CS270 Programming Assignment 7
"LC-3 Parser Project"

Goals

For this assignment you will write a program that parses LC-3 assembly code. The goals of this programming assignment are:

1. To extend your C programming skills by writing a larger C program
2. To learn new C features: structures, enumerations, and file input/output
3. To solidify your understanding of LC-3 assembly code.
4. To prepare for the final C programming project: an LC-3 assembler.

The Assignment

Develop a C program that reads and parses LC-3 assembly code and stores the instructions and symbol table information in data structures. All legal directives, instructions, operands and comments must be parsed correctly, with a few minor exceptions which are shown below. The program must have the ability to regenerate and output assembly code from the data structures into a .asm file that can be assembled through the *lc3as* assembler tool and run on the LC-3 simulator. In addition, the program must have the ability to output the symbol table into a .sym file similar to the one output by the LC-3 tools. When writing the program, keep in mind that PA9 will be an extension to this assignment that will add the generation of machine code in the .obj format. The goal of PA9 is to generate working machine code that can pass through *lc3convert* and run on the LC-3 simulator.

Program Specification

This section is a specification of the program for the assignment, you will be graded on how closely you follow this specification:

PROGRAM NAME

The name of your program should be *lc3parse*, with the following command line arguments:

usage: lc3parse <input file> <output file> <symbol file>

The input file is an LC-3 assembly program similar to the programs you wrote for PA5 and PA6..We have provided source code and example assembly files to get you started. The output file is the same LC-3 assembly program with comments stripped, and white space reformatted according to our specification. The symbol table contains the labels from the program and each of their addresses. The format of these files is described in detail below.

INPUT SPECIFICATION

Your program must parse all legal LC-3 assembly code, with the following exceptions:

- .ORIG, .END, .BLKW, and .FILL are supported, .EXTERNAL and .STRINGZ are not
- Labels cannot exceed 10 characters and must be on the same line as the instruction or data
- All .BLKW and .FILL directives must be preceded by labels

DATA STRUCTURES

The program must use an array of C structs to store LC-3 instructions and data declarations. The array can be allocated statically, and you can assume that the test programs will not have more than 1024 lines. The C struct for an instruction requires (at least) the following:

Structure Field	***Data Type***
Program Address	**Unsigned Integer**
Instruction Opcode	**Enumerated Type**
Label	**Character Pointer**
Destination Register	**Enumerated Type**
First Source Register	**Enumerated Type**
Second Source Register	**Enumerated Type**
Base Register	**Enumerated Type**
Base Offset	**Signed Integer**
Immediate Value	**Signed Integer**
Trap Vector	**Unsigned Integer**
Program Counter Offset	**Unsigned Integer**
Label Reference	**Character Pointer**
Data Value	**Unsigned Integer**
Negative	**Boolean**
Zero	**Boolean**
Positive	**Boolean**
Machine Code	**Hexadecimal Value**

Here is an example of the enumeration for the opcode, which is in the lc3.h file provided:

```
typedef enum Opcode {

  // LC3 opcodes
  BR=0,  ADD=1, LD=2, ST=3, JSR_JSRR=4, AND=5, LDR=6, STR=7,
  RTI=8, NOT=9, LDI=10, STI=11, JMP_RET=12, LEA=14, TRAP=15,

  // LC3 directives
  ORIG, FILL, BLKW, HALT, END

  // Undefined
  UNDEFINED
};
```

Here is a short explanation of the fields:

- Not all fields are needed for each instruction and directive.
- The register fields contain the destination, source, and base registers, as needed.
- The offset and immediate values are sign extended from the fields in the instruction.
- The label and label references are strings which must be allocated.
- The data value is zero for .BLKW, and the value specified for .FILL directives.
- The condition codes are set only for the BR instruction.
- The program counter offset and machine code fields are not required for PA7.

The program must use an array of C structs to store the LC-3 symbol table. The array can be statically allocated, and you can assume that the test programs will not have more than 128 symbols. The C struct for a symbol table entry requires (at least) the following:

Structure Field	***Data Type***
Label Name	**Character Pointer**
Label Address	**Unsigned Integer**

The program will also need to keep track of the number of instructions (entries in instruction array), number of symbols (entries in symbol table). The numbers described in this section can be declared as globals, all other variables should be local to functions.

OUTPUT SPECIFICATION

The first line of output of the assembly program must have the .ORIG directive, followed by one line per instruction or data declaration, and ending with a .END directive. No blank lines or comment lines are allowed. Each instruction line must be formatted as follows, with the last character set to ASCII linefeed. All fields should be left justified and filled with blanks if they are less then the specified number of characters, and there should be no lower case characters in the directives, operators, or operands.

