计算机组成与系统结构

陈泽宇 主编

上海交通大学

目　　录

第1章 计算机系统概论

1.1 计算机的分类、发展与应用

电子数字计算机（Electronic Digital Computer），通常简称为计算机（Computer），是按照一系列指令来对数据进行处理的机器，是一种能够接收信息，存储信息，并按照存储在其内部的程序对输入的信息进行加工、处理，得到人们所期望的结果，并把处理结果输出的高度自动化的电子设备。

计算机的发明和发展是20世纪人类最伟大的科学技术成就之一，也是现代科学技术发展水平的重要标志。

计算机拥有众多的物理形态。早期的计算机足有一间房间大小，而如今的计算机可以小到装入手表，用手表电池驱动。个人计算机（Personal Computer，PC）和便携计算机（Portable Computer，又称膝上型计算机Laptop Computer）已经成为信息时代的标志，它们是大多数人所认为的“计算机”。但是，到目前为止，使用最为广泛的计算机形态却是嵌入式计算机（Embedded Computer）。嵌入式计算机较为小型、简单通常用来控制其他设备，可以出现在各种机器中，从战斗机到工业机器人，从数码相机到儿童玩具。

根据Church–Turing理论，任何一台具有最基本功能的计算机，原则上都能够执行任何其他计算机可以执行的任务。因此，只要不考虑时间和存储容量，性能和复杂度均相差甚远的各种计算机，从个人数字助理（Personal Digital Assistant，PDA）到超级计算机（Supercomputer），都能够执行相同的运算任务。

图1-1为美国SGI公司（Silicon Graphics, Inc.）为NASA（National Aeronautics and Space Administration，美国国家航空和宇宙航行局）制造的Columbia超级计算机。图1-2为在GNUX（GNU+Linux）操作系统下运行视频会议软件的手表计算机。

图1-1 NASA Columbia超级计算机

图 1-2 手表计算机

1.1.1 计算机的分类

1．计算机分类

根据计算机的效率、速度、价格、运行的经济性和适应性来划分，计算机可分为通用计算机和专用计算机两大类。

通用计算机功能齐全，通用性强，适应面广，可完成各种各样的工作，但是牺牲了效率、速度和经济性。

专用计算机是专为某些特定问题而设计的功能单一的计算机，一般说来其结构要比通用计算机来得简单，具有可靠性高、速度快、成本低的优点，是最有效、最经济和最快速的计算机，但是其适应性很差。

2．通用计算机分类

通用计算机又可分为超级计算机（Supercomputer）、大型机（Mainframe）、服务器（Server）、工作站（Workstation）、微型机（Microcomputer）和单片机（Single-Chip Computer）6 类，它们的区别在于体积、复杂度、功耗、性能指标、数据存储容量、指令系统规模和价格，如图 1-3 所示。

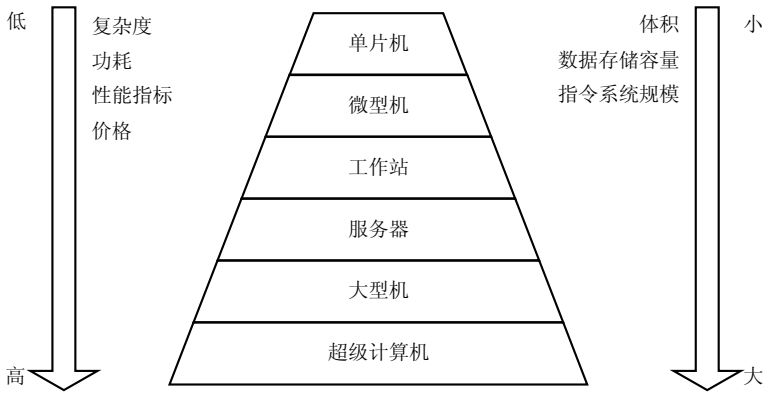


图 1-3 通用计算机的分类

一般而言，超级计算机主要用于科学计算，其运算速度远远超过其他计算机，数据存储容量很大，结构复杂，价格昂贵。单片机是只用单片集成电路（Integrated Circuit，IC）做成的计算机，体积小，结构简单，性能指标较低，价格便宜。介于超级计算机和单片机之间的是大型机、服务器、工作站和微型机，它们的结构规模和性能指标依次递减。但是，随着超大规模集成电路的迅速发展，微型机、工作站、服务器彼此之间的界限也在发生变化，今天的工作站有可能是明天的微型机，而今天的微型机也可能是明天的单片机。

1.1.2 计算机的发展概况

最初，“Computer”一词指的是从事数值运算的人，他们往往借助于某种机械运算装置来完成数值运算工作。随着时代的演变和技术的进步，“Computer”一词现在专指计算机，即电子数字计算机。

1．第一台通用电子数字计算机

一般认为，世界上第一台通用电子数字计算机是 1946 年在美国宾夕法尼亚大学问世的 ENIAC（Electronic Numerical Integrator And Computer，电子数字积分计算机），如图 1-4 所示。这台机器用了 18000 多个电子管，占地 170 平方米，总重量达 30 吨，耗电 140 千瓦，每秒能做 5000 次加减运算。从今天的眼光来看，这台计算机耗费巨大又不完善，但它却是科学史上一次划时代的创新，奠定了现代电子数字计算机的基础。

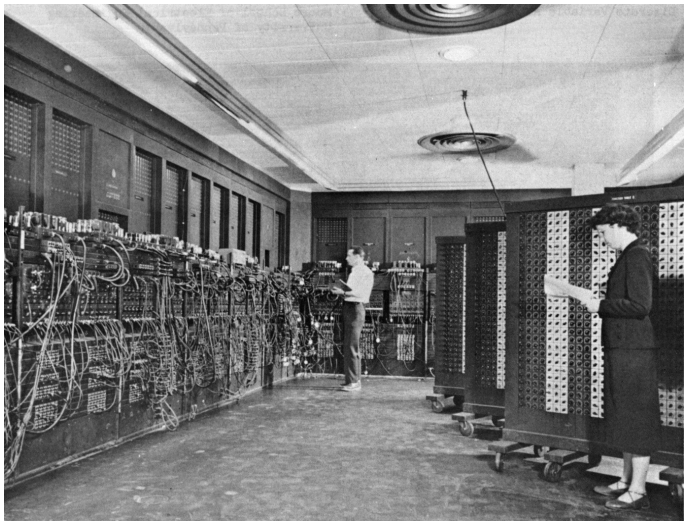


图 1-4 世界上第一台通用电子数字计算机 ENIAC

最初，ENIAC 的结构设计不够灵活，每一次重新编程都必须重新连线（Rewiring）。此后，ENIAC 的开发人员认识到这一缺陷，提出了一种灵活、合理得多的设计，这就是著名的存储程序体系结构（ Stored-Program Architecture）。在存储程序体系结构中，给计算机一个指令序列（即程序），计算机会存储它们，并在未来的某个时间里，从计算机存储器中读出，依照程序给定的顺序执行它们。现代计算机区别于其他机器的主要特征，就在于这种可编程能力。

由于早在 ENIAC 完成之前，数学家约翰·冯·诺伊曼（John von Neumann）就在其论文中提出了存储程序计算机的设计思想，因此，存储程序体系结构又称为冯·诺伊曼体系结构（von Neumann Architecture）。自从 20 世纪 40 年代第一台通用电子数字计算机出现以来，尽管计算机技术已经发生了翻天覆地的变化，但是，大多数当代计算机仍然采用冯·诺伊曼体系结构。

2．数字计算机的发展史

自从 ENIAC 计算机问世以来，从使用器件的角度来说，计算机的发展大致经历了 5 代的变化（如表 1-1 所示）：

表 1-1 数字计算机的发展史

	时间	使用器件	执行速度（次/秒）	典型应用
第 1 代	1946~1957	电子管	几千至几万	数据处理机
第 2 代	1958~1964	晶体管	几万至几十万	工业控制机
第 3 代	1965~1970	小规模/中规模集成电路	几十万至几百万	小型计算机
第 4 代	1971~1985	大规模/超大规模集成电路	几百万至几千万	微型计算机
第 5 代	1986~	甚大规模集成电路	几亿至上百亿	单片计算机

第一代计算机从 1946 年到 1957 年，使用电子管（Vacuum Tube）作为电子器件，使用机器语言与符号语言编制程序。计算机运算速度只有每秒几千次至几万次，体积庞大，存储容量小，成本很高，可靠性较低，主要用于科学计算。在此期间，形成了计算机的基本体系结构，确定了程序设计的基本方法，“数

据处理机”开始得到应用。

第二代计算机从1958年到1964年，使用晶体管（Transistor）作为电子器件，开始使用计算机高级语言。计算机运算速度提高到每秒几万次至几十万次，体积缩小，存储容量扩大，成本降低，可靠性提高，不仅用于科学计算，还用于数据处理和事务处理，并逐渐用于工业控制。在此期间，“工业控制机”开始得到应用。

第三代计算机从1965年到1970年，使用小规模集成电路（Small-Scale Integration，SSI）与中规模集成电路（Medium-Scale Integration，MSI）作为电子器件，而操作系统的出现使计算机的功能越来越强，应用范围越来越广。计算机运算速度进一步提高到每秒几十万次至几百万次，体积进一步减小，成本进一步下降，可靠性进一步提高，为计算机的小型化、微型化提供了良好的条件。在此期间，计算机不仅用于科学计算，还用于文字处理、企业管理和自动控制等领域，出现了管理信息系统（Management Information System，MIS），形成了机种多样化、生产系列化、使用系统化的特点，“小型计算机”开始出现。

第四代计算机从1971年到1985年，使用大规模集成电路（Large-Scale Integration，LSI）与超大规模集成电路（Very-Large-Scale Integration，VLSI）作为电子器件。计算机运算速度大大提高，达到每秒几百万次至几千万次，体积大大缩小，成本大大降低，可靠性大大提高。在此期间，计算机在办公自动化、数据库管理、图像识别、语音识别和专家系统等众多领域大显身手，由几片大规模集成电路组成的“微型计算机”开始出现，并进入家庭。

第五代计算机从1986年开始，采用甚大规模集成电路（Ultra-Large-Scale Integration，ULSI）作为电子器件，运算速度高达每秒几亿次至上百亿次。由一片甚大规模集成电路实现的“单片计算机”开始出现。

3．计算机体系结构的发展过程

生产、科研、应用的飞速发展，促使计算机的体系结构不断完善，形成了当代计算机的体系结构形式。

计算机问世后的60多年来，计算机体系结构的发展过程一直是在冯·诺伊曼体系结构的基础上，以提高速度、扩大存储容量、降低成本、提高系统可靠性、方便用户使用为目的，不断采用新的器件、研制新的软件的过程。就体系结构本身来说，主要是指令系统、微程序设计、流水线结构、多级存储器体系结构、输入/输出体系结构、并行体系结构、分布式体系结构、多媒体体系结构、操作系统和数据库管理系统的形成和发展。

1.1.3 计算机的应用

计算机之所以得到迅速发展，其主要原因在于它的广泛应用。计算机的应用范围几乎涉及人类社会的所有领域：从国民经济各部门到个人家庭生活，从军事部门到民用部门，从科学教育到文化艺术，从生产领域到消费娱乐，无一不是计算机应用的天下。

1．科学计算

科学研究和工程技术计算领域，是计算机应用最早的领域，也是应用较为广泛的领域，包括数学、化学、原子能、天文学、地球物理学、生物学等基础科学研究，以及航天飞行、飞机设计、桥梁设计、水力发电、地质找矿、天气预报等方面的大量计算。利用计算机进行数值计算，可以节省大量的时间、人力和物力。

2．自动控制

自动控制是涉及面极广的一门学科，应用于工业、农业、科学技术、国防以至我们日常生活的方方面面。有了体积小、价廉、可靠的微型机和单片机作为工具，自动控制进入了以计算机为主要控制设备的新的发展阶段。

3．测量测试

计算机在测量和测试领域中所占的比例也相当大，约占20%左右。在这个领域中，计算机主要起两个作用：第一，对测量和测试设备本身进行控制；第二，采集数据并进行处理。

4．信息处理

信息，是人类赖以生存和交际的媒介。通过五官和皮肤，人类可以看到文字图像，听到唱歌说话，闻到香臭气味，尝到酸甜苦辣，感到冷热变化，这些都是信息。人本身就是一个非常高级的信息处理系统。

计算机在发展初期仅仅用于数值计算。此后，计算机的应用范围逐渐发展到非数值计算领域，可用来处理文字、表格、图像、声音等各类信息。因此，确切地讲，一台计算机实际上就是一台信息处理机。

5．教育卫生

创立学校、应用书面语言、发明印刷术，被称为教育史上的三次革命。目前，计算机广泛应用于教育，被誉为“教育史上的第四次革命”。除了目前应用较为普遍的“计算机辅助教学（Computer-Aided Instruction，CAI）”之外，基于网络的现代远程教学（Distance Learning，或 e-Learning）也是近几年迅速发展起来的一种典型的教育应用。

计算机的问世，同样为人类健康长寿带来了福音。一方面，使用计算机的各种医疗设备应运而生，CT 图像处理设备、心脑电图分析仪、血液分析仪等先进的设备和仪器为及早发现疾病提供了强有力的手段。另一方面，利用计算机建成了集专家经验之大成的各种各样的专家系统，如中医专家诊疗系统、各种疾病的电子诊疗系统等等，事实表明，这些专家系统行之有效，为诊治疾病发挥了很大作用。

6．电子电器

目前，不仅各种类型的个人计算机早已进入家庭，而且在微波炉、洗衣机、家用空调、DVD 播放机、电子玩具、游戏机等电子电器产品中也广泛应用了各种嵌入式计算机。

除了可单独使用的独立电器之外，许多家用电器还可以通过各种有线或无线的网络连接（如 Internet、红外线、蓝牙等），完成自身程序的自动更新、远程控制等复杂任务。

7．人工智能

人工智能，简而言之就是使计算机模仿人的高级思维活动，像人类那样直接利用各种自然形式的信息，如文字、图像、颜色、自然景物、声音语言等。目前，在文字识别、图形识别、景物分析、语音识别、语音合成以及语言理解等方面，人工智能已经取得了不少的成就。

到目前为止，人工智能研究中最突出的成就非“机器人”莫属。世界上有大量的“工业机器人”在各种生产线上完成简单重复的工作，或是代替人类在高温、有毒、辐射、深水等恶劣环境下工作。现在，又出现了更为先进的“智能机器人”，它会自己识别控制对象和工作环境，自动作出判断和决策，直接领会人的命令和意图，避开障碍物，适应环境变化，灵活机动地完成指定的控制任务与信息处理任务。图 1-6 为汽车生产中的工业机器人。

图 1-6 汽车生产中的工业机器人

1.2 计算机的基本组成

一台完整的计算机由计算机硬件（Hardware）和计算机软件（Software）两部分组成，其中，硬件是基础，是软件活动的舞台；软件是灵魂，使硬件最大限度地发挥作用，两者缺一不可。

计算机硬件是由物理元器件构成的有形实体，主要是数字逻辑电路。

计算机软件则是由计算机程序构成的无形的东西，需要存储在有形的硬件（如主存储器、硬盘等）中，可以实现更高层次的逻辑功能。

1.2.1 计算机硬件

计算机硬件是组成计算机的所有电子器件和机电装置的总称，是构成计算机的物质基础，是计算机系统的核心。

目前大多数计算机都是根据冯·诺伊曼体系结构的思想来设计的，其主要特点是使用二进制数和存储程序，其基本思想是：事先设计好用于描述计算机工作过程的程序，并与数据一样采用二进制形式存储在机器中，计算机在工作时自动、高速地从机器中按顺序逐条取出程序指令加以执行。简而言之，冯·诺伊曼体系结构计算机的设计思想就是存储程序并按地址顺序执行。

在计算机存储器里把程序及其操作数据一同存储的思想，是冯·诺伊曼体系结构（或称存储程序体系结构）的关键所在。在某些情况下，计算机也可以把程序存储在与其操作数据分开的存储器中，这被称为哈佛体系结构（Harvard Architecture），源自 Harvard Mark I 计算机。现代的冯·诺伊曼计算机在设计中展示出了某些哈佛体系结构的特性，如高速缓存 Cache。

冯·诺伊曼体系结构的计算机具有共同的基本配置，即具有 5 大部件：控制器、运算器、存储器、输入设备和输出设备，这些部件用总线相互连接，如图 1-7 所示。

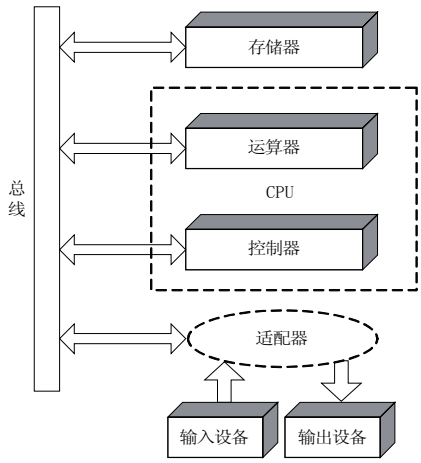


图 1-7 计算机的硬件组成

其中，控制器和运算器合称为中央处理器（Central Processing Unit，CPU）。早期的 CPU 由许多分立元件组成，但是自从 20 世纪 70 年代中期以来，CPU 通常被制作在单片集成电路上，称为微处理器（Microprocessor）。CPU 和存储器通常组装在一个机箱内，合称为主机。除去主机以外的硬件装置称为外围设备。

计算机系统工作时，输入设备将程序与数据存入存储器，运行时，控制器从存储器中逐条取出指令，将其解释成控制命令，去控制各部件的动作。数据在运算器中加工处理，处理后的结果通过输出设备输出。

1．控制器

控制器是计算机的管理机构和指挥中心，它按照预先确定的操作步骤，协调控制计算机各部件有条不紊地自动工作。

控制器工作的实质就是解释程序，它每次从存储器读取一条指令，经过分析译码，产生一系列操纵计算机其他部分工作的控制信号（操作命令），发向各个部件，控制各部件动作，使整个机器连续、有条不紊地运行。高级计算机中的控制器可以改变某些指令的顺序，以改善性能。

对所有 CPU 而言，一个共同的关键部件是程序计数器（Program Counter），它是一个特殊的寄存器，记录着将要读取的下一条指令在存储器中的位置。

2．运算器

运算器是一个用于信息加工的部件，用于对数据进行算术运算和逻辑运算。

运算器通常由算术逻辑单元（Arithmetic Logic Unit，ALU）和一系列寄存器组成。其中，ALU 是具体完成算术与逻辑运算的单元，是运算器的核心，由加法器和其他逻辑运算单元组成。寄存器用于存放参与运算的操作数。累加器是一个特殊的寄存器，除了存放操作数之外，还用于存放中间结果和最后结果。

特定 ALU 所支持的算术运算，可能仅局限于加法和减法，也可能包括乘法、除法，甚至三角函数和平方根。有些 ALU 只支持整数，而其他 ALU 则可以使用浮点来表示有限精度的实数。但是，能够执行最简单运算的任何计算机，都可以通过编程，把复杂的运算分解成它可以执行的简单步骤。所以，任何计算机都可以通过编程来执行任何的算术运算，如果其 ALU 不能从硬件上直接支持，则该运算将用软件方式实现，但需要花费较多的时间。

超标量（Superscalar）计算机包含多个 ALU，可以同时处理多条指令。图形处理器和具有单指令流多数据流 SIMD 和多指令流多数据流 MIMD 特性的计算机，通常提供可以执行矢量和矩阵算术运算的 ALU。

3．存储器

存储器的主要功能是存放程序和数据。程序是计算机操作的依据，数据是计算机操作的对象。不管是程序还是数据，在存储器中都是用二进制数的形式来表示的，统称为信息。向存储器存入或从存储器取出信息，都称为访问存储器。

计算机存储器是由可以存放和读取数值的一系列单元所组成的，每个存储单元都有一个编号，称为“地址”。向存储器中存数或者从存储器中取数，都要按给定的地址来寻找所选择的存储单元。存放在存储器中的信息可以表示任何东西，文字、数值甚至计算机指令都可以同样容易地存放到存储器中去。

由于计算机仅使用 0 和 1 两个二进制数字，所以使用位（bit，简写成 b）作为数字计算机的最小信息单位，包含 1 位二进制信息（0 或 1）。当 CPU 向存储器送入或从存储器取出信息时，不能存取单个的位，而是使用字节、字等较大的信息单位。一个字节（Byte，简写成 B）由 8 位二进制信息组成，而一个字（Word）则表示计算机一次所能处理的一组二进制数，它由一个以上的字节所组成。通常把组成一个字的二进制位数称为字长，例如微型机的字长可以少至 8 位，多至 32 位，甚至达到 64 位。

存储器中所有存储单元的总数，称为存储器的存储容量，通常用单位 KB（Kilobyte，千字节）、MB（Megabyte，兆字节）、GB（Gigabyte，千兆字节）表示，如 64KB、128MB、256GB。度量存储器容量的各级单位之间的关系为：1KB=1024B，1MB=1024KB，1GB=1024MB。存储容量越大，计算机所能存储记忆的信息就越多。

存储器是计算机中存储信息的部件，按照存储器在计算机中的作用，可分为主存储器、寄存器、闪速存储器、高速缓冲存储器、辅助存储器等几种类型，它们均可完成数据的存取工作，但性能及其在计算机中的作用差别很大。

4．输入输出设备

计算机的输入输出（I/O）设备是计算机从外部世界接收信息并反馈结果的手段，统称为 I/O 设备或外围设备（Peripheral，简称外设）。各种人机交互操作、程序和数据的输入、计算结果或中间结果的输出、被控对象的检测和控制等，都必须通过外围设备才能实现。

在一台典型的个人计算机上，外围设备包括键盘和鼠标等输入设备，以及显示器和打印机等输出设备。

5．总线

除了上述 5 大部件外，计算机系统中还必须有总线（Bus）。计算机系统通过总线将 CPU、主存储器及 I/O 设备连接起来。

总线是构成计算机系统的骨架，是多个系统部件之间进行数据传送的公共通路。借助于总线连接，计算机在各部件之间实现传送地址、数据和控制信息的操作。

按照信号类型，可将总线分为数据总线、地址总线和控制总线。其中，数据总线主要传送数据，是双向的，既可以输入，又可以输出；地址总线传送地址信息，是单向的，决定数据或命令传送给谁；而控制总线则传送各种控制信号。

1.2.2 计算机软件

计算机软件是程序的有序集合，而程序则是指令的有序集合。

在大多数计算机中，每一条指令都被分配了一个惟一的编号（称为操作码），以机器指令代码的形式存储。

因为计算机存储器能够存储数字，所以它也能存储指令代码。因此，整个程序（指令序列）可以表示成一系列的数字，从而可以像数字数据那样被计算机所处理。

1．软件系统

一台计算机中全部程序的集合，统称为这台计算机的软件系统。

事实上，利用计算机进行计算、控制或做其他工作时，需要有各种用途的程序。因此，凡是用于一台计算机的各种程序，统称为这台计算机的程序或软件系统。

计算机软件按其功能可分为应用软件和系统软件两大类。

1）应用软件

应用软件是用户为解决某种应用问题而编制的程序，如工程设计程序、数据处理程序、自动控制程序、企业管理程序、情报检索程序、科学计算程序等等。

2）系统软件

系统软件用于实现计算机系统的管理、调度、监视和服务等功能，其目的是方便用户，提高计算机使用效率，发挥和扩充计算机的功能及用途。

总之，软件系统是在硬件系统的基础上，为有效使用计算机而配置的。

2．程序设计语言

1）机器语言

在早期的计算机中，人们直接用机器语言（即机器指令代码）来编写程序。这种用机器语言书写的程序，计算机完全可以“识别”并执行，所以又叫做目的程序。

但是，用机器语言编写程序是一件非常繁琐的工作，需要耗费大量的人力和时间，而且容易出错，出错后寻找错误也相当费事，这种情况大大限制了计算机的使用。

2）汇编语言

为了编写程序方便、提高机器使用效率，人们想出了一种办法，用一些约定的文字、符号和数字按规定的格式来表示各种不同的指令，每条基本指令都被指定了一个表示其功能又便于记忆的短的名字，称为指令助记符（如 ADD、SUB、MULT、JUMP 等），然后再用这些指令助记符表示的指令来编写程序，这就是所谓的汇编语言（Assembly Language）。

把用汇编语言编写的程序转换为计算机可以理解的、用机器语言表示的目的程序，通常由被称为汇编程序（Assembler）的计算机程序来完成。

通常被归为低级编程语言的机器语言及汇编语言，对于特定类型的计算机而言是惟一的，也就是说，一台 ARM 体系结构的计算机（如 PDA）无法理解一台 Intel Pentium 计算机的机器语言。

3）算法语言

为了进一步实现程序自动化，便于程序交流，使不熟悉具体计算机的人也能很方便地使用计算机，人们又创造了各种接近于数学语言的算法语言。

所谓算法语言，是指按实际需要规定好的一套基本符号，以及由这套基本符号构成程序的规则。算法语言比较接近数学语言，它直观通用，与具体机器无关，只要稍加学习就能掌握，便于推广和使用。有影响的算法语言包括 BASIC、FORTRAN、C、C++、JAVA 等。

大多数复杂的程序采用抽象的算法语言来编写，能够更便利地表达计算机程序员的设计思想，从而帮助减少程序错误。

用算法语言编写的程序称为源程序（Source），这种源程序是不能由机器直接识别和执行的，必须给计算机配备一个即懂算法语言又懂机器语言的“翻译”，才能把源程序转换为机器语言。通常采用下面两种方法：

⑴给计算机配置一套编译程序（Compiler），把用算法语言编写的源程序翻译成目的程序，然后在运行系统中执行目的程序，得出计算结果。编译程序和运行系统合称为编译系统。由于算法语言比汇编语言更

为抽象，因此有可能使用不同的编译器，把相同的算法语言源程序翻译成许多不同类型计算机的机器语言目的程序。

⑵使源程序通过所谓的解释程序（Interpreter）进行解释执行，即逐个解释并立即执行源程序的语句。它不是编译出目的程序后再执行，而是逐一解释语句并立即得出计算结果。

3．操作系统

操作系统是随着硬件和软件不断发展而逐渐形成的一套软件系统，用来管理计算机资源（如处理器、存储器、外围设备和各种编译、应用程序），自动调度用户的作业程序，从而使得多个用户能有效地共用一套计算机系统。操作系统的出现，使计算机的使用效率成倍提高，并且为用户提供了方便的使用手段和令人满意的服务质量。

根据不同使用环境的要求，操作系统目前大致可分为批处理操作系统、分时操作系统、网络操作系统、实时操作系统等多种。

4．数据库

计算机在信息处理、情报检索及各种管理系统中的各类应用，要求大量处理某些数据，建立和检索大量的表格。这些数据和表格可以按一定的规律组织起来，形成数据库（Database，DB），使得处理和检索数据更为方便、迅速。

所谓数据库就是实现有组织、动态地存储大量相关数据，方便多用户访问的计算机软、硬件资源所组成的系统。数据库和数据库管理软件一起，组成了数据库管理系统（Database Management System，DBMS）。

数据库管理系统有各种类型，目前许多计算机，包括微型计算机，都配有数据库管理系统。

1.2.3 计算机固件

随着大规模集成电路技术的发展和软件硬化的趋势，要明确划分计算机系统软、硬件的界限已经比较困难了。因为任何操作既可以由软件来实现，也可以由硬件来实现；任何指令的执行都可以由硬件完成，也可以由软件来完成。因此，计算机系统的软件与硬件可以互相转化，它们之间可以互为补充。对于某一功能采用硬件方案还是软件方案，取决于器件价格、速度、可靠性、存储容量、变更周期等因素。

容量大、价格低、体积小、可以改写的只读存储器为软件固化提供了良好的物质手段。现在已经可以把许多复杂的、常用的程序制作成所谓的固件（Firmware），就其功能而言是软件，但从形态来说又是硬件。

目前，在一片芯片上制作复杂的逻辑电路已经变得现实可行，这就为扩大指令的功能提供了物质基础，因此，本来通过软件手段实现的某些功能，现在可以通过硬件直接解释执行。

当研制一台计算机的时候，设计者必须明确分配每一级的任务，确定哪些情况使用硬件，哪些情况使用软件，而硬件始终放在最低级。就目前而言，一些计算机的特点，就是把原来通过编制程序实现的操作，如整数乘除法指令、浮点运算指令、处理字符串指令等等，改为直接由硬件完成。

1.3 计算机系统的概念

1.3.1 计算机系统的层次结构

现代计算机系统是硬件、固件和软件组成的一个十分复杂的整体。为了对这个系统进行描述、分析、设计和使用，人们从不同的角度提出了观察计算机的观点和方法。其中常用的一种方法，就是从语言的角度出发，把计算机系统按功能划分成 5 个层次级别，每一级以一种不同的语言为特征，每一级都能进行程序设计。

计算机系统的层次结构如图 1-10 所示：

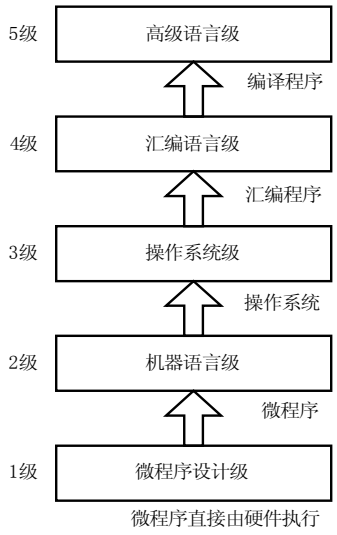


图 1-10 计算机系统的层次结构示意图

第 1 级是微程序设计级，属于硬件级，由机器硬件直接执行微指令，是计算机系统最底层的硬件系统。这一级也可直接用组合逻辑和时序逻辑电路实现。

第 2 级是机器语言级，也属于硬件级，由微程序解释机器指令系统。这一级控制硬件系统的操作。

第 3 级是操作系统级，属于（软硬件）混合级，由操作系统程序实现。这一级统一管理和调度计算机系统中的软硬件资源，支撑其他系统软件和应用软件，使计算机能够自动运行，发挥高效特性。

第 4 级是汇编语言级，属于软件级，由汇编程序支持和执行。这一级给程序设计人员提供了汇编语言这种符号形式语言，以减少程序编写的复杂性。

第 5 级是高级语言级，也属于软件级，由各种高级语言编译程序支持和执行。这一级是面向用户的，为方便用户编写应用程序而设置。

除第 1 级外，其他各级都得到下面各级的支持，同时也得到运行在下面各级上的程序的支持。第 1 级到第 3 级编写程序所采用的语言，基本上是二进制语言，机器执行和解释比较容易。第 4、5 两级编写程序所采用的语言是符号语言，用英文字母和符号来表示程序，因而便于大多数不了解硬件的人们使用计算机。

各层次之间关系紧密，上层是下层功能的扩展，下层是上层的基础，这是计算机系统层次结构的一个特点。

计算机系统中 5 个层次级别的特点如表 1-2 所示。

表 1-2 计算机系统中 5 个层次级别的特点

第 1 级	微程序设计级	由机器硬件直接执行微指令	硬件级	二进制语言
第 2 级	机器语言级	由微程序解释机器指令系统		
第 3 级	操作系统级	由操作系统程序实现	混合级	
第 4 级	汇编语言级	由汇编程序支持和执行	软件级	符号语言
第 5 级	高级语言级	由各种高级语言编译程序支持和执行		

1.3.2 计算机系统的3个术语

1．计算机体系结构

计算机体系结构（Computer Architecture）定义为机器语言程序员所看到的计算机系统的属性，包括概念性结构和功能特性。这些属性是机器语言程序员为使其设计的程序能在机器上正确运行所需遵循的计算机属性，是计算机系统中由硬件或固件完成的功能。对通用寄存器型机器来说，这些属性主要包括：数据表示、寻址规则、寄存器定义、指令集、终端系统、存储系统、信息保护、I/O 结构等。

计算机体系结构概念的实质是确定计算机系统中软硬件的界面，界面之上是软件的功能，界面之下是硬件和固件的功能。

2．计算机组成

计算机组成（Computer Organization）指的是计算机体系结构的逻辑实现，包括机器内部的数据流和控制流的组成以及逻辑设计等。它着眼于机器内各事件的排序方式与控制方式、各部件的功能以及各部件的联系。

3．计算机实现

计算机实现（Computer Implementation）指的是计算机组成的物理实现，包括处理机、存储器等部件的物理结构，器件的集成度和速度，模块、插件、底板的划分与连接，信号传输，电源、冷却及整机装配技术等。它着眼于器件技术和微组装技术，其中器件技术在实现技术中占主导作用。

上述三个术语具有不同的概念，各自包含不同的内容，但又有紧密的关系。

具有相同计算机体系结构（如指令系统相同）的计算机，因为速度要求不同等因素，可以采用不同的计算机组成。例如，取指令、指令译码、指令执行、访存取数、结果写回 5 个阶段，可以在时间上按顺序方式进行，也可以让它们在时间上按重叠方式进行（即时间并行），以提高执行速度。

同样，一种计算机组成可以采用多种不同的计算机实现。例如，存储器件可以采用静态 RAM（SRAM）芯片，也可以采用动态 RAM（DRAM）芯片，可以采用单片大规模集成电路，也可以采用中小规模集成电路进行构建。显然，这取决于性能价格比的要求与器件技术的现状。

1.3.3 计算机体系结构的分类

研究计算机系统分类方法，有助于人们认识计算机的体系结构和组成的特点，理解系统的工作原理和性能。

1966 年，Michael J. Flynn 从计算机体系结构的并行性能出发，按照指令流和数据流的不同组织方式，把计算机系统的结构分为 4 类（如表 1-3 所示）：

表 1-3 Flynn 分类法

	Single Instruction（单指令）	Multiple Instruction（多指令）
Single Data（单数据）	SISD（单指令流单数据流）	MISD（多指令流单数据流）
Multiple Data（多数据）	SIMD（单指令流多数据流）	MIMD（多指令流多数据流）

1．SISD 体系结构

SISD 计算机是传统的顺序执行的计算机，在同一时刻只能执行一条指令（即只有一个控制流）、处理一个数据（即只有一个数据流）。

SISD 计算机通常由一个处理器和一个存储器组成，如图 1-11 所示。通过执行单一的指令流，对单一的数据流进行操作，指令按顺序读取，数据在每一时刻也只能读取一个。

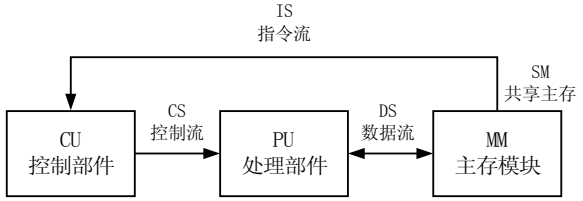


图 1-11 SISD 计算机

其主要缺点在于单个处理器的处理能力有限，同时，这种结构也没有发挥数据处理中的并行性潜力，在实时系统或高速系统中，很少采用 SISD 结构。

2．SIMD 体系结构

SIMD 计算机属于并行结构计算机，一条指令可以同时对多个数据进行运算。

SIMD 计算机由单一的指令部件控制，按照同一指令流的要求，为多个处理单元分配各不相同的数据并进行处理。SIMD 系统结构由一个控制器、多个处理器、多个存储模块和一个互连网络组成，如图 1-12 所示。

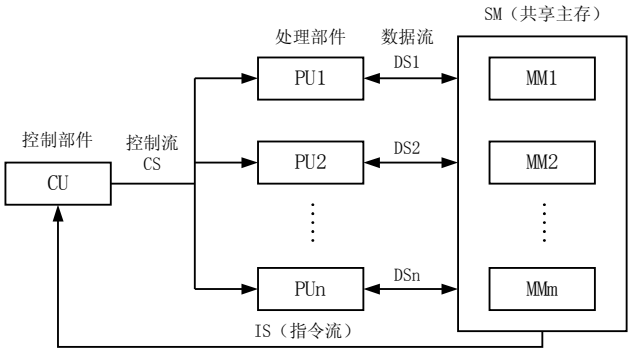


图 1-12 SIMD 计算机

SIMD 计算机以阵列处理机和向量处理机为代表。

3．MISD 体系结构

MISD 计算机具有多个处理单元，这些处理单元组成一个线性阵列，分别执行不同的指令流，而同一个数据流则顺次通过这个阵列中的各个处理单元，如图 1-13 所示。

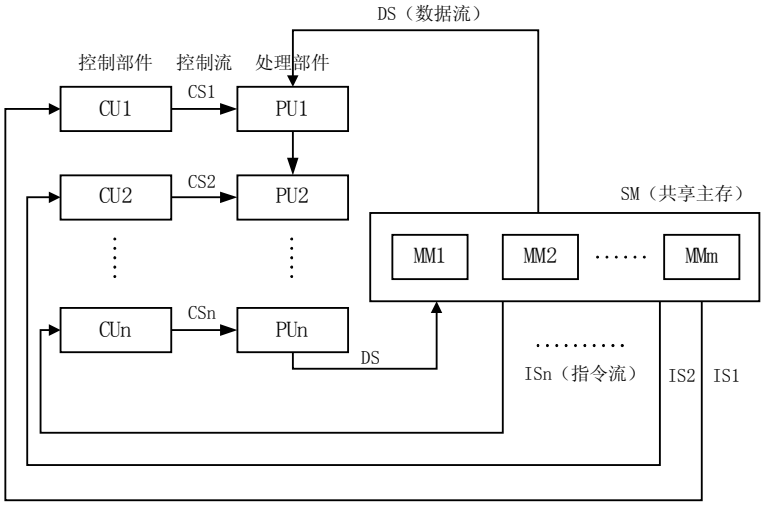


图 1-13 MISD 计算机

MISD 系统结构只适用于某些特定的算法，在目前常见的计算机系统中很少见。

4．MIMD 体系结构

MIMD 计算机属于并行结构计算机，多个处理单元根据不同的控制流程执行不同的操作，处理不同的数据。

典型的 MIMD 系统由多台处理机、多个存储模块和一个互连网络组成，如图 1-14 所示。每台处理机执行自己的指令，操作数也是各取各的。

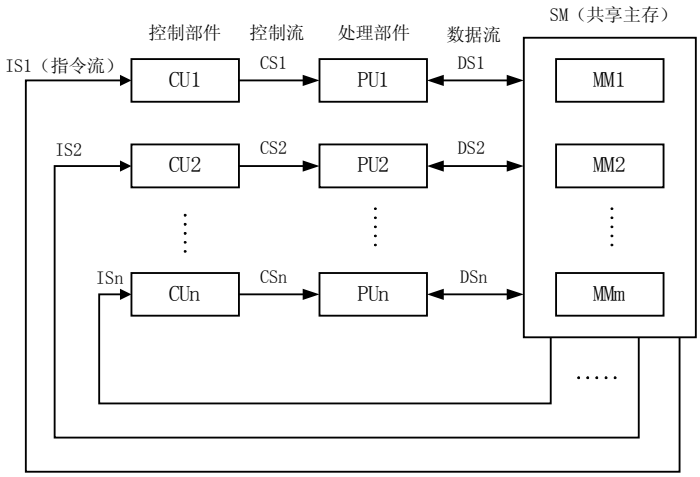


图 1-14 MIMD 计算机

MIMD 计算机是能够实现指令、数据作业、任务等各级全面并行计算的多机处理系统。

在 MIMD 结构中，每个处理器都可以单独编程，因而这种结构的可编程能力是最强的。但由于要用大量的硬件资源来解决可编程问题，硬件的利用率不高。

MIMD 计算机以多处理机和机群系统为代表。

第2章 运算方法和运算器

2.1 数据表示基础

计算机的基本功能是对数据、文字、声音、图形、图像和视频等信息进行加工处理，其中数据有两大类：一类是数值数据，如+314、-3.14、5^3等，有“量”的概念；另一类是非数值数据，如各种字母和符号。无论是数值数据还是非数值数据，在计算机中都是用二进制数码表示的，而文字、声音、图形、图像和视频等信息要在计算机中处理，都要事先数字化，即把文字、声音、图形、图像和视频等信息转换为二进制数码。在计算机内部，各种信息都必须以数字化编码的形式被存储、传送和加工处理。因此，学习计算机课程，首先必须掌握信息编码的概念与处理技术。

信息的数字化编码，是指用“0”和“1”这两个最简单的二进制数码，按照一定的组合规则来表示数据、文字、声音、图像、视频等复杂信息。本章主要讨论数据信息在计算机中的表示方法和运算方法。

2.1.1 进制数及其转换

数值数据是表示数量大小的数据，有多种表示方法。日常生活中一般采用十进制数进行计数和计算，但十进制数难以在计算机内直接存储与运算。在计算机系统中，通常将十进制数作为人机交互的媒介，而数据则以二进制数的形式存储和运算。

计算机采用二进制的主要原因有以下几点：

⑴易于物理实现

二进制在技术上最容易实现。这是因为具有两种稳定状态的物理器件很多，如门电路的导通与截止、电压的高与低等，而它们恰好可以对应表示“1”和“0”这两个数码。假如采用十进制，那么就要制造具有10种稳定状态的物理电路，而这是非常困难的。

⑵运算规则简单

数学推导已经证明，对R进制数进行算术求和或求积运算，其运算规则各有R(R+1)/2种。如采用十进制，则R=10，就有55种求和或求积的运算规则；而采用二进制，则R=2，仅有3种求和或求积的运算规则，

以加法为例：0+0=0，0+1=1 (1+0=1)，1+1=10，因而可以大大简化运算器等物理器件的设计。

⑶机器可靠性高

由于电压的高和低、电流的有和无等都是一种质的变化，两种物理状态稳定、分明，因此，二进制码传输的抗干扰能力强，鉴别信息的可靠性高。

⑷逻辑判断方便

采用二进制后，仅有的两个符号“1”和“0”正好可以与逻辑命题的两个值“真”和“假”相对应，能够方便地使用逻辑代数这一有力工具来分析和设计计算机的逻辑电路。

但是，用二进制表示一个数，其所使用的位数要比用十进制表示长得多，书写和阅读都不方便，也不容易理解。为了书写和阅读的方便，人们通常使用十六进制来弥补二进制的这一不足。

1. 进位计数制

在人类的生产和生活中，经常要遇到数的表示问题，人们通常采用从低位向高位进位的方式来进行计数，这种表示数据的方法称为进位计数制。讨论进位计数制要涉及到两个基本概念：基数（Radix）和权（Weight）。

1）十进制

在进位计数制中，每个数位所用到的数码符号的个数叫做基数。十进制是人们最熟悉的一种进位计数制，每个数位允许选用0~9共10个不同数码中的某一个，因此十进制的基数为10。每个数位计满10就向高位进位，即“逢10进1”。

在一个数中，数码在不同的数位上所表示的数值是不同的。每个数码所表示的数值就等于该数码本身

乘以一个与它所在数位有关的常数，这个常数叫做权。例如：十进制数6543.21，数码“6”所在数位的权为1000，这一位所代表的数值即为6×10^3=6000，“5”所在数位的权为100，这一位所代表的数值即为5×10^2=500，……。

2）二进制

计算机中信息的存储、处理和传送采用的都是二进制，不论是指令还是数据，或是多媒体信息（声音、图形、图像等），都必须采用二进制编码形式才能存入计算机中。

二进制是一种最简单的进位计数制，它只有两个不同的数码：“0”和“1”，即基数为2，逢2进1。任意数位的权是2^i。

3）十六进制

十六进制数的基数为16，逢16进1，每个数位可取0,1,…,9,A,B,…,F共16个不同的数码和符号中的任意一个，其中A~F分别表示十进制数值10~15。

既然有不同的进位计数制，那么在给出一个数的同时，就必须指明它是哪种进制的数，例如：$(1010)_2$、$(1010)_{10}$、$(1010)_{16}$ 所代表的数值完全不同，如果不用下标加以标注，就会产生歧义。除了用下标表示之外，还可以用后缀字母来表示不同的数制，后缀B表示该数是二进制（Binary）数，后缀H表示该数是十六进制（Hexadecimal）数，而后缀D表示该数是十进制（Decimal）数。十进制数在书写时可以省略后缀D，其他进制数在书写时一般不能省略后缀。例如：有3个数分别为375D、101B和AFEH，从后缀字母就可以知道它们分别是十进制数、二进制数和十六进制数。

2. 各种进制数之间的转换

1）二进制数转换为十六进制数

将一个二进制数转换成十六进制数的方法是将二进制数的整数部分和小数部分分别进行转换，即以小数点为界，整数部分从小数点开始往左数，每4位分成一组，当最左边的数不足4位时，可根据需要在数的最左边添加若干个“0”以补足4位；对于小数部分，从小数点开始往右数，每4位分成一组，当最右边的数不足4位时，可根据需要在数的最右边添加若干个“0”以补足4位，最终使二进制数的总的位数是4的倍数，然后用相应的十六进制数取而代之。

例如：

111011.1010011011B = 0011 1011.1010 0110 1100B = 3B.A6CH

2）十六进制数转换为二进制数

要将十六进制数转换成二进制数，只要将1位十六进制数写成4位二进制数，然后将整数部分最左边的“0”和小数部分最右边的“0”去掉即可。

例如：

3B.328H = 0011 1011.0011 0010 1000B = 111011.001100101B

3）二进制数转换为十进制数

要将一个二进制数转换成十进制数，只要把二进制数的各位数码与它们的权相乘，再把乘积相加，就得到对应的十进制数，这种方法称为按权展开相加法。

例如：

$100011.1011B = 1\times2^5 + 1\times2^1 + 1\times2^0 + 1\times2^{-1} + 1\times2^{-3} + 1\times2^{-4} = 35.6875D$

4）十进制数转换为二进制数

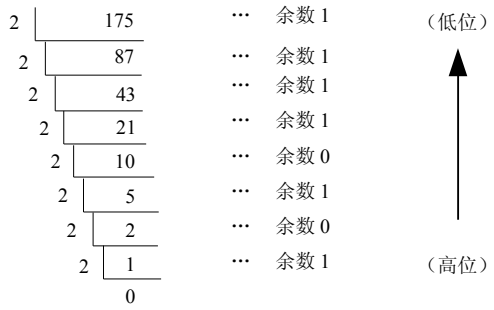
要将一个十进制数转换成二进制数，通常采用的方法是基数乘除法。这种转换方法是对十进制数的整数部分和小数部分分别进行处理，整数部分用除基取余法，小数部分用乘基取整法，最后将它们拼接起来即可。

