Homework #4

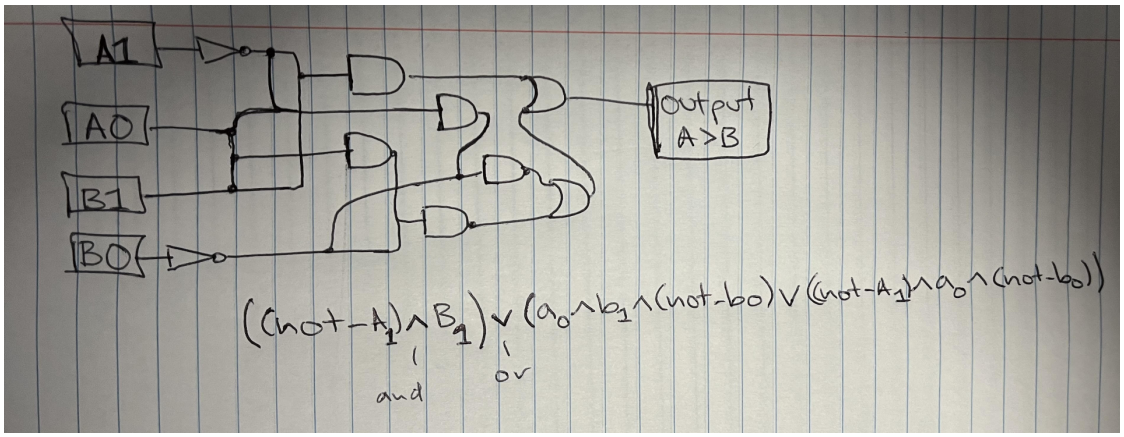
1.

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

a. "X := A or (B and C)"
b. "Y := not-A or C"
c. "X := (A nand A) nand (B nand C)
d. "Y := A nand (C nand C)"

2.

A1	A0	B1	B0	A>B
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0



3.

a. AND with 0xAAAAAAAA
b. OR with 0x00000007
c. AND with 0x00000007
d. OR with 0xFFFFFFFF
e. XOR with 0xC0000000
f. AND with 0xFFFFFFF8

4.

```
            JMP     start       ; begin by jumping over the data area
current:    0                   ; store the current value here, starting at 0
next:       1                   ; sets the incremental value, next, to 1
limit:      256                 ; limit of 256 since we are checking the next value
start:      LOAD    current     ; load the value of current
            WRITE     8         ; write the value of current out to port 0x8
            ADD     next        ; increments the value of current by 1 for the next value
            STORE   current     ; stores the next value in current for next time
            SUB     limit       ; subtracts the limit from current
            JLZ     start       ; checks if limit < 0, and if not, jump back to start
end:        JUMP    end         ; if the limit is 0, stop the program
```

5.

C0000004
00000000
00000001
00000100
00000001

30000008
40000002
10000001
50000003
E0000004
C000000A

6.

```
            JMP     start       ; begin by jumping to start
x:          0                   ; stores the value of x as 0
y:          0                   ; stores the value of y as 0
oldY:       0                   ; stores the value of oldY as 0
start:      LOAD    0x100       ; loads the original value at port 0x100
            STORE   x           ; stores the value of x
            LOAD    0x100       ; loads the value of 0x100
            STORE   y           ; stores the value of y
            LOAD    y           ; loads the value of y
            JZ      done        ; jump to done if zero
            STORE   oldY        ; stores the value of oldY
            LOAD    x           ; loads the value of x
            MOD     y           ; finds the remainder of y
            STORE   y           ; stores the value of y
            LOAD    oldY        ; loads the value of oldY
            STORE   x           ; stores the value of x
            JMP     start       ; jumps back to the start + repeats the steps
done:       LOAD    x           ; loads the value of x
            WRITE   0x200       ; writes the result to port 0x200
end:        JMP     end         ; jump to the end / terminates the program
```

7.

```
STORE   a1      ; a1 stores the original value in the accumulator
LOAD    0x30AA  ; load original value at 0x30AA
STORE   a2      ; a2 stores the original value in 0x30AA
LOAD    a1      ; loads value at a1 (the original value in the accumulator)
STORE   0x30AA  ; 0x30AA now stores the original value in the accumulator
LOAD    a2      ; accumulator now holds the original value of 0x30AA
```

8.

```
JGZ   0x837BBE1
JZ    0x837BBE1
```

9.

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

Part 1: These instructions will swap the values of r8 and r9.

Part 2: On line 1, we combine the values for r8 and r9 using XOR to get a "hybrid" of the two and store this value in r8. When using XOR to save information, you can remove it by doing an XOR again. On line 2, we take advantage of this, as we XOR the hybrid value with the r9 value, which cancels out all the r9 information, leaving us only with the original r8 value. We save this result back into r9, so the first swap has been done, with r9 holding r8's original value. On the last line, r8 still has the hybrid value, so we XOR this value again with the new value of r9, r8's original value, to cancel out all r8 information from the hybrid value. This leaves us with the original value of r9 for r8, completing the swap process. This can be seen in the following illustration.

		r8	**r9**
Initial Values:		**x**	**y**
Line 1:	XOR r8, r9	x XOR y	y
Line 2:	XOR r9, r8	x XOR y	y XOR (x XOR y) = x
Line 3:	XOR r8, r9	x XOR (x XOR y) = **y**	**x**