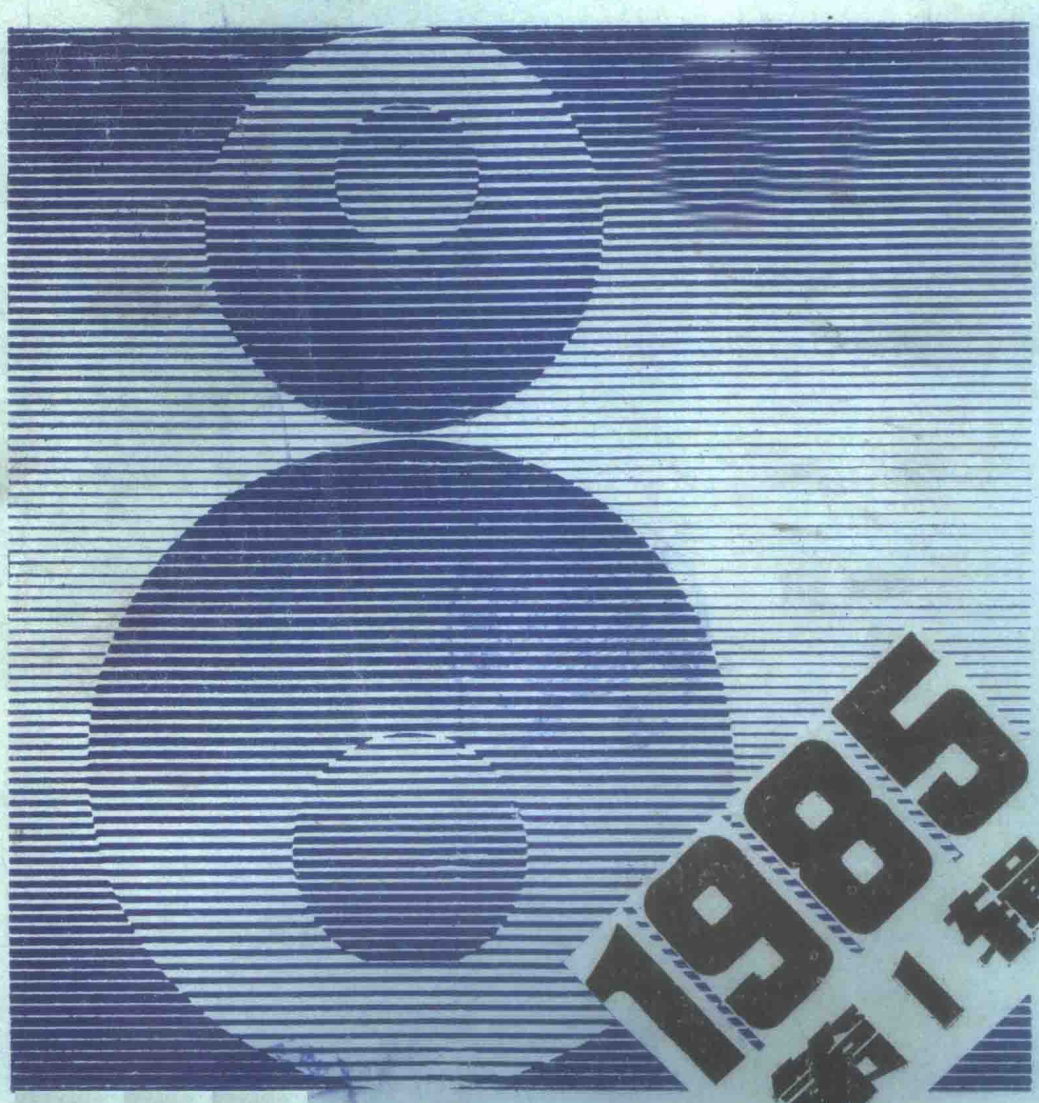
WEIXING DIANNAO YINGYONG

微型电脑应用



1985年1-5集

上海交通大学出版社

主　　编　张钟俊
常务主编　白英彩
责任编辑　叶安麒
封面设计　朱天明

微型电脑应用
（1985年 第1辑）
上海交通大学出版社出版
（淮海中路1984弄19号）
新华书店上海发行所发行
交通大学印刷厂排版印装

开本 787×1092 毫米 1/16　印张：8　字数 200000
1985年6月第1版　1985年7月第1次印刷
印数：[illegible]
统一书号：[illegible]　科技书目：101-246

定价：1.70 元

目　录

1985 年第 1 辑（总第 4 辑）

专 题 论 文

微 机 应 用

综　　述

译 文 选 载

多 机 与 局 网

控制流与数据流分析在反汇编中的应用

云南大学计算中心　颜　彦

一、问题的提出

反汇编程序是软件移植、维护、调试时的一种有效工具。它能把目标程序翻译成汇编语言的源程序。在目标程序中，指令与数据都以相同的形式——一串二进制代码存放于内存中，无任何标志加以区分；已知的信息也仅仅是目标程序的起始地址、结束地址或启动地址。机器在执行过程中可“动态”地区分指令区和数据区。而用户对于一个程序，只希望了解其“静态”形式。如果反汇编程序采取跟踪轨迹的方法，进行“动态”区分，有可能使呈现在用户面前的汇编源程序带“动态”迹象（如出现指令修改和覆盖等操作），这就非用户所需了；因此如何从目标程序的形式结构上入手，即实现“静态”区分就成了反汇编程序的主要任务。

通常，程序的启动入口，由部分控制类指令确定的地址单元内必存贮某条指令的首字节，该指令（假定不是控制类指令）到下一控制类指令之间的存贮单元里必存放着指令代码。因此，若知道了所有控制类指令确定的转向入口，就可完整地、正确地区分指令区和数据区。但是，由于指令系统中存在PCHL(8080)、JP(HL)、JP(IX)、JP(IY)(Z80)RET等指令。由它们决定的入口地址只有在执行过程中才能知道。遗漏一个入口，就可能会把很大一块指令区域误认为数据区域。

汇编语言编程有很多技巧，这是由于汇编指令的灵活多变所致。这些技巧也使反汇编程序的设计变得很复杂，主要难点是寻找“入口地址”，特别是寻找在程序动态执行时才形成的“入口地址”。

