微处理机程序设计入门和基本技巧

丁祖望 译

沈建东 校

电子技术出版组

一九八四年

第一章 引 言

第一节 微处理机

微处理机（以下简称微机）是一种小型电子设备，它正在电子工业中掀起一场革命，结果必将影响到每个人的生活。历史上电子工业一向是用硬件来解决问题的。就是说，要移动、处理、使用信号或数据时，必须研制特殊的电路来达到这一目的，这种电路的用户往往并不关心电路为什么要这样工作，而关心的是如何使用电路或如何使电路动作。譬如当一位妇女按下她的微波烘炉、洗衣机，电吹风或食品搅拌器的电钮时，并不关心她的开动，更动或使用的是什么电路，往往只关心这台设备是不是照她的期望可靠地又能按预定步骤动作。因此厂家经常力图使设备的操作使用尽可能简单。

随着用户需要愈来愈复杂的设备，为了保持操作步骤简单化，所需的电路的复杂性也增加了。过去人们倾向使用比较完善和预先编好动作程序的操作，操作员用较少较简单的动作即可开动，今后的趋向亦将继续如此。这些动作通常是由一次按下某个电钮或顺次按下几个电钮来启动，以后的操作就由这台设备自己接着干，让操作人员闲出来做其它事情。

随着功能愈趋复杂，而要求用人愈少，需要用技术来生产体型更小的电路，使系统的大小在合理的范围以内。这个趋势还迫使电路接受一种更通用的设计方法。即希望将电路在电气方面稍加修改就适用于各种使用场合。寻求万能电路的结果，必然导致使用可编程序设备——对这种设备加上某种信号组合或信号序列，就可吩咐它做什么样的工作。所有这些趋向综合起来就产生了微机。

一、什么是微处理机？

微机是一种极为精致而又复杂的电子电路，它能做高难功能的操作。大的微机片在5毫米至7毫米（1/4英寸）见方的面积中就能容纳多至7万个电子元件。它们通常包装在金属、陶瓷或塑料制成的扁平、有多个接头的盒子内，看来象多脚昆虫或毛毛虫。这些组件的尺寸，长度由1至5英寸，宽度由1/4至1英寸，接头和引线以不同的方向和形状伸向组件外方。这些成排的引线有时称为管脚或接线。

最近研制的微机，只用一种电源电压来工作。之所以如此，理由很多，其中一条是系统的造价。一般规律是对电源方面要求愈多，则该微机所在系统的成本就愈高。所有微机应用都以二进制为数学基础，并用二进制来进行数据操作。就是说它们处理成串或成组电信号，这些电信号的电压电平是相同的而且又有脉冲性质，处理的信息一定要格式化或只用0和1两个数字将信息编码。

微处理机本身不过是整个系统的一小部分，整个系统通常由好几部份组成——主机或中央处理机（CPU），存储部件和各种外部设备，外部设备作为中央处理机和外部世界之间的联系。没有存储器和配套设备，中央处理机是什么事也干不成的。

二、微机能做些什么？

微机的应用与日俱增。其应用大多数都显而易见——如小型商用计算机、小型家用计算机，复杂的电视游戏、汽车中计算每加仑汽油行驶里数和行车里数的仪表。

另一种不易观察到的应用是隐蔽的控制器，如厨具控制器、家用计时器、防盗报警器、恒温器、工业机械控制器、交通设备控制器、超级市场销售终端。这种不明显的服务项目是微机对人们生活产生的最大推动力。它减轻了人们每天所做属于简单监控性质的工作。这样在烹调时人们就不必固守在厨房内。只需将各项烹调项目的开始时间、烹调温度、停火时间简单地编成程序即可。装配线上的工人将大大的减轻了他们单调的机器另件装配工作，从而加强对生产过程的监督。一个激动人心而又有革新性的应用是在机器人方面，有智能的机器（机器人）的出现已见端倪。随着电子工程师设计出更为通用的机电设备，在控制方面使用微机更是屡见不鲜。这种在应用上的可变性和灵活性，关键在于能够给机器编各种不同的操作程序。设计工程师一旦建立某种机械的一般能力之后，其特殊操作和应用可由程序员来完成。

微机吸引人的主要特点是应用广泛，灵活。某项新产品在设计、样机制造和试验阶段时，如用微机做设计基础，具有成本低周转快的特点，制造厂家可节约大量的生产成本。

三、微机是怎样工作的？

从前面的讨论自然就会产生这样的问题：怎样才能达到这种灵活性和通用性呢？这个问题也许用打比方来回答最合适。我们用一台钢琴作比喻。钢琴具备足以弹奏任何一位名作曲家创作的乐曲的硬设备。钢琴有共鸣板、琴弦、踏板、琴键等等。钢琴之所以灵活，因为在各项硬设备组合下能够产生种种不同的音响效果。由钢琴演奏家熟练地弹奏，就能产生美妙和谐的声音。就如种种不同牌号的钢琴有不同的硬设备特性一样，牌号不同的微机也有不同的硬设备特性。从硬件特点来说，以电子琴比喻更恰当，因为对电子琴人们可以购买装在琴内的各种韵律，和音功能与音色装置。有些电子琴比其它的难使用，但是生产电子琴的目的只有一个，就是要提供产生令人满意的音乐的基本乐器。而音乐是否好听，演奏是否柔和，靠乐器有何特色倒不如说靠演奏者的技巧。一台钢琴和电子琴都必须有最低限度的硬设备，如琴键、琴弦和振荡器，否则就不能工作。微机也如此。微机的最低限度设备是存储器、中央处理机、输入/输出端口和连接线。没有这些最低限度的设备，系统就工作不了，但还可给它加上其它特色，使能超过原来具备的最低工作能力。编程方面也有类似的情况，微机系统一经建成，该系统的使用就全靠程序员了。

