海洋出版社

C语言接口与信号处理技术

林 君 编著

北京希望电脑公司计算机技术丛书

C语言接口与信号处理技术

林 君 编著
郁 辛 审校

海 洋 出 版 社
1993年

内 容 简 介

C语言应用十分广泛，涉及到许多软件开发领域，本书重点放在C语言与硬件接口以及信号处理技术，内容包括C语言接口基础、A/D转换系统与电路、信号采集与变换的数学基础、卷积、数学滤波、傅里叶分析、相关分析、卡尔曼滤波器、D/A转换与应用、信号处理的硬件、DMA高速数据采集等，书中通过具体实例对所述理论予以说明，供读者参考。

本书可作为计算机工程、电子技术、工业控制、仪器仪表等专业的工程师、科研人员、教师、高年级大学生作为参考读物，也可作为开设这门课程的教学用书。

需要本书的用户，请直接与北京8721信箱联系，邮码100080，电话2562329

(京)新登字087号

责任编辑 闫世尊

北京希望电脑公司计算机技术丛书

C语言接口与信号处理技术

林 君 编著

郝 辛 审校

*

海洋出版出版（北京市复兴门外大街1号）

海洋出版社发行 东升印刷厂印刷

开本：787×1092 1/16印张：14.6字数：330千字

1993年11月第一版 1993年11月第一次印刷

印数：1—5000

*

ISBN 7-5027-3795-3/TP·226

定价：15.00元

前　言

C语言现在已成为软件开发者们使用最多的一种语言。尽管讲解C语言及其编程技术的书已经不少，但是直接利用C语言与个人计算机接口并进行实时应用的却不多见。本书主要讨论端口、传感器接口、模拟到数字转换、卷积、数字滤波、傅里叶变换、卡尔曼滤波及数模转换，在本书的后三章还给出数字信号处理的硬件、利用C进行DMA数据实时采集以及C语言与汇编语言混合编程的实用技术。

本书没有把重点放在电路和接口的设计上，而是重点放在信号处理与实时应用上。书中所给出的实验程序均通过Microsoft C5.1编译并测试过。不可能在一本书中讨论所有的问题，但本书提供了用C语言接口与信号处理的方法和途径。

本书假定读者对C语言有较深入的了解，所以没有对C语言本身进行详细的讲解。凡是准备利用C语言进行信息采集与实时处理的读者，都可以从阅读本书中得到启发。

研究生别红霞主要参加了本书的编写工作。此外项葵葵、刘蓉、孙春晖和刘赛红也参加了本书的部分编写审校工作。北京希望电脑公司对本书的出版给予了大力的支持和帮助，在此表示衷心的感谢。

由于编者水平所限，收集和分析的资料还不够充分，书中一定存在许多不足之处，欢迎广大读者批评指正。

编　者

1993.4

目　录

第一章　C语言接口基础

C语言是一种中级编程语言，由贝尔实验室 (Bell Laboratories) 的戴尼斯·里查 (Dennis Ritchie) 设计，1972年在PDP-11上实现的。C语言的前身是B语言，B语言是1970年在PDP-7上，由肯·托姆逊 (Ken Thompson)为最初的UNIX系统而设计的。B语言又是由BCPL发展而来的 (Basic Cambridge Programming Language)，BCPL是马丁·里查德 (Martin Richard) 1967年在剑桥设计的一种系统编程语言。

由波廉·克尼汉(Brian Kernighan)和戴尼斯·里查编写的《C程序设计语言》(1978, Prentice Hall) 是权威性的版本。尽管它并没被作为国际标准，但通常被用作标准C语言。这个最初的高深的版本并不是介绍性的编程手册，它是在精通基本编程概念（如变量、赋值语句、循环和函数等）的基础上的——一旦你掌握了C语言就可能很容易阅读。随着C语言的广泛使用，出现了许多版本，其中有许多意在简化最初的克尼汉和里查版本。这些指导书各有特点，无疑，你可以选择最适合自己的书。每章的附录给出了被认为特别有参考价值的文献目录。

