〔美〕Russell Rector-George Alexy 著
李三立　潘孝梅　王秀玲　译
潘孝梅　校

十六位微处理机

8086

清华大学出版社

十六位微处理机

8086

李三立　潘孝梅　王秀玲　译
潘孝梅　校

清华大学出版社

内 容 简 介

本书由Russell Rector—George Alexy所著《THE 8086 Book include 8088》一书翻译而成。

全书共分十章。第一、二章讨论程序设计的一般概念及程序举例；第三、四章逐条地讨论了8086的指令系统；第五、六章为软件开发及8086汇编语言举例；第七、八章描述了8086微处理机的结构，並对8086的系统设计作了深入的讨论；第九、十章介绍多总线和多处理机的概念。

8086微处理机是当前流行的十六位机型之一，本书是目前较全面和较系统介绍8086的一本书，可供微型机领域内技术人员、高等院校高年级学生、研究生和教师参考。

十六位微处理机8086

李三立 潘孝梅 王秀玲 译

潘孝梅 校

☆

清华大学出版社出版

（北京清华园）

北京昌平县印刷厂印刷

新华书店北京发行所发行 各地新华书店经售

☆

开本：787×1092 1/16 印张：23.75 字数：547千字

1986年9月第1版 1986年9月第1次印刷

印数：00001～10000

统一书号：15235·209 定价：4.85元

译 者 的 话

随着八位微型机应用的普及和深入，各个领域对微型机的性能提出了更高的要求。大规模集成电路工艺的迅速发展，微处理机电路的集成度迅速提高，促使功能很强的微型机不断出现。目前多种十六位微型机系统在国际上已得到广泛使用，并日趋成熟。当前，国内微型机领域和其他领域内的技术人员对十六位微型机给予极大的关注，有些单位已开始使用十六位微型机系统，我们深信不久它将会得到广泛的使用。考虑到当前有关十六位微型机的资料还不多，而INTEL公司的8086微处理机又是当前流行的一种机型，我们决定将本书提供给读者。

本书系根据Russell Rector—George Alexy所著《The 8086 Book include 8088》一书翻译而成。作者对8086微处理机的编程作了基本介绍，对8086的指令系统进行了逐条的讨论，并对8086的结构，系统设计和多总线，多处理机等诸方面进行了广泛的阐述。

本书的第五、七、八、九、十章以及附录A、附录B由李三立翻译，第一、三章由潘孝梅翻译，第二、四、六章由王秀玲翻译，全书由潘孝梅校对。

由于译者水平有限，加之时间仓促，译文中定有错误和不妥之处，敬请读者给予指正。

译校者 1983年10月

目　录

第一章　程序设计基本知识 ……（1）
§1.1　汇编语言 ……（1）
§1.2　程序设计的任务 ……（3）
1.2.1　系统的设计说明 ……（4）
1.2.2　程序设计 ……（6）
1.2.3　实现 ……（6）
1.2.4　测试 ……（8）
1.2.5　文件 ……（9）
1.2.6　维护 ……（10）
第二章　一些程序举例 ……（11）
§2.1　排序程序 ……（11）
2.1.1　输入 ……（12）
2.1.2　计算 ……（12）
2.1.3　输入记录格式 ……（13）
2.1.4　排序方法 ……（13）
2.1.5　输出记录格式 ……（15）
2.1.6　输出 ……（15）
2.1.7　出错处理 ……（15）
2.1.8　程序设计 ……（15）
第三章　8086汇编语言指令系统 ……（18）
§3.1　一个输入/输出驱动程序 ……（20）
3.1.1　输入 ……（20）
3.1.2　计算 ……（23）
3.1.3　输出 ……（24）
3.1.4　程序设计 ……（24）
§3.2　8086指令系统 ……（26）
§3.3　8086寄存器和标志 ……（27）
3.3.1　通用寄存器 ……（28）
3.3.2　指针寄存器 ……（29）
3.3.3　变址寄存器 ……（29）
3.3.4　分段寄存器 ……（29）
3.3.5　标志寄存器 ……（30）
3.3.6　指令对标志寄存器的影响 ……（31）

