13. Representação dos inteiros

DIM0321

2015.1

Outline

Bits e bytes

Definição (Bit)

- Um **bit** é a unidade básica de informação no computador
- Tem 2 valores
  - 1 verdadeiro / ligado
  - 0 falso / desligado

Definição (Byte)

- Um **byte** é o menor agrupamento de bits manipulável pelo processador
- Hoje 1 byte = 8 bits (já foi 6 ou 7 bits)
- A metade de 1 byte é um **nibble**

Representação posicional numérica

Representações

- 22, XXII, vinte e dois, twenty-two, vingt-deux, zweiundzwanzig,, 0x16, 0o26 são representações da quantidade numérica "22".
- Em matemática, usa-se o sistema (posicional numérico) decimal
- Em ciências da computação, usa-se também os sistemas (posicionais numéricos) hexadecimal, binário, octal.

Sistemas posicionais

- Num sistema posicional, um número x pode ser representado num sistema de base b com o polinômio:

$$d_n b^n + d_{n-1} b^{n-1} + ... + d_1 b^1 + d_0 b^0 + d_{-1} b^{-1} + ... + d_m b^m$$

- Ou

$$(d_n d_{n-1} ... d_1 d_0 . d_{-1} ... d_m)_b$$

- $153,24_{10} = 1 * 10^2 + 5 * 10^1 + 3 * 10^0 + 2 * 10^{-1} + 4 * 10^{-2}$

Representação binária de naturais

Ordem de leitura

- As representações numéricas usuais se leem da esquerda para a direita.
- O computador (como o ser humano) pode ler nos dois sentidos.

Endianness (Lilliput vs. Blefuscu)

A **extremidade** maior determina a ordenação da sequência dos bits. Pode ser:

- **extremidade menor primeiro** (little-endian, ordem crescente) : x86, AMD, Z80
  - $2_{10} = 01_2$
- **extremidade maior primeiro** (big-endian, ordem decrescente) : Motorola 68000, PowerPC
  - $2_{10} = 10_2$
- A arquitetura ARM é **bi-endian**, sabe enviar dados em qualquer dos 2 formatos

Exercícios

1. Representar em formato binário $138_{10}, (43)_{10}, 200_{10}$
2. Representação decimal dos números *little-endian* $01010101_2, 10101010_2, 11001100_2$
3. Escrever um programa que lê um número natural entre 0 e 255 e imprime a sua representação binária.

Representação

Sequencia de bits

Um inteiro é um sequencia de bits, i.e. de dígitos na base 2. O tipo `int` tem (geralmente) 32 bits.

Sinal

- Tipos inteiros podem conter uma informação de sinal para representar números negativos.
- Esta informação, se tiver presente, ocupa **1 bit**

Bit de Sinal

- Se o bit de sinal for 1, o número é **negativo**
- Se for 0, o número positivo.
- $0_{10} = 0....0_2$: 0 é "positivo"

Tipos inteiros com sinal

Definição

Tipo	# bits	# bytes	Mínimo	Máximo	Formato
char	8	1	`SCHAR_MIN`	`SCHAR_MAX`	`%c, %hhi`
short	≥ 16	≥ 2	`SHRT_MIN`	`SHRT_MAX`	`%hi`
int	≥ 16	≥ 2	`INT_MIN`	`INT_MAX`	`%i, %d`
long	≥ 32	≥ 4	`LONG_MIN`	`LONG_MAX`	`%li, %ld`
long long	≥ 64	≥ 8	`LLONG_MIN`	`LLONG_MAX`	`%lli, %lld`

- Limites definidos em `limits.h`
- Tamanhos reais **dependem** da arquitetura da máquina (e do compilador)

O tipo int

Intervalos mínimos

Definição

Tipo	Intervalo mínimo
char	[-127, 127]
short	[32767,+32767]
int	[32767,+32767]
long	[2147483647,+2147483647]
long long	[9223372036854775807,+9223372036854775807]

Relação entre os tamanhos

- `long long` $\geq$ `long` $\geq$ `int` $\geq$ `short`
- `char` é sempre o menor tipo de dado representável

Tipos inteiros sem sinal

Definição

Tipo	# bits	# bytes	mínimo	máximo
unsigned char	8	1	0	`UCHAR_MAX` (255)
unsigned short	≥ 16	≥ 2	0	`USHRT_MAX`
unsigned int	≥ 16	≥ 2	0	`UINT_MAX`
unsigned long	≥ 32	≥ 4	0	`ULONG_MAX`
unsigned long long	≥ 64	≥ 8	0	`ULLONG_MAX`

