第5章 计算机系统

【学习内容】

本章介绍计算机系统软硬件相关内容，主要知识点如下。

(1) 计算机系统的基本概念及其组成。

(2) 冯·诺依曼体系结构及各部分工作机制。

(3) 操作系统的基本概念及其主要功能。

(4) 计算机软件系统的分类、层次结构及主要功能。

(5) 计算思维在计算机系统中的体现。

(6) 微处理器核设计初探。

(7) 操作系统运行机理模拟初探。

(8) 利用操作系统接口编程查看系统状态。

【学习目标】

通过本章的学习，读者应掌握如下内容。

(1) 了解计算机系统的组成，理解系统各部分的作用。

(2) 理解冯·诺依曼体系结构。

(3) 掌握中央处理器的工作过程。

(4) 理解存储系统的设计原理、构成和工作原理。

(5) 理解输入输出系统的构成和控制方式，掌握基本术语。

(6) 理解总线结构、工作原理及评价指标。

(7) 掌握操作系统的角色和基本功能。

(8) 理解进程管理、文件管理、设备管理、用户接口等基本概念。

(9) 掌握操作系统进程管理的基本功能和策略。

(10) 理解操作系统存储管理的概念、功能和常用方式。

(11) 理解文件的组织方式，了解文件管理的功能和基本策略。

(12) 理解操作系统设备管理的方式。

(13) 了解操作系统提供的不同用途的用户接口的要素和形式。

(14) 了解计算机软件系统的分类、层次结构及主要功能。

(15) 了解对复杂系统如冯·诺依曼体系结构的抽象与模拟的方法。

(16) 了解微处理器核设计描述的方法。

(17) 了解操作系统进程调度策略的模拟。

(18) 了解通过 Python 编程使用主流操作系统典型功能的方法。

本章主要介绍信息处理的核心装置——计算机系统，包括其软硬件构成与结构、如何支持信息处理，以及各部分在信息处理中的作用。首先介绍计算机硬件系统的体系结构，以冯·诺依曼体系结构为依据，介绍计算机系统的硬件构成。然后围绕着该体系结构各部件，介绍它们如何进行信息表示、信息传递和信息处理，偏重于各部件的核心构成及基本工作原理。

操作系统是计算机系统中最重要的软件，它对计算机系统的软硬件资源进行管理、协调，并代表计算机与外界进行通信。正是有了操作系统，才使得计算机硬件系统真正可用，本章介绍操作系统如何完成上述功能。介绍操作系统的基本概念，根据操作系统的系统管理角色和功能，依次介绍进程管理、存储管理、文件管理、设备管理和用户接口，以及操作系统的加载等所涉及的基本概念和策略。

最后，对硬件系统用模拟的方法进行了研究，展示了如何利用 Python 编程使用主流操作系统主要功能。

5.1 概　　述

一般来说，计算机是一种可编程的机器，它接收输入，存储并且处理数据，然后按某种有意义的格式进行输出。可编程指的是能给计算机下一系列的命令，并且这些命令能被保存在计算机中，并在某个时刻能被取出执行。

通常所说的计算机实际上指的是计算机系统，它包括硬件系统和软件系统两大部分。硬件系统指的是物理设备，包括用于存储并处理数据的主机系统，以及各种与主机相连的、用于输入和输出数据的外围设备(简称外设)，如键盘、鼠标、显示器和磁带机等，根据其用途又分为输入设备和输出设备。计算机的硬件系统，是整个计算机系统运行的物理平台。计算机系统要能发挥作用，仅有硬件系统是不够的，还需要具备完成各项操作的程序，以及支持这些程序运行的平台等条件，这就是软件系统。所以，一个实际的计算机系统的构成如图 5-1 所示。

目前占主流地位的计算机硬件系统结构是冯·诺依曼体系结构，由美国科学家冯·诺依曼在 1946 年提出。在此之前出现的各种计算辅助工具，如差分机等，其用途是固定的，即各种操作是在制造机器的时候就固定下来，不能用于其他用途。以常见的计算器为例，人们只能用它进行各类定制好的运算，而无法用它进行文字处理，更不能打游戏。要使这类机器增加新的功能，只能更改其结构，甚至重新设计机器。所以，这类计算装置是不可编程的。

