系统软件与软件工程技术基础

林国璋　张雪兰　编著

北京理工大学出版社

系统软件与软件工程技术基础

林国璋　张雪兰　编著

北京理工大学出版社

内 容 简 介

本书根据美国 IEEE'83 教程大纲并结合我国具体教学实践编写，作为计算机专业大学本科的核心课程《系统软件与软件工程技术基础》的基本教材。该教材以 MASM 宏汇编语言为主设计语言，以当今广为流行的 IBM 微型计算机系统及其 DOS 软件环境为主，深入地讨论了系统软件及其软件开发工程的基本原理、基本概念、基本方法和基本技巧。内容包括：程序设计师的硬件资源与软件资源、基本指令集、程序的控制和组织、汇编语言与汇编器、算术运算与代码转换、模块程序设计、模块的连接与装入、软件开发工程引论、系统软件设计技术、I/O 程序设计与基本输入输出系统等。本书由浅入深、循序渐进，既有系统的理论与概念，又有大量的实例与习题，内容丰富，图文并茂。

本书既可作为大专院校计算机及其应用专业 80～100 学时的专业教材，也可作为从事计算机开发及其研究的工程技术人员的参考书。

系统软件与软件工程技术基础

林国璋　张雪兰　编著

*

北京理工大学出版社出版

新华书店北京发行所发行　各地新华书店经售

北京理工大学出版社印刷厂印刷

*

850×1168 毫米　32 开本　19.125 印张　494 千字

1990 年 9 月第一版　1990 年 9 月第一次印刷

ISBN 7－81013－351－9/TP·25

印数：1－5000册　　定价：4.70元

序 言

《系统软件与软件工程》是美国 IEEE'83教程中计算机科学与工程系的一门核心课程。本书根据该课程的大纲要求并结合我国具体的教学实践进行编写，取名《系统软件与软件工程技术基础》，作为计算机专业本科生的教材。

按照83教程大纲要求，本课程的任务在于使学生在完成基础课、计算机概论课和高级语言程序设计学习的基础上，深入地学习计算机系统软件的设计基础。根据大纲要求，这种学习必须以汇编语言为工具，以便使学生在硬件与软件的结合上熟悉系统软件设计的基本概念、基本方法和基本技巧。

汇编语言是面向机器指令的语言，学习汇编语言离不开实际的计算机系统。因此选择一个典型的计算机系统，不仅对于组织教材是重要的，而且对于理论联系实际地进行教学也是十分重要的。本书选择了当今广为流行的以8086、8088或80286等 CPU 组成的微型计算机系统为典型系统，讨论了系统软件设计的环境、指令处理机的概念、基本指令集的分类与功能、程序的逻辑结构与实现方法、汇编语言的语法、语义以及使用汇编语言进行程序设计的基本方法和基本技巧。同时讨论了汇编语言的模块程序设计功能以及模块程序设计的基本概念和基本方法。对汇编语言的学习不是停留在汇编语言的程序设计上，而是深入学习汇编器的一般原理、连接程序的要求和连接过程、装入模块的结构、装入程序的工作原理以及重定位方法等内容，以便学生能深入了解系统程序的内部工作过程和设计原理，这对于提高学生系统软件的设计能力无疑是十分重要的。

为了培养、锻炼和提高学生在系统软件设计中分析问题和解决问题的能力，本书第八章比较系统地讨论了系统软件设计的技术技巧，其中包括表格处理技术、清单处理技术、链结构处理技术、

队列处理技术、堆栈处理技术、公用数据区处理技术、再入码设计技术等系统软件设计中经常要用到的基本技术。书中给出了大量的程序实例，以引导学生扎扎实实地从实践出发学习理论知识、掌握基本技能。

输入输出程序设计是系统软件设计的一项重要内容。本书最后一章讨论了输入输出程序设计的基本方法和基本技巧，其中包括查询方法的I/O程序设计、中断控制的I/O程序设计和直接存贮器存取(DMA)程序设计。进而讨论了如何为一个计算机系统构造和设计一个基本输入输出系统。在IBM微型计算机中，这种基本输入输出系统称为ROMBIOS。