就目前一些关于反汇编程序的资料和几种实际的反汇编程序而言、它们对目标程序都有一定要求，需要操作员干涉，不能自动完成反汇编工作，这些是它们的不足之处。

二、程序结构、程序流分析及其与反汇编的关系

1. 程序的结构

若知道了一个程序的所有入口地址，就可以把该程序分成若干基本块，即每块仅由指令组成且至多包括一条控制类指令，块内每条指令皆顺序执行。划分基本块的算法如下：

(1)将所有入口地址按从小至大的顺序排列，即ENTRY(1),ENTRY(2),…,ENTRY(n)，

(2)对每个入口 ENTRY(i) (i=1, 2, …, n)，从它开始分析目标程序，直到遇控制类指令或ENTRY(i+1)为止。那么，该ENTRY(i)至控制类指令或ENTRY(i+1)（不包括）之间的指令就构成一基本块。

显然，全体基本块描述了程序中的所有指令。为描述程序的结构，定义基本块B的前件基本块为：

PRO(B)={B′|B 紧接在 B′ 后执行}。

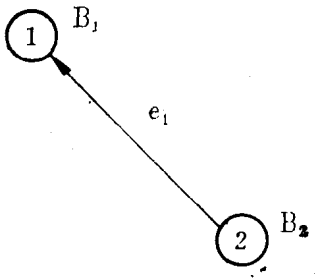


图 1

图1表示基本块 B_2 的前件是 B_1，为能方便地描述问题，用i表示基本块i，用箭头表示前件关系（箭头指向前件）。

定义了基本块及其前件关系，就可描述程序的指令及执行顺序。因此定义程序的结构为有向图 G=(V, E)，其中 V 是基本块集合 $V=\{B_1, B_2, \cdots, B_n\}$；E 是前件关系集合 $E=\{e_1, e_2, \cdots, e_n\}$。

知道了程序的结构，就很容易区分指令区和数据区。

目标程序的开始地址(BEGIN)、结束地址(END)、启动地址(START)都可知，在某种情况下，还可能知道另外一些启动地址(ENTRY)。因此区分指令区或数据区的工作可形式化地描述成：

$$\{(BEGIN, END)/START、ENTRYS\} \overset{划分}{\Longrightarrow} \{(ENTRY(1)、EXIT(1)),$$

$$(ENTRY(2), EXIT(2)), \cdots, (ENTRY(n), EXIT(n))\}$$

因为基本块完全可以用一个入口地址和一个出口地址表示。因此基本块可定义成：

B=(ENTRY, EXIT)

如图2所示。

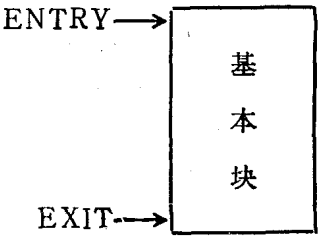


图 2

2. 程序流分析

程序流可分为控制流（CONTROL FLOW）和数据流(DATA FLOW)两类。控制流，狭义地讲是指程序中控制类指令的控制转移关系。广义地讲是指指令之间执行的先后次序。本文只谈控制转移关系。

实现最终划分，须一步一步地进行。假定某时刻有k个基本块（有待进一步细分）：

{(ENTRY(1), EXIT(1)), …, (ENTRY(k), EXIT(k))/ENTRYS}

其中ENTRYS={ENTRY(l), ENTRY(l+1), …, ENTRY(m)}表示还未分析过的“入口地址”。

从ENTRYS中选择一个最小的“入口地址”，记为min 1再从ENTRY(1), …, ENTRY(k),ENTRY(l),…,ENTRY(m)中选择一个“入口地址”,记为min 2,定义它为所有大于min 1的“入口地址”的最小值。从min 1开始分析目标程序，直至遇到min 2或者遇到一条控制指令。不论哪种情况，都可形成一个新的基本块，若遇到的是一条控制指令，并且由这条指令得到一个入口地址ENTRY，那么它与已有的入口地址和基本块有三种关系：

(1) 在ENTRY(1),…,ENTRY(k),…,ENTRY(m)中存在ENTRY(i)=ENTRY，则丢弃ENTRY。

(2) 存在 B_i=(ENTRY(i), EXIT(i))，使得

$$ENTRY(i) < ENTRY \leqslant EXIT(i),$$

此时把这个基本块再划分，如图3所示：

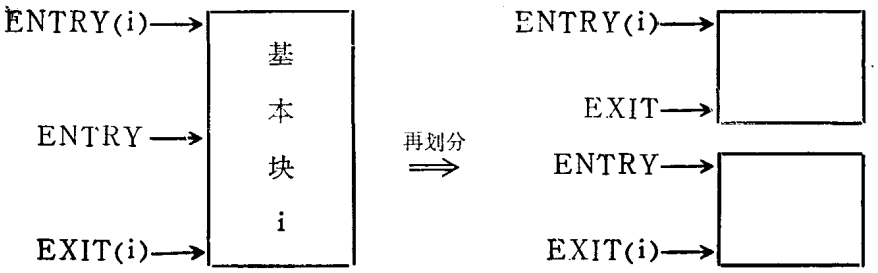


图 3

(3) 其它情况下，把 ENTRY 放到 ENTRYS 中。

重复上述划分直到 ENTRYS 空为止。这时似乎完成了基本块的划分，其实不然。因为指令系统中存在着 JP(HL)、JP(IX)RET 等指令，它们也决定着一定的控制转向，但静态时又无法立即得到控制转向，因为控制转向的值是存放于寄存器或栈顶，而它们的内容只有在动态时确定。因为执行过程中 HL 或栈顶值的生成是由一系列动作或操作实现，而动作是由一系列指令来描述的，因此若知道了这一系列指令，通过“分析”便可知道 HL 或栈顶的值，即可知道转向。微机的寄存器资源一般比较缺乏，对寄存器的操作可以提高速度，有经验的程序员总是设法充分利用寄存器资源，因此即将使用或正在使用的寄存器的值总是在较近的地方赋值，但栈顶之值的赋值点可能远一些，对这种情况也可往前“分析”一段距离得到栈顶之值。

由于基本块的规模不固定，因此赋值点有可能跟引用点在同一个基本块内，也可在前件基本块内(或前件之前件)。图 4 基本块 k 中 RET 的赋值点在 k 的前件 i，j 内，而图 5 基本块 n 中 JP(HL)的赋值点在 n 自身内。