第二节　程序的编制

一、编写程序要做些什么工作？

程序编制之于微机，犹如作曲之于钢琴或风琴，请记住这一比方。编程有其基本规律，正如作曲有其基本规律一样。程序可以从一台机器移植到另一台机器中，正如一首歌曲可以从一台电子琴转到另一牌号的电子琴上来弹奏一样。这两台电子琴可以有不同的特色，但可

用来弹奏同样的乐曲。

所谓编程是将一些基本指令组成一套完整的具有一定功能的程序。要做到这一点，就需要彻底了解使用该程序进行操作的微机所用的基本指令集。对编程规则也必须有基本了解。学习写程序需要花时间，需要善于学习，需要专心致志、并需要大量实践和灵感。

二、怎样才能写好程序？

编程工作的成果有不同水平，正如创作的歌曲有不同水平一样。读本书时要端正认识，要象初学作曲者接触钢琴那样来学习微机。

开始学习时，对如何完成程序编制，心中要有个总的看法。程序员主要参预下列三方面的工作：

1．分析提出的某一特定课题，并对该程序操作的那台微机的功能进行估量。

2．按完成的先后次序，使用的外设情况等进行算法设计。

3．写出程序，进行编码及调试来证实该程序的功能是适合的。

这些活动的结果，最后写成文件，这一文件要说明程序的复杂程度。文件还应对在系统中使用该程序的其他人提供所需的一切基本资料，因为程序系为该系统设计的。

编程工作是体力活动，也是脑力活动。编程的脑力活动部份是最花时间的。事实说明一位普通程序员每天最多能写出十行好的程序。因此在一个好程序中，大约有四十五分钟时间用来思考每一指令序列的判定。一位程序员在最后一天最终完成50至100行程序之前，可能要花费5至10天的时间来思考、分析和调试各种指令组合。

三、大家都来编制程序！

你可能会问谁来编制程序合适，我能学会编程吗？对第二个问题的答案是肯定的。谁都能将编程的基本东西学到手；然而不是每人都能成为这方面的行家。对第一个问题我们有两个答案，一个答案今天用得着，而另一个答案则为不久的将来使用。

今天，程序的编制主要是由公司雇用的个人或集体进行的，为制成供出售的设备和机器编写程序。这些人实际就是计算机时代最早的先驱。今天正在开发的程序和编程技巧，几乎是每人在最近的将来都用得着的基本构件。在不太遥远的将来，在日常生活中使用计算机和微处理器，就如我们今天驾驶汽车一样平凡。将来几乎每人都懂得编程基础知识并能写出程序。

到明天，那些不会写程序的用户，就如今天不能读书写字的用户一样罕见。今天的程序员正在研制任何人都容易学会使用的门外汉语言。因此今天认为是复杂的东西，到明天将成为基本的。

问　题

1．什么是微机？

2．什么是硬件？

3．什么是软件？

4．为什么学习编程在今天如此重要？

5．举出两种主要的微机应用范畴。

6．用你自己的话简括地说明程序员与程序时经历的步骤。

7．为什么微机在今天和将来对我们那么重要？

8．试举一些你所见到的微机应用。

9．什么是外部设备。

10．机器人这个术语指什么？

读者自行分析的问题

1．你认为你能学会编程吗？

2．你认为学会编程是难是易？

3．你的思维合乎逻辑性吗？

4．你要学好编程吗？

5．你工作耐劳吗？

6．你愿意腾出时间学习吗？

7．你接近过正在工作中的计算机吗？

第二章　程序设计概念

第一节　用程序来解决问题的一般步骤

人们在学习编程方法的时候，就应出现一种解题的设想。这也是个常识，就是说，如你碰上某问题，在试图找到解法之前，你自然想尽可能多知道与这问题有关的各个方面。需要了解问题的要求，有关的术语定义，如果解被找到了，则希望了解结果是否符合要求。在问题未完全弄懂之前，就不能指望得到解答，这一解题阶段称为提出程序设计要求，定义它的参数，或将解法分割。

一、定义问题

在探索如何确定问题的时候，要做很多决定求解方法的工作。问题的定义要尽可能完整和全面。这里可以做个颇为恰当的比方。假定我们的问题是要在一月十五日早上九时到达伊利诺州的Peoria。在明确求解方法之前，一定要回答一些基本问题：

