Sreyas Janamanchi(IMT2022554)

Pradyumna G(IMT2022555)

We have chosen bubble sort as our sorting algorithm.

MIPS code:-

1. We copied all the elements of the input array into the output array.
2. We created a loop in a loop and initialized counter variables.
3. To implement loop we made a branch condition, that branches when value of the loop counter reaches the maximum value. Just before the branch label a jump statement is used to jump back to the branch condition of the loop.
4. loopa: for(int i=1;i<N;i++)
   loopb: for(int j=1;j<N-i;j++)
5. In loopb we compare output[j] and output[j-1]. If output[j]>output[j-1] we branch ahead of a swap functionality snippet.

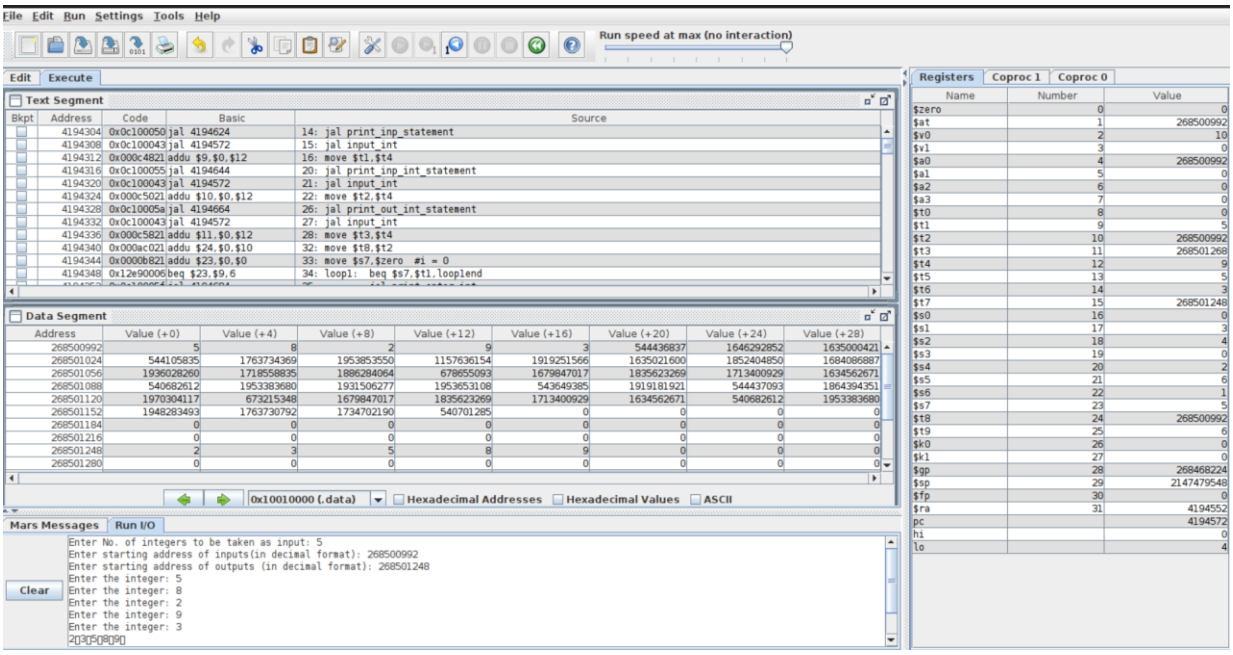
Assembler.py:-

1. We created 3 dictionaries: instructions (list of instructions against their opcodes), pseudcodes (list of pseudocodes against their MIPS instruction names), variables (list of registers against their address)
2. We initialized an empty list(data) that stores the machine code which is later printed out in another text file known as: “IMT2022554_IMT2022555_output.txt”.
3. The MIPS code is read from “IMT2022554_IMT2022555_sorting.asm”.
4. The code is split line by line and each word of a line is looped through.
5. We try matching each word with the keys present in “instructions” dictionary. And store the machine code of the instruction in the “data” list accordingly.

