第一篇

基础篇

首篇共分3章，包括微机系统导论、微机运算基础与标准微处理器，它们是学习微机原理及应用教材的公共基础知识，也是一个必要的入门。

第一章　微机系统导论

本章首先从总体上说明微型计算机（简称微机）系统组成的基本概念，并对硬件系统和软件系统两大部分的具体组成作一简要介绍。

然后，重点讨论典型的单总线微机硬件系统结构，微处理器组织及各部分的作用，存储器组织及其读写操作过程。在此基础上，将微处理器和存储器结合起来组成一个最简单的微机模型，通过具体例子说明微机的运行机理与工作过程。

最后，给出一个实际微处理器 Z80 的程序设计模型和评价微机系统性能的主要技术指标。

§1.1　微机系统组成

一、几个基本定义

微处理器、微型计算机和微型计算机系统，这是三个含义不同但又有着密切依存关系的基本概念。

（一）微处理器

微处理器简称 μP 或 MP (Microprocessor)，是指由一片或几片大规模集成电路组成的具有运算器和控制器功能的中央处理器部件，又称为微处理机。它本身并不等于微型计算机，而只是其中央处理器。有时为区别大、中、小型中央处理器 CPU (Central Processing Unit) 与微处理器，而称后者为 MPU (Microprocessing Unit)。通常，在微型计算机中直接用 CPU 表示微处理器。

（二）微型计算机

微型计算机 (Microcomputer)，简称 μC 或 MC，是指以微处理器为核心，配上存储器、输入/输出接口电路及系统总线所组成的计算机（又称主机或微电脑）。当把微处理器、存储器和输入/输出接口电路统一组装在一块或多块电路板上或集成在单片芯片上，则分别称之为单板、多板或单片微型计算机。

（三）微型计算机系统

微型计算机系统 (Microcomputer system)，简称 μCS 或 MCS，是指以微型计算机为中心，配以相应的外围设备、电源和辅助电路（统称硬件）以及指挥微型计算机工作的系统软件所构成的系统。

以上三者的含义及相互关系如图 1.1 所示。

二、微型计算机系统的组成

微型计算机系统与任何其它计算机系统一样，由硬件和软件两个主要部分组成。

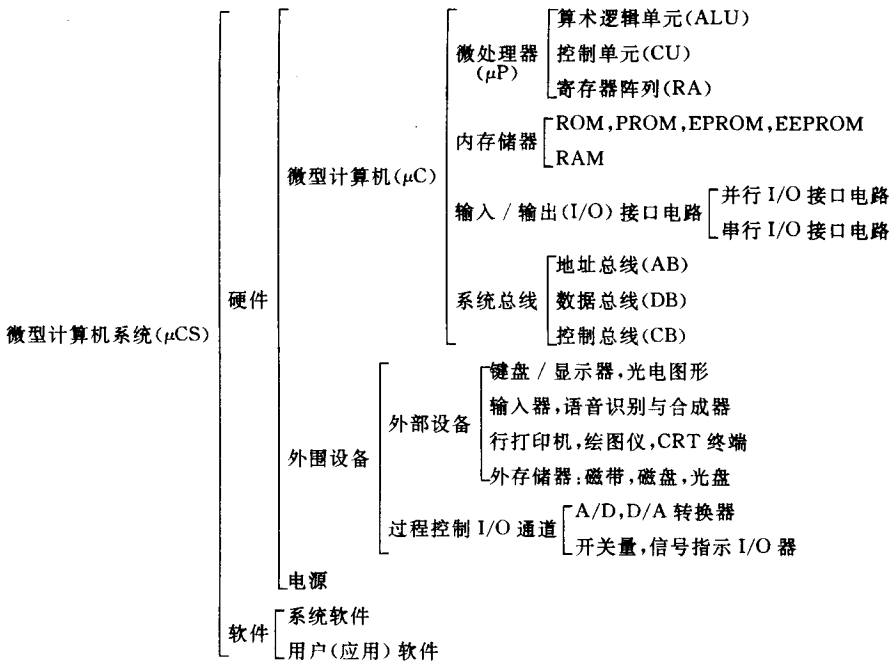

图 1.1 μCS，μC，μP 的相互关系

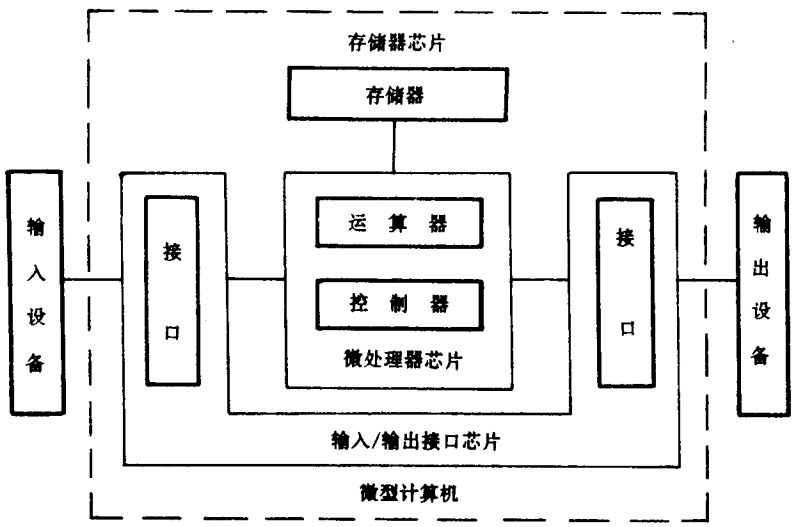

图 1.2 微机硬件系统组成

（一）硬件

微机硬件系统的组成如图 1.2 所示。图中，微处理器是微机的运算、控制中心，用来实现算术、逻辑运算，并对全机进行控制。存储器（简称主存或内存）用来存储程序或数据。

