‹Introdução a Bitwise›

Kaio Christaldo
Fabricio Matsunaga

Bitwise

Algorithm ‹bitwise›

- Operações bitwise (bit a bit) são estremamento importante para que código seja rápido e eficiente.
- Elas operam diretamente na representação binária dos números, sendo executadas em um único ciclo de clock do processador.
- Usadas para resolver problemas que envolvem conjuntos, estados e otimizações de baixo nível
- Operadores: &, |, ^, ~, >> e <<

Algorithm ‹bitwise›

- Velocidade Extrema: São ordens de magnitude mais rápidas que operações aritméticas tradicionais. x * 2 é mais lento que x << 1.
- Eficiência de Memória: Você pode armazenar um conjunto de até 64 flags (verdadeiro/falso) em uma única variável long long, em vez de usar um bool array[64].
- Representação de Conjuntos e Estados: Esta é a principal aplicação. Um único inteiro pode representar qualquer subconjunto de um conjunto de elementos. A isso chamamos de Bitmask.

Algorithm ‹bitwise›

Como funciona?

- Quando temos duas veriaveis inteiras a, b
- Transforma as variaveis em binário
  - Ex: a = 10 e b = 13
  - a = 1010
  - b = 1101
- Depois executamos a operação logica com os valores binários
- Depois transformamos o resultado em inteiro.

Algorithm ‹bitwise›

Conversão decimal - binário:

Existem alguns métodos para converter um número decimal (base 10) para binário (base 2). O método mais comum e fácil de aprender é o das divisões sucessivas por 2;

25 % 2 = 1
12 % 2 = 0
6 % 2 = 0
3 % 2 = 1
1 % 2 = 1

pegando os resultados de baixo para cima, 25 em binário é 11001

Algorithm ‹bitwise›

Conversão binário - decimal:

A conversão de binário (base 2) para decimal (base 10) é ainda mais direta. O método baseia-se na soma das potências de 2 .

1. Número Binário: 1 1 0 0 1
2. Potências de 2 (da direita para a esquerda):
    - 2^4 2^3 2^2 2^1 2^0
    - 16 8 4 2 1
3. Multiplique e alguns:
    - = (1 * 16) + (1 * 8) + (0 * 4) + (0 * 2) + (1 * 1)
    - = 16 + 8 + 0 + 0 + 1
    - = 25

Portanto, o número binário 11001 é igual a 25 em decimal.

Operador AND - &

Algorithm ‹bitwise›

Primeiro, convertemos o 13 e o 7:

- 13 é 1101 em binário.
- 7 é 111 em binário.

Alinhar e Fazer a "Continha"

- 1 & 1 => 1
- 0 & 1 => 0
- 1 & 1 => 1
- 1 & 0 => 0

O resultado binário da operação é 0 1 0 1: convertendo 5

Entrada para	Entrada B	Resultado (A & B)
1	1	**1**
1	0	0
0	1	0
0	0	0

Operador OR - |

Algorithm ‹bitwise›

Primeiro, convertemos o 13 e o 7:

- 13 é 1101 em binário.
- 7 é 111 em binário.

Alinhar e Fazer a "Continha"

- 1 | 1 => 1
- 1 | 1 => 1
- 0 | 1 => 1
- 1 | 0 => 1

O resultado binário da operação é 1 1 1 1: convertendo 15

Entrada para	Entrada B	Resultado (A | B)
1	1	**1**
1	0	**1**
0	1	**1**
0	0	**0**

Operador XOR ^

Algorithm ‹bitwise›

Primeiro, convertemos o 13 e o 7:

- 13 é 1101 em binário.
- 7 é 111 em binário.

Alinhar e Fazer a "Continha"

- 1 | 1 => 0
- 0 | 1 => 1
- 1 | 1 => 0
- 1 | 0 => 1

O resultado binário da operação é 1 0 1 0: convertendo 10

Entrada para	Entrada B	Resultado (A^B)
1	1	**0**
1	0	**1**
0	1	**1**
0	0	**0**

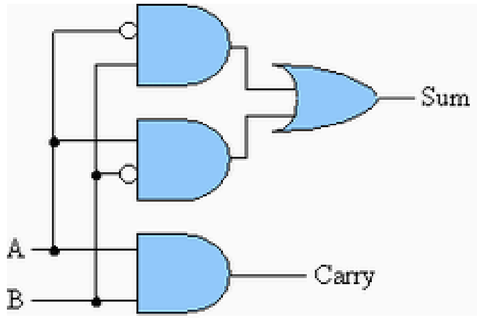
Apresentação Problema Motivador

beecrowd | 1026

Carrega ou não Carrega?

