

Representação de Dados (inteiros com sinal)

Noemi Rodriguez
Ana Lúcia de Moura

<http://www.inf.puc-rio.br/~inf1018>

Representação de Inteiros

Com **n** bits podemos representar **2^n** valores

Representação de Inteiros

Com **n** bits podemos representar **2^n** valores

para inteiros não negativos (*unsigned*) o intervalo de valores é $[0, 2^n - 1]$

Representação de Inteiros

Com n bits podemos representar 2^n valores

para inteiros não negativos (*unsigned*) o intervalo de valores é $[0, 2^n - 1]$

para inteiros com sinal, teremos 0, valores negativos e valores positivos

Representação de Inteiros

Com n bits podemos representar 2^n valores

para inteiros não negativos (*unsigned*) o intervalo de valores é $[0, 2^n - 1]$

para inteiros com sinal, teremos 0, valores negativos e valores positivos

Como representar esses valores?

há diferentes formas de representação

Sinal e Magnitude

A ideia é usar o bit mais significativo como "sinal"

"1" → valor negativo

Sinal e Magnitude

A ideia é usar o bit mais significativo como "sinal"

"1" → valor negativo

Com 4 bits temos sinal + 8 valores possíveis (0..7)

0000 ... 0111	0 a 7 decimal
1001 ... 1111	-1 a -7 decimal
1000	zero negativo?

Complemento a 2

Representação mais usual para inteiros com sinal

Alguns padrões de bits representam valores positivos e alguns representam valores negativos

uma única representação para 0

Complemento a 2

Representação mais usual para inteiros com sinal

Alguns padrões de bits representam valores positivos e alguns representam valores negativos

uma única representação para 0

O bit mais significativo também distingue valores negativos e não negativos



0 – valor não negativo
1 – valor negativo

Equivalência mod 2^n

Relação que define uma partição dos inteiros em classes de equivalência

$x \equiv y \pmod{k}$ se $|x-y| = m*k$ para algum m

Equivalência mod 2^n

Relação que define uma partição dos inteiros em classes de equivalência

$x \equiv y \pmod{k}$ se $|x-y| = m \cdot k$ para algum m

{..., -16, -8, 0, 8, 16, ...}

{..., -15, -7, 1, 9, 17, ...}

{..., -14, -6, 2, 10, 18, ...}

{..., -13, -5, 3, 11, 19, ...}

{..., -12, -4, 4, 12, 20, ...}

{..., -11, -3, 5, 13, 21, ...}

{..., -10, -2, 6, 14, 22, ...}

{..., -9, -1, 7, 15, 23, ...}

000

001

010

011

100

101

110

111

0

alguns números > 0

alguns números < 0

Representação complemento a 2

**Idéia central: é uma
representação mod 2^n**

Se $x \geq 0$ $\text{rep}_2(x) = x$

Se $x < 0$ então $\text{rep}_2(x) = 2^n + x$

→ menor positivo na classe
de equivalência de x

Representação complemento a 2

Idéia central: é uma representação mod 2^n

Se $x \geq 0$ $\text{rep}_2(x) = x$

Se $x < 0$ então $\text{rep}_2(x) = 2^n + x$

→ menor positivo na classe de equivalência de x

Exemplos para $n = 4$:

$$\text{rep}_2(-2) = 2^4 + (-2) = 14 = [1110]$$

$$\text{rep}_2(-8) = 2^4 + (-8) = 8 = [1000]$$

$$\text{rep}_2(-1) = 2^4 + (-1) = 15 = [1111]$$

binário	Compl-2	binário	Compl-2
0000	0	1111	-1
0001	1	1110	-2
0010	2	1101	-3
...	..	1001	-7
0111	7	1000	-8

É um mecanismo trabalhoso se o número de bits é muito grande...