输入/输出（I/O）芯片是微机与输入输出设备之间的接口。

（二）软件

计算机软件通常分为两大类：系统软件和用户软件。系统软件是指不需要用户干预的能生成、准备和执行其它程序所需的一组程序。用户软件是各用户为解题或实现检测与实时控制等不同任务所编制的应用程序。究竟应配置多少系统软件才能满足特定计算机系统的需要，这取决于具体的用途。程序的分级结构如图 1.3 所示。

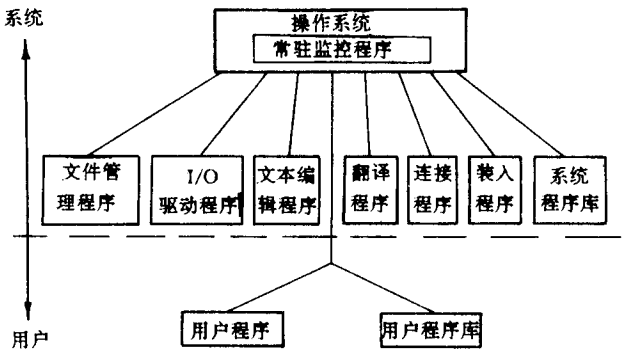

图 1.3　软件的分级结构

操作系统是一套复杂的系统程序，用于提供人机接口和管理、调度计算机的所有硬件与软件资源。它所包含的系统程序的具体分类尚不完全统一。其中，最为重要的核心部分是常驻监控程序。计算机开机后，常驻监控程序始终存放在内存中，它通过接收用户命令，并启动操作系统执行相应的操作。

操作系统还包括 I/O 驱动程序和文件管理程序。前者用于执行 I/O 操作；后者用于管理存放在外存（或海量存储器）中的大量数据集合。每当用户程序或其它系统程序需要使用 I/O 设备时，通常并不是由该程序执行操作，而是由操作系统利用 I/O 驱动程序来执行任务。文件管理程序与 I/O 驱动程序配合使用，用于文件的存取、复制和其它处理。

此外，系统软件还可包括各种高级语言翻译程序、汇编程序、文本编辑程序以及辅助编写其它程序的程序。程序设计分为三级：

1. 机器语言程序设计；
2. 汇编语言程序设计；
3. 高级语言程序设计。

机器语言程序是计算机能理解和直接执行的程序。汇编语言程序是用助记符语言表示的程序，计算机不能直接“识别”，需经过称之为汇编程序的翻译把它转换为机器语言方能执行。机器语言指令与汇编语言指令基本上一一对应，都面向机器。而高级语言是不依赖于具体机型只面向过程的程序设计语言，由它所编写的程序，需经过编译程序或解释程序的翻译方能执行。

文本编辑程序是供输入或修改文本（字母、数字和标点等组成的一组字符或代码序列）用的程序，存于海量存储器中；它有几种用途，主要可用来生成程序。

在编写程序时，还可能需要另外两种系统程序：系统程序库；连接程序与装入程序。一

般操作系统都有一个通用的系统程序库，用户还可以建立自己的程序库（一组子程序）。程序库中的子程序可附在任何系统程序或用户程序上以供调用。把待执行的程序与程序库及其它已翻译好的程序连接起来所用的准备程序称为连接程序或连接编辑程序；另一种准备程序是用来把待执行的程序送入内存，称为装入程序。有时，连接与装入功能可合成为一个程序。

应当指出，硬件系统和软件系统是相辅相成的，共同构成微型计算机系统，缺一不可。现代的计算机硬件系统和软件系统之间的分界线并不明显，总的趋势是两者统一融合，在发展上互相促进。

人是通过软件系统与硬件系统发生关系的。通常，由人使用程序设计语言编制应用程序，在系统软件的干预下使用硬件系统。

一个具体的微型计算机系统，它所包括的硬件和软件数量各不相同，究竟应包括多少，要根据应用场合对系统功能方面的要求来确定。

§1.2 微机硬件系统结构

所谓微机硬件系统结构系指按照总体布局的设计要求将各部件构成某个系统的连接方式。一种典型的微机硬件系统结构如图 1.4 所示。图中，用系统总线将各个部件连接起来。

系统总线是用来传送信息的公共导线，它们可以是电缆，也可以是印刷板上的连线。所有的信息都通过总线传送。通常，根据所传送信息的内容与作用不同，可将总线分为三类：数据总线 DB (Data Bus)，地址总线 AB (Address Bus)，控制总线 CB (Control Bus)。这是一种单总线系统结构，系统中各部件均挂在单总线上，所以又称为面向系统的单总线结构。

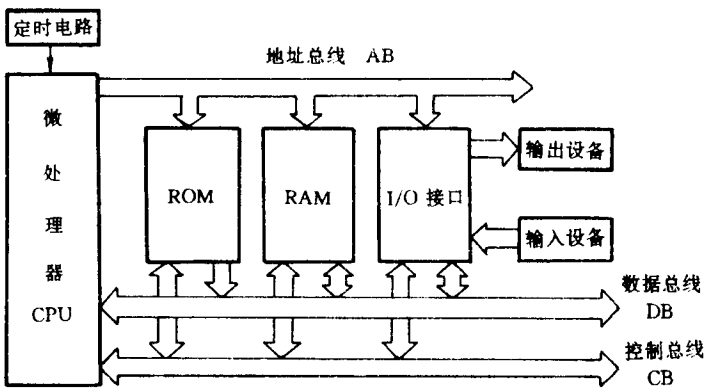

图 1.4 典型的微机硬件系统结构

在微型计算机中有两股信息流（数据信息流和控制信息流）在流动。在单总线系统中，通过单总线实现微处理器、存储器和所有 I/O 设备之间的信息交换。由于各部件均以同一形式挂在单总线上，结构简单、易于扩充，所以目前绝大多数微机硬件系统均采用这种结构。

