System Programming

Coding Project #1. Data Representation

1. Description

Bitwise Operation

Bitwise operation operates on the values(bit pattern) at the level of individual bits. It has diverse operators(! ~ | + >> << ...) which is simple action supported by processor. In this assignment, you will implement simple operations with using legal bitwise operators in each function.

Consider the case bitAnd operation.

bitAnd(int x, int y) : And operation of x and y at bit level.

Ex. bitAnd(12, 25) = (0000 1100) and (0001 1001) = 0000 1000 = 8

(In this example, only 8bits is expressed.)

bitAnd operation can be implemented as ~(~x | ~y).

2. What to do and submit

The only file you will be modifying and turning in is bits.c. The file contains a skeleton for each of the 18 programming puzzles. Your assignment is to complete each function skeleton using only straight line code for the integer puzzles (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:

! ~ & ^ | + << >>

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

You should implement several functions:

Bit Manipulations

Below list describes a set of functions that manipulate and test sets of bits. The "rating" field gives the number of points for the puzzle, and the "max operators" field gives the maximum number of operators you are allowed to use to implement each function. See

the comments in bits.c for more details on the desired behavior of the functions.

bitXor(int x, int y) : x^y using ~ and &
    legal operators : ~ &
    max operators : 14
    rating : 1

tmax(void) : return maximum two's complement integer
    legal operators : ! ~ & ^ | + << >>
    max operators : 4
    rating : 1

isNotEqual(int x, int y) : return 0 if x == y, and 1 otherwise
    legal operators : ! ~ & ^ | + << >>
    max operators : 6
    rating : 2

replaceByte(int x, int n, int c) : Replace byte n in x with c
    legal operators : ! ~ & ^ | + << >>
    max operators : 10
    rating : 3

fitsBits(int x, int n) : return 1 if x can be represented as an n-bit, two's complement integer
    legal operators : ! ~ & ^ | + << >>
    max operators : 15
    rating : 2

rotateLeft(int x, int n) : Rotate x to the left by n
    legal operators : ~ & ^ | + << >> !
    max operators : 25
    rating : 3

isPower2(int x) : return 1 if x is a power of 2, and 0 otherwise
    legal operators : ! ~ & ^ | + << >>
    max operators : 20
    rating : 6

rempwr2(int x, int n) : Compute x%(2^n), for 0 <= n <= 30

legal operators : ! ~ & ^ | + << >>

max operators : 20

rating : 6

conditional(int x, int y, int z) : return y if x is not 0, else z same as x ? y : z

legal operators : ! ~ & ^ | + << >>

max operators : 16

rating : 3

bitParity(int x) : returns 1 if x contains an odd number of 0's, else 0

legal operators : ! ~ & ^ | + << >>

max operators : 20

rating : 4

greatestBitPos(int x) : return a mask that marks the position of the most significant 1 bit If x == 0, return 0

legal operators : ! ~ & ^ | + << >>

max operators : 70

rating : 4

logicalNeg(int x) : implement the ! operator, using all of the legal operators except !

legal operators : ~ & ^ | + << >>

max operators : 12

rating : 5

bitAnd(int x, int y) : x&y using ~ and |

legal operators : ~ |

max operators : 8

rating : 1

logical_OR(int x, int y) : implement the || operator, using legal operators

legal operators : ~ & ^ | ! >> <<

max operators : 10

rating : 1

concatenate(int x, int y) : concate two input x, y
(ex. concate(2,3) = concate(0000 0010, 0000 0011) = 0000 0010 0000 0011 = 515(in decimal))

legal operators : ~ & | >> << +
max operators : 10
rating : 2

isMult4(int x) : return 1 if x is multiple of 4, else 0

legal operators : ! ~ & ^ | + << >>
max operators : 8
rating : 4

Floating-point operations

For this part of the assignment, you will implement some common single-precision floating-point operations. In this section, you are allowed to use standard control structures (conditionals, loops), and you may use both int and unsigned data types, including arbitrary unsigned and integer constants. You may not use any unions, structs, or arrays. Most significantly, you may not use any floating point data types, operations, or constants. Instead, any floating-point operand will be passed to the function as having type unsigned, and any returned floating-point value will be of type unsigned. Your code should perform the bit manipulations that implement the specified floating point operations.