⑴十进制整数转换为二进制整数（除基取余法）

十进制整数转换为二进制整数的规则是：除以基数（2）取余数，先得到的余数为低位，后得到的余数为高位。

具体的做法是：用 2 连续去除十进制整数，直到商等于 0 为止，然后按逆序排列每次的余数（先取得的余数为低位），便得到与该十进制数相对应的二进制数各位的数值。

例如，将 175D 转换成二进制数：



所以，175D=10101111B

⑵十进制小数转换为二进制小数（乘基取整法）

十进制小数转换为二进制小数的规则是：乘以基数（2）取整数，先得到的整数为高位，后得到的整数为低位。

具体的做法是：用 2 连续去乘十进制数的小数部分，直至乘积的小数部分等于 0 为止，然后按顺序排列每次乘积的整数部分（先取得的整数为高位），便得到与该十进制数相对应的二进制数各位的数值。

例如，将 0.3125D 转换成二进制数：

所以，0.3125D = 0.0101B

若要将十进制数 175.3125 转换成二进制数，应对整数部分和小数部分分别进行转换，然后再进行整合：

175.3125D=10101111.0101B

需要注意的是，十进制小数常常不能准确地换算为等值的二进制小数，存在有一定的换算误差。

例如，将 0.5627D 转换成二进制数：

0.5627×2 = 1.1254

0.1254×2 = 0.2508

0.2508×2 = 0.5016

0.5016×2 = 1.0032

0.0032×2 = 0.0064

0.0064×2 = 0.0128

……

由于小数位始终达不到 0，因此这个过程会不断进行下去。通常的做法是：根据精度要求，截取一定的数位，其误差值小于截取的最低一位数的权。

当要求二进制数取 m 位小数时，一般可求 m+1 位，然后对最低位作“0 舍 1 入”处理。

例如：

0.5627D = 0.100100…B

若取精度为 5 位，则由于小数点后第 6 位为“0”，被舍去，所以：

0.5627D = 0.10010B

2.1.2 数的机器码表示

1. 机器数和真值

二进制数有正负之分，如 N_1=+0.101101，N_2=-0.101101，则 N_1 是个正数，N_2 是个负数。机器不能直接把符号“+”、“-”表示出来，为了能在计算机中表示正负数，必须引入符号位，即把正负符号也用 1 位二进制数码来表示。把符号位和数值位一起编码来表示相应的数的表示方法包括：原码、补码、反码、移码等。

为了便于在计算机中表示，同时又便于与实际值相区分，在此首先引入机器数和真值的概念。

机器数 用二进制数“0”或“1”来表示数的符号，“0”表示正号，“1”表示负号，且把符号位置于该数的最高数值位之前，这样表示的数称为机器数（或称机器码），即把符号位和数值位一起编码来表示的数就是机器数。

真值 一般书写中用“+”、“-”来表示数的符号，这样表示的数称为真值。

例如：N_1 = +0.101101，N_2 = -0.101101，这是真值，表示成机器数（以原码为例）就是 $[N_1]_{原}$= 0.101101，$[N_2]_{原}$ = 1.101101。

机器数有原码、补码、反码和移码四种表示形式。下面以整数为例说明原码、补码、反码和移码的表示方法。

2. 原码

符号位为 0 表示正数，为 1 表示负数，数值部分用二进制数的绝对值表示的方法称为原码表示法，通常用$[X]_{原}$表示 X 的原码。

例如，要表示+59 和-59 的原码。假设机器数的位数 8 位（即机器的字长为 8 位），最高位是符号位，其余 7 位是数值位，那么，+59 和-59 的原码分别表示为：

$[+59]_{原}$=00111011　　$[-59]_{原}$=10111011

写成一般式则为：

正数的原码　$[X]_{原}=X$　　$(0<X<2^{n-1})$

负数的原码　$[X]_{原}=2^{n-1}-X$　　$(-2^{n-1}<X<0)$

注意：0 的原码有两个值，有“正零”和“负零”之分，机器遇到这两种情况都当作 0 处理。

$[+0]_{原}$=00000000　　$[-0]_{原}$=10000000

原码的表示方法简单易懂，与真值转换方便，但在进行加减法运算时，符号位不能直接参加运算，而是要分别计算符号位和数值位。当两数相加时，如果是同号，则数值相加；如果是异号，则要进行减法运算。而在进行减法运算时，还要比较绝对值的大小，然后用大数减去小数，最后还要给运算结果选择恰当的符号。

为了解决这些问题，人们引进了数的补码表示法。

3. 补码

什么是补码？我们先用日常生活中的实例来进行说明。假如现在时间是 7 点，而你的手表却指向了 9 点，如何调整手表的时间？有两种方法拨动时针，一种是顺时针拨，即向前拨动 10 个小时；另一种是逆时针拨，即向后拨 2 个小时。从数学的角度可以表示为：

(9+10) -12=19-12=7

或 9-2=7

可见，对钟表来说，向前拨 10 个小时和向后拨 2 个小时的结果是一样的，减 2 可以用加 10 来代替。这是因为钟表是按 12 进位的，12 就是它的“模”。对模 12 来说，-2 与+10 是“同余”的，也就是说，-2 与+10 对于模 12 来说是互为补数的。

计算机中的加法器是以 2^n 为模的有模器件，因此可以引入补码，把减法运算转换为加法运算，以简化运算器的设计。

补码的定义：把某数 X 加上模数 K，称为以 K 为模的 X 的补码。

$[X]_{补}=K+X$

因此，正数的补码的最高位为符号“0”，数值部分为该数本身；负数的补码的最高位为符号“1”，数

值部分为用模减去该数的绝对值。

通过用模 2^n 减去某数的绝对值的方法来求某数的补码比较麻烦，求一个二进制数的补码的简便方法是：正数的补码与其原码相同；负数的补码是符号位不变，数值位逐位取反（即求其反码），然后在最低位加1。

例如，$[+59]_{补}=[+59]_{原}=00111011$，而$[-59]_{原}=10111011$，因此，$[-59]_{补}$= 11000100+1 = 11000101。

注意：0 的补码只有一种形式，就是 n 位 0。

采用补码表示法进行加减法运算，比原码运算方便多了，符号位可以和数值位一起参加运算，而且不论数是正还是负，计算机总是做加法，减法运算可转换为加法运算。

4. 反码

引入反码的目的是便于求负数的补码。

正数的反码与原码相同，负数的反码是符号位不变，数值位逐位取反。

例如：$[+59]_{反}=[+59]_{原}=00111011$，而$[-59]_{原}=10111011$，因此，$[-59]_{反}=11000100$。

注意：0 的反码也有两个，$[+0]_{反}=00000000$，$[-0]_{反}=11111111$

在计算机中，求一个数的反码很容易，因此，求一个数的补码也就易于实现。

采用补码运算，计算机的控制线路较为简单，所以，目前大多数计算机均采用补码存储、补码运算，其运算结果仍为补码形式。

综上所述，在 n 位机中，用 n 位二进制数补码表示一个带符号的整数时，最高位为符号位，后面 n-1 位为数值部分。n 位二进制数补码表示的范围为-2^{n-1}～$+2^{n-1}-1$。例如，在 8 位机中，补码表示的范围为-128～+127。

表 2-2 列出了 8 位二进制数码在各种表示形式下的对应真值。

表 2-2 8 位二进制数的各种表示方法

二进制数码	真值			
	无符号数	原码	反码	补码
00000000	0	+ 0	+ 0	+ 0
00000001	1	+ 1	+ 1	+ 1
00000010	2	+ 2	+ 2	+ 2
…	…	…	…	…
01111101	125	+125	+125	+125
01111110	126	+126	+126	+126
01111111	127	+127	+127	+127
10000000	128	- 0	-127	-128
10000001	129	- 1	-126	-127
10000010	130	- 2	-125	-126
…	…	…	…	…
11111101	253	-125	-2	-3
11111110	254	-126	-1	-2
11111111	255	-127	-0	-1

5）移码

移码也称为增码或偏码，常用于表示浮点数中的阶码。

移码可由补码求得，只要把补码的符号位取反就得到了移码。

2.2 数值数据的表示

2.2.1 定点数的表示

计算机中常用的数据表示格式有两种，一是定点格式，二是浮点格式。所谓定点数和浮点数，是指在计算机中一个数的小数点的位置是固定的还是浮动的：如果一个数中小数点的位置是固定的，则为定点数；如果一个数中小数点的位置是浮动的，则为浮点数。一般来说，定点格式可表示的数值的范围有限，但要求的处理硬件比较简单。而浮点格式可表示的数值的范围很大，但要求的处理硬件比较复杂。

采用定点数表示法的计算机称为定点计算机，采用浮点数表示法的计算机称为浮点计算机。定点机在使用上不够方便，但其构造简单，造价低，一般微型机和单片机大多采用定点数的表示方法。浮点机可表示的数的范围比定点机大得多，使用也比较方便，但是比定点机复杂，造价高，在相同的条件下浮点运算

比定点运算速度慢。目前，一般大、中型计算机及高档微型机都采用浮点表示法，或同时具有定点和浮点两种表示方法。

所谓定点格式，即约定机器中所有数据的小数点位置是固定不变的。通常将定点数据表示成纯小数或纯整数。为了将数表示成纯小数，通常把小数点固定在数值部分的最高位之前；而为了把数表示成纯整数，则把小数点固定在数值部分的最后面，如图 2-1 所示。

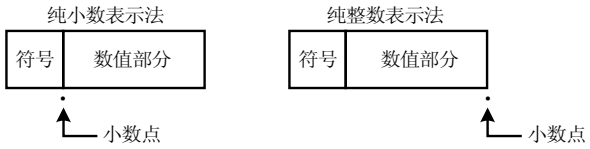


图 2-1 定点数表示法

图中所标示的小数点“.”，在机器中是不表示出来的，而是事先约定在固定的位置。对于一台计算机，一旦确定了小数点的位置就不再改变。

对纯小数进行运算时，要用适当的比例因子进行折算，以免产生溢出，或过多损失精度。

假设用一个 n 位字来表示一个定点数 $x= x_0\ x_1\ x_2 \dots x_{n-1}$，其中一位 x_0 用来表示数的符号位，其余位数代表它的量值。为了对所有 n 位进行统一处理，符号位 x_0 通常放在最左位置，并用数值 0 和 1 分别代表正号和负号。对于任意定点数 $x= x_0\ x_1\ x_2 \dots x_{n-1}$，如果 x 表示的是纯小数，那么小数点位于 x_0 和 x_1 之间，数的表示范围为：$0\leq |x| \leq 1-2^{-(n-1)}$；如果 x 表示的是纯整数，则小数点位于最低位 x_{n-1} 的右边，数的表示范围为：$0\leq |x| \leq 2^{n-1}-1$。

目前计算机中大多采用定点纯整数表示，因此将定点数表示的运算简称为整数运算。

2.2.2 浮点数的表示

在定点数表示中存在的一个问题是，难以表示数值很大的数据和数值很小的数据。例如，电子的质量（9×10^{-28} 克）和太阳的质量（2×10^{33} 克）相差甚远，在定点计算机中无法直接表示，因为小数点只能固定在某一个位置上，从而限制了数据的表示范围。

为了表示更大范围的数据，数学上通常采用科学计数法，把数据表示成一个小数乘以一个以 10 为底的指数。

例如，在计算机中，电子的质量和太阳的质量可以分别取不同的比例因子，以使其数值部分的绝对值小于 1，即：

$9\times10^{-28} = 0.9\times10^{-27}$

$2\times10^{33} = 0.2\times10^{34}$

这里的比例因子 10^{-27} 和 10^{34} 要分别存放在机器的某个单元中，以便以后对计算结果按此比例增大。显然，这要占用一定的存储空间和运算时间。

浮点表示法就是把一个数的有效数字和数的范围在计算机中分别予以表示。这种把数的范围和精度分别表示的方法，相当于数的小数点位置随比例因子的不同而在一定范围内自由浮动，改变指数部分的数值相当于改变小数点的位置。在这种表示法中，小数点的位置是可以浮动的，因此称为浮点表示法。

浮点数的一般表示形式为：

一个十进制数 N 可以写成：$N = 10^e\times M$

一个二进制数 N 可以写成：$N = 2^e\times M$

其中，M 称为浮点数的尾数，是一个纯小数；e 是比例因子的指数，称为浮点数的指数，是一个整数。在计算机中表示一个浮点数时，一是要给出尾数 M，用小数形式表示；二是要给出指数 e，用整数形式表示，常称为阶码。尾数部分给出有效数字的位数，因而决定了浮点数的表示精度；阶码部分指明了小数点在数据中的位置，因而决定了浮点数的表示范围。浮点数也是有符号数，带符号的浮点数的表示如图 2-2 所示。

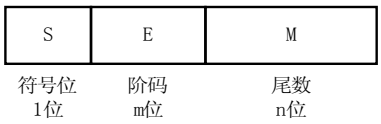


图 2-2 浮点数的表示

其中，S 为尾数的符号位，放在最高一位；E 为阶码，紧跟在符号位之后，占 m 位；M 为尾数，放在低位部分，占 n 位。

1. 规格化浮点数

若不对浮点数的表示做出明确规定，同一个浮点数的表示就不是惟一的。例如：

$$(1.75)_{10} = (1.11)_2 = 1.11\times2^0$$
$$= 0.111\times2^1$$
$$= 0.0111\times2^2$$
$$= 0.00111\times2^3$$

为了提高数据的表示精度，需要充分利用尾数的有效位数。当尾数的值不为 0 时，尾数域的最高有效位应为 1，否则就要用修改阶码同时左右移动小数点的办法，使其变成符合这一要求的表示形式，这称为浮点数的规格化。

2. IEEE-754 标准浮点格式

在 IEEE-754 标准出现之前，业界并没有一个统一的浮点数标准，相反，很多计算机制造商都在设计自己的浮点数规则以及运算细节。

为了便于软件的移植，浮点数的表示格式应该有一个统一的标准。1985 年，IEEE（Institute of Electrical and Electronics Engineers，美国电气和电子工程师协会）提出了 IEEE-754 标准，并以此作为浮点数表示格式的统一标准。目前，几乎所有的计算机都支持该标准，从而大大改善了科学应用程序的可移植性。

IEEE 标准从逻辑上采用一个三元组 {S, E, M} 来表示一个数 N，它规定基数为 2，符号位 S 用 0 和 1 分别表示正和负，尾数 M 用原码表示，阶码 E 用移码表示。根据浮点数的规格化方法，尾数域的最高有效位总是 1，由此，该标准约定这一位不予存储，而是认为隐藏在小数点的左边，因此，尾数域所表示的值是 1.M（实际存储的是 M），这样可使尾数的表示范围比实际存储多一位。为了表示指数的正负，阶码 E 通常采用移码方式来表示，将数据的指数 e 加上一个固定的偏移量后作为该数的阶码，这样做既可避免出现正负指数，又可保持数据的原有大小顺序，便于进行比较操作。

目前，大多数高级语言都按照 IEEE-754 标准来规定浮点数的存储格式。IEEE-754 标准规定，单精度浮点数用 4 字节（即 32 位）存储，双精度浮点数用 8 字节（即 64 位）存储，如图 2-3 所示：

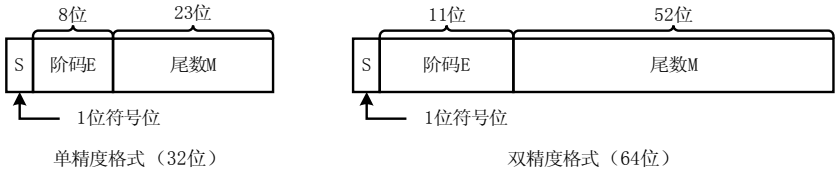


图 2-3 IEEE-754 标准浮点格式

单精度格式（32 位）：符号位（S）1 位；阶码（E）8 位，阶码的偏移量为 127（7FH）；尾数（M）23 位，用小数表示，小数点放在尾数域的最前面；

双精度格式（64 位）：符号位（S）1 位；阶码（E）11 位，阶码的偏移量为 1023（3FFH）；尾数（M）52 位，用小数表示，小数点放在尾数域的最前面。

在 IEEE-754 标准中，一个规格化的 32 位浮点数 X 的真值可表示为：

$$X = (-1)^s\times(1.M)\times2^{E-127} \qquad e = E-127 \qquad \text{（式 2-9）}$$

在 IEEE-754 标准中，一个规格化的 64 位浮点数 X 的真值可表示为：

$$X = (-1)^s\times(1.M)\times2^{E-1023} \qquad e = E-1023 \qquad \text{（式 2-10）}$$

由于双精度格式的原理与单精度格式相同，仅仅是表示的位数有所增加，所以，下面主要介绍单精度格式（32 位）浮点数的表示方法。

当一个浮点数的尾数为 0，不论其阶码为何值，或者当阶码的值遇到比它所能表示的最小值还小时，不管其尾数为何值，计算机都把该浮点数看成零值，称为机器零。

当阶码 E 为全 0 且尾数 M 也为全 0 时，表示的真值 X 为零，结合符号位 S 为 0 或 1，有正零和负零之分。当阶码 E 为全 1 且尾数 M 也为全 0 时，表示的真值 X 为无穷大（∞），结合符号位 S 为 0 或 1，有+∞和-∞之分。这样，在 32 位浮点数表示中，要除去 E 用全 0 和全 1（255）表示零和无穷大的特殊情况，因此，阶码 E 的取值范围变为 1～254，指数的偏移量不选 128（10000000B），而选 127（01111111B）。对于 32 位规格化浮点数，真正的指数值 e 为-126～+127，因此，数的绝对值的范围是 2^{-126}～2^{127}≈10^{-38}～10^{38}。

2.3 非数值数据的表示

2.3.1 字符与字符串的表示

非数值数据，通常指的是字符、字符串、图形符号、汉字等数据，它们并不用来表示数值的大小，一般情况下也不对它们进行算术运算。

1. ASCII 字符

由于计算机内部只能识别和处理二进制代码，所以字符必须按照一定的规则用一组二进制编码来表示。

字符编码方式有很多种，美国国家标准局（ANSI）制定的 ASCII（American Standard Code for Information Interchange，美国信息交换标准码）是现今最为通用的单字节编码系统，它主要用于显示现代英文字母和符号，已被国际标准化组织（ISO）定为国际标准，称为 ISO 646 标准。

ASCII 字符编码表如表 2-3 所示，表中的横轴为 7 位 ASCII 码高 3 位 $b_6b_5b_4$ 的二进制表示，纵轴为 ASCII 码低 4 位 $b_3b_2b_1b_0$ 的二进制表示，括号中的数字为对应的十六进制表示。

表 2-3 ASCII 字符编码表

$b_6b_5b_4$ $b_3b_2b_1b_0$	000 (0)	001 (1)	010 (2)	011 (3)	100 (4)	101 (5)	110 (6)	111 (7)
0000 (0)	NUL	DLE	SP	0	@	P	`	P
0001 (1)	SOH	DC1	!	1	A	Q	a	q
0010 (2)	STX	DC2	"	2	B	R	b	r
0011 (3)	ETX	DC3	#	3	C	S	c	s
0100 (4)	EOT	DC4	$	4	D	T	d	t
0101 (5)	ENQ	NAK	%	5	E	U	e	u
0110 (6)	ACK	SYN	&	6	F	V	f	v
0111 (7)	BEL	ETB	'	7	G	W	g	w
1000 (8)	BS	CAN	(	8	H	X	h	x
1001 (9)	HT	EM	)	9	I	Y	i	y
1010 (A)	LF	SUB	*	:	J	Z	j	z
1011 (B)	VT	ESC	+	;	K	[	k	{
1100 (C)	FF	FS	,	<	L	\	l	|
1101 (D)	CR	GS	-	=	M	]	m	}
1110 (E)	SO	RS	.	>	N	^	n	~
1111 (F)	SI	US	/	?	O	_	o	DEL

ASCII 码用 7 位二进制编码（0~127）表示一个字符，总共可以表示 128 个字符，其中有 95 个是可显示和打印的字符，包括 10 个十进制数字（0~9）、52 个英文大写和小写字母（A~Z，a~z）、以及若干个运算符和标点符号，除此之外的 33 个字符是不可显示和打印的控制符号，原先用于控制计算机外围设备的某些工作特性，现在多数已被废弃。

计算机通常用一个字节（8 位）来存放一个 ASCII 字符，字节的低 7 位表示不同的 ASCII 字符，而字节的最高 1 位固定为 0。在有些情况下，字节的最高 1 位也可以用作奇偶校验位以检验错误，或用作西文字符和汉字的区分标识。

除了使用字节最高位为 0 的标准 ASCII 码（0~127）之外，通过使用字节最高位为 1 的另外 128 个编码（128~255），许多公司和组织还自行定义了不少互不兼容的扩展 ASCII 码系统。扩展 ASCII 码用 8 位二进制表示一个字符，总共可以表示 256 个不同的字符。

2. Unicode 码

现今人类使用了接近 6800 种不同的语言，即使是扩展 ASCII 码这类 8 位代码也不能满足需要。解决问题的最佳方案是设计一种全新的编码方法，而这种方法必须有足够的能力来容纳全世界所有语言中任意一种语言的所有符号，这就是 Unicode（统一码）。Unicode 为每种语言中的每个字符设定了统一并且惟一的二进制编码，以满足跨语言、跨平台进行文本转换、处理的要求。

目前实际应用的 Unicode 对应于 UCS-2（2-byte Universal Character Set，2 字节通用字符集），每个字符占用 2 个字节，使用 16 位的编码空间，理论上允许表示 2^{16}=65536 个字符，可以基本满足各种语言的使用需要。实际上目前版本的 Unicode 尚未填充满这 16 位编码，从而为特殊的应用和将来的扩展保留了大量的编码空间。这个编码空间已经非常大了，但设计者考虑到将来某一天它可能也会不够用，所以又定义了 UCS-4 编码，每个字符占用 4 个字节（实际上只用了 31 位，最高位必须为 0），理论上可以表示 2^{31}=2 147 483 648 个字符。

在 PC 机中，若使用扩展 ASCII 码、Unicode UCS-2 和 UCS-4 方法分别表示一个字符，则三者之间的差别为：扩展 ASCII 码用 8 位表示，Unicode UCS-2 用 16 位表示，Unicode UCS-4 用 32 位表示。

3. 字符串

字符串是指连续的一串字符，它们通常占用主存中连续的多个字节，每个字节存放一个字符（以 ASCII 字符为例）。当主存字由 2 个或 4 个字节组成时，在同一个主存字中，既可按从低位字节向高位字节的顺序存放字符串的内容，也可按从高位字节向低位字节的顺序存放字符串的内容。这两种存放方式都是常用方式，不同的计算机可以选用其中的任何一种。

例如下列字符串：

IF␣A>=B␣THEN␣READ(C)

可以按图 2-4 所示从高位字节到低位字节依次存放在主存中。图中，主存单元长度是 4 个字节，每个字节中存放相应字符的 ASCII 值，文字表达式中的空格“␣”在主存中也占一个字节的位置。因此，每个字节分别存放十六进制的 49、46、20、41、3E、3D、42、20、54、48、45、4E、20、52、45、41、44、28、43、29。

I	F	␣	A
>	=	B	␣
T	H	E	N
␣	R	E	A
D	(	C	)

ASCII 字符

49	46	20	41
3E	3D	42	20
54	48	45	4E
20	52	45	41
44	28	43	29

ASCII 码（十六进制表示）

图 2-4 存放在主存单元中的字符串

2.3.2 汉字的表示

汉字处理是我国计算机推广应用中必须解决的问题。汉字的字数繁多，字形复杂，读音多变，常用汉字就有 7000 个左右。要在计算机中表示汉字，最方便的方法是为每个汉字设计一个编码，而且要使这些编码与西文字符和其他字符有明显的区别。

目前，在我国使用的计算机汉字操作平台中常见的有以下 4 种汉字字符集。

1. GB2312 字符集

GB2312 即国标码字符集 GB2312-80，全称为《信息交换用汉字编码字符集-基本集》，由中国国家标准总局发布，1981 年 5 月 1 日起实施，是中国国家标准的简体中文字符集。它所收录的汉字已经覆盖 99.75% 的使用频率，基本满足了汉字的计算机处理需要。

2. BIG5 字符集

BIG5 又称大五码，1984 年由台湾财团法人信息工业策进会和 5 家软件公司宏碁（Acer）、神通（MiTAC）、佳佳、零壹（Zero One）、大众（FIC）创立，故称大五码。BIG5 码的产生，一方面是因为当时台湾不同厂商各自推出不同的编码，如倚天码、IBM PS55、王安码等，彼此不能兼容；另一方面，台湾当时尚未推出官方的汉字编码，而 GB2312 编码亦未收录繁体中文字。BIG5 字符集共收录 13053 个中文字，该字符集在中国台湾使用。

尽管 BIG5 码内包含一万多个字符，但是没有考虑社会上流通的人名、地名用字、方言用字、化学及生物

学科等用字，没有包含日文平假名及片假名字母。

3. GBK 字符集

1995 年底推出的 GBK（汉字内码扩展规范）编码是中文编码扩展国家标准，该编码标准兼容 GB2312，共收录汉字 21003 个、符号 883 个，并提供 1894 个造字码位，简、繁体字融于一库。

GBK 字符集主要扩展了对繁体中文字的支持。

4. GB18030 字符集

GB18030 的全称是 GB18030-2000《信息交换用汉字编码字符集-基本集的扩充》，是中国政府于 2000 年 3 月 17 日发布的新的汉字编码国家标准，2001 年 8 月 31 日后在中国市场上发布的软件必须符合该标准。

GB18030 字符集标准解决了汉字、日文假名、朝鲜语和中国少数民族文字组成的大字符集计算机编码问题。该标准采用单字节、双字节和四字节三种编码方式，字符总编码空间超过 150 万个编码位，收录了 27484 个汉字，覆盖中文、日文、朝鲜语和中国少数民族文字，能满足中国大陆、香港、台湾、日本和韩国等东亚地区信息交换多文种、大字量、多用途、统一编码格式的要求，并且与 Unicode 3.0 版本兼容，与以前的国家字符编码标准兼容。

2.4 定点运算和定点运算器

计算机中的基本运算有两大类：算术运算和逻辑运算。

算术运算：主要是指加、减、乘、除四则运算，参加运算的数据一般要考虑符号和编码格式（即原码、反码还是补码）。由于数据有定点数和浮点数两大类，因此也可以分为定点数四则运算和浮点数四则运算。

逻辑运算：包括逻辑与、或、非、异或等运算，针对不带符号的二进制数。

2.4.1 定点运算

1. 定点加减法运算

定点加、减法运算属于算术运算，要考虑参加运算数据的符号和编码格式。在计算机中，定点数据主要有原码、反码、补码三种形式；在定点加减法运算时，三种编码形式从理论上来说都是可以实现的，但难度不同。

首先，原码是一种最直接、方便的编码方案，但是它的符号位不能直接参加加减运算，必须单独处理。在原码加减运算时，一方面要根据参加运算的两个数据的符号位，以及指令的操作码来综合决定到底是做加法还是减法运算，另一方面运算结果的符号位也要根据运算结果来单独决定，实现起来很麻烦。

其次，反码的符号位可以和数值位一起参加运算，而不用单独处理。但是反码的运算存在一个问题，就是符号位一旦有进位，结果就会发生偏差，因此要采用循环进位法进行修正，即符号位的进位要加到最低位上去，这也会带来运算的不便。

两个数进行补码运算时，可以把符号位与数值位一起处理。只要最终的运算结果不超出机器数允许的表示范围，运算结果一定是正确的。这样一来，补码运算就显得很简单，因为它既不需要事先判断参加运算数据的符号位，运算结果的符号位如果有进位，也只要将进位数据舍弃即可，不需做任何特殊处理。

因此，现代计算机的运算器一般都采用补码形式进行加减法运算。

1）补码加法

补码加法的公式是：

$$[x]_{补}+[y]_{补}=[x+y]_{补} \quad (\text{mod } 2) \qquad (式\ 2\text{-}11)$$

在模 2 意义下，任意两数的补码之和等于该两数之和的补码，这是补码加法的理论基础。之所以说是模 2 运算，是因为最高位（即符号位 x_0 和 y_0）相加结果中的向上进位是要舍去的。

由此可见，当两数以补码形式相加时，符号位可以作为数据的一部分参加运算而不用单独处理；运算的结果将直接得到两数之和的补码；符号位有进位也只要丢弃即可。这样的运算规则十分方便，这也是补码在计算机内大量使用的原因。

2）补码减法

由于减去一个数就是加上这个数的负数，

∴ $[x-y]_{补}=[x+(-y)]_{补}=[x]_{补}+[-y]_{补}$ （mod 2） （式 2-12）

从$[y]_{补}$求$[-y]_{补}$的法则：当已知$[y]_{补}$要求$[-y]_{补}$时，只要将$[y]_{补}$连同符号位“取反且最低位加 1”即可。

由此可见，补码定点减法和补码定点加法在本质上是相同的，因此，减法运算可以转换成加法运算，使用同一个加法器电路，无需再配减法器，从而可以简化计算机的设计。

3）溢出及其判断

在计算机中，由于机器码的位数是有限的，所以数的表示范围也是有限的。如果两数进行加减运算之后的运算结果超出了给定的取值范围，这就称为溢出。在定点数运算中，正常情况下溢出是不允许的。

两个正数相加，结果大于机器所能表示的最大正数，称为正溢。而两个负数相加，结果小于机器所能表示的最小负数，称为负溢。

溢出的判定方法一般有单符号位法和双符号位法两种，我们以下只讨论双符号位法。

双符号位法用两个符号位表示一个数据，由于有两个符号位，所以相加时是模 4 的相加运算。

用双符号位法进行溢出判断的方法是：如果两个数相加后，其结果的两个符号位一致（00 或 11），则没有发生溢出；如果两个符号位不一致（10 或 01），则发生溢出，具体来说，两个符号位为 01 时为正溢，10 时为负溢；不论溢出与否，运算结果的最高符号位始终指示正确的符号。

这样，在双符号位方式下，只要将两个符号位进行异或运算，异或结果为 0 的就表示正常，异或结果为 1 的就表示溢出。由此，机器就可以通过逻辑电路自动检查出这种溢出，并进行相应的中断处理。

2. 定点乘除法运算

基本运算器的功能只能完成数码的传送、加法和移位，并不能直接完成两数的乘除法运算，但在实际运算中，乘除法却又是计算机的基本运算之一，下面我们讨论实现乘除法运算的方法。

从实现角度来说，实现乘除法运算一般有三种方式：

① 采用软件实现乘除法运算。利用基本运算指令，编写实现乘除法的循环子程序。这种方法所需的硬件最简单，但速度最慢。

② 在原有的基本运算电路的基础上，通过增加左右移位和计数器等逻辑电路来实现乘除法运算，同时增加专门的乘除法指令。这种方式的速度比第一种方式快。

③ 自从大规模集成电路问世以来，高速的单元阵列乘除法器应运而生，出现了各种形式的流水式阵列乘除法器，它们属于并行乘除法器，也有专门的乘除法指令。这种方法依靠硬件资源的重复设置来赢得乘除运算的高速，是三种方式中速度最快的一种。

从编码角度考虑，由于乘除法结果的符号位确定比较容易，运算结果的绝对值和参加运算的数据的符号无关，所以用原码实现也很简单，但在现代计算机中一般还是采用补码进行乘除法运算。

3. 逻辑运算

在计算机中，运算器除了要进行加、减、乘、除等算术运算外，还要完成各种逻辑运算。参加逻辑运算的数据称为逻辑数，是不带符号位的二进制数，或者不用考虑是否有符号位以及数据格式，只是把它当成一种简单的数字“0”和“1”的组合。通常用“1”表示逻辑真，用“0”表示逻辑假。

利用逻辑运算可以进行两个数的比较，或者从某个数中选取某几位等操作。由于在文本、图片、声音等非数值数据中有着广泛的应用，因此逻辑运算也是一种非常重要的运算。

计算机中的逻辑运算主要包括逻辑非、逻辑与、逻辑或、逻辑异或等 4 种运算。

1）逻辑非运算（NOT）

逻辑非运算又称为取反运算，就是对某个操作数的各位按位取反，使每一位 0 变成 1，1 变成 0。逻辑非运算的运算符一般写成“ ¯ ”。

设 $x=x_0x_1x_2\ldots x_n$，则逻辑非标记为：

$$\bar{x} = \bar{x}_0\bar{x}_1\bar{x}_2\ldots\bar{x}_n$$

2）逻辑与运算（AND）

逻辑与运算也叫逻辑乘，表示两个操作数相同位的数据进行按位“与”运算，两个都是 1 则结果为 1，两个中只要有 1 个为 0 结果就是 0。逻辑与运算的运算符一般写成“∧”或“•”。

逻辑与的特点是：对任何数据逻辑与 0 都会变成 0，而逻辑与 1 则保持原有数据不变。所以在实际应用中，如果需要对一个数据的某几位清 0（其他位保持不变）时，常常会用到逻辑与运算。

3）逻辑或运算（OR）

逻辑或运算也叫逻辑加，表示两个操作数相同位的数据进行按位“或”运算，两个都是 0 则结果为 0，两个中只要有 1 个为 1 结果就是 1。逻辑或运算的运算符一般写成“∨”或“+”。

逻辑或的特点是：对任何数据逻辑或 1 都会变成 1，而逻辑或 0 则保持原有数据不变。所以在实际应用中，如果需要对一个数据的某几位置 1（其他位保持不变）时，常常会用到逻辑或运算。

4）逻辑异或运算（XOR）

逻辑异或运算又称按位加，表示两个操作数相同位的数据进行按位“模 2 加”运算，若两个都相同则结果为 0，若两个不同则结果为 1。逻辑异或运算的运算符一般写成“⊕”。

逻辑异或的特点是：对任何数据逻辑异或 1 都会取反，而逻辑异或 0 则保持原有数据不变。所以在实际应用中，如果需要对一个数据的某几位取反（其他位保持不变）时，常常会用到逻辑异或运算。

逻辑异或还有一个特点，就是对一个数连续进行两次逻辑异或，该数就会恢复到原来的状态，这一特点在一些需要数据可恢复的操作中是很有用的。

2.4.2 定点运算器

运算器是数据的加工处理部件，是 CPU 的重要组成部分。

运算器的主要功能是对数据进行加工处理，包括对数值数据的算术运算，如执行加、减、乘、除运算，变更数据的符号等，同时也包括对各种数据的逻辑运算，如进行与、或、非等运算。因此，实现对数据的算术和逻辑运算是运算器最重要的功能。

运算器通常由算术逻辑单元（Arithmetic Logic Unit，ALU）、寄存器、数据总线和其他逻辑部件组成 。ALU 是具体完成算术与逻辑运算的单元，是运算器的核心，由加法器及其他逻辑运算单元组成。寄存器用于存放参与运算的操作数，其中的累加器是一个特殊的寄存器，除了可以存放操作数外，还用于存放中间结果和最后结果。数据总线用于完成运算器内部的数据传送。

无论计算机的功能、规模有多大差异，其运算器的基本结构总是包括以下几个部分：

⑴能实现算术和逻辑运算的功能部件 ALU。

⑵存放待加工信息或加工后信息的通用寄存器组。

⑶按操作要求控制数据输入的部件，如多路开关或数据锁存器，可以接收来自外部设备或存储器的数据，也可以暂存通用寄存器的数据。

⑷按操作要求控制数据输出的部件，如输出移位开关和多路开关，可以将 ALU 的输出结果根据要求进行移位，并经总线送往其他部件，或作为中间结果送给通用寄存器，以便作为 ALU 的输入进行下一次运算。

⑸计算机与其他部件进行信息传输的总线，以及总线接收器和发送器。总线接收器和发送器通常是由三态门构成的。

下面介绍运算器的各个组成部分：

1. ALU

算术逻辑单元 ALU，不仅具有多种算术运算和逻辑运算的功能，而且具有先行进位逻辑，从而能实现高速运算。

特定 ALU 所支持的算术运算，可能仅局限于加法和减法，也可能包括乘法、除法，甚至三角函数和平方根。有些 ALU 只支持整数，而其他 ALU 则可以使用浮点来表示有限精度的实数。但是，能够执行最简单运算的任何计算机，都可以通过编程，把复杂的运算分解成它可以执行的简单步骤。所以，任何计算机都可以通过编程来执行任何算术运算，如果其 ALU 不能从硬件上直接支持，则该运算将用软件方式实现，但需要花费较多的时间。

超标量（Superscalar）计算机包含多个 ALU，可以同时处理多条指令。图形处理器和具有单指令流多

数据流 SIMD 和多指令流多数据流 MIMD 特性的计算机，通常提供可以执行矢量和矩阵算术运算的 ALU。

2. 数据总线

除了运算器的核心部件 ALU 之外，运算器中还包括各种寄存器、多路选择器、移位器等部件，它们之间的数据传送非常频繁。为了减少运算器内部的数据传送线同时便于控制，通常将一些寄存器之间的数据传送通路加以归并，组成总线结构，使不同来源的信息在此总线上分时传送。

根据总线所处的位置，总线可以分为内部总线和外部总线。内部总线是指 CPU 内各部件的连线；外部总线是指系统总线，即 CPU 与存储器、I/O 系统之间的连线。运算器内部的总线属于内部总线。

按照总线的逻辑结构，总线可以分为单向传送总线和双向传送总线。所谓单向传送总线，就是信息只能向一个方向传送，传送地址信息或控制信息的总线通常是单向传送总线；而双向传送总线，就是信息可以向两个方向传送，数据总线一般是双向传送总线，既可以发送数据，又可以接收数据。有时为了简化数据线的管理，也可以有只用于发送数据或者只用于接收数据的数据总线。

3. 运算器的基本结构形式

运算器的设计，主要是围绕 ALU 和寄存器同数据总线之间如何传送操作数和运算结果来进行的。在决定方案时，需要考虑数据传送的方便性和操作速度，在微型机和单片机中还要考虑在硅片上制作总线的工艺。

运算器的基本结构形式如图 2-8 所示。

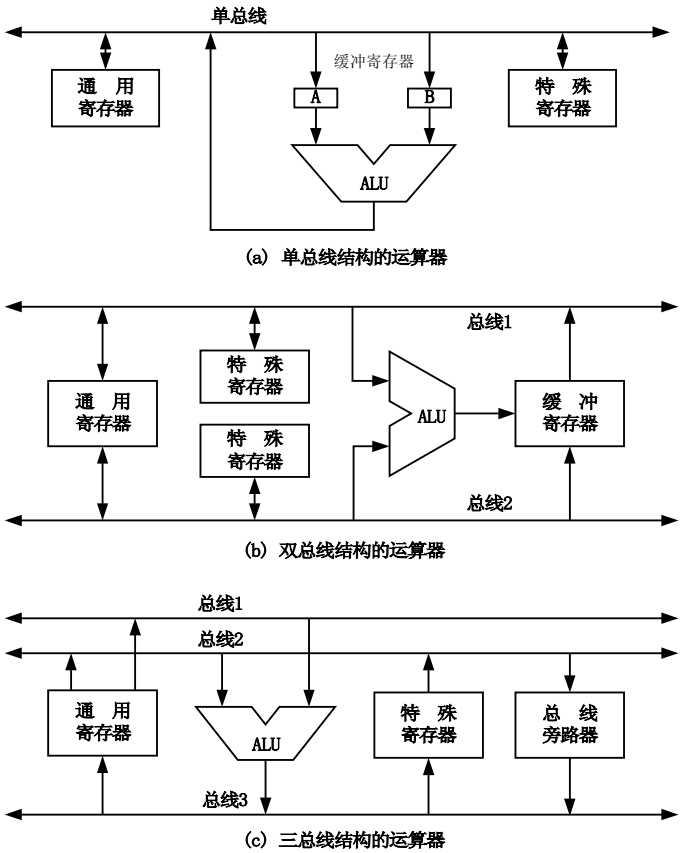


图 2-8 运算器的三种基本结构

1）单总线结构的运算器

单总线结构的运算器如图 2-8(a)所示，所有部件都连接到同一条总线上，所以数据可以在任何两个寄存器之间，或者在任一个寄存器和 ALU 之间传送。

这种总线结构的运算器内部只有一组数据总线，优点是总线的控制电路比较简单，有利于提高大规模集成电路的集成度。但是由于总线的分时性，同一时间内只能有一个操作数放到总线上。把两个操作数输入到 ALU，需要分两次来做，而且还需要 A、B 两个缓冲寄存器。因此，这种结构的主要缺点是操作速度较慢，

执行一个双操作数的运算，一般需要三次数据传送，花费三个单元时间。

以加法运算为例，首先把第一个操作数通过总线送往缓冲寄存器 A 暂存；再把第二个操作数通过总线送往缓冲寄存器 B 暂存；最后一步是将两个操作数从缓冲寄存器 A 和 B 送往 ALU，并将运算结果通过总线送往相应的通用寄存器。

2）双总线结构的运算器

双总线结构的运算器如图 2-8(b)所示，有两条总线同时连接 ALU 的两个输入端，可以与通用寄存器双向传送数据。为了防止总线冲突，在 ALU 的输出端设置缓冲寄存器。特殊寄存器分为两组，分别与两条总线相连。

在这种结构中，两个操作数同时加到 ALU 进行运算，只需要一次操作控制，而且马上就可以得到运算结果。ALU 的输出不能直接加到总线上去，这是因为，当形成操作结果的输出时，两条总线都被输入数占据，因而必须在 ALU 输出端设置缓冲寄存器。总之，双总线结构运算器比单总线结构运算器速度快，执行一个双操作数的运算，一般只需要两次数据传送，花费两个单元时间。当然，总线控制电路要相对复杂些。

还是以加法运算为例，首先将两个操作数分别送往总线 1 和总线 2，由于总线 1 和总线 2 直接连接 ALU 的输入端，因此可以直接运算，并把运算结果送往 ALU 输出端的缓冲寄存器；第二步把缓冲寄存器中存放的运算结果通过总线 1 送往相应的通用寄存器。

3）三总线结构的运算器

三总线结构的运算器如图 2-8(c)所示，共有三条总线。由于三总线结构的总线控制电路复杂，为了简化设备，每条数据总线都设计成单向总线。寄存器可以输出数据至总线 1 和总线 2，而总线 1 和总线 2 的数据分别送往 ALU 的两个输入端；与之相反，ALU 的输出端只能送往总线 3，而寄存器也只能接收来自总线 3 的数据。

在三总线结构中，ALU 的两个输入端分别由两条总线供给，而 ALU 的输出则与第三条总线相连。显然，三总线结构运算器的速度是最快的，双操作数运算可以一步完成。当然，总线控制也是最复杂的。

仍然以加法运算为例，可以将两个操作数通过总线 1 和总线 2 分别送往 ALU，同时将 ALU 的运算结果通过总线 3 送往相应的通用寄存器。

因为在三总线结构运算器中，每条总线都是单向传输数据，因此给寄存器之间的数据传送带来一定困难。为了避免寄存器之间的数据传送也要用到 ALU 从而降低速度，在三总线结构运算器中还专门设置了总线旁路器，只要开通总线旁路器，总线 2 的数据就可以直接送往总线 3 而不需要经过 ALU，这样就可以大大提高寄存器之间的数据传送速度。

4. 寄存器

寄存器一般指的是通用寄存器，多通用寄存器是现代计算机系统的结构特点之一。利用多个寄存器，可以存放运算过程的中间结果，使存取数据的速度提高，从而缩短指令周期，加快机器速度。

通用寄存器是指这些寄存器的用途广泛，除了用于存放操作数和运算结果外，还可以作为变址寄存器存放变址值，作为堆栈指示器存放堆栈指针等，可以被程序员直接使用。除了通用寄存器外，还有一些专用寄存器，它们对于程序员而言是透明的，不能直接使用，硬件系统在完成某项工作时会用到相关的专用寄存器。

除此之外，累加器是运算器中与 ALU 直接相连、使用频繁的一种寄存器，每次运算的操作数或运算的中间结果大多存放在累加器中。所以，累加器是与很多指令都相关的通用寄存器。

2.5 浮点运算和浮点运算器

2.5.1 浮点运算

1. 浮点数的加减法运算

设有两个浮点数 x 和 y，它们的规格化表示分别为：

$$x = 2^{Ex} \times Mx$$

$y = 2^{Ey} \times My$

其中 Ex 和 Ey 分别为数 x 和 y 的阶码，而 Mx 和 My 分别为数 x 和 y 的尾数。这样，浮点数加、减法运算的规则可以表示为：

$$x \pm y = 2^{E_x} \times M_x \pm 2^{E_y} \times M_y = \begin{cases} (M_x \times 2^{E_x - E_y} \pm M_y) \times 2^{E_y} & (Ex \le Ey) \\ (M_x \pm M_y \times 2^{E_y - E_x}) \times 2^{E_x} & (Ex > Ey) \end{cases} \quad （式 2-13）$$

之所以要这样分类，是因为要遵循“小阶向大阶靠拢”的对阶原则，下面会有详细的解释。根据公式，可总结出浮点数运算的几个步骤：

1）0 操作数检查

浮点数的运算过程比较复杂，如果能判断出两个操作数中有一个为 0，那么运算结果马上可知，而不必进行后续的一系列操作，以节省运算时间。

2）对阶

两个浮点数相加减，首先要看它们的阶码是否相同，即小数点位置是否对齐。如果阶码相同，则表示小数点位置是对齐的，尾数就可以直接进行加减运算；反之若两数阶码不同，则表示小数点位置没有对齐，不能直接进行加减运算。此时，必须通过“对阶”过程使两数的阶码相同，也就是使两数的小数点位置对齐。

要对阶就要改变两数中一个数的阶码，表面上看来改变哪一个都可以。由于随着阶码的改变，尾数也要做相应的移动才能使浮点数据的值保持不变，所以如果阶码变大，尾数要右移；阶码变小，尾数要左移。尾数的左右移都会造成有效数据的移出与丢失，但是右移丢失的是最低有效位，而左移丢失的却是最高有效位。显然，右移更能减小数据误差。所以，对阶必须遵循“小阶向大阶靠拢”的原则。两数中阶码较小的那个数的阶码要变大，变成和另一个数的阶码一样，而这个数的尾数要作相应的右移，右移多少取决于阶码变大多少。阶码每增加 1，尾数要相应右移 1 位，相当于小数点左移 1 位。

对阶时，一般首先求出两数阶码之差，即

$\triangle E = Ex - Ey$

如果△E=0，说明两数阶码相等，无需对阶；如果△E>0，表示 Ex>Ey，Ey 要向 Ex 靠拢，其尾数 My 要做相应右移；如果△E<0，表示 Ex<Ey，Ex 要向 Ey 靠拢，其尾数 Mx 要做相应右移。

3）尾数相加

对阶完成后，表示两数的小数点已经对齐，可以直接进行尾数的加减运算。无论加法运算还是减法运算，都与前述的定点数补码运算一样，按加法进行操作。要注意的是，相加过程中没有溢出，也就是对于定点数来说是溢出的结果，对于浮点数尾数相加来说是很正常的事情，所以我们常用双符号位表示尾数。