§ 3.4 8086寻址方式……（34）
3.4.1 程序存贮器寻址方式 ……（35）
3.4.2 数据存贮器寻址方式 ……（36）
3.4.3 寻址方式字节 ……（40）
3.4.4 分段跨越 ……（42）
3.4.5 存贮器寻址表 ……（43）
§ 3.5 指令系统助记方法……（43）
§ 3.6 8086汇编语言指令……（46）
§ 3.7 与汇编程序有关的助记符……（181）
第四章 8086 指令组……（182）
§ 4.1 数据传送指令……（182）
4.1.1 缓冲区到缓冲区传送子程序 ……（182）
4.1.2 保存机器状态 ……（193）
4.1.3 分段寄存器初始化 ……（194）
§ 4.2 算术运算指令……（195）
4.2.1 加法指令 ……（195）
4.2.2 减法指令 ……（199）
4.2.3 乘法指令 ……（199）
4.2.4 除法指令 ……（205）
4.2.5 比较指令 ……（206）
§ 4.3 逻辑运算指令……（211）
§ 4.4 串基本操作指令……（221）
4.4.1 REP前缀 ……（222）
§ 4.5 程序计数器控制指令……（223）
4.5.1 条件转移指令 ……（230）
4.5.2 LOOP指令 ……（230）
§ 4.6 处理器控制指令……（230）
§ 4.7 输入/输出指令 ……（230）
§ 4.8 中断指令……（240）
§ 4.9 循环和移位指令……（240）
第五章 软件开发 ……（248）
§ 5.1 编辑程序……（249）
5.1.1 编辑程序的功能 ……（250）
5.1.2 系统命令 ……（254）
§ 5.2 汇编程序……（255）
§ 5.3 调试程序……（257）
第六章 8086 汇编语言编程举例……（259）
§ 6.1 排序程序……（259）

§ 6.2 输入/输出驱动程序 ……（267）
第七章 8086 微处理机描述 ……（273）
§ 7.1 8086CPU 引线和信号 ……（273）
7.1.1 地址和数据线 ……（274）
7.1.2 控制和状态线 ……（275）
7.1.3 电源和定时线 ……（278）
§ 7.2 8086 概论与基本系统概念 ……（279）
7.2.1 8086总线周期定义 ……（279）
7.2.2 8086地址和数据总线概念 ……（281）
7.2.3 系统数据总线概念 ……（284）
7.2.4 8086执行部件和总线接口部件 ……（289）
7.2.5 8086指令队列 ……（290）
第八章 基本的8086单CPU系统设计 ……（294）
§ 8.1 操作方式 ……（294）
8.1.1 最小方式 ……（294）
8.1.2 最大方式 ……（294）
§ 8.2 时钟的产生 ……（299）
§ 8.3 复位 ……（303）
§ 8.4 “准备就绪”信号的产生和时序 ……（306）
§ 8.5 中断结构 ……（308）
8.5.1 预先定义中断 ……（309）
8.5.2 用户定义的软件中断 ……（310）
8.5.3 用户定义的硬件中断 ……（310）
8.5.4 中断响应时序 ……（311）
8.5.5 系统中断结构 ……（314）
§ 8.6 8086总线时序波形分析 ……（316）
§ 8.7 最小方式总线时序 ……（317）
8.7.1 地址和允许地址锁存(ALE) ……（317）
8.7.2 读周期时序 ……（317）
8.7.3 写周期时序 ……（318）
8.7.4 中断响应时序 ……（318）
8.7.5 准备就绪时序 ……（319）
8.7.6 总线控制传送时序 ……（320）
§ 8.8 最大方式总线时序 ……（320）
8.8.1 地址和允许地址锁存(ALE) ……（320）
8.8.2 读周期时序 ……（321）
8.8.3 写周期时序 ……（321）
8.8.4 中断响应时序 ……（322）

