CS105 – Computer Systems Spring 2026

Assignment 1: Binary Lab

Due: Tuesday, February 3, 2026 at 11:59pm

The purpose of this assignment is to give you familiarity with binary values and with various operations performed on these values. You will accomplish the goal by solving a series of programming "puzzles." Even though many of the puzzles are quite artificial, you will find yourself thinking much more about bits in working your way through them.

You must work in a group of two people in solving the problems; partners will be assigned for this assignment. You should complete this assignment using pair programming, and you and your partner should submit one solution. *I strongly recommend that you and your partner brainstorm before coding!*

Getting Started

The materials for this lab are available on the course web page and on the course VM. I strongly recommend that you complete this assignment on the VM.

First, ensure that you are connected to the Pomona network or the Pomona VPN. Then `ssh` to the VM using your Pomona username (e.g., abcd1234)::

```
% ssh USERNAME@itbdcv-lnx04p.campus.pomona.edu
```

and unpack the starter code into your home directory on the VM:

```
% tar xvf /cs105/starters/binarylab.tar
```

This will cause a number of files to be unpacked in the directory. The only file you will be modifying and is `bits.c`.

Begin by opening the file in an editor and *put both your names* in the comments at the top of the `bits.c` file. Do this right away!!

The `bits.c` file contains a skeleton for each of the 8 programming puzzles. Each function heading tells you what operations are allowed. You must complete each puzzle using only *straightline* code (no loops or conditionals) and a limited number of C arithmetic and logical operators. Further, you are not allowed to use any constants longer than 1 byte. See the comments in `bits.c` for detailed rules and a discussion of the desired coding style.

Compiling the Code

We strongly suggest that you do all your work on the course VM. You can be sure that the support programs `btest` and `dlc` will work there; in the past, some students have found that these programs do not run correctly on other machines.

Check the file `README` for documentation on running the compiler dlc and the `btest` program. You will find it helpful to work through the functions one at a time, testing each one as you go. You can use the `-e` flag to instruct `dlc` to count the number of operations you use (in addition to checking for disallowed operations). Once you pass the tests with `dlc`, you can test your function with `btest`. Note: you can use the `-f` flag to instruct `btest` to test only a single function, as in `./btest -f bitNor`.

Dig more deeply into the README file for information on some helper programs.

We have given you a Makefile to ease the burden of running the compiler. You can open the file and look at if you like. Type

```
% make btest
```

to compile the program btest and

```
% ./btest
```

to test your puzzle solutions (note: btest doesn't test for compliance with coding style or operation limits). You can also type

```
% make
```

to compile everything.

The dlc Program

The dlc program, a modified version of an ANSI C compiler, will be used to check your programs for compliance with the coding style rules. The typical usage is

```
% ./dlc bits.c
```

- Type ./dlc -help for a list of command line options. The README file is also helpful.
- The dlc program runs silently unless it detects a problem.
- Do not include <stdio.h> in your bits.c file (it confuses dlc results in non-intuitive errors).
- Running with the -e switch causes dlc to print counts of the number of operators used by each function.
- ANSI C, and hence dlc, disallows // comments.
- In ANSI C, you must make all variable declarations at the beginning of a function. The following code is not accepted by dlc.

```
int mask = 0x55 + (0x55 << 8);
mask = mask + (mask << 16);
int shift = (x >> 1);
int sum = (shift & mask) + (x & mask);
```

- You may ignore the warning about a "non-includable file."

The driver.pl Program

The driver.pl program will be used to grade your assignment. You can determine your current grade yourself by running

```
% ./driver.pl
```

Evaluation

Your code will be run and tested on the course VM. Your score will be computed out of a maximum of 28 points. Each function will be evaluated separately for correctness and performance.

- **Correctness (12 points):** We will use the programs driver.pl and dlc, supplied with the laboratory materials, to evaluate your code. No points will be given for a function if dlc reports an illegal

operator or another error.

- **Performance (12 points):** We will use the programs `driver.pl` and `dlc`, supplied with the laboratory materials, to evaluate your code. No points will be given for a function if `dlc` reports an illegal operator, too many operators, or another error.
- **Style (2 points):** For this assignment, "good style" is easy to attain. It means that your files are submitted correctly, your names are present at the top of each file, that your code is understandable and consistently indented, that comments—when necessary to explain—are present and easy to read, and that there is no extraneous material.
- **Feedback (2 points):** An additional 2 points will be awarded for submitting a completed feedback file.

Hint: remember that you can run the Perl script `driver.pl` to see your current Correctness and Performance scores. It will also report the total number of operations you used.

Submission Instructions

When you have finished, submit two files, `bits.c` and `feedback.txt`, on Gradescope. As always, you can download files from the VM to your local machine by running the `scp` command from your local machine. Be sure to tag your partner as your group member and submit both files in the same submission!

Part I: The BinaryLab Puzzles

Table 1 describes a set of functions that manipulate and test sets of bits. The "Rating" field gives the difficulty rating (the number of points) for the puzzle, and the "Max Ops" field gives the maximum number of operators you are allowed to use to implement each function.

Function `bitAnd` computes the bitwise and function. You may only use the operators `~` and `|`. Function `bitXor` should duplicate the behavior of the operation `^`, using only the operations `&` and `~`.

Function `isNotEqual` compares `x` to `y` for inequality. As with all *predicate* operations, it should return `1` if the tested condition holds and `0` otherwise. Function `copyLSB` replicates a copy of the least significant bit in all 32 bits of the result. Function `condition` returns `y` if `x` is true and `z` otherwise.

Function `logicalShift` performs logical right shifts. Function `rotateLeft` rotates the bits to the left by `n`. You may assume the shift/rotation amount n satisfies $0 \leq n \leq 31$.

Name	Description	Rating	Max Ops
`bitAnd(x,y)`	`x&y` using only `~` and `|`	1	8
`bitXor(x,y)`	`^` using only `&` and `~`	1	14
`isNotEqual(x,y)`	`x != y?`	2	6
`copyLSB(x)`	Set all bits to LSB of `x`	2	5
`conditional(x,y,z)`	`x ? y : z`	3	16
`logicalShift(x,n)`	Logical right shift `x` by `n`	3	16

Table 1: Bit-Level Manipulation Functions.

Part II: Feedback

Create a file called `feedback.txt` that answers the following questions:

1. How long did each of you spend on this assignment?
2. Any comments on this assignment?

How you answer these questions **will not affect your grade**, but whether you answer them will.