1.1　特点和背景

C的适应性使得它可以在个人8位计算机或Cray-1上运行，Cray-1是许多快速计算机之一。这种灵活的中级语言设计可以使程序快速简洁。C语言程序处理功能强、运算速度快，可以直接实现对系统硬件及外围接口的控制。许多情况下，为“高效率”而用汇编语言所编写的程序可以用相似的C语言程序所代替。尽管C语言是中级语言，它仍然具有通常与Pascal等高级语言相连的结构化编程特征。C语言是非常简明的，编程时可以使程序非常短小。它还有非常丰富的运算符，能理想地实现可编程输入输出设备和标志测试。

本书的目的是讲授有效的接口所需要的C语言的有关方面的知识。我们采用的办法是：随着内容深入，讲述C程序结构，同时“字节”型一定大小的块给出信息，试图使C程序更加有吸引力，更加易懂。我们尽量逐渐增加程序的复杂性，因此每一个程序都表示了C语言的一个新的特征或者给出了另外一种程序结构。只要有可能，都用程序流程图来说明程序结构，并在程序中给出大量的注释说明以助于理解。

所有给出的例程都已在IBM PC系列机上调试、运行过，用Microsoft C5.1版本编译器。本书重点在于有效的接口而不是精巧的编程技术。这里尽可能地通过讲述多变的程序结构来说明这种特殊语言的灵活性。本书鼓励读者运行这些程序，用C语言实验。不可避免地，一些程序很长，甚至有经验的程序员也会为之头痛，为了容易理解，单独地给出了其基本结构。许多程序需要改写，如果只是按照例程，你不可能学得更好，也不希望这样。

不是设计建立自己的接口电路，而是选用混合芯片技术 (Blue Chip Technology) 数据采集和控制卡。卡上的插座是端口映射式的，可以用任何语言驱动，这使我们可以集中地研究问题的编程方面，简化了接口的任务。

1.2 基本接口

将位状态送给外部世界的基本思想导致了非常高级的、用最少硬件的电子学科研项目，这主要是因为大部分问题都由有创造力的软件解决。假设一个Exocet导弹在寻找目标。在板上，计算机通过输入端接收来自导弹传感器的数据，实时地处理这些数据，将结果用于控制预期的、成功投掷的弹道。尽管这个问题很复杂，但基本的问题可以简化为：从与一输入端口相连的外围读取“0”和“1”，处理这些数据最后通过输出端口将重新排序的数据输出给外部世界。

这里的问题是：你如何找到可用的I/O空间？如何格式化控制字？如何控制I/O卡？如何处理数据？令人遗憾的是，除非你是一个熟练的汇编语言程序员，否则，这些问题会提出令人望而生畏的任务。我们目的不是为了“最佳的可能性设计”，而满足于“最佳设计的可能性”，并运用C语言设计。

1.3 可编程的输入输出设备

实际世界与个人计算机通信通过外围接口适配器（PIA）的端口或通用接口适配器（VIA）的端口。这些相对复杂的、高级的芯片可以被编程用作输入、输出设备，有效地缓冲来自受控外设的数据总线以保护系统。采用存储器映射式I/O使CPU把端口简单地作为地址的集合，这些地址与存储器的地址没有区别。如果输入输出设备已认真地设计好了，就几乎可以像常规地那样进行双向通信。事实上，这些设备的处理是模拟的，不用焊接就能够建立电路，这是因为通过设置适当的控制寄存器中所需要的位状态实现了必要的连接。

