Full Name:_____

Andrew ID (print clearly!):_____

15-213/18-213, Spring 2012

Exam 1

Tuesday, March 6, 2012

Instructions:

- Make sure that your exam is not missing any sheets, then write your full name and Andrew ID on the front.
- This exam is closed book, closed notes. You may not use any electronic devices.
- Write your answers in the space provided below the problem. If you make a mess, clearly indicate your final answer.
- The exam has a maximum score of 100 points.
- The problems are of varying difficulty. The point value of each problem is indicated. Good luck!

Problem	Your Score	Possible Points
1		15
2		16
3		15
4		16
5		16
6		5
7		17
Total		100

Problem 1. (15 points):

Bits and Bytes

A. Consider the C function below:

```
int func( int x )
{
  int y;
  y=(x<<31)>>31;
  return y;
}
```

When "`func(7)`" is called (i.e. `x=7`) on IA32, what is the return value? _____

B. The expression $x * x \geq 0$ holds uniformly for

(a) both signed and unsigned integers
(b) signed integers, but not for unsigned integers
(c) unsigned integers, but not for signed integers
(d) neither signed nor unsigned integers

_____

C. What is the evaluation result of expression 1110_2 ^ 1010_2?

(a) 1111_2
(b) 1010_2
(c) 0100_2
(d) 0110_2

_____

D. Assume that you are working on a machine with **8-bit `int`s** and arithmetic right shifts. Further assume that variable `x` is a signed integer represented in two's complement. Match each of the descriptions on the left with 0, 1, or more snippets of code on the right. Write the letter of each matching snippet in the blank under the description. Some code snippets may not match any of the descriptions.

(1) `13 * x`

_____

(2) Absolute value of `x`.

_____

(3) Round `x` down to nearest power of 2.

_____

(4) Round `x` to a multiple of 16

_____

(5) `x < 0`

_____

(6) Swap most significant and least significant byte of `x`.

_____

(a) `x & (x - 1)`

(b) `-((x | MAX_INT) >> 7)`

(c) `((~x & MIN_INT) == 0`

(d) `(x << 4) | ((x >> 4) & 0x0F)`

(e) `(x >> 4) << 4`

(f) `(x << 3) + (x << 2) + x`

(g) `(0x80 >> 4) & x`

(h) `x * (1 | (x >> 7))`

Problem 2. (16 points):

Floating point

Consider an 8-bit floating point representation based on the IEEE floating point format that has:

- 1 sign bit,
- 4 exponent bits (hence the $\mathrm{Bias} = 7$), and
- 3 fraction bits.

Numeric values are encoded as a value of the form $V = (-1)^S \times M \times 2^E$, where S is the sign bit, E is the exponent after biasing, and M is the significand value. The fraction bits encode the significand value M using either a denormalized (exponent field 0) or a normalized representation (exponent field nonzero). The exponent E is given by $E = 1 - \mathrm{Bias}$ for denormalized values and $E = e - \mathrm{Bias}$ for normalized values, where e is the value of the exponent field interpreted as an unsigned number.

A. Below, you are given some decimal values, and your task it to encode them in floating point format. If rounding is necessary, you should use *round-to-even*. In addition, you should give the rounded value of the encoded floating point number. Give these as whole numbers (e.g., 17) or as fractions in reduced form (e.g., $3/4$).

Value	Floating Point Bits	Rounded value
$9/32$	`0 0101 001`	$9/32$
40		
$15/2$		
$1/128$		

B. Given the 8-bit floating point representation described above, how many different *rational* numbers can be represented? (Assume that 0 and -0 count as just one number.)

_____

C. The relative error between a real number $x \neq 0$, and its closest IEEE floating point approximation, $\hat{x}$ is $\left|\frac{x-\hat{x}}{x}\right|$. Given the IEEE-like 8-bit floating representation described above (with 1 sign bit, 4 exponent bits and 3 fraction bits), what is the maximum relative error for the range of real numbers in the range $x \in (0, 8)$? (Note that x is strictly greater than 0 and strictly less than 8.)

