北京航空学院

科学报告会论文集

庆祝建校三十周年

第六分冊　計算机类、管理工程类、情报学类

1982.10

前　言

建校三十年来，我院科研工作遵循党的方针政策，密切结合教学，取得不少成果，在此基础上，总结出了一批有一定理论价值与实践意义的学术论文。

战斗在祖国各地的校友们，在为实现我国四个现代化这一伟大历史任务的过程中，同样做出了可喜的成绩，此次他们热情响应母校三十周年校庆征稿的号召，把自己多年来从事教学、科研与工程技术实际工作中所积累起来的宝贵经验，进行理论总结，书写成文，投寄我院。

为隆重纪念我院三十周年校庆日，加强院内外的学术交流与联系，我处特编辑这套校庆论文集。文集刊载本院与校友近期科研成果或技术总结的详细文摘。为节省篇幅，在编辑时将作者原列参考文献删去。

论文按专题分类出版：

1. 材料科学与工程类
2. 无线电电子学类
3. 自动控制类
4. 发动机类
5. 飞行器与力学类
6. 计算机类、管理工程类、情报学类
7. 制造工程类
8. 数理类
9. 机械设计类
10. 医疗类、体育类
11. 社会科学类
12. 大学生论文集

由于编辑出版力量有限，时间也较仓促，难免有遗漏和不妥之处，欢迎批评指正。

北京航空学院　科研处

1982年10月

第六分册　计算机类

目　　录

第六分册　管理工程类

目　　录

第六分册　情报学类

目　　录

一种灵活、简便的通用EPROM程序写入器

机电系　杨宗煦

序　言

微计算机系统的运行都是按照贮存在存贮器内的程序来进行的。这类存贮器目前大量采用半导体只读存贮器 *ROM*。因此构成一个微计算机系统时，把所需的程序放入到存贮器中是一个必要的步骤。对于通用设备所需之程序，因为需要量大，所以在生产存贮器时，厂家就直接把程序固化在存贮器内。这就是市場上供应的配置了各种程序的只读存贮器。但是，对于科研、产品试制等工作中，经常要求用戶自己能放入特殊的程序到存贮器中，并能修改这些程序后再次放入。因此试制出了一种可抹可写入程序的只读存贮器 *EPROM* 并得到了广泛的应用。由于 *EPROM* 种类繁多，而且直接带有*EPROM*的单片计算机不断问世（如 *Intel*8748等），目前市場上供应的 *EPROM* 程序写入器为了适用于各种 *EPROM* 通常都采用一种 *EPROM* 用一块专用电路插件板来相匹配。所以一个程序写入器必须配备相当数量专用电路插件板，这就增加了整个仪器的价格，而且有时还会配不到用戶所需之 *EPROM* 的电路插件板。为此提出了试制一种较灵活、价廉的程序写入器的问题。本文通过对程序写入过程的分析，以及一种以一块通用的*SDK*—85单板机作为核心的通用*EPROM*程序写入器的试制，在这方面作了一定的探讨。

为了了解程序写入器的工作过程，下面首先解说一下 *EPROM* 在结构上的特点。*EPROM*区别于*ROM*的是采用了具有悬浮栅极的場效应管，它除了通常的控制栅极外还有一个悬浮栅极如图1（*a*）所示。原始情况下悬浮栅上无电荷，对于*N*通道的場效应管来讲，它处于低开启电压下工作，控制栅上加上正电压管子即通导，对外反应为高电平输出“1”。假如我们希望得到“0”输出，则可以采用漏极与衬底间加上反向高电压并引起雪崩型击穿。一部分电子穿过绝缘层跑到悬浮栅极上，这样就提高了場效应管的开启电压如图1（*b*）所示。这开启电压始终超过控制栅极上所加之正电压，这样此場效应管就始终处于开断状态。所以 *EPROM* 的程序写入过程亦就是把原始输出为“1”的位按要求通过加高压把其改变为“0”。如需要输出的位仍为“1”，则不需改变该位的状态，让其仍然保持“1”。反之，抹去 *EPROM* 中的内容也就是把输出为“0”的位回复到输出“1”，这也就是除去跑到悬浮栅上的电子的过程。这