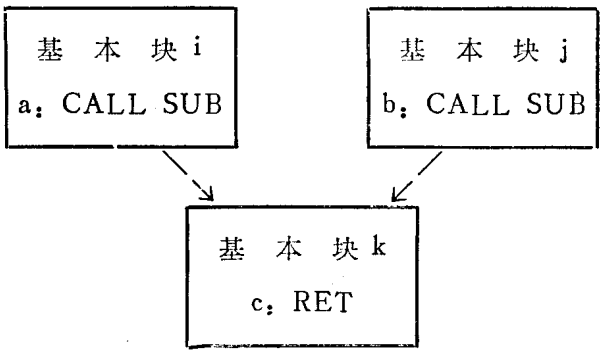


图 4

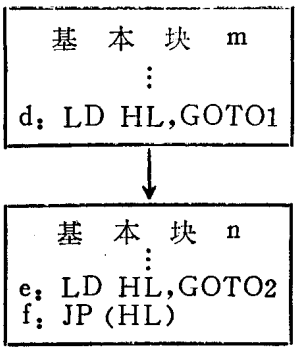


图 5

因此在控制流分析时，一方面需记录下基本块的信息，另一方面需记录下前件关系，记录基本块的信息以后将用于区分数据区和指令区，记录前件关系为后面的数据流分析提供必要信息。

完成控制流分析后，往往可初步了解程序的结构。图 6 实线部分表示已知的基本块，虚线部分表示未知基本块。由于 B_2 中有 JP(HL)，若不分析 B_2 或 B_1，就无法知道 B_3、B_4、B_5。但若知道了 B_5，通过分析 B_5、B_2，就可知道 B_6、B_7，从而得到完整的程序结构。

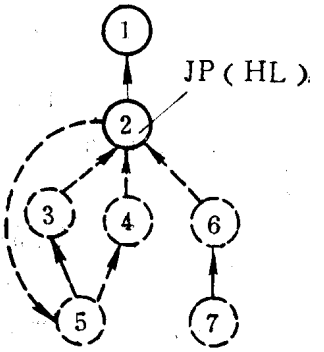


图 6

为保存信息，应有一张表格，格式如下：

基本块编号	入口地址	出口地址	前件信息	栈操作信息
⋮	⋮	⋮	⋮	⋮

“栈操作信息”栏的作用见数据流分析一节。

下面对控制类指令分类（以Z80指令系统为例，对8080、8085、6800、6502指令系统仍适合）。分类的根据是每条控制类指令确定的控制转向的数目和一些特殊性质，共分六类。

第一类：可确定一个控制转向，有一个后件(图7)，如CALL nn，JP nn，JR e等。

第二类：有两个后件(图8)，如CALL cc，nn；JP cc，nn；JR c，e；JR NC，e；JR Z；DJNZ。

第三类：JP(HL)、JP(IX)、JP(IY)、RET，遇到这类指令无法立即确定控制转向，只有进行数据流分析后，才能得出HL或栈顶之值（控制转向），它们至少有一个后件(图9)。

第四类：RET C，这类指令可立即确定一个后件，但有一个后件不能确定(图10)。

第五类：HALT，遇到这条指令不作任何处理，因它无后件(图11)。

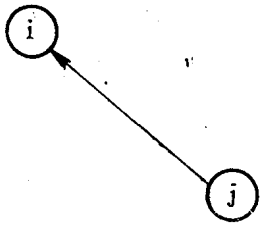


图7

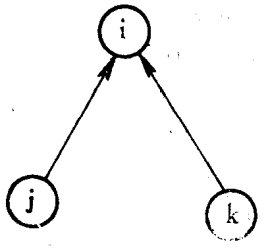


图8

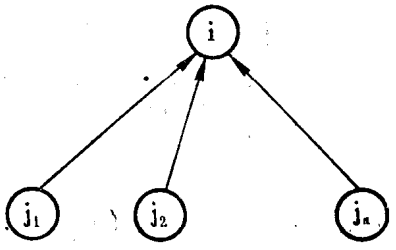


图9

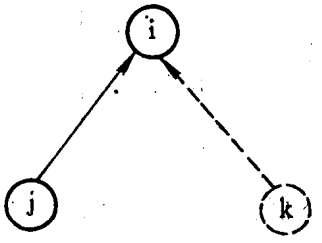


图10

图11

第六类：RET I、RET N、RST P、这类指令比较特殊，视需要对它们进行不同处理。

含有三、四类指令的基本块，要对它们进行数据流分析。因此在信息表中有必要对它们标特殊记号。

在编译程序中，应用数据流分析技术，通过建立引用—定值链，实现目标程序（中间码程序）的优化，这里我们利用数据流分析技术，对目标程序进行分析，建立引用—定值链，

以建立控制转向。下面先提出一些约定。

(1) 寄存器名称视为变量名。

(2) 存贮器单元的地址也视为变量名。

(3) 栈视为一数组变量，SP 为数组变量的下标变量。

(4) 连续存贮器单元也视为一维数组变量。

为区分标号值和存贮器单元地址，在存贮单元地址下加一下划线。如 ADDR 。有了上述约定，可把指令形式变换成：

LD A, 10 ⟹ A:=10

LD HL, HERE ⟹ HL:=HERE

JP SUB ⟹ GOTO SUB

因此汇编指令与高级语言的语句是等同的，现以实例来说明数据流分析(图12)。假设经过控制流分析，得基本块 i，j。(虚线方框内表示未知的基本块。)但通过数据流分析则可知基本块 j 中含有第三类指令，因此要对它进行数据流分析的目的是为了得知 HL 在位置㉓时的值。先分析基本块 j，建立引用—定值链⑳→㉒→㉓或 DE→SAVE→HL。由于链点是不知其值的 DE，因此要对前件基本块 i 分析建立 DE 的引用—定值链⑫→⑳或 HERE→DE，这时链头为一已知数，分析结束。接下去是综合，首先合并两链得 HERE→DE→SAVE→HL。因此 HL 的值是 HERE，得到基本块 k。

