如何产生每条机器指令所对应的微程序的入口地址,另一个是在微程序内如何产生后继微地址。

(1) 微程序入口地址的确定

如前所述,每一条机器指令对应着一段微程序。当“取指令”微命令(或者是微程序)把一条机器指令取到指令寄存器 IR 后,就要根据其操作码转移到对应的微程序入口地址。这是一种多分支转移。为了快速实现该多分支转移,需根据操作码一次直接产生转移地址(图 4.23 中 FCMAR 包含的地址变换逻辑),并送入微指令地址寄存器。下面要讨论的,实际上就是地址变换逻辑的实现方法。

地址变换逻辑通常有以下两种。

① 当操作码即 OP 位数与位置固定时,采用直接对应方法,即直接将操作码与微地址码的部分位对应。这时,不需要专门的硬件,只需用连线将对应位直接连接即可。在微地址码

的其余位中(微地址码长度大于操作码长度),对应位以下的低位(若存在的话)一般填0,对应位以上的高位可以固定填某个二进制常数。该常数可以看成是指出微程序入口地址的地址。不同的操作码可以看成是不同的偏移量。

② 在操作码的位数或位置不固定的情况下,需用专门的硬件来实现操作码到微地址的变换。通常是以查表的方法来实现,即用 PLA 或 ROM 实现一个表格,该表格中存放有每个操作码所对应的微程序入口地址。用指令的操作码作为该表格的输入,其输出即为该指令的微程序入口地址。实例计算机就可以采用这种方法。

有时,虽然指令操作码的位数和位置都是固定的,但为了便于微程序可以在控制存储器的任何位置存放,而且又要一次直接转移到入口地址,也往往采用这种方法。

(2) 后继微地址的产生

每条微指令执行后都必须根据要求产生后继微指令的地址。后继微指令地址的产生方法对微程序编程的灵活性以及微指令字长等都有很大的影响。

后继微地址的产生主要有两种方式:顺序-转移方式和断定方式。下面研究这两种方式的工作原理。

① 顺序转移方式

这种方法同用程序计数器 PC 来产生当前机器指令地址的方法相类似。在微程序控制器中,设置一个微程序计数器 μ PC,用于指出当前微指令的地址。在顺序执行微指令时,后继微指令地址由 μ PC 加上一个增量(通常为 1)来产生。遇到转移时,由微指令给出转移微地址,使微程序按新的顺序执行。

采用这种方式的微指令格式如下:

μ OCF	SCF	
	BAF	BCF
控制字段	地址字段	

这里地址字段分为两部分:转移控制字 BCF 和转移地址字段 BAF。BAF 用于给出转移用微地址,BCF 用于规定是顺序执行还是转移。在顺序执行时,BCF 指出后继微地址由“ μ PC + 1”产生,在转移时,BCF 指出转移地址的来源。转移地址的来源有以下三种:

- 由 BAF 确定的地址;
- 机器指令所对应的微程序的入口地址;
- 微子程序入口地址和返回地址(存放于返回地址寄存器或微堆栈中)。

BAF 字段的位数决定了转移范围的大小和灵活性。当在 BAF 位数较少的情况下,地址由 BAF 和 μ PC 组合(一般是拼接)而成,一般是用 BAF 代替 μ PC 的若干低位。这时转移目标只能在当前微指令的附近,转移地址受到限制。在 BAF 的位数与 μ PC 的位数相等的情况下,BAF 直接给出转移地址,可以转移到控制存储器的任何一个单元,灵活性好,但增加了微指令的长度。

下面通过一个例子来进一步说明顺序 - 转移方式。假设转移控制字段 BCF 为 3 位, 用于控制实现顺序执行、初始转移、无条件转移、条件转移、测试循环、转微子程序、微子程序返回等 7 种情况, 如表 4.18 所示, 图 4.29 为相应的顺序 - 转移方式控制原理图。

表 4.18 执行顺序 BCF 编码表

BCF 编码	顺序控制	测试条件	后继微地址
000	顺序执行	-	$\mu\text{PC} + 1$
001	初始转移	-	$\mu\text{PC} \leftarrow \text{OP}$
010	无条件转移	-	μPC 和 BAF 的组合
011	条件转移	成立	μPC 和 BAF 的组合
		不成立	$\mu\text{PC} + 1$
100	测试循环	成立	$\mu\text{PC} + 1$
		不成立	μPC 和 BAF 的组合
101	转微子程序	-	μPC 和 BAF 的组合
110	微子程序返回	-	$\mu\text{PC} \leftarrow \text{RR}$
111	(备用)	-	-

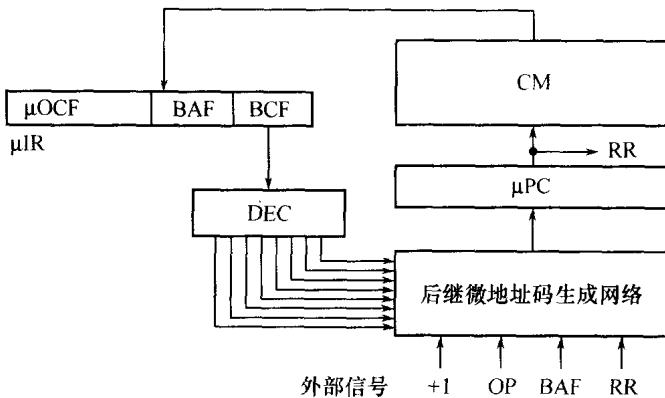


图 4.29 顺序 - 转移方式控制原理图

8 组编码的功能分别介绍如下：

- $\text{BCF} = 000$, 顺序执行, 后继微地址为 $\mu\text{PC} + 1$ 。
- $\text{BCF} = 001$, 初始转移, 后继微地址根据指令操作码变换产生, 变换后的操作码作为微地址码传送到 μPC 。
- $\text{BCF} = 010$, 无条件转移, 后继微地址为 μPC 与 BAF 字段组合形成。
- $\text{BCF} = 011$, 条件转移, 其后继微地址由转移条件的测试结果确定, 当测试条件成立时, 其后继微地址为 μPC 与 BAF 字段组合形成; 当测试条件不成立时, 其后继微地址为 $\mu\text{PC} + 1$ 。值得指出, 当测试的条件改变时, 条件转移微操作也要改变, 其转移控制码也应改变。

实际机器中有多种条件转移,故应有多个 BCF 控制码。

- $BCF = 100$, 测试循环, 后继微地址由测试循环条件来确定。当测试条件不成立, 即循环计数器内容不为零时, 循环尚需继续, 计数器减 1, 其后继微地址即循环入口微地址由 μPC 与 BAF 字段组合形成; 当测试条件成立时, 表明循环结束, 其后继微地址为 $\mu PC + 1$, 以此跳出循环。

- $BCF = 101$, 转微子程序, 其后继地址即微子程序入口微地址为 μPC 与 BAF 字段组合形成。为保存返回地址, 将 $\mu PC + 1$ 传送到返回寄存器 RR, 以备返回微主程序时使用。

- $BCF = 110$, 微子程序返回, 其后继微地址为 RR 寄存器内容, 以此实现从微子程序到微主程序之间的连接, 控制将 RR 内容传送到 μPC 。

- $BCF = 111$, 为备用控制码。

综上所述, 顺序 - 转移方式顺序控制与一般程序顺序控制很相似, 其优点是微指令中 SCF 字段较短, 后继微地址产生机构比较简单。其缺点是不利于解决两路以上的并行微程序转移, 从而不利于提高微程序执行速度。原则上讲, 2 分支转移可以解决多分支转移问题, 只是转移执行速度受到影响。顺序 - 转移控制方式另一缺点是微程序在 CM 中的物理分配不方便。其原因是既要满足微地址递增顺序规律, 又要能灵活地转向各种共用微程序与微子程序, 在实际微程序编码设计时会遇到许多矛盾, 需精心调整与安排才能解决。顺序 - 转移控制方式比较适用于速度要求不高的小型、微型计算机。

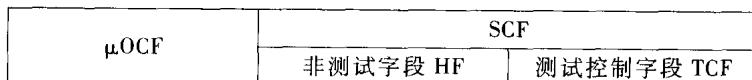
② 断定方式

断定是指后继微地址可由微程序设计者直接指定, 或者由微程序设计者指定的测试判别字段控制产生。它打破了控制存储器按连续地址分配的限制, 也打破了顺序执行微指令的控制方式。微程序在控制存储器中可以不连续存放, 每条微指令的地址必须由上一条微指令给出。

断定方式的微地址一般分为两部分, 一部分为非测试地址, 可由微程序设计者直接指定, 占高位部分; 另一部分为测试地址, 即由测试结果确定其地址值, 占低位部分。这样构成的微地址码结构为:



测试地址的位数确定了转移的并行度: 1 位为两路转移, 2 位为 4 路转移, n 位为 2^n 路转移。与这种微地址结构相对应的微指令结构为:



微指令中非测试字段 HF 直接生成微地址码中的非测试地址 HF, 测试控制字段 TCF 指出产生测试地址的测试条件。通常, 每位测试地址对应一个测试控制字段, 因实际微地址码中的测试地址可以有多位, 所以微指令中的测试控制字段也可以由多个。测试控制字段的

位数决定于测试条件的个数,两个测试源时使用1位测试控制位;不超过4个测试源时可以用2位;不超过 2^n 个测试源时可以用n位测试控制位。图4.30为具有两个测试字段的微地址码产生过程。

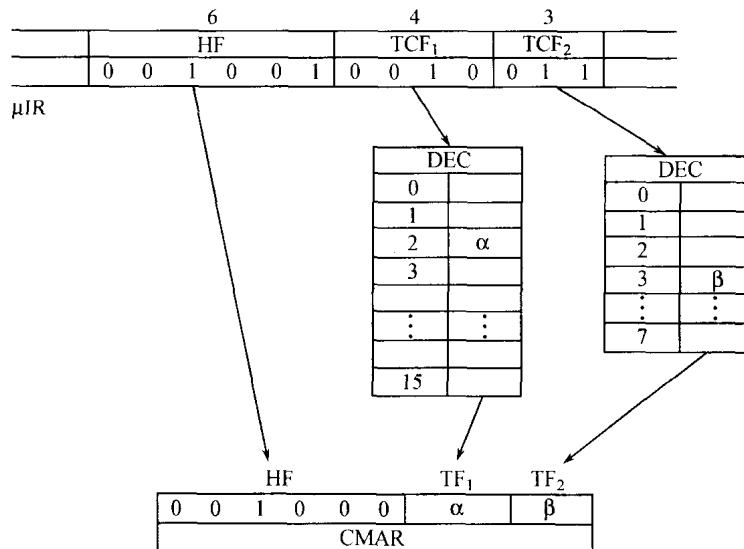


图4.30 具有两个测试字段的断定型微地址码

图中,微指令的HF为非测试字段地址,可直接送到CMAR的高6位。而 TCF_1 是 TF_1 位的测试控制字段,有4位,说明 TF_1 位可以有16个测试条件。图中 $TCF_1=2$,经译码将(的一位二进制值送 TF_1)。假定 α 表示“计数器 $CT=0$ ”的状态, $CT=0$ 时 $\alpha=1$,否则 $\alpha=0$ 。 α 送 TF_1 说明是根据“ $CT=0$ ”进行条件转移的微操作。 TCF_2 是 TF_2 的测试控制字段,有3位,说明 TF_2 位可有8个测试条件。图中 $TCF_2=3$,经译码将 β 的一位送 TF_2 。若 β 表示进位标志 $CF=1$,则 β 送 TF_2 说明是根据“ $CF=1$ ”进行条件转移的微操作。这样,后续微指令的地址由HF、 TF_1 和 TF_2 组成。其取值范围为00100000~00100011,从而决定了相应微指令具有4路并行转移功能。

断定型微地址码结构的优点是能以较短的SCF配合实现多路并行转移,提高微程序执行效率和执行速度;微程序在CM中分配物理空间很方便、灵活。其缺点是后继微地址码的生成机构比较复杂。一般在设计快速微程序控制器时使用断定型微地址码生成方式。

4.6 微指令时序控制

在确定微指令格式和微地址结构之后,微指令时序控制问题是微程序控制设计中首先要解决的问题。微指令的执行顺序由微程序指定,因此不需要像组合逻辑控制器用节拍来规定微操作的执行顺序,但微指令的执行仍离不开时序控制。时序控制方式确定后,才能着

手进行微程序控制的逻辑设计。从根本上来说,时序控制是机器系统设计问题,它与机器时钟周期、主存周期、控存读出时间以及机器速度要求密切相关。

4.6.1 微指令周期

微指令周期是指取微指令和执行微指令所需的时间 T_m 。具体来说, T_m 由以下四部分时间组成: $T_m = t_1 + t_2 + t_3 + t_r$, 其中 t_1 、 t_2 、 t_3 和 t_r 的意义说明如下:

t_1 : 从微指令打入微指令寄存器 μIR 到给出微操作控制信号所需时间。是下列各级延迟时间的总和:微指令从 CM 读入 μIR , μIR 触发器翻转稳定,微操作控制字段译码过程,直到微操作组合电路给出微操作控制信号;

t_2 : 微操作执行时间。微操作控制信号给出后,CPU 任意两部件间一次信息传送所需要的最长时间。它相当于组合逻辑控制中的节拍时间;

t_3 : 转移测试逻辑延迟时间。测试微指令形成测试条件以决定后继微地址所需的时间;

t_r : 控存 CM 的读出时间。也就是从后继微地址打入 CMAR 到微指令读到 μIR 输入线路所需时间。

由于微指令中有访存微操作,存储周期与微指令周期之间应有整数倍关系,一般存储周期为 2~4 倍微指令周期。

4.6.2 微指令周期多相控制

执行一条微指令时,对应微指令周期的不同阶段需要多个控制信号,这些控制信号需要时钟信号来同步。通常采用的同步方法是多相控制。所谓多相控制是把微周期划分为几个相(其中每相对应于单个时钟周期),从而使每个控制信号只在一个相中起作用。图 4.31 显示出典型的三项时钟控制原理图。 CP_1 , CP_2 , CP_3 为三相时钟,每相时钟周期等于微指令周

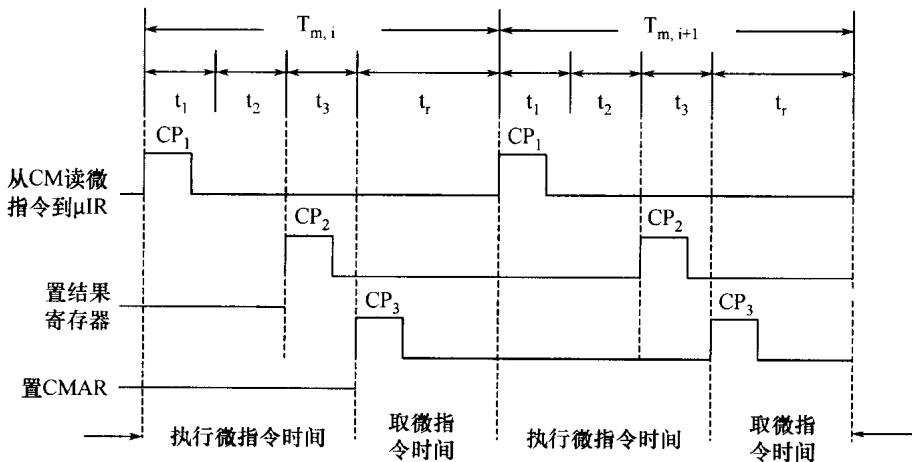


图 4.31 三项时钟控制原理图

期。 CP_1 相时钟主要用于同步 CM 读出的微指令打入 μ IR 寄存器的控制信号; CP_2 相时钟主要用于同步 $t_1 + t_2$ 时间内执行微操作的结果打入相应结果寄存器的控制信号; CP_3 相时钟主要用于同步 t_3 时间内微地址形成部件形成的后继微地址打入到 CMAR 寄存器控制信号。这种多相控制方式对微指令周期内各种控制信号的同步与控制比单相(单一时钟)控制要简单与灵活得多。因而多相控制方式被广泛采用。

小 结

控制器作为计算机的核心,能够控制存储器、运算器和输入/输出部件以及控制器自身,是通过产生对这些部件的控制信号序列来实现的。本章主要讨论的就是这些控制信号产生的时问和方法,这也是控制器设计的主要工作。

控制器按其组成和实现方式可以分为组合逻辑控制器和微程序控制器两种。从控制器的基本组成来看,两种控制器有许多相同的部件,如指令计数器、指令寄存器、指令译码器、时钟及启停控制线路、信息传送通路等;两种控制器的主要差别是微操作控制部件,它反映了不同的设计方法和设计原理。

由于速度和效率问题,现在主流计算机控制器大多数采用组合逻辑实现。但是,作为一种规范化的数字系统控制器设计和实现技术方法,目前微程序技术还在广泛应用和发展。大多数廉价的和 16 位以下的微处理器中,主要采用微程序控制器设计,同时,广泛使用的可编程控制器中,微程序还是其核心技术。与以前不同的是目前的微程序控制器系统已经大规模集成化,具有很好的开发环境。例如常见的美国 AMD(Advanced Micro Device)公司生产的 AM2900 系列位片式器件中就包含微指令顺序控制器芯片 AM2910,德克萨斯器件公司(Texas Instruments,简称 TI)开发的 TI8800 也可以进行微程序编程的 32 位微计算机系统。很多实用的系统也采用微程序设计技术,如磁盘控制器、打印机控制器、通信端口控制器等。

随着并行计算机技术的发展,为提高计算机性能,现代计算机控制器设计采用了一些新技术,例如指令流水线技术、分支预测技术、超顺序执行技术等。相关知识可参阅本系列教材《计算机系统结构》。

习 题

4.1 解释下列术语:

指令流	数据流	控制流	控制寄存器
微操作	节拍	中央控制	局部控制
微命令	微指令	微指令周期	

4.2 试述控制器基本功能。

4.3 控制器有哪几种控制方式,各有何特点?

- 4.4 试述指令周期、时钟周期、存储周期三者的关系。
- 4.5 试述节拍、节拍电位、节拍电位周期、时钟周期和指令周期的关系。
- 4.6 指令周期内节拍划分的原则是什么？
- 4.7 实例计算机中为什么要使用 T_3 信号？
- 4.8 结合实例计算机，试述局部控制和中央控制是如何相互转换的？
- 4.9 设计组合逻辑控制器的步骤有哪些？
- 4.10 试述机器指令与微指令、主存与控存、程序与微程序、指令周期与微指令周期的异同。
- 4.11 微指令编码有哪几种常用方法？它们各有何特点？
- 4.12 怎样确定多分支转移、微程序循环、微子程序的后继微地址？
- 4.13 试述顺序 - 转移型和断定型微地址的确定方法。
- 4.14 试述水平微指令格式与垂直微指令格式特点，并比较其优缺点。
- 4.15 试述组合逻辑控制器和微程序控制器的优缺点。

第五章 存 储 器

引 言

存储器是具有记忆功能的部件,用来存放程序和数据,是计算机的核心部件之一。目前绝大多数计算机硬件系统仍然是冯·诺依曼(John Von Neumann)“存储程序”式结构。“存储程序”思想的核心是将编好的程序和要加工处理的数据预先存入存储器,然后启动计算机工作,计算机则自动地从存储器中取出指令逐条执行,从而完成数值计算或非数值处理。因此存储器的速度会直接影响计算机的速度。换言之,计算机欲快速,必须配置与CPU速度匹配的高速存储器。要增强计算机的解题能力,扩大应用领域,提高工作效率并为用户提供灵活的界面,存储器除存放当前用户的程序外,还要存放系统程序和其他所需的程序及数据,所以人们希望存储器容量越大越好。存储器在位价格上显然不能太高,至少与系统中其他部件相比应是合理的。当然希望越低越好,这样性能价格比才会高。

综上所述,存储器应当满足速度快、容量大和位价格低三个要求。综合考察存储器的实现技术发现:速度越快,位价格就越高;容量越大,速度就越慢;只有容量作得大,位价格才能低。因此,要实现“容量大、价格低”的要求,应采用能提供大容量的存储技术;欲满足“速度快”的要求,又应采用昂贵且容量小的快速存储技术。很显然,若采用其中的一种技术设计存储器,存储器的设计就会陷入困境,因为对存储器的三个要求是相互矛盾的。解决矛盾的惟一方法就是采用多种存储技术构成具有层次结构的存储系统。

实际上,在现代计算机系统中,除设置与CPU速度相差一个数量级、容量很大的主存储器外,还在CPU中设置与CPU速度完全匹配的、容量足够大的高速暂存存储器和高速缓冲存储器(Cache),在外围设备中设置容量相当大甚至是海量的辅助存储器(Auxiliary Storage/Memory)。由它们构成具有层次结构的存储系统,以满足对存储器的各种需求,如图5.1所示。

本章重点阐述速度快的各类半导体存储器的基本概念、存储原理、基本结构组成、工作过

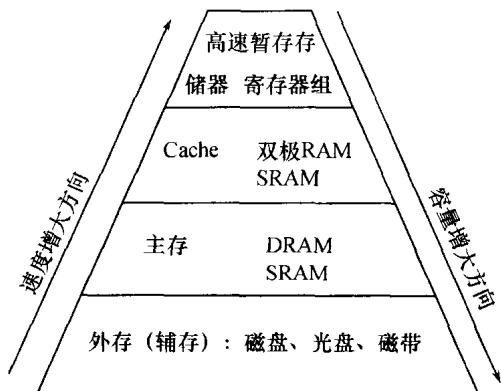


图 5.1 计算机中的存储系统

程和设计方法。阐述容量大、位价格低的磁表面存储器和光盘存储器的基本概念、存储原理和结构组成。最后讲授并行主存系统、存储系统和存储层次。目的是让读者掌握如何使用不同的存储技术,设计存储器和构建具有层次结构的存储系统。

5.1 存储器概论

5.1.1 存储器的分类

计算机问世 50 多年来,存储技术发展很快,不断出现新的存储介质和存储元器件。称记录一位二进制信息的存储介质区域或存储元器件为存储位元(Memory Cell)。从初期的汞、镍延迟线,到目前广泛使用的超大规模集成电路存储芯片,近几年又出现了光存储技术、铁电存储技术和其他正处于研制阶段的新存储技术。随着计算机的广泛应用和发展,计算机存储器种类不断增加、功能不断增强、容量大幅度提高。不仅存取方式多种多样,而且速度差异悬殊,保存信息的方式和手段亦不一样。下面从四个不同的角度对存储器进行分类。

1. 按存储器在计算机中的作用分类

(1) 高速暂存存储器(Scratch – pad Storage),亦称便笺式存储器,是由寄存器构成的,在速度上与 CPU 匹配,因此有的文献上称其为寄存器型存储器。它们用来暂时存放即刻要执行指令的、马上要使用的数据或得到的处理结果。这种类型的存储器容量一般不超过 2 KB,寄存器的位数与机器字长相同,它属于 CPU 的组成部分,如 CPU 中的先行寄存器组、后行寄存器组、指令流队列和通用寄存器组等。

(2) 高速缓冲存储器(Cache)。这个存储器属于 CPU,位于主存和 CPU 之间,存放当前正在执行程序的部分程序段或数据,以便向 CPU 快速提供即刻要执行的指令或马上要处理的数据。目前 Cache 一般采用双极型半导体存储器,也有采用 CMOS 半导体存储器的。速度可与 CPU 匹配,存取时间在几个 ns 到十几个 ns 之间,存储容量一般在几 KB 到几百 KB 之间。

(3) 主存储器。主存储器存放当前处于活动状态的程序和有关数据,它包括操作系统的常驻部分和当前正在运行的程序和要处理的数据。它与 CPU 的关系最密切,CPU 通过指令可直接访问它。存储容量比 Cache 大得多,一般在几百 KB 到几百 MB 之间。如我国自行设计自行制造的 YH - 1 巨型机主存为 32 MB,YH - 2 巨型机主存为 256 MB。目前主存储器大多为 MOS 半导体存储器,存取速度比 CPU 慢一个数量级,一般为几十到几百 ns。因主存属于主机范畴,所以常称它为内存(Internal Memory)。对于主存,从物理结构上看,若干个存储位元组成一个存储单元(Memory Location)。一个存储单元可存放一个机器字(Word)或一个字节(Byte)。存放一个机器字的存储单元,称字存储单元。存放一个字节的存储单元,称字节存储单元。

(4) 辅助存储器。前面已经谈到,辅存即外存,属于外围设备的范畴。辅存与主存的最大区别在于它不能由 CPU 的指令直接访问,必须通过专门的程序或专门的通道把所需的信息与主存进行成批交换,调入主存后才能使用。外存储器用来存放需要联机保存但暂不使用的程序和数据。计算机系统能提供丰富的软件资源,如操作系统、各种语言的翻译或解释程序、编辑程序、链接装配程序、调试程序以及各种软件工具等。某个用户程序可能只需使用其中的一部分,如某种语言的翻译程序,或者在工作的某一阶段只用其中的一部分,如编程时仅需编辑程序,编译时只用解释程序,链接装配时只需链接装配程序,调试时则只应用调试程序等。通常,都是将所有软件以文件的形式存入辅存,需要时再调入主存。对于大的用户程序,需要运行的程序量很大,程序涉及的数据可能也很多,而 CPU 在某一段时间里,运行的程序和要处理的数据只是其中的一部分,暂不运行的程序和数据也先存入辅存。同主存相比,辅存容量相当大,通常在几十 MB(2^{20} B)到几百 GB(2^{30} B)之间,有的甚至达到几个 TB(2^{40} B)。辅存存取速度比主存至少慢 5 个数量级,通常为若干个 ms。

(5) 其他功能的存储器。微程序控制的计算机和微程序控制的输入输出设备中,都必须设置存储微程序代码的控制存储器 CM。通常要求 CM 容量尽量小,速度尽可能快。为了加快处理速度,高性能计算机 CPU 中还设置各种表格存储器,通过查表实现各种计算功能,如三角函数表,对数表、倒数表等。有了倒数表,可以不设计除法部件,大大简化了 CPU 设计。在显示和印刷输出设备中,通常都设有字库和数据缓冲存储器。如图形图像显示缓冲存储器既要求容量大,又要求速度快;西文字库容量较小,一般不超过 2 KB;汉字简写体字库容量可达几百 KB,繁体字字库达 4 MB。

2. 按存储介质分类

凡仅有两种稳定的物理状态,能方便地检测出属于哪种稳定状态,两种稳定状态又容易相互转换的物质或元器件,都可以用来记忆二进制代码“1”和“0”,称这样的物质或元器件为存储介质或记录介质。存储介质不同,存储信息的机理也不同,因此,又把按存储介质分类称为按存储机理分类。存储介质种类繁多,下面仅就目前计算机中广泛使用的和我们认为很有发展前途的几种作简单介绍。

(1) 半导体存储器 (Semiconductor Memory, SCM)

早期的 SCM 采用典型的晶体管触发器作为存储位元,加上选址、读写等电路构成存储器。现代的 SCM 采用超大规模集成电路工艺制成存储芯片,每个芯片中包含相当数量的存储位元,再由若干芯片构成存储器。

从集成电路类型的角度,SCM 分为晶体管双极 (Bipolar) 型和场效应管 (Metal Oxide Semiconductor, MOS) 型。双极型又分射极耦合逻辑即 (Emitter Coupled Logic, ECL), 晶体管晶体管逻辑即 TTL 和集成注入逻辑即 I²L (Integrated Injection Logic) 三种类型。从制造工艺看,MOS 型有 PMOS (P channel MOS)、NMOS (N channel MOS) 和 CMOS (Complementary MOS) 三类,目前广泛采用的是后两类 MOS。从加电后能否长时间保持所存信息的角度,

MOS型的SCM又有静态存储器和动态存储器之分,前者存储位元电路是双稳态触发器,后者存储位元电路的关键部件是电容;前者只要电源电压正常供电,信息则长期保存,后者即使电源电压正常供电,信息也只能保持几个或十几个ms,因此必须在规定时间内刷新。若电源电压不正常或断电,两者信息都丢失。双极型存储器都是静态存储器,无静态、动态之分。

双极型存储器速度快,通常比MOS存储器至少要高一个数量级,但功耗大,集成度低,适用于快速小容量存储器,如高速暂存存储器和Cache。NMOS静态存储器制造工艺简单、集成度高、单片容量大,主要用作快速主存。CMOS静态存储器功耗最小,速度比NMOS快,集成度比双极型高得多,可靠性高,虽然制造工艺复杂,目前仍用得很广泛,主要用作快速主存和Cache。动态MOS存储器内部结构最简单,在各类SCM中集成度最高,功耗很小,速度虽比静态MOS和双极型存储器低一些,但其仍被广泛用作主存。

(2) 磁表面存储器(Magnetic Surface Memory, MSM)

磁表面存储器是用陶瓷、非磁性金属或塑料作载磁体,又称基体。在其表面涂敷、电镀、沉积或溅射一层很薄的矩磁材料,用磁层磁化后具有的、两种不同的剩磁状态记录信息“1”和“0”。磁层称为磁记录介质。依载磁体形状的不同,可分别称磁卡存储器、磁带存储器、磁鼓存储器和磁盘存储器。计算机中目前广泛使用的MSM是磁盘和磁带存储器。

MSM通过载磁体作高速旋转或平移,借助于软磁性材料制作的磁头实现读写。由于是机械运动方式,所以存取速度远低于SCM,为ms级。MSM的存储位元是磁层上非常小的磁化区域,可以小至 $20\text{ }\mu\text{m}^2$,所以存储容量可以很大,与SCM相比,每位价格低得多,因此广泛用作辅存。

(3) 光盘存储器(Optical Disk Memory, ODM)

和MSM类似,ODM也是将用于记录的薄层涂敷在基体上构成记录介质。不同的是基体的圆形薄片由热传导率很小,耐热性很强的有机玻璃制成。在记录薄层的表面再涂敷或沉积保护薄层,以保护记录面。记录薄层有非磁性材料和磁性材料两种,前者构成光盘介质,后者构成磁光盘介质。

ODM是目前辅存中记录密度最高的存储器,存储位元区域可小于 $1\text{ }\mu\text{m}^2$,存储容量很大且盘片易于更换。缺点是存取速度比硬盘低一个数量级。现已生产出与硬盘速度相近的ODM,不久会成为重要的辅存。

(4) 铁电存储器(Ferroelectric Memory, FeM)

20世纪80年代末,由于铁电薄膜技术的突破,铁电存储器发展很快,它是用两种极化状态表示信息“1”和“0”,因其所需电压低,存取速度快,加之可高密度集成,有望成为动态存储器的替代品。

3. 按存储方式分类

(1) 随机存取存储器(Random Access Memory, RAM)

这种存储器是以存储单元为单位组织信息和提供访问的,CPU通过指令可随机写入或

读出信息。随机是指对存储器的任何存储单元都可随时访问且访问所需时间都是相同的,与存储单元所处的物理位置无关。原因是这种存储器对每个存储单元都有惟一的、由电子线路译码器构成的选址机构。

这类存储器的特点是速度快、访问时间是 ns 级,用作 cache 和主存。目前广泛使用的是 SCM。

(2) 按内容寻址存储器 (Content Addressed Memory, CAM)

亦称相联存储器 (Associative Memory, AM), 也是一种随机存取存储器。除按地址可随机读写外,还有比较功能。可按信息内容寻址,然后再按地址访问,且主要是这种工作方式。这类存储器首先要求的是速度,常用于逻辑比较和快速查找等场合,智能计算机中的推理存储器和 Cache、主存层次的地址变换时用的快表都使用这种存储器。

(3) 只读存储器 (Read Only Memory, ROM)

这种存储器除正常工作时只能随机读出信息,不能随机写入信息外,其他特征均同 RAM。ROM 中的信息是在正常工作前事先写入的,信息一旦写入后,便可长期保存。目前广泛使用的是半导体大规模集成电路的 ROM。用户要求不同,写入方式也存在很大差异。

ROM 在计算机中得到广泛应用,主要用来存放管理、监控程序,成熟的用户程序,固定表格和常数,字库和微程序等。

(4) 顺序存取存储器 (Serial Access Storage, SAS)

这种存储器中的信息按文件组织,一个文件可包含若干个数据块,一个数据块又包含若干个字节,它们顺序地记录在存储介质上,存取时以块为单位,只能顺序查找块号,找到后即成块顺序读写,所以存取时间与信息所处的物理位置关系极大。

这类存储器速度慢、容量大、成本低,常作为后援(脱机)辅存。磁带存储器属此类,其存储时间为秒级。CCD(Charge Couple Device)和磁泡也属于顺序存储器,其速度比磁带虽快得多,但近几年发展十分缓慢,几乎处于停滞状态。VCD 光盘也属于此类,且应用广泛。

(5) 直接存取存储器 (Direct Access Storage/Direct Access Memory, DAS/DAM)

这种存储器信息的组织同 SAS,存取信息也是以块为单位。它是介于随机存取和顺序存取之间的一种存储器。对信息的存取分两步进行,首先随机指向存储器的一个区域,如磁道、光道,然后对这一部分区域进行顺序存取。磁盘、可擦写型光盘属于此类存储器,其存取时间为 ms 级。

4. 按信息的可保存性分类

(1) 挥发性存储器 (Volatile Memory) 和非挥发性存储器 (Non - volatile Memory)

挥发性存储器亦称易失性存储器,这种存储器的特点是断电后信息即丢失。非挥发性存储器亦称非易失性或永久性存储器,这种存储器特点是断电后信息不丢失。半导体随机存取存储器为挥发性存储器,半导体只读存储器、磁表面存储器和光盘存储器都是非挥发性存储器。

(2) 破坏性读出存储器 (Destructive Read Memory) 和非破坏性读出存储器 (Nondestructive Read Memory)

当读出某单元信息时,如果破坏了原存信息,则称这样的存储器为破坏性读出存储器;若读出时,不破坏原存信息,这样的存储器称为非破坏性读出存储器。破坏性读出存储器信息读出后必须马上有一个重写操作,亦称之为再生,以恢复被破坏的信息。ROM 和 MSM 存储器、半导体静态存储器都是非破坏性读出存储器。半导体动态存储器和铁电存储器则属于破坏性读出存储器。

5.1.2 内存的主要技术指标

如前所述,Cache 和主存统称内存,用于存放当前处于活动状态的程序和数据。换句话说,内存是 CPU 可直接随机访问的存储器,要求其容量足够大、速度尽量与 CPU 匹配。这不仅是设计人员追求的目标,也是评价内存性能的主要技术指标。

1. 存储容量 (Memory Capacity)

内存的存储容量指的是内存所能容纳的二进制位(bit)个数的总和,即构成内存的存储位元的总和。存储容量等于存储单元个数与每个存储单元包含的存储位元个数之积。对于字节编址计算机,以字节(Byte,简写为 B)数表示存储容量;对于字编址计算机,用二进制位(bit,简写为 b)数表示存储容量。现代计算机存储容量很大,用 B 或 b 作单位很小,加之字长均为字节的整数倍,故常用 KB、MB、GB、TB 表示存储容量。

$$1 \text{ KB} = 2^{10} \text{ B} = 1024 \text{ B}; 1 \text{ MB} = 2^{20} \text{ B} = 1024 \text{ KB}; 1 \text{ GB} = 2^{30} \text{ B} = 1024 \text{ MB}; 1 \text{ TB} = 2^{40} \text{ B} = 1024 \text{ GB}$$

2. 存取时间 (Memory Access Time, MAT)

存取时间亦称访问时间,指的是从启动一次存储器操作到完成该操作所用时间。如从发出读命令到将数据送入数据缓冲寄存器所用时间,或从发出写命令到将数据缓冲寄存器内容写入相应存储单元所用时间,用 T_A 表示。 T_A 是反映存储器速度的指标,其值取决于存储介质的物理特性及其使用的读出机构类型。 T_A 决定了 CPU 进行一次读或写操作必须等待的时间。目前内存的存取时间为 ns 级。

3. 存储周期 (Memory Cycle Time, MCT)

存储周期亦称存取周期、访问周期、读写周期。指的是连续两次启动同一存储器进行存取操作所需的最长时间间隔。用 T_M 表示。因为对任一种内存,当进行一次访问后,存储介质和有关控制线路都需要恢复时间,若是破坏性读出,还需重写时间,因此通常 $T_M > T_A$ 。 T_A 通常主要用来表示 CPU 发出读命令后要等待多长时间才能获得数据,这对 CPU 的设计有非常重要的意义。如果考虑计算机与访存有关的工作周期,则会涉及到 T_M 。

4. 可靠性 (Reliability)

计算机的一切工作都是通过运行程序实现的,而正在运行的程序和要加工处理的数据

都存放在内存中,因此,内存的可靠性处于非常重要的地位。通常用平均无故障时间(Mean Time Between Failures, MTBF)来衡量主存的可靠性,MTBF 表示两次故障之间的平均时间间隔。显然,MTBF 越大,可靠性越高。为了加大 MTBF,内存采用容错技术。所谓容错,就是在存储器出现故障时,能够纠正错误,使之正常工作,或者至少能报告错误,以便人工排除。通常通过增加冗余位实现,如 YH - 2 巨型机,CPU 字长 64 位,而存储器字长为 72 位,多用 8 位可纠正 1 位错误、检测出 2 位错误,这样大大提高了内存的可靠性。

5. 功耗与集成度 (Power Loss and Integration Level)

功耗反映了存储器件耗电多少,集成度标识单个存储芯片的存储容量。一般希望功耗小、集成度高,但两者是矛盾的,因此除设计和制作存储芯片时要同时考虑两者之外,用芯片构成内存时也应当考虑它们。对于 SCM,有维持功耗和工作功耗之分,通常要求维持功耗尽量小。

对于高密度组装的高速内存,则应采用风冷、液冷等强化散热措施,否则内存将不会稳定工作,甚至有烧毁的危险。

6. 性能价格比 (Cost Performance)

性能价格比是一个综合性指标,性能主要包括存储容量、存储周期、存取时间和可靠性等。价格包括存储芯片和外围电路的成本。通常要求性能价格比要高。

7. 存取宽度 (Access Width)

存取宽度亦称存储总线宽度,即 CPU 或 I/O 一次访存可存取的数据位数或字节数。存取宽度由编址方式决定。字节编址存取宽度为 8 位,字编址存取宽度为机器的字长,它一般是字节的整数倍。如 YH - 1 巨型计算机存取宽度为 64 位。低档微机存取宽度为 8 位、16 位。高档微机存取宽度为 32 位、64 位。

5.2 内存储器的工作原理

目前,内存储器绝大多数是由半导体元器件构成的。因此又将内存储器称为半导体存储器(SCM)。

SCM 最小逻辑单位是存储位元,它存储一位二进制信息。稍大些的逻辑单位是存储单元,它由若干存储位元的构成,存储一个或多个字节。再大些的逻辑单位是存储芯片(Memory Chip),它除包含一定数量(一般为 2 的整数次幂)的存储位元或存储单元外,还包括对存储单元操作的外围线路,如译码器、驱动器、读出放大器和写入电路等。若干存储芯片构成 SCM。对于大容量 SCM,为提高速度、减小功耗和方便控制,通常又分成多个存储体(Memory Bank),每个存储体由一定数量的存储芯片构成。

本节立足于半导体存储器,介绍内存储器的基本组成及读写操作过程、随机存取存储器

RAM、及其存储芯片和只读存储器 ROM。

5.2.1 内存储器的基本组成及工作过程

内存储器基本组成如图 5.2 所示。

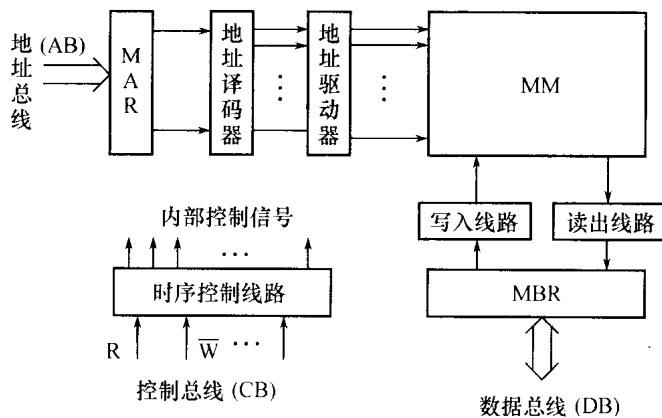


图 5.2 内存储器基本组成

下面简介各部分的组成和功能。

1. 内存储器的基本组成

(1) 存储矩阵 (Memory Matrix, MM)

存储矩阵亦称存储体、存储阵列。它是信息的驻在地，是存储单元的集合，是由大量存储位元构成的。因存储位元、存取方式和存储器功能不同，组织形式也存在很大差别。

(2) 选址系统 (Addressing System)

由存储地址寄存器 MAR、地址译码器和地址驱动器三部分组成。I/O 或 CPU 访问内存时，先将访问内存地址送 MAR，经地址译码器找到被访问的存储单元，最后由地址驱动器驱动该存储单元以便实现读或写。

(3) 读写系统 (Read and Write System)

包括存储缓冲寄存器 MBR，读出和写入线路。读出根据访存的读命令，借助于读出线路将由选址确定的存储单元内容读出送 MBR，供 I/O 或 CPU 使用。若是破坏性读出，还应启动写入线路将信息重写回原单元。写入是先将要写入的数据送 MBR，根据写命令，借助于写入线路，将 MBR 内容写入由选址确定的存储单元。

(4) 时序控制线路 (Sequential Control Circuit)

包括控制触发器，各种门电路和延迟电路等。接收 I/O 或 CPU 的启动、读写和清除等命令，产生一系列时序信号控制内存储器完成读写等操作。

2. 存储矩阵与选址

内存的四个基本组成中,存储矩阵是核心,其他三部分是存储矩阵的外围线路。其中选址是关键部件,线路复杂。选址的方法不同,直接影响存储矩阵的组织与结构。下面以 2^k 个存储单元,每个单元存储n位(bit)信息,即 2^k 字 \times n位为例说明两类存储矩阵与选址方法。

(1) 线选法二维存储矩阵

二维存储矩阵采用一维地址译码选址技术即线选(Linear Select)技术。实际上就是把K位地址码经译码后,得到 2^k 根驱动线,每线对应一个存储单元,其只与该存储单元的n个存储位元相连接。只要选中某条驱动线,则选中且仅选中该存储单元的所有位元,因此称 2^k 条驱动线为字线,它是二维存储矩阵的一维。对存储单元n位信息进行读出或写入,还需通过n根数据线,称这n根数据线为位(bit)线,它是二维存储矩阵的另一维。

二维存储矩阵字线和位线的交叉点上各有一个存储位元。假定n和K都是4,线选法二维存储矩阵如图5.3所示,图中小矩形表示存储位元C,位元C的下标前一位或前二位数表示该位元所在的存储单元的地址,最后一位数字表示是第几位。图中16条字线 $W_0 \sim W_{15}$ 和4根位线 $D_0 \sim D_3$ 相交处有64个存储位元,每条字线与4个存储位元连接,表示一个存储单元存储4位信息,即字长为4位。每根位线与16个存储位元连接,表示存储矩阵由16个存储单元组成。线选法二维存储矩阵组成的基本特征是,每根位线与同一位的所有存储位元连接,每根字线仅与同一存储单元的所有存储位元连接,选中一条驱动线,即能存取一个字。

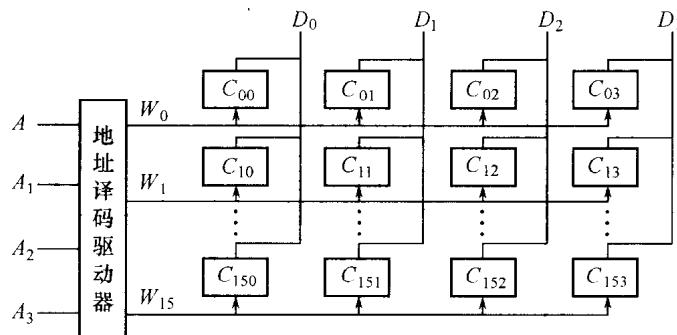


图 5.3 线选法 16 字 \times 4 位二维存储矩阵示意

线选法二维存储矩阵每条驱动线的负载是字长个存储位元,其负载轻,读写速度较快。但当地址码位数K很大,即存储单元数很多时,采用一维地址译码产生 2^k 条驱动线,一般要经过多级译码才能完成,译码线路将会很庞大。所以一维地址译码只在高速、小容量存储器中使用。

(2) 重合法三维存储矩阵

三维存储矩阵采用二维地址译码选址、一维读写。所谓二维地址译码是把K位地址码分成接近相等的两组,分别译码,产生一组行驱动线X(X_0, X_1, \dots, X_i)和一组列驱动线Y

(Y_0, Y_1, \dots, Y_j) , 然后 X 和 Y 驱动线在存储矩阵内部一一相与, 它们的每一交点处放一个存储位元构成存储矩阵的一个位面, 用一根位线连接该面所有位元。字长为 n 则有 n 个位面, 各位面的位线是相互独立的, 而 X 、 Y 驱动线按序一一连接在一起, 这样 n 个位面就构成三维存储矩阵。

n, K 都为 4 的重合法三维存储矩阵如图 5.4 所示, 图中 4 根位线各自连接 16 个位元, 每根驱动线也连接 16 个位元, 但分布在四个位面上, 每个位面 4 个位元。任选一地址都会选中两条驱动线, 两线交叉处的 4 个位元同时被两条驱动线选中, 称它们为全选位元, 两条驱动线上的其他位元称作半选位元。实际上全选位元才组成该地址所对应的存储单元, 也只能对全选位元进行读写, 故称此选址方法为重合法。

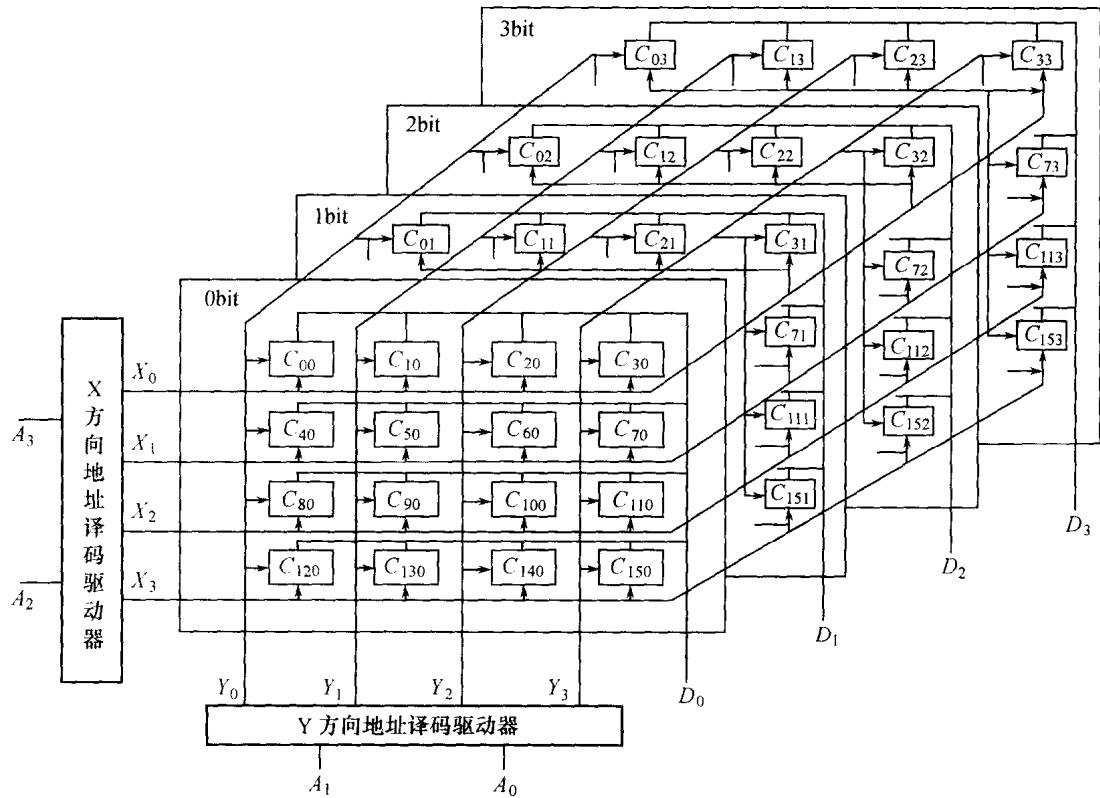


图 5.4 重合法 16×4 位三维存储矩阵

重合法实质上是把部分译码功能移到了存储矩阵内部, 从而使外部的译码线路大大简化, 以 K 是偶数为例, 线选法译码用 K 个输入端的 2^K 个“与”门进行一级译码, 产生 2^K 根驱动线, 而重合法只用 $K/2$ 个输入端的 $2 \times \sqrt{2^K}$ 个“与”门的译码器产生 $2 \times \sqrt{2^K}$ 根驱动线, K 值越大, 两种方法的差距越大。但是, 重合法的优点是以增加存储矩阵的复杂性为代价, 它要求每个存储位元具有“与”功能。当然, 只有利用存储位元本身物理性质上的“与”关系来

实现这种“与”功能才是最佳的方案。

在实际中采用哪种存储矩阵和寻址译码方式要取决于电路结构、制作工艺及对存储器速度的要求。

3. 读操作过程

(1) 送地址:控制器(CU)通过地址总线(AB)将指令或操作数的地址送 MAR;

(2) 发读命令:CU 通过控制总线(CB)将“读存储器”信号 R 送时序控制线路。

(3) 从存储器读出数据:时序控制线路依信号 R 产生一系列存储器的内部控制信号。在这些信号的控制下,MAR 中的地址经地址译码器选中并驱动存储矩阵中的某一个存储单元;读写系统工作,读出该单元中所有存储位元的信息,送 MBR;再经数据总线(DB)将读出的信息送控制器的 IR(读出的是指令)或送运算器(读出的是操作数)。

4. 写操作过程

(1) 送地址:CU 通过 AB 将要写信息的地址送 MAR;

(2) 送数据:将要写入存储矩阵中的信息经 DB 送 MBR;

(3) 发写命令:CU 通过 CB 将“写存储器”信号 \bar{W} 送时序控制线路。

(4) 将数据写入存储器:时序控制线路依信号 \bar{W} 产生一系列存储器的内部控制信号。在这些信号的控制下,MAR 中的地址经地址译码器选中并驱动存储矩阵中的某一存储单元,同时读写系统工作,将 MBR 中的数据写入被选中存储单元的各存储位元中。

5.2.2 随机存取存储器 RAM

存储矩阵由若干个存储单元构成。每个存储单元由多个存储位元构成。一个存储单元位元的个数是字节的整数倍,通常称其为计算机的字长。一个存储单元对应惟一的物理地址。下面简单介绍三种 RAM 存储位元电路。

1. 双极型存储位元电路

双极型存储位元电路式样繁多,至少有十几种,它的基本结构是双稳态触发器。我们仅介绍二极管开关集电极阻抗存储位元电路,其是利用二极管的变阻抗(非线性)特性来改变触发器集电极负载阻抗,使存储位元处于维持状态时是小电流,工作状态时是大电流。掌握了这种存储位元电路,遇到其他双极型存储位元电路即可触类旁通了。

(1) 位元电路的组成

这种存储位元电路如图 5.5 所示。它由两只晶体管 T_1 、 T_2 和负载电阻 R_{c1} 、 R_{c2} ,二极管 D_1 、 D_2 ,电阻 r_1 、 r_2 组成。其中 T_1 的一个射极和 T_2 的一个射极连接在一起,接到维持恒流源,另外两个射极分别接到位线 D 和 \bar{D} ,电阻 R_{c1} 、 r_1 、 R_{c2} 和 r_2

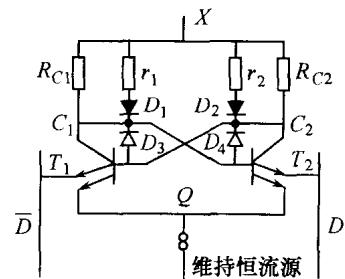


图 5.5 二极管开关集电极阻抗存储位元电路

接于驱动线 X 。此外,还有两个抗饱和肖特基二极管, D_3 和 D_4 。

(2) 工作原理

这种位元电路最适合匹配 ECL 外围电路,下面就以 ECL 电路作其外围电路说明存储位元电路的工作原理。假定 T_1 通导, T_2 截止为“1”状态; T_1 截止, T_2 通导为“0”状态。为便于说明,分为三种状态讨论。

① 保持:即不读不写维持原存信的状态。存储位元处于保持状态时,驱动线 X 是低电位(-1.7 V),两边位线 \bar{D} 和 D 都处于中间电位(-1.9 V)。这时,触发器必然是一个管通导、一个管截止且总电流通过 Q 点(V_Q 为 -2.7 V)流入维持恒流源,此恒流源吸收约 $100\text{ }\mu\text{A}$ 电流,使触发器能维持原有的稳定状态,保持已存的信息。值得注意的是存储位元处于保持状态时,集电极负载电阻所并联的二极管 D_1 、 D_2 都不导通,由于负载电阻很大,所以维持电流很小,维持功耗亦很小。

② 写入:写入就是当存储位元被选时,将要写入的信息(“1”或“0”)写入存储位元。假如是写“1”,首先驱动线 X 来正脉冲(-0.8 V),位线 \bar{D} 加 -2.4 V ,位线 D 加 -1.5 V 。显然,在驱动线 X 电位变高的瞬间,并联二极管 D_1 、 D_2 将会导通,使负载电阻变小、电流增大。但由于触发器的下端是接维持恒流源,限制了电流增加, Q 点电位将升高。当高过两边位线上的电位时,由于位线上的电位不平衡,位线 \bar{D} 电位低,位线 D 电位高,迫使 T_1 通导, T_2 截止,触发器变为“1”状态,即写入了“1”,当驱动脉冲去掉后,驱动线 X 上的电位变为 -1.7 V ,存储位元保持已写入的“1”状态。

写“0”和写“1”的过程相同,只是位线 \bar{D} 和 D 所加的电压调换即可,不再赘述。

③ 读出:读出时两边位线 \bar{D} 和 D 都是中值电位(约 -1.9 V),驱动线 X 来正脉冲(-0.8 V),在驱动线 X 上的电位变高的同时,并联二极管 D_1 、 D_2 瞬间导通, Q 点电位升高(因为维持恒流源存在),使通导管有较大的电流流往位线。假如存储位元原存信息为“1”,则必定是 T_2 截止, T_1 通导,且有较大电流从位线 \bar{D} 流出,位线 D 上无电流,代表读出“1”;如果存储位元原存信息为“0”,则 T_1 截止, T_2 通导,位线 D 上有电流,而位线 \bar{D} 上无电流,代表读出“0”。当驱动线 X 的电位再度变低时,存储位元继续保持其原存信息状态。所以它是不破坏读出。

该位元电路因采用二极管开关集电极阻抗,导致工作电流大,维持电流小。加之采取了抗饱和措施,使得该位元电路速度快,维持功耗小。由于其为双极型器件,功耗比后面要介绍的 MOS 型器件要大得多,故集成度低。

2. MOS 静态互补存储位元电路

双极型存储位元电路采用的是触发器形式,因为触发器具有两种不同的稳定状态,用它所处不同的稳定状态来代表一位二进制的信息。当没有外界信号作用时,触发器可以长久保持其所处的某种稳定状态,所以通常把这种存储位元电路叫做静态存储位元电路。不言而喻,以 MOS 晶体管触发器构成的存储位元电路,就是 MOS 静态存储位元电路。

(1) 位元电路的组成

MOS 静态互补存储位元电路如图 5.6 所示。核心是两个由 MOS 管构成的互补反相器直接耦合而成。在电路中是以 P 沟增强型 T_5 和 T_6 作负载, 而以 N 沟增强型 T_1 和 T_2 作开关管构成两个反相器。并将 T_1 、 T_5 的栅极与 T_2 、 T_6 的漏极相连, T_2 、 T_6 的栅极与 T_1 、 T_5 的漏极相连构成触发器。为了和外围电路传递信息, 增设两个 N 沟增强型门控管 T_3 和 T_4 , T_3 的漏极与 T_1 、 T_5 漏极相连接, T_4 漏极与 T_2 、 T_6 漏极相连接。两门控管栅极连在一起引出一根线, 叫作字 (word) 驱动线, 用 W 表示, 源极则分别接到位线, 亦称数据线, 用 \bar{D} 和 D 表示。

(2) 工作原理

假设 T_1 通导, T_2 截止为“1”状态; T_1 截止, T_2 通导为“0”状态。下面分三种状态讨论。

① 保持: 保持时字驱动线 W 处于低电位 (近于 0 V), 使门控管 T_3 和 T_4 都关闭, 切断了触发器和位线 \bar{D} 和 D 的电联系, 触发器自身处于保持状态。

② 写入: 写入时, W 来正脉冲 (近于 V_D), 打开 T_3 和 T_4 , 如果要写“1”, 则位线 \bar{D} 加低电位 (近于 0 V), 位线 D 加高电位 (近于 V_D)。此时, 不管触发器原来处于何种状态, 一定会使 T_1 导通 T_5 截止、 T_2 截止 T_6 导通, 使位元成为“1”状态。如果要写“0”, 则位线 \bar{D} 加高电位, 位线 D 加低电位, 迫使 T_1 截止、 T_5 导通、 T_2 导通、 T_6 截止, 使位元变为“0”状态。在 W 正脉冲消失后, T_3 、 T_4 被关闭, 位元保持刚写入的信息状态。

③ 读出: W 来正脉冲, 打开门控管 T_3 和 T_4 , 这时位元所存信息, 即 D_1 和 D_2 点电位, 就可通过 T_3 和 T_4 传送到位线上, 即读出了信息。若原存信息是“1”, 则因 T_1 通导, T_2 截止, 使 D_1 点为低电位, D_2 点为高电位。经 T_3 加到 D 线上是低电位, 经 T_4 加到 D 线上是高电位, 代表读出是“1”。如果原存信息是“0”, 读出时, 位线 \bar{D} 得到的是高电位, 位线 D 得到的是低电位, 代表读出是“0”。当 W 线上正脉冲消失, 门控管 T_3 和 T_4 关闭, 位元仍保持原来状态。

不难看出, 在读出过程中, 触发器的状态, 并未破坏, 原存信息仍然存在。故这种存储位元电路也是不破坏读出的, 读后不需要重写。

需要指出的是这种存储位元电路只是在状态转换过程中才会有较大的电流, 而平时反相器的两只晶体管总是一个导通, 一个截止。所以它的功耗比双极型存储位元小得多, 可靠性也高得多, 但制作工艺复杂, 成本较高, 速度也比双极型存储位元慢。

3. MOS 动态存储位元电路

动态存储位元电路和静态存储位元电路不同, 它是利用 MOS 晶体管极电容 (或 MOS 电容) 上充积的电荷来存储信息的。由于有漏电阻存在, 电容上的电荷不可能长久保存, 需要周期性地对电容进行充电, 以补充泄漏的电荷, 通常把这种补充电荷的过程叫刷新。所以动

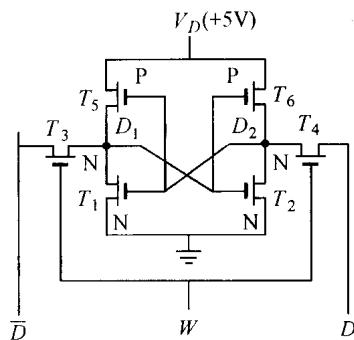


图 5.6 静态互补存储位元电路

态存储位元电路是由多时钟脉冲控制、在动态情况下工作的电路,因此得名为动态存储位元电路。

静态存储位元电路能够长久保持信息,不需要刷新,工作稳定可靠,为什么又提出动态存储位元电路呢?原因是静态存储位元电路存在着功耗大,集成度低的缺点。动态存储位元电路恰恰克服了静态存储位元电路的这一缺点。动态存储位元电路的出现,使MOS器件的优点更得以发挥。

(1) MOS 动态单管存储位元电路组成

图5.7(a)所示的单管动态存储位元电路,它是由一只晶体管和一个电容组成。其中晶体管T的栅极接字驱动线W,漏极接数据线D,源极接电容C,此电容为存储电容。它是特制的MOS电容,如图5.7(b)所示。在制作晶体管T的栅极时,在源极的左边也作一个较大的“栅极板”A,并引出电极接于电源 V_D ,由于“栅极板”A是接 V_D ,所以在板A下面所对应的硅表面会产生反型层,这个反型层就构成了电容C的另一个极板,并且与晶体管的源极连通,恰好是图5.7(a)的连接形式。可见,C是个二值电容,当板A接 V_D 时,就是一个大电容,当板A接地(或接电压较低不能产生反型层)时,C就不存在了。确切地说,C的电容值很微小。

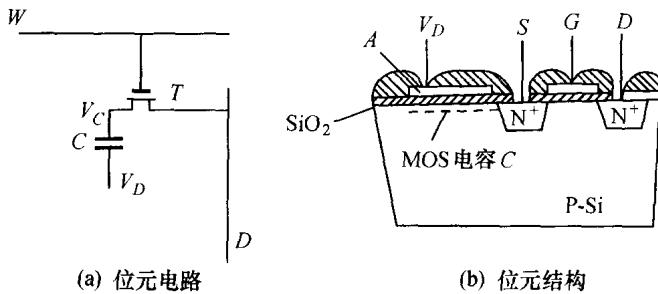


图 5.7 单管动态存储位元电路

(2) 工作原理

① 保持:单管动态存储位元电路,是以电容C上是否有电荷来表示存储信息的。C上有电荷表示存“1”;C上无电荷表示存“0”,平时字驱动线W是加低电位,T关闭,切断了电容C的通路,使其被充的电荷不能放掉,以保持存储的信息。当然,由于电容总要有一定的漏电,所以它不可能长久保存信息,因而需要刷新。

② 写入:写入时,先对数据线D充电到中值电平 V_M ,使其有确定的电位。 $V_M = (V_{cl} + V_{lo})/2$,其中 V_{cl} 为存储“1”时电容C上的电压, V_{lo} 为存储“0”时电容C上的电压。然后,把欲写入的信息以电位信号形式加到数据线D上。假定是写“1”,则数据线D加高电位,字驱动线W加正脉冲,将T打开,数据线高电位通过T对电容C充电,使C充满正电荷, V_c 变为高电位,即写入了“1”。如果是写“0”,则数据线加低电位(近于0V),当字驱动脉冲来时,T通导,电容C上的电荷几乎放光, V_c 变为低电位,即写入了“0”。在字驱动脉冲结束后,T关闭,刚写入的信息以电荷形式保存在电容C上。

③ 读出：首先对数据 D 充电到 V_M ，作好读取数据准备；然后加字驱动脉冲，使 T 通导，则电容 C 上所存的信息可通过 T 读到数据线 D 上；再由读出放大器输出。假如原存信息是“1”，即 C 上有电荷， V_c 为高电位，当字驱动脉冲到来时，打开 T ， V_c 通过 T 加到数据线 D 上，使数据线 D 上电位比 V_M 高，表示读出是“1”。如果原存信息是“0”，即 C 上几乎没有电荷； V_c 为低电位，当字驱动脉冲到来时，打开 T ，则传送到数据线 D 上的电位比 V_M 低表示读出是“0”。

单管动态存储位元电路是破坏性读出，信息被读出后必须马上恢复，否则信息会丢失。MOS 动态存储位元和上述两种存储位元比较，优点是功耗更小，集成度更高；缺点是控制线路复杂，读写速度慢，还需要刷新。

4. 随机存取存储器 RAM

双极型 RAM 存储位元电路与双极型外围电路配合构成的随机存取存储器称之为双极型随机存取存储器，简称为 BiRAM (Bipolar RAM)。MOS 型静态 RAM 存储位元电路与 MOS 型或双极型外围电路配合构成的随机存取存储器称之为静态随机存取存储器，简称为 SRAM (Static RAM)。MOS 型动态 RAM 存储位元电路与 MOS 型外围电路配合构成的随机存取存储器称之为动态随机存取存储器，简称为 DRAM (Dynamic RAM)。

BiRAM 因其读写速度快而多被用作高速缓冲存储器 Cache。DRAM 因其功耗极小且集成度可以做得很高而多用来作为主存。SRAM 因其速度和集成度处于 BiRAM 和 DRAM 之间，则在有的计算机中作为 Cache，在有的计算机中作为主存。如在高档微机中，片内 Cache 为 BiRAM，片外（二级）Cache 为 SRAM，主存则采用 DRAM。又如，我国自行研制的 YH-2 巨型计算机，为了速度的需要，主存也采用了 SRAM。

5.2.3 RAM 存储芯片

存储芯片是将一定数量的存储位元组成的存储矩阵连同与之对应的选址、读写等系统集成制造在一个硅片上，引出与外界联系所必需的端点，再用塑料或陶瓷予以封装，即制成了存储芯片。通常称引出的端点为引脚。使用不同种类的存储位元，构造出不同种类的存储芯片。若使用双极型 RAM 存储位元，则构造出 BiRAM 存储芯片；使用双极型 ROM 存储位元，则构造出双极 ROM 存储芯片。为扩展构造存储器的方便，存储芯片的容量，通常用多少个字乘以多少位表示。如 $32\text{ K} \times 8$ 位的存储芯片，说明该存储芯片内含 32 K 个字，每个字存储 8 位二进制信息；亦可理解为该芯片有 32 K 个地址，每个地址中存储了 8 位二进制信息。当然亦可认为该存储芯片容量为 256 Kb 。

从存储位元和外围电路的类型考虑，RAM 存储芯片有三类。一类是双极型芯片，存储位元和外围电路都是双极型的。另一类是 MOS 型芯片，存储位元和外围电路都是 MOS 型的。还有一类被称之为 BiCMOS (Bipolar CMOS) 芯片，存储位元是 CMOS 型，外围电路是双极型的。BiCMOS 是近几年的新产品，其容量可达每片 1 Mb ，速度接近双极型。本节主要介绍双极型 RAM 存储芯片和 MOS 型 RAM 存储芯片。

1. 存储芯片的基本输入输出引脚

(1) 地址引脚

地址引脚是外部逻辑访问存储芯片选择存储单元用的引脚。通常地址引脚个数与芯片容量的关系为 $n = \log_2 m$, m 为存储芯片的字数, n 为地址引脚数, 通常记作 $A_{n-1}A_{n-2}\cdots A_1A_0$, 且 A_{n-1} 一般与高位地址码对应。但也有例外, 如 DRAM 存储芯片, 就采用地址引脚复用技术, 以减少芯片引脚个数。

(2) 数据引脚

① 数据输入端引脚

数据输入端引脚是写入数据的入口, 只在写入操作时才使用。其符号表示可以是 D_i 、 I_i 或 DI_i , 下标 i 表示位号。在 m 字 $\times 1$ 位的芯片中, 数据输入端引脚一般记作 D_{in} , 也有记作 I 或 DI 的。

② 数据输出端引脚

数据输出端引脚是读出数据的出口, 只在读出操作时才使用。其符号表示可以为 Q_i 、 O_i 、 QO_i , 与数据输入端引脚类似。在 m 字 $\times 1$ 位的芯片中一般记作 D_{out} , 也有标作 O 或 QO 的。

③ 双向数据端引脚

由于写入和读出在操作时间上是互斥的, 为了减少引脚个数, 大多数芯片同一位的写入和读出数据共享一个引脚。双向数据端引脚记作 DQ_i 、 IQ_i 或 IQ_i , 下标 i 是位的序号。为实现共享引脚, 芯片需要增加输出使能控制引脚。

(3) 控制引脚

① 片选引脚

大容量存储器一般由若干存储芯片构成, 存储器地址码位数比芯片地址引脚个数多。芯片的地址引脚只是对片内单元进行选择, 剩下的地址码位数则用于选择存储芯片。片选引脚 CS 或 S 则是存储芯片是否被选的控制信号, 其一般受控于存储器高位地址码的译码信号。通常 CS 为低电平表示有效, 即说明该芯片被选中, CS 在芯片内部对数据通路或者对地址通路进行控制以实现选片功能。

② 写使能端引脚

写使能端引脚亦称读/写控制引脚, 简称读/写端。它接收外部逻辑访问存储芯片的读/写命令信号。写使能端引脚标示为 \overline{WE} 、 \overline{W} 。写入时, \overline{WE} 为低电平(记作 $\overline{WE} = 0$), 打开数据输入通路。读出时 \overline{WE} 为高电平(记作 $\overline{WE} = 1$), 打开数据输出通路。

③ 片使能端引脚

片使能端引脚标作 CE 或 E , CE 是存储芯片是否进入有效操作状态的控制信号。当 CE 为低电平时, 供电电源进入工作状态, 芯片能够进行正常的读出或写入操作; 当 CE 为高电平时, 供电电源进入维持状态, 在维持状态, 存储的信息可以保持, 但不能进行读写操作。 CE 在存储芯片内部与自动下电(降低电压)电路连接, 并控制数据输入输出通路、地址通路实现状态

的自动转换。 \overline{CS} 和 \overline{CE} 在逻辑功能上有类似之处,所以, \overline{CS} 和 \overline{CE} 的命名有混淆现象。对于具体的存储芯片,一般有 \overline{CS} 则不再有 \overline{CE} ,两者功能可以用其中之一代替,通常有自动下电功能的芯片一般采用 \overline{CE} 引脚,无自动下电功能的采用 \overline{CS} 引脚。有的芯片可能有多个 \overline{CS} 或多个 \overline{CE} 引脚,这些引脚的关系究竟是“与”还是“或”,请阅读相应芯片的说明。

(4) 输出使能端引脚

输出使能端引脚标记为 \overline{OE} 和 \overline{G} ,其用于控制双向数据端引脚,以决定数据输入输出通路如何开通。当 \overline{OE} 为低电平时只允许数据输出,不允许数据输入;当 \overline{OE} 为高电平时,不允许数据输出,只允许数据输入。

\overline{OE} 通常与 \overline{CS} (或 \overline{CE})、 \overline{WE} 配合使用,完成读写操作。

(5) 行、列地址选通信号引脚

对 DRAM 存储芯片,还有行、列地址选通信号引脚,标记为 \overline{RAS} 、 \overline{CAS} ,作用是分别将行、列地址打入芯片。应当指出,当 DRAM 芯片无 \overline{CS} 引脚时,DRAM 芯片的 \overline{RAS} 还可以兼作 \overline{CS} 使用。

对于其他类型的存储芯片还有各自的专用信号引脚,如可编程只读存储芯片设置编程引脚 \overline{PGM} 或 \overline{P} ,具有可自动刷新的存储芯片设置刷新控制引脚 \overline{F} 。对于专用信号引脚,当涉及相应芯片时再进行介绍。

(4) 其他引脚

电源引脚 V_{CC} 、 V_{SS} 、 V_{PP} 和 V_{DD} ,接地引脚 GND ,暂时不用引脚 NC 等。

2. BiRAM 存储芯片

BiRAM 存储芯片目前最大集成度为每片 64 Kb,一般为每片 4 Kb 或 16 Kb;存取时间最快为 3 ns,一般在 10 ns 左右,最慢也不大于 30 ns;工作功耗,即芯片被选中时一般为 0.2 mW/位,最小为 0.03 mW/位,最大为 0.9 mW/位。整个芯片工作功耗超过 1 W 时,则应加散热片或采取其他散热措施。维持功耗即芯片未被选时功耗,通常为工作功耗的 1%。

表 5.1 给出几种常用 ECL RAM 存储芯片。其中封装形式栏中的 DIP,指的是双列直插式封装。对于 ECL RAM,一般为陶瓷双列直插式封装,原因是其功耗大,温度很高,而陶瓷耐热性比塑料强。图 5.8 给出 F100474 四方扁平封装(QFP) BiRAM 存储芯片引脚示意图。该芯片 MAT 为 10 ns,工作功耗为 1 W, V_{CC} 为 -5 V。

表 5.1 ECL RAM 存储芯片

型号	容量(字 × 位)	MAT(ns)	工作功耗(mW/片)	封装形式
MCM10146	1 K × 1	29	700	16 脚 DIP
10415	1 K × 1	10	1000	16 脚 DIP
F10470	4 K × 1	30	1024	16 脚 DIP
μPB10047E	1 K × 4	3	1125	24 脚 DIP
μPB100484	4 K × 4	10	1170	28 脚 DIP
SY100480	16 K × 1	6	990	24 脚 DIP
SY100494	16 K × 4	5	1778	28 脚 DIP

应当指出,BiRAM 存储芯片地址引脚和数据引脚都不允许复用,原因是其存取速度快。所谓地址引脚不复用是指,访问存储芯片需要多少位地址,存储芯片就有多少个地址引脚,即一位地址对应一个地址引脚。而地址引脚复用则指的是两位地址分时使用一个地址引脚。数据引脚不复用,指的是一位数据需两个引脚。一个引脚用作数据的输入,即写入时用;另一个引脚则用来输出数据,即读出时使用。如图 5.8 所示的 F100474,其字位结构为 $1\text{ K} \times 4\text{ 位}$,因数据引脚和地址引脚都不允许复用,故有 10 个地址引脚,8 个数据引脚(4 个输入数据用,4 个读出数据用)。

BiRAM 芯片虽然速度快,但功耗大,集成度低且价格高,目前只能作为 Cache,随着计算机要求 Cache 容量不断加大,BiRAM 芯片在集成度上已很难再提高,为此人们已经研制出速度可与 BiRAM 媲美,每片容量达 1Mb 的 BiCMOS RAM 存储芯片。

3. MOS 型 RAM 存储芯片

MOS 型 RAM 芯片具有集成度高、功耗小、成本低、工艺简单等优点,仅在速度上尚不及双极型和 BiCMOS 型 RAM。MOS 型 RAM 芯片可分为 DRAM 和 SRAM 两种。DRAM 芯片不能长期保存信息,必须定时刷新,SRAM 芯片只要正常供电,所存信息即可长期保存,无须刷新。MOS 型 RAM 芯片代表着当代最先进的集成电路技术,90 年代前存储容量基本上以每两年翻四倍的速度增长,90 年代 DRAM 仍以两年四倍的速度增长,而 SRAM 自 16Mb 产品出现后,已经失去以前那样大容量化的势头。究其原因,一是 SRAM 位元面积约为 DRAM 位元面积的 4 倍,同容量的价格为 DRAM 的 10 倍,与 DRAM 比较,制造工艺复杂,再增加容量很困难;二是用户认为现有片容量已经足够大;三是出现了 SRAM 替代品,如 CDRAM(Cached DRAM)存储芯片等。表 5.2 列出 MOS 型 RAM 的技术发展情况。表中带有 * 的为实验室产品。

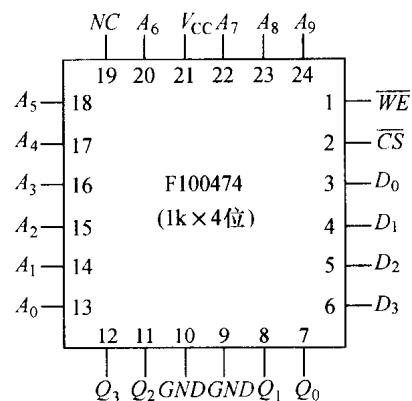


图 5.8 四方扁平封装 F100474 存储芯片引脚示意

表 5.2 MOS 型 RAM 芯片技术发展

年度	1981	1983	1985	1987	1989	1991	1993	1995	1997
DRAM 芯片容量(位)	64 k	256 k	1 M	4 M	16 M	64 M	256 M	1 G	4 G *
SRAM 芯片容量(位)	16 k	64 k	256 k	1 M	4 M	16 M			
最小线宽(μm)	3	2	1.2	0.8	0.6	0.25	0.2	0.16	0.12

目前 DRAM 芯片都采用单管位元电路结构,已研制出 4Gb 芯片,256Mb 芯片已投入市

场。美国半导体技术规划声称,到2010年,DRAM芯片集成度可达64Gb,最小线宽0.07μm。近几年,因生产大容量DRAM芯片所需工具投资巨大加之市场需求不迫切,单片DRAM大容量化趋势已不明显,单片超过4Gb的DRAM芯片市场上也未见到。

(1) SRAM 存储芯片

SRAM芯片位元电路所需元件较多,集成度不及DRAM芯片,但SRAM较DRAM有速度快和不需要刷新的优点。因此目前仍然是DRAM不可替代的半导体存储器件。

SRAM存储芯片容量从4 Kb至16 Mb有多种,主流芯片为256 Kb、512 Kb、1 Mb和4 Mb。从制造工艺分为NMOS、COMS和BiCMOS三类。三类比较,BiCMOS的SRAM芯片速度最快,容量适中,功耗最大,CMOS的SRAM芯片功耗最小,容量最大,速度仅次于BiCMOS的芯片;NMOS的SRAM芯片速度最慢,现在已经很少生产。目前广泛被采用的是CMOS和BiCMOS芯片,典型取数时间20~30 ns,最快只有几个ns。表5.3、表5.4示出常用的BiCMOS、NMOS和CMOS的SRAM存储芯片的型号和性能参数。

表 5.3 BiCMOS SRAM 存储芯片

型号	容量(字×位)	MAT(ns)	工作功耗	封装形式
μPD100494	16 K × 4	6	≤1 W	28脚DIP
NM100494	16 K × 4	12	≤1 W	28脚DIP
NM100490	64 K × 1	12	≤1 W	22脚DIP
NM100504	64 K × 4	15	≤1 W	28脚DIP
NM100500	256 K × 1	15	≤1 W	24脚DIP
μPD461008	128 K × 8	8	925 mW	32脚SOJ

表 5.4 SRAM 存储芯片

型号	容量(字×位)	MAT(ns)	功耗(工/维 mW)	工艺	封装形式
NMC2148H	1 K × 4	45	900/100	NMOS	18脚DIP塑
NMC2147H	4 K × 1	35	900/150	NMOS	18脚DIP塑
ATT7C116	2 K × 8	25	500/75(100 μW)	CMOS	24脚DIP/SOJ
ATT7C185	8 K × 8	25	750/75(500 μW)	CMOS	28脚DIP/SOJ
ATT7C164	16 K × 4	20	500/75(500 μW)	CMOS	24脚DIP/SOJ
CXK58258AP	32 K × 8	15,20,25	500,425,375	CMOS	28脚DIP/SOJ
ATT7C108	128 K × 8	15~25	880/110	CMOS	32脚DIP/SOJ
ATT7C106	256 K × 4	15~25	635/110	CMOS	28脚DIP/SOJ
ATT7C107	1 M × 1	15~25	635/100	CMOS	28脚DIP/SOJ
CXK584001P	512 K × 4	100,120	1.2/3 μW	CMOS	32脚DIP/SOJ

