Experiment no 1

```
from sys import exit
import operator as op
motOpCode = {
   "MOV": 1,"A": 2,   "S": 3,   "M": 4,   "D": 5,   "AN": 6,   "O": 7,   "STOP": 0,   "ADD": 1,   "SUB": 2
"MULT": 3,   "MOVER": 4,   "MOVEM": 5,   "COMP": 6,   "BC": 7,   "DIV": 8,   "READ": 9, "PRINT": 10
   "MOVER": 4
}
motSize = {
   "MOV": 1, "A": 1,   "S": 1,   "M": 1,   "D": 1,   "AN": 1,   "O": 1,   "STOP": 1,   "ADD": 1,   "SUB": 1,
"MULT": 1,  "MOVER": 1, "MOVEM": 1,   "COMP": 1,   "BC": 1,   "DIV": 1,   "READ": 1,   "PRINT": 1
}
potOpCode = {
   "START": 1,   "END": 2,   "ORIGIN": 3,   "EQU": 4,   "LTORG": 5,
}
potSize = {
   "START": 1,   "END": 1,   "ORIGIN": 1,   "EQU": 1,   "LTORG": 1,
}
RegOpCode = {
   "AREG": 1,   "BREG": 2,   "CREG": 3,   "DREG": 4
}
BCOpCode = {
   "LT": 1,   "LE": 2,   "EQ": 3,   "GT": 4,   "GE": 5,   "ANY": 6
}
BCOpSize = {
   "LT": 1,   "LE": 1,   "EQ": 1,   "GT": 1,   "GE": 1,   "ANY": 1
}
RegOpSize = {
   "AREG": 1,   "BREG": 1,   "CREG": 1,   "DREG": 1
}
DecOpCode = {
   "DC": 1,   "DS": 2
}

DecSize = {
   "DC": 1,   "DS": 1
}

symbolTable = {
   "N" : 1
}
l = []
relativeAddress = []
machineCode = []
symbolCode = []
symboladd = []
literalCode = []
literaladd = []
s = '='
ltorgIdx = 0
ltorgIdx1 = 0
```

```
RA = 0
current = 0
count = 0
j = "0"
PrevLoopRA = 0
PrevNextRA = 0
found = ''
DcFound = ''
n = int(input("Enter the no of instruction lines : "))
for i in range(n):
   instructions = input("Enter instruction line {} : ".format(i + 1))
   l.append(instructions)
l = [x.upper() for x in l]              # Converting all the instructions to upper case
for i in range(n):
   x = l[i]
   if " " in x:
      s1 = ''.join(x)
      a, b, c, d  = s1.split()
      if a == 'NULL':
         RA = int(c)
         relativeAddress.append(RA)
         out =  str(RA)
         machineCode.append(out)
         continue
      if a == 'LOOP':
         LoopRA = RA
         PrevLoopRA = LoopRA
         symbolCode.append(a)
         symboladd.append(RA)
      if a == 'NEXT':
         NextRA = RA
         PrevNextRA = NextRA
         symbolCode.append(a)
         symboladd.append(RA)

      if b in motOpCode:            # Checking if Mnemonics is present in MOT or not
          value = motOpCode.get(b)
          size = motSize.get(b)
          previous = size
          #RA += current
          current = previous
          RA += current
          relativeAddress.append(RA)
          found = 'xx'
      else:
         found = x
      if found == x:
         if b in potOpCode:
            value = potOpCode.get(b)
            size = potSize.get(b)
            previous = size
```

```
            #RA += current
            current = previous
            if b == 'ORIGIN' and c == 'LOOP':
                RA = PrevLoopRA + 5
            RA += current
            if b == 'ORIGIN' and c == 'LOOP':
              tempra = RA
              RA = 0
              relativeAddress.append(RA)
              RA = tempra - 1 # To ignore the latest incremnet value
            if b == 'ORIGIN' and c == 'NEXT':
              tempra = RA
              RA = 0
              relativeAddress.append(RA)
              RA = tempra - 1 # To ignore the latest incremnet value
            if b == 'END':
              RA = int(endRA) + 1
              relativeAddress.append(RA)
            else :
               if (c == "'" and b == 'LTORG') or (c == '' and b == 'END') or (b == 'ORIGIN' and c == 'NEXT'):
                  temp = x #Do nothing
               else:
                  relativeAddress.append(RA)
        elif b in DecOpCode:
            value = DecOpCode.get(b)
            size = DecSize.get(b)
            RA += 1
            relativeAddress.append(RA)
            DcFound = x
        else:
            print("Instruction is not in Op Code Table.")
