3.32

可根据IA32代码推断出d在12(%ebp)中，\*p在16(%ebp)中，c在8(%ebp)中，x在20(%ebp)中，又由movsbl,movswl推知c为short型，d为char型。

因此函数fun的原型为：int fun(short c, char d, int p, int x);

3.33

1. 起初esp的值为0x800040,pushl后，esp的值为0x800040-0x4=0x80003C，在第3行通过赋值，可知%ebp被设置成了0x80003C
2. %esp的值为0x80003C-0x28=0x800014
3. 由第12行可知是-4(%ebp)中的值减去-8(%ebp)中的值，即x-y，故可知第5行计算的是&x，第7行计算的是&y，因此x的存放地址为0x800038，y的存放地址为0x800034

D.

|  |  |
| --- | --- |
| 0x80003C | 0x800060(%ebp) |
| 0x800038 | 0x53(x) |
| 0x800034 | 0x46(y) |
| 0x800030 |  |
| 0x80002C |  |
| 0x800028 |  |
| 0x800024 |  |
| 0x800020 |  |
| 0x80001C | 0x800038 |
| 0x800018 | 0x800034 |
| 0x800014 | 0x300070(%esp) |

E.0x800020~0x800030是未使用的栈帧区域。

3.34

1. %ebx保存x的值

int rfun(unsigned x) {

if ( x==0 )

return 0;

unsigned nx = x>>1;

int rv = rfun(nx);

return rv + (x & 0x1);

}

C.

计算x中所有位之和。

3.36

|  |  |  |  |
| --- | --- | --- | --- |
| 表达式 | 类型 | 值 | 汇编代码 |
| S+1 | short \* |  | leal 2(%edx),%eax |
| S[3] | short |  | movw 6(%edx),%ax |
| &S[i] | short \* |  | leal (%edx,%ecx,2),%eax |
| S[4\*i+1] | short |  | movw 2(%edx,%ecx,8),%ax |
| S+i-5 | short \* |  | leal -10(%edx,%ecx,2),%eax |

3.37

由3-5行可知第8行%eax为mat1[7\*i+j]，由6-7行知第9行%eax为mat1[7\*i+j]+mat2[5\*j+i]，故M=5，N=7

3.39

A.

p:占用0-3，偏移量为0；

s.x占用4-7，偏移量为4；

s.y占用8-11，偏移量为8；

next占用12-15，偏移量为12；

B.

需要16个字节；

C.

首先解析一下汇编代码：第1行为获取sp，第2行为获取sp->s.y，第3行为sp->s.x = sp->s.y，第4行为复制sp->x，第5行为sp->p = &(sp->s.x)，第6行为sp->next = sp;

因此

void sp\_init(struct prob \*sp)

{

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

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

sp->next = sp;

}

3.40

|  |  |  |
| --- | --- | --- |
| EXPR | TYPE | Code |
| up->t1.s | int | movl 4(%eax), %eax  movl %eax, (%edx) |
| up->t1.v | short | movw (%eax), %ax  movw %ax, (%edx) |
| &up->t1.d | short \* | leal 2(%eax), %eax  movl %eax, (%edx) |
| up->t2.a | int \* | movl %eax, (%edx) |
| up->t2.a[up->t1.s] | int | movl 4(%eax), %ecx  movl (%eax, %ecx, 4), %eax  movl %eax, (%edx) |
| \*up->t2.p | char \* | movl 8(%eax), %eax  movb (%eax), %al  movb %al, (%edx) |

3.41

A: 偏移量：i:0, c:4, j:8, d:12，结构总大小为16，对齐要求为4

B: 偏移量：i:0, c:4, d:5, j:8，结构总大小为12，对齐要求为4

C: 偏移量：w:0, c:6，结构总大小为10，对齐要求为2

D: 偏移量：w:0, \*c:8，结构总大小为20，对齐要求为4

E: 偏移量：a:0, \*p:32，结构总大小为36，对齐要求为4

3.60

A.

扩展到三维后，数组元素A[i][j][k]的位置即为

B.

由第8行可知括号内S\*T\*i+T\*j+k=63\*i+9\*j+k，故S\*T=63，T=9，故S=7，又由第12行知2772/4=R\*S\*T，故R=11，因此R=11, T=9, S=7；

3.62

A.

由汇编代码第7行知M=76/4=19

B.

由汇编代码第8、9行知，判断条件为j<i，即j-i<0，因此i保存在%edi，j保存在%ecx

C.

void transpose(int A[M][M]) {

int i,j;

for (i=0; i<M; i++)

{

int \*pt\_a=&A[i][0];

int \*pt\_b=&A[0][i];

for (j = 0; j<i; j++) {

int t = \*pt\_a;

\*pt\_a = \*pt\_b;

\*pt\_b = t;

++pt\_a;

pt\_b+=M;

}

}

}

3.63

由汇编代码的第4行知%esi中保存着3n，即为E1(n)=3n；

由第8行知%ebx中保存着4\*E2(n)，又%ebx的值为4\*(2n-1)，故E2(n)=2n-1；

3.64

A.

第5行将result加载到%eax，第6行将s1.p加载到%ebx，第7行将s1.v加载到%edx

B.

首先，由第14行可以对应return s2.prob - s2.sum，故-4(%dbp)保存s2.sum

其次，由第13行可以推出-8(%dbp)保存s2.prob

又由第7，第9行可知s2.prob其下是s1.v和s1.p，最后保存的是&s2

故从%ebp到%esp，依次为s2.sum, s2.prob, s2.v, s2.p和&s2

C.

将欲传递的结构分解为结构的各个成员，依次传递

D.

将返回值变量的地址进行传递

3.66

1. CNT=196/28= 7
2. 由test函数中ap->x[ap->idx]=n可知idx是a\_struct的一个成员，且是int型；x[]也是一个成员，共有6个元素；

typedef struct {

int idx;

int x[6];

} a\_struct;

3.67

A.偏移量

e1:p: 0

e1.x: 4

e2:y: 0

e2.next: 4

B.共需要8个字节

C.

void proc (union ele \*up)

{

 up->e2.next->e1.x=\*(up->e2.next->e1.p) - up->e2.y;

}