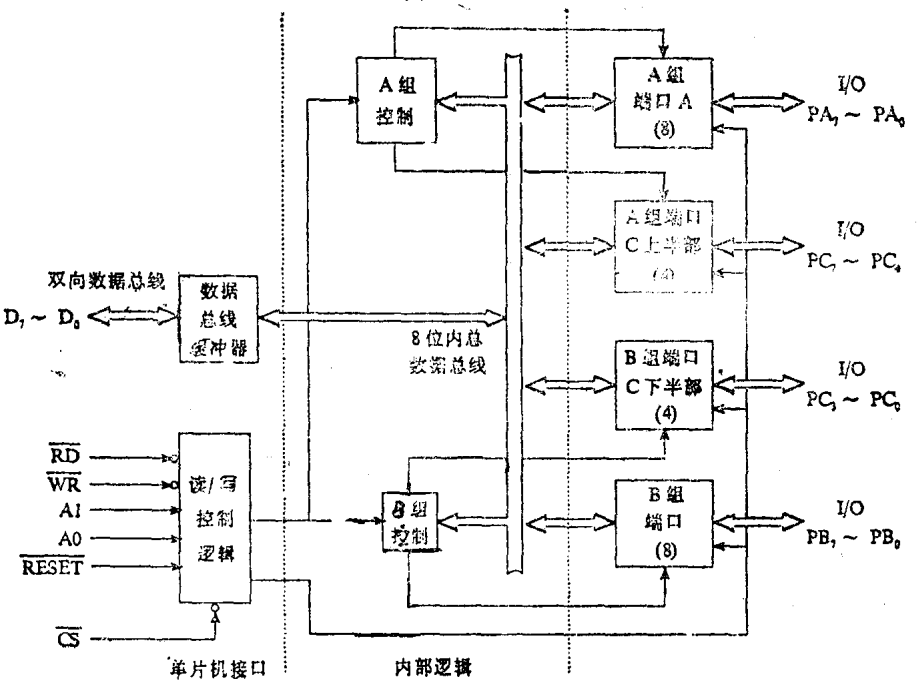
每种微处理器产品都用一种特定的可编程输入输出设备来适合它们自己的系统。熟悉微处理器指令集的同时，熟悉它的特定设备有助于用这个设备更好地为微处理器服务；并且这个设备很难改变。尽管这些设备中许多都有独特的特性，但它们还是有某些共性，这使得从相当原始的可编程外部接口PPI（如Intel 8255）能相对容易地转换为复杂的、先进的MOS工艺6522VIA。

为了清楚地解释采用的接口规约，必须讲述特定的芯片。然而，这又使本书受到时间的限制，但是其基本思想在一段时间后仍然是通用的。

1.4 8255可编程外围接口

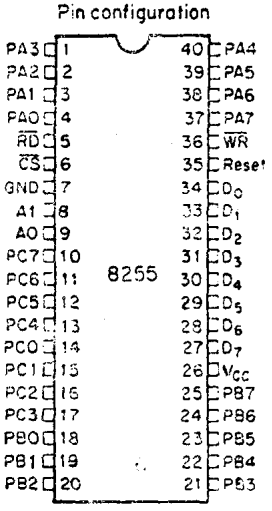
图1.1中的可编程外围芯片Intel 8255是一个相当简单的并行接口芯片。这种早期的芯片是市场上最早的接口适配器之一，最初设计用于8008系统和8080A系统中，但是现在又重新流行起来，因为它可以很容易地与IBM PC总线接口，实际是工作在最大模式的8088处理器的总线。大规模集成使并行输入／输出操作集中在一个单一的40脚的封装上。使芯片可软件编程，这提供了以后设计的灵活性——在控制寄存器中存放控制字，可以决定哪些组线是输入线，哪些是输出线。

微机与实际世界是通过24根输入输出线通信的，这24根线分为两组8根线的数据端口A



(a)

8255的管脚图



(b)

8255的管脚名

D_7—D_0	数据总线(双向)
Reset	复位
$\overline{CS}$	芯片选择
RD	读输入
$\overline{WR}$	写输入
A_0, A_1	端口地址线
PA7～PA0	端口A(位)
PB7～PB0	端口B(位)
PC7～PC0	端口C(位)
V_{cc}	+5V
GND	0V

(c)

图1.1 Intel 8255可编程端口，在本书前面部分用作编程模型。

和数据端口B，以及两个四根线组。这两个四根线组组成端口C，端口C可以作数据口或控制口，这取决于所选的工作方式。

在方式0中，端口A和端口B作为两个8位端口，而端口C用作两个4位端口。这种方式支持简单的数据传送，没有握手信号。

在方式1中，端口A和端口B可以用作输入口，也可以用作输出口，它们不能像某些其它可编程I/O设备的端口那样逐条线地分别定义，只能8位同时定义为输入或输出。端口C的6位被留作握手信号和中断控制。

