第5章 计算机系统组成与结构

【学习内容】

本章介绍计算机系统的相关内容，主要知识点包括如下。

(1) 计算机系统的基本概念及其组成。

(2) 冯·诺依曼体系结构及其工作机制。

(3) 中央处理器的结构和工作原理。

(4) 存储系统的基础知识与工作原理。

(5) 总线结构、工作过程及常用标准。

(6) 输入输出控制方式。

(7) 计算机软件系统的分类、层次结构及主要功能。

(8) 计算思维在计算机系统中的体现。

【学习目标】

通过本章的学习，读者应掌握以下内容。

(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，掌握基本术语和一些指标的计算方法。

(6) 理解总线结构、工作原理以及评价指标。

(7) 了解 CISC、RISC、流水及并行处理等概念。

(8) 了解计算机软件系统的分类、层次结构及主要功能。

(9) 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。

本章主要介绍信息处理的核心装置——计算机的硬件系统，包括其结构、如何支持信息处理，以及各部分在信息处理中的作用。首先从全局角度介绍计算机系统的体系结构，以冯·诺依曼体系结构为依据，介绍计算机系统的硬件构成。然后围绕该体系结构的各部件，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以及基本工作原理，并用模拟的方法进行研究。最后介绍软件系统的基础知识。

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，它包括硬件和软件两大部分。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部设备，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通常由图 5-1 所示的结构构成。

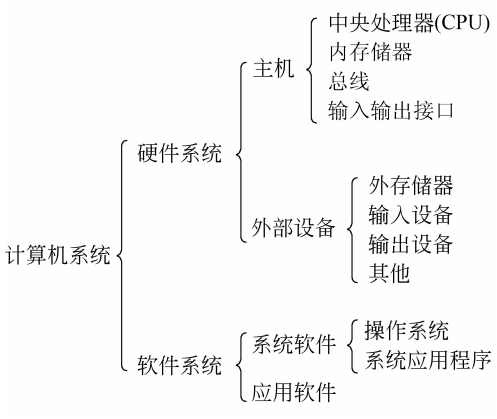

图 5-1　计算机系统的构成

5.1.1　计算机硬件系统的结构

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·诺依曼等在 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，并将各类运算转化为一组指令序列，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，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、输入和输出设备（见图 5-2）。在该体系结构中，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。

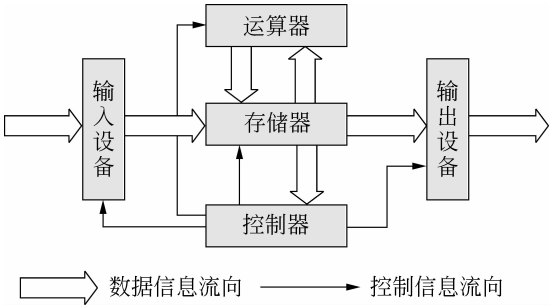

图 5-2　冯・诺依曼体系结构

在冯・诺依曼体系结构形成之前，人们将数据存储于主存中，而程序被看成是控制器的一部分，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系统（Symmetric Multi Processing，SMP）、大规模并行处理系统（Massively Parallel Processing，MPP）和集群等。

可以进一步将图 5-2 的冯・诺依曼体系结构细化，就得到了典型的计算机硬件组织结构（见图 5-3）。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单元，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。

下面结合该图，给出一些基本概念，这些概念将在后续几节中详细介绍。

1. 中央处理器

一般把中央处理器（Central Processing Unit，CPU）简称为处理器，它是执行存储在主存中的指令的引擎。处理器主要由控制单元、算术逻辑运算单元和寄存器组（分为通用寄存器和专用寄存器）构成。

