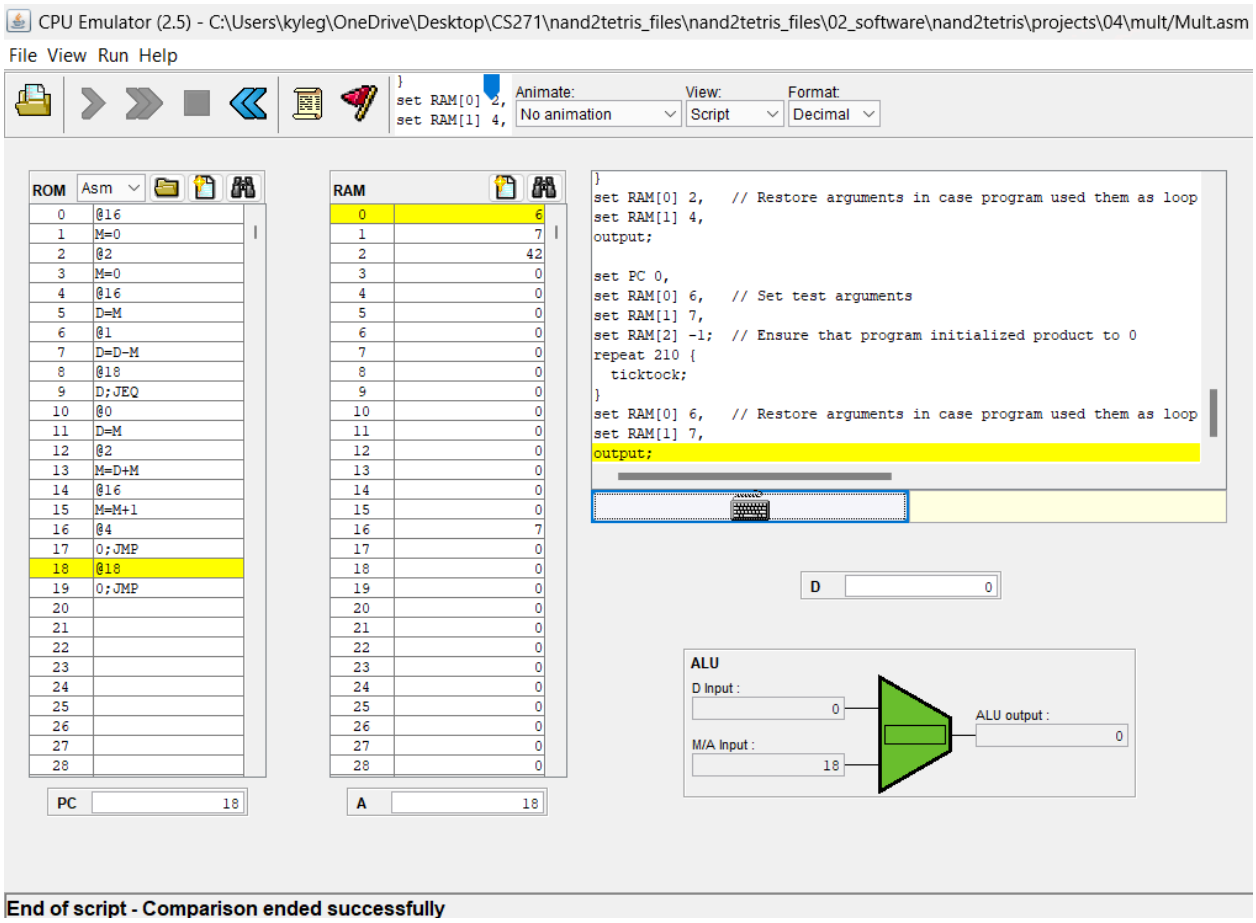
Mult.asm Code:

```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/04/Mult.asm

// Multiplies R0 and R1 and stores the result in R2.
// (R0, R1, R2 refer to RAM[0], RAM[1], and RAM[2], respectively.)
//
// This program only needs to handle arguments that satisfy
// R0 >= 0, R1 >= 0, and R0*R1 < 32768.