1．Peoria的确实位置。

2．本人现在的确实位置。

3．今天是什么日子，现在是什么时刻？（如现在是一月十四日下午六时而你又在旧金山，那末你一定要对旅行方式和解决问题的办法做出迅速决定。）

然后就得回答一些可能影响问题解决细节的问题：

1．大概要花多少钱？

2．到Peoria为的是什么？为着娱乐？还是因为业务？

3．需否回程？

其中要点是这些。显然问题的原来表述是清楚而又确切的，但要开始提出问题的解决办

法，陈述中提到的信息还是不够的。这种定义问题的过程显然影响到解法的选择。这方面的工作做得愈彻底，决定最后解法就愈容易。

初始问题一经回答得尽可能完满之后，下一步就是用更简明的术语来重新表达问题，摆出影响问题解法的细节。例如问题是要在下午十时坐班机离开旧金山飞往伊利诺州的Peoria。乘坐的是ACC航空公司的109次班机（带一套衣服）。旅行的目的：明天上午九时开业务会议。回程坐明天下午三时第247次班机。Peoria预报的明天气温是15°F，整天有雨雪（整理适时的行装）。解题到这个阶段人们可以百分之百地假定这次旅程能进行下去并可能至少有一种解决办法。因此现在就是采取行动去完成解法的时候了。安排的必要事项，每次只能完成一件，并且一定要分配它们的优先级别。决定第一、第二、第三等等该做什么的过程，称为问题排序。然后把问题分解为个别的作业来完成它。主要作业一经决定以后，就开始了解每一作业块中的有关问题，以便将它们进一步分解。将问题分解成一部份一部份的过程称为'分割'。这主要是一种智力活动，牵涉到提出正确问题和根据这些问题的答案作出正确判定。对这个过程人们可以附加很多术语和设想。但最终简缩成为清楚，有条理，全部合乎逻辑思维的解法设想。

要将程序设计的定义阶段做得正确，一定要明白在开始做设计之前，人们是不能够解答解法中固有的一切问题的。一些应有的问题只当开始执行解法时才表露出来。而开始时对问题的陈述愈清楚，则当接近最后结果时，就愈容易提出问题和解答问题，这一点是明确的。

计算机程序员碰到的是一个不寻常的问题，因为他有一台要它做什么就做什么的机器，只要程序员想得出如何一丝不苟地吩咐它做'什么'。因为计算机是极端愚蠢的，这就使任务更困难了。不能设想计算机会联想。交待给计算机干的事情必须'准确'而又'周到'。这种'准确度'和'周到性'却是人类的弱点。人们思考时一般都不够精细周全。在考虑问题时，大部分人是一般化，能够周密地考虑问题的人是少数人。

在编写程序时，人们碰到的障碍和他们所受到的逻辑思维和判断力方面的训练适成反比。

二、逻辑性、周密性和准确度

人们试图做程序设计工作之前，必须理解三个术语，就是'逻辑性'、'周密性'和'准确度'。

对很多人来说，逻辑性有很多含义。要是问一位电子工程师什么是逻辑性，他会说这是将电路中多个门和多个触发器相互连接以便获得适当组合的输出或按时序的输出。对程序员来说逻辑性意味着将各项定义了的事件适当排序以期达到要求的目的。所谓适当排序是指给每项操作安排一个完成顺序以便有效地达到最终结果。所谓定义事件是指将这个有序集合中发生的每次操作完全确定下来，并在每次操作开始之前，必须取得围绕和支持这次操作的种种参数，并把它们安排在恰当的地方，对程序员来说逻辑性是一种排序工作。完整地定义每次操作和它的参数的问题就要用到周密性和准确度两个术语。

周密性和准确度这两个术语，常常被人们不正确地替换使用。周密性是指完整性程度，而准确度指的是正确性。人们可以算出一个错误的答案到非常精密的程度，但答案虽精密，它仍是错误的。今天有许多靠手持计算器来计算的学生有这方面的教训。如某人使用计算器进行解题，应按的一些键没精密地按上（这里精密指小心）结果他得到一个8位（非常精密）的

错误答案。同样，如解题时不用逻辑和正确的顺序，键是精密无误地按上了，但答案仍是错误的。

要取得问题的正确答案，逻辑性、周密性和准确度三个方面缺一不可。首先人们必须要有逻辑性（使事情顺序合理），然后是准确性（不出差错），最后要周密（详尽）。

这里花了很多时间来解释一位编程人员在编写程序时应有的方法态度。不管读者最后采取什么方法，这种态度，这种开场训练，都是极为重要的。

三、程 序 设 计

细看图2-1中的流程图。任何一位程序员的第一步操作是将问题定义并确定它是否有解。寻解的任务一经决定，第二步工作是开始设计。如图2-2所示，主要任务是努力研求一种工作顺序以便得到合理的操作。在定义阶段，需要完成的主要功能经已安排好。然后这些主功能又可分解为与主程序块的数据流或控制流方面有关的子任务和子程序。'数据流'这个术语是指程序操作时数据什么时候停留在什么地方。例如，一般数据流首先是从计算机系统外部取得数据，加以处理，然后把它显示。用框图表示就象图2-3那样。将程序分块在精确化和设计的早期看来好象无关重要，但它们是日后扩展的基础。