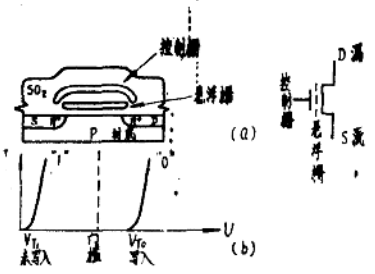

图1　具有悬浮栅的場效应管

过程是采用紫外线光照给以电子能量，从而使电子从悬浮栅上跑回衬底。所以这是一种非电的抹去方式。

EPROM 程序写入器的系统方框图示于图 2 上。系统方框图上的读写存贮器是为了暂

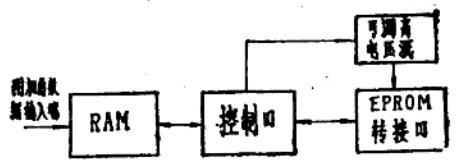

图2　系统方框图

时存放需要写入 *EPROM* 中的程序。控制器控制整个程序写入过程。高电压源按控制信号供给合适的高电压。转接器实际上是一块插座板，通过它把所有信号接到需写入程序的 *EPROM* 引出脚上去。下面以一种 *Intel* 公司 *SDK*—85单板机为核心的通用程序写入器为例作进一步具体介绍。

硬　件

此 *EPROM* 程序写入器的方框图表示在图 3 上。它由三块电路板组成：一块 *SDK*—85 单板机；一块可调高电压输出板；以及一块转接器板。下面分别予以介绍。

1. *SDK*—85单板机用作程序写入器的控制器，它按照程序写入程序来实现其控制作用。程序写入器的读写存贮器 *RAM* 亦安置在此单板机上。因为 *SDK*—85 单板机

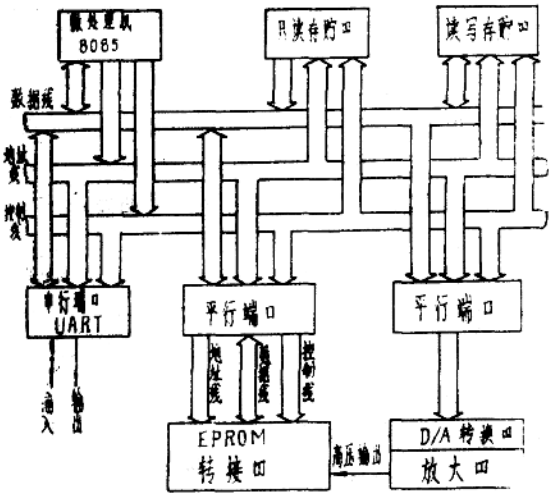

图3　程序写入器方框图

通常仅有256字节 *RAM*，容量较小。所以在 *SDK*—85 单板机的扩展母线上增加了四片2114 *RAM*，从而组成扩展的 2*KRAM*，用以扩充 *RAM* 的贮存容量。然而

有了RAM后，还有一个如何把数据放入RAM内的问题。这对程序写入器来讲是一个相当重要的环节。本系统提供了三种方式：第一，直接用SDK—85单板机上的键盘输入16进制的数码，亦即机器码和数值。这种方式对一个较大的程序就显得很麻烦，以至很少使用。但这种方式不用任何附加设备，对于一些简单的程序输入还是可以考虑采用的。第二，从其它计算机经过传输线接受数据并放入RAM中。它利用一个接收程序和$RS232C$串行传递线来接收数据并进行错误校验。这种方法可以容易地从其它计算机获得所需之数据，但是这接收程序必须与送数据的计算机之传送程序相配合，对于不同的计算机需要不同的接收程序。第三，利用外接终端设备直接输入汇编语言，通过汇编程序得到机器语言并存入RAM中。为此本系统配置了$Intel8080/8085$汇编程序和编辑程序。因此这SDK—85单板机配备终端后可直接用汇编语言来操作。这实际上具备了一定的通用计算机功能。

