CS 530

Assignment 2 - Disassembler

Points: **100**

Due Date: **Beginning of the class, 10/30/2023**

In this assignment, you will continue to accomplish the SIC/XE disassembler based on what you have built in the first assignment. Again, a disassembler is a computer program that translates the machine language into assembly language.

Task

In this assignment, you would implement a disassembler that should be able to open an object code file, parse the records in the object code file (partially done in your first assignment), and output the corresponding assembly code to an assembly listing file. The assembly listing file format can be seen in the sample assembly listing file out.lst.

Disassembler Input and Output Files

- **Input** Files:
    - Object Code File: **test.obj**
        - Object code file would have the typical SIC/XE object code structure that starts with a HEADER record, followed by several **TEXT** records, followed by (possibly) several MODIFICATION records, and ends with an END record. Note your disassembling would only need to **parse** the **TEXT** records.
    - Symbol Table File: **test.sym**
        - symbols, variables, constants and literals used in the assembly code.
        - See below for a detailed description of the symbol table file format.
    - These two input file names are passed as command line arguments to the disassembler executable (see "compilation and execution section" below).
- **Output** File:
    - Assembly Listing File: **out.lst**
    - Your disassembler **must** produce the output file as **out.lst**.

- There needs to be some spaces separating columns in the **out.lst** file. The number of spaces for the separation is flexible. It is recommended to set the columns with a consistent width of **12** characters so output format looks consistent.

Format of the Symbol Table file:

- The symbol table file (**test.sym**) contains **SYMTAB** and **LITTAB** tables generated from the assembling process, and they are needed for the disassembling process. The **first** table in the symbol file is the **SYMTAB** and the **second** table is the **LITTAB** (refer to lecture slides).
  - **SYMTAB** contains symbols or labels to the instruction statements as well as variables defined by the RESB or RESW directives.
    - You can assume the SYMTAB always starts with the first two lines in the test.sym file. The column headers are separated by an arbitrary number of white spaces.

```
Symbol  Address Flags:
----------------------
```

    - ***Symbol*** column has the name of each label or variable.
    - ***Address*** column has the address (**relative to the start**) or value of each symbol.
    - ***Flags*** column indicates if the symbol is a relative term (meaning an offset) to the starting address, in this assignment, only R would be appearing and applicable.
    - For PA2, the symbol table only includes the non-variable label “FIRST” in the test.sym file that is used to indicate the start of the assembly code.
  - **LITTAB** table lists both **CONSTANTS** and **LITERALS**, which are essentially constants.
    - You can assume the LITTAB always starts with the following two lines as seen in the test.sym file. The column headers are separated by an arbitrary number of white spaces.

```
Name    Lit_Const  Length Address:
---------------------------------
```

- ***Name*** column is only applicable to constants.
- ***Lit_Const*** column has the value of each constant.
- ***Length*** column has the size in nibbles (a nibble has 4 bits) for each constant, e.g., 2 means 1 byte.
- ***Address*** column has the **relative address** of each constant or literal to the starting address.
- Columns in the LITTAB are separated by an arbitrary number of white spaces; do not assume the number of white spaces is a fixed number.

Required Disassembler Features

- **Instruction formats**: your disassembler MUST support **format 2, 3, 4** instructions, and does NOT need to support format 1 instructions.
- **Constants**: your disassembler MUST support **BYTE constants** defined through the **BYTE** directive. You only need to worry, however, about **the constants with HEX format**. **Example: X'01' and X'000001'.** And you do **NOT** have to support:
  - constants defined with CHAR format (e.g., C'EOF'), or
  - WORD constants
- **OP codes**: your disassembler shall support the translation between all 59 operation codes (except format 1 instructions) and their corresponding mnemonics in the SIC/XE architecture. You already finished this task in PA1.

Teams

You can pair in a group of **2 students** and collaborate while coding for this assignment, but you should provide learning outcomes for each student in the group separately in a document. You should not collaborate with anyone else other than your teammate. Do NOT exchange your code with other teams. Do NOT copy and paste code from any online resource. You can do this assignment on your own as well.

Programming and testing

You must use C/C++ for this assignment.

- Refer to "C/C++ programming in Linux / Unix" page.

