1. IEEE 754 单精度浮点数采用 32 位进行表示

1) 该标准中如何表示数据零？请给出对应的 16 进制机器码

2) Intel IA32 平台中定义整型变量 int i，请问 C 语言表达式 i==（int）（float）i 是否恒成立，试给出理由？

3) 同样平台中运行 C 语言代码 float a=2016，如变量 a 存放在地址 0x00003000 处，请分别给出地址 0x00003000~0x00003003 存放的值的十六进制编码，请将答案填写在下表中。

地址	编码（16 进制）
0x00003000	
0x00003001	
0x00003002	
0x00003003	

2. 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。

```
int a[L][M][N];
int st_ele(int i, int j, int k, int *dst)
{
  *dst = a[i][j][k];
  return sizeof(a);
}
```

已知函数 st_ele 的过程体对应的汇编代码如下：

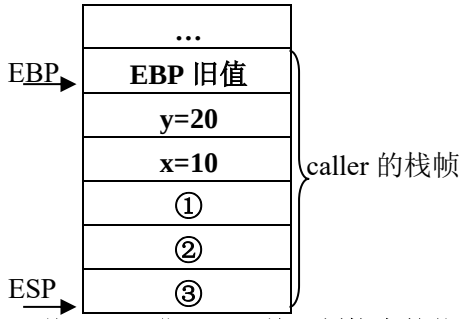
```
1 movl 8(%ebp), %ecx
2 movl 12(%ebp), %edx
3 leal (%edx,%edx, 8), %edx
4 movl %ecx, %eax
5 sall $6, %eax
6 subl %ecx, %eax
7 addl %eax, %edx
8 addl 16(%ebp), %edx
9 movl a(, %edx, 4), %eax
10 movl 20(%ebp), %edx
11 movl %eax, (%edx)
```

12 movl $4536, %eax

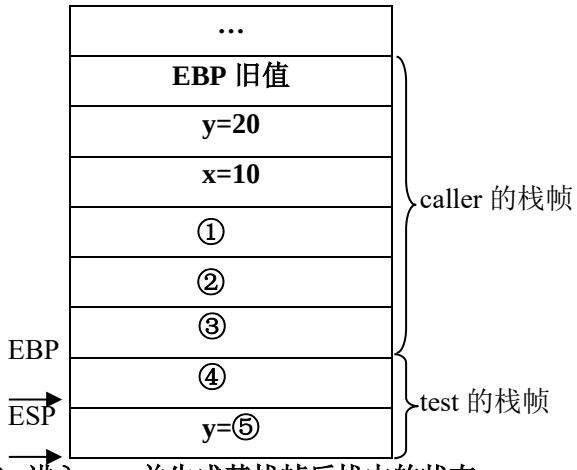
根据上述汇编代码，确定 L、 M 和 N 的值

3. 请根据下面的程序填写相应栈帧中①-⑨的内容。

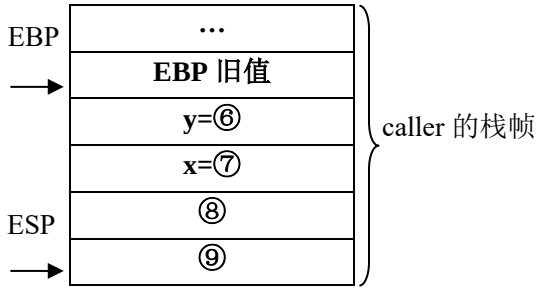
```
void test ( int x, int *ptr )
{     int   y = 10;
      *ptr += x + y;
}
void caller ( )
{     int x = 10, y = 20;
      test (x, &y);
}
```



1） 从 caller 进入 test 前一刻栈中的状态



2） 进入 test 并生成其栈帧后栈中的状态



3） test 执行完回到 caller 时帧中的状态

1） 请问什么是栈帧结构？栈帧结构主要存放那些信息？栈帧资源是否受限？
2） 请将以上三图中的①~⑨内容填写在下表中

请将答案填写在下表中

① **1**		②	
③		⑤	
⑥		⑥	
⑦		⑧	
⑨			

4.遵循 Lab1 的规则要求，给出以下函数的实现。

Lab1 的基本规则：只能使用顺序程序结构；仅能使用限定类型和数量的 C 语言算术和逻辑操作（详见各函数说明）；不得使用超过 8 位表示的常量、强制类型转换、数组/结构/联合等数据类型、宏等；不得定义或调用其他函数等。

