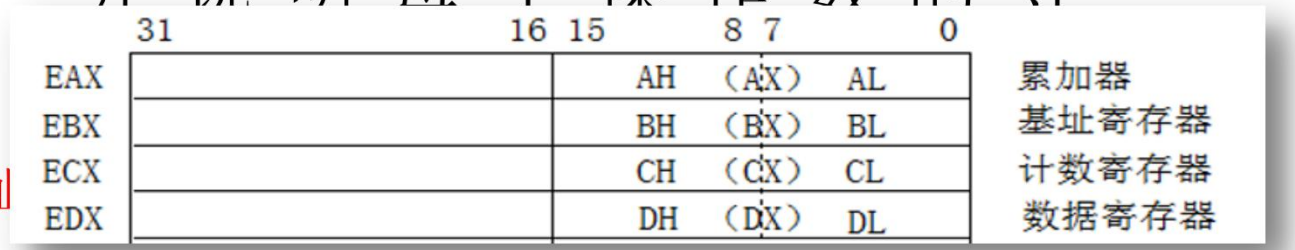
第三章 习题

C语言声明	Intel操作数类型	汇编指令长度后缀	存储长度（位）
(unsigned) char	整数 / 字节	b	8
(unsigned) short	整数 / 字	w	16
(unsigned) int	整数 / 双字	l	32
(unsigned) long int	整数 / 双字	l	32
(unsigned) long long int	-	-	2×32
char *	整数 / 双字	l	32
float	单精度浮点数	s	32
double	双精度浮点数	l	64
long double	扩展精度浮点数	t	80 / 96

扁指令，根据操作数的长度确定对应指
缀，并说明每个操作数的寻
器，变址



（1） mov 8(%ebp, %ebx, 4), %ax	（1）后缀：w， 源：基址+比例变址+偏移，	目：寄存器
（2） mov %al, 12(%ebp)	（2）后缀：b， 源：寄存器，	目：基址+偏移
（3） add (, %ebx,4), %ebx	（3）后缀：l， 源：比例变址，	目：寄存器
（4） or (%ebx), %dh	（4）后缀：b， 源：基址，	目：寄存器
（5） push $0xF8	（5）后缀：l， 源：立即数，	目：栈
（6） mov $0xFFF0, %eax	（6）后缀：l， 源：立即数，	目：寄存器
（7） test %cx, %cx	（7）后缀：w， 源：寄存器，	目：寄存器
（8） lea 8(%ebx, %esi), %eax	（8）后缀：l， 源：基址+变址+偏移，	目：寄存器

本题可参考第三章PPT的22页与23页的表来做题

5. 假设变量 x 和 ptr 的类型声明如下：

参考课本例题3.3

```
src_type   x;
dst_type   *ptr;
```

这里，src_type 和 dst_type 是用 typedef 声明的数据类型。有以下一个 C 语言赋值语句：

```
*ptr = (dst_type) x;
```

若 x存储在寄存器 EAX 或 AX或AL中，ptr存储在寄存器EDX 中，则对于表 3.12 中给出的 src_type 和dst_type的类型组合，写出实现上述赋值语句的机器级代码。要求用 AT&T格式汇编指令表示机器级代码。

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) 正确的！！！
int	int	movl %eax, (%edx)

6. 假设某个 C 语言函数 func 的原型声明如下：

void func(int *xptr, int *yptr, int *zptr);

函数 func 的过程体对应的机器级代码用 AT&T汇编形式表示如下：

```
1    movl    8(%ebp), %eax
2    movl    12(%ebp), %ebx
3    movl    16(%ebp), %ecx
4    movl    (%ebx), %edx
5    movl    (%ecx), %esi
6    movl    (%eax), %edi
7    movl    %edi, (%ebx)
8    movl    %edx, (%ecx)
9    movl    %esi, (%eax)
```

