Bryan Martinez

CSE 3666

10 February 2024

1)

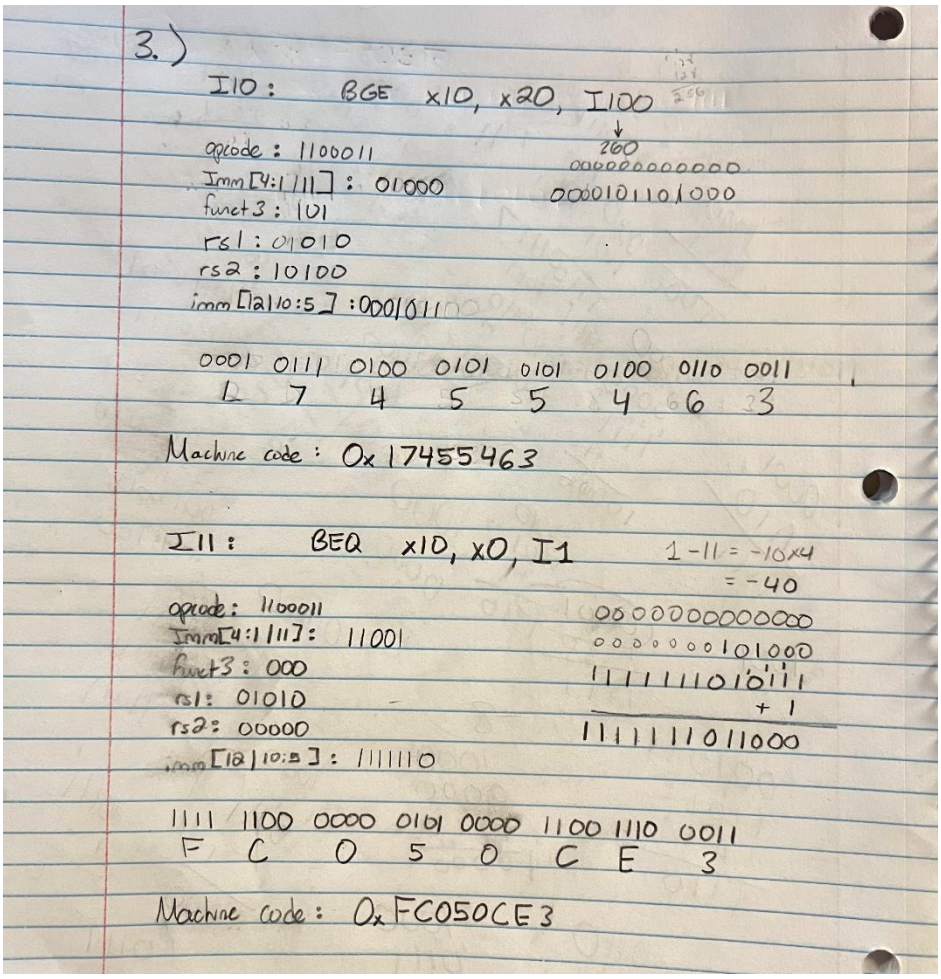
```
        .text
        .globl  main

foo:    addi    sp, sp, -20     # allocate memory to stack
        sw      ra, 16(sp)      # save return address to stack
        sw      a1, 12(sp)      # save a1 to stack
        sw      a0, 8(sp)       # save a0 to stack
        sw      s3, 4(sp)       # save memory address of d[i] to stack
        sw      s2, 0(sp)       # save s2 to stack


        addi    s1, x0, 0       # s1 = count
        addi    s2, x0, 0       # s2 = i counter


for:    beq     s2, a1, f_exit  # if n==i, then go to exit

        slli    t0, s2, 2       # word alignment of i
        add     s3, a0, t0      # go to memory address at d[i]
        add     a0, s3, x0      # save s3 to a0
        sub     a1, a1, s2      # assign n - i to a1

        jal     ra, bar         # go to bar
        bge     x0, a0, skip    # if t <= 0, then go to exit


        addi    s1, s1, 1       # incrememnt counter by 1


skip:   addi    s2, s2, 1       # i += 1
        beq     x0, x0, for     # go back to beginning of for loop


f_exit: lw      ra, 16(sp)      # load return address from stack
        lw      a1, 12(sp)      # load a1 from stack
        lw      a0, 8(sp)       # load a0 from stack
        lw      s3, 4(sp)       # load s3 from stack
        lw      s2, 0(sp)       # load s2 from stack

        addi    sp, sp, 20      # recover memory
        jalr    x0, ra, 0       # return

