1. 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)

请回答下列问题或完成下列任务

1. 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？
2. 根据上述机器级代码写出函数func的C语言代码

答案：

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;

}

1 movl 8(%ebp), %eax -- eax=&x

2 movl 12(%ebp), %ebx – ebx=&y

3 movl 16(%ebp), %ecx --ecx=&z

4 movl (%ebx), %edx edx=y

5 movl (%ecx), %esi esi=z

6 movl (%eax), %edi edi=x

7 movl %edi, (%ebx) \*y=x

8 movl %edx,( %ecx) \*z=y

9 movl &esi, (%eax) \*x=z

1. 假设变量x和y分别存放在寄存器EAX和ECX中，请给出以下每条指令执行后寄存器EDX中的结果。
2. leal (%eax),%edx
3. Leal 4(%eax,%ecx),%edx
4. Leal (%eax,%ecx,8),%edx
5. Leal 0xC(%ecx,%eax,2),%edx
6. Leal (,%eax,4),%edx
7. Leal (%eax,%ecx),%edx

答案：

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

9.假设函数operate的部分c代码如下：

1 int operate(int x, int y, int z, int k)

2 {

3 int v=\_\_\_\_\_\_;

4 return v;

5 }

以下汇编代码用来实现第三行语句的功能，请写出每条汇编指令的注释，并根据以下汇编代码，填写operate函数缺失的部分

1 movl 12(%ebp), %ecx

2 sall $8, %ecx

3 movl 8(%ebp), %eax

4 movl 20(%ebp), %edx

5 imull %edx, %eax

6 movl 16(%ebp), %edx

7 andl $65520, %edx

8 addl %ecx, %edx

9 subl &edx, %eax

答案：

1 movl 12(%ebp), %ecx //R[ecx]<-M[R[ebp]+12],将y送ECX

2 sall $8, %ecx // R[ecx]<- R[ecx]<<8,将y\*256送ECX

3 movl 8(%ebp), %eax // R[eax]<- M[R[ebp]+8]，将x送EAX

4 movl 20(%ebp), %edx // R[edx]<- M[R[ebp]+20]，将k送EDX

5 imull %edx, %eax // R[eax]<- R[eax]\* R[edx]，将x\*k送EAX

6 movl 16(%ebp), %edx // R[edx]<- M[R[ebp]+16]，将z送EDX

7 andl $65520, %edx // R[edx]<- R[edx]&65520，将z&0xfff0送EDX

8 addl %ecx, %edx // R[edx]<- R[edx]+ R[ecx]，将z&oxfff0+y\*256送EDX

9 subl &edx, %eax // R[eax]<- R[eax]- R[edx]，将x\*k-(z&0xfff0+y\*256)送EAX

第三行缺失部分为：

3 int v = x\*k-(z&0xfff0+y\*256);

1. 已知函数comp的C语言代码及其过程体对应的汇编代码

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

2 {

1. if(p&&x<0)
2. \*p+=x;

5 }

1. movb 8(%ebp),%dl
2. Movl 12(%ebp),%eax
3. Test1 %eax,%eax
4. Je .L1
5. Testb $0x80,%dl
6. Je .L1
7. Addb %dl,(%eax)
8. .L1:

要求回答下列问题或完成下列任务。

1. 给出每条汇编指令的注释，并说明为什么c代码只有一个if语句而汇编代码有两条条件转移指令
2. 按照主教材中图3.22给出的“if()goto...”语句形式写出汇编代码对应的C语言代码。

答案：

1. 汇编指令的注释如下：
2. movb 8(%ebp),%dl //R[dl]<-M[R[ebp]+8],将x送DL
3. Movl 12（%ebp）,%eax //R[eax]<-M[R[ebp]+12],将p送EAX
4. Test1 %eax,%eax //R[eax] and R[eax],判断p是否为0
5. Je .L1 //若p为0，则转.L1
6. Testb $0x80,%dl //R[dl] and 80H,判断x的第一位是否为0
7. 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申请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

1. 按照主教材中图3.22给出的“if()goto...”语句形式写出汇编代码对应的C语言代码如下：

Void comp(char x,int \*p)

{

If(p!=0)

If(x<0)

\*p+=x;

