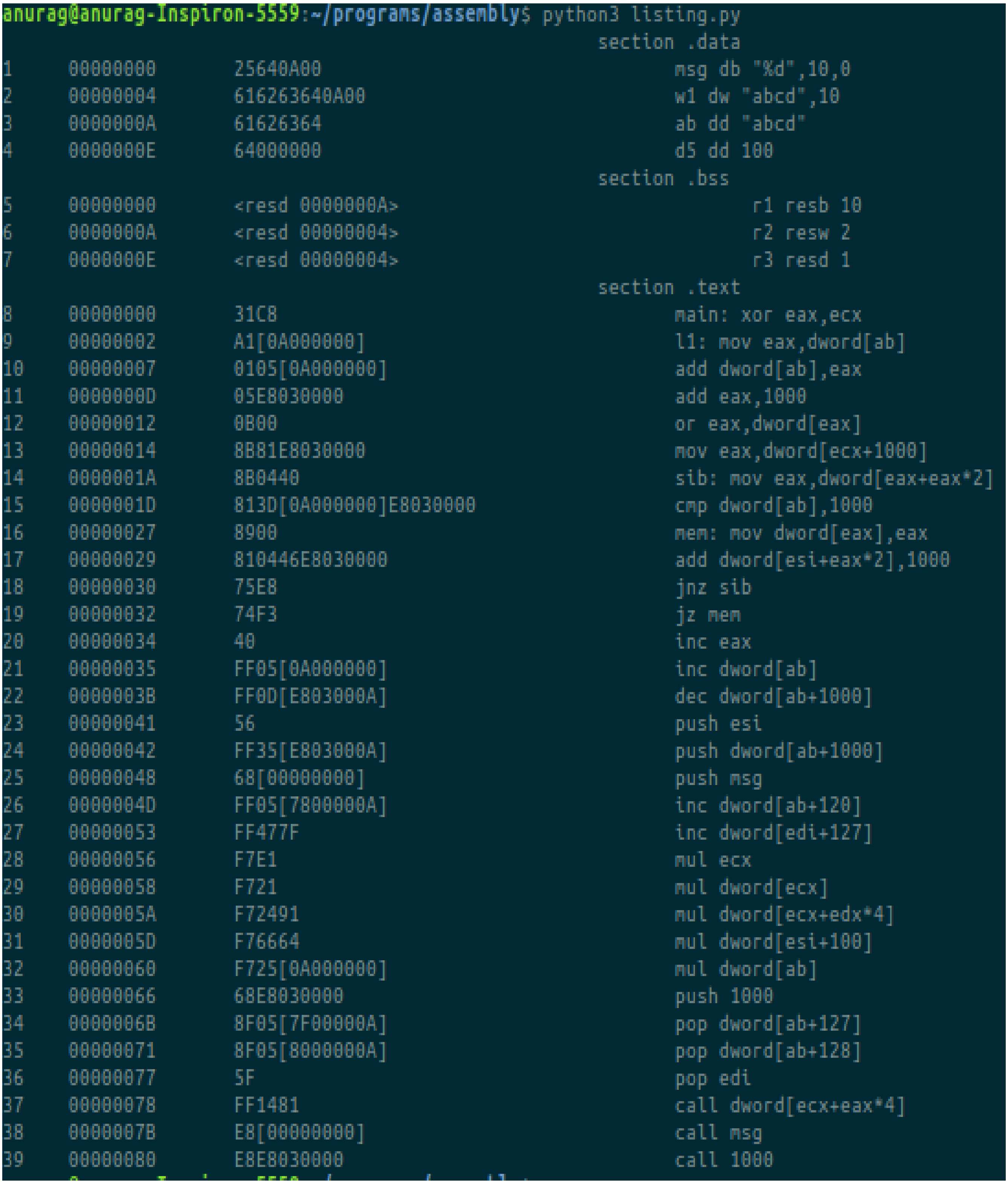
ASSIGNMENT-2 SYSTEMS-02

The file will help the reader to grasp the idea of how the symbol table is getting generated from the input file of type .asm and what sequence of programming pardigms have been taken into account.