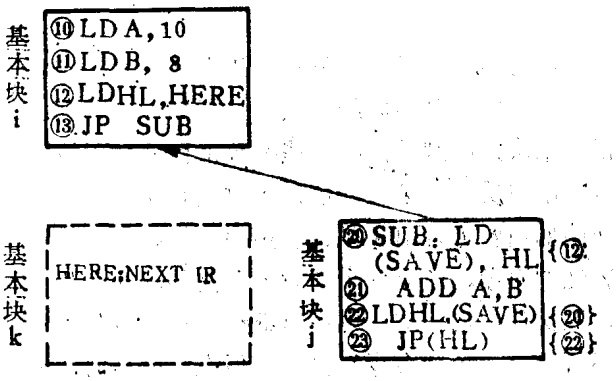


图 12

再谈谈“综合”这个问题，以下列说明：

⑩ LD IX, BEGIN

⑪ LD HL, 002

⑫ ADD HL, HL {⑪}

⑬ EX DE, HL {⑫}

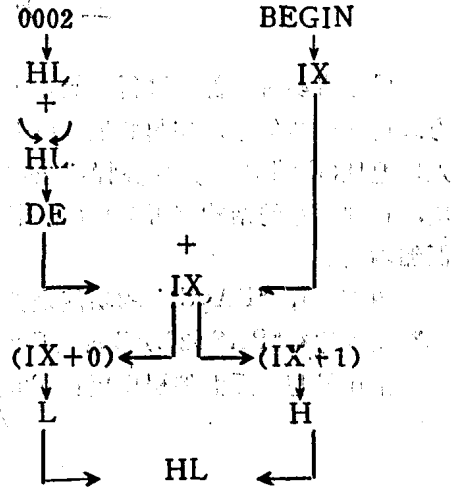
⑭ ADD IX, DE {⑩, ⑬}

⑮ LD L, (IX+0) {⑭}

⑯ LD H, (IX+1) {⑭}

⑰ JP HL {⑮, ⑯}

合并后的引用—定值链



模拟执行链上的指令，可得出 HL 的值。一般而言，形成转移地址所需的运算不太复杂，

数量也不多，故模拟代价小。

在程序设计中，常常使用堆栈，现论述栈的数据流分析，前面表中的“栈操作信息栏”主要记录两数据 n_1 和 n_2，前者是基本块内进栈操作次数，后者是基本块内出栈操作次数，栈数据流分析的目的是得出栈顶值。即定值点在哪点？下面是栈数据流分析算法：

(1) 任意正整数 n 赋给变量 SP，SP:=n>0。

(2) 分析时遇进栈指令，则 SP:=SP−2，遇出栈指令，则 SP:=SP+2，在含有引用点的基本块内，还需记录入栈指令的位置。

(3) 遇引用点时，如 sp<n。则定值点与引用点在同一基本块，令 m=(n−sp)/2，取第 m 个入栈指令位置（按位置的顺序从小到大顺序排列）作为定值点，如 SP≥n，则定值点与引用点不在同一基本块内故转(4)对前件基本块内进行分析。

(4) 按公式 SP:=n−2*(n_2−n_1)对 SP 重新赋值，分析时每当 SP=n 时，必须记录入栈操作位置于 INTO 变量，如新位置大于 INTO 中原值，则用新值代替原值，否则保持原值不变，分析完基本块后，如 INTO 非零，则定值点在 INTO 中，否则取出该基本块的前件的 n_2、n_1，对前件进行分析。

有时候会发觉栈定值点的值来源于某一寄存器，为此还需对寄存器进行数据流分析。

若决定控制转向的参数在目标程序中，那么根据上面的讨论，可自动完成反汇编工作，但如果决定控制转向的一些参数来自外界，该项处理就较困难，本文提出一些建议：

用户反汇编某个程序，应对它有所了解，至少是对外部特性的了解，外部特性无非是：功能如何？处理对象是谁？用户怎样控制？了解处理对象后，可人为地选择一个或几个有代表性的对象，建立一个“对象”文件，在分析中如发现链头寄存器的值来自外界，则从“对象”文件中取一值。以便再继续进行流分析。用户对一个程序的控制，通过命令实现，因此还可建立一个“命令”文件，但在流分析时，怎样才能知道所需数据来自“命令”文件或“对象”文件，这个数据应是两文件中的哪一个元素？这些问题还有待进一步探讨。

程序流分析的算法比较成熟（见参考文献[1]），因此反汇编程序流分析的一个难点是如何提供一条指令的足够信息，这方面也需作进一步探讨。

三、反汇编程序的总体结构

程序启动时，首先进行初始化工作，主要对表格及各种数据结构初始化。紧接着以对话方式询问一些信息。如起始地址(BEGIN)，结束地址(END)，启动地址(START)或其它入口地址(ENTRY)，“控制器”模块根据程序的结构特征选择调用“PASS1”或“PASS2”模块，直到程序的结构完善为止(见图13)，这时可根据用户需要在 CRT、LPT、DISK 上反汇编输出。

图 13 中，“PASS1”的功能是控制流分析。“PASS2”的功能是数据流分析。“中止显示”是为弥补“PASS2”功能现阶段的不完善性而临时配置的一个对话性模块。

上述设想的反汇编程序已在 Turbo DOS 操作系统支持下运行。

（下转第 85 页）

微型中文关系数据库管理系统 CRDBMS 1.2

上海交通大学微型计算机研究所　李慕靖

一、在微型机上实现数据库的意义

随着微型计算机在社会生活各个领域中的应用日益深入，人们想在微型机上分享计算机科学一切成果的愿望也越来越强烈。数据库技术是一门新兴科学，它不仅能帮助扩展人们的记忆，更重要的是能帮助人们去控制与之相关的事物，因此，数据库系统是现代信息控制系统不可缺少的重要构成部分。在应用微型计算机的热潮中，人们强烈地希望在有限的条件下建立自己的信息控制系统，而数据库就是这种系统的主要支柱。然而，一个功能比较完善的数据库往往是一个很复杂的系统，它对计算机的软硬件条件的要求比较高，而这些条件又恰恰是一般的微型机所不具备的。这是一个巨大的矛盾。为了解决这个矛盾，人们作了许多努力，使数据库技术能在微型机上发挥作用。我国所拥有的计算机中微型机占了绝大多数，因而这一点就更具有重要意义。

本文通过对在仅具两个软磁盘驱动器的国产 DJS-053 中西文系统上实现的微型中文关系数据库管理系统 CRDBMS 1.2 的设计思想和实现方法的讨论，提出了一些在微型机上实现数据库应当注意的问题和应该采取的策略。CRDBMS 1.2 具有如下的主要特点：

