计算机资料之一

DJS—6机

算法语言讲义补充

（关于操作使用）

北京工业大学计算站

1975·6

毛 主 席 语 录

人类总得不断地总结经验，有所发现，有所发明，有所创造，有所前进。

DJS—6 机算法语言操作使用补充

在工作中，许多同志已经利用 DJS—6 机算法语言成功地解决了大量实际问题，总结了许多宝贵的经验。我们热诚地希望兄弟单位和广大用户把宝贵经验介绍给我们，为了抛砖引玉，我们先就下面六个方面：

一、关于编译过程的一些问题；

二、关于区域溢出；

三、关于试算；

四、还要注意的几个问题；

五、控制台上的一些操作；

六、关于代码语句的修改指令

作一些粗浅的介绍，供同志们参考。

需要说明的是：我们自己实践得还很少，对机器和编译程序了解不够，调程序的各项措施也应用得很不灵活，这里只是作为我校算法语言讲义的一个补充，谈谈我们的一些肤浅的认识，其中肯定有许多错误与不妥之处，欢迎同志们批评指正。

§ 1 关 于 编 译

一、电传打出出错信息，编译没通过，怎么办？

如果错误不多，应争取在本次上机把错误改正，加以试算。因此，当电传打出出错信息后，应利用电传命令 *'OT'*，让电传把出错部位的源程序输出（一般让电传输出两行：出错部位的上一行和本行），想好改正的办法，然后用电传修改源程序（修改时，一般由后面的错误开始逐个往前修改，以免前面修改中增、删的行数对后面的行号有影响，见讲义 121 页）。

注意，修改完后，应再打 *'BE'* 命令，*'BE'* 通过，才能试算。

二、如果打出一大堆出错信息，怎么办？

如果出错信息很多，估计不可能在本次上机改正全部错误，则一般应首先立足于把出现错误的原因找出，而不应急于改正其中的某几个错误。因此，也要采用 *'OT'* 命令来对源程序进行局部输出。

如果错误信息很多，一般并不需要把每条出错信息附近的页行输出，这是因为估计其中有不少错误信息是由于连锁反应而产生的（见讲义 130—131 页），所以通常把电传所打的出

错信息按其部位分成比较密集的几段，把每段的第一个出错部位附近的源程序进行输出。

在错误较多的情况下，应争取时间把要输出的源程序都输出，至于细致的分析，可以放在下机之后。并且，应在下次上机前把纸带改正或穿好修改带，以免下次上机时用电传修改占掉过多的机时。

三、如果出错信息老打个没完或者指出的出错页行在本源程序中根本没有，怎么办?

这种情况很可能是由于连锁反应或者由于前面的错误引起了混乱而产生的。所以，出现了这种情况应立即先按单条键（*DT*）和总清键（*ZJ*），使之停机，然后按地址 053100 启动，再把已指出的出错部位分段，再利用 *'OT'* 命令局部输出源程序，当我们把前面的错误改正后再编译时，通常电传就不会再没完没了地打出错信息了。

产生这种现象还可能由于上一个题目太大冲掉了编译程序或者前面的错误破坏了编译程序，考虑到这种可能，那末应在按单条和总清键后，先按 000004 启动把编译程序从鼓中重新调入内存（停机地址是057403），再按前面所指出的步骤进行操作。

四、编译时电传指出有错，但输出源程序看却没错，这是怎么回事?

这有几种可能：

1. 由于前面的错引起了编译程序的误解而打出了这个错。如果是这个原因，则当我们把前面的错改正后，此处就不会打错了。
2. 由于我们对算法语言了解得不深入，因而有错看不出来。
3. 电传偶尔有误打的现象，可能把出错的页号或行号打错了。
4. 错误出现在前面，而我们输出源程序时没把前面有错的地方输出，因而没有发现。

例如：源程序中如下两行：

$$A:=B+C;$$
$$D[1]:=E+F;$$

穿孔时穿成了：

$$A:=B+C*$$
$$D[1]:=E+F;$$

显然，我们一看便知第一行结束处的分号错误地穿成乘号了，但编译程序却以为要求把表达式 $B+C*D[1]$ 的值赋给 A，这是符合语法的，而在表达式 $B+C*D[1]$ 之后又遇赋值号，则不允许，因此出错部位是指在第二行的赋值号处。如果我们不把第一行的源程序输出来，只输出了第二行，那末就看不出问题所在了。

五、修改后再编译，发现又有错，这次再修改时，上次的修改还有效吗?

若在这次修改前，又把源程序重新输入了，则上次的电传修改无效。若没有将源程序重新输入，则上次的修改仍有效。所以，若未将纸带重新输入，则在数页、行号时，不能以源程序的页、行号为准，而应以修改后的页、行号为准，为慎重起见，应把要改的页、行先输出来看一下，以判断它是否正是我们要改的页、行。

六、有的部位在上次编译时并未指出有错，而当我们改正了一些错误又重新编译时却指出这个部位有错了，这是怎么回事?

