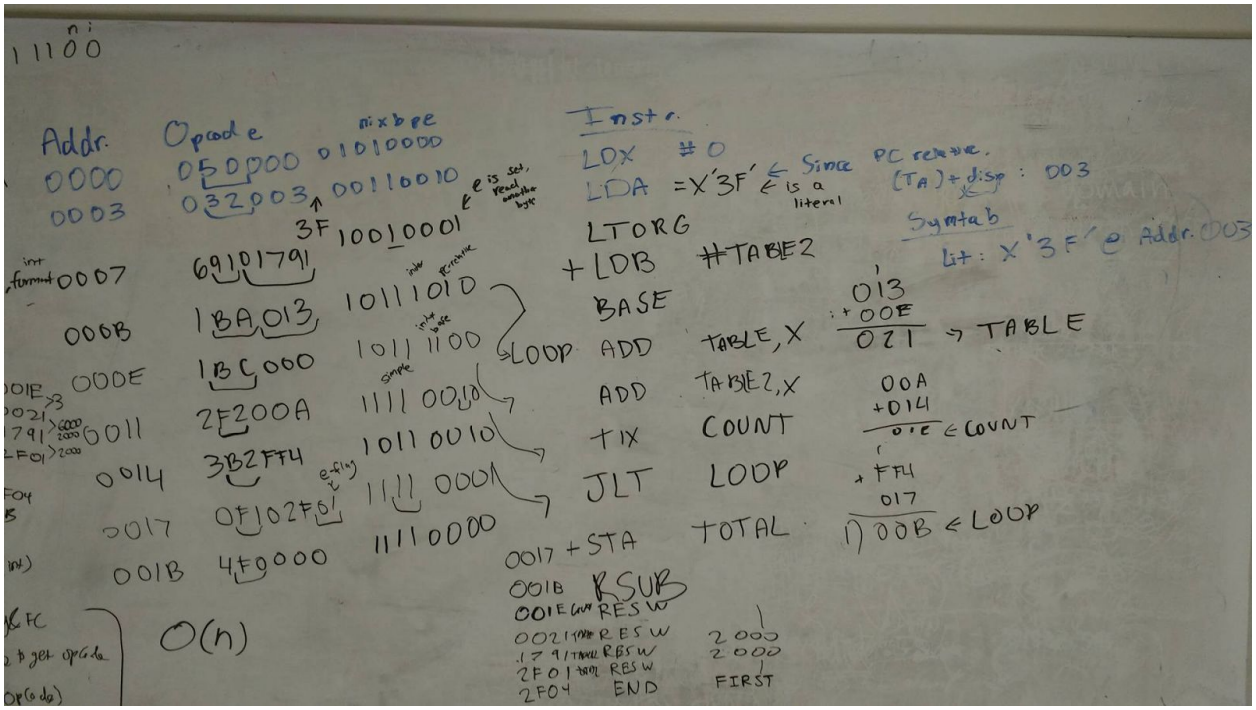
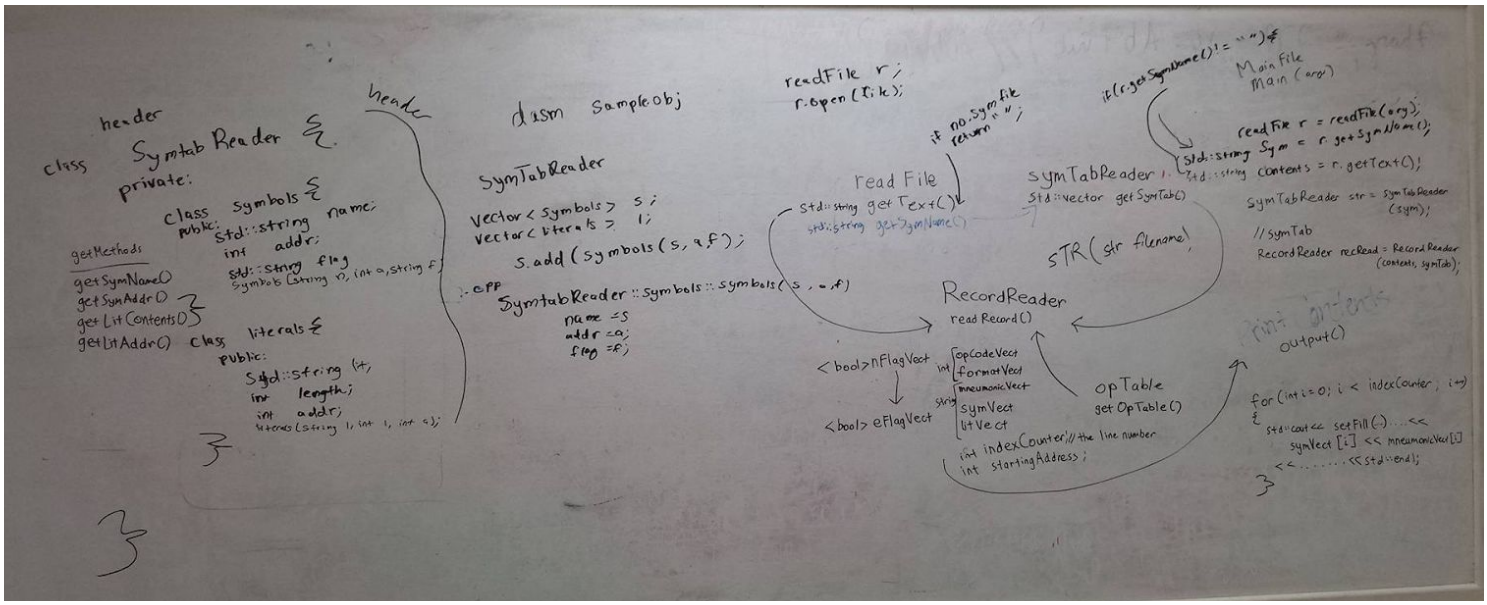
File name: Software Engineering Documentation
Members: Alexander Giang, Shawn Chua, Michael Yee, Tony La
Class Users: cssc0885, cssc0894, cssc0880, cssc0857
REDID: 818285579, 817662151, 819377332, 817862169
Class Information: CS530, Spring 2017
Assignment #2, XE Disassembler

