**Assignment No.5**

|  |
| --- |
| **Title of Assignment:**  Write X86 ALP to find, a) Number of Blank spaces b) Number of lines c) Occurrence of a particular character. Accept the data from the text file. The text file has to be accessed during Program\_1 execution and write FAR PROCEDURES in Program\_2 for the rest of the processing. Use of PUBLIC and EXTERN directives is mandatory. |
| **Relevant Theory:**  **A) Procedures in Nasm:-**  The procedure, or subroutine, is an important part of any computer system's architecture. A procedure is a group of instructions that usually performs one task, it is a reusable section of the software that is stored in memory once, but used as often as necessary. This saves memory space and makes it easier to develop software. The only disadvantage of a procedure is that it takes the computer a small amount of time to link to the procedure and return from it. The CALL instruction links to the procedure and the RET instruction returns from the procedure.  The Stack stores the return address whenever a procedure is called during the execution of a programme. The CALL instruction pushes the address of the instruction following it on the Stack. The RET instruction removes an address from the Stack so the programme returns to the instruction following the CALL. With the assembler, there are some finite rules for the storage of procedures. There are two types of procedure: NEAR (intrasegment) or FAR (intersegment). B) Assembler Directives:-1) EXTERN: Importing Symbols from Other Modules EXTERN is used to declare a symbol which is not defined anywhere in the module being assembled, but is assumed to be defined in some other module and needs to be referred to by this one. Not every object-file format can support external variables: the bin format cannot. The EXTERN directive takes as many arguments as you like. Each argument is the name of a symbol:  extern variable  extern variable1, variable2  The primitive form of EXTERN differs from the user-level form only in that it can take only one argument at a time: the support for multiple arguments is implemented at the preprocessor level. You can declare the same variable as EXTERN more than once: NASM will quietly ignore the second and later re-declarations. You can't declare a variable as EXTERN as well as something else, though. 2) GLOBAL: Exporting Symbols to Other Modules GLOBAL is the other end of EXTERN: if one module declares a symbol as EXTERN and refers to it, then in order to prevent linker errors, some other module must actually *define* the symbol and declare it as GLOBAL. Some assemblers use the name PUBLIC for this purpose.  The GLOBAL directive applying to a symbol must appear *before* the definition of the symbol. GLOBAL uses the same syntax as EXTERN, except that it must refer to symbols which *are* defined in the same module as the GLOBAL directive. For example:  Global variable1  Global variable1, variable2  Like EXTERN, the primitive form of GLOBAL differs from the user-level form only in that it can take only one argument at a time.  **C) Linking and Execution of FAR procedures:-**  a) Firstly, assemble your all the programs one after another using nasm command.  b) Link the programs using following format:-  ld –o prg1.exe prg1.o prg2.o \_ \_ \_ \_ (any number of arguments)  Make exe file of the main program (here prg1) from which the far procedure is called because it is the only programs that contains starting point. It can be then followed by number of arguments that are nothing but object files of programs in which far procedures are defined.  c) Finally run the exe file of main program. (here Prg1)  **System calls for File handling:**   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | rax | rdi | rsi | rdx | Return | | Sys\_open | 2 | File name | File opening mode(flag) | File Permissions | Rax=file descriptor(successful open)/  Negative value -error | | Sys\_close | 3 | File descriptor | -- | -- | -- | | Sys\_read | 0 | File descriptor | Name (Address) of buffer | Size (in bytes) of buffer | Rax =length of actually read bytes | | Sys\_write | 1 | File Descriptor | Name (Address) of buffer | Size (in bytes) of buffer | -- |   **Design Analysis/ Implementation Logic:**  **Algorithm**   1. Accept the file name and character to search from user. 2. Append the filename with null character. 3. Open the file. 4. Compare rax with 0. If less than 0 print error and goto step 15. 5. Initialize ncnt,scnt and ccnt to 0. 6. Read a character from the file. 7. Compare RAX with 0 (to check for end of file) if equal goto step 14. 8. Compare the character with ASCII value 10(new line) if not equal goto step 10. 9. Increment newline counter goto step 6 10. Compare the character with ASCII value 20h(space) if not equal goto step 12. 11. Increment space counter goto step 6 12. Compare the character with character to search if not equal goto step 6. 13. Increment newline counter goto step 6 14. Print the counters ncnt,scnt and ccnt. 15. End |
| **Testing:**  **Test Conditions:**  **Input:**  Textfile**.**  **Output:**  Count of new line,space and character given. |
| **FAQs:**   1. Explain FAR and NEAR procedure. 2. Explain extern, global. 3. Explain System call for file opening, file closing, reading from file and writing to file. 4. Explain file opening mode. |
| **Conclusion:**  Successfully implemented the ALP for counting spaces, lines and specified character using FAR procedure. |