- You may use C++ 11 standard for this assignment, use compilation flag *-std=c++11.*
- You are strongly recommended to set up your local development environment under a Linux environment (e.g., Ubuntu 18.04 or 20.04, or CentOS 8), develop and test your code there first, then port your code to Edoras (e.g., filezilla or winscp) to compile and test to verify.

Compilation and execution

- Compilation: You MUST provide an appropriate compilation command as you did in Assignment 1.
- Executable Name: Please generate your dissembler executable file with a name **disassem**. The autograder will fail and you will lose points if we need to modify your executable name to make the autograder work. Make sure to use the -o dissem flag in your compilation command, also you should use a -g compile flag to generate gdb debugging information.
- Before submitting, port your code to Edoras (using filezilla or scp) to compile and test to verify. Note the grader will use a similar environment as Edoras to compile and grade your code.
- Executable Command Line Arguments: The disassembler should use the object code file name and its accompanying symbol file name, in that order, as the command line arguments for execution. If either the object code file or the symbol file is not present, then the program should exit by displaying a message asking for the missing input file/s. The following is an example of how your executable will be called:

  ./disassem test.obj test.sym

Grading

Passing tests against your program execution may NOT give you a perfect score. The satisfaction of the requirements (see above), your code structure, coding style, and commenting will also be part of the rubrics (**see Syllabus Course Design - assignments**). Your code shall follow industry best practices:

- Be sure to comment your code appropriately. Code with no or minimal comments is automatically lowered to one grade category.
- NO hard code – magic numbers, etc.
- Have proper code structure between .h and .c / .cpp files, do not #include .cpp files.
- Design and implement clean interfaces between modules.

Turning In

For each pair programming group, submit the following artifacts by **ONLY ONE** group member in your group. Make sure that all files mentioned below (Source code files, Makefile, Affidavit) contain each team member's name and Red ID!

- Program Artifacts
  - Source code files (.h, .hpp, .cpp, .C, or .cc files, etc.), and **Makefile**
  - Do NOT **compress / zip** files into a ZIP file and submit, submit all files separately.
  - Do NOT submit any .o files or test files

- Academic Honesty Affidavit (no digital signature is required, type all student names and their Red IDs as signature)
  - Pair programming Equitable Participation and Honesty Affidavit with the members' names listed on it. Use the pair-programmer affidavit template.
  - If you worked alone on the assignment, use the single-programmer affidavit template.

- Number of submissions
  - For this assignment, you will be allowed a total of **15** submissions. As stressed in the class, you are supposed to do the testing in your own dev environment. It is also the responsibility of you as the programmer to sort out the test cases based on the requirement specifications instead of relying on the autograder to give the test cases.
  - Note as a group, you would **ONLY have a total of 15 submissions** for the group. If the two members of a group submit separately and the total submissions together between the two members exceed 10, a **30%** penalty on your overall assignment grade will be applied. This will be verified during grading.

Late Submission Policy

Refer to the Syllabus for the Late Submission Policy.

Academic honesty

Course materials including Assignment prompts are strictly SDSU properties. Posting this assignment on any online learning platform such as Chegg is considered committing plagiarism. See Syllabus for plagiarism policies.

We will be using plagiarism detection software to generate similarity reports of your code with your peers as well as from **online sources**. **We will also include solutions from the popular learning platforms (such as Chegg, Github, etc.) as part of the online sources used for the plagiarism similarity detection**. Note not only the plagiarism detection software used to check for matching content, but also it checks the structure of your code. This means that switching up a line below another line while copying someone else's work would still be detected. Additionally, the underlying software replaces all variable names with a placeholder name, which allows it to detect similarity even if the variable names have been altered. You are free to discuss ideas and strategies for approaching problems with others, but each student or group must complete his/her/their own work. So do not give access to your program in any form to another student; sharing a program carries the same penalty as using a shared program. Depending on the severity of the similarity, actions will be taken as outlined in CSU academic integrity policy above. If a student is found cheating or plagiarizing material, that student should expect to face disciplinary proceedings

through an academic dishonesty report filing. See course syllabus and University Policies for a thorough definition of academic honesty and dishonesty.