Assignment-1 Report

By:
- Divyam Sareen (IMT2022010)
- Aryan Mishra (IMT2022502)

Q1. We have implemented the Bubble sort algorithm to sort the elements in ascending order.

- Firstly, let's look at the pseudocode of Bubble sort algorithm:

```
bubbleSort(arr){
n = length(arr)
for i from 0 to n-1{
        for j from 0 to n-i-1{
                if arr[j] > arr[j+1]
                        swap(arr[j], arr[j+1])
        }
}
}
```

This basically compares the adjacent elements of the given array. If the adjacent elements are out of order(i.e increasing/decreasing depends on what the user chooses to have), they are swapped. This process is repeated iteratively throughout the array and so at the end of all iterations, we find the elements at their correct positions in order.

- Explanation of the code we wrote(for step by step detailed explanation please refer to the comments of the code):

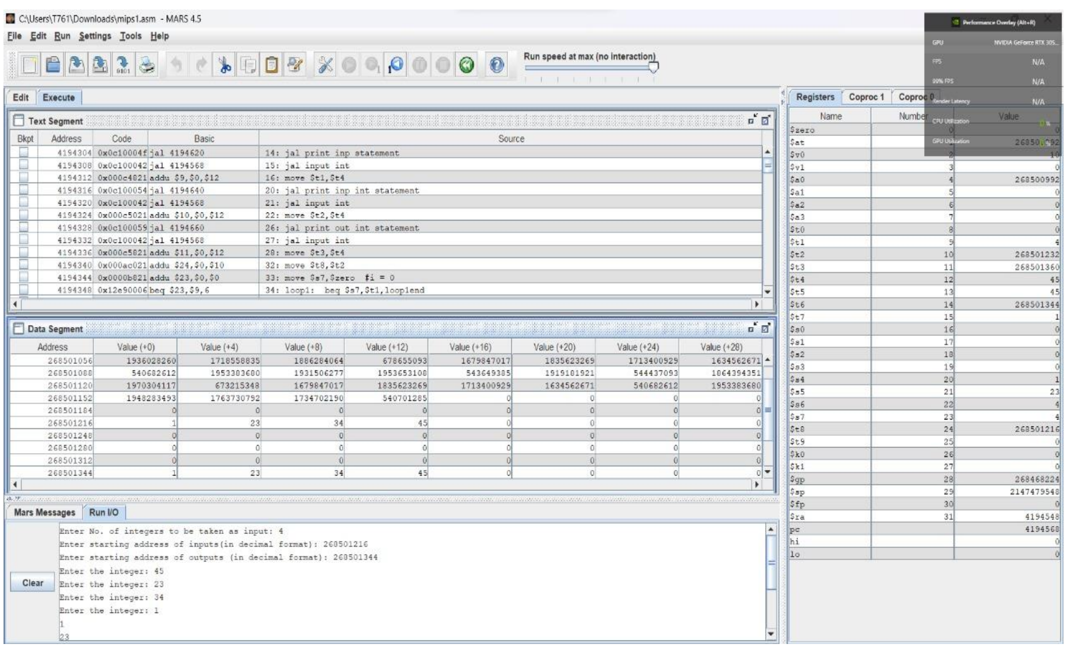
  It can be divided into several parts:

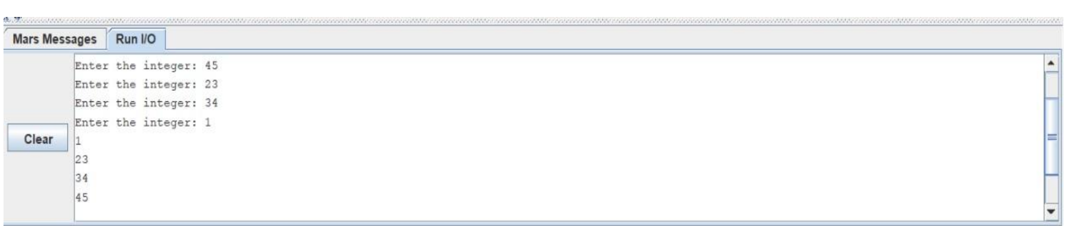
1. Initialisation:
   → Set $t5 to n - 1 (assuming $t1 contains n)
   → Initialize outer loop iterator to 0
   → Store the input address in $t8
2. Outer loop(loop 2):
   → Check if outer loop iterator is equal to n - 1, exit if true
3. Inner loop(loop 3) within outer loop:
   → Set $t7 to n - i - 1 (where i is the outer loop iterator)
   → Initialize inner loop iterator to 0
   → Restore the original input address for the inner loop
4. Inner loop body(loop 3):
   → Check if inner loop iterator is equal to n - i - 1, exit if true
   → Load arr[j] into $s4
   → Load arr[j+1] into $s5
   → If arr[j] > arr[j+1], jump to swap
   → Increment j
   → Increment inner loop iterator
   → Jump back to the start of inner loop
5. Swap function:
   → move arr[j] to a temporary register for swapping if the condition arr[j]>arr[j+1] is satisfied
   → move arr[j+1] to arr[j]
   → move value of arr[j] which was stored in a temporary register into arr[j+1] and hence swapping is completed
   → store the value of arr[j] in the memory
   → store the value of arr[j+1] in the memory
   → increment j
   → increment the iterator of inner loop
   → jump back to the start of inner loop
6. End of outer loop (loop 2):
   → Restore the original input address

→ Store the original output address in $t6
→ Initialize iterator for the next loop to 0

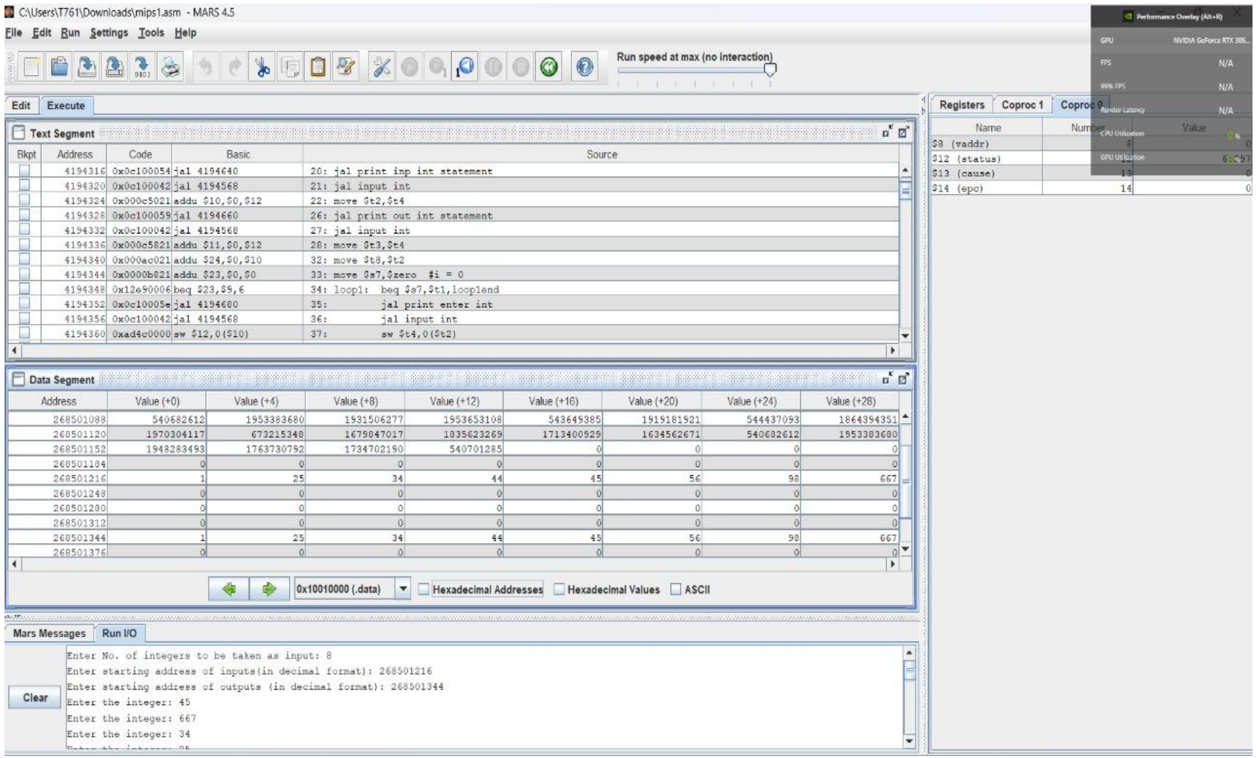
7. Copy result to output(loop 4):
   → Check if iterator is equal to n, exit if true
   → Load the first element of the input array into $t5
   → Store that element in the memory at the given output
   → Move to the next element of the output array
   → Move to the next element of the input array
   → Increment iterator
   → Jump back to the start of this loop
8. End of loop 4:
   → Restore the original output address into $t3

