Elements of Computing Systems - II

Batch B Team 17

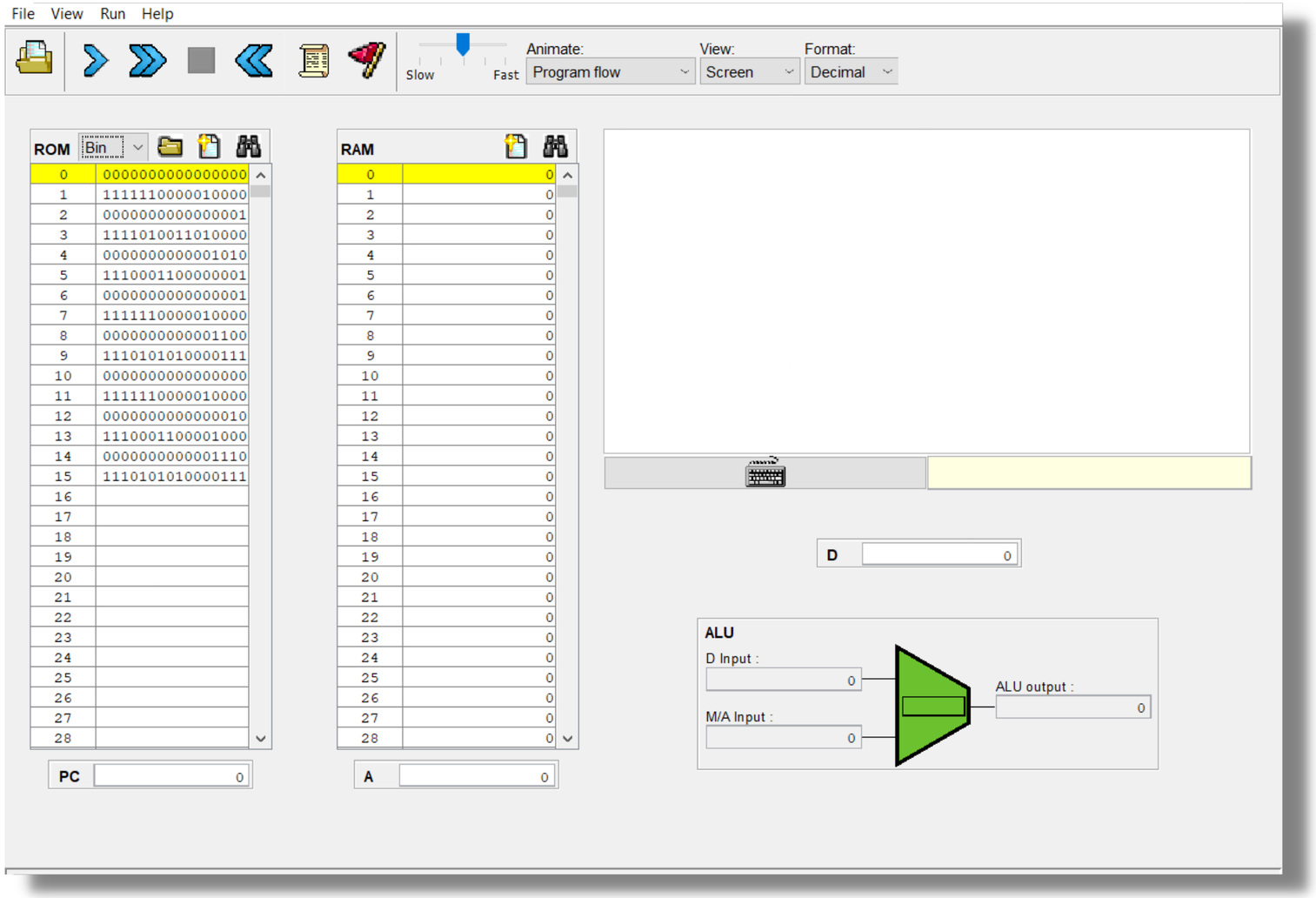
Team Members	Roll Numbers
Kalyana Sundaram	CB.EN.U4AIE21120
Kaushik Jonnada	CB.EN.U4AIE21122
Subikksha	CB.EN.U4AIE21167
Sarvesh Shashikumar	CB.EN.U4AIE21163

1. Locate the **Max.asm**' and **MaxL.asm'** programs in nand2tetris folder and perform the following actions. **[CO2]**
    a. Check the correctness of both the program using the CPU emulator. Comprehend the lines of codes. (You may be asked to explain the lines of codes).
    b. Generate machine code for the **'MaxL.asm'** program manually. Describe/comment on the translation line by line. Save the file as **'MaxL.hack'**
    c. Develop your own assembler (using python/Java) using limited instruction set used in the program **MaxL.asm'**. The assembler you developed should translate the input file **'MaxL.asm'** to **'MaxL.hack'.**
    d. Verify the machine codes generated by your own assembler and the 'assembler' tool provided in the software suite.
    e. Repeat (c)-(d) for **Max.asm** (Optional only, those who are good in programming can attempt it and bonus marks will be given)

a

The Max.asm and MaxL.asm

- These 2 files are stored in the "max" folder of "06", which is stored in the "project" folder of "nand2tetris".
- The Max.asm file is hack assembly code that finds the greater number of 2 given numbers, stored at RAM locations 0 and 1, and prints it in RAM location 3.
- The MaxL.asm too performs the same task, however, this hack assembly code does not use any variables or loops to perform the task. The 'L' stands for "less".
- These files can be executed and tested using the CPU Emulator provided in the software's "tools" folder.
- Both these files run perfectly without errors and perform the task mentioned earlier.



