Arquitectura de Computadores I 2007/08

1º Semestre de 2007/2008

Bernardo Cunha, José Luís Azevedo, Arnaldo Oliveira

Arquitectura de Computadores I 2007/08

Aula 11

Representação de números inteiros com sinal
- Sinal e módulo
- Complemento para um
- Complemento para dois

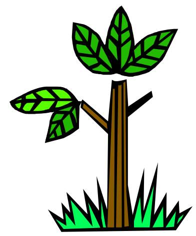
Exemplos de operações aritméticas

Overflow e mecanismos para a sua detecção

Arquitectura de Computadores I 2007/08

Um dia, encontrando-se um jardineiro persa com um grupo de doutores, colocou-lhes a seguinte questão:

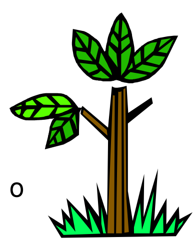
Digam-me doutos senhores
No vosso saber sem fim
Quantas folhas contais vós
Neste arbusto de jardim



Universidade de Aveiro — Slide 11 - 3

Arquitectura de Computadores I 2007/08

- Fácil, disse o matemático líbio: Há **00000101** folhas
- Não, - retorquiu o geómetra persa- neste arbusto há **01000011 01001001 01001110 01000011 01001111** folhas!
- Pois eu cá acho que há **01010110** folhas – contestou o contabilista romano
- Nada disso. Há **01000000101000000000000000000000** – afirmou o sábio grego!
- Por todos os deuses – disse o sírio – só um cego não vê que o número de folhas é **10000100**.



Universidade de Aveiro — Slide 11 - 4

Arquitectura de Computadores I 2007/08

- A resposta dada por cada um dos sábios, na realidade, foi a mesma. Apenas usaram uma linguagem (código) diferente.
- A extracção da informação requer, assim, o conhecimento do código usado, sob pena de as mensagens não passarem de colecções de bits sem sentido.

O matemático líbio codificou a sua resposta em binário: **00000101_2** = 5_{10}

O geómetra persa usou ASCII (em português):
01000011 01001001 01001110 01000011 01001111 = "CINCO"

O contabilista romano usou ASCII mas para representar numeração romana
01010110 = "V"

O sábio grego usou representação em vírgula flutuante
$01000000101000000000000000000000=1.01_2 x 2^2=5$

O sírio usou excesso 2^{n-1}-1 (com n=8). **10000100_2** = 5_{10}

Universidade de Aveiro Slide 11 - 5

Arquitectura de Computadores I 2007/08

Representação de inteiros

No sistema árabe, cada algarismo que compõe um dado número tem um peso que é função quer da sua posição no número quer do número de símbolos do alfabeto usado.

Um número com *n+1* dígitos $\mathbf{d_n\, d_{n-1}\, d_{n-2} \ldots d_1\, d_0}$

representado neste sistema, pode ser decomposto num polinómio da forma

$$\mathbf{d_n.b^n + d_{n-1}.b^{n-1} + d_{n-2}.b^{n-2} + \ldots + d_1.b^1 + d_0.b^0}$$

em que **b** é a base de representação e corresponde à dimensão do alfabeto

Exemplos:

1230_{10} = $1 \times 10^3 + 2 \times 10^2 + 3 \times 10 + 0$
110101_2 = $1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2 + 1 = 53_{10}$
721_8 = $7 \times 8^2 + 2 \times 8 + 1 = 465_{10}$
$5A8_{16}$ = $5 \times 16^2 + A \times 16 + 8 = 1448_{10}$