§1.3 微处理器组成

图 1.5 给出了一个简化的微处理器结构。由图中可知，微处理器由运算器、控制器和内部寄存器阵列三部分组成。现将各部件的功能简述如下。

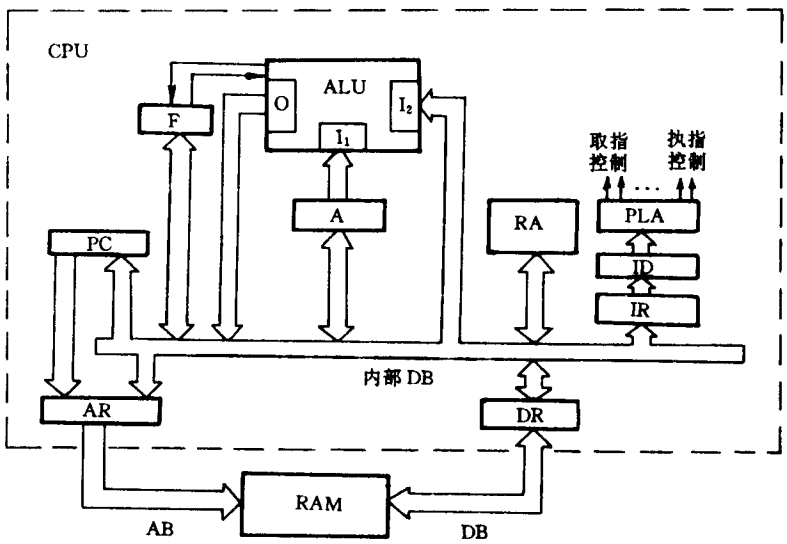

图 1.5 微处理器结构

一、运算器

运算器又称为算术逻辑单元 ALU (Arithmetic Logic Unit)，用来进行算术或逻辑运算以及位移循环等操作。参加运算的两个操作数，通常，一个来自累加器 A (Accumulator)，另一个来自内部数据总线，可以是数据寄存器 DR (Data Register) 中的内容，也可以是寄存器阵列 RA 中某个寄存器的内容。运算结果往往也送回累加器 A 保存。

二、控制器

（一）指令寄存器 IR (Instruction Register)

指令寄存器 IR 用来存放从存储器取出的将要执行的指令（实为其操作码）。

（二）指令译码器 ID (Instruction Decoder)

指令译码器 ID 用来对指令寄存器 IR 中的指令进行译码，以确定该指令应执行什么操作。

（三）可编程逻辑阵列 PLA (Programmable Logic Array)（也称为定时与控制电路）

可编程逻辑阵列用来产生取指令和执行指令所需的各种微操作控制信号。由于每条指令所执行的具体操作不同，所以，每条指令将对应控制信号的某一种组合，以确定相应的操作序列。

三、内部寄存器阵列

通常，内部寄存器阵列包括若干组寄存器。

（一）累加器 A

累加器是用得最频繁的一个寄存器。在进行算术逻辑运算时，它具有双重功能：运算前，用来保存一个操作数；运算后，用来保存结果。

（二）数据寄存器 DR

数据寄存器 DR 用来暂存数据或指令。从存储器读出时，若读出的是指令，经 DR 暂存的指令通过内部数据总线送到指令寄存器 IR；若读出的是数据，则通过内部数据总线送到有关的寄存器或运算器。

向存储器写入数据时，数据是经数据寄存器 DR，再经数据总线 DB 写入存储器的。

（三）程序计数器 PC（Program Counter）

程序计数器 PC 中存放着正待执行的指令的地址。根据 PC 中的指令地址，准备从存储器中取出将要执行的指令。通常，程序按顺序逐条执行。任何时刻，PC 均指示要取的下一个字节或下一条指令（对单字节指令而言）所在的地址。因此，PC 具有自动加 1 的功能。

（四）地址寄存器 AR（Address Register）

地址寄存器 AR 用来存放正要取出的指令的地址或操作数的地址。

在取指令时，将 PC 中存放的指令地址送到 AR，根据此地址从存储器中取出指令。

在取操作数时，将操作数地址通过内部数据总线送到 AR，再根据此地址从存储器中取出操作数；在向存储器存入数据时，也要先将待写入数据的地址送到 AR，再根据此地址向存储器写入数据。

（五）标志寄存器 F（Flag Register）

标志寄存器 F 用来寄存执行指令时所产生的结果或状态的标志信号。关于标志位的具体设置与功能将视微处理器的型号而异。根据检测有关的标志位是 0 或 1，可以按不同条件决定程序的流向。

§1.4 存 储 器

一、基本概念

存储器用来存放数据和程序。在计算机内部，数据和程序都用二进制代码的形式表示。

在微机中，一般用 8 位二进制代码作为一个字节（Byte）。用一个或几个字节组成一个字（Word）。如果用字表示一个数，称为数据字；表示一条指令，称为指令字。数据字和指令字也可以用双倍字长或多倍字长表示。

微机的字长多为 8 位和 16 位，高档微机的字长可达 32 位。目前，用于工业控制的微机其典型的字长为 8 位。

一个存储器可划分为很多存储单元。存储单元中的内容为数据或指令。为了能识别不同的单元，我们分别赋予每个单元一个编号。这个编号称之为地址。显然，各存储单元的地址与该地址中存放的内容是完全不同的意思，不可混淆。