冯·诺依曼体系结构的核心思想——存储程序改变了这一切。通过创造一组指令集，并将各类运算转换为一组指令序列，使得不需改变机器结构，就能使其具备各种功能。在冯·诺依曼体系结构中，程序和数据都是以二进制形式存放在计算机存储器中的，程序在控制单元的控制下顺序执行。程序是计算机指令的一个序列，指令是计算机执行的最

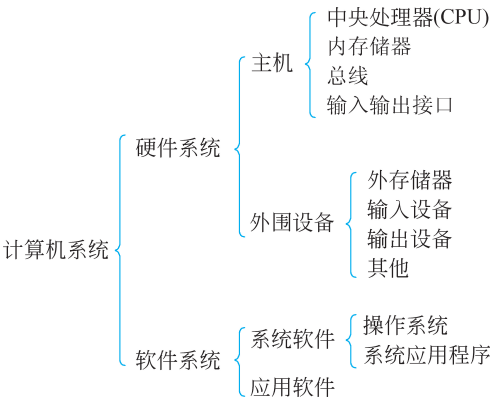


图 5-1　计算机系统的构成

小单位，由操作码和操作数两部分构成。操作码表示指令要执行的动作，操作数表示指令操作的对象是什么，即数据。

在该体系结构中，计算机由 5 部分组成：存储器、运算器、控制器、输入设备和输出设备（见图 5-2）。需要执行的程序及其要处理的数据保存于存储器中，控制器根据程序指令发出各种命令，控制运算器对数据进行操作、控制输入设备读入数据以及控制输出设备输出数据。

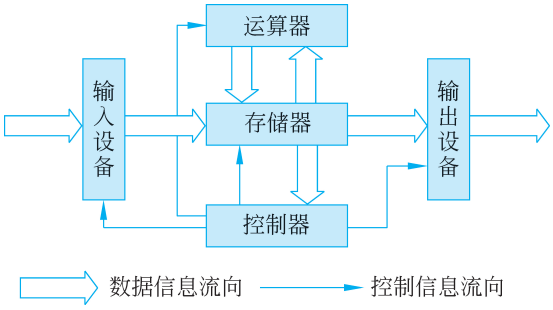


图 5-2　冯·诺依曼体系结构

在冯·诺依曼体系结构形成之前，人们将数据存储于主存中，而程序被看成控制器的一部分，两者是区别对待和处理的。而将程序与数据以同样的形式存储于主存中的特点，对于计算机的自动化和通用性，起到了至关重要的作用。

冯·诺依曼体系结构指的是单机体系结构。为了提高计算机的性能，科学家们提出了各种体系结构。例如，由多个计算机构成的并行处理结构、集群结构等。它们的出现，是为了满足特定任务的要求，这些任务要求计算机系统有更高的能力，以满足诸如气象预报、核武器数值模拟、航天器设计等任务的需求。目前，主流的并行计算结构有对称多处理（Symmetric Multi Processing，SMP）、大规模并行处理（Massively Parallel Processing，MPP）和集群等。

除了看得见摸得着的硬件之外，计算机系统中还包含各种计算机软件系统，简称为软件。计算机科学对软件的定义是，“软件是在计算机系统支持下，能够完成特定功能和性

能的程序、数据和相关的文档”。于是，软件可形式化地表示为

软件＝知识＋程序＋数据＋文档

程序是用计算机程序设计语言描述的。无论是低级语言（如汇编语言），还是高级语言（如 C++ 、Java、Python），程序都可以在相应语言编译器的支持下转换成操纵计算机硬件执行的代码。数据是程序加工的对象和结果。计算机直接加工的数据结构只有简单的整型数、浮点数、逻辑量、字符，人们可以根据需要，在此基础上定义复杂的数据结构。基于大量数据处理的软件需要数据库系统的支持，涉及数据的加工、存储、检索、传输、应用等。文档不但记录软件开发的活动和中间制品、软件的配置及变更，而且用于软件专业人员和用户的交流，以及用于软件开发、过程管理和运行阶段的维护。

软件系统是用户与硬件之间的接口，着重解决如何管理和使用计算机的问题。用户主要是通过软件系统与计算机进行交流。软件是计算机系统设计的重要依据。为了方便用户，以及使计算机系统具有较高的总体效用，在设计计算机系统时，必须通盘考虑软件与硬件的结合，以及用户的要求和软件的要求。没有任何软件支持的计算机称为裸机，其本身不能完成任何功能，只有配备一定的软件才能发挥功效。

