Adam Hancock

Brian Rafferty

Youssef Iraqi

Lab #2

CS 530

**Software Design Document**

**System Specifications:**

**System Inputs:**

<*filename*>.obj: File containing the Header record, Text records, Modification records, and End records.

<*filename*>.sym: File containing the SYMTAB and the LITTAB that references labels by addresses.

**System Outputs:**

<*filename*>.sic: Source File

<*filename>*.lis: Source File with addresses and opcodes

**Level of Error of Processing Required:**

The program will throw an error if any of the following errors occur: can not open sym or obj files, the obj file contains Format 1 or 2 operations, a given label does not exist, a symbol does not exist, or the amount of arguments passed through is incorrect.

**Performance Requirements:**

Completely disassembles any SIC/XE code when given the object file and SYMTAB and LITTAB.

**Design:**

Code is written in C++

There will be multiple files to handle the processing of the object file and sym file.

- OPCode retrieval file

- Symbol table lookup file

- Main file - combines all files and processes through the object file to get the desired SIC/XE code.

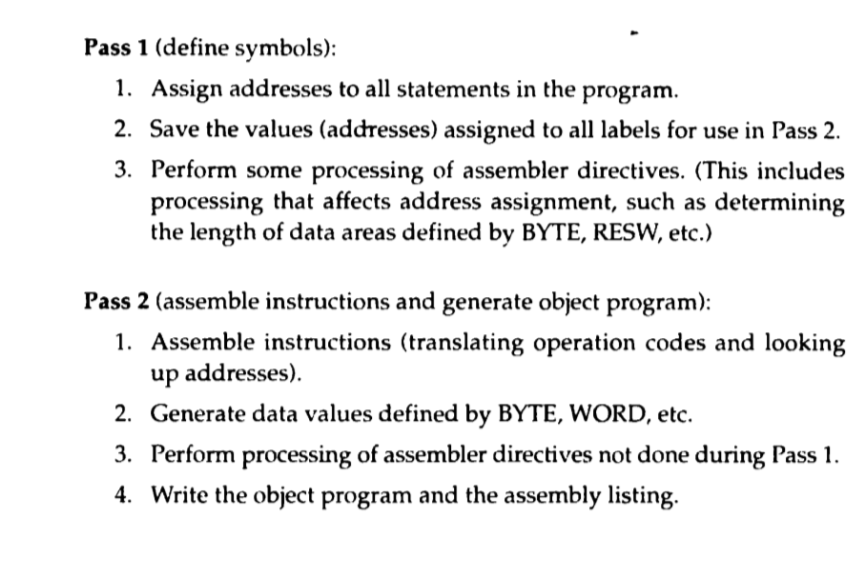
**System Software Design:**

In order to complete this assignment, there were weeks of planning and countless man-hours spent. We met up as much as was possible with our schedules in order to discuss the different possible processes to complete this task. When we would meet up, we would discuss topics such as: the best way to analyze the text records and then parse either three or four bytes, possible data structures to make modification and formatting easier, how to extract the information from the provided files, if there are any needs for special cases, and which members would be taking on which tasks.