- limites definidos em `limits.h`
- versões sem sinal dos tipos inteiros com sinal

Teste dos valores da arquitetura

```c
#include <limits.h>
#include <stdio.h>

int main(void)
{
     printf("%hhi %hhi %hhu\n", CHAR_MIN, CHAR_MAX, UCHAR_MAX);
     printf("%hd %hi %hu\n", SHRT_MIN, SHRT_MAX, USHRT_MAX);
     printf("%d %d %u\n", INT_MIN, INT_MAX, UINT_MAX);
     printf("%ld %ld %lu\n", LONG_MIN, LONG_MAX, ULONG_MAX);
     printf("%lld %lld %llu\n", LLONG_MIN, LLONG_MAX, ULLONG_MAX);
     return 0;
}
```

Resultado (AMD Athlon(tm) II X2 B28 Processor, GCC 4.8.2)

-128	127	255
-32768	32767	65535
-2147483648	2147483647	4294967295
-9223372036854775808	9223372036854775807	18446744073709551615
-9223372036854775808	9223372036854775807	18446744073709551615

Representação dos negativos

Bit de Sinal

- Se o bit de sinal for 1, o número é **negativo**
- Se for 0, o número positivo.

Complemento a 2

Para calcular a representação binária de $-n$:

1. Calcule a representação binária de n
2. Inverta os bits de n
3. Adicione 1 ao valor obtido na etapa 2

Exemplo

```c
#include <stdio.h>
#include <limits.h>

void ppint2bin(unsigned int n)
{
     unsigned m = n;
     unsigned i;
     for (i = 0; i < CHAR_BIT * sizeof(m); i++) {
  if (m & 1u) putchar('1');
  else putchar('0');
  m >>= 1;
     }
     printf("\n");
}

int main(void)
{
     ppint2bin(1);
     ppint2bin(-1);
     ppint2bin(~1);
     return 0;
}
```

```
10000000000000000000000000000000
11111111111111111111111111111111
01111111111111111111111111111111
```


Conversões / promoções

Definição

- A linguagem C automaticamente transforma um valor de um tipo em um outro.
- **Conversões** são transformações que preservem valores.
- **Promoções** são transformações que **não sempre** preservem valores.
- Os avisos do seu compilado são preciosos.

Conversões implícitas

- A linguagem tem regras internas para converter **automaticamente** valores de um tipo em um outro tipo.

Regras (operadores aritméticos)

1. Se um operando for `long double`, o outro é convertido em `long double`.
2. Senão, se um for `double` , o outro é convertido em ˜ double˜ .
3. Senão, se um for `float` , o outro é convertido em `float` .
4. Senão promoções inteiras são aplicadas aos 2 operandos
   1. Se um for `unsigned long`, o outro é convertido em `unsigned long`
   2. Senão se um for `long`, e o outro `unsigned int`
      1. se `unsigned` ⊆ `long`, então converta o em `long`
      2. senão os dois são convertidos em `unsigned long`
   3. Senão se um for `long`, o outro é convertido em `long`
   4. Senão se um for `unsigned`, o outro é convertido em `unsigned`
   5. Senão os dois operandos tem o tipo `int`.

Tipos assinados e não assinados

```
1 if (-1L < 1U) printf("foo\n");
2 if (-1L > 1UL) printf("bar\n");
```

- É um problema complexo : nem GCC 4.8.2, nem clang 3.4 respeitam o padrão!

Conversão explicita: coerção

Exemplo

```
1 if (-1L < 1U) printf("foo\n");
2 if (-1L > 1UL) printf("bar\n");
```

Com conversões explícitas

```
1 if ((unsigned long) -1L < (unsigned long) 1U) printf("foo\n");
2 if ((unsigned) -1L > 1UL) printf("bar\n");
```


Representação hexadecimal

Definição

- A base é 16
- Os dígitos são 0...9, A...F
- Um dígito hexadecimal representa **1 nibble** (a metade de 1 byte)

Exemplo

- $0_{10} = 0_{16}$
- $255_{10} = ff_{16}$

Exercício

1. Escreva em notação decimal
   - 0xda
   - 0x12
   - 0xffffffff
2. Representação binária de '0x100, 0xf0f0, 0xabcd'
3. Representação hexadeciamal de '1111111100000000', '1010000110000001'