(1) 具有中文关键字，能处理汉字信息，语法简单直观，易学易用，符合中国人的习惯；

(2) 数值处理范围比较大、具有双精度数，并可以进行各种类型的数的四则运算；

(3) 提供一个中文 BASIC 语言子集，增加了用户编制应用程序的灵活性；

(4) 在操纵体内，数据库的数据可以参加表达式的运算，中文 BASIC 的变量可以和元组属性交换信息，对属性值可以按名任意地读出和写入；

(5) 既提供固定格式的输出，也提供给用户以组织自己的输出格式的工具。

CRDBMS 1.2 的实现是对在微型机上实现数据库的一种尝试，希望能对关系数据库技术在微型机上的推广应用有所贡献。

二、微型机上实现数据库应当采取的策略

从微型计算机的资源和能力来看，运行数据库系统这种大型软件具有许多困难。要设计好这种数据库，必须解决资源短缺和数据库系统资源需求较多的矛盾以及系统的专用性和通用性的矛盾。

要解决好这两个矛盾，首先要认清系统所处的工作环境和服务对象。事实上，拥有微型机的单位都是些中小企事业单位，他们要求处理的数据类型比较单一，数据量相对来讲不是很

大，处理的方式比较固定，对系统响应时间的要求也比较低。在这样认识的基础上，解决这两个矛盾就有了方向。

为了解决资源短缺的矛盾，设计者必须注意充分利用每一个内存单元和每一块外存区，并使用较好的数据结构，有时还要牺牲时间以换取空间。

解决好通用性和专用性的矛盾，就要求设计者不拘泥于大型数据库的种种要求和规定，以实际应用上是否最经常出现为准则，削枝强干，甚至完全忽略某些方面的要求，而着重强化某些最经常使用的功能，使之真正成为一个可实用的系统，而不仅仅是一个可表演的系统。设想在微型机上建立一个完全通用的数据库，由于受到资源的限制，其功能必定会受到多方的限制，虽然表面上似乎能适应各种应用场合，实际上就会使任何一种应用都对其功能感到不满足，反而不实用。实际上，近年来出现的一些在微型机上运行的数据库系统都是这样一些简易系统。由于关系理论的日趋成熟，这种简易系统又往往取关系方法。CRDBMS 1.2 也是关系式的。

当然，评判一个系统是否为关系式的不能单凭设计者的主观愿望，而必需要有客观标准。著名的 E. F. Codd 提出了一个标准。他指出：一个数据库管理系统至少要满足下列条件，才能称为是关系式的，这些条件是：

(1) 数据库的所有信息都以表中的值的形式来表示。

(2) 这些表之间没有用户可见的导航键。

(3) 在任何一种合宜的语法中，系统至少应支持关系代数的选择、投影、等价连接或自然连接算符，而不要求助于迭代或递归命令，并且这些算符不受任何事先定义的访问途径的约束。

E.F.Codd 把满足上述三个条件的系统称为最小关系式系统。笔者认为设计微型机上的关系数据库，E.F.Codd 提出的三条可以作为最低的要求。CRDBMS 1.2 设计时注意满足了这些最基本的要求。

作为一个实用的数据库管理系统，仅仅满足上述的三条还是很不够的。因为实现数据库，用数据库存放数据，使之易于检索并不是用户迫切追求的根本目标。用户的根本目标是要能使用存贮在数据库里的数据来满足自己的各种各样的要求。由此可见用户所要求的简单的信息系统至少应该分两个层次，第一个层次是数据库，这是基础，第二个层次就是建筑在这个基础上的用户的应用程序，这种应用程序应能引用和修改数据库中的数据。在一般的大型或中小型数据库中这两个层次分别是由数据库管理系统和宿主语言来承担的。在微型计算机的环境下，特别是对 DJS-053 这样的仅具软盘驱动器的机器来说，可以把这两个层次的功能溶合在一起，使这两层功能都由数据库管理系统来实现。在这方面比较成功的典型例子就是 dBASE Ⅱ，它是建立在 IBM-PC 上的一个自立的数据库管理系统，它既有数据的存放、检索等功能，又提供给用户一些编制应用程序的语句，使用户能利用这些工具在数据库的基础上建立自己的应用系统。CRDBMS 1.2 也是以这种方法进行设计的，所不同的是 CRDBMS 1.2 采用一个中文 BASIC 语言的子集实现第二个层次的功能。由于 BASIC 语言是使用最广的高级语言之一，所以用户能够很方便地学会 CRDBMS 1.2 的使用。显然，这种把编制应用程序的工具交给用户，由用户按照自己的意愿在数据库上建立自己的应用系统的方法，比之由系统为用户提供几种可供选择的固定应用格式的方法要灵活得多。

三、CRDBMS 1.2 的实现

1. CRDBMS 1.2 的运行环境

CRDBMS 1.2 是在 DJS-053 中文微型机系统上实现的。DJS-053 中文微机系统的结构表示在图 1 中。它由以 8085 为 CPU 的 DJS-053 主机，一对 8″ 单面双密度软磁盘驱动器，一台 MIC-58C 汉字智能终端和 KC-3070 打印机等设备所组成。整个系统在 MDOS-53 操作系统的支持下工作。DJS-053 提供给用户使用的内存空间约为 46KB。CRDBMS 1.2 用 8080/8085 汇编语言写成，目标代码占用了大约 31 KB 内存空间，整个系统采取常驻内存的方案。这样做虽然留下的空间小了点，但避免了在只有软盘的环境下运行时把一些模块调进调出，从而提高运行速度。

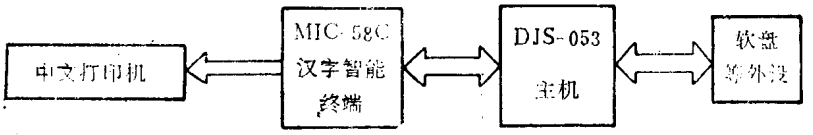


图 1　DJS-053 中文微机系统框图

2. CRDBMS 1.2 的逻辑结构

在微型机用户中，BASIC 语言因简单易学而深受欢迎。CRDBMS 1.2 为了方便一般用户的使用，也采用 BASIC 语言的形式。它的语句可以不带行号以即算方式执行（DDL 语句除外），也可以带行号执行，在这种情况下，系统将按行号从小到大逐句解释执行。CRDBMS 1.2 的逻辑结构可用图 2 表示。图中可见，如果略去 DDL 和 DML 模块，该系统就