    @i      //i is the counter
    M=0     //setting counter initially to 0
    @R2     //R2 is where the product will be stored
    M=0     //setting product to be initially 0
(LOOP)
    @i      //accessing i
    D=M     //setting D equal to memory of i
    @R1     //accessing R1
    D=D-M   //checking to see if i==R1
    @END
    D;JEQ   //if i==R1, then JUMP to END

    @R0     //accessing R0
    D=M     //setting D equal to memory of R0
    @R2     //accessing R2
    M=M+D   //repeated addition to represent multiplication

    @i      //accessing i
    M=M+1   //incrementing i by 1
    @LOOP   //starting the LOOP
    0;JMP   //JUMP to LOOP

(END)       //infinite loop
    @END
    0;JMP
```

Mult Test:




```
}
set RAM[0] 2,   // Restore arguments in case program used them as loop
set RAM[1] 4,
output;

set PC 0,
set RAM[0] 6,   // Set test arguments
set RAM[1] 7,
set RAM[2] -1;  // Ensure that program initialized product to 0
repeat 210 {
  ticktock;
}
set RAM[0] 6,   // Restore arguments in case program used them as loop
set RAM[1] 7,
output;
```

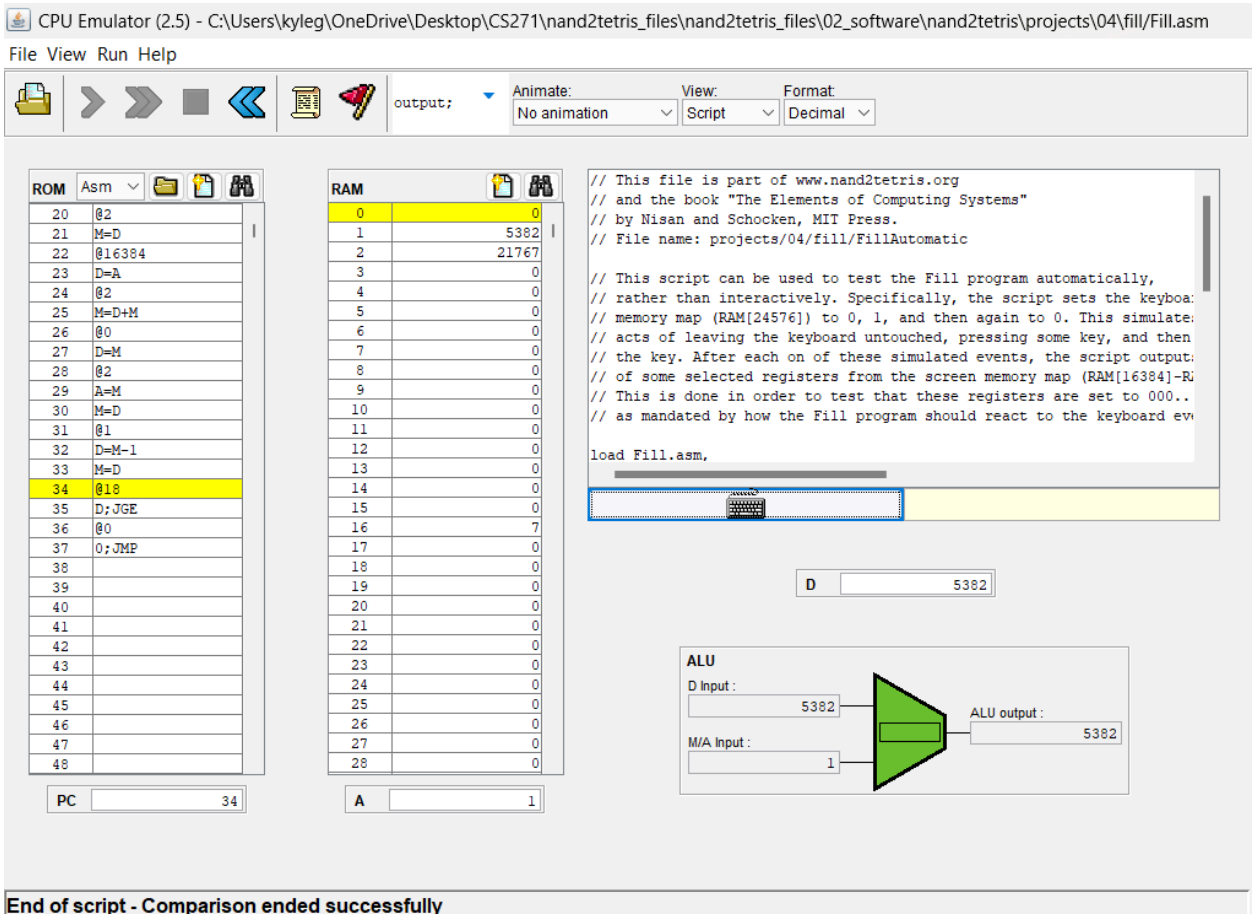
Fill.asm Code:

```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/04/Fill.asm

// Runs an infinite loop that listens to the keyboard input.