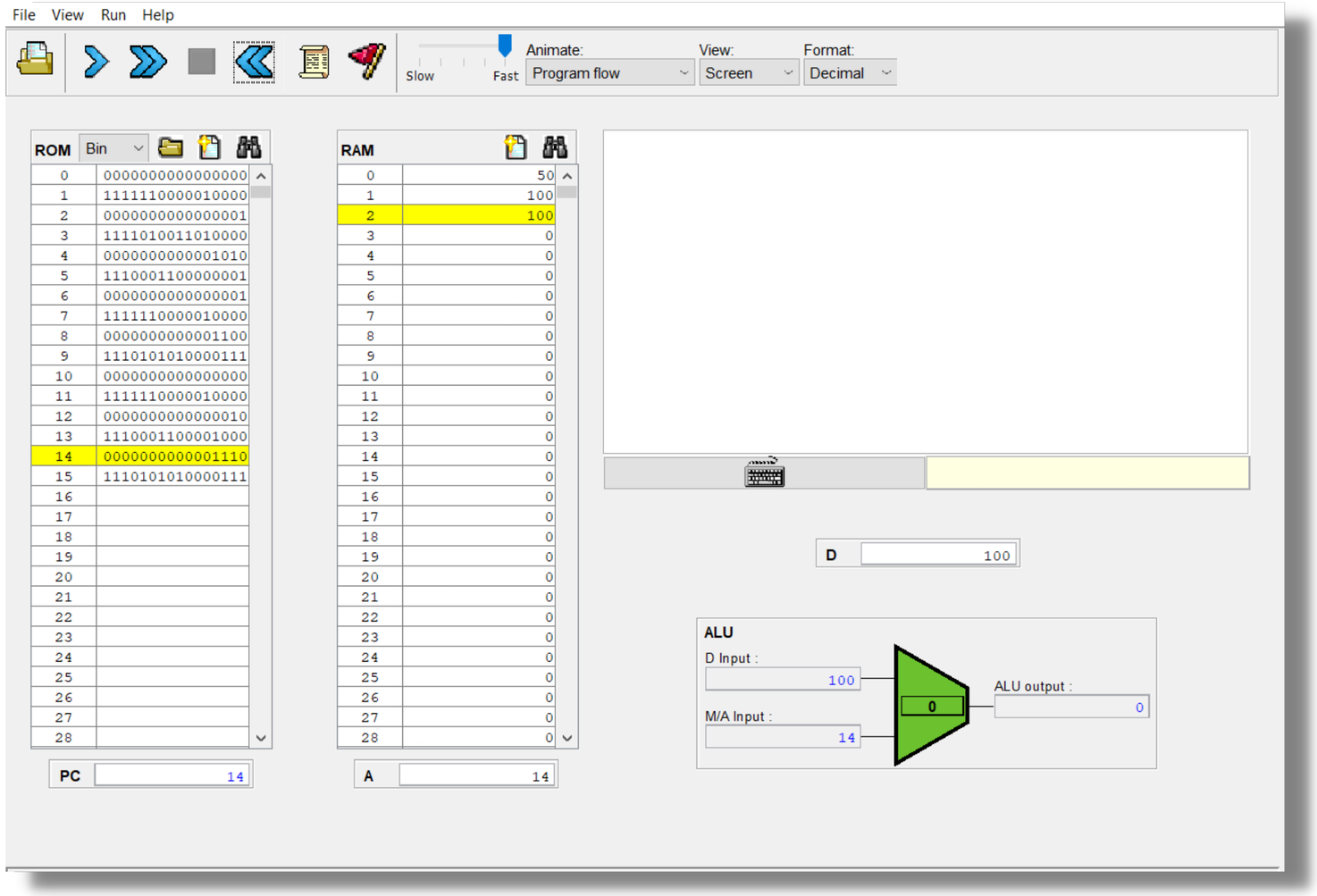
	ROM (Bin)	RAM
0	0000000000000000	0
1	1111110000010000	0
2	0000000000000001	0
3	1111010011010000	0
4	0000000000001010	0
5	1110001100000001	0
6	0000000000000001	0
7	1111110000010000	0
8	0000000000001100	0
9	1110101010000111	0
10	0000000000000000	0
11	1111110000010000	0
12	0000000000000010	0
13	1110001100001000	0
14	0000000000001110	0
15	1110101010000111	0
16		0
17		0
18		0
19		0
20		0
21		0
22		0
23		0
24		0
25		0
26		0
27		0
28		0

PC: 0

A: 0

D: 0

ALU
D Input : 0
M/A Input : 0
ALU output : 0


File View Run Help
Slow
Fast
Animate:
Program flow
View:
Screen
Format:
Decimal
ROM
Bin
RAM
PC
14
A
14
D
100
ALU
D Input :
100
M/A Input :
14
0
ALU output :
0

b

MaxL.asm

- The beside is the machine code for MaxL.asm, and has been saved as MaxL.hack.
- The A-instructions are converted to machine code by setting 0 as the first bit, or MSB, and the rest 15 bits are the binary conversion of the address to binary.
- The C-instructions are converted with the help of the symbol table.

```
1  0000000000000000    // @0
2  1111110000010000    // D=M
3  0000000000000001    // @1
4  1111010011010000    // D=D-M
5  0000000000001010    // @10
6  1110001100000001    // D;JGT
7  0000000000000001    // @1
8  1111110000010000    // D=M
9  0000000000001100    // @12
10 1110101010000111    // 0;JMP
11 0000000000000000    // @0
12 1111110000010000    // D=M
13 0000000000000010    // @2
14 1110001100001000    // M=D
15 0000000000001110    // @14
16 1110101010000111    // 0;JMP
```

C

About Assembler

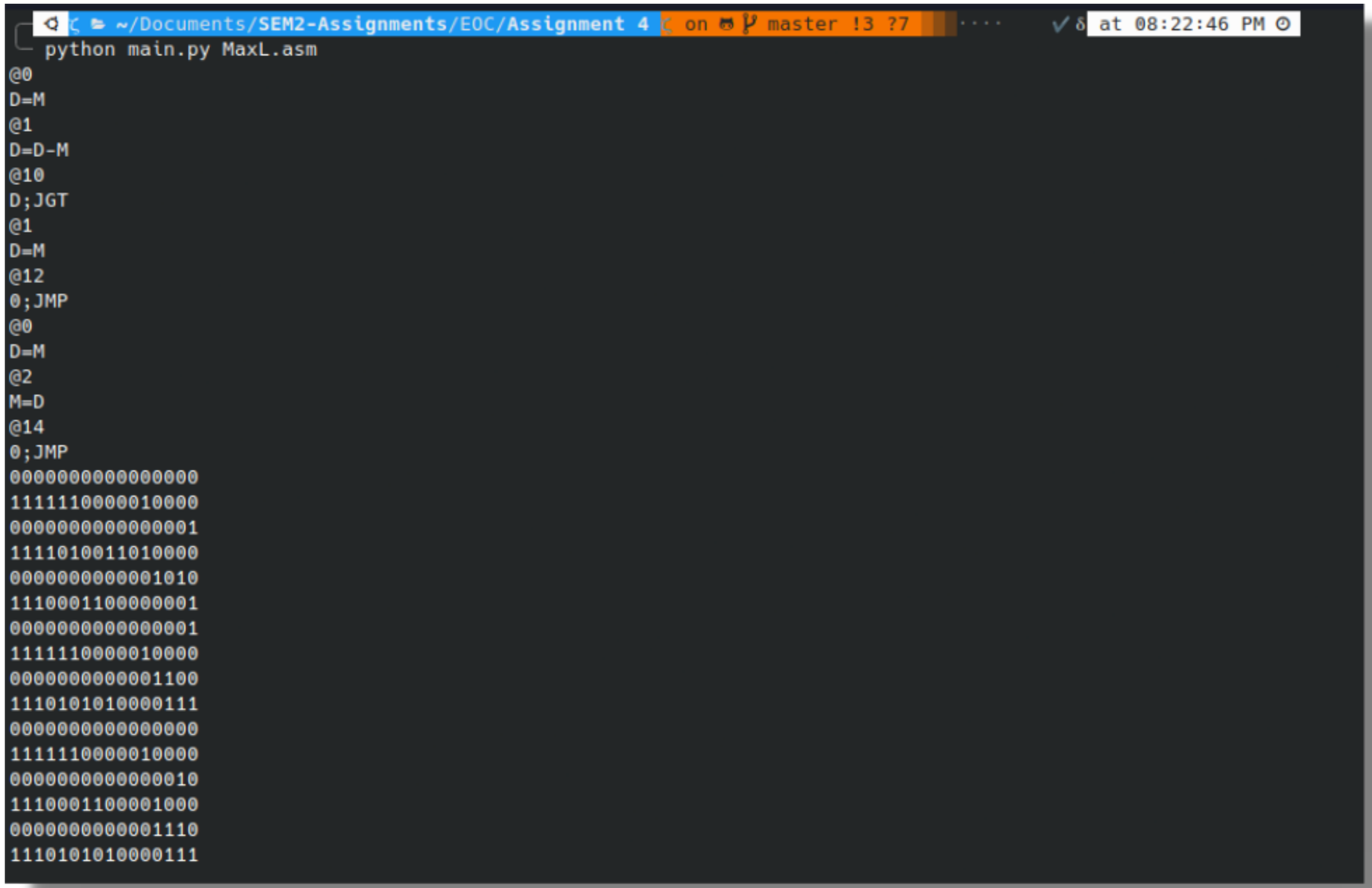
- The assembler converts the symbolic low-level programs to binary code.
- It connects the hardware platform to the software hierarchy that sits on top of it.
- It performs multiple functions like parsing, symbol tables, code generation, etc.

