PMR3309 - 2019
Data Lab: Manipulando Bits
Entregar até: meia-noite de 28 de agosto

Adaptado do Material de Bryant e O'Hallaron para o curso 15-213 da CMU

1 Introdução

O propósito deste exercício é familarizar-se com representações binárias de números inteiros e de ponto flutuantes. Para tanto serão resolvidos uma série de "puzzles". A maior parte deles impõe restrições artificais com – admitidamente – pouca aplicabilidade prática, mas elas irão expor aspectos importantes da representação binária.

2 Entrega

Este é um projeto individual. As entregas devem ser feitas eletronicamente pelo moodle da disciplina.

3 Instruções

Copie o arquivo `datalab.tar` em um diretório em uma máquina linux na qual você pretende trabalhar. O arquivo pode ser descompactado com o comando

```
unix> tar xvf datalab-handout.tar.
```

Atenção: Para este exercício, o *único* arquivo a ser modificado e entregue é o `bits.c`.

O arquivo `bits.c` contém um esqueleto para cada um dos 15 puzzles de programação. Sua tarefa é completar cada função. Para os inteiros, você deve usar apenas código *direto* (ou seja, não use desvio de fluxo como loops ou condições) e um conjunto *limitado* de operadores aritiméticos e lógicos em C. Especificamente, você pode usar apenas os seguintes operadores:

```
! ~ & ^ | + << >>
```

Atenção: Algumas funções impõe mais restrições a esta lista. Além disso, você não pode usar nenhuma constante maior do que 8 bits (de `0x00` a `0xFF`). Veja os comentários em `bits.c` para regras mais detalhadas.

4 Os Puzzles

4.1 Manipulação de bits

A tabela 1 descreve funções que manipulam e testam conjuntos de bits. A "Dificuldade" dá a quantidade de pontos que o puzzle vale e o "Max ops" dá o número máximo de operadores que você pode usar para implementar cada função. Veja comentários em `bits.c` para mais detalhes sobre o comportamento desejado de cada função. Você pode também verificar as funções de teste em `tests.c` (note que estas funções *violam* as restrições do problema!).

Nome	Descrição	Dificuldade	Max Ops
`bitAnd(x,y)`	`x & y` usando apenas `|` e `~`	1	8
`getByte(x,n)`	Recupera o byte `n` de `x`.	2	6
`logicalShift(x,n)`	Deslocamento à direita *lógico*.	3	20
`bitCount(x)`	Conta a quantidade de 1's em `x`.	4	40
`bang(x)`	Encontra `!n` sem o operador `!`.	4	12

Table 1: Funções de Manipulação de bits.

4.2 Aritimética de complemento de dois

A tabela 2 descreve funções que exploram a representação em complemento de dois de inteiros. Novamente, veja comentários em `bits.c` e as implementações de referência em `tests.c`.

Nome	Descrição	Dificuldade	Max Ops
`tmin()`	Menor inteiro negativo em complemento de dois	1	4
`fitsBits(x,n)`	O número `x` pode ser representado com `n` bits?	2	15
`divpwr2(x,n)`	Calcula $x/2^n$	2	15
`negate(x)`	$-x$ sem o operador $-$	2	5
`isPositive(x)`	`x > 0`?	3	8
`isLessOrEqual(x,y)`	`x <= y`?	3	24
`ilog2(x)`	Calcula $\lfloor \log_2(x) \rfloor$	4	90

Table 2: Funções Aritiméticas

4.3 Operações com Ponto Flutuante

Nesta parte você implementará algumas operações comuns de ponto flutuante. Aqui você poderá empregar estruturas de controle convencionais (condições, loops) e pode usar variáveis do tipo `int` e `unsigned`, incluindo constantes arbitrárias. Você não pode usar nenhum tipo com unions, structs, ou vetores. Mais importante, você *não* pode usar variáveis de ponto flutuante, operações ou constantes. Todos os argumentos

em ponto flutuante serão passados à função como `unsigned` e quaisquer valores de retorno em ponto flutuante serão retornados como `unsigned`. Seu código deve fazer as manipulações de bits que implementam as operações específicas de ponto flutuante.

A tabela 3 descreve o conjunto de funções que operam na representação de bits de números de ponto flutuante. Veja os comentários em `bits.c` e as implementações de referência em `tests.c` para mais informação.

Nome	Descrição	Dificuldade	Max Ops
`float_neg(uf)`	Calcula `-f`	2	10
`float_i2f(x)`	Calcula `(float) x`	4	30
`float_twice(uf)`	Calcula `2*f`	4	30

Table 3: Funções de Ponto Flutuante. O valor `f` é o número de ponto flutuante que tem a mesma representação em bits do inteiro `uf`.

As funções `float_neg` e `float_twice` devem ser capazes de tratar todos os possíveis values de pongo flutuante IEEE, incluindo *not-a-number* (NaN) e infinito. O padrão IEEE não especifica precisamente como tratar um NaN (e os processadores IA32 são "obscuros" neste aspecto). Para este exercício será seguida a convenção de que uma função que precise retornar um NaN deve usar a representação `0x7FC00000`.

O programa `fshow` ajudará a entender a representação de números com ponto flutuante. Para compilar `fshow` use o comando

```
unix> make
```

Você pode usar `fshow` para ver o que um padrão arbitrário de bits representa como número de ponto flutuante:

```
unix> ./fshow 2080374784

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

Você pode também passar a `fshow` valores hexadecimal e fracionários e ele exibirá a sua estrutura de bits.

5 Nota

A nota será baseada em um score de no máximo 76 pontos baseados na seguinte distribuição:

41 Pontos de correção: Cada um dos 15 puzzles tem uma nota de dificuldade de 1 a 4. Você terá nota cheia para um puzzle resolvido corretamente e zero caso contrário.

30 Pontos de desempenho: Cada puzzle tem um limite máximo de operadores. Se você resolver o puzzle dentro do limite, ganha dois pontos.

5 Pontos de estilo: 5 pontos para soluções claras, concisas e bem documentadas.