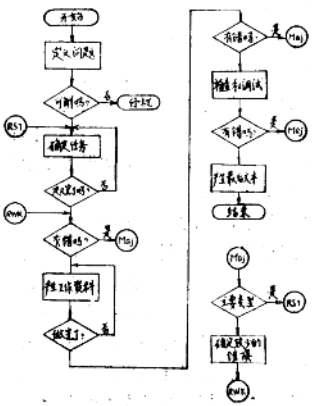


图2-1 解法顺序流程图

找出参数和限制	定义问题
表达要求的结果	
决定输入和输出的形式	
尽可能地将任务分割或模块化	
以逻辑次序来安排任务	设计程序
决定隶属的等级	
决定数据和联接的控制流	
决定采用的语言	产生工作资料
写出模块或子任务	
检查语法和术语	
产生流程图，源列表文件和目的文件	
运行程序	核对和检查程序
找出存在的问题	
得到最后的工作文件	

图2-2 解法顺序

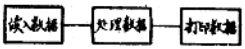


图 2-3 数据流例行程序

读者对控制流这一术语恐怕比传统用的所谓'流程图'更为熟悉。流程图早期定义为如何并在何时将资料处理。用一般措词来说，数据流定义这个过程做的是'什么'，而控制流则定义'如何'和'何时'做这个过程。这些初期的'思维'框图以后就成为程序员和由他所创立的系统之间保持联系的主要资料。随着一步步地把处理的细节不断精细化，它们间的相互关系和相互依赖就愈明显。这恰好说明设计的最初阶段为什么这样重要。如初期定义不清楚，以后的设计就可能成为改正、重新考虑、以及修补一份漏洞百出的程序的苦差事。

直到没有任何差错出现，整个设计任务才能说是完成了。同时对任务的陈述应该尽量用（英语这样的）常用的语言。在这一程序研制阶段如果使用某种特殊计算机语言或术语，只会使事情复杂化。这次努力的结果应得出可工作的解题算法，这种算法应超出对计算机系统的一般考虑，不是面向机器或语言的，机器本身特有的本领和采用什么语言放在编制工作资料时来考虑。

产生工作文件牵涉到将算法转换为某种特殊语言或机器程序。编程过程在这一阶段所碰到的困难很大程度上看使用的机器的语言能力。

四、语言等级

语言等级是指可供使用的指令的完善程度，这些指令是预先编制好了放在计算机里的。数字计算机曾一度在机器级上编程。也就是说它们的指令是用'0'和'1'形式给出的。计算机的每次操作，由一串 0 和 1 组成的序列确定并控制。这种编程工作其复杂性和工作量都很大。以后积累经验发展成为一种例行程序并在机内构成成套操作。这样，只用几行二进制语句或几个字就能启动该例行程序。于是就产生通常所谓汇编等级语言。汇编语言由用字母形式表示的缩写术语组成，这些术语称为助记符，其操作意义说明机器的活动，这样读起来就容易。这种指令完善程度的升级需要有一个汇编程序，用它来把这些术语翻译回机器语言。再高一级完善水平就是高级语言，它比之汇编语言更便于人——机对话。它的指令很多是以完整的英语单词表示的，并且常使用标准的代数符号。同样，它也需要一个称为'编译程序'的翻译程序。每种语言都有一些规则，它们支配符号的格式和书写方法。这些规则称为'语法'。必须遵守这些语法，否则指令就不会有正确的功能。

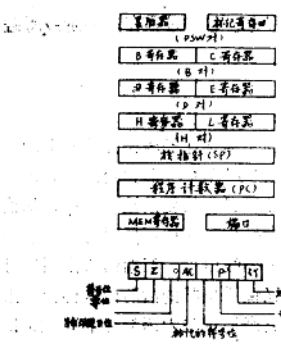
人们也可以向厂家或软件服务公司购买程序包，它允许程序员使用各种应用程序，不需自行编写。但人们使用这些商用程序包时，必须知道它们的规则、局限性和要求。

五、程序设计模型

今天制成的微机，只能接受机器指令。但是很多厂家已经研制出能够接受少量高级语言的微机系统。一般家庭用户喜欢购买一台这类的家用个人计算机并使用专为它们研制的程序。大部分个人计算机程序包是使用BASIC语言的，因此大部分家庭用户大可不必和主机

的机器语言打交道。然而，工程技术人员不大可能使用高级语言；和一台微机本身打交道的人很可能要使用机器码或汇编语言来操作；用这种程序设计语言来工作或分析时，程序员在心中或纸上必须有个清楚的图象或模型。

程序设计模型是个图解，它包括指令系统或语言命令所能操纵的所有的寄存器和功能块。在程序设计到达编制工作资料阶段，这时就要使用特定的机器级指令，如能把位处理、转移和标记结构直观化，对程序员会有很大帮助。本书自始至终使用图2-1和图2-3所示的两个程序设计模型。



寄存器名	每寄存器位数	助记符标志
累加器	8	A
标记寄存器	8	
程序状态指示寄存器	16	PSW
B寄存器	8	B
C寄存器	8	C
B对	16	B
D寄存器	8	D
E寄存器	8	E
D对	16	D
H寄存器	8	H
L寄存器	8	L
H对	16	H
栈指针	16	SP
程序计数器	16	PC
存贮器寄存器	8	M
端口 可变的		十六进制数

图 2-4 8080/8085机编程模型

六、调试和资料编写

设计功能的调试阶段，显然是指将研制出的程序作实际运行，监测差错、并重新修改设计。如一切先前的设计做得合适，则发现的差错应该是次要的而不是主要的。在程序研制这一阶段发现的主要差错，可能有很大的危害性，以致引起全部重新设计。这从时间和金钱观点来看，代价就会很高。