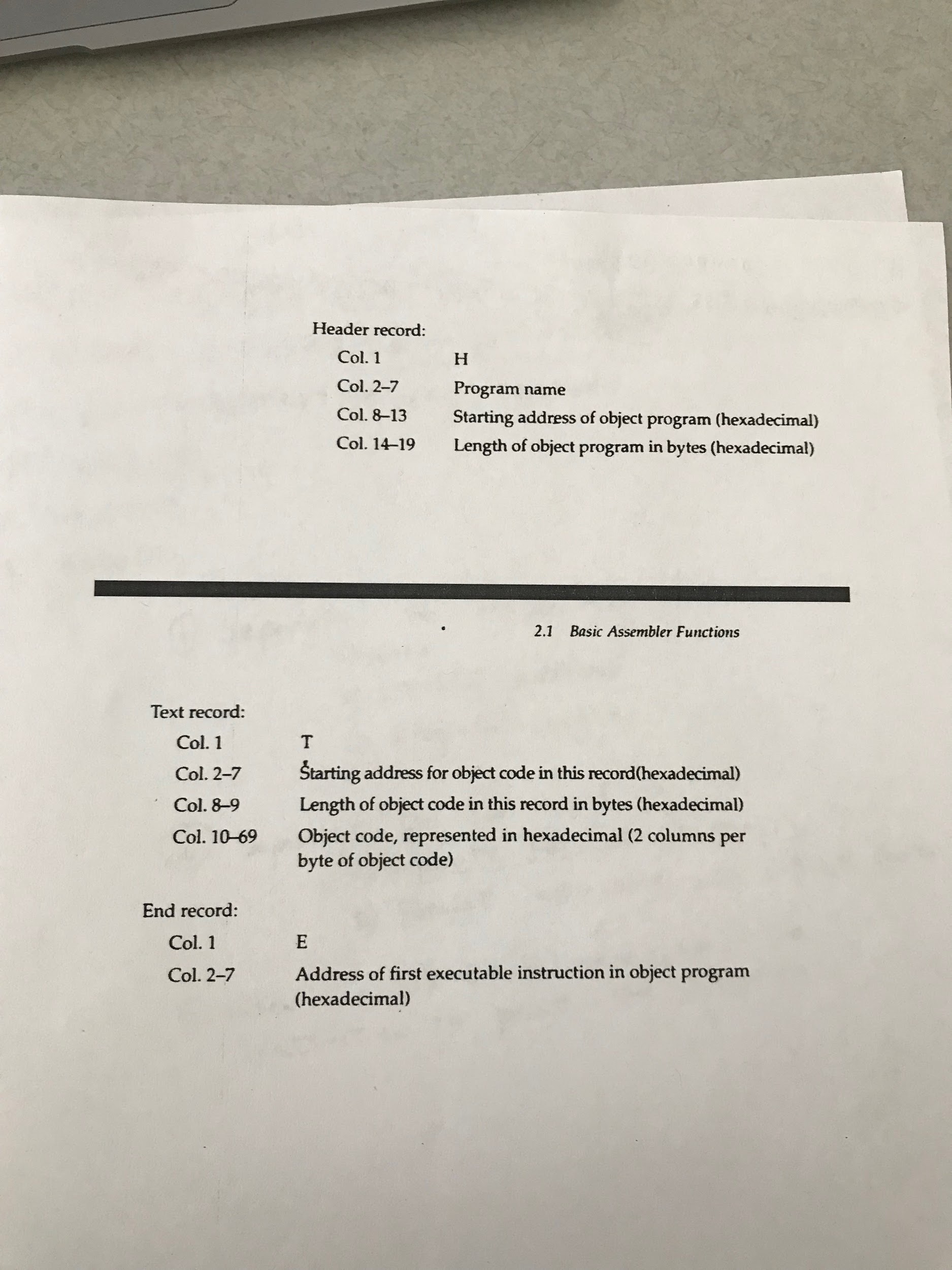
When we would meet, we were able to discuss how the project would read the files, how it would analyze the information from the .obj file and the .sym file, how to turn that data into the necessary .sic and .lis files, and finally worked on making everything look nice. We worked out how the program would analyze all given data by hand so that it would be easier to understand the algorithm when programming. When we were first writing the code, we had a large main file that did most of the necessary algorithm. Although we were able to get the program to work with such a large main, we felt that for any further debugging process and to make it easier to read that we should split it up into multiple files. Finally, once we divided up the program into all the necessary files and functions, we met up and put the final touches on it, including the comments and removing any of the unnecessary lines of code.

When the code is called using ./xed <*filename>*, first the program will check the command line arguments in case there are any errors on calling the program. Then the program will attempt to open a .obj and .sym file with the *<filename>* as well as creates the files: *<filename>*.sic and *<filename>*.lis. We have also implemented error checking for if the .obj and .sym files are unable to be opened. Once the files have been opened and there are no errors, the .obj file is read through. The .obj file is read in line by line and as each line is being read in, the program analyzes it to know all of the information for the .sic file such as the: program name,

length of file, all of the instructions, any modifications, and the address of the first executable instruction. Because the program goes through the .obj file one line at a time, we set it up in such a way that it checks if it is reading a header, text, modification, or end record.

