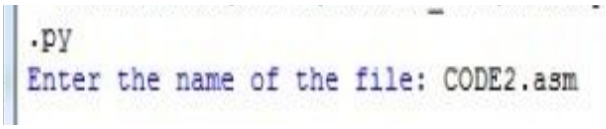
TWO PASS ASSEMBLER

HOW TO GET STARTED?

Open command prompt in your pc and run the file ‘cofinal.py’ in the command line.
The follow up to this would be the running program asking for an input file and so will be prompted by ‘File Name: ’. Here, write the name of the input file containing the assembly program provided that the address of the file which has the code and the input code is the same. The output will be generated at the same location as the input file.



FUNCTIONS IN THE PROGRAM..

The functions used are:
notExists:function to check if the symbol already exists or not

cal_Bytes: returns the number of bytes in decimal format

returnOpcode: this function will return the opcode of the operand

returnASCII: this function will return the ascii code of the character

returnAddress: this function will return the address of the label

FIRST PASS..

In the first pass, the whole input file is traversed. While doing so, two files, ‘SYMTAB.txt’ and ‘aCODE.txt’ are written up by the program. ‘SYMTAB.txt’ stores all the symbols or labels in the input code whereas ‘aCODE.txt’ stores the addresses of various variables declared and the opcodes of the operations.

ASSUMPTIONS: The input code uses attributes like ‘RESW’(register to store variable), ‘WORD’(variable) , ‘BYTE’(address), ‘RESB’(register to store an address) , ‘X’ . This is done while declaring the variables or registers so that a particular memory location can be allocated to them. The memory is a randomly generated address.

Assembler Directive statements like ‘START’ or ‘END’ are not converted to a machine code.

SECOND PASS..

In the second pass, the file ‘aCODE.txt’ generated above is read. Now, new files ‘objCODE.txt’ and ‘sic.o’ are written. ‘sic.o’ is the final output file with only object codes. Whereas, ‘objCODE.txt’ is the file which has both assembly code and object codes in it.

‘aCODE.txt’ is traversed and the instructions are converted to machine code as per the opcode provided in the ‘OPCODE.txt’ file.

ERROR HANDLING..

The code throws an error for various coding errors like:

1. If a variable used is not declared, an error saying
   "Error: Direct address of (mnemonic name) could not be found"

2. If the opcode in the input file is not present in the reference ‘OPCODE.txt’ , an error saying
   “Error: Opcode for (mnemonic name) could not be found” pops up.
3. If a symbol used for declaration more than once, an error saying
   “Error: (symbol name)- Multiple declaration’ pops up.”

4. If the referred variable or address is not of the type that the program can read(example, if it is not a WORD,RESW,OPCODE etc), an error saying
   “Error: Invalid mnemonic (mnemonic name)’ pops up.”

FILES USED:

The files read and used in the code are:

1. OPCODE.txt
2. ASCII.txt :to generate ascii code of a character for object coding for mnemonics like ‘WORD’ and ‘RESW’.

The files written in the code are:

1. SYMTAB.txt :for labels and symbols
2. aCODE.txt :for addresses
3. objCODE.txt :for object codes and assembly code
4. sic.o :for object code

The sample inputs provided:

1. CODE1.txt :obtaining product of a and b
2. CODE2.txt :obtaining the bigger number of a and b
3. CODE3.txt :obtaining product of a and b by taking input from user

INPUT-OUTPUT..

1. CODE1:

input

```
SUM     START   0000
FIRST   LAC     B
        MUL     A
        SAC     COUNT
        DSP     COUNT
A       WORD    4
B       WORD    5
COUNT   RESW    50
        END
```

output

```
0000    0001 1111
0011    1010 1100
0110    0010 10010
1001    1001 10010
```

2. CODE2:

input

```
SUM     START   0000
FIRST   LAC     B
        SUB     A
        BRP     L2
        BRN     L1
L1      DSP     A
L2      DSP     B
A       WORD    5
B       WORD    6
        END
```

output

```
0000    0001 10101
0011    0100 10010
0110    0111 1111
1001    0110 1100
1100    1001 10010
1111    1001 10010
```