软件是抽象的逻辑产品，而不是物理产品。由于不受材料的限制，也不受物理定律或加工过程的制约，具有很大的灵活性。软件的灵活性具有双重性，程序员通过编程可以让计算机巧妙地工作，同时也很容易让软件变得极为复杂，难以理解。软件开发过程的监督、控制、管理有着特殊的困难。因此，软件在开发、生产、维护和使用等方面与硬件相比存在明显的差异。

图 5-3　计算机软件系统的结构

软件的分类原则、方法很多：从软件的功能上分为系统软件和应用软件，从实时性上分为实时软件和非实时软件，从软件运行环境上分为单机软件和网络软件，从加工的数据类型上分为事务处理软件、科学和工程计算软件等，从计算方法上分为基于传统算法的软件、基于符号演算和推理规则的人工智能软件，等等。

计算机软件系统的结构如图 5-3 所示，这是典型的分层结构，下层系统向上层系统提供服务，上层系统利用下层系统提供的服务及特定的程序，可以完成指定的任务。使用计算机并不会直接操作计算机硬件，而是通过在操作系统和各种应用软件上的操作来控制计算机完成各种任务。

5.2　计算机硬件系统

目前占主流地位的计算机硬件系统结构是冯·诺依曼体系结构，如图 5-4 所示。该图中，冯·诺依曼体系结构中的控制器和运算器被集中于 CPU 中，分别对应控制器和算术逻辑部件，主存对应存储器，各种输入输出设备分别对应体系结构中的输入设备和输出设备，各种总线（图中以空心箭头表示）对应于冯·诺依曼体系结构图中的互连线，用于传

输命令和数据。

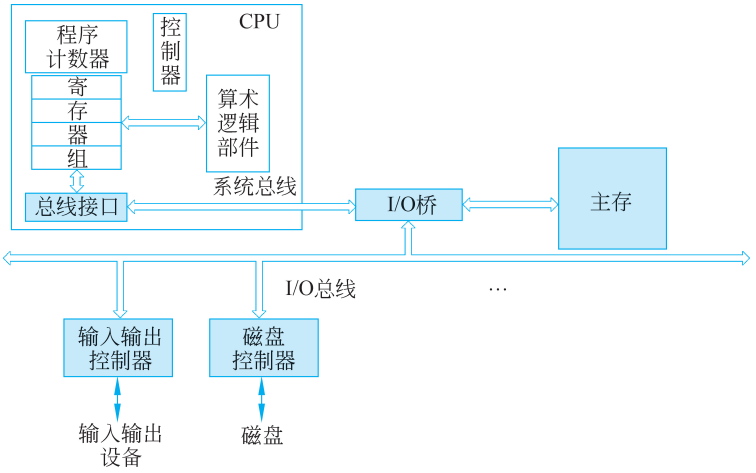


图 5-4　典型的计算机硬件系统结构

5.2.1　中央处理器

一般把中央处理器简称为处理器，是执行存储在主存中的指令的引擎。CPU 一般由算术逻辑部件（Arithmetic and Logic Unit，ALU）、控制单元（Control Unit，CU）和寄存器组构成，由 CPU 内部总线将这些构成连接为有机整体，如图 5-5 所示。

