

Elements of Computing Systems - II

Batch B Team 17

Team Members

Kalyana Sundaram

Kaushik Jonnada

Subikksha

Sarvesh Shashikumar

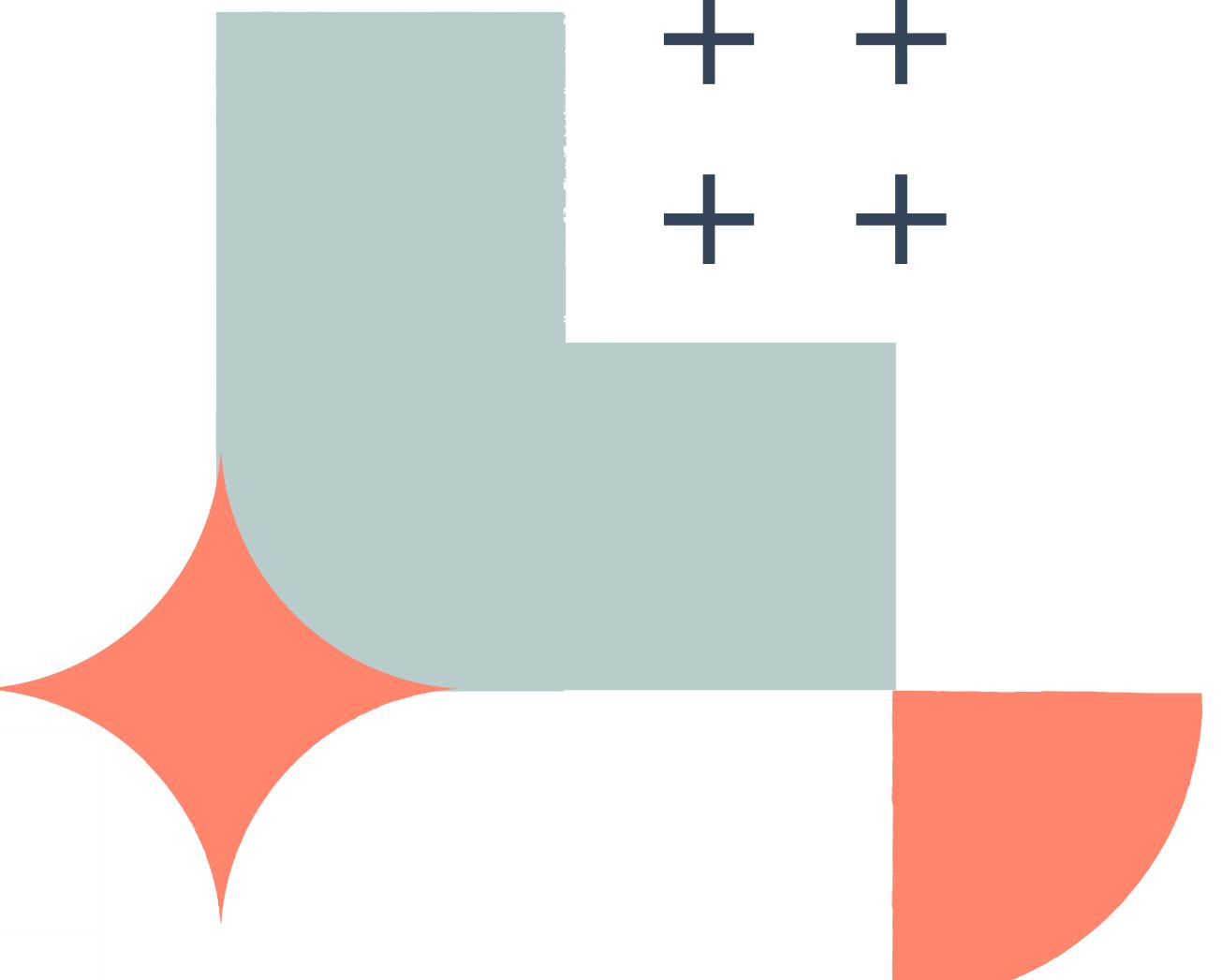
Roll Numbers

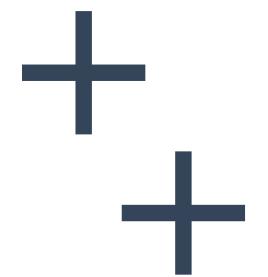
CB.EN.U4AIE21120

CB.EN.U4AIE21122

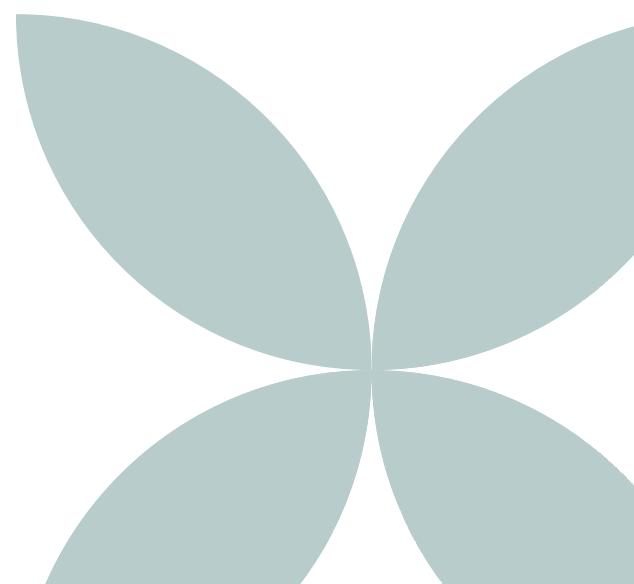
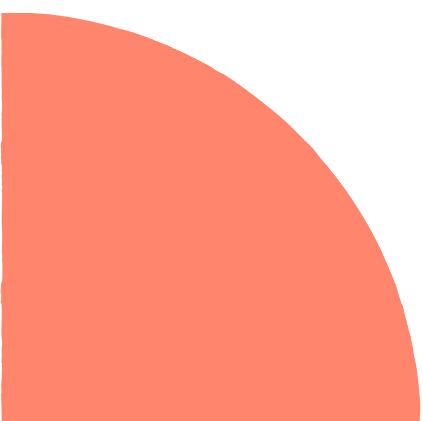
CB.EN.U4AIE21167