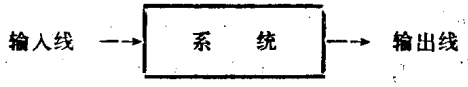
8.8.5 准备就绪时序 ……（322）
8.8.6 其它考虑 ……（323）
§8.9 总线控制传送(HOLD/HLDA和RQ/GT)……（323）
8.9.1 最小方式 ……（323）
8.9.2 最大方式($\overline{RQ}/\overline{GT}$) ……（327）
第九章 多总线 ……（332）
§9.1 启动信号线……（332）
§9.2 地址和禁止线……（332）
§9.3 数据线……（335）
§9.4 解决总线竞争线……（335）
§9.5 信号传送协议线……（336）
§9.6 非同步中断线……（336）
§9.7 电源供给线……（337）
§9.8 保留的引线……（337）
§9.9 多总线结构的概念……（340）
第十章 8086 多处理机结构……（342）
§10.1 协处理机……（342）
§10.2 共享系统总线的多重处理……（344）
§10.3 8289的总线访问和释放的选用方式……（350）
附录A 按字母排列的 8086 指令系统 ……（352）
附录B 按数字顺序排列的 8086 指令系统目标代码……（361）
附录C 8088 CPU ……（369）

第一章　程序设计基本知识

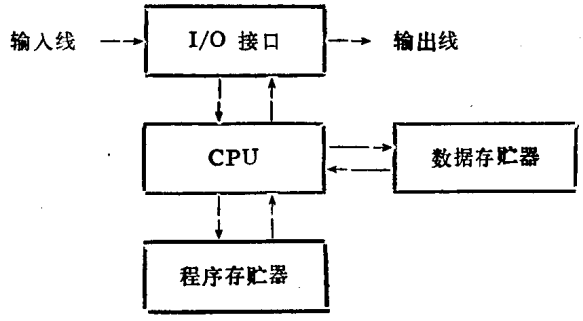
§1.1　汇编语言

在微型计算机系统中汇编语言的功能是什么？它与机器语言或更高级语言的程序设计有何区别？本章将通过评介汇编语言所起的各种作用来回答这些问题。

一般来说，所有的微型计算机系统具有以下的形式：



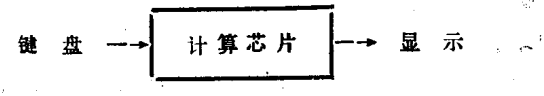
这里，输入线用来给系统提供信息，而输出线用来从系统发送信息。通常，系统由以下各部分组成：



中央处理部件(CPU)通过I/O接口从输入线得到数据；通过执行程序存贮器中的程序，CPU处理这数据。具结果经输出线输出。CPU把暂时的数据存入数据存贮器中。

CPU，I/O接口，以及物理存贮器是系统的硬件部分。驻留在程序存贮器中的数据是系统的软件部分或称程序。把 8086 汇编语言的元素组合起来形成 8086 汇编语言程序，该程序经处理并存入程序存贮器。因此，汇编语言用来说明驻留在程序存贮器中的程序。

为了理解程序的概念，考虑一台具有下述零件的销售点终端：



当有键敲入时，计算芯片进行运算，把每个键入转换成机器可接受的代码。这代码可代表被处理的数或所执行的计算。通过译码，计算芯片执行所要求的操作，并在显示器上指示出结果。

计算芯片通过执行任务序列以完成这些操作。例如，计算芯片可执行下面的任务序列来确定是否有键已经按下。

1．读入键盘状态字节。

2．从状态字节分离出位3。（如果位3为0，表示没有键被按下，如果位3为1，则已经有键按下）。

3．测试位3。（如果位3为0，回到第1步，如果位3为1。做第4步）。

4．执行下一个任务，这可以是一条清除键入位的命令或是一条屏蔽键盘的命令。

由计算芯片执行的整个任务组，包括所有的翻译和计算操作，称之为算法。算法是由具有确定任务的有序序列组成的，这种有序序列有一个起始语句，并有一个标准的停止语句。算法通常以上面例子的形式表达，即描述所执行任务的英语句子。不幸的是，CPU不能响应像“读入键盘状态字节”的英语句子，必须将英语句子组成的算法转换成CPU可解释的形式，它由二进制的CPU指令序列组成。用来实现算法的CPU指令组称为目的程序。