}

1. 已知函数func的C语言代码框架及其过程体对应的汇编代码如图3.2所示，根据对应的汇编代码填写C代码中缺失的表达式。

1 mov1 8(%ebp), %eax

2 mov1 12(%ebp), %edx

3 cmp1 $-100, %eax

4 jg .L1

5 cmpl %eax , %edx

6 jle .L2

7 addl %edx, %eax

8 jmp .L3

9 .L2：

10 sub1 %edx， %eax

11 jmp .L3

12 .L1:

13 cmp1 $16, %eax

14 j1 .L4

15 and1 %edx,%eax

16 jmp .L3

17 .L4

18 imull %edx, %eax

19 .L3

1 int func (int x, int y)

2 {

3 int z = \_\_\_\_\_\_\_;

4 if (\_\_\_\_\_) {

5 if (\_\_\_\_\_)

6 z = \_\_\_\_\_\_\_;

7 else

8 z= \_\_\_\_\_\_\_\_;

9 } else if (\_\_\_\_\_\_\_)

10 z = \_\_\_\_\_;

11 return z;

12 }

分析解答

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.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 }

函数do\_loop的过程体对应的汇编代码如下：

1 movw 8 ( % ebp ) ,%bx

2 movw 12 ( % ebp ), % si

3 movw 16 ( % ebp ), %cx

4 . LI :

5 movw %si， %dx

6 movw %dx，%ax

7 sarw $15, %dx

8 idiv %cx

9 imulw %dx，%bx

10decw %cx

11testw %cx，%cx

12jle .L2

13cmpw %cx, %si

14jp .L1

15.L2:

16movswl %bx, %eax

请回答下列问题或完成下列任务：

（1）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？ （2）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?

（3）为什么第7行中的DX寄存器需要算术右移15位？

答案：

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 . LI :

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[dx]<-R[bx]\*R[dx],将x\*(y%k)送BX

10decw %cx //R[cx]<-R[cx]-1,将k-1送CX

11testw %cx ,%cx // R[cx] and R[cx],得OF=CF=0，负数则SF=1，零则 //ZF=1

12jle .L2 //若k小于等于0，则转.L2

13cmpw %cx,%si //R[si]-R[cx]-1,将y与k相减得到各标志

14jp .L1 //若k大于等于0，则转.L1

15.L2

16movswl %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.已知函数f1的C语言代码框架及其过程体对应的汇编代码如图3.3所示，根据对应的汇编代码填写c代码中缺失的部分，并说明函数f1的功能

答案：

1 int { fl ( 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的功能是返回（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.已知函数sw的C语言代码框架如下：

Int sw(int x){

Int v=0;

Switch(x){

/\*switch语句中的处理部分省略\*/

}

Return v;

}

对函数sw进行编译，得到函数过程体中开始部分的汇编代码以及跳转表如图 3 . 4 所示。回答下列问题:函数sw中的switch语句处理部分标号的取值情况如何？标号的取值在什么情况下执行default分支?哪些标号的取值会执行同一个case分支？

答案：

函数sw只有一个入口参数x，由汇编代码的第 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.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 mov1 20（%edp）, %edx

2 movsbw 8(%edp) , %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人口参数a、b、c和p的类型和顺序。

分析解答：

根据第2、3行的指令可知，参数a是char型，参数p是指向char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语言代码框架如下：

1 int refunc(unsigned x) {

2 if ( \_\_\_\_\_\_\_\_\_\_\_ )

3 return \_\_\_\_\_\_\_\_\_\_\_;

4 unsigned nx = \_\_\_\_\_\_\_\_\_\_\_\_;

5 int rv = refunc(nx) ;

6 return \_\_\_\_\_\_\_\_\_ ;

7 }

上述递归函数过程体对应的汇编代码如下：

1 mov1 8(%esp), %ebx

2 mov1 $0, %eax

3 test1 %ebx, %ebx

4 je .L2

5 mov1 %ebx, %eax

6 Shr1 $1, %eax

7 mov1 %eax, (%esp)

8 call refunc

9 mov1 %ebx, %edx

10 andl $1, %edx

11 leal (%edx, %eax)

12 .L2:

...