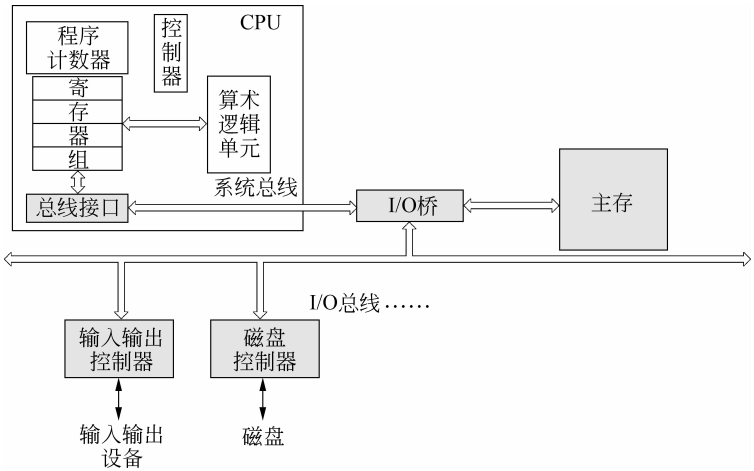

图 5-3　典型的计算机硬件组织结构

2. 主存

主存(又称为内存)是一个临时存储设备,在计算机执行程序过程中,用于存放程序和程序所处理的数据。逻辑上来说,主存是一个线性编组的单元格序列,每个单元格的长度是一个字节。每个单元格都有一个唯一的编号,即主存地址,地址是从零开始编号的。

3. 总线

总线是连接计算机各部件的是一组电子管线,它负责在各个部件之间传递信息。

4. 输入输出设备

输入输出设备是计算机与外界的联系通道,如用于用户输入的鼠标和键盘,用于输出的显示器,以及用于长期存储数据和程序的磁盘。每个输入输出设备通过一个控制器或适配器与输入输出总线连接。

5.1.2　计算机软件

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。严格来说,软件是指计算机系统中的程序、要处理的数据及其相关文档。程序是计算任务的处理对象和处理规则的描述,数据是使程序能正常操纵信息的数据结构,文档是为了便于理解程序所需的描述或说明性资料。程序必须存入计算机内部才能工作,文档一般是给人看的,不一定存入计算机。

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。

软件系统通常分为系统软件、支撑软件和应用软件。软件系统与硬件系统，以及软件系统之间的关系如图 5-4 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，以及特定的程序，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。

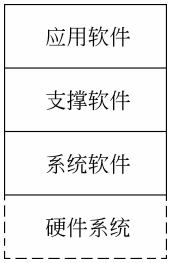

图 5-4　计算机软件系统结构

5.2　中央处理器

5.2.1　CPU 的结构

现代计算机的核心是 CPU，这是执行指令的地方，通过执行指令，控制各类硬件协同完成任务。CPU 一般由算术逻辑运算器(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。

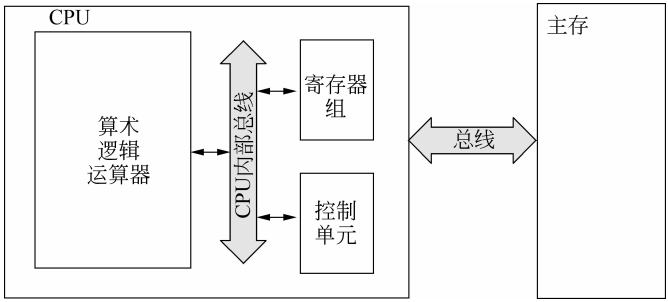

图 5-5　CPU 的内部结构

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，用于标识一些特殊情况，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，状态寄存器保存标识 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，以标识访问哪个寄存器，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 中所有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；地址总线用于传送 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。

CPU 处理的指令和数据来自于主存。由于 CPU 和主存之间的速度差异，直接对主存进行数据读写将会迟滞 CPU 的性能。所以，常常在 CPU 内部设计高速缓存——Cache，来补偿 CPU 访问主存的开销，并且常常设计多级 Cache。

5.2.2 指令系统

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 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，也可是可变的。图 5-6(b)给出了某款 CPU 的加法指令的示意图。该指令长度为 16 位(一个字长)，从左至右标识各位为 bit15～bit0。bit15～bit12 代表的是操作码，为 0001，在该 CPU 中表示加法操作。bit11～bit0 对应操作数的表示，由于该指令需要 3 个操作数，bit11～bit0 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)。bit11～bit9 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，bit8～bit6 与 bit2～bit0 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。bit5～bit3 用于扩展加法指令的操作，此处不做解释。

