Archie Ross
1/24/2023

CSE 3666 Homework 2

1. **Array code**
   a. **Array Copy Code:**

```
main:
        addi    s4, x0, 100
        addi    s1, x0, 0
        beq     x0, x0, test
loop:
        slli    t0, s1, 2
        add     t2, t0, s2
        lw      t1, 0(t2)
        addi    t1, t1, 4
        add     t3, t0, s3
        sw      t1, 0(t3)
        addi    s1, s1, 1
test:
        bne     s1, s4, loop
```

For this code, we had to add the line “addi t1, t1, 4” which will increment a word in A’s address and assign it to B. The number of instructions executed by this loop is 804. Because we have that test line, the beginning of the code runs 4 lines before actually starting at the loop label. Then when it reaches the loop label it is going to execute 0 to 99 times (100 times). Because there’s 7 lines in loop and then the test line, there will be 8*100 more executions. Plus that original 4 we have 804 instructions.

b. Loop Unrolling:

```
        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]
        lw t1, 0(t2)
        add t3, t0, s3 # compute addr of B[i]
        sw t1, 0(t3)
        addi t0, t0, 4
        lw t1, 4(t2)
        sw t1, 3(t3)
        addi t0, t0, 4
        lw t1, 8(t2)
        sw t1, 8(t3)
        addi t0, t0, 4
        lw t1, 12(t2)
        sw t1, 12(t3)
        addi s1, s1, 4
test:
        bne s1, s4, loop
```

For this code, we're "unrolling" the array and essentially copying 4 items in each iteration. This is why i is incrementing by 4 in the C code loop. The way we can count the number of instructions is by doing something similar to the last code and counting the first 2 instructions. Then since i is incrementing by 4 in each iteration, we simply take the number of instructions in the loop plus the test instruction and multiply it by 25 to get the total number of instructions. My code gets 402 instructions ran, 16 per loop iteration.

2. **Nested loop:**
    a.

```
        li s0, 1
        addi s1, x0, 0 #i
        addi s2, x0, 0 #j

        slli s3, s0, 4 #16
        slli s4, s0, 3 #8
        bne x0, x0, i_loop

execution: add t0, s2, s1
        slli t0, t0, 8
        slli t1, s1, 5
        slli t2, s2, 2
        add t3, t1, t2
        add t6, s9, t3
        sw t0, 0(t6)
        addi s2, s2, 1

j_loop: blt s2, s4, execution
        addi s1, s1, 1
i_loop: add s2, x0, x0
        blt s1, s3, j_loop


        #for each i, add 32 to the address, for each j add 4. We can accomplish this by slli i by 5 and slli j by 2
```

This code works by first establishing i and j as well as setting some registers to our iterable max values. Then, a branch is ran to start the loop. We start with T[0][0], and once that finishes executing it will increment j by 1 and continue looping until j is greater than 8 which then it will increment I by 1 and reset J to 0. To get the address of T I found that for every i you add 32 to the address and for every j we add 4, thus I added the lines in execution: that will multiply i and j by those numbers and add them together to get the exact address number. Then I store the result of the execution line and continue the loop.

3. **Encoding:**
    a. **OR s1, s2, s3**
        i. Find out register numbers: x9, x18, x19
        ii. R-Type
        iii. Immediate: N/A
        iv. Opcode: 0110011
        v. Rd: 01001
        vi. Funct3: 110
        vii. Rs1: 10010
        viii. Rs2: 10011
        ix. Funct7: 0000 000
        x. Machine code in bits: 0000 0001 0011 1001 0110 0100 1011 0011
        xi. Machine code in hex: 0x013964B3
    b. **SLLI t1, t2, 16**
        i. Find out the register numbers: x6, x7
        ii. R-Type
        iii. Immediate: 10000
        iv. Opcode: 0010011
        v. Rd: 00110
        vi. Funct3: 001
        vii. Rs1: 00111
        viii. Rs2 10000
        ix. Funct7: 0000000
        x. Machine code in bits: 0000 0001 0000 0011 1001 0011 0001 0011
        xi. Machine code in hex: 0x01039313
    c. **XORI x1, x1, -1**
        i. Find out register numbers: x1
        ii. I-type
        iii. Immediate: 1111 1111 1111
        iv. Opcode: 0010011
        v. Rd: 00001
        vi. Funct3: 100
        vii. Rs1: 00001
        viii. Rs2: N/A
        ix. Funct7: N/A
        x. Machine code in bits: 1111 1111 1111 0000 1100 0000 1001 0011
        xi. Machine code in hex: 0xFFF0C093