CB.EN.U4AIE21163





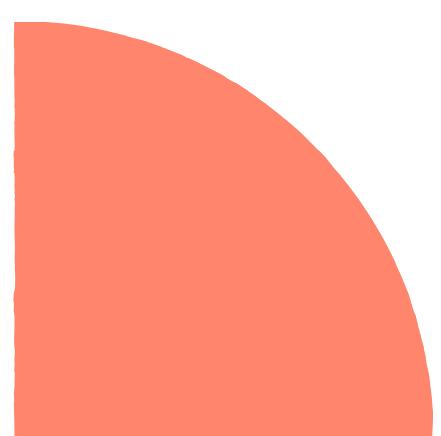
1. Locate the **Max.asm'** and **MaxL.asm'** programs in nand2tetris folder and perform the following actions. [CO2]
 - a. Check the correctness of both the program using the CPU emulator. Comprehend the lines of codes. (You may be asked to explain the lines of codes).
 - b. Generate machine code for 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)

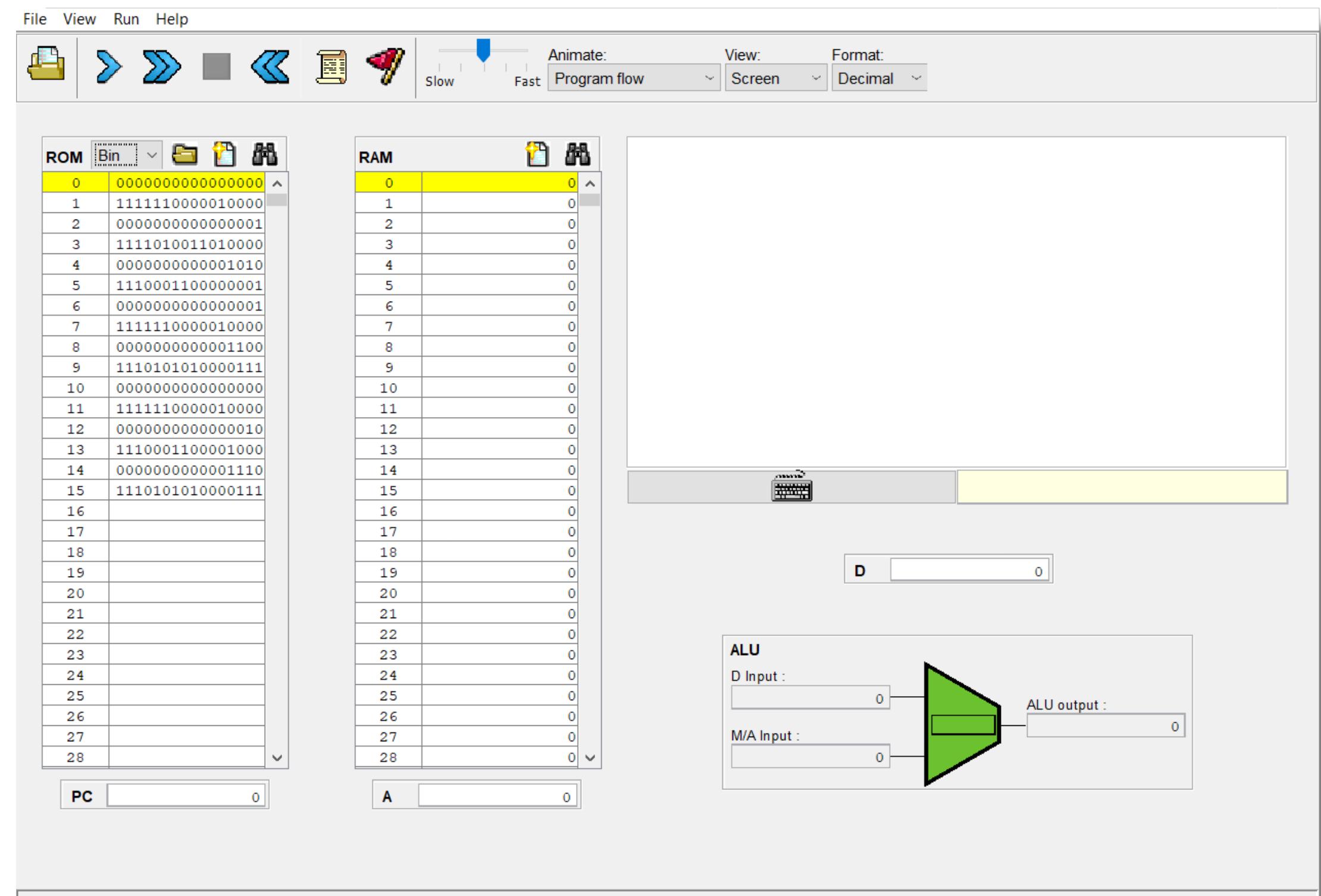


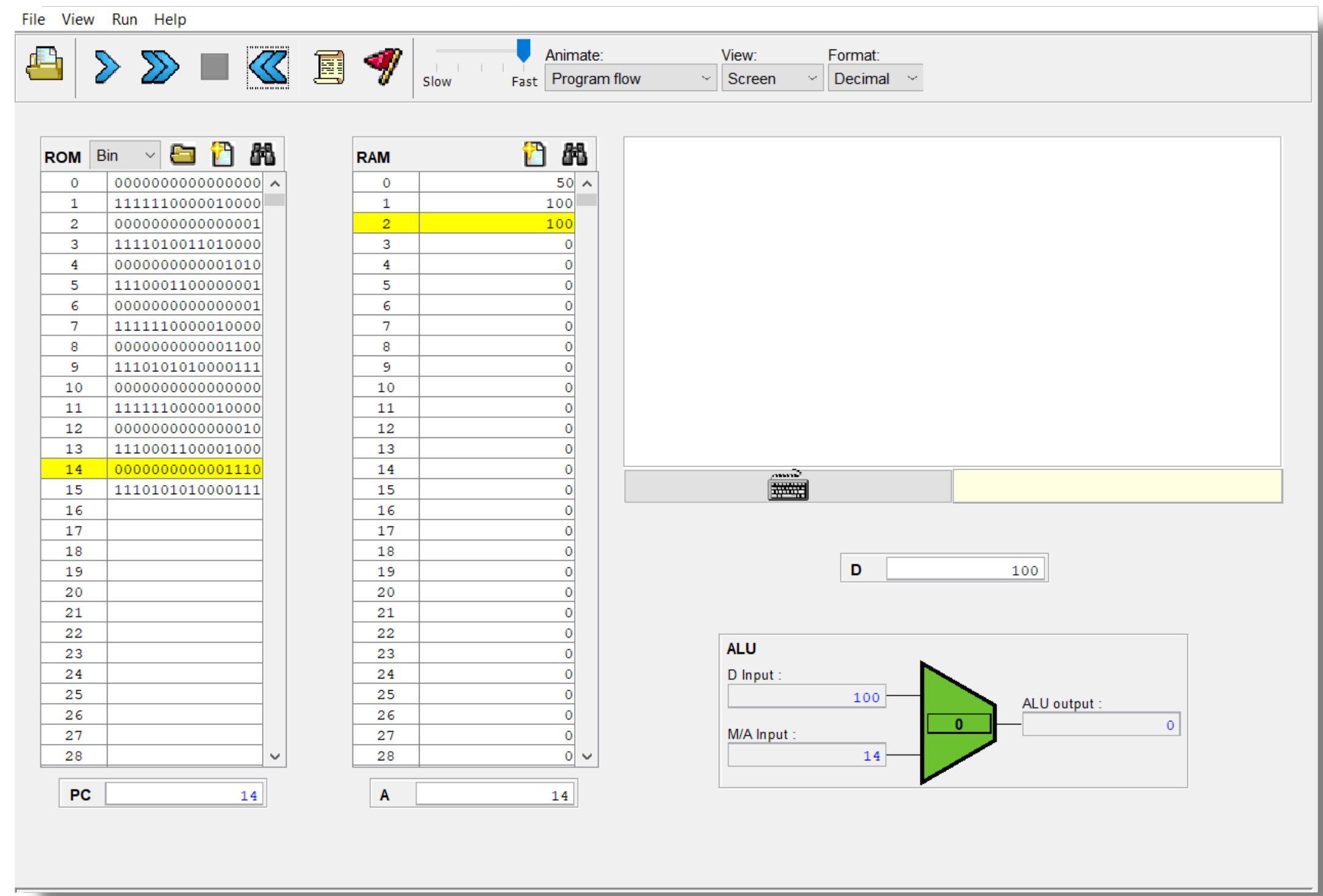
a

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.







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.