4）结果规格化

尾数相加完成之后，还需要进行规格化的判定，如果不满足规格化要求，则要对结果作规格化处理。尾数的规格化处理有两种情况：

如果尾数相加结果的两个符号位数据不相等，表明运算结果的尾数的绝对值大于 1，因此要“向右规格化”。由于尾数相加的绝对值不可能超过 2，因此向右规格化肯定是尾数右移 1 位，阶码加 1。

如果尾数相加结果的符号位与数据最高位相等，表示数据没有规格化，尾数要“向左规格化”，即尾数左移 n 位，阶码相应减 n。

5）舍入处理

在对阶和向右规格化的过程中，尾数都要向右移位，这样尾数的低位部分可能会丢失，从而造成一定的误差。为了减少误差，要进行舍入处理，常用的舍入方法有两种：

⑴ 0 舍 1 入

“0 舍 1 入”，就是指尾数右移时被丢掉的数据的最高位如果是 0，那就舍去；如果是 1，就在尾数的最低位加上“1”。这种方法，实际上类似于我们平时所说的“四舍五入”。

这种方法的优点是每次舍入产生的误差小，误差控制在 $2^{-(n+1)}$ 范围内，而且也不会造成误差的累积；但缺点是要进行一次“加 1”运算，特殊情况下还有可能造成再次“向右规格化”的现象。

⑵恒置 1

“恒置 1”，就是指尾数右移时，只要发生低位数据的丢失，尾数的最低位就被设定为 1。

这种方法每次舍入所产生的误差比“0 舍 1 入”要大一点，误差控制在 2^{-n} 范围内，而且也不会造成误差的累积。其关键特点是舍入处理时无需进行加法运算，所以速度快，也不会造成再次“向右规格化”的现象。

6）溢出处理

尾数相加的溢出不是真正的溢出，可以借由向右规格化作出调整，那么，浮点数的运算会不会产生溢出？什么时候才是真正的溢出呢？

当浮点数在做向左或向右规格化的过程中，阶码也会做相应的调整，也就是说，阶码可能要加上 1（向右规格化）或者减去 n（向左规格化，n 为尾数左移的位数）。显然，这些时候都有可能产生阶码溢出现象。

如果阶码减去 n 发生阶码溢出，也就是发生阶码的下溢，表示运算结果的精度超出了该浮点数可以表示的范围，也就是运算结果趋近于 0。在这种情况下，机器一般认为运算结果就是 0；如果阶码加上 1 发生阶码溢出，也就是发生阶码的上溢，表示数据超出了浮点数可表示的范围，一般认为是+∞或-∞（依赖于尾数的正负）。这种情况才是真正的溢出，机器的溢出标志会被置“1”。

综上所述，浮点数运算真正的溢出，是指在尾数相加的时候发生尾数上溢，并在向右规格化的时候使阶码也发生上溢。

2. 浮点数的乘除法运算

设有两个浮点数 x 和 y，它们的规格化表示分别为：

$x = 2^{Ex} \times Mx$

$y = 2^{Ey} \times My$

那么，两数的乘积为

$$x \times y=(2^{Ex} \times Mx) \times (2^{Ey} \times My)=2^{Ex+Ey} \times (Mx \times My) \qquad \text{（式 2-14）}$$

也就是说，两个浮点数相乘的结果就是它们的阶码相加，尾数相乘。

同理，两数相除的结果为

$$x \div y=(2^{Ex} \times Mx) \div (2^{Ey} \times My)=2^{Ex-Ey} \times (Mx \div My) \qquad \text{（式 2-15）}$$

也就是说，两个浮点数相除的结果就是它们的阶码相减，尾数相除。

这样，浮点数的乘除运算就可以转化成定点数的加、减、乘、除运算。

无论是浮点数的乘法还是除法，都可以分为下面 4 个运算步骤：

①0 操作数检查

②阶码加/减操作

③尾数乘/除操作

④规格化处理与舍入

2.5.2 浮点运算流水线

一条具体的指令执行过程，通常可以分为五个部分：取指令、指令译码、取操作数、运算（ALU）、写结果。其中前三步一般由指令控制器完成，后两步则由运算器完成。按照传统的方式，所有的指令均为顺序执行：首先指令控制器工作，完成第一条指令的前三步，然后运算器工作，完成后两步；接着指令控制器工作，完成第二条指令的前三步，然后运算器工作，完成第二条指令的后两步……。很明显，当指令控制器工作时，运算器基本上处于空闲状态，而当运算器工作时，指令控制器又处于空闲状态，造成相当大的资源浪费。一个显而易见的解决方法是，在完成第一条指令的前三步后，指令控制器不等运算器完成第一条指令的后两步，就立即开始第二条指令的操作，对于运算器也是如此，这就形成了一种与工厂中的装配流水线类似的流水线。

为了实现流水，首先必须把输入的任务分割为一系列的子任务，使各子任务能在流水线的各个阶段并发地执行。将任务连续不断地输入流水线，从而实现子任务级的并行。流水处理大幅度地改善了计算机的系

统性能，是在计算机上实现时间并行性的一种非常经济的方法。

在流水线中，原则上要求各个阶段的处理时间都相同。若某一阶段的处理时间较长，势必造成其他阶段的空转等待。因此，对子任务的划分，是决定流水线性能的一个关键因素，它取决于操作部分的效率，所期望的处理速度，以及成本价格等等。

假定作业 T 被分成 k 个子任务，可表达为

$T=\{T_1, T_2, \cdots, T_k\}$

各个子任务之间有一定的优先关系：若 i<j，则必须在 T_i 完成以后，T_j 才能开始工作。各子任务之间具有这种线性优先关系的流水线称为线性流水线。线性流水线的硬件基本结构如图 2-10 所示。

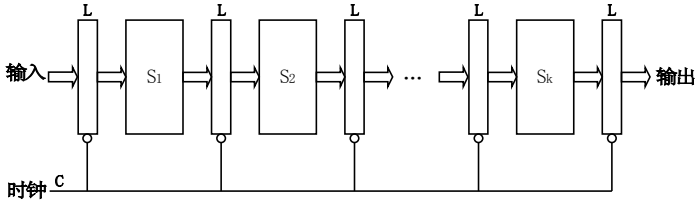


图 2-10 线性流水线的硬件基本结构

图中，处理一个子任务的过程为过程段（S_i）。线性流水线由一系列串联的过程段组成，各个过程段之间设有高速缓冲寄存器（L），以暂时保存上一过程子任务处理的结果。在一个统一的时钟（C）的控制下，数据从一个过程段流向下一个相邻的过程段。

设过程段 S_i 所需的时间为 τ_i，缓冲寄存器的延时为 τ_l，线性流水线的时钟周期定义为：

$$\tau = \max\{\tau_i\} + \tau_l \qquad \text{（式 2-16）}$$

流水线处理的频率 $f=1/\tau$。

在流水线处理中，当任务饱满时，任务源源不断地输入流水线，不论有多少级过程段，每隔一个时钟周期总能输出一个任务。从理论上说，一个具有 k 级过程段的流水线处理 n 个任务需要的时钟周期数为：

$$T_k=k+(n-1) \qquad \text{（式 2-17）}$$

其中 k 个时钟周期用于处理第一个任务。k 个周期后，流水线被装满，剩余的 n-1 个任务只需 n-1 个周期就完成了。

如果用非流水线的硬件来处理这 n 个任务，时间上只能串行进行，则所需时钟周期数为：

$$T_L=n\cdot k \qquad \text{（式 2-18）}$$

将 T_L 和 T_k 的比值定义为 k 级线性流水线的加速比：

$$C_k = \frac{T_L}{T_k} = \frac{n\cdot k}{k+(n-1)} \qquad \text{（式 2-19）}$$

当 n>>k 时，$C_k \rightarrow k$，也就是说，理论上 k 级线性流水线处理几乎可以提高 k 倍速度。但实际上，由于存储器冲突、数据相关等原因，这个理想的加速比不一定能达到。

2.5.3 浮点运算器实例

1. CPU 之外的浮点运算器

8087 是美国 Intel 公司为处理浮点数等数据的算术运算和多种函数计算而设计生产的专用算术运算处理器。由于其算术运算是配合 8086 CPU 进行的，所以 8087 又称为协处理器（Co-processor）。

以下说明 8087 浮点运算器的特点和内部结构。

⑴以异步方式与 8086 并行工作。8087 相当于 8086 的一个 I/O 部件，本身有它自己的指令，但不能单独使用，它只能作为 8086 主 CPU 的协处理器才能运算。如果 8086 从主存读取的指令是 8087 的浮点运算指令，则它们以输出的方式把该指令送到 8087，8087 接受指令后进行译码并执行浮点运算。8087 进行运算期间，8086 可取下一条其他指令予以执行，因而实现了并行工作。如果在 8087 执行浮点运算指令过程中，8086 又取来了一条 8087 指令，则 8087 给出“忙”标志信号加以拒绝，使 8086 暂停向 8087 发送命令。只有待 8087 完成浮点运算而取消“忙”标志信号以后，8086 才可以进行下一次发送操作。

⑵可处理包括二进制浮点数、二进制整数和压缩十进制数串三大类 7 种类型数据，其中浮点数的格式

符合 IEEE-754 标准，有 32 位、64 位、80 位（临时实数）三种。

2. CPU 之内的浮点运算器

奔腾（Pentium）CPU 将浮点运算器包含在芯片内，浮点运算部件采用流水线设计。

指令执行过程分为 8 段流水线。前 4 段为指令预取（DF）、指令译码（D_1）、地址生成（D_2）、取操作数（EX），在 U、V 流水线中完成；后 4 段为执行 1（X_1）、执行 2（X_2）、结果写回寄存器堆（WB）、错误报告（ER），在浮点运算器中完成。奔腾 CPU 内部的主要流水线是“U-Pipe”，能应付所有的 x86 指令；另一条流水线称为“V-Pipe”，只能对一些简单的整数指令和一个浮点运算指令“FXCH”解码。一般情况下，由 U 流水线完成一条浮点数操作指令。

浮点部件内有浮点专用加法器、乘法器和除法器，有 8 个 80 位寄存器组成的寄存器堆，内部的数据总线为 80 位宽，可支持 IEEE-754 标准的单精度和双精度格式的浮点数，还使用一种称为临时实数的 80 位浮点数。

第3章 存储系统

3.1 存储器概述

存储器是计算机系统中的记忆设备，用来存放程序和数据。现代计算机系统都是以存储器为中心，计算机若要开始工作，必须先把有关程序和数据装到存储器中，程序才能开始运行。在程序执行过程中，CPU所需的指令要从存储器中取出，运算器所需的原始数据要从存储器中取出，运算结果必须在程序执行完毕之前全部写到存储器中，各种输入输出设备也直接与存储器交换数据。因此，在计算机运行过程中，存储器是各种信息存储和交换的中心。

3.1.1 基本概念

构成存储器的存储介质，目前主要采用半导体器件和磁性材料。存储器中最小的存储单位可以是一个双稳态半导体电路或一个CMOS晶体管或磁性材料的存储元，它可存储一个二进制代码。这个二进制代码位是存储器中最小的存储单位，称为一个存储位或存储元，若干个存储位可以组成一个存储单元，许多存储单元可以组成一个存储器，这些存储单元的集合也称为存储体。

我们知道，存储器是用来存储程序和数据的，而程序和数据都是用二进制来表示的。二进制数中每一位（bit）“0”或“1”就是由存储器的一个存储位来存储的。存储器的容量以字节（Byte，简称为B）为单位表示，比如640KB、1MB、32MB、1GB等，其相互关系如下：

1KB=1024B，1MB=1024KB，1GB=1024MB，1TB=1024GB

CPU向存储器送入和从存储器取出数据信息时，通常采用较大的信息单位“字”（Word）来工作。

总而言之：

⑴ 位（bit）是二进制数的最小单位，也是数字计算机的最小信息单位，通常用“b”表示。

⑵ 字节（Byte）包含8个bit，通常用“B”表示。存储器容量一般都是以字节为单位的。

⑶ 字（Word）由若干个字节组成。至于一个字到底等于多少个字节，则取决于计算机的字长，即计算机一次所能处理的数据的最大位数。例如，对于32位机，1 Word = 4 Bytes = 32 bits。

3.1.2 存储器的分类

根据存储材料的性能及使用方法的不同，存储器可以有各种不同的分类方法。

1．按存储介质分

存储介质必须满足两个基本要求：

⑴ 必须有两个明显区别的状态，分别表示二进制代码0和1；

⑵ 两个物理状态的改变速度要快，它直接影响存储器的读写速度。

目前使用的存储介质主要是半导体器件、磁存储介质和光存储介质。用半导体器件组成的存储器称为半导体存储器，如计算机主存；用磁性材料做成的存储器称为磁表面存储器，它通过磁头和磁记录介质的相对运动完成读出和写入，如磁盘、磁带；利用激光技术在光存储介质上写入和读出信息的存储器称为光盘存储器，如只读型光盘（CD-ROM、DVD-ROM）、可读写光盘（CD-RW，DVD±RW）、一次性光盘等。

2．按存取方式分

如果任何存储单元的内容都能被随机存取，且存取时间和存储单元的物理位置无关，则这种存储器称为随机存储器，如半导体存储器；如果存储单元的内容只能按某种顺序来存取，存取时间与存储单元的物理位置有关，取决于访问存储单元的地址顺序，则这类存储器称为顺序存储器，如磁带存储器。与顺序存储器相比，随机存储器的存取速度快得多，但每一位的价格也要高很多。

3．按存储器的读写功能分

有些半导体存储器中存储的内容是固定不变的，只能读出而不能写入，通常用来存放固定不变的程序、汉字字型库等，在制造芯片时由厂家预先写入，这类半导体存储器称为只读存储器（ROM）；既能读出内容又能写入新内容的半导体存储器称为随机读写存储器（RAM），用来存放正在执行的程序和正在访问的数据。

4．按信息的可保存性分

断电后信息就消失的存储器称为非永久记忆存储器，如半导体存储器 RAM；断电后仍能保存信息的存储器称为永久记忆存储器，如磁介质存储器、光盘存储器。

5．按在计算机系统中的作用分

根据在计算机系统中所起的作用，存储器可分为主存储器、辅助存储器、高速缓冲存储器、控制存储器等。

3.1.3 存储器的分级结构

一个存储器的性能通常用速度、容量、价格三个主要指标来衡量。计算机对存储器的要求是容量大、速度快、成本低，需要尽可能地同时兼顾这三方面的要求。但是一般来讲，存储器速度越快，价格也越高，因而也越难满足大容量的要求。目前通常采用多级存储器体系结构，使用高速缓冲存储器、主存储器和外存储器，如图 3-1 所示。

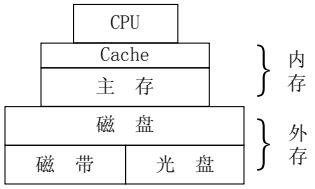


图 3-1 存储系统的分级结构

CPU 能直接访问的存储器称为内存储器（简称内存），包括高速缓冲存储器和主存储器。CPU 不能直接访问的存储器称为外存储器（简称外存，也叫辅助存储器），外存的信息必须调入内存才能被 CPU 使用。

高速缓冲存储器（Cache）是计算机系统中的一个高速、小容量的半导体存储器，它位于高速的 CPU 和低速的主存之间，用于匹配两者的速度，达到高速存取指令和数据的目的。和主存相比，Cache 的存取速度快，但存储容量小。

主存储器，简称主存，是计算机系统的主要存储器，用来存放计算机正在执行的大量程序和数据，主要由 MOS 半导体存储器组成。

外存储器，简称外存，是计算机系统的大容量辅助存储器，用于存放系统中的程序、数据文件及数据库。与主存相比，外存的特点是存储容量大，位成本低，但访问速度慢。目前，外存储器主要有磁盘存储器、磁带存储器和光盘存储器。

由 Cache 和主存储器构成的 Cache－主存系统，其主要目标是利用与 CPU 速度接近的 Cache 来高速存取指令和数据以提高存储器的整体速度，从 CPU 角度看，这个层次的速度接近 Cache，而容量和每一位的价格则接近主存；由主存和外存构成的虚拟存储器系统，其主要目的是增加存储器的容量，从整体上看，其速度接近于主存的速度，其容量则接近于外存的容量。计算机存储系统的这种多层次结构，很好地解决了容量、速度、成本三者之间的矛盾。这些不同速度、不同容量、不同价格的存储器，用硬件、软件或软硬件结合的方式连接起来，形成一个系统。这个存储系统对应用程序员而言是透明的，在应用程序员看来它是一个存储器，其速度接近于最快的那个存储器，存储容量接近于容量最大的那个存储器，单位价格则接近最便宜的那个存储器。

3.1.4 半导体存储器芯片

半导体存储器芯片按照读写功能可分为随机读写存储器（Random Access Memory，RAM）和只读存储器（Read Only Memory，ROM）两大类。RAM 可读可写，断电时信息会丢失；ROM 中的内容只能读出，不能写入，信息可永久保存，不会因为断电而丢失。

1．随机读写存储器

目前广泛使用的半导体随机读写存储器是 MOS 半导体存储器，按保存数据的机理分为静态存储器（Static RAM，SRAM）和动态存储器（Dynamic RAM，DRAM）。

1）静态存储器（SRAM）

利用双稳态触发器来保存信息，只要不断电信息就不会丢失。

静态存储器的集成度低，成本高，功耗较大，通常作为 Cache 的存储体。

2）动态存储器（DRAM）

利用 MOS 电容存储电荷来保存信息，使用时需要不断给电容充电才能保持信息。

动态存储器电路简单，集成度高，成本低，功耗小，但需要反复进行刷新（Refresh）操作，工作速度较慢，适合作为主存储器的主体部分。

刷新操作：为防止存储的信息电荷泄漏而丢失信息，由外界按一定规律不断地给栅极进行充电，补足栅极的信息电荷。

DRAM 工作时必须要有刷新控制电路，操作比较复杂。由于要不间断地进行刷新，故称这种存储器为动态存储器。动态 MOS 存储器主要采用“读出”的方式进行刷新，依次读出存储器的每一行，就可完成对整个 DRAM 的刷新。

DRAM 存储器的刷新需要有硬件线路的支持，这些控制线路可以集成在一个半导体芯片上，形成 DRAM 控制器。借助于 DRAM 控制器，可以把 DRAM 当作 SRAM 一样使用，从而为系统设计带来很大的方便。

3）增强型 DRAM（EDRAM）

EDRAM 芯片是在 DRAM 芯片上集成一个高速小容量的 SRAM 芯片而构成的，这个小容量的 SRAM 芯片起到高速缓存的作用，从而使 DRAM 芯片的性能得到显著改进。

当 CPU 从主存 DRAM 中读取数据时，会将包含此数据的整个数据块都写入高速缓存 SRAM 内，下次读取连续地址数据时，CPU 就可以从这个 SRAM 中直接取用，而不必到较慢的 DRAM 中读取，如此即可加快 CPU 的存取速度。

将由若干 EDRAM 芯片组成的存储模块做成小电路插件板形式，就是目前普遍使用的内存条。

2．只读存储器

只读存储器 ROM 是一种存储固定信息的存储器，其特点是在正常工作状态下只能读取数据，不能即时修改或重新写入数据。

只读存储器电路结构简单，且存放的数据在断电后不会丢失，特别适合于存储永久性的、不变的程序代码或数据（如常数表、函数、表格和字符等），计算机中的自检程序就是固化在 ROM 中的。

ROM 的最大优点是具有不易失性。

只读存储器有不可重写只读存储器（MROM、PROM）和可重写只读存储器（EPROM、EEPROM、闪速存储器等）两大类。

1）不可重写只读存储器

⑴掩模只读存储器（MROM）

掩模只读存储器，又称固定 ROM。这种 ROM 在制造时，生产厂家利用掩模（Mask）技术把信息写入存储器中，使用时用户无法更改，适宜大批量生产。

掩模只读存储器可分为二极管 ROM、双极型三极管 ROM 和 MOS 管 ROM 三种类型。

⑵可编程只读存储器（PROM）

可编程只读存储器（Programmable ROM，简称 PROM），是可由用户一次性写入信息的只读存储器，是在 MROM 的基础上发展而来的。

PROM 的缺点是用户只能写入一次数据，一经写入就不能再更改。

2）可重写只读存储器

这类 ROM 由用户写入数据（程序），当需要变动时还可以进行修改，使用起来比较方便。可重写 ROM 有紫外线擦除 EPROM、电擦除 EEPROM 和闪速存储器 Flash ROM 三种类型。

⑴光擦可编程只读存储器（EPROM）

EPROM 的特点是其中的内容可以用特殊的装置进行擦除和重写。EPROM 出厂时，其存储内容为全“1”，用户可根据需要改写为“0”，当需要更新存储内容时，可将原存储内容擦除（恢复为全“1”），以便写入新的内容。

EPROM 一般是将芯片置于紫外线下照射 15~20 分钟左右，以擦除其中的内容，然后用专用的设备（EPROM 写入器）将信息重新写入，一旦写入则相对固定。

在闪速存储器大量应用之前，EPROM 常用于软件开发过程中。

⑵电擦可编程只读存储器（EEPROM 或 E^2PROM）

用紫外线擦除 EPROM 的操作复杂，速度很慢。EEPROM 可以用电气方法将芯片中的存储内容擦除，擦除时间较快，甚至可以在联机状态下操作。

EEPROM 既可使用字擦除方式又可使用块擦除方式，使用字擦除方式可擦除一个存储单元，使用块擦除方式可擦除数据块中所有存储单元。

⑶闪速存储器（Flash ROM）

闪速存储器 Flash ROM 是 20 世纪 80 年代中期出现的一种块擦写型存储器，是一种高密度、非易失性的读/写半导体存储器，它突破了传统的存储器体系，改善了现有存储器的特性。

Flash ROM 中的内容或数据不像 RAM 一样需要电源支持才能保存，但又像 RAM 一样具有可重写性。在某种低电压下，其内部信息可读不可写，类似于 ROM，而在较高的电压下，其内部信息可以更改和删除，类似于 RAM。

Flash ROM 可以用软件在 PC 机中改写或在线写入，信息一旦写入即相对固定。因此，在 PC 机中可用于存储主板的 BIOS 程序。由于能进行改写，便于用户自行升级 BIOS，但这也给病毒以可乘之机，著名的 CIH 病毒正是利用这个特点来破坏 BIOS，从而导致整个系统瘫痪的。

另外，由于单片存储容量大，易于修改，Flash ROM 也常用于数码相机和 U 盘中，因其具有低功耗、高密度等特点，且没有机电移动装置，特别适合于便携式设备，成为替代磁盘的一种理想工具。

3.2 主存储器

3.2.1 主存储器的技术指标

主存储器是 CPU 能直接访问的存储器，由随机读写存储器 RAM 和只读存储器 ROM 组成，能快速地进行读或写操作。衡量一个主存储器性能的技术指标主要有存储容量、存取时间、存储周期和存储器带宽。

1．存储容量

在一个存储器中可以容纳的存储单元的总数称为存储容量（Memory Capacity）。

存储单元可分为字存储单元和字节存储单元。所谓字存储单元，是指存放一个机器字的存储单元，相应的单元地址称为字地址；而字节存储单元，是指存放 1 个字节（8 位二进制数）的存储单元，相应的地址称为字节地址。如果一台计算机中可编址的最小单位是字存储单元，则该计算机称为按字编址的计算机；如果一台计算机中可编址的最小单位是字节存储单元，则该计算机称为按字节编址的计算机。一个机器字可以包含数个字节，所以一个字存储单元也可包含数个字节存储单元。

为了描述方便和统一，目前大多数计算机采用字节为单位来表征存储容量。在按字节寻址的计算机中，存储容量的最大字节数可由地址码的位数来确定。例如，一台计算机的地址码为 n 位，则可产生 2^n 个不同的地址码，如果地址码被全部利用，则其最大容量为 2^n 个字节。一台计算机设计定型以后，其地址总线、地址译码范围也已确定，因此其最大存储容量是确定的，而实际配置存储容量时，只能在这个范围内进行选择，通常情况下主存储器的实际存储容量远远小于理论上的最大容量。一般而言，存储器的容量越大，所能存放的程序和数据就越多，计算机的解题能力就越强。

存储容量的单位通常用 KB、MB、GB 来表示，K 代表 2^{10}，M 代表 2^{20}，G 代表 2^{30}。

1KB=1024B，1MB=1024KB，1GB=1024MB

2．存取时间

存取时间即存储器访问时间（Memory Access Time），是指启动一次存储器操作到完成该操作所需的时间。

具体地说，读出时为取数时间，写入时为存数时间。取数时间就是指存储器从接受读命令到信息被读出并稳定在存储器数据寄存器中所需的时间；存数时间就是指存储器从接受写命令到把数据从存储器数据

寄存器的输出端传送到存储单元所需的时间。

3．存储周期

存储周期又称为访问周期，是指连续启动两次独立的存储器操作所需间隔的最小时间，它是衡量主存储器工作性能的重要指标。存储周期通常略大于存取时间。

4．存储器带宽

存储器带宽是指单位时间里存储器所存取的信息量，是衡量数据传输速率的重要指标，通常以位/秒（bps，bit per second）或字节/秒（Byte/s）为单位。

例如，总线宽度为32位，存储周期为250ns，则

存储器带宽 = 32b/250ns = 128Mb/s = 128Mbps

存取时间、存储周期、存储器带宽都反映了主存的速度指标。

3.2.2 主存储器的基本组成

主存储器由存储体、寻址系统、存储器数据寄存器、读写系统及控制线路等组成，如图3-3所示。

EMBED Visio.Drawing.11

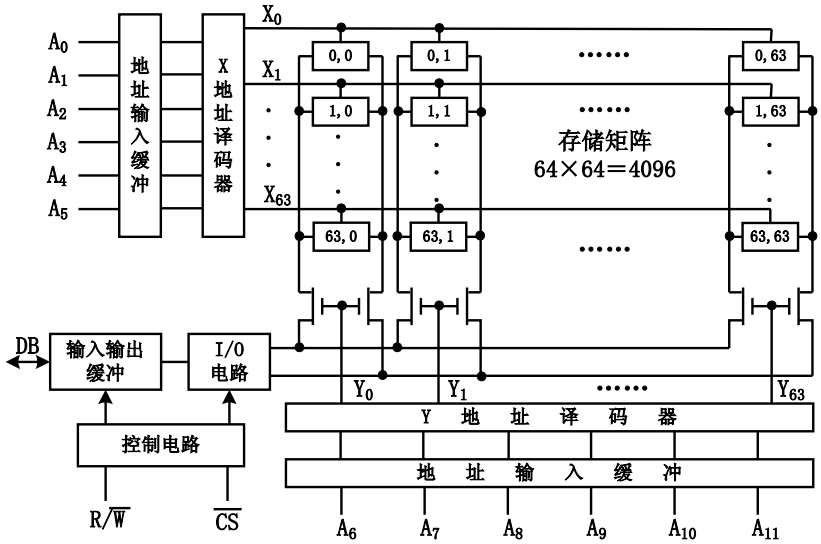


图3-3 主存储器的基本组成

1．存储体

存储体是一个由存储单元按照一定规则排列起来的存储阵列。存储体是存储器的核心，是存储信息的实体。

2．寻址系统

寻址系统就是读出和写入信息的地址选择机构，包括存储器地址寄存器（MAR）和地址译码器。

地址译码器接收来自地址寄存器的n位地址，经译码后产生2^n个地址选择信号，并从2^n个单元中选出一个单元。通常用X选择线（行线）和Y选择线（列线）的交叉来选择所需要的单元。

存储器地址寄存器MAR具有地址缓冲功能，可使CPU和主存的速度都得到充分发挥和提高。MAR从功能上看属于主存，但在一些微型机中常被放在CPU内，并可兼作别用，在速度要求较高的计算机中，CPU与主存中都设有地址寄存器。

3．存储器数据寄存器（MDR）

一般把存储器数据寄存器MDR作为存储器接收输入数据和发出输出数据用的数据缓冲器件。在数据传送中，它可以起到数据缓冲作用，使CPU与主存速度相匹配，从而使两者的速度都能得到发挥和提高。

4．读写系统

读写系统包括写入信息和读出信息所需线路。写入信息所需线路包括写入线路、写驱动器等；读出信息所需线路包括读出线路、读驱动器和读出放大器等。

5．控制线路

无论是读或写操作，都需要由一系列明确规定的连续操作步骤来完成，这就需要主存时序线路、时钟脉冲线路、读逻辑控制线路、写或重写逻辑控制线路以及动态存储器的定时刷新线路等，这些线路总称为存储器控制线路。存储器控制线路控制逻辑电路接收片选信号 CS（Chip Select）及来自 CPU 的读/写控制信号，形成芯片内部控制信号，并控制数据的读出和写入。

主存储器的工作原理：由 CPU 发来的地址送到存储器地址寄存器中，在读写控制线路的作用下，经过地址译码后，选中存储体中某一存储单元，对该存储单元进行读/写操作，读出或写入的信息都暂存于存储器数据寄存器中。

3.2.3 主存储器的扩展

CPU 对存储器进行读/写操作，首先由地址总线给出地址信号，然后发出读操作或写操作的控制信号，最后在数据总线上进行信息交流。因此，存储器同 CPU 连接时，要完成地址线、数据线和控制线的连接。

目前生产的存储器芯片的容量是有限的，在字数或字长方面与存储器的实际要求都有差距，所以需要在字向和位向两方面进行扩充才能满足实际存储器的容量要求，通常采用位扩展法、字扩展法、字位同时扩展法。

1．位扩展法

假定使用 8K×1 位的 RAM 芯片，那么组成 8K×8 位的存储器，可采用图 3-4 所示的位扩展法，此时只需把字长由 1 位加大到 8 位，而存储器的字数（8K）则与存储器芯片的字数一致。图中，每一片 RAM 的字数是 8K（2^{13}），故其地址线为 13 条（A_0~A_{12}），可满足整个存储体容量的要求；每一片 RAM 对应数据的 1 位（只有 1 条数据线），故只需将它们分别接到数据总线上的相应位即可。在这种方式中，对芯片没有选片要求，就是说芯片均按已被选中来考虑。在这种连接中，每一条地址总线接有 8 个负载，每一条数据线接有 1 个负载。

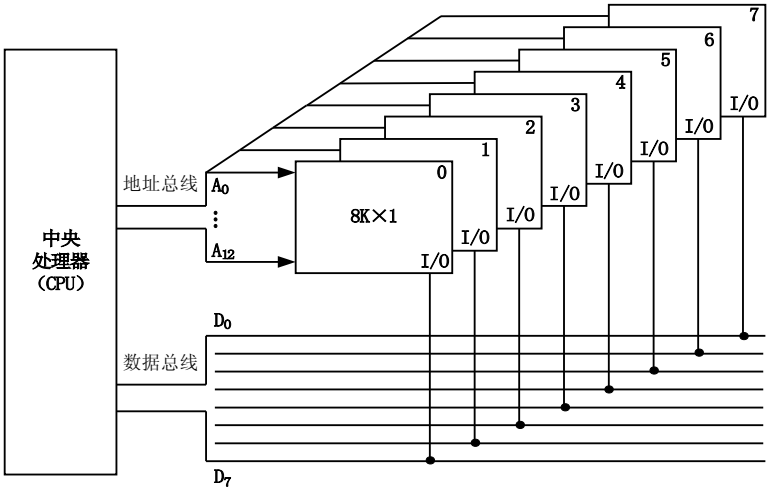


图 3-4 位扩展法组成 8K×8 位存储器

2．字扩展法

字扩展法仅在字向扩充，而位数不变，因此可以将芯片的地址线、数据线、读/写控制线并联，而由片选信号来区分芯片的具体地址，故片选信号端连接到选片译码器的输出端。

使用 16K×8 位的 RAM 芯片，采用字扩展法组成 64K×8 位存储器的连接如图 3-5 所示，其中每一片 RAM 的字数是 16K（2^{14}），故其地址线为 14 条（A_0~A_{13}）。图中，4 片芯片的数据线与数据总线的 D_0~D_7 相连，对应 8 位数据，地址总线低位地址 A_0~A_{13} 与各片芯片的 14 位地址端相连，两位高位地址 A_{14}、A_{15} 经 2:4 译码器与 4 片芯片的片选端相连，其地址空间分配见表 3-1。

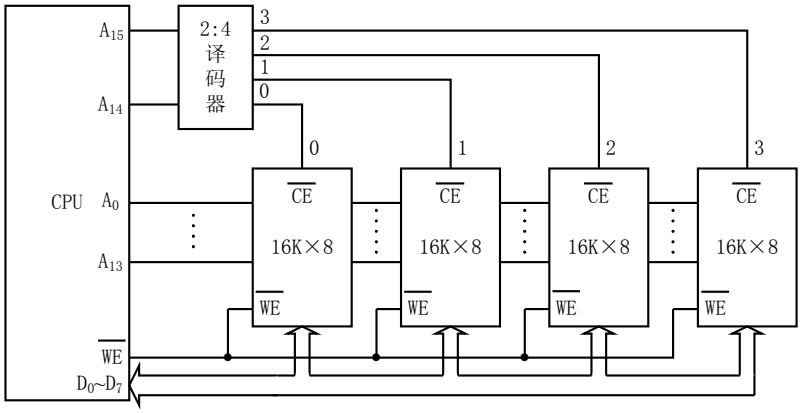


图 3-5 字扩展法组成 64K×8 位存储器

表 3-1 图 3-5 的地址空间分配表

地址 片号	片外	A_{15}	A_{14}	片内	A_{13}	A_{12}	A_{11}	…	A_1	A_0	说明
0		0	0		0	0	0	…	0	0	最低地址
		0	0		1	1	1	…	1	1	最高地址
1		0	1		0	0	0	…	0	0	最低地址
		0	1		1	1	1	…	1	1	最高地址
2		1	0		0	0	0	…	0	0	最低地址
		1	0		1	1	1	…	1	1	最高地址
3		1	1		0	0	0	…	0	0	最低地址
		1	1		1	1	1	…	1	1	最高地址

3．字位同时扩展法

一个存储器的容量假定为 M×N 位，若使用 L×K 位的芯片（L<M, K<N），则需要在字向和位向同时进行扩展，此时共需要（M/L）×（N/K）个存储器芯片。

3.3 高速存储器

高速 CPU 与主存储器在速度上不相匹配，而且在一个 CPU 周期中有可能需要用到几个存储器字，这成为了限制高速计算的主要问题。为了使 CPU 不至于因为等待存储器读写操作的完成而无事可做，可以采取一些特殊措施，以加速 CPU 和存储器之间的有效传输：

⑴主存储器采用更高速的技术来缩短存储器的读出时间，或加长存储器的字长；

⑵采用并行操作的双端口存储器；

⑶在 CPU 和存储器之间插入一个高速缓冲存储器（Cache），以缩短读出时间

⑷在每个存储器周期中存取几个字。

3.3.1 双端口存储器

常规存储器是单端口存储器，每次只接收一个地址，访问一个存储单元，从中读取或写入一个字节或字。主存储器是信息交换的中心，一方面 CPU 频繁地与主存交换信息，另一方面外设也较频繁地与主存交换信息，而单端口存储器每次只能接受一个访存者，或是读或是写，这就影响到存储器的整体工作速度。为此，在某些系统中采用了双端口存储器。

图 3-6 所示的双端口存储器具有两个彼此独立的读写口，每个读写口都有一套自己的地址寄存器和译码电路，可以并行地独立工作。两个读写口可以按各自接收的地址同时读出或写入，或一个写入而另一个读出。与两个独立的存储器不同，两个读写口的访存空间相同，可以访问同一个存储单元。通常使双端口存储器的一个读写口面向 CPU，另一个读写口则面向外设或输入输出处理机。

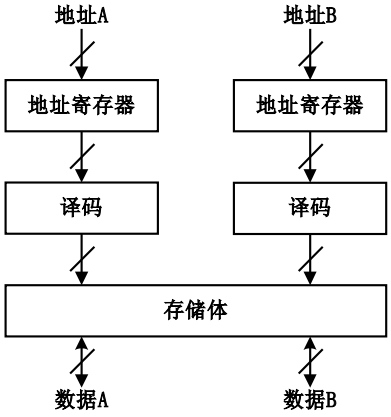


图 3-6 双端口存储器

由此可见，双端口存储器由于具有两组相互独立的读写控制线路，可以对存储器中任何位置上的数据进行并行、独立的存取操作，因而是一种高速工作的存储器。

如果两个端口同时访问存储器的同一个存储单元，便会发生读写冲突。为解决此问题，可以设置一个“忙”标志。在发生读写冲突时，片上判断逻辑决定对哪个端口优先进行读写操作，而对另一个被延迟的端口置“忙”标志，即暂时关闭此端口。等到优先端口完成读写操作，才将被延迟端口的“忙”标志复位，重新开放此端口，允许延迟端口进行存取。

3.3.2 多模块交叉存储器

1.存储器的模块化组织

速度和容量是主存储器设计的两大主要课题，计算机的发展对主存储器提出了更高速度和更大容量的要求。如果主存储体由物理上互相分隔的若干个模块构成，并给每个模块配置自己的存储器地址寄存器（MAR）、存储器数据寄存器（MDR）和读写电路，使每个模块都成为一个能独立进行读写操作的存储器，那么就有可能在任一给定时刻对几个模块同时执行读或写操作，从而提高整个主存的平均存取时间。由多个能独立操作的模块所组成的存储器，称为多模块存储器（多体存储器），其地址分配方案有两种：

1）顺序方式

在常规的主存储器设计中，访问地址采用顺序方式，如图 3-7 所示。CPU 送来的主存地址被分成高 n 位和低 m 位。主存地址的高 n 位表示模块号，其模块号为 0,1,2,…,2^n-1，共 2^n 个，译码后从 2^n 个模块中选中一个模块；主存地址的低 m 位表示块内地址，m 位译码后，选定模块中的一个具体的存储字单元。在一个模块内，程序从低位地址连续存放。这样，当 CPU 执行对主存连续单元的读写请求时，只有一个模块和 CPU 进行数据存取操作，其他模块则可停止工作或与外部设备进行直接存储器存取（DMA）操作。

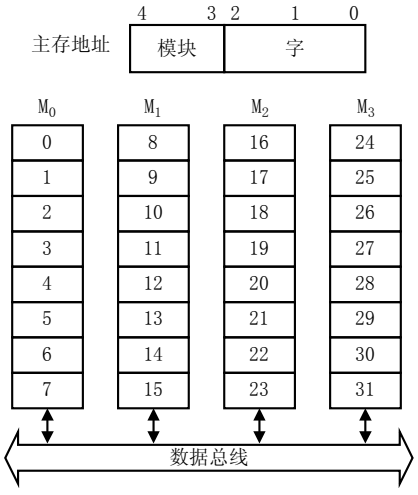


图 3-7 存储器模块的顺序组织方式

顺序方式的优点是可以通过简单地增加模块而方便地扩展系统存储容量，并且一旦发生故障，出现故障的模块仅仅影响存储空间的一个局部区域，其他模块可以照常工作。但在这种结构中，各模块彼此串行工作，存储器带宽受到很大限制，难以有效提高主存速度。

2）交叉方式

多模块存储器的另一种地址分配方案是交叉方式，如图 3-8 所示。图中的存储器容量为 32 个字，分成 4 个模块，每个模块 8 个字。将 4 个线性地址 0,1,2,3 依次分配给 M_0,M_1,M_2,M_3 模块，再将线性地址 4,5,6,7 依次分给 M_0,M_1,M_2,M_3 模块，……。当存储器寻址时，用地址寄存器的低 2 位选择 4 个模块中的 1 个，而用高 3 位选择模块中的 8 个字。

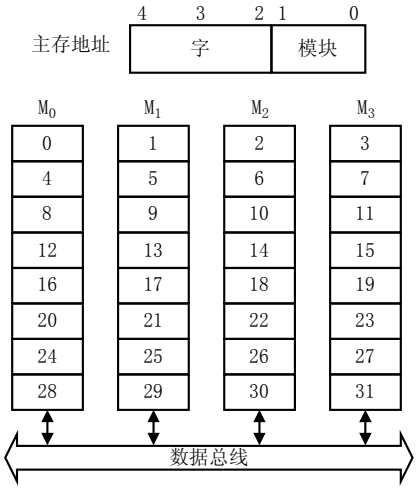


图 3-8 存储器模块的交叉组织方式

主存地址的低 n 位表示模块号，高 m 位表示块内地址，使得连续地址分布在相邻的不同模块内，而同一个模块内的地址都是不连续的，容量相同的不同模块各自以等同的方式与 CPU 交换信息。图中的模块 M_0,M_1,M_2,M_3 采用地址交叉编址方法，即将单元地址依次排在各个模块中，称为模 4 交叉编址。若有 m 个模块，则称为模 m 交叉编址。

由于采用交叉存储编址，对于任何 CPU 读写访问或与外设 DMA 传送，只要是对主存连续字的成块传送，就可以实现多模块流水式并行存取，亦即使多个模块在任一时刻同时并行工作，大大提高存储器的带宽（如图 3-9 所示）。由于 CPU 的速度比主存快，同时从主存取出多条指令，必然会提高机器的运行速度。

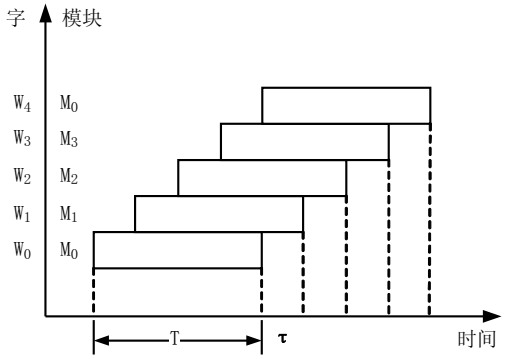


图 3-9 四模块交叉存储器的流水线方式存取示意图

CPU 访问 4 个存储体，可以分时启动、分时控制，即每经过 $\frac{1}{4}$ 个存储周期 T 就访问一个模块。在 0 时刻启动模块 M_0，在 $\frac{1}{4}T,\frac{2}{4}T,\frac{3}{4}T$ 时刻分别启动模块 M_1,M_2,M_3，经 T 周期后，四个模块就都进入了并行工作状态，各模块在存储周期内互相重叠访问。这样，对每个存储体来说，存储周期是 T，而对 CPU 来说，存储周期为

$\frac{T}{4}$。在理想情况下，主存的周期缩短为$\frac{T}{n}$，n 为模块数，T 为每个存储体的存储周期。程序转移及操作数寻址通常会导致对存储器的访问不是按顺序地址，甚至有可能多个地址访问同一个模块，这些都使得交叉存储器的实际有效周期略低于理论上的有效周期，但交叉存储器能大大提高存取速度则是显而易见的。

这种方案也有明显的不足，即任一模块一旦出现故障，都将影响到整个存储器。

2. 多模块交叉存储器的基本结构

图 3-10 所示为四模块交叉存储器的结构框图。

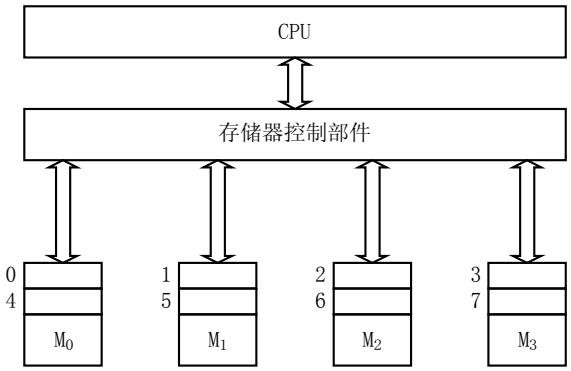


图 3-10 四模块交叉存储器结构框图

主存被分为 4 个相互独立、容量相同的模块，每个模块有自己的读写控制电路、地址寄存器和数据寄存器，各自以同等的方式与 CPU 交换信息。地址码的低位字段经过译码选择不同的模块，而高位字段则指向相应模块内的存储字。连续地址分布在相邻的不同模块内，同一个模块内的地址都是不连续的。

因此，借由交叉存储方式，可以实现对连续字成块传送的多模块流水式并行存取。CPU 同时访问 4 个模块，由存储器控制部件控制它们分时使用数据总线进行信息传递。对每一个存储器模块而言，从 CPU 给出访存命令直到读出信息仍然使用一个存取周期时间，但对 CPU 而言，它可以在一个存取周期内连续访问 4 个模块，各模块的读写过程重叠进行。所以多模块交叉存储器是一种并行存储器结构，可以大大提高存储器的带宽。

3.3.3 相联存储器

1. 相联存储器的基本原理

前面介绍的存储器都是按地址访问的存储器，而相联存储器则是按内容访问的存储器。

相联存储器是选择记录中的一个字段内容作为地址来存取的存储器。选用来寻址存储器的字段叫做关键字。例如，存储器中存放学生信息，如果选学号作为关键字，就用所给的学号作为地址来访问存储器，当要查找某个学号学生的其他信息时，就可以通过学号直接访问存储器，得到相关信息。

存放在相联存储器中的项可以看成具有下列格式：

KEY，DATA

其中 KEY 是地址，DATA 是被读写的信息。

相联存储器的基本原理是：把存储单元所存内容的某一部分作为检索项（即关键字项），用来检索存储器，并读出或写入存储器中与该检索项相符的存储单元的内容。

2. 相联存储器的组成

相联存储器由存储体、检索寄存器、屏蔽寄存器、符合寄存器、比较线路、代码寄存器、控制线路等组成，如图 3-11 所示。

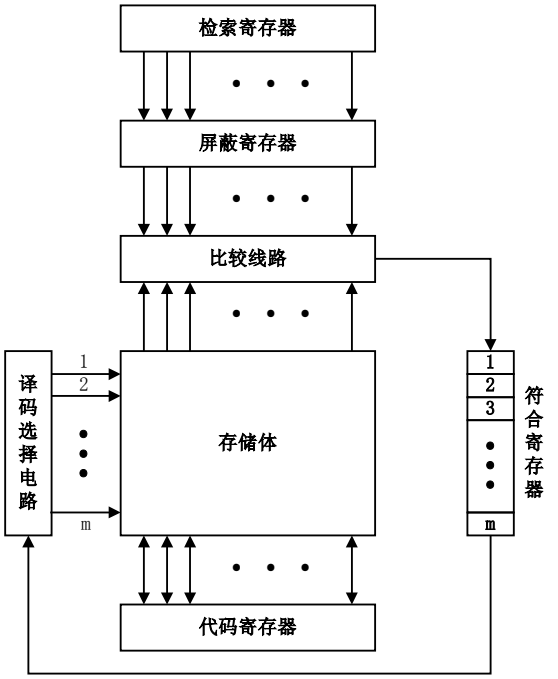


图 3-11 相联存储器的组成框图

存储体：由高速半导体存储器构成，以求快速存取。

检索寄存器：用来存放检索字，其位数与相联存储器的存储单元的位数相等，每次检索时，取其中若干位作为检索项（即关键字项）。◆

屏蔽寄存器：用来存放屏蔽码，其位数与检索寄存器位数相同，检索项所对应的位值为“1”，其他位值均为“0”。屏蔽寄存器用来将检索寄存器中除检索项以外的位置“0”。

符合寄存器：用来存放按检索项内容进行检索的存储体中与之符合的单元地址，其位数等于相联存储器的存储单元数，每一位对应一个存储单元，位的序数即为相联存储器的单元地址。

