汇编作业解答

下面的Homework 5~7为柴老师班的作业编号，对应王老师的Homework 5（括号内为王老师班的题号）

Homework 5

T1（T1）

Operand	Value
%ebx	0x100
$0x150	0x150
0x170	0x17
(%ebx)	0x10
(%ebx,%eax)	0x11
0x30(%ebx)	0x13
80(%ebx,%eax,2)	0x17

Instruction	Destination	Value
addl %eax,%ebx	%ebx	0x110
subl %eax,(%ebx)	0x100	0x0
leal 0x50(%eax),%edx	%edx	0x60
movzbl %al,%ebx	%ebx	0x10
movsbl %bh,%ecx	%ecx	0x1

Instruction	OF	SF	ZF	CF
leal (%eax),%ebx	0	0	0	0
subl %ebx,%eax	0	1	0	1
xorl %eax,%eax	0	0	1	0
test %eax,%ebx	0	0	1	0

T2（T2）

```
int -0xc(%ebp) = 3;
int -0x8(%ebp) = 2;
int -0x4(%ebp) = 1;
int %eax, -0x10(%ebp);
while(-0xc(%ebp)<=0x5)
{
    -0x10(%ebp) = -0x4(%ebp);
    -0x4(%ebp) = -0x8(%ebp);
    -0x8(%ebp) += -0x10(%ebp);
    -0xc(%ebp) += 1;
}
return -0x8(%ebp);
```

（王老师班 T3）

一种可能的汇编代码实现

```
.global solve
solve:
mov (%rdi), %eax
cmp %eax, 4(%rdi)
jge .L1
mov 4(%rdi), %eax
.L1:
cmp %eax, 8(%rdi)
jge .L2
mov 8(%rdi), %eax
.L2:
cmp %eax, 12(%rdi)
jge .L3
mov 12(%rdi), %eax
.L3:
cmp %eax, 16(%rdi)
jge .L4
mov 16(%rdi), %eax
.L4:
cmp %eax, 20(%rdi)
jge .L5
mov 20(%rdi), %eax
.L5:
cmp %eax, 24(%rdi)
jge .L6
mov 24(%rdi), %eax
.L6:
cmp %eax, 28(%rdi)
jge .L7
mov 28(%rdi), %eax
.L7:
cmp %eax, 32(%rdi)
jge .L8
mov 32(%rdi), %eax
.L8:
```

```
cmp %eax, 36(%rdi)
jge .L9
mov 36(%rdi), %eax
.L9:
ret
```

一种可能的反汇编结果

```
0000000000001159 <solve>:
    1159:   8b 07                   mov    (%rdi),%eax
    115b:   39 47 04                cmp    %eax,0x4(%rdi)
    115e:   7d 03                   jge    1163 <solve+0xa>
    1160:   8b 47 04                mov    0x4(%rdi),%eax
    1163:   39 47 08                cmp    %eax,0x8(%rdi)
    1166:   7d 03                   jge    116b <solve+0x12>
    1168:   8b 47 08                mov    0x8(%rdi),%eax
    116b:   39 47 0c                cmp    %eax,0xc(%rdi)
    116e:   7d 03                   jge    1173 <solve+0x1a>
    1170:   8b 47 0c                mov    0xc(%rdi),%eax
    1173:   39 47 10                cmp    %eax,0x10(%rdi)
    1176:   7d 03                   jge    117b <solve+0x22>
    1178:   8b 47 10                mov    0x10(%rdi),%eax
    117b:   39 47 14                cmp    %eax,0x14(%rdi)
    117e:   7d 03                   jge    1183 <solve+0x2a>
    1180:   8b 47 14                mov    0x14(%rdi),%eax
    1183:   39 47 18                cmp    %eax,0x18(%rdi)
    1186:   7d 03                   jge    118b <solve+0x32>
    1188:   8b 47 18                mov    0x18(%rdi),%eax
    118b:   39 47 1c                cmp    %eax,0x1c(%rdi)
    118e:   7d 03                   jge    1193 <solve+0x3a>
    1190:   8b 47 1c                mov    0x1c(%rdi),%eax
    1193:   39 47 20                cmp    %eax,0x20(%rdi)
    1196:   7d 03                   jge    119b <solve+0x42>
    1198:   8b 47 20                mov    0x20(%rdi),%eax
    119b:   39 47 24                cmp    %eax,0x24(%rdi)
    119e:   7d 03                   jge    11a3 <solve+0x4a>
    11a0:   8b 47 24                mov    0x24(%rdi),%eax
    11a3:   c3                      ret
```

差别

- 反汇编代码的所有数字变成了十六进制表示
- jump指令跳转目的地变成了实际的地址，没有人工写的代码里的L1等入口
- ……（其它，合理即可）

Homework 6

T1（T4）

Name	Value
%eax	0x10000000

Name	Value
%ecx	22
$0x10000004	0x10000004
0x10000012	NONE
0xFFFFFF8	NONE
(%eax, %ecx, 8)	44

这里访问0x10000012为非对齐访问，可能会引发硬件异常或未定义行为，具体取决于处理器架构。具体可以参考https://blog.csdn.net/zhangxiaio1/article/details/142257078

T2（T5）

```
int dw_loop(int x, int y, int n) {
     do{
            x += n;
            y *= n;
            n --;
      }while(n > 0 && y < n);
      return x;
}
```

T3（T6）

(1)

