C++反汇编初步3

肖浩宇

1 复习

1.1 标志位寄存器

32位标志寄存器

31	…	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
31	…	VF	RF		NT	IOPL		OF	DF	IF	TF	SF	ZF		AF		PF		CF

1.2 寻址

用[expression]来访问expression表达式所表示的地址中的数据
根据expression的不同分为直接寻址和简介寻址

2 分支与跳转

2.1 引子

程序要实现自动化，跳转是很重要的,，对吧！C语言中我们可以使用
goto（虽然现在不常用了）进行直接跳转
*if,switch*语句实现条件分支跳转
*while,for,do···while*实现循环
那么汇编肯定照样可以呀。我们一个一个来看。

2.2 jmp

jmp汇编指令很简单类似于C语言中的goto，就是从一个地址跳到另一个地址。

根据操作数不同，jmp可转移的范围也不同，生成的机器码的长度也会相应的不同。

但是我们无需关心这一点，我们在写汇编代码的时候，汇编器会帮助我们识别跳转的IP地址偏移量，从而确定操作数的长度

jmp只有一个操作数，它实现简单的跳转

```
jmp eax #可以理解为eip=eax
jmp Begin #跳转到Begin标志
Begin:
    ...
jmp [ecx] #可以理解为eip=*ecx
```

2.3 条件跳转

条件跳转需要借助我们上周讲的标志位寄存器,下面是一张条件跳转命令的表格

直接转移指令				
指令格式	机器码	测试标志	条件说明	符号
JO OPR	70	OF=1	结果有溢出	
JNO OPR	71	OF=0	结果无溢出	
JC OPR	72	CF=1	小于	<
JNC OPR	73	CF=0	大于或等于	>=
JZ/JE OPR	74	ZF=1	结果为0	==
JNZ/JNE OPR	75	ZF=0	结果非0	!=
JS OPR	78	SF=1	结果为负	<0
JNS OPR	79	SF=0	结果为正	>0
JP/JPE OPR	7A	PF=1	结果中1的个数为偶数	
JNP/JPO OPR	7B	PF=0	结果中1的个数为奇数	

间接转移指令-无符号数				
指令格式	机器码	测试标志	条件说明	符号
JB/JNAE OPR	72	CF=1	低于/不高于且不等于	<
JNB/JAE OPR	73	CF=0	不低于/高于或等于	>=
JBE/JNA OPR	76	(CF=1)|(ZF=1)	低于或等于/不高于	<=
JNBE/JA OPR	77	(CF=0)&(ZF=0)	不低于且不等于/高于	>

间接转移指令-有符号数				
指令格式	机器码	测试标志	条件说明	符号
JL/JNGE OPR	7C	(SF^OF)=1	小于/不大于且不等于	<
JNL/JGE OPR	7D	(SF^OF)=0	不小于/大于或等于	>=
JLE/JNG OPR	7E	(SF^OF)=1|(ZF=1)	小于或等于/不大于	<=
JNLE/JG OPR	7F	(SF^OF)=0&(ZF=0)	不小于且不等于/大于	>

条件跳转的指令虽然有很多，但是不用怕，记住他们其实不难我们可以这样理解

$N \longrightarrow Not$

$E \longrightarrow Equal$

$A \longrightarrow Above$

$B \longrightarrow Below$

$L \longrightarrow Less$

$G \longrightarrow Greater$

就是通过以上的这些条件跳转指令，c语言实现了if条件语句

比方说C语言中

```
if(a > 3) {...}
```

编译成汇编就会变成这样

```
cmp eax,3 #假设eax中存放a的值
jle BlockEnd
# Block
BlockEnd:
#...
```

简单吧!

其中cmp指令适用于比较的，它能够改变标志位.上面的cmp eax,3相当于(eax-3)但是它不改变eax的值，而只改变标志位

附：

能够直接影响标志寄存器的相关标志位的指令有:

1、算术运算指令: add、sub、adc、sbb、inc、dec、neg、mul、div、imul、idiv,等等;

2、按位逻辑运算: and、or、xor、not,等等;

3、比较运算指令: cmp、test;

4、移位操作指令: shr、shl、sar、sal、ror、rol、rcr、rcl;

5、BCD数调整指令: aaa、aas、daa、das、aam、aad;

6、标志处理指令: clc、stc、cmc、cld、std、cli、sti;

具体功能可以上网查或者差intel手册

2.4 switch语句

我们知道switch可以实现类似于if的功能，但是它们在底层的汇编代码上市不同的。比如说以下C语言代码

```
switch{a}   //假定a为无符号数
{
    case 1:
        func1();
    case 2:
        func2();
    default:
        func3();
}
```

它对应的代码大概是这样

```
cmp eax,1
jz L1
cmp eax,2
jz L2
jmp Default
L1:
call func1
L2:
call func2
Default:
call func3
```

实际上以上的汇编代码只会在以debug模式编译后出现。一般而言，编译器会对代码进行优化。

```
cmp eax,2
jg Default
dec eax
lea ecx,[L1+eax*4]
jmp ecx
L1:
call func1
L2:
call func2
Default:
call func3
```

3 循环

3.1 loop

用ecx来存储循环次数，每次循环后递减

```
Loop:
...
loop Loop
```

3.2 使用跳转实现循环

3.2.1 do···while

这个语句是汇编向C的一个很自然的过度

```
do
{
    Block
}while(Condition);
```

等价于汇编代码

```
Do: #循环开始
Block   #执行循环代码块
Condition   #执行语句设置标志位
jCondition Do   #根据标志位进行条件跳转
```

3.3 while

while比do···稍微复杂一些

```
while(Condition)
{
    Block
}
```

对应于汇编

```
Condition #执行Condition语句内容，设置标志位
jnCondition End #条件不成立则跳出
While:  #循环开始
Block   #循环体
Condition   #判断条件
jCondition While #条件成立继续执行循环语句
End:
```

3.3.1 for语句

```
for(A;B;C)
{
```

```
    Block
}
```

对应的汇编代码大致是这样的形式

```
Initial:
A   #初始化语句
B   执行判断语句
JNCondition Done     如果不成立则循环结束
For:
Block   #函数体
C   #执行C语句更新
B   #判断语句
JCondition For  如果条件成立继续循环
Done
```

3.4 用rep等指令实现内存中连续数据的转移

使用cpu提供的一些汇编指令能使我们快速搞笑地进行整块内存的移动。

比如rep movsb指令，不断将esi地址中内容复制到edi地址单元，并将ecx减1，直到ecx等于0。

这样的指令为我们操作整块的内存空间提供了便利