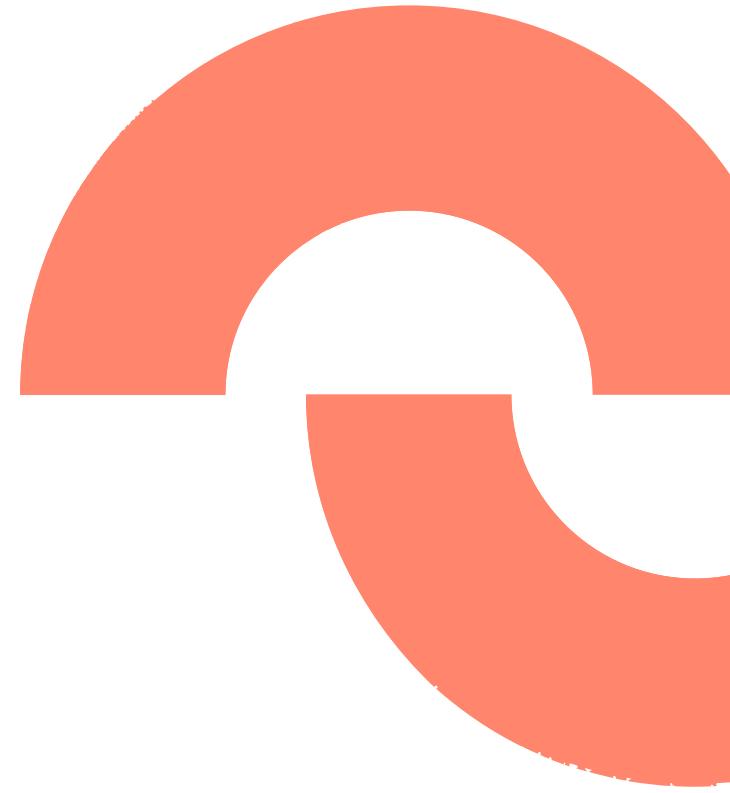
```
1 0000000000000000 // @0
2 1111110000010000 // D=M
3 0000000000000001 // @1
4 1111010011010000 // D=D-M
5 0000000000001010 // @10
6 1110001100000001 // D;JGT
7 0000000000000001 // @1
8 1111110000010000 // D=M
9 0000000000001100 // @12
10 1110101010000111 // 0;JMP
11 0000000000000000 // @0
12 1111110000010000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