这是正常的，因为第一次编译时，在语法检查阶段发现了错误就跳过了一段源程序从一

个新的起点接着往下作语法检查（见讲义131页）。被跳过的这一段源程序中的错误这次就发现不了。当我们把前面的错误改正后再进行编译，这段源程序不再被跳过，于是，其中的错误就被指出来了。

七、电传打出错信息时，出错部位怎么不是顺着排下来的，而有忽前忽后的现象？

这是正常的。因为编译程序对源程序一共进行三次扫描(注)，每次都从头看起，而每次都各自作了一些语法检查的工作。第一次扫描时发现的错误以 ER 形式打出，只有 $ER3$ 所指的出错部位有可能往前跳，而其他都是顺着排下来的，第二次扫描的出错信息以 $CUQU$ 形式打出，因为这次扫描又是从头开始的，所以刚进入这次扫描时打出的出错部位可能跳到前面去了，第三次扫描的出错信息以 CU 形式打出，这次扫描也是从头开始，所以刚进入这次扫描时打出的出错部位又可能跳到前面去了。因此，除了以ER3形式打出的出错信息外，正常的情况下只可能有两次出错部位往前跳，而且每跳一次出错信息的形式也有所改变。否则就是不正常的，说明机器中发生了混乱，应按照三中所述的办法处理。

八、一个程序在机上经过一段调试后想暂时下机检查，先调另一个程序，那么原来的程序如何设法保存在机器中？

可以在下机前用'YJ'命令把原来的程序记入鼓内，等到又调这个程序时用命令'YD'把该程序读入内存。

这样的好处是可以较充分地利用机时。并且，如果在打 *'YJ'* 前程序是作了多次电传修改的，那么将来打'YD'时读入的程序也是经过电传修改了的，因而比重新输入并作电传修改既省事又省时。

九、经过 *'BE'* 后，电传指出有错，当我们用电传命令 *'OT'* 让机器局部输出源程序时却输不出来或者输出的内容与纸带根本不符，这是什么原因，可采取什么措施？

这种情况的产生往往是由于题目比较大，在 *'BE'* 过程中产生的中间语言侵占了源程序的存放区域，于是输不出或输出的不对，或者源程序较长，一直存放到037400之后，而在 *'BE'* 的第二、三次扫描时037400之后的单元又作别用了，于是037400之后的源程序输不出来。

事实上，只要中间语言没有赶上源程序，并且中间语言没超过037400的范围，那么对于产生目标并无影响。但是在前述两种情况下，要输出源程序或修改源程序却不行了。为此，对于较大的题目，可以在输入源程序之后马上用 *'YJ'* 记入鼓内，*'BE'* 后发现有错要输出源程序或修改时，先用 *'YD'* 把源程序调入内存，就可正确地输出或修改了。

十、有的时候，算上一个题目时机器还是好好的，非常正常，可是当我们算完这个题，再计算另一个题时，却发现机器很不正常，这是怎么回事？

有的同志往往以为这是机器不稳造成的。当然也不排除这种可能性，但往往这不是机器的毛病，而是因为我们前面计算的题目较大，它的数组元素把内存中存放编译程序的单元侵占了。于是，当我们计算下一个题目时，由于编译程序不全，就出现了很不正常的情况。因此，算完一个大题目，再接着算下一个题时，必须把编译程序送入内存（用 *'DG'* 命令，或者直接启动000004，或者直接送编译带），才能正确地计算下一个题。当然，如果前面算的题不大，也没出现混乱，那末，算下一个题时，只要直接按一下启动键就行了。

十一、 *'BE'* 后打出 *LESS*，怎么办？

首先，打 *LESS* 是什么原因？

编译程序对源程序共进行三次扫描，第一次扫描时将源程序加工成为中间语言，中间语言以一个全零单元作为结束标志，第二、三次实际上是对中间语言进行扫描而不是对源语言进行的。第二次扫描是用状态矩阵法对中间语言进行语法检查，当它发现遇到的是全零单元时，则认为源程序中的一切字符均已扫视完毕，若此时状态表也正好退完，则进入第三次扫描，若此时状态表未退完，则编译程序认为源程序字符有缺，于是让电传打 *LESS*。

因此打 *LESS* 也没什么了不起，无非是源程序中语法有错致使字符与状态不匹配或者由于某种原因致使中间语言区内某处出现了一个全零单元罢了，把致使电传打 *LESS* 的因素消除就行了。

其次，电传打了 *LESS*，我们怎么办？

打出 *LESS* 后，机器已经停机，所以要再打入电传命令前，必须先按地址 053100 启动，否则不能打入电传命令。

电传打 *LESS*，往往是由于前面的错误引起的，因此可以照常根据前面部分已打出的出错信息来修改源程序，改正后再编译，一般情况下就不会再打 *LESS* 了。特别是，当把致使电传打出 *LESS* 的那个错改正后，就不会打 *LESS* 了。致使电传打出 *LESS* 的那个错误的出错信息必然打在 *LESS* 之前（它并不一定是紧挨着 *LESS* 的那个信息，因为紧挨着的那个出错信息本身可能是由于连锁反应而打出的）。最后，有的编译带，关于 *INPUTR* 的处理有错，当源程序中用到 *INPUTR* 时，*'BE'* 后就会打 *LESS*。只要把 *INPUTR* 改为 *INPUT*，*LESS* 就不打了。如果源程序中有许多个 *INPUTR*，修改太麻烦，也可用启动 075000 的办法输入有关 *INPUTR* 的修改带，利用它，就可不必改动源程序中的 *INPUTR* 了。

