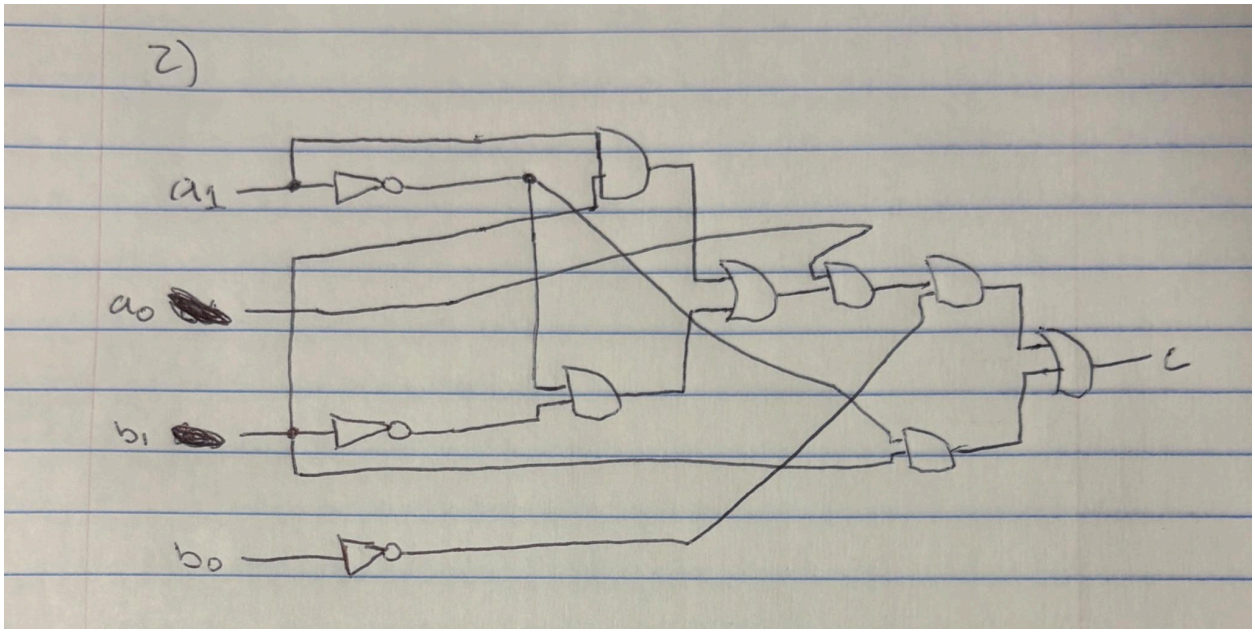
1)

a) X := A or (B and C)
b) Y := C or (not-A)
c) X := NAND(NAND(A,A), NAND(NAND(B,C), NAND(B,C)))
d) Y := NAND(A, NAND(C,C))

2) .



3) a) Register = Register AND 0xAAAAAAAA
b) Register = Register OR 0x00000007
c) Register = Register AND 0x00000007
d) Register = Register OR 0xFFFFFFFF
e) Register = Register XOR 0xC0000000
f) Register = Register AND 0xFFFFFFF8

4)

```
#include <stdio.h>

int main() {
   int N;

   printf("Enter a positive integer (N): ");
   if (scanf("%d", &N) != 1) {
      printf("Error: Invalid input. Please enter an integer.\n");
      return 1;
   }

   if (N <= 0) {
      printf("Error: number must be greater than zero.\n");
      return 1;
```

```
    }

    for (int i = 1; i <= N; i++) {
        if (i % 3 == 0 && i % 5 == 0) {
            printf("fizz-buzz\n");
        } else if (i % 3 == 0) {
            printf("fizz\n");
        } else if (i % 5 == 0) {
            printf("buzz\n");
        } else {
            printf("%d\n", i);
        }
    }

    return 0;
}
```

5)

```
        JMP     start
num:    0
adder: 1
limit:  255
start:  Load    [num]
        Write   0x8
        Add     [adder]
        Store   [num]
        Sub     [limit]
        JLZ     start
end:    JMP     end
```

6)

```
C0000004
00000000
00000001
000000FF
00000001
30000008
40000002
10000001
50000003
E0000004
C000000A
```

7) .

```
        JMP start
input1: 0
input2: 0
remainder: 0
start:  READ 0x100
        STORE [input1]
        READ 0x100
        STORE [input2]
        LOAD [input2]
        JZ end
        LOAD [input1]
        MOD [input2]
        STORE [remainder]
        LOAD [input2]
        STORE [input1]
        LOAD [remainder]
        STORE [input2]
        LOAD [input2]
        JGZ start
        LOAD [input1]
        WRITE 0x200
end:    JMP end
```

8) .

```
        JMP start

temp: 0

start:  Store temp
        Load 0x30AA
        Store 0xACC
        Load temp
        Store 0x30AA
end:    JMP end
```

9) .

```
JMP 0x837BBE1
```

10)

a) After executing these instructions, the programmer notices that each register has swapped their data with the other.

b) This effect happens because when you have the same value twice and you XOR it, they cancel out which allows both registers to swap their data.