3.

标号	后缀	源操作数	目的操作数
(1)	w	基址+比例变址+位移	寄存器
(2)	b	寄存器	基址+偏移
(3)	l	比例变址	寄存器
(4)	b	基址	寄存器
(5)	l	立即数	栈
(6)	l	立即数	寄存器
(7)	w	寄存器	寄存器
(8)	l	基址+变址+偏移	寄存器

5.

SRC_TYPE	**DST_TYPE**	机器级表示
char	int	movsbl %al,(%edx)
int	char	movb %al,(%edx)
int	unsigned	movl %edx,(%edx)
short	int	movswl %ax,(%edx)
unsigned char	unsigned	movzbl %al,(%edx)
char	unsigned	movsbl %al,(%edx)
int	int	movl %eax,(%edx)

6.

(1) R[ebp]+8,R[ebp]+12,R[ebp]+16

(2) void func(int *xptr, int *yptr, int *zptr){

```
  int tempx = *xptr;

  int tempy = *yptr;

  int tempz = *zptr;
```

```
    *yptr = tempx;

    *zptr = tempy;

    *xptr = tempz;

}
```

8.

(1) 寄存器EDX中内容改变, 改变后为 0x00000070, OF = 0, ZF = 0, SF = 0, CF = 1;

(2) 寄存器ECX中内容改变, 改变后为 0x80000008, OF = 1, ZF = 0, SF = 1, CF = 1;

(3) 寄存器BX中内容改变, 改变后为 0xFF00, OF = 0, ZF = 0, SF = 1, CF = 0;

(4) 不改变通用寄存器的内容, OF = 0, ZF = 0, SF = 1, CF = 0;

(5) 将存储单元0x8049380中内容×32 并将低 32 位放在 ECX 中, ECX 寄存器中内容改为 0x11e25500, OF = 1, CF = 1;

(6) 寄存器AX和BX内容相乘, 高 16 位存入 DX 中, 低 16 位存入 AX 中, DX 寄存器内容改为 0x0093, AX 寄存器内容改变为 0x0000, OF = 1, CF = 1;

(7) 寄存器CX的内容改变, 改变后为 0x000F, OF = 0, ZF = 0, SF = 0;

11.

(1) 目标地址: 0x804838C + 2 + 0x8 = 0x8048396

偏移量为 0000001EH, call指令机器代码共占5个字节, 故0x804838E + 5 + 0x1E = 0x80483B1;

(2) 目标地址: 0x8048390 + 2 + 0xF6 = 0x8048488

movl指令机器代码有10个字节, 前两个字节是操作码等, 后面八个字节为两个立即数, 第一个立即数是 0804A800H, 即汇编指令中目的地址 0x0804A800, 最后4个字节为 00000001H, 即常数 0x1;

(3) 0x80492E0 = x + 0x16, 可得x = 0x80492CA;

(4) 偏移量为 FFFFFF00H, jmp 指令的转移目标为 0x804829B + 0xFFFFFF00 = 0x804819B;

14.

(1)

```
movw 8(%ebp), %bx    // R[bx] ← M[R[ebp] + 8], 将 x 送 BX
movw 12(%ebp), %si   // R[si] ← M[R[ebp] + 12], 将 y 送 SI
movw 16(%ebp), %cs  // R[cx] ← M[R[ebp] + 16], 将 k 送 CX
.L1:
movw %si, %dx          // R[dx] ← R[si], 将 y 送 DX
movw %dx, %ax         // R[ax] ← R[dx], 将 y 送 AX
sarw $15, %dx          // R[dx] ← R[dx]>>15, 将 y 的符号扩展 16 位送 DX
idiv %cx                  // R[dx] ← R[dx - ax] ÷ R[cx] 的余数, 将 y % k 送 DX
                          // R[dx] ← R[dx - ax] ÷ R[cx] 的商, 将 y / k 送 AX
imulw %dx, %bx        // R[bx] ← R[bx] × R[dx], 将x × (y % k)送 BX
decw %cx                 // R[cx] ← R[cx] - 1, 将 k - 1 送 CX
testw %cx, %cx         // R[cx] and R[cx], 得 OF = CF =0, 负数则 SF = 1, 零则 ZF = 1
jle .L2                    // 若k小于等于 0, 则转 .L2
cmpw %cx, %si          // R[si] - R[cx], 将 y 与 k 相减得到各标志
jg .L1                     // 若 y 大于 k, 则转 .L1
.L2:
movswl %bx, %eax     // R[eax] ← R[bx], 将 x ×( y % k ) 送 AX
```