十二、 有的标识符明明前面曾对它作了一次说明，可是在 *'BE'* 时却又打出了一大堆 *ER3*，这是怎么回事？

一般来说，在说明部分中打出 *ER3*，那是由于对该标识符说明重复而打出的，在语句部分中打出 *ER3*，那是由于用到了前面未曾说明的标识符而打出的。

常见的情况是：在语句部分每当遇到这个标识符就打一次 *ER3*，于是打了一大堆 *ER3*。这往往是由于前面对该标识符所作的说明穿孔或输入有误而引起的。例如，*'REAL'A,B*；在穿孔时，逗号前忘了穿符号键，则输入后变成 *'REAL'ANB*；于是在下面凡遇标识符 *A* 和 *B* 机器皆认为未作说明，于是打出了一大堆 *ER3*。

十三、 引用整除运算"//"时，时常打错，这是什么原因？

这里有两个原因，第一是由于本语言规定//号两边必须都是整型量，违反这一规定时就要打出出错信息。

第二，若 *A*, *B*, *C*, 都是整型量，表达式中出现 *A***B*//*C* 时也要打错，这是因为编译程序处理时，不论两个量是什么类型，经过乘、除、幂运算后都变成实型，因此 *A***B* 为实型量，再做整除就要打错。改正的办法是：当 *A***B*>0 时把表达式 *A***B*//*C* 改为 *ENTIER*(*A***B*+0.5)//*C*，当 *A***B*<0 时，把表达式 *A***B*//*C* 改为 *ENTIER*(*A***B*−0.5)//*C*。

§2 关 于 区 域 溢 出

一、编译程序的内存分配

项　　目	占 用 内 存 单 元
留跡单元	000000
打码开关	000004—000015
控制台变量	000016—000017
中断程序	000020—000157
记各分程序中数组的首地址	000160—000177
五表区	000200—001637
记循环和过程的返回地址	001640—001723
换名表达式个数记数器	001724—001727
入口子程序用的工作单元区 e_i	001730—001777
目标程序的工作单元区 $\eta[k]$	002000—002177
常数区	002200—002577
行、代码区	002600—003777
简单变量、过程标识符、形参区 目标程序区 数组区	从 004000 开始，三个区顺序存放。只要形成的目标程序区不追上中间语言区且数组区最后不超过 067777 就行。
中间语言区	010000—037377 且中间语言区不得赶上源程序区，否则将产生混乱。
源程序区	030000—037777
控制下推表（σ表）	037400—037577
辅助下推表（α表）	037600—037777
翻译程序	040000—050477
语法检查程序	050500—054477
编排程序及其它	054500—067777
三次扫描的程序主要放在这三个区中，但还有个别指令零散地放在其它地方。	
标识符表区	060000—060677, 061400—062277
反表区	063000—063577, 064000—064577
目标中用的子程序区	070000—075577
转子记数器用区	075600—075777
打印缓冲区	076000—077777

以下估计容量问题时，凡提到“单元”都是指半字长单元。

二、关于源程序溢出

源程序自030000号单元开始存放，至多不得超过037777。因此，源程序区能容下的字符数不超过十进制16384个。每个单元可放四个电传字符（空白、空格、回车、字母键、号码键不占位置），换行必放在一个单元之首。可以事先估计每一份源程序能否在内存中被容纳下来。若在内存中已分配的区域中容纳不下，就称为溢出。

源程序区若发生溢出，则应是在输入源程序的过程中发生的。溢出信息由电传打出，应为：

*ER*8（页数，行数，字符数）

NE

S.P.HALT

溢出应发生在已输入了相当数量的纸带之后，若刚输入少量纸带就打出溢出信息，则说明发生了意外情况（可能是纸带有问题，某一行字符数过多等等），不是真正的源程序区溢出。

三、中间语言区溢出

编译过程进行三次扫描。第一次扫描时将源程序加以分析建立中间语言与各种表区以取代源语言。

关于语句部分：每一个标识符、常数、行（此处的"行"是指用'(和 ')括起来的字符串）各在中间语言区中占两个单元，每一个代码语句在中间语言区占一个单元（从一个代码语句的开括号 *'CODE'* 起直至闭括号 *'CODE'* 止，一共只占一个单元），其它（如复合定义符、运算符、赋值号等等）皆各占一个单元。关于源程序中的说明部分：凡是类型说明，不占用中间语言区；凡是过程说明，不论是一般过程还是函数过程，从过程说明开始直至过程导引结束，一共占中间语言区四个单元；凡是开关说明、数组说明及过程体中的语句部分占用中间语言区的情况与前面所述源程序的语句部分占用中间语言区的情况相同。据此可以估计每一份源程序要占多大的中间语言区。