CPU通过分析由二进制数0或1组成的信息组执行指令。简单的CPU有二个周期，取指和指令执行。在取指周期，CPU产生下条要执行的指令的单元地址，CPU请求存贮器提供在该单元中的信息组，存贮器产生相应的信息；下个指令执行周期中，CPU分析信息，并执行相应的动作。

例如，假设下述数据是存在于INTEL8086系统中用来执行上面例子所示的任务（地址和指令用二进制表示）。

地址	指令
0 0 0 0	1 1 1 0 0 1 0 0
0 0 0 1	0 0 0 0 1 0 1 0
0 0 1 0	0 0 1 0 0 1 0 0
0 0 1 1	0 0 0 0 1 0 0 0
0 1 0 0	0 1 1 1 0 1 0 1
0 1 0 1	1 1 1 1 1 0 1 0

如果8086从0000单元开始执行，先读取存放在0000单元中的第一条指令，并分析指令。CPU确定这是一条输入指令，并且在下个0001单元中包含了从中读取数据的设备地址，因此，设备码就是00001010。如果设备码为00001010的设备提供键盘状态字节，则执行此8086指令将把键盘状态字节读入8086的AL寄存器中。0000单元中的指令执行后，下条执行的指令将在0010单元中。0010中的指令用0011中的信息与AL中的内容执行逻辑乘操作。正如上面例子中第二项任务那样，在操作分离出位3。0100和0101中的指令判定位3是1或0，以便执行相应的动作。

CPU以0和1的信息操作，然而，人们不擅长使用1和0。因此，在CPU的1和0信息与人们之间提供一个中间步骤，这步骤就是汇编语言。人们用汇编语言书写程序而不是直接将1和0的信息送入计算机。用称之为汇编程序的程序将汇编语言程序转换成相应的1和0信息。用汇编语言书写的用户程序称为源程序。

例如，下面这些行的8086汇编码（源码）可输入到汇编程序以代替上面所示的用1和0建立的程序。

```
TOP: IN    AL, 0AH
     AND   AL, 08H
     JNZ   TOP
```

汇编程序把源码转换成上面例子的1和0信息(目的码)。

例如，8086 汇编语言指令：AND AL，08H由汇编程序转换成二字节的目的码：

```
0 0 1 0 0 1 0 0
0 0 0 0 1 0 0 0
```

该目的码由 8086 CPU 译作00001000与AL寄存器的内容执行逻辑乘的一条指令。

有理由说明汇编语言程序设计比用二进制码的程序设计更加有效。首先，使用汇编语言指令如AND，ADD，或XOR，书写汇编码远比用如01001000，10100010，或01110000书写指令容易得多。其次，送入CPU机器指令，出错可能性非常高；但书写汇编语言时，如果有错，汇编程序通常会发现它。

§1.2 程序设计的任务

现在考虑程序员与微型计算机系统之间的关系。为了使微型计算机系统工作，以下就是通常程序员所要执行的任务：

1．系统的设计说明。它包含关于系统所要提供的全部功能的一般讨论，没有关于系统所管理的输入和输出的字符的描述。

2．设计在给定的系统上实现设计说明的计算机程序。这就要求把设计说明翻译成一系列步骤，使所采用的系统适应特殊的应用。

3．使用特殊的计算机语言以实现程序设计，这步骤包含三个单独的任务：编码，调试以及综合。

4．测试全系统。测试数据组输入到系统，此数据用来考验程序逻辑和硬件元件。

5．系统的文件。这些文件要求包括整个系统如何工作的描述，操作员指南，以及程序的完整的文本。

6．系统的维护，如果提出新的要求或需要新的设备，则要有一个更新系统的计划。

在设计任何复杂系统的时候，总要使用上面的列表。然而，少数的情况下，并不需要包括关于系统扩充的三个部分的250页的设计说明、50页的操作员指南、以及严格的测试步骤。这些程序会按下述的类型出现：

1．一个串行的I/O通道失效。硬件人员指出了软件出错处，软件人员发现如此差的硬件片会没有故障是不可能的。解决的办法是设计一段短程序，该短程序首先初始化通道，然后，每次串行I/O通道指出数据有效时，就读出和显示数据。确定硬件是否正常工作，这应该是比较容易的；如果硬件工作正常，那末，几乎不用怀疑什么会出错。

