Andrew login ID: _____

Full Name: _____

CS 15-213, Fall 2001

Exam 1

October 9, 2001

Instructions:

- Make sure that your exam is not missing any sheets, then write your full name and Andrew login ID on the front.
- 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 64 points.
- The problems are of varying difficulty. The point value of each problem is indicated. Pile up the easy points quickly and then come back to the harder problems.
- This exam is OPEN BOOK. You may use any books or notes you like. You may use a calculator, but no laptops or other wireless devices. Good luck!

1 (09):	
2 (12):	
3 (03):	
4 (03):	
5 (08):	
6 (08):	
7 (09):	
8 (12):	
TOTAL (64):	

Problem 1. (9 points):

Assume we are running code on a 6-bit machine using two's complement arithmetic for signed integers. A "short" integer is encoded using 3 bits. Fill in the empty boxes in the table below. The following definitions are used in the table:

```
short sy = -3;
int y = sy;
int x = -17;
unsigned ux = x;
```

Note: You need not fill in entries marked with "–".

Expression	Decimal Representation	Binary Representation
Zero	0	
–	-6	
–		01 0010
ux		
y		
$x >> 1$		
TMax		
$-$TMin		
TMin + TMin		

Problem 2. (12 points):

Consider the following 8-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 $2^{3-1} - 1 = 3$.
- The last 4 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 is the biased exponent.

The rules are like those in the IEEE standard(normalized, denormalized, representation of 0, infinity, and NAN). FILL in the table below. Here are the instructions for each field:

- **Binary:** The 8 bit binary representation.
- **M:** The value of the significand. This should be a number of the form x or $\frac{x}{y}$, where x is an integer, and y is an integral power of 2. Examples include 0, $\frac{3}{4}$.
- **E:** The integer value of the exponent.
- **Value:**The numeric value represented.

  Note: you need not fill in entries marked with "—".

Description	Binary	M	E	Value
Minus zero				-0.0
—	`0 100 0101`			
Smallest denormalized (negative)				
Largest normalized (positive)				
One				1.0
—				5.5
Positive infinity		—	—	$+\infty$

Problem 3. (3 points):

Consider the following C functions and assembly code:

```
int fun7(int a)
{
    return a * 30;
}

int fun8(int a)
{
    return a * 34;
}

int fun9(int a)
{
    return a * 18;
}
```

```
pushl %ebp
movl %esp,%ebp
movl 8(%ebp),%eax
sall $4,%eax
addl 8(%ebp),%eax
addl %eax,%eax
movl %ebp,%esp
popl %ebp
ret
```

Which of the functions compiled into the assembly code shown?

Problem 4. (3 points):

Consider the following C functions and assembly code:

```
int fun4(int *ap, int *bp)
{
    int a = *ap;
    int b = *bp;
    return a+b;
}

int fun5(int *ap, int *bp)
{
    int b = *bp;
    *bp += *ap;
    return b;
}

int fun6(int *ap, int *bp)
{
    int a = *ap;
    *bp += *ap;
    return a;
}
```

```
pushl %ebp
movl %esp,%ebp
movl 8(%ebp),%edx
movl 12(%ebp),%eax
movl %ebp,%esp
movl (%edx),%edx
addl %edx,(%eax)
movl %edx,%eax
popl %ebp
ret
```

Which of the functions compiled into the assembly code shown?

Problem 5. (8 points):

Consider the source code below, where `M` and `N` are constants declared with `#define`.

```
int array1[M][N];
int array2[N][M];

int copy(int i, int j)
{
    array1[i][j] = array2[j][i];
}
```

Suppose the above code generates the following assembly code:

```
copy:
  pushl %ebp
  movl %esp,%ebp
  pushl %ebx
  movl 8(%ebp),%ecx
  movl 12(%ebp),%ebx
  leal (%ecx,%ecx,8),%edx
  sall $2,%edx
  movl %ebx,%eax
  sall $4,%eax
  subl %ebx,%eax
  sall $2,%eax
  movl array2(%eax,%ecx,4),%eax
  movl %eax,array1(%edx,%ebx,4)
  popl %ebx
  movl %ebp,%esp
  popl %ebp
  ret
```

What are the values of `M` and `N`?

