Peter Pettino
CSE 3666
7 February, 2024

Homework 2

1)

a)

```
# a)
main:   addi    s4, x0, 100     # s4 = 100
        addi    s1, x0, 0       # initialize counter s1 = 0

loop:   slli    t0, s1, 2       # calculate offset (multiply counter by 4)

        add     t2, t0, s2      # calculate offset for s2, store in t2
        lw      t1, 0(t2)       # load word from t2 address to t1
        addi    t1, t1, 4       # add 4 to t1

        add     t2, t0, s3      # calculate offset for s3, store in t2
        sw      t1, 0(t2)       # store t1 to address of t2

        addi    s1, s1, 1       # s1 += 1
        bne     s1, s4  loop    # if(s1 != s4) goto loop
```

The only change that we need to add to the array copy code discussed in lecture is where we add 4 to A[i], then we add it to B[i]. The total number of instructions executed by the loop is 800 as there are 8 instructions within the loop that repeat 100 times each, therefore, the total number of instructions executed is 802.

b)

```
# b)
main:   addi    s4, x0, 100     # s4 = 100
        addi    s1, x0, 0       # initialize counter s1 = 0

loop:   slli    t0, s1, 2

        add     t2, t0, s2      # calculate offset (multiply counter by 4)
        lw      t1, 0(t2)       # load word from t2 address to t1, t3, t4, t5
        lw      t3, 4(t2)
        lw      t4, 8(t2)
        lw      t5, 12(t2)

        addi    t1, t1, 4       # add 4 to t1, t3, t4, t5
        addi    t3, t3, 4
        addi    t4, t4, 4
        addi    t5, t5, 4

        add     t2, t0, s3      # calculate offset for s3, store in t2
        sw      t1, 0(t2)       # store t1 to address of t2
        sw      t3, 4(t2)       # store t3 to address of t2 + 4
        sw      t4, 8(t2)       # store t4 to address of t2 + 8
        sw      t5, 12(t2)      # store t5 to address of t2 + 12


        addi    s1, s1, 4       # s1 += 4
        bne     s1, s4, loop    # if(s1 != s4) goto loop
```

The method behind the code is that it iterates over memory locations loading 4 consecutive values at a time from array A. We are able to do this using immediate offsets 0, 4, 8, and 12. Then, increment each value by 4, and then store them back into its respective memory address in an array B. We iterate this exact same process 25 times.

The amount of instructions that will be executed by the new loop will be 425 as there are 17 instructions within the loop that repeat 25 times each. Therefore the total number of instructions executed is 427.

2)

```
        .data

buf:    .space  512

        .text
main:
        lui     s9, 0x10010

        addi    s0, x0, 0       # i = 0
        addi    s1, x0, 0       # j = 0
        addi    s2, x0, 16      # s2 = 16
        addi    s3, x0, 8       # s3 = 8

loop1:  beq     s0, s2, exit    # if(i == 16) goto exit

        blt     s1, s3, loop2   # if (j < 8) goto loop2
        addi    s1, x0, 0       # j = 0

        addi    s0, s0, 1       # i += 1
        beq     x0, x0, loop1   # goto loop1

loop2:  beq     s1, s3, loop1   # if(j == 8) goto loop1

        slli    t0, s0, 5       # t0 = i * 32
        slli    t1, s1, 2       # t1 = j * 4

        add     t2, t0, t1      # t2 += (32i + 4j)

        add     t2, t2, s9      # t2 += s9

        slli    t3, s0, 8       # t3 = i * 256
        add     t3, t3, s1      # t3 += j

        sw      t3, 0(t2)       # save t3 to address of t2

        addi    s1, s1, 1       # j += 1
        beq     x0, x0, loop2   # goto loop2

exit:
```

The way in which I implemented my nested for loops was using a strategy where I used two loops, where the first loop only kept track of counter i and would go straight to exit once it reached an i value of 16. Otherwise, it would skip to the second loop which calculated T[i][j]'s address and had its own loop counter j. The second loop would repeat until j was equal to 8 and return to the first loop. In order to calculate T[i][j]'s address, it is important to recognize the pattern when incrementing i and j. For instance, everytime j increases by 1, the memory address gets offset by 4 bytes. Furthermore, everytime i increases by 1, the memory address gets offset by 32 bytes. Therefore, by recognizing this pattern, you are able to easily calculate T[i][j]'s memory address by using the equation: $T[i][j] = 1000 + (i * 32) + (j * 4)$.

