1. (16 pts total) Given the C code on the right:

```c
int bitcnt(int n)
{
    unsigned m = 0;
    while ( n > 0 ) {
        m += n & 0x1;
    }
    return m;
}
```

a) (6 pts) `gcc -S` produces the assembly code below. **Explain** what each line does.

```assembly
pushl %ebp      __save frame pointer_
movl %esp, %ebp __make new frame pointer_
subl $16, %esp  _allocate 16 bytes on stack_
movl $0, -4(%ebp) ___store 0 in m________
jmp L2           ___jump to L2________
    L3:
movl 8(%ebp), %eax ___put n in eax________
andl $1, %eax      ___mask n with 1________
addl %eax, -4(%ebp) ___add to m___________
    L2:
cmpl $0, 8(%ebp) ___compare n≥?__________
jg L3               ___if > loop___________
movl -4(%ebp), %eax ___put m in eax_______
leave               ___restore stack________
ret                 ___exit______________
```

Comment [CKR1]: Simply saying what the operations are, e.g., put 0 in eax, is not explaining.
Comment [CKR2]: Just saying setup is too vague.
Comment [CKR3]: Subtract 16 from stack is not explaining.
Comment [CKR4]: “n > 0” is wrong – it’s the jump that determines that.
Comment [CKR5]: Leave loop is wrong.
b) (6 pts) gcc -S -O2 produces this assembly code. Explain what each line does.

```assembly
pushl %ebp  ; save frame pointer
movl %esp, %ebp  ; make new frame pointer
movl 8(%ebp), %eax  ; put n in eax
testl %eax, %eax  ; compare n: 0
jg L5  ; if n > 0 go to L5
xorl %eax, %eax  ; set return value to 0
popl %ebp  ; restore frame ptr
ret  ; exit

L5:
jmp L5  ; loop forever
```

Comment [CKR6]: “n & n” s also fine, but not “n > 0”

c) (4 pts) Explain the optimizations made in version (b).

1. testl for comparing n to 0 - pure register operation
2. xorl for clearing m - pure register operation
3. no stack space allocated
4. ultra fast infinite loop with no useless code executed!
2. (6 pts) `strlen()` in C returns the length of a string. Its prototype is:

```c
typedef unsigned int size_t;
size_t strlen(const char * s);
```

A student who didn’t take EECS 213 wrote this code:

```c
int is_longer_str(const char *s1, const char *s2)
{
    return strlen(s1) – strlen(s2) > 0;
}
```

Give an example where this will do the wrong thing, explain why, and give a simple fix. Be specific.

If `s1` is shorter, subtracting 2 unsigned numbers gives a large unsigned number $> 0$.

Simplest fix: return `strlen(s1) > strlen(s2)`

3. (13 pts) Fill in the following table for an IEEE floating point representation with 1 sign bit S, 3 exponent bits and 3 fraction bits. M should be an integer or fraction, e.g., 0, 1, $\frac{1}{4}$. M, E and V should be base 10. $V = (-1)^S \times M \times 2^E$.

<table>
<thead>
<tr>
<th>Binary</th>
<th>M</th>
<th>E</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 000 000</td>
<td>0</td>
<td>-2</td>
<td>0</td>
</tr>
<tr>
<td>1 110 110</td>
<td>1 + $\frac{3}{4}$</td>
<td>3</td>
<td>-14</td>
</tr>
<tr>
<td>0 011 110</td>
<td>1 + $\frac{1}{4}$</td>
<td>0</td>
<td>1.75</td>
</tr>
<tr>
<td>0 000 011</td>
<td>$\frac{3}{8}$</td>
<td>-2</td>
<td>$\frac{3}{32}$ or 0.09375</td>
</tr>
<tr>
<td>0 111 000</td>
<td>—</td>
<td>—</td>
<td>$\infty$</td>
</tr>
</tbody>
</table>

Bias is $2^{(3-1)} - 1 = 3$

Comment [CR7]: Changing the return type of a built-in library function is not an option, nor does signed int make sense for strlen().