- Sample outputs:
  1.

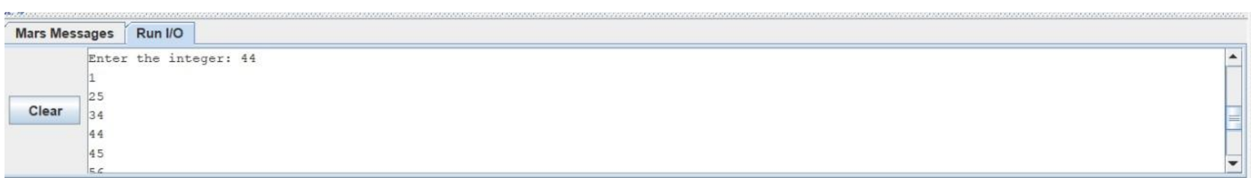


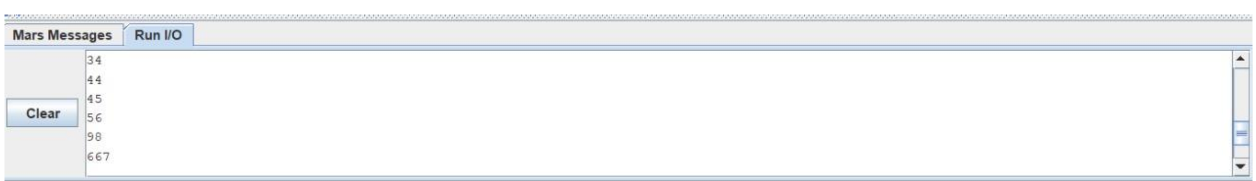


2.


C:\Users\T761\Downloads\mips1.asm - MARS 4.5
Text Segment
Data Segment
Mars Messages
Run I/O
Clear
Enter No. of integers to be taken as input: 8
Enter starting address of inputs(in decimal format): 268501216
Enter starting address of outputs (in decimal format): 268501344
Enter the integer: 45
Enter the integer: 667
Enter the integer: 34


Mars Messages
Run I/O
Clear
Enter the integer: 667
Enter the integer: 34
Enter the integer: 25
Enter the integer: 1
Enter the integer: 56
Enter the integer: 98


Mars Messages
Run I/O
Clear
Enter the integer: 44
1
25
34
44
45


Mars Messages
Run I/O
Clear
34
44
45
56
98
667

Q2. Assembler for the sorting algorithm:

{NOTE: We have only created the code which gives the machine code of the sorting algorithm part not input and outputs. So, the machine code generated by the code will be the subset of the actual code generated by MARS)
We have named the code generated by assembler as "machine_code.txt"
We have named the code generated by assembler as "machinecode7.txt"

Explanation:

- Step 1: Initialization and Definitions

→ The code begins by defining dictionaries for opcodes, function codes, register codes, numeric values, and label addresses. These dictionaries provide mappings between assembly mnemonics and their binary representations.
→ Additionally, the code initializes an empty list (`machine_code`) to store the generated machine code. Another dictionary (`label_dict`) is created to store labels and their corresponding addresses.

- Step 2: Function Definitions

The code defines several functions that will be used to assemble different types of MIPS instructions:
→ assemble_r_type(parts): Assembles R-type instructions.
→ assemble_i_type(parts): Assembles I-type instructions.
→ assemble_lw_sw_type(parts): Assembles LW/SW-type instructions.
→ assemble_j_type(parts): Assembles J-type instructions.
→ process_line(line, lineno, pass_no): Processes each line of the assembly code during both passes, collecting labels in the first pass and generating machine code in the second pass.