d. LW x2, -100(x3)

i. Find out the register numbers: x2, x3
ii. I-type
iii. Immediate: 1111 1110 0100
iv. Opcode: 0000011
v. Rd: 00010
vi. Funct3: 010
vii. Rs1: 00011
viii. Rs2: N/A
ix. Funct7: N/A
x. Machine code in bits: 1111 1110 0100 0001 1010 0001 0000 0011
xi. Machines code in hex: 0xFE81A103

4. **Decoding:**
    a. **0xfeaca823**
        i. First convert to binary:
            1. 1111 1110 1010 1100 1010 1000 0010 0011
        ii. Identify type:
            1. Opcode 0100011 represents an S-type instruction, likely SW
        iii. Identify Immediate:
            1. The first 7 bits skip 13 bits the next 5 bits
                a. 111111 10000
                    i. This makes -16
        iv. Identify rs2 and rs2:
            1. Rs2 the first 5 bits after the immediate: 01010
                a. This makes x10
            2. Rs1 the next 5 bits after Rs2: 11001
                a. This makes x25
        v. Identify the Funct3:
            1. 010, This means we can identify this as SW
        vi. Put it all together:
            1. SW x10, -16(x25)
    b. **0x04020713**
        i. First convert to binary:
            1. 0000 0100 0000 0010 0000 0111 0001 0011
        ii. Identify type:
            1. Opcode 0010011 represents an I type function
        iii. Identify Immediate:
            1. The first 12 bits:
                a. 0000 0100 0000
                    i. This makes 64
        iv. Identify rs1 and rd:
            1. Rs1 is the next 5 bits after the immediate:
                a. Rs1: 00100
                    i. This is just x4
                b. Rd is skip 3 bits after Rs1 (or the 5 bits after funct): 01110
                    i. This is just x14
        v. Identify the Funct3:
            1. 000, This means we can identify this as ADDI
        vi. Put it all together:
            1. ADDI x14, x4, 64

c. 0x00557bb3

i. First convert to binary:
   1. 0000 0000 0101 0101 0111 1011 1011 0011

ii. Identify type:
   1. Opcode 0110011 represents an R-type instruction

iii. Identify Funct7:
   1. 0000 000, this doesn't really narrow it down much

iv. Identify rs1, rs2 and rd:
   1. Rs2 is the next 5 bits after funct7: 00101
      a. This makes x5
   2. Rs1 is the next 5 bits after Rs2: 01010
      a. This makes x10
   3. Rd is the next 5 bits after skipping 3 (funct3): 10111
      a. This makes x23

v. Identify funct3:
   1. 111, This means in combo with funct7 we have the AND function

vi. Put it all together:
   1. AND x23, x10, x5

d. 0x414fdf13

i. First convert to binary:
   1. 0100 0001 0100 1111 1101 1111 0001 0011

ii. Identify type:
   1. Opcode 010011 which means it's another R type

iii. Identify Funct7:
   1. 0100 000

iv. Identify rs2, rs1 and rd:
   1. Rs2 is the next 5 bits after the Funct7: 10100
      a. This makes 20
   2. Rs1 is the next 5 bits after Rs2: 11111
      a. This makes x31
   3. Rd is the next 5 bits after the Funct3: 11110
      a. This makes x30

v. Identify funct3:
   1. 101 which makes this SRAI

vi. Put it all together:
   1. SRAI x30, x31, 20