Insper

Sistemas de Hardware e Software, 2019-2

Datalab: manipulação de bits

Entrega: 28/08

Introdução

O propósito desta atividade é torná-los mais proficientes na representação em nível de bits de números inteiros e de ponto-flutuante. Você irá resolver uma série de "puzzles" de programação envolvendo a manipulação direta dos bits de valores inteiros e de ponto-flutuante. Embora estes puzzles sejam obviamente artificiais, ao trabalhar nestes problemas você ganhará um entendimento mais profundo da representação de dados num computador real, e também irá praticar seu raciocínio algorítmico.

Preliminar

Este lab usa ferramentas para compilação de código 32 bits. Isto significa que é necessário instalar o pacote `gcc-multilib`.

Entrega

Você deverá submeter o arquivo **bits.c** via Blackboard até a data de entrega (13/03). A atividade é **individual**: trabalho em equipe não será permitido.

Instruções

Copie o arquivo **datalab.tar** para sua máquina Linux e use o seguinte comando para descompactar este arquivo:

```
$ tar xvf datalab-handout.tar
```

Um diretório datalab-handout será criado, contendo vários arquivos. **VOCÊ DEVERÁ MODIFICAR E ENTREGAR APENAS O ARQUIVO bits.c!**

O arquivo bits.c contém o esqueleto de cada um dos puzzles de programação. Sua tarefa é completar cada função respeitando algumas restrições:

- apenas código linear (ou seja, sem if ou loops) nos puzzles de números inteiros,
- usar apenas um número limitado de operadores aritméticos e lógicos da linguagem C. Especificamente, você deve usar somente os seguintes operadores: ! ~ & ^ | + << >>
- Em alguns casos haverá restrições ainda mais severas. Além disso, você não pode usar nenhuma constante maior que 8 bits (ou seja, só pode usar constantes 0x00 a 0xff). Os comentários de código no próprio arquivo bits.c explicam detalhadamente as regras e discutem o estilo de programação desejado.

Os puzzles

Cada exercício do arquivo bits.c tem uma dificuldade diferente. Além disto, se você fizer o trabalho usando um número pequeno de operações ganhará também pontos de performance.

Importante: Falhar em resolver algum puzzle de dificuldade 1 ou 2 implica em conceito D. Caso contrário, o conceito será atribuído conforme a porcentagem de pontos obtidos.

Utilitários

Alguns utilitários estão incluídos neste exercício:

btest

Verifica a corretude funcional de bits.c. Para compilar e usar esta ferramenta, digite os comandos:

$ make

$ ./btest

A cada vez que você modificar bits.c, recompile e rode btest. Para testar apenas uma função por vez, use a flag -f

$ ./btest -f bitAnd

Você pode testar seus próprios argumentos de entrada usando as flags -1, -2 e -3

$ ./btest -f bitAnd -1 7 -2 0xf

dlc

Uma versão modificada de um compilador C do pessoal do MIT CILK que pode ser usado para verificar aderência às regras do exercício. Exemplo de uso:

$./dlc bits.c

Se tudo estiver correto o programa rodará silenciosamente. Caso haja algum problema (operador ilegal, excesso de operadores, código não-linear, etc.) o usuário será notificado. A opção -e imprime o número de operadores usado.

$./dlc -e bits.c

driver.pl

Um script Perl que computa os pontos de correção e performance da sua solução. Este script vai automaticamente chamar os programas btest e dlc. Para usar, digite apenas

$ ./driver.pl

O professor irá usar driver.pl para computar os pontos de sua solução!

Dicas

Para debugar voce pode usar printf em bits.c, mas não inclua `<stdio.h>` (pois o dlc não entende `<stdio.h>` direito). O compilador gcc vai imprimir um warning sobre o uso de printf sem o header correspondente, mas você pode ignorar esse warning.

O dlc requer que as declarações de tipo estejam sempre no início do bloco, antes de qualquer statement que não seja declaração. Por exemplo, dlc vai reclamar do código a seguir:

```
1 int foo(int x) {
2   int a = x;
3   a *= 3;    /* Primeiro statement que não é declaração. */
4   int b = a; /* ERRO: dlc não permite declaração aqui. */
5 }
```