

C 语言开关语句的自动翻译*

侯济恭 余 坚

(华侨大学计算机科学系, 泉州 362011)

摘要 介绍 C 编译器开关语句目标结构、生成算法及实现该算法的程序, 讨论使尾码的生成非常简单且与目标机结构完全无关的自动移植方案。

关键词 编译系统, 代码自动生成, 开关语句, C 语言

分类号 TP 314

移植 C 语言编译程序主要工作有: 函数头尾码生成; 开关语句的转移控制代码生成(尾码); 变量初始化。这一部分工作在 PASS ONE 中完成。在 PASS TWO 主要工作是对模式匹配表的修改。在文[1]中, 我们讨论了函数头尾码生成。本文将针对单片机(51 系列)的结构, 讨论开关语句的目标代码结构, 转移控制代码结构及其生成算法。最后, 提出一个自动生成开关语句的目标代码方案, 由于其与目标机完全无关, 故称之为“自动生成”。

1 开关语句的目标结构

C 语言的开关语句一般形式与其目标代码结构如图 1 所示。

```
switch (e) {  
  case a1 : S11; ...; S1n; break;  
  case a2 : S21; ...; S2n; break;  
  case a3 : S31; ...; S3n; break;  
  :  
  :  
  :  
  default : S1; ...; Sn; break;  
}
```

各部份解说如次: (a) 头码: 关于选择子 e 的目标代码; (b) 开关体: 各个选择分支(case)的目标代码; (c) 尾码: 条件判断目标码, 控制转移至相对应的条件分支代码。

在 VAX-11 机中, 头码、开关体码由 PASS TWO 生成, 而尾码则由 PASS ONE 生成, 其形式有以下三种。

(1) 紧凑形式。这是一种与目标机结构高度相关的代码, 目的是为了提

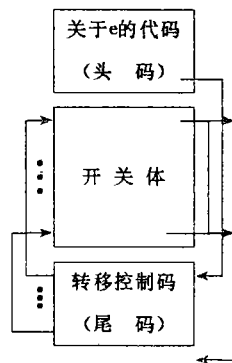


图1 switch 目标结构

* 本文 1994-04-09 收到; 福建省自然科学基金项目

接利用 VAX 机的 case1 指令

CASE1 R0, min-val, RANGE

生成一个转移表. 其中 R0 为选择子 e 的值, min-val 是选择条件最小值, RANGE 是选择条件取值范围. 紧凑翻译最大的问题是其顺序性太强(隐式排序: 最 min, min+1, ..., max). 例如, 如果选择条件之间步距太大, 则可能产生许多空支(C 编译把所有的选择条件均转换为整型量处理, 故然). 如目标代码为

```
switch(e) {
case 1 : ... ;break;
case 2 : ... ;break;
case 3 : ... ;break;
:
case 10 : ... ;break;
default : ... ;break;
}
case1 R0, $1, $9 · L22;
· word · L17 - · L22; 分支 1 入口 · L17
· word · L19 - · L22; 无分支 2 转共同入口 · L19
· word · L19 - · L22; 无分支 3 转共同入口 · L19
:
· word · L20 - · L22; 分支 10 转处理入口 · L20
· jbr · L19 ; 缺省分支入口 · L19
```

因此, 当 RANGE 的值与选择分支的比大于 3 时, C 编译就放弃这种翻译.

(2) 堆式. 所谓的堆式, 实际上就是采用二叉判定树方法生成目标代码. 其方法: 令选择条件生成一棵二叉判定树; (b) 前序遍历该树, 生成目标代码. 其目标码基本结构为

```
CMPL R0, $a ; a 为分支条件
JEGL · La1; 相等转 · La1
JGTR · Ly1; 大于转下一分支
CMPL R0, $b ; 小于处理分支
```

堆式算法的主要优点是定位快捷, 并且解决了紧凑方式的空支转移问题. 但是对于每一个 case 分支, 堆式算法至少需要 3 条目标指令, 随着 case 的增多, 目标指令以几何级数增长, 代码长度令人难以接受.