```

2)

```
1         .text
2         .globl  main
3
4   msort:  addi    sp, sp, -12     # allocate to stack
5           sw      ra, 8(sp)       # save return address to stack
6           sw      s2, 4(sp)       # save s2 to stack
7           sw      s1, 0(sp)       # save s1 to stack
8
9           addi    sp, sp, -1028   # allocate 256 elements to stack
10          sw      s0, 0(sp)       # save s0 to stack
11
12          blt     x1, a1, skip    # if n > 1, continue rest of msort in skip
13
14          lw      ra, 8(sp)       # load from stack to ra
15          lw      s2, 4(sp)       # load from stack to s2
16          lw      s1, 0(sp)       # load from stack to s1
17          addi    sp, sp, 12      # recover all memory
18
19          lw      s0, 0(sp)       # load from stack
20          addi    sp, sp, 1028    # recover memory
21
22          jalr    x0, ra, 0       # return
23
24
```

```
18  skip:   addi    s1, a1, 0       # saves n to s1
19          addi    s2, a0, 0       # saves d[] to s2
20          srli    s3, s1, 1       # saves n/2 to s3
21          slli    s4, s3, 2       # word alignment of n/2
22
23          add     a0, s2, x0      # set d[] to a0
24          add     a1, s3, x0      # set n/2 to a1
25          jal     ra, msort       # msort(d, n/2)
26
27          add     a0, s2, s4      # get the memory address at d[n/2]
28          sub     a1, s1, s3      # save n - n/2 to a1
29          jal     ra, msort
30
31          addi    a0, s0, 0       # save c to a0
32          addi    a1, s2, 0       # save d to a1
33          addi    a2, s3, 0       # save n/2 to a2
34          add     a3, s2, s4      # save &d[n/2] to a3
35          sub     a4, s1, s3      # save n - n/2 to a4
36          jal     ra, merge       # merge(c, d, n1, &d[n1], n - n1)
37
38          addi    a0, s2, 0       # save d[] to a0
39          addi    a1, s0, 0       # save c[] to a1
40          addi    a2, s1, 0       # save n to a2
41          jal     ra, copy        # copy(d, c, n)
```

3)



3.)

I10: BGE x10, x20, I100

260

opcode: 1100011
Imm[4:1|11]: 01000
funct3: 101
rs1: 01010
rs2: 10100
imm[12|10:5]: 0001011

0000000000000
0000101101000

0001 0111 0100 0101 0101 0100 0110 0011
1 7 4 5 5 4 6 3

Machine code: 0x17455463

I11: BEQ x10, x0, I1

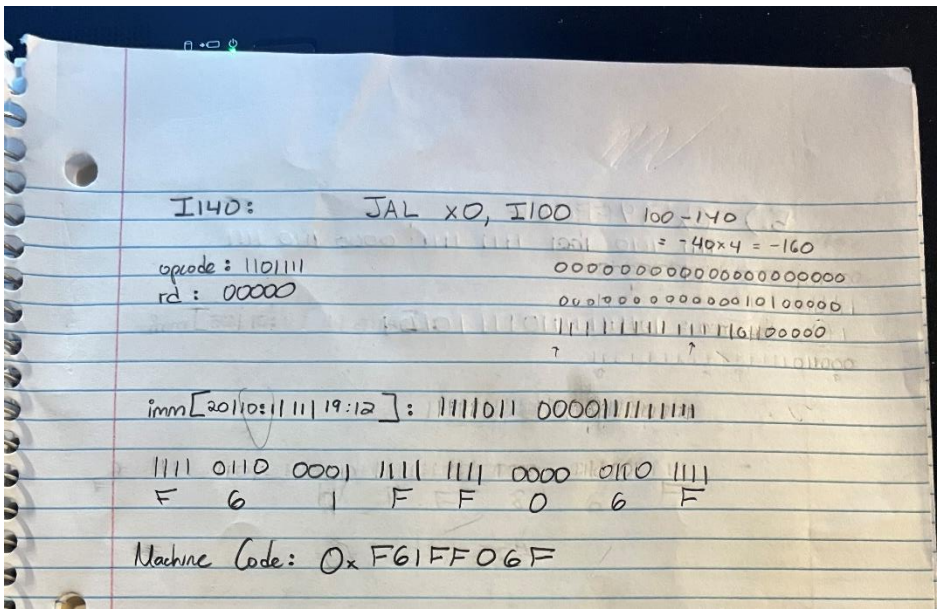
1 - 11 = -10x4
= -40

opcode: 1100011
Imm[4:1|11]: 11001
funct3: 000
rs1: 01010
rs2: 00000
imm[12|10:5]: 1111110