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.

+

+



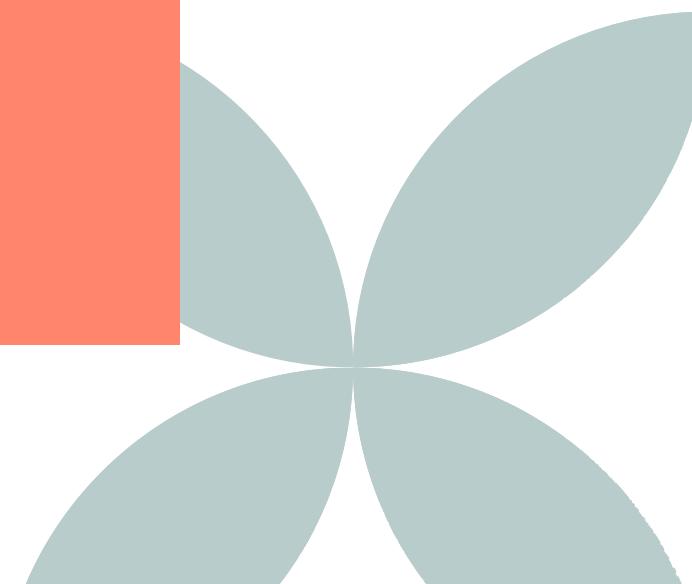
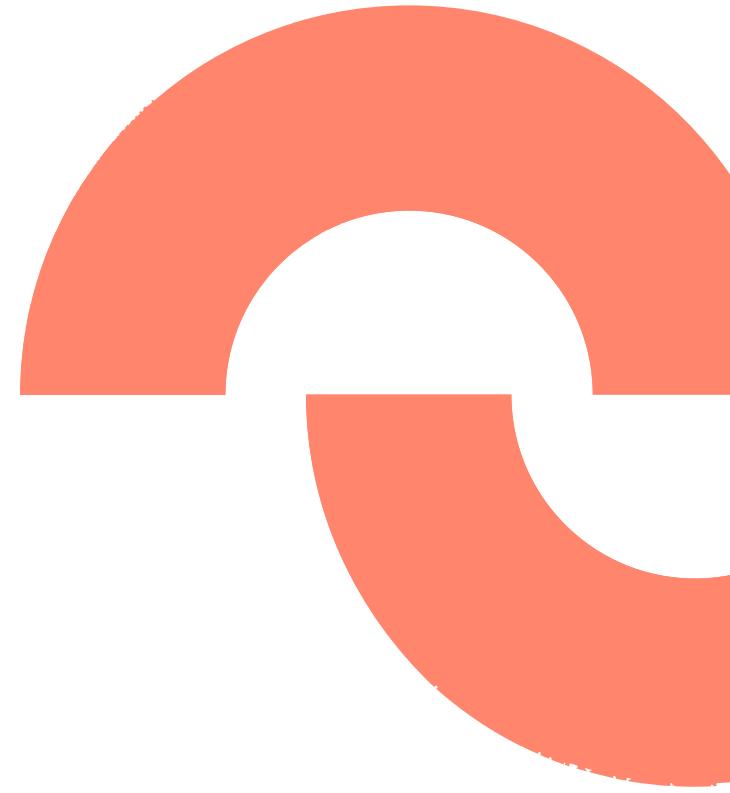
Our Assembler

- 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

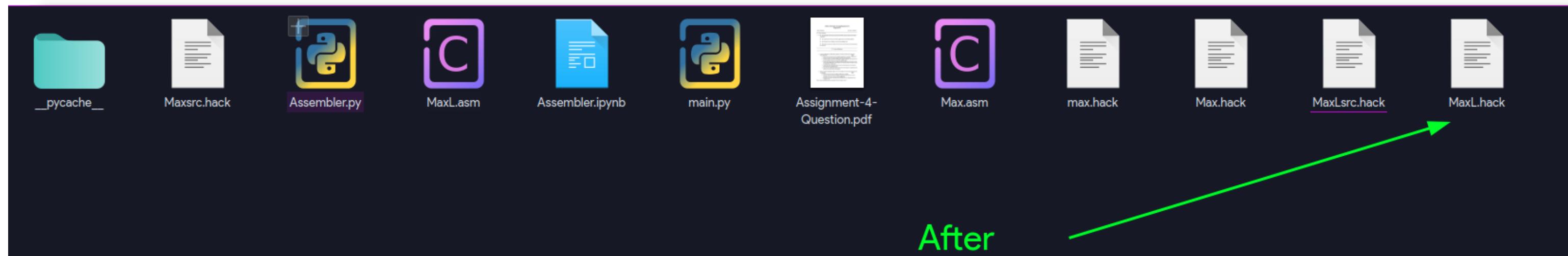
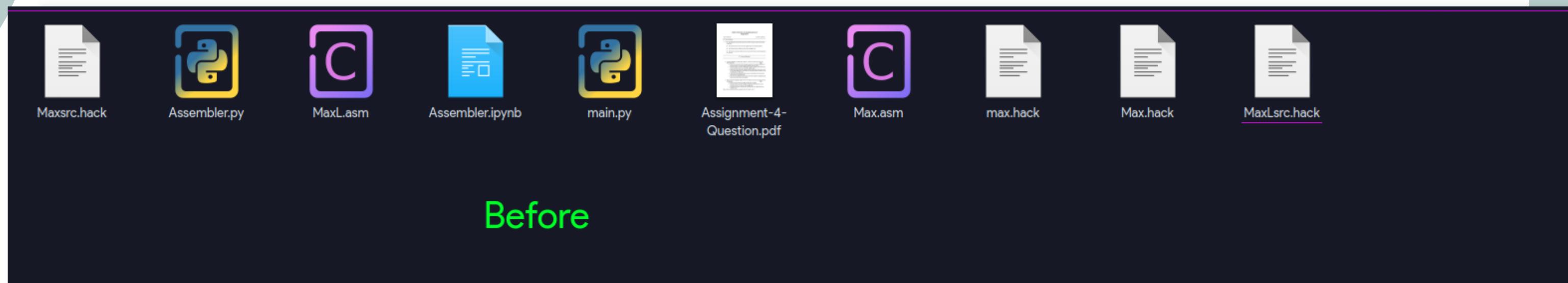


Our Assembler

- 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.