(3) 简式. 对于分支小于 8 个的选择条件, 则用最简单的判定方式产生目标代码, 其目标代码结构为

```
CMPL R0, $a ;
JEGL · La ; 相同转 · La 处理
CMPL R0, $b ;
JEGL · Lb ;
:
```

JBR · Ldefault ;转缺省处理

2 目标代码结构修正

C编译翻译开关语句的过程:(1)识别 $\text{switch}(e)$,生成关于 e 的头码部分;(2)每识别一个 $\text{case } a_n$,则产生一个标号序号 Li ,将 n 和 i 值填入开关表中(图2);(3)重复(2)直到 switch 结束;(4)将开关表以 n 为键值冒泡排序(升序);(5)扫描开关代码,逐一生成转移控制代码(尾码)。

根据单片机的存贮结构^[1]以及C编译的代码生成算法,我们将 switch 的目标代码修正如图3。使转移控制成为一个与目标代码完全无关的独立结构:头码、开关体、开关表和转移控制。头码和开关体保留原结构,开关表则为图2的映象,其形式为

```

Sn: DB a1 ;常数 a1
    DW L1 ;转移标号 L1
    DB a2 ;常数 a2
    DW L2 ;转移标号 L2
    :
    DB an ;常数 an
    DW Ln ;转移标号 Ln

```

value	label
缺省	Lo
a1	i
a2	i+1
ak	i+k

图2 开关表(已排序)

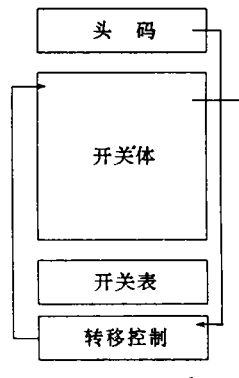


图3 修正目标结构

由此,原来的尾码生成(5)变为:(a)将开关表写入目标文件;(b)将转移控制代码写入目标文件。

3 交叉编译开关语句代码生成

根据以上的讨论,可得本C编译的开关语句尾码(转移表+转移控制)生成算法为(入口:表长 n , e 值存于累加器 A):(1)从扫描开关表判断值域是否与 A 相等;(2)相等,则从 S_n 中取出相应的入口,间址转移;(3)重复(2),直至表尾;(4)转缺省入口。

显然,我们可以采用任何一种扫描算法,以增快扫描速度。在这里,转移控制代码是独立于任何源程序,因此可以随移植者的意愿而定。以下是本交叉C编译器的目标代码生成函数及其目标代码,源程序结构为

```

switch(e){
case 10: ... ;break;
case 2: ... ;break;
case 1: ... ;break;
case 20: ... ;break;
case 5: ... ;break;
default : ... ;
}

```

以下是目标代码生成函数

```

gen-switch-3(struct sw * p,int n,int lab,FILE * fp){
int i,table,entry;

```

```

int loop,next,out;
able=getlab(); /* 生成表标号 */
fprintf(fp,"S%d:", table); /* 生成转移表 */
for(i=n-1;i>=0;i--){
    fprintf(fp,"\t DB%d\n",p[i],sval);
    fprintf(fp,"\t DW L%d\n",p[i],slab); /* 生成转移控制代码 */
}
fprintf(fp," L%d:\t MOV DPTR, #S%d\n", lab,table);
fprintf(fp," \t MOV RI, # %d\n",);
fprintf(fp," \t MOV B, A\n");
fprintf(fp," S%d:" loop=getlab());
fprintf(fp," \t MOVX A @ DPTR, \n");
fprintf(fp," \t INC DPTR, \n");
fprintf(fp," \t CJNE A,B,S%d\n",next=getlab());
fprintf(fp," \t SJMP S%d\n",out=getlab());
fprintf(fp," S%d: \t INC DPTR\n",next);
fprintf(fp," \t INC DPTR\n", next);
fprintf(fp," \t DJNZ R1,S% d\n", loop);
fprintf(fp," \t JMP L% d\n", lab);
fprintf(fp," S%d:\t MOVX A,@ DPTR\n",out,n);
fprintf(fp," \t PUSH A\n");
fprintf(fp," \t INC DPTR\n");
fprintf(fp," \t MOVX A,@DPTR\n");
fprintf(fp," \t MOV DPH,A\n");
fprintf(fp," \t POP DPL\n");
fprintf(fp," \t CLR A\n");
fprintf(fp," \t JMP @A+DPTR\n");}

