Name: - Dev Chhawachharia
Roll Number:- 322014
PRN:- 22010381
TY B

Assignment 1

AIM: - Generate Symbol table, Literal table, Pool table & Intermediate code along with error table for first pass of a two-pass Assembler for the given source code.

Source code:

```
import math

import pandas as pd
import numpy as np
emot_table = [                ['STOP','01','00'],
                              ['ADD','01','01'],
                              ['SUB','01','02'],
                              ['MULT','01','03'],
                              ['MOVER','01','04'],
                              ['MOVEM','01','05'],
                              ['COMP','01','06'],
                              ['BC','01','07'],
                              ['DIV','01','08'],
                              ['READ','01','09'],
                              ['PRINT','01','10'],
                              ['START','03','01'],
                              ['END','03','02'],
                              ['ORIGIN','03','03'],
                              ['EQU','03','04'],
                              ['LTORG','03','05'],
                              ['DS','02', '01'],
                              ['DC','02','02'],
                              ['AREG','04','01'],
                              ['BREG','04','02'],
                              ['CREG','04','03'],
                              ['EQ','05','01'],
                              ['LT','05','02'],
                              ['GT','05','03'],
                              ['NE','05','04'],
                              ['LE','05','05'],
                              ['GT','05','06'],
                              ['ANY','05','07']]
emot_table_df = pd.DataFrame(emot_table,columns=['Mnemonic','Class','Opcode'])
class_field = [['Imperative Statements','IS','01'],
               ['Declarative Statements','DL','02'],
               ['Assembler Directive','AD','03'],
               ['CPU Register','RG','04'],
               ['Conditional Codes','CC','05'],]
class_field_df = pd.DataFrame(class_field , columns=['Type','Symbol','Value of
Class Field'])
# print(class_field_df)
# print(emot_table_df)

##########################################functions#################################
##############################

def check_token(token_lc):
```

```
    if(token_lc in emot_table_df.values):
        return True
    else:
        return False

def handle_token(token_lc):
    classType = emot_table_df.loc[emot_table_df['Mnemonic']==token_lc ,
'Class'].values[0]
    opcode = emot_table_df.loc[emot_table_df['Mnemonic']==token_lc ,
'Opcode'].values[0]
    symbol = class_field_df.loc[class_field_df['Value of Class Field']==classType ,
'Symbol'].values[0]
    return ("(",symbol," , ",opcode,")")

def handle_literal(token_lc):
    global ltp
    if not literal_table.isin([token_lc]).any().any():
        ltp+=1
        entry = [str(ltp),token_lc]
        literal_table.loc[ltp-1] = entry


def hanle_pool_table(token_lc):
    global ptp
    if ltp==1 or token_lc=='LTORG':
        pool_table.loc[ptp] = ltp
        if token_lc=='LTORG':
            pool_table.loc[ptp] = ltp+1
        ptp+=1



def address_literal(local_lc):
    address = [i for i in range(local_lc,local_lc+len(literal_table))]
    literal_table["Address"] = address




def handle_symbol(token_lc):
    global stp
    pos = line.index(token_lc)
    if  not symbol_table.isin([token_lc]).any().any():
        if pos <=1:
            entry = [token_lc , lc]
            symbol_table.loc[stp] = entry
            stp+=1


def change_pool():
    global final_literal_table,literal_table
    final_literal_table = pd.concat([final_literal_table, literal_table])
    value = len(literal_table)-1
    literal_table = pd.DataFrame(columns=["Number","Literal"])
    return  value

def intermediate_code_function(line):
    global entry1,id
    entry2 = [[] for x in range(2)]
    for i in range(len(line)):
        token = line[i]
        check = check_token(token)
        if check:
            value = handle_token(token)
            if value[1] == "IS" or value[1] == "DL" or value[1] == "AD":
                entry1 = value
            if value[1] == "RG":
```

```
                entry2[0] = token
                entry2[0] = ''.join(map(str, entry2[0]))

        else:
            if token.__contains__("=") and token.__contains__("\'"):
                numb
=final_literal_table.loc[final_literal_table['Literal']==token,'Number'].values
                # print(numb)
