Date : 15 Sep 2021

Assignment Number = A-01 (B)

1) Problem Statement: Design suitable datastructure and implement pass-II assembler for psudo-code (machine code). Implementation shall consist a few instruction from each cateogory and assembly directive

2) Software / Hardware requirement:

* Software requirements
1) Java Devlopment kit
2) Integrated Development Environment OR
3) Notepad ++ / VsCode.

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

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

4) Learning outcome:

1) Understood the working of pass-II assembler.
2) Used appropriate data structure to sowe given problem.
3) Applied programming background and skills to sowe given problem.

5] Concept related Theory:

X Pass-II Assembler:

In pass-2 assembler, instruction are again read line by line, but from intermediate code which was output from pass I assembly process, looking only for Label definitions. All the Labels are collected, assigned address, and placed in the symbol table in this pass, no instructions

Pass-II Assembler:

In pass-2 assembler, instruction are again read line by line, but now this time from intermediate code which was output of pass-I assembler., and are assembled using the symbol table, Basicaly, the assembler goes through the program one line at a time, and generates machine code for that instruction. Then the assember proceeds to the next instruction. In this way the entire machine code program is created. For most instructions this process works fine, for example for instructions that only references registers the assembler can compute the machine code easily, since the assembler knows where the registers are.

Difference between pass 1 and pass 2 assembler :

A one pass assembler passes over the source file exactly once, in the same pass collecting tables, resolving future references and doing the actual assembly. The difficult part is to resolve future label references (the problem of forward referencing) and assemble code in one pass. The one pass assembler prepares an intermediate file, which is used as input by the two pass assembler.

A two pass assembler does two pass over the source file (the second pass can be over an intermediate file generated in the pass of the assembler). In the first pass all it does it looks for label definitions and introduces them in the symbol table (a dynamic table which includes the label name and address for each label in the source program). In the second pass, after the symbol table is complete, it does the actual assembly by translating the operations into machine codes and so on.

How does pass-II assembler works;

So, first of all pass 2 assembler requires intermediate code, literal table, symbol table pool table which are output of pass 1 assembly process.

So it starts by readiy whole intermediate code line by line. so if it encounters start assembler diretive, declarative statement such as

Define storage ; DS, Define Constant ; DS. pass 2 assembler simply ignore these cases.

If the word is IS, or other AD so the address associated with respective statement is inserted in machine code and further if there is register like AREG, BREG, CREG then their respective address 1, 2, or 3 is inserted next to to the previous word in same line of machine code. Again next if its literal/symbol then the values next to it is referenced as index for literal/symbol table respectively and the address at that index/position in inserted in machine code. These things are repeated for all statements in intermediate code.

Pass 2 Algorithm

```
1) Start
2) Code.area.address := address of code_area
   pooltab_ptr := 1
   loc_cntr = 0
3) While next statement is not an END statement
   (a) Clear machine_code_buffer.
   (b) If LTORG statement
       Assemble the literals in the machine_code buffer
       size := size of memory area required for the
       literals pooltab ptr := pooltab_ptr + 1
   (c) If a START or ORIGIN statement then
       loc_cntr := value specified in the operand field
```

```
size:= 0.
 (d) If a declare statement
   If a DC statement then
     assemble the constraint in the machine_code_buffer.
     size:= size of memory area required by DC/DS
 (e) If an imperative statement.
     Get the operand address from symmtab or
     littab assemble instruction in machine_code_buffer
       size := size of the instruction
 (f) If size not equal to
      move contents of machine_code_buffer to the
      address_code_area_address + loc_cntr
      loc_cntr := loc_cntr + size.

4) Processing of END statement
     perform step (3(b)) and (3(f))
     write code_area int output file.

5) End.
```

6] Conclusion

Understood working of pass 2 assembler and implemented it using programming knowledge

8] References

1) Geeks for Geeks.
2) Youtube.