Below list describes a set of functions that operate on the bit-level representations of floating-point numbers. Refer to the comments in bits.c and the reference versions in tests.c for more information.

unsigned float_neg(unsigned uf): return -uf

legal operators : Any integer/unsigned operations incl. ||, &&. also if, while
max operators : 10
rating : 2

unsigned float_twice(unsigned uf) : return 2*uf

legal operators : Any integer/unsigned operations incl. ||, &&. also if, while

max operators : 30
rating : 4

Functions float_neg and float_twice must handle the full range of possible argument values, including not-a-number (NaN) and infinity. The IEEE standard does not specify precisely how to handle NaN's, and the IA32 behavior is a bit obscure. We will follow a convention that for any function returning a NaN value, it will return the one with bit representation 0x7FC00000.

The included program fshow helps you understand the structure of floating point numbers. You can use fshow to see what an arbitrary pattern represents as a floating-point number:

```
$./fshow 2080374784
    Floating point value 2.658455992e+36
    Bit Representation 0x7c000000, sign = 0, exponent = f8, fraction = 000000
    Normalized. 1.0000000000 X 2^(121)
```

Implementation of functions (90 points)

Implement all functions in bits.c with legal bitwise operators in each function. You don't have to modify other files. In this assignment you can check your scores with btest. **Your goal is to pass all tests in btest without any errors and get the best performance. There also exists limitation of the number of operators and illegal operators.** Let me explain all files in assignment.

Makefile : Makes btest, fshow, and ishow
bits.c : The file you will be modifying and handing in
bits.h : Header file
btest.c : The main btest program
btest.h : Used to build btest
decl.c : Used to build btest
tests.c : Used to build btest
dlc* : Rule checking compiler library (data lab compiler)
Driverhdrs.pm, Driverlib.pm, driver.pl* : Driver program that uses btest, dlc to autograde bits.c
fshow.c : Utility for examining floating-point representations

ishow.c : Utility for examining integer representations

** ./fshow, ./ishow is used for check the values as floating point, int(unsigned, hex)

Report (10 points)

Write 1 ~ 2 page report explaining each function in bits.c. In this assignment, **please explain your code elaborately**. Write simple algorithm how you implement your functions in bits.c. Please do not exceed more than 2 pages, there will be penalty.

Submit

You must implement bits.c using the **C language (not C++)**. All you need to do is that implementing the function in bits.c file.

Please upload your codes(including bits.c) and update the code as you make process. Please bring your report(hard copy) to the practical class to submit to the tutor.

Code due date: 2018/10/5 11:59 PM

Report due date: 2018/10/5 1:00 PM before the start of practical class

Please do not upload the source code after due, there will be 50% mark penalty per one day delay.

How to use programs

- Modifying bits.c and checking it for compliance with dlc

```
$./dlc bits.c
```

It will check the function in bits.c whether it uses legal operator or not

It will return nothing if there are no problems with your code. Otherwise it prints messages that flag any problems.

Note: dlc will always output the following warning, which can be ignored:

/usr/include/stdc-predef.h:1: Warning: Non-includable file <command-line> included from includable file /usr/include/stdc-predef.h.

```
$./dlc –e bits.c
```

It will print counts of the number of operators used by each function.

- **Test with ./btest**

```
$./btest
```

Test all the functions for correctness and print out error messages

```
$./btest –f func
```

Test function func for correctness

****./btest do not check your code for coding guidelines.**

(ex. Check illegal operators)

- **Test your code & performance**

```
$./driver.pl
```

Test your all functions in bits.c in aspect of correctness & performance

***This assignment referenced*

1. CMU system programming datalab assignment