(a) $1/16$
(b) $1/8$
(c) $1/4$
(d) $1/2$
(e) 1

_____

Problem 3. (15 points):

x86-64 Assembly Code

For each of the following three C code functions (on the left), give the letter of the corresponding block of x86-64 assembly code (on the right).

```
long int func1(long int a)
{
  long int i;

  if (a > 100L)
    return a * 5L;

  for (i = 0L; i < 10L; i++) {
    a = a * 5L;
  }
  return a;
}
```

Assembly that matches `func1`: _____

```
long int func2(long int a)
{
  long int i = 0L;

  while (i < 10L) {
    a = a * 5L;
    i++;
  }
  if (a > 100L)
    return a;
  else return a*5L;
}
```

Assembly that matches `func2`: _____

```
long int func3(long int a)
{
  long int i = 0L;

  do {
    if(a > 100L)
      return a;

    a = a * 5L;
    i++;
  } while(i < 10L);
  return a;
}
```

Assembly that matches `func3`: _____

(A)

```
 0: 48 8d 04 bf     lea    (%rdi,%rdi,4),%rax
 4: 48 83 ff 63     cmp    $0x63,%rdi
 8: 7e 16           jle    0x20
 a: 48 0f af ff     imul   %rdi,%rdi
 e: 48 83 ff 0a     cmp    $0xa,%rdi
12: b8 64 00 00 00  mov    $0x64,%eax
17: ba 00 00 00 00  mov    $0x0,%edx
1c: 48 0f 4c c2     cmovl  %rdx,%rax
20: f3 c3           repz retq
```

(B)

```
 0: b8 00 00 00 00  mov    $0x0,%eax
 5: 48 8d 3c bf     lea    (%rdi,%rdi,4),%rdi
 9: 48 83 c0 01     add    $0x1,%rax
 d: 48 83 f8 0a     cmp    $0xa,%rax
11: 75 f2           jne    0x5
13: 48 83 ff 64     cmp    $0x64,%rdi
17: 48 8d 04 bf     lea    (%rdi,%rdi,4),%rax
1b: 48 0f 4e f8     cmovle %rax,%rdi
1f: 48 89 f8        mov    %rdi,%rax
22: c3              retq
```

(C)

```
 0: 48 89 f8        mov    %rdi,%rax
 3: ba 00 00 00 00  mov    $0x0,%edx
 8: 48 83 ff 64     cmp    $0x64,%rdi
 c: 7e 05           jle    0x13
 e: 48 8d 04 bf     lea    (%rdi,%rdi,4),%rax
12: c3              retq
13: 48 8d 04 80     lea    (%rax,%rax,4),%rax
17: 48 83 c2 01     add    $0x1,%rdx
1b: 48 83 fa 0a     cmp    $0xa,%rdx
1f: 75 f2           jne    0x13
21: f3 c3           repz retq
```

(D)

```
 0: 48 89 f8        mov    %rdi,%rax
 3: 48 83 ff 64     cmp    $0x64,%rdi
 7: 7f 1f           jg     0x28
 9: 48 8d 04 bf     lea    (%rdi,%rdi,4),%rax
 d: ba 01 00 00 00  mov    $0x1,%edx
12: eb 0e           jmp    0x22
14: 48 8d 04 80     lea    (%rax,%rax,4),%rax
18: 48 83 c2 01     add    $0x1,%rdx
1c: 48 83 fa 0a     cmp    $0xa,%rdx
20: 74 06           je     0x28
22: 48 83 f8 64     cmp    $0x64,%rax
26: 7e ec           jle    0x14
28: f3 c3           repz retq
```

Problem 4. (16 points):

Stack discipline.

A. Caller-save and callee-save register conventions change when making recursive function calls:

True or False: _____

B. Putting a special "canary" value on the stack just beyond a buffer can be used to detect overflows:

(a) Always
(b) Never
(c) Sometimes

_____

C. On x86-64 machines we tend to see more pushing and popping from the stack relative to IA32:

True or False: _____

D. Consider a C function with the following declaration:

```
void spawn_larva(int a, int b, int c, int d);
```

Assuming `spawn_larva` has been compiled for an x86 IA32 machine with 4-byte `int`s, what would be the address of the argument `b` in terms of `%ebp` in the stack frame of `spawn_larva`?

