

中间代码生成

(3. 其它)

魏恒峰

hfwei@nju.edu.cn

2021 年 12 月 25 日



```
switch (  $E$  ) {  
    case  $V_1$ :  $S_1$   
    case  $V_2$ :  $S_2$   
        ...  
    case  $V_{n-1}$ :  $S_{n-1}$   
    default:  $S_n$   
}
```

非 C 语言语义 (break)

```

switch (  $E$  ) {
    case  $V_1$ :  $S_1$ 
    case  $V_2$ :  $S_2$ 
        ...
    case  $V_{n-1}$ :  $S_{n-1}$ 
    default:  $S_n$ 
}

```

非 C 语言语义 (break)

```

code to evaluate  $E$  into  $t$ 
goto test
L1:    code for  $S_1$ 
        goto next
L2:    code for  $S_2$ 
        goto next
...
Ln-1:  code for  $S_{n-1}$ 
        goto next
Ln:    code for  $S_n$ 
        goto next
test:   if  $t = V_1$  goto L1
        if  $t = V_2$  goto L2
        ...
        if  $t = V_{n-1}$  goto Ln-1
        goto Ln
next:

```

```

switch (  $E$  ) {
    case  $V_1$ :  $S_1$ 
    case  $V_2$ :  $S_2$ 
        ...
    case  $V_{n-1}$ :  $S_{n-1}$ 
    default:  $S_n$ 
}

```

$V_i : L_i$ 队列

```

code to evaluate  $E$  into  $t$ 
goto test
L1:   code for  $S_1$ 
      goto next
L2:   code for  $S_2$ 
      goto next
...
Ln-1: code for  $S_{n-1}$ 
      goto next
Ln:   code for  $S_n$ 
      goto next
test:  if  $t = V_1$  goto L1
      if  $t = V_2$  goto L2
      ...
      if  $t = V_{n-1}$  goto Ln-1
      goto Ln
next:

```

```
code to evaluate  $E$  into  $t$   
goto test
```

```
L1: code for  $S_1$   
goto next  
L2: code for  $S_2$   
goto next  
...  
L $n-1$ : code for  $S_{n-1}$   
goto next  
L $n$ : code for  $S_n$   
goto next
```

```
test: if  $t = V_1$  goto L1  
if  $t = V_2$  goto L2  
...  
if  $t = V_{n-1}$  goto L $n-1$   
goto L $n$ 
```

```
next:
```

```
case  $t$   $V_1$  L1  
case  $t$   $V_2$  L2  
...  
case  $t$   $V_{n-1}$  L $n-1$   
case  $t$   $t$  L $n$   
next:
```

case 三地址代码

```

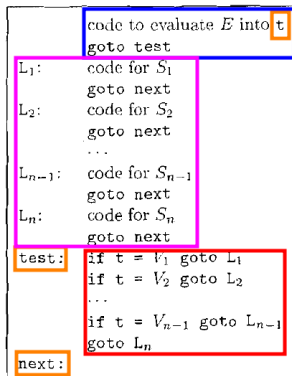
code to evaluate  $E$  into  $t$ 
goto test
L1:   code for  $S_1$ 
      goto next
L2:   code for  $S_2$ 
      goto next
...
L $n-1$ : code for  $S_{n-1}$ 
      goto next
L $n$ :  code for  $S_n$ 
      goto next
test:  if  $t = V_1$  goto L1
      if  $t = V_2$  goto L2
      ...
      if  $t = V_{n-1}$  goto L $n-1$ 
      goto L $n$ 
next:

```

```

case  $t V_1$  L1
case  $t V_2$  L2
...
case  $t V_{n-1}$  L $n-1$ 
case  $t t$  L $n$ 
next:

```



```

case  $t = V_1$  L1
case  $t = V_2$  L2
...
case  $t = V_{n-1}$  L $n-1$ 
case  $t = t$  L $n$ 
next:

```

Jump Table Structure

C code:

```

switch(x) {
  case 1: <some code>
          break;
  case 2: <some code>
          break;
  case 3: <some code>
          break;
  case 5: <some code>
          break;
  case 6: <some code>
          break;
  default: <some code>
}

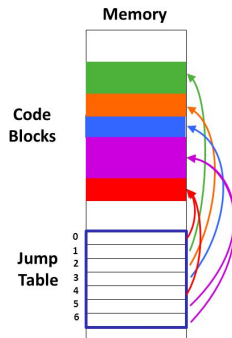
```

We can use the jump table when $x \leq 6$:

```

if (x <= 6)
  target = JTab[x];
  goto *target;
else
  goto default;

```



Winter 2013

x86 Programming III

5

Jump Table 优化

函数/过程的中间代码翻译

$n = f(a[i])$

- 1) $t_1 = i * 4$
- 2) $t_2 = a[t_1]$
- 3) **param t_2**
- 4) $t_3 =$ **call f, 1**
- 5) $n = t_3$

新增文法以支持函数定义与调用

$$\begin{aligned} D &\rightarrow \text{define } T \text{ id } (F) \{ S \} \\ F &\rightarrow \epsilon \mid T \text{ id } , F \\ S &\rightarrow \text{return } E ; \\ E &\rightarrow \text{id } (A) \\ A &\rightarrow \epsilon \mid E , A \end{aligned}$$

函数定义

$$\begin{array}{ll} D & \rightarrow \text{define } T \text{ id } (F) \{ S \} \\ F & \rightarrow \epsilon \mid T \text{ id } , F \\ S & \rightarrow \text{return } E ; \end{array}$$

函数名 `id` 放入当前符号表, 建立新的符号表, 处理形参 F 与函数体 S

函数调用

$$\begin{aligned} E &\rightarrow \text{id} (A) \\ A &\rightarrow \epsilon \mid E , A \end{aligned}$$

```
param  $x_1$   
param  $x_2$   
...  
param  $x_n$   
call  $p, n$ 
```

函数调用

```
S::=CALL id(Elist) { S.code := Elist.code  
A      || gencode("CALL", id.place, Elist.number) }  
Elist::=Elist1, E { Elist.code := E.code || Elist1.code 逆序  
      || gencode("PARAM", E.place);  
      Elist.number := Elist1.number + 1 }  
Elist::=E { Elist.code := E.code || gencode("PARAM", E.place);  
      Elist.number := 1 }
```

C 语言并未规定参数计算的顺序

函数调用

```
S ::= CALL id(Elist) A
{ Count := 0; S.code := Elist.code;
  while NOT EmptyQ(q) do
  begin
    t := HeadQ(q);
    S.code := S.code || gencode("PARAM", t);
    DelQ(q); Count := Count + 1
  end;
  S.code := S.code || gencode("CALL", id.place, Count)
}

Elist ::= Elist1, E { Elist.code := E.code || Elist1.code;
                      EnterQ(E.place, q) }

Elist ::= E { Elist.code := E.code; CreateQ(q);
             EnterQ(E.place, q) }
```

逆序

集中生成 param 指令, 代码更紧凑

Thank
You!



Office 926

hfwei@nju.edu.cn