2．需进行少量的并非无关紧要的计算。幸运的是可以采用FORTRAN系统。一个具有20条语句的FORTRAN程序将产生所希望的结果。

在上面两种情况下，几乎没有什么设计说明或程序设计记录下来，这些步骤是留在程序员的脑中，在这些情况下可能不产生文件，值得怀疑，维护是否必须。然而，记住

这些情况是法则的例外事情，这才是明智的。

上面列出的任务在有多批程序员的场合使用非常有效。某些程序员仅仅执行第1和2步，有些程序员只执行实现过程，某些人花费他们的大部分时间致力于测试程序系统，其他一些人从事文件编制和(或)维护任务，以及另外一些人执行深奥的任务组合。按这种方法，程序员将发展促使获得更高生产率的专门化技术。然而，在大部分汇编语言的文本中，汇编语言程序员调去执行上面全部的任务。本书着重以汇编语言方法探讨8086，所以对所有的这些任务只作一般性讨论。

1.2.1 系统的设计说明

当开始考虑要得到微型机系统时，可以由下面两种分析之一得出结论：

1．需要解决的特殊的问题。例如，一家航空制造厂将生产需要符合一定尺寸和速度要求的机载计算系统的导弹制导系统。

2．新微型计算机系统的特种市场。例如，原先买不起计算机化会计系统的小企业，当微型计算机在商业系统的价格跌得足够低时也想买了。

在两种之一的情况下，详细说明预想的系统将完成的确切功能是很重要的。在第一种情况中，问题的实质可能是限定该系统执行特定的功能。在第二种情况中，设计说明中的要求是容易达到的。对于小的商业系统，必须正确地规定将完成的会计功能，以及确切地规定系统中将提供多少个各种类型的记录。否则，微型计算机系统会承担比硬件所具有的处理能力更重的任务。

参阅本章前面的微型计算机系统的最简单的模型，设计说明将规定如下：

- 由系统接收的输入。
- 由系统执行的计算。
- 由系统建立的输出。

输入

微型计算机系统输入的技术条件很大程度上取决于所执行的程序设计的级别。一个从事应用的，用BASIC编程的程序员不大可能关心由磁盘控制器给出的命令的类型，他倒是关心磁盘上数据的类型，磁盘文件中记录是如何放置的，操作系统又是如何管理磁盘文件的等。因为本书关心汇编语言程序设计，以及关于数据库管理技术的任何适当的讨论总会超出本书的范围，我们将着重在硬件级上讨论输入和输出的技术条件。

在硬件级上，三个参数将确定一个输入通道的特性，它们是：

1．数据路径宽度。输入可以每次1位地来自处理机控制器出错系统。一个并行的或串行的I/O通道每次将输入8位。当请求时，软盘控制器可发送1024位(128字节)信息。

2．数据传送速度和类型(同步或异步)。数据根据实时钟每200μs到达一次。串行I/O通道可异步地每10ms输入数据。控制系统中的A/D转换器可按不同的速率传送数据，但不快于每500ms传送一次。

3．附带控制信息。当数据有效时，软盘可产生一次中断，键盘子系统可置状态位。A/D转换器可要求系统读取输入数据，并与前面的数据比较以确定新的数据是否有效。

对这些参数作了说明后，重要的是需要详细说明输人通道是如何实现的。

输入通道通常有三类端口：

1．数据端口。这些端口包含将被传送到系统的处理部件的数据。

2．状态端口。这些端口包含指明数据何时有效，是否在本通道发生了错误的信息，以及有关外界的其它信息。

3．控制端口。这些端口通常用来初始化通道的操作方式以及用来控制通道本身提供给外界的方法。

全部的三种端口并非总是工作着。在某些情况下，只是数据通道存在，某些情况下，当加电时通道自动地被初始化，因而使控制端口变得不是必须的。

计算

在说明微型计算机系统的计算部分时，有三方面的问题需要考虑：

1．处理来自输入部分的原始数据。可以采用转换成系统更易于使用的代码的方式(例如，从ASCII码到二进制)，可将数据块分成为各组成部分(例如，软盘上一个扇区的数据分成文件首部，首部检查和，数据，以及数据检查和)。