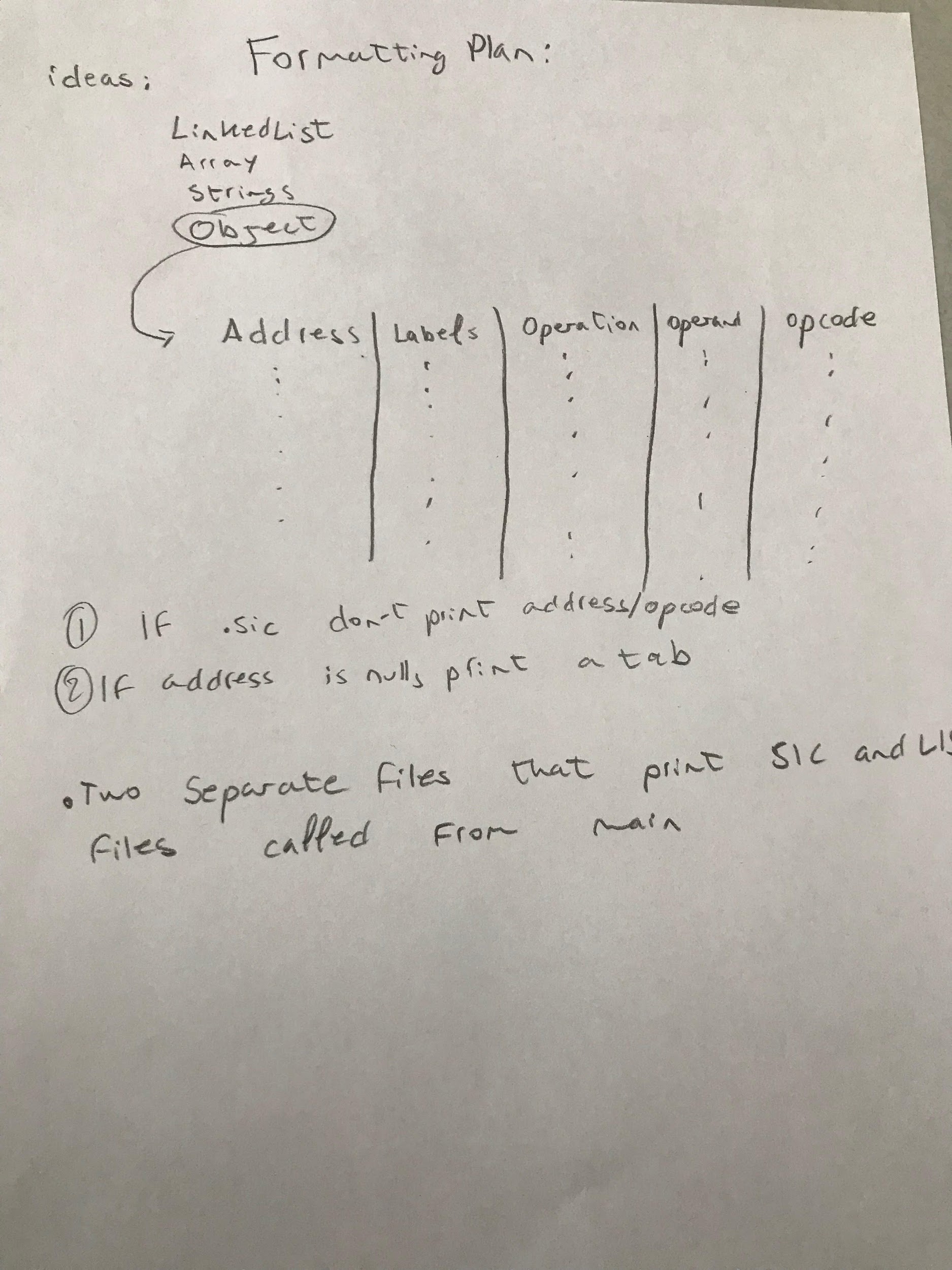


This picture describes the process that we were originally attempting to reverse. However, we realized that while using ofstream we cannot make a second pass because it will clear the previous information on the file when we made the second pass and so we had to change our thought pattern to adjust accordingly.



