Assignment No. 1: Design suitable Data structures and implement Pass-I and Pass-II of a two-pass assembler for pseudo-machine. Implementation should consist of a few instructions from each category and few assembler directives. The output of Pass-I (intermediate code file and symbol table) should be input for Pass-II.

Theory:

Function of the assembler.

An assembler accepts as input an assembly language program and produces as output its machine language equivalent.

Two Pass assembly: -

1. It can handle forward references easily.
2. Location counter (LC) processing is done in pass 1 and symbols defined in the program are entered in the symbol table.
3. The second pass uses this address information to generate target program for the loader.
4. In this scheme pass-1 constructs an intermediate representation of source program used by pass-2. This is called intermediate code (IC).

Single pass assembly: -

1. Here problem of forward reference is resolved using the technique of back-patching where the operand that is forward referenced is left blank initially and then is later filled.
2. These blank field are kept in a special table called as table of incomplete information. (TII)
3. Each entry in TII is of the form: (<Instruction address> <symbol>)
4. e.g. we have forward reference in following statement:

```
- - - -
MOVER, B, ONE
- - - - - .
- - - - - .
ONE DC 1
- - - - - .
```

This statement can be partially synthesized since ONE is a forward reference.
Hence entry in TII will be of the form (1001, ONE)

5. By the time we process END statement. TII will contain information about all forward references.
6. Assembler can process this entry by checking out the symbol table and finding the address of each forward reference.

Tasks Performed by the 2 PASS Assembler:

Pass I:

1. Separate symbol table mnemonic and operand fields
2. Build the symbol table.
3. Perform IC processing.
4. Construct intermediate code.

Pass II:

1. Synthesis of target program.
2. Evaluate fields and generate code.
3. Process pseudo –opcodes.

Input :

A sample program in assembly language

A machine operation table (MOT) specifying the Mnemonics op- code, machine opcode, length of instruction op- code, instruction length, instruction format etc.

A pseudo operation table (POT) specifying the Pseudo-ops.

Output (Expected):

The above assembly program should be stored in a text file which would be the source file for the assembler.

The output of the assembler is expected as follows:

After Pass I: construction of symbol table and literal table, pool table

(Draw tabular structure of symbol table and literal table, pool table**)**

Algorithm: For PASS I of TWO PASS ASSEMBLER

1. Start
2. Set initial values
LC=0, PoolTabptr=0, LTABptr = 1,
POOLTAB[1] =1
Symbol_No=1
3. /* Scan and process each statement till end statement encounters*/
a. If label is present then
i) This_label=symbol in label field
ii) Enter (this_label, LC) into SYMBOL TABLE.

b. If START statement then i.e. if opcode='START' then
i) Generate intermediate code(AD,01)
ii) If argument is provided to START statement then Reset value of LC = argument given on START statement. Change address of label (if any) on START start statement to value of LC obtained in above step (This address is reflected in symbol table entry for this label.)

c. If ORIGIN statement i.e. if OPCODE = 'ORIGIN' then
i) Calculate address specified specified in operand field of ORIGIN statement
ii) Reset LC=address calculated in above step.
(No intermediate code is generated for ORIGIN statement)

d. If EQU statement i.e. if OPCODE='EQU' then
i) Calculate address specified in operand field of 'EQU' statement.
ii) Correct symbol table entry for this label by changing its address to the address calculated in above step
(No intermediate code is generated for EQU statement)

e. /* Process declarative statement
i) If OPCODE='DC' then
(1) Generate code (AD,05)
(2) Size =size of constant
(3) LC=LC+size
ii) If OPCODE='DS' then

(1) Generate code(AD,06)
(2) Size=size specified as argument of DS statement also
reflect this size in symbol table.
(3) LC=LC+size

f. /*Process LTORG statement
If LTORG statement i.e. if OPCODE='LTORG' then
(1) Process LITTAB[POOLTAB[PTP]…] to LITTAB[LTP-1] So that memory location can be allocated to these literals and put addresses in address field of literal Table. Also update LC accordingly.