一种可能的汇编代码实现

```
mov %rdi, %rax
imul %rsi, %rax
mov %rdi, %rdx
add %rsi, %rdx
imul %rsi, %rdx
cmp %rdi, %rsi
cmovge %rdx, %rax
```

(2)

原因：因为乘法的时间复杂度比较高，所以x*y和(x+y)*y在计算上耗时较久。如果用条件传送的话，则在两种情况下都要实现这两个复杂的运算，由此导致时间开销比直接使用条件跳转还大，所以GCC选择不用条件传送。

T4（T7）

一种可能的汇编代码实现

```
    .section    .rodata
    .align  8
.L:
```

```
    .quad   .L1
    .quad   .L0
    .quad   .L3
    .quad   .L2
    .quad   .L2
    .quad   .L4
    .quad   .L4
; 这里假设x存在寄存器%rdx中（也可以假设成别的）
mov %rdx, %rcx
mov $0, %rax
sub $24, %rcx
cmp $6, %rcx
ja .L0
jmp *.L(,%rcx,8)
.section    .rodata
.L1:
    mov %rdx, %rax
    add %rdx, %rax
    jmp .LE
.L2:
    mov %rdx, %rax
    add $10, %rax
    jmp .LE
.L3:
    mov %rdx, %rax
    sal %rax
.L4:
    add $5, %rax
    jmp .LE
.L0:
    mov $3, %rax
.LE:
; switch 结束（后面可能有别的代码）
```

Homework 7

T1 (T8)

```
题目1
一个C函数fun具有如下代码体：(参数从右向左入栈)
*p = d;
return x-c;
执行这个函数体的IA32代码如下：
Movsbl  12(%ebp), %edx  ;较小的byte->dword, s表示符号填充，z表示0填充
Movl    16(%ebp), %eax
Movl    %edx, (%eax)
Movswl  8(%ebp), %eax
Movl    20(%ebp), %edx
Subl    %eax, %edx
Movl    %edx, %eax
写出函数fun的原型，给出参数p, d, x, c的类型和顺序。写出求解过程。

```

答案：`fuc(short c, char d, int *p, int x)`

首先观察代码的整体结构，可以发现汇编中仅有一行对应指针赋值的代码`Movl edx (%eax)`和一行对应减法的代码`Subl %eax %edx`，也就可以从这一点反推出来`Movl edx (%eax)`中%edx存的是d，%eax中存的是p，再观察前两行的mov指令，可以发现%edx对应的栈内地址为12(%ebp)。指针赋值执行的是Movsbl，说明d大小为1byte，符号拓展，对应的是char类型。%eax对应的地址是16(%ebp)。指针赋值执行的是Movsbl，说明大小是双字 四字节，对应的是int类型，所以p是int *类型。

接下来分析`Subl %eax %edx`，可以确认%edx中存有x，%eax中存有c，且根据

`Movswl 8(%ebp), %eax`和`Movl 20(%ebp), %edx`可以确定c在栈中的地址是8(%ebp), x在栈中的地址是20(%ebp)，且c是单字大小且符号拓展，为short类型，x是双字大小，int类型。

由于传参入栈操作执行于函数调用前，因此入栈顺序为`x p d c`，再结合从右往左入栈，得到答案为`fuc(short c, char d, int *p, int x)`

T2（T9)

```
•题目2

•Suppose the initial value of %esp is 0x7FFFFFC4, initial value of %ebp is
0x7FFFFFF4.

•The value stored in address 0x7FFFFFC0 is 0x120, value stored in address
0x7FFFFFC4 is 0x200, the value stored in address 0x7FFFFFF4 is 0x2710.

•We have following x86 assembly code executed sequentially:

•pushl %esp (instruction 1)
 movl %esp,%ebp (instruction 2)
 popl %ebp (instruction 3)

•Question: After each instruction executed, what is the value of %esp and
%ebp
```

这道题在布置的时候题目写错了（不过没什么影响），最开始那一行应该是pushl %ebp，

如果按照**pushl %esp**来写的话答案是：

(1) Instruction 1: `%ebp = 0x7FFFFFF4  %esp = 0x7FFFFFC0`
(2) Instruction 2: `%ebp = 0x7FFFFFC0  %esp = 0x7FFFFFC0`
(3) Instruction 3: `%ebp = 0x7FFFFFC0  %esp = 0x7FFFFFC4`

注意这里push是先减esp，再入栈。

如果按照**pushl %ebp**来写的话答案是：

(1) Instruction 1: `%ebp = 0x7FFFFFF4  %esp = 0x7FFFFFC0`
(2) Instruction 2: `%ebp = 0x7FFFFFC0  %esp = 0x7FFFFFC0`
(3) Instruction 3: `%ebp = 0x7FFFFFF4  %esp = 0x7FFFFFC4`

T3（T10)

c代码如下

```c
int main(){
    int b,a;
    scanf("%d %d",&b,&a);
    int c = a^b;
    printf("%d %d %d\n",c,a,b);
    return 0;
}
```

第24行是printf调用，此时的状态为

```
寄存器状态：
%edi: .LC1的地址 (格式字符串"%d %d %d\n"的地址)
%esi: a ^ b的结果
%edx: a的值
%ecx: b的值
%eax: 0
%rsp: 0x8000408

栈状态：
0x8000420 |           | <- 初始%rsp
0x800041C |           |
0x8000418 |           |
0x8000414 | b的值      | <- 0x8000408 + 12
0x8000410 | a的值      | <- 0x8000408 + 8
0x800040C |           |
0x8000408 |           | <- 当前%rsp (0x8000408)
```