（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;
}
```

请回答下列问题或完成下列任务。

a) 在过程体开始时三个入口参数对应实参所存放的存储单元地址是什么？（提示：当前栈帧底部帧指针寄存器 EBP 指示）

b) 根据上述机器级代码写出函数func 的 C 语言代码。

定点算术运算指令汇总

指令	显式操作数	影响的常用标志	操作数类型	AT&T 指令助记符	对应 C 运算符
ADD	2 个	OF、ZF、SF、CF	无/带符号整数	addb、addw、addl	+
SUB	2 个	OF、ZF、SF、CF	无/带符号整数	subb、subw、subl	−
INC	1 个	OF、ZF、SF	无/带符号整数	incb、incw、incl	++
DEC	1 个	OF、ZF、SF	无/带符号整数	decb、decw、decl	−−
NEG	1 个	OF、ZF、SF、CF	无/带符号整数	negb、negw、negl	−
CMP	2 个	OF、ZF、SF、CF	无/带符号整数	cmpb、cmpw、cmpl	<, <=, >, >=
MUL	1 个	OF、CF	无符号整数	mulb、mulw、mull	*
MUL	2 个	OF、CF	无符号整数	mulb、mulw、mull	*
MUL	3 个	OF、CF	无符号整数	mulb、mulw、mull	*
IMUL	1 个	OF、CF	带符号整数	imulb、imulw、imull	*
IMUL	2 个	OF、CF	带符号整数	imulb、imulw、imull	*
IMUL	3 个	OF、CF	带符号整数	imulb、imulw、imull	*
DIV	1 个	无	无符号整数	divb、divw、divl	/, %
IDIV	1 个	无	带符号整数	idivb、idivw、idivl	/, %

8. 假设以下地址以及寄存器中存放的机器数如下表：

分别说明执行以下指令后，哪些地址或寄存器中的内容会发生改变？改变后的内容是什么？条件标志 OF、 SF、 ZF 和 CF 会发生什么改变？

地址	机器数	寄存器	机器数
0x8049300	0xfffffff0	EAX	0x08049300
0x8049400	0x80000008	EBX	0x00000100
0x8049384	0x80f7ff00	ECX	0x00000010
0x8049380	0x908f12a8	EDX	0x00000080

（1） addl (%eax), %edx

（2） subl (%eax, %ebx), %ecx

（3） orw 4(%eax, %ecx, 8), %bx

（4） testb $0x80, %dl

（5） imull $32, (%eax, %edx)

（6） decw %cx

第三章PPT-34页或书本表3.5

（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+FFFFFFF0H

```
  0000 0000 0000 0000 0000 0000 1000 0000
+ 1111 1111 1111 1111 1111 1111 1111 0000
-----------------------------------------
1 0000 0000 0000 0000 0000 0000 0111 0000
```

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

```
  0000 0000 0000 0000 0000 0000 0001 0000
+ 0111 1111 1111 1111 1111 1111 1111 1000
-----------------------------------------
0 1000 0000 0000 0000 0000 0000 0000 1000
```

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

```
    0000 0001 0000 0000
or  1111 1111 0000 0000
-------------------------
    1111 1111 0000 0000
```

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

```
    1000 0000
and 1000 0000
-------------
    1000 0000
```

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**，也即存储单元0x8049380中的内容0x908f12a8与立即数32相乘，最后将乘积的低32位存放在ECX寄存器中。M[0x8049380]*32=0x908f12a8*32，乘法器将得到64位乘积，其结果就是将0x908f12a8先符号扩展成64位数，然后再左移5位所得到的64位结果，即：

1111 1111 1111 1111 1111 1111 1111 1111 1001 0000 1000 1111 0001 0010 1010 1000<<5

= 1111 1111 1111 1111 1111 1111 1111 1111 0010 0001 0001 1110 0010 0101 0101 0000 0000

因此，指令执行后，ECX寄存器中的内容改变为0x11e25500。因为高33位不是全0，也不是全1，因此OF=CF=1。

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

