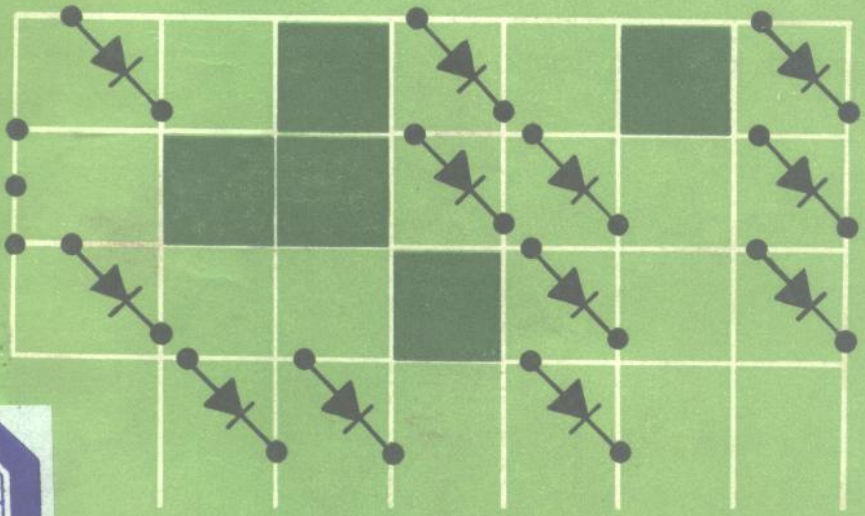
计算机结构和组织教学手册

复旦大学计算机科学系 编译



上海科学技术文献出版社

计算机结构和组织教学手册

复旦大学计算机科学系 编

上海科学技术文献出版社

计算机结构和组织教学手册
复旦大学计算机科学系编译
*
上海科学技术文献出版社出版
（上海市武康路2号）
新华书店上海发行所发行
上海新华印刷厂印刷
*
开本 787×1092 1/32　印张 4.5　字数 10
1984年8月第1版　1984年8月第1次
印数：1—44,000
书号：15192·320　定价：0.57元
《科技新书目》77-179

前　言

本手册主要是根据约翰P·海斯所著《计算机结构和组织》(译本由上海科学技术文献出版社1982年1月出版)的配套书《计算机结构和组织教学手册》"Instructor's Manual to Accompany Computer Architecture and Organization" (McGraw-Hill, Inc, 1978年版)编译的。

本手册按照《计算机结构和组织》一书分为六章。第一章计算机的演变；第二章设计方法学；第三章处理机设计；第四章控制部件设计；第五章存贮器组织；第六章系统组织。

我们在编译过程中注意保留《手册》的原有特色，对不易理解的解答作了局部的补充和充实。

本手册中部分习题可以作为计算机结构和组织等有关课程的课外习题。许多富有启发的思考讨论题可供有关教师教学参考。对于报考研究生的学生和有关自学者，它也是一本有用的参考书。

本手册第一、二、四章由彭澄廉编译，第三章由朱晓强编译，第五、六章由徐研人编译。最后由彭澄廉负责总审。李应华做了大量校订工作。

由于我们教学实践尚少并限于水平，编译中错误与不妥之处，恳请读者指正。

编　者

1983年9月

目　录

第一章　计算机的演变

1.1　设计一个用 2 来除一元自然数 n 的 Turing 机。n 是按下列格式存放在另外的空白带上

$$\cdots bbA\underbrace{11\cdots11}_{n\text{个}1}Bbb\cdots$$

其中 b 代表空白，而 A 和 B 为分割符。机器应计算出 $n/2$ 的整数部分 $\lfloor n/2 \rfloor$，并且按上面格式写出结果。带上符号只允许为 b, A, B 和 1。用 Turing 机程序描述你的机器，并附以适当注释。