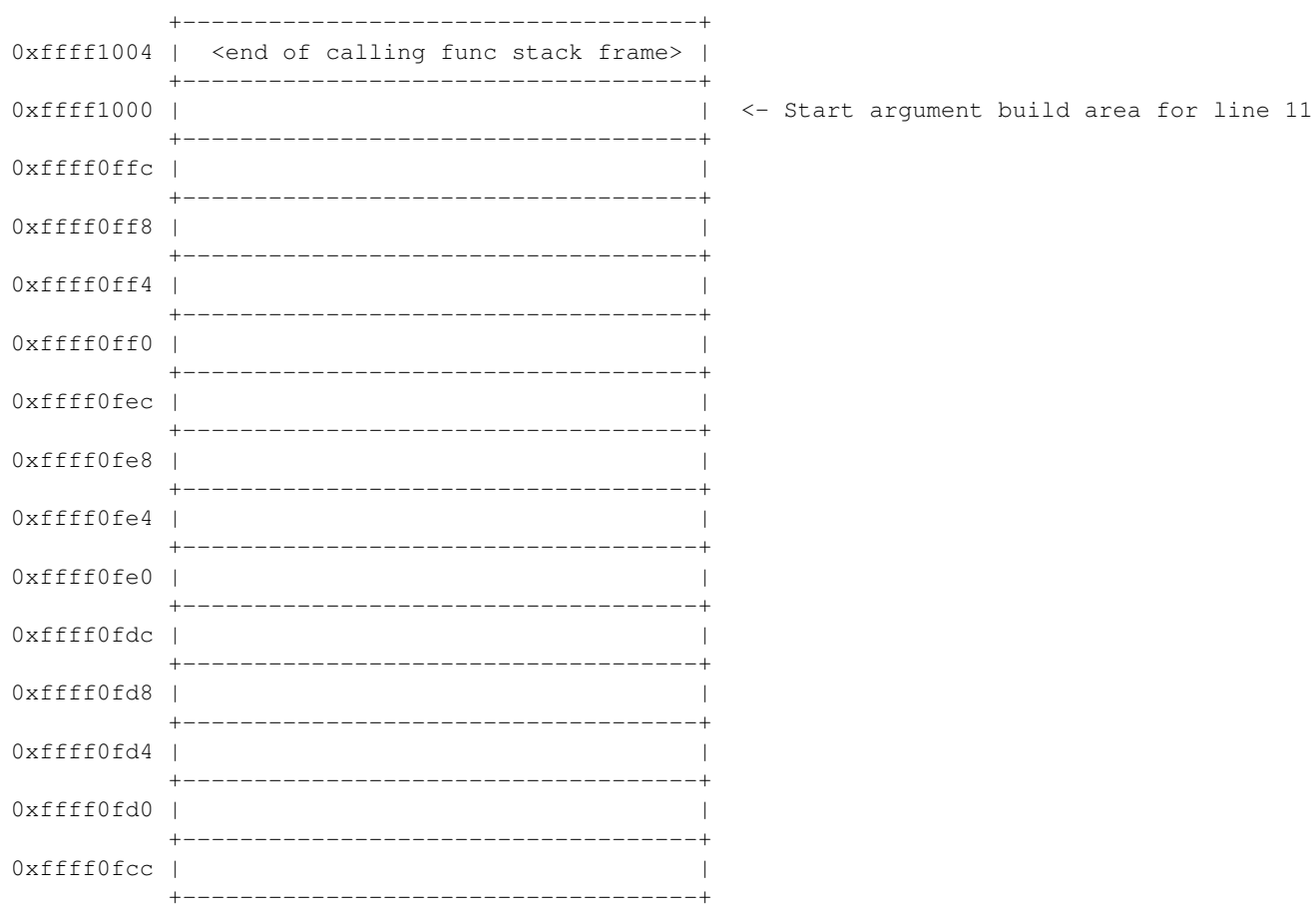
(a) `%ebp` + 8
(b) `%ebp` + 12
(c) `%ebp` + 16
(d) `%ebp` + 20