System Planning:

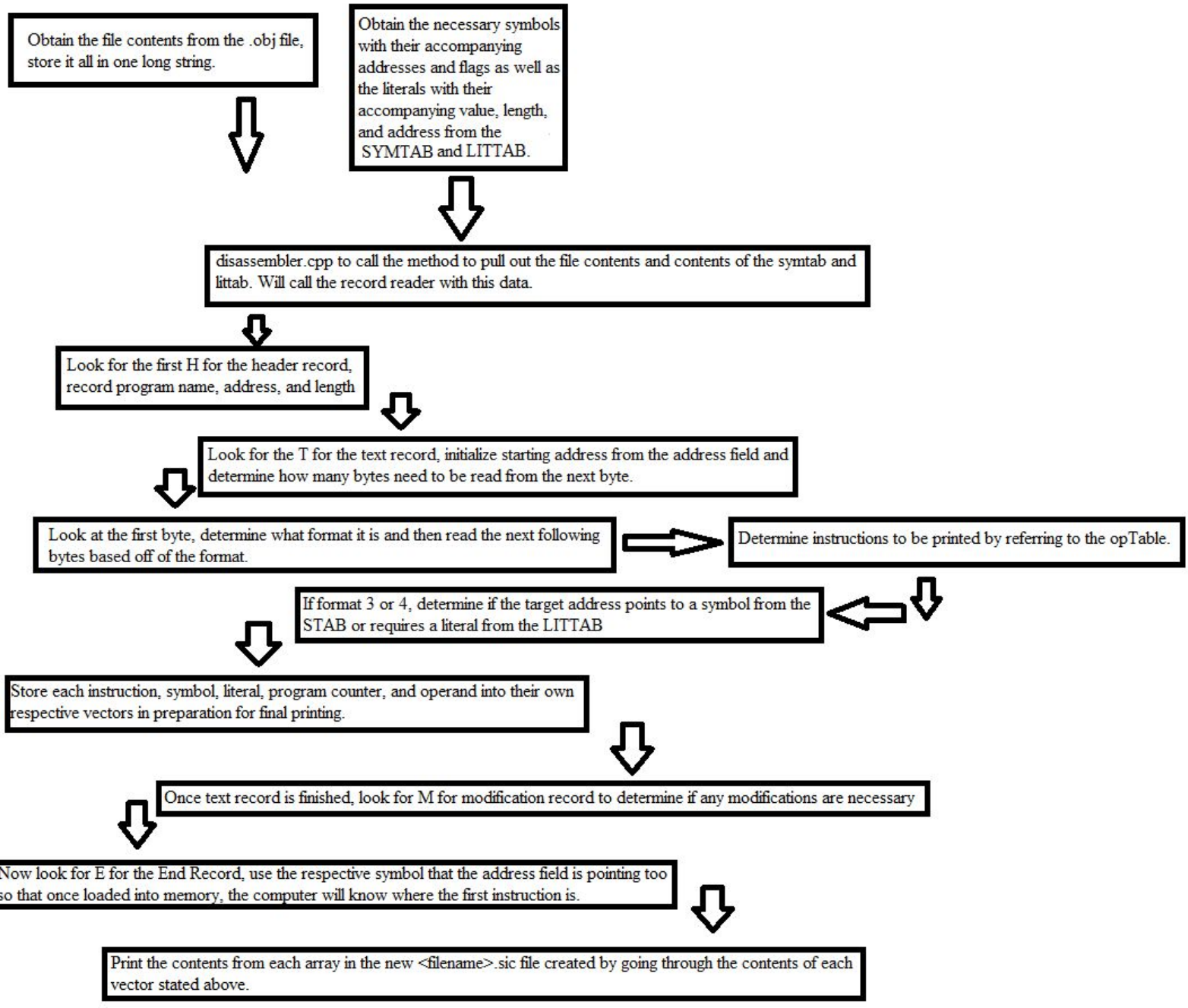
- Given the lab requirements and sample files to work with, we first tried to solve the problem by hand to figure out what parts we need for programming the disassembler:



- We followed an *Increment/Evolution* type of development because we were not sure how many methods and classes we needed at first:
    - To figure that out, we started by assigning each other parts:
        - Tony and Michael worked on disassembling
        - Shawn worked on reading and confirming the files
        - Alex worked on the SYMTAB and LITTAB reading
        - If we needed something from one another, we could allocate a blocked comment to be discussed with each other next meeting.
    - After a week, we came back together to figure out what else we needed from each other, such as reading the SYMTAB from one class to another.
        - We ended up shifting from working independently to helping each other on our parts.
    - After another week, we finalized anything else we needed via another collaboration. Here's a look at the process of our finalization:



- In between, we would upload our code on a github repository to store and review each other's codes and communicate through Facebook.
- Key points:
    - Since this is one of the first larger projects for most of us, we agreed on correctness over efficiency.


Obtain the file contents from the .obj file, store it all in one long string.
Obtain the necessary symbols with their accompanying addresses and flags as well as the literals with their accompanying value, length, and address from the SYMTAB and LITTAB.
disassembler.cpp to call the method to pull out the file contents and contents of the symtab and littab. Will call the record reader with this data.
Look for the first H for the header record, record program name, address, and length
Look for the T for the text record, initialize starting address from the address field and determine how many bytes need to be read from the next byte.
Look at the first byte, determine what format it is and then read the next following bytes based off of the format.
Determine instructions to be printed by referring to the opTable.
If format 3 or 4, determine if the target address points to a symbol from the STAB or requires a literal from the LITTAB
Store each instruction, symbol, literal, program counter, and operand into their own respective vectors in preparation for final printing.
Once text record is finished, look for M for modification record to determine if any modifications are necessary
Now look for E for the End Record, use the respective symbol that the address field is pointing too so that once loaded into memory, the computer will know where the first instruction is.
Print the contents from each array in the new <filename>.sic file created by going through the contents of each vector stated above.

