CS 4400: Computer Systems
Fall 2014

Midterm Exam 1

- Please give your solutions in the space provided on the exam. If you choose to show your work on the exam, be sure to clearly indicate your solution to each problem.
- The exam is closed book, but you are allowed a 1 page cheat sheet, no electronics.
- The number of points assigned to each problem is indicated at the top of the page.
- Unless a problem indicates otherwise, you should assume that C code will be compiled on an IA32 platform where, for example, an `int` is 32 bits
- Make sure that you have 7 numbered pages including this cover sheet.

Name: _____

uUID: _____

Question 1:	/ 12

Given the following variable definitions, list the type and value for the following statements.

```
int         i = -1;
unsigned    u = 20;
float       f = 1.0e-02;
double      d = 2.0e20;
```

Expression	Type	Value
`i >> 2`	`int`	`-1`
`u << 4`	`unsigned`	`320`
`i + u`	`unsigned`	`19`
`f + i`	`float`	`-0.99`
`d + f`	`double`	`2.0e20`
`d * f`	`double`	`2.0e18`

Question 2:	/ 12

Consider a 7-bit two's complement representation. Fill in the empty boxes in the following table. You need not fill in entries marked —. For this problem you are doing 2's complement operations, not evaluating C code, so there is no undefined behavior. Evaluate the results of all expressions using 2's complement arithmetic. Be aware of overflow behavior.

$$2^7 = 128, 2^6 = 64, 2^5 = 32, 2^4 = 16, 2^3 = 8, 2^2 = 4, 2^1 = 2, 2^0 = 1$$

Expression	Decimal Representation	7-bit Binary Representation
—	19	0010011
—	-54	1001010
TMin	-64	1000000
TMax	63	0111111
—	37	0100101
—	-36	1011100
TMax-TMin	-1	1111111
TMin-TMax	1	0000001
TMin + 1	-63	1000001
-TMin	-64	1000000

Question 2:	/ 12

Consider a 7-bit two's complement representation. Fill in the empty boxes in the following table. You need not fill in entries marked —. For this problem you are doing 2's complement operations, not evaluating C code, so there is no undefined behavior. Evaluate the results of all expressions using 2's complement arithmetic. Be aware of overflow behavior.

$$2^7 = 128, 2^6 = 64, 2^5 = 32, 2^4 = 16, 2^3 = 8, 2^2 = 4, 2^1 = 2, 2^0 = 1$$

Expression	Decimal Representation	7-bit Binary Representation
—	47	0101111
—	-9	1110111
TMin	-64	1000000
TMax	63	0111111
-TMin	-64	1000000
—	55	0110111
—	-6	1111010
TMin + 1	-63	1000001
TMax-TMin	-1	1111111
TMin-TMax	1	0000001

Question 2:	/ 12

Consider a 7-bit two's complement representation. Fill in the empty boxes in the following table. You need not fill in entries marked —. For this problem you are doing 2's complement operations, not evaluating C code, so there is no undefined behavior. Evaluate the results of all expressions using 2's complement arithmetic. Be aware of overflow behavior.

$$2^7 = 128, 2^6 = 64, 2^5 = 32, 2^4 = 16, 2^3 = 8, 2^2 = 4, 2^1 = 2, 2^0 = 1$$

Expression	Decimal Representation	7-bit Binary Representation
TMin	-64	1000000
TMax	63	0111111
—	30	0011110
—	-41	1010111
TMin + 1	-63	1000001
-TMin	-64	1000000
—	37	0100101
—	-51	1001101
TMax-TMin	-1	1111111
TMin-TMax	1	0000001

Question 3: / 9

Consider the following 13-bit floating-point number based on IEEE format.

- The most significant bit indicates the sign.
- The next 4 bits are the exponent.
- The last 8 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.

Fill in the table below. The following are the instructions for each field.

- Hex: The 13-bit binary representation, given in 4-digit hexadecimal.
- M: The value of the significand. This should be fractional form number.
- E: The integer value of the exponent.
- Value: The numeric value represented.

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

Description	Hex	M	E	Value
Negative zero	0x1000	0	-6	-0.0
Positive infinity	0x0f00	–	–	inf
-1.125	0x1720	$1+\frac{1}{8}$	0	-1.125
NaN	0x1f10	–	–	NaN
2.0	0x0800	1	1	2.0
Smallest denormalized > 0	0x0001	$\frac{1}{256}$	-6	0.000061
Largest normalized > 0	0x0eff	$1+\frac{255}{256}$	7	255.5

Question 3: / 9

Consider the following 13-bit floating-point number based on IEEE format.

- The most significant bit indicates the sign.
- The next 4 bits are the exponent.
- The last 8 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.

Fill in the table below. The following are the instructions for each field.

- Hex: The 13-bit binary representation, given in 4-digit hexadecimal.
- M: The value of the significand. This should be fractional form number.
- E: The integer value of the exponent.
- Value: The numeric value represented.

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

Description	Hex	M	E	Value
Negative zero	0x1000	0	-6	-0.0
Positive infinity	0x0f00	–	–	inf
2.5	0x0840	$1+\frac{1}{4}$	1	2.5
NaN	0x1f10	–	–	NaN
2.0	0x0800	1	1	2.0
Smallest denormalized > 0	0x0001	$\frac{1}{256}$	-6	0.000061
Largest normalized > 0	0x0eff	$1+\frac{255}{256}$	7	255.5

Question 3: / 9

Consider the following 13-bit floating-point number based on IEEE format.