ret

根据对应的汇编代码填写C代码中缺失的部分，并说明函数的功能。

分析解答：

第一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5-11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C代码中缺失的部分如下：

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&1）+rv ;

7 }

该函数的功能为计算x的各个数位中1的个数总数。

1. 假设结构类型node的定义、函数np\_init的部分C代码及其对应的部分汇编代码如下：

mov1 8(%ebp), %eax

mov1 8(%eax), %edx

mov1 %edx, 4(%eax)

leal 4(%eax), %edx

mov1 edx, (%eax)

mov1 %eax, 12( %eax )

void np\_init(struct node \*np)

{

np→s.x =\_\_\_\_\_;

np→p = \_\_\_\_\_;

np→next = \_\_\_\_\_\_；

}

struct node {

int \*p:

struct {

int x;

int y;

} s;

struct node \*next;

};

回答下列问题或完成下列任务。

1. 结构node所需存储空间有多少字节？成员p、s.x、s.y和next的偏移地址分别为多少？
2. 根据汇编代码填写np\_init中缺失的表达式。

分析解答：

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\_\_；

}

1. 以下是函数getline存在漏洞和问题的C语言代码实现，右边是其对应的反汇编部分结果：

假定有一个调用过程P调用了函数getline，其返回值为0x80458c8，在过程P中为了调用函数getline而执行call指令后，部分寄存器的内容如下：R[ebp] = 0xbffc0800,R[esp] =0xbffc07f

0,R[ebx] =0x5,R[esi] =0x10,R[edi] =0x8。执行程序时从标准输入读入的一行字符串为“0123456789ABCDEF0123456789A”,此时，程序会发生段错误（segmentation fault）并终止执行，经调试确认错误是在执行getline的ret指令时发生的。回答下列问题或完成下列任务。

1 0804840c <getline>

2 804840c: 55 push %ebp

3 804840d: 89 e5 mov %esp, %ebp

4 804840f： 83 ec 28 sub $0x28, $esp

5 8048412： 89 5d f4 mov %ebx, -0xc(%ebp)

6 8048415： 89 75 f8 mov %esi, -0x8(%ebp)

7 8048418： 89 7d fc mov %edi, -0x4(%ebp)

8 804841b：8d 75 ec lea 0x14(%ebp), %esi

9 804841e：89 34 24 mov %esi, (%esp)

10 8048412：e8 a3 ff ff ff call 80483c9 <gets>

char \*getline()

{

char buf[8];

char \*result;

gets(buf);

result = malloc(strlen(buf))

strcpy(result,buf);

return result;

}

1. 分别画出执行第7行和第10行汇编指令后栈中的信息存放情况。要求给出存储地址和存储内容，并指出存储内容的含义（如返回地址、EBX旧值、局部变量、入口参数等）。
2. 当执行到getline的ret指令时，假如程序不发生段错误，则正确的返回地址是什么？发生段错误是因为执行getline的ret指令时得到了什么样的返回地址?
3. 执行完第10行汇编指令后，哪些寄存器的内容已被破坏？
4. 除了可能发生缓冲区溢出以外，getline的C代码还有哪些错误？

分析解答：

1. 执行第7行指令后，栈中信息存放情况如图3.8a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.
2. 执行第10行指令后，跳转到gets函数执行。在gets函数中。将字符串“0123456789ABCDEF0123456789\0”作为buf的内容写入buf[0](地址为0xbffc07d8)处开始的存储单元中。信息存放情况如图3.8b所示。

备注：0的ASCII码是30，A的ASCII码是41；字符串的结束的标识符\0的ASCII码是0。

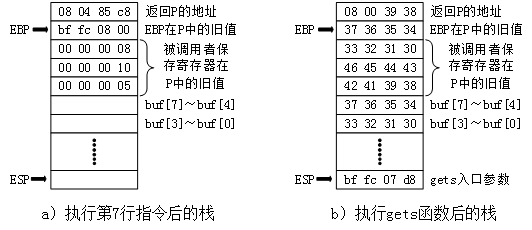


图3.8

1. 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时得到的返回地址为0x8003938，这个地址所在存储段是不可执行的代码段，因而发生了段错误（segmentation fault）
2. 执行完gets函数后，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。
3. 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。