ASSIGNMENT 1
COMPUTER ARCHITECTURE
IMT2022021 AND IMT2022098
RUTUL and HEMANG

Introduction->

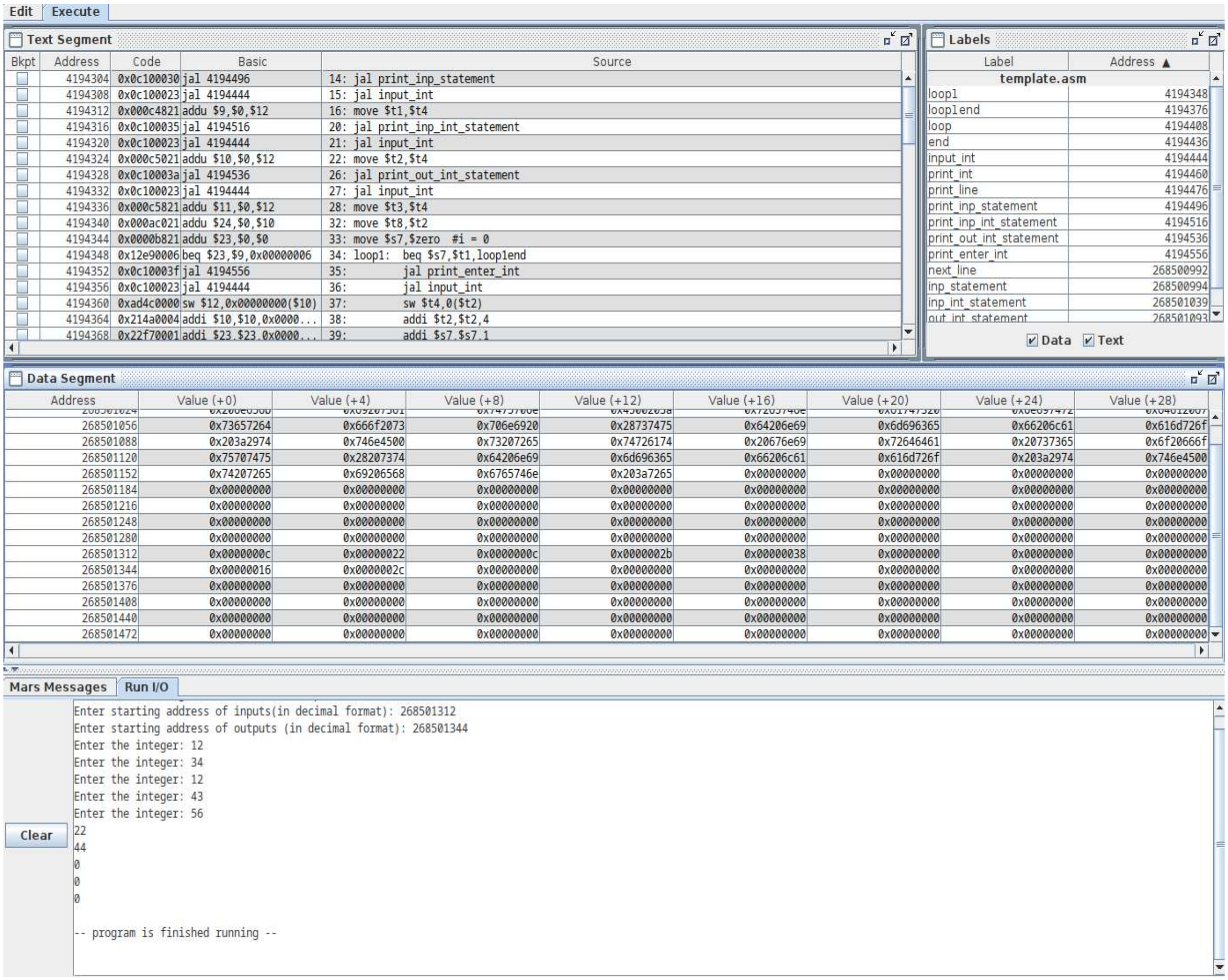
Here in this report we have given the detailed instructions on how to use our MARS code and the guidance to our own assembler which was running the same MIPS code. We have made our code in python.

Our chosen mode of display was hexadecimal format although through out our main function we have used binary standard values only.

Description->

1. Question 1 part (a)

Here in this code we had to consider the template.asm which was given to us. We had to check if it is working fine or not. This is working successfully hence it is correct.