```
  0000 0000 0001 0000
+ 1111 1111 1111 1111
-------------------------
1 0000 0000 0000 1111
```

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. 假设函数 product 的 的 C 语言代码如下，其中 num_type 是用 typedef 声明的数据类型。

```
void product(num_type *d, unsigned x, num_type y ) {
    *d = x*y;
}
```

```
1    movl    12(%ebp), %eax   //R[eax]←M[R[ebp]+12]，将 x 送 EAX
2    movl    20(%ebp), %ecx   //R[ecx]←M[R[ebp]+20]，将 yh送 ECX
3    imull   %eax, %ecx       //R[ecx]←R[ecx]*R[eax]，将 yh*x 的低 32 位送 ECX
4    mull    16(%ebp)         //R[edx]R[eax]←M[R[ebp]+16]*R[eax]，将 yl*x 送 EDX-EAX
5    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
6    movl    8(%ebp), %ecx    //R[ecx]←M[R[ebp]+8]，将 d 送 ECX
7    movl    %eax, (%ecx)     //M[R[ecx]]←R[eax]，将 x*y 低 32 位送 d 指向的低 32 位
8    movl    %edx, 4(%ecx)    //M[R[ecx]+4]←R[edx]，将 x*y 高 32 位送 d 指向的高 32 位
```

请给出上述每条汇编指令的注释，并说明 num_type 是什么类型。

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。

11. 已知 IA-32 是小端方式处理器，根据给出的 IA-32 机器代码的反汇编结果回答问题。

(1) 已知 je 指令的操作码为 01110100，je 指令的转移目标地址是什么？call 指令中的转移目标地址 0x80483b1 是如何反汇编出来的？

804838c : 74 08　　je　　xxxxxxx

804838e: e8 1e 00 00 00　　call　　80483b1<test>

（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) 已知 jb 指令的操作码为 01110010，jb 指令的转移目标地址是什么？movl 指令中的目的地址如何反汇编出来的？

8048390: 72 f6　　jb　　xxxxxxx

8048392: c6 05 00 a8 04 08 01　　movl　$0x1, 0x804a800

8048399: 00 00 00

（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) 已知 jle 指令的操作码为 01111110，mov 指令的地址是什么？

xxxxxxx: 7e 16　　jle　　80492e0

xxxxxxx: 89 d0　　mov　　%edx, %eax

（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) 已知 jmp 指令的转移目标地址采用相对寻址方式，jmp 指令操作码为 11101001，其转移目标地址是什么？

8048296: e9 00 ff ff ff　　jmp　　xxxxxxx

804829b: 29 c2　　sub　　%eax, %edx

（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. 已知函数 do_loop的 C 语言代码如下：

```
1   short do_loop(short x, short y, short k) {
2       do {
3           x*=(y%k) ;
4           k--;
5       } while ((k>0) && (y>k));
6       return x;
7   }
```

请回答下列问题或完成下列任务：

a) 给每条汇编指令添加注释，并说明每条指令执行后，目的寄存器中存放的是什么信息？

b) 上述函数过程体中用到了哪些被调用者保存寄存器和哪些调用者保存寄存器？在该函数过程体前面的准备阶段哪些寄存器必须保存到栈中？

c) 为什么第 7 行中的 DX 寄存器需要算术右移 15 位？

```
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
9    imulw  %dx, %bx            //R[ax]←R[dx-ax]÷R[cx]的商，将 y/k 送 AX
10   decw   %cx                 //R[bx]←R[bx]*R[dx]，将 x*(y%k) 送 BX
11   testw  %cx, %cx            //R[cx]←R[cx]-1，将 k-1 送 CX
12   jle    .L2                 //R[cx] and R[cx]，得 OF=CF=0，负数则 SF=1,零则 ZF=1
13   cmpw   %cx, %si            //若 k 小于等于 0，则转.L2
14   jg     .L1                 //R[si] - R[cx]，将 y 与 k 相减得到各标志
15 .L2:                         //若 y 大于 k，则转.L1
16   movswl %bx, %eax           // R[eax]←R[bx]，将 x*(y%k) 送 AX
```

b) 被调用者保存寄存器有BX 、SI，调用者保存寄存器有 AX 、CX 和 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 和和 ESI 必须保存到栈中。

c) 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，采用算术右移展以扩展 16 位符号，放在高 16 位的 DX 中，低 16 位在 AX

17. 已知函数 test 的入口参数有 a 、b 、c和 p ， C 语言过程体代码如下：