Universidade de Aveiro Slide 11 - 6

Arquitectura de Computadores I 2007/08

Representação de inteiros

- Uma vez que um computador é um sistema digital binário, a representação de inteiros faz-se sempre em base 2 (símbolos 0 e 1)
- Por outro lado, como o espaço de armazenamento de informação (numérica ou não) é limitado, a representação de inteiros é também necessariamente limitada. Tipicamente, um inteiro pode ocupar um número de bits igual à dimensão de um registo interno do CPU.
- A gama de valores inteiros representáveis é assim finita, e corresponde ao número máximo de combinações que é possível obter com o número de bits que compõem um registo.
- No MIPS, um inteiro ocupa 32 bits, pelo que o número de inteiros representável será:

$$N_{inteiros} = 2^{32} = 4294967296_{10} = [0 .. 4294967295_{10}]$$

Universidade de Aveiro Slide 11 - 7

Arquitectura de Computadores I 2007/08

Representação de inteiros

- Os circuitos aritméticos estão igualmente limitados a um número finito de dígitos (bits), geralmente igual à dimensão dos registos internos do CPU.
- Os circuitos aritméticos operam assim em aritmética modular, ou seja em **mod(2^n)** em que *n* é o número de bits de representação.
- O maior valor que um resultado aritmético pode tomar será portanto 2^n-1, sendo o valor inteiro imediatamente a seguir o valor zero (representação circular).
- Num CPU com registos de 8 bits, por exemplo, o resultado da soma dos números 11001011 e 00110111 seria:

11001011 + 00110111 = 1 $\boxed{00000010}$

Carry – não cabe no registo resultado

Resultado com 8 bits

Universidade de Aveiro Slide 11 - 8

Arquitectura de Computadores I 2007/08

Representação de inteiros negativos

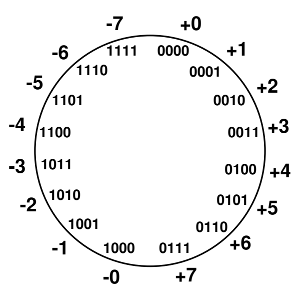
- A representação de números positivos é a mesma na maioria dos sistemas numéricos
- Os maiores problemas colocam-se quando se procura uma forma de representar quantidades negativas
- Os três esquemas mais usados são:
  - ✓ sinal e módulo
  - ✓ complemento para um
  - ✓ complemento para dois

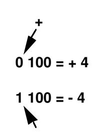
Por uma questão de simplicidade vamos admitir, na discussão subsequente, que a dimensão do registo interno do CPU é de 4 bits

Universidade de Aveiro Slide 11 - 9

Arquitectura de Computadores I 2007/08

Representação em sinal e módulo





- O bit mais significativo é o sinal: 0 = positivo (ou zero), 1 = negativo
- A magnitude é representada pelos 3 LSBits: 0 (000) a 7 (111)
- Gama de representação para n bits = $+/-2^{n-1}-1$
- 2 Representações para 0

Universidade de Aveiro Slide 11 - 10

Arquitectura de Computadores I 2007/08

Representação em sinal e módulo

Este método de representação de inteiros apresenta os seguintes problemas do ponto de vista da implementação numa ALU:

- Existem duas representações distintas para um mesmo valor (zero)
- É necessário comparar as magnitudes dos operandos para determinar o sinal do resultado
- É necessário implementar um somador e um subtractor distintos
- O bit de sinal tem de ser tratado independentemente dos restantes

Arquitectura de Computadores I 2007/08

Representação em complemento para um

Definição: Se N é um número positivo, então $\overline{N}$ é negativo e o seu complemento para 1 (complemento falso) é dado por:

$$\overline{N} = (2^n - 1) - N$$

Exemplo: determinar o complemento para 1 de 5

$N = 5_{10} = \mathbf{0101}_2$

$2^n = 2^4 = 10000$

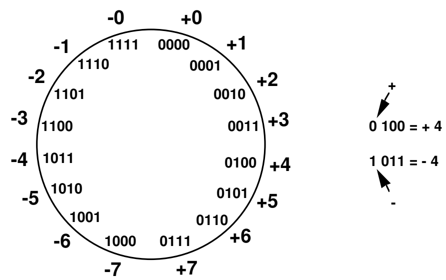
$(2^n - 1) = 10000 - 1 = 1111$

$(2^n - 1) - N = 1111 - 0101 = \mathbf{1010} = \overline{\mathbf{N}}$

O complemento para 1 pode ser calculado negando um a um os bits que compõem o número.

