COA PROJECT

TWO PASS ASSEMBLER

00701032020 Chandana Kuntala
02701032020 Mahima Pasricha

FORMAT OF INSTRUCTION

Each instruction may include four fields:a label,an operation,an operand and a comment.

Format of instruction:

<label> ~ <opcode> <operand> ; <comment>

1.**Label field**:It is used for defining a symbol.Label field ends with a tilde(~),can consist of alphanumeric characters,and the first character cannot be a numerical character

If a duplicate label exists,an error message is shown.The label field must not be keywords (opcodes).

2.**Operation field**:It defines the operation to be done.Operation field occurs after the label field and must be preceded by one white space character.The operation field must contain opcodes from given table:

Meaning	Assembly Opcode
Clear accumulator	CLA
Load into accumulator from address	LAC
Store accumulator contents into address	SAC
Add address contents to accumulator contents	ADD
Subtract address contents from accumulator contents	SUB
Branch to address if accumulator contains zero	BRZ
Branch to address if accumulator contains negative value	BRN
Branch to address if accumulator contains positive value	BRP
Read from terminal and put in address	INP
Display value in address on terminal	DSP
Multiply accumulator and address contents	MUL
Divide accumulator contents by address content. Quotient in R1 and remainder in R2	DIV
Stop execution	STP

3.**Operand field:** This field specifies either the address or the data.The operand field, if required, must follow the operation field, and must be preceded by one white-space character.The variable name cannot be digits alone, it can be alphanumeric.
4.**Comment field:**It allows the programmer to document the software.The comment field is separated from the operand field (or from the operation field if no operand is required) by one

white-space character and begins with a ‘;’.The comment field can contain any ASCII characters.
5.Each instruction ends with a newline character,the next instruction starts from the next line.

CODE

```
reserved_opcodes = {"CLA":(0,0), "LAC":(1,1), "SAC":(2,1), "ADD":(3,1), "SUB":(4,1),
"BRZ":(5,1), "BRN":(6,1), "BRP":(7,1), "INP":(8,1), "DSP":(9,1), "MUL":(10,1),
"DIV":(11,1), "STP":(12,0)}

sym_table = {}
opc_table = []
error_table = []

with open('input.txt','r') as file:
 asm_file = file.readlines()

def print_sym_table():
 print("\n********* SYMBOL TABLE *********")
 print("---------------------------------")
 print("Symbol\t|\tLocation")
 print("---------------------------------")
 for symbol,location in sym_table.items():
   print(symbol," \t|\t",format(location, "08b"))
 print("")

def print_opc_table():
 print("********* OPCODE TABLE *********")
 print("---------------------------------")
 print(" Location\t|\tOpcode")
 print("---------------------------------")
 for i in opc_table:
   print(format(i[0], "08b"),"\t|\t",format(i[1], "04b"))
 print("")


def check_valid_variable(symbol,line_num):
   try:
       k = int(symbol[0])
       error_table.append([line_num,"ERROR! Symbol cannot start with a number in " +
symbol])
       return False
   except:
       pass
   if (symbol in reserved_opcodes):
       error_table.append([line_num,"ERROR! Opcode already used as a variable, try
using a different variable name"])
       return False
   valid_chars = '_abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'
   flag = 0
   for character in symbol:
       if character not in valid_chars:
           flag = 1
           error_table.append([line_num,"ERROR! Invalid character found in " +
character + " in symbol " + symbol])
   if flag == 1:
       return False
   return True

def check_comment(line):
   if (';' in line):
       return True
   else:
       return False

def check_symbol(line):
   if('~' in line):
       return True
   return False

def add_symbol(line, lc, line_num):
   symbol = line.split('~')[0]
   if (len(symbol) == 0):
       error_table.append([line_num,"ERROR! No symbol found"])
       return
   if (symbol in reserved_opcodes):
       error_table.append([line_num,"ERROR! Opcode used as a label already, try using
a different opcode "])
       return
   if(symbol in sym_table):
     if type(sym_table[symbol]) != type(5):
       if(sym_table[symbol][0] == -2):
         error_table.append([line_num,"ERROR! Variable " + symbol + " a label already,
try using a different variable"])
         return
```

```
     else:
       error_table.append([line_num,"ERROR! Symbol " + symbol +" has already been
declared"])
       return
   if(check_valid_variable(symbol, line_num)==False):
       return
   sym_table[symbol] = lc

def pass_one(loc_ctr):
 line_num = 0
 while line_num != len(asm_file):
   line = asm_file[line_num]
   if(check_comment(line)):
       line = line.split(';')[0]
       line = line.strip()
       if(len(line) == 0):
           line_num += 1
           continue
   if(check_symbol(line)):
     add_symbol(line, loc_ctr, line_num)
     line = line.split('~')[1]

   line = line.split()

   opCode = line[0]
   if (line_num == len(asm_file) - 1):
       if(opCode != 'STP'):
           error_table.append([line_num,"ERROR! Expected end statement STP at the
end"])
   if opCode in reserved_opcodes.keys():
     opc_table.append([loc_ctr, reserved_opcodes[opCode][0]])
   else:
       error_table.append([line_num,"ERROR! " + opCode + " opcode not recognized"])
   line = line[1:]

   if (opCode in reserved_opcodes):
       if reserved_opcodes[opCode][1] != len(line):
           error_table.append([line_num,"ERROR! " + opCode + " expects " +
str(reserved_opcodes[opCode][1]) + " number of arguments " + "but " + str(len(line)) +
" given"])

   for var in line:
     if var not in sym_table:
       if(check_valid_variable(var,line_num)==False):
           continue
       if ('BR' in opCode):
           sym_table[var] = (-1,line_num)
       else:
           sym_table[var] = (-2,line_num)
     else:
       if ('BR' in opCode and sym_table[var][0] == -2):
           error_table.append([line_num,"ERROR! Invalid jump location " + var + " since it's already used as a variable"])
       if('BR' not in opCode and sym_table[var][0] == -1):
           error_table.append([line_num,"ERROR! Invalid use of " + var + ", it has already been used as a jump location"])

   line_num += 1
   loc_ctr += 1
 return loc_ctr


def get_variables(lc):
 for symbol in sym_table:
   if type(sym_table[symbol]) == type((1,1)):
     if sym_table[symbol][0] == -2:
       sym_table[symbol] = lc
       lc += 1
     elif sym_table[symbol][0] == -1:
       error_table.append([sym_table[symbol][1],"ERROR: Label " + symbol  + " used but not defined"])
 return lc

def pass_two(ofile):
 line_num = 0
 while line_num != len(asm_file):
   line = asm_file[line_num]
   if(check_comment(line)):
       line = line.split(';')[0]
       line = line.strip()
       if(len(line) == 0):
           line_num += 1
           continue
   if(check_symbol(line)):
```