0000000000000
0000000101000
1111111010111
+ 1
1111111011000

1111 1100 0000 0101 0000 1100 1110 0011
F C 0 5 0 C E 3

Machine code: 0xFC050CE3



I140: JAL x0, I100

100 - 140
= -40x4 = -160

opcode: 1101111
rd: 00000

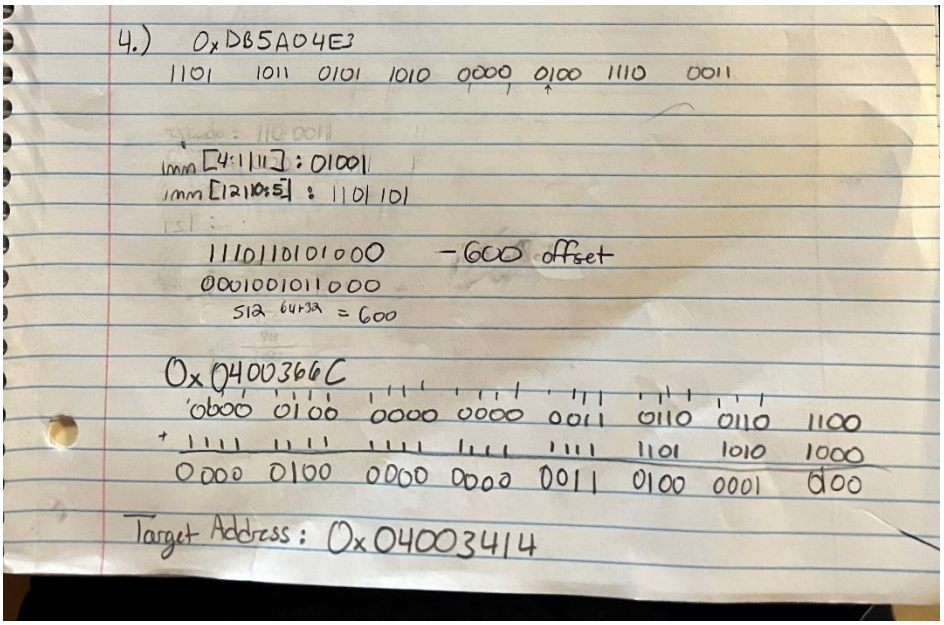
0000000000000000000000
0000000000010100000
1111111111101100000

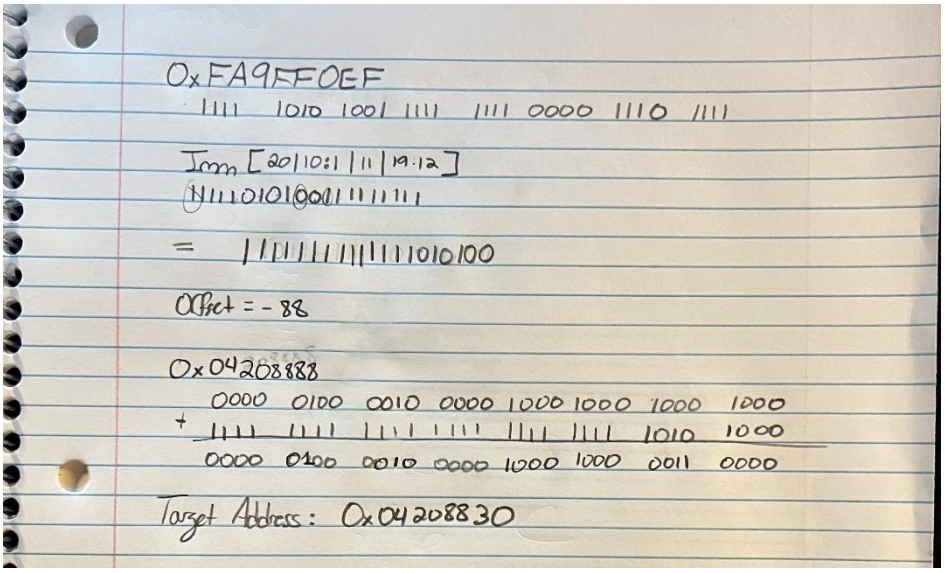
imm[20|10:1|11|19:12]: 1111011 0000111111111

1111 0110 0001 1111 1111 0000 0110 1111
F 6 1 F F 0 6 F

Machine Code: 0xF61FF06F

4)


4.) 0xDB5A04E3
1101 1011 0101 1010 0000 0100 1110 0011
imm[4:1|11] : 01001
imm[12|10:5] : 1101101
111011010100 -600 offset
0001001011000
512 64+32 = 600
0x0400366C
0b000 0100 0000 0000 0011 0110 0110 1100
+ 1111 1111 1111 1111 1111 1101 1010 1000
0000 0100 0000 0000 0011 0100 0001 0100
Target Address: 0x04003414


0xFA9FF0EF
1111 1010 1001 1111 1111 0000 1110 1111
Imm[20|10:1|11|19:12]
11111010100111111111
= 11111111111111010100
Offset = -88
0x04208888
0000 0100 0010 0000 1000 1000 1000 1000
+ 1111 1111 1111 1111 1111 1111 1010 1000
0000 0100 0010 0000 1000 1000 0011 0000
Target Address: 0x04208830