

SWITCH STATEMENT



Switch语句的汇编代码生成

黄 波

bhuang@dase.ecnu.edu.cn

SOLE

系统优化实验室
华东师范大学

SwitchTable内存放数值 (1)

```
int f(int a){
    switch (a) {
        case 2:
            a=a+1;
            break;
        case 4:
            a=a+2;
            break;
        case 8:
            a=a+3;
            break;
        default:
            a=a+10;
    }
    return a;
}
```

```
@switch.table.f = private unnamed_addr constant [7 x i32] [i32 1, i32 10, i32 2, i32 10, i32 10, i32 10, i32 3], align 4
```

```
; Function Attrs: norecurse nounwind readnone uwtable
define dso_local i32 @f(i32 %0) local_unnamed_addr #0 {
```

```
    %2 = add i32 %0, -2
    %3 = icmp ult i32 %2, 7
    br i1 %3, label %4, label %8
```

```
4:                                ; preds = %1
```

```
    %5 = sext i32 %2 to i64
    %6 = getelementptr inbounds [7 x i32], [7 x i32]* @switch.table.f, i64 0, i64 %5
    %7 = load i32, i32* %6, align 4
    br label %8
```

```
8:                                ; preds = %1, %4
    %9 = phi i32 [ %7, %4 ], [ 10, %1 ]
    %10 = add nsw i32 %9, %0
    ret i32 %10
}
```

Q1: 为什么这个跳转表的元素个数是7?

clang -O1 -S -emit-llvm switch.c

SwitchTable内存放数值 (2)

```
int f(int a){
    switch (a) {
        case 2:
            a=a+1;
            break;
        case 4:
            a=a+2;
            break;
        case 8:
            a=a+3;
            break;
        default:
            a=a+10;
    }
    return a;
}
```

clang -O1 -S switch.c

```
....
# %bb.0:
leal    -2(%rdi), %ecx
        movl    $10, %eax
        cmpl    $6, %ecx
        ja      .LBB0_2
# %bb.1:
        movslq   %ecx, %rax
        movl    .Lswitch.table.f(%rax,4), %eax
.LBB0_2:
        addl    %edi, %eax
        retq
....
.Lswitch.table.f:
        .long    1
        .long    10
        .long    2
        .long    10
        .long    10
        .long    10
        .long    10
        .long    3
        .size    .Lswitch.table.f, 28
```

Q2: 为什么用ja而不用jg?

Q1: 为什么这个跳转表的元素个数是7?

SwitchTable内存放数值地址

```
#include <stdio.h>
```

```
void f(int a)
{
    switch(a){
        case 1: printf("Monday\n"); break;
        case 2: printf("Tuesday\n"); break;
        case 3: printf("Wednesday\n"); break;
        case 4: printf("Thursday\n"); break;
        case 5: printf("Friday\n"); break;
        case 6: printf("Saturday\n"); break;
        case 7: printf("Sunday\n"); break;
        default: printf("error\n"); break;
    }
}
```