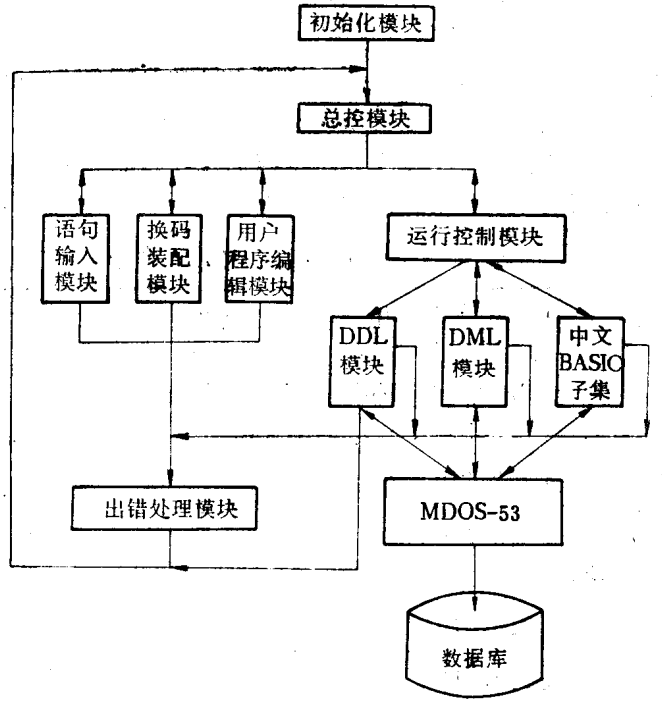


图 2　CRDBMS 1.2 的逻辑框图

是一个完整的 BASIC 语言解释程序的结构。只是受到 DJS-053 内存的限制，才只包含了一个 BASIC 语言的子集，由于已经具备了 BASIC 语言的主要功能模块，在资源允许的情况下，增加 BASIC 语言的功能已经不是一件难事了。从图中还可看到，除了 DDL 模块有些特殊以外，所有的 DML 语言和中文 BASIC 语言都是在运行控制模块的控制下运行的，这样处理的结果增加了用户操纵数据的灵活性。

3. CRDBMS 1.2 的汉字命令

现有的一些可接受汉字信息的数据库系统，其关键字往往还是西文的，这是因为关键字出现的频率比较高，并且汉字的代码又比较长，用户输入和机器处理都有一定的困难。CRDBMS 1.2 充分利用 MIC-58C 汉字智能终端的潜力，利用它提供的 64 个自定义键位，实现关键字的汉化。这些关键字分别由两个汉字或三个汉字所组成，例如：建库、表格、栏目、找出、其中、显示、最大值、平均值等。

使用时，只要先执行一段对这些自定义键进行初始化的程序，这些键位就可供使用了。用户在输入程序或命令时，只要用 MIC-58C 上的触笔点一下相应的键位，CRDBMS 1.2 就能在内部接收到一个代码同时在显示器上显示该关键字，用户使用时既直观又方便。

4. CRDBMS 1.2 的数据类型

CRDBMS 1.2 的数据类型分为字符型和数值型。字符型的数据在计算机内部是按照“信息交换用汉字编码字符集基本集 GB2312-80”中的编码存放的。这样处置的优点是对汉字和西文可以进行统一的处理，缺点是每个 ASCII 符在字符串中也将占两个字节。但是，考虑到CRDBMS 1.2 是在处理汉字信息的环境下应用的，在字符串中出现大量 ASCII 字符的机会比较少，这个缺点就不会产生严重的问题，而统一按国标码存放的好处却是明显的。

CRDBMS 1.2 的设计目标是灵活、实用。在许多微型机用户中，他们希望计算机能帮助他们摆脱日常烦琐的财务或账务工作。许多系统不能满足要求的原因是，要么数的表示格式和他们的要求不一致，要么所能表示的数的精度不够。CRDBMS 1.2 在设计时注意到了这一点，在这两方面尽量满足用户的要求。本系统把数分为整型数、单精度数和双精度数三种，这三种数的精度和所表示的最大范围列在表 1 中。

表 1 CRDBMS 1.2 的数值类型

数值类型	表示范围	存贮空间	后缀	例子
整数	−32768～+32767	2 字节	%	x% 3785
单精度浮点数 （七位有效数字）	$\pm 1.2\times10^{-38}$～ $\pm 3.4\times10^{38}$	4 字节	!	x! 3756.211 3756.211! 3.756211E+03
双精度浮点数 （十五位有效数字）	$\pm 2.2\times10^{-308}$～ $\pm 1.8\times10^{308}$	8 字节	#	x# 9436.5# 1234567890.12345 9.4365D+03

有些系统中，数值一超出整数所能表示的范围，就以浮点数的标准形式来表示，一般的用户总觉得不适应。本系统中的数在内部处理过程中能自动实行类型转换，以确保足够的精度。输出时整型数以原样输出自不待说，浮点数类型也是以一般的形式出现，直到该种精度的规定位数（单精度数七位，双精度数十五位）表示不了了，才会以浮点数的形式出现。此外，当用户需要进行人为的数值类型转换时，可以通过不同类型的变量之间的赋值语句来解决。由此可见，只要用户进行适当的定义，数值数据就会以用户所要求的形式出现。至于数值所能表示的巨大范围能满足用户的各种要求是不言而喻的。

由于在程序输入以后立即进行换码，数值在计算机内部都是以中间码的形式存放的，这样做的好处是它们的存放格式统一，机器处理起来效率比较高。当然，这样的安排在输入和输出的时候会增加开销，但由于输入是手工在键盘上进行的，这种操作和机器内部处理在速度上是不可比的，同样，输出时，显示器或打印机也是低速设备，因而用户也不会感到额外增加了开销。实际上，对各种语句在输入时进行换码，在输出或列表时进行反换码的意义也在于此。

5. 倒排的实现

在数据库系统中，倒排结构带来的好处很多，因此一般的数据库管理系统中都设有倒排机构。一个没有任何倒排索引机构的数据库系统要真正付诸实用是难以想象的。微型机上实现的数据库系统也不例外，但是真正在实施时又必须考虑到客观条件的许可程度，进行综合考虑，然后定出最佳方案。因为倒排机构也要占用资源，并且虽然在查询时能带来明显的好处，但在插入、删除元组及出错处理等方面又不可避免地要增加额外的开销。例如，在插入一个元组的时候必须把各个定义为倒排的属性的值插入相应的倒排机构，而在删除的时候正相反，在删除了一个元组以后还得在各个相应的倒排机构中进行删除。考虑到CRDBMS 1.2目前实现的环境是DJS-053，内外存容量比较小，速度也比较慢，因此不宜采用庞大的倒排机构。另外，从微型机上数据库系统运行的规律来分析，出现概率最大的是查询操作或以查询操作为基础的其他操作，因此抓住了提高查询速度这个矛盾，就抓住了主要矛盾。经过综合考虑，CRDBMS 1.2的倒排机构采用了如下的方案：