Encontrar Representação Binária

$$x < 0 \rightarrow \text{rep}_2(x) = 2^n + x$$

Encontrar Representação Binária

$$x < 0 \rightarrow \text{rep}_2(x) = 2^n + x$$

$$2^n + x = \boxed{2^n - 1 + x} + 1$$

The diagram illustrates the decomposition of the expression $2^n + x$. It shows $2^n + x = \boxed{2^n - 1 + x} + 1$. The term $2^n - 1 + x$ is enclosed in a rectangular box, and the term $+ 1$ is enclosed in a circle. A horizontal line with two upward-pointing arrows originates from a point below the space between the box and the circle, pointing to the bottom of the box and the bottom of the circle, indicating that the sum of the two terms in the box and the term in the circle equals the original expression.

Encontrar Representação Binária

$$x < 0 \rightarrow \text{rep}_2(x) = 2^n + x$$

$$2^n + x = \boxed{(2^n - 1) - (-x)} + 1$$

↓
+

Encontrar Representação Binária

$$x < 0 \rightarrow \text{rep}_2(x) = 2^n + x$$

$$2^n + x = \boxed{(2^n - 1) - (-x)} + 1$$

1111 1111

Encontrar Representação Binária

$$x < 0 \rightarrow \text{rep}_2(x) = 2^n + x$$

$$2^n + x = \boxed{(2^n - 1) - (-x)} + 1$$

1111 1111

$$\begin{array}{l} 1 - 1 = 0 \\ 1 - 0 = 1 \end{array}$$

Encontrar Representação Binária

$$x < 0 \rightarrow \text{rep}_2(x) = 2^n + x$$

$$2^n + x = \boxed{(2^n - 1) - (-x)} + 1$$

1111 1111

$$\begin{array}{l} 1 - 1 = 0 \\ 1 - 0 = 1 \end{array}$$

complemento bit-a-bit !

Resumindo ...

Encontrar a representação de $x < 0 \rightarrow (2^n - 1) - (-x) + 1$

Resumindo ...

Encontrar a representação de $x < 0 \rightarrow (2^n - 1) - (-x) + 1$

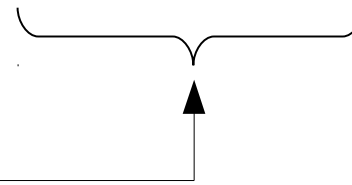
1) obter a representação de $-x$ 

Resumindo ...

Encontrar a representação de $x < 0 \rightarrow (2^n - 1) - (-x) + 1$

1) obter a representação de $-x$

2) inverter bit a bit



Resumindo ...

Encontrar a representação de $x < 0 \rightarrow (2^n - 1) - (-x) + 1$

1) obter a representação de $-x$

2) inverter bit a bit

3) somar 1



Exemplo

```
-5 → 5 → 0000 0101 → 1111 1010
                        + 1
                        1111 1011
```

para achar a representação
em comp2



Intervalo de Valores

Com **n** bits



o menor valor representado é -2^{n-1}

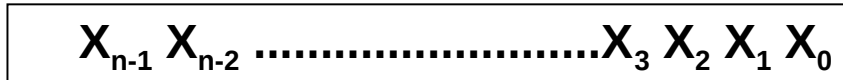
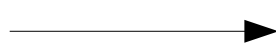


o maior valor representado é $2^{n-1}-1$



Intervalo de Valores

Com **n** bits



o menor valor representado é -2^{n-1}



o maior valor representado é $2^{n-1}-1$



Com 8 bits $\rightarrow [-2^7, 2^7-1] \rightarrow [-128, 127]$

Com 16 bits $\rightarrow [-2^{15}, 2^{15}-1] \rightarrow [-32768, 32767]$

Intervalo de Valores

Com **n** bits



o menor valor representado é -2^{n-1}



o maior valor representado é $2^{n-1}-1$



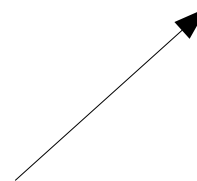
Com 8 bits $\rightarrow [-2^7, 2^7-1] \rightarrow [-128, 127]$

Com 16 bits $\rightarrow [-2^{15}, 2^{15}-1] \rightarrow [-32768, 32767]$

intervalo é assimétrico!

