Date: 8 Sept 2021.

Assignment number : A-01

1) Title : Design suitable data structure and implement Pass-I assembler for pseudo-machine. Implementation should consist a few instruction from each category and assembler dir

2) Software / Hardware requirement :

* Software requirements:
  1) Java Development kit.
  2) Integrated Development Enironment OR.
  3) Notepad ++.

* Hardware Requirements.
  1) Computer System
     Processor : i5 9th Gen
     Ram : 8 GB
  2) I/O Peripherals like keyboard & Mouse.
  3) Monitor : 720p / 1080p FHD/IPS.

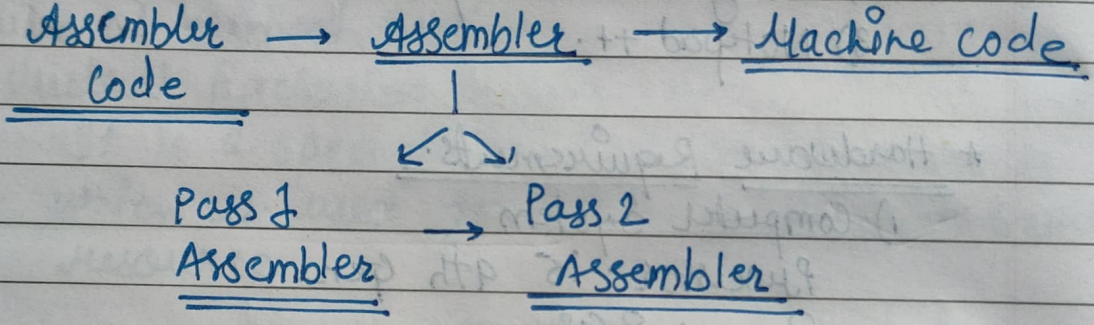
3) Learning Objective :
  1) To understand the working of pass-I assembler.
  2) To use appropriate data structure to solve given problem.
  3) To apply programming knowledge and skills to find optimum solution for given problem.

4) Learning Outcome :
  1) Understood the working of pass-I assembler.
  2) Used appropriate data structure to solve the given problem.

3) Applied programming background and skills to solve given problem.

5] Concept related Theory:

Assembler is a program for converting instructions written in low-Level assembly code into relocateable machine code and generating along information for the loader.



It generates instructions by evaluating the mnemonics in operations field and find the value of symbol and literals to produce machine code. Now, if assembler do this in one scan then it is called as single pass assembler, otherwise if it does in multiple scans then called multiple pass assembler.

- Pass 1 assembler:
  1) Defines Symbols and literals and remember them in symbol table and literal table respectively
  2) Keep track of location counter.
  3) Process pseudo-operations.

- Pass 2 Assembler.
  1) Generates Object Code by converting symbolic op-code into respective numeric op-code.
  2) Generate data for literals and look for values of symbols.

Opcode

-called mnemonics operation code. They're used to specify operation.
ex. add, sub, mul, etc.

Assembly Statements :

An assembly program consists of three kinds of statement

1) Imperative statements : Specifies an operation to be performed.
2) Declarative :
   DS is declared storage reserves area of memory and associates name with them.
   DC is declare constant - constructs memory word containing constants.
3) Assembler directives : These are the instruction to the assembler and not to the machine. These are sometime called pseudo code operations.
   eg 1) START
      2) END
      3) ORIGIN.

Forward reference:

The reference to an entity that precedes its definition in the program is called forward reference. An example is:

```
:
:
:
CALL JUMP
:
:
JUMP :---
:
:
```

Language processor pass

It is the processing of every statement in a source program or its equivalent representation to perform a language processing function. This is also used during a set of langige processing functions.

Literals

A literal is an operand with the syntax ='<value>'. It differs from a constant because its location cannot be specified in the assembly language program. This helps to ensure that its value is not changed during the execution of a program

E.g

1) ADD AREG, ='5'
2) FIVE DC '=5'

6) Algorithm.

1) Start.
2) loc_cntr = 0 (Default Value) (location counter)
pooltab_ptr = 1 ; POOLTAB[1] = 1; (points to entry of LITTAB)
littab_ptr = 1; (Points to an entry in POOLTAB)
3) While next readed statement is not END statement.
   a) If a label is present then
      i. this_label = Symbol in label field.
      ii. Enter (this_label, loc_cntr) in SYMTAB
   b) If an LTORG statement then
      Allocate memory for literals and increment pooltab_ptr.
      i. Process Literals LITTAB to allocate memory and put the address field. update loc_cntr accordingly
      ii. pooltab_ptr = pooltab_ptr + 1;
      iii. POOLTAB[pooltab_ptr] = littab_ptr.
   c) If a start or ORIGIN statement the
      Memory allocation process.
      i. loc_cntr = value specified in operand field;
   d) If an EQU statement then
      update the symbol table entry for Label.
      i. this_address = value specified in <address spec>;
      ii. Correct the symtab entry for this_label to (this_label, this_address);
   e) If a declaration statement then
      i. code = code of declaration statement.

ii. Size = size of memory area required by DC/DS
iii. loc_cntr = loc_cntr + size;
iv. Generate IC(DL, code).

f) If an imperative statement then
i. Code = Machine opcode from OPTAB.
ii. loc_cntr = loc_cntr + instructions length from OPTAB;
iii. If operand is a literal then.
this_literal = literal in operand field;
~~if~~ LITTAB [littab_ptr] = this_literal;
littab_ptr = littab_ptr + 1;
else
this_entry = SYMTAB entry ~~number~~ number of operand generate IC'(IS, code)(S, this_entry);

4) Processing END Statement.
a) Perform step 3(b) to allocate memory for literals.
b) Generate IC'(AD, 02) IC unit for END.

5) End.

7] Conclusion
Understood working of pass I assembler and implemented it using programming knowledge.

8] References:

1) GeeksforGeeks.
2) youtube.com / pass 1 assembler.
3) wbuthelp.com/chapter_file/2677.pdf.
4) slidetodoc.com/unit-4-unit-3-pushdown-assembler-automata-prof/

— x —