PROJECT NAME :SIC CODE

USER MANUAL DOCUMENTATION FOR SIC ASSEMBLER

BY ALEENA VARGHESE

ROLL NO: 25

CSE A

Documentation for SIC Assembler with GUI

- This documentation explains the code implementation of a Simple Instructional Computer (SIC) assembler program using a Graphical User Interface (GUI) on a Windows platform. The assembler performs two passes over the assembly program and generates intermediate files, symbol tables, and object code.

1.File Organization

- The program consists of several core components:
  - `passOne()`: Handles the first pass of the assembler.
  - `passTwo()`: Handles the second pass and generates the object code.
  - `displayPassOne()`: Displays the output of the first pass (intermediate file).
  - `displayPassTwo()`: Displays the object code after the second pass.
  - `displayObjectCode()`: Displays both the input from the intermediate file and the corresponding object code side by side.
- GUI components to trigger the assembler operations and display results.

2. Main Function (`WinMain`)

```
int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInst, LPSTR args, int nCmdShow)
```

- This function initializes the Windows application and the main window. It creates a window using the `CreateWindowW()` function with buttons that allow users to trigger the different stages of the assembler. The event handling of the window is performed by `WindowProcedure`.

Parameters:

- `hInst`: Handle to the current instance of the application.

- `hPrevInst`: Handle to the previous instance (unused).
- `args`: Command-line arguments (unused).
- `nCmdShow`: Controls how the window is to be shown.

3. Window Procedure

```
LRESULT CALLBACK WindowProcedure(HWND hwnd, UINT msg, WPARAM wp, LPARAM lp)
```

- This function handles all window events like button presses (`WM_COMMAND`), creation (`WM_CREATE`), and destruction (`WM_DESTROY`). Based on the `WPARAM`, it identifies which button was clicked and calls the respective function to run the assembler passes or display results.

Messages Handled:

- `WM_COMMAND`: Detects which button was pressed and triggers the appropriate assembler function.
- `WM_CREATE`: Calls `AddControls()` to add buttons to the window.
- `WM_DESTROY`: Exits the application.

4.AddControls

```
void AddControls(HWND hwnd)
```

This function creates the buttons and output box in the main window.

Buttons:

- "Run Pass 1": Runs the first pass of the assembler.
- "Run Pass 2": Runs the second pass of the assembler.
- "Display Pass 1": Displays the intermediate file generated in Pass 1.
- "passtwo address": Displays the object code after Pass 2.

- "Display Pass 2 object code": Displays both the input and object code side by side.

- **Output Box (`hOutputBox`)**: A multi-line text box where the results of the assembler operations are displayed.

5. Pass 1 Function

void passOne()

- This function performs the first pass of the assembler. It reads an assembly program from `input.txt`, generates a symbol table in `symtab.txt`, and produces an intermediate file `intermediate.txt`.

Input:

- Reads assembly instructions from `input.txt`.

Output:

- `symtab.txt`: Contains the labels with their corresponding addresses.
- `intermediate.txt`: Contains the intermediate representation of the assembly code with addresses.

Error Handling:

- ❖ If any of the required files cannot be opened, an error message box is shown.

6. Pass 2 Function

void passTwo()

- ➢ This function performs the second pass of the assembler. It reads the intermediate file and symbol table, generates the object code, and writes it to `objcode.txt`.

Input:

- ❖ Reads the intermediate file `intermediate.txt` and the symbol table `symtab.txt`.

Output:

- ❖ `objcode.txt`: Contains the final object code.

Error Handling:

- ❖ Displays an error message if the required files cannot be opened.

7. Display Functions

- 7.1 `displayPassOne`

```
void displayPassOne(HWND hwnd)
```

- Displays the contents of the `intermediate.txt` file in a formatted way in the output box (`hOutputBox`).

Format:

Address Label Opcode Operand

- 7.2 `displayPassTwo`

```
void displayPassTwo(HWND hwnd)
```

- Displays the contents of the `objcode.txt` file, which contains the object code produced after Pass 2.

- 7.3 `displayObjectCode`

```
void displayObjectCode(HWND hwnd)
```

- Displays both the intermediate file (address, label, opcode, and operand) and the corresponding object code from `objcode.txt`. It handles cases where there is no object code for specific instructions like `BYTE` or `RESW`.

Output Format

Address Label Opcode Operand Object Code

8. Error Handling

- The program uses `MessageBox` to notify the user if any file operations (opening or reading) fail, providing user-friendly error messages.