clang -O1 -S switch-2.c

```
.type f,@function
f:                                # @f
# %bb.0:
    pushq %rax
    movl %edi, %eax
    addl $-1, %eax
    movl $.Lstr.14, %edi
    cmpl $6, %eax
    ja .LBB0_2
# %bb.1:
    cltq
    movq .Lswitch.table.f(%rax,8), %rdi
.LBB0_2:
    callq puts
    popq %rax
    retq
```

```
.....
.Lswitch.table.f:
    .quad .Lstr.13
    .quad .Lstr.12
    .quad .Lstr.11
    .quad .Lstr.10
    .quad .Lstr.9
    .quad .Lstr.8
    .quad .Lstr
    .size .Lswitch.table.f, 56
```

```
.type .Lstr,@object                # @str
.section .rodata.str1.1,"aMS",@progbits,1
.Lstr:
    .asciz "Sunday"
    .size .Lstr, 7
    .type .Lstr.8,@object          # @str.8
.Lstr.8:
    .asciz "Saturday"
    .size .Lstr.8, 9
    .type .Lstr.9,@object          # @str.9
.Lstr.9:
    .asciz "Friday"
    .size .Lstr.9, 7
    .type .Lstr.10,@object         # @str.10
.Lstr.10:
    .asciz "Thursday"
    .size .Lstr.10, 9
    .type .Lstr.11,@object         # @str.11
.Lstr.11:
    .asciz "Wednesday"
    .size .Lstr.11, 10
    .type .Lstr.12,@object         # @str.12
.Lstr.12:
    .asciz "Tuesday"
    .size .Lstr.12, 8
    .type .Lstr.13,@object         # @str.13
.Lstr.13:
    .asciz "Monday"
    .size .Lstr.13, 7
    .type .Lstr.14,@object         # @str.14
.Lstr.14:
    .asciz "error"
    .size .Lstr.14, 6
    .type .Lswitch.table.f,@object # @switch.table.f
.....
```

SwitchTable内存放代码地址

```
extern int g(int);
extern int k(int);
```

```
int f(int a){
    switch (a) {
```

```
        case 2:
```

```
            a=a+1;
            a=g(a);
            break;
```

```
        case 3:
```

```
            a=a+1;
            a=g(a);
            break;
```

```
        case 4:
```

```
            a=a+100;
            a=k(a);
            break;
```

```
        case 8:
```

```
            a=a+3;
            break;
```

```
        default:
```

```
            a=a+10;
```

```
    }
    return a;
}
```

LLVM IR中有
switch语句，跟
每个case值对应
的是相应代码块
的标号

clang -O1 -S -emit-llvm switch-4.c

```
define dso_local i32 @f(i32 %0) local_unnamed_addr #0 {
    switch i32 %0, label %13 [
        i32 2, label %2
        i32 3, label %5
        i32 4, label %8
        i32 8, label %11
    ]
```

```
2:                                     ; preds = %1
```

```
    %3 = add nsw i32 %0, 1
    %4 = call i32 @g(i32 %3) #2
    br label %15
```

```
5:                                     ; preds = %1
```

```
    %6 = add nsw i32 %0, 1
    %7 = call i32 @g(i32 %6) #2
    br label %15
```

```
8:                                     ; preds = %1
```

```
    %9 = add nsw i32 %0, 100
    %10 = call i32 @k(i32 %9) #2
    br label %15
```

```
11:                                    ; preds = %1
```

```
    %12 = add nsw i32 %0, 3
    br label %15
```

```
13:                                    ; preds = %1
```

```
    %14 = add nsw i32 %0, 10
    br label %15
```

```
15:                                    ; preds = %13, %11, %8, %5, %2
```

```
    %16 = phi i32 [ %14, %13 ], [ %12, %11 ], [ %10, %8 ], [ %7, %5 ], [ %4, %2 ]
    ret i32 %16
```

```
}
```

SwitchTable内存放代码地址

```
extern int g(int);  
extern int k(int);
```

```
int f(int a){  
    switch (a) {  
        case 2:
```

```
        a=a+1;  
        a=g(a);  
        break;
```

```
        case 3:
```

```
        a=a+1;  
        a=g(a);  
        break;
```

```
        case 4:
```

```
        a=a+100;  
        a=k(a);  
        break;
```

```
        case 8:
```

```
        a=a+3;  
        break;
```

```
    default:
```

```
        a=a+10;
```

```
    }  
    return a;  
}
```