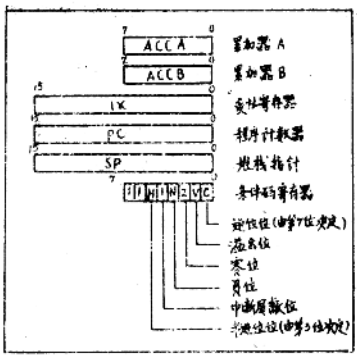


图 2-5 6800机编程模型

注意图2-1中的流程图，每次操作之后都有判定部分以便对先前的工作做逻辑性、周密性和准确度方面的检查，如发现任何差错，应立即改正，不应往后拖延，因为在设计阶段后期，要更改一个程序是困难的。

任何设计的最后阶段是编制程序文件和产生可操作指令文件。最后文件不仅是程序操作员使用的也是将来维护和修改程序的人员使用的。为了加强对程序的理解和说明如何进行操作。文件必须详细而又完整，包括维护、诊断说明、操作员使用的指令、流程图、程序列表文件。

本章的下一节论述更专门的设计方法。读者可以看到在讨论专门方法时，体现了本节介绍的编程方法次序。这里应再强调一下，应能给予读者以印象，知道编制程序时使用有条不紊，结构良好方法的重要性。这个重要概念，常常是直到研制的后期才为程序员所领会并接受。一些程序员从来就不接受有高水准的程序结构与有次序的方法作为‘较优的编程方法’。当然这类人在他们的编程同事当中，毕竟占少数

第二节 程序设计的方法

编制程序的最好同义语就是做计划，就是说，将需要完成什么样的操作和以什么样的次序来完成计划安排好。和做计划一样，程序设计可用很多不同方法进行。人们把他们的思想组织起来写出程序，方式各有不同，随人而异。写程序没有什么最好的方法。每人都一定要发展最适合他自己性格或工作技巧的方式方法。

编程工作本质上是一种有组织的任务，要学会编制程序和维护程序需要时间，干劲和一定的训练。本节讨论的一些方法和技巧是比较公认的，希望读者能将这些方法组合起来，为日后编程工作使用。

在开始讨论各种方法之前，作者有必要向读者指出他个人的偏好，因而读者可据而权衡这些材料。作者把自己看成是某种编程原理的提倡者，这种原理名为‘自顶向下分析法’，但用其它方法，如模块分析法和结构分析法加以补充。自顶向下分析法固有的定义明确，方法细致，并注意细节，这都是非常好的。同时，应该和结构分析编程法结合起来，按模块分析法体系一般化这个方法并使之尽可能的合理。这三种方法没有一种是十全十美的，因此取每种方法的长处，互为补充是最合适的办法。这三种方法有个共通的而又是编制高质量程序文件不可缺少的技巧，即绘制流程图。

一、流 程 图

绘制流程图也许是人所熟知而又最广泛公认的编程技术。它的应用普遍，用途也是多方

面的。流程图绘制技术也用于计算机编程以外很多领域中。因为流程图直观易读，程序员以外的人们普遍都认识并应用它。

大部分编程教材都提倡这种思想，认为绘流程图是程序设计的编写程序阶段使用的主要方法之一，并在正常的情况下，应在写出机器指令和代码之前，必须先将程序给出完整的流程图。但实际情况并不如此。大多数程序员不大重视流程图，认为它们碍事，常常在事后才绘制流程图，把它们拖后放到编程工作的资料编制阶段。这种编写程序方法浪费很多时间并使工作迷失方向。流程图和良好计划应走在机器编码和汇编过程之前。一般人脑子里不可能记住一个复杂程序中整个结构所必需的细节和资料。大部分程序员忽视误用流程图，其潜在的微妙原因在于大部分人进入编程工作时的方式方法。很显然人们学习编程工作时，不能够一开始就写复杂的程序，而写简单程序却可不需用直观辅助。因此在这阶段认为流程图总是以后的事。同样，大部分教师需要学生送程序时附入流程图来打分数。因而学生通常是在完成程序之后再绘流程图。这种习惯很早就形成，不容易改。只有某人在他以后的编程工作中，成为编程小组负责人，才重视研制流程图。

流程图在用脑力计划程序过程中是个直观的辅助工具。它们的优点通常不易觉察，直到面对复杂程序难以掌握时才会体会到。同样，除非人们被极端难解的程序困扰，流程图的一些缺点也就暴露不出来。对流程图的正确估价是一定要记住它们是用作辅助计划的。它们本身并不是要达到的目的。

流程图的优点

1．流程图清楚的表示出工作顺序，因而在调试过程中有助于定位出错处。

2．将设计方案分解成为小的易处理的任务时，流程图有很大帮助。

3．流程图易于阅读、理解，并能用于编程以外的其它领域。

4．流程图使用标准的框图、符号和广泛为人所接受的技巧。

流程图的缺点

1．流程图是一个象'鼠窝'一样的图，难于设计得清清楚楚。随着程序的复杂性增加，易读性就减少，在流程图到底包括多少操作细节和表达的清晰程度两者之间，程序设计员难以选择。