_____

E. Given the following function call, fill in the stack frame diagram with:

- Any function arguments (labeled by variable name: "x" if variable is `int x`)
- Return addresses (marked as "Return Address")
- The smallest location on the stack pointed to by `%esp` and `%ebp`

```
                                  00000000 <foo>:
 0:                                  0: 55                    push   %ebp
 1:  int foo(int n)                  1: 89 e5                 mov    %esp,%ebp
 2:  {                               3: 53                    push   %ebx
 3:    if( n<=1 ) return 1;          4: 83 ec 14              sub    $0x10,%esp
 4:    else                          7: 8b 5d 08              mov    0x8(%ebp),%ebx
 5:    n=n*foo(n-1);                 a: b8 01 00 00 00        mov    $0x1,%eax
 6:    return n;                     f: 83 fb 01              cmp    $0x1,%ebx
 7:  }                              12: 7e 0e                 jle    22 <foo+0x22>
 8:                                 14: 8d 43 ff              lea    -0x1(%ebx),%eax
 9:  ...                            17: 89 04 24              mov    %eax,(%esp)
10:                                 1a: e8 fc ff ff ff        call   <foo>
11:  x=foo(2);                      1f: 0f af c3              imul   %ebx,%eax
                                    22: 83 c4 14              add    $0x10,%esp
                                    25: 5b                    pop    %ebx
                                    26: 5d                    pop    %ebp
                                    27: c3                    ret
```



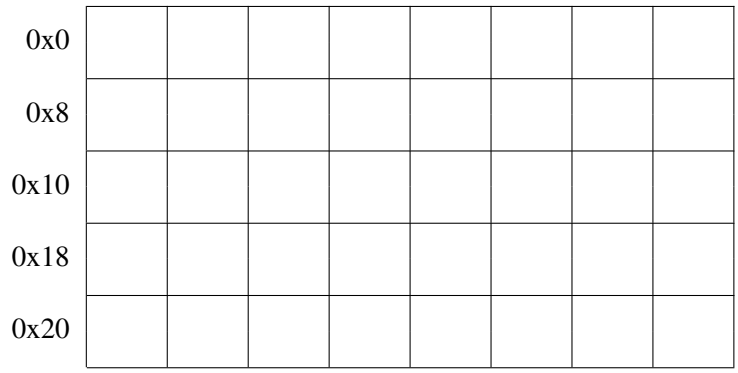
Smallest location pointed to by `%esp`: _____ and `%ebp`: _____
after calling foo();

Problem 5. (16 points):

Structure layout.

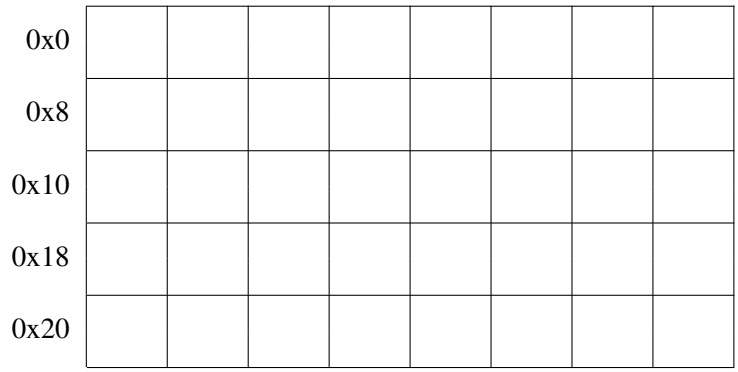
```
struct a {
  float* f;
  char c;
  int x;
  char z[4];
  double d;
  short s;
};

struct b {
  struct a a1;
  int y;
  struct a a2;
};
```

A. Show the layout of `struct a` in memory and shade in any bytes used for padding on a Shark Linux machine running in **IA32** mode (i.e. with **32-bit** addresses).



How many total bytes does `struct a` use in this case? _____

B. Now show the layout of `struct a` in memory (and shade in any bytes used for padding) on a Shark Linux machine running in **x86-64** mode (i.e. with **64-bit** addresses).