Motorola公司生产的SRAM存储芯片MCM60256A的内部结构和引脚示意如图5.9所示。

示。它有 28 个引脚。其中 $A_0 \sim A_{14}$ 为地址端、 \bar{W} 写使能端、 \bar{E} 片使能端、 \bar{G} 输出使能端、 $DQ_0 \sim DQ_7$ 双向数据端、 V_{CC} 接 +5 V, V_{SS} 接地。输入和输出电平与 TTL 兼容, 输出为三态门。256 K 个存储位元排成 512 行 \times 512 列的存储矩阵。 $A_5 \sim A_9, A_{11} \sim A_{14}$ 九位行地址译出 512 根行驱动线。 $A_0 \sim A_4, A_{10}$ 六位列地址译出 64 根列驱动线, 每根列驱动线同时连接存储矩阵中的 8 根列线, 这样构成了 8 个 512×64 的位面, 每当给定一个地址时, 选中存储矩阵中的 1 根行驱动线, 1 根列驱动线, 其连接 8 根列线。注意, 8 根列线分布于 8 个不同的位面, 从而实现整个存储单元的读写。该芯片有自动下电功能, 即维持状态时电压降到 2 ~ 3 V。

对于 SRAM 存储芯片, 地址引脚不允许复用。当芯片字位结构为多位结构时, 数据引脚进行复用。注意, 数据引脚复用时, 需增加一个输出使能端引脚 OE 或 \bar{G} , 以决定数据引脚何时作为输入, 何时作为输出。如图 5.9 所示的 32 K \times 8 位的 SRAM 芯片, 地址引脚不复用需 15 个, 数据引脚复用, 需要 8 个。当芯片的字位结构为 1 位结构时, 数据引脚没有必要复用, 因数据引脚复用的目的是为减少芯片总的引脚个数, 且这是以增加芯片复杂度和降低存取速度换来的, 此时, 数据引脚再复用, 既增加了芯片的复杂性而且芯片引脚又没有减少, 从而没有实际意义。

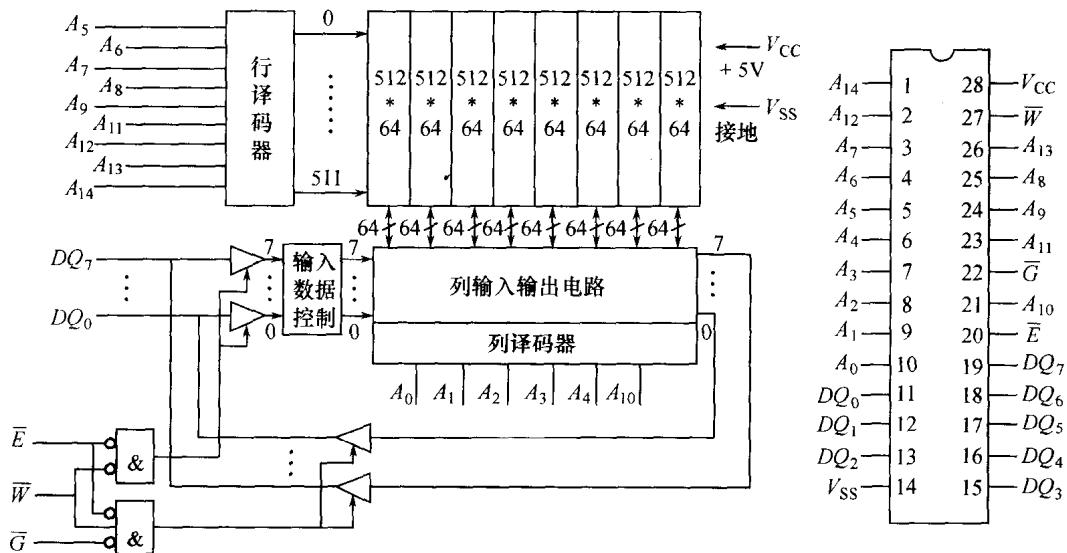


图 5.9 MCM60256A SRAM 存储芯片

(2) DRAM 存储芯片

同 SRAM 存储芯片比较, DRAM 存储芯片既有位元结构简单、存储容量大、功耗小、引脚少和价格便宜的优点, 又存在速度慢、控制复杂、需要定时刷新的缺点。

DRAM 芯片容量小至 4 Kb, 大到 4 Gb。主流芯片为 4 Mb、16 Mb、64 Mb 和 256 Mb。典型行选通取数时间 T_{RAC} 为 60 ~ 80 ns, 最快达 30 ns, 存储周期 120 ns 左右。表 5.5 列出 8 种

DRAM 芯片的型号及性能参数,带 * 者为 FPM DRAM 芯片。按照制造工艺可把 DRAM 芯片分为 NMOS DRAM 和 CMOS DRAM 两类。NMOS DRAM 芯片完全由 NMOS 管和电容构成,CMOS DRAM 芯片外围电路采用 CMOS, 存储矩阵由 NMOS 管和电容构造。采用 CMOS 作外围电路的优点是无静态功耗、输出电平幅度大、工作时不需要预充电。可采用全静态电路和简单的时序控制逻辑,时钟数少、速度快、工作可靠。

表 5.5 DRAM 存储芯片

型号	容量 (字 × 位)	MAT (ns)	MCT (ns)	功耗 (工/维 mW)	刷新周期 /刷新时间	封装形式
MCM4027	4 K × 1	200	375	470/36	64/2 ms	16 脚 DIP
MCM4517	16 k × 1	100 ~ 200	180 ~ 375	170/14	128/2 ms	16 脚 DIP
MCM6665	64 k × 1	150 ~ 200	280 ~ 375	303/22	128/2 ms	16 脚 DIP
* MB81C4256	256 k × 4	60 ~ 100	130 ~ 180	407 ~ 297/11	512/8.2 ms	20 脚 DIP
MB81C1000A	1 M × 1	60 ~ 100	130 ~ 180	407 ~ 297/11	1024/16.2 ms	18 脚 DIP
* MB814100A	4 M × 1	60 ~ 80	120 ~ 155	495 ~ 383/11	1024/16.2 ms	20 脚 DIP
MB8116400	4 M × 4	60 ~ 80	110 ~ 150	550 ~ 440/11	4096/65.6 ms	28 脚 TSOP
* MB8116100	16 M × 1	60 ~ 80	130 ~ 150	550 ~ 440/11	4096/65.6 ms	24 脚 SOJ

表 5.5 中,有些芯片的 MAT、MCT 和工作功耗不是惟一的,原因是同一种型号芯片可以有几种速度。当 MAT 不同时,MCT 和工作功耗也不同。速度越快、功耗越大,这里 MAT 指的是行选通取数时间 T_{RAC} 。

DRAM 存储芯片将结合 5.4 节内存储器设计进行介绍。对于 DRAM 存储芯片,当芯片字位结构为多位结构时,地址引脚和数据引脚都应复用。当字位结构为 1 位结构时,地址引脚复用,数据引脚不必复用,原因同 SRAM。注意,DRAM 芯片需增加 \overline{RAS} 、 \overline{CAS} 两个控制引脚,以便输入行、列地址。

4. 新型半导体 RAM 芯片

RAM 芯片速度远远跟不上 CPU 芯片速度增长,通过向 CPU 芯片增加一个 ALU 可以很容易地提高 CPU 性能,通常增加两倍的 ALU 几乎可获得两倍的 CPU 速度。然而对于 RAM 芯片,更多的存储位元只能意味着能够容纳更多的信息,却不能提高速度。为提高 RAM 的速度与功能,近几年出现多种新型 RAM 芯片。这些芯片通常采用两种技术,一种是 IRAM (Integrated RAM) 技术,即集成技术,有的文献亦称之为一体化技术。这种技术的特点是,在一个芯片内,除集成存储矩阵、行列译码与读写控制、地址输入和数据输入输出缓冲器外,还集成控制逻辑及时序产生器、刷新逻辑、访存与刷新裁决器、多个输出缓冲器,有的甚至集成不同种类的存储位元电路构成两种或两种以上存储矩阵和与之相关的电路。另一种是 ASIC RAM (Application Software Integrated Circuit RAM) 技术,采用 ASIC RAM 技术的存储芯

片,其根据用户需求而设计的专用半导体 RAM 芯片,它以 RAM 为中心,结合其他处理逻辑,能提高存储器系统性能和缩短存取时间。

(1) 高速缓冲动态随机存储器(Cached DRAM, CDRAM)

CDRAM 是在普通 DRAM 芯片上再集成一个 SRAM 存储矩阵和有关缓冲寄存器及逻辑控制电路作为片内 Cache, Cache 和 DRAM 通过片内总线连接。如 Mitsubishi 公司 16 Mb CDRAM 芯片中,内置 16 Mb 的 DRAM 并附加 16 Kb 的 SRAM 作为 Cache, 在一个 DRAM 存储周期传送 128 位到 Cache 中。这样从总体上看,CDRAM 具有较高速度,而成本比不在同一芯片上独立的 Cache、DRAM 要低得多。

三菱公司生产的 M5M44409TP 是 4 Mb 的 CDRAM, 它在一个芯片上集成地址、命令、数据输入输出和控制时钟四个缓冲寄存器,一个 $1\text{ M} \times 4$ 位的 DRAM 作为 CDRAM 的 DRAM, 一个 $4\text{ K} \times 4$ 位的 SRAM 作 CDRAM 的 Cache。DRAM 与 Cache 成组传送信息,一个 DRAM 存储周期(80 ns)传送 16 个数据(64 位)到 Cache 中,比不在同一芯片上独立的 Cache、DRAM 传输速度快了 16 倍。

(2) 快速页模式动态随机存储器(Fast Page Mode DRAM, FPM DRAM)

典型 DRAM 芯片读写时,先给出行地址选通信号打入行地址,然后给出列地址选通信号打入列地址,从而选中某存储位元实现读写。FPM DRAM 芯片在典型 DRAM 芯片基础上增加一些逻辑电路,使之能实现页面存取方式。页面存取方式是指在同一行地址下,快速存取不同列地址存储位元的信息。通常称同一行上所有不同列的存储位元构成的位片为一个页面,其大小从 256b ~ 4096b 不等,完全由芯片内部结构决定。

实现页面存取的方法是,维持行地址和行地址选通信号不变,而让列地址及其选通信号周期性变化,从而实现多个连续列地址中所存放信息的读写。页存取方式的第一个存储周期和典型 DRAM 存储周期时间是相同的,其他存储周期因不需等待行地址,约为典型 DRAM 存储周期时间的三分之一。如表 5.5 中 MB8116100 芯片属于 FPM DRAM 芯片,在快速页面模式下存储周期为 40 ~ 50 ns。这类芯片对主存、辅存或主存、Cache 成块交换信息非常有利,但对读取不连续地址中的信息则无好处可言。

(3) 同步动态随机存储器(Synchronous DRAM, SDRAM)

SDRAM 基于双存储体结构,内含两个交互工作的 DRAM 存储矩阵。工作时 CPU 和 SDRAM 通过一个时钟锁在一起,使 SDRAM 和 CPU 共享一个时钟周期,以相同的速度同步工作。当 CPU 正从一个存储矩阵存取数据时,另一个已经准备好读写的数据,通过两个存储矩阵的紧密切换,存取速度成倍提高。SDRAM 与系统时钟同步,采用流水线处理方式,当指定一个特定地址,SDRAM 就可读写多个数据,即实现突发传送。以读为例,第一步,指定地址;第二步,把数据读出送到输出电路;第三步,输出数据到 CPU。这三步是分别独立进行的,且与 CPU 同步。这就是 SDRAM 高速的关键所在。而非 SDRAM 存储芯片是从头到尾执行完这三步,即三步是一个整体,然后才能对下一个地址进行读写,因此速度慢。目前

SDRAM 的读写周期最快可达 7 ns。最近韩国三星公司宣布已研制出 1 Gb 速度达 31 ns 的 SDRAM 芯片, 图 5.10 给出了 32 Mb SDRAM 的逻辑结构。

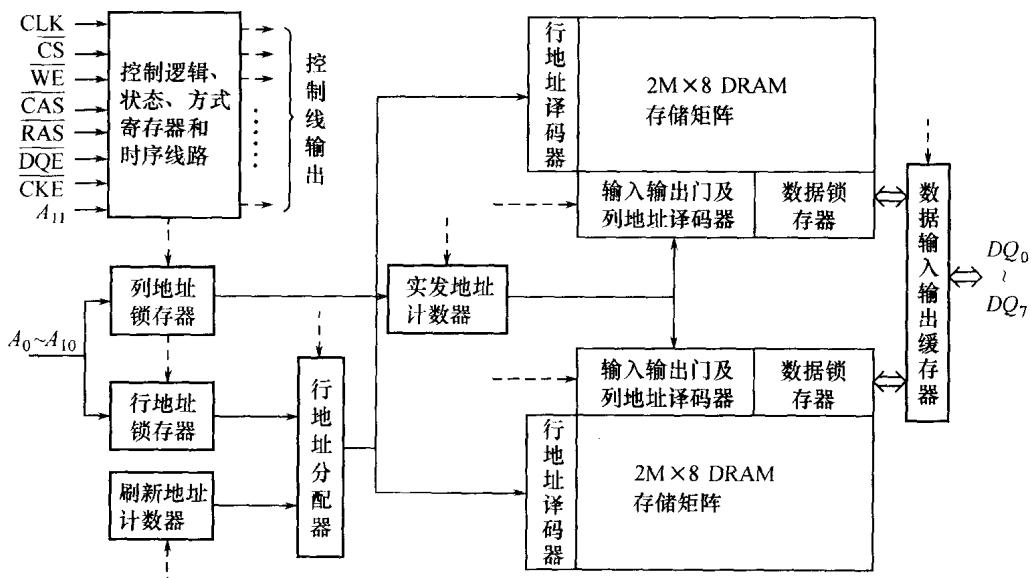


图 5.10 4 M × 8 位 SDRAM 存储芯片逻辑结构

(4) 同步内存 II(SDRAM II)

亦称 DDR(Double Data Rate), 是 SDRAM 的更新产品。DDR 的核心仍建立在 SDRAM 基础上, 但在速度和容量上都有提高。同 SDRAM 相比, 主要有两点不同: 首先它使用了更多、更先进的同步电路; 其次是增加了延时锁定回路(Delay Locked Loop, DLL), 当数据有效时, 存储控制器可通过它精确定位数据。它允许在时钟的上升沿和下降沿读出数据, 因此其速度是标准 SDRAM 的两倍。DDR 亦可使用更高的频率, 因此带宽(传输率)将大大提高。

(5) RDRAM(Rambus DRAM)

RDRAM 是 Rambus 开发的 DRAM, 使用独特的信号逻辑(Rambus Signaling Logic, RSL)技术, 能在常规的系统上达到 600 MB/s 的传输率。其关键部分除 DRAM 外, 还有专用集成电路单元(Rambus ASIC Cells)和被称为 Rambus 通道的内部互连电路。Rambus 公司目前有 RDRAM 和 Concurrent RDRAM 两种产品。RDRAM 使用低电压信号和 8 位接口, 在高速同步时钟的上升和下降的两边沿传输数据。Concurrent RDRAM 是 RDRAM 的增强型产品, 它在同步并发块数据导向和交叉传输时更有效, 数据传输率可达 800 MB/s 且同 RDRAM 兼容。在图形处理和运行多媒体程序时更能充分发挥高速传输的作用。

(6) 同步链接存储器(Synclink DRAM, SDRAM)

SLDRAM 是一种增强和扩展的 DDR, 它将 4 体 DRAM 结构的 DDR 增加到 16 个或更多, 并增加了新接口和控制逻辑电路。和 DDR 一样也是通过每个脉冲的两个边沿传输数据。

SLDRAM 在速度上可与 RDRAM 媲美。

(7) 视频随机存储器(Video RAM, VRAM)和其他图形图像视频专用芯片

高性能视频显示系统要求存储器能够按位、按行进行高速显示, VRAM 就是为满足这个需求而设计的。VRAM 采用双端口设计,一个端口接收处理器处理好的视频信息,另一个端口则快速向显示系统提供高速串行的信息,以便显示输出,前者为写端口,后者为读端口。通常片内除有容量较大的 DRAM 存储器外,还有一个位数为 256 或 512 具有移位功能的寄存器。

为获得更高的视频处理功能,利用高速缓冲和同步技术,派生出 CVRAM (Cached VRAM) 和 SVRAM (Synchronous VRAM) 两种新型 VRAM 芯片,它们的速度比 VRAM 更快,更适合高性能显示系统要求。

为适应多媒体计算机对图形图像及动画处理的需求,Samsung 公司在 DRAM 基础上增加一些图形图像及动画处理的智能电路,如增强屏幕重画能力的彩色写块和增加提高字符显示性能的逻辑电路生产出双端口 WRAM (Windows RAM) 芯片。Mitsubishi 公司生产出更为复杂的 3D RAM 芯片,主要用于三维视频处理。芯片内部集成多个 Z 缓冲电路和一个 ALU,使得芯片除能执行一般算术逻辑操作外,还可进行光栅、Alpha 混合与比较等视频图像的专有操作。采用 3D RAM 设计的视频卡,比采用 VRAM 的要快一个数量级。

近几年,新型半导体存储芯片层出不穷,上面介绍的是应用广泛的存储芯片。实际上还有很多种,如增强型动态随机存储芯片(Enhanced DRAM, EDRAM)、多体动态随机存储芯片(Multibank DRAM, MDRAM)、混合页式动态随机存储芯片 HPM DRAM、同步突发静态随机存储芯片(Synchronous Burst SRAM, SB SRAM)、管道突发静态随机存储芯片(Pipelined Burst SRAM, PB SRAM)、双端口静态随机存储芯片(Dual - Port SRAM, DP SRAM)和多端口静态随机存储芯片(Multi - Port SRAM, MP SRAM)等。其中,SB SRAM 地址取数时间可达 5 ns, PB SRAM 地址取数时间可小于 4 ns。

5. 铁电随机存储器(Ferroelectric RAM, FeRAM)

铁电薄膜有两种极化方向,两种极化方向翻转只需几伏的诱导电压,完成极化翻转的时间仅为 10^{-12} s (ps) 量级,人们就是利用铁电薄膜的这些特性制作存储记忆器件的。目前广泛使用的是被称为 PZT 的铁电薄膜,它是由 PbZrO₃ 和 PbTiO₃ 作原料,通过溶胶,凝胶 (Sol - Gel) 法制成。

用 PZT 制作存储器件有两种方法。一种是用 PZT 代替场效应管的栅介质,这种场效应管开启电压有两种明显不同的值,该值与 PZT 的极化方向有关。用这种方法制作的器件本身具有放大功能,可实现无数次的读写,且存储位元仅用一个场效应管,单片存储容量很大。缺点是信息容易丢失,PZT 也容易被损坏,所以目前用的很少。第二种方法是铁电薄膜介质电容器件,简称铁电电容,该器件由两层金属板电极间夹一层 PZT 介质构成。用 MOS 管和铁电电容构成存储位元,这种存储位元可靠性高,但读写次数受限,仅能读写 10^{12} 次。

FeRAM 属非挥发性存储器,信息一经写入,不改写,则会永远储存,即使去掉电源,信息也不会丢失。其读写速度也比较快,目前在 20 ns 以下,大部分时间延迟是由 PZT 中杂质的杂散效应引起的。PZT 固有开关时间只有 1 ns,杂质越少,杂散效应越小,时间延迟也会减少,速度可望提高到 15 ns 以下。 $1 \mu\text{m}^2$ 的 PZT 电容可在位线上产生 200 mv 的电位差,加之电源电压仅几伏,因此集成度可做得比 DRAM 大。现在市场上已有每片 16 Kb、256 Kb 产品出售,每片 1 Mb、4 Mb 也已研制成功。

目前 FeRAM 有两个突出的问题需要解决,对于用 PZT 代替场效应管栅介质构造的存储位元,主要解决可靠性问题。对于铁电电容构造的存储位元,主要解决极化读写次数受限问题。上述两个问题只要能解决一个,FeRAM 将真正成为 DRAM、SRAM 的替代品。

5.2.4 只读存储器 ROM

ROM 是随机存储器的一种,它是按地址寻址,CPU 只能随机读出,不能随机写入的存储器。目前的 ROM 同 RAM 一样也大都由半导体制作的。ROM 优点是信息的非挥发性、存储容量大、可靠性和集成度高、速度比 DRAM 快与 SRAM 相当、非破坏性读出,在大批量生产的情况下可以实现低成本;其缺点是所存储的信息不可更改或更改信息较为麻烦。

ROM 目前依写入方式,可分为 3 类 5 种。第一类是固定掩模型 ROM (fixed Mask ROM),缩写为 MROM。它是在制造时使用掩模工艺将信息写入存储器,用户不能作任何改动,它象印好的书刊、文件、杂志等,只能阅读,不能修改。第二类是一次可改写 ROM,即 PROM(Programmable ROM)。生产厂家在制造存储器时,就写入全“1”或全“0”,用户依需要在使用前可改写一次,且只能改写一次,它象用白纸印书刊、文件或杂志,只能印一次,印完后不能改动。第三类是可多次改写的 ROM,生产厂家在制造这种 ROM 时,也是写入全“1”或全“0”,用户根据自己需要可多次改写,它象黑板,可多次擦除,多次改写。在改写前都要先擦除,因擦除的方式不同,这类 ROM 又分为 3 种。第一种是紫外线擦除电可编程 ROM,即 UV EPROM(Ultra Violet Erasable PROM),通常为了简便,缩写为 EPROM. 第二种是电擦除电可编程 ROM,即 EEPROM(Electrically Erasable PROM)。第三种是快擦写存储器 FM(Flash Memory),亦称快闪存储器。该类存储器是电整片(或块)擦除电可编程的 ROM。

ROM 用途十分广泛,除存储微程序、固定常数外,还可用于各种字符发生器、函数查表器、汉字库、语音合成和固定程序的存储器。有些复杂运算,如三角函数与反三角函数的计算,可以通过查表得到,而且速度比正常运算逻辑的速度快。遍及世界的电子游戏卡就是编有各种游戏程序的 ROM。随着集成电路技术水平的提高,集成度的加大,不少软件也可用 ROM 进行硬化。总之,ROM 的应用前景非常广泛。

1. MROM

依用户要求,由厂家按确定工序,用是否制造元器件的方法,实现信息写入,制造完成后,不能再改变的 ROM,称之为 MROM。

半导体集成电路,是通过外延、氧化、光刻、蒸发、检测、封装等一系列工序制作的。各种元件尺寸和它们之间的连接,都是要靠光刻工艺来完成。所谓光刻是通过掩模版曝光刻出图形。因此在制作这类 ROM 时,则根据用户要求,在制作掩模版时,就把信息编排在掩模版里。如双极工艺中去掉三极管一个极(基极或射极),或者在 MOS 工艺中去掉晶体管的栅极等等。用这种办法制作的存储器,它所存的信息和掩模版是一致的,是不能改变的,故称作固定掩模型。图 5.11 所示为重合法双极掩模型 ROM 的逻辑框图。

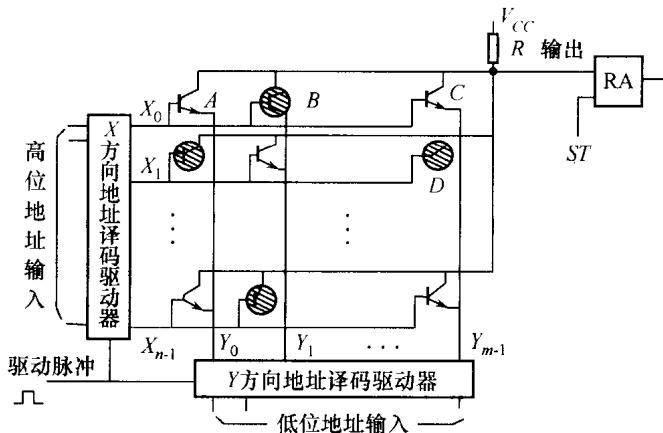


图 5.11 双极固定掩模型 ROM

图中展示的是 $n \times m$ 矩阵的一位,有三极管的地方表示信息“1”,打斜线的地方,由于被掩模遮盖,没做成三极管,表示信息“0”。

平时无驱动脉冲时, X 方向地址译码驱动器输出都为低电位, Y 方向地址译码驱动器输出都为高电位,所以整个存储矩阵的三极管都处于截止状态。

读出时也和随机存储器一样,要先把欲读出存储单元的地址分别送到 X 、 Y 方向地址译码驱动器,经过选址译码,当驱动脉冲到来时, X 方向和 Y 方向各有一条驱动线被选。假定读 0 号单元,则 X_0, Y_0 被选,则 X_0 为高电位, Y_0 为低电位。这时,驱动线 X_0 和 Y_0 交叉处 A 存储位元被选。 A 处有三极管存在,并处于开通状态,于是有电流经电阻 R 流过三极管,则输出为低电位,表示读出的是“1”。如果 A 处不存在三极管,则没有电流通路,输出为高电位,表示读出的是“0”。

MROM 也可用 MOS 集成工艺来实现,其结构和图 5.11 类似。MROM 适合于大批量生产,此时成本低。所以 MROM 适合作代码转换、各种字符库,汉字库、函数查表器等标准的大批量产品。

目前 MROM 芯片存储容量最高达 32 Mb,最低仅为 2 Kb。字位结构主要有 $m \times 4$ 位、 $m \times 8$ 位和 $m \times 16$ 位三种,后两者占多数,近几年几乎不再生产 $m \times 4$ 位的芯片。 m 为 2^n , n 为 8~22 中的任一整数,即 m 为 256~4 M。制造工艺有 TTL、NMOS、CMOS 和 BiCMOS 四种。

取数时间以 BiCMOS 最快, 可达 15 ns, 通常多为 70 ns, 最慢, 为 500 ns。图 5.12 所示是型号为 MB831000 字位结构为 $128 \text{ K} \times 8$ 位的 MROM 芯片引脚图。图中 A_i ($i = 0, 1, \dots, 16$), O_j ($j = 1, 2, \dots, 7$) 分别为地址和数据引脚, 且只有一个控制引脚 \overline{CE} 。其取数时间为 150 ns, 28 引脚 DIP 封装。两片该种芯片可构成 16×16 点阵的简易汉字库。繁体汉字通常采用 64×64 点阵表示, 目前已有单片 32 Mb 的 MROM 繁体汉字库产品。

2. PROM

MROM 只能用于特定场合, 用户使用受到很大限制。为此, 人们采用双极工艺制造出了在一定程度上克服了 MROM 缺点的 PROM。PROM 存储位元的基本结构有全“1”熔断丝型和全“0”肖特基二极管型两种, 如图 5.13 所示。

关于全“1”熔断丝型存储位元是很容易想象到的。在电源的回路中, 经常通过串联保险丝来保护线路, 即当超过额定负载时, 保险丝就会被熔断, 使电源和线路得到保护。如果在存储矩阵的各个存储位元(即晶体管)的电路中, 串联一个熔断丝, 如图 5.13(a)所示, 在正常工作电流下, 熔断丝不会被烧断, 而当通过几倍工作电流的情况下, 熔断丝会立即被烧断。在存储矩阵中, 当熔断丝被烧断的存储位元被选时, 构不成通路没有电流, 表示存储信息“0”, 熔断丝保留的存储位元, 当被选时, 三极管导通, 回路有电流, 表示存储信息“1”。与此类似的办法, 是用肖特基二极管来代替熔断丝, 如图 5.13(b)所示。当作用于肖特基二极管两端的反向电压超过它的击穿电压时, PN 结将被击穿, 它的反向特性遭到破坏, 且不能恢复, 并呈低阻抗。这正好与熔断丝的情况相反, 即肖特基二极管加反向电压击穿后, 呈低阻抗, 当存储位元被选时, 可形成电流通路, 表示存储信息“1”, 对于二极管没有被击穿的存储位元, 仍然是高阻抗, 当存储位元被选时, 没有电流通路, 表示存储信息“0”。

根据上述思想, 可组成全“1”信息熔断丝型只读存储器的存储矩阵如图 5.14 所示, $m \times n$ 的矩阵, 字驱动线 W_i ($i = 0, 1, \dots, m - 1$), 位线 W_j ($j = 0, 1, \dots, n - 1$), 阵列中每个存储位元由 1 个三极管和 1 个熔断丝组成, 可存储一位信息。

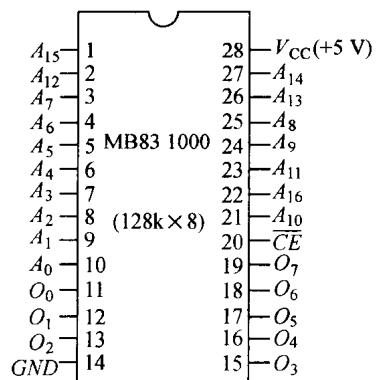


图 5.12 128 k × 8 位 MROM 芯片引脚图

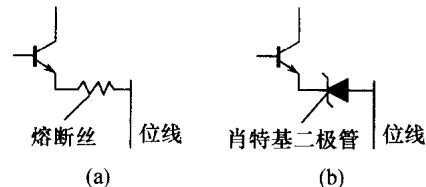


图 5.13 PROM 存储位元基本结构

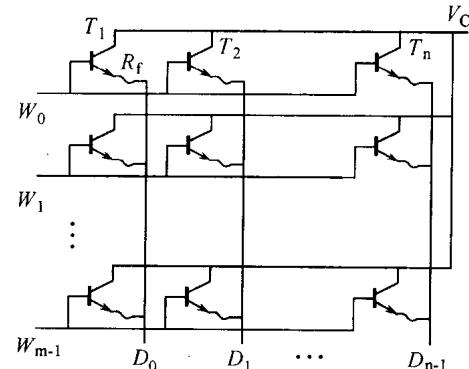


图 5.14 双极熔断丝型 PROM 存储矩阵

在写入状态,也就是用户在使用前进行编程时,按给定地址译码后,加到驱动线 W_i 上的是个比较高的电位,根据要改写的信息不同,在位线 D_j 上加不同的电位,例如第一个位元要写“1”,则对应位线 D_0 悬空(或接较大电阻)而使流经被选存储位元的电流很小,不足以烧断熔断丝,该存储位元仍保持“1”状态,相反如该位要写“0”,则位线 D_0 加负电位(-2 V),则瞬间通过被选存储位元的电流很大,致使熔断丝烧断,即改写为“0”。同理可改写其他存储位元。

在正常只读状态工作时,给定地址经译码后,加到字驱动线 W_i 上的是比较低的脉冲电位,但足以开通存储位元中的晶体管。这样,被选字驱动线 W_i 上存储位元的信息就一并读出了。是“1”则对应位线有电流,表示读出是“1”;是“0”则对应位线无电流,表示读出是“0”。由于每个被选存储位元中的三极管是工作在射极跟随器状态,当射极负载电阻一定时,流经被选存储位元的电流大小就取决于加到驱动线 W_i 上的电位。所以在只读状态,加到驱动线 W_i 上是较低的脉冲电位,工作电流将很小,不会造成熔断丝熔断,当然也就不会破坏原存信息。

不难看出这种只读存储器只能作一次编程写入,一旦写好后就不能再修改。因为熔断丝是位于芯片内,熔断后无法再接通。熔断丝型 PROM 的关键是熔断丝。熔断丝能否烧断,主要是取决于熔断丝的材料和阻值大小,阻值小需要的熔断电流大,同样,材料熔点高,需要熔断电流亦大,相反阻值大的熔点低,需要熔断电流小。但熔断电流又不能太小,如果小到和正常只读状态的工作电流相接近,则在正常只读工作条件下,会造成熔断丝破坏,影响可靠性。经验数据是熔断电流与工作电流之比在 10 左右比较合适。其次在材料上,一般是采用镍铬合金做熔断丝,因为这种材料用蒸发的办法做成一定阻值,较为容易控制,并且在熔断时不易产生飞溅现象。

PROM 芯片集成度达 512 Kb,ECL 小容量 PROM 取数时间在 3 ~ 10 ns,典型取数时间为几十 ns。PROM 芯片引脚和 MROM 一样,除电源引脚和若干个地址数据引脚外,也仅有一种片选(或片使能)端引脚。注意,PROM 是按位编程的。即选定一地址,依次对该地址的所有位分别编程,一个地址编程完毕,再改变地址,如上步骤,完成整片所有地址的编程。PROM 通常都是使用编程器实现编程。

3. EPROM

MROM 所存信息不能修改,PROM 也只能改写一次,不能再修改。从经济和使用的角度都要求开发可多次改写的 ROM。人们先开发出 EPROM。目前,EPROM 绝大多数采用叠栅注入 MOS(Stacked - gate Injection Mos,SIMOS)管。

(1) SIMOS 管结构、存储位元和编程机理

SIMOS 管结构如图 5.15(a)所示。它属于 NMOS,与普通 NMOS 不同的是有两个栅极,一个是控制栅 CG,另一个是浮栅 FG,FG 在 CG 的下面。两个栅极都是由多晶硅制作,FG 被 SiO_2 所包围,与四周绝缘。FG 与衬底 P-Si 表面之间有很薄的 $25 \sim 100 \text{ nm}$ ($1 \text{ nm} = 10^{-9} \text{ m}$)

的 SiO_2 , 沟道长度 $3 \sim 4 \mu\text{m}$ ($1 \mu\text{m} = 10^{-6} \text{ m}$)。目前高集成度的芯片沟道长度小到 $0.8 \mu\text{m}$ 。

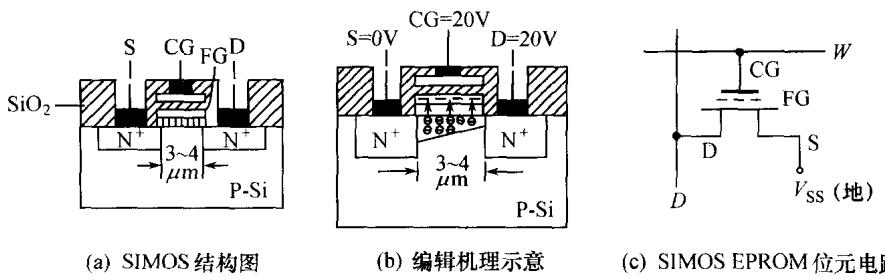


图 5.15 SIMOS EPROM 存储信息原理

单个 SIMOS 管构成一个 EPROM 存储位元, 如图 5.15(c) 所示。与 CG 连接的线称为字 (W) 线, 读出和编程时作选址用。漏极与位线 D 相连接, 读出或编程时用来输出、输入信息。源极接 V_{ss} 即接地。FG 上没有电子驻留, CG 开启电压为正常值, 通常在 0.8 V 左右。若 W 线上加高电平, 一般为 +5 V, 源、漏间也加 +5 V 电压, SIMOS 形成沟道并导通, 称此状态为“1”。FG 上有电子驻留, CG 开启电压升高超过 +5 V, 若 W 线加高电平 +5 V, 源、漏间仍加 +5 V 电压, SIMOS 不导通, 称此状态为“0”。人们就是利用 FG 上有无电子驻留来存储信息的, 有电子时存储的信息为“0”, 无电子时存储的信息为“1”, 反之亦可。因 FG 上电子被绝缘材料包围, 不获得足够能量很难跑掉, 所以可以长期保存信息, 即使断电也不丢失。

SIMOS EPROM 厂家出厂的产品 FG 上是没有电子的, 即都是存“1”位元。对它编程, 就是向某些位元的 FG 注入一定数量的电子, 将它们写为“0”。擦除, 就是设法将 FG 中的电子清除掉, 将它们写成“1”。

擦除: 用光强在 12 mW/cm^2 左右的紫外光, 照射 SIMOS 管 $15 \sim 20$ 分钟, 驻留在 FG 上的电子得到足够的能量, 跃迁回 P-Si 衬底, 使 SIMOS 成为低开启管, 实现擦除。

编程: SIMOS 的 CG 和漏极都加 +20 V 电压, 源极接地如图 5.15(b) 所示。此时沟道通导, 电子在高漏压作用下获得足够的能量, 变成热电子。部分热电子在 CG 强正电场作用下, 通过沟道时穿过很薄的 SiO_2 绝缘层进入 FG, 称这种物理现象为热电子隧道效应。随着时间的推移, FG 上电子越来越多并均匀分布在 FG 上, 去掉 CG 和漏极高压, 因 FG 被 SiO_2 包围, 电子驻留在 FG 上, 产生附加电场, 使 SIMOS 成为高开启管, CG 开启电压超过 +5 V, 写入“0”。编程时间将因绝缘层厚度和编程电压不同有较大差别, 最大编程时间为 50 ms 左右。

读出: SIMOS 的 CG 和漏极都加 +5 V 电压, 源极接地。若 SIMOS 导通, 说明 FG 上无电子, 处于低开启管状态, 读出“1”, 否则, 说明 FG 上有电子, 处于高开启管状态, 读出“0”。

编程之前都要先擦除, 然后再编程, 编程之后即可永久使用了。根据需要可多次重复进行擦除与编程。

(2) EPROM 存储芯片

EPROM 存储芯片种类繁多, 每片最小容量 1 Kb, 最高可达 64 Mb。目前, 这类芯片字位

结构只有 m 字 \times 8 位或 m 字 \times 16 位两类。如 4 Mb 芯片字位结构可为 $256 \text{ K} \times 16$ 或 $512 \text{ K} \times 8$, 1 Mb 芯片字位结构为 $64 \text{ K} \times 16$ 或 $128 \text{ K} \times 8$ 。表 5.6 给出了目前广泛使用的 EEPROM 存储芯片, 表的编程时间栏中的 B 表示字节, W 表示 16 位字。其中最高取数速度为 55 ns, 编程时间最快的是每个存储单元 $1.9 \mu\text{s}$ 。目前已生产出取数时间仅 15 ns 的 EEPROM 芯片。

表 5.6 UV EEPROM 存储芯片

公司	容量(b)	字位结构	编程电压 V_{PP}	编程时间	取数时间(ns)	封装及引脚个数
Intersil	16 k	$2 \text{ k} \times 8$	20 V	50 ms/B	200	24DIP
National	32 K	$4 \text{ k} \times 8$	12.5 V	10 ms/B	300	24DIP
Fujitsu	64 K	$8 \text{ k} \times 8$	21 V	20 ms/B	150	28DIP
National	128 K	$16 \text{ k} \times 8$	12 ~ 14 V	10 ms/B	120	28DIP
Hitachi	256 K	$32 \text{ k} \times 8$	12 ~ 14 V	0.5 ms/B	95	28DIP
Hitachi	1 M	$64 \text{ k} \times 16$	12.5 V	0.025 ms/W	55	40DIP
Hitachi	1 M	$128 \text{ k} \times 8$	12.5 V	0.5 ms/B	140	32DIP
Hitachi	4 M	$256 \text{ k} \times 16$	12.5 V	3.8 $\mu\text{s}/W$	55	40DIP
Toshiba	4 M	$512 \text{ k} \times 8$	12.5 V	1.9 $\mu\text{s}/W$	68	40DIP
NEC	16 M	$1 \text{ M} \times 16$	12.5 V	9.6 $\mu\text{s}/W$	85	42DIP
NEC	16 M	$2 \text{ M} \times 8$	12.5 V	4.8 $\mu\text{s}/B$	85	42DIP

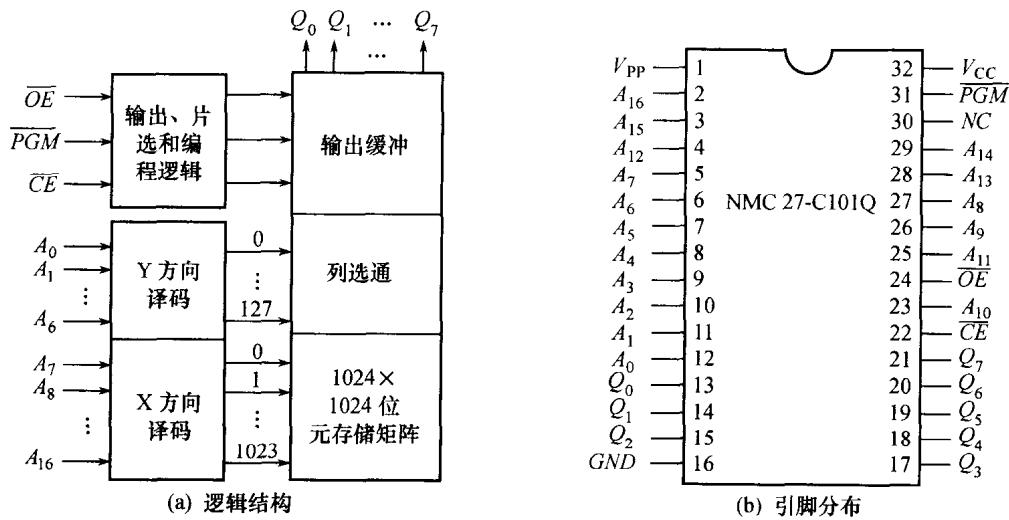


图 5.16 UV EEPROM NMC27C010Q 芯片的逻辑和引脚图

图 5.16 所示是型号为 NMC27C010Q、字位结构为 $128 \text{ K} \times 8$ 位的 1 Mb EEPROM 芯片的逻辑结构和引脚分布。图中, $A_0 \sim A_{16}$ 为 17 个地址端, 为访问 128 K 个存储单元提供地址。 Q_0

$\sim Q$, 只读时作为数据输出端; 编程时, 作为数据输入端。 \overline{CE} 、 \overline{OE} 、 \overline{PCM} 分别为片使能、输出使能和编程引脚。 V_{PP} 、 V_{CC} 为电源引脚, GND 为接地引脚, NC 为空闲未用引脚。

与 PROM 按位编程不同, EPROM 是以存储单元为单位借助于编程器编程的。若一个单元存储 8 位信息, 则一次写入 8 位; 若存储 16 位信息, 则一次写入 16 位。可连续按地址顺序编程, 也可直接对某个或某些地址单独编程。编程完成后要检查验证, 直到正确为止。EPROM 芯片不同, 使用的编程器不同, 具体编程步骤稍有差异, 但基本上是相似的, 具体细节要查阅编程器和具体芯片的说明。

根据有关文献介绍, 信息的保存时间, 在 125°C 时为 10 年, 常温下可保存 100 年。但改写的次数不宜过多, 随着改写次数的增加, 信息的保存时间将相应缩短。但一般正常使用, 可不用过多考虑。

4. EEPROM (E²PROM)

EPROM 从使用来看还是不方便, 主要是擦除时要把芯片从系统上取下来进行几十分钟的光照, 然后再用专门的编程器编程, 编好后再把芯片插回系统, 不能在线进行擦除和编程, 不能对芯片中个别需要修改的存储单元进行单独擦除和改写, 因此很不灵活。另外, 封装比较麻烦而且成本比较高, 原因是需要为擦除留出透明的石英窗口。平时使用时, 还要注意避免阳光和灯光尤其是荧光灯的直接照射, 以免引起芯片局部内容被擦除, 为此芯片窗口应该用不透明的物体遮挡, 为此开发了能在线编程和擦除的 EEPROM。

EEPROM 存储管分成两类, 两类存储管本质上都是一只 N 沟道 MOS 管。一类存储管是栅极的绝缘介质由两层不同物质构成, 另一类则与 EPROM 存储管一样是叠栅结构。我们仅介绍叠栅结构的被称为浮栅隧道氧化层 MOS (Floating - gate Tunneling Oxide MOS, FLOTOX) 存储管。

(1) FLOTOX 管结构、存储信息原理和存储位元

① FLOTOX 管结构:

和 EPROM 的 SIMOS 管(图 5.15)一样, FLOTOX 管也是叠栅结构, 如图 5.17 所示。不同之处在于, SIMOS 的 FG 和隧道氧化层都在沟道正上方, 隧道氧化层厚度至少为 25 nm 且面积较大; FLOTOX 的 FG 虽跨越整个沟道, 但隧道氧化层却仅在漏区的上方, 厚度不超过 20 nm 且面积很小。由于隧道氧化层厚度、面积和位置的不同, 导致两者在编程和擦除的实现上存在较大差别。SIMOS 擦除是通过紫外线光照, 编程则通过热电子隧道效应。

FLOTOX 管的 CG 与 FG、FG 的超薄氧化层区与部分漏区构成串接的两个平板电容 C_1 和 C_2 。在制作时, 使 C_1 电容很大, 编程和擦除都是借助这两个电容完成的。

② FLOTOX 管存储信息原理: 即如何擦除信息, 如何写入信息(编程)和怎样读出信息。

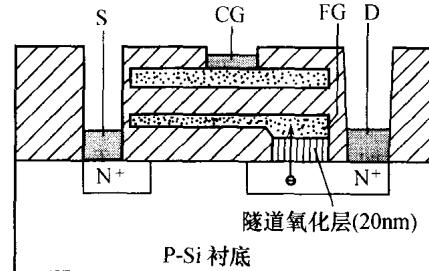


图 5.17 FLOTOX 管结构图

和 SIMOS 管一样,擦除是写“1”,编程是写“0”,且假定 FG 上有电子表示信息“0”,无电子表示信息“1”。

擦除:CG 接地,漏极加 +20 V,电容 C_1 、 C_2 分压使漏极和 FG 间存在强正电场(10^7 V/cm),FG 上的自由电子在强正电场作用下被拉回漏区。FLOTOX 管变为低开启管,写入“1”完成擦除,此时 FLOTOX 管门坎即开启电压很低,通常为 0.8 V。

编程:CG 加 +20 V,漏极接地,电容 C_1 、 C_2 分压使 FG 和漏极间存在强正电场,导致漏区的自由电子被强拉到 FG,FLOTOX 管变为高开启管,门坎即开启电压为 7 V,写入“0”,完成编程。

读出:编程后的 FLOTOX 管即可读出。通常 S、CG 和 D 极分别接地、+3 V 和 +5 V,若不导通,读出信息为“0”,否则读出信息为“1”。

在强正电场作用下,自由电子在漏(或源)与浮栅之间双向穿越薄氧化层的物理现象称否勒—诺德汉(Fowler-Nordheim)隧道效应,简称 F-N 隧道效应。由于自由电子在穿越薄氧化层时,沟道是不导通的,电子并未得到足够能量成为热电子,完全靠正电场的作用,故又称为冷电子隧道效应。

③ FLOTOX 管存储位元:如图 5.18 所示。

其中 T_1 和 T_2 构成存储位元, T_1 为选择管,是存储矩阵的一部分,是为说明问题才画出的。原则上,单个 FLOTOX 管即可构成一个存储位元电路。之所以采用双管位元主要是考虑可靠性,因为隧道氧化层很薄,要尽量减少它承受高压感应的可能性,否则氧化层将被破坏。采用双管位元,只有该位元被选中进行擦除或编程时,隧道氧化层才承受高压。未选中时,NMOS 管起到隔离高压的作用,从而保护了 FLOTOX 管。图中,WL 为字线,DL 为位线。

擦除:A 点接地,DL 接 +20 V, T_2 管 S 极浮空, WL 加 +20 V 的脉冲。使 T_2 管 FG 上的自由电子被拉回漏区, T_2 变为低开启管,写入“1”,完成擦除。

编程:A 点接 +20 V,DL 浮空, T_2 管 S 极接地址,WL 加 +20 V 的脉冲。使得 T_2 管 CG 为 +20 V 形成沟道,导致其漏极亦接地,使漏区的自由电子被强拉到 FG。 T_2 变为高开启管,写入“0”,完成编程。

读出:A 点接 +3 V, T_2 管 S 极接地,DL 接读出放大器 RA(图中未画出),WL 加 +5 V 的脉冲。若 T_2 不通导,说明它处于高开启管状态,RA 读出信息“0”,否则读出信息“1”。

(2) EEPROM 存储芯片

EEPROM 也是以存储芯片的形式提供给用户的。目前其最大片容量达 64 Mb,字位结

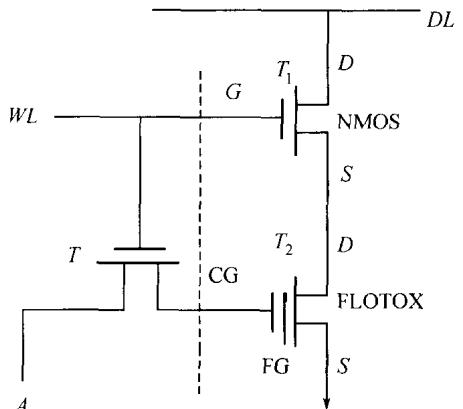


图 5.18 FLOTOX 双管位元电路

构分 m 字 \times 8 位、 m 字 \times 16 位和 m 字 \times 32 位三种形式, 最快取数速度 35 ns。引脚分布可以和 SRAM、EPROM 引脚相一致。新的数据写入 EEPROM 像写入 RAM 一样实现。EEPROM 要求每个写入操作之前都能自动执行一个擦除操作, 整个写入操作一般要 10 ms 才能完成。目前快速 EEPROM 芯片有的只需 2 ms。同 RAM 比较, 写操作比读操作要慢得多, 读写操作是不对称的, 好在这种情况下对 EEPROM 正常工作妨碍不大。EEPROM 有很好的擦写耐久性, 反复擦写 $10^4 \sim 10^5$ 次也不会损坏。在不受干扰情况下, 存储的数据至少 10 年有效。

图 5.19 所示为四种 EEPROM 存储芯片引脚图。

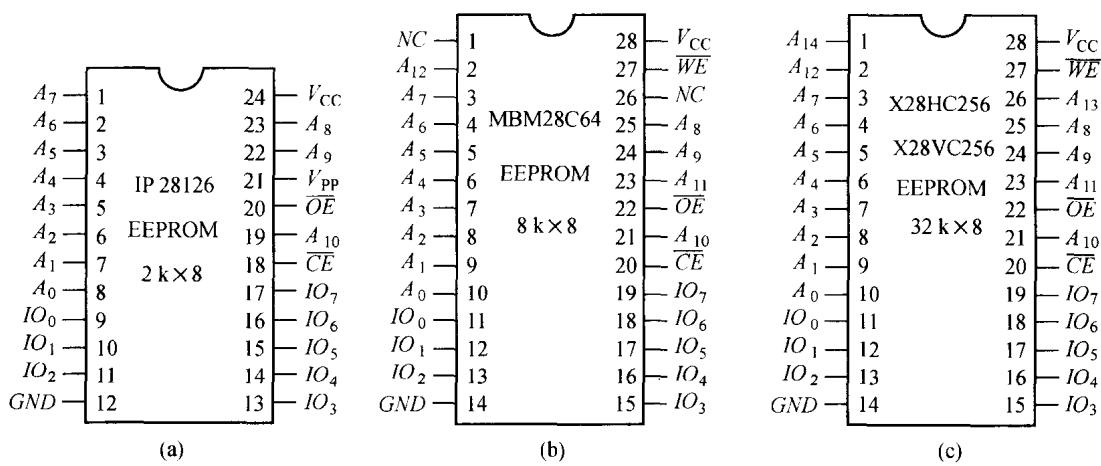


图 5.19 EEPROM 芯片引脚

$A_0 \sim A_{14}$ 、 \overline{CE} 、 NC 、 V_{CC} 、GND 引脚含义同 EPROM 芯片, $IO_0 \sim IO_7$ 为芯片的输入输出端, 只读时作输出端, 编程时作输入端。对于 IP2816 芯片, 专设置 V_{PP} 引脚, 由外界提供电压作编程和擦除使用。只读时, V_{PP} 同 V_{CC} 为 5 V; 编程和擦除时, V_{PP} 为 +20 V。MBM28C64、X28HC256、X28VC256 三芯片引脚分布类似于 SRAM。 \overline{WE} 为读、写(擦除、编程)控制端。对于 MBM28C64, \overline{OE} 由外界提供三值电平, 读出时为 TTL 低电平(0 ~ 0.8 V), 编程时为 TTL 高电平(+2 V ~ +5 V), 擦除时为 13.5 V。X28HC256、X28VC256 两者引脚相同, 都是单一电源+5 V 存储器件, 编程、擦除所需高压由片内电路自己产生, 区别是后者读出速度比前者更快, 可达 45 ns。近几年生产的 EEPROM 芯片, 一般都有数据轮询功能。该功能通过测试某个输入输出端(通常是 IO_7)电平, 判断最近一次写入是否完成, 以便确定能否读该次写入的信息。有的芯片如 IP2817A, 甚至增加一个 RDY/BUSY 引脚, CPU 通过该引脚电平即能确定最近一次写入是否完成, 可否进行读出操作。

EEPROM 不仅具有在线擦除和编程能力, 而且能实现单个存储单元的擦除与编程, 加之芯片封装不须留透明的光照擦除窗口等优点, 目前在计算机领域得到广泛应用。尽管 EEPROM 优点突出, 缺点亦很明显。FLOTOX 管 EEPROM 为双管存储位元, 位元尺寸比 SIMOS EPROM 大 10 倍, 集成度低。加之 EEPROM 编程前都需要电擦除然后编程, 所以功耗

较大。与同容量的 EPROM 比较,前者比后者成本和功耗都低得多,尤其使用 CMOS 工艺更是如此。从耐久性看,EPROM 也比 EEPROM 好。因此目前两种存储器件竞相推出,并行发展。就市场占有率看,EPROM 远大于 EEPROM,前者为后者的 5 倍左右。

5. 快擦写存储器 (Flash Memory)

(1) Flash 存储器概述

Flash 存储器 1983 年由 Intel 公司首先推出,商品化于 1988 年。Flash 存储器发展速度很快,集成度从最初的每片 64 kb 发展到现在的每片 64 Mb,并将推出每片 256 Mb 产品。读出速度也从 300 ns 提高到 45 ns。价格也不断降低,从 1991 年的 80 美元/Mb 降到目前的每 Mb 不足 0.5 美元。

Flash 存储技术是在 EPROM 和 EEPROM 的基础上发展起来的。它集中了两者的优点,既具有像 EPROM 一样的单管位元结构并沿用了传统 EPROM 的热电子隧道效应的编程机制,又具有 EEPROM 在线冷电子 F-N 隧道效应的擦除特点。位元尺寸与 EPROM 相当,比 EEPROM 小 10 倍,加之用低成本塑料封装,使 Flash 存储器既有 MROM 和 RAM 两者的性能,又有 MROM、DRAM 一样的高密度、低成本和小体积。它是目前惟一具有大容量、非挥发性、低价格、可在线改写和较高速度几个特性共存的存储器。Flash 存储器之所以被称为快闪存储器,是因为用电擦除且擦除的是整个存储矩阵或部分存储矩阵,通过公共源极或公共衬底加高压实现,速度很快,与 EEPROM 擦除一个存储单元的时间相同。擦除整个存储矩阵称为片擦除,擦除部分存储矩阵称为块擦除。早期 Flash 存储器都是片擦除结构,1991 年后的产品则同时具有两种擦除结构,可擦写次数为 10 万次。近几年产品又增加了很多新功能,可擦写次数也增到 100 万次,耐久性大大提高。

同 DRAM 比较,Flash 存储器存在两个缺点,即可擦写次数最高只有 10^6 次,最快取数时间 45 ns。而 DRAM 可读写次数可达 10^{15} 次,最快取数时间达 10 ns。所以从目前看,它还无望取代 DRAM。但它是一种理想的文件存储介质,当它代替硬盘时,除存取快、体积小、重量轻、功耗低和不易损坏外,还可就地执行 EIP(Execute In Place)。也就是说,用它存放操作系统、实用程序和数据文件时,开机后无需像使用硬盘那样,首先将操作系统、实用程序和数据文件调入主存储器,而是可以立即执行程序。它特别适用于在线编程的大容量、高密度存储领域。Flash 存储器在日益繁荣的膝上型、掌上型和笔记本型个人计算机中作为外存得到越来越广泛的应用。

(2) Flash 存储位元结构及存储信息原理

最简单的位元结构是与 EPROM 类似的单管叠栅结构。还有两种单管三栅结构,其中一种是在单管叠栅结构的基础上增加一个专用擦除栅 EG,另一种则是在单管叠栅基础上再增加一个 CG,它位于 FG 之下,故称为第二 CG。除单管结构外,还有双管结构、三极管和二极管结构的存储位元。采用多管结构的目的是改善 Flash 存储器的编程和擦除性能,克服单管叠栅过擦除问题和降低编程、擦除电压。这些结构都使位元变得复杂、尺寸加大、制造工艺

也变得繁琐,从而增加了芯片体积和价格。目前广泛使用的是单管叠栅结构存储位元。

单管叠栅位元依编程擦除方法不同,有多种结构。同 SIMOS 管一样,不论哪种结构,当 FG 上驻留电子,有附加电场存在,开启电压高($>5V$),表示存储“0”信息,当 FG 上无电子,开启电压低($<2V$),表示存储“1”信息。编程操作,就是使 FG 上充上电子,即写“0”。擦除操作,就是释放 FG 上的电子,即写“1”。读出则是在 CG 上加高开启电压与低开启电压的平均值,位元开启通导,读出信息“1”,否则读出信息“0”。

经典单管叠栅位元结构如图 5.20(a)所示,与 EPROM 的 SIMOS 管结构很相似,称这种结构为 ETOX(EPROM Tunnel Oxide) MOS 管结构。同 SIMOS 相比,不同之处在于 CG、FG 间氧化层厚度为 25 nm 左右,FG 与 P-Si 表面间的超薄隧道氧化层质量高,能经受多次高压冲击且厚度仅在 10 nm 左右,沟道长度(栅长)也很短,为亚微米级。如片容量 16 Mb 的 Flash 存储芯片,其位元 CG、FG 间氧化层厚度 20 nm,FG、P-Si 超薄隧道氧化层 10 nm,沟道长度 0.8 μm 。

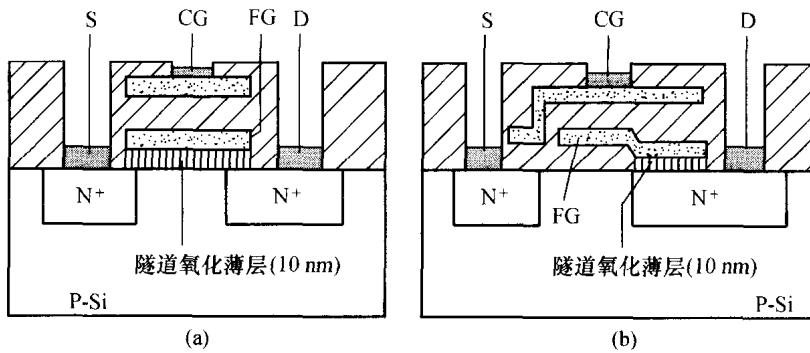


图 5.20 Flash 存储器单管叠栅位元结构

擦除:即写“1”,S 极加高电压($12 \sim 20V$),CG 接地,D 极接地或浮空,S 与 FG 极间发生 F-N 隧道效应。FG 上电子被拉回源区,开启电压变低(2 V 以下),成为存“1”位元。之所以源极擦除是因为存储矩阵或部分存储矩阵位元的源极都是连接在一起的,这样可实现整个芯片或分块快速擦除。如果存储位元矩阵或部分存储矩阵的源极不是连接在一起的,则高压不是加到源极,而是加到部分位元或全部位元的公共区衬底。这样也可以实现整个芯片或分块快速擦除。

编程:即写“0”,S 极接地,CG、D 极接高电压($12 \sim 20V$),发生热电子隧道效应,热电子穿过隧道氧化层进入 FG,开启电压变高(7 V),成为存“0”位元。

图 5.20(b)所示是另一种单管叠栅结构存储位元。其超薄隧道氧化层仅位于漏区上方,与 FLOTOX 管很相似,不同点在于超薄氧化层厚度仅为 10 nm 左右,FG 不跨越整个沟道,这种结构编程和擦除都是通过漏区超薄隧道氧化层的 F-N 隧道效应,加之 FG 不跨越整个沟道,使得编程和擦除效率都很高。

擦除: CG 接地, D 极接高压(12~20 V), S 极浮空。FG 电子通过 F-N 隧道效应返回到漏区, 附加电场消失, 变为低开启管, 写入“1”, 实现擦除。

编程: CG 加高压(12~20 V), D 极接地, S 极浮空。漏区电子通过 F-N 隧道效应进入 FG, 产生附加电场, 变为高开启管, 写入“0”, 实现编程。

读出: 通常 S、CG 和 D 极分别接地、+3 V 和 +5 V, 若不通导, 读出信息为“0”, 否则读出信息为“1”。

(3) Flash 存储芯片

Flash 存储芯片品种繁多, 按字位结构分有 m 字 \times 8 位、 m 字 \times 16 位和 m 字 \times 32 位三种形式。从最初单一的整片擦除型芯片发展为适合不同场合的芯片, 如适合于系统 BIOS 使用的启动区锁定的 Flash 存储芯片, 适合于文件系统的可分块擦写的 Flashfile 芯片和适合与 RAM 接口的 FlashRAM 芯片。Flash 存储芯片单地址编程时间为 μs 级, 整片编程时间为 s 级, 整片擦除时间为 s 级, 块擦除时间为 ms 级, 最快块擦除时间仅为 1 ms。Flash 存储芯片引脚除多一个编程电源引脚 V_{pp} 外, 同 SRAM。

单片 Flash 存储芯片可独自构成小容量快擦写存储器。多片 Flash 芯片组成存储卡, 构成大容量快擦写存储器, 这种存储器除多片 Flash 芯片互连外, 还需附加控制它们工作的智能控制器。

本节讲述了五种 ROM 的存储位元存储信息原理, 工作原理和存储芯片等问题。表 5.7 给出了五种 ROM 存储芯片的某些参数和性能的比较, 供读者参考。

表 5.7 非挥发性存储芯片比较

芯片种类	MROM	PROM	UV EPROM	EEPROM	Flash
最大集成度	32 Mb	512 kb	64 Mb	64 Mb	256 Mb
字位结构	$\times 4, \times 8, \times 16$	$\times 4, \times 8$	$\times 8, \times 16$	$\times 8, \times 16, \times 32$	$\times 8, \times 16, \times 32$
取数时间范围	25~400 ns	3~100 ns	15~300 ns	35~300 ns	45~300 ns
擦除方法	不能擦除	不能擦除	脱机紫外 线光照片擦除	在线电方法 单元擦除	在线电方法整片 或分块擦除
编程方法	掩模编程	用户编程器 按位元编程	用户编程器 按单元编程	在线按单元编程	在线按单元编程
功耗	20 mW~1 W	500 mW~1 W	100 mW~1.8 W	100 mW~250 mW	150 mW 左右
编程电压范围		5~7 V	12~25 V	10~30 V	6~20 V
应用领域	固定模式信息 如字符、汉字 库, 固定表格	控制存储器、 应用软件和 数据的固化	固定程序的存 储、半固定参数 和数据存储	同 UV EPROM	程序文件、 数据文件的存 储、半导体盘

5.3 按内容寻址存储器 CAM

5.3.1 概述

计算机应用中,常常需要从一批信息中查找具有一定特征的信息,而该信息在存储器中存在与否,有多少个,在哪个或哪些地址单元中存放却是未知的。比如一个对进入和离开机场空域飞机的机场监控系统,其内部的存储器存储着进出该机场所有飞机的型号、飞机的高度、速度、航向、载客量和载重量等信息。假设每架飞机的识别号码即为存储上述内容的地址,一般的查询应该首先给出地址——飞机识别号码,然后查询上述内容。而有时只知道上述全部或部分内容,却要查询飞机识别号码或其他部分内容,如已知飞行高度,需要查飞行速度、飞机识别号码和载客量,这就需要根据飞行高度这个内容,首先查到飞机识别号码——存储器地址,然后去读相应地址单元,得到飞行速度和载客量。若使用传统 RAM,则需要从存储器的首地址开始,按地址逐一读出上述内容,将读出信息的飞行高度字段与给出的飞行高度进行比较,如果相同,记录该地址,并根据该地址读存储器,得到飞行速度和载客量;否则继续读下一个地址单元,重复上述过程。若存在多重匹配情况,即使找到一次匹配,也应该重复上述过程,直到整个存储器查找完毕。因为上述操作是串行搜索,速度慢,效率低,因而在有些应用场合是不容许的。又比如在网络通信、人工智能和实时控制系统中,对信息的查找要求非常紧迫,速度缓慢将会造成整个系统瘫痪,甚至造成不可估量的损失。

为满足速度的要求,必须对传统 RAM 访存方式进行改进,使之能够依用户“最关心的内容”对存储器所有存储单元同时进行搜索,快速搜索与确定内容匹配的所有存储单元的地址,然后再依地址进行必要的操作。这个“最关心的内容”被称为“关键字”或“比较数”。通常指不是根据地址而是根据所存信息的全部或部分内容进行寻址的存储器为按内容寻址存储器 CAM,亦称为相联存储器 AM。CAM 除了具有和 RAM一样的随机读写和保持功能外,关键还具有比较功能,其基本组成如图 5.21 所示。

存储矩阵(Memory Matrix, MM):它是信息的驻在地,由 CAM 存储位元电路构成。

读出寄存器(Read Register, RR):亦称 CAM 缓冲寄存器,位数等于存储单元字长。暂时寄存由 MM 读出的信息,再通过数据总线 DB 送 CPU。

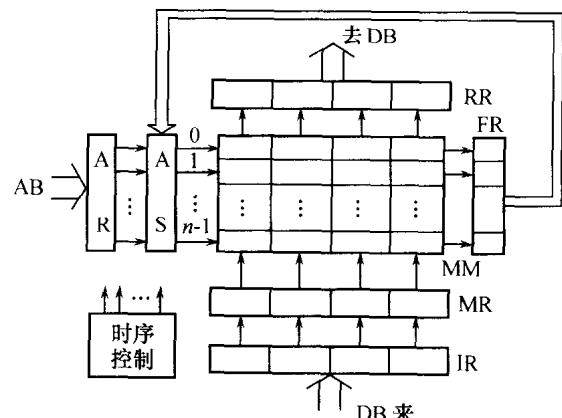


图 5.21 CAM 基本结构组成

输入寄存器 (Input Register, IR) : 亦称比较数寄存器, 位数同 RR。比较时存放关键字, 写入时存放要写入 MM 中的信息, 比较或写入信息时, 信息均来自数据总线 DB。

屏蔽寄存器 (Mask Register, MR) : 位数同 RR。MR 某位为“1”表示屏蔽相应位。比较时, 屏蔽是使相应位不参与比较, 即不论相应位是“1”还是“0”都不能影响比较结果; 写入时, 屏蔽是使原存信息的相应位保持不变, 只写入非屏蔽位。

标志寄存器 (Flag Register, FR) : 亦称状态寄存器, 每个单元至少有一位, 若 CAM 有 1 024 个单元, 则该寄存器至少有 1 024 位, 其位数总是存储单元个数的整数倍。它作为地址选择器 AS 的输入, 以便选择所需地址。

地址寄存器 (Address Register, AR) : 接收地址总线 AB 送来的地址, 相当于 RAM 的 MAR。提供按地址寻址时的地址, 其位数由 MM 的存储单元个数决定。

地址选择器 (Address Selector, AS) : 它是一个比较复杂的开关网络, 由译码器和编码器构成, 输入来自 AR 和 FR。若由 AR 提供地址, 译码器工作选择相应地址; 否则, 由 FR 控制编码器工作, 选择相应地址。

5.3.2 CAM 存储位元电路

CAM 存储位元电路具有存储和比较双重功能, 其电路图、逻辑框图和方框图如图 5.22 所示。

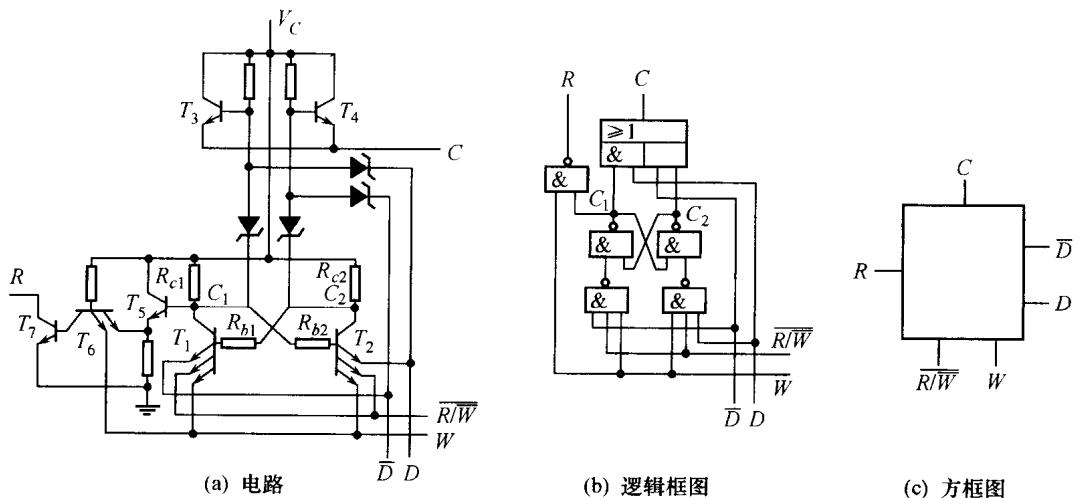


图 5.22 CAM 存储位元电路

其中 T_1 和 T_2 以及 $R_{c1}, R_{c2}, R_{b1}, R_{b2}$ 构成触发器, 用以存储信息。二极管和 T_3, T_4 组成比较器。 T_5, T_6, T_7 组成读出门, 受字线 W 控制, 当读出时, 它将存储位元的信息从 OC 门 T_7 的集电极输出。

写入时, 读/写控制 R/\bar{W} 加高电位, 欲写入的信息以互补形式加到数据线 D 和 \bar{D} 。如果

是写“1”，则线 D 上加高电位，线 \bar{D} 上加低电位，然后字驱动线 W 来正脉冲，使 T_1 通导， T_2 截止， C_1 点是低电位， C_2 点是高电位，即写入了“1”。类似地也可写入“0”。

读出时，读/写控制 R/\bar{W} 加低电位，字驱动线 W 来正脉冲，作为读出选通，使存储位元的信息 (C_1 点电位) 经读出门反相后输出。因读出门是开路集电极形式输出，因此可以线或。

比较时，读/写控制 R/\bar{W} 加低电位，要比较的信息以互补形式加到数据线 D 和 \bar{D} 上，通过比较器进行比较，如果相同，则比较器的符合输出 C 为低电位；如果不相同，则符合输出 C 为高电位。

保持时，只要字驱动线 W 或读 R/\bar{W} 处于低电位，存储位元电路就可长久保持信息，不会丢失。通常是 R/\bar{W} 处于低电位，因其连接所有存储位元，故使它们保持原信息不变。

5.3.3 CAM

1. 逻辑组成

采用 CAM 存储位元电路，构成 CAM 存储矩阵，在 CAM 矩阵基础上附加一些门电路构成 CAM。如图 5.23 所示，是 $4 \text{ 字} \times 4 \text{ 位}$ 线选法 CAM 逻辑简图，图中略去了所有的寄存器。

2. 工作原理

由 $\overline{W_0}, \overline{W_1}, \overline{W_2}, \overline{W_3}$ 四根字线对 CAM 进行字选择，读出时只能有一根线为低电位，即一个字被选。写入时，可以有多个字被选，即可有多根字线为低电位。 $D_0 \sim D_3$ 分别是 4 位数据的输入线，要向 CAM 写入的信息或要查寻的信息都加到数据输入线上。 $M_0 \sim M_3$ 分别是 4 位数据的屏蔽线， $M_i (i=0, 1, 2, 3)$ 为高时，则屏蔽相应的“写入”或“比较”位； M_i 为低时，则允许输入信息的相应位写入或参加比较。 R/\bar{W} 是读/写控制线，当 R/\bar{W} 为高电位时，执行“读”或“比较”；当 R/\bar{W} 为低电位时，执行“写”。 $C_0 \sim C_3$ 分别是字符输出。当 CAM 中字的内容与输入比较数相同时，则有符合信号输出。 $R_0 \sim R_3$ 分别是 4 位输出线，低电位代表“1”。CAM 有四种工作状态，简述如下：

(1) 写入：当 R/\bar{W} 为低电位时，可以向 CAM 写入。假如不屏蔽， $M_0 \sim M_3$ 都为低电位，允许数据 $D_0 \sim D_3$ 写入，在字驱动脉冲来时，假定是 $\overline{W_0}$ 被选，则将 4 位数据并行写入 $\overline{W_0}$ 地址单元。如果要屏蔽第 i 位，使其保留原来信息，则 M_i 加高电位，第 i 位数据被屏蔽，不允许写入，其他位允许写入，其余动作相同，不再重述。若欲将同一信息写入多个存储单元，则同时选中多个存储单元即可实现。

(2) 读出： R/\bar{W} 为高电位，可从 CAM 读取数据。在字驱动脉冲来时，打开读出门 T_6 。仍假定 $\overline{W_0}$ 被选，将 $\overline{W_0}$ 地址单元中 4 位数据并行读出，再经“线或”门输出到 R_i 。

(3) 比较： R/\bar{W} 为高电位、 $M_0 \sim M_3$ 都为低电位时，CAM 执行全位比较。要比较的信息加到数据线 $D_0 \sim D_3$ ，这时，它和 CAM 中的 4 个字的所有位同时进行比较，若第 i 字 4 位都相符时，该字给出符合信号， C_i 为高电位；如至少有 1 位不相符，则 C_i 为低电位，表示不符合。

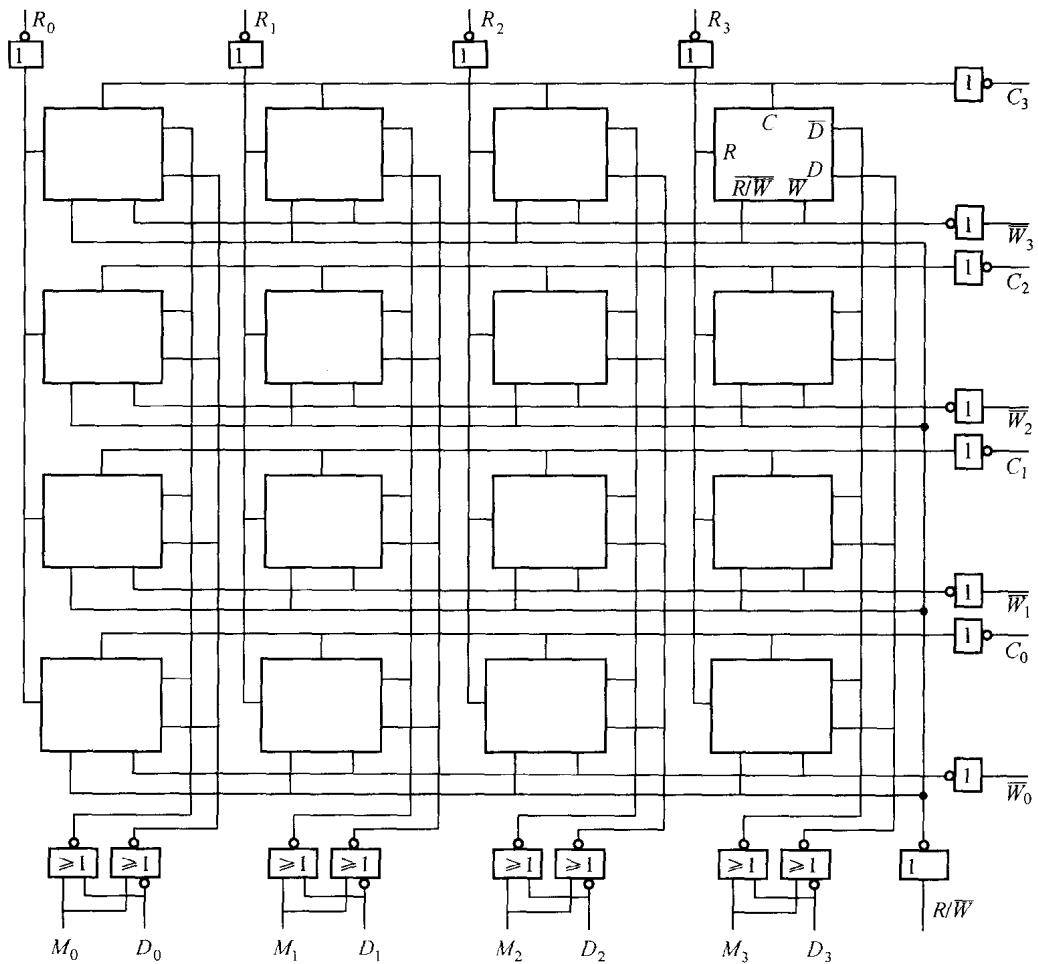


图 5.23 CAM 逻辑简图

若某位或某几位不参与比较，则将其对应位的 M 接上高电位，加以屏蔽即可。

(4) 保持：当 R/W 为高电位，或 \bar{W} 为高电位时，CAM 即处于保持信息状态。

3. CAM 特点

CAM 有很多优点，在计算机领域得到广泛应用。CAM 最突出的优点是可并行对存储器所有单元的所有位同时进行比较。利用这一优点，使 CAM 可以进行大于、小于、等于、是否处于给定的上下界范围、求最大值、最小值等各种类型的逻辑检索。利用 CAM 的屏蔽寄存器又可实现对存储器所有单元的一位或部分位同时进行比较，使 CAM 变得更加灵活方便。由于 CAM 存储位元、存储器结构都比较复杂，造价比较高，功耗也比较大。目前单片容量多在 16 Kb 左右，整个存储器容量很少有超过 256 KB 的。CAM 主要用于快速检索的场合，如 Cache 一主存层次地址映像和变换以及虚拟存储器地址映像和变换中使用的快表，人工智能

计算机和实时专家系统中都使用 CAM。在网络通信,尤其是网络之间的桥接通信,因为每个网络可能有成千上万个工作站点,要实现它们的通信,首先要搜索到站点使它们桥接起来,用普通 RAM,速度很难达到要求,目前大都使用 CAM。

5.4 内存储器的设计

在了解和掌握各种类型半导体存储芯片的工作原理和结构特点之后,还需要研究如何使用它们构成存储器的问题。因为存储芯片都是位片结构,如 $4\text{ K} \times 1\text{ 位/片}$ 、 $128\text{ K} \times 8\text{ 位/片}$ 、 $256\text{ K} \times 4\text{ 位/片}$ 等。而在一般应用中,存储器的实际容量比存储芯片本身容量要大得多,因此怎样利用存储芯片构造实际的存储器就是本节要研究和解决的问题。

