

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' signifying 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

```
• iiitb@hp:~/Downloads/CA_Assignment1_IMT2022089_IMT2022552/Assembler$ python3 Assembler.py
0x400000 00000001001000000110000000100000
0x400004 00100001100011001111111111111111
0x400008 00100000000011110000000000000000
0x40000c 00010001100000000000000000011011
0x400010 00010001111011000000000000010111
0x400014 00000001010000000110100000100000
0x400018 00000001011000000111000000100000
0x40001c 00100000000110000000000000000000
0x400020 0000000110001111100000000100010
0x400024 0001001100010000000000000001010
0x400028 10001101101100010000000000000000
0x40002c 10001101101100100000000000000100
0x400030 00000010010100011100100000101010
0x400034 0001011100100000000000000000011
0x400038 00100001101011010000000000000100
0x40003c 00100011000110000000000000000001
0x400040 0000100000010000000000000001001
0x400044 10101101101100100000000000000000
0x400048 10101101101100010000000000000100
0x40004c 00001000000100000000000000000110
0x400050 00100001111011110000000000000001
0x400054 00000010000000001001100000100000
0x400058 00000000000100111001100010000000
0x40005c 00000001110100110111000000100000
0x400060 10001101101101100000000000000000
0x400064 10101101110101100000000000000000
0x400068 00000001110100110111000000100010
0x40006c 00001000000100000000000000000100
0x400070 10001101101101100000000000000100
0x400074 10101101110101100000000000000000
0x400078 000010000001000000000000000100001
0x40007c 10001101010101100000000000000000
0x400080 10101101011101100000000000000000
```

By:- IMT2022089 (Aayush Bhargav)

IMT2022552 (Shannon Muthanna IB)