Andrew login ID:______________________________________

Full Name:______________________________________

CS 15-213, Fall 2002

Exam 1

October 8, 2002

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 66 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 (06):	
2 (16):	
3 (08):	
4 (12):	
5 (06):	
6 (12):	
7 (06):	
TOTAL (66):	

Problem 1. (6 points):

Assume we are running code on a 5-bit machine using two's complement arithmetic for signed integers. Fill in the empty boxes in the table below. The following definitions are used in the table:

```
int y = -9;
unsigned z = y;
```

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

Expression	Decimal Representation	Binary Representation
Zero	0	
–	-5	
–		`1 0010`
y		
z		
$y - z$		
TMax		
TMin		

Problem 2. (16 points):

Consider the following 10-bit floating point representation based on the IEEE floating point format:

- There is a sign bit in the most significant bit.
- The next $k = 4$ bits are the exponent. The exponent bias is 7.
- The last $n = 5$ bits are the significand.

Numeric values are encoded in this format as a value of the form $V = (-1)^s \times M \times 2^E$, where s is the sign bit, E is exponent after biasing, and M is the significand.

Part I

Answer the following problems using either decimal (e.g., 1.375) or fractional (e.g., $11/8$) representations for numbers that are not integers.

A. For denormalized numbers:

   (a) What is the value E of the exponent after biasing? __________

   (b) What is the largest value M of the significand? __________

B. For normalized numbers:

   (a) What is the smallest value E of the exponent after biasing? __________

   (b) What is the largest value E of the exponent after biasing? __________

   (c) What is the largest value M of the significand? __________

Part II

Fill in the blank entries in the following table giving the encodings for some interesting numbers.

Description	E	M	V	Binary Encoding
Zero		0	0	`0 0000 00000`
Smallest Positive (nonzero)				
Largest denormalized				
Smallest positive normalized				
One			1	
Largest odd integer				
Largest finite number				
Infinity	—	—	$+\infty$	

Problem 3. (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
  leal (%ebx,%ebx,2),%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 4. (12 points):

Consider the following C declarations:

```
typedef struct {
  char           name[5];
  unsigned short type;
  int            model;
  char           color;
  double         price;
} Product_Struct1;

typedef struct {
  char           *name;
  unsigned short type;
  char           color;
  unsigned short model;
  float          price;
} Product_Struct2;

typedef union {
  unsigned int      product_id;
  Product_Struct1   one;
  Product_Struct2   two;
} Product_Union;
```

A. Using the templates below (allowing a maximum of 24 bytes), indicate the allocation of data for structs of type `Product_Struct1` and `Product_Struct2`. 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. Assume the Linux alignment rules discussed in class.**

`Product_Struct1:`

```
  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                                                                       |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

`Product_Struct2:`

```
  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                                                                       |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

B. How many bytes are allocated for objects of type `Product_Struct1`, `Product_Struct2` and `Product_Union`, respectively?

(a) `sizeof(Product_Struct1) =` ________________

(b) `sizeof(Product_Struct2) =` ________________

(c) `sizeof(Product_Union)   =` ________________

C. Now consider the following C code fragment:

```
void init(Product_Union *p)
{
    /* This will zero all the space allocated for *p */
    bzero((void *)p, sizeof(Product_Union));

    p->one.type = 0xbeef;
    p->one.model = 0x10302ace;
    p->one.color = 0x8a;
    p->one.price = 1.25;
    strcpy (p->one.name, "abcdef");
    /* 'a' = 0x61  'b' = 0x62  'c' = 0x63
       'd' = 0x64  'e' = 0x65  'f' = 0x66 */
```

After this code has run, please give the value of each element of `Product_Union` listed below. 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) `p->product_id = 0x`________________

(b) `p->two.name   = 0x`________________

(c) `p->two.type   = 0x`________________

(d) `p->two.color  = 0x`________________

(e) `p->two.model  = 0x`________________

Problem 5. (6 points):

This problem tests your ability of matching assembly code to the corresponding C pointer code. Note that some of the C code below doesn't do anything useful.

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

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

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

```
pushl %ebp
movl %esp,%ebp
subl $24,%esp
movl 12(%ebp),%edx
movl 8(%ebp),%eax
movl %eax,-4(%ebp)
movl (%edx),%eax
sall $2,%eax
movl -4(%eax,%ebp),%eax
movl %ebp,%esp
popl %ebp
ret
```

Which of the functions compiled into the assembly code shown?