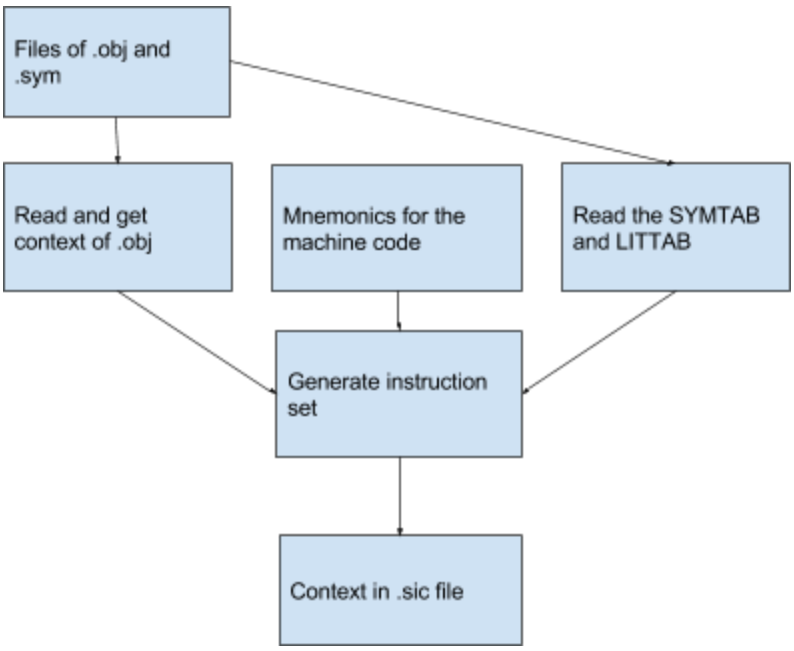
System Design:

- **We essentially have 4 core files:**
    1. *RecordReader.cpp*
    2. *opTable.cpp*
    3. *readFile.cpp*
    4. *symTabReader.cpp*

1. **RecordReader.cpp**
    a. **Purpose**: disassembling the object file to create the machine instructions
        i. Includes the Header, Text, End, and Modification Records
    b. **Files included**: opTable.cpp
        i. To figure out which OPCODEs match with which mnemonics
2. **opTable.cpp**
    a. **Purpose**: Table with all the mnemonics, opCode, and formats for each instruction.
        i. Includes getting the Format and the Mnemonic (e.g. LDA vs +LDA)
3. **readFile.cpp**
    a. **Purpose**: obtaining the contents of the .obj file and storing it into a string.
        i. Checks for filename validity, stores contents into string for further processing.
4. **symTabReader.cpp**
    a. **Purpose**: reading the SYMTAB and LITTAB
        i. Storing the contents into vectors (in case there are more symbols and literals) to be called when needed for outputting the machine code.

- **Other files**:
    a. disassembler.cpp
    b. makefile
    c. README.txt
- **disassembler.cpp** {main class}
    - **Purpose**: brings everything together, fully disassembling & outputting the machine code
- **makefile**
    - **Purpose**: compiles everything
- **README.txt**
    - **Purpose**: Contains information regarding the project

Diagram of XE Disassembler:



Verification & Test Design:

- We error-checked and tested our respective parts.
- Some of the methods used:
    - Creating our own tester classes & using the sample files given to check our work
    - Outputting the contents of the methods and/or variables to check if information is stored or manipulated correctly
    - While the file is open, parse the file.
- For error checking, we also attempted to run the program in a variety of situations to make sure that everything was functioning properly. We made sure if it delivered the proper error prompt provided the file to run with was not a .obj file or if the file did not have an accompanying .sym file.