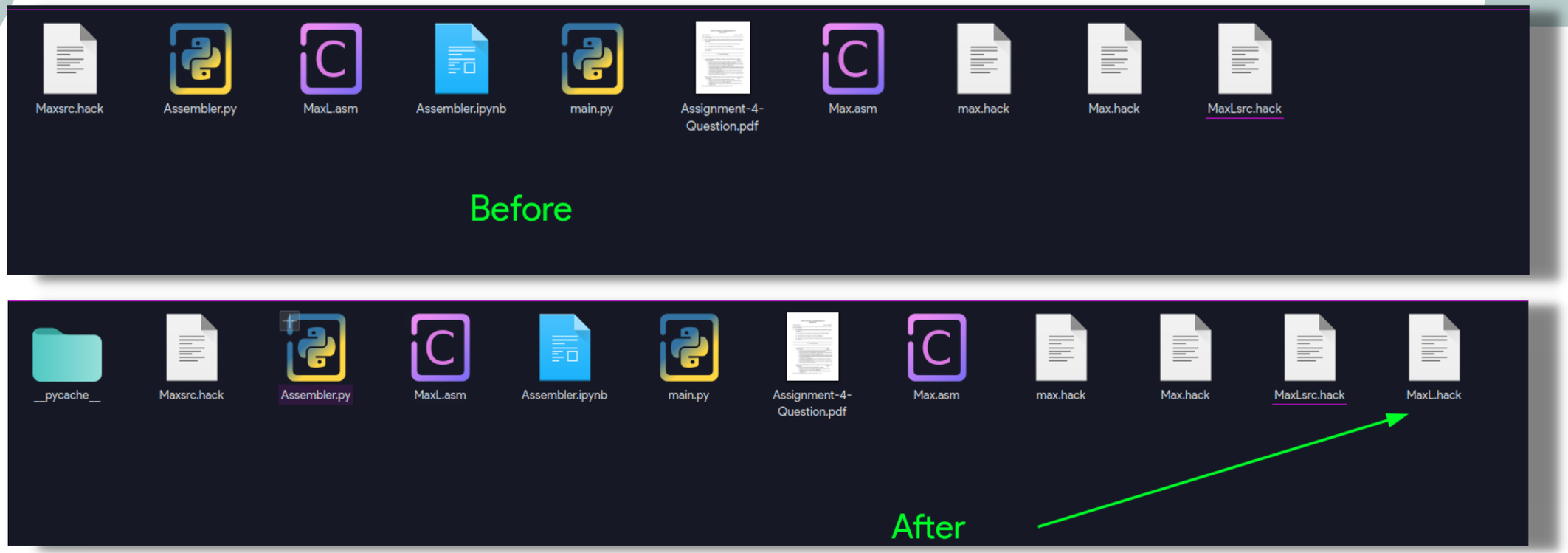
Our Assembler

- Our assembler was made entirely by us from scratch in python. It is a single class, with multiple functions, unlike the usual multiple classes for Assemblers built in other OOP languages.
- It has 8 functions:-
  - ***parser*** - This function removes all whitespaces and comments and identifies if an instruction is A-type or C-type.
  - ***assignVarLoc*** - this function assigns a location to variables, or calls already declared variables.
  - ***translate*** - this function translates A-instructions to binary, and splits C-instructions into computation, destination and jump instructions.
  - ***compFunc*** - this function refers to the symbol tables in the __init__ function of the class, and assigns the appropriate binary code to the computation part of C-instruction

Our Assembler

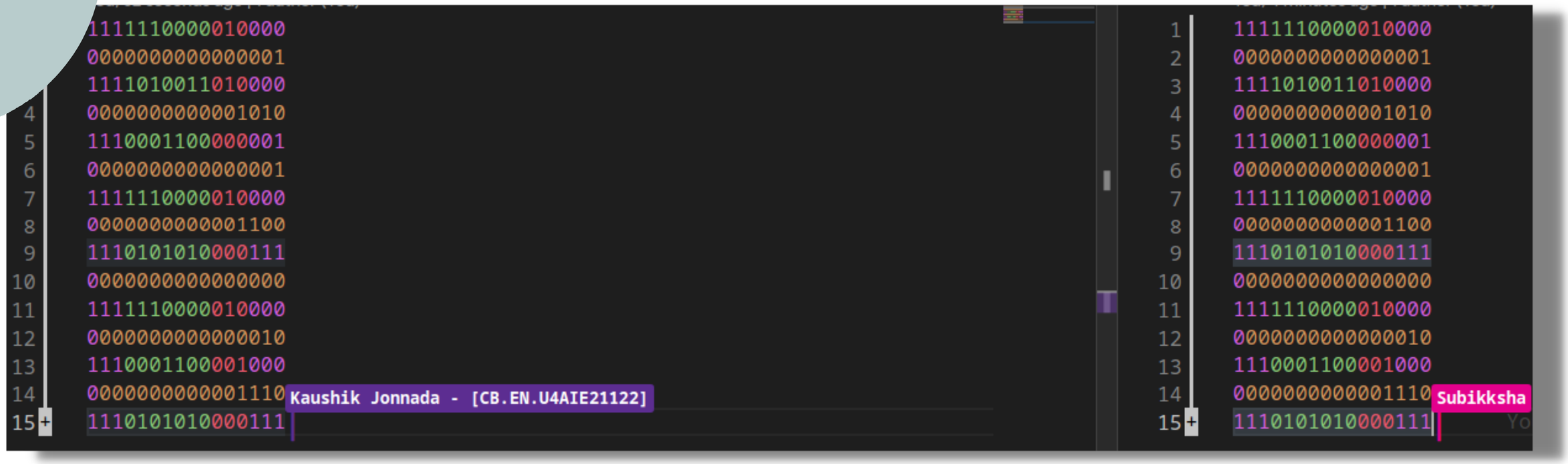
- It has 8 functions:-
  - ***destFunc*** - this function refers to the symbol tables in the __init__ function of the class, and assigns the appropriate binary code to the destination part of C-instruction
  - ***jumpFunc*** - this function refers to the symbol tables in the __init__ function of the class, and assigns the appropriate binary code to the jump part of C-instruction
  - ***setLabelLoc*** - this function assigns locations to the label to which the code must jump to.
  - ***main*** - this is the main function of the class, which calls some of the above mentioned functions as and when required, in the required order.