答：所需除以 2 的运算可用如下策略进行：Turing 机自左向右扫描带的内容，按序删除 A 以及 n 个 1。每当第二次删除 1 时，则在带上内容的右端写一个新的 1。这样当所有的 n 个 1 被删除后，带上内容如下：

$$\cdots bbB\underbrace{11\cdots11}_{\lfloor n/2 \rfloor\text{个}1}bb\cdots$$

余下的手续是把其中的 B 换成 A，以及把 B 插入一列 1 的右端。图 1-1 列出了实行此策略的程序。这里 S_0—S_7 是该 Turing 机的 7 种状态。假定其初态为 S_0，而且读写头恰在分隔符 A 上。

有关 Turing 机各种指令的含义参见《计算机结构和组织》，([美]约翰 P·海斯著，上海科学技术文献出版社出版，第 4～6 页，1982 年)。

S_0	A	b	S_0	删除 A
S_0	b	R	S_1	右移一格
S_1	1	b	S_2	删除一个1
S_2	b	S	S_3	右移一格
S_3	1	b	S_3	删去第二个1
S_3	b	R	S_4	移动到带内容的右端
S_4	1	R	S_4	
S_4	B	R	S_4	
S_4	b	1	S_5	在带右端记入一个1
S_5	1	L	S_5	返回到带内容的左边
S_5	B	L	S_5	
S_5	b	R	S_1	已达左边，重新进行删除1的过程
S_1	B	A	S_6	所有的1已删除，把 B 换成 A
S_3	B	A	S_6	
S_6	A	R	S_6	移动到带内容的右端
S_6	1	R	S_6	
S_6	b	B	S_5	在带右端记入 B
S_6	B	H	S_6	停止

图 1-1　一元数除以 2 的 Turing 机程序

1.2 (a)证明一个有限状态的机器，可以用来相加两个任意大的二进制数，并给出其逻辑设计。

(b) 一个有限状态机器能否对两个任意大的二进制数作相乘？证明你的结论。

答：(a) 令 N_1 与 N_2 为两个任意大的二进制数。我们可以很容易地设计一个有限状态机 M，它可把 N_1 和 N_2 串行地(一位一位地)相加。N_1 与 N_2 从最低位开始顺次进入 M。M 产生两个当前输入位的和，而进位则由前面的加法产生。结果的和位由 M 输出，而新的进位则存贮于 M。很明显，M 在任何时刻仅需存贮一位信息，即由前次一位加法所产生的进位。因而 M 仅需有二种状态，其作用可用以下状态表来描述：

		输入 $N_{1,i}N_{2,i}$			
		00	01	10	11
状态	S_0	S_0, 0	S_0, 1	S_0, 1	S_1, 0
	S_1	S_0, 1	S_1, 0	S_1, 0	S_1, 1

M 被称为串行加法器，是一个很有用的设计元件。图 1-2 给出了串行加法器的标准逻辑设计。

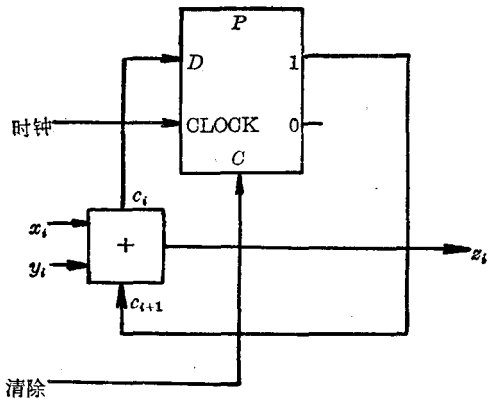


图 1-2　串行二进制加法器

(*b*) 不行。用反证法，假设少于 N 个状态的有限状态机 M 可以对任意大的两个二进制数作乘法的话。因为 M 机的输入和输出线数是有限的，一般地说，M 必须串行地处理输入数和输出数。不妨假定输入数的接收和输出数的发送都是一位一位进行的，就象本习题(*a*)中所做的那样。设要求用 M 做 2^n 与 2^n 的乘法，则结果应为 $2n$ 个 0 后面一个 1 的序列，也即 2^{2n} 的二进制形式。因而在全部输入信息接收以后，M 必须输出一个有 $n-1$ 个 0 而后面一个 1 的序列。M 为了产生这 n 个数字，必须进行 n 次状态转换。这时状态转换的输入条件可以看成全部为