```

以下是生成的目标代码

L29: DB 20	L28: MOV DPTR ,#S29
DW L30	MOV R1,#5
DB 10	MOV B,A
DW L31	S30: MOVX A,@ DPTR
DB 5	INC DPTR
DW L35	CJNE A,B,S31
DB 2	SJMP S32
DW L32	S31: INC DPTR
DB 1	INC DPTR
DW L34	DJNZ R1,S30

```

S32:  MOVX A,@DPTR
      PUSH A
      INC DPTR
      MOVX A,@DPTR
      MOV DPH ,A
      POP DPL
      CLR A
      JMP @A+DPTR

```

4 开关语句自动生成研究

以上讨论的移植方案虽然具有独立性,但是移植时还要人工修改.事实上,开关语句完全可以实现自动生成.我们知道,在汇编程序中,都有一条间址转移指令,如在 51 机中,间址转移指令为 `JMP @A+DPTR`,其中 `DPTR` 为间址指针, `A` 为偏移量.可惜的是,号称“高级汇编”的 C 语言没有提供此类命令,但是,我们可以经过在线汇编解决这一问题,不妨称此为半自动移植.其方法:(1)利用 C 语言的分程序功能,产生如下源程序段为

```

{
struct{ int sval;
      int slab;
      }table[]={e, L0,a1,L1,a2,L2,...}
while(--n>=0)
    if(table[n]==e){
        asm MOV DPTR, table[i].slab
        asm MOV JMP @A+DPTR
    }
}

```

(2) 将以上程序段嵌入当前编译缓冲区.

以上方法最终还得涉及到具体的目标机,最彻底的办法是对 C 语言作一个补充,亦即修正 `goto` 语句的语义,使其具有间址转移功能.为此,必须为 C 增加一个新的数据类型 `label`. 即

`label L1,L2, *PL;`

说明 `L1,L2` 为标号,而 `PL` 为标号指针.于是有

`PL=L1; goto *PL;`

即可转到相应的标号处执行.据此设想,开关语句的尾码部份便可写为

```

{
struct { int sval;
      label *slab;
      }table[]={e, L0,a1,L1,...,an, Ln};
while (--n>=0)
    if (table[i].sval==e) goto *table[i].slab;
}

```

}

经过以上修正,开关语句的尾码生成事实上只是填写 table,其翻译完全与目标机无关,因此就可以实现自动移植.当然,这一修改涉及 C 的语法,语义子程序的修改,所幸这一修改相当容易,几乎可以独立存在,因而本方案是完全可行的.

5 结束语

本文所讨论的移植方案,部份已经实现,所生成的目标代码确实可行.部分(自动移植)尚在研制之中.我们期望能引起同行的兴趣.

参 考 文 献

- 1 侯济恭,余 坚,李铮铮.单片机 C 语言交叉编译器存储分配研究.华侨大学学报(自然科学版),1994,15(1): 102~106
- 2 侯济恭.C 语言编译器结构分析.小型微型计算机系统 1992,(4): 98~101
- 3 史树民.等程序设计语言 C 编译系统国产化总体设计.计算机工程与应用,1992,(4): 73~76

Autocoding of Switching Statement in C Language

Hou Jigong Yu Jian

(Dept. of Computer Science, Huaqiao Univ., 362011, Quanzhou)

Abstract The authors discuss the automatic generation of the terminal code of compiler switching statement. The presentation include the object structure of switching statement, generating algorithm, and program for implementing that algorithm; and finally, the autocoding scheme which makes terminal code generation very simple and machine independent.

Keywords compiling system, auto coding, switching statement, C language