二、存储器组成

现假定存储器由 256 个单元组成，每个单元存储 8 位二进制信息，即字长为 8 位，其结构简图如图 1.6 所示。这种规格的存储器，通常称为 256×8 位的读/写存储器。

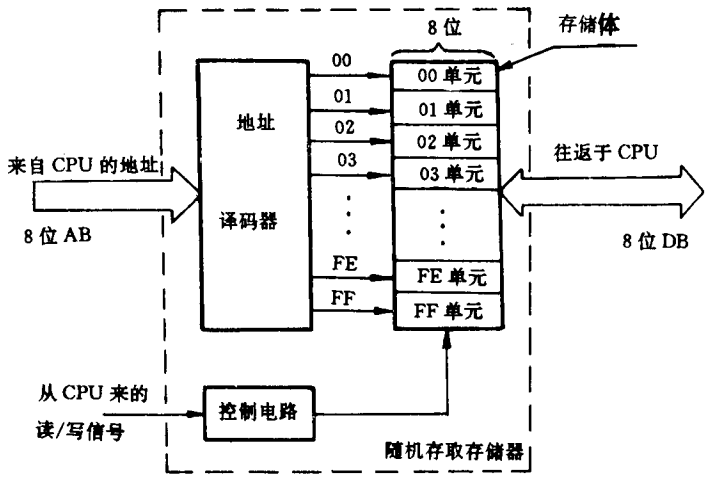

图 1.6　随机存取存储器结构简图

从图中可见，随机存取存储器由存储体、地址译码器和控制电路组成。

存储体共有 256 个存储单元，其编号从 00H（十六进制表示）到 FFH，即从 00000000 到 11111111。

地址译码器接收从地址总线 AB 送来的地址码，经译码器译码选中相应的某个存储单元，以便从中读出信息或写入信息。

控制电路用来控制存储器的读/写操作过程。

三、读/写操作过程

从存储器读出信息的操作过程如图 1.7（a）所示。假定 CPU 要读出存储器 04H 单元

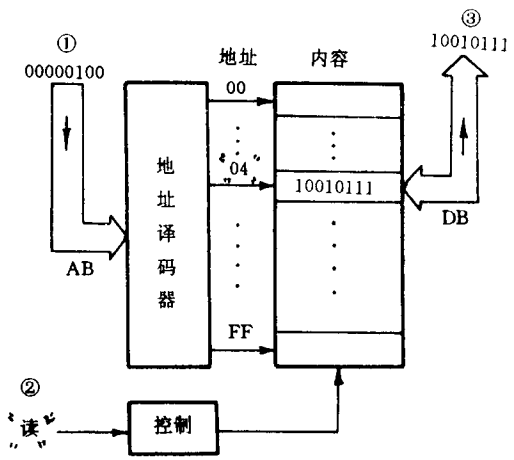

图 1.7（a）　存储器读操作过程示意图

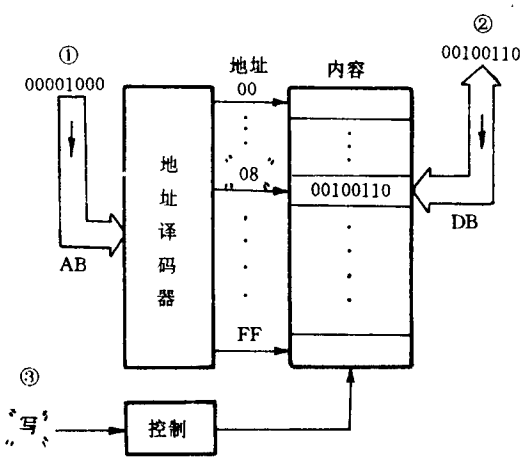

图 1.7（b）　存储器写操作过程示意图

的内容 10010111 即 97H，则：(1) CPU 的地址寄存器 AR 先给出地址 04H 并放到地址总线上，经地址译码器译码选中 04H 单元；(2) CPU 发出“读”控制信号给存储器，指示它准备把被寻址的 04H 单元中的内容 97H 放到数据总线上；(3) 在读控制信号的作用下，存储器将 04H 单元中的内容 97H 放到数据总线上，经它送至数据寄存器 DR，然后由 CPU 取走该内容作为所需要的信息使用。

应当指出，读操作完成后，04H 单元中的内容 97H 仍保持不变，这种特点称为非破坏性读出 (NDRO—Non Destructive Read Out)。这一特点很重要，因为它允许多次读出同一单元的内容。

向存储器写入信息的操作过程如图 1.7 (b) 所示。假定 CPU 要把数据寄存器 DR 中的内容 00100110 即 26H 写入存储器 08H 单元，则：(1) CPU 的地址寄存器 AR 先把地址 08H 放到地址总线上，经地址译码器选中 08H 单元；(2) CPU 把数据寄存器中的内容 26H 放到数据总线上；(3) CPU 向存储器发送“写”控制信号，在该信号的控制下，将内容 26H 写入被寻址的 08H 单元。

应当注意，写入操作将破坏该单元原存的内容，即由新内容 26H 代替了原存内容，原存内容将被清除。

上述类型的存储器称为随机存取存储器 RAM (Random Access Memory)。所谓“随机存取”即所有存储单元均可随时被访问，既可以读出也可以写入信息。

§1.5 微机工作过程

微机的工作过程就是执行程序的过程，而程序由指令序列组成，因此，执行程序的过程，就是执行指令序列的过程，即逐条地执行指令；由于执行每一条指令，都包括取指令与执行指令两个基本阶段，所以，微机的工作过程，也就是不断地取指令和执行指令的过程。微机执行程序过程示意图如图 1.8 所示。

