Computer Architecture

Assignment-1

1) Implementation of Sorting algorithm using MIPS (template.asm)

We implemented the bubble sort algorithm to sort the numbers in increasing order.We stored the variables 'n','n-1','i'(control variable for outer for loop) in different temporary registers. When 'n'=1,there's no need to enter the loop . So we jump to the label 'skiploop'. In the outer for loop , we store the starting address of input values,output values and control variable for inner for loop(j) in other temporary registers. In the inner for loop, we compare two adjacent elements. If the element at index a>element at index b(and a<b),then we swap the elements by calling the label 'swap' and then continue. At the end of each iteration of outer for loop,the largest element will occupy the last position . Then we write this element to the (starting address of output values+4*(n-i-1)).(So if n=4 and i=0 then at the end of the first iteration of the outer for loop, we write the element at a[n-i-1] to address=base output address+4*(n-1))

This part where we write to the output addresses is done in the label 'innerloopdone' signyfying it is called after each iteration of the outer loop. We continue this process till i becomes n-1 and once i=n-1 , we go to 'done' block where we write the smallest element in the array to the base output address. After this,the numbers entered are sorted in ascending order as proven by the output as well.

Output of MIPS code

```
Enter No. of integers to be taken as input: 4
Enter starting address of inputs(in decimal format): 268435456
Enter starting address of outputs (in decimal format): 268435856
Enter the integer: 7
Enter the integer: 4
Enter the integer: 9
Enter the integer: 2
2
```

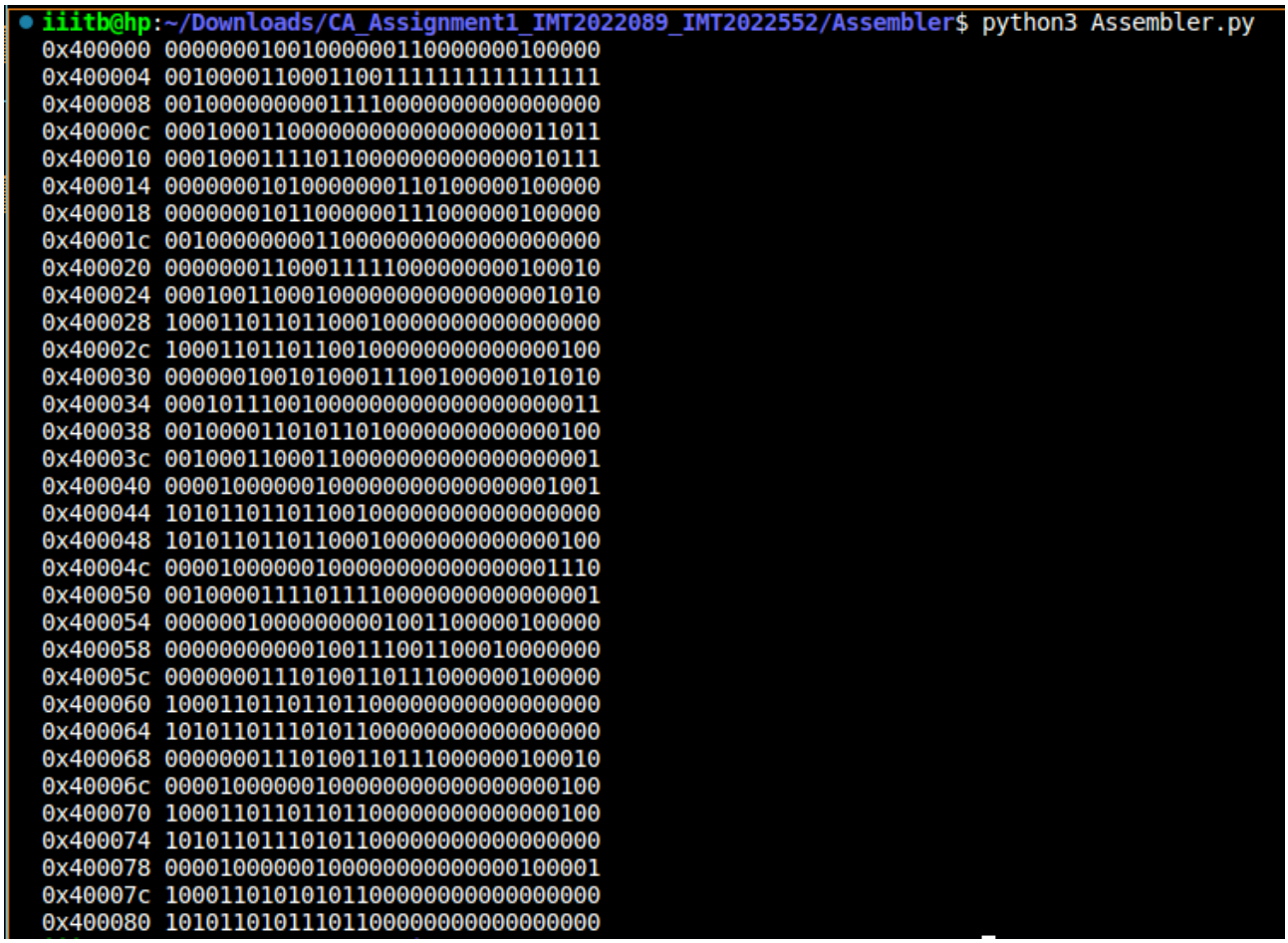
```
2
4
7
9

-- program is finished running --
```

2) Assembler.py

The python program Assembler.py creates an assembler for the template.asm code. An assembler converts assembly level code into machine level code. First we define few dictionaries which gives us the mapping between instructions and opcodes, registers and numbers, labels and immediates. Then we write a function called Convert_to_Binary() which takes a string input of a decimal number and gives us its binary equivalent output. The binary string length can be 5 or 16 depending on the parameter. We handle the negative number case by using the 2's complement representation. Conditional statements are used to check if a number is positive or negative. The two's complement is calculated in the function. Finally we use nested if-else statement to check the type of instruction and calculate its corresponding machine code. The machine code is represented using a string called 'output'. We use the split() function to make the file-input simpler. Output for each instruction is calculated based on its type ie R,I,J type instructions. We use a for-loop to loop through the template.asm code and print the corresponding hexadecimal PC value and the output. In the end the PC is incremented by four.

Output of Assembler.py



By:- IMT2022089 (Aayush Bhargav)

IMT2022552 (Shannon Muthanna IB)