- Step 3: First Pass (Label Collection)

The code opens the input file (`mips.txt`) and iterates through each line. In the first pass, it collects labels and their corresponding addresses:

→ It skips empty lines and removes comments and whitespace.

→ For lines with labels, it extracts the label and stores its address.

→ The line number is incremented.

- Step 4: Second Pass (Code Generation)

The code clears the `machine_code` list and opens the input file again. During the second pass, it generates machine code:

→ For each line, it skips empty lines and removes comments and whitespace.

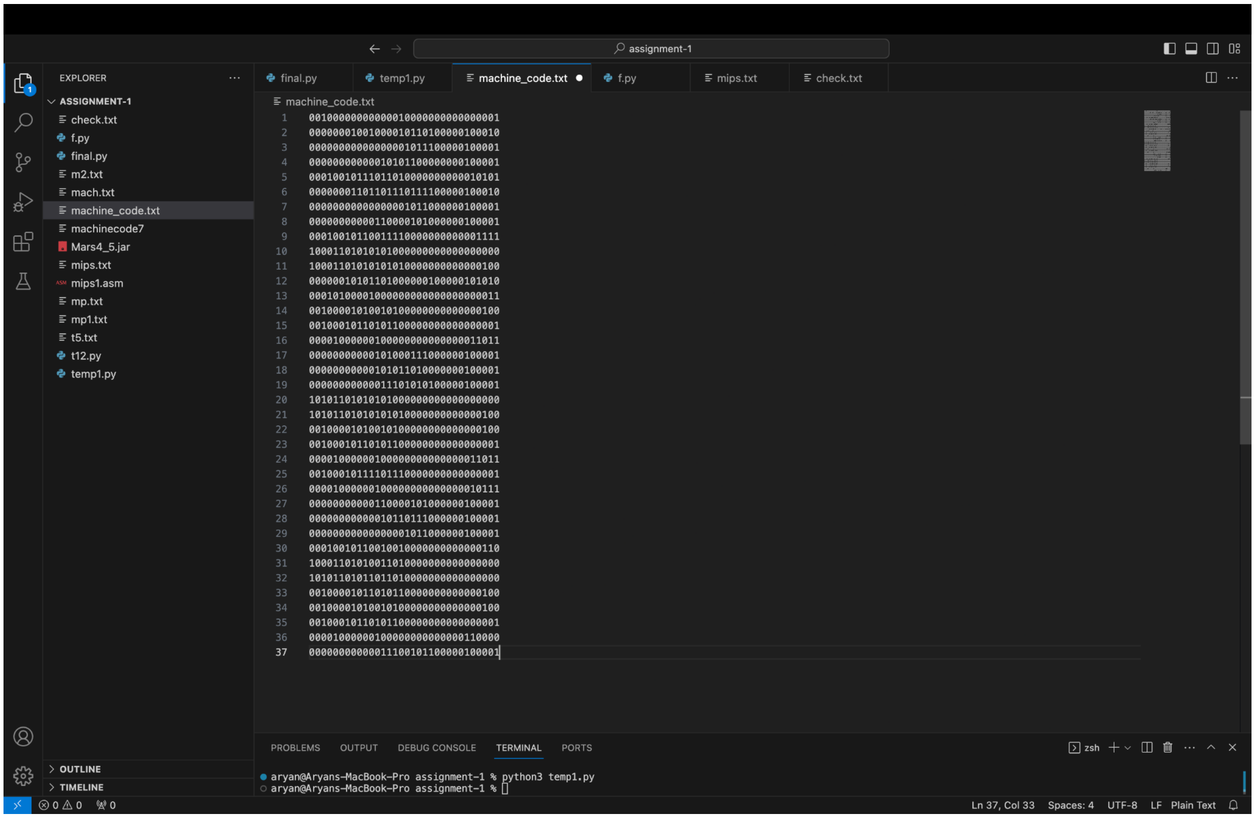
→ It calls the `process_line` function to assemble instructions based on their types.