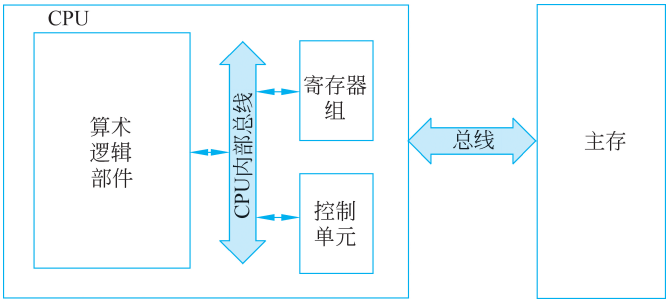


图 5-5　CPU 的内部结构

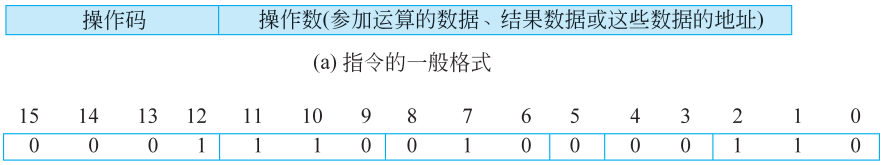
控制单元的主要功能包括指令的分析、指令及操作数的传送、产生控制和协调整个 CPU 工作所需的时序逻辑等。一般由指令寄存器（Instruction Register，IR）、指令译码器（Instruction Decoder，ID）和操作控制器（Operation Controller，OC）等部件组成。CPU 工作时，根据程序计数器保存的主存地址，操作控制器从主存取出要执行的指令，存放在指令寄存器 IR 中，经过译码，提取出指令的操作码、操作数等信息，操作码将被译码成一系列控制码，用于控制 CPU 进行 ALU 运算、传输数据等操作，通过操作控制器，按确定的时序，向相应的部件发出微操作控制信号，协调 CPU 其他部件的动作。操作数将被送到 ALU 进行相对应的操作，得出的结果在控制单元的控制下保存到相应的寄存器中。

ALU 的主要功能是实现数据的算术运算和逻辑运算。ALU 接收参与运算的操作数，并接收控制单元输出的控制码，在控制码的指导下，执行相应的运算。ALU 的输出是运算的结果，一般会暂存在寄存器组中。此外，还会根据运算结果输出一些条件码到状态寄存器，用于标示一些特殊情况，如进位、溢出、除零等。

寄存器组由一组寄存器构成，分为通用和专用寄存器组，用于临时保存数据，如操作数、结果、指令、地址和机器状态等。通用寄存器组保存的数据可以是参加运算的操作数或运算的结果。专用寄存器组保存的数据用于表征计算机当前的工作状态，如程序计数器保存下一条要执行的指令，状态寄存器保存标示 CPU 当前状态的信息，如是否有进位、是否溢出等。通常，要对寄存器组中的寄存器进行编址，以标示访问哪个寄存器，编址一般从 0 开始，寄存器组中寄存器的数量是有限的。

数据和指令在 CPU 中的传送通道称为 CPU 内部总线，总线实际上是一组导线，是各种公共信号线的集合，用于作为 CPU 中各组成部分传输信息共同使用的“公路”。一般分为数据总线(Data Bus，DB)、地址总线(Address Bus，AB)、控制总线(Control Bus，CB)。其中，数据总线用来传输数据信息；地址总线用来传送 CPU 发出的地址信息；控制总线用来传送控制信号、时序信号和状态信息等。

指令是 CPU 执行的最小单位，由操作码和操作数两部分构成，如图 5-6(a)所示。操作码表示指令的功能，即执行什么动作；操作数表示操作的对象是什么，例如寄存器中保存的数据、立即数等。计算机能识别的指令是由 0 和 1 构成的字符串，称为机器指令。指令的长度通常是一个或几个字长，长度可以是固定的，也可以是可变的。图 5-6(b)给出了某款 CPU 的加法指令示例。该指令长度为 16 位(一个字长)，从左至右标示各位为 bit 15～bit 0。bit 15～bit 12 代表的是操作码，为 0001，在该 CPU 中表示加法操作。bit 11～bit 0 对应操作数的表示，由于该指令需要 3 个操作数，bit 11～bit 0 将会被拆分为 3 段，分别对应两个相加数(源操作数)和一个求和结果(目的操作数)。bit 11～bit 9 对应保存目的操作数的寄存器地址，在该示例中为 110，表示寄存器 R6，bit 8～bit 6 与 bit 2～bit 0 分别对应保存源操作数的寄存器地址，分别为 R2 和 R6。则这条指令表示将寄存器 R6 和 R2 中保存的数值进行加运算，结果保存回寄存器 R6。bit 5～bit 3 用于扩展加法指令的操作，此处不做解释。



(a) 指令的一般格式

(b) 加法的机器指令示例

ADD R6，R2，R6

(c) 加法的汇编指令示例

图 5-6　指令

机器指令由 0 和 1 字符构成，计算机易于阅读和理解，但是，不适合人使用。所以，在

指令中引入助记符表示操作码和操作数，以帮助人理解和使用指令。这样的指令称为汇编指令。如图 5-6(c)所示，用 ADD 来标示该指令是加法指令，R6 和 R2 标示用到的寄存器。计算机不能直接执行汇编指令，要由汇编器将其翻译成对应的机器指令才可执行。对图 5-6(c)的 ADD 指令，汇编器会将其翻译成图 5-6(b)的形式。