Arquitectura de Computadores I — 2007/08

Representação em complemento para um



- O bit mais significativo também pode ser interpretado como sinal:
  0 = positivo 1 = negativo
- Há 2 Representações para 0
- A subtracção faz-se adicionando o complemento para 1

Universidade de Aveiro — Slide 11 - 13

Arquitectura de Computadores I — 2007/08

Exemplos de adições e subtracções em complemento para 1

4	0100	-4	1011
+ 3	0011	+ 3	0011
7	0111	-1	1110

> Quando ocorre *carry* este tem de ser somado ao resultado intermédio!

4	0100	-4	1011
- 3	1100	+ (-3)	1100
1	10000	-7	10111
carry final	1	*carry* final	1
	0001		1000

Universidade de Aveiro — Slide 11 - 14

Arquitectura de Computadores I 2007/08

Adições e subtracções em complemento para 1

Porque funciona o *carry* final?

É equivalente a subtrair 2^n e somar 1

		Rep. em complemento para 1
se M>N	**M - N positivo**	**M - N**
se M<=N	**M - N negativo ou nulo**	$2^n - 1 - (N - M)$

$$M + \overline{N} = M + (2^n - 1 - N) = (M - N) + 2^n - 1$$

se M>N	$(M - N) + 2^n - 1 > 2^n - 1$	**c/ soma carry e mod. = M - N**
se M<=N	$(M - N) + 2^n - 1 <= 2^n - 1$	**e é igual a** $2^n - 1 - (N - M)$

Arquitectura de Computadores I 2007/08

Adições e subtracções em complemento para 1

Porque funciona o *carry* final?

É equivalente a subtrair 2^n e somar 1

$$M - N = M + \overline{N} = M + (2^n - 1 - N) = (M - N) + 2^n - 1 \quad (M > N)$$

$$-M + (-N) = \overline{M} + \overline{N} = (2^n - M - 1) + (2^n - N - 1)$$

$$= 2^n + [2^n - 1 - (M + N)] - 1$$

$$M + N < 2^{n-1}$$

após *carry* final:

$$= 2^n - 1 - (M + N)$$

Representação correcta de -(M + N)

Representação em complemento para dois

Definição: Se N é um número positivo, então N* é o seu complemento para 2 (complemento verdadeiro) e é dado por:

$$N^* = 2^n - N$$

Exemplo: determinar o complemento para 2 de 5

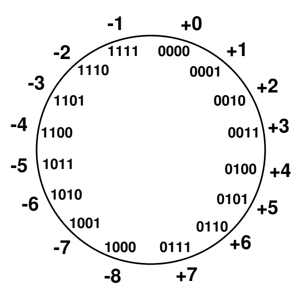
$N = 5_{10} = \mathbf{0101}_2$

$2^n = 2^4 = 10000$

$2^n - N = 10000 - 0101 = \mathbf{1011 = N^*}$

O complemento para 2 pode ser calculado obtendo o complemento para 1 e somando 1 ao resultado

Representação em complemento para dois



+
0 100 = + 4
1 100 = - 4
-

- O bit mais significativo também pode ser interpretado como sinal:
  0 = positivo 1 = negativo
- Uma única representações para 0
- Codificação assimétrica (mais um negativo do que positivos)

Representação em complemento para dois

Uma quantidade de 32 bits codificada em complemento para 2 pode ser representada pelo seguinte polinómio:

$$-(a_{31}.2^{31}) + (a_{30}.2^{30}) + \ldots + (a_1.2^1) + (a_0.2^0)$$

Onde o bit de sinal (a_{31}) é multiplicado por -2^{31} e os restantes pela versão positiva do respectivo peso

Exemplo: Qual o valor representado pela quantidade **10100101_2**, supondo uma representação com 8 bits e uma codificação em complemento para 2?

