Homework 1

Due Date: By the end of Friday, 9/16/2022.
Total points: 100

There are seven (7) questions.

Submit your work in a PDF file in HuskyCT.
Some answers are provided. You need to show how you find out the answers.
Practice converting small numbers without a calculator.
Always write brief comments in your code.

1. For each of the 10-bit numbers specified by hexadecimal digits below, write down 1) the 10 bits in the number, 2) its value in decimal when the bits are interpreted as 10-bit unsigned binary numbers, and 3) its value in decimal when the bits are interpreted as 10-bit two's complement numbers.
   a. `0x 1AB`
   b. `0x 2CD`

2. Convert the following decimal numbers to 8-bit 2's complement numbers, and then represent 8 bits with two hexadecimal digits.
   a. `95`
   b. `-101`

Continued on the next page.

3. For each instruction in the table, write 8 hexadecimal digits that represent the 32 bits in the destination register after the instruction is executed.

   Assume s0 is 0x98ABCD6A, s1 is 0x20FF5A98.

   Find out the answers by working on bits/hexadecimal digits. Do not convert large numbers from hexadecimal to decimal.

Instructions	Dest. reg. in 8 hexadecimal digits
`add  t0, s0, s1`	
`and  t1, s0, s1`	
`or   t2, s0, s1`	
`xor  t3, s0, s1`	
`addi t4, s0, 0x2FA`	
`andi t5, s0, -16`	
`slli t6, s0, 12`	
`srai s2, s0, 8`	

4. Suppose a value *x* is in register s1. Use a minimum number of RISC-V instructions to compute 24*x* (i.e., the product of 24 and *x*) and save the result in register s2. The instructions are shift, ADD, or SUB instructions. You do not need to use all of them. Assume the result has only 32 bits so it can be saved in a register without overflow. Explain your method.

   Note that a naïve method is to use 23 ADD instructions. It works, but apparently, it is not the correct answer.

5. Write RISC-V instructions to perform the operations in the following pseudocode on register s1. Note that bits 8 to 0 in s1 are not changed. For example, if s1 is 0x5555E3AB, it will become 0x000001AB. If s1 is 0x5555ECAB, it will become 0xFFFFFEAB. Use temporary registers like t0 and t1 to save intermediate values. Since we are dealing bits, it is more convenient to write immediate in hexadecimal.

```
IF bit 9 of s1 is 1
    Set bits 31 to 9 of s1 to 0
ELSE
    Set bits 31 to 9 of s1 to 1
```

6. The following RISC-V instructions calculate the Hamming weight (the number of 1's) of s0. The result is saved in register s1.

```
      addi   s1, x0, 0          # s1 = 0
      addi   t0, x0, 1          # Use t0 as mask to test each bit in s0
loop:
      and    t1, s0, t0         # extract a bit with the mask
      beq    t1, x0, skip       # if the bit is 0, do not increment s1
      addi   s1, s1, 1          # increment the counter
skip:
      slli   t0, t0, 1          # shift mask to left by 1
      bne    t0, x0, loop       # if the mask is not 0, continue
```

   a. If s0 is 0xFF00FF00, how many instructions are executed? Does the number of executed instructions depend on the number of 1's in s0? Does it depend on the location of 1's? Explain your answers.

   b. There are many ways to compute Hamming weight. We could test the most significant bit (bit 31) of s0. For example, extract bit 31 with an AND instruction, and compare it with 0. This is similar to the method in the code given. However, we can save one instruction. **If we treat s0 as a 2's complement number, s0 is less than 0 if and only if bit 31 in s0 is 1.** Using this method, write RISC-V instructions to compute the Hamming weight of s0. Explain your method in comments. We can start with the following two instructions. How many instructions are executed if s0 is 0xFF00FF00?

```
      addi   s1, x0, 0          # s1 = 0
      add    t0, x0, s0         # make a copy so s0 is not changed
```

7. Translate the following C code to RISC-V assembly code. Use a minimum number of instructions. Assume that the values of `a`, `i`, `j`, and `r` are in registers s1, s2, s3, and s4, respectively. All the variables are signed. Write brief comments in your code. Clearly mark the instructions that controlling the outer loop and the inner loop (for example, using different colors). There are 12 instructions in the solutions. One optimization is to load the constant (0xAABBCCDD) into register `t4` before the loop starts.

```
for (i = 1; i < a; i++)
   for (j = 0; j < i; j++)
       r += (j ^ 0xAABBCCDD);
```