1. 假设变量 *x* 和 *y* 分别存放在寄存器 EAX 和 ECX 中，请给出以下每条指令执行后寄存器 EDX 中的结果。

（1）leal (%eax), %edx
（2）leal 4(%eax, %ecx), %edx
（3）leal (%eax, %ecx, 8), %edx
（4）leal 0xC(%ecx, %eax, 2), %edx
（5）leal (, %eax, 4), %edx
（6）leal (%eax, %ecx), %edx

答：

（1）R[edx]=x
（2）R[edx]=x+y+4
（3）R[edx]=x+8*y
（4）R[edx]=y+2*x+12
（5）R[edx]=4*y
（6）R[edx]=x+y

2. 假设函数 operate 的部分 C 代码如下：

```
1    int  operate(int x, int y, int z, int k)
2    {
3        int  v = _____ ;
4        return v;
5    }
```

以下汇编代码用来实现第 3 行语句的功能，请根据以下汇编代码，填写 operate 函数缺失的部分。

```
1    movl    12(%ebp), %ecx
2    sall    $8, %ecx
3    movl    8(%ebp), %eax
4    movl    20(%ebp), %edx
5    imull   %edx, %eax
6    movl    16(%ebp), %edx
7    andl    $65520, %edx
8    addl    %ecx, %edx
9    subl    %edx, %eax
```

答：

```
movl  12(%ebp), %ecx    //R[ecx]←M[R[ebp]+12]，将 y 送 ECX
sall  $8, %ecx          //R[ecx]←R[ecx]<<8，将 y*256 送 ECX
movl  8(%ebp), %eax     //R[eax]←M[R[ebp]+8]，将 x 送 EAX
movl  20(%ebp), %edx    //R[edx]←M[R[ebp]+20]，将 k 送 EDX
imull %edx, %eax        //R[eax]←R[eax]*R[edx]，将 x*k 送 EAX
movl  16(%ebp), %edx    //R[edx]←M[R[ebp]+16]，将 z 送 EDX
andl  $65520, %edx      //R[edx]←R[edx] and 65520，将 z&0xFFF0 送 EDX
addl  %ecx, %edx        //R[edx]←R[edx] + R[ecx]，将 z&0xFFF0+y*256 送 EDX
subl  %edx, %eax        //R[eax]←R[eax]-R[edx]，将 x*k-(z&0xFFF0+y*256)送 EAX
```

根据以上分析可知，第 3 行缺失部分为：

3　　int　v = x*k-(z&0xFFF0+y*256) ;

3. 已知函数 func 的 C 语言代码框架及其过程体对应的汇编代码如下图所示，根据对应的汇编代码填写 C 代码中缺失的表达式。

```
1    int func(int x, int y)
2    {
3        int z = _____ ;
4        if ( _____ ) {
5            if ( _____ )
6                z = _____ ;
7            else
8                z = _____ ;
9        } else if ( _____ )
10           z = _____ ;
11       return z;
12   }
```

```
1    movl    8(%ebp), %eax
2    movl    12(%ebp), %edx
3    cmpl    $-100, %eax
4    jg      .L1
5    cmpl    %eax, %edx
6    jle     .L2
7    addl    %edx, %eax
8    jmp     .L3
9  .L2:
10   subl    %edx, %eax
11   jmp     .L3
12 .L1:
13   cmpl    $16, %eax
14   jl      .L4
15   andl    %edx, %eax
16   jmp     .L3
17 .L4:
18   imull   %edx, %eax
19 .L3:
```

答：

```
1    int func(int x, int y)
2    {
3        int z = x*y ;
4        if ( x<=-100 ) {
5            if ( y>x )
6                z = x+y ;
7        else
8                z = x-y ;
9        } else if ( x>=16 )
10           z = x &y ;
11       return z;
12   }
```

4. 已知函数 funct 的 C 语言代码如下：

```
1    int funct(viod) {
2        int x, y;
3        scanf("%x %x", &x, &y);
4        return x-y;
5    }
```

函数 funct 对应的汇编代码如下：

```
1  funct：
2    push1    %ebp
3    movl     %esp, %ebp
4    subl     $40, %esp
5    leal     -8(%ebp), %eax
```

```
6   movl    %eax, 8(%esp)
7   leal    -4(%ebp), %eax
8   movl    %eax, 4(%esp)
9   movl    $.LC0, (%esp)          //将指向字符串“%x %x”的指针入栈
10  call    scanf                  //假定 scanf 执行后 x=15,y=20
11  movl    -4(%ebp), %eax
12  subl    -8(%ebp), %eax
13  leave
14  ret
```

假设函数 funct 开始执行时，R[esp]=0xbc000020，R[ebp]=0xbc000030，执行第 10 行 call 指令后，scanf 从标准输入读入的值为 0x16 和 0x100，指向字符串“%x %x”的指针为 0x804c000。回答下列问题或完成下列任务。

（1）执行第 3、10 和 13 行的指令后，寄存器 EBP 中的内容分别是什么？

（2）执行第 3、10 和 13 行的指令后，寄存器 ESP 中的内容分别是什么？

（3）局部变量 *x* 和 *y* 所在存储单元的地址分别是什么？

（4）画出执行第 10 行指令后 funct 的栈帧，指出栈帧中的内容及其地址。

答：

每次执行 pushl 指令后，R[esp]=R[esp]-4，因此，第 2 行指令执行后 R[esp]=0xbc00001c。

（1）执行第 3 行指令后，R[ebp]=R[esp]=0xbc00001c。到第 12 条指令执行结束都没有改变 EBP 的内容，因而执行第 10 行指令后，EBP 的内容还是为 0xbc00001c。执行第 13 行指令后，EBP 的内容恢复为进入函数 funct 时的值 0xbc000030。

（2）执行第 3 行指令后，R[esp]=0xbc00001c。执行第 4 行指令后 R[esp]=R[esp]-40=0xbc00001c-0x28=0xbbfffff4。因而执行第 10 行指令后，未跳转到 scanf 函数执行时，ESP 中的内容为 0xbbfffff4-4=0xbbfffff0；在从 scanf 函数返回后 ESP 中的内容为 0xbbfffff4。执行第 13 行指令后，ESP 的内容恢复为进入函数 funct 时的旧值，即 R[esp]=0xbc000020。

（3）第 5、6 两行指令将 scanf 的第三个参数&y 入栈，入栈的内容为 R[ebp]-8=0xbc000014；第 7、8 两行指令将 scanf 的第二个参数&x 入栈，入栈的内容为 R[ebp]-4=0xbc000018。故 x 和 y 所在的地址分别为 0xbc000018 和 0xbc000014。

（4）执行第 10 行指令后，funct 栈帧的地址范围及其内容如图所示。