3. CODE3:

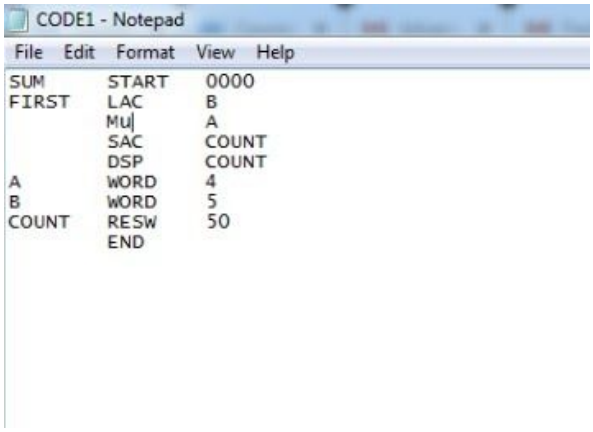
input

```
SUM     START   0000
FIRST   INP     A
        INP     B
        LAC     B
        MUL     A
        SAC     COUNT
        DSP     COUNT
A       RESB    30
B       RESB    40
COUNT   RESW    50
        END
```

output

```
0000    1000 10010
0011    1000 110000
0110    0001 110000
1001    1010 10010
1100    0010 1011000
1111    1001 1011000
```

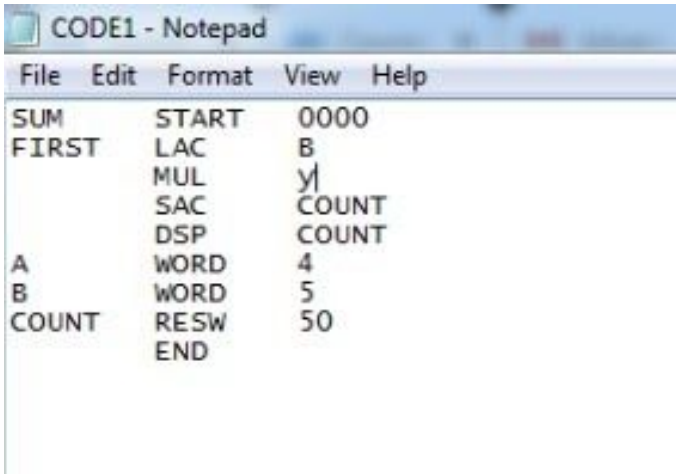
4.INPUT



OUTPUT

```
Error: Invalid mnemonic Mu
```

5.INPUT



OUTPUT

```
Error: Directed address of y not found
```

6.INPUT

File Edit Format View Help

```
SUM     START   0000
FIRST   LAC     B
        MUL     A
        SAC     COUNT
        DSP     COUNT
A       WORD    4
B       WORD    5
A       WORD    24
COUNT   RESW    50
        END
```

OUTPUT

```
Error: A - Multiple declaration
```

CODE..

1. assemF is the file with the input assembly code. addF is the variable through which aCode.txt is accessed and written. labelF is the variable through which SYMTAB.txt is accessed and written.

```
175  FileName=input("File name: ")
176
177
178  assemF=open(FileName,"r")
179  #assembly code file
180  addF=open("aCODE.txt","w") #harshal
181  #file that stores addresses in assembly code
182  labelF = open("SYMTAB.txt","w")#harshal
183  #file that stores labels in the assembly code
184
185  begin = assemF.readline();
186  #assumption: first line of the code is the start statement
187
```

2. the value of address where the program begins is stored in decimal

```
193
194   Line1 = begin.split('\t')
195   #print (Line1)
196
197   addHex = Line1[2]   #value of address in hex #addf
198   add = int(addHex,16)    #converted the value in decimal
199   addHD= addHex       #duplicate containing hex value #add1
200   #
```