2．系统所执行的实际算法。实际算法的完整描述通常是在程序设计中进行，设计说明中有关这部分应列出系统将执行的主要功能。

3．为输出部分处理数据。该处理可包括将数据转换成易于输出设备使用的形式(例如，二进制数转换成EBCDIC)。

输出

对于一个微型计算机系统输出的设计说明需要非常类似于输入部分所进行的分析。对于每个输出通道，有三个主要的参数：

1．通道发送的位数。

2．输出通道的数据传送速度。

3．附带的控制信息。它使系统知道发送器何时要求更多的数据，或何时可用来管理更多的数据。

对这些参数作说明后，需要详细说明通道是如何受控制的。正如输入通道那样，输出通道通常有三个重要的端口：

1．数据端口。这些端口接收发送到外界去的数据。

2．状态端口。这些端口包含了指明何时数据可以发送到数据端口，通道是否出错的信息，以及有关外界的其它信息。

3．控制端口。这些端口通常用来初始化通道的操作方式以及用来控制通常本身提供给外界的方法。

正如输入通道那样，控制一个输出通道并非所有端口都是必须的。

在对三个主要部分的每一个进行设计说明的过程中，有许多有用的技术应牢记：

1．在每部分中，造一个可能出现的出错条件和系统响应出错的列表。

2．在每部分中，造一个该部分所执行的全部功能的列表，例如，造一个全部输入通道，全部计算功能，以及全部输出通道的列表。在特定的部分操作结束时，用列表以不同方法相互校对该部分，则有希望保证对于系统所有的可能性都已经考虑到了。最初

写出的设计说明不一定是最后的结论，除非眼前的问题十分简单，几乎一定不是最后的说明。程序设计任务和实现任务可能出现有些功能在选定的硬件结构下不能执行，在这种情况下必须修改设计说明以改变硬件结构，或者修改有错的功能，使给定的硬件结构能实现此功能。

1.2.2 程序设计

程序设计包括取出设计说明中的语句，并写一个描述实现设计说明方法的英文语言步骤序列。通常这些英文语言步骤将提供对系统所能实现的功能的清晰而简练的描述。这里，并非一个简单的描述适用于所有系统，例如，不要指望找到一个IBM的DOS/VS操作系统的简要的描述。但另一方面，要使描述对整个系统是正确的，这就要求一个简练的描述在理想情况下应适应系统的各个部分(例如，一个打印机驱动程序或一个多字减法子程序)。当考虑一个很大系统中各组成部分的数目时，可以把一个大的任务分成许多较小的模块、然后让每个人完成整个程序设计员任务的一小部分工作。

在从事程序设计任务时，应记住以下这些启示：

1．今后，程序必定还要扩充以提供更强的功能。因此，程序应具有内部扩充的可能性。这将包括系统子程序，可扩充的表格以及数据列表，使系统增加更多功能的方便的文件方法，以及相当灵活的数据结构。

2．在一个典型的设计中，为实现任何给定的功能，可以使用多于一种的方法。在某些情况下，机器的局限性限制使用一种方法，在另外的情况下，由于时间的制约只能使用另一种方法。因为，这些因素(即机器和时间的限制)直到实现任务时才能被知道(在实现任务中才能进行实际编码)。所以在设计阶段，采用解决一个特殊问题的替换方法是明智的。在这阶段，寻找替代物的好处有两层意思：首先，如果列举的限制阻止使用一种方法，则准备使用另一种方法；其次，在设计过程中你可发现更高效率的方法。

3．设计时，说明特定的模块对其它模块的影响是非常重要的。同等重要的，还要说明其它模块对该模块的影响。当调试和综合程序模块时，模块之间的连接就又变得重要了。

当程序设计完善后，可以用该设计再审查设计说明。交替检查设计与说明可以暴露出设计和(或)设计说明中的缺陷或疏忽之处。应该在有规律的基础上审查设计。当执行实现和测试任务时，新的信息会变得有用，从而可以对程序设计重新作出评价。