SDK—85单板机与需写入程序的$EPROM$的联系是通过并行输入/输出端口来实现的，片子8355和8755提供了四个8位端口，用以传送地址、数据和控制信号。8155上二个8位端口是用来调节二个可调高电压源的输出电压值。在这些端口中仅传送数据的端口需要双向的，因为检验写入$EPROM$中的数据是否正确需要从$EPROM$中读出写入的数据，其它的端口都简单地置于输出模式。串联端口采用$UART$ 8251，它既用来与其它计算机相接，从而接收数据存入RAM中；亦可以用来与终端设备相接，用以输入汇编语言。串行传递的速率目前借助软件可以得到300波特和1200波特二种。假如需要其它速率，可以简单地改变串行传输电路中的分频器来实现。

2. 可调高电压输出板

当程序写入时，高电压是必须的。此可调高电压输出板上提供了相同结构的二路电压输出，但通常仅使用一路即够。为了便于调节电压值采用了D/A变换器，由微处理机输出不同数值就可以得到0—25V之任意电压值。当数值是16进制的"FF"时，输出电压为25V；当数值为"OO"时，输出电压为0V。因为输出电压值与数值是成线性关系的，所以根据需要的电压值可以方便地计算出相应的数值。此数值由使用者放入规定的RAM地址2008H和/或2009H内，从而在程序写入过程中可以得到所需的高电压输出。为了防止电压值漂移带来误差，本系统设有一子程序用以检查在数值"FF"时输出电压值是否是25V。假如有差别，使用者可以调整板上的电位器使输出电压等于25V。

可调电压板之电路图示于图4上。由于无信号时（输入悬浮状态）D/A转换器输出为最大值，这是不希望的而且会损坏$EPROM$。但如用电阻跨接在输入端与地之间来得到零输出，对D/A变换器$ZN425$来讲此电阻值必须小于1.4K。但对于SDK—85端口来讲此值太小，端口带不动此负载。为此采用一片8212作为缓冲器。8212可用6.9K的接地电阻，这是SDK—85端口所容许的。

3. $EPROM$转接器板

由于$EPROM$引出线的多种多样，一个通用插座用于所有的$EPROM$是很难实现的。为此对各种不同系列的$EPROM$提供了一系列的插座。这些插座连接到一个公共的标准插座，通过这公共的标准插座和电缆再与程序写入器其它部分相连。各种$EPROM$的差异仅由其插座与标准插座的不同连接来解决。对于同一系列的$EPROM$由于仅少数引出线不同，可以附加少量转换开关来实现这些引出线的变换从而合用同一插座。

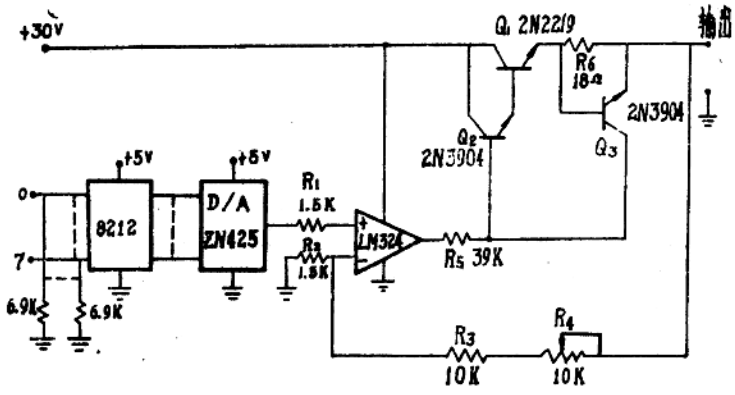

图 4　可调高电压板电路图

軟　件

整个软件包含用汇编语言所写的下列程序：

SDK—85监控程序；SDK—85扩展监控程序；

编辑程序；8080/8085汇编程序；

接收数据程序；程序写入程序。

SDK—85监控程序是SDK—85本身所固有并放置在ROM8355中。所有其它程序是为本系统而编写的，并分别固化在三片$EPROM$内。编辑和汇编程序在 8755 内，接收数据程序在2716内，其它在2758内。

