Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Fall Term 2024

SYSTEMS PROGRAMMING AND COMPUTER ARCHITECTURE
Assignment 1: Manipulating Bits

Assigned on: **17th September 2025**
Due by: **30th September 2025 23:59**

Introduction

The purpose of this assignment is to become more familiar with bit-level representations and manipulations of integers. You will do this by solving a series of programming "puzzles." Many of these puzzles are quite artificial, but you will find yourself thinking much more about bits in working your way through them.

Logistics

This is an individual project. Hand-in for this assignment is going to be using git. Please see the *Hand In Instructions* section for more information on how to submit your solution file. Any clarifications and revisions to the assignment will be posted on the course Moodle page.

Handout Instructions

Download the datalab handout tar ball (`datalab-handout.tar`) from the course Moodle page.

Start by copying `datalab-handout.tar` to a directory on a Linux machine in which you plan to do your work. Then give the command (`unix>` is not part of the command)

```
unix> tar xf datalab-handout.tar
unix> cd datalab-handout
```

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

The `bits.c` file contains a skeleton for each of the 16 programming puzzles. Your assignment is to complete each function skeleton using only *straightline* code (i.e., no loops or conditionals) and a limited number of C arithmetic and logical operators. Specifically, you are *only* allowed to use the following eight operators:

```
! ˜ & ˆ | + << >>
```

A few of the functions further restrict this list. Also, you are not allowed to use any constants longer than 8 bits. See the comments in `bits.c` for detailed rules and a discussion of the desired coding style.

The Puzzles

This section describes the puzzles that you will be solving in `bits.c`.

Part I: Bit Manipulations

Name	Description	Rating	Max Ops
`bitAnd(x,y)`	`(x&y)` using only ∼ and `|`	1	8
`bitXor(x,y)`	`^` using only `&` and ∼	2	14
`isEqual(x,y)`	`x == y?`	2	5
`getByte(x,n)`	Extract byte `n` from `x`	2	6
`fitsBits(x,n)`	return 1 if `x` can be represented as an n-bit integer	2	15
`anyEvenBit(x)`	return 1 if any even-numbered bit in word set to 1	2	12
`bitCount(x)`	Count number of 1's in `x`	4	40
`bang(x)`	Compute `!x` without using `!` operator	4	12
`leastBitPos(x)`	Mark least significant 1 bit	4	6

Table 1: Bit-Level Manipulation Functions.

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 AND function. That is, when applied to arguments `x` and `y`, it returns `(x&y)`. You may only use the operators ∼ and `|`. Function `bitXor` should duplicate the behavior of the bit operation `^`, using only the operations `&` and ∼.

Function `isEqual` compares `x` to `y` for equality. As with all *predicate* operations, it should return `1` if the tested condition holds and `0` otherwise.

Function `getByte` extracts a byte from a word. The bytes within a word are ordered from 0 (least significant) to 3 (most significant). Function `fitsBits` returns 1 if the given `x` fits in an n-bit integer. Function `anyEvenBit` returns `1` if any even-numbered bit is set to 1 otherwise `0`.

Function `bitCount` returns a count of the number of 1's in the argument. Function `bang` computes logical negation without using the `!` operator. Function `leastBitPos` generates a mask consisting of a single bit marking the position of the least significant one bit in the argument. If the argument equals 0, it returns 0.

Part II: Two's Complement Arithmetic

Table 2 describes a set of functions that make use of the two's complement representation of integers.

Function `tmax` returns the largest integer.

Name	Description	Rating	Max Ops
`tmax(void)`	largest two's complement integer	1	4
`isNonNegative(x)`	`x >= 0?`	3	6
`isGreater(x,y)`	`x > y?`	3	24
`multFiveEights(x)`	`(x*5/8)`	3	12
`satMul2(x)`	multiply `x` by 2, saturating to *TMin* or *TMax* if overflow	3	20
`rotateLeft(x,n)`	rotate `x` to the left by `n`	3	25
`absVal(x)`	absolute value	4	10

Table 2: Arithmetic Functions

Function `isNonNegative` determines whether `x` is greater than or equal to 0.

Function `isGreater` determines whether `x` is greater than `y`.

Function `multFiveEights` multiplies it's argument by 5/8 rounding toward 0. This function should exactly duplicate the effect of the C expression $(x * 5/8)$.

Function `satMul2` returns `x*2` if the multiplication does not overflow. On positive overflow the function returns *TMax*, and on negative overflow it returns *TMin*.

Function `rotateLeft` circularly shifts `x` to the left by `n`.

Function `absVal` is equivalent to the expression `x<0?-x:x`, giving the absolute value of `x` for all values other than *TMin*.

Note

If you want your solution auto-graded and ranked on the "Beat the Prof" website, then please follow the instructions in the *Hand In Instructions* section and also see the *Beat the Prof* section. Your code will be compiled with GCC.

The 16 puzzles you must solve have been given a difficulty rating between 1 and 4, such that their weighted sum totals to 43. We will evaluate your functions using the `btest` program, which is described in the next section.

Regarding performance, our main concern at this point in the course is that you can get the right answer. However, we want to instill in you a sense of keeping things as short and simple as you can. Furthermore, some of the puzzles can be solved by brute force, but we want you to be more clever. Thus, for each function we've established a maximum number of operators that you are allowed to use for each function. This limit is very generous and is designed only to catch egregiously inefficient solutions. You will receive two points for each correct function that satisfies the operator limit.

Your solutions should be as clean and straightforward as possible. Your comments should be informative, but they do not need to be extensive.

Autograding your work

We have included some autograding tools in the handout directory — `btest`, `dlc`, and `driver.pl` — to help you check the correctness of your work.