clang -O1 -S switch-4.c

跳转表

```
f:                                # @f  
# %bb.0:  
    pushq %rax  
    leal  -2(%rdi), %eax  
    cmpl  $6, %eax  
    ja    .LBB0_5  
# %bb.1:  
    jmpq  *.LJT10_0(%rax,8)  
.LBB0_2:  
    addl  $1, %edi  
    callq g  
    movl  %eax, %edi  
    jmp   .LBB0_6  
.LBB0_3:  
    addl  $100, %edi  
    callq k  
    movl  %eax, %edi  
    jmp   .LBB0_6  
.LBB0_5:  
    addl  $10, %edi  
    jmp   .LBB0_6  
.LBB0_4:  
    addl  $3, %edi  
.LBB0_6:  
    movl  %edi, %eax  
    popq  %rcx  
    retq
```

```
...  
.LJT10_0:  
    quad  .LBB0_2  
    quad  .LBB0_2  
    quad  .LBB0_3  
    quad  .LBB0_5  
    quad  .LBB0_5  
    quad  .LBB0_5  
    quad  .LBB0_4
```

计算跳转偏移量

间接跳转

无跳转表生成示例(1)

```
extern int g(int);
extern int h(int);
```

```
int f(int a){
    switch (a) {
        case 2:
            a=a+1;
            a=g(a);
            break;
        case 4:
            a=a+2;
            a=h(a);
            break;
        case 8:
            a=a+3;
            break;
        default:
            a=a+10;
    }
    return a;
}
```

LLVM IR中有
switch语句，跟
每个case值对应
的是相应代码块
的标号

clang -O1 -S -emit-llvm switch-3.c

```
define dso_local i32 @f(i32 %0) local_unnamed_addr #0 {
    switch i32 %0, label %10 [
        i32 2, label %2
        i32 4, label %5
        i32 8, label %8
    ]
```

```
2:                                     ; preds = %1
```

```
%3 = add nsw i32 %0, 1
%4 = call i32 @g(i32 %3) #2
br label %12
```

```
5:                                     ; preds = %1
```

```
%6 = add nsw i32 %0, 2
%7 = call i32 @h(i32 %6) #2
br label %12
```

```
8:                                     ; preds = %1
```

```
%9 = add nsw i32 %0, 3
br label %12
```

```
10:                                    ; preds = %1
```

```
%11 = add nsw i32 %0, 10
br label %12
```

```
12:                                    ; preds = %10, %8, %5, %2
```

```
%13 = phi i32 [ %11, %10 ], [ %9, %8 ], [ %7, %5 ], [ %4, %2 ]
ret i32 %13
```

```
}
```

无跳转表生成示例(2)

```
extern int g(int);  
extern int h(int);
```

```
int f(int a){  
    switch (a) {  
        case 2:  
            a=a+1;  
            a=g(a);  
            break;  
        case 4:  
            a=a+2;  
            a=h(a);  
            break;  
        case 8:  
            a=a+3;  
            break;  
        default:  
            a=a+10;  
    }  
    return a;  
}
```

clang -O1 -S switch-3.c

最终生成的汇编代码中没有跳转表，switch语句被转换成了逐个数值依次比较！

```
f:                                     # @f  
# %bb.0:  
    pushq %rax  
    movl  %edi, %eax  
    cmpl  $8, %edi  
    je    .LBB0_5  
# %bb.1:  
    cmpl  $4, %eax  
    je    .LBB0_4  
# %bb.2:  
    cmpl  $2, %eax  
    jne   .LBB0_6  
# %bb.3:  
    addl  $1, %eax  
    movl  %eax, %edi  
    callq g  
    popq  %rcx  
    retq  
.LBB0_5:  
    addl  $3, %eax  
    popq  %rcx  
    retq  
.LBB0_4:  
    addl  $2, %eax  
    movl  %eax, %edi  
    callq h  
    popq  %rcx  
    retq  
.LBB0_6:  
    addl  $10, %eax  
    popq  %rcx  
    retq
```


总结

- switch语句可能会被编译器转换成表形式，也可能被转换成一系列的值比较跳转语句
- 编译器判断是否转换成表形式的重要因素 (*tradeoff*)
 - N : case的数目
 - D : case值的最高值和最低值之间的差
 - $N/(D + 1)$: case值的“稠密程度”