2．流程图不能用于硬件，也就是说它们对于定时、短路、脱线、输入输出数据结构及类似的有关问题的定位，没有什么帮助。

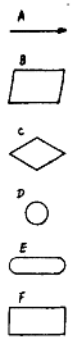
3．流程图不可能检测，检验流程图的唯一方法就是把它和最后编好的程序对比作检查。所谓调试流程图是一种心理活动，不能象将准备测试的电路那样连接起来测试。流程图的正确性除了用逻辑性来判断以外难以证实。

流程图的优点超过缺点。如果在研制的设计阶段使用得当，流程图对思维有巨大的帮助。但如把它们当作编制文件的事后添加物看待，缺点就显得大了。如程序最终已能工作，为什么要重做这些琐细的流程图编制工作呢？

概括地说，流程图的优点最好利用在设计功能方面。如这样使用，最后程序文件编制质量会高一些，而编制程序工作也会便当些。但是如想在流程图中包含过多细节就会使设计流程图几乎跟设计程序本身一样困难。

图2-6表示基本流程图符号，它们的名称和基本用法。这些符号都是标准的，并为便于画

图，将它们制成标准框板。以下各小节对每种符号作简略的介绍。



A．流线：表示执行顺序的流向。
B．输入/输出符：表示从系统中传出信息或系统接收信息。
C．判定符：表示逻辑比较部分（从这符号出来的流线方向，由比较后的答案'是'或'否'来决定。
D．连接符：（入口或出口）表示数据流或数据方向的变化。
E．终止符：和连接符类似，表示程序的开始，结束或中断处。
F．过程符：表示计算机的一个处理过程，通常和操纵功能有关。

图2-6　标准的流程图符号

二、流程图符号

输入/输出符：要弄清楚输入/输出符的功能，有个容易的办法，就是记住符号的形状。这符号不象过程符那样由一个矩形来表示。这表明它不是一种有严格定义的功能。也就是说，任何一个与输入或输出有关的操作或功能都可以用这个符号来代表。甚至亮指示灯，重绕磁带这一类的操作也可包括在内。有人也曾经使用过其它符号来表示输入/输出，使它更为形象化。但并不为人们普遍接受；图2-7给出了四个这样的符号。

连接符：连接符可以用来表示两种功能，并有四种不同的符号。它可用来表示一条转移指令或一处入口点。转移指令改变了操作的正常流向并让指令不按顺序执行，而在另一处继续执行。表示转移指令的连接符其流线总是从顶上或左方进入（见图2-8A）。

图2-8A中流线从顶上进入的符号，表示无条件转移。因为从流程图某一页出口的图形总是把流线划在底部或右方。条件转移总是和判定符号一起使用，而出口总是在右方；因此流线总是从左方进入，如图2-8B所示。

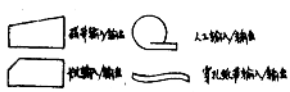


图2-7　其它输入/输出符号

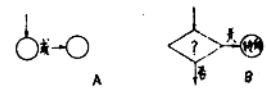


图2-8　（A）无条件转移　（B）条件转移

条件转移或无条件转移连接符代表着一条指令。但一个入口连接符则并不代表一条指令，它表示指令的标号。

标号包含在连接符内，对每个转移连接符，必定有个相应的入口连接符，它们是成对出现的符号，而连接符内的标号一定要一一对应。

入口符号要和从底部或右方出口的流线一起给出（见图 2 -9A）。同样，入口在程序的序列中间进入时，总是从这一页的顶部或左方进入。每个进入点一定要有唯一的标号，并且该标号只能出现一次。但分支点却不是唯一的，就是说对一个入口端点，可以有一个以上的分支，但是每个入口点都是唯一的。运用这些符号的例子见图 2 -9（B）。

现将使用连接符的规则总结如下：

1．连接符可以用于分支指令或入口点标号。

2．分支处有进入的流线，进入点处只有出去的流线。

3．无条件转移必定是序列中最后一个符号。

4．分支和入口是成对出现的。

5．必须从左边进入序列的中间。

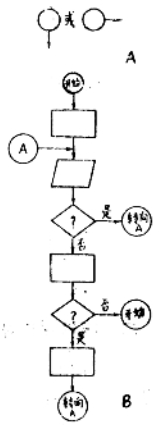


图 2 -9 （A）进入符号
（B）分支点

判定符：计算机之所以优于一台计算器，是因为它能根据在处理过程中产生的条件来改变操作顺序。这种能力来自它的提供定向判定或定向条件的指令，计算机利用将操作条件和原来存在的或规定的条件行进检查或比较后做出判定。比较的条件可以分为两类：

1．特定存储的内容（数值或文字的内容）

2．定量关系（<，>，=或≠）

行进比较，可以认为是作一次测试。然后计算机即可根据测试结果在两种操作路径中仅选择一种。当用符号表示这种测试或做出判断的过程，每个符号只容许做一次测试，每次测试结果只能在两种选择中取其一。换句话说，每次判断只能有一个入口，测试一种条件，并只有两种可能的出口，一个出口对应于答案是‘对’（即条件存在），另一个对应于‘不对’（即条件不存在）。这个符号是菱形的，在顶部，右方，底部都有流线。流线从顶部进入，并从右方或底部流出。菱形内部是用文字或数字表示的问题，说明正在进行比较的测试条件。（例如，是 0？，不是 0？，＜5？，＞7？；是‘M’？是‘0’？）如前所述，每次只能测试一个条件。注意其中非数字的字符放在引号内，而数字则仅列出必须的位数。结果如为‘是’通常使条件转移到次要或另一顺序上去（参看图 2 -10）。不一定要非如此做不可，但这是一种惯例。测试的结果如果是‘否’，通常就将正常的或主要的序列继续下去，正常没有改变的路径通常认为是主序列；而条件转移或经过更改的途径认为是异常或‘次要’序列。次要序列有时也称为‘子程序’。