- The input file is read line by line by the python program, at first .data and .bss section is focused.
- The instruction/ line is decoded by getting the variable, type of variable involved and the size of address it will take to get represented on the symbol table.
- We find the binary sequence of hexadecimal codes that represent the converted instructions by nasm.
- These extracted fields then are inserted into a linked list as node having all those arguments.
- Taking about the decoding of text section, I processed it line by line.
- If there was a label,that was inserted into records.
- A profile was made of every instruction that renaming the instruction as per the format in set ‘op’. Example **mov eax,23** was made **mov reg,imm**
- This profiling helped me to hit the right key in *‘op’ set in opcodes.py* and get the correct opcode and coversion instruction from the set.
- Having hexadecimal codes of all the lines in text section , we counted each byte from them and assigned address to each instruction.
- Jump statements were left unattended and handled afterwards.Wherever a jump statement was found, its corresponding label was located and the byte distance was counted to get the hexadecimal code.
- Every instruction address was obtained by keeping track of its previous address i.e. adding number of bytes of hexadecimal code to the previous address.

-ANURAG PUNDIR
(MCA-R19112033)

Output of the program (generating output as in .lst file my nasm):



```
anurag@anurag-Inspiron-5559:~/programs/assembly$ python3 listing.py
                                                      section .data
1     00000000       25640A00                                msg db "%d",10,0
2     00000004       616263640A00                            w1 dw "abcd",10
3     0000000A       61626364                                ab dd "abcd"
4     0000000E       64000000                                d5 dd 100
                                                      section .bss
5     00000000       <resd 0000000A>                                r1 resb 10
6     0000000A       <resd 00000004>                                r2 resw 2
7     0000000E       <resd 00000004>                                r3 resd 1
                                                      section .text
8     00000000       31C8                                    main: xor eax,ecx
9     00000002       A1[0A000000]                            l1: mov eax,dword[ab]
10    00000007       0105[0A000000]                          add dword[ab],eax
11    0000000D       05E8030000                              add eax,1000
12    00000012       0B00                                    or eax,dword[eax]
13    00000014       8B81E8030000                            mov eax,dword[ecx+1000]
14    0000001A       8B0440                                  sib: mov eax,dword[eax+eax*2]
15    0000001D       813D[0A000000]E8030000                  cmp dword[ab],1000
16    00000027       8900                                    mem: mov dword[eax],eax
17    00000029       810446E8030000                          add dword[esi+eax*2],1000
18    00000030       75E8                                    jnz sib
19    00000032       74F3                                    jz mem
20    00000034       40                                      inc eax
21    00000035       FF05[0A000000]                          inc dword[ab]
22    0000003B       FF0D[E803000A]                          dec dword[ab+1000]
23    00000041       56                                      push esi
24    00000042       FF35[E803000A]                          push dword[ab+1000]
25    00000048       68[00000000]                            push msg
26    0000004D       FF05[7800000A]                          inc dword[ab+120]
27    00000053       FF477F                                  inc dword[edi+127]
28    00000056       F7E1                                    mul ecx
29    00000058       F721                                    mul dword[ecx]
30    0000005A       F72491                                  mul dword[ecx+edx*4]
31    0000005D       F76664                                  mul dword[esi+100]
32    00000060       F725[0A000000]                          mul dword[ab]
33    00000066       68E8030000                              push 1000
34    0000006B       8F05[7F00000A]                          pop dword[ab+127]
35    00000071       8F05[8000000A]                          pop dword[ab+128]
36    00000077       5F                                      pop edi
37    00000078       FF1481                                  call dword[ecx+eax*4]
38    0000007B       E8[00000000]                            call msg
39    00000080       E8E8030000                              call 1000
```