比较线路：把检索项和从存储体中读出的所有单元内容的相应位进行比较，如果有某个存储单元与检索项符合，就把符合寄存器的相应位置“1”，表示该字已被检索。◆

代码寄存器：用来存放存储体中读出的数据，或者存放向存储体中写入的数据。

在计算机系统中，相联存储器主要用于虚拟存储器中存放段表、页表和快表，以及高速缓冲存储器 Cache 中存放块地址。这是因为，在这两种应用中，都需要快速查找。

3.4 高速缓冲存储器（Cache）

在计算机系统中，CPU 的运行速度与主存的访问速度极不匹配。例如，800MHz Pentium III CPU 一条指令的执行时间约为 1.25ns，而 133MHz SDRAM 的存取时间为 7.5ns，即在 83%的时间内 CPU 都处于等待状态，运行效率极低。为了提高 CPU 利用率，现代计算机体系结构中广泛采用高速缓冲存储器（Cache）技术。

3.4.1 Cache基本原理

在设计和开发系统程序和应用程序时，程序员通常采用模块化的程序设计方法。某一模块的程序，往往集中在存储器逻辑地址空间中很小的一块范围内，且程序地址分布是连续的。也就是说，CPU 在一段较短的时间内，是对连续地址的一段很小的主存空间频繁地进行访问，而对此范围以外地址的访问甚少，这种现象称为程序访问的局部性。

高速缓冲存储器（Cache）技术就是利用程序访问的局部性原理，把程序中正在使用的部分（活跃块）存放在一个小容量的高速 Cache 中，使 CPU 的访存操作大多针对 Cache 进行，从而解决高速 CPU 和低速主存之间速度不匹配的问题，使程序的执行速度大大提高。

1. Cache 的功能

Cache 是介于 CPU 和主存之间的小容量存储器，存取速度比主存快，接近 CPU。它能高速地向 CPU 提供指令和数据，提高程序的执行速度。Cache 技术是为了解决 CPU 和主存之间速度不匹配而采用的一项重要技术。

Cache 是主存的缓冲存储器，由高速的 SRAM 组成，所有控制逻辑全部由硬件实现，对程序员而言是透明的。随着半导体器件集成度的不断提高，当前有些 CPU 已内置 Cache，并且出现了两级以上的多级 Cache 系统。Cache 系统与 CPU 和主存的关系如图 3-12 所示。

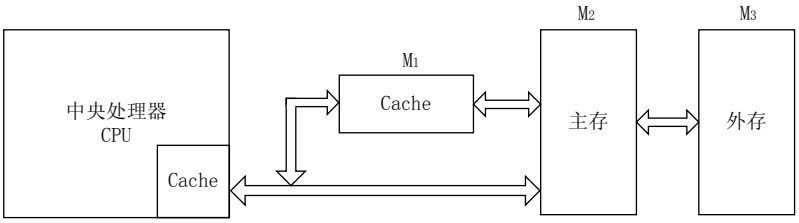


图 3-12 Cache 与 CPU 和主存的关系

2. Cache 的基本原理

CPU 与 Cache 之间的数据交换是以字为单位的，而 Cache 与主存之间的数据交换则是以块为单位的。一个块由若干个定长字组成。

当 CPU 读取主存中的一个字时，该字的主存地址被发给 Cache 和主存，此时，Cache 控制逻辑依据地址判断该字当前是否存在于 Cache 中：若在，该字立即被从 Cache 传送给 CPU；若不在，则用主存读周期把该字从主存读出送到 CPU，同时把含有这个字的整个数据块从主存读出送到 Cache 中，并采用一定的替换策略将 Cache 中的某一块替换掉，替换算法由 Cache 管理逻辑电路来实现。

Cache 原理图如图 3-13 所示。图中，按内容寻址的相联存储器（表），用于存放与 Cache 中数据相对应的主存地址，可以快速检索、判断 CPU 读取的某个字当前是否存在于 Cache 中。

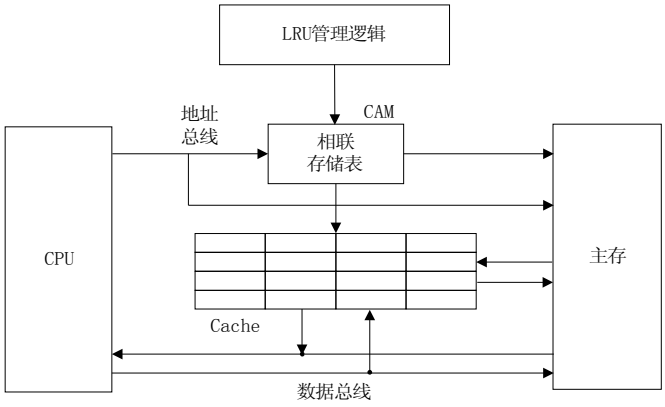


图 3-13 Cache 原理图

3. Cache 的命中率◆

基于程序访问的局部性原理，增加 Cache 使得要访问的数据绝大多数都可以在 Cache 中找到，这样才能在性能上使主存的平均读出时间尽可能接近 Cache 的读出时间。Cache 的工作效率通常用“命中率”来表示。

命中率指的是 CPU 要访问的信息在 Cache 中的概率，Cache 的命中率越高，CPU 访问主存的速度就越接近访问 Cache 的速度。通常 Cache 的容量越大，存储的块也越多，CPU 的命中率就越高。但是，当 Cache 的容量达到一定值时，命中率并不会随着容量的增大而增加，而且 Cache 容量的增大将导致成本的增加，所以，Cache 的容量一般是命中率与成本价格的折中。

在一个程序执行期间，设 Nc 表示 Cache 完成存取的总次数，Nm 表示主存完成存取的总次数，h 定义为命中率，则有

$$h = \frac{N_c}{N_c + N_m} \qquad \text{(式 3-1)}$$

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

$$t_a = ht_c + (1 - h)t_m \qquad \text{(式 3-2)}$$

设 e 表示访问效率，则有

$$e = \frac{t_c}{t_a} \qquad \text{(式 3-3)}$$

为提高访问效率 e，命中率 h 越接近 1 越好。命中率 h 与程序的行为、Cache 的容量、组织方式、块的大小有关。

3.4.2 地址映射

Cache 的容量很小，它保存的内容只是主存内容的一个子集，且 Cache 与主存的数据交换是以块为单位的。为了把信息放到 Cache 中，必须应用某种函数把主存地址定位到 Cache 中，这称为地址映射。在信息按这种映射关系装入 Cache 后，CPU 执行程序时，会将程序中的主存地址变换成 Cache 地址，这个变换过程叫做地址变换。

Cache 的地址映射方式有直接映射、全相联映射和组相联映射。假设某台计算机主存容量为 1 MB，被分为 2048 块，每块 512B；Cache 容量为 8KB，被分为 16 块，每块也是 512B。下面以此为例介绍三种基本的地址映射方法。

1. 直接映射

直接映射的 Cache 组织如图 3-14 所示。主存中的一个块只能映射到 Cache 的某一特定块中去。例如，主存的第 0 块、第 16 块、……、第 2032 块，只能映射到 Cache 的第 0 块；而主存的第 1 块、第 17 块、……、第 2033 块，只能映射到 Cache 的第 1 块……。

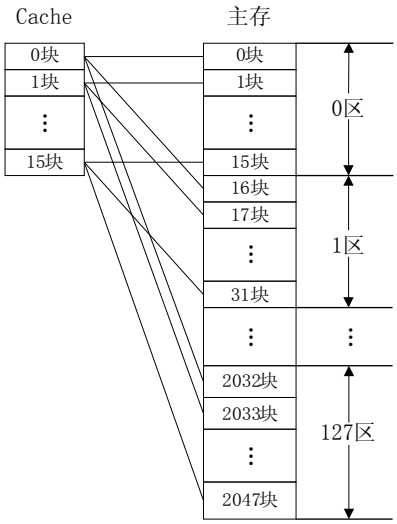


图 3-14 直接映射示意图

直接映射是最简单的地址映射方式，它的硬件简单，成本低，地址变换速度快，而且不涉及替换算法问题。但是这种方式不够灵活，Cache 的存储空间得不到充分利用，每个主存块只有一个固定位置可存放，容易产生冲突，使 Cache 效率下降，因此只适合大容量 Cache 采用。例如，如果一个程序需要重复引用主存中第 0 块与第 16 块，最好将主存第 0 块与第 16 块同时复制到 Cache 中，但由于它们都只能复制到 Cache 的第 0 块中去，即使 Cache 中别的存储空间空着也不能占用，因此这两个块会不断地交替装入 Cache 中，导致命中率降低。

2. 全相联映射

图 3-15 是全相联映射的 Cache 组织，主存中任何一块都可以映射到 Cache 中的任何一块位置上。

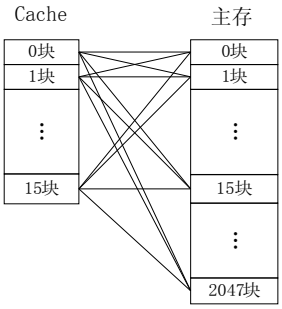


图 3-15 全相联映射示意图

全相联映射方式比较灵活，主存的各块可以映射到 Cache 的任一块中，Cache 的利用率高，块冲突概率低，只要淘汰 Cache 中的某一块，即可调入主存的任一块。但是，由于 Cache 比较电路的设计和实现比较困难，这种方式只适合于小容量 Cache 采用。

3. 组相联映射

组相联映射实际上是直接映射和全相联映射的折中方案，其组织结构如图 3-16 所示。主存和 Cache 都分组，主存中一个组内的块数与 Cache 中的分组数相同，组间采用直接映射，组内采用全相联映射。也就是说，将 Cache 分成 u 组，每组 v 块，主存块存放到哪个组是固定的，至于存到该组哪一块则是灵活的。例如，主存分为 256 组，每组 8 块，Cache 分为 8 组，每组 2 块。

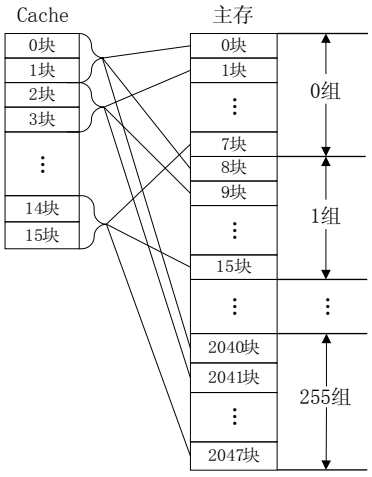


图 3-16 组相联映射示意图

主存中的各块与 Cache 的组号之间有固定的映射关系，但可自由映射到对应 Cache 组中的任何一块。例如，主存中的第 0 块、第 8 块……均映射于 Cache 的第 0 组，但可映射到 Cache 第 0 组中的第 0 块或第 1 块；主存的第 1 块、第 9 块……均映射于 Cache 的第 1 组，但可映射到 Cache 第 1 组中的第 2 块或第 3 块。

常采用的组相联结构 Cache，每组内有 2、4、8、16 块，称为 2 路、4 路、8 路、16 路组相联 Cache。组相联结构 Cache 是前两种方法的折中方案，适度兼顾二者的优点，尽量避免二者的缺点，因而得到普遍采用。

3.4.3 替换策略

Cache 工作原理要求它尽量保存最新数据，当从主存向 Cache 传送一个新块，而 Cache 中可用位置已被占满时，就会产生 Cache 替换的问题。替换问题与 Cache 的组织方式紧密相关：对直接映射 Cache 来说，只要把此可用位置上的主存块换出 Cache 即可；对全相联和组相联 Cache 来说，要从若干个可用位置中选取一个位置，把其中的主存块换出 Cache。

常用的替换算法有下面三种。

1. 最不经常使用（LFU）算法◆

LFU（Least Frequently Used，最不经常使用）算法将一段时间内被访问次数最少的那个块替换出去。每块设置一个计数器，从0开始计数，每访问一次，被访块的计数器就增1。当需要替换时，将计数值最小的块换出，同时将所有块的计数器都清零。

这种算法将计数周期限定在对这些特定块两次替换之间的间隔时间内，不能严格反映近期访问情况，新调入的块很容易被替换出去。

2. 近期最少使用（LRU）算法

LRU（Least Recently Used，近期最少使用）算法是把CPU近期最少使用的块替换出去。这种替换方法需要随时记录Cache中各块的使用情况，以便确定哪个块是近期最少使用的块。每块也设置一个计数器，Cache每命中一次，命中块计数器清零，其他各块计数器增1。当需要替换时，将计数值最大的块换出。

LRU算法相对合理，但实现起来比较复杂，系统开销较大。这种算法保护了刚调入Cache的新数据块，具有较高的命中率。LRU算法不能肯定调出去的块近期不会再被使用，所以这种替换算法不能算作最合理、最优秀的算法。但是研究表明，采用这种算法可使Cache的命中率达到90%左右。

3. 随机替换

最简单的替换算法是随机替换。随机替换算法完全不管Cache的情况，简单地根据一个随机数选择一块替换出去。随机替换算法在硬件上容易实现，且速度也比前两种算法快。缺点则是降低了命中率和Cache工作效率。

Cache命中率除了和替换算法有关外，还与Cache的容量及块的大小有关。

3.4.4 写操作策略

由于Cache的内容只是主存内容的一个子集，应当与主存内容保持一致，而CPU对Cache的写入更改了Cache的内容。为此，可选用写操作策略使Cache内容与主存内容保持一致。

1、写回法（Write-Back）

当CPU写Cache命中时，只修改Cache的内容，而不是立即写入主存；只有当此块被换出时才写回主存。

使用这种方法写Cache和写主存异步进行，显著减少了访问主存的次数，但是存在数据不一致的隐患。实现这种方法时，每个Cache块必须配置一个修改位，以反映此块是否被CPU修改过。

2、全写法（Write-Through）

当写 Cache命中时，Cache与主存同时发生写修改。

使用这种方法写Cache和写主存同步进行，因而较好地维护了 Cache与主存的内容一致性。实现这种方法时，Cache中的每个块无需设置修改位以及相应的判断逻辑，但由于Cache对CPU向主存的写操作没有高速缓冲功能，从而降低了Cache的功效。

3、写一次法（Write-Once）

写一次法是基于写回法并结合全写法的写操作策略，写命中与写未命中的处理方法与写回法基本相同，只是第一次写命中时要同时写入主存，以便于维护系统全部Cache的一致性。

3.5 虚拟存储器

3.5.1 虚拟存储器基本原理

1. 什么是虚拟存储器

当代计算机系统的主存主要由半导体存储器组成，由于工艺和成本的原因，主存的容量受到限制。然而，计算机系统软件和应用软件的功能不断增强，程序规模迅速扩大，要求主存的容量越大越好，这就产生了矛盾。为了给大的程序提供方便，使它们摆脱主存容量的限制，可以由操作系统把主存和辅存这两级存储系统管理起来，实现自动覆盖。也就是说，一个大作业在执行时，其一部分地址空间在主存，另一部分在辅存，当所访问的信息不在主存时，则由操作系统而不是程序员来安排I/O指令，把信息从辅存调入主存。从效果上来看，好像为用户提供了一个存储容量比实际主存大得多的存储器，用户无需考虑所编程序在主存中是否放得下或放在什么位置等问题。我们称这种存储器为虚拟存储器。

虚拟存储器只是一个容量非常大的存储器的逻辑模型，不是任何实际的物理存储器。它借助于磁盘等辅助存储器来扩大主存容量，使之为更大或更多的程序所使用。虚拟存储器指的是主存－外存层次，它以透明的方式为用户提供了一个比实际主存空间大得多的程序地址空间。

物理地址是实际的主存单元地址，由CPU地址引脚送出，是用于访问主存的。设CPU地址总线的宽度为m位，则物理地址空间的大小就是2^m。

虚拟地址是用户编程时使用的地址，由编译程序生成，是程序的逻辑地址，其地址空间的大小受到辅助存储器容量的限制。显然，虚拟地址要比实际地址大得多。程序的逻辑地址空间称为虚拟地址空间。

程序运行时，CPU以虚拟地址来访问主存，由辅助硬件找出虚拟地址和实际地址之间的对应关系，并判断这个虚拟地址指示的存储单元内容是否已装入主存。如果已在主存中，则通过地址变换，CPU可直接访问主存的实际单元；如果不在主存中，则把包含这个字的一个存储块调入主存后再由CPU访问。如果主存已满，则由替换算法从主存中将暂不运行的一块调回外存，再从外存调入新的一块到主存。

从原理角度看，虚拟存储器和Cache－主存层次有不少相同之处。事实上，前面提到的各种控制方法是先应用于虚拟存储器中，后来才发展到Cache-主存层次中去的。不过，Cache-主存层次的控制完全由硬件实现，所以对各类程序员是透明的；而虚拟存储器的控制是软硬件相结合的，对于设计存储管理软件的系统程序员来说是不透明的，对于应用程序员来说是透明的。

主存－外存层次和Cache－主存层次所使用的地址变换及映射方法和替换策略，从原理上看是相同的，都基于程序局部性原理。它们遵循的原则是：

⑴把程序中最近常用的部分驻留在高速的存储器中；

⑵一旦这部分变得不常用了，把它们送回到低速的存储器中；

⑶这种换入换出是由硬件或操作系统完成的，对用户是透明的；

⑷力图使存储系统的性能接近高速存储器，价格接近低速存储器。

两种存储系统的主要区别在于：在虚拟存储器中未命中的性能损失，要远大于Cache系统中未命中的损失。

2. 主存－外存层次的基本信息传送单位◆

主存－外存层次的基本信息传送单位可采用页、段和段页3种不同的方案。根据地址格式的不同，虚拟存储器可分成页式虚拟存储器、段式虚拟存储器和段页式虚拟存储器3种。

页式和段式存储结构采用二维地址格式，它们把整个存储器空间（包括主存、辅存和虚拟存储器）分成若干个页或段，每个页或段又包含若干个存储单元。段页式存储结构采用三维地址格式，它把整个存储器分成若干个段，每段又分成若干页，每页包含若干个存储单元。

页式管理系统以定长的页为基本信息传送单位，主存的物理空间也被分成等长的页，每一页等长的区域称为页面，页面在主存中的位置是固定的。因此，页面的起始地址和结束地址都是固定的，这给页表的制作带来很大的方便。新页调入主存也很容易，只要有空闲的页面就可容纳。

把主存按段分配的存储管理方式称为段式管理。段是利用程序的模块化性质，按照程序的逻辑结构划分成多个相对独立的部分，如过程、数据表、阵列等。段作为独立的逻辑单位可以被其他程序段调用，这样就形成了段间连接，产生规模较大的程序。因此，把段作为基本信息单位在主存－外存之间传送和定位是比较合理的。一般用段表来指明各段在主存中的位置，每段都有它的名称（用户名称或数据结构名称或段号）、段起点、段长等。段表也是主存的一个可再定位的段。段式管理的优点是段的分界与程序的自然分界相对应，段的逻辑独立性使它易于编译、管理、修改和保护，也便于多道程序共享。某些类型的段（例如堆栈、队列）具有动态可变长度，允许自由调度以便有效利用主存空间。但是，正因为段的长度各不相同，段的起始地址和结束地址不定，这给主存空间分配带来麻烦，而且容易在段间留下许多碎片不好利用，造成浪费，这种浪费比页式管理系统要大。

页式存储管理和段式存储管理各有优缺点，段页式存储管理则是结合两者优点的一种方案。程序按模块分段，段内再分页，进入主存仍以页为基本信息传送单位，用段表和页表（每段一个页表）进行两级定位管理。

3.5.2 页式虚拟存储器

以页为基本单位的虚拟存储器叫做页式虚拟存储器。主存空间和虚存空间都划分成若干个大小相等的页，主存（即实存）的页称为实页，虚存的页称为虚页。

虚存地址分为高低两个字段：高位字段为逻辑页号，低位字段为页内地址。实存地址也分为高低两个字段：高位字段为物理页号，低位字段为页内地址。虚存地址到实存地址的变换是通过存放在主存中的页表来实现的。在页表中，对应每一个虚存逻辑页号有一个表项，表项内容包含该逻辑页所在的主存页面地址（物理页号），用它作为实存地址的高字段，与虚存地址的页内地址字段相拼接，产生完整的实存地址，据此来访问主存。地址变换如图 3-17 所示。

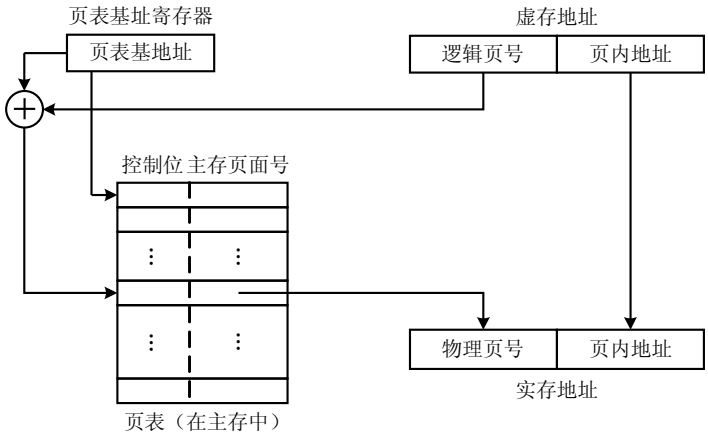


图 3-17 页式虚拟存储器结构

若计算机采用多道程序工作方式，则可为每个用户作业建立一个页表，硬件中设置一个页表基址寄存器，存放当前所运行程序的页表的起始地址。

页表中的表项除包含虚页号对应的实页号之外，还包括装入位、修改位、替换控制位等控制字段。若装入位为“1”，表示该页面已在主存中，将对应的实页号与虚地址中的页内地址相拼接就得到了完整的实地址；若装入位为“0”，表示该页面不在主存中，于是要启动 I/O 系统，把该页从外存中调入主存后再供 CPU 使用。修改位指出主存页面中的内容是否被修改过，替换时是否要写回外存。替换控制位指出需替换的页，与替换策略有关。

CPU 访存时首先要查页表，为此需要访问一次主存，若不命中，还要进行页面替换和页表修改，则访问主存的次数就更多了。为了将访问页表的时间降低到最低限度，许多计算机将页表分为快表和慢表两种。将当前最常用的页表信息存放在快表中，作为慢表部分内容的副本。快表很小，存储在一个小容量的快速存储器中，该存储器是按内容查找的相联存储器，可按虚页号名字进行查询，迅速找到对应的实页号。使用快表与慢表进行地址变换如图 3-18 所示。

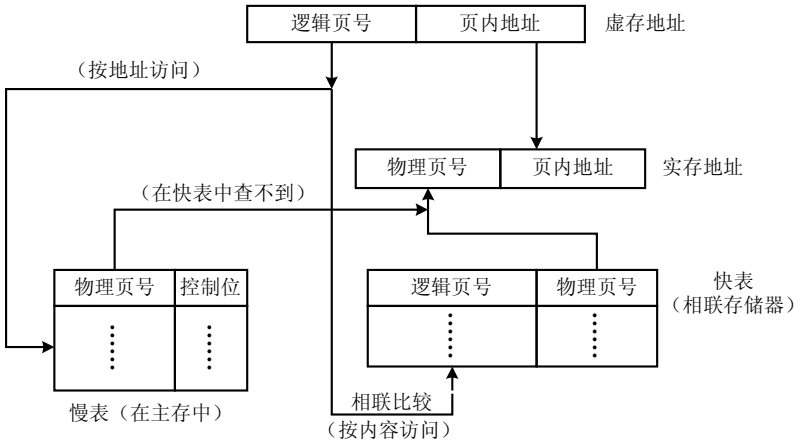


图 3-18 经快表和慢表实现内部地址变换

快表由硬件组成，比页表小得多，查表时，由逻辑页号同时去查快表和慢表。当在快表中有此逻辑页号时，就能很快地找到对应的物理页号送入实主存地址寄存器，从而做到虽采用虚拟存储器但访主存速度几乎没有下降；如果在快表中查不到，那就要花费一个访主存时间去查慢表，从中查到物理页号送入实存地址寄存器，并将此逻辑页号和对应的物理页号送入快表，替换快表中应该移掉的内容，这也要用到替换算法。

页式虚拟存储器的每页长度是固定的，页表的建立很方便，新页的调入也容易实现。但是由于程序不可能正好是页面的整数倍，最后一页的零碎空间将无法利用而造成浪费。同时，页不是逻辑上独立的实体，这使得程序的处理、保护和共享都比较麻烦。

3.5.3 段式虚拟存储器

段式虚拟存储器中的段是按照程序的逻辑结构划分的，各个段的长度因程序而异。在段式虚拟存储系统中，虚拟地址由段号和段内地址组成，为了把程序虚地址变换成主存实地址，需要一个段表。段表一般驻留在主存中，其中每一行记录了某个段对应的若干信息，包括段号、装入位、段起点和段长等。这里的段号指的是虚拟段号。装入位为“1”，表示该段已调入主存；装入位为“0”，表示该段不在主存中。由于段的大小可变，所以在段表中要给出各段的起始地址与段的长度。段表实际上是程序的逻辑结构段与其在主存中的存放位置之间的关系对照表，如图 3-19 所示。段表也是一个段，可以保存在外存中，但一般驻留在主存中。

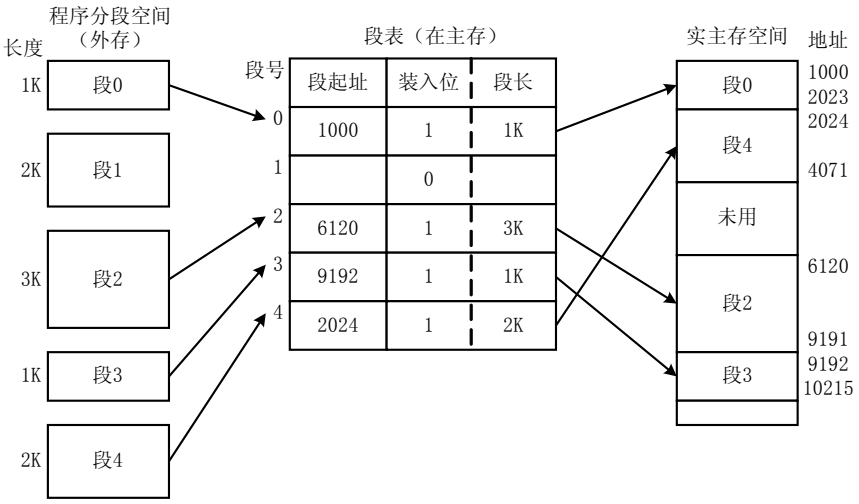


图 3-19 段式虚拟存储器中的段表

段式虚拟存储器的虚－实地址变换如图 3-20 所示。CPU 根据虚地址访存时，首先将段号与段表的起始地址相拼接，形成访问段表对应行的地址，然后根据段表的装入位判断该段是否已调入主存。若已调入主存，则从段表读出该段在主存中的起始地址，与段内地址（偏移量）相加，得到对应的主存实地址。

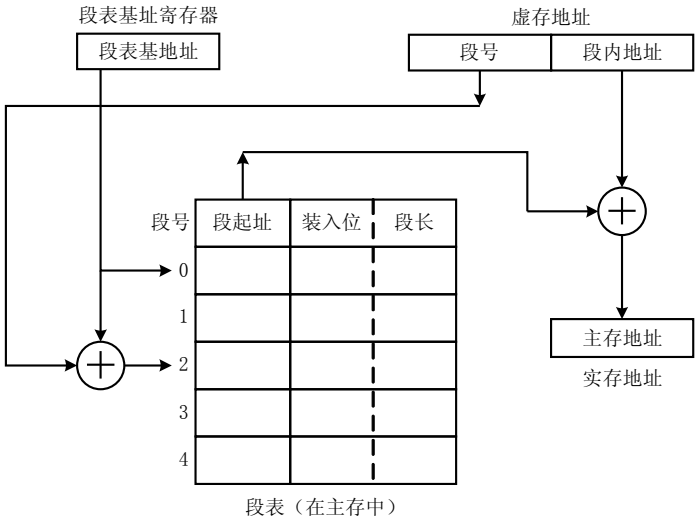


图 3-20 段式虚拟存储器地址变换

由于段的分界与程序的自然分界相对应，具有逻辑独立性，所以易于实现程序的编译、管理、修改和保护，也便于多道程序共享。但是，因为段的长度参差不齐，起点和终点不定，给主存空间分配带来了麻烦，容易在段间留下不能利用的零碎空间，造成浪费。

3.5.4 段页式虚拟存储器

段页式虚拟存储器是段式虚拟存储器和页式虚拟存储器的结合。它把程序按逻辑单位分段以后，再把每段分成固定大小的页。主存空间也划分为若干个同样大小的页。虚存和实存之间以页为基本传送单位，每个程序对应一个段表，每段对应一个页表。虚地址包含段号、段内页号、页内地址三部分。CPU 访问时，首先将段表起始地址与段号合成，得到段表地址，然后从段表中取出该段的页表起始地址，与段内页号合成，得到页表地址，最后从页表中取出实页号，与页内地址拼接形成主存实地址。

段页式存储器综合了前两种结构的优点，但要经过两级查表才能完成地址转换，要多花费一些时间。

3.5.5 替换算法

当 CPU 要访问的数据或指令不在主存中时，产生页面失效（缺页），此时就要从外存调进包含此数据或指令的页，若主存页面已全部占满，就需要采用某种替换算法将主存中的某一页替换出去。虚拟存储器中的页面替换策略与 Cache 中的块替换策略有很多相似之处，但有三点显著不同：

⑴缺页至少要涉及一次磁盘外存存取，以读取所缺的页，因缺页使系统蒙受的损失要比 Cache 未命中大得多。

⑵页面替换是由操作系统软件实现的，而 Cache 的块替换则是由硬件实现的。

⑶页面替换的选择余地很大，属于一个进程的页面都可替换。

虚拟存储器中的替换策略一般采用 LRU 算法、LFU 算法、FIFO 算法，或将两种算法结合起来使用。对于将被替换出去的页面，假如该页调入主存后没有被修改，就不必进行处理，否则就要把该页重新写入外存，以保证外存中数据的正确性。为此，在页表的每一行应设置一个修改位。

第4章 指令系统

4.1 指令系统概述

4.1.1 指令系统的发展

指令是计算机硬件能够识别并直接执行操作的命令，一台计算机中所有指令的集合构成了该机的指令系统。指令系统是表征计算机性能的重要因素，其格式与功能不仅直接影响到机器的硬件结构，也直接影响到系统软件，影响到机器的适用范围。因此，设计一个合理有效、功能齐全、通用性强、丰富的指令系统是至关重要的。

从计算机组成的层次结构来说，计算机的指令分为微指令、机器指令和宏指令三类。微指令是微程序级的命令，属于硬件；宏指令是由若干条机器指令组成的软件指令，属于软件；机器指令，也就是我们通常所说的指令，介于微指令与宏指令之间，每条指令可完成一个独立的算术运算或逻辑运算操作。

回顾计算机的发展历史，指令系统的发展经历了从简单到复杂的演变过程。早在20世纪50至60年代，计算机大多采用分立元件的晶体管或电子管组成，体积庞大，价格昂贵，因此计算机的硬件结构比较简单，所支持的指令系统只有十几至几十条最基本的指令，而且寻址方式简单。到20世纪60年代中期，随着集成电路的出现，计算机的功耗、体积、价格等不断下降，硬件功能不断增强，其指令系统也越来越丰富。20世纪60年代后期，基本指令系统相同、基本体系结构相同的系列计算机开始出现，从而解决了各机种的软件兼容问题，其必要条件是同一系列的各机种具有共同的指令集，而且新推出的机种其指令系统一定包含旧机种的全部指令。

在20世纪70年代，高级语言已成为大、中、小型机的主要程序设计语言，计算机应用日益普及。计算机的设计者利用当时已经成熟的微程序技术和飞速发展的VLSI技术，增设了各种各样复杂的、面向高级语言的指令，使指令系统越来越庞大，按这种方法设计的计算机系统称为复杂指令系统计算机（Complex Instruction Set Computer），简称CISC。如此庞大的指令系统不仅使计算机的研制开发周期变长，正确性难以保证，调试维护困难，而且大量使用频率很低的复杂指令造成了硬件资源的浪费。为此，人们又提出了便于采用VLSI技术实现的精简指令系统计算机（Reduced Instruction Set Computer），简称RISC，这是一种新型的计算机体系结构设计思想，其主要特点是：选取使用频率最高的一些简单指令，指令条数少；指令长度固定，指令格式种类少，寻址方式种类少；只有取数/存数指令访问存储器，其余指令的操作都在寄存器之间进行。我们将在第5章中央处理器（CPU）中详细介绍RISC技术。

4.1.2 指令系统的性能要求

指令系统的性能决定了计算机的基本功能，因此，指令系统的设计是计算机系统设计的一个核心问题，它不仅关系到计算机的硬件结构，同时也关系到用户的使用需要。一个完善的指令系统应满足以下四个方面的要求：

1. 完备性

指令系统的完备性是指用汇编语言编写各种程序时，指令系统直接提供的指令足够使用，而不必用软件来实现。完备性要求指令系统丰富、功能齐全、使用方便。一台计算机中必不可少的最基本的指令构成了指令系统的完备性，而其他一些指令则可以通过基本指令来实现，或者直接通过硬件来实现，两者只是在执行时间和编写程序的难易程度上有所差别。在指令系统中采用硬件指令，是为了提高程序执行速度，也便于用户编写程序。

2. 有效性

有效性是指利用指令系统提供的指令而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。通常，一个功能完善的指令系统必定有很好的有效性。

3. 规整性

规整性是指指令系统的对称性、匀齐性、指令格式和数据格式的一致性。

指令的对称性是指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，这对提高程序的可读性、简化程序设计带来便利。

指令的匀齐性是指一种操作性质的指令可以支持各种数据类型。例如，算术运算指令可支持字节、字和双字整数运算，十进制数运算，单、双精度浮点运算等。因此，程序设计者在选用指令时无须考虑数据类型，可提高编程效率。

指令格式和数据格式的一致性是指指令长度和数据长度有一定的关系，以方便处理和存取。

4．兼容性

兼容性是指计算机的体系结构设计基本相同，计算机之间具有相同的基本结构、数据表示和共同的基本指令集合，因此指令系统也是兼容的，即同一个软件可以不加修改就在其他系统结构相同的机器上使用。做到所有软件都完全兼容是不可能的。目前，对于同一系列的计算机，新推出机种的指令系统通常包含旧机种的全部指令，实现了“向上兼容”，即低档机上运行的软件不需任何修改便可在高档机上运行。

4.2 指令格式

机器指令是用机器字来表示的，表示一条指令的机器字称为指令字（简称指令）。

指令格式是指令字用二进制代码表示的结构形式，一般由两部分组成，包括操作码（Operation Code）字段和地址码（Address Code）字段，其基本格式如图 4-1 所示。操作码字段表征指令的操作特性与功能；地址码字段通常用来指定参与操作的操作数的地址。

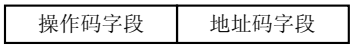


图 4-1 指令的基本格式

4.2.1 操作码

设计计算机时，对指令系统的每一条指令都要规定一个操作码,它是指明指令操作性质的命令码。CPU 从主存每次取出一条指令，指令中的操作码告诉 CPU 应该执行什么性质的操作。例如，可用操作码“000”表示“加法”操作，操作码“010”表示“减法”操作等。不同的操作码代表不同的指令。

组成操作码字段的位数一般取决于计算机指令系统的规模，所需指令数越多，组成操作码字段的位数也就越多。例如，一个指令系统只有 8 条指令，则需要 3 位操作码；如果有 32 条指令，则需要 5 位操作码。一般来说，一个包含 n 位操作码的指令系统最多能够表示 2^n 条指令。

4.2.2 地址码

指令系统中的地址码用来描述指令的操作对象。在地址码中可以直接给出操作数本身，也可以给出操作数在存储器或寄存器中的地址、操作数在存储器中的间接地址等。

根据指令功能的不同，一条指令中可以有一个、两个或者多个操作数地址，也可以没有操作数地址。一般情况下要求有两个操作数地址，但若要考虑存放操作结果，就需要有三个操作数地址。

根据地址码的数量，可以将指令的格式分为：零地址指令、一地址指令、二地址指令、三地址指令和多地址指令。各种不同地址码的指令格式如图 4-2 所示：

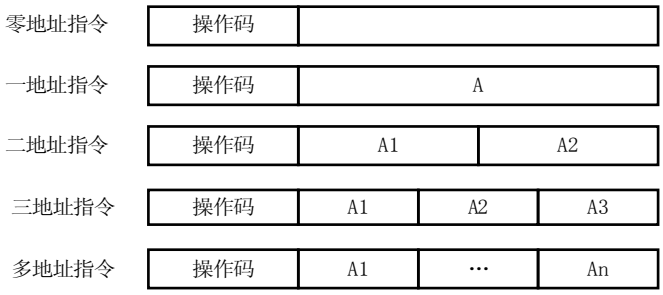


图 4-2 指令格式

1. 零地址指令

在该指令格式中没有地址码部分，只有操作码。

该类指令分两种情况：一种是无需操作数，如空操作指令、停机指令等；另一种则是操作数为默认的（或称隐含的），如操作数在累加器或者堆栈中，它们的操作数由硬件机构来提供。

2. 一地址指令

常称为单操作数指令，该指令中只有一个地址码。这种指令可能是单操作数运算，给出的地址既作为操作数的地址，也作为操作结果的存储地址；也可能是二元运算，指令中提供一个操作数，另一个操作数则是隐含的。例如，以运算器中累加寄存器AC中的数据为被操作数，指令字的地址码字段所指向的数为操作数，操作结果又放回累加寄存器AC中。其数学含义为

(AC) OP (A) → AC

式中，

OP表示操作性质，如加、减、乘、除等；

(AC)表示累加寄存器AC中的数；

(A)表示主存中地址为A的存储单元中的数，或者是运算器中地址为A的通用寄存器中的数；

→表示把操作（运算）结果传送到指定的地方。

注意：地址码字段A指明的是操作数的地址，而不是操作数本身。

3. 二地址指令

这是最常见的指令格式，又称为双操作数指令。通常情况下，在这种指令中包括两个参加运算的操作数的地址码，运算结果保存在其中一个操作数的地址码中，从而使得该地址中原来的数据被覆盖。其数学含义可表示为

（A1）OP（A2）→ A1

式中，两个地址码字段A1和A2分别指明参与操作的两个数在主存或通用寄存器中的地址，且地址A1兼做存放操作结果的地址。

4. 三地址指令

在这种指令中包括两个操作数地址码和一个结果地址码，可使得在操作结束后，原来的操作数不被改变。其数学含义可表示为

（A1）OP（A2）→ A3

式中，A1和A2指明两个操作数地址，A3为存放操作结果的地址。

5. 多地址指令

我们以四地址指令为例，四地址指令比三地址指令增加了下一条要执行的指令地址，因此其优点是非常直观，指令所用的所有参数都有各自的存放地址，并且有明确的下一条指令地址，程序的流程很明确，但是其缺点也是显而易见的，这就是指令所占的长度太长。

从操作数的物理位置来说，二地址指令格式又可归结为三种类型：

1）存储器-存储器（Storage-Storage，SS）型指令

这种指令在操作时需要多次访问主存，参与读、写操作的数都放在主存里。

2）寄存器-寄存器（Register-Register，RR）型指令

这种指令在操作时需要多次访问寄存器，从寄存器中取操作数，把操作结果放到寄存器中。

由于不需要访问主存，机器执行寄存器-寄存器型指令的速度很快。

3）寄存器-存储器（Register-Storage，RS）型指令

这种指令在操作时既要访问主存单元，又要访问寄存器。

计算机选择什么样的指令格式，包括多方面的因素：一般情况下，地址码越少，占用的存储器空间就越小，运行速度也越快，具有时间和空间上的优势；而地址码越多，指令内容就越丰富。因此，要通过指令的功能来选择指令的格式。在计算机中，一个指令系统中所采用的指令地址结构并不是惟一的，往往混合采用多种格式，以增强指令的功能。

4.2.3指令字长度

一个指令字中包含二进制代码的位数，称为指令字长度。计算机能直接处理的二进制数据的位数称为机器字长，它决定了计算机的运算精度，机器字长通常与主存单元的位数一致。

指令字长度等于机器字长的指令，称为单字长指令；指令字长度等于半个机器字长的指令，称为半字长指令；指令字长度等于两个机器字长的指令，称为双字长指令。例如，IBM 370 系列 32 位机的指令格式有半字长的，单字长的，还有一个半字长的。在 Pentium 系列机中，指令字长度也是可变的，有 8 位、16 位、32 位、64 位不等。

使用多字长指令的目的在于提供足够的地址位来解决访问主存任何单元的寻址问题。但是，使用多字长指令的一个主要缺点是必须两次或多次访问主存以取出一整条指令，这样做降低了 CPU 的运算速度，同时占用了更多的存储空间。

在一个指令系统中，如果各种指令字长度是相等的，就称为等长指令字结构，例如都采用单字长指令或半字长指令。这种指令字结构简单，且指令字长度不变。如果各种指令字长度随指令功能而异，比如有的指令是单字长指令，有的指令是双字长指令，这就称为变长指令字结构。这种指令字结构灵活，能充分利用指令长度，但指令的控制较为复杂。

4.2.4 指令助记符

计算机指令的操作码和地址码在计算机中用二进制数据来表示，这种表示方法对于书写和阅读程序却非常麻烦，因此，通常用一些比较容易记忆的文字符号来表示指令中的操作码和操作数，这种符号称为助记符。助记符通常是 3~4 个英文缩写字母，提示了每条指令的意义，因此书写和阅读起来比较方便，也易于记忆。例如，加法指令用 ADD 来代表操作码 001，减法指令用 SUB 来代表操作码 010，传送指令用 MOV 来代表操作码 011，等等。典型的指令助记符如表 4-1 所示。

表 4-1 典型的指令助记符

典型指令	指令助记符	二进制操作码	典型指令	指令助记符	二进制操作码
加法	ADD	001	转移	JSR	101
减法	SUB	010	存储	STR	110
传送	MOV	011	读数	LDA	111
跳转	JMP	100			

注意：在不同的计算机中，指令助记符的规定是不一样的。由于硬件只能识别二进制语言，因此指令助记符必须转换成与它们相对应的二进制操作码，这种转换可以借助汇编程序自动完成，汇编程序的作用相当于一个“翻译”。

4.3 指令分类

不同机器的指令系统是各不相同的。从指令的操作码功能来考虑，一个较为完善的指令系统中常见的指令类型包括：数据传送指令、算术运算指令、逻辑运算指令、程序控制指令、输入输出指令、字符串处理指令、系统控制指令。

4.3.1 数据传送指令

数据传送指令是最基本、最常用、最重要的指令，用来使数据在主存与 CPU 寄存器之间进行传输，可以一次传送一个数据或一批数据，包括取数指令 LOAD、存数指令 STORE、存储器或寄存器间数据传送指令 MOVE 等。

4.3.2 算术运算指令

算术运算是计算机能够执行的基本数值计算，包括加法 ADD、减法 SUB、乘法 MUL、除法 DIV 等指令。算术运算指令的操作数有多种类型，如定点数、浮点数，定点数又可以分为带符号数和无符号数，浮点数又可以分为单精度和双精度，因此，每一种运算指令也按操作数的类型分为带符号数定点运算、无符号数定点运算、单精度浮点运算、双精度浮点运算等。

4.3.3 逻辑运算指令

逻辑运算是对数据进行逻辑操作，包括逻辑与 AND、逻辑或 OR、逻辑非 NOT 等三种基本操作以及同或、异或等组合逻辑操作。

逻辑运算指令进行二进制数据的按位运算。例如逻辑与指令，当两个操作数的对应位都为“1”时，逻辑与操作结果中该位才为“1”，该指令常用于屏蔽或检测数据字中的某些位；逻辑或指令则是当两个操作数的对应位中有一个为“1”时，操作结果中的该位为“1”，该指令常用于将数据字中的某些位置为“1”；逻辑非就是把数据字中的所有位求反。

4.3.4 程序控制指令

该类指令主要控制程序的流程，使程序具有调试与判断功能，主要包括：转移指令、转子程序指令与子程序返回指令、程序中断指令等。

1．转移指令

转移指令包括条件转移和无条件转移指令，该指令将程序计数器 PC（Program Counter）中的指令地址值更新为需要转移的目标指令的地址值。条件转移指令是指当满足规定的条件后才执行转移，而无条件转移指令则不受任何约束地将程序转移。

2．转子程序指令与子程序返回指令

转子程序指令是实现子程序调用的指令。子程序是能够完成某一特定功能的程序段，由于经常要使用，所以将它独立出来作为子程序，在需要时可以随时由主程序调用，而不必多次编写重复代码，以实现共享的最大化，方便程序设计，节省存储空间。

为了能够从子程序中正确返回到主程序的断点（Breakpoint）并继续执行，在调用子程序时，首先将主程序中下一条指令的地址存放在一个临时存储单元中，然后转入执行子程序，等子程序执行到最后一条指令（通常是返回指令）时，将存放在临时存储单元中的地址取出作为下一条指令地址，这样就返回了主程序。

3．程序中断指令

中断一般是在计算机系统出现异常情况或接到特殊请求时随机产生的。当产生中断时，程序转入中断处理程序。为了在应用程序中使用中断服务程序，指令系统提供了各种引起中断的指令。

4.3.5 输入输出指令

这是主机与外围设备进行信息交换的一类指令，用于启动外设、检测外设的工作状态、读写外设的数据等。信息由外围设备传向主机称为输入（Input），反之则称为输出（Output）。有些计算机对于主存和外设未采用统一编码技术，因此需要专门的输入输出操作指令；有些计算机把外设看作一个特殊的存储单元而与存储器单元统一编址，因此用一般的访问存储器的指令即可访问外设。

4.3.6 字符串处理指令

随着计算机技术的发展，计算机的应用已不再局限于数值处理，也慢慢涉及到了非数值处理的应用。字符串处理指令包括字符串传送、转换、比较、查找、匹配、替换等，这些指令的设置可以大大加快文字处理软件的运行速度，因此，在现代计算机指令系统配置中越来越重视这类指令的设计。

