Homework 1

1) s0 = 0x98ABCD6A
   s1 = 0x20FF5A98

add t0, s0, s1:

```
                 1 1111 11111 11 1111 1111
s0 =  1001 1000 1010 1011 1100 1101 0110 1010
+s1 = 0010 0000 1111 1111 0101 1010 1001 1000
-------------------------------------------
t1 = 1011 1001 1010 1011 0010 1000 0000 0010
       B    9    A    B    2    8    0    2
```

and t1, s0, s1:

AND logic			
0	0	0	
0	1	0	
1	0	0	
1	1	1	

(using s0 and s1 from above)

```
t1 = 0000 0000 1010 1011 0100 1000 0000 1000
       0    0    A    B    4    8    0    8
```

or t2, s0, s1

OR logic			
0	0	0	
0	1	1	
1	0	1	
1	1	1	

```
s0 = 1001 1000 1010 1011 1100 1101 0110 1010
s1 = 0010 0000 1111 1111 0101 1010 1001 1000
t2 = 1011 1000 1111 1111 1101 1111 1111 1010
       B    8    F    F    D    F    F    A
```

xor t3, s0, s1

same as or, but 1 ^ 1 = 0 (exclusive OR)

```
t3 = 1011 1000 0101 0100 1001 0111 1111 0010
       B    8    5    4    9    7    F    2
```

addi t4, s0, 0x2FA

2FA →
256(2) + 16(15) + 1(10)
= 762 (in dec.) (irrelevant)
⇒ 0010 1111 1010 (in binary)

```
                 1 1111 1111
s0 =  1001 1000 1010 1011 1100 1101 0110 1010
+ 0x2FA (in binary)          0010 1111 1010
-------------------------------------------
t4 = 1001 1000 1010 1011 1101 0000 0110 0100
       9    8    A    B    D    0    6    4
```

andi t5, s0, -16

2's complement for -16:
$16_{10} = 10000_2$

flip all bits: 01111 + 1 (add 1)
$= 10000_2$

(See andi operation on next page)

s0 = 1001 1000 1010 1011 1100 1101 0110 1010

and -16: 1111 1111 1111 1111 1111 1111 0001 0000

t5 = 1001 1000 1010 1011 1100 1101 0110 0000

9 8 A B C D 6 0

slli t6, s0, 12

s0: ~~1001 1000 1010~~ 1011 1100 1101 0110 1010 0000 0000 0000

12 bits X

0's shifted in from right

t6 = 1011 1100 1101 0110 1010 0000 0000 0000

B C D 6 A 0 0 0

srai s2, s0, 8

s0: 1111 1111 (1)001 1000 1010 1011 1100 1101 ~~0110 1010~~

since sign bit = 1

s2 = 1111 1111 1001 1000 1010 1011 1100 1101

F F 9 8 A B C D

---

2)

```
andi t0, s1, 0b1_0000_0000  # checking bit 9
beq else, t0, x0
andi s1, s1, 0x1FF
beq done x0, x0
else:
    ori s1, s1, 0xFFFFFE00
done:
```

code (typed out):

```
andi t0, s1, 0b1_0000_0000
beq else, t0, x0
andi s1, s1, 0x1FF
beq done x0, x0
else:
        ori s1, s1, 0xFFFFFE00
done:
```

3) a) # of executions = 32. Final value depends on number of 1's in s0. Loop begins at s1 = 0 and t0 = 1. The mask is used to extract a bit from s0 by AND-ing s0 and t0, and this gets stored in t1. If the result is 0, then that means the bit is not set and loop would skip over the increment instruction and move onto the next bit. If result is not zero, loop increments counter s1 by 1. After each iteration, t0 (the mask) gets shifted to the left by 1 bit (with slli). Loop will then test whether t0 is equal to zero using the bne instruction. If t0 is not zero, loop continues to iterate by testing next bit. If t0 is equal to zero, loop terminates and final value of s1 is Hamming Weight of s0. When s0 is equal to 0xFF00FF00, we know it is a 32-bit value w/ 16 1's and 16 0's. Loop iterates over all 32 bits and for every bit '1' it encounters, counter s1 is incremented by 1. At the end of the loop, s1 equals 16.

b)

```
addi    s1, x0, 0          # s1 = 0
addi    t0, x0, 0x80000000 # Use t0 as mask to test MSB of s0
loop:
and     t1, s0, t0         # extract MSB with the mask
bnez    t1, increment      # if the bit is 1, increment s1
shift:
slli    t0, t0, 1          # shift mask to left by 1
bnez    t0, loop           # if the mask is not 0, continue
increment:
addi    s1, s1, 1          # increment the counter
j       shift              # jump back to shift the mask
```

The code calculates Hamming Weight of a 32-bit value in register s0 using MSB approach, which involves counting the # of 1's by testing each bit from the MSB to the LSB. This approach avoids the need to extract each bit using a separate instruction by using a mask that only has one bit set at a time, which is shifted leftwards in each iteration of the loop. With s0's given value, # of instructions correctly executed will be 15 if MSB approach is used.

4)

```
li s2, 0         # load 0 into s2 (i=0)
li t0, 0x91      # load 0x91 into t0 (mask for if condition)
xor s3, s3, s3   # initializes s3 to 0 (r=0)

loop_start:
        bge s2, s1, loop_end     # brand to loop_end if i >= a
        and t1, s2, t0           # calculate (i & 0x91)
        bnez t1, if_branch       # branch to if_branch if (i & 0x91)!=0

else_branch:
        srl s3, s3, 4            # shift r right by 4 (r>>4=4)
        j end_of_iteration       # jump to end

if_branch:
        xor s3, s3, s2           # XOR r with i (r ^= i)
        end_of_iteratoin:

        # increment i
        addi s2, s2, 1           # i += 1
        j loop_start             # jump back to loop_start

loop_end:
```