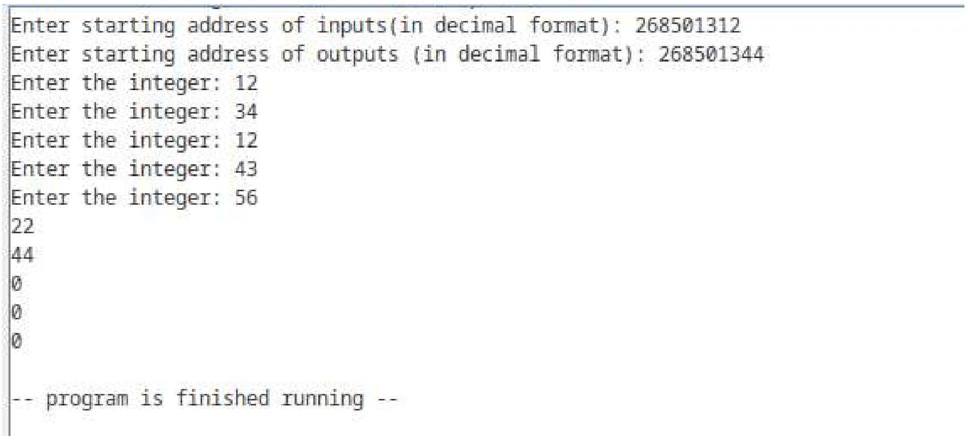
Bkpt	Address	Code	Basic	Source
☐	4194304	0x0c100030	jal 4194496	14: jal print_inp_statement
☐	4194308	0x0c100023	jal 4194444	15: jal input_int
☐	4194312	0x000c4821	addu $9,$0,$12	16: move $t1,$t4
☐	4194316	0x0c100035	jal 4194516	20: jal print_inp_int_statement
☐	4194320	0x0c100023	jal 4194444	21: jal input_int
☐	4194324	0x000c5021	addu $10,$0,$12	22: move $t2,$t4
☐	4194328	0x0c10003a	jal 4194536	26: jal print_out_int_statement
☐	4194332	0x0c100023	jal 4194444	27: jal input_int
☐	4194336	0x000c5821	addu $11,$0,$12	28: move $t3,$t4
☐	4194340	0x000ac021	addu $24,$0,$10	32: move $t8,$t2
☐	4194344	0x0000b821	addu $23,$0,$0	33: move $s7,$zero #i = 0
☐	4194348	0x12e90006	beq $23,$9,0x00000006	34: loop1: beq $s7,$t1,loop1end
☐	4194352	0x0c10003f	jal 4194556	35: jal print_enter_int
☐	4194356	0x0c100023	jal 4194444	36: jal input_int
☐	4194360	0xad4c0000	sw $12,0x00000000($10)	37: sw $t4,0($t2)
☐	4194364	0x214a0004	addi $10,$10,0x0000...	38: addi $t2,$t2,4
☐	4194368	0x22f70001	addi $23,$23,0x0000...	39: addi $s7,$s7,1

Label	Address ▲
template.asm	
loop1	4194348
loop1end	4194376
loop	4194408
end	4194436
input_int	4194444
print_int	4194460
print_line	4194476
print_inp_statement	4194496
print_inp_int_statement	4194516
print_out_int_statement	4194536
print_enter_int	4194556
next_line	268500992
inp_statement	268500994
inp_int_statement	268501039
out_int_statement	268501093

☑ Data ☑ Text

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
268501056	0x73657264	0x666f2073	0x706e6920	0x28737475	0x64206e69	0x6d696365	0x66206c61	0x616d726f
268501088	0x203a2974	0x746e4500	0x73207265	0x74726174	0x20676e69	0x72646461	0x20737365	0x6f20666f
268501120	0x75707475	0x28207374	0x64206e69	0x6d696365	0x66206c61	0x616d726f	0x203a2974	0x746e4500
268501152	0x74207265	0x69206568	0x6765746e	0x203a7265	0x00000000	0x00000000	0x00000000	0x00000000
268501184	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
268501216	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
268501248	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
268501280	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
268501312	0x0000000c	0x00000022	0x0000000c	0x0000002b	0x00000038	0x00000000	0x00000000	0x00000000
268501344	0x00000016	0x0000002c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
268501376	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
268501408	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
268501440	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
268501472	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

```
Enter starting address of inputs(in decimal format): 268501312
Enter starting address of outputs (in decimal format): 268501344
Enter the integer: 12
Enter the integer: 34
Enter the integer: 12
Enter the integer: 43
Enter the integer: 56
22
44
0
0
0

-- program is finished running --
```



```
Enter starting address of inputs(in decimal format): 268501312
Enter starting address of outputs (in decimal format): 268501344
Enter the integer: 12
Enter the integer: 34
Enter the integer: 12
Enter the integer: 43
Enter the integer: 56
22
44
0
0
0

-- program is finished running --
```

This shows that it is working correctly.

2. Question 1 part (b)

Our option for our team is Encryption and Decryption.

In this we have made a XOR cypher which contains the string which we enter. If we enter 0, then it will encrypt and then if we put 1 then it will decrypt it.