~/Documents/SEM2-Assignments/EOC/Assignment 4 on master !3 ?7
at 08:22:46 PM
python main.py MaxL.asm
@0
D=M
@1
D=D-M
@10
D;JGT
@1
D=M
@12
0;JMP
@0
D=M
@2
M=D
@14
0;JMP
0000000000000000
1111110000010000
0000000000000001
1111010011010000
0000000000001010
1110001100000001
0000000000000001
1111110000010000
0000000000001100
1110101010000111
0000000000000000
1111110000010000
0000000000000010
1110001100001000
0000000000001110
1110101010000111

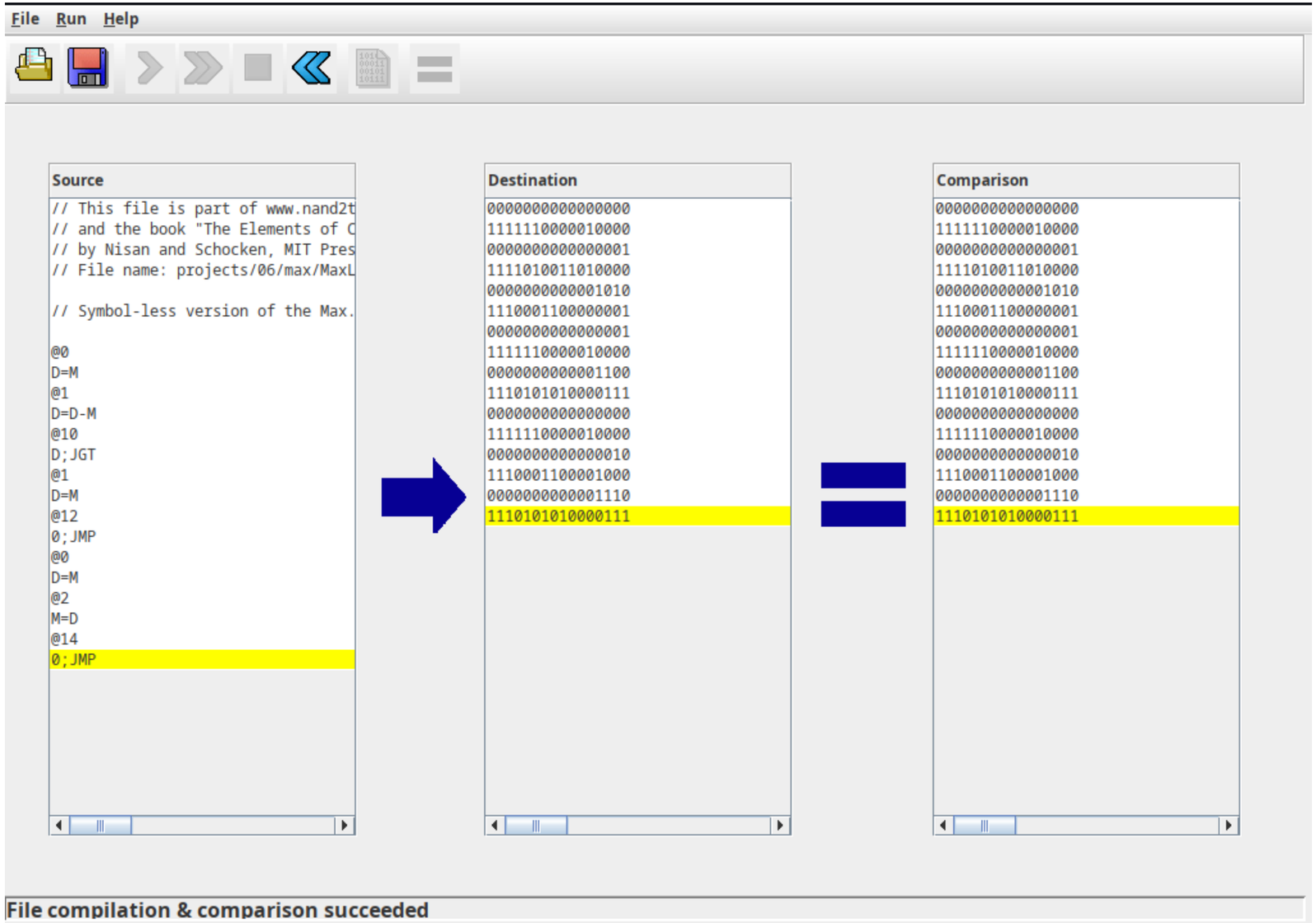

Maxsrc.hack
Assembler.py
MaxL.asm
Assembler.ipynb
main.py
Assignment-4-Question.pdf
Max.asm
max.hack
Max.hack
MaxLsrc.hack
Before
__pycache__
Maxsrc.hack
Assembler.py
MaxL.asm
Assembler.ipynb
main.py
Assignment-4-Question.pdf
Max.asm
max.hack
Max.hack
MaxLsrc.hack
MaxL.hack
After