SDK—85扩展监控程序是为了使 SDK—85 单板机配备终端设备而编写的。它可以执行编辑、汇编等命令，在CRT上显示存贮器内容以及更换 RAM 中的内容等。编辑程序是为了用终端输入程序而配置的。它具有所有的通常编辑功能但仅占用不到0.5K字节的位置，是比较紧凑的。汇编程序是二遍型式的并具有错误检测并加以显示出来之功能。此汇编程序占用约1.5K字节位置，同样是相当紧凑的。

接收数据程序具有计算偏置地址的能力。因为数据放入到 $EPROM$ 中的地址与它暂时存放在程序写入器 RAM 中的地址经常是不一致的，二者有一偏置值。通常其它计算机送出数据时同时送出放到$EPROM$内的地址，所以必须加上偏置值才能得到所需的RAM地址。在接受数据前此偏置值必须预先置于RAM地址2000H和2001H中。在传递过程中采用检查传递标识符，字符是否16进制数码（即机器码或数值），以及16个$ASCII$字符数值和是否正确等措施以保证正确传递。检查到任一种错误，传递过程将自动停止，并在 SDK—85地址場内显示"$LERR$"。

最后着重介绍程序写入程序。将一个程序写入 $EPROM$ 中大致过程可以用图 5 之程序流图来表示之。下面对图 5 作一些补充解说。

在程序写入过程中检查有否错误时，如发现错误将显示此错误内容及其地址并停止作进

一步运行。流图上延时环节起的作用是为了获得程序写入条件所需维持的时间。此外，从程序写入过程可知，对不同的 $EPROM$ 仅供给初始条件以及程序写入条件是不同的，其它基本相同。为此可以把这些相同部分按功能用各种子程序来表示。这儿一共编写了16个子程序以供使用者自行组合成一个所需的程序写入程序。在本系统内利用这些子程序编写并配置了一个可以用于 $EPROM$ 2708、2716、2758、2732、8755 的程序写入程序。此外还附有一个程序能仅仅用来检查上述 $EPROM$ 内容是否都是"FF"而不进行程序写入，这样使用者可以方便地检查 $EPROM$。