How many total bytes does `struct a` use in this case? _____

C. How many bytes does `struct b` use on a Shark Linux machine in IA32 mode (i.e. with 32-bit addresses)?

D. How many bytes does `struct b` use on a Shark Linux machine in x86-64 mode (i.e. with 64-bit addresses)?

Problem 6. (5 points):

Memory Hiearchy

A. What type of non-volatile memory would most likely store firmware like a computer's BIOS?

(a) RAM
(b) EEPROM
(c) Hard Disk
(d) Tape Drive

_____

B. Rank the following rates that you would expect from an SSD (such as Flash memory) from fastest (1) to slowest (3):

- Random read throughput: _____
- Sequential read throughput: _____
- Random write throughput: _____

Problem 7. (17 points):

Caches

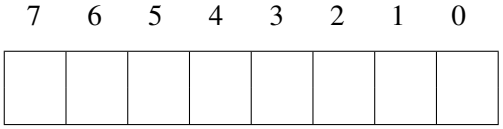
Consider a computer with an **8-bit address space** and a **direct-mapped 32-byte** data cache with **4-byte cache blocks**.

A. The boxes below represent the bit-format of an address. In each box, indicate which field that bit represents (it is possible that a field does not exist) by labeling them as follows:

BO: Block Offset

SI: Set Index

CT: Cache Tag



B. The table below shows a trace of load addresses accessed in the data cache. Assume the cache is initially empty. For each row in the table, please complete the two rightmost columns, indicating (i) the *set number* (in decimal notation) for that particular load, and (ii) whether that loads *hits* (H) or *misses* (M) in the cache (circle either "H" or "M" accordingly). Also, please indicate the total number of cache hits in the blank below the table.

Load No.	Hex Address	Binary Address	Set Number? (in Decimal)	Hit or Miss? (Circle one)
1	c7	1100 0111		H M
2	55	0101 0101		H M
3	1a	0001 1010		H M
4	c5	1100 0101		H M
5	e6	1110 0110		H M
6	56	0101 0110		H M
7	77	0111 0111		H M
8	28	0010 1000		H M
9	75	0111 0101		H M
10	94	1001 0100		H M

How many cache hits were there in total? _____

C. For the trace of load addresses shown in Part B, below is a list of possible final states for the cache, showing the hex value of the tag for each cache block in each set. Assume that initially all cache blocks are invalid (represented by X).

(a)

Set:	0	1	2	3	4	5	6	7
Tag:	X	7	1	X	X	4	1	X

(b)

Set:	0	1	2	3	4	5	6	7
Tag:	1	7	X	X	1	4	0	X

(c)

Set:	0	1	2	3	4	5	6	7
Tag:	X	1	1	X	0	2	0	X

(d)

Set:	0	1	2	3	4	5	6	7
Tag:	X	1	7	X	X	4	4	0

(e)

Set:	0	1	2	3	4	5	6	7
Tag:	X	1	7	X	4	4	0	X

(f)

Set:	0	1	2	3	4	5	6	7
Tag:	X	7	1	X	X	4	0	X

(g)

Set:	0	1	2	3	4	5	6	7
Tag:	7	X	1	0	4	4	0	X

Which of the choices above is the correct final state of the cache? _____

15-213/18-213 Midterm Exam Notes Sheet Spring 2011

Jumps

Jump	Condition
`jmp`	1
`je`	ZF
`jne`	~ZF
`js`	SF
`jns`	~SF
`jg`	~(SF^OF)&~ZF
`jge`	~(SF^OF)
`jl`	(SF^OF)
`jle`	(SF^OF)|ZF
`ja`	~CF&~ZF
`jb`	CF

Arithmetic Operations

Format		Computation
`addl`	*Src,Dest*	Dest = Dest + Src
`subl`	*Src,Dest*	Dest = Dest - Src
`imull`	*Src,Dest*	Dest = Dest * Src
`sall`	*Src,Dest*	Dest = Dest << Src
`sarl`	*Src,Dest*	Dest = Dest >> Src
`shrl`	*Src,Dest*	Dest = Dest >> Src
`xorl`	*Src,Dest*	Dest = Dest ^ Src
`adnl`	*Src,Dest*	Dest = Dest & Src
`orl`	*Src,Dest*	Dest = Dest | Src