d

MaxL.hack

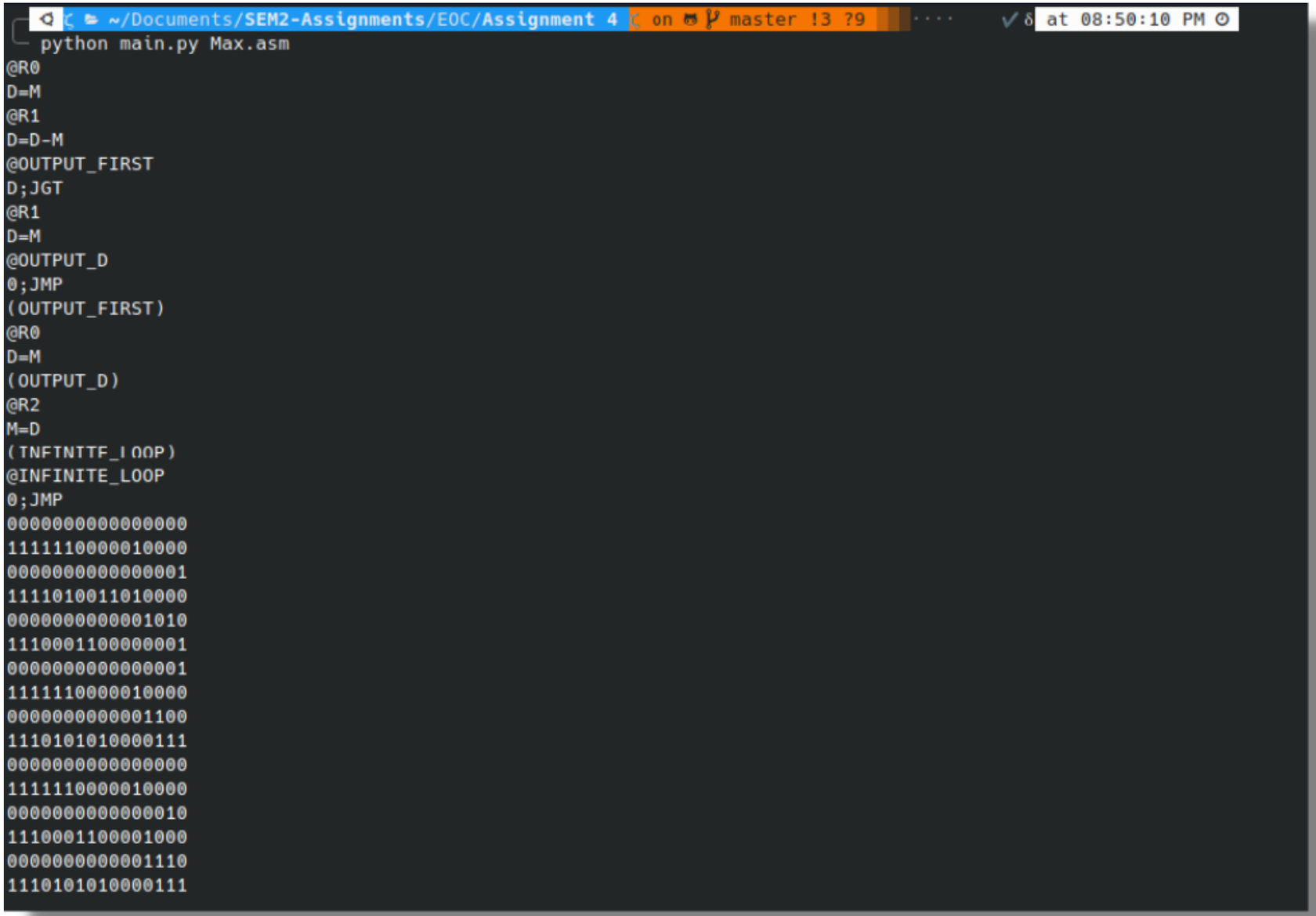

1111110000010000
0000000000000001
1111010011010000
0000000000001010
1110001100000001
0000000000000001
1111110000010000
0000000000001100
1110101010000111
0000000000000000
1111110000010000
0000000000000010
1110001100001000
0000000000001110
1110101010000111
Kaushik Jonnada - [CB.EN.U4AIE21122]
Subikksha