4.3.7 系统控制指令

这类指令用于改变计算机系统的工作状态，包括停机指令、空操作指令、条件码指令和开/关中断指令等。

当用户程序执行完毕时，可以安排一条停机指令，此时计算机不再继续执行程序。空操作指令除了递增程序计数器之外，不进行任何其他操作。条件码用来保存当前指令执行结果的特征，条件码指令对条件码进行置位或清除操作。开/关中断指令可以视作为特殊的条件码指令。开/关中断意味着对中断请求的允许或禁止，在某些计算机中可以用条件码中的一位标志位来进行设置，在其他计算机中可采用设定程序优先级的方法来实现开/关中断的功能。

除了以上提到的指令外，还有特权指令等。特权指令是指具有特殊权限的指令，主要用于系统资源的分配和管理，一般不直接提供给用户使用。

4.4　寻址方式

存储器既可用来存放指令，又可用来存放数据。在程序运行过程中，形成指令或操作数地址的方式，称为寻址方式。

寻址方式可以分为两类：指令寻址方式和数据寻址方式，前者较为简单，后者较为复杂。

4.4.1　指令寻址方式

指令的寻址方式有两种，一种是顺序寻址方式，另一种是跳跃寻址方式。

1．顺序寻址方式

由于指令地址在主存中顺序排列，当执行一段程序时，通常是一条指令接着一条指令地顺序执行。从存储器取出第一条指令，然后执行这条指令；接着从存储器取出第二条指令，再执行第二条指令；……，以此类推。这种程序顺序执行的过程称为指令的顺序寻址方式。指令顺序寻址方式的执行示意如图 4-3 所示。为此，必须使用程序计数器 PC 来计数指令的顺序号，该顺序号就是指令在主存中的地址。指令逐条顺序执行，由 PC+1→PC 控制。

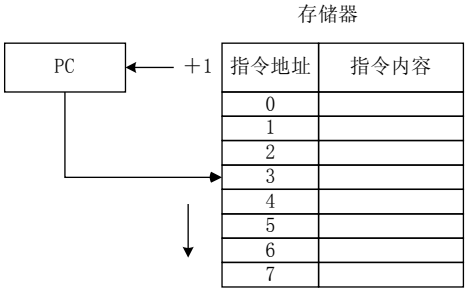


图 4-3 指令的顺序寻址方式

2．跳跃寻址方式

所谓指令的跳跃寻址，是指下一条指令的地址码不是由程序计数器给出，而是由本条指令直接给出。程序跳跃后，按新的指令地址开始顺序执行。因此，指令计数器的内容也必须相应改变，以便及时跟踪新的指令地址。指令跳跃寻址方式的执行示意如图 4-4 所示。采用指令跳跃寻址方式，可以实现程序转移或构成循环程序，从而缩短程序长度，或将某些程序作为公共程序调用。指令系统中的各种条件转移或无条件转移指令，就是为了实现指令的跳跃寻址而设置的。

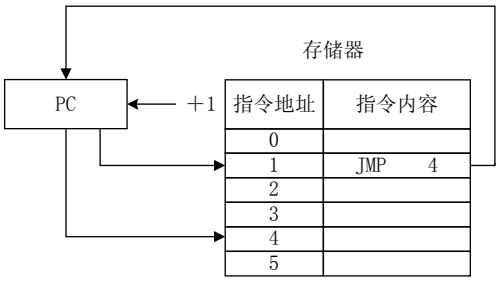


图 4-4 指令的跳跃寻址方式

4.4.2　操作数寻址方式

指令中形成操作数或操作数地址的方式称为操作数的寻址方式。一般把指令中直接给出的地址称为形式地址，从形式地址生成有效地址的各种方式称为各种不同的存储器寻址方式，每种寻址方式都有一种对形式地址进行变换处理的运算规则。

常用的操作数寻址方式有以下几种：

1．立即寻址方式

指令的地址码字段指出的不是地址，而是操作数本身，这种寻址方式称为立即寻址方式。立即寻址由于在取出指令的同时也取出了操作数，所以指令的执行速度很快。但由于操作数是指令的一部分，不便于

修改，降低了程序的通用性和灵活性。因此，立即寻址方式只适合于操作数固定的场合，通常用于为主存单元和寄存器提供常数。

2．直接寻址方式

直接寻址就是在指令的地址字段中直接指出操作数在主存中的地址，即形式地址等于有效地址，如图4-5所示。这种寻址方式简单、直观，是一种最基本的寻址方式。

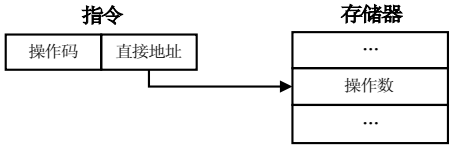


图4-5 直接寻址示意图

3．间接寻址方式

与直接寻址方式相比，间接寻址中指令地址码字段所指向的存储单元中存储的不是操作数本身，而是操作数的地址，如图4-7所示。因此，间接寻址方式需要多次访问主存储器，既增加了指令的执行时间，又要占用主存储器单元。但是，这种寻址方式也为编程人员带来了较大的灵活性，实现起来也很简便，而且，间接寻址指令可以访问较大的存储空间，从而扩大指令的寻址能力。由于地址码位数的限制，如果采用直接寻址方式，能够访问的存储空间十分有限，而间接寻址的地址码所指向的存储单元则有足够的位数，因此可以访问全部存储空间。

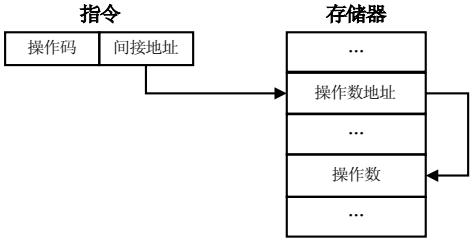


图4-7 间接寻址示意图

4．寄存器寻址方式

寄存器寻址方式就是指令中的地址码是寄存器的编号，而不是操作数地址或操作数本身。寄存器的寻址方式也可以分为直接寻址和间接寻址，两者的区别在于：前者的指令地址码给出寄存器编号，寄存器的内容就是操作数本身；而后者的指令地址码给出寄存器编号，寄存器的内容是操作数的地址，根据该地址访问主存后才能得到真正的操作数。寄存器寻址方式的优点是用寄存器来暂存操作数或其地址，无需访问主存，速度快。

5．基址寻址方式

基址寻址是将基址寄存器的内容加上指令中的形式地址而形成操作数的有效地址，其优点是可以扩大寻址能力。相对于形式地址，基址寄存器的位数可以设置得很长，从而可以在较大的存储空间中进行寻址。

6．变址寻址方式

变址寻址方式计算有效地址的方法与基址寻址方式很相似，它是将变址寄存器的内容加上指令中的形式地址而形成操作数的有效地址。使用变址寻址方式的目的不在于扩大寻址空间，而在于实现程序块的规律性变化。例如，有一个字符串存储在以AC1H为首址的连续主存单元中，只需要将首地址AC1H作为指令中的形式地址，而在变址寄存器中指出字符的序号，便可访问字符串中的任一字符。

变址寻址和基址寻址方法十分类似，但用途不同。变址寻址主要用于数组的访问，基址寻址则用于扩大寻址范围，从而在较大的存储空间中进行寻址。

7．相对寻址方式

相对寻址，是相对于当前的指令地址而言的寻址方式。相对寻址是把程序计数器PC的内容加上指令中的形式地址而形成操作数的有效地址，而程序计数器的内容就是当前指令的地址，所以相对寻址是相对于当前的指令地址而言的。此时的形式地址通常称为位移量，也就是操作数位置与当前指令位置之间的相对

距离，其值可正可负，相对于当前指令地址而浮动。在相对寻址方式中，由于指令的地址和它所涉及的操作数位置相对固定，因此，操作数与指令可以放在主存的任何地方，但仍能保证程序的正确执行。

4.4.3 堆栈寻址方式

计算机中的堆栈（Stack）是一组能存储和取出数据的暂时存储单元，所有信息的存入和取出均按照后进先出（LIFO）或先进后出（FILO）的原则进行。

堆栈存取方式决定了其“一端存取”的特点，数据按顺序存入堆栈称为进栈或压栈（Push），堆栈中一个单元的数据称为栈项，栈项按与进栈相反的顺序从堆栈中取出称为出栈或弹出（Pop），最后进栈的数据或最先出栈的数据称为栈顶元素。

1．寄存器堆栈

寄存器堆栈又称串联堆栈、硬堆栈。某些计算机在CPU中设置了一组专门用于堆栈的寄存器，每个寄存器可保存一个字的数据。因为这些寄存器直接设置于CPU中，所以它们是极好的暂存单元。CPU通过进栈指令（PUSH）把数据存入堆栈，通过出栈指令（POP）把数据从堆栈中取出。

寄存器堆栈如图4-14所示：⑴空栈表示栈顶无数据，即位于栈顶的寄存器中无可用的数据；⑵存入数据a，即把数据a存入栈顶，数据a可以来自主存、程序计数器PC等部件；⑶再存入数据b，数据b位于栈顶，先进入的数据a则移至下一个寄存器；⑷执行出栈操作，位于栈顶的数据b被取出，与此同时数据a移至栈顶。

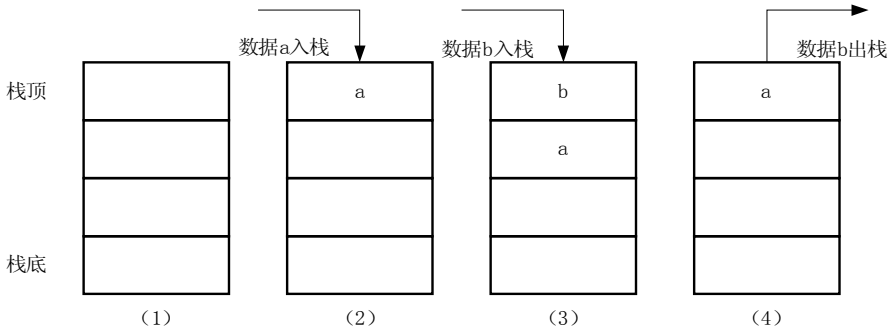


图4-14 寄存器堆栈示意图

从寄存器堆栈的数据进栈操作结果可见，最后进栈的数据位于栈顶，位于栈顶的数据出栈时最先被取出。在寄存器堆栈中，还必须有“栈空”和“栈满”的指示，以防在栈空时企图执行出栈、在栈满时企图执行进栈的误操作。这可以通过另外设置一个计数器来实现：每次进栈，计数器加1，计数值等于堆栈中寄存器个数时表示栈满；每次出栈，计数器减1，该计数值等于0时表示栈空。

寄存器堆栈的特点是仅有一个出入口，后进先出，且堆栈的容量固定，不需要占用主存。

2．存储器堆栈

当前计算机普遍采用的一种堆栈结构是存储器堆栈，也就是从主存中划出一块区域来作堆栈，又称软堆栈。这种堆栈的大小可变，栈底固定，栈顶浮动。由于主存的容量越来越大，存储器堆栈能够满足程序员对堆栈容量的要求，而且在需要时可建立多个存储器堆栈。

这种堆栈有三个主要优点：

⑴ 堆栈能够具有程序员要求的任意长度；

⑵ 只要程序员喜欢，愿意建立多少堆栈，就能建立多少堆栈；

⑶ 可以用对存储器寻址的任何一条指令来对堆栈中的数据进行寻址。

构成存储器堆栈的硬件有两部分，一是在主存中开辟用于堆栈的存储区，二是在CPU中设置一个专用的寄存器——堆栈指针SP（Stack Pointer）来保存栈顶地址。除了硬件之外，还必须有实现进栈、出栈操作的指令。

作为堆栈的存储区，其两端的存储单元有高、低地址之分，因此，存储器堆栈又可分为两种：从高地址开始生成堆栈和从低地址开始生成堆栈。

1）从高地址开始生成堆栈（自底向上生成堆栈）

从高地址开始生成堆栈是一种较常用的方式，这种堆栈的栈底地址大于栈顶地址，在建栈时，SP 指向堆栈中地址最大的单元（栈底），每次进栈时，首先把要进栈的数据存入 SP 所指向的存储单元，然后把指针 SP-1；出栈时，先把指针 SP+1，然后从 SP 所指向的存储单元取出数据，如图 4-15 所示：

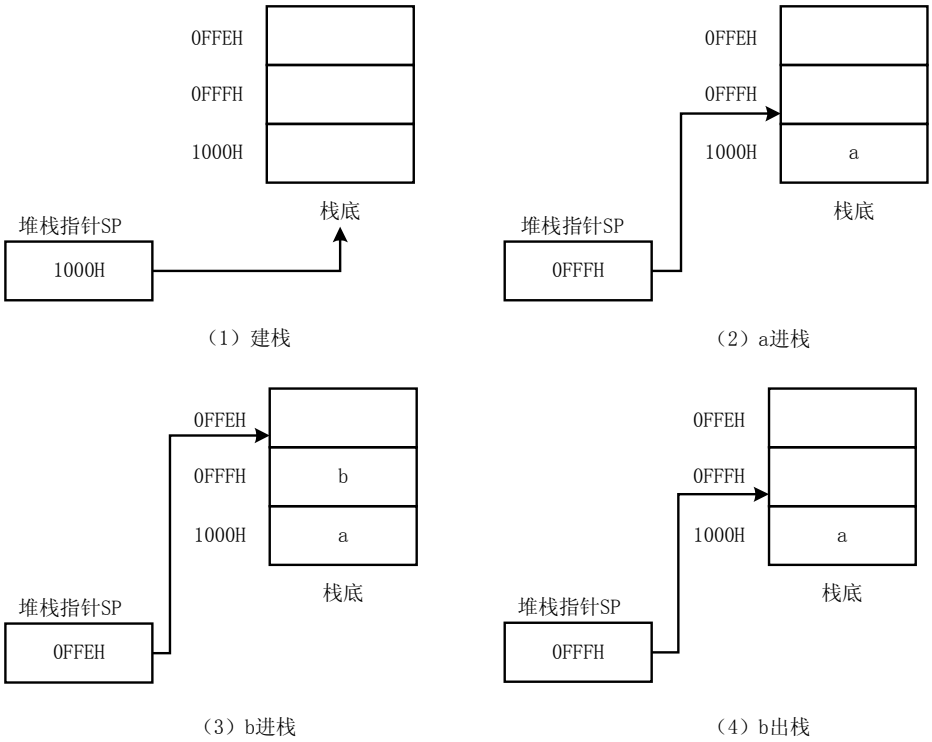


图 4-15 从高地址开始生成的存储器堆栈示意图

进栈操作：首先数据→Msp，然后指针(SP)-1→SP，Msp 表示 SP 所指定的存储单元；

出栈操作：首先(SP)+1→SP，然后(Msp)读出，(Msp)表示 SP 所指定的存储单元的内容。

2）从低地址开始生成堆栈（自顶向下生成堆栈）

这种堆栈与从高地址开始生成堆栈正好相反，它的栈底地址小于栈顶地址，建栈时 SP 指向堆栈中地址最小的单元（栈底）。具体操作如下：

进栈操作：首先数据→Msp，然后指针(SP)+1→SP；

出栈操作：首先(SP)-1→SP，然后(Msp)读出。

存储器堆栈的操作方式与寄存器堆栈不同，它移动的是栈顶，而在寄存器堆栈中移动的是数据。

堆栈中对数据的操作具有后进先出的特点，因此，凡是以后进先出方式进行的信息传送都可以用堆栈很方便地实现。例如，在子程序的调用中，用堆栈存放主程序的返回地址，实现子程序的嵌套和递归调用；在程序中断处理中，用堆栈存放多级中断的相关信息，实现多级中断的嵌套。

第5章　中央处理器（CPU）

计算机的工作过程就是计算机执行程序的过程。程序是一个指令序列，这个序列明确告诉计算机应该执行什么操作，在什么地方能够找到用来操作的数据。

一旦把程序装入主存储器，计算机就可以自动执行取出指令和执行指令的任务。专门用来完成此项工作的计算机部件称为中央处理器（Central Processing Unit，CPU），做成单片集成电路的CPU通常又称为微处理器（Microprocessor）。

计算机工业从20世纪60年代早期开始使用CPU这个术语。迄今为止，CPU从形态、设计到实现都已发生了巨大的变化，但是其基本工作原理却一直没有大的变化。

早期的CPU通常是为大型、特定的应用而定制的，这种为特定应用而设计定制CPU的昂贵方法，目前在很大程度上已经让位于开发可大规模生产的通用处理器。这种标准化趋势，大致开始于分立晶体管大型机（Mainframe）和小型机（Minicomputer）的年代，并且随着集成电路（IC）的普及而大大加速，集成电路可以把日益复杂的CPU设计制造在很小的空间里。CPU的小型化和标准化，大大增加了这些数字器件在现代生活中的应用范围，远远超出了专用运算机器这一有限的应用，现代微处理器已经随处可见，从汽车到手机，甚至儿童玩具。

5.1　CPU的功能和组成

作为控制并执行指令的部件，CPU对整个计算机系统的运行是至关重要的，它不仅要与计算机的其他功能部件进行信息交换，还要控制这些功能部件的操作。

当用计算机解决某个问题时，应当首先编写相应的程序，把程序连同原始数据预先通过输入设备送到主存储器中保存起来。计算机工作时，按顺序逐条取出指令，分析指令，执行指令，并自动转到下一条指令。计算机一条一条地执行指令，实现预先设计的程序控制，直到程序规定的任务完成为止。

5.1.1　CPU的基本功能

CPU控制整个程序的执行，具有以下基本功能：

1）程序控制

程序控制就是控制指令的执行顺序。

程序是指令的有序集合，这些指令的相互顺序不能任意颠倒，必须严格按照程序规定的顺序执行。

保证计算机按一定顺序执行程序是CPU的首要任务。

2）操作控制

操作控制就是控制指令进行操作。

一条指令的功能往往由若干个操作信号的组合来实现。因此，CPU管理并产生每条指令的操作信号，把各种操作信号送往相应的部件，从而控制这些部件按指令的要求进行操作。

3）时间控制

时间控制就是对各种操作实施定时控制。

在计算机中，各种指令的操作信号和指令的整个执行过程都受到严格定时。只有这样，计算机才能有条不紊地工作。

4）数据加工

数据加工就是对数据进行算术和逻辑运算。

完成数据的加工处理，是CPU的根本任务。

5.1.2　CPU的基本组成

传统上，CPU由控制器和运算器这两个主要部件组成。

随着集成电路技术的不断发展和进步，新型CPU纷纷集成了一些原先置于CPU之外的分立功能部件，如浮点处理器、高速缓存（Cache）等，在大大提高CPU性能指标的同时，也使得CPU的内部组成日益复杂化。

1．控制器

控制器是整个计算机系统的指挥中心。在控制器的指挥控制下，运算器、存储器和输入/输出设备等部件协同工作，构成一台完整的通用计算机。

控制器根据程序预定的指令执行顺序，从主存取出一条指令，按照该指令的功能，用硬件产生带有时序标志的一系列微操作控制信号，控制计算机内各功能部件的操作，协调和指挥整个计算机实现指令的功能。

控制器通常由程序计数器（PC）、指令寄存器（IR）、指令译码器（ID）、时序发生器和操作控制器组成。其主要功能包括：

⑴从主存中取出一条指令，并指出下一条指令在主存中的位置；

⑵对指令进行译码，并产生相应的操作控制信号，以便启动规定的动作；

⑶指挥并控制CPU、主存和输入/输出设备之间数据流动的方向。

2．运算器

运算器是计算机中用于实现数据加工处理功能的部件，它接受控制器的命令，负责完成对操作数据的加工处理任务，其核心部件是算术逻辑单元ALU。

相对控制器而言，运算器接受控制器的命令而进行动作，即运算器所进行的全部操作都是由控制器发出的控制信号来指挥的，所以它是执行部件。

运算器由算术逻辑单元（ALU）、累加寄存器（AC）、数据寄存器（DR）和程序状态字寄存器（PSW）组成。它有两个主要功能：

⑴执行所有的算术运算；

⑵执行所有的逻辑运算，并进行逻辑测试。

5.1.3 CPU中的主要寄存器

在CPU中至少要有六类寄存器：指令寄存器（IR）、程序计数器（PC）、地址寄存器（AR）、数据寄存器（DR）、累加寄存器（AC）、程序状态字寄存器（PSW）。这些寄存器用来暂存一个计算机字，其数目可以根据需要进行扩充。

1. 数据寄存器

数据寄存器（Data Register，DR）又称数据缓冲寄存器，其主要功能是作为CPU和主存、外设之间信息传输的中转站，用以弥补CPU和主存、外设之间操作速度上的差异。

数据寄存器用来暂时存放由主存储器读出的一条指令或一个数据字；反之，当向主存存入一条指令或一个数据字时，也将它们暂时存放在数据寄存器中。

数据寄存器的作用是 ：

⑴作为CPU和主存、外围设备之间信息传送的中转站；

⑵弥补CPU和主存、外围设备之间在操作速度上的差异；

⑶在单累加器结构的运算器中，数据寄存器还可兼作操作数寄存器。

2. 指令寄存器

指令寄存器（Instruction Register，IR）用来保存当前正在执行的一条指令。

当执行一条指令时，首先把该指令从主存读取到数据寄存器中，然后再传送至指令寄存器。

指令包括操作码和地址码两个字段，为了执行指令，必须对操作码进行测试，识别出所要求的操作，指令译码器（Instruction Decoder，ID）就是完成这项工作的。指令译码器对指令寄存器的操作码部分进行译码，以产生指令所要求操作的控制电位，并将其送到微操作控制线路上，在时序部件定时信号的作用下，产生具体的操作控制信号。

指令寄存器中操作码字段的输出就是指令译码器的输入。操作码一经译码，即可向操作控制器发出具体操作的特定信号。

3. 程序计数器

程序计数器（Program Counter，PC）用来指出下一条指令在主存储器中的地址。

在程序执行之前，首先必须将程序的首地址，即程序第一条指令所在主存单元的地址送入PC，因此PC的内容即是从主存提取的第一条指令的地址。

当执行指令时，CPU能自动递增PC的内容，使其始终保存将要执行的下一条指令的主存地址，为取下一条指令做好准备。若为单字长指令，则(PC)+1→PC，若为双字长指令，则(PC)+2→PC，以此类推。

但是，当遇到转移指令时，下一条指令的地址将由转移指令的地址码字段来指定，而不是像通常的那样通过顺序递增PC的内容来取得。

因此，程序计数器的结构应当是具有寄存信息和计数两种功能的结构。

4. 地址寄存器

地址寄存器（Address Register，AR）用来保存CPU当前所访问的主存单元的地址。

由于在主存和CPU之间存在操作速度上的差异，所以必须使用地址寄存器来暂时保存主存的地址信息，直到主存的存取操作完成为止。

当CPU和主存进行信息交换，即CPU向主存存入数据/指令或者从主存读出数据/指令时，都要使用地址寄存器和数据寄存器。

如果我们把外围设备与主存单元进行统一编址，那么，当CPU和外围设备交换信息时，我们同样要使用地址寄存器和数据寄存器。

5. 累加寄存器

累加寄存器通常简称累加器（Accumulator，AC），是一个通用寄存器。

累加器的功能是：当运算器的算术逻辑单元ALU执行算术或逻辑运算时，为ALU提供一个工作区，可以为ALU暂时保存一个操作数或运算结果。

显然，运算器中至少要有一个累加寄存器。

6. 程序状态字寄存器

程序状态字（Program Status Word，PSW）用来表征当前运算的状态及程序的工作方式。

程序状态字寄存器用来保存由算术/逻辑指令运行或测试的结果所建立起来的各种条件码内容，如运算结果进/借位标志（C）、运算结果溢出标志（O）、运算结果为零标志（Z）、运算结果为负标志（N）、运算结果符号标志（S）等，这些标志位通常用1位触发器来保存。

除此之外，程序状态字寄存器还用来保存中断和系统工作状态等信息，以便CPU和系统及时了解机器运行状态和程序运行状态。

因此，程序状态字寄存器是一个保存各种状态条件标志的寄存器。

5.1.4 操作控制器和时序发生器

1．微操作与数据通路

控制器在实现一条指令的功能时，总是把每一条指令分解成时间上先后有序的一系列最基本、最简单、不可再分的操作控制动作，这种最基本、最简单、不可再分的操作称为微操作（Microoperation）。

我们通常把许多寄存器之间传输信息的通路称为数据通路（Data Path），它控制信息从什么地方开始，中间经过哪个寄存器或多路开关，最后传送到哪个寄存器。

在数据通路中，微操作通过自身的控制作用和彼此之间的密切配合，使指令流、数据流等信息流按照预定的路径流动，以实现指令的功能。每一条指令的功能决定了它所需要的一系列带时序的微操作信号。

2．操作控制器

控制器的基本功能是负责指令的读出、识别和解释，并指挥协调各功能部件执行指令。

操作控制器是CPU中完成取指令和执行指令全过程的部件，其主要功能是根据指令操作码和时序信号的要求，产生各种操作控制信号，以便在各寄存器之间正确地建立数据通路，从而完成取指令和执行指令的控制。

根据设计方法不同，操作控制器可分为组合逻辑控制器和微程序控制器两种，二者的区别在于其中的控制信号形成部件不同，进而反映出不同的设计原理和方法。根据使用器件的不同，组合逻辑控制器又可进一步细分为硬连线控制器和门阵列控制器。

3．时序发生器

CPU 中除了操作控制器外，还必须包括时序发生器。

由于计算机的高速工作，每一个动作的时间必须非常严格，不能有任何差错。时序发生器的作用，就是对操作控制器产生的各种控制信号实施时间上的严格控制，产生各功能部件所需要的定时控制信号。

5.2　CPU的工作过程

CPU 的基本工作是执行预先存储的指令序列（即程序）。程序的执行过程实际上是不断地取出指令、分析指令、执行指令的过程。

CPU 从存放程序的主存储器里取出一条指令，译码并执行这条指令，保存执行结果，紧接着又去取指令，译码，执行指令……，如此周而复始，反复循环，使得计算机能够自动地工作，其过程如图 5-3 所示。除非遇到停机指令，否则这个循环将一直进行下去。

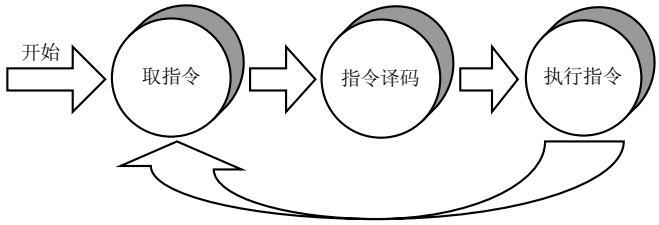


图 5-3 程序的执行过程

5.2.1　指令的执行过程

几乎所有的冯·诺伊曼型计算机的 CPU，其工作都可以分为 5 个阶段：取指令、指令译码、执行指令、访存取数、结果写回。

1．取指令阶段

取指令（Instruction Fetch，IF）阶段是将一条指令从主存中取到指令寄存器的过程。

程序计数器 PC 中的数值，用来指示当前指令在主存中的位置。当一条指令被取出后，PC 中的数值将根据指令字长度而自动递增：若为单字长指令，则(PC)+1→PC；若为双字长指令，则(PC)+2→PC，依此类推。

2．指令译码阶段

取出指令后，计算机立即进入指令译码（Instruction Decode，ID）阶段。

在指令译码阶段，指令译码器按照预定的指令格式，对取回的指令进行拆分和解释，识别区分出不同的指令类别以及各种获取操作数的方法。

在组合逻辑控制的计算机中，指令译码器对不同的指令操作码产生不同的控制电位，以形成不同的微操作序列；在微程序控制的计算机中，指令译码器用指令操作码来找到执行该指令的微程序的入口，并从此入口开始执行。

在传统的设计里，CPU 中负责指令译码的部分是无法改变的。不过，在众多运用微程序控制技术的新型 CPU 中，微程序有时是可重写的，可以通过修改成品 CPU 来改变 CPU 的译码方式。

3．执行指令阶段

在取指令和指令译码阶段之后，接着进入执行指令（Execute，EX）阶段。

此阶段的任务是完成指令所规定的各种操作，具体实现指令的功能。为此，CPU 的不同部分被连接起来，以执行所需的操作。

例如，如果要求完成一个加法运算，算术逻辑单元 ALU 将被连接到一组输入和一组输出，输入端提供需要相加的数值，输出端将含有最后的运算结果。

4．访存取数阶段

根据指令需要，有可能要访问主存，读取操作数，这样就进入了访存取数（Memory，MEM）阶段。

此阶段的任务是：根据指令地址码，得到操作数在主存中的地址，并从主存中读取该操作数用于运算。

5．结果写回阶段

作为最后一个阶段，结果写回（Writeback，WB）阶段把执行指令阶段的运行结果数据“写回”到某种存储形式：结果数据经常被写到CPU的内部寄存器中，以便被后续的指令快速地存取；在有些情况下，结果数据也可被写入相对较慢、但较廉价且容量较大的主存。许多指令还会改变程序状态字寄存器中标志位的状态，这些标志位标识着不同的操作结果，可被用来影响程序的动作。

在指令执行完毕、结果数据写回之后，若无意外事件（如结果溢出等）发生，计算机就接着从程序计数器PC中取得下一条指令地址，开始新一轮的循环，下一个指令周期将顺序取出下一条指令。

许多新型CPU可以同时取出、译码和执行多条指令，体现并行处理的特性。

5.2.2 指令周期

1．指令周期

指令周期是CPU取出一条指令并执行该指令所需的时间。

指令周期的长短与指令的复杂程度有关。

2．CPU周期

指令周期常常用若干个CPU周期数来表示。

由于CPU内部的操作速度较快，而CPU访问一次主存所花的时间较长，因此通常用从主存读取一条指令的最短时间来规定CPU周期。

CPU周期也称为机器周期。

3．时钟周期

一个CPU周期包含若干个时钟周期。

时钟周期是处理操作的最基本时间单位，由机器的主频决定。

一个CPU周期的时间宽度由若干个时钟周期的总和来决定。

图5-5为采用定长CPU周期的指令周期示意图。

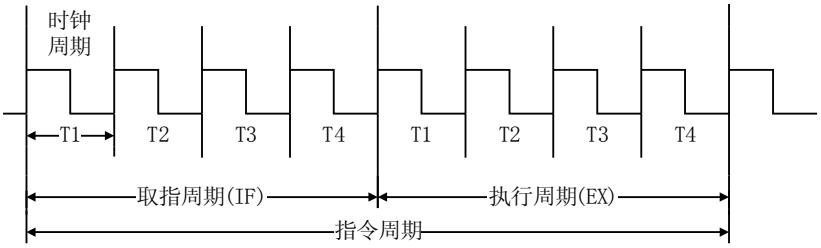


图5-5 指令周期

4．取出和执行任何一条指令所需的最短时间为2个CPU周期

任何一条指令，它的指令周期至少需要2个CPU周期，而复杂指令的指令周期则需要更多的CPU周期。这是因为，一条指令的取出阶段需要一个CPU周期时间，而一条指令的执行阶段则需要至少一个CPU周期时间。由于不同复杂度的指令其执行周期所需的CPU周期数不尽相同，因此，各种指令的指令周期也是不尽相同的。

5．用指令流程图表示指令周期

在进行计算机设计时，可以像画程序流程图那样，采用指令流程图来表示一条指令的指令周期。

在指令流程图中，

方框：代表一个操作步骤，方框中的内容表示数据通路的操作或某种控制操作；

菱形框：通常用来表示某种判别或测试，其动作依附于它前面的一个方框；

公操作符号“～”：表示一条指令已经执行完毕，转入公操作。所谓公操作，就是一条指令执行完毕后CPU进行的一些操作，这些操作主要是CPU对外设请求的处理。如果外设没有向CPU请求交换数据，那么CPU又转向主存取下一条指令。

一般的指令流程图有一个公共的流程段和许多并列的分支。公共流程段是取指令操作的流程序列，由于取指令操作是每条指令共同的操作步骤，而且指令读取步骤都是相同的，所以取指令的操作流程也是相

同的；由于每条指令在执行指令阶段的操作是互不相同的，所以在取指令阶段之后，流程就根据指令分成许多个分支，通常为每种指令都安排一个分支流程。

5.2.3 时序发生器

1．时序信号

在计算机高速运行过程中，计算机内各部件的每一个动作都必须严格遵守时间规定，不能有任何差错。

计算机内各部件的协调动作需要时间标志，而时间标志则是用时序信号来体现的。

计算机各部分工作所需的时序信号，在CPU中统一由时序发生器来产生。

2．时序发生器

CPU中的时序发生器，其功能是用逻辑电路来发出时序信号，实现时序控制，使计算机可以准确、迅速、有条不紊地工作。

时序发生器是产生控制指令周期的时序信号的部件，当CPU开始取指令并执行指令时，操作控制器利用时序发生器产生的定时脉冲的顺序和不同的脉冲间隔，提供计算机各部件工作所需的各种微操作定时控制信号，有条理、有节奏地指挥机器各个部件按规定时间动作。

从操作控制器设计方法而言，组合逻辑控制器的时序电路比较复杂，而微程序控制器的时序电路则相对简单。

5.2.4 控制方式

控制器控制一条指令运行的过程是依次执行一个确定的操作序列的过程。为了使机器能够正确执行指令，控制器必须能够按照正确的时序产生操作控制信号。

控制不同操作序列的时序信号的方法，称为控制器的控制方式。

控制方式通常分为三种：同步控制方式、异步控制方式、联合控制方式，其实质反映了时序信号的定时方式。

1. 同步控制方式

同步控制方式有时又称为固定时序控制方式或无应答控制方式，是指操作序列中每一步操作的执行，都由确定的具有基准时标的时序信号来控制，其特点是系统有一个统一的时钟，所有的控制信号均来自这个统一的时钟信号。

在同步控制方式中，在任何情况下，给定的指令在执行时所需的CPU周期数和时钟周期数都是固定不变的。

根据不同情况，同步控制方式可选取以下几种方案：

⑴采用完全统一的机器周期执行各种不同指令。显然，对简单指令和简单操作而言，这将造成时间上的浪费。

⑵采用不定长机器周期。将大多数操作安排在一个较短的机器周期内完成，而对于某些费时较多的操作，则采取延长机器周期的办法加以解决。

⑶中央控制与局部控制相结合。将大部分指令安排在固定的机器周期完成（称为中央控制），而对于少数复杂指令（如乘、除、浮点运算）则采用另外的时序进行定时（称为局部控制）。

同步控制方式设计简单，操作控制容易实现。

2. 异步控制方式

异步控制方式有时又称为可变时序控制方式或应答控制方式，是一种按每条指令、每个操作的实际需要而占用时间的控制方式，不同指令所占用的时间完全根据需要来决定。

在异步控制方式中，每条指令的指令周期既可由数量不等的机器周期数组成，也可由执行部件完成CPU要求的操作后发回控制器的应答信号来决定。也就是说，每条指令、每个操作控制信号的时间由其需要占用的时间来决定，需要多少时间就占用多少时间。

显然，用这种方式形成的操作控制序列，没有固定的CPU周期数和严格的时钟周期与之同步，所以称为异步方式。

在异步控制方式下，指令的运行效率高，但控制线路的硬件实现比较复杂。

异步控制方式在计算机中得到了广泛的应用。例如，CPU 对主存的读写、I/O 设备与主存的数据交换等一般都采用异步控制方式，以保证执行时的较高速度。

3. 联合控制方式

现代计算机系统中一般采用的控制方式是同步控制和异步控制相结合的方式，即联合控制方式。

联合控制方式的设计思想是：在功能部件内部采用同步控制方式，而在功能部件之间采用异步控制方式，并且在硬件实现允许的情况下，尽可能多地采用异步控制方式。

联合控制方式通常选取以下两种方案：

⑴大部分操作序列安排在固定的机器周期中，对某些时间难以确定的操作则以执行部件的应答信号作为本次操作的结束；

⑵机器周期的时钟周期数固定，但是各指令周期的机器周期数不固定。

5.3 操作控制器

根据设计方法不同，操作控制器可分为组合逻辑控制器和微程序控制器两种，二者的区别在于其中的控制信号形成部件不同，进而反映出不同的设计原理和方法。根据使用器件的不同，组合逻辑控制器又可进一步细分为硬连线控制器和门阵列控制器。

5.3.1 组合逻辑控制器

组合逻辑控制器包括硬连线控制器与门阵列控制器两种。

1．硬连线控制器

硬连线（Hard-wired）控制器是早期设计计算机控制器的一种方法。这种方法是把控制部件看作为产生专门固定时序控制信号的逻辑电路，而此逻辑电路以使用最少门电路和取得最高操作速度为设计目标。这种逻辑电路是一种由门电路和触发器构成的复杂逻辑网络。一旦控制部件构成后，除非重新设计和物理上对它重新连线，否则要想增加新的控制功能是不可能的。

硬连线控制器主要由组合逻辑网络、指令寄存器和指令译码器、时序发生器等部分组成，如图 5-13 所示。其中，组合逻辑网络产生计算机所需的全部操作命令，是控制器的核心。

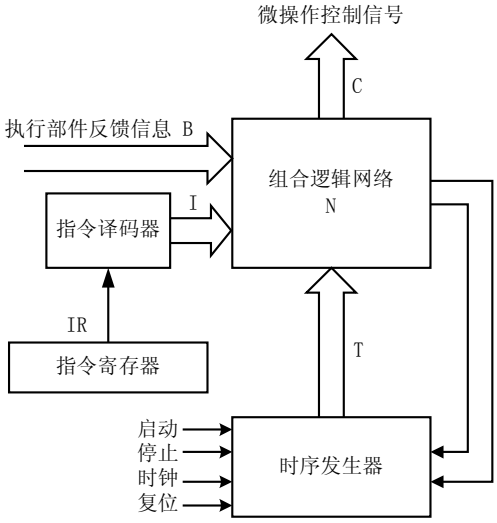


图 5-13 硬连线控制器结构方框图

组合逻辑网络的输入信号有三个来源：⑴来自指令译码器的输出 I；⑵来自执行部件的反馈信息 B；⑶来自时序发生器的时序信号 T。组合逻辑网络的输出信号就是微操作控制信号 C，它用来对执行部件的操作进行控制。

因此，组合逻辑网络输出的微操作控制信号 C，就是以上输入信号的逻辑函数，即：

$$C = f(I,B,T)$$

硬连线控制器的设计步骤如下：首先根据各条指令的功能要求，按照给出的数据通路，编写每条指令的操作流程；然后根据全部指令的操作流程，并与适当的时序信号相结合，写出每个微操作控制信号的逻辑表达式，并进行化简；最后按此逻辑表达式，用与门、或门和非门等逻辑门电路及触发器来产生微操作控制信号。

2．门阵列控制器

由大量的与门、或门阵列等电路构成的器件，称为门阵列（Gate Array）器件。典型代表产品包括：可编程逻辑阵列（Programmable Logic Array，PLA）、可编程阵列逻辑（Programmable Array Logic，PAL）、通用阵列逻辑（Generic Array Logic，GAL）等。

用门阵列器件设计的操作控制器，称为门阵列控制器，其工作原理与硬连线控制器基本相同，但门阵列控制器用门阵列器件代替硬连线控制器中的组合逻辑网络。

用门阵列实现微操作信号发生器时，把操作码、时序信号和状态条件作为门阵列的输入，按一定的“与”、“或”关系编排后，其输出便是微操作控制信号。

显然，门阵列控制器也是一种组合逻辑控制器，但是与常规的硬连线控制器不同，它是可编程的，并且不需要把一系列门电路和触发器通过硬连线组织起来。

门阵列控制器的设计步骤如下：首先根据各条指令的功能要求，按照给出的数据通路，编写每条指令的操作流程；然后根据全部指令的操作流程，并与适当的时序信号相结合，写出每个微操作控制信号的逻辑表达式，并进行化简；最后按此逻辑表达式，用门阵列器件来产生微操作控制信号。

3．组合逻辑控制的特点

组合逻辑控制方法包括硬连线方法与门阵列方法两种。

硬连线方法是分立元件时代的产物，采用这种方法的一项重要指标是尽量减少所用的逻辑门数目，以降低成本。但这样造成控制器结构不规整，各种操作控制信号以明显的随机形式散布在整个计算机中，不便于维修，可靠性低，并且造价高。

而门阵列方法则是用大规模集成电路来实现上述随机逻辑，从而克服了前者的缺点。

组合逻辑控制的特点如下：

⑴组合逻辑控制的设计和调试均非常复杂，且代价很大。

⑵与微程序控制相比，组合逻辑控制的速度较快，其速度主要取决于逻辑电路的延迟。

因此，尽管微程序控制技术已经在现代计算机设计中被广泛采用，但是近年来在某些新型的超高速计算机结构中，又重新选用了组合逻辑控制器，或与微程序控制器混合使用。

5.3.2 微程序控制器

微程序控制器是用微程序（Microprogram）实现计算机控制的控制器，具有规整性、灵活性、可维护性等一系列优点，因而在计算机设计中被广泛采用。

1．基本思想

微程序控制器的基本思想：将程序设计的思想方法引入控制器的控制逻辑，将微操作控制信号按一定规则进行编码，形成微指令，存放到一个只读存储器里；当机器运行时，逐条读出这些微指令，从而产生全机所需要的各种操作控制信号，使相应部件执行所规定的操作。

微程序控制技术，其实质是用程序设计的思想方法来组织操作控制逻辑。存放微程序的存储器称为控制存储器（Control Memory，CM，简称控存）。由于用微程序实现计算机的机器指令功能时，微程序是存储在控制存储器之中的，因此，改变控制存储器的内容就可以方便地改变指令特性、增删指令、甚至改变指令系统，这给计算机设计者和用户提供了相当大的灵活性。

在计算机系统中，微程序控制技术是利用软件方法来设计硬件的一项技术，能使机器逻辑设计规整，同时提高可靠性、可利用性和可维护性。

微程序开发在许多方面类似于软件开发，所以，软件工程中行之有效的一系列开发手段都可应用于微程序的开发上。

2．基本概念

1）微命令

由微程序控制器通过控制线向执行部件发出的微操作控制信号称为微命令（Microorder）。

执行部件接受微命令后所进行的操作就是微操作，微操作是计算机中最基本的操作。

2）微指令

在一个CPU周期中，实现一定操作功能的一组微命令的集合构成一条微指令（Microinstruction）。微指令存放在控制存储器中。

3）微地址

微地址（Microaddress）就是微指令在控制存储器中的地址。

4）微程序

一条机器指令的功能是用若干条微指令组成的序列来实现的，这个微指令序列通常称为微程序（Microprogram）。换句话说，微程序是由微指令组成、用以实现指令功能的程序。

由此可见，微命令按照一定的要求组合成微指令，微指令按照指令功能的要求组合成微程序，一条机器指令的功能是用一段微程序来实现的，机器指令执行的过程就是微程序执行的过程，而微程序的总和便可实现整个指令系统的功能。

微程序控制器将有关微操作控制信号写成微指令，若干微指令组成一个微程序，所有微程序都存放在控制存储器中，读出一条微指令就产生一组微操作控制信号，从而将原来的组合逻辑变成了存储逻辑，并且可用类似程序设计的方法来设计控制逻辑。

5.3.3 组合逻辑控制器与微程序控制器的比较

1．实现方式

从实现方式上说，组合逻辑控制方法由逻辑门电路组合实现，而微程序控制器的控制功能则是在存放微程序的控制存储器和存放当前微指令的微指令寄存器的直接控制之下实现的。

组合逻辑控制器的控制信号首先用逻辑表达式列出，经过简化后用门电路或门阵列器件实现，因而显得较为复杂，当修改指令或增加指令时非常麻烦，有时甚至没有可能。

微程序控制器的结构比较规整，大大减少了控制器的复杂性和非标准化程度，可以把硬件的用量限制在很小的范围内。由于各条指令控制信号的差别都反映在微程序上，因此，增加或修改指令只要增加或修改控制存储器中的内容即可，从而提供了很大的灵活性，使得控制器设计的变更、修改以及指令系统的扩充都不再成为难事。

因此，微程序控制得到了广泛的应用，尤其是指令系统复杂的计算机，一般都采用微程序控制方式来实现控制功能。

2．性能

从性能上来比较，在同样的半导体工艺条件下，组合逻辑控制方式比微程序控制的速度快。这是因为执行每一条微指令都要从控制存储器中读取一次微指令，从而影响了速度，而组合逻辑控制方式的速度则仅取决于电路的延迟。

因此，在超高速计算机的设计中，往往采用组合逻辑控制方法。

5.4 流水线技术

为了充分发挥计算机的效能，满足人们不断增长的应用需求，近几十年来，CPU的新技术层出不穷。其中，基于时间并行原理的流水线技术，使计算机系统结构产生了重大的变革。CPU技术的进一步发展，还包括优化编译，采用好的指令调度算法，重新组织指令执行顺序，降低相关技术带来的干扰，以及开发多发射技术（即设法在一个时钟周期内发出多条指令）等等。

5.4.1 并行处理技术概述

早期的计算机采用的是串行处理，计算机的各个操作只能串行地完成，即任一时刻只能进行一个操作。并行处理使得多个操作能同时进行，从而大大提高了计算机的速度。

广义地讲，并行性有两种含义：

一是同时性，指两个以上事件在同一时刻发生。

二是并发性，指两个以上事件在同一时间间隔内发生。

计算机的并行处理技术，概括起来主要有三种形式：

1. 时间并行

时间并行指时间重叠，就是让多个处理过程在时间上相互错开，轮流重叠地使用同一套硬件设备的各个部分，以加快硬件周转而赢得速度。

时间并行的实现方式就是采用流水处理部件，这是一种非常经济实用的并行技术，能保证计算机系统具有较高的性能价格比。目前的高性能计算机几乎无一例外地使用了流水技术。

2. 空间并行

空间并行指资源重复，就是以资源的重复配置来大幅度提高计算机的处理速度。

大规模和超大规模集成电路的迅速发展，为空间并行技术带来了巨大生机，因而成为目前实现并行处理的一个主要途径。

空间并行技术主要体现在多处理器系统和多处理机系统，但是在单处理器系统中也得到了广泛应用。

3. 时间并行+空间并行

指时间重叠和资源重复的综合应用，既采用时间并行性又采用空间并行性。相对而言，这种并行技术带来的高速效益是最好的。

现代计算机往往同时具有时间并行性和空间并行性。

5.4.2 流水线技术

在任一条指令的执行过程中，各个功能部件都会随着指令执行的进程而呈现出时忙时闲的现象。要加快计算机的工作速度，就应使各个功能部件并行工作，即以各自可能的高速度同时、不停地工作，使得各部件的操作在时间上重叠进行，实现流水式作业。

从原理上说，计算机的流水线（Pipeline）工作方式就是将一个计算任务细分成若干个子任务，每个子任务都由专门的功能部件进行处理，一个计算任务的各个子任务由流水线上各个功能部件轮流进行处理（即各子任务在流水线的各个功能阶段并发执行），最终完成工作。这样，不必等到上一个计算任务完成，就可以开始下一个计算任务的执行。