```
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
00000000000000010
1110001100001000
0000000000001110
1110101010000111
```



d

MaxL.hack

```
1111110000010000 1111110000010000
0000000000000001 0000000000000001
1111010011010000 1111010011010000
4 000000000001010 000000000001010
5 1110001100000001 1110001100000001
6 0000000000000001 0000000000000001
7 1111110000010000 1111110000010000
8 000000000001100 000000000001100
9 111010101000111 111010101000111
10 0000000000000000 0000000000000000
11 1111110000010000 1111110000010000
12 0000000000000010 0000000000000010
13 1110001100001000 1110001100001000
14 0000000000001110 0000000000001110
15 + 1110101010001111 1110101010001111
```

Kaushik Jonnada - [CB.EN.U4AIE21122]

Subikksha

Yo

MaxL.hack

The screenshot shows the nand2tflash software interface with three main panes: Source, Destination, and Comparison.

- Source:** Contains assembly-like code. A yellow bar highlights the instruction `0; JMP` at address 14.
- Destination:** Shows the binary representation of the source code. A yellow bar highlights the instruction `1110101010000111` at address 14.
- Comparison:** Shows the expected binary output. A yellow bar highlights the instruction `1110101010000111` at address 14.

A large blue arrow points from the Source pane to the Destination pane, indicating the flow of compilation.

Source	Destination	Comparison
<code>// This file is part of www.nand2tflash.org</code>	<code>0000000000000000</code>	<code>0000000000000000</code>
<code>// and the book "The Elements of Computing Systems"</code>	<code>1111110000010000</code>	<code>1111110000010000</code>
<code>// by Nisan and Schocken, MIT Press</code>	<code>0000000000000001</code>	<code>0000000000000001</code>
<code>// File name: projects/06/max/MaxL</code>	<code>1111010011010000</code>	<code>1111010011010000</code>
<code>// Symbol-less version of the MaxL</code>	<code>0000000000001010</code>	<code>0000000000001010</code>
<code>@0</code>	<code>1110001100000001</code>	<code>1110001100000001</code>
<code>D=M</code>	<code>0000000000001100</code>	<code>0000000000001100</code>
<code>@1</code>	<code>1110101010000111</code>	<code>1110101010000111</code>
<code>D=D-M</code>	<code>0000000000000000</code>	<code>0000000000000000</code>
<code>@10</code>	<code>1111110000010000</code>	<code>1111110000010000</code>
<code>D; JGT</code>	<code>0000000000000010</code>	<code>0000000000000010</code>
<code>@1</code>	<code>1110001100001000</code>	<code>1110001100001000</code>
<code>D=M</code>	<code>0000000000001110</code>	<code>0000000000001110</code>
<code>@12</code>	<code>1110101010000111</code>	<code>1110101010000111</code>
<code>0; JMP</code>		
<code>@0</code>		
<code>D=M</code>		
<code>@2</code>		
<code>M=D</code>		
<code>@14</code>		
<code>0; JMP</code>		

Comparing the Assembler.py generated file with inbuilt Assembler

```
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
(TNFNTNTF_LOOP)
@INFINITE_LOOP
0;JMP
0000000000000000
1111110000010000
0000000000000001
1111010011010000
0000000000001010
1110001100000001
0000000000000001
1111110000010000
0000000000001100
1110101010000111
0000000000000000
1111110000010000
00000000000000010
1110001100001000
0000000000000110
1110101010000111
```

The image shows a GitHub interface comparing two assembly files side-by-side. The left pane displays assembly code from a file pushed 17 minutes ago, and the right pane displays assembly code from a file pushed 11 seconds ago. The right pane includes a green status indicator labeled 'SSK0908'.

Left pane (Pushed 17 minutes ago):

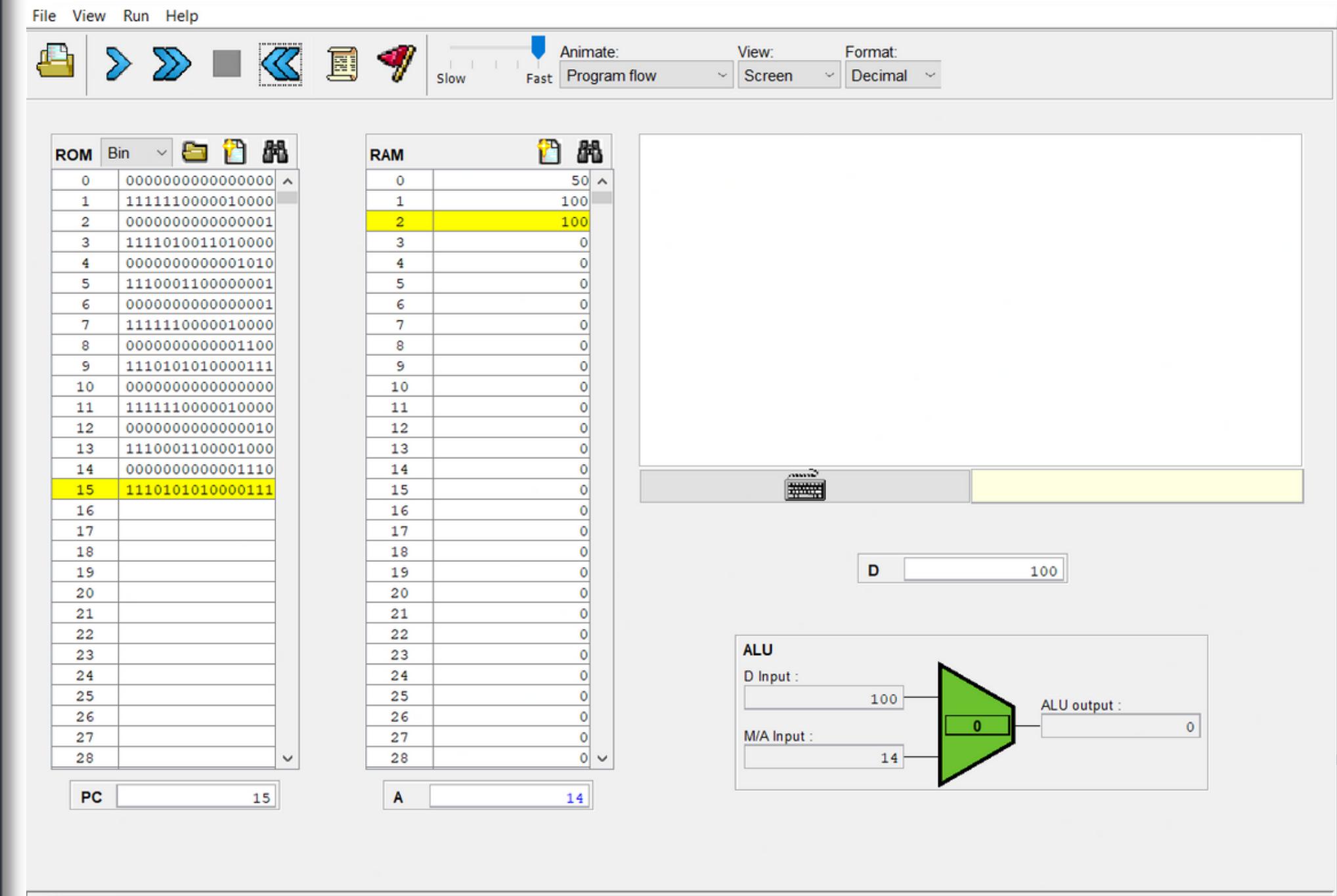
```
You, 17 minutes ago | 1 author (You)
1111110000010000
0000000000000001
1111010011010000
0000000000001010
1110001100000001
0000000000000001
1111110000010000
0000000000001100
1110101010000111
0000000000000000
1111110000010000
0000000000000010
1110001100001000
0000000000001110
1110101010000111
```

Right pane (Pushed 11 seconds ago):