本书第七章是软件开发工程引论。讨论了软件开发的一般过程、系统分析与问题定义的一般原则、系统设计的常用方法、程序编码与特定的开发环境、查错与测试技术以及文件编制的要求等。在系统软件设计技术和输入输出程序设计技术之前安排这一章内容，目的是使学生在总体上明确软件开发的步骤、要求和方法，培养学生在软件工程开发上的科学作风和严格精神。

本书在编写时企图增加屏幕处理技术、磁盘处理技术、文件存取技术以及DOS的核心修改技术等。但因篇幅所限，这些内容未能编入书中。

本书第二章、第三章、第五章和附录由张雪兰同志执笔，其余各章由林国璋同志执笔，并由林国璋同志负责全书的组织与定稿。

本书在编写过程中受到了北京理工大学计算机系领导和许多同志的热情帮助，也受到了北京航空航天大学计算机系领导的大力支持，并委派了李雅倩副教授对本书原稿进行审查。提出了许多宝贵的意见。在此，仅向为本书出版给予支持和帮助的所有领导和同志表示衷心的感谢！

由于编者水平所限，加上时间仓促，书中难免仍有错误和不妥之处，欢迎读者批评指正。

编者　1989.8

目　录

第一章　程序设计师的硬件资源与软件资源

本章讨论程序设计师的硬件资源与软件资源。程序设计师尤其是面向机器语言和汇编语言的程序设计师必须了解计算机系统向你提供的硬件资源和软件资源，包括它们的结构、功能及其使用方法。只有深刻地了解计算机系统提供的这些资源——程序设计舞台，才有可能设计出能充分发挥计算机内在潜力的高性能的软件系统，导演出各种精采的剧目。使用高级语言的程序设计师可以不关心或较少关心计算机的硬件资源和软件资源，但是利用高级语言编写的程序很难充分发挥和灵活调动计算机系统的内在潜力。因此程序运行效率比起用机器语言或汇编语言编写的程序要低得多。再者，任何一个用高级语言编写的程序总要翻译成机器语言，才能最终提交运行。因此如果要从事系统软件开发或编译系统开发，或者要进行用高级语言难于胜任的某些应用开发，就必须了解计算机的硬件资源，才能利用汇编语言甚至机器语言完成软件开发任务。

§1.1　硬件资源概述

计算机开始是作为一种计算工具而出现的。随着计算机研究和应用的不断发展和深入，计算机除了用于完成各种数值处理外，更多的更广泛的应用是各种非数值处理和控制系统。但是不管它们用于何种领域，我们都可以用图1—1和图1—2来说明计算机硬件的基本结构。

图1—1示出了一种较小的系统结构。这种结构是单处理机系统的典型结构。系统中有一个且只有一个指令处理机。这个指令处理机通常称为中央处理部件(CPU)。除了 CPU 之外，系统还必须有一个主存贮部件，这个主存贮部件通常也称为内存贮器，简称为主存或内存。当前要运行的程序和数据就在主存中。除了主存之外，计算机还包含一个或若干个辅助存贮部件，通常称为外存(贮器)或者二级存贮器。外存有磁盘、磁带等，一般具有较大的容量，但存取速度远比内存慢。外存用于存放待执行的程序和数据，各种程序和数据通常预先被存放在外存中。当程序需要提交运行时，相关的程序和数据被调入内存。硬件资源的另一部分就是外部设备。输入设备把程序和数据送入计算机，输出设备把处理的结果送给用户。有各种各样的输入输出设备，例如键盘输入，显示输出、打印输出等。输入输出设备通常不是直接连到 CPU 上的，而是通过输入输出接口逻辑与 CPU 进行数据交换的。

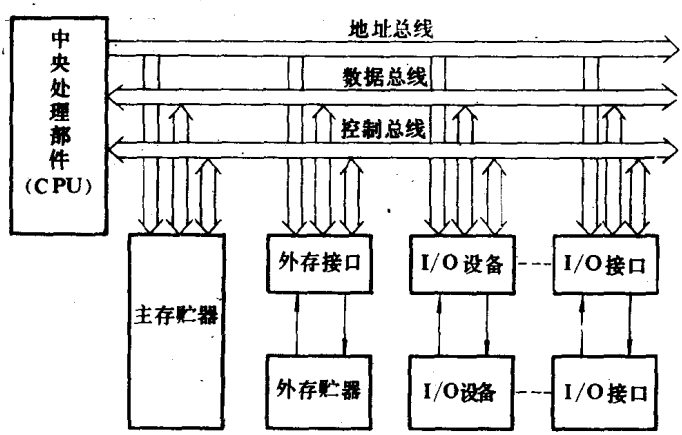

图1—1　典型的单处理机系统结构

系统中的各个部件通过一组总线进行连接。这一组总线包括地址总线、数据总线和控制总线。CPU 通过地址总线指明它所要存取的指令或数据的地址，包括主存地址、外存地址或 I/O 设备

