SIC-XE Disassembler Design Document

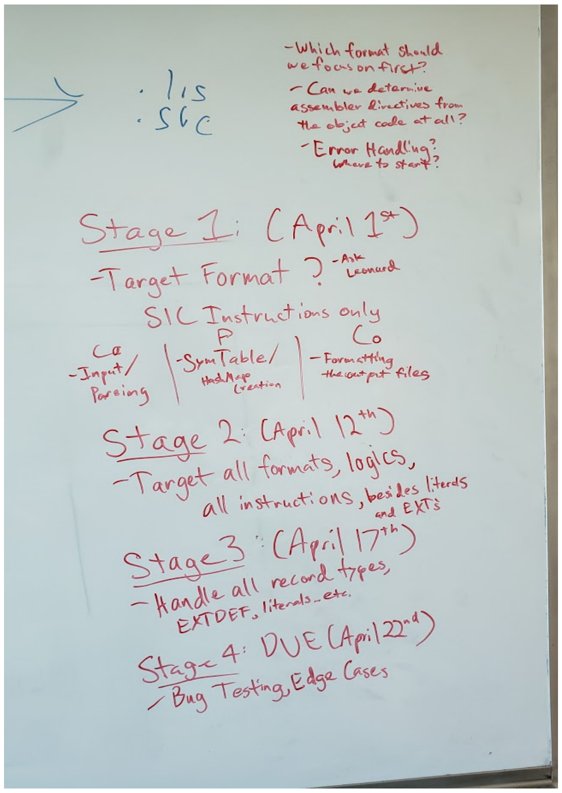
- Names: Carter Andrews, Colin Nugent, and Patrick Perrine
- Red ID's: 820214623, 820151963, and 820486635
- Class Accounts: cssc0425, cssc0440, and cssc0426
- CS 530, Spring 2019

Project Goal:

To develop, test, and deliver a disassembler program for the XE variant of the SIC/XE family of machines.

Project Objectives:

The XE disassembler program shall open an XE object file, .obj and its accompanying symbol file, .sym, then it will disassemble the object code, and generate an XE source file, and XE listing file, .lis, using the disassembled code. The symbol file, .sym will contain the SYMTAB and LITTAB the assembler generated when assembling the object file.



.lis
.sic

-Which format should we focus on first?
-Can we determine assembler directives from the object code at all?
-Error Handling? Where to start?

Stage 1: (April 1st)
-Target Format ? -Ask Leonard
SIC Instructions only

Ca	P	Co
-Input/ Parsing	-SymTable/ HashMap Creation	-Formatting the output files

Stage 2: (April 12th)
-Target all formats, logics, all instructions, besides literals and EXT's

Stage3: (April 17th)
-Handle all record types, EXTDEF, literals, etc.

Stage 4: DUE (April 22nd)
-Bug Testing, Edge Cases

Project Design:

Our team divided up our tasks to handle the specific functions of the program, and give ourselves timed stages to complete the project (See left photo.)

As seen in the photo, we had Carter and Colin initially handling input and output, respectively. Patrick was assigned to create the SymTable data structure.

In the next stage, we scheduled to have most of the project completed, and with the next stage having to deal with more specific cases and instructions.

We also scheduled the final stage to be purely bug testing.

We decided to divide up the code in the following design format:

- Driver Program (xed.h / xed.cpp)
- Main Processing Code (RecordProcess.h / RecordProcess.cpp)
- Helper Functions for Data Type Conversions (convert.h / convert.cpp)
- Helper Functions for OpCode Processing (opcode.h / opcode.cpp)
- Compilation (makefile)

We also utilized a private GitHub repository to handle version control and pair programming.

Lessons Learned:

As the project continued, we began to fall behind schedule, along with a member losing grasp of how the program was functioning as it was being developed. This was mostly due to scheduling conflicts of our other coursework.

The stages that we designed ended up not being beneficial to how the program should have been developed. This did not help with us staying on schedule or staying structured in our approach.

It was also difficult to conduct pair programming with a group of three members. Towards the end of the project, one person’s task was completed by another member of the project unknowingly. With this mishap, we learned the value of constant communication and task delegation.

In addition, we were ending up with a lot of code that looked functional, but lacked significant testing. One example of the kind of problems we ran into was that the SymTable processing was registering literals as instructions, and this problem took the entire team to figure out. We learned the importance of incremental testing through this.

On a positive note, all team members kept in good contact with one another throughout the project, especially towards the due date. We were able to talk about our development approaches in a concise manner, and have a reasonable respect for each of our programming abilities. This helped us delegate tasks, stay up-to-date with each other, and work well to create a functional program.

Project Outcome:

In the end, we were able to complete the project, as it functioned as intended with all the the test input files that we were given. Our disassembler properly disassembles given .obj and .sym files, and generates .sic and .lis files. However, we were only given one set of .obj and .sym files, so there may be certain functions/directives that we couldn't have accounted for.

This project helped us understand C/C++ programming better, understand the SIC/XE instruction set (which will help us with instruction sets in practice), and understand the role of an assembler/disassembler in executable compilation. We hope to remember the lessons learned with the project, and make effective programmers/software engineers in our future.