Homework 2

Benny Chen

September 20, 2022

Question 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.

Part A answer:

We are already given the bulk of the code from the lectures. The only change that we would need to make is to add 4 to A[i] after we load the value to t1. There would be a total of 802 executed instructions.

```
        addi    s4, x0, 100
        addi    s1, x0, 0
        beq     x0, x0, test # we know s1 < s4
loop:

        slli    t0, s1, 2 # t0 = i * 4
        add     t2, t0, s2 # compute addr of A[i]
        lw      t1, 0(t2)
        addi    t1,t1,4 # add 4 to A[i]
        add     t3, t0, s3 # compute addr of B[i]
        sw      t1, 0(t3)
        addi    s1, s1, 1

test:
        bne     s1, s4, loop # 7 instructions in the loop
```

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;
}
```

Part B answer:

The problem is similar to part A so we can start with a baseline from Part A. The change for Part B is that we are adding 4 to the index (i) and loading 4 values from A[i,i+1,i+2,i+3] to B[i,i+1,i+2,i+3]. Since we know how to find the address of A[i] and B[i] we can just add 4 to the address to increment the index by 1. There would be a total of 427 executed instructions.

```
        addi    s4, x0, 100 #100
        addi    s1, x0, 0 #i = 0
        beq     x0, x0, test # we know s1 < s4

loop:
        #Address calculation
        slli    t0, s1, 2 # t0 = i * 4

        # B[i] = A[i] + 4
        add     t2, t0, s2 # compute addr of A[i]
        lw      t1, 0(t2)
        addi    t1,t1,4 # add 4 to A[i]
        add     t3, t0, s3 # compute addr of B[i]
        sw      t1, 0(t3) # load A[i] + 4 to B[i]
        # We know A[i] and B[i] address so we just need
        # to load adress + 4 to have A and B [i+1]
        # B[i + 1] = A[i + 1] + 4
        lw      t1,4(t2) # load A[i + 1]
        addi    t1,t1,4 # add 4 to A[i + 1]
        sw      t1, 4(t3) # load A[i + 1] + 4 to B[i + 1]
        # B[i + 2] = A[i + 2] + 4
        lw      t1,8(t2) # load A[i + 2]
        addi    t1,t1,4 # add 4 to A[i + 2]
        sw      t1, 8(t3) # load A[i + 2] + 4 to B[i + 2]

        # B[i + 3] = A[i + 3] + 4
        lw      t1,12(t2) # load A[i + 3]
        addi    t1,t1,4 # add 4 to A[i + 3]
        sw      t1, 12(t3) # load A[i + 3] + 4 to B[i + 3]

        addi    s1, s1, 4 #i+=4

test:
        bne     s1, s4, loop # 7 instructions in the loop
```

Question 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;
```

Answer:

The orginal C code basically has a double for loop that in the first loop, iterates 16 times and in the second loop 8 times, all the while setting the value T[i][j] to $256 * i + j$. The RISC-V code works in a similar way. We first defined the values of i and the constants like 16 and 8. We then jump to the end where the test case is and checks if i ¡ 16 while setting j to 0. If the test case is true, we jump to the inner loop where we perform the operation to put in the value for $256 * i + j$. We then find the address of T[i][j] by first computing the address of T[i] then adding the address for j into it. We then store the value into the address then increment j and repeat the loop until all statments are true and executed.

```
        # s9 = T's address

        addi    s0,s0,0 #i = 0
        addi    s2,s2,16 #16
        addi    s3,s3,8 #8
        beq     x0,x0,test

action: slli    t0,s0,256 # t0 = i * 256
        add     t0,t0,s1 # t0 + j

        slli    t1,s0,2 # t1 = i * 4
        slli    t2,s1,2 # t2 = j * 4
        add     t3,t1,s9 # Compute address of T[i]
        add     t3,t3,t2 # Compute address of T[i][j]
        sw      t0,0(t3) # T[i][j] = t0

test2:  addi    s1,s1,1 #j++
        blt     s1,s3,action #j < 8
        addi    s0,s0,1 #i++
test:   addi    s1,x0,0 #j = 0
        blt     s0,s2,action # i < 16
```

Question 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)
```

Answer:

or s1, s2, s3

Instructions: or s1, s2, s3

slli t1, t2, 16

xori x1, x1, -1

lw x2, -100(x3)