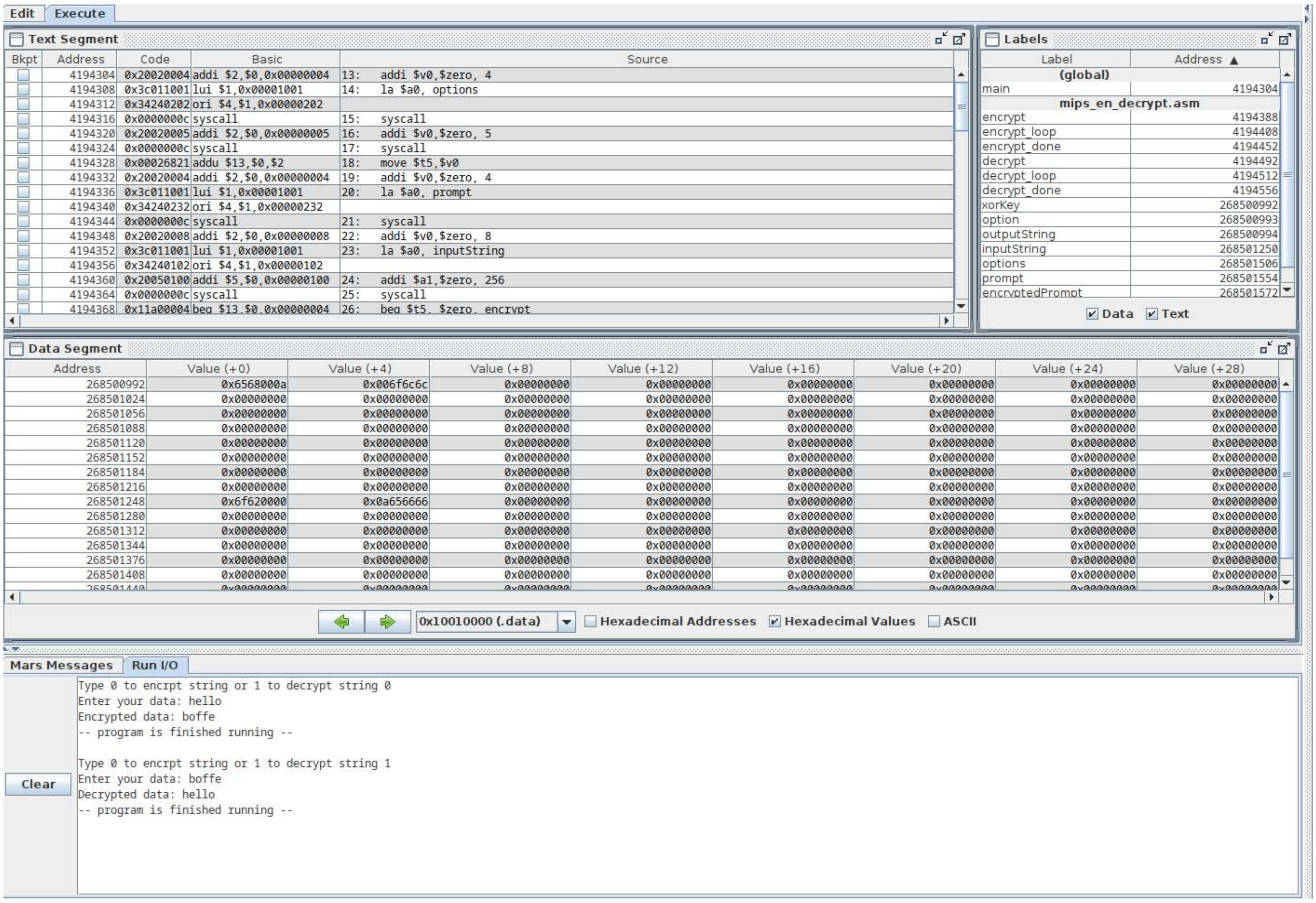
CONDITIONS->

We didn't put comments because its reading wasn't possible in the assembler. We didn't make the feature of comment reading in our assembler.

We started by declaring a 256 buffer string and then we used it to encrypt and decrypt.

HOW IS OUR DIFFERENT->

In our code we have managed to minimalism as our main feature. Our code is minimalistic and yet is very efficient providing a very efficient XOR cypher. Useage of different labels for different functions. NO Usage of loops in entire project to keep it as minimalistic as possible. We avoided using of pseudo labels as little as possible. It was to prevent confusion.



Bkpt	Address	Code	Basic	Source
□	4194304	0x20020004	addi $2,$0,0x00000004	13: addi $v0,$zero, 4
□	4194308	0x3c011001	lui $1,0x00001001	14: la $a0, options
□	4194312	0x34240202	ori $4,$1,0x00000202	
□	4194316	0x0000000c	syscall	15: syscall
□	4194320	0x20020005	addi $2,$0,0x00000005	16: addi $v0,$zero, 5
□	4194324	0x0000000c	syscall	17: syscall
□	4194328	0x00026821	addu $13,$0,$2	18: move $t5,$v0
□	4194332	0x20020004	addi $2,$0,0x00000004	19: addi $v0,$zero, 4
□	4194336	0x3c011001	lui $1,0x00001001	20: la $a0, prompt
□	4194340	0x34240232	ori $4,$1,0x00000232	
□	4194344	0x0000000c	syscall	21: syscall
□	4194348	0x20020008	addi $2,$0,0x00000008	22: addi $v0,$zero, 8
□	4194352	0x3c011001	lui $1,0x00001001	23: la $a0, inputString
□	4194356	0x34240102	ori $4,$1,0x00000102	
□	4194360	0x20050100	addi $5,$0,0x00000100	24: addi $a1,$zero, 256
□	4194364	0x0000000c	syscall	25: syscall
□	4194368	0x11a00004	beq $13,$0,0x00000004	26: beq $t5, $zero, encrypt

