CSE4100-1: System Programming (Spring 2019)
Project #3: Linking Loader

Handed out: April 9, Due: May 6 (Mon), 11:59PM (KST)
※ no late submissions accepted for this project

1. Goal

This project will add additional functions to the shell program you developed in project 1 and 2. In this project, you will read one or multiple object program and load them onto the memory at specified location, possibly linking the files together. Also, you will execute the program loaded on the memory.

2. Requirements

2.1. Overview

- This project consists of two parts: loading (and linking) the object program onto memory, and executing them.

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

· loader related commands: **progaddr**, **load**

· execution related commands: **run, bp**

2.2. User-Command Specification

2.2.1. Loader related commands

(1) progaddr [address]

- This command will set the **program starting address** to *address*. If the *address* argument is not given, the program starting address becomes 0x00.

- Also, initially when the shell program starts, the program starting address is 0x00.

- The argument *address* is assumed to be in hexadecimal format.

- When the loader loads an object program, it should load the program at the program starting address given by this command.

Example:

```
sicsim> progaddr 4000

Program starting address set to 0x4000.
```

(2) loader [filename1] [filename2] [filename3]

- This command will read one or multiple object files, link them and load them onto the memory.

- The maximum number of files that can be loaded together is 3.

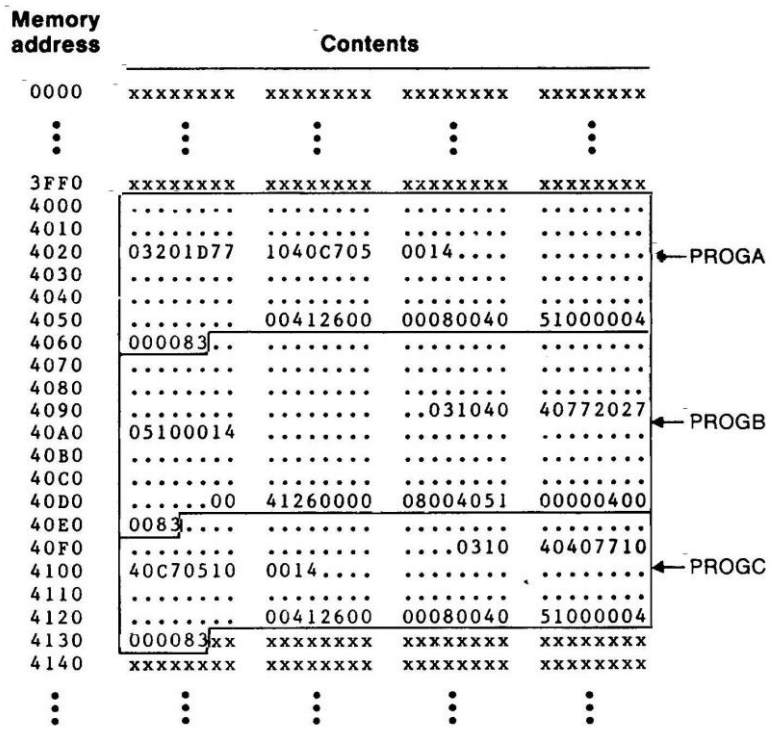
- If loading succeeds, show the load map on the screen (the table in textbook p.150). Refer to the example below.

- If there is an error, show the error message on the screen.

Example:

```
sicsim> progaddr 4000

Program starting address set to 0x4000.

sicsim> loader proga.obj progb.obj progc.obj

control          symbol          address          length
section          name
-----------------------------------------------------------
PROGA                            4000             0063
                 LISTA           4040
                 ENDA            4054
PROGB                            4063             007F
                 LISTB           40C3
                 ENDB            40D3
PROGC                            40E2             0051
                 LISTC           4112
                 ENDC            4124
-----------------------------------------------------------
                                 total length     0133

sicsim> dump 4000, 4133
```

- Your output should resemble Figure 3.12 in your textbook (also shown below.)



[Figure 3.12(a) in the textbook p. 146]

2.2.2. Execution related commands

(1) run

- This command will execute the program, starting at the program starting address (set by **progaddr**).

