



Capítulo 3

Aula 8

Aritmética Computacional
Representação Numérica



Representação Numérica

- Base binária (base 2) :
Símbolos: 0,1
□ Ex.: $124 = 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 1111100_2$
- Base octal (base 8):
Símbolos: 0,1,2,3,4,5,6,7
□ Ex.: $124 = 1 \times 8^2 + 7 \times 8^1 + 4 \times 8^0 = 174_8$
- Base decimal (base 10):
Símbolos: 0,1,2,3,4,5,6,7,8,9
□ Ex.: $124 = 1 \times 10^2 + 2 \times 10^1 + 4 \times 10^0 = 124_{10}$
- Base hexadecimal (base 16):
Símbolos: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
□ Ex.: $124 = 7 \times 16^1 + 12 \times 16^0 = 7C_{16}$

Generalizando

- Número de N dígitos na base B possibilita representar B^N elementos diferentes (coisas).

$$(d_{N-1}d_{N-2}d_{N-3}\dots d_2d_1d_0)_B$$

$$valor = \sum_{i=0}^{N-1} d_i \times B^i$$

3

Conversão Decimal, Binário, Hexa

00	0000	0
01	0001	1
02	0010	2
03	0011	3
04	0100	4
05	0101	5
06	0110	6
07	0111	7
08	1000	8
09	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Ex.: $1010\ 1100\ 0101_2 = AC5_{16}$

$10\ 1111_2 = 2F_{16}$

$3F9_{16} = 11\ 1111\ 1001_2$

4

Operações :

- Somar
- Subtrair
- Multiplicar
- Dividir
- Comparar
- ...

Decimal: Bom para Humanos (pq?)
 Binário: Bom para Computadores
 Hexa: Bom para quem? (conversão!)

Ex.: Decimal: $13 + 7 = 20$
 Binário: $1101 + 111 = 10100$
 Hexa: $1D + 7 = 24$

$$A + 3 + 10 = ??$$

↓
 Importante definir a Base!

5

Números:

- **Bits são apenas Bits!!** Sem nenhum significado inerente
 - Ex.: o que é: 10100101 ????

Pode representar um número, caractere, instrução, pixel (luminância), amostra de voz, temperatura, posição, qualquer **coisa**...

O que podemos representar com N bits?

Apenas 2^N coisas!

Logo: Bom para coisas limitadas (contáveis)

Ex.: 26 Letras: 5 bits é suficiente

Caracteres ASCII: 7 bits (de 8) (A,a,!))

Caracteres UNICODE: 16 bits (caracteres dos idiomas conhecidos)

6

Limitações:

- Se os bits representarem número:
Convenções definem a relação entre bits e números.
- Complicadores:
Números são infinitos!
Tipos: Naturais, Inteiros, Reais, Irracionais, Complexos...
Qual N necessário?????
- Ex.: Como representamos os Números Inteiros (com sinal)?

7

Exemplos de Representações Binárias

Sinal e magnitude	Complemento de um	Complemento de dois
000 = +0	000 = +0	000 = +0
001 = +1	001 = +1	001 = +1
010 = +2	010 = +2	010 = +2
011 = +3	011 = +3	011 = +3
100 = -0	100 = -3	100 = -4
101 = -1	101 = -2	101 = -3
110 = -2	110 = -1	110 = -2
111 = -3	111 = -0	111 = -1

- Qual é o melhor? Por quê?

8

Comparação

- Sinal e Magnitude:
 - ☺ Fáceis de entender(ler). Fácil de negar. Simetria na representação.
 - ☹ Circuitos aritméticos complexos. Existe +0 e -0.

- Complemento de 1:
 - ☺ Fácil de negar. Simetria na representação
 - ☹ Circuitos aritméticos ainda complexos. Existe +0 e -0

- Complemento de 2:
 - ☺ Circuitos aritméticos mais simples. 1 único Zero.
 - ☹ Representação assimétrica (+3,-4) (Maior faixa dinâmica! ☺)
 - Negação um pouco mais complexa.

9

Complemento de 2

■ Obtenção:

- **A partir do valor sem sinal:**
complementar e somar 1

Ex.: $5_{10} = 0101_2$
 $-5_{10} = 1010 + 1 = 1011_2$ ↷

- **Ponderação dos Dígitos:**

$$\text{valor} = -d_{N-1}2^{N-1} + d_{N-2}2^{N-2} + d_{N-3}2^{N-3} + \dots + d_22^2 + d_12^1 + d_02^0$$

