4. 语义分析

针对 TEST 语言，为了提高可读性和简化代码生成过程，采用一台抽象的栈式计算机作为目标机，生成的目标代码是抽象机的汇编指令代码。本章将对 TEST 语言的翻译采用递归下降的属性翻译。

4.1 中间代码

源程序的中间代码是在编译程序将高级语言程序翻译为汇编语言或机器代码的过程中产生的,在编译过程中起着桥梁的作用。如果不使用中间代码,若有 M 种语言要在 N 种机器上运行,则需要编制 M×N 套编译程序;而如果采用中间代码,则只需要编制 M+N 套编译程序。使用中间代码有如下优点。

(1)在生成中间代码时,可以不考虑机器的特性,使得编制生成中间代码的编译程序变得较为简单;

(2)由于中间代码形式与具体机器无关,所以,生成中间代码的编译程序能很方便地被移植到别的机器上,只需要为该中间代码开发一个解释器或者将中间代码翻译成目标机指令就能在目标机上运行;

(3)在中间代码上更便于做优化处理。

使用中间代码的主要缺点是:编译的效率比直接产生机器码编译的效率要低一些。

中间代码的选择也是一个重要的研究课题。人们期望能找到这样一种中间语言,它既适合于将各种高级程序设计语言翻译成该中间语言,又能比较方便地将该中间语言翻译成各种类型机器的目标语言。目前,中间代码有很多种,包括波兰后

缀表示、N-元表示、抽象语法树、抽象机代码等。

4.2 栈式抽象机及其汇编指令

目标平台采用的机器是一台抽象的栈式计算机，它用一个栈来保存操作数，并有足够的内存空间。该抽象机的常用汇编指令如下：

LOAD	D	将 D 中的内容加载到操作数栈;
LOADI	常量	将常量压入操作数栈;
STO	D	将操作数栈栈顶单元内容存入 D,且栈顶单元内容保持不变;
POP		将操作数栈栈顶出栈;
ADD		将次栈顶单元与栈顶单元内容出栈并相加,和置于栈顶;
SUB		将次栈顶单元与栈顶单元内容出栈并相减,差置于栈顶;
MULT		将次栈顶单元与栈顶单元内容出栈并相乘,积置于栈顶;
DIV		将次栈顶单元与栈顶单元内容出栈并相除,商置于栈顶;
BR	lab	无条件转移到 lab
BRF	lab	检查栈顶单元逻辑值并出栈,若为假(0)则转移到 lab;
EQ		将栈顶两单元做等于比较并出栈,并将结果真或假(1 或 0)置于栈顶;
NOTEQ		将栈顶两单元做不等于比较并出栈,并将结果真或假(1 或 0)置于栈顶;
GT		次栈顶大于栈顶操作数并出栈,则栈顶置 1,否则置 0;
LES		次栈顶小于栈顶操作数并出栈,则栈顶置 1,否则置 0;
GE		次栈顶大于等于栈顶操作数并出栈,则栈顶置 1,否则置 0
LE		次栈顶小于等于栈顶操作数并出栈,则栈顶置 1,否则置 0;
AND		将栈顶两单元做逻辑与运算并出栈,并将结果真或假(1 或 0)置于栈顶;
OR		将栈顶两单元做逻辑或运算并出栈,并将结果真或假(1 或 0)置于栈顶;
NOT		将栈顶的逻辑值取反;
IN		从标准输入设备(键盘)读入一个整型数据,并入操作数栈;
OUT		将栈顶单元内容出栈,并输出到标准输出设备上(显示器);
STOP		停止执行;

例如,有下段程序:

```
int a,b;
a=10;
b=20*a;
```

假设 a、b 在内存中存储地址分别为 0 和 2，则相对应的抽象机汇编程序如下:

```
LOADI   10
STO     0
POP
LOADI   20
LOAD    0
MULT
STO     2
POP
```

4.3 翻译方案

下面将针对各个语法规则制定翻译方案（语法规则中加入花括号括起来的语义动作），完成将程序语言转化为以上栈式抽象机的汇编指令。

（1） 声明的处理

符号表用来存储变量信息，由于 TEST 语言的变量只有整型，可以按如下建立一个数组作为符号表：

```
struct {
    char name[16];
    int address;
}vartable[max_var_num];
int datap = 0;
```

其中 name 保存变量名，address 保存变量地址，vartable 是符号表。由于是目标机，且只有整型变量，则符号表中每插入一个变量，则 datap 加一，用于指向最后一个变量。