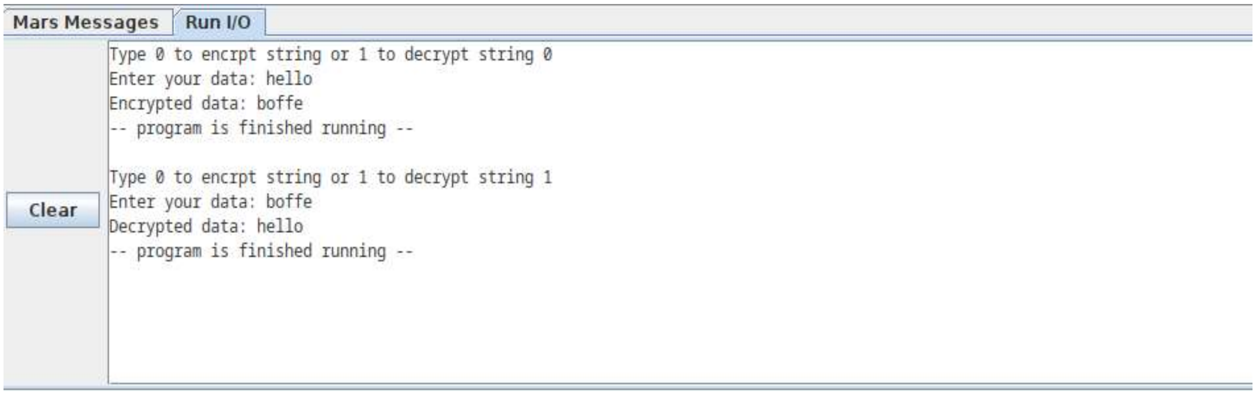
Label	Address ▲
(global)	
main	4194304
mips_en_decrypt.asm	
encrypt	4194388
encrypt_loop	4194408
encrypt_done	4194452
decrypt	4194492
decrypt_loop	4194512
decrypt_done	4194556
xorKey	268500992
option	268500993
outputString	268500994
inputString	268501250
options	268501506
prompt	268501554
encryptedPrompt	268501572

☑ Data ☑ Text

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
268500992	0x6568000a	0x006f6c6c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
268501024	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
268501056	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
268501088	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
268501120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
268501152	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
268501184	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
268501216	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
268501248	0x6f620000	0x0a656666	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
268501280	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
268501312	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
268501344	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
268501376	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
268501408	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
268501440	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x10010000 (.data) □ Hexadecimal Addresses ☑ Hexadecimal Values □ ASCII

```
Type 0 to encrpt string or 1 to decrypt string 0
Enter your data: hello
Encrypted data: boffe
-- program is finished running --

Type 0 to encrpt string or 1 to decrypt string 1
Enter your data: boffe
Decrypted data: hello
-- program is finished running --
```



```
Type 0 to encrpt string or 1 to decrypt string 0
Enter your data: hello
Encrypted data: boffe
-- program is finished running --

Type 0 to encrpt string or 1 to decrypt string 1
Enter your data: boffe
Decrypted data: hello
-- program is finished running --
```

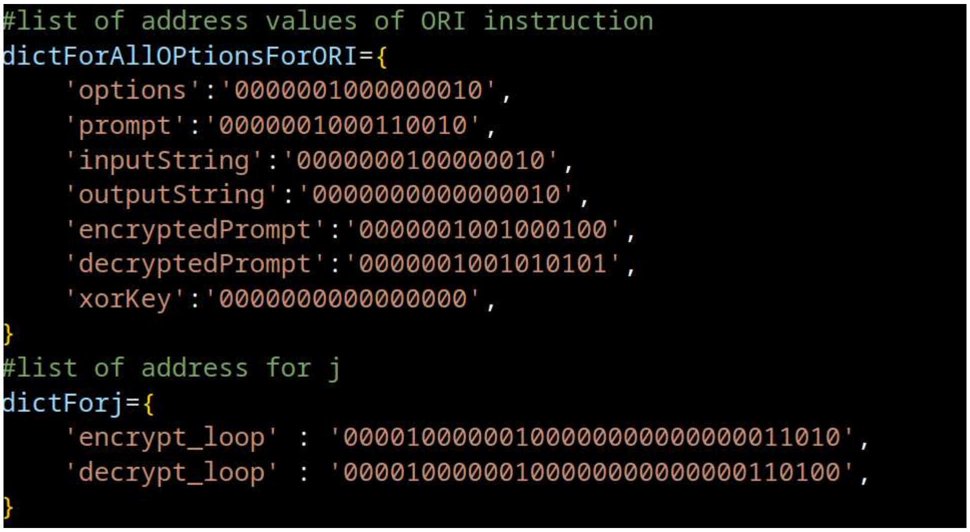
3. Question 2 aka Assembler

Here we have made our own assembler which produced the same output on doing the code dump to the hexa-decimal values.