- Columns 0:9 contain the (optional) label, or spaces if no label
- Column 10 contains an ASCII space character
- Column 11 contains an ASCII period character for directives, else it contains a space
- Columns 12:16 contain the directive or opcode
- Columns 17:18 contain ASCII space characters
- Columns 19:31 contain the operands, comma delimited, no spaces between operands

The symbol table must contain all of the labels in the program, along with their hexadecimal address. Each symbol table line must be formatted as follows, with the last character set to ASCII linefeed (0x0a), generated by a \n character in C.

- Columns 0:9 contain the label
- Columns 10:11 contain ASCII space characters
- Columns 12:16 contain the four digit hexadecimal address preceded by an "x".

ERROR HANDLING

The program must handle the following error conditions by reporting the exact error message specified below, then exiting the program:

- "Duplicate Symbol", caused by more than one label of the same name.
- "Invalid Operation", caused by an invalid operation or directive.
- "Invalid Operands", caused by incorrectly specified or the wrong number of operands.
- "Out of Range", caused by an out of range immediate value or offset.
- "Invalid Reference", caused by a reference to a nonexistent label.

CODE PROVIDED

We have provided you with the following files packaged into a tar file:

http://www.cs.colostate.edu/~cs270/.Spring14/assignments/PA7/src/pa7.tar

- lc3.h – a header file with hardware definitions
- lc3parse.c,.h – the main program for the assignment
- utilities.c,.h – utility functions for the assignment
- Makefile – a makefile example to build the provided code

Note: The provided code only partially defines the structures and enumerations. You must finish the structures and write much more code to implement the entire LC-3 instruction set. Part of the goal of this exercise is to pickup the supplied code and extend the functionality. This is a very common scenario in most development situations, you rarely develop code from scratch.

OTHER SPECIFICATIONS

The program must allocate memory for labels dynamically, and free the memory before exiting.

Reminders

1. Make sure that your tar file is named pa7.tar, all lowercase.
2. Make sure that your tar file unpacks to a directory named PA7.
3. Test your tar file in another directory using $ tar xvf pa7.tar
4. Comments headers are required for each file and function.

Testing

You will perform your own testing of this program, with the test files described below or other test files that you provide. Remember that labels must be 10 characters or less according to the specification. The test files we provide include simple.asm, which should work with the supplied code, example.asm, a small test program with labels and data, instructions.asm, which tests all opcodes and directives, and errors.asm which has all possible errors (some of which are commented out):

http://www.cs.colostate.edu/~cs270/.Spring14/assignments/PA7/data/simple.asm
http://www.cs.colostate.edu/~cs270/.Spring14/assignments/PA7/data/example.asm
http://www.cs.colostate.edu/~cs270/.Spring14/assignments/PA7/data/instructions.asm
http://www.cs.colostate.edu/~cs270/.Spring14/assignments/PA7/data/errors.asm

We have also provided .out and sym files that we believe are the correct outputs:

http://www.cs.colostate.edu/~cs270/.Spring14/assignments/PA7/data/simple.out
http://www.cs.colostate.edu/~cs270/.Spring14/assignments/PA7/data/simple.sym
http://www.cs.colostate.edu/~cs270/.Spring14/assignments/PA7/data/example.out
http://www.cs.colostate.edu/~cs270/.Spring14/assignments/PA7/data/example.sym
http://www.cs.colostate.edu/~cs270/.Spring14/assignments/PA7/data/instructions.out
http://www.cs.colostate.edu/~cs270/.Spring14/assignments/PA7/data/instructions.sym

To test, for example with instructions.asm, do the following:

```
$ lc3parse instructions.asm instructions.myout instructions.mysym
```

Now compare the answer from your program against the provided answer.

```
$ diff instructions.out instructions.myout
$ diff instructions.sym instructions.mysim
```

This works for simple.asm, example.asm, and instructions.asm. To test the errors.asm file, uncomment the errors one at a time and run lc3parse. You should see the error output, but no files will be generated.

Grading Criteria

To grade the assignment, we will examine and run the program on the example.asm file (20 points), our own test file (20 points), and we will check the symbol file generated by the test file (10 points). We will then parse a set of files with errors and make sure the program handles error correctly (15 points). We will look at the organization and modularity of your program, and make sure that you have no globals except those specified (15 points). Style points will be given for comments on module and function headers, indentation, and function and variable naming (20 points).