- You should be able to execute “copy.obj”. The assembly code for the program is shown in Figure 2.6 of the textbook. Specifically, you are required to implement the following 20 instructions. (You can implement more instructions if you want, but these 20 are mandatory.)

mnemonic	opcode	mnemonic	opcode
LDA	00	JLT	38
LDB	68	JEQ	30
LDT	74	RSUB	4C
LDCH	50	COMP	28
STA	0C	COMPR	A0
STX	10	CLEAR	B4
STL	14	TIXR	B8
STCH	54	TD*	E0
J	3C	RD*	D8
JSUB	48	WD*	DC

- The three instructions related to devices (TD, RD, and WD) cannot be implemented because there is no actual device. For these three instructions, do the following.

· TD (Test Device): Assume the device is always ready.

· RD (Read from Device): Assume the byte you read from the device is always 0x00.

· WD (Write to Device): Assume the write is done.

- After you run the program, print on the screen the contents of the following seven registers.

A X L PC B S T

※ (You should display the values of these seven registers, but you will also need to implement the SW register which contains the condition code.)

- If there are breakpoints, run until the nearest breakpoint. Otherwise, run the program until the end. (Refer to the **bp** command.)

Example:

```
sicsim> progaddr 4000

Program starting address set to 0x4000.

sicsim> loader copy.obj

control         symbol         address          length
section         name
-----------------------------------------------------------
COPY                           4000             1077
-----------------------------------------------------------
                               total length     1077

sicsim> run

A : 000046 X : 000003
L : 00402A PC: 005077
B : 004033 S : 000000
T : 000003
End program
```

(2) bp [address|clear]

- This command manipulates breakpoints.

- When you set a breakpoint, the “run” command will execute until the program counter comes to the breakpoint.

- There are three example usages of this command.

```
sicsim> bp 4010
```

- This command will set a breakpoint at memory address 0x4010.

- You may set multiple breakpoints.

- If the user is trying to set a breakpoint where there is already a breakpoint, do nothing.

```
sicsim> bp
```

- This command will print all breakpoints on the screen.

```
sicsim> clear
```

- This command will erase all breakpoints.

Example 1:

```
sicsim> bp 4010

[ok] create breakpoint 4010

sicsim> bp 4020

[ok] create breakpoint 4020

sicsim> bp 4030

[ok] create breakpoint 4030

sicsim> bp

breakpoints
-----------
4010
4020
4030

sicsim> bp clear

[ok] clear all breakpoints
```

```
sicsim> bp

no breakpoints set.
```

Example 2:

```
sicsim> progaddr 4000

Program starting address set to 0x4000.

sicsim> loader copy.obj

control         symbol         address          length
section         name
-----------------------------------------------------
COPY                           4000             1077
-----------------------------------------------------
                               total length     1077

sicsim> bp 4010

[ok] create breakpoint 4010

sicsim> bp 4020

[ok] create breakpoint 4020

sicsim> run

A : 000000 X : 000000
L : 00400A PC: 004010
B : 004033 S : 000000
T : 001000
Stop at checkpoint[4010]

sicsim> run

A : 454F46 X : 000000
L : 00400A PC: 004020
B : 004033 S : 000000
T : 001000
Stop at checkpoint[4020]

sicsim> run

A : 000046 X : 000003
L : 00402A PC: 005077
B : 004033 S : 000000
T : 000003
End program
```

```
sicsim> run

A : 000000 X : 000000
L : 00400A PC: 004010
B : 004033 S : 000000
T : 001000
Stop at checkpoint[4010]
```

※ If you run the program after reaching the end of program, the execution starts at the program starting address.

2.3. Notes

- Two example program codes are given to you:

(1) copy.obj
(2) proga.obj, progb.obj, progc.obj

You should be able to correctly load them onto the memory.

- For the program consisting of proga.obj, progb.obj, and progc.obj, only a few text records are shown and many lines are missing. You can just load whatever is there in the code correctly, and don't care about the missing records.

- You should be able to run the program copy.obj. You do not need to execute [proga.obj, progb.obj, progc.obj]. It will not run anyway.

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) copy.obj, proga.obj, progb.obj, progc.obj: the object programs.

3.2. Instructions on Submission

- Make a directory named “sp20161234_proj3”. 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_proj3/

  README
  document.docx
  20161234.c        ☜ This file contains the main function.
  20161234.h
  Makefile
  opcode.txt
  copy.obj
  proga.obj
  progb.obj
  progc.obj
```

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?