方式2用端口A的8条线作双向数据传送，由端口C的高5位作为握手信号。

1.5 对8255编程

8255的编程涉及到四个8位寄存器，即端口A、端口B、端口C和一个控制寄存器。按照在可利用的I/O空间内设置的地址，寄存器模式有如图1.2所示的四个相邻的地址。

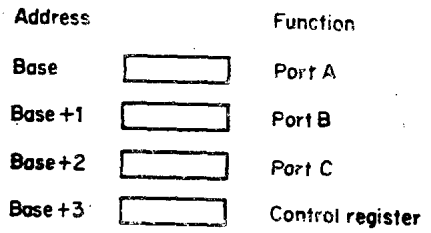


图1.2 8255编程模式

I/O端口的操作由写到控制寄存器的8位字来控制，控制寄存器的地址为（基址+3），控制的格式示于图1.3中。简单的输入输出操作，没有握手信号，需要将控制字设

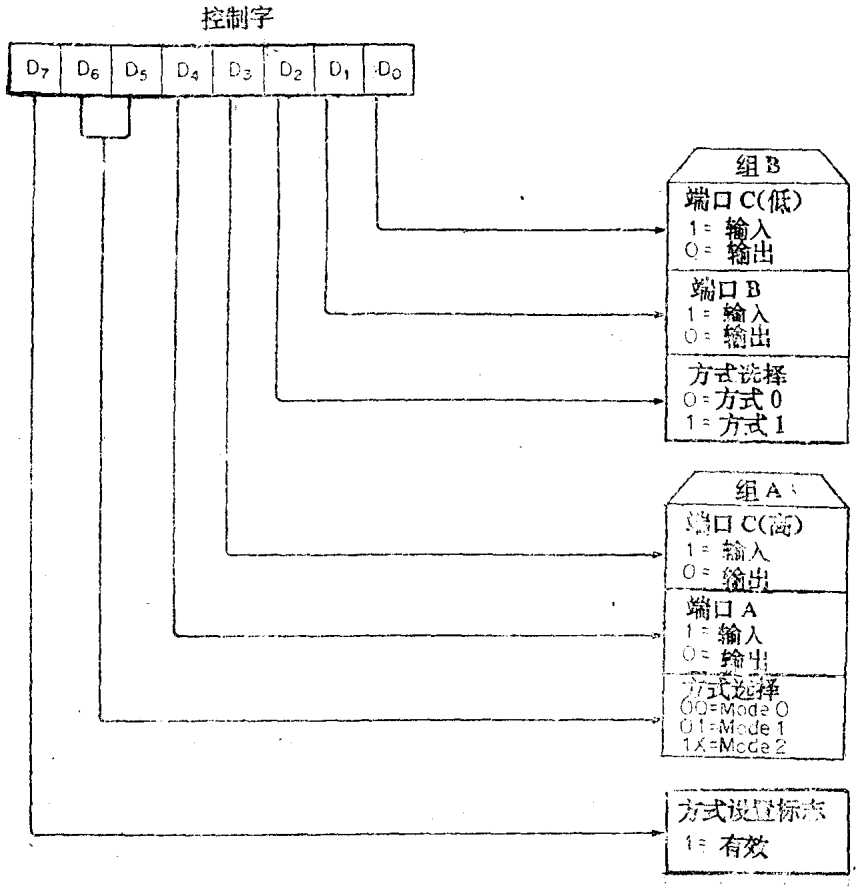


图1.3 8255控制字位功能

置为方式0。表1.1给出了方式0的端口解释图表，这个图表在考虑样倒程序时是一个有效的参考源。

用这种方式，三个端口用作简单的输入输出而不需要握手信号，数据只是从一个指定端口简单地读写。

表1.1 方式0的端口说明图表

No	Control Word Bits								Group A		Group B	
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Port A	Port C (Upper)	Port B	Port C (Lower)
0	1	0	0	0	0	0	0	0	OUTPUT	OUTPUT	OUTPUT	OUTPUT
1	1	0	0	0	0	0	0	1	OUTPUT	OUTPUT	OUTPUT	INPUT
2	1	0	0	0	0	0	1	0	OUTPUT	OUTPUT	INPUT	OUTPUT
3	1	0	0	0	0	0	1	1	OUTPUT	OUTPUT	INPUT	INPUT
4	1	0	0	0	1	0	0	0	OUTPUT	INPUT	OUTPUT	OUTPUT
5	1	0	0	0	1	0	0	1	OUTPUT	INPUT	OUTPUT	INPUT
6	1	0	0	0	1	0	1	0	OUTPUT	INPUT	INPUT	OUTPUT
7	1	0	0	0	1	0	1	1	OUTPUT	INPUT	INPUT	INPUT
8	1	0	0	1	0	0	0	0	INPUT	OUTPUT	OUTPUT	OUTPUT
9	1	0	0	1	0	0	0	1	INPUT	OUTPUT	OUTPUT	INPUT
10	1	0	0	1	0	0	1	0	INPUT	OUTPUT	INPUT	OUTPUT
11	1	0	0	1	0	0	1	1	INPUT	OUTPUT	INPUT	INPUT