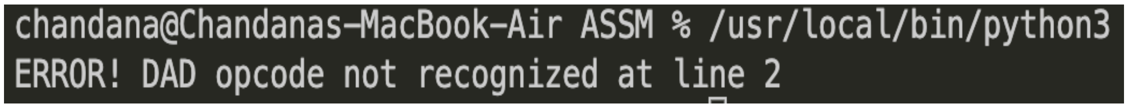
```python
     line = line.split('~')[1]
   line = line.split()
   opCode = line[0]
   if opCode in reserved_opcodes.keys():
     ofile.write(format(reserved_opcodes[line[0]][0], "04b"))
   line = line[1:]
   if(len(line) == 0):
     ofile.write(format(0,"08b"))
   else:
     for var in line:
       ofile.write(format(sym_table[var],"08b"))

   ofile.write("\n")
   line_num += 1



loc_ctr = pass_one(0)

get_variables(loc_ctr)
if(len(error_table) == 0):
 with open("output.txt","w+") as ofile:
   pass_two(ofile)
 print_sym_table()
 print('================================\n')
 print_opc_table()
else:
   for i in error_table:
       print (i[1],"at line " + str(i[0]+1))
```

INPUTS GIVEN

INPUT 1

CLA
DAD X

OUTPUT:

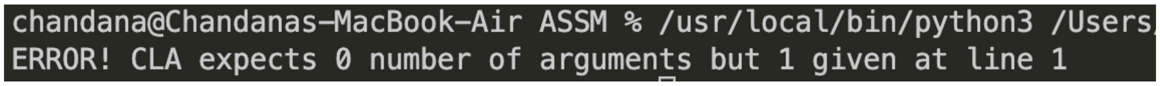


```
chandana@Chandanas-MacBook-Air ASSM % /usr/local/bin/python3
ERROR! DAD opcode not recognized at line 2
```

INPUT 2

CLA X
ADD X

OUTPUT:

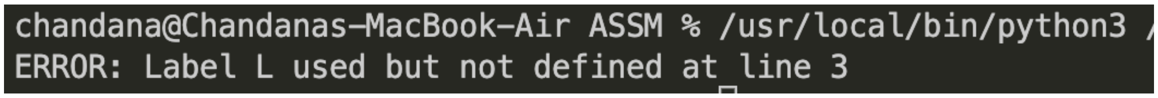


```
chandana@Chandanas-MacBook-Air ASSM % /usr/local/bin/python3 /Users
ERROR! CLA expects 0 number of arguments but 1 given at line 1
```

INPUT 3

CLA
ADD X
ADD Y
BRN L

OUTPUT:

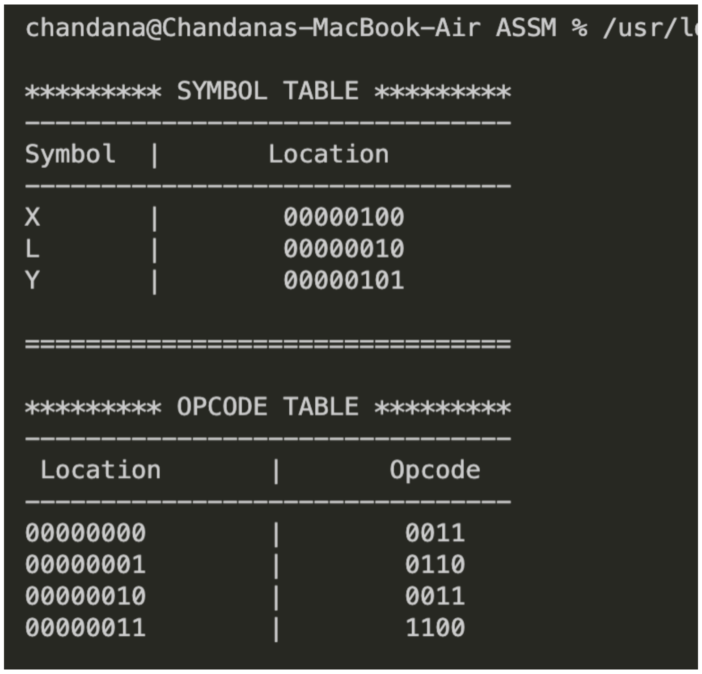


```
chandana@Chandanas-MacBook-Air ASSM % /usr/local/bin/python3
ERROR: Label L used but not defined at line 3
```

INPUT 4

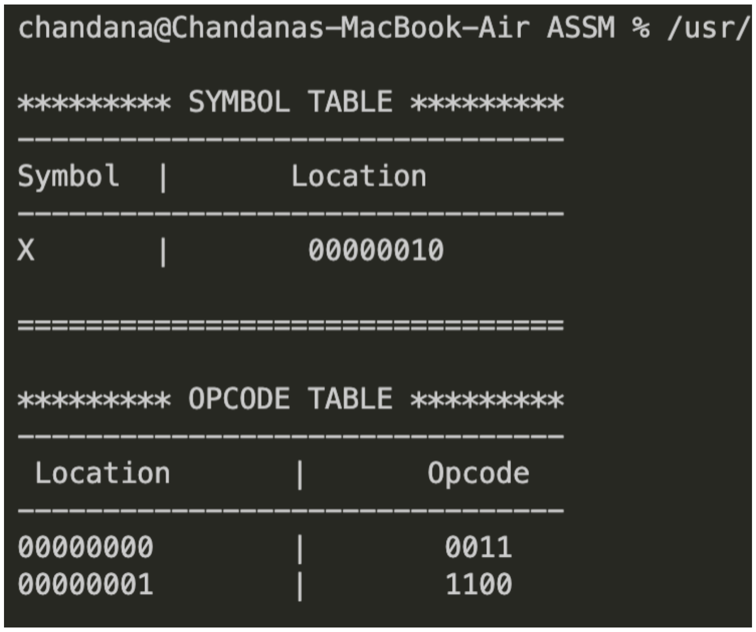
CLA
ADD X
L~ADD Y
BRN L

OUTPUT:



```
chandana@Chandanas-MacBook-Air ASSM % /usr/l

********* SYMBOL TABLE *********
--------------------------------
Symbol  |       Location
--------------------------------
X       |        00000100
L       |        00000010
Y       |        00000101

================================

********* OPCODE TABLE *********
--------------------------------
 Location       |       Opcode
--------------------------------
00000000        |        0011
00000001        |        0110
00000010        |        0011
00000011        |        1100
```

INPUT 5
ADD X ; This is to add 'x' to accumulator
OUTPUT:



```
chandana@Chandanas-MacBook-Air ASSM % /usr/

********* SYMBOL TABLE *********
--------------------------------
Symbol  |       Location
--------------------------------
X       |        00000010

================================

********* OPCODE TABLE *********
--------------------------------
 Location       |       Opcode
--------------------------------
00000000        |        0011
00000001        |        1100
```

ERROR REPORTING

1. Characters used in symbols should be valid characters. If invalid character is detected, an error is reported.
2. If no symbol is found before (~) error is thrown.
3. Symbols used should also be declared,otherwise an error is thrown.
4. A symbol should not begin with a number.
5. If a symbol name is used again, an error is reported.
6. Opcode used should be one of the predefined opcodes in the opcode table. Otherwise invalid opcode error is thrown.
7. Opcode cannot be used as a label or a variable.
8. A variable cannot be used as a label.
9. Opcodes should be given a valid number of inputs, depending on the type of opcode(0 argument or 1 argument).
10. Invalid jump location will also give error.
11. If STP opcode, denoting end of program, is not present, an error is shown.