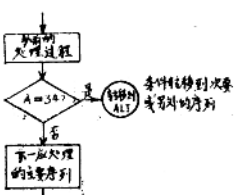


图 2 -10 条件转移到次要或另外的序列

在流程图中，次要序列可以在图画上任何认为方便的地方给出。次要序列图应放在什么地方没有什么成文规定。这里要讲清楚一点。所有子程序都认为是次要序列，但所有次要序列并不一定都是子程序。由于条件转移而发生的任一序列称为次要序列，而由于提高效率而转移的任何序列则称为子程序。它可同时

用于条件转移或无条件转移。（子程序在第六章中讨论）。

过程符：这个符号用代数形式或语句形式（在矩形框内）说明在程序中该处所完成的操作或处理。

终止符，终止符是程序或序列中的最后一条指令，通常是一条STOP或HALT指令。它也常常与作业例行程序结束联系在一起，这种例行程序在完成某项作业功能，如读一叠卡片或从纸带上接受一个数据块之后即暂告一段落。有时在例行程序开始时标记初始启动之处也可以看到终止符，它指示了在一个操作开始时设置的一些初始参数或初始条件。

三、自顶向下分析法

自顶向下分析法对程序设计工作与其说是一种方法不如说是一项原则。它就是本章前半部分已经说明过的方法。为了更清楚了解起见，有必要重提一下其基本要点。它进行的次序如下：

1．定义问题

2．进行设计

3．进行编码

4．进行校验和调试

5．编制文件资料

自顶向下分析法需要严格遵守准确的设计定义和功能说明。它的设计阶段以分割子任务并分解为更详细而又更简单的子功能为其特征。自顶向下分析法显著的优点是它的质量有可靠的保证。并在程序研制的每个阶段都可进行差错检查。

采用自顶向下分析法做编程工作的程序员可以毫不犹豫地说如果坚定地遵循这一原则，那末，每一个编好的程序第一次就能正确地运行。虽然在实际上这种百分之百的成功率会打折扣，但用这种方法能做到成功率大，调试时间少，这也是事实。

使用其它设计方法的程序员，他们用于调试和修改指令的时间要比用于做设计的时间多得多。自顶向下编程法使程序员不得不先行思考，然后编写程序。而这正是解决任何问题最好的原则。使用自顶向下分析法也很自然会导致模块化程序分析法和结构分析编程法技术相结合。

自顶向下分析法有个心理上的优势，就是它有引起人们确立程序和运行程序的初感。其方法是用假定的程序运行结果作为断点来检查程序操作到给定点的情况。使用能提供特殊响应给程序的模拟模块，好过实际完成子任务模块。也许使用自顶向下分析法的主要优点是，它减少了给定程序块或子功能在其设计、编码和调试时与整个系统不配合的可能性。

有了自顶向下分析系统，整个程序的各个研制级别或阶段，在设计继续向前进行之前，都可以检查其正确性。

自顶向下分析法的缺点不是十分明显的：

1．虽然自顶向下分析法原则上接受了模块化方法，得出的结果常常不是一种能运行于其它程序或能为程序库系统所采用的通用程序块。这因为它的设计原则是针对解决某一特殊要求的而不是解决一般性的要求的。

2．有些程序模块很难编写，特别是如它们要满足主程序几个不同部分的要求的话。

3．因为自顶向下分析法解法的专用性，它不能很好地利用现有程序或充分利用程序库。

4．自顶向下分析法的错误，通常早就在设计中被发现。如一项主要错误在检查和综合系统时漏掉，其影响将不堪设想，问题将波及整个系统而不是局限在某一个程序模块当中。

如要对自顶向下分析法的缺点说句概括的话，可以认为该方法过于专门．不好用来产生有一般功能的程序，程序的结果可能过于'精细'并互相牵连太大，以致象经过细调的机器，对外部世界这种平衡有时过于娇弱了，稍作调整就会全部崩溃。

四、模块化程序设计法

将程序和任务分成子任务的方法称为'模块化程序设计'。模块化程序设计的优点和缺点如下：

模块化程序设计的优点

1．如这些模块被写成有独立特性的模块，这些模块即可用于其它很多程序中，并便于用它们来建立程序库。同样，预先写好了的程序模块通常可以为现在需要而采用。

2．错误通常只局限于某一个模块，如有更动对整个系统的影响不大。

模块化程序设计的主要优点，也许就在于它能得到普遍使用的模块。这种普遍使用的模块，其形式尽可能是独立的、完整的。这种功能的独立性明显地便利了对程序设计的调试和检验。如把一点这种普遍原则和分割功能对采用的严格的自顶向下分析法相结合，将有助于消除两种方法的一些缺点。模块化系统的缺点如下：