A) `fun4` **B)** `fun5` **C)** `fun6`

Problem 6. (12 points):

This problem tests your understanding of the stack discipline and byte ordering. Consider the following C functions and assembly code:

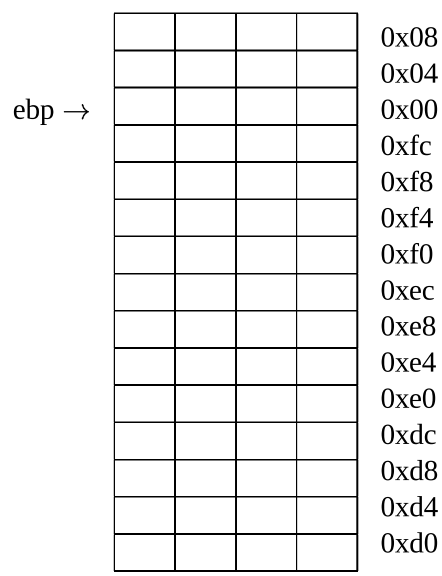
```
void check_password()
{
  char buf[8];
  scanf("%s", buf);
  if(0 != string_compare(buf, "Biggles"))
  {
    exit(1);
  }
}

int main()
{
  printf("Enter your password: ");
  check_password();
  printf("Welcome to my evil lair!\n");
  return 0;
}
```

```
80484ac <check_password>:
80484ac:    55                  push   %ebp
80484ad:    89 e5               mov    %esp,%ebp
80484af:    83 ec 24            sub    $0x24,%esp
80484b2:    53                  push   %ebx
80484b3:    83 c4 f8            add    $0xfffffff8,%esp
80484b6:    8d 5d f8            lea    0xfffffff8(%ebp),%ebx
80484b9:    53                  push   %ebx
80484ba:    68 78 85 04 08      push   $0x8048578
80484bf:    e8 a0 fe ff ff      call   8048364 <scanf>
80484c4:    83 c4 f8            add    $0xfffffff8,%esp
80484c7:    68 7b 85 04 08      push   $0x804857b
80484cc:    53                  push   %ebx
80484cd:    e8 be ff ff ff      call   8048490 <string_compare>
80484d2:    83 c4 20            add    $0x20,%esp
80484d5:    85 c0               test   %eax,%eax
80484d7:    74 0a               je     80484e3 <check_password+0x37>
80484d9:    83 c4 f4            add    $0xfffffff4,%esp
80484dc:    6a 01               push   $0x1
80484de:    e8 c1 fe ff ff      call   80483a4 <exit>
80484e3:    8b 5d d8            mov    0xffffffd8(%ebp),%ebx
80484e6:    89 ec               mov    %ebp,%esp
80484e8:    5d                  pop    %ebp
80484e9:    c3                  ret
```

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.
- `string_compare(s1, s2)` returns `0` if `s1` equals `s2`.
- `exit(1)` halts execution of the current process without returning.
- Recall that Linux/x86 machines are Little Endian.

You may find the following diagram helpful to work out your answers. However, when grading we will **not** consider anything that you write in it.



A. **Circle the address** (relative to `ebp`) of the following items. Assume that the code has just finished executing the prolog for `check_password` (through the `push` instruction at `0x80484b2`).

return address:	0x08	0x04	0x00	0xfc	0xf8	0xf4	0xf0	...	0xdc	0xd8	0xd4	0xd0
saved `%ebp`:	0x08	0x04	0x00	0xfc	0xf8	0xf4	0xf0	...	0xdc	0xd8	0xd4	0xd0
`&buf`:	0x08	0x04	0x00	0xfc	0xf8	0xf4	0xf0	...	0xdc	0xd8	0xd4	0xd0
saved `%ebx`:	0x08	0x04	0x00	0xfc	0xf8	0xf4	0xf0	...	0xdc	0xd8	0xd4	0xd0
`%esp`:	0x08	0x04	0x00	0xfc	0xf8	0xf4	0xf0	...	0xdc	0xd8	0xd4	0xd0

B. Let us enter the string “`Biggleswort h`” (not including the quotes) as a password. Inside the `check_password` function `scanf` will read this string from stdin, writing it value into `buf`. Afterwards what will be the value in the 4-byte word pointed to by `%ebp`? You should answer in hexadecimal notation.

The following table shows the hexadecimal value for relevant ASCII characters.

Character	Hex value	Character	Hex value
'B'	0x42	'i'	0x69
'g'	0x67	'l'	0x6c
'e'	0x65	's'	0x73
'w'	0x77	'o'	0x6f
'r'	0x72	't'	0x74
'h'	0x68	\0	0x00

`(%ebp) = 0x`________________________

C. The `push` instruction at `0x80484b2` saves the value of the callee-save register `%ebx` on the stack. Give the address of the instruction that restores the value of `%ebx`. You should answer in hexadecimal notation.

`0x`________________________

Problem 7. (6 points):

This problem tests your understanding of how `for` loops in C relate to IA32 machine code. Consider the following IA32 assembly code for a procedure `foo()`:

```
foo:
        pushl %ebp
        movl %esp,%ebp
        movl 16(%ebp),%ecx
        movl 12(%ebp),%eax
        movl 8(%ebp),%edx
        cmpl %ecx,%edx
        jl .L19
.L21:
        addl %edx,%eax
        decl %edx
        cmpl %ecx,%edx
        jge .L21
.L19:
        movl %ebp,%esp
        popl %ebp
        ret
```

Based on the assembly code, fill in the blanks below in its corresponding C source code. (Note: you may only use symbolic variables x, y, z, i, and $result$, from the source code in your expressions below —do *not* use register names.)

```
int foo(int x, int y, int z)
{
  int i, result;

  result = ________;

  for (i = ________; _____________; ________)  {

       result = _________________;
     }
  }
  return result;
}
```