0,不再变化。假设 M 的状态数小于 $n-1$，则在产生 $n-1$ 个 0 的 $n-1$ 次状态转换中，M 的状态至少要重复一次。但从第一次重复开始，M 必须进入一个闭合回路，并无限循环下去，在其中产生的输出总是 0。因此乘积中最后一位 1 总是产生不出来。所以问题中相乘的数对 M 来说是太大了。

加法与乘法的关键区别在于，将两个 n 位数相乘时，乘积的长度超过 n 的量是个随 n 增长的数，而同样情况下求和时，和的长度最多为 $n+1$。

1.3 计算机程序中两条指令 I_1 和 I_2，如果其执行次序对所有输入数据组来说，均不影响其程序的最后结果，则称它们为"并行"的。这个意思即为 I_1 和 I_2 可以并行地在各自的处理机中执行。并行指令的识别在高速并行处理的系统设计中是十分重要的。试证明，对任意程序，要确定任何两条指令是否并行的问题是不可解的。

答：令 P 为任意一程序(P 也可解释为一 Turing 机)。我们将修改 P，加入几条包括 I_1 和 I_2 的新指令成为程序 P′。结果是，仅当 P 是不能停止时，I_1 与 I_2 才能在 P′ 中是并行的。

首先，我们将 P 中所有停机指令 HALT 集中于一条标号为 H 的停机语句。这可用在适当地方插入 go to H 语句来达到。那么 P 可认为有以下一般形式：

```
START
...
H: HALT;
...
END
```

令 A，B 及 C 为未出现于 P 中的变元。将该 START 语句替换为下列指令序列：

START

B←0;

I_1: A←A+B;

I_2: C←A;

if B≠0 then HALT;

并将语句 H: HALT 替换为:

H: B←1;

go to I_1;

将修改过的程序称为 P′。如果 P(因而也有 P′)不能停机的话，那么新的 H 语句也绝不能执行到。从而 I_1 等价于 A←A。这意味着 I_1 与 I_2 可以并行地执行，因为 I_1 实际上是一条"空操作"指令。然而，若 P 可以停机的话，P′ 必须执行 H，结果 I_1 成为 A←A+1。这将意味着 I_1 与 I_2 不可以并行地执行，因为 I_1 的结果将影响 I_2 的结果。因此，仅当原始程序 P 不能停机时，我们才能断定 I_1 与 I_2 是并行的指令。

假设存在算法 X，它能确定任意程序中的指令是否能并行地执行。如将 X 用于 P′，它就可以确定 I_1 与 I_2 是否可并行地执行。因而解决了程序 P 的停机问题。但是对任意程序的停机问题是不可判定的，所以 X 是不可能存在的。

本解答的原始叙述和另一种证明方法可在以下文献中查到: A. J. Bernstein: "Analysis of programs for parallel processing," IEEE Trans. Electronic Computers, vol. EC-15, pp. 757～763, Oct. 1966.

1.4 设 A_1, A_2, A_3 是在已知的计算机上用来解具有输入规模为 n 的同一问题 $Q(n)$ 的三种不同算法。并设 A_1, A_2 和 A_3 的精确的时间复杂性分别为 $100n$, $10^{-3}n^{10}$ 和 n^2。

(*a*) 假定 n 为正整数，求对每种算法最合适的 n 的数值范

围。

(*b*) 哪一种算法对 $Q(1)$, $Q(10)$ 和 $Q(100)$ 提供最快的解？

答：对于一个输入规模为 n 的问题，如果算法 A_I 的时间复杂性小于 A_J 的时间复杂性的话，就认为对 n 值来说，算法 A_I 比 A_J 好。为确定成立上述关系的最小 n 值，令 A_I 及 A_J 的时间复杂性相等，对 n 解方程。例如，比较 A_1 与 A_2，解方程 $100n = 10^{-3} \cdot n^{10}$ 得到：$n = \sqrt[9]{100,000} = 3.59$。这意味着，当 $n \leqslant 3$ 时，A_2 较好，而对 $n \geqslant 4$ 时，A_1 较好。