若中间语言区已超出了030000（030000是源程序区首地址），则称中间语言区侵占了源程序区。可想而知，若源程序区一被侵占，*'BE'* 后将不能正确地输出源程序。

如果源程序区中被侵占的仅仅是已被取用分析过的源程序，则 *'BE'* 仍能正常地进行下去。但如果侵占的速度越来越快以至于刚分析到源程序的这一单元而中间语言也要放到这一单元，就称为中间语言区赶上了源程序区，这时就不能正确地编译下去，将产生混乱。

中间语言区溢出指的又是另一种情况：中间语言区并未赶上源程序区，但源程序还没分析完而中间语言区已放到区尾（037377号单元），如果继续分析源程序，就应将相应的中间语言放到037400为首的单元中去，但这些单元在第二、三次扫描时有别的用处，不能用以存放中间语言。因此，遇到这种情况，电传将打出溢出信息：

*ER*8（页数，行数，字符数）

NE

I.P.HALT

遇到源程序区或中间语言区溢出，首先应设法对源程序加以改进（多用循环与过程将能压缩源程序），如果估计压缩了还放不下，则应考虑扩体，即先将扩体程序纸带（机房有）输入，再输入源程序加以编译。

四、标识符区溢出

内存中专有一块区域存放有关标识符的信息。标准函数（像 SIN，COS等）和标准过程（像$INPUT$，$READR$等）的信息已事先存放在标识符区中。剩下的标识符区就存放源程序中自己引进的标识符。

源程序中自己引进的每一个标识符占四个单元。并列的分程序中标识符单元重复分配，嵌套的分程序中标识符单元不重复分配而是顺次往下分配。过程说明也看成分程序，但凡遇过程说明则其中的标识符单元一直不能重复使用，要专用直至该过程说明所在的分程序结束。

根据标识符区的容量及以上的分配原则，依分程序（及过程说明）的嵌套关系来计算，源程序中引进的标识符个数最多不得超过 183 个。如果超过则将在电传上打出溢出信息：

$ER8$（页数，行数，字符数）

NE

$ID.HALT$

应该在事先估计标识符的个数。如果发现标识符个数太多，可考虑如下两个改进办法：

1. 充分发挥各标识符的作用，存放中间结果用的标识符尽可能地重复使用，这样，标识符个数就能减少（见讲义 36 页）。

2. 把一些标识符合并改用数组来代替，这样也能减少标识符的个数。

五、简单变量、过程标识符、形参区，目标程序区，数组区

从 004000 号单元开始，先分配简单变量、过程标识符、形参的地址（由第一次扫描完成，分配的原则与四中所述标识符分配地址的原则一样），到第二次扫描时紧接着这个区域存放目标程序，到执行目标程序时，又紧接着目标程序区分配数组的地址（分配的原则也与四中所述原则相同）。

由于第二、三次扫描是对中间语言进行扫描，所以，简变、过程、形参区不得侵占中间语言区（即，它们从 004000 开始，不得超过 010000），否则，编译过程将产生混乱。

如果第三次扫描时建立的目标程序区赶上了中间语言区（侵占是允许的，赶上则不行）则编译过程将产生混乱。

还有，在内存中 070000 号单元之后存放的是在执行目标时要用到的一些程序（例如：SIN，COS，输入，输出，…），因此，数组区最后不得超过 067777 号单元。

如果数组区占用单元过多，可考虑将数组区搬到二体，搬的办法是在'BE'回答DD后将下面三条机器指令送入内存中指定的单元：

地址	指令
072115	000 100 000
052541	000 000 000
071557	000 177 777

将机器指令送入内存的办法有两种：

1. 将这些机器指令按手编程序的方法在纸带上穿孔，按地址 075000 启动，就可将此

纸带输入；（输入后还必须按地址 0531 的启动，然后才能打'*ST*'命令）。

2. 在控制台上逐条拨入。拨的方法见 §5 五。

六、反表区溢出

在语句中（过程体中的语句除外）用到的标识符必须是前面已说明过的，否则就要用 ER_3形式打出出错信息。但在说明部分（过程体也是说明部分！）所用到的标识符则允许先应用而后被说明。例如：

例 1. *'PROCEDURE' F (X, Y); 'REAL' X, Y;*

Y: =F1 (X) +F1 (X) ↑2;

'REAL' 'PROCEDURE' F1 (A); 'REAL' A;

F1 : =A ↑ 4 −A ↑ 3+2 ↑ A ;

例 2. *'ARRAY' A[1:G (2)];*

'INTEGER' 'PROCEDURE'G (X); ……

例 3. *'SWITCH' SW1:=S1, SW2[3];*

'SWITCH' SW2:=L1, L2, L3;