地址，CPU 通过数据总线与这些资源进行数据交换，把数据送到存贮器或外部设备的某一地址中，或从存贮器或外部设备的某一地址中获取数据。控制总线用于控制各个部件协调工作。

图1—2示出了一种较大的系统结构。

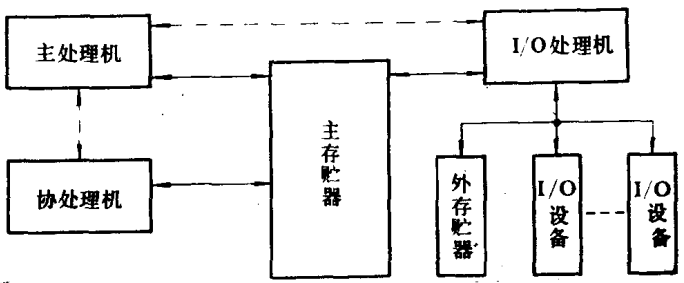

图1—2　多处理机系统结构

在这种系统中，有多个处理机。每个处理机之间可以互相通讯，每个处理机也都可以与主存贮器进行通讯。这些处理机往往具有明确的分工。例如图1—2中有三个处理机，其中一个是主处理机，另一个是协处理机。协处理机可能是主处理机的功能支持部件，例如主处理机可能不具备浮点运算逻辑，这时协处理机可以是一个浮点处理机。系统中还可以有一个专门负责输入输出的 I/O 处理机。象这种由多个处理机构成的系统往往具有很高的数据处理和吞吐能力。

无论何种结构的计算机系统，处理机（主处理机或其它处理机）通常也称为指令集处理机。因为所有的处理机都是通过执行特定的指令序列来完成特定的工作的。处理机所能解释执行的所有指令称为这种处理机的指令集或指令系统。指令系统中的每一条指令表示了该处理机的固有功能或称为硬件功能。

指令是计算机编程的最小单位。在使用机器语言或汇编语言进行程序设计时，每一条指令都可以是程序中的一条基本语句。可见了解、熟悉一台计算机的指令系统对于在该计算机上进行程序

设计、软件开发是多么重要。

计算机的指令通常可以用下面的形式来说明：

操作码	地址码

其中操作码用于指明该指令所要完成的基本操作，如数据传送、二进制加法、二进制移位、程序转移等。而地址码部分用于指明基本操作所涉及到的操作数及其结果的存放地址。地址码部分可以指定一个地址，也可以指定两个甚至三个地址，依不同的机器、不同的指令而定。例如把一个二进制操作数逻辑求反的指令只需要指定一个操作数地址；而数据传送指令则要求指定两个操作数地址；一个称为源操作数地址，另一个称为目标操作数地址，指令从源操作数地址中取出数据并送到目标操作数地址中。算术加法指令要求对两个加数进行相加并产生一个和数，因此它可以要求指明三个地址。但大多数的微小型计算机对于这种算术运算指令只要指明两个地址就可以了。例如对于加法指令，它完成：

源操作数＋目标操作数⇒目标操作数

的操作，即把源地址中的内容加上目标地址的内容，结果送回到目标地址。通常我们把这样的目标地址称为累加和地址。显然，两地址的算术运算指令改变了目标地址中原来的内容，而三地址的加法指令可以完成：

第一个源操作数＋第二个源操作数⇒目标操作数

的操作。

指令的地址码通常并不一定是直接给出操作数的真正地址，而是给出寻找操作数地址的方法，这称为指令的寻址方式。指令的寻址方式对于程序设计师来说至关重要。因为任何一个程序都要处理数据。数据在什么地方呢？程序设计员如何组织自己的数据呢？程序如何能灵活高效地处理数据呢？所有这些问题无疑都是程序设计员十分关心的问题。

