Minor Project- Report

Aug-2019-2020

Course Faculty: Mrs. Rashmi S.R                    Course Name & code: SS Lab

Semester: 6th Date: 2nd April 2020

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TITLE OF THE PROJECT | Single Pass Assembler | | | |
|  |  | | | |
| STUDENT NAME | Monu Singh | Mrinal Singh Walia | Mrinalini Raghavendran | Nehal Sharan Chandu |
| USN | 1DS17CS067 | 1DS17CS068 | 1DS17CS069 | 1DS17CS070 |
| INDIVIDUAL  CONTRIBUTION |  |  |  |  |
| GUIDE | Dr. Vindhya P Malagi | | | |
|  |  | | | |
| PROJECT ABSTRACT : | Assemblers a program that turns symbols into machine instructions. ISA-specific:close correspondence between symbols and instruction set mnemonics for opcodes, labels for memory locations, additional operations for allocating storage and initializing data.  The ability to compile in a single pass is often seen as a benefit because it simplifies the job of writing a compiler and one pass compilers generally compile faster than multi-pass compilers. Many languages were designed so that they could be compiled in a single pass (e.g., Pascal).  In some cases the design of a language feature may require a compiler to perform more than one pass over the source. For instance, consider a declaration appearing on line 20 of the source which affects the translation of a statement appearing on line 10. In this case, the first pass needs to gather information about declarations appearing after statements that they affect, with the actual translation happening during a subsequent pass.  The disadvantage of compiling in a single pass is that it is not possible to perform many of the sophisticated optimizations needed to generate high quality code. It can be difficult to count exactly how many passes an optimizing compiler makes. For instance, different phases of optimization may analyse one expression many times but only analyse another expression once. | | | |
| PLATFORM USED  (H/W & S/W TOOLS TO BE USED | SOFTWARE/ PLATFORM: Linux System Environment/ TurboC | | | |
|  |  | | | |
| INTRODUCTION | Assembler is a program for converting instructions written in low-level assembly code into re locatable machine code and generating along information for the loader.  In this project we implement a single pass assembler. A single pass assembler scans the program only once and creates the equivalent binary program. The assembler substitute all of the symbolic instruction with machine code in one pass.  We provide an input file with the codes containing labels and mnemonics and operands and the output file will contain the opcodes for the assembly instructions. | | | |
|  |  | | | |
| DESIGN | one psss.png | | | |
|  |  | | | |
| PROJECT SOURCE CODE LINK (GITHUB) | [***https://github.com/abhiwalia15/One-Pass-Assembler-in-C***](https://github.com/abhiwalia15/One-Pass-Assembler-in-C) | | | |
|  |  | | | |
| CONCLUSION /FUTURE ENHANCEMENT | Through the implementation of the code we can see that the opcode and addresses are generated in only a single pass.  For future enhancements we would like to make improvements with respect to the time and space complexity. | | | |
|  |  | | | |
| UI SCREENSHOTS | input1.png    FIG 1: INPUT FILE  optab1.png  FIG 2: OPTAB FILE  outpu1.png  FIG 3: OUTPUT FILE  outputhte1.png  FIG 4: H T E FILE  symtab.png  FIG 5: SYMTAB FILE | | | |