CSCI 1730 Breakout Lab 05

Two's Complement Representation of Signed Integers

Learning Outcomes

- Understand how signed integers are represented and stored in memory using two's complement representation by converting signed integers to and from base-10 (decimal) to binary.
- Trace, design, and implement software solutions to non-trivial problems using the C programming language.
- Design and implement programs that use command line arguments.
- Design and implement programs that use basic programming and flow of control structures.
- Use a Makefile to compile and run a program.

Problem / Exercise

The purpose of this lab is to get experience and a deeper understanding of how signed integers are encoded in binary using two's complement representation. You will implement a C program that will process command line arguments in various ways as described below.

- Your program should process a flag `-decimal` followed by a signed decimal (base-10) integer x and a flag `-bits` followed by n, which represents the number of bits to encode x using two's complement representation. Your program should convert x into binary using n-bit two's complement representation as shown in the examples. The `-decimal` flag and `-bits` flag might be inputted in any order (as shown in the examples), which is common for flags and options inputted for commands on Unix/Linux based systems.
- Your program should process a flag `-binary` followed by a bit string of length n. Your program should convert the bit string to a signed decimal (base-10) integer using n-bit two's complement representation as shown in the examples.

For this assignment, you should assume command line arguments will be inputted in such a manner that all of the following is true.

- $1 \leq n \leq 64$.
- $-2^{n-1} \leq x \leq 2^{n-1} - 1$.

Before you write any code for this assignment, you should work through all examples with paper and pencil.

Examples

Your C source code should be in a file called `lab05.c`, and it should be compiled into an executable called `lab05.out` using the provided Makefile for this assignment. Your C program must look exactly like the examples below when run on the command line on odin or a vcf node, it must compile correctly, and it must run correctly with any valid sequence of inputs provided as command line arguments. You may assume that the command line arguments will be entered in a correct manner as demonstrated by the examples. Each example is a separate execution of a correct program for this assignment.

```
./lab05.out -decimal -16 -bits 12
-16 in decimal is equal to 111111110000 in binary using 12-bit two's complement representation

./lab05.out -bits 15 -decimal -119
-119 in decimal is equal to 111111110001001 in binary using 15-bit two's complement representation

./lab05.out -decimal -16 -bits 17
-16 in decimal is equal to 11111111111110000 in binary using 17-bit two's complement representation
```

```
./lab05.out -bits 8 -decimal 19
19 in decimal is equal to 00010011 in binary using 8-bit two's complement representation

./lab05.out -binary 110101
110101 in binary is equal to -11 in decimal using 6-bit two's complement representation

./lab05.out -binary 001100001
001100001 in binary is equal to 97 in decimal using 9-bit two's complement representation

./lab05.out -binary 1
1 in binary is equal to -1 in decimal using 1-bit two's complement representation

./lab05.out -binary 0
0 in binary is equal to 0 in decimal using 1-bit two's complement representation
```

1 C Program

1.1 General Requirements

1. Place the files for this assignment in a directory called `lab05`.
2. Your program's source file(s) must compile on odin or a vcf node using the provided Makefile and using the required compiler for this course (`gcc version 11.2.0`).
3. Place your C source code for this assignment in a file named `lab05.c`. See the Examples section to see what your C program should output. Compile your C program using the provided Makefile for this assignment. Do NOT modify any commands, targets, or dependencies that are associated with compiling source code in the provided Makefile for this assignment. If there are any warnings or errors from the compiler, then you should fix them before you submit this assignment to us for grading.
4. The only include directives that you are authorized to use in your source code for this assignment are listed below.

   ```
   #include <stdio.h>
   #include <string.h>
   #include <stdlib.h>
   #include <stdbool.h>
   ```

   You CANNOT use any other include directives anywhere in your source code.
5. All written and verbal instructions stated by the teaching staff (lecture instructor, lab instructor(s), etc.) must be followed for this assignment. Failure to follow instructions may result in losing points.

1.2 Coding Style Requirements

1. All functions must be commented. Comments must include a brief description of what the function does, its input(s), its output, and any assumptions associated with calling it. If your function has a prototype and an implementation separated into different parts of your source code, then you only need to put the comments for that function above its prototype (there is no need to comment both the prototype and implementation; commenting the prototype is sufficient).
2. All structs, unions, and enums must be commented.
3. All global variables and static variables must be commented.
4. All identifiers must be named well to denote their functionality. Badly named identifiers are not allowed. For example, identifiers like a, aaa, b, bbb, bbbb are bad names for identifiers.
5. Every line of source code must be indented properly and consistently.

1.3 README.txt File Requirements

Make sure to include a `README.txt` file (use a .txt file extension) that includes the following information presented in a reasonably formatted way:

- Your First and Last Name (as they appear on eLC) and your 810/811#
- Instructions on how to compile and run your program. Since we are using a Makefile for this assignment, then these instructions should pretty simple based on the provided Makefile.

2 Submission

Submit your files before the due date and due time stated on eLC. Submit your files on eLC under the Assignment named Lab 05. Submit only the following files.

1. All source files required for this lab: lab05.c
2. A README.txt file filled out correctly

Do not submit any compiled code. Also, do not submit any zipped files or directories. Do not submit a Makefile. We will use our own copy of the provided Makefile to compile your program. We only need the files mentioned above with the file extensions aforementioned.

3 Breakout Lab Attendance: 3 points

Breakout lab attendance is required for this assignment, and it is worth three points. Students must physically attend the entire period of the breakout lab section that they registered for during the week of this assignment to earn these three points for attendance. If you complete the assignment before the end of your breakout lab period during the week of this assignment, then you are still required to attend the entire breakout lab period to earn attendance points, and you may use this time for independent study for the next exam in this course.

4 Grading: 7 points

If your program does not compile on odin or a vcf node using the provided Makefile and using the required compiler for this course (`gcc version 11.2.0`), then you'll receive a grade of zero on this part of the assignment. Otherwise, your program will be graded using the criteria below.

Program runs correctly with various test cases on odin	7 points
README.txt file missing or incorrect in some way	-1 point
Not submitting to the correct Assignment on eLC	-1 point
Late penalty for submitting 0 hours to 24 hours late	-2 points
One or more compiler warnings	-2 points
Not adhering to one or more coding style requirements	-2 points
Not submitting this assignment before its late period expires	-7 points
Penalty for not following instructions (invalid I/O, etc.)	Penalty decided by grader

You must test, test, and retest your code to make sure it compiles and runs correctly on odin with any given set of valid inputs. This means that you need to create many examples on your own (that are different than the aforementioned examples) to ensure you have a correctly working program. Your program's I/O must match the examples given in the Examples section. We will only test your program with valid sets of inputs.