12	1	0	0	1	1	0	0	0	INPUT	INPUT	OUTPUT	OUTPUT
13	1	0	0	1	1	0	0	1	INPUT	INPUT	OUTPUT	INPUT
14	1	0	0	1	1	0	1	0	INPUT	INPUT	INPUT	OUTPUT
15	1	0	0	1	1	0	1	1	INPUT	INPUT	INPUT	INPUT

Blue Chip数据采集系统，在本书中用作讲授的例子，它将两个8255端口映射到同一个插入式卡上，提供了48根线。端口接在IBM PC机后面的一个单个的50线连接器中。使基地址在试验区300H到377H范围内是可选择的，这可以避免总线竞争。

1.6 IBM PC总线

如图1.4所示，PC-XT总线是8位的数据总线，在62脚的边缘连接器上。总线信号中的许多信号用于直接存储器访问和中断处理，在初次读时可以忽略。

地址线A0-A19可以寻址1M字节的地址空间。尽管8088处理器可以使用所有的16根线A0～A15来寻址64K字节的I/O空间，但是只有10根线实际参与译码，使可利用的端口数目限制为1024。这些可用的I/O地址许多已被IBM本身所采用，这些指定的地址示于表1.2中。尽管I/O空间差不多已用满，还是有一些可利用的端口，特别是在试验区域300H—31FH内。然而，某些外围板厂家用了这些地址的一部分供他们自己的产品使用，这样你就必须寻找另外的I/O空间。很明显，最简单的方法就是使用未被占用的指定的I/O地址。

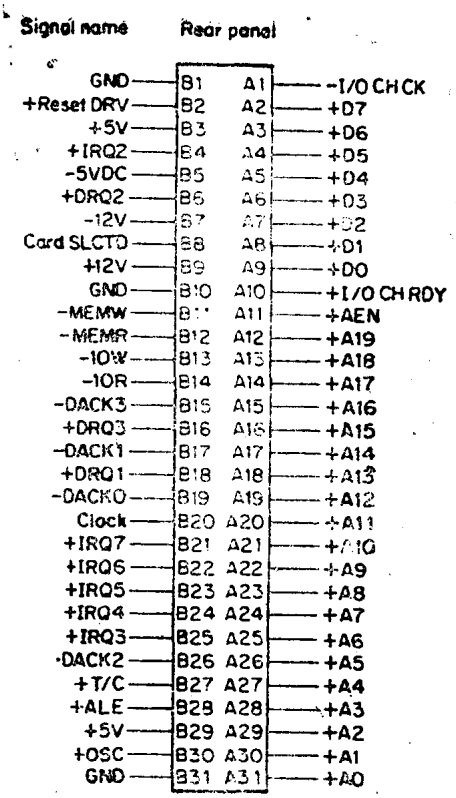


图1.4 IBMPC总线结构

表1.2 IBM I/O设备的寻址空间

000—00F	DMA芯片8237A-5	2F8—2F7	异步通信(第二个)
020—021	中断8259A	300—31F	试验卡
040—043	定时器8253-5	320—32F	硬磁盘适配器
060—063	可编程外围接口8255A-5	378—37F	打印机
080—083	DMA页寄存器	380—38C	SDLC通讯
0AX	NMI屏蔽寄存器	380—389	双同步通讯(第二个)
0CX	保留	3A0—3A9	双同步通讯(第一个)
0EX	保留	3B0—3BF	IBM单色显示/打印机
200—20F	游戏控制	3C0—3CF	保留
210—21F	扩展部件	3D0—3DF	颜色/图形
220—24F	保留	3E0—3EF	保留
278—27F	保留	3F0—3F7	磁盘
2F0—2F7	保留	3F8—3FF	异步通信(第一个)

