3.

| 后缀 | 源操作数 | 目的操作数 |
|---|---|---|
| w | 基址，比例变址，偏移 | 寄存器 |
| b | 寄存器 | 基址，偏移 |
| l | 比例变址 | 寄存器 |
| b | 基址 | 寄存器 |
| l | 立即数 | 栈 |
| l | 立即数 | 寄存器 |
| w | 寄存器 | 寄存器 |
| l | 基址，编址，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 %eax, (%edx) |
| short | int | movswl %ax, (%edx) |
| unsigned char | unsigned | movzbl %al, (%edx) |
| char | unsigned | movsbl %al, (%edx) |
| int | int | movl %eax, (%edx) |

6.

（1）分别是 R[ebp]+8,R[ebp]+12,R[ebp]+16

（2）代码如下

```
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 中的值会改变，改变后的值为 0x00000070，OF=0，ZF=0，SF=0，CF=1

（2）寄存器 ecx 中的值会改变，改变后的值为 0x80000008，OF=1，ZF=0，SF=1，CF=1

（3）寄存器 bx 中的值会改变，改变后的值为 0xFF00，OF=0，ZF=0，SF=1，CF=0

（4）不改变任何通用寄存器，OF=0，ZF=0，SF=1，CF=0

（5）单元 0x8049380 的值会改变，改变后的值为 0x11e25500，不影响标志位

（6）寄存器 cx 中的值会改变，改变后的值为 0x000F，OF=1，ZF=0，SF=0

10.