```
You, 11 seconds ago | 1 author (You)
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



File Run Help

Source

```
// This file is part of www.nand2  
// and the book "The Elements of  
// by Nisan and Schocken, MIT Pre  
// File name: projects/06/max/Max  
  
// Computes R2 = max(R0, R1)  (R0  
  
    @R0  
    D=M          // D = first  
    @R1  
    D=D-M        // D = first  
    @OUTPUT_FIRST  
    D;JGT         // if D>0 (fi  
    @R1  
    D=M          // D = second  
    @OUTPUT_D  
    0;JMP         // goto output  
(OUTPUT_FIRST)  
    @R0  
    D=M          // D = first  
(OUTPUT_D)  
    @R2  
    M=D          // M[2] = D (infinite loop  
(INFINITE_LOOP)  
    @INFINITE_LOOP  
    0;JMP         // infinite loop
```

Destination

```
0000000000000000  
1111110000010000  
0000000000000001  
1111010011010000  
00000000000001010  
1110001100000001  
0000000000000001  
1111110000010000  
00000000000001100  
1110101010000111  
00000000000000000  
1111110000010000  
00000000000000010  
1110001100001000  
00000000000001110  
1110101010000111
```

Comparison

```
0000000000000000  
1111110000010000  
0000000000000001  
1111010011010000  
00000000000001010  
1110001100000001  
0000000000000001  
1111110000010000  
00000000000001100  
1110101010000111  
00000000000000000  
1111110000010000  
00000000000000010  
1110001100001000  
00000000000001110  
1110101010000111
```





File compilation & comparison succeeded

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 -

```
1  @R0
2  D=M
3
4  @n
5  M=D
6
7  //iter=0
8  @iter
9  M=0
10
11 // initialize sum=0
12 @sum
13 M=0
14
15 (LOOP)
16 @iter
17 D=M
18
19 @n
20 D=D-M
21
22 // if i>n goto loop MEAN
23 @MEAN
24 D;JGT
25
26 @sum
27 D=M
28
29 @iter
30 D=D+M
31
32 @sum
33 M=D
34
```

```
35 @iter
36 M=M+1
37
38 @LOOP
39 0; JMP
40
41 (MEAN)
42 @sum
43 D=M
44
45 @R1
46 M=0
47
48 (MEANLOOP)
49 @R0
50 D=D-M
51
52 @END
53 D; JLT
54
55 @R1
56 M=M+1
57
58 @Remainder
59 M=D
60
61 @R2
62 M=D
63
64 @MEANLOOP
65 D; JGT
66
67 (END)
68 @END
69 0; JMP // Infinite loop
70
```

ROM Asm

15	D=M
16	@17
17	D=D+M
18	@18
19	M=D
20	@17
21	M=M+1
22	@8
23	0;JMP
24	@18
25	D=M
26	@1
27	M=0
28	@0
29	D=D-M
30	@40
31	D;JLT
32	@1
33	M=M+1
34	@19
35	M=D
36	@2
37	M=D
38	@28
39	D;JGT
40	@40
41	0;JMP
42	
43	

RAM

0	7
1	4
2	0
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	8
18	28
19	0
20	0
21	0
22	0
23	0
24	0
25	0
26	0
27	0
28	0

ALU

D Input: 0 M/A Input: 40 ALU output: 0

D: 0 A: 40

PC: 40

Number 7 at RAM0

ROM Asm

0	@0
1	D=M
2	@16
3	M=D
4	@17
5	M=0
6	@18
7	M=0
8	@17
9	D=M
10	@16
11	D=D-M
12	@24
13	D;JGT
14	@18
15	D=M
16	@17
17	D=D+M
18	@18
19	M=D
20	@17
21	M=M+1
22	@8
23	0;JMP
24	@18
25	D=M
26	@1
27	M=0
28	@0

RAM

0	8
1	4
2	0
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	8
17	9
18	36
19	4
20	0
21	0
22	0
23	0
24	0
25	0
26	0
27	0
28	0