1.2.3 实现

实现任务包括使用程序设计任务中特定的英文语言算法并使它在特定的微型计算机系统上工作。

进入实现任务有两项不同的工作要做：

1．编码。这是把在程序设计任务中建立的英文语言步骤转换成特定的计算机语言的过程。

2．调试和综合。这是从编码项转换成计算机语言的程序设计模块过程中排除错误，然后，将这些模块归并到一个工作系统的过程。

编码

程序设计转换成一种特定的计算机语言是程序员较容易的任务之一。如果程序设计的工作已经正确完成，每个单独的模块将由一组简要的英文语言语句来描述。编码时，应记住以下的启示：

1．尽可能使用标准子程序或程序。子程序非常有用，可以单独地对子程序调试。子程序排除故障后，主程序代码行的调试就更加容易了。此外，标准子程序使系统增加新的特点就更加容易了。

2．为代码提供的资料尽可能清楚。除了描述各个模块或代码部分的注释语句外，由助记符组成的标号的有效位数应具有大的值。某些汇编程序限制这种做法，例如限制标号中的字符数为6位或更少。然而，在大多数情况下，能够给程序和数据区中的标号赋予特殊的助记符值，并可充分地运用。

在每个程序设计模块转换成相应的计算机语言后，应做一系列的检查以保证编码任务正确地进行，并且在执行调试任务时，避免潜在的困难。该检查系列，有时称为桌上检验，是编码步骤的一部分。但是也与调试任务共享许多元素。

应做的检验包括：

1．保证代码包含全部程序设计模块。

2．保证全部程序设计中的判定都包括在代码中，检验全部判定点处的逻辑，以保证程序分支能正确地执行。

3．保证以足够的信息提供给每个程序设计模块，使它能正确地运行，可通过确定本模块希望其它模块提供何种信息，对每个模块执行检查。根据以下进行选择：

- 寄存器的内容。
- 数据结构的内容。
- 由本模块使用的I/O设备的状态。
- 状态设置。

4．保证每个模块以正确的信息提供给下一个模块。可通过确定本模块必须提供给其它模块何种信息，对此模块执行检查。根据以下进行选择：

- 寄存器内容。
- 数据结构的内容。
- 由下一个模块使用的I/O设备的状态。
- 状态设置。

5．保证代码已进入到该模块，以处理下述的情况：

- 出错。
- 特殊情况。
- 边界情况。
- 无用情况。

这些检验完成后，调试步骤即可开始。

调试和综合

调试和综合任务是由从代码中排除错误和把调试过的模块并入最后工作的系统这二

部分组成。调试任务期间所执行的功能非常类似于桌上检验所执行的功能。调试任务不同之处是：当检查在系统硬件或系统硬件的仿真器上运行的代码时，执行此任务。有一系列调试过程中使用的工具，而这些在桌上检验时是无用的。这些工具通常是(但并非总是)由称为调试程序的软件模块提供。调试程序提供的典型的特性包括：

· 单步功能。本功能使用户能按程序逻辑逐条执行指令。

· 检查/更换存贮器单元或寄存器的内容。

本功能允许用户查看存贮器/寄存器内容，并选择性地改换内容。

· 断点功能。本功能允许用户根据某些条件中断执行正在调试的程序。典型的断点条件包括访问一个特殊的地址，在该地址处访问数据或取指令。

在调试一个程序时，应该记住下述的启示：

1．调试过程可从调试公用的或系统子程序开始。如果系统软件中最低层的例行子程序已知能有效地运行，则发现出错源就简单了，例如，可能是主程序行有错或使用系统子程序不当。

2．如有可能，设法个别地调试设计说明的每个分区。比较恰当的是先个别地调试设计说明中输入部分的各分段，接着是设计说明中计算部分的各分段，随之，调试设计说明中输出部分的各分段。当各别地调试设计说明的各分段时，不受系统其它部分的影响来查看每个分段是可能的。理论上说，当全部的各个模块调试完后，只需调试综合项藉以使程序模块彼此相互连接。

当全部的单独的模块调试完后，综合项的工作开始。在这项工作中，将单独的各个模块连成一个子系统，然后作为一个子系统进行调试。例如，所有影响程序输入部分的程序设计模块连在一起，并进行调试。当每个子系统调试后，就能与其它子系统连接，直接调试最后的系统。如上所述，综合项所执行的唯一功能是保证模块（和最终时的子系统)之间的连接得到正确地处理。

