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 Literal Table Pool Table

A screenshot of a computer screen

Description automatically generated with low confidence Graphical user interface

Description automatically generated Graphical user interface, text

Description automatically generated

Intermediate Code:

A picture containing text, outdoor

Description automatically generated