October 27, 2010 Name ______________________________

EECS 213
Introduction to Computer Systems
Midterm Exam

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

```
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.

Comment [CKR1]: Simply saying what the operations are, e.g., put 0 in eax, is not explaining.

```
      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:0______________

      jg     L3                  ________if > loop________________

      movl   -4(%ebp), %eax      ________put m in eax_____________

      leave                      ________restore stack___________

      ret                        ____________exit____________
```

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.

```
    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:

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

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

```
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)

Comment [CKR7]: Changing the return type of a built-in library function is not an option, nor does signed int make sense for strlen().

Comment [CKR8]: Casting strlen() results to int doesn't work. Consider strlen(s1) = 0 and strlen(s2) = large unsigned that is a negative integer.

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, ¾. M, E and V should be base 10. $V = (-1)^S * M \cdot 2^E$

Binary	M	E	V
0 000 000	0	-2	0
1 110 110	1 + 3/4	3	-14
0 011 110	1 + ¾	0	1.75
0 000 011	3/8	-2	3/32 or 0.09375
0 111 000	—	—	∞

Bias is 2(3-1) - 1 = 3

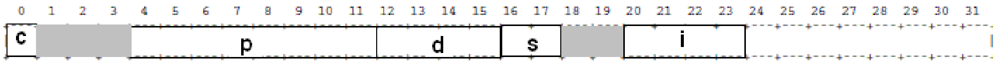
4. (19 pts) Fill in the table for a 5-bit two's complement integer representation.

Name	**Decimal**	**Binary**
—	14	0 1110
—	9	0 1001
—	-9	1 0111
—	12	0 1100
—	-12	1 0100
TMax	15	0 1111
TMin	-16	1 0000
Tmin + Tmax	-1	1 1111
TMin + 1	-15	1 0001
TMax + 1	-16	1 0000
-TMax	-15	1 0001
-TMin	-16	1 0000

5. (15 pts) Given:

```
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.



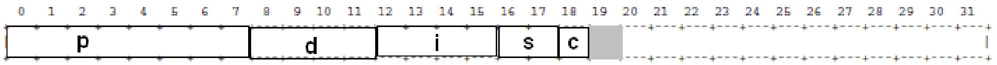
B. How many bytes are allocated for an object of type `Struct1`?

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.



20 bytes

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 MAX_INT is the maximum integer, 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.

Description	Choice	Code
`a`	b	**a.** `˜(˜a | (b ˆ (MIN_INT + MAX_INT)))`
`a & b`	a	**b.** `((a ˆ b) & ˜b) | (˜(a ˆ b) & b)`
`a * 7`	i	**c.** `a >> 3`
`a / 8`	e	**d.** `˜((a >> W) << 1)`
`(a < 0) ? 1 : -1`	d	**e.** `((a < 0) ? (a + 7) : a) >> 3`
`a * 14`	h	**f.** `((~a & b) | a) & ((~a & b) | ~b)`
`a ^ b`	f	**g.** `˜((a | (˜a + 1)) >> W) & 1`
		h. `(a << 3) + (a << 2) + (a << 1)`
		i. `1 + (a << 3) + ˜a`

Comment [CKR10]: this is not the same as division if a is negative