(1) 倒排机构全部用B^+树结构，因为对B^+树作查找是十分方便的。假定在d阶的B树中存有N个记录，那么这棵B树的层数L可以从下面的公式得知：

$$L \leqslant 1+\log_{(d+1)} N/2$$

因此，访问一个记录所需对外存的存取不会超过$2+\log_{(d+1)} N/2$次，也就是说，当N=10000,d=9时，最多六次“读入”就可以了。

(2) B^+树的阶数可变。这是为了充分利用外存空间，提高查询效率而采取的措施。以数值为例，整型数只占2个字节，而双精度数要占8个字节，若采取统一的模式，就必须考虑以双精度数为标准，因而在实现整数倒排时就会造成空间的浪费和增加不必要的读盘次数。B^+树的阶数是在执行DDL语句时由系统决定的，系统根据被定义的属性值的长度决定B^+树的阶数，并把该信息保存在数据库字典中。例如整数属性的B^+树的阶数是15，而双精度数属性的阶数为6。

(3) CRDBMS 1.2允许用户指定某一个关系中的任何一个属性为关键值倒排或非关键值倒排，被规定为关键值倒排的属性值不允许出现重复，而非关键值倒排则没有这种限制。

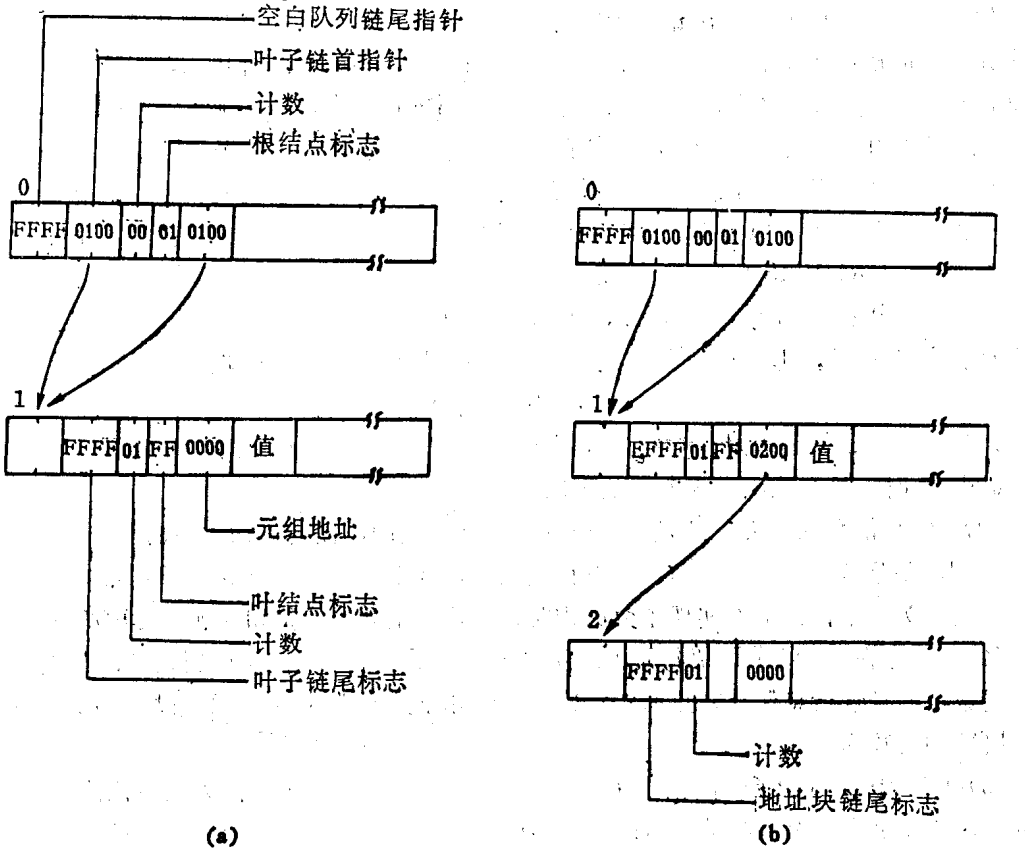
(4) B^+ 树的每一个结点占用外存一块（128个字节），其中包含6个字节的系统管理控制信息。为了防止在经过一段时间的运行以后，B^+ 树占用过多的外存空间，还考虑了空白块的回收。具体措施是每一棵 B^+ 树设置一个空白队列，首指针存放在该树的根结点中，而根结点的信息则存放在数据库字典中。此外，当 B^+ 树中的信息删除完以后，系统将先删除这棵 B^+ 树，然后再重新生成，这样就释放了全部占用的空间。