计算机最终只能理解由“0”和“1”组成的二进制代码，因此指令也是由二进制代码组成的。一台计算机通常都有上百条甚至几

百条指令。如果程序设计就用这些二进制代码组成的机器语言来编写，那将是十分麻烦的。虽然如此，早期的计算机程序设计却只能这样进行。取代机器语言的首先是汇编语言。汇编语言很接近机器语言。汇编语言编程的基本单位仍然是处理机的指令。但是每一条指令都可以用一些具有明确意义的符号来代表，包括指令操作数的地址也可以由符号来表示。例如我们可以用下面的语句来代表数据传送指令：

```
MOV     dest,src
```

这里 MOV 表示数据传送操作，dest 可以是一个有意义的符号或简单的表达式。它指明数据传送的目标地址，而 src 也可以是一个有意义的符号或简单的表达式，它指明被传送的数据的源地址。又如：

```
ADD     dest,src
```

是一条表示二进制的加法的汇编语句，它是一条二进制加法指令。ADD 表示二进制加法操作，而 dest 和 src 同样可以是一些有意义的符号或表达式，它们分别表示被加数或者和数的地址和加数的地址。再如：

```
JMP     label
```

是一条无条件转移指令，其中 JMP 表示转移，而 label 可以是一个有意义的符号，它指明转移的目标地址。

虽然汇编语言可以用符号来表示操作数地址，但是它不同于任何一种高级语言。在高级语言中，程序设计员可以定义数据变量或语句标号，但这些变量或标号完全不依赖于实际机器的硬件资源。例如把三个整数相加并且打印它的和，你可以用 C 语言写出如下的程序：

```
MAIN(  )/* sum.c,the sum of a,b,c */
{
  INT   a,b,c,sum;
  a=1;
```

```
    b=2;
    c=3;
    sum=a+b+c;
    PRINTF("sum  is  %d\n",sum);
}
```

程序首先说明 a,b,c 和 sum 是四个整形变量,然后把1、2和3三个整数分别赋给变量 a、b、c,接着利用一个赋值语句 sum=a+b+c 求出这三个数的和并把和赋给 sum 变量,最后用 PRINTF 语句输出 sum 值。这里变量可以通过类型说明来定义,又可以通过赋值语句来赋初值,还可以通过表达式来计算要求完成指定的操作,最后还可以通过格式打印语句要求按指定的格式输出指定变量的当前值。这样,可以完全不知道计算机的硬件资源而写出正确的程序。但是使用汇编语言时,问题就要复杂些。由于汇编语言编程的基本语句是指令,而指令又直接涉及到硬件资源。例如上面的例子中,如何定义数据类型,数据在计算机中应当如何表示和存放,如何通过指令完成变量的赋值,又如何通过指令完成变量的运算和输出等等,都必须了解计算机的硬件资源以及指令如何控制这些硬件资源以完成各自的功能之后,才能够完成。

§1.2 处理机资源

处理机资源是计算机系统中最基本的硬件资源。程序设计员需要了解的处理机资源包括:

(1)可同时处理的数据位数

(2)程序可访问的数据寄存器的数量、名称及其作用

(3)处理机提供哪些手段用于指定内存地址

(4)处理机提供哪些指令用于实现或方便程序设计

(5)是否具备较强功能指令的处理部件与逻辑如乘除法指令、块操作指令、浮点运算指令等等。

(6)反映处理机程序状态及其控制的能力

详细地讨论各种计算机的内部结构及其逻辑并非本课程的任务，我们的任务在于从程序设计的观点来了解指令处理机的结构。因此我们将从以下几个方面来介绍处理机资源。

1.2.1 运算能力

运算能力就是通常说的处理机的字长和算术逻辑运算功能。处理机的运算能力主要由其算术逻辑运算部件决定。算术逻辑运算部件用于完成二进制的算术逻辑运算操作。运算的数据总是二进制数据或者被化为二进制数据。运算的结果仍然是二进制的数据。算术逻辑运算部件的运算器的位数决定了它一次可处理的二进制数的长度。