……

以上三个例子中的标识符 $F1, G, SW2$ 都是先用到而后说明的，在说明部分出现这样的情况，那是允许的。为了保证这一点，编译时若在说明部分遇到了前面尚未说明过的标识符，就先将它的有关信息记录下来，到适当的时候，查看一下刚才所记录的标识符是否已被说明过了，再决定把它的有关信息清除掉（若它后来被说明过了，就清除掉）还是以 ER_3 的形式打错后清除掉，还是继续保留这些信息留待下次查看。

反表区就是用来记录说明部分中遇到的前面尚未说明过的标识符的有关信息的。在反表区中记录时，每一个这样的标识符占用反表区中四个单元（分配地址的原则与四中所述原则相同）。反表允许占用的区域是 063000—063577，064000—064577，因此，依分程序嵌套关系来计算，列入反表区的标识符最多不得超过193个。如果超过，则将在电传打出溢出信息：

ER_8（页数，行数，字符数）

NE

IV.HALT

一般情况下，反表区是不易溢出的。但目前有的编译带中误将标识符表区溢出的信息处理为反表溢出的信息，因此，当打出反表溢出的信息时，应考虑标识符区溢出之可能性。

七、K表区溢出

打入 *'BE'* 命令后，在第一次扫描时，凡遇数组说明、开关说明、过程说明则按一定格式记录下来一些信息，由于这些信息有一定的格式，故称之为表，这三种表都称之为K表。K表可占用的区域是 000200—001637。

每一个开关说明在K表区中所占用的单元个数是：$N+1$。其中N为在该开关说明中的赋值号右端所罗列的开关表元素的个数。

每一个数组说明在 K 表区中所占用的单元个数是从偶地址开始的 $2\times M+4$ 个单元。其中M为该数组的维数。

每一个过程说明在K表区中所占用的单元个数是：$L+4$。其中L为该过程说明的形参表中形参的个数。

注意：K表中的单元不像标识符单元那样有重复使用，它纯粹是顺序分配，同此很容易在上机前先检查K表会不会溢出。

若扫描中发现K表区溢出，电传应该打出信息：

$ER8$（页数，行数，字符数）

NE

$LIST\quad HALT$

有些编译程序在处理时，K表区超过1777方能打出溢出信息，于是，若K表区超过1637又未超过1777，则电传不打溢出信息，而算题时会出现混乱。

八、常数区溢出

常数区自002200开始，直至002577。常数区存放写在源程序中的常数，不存放计算时输入的初始数据。

除002200中放逻辑值$'TRUE'$，002201中放逻辑值$'FALSE'$外，每一个常数，不论是实型还是整型，一律占两个单元。

相同的常数（类型和数值都相同）不重复分配单元。例如：2.0与0.2E1算是同一个常数，因此，它们在常数区的地址是一样的。

数值相同、类型不同的常数仍不算作相同的常数。因此，源程序中的2与2.0在常数区中的地址是不同的。

被运算符隔开的常数将被分别记入常数区。例如，源程序中若出现1/2，则在常数区分别记入1，2两个常数，而不是把数值0.5记入常数区。

常数区若不够用，则电传打出溢出信息：

$ER8$（页数，行数，字数）

NE

$CT.HALT$

常数区若溢出，可考虑如下改进方法：

1. 将一些常数合并。例如，既有0.1又有1/10时，既有2又有2.0时全用其中之一来代替。

2. 将一部分常量改为简单变量或下标变量，由纸带输入它们的值。但这样做也带来一些不好的后果：若都改为简单变量，将使标识符增多，可能导致标识符区溢出；若都改为下标变量，则由于下标变量地址计算比较花时间，所以将使算题速度降低。一般，挑源程序中出现次数最少的常量来改，这样，也可使改源程序的工作量减轻。

九、行、代码区溢出

此处所说的“行”，指的是源程序中凡用'(与')括起来的字符串。同一个字符串中，每四个字符(空白、空格、字母档、符号档不算)占行区一个单元。每一个行放完后，还要占用一个单元放行结束信息。

代码语句中的机器代码指令也与行放在同一个区中，每一个代码语句占用行、代码区以

偶地址开始的 3 + N 个单元，其中 N 为该代码语句中的机器代码指令的条数。

编译程序分配给行和代码的区域是：002600—003777，其中最后一个单元放代码语句的返回地址，因此，一份源程序中的全部行和代码语句所占用的单元个数不得超过十进制 639 个。若这些单元还不够用则电传打出溢出信息：

