##########################################################################

## README

Team Members: Sierra Morehardt, Stefani Bekker, Ryan Aguiar

Class Accounts: Cssc0419, Cssc0424, Cssc0480

CS 530, Spring 2019

Assignment #2, XE Disassembler

## README

## Files Included:

Main.cpp: This file entails the main functionality for the program.

bitMask.cpp: This file changes last two bits of the first byte of object code to 11, in order to properly find the opcode.

calculateAddress.cpp: Calculates the addresses of each line as well as the base register if there is one.

Convert.cpp: Converts between different data types, which is necessary for proper functionality.

file\_parser.cpp: Deletes any comments from the code and separates it into different tokens for easier parsing

Optab.cpp: Creates an opcode table. We have it creating two different ones because as we were working separately, two of us created different ones and used them differently in the code. This is one of the times where we know the functionality could be improved, and we learned it was better to work together as much as we could in order to avoid issues such as this one.

printRecords.cpp: Prints out header, text, and end record as well as resw and resb.

symtab.cpp: Creates a symtab out of the given .sym file. This is another time where we have some duplicate functions due to working separately.

## Header files:

bitMask.hpp, calculateAddress.hpp, convert.hpp, file\_parse\_exception.hpp, file\_parser.hpp, opcode\_error\_exception.hpp, Optab.hpp, printRecords.hpp, sicxe\_asm.hpp, symtab\_exception.h, symtab.hpp.

## Makefile:

makefile

## Test Files:

sample.obj

sample.sym

## Compile and Run:

To compile and run our program, the user must first enter the directory containing the files above. We have included a ‘makefile’ in order to compile the necessary files to create the executable. Running the ‘make’ command produces the executable ‘xed’. Following compilation the user runs the executable on the command line by calling ‘./xed <filename>.obj’. The executable disassembles the provided file per the necessary requirements. The result of a successful run is a populated <filename>.sic and <filename>.lis.

Example:

> cd XE\_Disassembler

> make

> ./xed sample.obj

## Design:

Our design for the disassembler followed the prompt accordingly. To reiterate, “*The XE disassembler program shall open an XE object file, <filename>.obj and its accompanying symbol file, <filename>.sym, then it will disassemble the object code, and generate an XE source file, <filename.sic> and XE listing file, <filename>.lis using the disassembled code. The symbol file, <filename>.sym will contain the SYMTAB and LITTAB the assembler generated when assembling the object file. The disassembler will then use “filename” for the name of the source file it generates, <filename>.sic, and the accompanying listing file, <filename>.lis. Note, the symbol file <filename>.sym will be provided with the object code file. If neither the <filename>.obj or <filename>.sym are present, the xed program shall gracefully exit.”* Along with this design, we implemented some of our own functions and files. In the ‘calculateAddress’ header and program file you can see how we accounted for the assortment of calculations required based on the address mode. For example, when calculating an address in Base or PC relative addressing, it was necessary that additional steps be taken to include the value of these registers. Another important decision made was how to tokenize the provided object file. We included a file called ‘file\_parser’ that removed any and all characters that would not be needed in order to populate our .lis and .sic files. Lastly, we included a version of a number type converter called ‘convert’. This file contained the functions needed to jump between numbers in binary, decimal, and hex format when necessary. In regards to the remaining files, they followed the design that was instructed via the prompt.

## Extra Functionality:

Aside from the requirements, we added some extra error checking. This includes, whether or not the files exist, checking if the file passed in is empty, and checking if it has the wrong extension. Inside the .lis file, we also added line numbers in order to be able to better read the code.

## Conclusion:

In all, our team would consider our project a success, however as a newly formed group, we experienced our share of roadblocks. In regards to the program itself, there are a few instances where we implemented the same functions. As we attempted to divy up the project, we found ourselves needing some of the same functions. In regards to our team, we found it increasingly difficult as students to work simultaneously on the project. When working in a team, typically there is much time to work side by side, however due to our varying schedules, we found some difficulty doing so. To make up for this, we were in constant digital communication, as well as repeatedly updating our each other on our progress.

##########################################################################