**第三章作业**

**6**

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

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

{int a=\*yptr;

Int b=\*zptu;

Int c=\*xptu;

\*yptr=c;

\*zptu=a;

\*xptu=b}

**7**

1. R[edx]=x
2. R[edx]=x+y+4;
3. R[edx]=x+y\*8
4. R[edx]=y+x\*2+12
5. R[edx]=4\*y
6. R[edx]=x+y

**9**

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

缺失部分为：int v= x\*k-(z&0xFFF0+y\*256) ;

**12**

(1)

1 movb 8(%ebp), %dl //R[dl]←M[R[ebp]+8]，将*x*送DL

2 movl 12(%ebp), %eax //R[eax]←M[R[ebp]+12]，将*p*送EAX

3 testl %eax, %eax //R[eax] and R[eax]，判断*p*是否为0

4 je .L1 //若*p*为0，则转.L1执行

5 testb $0x80, %dl //R[dl] and 80H，判断*x*的第一位是否为0

6 je .L1 //若*x*>=0，则转.L1执行

7 addb %dl, (%eax) //M[R[eax]]←M[R[eax]]+R[dl]，即\*p+=x

8 .L1:

因为C语言if语句中的条件表达式可以对多个条件进行逻辑运算，而汇编代码中一条指令只能进行一种逻辑运算，并且在每条逻辑运算指令生成的标志都是存放在同一个EFLAGS寄存器中，所以，最好在一条逻辑指令后跟一条条件转移指令，把EFLAGS中标志用完，然后再执行另一次逻辑判断并根据条件进行转移的操作。

(2)

1 void comp(char x, int \*p)

2 {

3 if (p!=0)

4 if (x<0)

5 \*p += x;

6 }

**(问题：题中的汇编代码的6 je 应该改成jge 吧)**

**13**

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 }

**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[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。

**15**

1 int f1(unsigned x)

2 {

3 int y = 0 ;

4 while ( x!=0 ) {

5 y ^=x ;

6 x>>=1 ;

7 }

8 return y&0x1 ;

9 }

函数f1的功能返回：( x ^ x>>1 ^ x>>2 ^ ….) & 0x1，因此f1用于检测x的奇偶性，当x中有奇数个1，则返回为1，否则返回0。

**16**

函数sw只有一个入口参数*x*，根据汇编代码的第2~5行指令知，当*x*+3>7时转标号.L7处执行，否则，按照跳转表中的地址转移执行，*x*与跳转目标处标号的关系如下：

*x*+3=0：.L7

*x*+3=1：.L2

*x*+3=2：.L2

*x*+3=3：.L3

*x*+3=4：.L4

*x*+3=5：.L5

*x*+3=6：.L7

*x*+3=7：.L6

由此可知，switch (x) 中省略的处理部分结构如下：

case -2:

case -1:

…… // .L2标号处指令序列对应的语句

break;

case 0:

…… // .L3标号处指令序列对应的语句

break;

case 1:

……. // .L4标号处指令序列对应的语句

break;

case 2:

…… // .L5标号处指令序列对应的语句

break;

case 4:

…… // .L6标号处指令序列对应的语句

break;

default:

…… // .L7标号处指令序列对应的语句

**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);

**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*>>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。

**25**

（1）node所需存储空间需要4+(4+4)+4=16字节。成员p、s.x、s.y和next的偏移地址分别为0、4、8和12。

（2）np\_init中缺失的表达式如下：

void np\_init(struct node \*np)

{

np->s.x = np->s.y ;

np->p = &(np->s.x) ;

np->next= np ;

}

**29**

（1）执行第7行和第10行指令后栈中的信息存放情况如下图所示。其中gets函数的入口参数为buf数组首地址，应等于getline函数的栈帧底部指针EBP的内容减0x14，而getline函数的栈帧底部指针EBP的内容应等于执行完getline中第2行指令（push %ebp）后ESP的内容，此时，R[esp]= =0xbffc07f0-4=0xbffc07ec，故buf数组首地址为R[ebp]-0x14= R[esp]-0x14=0xbffc07ec-0x14=0xbffc07d8。

a) 执行第7行后的栈

ESP

EBP

EBP在P中旧值

08 04 85 c8

返回P的地址

bf fc 08 00

00 00 00 08

00 00 00 10

00 00 00 05

被调用者保存寄存器在P中的旧值

buf[7]~ buf[4]

buf[3]~ buf[0]

b) 执行第10行后的栈

ESP

EBP

EBP在P中旧值

08 41 39 38

返回P的地址

37 36 35 34

33 32 31 30

46 45 44 43

42 41 39 38

被调用者保存寄存器在P中的旧值

buf[7]~ buf[4]

buf[3]~ buf[0]

33 32 31 30

37 36 35 34

gets入口参数

bf fc 07 d8

（2）当执行到getline的ret指令时，假如程序不发生段错误，则正确的返回地址应该是0x80485c8，发生段错误是因为执行getline的ret指令时得到的返回地址为0x8413938，这个地址所在存储段可能是不可执行的数据段，因而发生了段错误（segmentation fault）。

（3）执行完第10行汇编指令后，被调用者保存寄存器EBX、ESI和EDI在P中的内容已被破坏，同时还破坏了EBP在P中的内容。

（4）getline的C代码中malloc函数的参数应该为strlen(buf)+1，此外，应该检查malloc函数的返回值是否为NULL。