CPU 的指令是由指令集体系结构(Instruction Set Architecture，ISA)规定的。每款 CPU 在设计时就规定了一系列与其硬件电路相配合的指令系统。ISA 是与程序设计有关的计算机结构的一部分，定义了指令类型、操作种类、操作数数目与类型，以及指令格式等，可用 CPU 指令集的指令来编写程序。程序就是用于控制计算机行为完成某项任务的指令序列。在指令集中，通常定义的指令类型有 3 种。

(1) 操作指令：操作指令是处理数据的指令，例如算术运算和逻辑运算都是典型的操作指令。

(2) 数据移动指令：它的任务是在通用寄存器组和主存之间、寄存器和输入输出设备之间移动数据。例如，将数据从主存移入寄存器的 LOAD 指令，以及反方向移动数据的 STORE 指令等。

(3) 控制指令：能改变指令执行顺序的指令。例如无条件跳转指令，将程序计数器的值更改为一个非顺序的值，使得下一条指令从新位置开始。

图 5-7 是一个程序示例，为了便于阅读，采用了汇编指令编写，分号后面是程序的注释，帮助人们阅读和理解程序，而计算机将忽略这些注释。

```
        mov #0, R0      ; 将寄存器R0置为0
        mov #1, R1      ; 将寄存器R1置为1
loop:   add R1, R0      ; 将R1与R0相加，结果保存到R0
        add #1, R1      ; R1加1
        cmp R1, #1000   ; 比较R1与1000的大小
        ble loop        ; 如果R1小于或等于1000，从loop那条指令开始执行
        halt            ; 程序结束
```

图 5-7　程序示例

这段程序用于计算 1＋2＋…＋1000 的值。利用寄存器 R1 和 R0，开始时将 R0 设为 0，R1 设为 1。然后将 R1 的值加到 R0 上，同时 R1 增 1。此后将 R1 的值与 1000 进行比较，如果 R1 比 1000 小，则重复执行将 R1 加到 R0 上以及 R1 增 1 的操作，然后再比较。这种重复执行将在 R1 大于 1000 时结束，同时将结束程序的运行。程序中，add 是操作指令，ble 是控制指令。ble 与 cmp 一起使用，当 cmp 的比较结果小于或等于 1000 时，该指令被执行，将执行顺序跳转到 loop 所标示的指令。

CPU 的工作过程是循环执行指令的过程。指令的执行过程是在控制单元的控制下，精确地、一步一步地完成的，称这个执行的步骤顺序为指令周期，其中的每一步称为一个节拍。不同的 CPU 可能执行指令的节拍数不同，但是通常都可归为以下 4 个阶段(见图 5-8)。

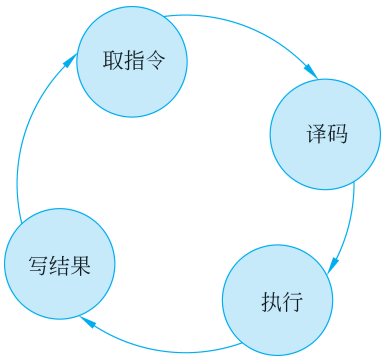


图 5-8　指令执行常见节拍划分

(1) 取指令：指令通常存储在主存中，CPU 通过程序计数器获得要执行的指令存储地址。根据这个地址，CPU 将指令从主存中读入，并保存在指令寄存器中。

(2) 译码：由指令译码器对指令进行解码，分析出指令的操作码和所需的操作数存放的位置。

(3) 执行：将译码后的操作码分解成一组相关的控制信号序列，以完成指令动作，包括从寄存器读数据、输入 ALU 进行算术或逻辑运算。

(4) 写结果：将指令执行节拍产生的结果写回到寄存器，如果有必要，将产生的条件反馈给控制单元。

以上的节拍划分是粗粒度的，通常每个节拍所包含的动作很难在一个时钟周期内完成。因此，会进一步将每个节拍进行细化，细化后的每个动作可在一个时钟周期内完成，不可再细分。例如，取指令阶段可以再细分如下。

(1) 将程序计数器的值装入主存的地址寄存器。

(2) 将地址寄存器所对应的主存单元的内容装入主存数据寄存器。

(3) 控制单元将主存数据寄存器的内容装入指令寄存器，同时对程序计数器“增 1”。

可见，取指令这个节拍要花费 3 个时钟周期。对现代计算机来说，每个时钟周期非常短。例如对主频为 3.3GHz 的 CPU，每秒将完成 33 亿个时钟周期，每个时钟周期的时间长度为 0.303ns，而取指令节拍将花费 0.909ns。