Conversão

```
1  #include <stdio.h>
2  #include <limits.h>
3
4  void pphex(unsigned n)
5  {
6       if (h < 10) printf("%u", h);
7       else switch (h) {
8    case 10:
9         printf("a");
10        break;
11   case 11:
12        printf("b");
13        break;
14   case 12:
15        printf("c");
16        break;
17   case 13:
18        printf("d");
19        break;
20   case 14:
21        printf("e");
22        break;
23   case 15:
24        printf("f");
25        break;
26   default:
27        /* not reached */
28        break;
29   }
30 }
```

```
1  void bin2hex(unsigned n)
2  {
3       unsigned m = n;
4       unsigned i, j, h, b;
5       for (i = 0; i < sizeof(unsigned); i++) {
6    h = 0;
7    b = 1;
8    for (j = 0;  j < 4; j++, b *= 2) {
9         if (m >> j & 1u) h += b;
10   }
11   pp_hex(h);
12   m >>= 4;
13      }
14      printf("\n");
15 }
16
17 int main(void)
18 {
19      unsigned n;
20      scanf("%u", &n);
21      bin2hex(n);
22      printf("%x\n", n);
23      return 0;
24 }
```

Aritmética modulo

Limitações

- Os inteiros básicos de C são limitados
- Existe também meios de usar aritmética de precisão arbitrária
- Para C = biblioteca `GMP`

```c
#include <limits.h>
#include <stdio.h>
int main(void)
{
     printf("[%d, %d]\n", INT_MIN, INT_MAX);
     printf("%d\n", INT_MAX + 1);
     printf("%d\n", INT_MIN - 1);
     return 0;
}
```

```
[-2147483648,    2147483647]
 -2147483648
  2147483647
```


Operadores bit a bit

Definição

Nome	Sintáxe	Comentário
Deslocamento à esquerda	<<	novo valor = 0, $k << n == k * 2^n$
Deslocamento à direita	>>	novo valor = 0, $k >> n == k/2^n$
Negação	~	inversão de bit
Conjunção	&	conjunção bit por bit
Disjunção	|	disjunção bit por bit
Disjunção exclusiva	^	bit por bit

Versões curtas

- >>=, <<=, &=, |=, ^= são definidos
- `x |= 1` é `x = x | 1`

Exemplo

```c
#include <stdio.h>

int main(void)
{
     int n = 16;
     unsigned m;
     n >>= 1;
     printf("%d\n", n);
     m = n >> 3 & 1;
     printf("%u\n", m);
     n |= 2;
     printf("%d\n", n);
     n ^= 8;
     printf("%d\n", n);
     return 0;
}
```

Resultados

```
8
1
10
2
```

Exercício

- O que faz o código abaixo ?

```c
int popcount(unsigned n)
{
    int count = 0;
    while (n != 0) {
  if (n & 1u) count++;
  n >>= 1;
    }
    return count;
}
```


Referências

Precisões

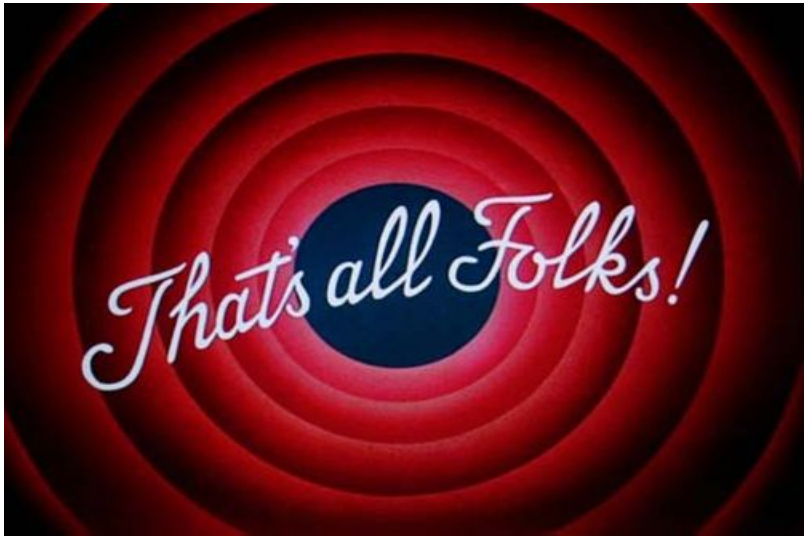
[KR88] 2.9, 2.12, A.6, B.11

[Bac13] 2.1.3, 2.1.4, 3.5, 3.8, 14.3.3, 15.1

André Backes, *Linguagem C completa e descomplicada*, Elsevier, 2013.

Brian W. Kernighan and Dennis M. Ritchie, *The (ANSI) C Programming Language*, 2nd ed., Prentice Hall Professional Technical Reference, 1988.

Perguntas ?



http://dimap.ufrn.br/~richard/dim0321