3. error reporting for labels and symbols done and the symbols are written to SYMTAB.txt.

```
# ENTRY INTO SYMTAB  #harshal
if(line1[0]!=''):
    u=True                  #if label is not empty
    if(notExists(line1[0],-1,u)):       #checks if the label is already present or not  #harshal
        sym = line1[0] + "\t" + addHD    #store the zeroth element and the second element in the sym tab or else report the error



        #print(symbol)
        labelF.write(sym)
        labelF.write("\n")
        labelF.flush()
    else:
        error= "Error: " + line1[0] + " - Multiple declaration "
        print(error)
        input()
        exit(0)
```

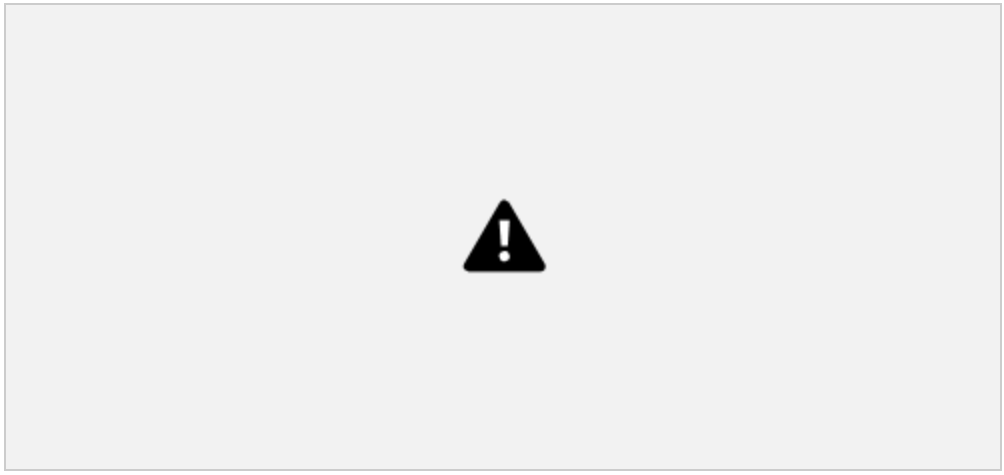
4. the addresses are written into aCODE.txt along with the instructions.

```
        #WRITING INSTRUCTIONS ALONG WITH ASSIGNED ADDRESSES
        lineWrite= addHD + "\t" + line      #check if \n is a part of line  #Add1 is hex
        #print(writeLine)                   #PRINTS ON-SCREEN
        addF.write(lineWrite)               #writes into file  #harshal
        addF.write("\n")
        addF.flush()

        #CALCULATION OF NEXT ADDRESS
        add = add + nBytes              #performs decimal addition
        addHD="{0:b}".format(add)    #str(format(add,'04x'))    #converts to hex before storing 4x
        length = 4-len(addHD)
        addHD = "0"*length + addHD


labelF.close()
assemF.close()
addF.close()
```

5. error reporting for incorrect direct addressing and also obtaining the machine code for given inputs done.

```
    opsplit = operand.split(',')
    length = len(opsplit)

    dirAdd =returnAddress(-1,opsplit[0])
    if(dirAdd==-1):
        error="Error: Directed address of " + opsplit[0] + " not found"
        print(error)
        input()
        exit(0)


    if(length==2 and opsplit[1]=="X"):
        string=dirAdd
        a = string[:1]
        b = string[1:]

        a = int(a)
        a = a + 8
        a = "{0:b}".format(a)#str(format(a,'#010b'))[2:]
        length = 4-len(a)

        a="0"*length + a


        dirAdd = a+b

    objLine = opcode + " "+  dirAdd
```

6. object coding for declarative statements like 'WORD' and 'RESW' done. Further, the final machine code is written into objCODE.txt and sic.o



7. FINAL MACHINE CODE PRINTED AS OUTPUT.

```
416
417  objCode = open("objCODE.txt","r")
418
419  for i in objCode:
420      line = i[:-1]
421      print(line)
422
423  #output file with apt addresses and codes generated
424  #*************************************************************************#
425  get = input()
426
```

SUBMITTED BY:
HARSHAL DEV (2019306)
ISHIKA JOSHI (2019310)