Data Lab: Manipulating Bits

Instructors:

Sungyong Ahn

Introduction

- The purpose of this assignment is to become more familiar with bit-level representations of common patterns, integers, and floating-point numbers.
- Solve a series of programming "puzzles."
- Please Read the Writup.

Handout Instructions

- **Lab materials are contained in a Unix tar file called *datalab-handout.tar***
  - You can download from **PLATO**
- **You have to extract the tar file in Linux**
  - `$> tar xvf datalab-handout.tar`
- **The only file you will be modifying and handing in is *bits.c*.**
  - 15 programming puzzles
  - You have to follow a strict set of coding rules
    - No loops or conditionals
    - Limited number of C arithmetic and logical operators
    - you are *only* allowed to use the following eight operators
      - ! ˜ & ˆ | + << >>
- WARNING: Do not let the Windows WinZip program open up your `.tar` file

The Puzzles

Bit Manipulations

Name	Description	Rating	Max Ops
`bitOr(int x, int y)`	`x|y` using only `~` and `&`	1	8
`isEqual(int x, int y)`	return 1 if `x==y`, and 0 otherwise	2	5
`anyEvenBit(int x)`	return 1 if any even-numbered bit in word set to 1	2	12
`allEvenBits(int x)`	return 1 if all even-numbered bits in word set t 1	2	12
`rotateLeft(int x, int n)`	Rotate `x` to the left by `n`	3	25
`copyLSB(int x)`	set all bits of result to least significant bit of `x`	2	5

Two's Complement Arithmetic

Name	Description	Rating	Max Ops
`isTmax(int x)`	returns 1 if `x` is `Tmax`	1	10
`logicalNeg(int x)`	implement the `!` operator	4	12
`subOK(int x, int y)`	Determine if can compute `x-y` without overflow	3	20
`isLessOrEqual(int x, int y)`	if `x <= y` then return 1, else return 0	3	24
`satMul3(int x)`	multiplies by 3, saturating to `Tmin` or `Tmax` if overflow	3	25
`tc2sm(int x)`	Convert from two's complement to sign-magnitude	4	15

부산대학교 PUSAN NATIONAL UNIVERSITY

The Puzzles: INTEGER CODING RULES

- **You are expressly forbidden to:**
  1. Use any control constructs such as if, do, while, for, switch, etc.
  2. Define or use any macros.
  3. Define any additional functions in this file.
  4. Call any functions.
  5. Use any other operations, such as &&, ||, -, or ?:
  6. Use any form of casting.
  7. Use any data type other than int. This implies that you cannot use arrays, structs, or unions.
- **You may assume that your machine:**
  1. Uses 2s complement, 32-bit representations of integers.
  2. Performs right shifts arithmetically.
  3. Has unpredictable behavior when shifting an integer by more than the word size.
- **YOU MUST CAREFULLY READ THE CODING RULES DESCRIBED in bits.c**

The Puzzles

- **Floating-Point Operations**
  - You are allowed to use standard control structures (conditionals, loops)
  - You may not use any floating point data types, operations, or constants

Name	Description	Rating	Max Ops
`float_abs(unsigned uf)`	Return bit-level equivalent of absolute value of `f`	2	10
`float_neg(unsigned uf)`	Return bit-level equivalent of expression `-f`	2	10
`float_twice(unsigned uf)`	Return bit-level equivalent of expression `2*f`	4	30

The Puzzles: FLOATING POINT CODING RULES

- **You are expressly forbidden to:**
  1. Define or use any macros.
  2. Define any additional functions in this file.
  3. Call any functions.
  4. Use any form of casting.
  5. Use any data type other than int or unsigned. This means that you cannot use arrays, structs, or unions.
  6. Use any floating point data types, operations, or constants.

- **YOU MUST CAREFULLY READ THE CODING RULES DESCRIBED in bits.c**

The Example Puzzles

```
/*
 * bitXor - x^y using only ~ and &
 *   Example: bitXor(4, 5) = 1
 *   Legal ops: ~ &
 *   Max ops: 14
 *   Rating: 1
 */
int bitXor(int x, int y) {
  return 2;
}
```

The Example Puzzles

```
/*
 * bitXor - x^y using only ~ and &
 *   Example: bitXor(4, 5) = 1
 *   Legal ops: ~ &
 *   Max ops: 14
 *   Rating: 1
 */
int bitXor(int x, int y) {
  return x^y;
}

$> make
$> ./driver.pl
```