*ER*8（页数，行数，字符数）

NE

ST.HALT

有些编译程序，处理行时检查了溢出问题，处理代码语句时则未检查溢出问题。

一般情况下，行、代码区是不易溢出的，若原估计不会溢出而又打出了上述信息，应考虑下面可能性：

1. 源程序纸带中只有行的开括号'（ 而漏掉了闭括号'）。致使后面的全部源程序全被误认为是行内符号而记入行、代码区，于是发生了溢出；

2. 穿孔或复制有错，误将 ' 与（连在一起，变成了行的开括号，引起了编译程序的误解，把后面全部符号记入行、代码区，引起溢出。

十、其它限制：

1. 循环（一个 *'FOR'* 算一个循环）与过程说明的总个数不得超过 52 个。

2. 动态进入分程序（过程说明也算作分程序）不得超过 16 重。

3. 过程说明套过程说明不得超过四重。

4. 每一个过程的形参个数不得超过 20 个。

十一、需要说明一下，编译程序对以上各种溢出检查得并不完善，可能发生了溢出却未打出信息来，因此我们应尽量在编制程序时估计可能出现溢出的项目并加以避免。

§3 关 于 试 算

任何一份程序在正式交付使用之前都应该经过若干次试算，并根据试算中出现的情况不断加以修正。下面谈的是有关试算的一些问题。

首先，试算应把程序的各个部分都考验到。如果程序很大，为了查对方便，可以把程序分块加以试算，都通过后再连在一起。其次，试算不能满足于算出结果，而应要求算出正确的结果。为此，试算时送的初始数据应送能将计算结果与已有资料（通过实际测量、心算、笔算、电动计算机计算等方式得出的）核对。

试算若发生错误停机或试算得出的结果与实际的结果不符，都称为试算不通过。下面讨论试算不通过的一些有关问题。

一、若试算中打出出错信息，怎么办？

试算中发现错误与编译中发现错误不同，编译时若发现错误，出错信息全由电传打出，试算时若发现错误，则不仅由电传打印出错性质与出错部位，而且还从快打上打印一些信息（详见讲义 133 页），这些信息对分析错误原因极为有用，因此必须注意及时把快打上的信息取下，进行分析。一般，根据出错前后三条指令能分析出出错原因。

二、试算出错时，如果根据快打中打出的三条指令还分析不出错误原因，怎么办？

因为在打出这三条指令的同时，快打中还把出错地址打出，所以，如果嫌这三条太少，分析不出，可以用印内存的办法把要印的始末地址（可以根据出错地址及我们的需要来决定要印的范围）分别拨在 000016 和 000017 的地址部分，然后用电传打命令 '*YN*'，快打就会把我们要印的各地址中的内容印出来了。

三、四则运算溢出时，怎样知道是四则运算中哪一种运算发生了溢出？

因为从快打中打出了出错前后三条指令，我们把这三条指令的操作码部分（即每条指令九位数字中的第二第三两位）取出，即可进行分析。下面把与四则运算有关的操作码列在后面：

加：

04（浮点加）　44（定点加）　34（封规加）　14（模加）

减：

05（浮点减）　45（定点减）　16（反减）　35（封规减）　15（模减）

乘：

06（浮点乘）　26（浮点不舍乘）　36（封规乘）　46（定点乘）

除：

07（浮点除）　17（浮点反除）　47（定点除）　37（封规除）

例如，有这样的一份源程序：

```
'BEGIN'
  'REAL' A,B;
    A:=1;
    B:=0;
    A:=A/B;
'END'
×××
```

在执行时，电传打出 *TC7*(+0;　　+5;)，在快打打出：

```
01      400 000 000      B寄存器内容
        000 004 024      目标地址
        000 000 020      转子计数器重数为 2 重
        000 004 024      出错地址为 4024
        003 004 000
        007 004 002      出错前后三条指令
        002 004 000
```

我们取出出错前后三条指令的操作码部分来分析：

```
03    取某单元中的数
07    除某单元中的数
02    送入某单元之中
```

即可看出这是除法运算溢出。

四、试算时，机器指出某页某行有错，但是仔细检查该段源程序，却没有错这是怎么回事？

除了极个别的情况下机器会误打外，一般只要机器输出出错信息，源程序就一定有错。如果我们根据电传与快打的信息的确把错找出，就很好了。但一时找不出错也是可能的。这是由于编译程序对语言有种种限制（特别是关于过程部分限制最多，详见讲义§16，另外由于内存单元有限，所以对标识符个数、常数个数等等也有限制，详见讲义§20），但是对于这些限制却没有仔细检查，在许多情况下违反了这些制限时，机器造成了混乱，打出来的错误信息仅仅是“后遗症”，而不是打出原先的出错部位与出错性质。所以，遇到仔细检查该段源程序而未发现有错时，应回头再仔细检查前面的源程序是否在语义上违反了某些限制。

个别情况下还可能是编译程序本身按排不当，以致产生的目标程序中有一些工作单元重复用了，或者变址器重复用了，以致计算到某处时产生溢出或超界。

如果我们检查不出源程序是否违犯了什么语义限制，那末就需要通过追踪手段或查目标程序的手段来找错了。

五、试算时既不输出出错信息又不输出计算结果，这是怎甚回事?

一种可能是程序结构有问题，程序变成了死循环，这种错误通常用追踪手段能查出。

另一种可能是与四相同，违反了一些语义限制运算时造成混乱，个别时候是形成的目标有错，运算时造成混乱。因此，首先也是注意一下有沒有违反语义上的一些限制，检查不出时采用追踪手段或者查目标程序。

六、试算结果不对有哪些可能的原因?