// When a key is pressed (any key), the program blackens the screen,
// i.e. writes "black" in every pixel;
// the screen should remain fully black as long as the key is pressed.
// When no key is pressed, the program clears the screen, i.e. writes
// "white" in every pixel;
// the screen should remain fully clear as long as no key is pressed.

//Note that I did not work on this code independently. I had alot of
struggle starting this portion of the
```

```
//project, and heavily relied on online resources for its completion. I followed along how another person
//completed the assignment and understood the process. This is why I included many comments through the
//code to not only better my understanding, but to show that I actually understand what is going on.

//R0: 16-bit integer. If -1, then program enters ON. If 0, then program enters OFF
//R1: Row counter.
//R2: current position holder
//LOOP: Main loop where keyboard input is always being sensed
//ON: Loop for if Keyboard is pressed; for blackshading
//OFF: Loop for it Keyboard is NOT pressed; for whiteshading
//ACTION: Loop for keeping track of pixel location and filling/unfilling pixels
//INRCEM: Loop for incrementing each row of the SCREEN, which there are 255 of them

(LOOP)
    @KBD    //Keyboard register value, 16384
    D=M     //sets D as Keyboard register value
    //If KBD>0, executes ON
    @ON
    D;JGT
    //If KBD=0, executes OFF
    @OFF
    D;JEQ

(ON)
    @R0
    //Setting M=-1 allows for setting 16 pixels black at a time
    M=-1    //-1 = 1111111111111111
    @ACTION
    0;JMP

(OFF)
    @R0
    M=0
    @ACTION
```

```
    0;JMP

(ACTION)
//since one row is 512 bit and I can assign 16 bits at a time, i want an
increment of 512*16=8192
    @8191
    D=A
    @R1 //setting increment counter to 8191
    M=D
    (INCREM)
        @R1
        D=M     //setting D to be total number of iterations I need to do
per row
        @R2
        M=D     //setting R2 to be total number of iterations I need to do
per row
        @SCREEN
        D=A    //getting D equaling to the register address of the SCREEN
        @R2
        M=M+D  //incremented by one row

        @R0
        D=M    //setting D= -1 or 0, which maps back to SCREEN
        @R2
        A=M //memory mapping R0 to R2, in which points to SCREEN meory
address
        M=D //changing the address of R2 to map to ON (-1) or OFF (0)

        //removing 1 row because 1 row has been filled or unfilled
        @R1
        D=M-1
        //resetting counter
        M=D
        //if still needs to go through rest of rows so loop until D=0
        @INCREM
        D;JGE
    @LOOP
    0;JMP
```

Fill Test:


CPU Emulator (2.5) - C:\Users\kyleg\OneDrive\Desktop\CS271\nand2tetris_files\nand2tetris_files\02_software\nand2tetris\projects\04\fill/Fill.asm
File View Run Help
output;
Animate:
No animation
View:
Script
Format:
Decimal
ROM
Asm
20 @2
21 M=D
22 @16384
23 D=A
24 @2
25 M=D+M
26 @0
27 D=M
28 @2
29 A=M
30 M=D
31 @1
32 D=M-1
33 M=D
34 @18
35 D;JGE
36 @0
37 0;JMP
PC 34
RAM
0 0
1 5382
2 21767
3 0
4 0
5 0
6 0
7 0
8 0
9 0
10 0
11 0
12 0
13 0
14 0
15 0
16 7
17 0
18 0
19 0
20 0
21 0
22 0
23 0
24 0
25 0
26 0
27 0
28 0
A 1
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/04/fill/FillAutomatic
// This script can be used to test the Fill program automatically,
// rather than interactively. Specifically, the script sets the keyboa
// memory map (RAM[24576]) to 0, 1, and then again to 0. This simulate
// acts of leaving the keyboard untouched, pressing some key, and then
// the key. After each on of these simulated events, the script output
// of some selected registers from the screen memory map (RAM[16384]-R
// This is done in order to test that these registers are set to 000..
// as mandated by how the Fill program should react to the keyboard ev
load Fill.asm,
D 5382
ALU
D Input : 5382
M/A Input : 1
ALU output : 5382
End of script - Comparison ended successfully