1.7 用Basic和C访问指定的存储地址

在成功地用C接口之前，必须先从指定的存储器地址中存取数据。为了便于理解，我们不一下子给出所需的C语言设计，而是先复习所熟悉的Basic命令和程序结构。

注：从GW-BASIC和C编程手册中摘录的内存和口地址读写

POKE语句

句法：poke address, byte

作用：写一字节到一内存地址中。

注解：变量address代表内存地址，byte为要写入的数据字节，它必须在0-255范围内。address必须在-32768-65535范围，它是当前段的偏置，该段由最近的DEF SEG语句设定。对于address的负值的解释见"VARPTR函数"。与poke相对应的函数是PEEK。

注意：要小心使用poke，若使用不正确，可能导致系统被破坏。另见：

DEF SEG, PEEK, VARPTR

例：10 POKE &H5A00, &HFF

PEEK函数

句法：peek(n)

作用：返回由指定的存贮器地址中字节值

注解：返回值是0-255内的一个整数。

整数n必须是在-32768-65535间。变量 n 是当前段的偏移，该段由最近的DEF SEG语句设定，对于n的负值的解释，见"VARPTR函数"。peek是poke语句的对应函数。

例：a=peek(&h5a00)

这个例子中，在十进制地址5A00位置的值被装入变量A。

OUT语句

句法：out port, data

作用：送一字节到一机器输出口

注解：port是口地址，它必须是在0-65535范围的一个整型表达式。data变量是要传送的数据，它必须是在0-255一个整型表达式。

例：

```
100 OUT 12345, 255
```

用8086汇编语言，这等价于：

```
MOV DX, 12345
MOV AL, 255
OUT DX, AL
```

INP函数

句法：INP (PORT)

作用：返回由port口读取的字节port必须是0-65535范围的一个整数。

注解：inp是out语句的对应函数。

另见 out

例：这条指令从口54321读取一个字节并把它赋给变量A，

```
100 A=INP(54321)
```

用8086汇编语言，这等价于：

```
mov dx, 54321
in al, dx
```

在IBM PC上运行的GW-Basic用Peek和Poke支持存储器映射式I/O，同时用Inp和Out命令支持端口映射式I/O。C语言有类似的结构。前者需要使用指针——如果不熟悉的话，用指针是非常棘手的事情。然而，在为I/O设备没有单独的地址空间的计算机系统中(如IBM PC)，指针不能用来访问I/O设备。IBM分配地址从768到799（十进制）的范围，专门用于I/O试验，见表1.2。

对于这种类型I/O系统的机器，C编译器通常有库函数，允许直接访问端口映射式I/O空间。例如，Microsoft C5.1版本提供在头文件conio.h中定义的函数inp()和outp()，结合另外的编译器指令#include <conio.h>，这两个函数在编译程序时将成为程序的一部分。

不管使用何种语言，从指定的存储器单元存取数据对接口来说都是很重要的。所以，有必要从GW-Basic和Microsoft C编程人员手册中做出前面的摘录，以便于比较。

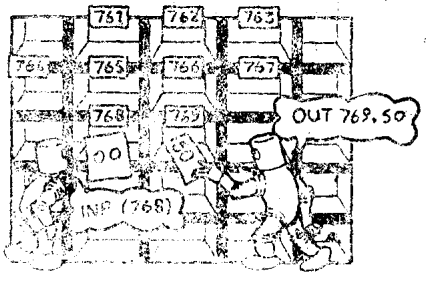


图1.5 Peeking和Poking

相信一张图胜过千言，我们给出了图1.5中这个已经熟悉形象的图形用来说明如何用GW-Basic和Microsoft C来存取端口映射式数据。下面的例程

中没有包括初始化部分。

程序例1：

```
10 REM PEEKING I/O
20 P = INP(768):REM READ PORT A
30 PRINT P
Example 2
10 REM POKING I/O
20 OUT 769,50:REM WRITE PORT B
Example 3
/*********************
* PEEKING I/O WITH C *
*********************/
#include<stdio.h>
#include<conio.h>
main()
{
unsigned char p;
p = inp(768);
/*----------
READ PORT A
----------*/
printf("%d\n",p);
}
Example 4
/*****************
* POKING WITH C *
*****************/
#include<stdio.h>
#include<conio.h>
main()
{
outp(769,50)
/*-------------
WRITE TO PORT B
--------------*/
}
```