在编写程序写入程序时，有二点要加以注意：

1. 避免母线冲突。
2. 按照 $EPROM$ 程序写入时序图保持合适的写入次序。

下面以 $EPROM$ 2732为例加以说明，其时序图表示在图 6 上。

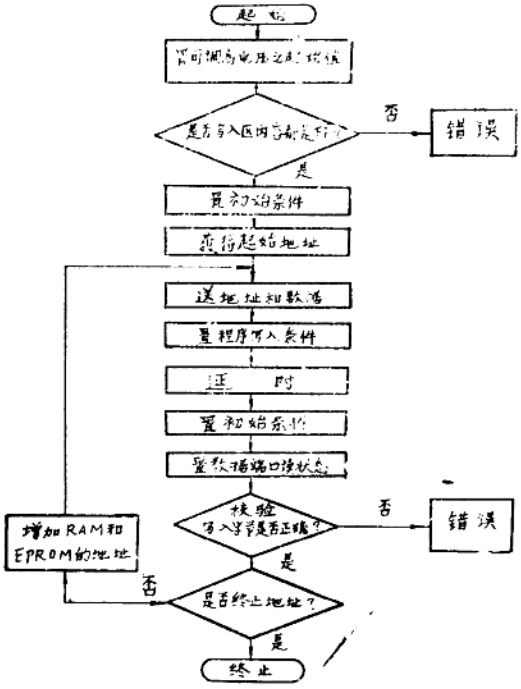

图 5　程序写入程序流图

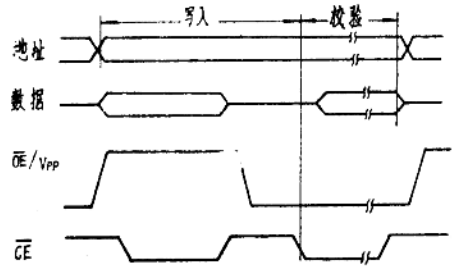

图 6　$EPROM$ 2732程序写入时序图

当程序写入后，为了检查就需要读出所写入的内容并与 RAM 中之内容相比较，这时就要把数据端口由输出模式转为输入模式。设想假如为了读 $EPROM$ 2732 首先让 $\overline{CE}$ 变低使片子选上，这时由于 $\overline{OE}=0$，输出可能，$EPROM$ 就要输出数据到数据母线上。如这时数据端口仍为输出模式，则保存在端口缓冲器内的数据继续输出到数据母线上。这时发生二

者同时输出到数据母线上的问题亦即所谓母线冲突，它可能损坏器件。为此必须让端口由输出转为输入模式先于$\overline{CE}$变低，从而避免母线冲突。通常为了安全起见总是把输入状态作为双向端口的初始状态。

結　　語

此 *EPROM* 程序写入器经长期运行，工作正常并具有较大的灵活性，适用于实验室以及无专用开发系统等場所，其本身配上终端亦具有一定的开发能力。本系统如进一步再配上电流源等可以扩展用以给双极型 *PROM* 进行程序写入。目前由于場效应管速度较慢不能适应某些高速运行的要求，故双极型*PROM*已获得较广泛的应用。

BH-81实验数据库管理系统及其实现*

计算机科学与工程系　李隆江、潘升庆、张静英、李昭智
制造工程系　唐梓荣

引　言

在数据库理论和技术中占有非常重要地位的 *CODASYL* 方法适于处理网状结构数据库。它允许记录类型之间有网状关系。

以研究生为主体的北京航空学院*BH*81研制小组以约三个人年在 *FELIX* 机上实现了 *CODASYL* 方法的基本思想，研制出实验数据库管理系统。总的说来，无论在数据描述语言方面，还是在数据操纵语言方面，*BH*81都是*CODASYL* 方法的子集，但是前者在有的方面扩充了后者。*BH*81不仅允许用户进行*CODASYL*提出的数据操纵，而且可以直接使用数据操纵语句，这和常见的用 *CALL* 语句间接地进行数据操纵是不同的。这大大方便了应用程序员的程序设计工作。此外，*BH*81 以*FORTRAN* 语言做为主语言，以适应*FORTRAN*程序员存取数据库的需要。上述两点是*BH*81系统的两个主要特点。

系统的用户界面

*BH*81实验数据库管理系统由三部分组成，它们是：

*DDL*编译程序

*DML*预编译程序（预处理程序）

功能子程序——*DBCS*（数据库控制系统）

和 *DBMS* 打交道的有三种人：*DBA*（数据库管理员）、应用程序员和参数程序员。*DBA*把用*DDL* 语言写的源模式交给*DDL*编译程序处理，产生目标模式(数据库结构表)。应用程序员把用 *DML/FORTRAM*写的应用程序交给 *DML* 预处理程序处理，产生由 *FORTRAN* 语句组成的二次源程序，由一个专用子程序把它们送入源程序库，然后经过 *FORTRAN*编译，连接编辑（包括把功能子程序编入目标程序），变成可以执行的目标程序（*IMT*）。目标程序执行应用程序员的意图，包括处理可能有的数据卡片，在行式打印机上给出答案。应用程序员要做的工作只是把自己的*DML/FORTRAN*程序和数据交给 *DML* 预编译程序处理，以下各阶段都是自动进行的，如图 1 所示。

参数程序员实际上是非程序员用户，他们使用简单的 *CALL* 语句（填好参数）向*DBMS*发出命令，这些命令经由已经入库的应用子程序解释后由 *DBMS* 执行。本文最后要介绍的两个子系统就是为参数程序员服务的。

