EECS 213: Midterm Exam

From a tour of computer systems to machine level representation of programs.

Spring 2007

Name:

Major/Department/School:

Some words of advice:

- *Read all the questions first.*
- *Start from the easiest one and leave the harder ones for the end.*
- *Approximate results are almost always a valid answer; for sure I do not need 5-decimal precision answers!*
- *This is an* Open Book *exam; you may use any book or notes you like.*
- *Write clearly; if I can't read it I can't grade it.*

Good luck!

Question	Points	Credited
1	10	
2	8	
3	10	
4	10	
5	12	
Total	50	

Problems ...

1. (*10 points*) Consider a **5-bit** two's complement representation. Fill in the empty boxes in the following table. Addition and subtraction should be performed based on the rules for 5-bit, two's complement arithmetic.

   Answers in bold.

Number	Decimal Representation	Binary Representation
Zero	0	**0 0000**
n/a	-2	**1 1110**
n/a	9	**0 1001**
n/a	-14	**1 0010**
n/a	**12**	0 1100
n/a	**-12**	1 0100
TMax	**15**	**0 1111**
TMin	**-16**	**1 0000**
TMin+TMin	**0**	**0 0000**
TMin+1	**-15**	**1 0001**
TMax+1	**-16 (Tmin)**	**1 0000**
−TMax	**-15**	**1 0001**
−TMin	**-16(Tmin)**	**1 0000**

2. (*8 points*) Consider the following 5-bit floating point representation based on the IEEE floating point format. There is a sign bit in the most significant bit. The next three bits are the exponent, with an exponent bias is 3. The last bit is the fraction. The rules are like those in the IEEE standard (normalized, denormalized, representation of 0, infinity, and NAN).

   As discussed in class, we consider the floating point format to encode numbers in a form:

$$V = (-1)^s \times M \times 2^E$$

   where M is the *significand* and E is the *exponent*.

   Fill in missing entries in the table below with the following instructions for each column:

   Description: Some unique property of this number, such as, "The largest denormalized value."

   Binary: The 5 bit representation.

   M**:** The value of the Mantissa written in decimal format.

   E**:** The integer value of the exponent.

   Value: The numeric value represented, written in decimal format.

   You need not fill in entries marked "—". For the arithmetic expressions, recall that the rule with IEEE format is to round to the number nearest the exact result. Use "round-to-even" rounding.

   Answers in bold.

Description	Binary	M	E	Value
Minus Zero	**10000**	**0**	**-2.0**	-0.0
Positive Infinity	**01110**	—	—	$+\infty$
Largest Number	01101	**1.5**	**3**	**12.0**
Smallest number > 0	**00001**	**0.5**	**-2**	**0.125**
One	**00110**	**1.0**	**0**	1.0
$4.0 - 0.75$	**01001**	**1.5**	**1**	**3.0**
$2.0 + 3.0$	**11010**	**1.0**	**2**	**4.0**

3. (*10 points*) A C function `looper` and the assembly code it compiles to on an IA-32 machine running Linux/GAS is shown below:

```
looper:
  pushl %ebp
  movl %esp,%ebp
  pushl %esi
  pushl %ebx
  movl 8(%ebp),%ebx
  movl 12(%ebp),%esi
  xorl %edx,%edx
  xorl %ecx,%ecx
  cmpl %ebx,%edx
  jge .L25
.L27:
  movl (%esi,%ecx,4),%eax
  cmpl %edx,%eax
  jle .L28
  movl %eax,%edx
.L28:
  incl %edx
  incl %ecx
  cmpl %ebx,%ecx
  jl .L27
.L25:
  movl %edx,%eax
  popl %ebx
  popl %esi
  movl %ebp,%esp
  popl %ebp
  ret
```

Answer between * **

```
int looper(int n, int *a) {
  int i;
  int x = _____________;        *** = 0; ***
  for(i = _________; _________;  *** 0; i < n ***
      i++) {
    if (___________________)     *** a[i] > x or !(a[i] <= x) ***
       x = _________________;    *** x = a[i] ***
    ___________________;        *** x++ ***
  }
  return x;
}
```

Based on the assembly code, fill in the blanks in the C source code.