`M` =

`N` =

Problem 6. (8 points):

Consider the following assembly representation of a function `foo` containing a `for` loop:

```
foo:
  pushl %ebp
  movl %esp,%ebp
  pushl %ebx
  movl 8(%ebp),%ebx
  leal 2(%ebx),%edx
  xorl %ecx,%ecx
  cmpl %ebx,%ecx
  jge .L4
.L6:
  leal 5(%ecx,%edx),%edx
  leal 3(%ecx),%eax
  imull %eax,%edx
  incl %ecx
  cmpl %ebx,%ecx
  jl .L6
.L4:
  movl %edx,%eax
  popl %ebx
  movl %ebp,%esp
  popl %ebp
  ret
```

Fill in the blanks to provide the functionality of the loop:

```
int foo(int a)
{
    int i;
    int result = _____________;

    for( ________; ________; i++ ) {

        ___________________;

        ___________________;

    }
    return result;
}
```

Problem 7. (9 points):

Consider the following C declarations:

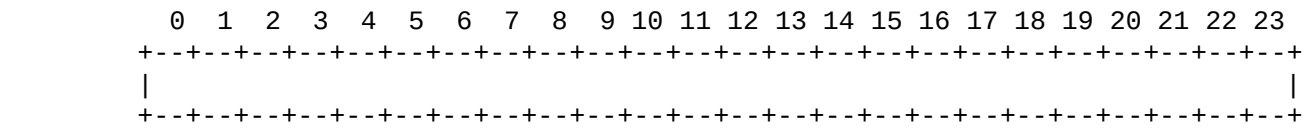
```
typedef struct {
    short code;
    long start;
    char raw[3];
    double data;
} OldSensorData;

typedef struct {
    short code;
    short start;
    char raw[5];
    short sense;
    short ext;
    double data;
} NewSensorData;
```

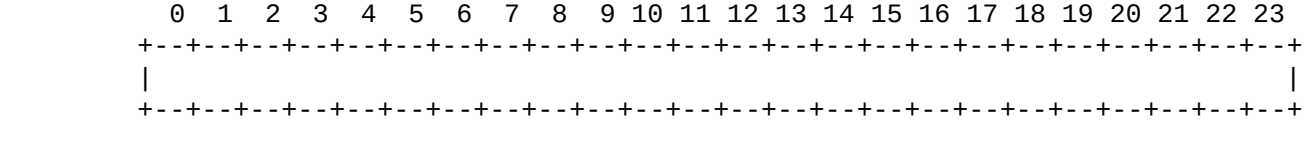
A. Using the templates below (allowing a maximum of 24 bytes), indicate the allocation of data for structs of type `OldSensorData` `NewSensorData`. Mark off and label the areas for each individual element (arrays may be labeled as a single element). **Cross hatch the parts that are allocated, but not used (to satisfy alignment).**

Assume the Linux alignment rules discussed in class. **Clearly indicate the right hand boundary of the data structure with a vertical line.**

`OldSensorData:`



`NewSensorData:`



B. Now consider the following C code fragment:

```
void foo(OldSensorData *oldData)
{
    NewSensorData *newData;

    /* this zeros out all the space allocated for oldData */
    bzero((void *)oldData, sizeof(oldData));

    oldData->code = 0x104f;
    oldData->start = 0x80501ab8;
    oldData->raw[0] = 0xe1;
    oldData->raw[1] = 0xe2;
    oldData->raw[2] = 0x8f;
    oldData->raw[-5] = 0xff;
    oldData->data = 1.5;

    newData = (NewSensorData *) oldData;

    ...
```

Once this code has run, we begin to access the elements of newData. Below, give the value of each element of `newData` that is listed. Assume that this code is run on a Little-Endian machine such as a Linux/x86 machine. You must give your answer in hexadecimal format. **Be careful about byte ordering!**.