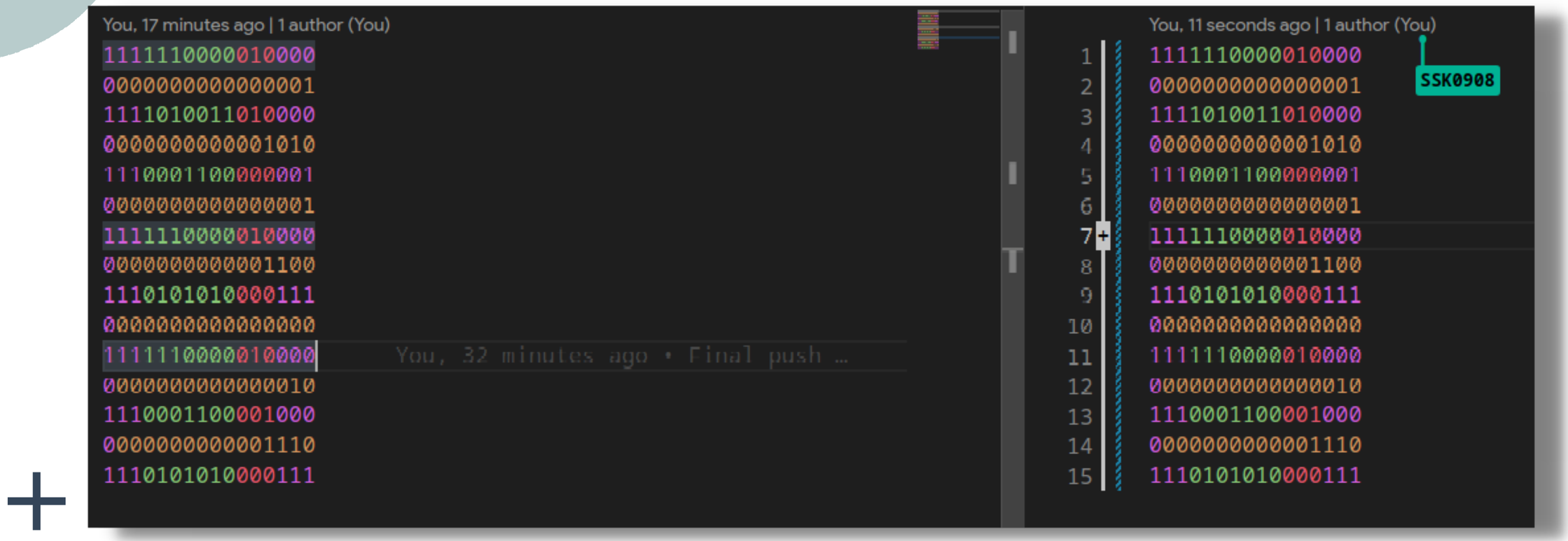
MaxL.hack



Comparing the Assembler.py generated file with inbuilt Assembler

e

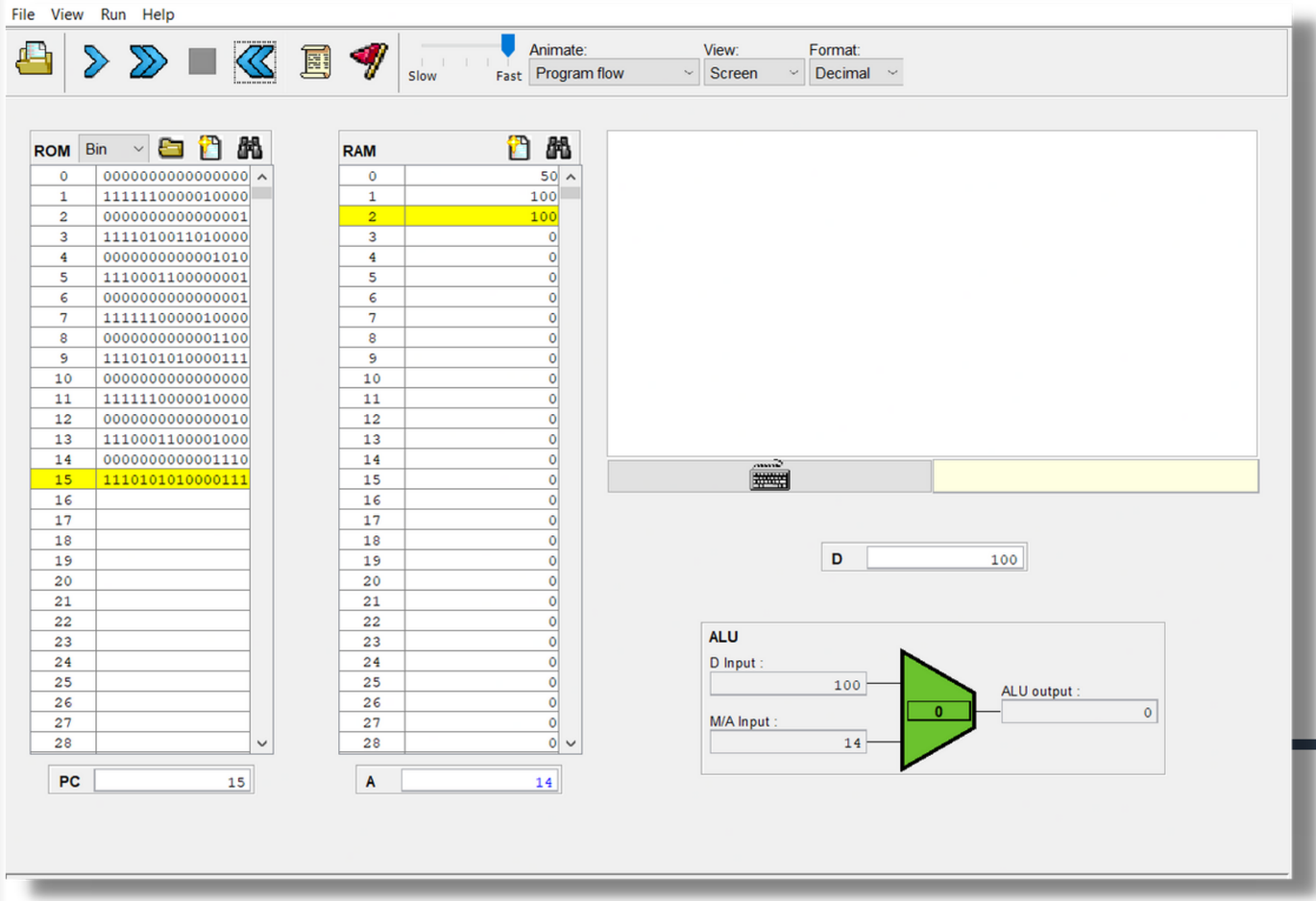

~/Documents/SEM2-Assignments/EOC/Assignment 4 on master !3 ?9
at 08:50:10 PM
python main.py Max.asm
@R0
D=M
@R1
D=D-M
@OUTPUT_FIRST
D;JGT
@R1
D=M
@OUTPUT_D
0;JMP
(OUTPUT_FIRST)
@R0
D=M
(OUTPUT_D)
@R2
M=D
(INFINITE_LOOP)
@INFINITE_LOOP
0;JMP
0000000000000000
1111110000010000
0000000000000001
1111010011010000
0000000000001010
1110001100000001
0000000000000001
1111110000010000
0000000000001100
1110101010000111
0000000000000000
1111110000010000
0000000000000010
1110001100001000
0000000000001110
1110101010000111

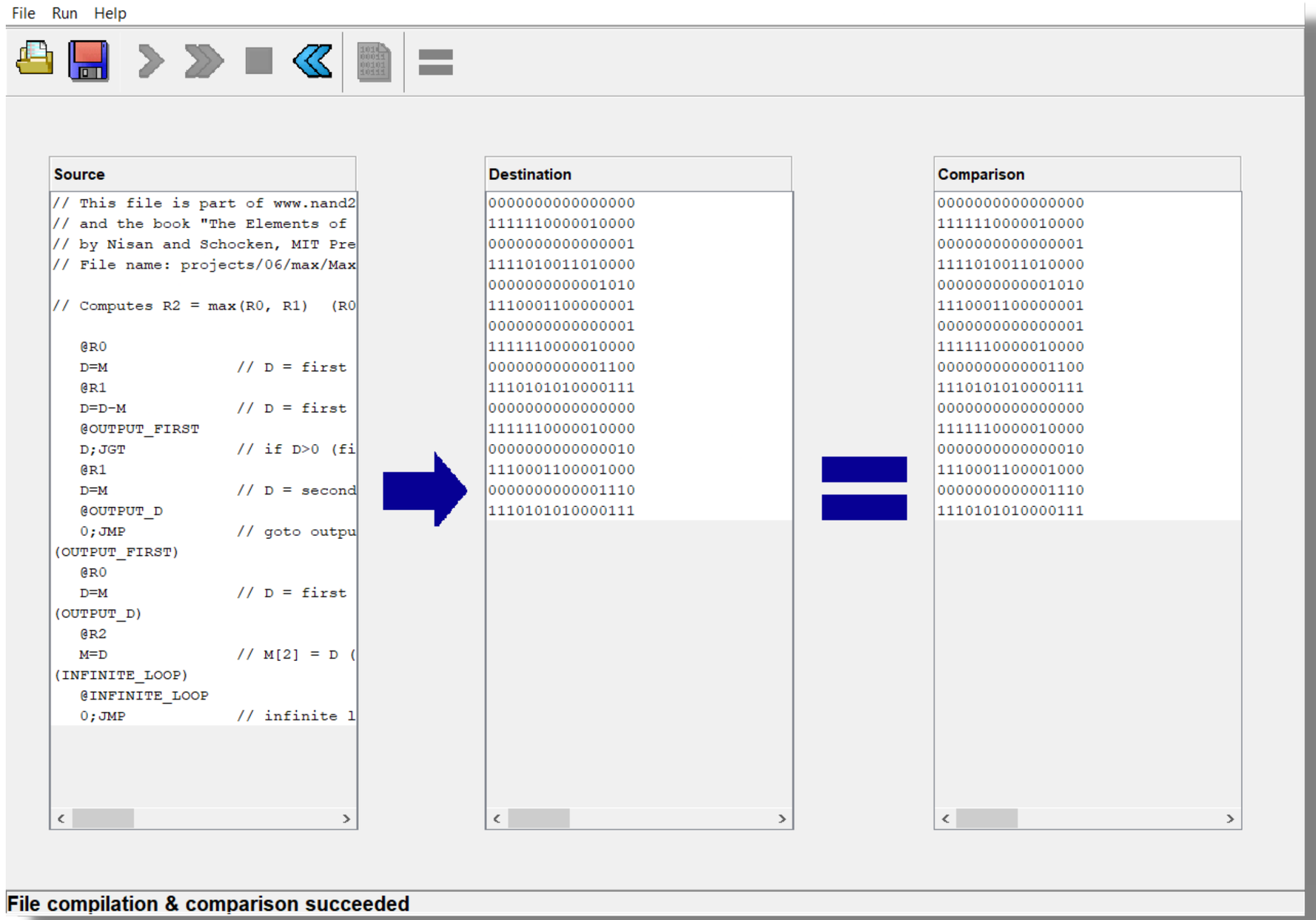


Side by side view of Assembly code
(left) Self Assembler .hack file (Right) Inbuilt Assembler .hack file

Max.hack

```
0000000000000000
1111110000010000
0000000000000001
1111010011010000
0000000000001010
1110001100000001
0000000000000001
1111110000010000
0000000000001100
1110101010000111
0000000000000000
1111110000010000
0000000000000010
1110001100001000
0000000000001110
1110101010000111
```