除了运算器的位数之外，它的运算能力通常还包括它所能处理的算术逻辑操作的种类，或者说它所能执行的算术逻辑运算指令的种类。

反映运算器运算能力的另一指标是它的运算速度。运算速度除了与机器主频直接有关外，一般与运算器本身的逻辑结构有着密切的关系。

1.2.2 程序可使用的寄存器

指令处理机内部往往都包含有为数不同的一些寄存器，这些寄存器对于程序设计师来说十分重要。因为这些寄存器不仅是指令处理机与外部(例如存贮器或 I/O 设备)交换数据的主要通道，而且这些寄存器在程序设计中往往有特定的功能。例如有些寄存器被指定为累加器，有些寄存器被指定为计数器，还有些寄存器被指定为内存指针或内存基地址等等。程序设计师在编码过程中要频繁地使用这些寄存器，它们对于程序设计来说是十分宝贵的资

源。这些寄存器往往都有指定的名字。例如，假定一个指令处理机包括有四个通用寄存器 AX、BX、CX 和 DX，那么上面计算 sum＝a＋b＋c 的问题可以用下面的指令序列来实现：

```
MOV   AX,a   ;把 a 的值赋给 AX
MOV   BX,b   ;把 b 的值赋给 BX
MOV   DX,c   ;把 c 的值赋给 DX
ADD   AX,BX  ;AX＝a＋b
ADD   AX,DX  ;AX＝a＋b＋c
```

于是 AX 中将保留有这三个数的和。

1.2.3 程序可使用的处理机状态字 PSW

处理机状态字用于反映处理机的当前状态。处理机状态字包括处理机的当前状态以及一条指令执行以后的结果特征。处理机的当前状态可能包括处理机当前是否允许中断，是否处于连续运行状态等。而一条指令执行后的结果特征可能包括当前结果是否为零，是否为负，是否产生进位或借位以及是否溢出等。这些结果特征常常构成程序分支的条件。例如求 a，b 中的较大者，用高级语言如 C 语言写出来是：

```
IF(a>b)
  Z=a;
ELSE
  Z=b;
```

这时无需关心计算机将如何判断 a 是否大于 b。但是当用机器语言或汇编语言来写这个程序时，首先必须知道如何确定 a 与 b 的大小。为了比较 a 与 b 的大小，计算机必须做一次比较(相减)，并把比较的结果登记在处理机状态字的有关标志位中，然后程序才可以检查这些标志位以确定它们的大小关系。

用于反映当前指令执行结果特征的标志常常包括进位标志、

结果为零标志、符号标志、溢出标志、奇偶标志等。例如：如果 a 和 b 都是无符号数据，则上述比较 a、b 中的较大者，用汇编语言也许可以用下面的指令序列来实现：

```
MOV     AX,a      ;把 a 赋给 AX
MOV     BX,b      ;把 b 赋给 BX
CMP     AX,BX     ; AX－BX⇒AX，影响结果标志
JNC     DONE      ;若无借位则转到结束，AX＝a 为较大者
MOV     AX,BX     ;若有借位，把 BX⇒AX，AX＝b 为较大者
DONE:HLT
```

这里假定 a 与 b 是无符号数。两个无符号数比较大小，我们可以把它们作一次减法，然后根据相减时是否产生借位来决定大小。若产生借位，说明 a＜b，否则 a≥b。但是，若 a 和 b 是带符号的数据，则就不能简单地用是否有借位来判断大小。这时首先要判断这两个数相减是否产生溢出，再利用结果的符号标志来判断大小。在不溢出的情况下，结果的符号位为“0”表示 a≥b，为“1”表示 a＜b。而在溢出的情况下，结果的符号位为“0”却表示 a＜b，为“1”才表示 a＞b。如果结果的零标志为“1”，则表示 a＝b。于是为了比较两个带符号数据的大小关系和是否相等，程序员就需要关心处理机的溢出标志、符号标志和零标志。

1.2.4 指令执行过程与指令周期

程序是如何一条一条地被处理机执行的呢？处理机执行一条指令需要多少时间呢？使用机器语言或汇编语言的程序员必须知道这些问题。

处理机中一般还包含两个寄存器，一个称为指令寄存器，另一个称为程序计数器(PC)或称指令指针(IP)。这两个寄存器的内容一般是由处理机的硬件逻辑自动更新的。程序设计员很少能对它们实行干预。程序是放在存贮器中的。程序功能的实现是通过逐条