Próximo:Funções matemáticas, Anterior:Operações de arredondamento, Acima:Números [Conteúdo][Índice]

3.8 Operações bit a bit em inteiros

Em um computador, um inteiro é representado como um número binário, uma sequência de *bits* (dígitos que são zero ou um). Conceitualmente, a sequência de bits é infinita à esquerda, com os bits mais significativos sendo todos zeros ou todos uns. Uma operação bit a bit atua nos bits individuais de tal sequência. Por exemplo, o *deslocamento* move toda a sequência para a esquerda ou para a direita em um ou mais lugares, reproduzindo o mesmo padrão movido.

As operações bit a bit no Emacs Lisp se aplicam somente a inteiros.

Função: *contagem de inteiro1 de* cinzas

`ash`(*deslocamento aritmético*) desloca os bits em *integer1 para os locais de contagem* à esquerda ou para a direita se a *contagem* for negativa. Os deslocamentos à esquerda introduzem zero bits à direita; deslocamentos à direita descartam os bits mais à direita. Considerada como uma operação de número inteiro, `ash`multiplica *integer1* por 2** *count* e, em seguida, converte o resultado em um número inteiro arredondando para baixo, em direção a menos infinito.

Aqui estão exemplos de `ash`, deslocando um padrão de bits um lugar para a esquerda e para a direita. Esses exemplos mostram apenas os bits de ordem inferior do padrão binário; bits iniciais todos concordam com o bit de ordem mais alta mostrado. Como você pode ver, deslocar para a esquerda por um é equivalente a multiplicar por dois, enquanto deslocar para a direita por um é equivalente a dividir por dois e depois arredondar para menos infinito.

```
(cinza 7 1) ⇒ 14
;; O decimal 7 se torna o decimal 14.
…000111
     ⇒
…001110


(cinza 7 -1) ⇒ 3
…000111
     ⇒
…000011


(cinza -7 1) ⇒ -14
…111001
     ⇒
…110010


(cinza -7 -1) ⇒ -4
…111001
     ⇒
…111100
```

Aqui estão exemplos de deslocamento para a esquerda ou para a direita em dois bits:

```
                  ;         valores binários
(cinza 5 2); 5 =   …000101
     ⇒ 20 ; =   …010100
(cinza -5 2) ; -5 =   …111011
     ⇒ -20 ; =   …101100
(cinza 5 -2)
     ⇒ 1 ; =   …000001
(cinza -5 -2)
     ⇒ -2 ; =   …111110
```

Função: lsh *integer1 count*

`lsh`, que é uma abreviação de *deslocamento lógico* , desloca os bits em *integer1 para as posições de contagem* à esquerda ou à direita se a *contagem* for negativa, trazendo zeros para os bits desocupados. Se *count* for negativo, *integer1* deve ser um fixnum ou um bignum positivo e `lsh`trata um fixnum negativo como se não tivesse sinal subtraindo duas vezes `most-negative-fixnum`antes de deslocar, produzindo um resultado não negativo. Esse comportamento peculiar remonta a quando o Emacs suportava apenas fixnums; hoje em dia `ash`é uma escolha melhor.

Como `lsh`se comporta `ash`, exceto quando *integer1* e *count1* são negativos, os exemplos a seguir se concentram nesses casos excepcionais. Esses exemplos pressupõem números fixos de 30 bits.

```
                ;      valores binários
(ash -7 -1); -7 = …111111111111111111111111111001
     ⇒ -4 ; = …111111111111111111111111111100
(lsh -7 -1)
     ⇒ 536870908 ; = …011111111111111111111111111100
(cinza -5 -2); -5 = …111111111111111111111111111011
     ⇒ -2 ; = …111111111111111111111111111110
(lsh -5 -2)
     ⇒ 268435454 ; = …001111111111111111111111111110
```

Função: logand *& rest ints-or-markers*

Esta função retorna o AND bit a bit dos argumentos: o *n* - ésimo bit for 1 no resultado se, e somente se, o *n* - ésimo bit for 1 em todos os argumentos.

Por exemplo, usando números binários de 4 bits, o AND bit a bit de 13 e 12 é 12: 1101 combinado com 1100 produz 1100. Em ambos os números binários, os dois bits mais à esquerda são 1, então os dois bits mais à esquerda do valor retornado são ambos 1. No entanto, para os dois bits mais à direita, cada um é 0 em pelo menos um dos argumentos, portanto, os dois bits mais à direita do valor retornado são ambos 0.

Portanto,

```
(log e 13 12)
     ⇒ 12
```

Se `logand`não for passado nenhum argumento, retorna um valor de -1. Este número é um elemento de identidade `logand` porque sua representação binária consiste inteiramente em uns. Se `logand`for passado apenas um argumento, ele retornará esse argumento.

```
                  ;        valores binários

(log e 14 13); 14 =   …001110
                  ; 13 =   …001101
     ⇒ 12 ; 12 =   …001100


(log e 14 13 4); 14 =   …001110
                  ; 13 =   …001101
                  ; 4 =   …000100
     ⇒ 4 ; 4 =   …000100


(logand)
     ⇒ -1 ; -1 =   …111111
```

Função: logior ***& resto ints-or-markers***

Esta função retorna o OR inclusivo bit a bit de seus argumentos: o *n* - ésimo bit for 1 no resultado se, e somente se, o *n* - ésimo bit for 1 em pelo menos um dos argumentos. Se não houver argumentos, o resultado será 0, que é um elemento de identidade para esta operação. Se `logior`for passado apenas um argumento, ele retornará esse argumento.

```
                  ;        valores binários

(logior 12 5); 12 =   …001100
                  ; 5 =   …000101
     ⇒ 13 ; 13 =   …001101


(logior 12 5 7); 12 =   …001100
                  ; 5 =   …000101
                  ; 7 =   …000111
     ⇒ 15 ; 15 =   …001111
```

Função: logxor ***&rest ints-or-markers***

Esta função retorna o OR exclusivo bit a bit de seus argumentos: o *n* - ésimo bit for 1 no resultado se, e somente se, o *n* - ésimo bit for 1 em um número ímpar de argumentos. Se não houver argumentos, o resultado será 0, que é um elemento de identidade para esta operação. Se `logxor`for passado apenas um argumento, ele retornará esse argumento.

```
                  ;        valores binários

(logxor 12 5); 12 =   …001100
                  ; 5 =   …000101
     ⇒ 9 ; 9 =   …001001


(logxor 12 5 7); 12 =   …001100
                  ; 5 =   …000101
                  ; 7 =   …000111
     ⇒ 14 ; 14 =   …001110
```

Função: lognot ***inteiro***

Esta função retorna o complemento bit a bit de seu argumento: o *n* - ésimo bit é um no resultado se, e somente se, o *n* - ésimo bit for zero em *integer* e vice-versa. O resultado é igual a -1 - *integer* .

```
(lognot 5)
     ⇒ -6
;; 5 =   …000101
;; torna -se
;; -6 =   …111010
```

Função: logcount *inteiro*

Esta função retorna o *peso de Hamming* de *integer* : o número de uns na representação binária de *integer* . Se *integer* for negativo, ele retornará o número de bits zero em sua representação binária de complemento de dois. O resultado é sempre não negativo.

```
(contagem de logs 43); 43 = …000101011
     ⇒ 4
(contagem de logs -43); -43 = …111010101
     ⇒ 3
```

Próximo:Funções matemáticas, Anterior:Operações de arredondamento, Acima:Números [Conteúdo][Índice]