5.4.1 设计内存储器的一般原则和方法

现代计算机的存储器,一般都是 ROM 和 RAM 并存,在一些中、大型机,甚至微型机中,为了解决速度、容量和价格的矛盾,出现多体和多层次的存储系统。特别在解决主存与 CPU 速度匹配问题时,又引进了高速缓存 Cache。从当前世界各国生产和研制的计算机来看,其主存和 Cache 都毫无例外地采用半导体存储器。下面仅就这两类存储器来讨论其设计的一般原则和方法。总的原则是根据使用要求,结合实际条件进行设计,使其具有良好的性能价格比。所谓使用要求,一是指设计的存储器是主存还是 Cache,是设计 RAM 还是 ROM,还是即有 ROM 又有 RAM;二是指设计的主要技术指标,如速度、容量、字长、结构以及环境条件。所谓结合实际条件,就是立足于现有的器材条件、技术条件和生产工艺条件。对于一个具体存储器的设计大致可分为三步进行,即系统设计、逻辑设计和工艺设计。

1. 系统设计

就是从计算机系统的角度,提出对存储器主要技术指标、功能以及结构形式等的要求,如存储容量、存储字长、取数时间、读写周期、总线宽度、控制方式、纠错错能力、环境温度以及可靠性要求等。另外还要确定存储器类型和外围电路形式,如 ECL、TTL、MOS,还要考虑体的划分及检测系统。

2. 逻辑设计

根据系统设计提出的要求进行逻辑设计,考虑逻辑电路的扇出和负载、信号的传输和衰减、逻辑级的延迟与匹配。在满足技术指标的条件下,要尽量简化逻辑,节省器材。

3. 工艺设计

依据系统设计和逻辑设计的资料,通过工艺设计落实到生产。工艺设计完成后,交出全部生产图纸和文件,包括元器件明细表、器件老化、测试、筛选、印制板、机器加工、电装、调试、考核等工艺条件和技术规范。

实际上三个设计阶段并不是互相孤立的,而是紧密相连并且互相影响的。如系统设计对逻辑设计提出了要求,但逻辑设计往往又会影响到系统的改变。另外逻辑设计与电路器件的关系更为密切,逻辑设计必须考虑电路的特性,并对电路有一定要求,而新电路的采用,又反过来会引起逻辑设计的改变。所以在整个设计过程中,常经过多次反复比较调整,不断修改,使设计逐渐完善。

5.4.2 内存储器的逻辑设计

1. 容量扩展

根据实际应用情况,容量扩展可能有以下三种形式:

(1) 位扩展

位扩展就是位数扩充,加大字长,以满足存储器字长的要求,而存储器的字数与存储芯片的字数相同。例如用 $mk \times n$ 位的存储芯片组成 $Mk \times N$ 位的存储器,需 $\lceil N/n \rceil$ (取上整数) 个 $mk \times n$ 位的存储芯片。令 $\lceil N/n \rceil$ 为 j ,其连接结构如图 5.24 所示。

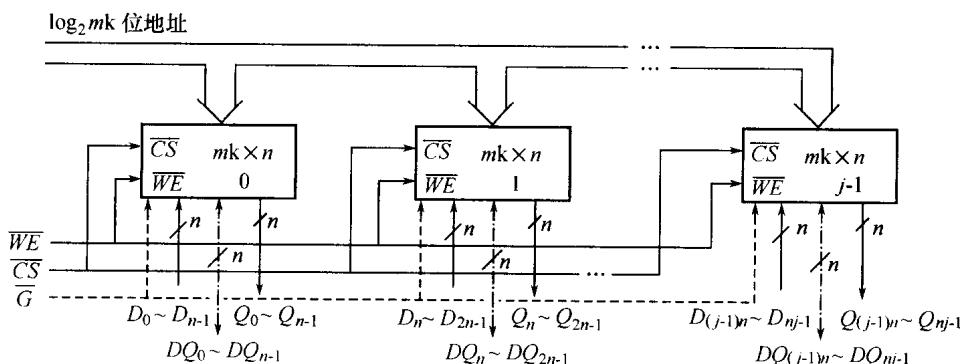


图 5.24 位扩展连接图

图中各存储芯片的地址、片选、写使能端和输出使能端(双向数据引脚控制端)都对应并接,数据输入、输出端则各自单独引出,即实现了位向扩展。图中 D_{nj-1}, Q_{nj-1} 即为字长的末位 D_{n-1}, Q_{n-1} ,而 $DQ_i (i=0, 1, \dots, nj-1)$ 为双向数据引脚的表示,其与 $D_i, Q_i (i=0, 1, \dots, nj-1)$ 单向引脚是互斥的,有前者即无后者,反之亦然。之所以这样画出,是考虑芯片数据引脚复用情况。图中点画线示出数据引脚复用时,数据引脚及其控制引脚的连接方法。

(2) 字扩展

字扩展是增加存储器的字数,即存储单元个数,而存储器的位数即每个存储单元的位数与存储芯片的位数相同。字扩展通常通过控制片选端来实现。例如用 $mk \times n$ 位的存储芯片组成容量为 $Mk \times n$ 位的存储器,则需要 $\lceil M/m \rceil$ 个 $mk \times n$ 位的存储芯片。令 $\lceil M/m \rceil$ 为 i ,其连接结构如图 5.25 所示。图中各芯片的地址、数据输入、数据输出、写使能端和输出使能端(双向数据引脚控制端)对应并接。片选单独引出,分别由存储器高 $\log_2 i$ 位地址译码输出。

控制,在某一时刻只有一个片选有效。存储器的低 $\log_2 m k$ 位地址直接与芯片地址连接。图中, $DQ_0 \sim DQ_{n-1}$ 的含义同位扩展。

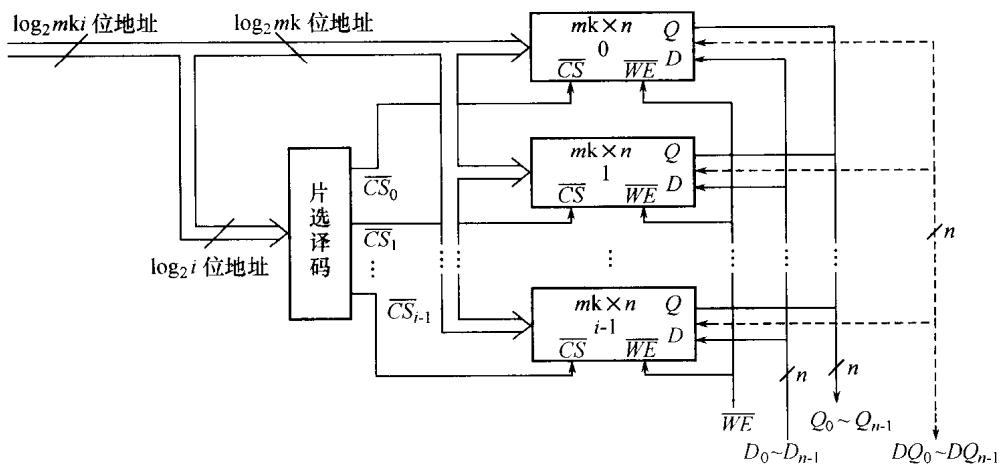


图 5.25 字扩展连接图

(3) 字和位同时扩展

大多数存储器都是在选定存储芯片的基础上通过字和位同时扩展而成的。了解了字和位的扩展方法之后,字、位同时扩展就不述自明了。假定存储芯片容量为 $m k \times n$ 位/片,设计的存储器容量为 $M k \times N$ 位(此处 M 是英文字母,不是“兆”的缩写)。共需 $\lceil M/m \rceil \lceil N/n \rceil$ 个 $m k \times n$ 位的存储芯片,连接结构如图 5.26 所示。图中所有芯片写使能端并接,所有芯片地址端对应并接直接连到存储器低 $\log_2 m k$ 位地址。同一行的片选端并接,行与行之间是独立的。令 $\lceil M/m \rceil$ 为 i ,则各行分别由存储器高 $\log_2 i$ 位地址译码的输出控制。输入、输出数据端同一列并接,列与列间是独立的。此图未考虑芯片数据引脚复用情况。

从纵向看,每列存储芯片给出不同存储单元的相同位。从横向看,每行存储芯片给出相同存储单元的不同位。

2. 驱动负载问题

上述三种扩展连接图只是一种示意形式,在实际设计时,还必须考虑负载和门的扇出能力问题。从图 5.26 可以看出,采用 $m k \times n$ 位芯片组成 $M k \times N$ 位存储器,存储芯片各端点,都需通过外围逻辑电路连接起来,这些芯片的端点就是外围逻辑电路的负载。由于逻辑电路的负载能力是有限的,所以在设计时,不能超过逻辑电路的负载能力。对于双极型存储芯片,负载为电流。对于 MOS 型存储芯片,负载为电容。存储芯片某种端点中的一个端点的负载量称为该种端点的负载因数,如 f_A ,表示地址端的负载因数,又如 f_{we} ,表示写使能端的负载因数。这样,驱动电路的负载,可用其所带的各种端点数表示,即负载因数的倍数表示。

- 地址驱动线的负载为: $L_A = \lceil M/m \rceil \lceil N/n \rceil f_A$

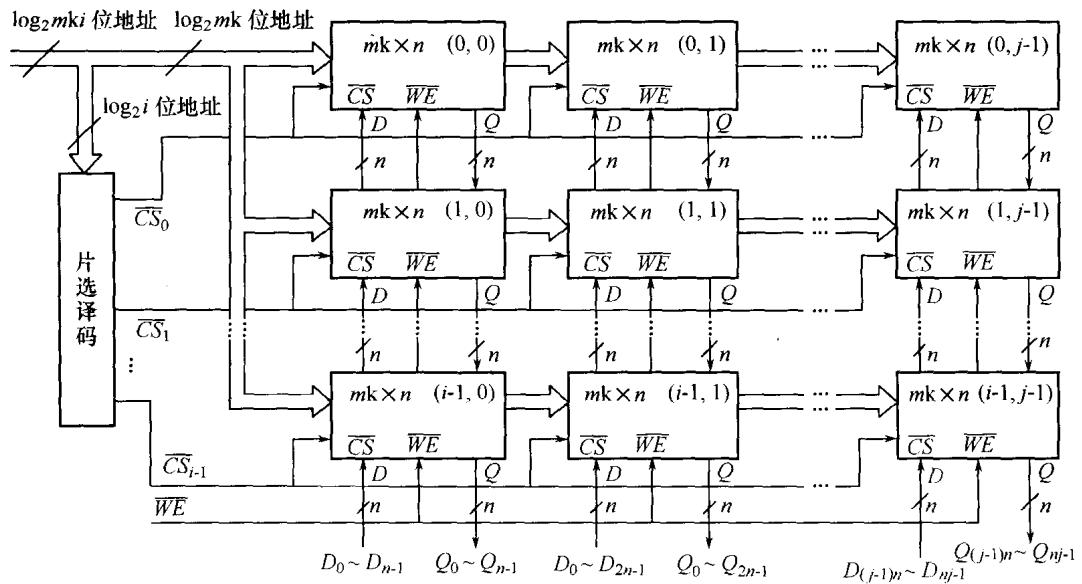


图 5.26 字、位同时扩展连接图

- 读/写驱动线的负载为: $L_{WE} = \lceil M/m \rceil \lceil N/n \rceil f_{WE}$
- 行片选驱动线的负载为: $L_{CS} = \lceil N/n \rceil f_{CS}$
- 数据输入线的负载为: $L_{DI} = \lceil M/m \rceil f_{DI}$
- 数据输出线的负载为: $L_{DO} = \lceil M/m \rceil f_{DO}$

当存储器的容量 $Mk \times N$ 位很大时, 外围逻辑电路的负载就会加重, 以致超出其驱动能力。这时, 就要采用“分载”的办法, 用两个或数个门并联使用, 分别驱动, 来扩大驱动能力。如果设计的存储器速度要求不高, 而且是 MOS 存储器件, 即都为电容负载时, 则驱动负载量可以放宽, 究竟带多少合适, 要视具体情况而定。

3. 存储器的系统存储周期和系统取数时间

这是存储器的两项重要指标, 逻辑设计是否满足系统要求, 要进行估算, 不满足时还要修改设计。由于存储器是由存储芯片和外围逻辑电路构成的, 信息传递经过外围电路要产生延时, 因此存储器的系统存储周期和系统取数时间, 要分别大于存储芯片的读写周期和取数时间, 如图 5.27 所示。

(1) 系统取数时间 T_{SA}

① 当读写控制端 \overline{WE} 和片选 \overline{CS} 先于地址 A 到达存储芯片时, 则地址取数时间为:

$$t_{SAA} = t_{AD} + t_{AA}$$

其中 t_{AD} 为地址缓冲器延时, t_{AA} 为存储芯片的地址取数时间。

② 当地址 A 和读/写控制端 \overline{WE} 先于片选 \overline{CS} 到达存储芯片时, 则片选取数时间为:

$$t_{SCSA} = t_{ADC} + t_{ACS}$$

其中 t_{ADC} 为片选择码延时, t_{ACS} 为存储芯片的片选取数时间。在实际中, 两种取数时间 t_{SAA} 和 t_{SCSA} 并非共存, 所以要取两者中大者为系统取数时间。如果输出还有缓冲器, 还应加上输出缓冲延时 t_{BD} 。所以系统取数时间 T_{SA} 等于 $\max(t_{SAA}, t_{SCSA}) + t_{BD}$ 。

对于 DRAM 存储芯片, 通常是用 RAS 作字容量扩展, 即作片选。有的 DRAM 存储芯片甚至都没有片选端, 即使有片选端, 一般也是接地。因此, 用 DRAM 存储芯片构成存储器时, 系统取数时间 T_{SA} 等于 $\max(t_{SAA}, t_{RASA}, t_{CASA}) + t_{BD}$ 。其中 t_{RASA}, t_{CASA} 分别为行选通取数时间、列选通取数时间。

(2) 系统存储周期 T_{SM}

系统存储周期 T_{SM} 由下式求得:

$$T_{SM} = T_M + t_D + t_R$$

其中 T_M 为芯片存储器周期, 一般从芯片数据手册可以查到。 t_D 为系统传输时延, 由存储芯片外围电路的逻辑级数确定。 t_R 为系统恢复时间, 通过系统测试可以得到。关于系统取数时间和系统存储周期的计算, 将在存储器设计举例中作具体说明。

5.4.3 内存储器逻辑设计举例

了解了内存储器设计的一般原则与方法, 本节首先分别以 BiRAM 芯片构造 Cache 和以 DRAM 芯片构造主存为例, 较为详细地阐述内存储器逻辑设计的步骤与方法。然后简介存储模块, 主存与系统连接时所需的控制信号及其时序控制线路。实例中使用的芯片, 其集成度及有关性能虽然不能代表当前半导体存储芯片的最高水平, 但是其逻辑设计的思想方法却不失一般性与示范性。

1. ECL 高速缓存 Cache 逻辑设计

用 BiRAM 存储芯片 MCM10146 和 ECL 外围电路构成 $4\text{ K} \times 64$ 位高速缓存 Cache。芯片逻辑框图如图 5.28 所示。

MCM10146 是 1024×1 位 BiRAM, 由 10 位地址 ($A_0 \sim A_9$) 来进行地址选择。器件上有一个数据输入端 D_{in} , 一个数据输出端 D_{out} , 一个片选端 CS 。当 CS 处于低电位时, 存储芯片

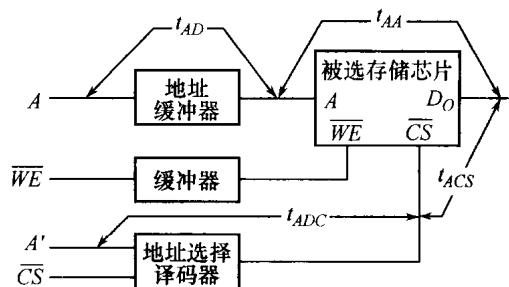


图 5.27 存储器系统取数时间示意图

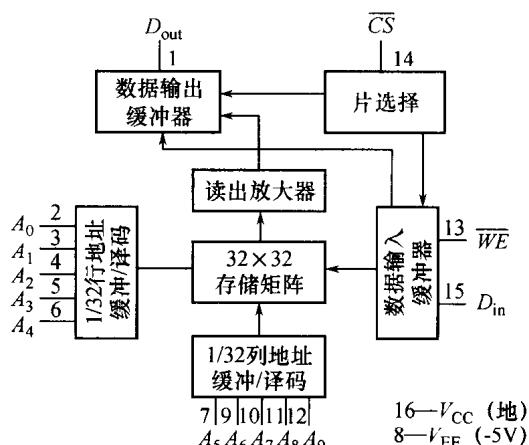


图 5.28 MCM10146 逻辑框图

可以进行读写操作。芯片上还有一读/写控制端 WE , 用来控制存储器的工作方式。当 WE 为低电位时, 表示执行写操作, 可把数据输入端 D_{in} 的数据写入地址码所指定的存储位元, 同时数据输出端为低电位。当 WE 为高电位时, 表示执行读操作, 这时, 被选存储位元的数据可以送到数据输出端 D_{out} 。芯片的输入输出均与 ECL 相匹配, 其主要性能为: 功耗 $P = 700 \text{ mW}$; 存储周期 $T_M = 45 \text{ ns}$; 地址取数时间 $t_{AA} = 29 \text{ ns}$; 片选取数时间 $t_{ACS} = 7 \text{ ns}$ 。

MCM10146 存储芯片逻辑功能如表 5.8 所示。表中 L 表示低电平, H 表示高电平, Q 表示 D_{out} 的输出, Φ 表示随机状态, 或为 0, 或为 1。

表 5.8 MCM10146 功能

工作方式	输入			输出 D_{out}
	\overline{CS}	\overline{WE}	D_{in}	
写“0”	L	L	L	L
写“1”	L	L	H	L
读	L	H	Φ	Q
禁止	H	Φ	Φ	L

4 K × 64 位 Cache 的逻辑设计的过程如下;

(1) 确定存储芯片数量

由于给定存储芯片的字位结构为 1 K × 1 位, 其容量远小于要求的容量 4 K × 64 位, 因此要采用字位同时扩展的办法构成存储矩阵。所需存储芯片数量为:

$$\lceil M/m \rceil \lceil N/n \rceil = \lceil 4/1 \rceil \lceil 64/1 \rceil = 256 \text{ 片}$$

按照逻辑关系可画出存储矩阵, 如图 5.29 所示。

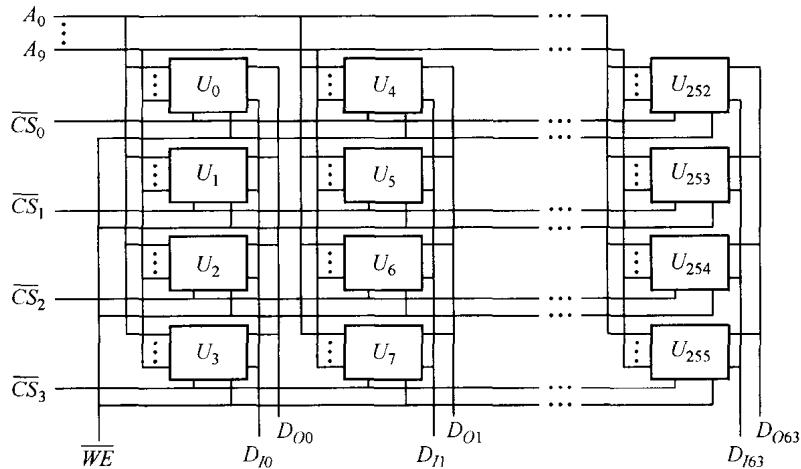


图 5.29 4 k × 64 位存储矩阵

(2) 负载计算

假定不论是什么负载端,每个逻辑门都只能驱动 8 个负载端。

- 每位地址驱动需要的分载门数 N_A 。 $L_A/8f_A = \lceil M/m \rceil \lceil N/n \rceil f_A/8 f_A = 256/8 = 32$; 因为所需门数超过 8,故应当再加 1 级门驱动,即 $32/8 = 4$,所以 $N_A = 32 + 4 = 36$ 。

- 读写控制驱动所需的分载门数 N_{WE} 同 N_A 。
- 数据输入所需的分载门数 N_{DI} 。因为 $L_{DI}/8f_{DI} = \lceil M/m \rceil f_{DI}/8f_{DI} = 4/8 = 0.5$, 故取 $N_{DI} = 0$ 。注意:当分载门数不大于 1 时,取分载门数为零,即不需分载,可以直接驱动。
- 数据输出是采用“线或”连接,它也按 8 个负载端考虑,取 $N_{DO} = 0$ 。
- 片选驱动需要分载门数 N_{CS} 。由于片选是分四行驱动的,设每行需分载门为 N_{CSI} 则有: $N_{CSI} = L_{CS}/8f_{CS} = \lceil N/n \rceil f_{CS}/8f_{CS} = 64/8 = 8$ 。所以,所有行片选所需门数为: $N_{CS} = 4N_{CSI} = 4 \times 8 = 32$ 。

存储器地址需 12 位 $A_0 \sim A_{11}$,根据以上分析,其中的低 10 位 $A_0 \sim A_9$ 应当用两级门分载驱动, A_{10}, A_{11} 译码在片选信号控制下生成各行的片选。读/写控制应当用两级门分载驱动。其驱动连接如图 5.30 所示。

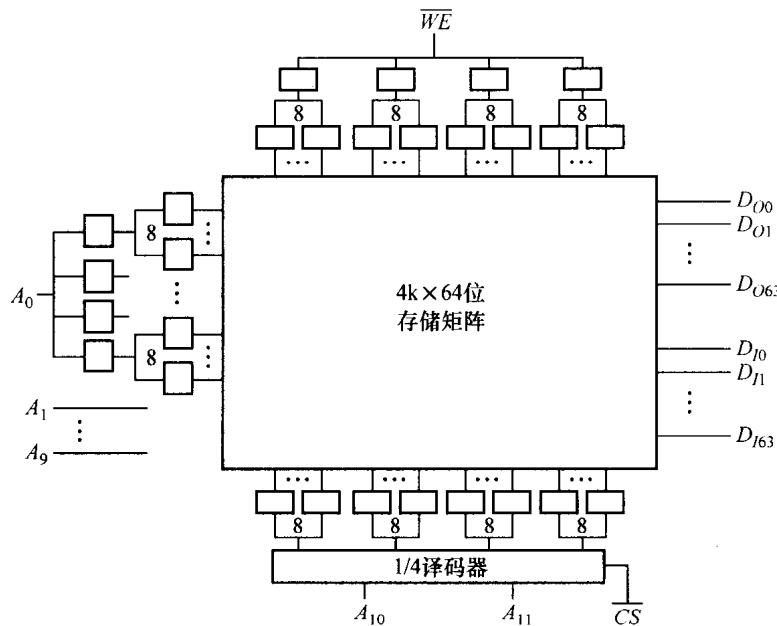


图 5.30 4 k × 64 位 Cache 驱动连接

(3) 速度估算

从地址输入算起,假定每级门延迟为 2.5 ns、忽略走线延时、系统恢复时间 t_R 为 2 ns,则存储器系统取数时间 T_{SA} 和系统存储周期 T_{SM} 为:

$$T_{SA} = t_{AA} + t_{AD} = 29 + 2 \times 2.5 = 34 \text{ ns}$$

$$T_{SM} = T_M + t_D + t_R = 45 + 2 \times 2.5 + 2 = 52 \text{ ns}$$

2. DRAM 主存设计

用 DRAM 存储芯片构成主存储器, DRAM 存储芯片特性在 5.2.3 节已经介绍, 本节讨论如何用 MCM511000A DRAM 存储芯片构成 $4 \text{ M} \times 32$ 位的主存储。

(1) MCM511000A 存储芯片简介

MCM511000A 是 Motorola 公司生产的 $1 \text{ M} \times 1$ 位的 DRAM 存储芯片, 它采用 $1.0 \mu\text{m}$ CMOS 工艺, 封装有 DIP、SOJ 和 ZIP 三种形式。如图 5.31 和表 5.9 所示, 为 DIP 封装及引脚功能说明。该芯片的输入输出电平与 TTL 兼容, 三态输出, 引脚 TF 测试时加 $+10 \text{ V}$ 高电压, 正常工作时接低电平或开路。

表 5.9 MCM511000A 引脚功能

引脚	名称/功能
$A_0 \sim A_9$	地址输入
D, Q	数据输入、数据输出
\bar{W}	读写控制即写使能端
RAS, CAS	行、列地址选通
TF	测试功能许可
V_{cc}	电源 $+5 \text{ V}$
V_{ss}	接地

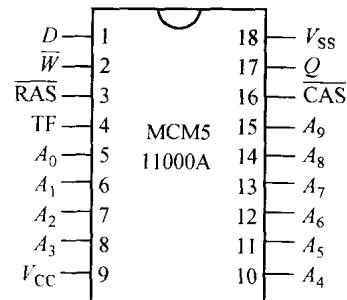


图 5.31 MCM511000A 引脚图

MCM511000A 是高速 DRAM 芯片, 虽然存储容量不大, 却具有现代芯片多种存储器操作。它包括随机读、随机写、页面读、页面写、随机先读后写和页面先读后写等存取操作, 为用户提供了可选择使用的灵活性。同时也导致芯片逻辑的复杂和性能参数的繁多, 如表 5.10 所示, 为该芯片主要性能参数。

图 5.32 为 MCM511000A 逻辑示意图。行、列地址锁存器用于锁存分时打入的行、列地址。数据输入和输出缓冲用于暂存要写入的或已读出的数据。刷新控制器和刷新计数器为实现刷新提供硬件支持。关于刷新, 5.4.4 节将专门介绍。四个 512×512 存储矩阵构成 1 M 个存储位元的存储体。每个存储矩阵都由相同的 9 位行地址码和 9 位列地址码选择。9 位行地址码经译码译出 512 条地址驱动线, 分别选择 512 行; 9 位列地址码译出 512 条地址驱动线, 分别选择 512 列。当行地址锁存器将低 9 位行地址加到四个存储矩阵上时, 每个存储矩阵都被选中一行。当列地址锁存器将低 9 位列地址加到四个存储矩阵上时, 每个存储矩阵又被选中一列。和被选中的行相与, 选中四个存储位元, 在 $1/4$ 输入输出控制门的作用下选中存储体的一个存储位元。注意, $1/4$ 输入输出控制门受控于行、列地址锁存器的最高位和列地址选通 CAS 。

表 5.10 MCM511000A 主要性能参数

参数	符号	MCM511000A - 70		MCM511000A - 80		MCM511000A - 10	
		最小	最大	最小	最大	最小	最大
随机读或随写周期	t_{RC}	130 ns		150 ns		180 ns	
先读后写周期	t_{RWC}	155 ns		175 ns		210 ns	
页读或页写周期	t_{PC}	40 ns		45 ns		50 ns	
页先读后写周期	t_{PRWC}	65 ns		70 ns		80 ns	
从 RAS, 起算的取数时间	t_{RAC}		70 ns		80 ns		100 ns
从 CAS 起算的取数时间	t_{CAC}		20 ns		20 ns		25 ns
从列地址起算的取数时间	t_{AA}		35 ns		40 ns		50 ns
刷新间隔时间	t_{RFSH}		8 ms		8 ms		8 ms
工作功耗	P_{DA}		600 mW		550 mW		500 mW
待机(维持)功耗	P_{DS}		5 mW		5 mW		5 mW
数据输入建立时间	t_{DS}	0		0		0	
数据输入维持时间	t_{DH}	15 ns		15 ns		20 ns	
RAS, 预充电恢复时间	t_{RP}	50 ns		60 ns		70 ns	
页方式 RAS, 脉冲宽度	t_{RASP}	70 ns	100 μ s	80 ns	100 μ s	100 ns	100 μ s
随机方式 RAS, 脉宽	t_{RAS}	70 ns	10 μ s	80 ns	10 μ s	100 ns	10 μ s
$A_0 \sim A_9$ 、D 输入电容	C_{in1}		5PF		5PF		5PF
TF, RAS, CAS, \bar{W} 输入电容	C_{in2}		7PF		7PF		7PF
Q 输出电容	C_{out}		7PF		7PF		7PF

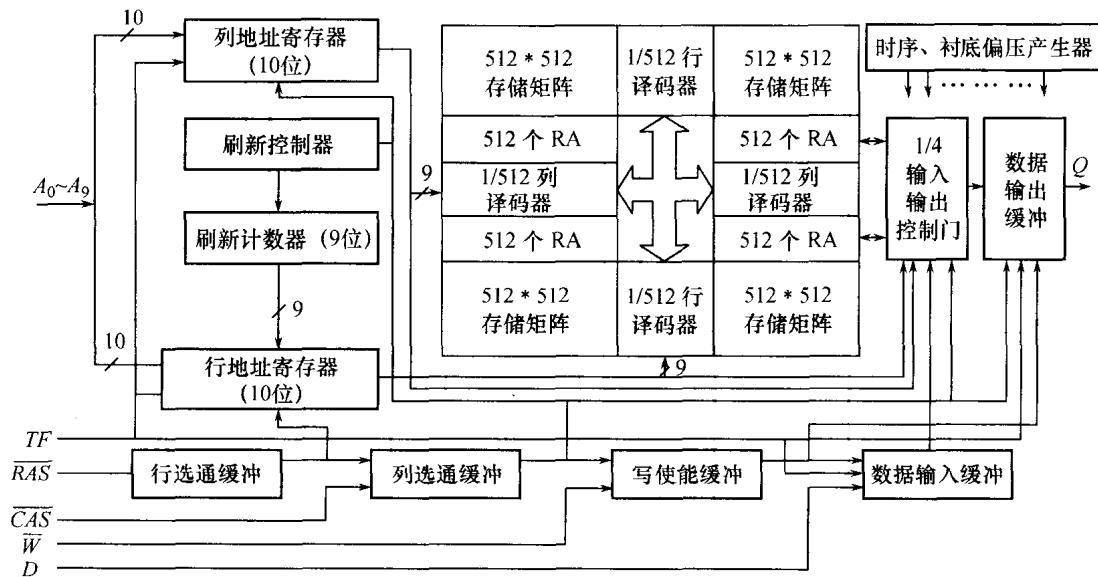


图 5.32 MCM511000A DRAM 逻辑框图

是读出还是写入数据由 \bar{W} 控制, 当 \bar{W} 为高电平时, 选中的四个存储位元中的信息经鉴别和重写后在 CAS 、 \bar{W} 控制下, 经 $1/4$ 输入输出控制门选中一个存储位元信息通过数据输出缓冲在 Q 引脚读出。当 \bar{W} 为低电平时, D 引脚上要写入的信息, 经数据输入缓冲、 $1/4$ 输入输出控制门写入到指定存储位元。

行、列选通及写使能缓冲与时序、衬底偏压发生器提供芯片内部的时序和控制电平, 使芯片平常工作。

(2) $4 M \times 32$ 位 DRAM 主存储器逻辑设计

① 确定存储芯片数量

采用 MCM511000A - 70 存储芯片组成 $4 M \times 32$ 位 DRAM 主存储器, 需要进行字向和位向同时扩展, 共需存储芯片数为:

$$\lceil M/m \rceil \lceil N/n \rceil = \lceil 4/1 \rceil \lceil 32/1 \rceil = 128 \text{ 片}$$

存储芯片各引脚连接见图 5.33。由图可知, 所有存储芯片相应地址端并联; 同一位的数据输入 D 、数据输出 Q 分别并联; 所有存储芯片的 \bar{W} 、 CAS 分别并联; RAS 作为字向扩展控制

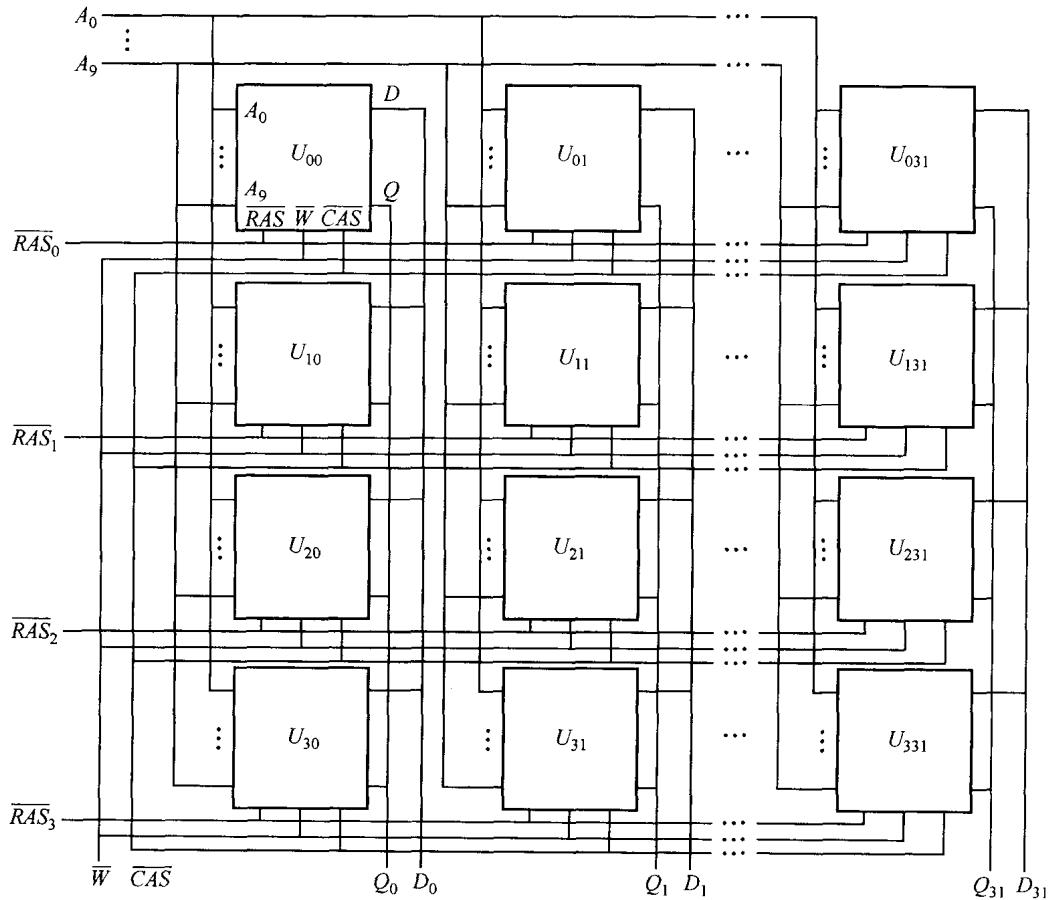


图 5.33 $4 M \times 32$ 位 DRAM 存储矩阵示意

端,受最高两位地址译码控制,所以一个 \overline{RAS} 分为四个 \overline{RAS}_i ($i = 0, 1, 2, 3$), 每个连接 32 个存储芯片。

② 负载计算

存储芯片各引脚都是由外围电路驱动的,由于各引脚负载不同,连接方式也不同,所以在进行逻辑设计时,必须考虑负载问题。对于 MOS 存储芯片,其负载主要是电容(见表 5.10)。

电容越大,负载越重,即要获得的速度、电容越大,则需要的驱动力越强。根据这一思路,应当计算各种引脚负载以及要分载驱动的门数。假定外围电路的一个门可驱动电容负载的能力为 40 PF。

- 地址驱动:按逻辑要求,每根地址线要接到所有存储芯片相应地址引脚,即每根地址线要驱动 128 个地址端。因为每个地址端负载为 5 PF,所以一个门只能驱动 8 个地址端,128 个地址端需 16 个门。通常地址线也都是由触发器或门电路输出的,它不能直接驱动 16 个门电路,还要加 1 级需两个门电路接力。所以每根地址线都需 $16 + 2$ 即 18 个门驱动。

- \overline{W} 驱动:和地址线一样, \overline{W} 驱动也需接 128 个 \overline{W} 端。但每个 \overline{W} 端载为 7 PF,所以一个门只能驱动 $[40/7]$ 即 5 个 \overline{W} 端,需要 $[128/5]$ 即 26 个门。此时还要加 1 级需要 $[26/8]$ 即 4 个门进行接力,因此, \overline{W} 驱动需 $26 + 4$ 即 30 个门。

- \overline{CAS} 驱动:和 \overline{W} 驱动一样需 30 个门。

- \overline{RAS} 驱动:分 4 行驱动,每行 32 个 \overline{RAS} 端。每个 \overline{RAS} 端负载为 7 PF,一个门只能驱动 5 个 \overline{RAS} 端,每行需 $[32/5]$ 即 7 个门。4 行共需 28 个门。

- D 和 Q 驱动:数据输入和数据输出负载较轻,同一位的对应端并联后直接连接,不必进行分载。

③ 速度估算

进行存储器的逻辑设计时,一定要考虑速度是否满足设计要求,速度是存储器的一项重要指标,包括取数时间和存储周期。通常对这一指标要求很苛刻,如不满足要求,则要修改设计,直到符合要求为止。

由负载计算可知,地址驱动、 \overline{W} 和 \overline{CAS} 驱动均需经两级门才到达芯片引脚。 \overline{RAS} 驱动分 4 行,虽只需一级门,但其要受控于存储器最高两位地址 $A_{21} \sim A_{20}$ ($A_0 \sim A_9$ 作为芯片的列地址, $A_{10} \sim A_{19}$ 作为芯片的行地址) 的译码,实际上也是经两级门才到达芯片引脚。假定每级门延迟 2.5 ns,系统恢复时间 t_R 为 2 ns,忽略走线延时。依表 5.10 速度估算如下:

$$\text{系统随机取数时间 } T_{RA}: T_{RA} = t_{RAC} + t_{AD} = 70 + 2 \times 2.5 = 70 \text{ ns}$$

$$\text{系统页式取数时间 } T_{PA}: T_{PA} = t_{AA} + t_{AD} = 35 + 2 \times 2.5 = 40 \text{ ns}$$

$$\text{系统随机读或随机写存储周期 } T_{RM}: T_{RM} = t_{RC} + t_D + t_R = 130 + 2 \times 2.5 + 2 = 137 \text{ ns}$$

$$\text{系统随机先读后写存储周期 } T_{RWM}: T_{RWM} = t_{RWC} + t_D + t_R = 155 + 2 \times 2.5 + 2 = 162 \text{ ns}$$

$$\text{系统页式读或写存储周期 } T_{PM}: T_{PM} = t_{PC} + t_D + t_R = 40 + 2 \times 2.5 + 2 = 47 \text{ ns}$$

$$\text{系统页式先读后写周期 } T_{PWM}: T_{PWM} = t_{PRWC} + t_D + t_R = 65 + 2 \times 2.5 + 2 = 72 \text{ ns}$$

3. 存储模块

存储模块(Memory Modules)亦称存储器组件,是近几年出现的存储器构件。它是以存储芯片为基本器件,通过位扩展、字扩展或字位同时扩展构成存储器的半成品。一个存储体(Bank)由一个或多个存储模块构成,一个或多个存储体组成一个大容量的存储器,由存储控制芯片控制管理。借助插件、插槽、插座和开关实现存储器的扩充或压缩。从而使存储器扩充容易,安装方便,容量可大可小。在中、小和微型计算机系统中,存储模块应用非常广泛,在大型机中也有采用的趋势。

存储模块实际上是把若干个存储芯片,按照一定的逻辑关系连接起来,高密度地安装在对外有若干个引脚的印刷电路板上或密封在对外有若干引线的陶瓷或塑料管壳中。这样,存储模块作为一个独立的不可分割的整体存在。存储模块对外的引脚称为“线”,一个模块对外有多少个引脚,则称其为多少“线”的存储模块。存储模块的读写速度和存储芯片的读写速度基本相同。芯片重叠安放,呈立方体的存储模块称为三维存储模块,俗称芯片堆。芯片在同一平面安放,厚度很薄,故称其为二维存储模块,俗称内存条。下面主要介绍内存条。

目前,既有DRAM内存条,也有SRAM内存条。我们仅介绍广泛使用的DRAM内存条。DRAM内存条分为两类,一类是单列直插存储模块(Single Inline Memory Modules,SIMM),另一类是双列直插存储模块(Dual Inline Memory Modules,DIMM)。

(1) 单列直插存储模块 SIMM

SIMM有30线和72线两种。30线为单字节型模块,即字位结构为 $4\text{ M} \times 8$ 位、 $4\text{ M} \times 9$ 位、 $1\text{ M} \times 8$ 位、 $1\text{ M} \times 9$ 位和 $256\text{ K} \times 8$ 位等。72线为多字节型模块,即字位结构为 $8\text{ M} \times 36$ 位、 $8\text{ M} \times 32$ 位、 $4\text{ M} \times 40$ 位、 $1\text{ M} \times 36$ 位、 $2\text{ M} \times 32$ 位、 $4\text{ M} \times 32$ 位等。字位结构中的位数,若不是8的整数倍,其多余位数是为提高存储器可靠性和冗余容错设置的。9位结构的第9位用于奇偶校验,其他位结构的多余位数用于纠错和检错。目前SIMM因存储容量小已不再生产和使用。

(2) 双列直插存储模块 DIMM

DIMM的DRAM模块近几年才问世,目前主要有四种规格。第一种是72线小尺寸DIMM,其字位结构为 $1\text{ M} \times 32$ 位、 $2\text{ M} \times 32$ 位、 $4\text{ M} \times 32$ 位等,目前也很少生产和使用;第二种是字长为8字节以上的168线DIMM,字位结构为 $1\text{ M} \times 64$ 位(8MB)、 $4\text{ M} \times 64$ 位(32MB)、 $4\text{ M} \times 72$ 位、 $4\text{ M} \times 80$ 位、 $8\text{ M} \times 64$ 位(64MB)、 $16\text{ M} \times 64$ 位(128MB)等,采用的是EDO DRAM或SDRAM存储芯片;第三种是为适应计算机系统高速、大容量要求,采用DDR DRAM存储芯片的184线DIMM,字位结构为 $16\text{ M} \times 64$ 位(128MB)、 $32\text{ M} \times 64$ 位(256MB)、 $64\text{ M} \times 64$ 位(512MB)等;最近为适应工作站和大型计算机系统又推出200线DIMM内存条,分缓冲型和非缓冲型两种。存储容量很大,速度也很快,采用SDRAM、DDR DRAM存储芯片组成,工作时钟都在83MHz以上,字位结构中的位数都采用72位或80位。

目前,EDO DRAM内存条因速度慢已退出市场,SDRAM内存条也将逐渐退出市场。DDR DRAM内存条取数时间通常为7ns,有的甚至更高,已为市场的主流产品。

4. 主存与系统连接时所需的控制信号及其时序控制线路

主存是计算机硬件系统的五大组成部件之一。下面以总线互联结构为例简述 CPU 或输入输出设备访问主存的几个控制信号。本书第七章将较详细的介绍现代计算机广泛采用的总线互联结构。CPU 或输入输出设备要访问主存, 需先发出访存请求信号 $MREQ$ 给控制总线, 主存的时序控制线路接收控制总线的 $MREQ$ 信号后, 若存储器的闲置, 则经控制总线向请求访问主存的 CPU 或输入输出设备发回允许访问主存的信号 $MGNT$ 。CPU 或输入输出设备接收到 $MGNT$ 后, 又经控制总线向主存的时序控制线路发出读写控制信号 R/\bar{W} , 注意, 有的总线读写控制信号被分成存储器写 $MEMW$ 和存储器读 $MEMR$ 两个信号。与此同时, 地址总线向主存送出读写所需的地址, 若是写信息, 数据总线还需送出要写的数据。此后的读写操作, 则由主存时序控制线路控制主存的其他部件实现。

对于 BiRAM 和 SRAM, 工作时仅需上述控制信号, 通常由系统总线控制器即可提供。对于 DRAM, 要使存储器正确无误地执行读、写和刷新操作, 必须产生 RAS 、 CAS 、 \bar{W} 、刷新请求 $REFREQ$ 、刷新允许 $REFGNT$ 等一系列时序控制信号, 以控制存储器有条不紊地工作。这一系列时序信号可自行设计产生, 如用环形移位寄存器、定时单稳、定时延迟线电路、微分积分电路等。也可选用专用芯片, 如 Intel 公司的 8203 DRAM 系统控制器和 WACOM 公司的 W4006AF DRAM 系统控制器, 有关细节可查阅相应公司的数据手册。

5.4.4 DRAM 刷新

DRAM 单管存储位元是靠电容上的电荷存储效应记忆信息的。尽管 MOS 管截止时输入阻抗高达 $10^{14}\Omega$, 但电容上电荷的泄漏仍然是不可避免的。随着时间的推移, 电荷的泄漏会丢掉存储的信息“1”。为了保证所存信息的正确性, 可用充电的方法及时地使电容上电荷恢复到泄漏前的状态。通常把按一定的时间间隔为记忆电容充电的过程称之为 DRAM 刷新。

从 DRAM 存储芯片本身的结构来看, 其存储矩阵均为重合法三维存储矩阵, 对任意存储位元读写时, 该位元所在行的其余存储位元也都得到了刷新。由于读写操作是随机的, 不能保证所有位元在规定的时间间隔内都有这种充电的机会, 所以必须设置专门的刷新机构。但读写操作给刷新操作以提供如下启示, 即只要周而复始选取各行, 就能对整个存储芯片进行刷新, 且与选行的顺序无关。要指出的是存储器的刷新和存储芯片的刷新是一致的, 因为存储器的刷新是对所有存储芯片同时进行。

1. DRAM 刷新的有关参数

(1) 信息保持时间 T_{ref}

从信息以电荷形式存入电容起, 到电容电荷经过一段时间泄漏, 读放仍能鉴别出原存信息止的这一段时间称为信息保持时间。若再继续延长时间, 读放就不能有效地鉴别所存信息。

(2) 刷新周期 T_{rc} (Refresh cycle Time)

对同一存储位元连续两次刷新, 仍能保证鉴别出原存信息的最大允许间隔时间叫刷新

周期,亦称刷新间隔时间 T_{ri} (Time of refresh interval)。芯片厂家给出的 T_{rc} 远小于 T_{ref} ,从而确保读出和刷新结果的正确性。目前 T_{rc} 最小为 2 ms,也有 4 ms、8 ms 和 64 ms 的。在一个 T_{rc} 时间内,应刷新存储芯片的所有存储位元。

(3) 刷新操作周期 T_{roc} (Refresh operating cycle Time)

刷新一行存储位元即一次刷新操作所需的时间叫刷新操作周期。通常它与随机读写周期时间相同,即 $t_{RC} = T_{roc}$ 。

(4) 刷新周期数 N_r (Number of refresh operating cycle)

存储芯片所有位元刷新一遍所需的刷新操作周期的个数称作刷新周期数。它与存储芯片的内部结构有关。刷新操作是按行进行的,即每个刷新操作周期对存储矩阵的一行实现刷新,所以 N_r 决定于存储矩阵的行数。如 MCM4027 是 DRAM $4\text{ K} \times 1$ 位的存储芯片,其内部为 64 行 64 列的存储矩阵,所以每个刷新操作周期刷新一行的 64 个位元,共需 64 个刷新操作周期完成整个芯片的刷新。上节谈及的 MCM511000A 芯片,因其内部由 4 个 512×512 存储矩阵构成,刷新时提供 9 位行地址,在每个存储矩阵中都选中一行,同时进行刷新,因此每个刷新操作周期刷新 2 048 个存储位元,刷新周期数为 512。这种芯片的内部结构与同容量的存储矩阵为 $1\text{ 024} \times 1\text{ 024}$ 的芯片比较,刷新周期数减少了一半,从而提高了整个芯片的刷新速度,为访存提供了更多的时间。

2. 刷新方法

通常对 DRAM 刷新有三个要求:一是定时刷新,即严格按刷新周期规定的时间刷新;二是刷新优先于访存,但不能打断访存周期;三是刷新期间不准访存,换句话说,刷新和访存是互斥的。刷新方法有如下两种分类方法。

(1) 按刷新操作周期的分配方式分类

这种分类有三种方法。以 MCM511000A 芯片为例给予说明。MCM51000A 刷新周期 T_{rc} 为 8 ms,刷新操作周期 T_{roc} 假定为 160 ns,刷新周期数为 512,即 512 个 T_{roc} 完成整个芯片 1 M 位元的刷新,每个 T_{roc} 刷新 2 048 个位元。

① 集中式刷新

集中式刷新亦称批刷新。就是从刷新周期 T_{rc} (8 ms)中抽出最后 512 个访存周期作为刷新操作周期集中进行刷新。刷新操作周期和访存周期所需时间是相同的,对于 MCM511000A 芯片两者都是 160 ns。如图 5.34(a)所示,8 ms 中有 5 万个访存周期,其中前 49 488 个用于访存,后 512 个作为刷新周期,此时停止了访存。因此,刷新操作周期所占的比例为

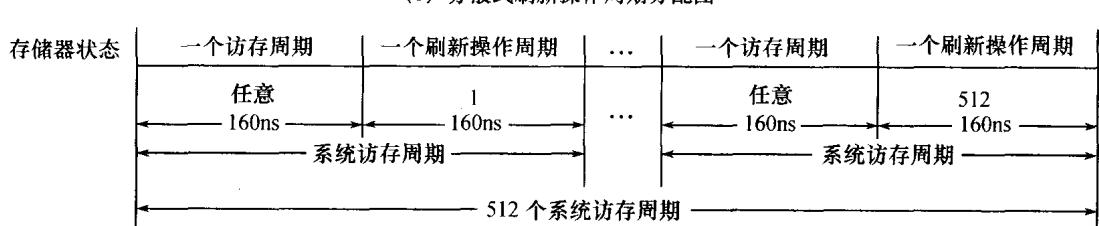
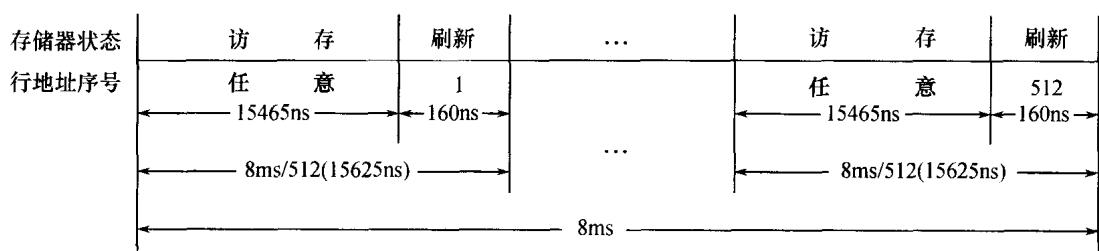
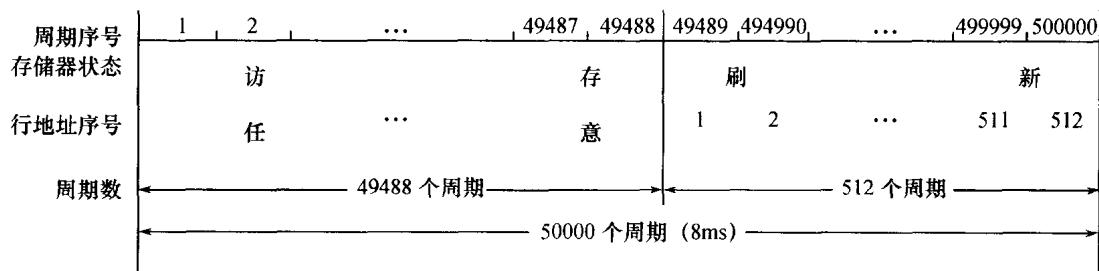
$$512/50000 \times 100\% = 1.024\%$$

可见,集中式刷新使得 98.976% 的时间充分利用了存储器,可随机访存,使存储器的效能得以充分发挥。但是 1.024% 的时间是纯等待时间,即每一个 8 ms 中都有 81.92 μs 不允许访存。这就给使用存储器带来很大的不便,影响了整个计算机的工作效率。但这种刷新

方式的控制逻辑比较简单,设计容易实现。

② 分散式刷新

这种刷新是把集中到一起的不允许访存的刷新时间分散开。即把 8 ms 按行地址译码译出的 512 条驱动线分成 512 等份,在每等份的最后留一个访存周期作为刷新操作周期,以完成一行存储位元的刷新,而其余时间则用于访存,如图 5.34(b)所示。



(c) 透明式刷新操作周期分配图

图 5.34 刷新操作周期分配图

从刷新所占的时间比例来看,分散式刷新和集中刷新是一样的,似乎没有什么好处,但实际并非如此。因为每次只刷新一行存储位元,仅占用一个访存周期,时间很短,这就有可能遇到计算机正好执行不访存的操作,不会因存储器刷新而影响计算机工作。也就是说,刷新可能插在计算机执行与存储器无关的操作时间内进行。从整体上看,它有助于提高计算机的工作效率。但这种刷新方式的控制逻辑比较复杂。

③ 透明式刷新

这种刷新方式的基本思想是设一个系统访存周期,它是存储器实际访存周期的两倍,并令系统访存周期的前半周为存储器访存周期,后半周为存储器刷新操作周期。这样,每当系

统访问一次存储器,就自动顺序刷新存储芯片中的一行位元。对于 MCM511000A 芯片,每经过 512 次访存,存储器就被刷新一遍。透明式刷新控制简单,设计容易,不需增加多少器材。但由于系统访存周期是存储器实际访存周期的两倍,存储器的速度损失了一半,存储器没有得到充分利用,所以只用于低速存储系统。

(2) 按刷新操作控制方式分类

① 同步刷新方式

如果存储器的刷新操作和访存操作都由 CPU 的时钟启动,称这样的刷新方式为同步刷新方式,同步刷新方式中的刷新地址和刷新控制信号都由 CPU 提供。为避免刷新和访存冲突,通常 CPU 都将刷新操作周期安排到非访存的时间内。同步刷新控制方式,要求存储芯片有较高的速度,以应付 CPU 频繁的访存和刷新请求。多用于对速度要求较低的计算机系统中。

② 异步刷新控制方式

异步刷新时,刷新请求和访存请求是两个独立发生的事件。访存请求来自 CPU 或 I/O 设备,刷新请求来自刷新定时器。因此,异步刷新控制逻辑中必须有裁决器,以便对这两种请求进行裁决,其控制逻辑如图 5.35 所示。

图中访存/刷新裁决器接收刷新定时器的刷新请求和 CPU 或 I/O 的访存请求,裁决后产生控制信号控制多路地址选择器。多路地址选择器接

收刷新地址计数器顺序产生的要刷新的行地址和系统地址总线(AB)送来的访存地址,在访存/刷新裁决器控制下将访存或刷新地址送 DRAM 存储矩阵,以便完成刷新或访存。时序产生器输出访存和刷新所需的控制信号。图中“系统时钟”虚线是半同步刷新时所需要的。异步刷新的优点是存储系统独立性强,易于同 CPU 匹配。缺点是控制比较复杂,所需器件多,花费也较大。

③ 半同步刷新控制方式

半同步刷新是介于同步刷新和异步刷新之间的刷新方式。其控制逻辑与异步刷新非常相似,惟一的区别是刷新定时器是由 CPU 的系统时钟驱动的,如图 5.35 所示。这样可以避免访存请求和刷新请求的冲突。例如可让访存请求发生在时钟的前沿,而让刷新请求发生在时钟后沿,以减少竞争并简化裁决逻辑。

通常在简单和慢速系统中采用同步刷新,而在快速和复杂系统中采用异步刷新,也可以采用半同步刷新。

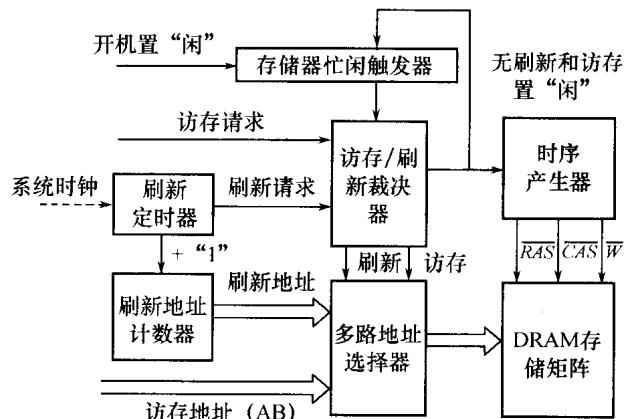


图 5.35 异步刷新控制逻辑组成

5.5 磁表面存储器

借助于磁头,利用磁记录介质局部磁化的两种剩磁状态,记忆存储信息的装置被称为磁表面存储器 MSM。磁盘、磁带、磁鼓和磁卡都属于 MSM,它们都是利用形状不同的基体表面附着的薄层磁性材料存储信息的。这些装置均含有机电部件,存取速度相对较慢且不能由 CPU 随机访问而有别于内存储器。通常归属于外围设备,它们和光盘一起统称为外存储器(External Storage/Memory)、二级存储器或辅助存储器。

5.5.1 磁记录原理

1. 数字磁记录介质

数字磁记录介质是记录二进制信息“1”和“0”的。我们简称其为磁介质。它是通过把矩磁材料涂敷、电镀或沉积在基体上制造而成的。所谓矩磁材料就是具有矩形磁滞回线的磁性材料,如图 5.36(a)所示。其有连续型和颗粒型两种。

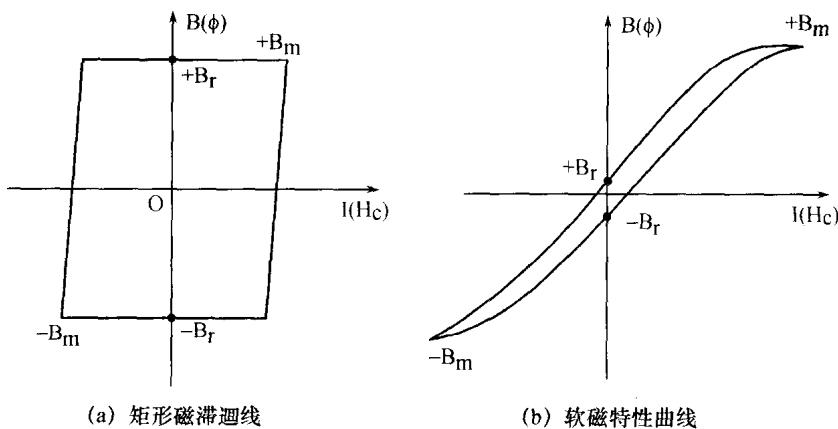


图 5.36 磁性材料特性曲线

颗粒型材料又分为金属氧化物和金属微粒,前者如 $r - Fe_2O_3$ 、 Fe_3O_4 和 CrO_2 等,后者如 Fe 、 Co 、 Ni 等,最常用的是 $r - Fe_2O_3$ 。将它们研磨成磁粉后,经筛选再与聚合粘合剂混合形成磁胶。应用旋转甩涂法涂敷在柔性盘基上制成软磁盘,用刮刀涂布法涂敷在带基上制成磁带。为改善磁性能,通过磁场取向使针状磁粒均匀整齐排列,最后烘干抛光成为磁记录介质。

连续型材料有 $Co - Ni - P$ 、 $Fe - Ni - Co$ 和 $Co - P$ 以及这些材料不同比例的组合。采用电镀、化学沉积、真空沉积和溅射的方法在刚性基体表面形成连续的磁性薄膜。

磁记录材料的最基本要求是具有硬磁特性,即在外磁场撤消后仍保持磁化状态,且具有较高的剩磁 B_r 值,适当的矫顽力 H_c 值和良好的矩形特性,即矩形系数 $S = |B_r/B_m|$ 要大。

基体分柔性和刚性两种。柔性基体通常为聚酯塑料制成的盘、带和卡；刚性基体为合金铝、有机玻璃或陶瓷。要求基体不导(绝)磁且重量轻。

对磁介质的要求是磁层厚度薄且均匀、表面光滑，温度特性好、耐磨且抗冲击。

2. 磁头

磁头可将电脉冲信号转换成介质上的磁化状态，又可将介质的磁化状态转换成电脉冲信号，因此它是电磁转换的桥梁，是磁表面存储器中的关键元件之一。简言之，磁头是对磁介质实现信息写入、读出、抹去等功能的电磁转换装置。

在一个具有缝隙的环形导磁体上绕上线圈，便构成了磁头。为加工方便，导磁体一般由两半对接而成，并包以外壳，如图 5.37(a) 所示。这里磁头存在前后两个间隙。后间隙的存在增大了导磁体的磁阻，作用是减少涡流，故一般做得非常小。前间隙是磁头极尖处的缝隙，通过它完成信息的读写，故又称它为工作间隙。工作间隙一般装有非磁性材料的隔片，如云母、玻璃等，目的是增大磁阻，使导磁体的磁力线绕过工作间隙形成漏磁场使介质可靠磁化。

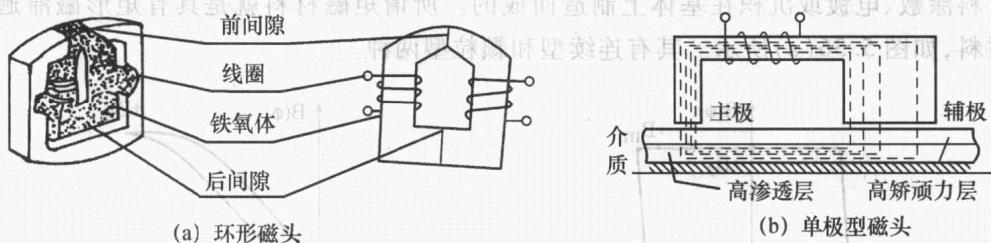


图 5.37 磁头

环形磁头的磁路是闭合的，适用于水平磁记录。而单极型磁头的磁路是敞开的，适用于垂直磁记录。如图 5.37(b) 所示，单极型磁头由主极和辅极组成，它们位于记录介质的同一侧。磁头产生的磁力线集中到主磁极的探针头，再进入双层介质。磁通经过第一层高矫顽力介质引起垂直磁化，高渗透的第二层作磁通返回通路，使磁通从辅极回到磁头。辅极面积是主极面积的千倍以上，故当磁通以极分散形式穿过第一层时，不会影响该层已记录的信息。

薄膜磁头是采用薄膜形成技术，如蒸发、溅射和电镀，加上部分集成电路工艺如光刻等技术制成的磁头。如图 5.38(a) 所示，导磁体为环形结构，与一般磁头类似，线圈为 8~30 匝，匝数愈多，写入磁场的安匝数愈大，此时电流可以较小。

无论是环形、单极型磁头，还是薄膜磁头，原理上都是感应式磁头。读出信号与通过磁头感应线圈的磁通变化率成正比，记录介质的运动速度愈快，读出信号愈大。磁阻磁头 (Magneto Resistive Head, MR) 是一种新型磁头。它采用了对磁通极为敏感的磁阻元件，感受磁场后，其电阻率随磁场的强弱成比例增减，而与介质运动速度即与磁通的变化率无关。磁阻磁头的灵敏度很高，特别适宜于磁道极窄和线速度不高的磁表面存储器中。近几年巨磁阻 (Giant MR, GMR) 磁头也投入使用，其有比 MR 磁头更高的灵敏度。图 5.38(b) 示出屏蔽

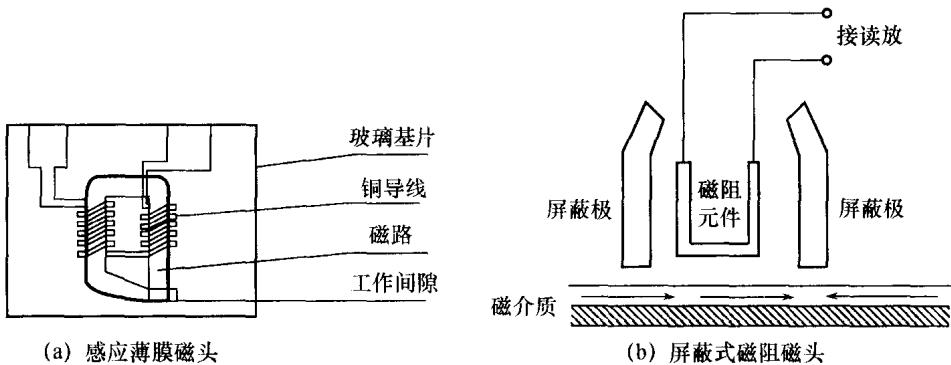


图 5.38 感应薄膜磁头和屏蔽式磁阻磁头

式磁阻磁头。MR、GMR 磁头只能用于读出, 读出原理是磁阻元件通过某方向的磁场时, 电阻减少, 电流增加, 读放输入端电压变化, 从而读出“1”或“0”。注意, 对于装有 MR 或 GMR 磁头的磁表面存储器, 还必须配备感应式磁头, 以便完成信息的写入。

磁头材料要求具有软磁特性, 如图 5.36(b)所示。即当外磁化场撤消后, 随即回到未磁化状态。通常的要求是: 导磁率高, 饱和磁感应强度(磁通密度)大, 矫顽力小, 剩余磁化强度小。这样的材料容易磁化, 也容易去磁。除此而外, 还要求高频特性好, 硬度大, 居里点高, 加工特性好。这样的材料不容易磨损, 不因温度升高而特性劣化, 且容易加工制作。

制造磁头通常有合金和铁氧体材料两类。合金材料有坡莫合金、铁硅铝合和非晶态合金等。坡莫合金的机加工性能好, 矫顽力低, 饱和磁化强度大, 但耐磨性差, 多用作磁带机磁头。非晶态合金软磁材料如钴锆、钴铁铌等具有很高的导磁率和饱和磁化强度, 矫顽力低, 耐磨性好, 多用于薄膜磁头中。

铁氧体材料如锰锌铁氧体、镍锌铁氧体等, 它们耐磨, 耐腐蚀, 高频损耗较小, 但导磁率和饱和磁感应强度(磁通密度)不高, 机加性能差。

按照磁头的使用功能, 可分为多种类型的磁头。如只有单一写入功能的写入磁头, 只能读出不能写入的读出磁头, 既能读出又能写入的读写磁头, 具有双工作间隙的写后读磁头, 用于抹除信息的抹除磁头、检测介质性能的检测磁头和专门用来读写磁道位置信息的伺服磁头等。

按照磁头的工作方式, 磁头可分为接触式磁头(Contact Head)和浮动式磁头(Float Head)。在磁带机和软盘驱动器中, 磁头与介质是接触的, 故称之为接触式磁头。由于头面之间的磨损, 将影响磁头和介质的使用寿命, 介质的运动速度也不能很高。在硬盘驱动器中, 磁头与介质是不接触的, 故称之为浮动式磁头(Float Head)。

按组合结构, 磁头又可分为单个磁头和组合磁头。组合磁头是多个磁头组合成一个整体, 以便完成特定功能。其又分成两类, 一类是多个磁头组合时, 其导磁体在同一平面, 工作间隙不在同一平面, 有前后之分。这种磁头可对相同磁道进行不同操作。另一类是多个磁

头组合,导磁体叠放,工作间隙在同一平面。这种磁头可同时对多个磁道读或写。

3. 磁记录原理

MSM 是通过磁头与磁介质的相对运动实现信息的写入和读出的。写入时,被记录的信息转换成一连串电脉冲,通过磁头在磁介质上留下一连串的饱和磁化翻转。读出时,磁头又将一连串的饱和磁化翻转转换成一连串电脉冲。

(1) 写入

如图 5.39(a)所示,记录介质在磁头下匀速移动,在磁头线圈中通入一定方向和大小的脉冲电流,则磁头导磁体被磁化,建立起一定方向和强度的磁场。由于磁头工作间隙处有较大磁阻,故在其极尖区下方形成漏磁场。在此漏磁场作用下,绝大部分漏磁通过介质与磁头导磁体形成闭合磁路。于是介质表面的微小区域的磁粒子向某一方向磁化,形成一个磁化位元。将脉冲电流转换为介质上相应磁化状态的过程称为写入。

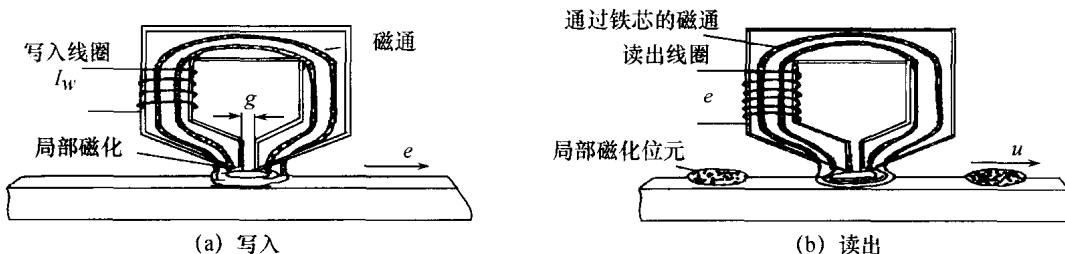


图 5.39 读写原理

(2) 存储

磁头采用的是良好的软磁材料,当写入脉冲电流消失时,磁头的导磁体马上回到未磁化状态。而记录介质采用的是矩磁材料,漏磁场即使消失,记录介质的饱和磁化状态仍保持不变。磁头写入线圈通入不同方向的电流,则漏磁场方向不同,介质形成磁化位元的饱和磁化状态也不同,两种不同的剩磁状态可分别代表“1”和“0”。因此,当记录介质运动离开磁头极尖区后,介质上的不同磁化位元即完成了信息的存储。

(3) 读出

如图 5.39(b)所示,记录有信息的介质在磁头下经过时,不论是哪一种磁化状态,磁通都将通过工作间隙耦合进入磁头导磁体形成闭合回路。由于介质与磁头间的相对运动,不同磁化状态的磁化位元使耦合到磁头的磁通发生变化,因而在读出线圈两端产生感应电势 e 。根据法拉第定律有:

$$e = -\frac{n d\Phi}{dt} = -\frac{n d\Phi}{dx} \times \frac{dx}{dt} = -nv \frac{d\Phi}{dx}$$

式中, Φ 为进入磁头的磁通量, n 为读出线圈的匝数, x 为磁头沿介质表面运动的距离, v 为介质的运动速度。显而易见,感应电压不仅与介质运动速度有关,而且与磁通的变化率成

正比。在两个磁化位元的相接处即磁通翻转过渡区中心处, $d\Phi/dx$ 最大;而在磁化位元中心处, $d\Phi/dx$ 趋近于零。感应电势 e 经放大、整形和选通后,获得符合要求的数字信号,如图 5.40 所示。

(4) 清除

清除是磁头通入交流或直流电流产生磁场,将已记录信息或残余杂散信息抹除的过程。抹除的结果,或使磁粒子无序分布,或按同一方向有序排列。数字磁记录设备中,磁带机多采用直流清除,而磁盘则多采用重写覆盖而不再专门抹除的方式。

4. 磁记录方式简介

磁记录原理只是阐述了 MSM 实现读写和存储信息最基本的原理。实际上任一种 MSM,在写入信息时,总是将要写入的信息序列,依据“某种规则”转换成记录序列,然后再写入,最终变为磁介质上的磁化位元序列;这个“某种规则”称为记录方式或磁记录编码。而将磁介质上的磁化位元序列还原为写入时的信息序列,称之为磁记录译码。

早期的 MSM,因记录密度低而采用简单的基本记录方式。基本记录方式有 8 种,它们是归零 (Return to Zero, RZ)、归偏制 (Return to Bias)、双向归零制 (Bipolar Return to Zero, BRZ)、异码变化不归零制 (Non - Return to Zero Change, NRZc)、逢“1”变化不归零 (Non - Return to Zero Change one, NRZ₁)、相位编码 (Phase Encoding, PE)、调频制 (Frequency Modulation, FM) 和改进调频制 (Modified Frequency Modulation, MFM)。现代的 MSM 为了增加记录密度,增大存储容量,提高可靠性和减少误码率,大都采用游程长度受限码 (Run Length Limited Code, RLLC) 记录方式。RLLC 有多种形式,如成组编码 (Group Coded Recording, GCR)、三单元调制码 (Three Position Modulation, 3PM) 及改进的 3PM 记录方式等。RLLC 记录方式是在基本记录方式基础上提出的,其编译码电路同基本记录方式比较要复杂得多。要掌握 RLLC,必须精通基本记录方式及其优劣的评测方法,因教学时数的限制,这里不再详述,读者可以选修《存储与外设》课程或阅读参考文献 [1]。

5.5.2 磁表面存储器的主要技术指标

和内存储器一样,容量和速度也是磁表面存储器的主要性能参数。但磁表面存储器因其自身的结构特点和存储原理,还有一些表征其性能的特定参数。

1. 记录密度

单位长度或单位面积磁层表面存储的二进制信息数量称为记录密度,通常用道密度和

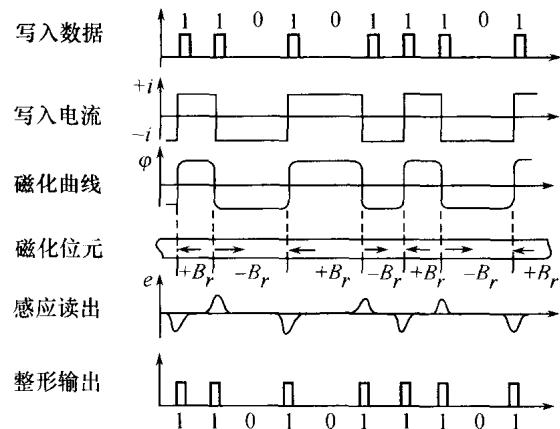


图 5.40 信息写入读出波形

位密度表示,也可用两者的乘积——面密度表示。

磁道或光道:读写时磁头或光头在介质表面扫过的轨迹。磁道与光道均有一定的宽度,叫道宽,用 W 表示。对于磁表面存储器,它取决于磁头工作间隙,对于光存储器,它取决于光头读写激光束的大小。为了避免信息道间的相互干扰,道与道之间需保持一定距离。两相邻道边缘之间的距离即为道间距或沟槽间距,用 G 表示。两相邻道中心线之间的距离叫道距,用 P_t 表示。显然, $P_t = W + G$ 。

(1) 道密度 D_t

道密度:在垂直于信息磁道或光道方向上单位长度介质所容纳的信息道数。道密度可用下式计算:

$$D_t = 1/(W + G) = 1/P_t \text{ (TPM 或 TPI)}$$

式中, W 为信息道宽度, G 为道间距, P_t 为道距, 三者单位均为毫米或英寸。因此, 道密度的单位为道/毫米(TPM)或道/英寸(TPI)。显然, 道宽和道间距愈小, 道密度就愈高。

(2) 位密度 D_b

位密度:单位长度信息磁道上所能记录的二进制信息的位数。对于磁带机, 位密度为:

$$D_b = f/v \text{ (bpm 或 bpi)}$$

式中, v 为磁带机的走带速度, 单位为毫米/秒(MPS)或英寸/秒(IPS); f 为数据传输率, 单位为位/秒(bps)。因此, 位密度的单位为位/毫米(bpm)或位/英寸(bpi)。对于磁盘或光盘机, 位密度为

$$D_b = ft/\pi D_{min} \text{ (bpm 或 bpi)}$$

式中, f 为数据传输率或记录频率, 单位为位/秒; t 为每转时间, 单位为秒; D_{min} 为最内圈磁道直径, 单位为毫米或英寸。

(3) 面密度 D_s

面密度:道密度与位密度的乘积。即单位面积上记录二进制信息的位数。

$$D_s = D_t \times D_b \text{ (bpm}^2 \text{ 或 bpi}^2\text{)}$$

对于磁盘, 通常是内圈磁道位密度大, 外圈磁道位密度小, 故不能用上述公式求 D_s 。在磁盘说明书中给出的面密度是平均面密度。

现代硬磁盘位密度很大, 其范围为 1000 ~ 10000 bpm; 道密度范围为 80 ~ 500 TPM。

2. 磁盘转速和磁带走带速度

(1) 磁盘转速 $V_{转}$

单位时间磁盘所转圈数。单位为: 转/分, 用 RPM(Revolutions Per Minute)表示。磁盘转速即磁盘主轴电机转速, 目前磁盘转速有 3600、4500、5400、7200、10033RPM 和 15000RPM 几种。

(2) 磁带走带速度

单位时间磁带的移动距离。单位为毫米/秒, 用 MPS 表示: 如 2000 mm/s; 8000 mm/s。单位还可用英寸/秒, 即 IPS 表示。

3. 存储容量

存储容量：外存储设备所能容纳的二进制信息的总量。单位为位或字节。因这两个单位太小，通常用 MB、GB 或 TB 表示。容量有非格式化和格式化之分。非格式化容量 C_n 即完全从记录密度考虑的容量；而格式化容量 C_f 即格式化后所能记录的信息量。

因格式化时，要在磁道上留出各种间隙，要写入磁表面存储器的各种标志、地址、纠错等信息。故 C_f 小于 C_n ，约为 C_n 的 74% 左右。

对于磁盘存储器，非格式化容量为： $C_n = f t m n$ (bit)

式中， f 为数据传输率，单位为位/秒； t 为磁盘转一圈的时间，单位为秒； m 为记录盘面数； n 为每面信息磁道数。因为 $f t$ 即磁道容量，用 C_t 表示，所以非格式化容量又可表示为

$$C_n = C_t m n \text{ (bit)}$$

对于磁盘存储器，格式化容量为： $C_f = B_s S_t m n$ (Byte)

式中， B_s 为每扇区的字节数， S_t 为每信息道扇区数。

对于磁带机，非格式化容量 C_n 为： $C_n = D_b L T$ (bit 或 Byte)