Comment [CR8]: Casting strlen() results to int doesn’t work. Consider `strlen(1) = 0` and `strlen(0) = large unsigned that is a negative integer.`
4. (19 pts) Fill in the table for a 5-bit two’s complement integer representation.

<table>
<thead>
<tr>
<th>Name</th>
<th>Decimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>14</td>
<td>01110</td>
</tr>
<tr>
<td>-</td>
<td>9</td>
<td>01001</td>
</tr>
<tr>
<td>-</td>
<td>.9</td>
<td>10111</td>
</tr>
<tr>
<td>-</td>
<td>-12</td>
<td>10100</td>
</tr>
<tr>
<td>Tmax</td>
<td>15</td>
<td>01111</td>
</tr>
<tr>
<td>Tmin</td>
<td>-16</td>
<td>10000</td>
</tr>
<tr>
<td>Tmin + Tmax</td>
<td>-1</td>
<td>11111</td>
</tr>
<tr>
<td>Tmin + 1</td>
<td>-15</td>
<td>10001</td>
</tr>
<tr>
<td>Tmax + 1</td>
<td>-16</td>
<td>10000</td>
</tr>
<tr>
<td>-Tmax</td>
<td>-15</td>
<td>10001</td>
</tr>
<tr>
<td>-Tmin</td>
<td>-16</td>
<td>10000</td>
</tr>
</tbody>
</table>
5. (15 pts) Given:

```c
typedef struct {  
    char c;  
    double p;  
    float d;  
    short s;  
    int *i;  
} Struct1;
```

A. Use vertical lines and labels to indicate clearly how data would be allocated for each element of a structure of type `Struct1` on an IA32 (x86) machine using Linux alignment rules. Crosshatch areas that are allocated but not used.

B. How many bytes are allocated for an object of type `Struct1`?

24 bytes

C. What alignment is required for an object of type `Struct1`? I.e., if an object must be aligned on an x-byte boundary, then say what x is.

4 byte

D. Do (A) again, with the fields of `Struct1` re-ordered to use the least number of bytes. Crosshatch areas that are allocated but not used.

Comment [CKR9]: some other orderings work. longest to shortest requires least thought.
6. (14 pts) Assume the variables \( a \) and \( b \) are signed integers. Assume two’s complement representation. Assume that \( \text{MAX\_INT} \) is the maximum integer, \( \text{MIN\_INT} \) is the minimum integer, and \( W \) is word length minus one, e.g., \( W = 31 \) for 32-bit integers. Next to each item on the left, write the letter of the code on the right that best matches it.

<table>
<thead>
<tr>
<th>Description</th>
<th>Choice</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a )</td>
<td>b</td>
<td>a. ( \sim a \</td>
</tr>
<tr>
<td>( a \ &amp; b )</td>
<td>d</td>
<td>b. ( ((a \ ^ b) \ &amp; \sim b) \</td>
</tr>
<tr>
<td>( a \ * 7 )</td>
<td>i</td>
<td>c. ( a \ &gt;&gt; \ [3] )</td>
</tr>
<tr>
<td>( a \ / 8 )</td>
<td>e</td>
<td>d. ( \sim ((a \ &gt;&gt; W) \ &lt;\ &lt; 1) )</td>
</tr>
<tr>
<td>( (a &lt; 0) \ ? \ 1 : -1 )</td>
<td>d</td>
<td>e. ( ((a &lt; 0) \ ? (a + 7) : a) \ &gt;&gt; 3 )</td>
</tr>
<tr>
<td>( a \ * 14 )</td>
<td>h</td>
<td>f. ( ((\sim a \ &amp; b) \</td>
</tr>
<tr>
<td>( a \ ^ b )</td>
<td>f</td>
<td>g. ( ((a \</td>
</tr>
<tr>
<td></td>
<td></td>
<td>h. ( (a \ &lt;\ &lt; 3) \ + (a \ &lt;\ &lt; 2) \ + (a \ &lt;\ &lt; 1) )</td>
</tr>
<tr>
<td></td>
<td>i</td>
<td>i. ( 1 + (a \ &lt;\ &lt; 3) + \sim a )</td>
</tr>
</tbody>
</table>

**Comment [CKR10]:** this is not the same as division if \( a \) is negative