ALU

D Input: -4 M/A Input: 40 ALU output: 0

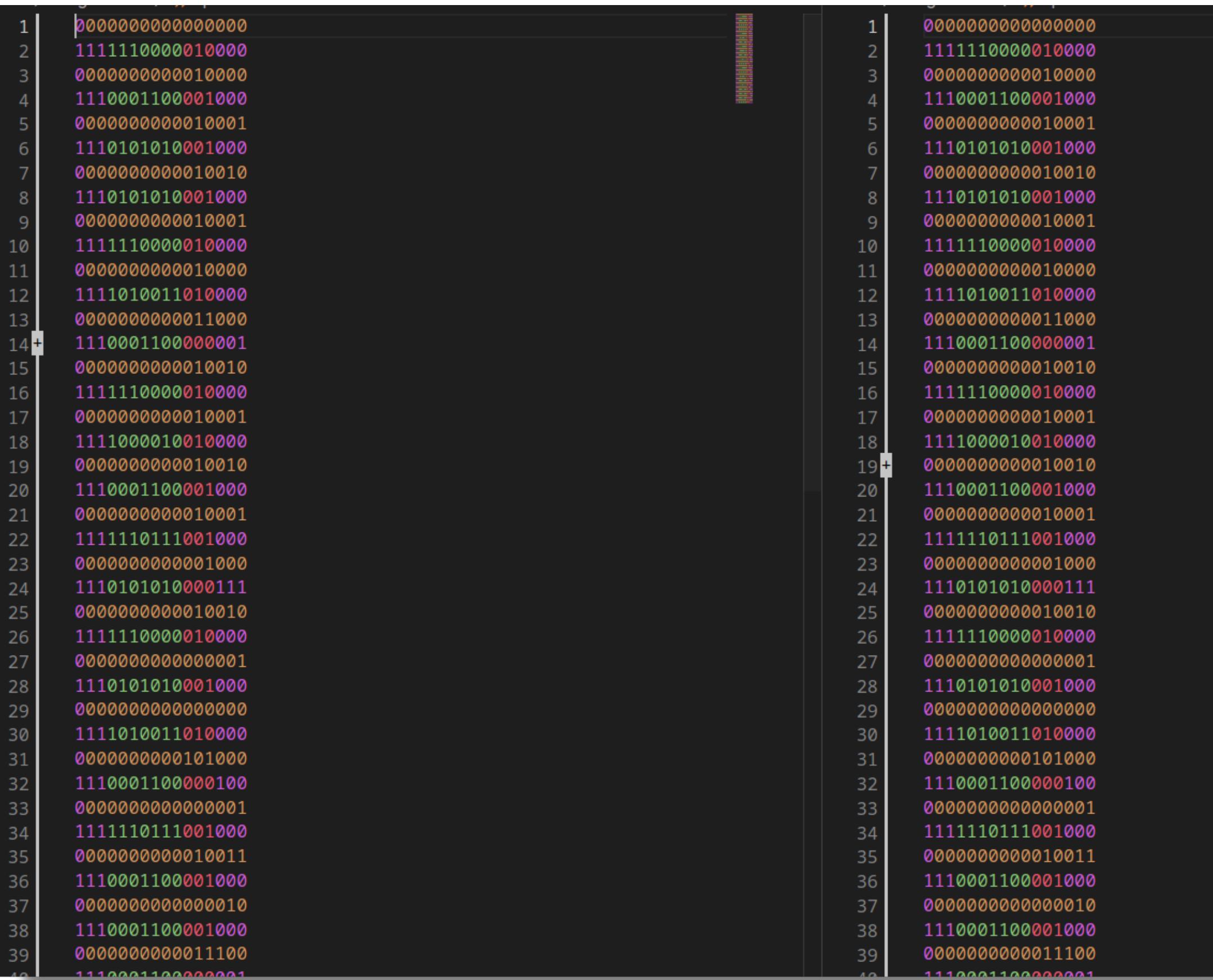
D: -4 A: 40

PC: 40

Number 8 at RAM0

```
python main.py qbMax1.asm
0000000000000000
1111110000010000
0000000000010000
1110001100001000
0000000000010001
1110101010001000
0000000000010010
1110101010001000
0000000000010001
1111110000010000
0000000000010000
1111010011010000
0000000000011000
1110001100000001
0000000000010010
1111110000010000
0000000000010001
1111000010010000
0000000000010010
1110001100001000
0000000000010001
1111110111001000
0000000000001000
1110101010000111
0000000000010010
1111110000010000
0000000000000001
1110101010001000
0000000000000000
1111010011010000
000000000000101000
1110001100000100
0000000000000001
1111110111001000
0000000000010011
1110001100001000
0000000000000000
1110001100001000
00000000000011100
1110001100000001
00000000000101000
1110101010000111
```

.hack file generated using Assembler.py from Question 1



```
1 0000000000000000
2 1111110000010000
3 000000000010000
4 1110001100001000
5 000000000010001
6 1110101010001000
7 000000000010010
8 1110101010001000
9 000000000010001
10 1111110000010000
11 000000000010000
12 111010011010000
13 000000000011000
14 + 1110001100000001
15 0000000000010010
16 1111110000010000
17 000000000010001
18 1111000010010000
19 000000000010010
20 1110001100001000
21 0000000000010001
22 1111110111001000
23 000000000001000
24 111010101000111
25 0000000000010010
26 1111110000010000
27 0000000000000001
28 1110101010001000
29 0000000000000000
30 1111010011010000
31 0000000000101000
32 1110001100000001
33 0000000000000001
34 1111110111001000
35 0000000000010011
36 1110001100001000
37 0000000000000010
38 1110001100001000
39 000000000011100
40 1110001100000001
```

```
1 0000000000000000
2 1111110000010000
3 000000000010000
4 1110001100001000
5 000000000010001
6 1110101010001000
7 000000000010010
8 1110101010001000
9 000000000010001
10 1111110000010000
11 000000000010000
12 111010011010000
13 000000000011000
14 + 1110001100000001
15 0000000000010010
16 1111110000010000
17 000000000010001
18 1111000010010000
19 + 0000000000010010
20 1110001100001000
21 0000000000010001
22 1111110111001000
23 0000000000001000
24 111010101000111
25 0000000000010010
26 1111110000010000
27 0000000000000001
28 1110101010001000
29 0000000000000000
30 1111010011010000
31 0000000000101000
32 1110001100000001
33 0000000000000001
34 1111110111001000
35 0000000000010011
36 1110001100001000
37 0000000000000010
38 1110001100001000
39 000000000011100
40 1110001100000001
```

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

The figure shows a software interface with three main panes: Source, Destination, and Comparison. A large blue arrow points from the Source pane to the Destination pane, indicating a transformation or comparison process.