```
*p = a;
    return b*c;
```

函数 test过程体对应的汇编代码如下：

```
1    movl     20(%ebp), %edx
2    movsbw       8(%ebp), %ax
3    movw     %ax, (%edx)
4    movzwl   12(%ebp), %eax
5    movzwl    16(%ebp), %ecx
6    mull      %ecx, %eax
```

写出函数 test 的原型，给出返回参数的类型以及入口参数 a 、b 、c和 p的类型顺序。

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);
```

19. 已知递归函数 refunc的C函数 test过程体对应的汇编代码如下：
根据对应的汇编代码填写 C代码中缺失部分，并说明函数的功能

```
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   }
```

```
1   movl    8(%ebp), %ebx
2   movl    $0, %eax
3   testl   %ebx, %ebx
4   je      .L2
5   movl    %ebx, %eax
6   shrl    $1, %eax
7   movl    %eax, (%esp)
8   call    refunc
9   movl    %ebx, %edx
10  andl    $1, %edx
11  leal    (%edx, %eax), %eax
12 .L2:
    ......
    ret
```

该函数的功能为计算 *x* 的各个数位中 1 的个数。

21. 假设 short 型数组 S 的首地址 As 和数组下标（索引）变量 i（int 型）分别存放在寄存器 EDX和ECX 中，下列给出的表达式的结果存放在 EAX 或 AX 中，仿照例子填写表，说明表达式的类型、值和相应的汇编代码。

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=*i*	*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. 假设函数 sumij的 C 代码如下，其中，M和N是用#define声明的常数。

```
1    int a[M][N],   b[N][M];
2
3    int sumij(int i, int j) {
4          return a[i][j] + b[j][i];
5    }
```

```
1    movl     8(%ebp), %ecx
2    movl     12(%ebp), %edx
3    leal     (,%ecx, 8), %eax
4    subl     %ecx, %eax
5    addl     %edx, %eax
6    leal     (%edx, %edx, 4), %edx
7    addl     %ecx, %edx
8    movl     a(, %eax, 4), %eax
9    addl     b(,%edx, 4), %eax
```

根据上述汇编代码，确定 *M* 和 *N* 的值。

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. 假设函数 st_ele 的 C 代码如下，其中，L 、M和N是用#define声明的常数。根据下述汇编代码，确定 L 、M和N的值

```
1   int a[L][M][N];
2
3   int st_ele(int i, int j, int k, int *dst) {
4       *dst = a[i][j][k];
5       return sizeof(a);
6   }
```

```
1   movl    8(%ebp), %ecx
2   movl    12(%ebp), %edx
3   leal    (%edx,%edx, 8), %edx
4   movl    %ecx, %eax
5   sall    $6, %eax
6   subl    %ecx, %eax
7   addl    %eax, %edx
8   addl    16(%ebp), %edx
9   movl    a(, %edx, 4), %eax
10  movl    20(%ebp), %edx
11  movl    %eax, (%edx)
12  movl    $4536, %eax
```

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，可以推断出中间的 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*N*M)=18。

28. 以下是结构 test 的声明：

假设在 Windows 平台上编译，则这个结构中每个成员的偏移量是多少？结构总大小为多少字节？如何调整成员的先后顺序使得结构所占空间最小？

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

计算结构体大小的规则：

a) 每一个成员的偏移量都必须是该成员的倍数。
b) 结构体的大小必须是所有成员大小的倍数。

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。