(*a*) 当 $n=1$ 及 2 时，A_2 最好；当 $3 \leqslant n \leqslant 9$ 时 A_3 最好；当 $n \geqslant 10$ 时 A_1 最好。图 1-3 列出了对于 12 以下的 n 值的算法复杂性。

n	A_1: $100n$	A_2: $10^{-3}n^{10}$	A_3: 2^n
1	100	0.001	2
2	200	1.024	4
3	300	59.049	8
4	400	1048.576	16
5	500	9765.625	32
6	600	$6.047\ 10^4$	64
7	700	$2.825\ 10^5$	128
8	800	$1.074\ 10^6$	256
9	900	$3.487\ 10^6$	512
10	1000	$1.000\ 10^7$	1024
11	1100	$2.594\ 10^7$	2048
12	1200	$6.192\ 10^7$	4096

图 1-3　对于小的 n 值三种算法的时间复杂性

(*b*) 对 $Q(1)$, $Q(10)$ 以及 $Q(100)$ 的最快解法分别为 A_2, A_1 及 A_1。

1.5　给下列各计算机配上最适当的描述：

——STARAN IV	(*a*) 平行处理机。
——Von Neumann's machine	(*b*) 流水线处理机。
——ILLIAC IV	(*c*) 相联处理机。
——TI ASC	(*d*) 多功能处理机。
——PDP 11	(*e*) 单处理机。
——CDC 7600	(*f*) 多处理机。
——B 7700	(*g*) 小型计算机。

答：描述应按以下顺序排列：*c*, *e*, *a*, *b*, *g*, *d*, *f*。

1.6 给下列计算机配上最适当的描述：

——ATLAS	(*a*) 早期的继电器计算机。
——B 5000	(*b*) 带有流水线控制部件的第一批计算机之一。
——ENIAC	
——IBM 360	(*c*) 第一个集成电路计算机。
——STRETCH	(*d*) 带有页式存贮器的第一个系统。
	(*e*) 第一个浮点计算机。
	(*f*) 早期的真空管计算机。
	(*g*) 首批可兼容计算机系列之一。
	(*h*) 首批堆栈计算机系列之一。

答：最适当的描述顺序如下：*d*, *h*, *f*, *g*, *b*。

1.7 考虑计算从1到200,000的整数，算至十进制19位的自然对数表的问题。这是一个由Piony小组于1795年左右手工计算的任务。选择你所熟悉的任何市售计算机系统，估算计算和打印该表所需时间，并对用于你的估算的所有参数作出定义。

答：有多种不同的级数展开式可用于计算 $\ln x$。例如在"标准数学用表"(Standard Mathematical Tables, 14th Ed., Cleve-land, Ohio, The Chemical Rubber Co., 1965.)中就列

出了8种。使用这些级数必须十分当心，因为有许多级数收敛得极慢。为达到要求的精度，就需要很大数量的项数。许多学生选择如下型式的级数：

$$\ln x=\frac{x-1}{x}+\frac{1}{2}\left(\frac{x-1}{x}\right)^2+\frac{1}{3}\left(\frac{x-1}{x}\right)^3+\cdots \qquad (1)$$

上式对 $x>\frac{1}{2}$ 成立。为使取其 $k-1$ 项而达到19位小数的精度，必须成立：

$$\frac{1}{k}\left(\frac{x-1}{x}\right)^k<10^{-19}$$

k 是很大的，甚至对给定范围内很小的 x 也如此。例如当 $x=10,000$ 时，$k\approx 3.1\times 10^5$，而在最坏情况，当 $x=200,000$ 时，k 则超过了五百万！