式中， D_b 为位密度，位密度的单位为位/毫米或位/英寸； L 为磁带长度，单位为毫米或英寸； T 为磁道数，单位为道/毫米或道/英寸。

格式化容量 C_f 与记录区长度 l 和记录区之间的间隙长度 IRG 有关，公式为：

$$C_f = C_n l / (1 + IRG) \text{ (bit 或 Byte)}$$

4. 存取时间

外存储器是机电装置，存取时间的计算与内存储器相距甚大，其主要部分是驱动磁头到指定读写位置所需时间。因此，外存的存取时间为：

$$t_a = t_s + t_w + t_{rw}$$

式中 t_s 为寻找磁道所需的找道时间， t_w 为等待所需读写的扇区旋转到磁头位置的等待时间， t_{rw} 是读写时间。显然，寻找相邻道的找道时间最短，而从首道到末道或从未道到首道的找道时间最长，前者称最小找道时间 t_{smin} ，后者称最大找道时间 t_{smax} ，两者的平均值为平均找道时间。同样，等待时间也有最小等待时间 t_{wmin} 和最大等待时间 t_{wmax} 之分。而磁盘和光盘则以转一周时间的一半为平均等待时间 \bar{t}_w 。因此，平均存取时间 \bar{t}_a 为

$$\bar{t}_a = (t_{smin} + t_{smax})/2 + (t_{wmin} + t_{wmax})/2 + t_{rw} = \bar{t}_s + \bar{t}_w + t_{rw}$$

t_{rw} 时间为 μs 级，而 t_s 和 t_w 均为 ms 级， t_{rw} 与 t_s 、 t_w 比较相差较大，所以有时可忽略不计。这里谈及的磁盘存取时间是完全从硬件的角度考虑的。实际上，操作系统对磁盘调度管理时，通过调度算法对一批磁盘请求进行调整优化，从总体上看，使磁盘存取时间中的 t_s 减小，从而缩短了存取时间。有关这方面的具体内容，《操作系统》课程将作详细讨论。

5. 数据传输率

数据传输率 f ：单位时间内外存储设备传送数据的位数或字节数。它是一个表征传输速

度的参数。

对于磁带机,数据传输率 f 为:

$$f = D_b v \text{ (bit/s)}$$

式中, v 为磁带机走带速度,单位为毫米/秒(MPS)或英寸/秒(IPS);位密度 D_b 的单位为位/毫米(bpm)或位/英寸(bpi), f 单位为位/秒(bps)。

对于磁盘或光盘机,数据传输率 f 为:

$$f = D_b \pi D_{min} / t = C_t / t$$

式中, D_b 为位密度,单位为位/毫米(bpm)或位/英寸(bpi); D_{min} 为最内圈磁道直径,单位为毫米或英寸; t 为每转所用时间,单位为秒; f 单位为位/秒(bps), C_t 为道容量。计算时可将位/秒这个单位换算为字节/秒单位。

6. 误码率

误码率:读出数据出错位数与总位数之比,它是外存设备可靠性指标之一。错误可分为软错误与硬错误两种。软错误是一种随机、独立出现的偶然性错误,故称为随机错误,如由偶然的电磁干扰引起的错误等,通常可通过重复执行读写操作予以纠正。硬错误是经重试仍不能消除的错误,如介质表面的伤痕,磁层脱落所引起的永久性差错。这种错误通常是互相牵连的成串位错,故称突发错误。突发错可通过使用备份磁道或更换盘片来处理。无论是随机错还是突发错,都可通过设置纠错编码来纠正错误。经过处理后,一般软错误应小于 10^{-9} ,硬错误应小于 10^{-12} 。

7. 磁盘计算的举例

磁盘机的盘组由9个盘片组成,其中专设1个盘面为伺服面,其他盘面为记录数据的盘面。盘存储区域内直径为6.1 cm,外直径为12.9 cm,道密度为180TPM,位密度为8 000 bpm。每道分为64个扇区,每个扇区2 K字节,平均寻道时间为12 ms,磁盘转速为7200RPM。假定 $\pi=3$,试计算:盘组容量 C_n 和 C_f 各是多少字节?格式化前、后的数据传输率 f_n 、 f_f 分别是多少字节/秒?往任一磁道1次连续写入5个扇区数据信息的平均存取时间是多少?假定系统配备上述磁盘机16台,试为该磁盘系统设计一个地址方案。

解:由已知得:

$$m = 9 \times 2 - 1 = 17$$

$$n = (12.9 - 6.1) / 2 \times 10 \times 180 + 1 = 6121$$

$$t = t_{max} = 60 / 7200 = 1 / 120 \text{ (s)}$$

$$(1) C_n = D_b \pi D_{min} mn$$

$$= 8000 \times 3 \times 6.1 \times 10 \times 17 \times 6121 / 8 = 19042431000 \text{ (B)} \approx 17.735 \text{ (GB)}$$

$$C_f = B_s S_t mn = 2 \text{ KB} \times 64 \times 17 \times 6121 = 13317120 \text{ KB} \approx 12.7 \text{ GB}$$

$$(2) f_f = C_f / t = B_s S_t / t$$

$$= 2 \text{ KB} \times 64 \times 120 = 15360 \text{ (KB/s)} \approx 15 \text{ MB/s}$$

$$f_n = C_{in}/t = D_b \pi D_{min}/t$$

$$= 8000 \times 3 \times 6.1 \times 10 \times 120/8 = 21960000 (\text{B/S}) \approx 20.943 \text{ MB/S}$$

$$(3) \bar{t}_a = \bar{t}_s + \bar{t}_w + t_{wr}$$

$$= 12 + 1/240 \times 1000 + 5/64 \times (1/120 \times 1000) = 12 + 25/6 + 5/64 \times 25/3$$

$$\approx 16.8 (\text{ms})$$

关于磁盘地址方案的设计在 5.5.3 中说明。

5.5.3 磁盘存储器

表面涂敷、电镀或沉积矩磁材料的圆形盘片叫磁盘。以旋转的磁盘作磁记录介质的装置称为磁盘驱动器(Magnetic Disk Driver, MDD),又称磁盘机。磁盘存储器由系统级接口、磁盘控制器、磁盘驱动器和盘片四部分构成如图 5.41 所示。盘片的基体有刚性和柔性的区别,刚性的盘片,称为硬盘;柔性的盘片称为软盘。所以出现了硬盘驱动器(Hard Disk Driver, HDD)和软盘驱动器(Floppy Disk Driver, FDD),因此磁盘存储器也就分为硬磁盘存储器和软磁盘存储器两大类。它们有很多相似之处,且硬磁盘存储器较软磁盘存储器复杂,加之近几年出现的半导体盘又有代替软盘的趋势,所以我们只讨论硬磁盘存储器。

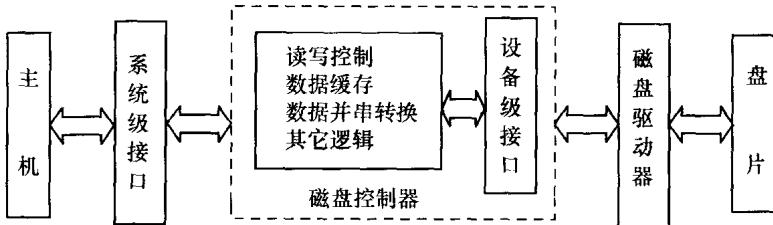


图 5.41 磁盘存储器示意

硬磁盘存储器是计算机系统中最主要的在线外存设备。自 1956 年美国 IBM 公司生产出第一台商品化的硬盘以来,硬盘在结构、性能和原理等诸方面都有了很大的发展和改进。

1. HDD 的分类及特点

由于结构上的不同,不同类型的 HDD 在性能和原理上存在一定的差异,通常可将 HDD 分为固定磁头型、移动磁头固定盘和移动磁头可换盘三大类。

(1) 固定磁头(Fixed Head) HDD

每一磁道均对应一个磁头,故又称之为每道一头磁盘。当要读写某一磁道时,只需通过电子译码器构成的磁头选择开关选取对应磁头工作。因无需寻道操作,故存取速度很快。但每道一头,磁头数量的增加使造价提高,而且因机械装配限制了道密度的提高。因此仅用于少数要求快速存取而容量不大的场合,如专用计算机系统等。

(2) 移动头固定盘(Moving Arm Fixed Disk) HDD

一片或一组磁盘片固定安装在主轴上,不可更换;每个盘面仅一个或两个磁头,所有盘

面的磁头成梳状安装在磁头架上,通过磁头定位机构驱动磁头沿盘面径向移动,以定位所需磁道。这类 HDD 大大减少了磁头数量,使造价降低且提高了道密度,扩大了存储容量。

(3) 移动头可换盘(Moving Arm Changeable Disk)HDD

顾名思义,这类 HDD 不但磁头可沿径向移动以定位所需磁道,而且盘片或盘组构成盘盒形式可由用户装卸。由于盘体可换,提供了脱机存储的能力,也就无限制地扩大了存储容量,但对制造公差要求较为严格。

实际应用中,可能以混合形式的产品出现。例如,用一片固定、一片可换的半固定 HDD,常驻信息如操作系统可记录在固定盘片上,而用户程序与数据则存放在可换盘片上。又如,有的盘组除采用活动磁头外,还在底层盘面配置固定磁头,以利于频繁访问数据的快速存取。

相对于 FDD 而言,HDD 记录密度高,存储容量大,存取速度快。主要有以下特点:

① 采用浮动式头盘结构。HDD 依靠头面之间高速气流形成微小的楔形间隙,使磁头与盘面脱离接触。这样 HDD 的高速旋转能获得较小的平均等待时间和很高的数据传输率。

② 采用快速高精度的磁头定位驱动系统。磁头能快速沿径向移动,在极短的时间内准确定位到目标磁道,从而大大缩小了平均找道时间。

③ 采用薄膜磁头或磁阻磁头,大幅度提高位密度和道密度,因而扩大了容量,提高了数据传输率。

④ 采用伺服盘或嵌入式伺服跟踪技术,从根本上解决了因温度变化引起的磁道偏离,为提高道密度和增加抗干扰能力提供了保证。

⑤ 采用直流无刷电机及调速装置和密封过滤系统,盘腔洁净,运转平稳,工作可靠,寿命增加。

⑥ 采用温彻斯特(Winchester)技术构成所谓温盘,大大提高了磁盘的整体特性。温彻斯特技术是指:

- 将磁头、盘组、定位机构以至主轴电机密封在一个盘盒内构成头盘组件(Head Disk Assembly , HAD),这样可大大减少灰尘污染,降低磁头浮动高度。若为可换式,整体装卸可消除影响磁头定位精度的机械变动因素。

- 采用小尺寸小浮力的浮动磁头和接触启停方式,介质表面涂覆润滑剂,既消除了磁头加载对盘面可能造成的冲击,又可减少头面间隙而提高记录密度。

- 将写放大器、读出前置放大器和磁头选择开关等集成化并安装在磁头臂上,改善了读写信号的高频传输特性。

2. 移动头 HDD 的结构组成

HDD 是一种精密的机电合一装置。通常由盘片与主轴驱动系统,磁头定位系统和数据控制系统组成,如图 5.42 所示。

(1) 盘组与主轴驱动系统

该系统由盘组即多个盘片、主轴电机和驱动及调速电路三部构成。其功能是使盘组转

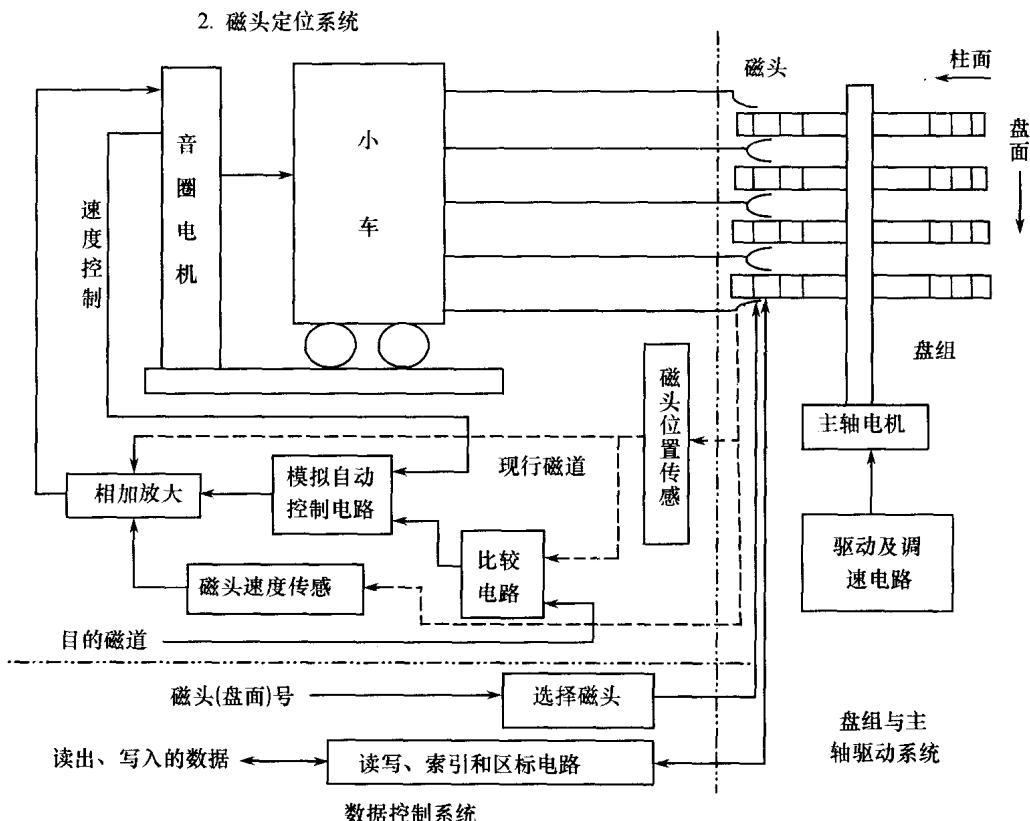


图 5.42 移动头固定磁盘 HDD 的结构组成

准、转稳。所谓转准就是达到额定的转速，转稳就是盘组转动时不偏摆，以便实现对盘组的读写。

硬盘盘片多采用铝合金盘基，值得注意的是有不少厂家采用有机玻璃盘基。有机玻璃密度高，表面光滑耐磨，对温度变化不敏感，因而易通过降低磁头飞行高度来提高记录密度，而且盘片可以做得更薄，从而降低整个 HDD 高度。

硬盘盘片直径有 14、12、10.5、8、5.25、3.5、2.5、1.8 和 1.3 英寸 9 种尺寸。8 英寸以上硬盘多用于巨、大型机；5.25、3.5 和 2.5 英寸硬盘多用于工作站、台式计算机和小型商用计算机中；1.8 和 1.3 英寸硬盘则多用于笔记本型和掌上型计算机中。

通常磁盘的 1 个盘片上下两个记录面都记录信息，且记录信息区域为环形区域。盘面的外侧一般有 5% 区域不记录信息；接着为记录信息区域，且以“0”磁道开始，越往内侧磁道号越大，通常都用外侧的若干个磁道记录系统程序；记录区域的径向长度约占外直径的 50% 左右。再往内侧也是不记录信息的空白区域。

主轴电机多采用永磁式直流无刷电机。这种电机轴向尺寸小、无粉尘，易于通过调速器

保证盘片高匀速平稳转动。现代磁盘主轴转速大多在 7200RPM 以上,希捷公司的 X15 型磁盘转速已高达 15000RPM、转速的提高大大缩短了平均等待时间,从而也减少了平均存取时间,提高了数据传输率。

(2) 磁头定位系统

在移动磁头 HDD 中,驱动磁头寻找目的磁道的机构即磁头定位驱动系统。定位的功能主要有两点:一是寻找磁道,二是跟踪磁道。定位驱动系统需要具有以下功能:

① HDD 被启动后或中途找道出错时,使磁头准确回到零道并等待找道指令。因引导程序一般存储在零道,故应具备校准零道位置的功能。

② 根据指令要求,能快速准确地将磁头从现行磁道移到目标磁道,而且能自动克服磁盘旋转偏摆所引起的磁道偏离,即可靠跟踪。

③ 停机、掉电及转速低于额定转速的 70% 时,能使磁头迅速退出盘面或进入启停区,以保护磁头和介质。一般小尺寸 HDD 的启停区在最内圈磁道以内区域。

显而易见,定位性能主要体现在速度和精度。速度快意味着寻道时间短,从而实现快速存取以满足计算机系统的要求。精度高意味着随机定位的磁道与实际的磁道误差小,这样不但能保证可靠读写,更重要的是有利于提高道密度。

定位驱动系统由驱动电机、传动部件和运载部件等组成,通常多采用步机电机或音圈电机驱动的直线运动方式或摇臂运动方式。所谓直线运动方式,磁头取数臂是沿盘面径向作直线运动寻找磁道。这种方式定位精度高,但占用空间大。所谓摇臂运动方式,是指磁头安装在摇臂上,电机通过摇臂使磁头在磁记录区域作圆弧运动寻找磁道。这种方式占用空间小,多适用于小盘径的 HDD 中。

步进电机也称脉冲电机,其工作原理是输入给电机一个脉冲,电机就转动一个角度,输入多个脉冲,就转动多个角度。下面结合图 5.43(a)说明步机电机开环定位的工作原理。步机电机开环定位是将步机电机转动变平动,再通过小车驱动磁头取数臂作摇臂运动或径向位置移动,使磁头定位到目标磁道。具体过程是:目标磁道地址与磁头现行地址比较,差值大小控制变频振荡器的频率,差值符号控制电机的转向。这样变频振荡器输出的脉冲经环形分配器按序输入到步进电机的各绕组,使步机电机转动,转动变平动,再通过小车驱动磁头寻找目标磁道。当进行到差值为零时,关闭环形分配器的输入到步进电机绕组的脉冲,即到达目的磁道。如输入 5 个脉冲,定位机构通过一个磁道道距,若目的磁道和现行磁道(磁头现在所在磁道)道差为 6,则应为步进电机提供 30 个脉冲,以便从现行磁道到达目的磁道。

之所以称之为开环定位,是因为在寻道过程中,并没有从盘片的物理磁道上取得磁道位置的反馈信号,即是否真正定位到目的磁道中心是不清楚的,也就是说,有可能偏离磁道中心。步进电机的启动频率受限,而且无论寻道距离多长,均需一步一步行进,故寻道速度较慢。因它采用开环控制方式,不能精确跟踪磁道,因此多用于道密度较低的 HDD 和 FDD 中。

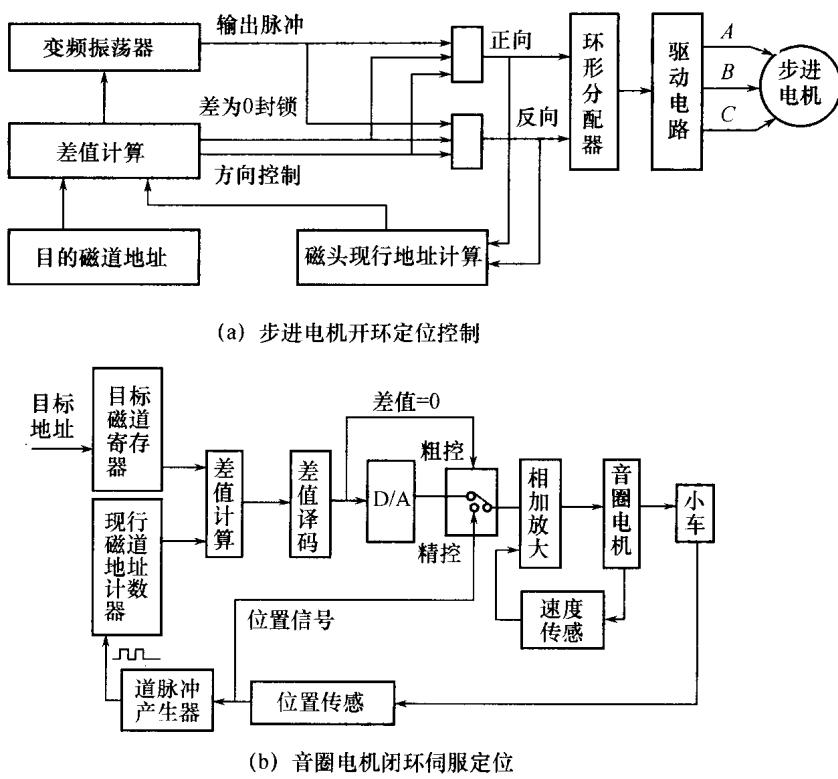


图 5.43 磁头定位控制

现代 HDD 普遍采用音圈电机闭环伺服定位系统,该系统是针对步进电机开环定位系统提出来的。音圈电机闭环伺服定位系统则是使用音圈电机的直线运动或摇臂运动工作方式寻找磁道。在寻道过程中,通过磁头位置传感装置时刻从盘片反馈物理磁道的位置信息,使磁头能精确定位到目的磁道中心。另外该系统的速度检测装置又能使磁头快速抵达目的磁道。下面结合图 5.42 和 5.43(b)说明音圈电机闭环伺服定位的具体过程。

音圈电机闭环伺服定位设计成两步进行。第一步是粗控,当磁头离目标磁道的差值大于 1 时,按最优调节原理控制速度,即追求快速。第二步是精控,即磁头离目标磁道中心只有 $1/2 \sim 1/4$ 的道差时,转为按位置信号调节,即提高精度。两种方式都通过模拟开关自动切换。

无论是粗控或精控,均需要将现行磁道地址与目标磁道地址进行差值计算,D/A 变换器将差值转换为电压,并以此作为控制小车运动速度的基准和控制粗控、精控的转换。速度传感器检测磁头的实际速度,输出的速度反馈信号与速度基准电压比较,所得误差电压经功放驱动音圈电机,带动小车以最佳速度寻道。位置传感器检测磁头的实际位置,磁头每经过一个磁道,产生道脉冲使现行磁道地址计数器计数。差值计算产生新的道差,再产生新的速度基准电压和新的寻道速度。

主机经硬盘控制器送来的目的磁道与磁头位置传感装置检测到的现行磁道比较,其结果和速度反馈信号同时控制模拟自动控制电路,经相加放大输出电流给音圈电机。因音圈电机的动圈骨架与小车连成一体,动圈骨架的平动将带动小车,小车拖动磁头架并使磁头沿盘面移动,从而快速到达目的磁道,这个过程称为粗控定位。众所周知,在寻道过程中,磁盘一直是高匀速旋转的。为跟踪磁道,使磁头一直处于目的磁道的中心,磁头位置传感装置此时直接控制相加放大,产生合适的电流,使音圈电机控制小车微动,这过程称为精确定位。

实现闭环定位的关键是磁头位置检测,目前有多种方法。如光栅位置检测法、感应同步器检测法、嵌(埋)入式伺服检测法和伺服盘定位检测法等。伺服盘定位检测法就是在盘组中专门取出一个盘面记录磁道的位置信息,其他盘面记录数据信息,前者称为伺服盘,后者称为数据盘面。数据盘面磁道的定位通过读伺服盘上的磁道位置确定。若磁头位置检测采用伺服盘定位检测法,磁盘出厂时就已经在伺服盘上写好了磁道位置信息。通常多片盘的HDD大都采用伺服盘定位检测法,而双片或单片盘的HDD则采用嵌入式伺服检测法。关于各磁头位置检测方法的细节,感兴趣的读者可阅读参考文献[1]。

(3) 数据控制系统

数据控制系统的功能是控制和实现数据的写入和读出。它包括磁头、磁头选择译码器、读出放大器、写入电路和索引、区标电路等。

3. 硬磁盘控制器 (Hard Disk Controller, HDC)

由图5.41可知,主机是通过系统级接口和HDC控制HDD实现磁盘的各种操作,实际上有的文献将系统级接口和HDC统称为HDC。系统级接口是为HDC提供数据通路并通过系统总线实现主机与磁盘的信息交换,第六章将专门进行讨论。HDC的功能是接收主机发来的命令,将它转换成HDD的控制命令,实现主机和HDD间的数据格式转换和数据传送,并控制HDD的读写。由于大规模集成电路技术的高度发展,为现代磁盘提高可靠性,加快存取速度和提高性能价格比提供了条件。导致HDC和HDD之间的功能分工不明确,使得不同类型的磁盘、相同类型不同档次的磁盘HDC和HDD之间没有清晰的标准界限。

磁盘存储器读写控制逻辑如图5.44所示。读取时,磁盘上的信息从磁头读线圈读出后,首先由读放放大,然后经数据译码器译码,再做串行/并行转换、数据装配,最后送入数据缓冲器,再经DMA或I/O通道控制将数据传送到主机。写入时,主机的数据在DMA或I/O通道控制下,首先送数据缓冲器,然后进行数据拆卸、并行/串行数据转换,再进行数据编码,最后经写入电路将写电流送磁头写线圈完成数据写入。HDC和HDD的界面可设在图5.44的①处,HDD只完成读写和放大,数据编码译码以后的控制逻辑构成HDC。ST506接口就是这种方式。如果将界面设在②处,则HDD还应完成数据编码译码操作,然后再将数据送到HDC。HDC由串/并转换、数据装拆和DMA或I/O通道控制等逻辑构成。属于这种方式的接口有增强型小型设备接口(Enhanced Small Device Interface, ESDI)、SMD等。第三种方式是将接口的界面设在③处,HDC的功能完全放到HDD中,HDD与主机之间采用标准的系

统级接口。小型计算机系统接口 (Small Computer System Interface, SCSI)、集成驱动电路 (Integrated Drive Electronics, IDE) 和增强集成驱动器电路 (Enhance IDE, EIDE) 等就是这种形式。目前的趋势是增强 HDD 的功能,以便磁盘相对独立。

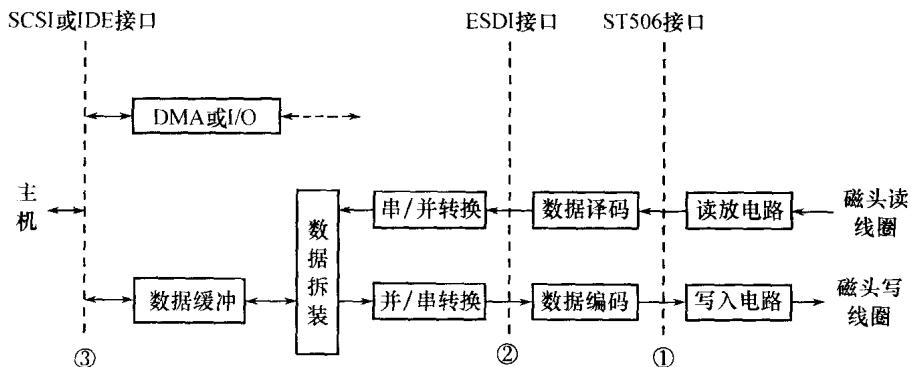


图 5.44 磁盘存储器读写控制逻辑

4. 磁盘的信息存储方式和记录格式

磁盘与主存间成批交换信息,每次交换的一批信息对应磁介质上的一个或多个信息块,通常信息块是信息交换和存储的基本单位。信息块由若干字节组成,一个字节可以表示字母、数字或符号,用它们可以表示数据、程序和其他信息。在磁盘中,有两种信息块记录:定长度记录和不定长度记录。所谓定长度记录就是每个信息块记录的字节数是固定不变的,一个文件有多个信息块,最后一个信息块可能有空白位置,因此定长记录的盘面利用率不很高。不定长记录就是各信息块记录的字节数不定,实质上一个信息块就是一个独立的文件,这种记录的盘面利用率较高,但它的寻址、校验及对标志等的设置均很复杂,对硬件和软件的要求高,所以目前采用的不多。

HDD 通常都驱动多个盘片,称这多个盘片为盘组。信息块记录在盘面上,一般按柱面、盘面和扇区三个层次存储。盘组中所有盘面相同半径磁道的集合构成一个柱面 (Cylinder),又称圆柱,如图 5.42 所示中的盘组与主轴驱动系统。一个磁道就是一个圆周,将一个磁道等分成若干个弧段,称每个弧段为一个扇区或扇段 (Sector)。通常每个扇区记录一个信息块。

在移动头盘组中,磁头定位机构一次定位的磁道集合正好是一个柱面。显然,磁盘的柱面数就是其中任意一个盘面上的磁道数。柱面选定之后,再由磁头选择译码电路选定某一个磁头,也就是选定某一盘面的某一磁道。因此,信息交换时不能跨柱面交换,也就是说,磁盘一次操作,最多能交换一个柱面的信息量。之所以如此,是为了提高数据传输性能。若跨柱面交换,则需要多次磁盘操作。

和主存一样,磁盘存储器也是凭地址存取,不同在于其与主机是成批交换信息,地址指向起始扇区,且以扇区为单位交换信息。所以磁盘存储器地址的一般形式为:

台号·柱面(磁道)号·盘面(磁头)号·扇区(扇段)号

其中台号用以区别是计算机系统的哪一个磁盘驱动器,若系统仅配一台磁盘,则台号可省略。磁盘存储器地址亦称磁盘设备内部地址,并在 HDC 中设置相应的寄存器寄存。将磁道划分成扇区有硬划分和软划分两种方法。硬划分即物理划分,即在磁盘上设置物理标志,如孔、凹凸等,通过光电或磁电的方法识别扇区。软划分就是在扇区起始处记录磁盘地址和标志信息,扇区间留有间隙,间隙中记入特殊信息或空白,地址和标志等信息称为标识或 ID。目前大多数采用软分区法,因为这种方法划分扇区灵活方便。扇区的数目及存储信息的多少因不同的机器而异。

SA1000 型 5.25 英寸温盘的记录格式,如图 5.45 所示,采用的是软分区法,扇区的数目、大小及间隙如表 5.11 所示。

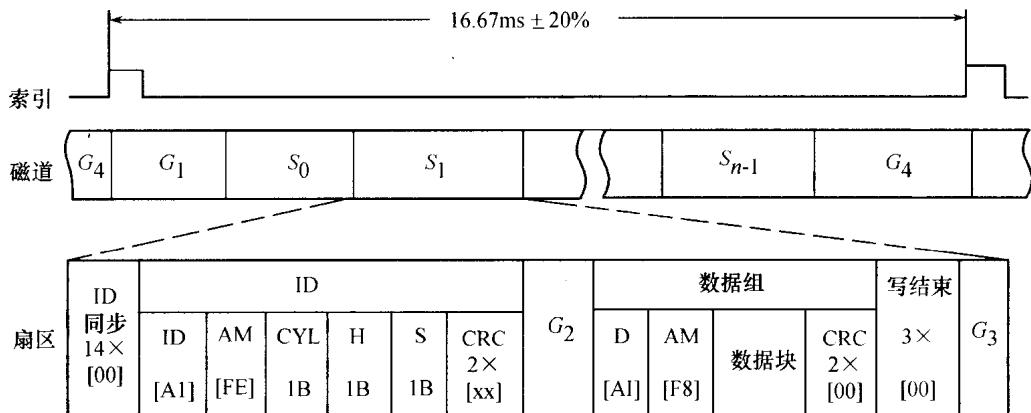


图 5.45 5.25 英寸温盘记录格式

表 5.11 5.25 英寸温盘格式

扇区数	每扇区字节数	G ₁	G ₂	G ₃	G ₄
32	256	30 × [4E]	15 × [00]	15 × [4E]	340 × [4E]
17	512	30 × [4E]	15 × [00]	30 × [4E]	442 × [4E]
9	1024	30 × [4E]	15 × [00]	60 × [4E]	224 × [4E]

磁道内为什么要留有间隙呢? 其作用是:区分不同扇区和扇区内不同信息区;留出控制余量,以便在环境、电压变化或换盘组时,确保可靠读出;为同步、伺服定位和读地址等信息提供缓冲时间。硬盘出厂前要进行格式化,所谓硬盘格式化是指:依记录格式要求,对磁道划分扇区,留出间隙并写入各种标志信息、地址信息和其他控制信息,如纠错编码信息 CRC、ECC 等。即写入除数据信息外的其他所有信息,请注意,这种格式化,是初级格式化,其和操作系统中的高级格式化是有差别的。

下面我们解决上节遗留的磁盘计算举例中为磁盘系统设计一个地址方案的问题。

磁盘地址方案：

台号位数： $\lceil \log_2 16 \rceil = 4$ 位；柱面号位数： $\lceil \log_2 6121 \rceil = 13$ 位

盘面号位数： $\lceil \log_2 17 \rceil = 5$ 位；扇区号位数： $\lceil \log_2 64 \rceil = 6$ 位

27.....24 23.....11 10.....6 5.....0

台号	柱面号	盘面号	扇区号
----	-----	-----	-----

5. 现代硬盘采用的新技术措施

硬盘的发展已经历了 40 多年。1956 年美国 IBM 公司制造出第一台实用硬盘，存储容量仅 5 MB，盘直径为 600 mm(24 英寸)，重量却高达 100 公斤，耗资数十万美元。即使到了 1973 年，硬盘结构发生了质的变化，使用温彻斯特(Winchester)技术生产的第一台“温盘”，存储容量也仅为 640 MB，盘径仍然较大，为 300 mm(12 英寸)，重量仍达 30 公斤。直到 20 世纪 90 年代多媒体计算机的出现，为了满足存储视频、语音和音乐等信息，以及 Windows 操作系统和便携机的需要，硬盘才呈现突飞猛进的发展。近 10 年来，磁盘的直径缩小了 10 倍，单位面积存储密度提高了数十倍。21 世纪初，市场上单片硬盘存储容量高达 40 GB，盘直径一般为 133 mm(5.25 英寸)和 89 mm(2.5 英寸)，重量不足 1 公斤，每 GB 价格低于 2 美元，平均存取时间最快达 5.2 ms。数据传输率通常在 20 ~ 50 MB/S，WD1200JB 磁盘的数据传输率已高达 65.6 MB/S，有的甚至高达 90 MB/S。表 5.12 列出了几种现代磁盘的主要参数。

表 5.12 现代硬盘参数一览表

型号	容量(GB)	转速(RPM)	平均寻道时间	缓存	盘片数	单片容量	接口
92041U4	20.4	5400	9.5 ms	512 KB	2	10.2 GB	UDMA66
96147U8	61.5	5400	小于 9 ms	2 MB	4	15.3 GB	UDMA66
31536U2	15.3	5400	小于 9.5 ms	512 KB	1	15.3 GB	UDMA66
5T060H6	60	7200	小于 8.7 ms	2 MB	3	20 GB	UDMA100
6L080J4	80	7200	小于 8.5 ms	2 MB	2	40 GB	UDMA133
6L040J2	40	7200	小于 8.5 ms	2 MB	1	40 GB	UDMA133
ST114601	146.8	10033(10000)	小于 4.9 ms	4 MB	3	48.9 GB	SCSI
ST318150	18.4	15000	3.2 ms	8 MB	1	18.4 GB	Ultra320
ST19171	9.1	7200	5 ms	2 MB	10	0.9 GB	SCSI

那么究竟采用了哪些新技术新措施，使磁盘有如此之高的性能价格比呢？

(1) 提高记录密度，增大存储容量采取的技术措施

- 采用高密度记录磁头和高编码效率的 RLLC 记录方式；
- 采用浮动磁头，降低浮动高度，使磁头和磁介质间距仅为 0.1 μm ；
- 采用高性能磁介质并使磁层尽量薄；

- 采用单极型磁头,变水平磁化位元为垂直磁化位元;
- 采用嵌入式伺服或光伺服技术,提高道密度,目前道密度为 100 ~ 700TPM。
- 采用 MR 和 GMR 磁头并采用 PRML 信号处理技术。PRML (Partial Response Maximum Likelihood) 部分响应最大似然读磁道技术。其利用一个逻辑规则分析磁头读出的一组数据,并归纳出一个最接近于标准数据的信号,传输出去使用。据统计可提高记录密度 50%。

(2) 提高转速,缩短存取时间采取的技术措施

- 采用“液态动力轴电机”技术,提高磁盘转速,延长使用寿命。通常为 5400/7200RPM,最高达 15000RPM。增加转速,减少等待时间。

- 从盘片厚度和盘片直径两个方面减小磁盘尺寸,从而在转速增加的同时也缩短了磁头行程,减少寻道时间。

- 在 HDD 中设置“磁盘 Cache”,容量在 1MB 至几 MB,通常由 SRAM 或 DRAM 组成。“磁盘 Cache”是在主存和磁盘之间设置的高速缓存,其速度同主存相匹配,“磁盘 cache”一次存取数量大,数据集中,速度要求比 CPU 的高速缓冲 Cache 低,且大容量缓存管理工作较复杂,因此“磁盘 Cache”的管理和实现一般由硬件和软件共同完成。

(3) 提高可靠性,减少误码率采取的技术措施

- 继续完善“温彻斯特”技术,使磁盘结构更紧密,体积更小,重量更轻。
- 采用更高效的纠错技术 (Cyclic Redundancy Check/Character, CRC) 和 ECC (Error Corrected Code)

- 采用 SMART 技术。SMART (Self - Monitoring Analysis and Reporting Technology) 技术即自动监测分析和报告技术。自动监测分析硬盘的性能和工作状态,并在 CRT 上显示出来。该技术是软件为主,硬件为辅实现的,其监测磁头、磁介质、主轴电机、电路及芯片的故障和驱动器与总线连接上是否有问题等。

(4) 从体系结构考虑,构造磁盘子系统——廉价冗余磁盘阵列 (Redundant Array of Inexpensive Disk, RAID)

尽管单台磁盘存储器速度有了很大提高,但与主机相比,仍有很大差距。这就使得磁盘存储器成了整个计算机系统功能提高的瓶颈,为此提出廉价冗余磁盘阵列 RAID。RAID 是用多台磁盘存储器组成的大容量磁盘子系统。它的基本原理是将并行处理技术引入到磁盘子系统中,即将数据分放在多台磁盘上,使之可以并行存取。在阵列控制器的组织管理下,能实现数据的并行、交叉存储或单台存储操作。由于阵列中一部分磁盘存有冗余信息,若系统中某一磁盘失效,可利用冗余信息重建用户数据。RAID 使存储容量大大增加,使数据传输率成数倍增长。有关 RAID 的具体内容,《计算机体系结构》课将作详细讲述。

6. 半导体盘

半导体盘,俗称 U 盘。它是以半导体存储芯片为核心,以半导体逻辑芯片作为外围电

路,在功能上模拟硬盘,即按硬盘的工作方式存储信息。实际上,它即没有盘,也无其他运动部件,更没有机电装置。因此其存取速度比硬盘快得多,通常是几十至几百微妙。

半导体盘通常使用的是 EEPROM 或“Flash Memory”存储芯片。主要是使用 Flash Memory 存储芯片,因其位功耗小故集成度可以很高,且能整片或分块擦除,单地址写入。目前市场上有 32 MB、256 MB、512 MB 和 1 GB 等多种容量的半导体盘。半导体盘目前惟一的缺点是和同容量的硬盘相比,价格要贵得多。

5.6 光盘存储器

将激光聚焦成极细光束对存储介质读出信息,就是所谓的光存储器(Optical Memory)。光存储的本质不在于其记录过程,也不在于介质上信息的存储形式,而在于读出过程。因此,无论采用哪一种方式存储信息,只要使用的是光学方式读出,就可认为是光存储器。

广义上说,光存储器包括激光逐位存储、全息存储及其他光存储,甚至还包括条码阅读器、光电阅读机等。一般情况下,光存储器仅指那些作为计算机外存的存储器,如光盘机、光带机、光卡机等,其中以光盘机应用最广。

与磁盘存储器相比,光盘存储器记录密度高,单片盘存储容量大,非接触式读写,光盘易于更换便于保管,对环境条件没有苛求。但是,光盘机寻道时间长,可擦写性能不如磁盘快。尽管目前光盘的某些性能稍逊色于磁盘,但其潜力巨大,是计算机系统中一种新型的外存储器。

光盘存储器逻辑框图如图 5.46 所示。光盘盘片是记录信息的基体,光盘驱动器(Optical Disk Driver, ODD)主要由主轴电机驱动机构、定位机构、光头装置及它们的逻辑电路组成,其中光头装置最复杂,是驱动器的关键部件。ODD 功能是驱动光盘转稳、转准,寻找光道并借助于光头和激光器完成读写操作。光盘控制器(Optical Disk Controller, ODC)主要由数据输入输出缓冲器、编码器、译码器和并串转换电路等组成。其功能是通过 I/O 接口实现主机与 ODD 的信息交换,控制 ODD 驱动盘片旋转、寻光道和进行读写操作。

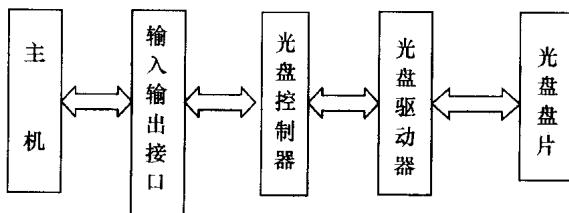


图 5.46 光盘存储器逻辑示意

5.6.1 光盘记录介质

光盘盘片主要由盘基、记录介质层和密封保护层构成。

1. 盘基

盘基直径尺寸有 12 英寸 (300 mm)、8 英寸 (200 mm)、5.25 英寸 (133 mm)、4.75 英寸 (120 mm) 和 3.5 英寸 (90 mm) 等多种, 厚度通常为 1.1 ~ 1.25 mm 左右。盘基材料有聚甲基丙烯酸甲脂 (PMMA)、聚碳酸酯 (PC)、硼硅酸玻璃和二氧化硅等。PMMA 是一种耐热的有机玻璃, 热传导率低, 用于记录信息的激光功率小, 应用较为普遍。盘基材料要求热传导率低、有较好的强度和平直度。

2. 记录介质层

光盘记录介质层, 简称纪录层。其厚度为纳米级 (nm), 通常为数十至数百 nm。依其工作原理可分为形变型、相变型和磁光型三类。

形变型记录层在激光束照射下发生永久性变形, 而变形的方式可以是有凹坑型、发泡型和热平滑型等。凹坑型是最常见的一种, 可采用有机染料苏丹黑 B、碲或采用新型的碲 - 碳 (Te - C) 等材料。这些材料对激光有良好的吸收能力, 熔点低且有较好的灵敏度和稳定性, 便于用涂布工艺连续制造, 易形成大面积均匀记录层的特点。

相变型记录层由于在晶态与非晶态时, 光学性能有明显的差异。有些物质如锑硒 (Sb - Se) 化合物和碲的低氧化物 TeO_x ($x \approx 1.1$) 等, 在激光照射下可实现从晶态至非晶态的转换。不同状态下对入射光有不同的反射率, 如非晶态反射率约为 10%, 晶态时反射率可提高至 30% 左右。但上述转换是不可逆的, 故只能写入一次。称这种记录层为不可逆相变型记录层。

在不可逆相变记录层中增加一些元素即变为可逆相变记录层。如在 TeO_x 中添加锡 (Sn)、锗 (Ge), 或在碲中掺入锡锗 (TeGeSn)、掺入锗砷 (TeGeAs)、掺入锗锑硫 (TeGeSbS), 只要激光加热过程的温度与时间适宜, 就可实现晶态、非晶态的相互转换, 这种可逆性的特点正好满足可改写型光盘记录层的要求。

磁光型记录层易磁化方向垂直盘片表面。目前这类材料有三种: 一是锰铋系晶体, 如 $\text{MnBi}/\text{MnCuBi}$ 等; 二是柘榴石系单晶, 如 TbFeO_3 等; 三是稀土类铁系非晶体, 如铽铁 (TbFe)、钆铁 (GdFe) 和钆钴 (GdCo) 等。

3. 保护层

保护层的作用是使记录层免受水蒸气等的腐蚀, 减少灰尘、指印和划痕等对读出的影响。通常的方法是在介质记录层表面直接覆盖一层厚度约 50 ~ 200 μm 的透明聚合物。也有将盘基与保护层功能合一, 通过垫环, 将两张基片与记录层粘结成一个空腔。腔内充以惰性气体, 使记录层与大气隔绝从而达到保护的目的。

5.6.2 光盘分类

按存取方式光盘有只读型、追记型和可擦写型三大类。

1. 只读型光盘 CD - ROM

这是一种只能读出不能写入的光盘, 由生产厂家记录好信息, 用户使用时不能更改、增

添和删除。这种光盘直径为 4.75 英寸,容量为 650 MB,可存储全数字化的文字、声音、图形、动画及全活动视频影像。这类光盘有各种不同的产品:小型数字化音频唱片(Compact Disc - Digital Audio, CD - DA)、交互式光盘(CD - Interactive, CD - I)和激光图示唱片(CD - Graphic, CD - G)等。

这类光盘采用形变型介质,光道是由内向外延伸的一条螺旋线,扇区长度相同、位密度也相同。工作时,线速度相同,角速度即转速时刻变化,故称其为恒线速盘(Constant Linear Velocity, CLV)。通常用铝作反射层,俗称银盘。

2. 追记型光盘(Write Once Read Many, WORM)

用户可在空光盘上写入信息,但一经写入便不可修改重写,是一种一次写入多次读出光盘。在文档存储和检索、图像存储和处理方面显示出其重要的作用,但其驱动器价格较高。

WORM 有两种尺寸:5.25 英寸和 12 英寸。存储容量从 2 GB 至 3 GB。

可录小型光盘(Recordable, CD - R)也是可写一次不可重写的光盘,其尺寸与 CD - ROM 相同,但用金作反射层,俗称金盘,CD - R 盘片容量有 157、550 和 650 MB 三种,分别对应 18、63 和 74 分钟。按照 CD - ROM 格式写好的 CD - R 盘可在 CD - ROM 驱动器中读出;按照 CD - I 和激光唱盘格式写好的 CD - R 盘可在 CD - I 系统或激光唱机上播放。

其实 CD - R 是综合 WORM 和 CD - ROM 的特性开发而成。不过,WORM 的着眼点主要是大信息量的永久保存,光道通常为一组同心圆;而 CD - R 则着眼于多媒体系统,光道为一条从内向外的螺旋线。这类光盘可采用不可逆相变型材料或形变型材料作记录层。

3. 可擦写型光盘(Rewritable)

这种光盘类似于磁盘,用户不仅可以读取和记录信息,而且可以抹除和改写信息。其可采用磁光型或可逆相变型两种记录层。

磁光型光盘(Magneto Optical, MO)是利用热磁效应写入,使磁光型记录层磁化取向向上或向下来实现信息的记录。MO 光盘有两种尺寸规格:3.5 英寸,容量为 128 MB;5.25 英寸,容量为 1 GB。一般的 MO 缺乏直接覆盖重写能力,即先要将 MO 上的信息抹除,然后才能写入,而不能像磁盘那样直接覆盖重写。但最新式的 MO 不仅单面容量可达 2.3 GB、双面可达 4.6 GB,而且具有可直接覆盖重写功能,其读写速度几乎与普通硬盘一样快。随着技术的发展,磁光盘将实现超高密度,预计 3.5 英寸磁光盘的容量高达 6 GB,5.25 英寸的容量可提高到 29 GB。

相变型光盘(Phase Change Disc, PCD)利用激光的热效应写入,使介质在晶态和非晶态之间的可逆相变实现反复擦写。5.25 英寸 PCD 容量可达 1 GB。

表 5.13 对上述几种光盘的性能进行了比较。

可擦写型光盘亦可缩写为 CD - R/W 或 E - R/W,其光道为同心圆。工作时,角速度相同,线速度不同,因此扇区长度和位密度将因光道不同而异,故称其为恒角速盘(Constant Angular Velocity, CAV)。

表 5.13 几种典型光盘参数对比

名称	CD - ROM		WORM		CD - R	MO		PCD
尺寸		4. 75"	12"	5. 25"	4. 75"	5. 25"	3. 5"	5. 25"
容量 (MB)	650	650	3000	2500	550 650	1000	128	1000
传输率 (MB/s)	0.3	0.6	1.1	0.8	0.307 0.353	4.2 1.4	3.0 0.625	最大 10.30 平均 7.92
$t(ms)$	195	195	190	90	1000	19	< 40	90
RPM	-	-	720 ~ 360	1800	-	3600	3000	-

按记录介质又可把光盘分为形变型光盘,相变型光盘和磁光型光盘三类。

形变型光盘依记录层的形态发生变化记录信息,其是不可逆的,VCD、DVD - ROM 和 CD - ROM 属于此类。

相变型光盘是依记录层的相变记录信息的,其又分为不可逆相变和可逆相变两种光盘。追记型光盘属于不可逆相变光盘。

磁光型光盘是依记录磁层的两种剩磁状态记录信息的。其和可逆相变光盘又统称为可擦写型光盘。

5.6.3 光盘读写原理

无论是形变型、相变型光盘,还是磁光型光盘,信息的记录与读出原理均与磁记录原理有较大的差别。

1. 形变型光盘

形变型光盘盘片通常从下往上共分为四层,依次为盘基、形变型记录层、反射层和保护层,如图 5.47 所示,盘片总厚度为 1.1 ~ 1.3 mm。反射层厚度 200 nm 左右。其材料通常为金属铝或者金属金,其作用是读出时反射激光。聚焦成直径不到 1 μm 的激光束照射到存储介质上。由于记录层的吸光能力强,熔点低,被照区域迅速升温而被熔化、蒸化并形成凹坑。凹坑的有无即代表二进制数据“1”和“0”。显然,凹坑一旦形成,便无法复原,因此这种光盘是不能改写的。

读出时,经聚焦的比写入时功率低的激光束投射到光盘上,依激光在凹坑上和非凹坑上反射激光的强弱不同,通过光电转换器件变成电信号,即可判断出所存信息是“1”还是“0”。

2. 相变型光盘

写入激光束加热记录介质不是烧蚀凹坑,而是改变记录层的晶体状态。依结晶态和非结晶态对入射激光不同的反射率来检测“1”和“0”。如图 5.48 所示,写入时,用高功率激光束(如 18 mW)去照射相变记录层,使其局部温度升高到熔点(如 600°C),然后急剧冷却,使之处于非结晶状态。抹除时,用中功率激光(如 8 mW)使局部温度升高到结晶温度(100°C)

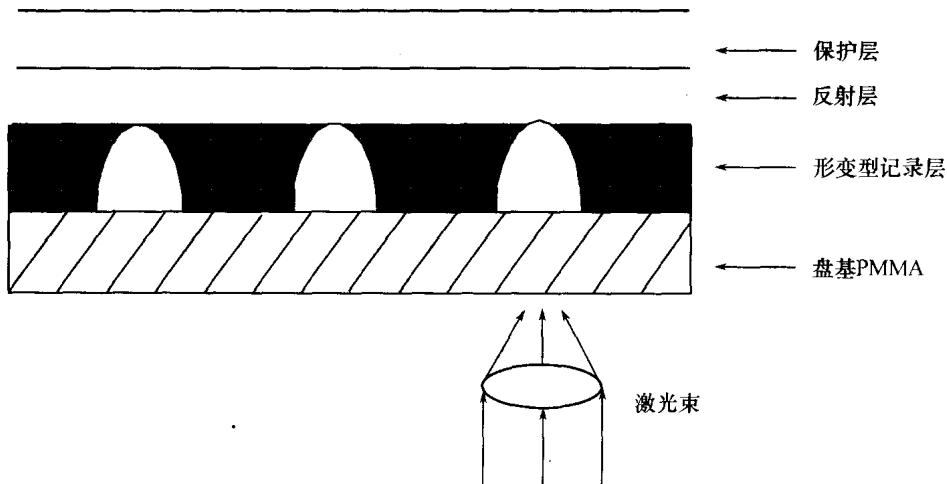


图 5.47 形变型光盘记录原理

以上),然后急剧冷却,它将处于结晶状态。读出时,用低功率的激光(如 2 mW)照射,不会改变记录层的结晶状态。当然在常温下,晶体的状态均保持不变。

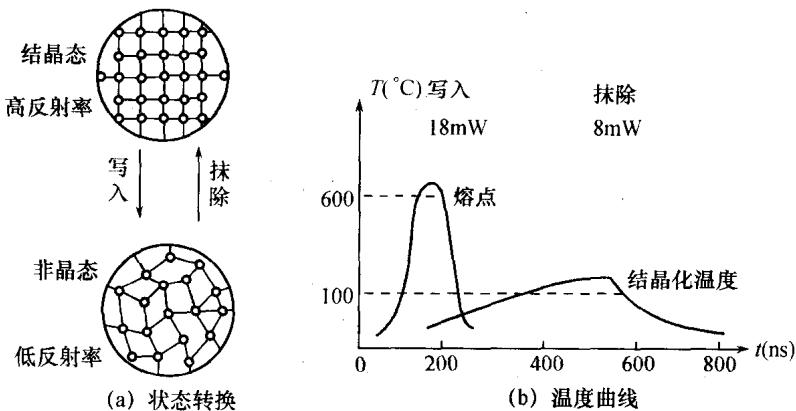


图 5.48 相变型光盘记录原理

3. 磁光型光盘 MO

MO 的特点是热磁效应写入,光磁效应读出。对于目前应用较多的稀土类铁磁介质,易磁化方向垂直于盘片表面。通常工作温度下,其矫顽力 H_c 较大,当温度升高达到某个值时,介质记录层矫顽力 H_c 将大大减小,变得容易磁化。称该温度值为居里点,通常在 150℃ ~ 200℃ 之间。MO 热磁效应写入过程如 5.49(a) 所示,写入时在盘片下方设置一个单极性磁头,其线圈通电时,产生一个稳定的垂直磁场 H_v 。该磁场不会使介质记录层的磁化状态发生变化。当聚焦的激光束和磁头一起对准光盘上某一记录位元位置时,该点温度上升,介质

的矫顽力随温度的升高而减小,当温度达到居里点后,磁头磁场 H_w 大于介质矫顽力 H_c ,使介质记录层磁化状态翻转,即磁化方向向下,从而实现了写入信息“1”。磁头和激光束离开该位元,温度从居里点下降到工作温度,翻转了的饱和磁化强度和矫顽力随温度降低而增大,保持已翻转的磁化状态,如图 5.49 (b) 所示。

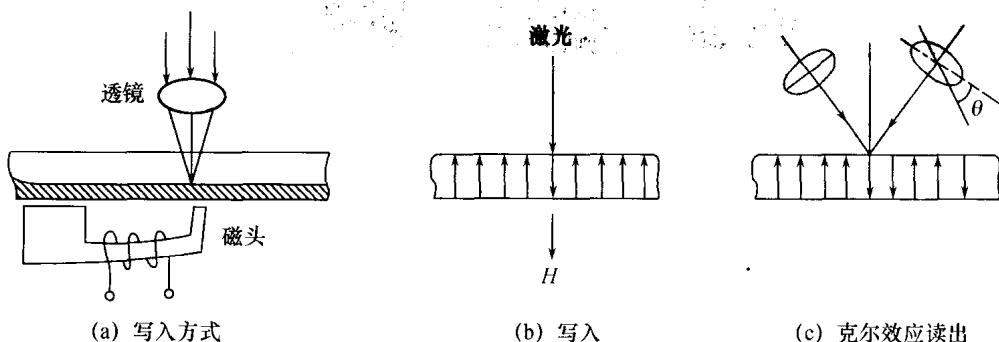


图 5.49 MO 信息的写入与读出

抹除原理类似于写入。用激光照射的同时,加入一个与写入磁场 H_w 相反的磁场,使这一区域磁化翻转恢复到向上的磁化状态,即“0”状态。

读出时,利用光磁相互作用即磁光效应来检测介质的磁化方向。所谓磁光效应是指对应于不同磁化方向上的反射光或透射光,其偏振面将向不同方向偏转的现象。若反射光偏振面发生偏转,称为克尔效应,其偏转角 θ 称为克尔角。若透射光偏振面发生偏转,称为法拉第效应,其偏转角称为法拉第角。

克尔效应适用于不透明的垂直磁化介质。激光束通过起偏器变成线偏振光,当其照射到介质表面时,反射光偏振面与磁化方向有关。若磁化方向向上,反射光的偏振面旋转 $+ \theta$;若磁化方向向下,反射光偏振面旋转 $- \theta$ 。如果检偏器与 $+ \theta$ 垂直,那么磁化方向向上的反射光不能通过检偏器,检测器不进行光电转换,无信号输出,视为“0”;磁化方向向下的反射光通过检偏器,光电转换器有信号输出,视为“1”,如图 5.49 (c) 所示。图中实线示出磁化方向向下时反射光的偏振面,虚线示出无磁化时的反射光的偏振面,相对于无磁化时偏转了 $- \theta$ 。

法拉第效应适用于透明的记录介质。如图 5.50 所示,偏振光透过介质层后,偏振面会发生偏转。当偏振光透过与磁化方向向上的磁层时,偏转角为 $+ \theta$;透过与磁化方向向下的磁层时,偏转角为 $- \theta$ 。通过检偏器,用和克尔效应同样的方法,可以读出“1”和“0”。

磁盘可重写覆盖,即磁头扫描一次即完成了写入。因此,其读和写的时间相等。而 MO

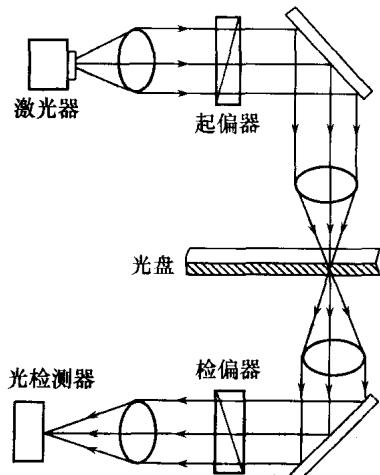


图 5.50 法拉第效应读出

写入前必须将原有信息抹除,需二次扫描。因此,写入时间是读出的 2 倍。如写入后还考虑读出校验,则写入时间是读出的 3 倍。

5.7 计算机的存储系统

前六节介绍的半导体存储器、磁表面存储器、光盘存储器主要是从存储原理、存储器本身结构、存取方式等方面论述的。本节拟从存储系统组织的角度简单讨论一些技术,如为提高主存存取速度和数据传输率的并行主存系统,为满足 CPU 对访存速度要求设置的 Cache、主存层次,为扩大用户编程使用空间的要求增设了主存、辅存层次等。

5.7.1 并行主存系统

计算机自动从主存中取出指令、依需要还可能从主存取出要加工处理的信息,然后执行指令,从而完成数值计算或非数值处理。因此,主存速度将直接影响着计算机速度。计算机期望主存速度快、容量大、位价格低。但从计算机技术发展看,存在两个不争的事实,一个为主存速度总是落后于 CPU,原因是对于 CPU 芯片,若晶体管数量增加一倍,速度也几乎增加一倍;而对于存储芯片,若其晶体管数量增加一倍,速度不但不能加快反而会变慢。这就导致 CPU 芯片以年 60% 的速度增加,而主存芯片仅以 7% 速度增长。另一个是主存的容量总是不满足软件的需求,原因是不管有多大的主存,人们都能开发出足够大的应用程序,使得主存不够使用。而目前主存单体即物理存储体最大为 256 MB,再加大容量十分困难。因此,单从存储技术本身提高主存速度和容量不能满足计算机系统要求,必须采取有效措施,构成主存储系统以满足计算机对主存的要求。所以人们为提高主存速度而构建多端口主存系统,为提高主存容量和存取速度,提出单体多字并行主存、多体交叉编址并行主存系统。

在计算机系统中,主存数据传输率即主存频宽用下式表示:

$$B_m = w/T_m$$

其中 T_m 为存储周期; w 是一次读写的位数,即存储总线可同时传输的数据位数; B_m 表示单位时间通过存储总线的数据流量,即主存的数据传输率,通常希望它越大越好。

从上式可知,在 T_m 一定的条件下,要成倍提高 B_m ,只有成倍加大 w ,即一个存储周期读写更多位。换句话说,即一个存储单元包含存储位元的数目成倍增加,由 5.2 节所学知识可知,通常主存速度就比 CPU 慢一个数量级,对于单体单字主存,当 w 增大时, T_m 变大,这将使 B_m 又趋减小。为解决这些问题,增大主存频宽,弥补主存与 CPU 的速度差距,人们提出采用并行主存结构。其特征是一个存储周期可读出多个字。

1. 单体多字并行主存系统

图 5.51 所示的主存系统称为单体多字并行主存系统,该系统由存储单元数为 m 、字长为 w 的 n 个物理存储体组成, n 一般为 2 的整数次幂。不难看出,该主存系统的特征是多个

物理存储体共有一套地址寄存器 MAR 和地址译码器 MAD; 按同一地址码并行地读写各物理存储体的对应单元, 如地址总线 AB 送来地址为 $m - 2$, 则 n 个物理存储体同时读写 $m - 2$ 号存储单元。这样就可把 n 个物理存储体构成的存储器看作一个大的逻辑上是一个体的存储器, 该存储器每个地址中存储着 $w \times n$ 位信息, 只要给定一个地址, 就可同时读出或写入 $w \times n$ 位信息。显然, 单位时间里, 存储器提供的信息, 同单体单字相比提高了 n 倍。但这种高速并非任何情况下都是有效的, 仅在向量一类运算的特定环境下, 或者指令都是顺序执行时, 访存速度才真正提高了 n 倍。如若遇到转移指令或数据在主存中不是连续存放时, 实际访存速度就会明显下降。所以这种主存系统实际上用的不多。

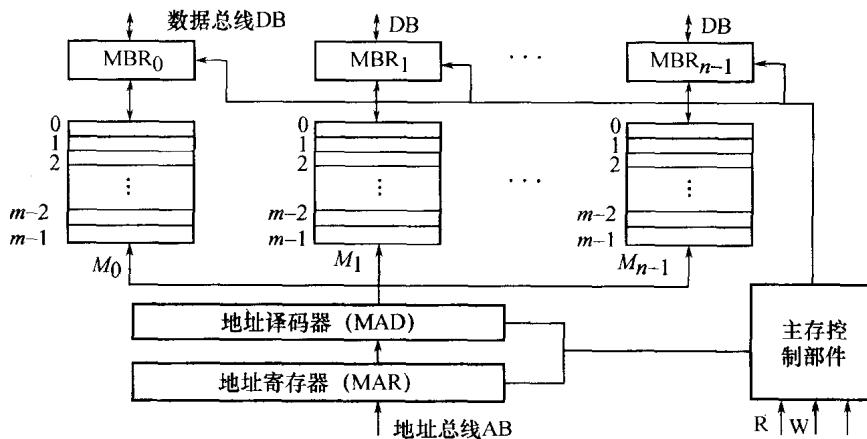


图 5.51 单体多字并行主存系统

2. 多体交叉编址并行主存系统

通常使用 n 个容量相同的存储体, 每个存储体都有自己的 MAR、MAD、MBR、时序和读写线路, 构成 n 个完全独立的存储器, 在存储控制器的控制与协调下, 实现重叠并行存取。称这样的主存系统, 为多体交叉编址并行主存系统。

(1) 两种交叉编址方式

如图 5.52 所示分别为 M_0, M_1, \dots, M_{n-1} 的 n 个存储体, 每个体包含 m 个存储单元的多体交叉编址并行主存系统。编址方式有低位交叉和高位交叉两种。所谓交叉编址, 就是将通常被称为系统地址的一套统一的地址, 按顺序交叉地分配给各个存储体。图中, 每一个 M_i ($i = 0, 1, \dots, n - 1$) 中间的序号为体内地址, 左侧的序号为低位交叉编址时的系统地址, 右侧的序号为高位交叉编址时的系统地址。高位交叉编址时, 系统地址的连续空间落在同一存储体内, 容易发生访存冲突, 并行存取的可能性很小, 所以目前只用于非共享主存(即每个处理机仅享用统一编址主存的部分连续地址空间)和专用 Cache 的多处理机系统中。所谓访存冲突, 就是同时有两个或两个以上访存地址指向同一存储体, 这时, 不能同时进行访存。发生访存冲突, B_m 将减小。低位交叉编址时, 系统地址在同一存储体中不是连续的, 而是以 n

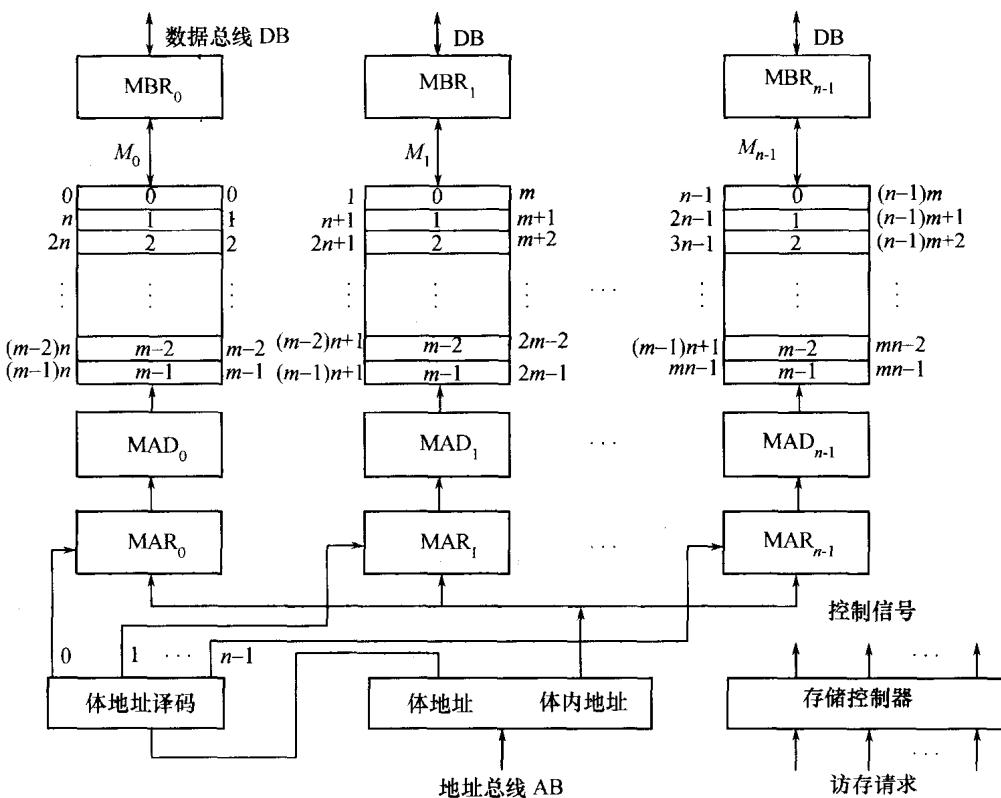


图 5.52 多体交叉编址并行主存系统

为模交叉编址的。所以连续的程序或数据将交叉的存放在 n 个存储体中, 可实现以 n 为模的交叉并行存取, 访存冲突的概率就会变得很小。为充分发挥并行性, 大多数计算机都采用低位交叉编址方式。

(2) 多体低位交叉并行主存系统

假定 A 为系统地址, n 为存储体个数, j 为体号 ($j = 0, 1, \dots, n - 1$), m 为每个存储体的存储单元个数, i 为体内地址 ($i = 0, 1, \dots, m - 1$) 则有

$$A = n \times i + j$$

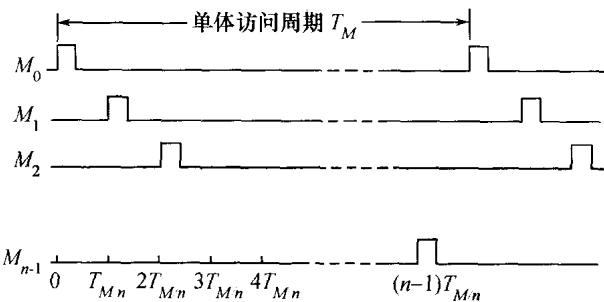
式中 $i = [A/n]$, $j = A \bmod n$ 。系统地址码位数为 $\lceil \log_2^n + \log_2^m \rceil$, 体号地址占低 $\lceil \log_2^n \rceil$, 体内地址占高 $\lceil \log_2^m \rceil$ 位。若 n 为 4, 则编址情况如表 5.14 所示。低位交叉编址的目的是为了便于各存储体同时并行工作。以 4 体并行为例, 假设有 4 条顺序执行的指令, 每条指令占一个存储单元。系统地址分别为 0、5、A、F, 恰好分布在四个存储体上。由于四个体都能独立存取, 虽然地址并不连续, 但四个地址寄存器可分别同时接收它们, 可在一个存储周期中并行读出四条指令。可见, 多体低位交叉编址主存和单体多字并行主存虽然最大频宽相同, 但多体可更灵活地设置地址, 只要信息在不同的存储体中, 就能得到比单体更大的实际频宽。

但由于程序经常出现转移,数据的分布也是随机的,故实际访问的系统地址,不见得都均匀分布在各存储体上,而使得实际频宽约为最大频宽的 1/3 左右。

表 5.14 4 体低位交叉编址表

存储体	址编址	地址码低 2 位
M_0	$0, 4, 8 \dots 4i + 0, \dots 4(n - 1) + 0$	00
M_1	$1, 5, 9 \dots 4i + 1, \dots 4(n - 1) + 1$	01
M_2	$2, 6, 10 \dots 4i + 2, \dots 4(n - 1) + 2$	10
M_3	$3, 7, 11 \dots 4i + 3, \dots 4(n - 1) + 3$	11

主存通过总线与 CPU 交换信息的宽度一般与存储体字长相同。多体低位交叉编址主存为在一个存储周期访问 n 个存储单元,目前广泛采用的是分时访问法。假定存储周期为 T_M , 所谓分时访问,即每隔 T_M/n 时间启动一个存储体, n 个存储体以 T_M/n 的时间间隔进入并行工作状态,如图 5.53 所示。若 T_M 为 400 ns, 则当 n 体并行分时工作时,在 400 ns 内,CPU 依次等间隔发出 n 个访存命令,对每个存储体来说,存储周期仍为 400 ns,但对 CPU 来说,相当于每 T_M/n 时间访问一次主存。对主存系统来说,地址流以 T_M/n 速度流入,信息流也以同样速度流入或流出。

图 5.53 n 体并行分时工作

另一种并行存取的处理方式是同时启动 n 个存储体,使 n 个存储体同时被访问。通过分时使用总线对外传送信息或接收要写入的信息。

为了使多体并行主存有条不紊的工作,需要一套存储控制逻辑,简称存控,其比单个存储体的控制逻辑要复杂得多。存控的主要功能是:接收 CPU 和 I/O 设备的访存请求,并按预定的请求源的优先顺序进行排队,响应优先级高的请求源;判各存储体忙闲,接收各请求源发出的访存地址并进行转换送相应的存储体;产生各存储体所需的读写时序;接收和发送信息等。为了实现上述功能,存控应为每个存储体设置一个忙闲触发器,用以判断存储体忙闲,并进行相应控制。设置排队线路,以便区分访存请求源的轻重缓急,选出优先请求源。当优先请求源要求访问的存储体正处于工作状态无法接受访问时,则暂时取消该访问请求,

并保存该请求,一旦该存储体闲置,立即响应该访存请求。与此同时,分配优先级稍低的请求源访问其他存储体。此外还需设置与系统总线连接的数据接收和发送通路。

目前几乎所有计算机都采用多体低位交叉编址并行主存,甚至微型计算机也不例外。这就大大提高了主存频宽,从而提高了计算机系统的性能。并行主存的实际频宽总是小于最大频宽,换句话说,访存冲突总是存在的。究其原因,除了程序不总是顺序执行和数据随机存放之外,还与存储体个数一般为 2 的整数次幂有关。可以证明当存储体个数 n 取 5 以上的素数时,访存冲突将大大减小,且 n 取的素数越大,访存冲突越少。因此称采用素数个存储体的低位交叉并行主存为无访问冲突并行主存。其实际频宽接近于最大频宽。YH-1 巨型机采用的则是 31 个存储体构成的无冲突并行主存结构。无冲突并行主存,因存储体个数为素数,由系统地址变成体号地址和体内地址是很复杂的,实现较为困难。有兴趣读者请参阅文献[10]。

5.7.2 存储系统及其层次结构

首先介绍两个概念,存储系统(Memory System)是计算机中用于存储程序及其要加工处理信息的各类存储器构成存储系统;存储层次(Memory Hierarchy)是存储系统中的各类存储器通过硬件和软件有机地组成统一的整体,自动的向程序员提供足够大的存储空间,最大限度地与 CPU 速度相匹配。这样的存储系统称为存储层次,也称为存储体系。

1. 两级存储系统

早期计算机存储系统被分成两级,即主存和辅存,如图 5.54 所示。主存属于主机范畴,CPU 可直接访问,故又称为内存。辅存属于外围设备,它存储的信息只有先送往主存,然后 CPU 才能作用,故称它为外存。目前主存通常采用半导体存储器,速度快、容量不大、且成本较高。其存取方式可为随机存取和只读,容量通常为几 MB ~ 几百

MB,速度在十几 ~ 几百 ns 之间。辅存通常由磁表面存储器和光盘存储器构成,速度慢、容量大、成本低。辅存又可分为联机辅存和脱机辅存。脱机辅存又称后援辅存、海量辅存,其存储容量可以说是无限的,速度比联机辅存慢一个数量级,多由磁带库、光盘库和可换大容量磁盘组成。联机辅存主要采用硬磁盘存储器,目前每台容量在几十 ~ 几百 GB 之间,以扇区为单位存取,速度在几 ~ 十几 ms 之间。

两级存储系统可把要存储的信息分成两类,即“活跃”信息和“待命”信息。所谓“活跃”信息是指经常被 CPU 使用或实时性要求很高的程序和数据,例如管理计算机的操作系统、当前 CPU 正在运行的用户程序及其要处理的数据等。所谓“待命”信息是指在相当时间范围内不会被使用的程序及与之有关的数据。主存容量因受速度和成本的限制不可能很大,主要用于存储“活跃”的信息,以便 CPU 快速从主存中获得所需信息,从而提高整个计算机

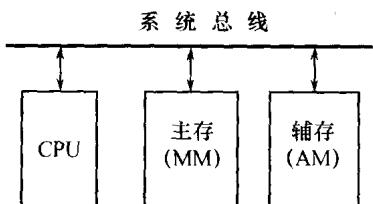


图 5.54 两级存储系统

系统的工作效率。而“待命”信息则存储在辅存中，只在需要时才调入主存。

为了能调用和执行辅存里的程序和数据，通常利用程序复盖技术实现主存、辅存之间的信息传送。因此程序员需要预先把程序及数据分成很多块（程序块和数据块）并存储到辅存中，块的大小由主存容量确定。然后根据程序的运行情况在程序员的干预下，不断地把这些块装入主存，覆盖或者调出已在主存，但目前没有使用价值的程序块或数据块，维持整个程序的运行。两级存储系统通过程序员不断地对主存进行动态再分配，使得在较小的主存空间中运行较大存储空间的程序。这样计算机处理的信息得到比较合理的分配，较充分地发挥了CPU的高速性能。

2. 三级存储系统

两级存储系统只是主存、辅存简单的组合，主要存在两个缺点。首先，虽然辅存扩大了信息存储的场地，但如何合理准确划分程序块和数据块，在哪一时刻调入哪一程序块或数据块，覆盖哪一片主存区域，计算机不能自动进行，而需要由程序员编程时规划、设计。所以程序员除设计解决某问题的程序外，还要花费大量的时间和精力熟悉硬件以便设计主、辅存信息交换的程序。这就增加了编程难度，延长了程序调试周期，对程序员也提出了更高要求。其次，两级存储系统并没有针对CPU与主存速度的差距，解决主存速度与CPU速度的匹配问题。目前CPU速度比主存速度至少高一个数量级，CPU高速效能没有得到充分发挥。因此，存储系统的设计必须突破主存、辅存简单组合的模式，从系统结构上采取措施，采用速度不同、价格不同、存储技术不同的多种存储器件，按存储层次组成存储系统。各存储层次间通过硬件和软件有机地组合成统一的整体，使之无需程序员干预而由计算机自动实现调度的情况下，向程序员提供足够大的存储空间，同时最大限度地与CPU速度相匹配。

随着操作系统和硬件技术的发展，应用程序员完全摆脱了主存-辅存之间地址定位这一复杂工作，而由辅助的软硬件自动实现，使主、辅存成为统一的整体，构成主存-辅存存储层次，如图5.55所示。从整体上看，该层次速度接近于主存，容量和每位价格接近于辅存。主、辅存存储层次对应用程序员是透明的，解决了存储器大容量和低成本之间的矛盾。

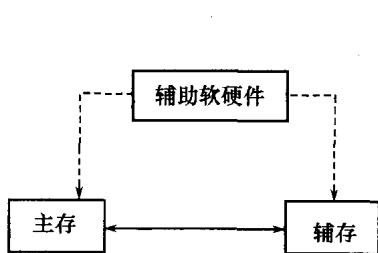


图 5.55 主存_辅存存储层次

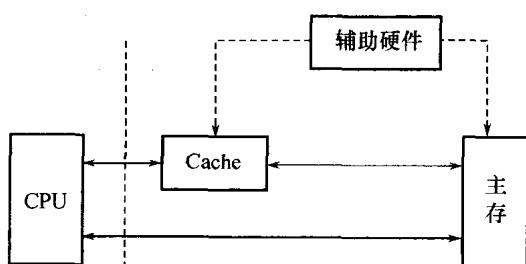


图 5.56 Cache_主存存储层次

为解决主存和CPU速度的矛盾。充分发挥CPU的高速度潜力，借鉴主存辅存存储层次的思想，在CPU和主存之间增加一级速度同CPU匹配、容量比主存小、位价格比主存稍高的

Cache。完全借助于辅助硬件把 Cache 和主存构成一个整体,组成 Cache-主存存储层次,如图 5.56 所示。从整体上看,该层次具有接近于 Cache 的速度、主存的容量和主存的平均位价格,解决了存储器高速度和低成本之间的矛盾。这个层次工作时完全由辅助硬件实现,对所有程序员都是透明的。

在现代计算机中,为提高计算机系统的存储容量和存取速度,提出采用半导体存储器、磁表面存储器、光盘存储器构建具有存储层次的存储系统。大多数同时采用上述两种存储层次,构成 Cache、主存、辅存三级存储层次,如图 5.57 所示。有的甚至采用多级存储层次,如图 5.58 所示。图中设有 L_1 、 L_2 、 L_3 三个 Cache。 L_1 为 CPU 芯片内部的 Cache,由双极型 RAM 构成,称一级 Cache。通过 CPU 内部的总线与 CPU 交换信息,不需要通过系统总线,因此大大提高了 CPU 的高性能。 L_2 在 CPU 芯片之外,通常由 SRAM 构成,通过系统总线和 L_1 或 CPU 交换信息,速度要慢些,称二级 Cache。 L_1 、 L_2 和 CPU 都是通过硬件直接连接的,对所有程序员都是透明的。 L_3 在主存内部或在外存的控制器中,作为辅存的高速缓冲存储器,其将大大弥补和缓解辅存与主存之间高达 10^5 倍的速度差距。

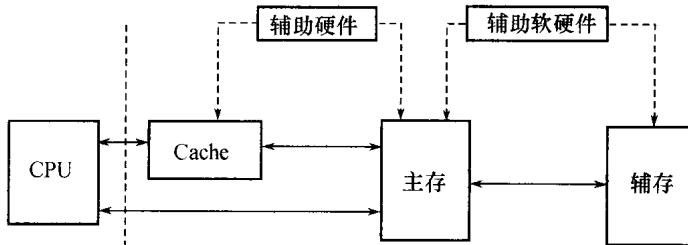


图 5.57 三级存储层次

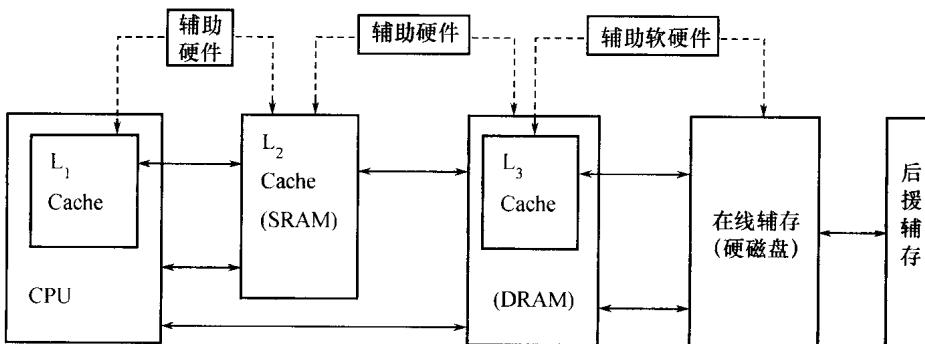


图 5.58 多级存储层次示意

多级存储层次将不同类型的存储器有机的结合在一起,层次中各类存储器借助于辅助的软件和硬件,通过合适的算法,实现层次间各类存储器信息的调入和调出,使之成为高速度、大容量和低成本的存储系统。

小 结

存储器不仅是计算机自动工作的关键部件,还是影响计算机速度的重要部件之一。本章从计算机硬件组成部件级的角度首先阐述存储器的功能、分类和性能参数。其次讲授半导体存储器 RAM、ROM 和 CAM 的基本概念、基本组成、基本工作原理和工作过程;重点阐述半导体存储芯片的外特性以及如何应用它们设计内存储器的方法和步骤;以磁盘为重点,阐述作为外存的磁表面存储器和光盘存储器的有关概念、性能参数、介质特性、读写原理,尤其是磁盘的基本组成、信息存储方式和地址形式。最后讲授了为解决主存、CPU 速度匹配这个计算机瓶颈的问题,构建并行主存系统和存储层次的有关概念、基本思想和实现的基本方法。

最后提请读者注意三个问题:一是半导体存储器的存储位元和存储矩阵结构仅要求读者了解其功能。具体细节将在《存储与外设》课程讲授。二是应当关注其他存储器件的应用与发展。目前虽然半导体器件广泛用来制造内存储器,但应关注有望取代它作为内存储器的新型器件,如铁电电容和铁电介质的场效应管存储器件。它们共同的优点是读写速度仅几个 ns,所需电压低,集成度高和非挥发性存储。铁电电容的缺点是使用寿命不如半导体器件长,铁电电容只能读写 10^{12} 次,半导体存储器件却能允许读写 10^{15} 次。铁电介质的场效应管制造的存储位元可靠性不如半导体器件高。两个问题只要解决一个,铁电存储器件将取代半导体器件制造内存储器。三是有关存储层次的具体实现和性能评测,尤其是 Cache 存储器的基本组成、工作原理和块调度算法,以及 Cache、主存层次地址映像和变换等内容。在《计算机体系结构》教材中将会深入的阐述和讨论。

习 题

5.1 解释下列术语

存储位元	存储单元	挥发性存储器	非挥发性存储器	RAM
ROM	CAM	存储周期	主存存取时间	DRAM
SRAM	刷新周期	刷新操作周期	柱面	磁头
磁盘格式化	扇区	磁光盘	磁盘数据传输率	主存频宽
可擦写型光盘	相变形光盘	存储层次	访存冲突	

5.2 存储器有哪几种分类方法? 它们是怎样分类的?