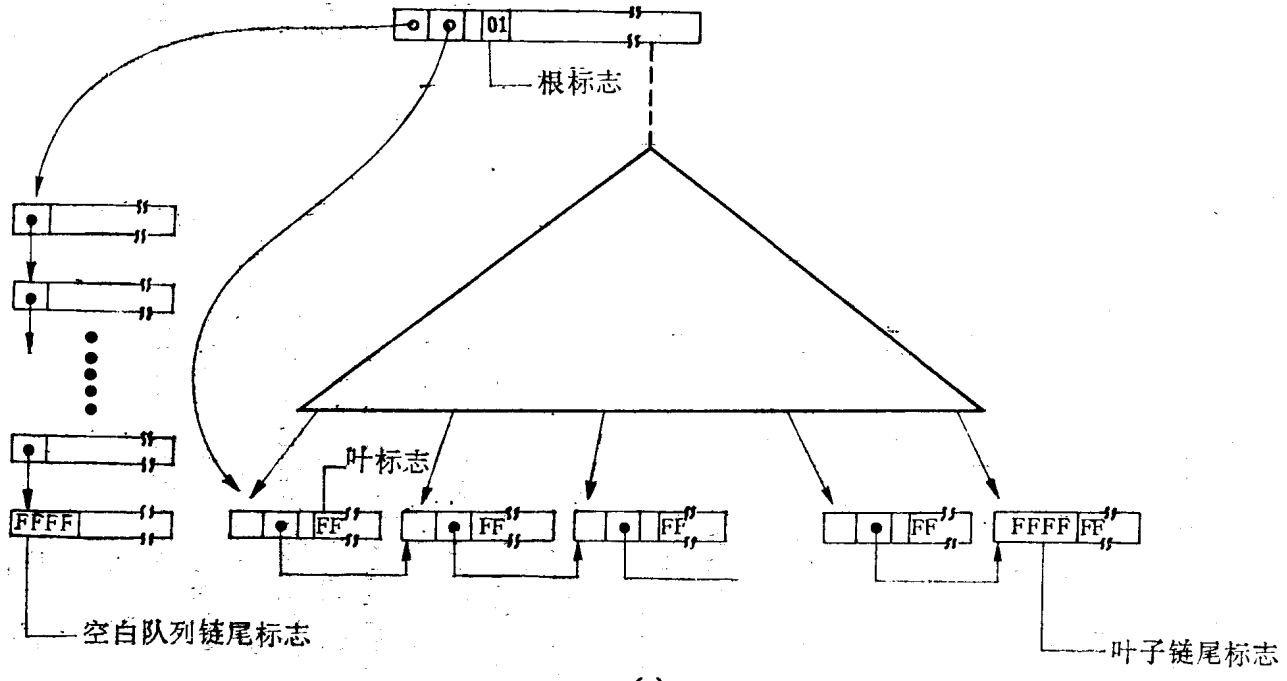
(5) 两种 B^+ 树的区别在于叶结点，关键值 B^+ 树的叶结点中存放的是具有某属性值的元组地址，而非关键值 B^+ 树中存放的则是具有该属性值的元组地址块链的首指针，具体元组的地址集中存放在地址块中。图3(a)、(b)分别为刚生成的只具有一个值的关键值和非关键值倒排 B^+ 树的示意图。(c)是系统运行一段时间以后达到动态平衡时的 B^+ 树。

6. 中文 BASIC 语言子集的设置

前面已经提到过，CRDBMS 1.2 拥有中文 BASIC 语言子集将大大增加用户操纵数据的灵活性。某些在微型机上运行的数据库系统，致命的弱点往往在于缺乏高级语言的支持，缺乏操纵数据库中数据的灵活性。例如某一表中的某些数据是另外一些数据的函数，或是另外一些表中的某些数据的函数。这些函数一般都比较简单，用四则运算就可以解决。然而，那些数据库却在这些问题面前束手无策，从而使数据库系统不能发挥出应有的效益。

CRDBMS 1.2 注意解决了这个矛盾，它提供了一些中文 BASIC 语句。虽然这些语句只是中文 BASIC 语言的一个子集，但包含了最基本的语句，实践证明可以解决很大一部分问题。中文 BASIC 语言子集包含赋值、注释、编辑、循环、条件、打印、送值、转子、转向、暂





(c)

图 3 CRDBMS 1.2 的倒排 B+ 树结构

停、继续、跟踪、解除、列表等语句。其中赋值语句是一条关键语句（包括性质类似的送值语句），CRDBMS 1.2就是把赋值语句作为纽带使中文BASIC语言子集和数据库中存放的信息连接在一起并进行交换的。利用赋值语句，可以把数据库中的数据送入变量或直接投入表达式的运算，然后再把结果送到用户程序指定的地方。CRDBMS 1.2不仅能十分方便地对一个关系中的数据进行处理，而且还能对同一库中几个不同关系中的数据甚至是几个不同库中的关系进行处理，因而是非常灵活的。

要实现这一目标，只要仔细分析一下BASIC语言赋值语句的解释程序，就能找到突破口。让我们先看一条典型的BASIC赋值语句X=X+1。执行时，解释程序是从左至右扫描该语句的。查到变量X以后，记下该变量在变量表中放数据的地址，然后计算"="右边的表达式，即取出变量X的值，把它和1相加，然后把结果按开始时记下的地址送到变量表中X的数据单元中去，这样变量就获得了更新后的值。对关系中的属性赋值也是这样，CRDBMS 1.2的赋值语句程序中增加了按属性名查找属性在元组缓冲区中的地址的功能。显然，变量表中数据存放单元的地址和属性在元组缓冲区中的地址对于表达式运算来说，它们的作用是相同的，系统只要按照这个地址读出和写入数据就行了。当然在实现时还要考虑该属性是否定义为倒排的，如果是倒排的，则在对该属性赋值时要先删除B^+树中的值，然后插入新值，最后再对该元组中的属性赋值。为了处理上的方便，CRDBMS 1.2每当扫描到$"属性名"或¥"属性名"，就会从指定的属性位置上取出值或送入值。例如，设某一关系中的一个元组的"基本工资"属性值=80，而"实际工资"属性应由"基本工资"和放在变量X中的"浮动工资"之和决定。表达式$"实际工资"=$"基本工资"+X，就能完成在该元组的"实际工资"属性位置上填入实际工资的任务。当然，这只是一个简单的例子，但要说明问题已经够了。

7. CRDBMS 1.2中的操纵体

CRDBMS 1.2中的中文BASIC语言子集可以自成系统地运行一般BASIC语言的功能，也可以在指定的条件下执行和数据库交换信息的功能。通过分析不难发现，用户在使用数据库的数据时一般总是先找出满足某种条件的全部元组，然后对每一个元组中的某些属性值进行取值或赋值（例如结算工资），或是取出某一关系中的某一元组的某个属性的值来推算该关系中的或另一关系中的其他属性值（例如核算成本），这些操作的流程可以用图4来表示。为了最大限度地提供数据操纵的灵活性，CRDBMS 1.2提供了"操纵—拨动"语句。这两条语句配合在一起执行图4虚线框中的功能。从程序设计的角度来讲，介于这两条语句之间的语句集合就称为操纵体，操纵体可以由CRDBMS 1.2的数据操纵语言和中文BASIC语言的语句所组成。以上面提到的计算工资的问题为例，假如数据库中满足条件的元组有100个，执行如下所示的程序：

```
50  操纵@
60  $"实际工资"=$ "基本工资"+X
70  拨动
    ⋮
```

就会在100个元组的"实际工资"属性栏中填入按照"基本工资"和X的值更新过的值。

"操纵—拨动"语句的意义是很明确的，"操纵"语句主要是为进入操纵体作各种准备工