试算出来的结果不正确，原因很多，下面提到的是常见的几种：

1. 初始数据不对

2. 编程序的同志对语言中一些概念未了解清楚，以致程序有错（往往是由于对过程中一些基本概念及标识符的作用域了解不清而引起）

3. 由于编程序时的一些疏忽，致使程序未体现出题目的本来意图，以下一些疏忽是常见的：

1°一些标识符用以累加中间结果用的，但忘记给它送初值；

2°循环体内包括许多语句，此时应该用 *'BEGIN' 'END'* 把它们括起来，而编程序时忘掉了。于是机器只对循环体的第一个语句反复执行多次，其它语句却被机器认为是循环体外的语句；

3° 用 $A[K,K]$ 来除以 A 的第 K 行中的全部元素，编出以下程序：

'FOR' $J:=1$ *'STEP'* 1 *'UNTIL'* N *'DO'* $A[K,J]:=A[K,J]/A[K,K]$;

这是错误的！应改写为：

$C:=A[K,K]$; （C 是一个工作单元）

'FOR' $J:=1$ *'STEP'* 1 *'UNTIL'* N *'DO'* $A[K,J]:=A[K,J]/C$;

或改写为：

'FOR' $J:=1$ *'STEP'* 1 *'UNTIL'* $K-1$, $K+1$ *'STEP'* 1 *'UNTIL'N'DO'*

$A[K,J]:=A[K,J]/A[K,K]$;

$A[K,K]:=1$;

4. 在极个别的情况下，可能是翻译得出的目标程序不对

以上所谈到的四项，前三种用追踪方法一般都能查出，后一种通常需查目标。

七、怎么进行追踪？

追踪有两种：一般追踪与局部页行追踪。

要查程序的结构有无问题应采用一般追踪。只要在编译前按下开关跳 1，那么在执行目标程序时快打就会把以下一些特征打出来：

执行到带标号的语句，打出：	页号	行号	0
执行转向语句，打出：	页号	行号	2
进入过程说明，打出：	页号	行号	4
调用过程，打出：	页号	行号	6
退出过程说明，打出：	页号	行号	8

局部页行追踪可以有助于寻找程序编制者的一些疏忽。只要在编译前按下开关跳 3 并把追踪的范围拨在控制台上 000016 和 000017 单元，那末执行目标程序时进入追踪范围内每一行，快打上都打出页号行号和字母 A，并且在追踪范围内每执行一个赋值语句，就把执行后赋值语句左部变量的值从快打上输出。

在进行局部页行追踪之前，应通过一般追踪及其它分析，把出错的可疑范围逐步缩小，然后在此范围内实行局部页行追踪。另外，为了便于分析，实行局部页行追踪时送的初始数据应选用较为简单的，最好使得计算的各个中间结果能方便地进行心算，以便把心算的结果与局部页行追踪打出的结果进行对照。

八、如果要查目标，需要先打出哪些信息？

如果经过追踪后的分析，发现源程序执行中有问题而源程序本身又没有语法和语义上的错误，那末就要查目标程序了。如果对指令不太熟悉，可以请较熟悉的同志帮着查。为了查目标方便，最好打出以下信息：

1. 标识符名与分配的地址对照表（编译前按下开关跳 6）。
2. 源程序页行与目标地址对照表（编译前按下开关跳 5）。
3. 试算后用电传命令 'SM' 把目标打出来。
4. 试算后用电传命令 'YN' 把 000200 为首的内存区内容（即 K 表区）打下来。（可以根据题目大小估计一下 K 表约多大，先把所估计的范围例如 000200 至 000500 拨在 000016 和 000017 上，打命令 'YN'，如果发现连续打出 0 则不必再打，否则表示尚未打完，可把接着的一个范围，例如 000500—000700 再拨上，再打 'YN'，直至打出一批 0 为止）
5. 试算后用电传命令 'YN' 把 001640－001723 的内容打下来。
6. 数组与分配的地址对照表（执行目标前按下开关跳 6）。

§ 4 还要注意的几个问题

一、为了调程序的方便，特别是为了了解清楚程序的结构是否正确、执行程序时的流向是否正确，除了利用一般追踪手段之外，还可以在程序的一些关键地方、转折之处多安插一些打印信息（如果打印的量不多，可以用 $PRINTS$ 语句让电传印一些事先想好的字符；

如果打印量大，可以让快打打印一些规定好的特殊数，如11111，22222，12345，……），在这些打印语句之前应该写上↑↑或↑↑↑。这样，当我们把程序调好之后，不再需要打印这些信息时，只要不按下相应的开关跳，就不会打印）。

二、不论是正式计算还是试算，核对初始数据总是十分必要的，如果初始数据不多，可以直接核对纸带，如果数据非常多，直接核对还不一定保险，那末，应该在程序中按排输出语句把初始数据输出。输出初始数据的语句不要放在源程序太靠后的地方，以免程序执行到中途出现溢出等意外情况停机时，初始数据输不出来。一般，在初始数据输入后不久就按排它的输出语句。

