

# CSE4100-1: System Programming (Spring 2019)

## Project #2: Assembler

Handed out: March 26, Due: April 8, 11:59PM (KST)

### 1. Goal

This project will add additional functions to the shell program you developed in project 1. Now the program will read an SIC/XE assembly program source file as an input, and create an object file and an assembly listing file. Also, the program should be able to display the symbol table and the object file for the user. You should implement functions specified in the textbook chapters 2.1 and 2.2.

### 2. Requirements

#### 2.1. Overview

- You need to support the following commands. Note that your program should also support all commands from the previous project.

- shell related commands: **help, type**
- assembler related commands: **assemble, symbol**

#### 2.2. User-Command Specification

##### 2.2.1. Shell related commands

(1) h[elp]    ➡ user can enter 'h', or 'help'.

This command will display all commands supported by the shell. This command was in the project 1, but should be modified to display newly added commands.

Example:

```
sicsim> help
```

```
h[elp]
d[ir]
q[uit]
hi[story]
du[mp] [start, end]
e[dit] address, value
f[ill] start, end, value
reset
opcode mnemonic
```

opcode list  
assemble filename  
type filename  
symbol

※ For this command and all other commands, the shell prompt should appear again once the program executes one command. (The program should not end after executing one command.)

(2) type filename

This command displays content of a file. If the file does not exist, an error message is printed.

※ You must NOT implement this command using system call functions.

Example:

```
sicsim> type a.obj
```

```
HCOPY 00100000107A
T00101E150C10364820610810334C0000454F46000003000000
E001000
```

## 2.2.2. Assembler related commands

(1) assemble filename

This command reads the given source file, and outputs an object file and a listing file.

- Filename extension of a source file is “.asm”.
- Filename extension of a listing file is “.lst”. The name must be the same as the source file.
- Filename extension of an object file is “.obj”. The name must be the same as the source file.
- If there is an error in the source file, do not create either file. Print the errors on the display and finish the command.
- For debugging, specify which lines have errors.

Example:

```
sicsim> assemble 2_5.asm
```

This command will create “2\_5.lst” and “2\_5.obj” as outputs.

```
sicsim> type 2_5.asm      (Figure 2.5 in the textbook)
```

```
COPY      START    0
FIRST     STL      RETADR
...
```

```
END      FIRST
```

```
sicsim> assemble 2_5.asm
```

```
output file: [2_5.lst], [2_5.obj]
```

```
sicsim> type 2_5.lst      (Figure 2.6 in the textbook)
```

```
0000    COPY    START    0
0000    FIRST   STL      RETADR    17202D
0003                LDB      #LENGTH 69202D
...
                END      FIRST
```

```
sicsim> type 2_5.obj      (Figure 2.8 in the textbook)
```

```
HCOPY 000000001077
T0000001D17202D69202D....
T00001D....
...
E000000
```

## (2) symbol

This command displays symbol table created during the assembling process. Symbol table should be implemented using a hash table, similar to the opcode table. Design of hash function is up to you, but you should follow the output format below.

- You should display the symbol table of the most recently assembled file.
- Symbols should be sorted by their names in descending order. (Z first, A last)
- Each line should start with a tab (\t), followed by the symbol name, followed by another tab (\t), followed by the address of the symbol, and finally a newline character (\n).

- Example

```
sicsim> assemble 2_5.asm
```

```
output file: [2_5.lst], [2_5.obj]
```

```
sicsim> symbol
      WRREC 105D
      WLOOP 1062
      ...
```

- Remember that the spaces before the symbol name and the address are tabs!
- Also, for only the last line, do not put the newline character at the end.

## 2.3. Notes

- In project 1, when you built the opcode table, you may or may not have included the format of each instruction in the table. In this project, you will need to know the format for each instruction in order to assemble the code. If you didn't include the format in the opcode table in project 1, modify the opcode table so that you can obtain the format in addition to the opcode, for a particular mnemonic.
- Your assembler should support SIC/XE assembly code. Generally, the assembler should be compatible with the original SIC assembly code, but we will not check the compatibility issues in this project.
- You are given an example assembly code, "2\_5.asm". But your assembler should assemble any code that is written in SIC/XE grammar.
- You need to implement all the features described in chapter 2.1 and 2.2 of the textbook. You do not need to implement anything that is in chapter 2.3 and beyond.
- The test assembly code "2\_5.asm" is the code in Figure 2.5 of the textbook. For this test case, you can check your result with Figure 2.6 (listing) and 2.8 (object program) for the result.

## 3. Submission

### 3.1. Things to Submit


- (1) program source: you must document your source code so that others can understand what each part of code is doing.
- (2) Makefile: you should write a Makefile that will compile your source. The binary file should be named using your student ID (explained later.)
- (3) A document file: this document file should describe how you implemented your program. When you write your document file, think like this: a new employee needs to take over your work and extend your program. He/she should be able to understand your code and be able to work on it after reading your document. A sample document will be posted on cyber campus.
- (4) A README file that includes simple instructions to compile and run the program.
- (5) opcode.txt: the file used as an input to the program.
- (6) 2\_5.asm: the test file

### 3.2. Instructions on Submission

- Make a directory named “sp20161234\_proj2”. The numeric part should be **your student ID**.
- Put all the files in the directory, and compress the directory itself using **tar**.
- Do NOT put binary files in the directory.
- For the source files, you may have one or more c files. Use your student ID as the name for the c file that contains the main function.
- You should have at least one header file. Use your student ID as the name for one of the h files.
- Write the Makefile so that the output binary file is named “20161234.out”. The numeric part should be **your student ID**.
- When you make a tar file, do NOT use the z option (which makes a gz compressed file.)

Example:

sp20161234\_proj2/

```
README
document.docx
20161234.c       This file contains the main function.
20161234.h
Makefile
opcode.txt
2_5.asm
```

The file for submission

sp20161234\_proj2.tar

upload this file on the **cyber campus**.

### 3.3. List for Self-Check before Submission

- (1) Did you implement everything specified in the requirements (including specific instructions for what data structure to use)?
- (2) Does your program compile successfully using gcc?
- (3) Does your program compile without giving any warnings?
- (4) Does your program run all the commands correctly?

- (5) Does your program handle exceptional cases such as wrong inputs?
- (6) Did you name all your files based on given instructions?
- (7) Do you have at least one header file?
- (8) Did you write a Makefile?
- (9) Did you write a README.
- (10) Did you document your source code properly?
- (11) Did you write a report document?
- (12) Did you include everything necessary in your tar file?

### **3.4. Late Submissions**

Late submissions are accepted for five days after the deadline. 10% of the points are deducted for each day. Submissions are not accepted after five days.