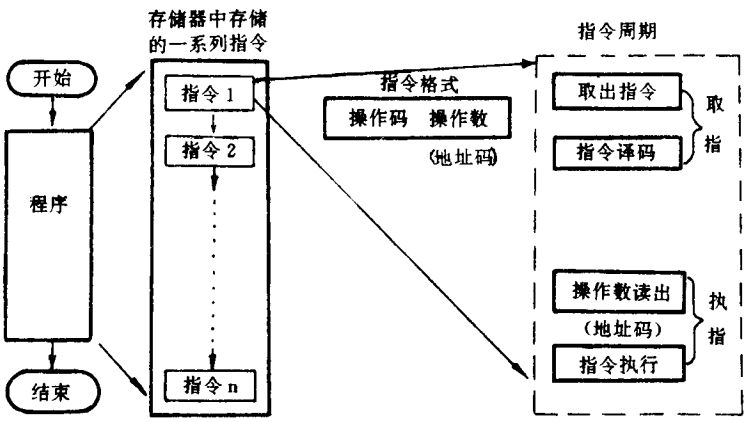

图 1.8 程序执行过程示意图

假定程序已由输入设备存放到内存中。当计算机要从停机状态进入运行状态时，首先应把第一条指令所在的地址赋给程序计数器 PC，然后机器就进入取指阶段。在取指阶段，

CPU 从内存中读出的内容必为指令，于是，数据寄存器 DR 便把它送至指令寄存器 IR；然后由指令译码器译码，控制器就发出相应的控制信号，CPU 便知道该条指令要执行什么操作。在取指阶段结束后，机器就进入执指阶段，这时，CPU 执行指令所规定的具体操作。当一条指令执行完毕以后，就转入了下一条指令的取指阶段。这样周而复始地循环一直进行到程序中遇到暂停指令时方才结束。

取指阶段都是由一系列相同的操作组成的，所以，取指阶段的时间总是相同的，它称为公操作。而执指阶段将由不同的事件顺序组成，它取决于被执行指令的类型，因此，执指阶段的时间从一条指令到下一条指令变化相当大。

应当指出的是，指令通常包括操作码（Operation Code）和操作数（Operand）两大部分。操作码表示计算机执行什么具体操作，而操作数表示参加操作的数的本身或操作数所在的地址，也称之为地址码。在 8 位机中，由于一个存储单元只能存放一个字节，而指令根据其所含内容不同而有单字节、双字节、三字节乃至最多四字节之分，因此，在执行一条指令时，就可能要处理 1—4 个不等字节数目的代码信息，包括操作码、操作数或操作数的地址。

为了进一步说明微机的工作过程，我们来具体讨论一个模型机怎样执行一段简单的程序。例如，计算机如何具体计算 3+2=？虽然这是一个相当简单的加法运算，但是，计算机却无法理解。人们必须要先编写一段程序，以计算机能够理解的语言告诉它如何一步一步地去做，直到每一个细节都详尽无误，计算机才能正确地理解与执行。

在编写程序之前，必须首先查阅所使用的微处理器的指令表（或指令系统），它是某种微处理器所能执行的全部操作命令汇总。不同系列的微处理器各自具有不同的指令表。假定查到模型机的指令表中可以用三条指令求解这个问题。表 1.1 示出了这三条指令及其说明。

表中第一列为指令的名称。编写程序时，写指令的全名是不方便的，因此，人们给每条指令规定了一个缩写词，或称作助记符。第二列即助记符。第三列为机器码，机器码用二进制和十六进制两种形式表示，计算机和程序员用它来表示指令。最后一列，确切地说明了执行一条指令时所完成的具体操作。

表 1.1　模型机指令表之一

名　称	助 记 符	机 器 码		说　　明
立即数取入累加器	LD A，n	00111110 n	3E n	这是一条双字节指令，把指令第二字节的立即数 n 取入累加器 A 中。
加立即数	ADD A，n	11000110 n	C6 n	这是一条双字节指令，把指令第二字节的立即数 n 与 A 中的内容相加，结果暂存 A。
暂　停	HALT	01110110	76	停止所有操作

现在我们来编写 3+2=？的程序。根据指令表提供的指令，用助记符形式和十进制数表示的加法运算的程序可表达为：

```
LD A，3
ADD A，2
HALT
```

但是，模型机却并不认识助记符和十进制数，而只认识用二进制数表示的操作码和操作数。因此，必须按二进制数的形式来写程序，即用对应的操作码代替每个助记符，用相应的二进制数代替每个十进制数。

```
LD A，3  变成    0011 1110；操作码（LD A，n）
                 0000 0011；操作数（3）
ADD A，2 变成    1100 0110；操作码（ADD A，n）
                 0000 0010；操作数（2）
HALT     变成    0111 0110；操作码（HALT）
```

注意，整个程序是 3 条指令 5 个字节。由于微处理器和存储器均用 8 位字或一个字节存放与处理信息，因此，当把这段程序存入存储器时，共需要占 5 个存储单元。假设我们把它存放在存储器的最前面 5 个单元里，则该程序将占有从 00H 至 04H 这 5 个单元，如图 1.9 所示。

地址 十六进制	地址 二进制	指令的内容	助记符内容
00	0000 0000	0011 1110	LD A，n
01	0000 0001	0000 0011	03
02	0000 0010	1100 0110	ADD A，n
03	0000 0011	0000 0010	02
04	0000 0100	0111 0110	HALT
⋮	⋮	⋮	
FF	1111 1111		

图 1.9　存储器中的指令

还要指出：每个单元具有两组和它有关的 8 位二进制数，其中方框左边的一组是它的地址，框内的一组是它的内容，切不可将两组数的含义相混淆。地址是固定的，在一台微机造好以后，它的地址也就确定了；而表示的内容则可以随时由于存入新的内容而改变。