A picture of the rules for parsing the .obj file

When reading in from the text records, the program takes the first byte and analyzes it to determine if it is a format 3 or format 4 as well as reading in from the .sym file to check if there any labels that are supposed to be attached to the operation. It then checks the rest of the bytes, dependent on the format, and assigns the operation parameters along with the proper opcode. The program then goes in and adds a ‘+’ in front of the operation if it is a format 4. Then once it has the address, label, operation, operation parameters, and the opcode, it outputs the information in the proper format onto either the .sic or the .lis file. The .sic file will only contain information about the label, operation, and operation parameters, while the .lis also will contain the address for the commands and the corresponding opcodes. Then the program closes the files and finishes.



Formatting:

For the formatting we decided to use an object that contains :

string address;

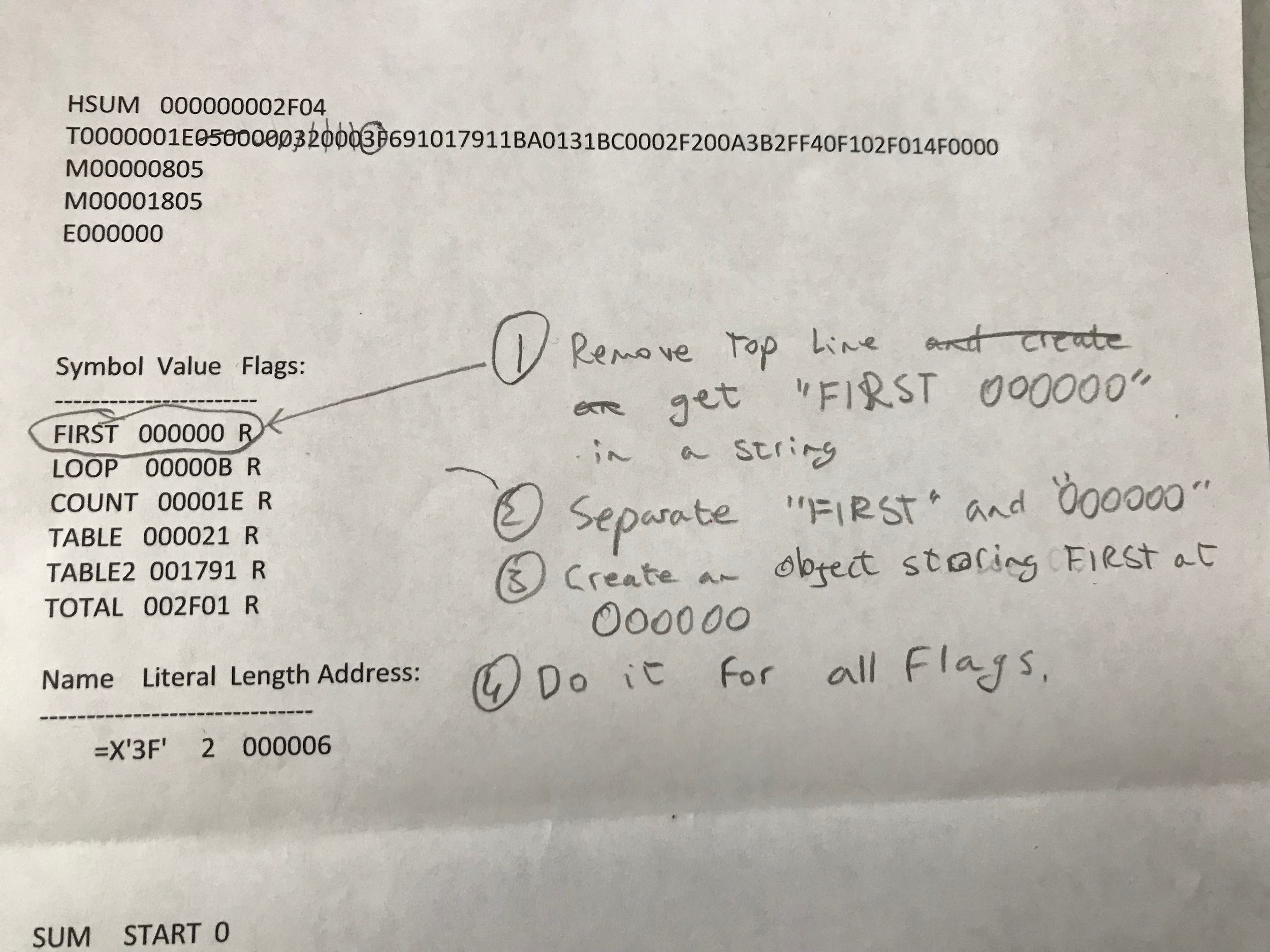
string label;