R1: $10100101_2 = (1\text{x}{-2^7}) + (1\text{x}2^5) + (1\text{x}2^2) + (1\text{x}2^0) = -128+32+4+1 = -91_{10}$

R2: Complemento para 2 de 10100101 = 01011010 + 1
= $01011011_2 = 5B_{16} = 91_{10}$. Ou seja, o valor representado é -91_{10}

Exemplos de adições e subtracções em complemento para 2

4	0100	-4	1100
+ 3	0011	+ (-3)	1101
7	0111	-7	11001
4	0100	-4	1100
- 3	1101	+ 3	0011
1	10001	-1	1111

Este esquema simples de adição com sinal torna o complemento para 2 o preferido para representação de inteiros em arquitectura de computadores

Arquitectura de Computadores I 2007/08

Cálculos em complemento para 2

Porque pode o *carry-out* ser ignorado?

-M + N quando **N > M:**

$$M^* + N = (2^n - M) + N = 2^n + (N - M)$$

Ignorar o *carry-out* é como subtrair 2^n

-M + -N onde N + M <= 2^{n-1}

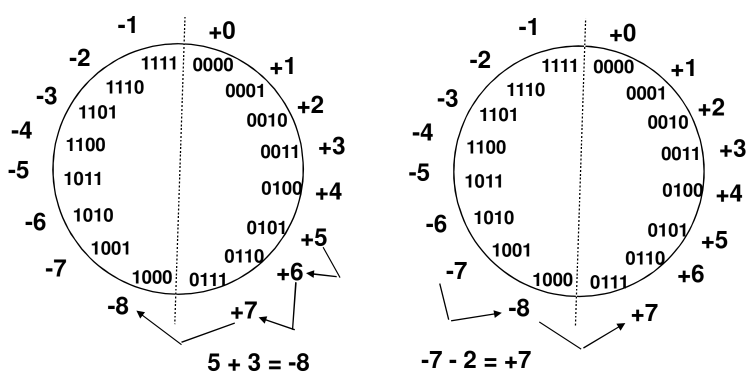
$$-M + (-N) = M^* + N^* = (2^n - M) + (2^n - N)$$
$$= 2^n - (M + N) + 2^n$$

Depois de ignorar o *carry*, estamos perante a representação correcta de -(M + N)!

Universidade de Aveiro Slide 11 - 21

Arquitectura de Computadores I 2007/08

Overflow em complemento para 2



Ocorre ***overflow*** quando somamos dois positivos e obtemos um negativo ou somamos dois negativos e obtemos um positivo

Universidade de Aveiro Slide 11 - 22

Overflow em complemento para 2

	0 0 0 0		1 1 1 1
5	0 1 0 1	-3	1 1 0 1
2	0 0 1 0	-5	1 0 1 1
7	0 1 1 1	-8	1 1 0 0 0
Sem overflow		Sem overflow	

	0 1 1 1		1 0 0 0
5	0 1 0 1	-7	1 0 0 1
3	0 0 1 1	-2	1 1 0 0
-8	0 1 0 0 0	7	1 0 1 1 1
Overflow		Overflow	

O Overflow ocorre quando o *carry-in* do bit de sinal não é igual ao *carry-out*

Overflow em operações aritméticas de adição:

- **Em operações sem sinal:**

  Quando **A+B** > **2^n -1** ou **A-B c/ B>A**

  A detecção dá-se quando o bit de *carry* C_n = 1

Como fazer a detecção de *overflow* em operações sem sinal no MIPS?

- **Em operações com sinal:**

  Quando **A + B** > **2^{n-1}-1** ou **A + B** < **-2^{n-1}**

  A detecção dá-se quando $\underbrace{\mathbf{C_{n-1}} = 1 \text{ e } \mathbf{C_n} = 0 \text{ ou } \mathbf{C_{n-1}} = 0 \text{ e } \mathbf{C_n} = 1}$

Ou seja, há *overflow* quando $C_{n-1} \oplus C_n = 1$