Archie Ross
1/24/2023

CSE 3666

1. For each of the 10-bit numbers specified by hexadecimal digits below, write down 1) the 10 bits in the number, 2) its value in decimal when the bits are interpreted as 10-bit unsigned binary numbers, and 3) its value in decimal when the bits are interpreted as 10-bit two's complement numbers.
    a. 0x 1AB
        i. 0110101011
        ii. 427
        iii. 427 (because the first bit is a 0, the number remains positively signed)
    b. 0x 2CD
        i. 1011001101
        ii. 717
        iii. -307 (because the first bit is a 1, we can use two's complement and get 0100110010 which equates to -307)
2. Convert the following decimal numbers to 8-bit 2's complement numbers, and then represent 8 bits with two hexadecimal digits.
    a. 95
        i. 95 - 128 = N/A | 0, 95 - 64 = 31 | 1, 31 - 32 = N/A | 0, 31 - 16 = 15 | 1, 15 - 8 = 7 | 1, 7 - 4 = 3 | 1, 3 - 2 = 1 | 1, 1 - 1 = 0 | 1.
        Binary → 01011111
        Two's Complement Binary →10100001
        Hexadecimal → 5F
    b. -101
        i. The same process as above. However after getting it we have to do two's complement to demonstrate it as negative 101.
        Binary of 101 then -101 → 01100101 → 10011011
        Hexadecimal → 9B
3. For each instruction in the table write 8 hexadecimal digits that represent the 32 bits in the destination register after the instruction is executed.
    a. For this section of the assignment, it requires us to use simple operations using s0 and s1 which are then stored in the temporary registers t0-t6 and s2.
        i. 0xb9ab2802
        ii. 0x00ab4808
        iii. 0xb8ffdffa
        iv. 0xb85497f2
        v. 0x98abd064
        vi. 0x98abcd60
        vii. 0xbcd6a000
        viii. 0xff98abcd

4.

```
1           lui     t6, 0xFFFFF
2           addi    t6, t6, 0xF00
3           slli    t0, s1, 9
4           andi    t1, s1, 1
5           beq     t1, 1, zeros
6           ori     t3, s1, t6
7           beq     x0, x0, end
8   zeros:  andi    t2, s1, 0x1FF
9   end:
10
```

This is my version of this code. I wasn't able to minimize instructions however I do believe my mind was on the right track towards a minimal line program.

5. Hamming Weight Code:
   a. To start the code, we have two initializing instructions that will always execute and will only ever execute once when code has begun. This adds 2 to our counter. Whenever a 0 is found, 4 instructions are executed. Whenever a 1 is found, 5 instructions are executed. For 0xFF00FF00 we can rewrite it as 0x11111111000000001111111100000000 . Then for each 0 we can add 4 and for each 1 we can add 5. That comes out to 144, plus the 2 initial instructions which makes 146.
   b.

```
1
2           addi    s1, x0, 0 # s1 = 0
3           add     t0, x0, s0 # make a copy so s0 is not changed
4   loop:
5           bltz    t0, skip # if the number is greater than zero don't increment. If it's less than 0, we know that there is a 1 at the MSB
6           addi    s1, s1, 1 # increment the counter
7   skip:
8           slli    t0, t0, 1 # shift the temporary s0 to the left 1
9           bne     t0, x0, loop # if the temporary isn't 0, loop again
```

6.

```
1           add s2, s2, x0  #this sets i = 0
2
3   loop:   bge s2, s1, end #this initiates the loop and ends it when i > a
4
5   if:     andi t0, s2, 0x91 #this does the operation in the parenthesis of the if statement
6           beq t0, 0 else: #this checks if the operation above is equal to 0, if it is, go the else branch
7           slli t1, s2, 4 #this is the first part of the if condition shifting i to the left 8 bits
8           xor t4, s3, s2 #this is the xor
9           add s3, t4, s3 #this makes it xorequals
10          addi s2, s2, 1 #this increments i for the next loop
11          beq x0, x0, loop #this restarts the loop after i has been incremented
12
13
14  else:   srli t2, s2, 2 #this shifts i to the right 4
15          add s3, t1, s3 #this adds the i shifted to the right 4
16          addi s2, s2, 1 #this increments i for the next loop
17          beq x0, x0, loop #this restarts the loop after i has been incremented
18
19  end: #this is the end branch and denotes the end of the given loop.
20
21
```

   a. I managed to get it to 13 instructions and possibly could have made it twelve by deleting line 9 and simply having the xor of s3 and s2 stored in s3. This required 4 branches.