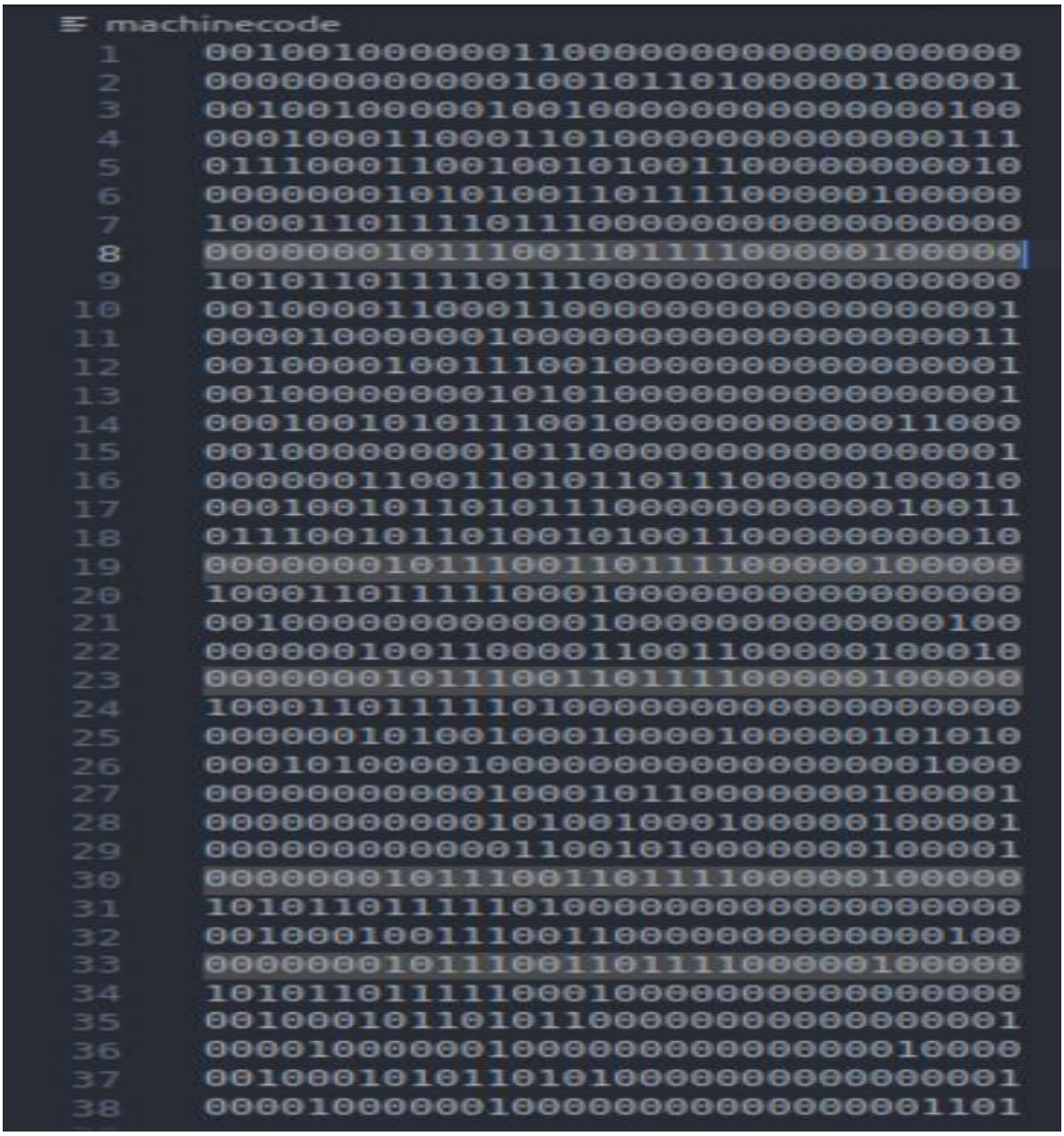
6. To find the immediate for beq instruction and j instruction we use beqHelper and jHelper functions respectively.
7. To convert decimal number to the desired binary length we created “decimal_to_binary” function.
8. Each element in “data” list is printed in: “IMT2022554_IMT2022555_output.txt”.

SCREENSHOTS

1) Execution of MIPS code



2)Machine code of MIPS sorting algorithm generated by MIPS



3)Machine code output from assembler.py

IMT2022554_IMT2022555_output.txt

```
1  00100100000011000000000000000000
2  00000000000010010110100000100001
3  00100100000100100000000000000100
4  00010001100011010000000000000111
5  01110001100100101001100000000010
6  00000000101010011011110000010000
7  10001101111011100000000000000000
8  00000000101110011011110000010000
9  10101101111011100000000000000000
10 00100001100011000000000000000001
11 00001000000100000000000000000011
12 00100001001110010000000000000001
13 00100000000101010000000000000001
14 00010010101110010000000000011000
15 00100000000101100000000000000001
16 00000001100110101101110000010010
17 00010010110101110000000000010011
18 01110010110100101001100000000010
19 00000000101110011011110000010000
20 10001101111100010000000000000000
21 00100000000000100000000000000100
22 00000001001100001100110000010010
23 00000000101110011011110000010000
24 10001101111101000000000000000000
25 00000001010010001000010000010101
26 00010100001000000000000000001000
27 00000000000010001011000000100001
28 00000000000010100100010000010001
29 00000000000011001010000000100001
30 00000000101110011011110000010000
31 10101101111101000000000000000000
32 00100001001110011000000000000100
33 00000000101110011011110000010000
34 10101101111100010000000000000000
35 00100010110101100000000000000001
36 00001000000010000000000000010000
37 00100010101101010000000000000001
38 00001000000010000000000000001101
```