(2) PTP=PTP+1
(3) POOLTAB[PTP]=LTP

g. /*Process Imperative statement*/
If IMPERATIVE statement then
(1) Code1:=Machine code for given OPCODE in MOT
(2) LC:=LC+size of instruction (obtained from MOT)
(3) Code:=(IS,code1)

4. /* Process operands which can be registers, symbol i.e.(storage variables or constant) or integer constant.
If OPERAND is a register then Register operand can be either of AREG, BREG, CREG, DREG therefore corresponding code generated is 1,2,3,4 resp.
If OPERAND is symbol then Entry-no=symbol table entry number of this symbol Generated code as (S,Entry-No)
If OPERAND is a literal This-literal:= literal in operand field Generate code as (L, this-literal no in literal table) LTP:=LTP+1
If OPERAND is integer constant then This-constant=constant value Generate code(C, this-constant)

5. /*Process END statement*/
a. If END statement i.e. if OPCODE='END' statement then Perform 3.f
b. Generate (AD,02)
c. GOTO Pass II

Algorithm: For PASS II of TWO PASS ASSEMBLER

1. Start
2. Initialize various pointers
PTP=1
LC:=0
3. /* Read intermediate code i.e. set of intermediate instructions and depending upon intermediate opcode and intermediate operand code process it as follow till intermediate code for END statement encounters*/
(a) Clear machine-code-buffer
(b) /* START/ORIGIN statement*/
if START/ORIGIN statement i.e. if intermediate code for START/ORIGIN statement is encountered then
    a. LC:= value specified in it's operand field
    b. Size:=0 (as no machine code is generated for START /ORIGIN construction)
(c) /*Process DC statement*/
if DC statement i.e. if intermediate code for DC statement is encountered then
i. Machine opcode field will have blank or zero
ii. First operand field will have also blank or zero
iii. Second operand field will contain value of constant being defined (This is how constant being declared is stored at memory location pointed by current value of LC)
iv. This code is assembled in machine-code buffer
v. Size:=size of DC statement (which can be obtained from POT
vi. LC=LC+size

(d) /*Process DS statement*/
if DS statement i.e. if intermediate code for DS statement is encountered then
    a. size:= argument value specified in DS statement
    b. memory locations of 'size' number of times are reserved
    c. and all three fields of those memory locations are left blank

d. LC:=LC+size

i. /*Process LTORG statement*/

If LTORG statement is encountered i.e. If intermediate code for LTORG is encountered then

(i) No-Of-literals-in –this- POOL=0 Process literals LITTAB[POOLTAB[PTP] to LITTAB[PTP}

(ii) Process literals in current pool in the same as processing of constants in DC statements

(A) process each literal in current pool, so that literals will be defined at the memory locations pointed by LC.

(1) M/C opcode field will have blank/zero.

(2) OPERAND1 field will have blank/zero.

(3) OPERAND1 field will have literal-value.

(4) LC:=LC+1

(5) No-of-Literals-in this Pool=No-of literals-in this Pool+1

(Repeat steps 1-5 for each literal)

(B) Size:=no-of literals-in this-Pool.

(C) PTP:=PTP+1

ii. /*Process imperative statements*/

if IMPERATIVE statement is encountered i.e. Intermediate code for imperative statement is encountered then

(i) Machine code field will have only equivalent machine code for imperative opcode. For example, if intermediate opcode is (IS, 01) then discard IS and store only 01 in machine opcode field.

(ii) OPERAND1 will have machine constant representing machine register.

(iii) By referring second operand field of intermediate instruction Pass-II can decide whether second operand is symbol or LITERAL.

(iv) Depending upon type of second operand obtain its address by referring to SYMBOL TABLE or LITERAL TABLE.

(v) OPERAND2 field of machine code will contain address obtained in above step.

(vi) Size:=size of instruction

iii. IF size#0 then

i. Move contents of machine code buffer to address=code area addr + LC

ii. LC: = LC + size.

4. /*Process END statement*/

If END statement is encountered i.e. intermediate code for END statement is encountered then

(i) Perform STEP (2.f) and (2.b). Because processing of LTORG and END statement is same as both of them cause the allocation of addresses to the literals in current-pool. (i.e. literal pool ending at LTORG or END statement)

(ii) AS it is end of program contents of code area are moved into OUTPUT file.

Conclusion: Hence we have successfully designed suitable data structures and implemented Pass I and Pass II of a two pass assembler for pseudo-machine and generated intermediate code file and symbol table for Pass I and machine code for Pass II.