Ex.:

$$1010_2 = -6_{10} = -1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$

10

MIPS

Números de 32 bits com sinal:

```

0000 0000 0000 0000 0000 0000 0000 0000bin = 0dec
0000 0000 0000 0000 0000 0000 0000 0001bin = 1dec
0000 0000 0000 0000 0000 0000 0000 0010bin = 2dec
...
0111 1111 1111 1111 1111 1111 1111 1101bin = 2.147.483.645dec
0111 1111 1111 1111 1111 1111 1111 1110bin = 2.147.483.646dec
0111 1111 1111 1111 1111 1111 1111 1111bin = 2.147.483.647dec
1000 0000 0000 0000 0000 0000 0000 0000bin = -2.147.483.648dec
1000 0000 0000 0000 0000 0000 0000 0001bin = -2.147.483.647dec
1000 0000 0000 0000 0000 0000 0000 0010bin = -2.147.483.646dec
...
1111 1111 1111 1111 1111 1111 1111 1101bin = -3dec
1111 1111 1111 1111 1111 1111 1111 1110bin = -2dec
1111 1111 1111 1111 1111 1111 1111 1111bin = -1dec

```

11

Operações de complemento a dois

- Negar um número de complemento a dois:

Inverta todos os bits e some 1

lembre-se: “negar” e “inverter” são muito diferentes!

$$x + \bar{x} = 111\dots111_2 = -1$$

$$\bar{x} + x + 1 = 0$$

$$\bar{x} + 1 = -x$$

- **Extensão de Sinal:**

Converter números de n bits em números com mais de n bits:

Ex.: o campo imediato de 16 bits do MIPS é convertido em 32 bits para aritmética

Copie o bit mais significativo (o bit de sinal) para os outros bits

0010 -> 0000 0010 = 2 (infinitos zeros)

1010 -> 1111 1010 = -6 (infinitos uns)

Mostrar calc windows

- “extensão de sinal” (lbu versus lb):
Carrega 1 byte da memória para um registrador.

12

Comparação de números com e sem sinal

- Suponha que:

\$s0 armazene o número

1111 1111 1111 1111 1111 1111 1111 1111₂

\$s1 armazene o número

0000 0000 0000 0000 0000 0000 0000 0001₂

Quais os valores de \$t0 e \$t1 dado os comandos abaixo?

slt \$t0, \$s0, \$s1 #comparação com sinal

sltu \$t1, \$s0, \$s1 #comparação sem sinal

Logo: \$t0=1 e \$t1=0 pq?

- Exemplo: Considere a verificação de um índice i que aponte para um elemento válido de $v[\text{dim}]$.

```
if( $i < 0 || i \geq \text{dim}$ )
    goto indice_fora_limite;
```

stlu \$t0, \$a1, \$t2 # \$t0=0 se \$a1 (i) >= \$t2 (dim) ou \$a1 (i) < 0

beq \$t0, \$zero, indice_fora_limite

- Obs.: Tipos em C

- unsigned char e char : 8 bits : 0...255 , -128...127
- unsigned short e short : 16 bits: 0...65536 , -32768...32767
- unsigned int e int : 32 bits: 0...2³²-1 , -2³¹...2³¹-1

Adição e subtração

- Exatamente como base decimal (emprestar/vai 1s)

$$\begin{array}{r} 0111 \\ + 0110 \\ \hline \end{array} \quad \begin{array}{r} 0111 \\ - 0110 \\ \hline \end{array} \quad \begin{array}{r} 0110 \\ - 0101 \\ \hline \end{array}$$

- Facilidade de operações do complemento de dois
- subtração usando adição para números negativos

$$\begin{array}{r} 0111 \\ + 1010 \\ \hline \end{array}$$

- Overflow (resultado muito grande para a word finita do computador):
Somar dois números de n bits pode produzir um número de n+1 bits.

Obs.:
$$\begin{array}{r} 0111 = 7 \\ + 0001 = +1 \\ \hline 1000 \neq 8 \end{array}$$
 *note que o termo overflow é um pouco confuso;
ele não significa que um carry simplesmente "transbordou"*

Detectando overflow

- Nenhum overflow quando somar um número positivo com um negativo
- Nenhum overflow quando sinais são iguais para subtração
- O overflow ocorre quando o valor afeta o sinal:
 - overflow ao somar dois positivos produz um negativo
 - ou, somar dois negativos produz um positivo
 - ou, subtrair um negativo de um positivo e obtenha um negativo
 - ou, subtrair um positivo de um negativo e obtenha um positivo



Efeitos do overflow

- Uma exceção (interrupção) ocorre
 - O controle salta para um endereço predefinido para exceção
 - O endereço interrompido é salvo para uma possível retomada
- Detalhes baseados na linguagem/sistema de software
 - Nem sempre desejamos detectar overflow — novas instruções
 - MIPS: addu, addiu, subu
 - Nota:* addiu ainda com extensão de sinal
 - Nota:* sltu, sltiu para comparações sem sinal

Obs.: C *versus* FORTRAN:

Propriedade do Complemento de 2:

Se o resultado final de uma soma (subtração) de várias parcelas não gerar overflow, ele estará correto mesmo que as parcelas intermediárias o gerem.