Por Monirul Hasan Tomal, SEU Bangladesh

Timelimit: 2



6+9=15 parece ok. Mas como pode estar certo 4+6=2?

Veja só. Mofiz trabalhou duro durante seu curso de Eletrônica Digital, mas quando lhe foi solicitado que implementasse um somador de 32 bits como exame no laboratório, ele acabou fazendo algum erro na parte de projeto. Depois de vasculhar seu projeto por uma hora e meia, ele encontrou seu erro. Ele estava fazendo soma de bits, mas seu carregador de bit (carry) sempre apresentava como saída o valor zero. Portanto,

```
4  = 00000000 00000000 00000000 00000100
+6 = 00000000 00000000 00000000 00000110
----------------------------------------
2  = 00000000 00000000 00000000 00000010
```

Claro que já é uma boa coisa ele finalmente ter encontrado o seu erro, mas isso foi muito tarde. Considerando seu esforço durante o curso, o instrutor deu a ele mais uma chance: Mofiz teria que escrever um programa eficiente que pegaria 2 valores decimais de 32 bits sem sinal como entrada e deveria produzir um número de 32 bits sem sinal como saída, ou seja, somando do mesmo modo como o circuito faz.

Entrada

Em cada linha de entrada haverá um par de inteiros separado por um único espaço. A entrada termina com EOF.

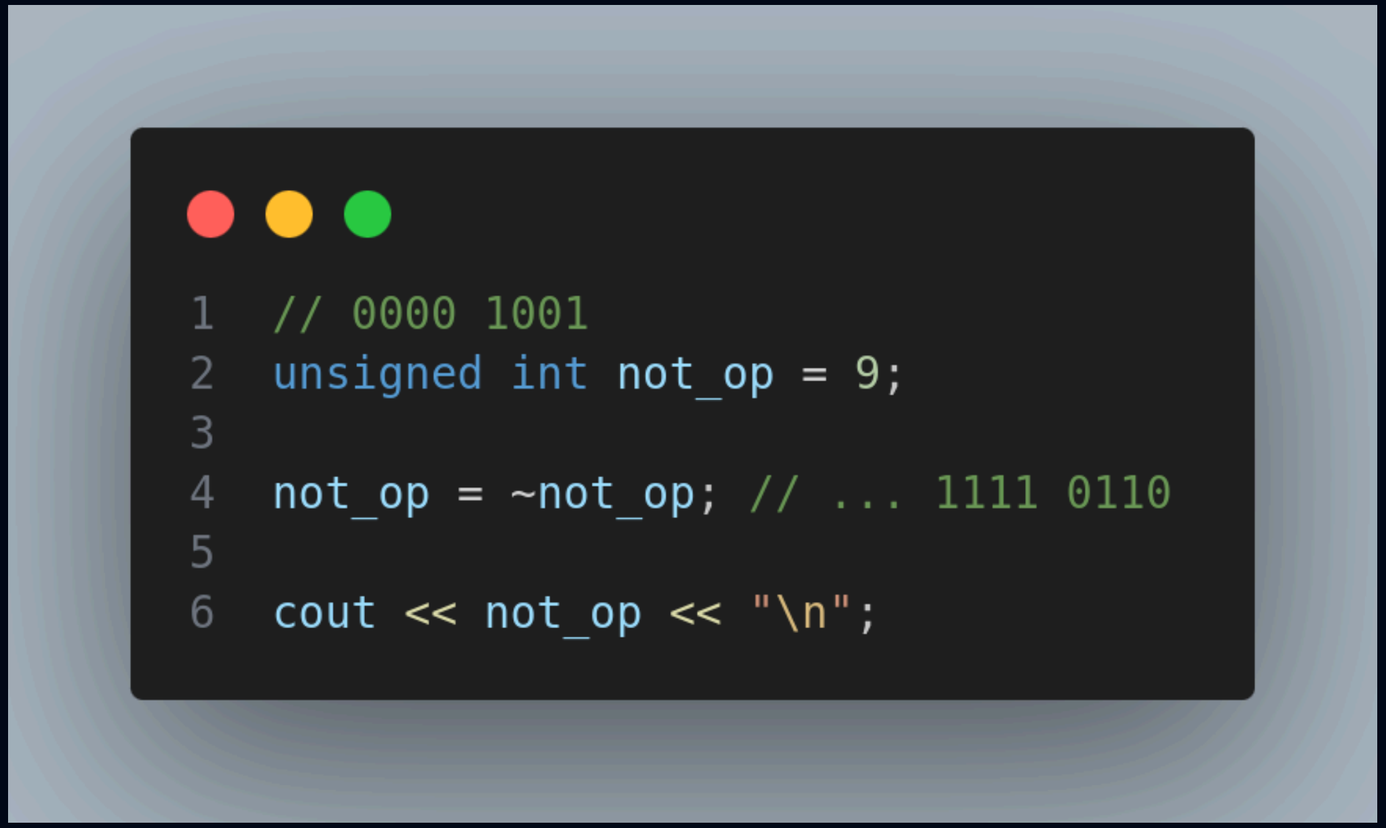
Saída

Para cada linha de entrada, o programa deverá fornecer uma linha de saída, que é o valor após somar dois números no modo "Mofiz".