string operation;

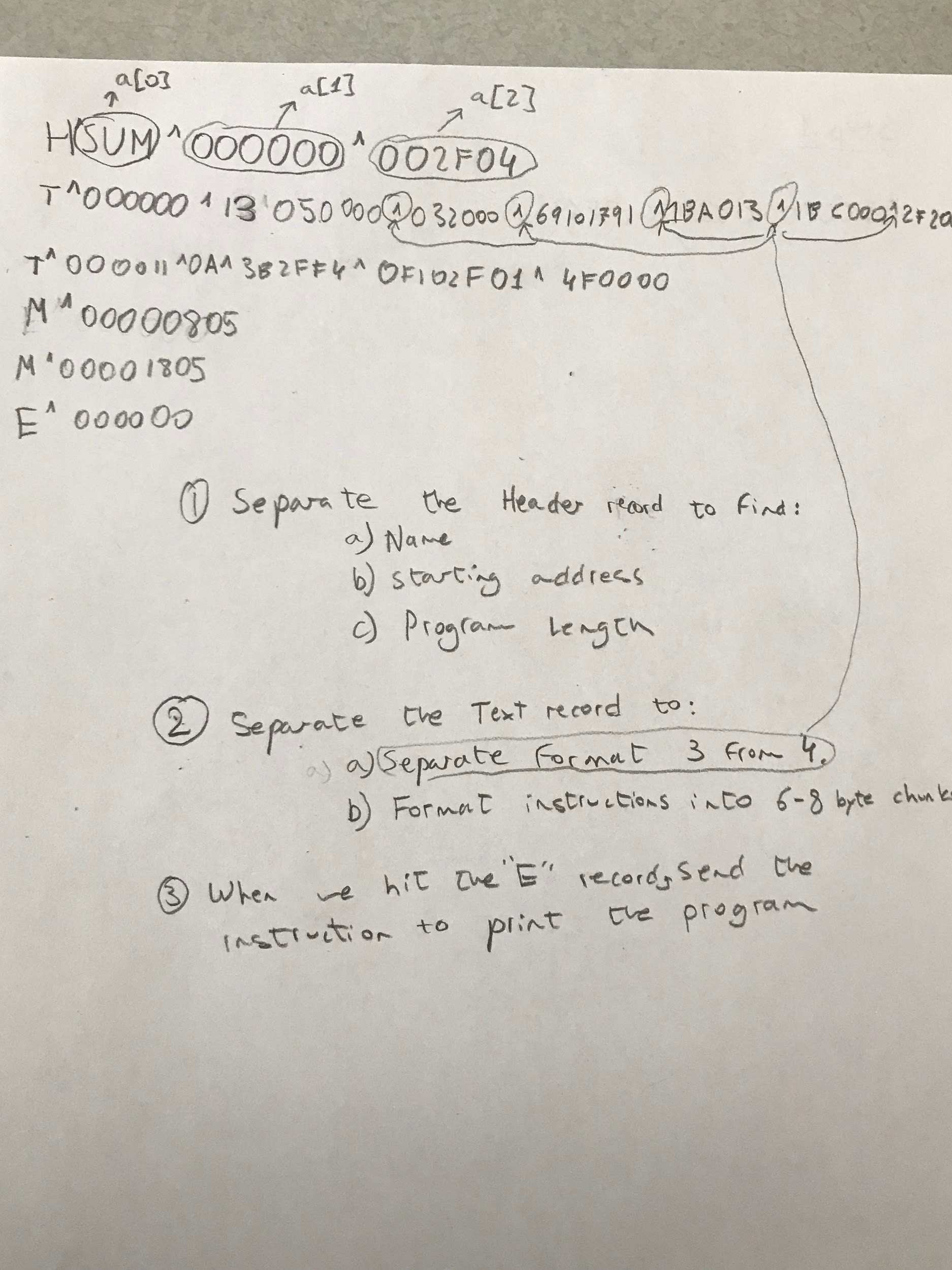
string operand;

