1. Bits (16 points)  The following two questions are similar to the questions in Lab 1 and
the same ground rules apply:

- Assume all values are 32-bit 2’s complement integers.
- You may only use the operators !, ~, &, ^, |, +, <<, and >> and integer constants
  from 0 through 255 (0xFF). No other operators, control constructs, data
  structures, etc., and you may not call other functions.
- You may only use function arguments and any additional int local variables you
  declare. No global variables or other external data, no data structures like arrays.

For this exam problem, you may assume that no arithmetic overflow will occur if you add
two ints.

(a) (6 points)
/* Return the 2’s complement of x. (i.e., compute */
/* -x without using "-".) */
int negate(int x) {
    return ___~x + 1____;
}

(b) (10 points)
/* Return 1 if x < y, else return 0. */
int isLess(int x, int y) {
    int diff = x + (~y+1);  // x-y
    return ____((diff >> 31) & 0x1____;
2. Number representation – Integers (20 points) Suppose we are working on a machine with 6-bit, 2-s complement integers.

(a) What are the largest possible and smallest possible 2’s complement numbers that can be expressed in 6 bits? i.e., $T_{Max_6}$ and $T_{Min_6}$?

\[ T_{Max_6} = 31 \quad (= 2^5 - 1) \]
\[ T_{Min_6} = -32 \quad (= -2^5) \]

(b) What is the decimal value of $110100_2$ if we interpret it as a 6-bit 2’s complement signed number?

-12

(c) What is the decimal value of $110100_2$ if we interpret it as a 6-bit unsigned number?

52

(d) Add 110100 and 010011 as 2’s complement integers and convert the result to decimal.

\[
\begin{array}{c}
110100 \\
+ 010011 \\
\hline
000111 = 7
\end{array}
\]
3. Number representation – Floating Point (14 points)  A floating-point number has the form \((-1)^S \times M \times 2^E\). M is the mantissa and \(2^E\) is the exponent. Also recall that M ranges from 1.0 to (almost) 2.0.

If we convert the decimal number 9.625 to floating point, what is the binary value of M and the decimal value of E? You do not need to encode this in IEEE 754 representation with a biased exponent. Just give the values of M and E. (Hint: see the tables on the last few pages for powers of 2.)

Decimal 9.625 is 1001.101 in binary. So

\[ M = 1.001101 \]
\[ E = 3 \]

What is the binary value of frac (the fractional bits of the mantissa that are actually stored in the IEEE floating point representation)?

\[ \text{frac} = 001101 \ (\text{i.e., the bits of M excluding the leading 1}) \]
4. x86 and C code. (25 points)

We have a small x86 assembly language function that we need to disassemble.

mystery:
  pushl  %ebp
  movl  %esp,%ebp
  movl  16(%ebp),%eax
  cmpl  12(%ebp),%eax
  jge .L3
  testl %eax,%eax
  js  .L3  # js means jump if sign bit set
  movl  8(%ebp),%edx
  movl  (%edx,%eax,4), %eax
  leave
  ret
.L3:
  movl  $0, %eax
  leave
  ret

Your job is to write an equivalent C function on the next page. The name of the function is mystery (the label at the start of the assembly language code).

The original C version of the function did not use any explicit pointer operators (no * or &) so for full credit your solution should not use them either (although we’ll award generous partial credit if you can only convert the program to C with a few pointer operations). Think about other possible data structures that might be involved.

You can detach this page if it is convenient – it does not need to be turned in.

Hint: Remember that the result of an integer-valued function is returned in register eax.

Hint: In the x86 calling convention for a function mystery(a, b, c), the leftmost argument a is stored in the stack at 8(%ebp), b is at 12(%ebp), c is at 16(%ebp), and so on.

Hint: remember there is reference information on the last two pages of the exam.

It would be worth taking a minute to figure out where variables are stored in memory or registers.
4. x86 Code and C (cont.)

Write your C version of the assembly language function from the previous page here. Your answer should only need a few lines of code.

Here is the original C code that generated most of the assembly-language version. (The setup code to push %ebp and set up the stack frame, and the exit code were added by hand. They were omitted by the compiler since they weren’t really needed for this simple function.)

```c
int mystery(int a[], int n, int k) {
    if (k >= 0 && k < n)
        return a[k];
    else
        return 0;
}
```
Write an x86-64 assembly language function that is equivalent to the following C code:

```c
int loop(int x, int y) {
    while (x < y)
        x = x + y;
    return x;
}
```

Remember to use the 64-bit register and function call conventions (summarized on the last pages). The solution does not require too many assembly language instructions – maybe a dozen or so at the most. Please include brief comments in your code to help us understand how you are using the registers and what assembly code corresponds to the different parts of the C code.

Here is the version generated by gcc using gcc -S -O, with some editing to remove debugging information and to add comments in the margin.

```assembly
# x in %rdi (edi), y in %rsi (esi)

loop:
    movl %edi, %eax       # copy x to eax
    cmpl %esi, %edi       # compare x:y
    jge .L2               # jump if x>=y

.L3:
    addl %esi, %eax       # x+=y
    cmpl %eax, %esi       # compare y:x
    jg .L3                # jump if y > x

.L2:
    ret
```

There are, of course, many other ways of writing this function and all received credit provided they used the correct x86-64 conventions and properly translated the C code.