Alexandria University

Final Project – Phase II

SIC/XE assembler

Zahraa Emara – 4558

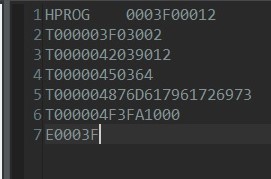
Sarah Yousry – 4582

Mayar El Mahdy – 4639

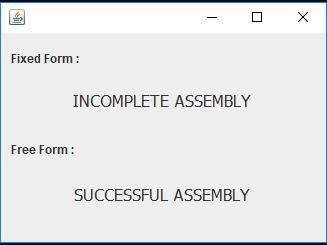
Mariam Beltagy – 5109

1. **Requirement specification**
2. OBJECT FILE

-Example of the object file



The object file is created by the three classes:

1. Header Record
2. Text Record
3. End Record
4. **Design**we used a simple GUI to display a message to show whether the assembling was successful or not. ****
5. **Main Data structures**

* Arrays
* Adjacency Matrix for creating the memory

1. **Algorithms Description**

* After completing the symbol table –Which is the last thing in phase one – Phase two will begin by adding the –Header record—in the Object file.



as you can see here after checking that there is no error in phase one then the symbol table is written then first the header record, then every text record and then finally the end record.

* The **Header Record** has the program name –if it has one- else it just puts empty spaces from column 2 to 7 , then column 8 to 13 is the starting address of the program that the user wrote , finally column 14 to 19 is the length of the program which will be (Last address – Start address – 1) .
* The **Text Record** will be written depending on the opcode , so there is a 2D array that has all the operations and their opcodes every operation has similar ways to be written in the Object file , as first you write H following the starting address of this operation, length of this operation and then it’s opcode , but –**WORD**—and –**BYTE**— are a bit different as in WORD it writes the starting address, it’s length and the number defined but changed to hexadecimal, while BYTE has two cases either it is a character or a hexa-decimal string , if it is a character then each character will be converted to asci code –HEXA—, then it will be written as the WORD but the number defined will be the asci code of each character , If the BYTE is a hexa-decimal string then the hexadecimal will be added as the number defined with the same value unchanged. Operations such as RESB and RESW are skipped as they are not translated to object code. As for the operand conversion we first check if the given string contains ‘+’ or ‘-’ so the string is split, and the expression is evaluated using the ExpressionEvaluation() method.
* The **End Record** simply writes at the end of the object file E in Column 1 and the address of first executable instruction.

A screenshot of a cell phone screen with text

Description automatically generated

1. **Assumptions**

* Assuming every operation done has its own text record – meaning that every line in the code except the start and end will have a full line of its own text record.
* Every number written in the object code is in hexa-decimal.

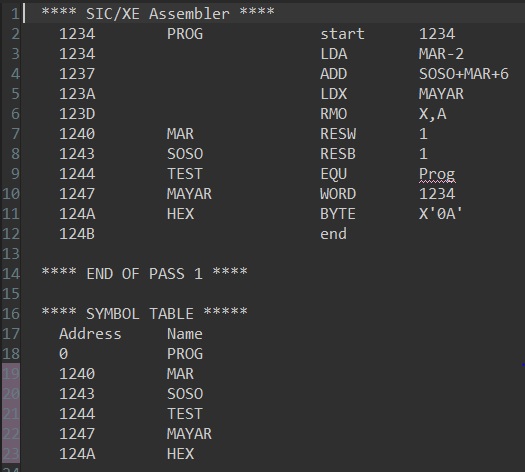
1. **Sample Runs**

### Implementation of the FIXED format with some error examples:

### **Source File:** **List File:**

### Implementation of the FIXED format (no errors):

### **Source File:** **List File:**

****

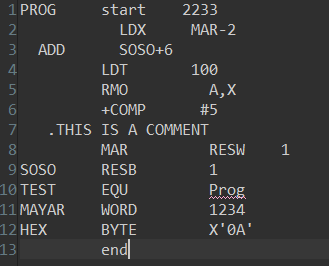
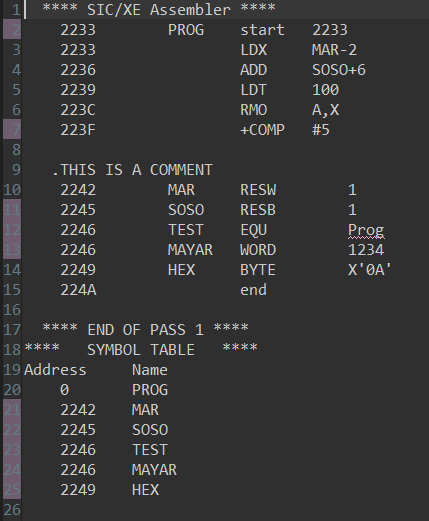
**Object File:**

**A close up of text on a black background

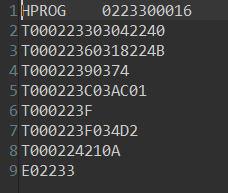
Description automatically generated**

### Implementation of the FREE format:

### **Source File:** **List File:**

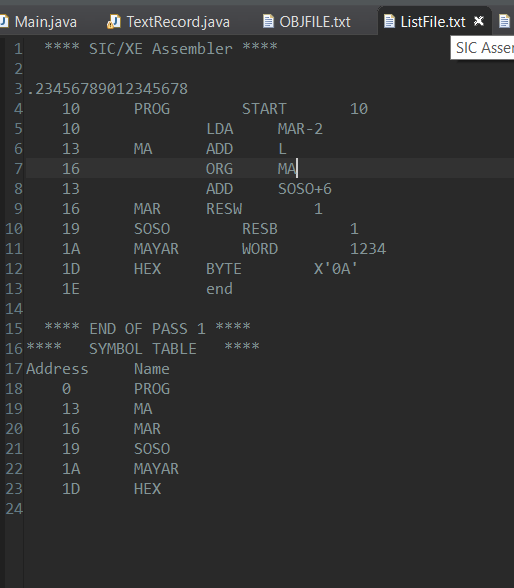
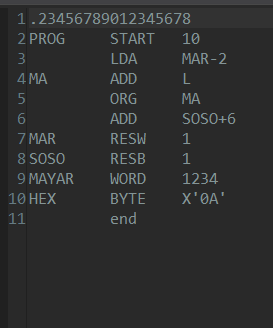
****

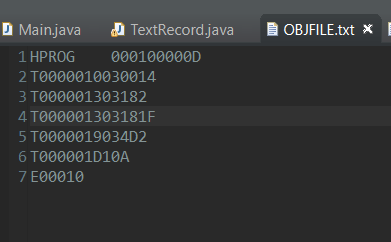
**Object File:**

****

### An example of using ORG:

### **Source File:** **List File:**

****

**Object File:**