---

* 本文是*BH*81系统的综合介绍。关于*BH*81 的*DDL*，*DML*，*DBCS* 张静英、李昭智、潘升庆分别有报告论述。

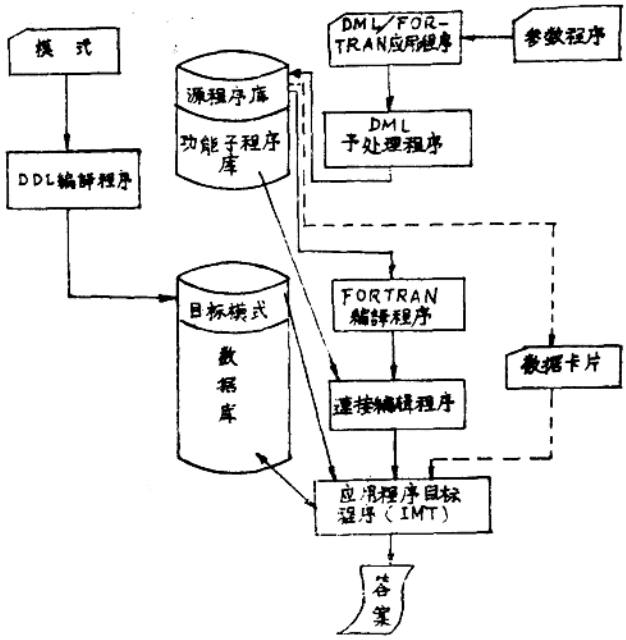

图 1

数据描述語言——DDL

一个数据库的数据结构由模式来描述。模式有外模式、概念模式和内模式三种。外模式也叫子模式。*BH*81的*DD*81语言用来定义数据库的概念模式和内模式（统称模式）。

*BH*81沒有引入子模式概念。用戶用主语言声明*UWA*（用戶工作区）时，只选择和自己有关的记录类型声明记录域，这相当于从模式中取子集。从这个意义上讲，这种做法也是定义子模式的一种形式。在这一点上，*BH*81和*TOTAL*系统是类似的。当然这种做法并不产生目标子模式，应用程序访问数据库时查询的仍是目标模式。

因此，*BH*81系统的数据描述语言不包括子模式 *DDL*，只有模式 *DDL*，简称*DDL*。它有三种条目，八种语句（三种条目出现的顺序和每种条目中各语句的出现顺序都是固定的），如表 1 所示。

*BH*81的*DDL*是一种型*Chomsky*Ⅱ语言，所以编译程序在句法分析时采用有限状态算法进行分析。以*LOCATION*语句为例，其有限状态表如表 2 所示。

源模式由 *DDL* 编译程序进行句法分析后，若无错误则产生目标模式。目标模式是一个数据库数据结构表，它分为 5 个区：主目录区，记录类型索引表，记录结构描述区，系结构描述区，链描述区。

表1 *DDL*语句表

条目	语句名	语句书写格式	功能
模式条目	*SCHEMA*语句	*SCHEMA* 模式名 记录类型数	标识数据库模式名
记录条目	*RECORD*语句	*RECORD* 记录名 记录数	命名一个记录类型
	*LOCATION*语句	*LOCATION MODE* { *SYSTEM* / *CALC* 库过程名 / *USING* 数据项名 }	规定记录值在库中的存放方式
	ITEM 语句	*ITEM* 数据项名 *TYPE* { *FW.d* / *IW* / *AW* }	命名数据项并给出其类型
系条目	*SET* 语句	*SET* 系名	命名一个系类型
	OWNER 语句	*OWNER* { *SYSTEM* / 记录名 }	指定系主记录类型
	*MEMBER*语句	*MEMBER* 记录名	指定成员记录类型
	ORDER 语句	*ORDER* { *NEXT* / *SORTED BY* { *ASCENDING* / *DESCENDING* / 数据项名 } }	指定系内排序规则

表2 *LOCATION*语句的限状态表

序号	*LOCATION*	*MODE*	*SYSTEM*	*CALC*	名字	*USING*	*A*
1	②						
2		③					
3			⑦	④			
4					⑤		
5						⑥	
6					⑦		
7							口出