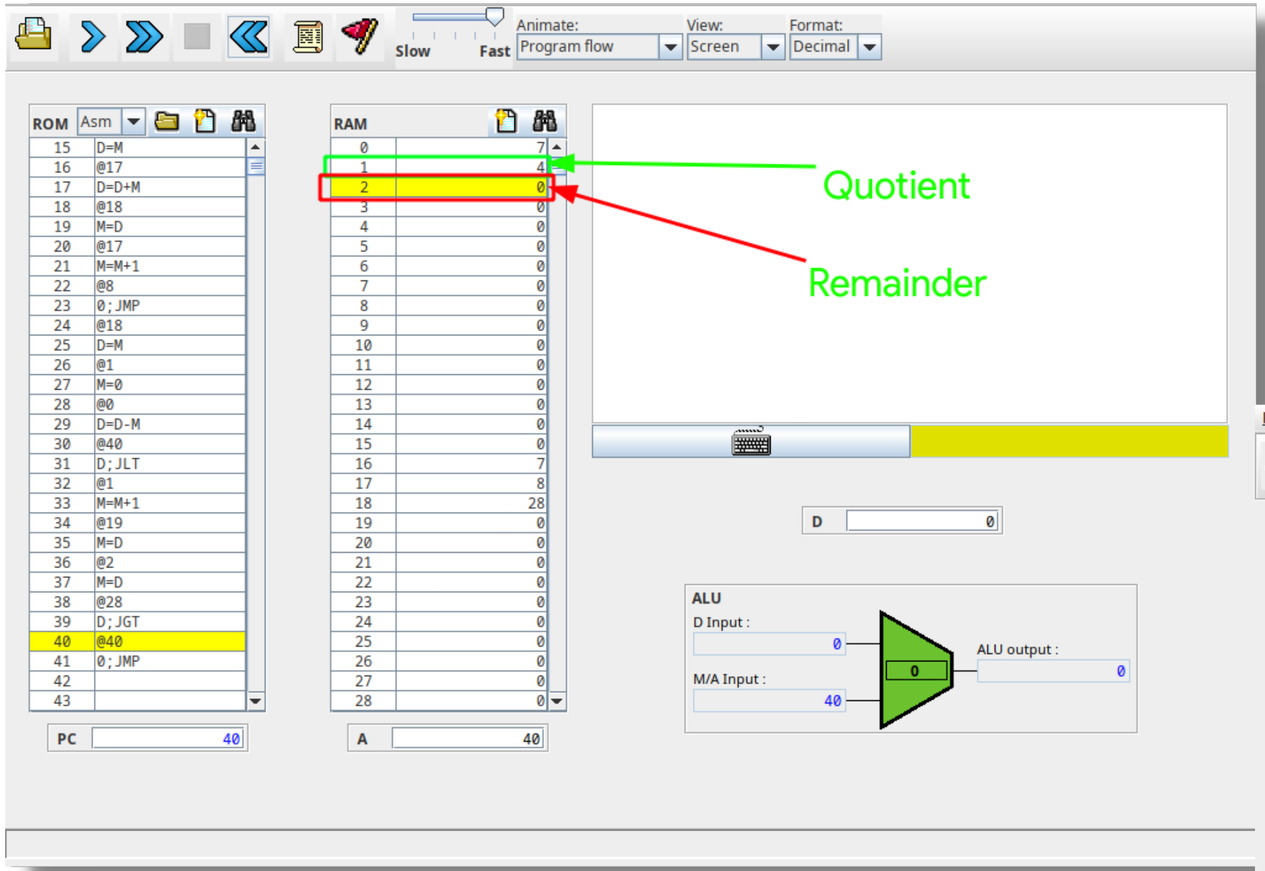


Comparing the Assembler.py generated file with inbuilt Assembler

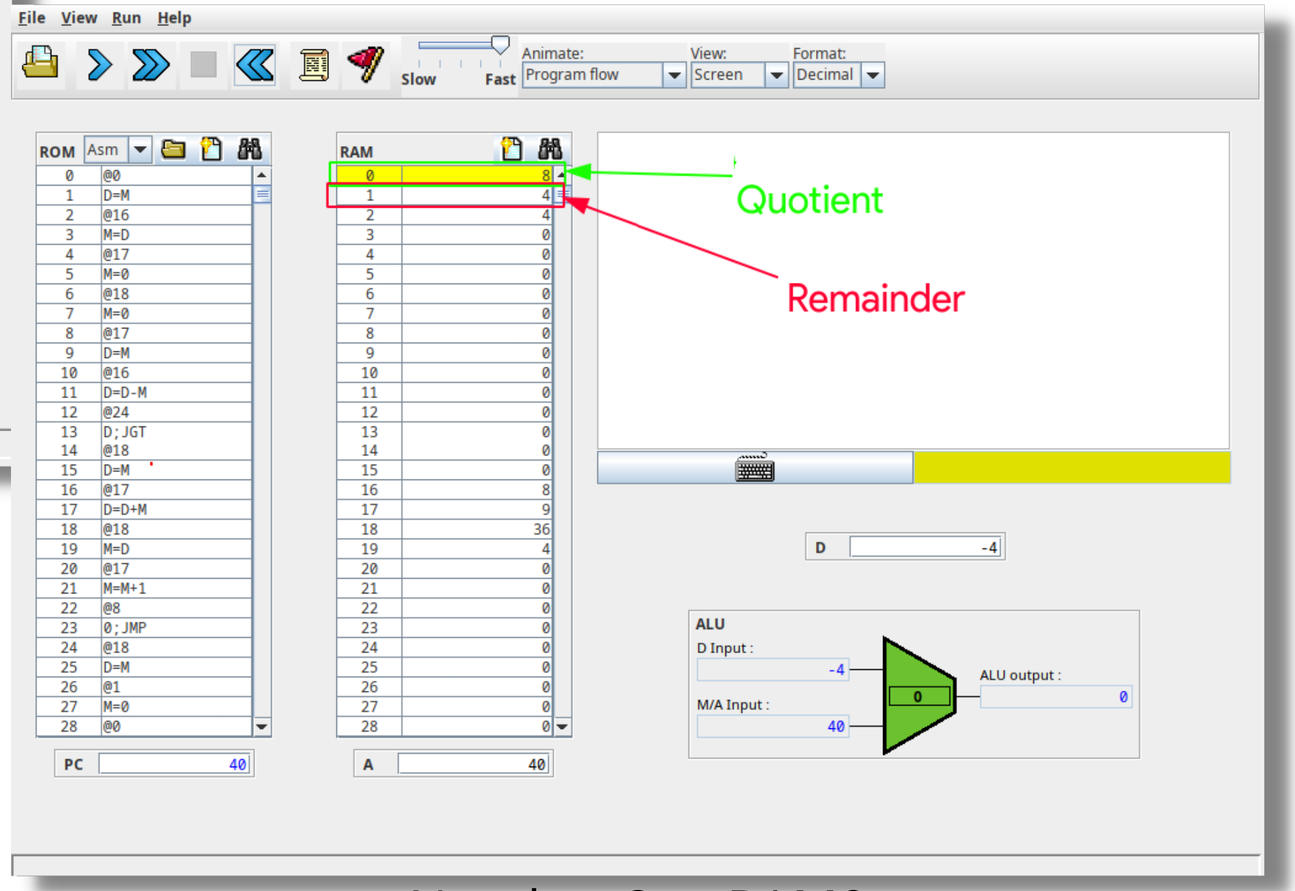
2. Write an assembly language program to find the average of **N** numbers and save the file as **‘average.asm’** **[CO2]**
    a. Check the correctness of the program using the CPU emulator.
    b. Generate machine code for the program manually. Describe/comment on the translation line by line. Save the file as **‘average.hack’**
    c. Compare the machine codes generated manually and the one generated by the ‘assembler’ tool.