输出初始数据的语句最好也在前面写上↑↑或↑↑↑，因为同一份初始数据一般输出一次就够了，复算时无必要再把初始数据输出，那就只要不按下相应的开关跳好了。

三、有时，在程序执行过程中需要查看一下当时某个变量的值，或者向某变量所在的单元送某个值，或者想改变一下控制台变量的值，或者想先检查一下中间结果或初始数据，…考虑到这种要求，在源程序中适当的地方按排一些↑↑*STOP*或↑↑↑*STOP*是有好处的。当把相应开关跳按下时，程序执行到此处便处于停机状态，如果我们要做的工作完成了，要程序继续执行，只须拍一下启动键，机器便又接着工作了。当然，开关跳不按，*STOP*便可不起作用。

四、电传和快打都会有误打，所以，作为正式交付使用的结果数据，最好能有复算，以便把两次打出的结果加以核对。若第一次算完马上进行复算且前面的停机是正常停机（停机地址是074326），则只要把数据纸带退回，并拍一下启动键，马上就可打入命令'*ST*'进行复算了。

§5 控制台上的一些操作

一、按某地址启动

1. 按*DT*（使机器处于单条状态）
2. 按*ZJ*（即总清：清运算器、控制器）
3. 拨启动地址于*DM*上
4. 按J_{SZ}（把启动地址置入J_{SZ}中）
5. 恢复单条键（使机器能连续工作）
6. 按*QD*（启动）

二、强迫停机

1. 强迫光电机停：按一下单条（*DT*键）和外部（*WB*键）即行。
2. 强迫主机停：按一下单条*DT*即可。

按下*DT*停机后，若要机器继续工作，只要把*DT*键恢复起来，再按一下启动键就行了。

三、打印内存单元中的内容

1. 把要打印的内存起始地址拨在控制台上的排码开关000016上。把要打印的最后一个内存单元的地址拨在000017上。
2. 电传打命令'*YN*'

于是，快打就以指令形式把要打的内存内容打出了，打完后，电传打出'*DD*'以示结束。

如果电传有故障，则可采用如下方法：

1. 按单条 *DT*
2. 按总清 *ZJ*
3. 首末地址拨在000016和000017上
4. 在 *DM* 上置075250
5. 抬单条 *DT*
6. 置 *B*（按 *B* 键，它在 *Jsz* 的边上）
7. 置地址（按 *Jsz*）
8. 按启动键

于是，快打就以指令形式把要打的内容打出来了。

四、读某一内存单元内容

1. 按 *DT*（单条）
2. 按 *ZJ*（总清）
3. 把要读单元的地址按在 *DM* 上
4. 按 *DU*（读）中的 *QZ*（读全字长：即读出48位内容）或 *BZ*（读半字长：即读出24位内容），此时机器就把该单元内容从内存送至寄存器 *C*
5. 按显示（*Xs*）中的 *C* 键，此时机器就把该单元内容显示在控制台第二排氖灯上。

五、向某内存单元写入内容

1. 按 *DT*（单条）
2. 按 *ZJ*（总清）
3. 把该单元的地址按在 *DM* 上
4. 把要写入的内容拨在控制台上的打码开关 000016 上。如要写入的是以该单元为首的48位内容（但此时要求该单元地址必须是偶地址）则把写入内容拨在000016和000017上。
5. 按 *Xe*（写）中的 *BZ*（写入24位内容）或 *QZ*（写入48位内容）

六、符合机机

需要用符合停机时；

1. 按 *FH*（符合）中的 *Z*（指令——即与指令地址符合时停机）或 *D*（地址——即与指令中出现的地址符合时停机）
2. 将需要符合停机的地址码按在 *FHDM* 上

这样，机器在执行过程中遇到这个地址就停机了。

§ 6 关于代码语句的修改指令

本校关于代码语句的修改纸带由下列指令组成；

```
065404:    062  050  415
065756:    243  110  000
           043  047  441
```

```
                242  113  000
                043  046  556
                242  113  000
                043  047  442
                242  113  000
                243  071  000
                062  065  402
065233:         066  056  321
                052  054  716
                264  000  001
                062  065  252
                066  065  310
065156:         064  050  474
050474:         066  056  400
                024  002  010
                065  065  222
                062  065  252
065225:         300  050  170
054627:         000  003  777
```

将上述指令按手编程序方法穿孔（注意最后一条指令之后勿忘穿结束号否则输入时光电机将停不住）就得到一条纸带。

如果源程序中有代码语句，则算题前应先将上述纸带输入到内存，具体操作步骤是：

1° 把编译带送入内存(用电传命令'*DG*'或直接送编译带或按地址 000004 启动都行)；

2° 将上述修改纸带装在光电机上，然后按地址 075000 启动，本修改带便被送入内存了；

3° 按地址 053100 启动；

4° 电传打命令 '*IN*'，……，以下的操作与以往相同。

注：本编译程序实际上一共扫描五次，其中第三、四两次扫描程序量较少、不进行任何语法检查，因而它们与用户没有直接关系。通常都把第一、二、五次扫描分别称为第一、二、三次扫描，因此就说编译程序对源程序一共进行三次扫描。以下谈到第一、二、三次扫描都是指原第一、二、五次扫描。