GCC的汇编

quant67

2015 年 1 月 25 日

在课堂上学过汇编语言，便想知道编译器生成的代码长什么样。于是我在fiboRec.c文件写下一些代码：

```
int fiboRec(int n) {
    if (n == 0 || n == 1) return 1;
    return fiboRec(n - 1) + fiboRec(n - 2);
}
```

ccode/fiboRec.c

按照GCC的文档，在ccode目录下键入下面的汇编命令，让编译器产生汇编代码：

```
gcc -S fiboRec.c
```

下面是生成的汇编代码：

```
    .file   "fiboRec.c"
    .text
    .globl  _fiboRec
    .def    _fiboRec;   .scl    2;  .type   32; .endef
_fiboRec:
    pushl   %ebp
    movl    %esp, %ebp
    pushl   %ebx
    subl    $20, %esp
    cmpl    $0, 8(%ebp)
    je  L2
    cmpl    $1, 8(%ebp)
    jne L3
L2:
    movl    $1, %eax
    jmp L4
L3:
    movl    8(%ebp), %eax
    decl    %eax
```

```
    movl    %eax, (%esp)
    call    _fiboRec
    movl    %eax, %ebx
    movl    8(%ebp), %eax
    subl    $2, %eax
    movl    %eax, (%esp)
    call    _fiboRec
    addl    %ebx, %eax
L4:
    addl    $20, %esp
    popl    %ebx
    popl    %ebp
    ret
```

ccode/fiboRec.s

我是学过汇编语言的，连TAOCP上的MIX都知道，但这是什么意思呢？这跟老师教我的根本不一样！这就是为什么要学习GCC的汇编(GAS)。

1 mov后面的l是什么意思？

GAS为不同长度的数据准备了不同版本的代码，区别在于后缀。对于整形变量，一个字节的后缀是b，两个字节后缀是w，四个字节后缀是l。对于浮点数，我们常常使用的float（4字节）后缀是s，double（8字节）后缀是l，long double （$10 \sim 12$字节）后缀是t。

所以，不光有movl，还有movb，movw。

2 寻址方式

Type	**Form**	**value**	**Name**
Immediate	$Imm	Imm	Immediate
Register	E_a	$R[E_a]$	Register
Memory	Imm	M[Imm]	Absolute
Memory	(E_a)	$M[R[E_a]]$	Indirect
Memory	$Imm(E_b)$	$M[Imm + R[E_b]]$	Base+disolacement
Memory	(E_a, E_b)	$M[R[E_a] + R[E_b]]$	Indexed
Memory	$Imm(E_a, E_b)$	$Imm + M[R[E_a] + R[E_b]]$	Indexed
Memory	(E_a, E_b, s)	$M[R[E_a] + R[E_b] \cdot s]$	Scaled indexed
Memory	$Imm(E_a, E_b, s)$	$Imm + M[R[E_a] + R[E_b] \cdot s]$	Scaled indexed

3 数据传送

movl S, D : $D \leftarrow S$。还有movw和movb，后缀的意思在前面已经讲过。需要注意的是movsbl和movzbl，前者“move sign-extended byte”，后者“move zero-extended byte”。

pushl S : $R[\%esp] \leftarrow R[\%esp] - 4$; $M[R[\%esp]] \leftarrow S$。

popl D : $D \leftarrow M[R[\%esp]]$; $R[\%esp] \leftarrow R[\%esp] + 4$。

4 算数运算和逻辑运算

Instruction	**Effect**
leal S, D	$D \leftarrow \&S$
incl D	$D \leftarrow D$
decl D	$D \leftarrow D - 1$
negl D	$D \leftarrow -D$
notl D	$D \leftarrow\sim D$
addl S, D	$D \leftarrow D + S$
subl S, D	$D \leftarrow D - S$
imull S, D	$D \leftarrow DS$
xorl S, D	$D \leftarrow D \otimes S$
orl S, D	$D \leftarrow D\|S$
andl S, D	$D \leftarrow D\&S$
sall k, D	$D \leftarrow D \ll k$
shll k, D	$D \leftarrow D \ll k$
sarl k, D	$D \leftarrow D \gg k$(Arithmetic)
shrl k, D	$D \leftarrow D \gg k$(logical)

还有一些特殊的运算指令：

Instruction	**Effect**	**Description**
imull S	$R[\%edx] : R[\%eax] \leftarrow S \times R[\%eax]$	signed full multiply
mull S	$R[\%edx] : R[\%eax] \leftarrow S \times R[\%eax]$	unsigned full multiply
cltd	$R[\%edx] : R[\%eax] \leftarrow SignExtend(R[\%eax])$	Convert to quad word
idivl S	$R[\%edx] \leftarrow R[\%edx]:R[\%eax] \ mod \ S$; $R[\%eax] \leftarrow R[\%edx]:R[\%eax] \div S$;	Signed divide
divl S	$R[\%edx] \leftarrow R[\%edx]:R[\%eax] \ mod \ S$; $R[\%eax] \leftarrow R[\%edx]:R[\%eax] \div S$;	Unigned divide

5 状态寄存器相关

cmpl S2, S1 : $S1 - S2$。

testl S2, S1 : $S1\&S2$。

sete D : $D \leftarrow ZF$ (Equal or zero).

此外还有setne, sets, setns, setg, setge, setl, setle, seta, setae, setb, setbe.

6 PC相关

jmp Label :修改%eip (程序计数器)的值为Label指向的地址。
此外还有按条件跳转的je, jne, js, jns, jg, jge, jl, jle, ja, jae, jb, jbe.
如果Label不给出，而由其它值给出，需要加星号：
jmp *(%eax)

7 .开头

这些都是给链接器使用的标记，我们不必知道。如果要追根究底，看几遍生成的代码就清楚了。