$\frac{1}{2}$ para os negativos

$\frac{1}{2}$ para positivos + 0



Intervalos com e sem Sinal

Exemplos para $n = 16$

	Decimal	Hex	Binary
UMax	65535	FF FF	11111111 11111111
TMax	32767	7F FF	01111111 11111111
TMin	-32768	80 00	10000000 00000000
-1	-1	FF FF	11111111 11111111
0	0	00 00	00000000 00000000

Soma e Subtração

Em complemento a 2 somas e subtrações usam **adição**

subtração é a soma do complemento

achar o complemento é trivial

Soma e Subtração

Em complemento a 2 somas e subtrações usam **adição**

subtração é a soma do complemento

achar o complemento é trivial

Aritmética módulo 2^n garante correção mesmo com sinais diferentes (a menos de *overflow*)

$$\begin{aligned}(1 - 2) \pmod{8} &\rightarrow (1 + (-2)) \pmod{8} \\ &\rightarrow (1 \pmod{8}) + (-2 \pmod{8}) \\ &\rightarrow 1 + 6 \\ &\rightarrow 7 \\ &\rightarrow \text{rep}[-1 \pmod{8}]\end{aligned}$$

Exemplos com 4 bits

$$2 + 3 \rightarrow 0010 + 0011 \rightarrow 0101 \rightarrow 5$$

$$7 - 1 \rightarrow 7 + (-1) \rightarrow 0111 + 1111 \rightarrow 0110 \rightarrow 6$$

$$(-3) + 6 \rightarrow 1101 + 0110 \rightarrow 0011 \rightarrow 3$$

$$(-1) + (-1) \rightarrow 1111 + 1111 \rightarrow 1110 \rightarrow (-2)$$

Inteiros com Sinal em C

O padrão não requer representação complemento a 2
mas a maioria das máquinas o faz

Não é boa prática assumir a faixa de valores

`<limits.h>` define constantes para os tipos de dados
inteiros

`INT_MAX`, `INT_MIN`, `UINT_MAX`

Signed e Unsigned em C

Na conversão entre tipos **de mesmo tamanho** o padrão de bits não muda apenas a **interpretação** desse padrão

Signed e Unsigned em C

Na conversão entre tipos **de mesmo tamanho** o padrão de bits não muda apenas a **interpretação** desse padrão

```
short int x = -12345; /* 1100 1111 1100 0111 */
```

Signed e Unsigned em C

Na conversão entre tipos **de mesmo tamanho** o padrão de bits não muda apenas a **interpretação** desse padrão

```
short int x = -12345; /* 1100 1111 1100 0111 */  
unsigned short ux = (unsigned short) x; /* 53191 */
```

Signed e Unsigned em C

Na conversão entre tipos **de mesmo tamanho** o padrão de bits não muda apenas a **interpretação** desse padrão

```
short int x = -12345; /* 1100 1111 1100 0111 */
unsigned short ux = (unsigned short) x; /* 53191 */

ux = 65535;          /* 1111 1111 1111 1111 */
```

Signed e Unsigned em C

Na conversão entre tipos **de mesmo tamanho** o padrão de bits não muda apenas a **interpretação** desse padrão

```
short int x = -12345; /* 1100 1111 1100 0111 */
unsigned short ux = (unsigned short) x; /* 53191 */

ux = 65535;          /* 1111 1111 1111 1111 */
x = (short int) ux; /* -1 */
```

Operadores Relacionais

Operações de comparação (<, <=, etc) devem tratar operandos com e sem sinal

existem instruções de máquina para cada caso

o compilador C gera o código adequado ao tipo de operandos

```
int a, b;  
unsigned int c, d;  
  
...  
if (a < b)    /* operandos com sinal */  
    ...  
if (c < d)    /* operandos sem sinal */  
    ...
```

Comportamento peculiar...

Em expressões com operandos de tipo **int** com e sem sinal, todos os valores são tratados como **unsigned**

```
int a[2] = {-1, 0};  
unsigned int z = 0;  
  
if (a[0] < a[1])
```


Comportamento peculiar...