针对声明语法规则，其翻译方案如下：

<declaration_stat> → int ID {name-def function}

其中 name-def function 完成在符号表中插入当前变量：首先查询符号表，

从表中最后一个变量开始向前查找直到第一个变量，如果找到了当前这个变量，则报重复定义的错误，否则，将当前变量插到符号表的最后。

（2） 表达式语句

其翻译方案为：

<expression_stat>→<expression> {POP;}

因为表达式的计算结果会保留在操作数栈的栈顶，因此表达式语句的翻译方案只是在表达式后面加上动作 POP，将栈顶内容弹出栈。

关于 expression，其翻译方案如下：

<expression>→ ID {LOOKUP(n,d);ASSIGN;} = <bool_expr> {STO d} | < bool_expr >
<bool_expr> → <additive_expr>
| <additive_expr> > <additive_expr> {GT}
| <additive_expr> < <additive_expr> {LES}
| <additive_expr> >= <additive_expr> {GE}
| <additive_expr> <= <additive_expr> {LE}
| <additive_expr> == <additive_expr> {EQ}
| <additive_expr> != <additive_expr> {NOTEQ}
<additive_expr> → <term> {+<term>{ADD} | - <TERM> {SUB} }
<term> → <factor> {* <factor> {MULT} | / <factor> {DIV} }
<factor> → (<expression >) | ID {LOOKUP(n,d); LOAD d;} | NUM {LOADI I;}

翻译方案中的各个动作解释如下：

LOOKUP(n,d)：在符号表中查找当前变量 n，返回地址 d；若没有，则变量未定义，报错。

ASSIGN：超前读一个符号，如果是=，则表示进入赋值表达式；如果不是=，则选择<bool_expr>，然后将超前读的符号退回。

STO d：输出指令代码 STO d;

LOAD d：输出指令代码 LOAD d;

LOADI i：输出指令代码 LOADI i;

GT、ADD 等：输出指令代码 GT、ADD 等。

（3） if 语句

其翻译方案为：

<if_stat>→if(<expression>) {BRF(label1);} <statement> {BR(label2); SETLabel(label1);}
[else<statement>] {SETLabel(label2);}

翻译方案中的各个动作解释如下：

BRF(label1)：输出 BRF label1

BR(label2)：输出 BR label2

SETLabel(label1)：设置标号 label1

例如 TEST 语言的一个语句：if(a>5) a=1; else a=2;

按照以上翻译方案，设当前语句标号的顺序为 1，则应产生下列目标代码：

```
    LOAD        0           //表达式 a>5 的代码
    LOADI       5
    GT
    BRF     LABEL1          //执行动作 BRF(label1)
    LOADI       1           //a=1;的代码
    STO         0
    POP
    BR          LABEL2      //执行动作 BR(label2)
LABEL1:                     //执行动作 SETLabel(label1)，设置标号 label1
    LOADI       2
    STO         0
```

POP

LABEL2:

（4） while 语句

其翻译方案为：

<while_stat>→while{SETLabel(label1);} (<expression>) {BRF(label2);}

<statement> {BR(label1); SETLabel(label2);}

例如，有 TEST 语句：while(a<3) a = a+2;,假设目前的语句标号的顺序为 2，则上面的翻译方案应产生下列代码：

LABEL2:

LOAD 0

LOADI 3

LES

BRF LABEL3

LOAD 0

LOADI 2

ADD

STO 0

POP

BR LABEL2

LABEL3:

（5） for 循环语句

其翻译方案为：

<for_stat>→for(<expression>{POP;} ;

{SETLabel(label1);} <expression> {BRF(label2);BR(label3);} ;

{SETLabel(label4);} <expression> {POP; BR(label1);})

{SETLabel(label3);} <statement> {BR(label4); SETLabel(label2);}

例如，有 TEST 语句：

for(i=1; i<3; i=i+1) a = a + 10;

假设目前的语句标号的顺序为 6，则以上翻译方案应产生下列代码：

```
    LOADI       1
    STO         2           //i 的地址是 2
    POP
LABEL6:                     //label1
    LOAD        2
    LOADI       3
    LES
    BRF         LABEL7      //label2
    BR          LABEL8      //label3
LABEL9:                     //label4
    LOAD        2
    LOADI       1
    ADD
```

```
    STO     2
    POP
    BR      LABEL6      //label1
LABEL8:                 //label3
    LOAD    0           //a 的地址是 0
    LOADI   10
    ADD
    STO     0
    POP
    BR      LABEL9      //label4
LABEL7:
```

（6） read 语句

其翻译方案如下：

<read_stat>→ read ID {LOOKUP(n,d); IN; STO(d); POP}

其中的某些动作解释如下：

LOOKUP(n,d)：在符号表中查找当前变量 n，返回地址 d；若没有，则变量未定义，报错。

IN：输出指令代码 IN

STO(d)：输出指令代码 STO d

（7） write 语句

其翻译方案如下：

<write_stat> → write <expression> {OUT;}

其中动作符号解释如下：

OUT：输出指令代码 OUT。