- `btest`: This program checks the functional correctness of the functions in `bits.c`. To build and use it, type the following two commands:

```
unix> make
unix> ./btest
```

  Notice that you must rebuild `btest` each time you modify your `bits.c` file.

  You'll find it helpful to work through the functions one at a time, testing each one as you go. You can use the `-f` flag to instruct `btest` to test only a single function:

```
unix> ./btest -f bitAnd
```

  You can feed it specific function arguments using the option flags `-1`, `-2`, and `-3`:

```
unix> ./btest -f bitAnd -1 7 -2 0xf
```

  Check the file `README` for documentation on running the btest program.

- `dlc`: This is a modified version of an ANSI C compiler from the MIT CILK group that you can use to check for compliance with the coding rules for each puzzle. The typical usage is:

```
unix> ./dlc bits.c
```

  The program runs silently unless it detects a problem, such as an illegal operator, too many operators, or non-straightline code in the integer puzzles. Running with the `-e` switch:

```
unix> ./dlc -e bits.c
```

  causes `dlc` to print counts of the number of operators used by each function. Type `./dlc -help` for a list of command line options.

- `driver.pl`: This is a driver program that uses `btest` and `dlc` to compute the correctness and performance points for your solution. It takes no arguments:

```
unix> ./driver.pl
```

  Your instructors will use `driver.pl` to evaluate your solution.

Hand In Instructions

- Remove any extraneous print statements.
- To complete this and forthcoming assignments, it is necessary to have a basic knowledge of the shell. A good crash course can be found on [1].
- You are going to use git [2] to handin your assignments for this course. Make sure you have it installed on your computer. On Ubuntu, you can install it by typing

```
unix> sudo apt install git openssh-client
```

  Git is a complex tool and can be overwhelming for first-time users, but it is the de-facto standard version control tool today. That also means there is an abundance of guides and tutorials online (for example [3])

[1] `https://www.computervillage.org/articles/CommandLine.pdf`
[2] `https://git-scm.com`
[3] `https://rogerdudler.github.io/git-guide/`

- To setup the initial git configuration, execute the following commands

```
unix> git config --global user.name "Jane Doe"
unix> git config --global user.email "jdoe@student.ethz.ch"
```

- We have set up a repository for each student in the D-INFK gitlab [4]. You can login to the web interface using your nethz credentials. To clone the repository, you need to generate a ssh public key and upload it to gitlab. You will find detailed instructions how to do so in [5]. In a nutshell you will execute the following commands:

```
unix> ssh-keygen -o -t rsa -b 4096
< confirm with enter twice >
unix> cat ~/.ssh/id_rsa.pub
```

  Then copy and paste the output to the gitlab ssh keys [6].

- Login to gitlab, open your personal projects and you should find a repository named *spca2025-NETHZ-hand-in*. Alternatively, open the following link `https://gitlab.inf.ethz.ch/COURSE-SPCA2025/spca2025-NETHZ-hand-in`. Open the project and click on the blue clone on the top right. Copy the *Clone with ssh* URL to the clipboard. Open a terminal and *clone* the repository using the following command.

```
unix> git clone git@gitlab.inf.ethz.ch:COURSE-SPCA2025/spca2025-NETHZ-hand-in.git
```

  This will create a folder `spca2025-NETHZ-hand-in` in your current working directory. If you do this the first time, you will get a warning that you are cloning an empty repository. You can ignore this warning.

- Enter this newly created folder and create a new directory called `assignment1` inside your directory.

```
unix> cd spca2025*
unix> mkdir assignment1
```

- Copy your solution file (`bits.c`) into the newly created assignment folder and then type:

```
unix> git add bits.c
unix> git commit -m assignment1
unix> git push
```

  The first command will add your file to the git staging area (a place to accumulate changes which will become a commit). `commit` will create a commit out of the staging area. So far nothing has been transferred to the gitlab server. This is where the last command comes in. It will push all new commits to the server. You can check that everything has worked by logging into gitlab and checking your bits.c there.

- You can repeat the `add-commit-push` process as many times as you want.

[4] `https://gitlab.inf.ethz.ch`
[5] `https://docs.gitlab.com/ee/gitlab-basics/create-your-ssh-keys.html`
[6] `https://gitlab.inf.ethz.ch/profile/keys`

Advice

You are welcome to do your code development using any system or compiler you choose. Just make sure that the version you turn in compiles and runs correctly with the `dlc` and `btest` programs. If it does not compile, we can not grade it. All required tools to compile the programs should be installed on the CAB H56 and H57 lab machines (i.e. other ETH lab machines, such as in HG, are not supported). If you want to build it on your own machine, make sure to install gcc (e.g. on Ubuntu: `sudo apt install build-essential flex bison`)

- Don't include the `<stdio.h>` header file in your `bits.c` file, as it confuses `dlc` and results in some non-intuitive error messages. You will still be able to use `printf` in your `bits.c` file for debugging without including the `<stdio.h>` header, although `gcc` will print a warning that you can ignore.
- The `dlc` program enforces a stricter form of C than the standard. In particular, any declaration must appear in a block (what you enclose in curly braces) before any statement that is not a declaration. For example, it will complain about the following code:

```
int foo(int x)
{
  int a = x;
  a *= 3;     /* Statement that is not a declaration */
  int b = a;  /* ERROR: Declaration not allowed here */
}
```

Beat the Prof

You can check-in solutions to your git repository as often as you like. Solutions are going to be graded periodically and rankings are going to be listed on the course page using your nethz name. In this assignment, you are going to compete against each other and the professor. The goal is to solve each data puzzle using the fewest number of operators. Entries are sorted by score, defined as (total instructor operations - total student operations). Students who match or beat the professor's operator count for each puzzle will be winners.

The grader website is http://spca.ethz.ch and is only accessible from the internal network of ETHZ. For remote access, use the ETH VPN service.