当程序存入存储器以后，我们来进一步讨论微机内部执行程序的具体操作过程。

开始执行程序时，必须先给程序计数器 PC 赋以第一条指令的首地址 00H，然后就进入第一条指令的取指阶段，其具体操作过程如图 1.10 所示。

①把 PC 的内容 00H 送到地址寄存器 AR。

②一旦 PC 的内容可靠地送入 AR 后，PC 自动加 1，即由 00H 变为 01H。注意，此时 AR 的内容并没有变化。

③把地址寄存器 AR 的内容 00H 放在地址总线上，并送至存储器，经地址译码器译码，选中相应的 00H 单元。

④CPU 发出读命令。

⑤在读命令控制下，把所选中的 00H 单元中的内容即第一条指令的操作码 3EH 读到数据总线 DB 上。

⑥把读出的内容 3EH 经数据总线送到数据寄存器 DR。

⑦在取指阶段的最后一步是指令译码。因为取出的是指令的操作码，故数据寄存器 DR 把它送到指令寄存器 IR，然后再送到指令译码器 ID，经过译码，CPU“识别”出这个操作码就是 LD A，n 指令，于是，它“通知”控制器发出执行这条指令的各种控制命令。这就完成了第一条指令的取指阶段。

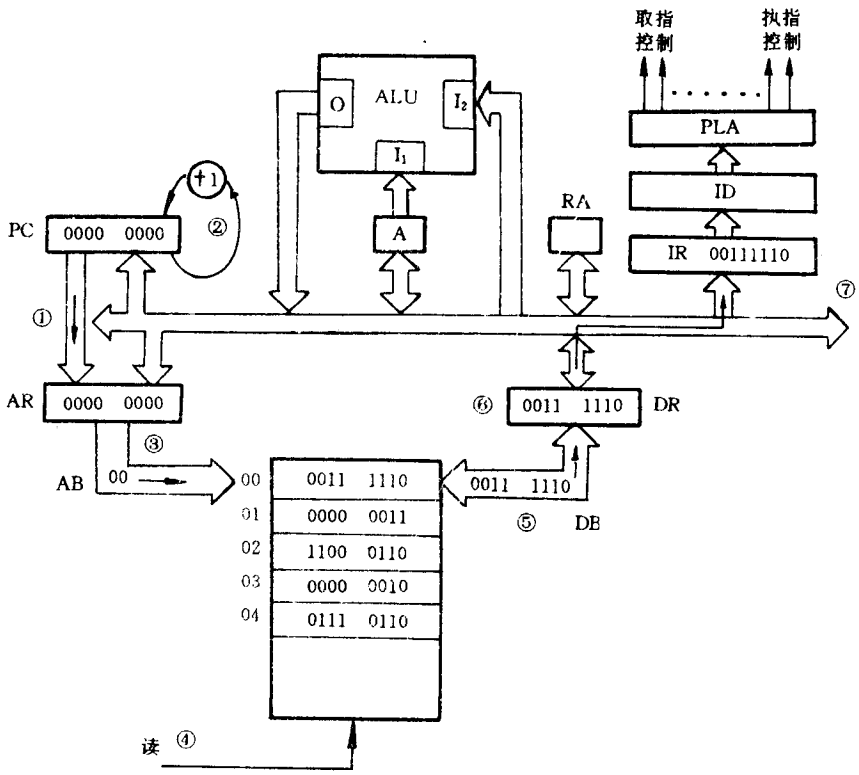

图 1.10 取第一条指令的操作示意图

然后转入执行第一条指令的阶段。经过对操作码 3EH 译码后，CPU 就“知道”这是一条把下一单元中的操作数取入累加器 A 的双字节指令，所以，执行第一条指令就必须把指令第二字节中的操作数取出来。

取指令第二字节的过程如图 1.11 所示。

①把 PC 的内容 01H 送到地址寄存器 AR。

②当 PC 的内容可靠地送到 AR 后，PC 自动加 1，变为 02H。但这时 AR 中的内容 01H 并未变化。

③地址寄存器通过地址总线把地址 01H 送到存储器的地址译码器，经过译码选中相应的 01H 单元。

④CPU 发出读命令。

⑤在读命令控制下，将选中的 01H 单元的内容 03H 读到数据总线 DB 上。

⑥通过 DB 把读出的内容送到数据寄存器 DR。

⑦因 CPU 根据该条指令具有的字节数已知这时读出的是操作数，且指令要求把它送

到累加器 A，故由数据寄存器 DR 取出的内容就通过内部数据总线送到累加器 A。于是，第一次执指阶段完毕，数 03H 被取入累加器 A 中；并进入第二条指令的取指阶段。