```
movl    12(%ebp), %eax    //R[eax]←M[R[ebp]+12]，将 x 送 EAX
movl    20(%ebp), %ecx    //R[ecx]←M[R[ebp]+20]，将 yh 送 ECX
imull   %eax, %ecx        //R[ecx]←R[ecx]*R[eax]，将 yh*x 的低 32 位送 ECX
mull    16(%ebp)          //R[edx]R[eax]←M[R[ebp]+16]*R[eax]，将 yl*x 送 EDX-EAX
leal    (%ecx, %edx), %edx
        // R[edx]←R[ecx]+R[edx]，将 yl*x 的高 32 位与 yh*x 的低 32 位相加后送 EDX
movl    8(%ebp), %ecx     //R[ecx]←M[R[ebp]+8]，将 d 送 ECX
movl    %eax, (%ecx)      //M[R[ecx]]←R[eax]，将 x*y 低 32 位送 d 指向的低 32 位
movl    %edx, 4(%ecx)     //M[R[ecx]+4]←R[edx]，将 x*y 高 32 位送 d 指向的高 32 位
```

num_type 的数据类型为 unsigned long long

11.

（1）因为 je 指令的操作码为 01110100，所以机器代码 7408H 中的 08H 是偏移量，故转移目标地址为：0x804838c+2+0x8=0x8048396。

call 指令中的转移目标地址 0x80483b1=0x804838e+5+0x1e，由此，可以看出，call 指令机器代码中后面的 4 个字节是偏移量，因 IA-32 采用小端方式，故偏移量为 0000001EH。call 指令机器代码共占 5 个字节，因此，下条指令的地址为当前指令地址 0x804838e 加 5。

（2）jb 指令中 F6H 是偏移量，故其转移目标地址为：0x8048390+2+0xf6=0x8048488。

movl 指令的机器代码有 10 个字节，前两个字节是操作码等，后面 8 个字节为两个立即数，因为是小端方式，所以，第一个立即数为 0804A800H，即汇编指令中的目的地址 0x804a800，最后 4 个字节为立即数 00000001H，即汇编指令中的常数 0x1。

（3）jle 指令中的 7EH 为操作码，16H 为偏移量，其汇编形式中的 0x80492e0 是转移目的地址，因此，假定后面的 mov 指令的地址为 $x$，则 $x$ 满足以下公式：0x80492e0=$x$+0x16，故 $x$=0x80492e0−0x16=0x80492ca。

（4）jmp 指令中的 E9H 为操作码，后面 4 个字节为偏移量，因为是小端方式，故偏移量为 FFFFFF00H，即 −100H=−256。后面的 sub 指令的地址为 0x804829b，故 jmp 指令的转移目标地址为 0x804829b+0xffffff00=0x804829b−0x100=0x804819b。

14.

(1)

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

（2）被调用者保存寄存器：BX,SI　调用者保存寄存器：AX,CX,DX

（3）因为执行除法指令前必须将操作数拓展到 32 位，所以要采用算数右移拓展 16 位富豪，放在高 16 位的 DX 中，低十六位放在 AX 中

17.

test 函数的原型为

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

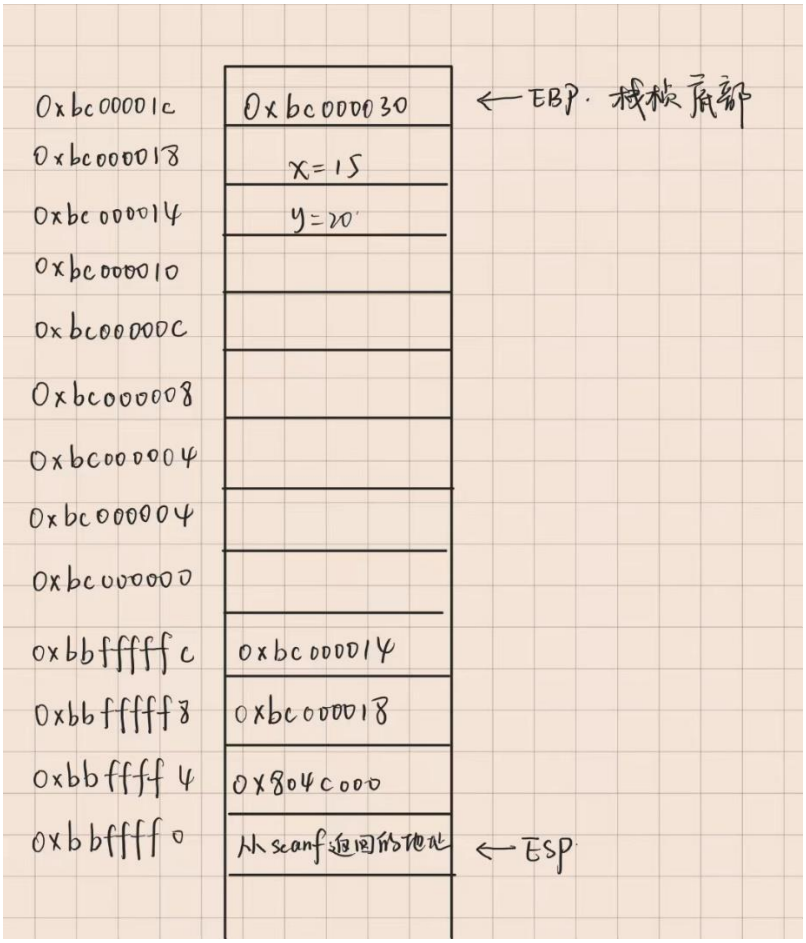
18.

（1）第 2 行指令执行后 R[esp]=0xbc00001c，执行第三条指令之后，R[esp]=R[esp]-4，直到第十二条指令执行结束后都没有改变 EBP 的内容，所以执行第十条指令后，EBP 的内容保持不变，仍为 0xbc00001，但在执行第 13 条指令后，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。所以执行第 4 行指令后 R[esp]=R[esp]-40=0xbbffffff4，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 0xbbffffff4-4=0xbbffffff0，但在从 scanf 函数返回后，ESP 中的内容为 0xbbffffff4 执行第 13 行指令后，ESP 的内容恢复为进入函数 funct 时的旧值，R[esp]=0xbc000020。

（3）第 5，6 两个指令将 scanf 的第三个参数&y 入栈后，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，人栈的内容为 R[ebp]-4=0xbc000018。所以 x 和 y 所在的地址分别为 0xbc000018 和 0xbc000014

（4）



19.

```
1  int refunc(unsigned x) {
2          if ( x==0 )
3              return 0 ;
4          unsigned nx = x>>1 ;
5          int rv = refunc(nx) ;
6          return (x&0x1)+rv ;
7  }
```

功能：对第x的各个位中1的个数.

21.

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

22.

M=5 N=7

23.

L=18　M=9　N=7

28.

偏移量:

| c | d | i | s | p | l | g | V |
|---|---|---|---|---|---|---|---|
| 0 | 8 | 16 | 20 | 24 | 28 | 32 | 40 |

总大小：48 字节

可以调整为:

```
struct{
    double d;
    long long g;
    int i;
    char *p;
```

```
    long l;
    void *v;
    short s;
    char c;
}
```
总大小为 40 字节