级别	函数名	功能	约束条件	最多操作符数量
2	byteXor	比较 x 和 y 的第 n 个字节（字节从 LSB 开始到 MSB 依次编号为 0-3），若不同，则返回 1；若相同，则返回 0	仅能使用! ~ & ^ | + << >>	20
2	mul2OK	计算 2*x，如果不溢出，则返回 1，否则，返回 0	仅能使用 ~ & ^ | + << >>	20

```
/*      byteXor - compare the nth byte of x and y, if it is same, return 0, if not, return 1
  *      example: byteXor(0x12345678, 0x87654321, 1) = 1
                      byteXor(0x12345678, 0x87344321, 2) = 0
  *      Legal ops: ! ~ & ^ | + << >>
  *      Max ops: 20
*/
int byteXor(int x, int y, int n) {

}


/* * mul2OK - Determine if can compute 2*x without overflow
  *      Examples: mul2OK(0x30000000) = 1
  *                      mul2OK(0x40000000) = 0
  *      Legal ops: ~ & ^ | + << >>
```

```
  *      Max ops: 20*/
int mul2OK(int x) {

}
```

5. 已知以下关于 Lab3 Bang 阶段的信息，请完成填空，注意涉及数值全部使用 16 进制

```
08048e6d <test>:
 8048e6d:   55                      push   %ebp
 8048e6e:   89 e5                   mov    %esp,%ebp
 8048e70:   53                      push   %ebx
 8048e71:   83 ec 24                sub    $0x24,%esp
 8048e74:   e8 6e ff ff ff          call   8048de7 <uniqueval>
 8048e79:   89 45 f4                mov    %eax,-0xc(%ebp)
 8048e7c:   e8 6b 03 00 00          call   80491ec <getbuf>
 8048e81:   89 c3                   mov    %eax,%ebx
 8048e83:   e8 5f ff ff ff          call   8048de7 <uniqueval>
......
```

```
080491ec <getbuf>:
 80491ec:   55                      push   %ebp
 80491ed:   89 e5                   mov    %esp,%ebp
 80491ef:   83 ec 38                sub    $0x38,%esp
 80491f2:   8d 45 d8                lea    -0x28(%ebp),%eax
 80491f5:   89 04 24                mov    %eax,(%esp)
 80491f8:   e8 55 fb ff ff          call   8048d52 <Gets>
 80491fd:   b8 01 00 00 00          mov    $0x1,%eax
 8049202:   c9                      leave
 8049203:   c3                      ret
```

```
08048d05 <bang>:
 8048d05:   55                      push   %ebp
 8048d06:   89 e5                   mov    %esp,%ebp
 8048d08:   83 ec 18                sub    $0x18,%esp
 8048d0b:   a1 18 c2 04 08          mov    0x804c218,%eax
 8048d10:   3b 05 20 c2 04 08       cmp    0x804c220,%eax
 8048d16:   75 1e                   jne    8048d36 <bang+0x31>
 8048d18:   89 44 24 04             mov    %eax,0x4(%esp)
 8048d1c:   c7 04 24 e4 a2 04 08    movl   $0x804a2e4,(%esp)
 8048d23:   e8 a8 fb ff ff          call   80488d0 <printf@plt>
……
```

```
Breakpoint 2, 0x080491f2 in getbuf ()
(gdb) info r
eax            0x6f50c1c5   1867563461
ecx            0xf7fbd068   -134492056
edx            0xf7fbd3cc   -134491188
ebx            0x0  0
esp            0x55683458   0x55683458 <_reserved+1037400>
ebp            0x55683490   0x55683490 <_reserved+1037456>
esi            0x55686018   1432903704
edi            0x1  1
eip            0x80491f2    0x80491f2 <getbuf+6>
……
```

```
(gdb) x 0x804c218
0x804c218 <global_value>:   0x00000000
```

```
acd@ubuntu:~/Lab3$ ./makecookie U201414XXX
0x250d38e7
```

```
int global_value = 0;

void bang(int val)
{
    if (global_value == cookie) {
        printf("Aha Bang!: You set global_value to 0x%x.\n", global_value);
        validate(2);
    } else
        printf("Oh Misfire: global_value = 0x%x\n", global_value);
    exit(0);
}
```

1) 调用 getbuf 后的返回地址是: (1)

2) 调用 getbuf 时，存放 getbuf 返回地址的单元是: (2)

3) 调用 getbuf 时，getbuf 的栈帧顶地址是： (3)

4) getbuf 中缓冲区 buf 的起始地址是： (4)

5) 攻击字符串的长度为: (5) 字节

6) 使用攻击字符串填充缓冲区时，所填充的缓冲区的第一个字节的地址是： (6)