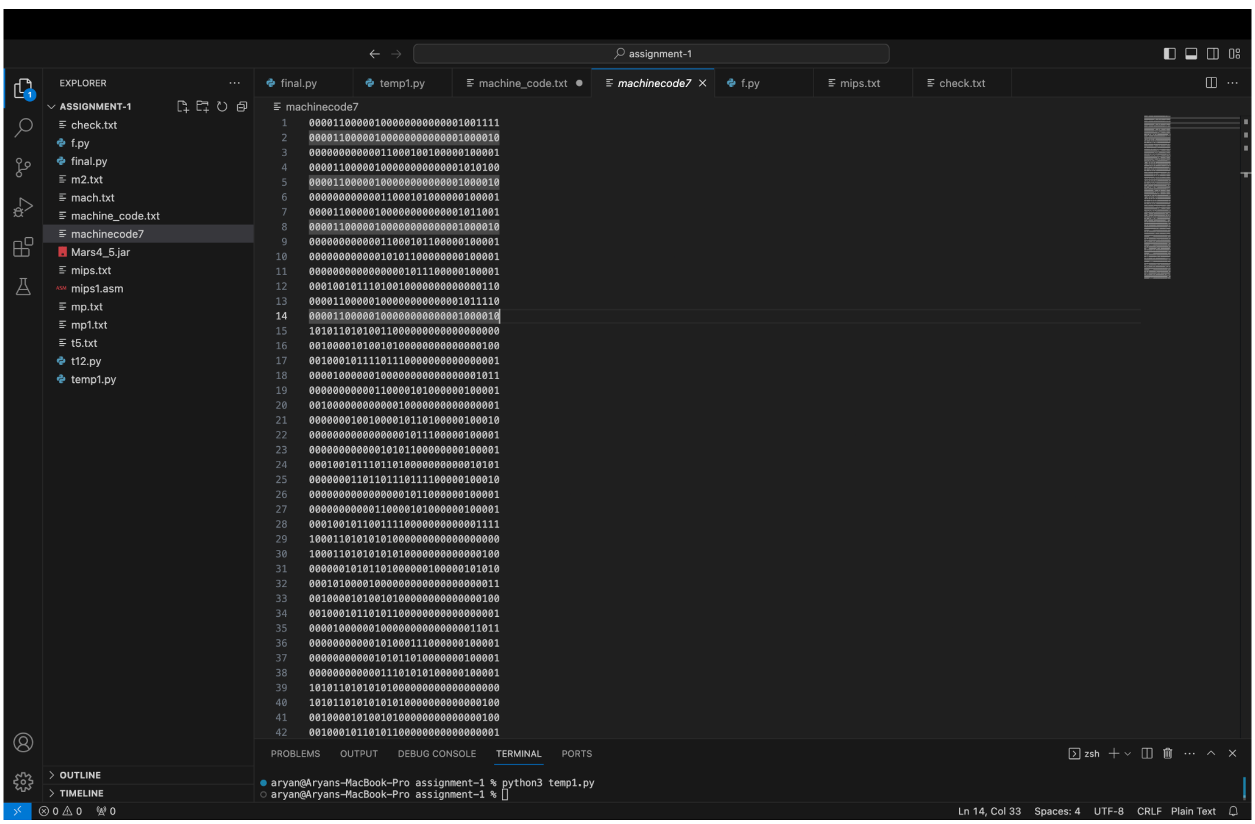
→ Special handling is provided for the `move` and `bgt` instructions as they are pseudo-instructions consisting of more than one instruction.

