Homework 2

Due Date: By the end of Friday, 2/10/2023.

Total points: 100

Submit your work in a single PDF file in HuskyCT.

Some answers are provided. You need to show how you find out the answers.

Always write brief comments in your code.

1. Suppose A and B are word arrays. The following C loop increments elements in A by 4 and saves the results into B.

```
for (i = 0; i < 100; i += 1)
      B[i] = A[i] + 4;
```

The following table shows the mapping between variables and registers.

Register	s1	s2	s3
Variable/value	i	Address of A	Address of B

We will study two implementations in RISC-V.

a) The first implementation is based on the array copy code we discussed in lecture. We just need to revise it slightly. What changes do we need? How many instructions will be executed for the loop? Note that we do not need to jump to the condition test before the first iteration because we are sure the condition is true at the beginning.

b) Loop unrolling is an optimization technique to improve the performance of programs. In the second implementation, we unroll the loop and process four array elements in A in each iteration. The unrolled loop in C is shown below. Translate the loop to RISC-V instructions. Try to minimize the number of instructions that are executed. Explain your code. How many instructions will be executed for the new loop?

```
for (i = 0; i < 100; i += 4) {
      B[i] = A[i] + 4;
      B[i+1] = A[i+1] + 4;
      B[i+2] = A[i+2] + 4;
      B[i+3] = A[i+3] + 4;
}
```

2. A two-dimensional array in C (and some other languages) can be considered as an array of one-dimensional array. For example, the following define T as an 16x8 array in C.

```
int T[16][8];
```

The two-dimensional array can be considered as an array of 16 elements, each of which is a one-dimensional array of 8 integers/words. In total there are 128 words. The words are stored in memory in the following order:

```
T[0][0],  T[0][1], …,  T[0][6],  T[0][7],
T[1][0],  T[1][1], …,  T[1][6],  T[1][7],
…
T[14][0], T[14][1], …, T[14][6], T[14][7],
T[15][0], T[15][1], …, T[15][6], T[15][7]
```

Row 0, consisting of T[0][0], T[0][1],…, and T[0][7], goes first. Row *i* is stored right after row *i* – 1, for *i* = 1, 2, …, 15. For example, T[1][0] is stored right after T[0][7]. If T[0][0] is located at address 1000, T[0][7] is located at address 1028 = 1000 + 7 * 4. And T[1][0] is located at address 1032. Similarly, we can calculate that T[2][0] is located at 1064, T[3][0] is located at 1096, and so on.

Translate the following C code to RISC-V instructions. Assume T's address is already in s9. As a practice of accessing two-dimensional arrays, do not use pointers. Explain your code, especially how you implement the loops and how you calculate T[i][j]'s address.

```
for (i = 0; i < 16; i += 1)
    for (j = 0; j < 8; j += 1)
        T[i][j] = 256 * i + j;
```

3. Encoding. For each RISC-V instruction, find out its encoding format, *the bits* in each field, and the machine code as 8 hexadecimal digits. An example is shown below. Pay attention to the number of bits in each field.

```
or    s1, s2, s3
slli  t1, t2, 16
xori  x1, x1, -1
lw    x2, -100(x3)
```

Example:

```
Instruction: addi  t0, t1, 3
Find out register numbers: addi x5,x6,3
I-type
Immediate: 000000000011
opcode:   0010011
rd:       00101
funct3:   000
rs1:      00110
rs2:      00011
funct7:   0000000
machine code in bits: 00000000001100110000010100 10011
machine code in hex: 00330293
```

A table like the following helps to place bits in the right location.

funct7	rs2	rs1	funct3	rd	opcode
0000000	00011	00110	000	00101	0010011

4. Decoding. Each 8-digit hexadecimal number in the following table represents a RISC-V instruction. For each machine code, find its encoding format, bits in each field, and then decode it into a RISC-V instruction. The steps are the reverse of those in Problem 3. Use register numbers (like x0 instead of zero). Any immediate or displacement (offset) should be in decimal.

	Machine Code
a	`0xfeaca823`
b	`0x04020713`
c	`0x00557bb3`
d	`0x414fdf13`