Q1:

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;
```

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.

We just need to add a `addi t1, t1, 4` instruction after we load the value from `A`. This means every loop iteration will have 8 instructions executed, making a total of 800 instructions total.

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?

```
        addi s4, x0, 100
        addi s1, x0, 0
loop:
        slli t0, s1, 2  # t0 = i * 4
        add t2, t0, s2  # compute addr of A[i]
        add t3, t0, s3  # compute addr of B[i]
        lw t1, 0(t2)    # t1 = *t2 = *&A[i]
        addi t1, t1, 4  # t1 += 4 = A[i] + 4
        sw t1, 0(t3)    # B[i] = t1 = A[i] + 4
        lw t1, 4(t2)
        addi t1, t1, 4
        sw t1, 4(t3)    # B[i+1] = A[i+1] + 4
        lw t1, 8(t2)
        addi t1, t1, 4
        sw t1, 8(t3)    # B[i+2] = A[i+2] + 4
        lw t1, 12(t2)
        addi t1, t1, 4
        sw t1, 12(t3)   # B[i+3] = A[i+3] + 4
        addi s1, s1, 4  # i += 4
        blt s1, s4, loop
```

In this new loop, there are 17 instructions per iteration. However, there will only be 25 iterations, which brings the number of instructions down to only 425, an improvement of around 47%. (Theoretically, with all 100 iterations unwound in the loop, there would be a theoretical minimum of only 300 instructions needed.)

Q2:

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;
```

T[i][j]'s address is computed with two adds and two left shifts. The loops are implemented by iterating over them backwards (i.e: from i=15 to i=0, and j=7 to j=0), saving two instructions loading the values 7 and 15 into the registers for i and j (this transformation is valid since all loop iterations are independent).

```
Q_2:    # T is in s9
        # NOTE: iterating over the loop backwards so i can compare to x0
        li      t1, 15          # int i = 15
        li      t2, 7           # int j = 7
loop:
        # NOTE: T[i][j] = T + (i*8 + j)*4
        slli    t0, t1, 3       # i * 8
        add     t0, t0, t2      # t0 += j
        slli    t0, t0, 2       # t0 *= 4
        add     t0, t0, s9
        # t3 = 256 * i + j
        slli    t3, t1, 8
        add     t3, t3, t2
        # actually do the assignment
        sw      t3, 0(t0)
        # inner loop
        addi    t2, t2, -1      # j -= 1
        bge     t2, x0, loop
        # outer loop
        addi    t1, t1, -1      # i -= 1
        bge     t1, x0, loop
```

Q3. Decimal strings. Write RISC-V code to add two (non-negative) numbers whose decimal representations are stored in strings. Skeleton code is provided. The program reads two numbers from the console (stdin) and prints their sum. The inputs to the program are two decimal numbers of the same length. The numbers have at least one but less than 100 decimal digits. There is no sign. We also keep the same number of digits in the sum, which means the carry generated from the highest place is discarded. Examples of input and output are shown below. In the submitted PDF file, include the code you write (not the skeleton code provided) and explain how you do addition of digits stored in memory as characters.

```
# Addition of decimal strings

.data
dst:            .space  128
str1:           .space  128
str2:           .space  128
error_message:  .asciz "Invalid input.\n"

.text
main:
        # load adresses of strings into s1, s2, and s3
        # s3 is dst, where we store the result
        lui     s3, 0x10010
        addi    s1, s3, 128
        addi    s2, s1, 128
        # read the first number as a string
        addi    a0, s1, 0
        addi    a1, x0, 100
        addi    a7, x0, 8
        ecall
        # read the second number as a string
        addi    a0, s2, 0
        addi    a1, x0, 100
        addi    a7, x0, 8
        ecall
        # find the length of str1 and store it in s4
        add     t1, s1, x0
strlen_loop:    # NOTE: don't need to jump to comparison first, since the scanned string will have '\n' before the null terminator, and so will have length > 0
        addi    t1, t1, 1
        lb      t2, 0(t1)
        bne     t2, x0, strlen_loop
        sub     s4, t1, s1      # t1 has the address of '\0', so t1 - s1
is the length of the string
        # first assert str1[s4] == str2[s4] == '\n'
        addi    s4, s4, -1
        addi    t3, x0, '\n'
        add     t1, s1, s4
        lb      t0, 0(t1)
        bne     t0, t3, error
        add     t2, s2, s4
        lb      t0, 0(t2)
        bne     t0, t3, error
        # t3 = s3 + i
        add     t3, s3, s4
        addi    t0, x0, '\n'
        sb      t0, 0(t3)
        # use t0 as the carry bit
        addi    t0, x0, 0
addition_loop: # t2 is &str2[i-1] and t1 is &str1[i-1]
        # actually do the addition, using t1 as the carry bit
        # NOTE: dst[i] = (str1[i]-'0') + (str2[i]-'0') + carry mod 10
        addi    t1, t1, -1
        addi    t2, t2, -1
        addi    t3, t3, -1
        lb      t4, 0(t1)
        addi    t4, t4, -48 # -'0'
        lb      t5, 0(t2)
        addi    t5, t5, -48 # -'0'
        add     t4, t4, t5
        add     t4, t4, t0 # add the carry
        # compare value to 10
        addi    t5, x0, 10
        blt     t4, t5, skip_carry
        addi    t4, t4, -10
        addi    t0, x0, 1
skip_carry:
        # store the value
        addi    t4, t4, '0'
        sb      t4, 0(t3)
        # keep going until t1 gets back to s1
        bne     t1, s1, addition_loop

        # print the result
        addi    a7, x0, 4
        add     a0, s3, x0
        ecall
        # exit
        addi    a7, x0, 10
        ecall
error:  # print the error message
        addi    a7, x0, 4
        addi    a0, s2, 128
        ecall
        # exit
        addi    a7, x0, 10
        ecall
```

Q4:
or s1, s2, s3
R type
0000000 10011 10010 110 01001 0110011
013964b3

slli t1, t2, 16
I type
000000010000 00111 001 00110 0010011
01039313

xori x1, x1, -1
I type
111111111111 00001 100 00001 0010011
fff0c093

lw x2, -100(x3)
S type
111110011100 00011 010 00010 0000011
f9c1a103

Q5:
feaca823
1111111 01010 11001 010 10000 0100011
S type
sw x10, -16(x25)

04020713
000001000000 00100 000 01110 0010011
I type
addi x14, x4, 64

00557bb3
0000000 00101 01010 111 10111 0110011
R type
and x23, x10, x5

414fdf13
0100000 10100 11111 101 11110 0010011
R type (?)
srai x30, x31, 20