

EXPERIMENT 7

Aim:

To implement two pass assembler

Theory:

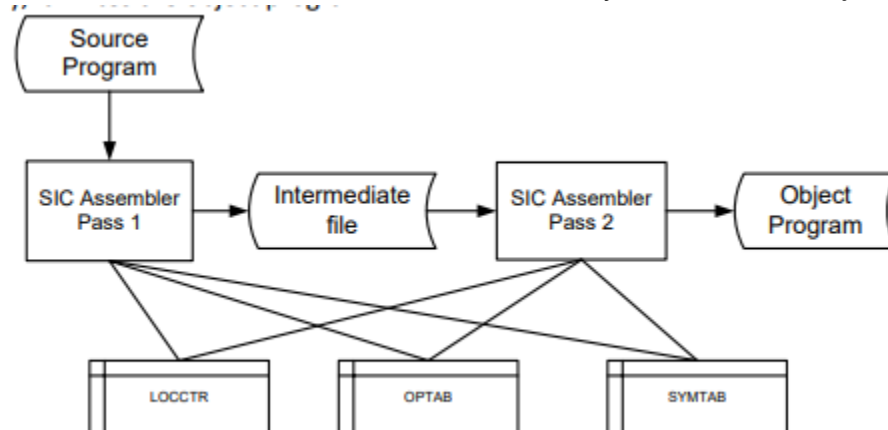
Assembler:

A program that accepts an assembly language program as input and produces its machine language equivalent along with information for the loader.

Details – The assembler performs the following functions. It converts mnemonic operation codes and symbolic operands to their machine language equivalents. It builds the machine instructions and converts the data constants to internal machine representations. Finally, it writes the object program and the assembly listing. The details of two-pass of SIC assembler is as follows.

Pass 1 defines symbols by assigning addresses. It assigns addresses to all statements in the program. It then save the values assigned to all labels. It then processes some assembler directives.

Pass 2 assemble instructions and generate object program. Pass 2 assemble instructions and then generate data values defined by BYTE and WORD etc. It then process remaining assembler directives not done in Pass 1. Finally, it writes the object program



Problem Definition & Assumptions – The two pass SIC assembler performs two functions namely i) define symbols and ii) assembler instructions and generate object program.

- 1) The two pass assembler uses three data structures: i) Operation Code Table (OPTAB), ii) Symbol Table (SYMTAB) and iii) Location Counter (LOCCTR).
- 2) OPTAB stores mnemonic machine code and its machine language equivalent. It also include instruction format, length etc. Pass 1 loop up and validate operation codes in source using OPTAB. Pass 2 translate operation codes to machine language code using OPTAB.
- 3) SYMTAB stores label name, its value as addresses, flag to show label type and length of storage. Pass 1 uses SYMTAB to enter labels and location counters (LOCCTR). Pass 2 obtains the addresses of operands from SYMTAB.
- 4) LOCCTR is a variable to store assignment of addresses. LOCCTR counts the addresses in bytes.

Code:

```
from ast import literal_eval as le

def op_opr(a):
    return '0'*(6-len(a))+a

def op_syt(a):
    return '0'*(4-len(a))+a

def op_name(a):
    res = a+' '*max(0,6-len(a))
    return res[:6]

ins = {
    'ADD': '18',
    'AND': '40',
    'COMP': '28',
    'DIV': '24',
    'J': '3C',
    'JEQ': '30',
    'JGT': '34',
    'JLT': '38',
    'JSUB': '48',
    'LDA': '00',
    'LDCH': '50',
    'LDL': '08',
    'LDX': '04',

    'STA': '0C',
```

```

}

sudo_op = ['START', 'END', 'RESW', 'WORD']

inp = []
f = open("inp.txt", "r")
for x in f:
    inp.append(x.strip().split())
print(inp)
op = []

name = ''
loc = '0x'
ind = -1

for j,i in enumerate(inp):
    if 'START' in i:
        name = i[0]
        ind = j
        if i[-1] == 'START':
            loc += '0'
        else:
            loc = '0x'+i[-1]
        break

symbol_table = dict()
error = False

ogloc = loc
ogind = ind
symbol_table[inp[ind][0]] = op_syt(loc[2:])
_name = inp[ind][0]

# FIRST PASS

dec_loc = le(loc)
while not error:
    cur_op = ''
    ind+=1
    ind2 = 0
    if inp[ind][ind2] not in ins and inp[ind][ind2] not in sudo_op:
        if inp[ind][ind2] in symbol_table:
            print('ERROR!!!')
            error = True
        else:

```

```

        symbol_table[inp[ind][ind2]] = op_syt(hex(dec_loc)[2:])
        ind2+=1
    instruction = inp[ind][ind2]
    if instruction not in sudo_op:
        cur_op = ins[instruction]
        ind2+=1
        operand = inp[ind][ind2]
        if operand in symbol_table:
            cur_op += symbol_table[operand]
        else:
            cur_op += '????'

    elif instruction == 'END':
        break
    else:
        if instruction == 'RESW':
            wrds = le('0x'+inp[ind][ind2+1])
            cur_op = '*****'
            for i in range(wrds-1):
                op.append(cur_op)

        if instruction == 'WORD':
            cur_op = op_opr(inp[ind][ind2+1])
        if instruction == 'END':
            break

    op.append(cur_op)
    dec_loc += 3

with open('op1.txt', 'w') as f:
    l1 = ['H', op_name(_name), op_opr(ogloc[2:]), op_opr(hex(dec_loc-
le(ogloc))[2:])]
    l1 = '|'.join(l1)
    f.write(l1+'\n')

bytes = len(op)
for i in range((bytes)//20):
    l2 = ['T']
    l2.append(op_opr(hex(le(ogloc)+20*i)[2:]))
    l2.append('1E')
    for j in range(20):
        l2.append(op[20*i+j])
    l2 = '|'.join(l2)
    f.write(l2+'\n')

```

```

if bytes-20*(bytes//20)>0:
    l2 = ['T']
    l2.append(op_opr(hex(le(ogloc)+20*(bytes//20))[2:]))
    lenobj = hex(bytes-20*(bytes//20))[2:]
    if len(lenobj)==1:
        lenobj = '0'+lenobj
    l2.append(lenobj)
    for j in range(bytes-20*(bytes//20)):
        l2.append(op[20*(bytes//20)+j])
    l2 = '|'.join(l2)
    f.write(l2+'\n')

l3 = ['E',op_opr(ogloc[2:])]
l3 = '|'.join(l3)
f.write(l3+'\n')

loc = ogloc
ind = ogind
opind = 0
op2 = []

# SECOND PASS
dec_loc = le(loc)
while not error:
    cur_op = ''
    ind+=1
    ind2 = 0
    if inp[ind][ind2] not in ins:
        if inp[ind][ind2] not in symbol_table:
            if inp[ind][ind2] == 'END':
                break
            print('BUGGG!!!',inp[ind][ind2])
            error = True
        ind2+=1
    instruction = inp[ind][ind2]
    if instruction == 'END':
        break
    if instruction not in sudo_op:
        cur_op = ins[instruction]
        ind2+=1
        operand = inp[ind][ind2]
        if operand in symbol_table:
            cur_op += symbol_table[operand]

```

```

        op[opind]=cur_op
    else:
        print('ERROR')
        error = True

    opind += 1
    dec_loc += 3

print("\n\nSYMBOL TABLE")
for i in symbol_table:
    print(i,'\t-> ',symbol_table[i])

with open('op.txt', 'w') as f:
    l1 = ['H', op_name(_name), op_opr(ogloc[2:]), op_opr(hex(dec_loc-
le(ogloc))[2:])]
    l1 = '|'.join(l1)
    f.write(l1+'\n')

    bytes = len(op)
    for i in range((bytes)//20):
        l2 = ['T']
        l2.append(op_opr(hex(le(ogloc)+20*i)[2:]))
        l2.append('1E')
        for j in range(20):
            l2.append(op[20*i+j])
        l2 = '|'.join(l2)
        f.write(l2+'\n')

    if bytes-20*(bytes//20)>0:
        l2 = ['T']
        l2.append(op_opr(hex(le(ogloc)+20*(bytes//20))[2:]))
        lenobj = hex(bytes-20*(bytes//20))[2:]
        if len(lenobj)==1:
            lenobj = '0'+lenobj
        l2.append(lenobj)
        for j in range(bytes-20*(bytes//20)):
            l2.append(op[20*(bytes//20)+j])
        l2 = '|'.join(l2)
        f.write(l2+'\n')

    l3 = ['E',op_opr(ogloc[2:])]
    l3 = '|'.join(l3)
    f.write(l3+'\n')

```

Result:**Input:**

```

≡ inp.txt
1  TEST START 2003
2  FIRST LDA FIVE
3  STA ALPHA
4  ALPHA RESW 1
5  FIVE WORD 5
6  END FIRST

```

Output:

```

PS D:\SPIT\SEM 6\CC Lab> py exp7.py
[['TEST', 'START', '2003'], ['FIRST', 'LDA', 'FIVE'], ['STA', 'ALPHA'], ['ALPHA', 'RESW', '1'], ['FIVE', 'WORD', '5'], ['END', 'FIRST']]

```

```

SYMBOL TABLE
TEST -> 2003
FIRST -> 2003
ALPHA -> 2009
FIVE -> 200c

```

```

≡ op1.txt
1  H|TEST |002003|00000c
2  T|002003|04|00????|0C????|*****|000005
3  E|002003
4

```

```

≡ op.txt
1  H|TEST |002003|00000c
2  T|002003|04|00200c|0C2009|*****|000005
3  E|002003
4

```

Conclusion:

In this Experiment, I learned about the concept of 2 pass assembler and was able to implement it using Python covering some of the edge cases.