模块化程序设计的缺点

1．因为模块化程序设计是以独立方式写的（通常的情况下整个程序是由不同的人写的），模块在整个系统中有不易配合的倾向。

2．由于模块是分开写的，产生操作功能的重复，浪费时间，也浪费内存。这种分开来编写的模块也增加了编制文件的费用和检验与调试模块方面的困难。

自顶向下分析法多少有点过于严格和专门，而模块化方法从设计标准看来又过于松弛，对解决专门任务显得太一般化了。作者相信取两种方法之长才是正确的方法。在定义和设计初期遵循自顶向下分析法，同时完成分割模块的工作，这样就克服模块化方法的主要缺陷，在编程结束时将模块互相联系起来并力图使其合为一体。

五、结构程序设计法

结构程序设计法由三种流程图结构组成，这三种结构满足所有程序的要求。三种结构可以嵌套和多次地迭代来取得要求的结果。图2-11表示三种这样的结构。

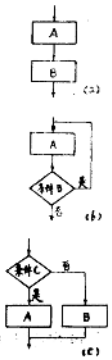
结构程序设计法的缺点

1．很多程序员在尝试求解时用此法会感到呆板，从而“禁锢”了他们的创造性。

2．结构程序设计不能用于高级语言。（这对微机程序员实际上是有利的。因为他通常用低级语言工作）。

所谓完成结构设计实质上是一次操练。它强迫做模块化工作的程序员将他写的程序去适应一种有系统有组织的格式。今天大多数程序员对结构程序设计还不熟悉，结果他们认为这是对自己写程序的风格的挫伤和限制。结构程序设计法最适用于编制大型、低级语言写的程

序，在这里检验和调试方面的软件研制成本占有重要的地位。结构程序设计法给不受拘束的模块化程序设计方法多少带来了约束和限制。模块化程序设计法有助于把自顶向下分析法的专用性普遍化。总的来说，我们推荐下述方法。用自顶向下的分析原则定义程序并对它的正确性经常检查。用结构法的形式构造生成模块，最后，只要合理的话就将模块普遍化以便日后应用。



(a) 定义和展开法则

先做A后做B（简单的按序完成）

在先做A后做B中，A可由任何其它功能块C来代替，它可分解为C=B（A）这样的形式，即功能A的结果由B来继续操作。

(b) 在B的条件下做A（一次迭代循环）

只要迭代操作产生的条件合于B，在改变A的功能的同时做A。

(c) 如C成立则做B（选择过程）一个 If Tnen Else 操作代替了功能块。这里条件C代替了功能的定义，这种情况下A和B的功能是完全确定的。

图 2-11　结构程序设计法的三种基本结构

问　题

1．在试图找出问题的解法之前，该做些什么工作？
2．什么是分割？
3．准确度和周密性有何区别？
4．编写程序时，我们内心应抱什么态度？
5．程序研制的整个解法顺序是什么？
6．初期思维图解有何价值？
7．什么是语言等级？
8．汇编语言和机器语言有何不同？
9．技术人员最可能使用的是什么等级的语言？
10．为什么程序设计模型有用？
11．为什么经常的文件资料编制工作如此重要？
12．程序编制通用的基本技巧是什么？
13．流程图的主要优点是什么？
14．流程图什么时候最有用？
15．流程图有什么主要缺点？
16．列出并标明常用的流程图符号。
17．计算机与计算器有何区别？
18．说明主要序列和次要序列之间有何不同？

19．自顶向下分析法最显著的特点是什么？

20．自顶向下分析法的主要缺点是什么？

21．程序的模块法编程概念和自顶向下分析法有何不同？

22．试举出结构程序编制法的三种结构。

23．在文件编制过程中，结构工作有何助益？

24．结构程序编制法的主要缺点是什么？

25．你认为为什么作者要推荐将三种编程方法综合？你认为他对吗？

专供技巧训练的练习

1．将某人横过交通繁忙的十字路口的实际过程绘成流程图，其情况如下：

A．路口有可以'步行'和'不准步行'的交通指示灯。

B．开始时该行人在十字路口，交通正在进行。

C．确信包含下列两项：

①问题有良好定义。

②有一张功能程序块表。

D．做下列两项：

①不要认为'步行'过程是预定的。

②工作要完全，并且心目中要'追踪'该操作看看这位行人是不是在这个过程中不发生意外。（他是否等候别人管制或启闭交通灯？）

2．使用适当符号及步骤绘出下列例行程序的流程图。

①从电传打字机输入数据。

②检查数据是否有效。

③如数据有效，把数据移到输出存储单元。

④把数据送到打印机。

⑤如数据无效则输入更多的数据。

⑥数据打印以后，重复整个过程。该过程是连续的，不终止。

3．将下列的条件陈述，绘成流程图：如条件A存，则进行功能G，然后进行H；如条件A不存在，检查条件B是否存在。如B存在，进行功能F，然后进行H；如条件B不存在，进行功能H。

4．考虑下列流程图。将可能要打印出的数字写出来。

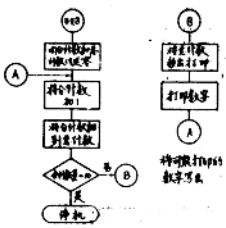


图 2-12