在最后一个节拍完成后，控制单元复位指令周期，从取指令节拍重新开始运行，此时，程序计数器的内容已被自动修改，指向下一条指令所在的主存地址。操作指令和数据移动指令的执行不会主动修改程序计数器的值，程序计数器将会自动指向程序顺序上的下一条指令。而控制指令的执行将会主动改变程序计数器的值，使得程序的执行将不再是顺序的。

以图 5-7 的程序为例来理解 CPU 的工作过程。假设这段程序存放在主存中的存储形式如图 5-9 所示。开始执行这段代码时，将会由操作系统将程序计数器的值设为 A0，在取指令阶段将该地址的指令“mov ＃0，R0”取出存入指令寄存器，同时程序计数器“增 1”为 A1。mov 指令经译码后，在控制单元控制下将寄存器 R0 置为 0。此时指令执行结束，控制单元复位，从取指令重新执行——根据程序计数器的值 A1 取下一条指令。该过程将一直执行到 ble 指令。该指令执行完后，将会对程序计数器进行覆盖，将 loop 对应的指令地址写入程序计数器，使得下一条指令将不再是顺序执行的，而是跳转到 loop 指令开始执行。当条件满足时，ble 指令的执行不修改程序计数器的值，此时，将取 halt 指令

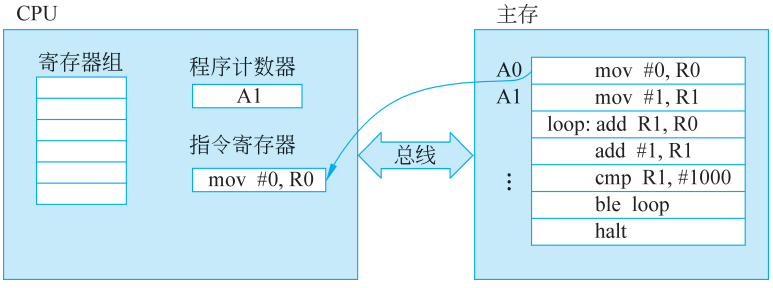


图 5-9　程序在主存中的存储形式

开始执行。

5.2.2 存储系统

计算机系统中的存储器一般分为主存(又称为内存)和辅存(又称为外存),主存可与CPU直接进行信息交换,其特点是运行速度快,容量相对较小,在系统断电后,其保存的内容会丢失。辅存属于外围设备的范畴,如硬盘、光盘等,它们通过各种专门接口与计算机通信。辅存与CPU之间不能直接交换数据,其特点是存储容量大,存取速度比主存慢,系统断电后其保存的信息不会丢失,存储的信息很稳定。

主存储器的一般结构如图5-10所示,包括用于存储数据的存储体和外围电路,外围电路用于数据交换和存储访问控制,与CPU或高速缓存连接。外围电路中有两个非常重要的寄存器——数据寄存器MDR(Memory Data Register)和地址寄存器MAR(Memory Address Register),前者是用于临时保存读出或写入的数据,后者用于临时保存访问地址。要访问主存时,首先将要访问的地址送入MAR,如果是读主存,则在控制电路的控制下,将MAR指向的主存单元数据送入MDR,然后发送到CPU或高速缓存;如果是写主存,则首先要将需写入的数据送到MDR,在控制电路控制下,将MDR数据写入MAR指向的主存单元。