Em expressões com operandos de tipo **int** com e sem sinal, todos os valores são tratados como **unsigned**

```
int a[2] = {-1, 0};  
unsigned int z = 0;  
  
if (a[0] < a[1]) /* true */
```

Comportamento peculiar...

Em expressões com operandos de tipo **int** com e sem sinal, todos os valores são tratados como **unsigned**

```
int a[2] = {-1, 0};  
unsigned int z = 0;  
  
if (a[0] < a[1]) /* true */  
  
if (a[0] < z)
```

Comportamento peculiar...

Em expressões com operandos de tipo **int** com e sem sinal, todos os valores são tratados como **unsigned**

```
int a[2] = {-1, 0};
unsigned int z = 0;

if (a[0] < a[1]) /* true */

if (a[0] < z) /* false !!! */
```

Comportamento peculiar...

Em expressões com operandos de tipo **int** com e sem sinal, todos os valores são tratados como **unsigned**

```
int a[2] = {-1, 0};
unsigned int z = 0;

if (a[0] < a[1]) /* true */

if (a[0] < z) /* false !!! */

if (a[0] < 0)
```

Comportamento peculiar...

Em expressões com operandos de tipo **int** com e sem sinal, todos os valores são tratados como **unsigned**

```
int a[2] = {-1, 0};
unsigned int z = 0;

if (a[0] < a[1]) /* true */

if (a[0] < z) /* false !!! */

if (a[0] < 0) /* true */
```

Comportamento peculiar...

Em expressões com operandos de tipo **int** com e sem sinal, todos os valores são tratados como **unsigned**

```
int a[2] = {-1, 0};
unsigned int z = 0;

if (a[0] < a[1]) /* true */

if (a[0] < z) /* false !!! */

if (a[0] < 0) /* true */

if (a[0] < 0U)
```

Comportamento peculiar...

Em expressões com operandos de tipo **int** com e sem sinal, todos os valores são tratados como **unsigned**

```
int a[2] = {-1, 0};
unsigned int z = 0;

if (a[0] < a[1]) /* true */

if (a[0] < z) /* false !!! */

if (a[0] < 0) /* true */

if (a[0] < 0U) /* false !!! */
```

Extensão de Representação

Conversões que aumentam o tamanho

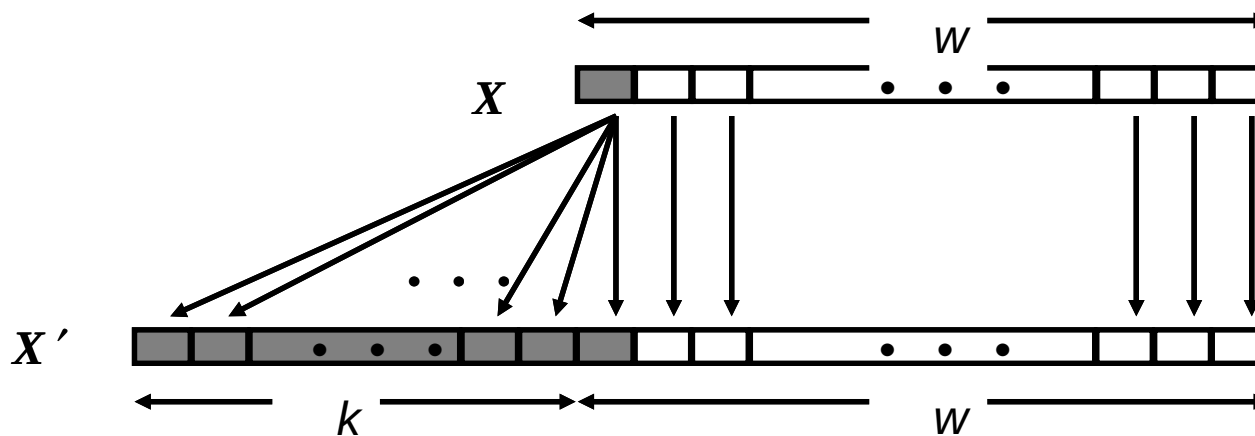
char para short, short para int, int para long, ...

