Name: Dhroov Makwana
Gr no.: 22010538
Roll no.: 321042
Batch: A2

Assignment 1

Aim: Generate Symbol table, Literal table, Pool table, Intermediate code of a two-pass Assembler and display errors for the given source code.

Theory:

Assembler is a software that converts an assembly language code to machine code. It takes basic computer commands and converts them into binary code that computer's processor can use to perform its basic operations. These instructions are assembler language or assembly language. It uses opcode for the instructions. An opcode basically gives information about the instruction. The symbolic representation of the opcode (machine level instruction) is called mnemonics. Programmers use them to remember the operations in assembly language.

Types of Assemblers:

On the basis of phases used to convert to machine code, it has two types

1. Single pass assembler: These assemblers perform the whole conversion of assembly code to machine code in one go.
2. Multi-pass/Two pass assembler: These assemblers first process the assembly code and store values in the opcode table and symbol table**.** And then in the second step, they generate the machine code using these tables.

a) Pass 1

- Symbol table and opcode tables are defined.
- Record of the location counter is kept.
- Processes the pseudo instructions.

b) Pass 2

- Converts the opcode into the corresponding numeric opcode.

- Generates machine code according to values of literals and symbols.

Data structures used:

- Location counter (LOCCTR): It maintains the track of machine addresses of symbolic tables.
- Machine opcode table (MOT): It is used to accept the instructions and convert/gives its binary opcode. In pass 1, using mnemonic Opcode, MOT is consulted to update location Counter (LC).

  In pass 2, using mnemonic opcode, MOT is consulted to obtain

  1) Binary opcode (to generate the instruction).

  2) Instruction length (to update the instruction).

  3) Instruction Format (to assemble the instruction).

- Symbol Table (SYMTAB): - Each entry in symbol table has two primary fields
  name and address. The symbol table uses concept of forward reference to achieve address of symbols.
- Literal Table (LITTAB): - Literal table contains all literals and address of all
  literals. For literals LTORG directives places all constants at consecutive memory locations. If we are not using LTORG then all literals are placed after END
  in memory
- Pool Table (POOLTAB): - Record of different literal pools is maintained using the auxiliary table POOLTAB. At any Stage, the current literal pool is the
  last pool in LITTAB.

Algorithm:

Pass-1:

• Create Mnemonic Opcode table with mnemonic opcode, class, opcode and length. Make entries in
it statically.

• Create data structures for Symbol table with symbol, address and length, Literal table with literal
and address and Pool table with literal_no.
• Create and open a source file in read mode having assembly language program. (Sample1.asm)

Pass-2:

• Open a target file in write mode
• Initialize variables like location counter=0 and so on
• Scan/read source file word by word and proceed as per the algorithm i.e compare tokens with
Opcode table.
• While doing the above step make entries in symbol table, literal table, pool table and intermediate
code in target file
• Show errors at the end
• Close all files

