homework 5

处理输入输出的c函数如下：

```
#include<stdio.h>
int impl(int*);
int main()
{
    int a[10];
    for(int i=0;i<10;i++)
    {
        scanf("%d",&a[i]);
    }
    printf("%d",impl(a));
    return 0;
}
```

impl函数用一个循环来实现了找最小值的功能，除了存储传过来的参数地址，还有一个临时变量用来存目前的最大值(初始化为传入数组的第一个值)，一个临时变量来存循环次数。

分别存在了`-8(%rbp)`,`-4(%rbp)`两个位置。

用jump实现了一个for循环，用于寻找最小值。

由于传入的参数是一个int数组的首地址,每个int占4个字节,所以用`leaq (,%rax,4),%rdx`,`movq -16(%rbp),%rax`,`addq %rdx,%rax`来取对应位置的值

impl函数如下：

```
.global impl
impl:
pushq %rbp
movq %rsp,%rbp
movq %rdi,-16(%rbp)
movq -16(%rbp),%rax
movl (%rax),%eax
movl %eax,-8(%rbp)
movl $1,-4(%rbp)
jmp for1
l1:
    movl -4(%rbp),%eax
    leaq (,%rax,4),%rdx
    movq -16(%rbp),%rax
    addq %rdx,%rax
    movl (%rax),%eax
    cmpl %eax,-8(%rbp)
    jl add1
    movl -4(%rbp),%eax
    leaq (,%rax,4),%rdx
    movq -16(%rbp),%rax
    addq %rdx,%rax
    movl (%rax),%eax
    movl %eax,-8(%rbp)
add1:
```

```
    addl $1,-4(%rbp)
for1:
    cmpl $9,-4(%rbp)
    jle l1
    movl -8(%rbp),%eax
    popq %rbp
    ret
```

用ida反汇编得到的结果：

```
.text:0000000000001159 55                              push    rbp
.text:000000000000115A 48 89 E5                        mov     rbp, rsp
.text:000000000000115D 48 89 7D F0                     mov     [rbp+var_10], rdi
.text:0000000000001161 48 8B 45 F0                     mov     rax, [rbp+var_10]
.text:0000000000001165 8B 00                           mov     eax, [rax]
.text:0000000000001167 89 45 F8                        mov     [rbp+var_8], eax
.text:000000000000116A C7 45 FC 01 00 00 00            mov     [rbp+var_4], 1
.text:0000000000001171 EB 34                           jmp     short for1
.text:0000000000001171
.text:0000000000001173                                 ; -------------------------
-------------------------------------------------
.text:0000000000001173
.text:0000000000001173                                 l1:
            ; CODE XREF: impl+52↓j
.text:0000000000001173 8B 45 FC                        mov     eax, [rbp+var_4]
.text:0000000000001176 48 8D 14 85 00 00 00 00         lea     rdx, ds:0[rax*4]
.text:000000000000117E 48 8B 45 F0                     mov     rax, [rbp+var_10]
.text:0000000000001182 48 01 D0                        add     rax, rdx
.text:0000000000001185 8B 00                           mov     eax, [rax]
.text:0000000000001187 39 45 F8                        cmp     [rbp+var_8], eax
.text:000000000000118A 7D 17                           jl     short add1
.text:000000000000118A
.text:000000000000118C 8B 45 FC                        mov     eax, [rbp+var_4]
.text:000000000000118F 48 8D 14 85 00 00 00 00         lea     rdx, ds:0[rax*4]
.text:0000000000001197 48 8B 45 F0                     mov     rax, [rbp+var_10]
.text:000000000000119B 48 01 D0                        add     rax, rdx
.text:000000000000119E 8B 00                           mov     eax, [rax]
.text:00000000000011A0 89 45 F8                        mov     [rbp+var_8], eax
.text:00000000000011A0
.text:00000000000011A3
.text:00000000000011A3                                 add1:
            ; CODE XREF: impl+31↑j
.text:00000000000011A3 83 45 FC 01                     add     [rbp+var_4], 1
.text:00000000000011A3
.text:00000000000011A7
.text:00000000000011A7                                 for1:
            ; CODE XREF: impl+18↑j
.text:00000000000011A7 83 7D FC 09                     cmp     [rbp+var_4], 9
.text:00000000000011AB 7E C6                           jle     short l1
.text:00000000000011AB
.text:00000000000011AD 8B 45 F8                        mov     eax, [rbp+var_8]
.text:00000000000011B0 5D                              pop     rbp
.text:00000000000011B1 C3                              retn
```

对比：
除了语法格式用的AT&T和Intel的区别，别的基本上是一样的。

但是Intel的跳转标签前有个short，at&t下没有，还有就是寻址的方式也略有区别。