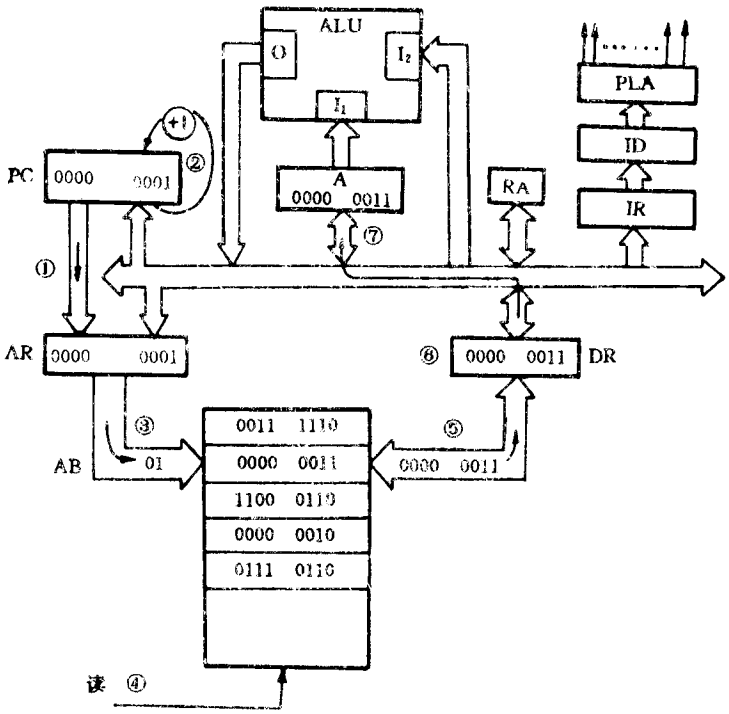

图 1.11 取立即数的操作示意图

取第二条指令的过程如图 1.12 所示。它与取第一条指令的过程相同，只是在取指阶段的最后一步，读出的指令操作码 C6H 由 DR 把它送到指令寄存器，经过译码发出相应的控制信息。当指令译码器 ID 对指令译码后，CPU 就“知道”操作码 C6H 表示一条加法指令，意即以累加器 A 中的内容作为一个操作数，另一个操作数在指令的第二字节中；执行第二条指令，必须取出指令的第二字节。

取第二字节及执行指令的过程如图 1.13 所示。

①把 PC 的内容 03H 送到地址寄存器 AR。

②当把 PC 的内容可靠地送到 AR 后，PC 自动加 1。

③AR 通过地址总线把地址号 03H 送到地址译码器，经过译码，选中相应的 03H 单元。

④CPU 发出读命令。

⑤在读命令控制下，把选中的 03H 单元中的内容即数 02H 读至数据总线上。

⑥数据通过数据总线送到数据寄存器 DR。

⑦因在对指令译码时，CPU 已知读出的数据 02H 为操作数，且要将它与已暂存于 A 中的内容 03H 相加，故数据由 DR 通过内部数据总线送至 ALU 的另一输入端 I_2。

⑧A 中的内容送 ALU 的输入端 I_1，且执行加法操作。

⑨把相加的结果 05H 由 ALU 的输出端又送到累加器 A 中。

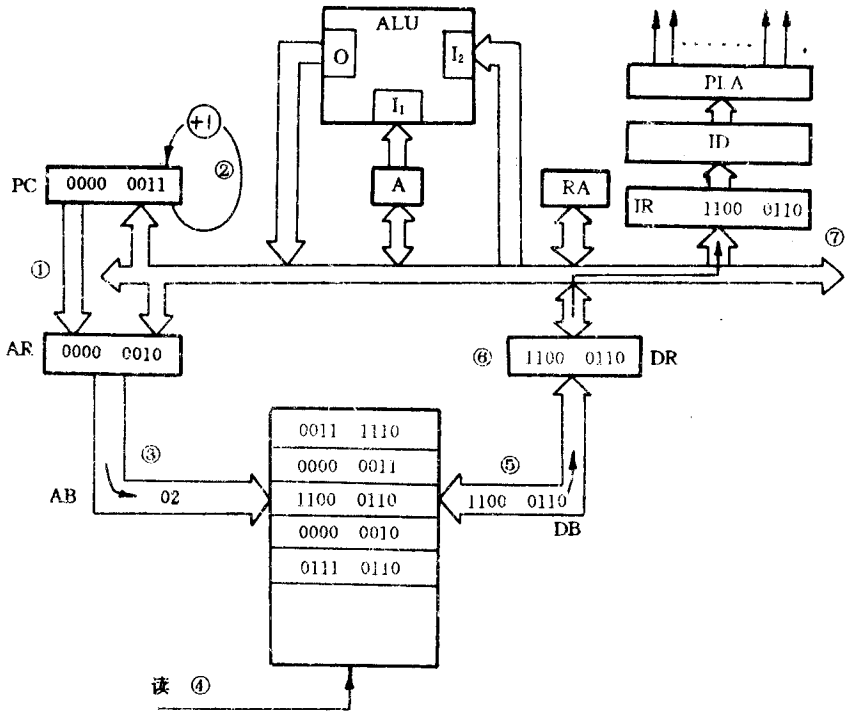

图 1.12 取第二条指令的操作示意图

至此，第二条指令的执行阶段结束；A 中存入和数 05H，而将原有内容 03H 冲掉；于是，就转入第三条指令的取指阶段。

程序中的最后一条指令是 HALT。可用类似上面的取指过程把它取出。当把 HALT 指令的操作码 76H 取入数据寄存器 DR，因是取指阶段，故 CPU 将操作码 76H 送指令寄存器 IR，再送指令译码器 ID；经译码，CPU "已知" 是暂停指令，于是，控制器停止产生各种控制命令，使计算机停止全部操作。这时，程序已完成 3+2 的运算，并且和数 5 已放在累加器中。

从上面所讨论的程序中，我们看出它使用了两种不同字节类型的指令，一种是单字节指令，如 HALT 指令，它只有一个字节的操作码而不需要操作数；另一种是双字节指令，如 LD A，n 与 ADD A，n 指令，其第一字节为操作码，而第二字节为操作数，并且，操作数的地址就紧跟着指令操作码的地址，只要在某个地址中取出操作码，则在下一个地址中就立即能取出操作数。这是一种简单的情况，我们将这种可以立即确定操作数地址的方式，称作立即寻址。

但是，在一般情况下，操作数是存放在存储器的某一单元中，而这个单元需要根据不同情况用一定方式去寻址。只有找到某操作数的存放地址，才能从中取出该操作数。这种按不同方式寻找操作数地址的方法称为寻址方式。各种微处理器的寻址方式不同，少则几种，多同十几种；寻址方式越多，功能就越强。由于执行程序的大量操作是取操作数，而取操作数又必须要寻址，因此，寻址方式对编写与执行程序的质量十分重要。