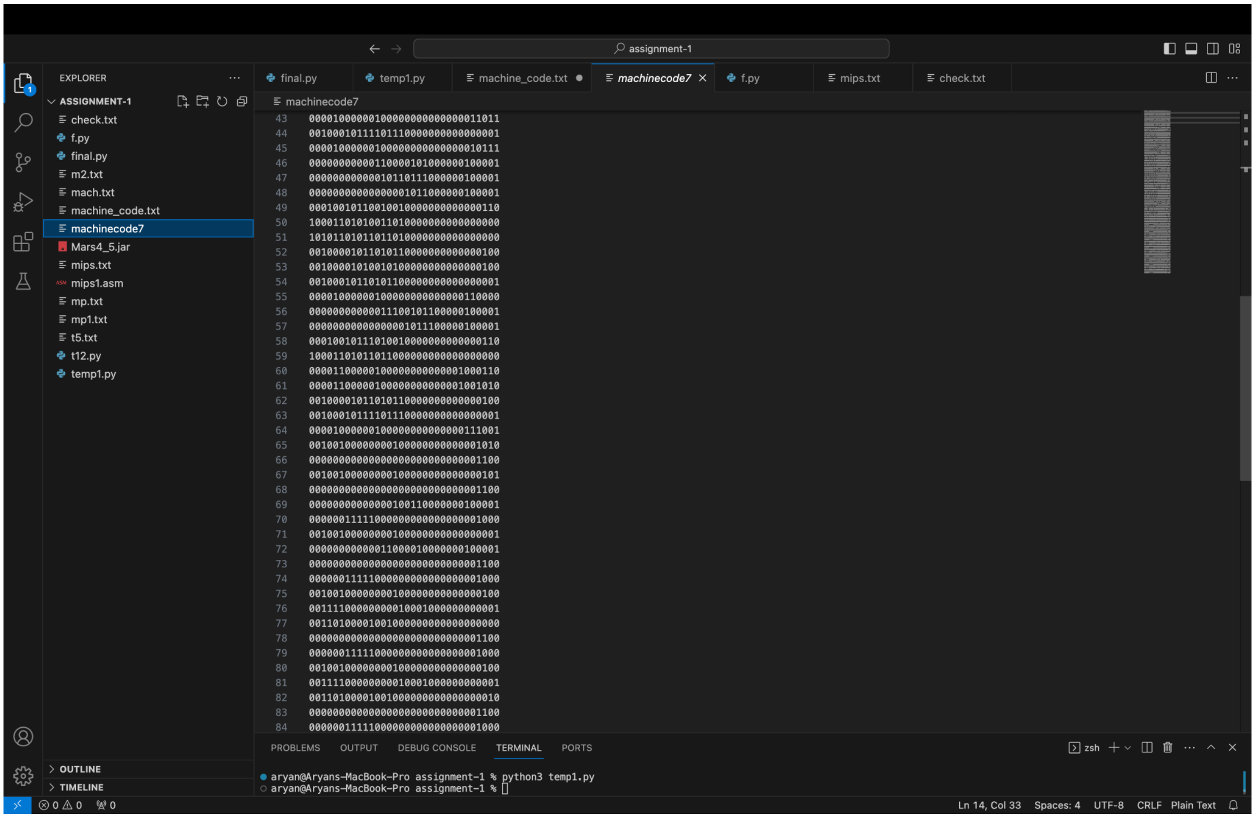
    - `move` is implemented as an `addu` instruction.
    - `bgt` is implemented as `slt` followed by `bne`.