[1026 - Carrega ou não Carrega ?]

Operador NOT - ~

‹Operação NOT›



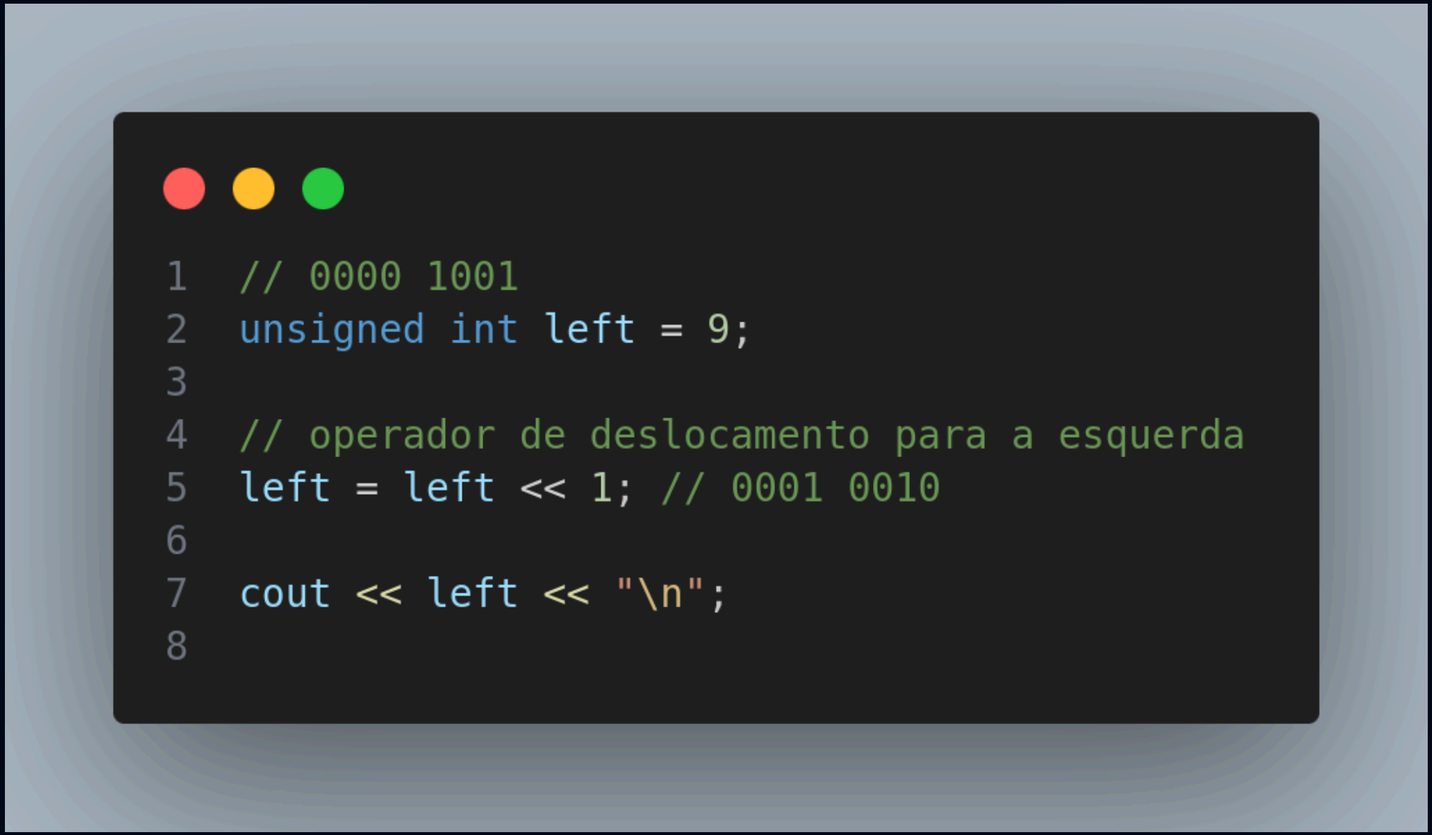
```cpp
// 0000 1001
unsigned int not_op = 9;

not_op = ~not_op; // ... 1111 0110

cout << not_op << "\n";
```