9. Summary of Files Used

- `input.txt`: Contains the source assembly code.
- `symtab.txt`: Stores the symbol table generated during Pass 1.
- `intermediate.txt`: Intermediate file generated during Pass 1.
- `objcode.txt`: Final object code generated during Pass 2.

10. User Interaction

- The user interacts with the program through the buttons displayed in the GUI. The actions available include:
  - Running Pass 1.
  - Running Pass 2.
- Displaying the intermediate file or object code in the output window.

11. User Perspective for running a sic assembler code

11.1 Install a C Compiler

To compile and run C code, you need a C compiler. Here are some common options:

- **Windows**: Install the **MinGW** compiler or use an IDE like **Code::Blocks** that comes with GCC (GNU Compiler Collection) pre-packaged.
- **Mac**: Use **Xcode Command Line Tools**. You can install it by running `xcode-select --install` in the terminal.
- **Linux**: GCC is usually pre-installed on most Linux distributions. If not, you can install it using your package manager (`sudo apt-get install gcc` on Ubuntu/Debian).

11.2 Choosing an IDE or Editor

- **IDEs** (like **Code::Blocks**, **Eclipse**, **Visual Studio Code**, or **Dev-C++**) provide a graphical interface to write, compile, and run your code easily.
- **Text Editors** (like **VS Code**, **Notepad++**, **Sublime Text**) allow you to write the code, and you can compile and run it via a terminal.

Compiling and Running the C Code

- There are different ways to compile and run C programs depending on whether you are using a **command-line tool** or an **IDE**.

Using the Command Line (Terminal/Command Prompt)

11.3 For Windows (Using MinGW)

1. **Open Command Prompt**:
   - Press `Win + R`, type `cmd`, and hit Enter.
2. **Navigate to the Program Location**:
   - Use the `cd` command to navigate to the directory where your C file is saved.

     ```
     bash
     Copy code
     cd path\to\your\program
     ```

3. **Compile the C Program**:
   - Run the following command to compile the C code using the **GCC** compiler:

```
bash
Copy code
gcc hello.c -o hello.exe
```

- If there are no errors, this command will generate an executable file (`hello.exe`).

4. **Run the Program**:
   - After compiling successfully, run the program with:

```
bash
Copy code
hello.exe
```

11.4 For Linux/Mac (Using GCC)

1. **Open the Terminal**:
   - On Linux/Mac, open a terminal (use `Ctrl + Alt + T` on Linux).
2. **Navigate to the Program Location**:
   - Use the `cd` command to go to the directory where your file is saved.

```
bash
Copy code
cd /path/to/your/program
```

3. **Compile the Program**:
   - Run the following command to compile the C code:

```
bash
Copy code
gcc hello.c -o hello
```

   - This will generate an executable file named `hello`.
4. **Run the Program**:
   - Execute the program by typing:

```
bash
Copy code
./hello
```

12.Programmer perspective for running a sic assembler code.

Using an IDE (Code::Blocks)

For users who prefer an IDE, here are steps to run the same C program in **Code::Blocks**:

12.1 Open Code::Blocks

- Launch **Code::Blocks** from your desktop or start menu.

12.2 Create a New Project

1. Click on **File > New > Project**.
2. Select **Console Application** and click **Next**.
3. Choose **C** as the language and click **Next**.
4. Enter the project name (e.g., `HelloWorld`) and the location where you want to save it. Click **Next** to finish.

12.3 Write Your C Program

1. Once the project is set up, click **File > New > Empty File**.
2. Write your C code (e.g., the `hello.c` program).
3. Save the file as `main.c` in the project directory.

12.4 Compile and Run the Program

1. To compile the program, click **Build > Build** or press `Ctrl + F9`.
2. To run the program, click **Build > Run** or press `F9`.
    - A console window will pop up showing the output (`Hello, World!`).

12.5 Build and Run Together

- To compile and run the program in one step, you can click **Build > Build and Run** or press `F9`.

---

6. Debugging and Handling Errors

12.6 Compilation Errors

- If there are any syntax errors, the compiler will display error messages. Fix the errors and recompile the program.

- Example error: `expected ';' before '}'` – this means you're missing a semicolon in your code.

12.7 Using Debugging Tools

- In IDEs like Code::Blocks, you can use the built-in debugger to find logic errors.
  - Set breakpoints to pause execution at specific lines and inspect variables or program flow.
  - Use **Debug > Start/Continue** to start the debugging process.

13. Conclusion

- This program demonstrates a basic SIC assembler with a Windows-based GUI. It performs two passes over the assembly code, generates a symbol table, intermediate file, and object code, and displays these outputs in a user-friendly manner.

****