string opcode;

Depending on if we are printing the .lis or .sic , we will send to the outputStream the elements needed to be displayed from the object created.



An example of how the program checks the .sym file when reading in from the text records. It compares the address from the .sym file to the address that it is currently on and if they match then it applies the label to the object before printing out to the file.



We decided to parse the obj file and take 4 steps. We check for H,T,M,E records, and work on each one separately. For the H record, we separate it into start address, and length, for T record, we parse it into format ¾ instruction sizes. Whenever we hit the E record, we sent the program to print everything out.

Testing:

For the testing part of our project what we did is use alot of “cout” to perform debugging and as well as breakpoints in code to see where errors were generated. One big issue was that when we tried calling elements from other classes things would not work since not everything was declared, and thus we used header files with public variables and methods.

Resources used:

<http://www.cplusplus.com/doc/>

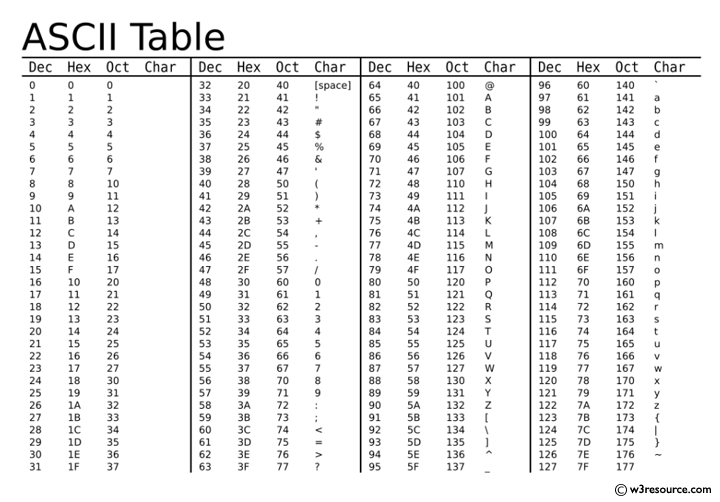
* We referred alot to the link above which is C++ documentation as we were debugging our programs and when we did not know how to use a specific element in c++.

<https://www.programiz.com/cpp-programming/object-class>

* We used this documentation to learn more about objects and classes so we could build our own object to save all of the information we were gathering

<https://www.quora.com/How-do-I-convert-a-char-to-int-in-C++>

* We were having trouble understanding char conversion so we used this forum which ultimately led us to this image of char to ascii conversion



*System Software: An Introduction to Systems Programming,* 3rd edition, Beck, Leland; Addison Wesley

* We referred to the assigned textbook a lot for further understanding of the XE and SIC architecture, as it took extensive understanding of the machines to build the program

<https://www.youtube.com/watch?v=W0aE-w61Cb8>

<https://www.youtube.com/watch?v=DTxHyVn0ODg>

* Two youtube videos that we watched on C++ pointers to familiarize and refresh on how pointers function and how to go about using them in our program as they were an inevitable part of our program

<https://stackoverflow.com/questions/247873/signed-versus-unsigned-integers>

* Learning about signed vs unsigned integers for bit manipulations.