7) 以下是所设计的一个带有注释的攻击字符串文件的内容，请填空：

c3 c3 c3 c3

b8 (7)_____ /* mov (8)_____,%eax */

a3 18 c2 04 08 /* mov %eax, (9)_____*/

68 05 8d 04 08 /* push (10)_____*/

c3 /* ret */

90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

(11)_____ (12)_____ (13)_____ 55

8) 上述攻击字符串可使 bang 输出：

(14)_____

6. 如下两个 C 语言源文件：main.c 和 test.c，它们的内容如下图所示，回答下列问题。

```
// main.c
#include <stdio.h>
#include <stdlib.h>

extern int fun(void);

int global_var1 = 0xff00ff00;
int global_var2 = 0x00ff00ff;

int main(int argc, const char *argv[])
{
    printf(" global_var1 = %x\n", global_var1);
//%x 表示打印无符号以十六进制表示的整数
    printf("sizeof(global_var1) = %d\n", sizeof(global_var1));
    printf(" global_var2 = %x\n", global_var2);
    printf("sizeof(global_var2) = %d\n", sizeof(global_var2));
    fun();
    printf("global_var1 = %x\n", global_var1);
    printf("global_var2 = %x\n", global_var2);
    return 0;
}
```

```
// test.c
#include <stdio.h>
```

```
#include <stdlib.h>
#include <string.h>

double global_var1;

int fun(void)
{
    printf("sizeof(global_var1) = %d\n", sizeof(global_var1));

    memset(&global_var1, 0, sizeof(global_var1));
//void *memset(void *s, int ch, size_t n);
//函数解释：将 s 中前 n 个字节用 ch 替换并返回 s 。

    return 0;
}
```

1) 如何判断文件中的强符号和弱符号？

2) 在这两个文件中出现的符号哪些是强符号？哪些是弱符号？

3) 程序执行后打印的结果是什么？

7. 有如下源代码，试回答问题：

```
1.        /*main.c
2.         .rel.text:     Use golbal
3.         .rel.data:     golbal's initization is address of golbal
4.         */
5.
6.        int x = 1;
7.
8.        extern int a;
9.
10.    int *p = &a;
11.
12.    void out(void);
13.    void local(void);
14.
15.    int main(void)
16.    {
17.            x       = *p;
18.            out();
19.            local();
20.            return 0;
21.    }
22.
```

```
23.     void local(void)
24.     {   }
```

```
1.          /*fun.c*/
2.
3.          int a = 3;
4.          void out(void)
5.          {    }
```

main.c 的.rel.text 和.rel.data 节的内容如下：

```
55 Relocation section '.rel.text' at offset 0x868 contains 4 entries:
56   Offest       Info       Type           Sym.Value   Sym.Name
57  00000007    00000f01   R_386_32          00000004     p
58  0000000e    00000e01   R_386_32          00000000     x
59  00000013    00001202   R_386_PC32        00000000     out
60  00000018    00001302   R_386_PC32        00000023     local
61
62 Relocation section '.rel.data' at offset 0x888 contains 1 entries:
63   Offest       Info       Type           Sym.Value   Sym.Name
64  00000004    00001001   R_386_32          00000000     a
```

main.c 在链接前的反汇编代码如下：

```
7   000000000 <main>:
8       0:      55                              push    %ebp
9       1:      89 e5                           mov     %esp, %ebp
10      3:      83 e4 f0                        and     $0xfffffff0, %esp
11      6:      a1 00 00 00 00                  mov     0x0, %eax
12      b:      8b 00                           mov     (%eax),%eax
13      d:      a3 00 00 00 00                  mov     %eax,0x0
14     12:      e8 fc ff ff ff                  call    13 <main+0x13>
15     17:      e8 fc ff ff ff                  call    18 <main+0x18>
16     1c:      b8 00 00 00 00                  mov     $0x0, %eax
17     21:      c9                              leave
18     22:      c3                              ret
```

1) “rel.text”节的 59 行和 64 行告诉了我们什么？
2) 假定链接后，在生成的可执行目标文件中，out()的虚拟地址为 0x8048404,Local()的虚拟地址为 0x80483ff,试补充填写完成下表中 main.c 的可执行目标文件的反汇编代码。并解释其中补充填写内容的计算方法。

```
145  080483dc <main>:
146   80483dc:      55                              push    %ebp
147   80483dd:      89 e5                           mov     %esp, %ebp
148   80483df:      83 e4 f0                        and     $0xfffffff0, %esp
149   80483e2:      a1 8c 96 04 08                  mov     0x804968c, %eax
```

150	80483e7:	8b 00	mov	(%eax),%eax
151	80483e9:	a3 88 96 04 08	mov	%eax,0x8049688
152	80483ee:	e8 _____	call	_____
153	80483f3:	e8 _____	call	_____
154	80483f8:	b8 00 00 00 00	mov	$0x0, %eax
155	80483fd:	c9	leave	
156	80483fe:	c3	ret	