5.3 作为存储元件的器件应有何特征?

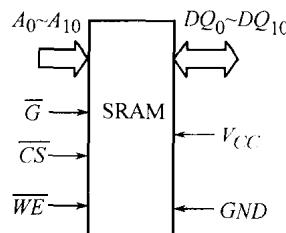
5.4 内存储器有哪两种选址方法? 它们是怎样实现选址的?

5.5 分别绘出 4 字 \times 4 位的线选法和重合法存储矩阵示意图。

5.6 说明如下存储芯片引脚的含义: A_2 、 D_7 、 Q_5 、 O_4 、 I_0 、 DQ_3 、 IO_6 、 GND 、 NC 、 V_{CC} 、 V_{PP} 、 \overline{CS} 、 \overline{WE} 、 \overline{CE} 、 \overline{OE} 、 \overline{G} 、 \overline{W} 、 \overline{F} 、 RAS 、 CAS 、 PGM 。

5.7 试从速度、集成度、功耗、引脚等方面说明 BiRAM、SRAM 和 DRAM 的异同点。

- 5.8 某 DRAM 存储芯片,其字位结构为 $1M \times 1$ 位,试问其地址、数据引脚各是多少个?应设置哪些控制引脚?
- 5.9 某 SRAM 存储芯片,其字位结构为 $512K \times 8$ 位,试问其地址、数据引脚各是多少个?应设置哪些控制引脚?
- 5.10 试述 CAM 基本结构组成、应用场合及优缺点。
- 5.11 试说明 RAM、ROM 和 CAM 的异同点。
- 5.12 试述熔断丝型 PROM 和肖特基二极管型 PROM 的工作原理。
- 5.13 试从速度、集成度、擦除与编程方法、应用等方面说明 MROM、PROM、EPROM 和 EEPROM 的异同点。
- 5.14 主存储器设计一般应分成哪三个阶段?各阶段都进行哪些工作?
- 5.15 用 BiRAM 存储芯片 MCM10146 和 MECL10K 系列门电路组成容量为 $8K \times 48$ 位的 Cache,假定 MECL10K 系列每个门电路输出只能带 8 个负载端。要求:
- (1) 计算要多少片 MCM10146;
 - (2) 画出构成 Cache 的存储矩阵图;
 - (3) 分别计算驱动 A_i 、 D_{in} 、 \overline{CS} 、 \overline{WE} 所需门数。
- 5.16 已知某存储器存储容量为 4 MB,当分别按字节、按 16 位字长、32 位字长和 64 位字长编址时,地址线(地址码位数)分别为多少根(位)?数据线分别为多少根?寻址范围分别为多大?
- 5.17 已知某 32 位字长的计算机主存采用字编址的半导体存储器,其地址线为 20 根,使用 $32K \times 8$ 位的 SRAM 存储芯片组成该机最大的存储空间。试问该存储空间的存储容量为多少字节?需多少 SRAM 存储芯片?哪几位地址作为字扩展去控制 CS ?
- 5.18 用 $4K \times 4$ 位的 SRAM 存储芯片构成具有 16 根地址线,字长 16 位的存储器。试回答如下问题:
- (1) 画出存储芯片引脚示意图;
 - (2) 该存储器的存储容量为多少 KB?构成该存储器需多少存储芯片?
 - (3) 画出片选译码和存储矩阵示意图。
- 5.19 外围线路采用 TTL 电路,使用 $256K \times 4$ 位的 DRAM 存储芯片 MB81C4256,构成 $4M \times 64$ 位的主要存储器。试回答:
- (1) 画出存储芯片引脚示意图。
 - (2) 需要多少个 MB81C4256 存储芯片?
 - (3) 存储器地址码位数?作为片选译码的地址码位数?
 - (4) 计算存储器的 \overline{WE} (读写控制)端需多少 TTL 门电路驱动?假定一个 TTL 门电路可驱动 8 个 \overline{WE} 端。
- 5.20 已知 SRAM 存储芯片引脚示意如下图:



DQ_i :数据输出输入端; $A_0 \sim A_{10}$:地址输入端; \overline{CS} :片选端; \overline{G} :输出输入控制; \overline{WE} :读写控制端。试回答:

(1) 依地址和数据端的数量计算存储芯片容量。

(2) 试用该芯片构成 $16\text{ K} \times 32$ 位的存储器：

① 计算需用多少片 SRAM 存储芯片？

② 存储器需多少位地址码？片选应用哪几位地址码译码产生？

③ 假定一个门电路能驱动 8 个 \overline{WE} 负载，求该存储器的 \overline{WE} 所需驱动门数。

5.21 用 $64\text{ K} \times 1$ 位的 DRAM 存储芯片，构成以字节编址的 $4\text{ M} \times 8$ 位主存，并将它分装在存储容量为 $1\text{ M} \times 8$ 位的存储板上。试回答如下问题：

(1) 需多少存储芯片？分装在几块板上？

(2) 主存所需地址位数？板选、片选所需地址位数？画出板选、片选逻辑示意图。

(3) 计算 \overline{WE} 负载端数？若每个门电路能驱动 8 个 \overline{WE} 负载，需多少个门电路？

(4) 每块存储板至少需多少引脚？若为 SRAM 芯片，每块板至少需多少引脚？

5.22 何谓 DRAM 刷新？DRAM 为什么要刷新？DRAM 刷新的要求是什么？

5.23 按刷新操作周期分配方式分类，有哪几种刷新方法？各自优缺点怎样？

5.24 磁记录材料和磁头材料各应有何特点？

5.25 什么是矩磁材料？什么是软磁材料？各有何特点？在磁记录中，各应用于什么场合？为什么？

5.26 磁头有几种分类方法？它们是怎样分类的？

5.27 磁头工作间隙有何作用？间隙宽度以及磁头和介质表面之间的距离对记录性能有何影响？为什么？

5.28 从磁头和磁介质考虑，如何提高记录密度？

5.29 硬磁盘驱动器如何分类？各有何特点？

5.30 何谓温彻斯特技术？采用该技术制造磁盘有何好处？

5.31 移动磁头固定盘驱动器组成及各组成部分的功能？

5.32 磁盘存储器地址由哪几部分构成？各部分的意义是什么？

5.33 磁盘的存储容量由哪些因素确定？分析提高数据传输率和缩小存取时间的途径。

5.34 磁盘和内存交换信息时，为什么一般不跨柱面交换？

5.35 有一单片盘驱动器，每面可记录 3000 道，道容量为 125000 位，转速为 7200RPM，求其平均等待时间、数据传输率和存储容量。

5.36 磁盘机的盘组有 20 个记录数据的盘面，盘直径为 12 英寸，其中 3.5 英寸区域用于记录信息，假定盘外侧空白区域为 0.3 英寸。其记录密度为 2500TPI 和 127000 bpi，磁盘转速为 5400RPM，假定 $\pi = 3$ ，计算盘组容量和数据传输率。

5.37 磁盘机的盘组由 6 个盘片组成，其专设 1 个伺服面，其他盘面为记录数据的盘面。转速为 7200RPM，有效记录区域的外直径为 13.3 厘米，内直径为 6.3 厘米，记录密度为 6640bpm，磁道道距 P_i 为 0.01 毫米。假定 $\pi = 3$ ，试计算：

(1) 计算盘组存储容量。

(2) 数据传输率为每秒多少字节？

(3) 若主机字长 32 位，主存存取周期为 50 ns，以 DMA 方式和盘组交换信息，试计算信息交换所用存取周期数与主存正常工作周期数之比。

5.38 磁盘机的盘组由 11 个盘片组成，其中专设 1 个盘面为伺服面，其他盘面为记录数据的盘面。盘

存储区域内直径为 14.3 cm, 外直径为 30.3 cm, 道密度为 80TPM, 每道分为 96 个扇区, 每个扇区 4 K 字节, 磁盘转速为 3600RPM。试计算:

(1) 盘组容量 C_f 是多少字节?

(2) 平均等待时间是多少?

(3) 数据传输率是多少字节/秒?

5.39 磁盘机的盘组由 9 个盘片组成, 其中专设 1 个盘面为伺服面, 其他盘面为记录数据的盘面。盘存储区域内直径为 6.1 cm, 外直径为 12.9 cm, 道密度为 180TPM, 位密度为 5000 bpm, 平均寻道时间为 12 ms, 磁盘转速为 7200RPM。假定 $\pi = 3$, 试计算:

(1) 盘组容量 C_N 是多少字节?

(2) 数据传输率是多少字节/秒?

(3) 从任一磁道读取 20000 个字节数据的平均存取时间是多少?

5.40 磁盘机的盘组由 4 个盘片组成, 其中专设 1 个盘面为伺服面, 其他盘面为记录数据的盘面。盘存储区域内直径为 4.3 cm, 外直径为 8.9 cm, 道密度为 100TPM, 位密度为 5000 bpm, 磁盘转速为 7200RPM。假定 $\pi = 3$ 。试计算:

(1) 数据盘面数和柱面数;

(2) 盘组容量 C_n 是多少字节?

(3) 数据传输率是多少字节/秒?

(4) 假定系统配备上述磁盘机 12 台, 每个磁道分成 64 个扇区, 试为该磁盘系统设计一个地址方案。

5.41 现代磁盘传输率高、容量很大, 试述现代磁盘采用的新技术。

5.42 按存储介质光盘分为哪几类? 各类有何特点?

5.43 按存取方式光盘分为哪几类? 各类有何特点?

5.44 形变型光盘、相变型光盘和磁光盘各是怎样写入和读出信息的?

5.45 主存、Cache、通用寄存器、磁盘、光盘和磁带都可用来存储信息, 试按存取时间由快至慢排序。

5.46 多体低位交叉编址并行主存系统有何特征? 同单体多字并行主存系统比较, 有何优点?

5.47 提高存储器速度可采取哪些措施? 试简要说明。

5.48 为什么要把存储系统分成若干存储层次? 主要有哪两个层次? 它们各自解决什么问题?

5.49 现代计算机存储系统为什么采用层次结构? 试说明 Cache、主存和主存、辅存两个层次在功能上、技术上和实现方法上有何不同?

第六章 输入/输出(I/O)控制

引　　言

现代计算机系统的一个显著特点是外围设备的品种多,数量大。各类外设不但结构不同,性能迥异,而且与主机的连接和控制也复杂多变。但是若从外设与主机之间所进行的工作来分析,则都是也仅仅是交换信息。即或是外设向主机输入信息,或是主机向外设输出信息。对输入和输出操作进行硬件和软件的控制就是输入/输出控制即 I/O 控制。I/O 控制不但要使外设和主机联系起来,构成一个 I/O 子系统,而且要使 I/O 子系统具有高的吞吐能力和高的工作效率。实际上,I/O 子系统是影响计算机工作效率的又一个关键部件。

本章主要从硬件角度讨论 I/O 控制,从软件角度讨论 I/O 控制是“操作系统”课程的主要内容之一。本章首先简单介绍 I/O 设备的分类、作用与地位、I/O 设备和操作的特点及与主机的连接方式,其次介绍 I/O 控制及 I/O 子系统的基本概念,最后重点阐述中断及中断系统并深入讨论几种主要的 I/O 控制方式,它们的结构及工作原理。

6.1 外围设备简介

输入/输出设备简称 I/O 设备,和外存储器等一起属于外围设备的范畴。外围设备(Peripherals)或外部设备,简称外设,它是计算机系统中围绕主机设置的,与主机进行信息交换,并改变信息形态的装置。

当前的计算机系统,无论是巨型、大型、小型机,还是微型、便携、掌上机,都配置各种各样的外设。外设种类数量多,涉及技术广,性能差异大。对于计算机原理课程全面详尽地讲授所有外设的结构组成和工作原理,既无可能,也不必要。本节仅对外设的分类、作用与地位、I/O 设备和操作的特点及与主机的连接方式作简单介绍。

6.1.1 外设分类

外设品种多,可从不同角度讨论分类。一般按它们各自的功能,可分为输入设备、输出设备、外存储器、数/模转换设备、终端设备和数据通信设备。

1. 输入设备

将数据、程序等信息转换成计算机所能接收的形式送到计算机中去的设备称之为输入设备。它包括扫描输入装置、字符识别装置、图形输入装置、键盘和声音输入装置等。

2. 输出设备

将计算机处理的最终结果或中间结果以人所能识别的字符、图形、图像等形式表示出来的设备叫输出设备。它包括显示器、打印机、绘图机,以及缩微胶卷照相装置等。由于至今还没有一种输出手段能完全满足用户在速度、精度及功能上的种种要求,因此目前多种输出方式并存。打印机和显示器是应用最多的输出设备,绘图机、汉字输出设备及缩微胶卷装置的应用也日趋普及。

3. 外存储器

外存储器用来存放不直接与 CPU 发生关系的数据和程序,它可与主存直接成批地交换信息。磁表面存储器目前仍是外存的主角,除磁卡多用于特殊场合外,磁盘、磁带仍在各类计算机系统中大量使用。当然,光盘因其单片容量大也得到愈来愈广泛的应用。结构上属于半导体存储器范畴的快擦写存储器 Felash 构成的固态盘亦将是外存储器的重要成员。

4. 模/数转换设备

在过程控制计算机系统中,需将从控制对象采样得到的诸如温度、压力、速度、角度及位移等模拟量转换成数字量输入到计算机中。而经计算机处理后的控制信息又要转换成模拟量,再经执行机构对控制对象进行调节、控制。完成上述功能的装置叫模/数转换设备,简称 A/D 和 D/A 设备。

5. 网络通信及终端设备

计算机发展的热点之一是计算机网络。将各自用户的计算机通过网络通信设备与因特网(Internet)相连结,不但可与世界各地的大型数据库相通,查阅资料,遨游广阔的信息世界,而且可收发传真和电子邮件。目前,计算机与通信技术的结合,已经产生了无穷的威力。网络通信与互连设备有调制解调器、网卡、中继器、集线器、网桥、路由器和网关等。

调制解调器(MODEM)是计算机通信的基本设备。计算机向外发送信息时,通过 MODEM 将信号源来的数字信号进行数/模转换,用载波将信号加以调制,形成适合于信道需求的模拟信号送上通信线路。计算机从通信网络接收信息时,MODEM 对传来的载波混合信号进行解调,也就是形成数字信号,送入计算机。

网络互连设备和通信线路将各地的计算机形成全球性网络。常用网络互连设备有网络卡、集线器(Hub)、路由器(Router)、中继器(Repeater)、网桥(Bridge)和网关(Gateway)等。

用户用来与计算机通信的输入/输出设备叫终端设备。终端设备可分为通用和专用两大类,而通用终端又可分为简易终端和智能终端等。简易终端是仅有输入/输出功能而无数据处理能力的装置。例如,采用键盘显示器具有一般的人机对话功能和低速的字符输入/输

出能力,根据需要,也可配置打印机或图形输入/输出设备。

智能终端除可完成信息的输入/输出外,还具有一定的数据处理功能。它不但配有处理器,还有较丰富的管理软件及用户程序等。

专用终端是根据使用需求完成规定功能的终端,如订票终端、银行取款机、商场和交通场所的咨询设备等。它们大都配有键盘或触摸屏的显示器,通过人机对话方式完成订票、取款、咨询等单一的工作,一般不具备其他方面的功能。

6.1.2 外设的地位与作用

半个世纪以来,计算机经历了四代的发展历程。随着计算机性能日益提高,应用日益广泛,外设在整个系统中的作用与地位愈来愈重要。究竞外设在计算机系统中处于什么样的地位?有哪些作用呢?

1. 外设是计算机系统的重要组成部分

众所周知,作为冯·诺依曼计算机,输入和输出是系统的基本组成部件。毫无疑问,如果没有与主机配套的输入/输出设备。则计算机将无任何作为。随着计算机不断地更新换代,主机的速度愈来愈高,相应的软件愈来愈多,信息的交换与流量也愈来愈大,这就要求有性能更高、功能更强的众多外设与之相配,否则,速度、软件等均是一句空话。正因如此,人们已愈来愈清楚地认识到外设的重要性,如果说初期为追求主机的发展而对外设有所忽视,那么现今外设的研制与开发则与主机取得了同步的发展。各种类型、不同档次的外设在计算机系统中的造价比重愈来愈高。据统计,第一代计算机中,外设在整机中的造价比重不到5%,第二代机中迅速上升至30~40%。第三代机中已超过50%,而现在已达到80%以上。

2. 外设是人机对话的工具

计算机是人类的运算工具。程序、数据和命令要送入计算机,计算机的运算结果、运行状态要输送出来,都要通过外设来实现。因此,外设是人机对话的惟一通道。早期计算机中,人机通信主要借助电信专用设备——电传打字机。现在,利用键盘、鼠标和显示器等进行人机通信,直观、方便、高效,已成为人机联系不可缺少的手段。

3. 外设是完成数据媒体变换的装置

计算机的控制流与数据流都是以电信号表示的二进制代码,而人却习惯于用字符、图形、图像等表达信息的含义。因此人机的信息交换,必须完成两类不同信息表示方法的变换,这种变换只能由外设来完成。

4. 外设是系统软件及信息的驻在地

信息量的迅速膨胀,是当今信息社会的主要特征。随着计算机处理能力的增强,特别是多媒体技术的普及,系统软件、数据库、图形图像的存储量越来越大。设置庞大的内存不仅无此可能,也无必要。以磁盘和光盘为代表的外存储器作为信息的驻在地是必不可少的手

段。当今计算机系统,外存的有无、性能的优劣是衡量系统性能的重要标志之一。

5. 外设是计算机推广应用的桥梁

现今计算机的应用由数值计算已扩充到信息管理、自动控制和辅助设计等诸多领域。新型外设的出现为计算机的推广应用提供了可能。例如,在气象预报、大地测量和地震数据处理中,必须有各种图形图像处理设备。在商业、银行和交通部门,必须有磁卡、IC 卡、条形码阅读机,必须有带键盘或触摸屏的显示器。在医疗诊断、治疗和监护单位,必须有计算机断层扫描仪、心脏监护仪、智能直线加速器等。在办公自动化领域,则必须有各种打印机、复印机、传真机等。可见,计算机的普及应用促进了外设的发展,而新型外设的出现又拓宽了计算机在不同领域的推广应用。

6.1.3 外设与主机的连接方式

除外存储器在输入/输出设备与设备控制器之间必须加驱动器外,其他外设与主机的连接方式如图 6.1 所示。

本章输入/输出控制即 I/O 控制主要讲授的就是图 6.1 中的输入/输出系统接口。该系统接口将因设备的不同、主机的要求不同、I/O 的控制方式不同及与主机连接方式不同而存在很大差别。

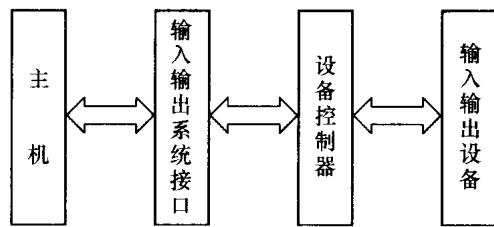


图 6.1 外设与主机的连接示意

6.2 I/O 控制的有关问题

I/O 子系统即输入/输出子系统,完成 I/O 操作的所有软件和硬件构成 I/O 子系统。对 I/O 操作的软件和硬件管理,称为 I/O 控制。I/O 控制即输入/输出控制,I/O 控制就是对 I/O 子系统的管理。其主要任务是:协调主机与 I/O 子系统之间的工作;将外界的请求转换成 I/O 子系统的基本操作;管理控制 I/O 子系统内部的信息传送和各种并行活动。因此,I/O 控制的基本功能是:控制外设的各种动作,如启动、停止、输纸、定位等;提供主机与外设之间的数据传送路径,组织数据交换;组织协调各外设分时享用传送信息的硬件和软件,从而控制主机与外设,外设与外设之间的并行工作;平衡外设与主机之间的数据流量,提供数据缓冲,使单位时间的数据流量不因过大而造成数据溢出与丢失,也不因过小而使设备得不到充分利用;向主机报告外设的状态,使主机合理安排程序转移;对外设产生的数据错误和数据传送中的错误,进行检测与校正;组织并实施对外设及 I/O 子系统的检查、诊断和维护。

I/O 控制不但与主机的性能及结构相关,更与 I/O 设备的特性和操作的特点相关。从计算机系统的角度来分析,I/O 设备有哪些特点,实现 I/O 控制的基本依据有哪些,I/O 控制应有什么功能和 I/O 控制应有哪些基本类型都是影响 I/O 控制的基本因素。本节先就上述

I/O 控制的有关基本概念予以阐述。

6.2.1 I/O 设备与 I/O 操作特点

I/O 设备和操作的特点直接影响 I/O 控制及 I/O 子系统的组织。为更好的对 I/O 操作进行控制、管理和设计高效的 I/O 控制及 I/O 子系统,先介绍 I/O 设备和 I/O 操作的特点。

1. I/O 设备的特点

外设或 I/O 设备与主机相比,有如下明显的特点:一是外设的速度慢。大多数的外设均是机电混合装置,工作速度是微秒级、毫秒级甚至更慢,而主机的操作速度则多是在纳秒级。二是外设的多样性及复杂性。各类外设广泛涉及到机、电、光、磁、声、自动控制等多种学科,传输速率差异大,传输方式也不相同。

要使各类外设能与主机构成一个有机整体,至少要做如下工作:一是完成匹配。它不但包括工作速度的匹配,以解决主机的高速与外设低速之间的矛盾;而且还包括代码及数据格式的转换,主机及外设之间的通信与联络和建立同步关系等。二是要提高子系统的性能。它不但包括软、硬件功能的分配,软、硬件组织的繁简及操作控制的难易程度,而且特别注重提高输入/输出的传输率,提高主机与外设、外设与外设并行工作的能力。

2. I/O 操作的特点

I/O 操作的特点集中反映在异步性、实时性及与设备的无关性。

(1) I/O 操作的异步性

由于外设的速度低,故相对于主机来说都是异步工作的。外设的操作一般虽不采用统一的中央时钟,但在某些时刻又要接受 CPU 的控制。当外设准备好,并要与 CPU 交换信息时,需要向 CPU 申请服务。这种申请对 CPU 而言是随机的,两次申请的时间间隔也可能很长,不同的外设更是各不相同。要提高计算机系统的效率,在连续两次外设与 CPU 交换信息之间,CPU 应能继续执行原程序。从而保证主机与外设之间,外设与外设之间实现并行工作,这必然造成 I/O 操作相对于主机工作是任意性与异步性的。

(2) I/O 操作的实时性

当外设申请服务时,主机需在规定的时间内中断现行程序而去处理这种请求,以便不失时机地输入或输出信息。要做到这一点,主机需具有与不同外设相适应的工作或传送方式,或处理单个字符的传送,或处理数据块的传送,否则就可能造成信息的丢失。因此,处理机需按外设的工作速度,全面考虑、统筹安排。以确定合适的工作方式及信息流量,这就是 I/O 操作的实时性。

(3) I/O 操作的实现与设备的无关性。

外设的品种繁多,性能迥异,与主机交换信息的工作方式又不尽相同,必然造成 I/O 控制组织的多样性及复杂性。但作为主机及 I/O 控制,希望能有统一的形式与外设连接。让外设的特殊性隐藏在各自的设备控制器及接口的某些可变部分中。这样,处理机无需了解

各特定外设的具体细节,而是统一的控制、调用外设。在更换设备时,也无需更改标准的接口及基本的控制软件。因此,要尽量使 I/O 操作的实现与设备的无关性。

6.2.2 四级 I/O 子系统和三级 I/O 子系统

既然 I/O 操作的异步性、实时性及其实现与设备的无关性是 I/O 操作的固有特性。因此,在进行 I/O 控制和设计 I/O 子系统时就必须周密考虑,统筹兼顾。自治控制、分类处理和层次结构是进行 I/O 子系统设计的基本原则与方法。

自治控制即将功能分散化。也就是说,要使输入/输出功能尽可能从 CPU 中分离出来,由专门的部件去完成。这样就减少对 CPU 的打扰,使之致力于高速运算处理操作。分类原则是根据不同性质的外设,特别是工作频率的高低,分类组织 I/O 控制。层次结构是将 I/O 子系统的功能按不同的层次分布,既可方便地采用程序控制,又能赋予硬件组成以必要的灵活性。通常的做法是将标准的操作及控制功能放在与主存及 CPU 相连的层次上,即系统级接口中;而将非标准的操作及控制功能放在与设备相连的层次上,即设备级接口中。

在中、大型计算机系统中,一般采用四级层次结构的 I/O 子系统。如图 6.2 所示。它由外围设备、设备控制器、I/O 通道和 CPU 的一部分构成。

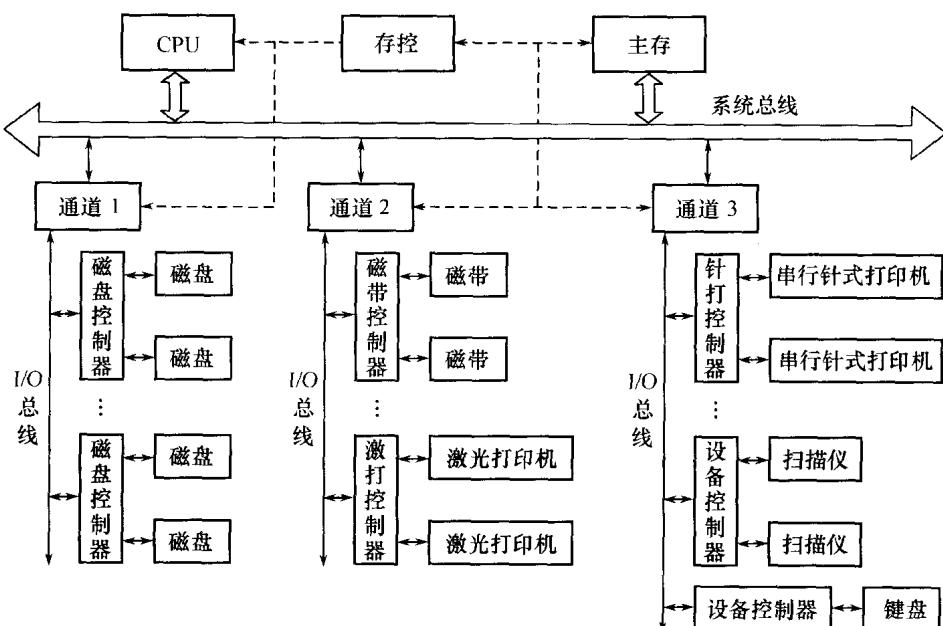


图 6.2 四级 I/O 子系统结构

CPU 的作用是执行 I/O 指令,即负责启动和查询指定的通道与外设,并处理来自 I/O 通道的请求。I/O 通道是 I/O 子系统的核心。在 CPU 启动通道后,它代替 CPU 调度、管理和控制连接于该通道的所有外设。设备控制器 (Device Control Unit) 是控制一个或多个外设完

成数据读写或其他所需操作的逻辑部件。不同设备的设备控制器虽不相同,但均以标准的方式与通道连接。通道发出的任一信号对各设备控制器是等价的。外围设备即指 I/O 设备、外存储器和终端设备等。随着计算机科学技术的发展,其含义已变得十分广泛,它可以是一台外设,也可以是另一台计算机或终端用户。

在小、微型计算机系统中,为了缩小体积,降低造价,一般采用 CPU、接口、外设的三级层次构,如图 6.3 所示。连接于系统的所有外设均通过相应的 I/O 接口与主机发生关系,接口是主机和外设之间的界面,是三级 I/O 子系统的核心。显然,四级结构与三级结构在功能上相似,而性能上不同。前者比后者独立性强,效率也高。通常称四级结构为独立型通道,称三级结构为结合型通道。

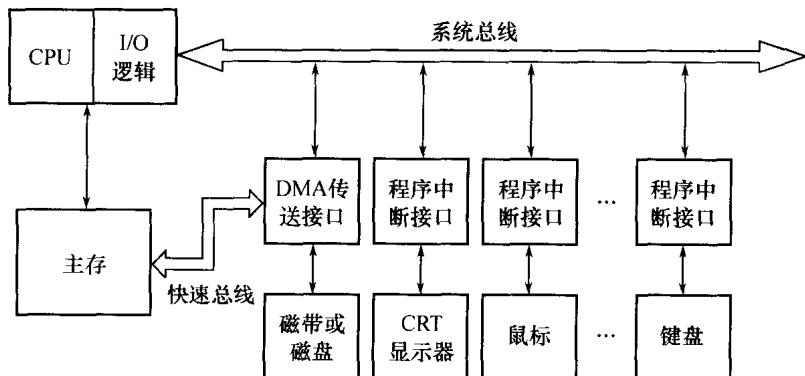


图 6.3 三级 I/O 子系统结构

6.2.3 I/O 控制的类型

按照 I/O 控制组织的演变顺序和外设与主机并行工作的程度,以及数据传送和控制的方式,I/O 控制一般可分为 5 种类型。

1. 程序直接控制方式

所谓程序直接控制(Programmed Direct Control,PDC)方式,顾名思义,就是完全通过程序来控制主机和外设之间的信息传送。程序直接控制方式可采用无条件传送和程序查询传送两种方式。由于数据的交换及控制均要通过 CPU,因此,它们均是一种以 CPU 为中心的系统结构。

2. 程序中断传送方式

借助于 I/O 中断控制数据传送就是程序中断传送(Program Interrupt Transfer,PIT)。在这种方式中,通常在程序中安排启动某一外设,然后机器继续执行其程序。当外设完成数据传送的准备后,便向 CPU 发出被称为中断请求(Interrupt Request)的信号。CPU 接到请求后,若允许停止正在运行的程序,则转而执行数据传送的中断服务程序,以完成外设和主机

之间的数据交换。交换完毕仍返回原来的程序继续执行。程序中断传送也是一种以 CPU 为中心的系统结构。

3. 直接内存存取方式

对于快速外设的成批信息交换通常采用直接访问内存的输入/输出控制方式,这就是直接内存存取(Direct Memory Access, DMA)。

这种方式的基本思想是在外设和内存之间开辟直接的数据交换通路。机器在正常工作时,所有的工作周期均用于执行 CPU 的程序。当外设将要输入或输出的数据准备好后,窃取或挪用一个访存周期。在这个周期内,外设和内存直接交换数据。这个周期过后,CPU 又继续执行原程序,因此,DMA 又被称为周期窃取或周期挪用方式(Cycle Stealing)。由于数据的交换及单个数据的传送控制均无需 CPU 干预,因此,DMA 是一种以内存为中心的系统结构。

4. I/O 通道控制方式

I/O 通道即 I/O Channel(I/O CH)方式是通过 I/O 通道这样一个独立部件来控制多台外设和主存直接交换信息。当执行程序中需要与某外设交换数据时,CPU 只要执行启动某一通道的 I/O 指令,然后继续执行原程序。而与外设交换数据的实际操作,则在通道的控制下,通过执行通道程序来完成。这样,CPU 的操作和通道的输入/输出操作可以并行。而且,通道还可以同时控制多台外设并行工作。因此,I/O 通道方式使得系统的效率得到进一步提高。和 DMA 方式一样,I/O 通道方式也是一种以主存为中心的系统结构。

5. 外围处理机方式

在现代超大型或巨型计算机系统中,CPU 的运算速度已高达数千万次甚至数万亿次。在这类机器中,如果还采用 I/O 通道来管理和控制外设,则势必影响系统的效率。为使 CPU 发挥高速运算、特别是向量运算的功能,在系统中通常设置外围处理机(Peripheral Processor Unit, PPU)来承担对外设的管理与控制。

这种 PPU 结构更接近一般的处理机,甚至就采用一般的小型或微型计算机。它既可完成 I/O 通道所要完成的 I/O 控制,还可完成码制变换、格式处理、数据块的检纠错等操作。有了 PPU,不但可简化外设的设备控制器,而且可用它作为维护、诊断、通信控制、系统工作显示和人机联系工具。

和 I/O 通道不同,PPU 基本上是独立于主处理机工作的,在大多数系统中,有的设置多台 PPU,分别承担 I/O 控制、通信、维护诊断等任务。这样,机器的系统结构则有了质的飞跃,系统由功能集中的集中式系统(Centralized System)发展为功能分散的分布式系统(Distributed System)。

6.2.4 I/O 接口

广而言之,接口(Interface)是两个不同系统的交接部分。例如,两种硬设备之间的接口

是一个由电子线路构成的逻辑部件;两个程序块之间的接口是一个接口程序等。连接主机和外部设备之间的接口则是输入/输出接口(I/O Interface),它指的是主机与外设之间设置的一个硬件电路及其相应的软件控制。若I/O接口采用标准化结构,即在结构尺寸、接插连接、信号电平、逻辑电路和传输总线等方面均采用统一的格式,称这样的I/O接口为标准化I/O接口(Standard I/O Interface)。由图6.1可知,不同的设备都有相应的设备控制器,而它们往往都是通过标准化I/O接口与主机连接起来。采用标准接口,不但可使主机的设计与外设无关,而且也可使外设的设计生产与主机无关。因此,采用标准I/O接口既有利于提高计算机系统的灵活性、可靠性和可扩性,也有利于外设产品的系列化,标准化与多样化。

1. I/O接口的功能

(1) 寻址功能

通常一台计算机配置多台外设,为了使用它们,主机对各外设进行编号,即分配给它们地址。接口应能接收地址信息,并对其译码,选择多台外设中的某一台,以及该接口中的某一端口。故接口中应设置选择线路。应当指出,接口和端口(Port)是两个不同的概念。端口是指接口中的某些寄存器,这些寄存器分别用来存放数据信息、控制信息和状态信息。它们相应的被称为数据端口、控制端口和状态端口,若干端口加上控制逻辑组成接口。通常,一台外设至少具有三个端口。主机通过I/O指令可从端口读出或向端口写入信息。

(2) 数据格式转换和电平变换功能

接口与主机之间通常以并行方式传送信息,而接口与外设之间,由于各设备的特性不同,有的采取并行传送,有的采取串行传送。因此,接口应能完成串、并转换。外设及其控制线路所需电源与主机可能不同,导致外设、主机和外设、外设信号电平存在差异,故接口应能完成信号电平的变换。

(3) 数据缓存和传送数据功能

由于接口处于主机与外设之间,所以数据必须通过接口才能实现在主机与外设之间的传送。这就要求接口具有数据通路并完成数据传送。因为主机与外设的数据存取速度存在很大差异,接口应具有数据缓冲能力。接口中一般都设有数据缓冲寄存器(Data Buffer Register,DBR),用它暂存外设、主机准备交换的信息。每个外设接口中的DBR的位数可以是不同的,其将因设备不同而异。

(4) 提供外设和接口状态的功能

主机要使用、控制外设,外设也要向主机提出各种请求,因此主机就必须能随时掌握外设和接口的状态。故接口中必须设置反应设备和接口状态的触发器。如忙闲触发器B(Busy)、完成(就绪)触发器D(Done)、中断触发器INTR(Interrupt)、屏蔽触发器MASK、数据传送错触发器、设备故障触发器等。接口中所有状态触发器组成状态寄存器。

(5) 实现主机对外设的通信和控制功能

主机向外设发出各种命令信息,外设向主机提出各种请求或回送必要的信息。如设备

和端口的选择信息；启动外设和命令外设读写、寻道等控制信息；中断或 DMA 的请求与响应等信息。故接口应有控制和通信功能。一般接口应设有存放主机命令的控制寄存器及其译码器和通信回送逻辑。

上述五种功能是接口的基本功能。在微型计算机中，实现这些功能的电路目前都制作在一个芯片内，称其为通用接口芯片。

2. I/O 接口的分类

(1) 按接口与设备之间的传输宽度分类

① 并行接口。主机与接口、接口与外设之间都是以并行方式传送信息，即每次传送一个字或一个字节的全部代码。并行接口的数据通路是按字或字节设置的。通常当外设本身是并行工作且主机与外设之间距离较近时，采用并行接口。

② 串行接口。接口与主机之间按并行方式传送信息，而接口与外设之间则按每次传送一位的方式实现信息的传送。因此，串行接口必须能实现信息的串、并转换，同时接口中应设置同步定时脉冲来控制信息的传送速率，保证接口与外设之间实现同步串行传送。串行接口用于串行工作设备（如鼠标或计算机网络的远程终端设备）与计算机的连接。

(2) 按主机与外设通信控制方式分类

① 同步接口。主机与接口、接口与外设之间信息传送都由统一的时钟控制。这种接口控制简单，要求接口与同步总线连接，信息传送与 CPU 时钟同步，即信息传送时间取时钟的整数倍。

② 异步接口。主机与接口、接口与外设之间信息传送无统一的时钟控制，而是采用应答方式。通常将要进行信息交换的两部件分别称为主设备和从设备，如将 CPU 称为主设备，外设称为从设备，或者反之。信息交换时，由主设备发出“请求”信号，经接口传送给从设备，从设备完成主设备指定的操作后向主设备发出“回答”信号，按一问一答分步完成信息的交换。从“请求”到“回答”之间的时间由操作所需的实际时间决定，与统一的时钟无关。

(3) 按信息传送的控制方式分类

① 程序直接控制传送接口。主机与外设之间的信息传送采用程序直接传送方式，为此配置的接口，称之为程序直接控制传送接口。这种接口是最简单的。

② 程序中断控制传送接口。主机与外设之间的信息传送采用程序中断传送方式，则在程序直接控制传送接口的基础上，增加相应的中断系统所需逻辑，这样的接口称之为程序中断控制传送接口。

③ DMA 接口。主机与高速外设之间的信息传送采用 DMA 传送方式，是在程序中断控制传送接口的基础上，还应增加 DMA 请求、响应和控制的逻辑，这样的接口称之为 DMA 接口。

④ I/O 通道。主机与外设之间的信息传送采用 I/O 通道控制方式，I/O 通道是一个独立的部件，它是四级 I/O 子系统的核心。其结构比 DMA 接口更复杂，与 CPU 的并行度更高。

(4) 按接口的连线方式分类

① 硬线连接并行接口。这种接口的工作方式及功能已被硬线连接设定,不能用编程的方法加以改变。因此,其灵活性差,且功能单一,设计简单。

② 可编程并行接口。这种接口除具有硬线连接接口的性能外,最主要的特点是可编程,因而具有选择性。其可以在不改变硬件连接的情况下,仅通过软件编程来改变接口的功能,因此方便灵活,功能很强。Intel 8255、8237、8279、82380、82380SL 等接口芯片是目前应用较广的可编程接口芯片。它们的芯片结构、引脚功能、工作方式和接口方法将在“微型计算机及应用”等课程中讲述。

3. 影响 I/O 接口结构组成的主要因素

在外设与主机连接时,下述因素将影响 I/O 接口的结构与组成:

(1) I/O 控制的类型

在通常的程序直接控制传送、程序中断传送、DMA、I/O 通道和 I/O 处理机 5 种方式中,选择类型的不同,I/O 接口构成将不相同,且其构成的硬、软件将依次变得复杂。

(2) 数据传送宽度

选择串行传送或并行传送,不但数据缓冲寄存器的结构不同,而且还会影响相应的控制逻辑。

(3) 通信控制方式

在外设与主机之间数据传送采用何种通信控制方式,也将影响 I/O 接口组成。通常有两种方式:同步通信控制在发送端与接收端之间设有统一的时钟;异步通信控制是采用应答控制方式,在发送端与接收端之间无统一的时钟。

(4) 传送信息的种类

通过 I/O 接口传送信息的种类将因不同外设而不同,有的外设多,有的外设少。显然,它必将影响 I/O 接口的组成。通常传送的信息主要有设备地址、数据信息、设备状态信息及控制信息。有的设备还需提供设备内部地址信息,如磁盘的柱面、盘面和扇区号;光盘的分、秒和段等。

4. I/O 接口的结构组成

由 I/O 接口的功能可推知 I/O 接口的基本结构组成如图 6.4 所示。

数据线是外设与主机之间数据信息的传送线,其根数通常为存储字长的位数或字节数,一般是双向的。地址线是传送设备地址(码)的,其根数由 I/O 指令中设备地址码的位数决定,它通常是一组单向线。命令线用于传送 CPU 向设备发出的命令信号,如读、写、启停等,其根数与命令多少有关,它是一组单向线。状态线是将外设和接口的状态向主机报告的信号线,其根数与状态的多少和与 CPU 的连接方式有关。它也是一组单向线。

数据缓冲寄存器 DBR:用于暂存外设与主机要交换的信息,其与数据线相连。

设备状态寄存器 DSR:用于存放外设和接口的状态信息,其与状态线相连。

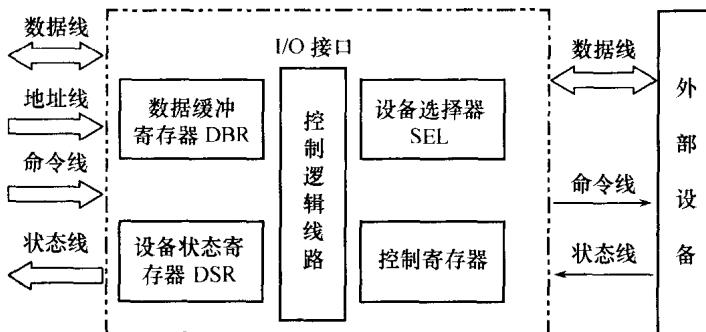


图 6.4 I/O 接口基本组成示意

设备选择器 SEL:接收地址线送来的设备地址,以便选中相应的设备。

控制寄存器:亦称命令寄存器,用来寄存 I/O 指令中的命令码。

控制逻辑线路:接口中所有控制线路的集合,控制实现外设与主机的信息传送。

6.2.5 I/O 指令

CPU 和 I/O 设备之间的数据,以及其他信息的传送均要通过执行输入/输出指令即 I/O 指令来实现。I/O 指令即 CPU 控制、使用外设的指令,它是计算机指令系统的一部分。I/O 指令由 CPU 发出并控制执行。

1. I/O 编址

I/O 编址指的是计算机系统给其所连接的外设分配设备码,即对外设编号。以便主机通过该编号控制、使用它们。因设备码又可称为地址码,所以称给外设分配地址码为 I/O 编址。对大、中型计算机系统,通常一台外设只给一个编号;而对小、微型机,一台外设所分配的地址个数,将依该外设及其接口中允许主机访问的寄存器的多少而定。通常 I/O 编址有两种方法。

(1) 统一编址

该方法把 I/O 操作和存储器读写同等对待,在主存地址空间中划出部分地址空间作为外设地址。即将外设及其接口中允许主机访问的寄存器和存储器的存储单元一样对待,统一编址,这样就可用访存指令去访问外设的寄存器。不需要专门的 I/O 指令组,使输入/输出程序设计十分灵活。缺点是外设占据了主存的地址空间,使之成为专用,非输入/输出程序设计不能再使用。加之外设地址和主存地址一样长,使得外设的地址选择电路也变得复杂了。VAX 系列计算机采用统一编址。

(2) 独立编址

该方法把 I/O 操作和存储器读写截然分开,为外设开辟独立的地址空间,存储单元地址和外设地址毫无关系。独立编址需专设 I/O 指令。独立编址,外设的地址选择电路简单,但

需专门的 I/O 控制信号, 指令系统变的复杂了。大多数计算机采用独立编址, 如 Intel 系列微机和 YH 系列巨型机等。

2. I/O 指令

I/O 指令分为通道 I/O 指令和非通道 I/O 指令两类, 通道 I/O 指令为四级 I/O 子系统设置, 非通道 I/O 指令为三级 I/O 子系统设置。四级 I/O 子系统中的外设是通过 I/O 通道与 CPU 连接的, CPU 只需通过通道 I/O 指令查询通道、外设和启停通道, 通道通过执行通道程序完成数据传送, 因此四级 I/O 子系统对 CPU 的打扰很小。三级 I/O 子系统中的外设是通过 I/O 接口与 CPU 连接的, CPU 借助于非通道 I/O 指令通过 I/O 接口实现对外设的管理、控制和使用, 三级 I/O 子系统对 CPU 的打扰较多。通道 I/O 指令和非通道 I/O 指令结构是有差别的。

(1) 通道 I/O 指令

通道 I/O 指令的格式如下:

操作码	Reg	通道地址	设备地址	首控制字地址
-----	-----	------	------	--------

操作码: 指出执行什么操作, 通常其操作为查询通道、外设状态或启动、停止通道。

Reg: 指出与操作有关的 CPU 寄存器。如操作为查询通道、外设状态, 则该 CPU 寄存器用来存放通道和外设的状态信息。

通道地址: 给出通道的地址码即通道的编码(号), CPU 依据它选择不同的通道。

设备地址: 给出设备的地址码即设备的编码(号), CPU 依据它选择不同的设备。

首控制字地址: 通道程序首条控制字的地址。关于控制字和通道程序及通道 I/O 指令举例将在 6.6.2 节介绍。

(2) 非通道 I/O 指令

非通道 I/O 指令的格式如下:

操作码	Reg	控制命令码	设备地址
-----	-----	-------	------

操作码: 不同的计算机此字段的意义不同。有的指出执行什么操作, 如读写、查询外设状态、走纸等; 有的只是用来区别是 I/O 指令还是其他类型指令。

Reg: 指出与操作有关的 CPU 寄存器。如操作为查询外设状态, 则该 CPU 寄存器用来存放外设的状态信息; 如操作为读数据到 CPU, 则该 CPU 寄存器用来存放从外设读入的数据。

控制命令码: 指出对设备如何控制, 如启停、落笔、绘图、扫描、走纸等。若操作码字段只是用来区别是 I/O 指令还是其他类型指令, 则外设的所有操作均由该字段指出。

设备地址: 给出设备的地址码即设备的编码(号), CPU 依据它选择不同的设备。

系统要使用某一 I/O 设备时, CPU 借助于 I/O 指令, 通过总线向 I/O 设备的接口发送设备地址, 只有被选中的设备及其接口才能响应主机的控制并与主机交换数据。

Intel 系列微机有输入、输出两类共 12 种形式的 I/O 指令。

IN	AL,n	; AL←端口 n 中的字节信息(n 为 8 位立即数, 表示端口地址, 范围为 0 ~ 255)
IN	AX,n	; AX←端口 n+1 和 n 中的字信息。
IN	EAX,n	; EAX←端口 n+3,n+2,n+1 和 n 中的双字信息。
IN	AL,DX	; AL←端口(DX)中的字节信息((DX)表示 DX 中的内容, 范围为 0 ~ 65535)
IN	AX,DX	; AX←端口(DX)+1 和 (DX)中的字信息。
IN	EAX,DX	; EAX←端口(DX)+3,(DX)+2,(DX)+1 和 (DX)中的双字信息。
OUT	n,AL	; 端口 n 中 ←AL 的内容
OUT	n,AX	; 端口 n+1 和 n 中 ←AX 的内容
OUT	n,EAX	; 端口 n+3,n+2,n+1 和 n 中 ←EAX 的内容
OUT	DX,AL	; 端口(DX)中 ←AL 的内容
OUT	DX,AX	; 端口(DX)+1 和 (DX)中 ←AX 的内容
OUT	DX,EAX	; 端口(DX)+3,(DX)+2,(DX)+1 和 (DX)中 ←EAX 的内容

6.3 程序直接控制传送

程序直接控制传送的特点是 I/O 操作完全处于 CPU 的指令控制之下, 通常, I/O 操作都是在 CPU 的寄存器与外设接口中的寄存器之间进行, I/O 设备不能直接访问内存。

6.3.1 无条件传送方式

无条件传送是所有传送方式中最简单的一种, 实际上是一种无需查询外设工作状态的 I/O 操作方式, 因此, 其所需的硬件和软件数量极少。这种方式仅在外设的各种动作时间是固定的、已知的条件下才能使用, 否则容易出错。为使传送可靠, 编程人员需了解外设的动态使用情况, 显然, 这一方面将增加编程人员的负担, 另一方面也易造成数据的丢失。

无条件传送的操作步骤一般如下:

- (1) CPU 把一个设备地址送到地址总线上, 经译码选定一台外部设备。
- (2) 若为输出操作, 则 CPU 向数据总线发送数据; 若为输入操作, 则 CPU 等待外设的数据缓冲寄存器中的数据出现在数据总线上。
- (3) 输出时, CPU 发出写命令, 将数据总线上待输出的数据送往外设的数据缓冲寄存器; 输入时, CPU 发出读命令, 从数据总线上将要输入的数据取来送到指定的 CPU 寄存器中。

无条件传送一般适合于对采样点的定时采样或对控制点的定时控制等场合。通常是根据外设的定时操作, 将 I/O 指令插入到程序中, 使程序的执行与外设同步。因此, 这种传送方式又称为程序定时传送方式, 只适用于个别的慢速外设。如读时钟并显示。

6.3.2 程序查询传送方式

在传送数据前, 需要查询即了解外设的工作状态。只有查询的状态满足条件时, 才能执

行这次传送。因此,这种方式又称之为条件传送或状态驱动传送方式。

1. 程序查询方式接口

如图 6.5 所示,程序查询方式接口主要由数据缓冲寄存器 DBR、设备状态与控制寄存器 (Status and Control Register, SCR)、设备地址译码器及有关控制逻辑组成。

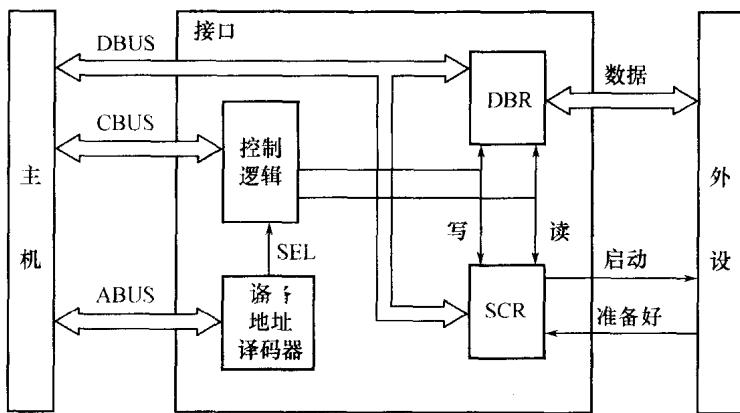


图 6.5 程序查询方式接口

SCR 寄存着主机发送给外设的命令信息和外设的状态信息。外设的每一种状态可由 1 个触发器表示,外设有多少种需表示的状态,SCR 中就有多少个触发器与之对应。外设的状态通常有“忙闲”、“出错”、“就绪”、“设备良故”等。“忙闲”指出外设是正在工作,还是闲置;“就绪”表示外设单位数据已经准备好,单位数据指的是 1 次可传输数据的二进制位数,通常是 DBR 的位数。对于输入设备,准备好意味着输入的单位数据已送入接口中的 DBR,对于输出设备,准备好意味着 CPU 送到接口 DBR 中的单位数据已被外设取走;“出错”指出外设在工作出现错误;“设备良故”指出外设目前是否良好可用。主机通过 SCR,可了解外设的现有状态,以便采取相应的处理措施。

设备地址译码器用来识别外设。如果地址总线 ABUS 送来的地址码是针对本外设接口的,则产生“选中”信号 SEL,此时控制总线 CBUS 送来的有关控制信号才能被接收,才会对该接口起控制作用。

2. 程序查询方式数据传送的工作过程

CPU 启动外设前应作如下准备工作:

- (1) 因传送往往是一批数据,故需设置外设与主机交换数据的计数值(个数);
- (2) 设置该批数据在主存中存放的始地址;
- (3) 通过 I/O 查询指令,了解外设是否是良好且闲置的,若是闲置的,才正式进行程序查询方式的数据传送。若是输出,还应将首次要输出的单位数据送到相应接口的 DBR 中。

程序查询方式实现 CPU 和外设之间的数据传送的一般操作过程如下:

- (1) I/O 指令启动外设并将命令码送 SCR 中；
- (2) 通过 I/O 查询指令，了解外设是否准备就绪，即判断 SCR 中的“就绪”是否为“1”
- (3) 如外设未准备就绪，则踏步等待即重复(2)，否则，往下执行；
- (4) CPU 执行 I/O 指令，若为输入，CPU 从接口的 DBR 中取走单位数据；若为输出，CPU 再输出一个单位数据至接口的 DBR。同时将接口中的“就绪”标志清“0”，“忙闲”标志置“1”；
- (5) 修改主存地址和计数值，并判计数值是否为零；
- (6) 若计数值不为零，重复(1)~(5)，直至计数值为零即一批数据交换完毕，结束 I/O 传送，继续执行其他程序。

上述操作过程可用图 6.6 描述。显而易见，这种方式中，CPU 的操作可以和外设操作同步，接口电路也很简单。但程序进入循环时，CPU 只能长时间的踏步等待，不能处理其他任务，系统效率很低。因此，这种传送方式也仅用于 CPU 速度不高、外设配备不多的情况。

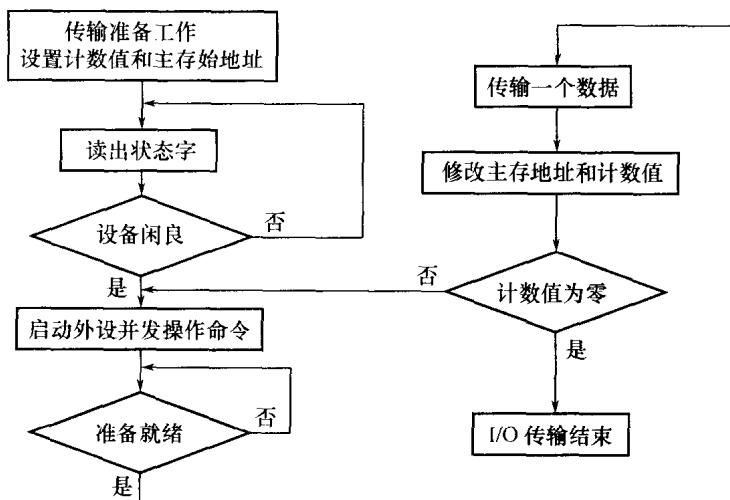


图 6.6 程序查询数据传送的程序流程

下面是一个采用查询方式打印一批字符的程序。程序通过反复测试打印机的状态以决定何时输出。在打印机接口中，假定，数据缓冲寄存器的端口地址为 378H，状态寄存器的端口地址为 379H，控制寄存器的端口地址为 37AH，三个均为 8 位的寄存器。假定，状态寄存器的最高位为忙闲位，“1”表示闲；最低位为良故位，“1”表示故障。控制寄存器第 0 位为启停位，“1”表示启动打印机输出，“0”表示停止打印机输出；第 1 位为自动换行位，“1”表示自动换行；第 2 位为初始化位，“1”表示初始化正常；第 3 位为接通位，“1”表示与打印机接通。

```

;程序查询方式打印字符的程序 PRINT_CHAR_PROGRAM
DATA SEGMENT
CHARARRAY DB 'PRITER IS NORMAL',0DH,0AH

```

```

        DB 'ABCDEFGHIJKLMNPQRSTUVWXYZ',0DH,0AH
        DB 'abcdefghijklmnopqrstuvwxyz',0dh,0ah      ;定义要打印的信息
COUNT      EQU $ - CHARARRAY           ;定义要打印的信息的长度
DATA       ENDS
STACK      SEGMENT PARA STACK 'STACK'
          DB 100 DUP(?)
STACK      ENDS
CODE      SEGMENT
ASSUME CS:CODE,SS:STACK,DS:DATA
PRINT:    MOV AX,DATA
          MOV DS,AX           ;设置数据段段址
          LEA SI,CHARARRAY
          MOV CX,COUNT         ;设置要打印字符的始地和个数
GOON:    MOV DX,379H
WAIT:     IN AL,DX           ;读状态字
          TEST AL,80H
          JZ WAIT             ;忙,等待!
          TEST AL,1
          JNZ WAIT             ;故障,等待!
          MOV AL,[SI]           ;闲良,取字符
          MOV DX,378H
          OUT DX,AL            ;将字符送打印机接口中的 DBR
          MOV DX,37AH
          MOV AL,0DH
          OUT DX,AL             ;向打印机控制口发打印启动命令
          MOV AL,0CH
          OUT DX,AL             ;向打印机控制口发打印停止命令
          INC SI
          LOOP GOON            ;继续打印其他字符,直至打印完
          MOV AH,4CH
          INT 21H                ;打印完返 DOS
CODE ENDS
END PRINT

```

6.4 中断及程序中断控制传送

中断和中断系统是现代计算机不可缺少的技术。什么是中断,中断有什么作用,中断有哪些类型,中断系统怎样构成,中断的过程怎样,中断是怎么实现的,它又是如何应用于外设与主机之间的数据传送的。本节将对此逐一进行讨论。

6.4.1 中断的有关问题

什么是中断?中断(Interrupt)是指CPU暂时中止现行程序,转去执行“处理随机发生的

“紧急事件或特殊请求”的程序,处理完后自动返回被中止程序继续运行的功能。通常把实现中断功能所需的软硬件技术,通称为中断技术。中断系统是计算机实现中断功能的软硬件总称。为了处理各种中断,一般在CPU内设置处理中断的机构,以解决各种中断共性的问题。在外设接口中设置中断控制逻辑,以解决程序中断控制数据传送所需的硬件问题。在软件上设置相应的中断服务程序,以实现对随机发生的紧急事件或特殊请求的处理。

1. 中断的作用

中断是现代计算机能有效合理的发挥效能和提高效率的十分重要的功能。它赋予计算机实时处理和自动处理计算机内部故障的能力,将有序的运行程序与无序的突发事件统一起来,大大增强了系统的处理能力。它是计算机发展史上的一个重大里程碑,现代计算机中都具有中断功能和相应的中断系统。中断的作用主要是实现I/O的控制、处理故障、人机通信、多道程序和实时控制等。

(1) 实现主机和外设的并行工作

中断用于I/O控制主要为了实现主机和外设、外设和外设的并行工作,从而提高计算机系统的工作效率。不使用中断技术时,外设与主机的数据交换采用程序直接控制,在查询外设是否准备好数据时,CPU处于踏步等待状态,因而系统的效率极低。引入中断后,CPU查询外设状态的时间被用来运行程序,系统工作效率大大提高。同时,多台外设可在不同的时刻向主机发中断请求,主机可分别及时地处理这些请求,以完成数据交换。因而从整个系统观察,多台外设也是并行工作的。

(2) 处理故障

计算机运行时,可能会发生运算溢出、除数为零、无休止循环、非法指令、存储器超量装载、电压下降(电源掉电)、校验出错、外设传输错等软硬件故障。这些故障一旦出现则可能使系统瘫痪。中断系统可在故障出现时发出中断请求信号,调用相应处理程序,将故障的危害降低到最低程度,从而提高了计算机系统的可靠性。

(3) 实现多道程序和分时操作

多道程序运行是提高机器效率的有效手段。多道程序的切换运行需借助于中断系统。在一道程序的运行切换到另一道程序运行时,可以通过分配每道程序一个固定的时间片,利用时钟定时发中断进行程序切换;亦可由I/O中断系统进行程序的切换。

(4) 实现实时控制

实时控制,是指在某个事件或现象出现时及时的进行控制。如在某过程控制系统中,当出现温度过高、压力过大或烟雾达到某种程度时,必须通过相应的传感器及时通知给计算机,以便对其进行控制。这些事件是随机出现的,程序本身是不能预见的,因此,要求计算机中断正在执行的程序,转去执行相应的中断服务程序。从而达到控制受控对象的目的。

(5) 实现人机联系与通信

计算机运行过程中,人可能要随机地干预机器,如抽查中间结果,了解工作状态,下达临

时命令等。无论是通过终端设备或是控制台进行干预,在没有中断系统的计算机中这些功能几乎是无法实现的。而利用中断系统进行人机通信既有效又方便。

此外,在多机系统中,各处理机之间的信息交流和任务切换须通过中断系统来实现。目态程序和操作系统(管态程序)的联系也须通过中断系统实现,通常是在用户程序中(目态)安排一条访管指令来调用操作系统的管理程序(管态)。中断系统与操作系统密切相关,操作系统是借助于中断来控制和管理计算机系统的,甚至可称操作系统为中断驱动的程序。

2. 中断的类型

在阐述中断分类之前,先引入两个名词术语。

- (1) 中断源(Interrupt Source):引起处理机中断的事件,称之为中断源。
- (2) 广义指令:亦称自愿进管指令。由一条特殊的转移指令和若干参数组成。

中断可以从不同角度分类,最常见的分类有以下几种方式。

- (1) 自愿中断和强迫中断

自愿中断:又称程序自中断,是在程序中预先安排的由广义指令引起的中断。自愿中断是可重现的、预知的。设置广义指令的目的是便于软件调试、调用系统功能和用户在目态情况下使用外设等。用户程序执行到使用外设的广义指令时,从目态转入管态,把参数加工成使用外设所需的信息并启动外设工作,然后从管态返回目态,使CPU与外设并行工作。

强迫中断:不是由程序事先安排好的,而是随机产生、不可预知的中断。这种中断发生时,中断系统强迫机器中止现行程序,转入中断服务程序以处理突发的事件,处理完后再继续执行被中止程序。

- (2) 内中断与外中断

内中断:指主机内部硬件和软件原因引起的中断,它又可分为硬件故障中断和陷阱(Trap)。硬件故障如电源断电、各种校验错误等,这类故障需紧急处理。处理这类中断时,或者将程序运行现场存入存储器,以期修复后程序继续运行;或者切除故障部件,向操作员发出警告或呼叫信号,以请求人工干预。而陷阱是指由于程序本身运行的原因引起的中断,如非法操作码、阶码上下溢、“0”作除数、堆栈溢出、地址越界等。这种中断与程序是同步的,即将程序重复执行时,陷阱将在同样位置出现,故又称其为程序性中断。

外中断:指主机以外的部件引起的中断,如外设引起的I/O中断、操作员通过控制台对机器干预的中断、定时器用于计时计费引起的时钟中断、其他机器或系统产生的外部信号中断等。

操作系统教科书中称的异常,包括自愿中断和陷阱这两类中断。

- (3) 单重中断与多重中断

单重中断:在执行中断服务程序的过程中,不允许再打断该服务程序,只有在该服务程序执行完后,才能响应新的中断请求。称这样的中断系统为单重中断系统。

多重中断:若CPU在执行某个中断服务程序时,还可响应优先级别高的中断请求,称这

样的中断系统为多重中断系统。这种重叠处理中断的现象又称中断嵌套。多重中断与单重中断表明机器中断功能的强弱,有的机器可实现三级甚至更多级的多重中断。

(4) 可屏蔽中断和不可屏蔽中断

可屏蔽中断:可不响应或暂不响应,或有条件响应的中断,称为可屏蔽中断。当中断源产生中断时,用程序方法有选择性的封锁部分中断,使之不发出中断请求,而允许其余部分中断发出中断请求并得响应,称它为中断屏蔽(INTerrupt Mask)。可屏蔽中断就是通过中断屏蔽实现的。具体实现方法是,在硬件上为每个可屏蔽中断源设1个屏蔽触发器,用程序方法将该触发器置“1”,则相应中断源不能发出中断请求,若将其置“0”,则允许该中断源发出中断请求。

不可屏蔽中断:必须立即响应、不能回避和禁止的中断,称为不可屏蔽中断。不可屏蔽的中断源产生中断必须立即响应,即它们具有高的优先级,如断电中断是具有最高优先级的不可屏蔽中断,故对断电中断的处理安排在DMA和所有中断之前。自愿中断也属于不可屏蔽中断。

3. 中断系统需解决的问题

中断系统实现中断功能,依中断功能可推知中断系统必须解决如下主要问题。

如何标识并记录各中断源是否产生中断及若已产生中断又怎样向CPU提出中断请求;当系统有多个中断源同时产生中断并向CPU提出中断请求时,中断系统如何确定优先响应哪个中断源的请求;CPU在什么条件下可响应中断及响应中断时应做哪些工作;CPU响应中断后怎样暂停现行程序转去执行中断服务程序;中断服务程序的基本结构组成怎样;中断处理完后如何返回被暂停程序的间断处继续执行;若中断系统为多重中断系统,在中断处理过程中如出现新的中断请求,应如何处理等问题。

4. 中断的全过程

中断的全过程是指从中断源发出中断请求开始,CPU响应请求,暂停现行程序,进入中断服务程序,直至服务程序执行完毕,CPU再返回现行程序继续执行的整个过程。通常中断全过程可分为中断请求、中断判优、中断响应、中断处理和中断返回五个阶段。需要指出的是,因自愿中断是在程序中预先安排的,其过程只有中断响应、中断处理和中断返回三个阶段。

6.4.2 中断系统的结构组成

本小节依据中断系统需解决的主要问题及中断的全过程,以外中断为例,讨论多重中断系统的结构组成及其工作原理。构建中断系统,首先考虑哪些问题用硬件解决、哪些问题用软件解决、哪些问题既可用软件又可用硬件解决,哪些问题须软、硬件结合才能解决。首先应考虑速度,用硬件速度快、但不灵活;用软件灵活、但速度慢。其次考虑效率、灵活性、可靠性、经济等诸方面因素,综合利弊,将解决所有问题的软硬件有机地组织起来,即构成完善的中断系统。

1. 中断寄存器、中断请求寄存器和中断屏蔽寄存器

(1) 中断寄存器和中断字

计算机执行程序过程中,可能出现“定点溢出”、“阶码上溢”,外部设备请求 I/O 等中断。为了记录并区分不同的中断,通常计算机系统为每 1 个中断源设置 1 位中断触发器。当某一意外事件发生时,将与之对应的中断触发器置为“1”,表明该中断源产生了中断;若该中断触发器为“0”,表明该中断源未产生中断。由于中断是随机产生的,故中断触发器置“1”的时间也是随机的。系统的所有中断触发器组成中断寄存器,其内容称为中断编码或中断字。CPU 在进行中断处理时,不可屏蔽中断由中断触发器直接发出中断请求,CPU 响应后便转到相应的中断服务程序进行处理。

(2) 中断请求寄存器

计算机系统为每个可屏蔽中断源还设 1 个中断请求触发器,系统的所有中断请求触发器组成中断请求寄存器。若某中断请求触发器为“1”,则表明该中断源发出了中断请求;否则,表明该中断源未发出中断请求。未发出中断请求的中断源,不能肯定无中断发生,因有无中断发生由该中断源的中断触发器决定。可屏蔽中断由中断请求触发器发出中断请求。

(3) 中断屏蔽寄存器和中断屏蔽字

计算机系统还为每个可屏蔽中断源设 1 个中断屏蔽触发器,系统的所有中断屏蔽触发器组成中断屏蔽寄存器,称其内容为中断屏蔽字或中断屏蔽码。它的功能是控制相应中断触发器是否发出中断请求,若某中断屏蔽触发器为“1”,则不允许该中断源发出中断请求。可屏蔽中断源发出中断请求的条件是“中断触发器为‘1’且中断屏蔽触发器为‘0’”,既有中断请求且不被屏蔽。

产生中断请求后,若 CPU 因某种原因(如执行操作系统中的原语),在某时间段内不能中止现行程序的执行,即不能响应中断,称为禁止中断,若 CPU 可以响应中断,叫允许中断。通常在 CPU 内设置“中断允许”触发器,通过“开中断”指令将该触发器置“1”,表示允许中断;通过“关中断”指令将该触发器置“0”,表示禁止中断。如 Intel 系列微机设置“中断允许”触发器 IF,“开、关中断”指令分别为 STI 和 CLI。

2. 中断判优

(1) 几个名词术语

中断优先权(INTerrupt Priority, INTP):中断响应的先后次序。中断优先权在中断系统设计时就已按中断的性质和请求处理的轻重缓急确定好,一般在程序运行中保持不变。系统程序员可通过改变中断屏蔽码变更中断优先权,但要慎重,不得随意改变。因为优先权的排列顺序是根据中断的性质,以及中断对整机工作的影响来确定的。例如机器发生故障,将使机器无法工作,而输入/输出则仅仅影响数据的传送,因此故障中断的优先级高于 I/O 中断;高速设备中断的优先级高于低速设备中断的优先级。

中断排队:依中断优先权给多个同时发生的中断请求,排出中断响应先后次序的过程。

中断判优:通过中断排队,从多个同时发生的中断请求中选出需优先处理的中断源。

(2) 中断源的分级

对于中断源较少的计算机系统,采用统一集中排队。当计算机系统中断源很多时,一般采用分级排队法。即依各中断的性质和轻重缓急把中断源分成若干级,每级再包括若干个中断。如某机分为3级中断,见图6.7所示。每一级均有自己的中断寄存器。第1级中断优先权最高,包括电源断电及各种硬件故障中断。第2级为程序性中断,第3级为输入/输出中断,中断优先权最低。每一级内部的中断优先权是按中断字左高右低排列。例如在第3级中,中断字的第1、3位为“1”,说明同时出现了两种中断,CPU应先处理第1位对应的外部信号中断,而后处理第3位对应的控制台中断。

中断源分级时,因不可屏蔽中断种类不多,一般把它们分在同一级中,在硬件配置上只需中断寄存器。可屏蔽中断依具体情况可分为若干级,每级除配置中断寄存器外,还需有中断请求寄存器和中断屏蔽寄存器。

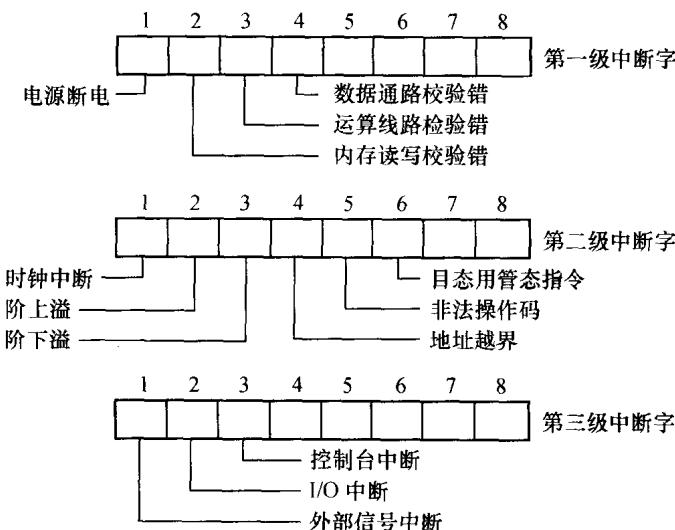


图6.7 中断源分级与中断字

(3) 中断判优逻辑

任一中断系统,在某一时刻只能响应一个中断请求,为一个中断服务。当多个中断源同时提出中断请求时,则需通过中断判优逻辑确定优先中断源以便为其服务。中断判优既可采用软件方法实现,也可采用硬件的方法实现。

① 硬件判优线路

硬件判优线路,亦称硬件排队器,目前广泛采用的有两种。一种为最左判优线路,如图6.8所示。它是广泛使用的一种并行中断判优线路,速度快,但较费器材。

最左边的中断请求优先权最高,从左到右优先级递减。当高一级的有请求时,经反相器

输出封锁了低一级的中断请求；仅当所有高一级的中断源无请求时，低一级的中断请求方能排上队。显而易见，最左判优线路既可用于各中断级之间的判优，也可用于级内各中断源之间的判优。另一种为链型判优线路，如图 6.9 所示。图中下部一排门电路为链型判优线路的核心，每个中断源配置一个非门和一个与非门，作为链型判优线路链中的一“环”。该线路由四“环”构成，故为四个中断源的链型判优线路，其优先权也是左高右低，即最高的是 1 号，其次是 2、3、4 号。

只要有中断源提出中断请求，不论几个，线路输出端 IP_i 只有一个高电平，即只有一个为优先中断源。当无中断请求时， $INTR_i$ ($i = 1, 2, 3, 4$) 均为低电平， IP'_i 、 \overline{INTR}_i 均为高电平， IP_i 均为低电平。一旦某中断源提出中断请求，就迫使比其优先权低的中断源之 IP'_i 和 IP_i 均变为低电平，即均为非优先中断源，只有其本身为优先中断源。如 2 和 4 号中断源同时有中断请求，经分析可推知 IP'_1 、 IP'_2 为高电平， IP'_3 、 IP'_4 为低电平，从而只使链型判优线路的 IP_2 为高电平，即 2 号为优先中断源。

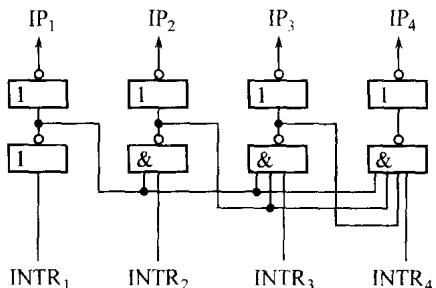


图 6.8 最左判优线路

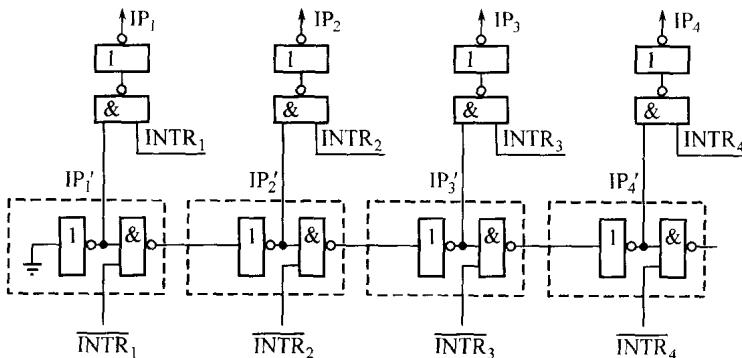


图 6.9 链型判优线路

链型判优线路属于串行中断判优线路，通常用于 I/O 中断的判优，构成链型判优线路的各“环”分布在相应中断接口的逻辑中。这样的构造既给系统扩充外设提供极大方便，又省器材。

② 软件判优程序

软件判优指的是通过执行中断查询程序寻找优先中断源，而后转入相应的中断服务程序。程序按中断源优先权从高到低的顺序编写，中断响应时执行中断查询程序，就保证 CPU 首先响应级别高的中断请求。改变中断查询程序的编写顺序，即修改了中断源的优先权。中断查询程序的框图如图 6.10 所示。软件判优法优点是对中断源的判优和识别不需增加硬件设备，且能灵活修改中断源的优先权。缺点是判优需执行中断查询程序，增加了 CPU