(a) `newData->start  = 0x`_____

(b) `newData->raw[0] = 0x`_____

(c) `newData->raw[2] = 0x`_____

(d) `newData->raw[4] = 0x`_____

(e) `newData->sense  = 0x`_____

The next problem concerns the following C code. This program reads a string on standard input and prints an integer in hexadecimal format based on the input string it read.

```
#include <stdio.h>

/* Read a string from stdin into buf */
int evil_read_string()
{
    int buf[2];

    scanf("%s",buf);
    return buf[1];
}

int main()
{
    printf("0x%x\n", evil_read_string());
}
```

Here is the corresponding machine code on a Linux/x86 machine:

```
08048414 <evil_read_string>:
 8048414:  55                      push   %ebp
 8048415:  89 e5                   mov    %esp,%ebp
 8048417:  83 ec 14                sub    $0x14,%esp
 804841a:  53                      push   %ebx
 804841b:  83 c4 f8                add    $0xfffffff8,%esp
 804841e:  8d 5d f8                lea    0xfffffff8(%ebp),%ebx
 8048421:  53                      push   %ebx                    address arg for scanf
 8048422:  68 b8 84 04 08          push   $0x80484b8              format string for scanf
 8048427:  e8 e0 fe ff ff          call   804830c <_init+0x50>    call scanf
 804842c:  8b 43 04                mov    0x4(%ebx),%eax
 804842f:  8b 5d e8                mov    0xffffffe8(%ebp),%ebx
 8048432:  89 ec                   mov    %ebp,%esp
 8048434:  5d                      pop    %ebp
 8048435:  c3                      ret

08048438 <main>:
 8048438:  55                      push   %ebp
 8048439:  89 e5                   mov    %esp,%ebp
 804843b:  83 ec 08                sub    $0x8,%esp
 804843e:  83 c4 f8                add    $0xfffffff8,%esp
 8048441:  e8 ce ff ff ff          call   8048414 <evil_read_string>
 8048446:  50                      push   %eax                    integer arg for printf
 8048447:  68 bb 84 04 08          push   $0x80484bb              format string for printf
 804844c:  e8 eb fe ff ff          call   804833c <_init+0x80>    call printf
 8048451:  89 ec                   mov    %ebp,%esp
 8048453:  5d                      pop    %ebp
 8048454:  c3                      ret
```

Problem 8. (12 points):

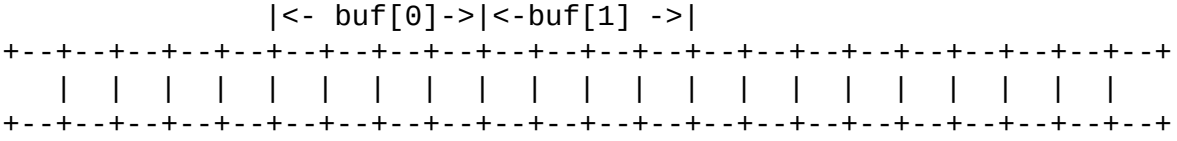
This problem tests your understanding of the stack discipline and byte ordering. Here are some notes to help you work the problem:

- `scanf("%s", buf)` reads an input string from the standard input stream (stdin) and stores it at address `buf` (including the terminating '`\0`' character). It does **not** check the size of the destination buffer.
- `printf("0x%x", i)` prints the integer i in hexadecimal format preceded by "`0x`".
- Recall that Linux/x86 machines are Little Endian.
- You will need to know the hex values of the following characters:

Character	Hex value	Character	Hex value
'd'	0x64	'v'	0x76
'r'	0x72	'i'	0x69
'.'	0x2e	'l'	0x6c
'e'	0x65	'\0'	0x00
		's'	0x73

A. Suppose we run this program on a Linux/x86 machine, and give it the string "`dr.evil`" as input on stdin.

Here is a template for the stack, showing the locations of `buf[0]` and `buf[1]`. Fill in the value of `buf[1]` (in hexadecimal) and indicate where `ebp` points just **after** `scanf` returns to `evil_read_string`.



What is the 4-byte integer (in hex) printed by the `printf` inside main?

0x_____

B. Suppose now we give it the input "`dr.evil.lives`" (again on a Linux/x86 machine).

(a) List the contents of the following memory locations just **after** `scanf` returns to `evil_read_string`. Each answer should be an unsigned 4-byte integer expressed as 8 hex digits.

`buf[0] = 0x__________`

`buf[3] = 0x__________`

(b) Immediately **before** the `ret` instruction at address `0x08048435` executes, what is the value of the frame pointer register `%ebp`?

`%ebp  = 0x__________`

You can use the following template of the stack as *scratch space*. *Note:* this does **not** have to be filled out to receive full credit.