在实现的任一阶段，必须回到程序设计，甚至设计说明任务。考虑以下这些例子：

1．在编码阶段，显然，提供所指定功能的代码将要求存贮器比硬件设计时所提供的更大。首先，返回到程序设计任务，以确定替换方法是否允许使用较少的存贮器空间，如果这不能解决问题，这就要回到设计说明项，并按某个方式重新构成系统。

2．在调试项中，注意到，如果试图运行全部系统所控制的设备，而系统不能足够快地响应。注意，这困难在调试阶段的早期并不明显，因为设计说明的输入和输出部分通常在执行综合项之前是分开调试的。在这情况下，需返回到编码任务看看，输入或输出代码的执行时间能否减少。如果不行，就要返回到程序设计任务，以确定是否可采用更有效的算法。如果太麻烦，也不成功，就回到设计说明阶段，重新改进系统。

1.2.4 测试

测试任务包括引入特殊的数组使系统运行以及检验产生的结果是否正确。这种任务在具有大量程序员的场所是很普通的。例如，任何一家自行设计软件的公司在出版新版本的操作系统之前，新系统将经受严格的考核。汽车制造厂在推出一种车载计算机系统之前，也将执行严格的测试。但是，在只有很少汇编语言程序员的情况下，测试经常被

忽视。忽视测试的主要理由是非常费时间，因而也非常费钱，此外，没有很好理解其作用。

在实现任务的调试项期间，可以完成测试任务的有效部分。例如，在调试段中，通过产生边界条件的数据运行模块来测试每个模块，因而考验模块作出判别的能力。作为一个例子，假设编码一模块，使得当一个数据块的第一字节是在30_{16}到39_{16}（包括30_{16}，39_{16}）范围内时，模块执行某一功能；如果数据块第一字节是在41_{16}到46_{16}（包括41_{16}，46_{16}）范围内，则执行另一功能；如果字节不在上面两者之一的范围内，则执行第三功能。典型的测试数据可采用以下字节作为第一字节的数据块：

$2F_{16}$，30_{16}，39_{16}，$3A_{16}$，40_{16}，41_{16}，46_{16}，47_{16}，00_{16}，FF_{16}。

这些块可测试系统在不同数据类型间的区别能力。

当决定测试数据提供给系统时，记住下述的启示：

1、三种供进入的基本数据类型是：

- 系统通常遇到的典型的数据流；
- 一串考验系统执行正确判别能力的边界条件；
- 包含合法和非法数据的随机选择数据。

2．数据应以下述的速度提供给系统：

- 系统通常遇到的典型数据速率；
- 系统藉以工作的最快数据速率；
- 随机选择的数据速率。

1.2.5 文件

文件任务由记录与系统有关的全部资料组成。在系统的文件中，有三个基本的组成部分：

1．程序文件。正如在实现任务的讨论中注意到的，重要的是需解释代码是如何一个模块一个模块地工作，以及在某些情况下，代码为何如此工作。这类文件使新的读者易于自己熟悉代码。此外，如果他们希望改变代码，好的文件可以提供比较好的有关如何改变的方法。程序文件能帮助正在检查过去所写程序的原程序员回想起原来的程序。

2．系统指南。系统指南应包括程序设计的描述，如何修改程序的描述，以及系统想了解的外部信息的摘要，例如由谁驱动输入线，及谁接收输出线上的数据。一个系统指南将十分简单地把前面各任务中已经写出的主要元素汇总在一起。

3．用户指南。这是最重要的文件篇幅。如果代码具有很好的文件形式，并且书写了极好的系统指南，那末，其他程序员就能修改或改进程序；但是如果没有用户指南，任何人就不可能使用程序，在这种情况下，就无人关心修改或改进代码，并且所有的工作将是白费的。在某些系统中，外部用户可书写与系统连接的程序，因而用户指南尤其重要。在这种情况下，通过修改用户指南，把对系统的任何再版或增添通知用户是极端重要的。