The Example Puzzles

```
1. Running './dlc -z' to identify coding rules violations.
/usr/include/stdc-predef.h:1: Warning: Non-includable file <command-line> included from includable file /usr/include/stdc-
predef.h.
dlc:bits.c:143:bitXor: Illegal operator (^)
dlc:bits.c:144:bitXor: Zapping function body!

Compilation Successful (1 warning)

2. Running './bddcheck/check.pl -g' to determine correctness score.

3. Running './dlc -Z' to identify operator count violations.
/usr/include/stdc-predef.h:1: Warning: Non-includable file <command-line> included from includable file /usr/include/stdc-
predef.h.
dlc:save-bits.c:143:bitXor: Illegal operator (^)
dlc:save-bits.c:144:bitXor: Zapping function body!

Compilation Successful (1 warning)

4. Running './bddcheck/check.pl -g -r 2' to determine performance score.

5. Running './dlc -e' to get operator count of each function.

Correctness Results     Perf Results
Points  Rating  Errors  Points  Ops     Puzzle
0       1       1       0       0       bitXor
0       1       1       0       0       tmin
0       2       1       0       0       isTmax
0       2       1       0       0       allOddBits
0       2       1       0       0       negate
0       3       1       0       0       isAsciiDigit
0       3       1       0       0       conditional
0       3       1       0       0       isLessOrEqual
0       4       1       0       0       logicalNeg
0       4       1       0       0       howManyBits
0       4       1       0       0       float_twice
0       4       1       0       0       float_i2f
0       4       1       0       0       float_f2i

Score = 0/63 [0/37 Corr + 0/26 Perf] (0 total operators)
```

The Example Puzzles

```
/*
 * bitXor - x^y using only ~ and &
 *   Example: bitXor(4, 5) = 1
 *   Legal ops: ~ &
 *   Max ops: 14
 *   Rating: 1
 */
int bitXor(int x, int y) {
  int x_and_y = x&y;
  int x_or_y = ~(~x & ~y);
  return x_or_y & ~x_and_y;
}

$> make
$> ./driver.pl
```

The Example Puzzles

```
1. Running './dlc -z' to identify coding rules violations.
/usr/include/stdc-predef.h:1: Warning: Non-includable file <command-line> included from includable file /usr/include/stdc-
predef.h.

Compilation Successful (1 warning)

2. Running './bddcheck/check.pl -g' to determine correctness score.

3. Running './dlc -Z' to identify operator count violations.
/usr/include/stdc-predef.h:1: Warning: Non-includable file <command-line> included from includable file /usr/include/stdc-
predef.h.

Compilation Successful (1 warning)

4. Running './bddcheck/check.pl -g -r 2' to determine performance score.

5. Running './dlc -e' to get operator count of each function.

Correctness Results     Perf Results
Points  Rating  Errors  Points  Ops     Puzzle
1       1       0       2       8       bitXor
0       1       1       0       0       tmin
0       2       1       0       0       isTmax
0       2       1       0       0       allOddBits
0       2       1       0       0       negate
0       3       1       0       0       isAsciiDigit
0       3       1       0       0       conditional
0       3       1       0       0       isLessOrEqual
0       4       1       0       0       logicalNeg
0       4       1       0       0       howManyBits
0       4       1       0       0       float_twice
0       4       1       0       0       float_i2f
0       4       1       0       0       float_f2i

Score = 3/63 [1/37 Corr + 2/26 Perf] (8 total operators)
```

Helper Programs

- **The *ishow* and *fshow* programs to help you decipher integer and floating point representations respectively.**
  - $> make

```
$> ./ishow 15213
Hex = 0x00003b6d,          Signed = 15213, Unsigned = 15213

$> ./fshow 15213
Floating point value 2.131795354e-41
Bit Representation 0x00003b6d, sign = 0, exponent =
0x00, fraction = 0x003b6d
Denormalized.  +0.0018135309 X 2^(-126)
```

Evaluation

- **Maximum of 68 points based on the following distribution:**
  - **38** Correctness of code
  - **30** Performance of code, based on number of operators used in each function.

- **Your handin was properly autograded.**

Autograding your work

- Handy autograding tools
  - `Btest`
    - Checks the functional correctness of the functions in bits.c
  - `dlc`
    - Check for compliance with the coding rules for each puzzle
  - `driver.pl`
    - Compute the correctness and performance points for your solution
    - `$> ./driver.pl`

Test and debug your function

- **Step 1.** Test and debug one function at a time using btest
  - `./btest -f bitXor -1 23 -2 0xabcd`
- **Step 2.** Use `btest -f` to check the correctness of your function against a large number of different input values:
  - `./btest -f isLess`
- **Step 3.** Use `dlc` to check that you've conformed to the coding rules
  - `./dlc bits.c`
- **Step 4.** Repeat Steps 1–3 for each function. At any point in time, you can compute the total number of correctness and performance points you've earned by running the driver program:
  - `./driver.pl`

Submission

- **Due to 4/7 (수) 23:59**
  - 하루 딜레이 시 만점기준 25% 감점

- **학번_bits.c, 학번_selfgrade.jpg PLATO에 제출**
  - 학번_selfgrade.jpg 는 최종 `driver.pl` 출력 캡처 이미지

- **Please Read the Writup.**