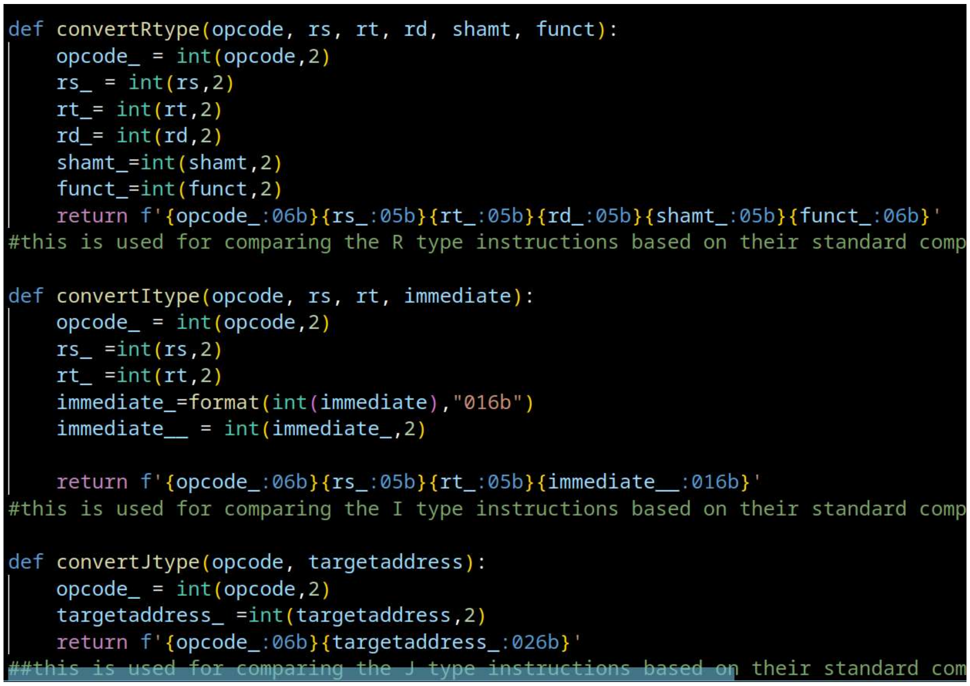
ASSUMPTIONS TAKEN->

1. Here for some instructions especially the I type instructions their address values were characterised by 16 digit address values which was subjected to the MARS simulator only so for that we made a dictionary of their binary addresses.
2. We have not consider the cases for the comments as the semicolon starting ones.
3. We have taken 'lui' command as a definite hard coded because its value was same throughout the MARS output.
4. 'ori' address values had to be mapped through a dictionary and so for the 'j' type of instruction that was used.
5. Lastly all commas and backspaces were stripped while reading the file
6. The code is passed through the main code only in quotes.

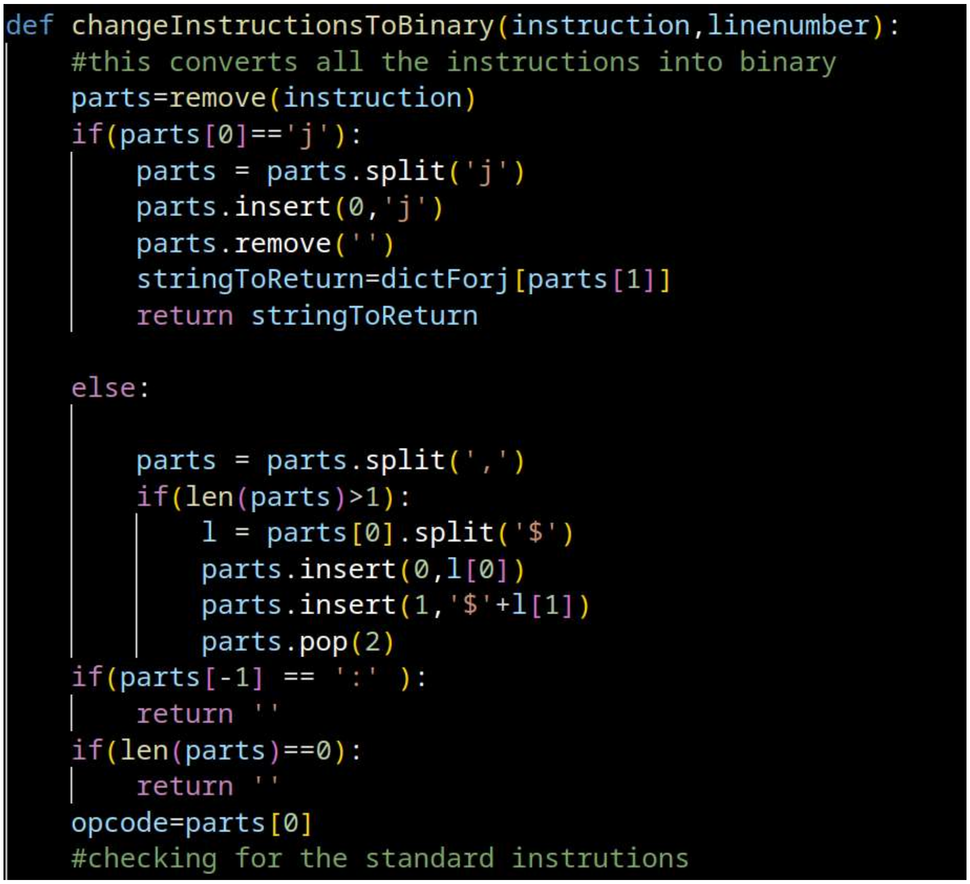
7. Some beq conditions and preconditions as well as pseudo instructions were also considered.



