Goal:	An introduction to binary number systems and integer arithmetic, particularly the 2's complement representation of signed integers. Exploring alternatives to multiplication and division that computers can execute more quickly.

Objective: To correctly answer several questions about number systems and integer arithmetic.

Assignment: The questions below are organized into 8 major groups. Each group corresponds to an part of the webpage:

F:\LabAssignments\Exploring Integer Arithmetic\Exploring Integer Arithmetic.htm.

For each question below, use the webpage to help determine the correct answer. Write your answers on a separate piece of paper and turn the paper in to the teaching assistant when completed.

Questions:

1. Representation and Interpretation: Binary number systems define a relationship between a representation (the *bit pattern*) and its corresponding interpretation (the *value*). For example, the 8-bit binary *pattern* 00001111 is usually interpreted to represent the *value* 15.

    a. What 8-bit pattern corresponds to the value -1 when interpreted using :

    - 1's complement
    - 2's complement
    - Sign+magnitude

    Regardless of how the patterns are interpreted into values, using a fixed number of digits always imposes limitations on the range of values that can be represented within that system.

    b. What is the most negative 8-bit value that can be represented in:

    - 1's complement
    - 2's complement
    - Sign+magnitude

    Note that for "N" binary digits there is an even number of patterns, 2^N. If these are split 50/50 between positive and negative signed values, then either the magnitude of the end points of the range must be different, or else there must be two representations of one particular value.

    c. Which of the following number systems have more than one representation of zero?

    - unsigned
    - 1's complement
    - 2's complement
    - Sign+magnitude

When counting, "Rollover" refers to a bit pattern transition from all 1's to all 0's or vice-versa, while "overflow" refers to a transition (in either direction) that causes the interpreted value to exceed the range of representation. Whether or not rollover and overflow transitions occur simultaneously during counting depends on how the patterns are interpreted into values.

d. What 8-bit pattern transitions correspond to overflow when counting in:

- unsigned
- 1's complement
- 2's complement
- Sign+magnitude

2. Addition/Subtraction: Hardware adders and subtractors are merely logic ciruits that manipulate bit patterns. A single adder or subtractor can be used for numbers represented according to either the unsigned or 2's complement number systems – the only difference is how the bit patterns are *interpreted* and how overflow is detected.

   a. What are the 8-bit patterns of the addend, augend, sum, and generated carries when adding the two 8-bit unsigned numbers 174_{10} and 117_{10}?

   b. What are the bit patterns of the addend, augend, sum, and generated carries when adding the signed 2's complement numbers -82_{10} and $+117_{10}$?

   Humans recognize overflow by noticing that the result is either less than the minimum or greater than the maximum that the representation allows. Hardware detection of overflow, however, uses a different technique that does not rely on magnitude comparisons.

   c. Without considering the sum, how can overflow be detected during unsigned addition?

   d. Without considering the sum, how can overflow be detected during signed 2's complement addition?

3. Multiplication: Multiplying two binary numbers of N bits each produces a product with as many as 2N bits.

   a. What are the bit patterns of the multiplicand, multiplier and product when multiplying the unsigned numbers 200_{10} and 89_{10}?

   b. What are the bit patterns of the multiplicand, multiplier and product when multiplying the signed 2's complement numbers -56_{10} and $+89_{10}$?

   Unlike addition and subtraction, different hardware is required for unsigned versus 2's complement multiplication, with the difference occurring only in the most-significant half of the product.

   c. If the product of two 8-bit unsigned numbers can be represented correctly in the least-significant 8-bits of the 16-bit product, what must be true about the bit pattern found in the most-significant 8 bits?

   d. If the product of two 8-bit signed 2's complement numbers can be represented correctly in the least-significant 8-bits of the 16-bit product, what must be true about the bit pattern found in the most-significant 8 bits?

4. Division: Division is often described as the inverse of multiplication, and most hardware implementations divide a dividend of 2N bits by a divisor of N bits to produce a quotient of N bits and a remainder of N bits.

    a. What determines the maximum magnitude of the remainder in unsigned division?

    b. What determines the sign of the remainder in signed 2's complement division?

    Most hardware only detects those errors due to an attempt to divide by zero. Having a double-length dividend, however, means that there are *many* combinations of dividend and divisor values that produce a quotient that will not fit within N bits.

    c. Unsigned division: If the divisor is 255_{10}, what is the maximum value that can be used as a dividend without overflowing the quotient?

    d. Signed 2's complement division: If the divisor is -128_{10}, what is the maximum positive value that can be used as a dividend without overflowing the quotient?

5. Using Left Shift to Multiply by 2^k: Multiplication instructions are sometimes 10-100 times slower than most other instructions. Time-critical computations require that we use faster alternatives when possible. Shifting a binary value left by one bit is equivalent to multiplying that value by two. Shifting it left N bits is equivalent to multiplying it by 2^N.

    a. By how many bit positions must a bit pattern be shifted left in order to multiply its value by 8_{10}?

    b. How many bit positions may the 8-bit two's complement representation of -25_{10} be shifted left without overflow?

6. Using Right Shift to Divide by 2^k: Division instructions are even slower than multiplication instructions. However, dividing by 2^k is not as simple as shifting the dividend right by k bits. When shifting right the left-most bit position must retain its original value. For example, consider the result of dividing the four bit 2's complement representation of +/-4 by 2 by shifting right one bit position:

    $+4_{10}$ (0100_2) divided by 2 must yield $+2_{10}$ (0010_2)

    -4_{10} (1100_2) divided by 2 must yield -2_{10} (1110_2)

    When the quotient is not a whole number, normal integer division truncates the result towards zero (e.g., -3.5 becomes -3). However, shifting right truncates the quotient towards negative infinity (e.g., -3.5 becomes -4). Moreover, shifting doesn't provide a remainder.

    a. Arithmetic right shift by 1 is not equivalent to divide by 2 when the dividend is a negative odd value. What adjustment to the dividend is required prior to the arithmetic right shift to make the quotient correct?

7. Multiplication by a Constant: When the multiplier is a constant *other* than 2^k, often there are still faster solutions than using a multiply instruction. For example, since $10 \times y = 8y + 2y = 2^3y + 2^1y$, multiplication by 10 may be implemented by two shifts and an addition.
    a. What sequence of additions, subtractions, and left shifts is equivalent to multiplication by 14_{10}?
    b. What sequence of additions, subtractions, and left shifts is equivalent to multiplication by -19_{10}?
    c. What bit pattern characteristics produce the longest sequence of additions, subtractions, and left shifts?
8. Division by a Constant ("Reciprocal Multiplication"): Most divisions found in programs involve constant divisors, and may be implemented as multiplication by a different constant that is the reciprocal of the first. It may even be possible to replace the equivalent multiplication by a faster sequence of other instructions as seen earlier.
    a. Suppose you would like to divide a number by -21_{10}. What multiplier produces a double-length product whose most-significant half will be identical to that of the desired quotient?
    b. Give one example of a number that when divided by -21 using reciprocal multiplication requires a post correction to produce an accurate quotient. What is the nature of the correction?
    c. When should division be implemented by right shift instead of reciprocal multiplication?