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

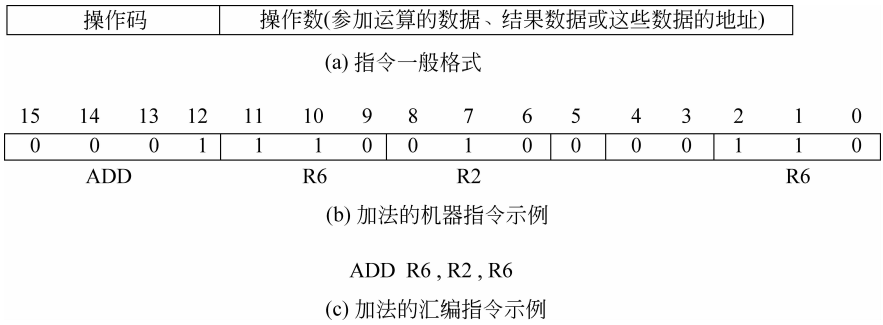

图 5-6　指令

编指令。如图 5-6(c)所示,用 ADD 来标识该指令是加法指令,R6 和 R2 标识用到的寄存器。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、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: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,和反方向移动数据的 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。

在使用计算机时,人们可以用该计算机所装备的 CPU 的指令集中的指令编写程序。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 比较结果为小于等于 1000 时，该指令被执行，将执行顺序跳转到 loop 所标示的指令。

通常，用机器指令和汇编指令来编写程序是非常困难的，会把大量的精力花在记忆指令格式、操作码和操作数等与程序功能无关的方面。为此，设计了更加贴近于自然语言的程序设计语言，使人们在编程时，更多地关注程序要完成的任务，这种语言称为高级语言。用高级语言编写的程序经过编译器编译和链接，即可生成计算机能识别的机器指令构成的程序。

5.2.3 CPU 的工作过程

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)。

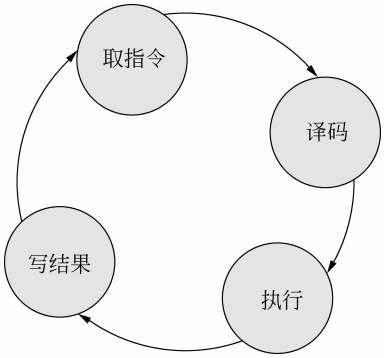

图 5-8　指令执行常见节拍划分

(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，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、输入到 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为如下。

(1) 将程序计数器的值装入到主存的地址寄存器。

(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排列形式如图 5-9 所示。要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 指令开始执行。

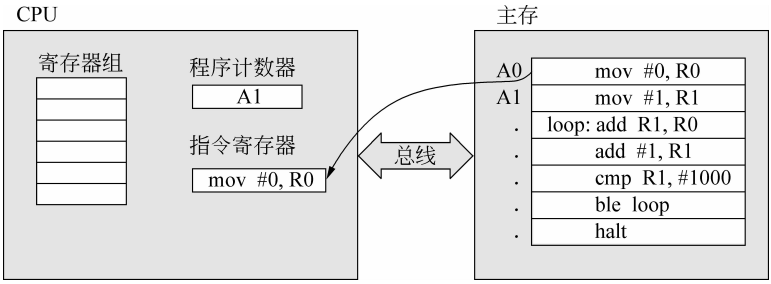

图 5-9　程序在主存中的存储形式

5.2.4　CPU 高级话题

1. CISC 与 RISC

CISC(Complex Instruction Set Computer)即复杂指令计算机，其设计理念是要用最少的机器指令来完成所需的计算任务，所以设计上用专用指令来完成特定的功能，以提高计算机的处理效率。这样，随着 CPU 设计的发展，不得不将越来越多的复杂指令加入到指令系统中。CISC 带来的好处是减少了程序设计语言和机器语言之间的语义差别，而且简化了编译器的结构，但同时增加了微处理器结构的复杂性。Intel 公司的 x86 系列 CPU 是典型的 CISC 体系的结构，从最初的 8086 到后来的 Pentium 系列，每出一代新的 CPU，都会有自己新的指令。