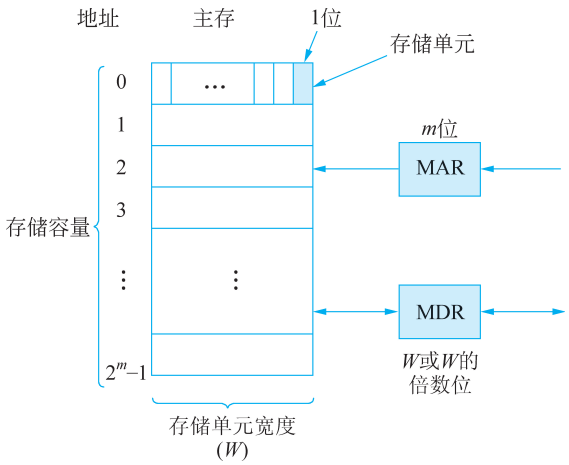


图5-10 主存储器的一般结构

主存中存储的最基本单元是一个0-1符号串,可以代表数字、字符等信息。存储器由很多可存放长度(位数)相同的0-1符号串的单元组成,称为主存单元,每个主存单元有一个编号,这个编号就是主存地址。主存地址用二进制数来表示,如果表示地址的二进制数有 m 位,则主存地址最大可编码到 2^m-1(从0开始编码),也就是说最多可以有 2^m 个主存单元,称为存储容量。可以通过主存地址来对主存单元存放的0-1符号串进行读写,这种读写操作通常被称为访问主存。访问主存时可根据地址独立地对各单元数据进行读写,访问时间与被访问地址无关,因此,主存又称为随机访问存储器(Random Access Memory,RAM)。为了规整化,主存单元的长度一般标准化为8位,即一字节(Byte),再

由字节组合成字。

主存中存储电路的原理类似于电容，主存中通过对电路进行充电来存储信息，但是这很容易流失。因此，需要在很短的时间内不断地充电，称为刷新。采用这种技术的主存又称为动态存储器(Dynamic RAM)。

根据存储能力与电源的关系可将主存分为易失性存储器和非易失性存储器，计算机系统主存一般都包含这两类存储器。前者指的是当电源供应中断后，存储器所存储的数据便会消失的存储器，如 RAM、DRAM 等，断电后保存的信息将会丢失。后者指即使电源供应中断，存储器所存储的数据并不会消失，重新供电后，就能够读取其中数据的存储器，如只读存储器(Read-Only Memory，ROM)等，断电后保存的信息不会丢失，ROM 也可随机访问。

除主存容量外，主存的另两个重要指标是存储器访问时间和存储周期。存储器访问时间指从启动一次存储器操作到完成该操作所经历的时间。具体讲，从一次读操作命令发出到该操作完成，将数据读入数据寄存器为止所经历的时间。存储周期指连续启动两次独立的存储器操作(如连续两次读操作)所需间隔的最小时间，通常，存储周期略大于存储器访问时间。目前，主存访问速度总比 CPU 速度慢得多。一次访问时间为 5～10ns，比 CPU 的速度慢很多。

由于电源线的尖峰电压或被高能粒子冲击等原因，主存中偶尔也会出错，即保存的信息在某个瞬间由 0 变为 1 或由 1 变成了 0。主存中经常采用检错码或纠错码，即在存储的信息中附加一些位，用于检测主存是否出错，其中检错码能检测出 1 位或多位错，而纠错码能在检测出错误后将出错位改回其正确值。以最常用的奇偶校验码为例，它是在原数据基础上，附加上 1 位奇偶校验位。根据原数据中 1 的位数来确定校验位，使整个码字中 1 的位数为偶数(或奇数)。当某一位出错时，造成校验位不正确，以此检测出发生了错误。例如，假设字节中保存的信息为 11100101，该数有 5 个 1，为奇数，如果采用偶校验，则校验位应为 1；如果采用奇校验，校验位为 0。当该字节被读出时，会再次对原有的 8 位进行判定，如果某 1 位由 1 变成 0 或由 0 变成 1，则计算出的奇偶校验与原校验位不符，表示发生了错误。至于要进行纠错，则需要更复杂和强大的编码。

一直以来，CPU 的速度总比主存访问速度快。虽然随着工艺水平的提高，主存的访问速度也在不断提高，但是仍然赶不上 CPU 速度的提高。CPU 发出访问主存请求后，往往要等多个时钟周期后才能得到主存内容。存储器越慢，CPU 等待的时间就越长。目前，技术上的解决办法是利用更小更快的存储设备与大容量低速的主存组合使用，以适中的价格得到速度和高速存储器差别不大的大容量存储器。

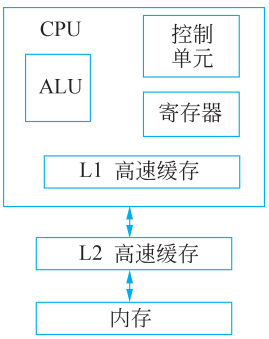


图 5-11 典型的高速缓存配置

这种更小更快的存储设备称为高速缓存存储器(Cache)，简称高速缓存。高速缓存逻辑上介于 CPU 和主存之间，可以将其集成到 CPU 内部，也可置于 CPU 之外。图 5-11 给出了一种典型的高速缓存配置方式。位于 CPU