1. Consider the function with three inputs (A,B,C) and two outputs (X,Y) that works like this:

A	B	C	X	Y
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	1	1
1	0	0	1	0
1	0	1	1	1
1	1	0	1	0
1	1	1	1	1

Design two logic circuits for this function, one using AND, OR and NOT gates only, and one using NAND gates only. *You DO NOT HAVE to draw the circuit*, but it might be helpful to do that to visualize and trace the logic. However, for this question you are only required to write the four formulas — one for computing `X` and one for computing `Y` for each circuit type. They can take the form of a logical equation such as

`X := A and B` or such as `Y := not-B and (A or C)`.

X := A or (B and C)

Y:= not-A or (A and C)

NAND

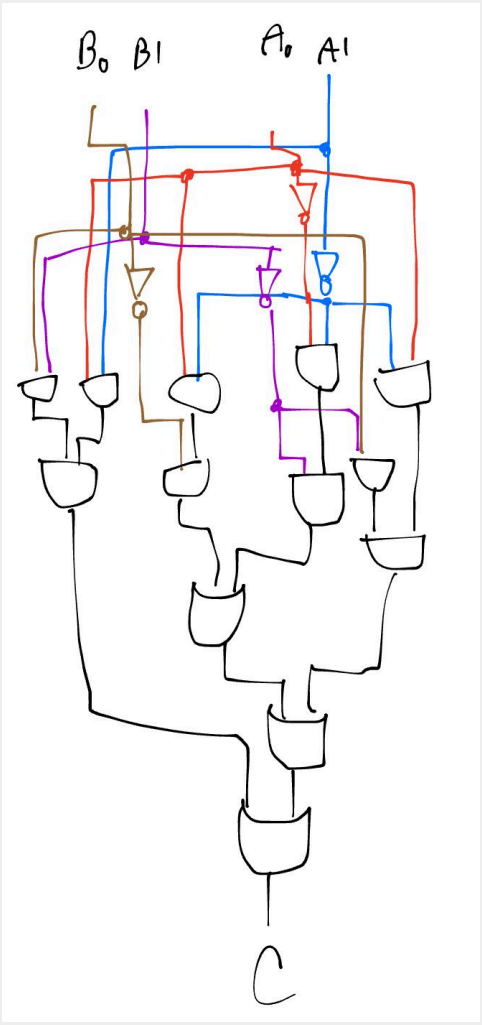
X:= (A NAND A) NAND (B NAND C)

Y:= (C NAND C) NAND A

2. Draw a logic circuit that compares two 2-bit signed numbers as follows. It should have four inputs a_1, a_0, b_1, and b_0. a_1a_0 is a 2-bit signed number (call it a)

and b_1b_0 is a 2-bit signed number (call it b). The circuit has one output, c, which is 1 if a > b and 0 otherwise.

A1	A0	B1	B0	C
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0



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.

   AND 0xAAAAAAAA

b. Set the last three bits.

   OR 0x00000007

c. Compute the remainder when divided by 8.

   AND 0x00000007

d. Make the value -1

   OR 0xFFFFFFFF

e. Complement the two highest order bits

   XOR 0xC0000000

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

   AND 0xFFFFFFF8

4. For the sample single-accumulator computer discussed in class, write a complete assembly language program in the `stanley/penguin` language that sends the values `0` through `255` out to port `0x8`. NOTE: the machine code for this will be written in the next problem.

```
        JMP Start
var: 0
Start: LOAD     var
       WRITE    0x8
       ADD      1
       STORE    var
       SUB      0xFF
       JLZ      Start
end:   JMP      End
```

5. Translate your assembly language program in the previous problem to machine language.

C0000002 ; jump to start (skips 1 line to line 3)

00000001 ; initialize var (0)

00000000 ; load var (0) AND start

30000008 ; write var in port 8

40000001 ; add 1

10000001 ; store var

500000FF ; subtract 255

E0000002 ; jump to start if less than 0

C0000009 ; jump to end and stop

6. For the sample single-accumulator computer discussed in class, write a complete assembly language program in the `stanley/penguin` language that computes a greatest common divisor. Assume the two inputs are read in from port `0x100`. Write the result to port `0x200`. You do not need to write machine code for this problem.

```
        JMP         Start
a: 0
b: 0
r: 0
Start : READ        0x100   ; Input a
        STORE       a       ; Store input a
        READ        0x100   ; input b
        STORE       b       ; store input b
loop : JZ           return  ; if b is 0, jump to return area
        MOD         a       ;  b mod a = remainder
        STORE       r       ; store remainder
        LOAD        b       ; load b
        STORE       a       ; store b into a
        LOAD        r       ; load r
        STORE       b       ; store r into b
        JMP         loop    ; start the loop again
```

```
return: LOAD    a       ; loading the remainder
        WRITE   0x200; printing out to port 200

end:    JMP     end
```

7. For the sample single-accumulator computer discussed in class, give a code fragment, in assembly language of the `stanley/penguin` language, that swaps the accumulator and memory address `0x30AA`. You do not need to write machine code for this problem.

```
        JMP start
Var = 0
start:  READ    0x08
        STORE   Var
        LOAD    0x30AA
        STORE   Var
end:    JMP     end
```

8. For the sample single-accumulator computer discussed in class, give a code fragment, in assembly language of the `stanley/penguin` language that has the effect of jumping to the code at address `0x837BBE1` if the value in the accumulator is greater than or equal to `0`. You do not need to write machine code for this problem.

```
      JLZ   end
      JMP   0x837BBE1
End:  JMP end
```

9. Part 1 of 2: Explain, at a high-level, what the following sequence of instructions does. In other words, suppose a programmer has stored data in `r8` and `r9`. After executing these instructions, what does the programmer notice about the data?

```
xor r8, r9
xor r9, r8
xor r8, r9
```

The sequence of instructions makes it so that the variable r8 and r9 swap with each other. First it combines the values of r8 and r9 and stores them in r8. Then it gives the combined value of r8 and r9 and extracts the original value of r8 and stores it in r9. Lastly, it uses the updated r9 value to retrieve the original value of r9 and stores it in r8. The programmer would notice that the values would be switched.

Part 2 of 2: Also state as briefly as possible why that effect happens.

The variables are swapped because XOR is able to create a switch without storing registers, and it leads to swapping the two values.

R9 = 0
R8 = 1