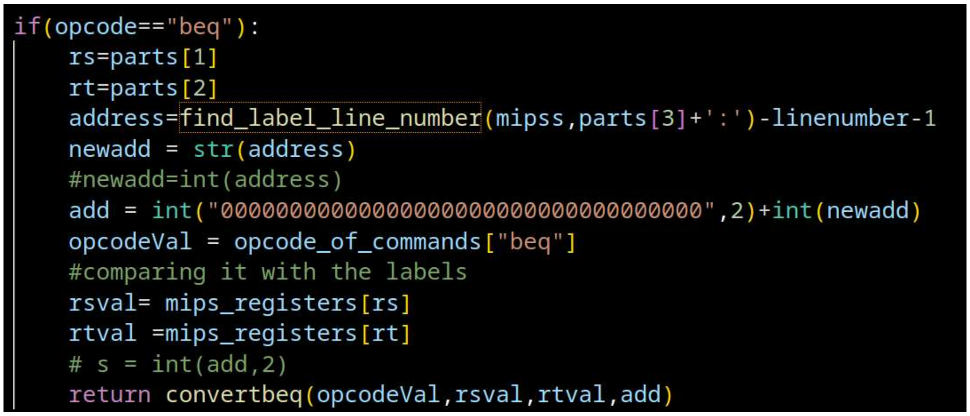
```
#list of address values of ORI instruction
dictForAllOPtionsForORI={
    'options':'0000001000000010',
    'prompt':'0000001000110010',
    'inputString':'0000000100000010',
    'outputString':'0000000000000010',
    'encryptedPrompt':'0000001001000100',
    'decryptedPrompt':'0000001001010101',
    'xorKey':'0000000000000000',
}
#list of address for j
dictForj={
    'encrypt_loop' : '00001000000100000000000000011010',
    'decrypt_loop' : '00001000000100000000000000110100',
}
```



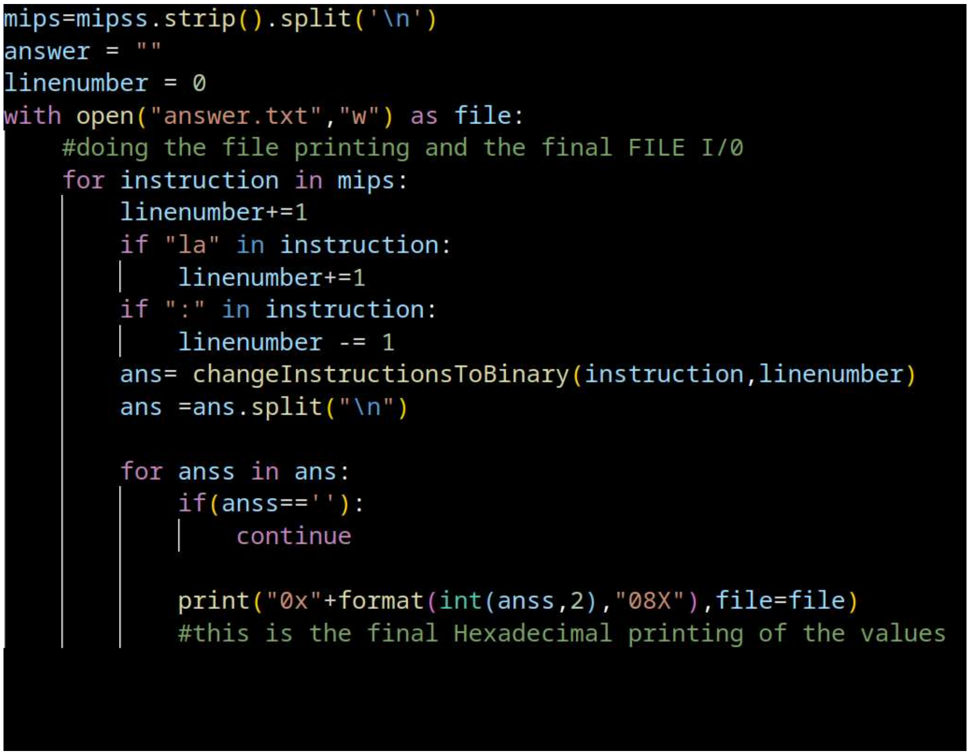
```
def convertRtype(opcode, rs, rt, rd, shamt, funct):
    opcode_ = int(opcode,2)
    rs_ = int(rs,2)
    rt_= int(rt,2)
    rd_= int(rd,2)
    shamt_=int(shamt,2)
    funct_=int(funct,2)
    return f'{opcode_:06b}{rs_:05b}{rt_:05b}{rd_:05b}{shamt_:05b}{funct_:06b}'
#this is used for comparing the R type instructions based on their standard comp

def convertItype(opcode, rs, rt, immediate):
    opcode_ = int(opcode,2)
    rs_ =int(rs,2)
    rt_ =int(rt,2)
    immediate_=format(int(immediate),"016b")
    immediate__ = int(immediate_,2)

    return f'{opcode_:06b}{rs_:05b}{rt_:05b}{immediate__:016b}'
#this is used for comparing the I type instructions based on their standard comp

def convertJtype(opcode, targetaddress):
    opcode_ = int(opcode,2)
    targetaddress_ =int(targetaddress,2)
    return f'{opcode_:06b}{targetaddress_:026b}'
##this is used for comparing the J type instructions based on their standard com
```



```
def changeInstructionsToBinary(instruction,linenumber):
    #this converts all the instructions into binary
    parts=remove(instruction)
    if(parts[0]=='j'):
        parts = parts.split('j')
        parts.insert(0,'j')
        parts.remove('')
        stringToReturn=dictForj[parts[1]]
        return stringToReturn

    else:

        parts = parts.split(',')
        if(len(parts)>1):
            l = parts[0].split('$')
            parts.insert(0,l[0])
            parts.insert(1,'$'+l[1])
            parts.pop(2)
    if(parts[-1] == ':' ):
        return ''
    if(len(parts)==0):
        return ''
    opcode=parts[0]
    #checking for the standard instrutions
```

