HOMEWORK 4

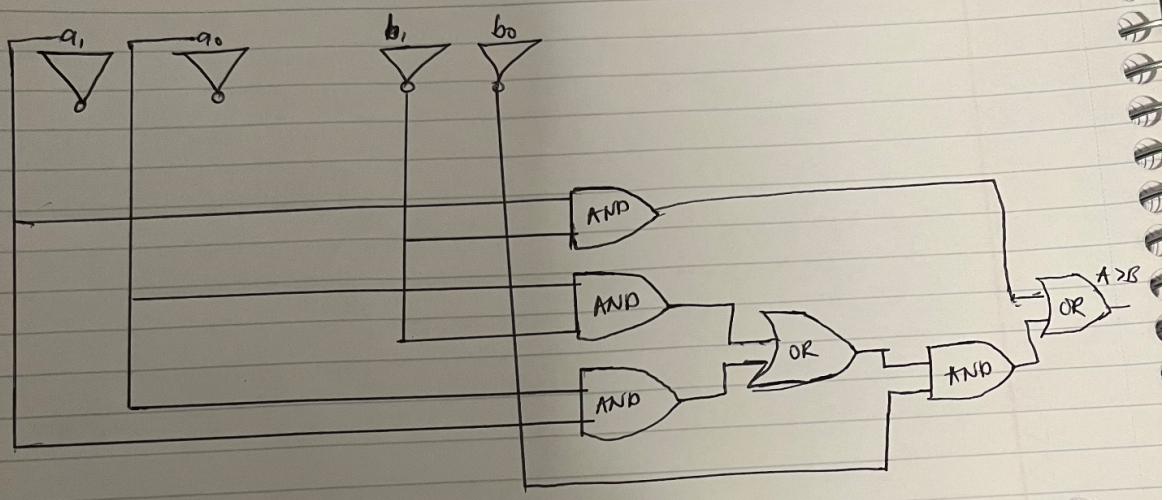
1.

a. X := A or (B AND C)

b. Y := NOT-A OR (NOT-B OR C)

2.

a1	a0	b1	b0	A > B	B > A	A = B	C
0	0	0	0	No	No	Yes	0
0	0	0	1	No	Yes	No	0
0	0	1	0	No	Yes	No	0
0	0	1	1	No	Yes	No	0
0	1	0	0	Yes	No	No	1
0	1	0	1	No	No	Yes	0
0	1	1	0	No	Yes	No	0
0	1	1	1	No	Yes	No	0
1	0	0	0	Yes	No	No	1
1	0	0	1	Yes	No	No	1
1	0	1	0	No	No	Yes	0
1	0	1	1	No	Yes	No	0
1	1	0	0	Yes	No	No	1
1	1	0	1	Yes	No	No	1
1	1	1	0	Yes	No	No	1
1	1	1	1	No	No	Yes	0


a_1
a_0
b_1
b_0
AND
AND
AND
OR
AND
OR
A >B

Q.3 Given a 32-bit register, write logic instructions to perform the following operations. For parts (c) and (f) assume an unsigned interpretation; for part (d) assume a signed interpretation.

a. Clear all even numbered bits.

X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X

AND

10101010101010101010101010101010

B. Set the last three bits.

X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X

OR

1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

C. Compute the remainder when divided by 8.

X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X

MOD

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0

D. Make the value -1

X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X

ADD

~ X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X

ADD

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

Explanation: ~ means convert to 2's complement. So the statements above first add the bits to their 2's complement, which results in zero, and then adds -1 in signed binary representation. The result should be -1

E. Complement the two highest order bits

X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X

XOR

1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

F. Compute the largest multiple of 8 less than or equal to the value itself

X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X

MOD

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0

STORE Y

X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X

SUB

Y

This means take the bits, find the mod when divided by 8. Store this mod as bits in Y. Then subtract Y from the original number to get the to get the largest multiple of 8 less than or equal to the original value.

4. JMP start
   next: 0
   Current: 0
   lastNum: 256

```
start:  LOAD     current
        WRITE    0x8
        ADD      1
        STORE    next
        SUB      lastNum
        JZ       end
        JUMP     start
end:    JUMP     end
```

5. C0000003
00000001
30000008
40000001
10000002
50000002
D0000000
C0000003

6.
```
      JMP          start
nums: []
num1: 0
num2: 0
divisor: 0
dividend: 0
checker: 0
start: LOAD    checker
       JZ      initialize
       JMP     getDivisor

end:   JMP     end
```

```
initialize:     LOAD        0x100
                STORE       nums
                LOAD        nums[0]
                STORE       num1
                LOAD        nums[1]
                STORE       num2
                LOAD        checker
                ADD         1
                STORE       checker
                JMP         start

getDivisor:     LOAD        num1
                LOAD        num2
                SUB         num1
                JGZ         dividendIsNum2
                JLZ         dividendIsNum1

dividendIsNum1: LOAD        num1
                STORE       dividend
                LOAD        num2
                STORE       divisor
                JMP         Euclid

dividendIsNum2: LOAD        num1
                STORE       divisor
```

```
                    LOAD        num2
                    STORE       dividend
                    JMP         Euclid

Euclid:             LOAD        dividend
                    MOD         divisor
                    STORE       num1
                    JZ          gcdFound
                    LOAD        divisor
                    STORE       num2
                    JMP         start

gcdFound:           LOAD        num1
                    WRITE       Ox200
                    JMP         end
```

7. Ask if he meant swap values!

```
          JMP               start
          start:  LOAD      0x30AA
```

8.

```
          JMP               start
          start: JGZ        0x837BBE1
```

9. Part 1: After the instruction xor r8, r9 , the data in r8 will change. After the operation r9, r8, the data in r9 will change. After the operation xor r8, r9, the data in r8 will change again

Part 2: This is because the xor operation performs an exclusive or on the bits stored at r8 and r9. It will sum the bits, without carrying anything over, and place the result of the operation in the first operand. For example: say r8 has the data 00111011 and r9 has the data 00111111. The instruction

xor r8, r9

Will perform the operation:
00111011 + 00111111 = *00000100*

The result *00000100* stored in r8. So the data in the first register in each instruction will keep on changing has stated in part 1.