            exit(0)

    if b in potOpCode:
        value = potOpCode.get(b)
        size = potSize.get(b)
if c in RegOpCode:
        valueReg = RegOpCode.get(c)
        sizeReg = RegOpSize.get(c)
        valueBC = 'None'
    else:
        if c == 'LOOP':
            valueReg = '' #LoopRA
        elif c == 'NEXT' and b == 'ORIGIN':
            valueReg = PrevNextRA + 3
            valueReg = ' '
        else:
            if c in BCOpCode:
                valueBC = BCOpCode.get(c)
                sizeBC = BCOpSize.get(c)
                valueReg = 'None' #to make sure valureg id defined
```

```
            elif DcFound == x:
                temp = x  #Do nothing
            else:
               if b != 'LTROG' and c != "'":
                  print("Instruction is not in Reg Code Table.")
                  exit(0)                                  # EXIT if Mnemonics is not in MOT
      if a in symbolTable:
         symbolCode.append(a)
         symboladd.append(RA)
      if b == 'LTORG':
         ltorgIdx = i
         ltorgIdx1 = ltorgIdx + 1
      if(op.contains(d,s)):
        literalCode.append(d)
      var1 = '0' # to prefix opcode with zeros
      if d.isalpha() is True:
         if valueReg != 'None':
            valueReg = j + str(valueReg)
            value = j + str(value)
            machineCode.append(str(value) + "," + str(valueReg))
            endRA = RA
         elif valueBC != 'None':
            valueBC = j + str(valueBC)
            value = j + str(value)
            machineCode.append(str(value) + "," + str(valueBC))
            endRA = RA
      else:
         if valueReg != 'None':
            valueReg = j + str(valueReg)
            value = j + str(value)
            machineCode.append(str(value) + "," + str(valueReg))
            endRA = RA
         elif valueBC != 'None':
            valueBC = j + str(valueBC)
            value = j + str(value)
            machineCode.append(str(value) + "," + str(valueBC))
            endRA = RA
   else:
      if x in motOpCode:
         value = motOpCode.get(x)
         size = motSize.get(x)
         previous = size
         RA += current
         current = previous
         relativeAddress.append(RA)
         machineCode.append(value)
      else:
         print("Instruction is not in Op Code Table.")
         exit(0)
print("-------------------------------------------")
print("Relative Address          Instruction          OpCode")
```

```
for i in range(n):
   if i == ltorgIdx:
     literaladd.append(relativeAddress[i])
   #if i == ltorgIdx1:
     literaladd.append(relativeAddress[i]+1)
   print(
      "{}              {}            {}".format(relativeAddress[i], l[i], machineCode[i]))
print("-----------------------------------------")
m = len(symbolCode)
print("Symbol          Address")
for i in range(m):
   print(
      "{}             {}".format(symbolCode[i] , symboladd[i]))
print("-----------------------------------------")
m = len(literaladd)
print("Literal          Address")
for i in range(m):
   print(
      "{}             {}".format(literalCode[i] , literaladd[i]))
```

Input:

```
null start 200 ''
'' mover breg ='4'
loop mover areg n
'' add breg ='2'
'' origin loop +5
NEXT BC ANY LOOP
'' LTORG '' ''
'' origin next +3
n DC 5 ''
'' end '' ''
```

Output:

```
-----------------------------------------
Relative Address                Instruction              OpCode
200                   NULL START 200 ''                 200
201                   '' MOVER BREG ='4'                  04,02
202                   LOOP MOVER AREG N                 04,01
203                   '' ADD BREG ='2'                  01,02
0                   '' ORIGIN LOOP +5                   03,0
206                   NEXT BC ANY LOOP                  07,06
207                   '' LTORG '' ''                  05,006
0                   '' ORIGIN NEXT +3                   03,0
209                   N DC 5 ''                 01,00
210                   '' END '' ''                02,000
-----------------------------------------
Symbol            Address
LOOP                201
NEXT                206
N                 209
-----------------------------------------
Literal            Address
='4'                207
='2'                208
```