- Step 5: Output

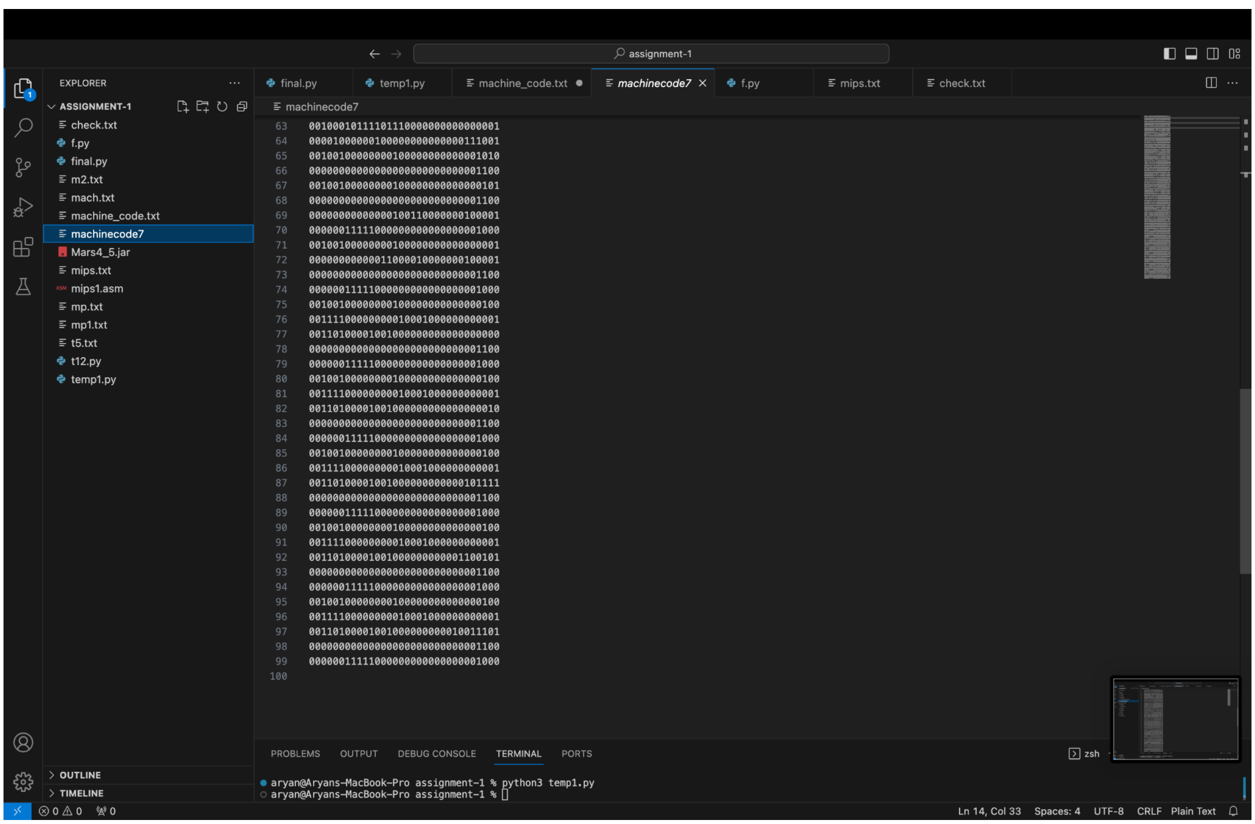
The generated machine code is written to an output file (`machine_code.txt`).


machine_code.txt
00100000000000001000000000000001
00000001001000010110100000100010
00000000000000001011100000100001
00000000000010101100000000100001
00010010111011010000000000010101
00000001101101110111100000100010
00000000000000001011000000100001
00000000000110000101000000100001
00010010110011110000000000001111
10001101010101000000000000000000
10001101010101010000000000000100
00000010101101000000100000101010
00010100001000000000000000000011
00100001010010100000000000000100
00100010110101100000000000000001
00001000000100000000000000011011
00000000000101000111000000100001
00000000000101011010000000100001
00000000000111010101000000100001
10101101010101000000000000000000
10101101010101010000000000000100
00100001010010100000000000000100
00100010110101100000000000000001
00001000000100000000000000011011
00100010111101110000000000000001
00001000000100000000000000010111
00000000000110000101000000100001
00000000000101101110000000100001
00000000000000001011000000100001
00010010110010010000000000000110
10001101010011010000000000000000
10101101011011010000000000000000
00100001011010110000000000000100
00100001010010100000000000000100
00100010110101100000000000000001
00001000000100000000000000110000
00000000000111001011000000100001
aryan@Aryans-MacBook-Pro assignment-1 % python3 temp1.py
aryan@Aryans-MacBook-Pro assignment-1 %


machinecode7
00001100000100000000000001001111
00001100000100000000000001000010
00000000000110001001000000100001
00001100000100000000000001010100
00001100000100000000000001000010
00000000000110001010000000100001
00001100000100000000000001011001
00001100000100000000000001000010
00000000000110001011000000100001
00000000000010101100000000100001
00000000000000001011100000100001
00010010111010010000000000000110
00001100000100000000000001011110
00001100000100000000000001000010
10101101010011000000000000000000
00100001010010100000000000000100
00100010111101110000000000000001
00001000000100000000000000001011
00000000000110000101000000100001
00100000000000001000000000000001
00000001001000010110100000100010
00000000000000001011100000100001
00000000000010101100000000100001
00010010111011010000000000010101
00000001101101110111100000100010
00000000000000001011000000100001
00000000000110000101000000100001
00010010110011110000000000001111
10001101010101000000000000000000
10001101010101010000000000000100
00000010101101000000100000101010
00010100001000000000000000000011
00100001010010100000000000000100
00100010110101100000000000000001
00001000000100000000000000011011
00000000000101000111000000100001
00000000000101011010000000100001
00000000000111010101000000100001
10101101010101000000000000000000
10101101010101010000000000000100
00100001010010100000000000000100
00100010110101100000000000000001
aryan@Aryans-MacBook-Pro assignment-1 % python3 temp1.py
aryan@Aryans-MacBook-Pro assignment-1 %