流水线的硬件基本结构如图 5-18 所示。流水线由一系列串联的功能部件（Si）组成，各个功能部件之间设有高速缓冲寄存器（L），以暂时保存上一功能部件对子任务处理的结果，同时又能够接受新的处理任务。在一个统一的时钟（C）控制下，计算任务从功能部件的一个功能段流向下一个功能段。在流水线中，所有功能段同时对不同的数据进行不同的处理，各个处理步骤并行地操作。

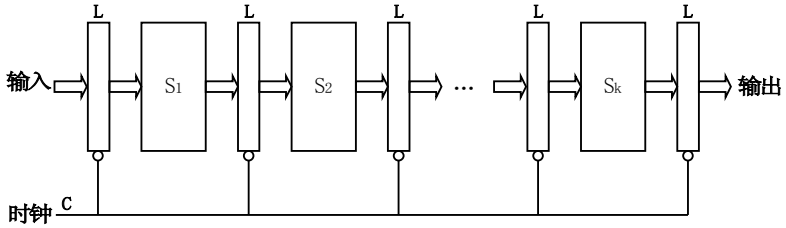


图 5-18 流水线的硬件基本结构

当任务连续不断地输入流水线时，在流水线的输出端便连续不断地输出执行结果，流水线达到不间断流水的稳定状态，从而实现了子任务级的并行。

当指令流不能顺序执行时，流水过程会中断（即断流）。为了保证流水过程的工作效率，流水过程不应经常断流。在一个流水过程中，实现各个子过程的各个功能段所需要的时间应该尽可能保持相等，以避免产生瓶颈，导致流水线断流。

流水线技术本质上是将一个重复的时序过程分解成若干个子过程，而每一个子过程都可有效地在其专用功能段上与其他子过程同时执行。采用流水线技术通过硬件实现并行操作后，就某一条指令而言，其执行速度并没有加快，但就程序执行过程的整体而言，程序执行速度大大加快。

流水线技术适合于大量的重复性的处理。

5.4.3 流水线的分类

1．按级别分类

一个计算机系统，可以在不同的并行等级上采用流水线技术。按照流水的级别，可以把流水线分为以下几类：

1）算术流水线

算术流水线指运算操作步骤的并行，它是部件级流水线。

我们可以把处理器的算术逻辑部件分段，使各种数据类型均能进行流水操作，如流水加法器、流水乘法器、流水除法器等。也可以将具体的算术逻辑运算分成多个阶段，分别由不同的部件实现，例如，可将浮点加法操作分成求阶差、对阶、尾数相加以及结果规格化 4 个子过程来进行流水处理。

现代计算机中已广泛采用了流水的算术运算器。

2）指令流水线

指令流水线表示指令步骤的并行，它是处理器级流水线。

通常可以将指令的执行过程划分为取指令、译码、执行、取数、写回 5 个并行处理的过程段，并按流水方式组织起来，形成指令流水线。

目前，几乎所有的高性能计算机都采用了指令流水线。

3）处理机流水线

处理机流水线指程序步骤的并行，又称为宏流水线。

处理机流水线由一串级联的处理机构成流水线的各个过程段，每台处理机负责某一特定的任务。数据流从第一台处理机输入，经处理后被送入与第二台处理机相连的缓冲存储器中，第二台处理机从该存储器中取出数据进行处理，然后传送给第三台处理机，……，如此一直串联下去。

处理机流水线大多应用在多机系统中，但随着高档微处理器芯片的出现，构造处理机流水线现在变得更为容易了。

2．按数据分类

按照数据表示，流水线可分为标量流水线和向量流水线两种。

1）标量流水线

只能对标量数据进行流水处理。

2）向量流水线

具有向量指令，能对向量数据的各元素进行流水处理。

5.4.4 流水计算机的组成

现代流水计算机系统的组成，包括存储器体系和流水 CPU 两大部分，如图 5-19 所示。

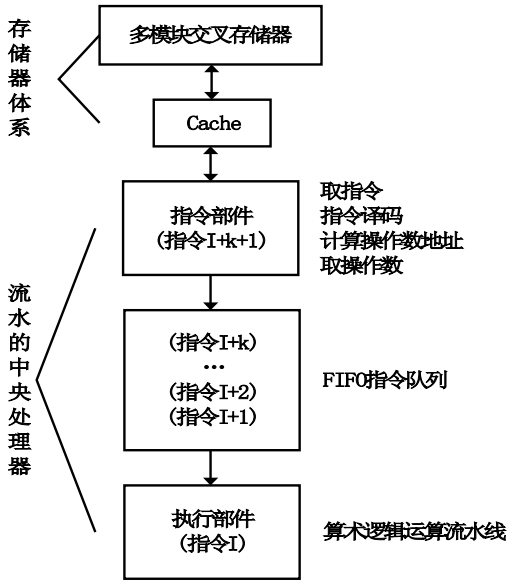


图 5-19 流水计算机系统组成原理示意图

1．存储器体系

为了解决存储器的速度匹配问题，使存储器的存取时间与流水线其他各过程段的速度相匹配，一般都采用多模块交叉存储器。在现有的流水线计算机中，存储器几乎都是采用交叉存取的方式工作的。

另一方面，高速缓存 Cache 的普遍采用，也大大提高了 CPU 对存储器的访问速度。

2．流水 CPU

CPU 内部通常按流水线方式进行组织，由指令部件、指令队列、执行部件 3 部分组成，这 3 个功能部件可以组成一个 3 级流水线。

1）指令部件

指令部件本身又构成一个流水线，即指令流水线，由取指令、指令译码、执行指令、访存取数、结果写回等几个过程段组成。

2）指令队列

指令队列是一个先进先出（FIFO）的寄存器栈，用于存放经过译码的指令和取来的操作数，同时也是由若干个过程段组成的流水线。

3）执行部件

执行部件可以具有多个算术逻辑运算部件，这些部件本身又用流水线方式构成。

由图 5-19 可见，当执行部件正在执行第 I 条指令时，指令队列中存放着 I+1，I+2，…，I+k 条指令，而与此同时，指令部件正在取第 I+k+1 条指令。

执行段的速度匹配问题，通常采用并行的运算部件以及部件流水线的方式来解决。一般采用的方法包括：

①将执行部件分为定点执行部件和浮点执行部件两个可并行执行的部分，分别处理定点运算指令和浮点运算指令；

②在浮点执行部件中，包括浮点加法部件和浮点乘/除法部件，它们可以同时执行不同的指令；

③浮点运算部件均以流水线方式工作。

5.4.5 流水计算机的时空图

描述流水线的工作过程，通常采用时（间）空（间）图的方法。在时空图中，纵坐标表示指令序列，横坐标表示时间。

1．指令流水线过程段

图 5-20 表示流水 CPU 中一个指令周期的任务分解。假设指令周期包含取指令（IF）、指令译码（ID）、指令执行（EX）、访存取数（MEM）、结果写回（WB）5 个子过程（过程段），流水线由这 5 个串联的过程段组成，各个过程段之间设有高速缓冲寄存器，以暂时保存上一过程段子任务处理的结果，在统一的时钟信号控制下，数据从一个过程段流向相邻的过程段。

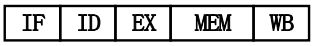


图 5-20 指令流水线过程段

2．非流水计算机工作方式

图 5-21 表示非流水计算机的时空图。

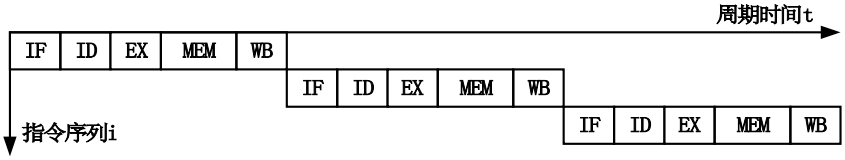


图 5-21 非流水计算机时空图

对于非流水计算机而言，上一条指令的 5 个子过程全部执行完毕后才能开始下一条指令，每隔 5 个时钟周期才有一个输出结果。因此，图 5-21 中用了 15 个时钟周期才完成 3 条指令，每条指令平均用时 5 个时钟周期。

非流水线工作方式的控制比较简单，但部件的利用率较低，系统工作速度较慢。

3．标量流水计算机工作方式

标量（Scalar）流水计算机是只有一条指令流水线的计算机。图 5-22 表示标量流水计算机的时空图。

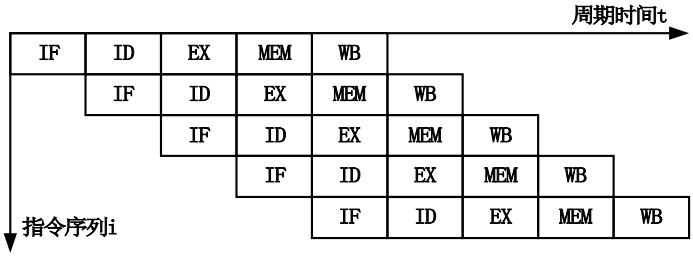


图 5-22 标量流水计算机时空图

对标量流水计算机而言，上一条指令与下一条指令的 5 个子过程在时间上可以重叠执行，当流水线满载时，每一个时钟周期就可以输出一个结果。因此，图 5-22 中仅用了 9 个时钟周期就完成了 5 条指令，每条指令平均用时 1.8 个时钟周期。

采用标量流水线工作方式，虽然每条指令的执行时间并未缩短，但 CPU 运行指令的总体速度却能成倍提高。当然，作为速度提高的代价，需要增加部分硬件才能实现标量流水。

4．超标量流水计算机工作方式

一般的流水计算机因只有一条指令流水线，所以称为标量流水计算机。所谓超标量（Superscalar）流水计算机，是指它具有两条以上的指令流水线。图 5-23 表示超标量流水计算机的时空图。

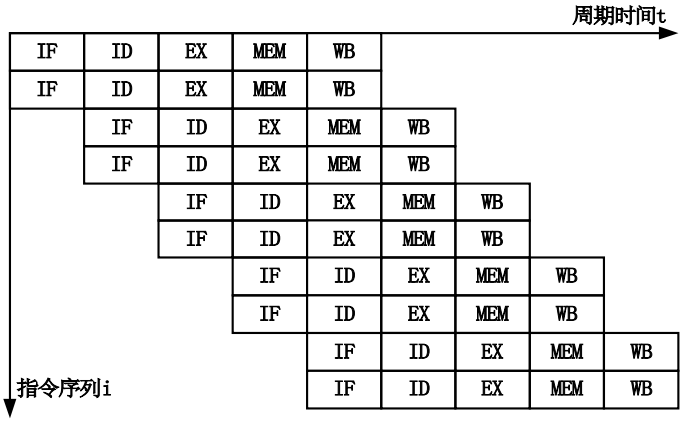


图 5-23 超标量流水计算机时空图

当流水线满载时，每一个时钟周期可以执行 2 条以上的指令。因此，图 5-23 中仅用了 9 个时钟周期就完成了 10 条指令，每条指令平均用时 0.9 个时钟周期。

超标量流水计算机是时间并行技术和空间并行技术的综合应用。

5.4.6 指令的相关性

指令流水线的一个特点是流水线中的各条指令之间存在一些相关性，使得指令的执行受到影响。

要使流水线发挥高效率，就要使流水线连续不断地流动，尽量不出现断流情况。然而，由于流水过程中存在的相关性冲突，断流现象是不可避免的。

1．数据相关

在流水计算机中，指令的处理是重叠进行的，前一条指令还没有结束，第二、三条指令就陆续开始工作。由于多条指令的重叠处理，当后继指令所需的操作数刚好是前一指令的运算结果时，便发生数据相关冲突。由于这两条指令的执行顺序直接影响到操作数读取的内容，必须等前一条指令执行完毕后才能执行后一条指令。在这种情况下，这两条指令就是数据相关的。因此，数据相关是由于指令之间存在数据依赖性而引起的。

根据指令间对同一寄存器读和写操作的先后次序关系，可将数据相关性分为写后读（Read-After-Write，RAW）相关、读后写（Write-After-Read，WAR）相关、写后写（Write-After-Write，WAW）相关三种类型。

解决数据相关冲突的办法如下：

⑴采用编译的方法

编译程序通过在两条相关指令之间插入其他不相关的指令（或空操作指令）而推迟指令的执行，使数据相关消失，从而产生没有相关性的程序代码。这种方式简单，但降低了运行效率。

⑵由硬件监测相关性的存在，采用数据旁路技术设法解决数据相关

当前一条指令要写入寄存器而下一条指令要读取同一个寄存器时，在前一条指令执行完毕、结果数据还未写入寄存器前，由内部数据通路把该结果数据直接传递给下一条指令，也就是说，下一条指令所需的操作数不再通过读取寄存器获得，而是直接获取。这种方式效率较高，但控制较为复杂。

2．资源相关

所谓资源相关，是指多条指令进入流水线后在同一机器周期内争用同一个功能部件所发生的冲突。

例如，在图 5-22 所示的标量流水计算机中，在第 4 个时钟周期时，第 1 条指令处于访存取数（MEM）阶段，而第 4 条指令处于取指令（IF）阶段。如果数据和指令存放在同一存储器中，且存储器只有一个端口，这样便会发生这两条指令争用存储器的资源相关冲突。

因为每一条指令都可能需要 2 次访问存储器（读指令和读写数据），在指令流水过程中，可能会有 2 条指令同时需要访问存储器，导致资源相关冲突。

解决资源相关冲突的一般办法是增加资源，例如增设一个存储器，将指令和数据分别放在两个存储器中。

3．控制相关

控制相关冲突是由转移指令引起的。当执行转移指令时，依据转移条件的产生结果，可能顺序取下一条指令，也可能转移到新的目标地址取指令。若转移到新的目标地址取指令，则指令流水线将被排空，并等待转移指令形成下一条指令的地址，以便读取新的指令，这就使得流水线发生断流。

为了减小转移指令对流水线性能的影响，通常采用以下两种转移处理技术：

⑴延迟转移法

由编译程序重排指令序列来实现。其基本思想是“先执行再转移”，即发生转移时并不排空指令流水线，而是继续完成下几条指令。如果这些后继指令是与该转移指令结果无关的有用指令，那么延迟损失时间片正好得到了有效的利用。

⑵转移预测法

用硬件方法来实现。依据指令过去的行为来预测将来的行为，即选择出现概率较高的分支进行预取。通过使用转移取和顺序取两路指令预取队列以及目标指令Cache，可将转移预测提前到取指令阶段进行，以获得良好的效果。

5.5 CPU新技术（上）

5.5.1 SIMD技术

单指令流多数据流（SIMD）是一种实现数据级并行的技术，其典型代表是向量处理器（Vector Processor）和阵列处理器（Array Processor）。

SIMD技术最初主要应用在大规模的超级计算机中，但是近些年来，小规模SIMD技术也开始在个人计算机上得到广泛应用。

SIMD技术的关键是在1条单独的指令中同时执行多个运算操作，以增加处理器的吞吐量。为此，SIMD结构的CPU有多个执行部件，但都在同一个指令部件的控制之下，中央控制器向各个处理单元发送指令，整个系统只要求有一个中央控制器，只要求存储一份程序，所有的计算都是同步的。

为了了解SIMD在性能上的优势，我们以加法指令为例进行说明：

单指令流单数据流（SISD）型CPU对加法指令译码后，执行部件先访问主存，取得第一个操作数，之后再一次访问主存，取得第二个操作数，随后才能进行求和运算；而在SIMD型CPU中，指令译码后，几个执行部件同时访问主存，一次性获得所有操作数进行运算。

这一特点使得SIMD技术特别适合于多媒体应用等数据密集型运算。

1．MMX技术

MMX（Multi-Media Extension，多媒体扩展）是Intel设计的一种SIMD多媒体指令集。作为一种多媒体扩展技术，MMX大大提高了计算机在多媒体和通信应用方面的能力，带有MMX技术的CPU适合于数据量很大的图形、图像数据处理，从而使三维图形、动画、视频、音乐合成、语音识别、虚拟现实等数据处理的速度有了很大提高。

MMX技术的优点是增加了多媒体处理能力，可以一次处理多个数据，缺点则是仅仅只能处理整型数，并且由于占用浮点数寄存器进行运算，以至于MMX指令集与x87浮点运算指令不能够同时执行，必须做密集的切换才可以正常执行，这种情况势必造成整个系统运行质量的下降。

2．SSE技术

1999年，Intel在其Pentium III微处理器中集成了SSE（Streaming SIMD Extensions）技术，有效增强了CPU浮点运算的能力。

SSE兼容MMX指令，可以通过SIMD和单时钟周期并行处理多个浮点数据来有效提高浮点运算速度，对图像处理、浮点运算、3D运算、视频处理、音频处理等诸多多媒体应用起到全面强化作用。

具有SSE指令集支持的处理器有8个128位的寄存器，每一个寄存器可以存放4个单精度（32位）浮点数。SSE同时提供了一个指令集，其中的指令允许把浮点数加载到这些128位寄存器中，这些数就可以在这些寄存器中进行算术逻辑运算，然后把结果送回主存。也就是说，SSE中的所有计算都可以针对4个浮点数一次性完成，这种批处理带来了效率的提升。

例如，考虑下面这个任务：计算一个很长的浮点型数组中每一个元素的平方根。

实现这个任务的算法一般可以写为：

```
for each f in array
{
    把f从主存加载到浮点寄存器
    计算平方根
    再把计算结果从寄存器中取出写入主存
}
```

而在采用SSE技术后，算法可以改写为：

```
for each 4 members in array //对数组中的每4个元素
{
    把数组中的这4个数加载到一个128位的SSE寄存器中
    在一个CPU指令执行周期中完成计算这4个数的平方根的操作
    把所得的4个结果取出写入主存
}
```

3．SSE2技术

2001年，Intel配合其Pentium 4微处理器，推出了SSE2（Streaming SIMD Extensions 2）指令集，扩展了SSE指令集，并可完全取代MMX。

SSE2指令集是Intel公司在SSE指令集的基础上发展起来的。相比于SSE，SSE2使用了144个新增指令，扩展了MMX技术和SSE技术，提高了诸如MPEG-2、MP3、3D图形等应用程序的运行性能。

在整数处理方面，随MMX技术引进的SIMD整数指令从64位扩展到了128 位，使SIMD整数类型操作的执行效率成倍提高；在浮点数处理方面，双精度浮点SIMD指令允许以 SIMD格式同时执行两个浮点操作，提供双精度操作支持有助于加速内容创建、财务、工程和科学应用。除SSE2指令之外，最初的SSE指令也得到增强，通过支持多种数据类型（例如双字、四字）的算术运算，支持灵活、动态范围更广的计算功能。

4．SSE3技术

2004年，Intel在其基于Prescott核心的新款Pentium 4处理器中，开始使用SSE3（Streaming SIMD Extensions 3）技术。

SSE3指令集是Intel公司在SSE2指令集的基础上发展起来的。相比于SSE2，SSE3在SSE2的基础上又增加了13条SIMD指令，以提升Intel超线程（Hyper-Threading）技术的效能，最终达到提升多媒体和游戏性能的目的。

5.5.2RISC技术

按照指令系统分类，计算机大致可以分为两类：复杂指令系统计算机（Complex Instruction Set Computer，CISC）和精简指令系统计算机（Reduced Instruction Set Computer，RISC）。CISC是CPU的传统设计模式，其指令系统的特点是指令数目多而复杂，每条指令的长度不尽相等；而RISC则是CPU的一种新型设计模式，其指令系统的主要特点是指令条数少且简单，指令长度固定。

1．CISC的产生和发展

计算机的指令系统最初只有很少一些基本指令，而其他的复杂指令全靠软件编译时通过简单指令的组合来实现。后来，越来越多的复杂指令被加入到了指令系统中，可用硬件实现复杂的运算。但是，一个新的

问题很快就产生了：一个指令系统的指令条数受到指令操作码位数的限制，如果操作码为 8 位，那么指令条数最多为 256 条（2^8），而指令的宽度则是很难增加的。

聪明的设计师们想出了一种解决方案——操作码扩展：在指令格式中，操作码后面跟的是地址码，而有些指令是用不着地址码或只用少量位数的地址码的，那么就可以把操作码扩展到地址码的位置，使操作码的位数得以增加。

举个简单的例子，如果一个指令系统的操作码为 2 位，那么可以有 00、01、10、11 四条不同的指令。现在把 11 作为保留，把操作码扩展到 4 位，那么就可以有 00、01、10、1100、1101、1110、1111 七条指令，其中 1100、1101、1110、1111 这四条指令的地址码部分必须减少两位。

然后，为了达到减少地址码这一操作码扩展的先决条件，设计师们又发明了各种各样的寻址方式，如基址寻址、相对寻址等，以最大限度地压缩地址码长度，为操作码留出空间。

于是，CISC 指令系统逐渐形成。大量的复杂指令、可变的指令长度、多种寻址方式是 CISC 的特点，也是 CISC 的缺点，因为这些都大大增加了译码的难度，而在高速硬件迅猛发展的今天，复杂指令所带来的速度提升早已不及在译码上所浪费的时间了。

2．RISC 的产生

1975 年，IBM 的设计师 John Cocke 研究了当时的 IBM 370 CISC 系统，发现其中仅占总指令数 20%的简单指令却在程序调用中占据了 80%，而占指令数 80%的复杂指令却只有 20%的机会被调用到。由此，他提出了 RISC 的概念。

第一台 RISC 计算机于 1981 年在美国加州大学伯克利分校问世。20 世纪 80 年代末开始，各家公司的 RISC CPU 如雨后春笋般出现，占据了大量的市场。到了 20 世纪 90 年代，x86 的 CPU（如 Pentium）也开始使用先进的 RISC 技术。

3．RISC 的特点

RISC 的主要特点是指令长度固定，指令格式和寻址方式种类少，大多数是简单指令且都能在一个时钟周期内完成，易于设计超标量与流水线，寄存器数量多，大量操作在寄存器之间进行。

RISC 体系结构的基本思想：针对 CISC 指令系统指令种类太多、指令格式不规范、寻址方式太多的缺点，通过减少指令种类、规范指令格式、简化寻址方式，方便处理器内部的并行处理，提高 VLSI 器件的使用效率，从而大幅度地提高处理器的性能。

RISC 的目标决不是简单的缩减指令系统，而是使处理器的结构更简单，更合理，具有更高的性能和执行效率，同时降低处理器的开发成本。

由于 RISC 指令系统仅包含最常用的简单指令，因此，RISC 技术可以通过硬件优化设计，把时钟频率提得很高，从而实现整个系统的高性能。同时，RISC 技术在 CPU 芯片上设置大量寄存器，用来把常用的数据保存在这些寄存器中，大大减少对存储器的访问，用高速的寄存器访问取代低速的存储器访问，从而提高系统整体性能。

RISC 的三个要素是：⑴一个有限的简单的指令集，⑵CPU 配备大量的通用寄存器，⑶强调对指令流水线的优化。

RISC 的典型特征包括：

⑴指令种类少，指令格式规范：RISC 指令集通常只使用一种或少数几种格式，指令长度单一（一般 4 个字节），并且在字边界上对齐，字段位置（特别是操作码的位置）固定。

⑵寻址方式简化：几乎所有指令都使用寄存器寻址方式，绝不出现存储器间接寻址方式，寻址方式总数一般不超过 5 个。其他更为复杂的寻址方式，如间接寻址等，则由软件利用简单的寻址方式来合成。

⑶大量利用寄存器间操作：RISC 强调通用寄存器资源的优化使用，指令集中大多数操作都是寄存器到寄存器的操作，只有取数指令、存数指令访问存储器，指令中最多出现 RS 型指令，绝不出现 SS 型指令。因此，每条指令中访问的主存地址不会超过 1 个，访问主存的操作不会与算术操作混在一起。

⑷简化处理器结构：使用 RISC 指令集，可以大大简化处理器中的控制器和其他功能单元的设计，不必使用大量专用寄存器，特别是允许以硬连线方式来实现指令操作，以期更快的执行速度，而不必像

CISC 处理器那样使用微程序来实现指令操作。因此，RISC 处理器不必像 CISC 处理器那样设置微程序控制存储器，从而能够快速地直接执行指令。

⑸便于使用 VLSI 技术：随着 LSI 和 VLSI 技术的发展，整个处理器（甚至多个处理器）都可以放在一片芯片上。RISC 体系结构为单芯片处理器的设计带来很多好处，有利于提高性能，简化 VLSI 芯片的设计和实现。基于 VLSI 技术，制造 RISC 处理器的工作量要比 CISC 处理器小得多，成本也低得多。

⑹加强处理器的并行能力：RISC 指令集非常适合于采用流水线、超流水线和超标量技术，从而实现指令级并行操作，提高处理器的性能。目前常用的处理器的内部并行操作技术，基本上都是基于 RISC 体系结构而逐步发展和走向成熟的。

⑺RISC 技术的复杂性在于它的优化编译程序，因此软件系统开发时间比 CISC 机器要长。

4．RISC 与 CISC 的主要特征对比

RISC 与 CISC 的主要特征对比如表 5-2 所示。

表 5-2 RISC 与 CISC 的主要特征对比

比较内容	CISC	RISC
指令系统	复杂、庞大	简单、精简
指令数目	一般大于 200	一般小于 100
指令格式	一般大于 4 种	一般小于 4 种
寻址方式	一般大于 4 种	一般小于 4 种
指令字长	不固定	等长
可访存指令	不加限制	只有取数、存数指令
各种指令使用频率	相差很大	相差不大
各种指令执行时间	相差很大	绝大多数在一个周期内完成
优化编译实现	很难	较容易
程序源代码长度	较短	较长
控制器实现方式	绝大多数为微程序控制	绝大多数为硬连线控制
软件系统开发时间	较短	较长

5.5.3 超线程/多核技术

1．超线程技术

每个单位时间内，CPU 只能处理一个线程（Thread）。除非有两个核心处理单元，否则要想在单位时间内处理超过一个的线程是不可能的。

超线程 HT（Hyper-Threading）技术是在单个核心处理单元中集成两个逻辑处理单元，也就是一个实体内核（共享的运算单元），两个逻辑内核（有各自独立的处理器状态），从而可以在单位时间内处理两个分别进行整数和浮点运算的线程，模拟双内核运作。

线程是程序执行的基本原子单位，一个进程可以由多个线程组成。

在分布式程序设计中正确使用线程，能够很好地提高应用程序的性能及运行效率，其实现原理是将一个进程分成多个线程，然后让它们并发异步执行，从而提高运行效率。

所谓“并发执行”，在单核情形下并不是各线程同时执行（占有 CPU），在任意时刻还是只能有一个线程占用 CPU，只不过它们彼此轮换使用 CPU 相对频繁一些，感觉上似乎都在运行。

下面通过一个简单的例子来说明超线程的工作原理。

设一个进程要完成两个任务：任务 1 和任务 2，并且任务 1 要经历 A1→B1→C1 三个步骤才能完成，任务 2 要经历 A2→B2→C2 三个步骤才能完成。

①如果两个任务同步执行的话，完成两个任务是这样执行的：

花费时间段:　1　→　2　→　3　→　4　→　5　→　6

A1　→　B1　→　C1　→　A2　→　B2　→　C2

这样从 A1 到 C2 只能一个一个地执行，当 A1 执行时，CPU 被占用，B1 到 C2 的线程只能等待，甚至当它们彼此之间并不竞争同一个资源时，也要等待前面的线程执行完毕后才能执行。

②如果两个任务异步执行的话，完成两个任务是这样执行的：

花费时间段:　1　2　3　4　5　6

A1　→　B1　→　C1

A2　→　B2　→　C2

这样，任务 1 和任务 2 就分成两个独立的执行对象，也就是说： A1→B1→C1 和 A2→B2→C2 是并发执行的。当 A1 在执行某个运算时，A2 线程可以去做其他的一些事情，比如访问磁盘等外部设备等。

对比①和②两种执行方式，完成所有任务，方式①需要 6 个时间段，而方式②只需要 3 个时间段，方式②所需时间是方式①的一半，所以方式②完成整个任务要快于方式①。

2．多核技术

多核（Multi-Core）是指在一片处理器中包含两个或两个以上的独立的内核，可以在单位时间内同时处理多个线程，如图 5-25 所示。

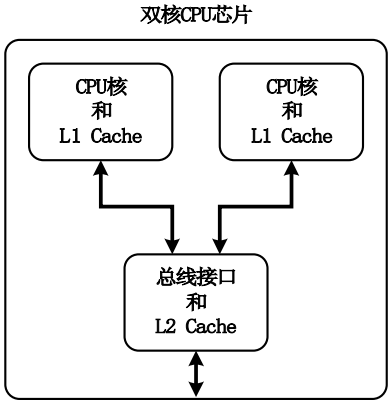


图 5-25 多核 CPU 示意图

多核技术的开发源于这样的认识：在单核芯片中，仅仅提高芯片的速度会产生过多的热量，且无法带来所预期的性能改善。

多核处理器是单片芯片，能够直接插入单一的处理器插槽中，但操作系统会利用所有相关的资源，将它的每个执行内核作为分立的逻辑处理器使用。通过在两个执行内核之间划分任务，多核处理器可以在特定的时钟周期内执行更多的任务。

处理器的超线程/多核发展，如图 5-26 所示。

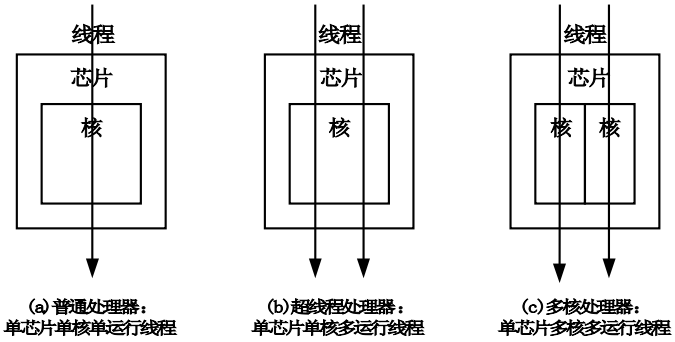


图 5-26 处理器的超线程/多核发展示意图

5.6 CPU新技术（下）

5.6.1 动态执行技术

1．指令调度

为了减少指令相关性对执行速度的影响，可以在保证程序正确性的前提下，调整指令的顺序，即进行指令调度。

指令调度可以由编译程序进行，也可以由硬件在执行的时候进行，分别称为静态指令调度和动态指令调度。静态指令调度是指编译程序通过调整指令的顺序来减少流水线的停顿，提高程序的执行速度；动态指令调度用硬件方法调度指令的执行以减少流水线停顿。

流水线中一直采用的有序（in-order）指令启动是限制流水线性能的主要因素之一。如果有一条指令在流水线中停顿了，则其后的指令就都不能向前流动了，这样，如果相邻的两条指令存在相关性，流水线就将发生停顿，如果有多个功能部件，这些部件就有可能被闲置。消除这种限制流水线性能的因素从而提高指令执行速度，其基本思想就是允许指令的执行是无序的（out-of-order，也称乱序），也就是说，在保持指令间、数据间的依赖关系的前提下，允许不相关的指令的执行顺序与程序的原有顺序有所不同，这一思想是实行动态指令调度的前提。

2．乱序执行技术

乱序执行（Out-of-order Execution）是以乱序方式执行指令，即CPU允许将多条指令不按程序规定的顺序而分开发送给各相应电路单元进行处理。这样，根据各个电路单元的状态和各指令能否提前执行的具体情况分析，将能够提前执行的指令立即发送给相应电路单元予以执行，在这期间不按规定顺序执行指令；然后由重新排列单元将各执行单元结果按指令顺序重新排列。乱序执行的目的，就是为了使CPU内部电路满负荷运转，并相应提高CPU运行程序的速度。

实现乱序执行的关键在于取消传统的“取指”和“执行”两个阶段之间指令需要线性排列的限制，而使用一个指令缓冲池来开辟一个较长的指令窗口，允许执行单元在一个较大的范围内调遣和执行已译码的程序指令流。

3．分支预测

分支预测（Branch Prediction）是对程序的流程进行预测，然后读取其中一个分支的指令。采用分支预测的主要目的是为了提高CPU的运算速度。

分支预测的方法有静态预测和动态预测两类：静态预测方法比较简单，如预测永远不转移、预测永远转移、预测后向转移等等，它并不根据执行时的条件和历史信息来进行预测，因此预测的准确性不可能很高；动态预测方法则根据同一条转移指令过去的转移情况来预测未来的转移情况。

由于程序中的条件分支是根据程序指令在流水线处理后的结果来执行的，所以当CPU等待指令结果时，流水线的前级电路也处于等待分支指令的空闲状态，这样必然出现时钟周期的浪费。如果CPU能在前条指令结果出来之前就预测到分支是否转移，那么就可以提前执行相应的指令，这样就避免了流水线的空闲等待，也就相应提高了CPU的运算速度。但另一方面，一旦前条指令结果出来后证明分支预测是错误的，那么就必须将已经装入流水线执行的指令和结果全部清除，然后再装入正确的指令重新处理，这样就比不进行分支预测而是等待结果再执行新指令还要慢了。

因此，分支预测的错误并不会导致结果的错误，而只是导致流水线的停顿，如果能够保持较高的预测准确率，分支预测就能提高流水线的性能。

5.6.2多重指令启动技术

为了进一步提高指令流水线的性能，可以设法在一个时钟周期内启动多条指令，使得每个周期平均能完成多条指令，这样就构成了多重指令启动（Multi-Launch，也称多发射）的流水方式。

多重指令启动的方法有两种：动态多重指令启动和静态多重指令启动。

1．动态多重指令启动

动态多重指令启动方法是指由硬件在每个时钟周期内启动可变数量的指令，这些指令可以采用静态指令调度，也可以采用动态指令调度，常见的技术有超标量技术、超流水线技术。

1）超标量技术

超标量（Superscalar）技术就是在每个时钟周期内同时并发多条独立指令，即将两条或两条以上的指令并行编译、执行。

超标量处理器支持指令级并行，每个时钟周期可以发射多条指令（2-4条居多），这样可以使得CPU的IPC（Instruction Per Clock，每时钟指令数）＞1，从而提高CPU的处理速度。

超标量流水计算机具有两条或两条以上指令流水线，当流水线满载时，每一个时钟周期可以执行2条以上指令。采用超标量流水线工作方式，机器速度更高，但硬件也更为复杂。

2）超流水线技术

超流水线（Superpipeline）技术是使指令周期的各个子过程内部的流水线进一步细化，使其工作速度加倍，从而在一个时钟周期中执行两条或更多条指令。

超流水线技术通过将一些流水线寄存器插入到流水线各个过程段中，对流水线再分，使每段的长度近似相等，以便现有的硬件在每个周期内使用多次，即每个超流水线段都以数倍于基本时钟频率的速度运行。

在超流水结构中，由于时钟频率提高了，而功能部件的速度不变，实际上使得流水线的周期数更多，这样会使得指令相关性对流水线性能的影响更大，从而对分支预测等部件提出更高的要求，而且由于并行执行的指令数量更多，要求具有更多的功能部件或者功能部件的流水速度更高。

图 5-28 是四种标量流水技术执行指令的时空比较。

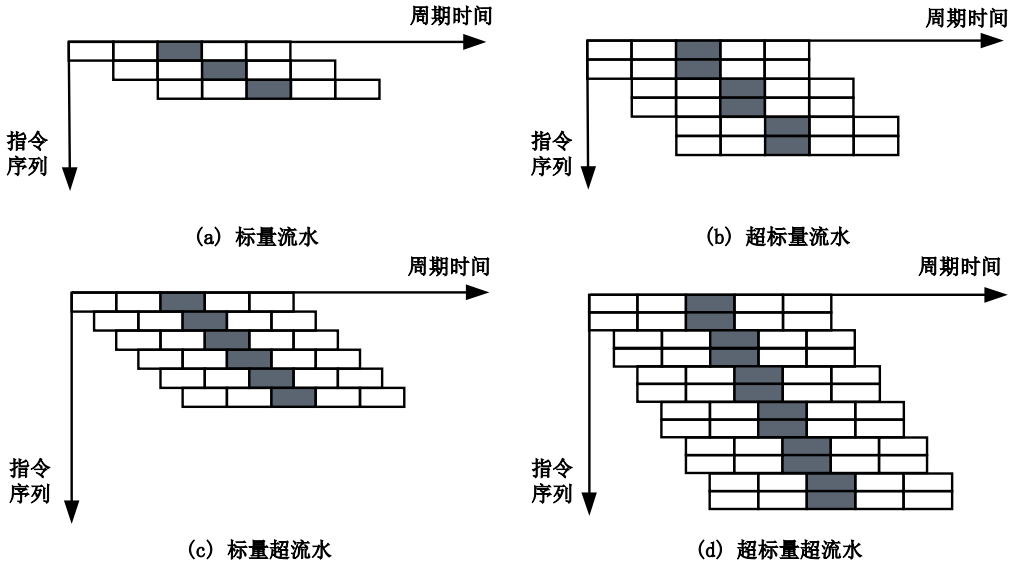


图 5-28 四种标量流水技术的时空图（阴影方框表示指令的执行阶段）

从图中可见，

①在一般标量流水中，每个时钟周期启动 1 条指令，如图 5-28(a)所示；

②在超标量流水中，每个时钟周期启动 2 条指令，如图 5-28(b)所示；

③在超流水技术中，每个时钟周期启动 2 次，每次 1 条指令，如图 5-28(c)所示；

④在超标量超流水技术中，每个时钟周期启动 2 次，每次启动 2 条指令，每个周期共启动 4 条指令，如图 5-28(d)所示。

2．静态多重指令启动

静态多重指令启动方法是指每次启动固定数量的指令，这些指令由编译程序组合成一条超长指令（或指令包），常见的技术有超长指令字技术。

超长指令字（Very Long Instruction Word，VLIW）技术将多条指令放入一个指令字，可以有效提高 CPU 各个功能部件的使用效率，提高程序性能。

首先由编译程序在编译时挖掘出指令间潜在的并行性，然后把多条能并行执行的指令组合成一条具有多个操作段的超长指令，再由这条超长指令的超长指令字来控制机器中多个独立工作的部件，每个操作段控制一个部件，相当于同时执行多条指令。

VLIW 计算机使用多个独立的功能部件，所有功能部件由同一个机器时钟来驱动，一般具有以下特点：

⑴单一控制流。机器中只有一个程序计数器、一个控制单元，每个时钟周期启动一条 VLIW 指令；

⑵指令被划分为许多字段，每段控制一个特定的功能部件；

⑶机器中设置大量的数据通路和功能部件，功能部件的操作可采用流水技术来进一步提高机器性能，每个操作的执行周期数是已知的，编译器在对操作进行调度时已经考虑了可能出现的数据相关和资源冲突，控制硬件比较简单。

在动态多重指令启动方式下，随着启动数量的增加，确定多个指令是否可同时启动的硬件的复杂性越来越大，而VLIW则可以减少实现多重启动处理所需要的硬件数量。VLIW使用多个独立的功能部件完成多个操作，并将多个操作命令包装在一个很长的指令中，将选择同时启动的多个操作的工作交给了编译程序，因此，提高VLIW计算机性能的关键在于其编译器。

5.6.3 低功耗管理技术

对于高性能通用处理器而言，低功耗研究主要解决处理器局部过热和功率过高的问题。局部过热（Hotspot）会导致芯片不能正常工作，功率过高则使得散热设备日趋昂贵，节省散热设备成本和能量损耗可以提高产品的竞争力。对于移动计算（嵌入式处理器）来说，最重要的是提高能量的效率，即计算相同的问题，使用更少的能量（一方面降低功率，一方面减少计算时间），其主要目的在于延长电池的寿命，提高产品竞争力。

1．制程提升

解决CPU的高功耗，制程的提升是最直接的改善方法。

一条粗的电阻丝比一条细的电阻丝的功耗更大。在CPU中使用了电路与各个细小元件的连接，虽然这些电路极其细微，但如果全部连接起来的话，CPU这类超大规模集成电路的线路长度将达到可观的数量级，其功耗会在这些线路中被转换成热量。制程的提升就是把这些线路变得更细，功耗可因此而大幅下降，用65nm工艺制造的奔腾CPU比90nm工艺制造的同样CPU功耗下降30W就是最好的例证。

2．降低电压

高电压是造成功耗提升的另一个重要因素，电压与功耗总是成正比关系。

在CPU中，最大功耗可由核心电压×最大电流简单计算而估得。通常CPU内部的电流都较大，而且是不易减小的，因此，虽然供给CPU的电压并不高，但与大电流相乘后，带来的功耗也是不容忽视的。所以，降低电压，即使降低的幅度不太大，所带来的功耗下降也将相当明显。但是如果电压降得过低，CPU内部的CMOS管就会变得不稳定，工作可靠性也随之大大降低。

3．减少晶体管数量

微处理器领域总是使用晶体管的数量来衡量集成技术的高低。在Prescott核心的奔腾4芯片上，晶体管数量已经达到了1.69亿的水平，比前辈Northwood核心增加了2倍以上，因此虽然工艺更先进，但功耗反而继续提升。随着多核和大缓存技术的流行，晶体管的数量也成几何速度直线增长，数以亿计的晶体管本身就是消耗能源的大户。

在相同制程下，越少的晶体管数量可以拥有越低的功耗，因此，通过优化设计，减少晶体管数量是行之有效的降低功耗手段之一。

4．降低频率

实际上，过于注重频率的提升，也是导致CPU功耗日益加大的重要因素。

之前，人们一直认为频率是衡量CPU性能的最重要标志，频率并不等于性能的说法直到近几年才被意识到。

提高频率有很多方法，如采用全新的设计、提升电压、制程提升等，但更为简单直接的却是采用超长流水线设计。在此设计中，CPU的流水线被划分得相当细密，频率提升的空间也相应增大，这就如同更细密的生产流水线拥有更高的效率一样。但是问题在于，流水线过多，其延时和错误率也会增加，最终导致CPU效率直线下降，性能反而不佳。

降低流水线等级在近几年中得到了大量的应用，如Intel启用了短流水线设计的酷睿2（Core 2）。

除此之外，降低功耗的技术还有以下几种：

高级分支预测（Advanced Branch Prediction）：采用多分支预测机制，大幅度提高预测的准确度，缩短任务执行时间，进而降低功耗。

宏指令融合（Macro-Op Fusion）：将两个宏指令归并为一个，实现“两个操作一次执行”，从而加快执行速度，降低功耗。

功耗优化总线（Power Optimized Bus）：根据需要打开或关闭处理器总线，从而降低非使用状态部分总线的能耗。

专属堆栈管理器（Dedicated Stack Manager）：通过设置硬件堆栈管理器，可以明显减少堆栈管理的微操作数，达到降低功耗的目的。

第6章 总线系统

总线是计算机系统中一个重要的子系统，如果把中央处理器比作“大脑”，那么总线就是“中枢神经”，它是计算机系统中多个功能部件之间进行数据传送的公共通路，提供了信息传输和功能扩展的通道。采用总线结构方式，主要是由于其在系统设计、生产、使用和维护方面具有诸多的优越性。本章首先讲述总线系统的基本概念和技术，然后介绍总线系统的几个实例。

6.1 总线系统概述

数字计算机是由若干个系统功能部件构成的，这些系统功能部件连接在一起才能形成完整的计算机系统。

总线是构成计算机系统的互连机构，是多个系统功能部件之间进行数据传送的公共通路，由系统中各个功能部件所共享。总线的特点在于其公用性，可同时挂接多个部件或设备。借助于总线连接，计算机在各系统功能部件之间实现地址、数据和控制信息的交换，并在争用资源的基础上进行工作。

一个单处理器系统中的总线，大致可分为三类：

⑴内部总线：CPU 内部连接各寄存器及运算部件的总线。

⑵系统总线：CPU 同计算机系统的其他功能部件（如存储器、通道等）连接的总线。系统总线有多种标准接口，从 16 位的 ISA，到 32/64 位的 PCI、AGP 乃至 PCI Express。系统总线中包括局部总线，局部总线是系统总线向多层结构发展的结果。

⑶外部总线：又称为 I/O 接口，是用来连接外部设备或其他计算机的总线，如用于连接并行打印机的 Centronics 总线，用于串行通信的 RS-232 总线、通用串行总线 USB 和 IEEE-1394，用于硬磁盘接口的 IDE、SCSI 总线等。

本章主要介绍系统总线，外部总线将在第 7 章输入输出（I/O）系统中介绍。

6.1.1 总线的基本概念

1. 总线的特性

1）物理特性

总线的物理特性是指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，引脚线的排列方式等。

2）功能特性

总线的功能特性描述总线中每一根线的功能。例如，地址总线的宽度指明了总线能够直接访问的存储器地址空间范围；数据总线的宽度指明了访问一次存储器或外设所能交换数据的位数；控制总线包括 CPU 发出的各种控制命令（如存储器读/写、I/O 读/写等），请求信号与仲裁信号，外设与 CPU 的时序同步信号，中断信号，DMA 控制信号等等。

3）电气特性

总线的电气特性定义每一根线上信号的传递方向及有效电平范围。送入 CPU 的信号叫输入信号（IN），从 CPU 发出的信号叫输出信号（OUT）。例如，IBM PC/XT 总线的 A0～A19 是地址输出线，D0～D7 是双向数据线，既可作为数据输入线又可作为数据输出线。总线的电平都符合 TTL 电平的定义。

4）时间特性

总线的时间特性定义了每根线在什么时间有效，即规定了总线上各信号有效的时序关系。

2. 总线的标准化

对于相同的指令系统、相同的功能，不同厂家生产的功能部件在具体实现上几乎没有相同的，但各厂家生产的相同功能部件却可以互换使用，这是由于它们都遵守了相同的系统总线要求，这就是系统总线的标准化问题。

例如，IBM PC 兼容微机系统中采用的标准总线，从 ISA 总线（16 位，带宽 16MB/s）发展到 EISA 总线（32 位，带宽 33MB/s）、VESA 总线（32 位，带宽 133MB/s），又发展到 PCI 总线（64 位，带宽

533MB/s）。

3. 总线的主要参数

1）总线宽度

总线宽度，指的是总线能同时传送的数据的二进制位（bit）数。如 16 位总线、32 位总线指的就是总线具有 16 位或 32 位的数据传输能力。

2）总线频率

作为总线工作速度的一个重要参数，总线频率是总线的实际工作频率，也就是一秒钟传输数据的次数，工作频率越高，速度越快。总线频率通常用 MHz 表示，如 33MHz、100 MHz、400 MHz、800 MHz 等，1Hz＝1/s。

3）总线带宽

总线带宽指的是总线本身所能达到的最高数据传输速率，单位是兆字节每秒（MB/s），这是衡量总线性能的重要指标，总线带宽越宽，传输效率也就越高。总线带宽与总线宽度和总线频率的关系为：

$$\text{□□□□}\ \ (\text{MB/s}) = \frac{\text{□□□度}\ \ (\text{bit})}{8\,(\text{bit/B})} \times \text{□□□率}\ \ (\text{MHz})$$

