

补充几道题，课下随便做做：

- 第一章：1.4
- 第二章：2-18，2-34（奇数题）、2-35（偶数题）
- 第三章：3-2

第三章习题参考答案

3.   （1）后缀：w，    源：基址+比例变址+偏移，        目：寄存器  
      （2）后缀：b，    源：寄存器，                    目：基址+偏移  
      （3）后缀：l，    源：比例变址，                  目：寄存器  
      （4）后缀：b，    源：基址，                      目：寄存器  
      （5）后缀：l，    源：立即数，                    目：栈  
      （6）后缀：l，    源：立即数，                    目：寄存器  
      （7）后缀：w，    源：寄存器，                    目：寄存器  
      （8）后缀：l，    源：基址+变址+偏移，          目：寄存器

5.

表 3.12 题 5 用表

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）xptr、yptr 和 zptr 对应实参所存放的存储单元地址分别为：R[ebp]+8、R[ebp]+12、R[ebp]+16。

      （2）函数 func 的 C 语言代码如下：

```
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）指令功能为：R[edx]←R[edx]+M[R[eax]]=0x00000080+M[0x8049300]，寄存器 EDX 中内容改变。改变后的内容为以下运算的结果：00000080H+FFFFFFFF0H

$$\begin{array}{r}
0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1000\ 0000 \\
+ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 0000 \\
\hline
1\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111\ 0000
\end{array}$$

因此，EDX 中的内容改变为 0x00000070。根据表 3.5 可知，加法指令会影响 OF、SF、ZF 和 CF 标志。OF=0，ZF=0，SF=0，CF=1。

- (2) 指令功能为：R[ecx]←R[ecx]−M[R[eax]+R[ebx]]=0x00000010+M[0x8049400]，寄存器 ECX 中内容改变。改变后的内容为以下运算的结果：00000010H−80000008H

$$\begin{array}{r}
0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001\ 0000 \\
+ 0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1000 \\
\hline
0\ 1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1000
\end{array}$$

因此，ECX 中的内容改为 0x80000008。根据表 3.5 可知，减法指令会影响 OF、SF、ZF 和 CF 标志。OF=1，ZF=0，SF=1，CF=1⊕0=1。

- (3) 指令功能为：R[bx]←R[bx] or M[R[eax]+R[ecx]\*8+4]，寄存器 BX 中内容改变。改变后的内容为以下运算的结果：0x0100 or M[0x8049384]=0100H or FF00H

$$\begin{array}{r}
0000\ 0001\ 0000\ 0000 \\
\text{or } 1111\ 1111\ 0000\ 0000 \\
\hline
1111\ 1111\ 0000\ 0000
\end{array}$$

因此，BX 中的内容改为 0xFF00。由 3.3.3 节可知，OR 指令执行后 OF=CF=0；因为结果不为 0，故 ZF=0；因为最高位为 1，故 SF=1。

- (4) test 指令不改变任何通用寄存器，但根据以下“与”操作改变标志：R[dl] and 0x80

$$\begin{array}{r}
1000\ 0000 \\
\text{and } 1000\ 0000 \\
\hline
1000\ 0000
\end{array}$$

由 3.3.3 节可知，TEST 指令执行后 OF=CF=0；因为结果不为 0，故 ZF=0；因为最高位为 1，故 SF=1。

- (5) 指令功能为：M[R[eax]+R[edx]]←M[R[eax]+R[edx]]\*32，即存储单元 0x8049380 中的内容改变为以下运算的结果：M[0x8049380]\*32=0x908f12a8\*32，也即只要将 0x908f12a8 左移 5 位即可得到结果。

$$\begin{aligned}
&1001\ 0000\ 1000\ 1111\ 0001\ 0010\ 1010\ 1000 \ll 5 \\
&= 0001\ 0001\ 1110\ 0010\ 0101\ 0101\ 0000\ 0000
\end{aligned}$$

因此，指令执行后，单元 0x8049380 中的内容改变为 0x11e25500。显然，这个结果是溢出的。但是，根据表 3.5 可知，乘法指令不影响标志位，也即并不会使 OF=1。

- (6) 指令功能为：R[cx] ←R[cx]−1，即 CX 寄存器的内容减一。

$$\begin{array}{r}
0000\ 0000\ 0001\ 0000 \\
+ 1111\ 1111\ 1111\ 1111 \\
\hline
1\ 0000\ 0000\ 0000\ 1111
\end{array}$$

因此，指令执行后 CX 中的内容从 0x0010 变为 0x000F。由表 3.5 可知，DEC 指令会影响 OF、ZF、SF，根据上述运算结果，得到 OF=0，ZF=0，SF=0。

10. 从汇编代码的第 2 行和第 4 行看，y 应该是占 8 个字节，R[ebp]+20 开始的 4 个字节为高 32 位字节，记为  $y_h$ ；R[ebp]+16 开始的 4 个字节为低 32 位字节，记为  $y_l$ 。根据第 4 行为无符号数乘法指令，得知 y 的数据类型 num\_type 为 unsigned long long。