数据操纵語言——DML

数据操纵语言（*DML*）的作用是对数据库进行存贮、删除、更新和检索操作。

*BH*81允许用戶直接使用 *DML* 语句实现上述各种操作。这些语句经过预编译转换为*FORTRAN*语句序列，主要是一个调用功能子程序的*CALL*语句。这样做，和常见的*CALL*连接模块的方式相比，有如下优点：

(1) 应用程序员编程序较为容易。

(2) 预处理时可以发现句法错误和某些逻辑错误，从而能够避免应用程序的无意义的编译、连接编辑和运行。

*BH*81的*DML*语句取自*CODASYL*的*COBL JOD*文本（2，5），不同之处是：

(1) 语句格式有所改变，使其接近*FORTRAN*程序书写格式。

(2) 舍去了与保密性、并发操作有关的语句和用主语言*FORTRAN*容易实现的语句。

(3) 增加了适应*FORTRAN*模块式结构的*DBSUB*语句，增加了主语言*FORTRAN*不易实现的字符串传输语句*MOVE*。

(4) 增加了包括六种关系符（大于，等于，小于，不大于，不等于，不小于）在内的属性数值条件的*FIND SUCH*语句，从而增强检索功能。其格式如下：

FIND SUCH 记录名（数据项名1 关系符1 { 变量1 / 常量1 }

[数据项名2 关系符2 { 变量2 / 常量2 }]……）/系名

表3是*BH*81的*DML*语句表。此表除了给出*BH*81的诸*DML*语句名称和它们的功能之外，还给出了对应的*COBOL JOD*的*DML*语句名称，以便于比较。

DML 预处理程序的词法分析，对于保留字的检查方式和一般的编译程序是一样的，对于名字（包括记录名、数据项名、系名）的检查则直接和目标模式（数据结构表）的对应项进行比较，有定义的为正确，无定义的为错误，这样既提高了处理速度，又能准确指示错误。对于在*DML*语句中的*FORTRAN*语言元素（如变量名、数组名、常数、语句标号）则不做任何检查。这样做既提高了预处理效率，也不会漏掉可能出现的错误。

句法分析也是采用有限状态算法，这一点和*DDL*编译程序是一样的。

代码生成采用“预制框架”式。由于每一种语句生成的目标（*FORTRAN*语句序列，主要的是一个 *CALL* 语句）都有一定的格式，预处理准备了各种不同的“框架”，上面镶嵌着目标语句中不变的部分，可变部分则由预处理程序分析后填入适当的参数。

生成的目标语句序列中的 *CALL* 语句的参数通常已经不是记录名、系名、数据项名，而是记录类型号、系类型号、数据项号。也就是说，由名换号的工作是在预处理阶段完成的，而不是在运行阶段完成的。实现由名换号要访问目标模式，如果在预处理阶段完成，只需要做一次；如果在运行阶段完成，则需要做$n(n\geqslant 1)$次，这也就是两度访问模式带来时间收益的原因。

表3 DML语句表

序 号	BH81	COBOL JOD	基 本 功 能
1	READY	READY	打开数据库文件，为用户准备系统通讯单元等
2	FINISH	FINISH	结束对数据库的使用，关闭数据库文件
3	STORE	STORE	向数据库存贮一个记录值
4	ERASE	ERASE	从数据库中删除一个记录
5	FIND	FIND	在库中寻址一个记录，使它成为运行单位当前值
6	GET	GET	把运行单位当前值记录从系统缓冲区送到用户工作区
7	MODIFY	MODIFY	向库中回送一个修改的记录
8	CONNECT	CONNECT	将一个记录插入到一个指定的系值中
9	DISCONNECT	DISCONNECT	把一个记录从指定系值中移出
10	MOVE		把一个字符串送到指定记录域的数据项中
11	DBSUB		构造含有DML 语句的子程序
12	DATA		控制处理用户程序中的数据卡
13	DATAEND		同 上
14		ORDER	
15		ACCEPT	
16		IF	
17		USE	
18		KEEP	
19		FREE	
20		REMONITOR	