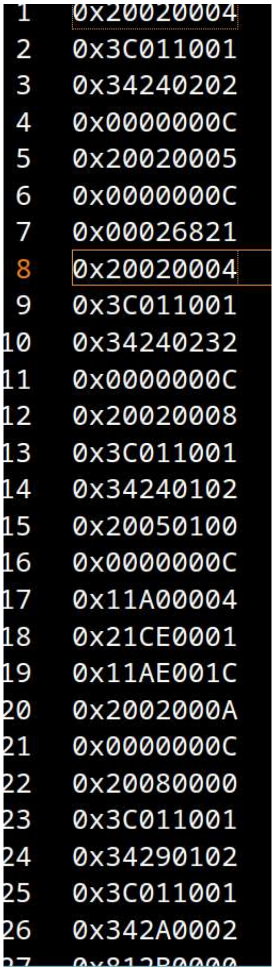


```
if(opcode=="beq"):
    rs=parts[1]
    rt=parts[2]
    address=find_label_line_number(mipss,parts[3]+':')-linenumber-1
    newadd = str(address)
    #newadd=int(address)
    add = int("00000000000000000000000000000000000",2)+int(newadd)
    opcodeVal = opcode_of_commands["beq"]
    #comparing it with the labels
    rsval= mips_registers[rs]
    rtval =mips_registers[rt]
    # s = int(add,2)
    return convertbeq(opcodeVal,rsval,rtval,add)
```

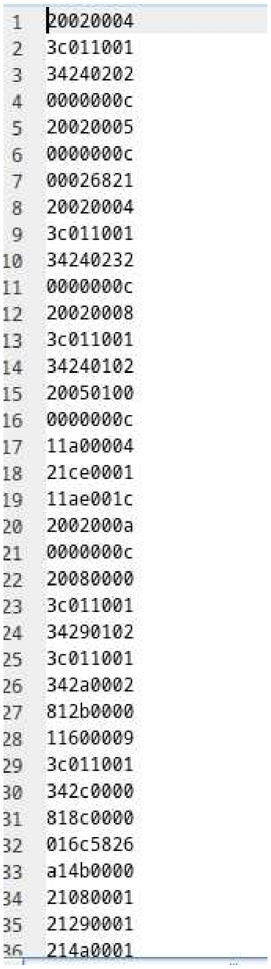


```
mips=mipss.strip().split('\n')
answer = ""
linenumber = 0
with open("answer.txt","w") as file:
    #doing the file printing and the final FILE I/0
    for instruction in mips:
        linenumber+=1
        if "la" in instruction:
            linenumber+=1
        if ":" in instruction:
            linenumber -= 1
        ans= changeInstructionsToBinary(instruction,linenumber)
        ans =ans.split("\n")

        for anss in ans:
            if(anss==''):
                continue

            print("0x"+format(int(anss,2),"08X"),file=file)
            #this is the final Hexadecimal printing of the values
```

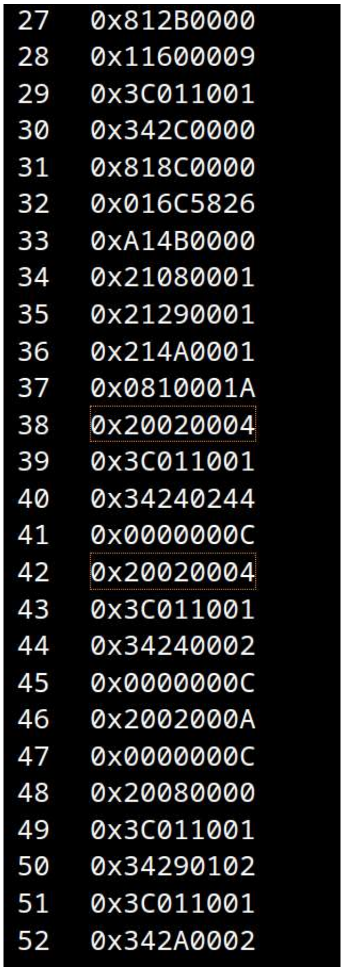
Our FINAL OUTPUT MATCHES WITH THE MARS OUTPUT