(2) 被调用者保存寄存器有 BX 、SI , 调用者保存寄存器有 AX 、 CX 和 DX , 在该函数过程体前面的准备阶段, 被调用者保存的寄存器 EBX 和 ESI 必须保存在栈中;

(3) 因为执行第 8 行除法指令前必须先将被除数扩展为 32 位, 而这里是带符号数除法, 因此, 采用算术右移以扩展 16 位符号, 放在高 16 位的 DX 中, 低 16 位在 AX 中;

17.

unsigned int test (char a, unsigned short b, unsigned short c, short *p);

18.

(1) 第 3 行: 0xbc00001C, 第 10 行: 0xBC00001C, 第 13 行: 0xBC000030

(2) 第 3 行: 0xbc00001C, 第 10 行: 0xBBFFFFF0, 第 13 行: 0xBC000020

(3) x: 0xBC000018 , y: 0xBC000014

(4)

0XBC00001C	**0XBC000030 ← EBP栈帧底部**
0xBC000018	x = 15
0xBC000014	y = 20
0xBC000010	
0xBC00000C	
0xBC000008	
0xBC000004	
0xBC000000	
0xBBFFFFFC	0xBC000014
0xBBFFFFF8	0xBC000018
0xBBFFFFF4	0x804C000
0xBBFFFFF0	从scanf返回的地址 ← ESP

19.

(1)x == 0　(2) 0　(3) x >> 1　(4) (x & 0x1) + rv

功能为计算x中1的个数总数

21.

表达式	类型	值	汇编代码
S	short *	A_S	leal (%edx), %eax
S+i	short *	A_S+2*i	leal (%edx, %ecx, 2), %eax
S[i]	short	M[A_S+2*i]	movw (%edx, %ecx, 2), %ax
&S[10]	short *	A_S+20	leal 20(%edx), %eax
&S[i+2]	short *	A_S+2*i+4	leal 4(%edx, %eax, 2), %eax
&S[i]-S	int	i	movl %ecx, %eax
S[4*i+4]	short	M[A_S+2(4*i+4)]	movw 8(%edx, %ecx, 8), %ax
*(S+i-2)	short	M[A_S+2*(i-2)]	movw -4(%edx, %ecx, 2), %ax

22.

M = 5, N = 7

23.

M = 9, N = 7, L = 18

28.

c: 0, d: 8, i: 16, s: 20, p: 24, l: 28, g: 32, v: 40

因为d和g按照8字节边界对齐，所以有 44 + 4 = 48 字节

可调整为如下顺序

```
struct{
   double d;
   long long g;
   int i;
   char *p;
   long l;
   void *v;
   short s;
   char c;
}test;
```

得到大小为 8 + 8 + 4 + 4 + 4 + 4 + 2 + 6 = 40 字节

31.

(1)

```
movl 8(%ebp), %edx            // R[edx] ← M[R[ebp] + 8], 将 x 送 EDX
movl 12(%ebp), %ecx           // R[ecx] ← M[R[ebp] + 12], 将 x 送 ECX
movl $255, %esi               // R[esi] ← 255, 将 255 送 ESI
movl $-2147483648, %edi   // R[edi] ← -2147483648, 将 0x80000000 送 EDI
.L3:
movl %edi, %eax               // R[eax] ← R[edi],
andl %edx, %eax               // R[eax] ← R[eax] and R[eax], 将 i and x 送 EAX
```

xor %eax, %esi // R[esi] ← R[esi] xor R[eax], 将 val xor (i and x)送ESI

movl %ecx, %ebx // R[ebx] ← R[ecx], 将 k 送 EBX

shr %bl, %edi // R[edi] ← R[edi] >> R[bl], 将 i 逻辑右移 k 位送 EDI

testl %edi, %edi

jne .L3 // 若 R[edi] ≠ 0, 则转 .L3

movl %esi, %eax // R[eax] ← R[esi]

(2) x 和 k 分别存放在 EDX 和 ECX 中, 局部变量 val 和 i 分别存放在 ESI 和 EDI 中;

(3) 局部变量 val 和 i 的初始值分别是 255 和 -2147483648

(4) 循环终止条件为 i = 0, 循环控制变量 i 每次循环被逻辑右移 k 位;

(5) ① int val = 255;

② for (i = -2147483648; i != 0; i = (unsigned) i >> k)

③ val ^= (i & x);