作业2

练习

3.1

操作数	值
%eax	0x100
0x104	0xAB
$0x108	0x108
(%eax)	0xFF
4(%eax)	0xAB
9(%eax,%edx)	0x11
260(%ecx,%edx)	0x13
0xFC(,%ecx,4)	0xFF
(%eax,%edx,4)	0x11

3.3

```
movb $0xF, (%bl) //%bl不能作地址寄存器
movl %ax, (%esp) //后缀应该是w而不是l
movw (%eax),4(%esp) //源操作数和目的操作数不能都是内存中的
movb %ah, %sh //没有寄存器%ah和寄存器%sh
movl %eax, $0x123  //立即数不能作为目的操作数
movl %eax, %dx // 后缀应该是w不是l
movb %si, 8(%ebp) //后缀应该是w而不是b
```

3.6

指令	结果
leal 6(%eax), %edx	x+6
leal (%eax,%ecx),%edx	x+y
leal (%eax,%ecx,4), %edx	x+4y
leal 7(%eax,%eax,8), %edx	9x+7
leal 0xA(,%eax,4), %edx	4x+10
leal 9(%eax,%ecx,2), %edx	x+2y+9

3.7

指令	目的	值
addl %ecx, (%eax)	0x100	0x100
subl %edx, 4(%eax)	0x104	0xA8
imull $16, (%eax,%edx,4)	0x10C	0x110
incl 8(%eax)	0x108	0x14
decl %ecx	%ecx	0x0
subl %edx, %eax	%eax	0xFD

3.9

```
int arith(int x,int y,int z){
    int t1=x^y;
    int t2=t1>>3;
    int t3=~t2;
    int t4=t3-z;
    return t4;
}
```

3.12

A.从汇编代码中可以看出，程序在内存中将乘积的低32位和高32位分开存储，所以是64位乘积，又根据第四行的无符号乘法，所以num_t的数据类型是unsigned long long

B.y=2^(32)*yh+yl,yh表示y的高32位，yl表示y的低32位。所以x*y=x*2^(32)*yh+x*yl,最终需要取结果的低64位。下面对代码进行注释说明：

```
movl 12(%ebp), %eax   //x
movl 20(%ebp), %ecx   //yh
imull %eax, %ecx      //m=x*yh(用x乘y的高32位，结果取低32位)
mull 16(%ebp)         //n=x*yl
leal (%ecx,%edx), %edx//m+nh(用m加上x乘y的低32位乘积的高32位作为结果的高32位)
movl 8(%ebp), %ecx //dest
movl %eax, (%ecx)//存储最终结果的低32位
movl %edx, 4(%ecx)//存储最终结果的高32位
```

3.13

A.后缀l表示32位，比较是补码的<,所以data_t是int

B.后缀w表示16位，比较是补码的>=,所以data_t是short

C.后缀b表示8位，比较是无符号数的<,所以data_t是unsigned char

D.后缀表示32位，比较是!=,不能看出是补码还是无符号甚至是指针。所以data_t可能是int,unsigned 或者long unsigned long,或者是指针。

3.15

A. 0x8048291+0x05=0x8048296

B. 0x8048359+0xe7=0x8048359-0x19=0x8048340

C. 0x8048391-0x12=0x804837f

D. 0xfffffffe0+0x80482c4=0x80482c4-0x20=0x80482a4

E. ff 25表示指令间接跳转，后面的地址因为用小端法表示，所以08049ffc表示为fc 9f 04 08

3.16

A.对汇编代码写注释：

```
movl    8(%ebp), %edx    get a
movl    12(%ebp), %eax   get p
testl   %eax,%eax
je      .L3              if p==0 goto .L3
testl   %edx, %edx       if a<=0 goto .L3
jle     .L3
addl    %edx, (%eax)     *p +=a
.L3:
```

C语言goto版本：

```
void goto_cond(int a,int *p){
        if(p==0)
            goto done;
        if(a<=0)
            goto done;
        *p +=a;
    done:
        return;
}
```

B. 汇编代码的两个条件分支其实都是对(p && a>0)的测试，如果p为0，则跳过对后面a>0的测试，这在汇编代码中就表现为如果第一个分支就跳转，则跳过第二个分支。

3.18

还原后的C代码如下：

```
int test(int x,int y){
    int val=x ^ y;
    if(x<-3){
        if(y>x)
            val = x*y;
        else
            val = x+y;
    }else if(x>2)
        val = x-y;
    return val;
}
```

3.21

A. 从汇编代码中可以看出寄存器%edx被初始化为a+b,每循环一次都+1，同时寄存器%ecx中的a的值每循环一次也+1.这样，每次只需比较寄存器%ecx和%ebx中的值，而寄存器%edx中的值永远是a+b.

B.创建寄存器使用表：

寄存器	变量	初值
%ecx	a	a(每次+1)
%ebx	b	b
%edx	a+b	a+b(每次+1)
%eax	result	1

C. 给代码添加注释：

```
    movl    8(%ebp), %ecx          get a
    movl    12(%ebp), %ebx         get b
    movl    $1, %eax               set result = 1
    cmpl    %ebx, %ecx            compare a:b
    jge     .L11                  if a>=b, goto done
    leal    (%ebx,%ecx), %edx      calculate a+b
    movl    $1, %eax              set result = 1
.L12:                             loop:
```

```
    imull   %edx, %eax              result*=(a+b)
    addl    $1, %ecx                a++
    addl    $1, %edx                (a+b)++
    cmpl    %ecx, %ebx              compare b:a
    jg      .L12                   if(b>a) goto loop
.L11:                             done:
```