响应中断的时间，降低了系统效率。对中断源较多的计算机系统，甚至会因响应中断的时间增加使系统瘫痪。因此，软件判优法仅用于中断源较少、速度较慢的计算机系统。

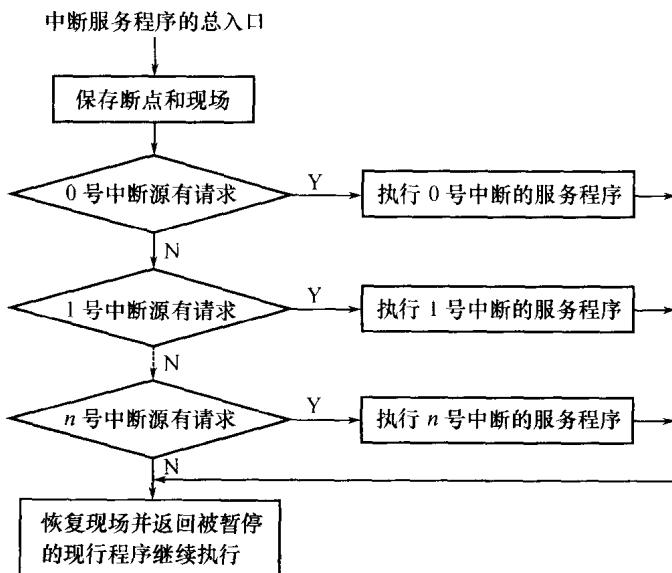


图 6.10 中断查询程序流程

3. 中断响应

(1) CPU 响应中断的条件：一是中断源有中断请求；二是 CPU 允许中断即开中断；三是 CPU 在一条指令执行完毕，且没有更紧迫的任务或事件，如断电中断、DMA 传送。图 6.11 示出图 6.7 三个中断级中断请求逻辑示意图。图中，INTR - 1、INTR - 2、INTR - 3 表示相应中断级有否中断请求，低表示有中断请求。

(2) 中断响应完成的任务：完成暂停现行程序和进入中断服务(处理)程序两项任务。

(3) 暂停现行程序要做哪些工作

① 关中断，进入不可再次响应中断的状态，以便在保护中断现场时，不被新的中断所打断，从而保证被中断的程序在中断处理完后能接着正确地执行下去。

② 保存断点：保存程序计数器 PC 的内容。PC 内容指示程序执行到哪条指令，即给出被暂停程序尚未执行的指令地址，该地址称为断点。保存断点是保证中断服务程序执行完后，能正确返回被中断的程序。保存断点最常见的方法是压入堆栈，中断处理完后，再由堆栈弹出以恢复断点。

③ 保存其他硬件现场：保存程序状态字 (Program Status Word, PSW) 中的某些内容，如条件码、中断码和状态标志等。所谓程序状态字是指反映程序运行状态的一组字。通常包括该程序的中断屏蔽字、中断字、条件码、程序运行的状态标志和程序计数器 PC 的值。一般设置程序状态字寄存器存放程序状态字。这里谈及的保存程序状态字的某些内容，是因为计

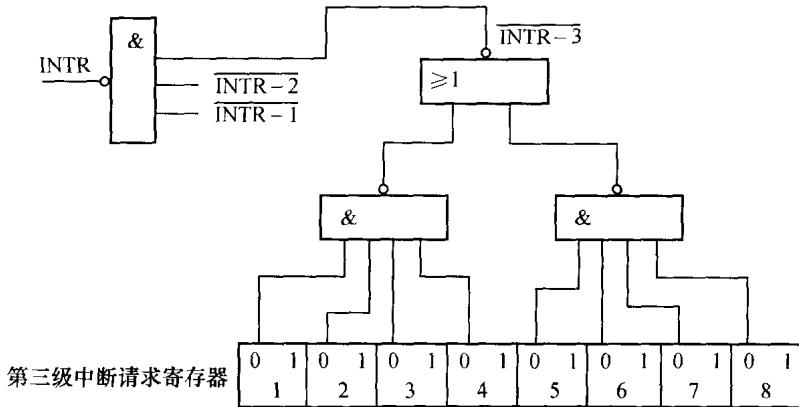


图 6.11 三个中断级中断请求逻辑示意

计算机结构和中断机构不同,导致程序状态字组成不同,使得保存内容的多少也不同。

④ 清除当前正被响应的中断请求,以防止一次中断请求被多次响应。

暂停现行程序要做的这四项工作通常由中断隐指令完成。中断隐指令没有操作码,只在响应中断时才由硬件产生以完成中断响应的各项工,它像一条指令,但计算机指令系统中又无此指令,故称中断隐指令。为实现中断隐指令的功能,CPU 的控制器特设一个中断(PI)周期。该周期和取指周期、执行周期的地位是相同的。只要 CPU 响应中断,执行周期结束时,不是进入取指周期,而是进入中断周期,中断周期执行完后,才进入取指周期。

(4) 进入中断服务程序的方法

进入中断服务程序的具体工作是寻找中断服务程序的入口地址,然后将该地址送程序计数器即可。进入中断服务程序,可采用软件法,亦可采用硬件的方法完成。

① 软件法:即通过执行中断查询程序寻找优先中断源。中断查询程序流程如图 6.10。由图可知,当查到某中断源有中断请求时,则安排一条转移指令,进入相应中断服务程序对中断进行处理。而各中断源中断服务程序的入口地址,则由系统事先约定。

② 硬件法即向量中断法:每个中断源都对应着一个中断服务程序,各有其入口地址,这些入口地址的有序集合,称为中断向量。中断向量组成的一维表格,称为中断向量表,它存放在一段连续的存储区域中。访问中断向量表所需地址叫向量地址,或称中断指针。

向量中断法就是将各中断服务程序的入口地址(或包括程序状态字的部分内容)组成中断向量表;响应中断时,借助于硬件判优线路,找出优先中断源,再由硬件产生相应中断源的向量地址;据此访问中断向量表,从中读取服务程序入口地址,并由此进入中断服务程序。整个工作是由中断隐指令完成的。

向量中断法的一个关键问题是向量地址如何产生?其产生方法有以下几种:

(1) 中断响应时,由硬件直接产生与中断源对应的向量地址。如在 Intel 系列微机中,中断向量表存放在地址为 0~1 023(十进制)的主存单元中。每个中断源占用 4 个存储单元,

存放相应服务程序入口地址,其中前2个单元存放其偏移量,后2个单元存放段地址。因此整个中断向量表可对应256个中断源,即对应中断类型码0~255。当CPU响应中断请求时,向中断控制逻辑发送中断响应信号INTA;从中断控制逻辑取出优先中断源的中断类型码;中断类型码乘以4形成向量地址;访问主存,从中断向量表读取服务程序入口地址;从此进入中断服务程序。图6.12示出向量地址形成部件和中断向量表。 $IP_0, IP_1, \dots, IP_{255}$ 为中断判优电路的输出,在某一时刻,只能有一个为高电平,因此输出的向量地址也是惟一的。

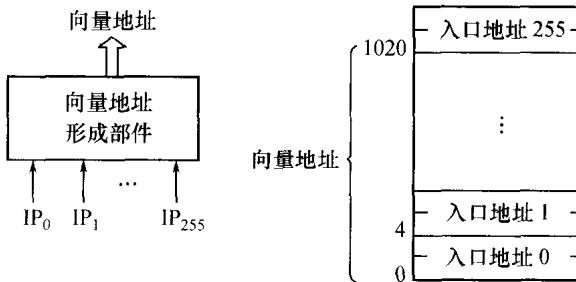


图 6.12 向量地址形成部件和中断向量表

(2) 中断响应时,由硬件产生的不是与中断源对应的向量地址,而是相对于中断向量表始地址的偏移量。系统将向量表的始地址存入CPU向量表基址寄存器中,向量地址为该寄存器内容与偏移量之和。这种方法使得中断向量表可存放在主存的任何位置。

(3) 中断响应时,由硬件产生与中断源对应的向量地址的低位部分,而在CPU设置中断向量寄存器,用以存放向量地址的高位部分,二者拼接形成完整的向量地址。这种方法使得中断向量表亦可存放在主存的任何位置。

(4) 在具有多根中断请求线的计算机系统中,通过对请求线编码产生各中断源的向量地址,见图6.13,该向量地址就是相应中断服务程序的入口地址。

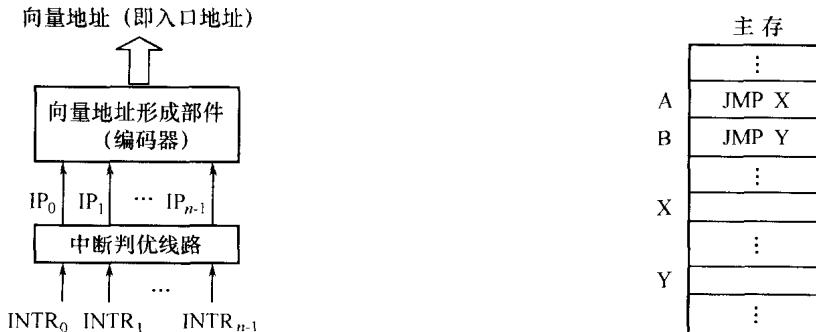


图 6.13 由请求线直接给出入口地址

图 6.14 向量地址中存储转移指令示意

(5) 中断响应时,由硬件直接产生与中断源对应的向量地址,但向量地址中存储的内容不是中断服务程序的入口地址,而是一条无条件转移到中断服务程序的指令,如图6.14。图

中 A、B 为向量地址, X、Y 为中断服务程序的人口地址。

4. 中断服务程序的结构组成

执行中断服务程序就是处理中断事务。对不同的中断源,服务程序的内容各不相同。对不同的计算机系统,中断处理的过程也各不相同,但主要包括以下几个部分。

(1) 保护现场:凡中断服务程序用到的寄存器,包括中断屏蔽寄存器。因执行中断处理程序时,可能破坏它们的内容。故把它们的内容保存起来,谓之保护现场。通常的办法是压入堆栈保存或使用存数指令存储到主存保存;

- (2) 屏蔽低级中断:送新的屏蔽码;
- (3) 开中断:即将中断允许触发器置“1”,允许响应比本中断更高级的中断请求;
- (4) 实现对突发事件的处理:执行相应中断源的处理程序;
- (5) 关中断:即将中断允许触发器清“0”,不允许响应任何可屏蔽中断;
- (6) 恢复现场:即将保存起来的寄存器内容予以恢复,它是保护现场的逆过程。
- (7) 开中断:即将中断允许触发器置“1”,允许响应中断请求;

(8) 返回被暂停程序:中断服务程序的最后一条是中断返回指令,它将原来压入堆栈的断点和状态字弹出,使整个系统恢复到被中断程序的运行状态。

注意:两次开中断、一次关中断是因为保护和恢复现场时不允许响应中断。一旦保护和恢复现场完毕,即开中断,允许响应中断。

CPU 响应与处理中断的过程如图 6.15 所示。

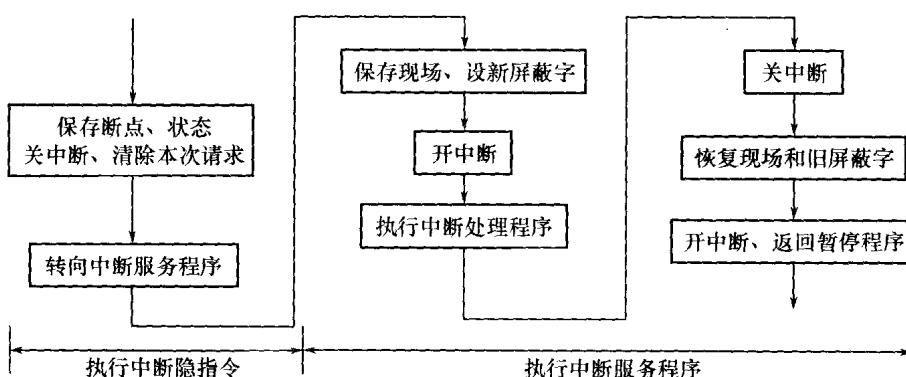


图 6.15 中断响应和中断处理过程

5. 中断屏蔽技术与多重中断

(1) 中断屏蔽技术

前面已谈及,计算机系统为每个可屏蔽中断源设 1 个中断屏蔽触发器,系统的所有中断屏蔽触发器组成中断屏蔽寄存器,称其内容为中断屏蔽字或中断屏蔽码。为实现屏蔽功能,应预先为每个中断源设一个中断屏蔽码。屏蔽码与中断源的优先级别是一一对应的,不同

中断源的屏蔽码是不同的。表 6.1 是对应 8 个中断源且具有 8 个优先级的屏蔽码。1 级中断源的屏蔽码为 8 个 1, 2 级中断源的屏蔽码从左数第 2 位开始共 7 个 1, 依此类推, 第 8 级中断源的屏蔽码第 8 位为 1 其余位为 0。执行中断服务程序时, 通过屏蔽指令将相应屏蔽码送中断屏蔽寄存器, 即达到屏蔽目的。如 CPU 已响应了 1 级中断, 因设置了全“1”屏蔽码, 便保证在执行该中断服务程序过程中不会再响应任何可屏蔽中断请求, 即此时不能实现多重中断。再如若 CPU 响应了 3 级中断, 因设置的屏蔽码为“00111111”, 使得 3 级 ~ 8 级的中断源被屏蔽, 而 1 和 2 级中断源的中断请求可以中断该级中断源的中断服务程序, 从而实现了多重中断。

表 6.1 中断优先级和中断屏蔽码

中断源序号	优先级	中断屏蔽码
1	1	11111111
2	2	01111111
3	3	00111111
4	4	00011111
5	5	00001111
6	6	00000111
7	7	00000011
8	8	00000001

利用中断屏蔽技术可改变中断优先级, 如在 3 级中断服务程序中设置的屏蔽码为“00110111”, 则当 4 和 5 级中断同时有请求时, 由于 4 级被屏蔽, 而 5 级未被屏蔽, 因此 CPU 将中止 3 级中断服务程序的执行而去响应 5 级中断。只有当处理完 5 级中断源的请求后, 再继续执行 3 级中断服务程序。3 级中断服务程序执行完后, 若又产生 3 级中断, 而此时在 3 级中断服务程序设置的屏蔽码为“00111111”, 这样在 3 级中断服务程序执行过程中, 3 级和比 3 级低的中断都不会被响应。

(2) 多重中断的实现

借助于中断屏蔽寄存器和屏蔽码, 通过屏蔽指令可实现多重中断。即若新发生的中断同现处理的中断相比是低级或同级的中断, 可不予响应; 若为高级的中断, 可中断现行的中断处理程序, 优先处理高级中断。图 6.16 是一个 3 级中断的例子, 其中 k 、 l 、 m 为响应点, $k+1$ 、 $l+1$ 、 $m+1$ 为断(恢复)点。当现行程序运行到第 k 条指令时, 发生了 3 级中断, CPU 响应 3 级中断, 运行 3 级中断处理程序。当 3 级中断处理程序运行到第 1 条指令时, 又发生了 2 级中断, CPU 又响应 2 级中断, 此时中止 3 级中断处理程序, 转入运行 2 级中断处理程序。当 2 级中断处理程序执行到第 m 条指令时, 又发生了 1 级中断, CPU 要中止 2 级中断处理程序, 转入运行 1 级中断处理程序。1 级中断处理完后, 再返回 2 级中断处理程序的第 $m+1$

条指令,继续处理2级中断。2级中断处理完后,再返回3级中断处理程序的第 $l+1$ 条指令,继续处理3级中断。3级中断处理完后,才返回现行程序的第 $k+1$ 条指令,继续运行现行程序。在上述程序的转返过程中,中断系统需对3个断点按正序保存,按逆序恢复,通常都是借助于堆栈实现。

6.4.3 程序中断控制传送及其接口

采用程序查询方式传送数据,明显存在以下缺点:在查询过程中,CPU处于踏步状态,系统效率大大降低;CPU在一段时间内只能和一台外设交换信息,其他外设无法同时工作;无法发现和处理突发的错误与异常情况。引入中断技术后,可实现CPU与外设数据传送的并行工作,也可实现多台外设的并行工作,大大提高了系统的效率。图6.17示出两台外设采用程序中断方式传送数据示意。外设一速度慢,外设二速度快,外设二的优先级高于外设一的优先级,当两外设同时发生中断请求时,将优先响应外设二。CPU启动外设后,外设可以准备数据,而CPU可继续执行程序;一旦外设单位数据准备就绪,就向CPU发一个中断请求;CPU响应中断,暂停正在执行的程序,转去执行中断服务程序,为中断源服务;服务完毕,CPU恢复执行原来的程序。所谓单位数据是指,CPU程序被中断一次,主机与外设交换的数据量。如键盘的一个键值,鼠标的坐标值及该坐标的位移量,扫描仪的一个条形码等。

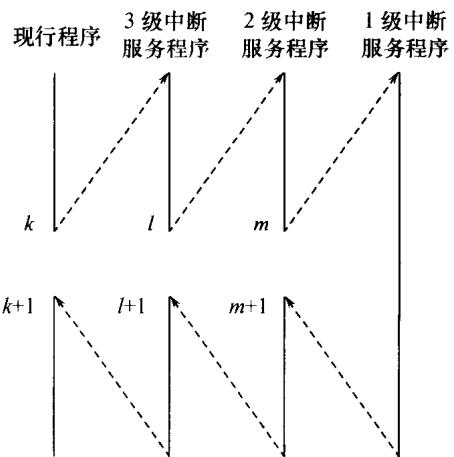


图 6.16 多重中断

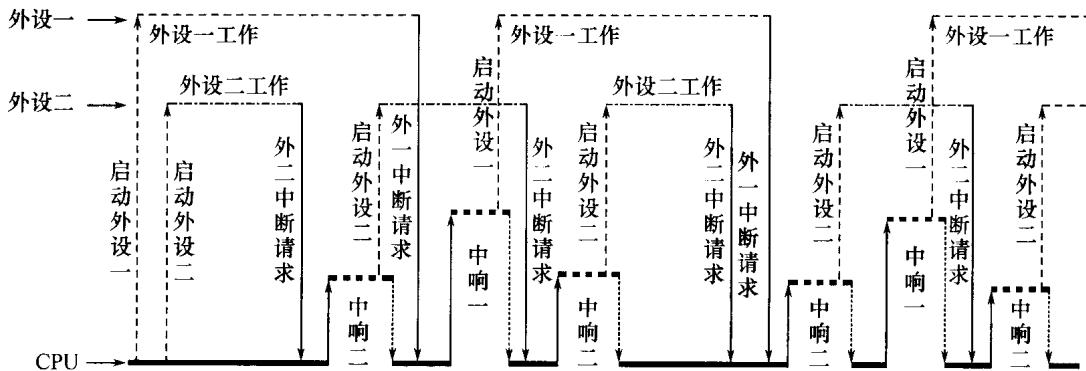
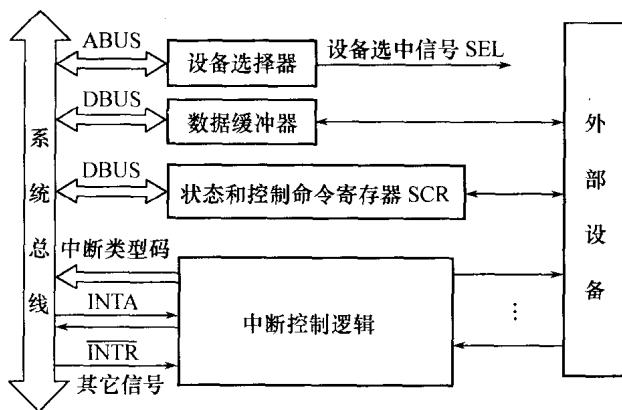


图 6.17 主机与外设、外设与外设程序中断传送示意

1. 程序中断接口

具有中断能力的外设接口需具备下述功能:选择和区别不同设备;在外设和CPU之间

起数据传输的桥梁作用,提供数据缓冲;将 CPU 的命令传送给外设;将外设的工作状态传送给 CPU;能参与排队判优,按优先顺序通知 CPU 处理外设的服务请求。因此,程序中断接口一般由如图 6.18 所示的逻辑电路组成。



(1) 设备选择器

实际上是一个译码器,它将 CPU 送来外设地址码或设备编号进行译码。故当 CPU 选择某一外设时,只有与该外设对应接口的设备选择器才产生选中信号,于是该接口及相应外设才能接受并响应 CPU 送来的控制信号,进行相应的操作。

通过设备选择器译码产生的选中信号与 I/O 指令中的操作码相配合,即可选择和控制相应外设及其接口实现 I/O 操作。

(2) 数据缓冲器

用以寄存要输入或输出的信息,其容量视外设单位数据的大小而定,通常为数据缓冲寄存器 DBR,如键盘、鼠标和扫描仪等。而有的设备如串行行式打印机,其容量应可寄存一行字符的数据,此时缓冲寄存器实际上已是一个小容量存储器。

(3) 状态与控制命令寄存器 SCR

该寄存器寄存两部分内容。一部分用于存放 CPU 送来的控制命令,以便指明接口及外设的操作方式。如读或是写,是走纸,还是打印等。另一部分用来记录和表示外设及接口的运行状态信息,如在接口中设置“忙闲 BUSY”、“完成(就绪) DONE”、“中断请求 INTR”、“中断屏蔽 MASK”,以及“故障”、“校验错”等触发器。CPU 可根据接口的实际运行状态对 I/O 操作进行动态控制,可根据状态寄存器了解故障性质,从而作出相应处理。

(4) 中断控制逻辑

是实现程序中断方式的数据传送和完成对外设的操作控制所必需的逻辑线路。通常有中断请求信号 INTR 的产生逻辑、与 CPU 之间的应答逻辑、中断屏蔽和中断选优判优逻辑、中断类型(设备)码回送逻辑以及面向外设的特殊操作控制逻辑等。以便形成送往 CPU 的

INTR信号、状态信息、中断类型(设备)码及接收CPU发出的中断响应信号INTA,从而进入中断服务程序。

其中,设备选择器、中断控制逻辑和状态与控制命令寄存器中的“忙闲BUSY”、“完成DONE”、“中断请求INTR”、“中断屏蔽MASK”构成程序中断传送的标准部分,称之为程序中断标准接口。其余部分将因外设的不同而异,称为接口的非标准部分。

2. 程序中断传送工作过程

(1) 初始化:CPU设置本次传输的主存始地、传输长度并通过I/O指令查询外设是否良好和闲置,若为空闲且良好,则由程序对接口初始化,如设置工作方式,复位或置位接口中的触发器。若为输出,还应将首次输出的单位数据送数据缓冲器。

(2) 启动外设:CPU通过I/O指令使接口的忙闲标志BUSY为“1”,完成标志DONE为“0”,据此启动外设工作。

(3) 中断请求:当外设1个单位数据准备好或已完成1个单位数据的输出,使忙闲标志BUSY为“0”,完成标志DONE为“1”,在不屏蔽即MASK为“0”情况下,形成中断请求信号INTR。

(4) 中断判优:经中断判优逻辑,向CPU发出中断请求INTR,并形成中断类型码即向量地址。

(5) 中断响应:CPU发中断响应信号INTA,进入中断周期执行中断隐指令。暂停现行程序,中断控制逻辑送出向量地址,转入中断服务程序。

(6) 中断处理和中断返回:执行中断服务程序,完成相应I/O操作后返回被暂停的程序。

不难看出,程序中断控制传送主要是中断的请求、判优、响应、处理和返回。CPU只是在响应中断后,才中止现行程序,转去执行中断服务程序以实现外设与CPU之间的数据传送。注意,CPU与外设之间每传送1个单位数据,就必须进行(2)~(6)5步一次,若传送100个单位数据,则要中断CPU100次,执行100次中断服务程序。采用程序中断实现输入/输出较好地解决了慢速外设和高速主机之间的矛盾。在启动外设后,外设致力于本身的机电输入/输出操作,而主机则可执行现行程序。由于外设的机电操作速度较慢,因而在这段时间内主机可执行相当多的指令,直到外设发出中断请求并被响应为止,可见,这样使主机和外设的操作达到一定的并行程度。只有主机响应外设的中断请求,并执行中断服务程序以完成主机和外设之间的数据交换时,中断服务程序与主机的现行程序才是串行的。即这期间主机才停止执行现行程序。

此外,若多个外设同时工作,如图6.17所示。主机则可根据各外设的工作频率即各自的优先级别先后响应它们的中断请求,分别执行不同的服务程序以完成各外设的数据交换。因此,程序中断方式也使多个外设的并行工作成为可能。

3. 程序中断传送接口举例

图6.19是某机所配X-Y记录仪的程序中断接口。接口下部与X、Y数据缓冲器连接

的部分为非标准部分,功能是接收并寄存主机要输出的点的坐标数据。其余为程中传送标准接口,为深入掌握程中传送标准接口和说明 X-Y 记录仪程序中断传送的方便,对其进行较详细介绍。设备选择器选中设备,产生设备选中信号 SEL 并去控制本接口的数据输出、中断控制和状态触发器 BUSY、DONE。4 个控制与状态触发器 Busy、Done、INTR 和 MASK 为状态与控制命令寄存器的一部分。Busy 是忙闲状态触发器,“1”为忙,“0”为闲;Done 是完成触发器、亦称就绪或准备好触发器(相当于中断触发器),为“1”表示一次传输完成;INTR 是中断请求触发器,“1”表示有中断请求,“0”表示无中断请求;MASK 是中断屏蔽触发器,“1”表示屏蔽中断,即使 Done 为“1”,也不允许 INTR 发出中断请求,为“0”时,才允许发出中断请求。中断控制、中断排队和设备码回送属于中断控制逻辑,它们将分别实现中断查询、中断选优判优、中断响应、中断向量地址的回送等工作。

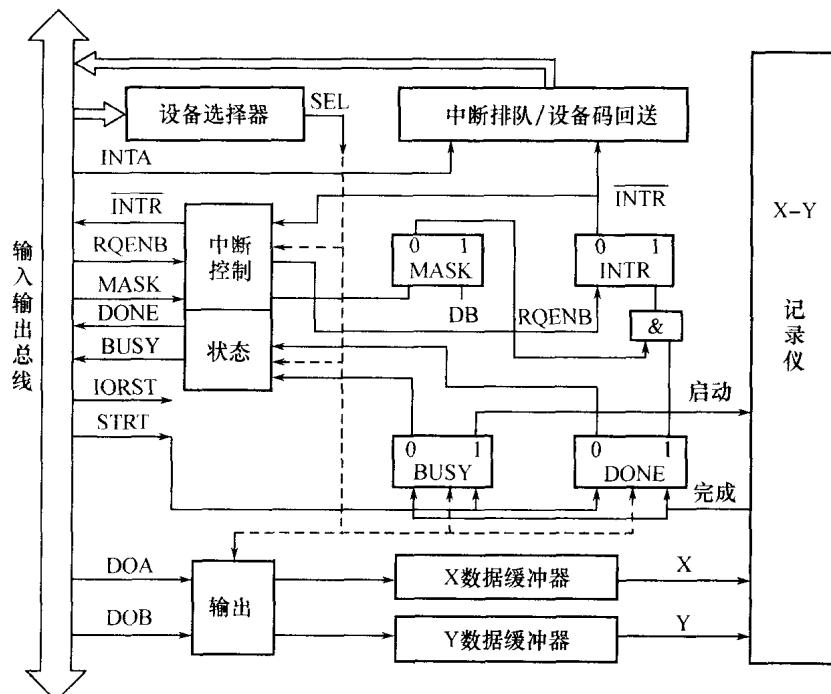


图 6.19 X-Y 记录仪程序中断传送接口

下面简述 X-Y 记录仪画一曲线的工作过程。CPU 设置好存储该曲线信息的主存始地、信息长度后并通过 I/O 指令查询 X-Y 记录仪是否良好和闲置,若为闲且良好,则由程序对接口初始化。因 X-Y 记录仪为输出设备,故还需 CPU 通过 I/O 指令先向 X-Y 记录仪的数据缓冲器发送 1 组数据 X 和 Y,而后启动 X-Y 记录仪工作。CPU 经总线向所有外设接口发送设备地址码,经设备选择器译码只有 X-Y 记录仪的程序中断接口产生选中信号 SEL,即 CPU 此时发出的输入/输出控制,只对 X-Y 记录仪有效。因此该接口中的 BUSY

被置“1”、DONE 被清“0”，X-Y 记录仪工作。X-Y 记录仪从接口中 X、Y 两个数据缓冲器取走数据并绘图的同时，产生完成信号将 DONE 置“1”、BUSY 清“0”，表示本次输出操作完成，并再次产出中断，要求 CPU 提供下 1 组数据 X 和 Y。CPU 在每条指令执行之初均发出中断查询信号 RQENB，如该接口未被屏蔽，将中断请求触发器 INTR 置“1”，从而向 CPU 发出中断请求 INTR。CPU 接到 INTR，如可以响应中断，则发中断响应信号 INTA。此时若该接口处于优先，则实施对 X-Y 记录仪的中断响应。即执行中断隐指令，暂停现行程序，将 X-Y 记录仪的设备地址码回送给 CPU，回送的设备地址码形成向量地址，从而转入为 X-Y 记录仪服务的中断服务程序。若曲线未画完，则中断服务程序再次向接口发送 1 组数据 X 和 Y，并再次启动 X-Y 记录仪工作，而后自动返回暂停程序继续执行。X-Y 记录仪再次从 X、Y 数据缓冲器取走数据，又一次发出中断，……，如此多次中断，多次执行中断服务程序直至整个曲线画完为止。

由于 X-Y 记录仪是机电装置，每进行一次操作的时间是数百毫秒，因此记录仪在画图的动作时段内，CPU 可运行现行程序。只有在记录仪操作完毕，向 CPU 发中断请求后，CPU 才将视情况转入中断服务程序，即再输出数据。因此，通过程序中断接口控制它们之间的数据传送，可在一定程度上实现 CPU 与外设，以及外设与外设之间的并行操作。

6.4.4 可编程中断控制器 8259A 简介

程序中断传送标准接口已由 Intel 公司于 20 世纪 70 年代末制成专用集成电路芯片——可编程中断控制器 8259A，其为双列直插式芯片。它允许用户通过编程设定多种工作状态和工作方式，使用灵活方便、实用性强。其有如下特点：每片可管理 8 级（个）外部中断，且每级中断可单独被屏蔽；具有 4 种优先权管理方式；具有级连功能，通过多片 8259A 级连，最多可控制 64 级中断，其中一个主片多个从片；可直接向 CPU 提供中断类型码，即向量地址。将 8259A 与系统总线连接，即可实现外部中断的处理。

1. 8259A 的内部结构及功能

8259A 的内部结构及引脚如图 6.20 所示。

(1) 数据总线缓冲器：其为 8 位双向三态缓冲器，它是系统数据总线与 8259A 的接口，暂存内部总线和系统总线传输的信息。系统编程时要写入的控制命令字、8259A 生成的中断类型码及状态、CPU 或外设要传送的数据等均通过它的 D₀ ~ D₇ 双向数据引脚与系统总线连接。

(2) 读写控制电路：它负责接收地址信息，选中相应的内部寄存器；接收读/写命令以控制读/写，其对外有 4 个引脚 WR、RD、CS 和 A₀。CS 为片选引脚，该输入信号由系统地址译码产生，只有 CS 为低电平时，8259A 才会处于工作状态；WR 为写控制引脚，低电平有效，用来控制 8259A 从系统总线接收信息；RD 为读控制引脚，低电平有效，用来控制 8259A 将内部某一寄存器的内容送系统总线；A₀ 为地址引脚，其输入来自系统地址 A₀ (8088 微机系统) 或 A₁

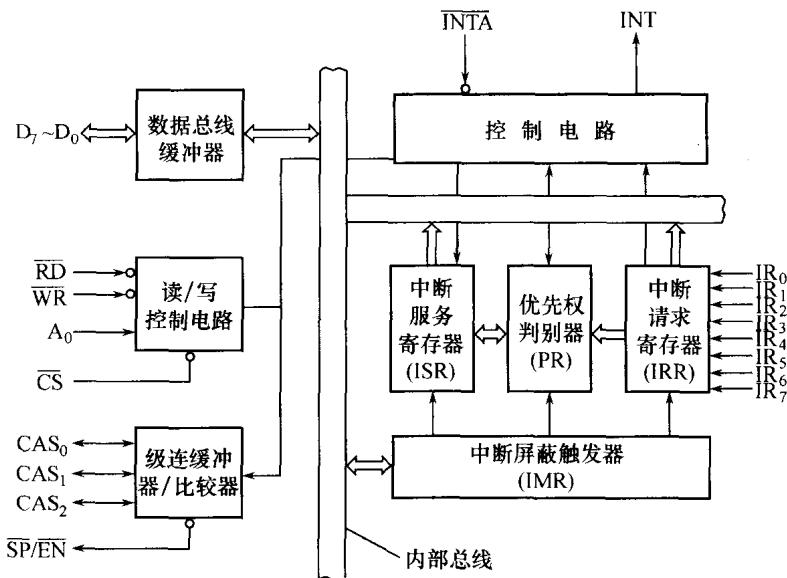


图 6.20 8259A 的内部结构及引脚

(8086 微机系统)。用于选择 8259A 内部各寄存器。

(3) 中断控制逻辑

① 中断请求寄存器 (Interrupt Request Register, IRR)：8 个触发器构成，保存中断信号 $IR_0 \sim IR_7$ 。是否发出中断请求，由中断屏蔽字决定。

② 中断屏蔽寄存器 (Interrupt Mask Register, IMR)：8 个触发器构成，保存中断屏蔽字，其内容可由用户编程决定。它的每一位与 IRR 的相应位对应，当某一位为“1”时，则封锁 IRR 中相应位的中断请求，使之不能发出中断请求，即中断请求无效。

③ 中断服务寄存器 (In_Service Register, ISR)：8 个触发器构成，保存正在被服务中断源的优先权编码。

④ 优先权判别器：选出优先中断源。若当前 CPU 没有执行中断服务程序，优先权判别器将对 IRR 中的各个有效中断请求进行判优，并选出优先中断源。在中断响应周期将该优先中断源的优先权编码送 ISR。若 CPU 正在执行某一中断服务程序，同时又有新的中断请求，该判别器将把新中断请求的优先权编码与 ISR 中的优先权编码比较，若优先权级别高于正在被服务的中断源，则中断现行的中断服务程序，响应更高级的中断；否则，对新的中断请求不予响应。

⑤ 控制电路：依 IRR、IMR、ISR 的状态和优先权判别器的工作情况，由 INT 引脚向 CPU 发出中断请求信号；接收 CPU 发出的中断响应信号；生成优先中断源的中断类型码并送往 CPU；清除正被服务中断源的中断请求位。

(4) 级连缓冲器/比较器：控制多片 8259A，以使 8259A 适用于中断源超过 8 个的场合。

为实现多片 8259A 的互连,该部分对外有 3 个级连信号引脚 CAS₀、CAS₁、CAS₂ 和 1 个控制引脚 SP/EN 且它们都是双向的,既可作输入端又可作输出端。一片 8259A 最多能控制 8 个外部中断,多片 8259A 级连最多可控制 64 个外部中断。多片 8259A 级连时有主片、从片之分,且只允许 1 个主片,最多 8 个从片。

2. 8259A 编程简述

8259A 为可编程芯片,可编程指的是通过编写程序设定芯片工作状态和工作方式。8259A 编程分初始化和工作方式两类编程。初始化编程就是通过程序向 8259A 的初始化命令字(Initialization Command Word, ICW)寄存器写入信息。8259A 设有 ICW₁、ICW₂、ICW₃ 和 ICW₄ 4 个初始化命令字寄存器,8259A 工作前 ICW₁、ICW₂ 必须写入所需信息,而 ICW₃ 和 ICW₄ 可依工作方式选择不同决定是否写入。工作方式编程,顾名思义即通过程序向工作方式命令字(Operation Command Word, OCW)寄存器写入信息,以实现对中断过程的动态控制。8259A 设有 OCW₁、OCW₂、OCW₃ 3 个工作方式命令字寄存器。工作方式编程的具体工作就是向 8259A 写入工作方式命令字。应当指出的是,ICW 和 OCW 在编程要求上是不同的。初始化命令字 ICW 通常是在系统启动时由初始化程序一次性写入的;而对于工作方式命令字 OCW,只要端口地址能够满足 8259A 引脚 A₀ 要求的条件,即可由程序在任何需要的时刻多次写入。ICW、OCW 具体的含义、功能及编程方法本书不作详细介绍,欲知其详的读者,请查阅 Intel 微机的教科书或有关文献。

3. 级连的实现

假定是 1 个主片、8 个从片的最大级连的情况。级连时主 8259A 的 3 个级连信号引脚 CAS₀、CAS₁、CAS₂ 与 8 个从 8259A 的相应端连接;主 8259A 的引脚 IR₀ ~ IR₇ 分别与从 8259A 的引脚 INT 连接。工作前,每一片都应分别初始化和设定工作状态。

8259A 与总线的连接分为缓冲和非缓冲两种方式。在非缓冲方式下,8259A 的数据引脚直接与系统的数据总线连接,此时由控制引脚 SP/EN 输入电平的高低区分主片和从片。若 SP/EN 为高电平,表明该片为主片;否则为从片。在缓冲方式下,8259A 的数据引脚通过外加的总线驱动器与系统的数据总线连接,此时由控制引脚 SP/EN 输出电平作为总线驱动器的启动信号。而多片级连时主、从片的区分将由初始化控制命令字 ICW₄ 的第 2 位(D₂)决定。若 D₂ 为“1”,表示该片是主片,否则该片是从片。而 ICW₄ 中的信息是系统工作前,用户对 8259A 编程时设定的。

当任一从 8259A 的 IR 引脚有中断请求时,则可通过该片的 INT 引脚传往主片相应的 IR 端,再由主片的 INT 引脚向 CPU 发出中断请求。若 CPU 给予响应,主片的 CAS₀、CAS₁、CAS₂ 将输出从片的编码,选中被响应中断的从片,该从片将相应中断源的中断类型码通过数据总线送给 CPU,从而进入相应中断服务程序,实现对该中断的处理。

4. 8259A 中断优先级选择方式

8259A 中断优先级选择有完全嵌套、轮换优先级 A、轮换优先级 B 和查询四种方式。

- 完全嵌套方式:是一种固定优先级方式,连到 IR0 的外设中断优先权最高,连到 IR7 的外设中断优先权最低。只要优先权高的中断请求线上有中断请求,或 CPU 正在为优先权高的中断服务,优先权低的中断请求就不会被响应。
- 轮换优先级 A 方式:完全嵌套方式对级别低的中断很不利,在某些情况下,可能会出现级别低的中断一直不能被处理,这将会导致计算机系统出现错误。而轮换优先级 A 方式使得每个级别的中断保证有机会被处理,它把给定的中断级别处理完后,立即把它放到最低级别的位置上去。
- 轮换优先级 B 方式:该方式允许 CPU 在任何时间规定某 IR 引脚为最低优先权,然后顺序的确定其他 IR 引脚上的优先权。

- 查询方式:该方式即通过程序查询确定优先中断源,以便为该中断源服务。它通过向 8259A 工作方式命令字 OCW₃ 写入控制信息,8259A 则给出一个 8 位的状态字,由该状态字确定系统有否中断,若有则给出产生中断的中断源编码,CPU 依此编码进入相应中断服务程序。状态字各位含义如下。

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁
I	未用	W ₂	W ₁	W ₀		

其中,I 为“0”,表示 8259A 无中断请求;I 为“1”表示有中断请求,且此时 W₂、W₁、W₀ 的值就是发出中断请求的中断源编码。

5. 8259A 屏蔽方式

8259A 提供简单屏蔽和特殊屏蔽两种方式。简单屏蔽方式提供了 8 位的屏蔽字,屏蔽字的各位分别对应 IR₀ ~ IR₇。屏蔽字被置“1”的任一位将禁止相应 IR 发出中断请求。特殊屏蔽方式允许低优先权外设中断高优先权外设的中断服务程序。当 8 位屏蔽字的某一位被清“0”时,则对应此位的中断可中断比它优先权高的中断服务程序。若屏蔽字为 11101111,则中断请求寄存器 IR₄ 对应的中断可中断任何高级别的中断服务程序。

8259A 两种屏蔽方式的设定是通过向工作方式命令字 OCW₃ 的 D₆D₅ 位写入控制信息实现的。若 D₆D₅ 为 11,表示 8259A 处于特种屏蔽方式;若 D₆D₅ 为 10,表示 8259A 处于简单屏蔽方式。

6.5 直接存储器访问 DMA

观察程序中断控制传送易知,主机和外设操作实现了一定程度的并行,提高了系统的效率。但是每交换一个单位数据,均要中断主机一次,且要花费相当数量的指令进行保护现场、设新屏蔽字、开关中断和恢复现场等工作。对于慢速外设,单位数据之间的时间间隔一般都在毫秒级。因此主机还能有一定的时间用于执行被暂停程序,故这种方式对于慢速外

设是可行的。然而象磁盘、光盘、激光打印机、图像处理、高速数据采集系统等高速外设, 单位数据之间的时间间隔是微秒级甚至更短, 且数据的交换又是大量的、成批的。如果还采用程序中断控制方式进行数据交换, 则可能出现两种情况; 若主机响应中断请求而致力于数据传送, 因单位数据之间的时间间隔很短, 甚至无法再利用, 故必须让一批数据交换完, 这样只要一交换数据, 便完全占用主机, 因而又回到完全串行工作状态。如果单位数据完成了交换的准备, 而主机一时不能马上响应请求, 便可能冲掉信息而造成数据丢失。因此, 高速外设一般不采用程序中断而采用 DMA 方式交换数据。

6.5.1 DMA 概述

1. DMA 和程序中断传送的数据通路

DMA 用于高速外设的成组数据传送, 是按照连续地址直接访问主存储器的。为减小程序或 CPU 的开销, 它开辟外设和主存的直接数据通路, 其数据通路和程序中断传送的数据通路是完全不同的。图 6.21 为 DMA 和程序中断传送的数据通路示意图。

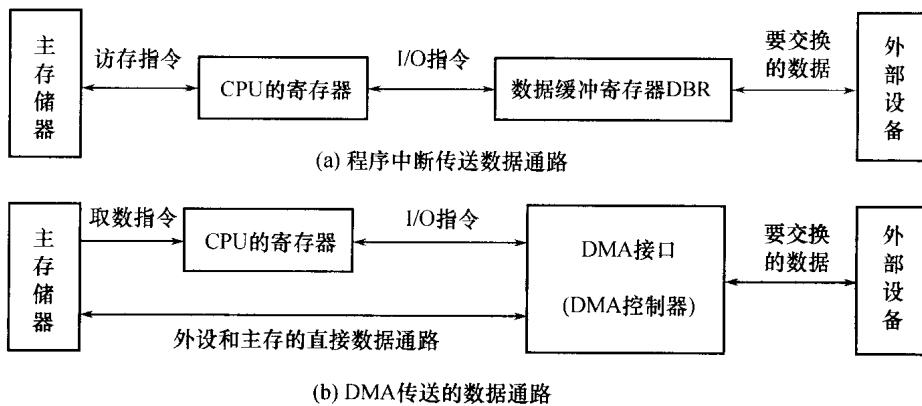


图 6.21 程序中断传送和 DMA 传送的数据通路

2. DMA 传送的特点

程序中断传送时, 若输入数据, 由外设读出的要交换的数据先存放到 DBR, 然后由中断服务程序中的输入指令送到 CPU 寄存器, 最后由存数指令送主存; 若输出数据, 主存中要交换的数据由中断服务程序中的取数指令先送到 CPU 寄存器, 然后由中断服务程序中的输出指令送到 DBR, 最后外设从 DBR 取走输出。程序中断控制的数据传送必须通过 CPU 寄存器和使用 CPU 的指令, 它是以 CPU 为中心的。

DMA 传送是以主存为中心, 它在外设主存间开辟了直接数据传输通路, 且该通路是完全用硬件实现的。主要体现在, 主存流动地址的确定、传输长度的计数与控制、主存与外设间的数据传送等均由 DMA 接口的逻辑电路完成。由于靠硬件、不必执行中断服务程序交换

数据,故无须占用 CPU 的寄存器及程序计数器,这样省去了保存断点、保护和恢复现场、设新屏蔽字等大量操作。因此一方面保证了 CPU 程序的执行,另一方面也加快了外设与主存之间数据交换的速度。

由图 6.21(b)可知,在 DMA 传送的数据通路中,仍保留着 CPU、外设间的程序中断通路。注意,此通路不是用来传送数据,而是用来给 DMA 接口指定传输参数,CPU 查询、启停外设和 DMA 数据传输结束时使用。因 DMA 方式交换一批数据的开始前和结束后仍要通过程序进行处理。开始前,CPU 需了解外设和接口的有关状态即查询外设是否良好闲置。若良好空闲,向 DMA 接口送传输时所需参数,如主存缓冲区首地址、传输长度、外设内部地址(如磁盘的柱面、盘面、扇区、磁带的组号等)和外设工作方式(如读、写、寻道、走带等)等。若为输出,还需将首次输出的单位数据送 DMA 接口中的数据缓冲寄存器,而后启动外设进行数据传送。上述工作需通过一段程序实现。在一批数据传送结束后或发生故障时,则通过程序中断方式告知 CPU,以便 CPU 响应中断,执行中断服务程序,决定 DMA 是继续传送数据?还是停止传输或是进行故障处理。

3. DMA 工作方式

DMA 传送是以主存为中心,即 CPU 和与 DMA 连接的设备共享主存。当两者同时要求访存时,出现冲突怎么办,通常要根据优先级进行排队和处理,快速外设的优先级高于低速外设,输入/输出的优先级高于解题程序。实际上,通常有三种处理方式。

(1) 全串行方式

对于有些传输率极高的外设,由于成批的数据传送需不断地访问主存,为保证传送可靠进行而采用 DMA 和 CPU 串行工作方式。通常的做法是:当外设要求传送数据时,由 DMA 接口发信号给 CPU,要求 CPU 放弃它对地址总线、数据总线及有关控制线的使用权,获得对总线控制权后开始数据传送。只有当一批数据传送完后,DMA 接口才告知 CPU 可以访存并把总线控制权交回给 CPU。

这种方式的优点是控制简单。但由于是全串行方式,DMA 在访存期间,CPU 基本上处于不工作状态或保持原状态。外设,即使是高速外设,两个单位数据的准备间隔时间一般也总是大于存储周期,存储器的效能未能充分发挥。因而在 DMA 访存时,主存总有一部分周期是空闲的,这些空闲的周期 CPU 不能加以利用,不能不说是一种浪费。解决的办法是在全串行方式 DMA 接口中,往往设置一个小容量的半导体存储器。外设先与小容量的半导体存储器交换信息,然后小容量的半导体存储器再与主存交换信息,这样就减少了 CPU 暂停工作时间,同时也提高了存储器的效率。

(2) 周期挪用(亦称周期窃取)

当 DMA 没有请求时,CPU 按要求访问主存,当 DMA 有请求,则可挪用一个或若干个存储周期,这就是周期挪用。

DMA 要求访存可能有三种情况,一是 CPU 此时无需访存,如执行除法指令,指令执行时

间较长,于是二者没有冲突,DMA 挪用一、二个存储周期对 CPU 执行程序没有影响。二是 DMA 要求访存时,CPU 正在访存,此时须待 CPU 访存结束,DMA 才能占用总线访存。三是 DMA 要求访存时,CPU 也要求访存,这就发生了冲突。毫无疑问,此时 DMA 必须优先于 CPU 访存。因为外设的工作特性决定了前一个数据必须在 DMA 下一个访存请求到来之前存取完毕,否则将会丢失数据。因此,这种情况下的周期挪用延缓了 CPU 指令的执行。

显而易见,周期挪用既实现了数据的 I/O 传送,又保证了 CPU 执行程序。因它较好地发挥了主存和 CPU 的效率,故广为采用,通常说的 DMA,即指这种方式。周期挪用或周期窃取又称之为简单中断。应当指出,DMA 每挪用一个存储周期,都要申请总线控制权、建立总线控制权和归还总线控制权。尽管传送一个单位数据仅占用一个存储周期,但对于 DMA 接口而言,因接口逻辑线路的延迟,故 DMA 设备的读/写周期(准备一个单位数据的时间)应大于存储周期。

(3) 交替访存方式

如果系统中 CPU 的工作周期(机器周期)比主存存储周期长得多,则可采用 CPU 和 DMA 交替访存的方式。这种方式不需要总线使用权的申请、建立、和归还过程,具有很高的 DMA 传送效率。

在这种方式中,CPU 和 DMA 各有自己的内存地址缓冲寄存器、数据缓冲寄存器和读/写信号控制触发器。由于此时的 DMA 传送对 CPU 无任何影响,因而 CPU 感觉不到 DMA 的存在,故又称为透明的 DMA。

透明的 DMA 既不停顿主程序的执行,又能保证 DMA 的完成,是一种高效率的方式。当然,要使两者访存配合得天衣无缝,其硬件逻辑显然最为复杂,且要求存储器速度快。

6.5.2 DMA 接口的基本组成

DMA 传送以主存为中心,完全用硬件开辟外设主存间直接数据传输通路。因此,DMA 接口应具有完成如下功能的硬件,即 CPU 查询外设、启停外设和向 DMA 接口设置传输时所需参数硬件;为数据传送时申请、建立和归还总线控制权、修改某些参数和对传送全程进行控制的硬件;为数据传送结束或数据传输过程中出现错误,向 CPU 发中断请求处理的硬件;以及为 DMA 接口解决多台 DMA 设备,同时发出 DMA 传输请求时判优选优的硬件。为此,周期挪用 DMA 接口应包括如下几个部分,如图 6.22 所示。

1. 接口寄存器

(1) 地址缓冲寄存器(Address Buffer Register, ABR):存放主存起始地址。传送开始前,程序将主存缓冲区首地址送入 ABR;在 DMA 传送期间,每交换一个单位数据,由硬件逻辑将其自动加 1,成为下一次数据的主存地址。直至一批数据传送完为止。因此又称它为主存流动地址寄存器。

(2) 字计数器(Word Counter, WC):传送开始前,由程序将要交换的数据个数送入 WC,

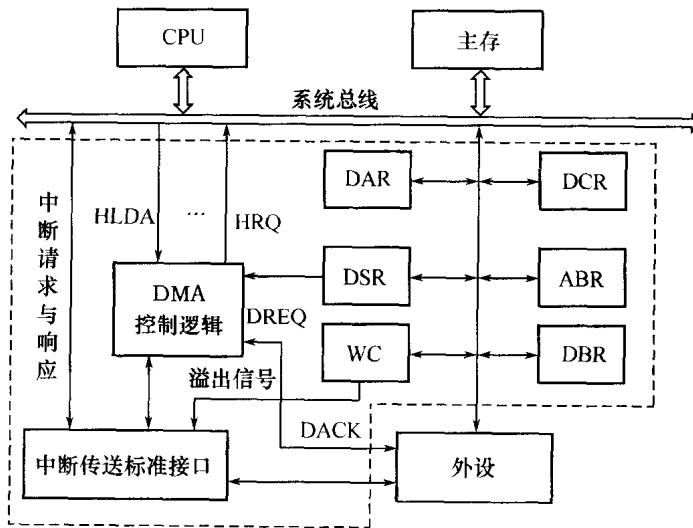


图 6.22 周期挪用 DMA 传送接口基本组成

一般送入的是数据个数的补码。DMA 传送期间,每交换 1 个单位数据,硬件逻辑将其自动加 1,直到一批数据交换完毕,WC 产生字溢出信号,向 CPU 发出结束中断请求。

(3) 数据缓冲寄存器 (Data Buffer Register, DBR): 它用于寄存 DMA 一次传输的数据,即单位数据,一般是读写存储器一次的数据量。有些设备读/写时,一次读写的数据量较小,如 1 位或字节长度。则接口中还应有装配和分拆数据的硬件逻辑,如具有移位功能的寄存器、字节计数器等。

(4) 设备状态寄存器 (Device Status Register, DSR): 寄存设备及接口的有关状态,是 DMA 传输前和发生故障时 CPU 需要了解的,一般通过程序中的输入指令取状态。

(5) 设备控制寄存器 (Device Control Register, DCR): 寄存 CPU 启动接口及设备的有关控制命令,如读/写或寻道等,在交换前由程序送入该寄存器。有的接口将状态和控制两个寄存器合为一个寄存器。

(6) 设备地址寄存器 (Device Address Register, DAR): 通常存放设备内部地址,如:磁带机的组(区)号或磁盘的柱面号、盘面号、扇区号。有的接口该寄存器还存放外设地址。有的计算机将该寄存器作为设备控制器的组成部分

2. 中断传送标准接口

构成见 6.4.3 节。功能有二,一是当 WC 产生溢出说明一批数据传送完或传输中出现故障,由它向 CPU 提出中断请求,让 CPU 完成 DMA 传送的后处理。这里的中断与 6.4.3 节介绍的中断技术相同,但目的不同。后者的目的是数据传送,而此处的目的是传送的后处理。它们是功能完全不同的中断事件;二是 DMA 传送前的所有准备工作也要借助于它实现,如:查询、启停外设,设置传输参数等。

3. DMA 控制线路

它负责管理 DMA 的数据传送过程。一般由单位数据准备就绪、DMA 请求和响应等多个触发器, DMA 优先排队、DMA 向 CPU 发请求使用总线和从 CPU 接收允许使用总线的线路, 以及回送命令码等线路组成。每当外设准备好 1 个单位数据或 1 个单位数据输出结束, 外设向 DMA 控制逻辑提出传输请求 DREQ, DMA 控制逻辑则向 CPU 发出申请总线使用权信号 HRQ。CPU 响应 DMA 请求, 为 DMA 建立一个 DMA 周期, 并向 DMA 控制逻辑发响应信号 HLDA。DMA 控制逻辑又向外设发出 DACK, 告知外设可以传输。DMA 接口就是依靠 CPU 建立的 DMA 周期完成外设与主存的直接数据传送的。DMA 周期除完成访存外, 还应实现回送命令码及 ABR、WC 分别加 1 等微操作。DMA 周期和取指周期、执行周期、中断周期一样, 都是 CPU 的工作周期。

全串行 DMA 接口与周期挪用接口类似, 但要简单些, 其传送一批数据, 仅向 CPU 发请求使用总线一次。透明 DMA 接口要复杂些, 与周期挪用接口的差异究竟有哪些, 请读者自己思考。

6.5.3 DMA 的工作过程

1. DMA 的传送过程

分传送初始化、数据传送和结束处理三个阶段, 其传送流程如图 6.23 所示。

(1) 传送初始化: CPU 通过一条 I/O 指令取状态, 查询外设是否闲置良好。若良闲, 则再通过几条 I/O 指令为 DMA 接口预置初值和传送需要的有关参数, 并启动外设。

(2) 数据传送: DMA 传送为成组传送, 下面结合图 6.22 并以周期挪用 DMA 为例, 说明数据传送过程。

- 外设被启动后, 若为输入数据, 则要作如下操作:

① 从输入介质读入 1 个单位数据送到数据缓冲寄存器 DBR, 表示准备就绪。如果外设是面向字符的, 则组成一个单位数据需经适当的装配过程;

② 外设向 DMA 接口发请求 DREQ, DMA 接口向 CPU 申请总线控制权 HRQ;

③ CPU 发回 HLDA 信号, 表示 DMA 接口已取得总线控制权, 同时由 DACK 信号通知外设已被授予 1 个 DMA 周期, 可以挪用 1 个存储周期传送 1 个单位数据;

④ 将 DMA 接口 ABR 中的主存地址、DBR 中的单位数据分别经地址、数据总线送主存的 MAR 和 MBR。并回送写存储器命令码到主存;

⑤ 将 WC、ABR 的内容分别加 1, 给出下 1 个单位数据的地址。同时启动写操作, 将数据写入主存指定存储单元。注意: 初始化时, 送入 WC 的是传输长度取反加 1, WC 减 1 变为加 1;

⑥ 判断 WC 是否溢出, 若无溢出, 检查也无错误, 准备下 1 个单位数据的输入, 否则, 向 CPU 发错误中断请求。若 WC 溢出, 表明一批数据已交换完毕, 此时应置结束状态标志, 向

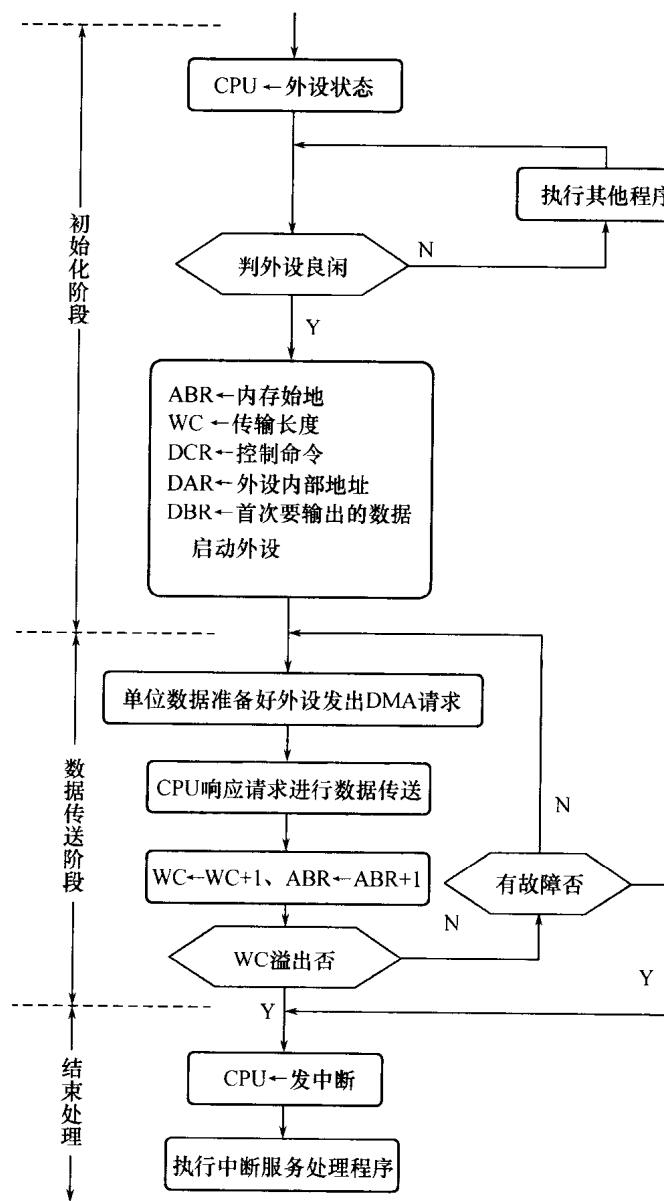


图 6.23 DMA 数据传送流程

CPU 发结束中断请求。

- 若为输出数据，则应做如下操作：
 - ① 当 DMA 接口数据缓冲寄存器 DBR 中的单位数据已被外设取走，表示准备就绪，外设可以再接收 1 个单位数据；
 - ② 外设向 DMA 接口发请求 DREQ，DMA 接口向 CPU 申请总线控制权 HRQ；

③ CPU 发回 HLDA 信号, 表示 DMA 接口已取得总线控制权, 同时由 DACK 信号通知外设已被授予 1 个 DMA 周期, 可以挪用 1 个存储周期传送 1 个单位数据;

④ 将 DMA 接口 ABR 中的主存地址经地址总线送主存的 MAR。并回送读存储器命令码到主存;

⑤ 启动读操作, 将主存指定存储单元的内容读到主存 MBR 中;

⑥ 将主存 MBR 中的数据经数据总线送 DMA 接口的 DBR;

⑦ 将 WC、ABR 的内容分别加 1, 给出下 1 个单位数据的地址, 同时将 DBR 的内容写到输出介质上, 若外设为面向字符的设备, 则需将 DBR 内容分拆成字符写入介质;

⑧ 判断 WC 是否溢出, 若无溢出, 此时检查也无错误, 准备接收下 1 个单位数据, 否则, 向 CPU 发错误中断请求。若 WC 溢出, 表明一批数据已交换完毕, 此时应置结束状态标志, 向 CPU 发结束中断请求。

(3) DMA 结束处理: CPU 响应 DMA 的中断请求, 暂停现行程序转去执行中断服务程序。中断服务程序从 DMA 接口的 DSR 中取出状态, 进行判断, 若为传输错引起的中断, 则转错误诊断及处理程序。若需继续传送, 则再次对 DMA 接口初始化; 若不需要再传送, 则停外设。

2. 程序中断传送与周期挪用 DMA 传送的区别

(1) 程序中断传送主要靠软件实现数据传送, DMA 主要靠硬件实现数据传送。所以程序中断传送速度慢, 而 DMA 传送速度快。

(2) DMA 请求的响应, 只挪用 1 个存储周期。若此时正逢 CPU 不访存, 不影响 CPU 执行现行程序, 这使 CPU 操作与 DMA 传送重叠; 若 CPU 也要访存, 只要求 CPU 暂停 1 个存储周期, 实现 1 个单位数据的传送, 而后 CPU 继续执行程序。DMA 传送开始和结束两个阶段由 CPU 执行一段程序, 在整个数据传送期间不占用 CPU 硬件资源。程序中断传送则中止现行程序的运行, 转去执行中断服务程序, 实现 1 个单位数据的传送。在两个程序的切换时, 要花费较多指令保存现场、设新屏蔽字、开关中断和恢复现场。因此, 程序中断传送的效率低。

(3) 程序中断传送只适用于慢速外设, DMA 适用于高速成组传送的外设。

(4) 程序中断传送的响应必须在一条指令执行之末, 而 DMA 原则上可在 CPU 不访存的任何时刻得到响应。

(5) DMA 的优先权比程序中断传送的优先权高。中断的功能强, 可处理各种异常或复杂突发事件, 而 DMA 只适合于对数据传送的控制。

6.5.4 DMA 传送接口类型

1. 广义型 DMA 传送接口

如图 6.22 所示是计算机早期经典 DMA 接口的基本组成。现代计算机采用大规模集成电路技术, 为设计标准化、模块化和与具体外设尽量无关, 通常把经典 DMA 接口分立为

DMA 控制器和 DMA 接口两个部件,如图 6.24 所示。DMA 控制器是公共逻辑,与具体外设无关,只要以 DMA 方式与主机交换信息,就必须配置 DMA 控制器。DMA 控制器通常由 DCR、ABR 和 WC 以及中断传送标准接口、DMA 控制逻辑和时序电路组成。其功能是控制 DMA 设备传送的全过程。DMA 接口体现外设的特定要求,与 DMA 连接的外设不同,其接口亦会存在较大差异。因此,设计 DMA 接口时,在满足外设要求的前提下,应尽量简化。DMA 接口一般应包括 DBR、DAR、DSR 和 DMA 请求等逻辑,其功能是在 DMA 控制器的控制下完成数据传送。称由这两个部件组成的 DMA 接口为广义型 DMA 传送接口。

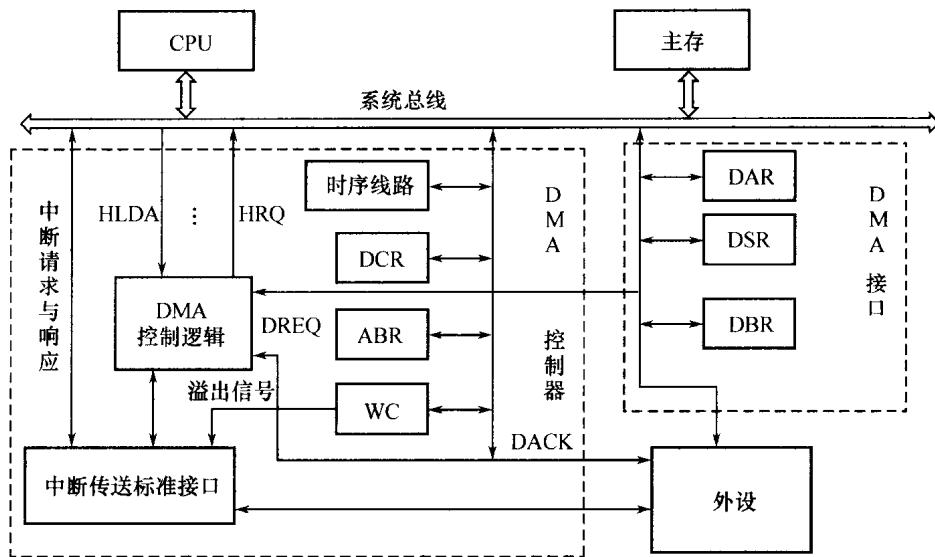


图 6.24 广义型 DMA 传送接口组成示意

广义型 DMA 接口增加时序线路目的是,当 DMA 取得总线控制权进入 DMA 周期传送数据时,不再由 CPU 产生微操作命令信号,而是由 DMA 控制器自身产生。使得 DMA 传送时占用 CPU 的资源更少了。

2. 选择型 DMA 传送接口

选择型 DMA 接口组成如图 6.25 所示。其与广义型 DMA 接口在组成上有两点不同,一是可连接多台高速外设,二是设备地址寄存器 DAR 的位数要长,因其除存放设备内部地址外,还应存放设备地址,以便选择多台外设中的一台。该种接口虽然在物理上可连接多台高速外设,但在逻辑上只允许连接一台外设。即在同一时间段内,DMA 接口只能为一台外设服务。关键是在初始化时将所选设备地址送入 DMA 控制器 DAR 中,据此选中某台外设。当被选中设备一批数据传送完后,CPU 才可重新初始化,选择另一台设备,不允许传送中切换设备。选择型 DMA 接口适用于数据传输率很高,甚至接近主存速度的外设。

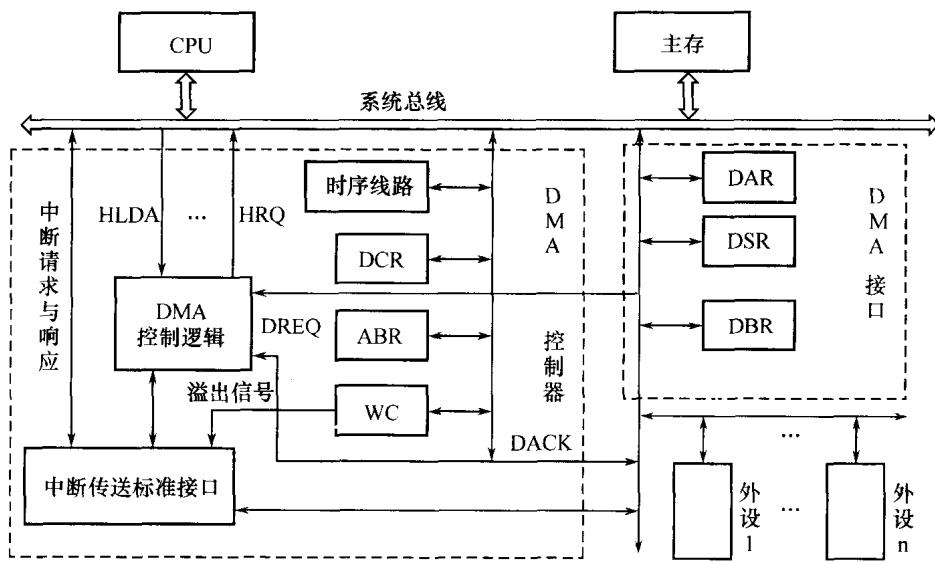


图 6.25 选择型 DMA 传送接口组成示意

3. 多路型 DMA 传送接口

多路型 DMA 传送接口如图 6.26 所示。其不仅在物理上可以连接多台外设,而且在逻辑上允许多台外设同时工作,各外设采用字节或字交叉方式进行数据传送。这种类型接口由一个 DMA 控制器和多个 DMA 接口组成,有多少个 DMA 接口,就可连接多少台外设。

接口中 DMA 控制器给每个与其连接的外设都配置了一套数据传送时所需寄存器,以便存储各自的参数。同时,附加了 DMA 优先权控制逻辑和专用的一组寄存器,前者解决当多个 DMA 外设同时提出传输请求时,由其决定先为哪台外设提供 DMA 服务;后者用于记录哪些设备有 DMA 请求、可对哪些设备的请求屏蔽、寄存对所连外设均起控制作用的命令信息和外设工作状态信息,如:优先级的设定方式、DREQ 和 DACK 什么电平有效及当前哪个外设传输结束等。多个 DMA 接口将依外设的不同而存在差别。通常应当设置数据缓冲寄存器或缓冲存储器、外设内部地址寄存器和状态寄存器,产生并发出 DMA 传送请求和接收 DMA 响应的逻辑,以及对外设读写和其他动作控制等逻辑。

这种类型接口适合于连接多个数据传速率不很高的外设。实用的该类接口往往兼有选择型 DMA 传送接口的功能,即也能进行成组传送。

上述三种 DMA 传送接口进行数据传送时,数据并不通过 DMA 控制器,而是经 DMA 接口与数据总线直接传送。DMA 控制器只控制数据传输,即负责、接管总线,并将总线控制权转交给 DMA 接口。Intel 公司生产的 8237,8257 和 8258 半导体芯片,其结构组成和功能非常类似于多路型 DMA 传送接口中的 DMA 控制器。

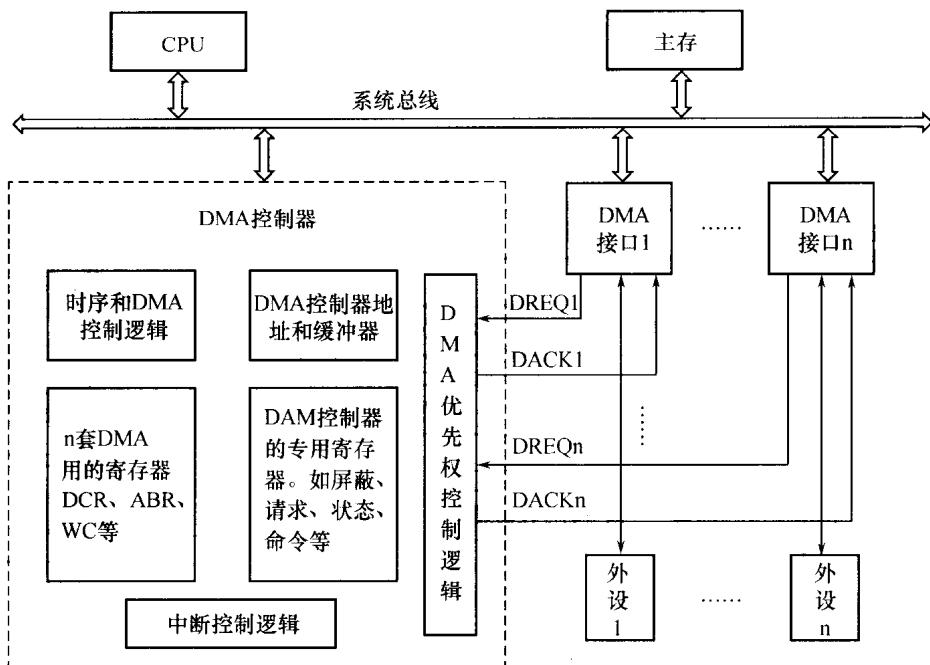


图 6.26 多路型 DMA 传送接口组成示意

6.5.5 DMA 传送举例

硬盘、软盘和光盘是各类计算机系统普遍使用的外存储设备。它们多采用 DMA 方式实现与主机之间的数据交换。

1. 硬磁盘 DMA 传送子系统

该子系统由选择型 DMA 传送接口、硬磁盘控制器及硬盘驱动器和盘组三大部分组成，如图 6.27 所示。

其通过系统总线与主机连接。选择型 DMA 传送接口组成及功能，6.5.4 节已谈及不在重述。硬盘驱动器及盘组第五章也已作过介绍，下面主要讲述硬磁盘控制器。硬磁盘控制器由数据缓冲寄存器 DBR1 和 DBR2、字计数器、磁盘内部地址寄存器、读写启停控制逻辑和硬磁盘适配器组成。输出（写盘）时，DBR1 先从选择型 DMA 传送接口中的 DBR 接收数据，然后再送 DBR2；输入（读盘）时，DBR1 先接收 DBR2 数据，然后再送到选择型 DMA 传送接口的 DBR 中。从原理考虑，DBR1 可不设置，设置的目的是，可延长主机响应磁盘 DMA 请求的时间，该时间为磁盘读或写 1 个单位数据的时间。DBR2 为可左移一位的数据缓冲寄存器，读盘时将串行读出的数据变换为字后送 DBR1；写盘时将 DBR1 送来的字变换为位，然后按位串行写入磁盘。字计数器记录读写的字数，当计满一个扇区所容纳的字数时，溢出修改扇区号。如果扇区容量为 512 位，字计数器应由 9 位触发器构成。磁盘内部地址寄存器初

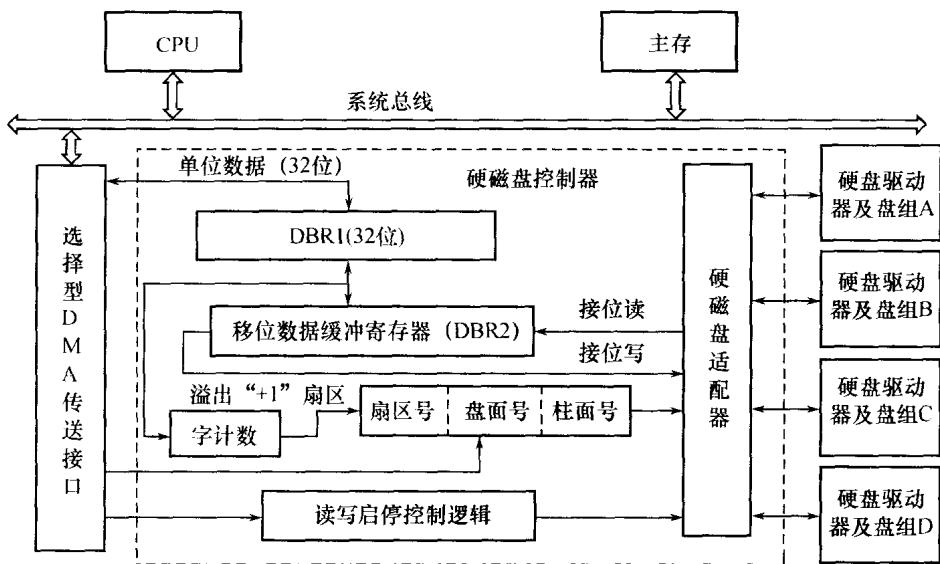


图 6.27 硬盘 DMA 传送子系统组成示意

始值来自选择型 DMA 传送接口中的 DAR，在传送过程中该值是变化的，它为读写盘提供地址信息。通常该寄存器不能用 DAR 替代，因传送出错时还需用到初始磁盘内部地址。读写启停控制逻辑接收接口的命令信息送硬磁盘适配器。

适配器 (Adapter): 所谓适配器是指处于两个不能直接连接的系统、机器或部件之间，确保它们相互可靠连接和通信的装置。硬盘适配器处于控制器的标准电路与驱动器的非标准电路之间，是选择、启动磁盘和读写磁盘的装置。它包括某种记录方式的编译码电路、扇区比较电路、读写电路、校验电路和电平转换匹配电路等。

2. 磁盘 DMA 传送的工作过程

(1) 初始化阶段: 在 DMA 操作开始前, 通过一段程序完成下述操作:

① 读取接口寄存器 DSR 中的磁盘及接口的状态信息, 确定磁盘是否闲置良好。若空闲良好, 则进行第二步, 否则, 执行其他程序;

② 送台号、柱面号、盘面号(磁头号)和起始扇区号到接口中的 DAR, 供选择磁盘和磁盘定位使用;

③ 分别送读写等命令信息、主存始地址和传输的字数到接口中的 DCR、ABR 和 WC;

④ 若为写盘, 应将第一个要写入的单位数据(32位字)送接口中的 DBR;

⑤ 启动磁盘, 而后继续执行其他程序。

(2) 数据传输阶段: 在初始化操作完成后, 磁盘开始工作, 找到要读写的扇区, 每写入或读出 1 个 32 位字, 接口向 CPU 发一次 DMA 请求, CPU 响应 DMA 请求, 暂停 1 个存储周期以完成数据交换。DMA 请求处理完毕, CPU 仍继续执行程序。由于磁盘工作频率总是大大低

于 CPU 工作频率,故每交换 1 个 32 位字,即两次 DMA 请求期间 CPU 仍可执行多条指令。如此不断地 DMA 请求,不断地周期挪用以交换数据,挪用间隙 CPU 又不断地执行指令。这样,在 DMA 工作时,无需执行中断服务程序,因而减少了对 CPU 的打扰。使得 CPU 执行指令和磁盘的读写操作得以并行进行。

(3) 结束阶段:数据传送完成或传送中出错均要向 CPU 发中断请求。CPU 接到中断请求后,读取接口 DSR 中的状态信息,以确定是错误中断还是正常结束中断。如果是错误中断,则转相应出错处理程序;如果是正常结束中断,则转正常结束中断服务程序,或继续下一批数据交换,或交换结束关闭磁盘。

6.6 I/O 通道

对于高速外设的成组交换,采用 DMA 方式不但节省了 CPU 的开销,而且提高了系统的吞吐能力。在小型、微型计算机中,采用程序中断和 DMA 两种方式进行系统的输入/输出处理,是行之有效的。但是在大、中型计算机中,外设配置多,数据交换频繁,如仍采用 DMA 方式处理高速外设的输入/输出,则明显会存在下述问题:

(1) DMA 传送效率的提高是以配备专用控制硬件——DMA 传送接口作为代价的。在大、中型计算机中,如果为数量众多的外设都配置 DMA 传送接口,则将大幅度增加硬件,因此很不经济。

(2) 如众多的外设都采用 DMA 方式,其访存冲突增加,将使 I/O 控制变得十分复杂。况且,接连不断的周期挪用也将降低 CPU 执行程序的效率。

(3) 每一台外设的 DMA 均要使用较多的 I/O 指令进行初始化,这样势必会占用 CPU 更多的时间。

在大、中型计算机系统中,为避免上述弊病,多采用 I/O 通道来组织输入/输出。

6.6.1 I/O 通道概述

1. 通道计算机的系统结构

具有 I/O 通道的计算机一般是大、中型计算机,且 I/O 通道是计算机系统中代替 CPU 管理、调度和控制外设的独立部件,是 4 级 I/O 子系统的核心装置。一个通道可以连接多个设备控制器,从而控制多台外设与主存交换信息。一般说来,具有通道的计算机系统的结构如图 6.28 所示。其有如下特点:

(1) 与一般的单总线结构不同,它有系统总线和通道总线两类总线。系统总线承担通道与主存、CPU 与主存之间的数据传送;I/O 总线即通道总线承担着外设与通道之间数据传送。两类总线可分别按各自时序同时工作。