6.1.2 总线的内部结构

1．早期总线的内部结构

早期计算机总线的内部结构如图 6-1 所示，它实际上是处理器芯片引脚的延伸，是处理器与 I/O 设备适配器的通道。这种简单的总线一般由 50~100 根信号线所组成，按照这些信号线的功能特性可分为三类：数据（总）线、地址（总）线和控制（总）线。

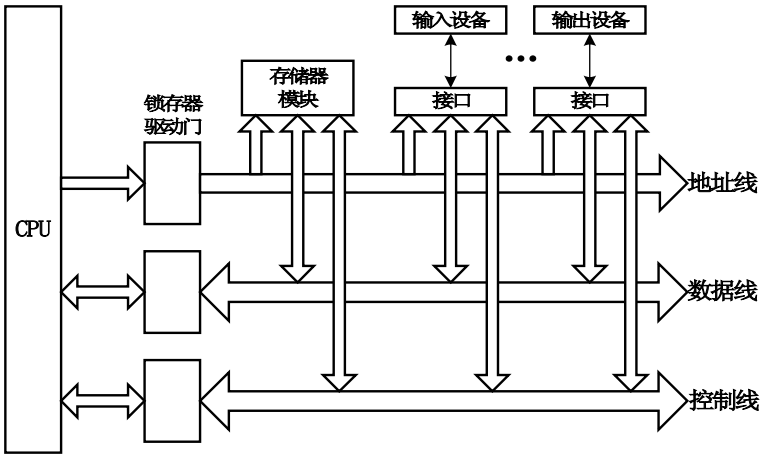


图 6-1 早期总线的内部结构

1）数据总线

数据总线（Data Bus，DB）是在计算机系统各个部件之间传输数据信息的信号线。数据总线是双向的。通常，数据总线由 8 根、16 根、32 根或 64 根数据线组成，数据线的根数称为数据总线的宽度。由于每一根数据线每次传送 1 位二进制数，所以数据线的根数决定了每一次能同时传送的二进制的位数，由此可见，数据总线的宽度是表现系统总体性能的关键因素之一。例如，如果数据总线的宽度为 8 位，而每条指令的长度为 16 位，那么在每个指令周期中需要两次访问存储器才能取回完整的 16 位指令。

2）地址总线

地址总线（Address Bus，AB）是在计算机系统各个部件之间传输地址信息的信号线，用来规定数据总线上的数据来自何处或将被送往何处。地址总线是单向的。如果 CPU 要从存储器中读取一个信息，那么，首先必须将要读取的信息的存储器地址放到地址总线上，然后才可以从给定的存储器地址中取出所需要的信息。地址总线的宽度决定了计算机系统能够使用的最大的存储器容量。在对输入输出端口进行寻址时也要使用地址总线来传送地址信息。实际操作时，总是用地址总线的高几位选择总线上指定的存储器段，而用

地址线的低几位去选择存储器段内具体的存储器单元或输入输出端口地址。

3）控制总线

控制总线（Control Bus，CB）是在计算机系统各个部件之间传输控制信息的信号线，其作用是对数据总线、地址总线的访问及使用情况实施控制。控制线中每一根线都是单向的，用来指明数据传送的方向、中断请求和定时控制等。由于计算机中的所有部件都要使用数据总线和地址总线，所以用控制总线对它们实施控制既是必要的，也是必须的。控制总线上传输的控制信息，其作用就是在计算机系统各个部件之间发送操作命令和定时信息，命令信息规定了要执行的具体操作，而定时信息则规定了数据信息和地址信息的时效性。

这种简单的总线结构被早期的计算机所广泛采用。随着计算机技术的发展，这种简单总线结构逐渐暴露出一些不足：第一，CPU 是总线上的惟一主控者，即使后来增加了具有简单仲裁逻辑的 DMA 控制器以支持 DMA 传送，但是仍不能满足多 CPU 环境的要求；第二，总线信号是 CPU 引脚信号的延伸，所以总线结构与 CPU 紧密相关，通用性较差。

2．当代总线的内部结构

当代计算机总线的内部结构如图 6-2 所示。当代总线是一些标准总线，追求与结构、CPU、技术无关的开发标准，满足包括多 CPU 在内的主控者环境需求。

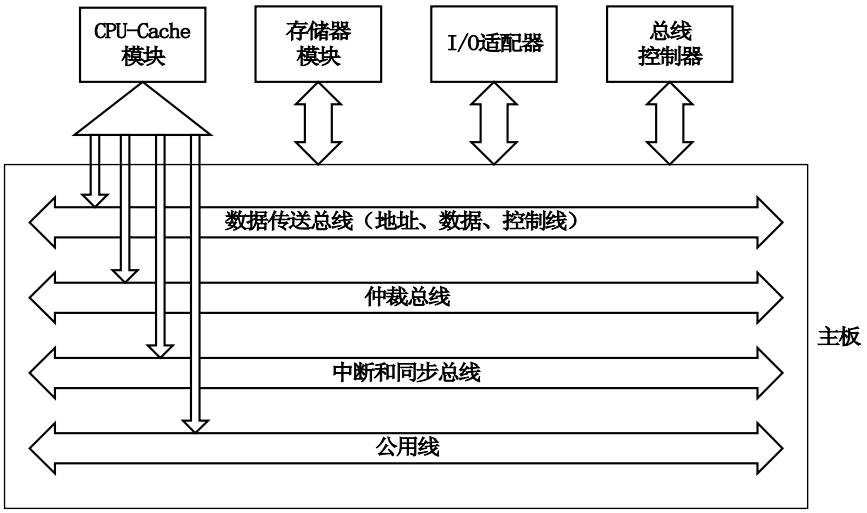


图 6-2 当代总线的内部结构

在当代总线结构中，CPU 与 Cache 作为一个模块与总线相连，系统中允许存在多个这样的处理器模块，而总线控制器则负责在几个总线请求者之间进行协调与仲裁。整个总线结构分成如下四个部分：

1）数据传送总线

数据传送总线由地址线、数据线、控制线组成，其结构与早期总线类似，但一般有 32 条地址线，32 或 64 条数据线。为了减少布线，64 位数据的低 32 位数据线往往与 32 位地址线进行复用。

2）仲裁总线

仲裁总线包括总线请求线和总线授权线。

3）中断和同步总线

中断和同步总线用于处理带优先级的中断操作，包括中断请求线和中断认可线。

4）公用线

公用线包括时钟信号线、电源线、地线、系统复位线以及加电或断电的时序信号线等。

6.1.3 总线接口

当代计算机的用途，在很大程度上取决于它所能连接的外围设备的范围。遗憾的是，由于外围设备种类繁多，速度各异，不可能简单地把外围设备全部连接到 CPU 上，必须寻找一种方法，将外围设备同某种计算机部件连接起来一同工作。这项任务通常由适配器（Adapter）部件来完成，通过适配器可以实现高

速 CPU 与低速外设之间工作速度上的匹配和同步，并完成计算机和外设之间的所有数据传送和控制。适配器通常称为接口（Interface）。

广义地讲，接口就是指 CPU 和主存、外围设备之间通过总线进行连接的逻辑部件，接口部件在动态连接的两个部件之间起着“转换器”的作用，以便实现彼此之间的信息传送。

一个典型的计算机系统具有不同类型的外围设备，因而会有不同类型的接口。图 6-3 展示了 CPU、接口和外围设备之间的连接关系：外围设备本身带有设备控制器，设备控制器是控制外围设备进行操作的控制部件，通过接口接收来自 CPU 的各种信息，并将信息传送到设备，或者从设备中读出信息传送到接口，然后由接口传送给 CPU。由于外围设备种类繁多且速度不同，因而每种设备都有适应自己工作特点的设备控制器，图中将外围设备本身与它自己的控制电路画在一起，统称为外围设备。

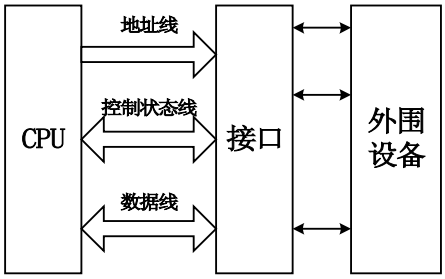


图 6-3 外围设备的连接方法

为了使所有的外围设备彼此兼容并能在一起正确地工作，CPU 规定了不同的信息传送控制方法。不管什么样的外围设备，只要选用某种数据传送方法，并按其规定通过总线和主机连接，就可以进行信息交换。通常在总线和每个外围设备的设备控制器之间使用一个适配器（接口）电路来保证外围设备用计算机系统特性所要求的形式发送和接收信息。接口逻辑通常做成标准化的部件，称为标准接口。一个标准接口可能连接一个设备，也可能连接多个设备。

典型的接口通常具有如下功能：

1）控制

接口靠程序的指令信息来控制外围设备的动作，如启动、关闭设备等；

2）缓冲

接口在外围设备和计算机系统其他部件之间起到缓冲器的作用，用以补偿各种设备在速度上的差异；

3）状态

接口监视外围设备的工作状态并保存状态信息（包括数据“准备就绪”、“忙”、“错误”等），供 CPU 查询外围设备时进行分析之用；

4）转换

接口可以完成所要求的数据转换工作（如并－串转换或串－并转换），因此数据能够在外围设备和 CPU 之间正确地进行传送；

5）整理

接口可以完成一些特别的功能，例如在需要时可以修改字计数器或当前主存地址寄存器；

6）程序中断

每当外围设备向 CPU 请求某种动作时，接口即向 CPU 发出一个中断请求信号，如果设备完成了一个操作或设备发生错误，接口也会产生中断。

事实上，一个适配器必然有两个接口：一是和系统总线的接口，CPU 和适配器进行数据交换，采用并行方式；二是和外设的接口，适配器和外设进行数据交换，可采用并行方式，也可采用串行方式。因此，根据外围设备采用的数据交换方式的不同，适配器（接口）可以分为串行数据接口和并行数据接口两大类。

6.1.4 总线的连接方式

大多数总线都是以相同方式构成的，其不同之处仅在于总线中数据线和地址线的数目，以及控制线的多少及其功能。然而，总线的排列布置、总线与其他各类部件的连接方式，对计算机系统性能而言则显得尤

其重要。根据连接方式的不同，单机系统中采用的总线结构可分成三种基本类型：单总线结构、双总线结构、三总线结构。

1. 单总线结构

在许多单处理器的计算机中，使用一条单一的系统总线来连接 CPU、主存和 I/O 设备，称为单总线结构，如图 6-4 所示：

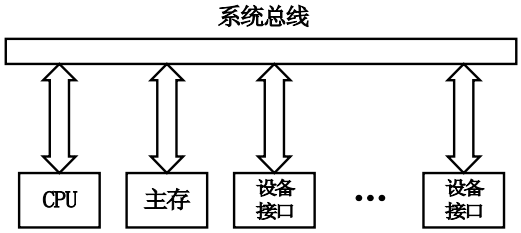


图 6-4 单总线结构

在单总线结构中，要求连接到总线上的逻辑部件都必须高速运行，以便在某些设备需要使用总线时能够迅速获得总线控制权，当不再使用总线时也能迅速放弃总线控制权。否则，由于一条总线由多个功能部件共用，有可能导致很大的时间延迟。

在单总线结构中，当 CPU 取一条指令时，首先把程序计数器 PC 中的地址同控制信息一起送至总线上。该地址不仅送至主存，同时也送至总线上的所有外围设备，然而只有与总线上的地址相对应的设备才执行数据传送操作，在取指令情况下的地址是主存地址。因此该地址所指定的主存单元中的指令被传送给 CPU，CPU 检查指令中的操作码，确定对数据执行什么操作，以及数据是流进还是流出 CPU。

在单总线系统中，对输入/输出设备的操作与主存的操作方法完全一样。当 CPU 把指令的地址字段送到总线上时，如果该地址字段对应的地址是主存地址，则主存予以响应，从而在 CPU 和主存之间发生数据传送，数据传送的方向由指令操作码决定；如果该地址字段对应的地址是外围设备地址，则外围设备予以响应，从而在 CPU 和对应的外围设备之间发生数据传送，数据传送的方向也由指令操作码决定。

单总线结构的优点在于容易扩展成多 CPU 系统，只要在系统总线上挂接多个 CPU 即可。但是，在单总线结构中，由于所有逻辑部件都挂在同一个总线上，因此总线只能分时工作，即某一个时间只能允许一对部件之间传送数据，这就使信息传送的吞吐量受到限制。

2. 双总线结构

图 6-5 所示为双总线系统结构，这种结构保持了单总线系统简单、易于扩充的优点，但又在 CPU 和主存之间专门设置了一组高速的存储总线，使 CPU 可通过专用的存储总线与存储器交换信息，以减轻系统总线的负担，同时主存仍可通过系统总线与外设进行 DMA 操作，而不必经过 CPU。当然，这种双总线系统是以增加硬件为代价的。

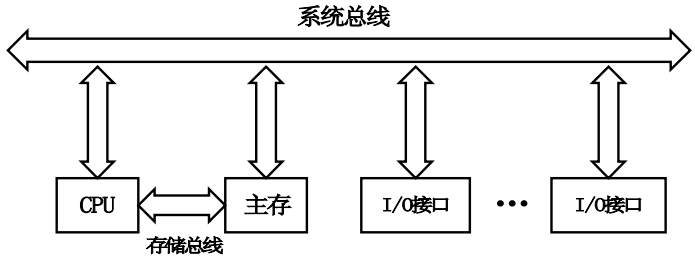


图 6-5 双总线结构

3. 三总线结构

图 6-6 所示为三总线系统结构。三总线结构是在双总线系统的基础上增加 I/O 总线形成的，其中系统总线是 CPU、主存和通道（IOP）之间进行数据传送的公共通路，而 I/O 总线则是多个外围设备与通道之间进行数据传送的公共通路。

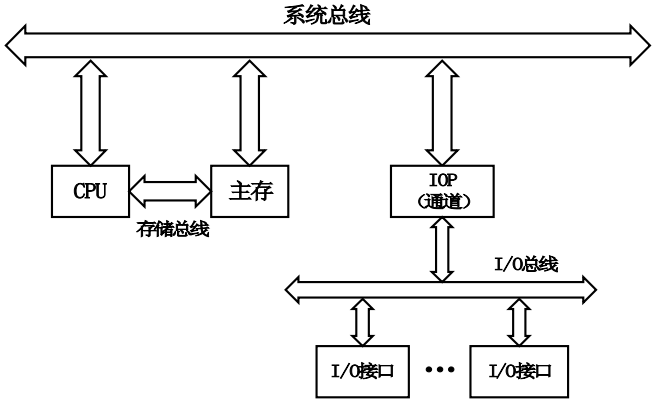


图 6-6 三总线结构

在 DMA 方式中，外设与主存间直接交换数据而不经过 CPU，从而减轻了 CPU 对数据输入输出的控制，而通道方式 I/O 则可进一步提高 CPU 的效率。通道实际上是一台具有特殊功能的处理器，又称为 IOP（I/O Processor，I/O 处理器），它分担了 CPU 的一部分功能，实现对外设的统一管理，完成外设与主存之间的数据传送。这一思想与基于总线的网络（如以太网）将集线器（Hub）转换成交换机（Switch）以提高通信速率的思想是一致的。显然，由于增加了 IOP，整个系统的工作效率可以大大提高，然而，这是以增加更多的硬件为代价的。

6.2 总线的控制与通信

6.2.1 总线的控制

总线的控制就是决定共享总线的部件如何获得总线的使用权（控制权）的问题，总线控制部件是总线的仲裁机构。

连接到总线上的功能模块有主动和被动两种模式，主动模式的模块称为主方（Master），它可以启动一个总线周期，被动模式的模块称为从方（Slave），它只能响应主方的请求。例如，CPU 模块在不同的时间里既可以用作主方，也可以用作从方，而存储器模块只能用作从方。由于总线是在多个部件之间共享的，每一次总线操作只能有一个主方占用总线控制权，但是同一时间里可以有一个或多个从方。从这种意义上说，主方是那些在某个时刻独占总线的部件，一般会在占有总线之前发出总线占用请求。除了 CPU 模块外，I/O 功能模块、DMA 控制器也可以作为主方提出总线请求。

为了解决多个主设备同时竞争总线控制权的问题，必须设置总线仲裁部件，以某种方式选择其中一个主设备作为总线的下一个主方。被授权的主方在当前总线业务结束时，立即接管总线控制权，开始新的信息传送，主方持续控制总线的时间称为总线占用期。

按照总线仲裁电路位置的不同，仲裁方式分为集中式仲裁和分布式仲裁两类。

1. 集中式仲裁

在集中式仲裁中，每个功能模块有两条线连到中央仲裁器：一条是送往仲裁器的总线请求信号线 BR，一条是仲裁器送出的总线授权信号线 BG。对于单处理器系统总线而言，中央仲裁器又称为总线控制器，它是 CPU 的一部分。按照目前的总线标准，中央仲裁器一般是一个单独的功能模块，其结构如图 6-8 所示，图中 A 表示地址线，D 表示数据线。

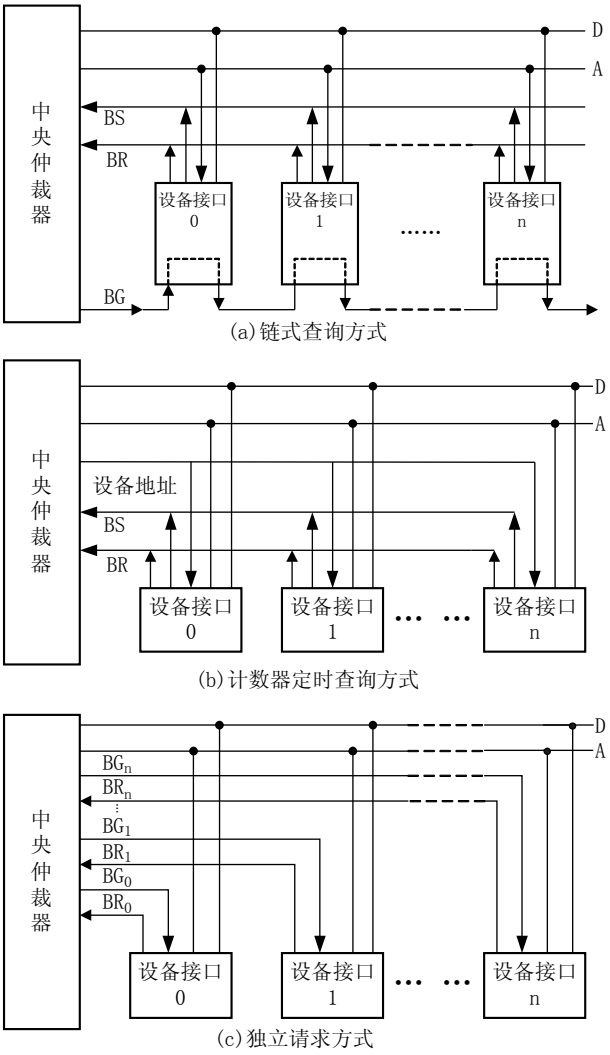


图 6-8 集中式总线仲裁方式

1）链式查询方式

链式查询方式如图 6-8(a)所示，总线中有一条 BS 线，它标明总线的状态：1 表示总线正被某个主方所使用，0 表示总线空闲。

链式查询方式的主要特点：总线授权信号 BG 采用串行方式从一个 I/O 接口传送到下一个 I/O 接口。假如 BG 到达的接口无总线请求，则继续往下查询；假如 BG 到达的接口有总线请求，BG 信号便不再往下查询，这意味着该 I/O 接口获得了总线控制权。显然，在查询链中离中央仲裁器最近的设备具有最高优先级，离中央仲裁器越远的设备其优先级越低。因此，链式查询是通过接口的优先级排队电路来实现的。

链式查询方式的优点是只用很少几根线就能按一定的优先次序实现总线仲裁，并且这种结构很容易扩充新的设备。链式查询方式的缺点是对查询链的电路故障非常敏感，如果第 i 个设备的接口中有关查询链的电路发生故障，那么第 i 个以后的设备就都不能工作了。此外，查询链的优先级是固定的，如果优先级高的设备频繁发出总线请求，则优先级较低的设备有可能长期无法使用总线。

2）计数器定时查询方式

计数器定时查询方式如图 6-8(b)所示。总线上的任一设备需要使用总线时，通过 BR 线发出总线请求，中央仲裁器接到请求后，在 BS 线为“0”的情况下让计数器开始计数，计数值通过一组地址线发向各个设备。每个设备接口都有一个设备地址判别电路，当地址线上的计数值与请求总线的设备的地址一致时，该设备将 BS 线置“1”，获得总线使用权，同时终止计数查询。

每次计数既可以从“0”开始，也可以从终止点开始：如果从“0”开始，各设备的优先次序与链式查询

法相同，优先级的顺序是固定的；如果从终止点开始，则每个设备使用总线的优先级是相等的。计数器的初值也可用程序来设置，这样可以方便地改变优先次序，但这种灵活性是以增加线数为代价的。

3）独立请求方式

独立请求方式如图 6-8(c)所示。在独立请求方式中，每一个共享总线的设备均有一对总线请求线 BR_i 和总线授权线 BG_i。当设备要求使用总线时，便发出该设备的请求信号。中央仲裁器中有一个排队电路，根据自己的优先策略决定首先响应哪个设备的请求，给该设备以授权信号 BG_i。

独立请求方式的优点首先是响应速度快，确定优先响应的设备所花费的时间少，用不着一个设备接一个设备地查询；其次，对优先次序的控制相当灵活，可以预先固定，也可以通过程序来改变，还可以用屏蔽（禁止）某个请求的办法，不响应来自无效设备的请求。因此，现代的总线标准普遍采用独立请求方式。

2. 分布式仲裁

分布式仲裁不需要中央仲裁器，每个潜在的主方功能模块都有自己的仲裁号和仲裁器。当它们有总线请求时，把它们惟一的仲裁号发送到共享的仲裁总线上，每个仲裁器将仲裁总线上得到的号与自己的号进行比较，如果仲裁总线上的号大，则它的总线请求不予响应，并撤消它的仲裁号，最后，获胜者的仲裁号保留在仲裁总线上。显然，分布式仲裁是以优先级仲裁策略为基础的。

6.2.2 总线的通信

总线的通信就是决定共享总线的各个部件之间如何进行通信、如何实现数据传输的问题。

总线的一次信息传送过程，大致可分为如下五个阶段：请求总线，总线仲裁，寻址（目的地址），信息传送，状态返回（或错误报告）。

为了同步主方、从方的操作，必须制订通信定时协议。所谓定时，是指事件出现在总线上的时序关系。对此，计算机系统中有两种截然不同的通信方式：同步通信和异步通信。

1. 同步通信

在同步通信协议中，事件出现在总线上的时刻由总线时钟信号来确定。图 6-9(a)表示读数据的同步时序，所有事件都出现在时钟信号的上升沿，大多数事件只占据一个时钟周期。CPU 首先发出读命令信号，并将存储器地址发到地址线上，它也可发出一个启动信号，指明控制信息和地址信息均已出现在总线上。存储器模块识别地址码，经过一个时钟周期延迟（存取时间）后，将数据和认可信息放到总线上，被 CPU 读取。

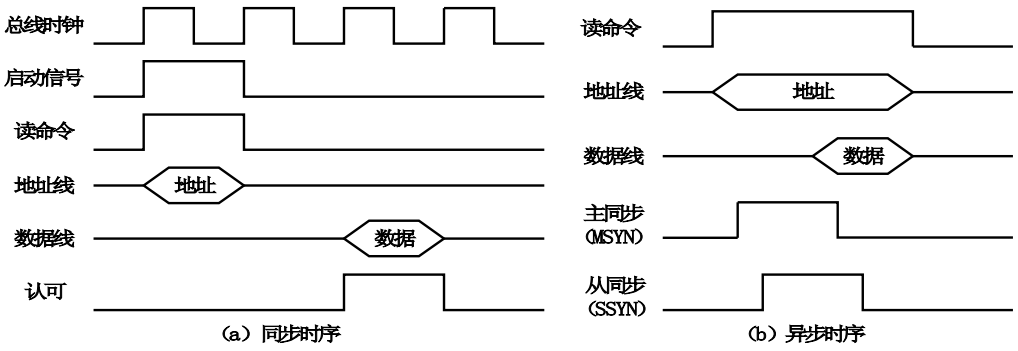


图 6-9 读数据的同步时序和异步时序

由于采用了公共时钟，每个功能模块什么时候发送或接收信息都由统一时钟规定，因此，同步通信具有较高的传输频率。

同步通信适用于总线长度较短、各功能模块存取时间比较接近的情况，这是因为同步方法对于任何两个功能模块的通信都给予相同的时间安排。但由于同步总线必须按最慢的模块来设计公共时钟，当各功能模块的存取时间相差很大时，总线效率会大大损失。

2. 异步通信

在异步通信协议中，后一事件出现在总线上的时刻取决于前一事件的出现，即建立在应答式或互锁机制基础上。在这种系统中，不需要统一的公共时钟信号，总线周期的长度是可变的。图 6-9(b)表示采用异步

通信协议的读数据操作过程。CPU 发出读命令信号和存储器地址信号，经过一段时间的延迟，待信号稳定后，它启动主同步（MSYN）信号，引发存储器以从同步（SSYN）信号予以响应，并将数据放到数据线上。这个 SSYN 信号使 CPU 读数据，然后撤消 MSYN 信号，MSYN 信号撤消又使 SSYN 信号撤消，最后地址线、数据线不再有有效信息，于是读数据总线周期结束。

异步通信的优点是总线周期长度可变，不把响应时间强加到功能模块上，因而允许快速和慢速的功能模块连接到同一总线上，但这是以增加总线的复杂性和成本为代价的。正因为如此，目前多数微机的总线还是采用同步通信的方法。

6.2.3 信息传送方式

数字计算机使用二进制数，它们或用电位的高、低来表示，或用脉冲的有、无来表示。在前一种情况下，电位高时表示数字“1”，电位低时表示数字“0”，这种技术称为电位传送；在后一种情况下，有脉冲时表示数字“1”，无脉冲时表示数字“0”，这种技术称为脉冲传送。

计算机系统中，传送信息采用三种方式：串行传送、并行传送和分时传送，但是出于速度和效率上的考虑，系统总线上传送的信息必须采用并行传送方式。

1. 串行传送◆

当信息以串行方式传送时，只有一条传输线，且采用脉冲传送。在串行传送时，按顺序传送表示一个数码的所有二进制位（bit）的脉冲信号，每次一位，通常以第一个脉冲信号表示数码的最低有效位，最后一个脉冲信号表示数码的最高有效位。串行传送的示意图如图 6-10 所示。

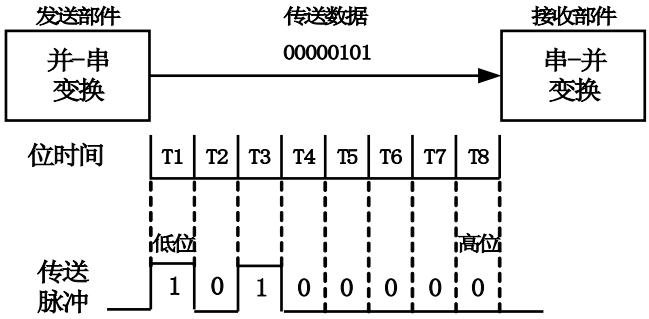


图 6-10 信息的串行传送方式

当串行传送时，有可能按顺序连续传送若干个“0”或若干个“1”。如果在编码时用有脉冲表示二进制数“1”，无脉冲表示二进制数“0”，那么当连续出现几个“0”时，则表示某段时间间隔内传输线上没有脉冲信号。为了确定究竟传送了多少个“0”，必须采用某种时序格式，以便使接收设备能加以识别，通常采用的方法是指定“位时间”，即指定一个二进制位在传输线上占用的时间长度，显然，“位时间”是由同步脉冲来体现的。

假设串行数据是由“位时间”组成的，那么传送 8 个比特需要 8 个位时间。例如，假设接收设备在第一个位时间和第三个位时间接收一个脉冲，其余的 6 个位时间没有接收脉冲，那么我们就可以知道所接收到的二进制信息是 00000101。注意，串行传送时，低位在前，高位在后。

在串行传送时，被传送的数据需要在发送部件进行并－串变换，这称为拆卸；而在接收部件又需要进行串－并变换，这称为装配。

串行传送的主要优点是只需要一条传输线，这一点对长距离传输尤其重要，不管传送多少数据量都只需要一条传输线，成本比较低廉。

2. 并行传送

用并行方式传送二进制信息时，对应于每个数据位都需要一条单独的传输线，信息由多少二进制位组成，就需要多少条传输线，从而使得二进制数“0”或“1”在不同的线上同时进行传送。

并行传送的过程如图 6-11 所示。如果要传送的数据由 8 位二进制位组成（1 个字节），那么可以使用 8 条线组成的扁平数据电缆，每一条线代表二进制数的不同位值。例如，最上面的线代表最高有效位，最下

面的线代表最低有效位，因而图中正在传送的二进制数就是 10101100。

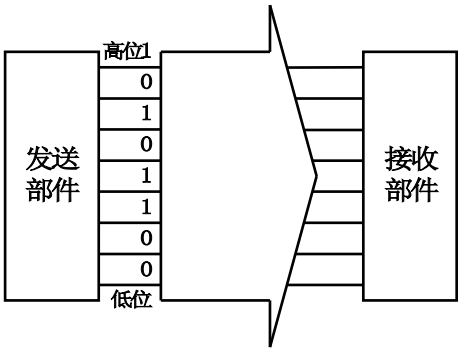


图 6-11 信息的并行传送方式

并行传送一般采用电位传送。由于所有的位同时被传送，所以并行数据传送比串行数据传送快得多。例如，使用 32 根单独的地址线，可以从 CPU 的地址寄存器同时传送 32 位地址信息给主存。

3. 分时传送

分时传送有两种概念。一是采用总线复用方式，某个传输线上既传送地址信息，又传送数据信息，为此必须划分时间片，以便在不同的时间间隔中完成传送地址和传送数据的任务；分时传送的另一种概念是共享总线的部件分时使用总线。

6.3 总线系统实例

随着计算机技术的进步，总线技术的标准也在不断发展。例如，微型计算机总线就经历了 PC/XT 总线（1981 年）、PC/AT 或 ISA 总线（1984 年）、EISA 总线（1988 年）、VESA 总线（1989 年）、PCI 总线（1991 年）、AGP 总线（1997 年）、PCI-X 总线（1998 年）和 PCI Express（2004 年）等发展阶段。本节重点介绍在微型计算机系统中有代表意义的几个总线标准。

6.3.1 ISA总线

ISA（Industry Standard Architecture，工业标准结构）总线是由 IBM PC/XT 和 PC/AT 使用的 8 位总线发展而来的总线标准。ISA 是 8/16 位兼容总线，因此 I/O 插槽有 8 位和 8/16 位两种类型：8 位扩展槽由 62 个引脚组成，其中包括 20 条地址线和 8 条数据线，用于 8 位数据传输；8/16 位扩展槽除了一个 8 位 62 线的连接器外，还有一个附加的 36 线连接器，这种扩展插槽既可以支持 8 位插接板，也可支持 16 位插接板（24 条地址线和 16 条数据线）。ISA 总线的应用范围很广，一般用于连接中、低速 I/O 设备。

6.3.2 PCI总线

PCI（外设部件互连，Peripheral Component Interconnect）总线是 1991 年由 Intel、IBM、Compaq、Apple 等几家公司联合推出的。PCI 是一个与处理器无关的高速外围总线，又是至关重要的层间总线，可支持 10 台外部设备，它采用同步时序协议和集中式仲裁策略，并具有自动配置能力。

整个系统里有如下三种不同的总线：

⑴ HOST 总线：又称宿主总线，该总线不仅连接主存，还可以连接多个 CPU。

⑵ PCI 总线：用于连接各种高速 PCI 设备。PCI 设备可以是主设备，也可以是从设备，或兼而有之。在 PCI 设备中不存在 DMA 的概念，这是因为 PCI 总线支持无限的猝发式传送，这样，传统总线上用 DMA 方式工作的设备移植到 PCI 总线上时，只要采用主设备工作方式即可。系统中允许有多条 PCI 总线，它们可以使用 HOST 桥与 HOST 总线相连，也可以使用 PCI-PCI 桥与 PCI 总线相连，从而扩充整个系统的 PCI 总线负载能力。

⑶ LEGACY 总线：可以是 ISA、EISA 这类性能较低的传统总线，以便充分利用市场上丰富的适配器卡，支持中、低速 I/O 设备。

在 PCI 总线体系结构中有三种桥：HOST 桥、PCI-PCI 桥、PCI-LEGACY 桥，其中，HOST 桥又是 PCI 总线控制器，含有中央仲裁器。桥具有很重要的作用，它一方面连接两条总线，使彼此之间相互通信，另

一方面又是一个总线转换部件，可以把一条总线的地址空间映射到另一条总线的地址空间上，从而使系统中任意一个总线主设备都能看到同样的一份地址表。桥本身的结构可以十分简单（如只有信号缓冲能力和信号电平转换逻辑），也可以相当复杂（如有规程转换、数据快存、装拆数据等）。

6.3.3 AGP总线

AGP（Accelerated Graphics Port，加速图形端口）总线标准是在 PCI 总线难以适应高总线频率和高视频带宽需求的情况下出现的。随着 Pentium II 的出现，PC 机的总线频率达到了 66MHz，甚至 100MHz，另一方面，多媒体的深入应用，3D 纹理及集合材质都需要大量显存和更高的总线带宽，PCI 已不能满足快速数据传输的要求，于是，AGP 应运而生。AGP 是微型计算机系统中专门为 3D 显示而设置的加速图形接口，它以 66MHz（AGP 的基频）PCI 2.1 规范为基础，支持点对点连接，允许 3D 图形数据越过 PCI 总线，解决了 PCI 总线系统设计对于超高速系统的瓶颈问题。

6.3.4PCI Express总线

随着 Pentium 4 前端总线频率的迅速提高（高达 1GHz 以上），原有的 PCI 总线标准已难以适应新的要求，因此，统一总线标准、提高总线带宽成为业界的普遍呼声，这时， PCI Express 就应运而生了。PCI Express 是一种基于串行技术、高带宽连接点、芯片到芯片连接的新型总线技术。有别于 PCI 并行技术，PCI Express 采用 4 根信号线，两根差分信号线用于接收，另外两根差分信号线用于发送；信号频率 2.5GHz，采用 8/10 位编码；定义了用于多种通道的连接方式，如×1、×4、×8、×16 以及×32 通道的连接器，分别对应于 500MB/s、2GB/s、4GB/s、8GB/s 和 16GB/s 的带宽。

PCI Express 卡支持热插拔和热交换，采用的三个电压是+3.3V、+3.3Vaux 和+12V。用于取代 AGP 插槽的接口是×16 的，带宽为 5GB/s，有效带宽 4GB/s。

采用 PCI Express 总线标准的最大意义在于其通用性和兼容性，通过与 PCI 软件模块的完全兼容，可以确保现有设备和驱动程序不用修改仍能正常工作。PCI Express 不仅可以与其他设备连接，延伸到芯片组之间的连接，还可以用于图形芯片的连接，这样就将整个 I/O 系统统一起来，进一步优化微机系统的设计，增加计算机的可移植性和模块化。

第7章 输入输出（I/O）系统

键盘、鼠标、打印机都是常见的计算机外围设备，这些设备就是通常意义上所说的输入输出设备。从功能上可以将输入输出设备分为两类，一类是完成输入输出操作的设备，另一类是作为外部存储器的设备，外部存储器的访问需要通过输入输出接口进行，因此也可以看作是一种输入输出设备。

各种外围设备通过输入输出接口与计算机主机相连，完成主机分配的任务并进行信息交换，这就是输入输出系统的功能。

输入输出接口需要连接各种不同类型、不同工作速度和数据传输速度的外围设备，因此产生了各种不同的输入输出控制方式。

本章首先介绍5种输入输出控制方式，然后重点介绍程序中断、DMA、通道这三种常用方式，最后介绍几个通用输入输出接口的实例。

7.1 输入输出控制方式

一般而言，CPU管理外围设备的输入输出控制方式有5种：程序查询方式、程序中断方式、DMA方式、通道方式、外围处理机方式，前两种方式由软件实现，后三种方式由硬件实现。

1. 程序查询方式

程序查询方式是早期计算机中使用的一种方式，CPU与外围设备的数据交换完全依赖于计算机的程序控制。

在进行信息交换之前，CPU要设置传输参数、传输长度等，然后启动外设工作，与此同时，外设则进行数据传输的准备工作；相对于CPU来说，外设的速度是比较低的，因此外设准备数据的时间往往是一个漫长的过程，而在这段时间里，CPU除了循环检测外设是否已准备好之外，不能处理其他业务，只能一直等待；直到外设完成数据准备工作，CPU才能开始进行信息交换。

这种方式的优点是CPU的操作和外围设备的操作能够完全同步，硬件结构也比较简单。但是，外围设备的动作通常很慢，程序进行循环查询白白浪费了宝贵的CPU时间，数据传输效率低下。在当前的实际应用中，除了单片机之外，已经很少使用程序查询方式了。

2. 程序中断方式

中断是外围设备用来“主动”通知CPU，准备发送或接收数据的一种方式。

通常，当一个中断发生时，CPU暂停其现行程序，转而执行中断处理程序，完成数据I/O工作；当中断处理完毕后，CPU又返回到原来的任务，并从暂停处继续执行程序。

这种方式节省了CPU时间，是管理I/O操作的一个比较有效的方法。中断方式一般适用于随机出现的服务，并且一旦提出要求，应立即执行。与程序查询方式相比，程序中断方式的硬件结构相对复杂一些，服务成本较大。

3. DMA方式

DMA方式就是直接存储器存取（Direct Memory Access）方式，是一种完全由硬件执行I/O交换的工作方式。

在该方式中，DMA控制器从CPU完全接管对总线的控制权，数据交换不经过CPU而直接在主存和外围设备之间进行，以便高速传送数据。

这种方式的主要优点是数据传送速度很高，传送速率仅受限于主存的访问时间。与程序中断方式相比，这种方式需要更多的硬件，适用于主存和高速外围设备之间大批量数据交换的场合。

4. 通道方式

DMA方式的出现减轻了CPU对I/O操作的控制，使得CPU的效率显著提高，而通道的出现则进一步提高了CPU的效率。

通道是一个具有特殊功能的处理器，又称为输入输出处理器（IOP），它分担了CPU的一部分功能，可以实现对外围设备的统一管理，完成外围设备与主存之间的数据传送。

通道方式大大提高了 CPU 的工作效率，然而这种效率的提高是以增加更多的硬件为代价的。

5. 外围处理机方式

外围处理机（Peripheral Processor Unit，PPU）方式是通道方式的进一步发展。

PPU 基本上独立于主机工作，它的结构更接近于一般的处理机，甚至就是微小型计算机。在一些系统中，设置了多台 PPU，分别承担 I/O 控制、通信、维护诊断等任务，从某种意义上说，这种系统已经变成了分布式多机系统。

综上所述，计算机外围设备的输入/输出方式如图 7-1 表示。其中，程序查询方式和程序中断方式适用于数据传输率比较低的外围设备，而 DMA 方式、通道方式和外围处理机方式则适用于数据传输率比较高的外围设备。

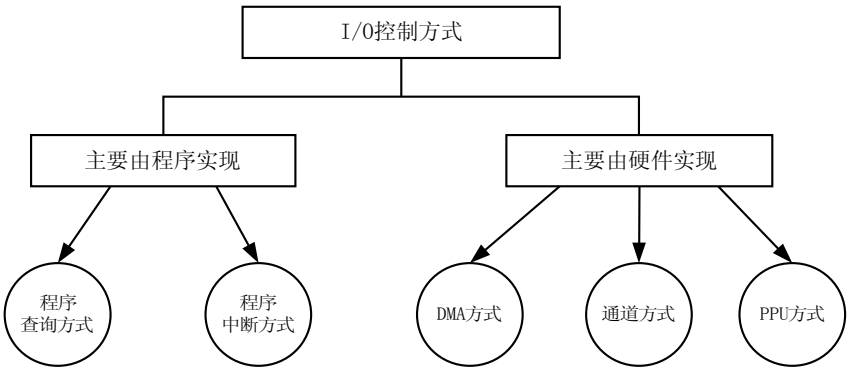


图 7-1 外围设备的输入/输出方式

7.2 程序中断方式

7.2.1 中断的基本概念

程序查询方式要求 CPU 不断地用指令检测方法来获取外设工作状态，造成 CPU 的运行效率极低。20 世纪 50 年代中后期中断概念的出现，是计算机系统结构设计中的一项重大变革。

在程序中断方式中，某一外设的数据准备就绪后，它“主动”向 CPU 发出中断请求信号，请求 CPU 暂时中断目前正在执行的程序转而进行数据交换；当 CPU 响应这个中断时，便暂停运行主程序，自动转去执行该设备的中断服务程序；当中断服务程序执行完毕（数据交换结束）后，CPU 又回到原来的主程序继续执行。

中断处理示意图如图 7-2 所示，由图可见，CPU 只是在外围设备 A、B、C 的数据准备就绪后，才去执行对应的中断服务程序，进行数据交换；而当低速的外围设备准备自己的数据时，CPU 则照常执行自己的主程序。从这个意义上说，CPU 和外设的一些操作是异步并行进行的，因而与串行进行的程序查询方式相比，计算机系统的效率的确是大大提高了。

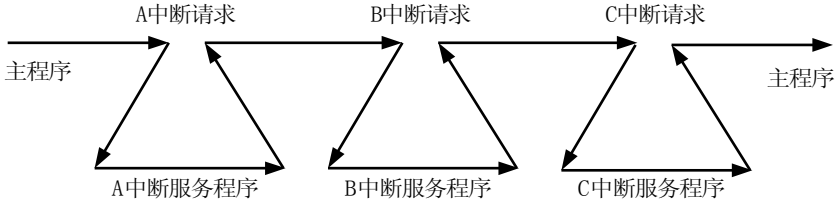


图 7-2 中断处理示意图

CPU 只有在当前一条指令执行完毕后，即转入公操作时，才会受理外围设备的中断请求。

为了在中断服务程序执行完毕以后，能够正确地返回到原来主程序被中断的地方（断点）继续执行，必须把程序计数器 PC 的内容，以及当前指令执行结束后 CPU 的状态（包括寄存器的内容和一些状态标志位）都保存到堆栈中去，这些操作称为保存现场；在中断服务程序执行完毕后，需要执行恢复现场操作，从堆栈中恢复 PC 内容和 CPU 状态，以便从断点处继续执行主程序。

中断处理过程是由硬件和软件结合来完成的，中断周期由硬件实现，而中断服务程序则由机器指令序列实现。

计算机的中断过程类似于子程序调用，但在本质上又有所区别：子程序的调用是事先安排好的，而中断则是随机产生的；子程序的执行往往与主程序有关，而中断服务程序则可能与主程序毫无关系，比如发生电源掉电等异常情况。

7.2.2 单级中断与多级中断

根据计算机系统对中断处理策略的不同，中断系统可以分为单级中断系统和多级中断系统。单级中断系统是中断结构中最基本的形式。

在单级中断系统中（图 7-3），所有的中断源都属于同一级，所有中断源触发器排成一行，其优先次序是离 CPU 越近优先级越高。当响应某一中断请求时，CPU 执行该中断源的中断服务程序，在此过程中，中断服务程序不允许被其他中断源所打断，即使优先级比它高的中断源也不例外，只有当该中断服务程序执行完毕之后，才能响应其他中断。

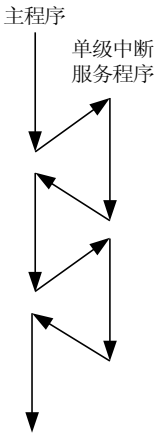


图 7-3 单级中断示意图

多级中断系统是指计算机系统中的多个中断源，根据中断事件的轻重缓急程度不同而分成若干个级别，每一个中断级分配一个优先级。一般而言，优先级高的中断级可以打断优先级低的中断服务程序，以程序嵌套方式进行工作。中断嵌套是指当一个中断服务程序正在执行时，一个优先级比它更高的中断源发出中断请求，CPU 暂停当前中断服务程序的执行，转而执行优先级更高的中断服务程序，如图 7-4 所示。图中，CPU 嵌套响应了系统中的两个中断服务程序。多级中断的出现，扩大了系统中断功能，进一步加强了系统处理紧急事件的能力。

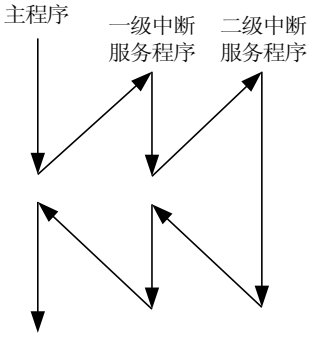


图 7-4 多级中断示意图

为了能够及时处理最为紧迫的中断，必须判断多级中断中哪个中断的优先级更高，通常可采用以下两种处理方法：

1. 软件查询法

所谓软件查询法，就是采用程序查询技术来确定发出中断请求的中断源及其中断优先级。最先查询的中断具有最高优先级，最后查询的中断则为最低优先级。因此，查询的先后顺序决定了中断优先级的高低。如果中断请求正好来源于最后查询的那个中断，那么就浪费了此前的大量查询时间，因此，软件查询的效率很低。

2.硬件处理法

为了提高处理效率，通常采用硬件处理方法，即采用优先级排队电路或专用中断控制器等硬件电路来管理中断。

7.2.3 中断控制器

中断控制器是一块专用的集成电路芯片，它将中断接口与优先级判断等功能集于一身。

在80x86的早期系统中，采用一片8259A芯片作为中断控制器；到了80386系统中，则采用可编程中断控制器PIC（Programmable Interrupt Controller），也就是两块8259A芯片的级联；在Pentium以及后来的CPU中，集成了高级可编程中断控制器APIC（Advanced Programmable Interrupt Controller），可用于多处理器。

7.3 DMA方式

7.3.1 DMA基本概念

DMA方式是一种完全由硬件执行I/O交换的工作方式。在这种方式中，DMA控制器从CPU完全接管对总线的控制，数据交换不经过CPU，而直接在主存和I/O设备之间进行。DMA控制器向主存发出地址和控制信号，修改主存地址，对传送的字的个数进行计数，并且以中断方式向CPU报告传送操作的结束。DMA方式控制简单，适用于高数据传输率设备进行成组传送。

DMA方式的主要优点是速度快，由于CPU不参加传送操作，因此省去了CPU取指令、取数、送数等操作，也没有保存现场、恢复现场之类的工作。而且，主存地址的修改、传送字个数的计数等也不由软件实现，而是用硬件线路直接实现的。所以，DMA方式能够满足高速I/O设备的要求，也有利于CPU效率的发挥，一般用于高速传送成组数据。