Notes:

- You may only use the C variable names `n`, `a`, `i` and `x`, not register names.
- Use array notation in showing accesses or updates to elements of `a`.

4. (*10 points*) Consider the following incomplete definition of a C struct along with the incomplete code for a function `func` given below.
Answer between ***

```
typedef struct node {

  _______________ x; *** double ***

  _______________ y; *** unsigned short ***

  struct node *next;

  struct node *prev;

} node_t;

node_t n;

void func() {

  node_t *m;

  m = _____________________; *** n.next->prev ***

  m->y /= 16;

  return;
}
```

When this C code was compiled on an IA-32 machine running Linux, the following assembly code was generated for function `func`.

```
func:
  pushl %ebp
  movl n+12,%eax
  movl 16(%eax),%eax
  movl %esp,%ebp
  movl %ebp,%esp
  shrw $4,8(%eax)
  popl %ebp
  ret
```

Given these code fragments, fill in the blanks in the C code given above. Note that there is a unique answer.

The types must be chosen from the following table, assuming the sizes and alignment given.

Type	Size (bytes)	Alignment (bytes)
char	1	1
short	2	2
unsigned short	2	2
int	4	4
unsigned int	4	4
double	8	4

5. (*12 points*) The following problem concerns the following, low-quality code:

```
void foo(int x)
{
  int a[3];
  char buf[4];
  a[0] = 0xF0F1F2F3;
  a[1] = x;
  gets(buf);
  printf("a[0] = 0x%x, a[1] = 0x%x, buf = %s\n", a[0], a[1], buf);
}
```

In a program containing this code, procedure `foo` has the following disassembled form on an IA32 machine:

```
080485d0 <foo>:
80485d0: 55              pushl  %ebp
80485d1: 89 e5           movl   %esp,%ebp
80485d3: 83 ec 10        subl   $0x10,%esp
80485d6: 53              pushl  %ebx
80485d7: 8b 45 08        movl   0x8(%ebp),%eax
80485da: c7 45 f4 f3 f2  movl   $0xf0f1f2f3,0xfffffff4(%ebp)
80485df: f1 f0
80485e1: 89 45 f8        movl   %eax,0xfffffff8(%ebp)
80485e4: 8d 5d f0        leal   0xfffffff0(%ebp),%ebx
80485e7: 53              pushl  %ebx
80485e8: e8 b7 fe ff ff  call   80484a4 <_init+0x54>  # gets
80485ed: 53              pushl  %ebx
80485ee: 8b 45 f8        movl   0xfffffff8(%ebp),%eax
80485f1: 50              pushl  %eax
80485f2: 8b 45 f4        movl   0xfffffff4(%ebp),%eax
80485f5: 50              pushl  %eax
80485f6: 68 ec 90 04 08  pushl  $0x80490ec
80485fb: e8 94 fe ff ff  call   8048494 <_init+0x44>  # printf
8048600: 8b 5d ec        movl   0xffffffec(%ebp),%ebx
8048603: 89 ec           movl   %ebp,%esp
8048605: 5d              popl   %ebp
8048606: c3              ret
8048607: 90              nop
```

For the following questions, recall that:

- `gets` is a standard C library routine.
- IA32 machines are little-endian.
- C strings are null-terminated (i.e., terminated by a character with value 0x00).
- Characters '0' through '9' have ASCII codes 0x30 through 0x39.

Fill in the following table indicating where on the stack the following program values are located. Express these as decimal offsets (positive or negative) relative to register %ebp:

Answers in bold.

Program Value	Decimal Offset
`a`	**-12**
`a[2]`	**-4**
`x`	**+8**
`buf`	**-16**
`buf[3]`	**-13**
Saved value of register %ebx	**-20**