Extensão de Representação

Conversões que aumentam o tamanho

char para short, short para int, int para long, ...

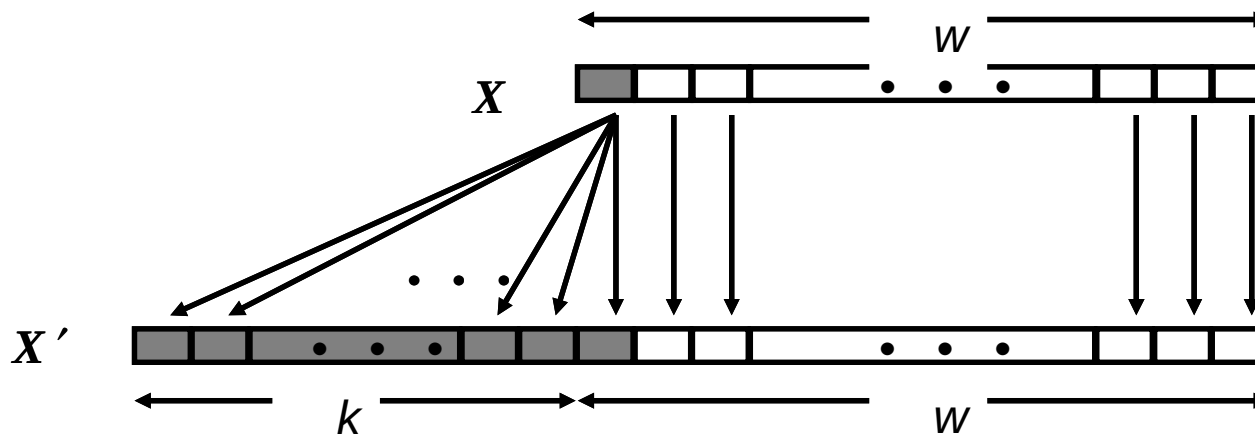
Converter representação com w bits para uma representação com $w+k$ bits, mantendo o valor



Extensão com e sem Sinal

Extensão sem sinal (zero extension)

adicionar k bits 0 à esquerda



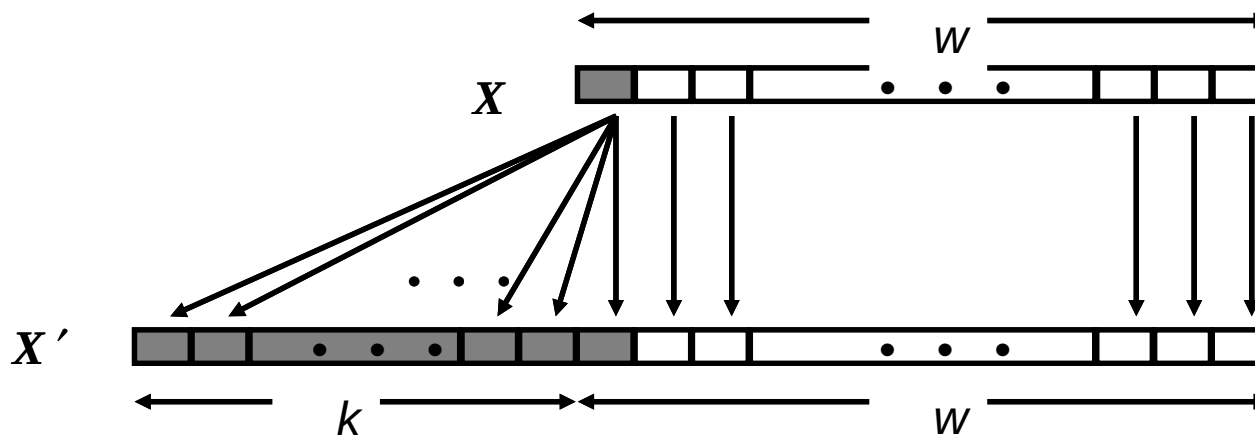
Extensão com e sem Sinal

Extensão sem sinal (zero extension)

adicionar k bits 0 à esquerda

Extensão para complemento a 2 (sign extension)

adicionar k cópias do bit mais significativo à esquerda



Exemplo de Extensão com Sinal

$\text{rep}_2(-3)$ em 8 bits $\rightarrow 2^8 + (-3) \rightarrow 253 \rightarrow$ 1111 1101