DMA方式的工作过程如下：首先，当要求通过DMA方式传输数据时，DMA控制器向CPU发出请求，CPU释放总线控制权，交由DMA控制器管理；然后，DMA控制器向外设返回一个应答信号，外设与主存开始进行数据交换；最后，当数据传输完毕后，DMA控制器把总线控制权交还给CPU。在这种方式下，DMA控制器与CPU分时使用总线，其时间图如图7-7所示。

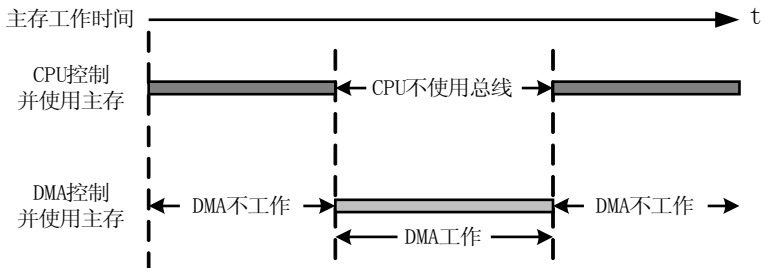


图7-7 DMA传送方式时间图

在DMA方式中，批量数据传送前的准备工作，以及传送结束后的处理工作，仍由CPU通过执行管理程序来承担，DMA控制器只负责具体的数据传送工作。

7.3.2 DMA数据传送过程

一次DMA数据块传送过程可分为三个阶段：传送前预处理、正式传送、传送后处理，如图7-8所示。

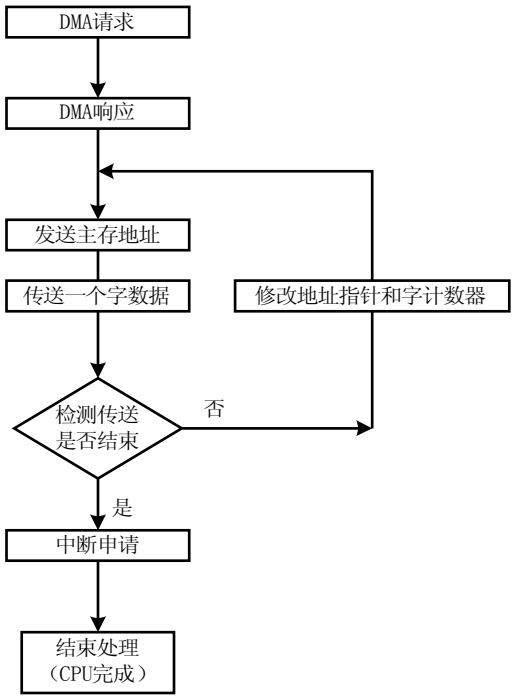


图 7-8 DMA 传输数据的过程

1）预处理阶段

CPU 执行几条输入输出指令，测试设备状态，向 DMA 控制器的设备地址寄存器中送入设备号并启动设备，向主存地址计数器中送入起始地址，向字计数器中送入交换数据字个数。在这些工作完成后，CPU 继续执行原来的主程序。◆

当外设准备好发送数据（输入）或接收数据（输出）时，它发出 DMA 请求，由 DMA 控制器向 CPU 发出总线使用权请求 HOLD。

2）正式传送阶段

当外围设备发出 DMA 请求时，CPU 在本机器周期执行结束后响应该请求，并使 CPU 的总线驱动器处于第三态（高阻状态）。之后，CPU 与系统总线相脱离，而 DMA 控制器则接管数据总线与地址总线的控制，并向主存提供地址，于是在主存与外围设备之间进行数据交换。每交换一个字，地址计数器和字计数器加“1”，当字计数器溢出时，DMA 操作结束，DMA 控制器向 CPU 发出中断报告。

DMA 数据传送是以数据块为基本单位进行的，因此，每次 DMA 控制器占用总线后，无论是数据输入操作，还是输出操作，都是通过循环来实现的。当进行输入操作时，外围设备的数据（一次一个字或一个字节）传向主存；当进行输出操作时，主存的数据传向外围设备。

3）后处理阶段

一旦 DMA 的中断请求得到响应，CPU 停止主程序的执行，转去执行中断服务程序，完成 DMA 结束处理工作，这些工作包括校验送入主存的数据是否正确，决定继续 DMA 传送还是结束，测试传送过程中是否发生错误等等。◆

基本 DMA 控制器与系统的连接方式有两种，一种是公用的 DMA 请求方式，另一种是独立的 DMA 请求方式。

7.3.3 选择型和多路型DMA控制器

最简单的 DMA 控制器，一个控制器只控制一个 I/O 设备，而在实际应用中情况要复杂得多，因此通常采用选择型 DMA 控制器和多路型 DMA 控制器。

1. 选择型 DMA 控制器

选择型 DMA 控制器在物理上可以连接多个设备，而在逻辑上只允许连接一个设备。换句话说，在某一个时间段内只能为一个设备提供服务。

选择型DMA控制器的工作原理与基本DMA控制器大致相同。除了前面提到的基本逻辑部件外，还有一个设备号寄存器。数据传送是以数据块为单位进行的，在每个数据块传送之前的预置阶段，除了用程序中的I/O指令给出数据块的传送个数、起始地址、操作命令外，还要给出所选择的设备号。从预置开始，一直到这个数据块传送结束，DMA控制器只为所选的设备提供服务。下一次预置时再根据I/O指令指出的设备号，为所选择的另一设备提供服务。显然，选择型DMA控制器相当于一个逻辑开关，根据I/O指令来控制此开关与某个设备连接。

选择型DMA控制器只增加了少量的硬件就达到为多个外围设备提供服务的目的，它特别适合于数据传输率很高甚至接近于主存存取速度的设备，在高速传送完一个数据块后，控制器又可为其他设备提供服务。

2. 多路型DMA控制器◆

与选择型DMA方式相比，多路型DMA不仅在物理上可以连接多个外围设备，而且在逻辑上也允许这些外围设备同时工作，各个设备以字节交叉方式通过DMA控制器进行数据传送。

多路型DMA控制器适合于同时为多个慢速外围设备提供服务。

多路型DMA控制器可以对多个独立的DMA通路进行控制。当某个外围设备请求DMA服务时，操作过程如下：◆

⑴DMA控制器接到设备发出的DMA请求，将请求转送到CPU。◆

⑵CPU在适当的时刻响应DMA请求。若CPU不需要占用总线则继续执行指令；若CPU需要占用总线则进入等待状态。◆

⑶DMA控制器接到CPU的响应信号后，进行以下工作：①对现有DMA请求中优先权最高的请求予以响应；②选择相应的地址寄存器的内容来驱动地址总线；③根据所选设备操作寄存器的内容，向总线发出读、写信号；④外围设备向数据总线传送数据，或从数据总线接收数据；⑤每个字节传送完毕后，DMA控制器使相应的地址寄存器和长度寄存器加“1”或减“1”。

以上是一个DMA请求的过程，在一批数据传送过程中，要多次重复上述过程，直到外围设备表示一个数据块已传送完毕，或该设备的长度控制器判定传送长度已满。

7.4 通道方式

7.4.1 通道的功能

DMA方式解决了快速外设和主机成批交换信息的难题，简化了CPU对数据传送的控制，提高了主机与外设并行工作的程度，提高了系统的效率。但是，在DMA方式下，CPU仍然摆脱不了管理和控制外设的沉重负担，难以充分发挥高速运算的能力。随后出现的通道方式，将控制I/O操作和信息传送的功能从CPU中独立出来，代替CPU管理和调度外设与主机的信息交换，从而进一步提高了CPU的效率。

通道是一个特殊功能的处理器，是计算机系统中代替CPU管理控制外设的独立部件。它有自己的指令和程序，专门负责数据输入输出的传输控制，而CPU在将“传输控制”功能下放给通道后只负责“数据处理”功能。这样，通道与CPU分时使用主存，实现了CPU内部运算与I/O设备的并行工作。

通道的基本功能是执行通道指令，组织外围设备和主存进行数据传输，按I/O指令要求启动外围设备，向CPU报告中断等。

CPU通过执行I/O指令以及处理来自通道的中断，实现对通道的管理。来自通道的中断有两种，一种是数据传送结束中断，另一种是故障中断。

通道使用通道指令控制设备控制器进行数据传送操作，并以通道状态字接收设备控制器反映的外围设备的状态。因此，设备控制器是通道对I/O设备实现传输控制的执行机构。

7.4.2 通道的工作过程

系统在进行一次通道操作之前，CPU要完成准备通道程序、安排数据缓冲区、给通道和外设发起命令等工作。在通道接到启动命令后，便到指定点取通道地址，指定点是系统设计好的，由通道硬件实现。通道根据指定点提供的主存地址，从主存中取出CPU为它准备的通道程序。

在执行第一条通道程序之前，通道首先要选择外设，启动外设的设备号，看其是否有响应，总线上的外设都有自己的地址译码器，用于判断总线上的呼叫地址是否是本设备地址；选择设备后，通道向外设接口发出命令，外设接口接到命令后返回状态码，通道便以条件码形式回答CPU，表示这次启动成功；于是CPU便可以转去执行其他程序，而通道程序则由通道独立完成；当通道与外设之间的信息交换完成后，通道向CPU发出中断信号，CPU根据通道状态字分析这次通道操作的执行情况。

7.4.3 通道的类型

根据通道的工作方式，通道分为字节多路通道、选择通道、数组多路通道三种类型。一个系统可以兼有多种类型的通道，也可以只有其中一、二种。

1. 字节多路通道

字节多路通道是一种简单的共享通道，主要用于连接控制多台低速外设，以字节交叉方式传送数据。例如，某个外设的数据传输率只有1000B/s，即传送1个字节的时间间隔是1ms，而通道从设备接收或发送一个字节只需要几百ns，因此，通道在传送两个字节之间有很多空闲时间，字节多路通道正是利用这个空闲时间为其他设备提供服务。每个设备分时占用一个很短的时间片，不同的设备在各自分得的时间片内与通道建立连接，实现数据的传输。

2. 选择通道

选择通道又称高速通道，在物理上它可以连接多个设备，但是这些设备不能同时工作，在某一个时间段内通道只能选择一个设备进行工作。选择通道很像一个单道程序的处理器，在一段时间内只允许执行一个设备的通道程序，只有当这个设备的通道程序全部执行完毕后，才能执行其他设备的通道程序。◆

选择通道主要用于连接高速外围设备，如磁盘、磁带等，信息以成组方式高速传输。由于数据传输率很高，如达到1.5MB/s，通道在传送两个字节之间只有很少的空闲时间，所以，在数据传送期间只为一台设备服务是合理的。但是，这类设备的寻址等辅助操作的时间往往很长，在这样长的时间里通道一直处于等待状态，因此，整个通道的利用率还不是很高。◆

3. 数组多路通道◆

连接控制多个高速外设并以成组交叉方式传送数据的通道称为数组多路通道。数组多路通道是对选择通道的一种改进，当某个设备进行数据传送时，通道只为该设备提供服务；当设备在执行寻址等控制性动作时，通道暂时断开与该设备的连接，挂起该设备的通道程序，而转去为其他设备提供服务，即执行其他设备的通道程序。所以，数组多路通道很像一个多道程序的处理器。

对于磁盘一类的高速外设，采用数组多路通道，可在其中一个外设占用通道进行数据传送时，让其他外设进行寻址等辅助操作，使一个设备的数据传送操作与其他设备的寻址操作彼此重叠，实现成组交叉方式的数据传送，从而使通道具备多路并行工作的能力，充分发挥通道高速信息交换的效能。

由于数组多路通道既保留了选择通道高速传送数据的优点，又充分利用控制性操作的时间间隔为其他设备提供服务，使通道的效率得到充分的发挥，因此，数组多路通道在实际系统中得到较多的应用。

字节多路通道和数组多路通道都是多路通道，在一段时间内均能交替执行多个设备的通道程序，使这些设备同时工作。不同之处在于：数组多路通道允许多个设备同时工作，但只允许一个设备进行传输型操作，其他设备进行控制型操作；而字节多路通道不仅允许多个设备同时操作，而且也允许它们同时进行传输型操作。另外，数组多路通道与设备之间进行数据传送的基本单位是数据块，而字节多路通道与设备之间进行数据传送的基本单位则是字节。◆

7.5 通用I/O接口

7.5.1 RS-232接口

RS-232接口是一种常用的串行通信接口，它是在1970 年由美国电子工业协会（EIA）制定的串行通信标准，其全名是“数据终端设备（DTE）和数据通信设备（DCE）之间串行二进制数据交换接口技术标准”。

RS-232 接口最初是为了在 DTE 与 DCE 之间进行远程连接而制定的，后来则用于在计算机与其外围设备或终端之间建立近距离的连接。由于这个接口在诸如信号功能、电器特性和机械特性上都进行了明确细致的规定，加上通信接口与设备制造厂商生产的通信设备均与 RS-232 兼容，因此，RS-232 接口在计算机系统中成为一种用来实现与打印机、CRT 终端、键盘、调制解调器等外围设备进行异步串行数据通信的标准硬件接口。

1. RS-232 的系统结构

RS-232 标准规定采用一个 25 针的 DB-25 连接器，并对连接器的每个引脚的信号内容以及各种信号的电平进行了规定，后来 IBM PC 机将 RS-232 简化成了 DB-9 连接器。某些设备与 PC 机连接的 RS-232 接口，由于不使用传送控制信号，因此只需要三根信号线就可以实现双向通信，即“发送数据 TXD”、“接收数据 RXD”和“信号地 GND”。RS-232 传输线采用屏蔽双绞线。

2. RS-232 的技术指标

1）传送速率

RS-232 接口支持的最大波特率为 20000bps。一般而言，接收和发送的波特率不一定要完全相同，然而在大部分比较简单的系统中，往往把它们设置为相同的值。

2）传送距离

RS-232 标准规定，在码元畸变小于 4%的情况下，传输电缆长度应为 50 英尺，其实这个 4%的码元畸变是很保守的，在实际应用中，约有 99%的用户是按码元畸变 10-20%的范围工作的，所以，实际使用中最大距离会远超过 50 英尺。

经过许多年来 RS-232 接口器件以及通信技术的改进，RS-232 接口的通信距离已经大大增加。波士电子的 RS-232 增强器可以将普通的 RS-232 接口的通信距离延长到 1000 米。然而，不同的波特率其最大传送距离的差别也很大，美国 DEC 公司规定允许码元畸变为 10%而得出的实验结果显示，当波特率为 110bps 时，其最大传送距离为 1500 米，当波特率为 9600bps 时，其最大传送距离缩短为 75 米。

7.5.2 IDE接口

IDE 接口是一种用于在 PC 机中连接硬盘驱动器的接口，其英文全称为“Integrated Drive Electronics”，即“电子集成驱动器”，其本意是指把“硬盘控制器”与“盘体”集成在一起的硬盘驱动器，代表着硬盘的一种类型。IDE 接口技术一直在不断发展，其性能也在不断的提高，拥有价格低廉、兼容性强的特点。近几年来，随着硬盘接口技术的发展，IDE 接口已经慢慢趋于淘汰，而其后发展分支出的硬盘接口，如 ATA、Ultra ATA、DMA、Ultra DMA 等，也都属于 IDE 硬盘。

1. IDE 的系统结构

IDE 接口硬盘的控制电路集成在硬盘上，与 IDE 驱动器通信所需的软件程序则存储在 PC 机主板的 BIOS 芯片中。IDE 接口用一个 40 针电缆连接硬盘与主板（电源由另外的电缆提供），其中包括 3 个寻址的信号线引脚，16 个传输数据的双向数据线引脚，以及其他用于控制信号、驱动信号和状态表示等的引脚。

2. IDE 模式的发展

随着技术的发展，计算机产品对数据传输速度的要求日益提高，IDE 硬盘接口的数据传输模式也经历了三次技术变化，由最初的 PIO 模式，到 DMA 模式，直到 Ultra DMA 模式。

1）PIO 模式

PIO（Programming Input/Output Model）模式是一种通过 CPU 执行 I/O 端口指令来进行数据读写的交换模式。作为最早的硬盘数据传输模式，其数据传输速率低下，CPU 占用率很高，大量传输数据时会因为占用过多的 CPU 资源而导致系统停顿，无法进行其他的操作。PIO 数据传输模式又分为 PIO mode 0、PIO mode 1、PIO mode 2、PIO mode 3、PIO mode 4 等几种模式，数据传输速率从 3.3MB/s 到 16.6MB/s 不等。受限于传输速率低下和极高的 CPU 占用率，这种数据传输模式很快就被淘汰了。

2）DMA 模式

DMA 是一种不经过 CPU 而直接从主存存取数据的数据交换模式。PIO 模式下硬盘和主存之间的数据传输是由 CPU 来控制的，而在 DMA 模式下，CPU 只须向 DMA 控制器下达指令，让 DMA 控制器来处理

数据的传送，数据传送完毕再把信息反馈给 CPU，这样就在很大程度上降低了 CPU 资源占用率。DMA 模式与 PIO 模式的区别在于，DMA 模式不过分依赖 CPU，可以大大节省系统资源，二者在传输速度上的差异并不十分明显。DMA 模式可以分为 Single-Word DMA（单字 DMA）和 Multi-Word DMA（多字 DMA）两种，但其所能达到的最大传输速率只有 16.6MB/s。

3）Ultra DMA 模式

Ultra DMA（Ultra DMA，一般简写为 UDMA）的含义是高级直接主存访问。UDMA 模式以 16-bit Multi-Word DMA（16 位多字 DMA）模式为基准，可以理解为 DMA 模式的增强版本，它在包含 DMA 模式优点的基础上，又增加了 CRC（Cyclic Redundancy Check，循环冗余码校验）技术，以提高数据传输过程的准确性、安全性。在以往的硬盘数据传输模式下，一个时钟周期只传输一次数据，而在 UDMA 模式中逐渐应用了 Double Data Rate（双倍数据传输）技术，因此数据传输速度有了很大的提高，此技术就是在时钟的上升期和下降期各进行一次数据传输，可以使数据传输速度成倍增长。

在 UDMA 模式发展到 UDMA 133 之后，受限于 IDE 接口的技术规范，无论是连接器、连接电缆、信号协议都显现出了很大的技术瓶颈，而且所支持的最高数据传输率也很有限。同时，随着 IDE 接口传输率的提高（也就是工作频率的提高），IDE 接口的交叉干扰、地线增多、信号混乱等缺陷也给其发展带来了很大的制约，最终被新一代的 SATA 接口所取代。

7.5.3 SATA接口

SATA（Serial ATA，串行 ATA）是一种连接存储设备（大多为硬盘）的串行接口，用于取代传统的并行 ATA 接口。2001 年，由 Intel、APT、Dell、IBM、希捷、迈拓这几大厂商组成的 SATA 委员会正式确立了 SATA 1.0 规范，2002 年又确立了 SATA 2.0 规范。SATA 接口使用嵌入式时钟信号，具备了更强的纠错能力，与以往相比，其最大的改进在于能对传输指令（不仅仅是数据）进行检查，并且在发现错误后可以自动矫正，这在很大程度上提高了数据传输的可靠性。SATA 接口还具有结构简单、支持热插拔等优点。

1. SATA 的物理结构

SATA 的物理设计，是以 Fibre Channel（光纤通道）作为蓝本的，所以采用了四芯电缆。同时，所需的电压也从传统的 ATA 接口的 5V 大幅减低至 250mV（最高 500mV），由此可以给 SATA 硬盘附加上热插拔（Hot Swapping）等高级功能。更重要的是，在连接形式上，除了传统的点对点（Point-to-Point）形式之外，SATA 还支持星型连接，这样就为 RAID 等高级应用提供了设计上的便利。在实际使用中，SATA 的主机总线适配器（Host Bus Adapter，HBA）就好像网络交换机一样，可以以通道形式与每个硬盘单独通信，即每个 SATA 硬盘都可独占一个传输通道，所以不存在像 ATA 那样的主/从控制问题。

SATA 以连续串行的方式传送数据，一次只传送 1 位数据，这样能够减少 SATA 接口的针脚数目，使连接电缆数目变少，效率也会更高。实际上，SATA 仅用四个针脚就能完成所有的工作，他们分别用于连接电缆、连接地线、发送数据和接收数据，同时，这样的架构还能降低系统的复杂性和能耗。

2. SATA 的特点

SATA 规范不仅立足于未来，而且还保留了多种向后兼容方式，在使用上不存在兼容性的问题。在硬件方面，SATA 标准允许使用转换器提供与 ATA 设备的兼容性，转换器能把来自主板的 ATA 信号转换成 SATA 硬盘使用的串行信号，这在某种程度上保护了计算机用户的原有投资，降低了升级成本；在软件方面，SATA 和 ATA 保持了软件兼容性，这意味着厂商不必为了使用 SATA 而重写任何驱动程序和系统代码。

另外，SATA 接线较传统的 ATA 接线简单得多，而且容易收放，能够明显改善机箱内的气流及散热。而且，与始终被困在机箱之内的 ATA 不同，SATA 硬盘的扩充性很强，可以置于机箱之外，外置式机柜不但可以提供更好的散热及插拔功能，也可以用多重连接来防止单点故障；由于 SATA 与光纤通道的设计如出一辙，所以其传输速度可以用独立通道的形式得到保证，这在服务器和网络存储上具有重要的意义。

与 ATA 相比，SATA 的起点更高，发展潜力更大，SATA 1.0 定义的数据传输率可达 150MB/s，这比 ATA 所能达到的 133MB/s 的最高数据传输率还要高，而 SATA 2.0 的数据传输率则达到 300MB/s，最终 SATA 将实现 600MB/s 的最高数据传输率。

7.5.4 USB接口

USB（Universal Serial Bus，通用串行总线）接口是近几年推出的一种全新的外部设备接口，它是由Compaq、DEC、IBM、Intel、Microsoft、NEC等公司为简化PC与外设之间的互连而共同研究开发的一种标准化接口，支持各种PC与外设之间的连接，还可实现数字多媒体集成。

USB接口的主要特点是即插即用，允许热插拔。USB连接器将各种各样的外设I/O端口合而为一，使之可以热插拔，并具有自动配置能力，用户只要简单地将外设插入到USB连接器上，PC机就能自动识别和配置USB设备。除此之外，USB接口的其他优点，如带宽更大、增加外设时无需添加接口卡、多个USB集线器可互传数据等，也使得PC机可以用全新的方式控制外设。

随着时间的推移，USB已成为PC机的标准配置，基于USB的外设也逐渐增多，包括调制解调器、键盘、鼠标、光驱、游戏手柄、软驱、扫描仪等。

近年来，USB总线标准由1.1版升级到了2.0版，传输率由12Mbps增加到了480Mbps，连接距离也由原来的5米增加到近百米。

USB设计者尽其所能地在USB接口上体现出未来计算机对接口的需求标准：易用性、稳定性、兼容性、扩展性、完备性、网络性和低耗性。

1. USB的系统结构

USB总线结构简单，信号定义仅由2条电源线和2条信号线组成。

从硬件结构来说，USB系统采用级联星型拓扑，该拓扑由三个基本部分组成：主机（Host）、集线器（Hub）和功能设备。

⑴主机，也称为根、根结点或根Hub，做在计算机主板上或作为适配卡安装在计算机上。主机包含有主控制器和根集线器（Root Hub），控制USB总线上数据和控制信息的流动，每个USB系统只能有一个根集线器，它连接在主控制器上。

⑵集线器是USB结构中的特定部分，它提供端口（Port），以便将设备连接到USB接口上，同时检测连接在总线上的设备，并为这些设备提供电源管理，负责总线的故障检测和恢复。

⑶功能设备通过端口与总线连接。

从软件结构来说，每个USB只有一个主机，它包括以下几层：

⑴USB总线接口：USB总线接口处理电气层与协议层的互连。

⑵USB系统：USB系统用主控制器管理主机与USB设备间的数据传输，它与主控制器间的接口依赖于主控制器的硬件定义。同时，USB系统也负责管理USB资源，例如带宽和总线能量，这使得客户访问USB成为可能。

⑶USB客户软件：位于软件结构的最高层，负责处理特定USB设备驱动器。客户程序层描述所有直接作用于设备的软件入口。当设备被系统检测到后，这些客户程序将直接作用于外围硬件。

2. 数据传输模式

主控制器负责主机与USB设备间数据流的传输，这些传输数据被当作连续的比特流。每个设备提供了一个或多个可以与客户程序通信的接口，每个接口由0个或多个管道组成，它们分别独立地在客户程序与设备的特定终端间传输数据。通用串行总线驱动程序（USBD）为主机软件的现实需求建立了接口和管道，当提出配置请求时，主控制器根据主机软件提供的参数提供服务。

USB支持四种基本的数据传输模式：控制传输、等时传输、中断传输、数据块传输。每种传输模式应用到同名终端，则具有不同的性质。

⑴控制传输类型：支持外设与主机之间的控制、状态、配置等信息的传输，为外设与主机之间提供一个控制通道。每种外设都支持控制传输类型，这样主机与外设之间就可以传送配置和命令/状态信息。

⑵等时（Isochronous）传输类型：支持有周期性、有限的时延和带宽、且数据传输速率不变的外设与主机间的数据传输。该类型无差错校验，故不能保证正确的数据传输，支持诸如计算机－电话集成系统（CTI）、音频系统与主机的数据传输。

⑶中断传输类型：支持诸如游戏手柄、鼠标、键盘等输入设备，这些设备与主机间的数据传输量小，无周期性，但对响应时间敏感，要求马上响应。

⑷数据块（Bulk）传输类型：支持打印机、扫描仪、数码相机等外设，这些外设与主机间的数据传输量大，USB在满足带宽的情况下才进行该类型的数据传输。

USB为计算机外设输入输出提供了新的接口标准，它使设备具有热插拔、即插即用、自动配置的能力，使设备连接标准化。USB的级联星型拓扑结构大大扩充了外设数量，使增加、使用外设更加便捷、快速，而USB2.0标准更是将数据传输速率提高到了一个新的高度，使之拥有美好的应用前景。

7.5.5 SCSI接口

SCSI接口是小型计算机系统接口（Small Computer System Interface）的简称，是一种并行I/O标准接口，其设计思想来源于IBM大型机系统的I/O通道结构，目的是使CPU摆脱对各种设备的繁杂控制。它是一个高速智能接口，可以混接各种磁盘、光盘、磁带机、打印机、扫描仪、条码阅读器以及通信设备。它由SCSI控制器（又称为主机适配器）进行数据操作，SCSI控制器相当于一块小型的CPU，有自己的命令集和缓存。SCSI接口具有应用范围广、多任务、带宽大、CPU占用率低、热插拔等优点，主要应用于中、高端服务器和高档工作站中。

SCSI系统可以只有一个主机、一个主机适配器和一个外设控制器，也可以有多个主机以及多个主机适配器和外设控制器，主机适配器和外设控制器统称为SCSI设备。SCSI规定，系统至多可以有的SCSI设备的数目，是SCSI数据总线的位数，如采用32位数据总线，则至多可以有32个SCSI设备，其中有一个是主机适配器。

每个外设控制器可以带一个或多个设备。每台SCSI设备都有自己的识别ID，每台外设也都有自己的识别号。主机和主机适配器之间是系统总线，主机适配器和外设控制器之间是SCSI总线，外设控制器和外设之间是设备总线。这样的系统构成使总线并不直接与外设接口，不需要按照具体外设的物理性能给予特别的处理，而是用一组通用的高级命令通过外设控制器去控制各种设备，从而使得SCSI总线具有很好的通用性。

SCSI设备以菊花链连接成一个系统，使用50针A电缆和可选的68针B电缆，单端方式和差分方式在一个系统中不能同时存在。

SCSI是一种不断前进的技术，为了提高数据传输率、改善接口的兼容性，20世纪90年代又陆续推出了SCSI-2和SCSI-3标准。除此之外，串行SCSI也由并行SCSI接口演化而来，数据传输率最高可达到600MB/s。

7.5.6 IEEE-1394接口

IEEE-1394是IEEE制定的一项具有视频数据传输速度的串行接口标准，支持外接设备热插拔，同时可为外设提供电源，省去了外设自带的电源，支持同步数据传输。IEEE-1394接口的速度很快，而且相对于SCSI来讲又要小巧许多，所以逐渐被人们所接受，并日益普及。

1. IEEE-1394的性能特点

与SCSI等并行接口相比，IEEE-1394串行接口具有如下三个显著特点：

⑴数据传送的高速性：IEEE-1394标准定义了三种传输速率：100Mbps、200Mbps和400Mbps，这个速度完全可以用来传输未经压缩的动态画面信号。

⑵数据传送的实时性：保证多媒体数据的实时传送，避免出现图像和声音时断时续的现象。

⑶体积小易安装，连接方便：使用6针电缆，插座体积小，具有“热插拔”能力，即使在全速工作时也可以插入或拆除设备。

2．IEEE-1394的结构配置

IEEE-1394采用菊花链式配置，也允许采用树型结构配置，不过仍是以线性连接菊花链组成树型结构的各种线性分支。IEEE-1394接口也需要一个主适配器和系统总线相连，通常将主适配器及其端口称为主端口。主端口是IEEE-1394接口树型配置结构的根节点，一个主端口最多可连接63台设备，这些设备称为节点，两个相邻节点之间的线缆最长为4.5米，两个节点之间进行通信时中间最多可经越15个节点的转接

再驱动，因此通信的最大距离是 72 米，线缆不需要终端器。IEEE-1394 标准接口结构的所有资源都是统一存储编址，并用存储变换方式识别，实现资源配置和管理。

总之，IEEE-1394 是一种高速串行 I/O 标准接口，各被连接装置的关系是平等的，不用 PC 介入也能自成系统，IEEE-1394 已经在多媒体领域被广泛接受。

第8章 并行计算机系统

计算机系统性能和容量的快速增长，除了归功于底层 VLSI 技术的发展之外，另一个重要因素在于计算机体系结构的不断改进，而并行性则是其中的一个主要方面。本章介绍并行性、向量处理机、阵列处理机、多处理机系统、机群系统、网格计算等并行计算机系统的基本概念。

8.1 并行性的概念

所谓并行性，是指计算机系统具有可以同时进行运算或操作的特性，它包括同时性与并发性两种含义。

同时性（Simultaneity）是指两个或两个以上的事件在同一时刻发生。

并发性（Concurrency）是指两个或两个以上的事件在同一时间间隔内发生。

所谓并行计算（Parallel Computing）是指通过网络相互连接的两个以上的处理机相互协调工作，同时计算同一个任务的不同部分，从而提高问题求解速度，或者求解单机无法解决的大规模问题。

并行计算的目的有两个：

⑴提高速度

对于一个固定规模的问题，采用并行计算技术可以使求解时间更少。现在的微型机也开始借助于流水线技术、多核技术等并行计算技术来提高系统的速度。

⑵扩大问题求解规模

由于器件本身的限制，任何单处理器的速度不能超过某个上限，要突破这个上限，必须采用并行计算技术。例如，通过采用并行计算技术对密钥空间进行穷举搜索，实现了对 DES（Data Encryption Standard，数据加密标准）的破解，而单个计算机是无法在有效时间内完成这一工作的。

8.1.1 并行性分类

计算机系统中的并行性有不同的等级。

从处理数据的角度看，并行性等级从低到高可分为：

⑴字串位串：同时只对一个字的一位进行处理。这是最基本的串处理方式，不存在并行性。

⑵字串位并：同时对一个字的全部位进行处理，不同字之间是串行的。这里已开始出现并行性。

⑶字并位串：同时对许多字的一位进行处理。这种方式有较高的并行性。

⑷全并行：同时对许多字的全部位进行处理。这是最高一级的并行性。

从执行程序的角度看，并行性等级从低到高可分为：

⑴指令内部并行：一条指令执行时各微操作之间并行。

⑵指令级并行：并行执行两条或多条指令。

⑶任务级或过程级并行：并行执行两个以上过程或任务（程序段）。

⑷作业或程序级并行：并行执行两个以上作业或程序。

在单处理机系统中，这种并行性升到某一级别后（如任务级或过程级并行），需要通过软件（如操作系统中的进程管理、作业管理）来实现；而在多处理机系统中，已具备了完成各项任务或作业的处理机，其并行性是由硬件实现的。

在一个计算机系统中，可以采取多种并行性措施，既可以有数据处理方面的并行性，也可以有执行程序方面的并行性。当并行性提高到一定级别时，称为进入并行处理领域。例如，处理数据的并行性达到字并位串级，或者执行程序的并行性达到任务或过程级，就可以认为进入并行处理领域。

并行处理着重挖掘计算过程中的并行事件，使并行性达到较高的级别。因此，并行处理是体系结构、硬件、软件、算法、语言等多方面综合研究的领域。

8.1.2 提高并行性的技术途径

计算机系统中提高并行性的措施多种多样，就其基本思想而言，可归纳成如下四条途径：

1．时间重叠

即时间并行。在并行性概念中引入时间因素，多个处理过程在时间上相互错开，轮流重叠地使用同一

套硬件设备的各个部分，以加快硬件周转时间而赢得速度。

2．资源重复

即空间并行。在并行性概念中引入空间因素，采用以数量取胜的原则，通过重复设置硬件资源，大幅度提高计算机系统的性能。随着硬件价格的降低，这种方式在单处理机中广泛应用，而多处理机本身就是实施资源重复原理的结果。

3．资源重复+时间重叠

在计算机系统中同时运用空间并行和时间并行技术，这种方式在计算机系统中应用广泛，成为主流的并行技术。

4．资源共享

这是一种软件方法，它使多个任务按一定时间顺序轮流使用同一套硬件设备。例如多道程序、分时系统就是遵循资源共享原理而产生的。资源共享既降低了成本，又提高了计算机设备的利用率。

8.1.3 并行性的发展

1．单机系统并行性发展

在发展高性能单处理机过程中，起着主导作用的是时间并行技术。实现时间并行的物质基础是部件功能专用化，即把一件工作按功能分割为若干个相互联系的部分，把每一部分指定给专门的部件完成；然后按时间重叠原理把各部分执行过程在时间上重叠起来，使所有部件依次分工完成一组同样的工作。

指令流水线是由重叠（Overlap）发展而来的。通常，CPU 执行一条指令的过程分为 5 步：从主存中取指令（IF）、指令译码（ID）、执行指令（EX）、访存取数（M）、写回结果（WB）。CPU 在执行其中每一步的时候，其他部分的功能处于闲置状态。如果能够将这些步骤重叠起来，比如在译码分析一条指令的同时将下一条指令取出，就可以提高 CPU 的效率。流水线是对重叠的引申，它将 CPU 执行一条复杂指令的处理过程分成多个复杂程度相当、处理时间大致相等的子过程，每个子过程由一个独立的功能部件来完成，将这些独立的功能部件按流水线方式连接起来，处理对象在这条流水线上连续流动，满足时间重叠原理。在同一时间里，多个部件同时进行不同的操作，完成对不同子过程的处理，从而使得处理机内部同时处理多条指令，提高了处理机的速度。显然，时间并行技术实现了计算机系统中的指令级并行。

在单处理机中，空间并行技术的运用也已经十分普遍。不论是非流水处理机，还是流水处理机，多体交叉存储器和多操作部件都是空间并行技术成功应用的结构形式。在多操作部件处理机中，通用部件被分解成若干个专用的操作部件，如加法部件、乘法部件、除法部件、逻辑运算部件等，一条指令所需的操作部件只要空闲，就可以开始执行这条指令，这就是指令级并行。

在单处理机中，资源共享概念的实质就是用单处理机模拟多处理机的功能，形成所谓的“虚拟机（Virtual Machine，VM）”的概念。例如在分时系统中，在多终端情况下，每个终端上的用户都感觉自己好像独立拥有一台处理机一样。

2．多机系统并行性发展

多机系统也遵循时间重叠、资源重复、资源共享原理，向三种不同的多处理机方向发展，但在采取的技术措施上与单处理机系统有些差别。

为了反映多机系统各机器之间物理连接的紧密程度及交互能力的强弱，我们引入“耦合度”这个术语。按照耦合度的不同，可将多机系统分为紧耦合系统和松耦合系统两大类。

紧耦合系统又称直接耦合系统，指计算机间物理连接的频带较高，一般通过总线或高速开关实现计算机间的互连，可以共享主存。由于具有较高的信息传输率，因而可以快速并行处理作业或任务。

松耦合系统又称间接耦合系统，一般通过通道或通信线路实现计算机间的互连，可以共享外存设备（磁盘、磁带等），机器之间的相互作用是在文件或数据集一级上进行的。松耦合系统表现为两种形式：一种是多台计算机和共享外存设备连接，不同机器之间实现功能上的分工（功能专用化），机器处理的结果以文件或数据集的形式送到共享外存设备，供其他机器继续处理；另一种是计算机网络，通过通信线路连接，以求得更大范围的资源共享。

多处理机中为了实现时间重叠，将处理机功能分散给各台专用处理机去完成，即功能专用化，各处理

机之间则按时间重叠原理工作。例如，输入/输出功能的分离，导致由通道向专用外围处理机发展，许多主要功能（如数组运算、高级语言编译、数据库管理等）也逐渐分离出来，交由专用处理机完成，机器间的耦合程度逐渐加强，从而发展成为异构型多处理机系统。

通过设置多台相同类型的计算机而构成的容错系统，可使系统工作的可靠性在处理机一级得到提高。各种不同的容错多处理机系统方案对计算机间互连网络的要求是不同的，但正确性、可靠性是其首要要求。如果提高对互连网络的要求，使其具有一定的灵活性、可靠性和可重构性，则可将其发展成一种可重构系统。在这种系统中，平时几台计算机都像通常的多处理机系统一样正常工作，一旦发生故障，就使系统重新组织，降低档次继续运行，直到排除故障为止。

随着硬件价格的降低，人们追求的目标是通过多处理机的并行处理来提高整个系统的速度，因此对计算机间互连网络的性能提出了更高的要求。高带宽、低延迟、低开销的机间互连网络，是高效实现任务级并行处理的前提条件。为了使并行处理的任务能够在处理机间随机地进行调度，就必须使各处理机具有同等的功能，从而成为同构型多处理机系统。

8.2 并行计算机系统

8.2.1 向量处理机

向量处理机是指令级并行的计算机，能较好地发挥流水线技术的特性，达到较高的计算速度，而新型的向量流水处理机则采用了多处理机的体系结构。

向量处理

从数学的概念上讲，标量（Scalar）是指单个量，而向量（Vector）是指一组标量。例如，有一个数组$A=(a_1, a_2, a_3,\ldots, a_n)$，其中括号内的每一个元素$a_i$就是一个标量，而 A 则称为向量，它由一组标量组成。

一条向量指令可以处理 N 个或 N 对操作数，我们把这 N 个互相独立的数称为向量，对这样一组数的运算称为向量处理。因此，向量指令的处理效率要比标量指令的处理效率高得多。

8.2.2 阵列处理机

阵列处理机又称并行处理机，主要技术手段是采用硬件资源重复的方法来实现并行性，属于 SIMD 结构计算机。单指令流多数据流 SIMD 计算机用一个控制部件同时管理多个处理单元，所有处理单元均收到从控制部件广播来的同一条指令，但是操作的对象却是不同的数据。

向量流水处理机和阵列处理机都能对大量数据进行向量处理，但它们之间存在很大的区别，阵列处理机有着向量处理机所不具备的特点：

⑴阵列机是以单指令流多数据流方式工作的。

⑵阵列机采用资源重复方法引入空间因素，即在系统中设置多个相同的处理单元来实现并行性，这与利用时间重叠的向量流水处理机是不一样的。此外，阵列机利用并行性中的同时性，所有处理单元必须同时进行相同的操作。

⑶阵列机是以某一类算法（如图像处理）为背景的专用计算机。这是由于阵列机中通常都采用简单、规整的互连网络来实现处理单元间的连接操作，从而限定了其所适用的求解算法类别。因此，对互连网络设计的研究成为阵列机研究的重点之一。

⑷阵列机的研究必须与并行算法的研究密切结合，以使其求解算法的适应性更强一些，应用面更广一些。

⑸从处理单元来看，由于结构都相同，因而可将阵列机看成是一个同构型并行机。但其控制器实质上是一个标量处理机，而为了完成 I/O 操作及操作系统管理，尚需一个前端机，因此，实际的阵列机系统是由上述三部分构成的一个异构型多处理机系统。

8.2.3 多处理机系统

随着集成电路技术的不断发展，基于微处理器的多处理机并行系统由于其突出的性价比而逐渐成为高性能计算机的主流。

1. 多处理机系统的特点

流水线机器通过若干级流水线的时间并行技术来获得高性能，阵列处理机器由多台处理机组成，每台处理机执行相同的程序。这两类机器都是执行单个程序，可对向量或数组进行运算。这种体系结构能高效地执行适合于 SIMD 的程序，所以这类机器对某些应用问题非常有效。但是，有些大型问题在这种 SIMD 结构机器上运行并不那么有效，原因是这类问题没有对结构化数据进行重复运算的操作，其所要求的操作通常是非结构化的而且是不可预测的。要想解决这类问题并保持高性能，只能在多处理机结构中寻找出路。

多处理机的体系结构由若干台独立的计算机组成，每台计算机能够独立执行自己的程序。在多处理机系统中，处理机与处理机之间通过互连网络进行连接，从而实现程序之间的数据交换和同步。

多处理机属于 MIMD 计算机，它与属于 SIMD 计算机的阵列处理机相比有很大的差别，其本质差别在于并行性等级不同：多处理机要实现任务或作业一级的并行，而阵列处理机只实现指令一级的并行。

2. 多处理机系统的分类

多处理机系统由多个独立的处理机组成，每个处理机都能够独立执行自己的程序。多处理机系统有多种分类方法。

按多处理机各机器之间物理连接的紧密程度与交互作用能力的强弱来分，多处理机可分为紧耦合系统和松耦合系统两大类。在紧耦合多处理机系统中，处理机间物理连接的频带较高，一般是通过总线或高速开关实现互连，可以共享主存储器，由于具有较高的信息传输率，因而可以快速并行处理作业或任务。松耦合多处理机系统由多台独立的计算机组成，一般通过通道或通信线路实现处理机间的互连，可以共享外存设备，机器之间的相互作用是以较低频带在文件或数据集一级上进行的。

按处理机的结构是否相同来分，如果每个处理机是同类型的，且完成同样的功能，称为同构型多处理机系统。如果多处理机是由多个不同类型，且担负不同功能的处理机组成，则称为异构型多处理机系统。

8.2.4机群系统

机群系统（Cluster）的定义是：一组完整的计算机互连，它们作为一个统一的计算机资源一起工作，并能产生一台机器的印象。完整的计算机是指一台计算机离开机群系统仍能运行自己的任务。在文献中，机群系统中的每台计算机一般称为结点。

根据上述定义，机群是并行或分布计算机系统的一种类型，它是由一组完整的计算机（结点）通过高性能的网络或局域网互连而成的系统，作为一个单独的统一计算资源来使用。

首先，机群由完整的计算机（结点）互连而成。机群的结点可以是一个工作站，或者是一台个人计算机，但也可以是一台规模相当大的对称多处理机 SMP。在机群的每个结点上除了有一台或多台处理机外，还有足够的存储器、磁盘、I/O 设备和一套完整的标准操作系统，因此机群的结点还可以按常规的交互方式来单独使用。结点间的互连可以通过普通的商品化网络（如以太网、FDDI、ATM 等），使用标准的通信协议，也可以采用专门设计的网络。

其次，机群应能作为一个单独的统一计算资源来使用。对于机群，最主要的是要具有单一系统形象。所谓单一系统形象是指从用户角度来看，整个机群就像一个系统，用户感觉到使用的是一个单一的系统，可以从任何地点的结点上来使用这个机群，而不必关心提供服务的设备在什么地方。

机群不同于局域网。局域网是一个分布式系统，在局域网中各台计算机基本上都是各自独立工作的，它们只是通过局域网共享资源，局域网没有单一系统形象。而机群中的各台计算机既可以单独使用，又是多台计算机连成的一个整体中的一部分，因此，机群可以充分利用机器资源，充分利用通用计算机产品，达到高并行性和高可靠性的要求。

8.2.5 网格计算

所谓网格计算（Grid Computing），是指由一群松散耦合的普通计算机组织起来的一个虚拟化的超级计算机，通过互联网共享强大的计算能力和数据存储能力，常用来协同地解决大型的科学与工程计算问题。理解网格的一个很好的类比就是“电力网”，用电者不需要知道自己用的电是哪个发电厂送出的。像使用电力资源那样使用计算资源成为网格研究者们的一个梦想，当然这还有很远的路要走。

网格计算不仅在理论和技术上发展迅速，在应用方面也获得了广泛的成功。加州大学伯克利分校的SETI@home就是获得广泛成功的网格项目之一。SETI@home是一个通过网格计算对来自其他宇宙文明社会的电波信号进行灵敏搜索的项目，它能对47种不同CPU和操作系统分发客户端软件，客户从SETI@home网站下载并安装客户端软件之后，即加入了该项目，开始了相应的计算。

8.2.6云计算

云计算（Cloud Computing）是一种基于互联网的超级计算方式，通过这种方式，共享的软硬件资源和信息可以按需提供给计算机和其他设备。云计算描述了一种基于互联网的IT设施或服务的增加、交付和使用模式，通常涉及通过互联网来提供动态、易扩展且通常是虚拟化的资源。云计算服务通常提供通用的在线商业应用，可通过浏览器等软件或者其他Web服务来访问，软件和数据可存储在数据中心。

“云”其实是网络、互联网的一种比喻说法。过去在图中往往用云来表示电信网，后来也用来表示互联网和底层基础设施的抽象。狭义云计算是指IT基础设施的增加、交付和使用模式，指通过网络以按需、易扩展的方式获得所需资源；广义云计算是指服务的增加、交付和使用模式，指通过网络以按需、易扩展的方式获得所需服务，这种服务可以是IT和软件、互联网相关，也可是其他服务，它意味着计算能力也可作为一种商品通过互联网进行流通，就像煤气、水电一样，取用方便，费用低廉。

云计算包括以下几个层次的服务：基础设施即服务（IaaS）、平台即服务（PaaS）和软件即服务（SaaS）。

基础设施即服务 (IaaS)

用户通过Internet从完善的计算机基础设施获得服务。例如: Amazon AWS、Rackspace。

平台即服务 (PaaS)

用户通过Internet获得软件开发平台服务。例如: Google App Engine。

软件即服务 (SaaS)

用户通过Internet向提供商租用基于Web的软件服务，比较常见的是获得一组帐号密码。例如: Microsoft CRM与Salesforce.com。

2006年8月9日，Google首席执行官埃里克·施密特（Eric Schmidt）在搜索引擎大会（SES San Jose 2006）上首次提出“云计算”的概念。自此之后，云计算技术的发展日新月异，涌现了大量典型的云计算应用，Google、IBM、Microsoft、Amazon等IT业巨头纷纷推出了自己的云计算服务。