                list1.append(list(numb))
                # print((list1))
                count = math.ceil((list1.count(list(numb)))/len(pool_table))
                # print(count)
                if count>1:
                    entry2[1] = ("(","L",",",numb[count-1],")")
                    entry2[1] = ''.join(map(str, entry2[1]))
                else:
                    entry2[1] = ("(", "L", ",", numb[0], ")")
                    entry2[1] = ''.join(map(str, entry2[1]))
                # print(entry2)

            elif token.isnumeric():
                entry2[0] = " "
                entry2[1] = ("(", "C", ",", token, ")")
                entry2[1] = ''.join(map(str, entry2[1]))


            else:
                entry2[1] = (" ",token)
                entry2[1] = ''.join(map(str, entry2[1]))

    entry1 = ''.join(map(str,entry1))
    entry2 = ' '.join(map(str, entry2))
    # print(len(entry2))

    if len(entry2)==5:
        entry = [entry1," "]
    else:
        entry = [entry1,entry2]
    intermediate_code.loc[id] = entry
    # print(entry)
    id+=1







########################################driver
code######################################################################

lc =0
ltp =0
ptp = 0
stp  = 0
id=0
count =0
list1 =[]
f1 = open("input.txt",mode="rt")
symbol_table = pd.DataFrame(columns=['Symbol','Address'])
literal_table = pd.DataFrame(columns=['Number','Literal'])
final_literal_table = pd.DataFrame(columns=['Number','Literal','Address'])
pool_table = pd.DataFrame(columns=['Liternal Number'])
intermediate_code = pd.DataFrame(columns=['Type','Description'])
lc = int(f1.readline().split()[1])
f1.close()
# print(lc)
```

```
f1 = open("input.txt",mode="rt")
for x in f1:
    line = x.split()
    count+=1
    for token in line:
        check = check_token(token)
        if check:
            value = handle_token(token)
            if token == 'LTORG' or token == "END":
                address_literal(lc)
                value = change_pool()
                hanle_pool_table(token)
                if token == 'LTORG':
                    lc = lc + value
            if token == 'ORIGIN':
                nxt = line[line.index(token)+1]
                if symbol_table.isin([nxt]).any().any():
                    add =
symbol_table.loc[symbol_table['Symbol']==nxt,'Address'].values[0]
                    op = line[line.index(nxt)+1]
                    value = line[line.index(op)+1]
                    lc = eval(f'{add}{op}{value}')-1
            if token == 'EQU':
                prev = line[line.index(token)-1]
                nxt = line[line.index(token)+1]
                symbol_table.loc[symbol_table['Symbol'] == prev, 'Address']=
symbol_table.loc[symbol_table['Symbol']==nxt,'Address'].values[0]
            if token == 'DS':
                value = line[line.index(token)+1]
                lc = lc+int(value)-1

        else:
            if token.__contains__("=") and token.__contains__("\'"):
                handle_literal(token)
                hanle_pool_table(token)
            elif token.isnumeric():
                continue

            else:
                handle_symbol(token)
    if count==1:
        continue
    lc+=1
f1.close()
f1 = open("input.txt", mode="rt")
for x in f1:
    line = x.split()
    intermediate_code_function(line)



# print(lc)
# print(literal_table)
# final_literal_table = pd.concat([final_literal_table,literal_table])


# print(final_literal_table)
# print(pool_table)


f2 = open("symbol_table.txt", mode="wt")
dfasString = symbol_table.to_string(index = False)
f2.write(dfasString)

f3 = open("literal_table.txt", mode="wt")
dfasString = final_literal_table.to_string(index = False)
f3.write(dfasString)
```

```
f4 = open("pool_table.txt", mode="wt")
pool_table= pool_table.drop_duplicates()
dfasString = pool_table.to_string(index = False)
f4.write(dfasString)

f5 = open("intermediate_code.txt", mode="wt")
intermediate_code = intermediate_code.drop_duplicates()
dfasString = intermediate_code.to_string(header=False, index = False)
f5.write(dfasString)
```

Outputs: -

Symbol Table

Symbol	Address
LOOP	502
N1	506
NEXT	502
X	515

Literal Table

Number	Literal	Address
1	'=6'	508
2	'=2'	509
3	'=4'	510
4	'=3'	511
5	'=3'	516

Pool Table

Liternal Number
1
5

Intermediate Code:

```
(AD , 01)    (C,500)
(IS , 04) BREG (L,1)
(IS , 01)    BREG  X
(IS , 04) AREG (L,2)
(AD , 03)      (C,4)
(IS , 01) AREG (L,3)
(IS , 02) BREG (L,4)
(AD , 05)
(AD , 04)
(IS , 03) AREG (L,4)
(IS , 00)
(DL , 01)      (C,1)
(AD , 02)
```