1.8 用指针读I/O空间的内容

这里初始的目标是说明如何编写C程序来读出并显示如表1.2中所示的I/O地址空间的内容。可以有两种方法：要么先知识性地给出必要的C结构，然后给出程序；要么先给出程序，由读者自己运行它，然后再说明合适的结构。这里采用后一种方法，因为一个成功的程序会给读者正确的指导，促使读者能批判性地学习书中要讨论的内容。请注意：程序1.1不是一个基本程序，它有许多复杂的地方。学完第一章后再考虑其细节，然后对它作出改进。

程序列表1.1：

```
Listing 1.1
/*************************
* READING I/O ADDRESSES *
* USING POINTERS *
*************************/
#include<stdio.h>
main()
{
```

```
* int *port_x;
  unsigned char contents
  int i,j,x;
  /*-----------------------------------
    port_x IS A POINTER DECLARED AS
  AN INTEGER. THE VARIABLE contents
  IS AN UNSIGNED CHARACTER. THE
  VARIABLES i, j AND x ARE INTEGERS
  -----------------------------------*/
  clrscr();printf("Input base address");
  scanf("%d",&x);
  /*-----------------------------------
  ENTER A DENARY INTEGER x FROM THE
  KEYBOARD
  -----------------------------------*/
  for(j = x;j <= 16 + x;j++)
  {
  i = j;
  port_x = (int *)i;
  /*-----------------------------------

  THIS CONSTRUCTION ESTABLISHES THE
  ADDRESS OF THE POINTER
  -----------------------------------*/
  contents = *port_x;
  /*-----------------------------------
  WHEN * IS USED AS A PREFIX TO AN
  INTEGER VARIABLE NAME WE RECOVER
  THE VALUE AT THAT ADDRESS
  -----------------------------------*/
  printf("%d\n",contents);
  /*-----------------------------------
  PRINT THE DENARY CONTENTS OF THE
  I/O ADDRESSES ON THE SCREEN
  -----------------------------------*/
  }
  goto start;
  }
```

连接、编译程序1.1，然后执行。程序要求你从键盘输入十进制的基址。参照表1.2找到I/O地址合适的值。执行结果是从基址开始的16个相邻地址的内容，并显示在监示器上。现在，输入一个新的基址，观察程序重复上述过程，未被占用的I/O地址中的内容被设置为0。

1.9 C程序的开发

C语言需要编译，程序语句，即源代码，并不像解释语言那样可以直接执行。它们需要用文本编辑器或字处理器写入一个叫源程序的文件中，然后由C编译器对源程序进行处理。编译器的输出就是源程序相应的

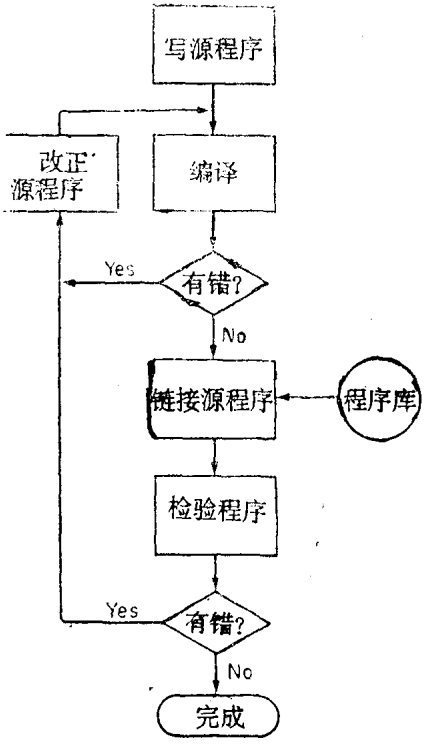


图1.6 C程序开发