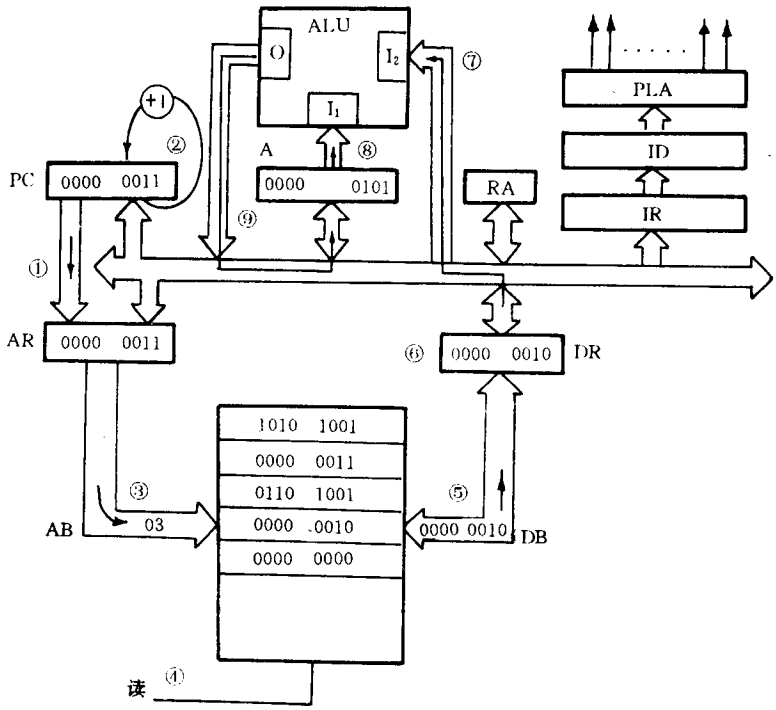

图 1.13 执行第二条指令操作示意图

在这里，我们暂时不去着重讨论各种不同的寻址方式，而只是用上面举过的简单例子来说明不同寻址方式对程序执行过程有不同的影响。例如，我们用不同的寻址方式来讨论3+2=？的编程及其操作过程。

在表 1.2 中列出了两种不同类型的寻址方式。首先是取入累加器指令：LD A，(n)。仔细阅读表中的说明并注意这条指令和前面讨论过的立即寻址指令 LD A，n 之间的区别。例如，在立即寻址方式中，指令 LD A，17H 的意思是把 17H 这个操作数（叫立即数）取入累加器。但在直接寻址方式中，指令 LD A，(17H) 的意思是把存储器 17H 单元内的某操作数（例如 03H ）取入累加器。

表 1.2 模型机指令表之二

名　称	助记符	操作码		说　明
取入累加器	LD A，(n)	00111010 n	3A n	将指令第二字节给出的单元地址中的内容取入累加器：A←(n)
加　法	ADD A，n	11000110 n	C6 n	将指令第二字节给出的内容和累加器的现有内容相加，其结果放在累加器中：A←A+n
累加器存数	LD (n)，A	00110010 n	32 n	将累加器中的内容送存至指令第二字节所给出的存储单元中：(n) ←A

从表中可以看出，同一条指令由于寻址方式不同，其指令的操作码就不同。所以，操作码不仅要告诉CPU执行操作的性质，还要告诉它是什么寻址方式。

第二条加法指令ADD A，n与前述的加法指令相同，也是把立即数n和累加器的内容相加。最后一条指令是累加器存数指令。在直接寻址方式中，例如：LD（18H），A意思是将累加器的内容存入存储器18H单元中。

现在我们用两种寻址方式指令来确定两个数（例如3和2）相加的一段程序，并将相加的结果存入存储器。

表1.3表示程序存放在存储器中的情况。假设，我们已经把程序存入任意指定的初始地址为00010000（10H）的存储单元中。地址10H和11H放第一条指令：

LD A，(17H)

这条指令告诉CPU把存储单元17H的内容取入累加器。单元17H中的内容是操作数03H。于是，第一条指令把03H取入累加器。

第二条指令放在存储单元12H和13H中，它是：

ADD A，02H

这条指令告诉CPU把单元13H中的立即数02H与累加器中的数03H相加。最后，把相加的和05H放在累加器中。

第三条指令放在存储单元14H和15H中，它是：

LD（18H），A

这条指令告诉CPU把累加器的内容存入存储单元18H。执行该指令后，和数05H将暂存于单元18H中。

表1.3　采用立即与直接寻址的程序实例

地　址	内　容	助记符/内容	
10H　0001　0000	0011　1010	LD A,(n)	第一条指令
11H　0001　0001	0001　0111	17H	
12H　0001　0010	1100　0110	ADD A,n	第二条指令
13H　0001　0011	0000　0010	02H	
14H　0001　0100	0011　0010	LD (n),A	第三条指令
15H　0001　0101	0001　1000	18H	
16H　0001　0110	0111　0110	HALT	第四条指令
17H　0001　0111	0000　0011	03H	数　据
18H　0001　1000	0000　0101	05H	保存和
19H　0001　1001			

最后一条是使CPU停机的指令。HALT指令在程序中的作用是使CPU强迫中断程序。如果没有HALT指令，CPU就无法知道指令于何处停止，数据于何处开始存放，并且，CPU将把下一字节误当成操作码来执行，从而产生错误。

下面我们来看这个程序的执行过程。

首先，把第一条指令的地址10H赋于PC，然后就进入第一条指令的取指阶段。其操作过程与上述的类似，如图1.14所示。

当10H单元中第一条指令操作码3AH由数据寄存器DR送到指令寄存器IR，再经指