Source:

```
(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
```

Destination:

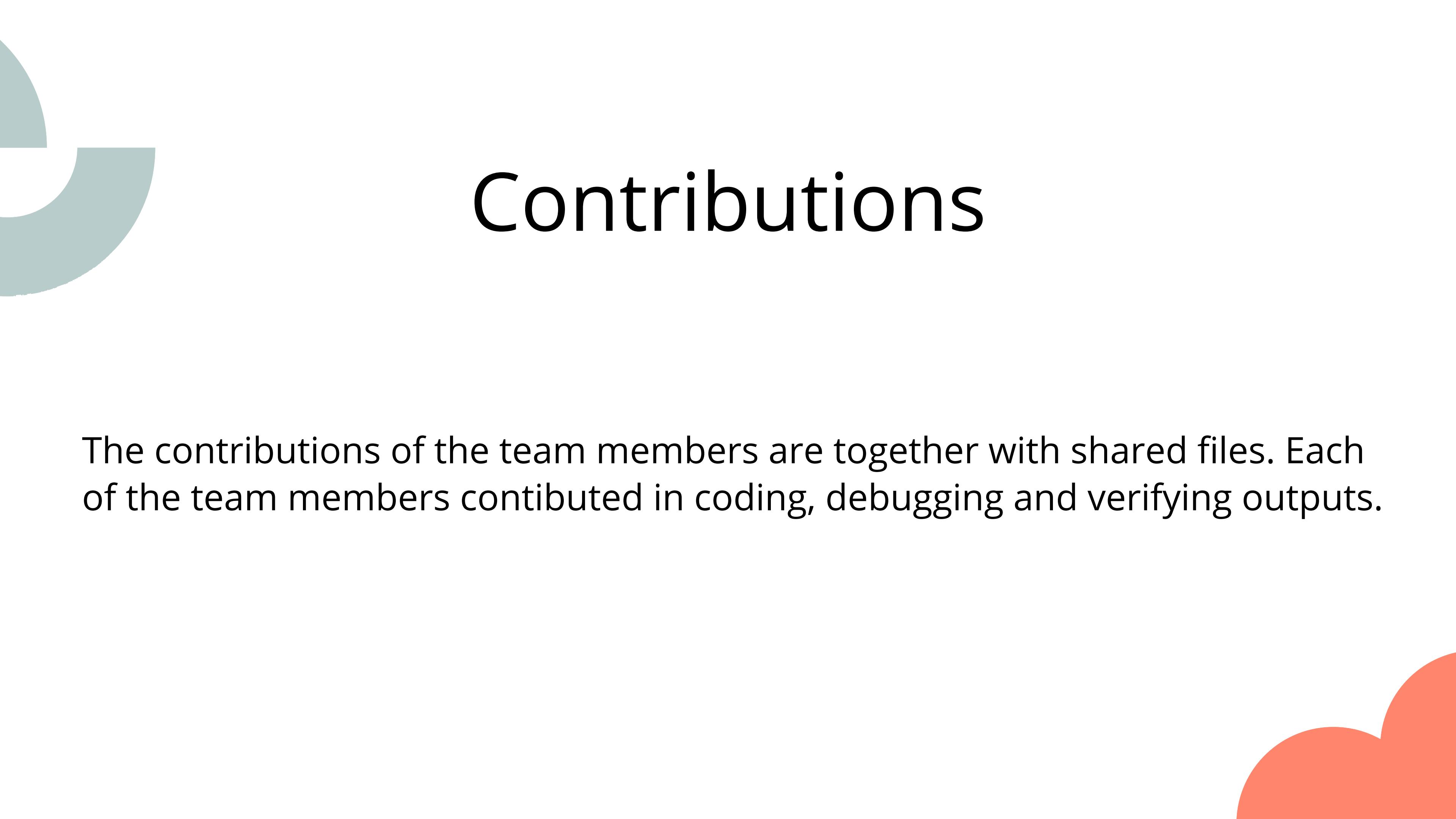
0000000000011000
1110001100000001
0000000000010010
1111110000010000
0000000000010001
1111000010010000
0000000000010010
1110001100001000
0000000000010001
1111110111001000
0000000000010000
1110101010000111
0000000000010010
1111110000010000
0000000000000001
1110101010001000
0000000000000000
1111010011010000
0000000000010100
1110001100000100
0000000000000001
1111110111001000
00000000000010011
1110001100001000
0000000000000010
1110001100001000
0000000000011100
1110001100000001
00000000000101000
1110101010000111

Comparison:

0000000000011000
1110001100000001
0000000000010010
1111110000010000
0000000000010001
1111000010010000
0000000000010010
1110001100001000
0000000000010001
1111110111001000
0000000000000001
1110101010000111
0000000000010010
1111110000010000
0000000000000001
1111110111001000
0000000000000000
1110101010001000
0000000000000000
1111010011010000
0000000000010100
1110001100000100
0000000000000001
1111110111001000
00000000000010011
1110001100001000
0000000000000010
1110001100001000
0000000000011100
1110001100000001
00000000000101000
1110101010000111

Status Bar: File compilation & comparison succeeded

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 contributed in coding, debugging and verifying outputs.

Thank you!

