ASSIGNMENT 1- MARS ASSEMBLER

BY – VEDANT MUNDADA (BT2024268)

- **CONTENTS-**

1. Assembler.py (to assemble the MIPS code)
2. Machine_code.txt (the assembler output is stored in this file)
3. Binary_search.asm
4. GCD.asm
5. GCD folder –
    1. Code for GCD (GCD.cpp)
    2. Assemble code for GCD (GCD.asm)
    3. Machine code generated from MARS – (GCD_MARS.bin)
    4. Machine code generated form my assembler – (GCD_mc.bin)
6. Binary Search-
    1. Code for Binary Search (Binary_search.cpp)
    2. Assembly code for Binary Searcg (Binary_search.asm)
    3. Machine code generated from MARS – (Binary_search_MARS.bin)
    4. Machine code generated form my assembler – (Binary_search_mc.bin)
7. Comparator Folder-
    1. Assembled code to be placed here (Assembled.txt)
    2. MARS code to be placed here (Original.txt)
    3. Compare.py (To compare the codes in Assembled.txt and Original.txt and find any discrepancies)

- **GCD MIPS code-**

The code follows the Recursive Stein's Algorithm-

After assembling it will ask to input the two numbers for which the we want to find the GCD-

```
Enter the first number: 3
Enter the second Number: 4
```

After pressing enter, the output will be displayed-

```
Enter the first number: 3
Enter the second Number: 4
The GCD of the two numbers is: 1
```

- **Binary Search Code-**

After assembling, it will ask to input the number of integers in the array

```
Enter No. of integers to be taken as input: 4
```

After that it will ask to input the staring address of the array location (ex – 268501216)

```
Enter starting address of inputs(in decimal format): 268501216
```

After that we have to enter the numbers one by one in a sorted manner

```
Enter the number in sorted order: 2
Enter the number in sorted order: 3
Enter the number in sorted order: 4
```

Then we have to input the number we want to search for-

```
Enter the number to search for: 3
```

The output will give the index (0 – indexed) if the number exists or it will say that the number is not found-

```
Number found at index(0 - indexing): 2
-- program is finished running --
```

- ## Assembler

The assembler will take a .asm or .txt file as input and put the output in a file called "machine_code.txt"
For giving the input file, we have to put the file name at this location and compile the code:

```
46    text_instructions = []
47
48    with open('Binary_search.asm', 'r', encoding='utf-8') as f:
49        lines = [line.strip() for line in f.readlines()]
50
51    in_data = False
```

Output file:

```
with open('machine_code.txt', 'w') as f:
    for _, binary in machine_code:
        f.write(binary + '\n')
```

After compiling, the machine code will be available in the output file and the process the assembler went with will be displayed in the cmd lines:

```
Data labels found:
next_line
inp_statement
inp_int_statement
inp_number
enter_int
not_found
found_statement
```

```
Text labels found:
loop
loop1end
loop1
cond1
cond2
found
notfound
end
input_int
print_int
print_line
```

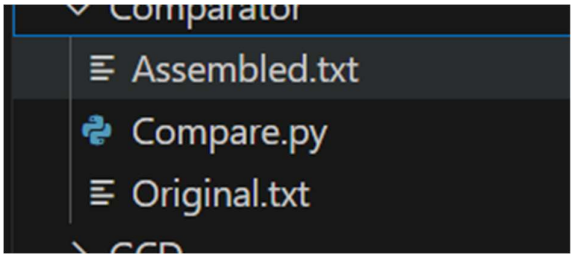
```
Text instructions found:
0x400000: jal print_inp_statement
0x400004: jal input_int
0x400008: addi $t1,$t4,0
0x40000c: jal print_inp_int_statement
0x400010: jal input_int
0x400014: addi $t2,$t4,0
0x400018: addi $t8,$t2,0
0x40001c: addi $s7,$0,0
```

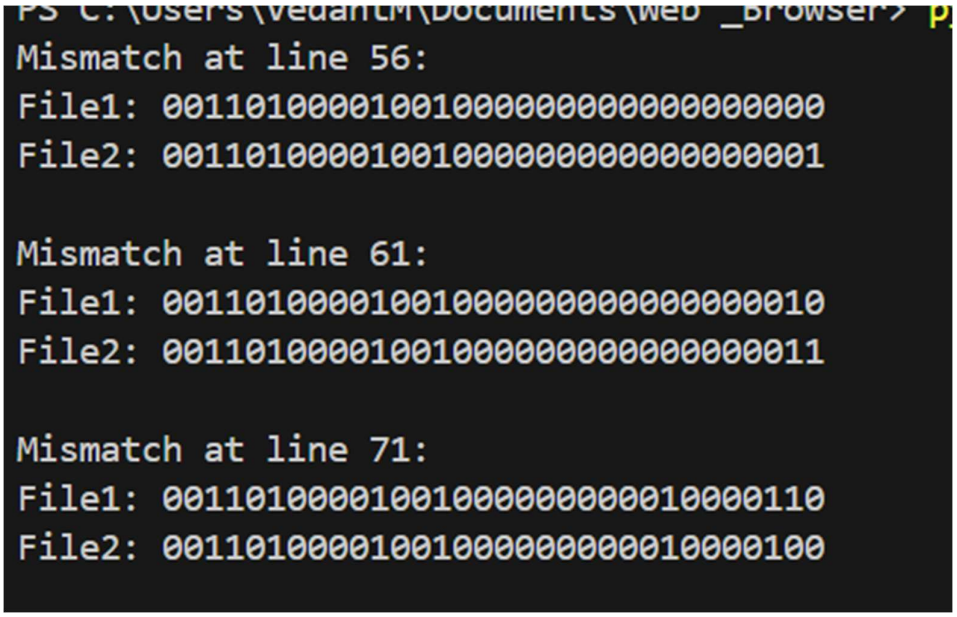
```
Data label addresses assigned:
next_line: 0x10010000
inp_statement: 0x10010002
inp_int_statement: 0x1001002e
inp_number: 0x10010063
enter_int: 0x10010083
not_found: 0x100100a5
found_statement: 0x100100b5

Assembly successful... The output is printed in the machine_code.txt file!!
```

- **Comparator**

It just a checking tool that can be used to check two text files Assembled.txt and Original.txt, if the codes are exactly the same then it will output that the files are same, but it they are not then it will display the lines where there is a mismatch-





How to run-
Command = python .\Compare.py .\Original.txt .\Assembled.txt

Disclaimer:

While giving the input to the assembler, we have to give the entire file including the .data segment as well, as it will use that to make address references for the data segments as well, and when we compare the assembled text with the MARS machine code, the mismatches shown are all address mis-matches of the .data segment as the assembler is not able to precisely calculate the address of all the .data segments as the MARS simulator, apart from that everything else should match.

The instruction included in the assembler are:

1. Li
2. Ble
3. La
4. Jal
5. J
6. Addi
7. Or
8. Andi
9. Bne
10. Srl
11. Beq
12. Sub
13. Sllv
14. Jr
15. And
16. Sll
17. Lw
18. Sw
19. Slt
20. Syscall