$\text{rep}_2(-3)$ em 16 bits $\rightarrow 2^{16} + (-3) \rightarrow 65533 \rightarrow$ 1111 1111 1111 1101

Exemplo de Extensão com Sinal

$\text{rep}_2(-3)$ em 8 bits $\rightarrow 2^8 + (-3) \rightarrow 253 \rightarrow$ 1111 1101

$\text{rep}_2(-3)$ em 16 bits $\rightarrow 2^{16} + (-3) \rightarrow 65533 \rightarrow$ 1111 1111 1111 1101

$$2^8 + (-3) + \textcircled{x} = 2^{16} + (-3)$$

Exemplo de Extensão com Sinal

$\text{rep}_2(-3)$ em 8 bits $\rightarrow 2^8 + (-3) \rightarrow 253 \rightarrow$ 1111 1101

$\text{rep}_2(-3)$ em 16 bits $\rightarrow 2^{16} + (-3) \rightarrow 65533 \rightarrow$ 1111 1111 1111 1101

$$2^8 + (-3) + \textcircled{x} = 2^{16} + (-3) \rightarrow x = 2^{16} + \cancel{(-3)} - 2^8 - \cancel{(-3)}$$

Exemplo de Extensão com Sinal

$\text{rep}_2(-3)$ em 8 bits $\rightarrow 2^8 + (-3) \rightarrow 253 \rightarrow$ 1111 1101

$\text{rep}_2(-3)$ em 16 bits $\rightarrow 2^{16} + (-3) \rightarrow 65533 \rightarrow$ 1111 1111 1111 1101

$$2^8 + (-3) + \textcircled{x} = 2^{16} + (-3) \rightarrow x = 2^{16} + \cancel{(-3)} - 2^8 - \cancel{(-3)}$$

$$\begin{array}{r} 1\ 0000\ 0000\ 0000\ 0000 \\ - \\ \hline 1111\ 1111\ 0000\ 0000 \end{array}$$

Truncamento

Conversões que diminuem o tamanho

short para char, int para short, long para int, ...

Truncamento

Conversões que diminuem o tamanho

short para char, int para short, long para int, ...

Converter uma representação com $w+k$ bits para uma representação com w bits

truncamento → remover os k bits mais significativos
nem sempre é possível manter o valor!

Truncamento

Conversões que diminuem o tamanho

short para char, int para short, long para int, ...

Converter uma representação com $w+k$ bits para uma representação com w bits

truncamento → remover os k bits mais significativos
nem sempre é possível manter o valor!

0000 1001	→	9
1001	→	9
1001	→	-7

unsigned
signed

Truncamento

Conversões que diminuem o tamanho

short para char, int para short, long para int, ...

Converter uma representação com $w+k$ bits para uma representação com w bits

truncamento → remover os k bits mais significativos
nem sempre é possível manter o valor!

0000 1001	→ 9
1001	→ 9
1001	→ -7

unsigned
signed

0001 1001	→ 25
1001	→ 9
1001	→ -7

unsigned
signed

Overflow em Complemento a 2

Resultado não é representável em n bits

Operandos: w bits



Soma real: $w+1$ bits



Descarta bit $w+1$



Overflow em Complemento a 2

Resultado não é representável em n bits

Operandos: w bits



Soma real: $w+1$ bits



Descarta bit $w+1$

$TAdd_w(u, v)$



Hardware indica situações de overflow signed e unsigned

Mas a indicação é ignorada...

compilador C não gera testes