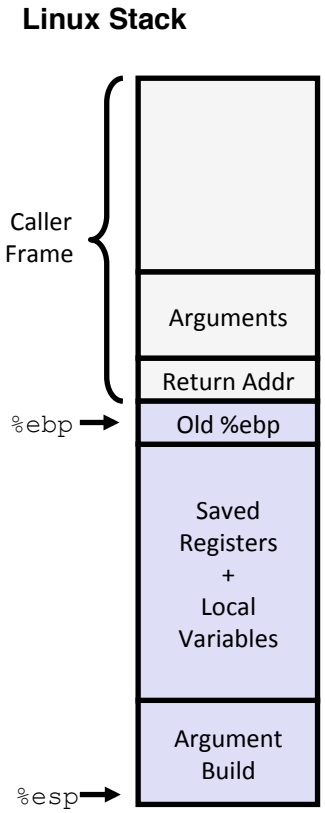
Memory Operations

Format	Computation
`(Rb, Ri)`	Mem[Reg[Rb]+Reg[Ri]]
`D(Rb,Ri)`	Mem[Reb[Rb]+Reg[Ri]+D]
`(Rb,Ri,S)`	Mem[Reg[Rb]+S*Reg[Ri]]

Registers

63	31	15	8 7	0	
`%rax`	`%eax`	`%ax`	`%ah`	`%al`	Return value
`%rbx`	`%ebx`	`%bx`	`%bh`	`%bl`	Callee saved
`%rcx`	`%ecx`	`%cx`	`%ch`	`%cl`	Argument #4
`%rdx`	`%edx`	`%dx`	`%dh`	`%dl`	Argument #3
`%rsi`	`%esi`	`%si`		`%sil`	Argument #2
`%rdi`	`%edi`	`%di`		`%dil`	Argument #1
`%rbp`	`%ebp`	`%bp`		`%bpl`	Callee saved
`%rsp`	`%esp`	`%sp`		`%spl`	Stack Pointer
`%r8`	`%r8d`	`%r8w`		`%r8b`	Argument #5
`%r9`	`%r9d`	`%r9w`		`%r9b`	Argument #6
`%r10`	`%r10d`	`%r10w`		`%r10b`	Reserved
`%r11`	`%r11d`	`%r11w`		`%r11b`	Used for linking
`%r12`	`%r12d`	`%r12w`		`%r12b`	Callee saved
`%r13`	`%r13d`	`%r13w`		`%r13b`	Callee saved
`%r14`	`%r14d`	`%r14w`		`%r14b`	Callee saved
`%r15`	`%r15d`	`%r15w`		`%r15b`	Callee saved

Linux Stack



Specific Cases of Alignment (IA32)

1 byte: char, …

no restrictions on address

2 bytes: short, …

lowest 1 bit of address must be 0_2

4 bytes: int, float, char *, …

lowest 2 bits of address must be 00_2

8 bytes: double, …

Windows (and most other OS's & instruction sets):

lowest 3 bits of address must be 000_2

Linux:

lowest 2 bits of address must be 00_2

i.e., treated the same as a 4-byte primitive data type

12 bytes: long double

Windows, Linux:

lowest 2 bits of address must be 00_2

i.e., treated the same as a 4-byte primitive data type

C Data Type	Intel IA32	x86-64
char	1	1
short	2	2
int	4	4
long	4	8
long long	8	8
float	4	4
double	8	8
long double	10/12	10/16
pointer	4	8

Specific Cases of Alignment (x86-64)

1 byte: char, …

no restrictions on address

2 bytes: short, …

lowest 1 bit of address must be 0_2

4 bytes: int, float, …

lowest 2 bits of address must be 00_2

8 bytes: double, char *, …

Windows & Linux:

lowest 3 bits of address must be 000_2

16 bytes: long double

Linux:

lowest 3 bits of address must be 000_2

i.e., treated the same as a 8-byte primitive data type

Byte Ordering

4-byte variable 0x01234567 at 0x100

Big Endian

Least significant byte has highest address

0x100	0x101	0x102	0x103
01	23	45	67

Little Endian

Least significant byte has lowest address

0x100	0x101	0x102	0x103
67	45	23	01

Floating Point

Bias = $2^{k-1} - 1$