“标准数学用表”中一个收敛相当快的级数是：

$$\ln(x+1)-\ln(x-1)=2(1/x+1/3x^3+1/5x^5+\cdots) \qquad (2)$$

这是一个递归级数，$\ln x$ 要用 $\ln(x-2)$ 来计算。正是因为 $\ln(x-2)$ 必须先计算，该级数非常适合于计算所要求的表格。为了取公式(2)的前 $(k-1)/2$ 项而达到所要求的精度，我们有：

$$2(1/kx^k)<10^{-19}$$

为满足此不等式，必须是当 x 越小时，k 越大才行。$\ln 1$ 和 $\ln 2$ 是递归计算的初始常数，必须单独计算。显然，$\ln 1=0$，而 $\ln 2$ 则可用公式 (1) 这类非递归级数来计算。因而 $\ln 3$ 对公式 (2) 来说是最坏情况了。容易证明，当 $x=3$ 时，$k\approx 38$，故需要取级数的大约 19 项来计算 $\ln 3$ 以达到 19 位精度。但对 $x=100$，仅需 5 项就够了。其实平均取级数的 3 项就够了。

考虑公式 (2) 的前 3 项可知，平均计算一次 $\ln x$ 大约需要七次乘除法和四次加法。为达到该必要的精度，大多数计算机需要采用多精度算术计算。假定我们使用一台相当慢的计算

机，每次乘除法计算需要100 μs，而加法要10 μs。那么 $\ln x$ 的一次计算平均要用740 μs 时间，再算入登录指令执行时间，就算是1 ms 吧，则全部表格可以在200秒或者说在3.3分钟时间内计算完成。

再考虑打印时间。一个比较快的行式打印机，每分钟要打印1000行之多，每行字符数可达132。$\ln x$ 包括22个字符，所以每行可以打印5个结果。全部的200,000/5=40,000行，要用40分钟打完。所以，如果计算和打印同时进行的话，则完成该任务的全部时间是40分钟。很明显，问题在于IO的限制，而不在于CPU的限制。但如果用象公式(1)这样的慢收敛级数的话，情形就反过来了。(有一个50个大学生的班级，做这个问题，给出了五花八门的答案。从几分之一秒到10^{14}年都有!)

1.8 讨论将程序和数据存放在同一存贮器的优缺点(存贮程序的概念)。在什么情况需要将程序和数据存放在各自的存贮器中?

答：(*a*) 优点：指令在执行过程中可以修改，因而可以编写可以自我修改的程序。对于存取指令和数据，仅需一组寻址电路。由于不必要预先区分指令和数据，使得实现存贮管理软件容易进行。因为程序和数据可以分配于任何可用空间，故可以更有效地利用存贮空间。

(*b*) 缺点：自修改程序是难以编制、调试和使用的。事实上这类程序是靠不住的。当程序和数据存在一起的时候，是很难保护程序的。必须十分当心，不要因疏忽而把指令当数据修改。另一方面，要保证不要试图把数据当指令来执行。确定指令和数据字长的尺度是不同的。如果使用分离的程序的和数据的存贮器，则每个存贮器的字长都可以最优化。而当对指令和数据使用单一内存的时候，这是不可能做到的。分离的存贮器，

允许并行地取指令和数据，这可能导致处理速度的提高。

(c) 分离的程序的和数据的存贮器，可以用来消除上面提到的单个存贮器的缺点。它普遍地应用于基于微处理器的系统中，这里总是使用同样的软件，把程序存于只读存贮器 ROMs 中，而数据则存于读写存贮器 RAMs 中。这避免了错把数据和指令换位的疏忽。这也在电源发生故障时保护了程序，因为 ROMs 是固定不变的，而多数 RAMs 则是易变的。分离的存贮器，允许程序和数据同时存取。这意味着程序和数据的存取方法可以独立地优化，从而改进系统性能。例如，每个存贮器可有不同的字长以及不同的地址间隔方案。在诸如实时数字信号处理的应用中，数据可能独立于 CPU 而变化，这里分离程序和数据的存贮器也是期望的。

1.9 考虑图 1-4 给出的 IAS 的 11 条数据传送和转移指令的集合 S，按以下意义试鉴别 S 中称为"多余"的指令：I 为多余的，即若由 I 所实现的全部操作均可由除 I 外的任何指令编制的程序来实现。