Operador SHIFT LEFT - <<

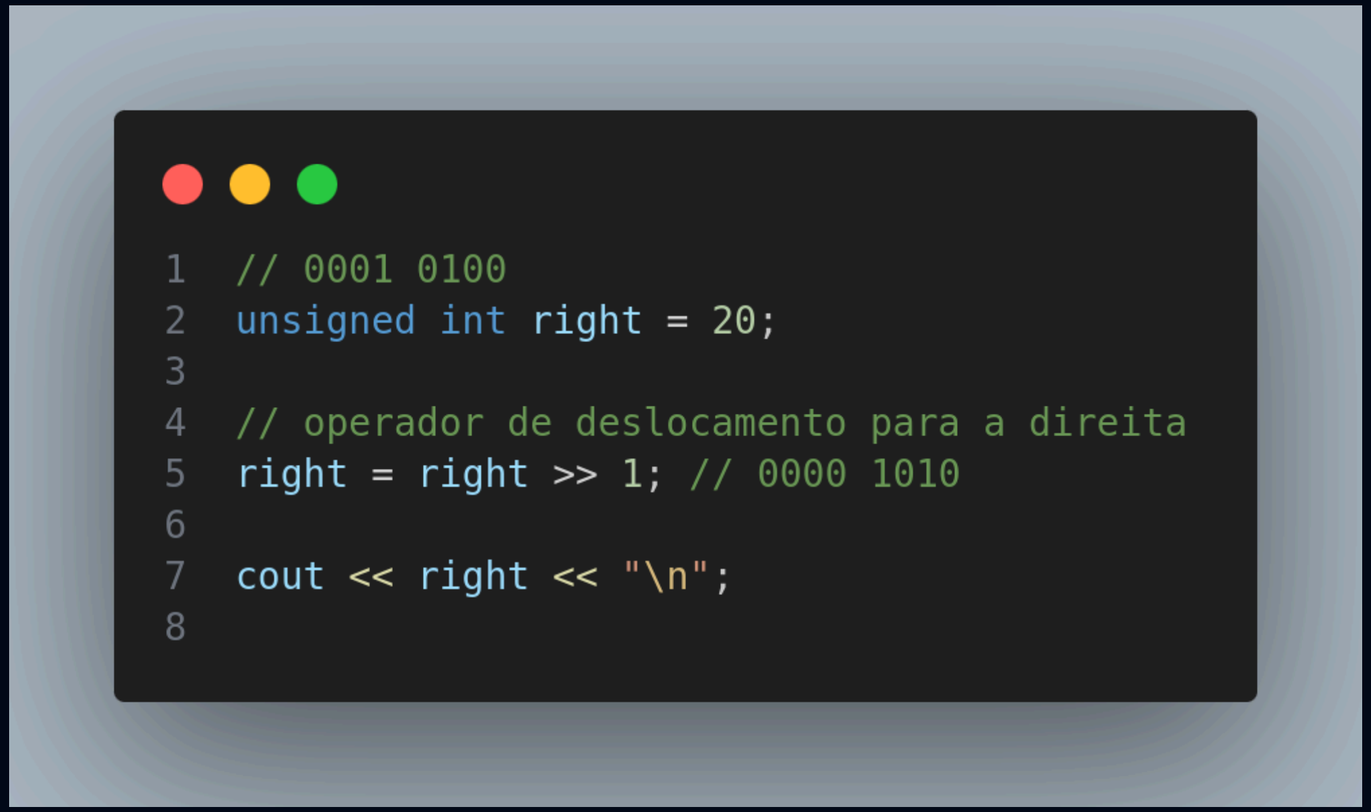
‹Operação LEFT SHIFT›



```
1  // 0000 1001
2  unsigned int left = 9;
3
4  // operador de deslocamento para a esquerda
5  left = left << 1; // 0001 0010
6
7  cout << left << "\n";
8
```

Operador SHIFT RIGHT - >>

‹Operação RIGHT SHIFT›



```cpp
// 0001 0100
unsigned int right = 20;

// operador de deslocamento para a direita
right = right >> 1; // 0000 1010

cout << right << "\n";

```

Resolução do Problema Motivador

- **Ideia:**
  - Usar um Map para guardar valor e expoente
  - Usar a fórmula:

$$NOD(N) = (a_1 + 1) \times (a_2 + 1) \times \ldots \times (a_k + 1)$$

obs: a1, a2, ak, é o expoente

A resolução estará disponível no Drive. Tente resolver por conta própria e, se precisar, compare com a solução! 😊

Lista de Exercícios

Se tiver alguma dúvida ou dificuldade na resolução de algum exercício, sinta-se à vontade para perguntar! 😊