Code:

```
# Reading file and storing contents
f = open("sample.asm", 'r')
file = (f.read()).split("\n")
f.close()
SC = []
for i in file:
    a = i.split("\t")
    SC.append(a)
del file

# Displaying File Content
f = open("sample.asm", 'r')
print("Source Code : \n")
print(f.read())

# Creating MOT table
MOT = {
    "STOP": ("IS", "00"),
    "ADD": ("IS", "01"),
    "SUB": ("IS", "02"),
    "MULT": ("IS", "03"),
    "MOVER": ("IS", "04"),
    "MOVEM": ("IS", "05"),
    "COMP": ("IS", "06"),
    "BC": ("IS", "07"),
    "DIV": ("IS", "08"),
    "READ": ("IS", "09"),
    "PRINT": ("IS", "10"),
    "LOAD": ("IS", "11"),
```

```
    "START": ("AD", "01"),
    "END": ("AD", "02"),
    "ORIGIN": ("AD", "03"),
    "EQU": ("AD", "04"),
    "LTORG": ("AD", "05"),
    "DS": ("DL", "01"),
    "DC": ("DL", "02"),
    "AREG": ("RG", "01"),
    "BREG": ("RG", "02"),
    "CREG": ("RG", "03"),
    "DREG": ("RG", "04"),
    "EQ": ("CC", "01"),
    "LT": ("CC", "02"),
    "GT": ("CC", "03"),
    "LE": ("CC", "04"),
    "GE": ("CC", "05"),
    "ANY": ("CC", "06")
}

# Display table
print()
print("Displaying MOT Table")
for i in MOT:
    print(i, "\t", MOT[i])

ST = {}
LT = []
PT = {}
er = []
LC = 0
flag = 1

print("----------------------")
print("Intermediate Code : ")
print("----------------------\n")
for i in SC:
    label = i[0]
    opcode = i[1]
    operand = i[2]
    if opcode != 'EQU' and opcode != 'END' and opcode != 'LTORG'
and opcode != 'ORIGIN':
        print("LC=", LC, end='\t\t')
    else:
        print('', end="\t\t\t")
    if label.isalpha():
        if label not in ST.keys():
            ST[label] = LC
        elif ST[label] == '':
            ST[label] = LC
        else:
            temp = label + " is already declared"
            er.append(temp)
    if opcode == 'START':
        LC = int(operand) - 1
    if opcode != '':
```

```
        if opcode in MOT.keys():
            print(MOT[opcode], end='\t')
        else:
            print("**", end="\t")
            temp = opcode + ":Invalid mnemonic"
            er.append(temp)
    else:
        print("\t", end='\t')
    if opcode == 'ORIGIN':
        if operand.isnumeric():
            LC = int(operand)
        else:
            LC = int(ST[operand[0]]) + int(operand[2:])
    if opcode == 'EQU':
        ST[label] = ST[operand]
    if operand.isnumeric():
        print("(C,", operand, ")", end="\t")
    if operand.isalpha():
        if operand not in ST.keys():
            ST[operand] = ''
        a = list(ST.keys()).index(operand) + 1
        print("(S,", a, ")", end="\t")
    if operand != '' and operand[0] == '=':
        if label == '' and opcode == '':
            for j in LT:
                if j[0] == operand and len(j) == 1:
                    j.append(LC)
                    a = LT.index(j) + 1
            if flag == 1:
                PT[a] = ''
        else:
            LT.append(list(operand.split(" ")))
            a = len(LT)
        print("(L,", a, ")")
    if opcode == '':
        flag = 0
    else:
        flag = 1
    if opcode != 'EQU' and opcode != 'END' and opcode != 'LTORG'
and opcode != 'ORIGIN':
        LC += 1
    print("\n")
temp = list(PT.keys())
for j in range(0, (len(temp) - 1)):
    PT[temp[j]] = temp[j + 1] - temp[j]
    PT[temp[-1]] = len(LT) - temp[-1] + 1

# Printing Output
print("\n\n---------------------")
print("Symbol Table :")
print("---------------------\n")
print("Symbol\tAddress")
for i in ST:
    print(i, "\t", ST[i])
    if ST[i] == '':
```

```
        temp = i + " is not declared"
        er.append(temp)
print("\n\n----------------------")
print("Literal Table :")
print("----------------------\n")
print("Literal\tAddress")
for i in LT:
    print(i[0], "\t", i[1])
print("\n\n----------------------")
print("Pool Table : ")
print("----------------------\n")
print("#P\t#L")
for i in PT:
    print(i, "\t", PT[i])
print("\n\nErrors : ")
for i in er:
    print(i)
```

Input (sample.asm) (Tab separated) :-

```
    START  100
A   DC 01 ;sample program
    LOAD   A
    LOAD   C  ;c variable
    ADD    ='5'
    AD D
    ORIGIN A+2
    MULT   ='10'
    ADD    L
    LTORG
       ='5'
       ='10'
L   ADD    ='5'
    ADD    B
B   DS 1
C   EQU    B
A   DS 1
    END
       ='5'
```

Output:-

```
Source Code :

    START   100
A   DC  01  ;sample program
    LOAD    A
    LOAD    C   ;c variable
    ADD ='5'
    AD  D
    ORIGIN  A+2
    MULT    ='10'
    ADD L
    LTORG
        ='5'
        ='10'
L   ADD ='5'
    ADD B
B   DS  1
C   EQU B
A   DS  1
    END
        ='5'
```

```
--------------------
Symbol Table :
--------------------

Symbol  Address
A    100
C    108
D
L    106
B    108
```

```
---------------------
Literal Table :
---------------------

Literal Address
='5'      104
='10'     105
='5'      110
```

```
--------------------
Intermediate Code :
--------------------

LC= 0       ('AD', '01')    (C, 100 )

LC= 100     ('DL', '02')    (C, 01 )

LC= 101     ('IS', '11')    (S, 1 )

LC= 102     ('IS', '11')    (S, 2 )

LC= 103     ('IS', '01')    (L, 1 )


LC= 104     **  (S, 3 )

            ('AD', '03')

LC= 102     ('IS', '03')    (L, 2 )
LC= 103     ('IS', '01')    (S, 4 )

            ('AD', '05')

LC= 104             (L, 1 )


LC= 105             (L, 2 )


LC= 106     ('IS', '01')    (L, 3 )


LC= 107     ('IS', '01')    (S, 5 )

LC= 108     ('DL', '01')    (C, 1 )

            ('AD', '04')    (S, 5 )

LC= 109     ('DL', '01')    (C, 1 )
              ('AD', '02')

  LC= 110             (L, 3 )
```

```
---------------------
Pool Table :
---------------------

#P  #L
1    2
3    1
```

```
Errors :
AD:Invalid mnemonic
A is already declared
D is not declared

Process finished with exit code 0
```