```

movl    12(%ebp), %eax    //R[edx]←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[edx], 将 yh*x 的低 32 位送 ECX
mull    16(%ebp)          //R[edx]R[ecx]←M[R[ebp]+16]*R[ecx], 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[edx], 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

```

11. 根据第 3.3.4 节得知, 条件转移指令都采用相对转移方式在段内直接转移, 即条件转移指令的转移目标地址为: (PC)+偏移量。

(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 FFFFFFF0H, 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+0xffffffff=0x804829b-0x100=0x804819b。

14. (1) 每个入口参数都要按 4 字节边界对齐, 因此, 参数 x、y 和 k 入栈时都占 4 个字节。

```

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[edx]←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 必须保存到栈中。

(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. 根据第 2、3 行指令可知，参数 a 是 char 型，参数 p 是指向 short 型变量的指针；根据第 4、5 行指令可知，参数 b 和 c 都是 unsigned short 型，根据第 6 行指令可知，test 的返回参数类型为 unsigned int。因此，test 的原型为：  
unsigned int test(char a, unsigned short b, unsigned short c, short \*p);

其内容如下：

19. 第 1 行汇编指令说明参数 x 存放在 EBX 中，根据第 4 行判断 x=0 则转.L2，否则继续执行第 5~10 行指令。根据第 5、6、7 行指令可知，入栈参数 nx 的计算公式为  $x \gg 1$ ；根据第 9、10、11 行指令可知，返回值为  $(x \& 1) + rv$ 。由此推断出 C 缺失部分如下：

```
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  }
```

该函数的功能为计算 x 的各个数位中 1 的个数。

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, %ecx, 2), %eax
&S[i]-S	int	$(A_S+2*i-A_S)/2=i$	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. 根据汇编指令功能可以推断最终在 EAX 中返回的值为：

$M[a+28*i+4*j]+M[b+20*j+4*i]$ ，因为数组 a 和 b 都是 int 型，每个数组元素占 4B，因此，M=5, N=7。

23. 执行第 11 行指令后，a[i][j][k]的地址为  $a+4*(63*i+9*j+k)$ ，所以，可以推断出 M=9, N=63/9=7。根据第 12 行指令，可知数组 a 的大小为 4536 字节，故  $L=4536/(4*L*M)=18$ 。

28. Windows 平台要求不同的基本类型按照其数据长度进行对齐。每个成员的偏移量如下：

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

结构总大小为 48 字节，因为其中的 d 和 g 必须是按 8 字节边界对齐，所以，必须在末尾再加上 4 个字节，即 44+4=48 字节。变量长度按照从大到小顺序排列，可以使得结构所占空间最小，因此调整顺序后的结构定义如下：

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

```

        void      *v;
        short     s;
        char      c;
    } test;
d   g   i   p   l   v   s   c
0   8   16  20  24  28  32  34  结构总大小为 34+6=40 字节。

```