3)

```
        add     t0, x0, s1      # t0 = s1
        addi    s4, s4, -1      # s4 = -1

loop1:  lb      t1, 0(t0)       # load data from t0 to t1
        blt     t1, a4, loop2   # if(t1 < "0") goto next
        addi    t0, t0, 1       # increment 1 byte

        addi    s4, s4, 1       # increment counter
        beq     x0, x0, loop1   # goto loop1


loop2:  blt     s4, x0, print   # if s4 < 0 goto print

        add     t0, s4, s1      # t0 = s4 + s1
        add     t1, s4, s2      # t1 = s4 + s2
        add     t2, s4, s3      # t2 = s4 + s3

        lb      t0, 0(t0)       # load data from t0 to t0
        lb      t1, 0(t1)       # load data from t1 to t1

        sub     t0, t0, a4      # convert t1 decimal
        sub     t1, t1, a4      # convert t1 to decimal

        add     t3, t0, t1      # t3 = t0 + t1
        add     t3, t3, t4      # add carry to t3 if needed
        add     t4, x0, x0      # reset carry

        blt     t3, a5  skip    # if t3 < 10 goto skip
        sub     t3, t3, a5      # t3 -= 10
        addi    t4, x0, 1       # initialize carry

skip:   add     t3, t3, a4      # convert t3 to ascii
        sb      t3, 0(t2)       # store t3 to address of t2

        addi    s4, s4, -1      # decrement loop counter
        beq     x0, x0, loop2   # goto loop2
```

In order to do addition of digits stored in memory as characters first we must calculate the total amount of digits in the string. We do this by iterating through each digit and testing if it is less than ‘0’ as it indicates a NULL character thus indicating the end of the string. Once we calculate the total amount of digits in the string we are then able to iterate through the strings starting at the very right of them. We first load the first byte starting from the right side of each string, we then convert both of them to decimal by subtracting 10 in order to do addition. If the result from the addition is greater than or equal to 10, we first subtract the result by 10 and then initialize a carry value that is taken into consideration the next iteration. Finally, we convert the decimal to ascii by adding 10 and store the result to its respective address in s3.

4)

a)

Instruction: or s1, s2, s3

Find out register numbers: or x9, x18, x19

It is an R-Type instruction

funct7	rs2	rs1	funct3	rd	opcode
0000000	10011	10010	110	01001	0110011

Machine code in bits: 00000001001110010110010010110011

Machine code in hex: **013964B3**

b)

Instruction: slli t1, t2, 16

Find out register numbers: slli x6, x7, 16

It is an I-Type instruction

imm[11:0]	rs1	funct3	rd	opcode
000000010000	00111	001	00110	0010011

Machine code in bits: 00000001000000111001001100010011

Machine code in hex: **01039313**

c)

Instruction: xori x1, x1, -1

Find out register numbers: xori x1, x1, -1

It is an I-Type instruction

imm[11:0]	rs1	funct3	rd	opcode
111111111111	00001	100	00001	0010011

Machine code in bits: 11111111111100001100000010010011

Machine code in hex: **FFF0C093**

d)

Instruction: lw x2, -100(x3)

Find out register numbers: lw x2, -100(x3)

It is an I-Type instruction

imm[11:0]	rs1	funct3	rd	opcode
111110011100	00011	010	00010	0000011

Machine code in bits: 11111001110000011010000100000011
Machine code in hex: **F9C1A103**

5)

a)

Machine code in hex: FEACA823

Machine code in bits: 11111110101011001010100000100011

It is an S-Type instruction

imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
1111111	01010	11001	010	10000	0100011

RISC-V Instruction: **sw x10, -16(x25)**

b)

Machine code in hex: 04020713

Machine code in bits: 00000100000000100000011100010011

It is an I-type instruction

imm[11:0]	rs1	funct3	rd	opcode
000001000000	00100	000	01110	0010011

RISC-V Instruction: **addi x14, x4, 64**

c)

Machine code in hex: 00557BB3

Machine code in bits: 00000000010101010111101110110011

It is an R-type Instruction

funct7	rs2	rs1	funct3	rd	opcode
0000000	00101	01010	111	10111	0110011

RISC-V Instruction: **and x23, x10, x5**

d)

Machine code in hex: 414FDF13

Machine code in bits: 01000001010011111101111100010011

This is an I-type instruction

imm[11:0]	rs1	funct3	rd	opcode
010000010100	11111	101	11110	0010011

RISC-V Instruction: **srai x30, x31, 20**