1. Below is a partially filled out truth table for a function called High3, which takes a three-bit input and outputs a two-bit code indicating the highest index of the input that is 1. For example, if the input is 001, then the output would be 00 (since in[0] is the highest index for which an input bit is 1). If the input is 101, then the output would be 10. If no input bits are 1, then the output should be 00.

    a. Complete the truth table for this function:

in[2]	in[1]	in[0]	out[1]	out[0]
0	0	0	0	0
0	0	1	0	0
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

    b. Write boolean expressions for out[1] and out[0] based on the truth table above.

    out[1] = in[2]
    out[0] = (~in[2]) & in[1]

    c. Implement your boolean expressions by either drawing a circuit diagram or writing the equivalent HDL syntax. If you use a circuit diagram, name your inputs in[2], in[1], in[0] and your outputs out[1], out[0].

```
CHIP High3 {
      IN in[3];
      OUT out[2];

      PARTS:
      // Your code here:

      And(a=true, b=in[2], out=out[1]);
      Not(in=in[2], out=notin2);
      And(a=notin2, b=in[1], out=out[0]);

}
```

2. Implement the following logic in Hack assembly, using the registers R0, R1, and R2 as appropriate. You may use other registers as needed.

```
if (R0 + R1 < 0) {
    R2 = R0;
} else if (R0 + R1 > 3) {
    R2 = R1
} else {
    R2 = 0;
}
```

One possible solution:

```
(START)
    @R0
    D = M
    @R1
    D = D + M
    @BRANCH1
    D; JLT
    @3
    D = D - A
    @BRANCH2
    D; JGT
    // else case
    @R2
    M = 0
    @END
    0; JMP
(BRANCH1) // First if case
    @R0
    D = M
    @R2
    M = D
    @END
    0; JMP
(BRANCH2) // else if case
    @R1
    D = M
    @R2
    M = D
(END)
    @END
    0; JMP
```

3. Here is a buggy Hack assembly program:

```
01.       @9
02.       D = A
03.       @0
04.       M = D
05.   (L0)
06.       @0
07.       D = M
08.       // PART I
09.       @END
10.       D; JLE
11.       @0
12.       M = D - 1
13.       A = D
14.       // PART II
15.       M = A - 1
16.       @L0
17.       0; JMP
18.   (END)
19.       @END
20.       0; JMP
```

Here is the state of part of memory **before** the above code runs (we will use this to answer the questions below):

Address	**Value**
0	3
1	10
2	95
3	6
4	32
5	17
6	24
7	13
8	3
9	2

a. Walk through the code using the state of memory given above. Indicate the value of the registers A, D, and M at each of the following locations commented “PART #” the first time that location is reached when executing the code.

   i. Values of A, D, and M when first reaching comment with “PART I”

      A = 0
      D = 9
      M = 9

   ii. Values of A, D, and M when first reaching comment with “PART II”

      A = 2
      D = 9
      M = 9

b. What are the values stored in address 0, address 1, and address 2 after the above code runs to completion, (enters the END infinite loop), again starting with the state of memory given before the code runs?

   i. Value stored in address 0: **0**

   ii. Value stored in address 1: **0**

   iii. Value stored in address 2: **1**

c. The above code attempts to subtract 1 from the first 9 values in memory starting at address 1. The goal of the program above is to be equivalent to the following pseudocode:

```
for (i = 1; i <= 9; i++) {
      ram[i] -= 1
}
```

   As stated above, the program is buggy. It turns out it can be fixed by changing one line. Give the line number for the line of code that you would change to fix the program and what you would change it to.

   i. Line number that should be fixed: **15**

   ii. New line of code: **M = M - 1**