D. c语言代码：

```
int loop_while_goto(int a,int b){
    int result = 1;
    if(a>=b)
        goto done;
    int a_plus_b = a+b;
 loop:
    result *= a_plus_b;
    a++;
    a_plus_b++;
    if(b>a)
        goto loop;
 done:
    return result;
}
```

3.22

A.填写C代码中缺失的部分：

```
int fun_a(unsigned x){
    int val = 0;
    while(x){
        val = val ^ x;
        x>>=1;
    }
    return val&0x1;
}
```

B. 这个函数是用来计算x的奇偶性的，如果x中有奇数个1，返回1，如果有偶数个1就返回0

3.23

A. 填写C代码中缺失的部分

```
int fun_b(unsigned x){
    int val = 0;
    int i;
    for(i=0;i<32;i++){
        val = (val<<1)|(x&0x1);
        x>>=1;
    }
    return val;
}
```

B. 这个函数的作用是把x的二进制表示的位反向，x的位从左往右的顺序等于val的位从右往左的顺序。

3.27

填补C代码中缺失的表达式：

```
int test(int x,int y){
    int val = 4*x;
    if(y>0){
        if(x<y)
            val = x-y;
        else
            val = x^y;
    }else if(y<-2)
        val = x+y;
    return val;
}
```

3.28

A. 根据汇编代码第二行将x+2，所以最小的情况标号是-2.又因为汇编代码第三行是和6比较，所以最大的情况标号为6-2=4.又发现在汇编代码中，如果情况>6,将会跳转到.L2, 又发现跳转表中第二行也跳转到.L2, 它对应的情况标号是-1，所以可以判断-1缺失。综上所述，switch语句体内情况标号的值为-2，0，1，2，3，4

B. 观察到汇编代码第六第七行都跳转到情况.L6，说明情况重复，其对应的情况标号是2和3，综上所述，C代码中.L6情况有多个标号2和3

作业

3.54

等价于汇编代码的C代码：

```
int decode2(int x,int y,int z){
    int a=z-y;
    int b=(a<<15)>>15;
    int c=x^a;
    int d=b*c;
    return d;
}
```

3.55

对汇编代码添加注释：(用l表示低32位，h表示高32位)

```
movl    12(%ebp), %esi get xl->esi
movl    20(%ebp), %eax  get y(int只有32位)
movl    %eax, %edx      y->edx(yl->edx)
sarl    $31, %edx       把y算术右移31位，即用符号填充，用以充当y的高32位，即yh
movl    %edx, %ecx      yh->ecx
imull   %esi, %ecx      a=xl*yh->ecx
movl    16(%esp), %ebx   get xh->ebx
imull   %eax, %ebx      b=yl*xh->ebx
addl    %ebx, %ecx       a+b
mull    %esi             c=xl*yl 低32位在寄存器%eax中 高32位在寄存器%edx中
```

```
leal    (%ecx,%edx), %edx  a+b+ch
movl    8(%ebp), %ecx      dest
movl    %eax, (%ecx)       储存结果低32位
movl    %edx, 4(%ecx)      储存结果高32位
```

说明：x=2^(32)*xh+xl,y=2^(32)*yh+yl,所以xy=2^(64)*x*y+2^(32)*(xl*yh+xh*yl)+xl*yl,因为结果只有64位，所以忽略第一项，而最终结果的低32位是xl*yl的低32位，最终结果的高32位是xl*yh的低32位+xh*yl的低32位+xl*yl的高32位。

3.56

A.

值	x	n	result	mask
寄存器	%esi(第一行)	%ebx(第二行)	%edi(最后一步从%edi到%eax)	%edx

B. result的初始值是1431655765,mask的初始值是-2147483648

C. mask的测试条件是mask非0

D. mask = mask>>n(第十行)

E. result = result^(x&mask)

F. 填写代码中缺失部分：

```
int loop(int x,int n){
    int result = 1431655765;
    int mask;
    for(mask=-2147483648;mask!=0;mask=mask>>n){
        result = result^(x&mask);
    }
    return result;
}
```

3.58

填写C代码中缺失的部分：

```
typedef enum{MODE_A,MODE_B,MODE_C,MODE_D,MODE_E}mode_t;
int switch3(int *p1,int *p2,mode_t action){
    int result = 0;
    switch(action){
        case MODE_A:
            result = *p1;
            *p1 = *p2;
            break;
        case MODE_B:
            *p2 = *p2+*p1;
            result = *p2;
            break;
        case MODE_C:
            *p2 = 15;
            result = *p1;
            break;
        case MODE_D:
            *p2 = *p1;
```

```
        case MODE_E:
            result = 17;
            break;
        default:
            result = -1;    //(初始化为-1)
    }
    return result;
}
```

3.59

用C代码填写开关语句的主体：

```
int switch_prob(int x,int n){
    int result = x;
    switch(n){
        case 40:
        case 42:
            result<<=3;
            break;
        case 43:
            result>>=3;
            break;
        case 44:
            result *= 7;
        case 45:
            result *= result;
        default:
            result += 17;
    }
    return result;
}
```