(2) 按通道的工作方式,通道可分为字节多路通道、选择通道和数组多路通道三种类

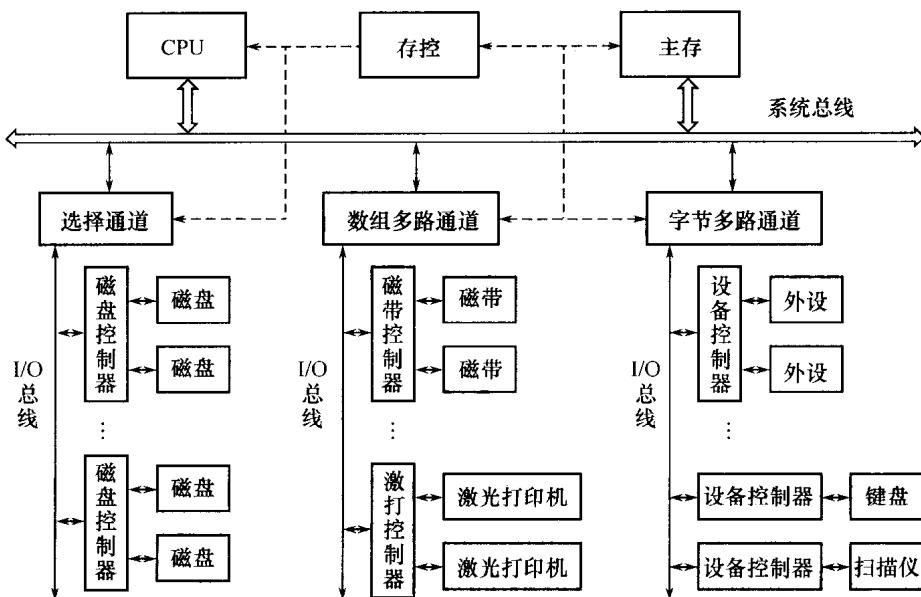


图 6.28 通道计算机的系统结构示意

型。一个系统通常可以兼有三种类型的通道,亦可只有其中的一种或两种。之所以配置多种通道,是因为系统数据流量很大时,若所有外设都接在一个通道上,则通道有可能成为限制系统效能的瓶颈。而且,设立多个通道便于对不同类型外设分类管理,以提高整个 I/O 子系统的效能。

(3) 系统中设有存控部件,其主要任务是根据预先测定的优先次序,决定下一周期由哪个部件使用存储总线访问主存。当通道与 CPU 同时请求访存时,通道优先于 CPU。当多个通道请求同时访存时,选择通道、数组多路通道优先于字节多路通道。

2. CPU 对通道的管理与服务

CPU 对通道的管理与服务是通过执行通道 I/O 指令、编写通道程序和处理来自通道的中断实现的。

通道 I/O 指令是 CPU 指令系统的一部分,是专门控制输入/输出操作的指令。CPU 通过了解查询通道、外设状态和启停通道的通道 I/O 指令管理通道;通过运行操作系统的设备管理程序为用户编写通道程序。通道程序一般存储到主存指定位置,若通道专设存储器,则存入该存储器。大多数计算机都是在主存中开辟专用区域存储通道程序,而不专设存储器。

通道的中断分为两种,一种为数据传输结束中断,另一种为故障中断。CPU 响应中断后,通过查询通道、外设状态确定是哪一种中断,以便执行相应中断服务程序,决定通道后续还需做什么工作。

3. I/O 通道的功能

- (1) 接收通道 I/O 指令，并按指令要求向指定通道和外设发出操作命令。
- (2) 指出外设读/写信息的所在位置，即提供外设内部地址，同时指出与外设交换信息的主存首地址和传输长度。
- (3) 执行通道程序，控制外设与主存之间的数据交换，不但提供数据缓冲，而且完成要交换信息的分拆与装配。
- (4) 接收外设、子通道 (Subchannel, SCH) 的状态信息，形成并保存通道本身的状态信息，以便 CPU 随时查询和了解。
- (5) 将外设、子通道和通道的传输结束中断和故障中断的请求，排队并按序及时告知 CPU。

因此，通道应具有与 DMA 相类似的硬件结构。它不但承担了 DMA 的功能，承担了 CPU 对 DMA 的初始化工作，而且将低速外设单位数据传送的程序中断功能也纳入自己职能范围。通道分担了计算机系统中全部或大部分输入/输出功能，提高了 CPU 的效率，同时计算机系统功能分散化的程度也得到提高。

4. 通道分类及其结构组成

(1) 字节多路通道

连接控制多台慢速外设以字节方式交叉传送数据的通道叫字节多路通道 (Byte Multiplexer Channel)。它可以连接多个子通道，例如 8 个、16 个、32 个甚至更多。每个子通道可以接一台或多台同种设备或速度相近设备。不同子通道可以同时工作，但同一子通道内的设备只能串行工作。图 6.29 所示，A, B, C 三个数据块分别来自三个不同的慢速外设，通过字节多路通道和主机进行交换。若在某一时刻开始传送 A 块的一个字节，则 B 和 C 所属设备可以做机电辅助操作。A 块的一个字节传送完毕，则释放通道。若此时 B 的数据已准备好，则可传送 B 块中的一个字节，而 A 和 C 所属设备又可做机电辅助操作。B 块的一个字节传送完毕，又释放通道。若此时 C 准备好数据，则又可传送 C 字节，……，通道就是这样以字节方式交叉地同时为多台外设服务。在一段时间内，连接于通道的外设都在工作，它们分时地占用通道并和主机交换数据，所以通道一直处于忙碌状态。

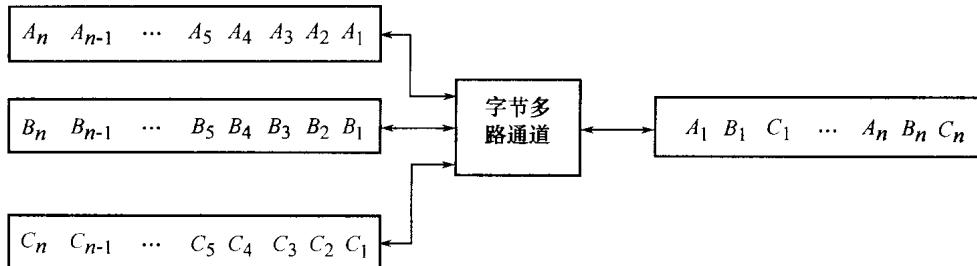


图 6.29 字节交叉传送

字节多路通道结构如图 6.30 所示, 它主要可分为寄存和控制两部分。寄存部分包括数据缓冲寄存器, 主存地址计数器, 字节计数器和状态寄存器。控制部分包括分时操作、数据缓冲和装配分拆以及数据传送控制等逻辑电路。

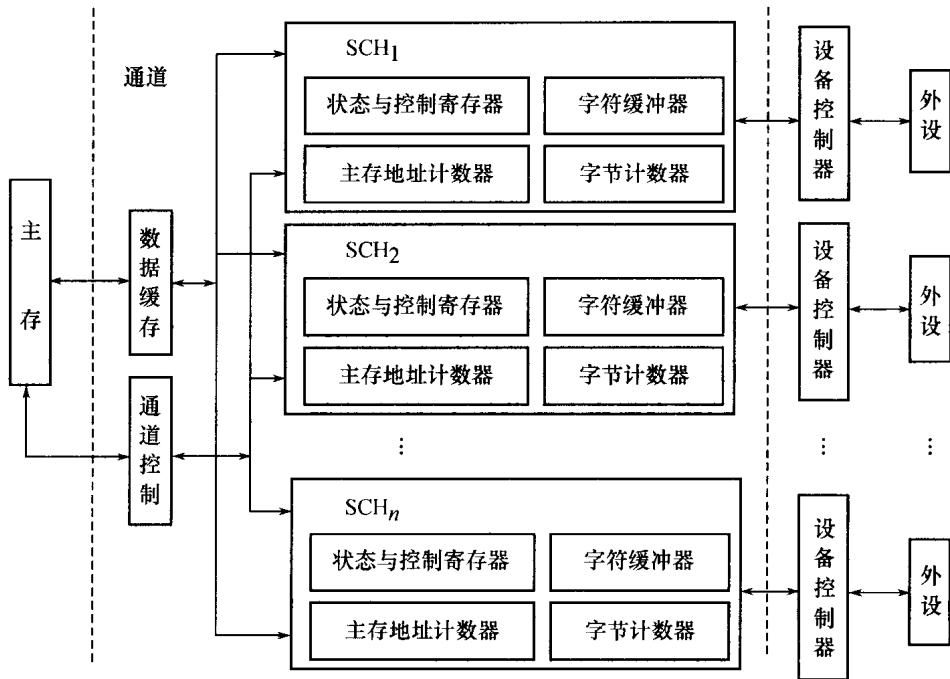


图 6.30 字节多路通道

一个字节多路通道通常包含若干个子通道, 每个子通道服务于一个设备控制器, 即控制一台外设与主存之间的数据交换。如果每个子通道都独立设置一套硬件, 势必造成硬件的过分庞大。因此, 一般均将控制部分由各子通道共享, 而寄存部分则各子通道独立。例如, 各子通道设置自己的字符缓冲寄存器、主存地址计数器、状态与控制寄存器和指明外设地址的寄存器。图 6.30 未示出外设地址寄存器, 通常将它设置到设备控制器中。而字节计数器和主存地址计数器则既可设置于各子通道, 也可由各子通道共享。

(2) 选择通道

许多高速外设, 如磁带、磁盘、高速数据采集系统等, 其数据传输率可高达每秒数百 KB 甚至数十 MB 以上, 它们和主存交换数据是以成组方式进行的, 这些设备数据块相邻字节之间的时间间隔极短, 无法再利用。它们占用通道时, 仅当一批数据交换完毕, 通道才能转而为其他外设服务。显然, 若将这种高速外设连接于字节多路通道, 势必影响连接于该通道的其他外设, 严重时将引起某些设备数据丢失。因此, 一般将高速设备从字节多路通道中独立出来, 而设置选择通道。

连接控制一台或多台同种高速外设以成批方式顺序传送数据的通道叫选择通道 (Select

Channel)。

选择通道每次只能从所连接的外设中选择一台进行数据交换。如图 6.31 所示,当传送 C 数据块的设备占用通道时,其他设备均不能交换。只有等 C 每次传送一个字,直到整个数据块传送完毕并释放该通道,其他设备才可占用通道,并开始交换信息。

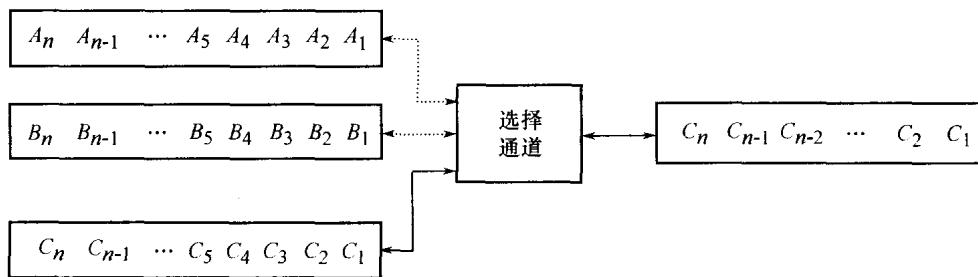


图 6.31 选择通道成批传送

选择通道可认为是只有一个以成组或突发方式工作的子通道。它仅仅设置一套完整的硬件,在一段时间内单独为一台外设服务,在不同的时间段内可以选择不同的外设。

选择通道的结构如图 6.32 所示。它包括:指出读/写数据所需的主存地址计数器,存放外设本次交换所需长度的字计数器,以及设备内部地址寄存器、数据缓冲寄存器和数据格式变换线路等等。

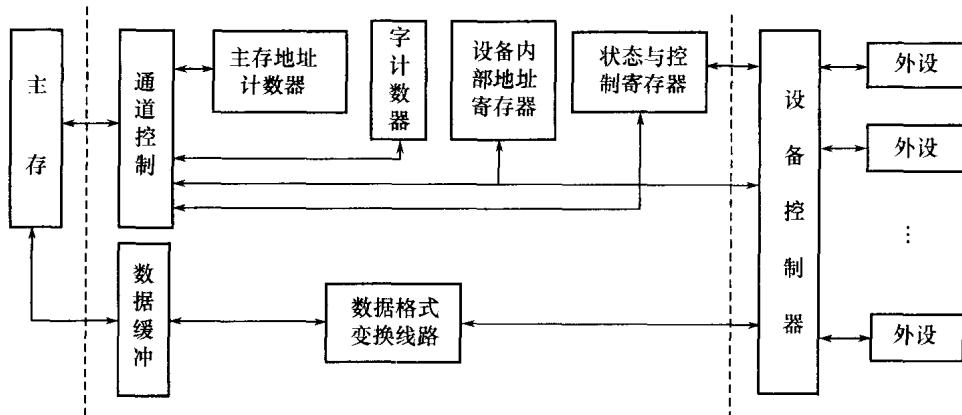


图 6.32 选择通道逻辑结构

(3) 数组多路通道

分别设置字节多路通道和选择通道,既可保证慢速外设以字节交叉方式传送数据,又可保证高速外设以成组方式传送数据,这已似乎是各得其所,完美无缺了。但稍加分析就会发现,类似于磁带、磁盘一类高速外设,其寻址所用的机电辅助操作时间相当可观。用选择通道控制这类设备,则通道启动设备后,首先要进行寻址操作。在这一段时间内,通道并没有

用来传送数据,而是处于等待状态。因此,通道的效能并没有得到充分的发挥。如果能让通道连接多台高速外设,在其中一台外设占用通道进行数据传送时,其他外设则作机电寻址辅助操作。当一台外设数据传送完毕,做完寻址操作的外设又可开始数据传送。按此方式则通道可一直处于忙碌状态,由此提出设计数组多路通道。

连接控制多台高速外设以成组方式交叉传送数据的通道叫数组多路通道(Block Multiplexer Channel)。显而易见,数组多路通道是字节多路通道和选择通道相结合的产物。它使一台设备的数据传送操作和其他设备的机电寻址操作相重叠,实现成组交叉方式传送数据,从而使通道具备了多路并行工作的能力,即连接于通道的多台外设可以并行工作,而通道则充分发挥了高速交换信息的效能。

由于大规模集成电路技术的发展,硬件价格大幅度降低,而高速外设作机电寻址辅助操作的时间日益缩短,有的已在 5 ms 以下,加之数组多路通道控制复杂,很少有计算机采用数组多路通道。

6.6.2 通道命令字和通道程序

通道的最重要特点是有自己的通道命令字,并可独立取出通道命令字并执行之。通过执行一条或多条通道命令字构成的通道程序实现数据传送。那么,什么是通道命令字,什么是通道程序,通道程序如何编写由谁编写以及通道命令字格式怎样,本节将阐述这些问题。

1. 通道命令字

通道命令字,亦称通道控制字(Channel Control/Command Word),缩写为 CCW,又称硬控制字。规定通道及其连接的外设执行何种操作及如何执行该操作的命令字叫通道命令字,它指出通道及其连接的外设执行什么操作和操作所需参数。通道命令字是体现操作员意图或 CPU 命令的一组二进制控制信息,是一种指令。但是,通道命令字是一种功能有限的专用于输入/输出且由通道执行的指令,与由 CPU 执行的属于计算机指令系统的 I/O 指令相区别,故称之为通道命令字。通道执行通道命令字和 CPU 执行指令是可以并行的。

同计算机指令系统一样,计算机不同,通道命令字格式也可能不同,但基本组成是相同或相似的。CCW 的一般格式为

命令码	主存地址	传输长度	特征位	CCW 的首地址	外设内部地址
-----	------	------	-----	----------	--------

- **命令码:**指出外设执行什么操作。读、写、打印、走带、找道等。
- **主存地址:**对输出而言,指出被读出数据的主存始地址;对于输入而言,指出被存数据的主存始地址。
- **传输长度:**指出要传输的单位数据个数。
- **特征位:**表示通道命令字的链接方法或操作特点。如可设置命令链和数据链特征位。利用数据链特征位可将主存不连续区域的数据连续地传送给外设,或将外设不连续区

域的数据传送到主存的一个连续区域。这种操作对外设而言好象执行一条 CCW, 可避免大的数据缓冲区的调度, 有利于页面式的数据交换。利用命令链特征位可将对同一台外设操作的通道命令字链接成一个通道程序。因此, 启动一次外设, 可执行几种不同操作。这样既增强了通道的控制能力, 又减少了对 CPU 的打扰。

外设内部地址: 给出设备内部地址, 该字段是可选的, 因设备的不同而异。如: 磁盘的柱面号、盘面(磁头)号和扇区号; 磁带的组号或区号。

CCW 的首地址: 若是 CCW 串时, 此字段给出存放下一个 CCW 的首地址; 不是 CCW 串时, 该字段信息无意义。该字段也是可选字段, 因不同计算机而异。

我国于 20 世纪 70 年代初生产的 32 位字长的百万次 151 中型计算机, 配置了通道, 其通道命令字格式如下:

首字:	4 位 特征位 (4 位)				24 位			
	命令码	B ₁	B ₂	B ₃	B ₄	D _{CCW} (CCW 的首地址)		
第 2 字:	12 位				20 位			
	WC(传输长度)			D _S (主存地址)				
第 3 字:	18 位		2 位	12 位				
	空	控制		PD _A (外设内部地址)				

151 机通道命令字最大长度为 3 个字, 第 3 个字是可选的, 只有外存磁带或磁盘才使用, 151 机当时未配置磁盘, 但为磁盘的配置留了接口。其中, 命令码、传输长度、主存地址、CCW 的首地址和外设内部地址含义同上述一般格式的说明。下面仅对特征码的含义解释如下:

B₁ = 1: 表示是磁带(盘)机, 即说明本 CCW 由 3 个字组成;

B₂ = 1: 表示是 CCW 串, D_{CCW} 中有下一个 CCW 的首地址;

B₃B₂ = 10: 本 CCW 执行完毕, 则停通道及其外设;

B₄B₂ = 10: 屏蔽结束中断, 即结束时也不发中断。假读即检测时使用;

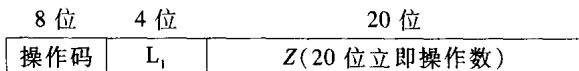
B₄B₂ = 11: 屏蔽缓冲区满中断。

2. 通道程序

通道命令字的有序集合, 构成通道程序 (CHannel Program, CHP)。通道独立执行通道程序, 无须 CPU 干预即可完成数据传送。换言之, 通道程序与 CPU 的现行程序是并行执行的。用户通过自愿进管指令向 CPU 提出通道 I/O 请求, 并提供参数。参数通常包括: 外设名称、内存始地、传输长度、外设内部地址、操作命令等信息。通常称用户提供的这些参数为软控制字。操作系统依用户提供的软控制字, 编制通道程序, 并存到主存指定区域或通道专设的存储器中。

计算机的指令从使用权限可分为管态和目态两类指令, 管态指令又称特权指令。计算机的运行状态也分为管态和目态两种运行状态。计算机处于管态时, 两类指令均可执行; 处于目态时, 只能执行目态指令。因大、中型机的软硬件资源为多用户共享, 不能分给某用户

专用,所以,通道 I/O 指令也是特权指令。用户程序都在目态下运行,用户欲使用外设必须提供传输参数并进入管态。下面以字长为 32 位的 151 机为例简要说明操作系统编制通道程序的过程。该机为用户使用外设提供了一条自愿进管的目态指令,格式为:



在计算机系统中,对于某些经常需要使用的功能,若硬件不能直接提供,或虽有类似功能但不能直接交给用户使用,则由操作系统提供事先编好的所需功能的“程序段”,然后通过进入管理程序的方式调用这个“程序段”,称实现调用操作系统“程序段”的指令为广义指令。一台计算机可有多种广义指令,自愿进管指令是广义指令的一种,用户就是通过它调用一个“程序段”。对用户来说,“自愿进管指令”就像一条功能很强的指令,在使用上与其他指令无本质区别,但实际上并非用硬件直接完成,而是用软件方法实现的。因此,有的教科书或文献把自愿进管指令及其所完成的工作称之为系统功能调用。

自愿进管指令硬件执行的操作是,将 Z 送到 L_1 累加器的低 20 位,并将高 12 位清“0”;同时发出自愿进管中断信号,从而进入管理程序。151 机 CPU 中设置了 16 个累加器,本指令中的 L_1 累加器规定为 0 号累加器。具体进入操作系统的哪种功能的程序段,即自愿进管指令具体要完成什么工作,由 Z 的值决定。当 Z 为 3~6 中的任一整数值时,表示是使用外设类的自愿进管指令,此时用户应提供传输所需参数,以便进管后编写通道程序。因此,在使用 Z 的值为 3、4、5、6 的自愿进管指令之前,用户必须在主存中开辟容量为 4 个 32 位字长的存储空间,并在其中存放一个软控制字,同时将该空间的主存始地址送入 1 号累加器。软控制字结构如图 6.33 所示。

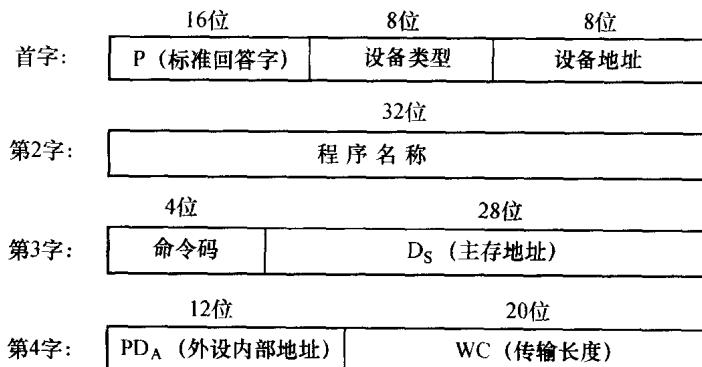


图 6.33 151 机软控制字

软控制字与硬控制字的组成非常相似,现仅就比硬控制字多出的几项作简要说明。设备类型指出本次传输的设备是进行输入/输出,还是其他操作。标准回答字 P 由操作系统填写,用户通过程序可查询,其标准回答为:P 为 1,表示正在传输;P 为 2,传输正常结束;P 为

3,传输有错;P 为 4,外设故障。程序名称为编好的通道程序存放到主存的指定区域提供标识信息。执行自愿进管指令,进入管理程序,管理程序依 Z 的值和 1 号累加器内容取出软控制字,加工处理后编制成一个通道程序,并将它存放到主存的指定区域。

3. 151 机通道 I/O 指令

6.2.5 节曾谈及配有通道的计算机需设置通道 I/O 指令。通道命令字中的命令码和通道 I/O 指令中的操作码都指出执行什么操作,但两者操作是互斥互补的。如通道 I/O 指令的操作通常为:查询通道和外设(即读状态),启停通道(外设),读回控制字等;而通道命令字的操作则为:读、写、打印、走带、找道等。通道 I/O 指令由 CPU 执行,而通道命令字则由通道执行。

在说明 151 机了解通道和外设、启动通道和外设及读回控制字三种通道 I/O 指令前,先介绍通道状态字这一重要名词。反映通道、子通道及其连接的外设状态的一组二进制信息叫通道状态字(Channel State word,CSW)。CSW 存放在通道的状态与控制寄存器中,它指出通道是否正在交换信息,所接外设是否良好,传输正确结束还是传输中发生错误等。使用外设前需取出 CSW 了解通道状态,而当 CSW 指示设备出错时,通道将向 CPU 发中断请求。

151 机的三种通道 I/O 指令均为特权指令。指令中,CM 为操作码,L₁、L₂ 为 CPU 中任意两个不同的累加器。

(1) 了解通道和外设指令格式为:

	CM(8 位)	L ₁ (4 位)	L ₂ (4 位)
L ₁ :	CSW(6 位)	00(2 位)	保持原状态不变(24 位)
L ₂ :	空(16 位)	外设地址(8 位)	通道地址(8 位)

指令执行前,将外设地址、通道地址送 L₂,指令执行后,相应通道的通道状态字被送到 L₁ 的高 6 位,中间两位清“0”,后 24 位保持原状态不变。该指令用来查询通道和外设状态。

(2) 启动通道和外设指令格式为:

	CM(8 位)	L ₁ (4 位)	L ₂ (4 位)
L ₁ :	空(8 位均填为零)	CHP 的首条 CCW 的地址(24 位)	
L ₂ :	空(16 位均填零)	PD 地址(8 位)	CH 地址(8 位)

指令执行前,将外设地址、通道地址送 L₂,将 CHP 的首条 CCW 的地址送 L₁。指令执行后,若该通道相应外设空闲且良好,则被启动工作。

(3) 读回控制字的格式为:

	CM(8 位)	L ₁ (4 位)	L ₂ (4 位)
L ₁ :	已读长度(12 位)	传送最后一个数据的内存地址(20 位)	
L ₂ :	空 (24 位均填零)		CH 地址(8 位)

指令执行前,通道地址送 L_2 低 8 位。指令执行后,将把通道现行 CCW 的第 2 字内容送 L_1 ,以便 CPU 处理。

6.6.3 通道传送的工作过程

1. 通道传送的工作过程

和 DMA 类似,通道传送也分为传送准备、数据传送和结束处理三个阶段。与 DMA 比较前两个阶段差别较大,结束处理阶段完全相同。

(1) 传送准备工作

用户使用通道的外设传送信息应做两项工作。一是通过访存指令提供传送时所需参数,即软控制字;二是通过自愿进管指令以系统调用的方式向 CPU 提出通道 I/O 请求,进入操作系统的设备管理程序。

操作系统的设备管理程序要做三件事,一是根据用户提供的软控制字编制通道程序 CHP,并存到主存指定区域;二是在主存中安排数据缓冲区;三是通过了解外设指令,取出通道状态字 CSW,并判用户所需外设是否良闲?若良闲,则执行启动通道和外设指令,进行数据传送。

(2) 数据传送

① 通道从内存指定位置取出 CHP,执行 CHP,进行数据传送。

② 数据传送正常结束,或传输中出现错误,通道向 CPU 发中断请求,等待 CPU 响应处理。

(3) 结束处理

CPU 响应中断,进入中断服务程序,取出 CSW 并判断,若正常结束,执行正常结束中断服务程序,否则执行错误处理中断服务程序。

2. 程序中断控制传送与通道数据传送的区别

程序中断控制传送靠中止 CPU 现行程序,转去执行中断服务程序实现;通道则是通过执行 CHP 实现。程序中断控制传送的中断服务程序与 CPU 的现行程序是串行工作的;而通道的 CHP 与 CPU 的现行程序是并行工作的。通道是集中独立的硬件,可连接多台快、慢速外设并行工作;程序中断控制传送只适用于慢速外设,且每个外设都有自己独立的接口和中断服务程序。中断的功能强,可处理各种异常或突发事件;而通道只适合于对数据传送的控制。程序中断控制传送以 CPU 为中心;通道则和 DMA 一样以 MEM 为中心。

3. 通道数据传送与 DMA 传送的区别

DMA 主要靠专用接口硬件实现数据传送;通道则靠执行 CHP 实现数据传送。DMA 初始化要由 CPU 执行多条指令完成,通道则自动取 CHP 执行,无需初始化,原因是 CCW 包含了初始化信息,通道从存储器中取出 CCW,存放到通道的寄存器,就相当于对通道传送初始化。DMA 一般用来控制高速外设成组传送,通道既可控制高速外设成组传送、也可控制低

速外设字或字节交叉传送。

小 结

外设是计算机系统的重要组成部分。如何把系统配备的品种繁多、速度各异的多台外设组成 I/O 子系统,与主机连接成一个有机体,并使之具有高的吞吐能力和高的工作效率。是本章研究和探讨的核心问题。自计算机问世以来,为提高计算机的速度和效率,计算机的设计者们构造了多种 I/O 子系统,并且正在努力构造新型的 I/O 子系统。必须指出,目前 I/O 子系统仍然是计算机系统的一个速度瓶颈,影响着计算机系统速度和效率的提高。但也应当看到,随着大规模集成电路(VLSI)技术和计算机技术的发展,这个问题正在逐步得到解决。

本章简述了外设的分类及其与主机连接的方式;阐述了 I/O 控制和 I/O 子系统有关概念以及构造 I/O 子系统的基本原则;重点讲授了中断系统与中断技术以及程序查询方式传送、程序中断控制传送和 DMA 传送的接口组成、工作原理和工作过程;最后介绍了 I/O 通道的分类、结构组成和数据传送过程。

通过本章的学习,一是试图使读者了解和掌握 I/O 控制和 I/O 子系统的基本概念、发展历史和现状以及目前急需解决的问题和途径。二是培养解剖分析几种 I/O 控制和 I/O 子系统的结构组成及工作原理的能力,以便主动应对日新月异的计算机及外部设备的发展变化。三是提请读者关注 VLSI 技术在计算机领域的应用,尤其是高性能多功能外设接口芯片、高集成度外设接口芯片和高集成度外设控制器芯片,如在高档微机主板上的南、北桥芯片以及 INTEL 公司生产的 82380、82C206、82360SL 等芯片。因为使用它们是解决 I/O 子系统这个速度瓶颈的重要途径之一。

习 题

6.1 解释下列术语

接口	标准化 I/O 接口	并行接口	串行接口	中断
中断系统	I/O 子系统	内中断	外中断	单重中断
多重中断	中断源	中断优先权	禁止中断	向量地址
字节多路通道	周期挪用	选择通道	中断屏蔽	通道控制字
数组多路通道	通道状态字	自愿中断	中断隐指令	通道程序

- 6.2 何谓外设? 按功能外设是如何分类的?
- 6.3 试述外设在计算机系统中的地位与作用?
- 6.4 什么是 I/O 控制? 其主要功能是什么?
- 6.5 试说明 I/O 操作和 I/O 设备的特点?
- 6.6 I/O 组织有哪些基本原则? 为什么规定这些原则?

- 6.7 四级和三级I/O子系统各是怎应构成的？各级的功能怎样？各有何特点？
- 6.8 I/O接口有哪些功能？其构成与哪些因素有关？
- 6.9 外设有几种编址方法？各有何特点？
- 6.10 结合程序查询方式接口，说明其工作过程。
- 6.11 中断在计算机系统中有何作用？
- 6.12 何谓中断判优？说明有几种方法？各有何特点？
- 6.13 采用程序中断方式传送的接口应由哪些部分构成？各部分功能是什么？
- 6.14 CPU响应中断的条件是什么？响应中断时应做哪些工作？
- 6.15 整个程序中断过程中，哪些工作由硬件完成？哪些工作由软件完成？哪些工作既可由硬件，也可由软件完成？
- 6.16 说明向量地址与中断服务程序入口地址的区别与联系。
- 6.17 以鼠标为例，结合中断接口线路，说明程序中断传送的工作过程
- 6.18 DMA工作方式有哪几种？各有何特点？
- 6.19 在周期挪用的DMA方式中，每交换一个单位数据，外设实现上也要求中断主机一次。这种中断与程序中断有何不同？
- 6.20 周期挪用DMA接口由哪几部分构成？各部分功能是什么？
- 6.21 DMA接口有几种类型？说明它们的组成及异同点？
- 6.22 以读磁盘为例，结合周期挪用DMA接口线路，说明DMA传送的工作过程
- 6.23 试述DMA传送与程序中断传送的区别。
- 6.24 I/O通道有几种类型？它们的结构组成怎样？
- 6.25 试述通道命令字与通道I/O指令的异同点。
- 6.26 试述通道传送的工作过程。
- 6.27 试述通道传送与DMA传送的区别。
- 6.28 试述通道传送与程序中断传送的区别。
- 6.29 说明选择通道与选择型DMA接口的异同点。
- 6.30 说明字节多路通道与多路型DMA接口的异同点。

第七章 计算机模块结构与互连

引言

冯·诺依曼结构计算机由 5 大部件组成,运算器主要完成数据的加工,控制器主要完成对所有部件的控制,存储器主要完成数据的存储,输入/输出系统主要完成主机和外界环境的数据交换。这些部件或者子部件构成了计算机系统的模块,各个模块在计算机系统中都不是互相独立的,它们共同完成对数据的存储、加工和处理。这些需要加工的数据和控制信息就是通过互连系统进行传输的,也就是说互连系统是模块间协同工作的桥梁。

本章首先介绍计算机的模块以及各种互连结构,然后讨论总线这种常用的互连结构的基本组成、基本工作原理、结构及其设计问题,最后介绍总线标准及实例——PCI 总线。

7.1 模块结构与互连

如果将冯·诺依曼结构计算机的 5 大部件的功能进行合并,计算机也可以看成是由三大部件构成,即中央处理器(CPU)、存储器(M)和输入/输出系统(I/O)。而计算机工作过程中信息的流动,就表现在这三大部件之间的通信。实际上,现代计算机是由相互通信的这三种部件(模块)构成的网络。连接各种模块的通路的集合被称为互连结构(Interconnection Structure),模块间通过互连结构交换信息。从功能上看,互连系统是完成 CPU 和外界信息交换的,故互连结构有时也被看成是输入/输出系统的一部分。

7.1.1 计算机的模块结构

计算机由 CPU、存储器和 I/O 设备构成,基本的计算机模块如图 7.1 所示。各模块都具有与其他模块通信的接口,模块间通过这些接口和互连结构交换数据和控制信息。

图 7.1 通过指定每种模块类型的主要输入/输出形式给出了所需交换的信息:

- 存储器模块:这里的存储模块指的是内存储器,包含 N 个字,每个字 W 位。除了包含一个 $N \times W$ 的存储矩阵,它的接口应该包括宽度为 W 的数据通路、宽度为 $\lceil \log_2 N \rceil$ 的地址通路、读/写控制信号等。

- I/O 设备模块:除了设备本身的功能以外,它的接口主要包括数据通路、地址通路、读

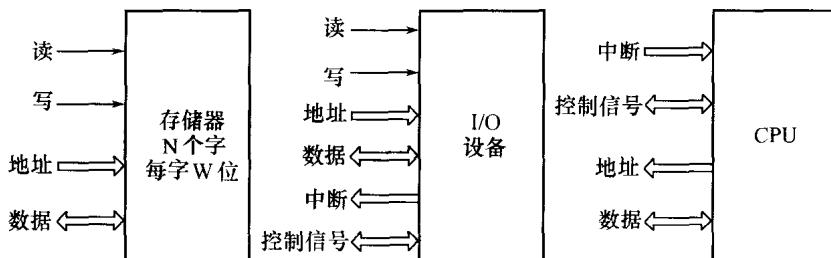


图 7.1 计算机模块及模块接口信息

写控制信号、中断信号、DMA 信号等。I/O 模块可以控制多个外设。

- CPU：中央处理器与外界联系的主要通路包括数据通路（也是指令通路）、地址通路、中断信号、DMA 信号以及其他一些控制信号。

7.1.2 常见互连结构

计算机系统，就是将 7.1.1 节中介绍的模块连接起来进行处理，各模块间通过互连结构交换信息。通过模块连接成的计算机系统如图 7.2 所示。

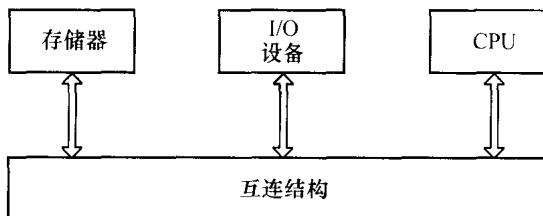


图 7.2 通过模块连接的计算机系统

自计算机问世以来，人们在互连结构上进行了大量的研究，产生了各种各样的互连模式。如果将计算机各个部件或者模块称为结点，则结点的位置及其互连的几何布局也被称为拓扑结构（Topology Structure）。一些常见的拓扑结构如图 7.3 所示。

这些常见的拓扑结构包括：

- 星型（Star）结构：该结构中，通常将 CPU 作为“中央结点”，由它控制其他结点。所有结点间的通信必须经过中央结点。
- 树型（Tree）结构：该结构中的结点有着明显的层次关系，不同层次的结点承担不同的功能，越高层功能越强，通常将 CPU 作为最高层结点——根结点。
- 交叉开关（Crossbar）结构：该结构又被称为全连通结构。所有结点间均具有直接的信息通路。
- 总线（Bus）结构：所有结点被连接在一条公共的信息通路上，任何一个结点发出的信息均可以被总线上的其他结点接收。

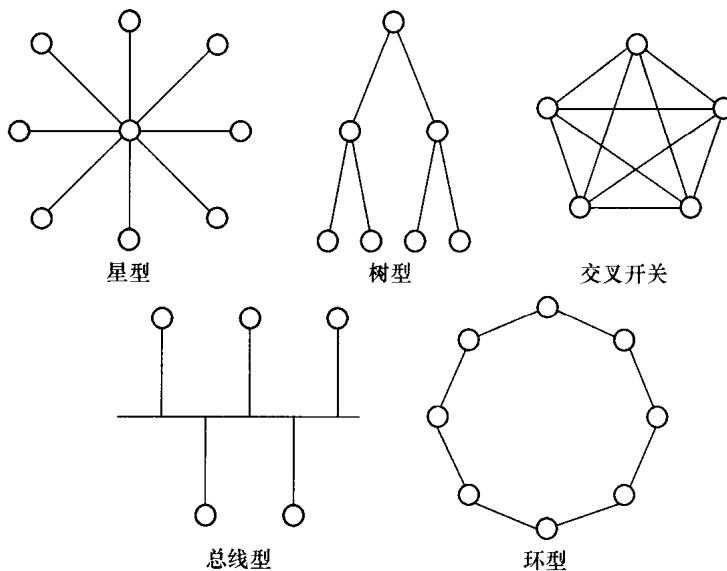


图 7.3 常见的拓扑结构

- 环形(Ring)结构:该种结构与总线结构基本相同,区别在于其公共信息通路被连接成为环形,信息在环形线路内单向或双向传输。

在这些常见的结构中,总线结构是应用最广泛的一种拓扑结构。使用总线连接而成的计算机系统结构简单,而且价格低廉。本章后续各节研究的主要内容就是总线。

7.2 总线系统概述

总线是构成计算机系统的互连结构之一,是多个系统功能部件之间进行信息传送的公共通路。1970年,DEC公司在其PDP11/20小型计算机上首次采用了Unibus总线。自从1975年Altair诞生以来,微型计算机都采用了总线这种结构。Altair使用的总线就是后来的S100总线。

对于CPU来说,总线是微处理器与外部硬件接口的核心。自IBMPC问世以来,随着微处理器技术的飞速发展,总线技术也得到不断创新。由历史上出现过的PC/XT、ISA、MCA、EISA、VESA总线,再到目前流行的PCI、AGP、IEEE1394、USB总线等。总线技术的飞速发展源于系统各功能部件间对数据传输的需求。先进的总线技术对于解决系统瓶颈,提高整个计算机系统的性能有着十分重要的影响。

总线系统由总线、总线设备、总线设备接口和总线控制器四部分构成。总线是由传输线、三态门和输入/输出缓冲构成的一组共享信息传输线。总线设备通过总线设备接口可以连接到总线上。由于传输线共享,总线的管理和使用必须通过总线控制器进行。下面分别

介绍总线的基本组成,总线系统的基本工作原理,总线的分类以及总线的性能。

7.2.1 总线的基本组成

总线的基本组成元件是集电极开路门(OC门)或者三态门。三态门的逻辑符号及真值表如图7.4所示。

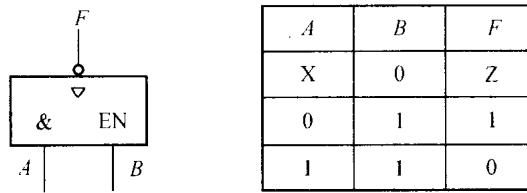


图7.4 三态门的逻辑符号及真值表

三态门的输入端为A、B为选择端(亦称为控制端),输出端为F。当B为低电平时,无论A为何种信息,F都处于高阻(开路)状态,此时可以认为A与F开路。当B为高电平时,F的信号为A的信号取反,此时可以认为A与F接通。总线的工作正是利用三态门来控制哪些连接在总线上的设备可以使用总线(接通状态),禁止哪些设备使用总线(开路状态)。三态门可以实现“线或”,从而可以使多个设备通过三态门与信息传输线构成总线系统。例如,由四路信号A、B、C和D通过三态门实现的1位单向总线如图7.5所示。

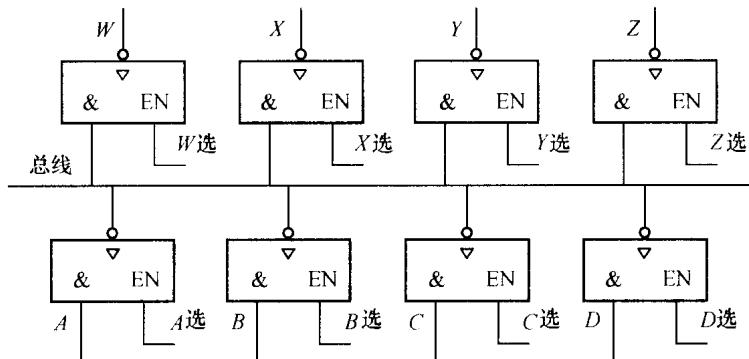


图7.5 由三态门组成的一位总线

四路信号A、B、C和D经过三态门送到总线,4个三态门的输出线或在一起。总线通过三态门的输出信号为W、X、Y和Z。某一时刻,哪路信号送入总线,总线信号从哪路输出,由相应的门控制。由于三态门并没有保存数据的功能,为保证数据的正确传输,通常需要在三态门的输入端增加数据缓冲器或者总线锁存器以保持数据。一种8位的双向总线的基本组成如图7.6所示。

图7.6中总线由8位组成,假定ID₇~ID₀端与CPU连接,而DB₇~DB₀端与外部设备连

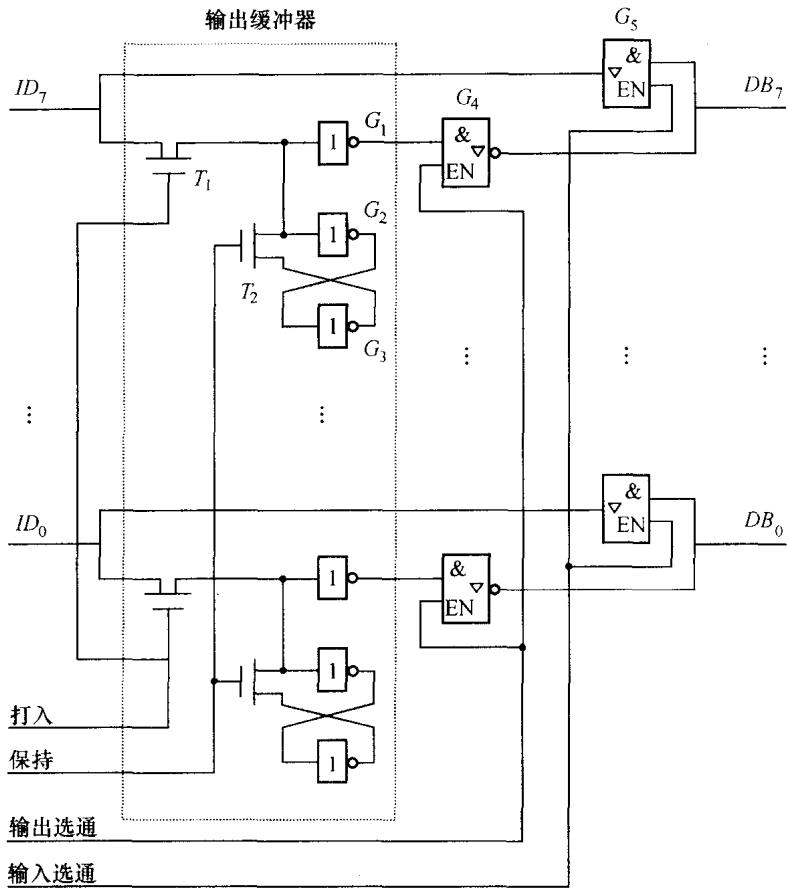


图 7.6 双向总线的组成

接。输入时,由“输入选通”信号控制三态门 G_5 ,使 $DB_7 \sim DB_0$ 端的信号直接送到 $ID_7 \sim ID_0$ 端,然后就可以送到 CPU 指定的寄存器中。输出时,在“打入”和“保持”信号控制下, T_1 和 T_2 接通, $ID_7 \sim ID_0$ 端的数据先送入由 G_2 和 G_3 组成的数据锁存器暂存,然后由“输出选通”信号控制三态门将锁存的数据经门 G_1 和 G_4 输出到 $DB_7 \sim DB_0$ 端。需要说明的是,由于 CPU 的速度比外部设备的快,所以三态门 G_5 的输入端不必设数据缓冲器。

7.2.2 总线系统的基本工作原理

总线系统的工作简单地说就是在总线控制器的作用下,通过总线设备接口控制、管理连接在总线上的设备使用总线。一个简单的总线系统如图 7.7 所示。

总线的主要特征是信息传输线是共享的。总线设备为了使用总线，必须首先获得总线的使用权。总线设备使用完总线后，必须释放使用权给其他设备使用。这些操作是通过设备和总线控制器间的请求与应答信号完成的。设备使用总线的时序如图 7.8 所示。

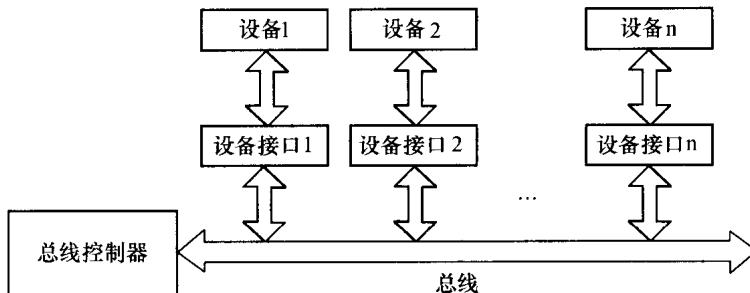


图 7.7 一个简单的总线系统的逻辑结构组成

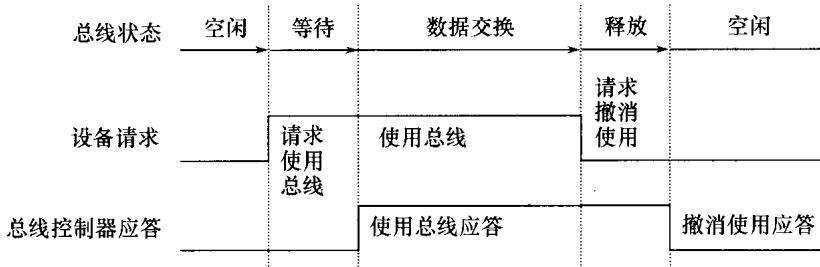


图 7.8 总线设备使用总线的时序图

图 7.8 中，设备使用总线的过程描述如下：

- 首先，设备发出总线使用请求，并等待获得总线使用权；
- 总线控制器根据使用总线的规则，对该请求给出应答，表明该设备可以使用总线；
- 设备在得到应答以后，就开始使用总线进行数据交换；
- 数据交换完成以后，设备将发出撤销使用总线的请求，表示设备本次使用总线完毕；
- 总线控制器在接收到撤销使用总线的请求信号后，收回总线使用权，使总线处于释放状态，然后发出总线撤销使用应答信号；
- 总线撤销使用应答信号发出以后，总线进入空闲状态，可以接受新的请求。

7.2.3 总线的分类

总线是连接计算机部件内部单元、计算机各部件之间或者各计算机之间的一束传输信息的公共通路。由于计算机部件内部单元、计算机部件以及计算机的多样性，决定了总线的多样性。

从规模上看，计算机的总线可以分为：片内总线、系统总线、设备总线和通信总线。片内总线指的是在芯片内部设置的总线，其目的是完成芯片内部功能单元的信息交换。例如第三章中介绍的 ALU 内部的总线就属于片内总线。系统总线又称为板级总线，用于计算机内部各大功能部件间的信息传输，因这些部件都制作在各自的插件板上，故此得名。设备总线

又称为 I/O 总线,用于连接各种外部设备,例如 PCI 总线。通信总线又称为外部总线,用于完成计算机间以及计算机与其他系统间的信息通信,例如 RS-232 总线。

对于系统总线和设备总线,按照总线上传输的信息种类,又可以将总线分为地址线(Address Bus, AB)、控制线(Control Bus, CB)和数据线(Data Bus, DB)三种。这就是人们常说的三大总线。了解这三种总线的组成、用途及相互作用关系,对于理解计算机系统的应用及接口问题十分重要。

地址线又称为地址总线,它是单向总线,用来标明发送或接收数据的设备编号和需要访问的设备内部单元的地址。

控制线又称为控制总线,用于控制总线设备对总线的使用。控制信号主要包括完成设备之间进行信息交换时的定时信号和命令信号两类。定时信号标明有效的地址和数据出现在总线上的时间;命令信号定义总线上所要完成的操作。常用的命令信号包括总线请求/应答、读/写等信号。

数据线又称为数据总线,是总线设备进行数据信息交换的通路。

总线还可以按其他的标准进行分类。例如,按照总线一次传输的数据位数分为 1 位、8 位、16 位、以及 32 位总线等;按照总线信号是否有多个功能分为专用和复用总线;按照总线的定时方式分为同步和异步总线等。这些分类方法涉及到总线的设计问题,关于这些设计问题将在 7.4 节中讨论。

7.2.4 总线的特点和性能

从总线的基本组成和基本工作过程中可以知道,无论是哪种总线均具有如下特点:

- 多个部件之间采用总线互连方式,可以大大降低部件之间互连的复杂性,大幅减少连线数量。
- 使用总线互连后,各部件之间连接的多个连接控制接口变成了每个部件与总线之间的单一连接控制接口,接口的器材量大幅减少。
- 如果设备之间没有或者很少有多个部件同时申请使用总线,采用总线方式连接这些部件可以有效地发挥总线连接的优点。但是,如果存在大量同时工作的部件,采用分时共享信息通路的总线方式,会严重降低部件之间进行信息交换的效率。

作为计算机各功能部件间通信的桥梁,总线的性能对于整个计算机系统的性能有着十分重要的影响。总线的性能可以通过总线带宽进行衡量。总线带宽,又称为总线数据传输率,指的是单位时间内总线上传送的数据量,通常以 MB/s 为单位。一般来说,带宽越大的总线性能越高。

与总线带宽密切相关的两个参数是总线宽度和总线工作时钟频率。总线宽度指的是数据总线一次能传送的数据位数,总线宽度越宽则总线每次传输的数据量越大。总线工作时钟频率以 MHz 为单位,工作频率越高则总线工作速度越快。单方面提高总线宽度或工作时

钟频率都只能部分提高总线的带宽，并容易达到各自的极限。只有两者配合才能使总线的带宽得到更大的提升。

除了总线宽度和工作频率这两个最直接的参数以外，影响总线带宽的因素还有很多。例如，采用成组数据传送方式将获得比基本数据传送方式更大的带宽，减少总线仲裁的时间将有助于提高总线带宽等。这些总线的设计问题将在 7.4 节中介绍。

7.3 总线系统的结构

总线系统的基础是共用的信息传输线。除此而外，总线系统还包括总线设备接口、总线设备以及总线控制器。图 7.7 就是一个简单的总线系统。为了将不同速度的总线设备连接在一起，通常需要采用多级总线的结构进行连接。下面分别介绍总线设备、总线设备接口、总线控制器以及多级总线结构。

7.3.1 总线设备

虽然使用的是同一组信息传输线，但总线上的设备是多种多样的。这些设备通过总线设备接口与总线连接。从申请总线使用权的角度，可以把总线设备分为主设备和从设备。从数据传输的方向，可以把总线设备分为源设备和目标设备。从访问总线设备的方法，可以将设备分为存储器设备和 I/O 设备。

1. 总线主设备和总线从设备

从总线的工作过程可以看出，要完成一次总线操作，就必须有设备提出使用总线的申请，以获得总线的使用权。在连接到总线上的所有设备中，能够申请并获得总线使用权的设备，称为总线主设备（Master Device）。不具有申请总线使用权的设备，称为总线从设备（Slave Device）。总线主设备可以引发一次总线操作，一般具有较完备的总线控制功能；而总线从设备则不能够引发总线操作，它只能在总线操作中做为被操作的对象。

由于总线是被所有总线设备共享的，所以在任何时候，一条总线上工作的总线主设备不允许超过一个，否则将导致总线使用权的混乱，从而引发总线上信息的混乱。这是总线系统工作的基本原则之一。如果同时有多个总线主设备申请使用总线，则只有一个主设备的请求被允许。决定由哪个总线主设备使用总线涉及总线仲裁电路的功能。

总线主设备和总线从设备的概念是按总线设备的逻辑功能划分的。连接到总线上的物理设备，可以是总线主设备，例如 CPU；也可以是总线从设备，例如存储器模块；也可以既是主设备又是从设备，例如智能化磁盘控制器。

2. 总线源设备和总线目标设备

从总线的工作过程可以看出，在使用总线的设备中有些设备能够发送数据，有些设备能够接收数据。将发送数据的设备称为总线源设备（Source Device），而将接收数据的设备称

为总线目标设备 (Target Device)。需要注意的是,总线源设备/目标设备与总线主/从设备间没有必然的联系。总线主设备可以是总线源设备,可以是总线目标设备,还可以既不是总线源设备也不是总线目标设备。

3. 存储器设备和 I/O 设备

存储器设备和 I/O 设备是以访问总线设备的方法来区分的。使用访问存储器的方法访问的设备称为存储器设备 (Memory Device), 这种设备需要使用访存型总线命令来访问; 使用访问 I/O 的方法访问的设备称为 I/O 设备 (I/O Device), 这种设备需要使用 I/O 型总线命令来访问。例如, 主存是存储器设备, 而磁盘是 I/O 设备。

下面以主存数据使用 DMA 方式写入磁盘的过程为例, 来说明不同总线设备的类型:

- 首先, DMA 控制器根据 CPU 的命令, 申请使用总线。因此, 在总线操作过程中, 总线主设备是 DMA 控制器, 而主存和磁盘都是总线从设备。
- 然后, DMA 控制器控制数据从主存写入磁盘中。因此, 在总线操作过程中, 主存为总线源设备, 磁盘为总线目标设备。
- 从总线上访问主存, 必须以“存储器读”总线命令来完成; 而将数据写入磁盘, 是使用“I/O 写”总线命令来完成。因此, 在总线操作过程中, 主存为存储器设备, 磁盘为 I/O 设备。

7.3.2 总线设备接口

由于总线设备的多样性, 总线设备和总线的连接必须通过总线设备接口进行。这个接口能够接收并且识别总线控制信号, 并根据设备的特性产生控制设备的信号, 如图 7.9 所示。

总线设备接口是连接设备与总线的桥梁, 其功能是完成设备信号和总线信号之间的协调和转换。因此, 总线设备接口应配置支持总线和设备工作的逻辑。

1. 总线设备接口对总线配置的逻辑主要完成以下工作:

- 对于总线主设备, 应产生总线使用请求信号 (Bus Request), 接收总线使用应答信号 (Bus Grant/Acknowledge), 并在获得应答后, 按照总线协议使用总线。
- 对于总线从设备, 则应接收设备地址译码器, 以确定该从设备是否能够使用总线。若译码后命中该总线从设备, 则按照总线命令的指示进行总线操作, 发送或者接收数据。
- 如果设备是以中断方式与总线上的其他设备进行数据交换, 则在设备数据就绪后, 按照总线协议产生相应的中断请求信号 INTR、接收中断响应信号 INTA 以及中断类型码, 等候 CPU 进行相应的处理。
- 如果设备是以 DMA 方式与总线上的其他设备进行数据交换, 则按照总线协议产生相应的 DMA 请求信号 HRQ、接收 DMA 应答信号 HLDA, 在获得 HLDA 信号后, 根据 DMA 控

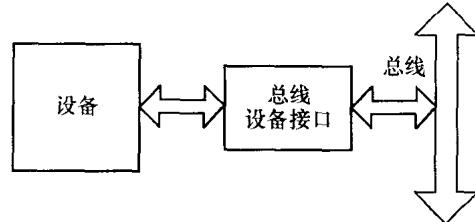


图 7.9 总线、总线设备和总线设备接口

制器的控制信号,完成总线源设备与总线目标设备之间的数据交换。数据交换完毕以后,总线设备接口必须撤销 DMA 请求信号。

2. 总线设备接口对设备应具有下述功能信号的处理能力:

- 总线使用请求 (Bus Request): 标明本设备申请总线使用权;
- 总线使用应答 (Bus Grant/Acknowledge): 标明本设备获得总线使用权;
- 存储器操作 (Memory Operation): 表示根据地址线上给出的存储器单元进行操作;
- I/O 操作 (I/O Operation): 表示对地址线上确定的 I/O 单元进行操作;
- 读操作 (Read Operation): 标明总线上正在进行的是读操作;
- 写操作 (Write Operation): 标明总线上正在进行的是写操作;
- 中断信号 (Interrupt Signals): 表示对总线上的设备进行中断处理;
- DMA 信号 (DMA Signals): 表示对总线上的设备进行 DMA 操作。

总线设备接口是否具有以上这些信号,与设备本身的特性相关。设备本身产生这些信号的时机与总线要求的时间不一定对应,信号规范也不一定一致。总线设备接口的功能就是完成设备与总线之间信号的协调,故一种设备只需要通过修改总线设备接口就可以连接到不同的总线上。

需要注意的是,在本书第六章中谈及的 I/O 系统接口,其功能已经包括了总线设备接口的功能。

7.3.3 总线控制器

总线控制器是总线系统的核心,如图 7.7 所示。总线控制器的任务概括地说就是管理总线的使用,包括总线上设备的管理和设备使用总线的过程管理。需要说明的是,在实现总线控制器时,并不一定需要一个专用的总线控制器,它的功能可能分布到总线的各个部件或者各个设备上。这一点,在 7.4.4 节中关于总线的设计和使用时还会详细讨论。

总线控制器的作用是实现总线协议,它有以下主要功能:

- 总线系统资源的管理。总线系统的资源主要有存储空间、设备端口 (I/O) 空间、通道、中断等,总线控制器具有进行资源的分配、解决资源冲突、设备的选择、启动、复位等功能。
- 总线系统的定时。无论是同步总线还是异步总线,系统都必须有定时控制。它的主要工作是产生各种总线命令和标识信号,协调设备的工作过程,辅助总线仲裁电路工作,控制设备占用总线的时间,产生各种定时信号等。
- 总线的仲裁。当总线中有若干个总线主设备同时产生使用总线的请求时,需要对这些请求进行仲裁,确定哪一个主设备可以获得总线使用权。
- 总线的连接。对于系统中存在多种总线,需要由总线控制器完成不同总线协议之间的转换;对于系统中有多条总线,总线控制器要完成这些总线之间的连接。

7.3.4 总线的连接方式

总线是设备共享的信息传输线。如果有许多设备连接到总线上，总线的性能就会下降。其主要原因是：总线上连接的设备越多，同时申请使用总线的总线主设备就可能越多。而总线系统工作的基本原则要求获得总线使用权的总线主设备只能有一个，这样其他的主设备就会因为不能获得总线使用权而必须等待。另外，多个总线主设备竞争使用总线，总线仲裁电路的仲裁时间也可能会增加。

随着总线上传输请求的增加，总线便会成为瓶颈。通过提高总线的数据传输率或者使用更宽的总线，这个问题会在某种程度上可以得到解决。但是事实证明，总线上挂接设备数量的增长速度比总线数据传输率或者总线宽度的提高速度要快得多，所以单纯依靠提高总线的数据传输率或者使用更宽的总线是不能解决总线的瓶颈问题的。

如何合理地解决这个问题，就涉及到总线的连接方式。根据连接方式的不同，总线结构被分为单总线结构、双总线结构和多总线结构。单总线结构最简单，但是容易产生数据传输的瓶颈问题。因此，多数的计算机系统都选择使用双总线或者多总线结构。

1. 单总线结构

单总线结构是最简单的总线系统结构，它由一条总线构成。使用单总线的计算机系统中，CPU、存储器设备、输入/输出设备等都以总线设备的形式连接到总线上。这种简单的总线系统往往没有独立的总线控制器，总线控制器的功能由 CPU 担任。在比较复杂的总线系统中，考虑到系统效率、系统的设备扩展能力和标准化等因素，需要使用独立的总线控制器。图 7.10 给出了这种简单计算机系统的典型结构。

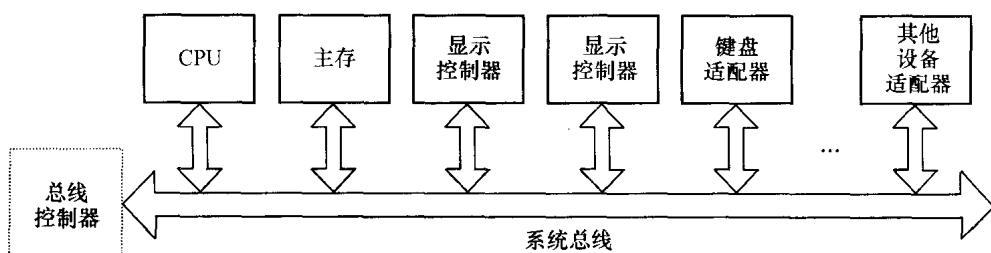


图 7.10 单总线计算机系统

显然，在单总线计算机系统中，设备可以直接通过总线设备接口连接到总线上。但是单总线结构也存在系统工作效率不高的问题。这个问题可以归结为以下两个方面的原因：

(1) 多设备竞争总线使用权。例如，目前的计算机系统中，都设有显示系统。如图 7.10 所示。为了保证显示器正常工作，显示控制器需要不断地从显示存储器中取出数据送往显示器，这个过程就是显示器的刷新过程。如果显示控制器和 CPU 竞争使用总线，这无疑对于 CPU 和显示刷新的效率都会产生影响，尤其对于高速的 CPU 来说，将产生严重的影响。

(2) 多种速度相差很大的设备连接在一条总线上。计算机可连接设备的速度千差万别,有速度很快的显示控制器,也有速度较慢的键盘等设备。键盘使用总线的速度一般在每分钟两百次以内。慢速设备使用一次总线,就会对快速设备使用总线产生影响,从而降低了整个系统的效率。

单总线结构的另外一个问题是计算机的扩展能力受到很大的限制。由于总线的驱动能力有限导致在单总线上连接的设备不能很多。较多的主设备使得总线控制器的仲裁时间增加,而这种开销对计算机的整体性能是不利的。

单总线系统存在问题的产生原因,是由于所有设备都挂接在一条总线上,该总线只能分时被多个设备共享。

对系统效率不高这个问题的解决方法之一是:在单总线结构中,要求连接到总线上的逻辑部件必须能够高速运行,以便在某些设备需要使用总线时,能够迅速获得总线使用权;而当设备不再使用总线时,能迅速释放总线使用权。否则,由于一条总线由多种设备共用,可能导致访问请求的冲突,导致较长的等待延迟,从而降低了效率。

解决单总线存在问题的另一个方法是使用双总线结构。

2. 双总线结构

双总线结构中存在着两条可使用的总线,图 7.11 给出了一种双总线结构的计算机系统。

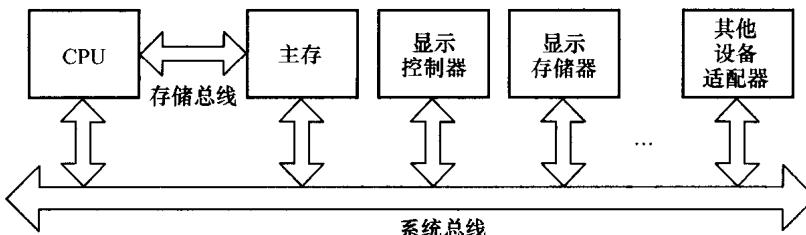


图 7.11 一种双总线结构计算机系统

图 7.11 所示的双总线结构保持了单总线结构简单的特点,只是在 CPU 和主存之间增加了一组高速的存储总线。由于 CPU 和主存独占使用存储总线,故在存储总线上没有请求冲突,数据传输率很高。存储总线同时也减轻了系统总线的负担,CPU 访问主存和显示控制器访问显示存储器可以并行进行。同时,主存仍可以通过系统总线与其他外设之间实现 DMA 操作。双总线结构以增加一组总线为代价,在一定程度上缓解了总线上的冲突,但并没有解决系统总线上慢速设备对快速设备效率的影响等问题。

3. 多总线结构

为解决多设备竞争总线使用权而带来的问题,在设备间可以设置更多的总线。多总线结构可以有效地改善设备对总线使用的竞争,设备既可以同时使用不同的总线,又可以通过共享的总线进行信息交换。一种三总线结构的计算机系统如图 7.12 所示。

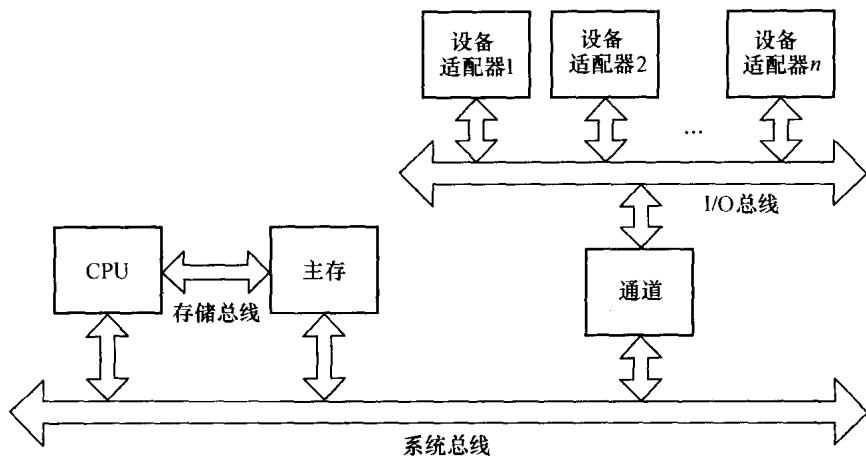


图 7.12 一种三总线结构计算机系统

该三总线系统是在图 7.11 所示的双总线的基础上增加 I/O 总线形成的。其中，系统总线是 CPU、主存和通道间共享的信息通路，而 I/O 总线是多个设备间信息传输的共用通路。通过第六章的学习可知，采用 DMA 输入/输出方式时，外设和存储器间可以直接进行数据交换，从而减少了输入/输出设备对 CPU 的打扰。采用通道方式时，由于通道分担了部分 CPU 的功能，使整个系统的效率大为提高。多总线的结构在计算机系统设计中广泛存在。

多总线结构的另外一个优点是可以增加系统外接设备的数量。在实际系统中，每条总线的驱动能力是有限的，通过设置多条总线，可以有限地增加计算机系统可以连接设备的数量。

通常，计算机上总是存在以各种速度工作的设备。如果使用总线连接，只有总线速度与设备速度匹配才能够获得好的系统效率。否则，慢速设备使用总线将对快速设备使用总线产生很大影响。为解决速度差异较大的多种设备连接在同一条总线上而带来的速度不匹配问题，可以将不同速度的设备连接在不同速度的总线上，低速总线作为高速总线的一个设备工作，总线间通过桥接器(Bridge)进行连接，这样构成的多总线结构称为多级总线结构。一种具有多级总线结构的计算机系统如图 7.13 所示。

显然，多级总线是多总线结构的一种特例，每一级总线对应管理一定速度范围内的设备。和多总线结构一样，多级总线可以解决单总线面临的系统效率较低和驱动能力不足等问题。另外，由于多级总线可以分别连接速度不同的设备，减少了不必要的高速连接端口，有利于降低系统造价。目前大多数计算机都采用多级总线的结构，以获得良好的性能价格比。

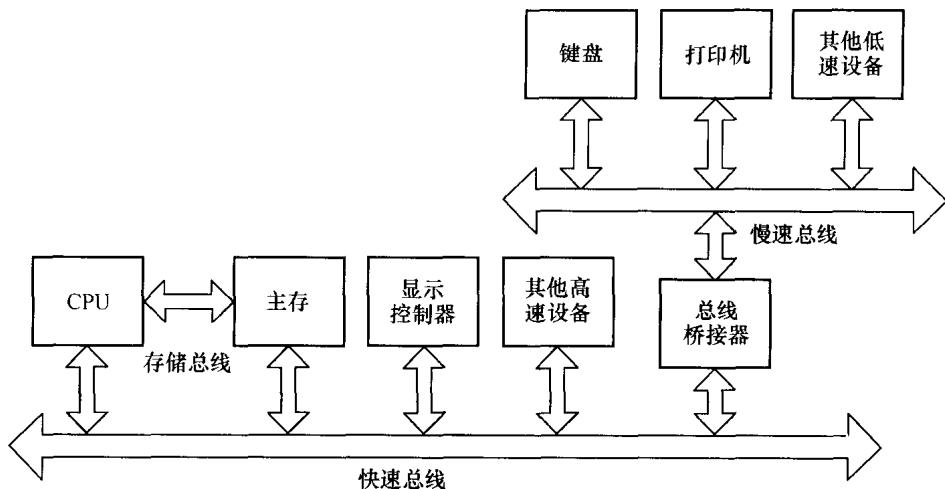


图 7.13 一种多级总线结构计算机系统

7.4 总线设计要素

总线设计就是设计一个总线结构,以便计算机系统中各部件之间进行互连使用。总线的结构不仅包括总线的信息传输线,还包括总线设备、总线设备接口以及总线的连接方式。因此,总线结构的设计可能有许多不同的实现方法。虽然总线的实现方法各不相同,但基本参数和设计要素却是相同或相似的,它们是:

- 总线的宽度;
- 总线的复用方式;
- 总线的定时方式;
- 总线的仲裁机制;
- 总线的数据传送方式。

这些参数和要素将对总线的性能产生重要影响,它们同时也是常用的总线分类方法。下面就分别从这几个要素讨论总线的设计问题。

7.4.1 总线宽度

7.2.4 节讲过,总线宽度指的是数据总线连接线的数量,通常以位(bit)为单位。每根总线一次能够传送一位数据。总线连接线根数越多,则总线宽度越宽。总线宽度是总线设计中一个重要的参数,对总线的性能和成本有重要影响。较宽的总线可以获得较高的性能,同时因为总线设备接口和连接的复杂性将导致系统成本的增加。

总线的宽度决定了计算机一次能够传输的数据位。总线越宽,一次传输的数据量就越

大。计算机数据总线的位数通常与计算机的字长相同。随着微处理器字长的不断提高,CPU与外界交换数据量的增加,导致了数据总线的宽度不断提高。常见的数据总线有1位、8位、16位、32位和64位。如串行总线为1位总线,ISA总线有16根数据线,PCI总线有32或者64根数据线,等等。目前某些高性能总线的数据信号线数量达到128根甚至更多。

另外,地址总线的信号线越多,计算机能够直接寻址的空间就越大。若地址总线为 n 位,则计算机可以直接寻址的地址空间为 2^n 个存储单元。地址总线还可以用来寻址外部设备,这也意味着,地址总线的位数决定了总线上可连接设备的数量。常见的地址总线有8位、16位和32位。虽然有些总线定义了64位地址线,但目前很少能使用到64位地址线。

为了传输更多的数据,或者达到更大的直接寻址空间,通常需要设置更多的数据或地址线。但这需要增加总线连接线的数量,这样就需要占用更大的物理空间(例如主板的面积),总线连接设备(例如总线缓冲区)也需要进行相应的调整。这些因素都使得总线的连接和控制变得复杂,导致总线价格变得更加昂贵。所以在总线设计时,需要在总线性能和总线系统价格间进行合理的折衷。如果性能需求使用超宽的总线时,则可以考虑采用复用总线的方式来降低系统代价。

7.4.2 总线的复用方式

为了节省总线的连接线,可以让一些不同时工作的信号分时使用同一根总线连接线,这种将一条信号线用于多种目的的方法被称为复用。根据信号是否复用,可以将总线分为专用型(Dedicated)和复用型(Multiplexed)两类。如果总线中的某一根连接线,仅仅定义了一种功能或者仅仅连接到一个总线设备,则称这一根连接线为专用型的。相反,如果在总线的一条连接线上定义了多种意义的信号或者连接多个总线设备,则称这根连接线为复用型的。

对于总线系统影响最大的是数据线和地址线的复用,因为这两组信号在总线系统中的连接线数量最多。有的总线采用地址线和数据线分离的专用型策略,例如ISA总线、EISA总线等,有的总线采用地址信号和数据信号复用的策略,例如PCI总线、NuBUS总线等。

专用型策略下每个信号都有自己的连接线,只要信号有效就可以使用。专用总线被始终赋予一种功能,总线的设计和使用都相对简单。专用总线的优点是总线冲突较少,所以具有较高的吞吐量。其缺点是增加了系统的尺寸和成本。

专用型策略下,系统为数据总线和地址总线单独设置连接线,但有时这是不必要的。如对存储器的读出需要先给出地址,经过一段时间后才会出现数据,这就使得数据和地址信号可以复用相同的总线。需要注意的是,复用型策略下,为了区分是哪一组功能的信号在使用连接线,需要增加一根控制信号线。例如,使用地址有效(Address Valid)信号或者地址锁存(Address Latch)信号,就可以控制数据信息和地址信息通过同一组信号线传输。在数据传输开始时,地址信息首先出现在总线上,地址有效信号有效。此时,总线设备在规定的时间内检查自己是否是被寻址的设备,然后复制地址信息。接着,地址有效信号被撤消,同一组

信号线被用于随后读写数据信息的传输。这个过程如图 7.14 所示。

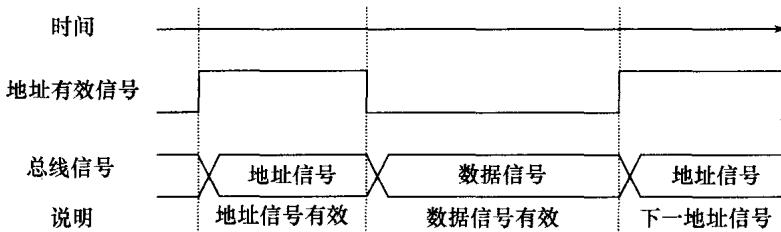


图 7.14 复用型总线上的地址和数据信号

典型的地址数据复用总线如 8086 和 8088 的系统总线,它的 AD₀ 到 AD₁₅ 信号线则是数据和地址复用的。复用的信号由地址锁存使能(Address Latch Enable, ALE)信号控制。当 ALE 有效时,表明从 AD₀ 到 AD₁₅ 上存在的是地址。除了地址和数据信号可以复用以外,常见的复用信号还有读与写控制信号复用,存储器访问与 I/O 访问控制信号复用,等等。

分时复用的优点是使用了较少的信号传输线,从而节省了空间和成本。其缺点主要是每个设备需要较复杂的控制电路。另外,潜在的问题是可能降低系统的性能,因为复用的信息(例如数据和地址信息)不能同时出现在总线上。

7.4.3 总线定时方式

总线定时方式实质是总线上的事件之间协同工作的方法,它包括同步定时(Synchronous Timing)和异步定时(Asynchronous Timing)两种方式。采用同步定时方式工作的总线为同步总线(Synchronous Bus),采用异步定时步方式工作的总线为异步总线(Asynchronous Bus)。

1. 同步总线

对于同步总线,总线上事件的发生由时钟决定。时钟是一个按照周期出现的 1~0 信号序列。时钟的每一次 1~0 信号转换被称为一个总线时钟周期(Bus Clock Cycle)或时间槽(Time Slot),这是同步总线上操作的基本时间单位。时钟周期的倒数为时钟频率。通常,总线的时钟频率比 CPU 的时钟频率要慢得多,例如 PCI 总线的时钟频率只有 33 MHz 或 66 MHz。造成当前总线频率较低的原因主要有两个方面:一方面是总线自身的原因,例如总线信号的传输延迟容易造成总线信号的偏离;另一方面就是总线设计中需要做到向后兼容,即总线应该能够支持较早出现的慢速总线设备。

同步工作方式指的是系统有一个供所有设备使用的统一时钟,设备之间按照约定的时钟时间进行信息交换。总线上所有的设备都能读取时钟信号,按照总线的要求,在给定的时间里进行规定的总线操作,完成设备之间的信息交换。下面以存储器读为例来说明同步总线的工作方式和典型操作的工作过程。如图 7.15 所示。

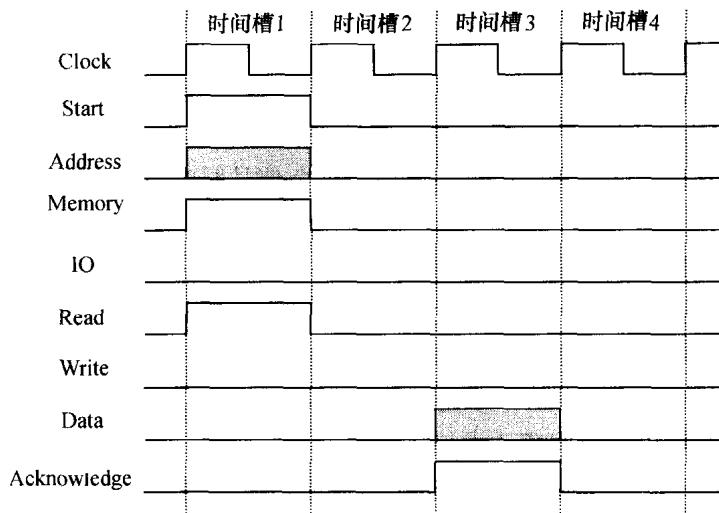


图 7.15 同步总线的存储器读过程

图 7.15 中的信号说明如下：

- **Clock**: 总线时钟；
- **Start**: 启动命令，总线命令之一，表示一个总线操作周期的开始；
- **Address**: 地址总线，给出总线操作需要访问的设备编号或者内部单元的地址；
- **Memory**: 访问存储器命令信号。信号有效时，表示总线进行存储器访问操作；
- **IO**: 访问 I/O 设备命令信号，信号有效时，表示总线进行 I/O 设备访问操作；
- **Read**: 读操作命令信号，信号有效时，表示总线主设备进行数据读操作；
- **Write**: 写操作命令信号，信号有效时，表示总线主设备进行数据写操作；
- **Data**: 数据总线，总线操作中的数据；
- **Acknowledge**: 总线应答，总线命令之一，表示总线从设备操作就绪。

图 7.15 亦称为同步总线进行存储器读操作的时序图。使用总线进行一次存储器读操作的过程被称为总线的存储器读操作周期。图中，存储器读操作周期被分为 4 个部分：时间槽 1、时间槽 2、时间槽 3 和时间槽 4。