- The most significant bit indicates the sign.
- The next 4 bits are the exponent.
- The last 8 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.

Fill in the table below. The following are the instructions for each field.

- Hex: The 13-bit binary representation, given in 4-digit hexadecimal.
- M: The value of the significand. This should be fractional form number.
- E: The integer value of the exponent.
- Value: The numeric value represented.

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

Description	Hex	M	E	Value
Negative zero	0x1000	0	-6	-0.0
Positive infinity	0x0f00	–	–	inf
2.5	0x0840	$1+\frac{1}{4}$	1	2.5
-1.125	0x1720	$1+\frac{1}{8}$	0	-1.125
NaN	0x1f10	–	–	NaN
Smallest denormalized > 0	0x0001	$\frac{1}{256}$	-6	0.000061
Largest normalized > 0	0x0eff	$1+\frac{255}{256}$	7	255.5

Question 5:	/ 15

Match each of `z()` functions with one of `a`, `b`, `c`, `d`, and `e` on the right. Remember that IA32 return values are passed in `%eax`. Note: `cmov` is a conditional move.

[a]

```
int z (int x, int y)
{
    if (x>y) return x;
    return y;
}
```

[b]

```
int z (int x, int y)
{
    if (x+y) return 0;
    return y;
}
```

[c]

```
int z (int x, int y)
{
    if (x||y) return x;
    return 0;
}
```

[d]

```
int z (int x, int y)
{
    if (x+1) y++;
    return x-y;
}
```

[e]

```
int z (int x, int y)
{
    if (!x&&!y) x++;
    return y-x;
}
```

```
a:  pushl   %ebp
    movl    %esp, %ebp
    movl    8(%ebp), %eax
    movl    12(%ebp), %edx
    popl    %ebp
    cmpl    %eax, %edx
    cmovge  %edx, %eax
    ret
```

```
b:  pushl   %ebp
    movl    %esp, %ebp
    movl    12(%ebp), %eax
    movl    8(%ebp), %edx
    popl    %ebp
    addl    %eax, %edx
    testl   %edx, %edx
    movl    $0, %edx
    cmovne  %edx, %eax
    ret
```

```
c:  pushl   %ebp
    movl    %esp, %ebp
    movl    8(%ebp), %eax
    popl    %ebp
    ret
```

```
d:  pushl   %ebp
    xorl    %edx, %edx
    movl    %esp, %ebp
    movl    8(%ebp), %eax
    cmpl    $-1, %eax
    setne   %dl
    addl    12(%ebp), %edx
    popl    %ebp
    subl    %edx, %eax
    ret
```

```
e:  pushl   %ebp
    movl    %esp, %ebp
    movl    12(%ebp), %eax
    movl    8(%ebp), %edx
    testl   %eax, %eax
    jne     .L2
    testl   %edx, %edx
    movl    $1, %ecx
    cmove   %ecx, %edx
.L2:
    subl    %edx, %eax
    popl    %ebp
    ret
```

Question 6:	/ 15

Consider the following x86 assembly code for a function `loopy()`:

```
.globl loopy
     .type   loopy, @function
loopy:
     pushl   %ebp
     movl    $1, %ecx
     movl    %esp, %ebp
     movl    $1, %eax
     pushl   %ebx
     movl    8(%ebp), %edx
     movl    12(%ebp), %ebx
     cmpl    %ebx, %edx
     jae .L3
.L6:
     addl    $1, %edx
     addl    %ecx, %ecx
     cmpl    %edx, %ebx
     ja  .L6
     movl    %ecx, %eax
.L3:
     popl    %ebx
     popl    %ebp
     ret
```

Based on the assembly code above, fill in the blanks below in its corresponding C source code. You may not refer to registers in the source code.

```
unsigned loopy (unsigned x, unsigned y)
{
     int result = 1;
     while (x < y) {
          result = result*2;
          x = x+1;
     }
     return result;
}
```

Question 6:	/ 15

Consider the following x86 assembly code for a function `loopy()`:

```
.globl loopy
     .type   loopy, @function
loopy:
     pushl   %ebp
     movl    $2, %eax
     movl    %esp, %ebp
     movl    8(%ebp), %edx
     movl    12(%ebp), %ecx
.L2:
     subl    $1, %edx
     sall    $2, %eax
     cmpl    %ecx, %edx
     jb .L2
     popl    %ebp
     ret
```

Based on the assembly code above, fill in the blanks below in its corresponding C source code. You may not refer to registers in the source code.

```
unsigned loopy (unsigned x, unsigned y)
{
     int result = 2;
     do {
          result = result*4;
          x = x-1;
     } while (x < y);
     return result;
}
```

Question 6:	/ 15

Consider the following x86 assembly code for a function `loopy()`:

```
.globl loopy
     .type   loopy, @function
loopy:
     pushl   %ebp
     xorl    %eax, %eax
     movl    %esp, %ebp
     movl    8(%ebp), %edx
     movl    12(%ebp), %ecx
     cmpl    %ecx, %edx
     jae .L3
     subl    %edx, %ecx
     xorl    %edx, %edx
.L4:
     addl    %edx, %eax
     addl    $1, %edx
     cmpl    %ecx, %edx
     jne .L4
.L3:
     popl    %ebp
     ret
```

Based on the assembly code above, fill in the blanks below in its corresponding C source code. You may not refer to registers in the source code.

```
unsigned loopy (unsigned x, unsigned y)
{
     int i;
     int result = 0;
     for (i = 0; x < y; i++) {
          result = result+i;
          x = x+1;
     }
     return result;
}
```