RISC(Reduced Instruction Set Computer)即精简指令集计算机，其设计思路尽量简化计算机指令功能，只保留那些功能简单、能在一个节拍内执行完成的指令，而把较复杂的功能用一段子程序来实现。所以，在 RISC 机器上实现特殊功能时，效率可能较低。但可以利用流水技术和超标量技术加以改进和弥补。RISC 带来的好处是使 CPU 设计实现更容易，指令并行执行程度更好，编译器的效率更高。常用的精简指令集微处理器包括 ARM、MIPS、PowerPC 和 SPARC 等。

2. 指令流水线

人们已经知道，从主存中取指令的过程是提高指令执行速度的瓶颈。一种解决办法

是在 CPU 中增加一组被称为预取缓冲的寄存器，用于保存从主存中预先取出的指令，这样需要执行指令时，直接从预取缓冲中取得，不必等待一个读主存周期。预取缓冲的作用是将取指令的时间开销缩短为 CPU 内部操作的时间开销，为指令流水线的实现提供了基础。

指令流水线是在指令执行周期分节拍的基础上，将指令执行分解成更细的步骤（10 个或更多个），每个步骤由精心设计的硬件分别执行，使得同一时刻 CPU 能执行多条指令，实现指令级并行。

图 5-10(a)给出了将指令执行分解为 5 个节拍的流水执行过程。节拍 1 从主存中取指令到预取缓冲备用，节拍 2 对指令进行译码，节拍 3 从寄存器找到并取出操作数，节拍 4 完成实际指令功能，节拍 5 将结果写回目的寄存器。并且假设每个节拍可在一个时钟周期内完成。

图 5-10(b)给出了随时间变化的流水执行过程。S1～S5 表示 5 个节拍，时间轴每一格代表一个时钟周期，带方框的数字表示第几条指令。在时钟周期 1，指令 1 处于指令执行的第 1 个节拍——取指令。到第 2 个时钟周期时，指令 1 处于第 2 个节拍——译码，而此时用于取指令的部件已经空闲下来，可以用于执行其他指令，图中第 2 个时钟周期开始执行第 2 条指令的取指令节拍。第 3 个时钟周期，可以同时执行指令 1 的第 3 节拍、指令 2 的第 2 节拍，以及指令 3 的第 1 节拍。以此类推，这 5 个节拍构成了指令流水线。

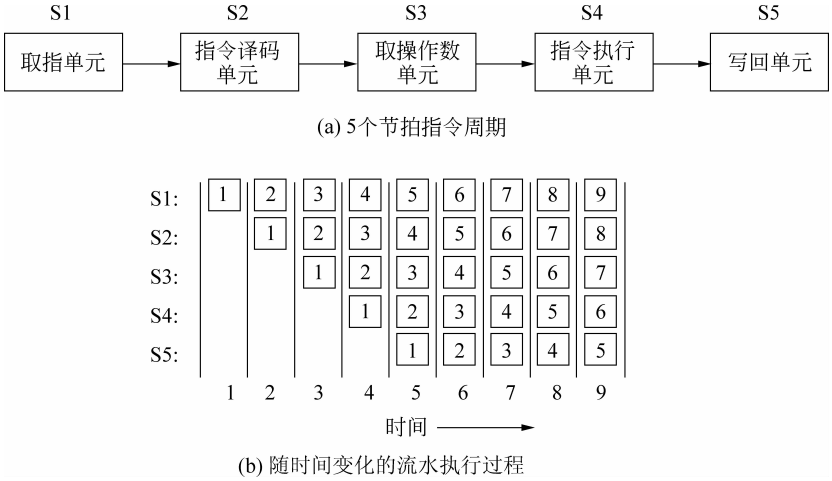

图 5-10　5 个节拍指令周期和随时间变化的流水执行过程

假设该 CPU 的时钟周期为 1ns，则一条指令经过完整流水线需要 5ns。但是相比于串行执行每 5ns 才执行完一条指令，流水线以 1ns 执行完一条指令的速度运行，速度接近原来的 5 倍。

更进一步，既然一条流水线可以提高 CPU 的性能，那么多条流水线更能提高性能了，这就是超标量体系结构的核心思想。图 5-11 是双流水线的一种设计示例。该设计中两条流水线共用一个取指令部件，可以一次取两条指令，然后分别将指令送到各自的流水线执行。要实现这种设计，需要多个功能单元来同时执行指令，并且同时执行的指令不能