时间槽 1 为使用总线准备时间。总线发出启动命令 Start，同时发出操作命令 Memory 和 Read 表示进行存储器读操作，还发出地址到 Address 总线指出被操作的存储单元。总线操作命令 IO 和 Write 在这个阶段无效。实际上，总线上一般不允许同时发出 Memory 和 IO 命令，也不允许同时发出 Read 和 Write 命令。根据 7.4.2 节中的知识，在总线设计时，可将 Memory 与 IO 信号复用成一条信号线 (M/IO)，也可将 Read 与 Write 命令复用成一条信号线 (R/W)。

时间槽 2 为总线设备操作时间。存储器读访问操作经过这段时间，存储器中的数据将

送到 Data 总线上,供总线设备取走。实际上,在一次总线操作过程中,时间槽 2 不是必须的。如果存储器的速度足够快,则可以将时间槽 2 去掉。

时间槽 3 为总线设备取数时间。在这段时间里,总线应答信号 Acknowledge 有效。这个信号是由存储器发出的,表示存储器已经将数据准备就绪。总线设备只有接收到这个信号以后,才知道数据总线上已经有数据可以使用。这个信号同时也用来通知总线控制器,本次总线操作可以结束了。

时间槽 4 是总线恢复时间。总线控制器在接收到应答信号 Acknowledge 以后,知道本次总线操作结束。总线控制器将收回总线使用权,同时将根据总线使用请求和仲裁的情况,准备将总线使用权提供给下一个总线主设备。至此,同步总线上进行存储器读操作的一个完整的总线操作周期结束。同样,在一次总线操作过程中,时间槽 4 也不是必须的。如果总线恢复时间足够短,则可以将时间槽 4 去掉。

同步总线的时序中,启动命令信号 Start 和应答命令信号 Acknowledge 对于总线的定时至关重要。只有启动命令信号 Start 有效,才能够开始一次总线操作。总线上所有设备,只有通过这个信号才能够了解总线当前状态。应答命令信号 Acknowledge 表示一次总线操作可以结束。上例中,如果在时间槽 3 到来时,Acknowledge 信号仍然无效,则表示存储器此时还不能提供数据,总线操作将持续下去,导致总线操作周期的延长。总线操作周期将每次增加一个等待时间槽,一直到存储器提供数据使得应答命令信号 Acknowledge 有效为止。通过这种方法,可以使一些速度不匹配的总线设备能够协调工作。可见,总线操作周期不是固定不变的。

通过同步总线工作过程的分析,可以发现同步总线具有如下特点:

- 所有总线设备在统一的总线时钟下进行总线操作。即设备与总线设备接口必须借助于总线时钟进行工作。
- 所有总线信号和命令信号必须与总线时钟同步,即总线上所有事件都在总线时钟开始时(时钟信号由 0 变为 1 时)发生。
- 所有总线操作必须以总线时钟为基本时间单位,即总线所用时钟数必须是整数。

同步总线中,无论设备的速度快慢,总线操作都必须以总线系统时钟周期为基本单位,这样必然会造成时间浪费。另外,总线周期必须设计成能够满足速度最慢的总线设备使用,而快速设备就不能高效地使用总线。但相对于异步总线,同步总线的设计、使用和调试都比较简单。因此,目前使用的总线大部分是同步总线。

2. 异步总线

异步总线最根本的特征是总线系统中没有统一的时间标志。在异步总线上,任何一个事件的发生都取决于前面一个或者一些事件的结果,所有设备以信号握手(handshaking)的方式进行联系,从而完成总线操作等工作。下面还是通过存储器读为例来说明异步总线的工作方式和典型操作的工作过程,如图 7.16 所示。

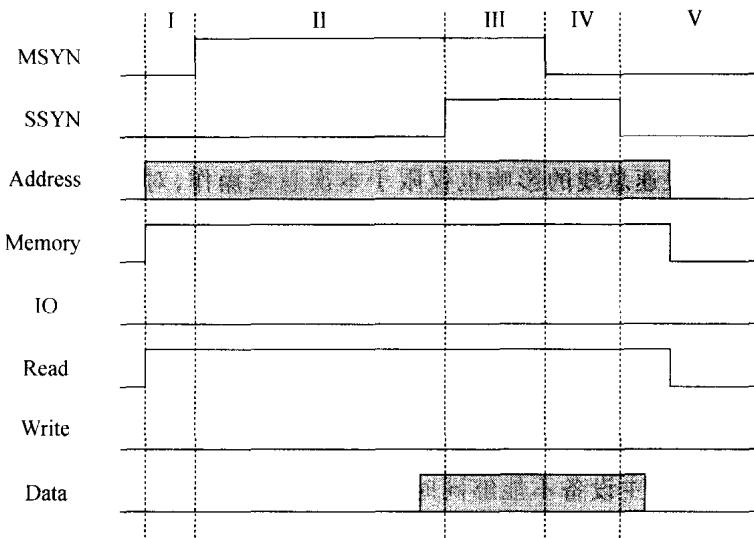


图 7.16 异步总线的存储器读过程

图 7.16 中的信号说明如下：

- MSYN：总线主设备同步请求。信号有效时，表示总线上的地址和控制信号已经有效；
- SSYN：总线从设备同步请求。信号有效时，表示总线从设备已经对总线主设备的请求做出应答，即从设备已经完成主设备要求的工作；
- Address：地址总线，给出总线操作需要访问的设备编号或者内部单元的地址；
- Memory：访问存储器命令信号。信号有效时，表示总线进行存储器访问操作；
- IO：访问 I/O 设备命令信号，信号有效时，表示总线进行 I/O 设备访问操作；
- Read：读操作命令信号，信号有效时，表示总线主设备进行数据读操作；
- Write：写操作命令信号，此信号有效时，表示总线主设备进行数据写操作；
- Data：数据总线，总线操作中的数据。

图 7.16 亦称为异步总线进行存储器读操作的时序图。异步总线存储器读操作过程可以分为 5 个阶段，对应于图中的 I、II、III、IV 和 V 5 个部分。

第一阶段 I：总线主设备将地址和操作的命令信号准备好，发出主设备同步请求信号 MSYN。

第二阶段 II：总线从设备接收到总线主设备的命令信号后，进行存储器读操作。操作完成时，读出的数据送到数据总线上。然后发出总线从设备同步请求信号 SSYN。

第三阶段 III：总线主设备接收到总线从设备同步请求信号 SSYN 后，从数据总线上接收数据，然后撤销请求 MSYN。

第四阶段 IV：总线从设备发现总线主设备撤销了请求信号，知道主设备操作完毕，就撤销从设备同步请求信号 SSYN。

第五阶段 V: 在总线主设备和总线从设备都撤销同步请求以后, 地址总线和操作的命令信号线进入恢复阶段。总线控制器经过恢复后, 可以处理后续请求。

显然, 异步定时方式与总线时钟无关。每个事件都由前一个事件引起, 而不是由时钟脉冲控制。从性能角度考虑, 相对于同步总线而言, 异步总线对于设备速度上差异的适应性更强。慢速总线设备对快速总线的影响也仅限于本次总线操作, 对下一次总线操作中快速设备的效率不会产生影响, 这是异步总线优越的方面。但是, 在总线的简单性, 总线设备接口的标准化, 总线设备的调试等方面, 异步总线都要比同步总线复杂。

7.4.4 总线仲裁

由于总线主设备在信息传送过程中需要掌握总线控制权, 而在这个传输过程中不一定有 CPU 参与工作, 因此必须使用某种方法将总线控制权交给主设备。从信息传输的角度看, 连接到总线上的总线主设备不能够同时拥有总线的使用权。一个总线系统中总线主设备获得总线控制权的过程称之为总线使用权的仲裁, 简称总线仲裁。总线的仲裁机制就是分配总线使用权的策略, 仲裁机制也被称为仲裁方式。总线仲裁是总线控制器的核心功能之一。在结构上, 总线仲裁系统可以分为请求、仲裁和应答等部分。

总线的仲裁方式有很多种。按照总线仲裁电路的位置不同, 可以将仲裁方式分为集中式仲裁和分布式仲裁。按照对总线主设备请求进行仲裁的优先权是否可以变化, 可以将仲裁方式分为固定优先权仲裁和动态优先权仲裁。按照总线控制器是通过仲裁算法还是通过查询来进行仲裁, 可以将仲裁方式分为设备请求仲裁和控制器查询仲裁。按照总线仲裁电路能够同时处理的请求数量, 可以将仲裁方式分为串行仲裁和并行仲裁。这些不同的仲裁方式使得总线的使用具有极大的灵活性。

1. 集中式和分布式仲裁

在集中式仲裁方式中, 需要有一个集中式的总线仲裁电路, 所有的总线主设备向该仲裁电路发出请求。仲裁电路根据一定的策略选择向其中的某个总线主设备发出应答信号, 标志该主设备拥有总线使用权。集中式仲裁电路可以是独立的部件, 也可以是 CPU 的一部分。集中式仲裁方式的系统构成如图 7.17(a) 所示。

在分布式仲裁方式中, 没有集中式仲裁电路, 所有的总线主设备均具有自己的仲裁电路。当总线主设备发出请求时, 各个仲裁电路间根据一定的策略互相作用, 共同决定总线使用权的归属。分布式仲裁方式的系统构成如图 7.17(b) 所示。

无论是集中式还是分布式仲裁, 每个总线主设备都有两条线连接到总线仲裁电路: 总线请求信号 (Bus Request, BR) 和总线应答信号 (Bus Grant, BG)。它们的目的都是为了指定一个总线主设备, 然后由这个主设备启动与其他设备间的数据传输。

集中式仲裁具有集中式仲裁电路, 仲裁过程以及总线设备接口十分简单。但是, 与所有的集中式控制器一样, 其可靠性不高。一旦仲裁电路出现故障, 将导致整个系统无法工作。

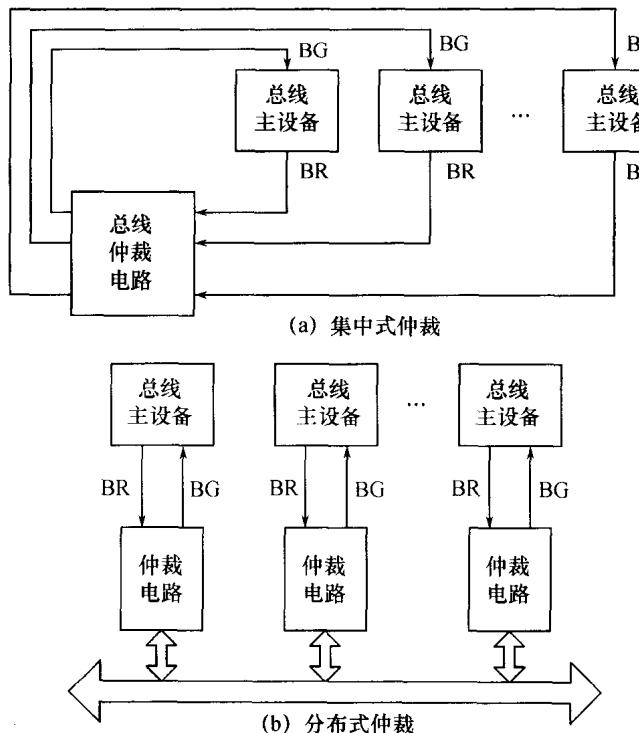


图 7.17 集中式仲裁与分布式仲裁

采用集中式仲裁电路的另一个问题是设备扩展的难度较大。如果需要增加总线主设备，需要对集中式仲裁电路进行修改。

与集中式仲裁相比，分布式仲裁的优点是线路可靠性高，不会因为某个总线主设备的仲裁电路故障而导致系统不能够工作。但是，系统往往需要进行超时判断，以确定总线主设备是否还在正常工作。另外，使用分布式仲裁时，设备扩展灵活性较大，设备接插比较随意。但是，由于每个总线主设备需要在其接口电路中包含仲裁电路，这将导致设备设计的复杂性加大。

2. 固定优先权仲裁和动态优先权仲裁

在仲裁电路中，一个非常重要的问题就是优先权的分配算法，即在所有总线主设备都提出请求的情况下，哪一个主设备可以获得总线使用权。总线仲裁的优先权分配算法可以分为固定优先权和动态优先权两类。

固定优先权仲裁是相对简单的仲裁策略。在这种仲裁方式中，每个总线主设备各有其固定的优先级。下面以 4 个总线主设备为例子，研究固定优先权的集中式和分布式的仲裁电路。

假设有 4 个总线主设备：设备 0、1、2 和 3，它们使用总线的请求依次对应为 BR0、BR1、

BR2 和 BR3, 应答信号对应为 BG0、BG1、BG2 和 BG3。总线采用固定优先权仲裁, 设备 3 优先权最高, 依次递减, 设备 0 优先权最低。

根据这些假设可知, 当设备 3 的请求信号 BR3 有效时, 由于设备 3 的优先级最高, 其他所有设备的应答信号都应该被封锁; 当设备 2 的请求信号 BR2 有效时, 设备 1 和设备 0 的应答信号应该被封锁; 当设备 1 的请求信号 BR1 有效时, 设备 0 的应答信号应该被封锁; 只有当其他所有设备都没有请求时, 设备 0 的请求才能被响应。那么, 是不是设备 3 提出使用总线的请求以后, 就可以立即得到总线的使用权呢? 答案是否定的。无论设备的优先权多高, 只有当总线空闲时, 请求才会被响应。记总线状态为 BUSY, BUSY 为 1 时表示总线忙, BUSY 为 0 时表示总线空闲。根据以上的分析, 可得每个设备的应答信号所对应的表达式:

$$\begin{aligned} BG0 &= \overline{BUSY} \cdot \overline{BR3} \cdot \overline{BR2} \cdot \overline{BR1} \cdot BR0 \\ BG1 &= \overline{BUSY} \cdot \overline{BR3} \cdot \overline{BR2} \cdot BR1 \\ BG2 &= \overline{BUSY} \cdot \overline{BR3} \cdot BR2 \\ BG3 &= \overline{BUSY} \cdot BR3 \end{aligned} \quad (7.1)$$

式(7.1)的逻辑图如图 7.18 所示。

图 7.18 中画出的逻辑图实际上是集中式固定优先权仲裁电路。在分布式仲裁电路中, 每个总线主设备均有一个仲裁部件, 用于接收所有部件的请求信号, 并产生该部件的应答信号。与式(7.1)对应的固定优先权分布式仲裁电路如图 7.19 所示。

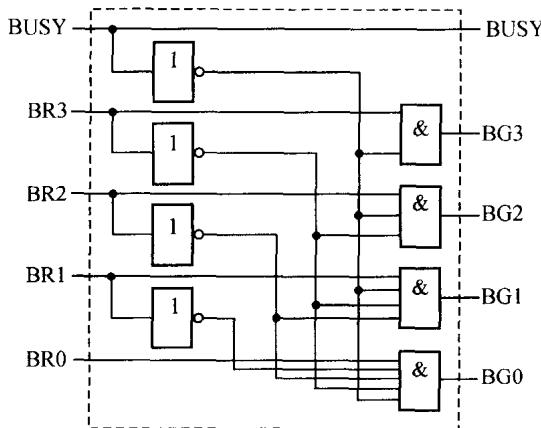


图 7.18 固定优先权集中式仲裁电路

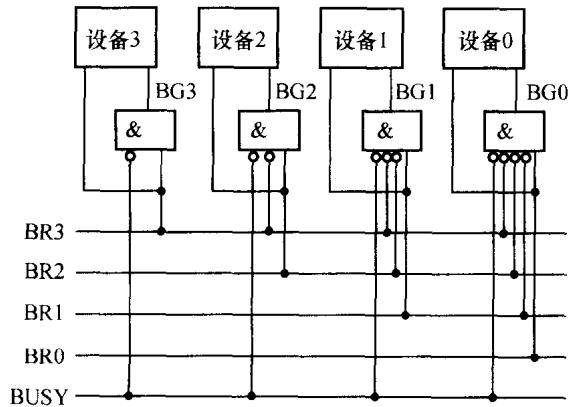


图 7.19 固定优先权分布式仲裁电路

固定优先权电路虽然结构简单, 但是存在某种明显的不合理性。在很多实际应用中, 一个总线主设备如果长时间具有很高的优先权, 而另外一些主设备始终优先权很低, 这对于提高系统的效率可能是不利的。如果希望所有总线主设备都具有公平的机会来获得总线的使用权, 就必须使用动态优先权仲裁策略。

在动态优先权算法中, 每个总线主设备的优先权是随时间变化的。常用的动态优先权

算法有以下几种：

- 简单的轮转优先权(Simple rotating priority)策略：假定有 n 个总线主设备和 n 级优先权，每个主设备对应一级优先权；假定第 n 级优先权最高，第1级优先权最低。每次仲裁后，具有最低(1级)优先权的总线主设备的优先权被置为最高(n 级)，而其他主设备的优先权均减1。

- 相关的轮转优先权(Acceptance-dependent rotating priority)策略：假定有 n 个总线主设备和 n 级优先权，每个主设备对应一级优先权；假定第 n 级优先权最高，第1级优先权最低；假定获得总线使用权的主设备的优先级为 i 。所有优先权比 i 大的总线主设备的优先级保持不变，所有优先权比 i 小的主设备的优先级加1，优先权为 i 的主设备的优先级被置为最低(1级)。通常所说的轮转优先权(Rotating priority)策略指的就是这种策略。

- 随机优先权(Random priority)策略：每次总线使用权仲裁以后，所有总线主设备的优先权将通过一个伪随机数发生器进行重新分配。

- 最近最少使用(Least Recently Used, LRU)策略：最长时间没有获得总线使用权的总线主设备具有最高优先权。

可见，动态优先权仲裁策略比固定优先权仲裁策略更加合理，使用范围也更加广泛。但是，动态优先权算法的实现比固定优先权算法的实现要复杂得多，因为每次仲裁后都需要对优先权进行调整。实际上，动态优先权电路往往是一个时序网络，而不象固定优先权电路仅仅使用组合逻辑网络就可以完成。

3. 设备请求仲裁与控制器查询仲裁

在上面的讨论中，总线上的设备都是以“请求-应答”方式工作的，即由设备发出使用总线的请求，总线仲裁电路根据一定的策略，对该请求进行仲裁以决定总线主设备是否获得总线的使用权。

另外还有一种查询式仲裁方法，当有使用总线的请求时，控制器将通过查询地址来定位总线主设备。总线主设备通过“请求-查询-应答”方式来获得总线使用权。如图7.20所示。

当控制器发现有总线请求时，查询工作由总线仲裁电路开始。总线仲裁电路根据优先权算法产生查询地址，并将它们发送到查询地址总线上。然后，每个总线主设备对接收到的查询地址进行译码，以确定被查询的主设备。如果被查询的总线主设备有使用总线的请求，则该主设备将获得总线使用权。显然，总线主设备获得总线使用权的优先权顺序就是产生的查询地址的先后顺序。

由于优先权是通过仲裁电路产生的设备查询地址顺序来确定的，因此该电路的优先权算法比较容易控制和修改。图7.20所示的是集中式查询仲裁电路。如果将查询地址产生电路分配到每个总线主设备上就可以构成分布式查询仲裁电路。当某个总线主设备被查询定位时，将由它对应的仲裁电路产生下一个被查询主设备的地址。

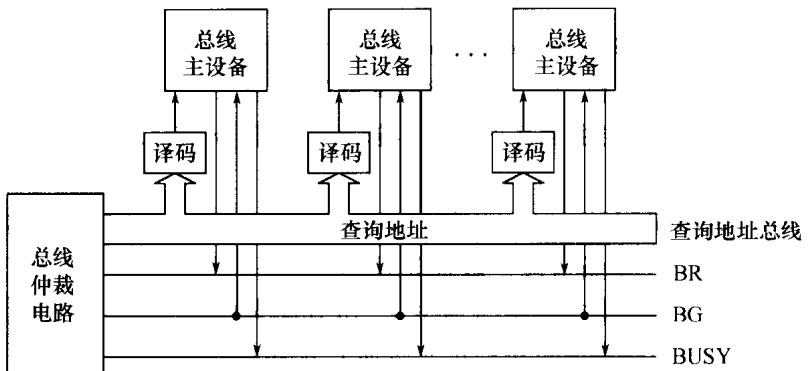


图 7.20 查询式仲裁电路

查询方式还可以采用软件来实现,即由 CPU 产生查询地址。这样做的好处是节省硬件电路,总线主设备的优先级容易调整。带来的问题主要是 CPU 的开销加大,系统速度比使用硬件时要低,因此采用软件查询方式的总线仲裁策略在实际使用中非常少见。

4. 串行仲裁与并行仲裁

串行仲裁一次只能处理一个总线主设备发出的请求信号。下面以固定优先权的串行仲裁电路为例来说明该种仲裁方式的特点。

固定优先权串行仲裁电路最显著的特点是优先权信号在一组总线主设备上传送,形成雏菊链(Daisy chain)结构,简称链式结构。当总线仲裁电路发现总线请求后,它发出一个总线授权信号,这个信号被依次串连到所有的总线主设备上。当物理上离仲裁电路最近的那个总线主设备得到授权信号时,由这个主设备检查是否是它发出了总线请求。如果是,该总线主设备就得到了总线的使用权,并禁止授权信号继续往下传播。若该总线主设备没有发出总线请求,则将授权信号传播到下一个主设备。接到授权信号的这个总线主设备重复上述动作,直到有一个主设备接管总线为止。

为了处理优先权,雏菊链可以使用应答(Grant)信号、请求(Request)信号或者使能(Enable)信号构成仲裁链。其中,应答信号链仲裁电路是最常用的固定优先权串行仲裁电路,其工作原理图如图 7.21 所示。

应答信号链串行仲裁电路中,所有总线主设备的请求信号(BR)经过“线或”(wired-OR)连接到同一根请求信号线上。当有总线主设备发出请求信号以后,总线仲裁电路将应答信号(BG)作为授权信号来确定发出请求的主设备。应答信号在总线主设备间依次传输构成应答信号链。一旦发现某个总线主设备发出了请求,则该主设备就获得了总线使用权,并且禁止授权信号(应答信号)继续往下传输。请求信号链仲裁电路将请求信号作为授权信号,而使能信号仲裁电路将设备的使能信号作为授权信号。无论采用哪一种链式仲裁电路,有一条原则是公共的:当有总线主设备获得总线使用权后,建立的总线忙(BUSY)信号将阻

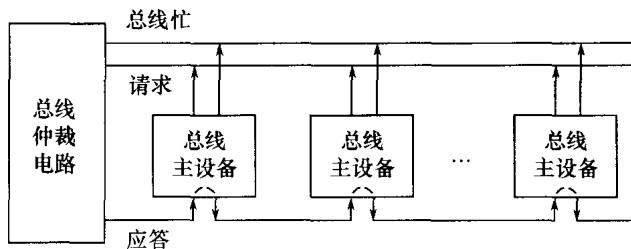


图 7.21 应答信号链串行仲裁电路

止所有主设备申请总线使用的处理。

串行仲裁电路具有结构简单,可扩展性好的优点。但是,由于授权信号传输的原因,串行仲裁电路的仲裁速度较慢。因此,串行仲裁电路适用于低速设备。如果对速度要求较高,则需要采用并行仲裁策略。

并行仲裁一次可以处理多个总线主设备发出的请求信号,并对它们进行仲裁以分配总线使用权。在仲裁电路中,一个重要的问题就是对总线主设备的优先权进行分配,优先权分配算法可分为固定优先权和动态优先权两大类,而仲裁电路又可以分为集中式和分布式的两种。图 7.17、图 7.18 和图 7.19 给出的就是并行仲裁电路。

与串行仲裁比较,并行仲裁电路的特点是仲裁速度较快,原因是仲裁电路可以同时接收多个请求信号并进行仲裁。但从式(7.1)可知,当有 n 个总线主设备时,需要使用 n 个输入端的与门才能形成 BG0。这样,当同时发出请求的总线主设备较多时,仲裁电路中的与门输入端会非常多,不便于实现。为了克服这个缺点,可以采用级联式仲裁或者多级仲裁方式。级联式仲裁或者多级仲裁方式的基本思想都是将请求信号分组。

假定分组后,一个仲裁电路模块可以同时对 4 个总线主设备发出的请求进行并行仲裁,那么具有 16 个请求源的级联式并行仲裁电路如图 7.22 所示。

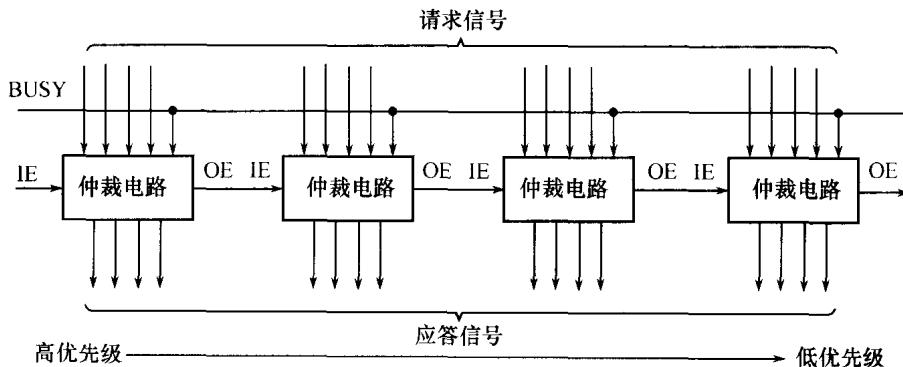


图 7.22 级联式并行仲裁

图 7.22 所示的电路中增加了两个信号,一个是输入使能(Input Enable, IE),另外一个是输出使能(Output Enable, OE)。当 IE 有效时,才允许本级仲裁电路工作;只要本级的 4 个输入端中有一个输入端有信号,本级仲裁电路在完成仲裁工作的同时使 OE 失效,禁止下级仲裁电路工作;当本级所有的请求端均没有输入信号时,OE 信号有效使下一级仲裁电路工作,即本级仲裁电路的 OE 信号与下一级仲裁电路的 IE 信号相连。图 7.22 中每一组的仲裁表达式为:

$$\begin{aligned}
 BG0 &= IE \cdot \overline{BUSY} \cdot \overline{BR3} \cdot \overline{BR2} \cdot \overline{BR1} \cdot BR0 \\
 BG1 &= IE \cdot \overline{BUSY} \cdot \overline{BR3} \cdot \overline{BR2} \cdot BR1 \\
 BG2 &= IE \cdot \overline{BUSY} \cdot BR3 \cdot BR2 \\
 BG3 &= IE \cdot \overline{BUSY} \cdot BR3 \\
 OE &= IE \cdot \overline{BUSY} \cdot \overline{BR3} \cdot \overline{BR2} \cdot \overline{BR1} \cdot \overline{BR0}
 \end{aligned} \tag{7.2}$$

从图 7.22 所示仲裁电路的工作过程可知,该仲裁电路只能同时处理 4 个总线主设备发出的请求。级联式并行仲裁实际上是并行仲裁和串行仲裁的结合,组内是并行仲裁,组间是串行仲裁。

级联式并行仲裁是扩大并行仲裁电路的有效途径。优点是扩充电路比较简单。缺点是当请求信号较多时,电路延迟增大,低优先级的请求获得总线使用权的速度慢,各组应答信号形成的时间相差很大。因此级联式并行仲裁不适合于有大量请求的情况。

与级联式仲裁不同,多级仲裁则可以同时对所有的请求信号进行并行仲裁。同样假定分组后,一个仲裁电路模块可以同时对 4 个总线主设备发出的请求进行并行仲裁,那么具有 16 个请求源的两级并行仲裁电路如图 7.23 所示。

图 7.23 中,第一级仲裁电路对组内请求进行并行仲裁,从每组中选出优先级最高的请求;第二级仲裁电路对组间请求进行并行仲裁,确定具有最高优先级的请求所在的组。通过两级并行仲裁电路,就可以选出最高优先级组中的最高优先级的请求。

多级并行仲裁也是扩大并行仲裁电路的有效途径。从电路结构比较,采用多级并行仲裁结构,其电路比级联式并行仲裁要复杂,器材消耗量也大。但采用这种结构,每个请求从输入到输出应答的电路延迟相同,电路的时间特性比较好。按照这种方法,还可以进一步扩展到更多级的仲裁电路,以适应更多请求信号的需要。

总线的仲裁机制是总线的核心技术问题之一,对于总线系统的设计和使用都有很大的影响。在设计或者使用总线时,必须清楚总线的仲裁机制。

需要说明的是,虽然仲裁机制是总线的核心机制之一,但并不是所有总线系统都明确定义它的仲裁策略。例如,PCI 总线并没有规定仲裁策略,具体的仲裁策略由设计和生产总线控制器的厂商决定。

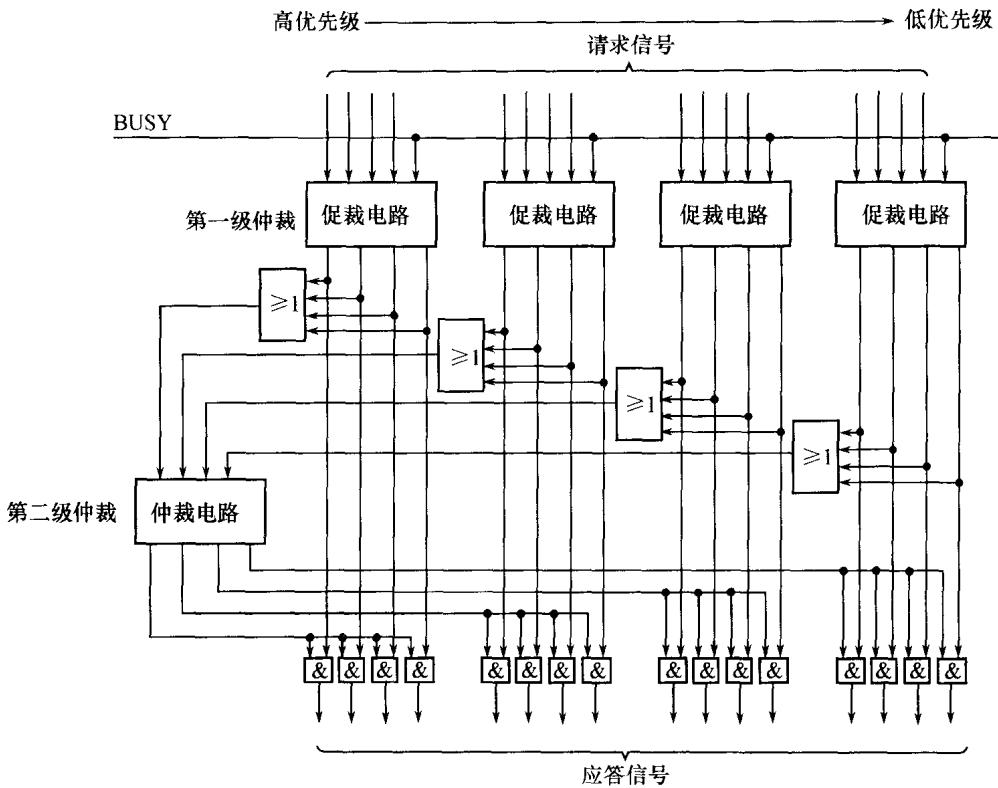


图 7.23 两级并行仲裁

7.4.5 总线数据传送方式

总线的基本功能是信息传输。总线上的信息交换包括两个阶段：地址命令阶段（简称地址期）和数据传送阶段（简称数据期）。总线上数据传送的方式有三种：基本数据传送方式、成组数据传送方式以及特殊数据传送方式。大部分情况下，总线都使用基本数据传送方式。为提高信息交换的效率，可以采用成组数据传送方式。在一些特殊的应用场合，有些总线还提供特殊的数据传送方式。

1. 基本数据传送方式

所有的总线都支持读操作和写操作。总线最基本的数据传送方式就是单数据读和单数据写。复用型总线和专用型总线的基本数据传送方式如图 7.24 所示。这种方式的典型特征是数据传送阶段只有一次数据传送操作。

在复用型总线中，总线先用于指定地址，然后用于传输数据。对于总线主设备对总线从设备的写操作，地址命令的持续时间需要保证能够寻址到从设备且从设备可以识别总线上的地址信息，地址命令结束后就可以进行数据发送。对于总线主设备对总线从设备的读操

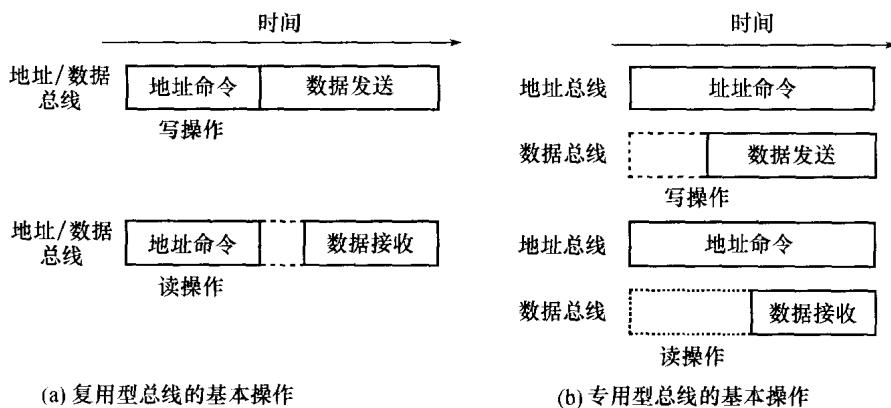


图 7.24 基本数据传送方式

作,由于从设备提供数据的速度可能较慢,故从时间上看,在地址命令撤消和开始接收数据间可能存在时间延迟。

在专用型总线中,要求地址信息放到地址总线上并保持到数据出现在数据总线上之后。对于总线主设备对总线从设备的写操作,地址一旦稳定,主设备就把数据放到数据总线上,这时从设备已经可以识别地址总线上的地址信息。对于总线主设备对总线从设备的读操作,从设备一旦识别出地址并准备好数据,就将数据放到数据总线上。

2. 成组数据传送方式

总线采用成组数据传送方式的目的是为了提高总线信息交换的效率。目前较高性能的总线对成组数据交换都提供了支持。这种方式要求数据传送过程中的源数据和目的数据为一片连续的单元,构成地址连续的数据块。在基本数据传送方式中,地址命令处理和数据传送过程在时间上是串行的,并且占用的时间大致相当。如果对于连续地址空间的数据采用基本数据传送方式,地址命令的处理将占用大量的时间。因此可以采用类似全串行 DMA 的思想,在一个首地址后面跟着一串数据的处理方法,以减少数据交换过程中地址命令处理的时间。采用成组数据交换,省去了首数据以外其他数据的地址命令处理时间,整个信息交换时间得到较大幅度的减少。图 7.25 是成组数据传送方式的时序图,其中(a)为复用型总线的时序,(b)为专用型总线的时序。

在成组数据传送中,还存在一个问题,就是何时数据传送结束。在实际的总线中,往往采用两种方法:其一是采用固定长度的数据传送,使用计数器来控制数据传送的结束;其二是定义一条专用的信号线,信号线有效进行成组数据传送,信号线失效,表示一次成组传送使用总线结束。

下面通过一个例子,进一步分析成组数据传送方式的特点。

某总线地址命令处理时间为 1 个时钟周期,一个数据传送的时间为 2 个时钟周期,若传

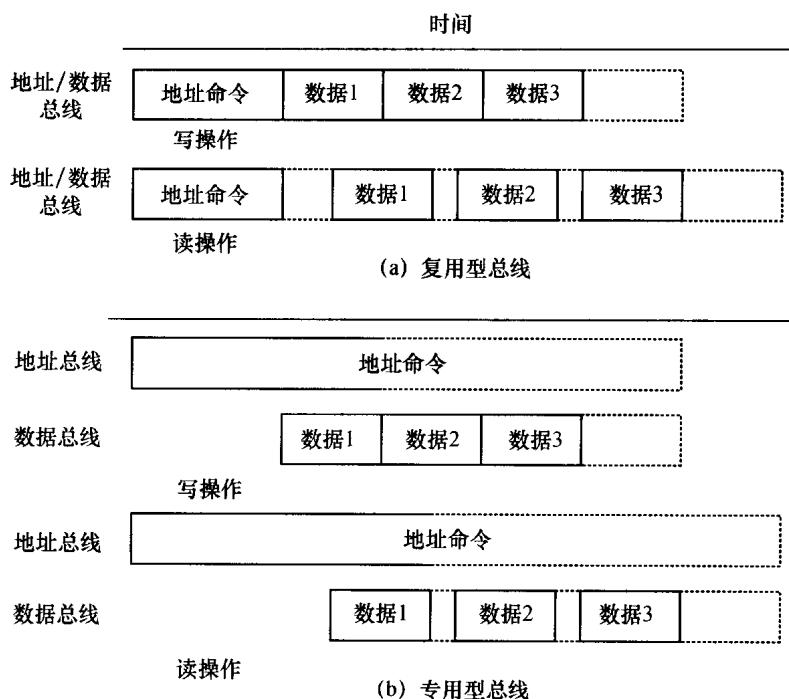


图 7.25 成组数据传送方式

送 1 024 个单位数据,采用基本数据传送方式,则每个数据传送需要 3 个时钟周期,传送 1 024 个单位数据总共使用时间为:

$$3 \text{ 周期}/\text{数据} \times 1 024 \text{ 数据} = 3 072 \text{ 周期}$$

如果采用成组数据交换,每组数据为 4 个,即每 4 个数据需要一个首地址,传送一组数据需要的时间为:

$$1 \text{ 周期}/\text{地址} \times 1 \text{ 地址}/\text{组} + 2 \text{ 周期}/\text{数据} \times 4 \text{ 数据}/\text{组} = 9 \text{ 周期}/\text{组}$$

故传送这 1 024 个数据需要的时间为:

$$1 024 \text{ 数据} \div 4 \text{ 数据}/\text{组} \times 9 \text{ 周期}/\text{组} = 2 304 \text{ 周期}$$

在这种情况下,平均每个数据的传输时间为:

$$2 304 \text{ 周期} \div 1 024 \text{ 数据} = 2.25 \text{ 周期}/\text{数据}$$

如果每组数据增加为 256 个,即每 256 个数据一个首地址,传送一组数据需要的时间为:

$$1 \text{ 周期}/\text{地址} \times 1 \text{ 地址}/\text{组} + 2 \text{ 周期}/\text{数据} \times 256 \text{ 数据}/\text{组} = 513 \text{ 周期}/\text{组}$$

故传送这 1 024 个数据需要的时间为:

$$1 024 \text{ 数据} \div 256 \text{ 数据}/\text{组} \times 513 \text{ 周期}/\text{组} = 2 052 \text{ 周期}$$

这种情况下,平均每个数据的传输时间为:

$$2 052 \text{ 周期} \div 1 024 \text{ 数据} \approx 2.004 \text{ 周期}/\text{数据}$$

可见,成组数据传送方式中,当每组数据很长时,平均数据传输时间中地址命令处理时间的影响几乎可以忽略不计。这是总线采用成组数据传送方式的最大优越性。

在不同的总线系统中,成组数据(Group Data)传送方式有不同的名称,如块(Block)方式、猝发(Bursts)方式、流水(Pipeline)方式、页(Page)方式等。

3. 特殊数据传送方式

在不同的应用中,对总线的数据交换可能有一些特殊要求。虽然这些特殊的要求可以采用多种数据传送形式组合实现,但是其效率往往较低,而且有时还难以保证信息交换的正确性。这里主要介绍两种特殊的数据传送方式:“读后写”和“写后读”。

(1) “读后写”(Write - After - Read)数据传送方式

“读后写”又被称为“读-修改-写”(Read - Modify - Write)。下面举例说明对这种特殊数据传送方式的需求场合。

假定总线上有计数器C,用于对总线设备D1和D2的操作进行计数,即总线设备D1和D2每进行一次操作都对将计数器C进行“+1”计数。采用常规操作,在图7.26(a)所示的情况下,计数器C中最后的值将发生错误。计数器C的正确值应该为“+2”以后的值,而现在却为“+1”以后的值。

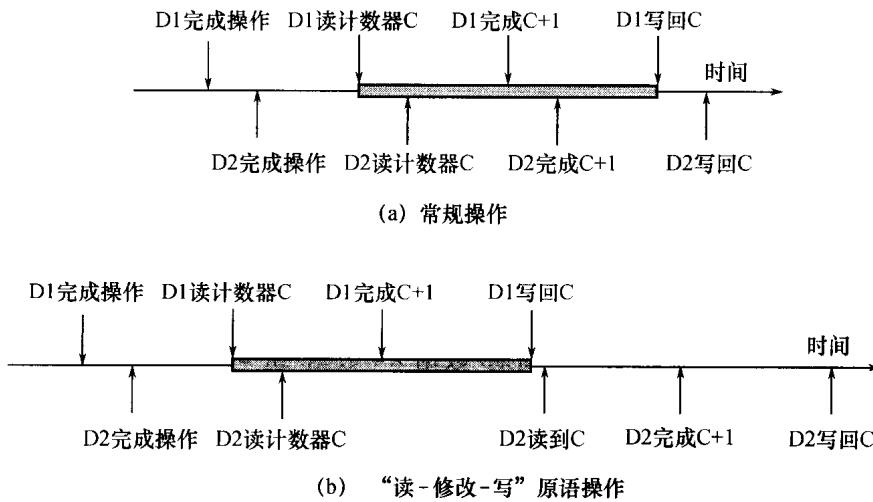


图 7.26 总线设备 D1 和 D2 对计数器 C 的计数操作

出现这个错误的原因在于D1对C的修改还没有完成,D2读到计数器C的值是错误的。如果在D1完成操作之前,即在图7.26(a)中所示的阴影部分时间里,其他总线设备都不能使用计数器C,就可以保证正确性,如图7.26(b)所示。

“读后写”数据传送方式利用了“读-修改-写”原语操作保证了对计数器C修改的正确性,该操作包括三个部分:读数据、修改数据、数据写回。它们的时序关系如图7.27所示。



图 7.27 “读 - 修改 - 写”总线操作时序

采用“读 - 修改 - 写”方式，操作的整个过程不允许被打断，必须一次完成。对于总线操作而言，进行“读 - 修改 - 写”的总线设备一旦获得了总线的使用权，就一直占用总线，直到完成数据写为止。具有这种特征的操作，称之为原语操作。有关原语操作的知识，将在“操作系统”等课程中学习。

(2) “写后读”(Read - After - Write)数据传送方式

“写后读”数据传送方式往往用于对数据传送的可靠性要求非常高的场合，对写的数据立即取回进行校验，以确定数据是否被正确写入。“写后读”过程的时序关系如图 7.28 所示。

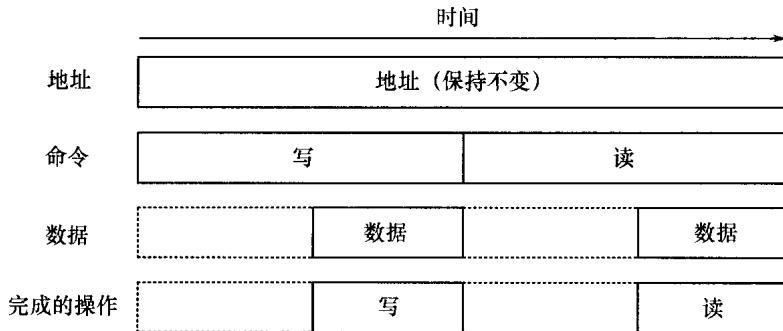


图 7.28 “写后读”总线操作时序

与“读后写”相似，“写后读”中的总线操作包括两个部分：写数据和读回数据。这个操作过程也是原语操作，整个过程不允许中断，必须一次完成。

上面讨论了两种特殊的数据传送方式。总线的实际使用中，为了一些特殊要求，往往需要设计一些特殊的传送方式。例如，Intel 系列 80486 以后的微处理器，由于芯片内部使用了高速缓存 Cache，便设计了专门用于 Cache 操作的总线周期。再如，随着超大规模集成电路 VLSI 的广泛使用，很多总线上专门设计了用于电路验证的数据通路（例如 JTAG 通路），用于对集成电路进行测试。实际上，在设计总线的时候，可以根据具体的需要来设计各种特殊的

总线数据传送方式,以便针对应用,获得高效率。

7.5 PCI 总线标准

随着计算机技术的不断提高,计算机各部件对交换信息的速度要求越来越高。作为常见的互连结构,总线技术得到了长足的发展,出现了各种各样的总线。总线的名称通常以该种总线的标准命名。

总线标准指的是通过总线将各个设备连接成一个系统所必须遵循的规范。总线标准为各种设备的互连提供了一种标准界面,只要设备与总线的接口符合这种规范,就可以直接连接到总线上。一般情况下,总线标准有两类:正式标准和业界标准。正式标准是由 IEEE(电气电子工程师协会)或 CCITT(国际电报电话咨询委员会)等国际标准化组织正式确定、承认和公布,并有严格的标准。业界标准就是首先由某一厂家提出,而后又得到了其他厂家的广泛采用。业界标准通过国际标准化组织的认可或改良就成为正式标准。

总线标准通常需要包括以下内容:

- 物理特性:即总线的物理连接方式,包括总线的根数,总线的插头,插座的形状,信号线的排列方式等。
- 功能特性:用来描述总线中每一根线的功能。
- 电气特性:用来定义每一根线上信号的传递方向及有效电平范围。例如某设备的地址总线是输出信号线,信号线高电平有效,电平符合 TTL 电平的定义。
- 时间特性:用来定义每根信号线在什么时间有效。也就是说,只有规定了总线上各信号有效的时序关系,总线设备才能正确无误地使用总线。

目前使用的总线标准有很多,为了更好地理解本章的知识,本节主要介绍 PCI 总线标准中相关的内容。

7.5.1 PCI 总线概述

总线的标准化对于提高计算机系统的性能和方便系统的开发是至关重要的。从总线的发展历史上看,出现了 ISA、EISA 和 PCI 等 I/O 总线标准。它们的比较见表 7.1。

表 7.1 ISA、EISA 和 PCI 的比较

比较内容	ISA	EISA	PCI
数据传输位数	8/16	8/16/32	32/64
峰值传输率	8 MB/s	33 MB/s	533 MB/s
系统配置能力	人工配置	有条件地自动配置	自动配置,即插即用
设备驱动程序	依靠硬件类型	依靠硬件类型	与硬件无关

(续表)

比较内容	ISA	EISA	PCI
可连接的设备	低速设备	中速设备	允许高速设备
插座的引脚数	98	188	124(32位)/188(64位)
物理尺寸	大、中、小齐备	大小相对灵活	32位/64位两类,尺寸小
成本	低	高	中、低

ISA (Industry Standard Architecture) 总线是采用 80286 芯片的 IBM PC/AT 中使用的总线,故又称为 PC - AT 总线,它是 IBM 公司为 80286 及其兼容机制定的总线标准。EISA (Extended Industry Standard Architecture) 总线标准由 COMPAQ 等公司制定,EISA 总线标准兼容 ISA 总线标准。由于 ISA 和 EISA 总线的峰值传输率不高等原因,它们已经不能适应当前计算机系统工作的要求。目前,ISA 和 EISA 这两种总线标准已经被淘汰,被 PCI 总线标准所替代。

PCI (Peripheral Component Interconnect, 设备部件互连) 标准由 Intel 公司制定,由包括 Intel 在内的几十家公司组成的 PCI 特别兴趣组 (Special Interest Group, SIG) 发布实施。PCI 1.0 版于 1992 年发布,1993 年发布了 PCI 2.0 版,1995 年发布了目前广泛使用的 PCI 2.1 版,最新版本是 1999 年发布的 PCI 2.2 版。

PCI 总线是一种高性能的总线,目前已经广泛使用于 PC、工作站、服务器等各种计算机系统中。一种包含 PCI 总线的典型计算机系统结构如图 7.29 所示。

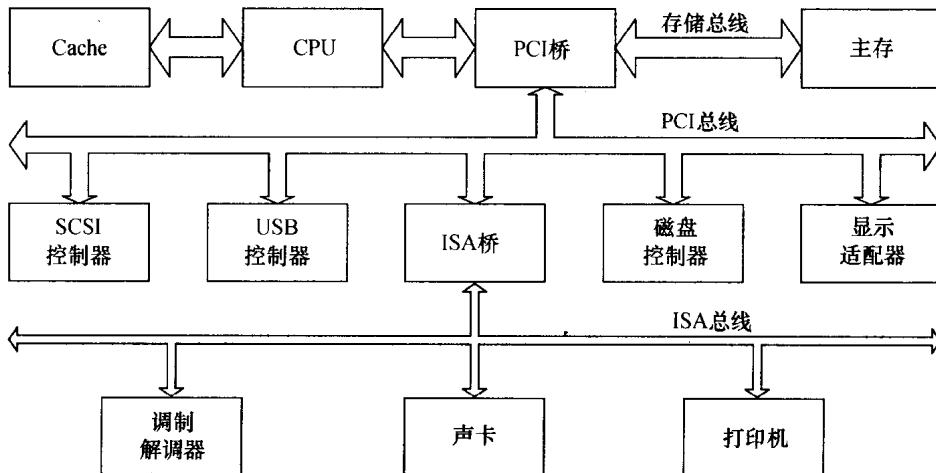


图 7.29 包含 PCI 总线的多级总线结构

图 7.29 所示的多级总线结构由三级总线构成。第一级是带宽很高的系统总线,连接的设备包括 CPU、Cache 和主存;第二级就是 PCI 总线,连接速度较快的 I/O 设备,PCI 总线通

过 PCI 桥与系统总线连接;第三级是 ISA 总线,连接速度较慢的 I/O 设备,PCI 总线通过 ISA 桥与 ISA 总线连接。图中宽度较窄的箭头对应的总线带宽也较小。

PCI 总线的部分特点如下:

(1) 总线宽度大。PCI 总线的总线宽度为 32 位或 64 位。PCI 总线支持 64 位总线扩展。这样 PCI 总线一次可以传输的数据量较大,其总线宽度和现代微处理器字长相当。

(2) 信号较少,总线信号使用经济。部分信号线采用复用型策略,例如地址信号和数据信号复用,总线命令信号与字节使能信号复用等。PCI 总线信号将在下节中介绍。

(3) 工作频率高。PCI 总线是一种同步总线,所有的总线操作都和系统时钟同步。PCI 总线的工作时钟频率为 33 MHz 或 66 MHz。在 PCI 2.1 版本中增加了对 66 MHz 的总线操作的支持。

(4) 与处理器独立。处理器可以高效使用 PCI 总线;设备可以独立于处理器升级。

(5) 通过桥接器(Bridge)构成多级总线结构。使用 Host/PCI 桥(通常被称为北桥)可以实现 PCI 总线与系统总线的连接;使用 PCI/ISA 桥(通常被称为南桥)可以实现 PCI 总线与 ISA、EISA 等总线的连接。

(6) 采用集中式仲裁方式,支持隐蔽式仲裁。关于 PCI 的仲裁方式将在 7.5.4 节中介绍。

(7) 支持成组数据传送方式,数据传输率高。PCI 总线理论上的数据传输率可达 533 MB/s,这样的性能基本能够满足当前设备输入/输出数据传输量的要求。关于 PCI 总线的数据传送方式将在 7.5.5 节中介绍。

(8) 提供自动配置能力。使用配置寄存器来支持设备的自动识别和配置。

(9) 可靠性高。在地址、命令和数据线上提供奇偶校验。

与其他一些总线标准相比较,PCI 总线还采用了很多最新的工业技术,如即插即用(Plug and Play, PnP)技术、3.3 V 电压标准电源技术、表面封装(Surface Mount Technology, SMT)和高密度组装技术等。关于这些技术及其特点,有兴趣的读者可参阅相关资料。

7.5.2 PCI 总线信号

PCI 可以配置成 32 位或 64 位总线。根据 PCI 总线的要求,作为 PCI 总线上的设备,至少需要使用 49 根信号线,这些信号称为基本信号。此外,PCI 规范还定义了 51 根可选的扩展信号线。如图 7.30 所示。

图 7.30 中信号的方向是从设备的角度来看的,信号后面有“#”标志的表明该信号是低电平有效,没有“#”标志的为高电平有效。PCI 的基本信号按照功能可以分为以下几组:

- 地址和数据信号:包含 32 根分时复用的地址/数据线。依 7.4.2 节可知,此时需要使用控制线来指明信号线上传送的是地址还是数据信号;
- 接口控制信号:控制数据交换的时序,使发送端和接收端协调工作;
- 错误报告信号:用于报告奇偶校验错误以及其他错误;

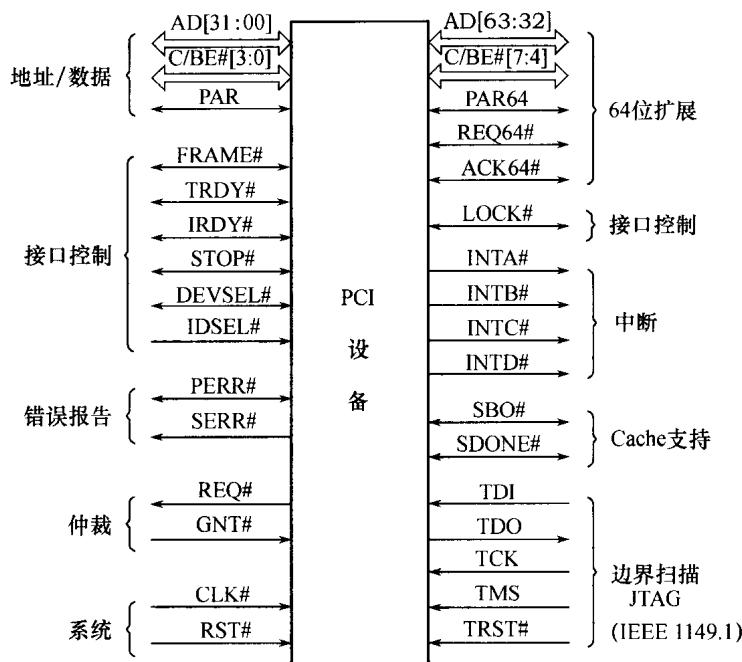


图 7.30 PCI 总线的基本信号和扩展信号

- 仲裁信号：是非共享的专用信号线。每个 PCI 总线主设备均有自己的一对仲裁信号线，它们直接与总线仲裁电路相连；
- 系统信号：包括时钟和复位信号。

PCI 的扩展信号按照功能可以分为以下几组：

- 64 位总线扩展信号：包括 32 位分时复用的地址和数据信号线 AD[63:32]，它们与 PCI 基本信号 AD[31:0] 相结合形成 64 位地址/数据信号线。同样，依 7.4.2 节可知，此时需要使用控制线来解释地址数据总线上的信号。相比于基本信号，本组中增加了两根控制线，用于协调两个 PCI 设备使用 64 位总线；
- 接口控制信号：LOCK 信号，用来锁定总线；
- 中断信号：提供给那些必须产生中断服务请求的设备使用。同仲裁信号一样，它们也是非共享的专用信号。每个 PCI 设备通过自己的中断信号线和中断控制器连接；
- 高速缓存 (Cache) 支持信号：用来支持 PCI 上的存储器设备，使得存储器设备的数据可以缓存到处理器或其他设备上的高速缓存中。为保证数据的一致性，这些信号采用高速缓存侦听 (snoopy) 协议，支持高速缓存写回 (write-back) 和写直达 (write-through) 策略。关于高速缓存 Cache、数据一致性协议、写回和写直达策略等知识将在“计算机体系结构”课程中介绍。
- JTAG/边界扫描信号：用以支持 IEEE 1149.1 标准定义的测试程序，对设备进行标准

的功能和故障测试。

图 7.30 左排表示的信号就是 PCI 的基本信号,这些信号的说明列于表 7.2。

表 7.2 PCI 基本信号

功能组	信号	主设备	从设备	描述
地址 数据	AD[31:0]	√	√	复用的地址和数据信号
	C/BE#[3:0]	√		复用的指示总线命令和指示字节有效信号
	PAR	√		地址或数据校验位
接口	FRAME#	√		总线周期启动信号,指出 AD 和 C/BE 信号已发出
	IRDY#	√		主设备就绪
控制	TRDY#		√	从设备就绪
	STOP#		√	从设备需要立即中止传送
	IDSEL#	√		主设备选择信号,选定读配置区
报错	DEVSEL#		√	从设备选择信号,从设备被选中
	PERR#			检测到数据校验错
	SERR#			检测到地址校验错或系统错
仲裁	REQ#			申请总线使用权
	GNT#			得到总线使用权
系统	CLK			同步时钟(33 MHz 或 66 MHz),上升沿采样
	RST#			复位信号,回到初始状态

表 7.2 中,“主设备”和“从设备”两列分别指出在总线传输中,由哪个设备发出和接收这些信号。若该信号由其他设备发出,则两列均为空。

图 7.30 右排表示的信号为 PCI 的扩展信号,这些信号的说明列于表 7.3。

表 7.3 PCI 扩展信号

功能组	信号	主设备	从设备	描述
64 位扩展	AD[63:32]	√		复用的地址和数据信号
	C/BE#[7:4]	√		复用的指示总线命令和指示字节有效信号
	REQ64#	√		用于请求 64 位传输
	ACK64#		√	授权进行 64 位传输
	PAR64	√		附加的 32 位地址或数据校验位
接口控制	LOCK#	√		锁定总线
中断	INTA#			中断请求
	INTB#			中断请求,仅对多功能设备有意义
	INTC#			中断请求,仅对多功能设备有意义
	INTD#			中断请求,仅对多功能设备有意义

(续表)

功能组	信号	主设备	从设备	描述
Cache 支持	SBO#			探测取消
	SDONE			探测完成
边界扫描	TCK			JTAG 测试时钟
	TDI			JTAG 测试输入
	TDO			JTAG 测试输出
	TMS			JTAG 测试模式选择
	TRST#			JTAG 测试复位

PCI 还有一些其他的信号线用于电源线、地线等,本书就不一一列出了。

7.5.3 PCI 总线命令

设备通过 PCI 总线能够进行的操作与 PCI 总线命令有关。当一个总线主设备获得了总线的控制权,它就决定了进行交换的类型。通过 C/BE#[3:0]信号指出 PCI 总线的命令,这些命令列于表 7.4。

表 7.4 PCI 总线命令

C/BE#[3:0]	命令名称	命令说明
0000	中断应答	确认响应中断,使中断控制器工作
0001	专用周期	主设备将消息广播给一个或多个从设备
0010	I/O 读	I/O 设备读操作
0011	I/O 写	I/O 设备写操作
0100	保留	
0101	保留	
0110	存储器读	存储器设备读操作
0111	存储器写	存储器设备写操作
1000	保留	
1001	保留	
1010	配置读	读设备的配置空间
1011	配置写	写设备的配置空间
1100	存储器行读	高速缓存存储器行读(成组传输,可读取 Cache 一行中的多个双字)
1101	双地址周期	指明使用 64 位存储器地址
1110	存储器多行读	高速缓存存储器多行读(成组传输,可读取不同 Cache 行的多个双字)
1111	存储器写和失效	存储器写并且进行 Cache 数据一致性校验

关于这些命令的详细定义,感兴趣的读者可参阅相关参考文献。

7.5.4 PCI 总线仲裁

PCI 总线主设备在进行数据传送之前,必须先向总线仲裁电路发出使用总线的请求。PCI 总线采用集中式仲裁方式。每个 PCI 总线主设备都有独立的总线请求(REQ#)和总线授权(GNT#)两条信号线与仲裁电路连接,如图 7.31 所示。

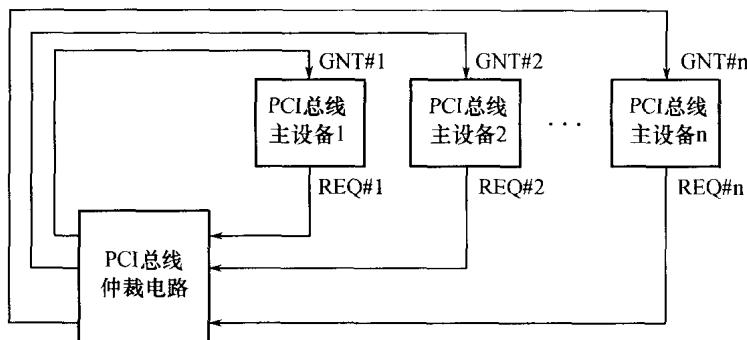


图 7.31 PCI 总线使用集中式仲裁方式

接到总线主设备的请求后,仲裁电路根据一定的算法对主设备的申请进行仲裁,决定总线使用权的归属。但是 PCI 总线标准并没有规定仲裁算法,仲裁电路可使用先来先服务的仲裁算法,也可使用轮转法或其他的优先级调度算法,具体算法完全由设计和生产总线控制器的厂商决定。对两个 PCI 总线主设备——主设备 1 和主设备 2 进行数据发送的仲裁过程如图 7.32 所示。

图 7.32 所示的仲裁过程开始前,主设备 1 首先将自己的请求信号 REQ#1 降低,表示发出总线请求开始一次总线仲裁。

在时钟周期 T1 内,总线仲裁电路在时钟上升沿采样到 REQ#1 这个请求信号。总线仲裁电路将 GNT#1 信号变成低电平表示允许主设备 1 对总线的使用。同时主设备 2 的 REQ#2 信号变成低电平表明它也提出了使用总线的请求。

在时钟周期 T2 内,主设备 1 在时钟的上升沿采样到 GNT#1 信号有效,知道自己的请求得到满足。同时主设备 1 发现 IRDY#信号和 TRDY#信号都为高电平表示总线空闲,从而它将 FRAME#信号变为低电平表明开始一次总线传输。主设备 1 将地址信息放在 AD 线上,将命令放在 C/BE#线上。保持 REQ#1 信号有效,说明它随后又发出了一次进行数据发送的总线请求。

在时钟周期 T3 内,总线仲裁电路在时钟的上升沿采样所有的 REQ#信号和 GNT#信号,并且决定将总线裁决给主设备 2 使用,于是降低 GNT#2 信号的电平(有效),升高 GNT#1 信号的电平(无效),此时由于主设备 1 还在传送数据(一次数据传送没有完成),总线没有空闲,因此主设备 2 不能立即使用总线。主设备 1 升高 FRAME#信号,表示最后一个数据正在

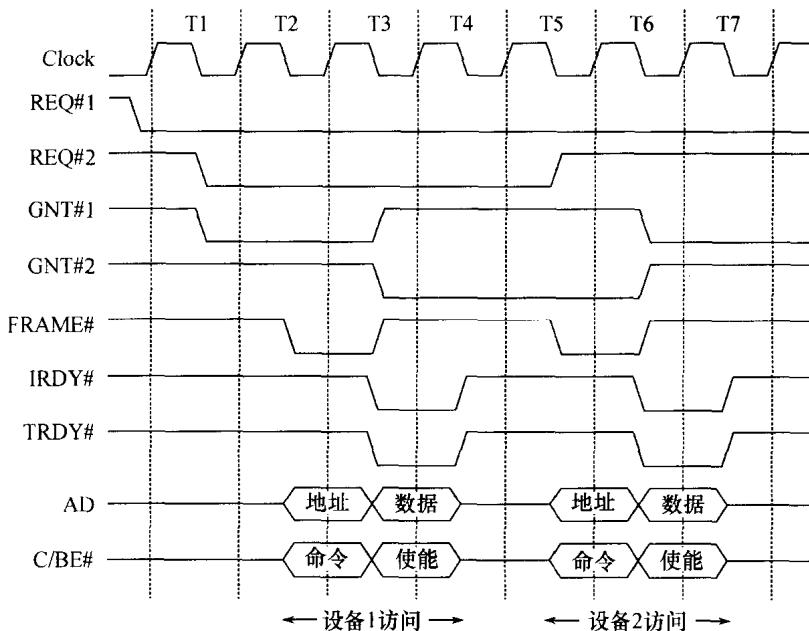


图 7.32 PCI 总线的仲裁过程

传输。IRDY#信号和 TRDY#信号均为低电平表明主设备 1 和对应的从设备均就绪。

需要注意的是，总线仲裁电路对主设备 2 的请求进行仲裁与获得总线使用权的主设备 1 进行数据传输是同时进行的，因而总线仲裁电路对主设备 2 的仲裁并不需要额外的时钟周期，这有利于提高总线的使用效率，这种仲裁方式被称为隐藏式仲裁（Hidden Arbitration）。

在时钟周期 T4 内，数据已经出现在 AD 上，从设备采样数据后传输完成。IRDY#信号和 TRDY#信号均变成高电平表明总线空闲。

在时钟周期 T5 内，主设备 2 在时钟上升沿采样 IRDY#信号和 TRDY#信号发现总线已经空闲，并且自己已经获得总线使用权（GNT#2 信号有效），于是它降低 FRAME#信号开始使用总线。同时主设备 2 撤销自己的总线使用请求 REQ#2 信号，表明它只需要进行一次数据发送。这样，主设备 1 将在主设备 2 使用完总线后获得总线的使用权。

在时钟周期 T6 中的操作与在时钟周期 T3 中的操作类似，在时钟周期 T7 中的操作与在时钟周期 T4 中的操作类似，只是进行数据传输的设备不同。

为了防止总线主设备不合理地占用总线，PCI 总线仲裁电路还需要对获得总线使用权的主设备进行控制。一个提出申请并被授权的总线主设备，应在总线空闲后尽快开始新的总线操作。如果获得总线使用权的总线主设备在 FRAME#信号和 IRDY#信号变为高电平后，连续 16 个时钟周期没有使用总线（总线空闲），总线仲裁电路将认为这个主设备发生故障，收回总线使用权并且不再授权给这个主设备。

7.5.5 PCI 总线数据传送方式

PCI 总线支持基本数据传送方式和成组数据传送方式。通过 PCI 总线进行基本数据传输的时序图如图 7.33 所示。

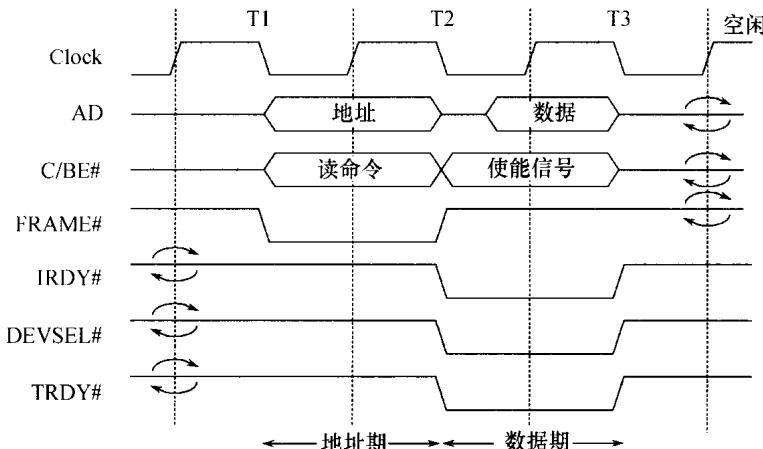


图 7.33 PCI 总线基本数据传送方式

图 7.33 给出了一个总线主设备进行读操作的时序, 读操作周期由三个时钟周期(T1~T3)组成。图中的环形箭头符号表示某信号线由一个设备驱动转换成另一个设备驱动的过渡期, 以此过渡期来避免两个设备同时驱动一条信号线发生冲突。当检测到总线空闲(IRDY#信号和 TRDY#信号都为高电平)时, 通过仲裁, 总线主设备获得总线使用权, 开始一次总线读操作周期。

在时钟周期 T1 内, 当出现时钟下降沿时, 主设备将地址送到 AD 线上, 并将总线命令送到 C/BE#线上, 然后发出 FRAME#信号来启动总线操作。

在时钟周期 T2 内, 主设备停止对 AD 线的驱动, 使从从设备读出的数据可以放在 AD 线上。主设备修改 C/BE#信号指出需要读入双字的哪个字节。主设备发出 IRDY#信号表明已经准备好接收第一个数据。同时, 总线上所有的从设备在时钟周期 T2 的上升沿采样 AD 线、C/BE#线和 FRAME#信号, 并开始译码以选定从设备。一旦译码确定某个从设备, 该从设备将发出 DEVSEL#信号表明从设备已经选中并准备响应主设备的请求。从设备一旦准备好数据就把数据送到 AD 线上, 并发出 TRDY#信号表明从设备完成操作。一旦数据准备传输, FRAME#信号将变为高电平。

在时钟周期 T3 内, 主设备在时钟周期的上升沿采样到 IRDY#信号和 TRDY#信号有效, 主设备从 AD 上采样数据。基本数据传送方式的一次数据读过程结束。DEVSEL#信号、IRDY#信号和 TRDY#信号均变成高电平。

如果在时钟周期 T3 开始时 DEVSEL#信号无效, 表明从设备尚未准备好传输数据, 这样

就需要在 T2 和 T3 间延迟一个或几个周期(等待状态)。在等待状态中,C/BE#信号和 IRDY#信号保持原状态。在时钟周期 T3 结束时,如果没有新的总线操作,总线将进入空闲状态。

从图 7.33 中可以看出,一次 PCI 总线操作实际被分成了地址期和数据期两部分。在地址期中,给出地址 AD 和发出命令 C/BE#需要一个时钟周期(从 T1 的下降沿到 T2 的下降沿)。在数据期中,如果不考虑等待状态,AD 线上的数据也需要稳定一个周期(从 T2 的下降沿到 T3 的下降沿)。这表明采用基本数据传送方式时,最快情况下 PCI 总线平均每两个时钟周期传送一个双字(4B)。如果 PCI 总线的工作频率为 33 MHz,此时 PCI 总线的最大数据传输率为: $33 \text{ MHz} \times 4 \text{ B/s} = 66 \text{ MB/s}$ 。

下面以一次读操作为例来说明通过 PCI 总线进行成组数据传输的过程。假定成组数据传输一次需要读出三个数据,其读时序如图 7.34 所示。

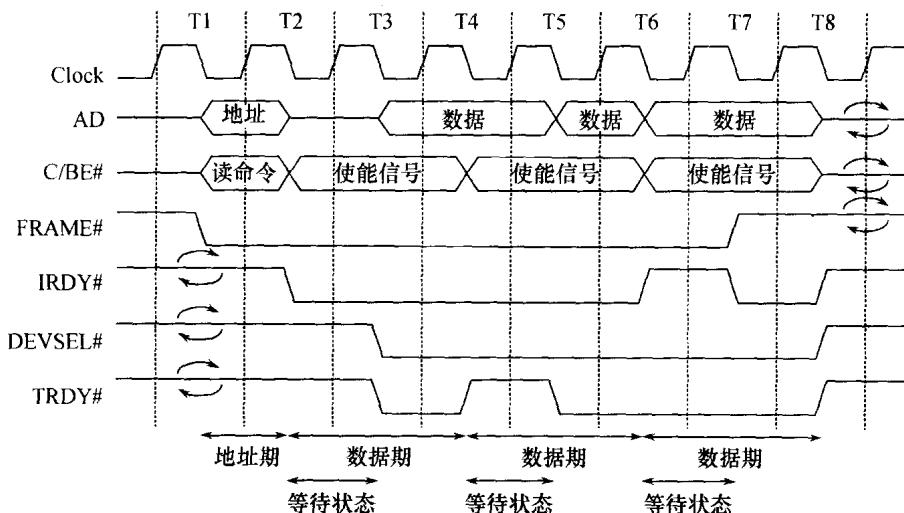


图 7.34 PCI 总线成组数据传送方式

一旦某个总线主设备获得了总线使用权,将开始一个总线读操作周期。

在时钟周期 T1 内,主设备启动 FRAME#信号表明一个新的总线操作开始。FRAME#信号将一直保持到主设备接收最后一个数据时止。这表明成组数据传输是通过 FRAME#信号来控制结束的。同时,主设备将成组数据的起始地址放在 AD 线上,将读命令送 C/BE#线。

在时钟周期 T2 内,从设备根据 AD 线和 C/BE#线上的信号识别出自己的地址。主设备停止对 AD 线的驱动,地址信号终止。改变 C/BE#线上的信号来表明 AD 线上的哪些字节是有效数据。使 IRDY#信号降低,由于是读操作,表示主设备已准备好接收第一个数据。由于此时 DEVSEL#信号无效,表明选中的从设备还不能提供数据,需要插入一个等待状态。

在时钟周期 T3 内,从设备降低 DEVSEL#信号表明已识别出自己的地址和命令,并准备应答。它将请求的数据放在 AD 线上,并降低 TRDY#信号表明数据已经有效。

在时钟周期 T4 内,由于 IRDY#信号和 TRDY#信号都有效,主设备从 AD 线上读第一个数据。同时改变 C/BE#线的内容,为读出下一个数据做准备。

在时钟周期 T5 内,插入一个等待状态。由于从设备的某种原因(例如访问速度较慢),它尚未准备好第二个数据。从设备通过升高 TRDY#信号告诉主设备在接下来的周期里 AD 线上没有有效的数据。于是主设备不从 AD 线上读取数据,并且在此时钟周期内保持 IRDY #信号以及 C/BE#信号的内容。

在时钟周期 T6 内,由于 IRDY#信号和 TRDY#信号都有效,主设备从 AD 线上读第二个数据。同时改变 C/BE#线的内容,为读出下一个数据做准备。

在时钟周期 T7 内,插入一个等待状态。虽然从设备已经将数据放在 AD 线上,但是主设备由于某种原因(例如缓冲区满)并没有做好读取这个数据的准备,因此升高 IRDY#信号,导致从设备将总线上的数据再保持一个时钟周期。主设备在第三次数据开始传输时,升高 FRAME#信号,告诉从设备和总线控制器这是最后一个数据传输。

在时钟周期 T8 内,由于 IRDY#信号和 TRDY#信号都有效,主设备从 AD 线上读第三个数据。成组数据传送方式的一次数据读过程结束。DEVSEL#信号、IRDY#信号和 TRDY#信号均变成高电平。如果没有新的总线操作,总线将进入空闲状态。

从图 7.34 中可以看出,三个数据的读出只需要给出起始数据的地址。如果不考虑等待状态,第一个数据的读出需要一个地址期和一个数据期,而其后每个数据的读出都只需要一个数据期。如果有大量的数据进行成组数据传输,依 7.4.5 节可知,第一个数据传输的地址期对整个数据传输时间的影响可以忽略。这表明采用成组数据传送方式时,PCI 总线平均每个时钟周期传送一个双字(4 B)。如果 PCI 总线的工作频率为 33 MHz,此时 PCI 总线的最大数据传输率为:33 MHz × 4 B = 132 MB/s。

以上从总线信号、总线命令、总线仲裁和总线的数据传送方式等方面介绍了 PCI 总线标准,关于 PCI 总线的详细内容和具体使用可以参见相关的技术资料。

小 结

计算机的硬件系统由 CPU、存储器和 I/O 设备组成,这些部件为了协同工作,必须进行信息交换。计算机互连结构的作用就是完成部件的连接以及通信的。总线是常见的互连结构之一。

总线系统由总线、总线设备、总线设备接口和总线控制器四部分构成。总线的基本组成元件是集电极开路门或者三态门。总线设备的多样性决定了总线的多样性,但是总线的本质都是一样的,即总线是多个系统功能部件之间进行信息传送的公共通路。总线设备通过总线接口可以很方便地连接到总线上,总线的使用由总线控制器进行管理。

总线的分类标准有很多,许多标准都涉及到总线的设计问题。在总线设计时,要根据总

线的功能、总线宽度、总线复用方式、总线定时方式、总线仲裁机制以及总线数据传送方式等方面进行综合考虑。无论选择哪种方式,设计者的目标都是一样的,那就是尽可能地获得较高的总线数据传输率。

通过总线将各个设备连接成一个系统所必须遵循的规范就是总线标准,总线的名称通常以该种总线的标准命名。本章介绍了一种总线标准及实例——PCI 总线。

习 题

7.1 解释下列术语:

互连结构	总线	总线带宽	总线宽度
总线主设备	总线从设备	总线源设备	总线目标设备
总线标准	隐蔽式仲裁		

7.2 从规模上看,计算机的总线可以分为哪几种?

7.3 为什么说总线宽度是连接到总线上的设备可能获得的最大性能的决定因素之一?为什么说总线宽度又是影响系统性能的关键因素之一?

7.4 什么是控制总线?控制总线在总线通路中有何作用?

7.5 什么是总线设备?它有何特点?试举例说明。

7.6 单总线结构存在什么缺点?存在的主要问题有哪些?解决此问题的途径是什么?

7.7 什么是总线控制器?它的功能是什么?

7.8 何谓分时多路复用?以常见的读写复用为例,说明复用策略下信息传送的过程。

7.9 试用表格分析比较同步总线和异步总线的优缺点。

7.10 以 I/O 设备访问操作为例。说明同步总线操作过程,并画出其时序图。

7.11 何谓集中式仲裁和分布式仲裁?二者各有何优缺点?

7.12 什么是固定优先权和动态优先权算法?它们各有什么特点?动态优先权算法一般有哪些?

7.13 什么是查询式优先权仲裁?它和一般的仲裁方式有何不同?它具有什么特点?

7.14 什么是并行仲裁和串行仲裁?它们各有什么特点?

7.15 什么是成组数据传送?与基本的数据传送方式相比,它有何特点?成组数据传送时,怎样控制数据传送结束?

7.16 什么是“读后写”和“写后读”操作?它们各有何特点?

参 考 文 献

- 1 宋焕章等. 计算机原理与设计. 长沙:国防科技大学出版社,1999
- 2 李勇等. 计算机原理与设计. 长沙:国防科技大学出版社,1989
- 3 王爱英. 计算机组装与结构. 第三版. 北京:清华大学出版社,2001
- 4 唐朔飞. 计算机组装原理. 北京:高等教育出版社,2000
- 5 潘新民等. 微型计算机原理汇编接口技术. 北京:北京希望电子出版社,2002
- 6 Hennessy J L, Patterson D A, Computer Organization & Design The Hardware/Software Interface. (影印版), 机械工业出版社, 1999
- 7 William S. Computer Organization and Architecture;Design for Performance. 4th ed. Prentice Hall, Inc. 1996
- 8 Hennessy J L, Patterson D A. Computer Organization and Design, 2th ed. Morgan Kanfman Publishers, Inc. 1998
- 9 李华贵等. 高档微机维修技术. 成都:电子科技大学出版社,1997
- 10 慈云桂,杨晓东. 一种新型无冲突访问存储系统分析. 中国科学, 1984
- 11 Tom Shanley,Don Anderson 著,刘晖等译. PCI 系统结构(第 4 版). 北京:电子工业出版社,2000