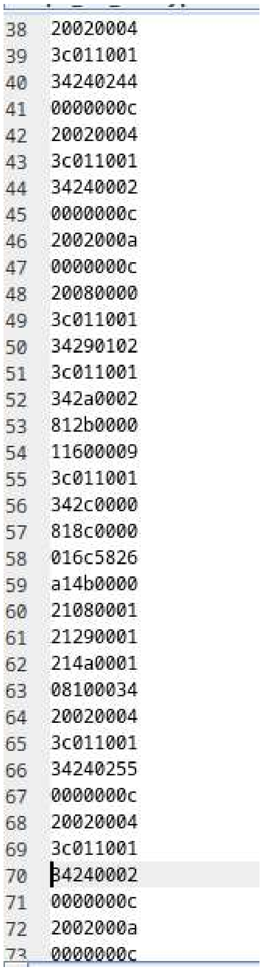
```
0x20020004
0x3C011001
0x34240202
0x0000000C
0x20020005
0x0000000C
0x00026821
0x20020004
0x3C011001
0x34240232
0x0000000C
0x20020008
0x3C011001
0x34240102
0x20050100
0x0000000C
0x11A00004
0x21CE0001
0x11AE001C
0x2002000A
0x0000000C
0x20080000
0x3C011001
0x34290102
0x3C011001
0x342A0002
0x812B0000
```



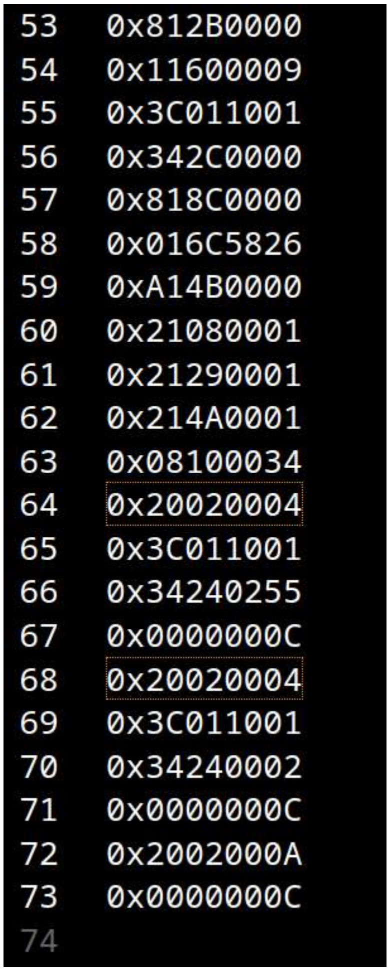
```
20020004
3c011001
34240202
0000000c
20020005
0000000c
00026821
20020004
3c011001
34240232
0000000c
20020008
3c011001
34240102
20050100
0000000c
11a00004
21ce0001
11ae001c
2002000a
0000000c
20080000
3c011001
34290102
3c011001
342a0002
812b0000
11600009
3c011001
342c0000
818c0000
016c5826
a14b0000
21080001
21290001
214a0001
```



```
27   0x812B0000
28   0x11600009
29   0x3C011001
30   0x342C0000
31   0x818C0000
32   0x016C5826
33   0xA14B0000
34   0x21080001
35   0x21290001
36   0x214A0001
37   0x0810001A
38   0x20020004
39   0x3C011001
40   0x34240244
41   0x0000000C
42   0x20020004
43   0x3C011001
44   0x34240002
45   0x0000000C
46   0x2002000A
47   0x0000000C
48   0x20080000
49   0x3C011001
50   0x34290102
51   0x3C011001
52   0x342A0002
```



```
38 20020004
39 3c011001
40 34240244
41 0000000c
42 20020004
43 3c011001
44 34240002
45 0000000c
46 2002000a
47 0000000c
48 20080000
49 3c011001
50 34290102
51 3c011001
52 342a0002
53 812b0000
54 11600009
55 3c011001
56 342c0000
57 818c0000
58 016c5826
59 a14b0000
60 21080001
61 21290001
62 214a0001
63 08100034
64 20020004
65 3c011001
66 34240255
67 0000000c
68 20020004
69 3c011001
70 34240002
71 0000000c
72 2002000a
73 0000000c
```



```
0x812B0000
0x11600009
0x3C011001
0x342C0000
0x818C0000
0x016C5826
0xA14B0000
0x21080001
0x21290001
0x214A0001
0x08100034
0x20020004
0x3C011001
0x34240255
0x0000000C
0x20020004
0x3C011001
0x34240002
0x0000000C
0x2002000A
0x0000000C
```