SEM2-Assignments -

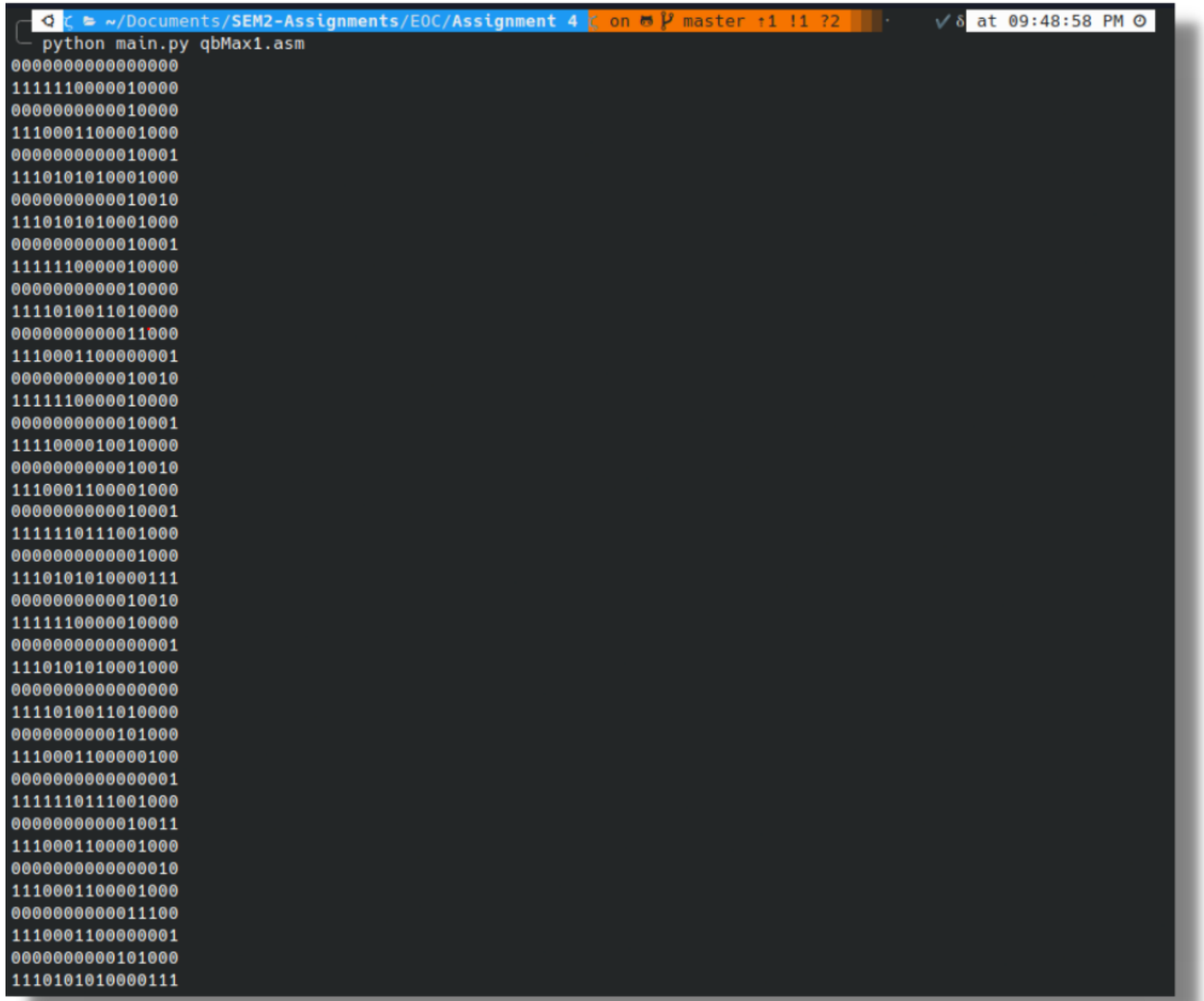
```
@R0
D=M

@n
M=D

//iter=0
@iter
M=0

// initialize sum=0
@sum
M=0

(LOOP)
@iter
D=M

@n
D=D-M

// if i>n goto loop MEAN
@MEAN
D;JGT

@sum
D=M

@iter
D=D+M

@sum
M=D

@iter
M=M+1

@LOOP
0;JMP

(MEAN)
@sum
D=M

@R1
M=0

(MEANLOOP)
@R0
D=D-M

@END
D;JLT

@R1
M=M+1

@Remainder
M=D

@R2
M=D

@MEANLOOP
D;JGT

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



Number 7 at RAM0

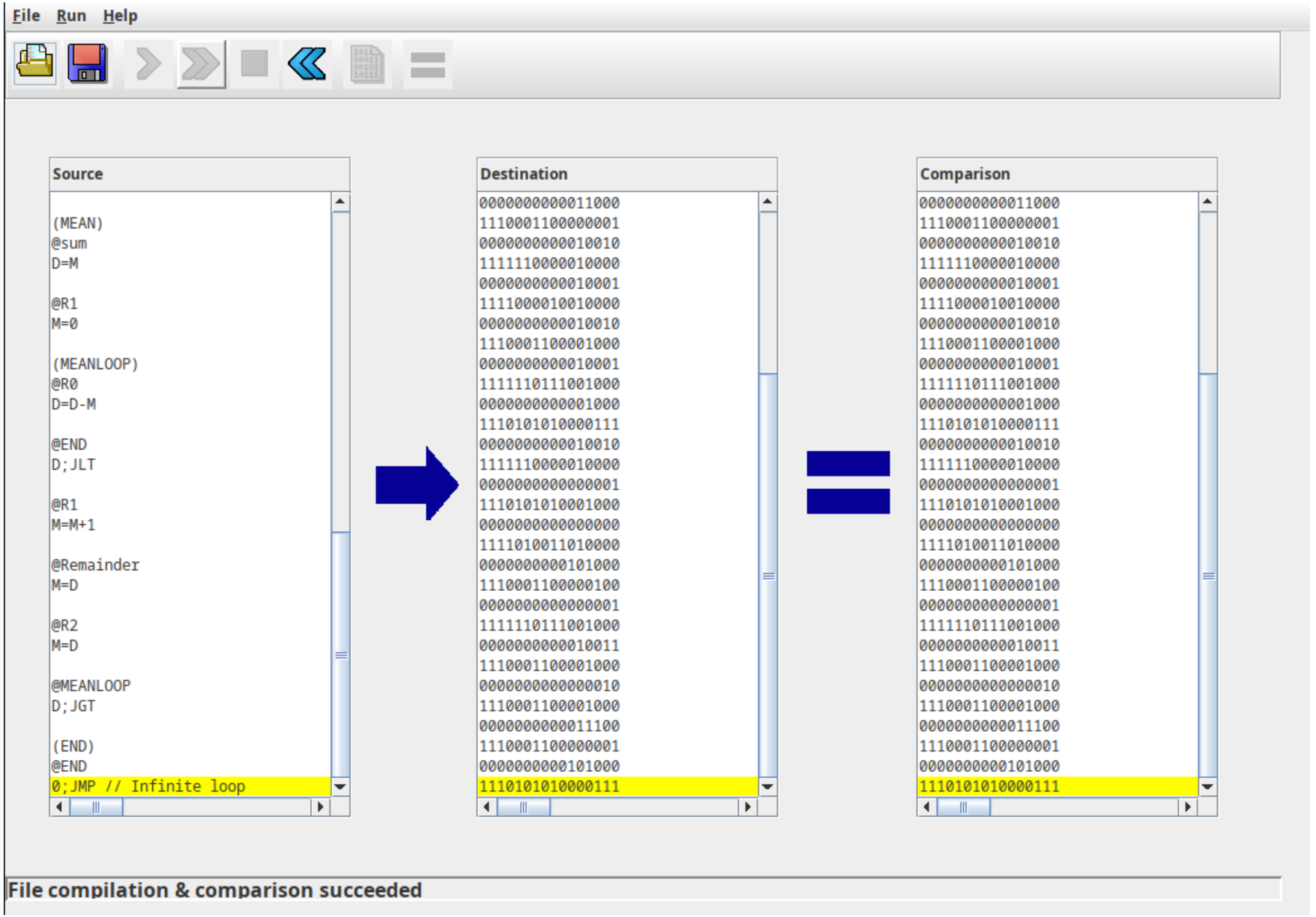


Number 8 at RAM0



.hack file generated using Assembler.py from Question 1

Side by side view of machine code written manually
(left) and Assembler .hack file (Right) from Inbuilt Assembler .hack file



Comparing the manually written file with inbuilt Assembler

Contributions

The contributions of the team members are together with shared files. Each of the team members contibuted in coding, debugging and verifying outputs.

Thank you!