assignment-1
EXPLORER
ASSIGNMENT-1
check.txt
f.py
final.py
m2.txt
mach.txt
machine_code.txt
machinecode7
Mars4_5.jar
mips.txt
mips1.asm
mp.txt
mp1.txt
t5.txt
t12.py
temp1.py
OUTLINE
TIMELINE
final.py
temp1.py
machine_code.txt
machinecode7
f.py
mips.txt
check.txt
machinecode7
00001000000100000000000000011011
00100010111101110000000000000001
00001000000100000000000000010111
00000000000110000101000000100001
00000000000010110111000000100001
00000000000000001011000000100001
00010010110010010000000000000110
10001101010011010000000000000000
10101101011011010000000000000000
00100001011010110000000000000100
00100001010010100000000000000100
00100010110101100000000000000001
00001000000100000000000000110000
00000000000111001011000000100001
00000000000000001011100000100001
00010010111010010000000000000110
10001101011011000000000000000000
00001100000100000000000001000110
00001100000100000000000001001010
00100001011010110000000000000100
00100010111101110000000000000001
00001000000100000000000000111001
00100100000001000000000000001010
00000000000000000000000000001100
00100100000001000000000000000101
00000000000000000000000000001100
00000000000000100110000000100001
00000011111000000000000000001000
00100100000001000000000000000001
00000000000110000100000000100001
00000000000000000000000000001100
00000011111000000000000000001000
00100100000001000000000000000100
00111100000000010001000000000001
00110100001001000000000000000000
00000000000000000000000000001100
00000011111000000000000000001000
00100100000001000000000000000100
00111100000000010001000000000001
00110100001001000000000000000010
00000000000000000000000000001100
00000011111000000000000000001000
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
zsh
aryan@Aryans-MacBook-Pro assignment-1 % python3 temp1.py
aryan@Aryans-MacBook-Pro assignment-1 %
Ln 14, Col 33
Spaces: 4
UTF-8
CRLF
Plain Text



assignment-1
EXPLORER
ASSIGNMENT-1
check.txt
f.py
final.py
m2.txt
mach.txt
machine_code.txt
machinecode7
Mars4_5.jar
mips.txt
mips1.asm
mp.txt
mp1.txt
t5.txt
t12.py
temp1.py
OUTLINE
TIMELINE
final.py
temp1.py
machine_code.txt
machinecode7
f.py
mips.txt
check.txt
machinecode7
00100010111101110000000000000001
00001000000100000000000000111001
00100100000001000000000000001010
00000000000000000000000000001100
00100100000001000000000000000101
00000000000000000000000000001100
00000000000000100110000000100001
00000011111000000000000000001000
00100100000001000000000000000001
00000000000110000100000000100001
00000000000000000000000000001100
00000011111000000000000000001000
00100100000001000000000000000100
00111100000000010001000000000001
00110100001001000000000000000000
00000000000000000000000000001100
00000011111000000000000000001000
00100100000001000000000000000100
00111100000000010001000000000001
00110100001001000000000000000010
00000000000000000000000000001100
00000011111000000000000000001000
00100100000001000000000000000100
00111100000000010001000000000001
00110100001001000000000000101111
00000000000000000000000000001100
00000011111000000000000000001000
00100100000001000000000000000100
00111100000000010001000000000001
00110100001001000000000001100101
00000000000000000000000000001100
00000011111000000000000000001000
00100100000001000000000000000100
00111100000000010001000000000001
00110100001001000000000010011101
00000000000000000000000000001100
00000011111000000000000000001000
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
zsh
aryan@Aryans-MacBook-Pro assignment-1 % python3 temp1.py
aryan@Aryans-MacBook-Pro assignment-1 %
Ln 14, Col 33
Spaces: 4
UTF-8
CRLF
Plain Text