答：并无已知的算法来鉴别多余或构造与之等价的程序。事实上，可以证明，对任意指令集的多余性判别的一般问题是不可判定的。下面的解答是基于启发式的或直观的方法。

S 的前 3 条数据传送指令是非多余的。AC←MQ 以及 MQ←M(X)是非多余的，这是因为它们分别是仅有的以 MQ 做为输入或输出操作数的指令。M(X)←AC 是仅有的以 M(X)做为输出操作数的指令。余下的 8 条 S 中的指令是多余的。图 1-5 对 4 条有代表性的指令列出了其等价程序。余下的 4 条指令可类似于图 1-5 中的一条指令而得出其等价程序。

此问题可以扩展为考虑整个 21 条 IAS 指令的多余性。这时仅有一条指令 M(X)←AC 看来是非多余的。模拟乘法、除

指令类型	速　记　法	描　　述
数据传送	AC←MQ	将寄存器 MQ 的内容送到累加器 AC 中
	MQ←M(X)	将存贮单元 X 的内容送到 MQ 中
	M(X)←AC	将累加器内容送到存贮单元 X 中
	AC←M(X)	将 M(X)送到累加器中
	AC←−M(X)	将 −M(X)送到累加器中
	AC←|M(X)|	将 M(X)的绝对值送到累加器中
	AC←−|M(X)|	将 −|M(X)|送到累加器中
无条件转移	go to M(X, 0:19)	将 M(X)的左半作为下一条指令
	go to M(X, 20:39)	将 M(X)的右半取作为下一条指令
条件转移	if AC≥0 then go to M(X, 0:19)	若累加器中的数为非负数，则取 M(X)的左半作为下一条指令
	if AC≥0 then go to M(X, 20:39)	若累加器中的数为非负数，则取 M(X)的右半作为下一条指令
算术运算	AC←AC+M(X)	将 M(X)加于 AC; 且其结果放入 AC
	AC←AC+|M(X)|	将 |M(X)|加于 AC; 且其结果放入 AC
	AC←AC−M(X)	将 M(X)从 AC 中减去; 其结果放入 AC
	AC←AC−|M(X)|	将|M(X)|从 AC 中减去; 其结果放入 AC
	AC, MQ ←MQ×M(X)	MQ 乘以 M(X); 其结果高位放在 AC, 低位放在 MQ
	MQ, AC←AC÷M(X)	AC 被 M(X)除; 商在 MQ, 余数在 AC
	AC←AC×2	累加器乘以 2, 即累加器向左移一位
	AC←AC÷2	累加器除以 2, 即累加器向右移一位
地址修改	M(X, 8:19) ←AC(28:39)	AC 的最右 12 位替代 M(X)中的左指令地址场
	M(X, 28:39) ←AC(28:39)	AC 的最右 12 位替代 M(X)中的右指令地址场

图 1-4　IAS 计算机的指令系统

指　　令	等　　价　　程　　序
AC←M(X)	AC←−M(X) if AC≥0 then go to M(Y, 0:19) AC←|M(X)| go to M(Y, 20:39) M(Y, 0:19):AC←−|M(X)| M(Y, 20:39):…
AC←|M(X)|	AC←M(X) if AC≥0 then go to M(Y, 0:19) AC←−M(X) M(Y, 0:19):…
go to M(X, 0:19)	M(Y)←AC　存 AC. AC←|M(Y)|　使 AC≥0. if AC≥0 then go to M(X−1, 20:39) … M(X−1, 20:39):AC←M(Y)　恢复 AC. M(X, 0:19):…　转移地址
if AC≥0 then go to M(X, 0:19)	if AC≥0 then go to M(Y, 20:39) … M(Y, 20:39):go to M(X, 0:19)

图 1-5　对某些 IAS 指令的等价程序

法和地址修改指令的程序是相当长的。其它的指令的等价程序则类似于图 1-5 中的那些程序。

1.10　假设 10 个非负数组成的一个向量，存放在 IAS 计算机存贮器中从单元 100 开始的连续单元里。试用图 1-4 中的指令系统写出求该数组中最大数的地址的程序。若有几个单元都