UNIVALI

Laboratório de Sistemas Embarcados e Distribuídos

Aritmética Computacional

Parte I

Histórico de revisões

Revisão	Data	Responsável	Descrição
0.1	03/2016	Prof. Cesar Zeferino	Primeira versão
0.2	04/2017	Prof. Cesar Zeferino	Atualização do modelo
0.3	03/2023	Prof. Felski	Atualização para RISC-V

Introdução

- **Objetivo**
  - Representar números inteiros e realizar operações aritméticas com números inteiros representados na forma binária
- **Conteúdo**
  - Números inteiros no RISC-V (sem sinal e com sinal)
  - Operações aritméticas de adição e subtração

Introdução

- **Bibliografia**
  - PATTERSON, David A.; HENNESSY, John L. Abstrações e tecnologias computacionais. *In*: _____. **Organização e projeto de computadores**: a interface hardware/software. 4. ed. Rio de Janeiro: Campus, 2014. cap. 3. Disponível em: <http://www.sciencedirect.com/science/article/pii/B9788535235852000032>. Acesso em: 25 abr. 2017.
  - Edições anteriores
    - Patterson & Hennessy (1998, p. 121-134)
    - Patterson & Hennessy (2005, p. 120-132)

2 Números com sinal e números sem sinal

- **Números inteiros sem sinal**
  - Palavra de dados do RISC-V: 32 bits / 64 bits
  - Valores representáveis:
    - $00000000000000000000000000000000_{dois} = 0_{dez}$
    - $00000000000000000000000000000001_{dois} = 1_{dez}$
    - $00000000000000000000000000000010_{dois} = 2_{dez}$

      . . . .
    - $11111111111111111111111111111101_{dois} = 4.294.967.293_{dez}$
    - $11111111111111111111111111111110_{dois} = 4.294.967.294_{dez}$
    - $11111111111111111111111111111111_{dois} = 4.294.967.295_{dez}$

Exemplo de uso de números sem sinal:
- Endereços de memória são sempre positivos

2 Números com sinal e números sem sinal

- **Números inteiros com sinal** (números representados em complemento de 2)
  - Palavra de dados do RISC-V : 32 bits / 64 bits
  - Valores representáveis:
    - $00000000000000000000000000000000_{dois} = 0_{dez}$
    - $00000000000000000000000000000001_{dois} = 1_{dez}$
    - $00000000000000000000000000000010_{dois} = 2_{dez}$

      . . . .
    - $01111111111111111111111111111101_{dois} = 2.147.483.645_{dez}$
    - $01111111111111111111111111111110_{dois} = 2.147.483.646_{dez}$
    - $01111111111111111111111111111111_{dois} = 2.147.483.647_{dez}$
    - $10000000000000000000000000000000_{dois} = -2.147.483.648_{dez}$
    - $10000000000000000000000000000001_{dois} = -2.147.483.647_{dez}$
    - $10000000000000000000000000000010_{dois} = -2.147.483.646_{dez}$

      . . . .
    - $11111111111111111111111111111101_{dois} = -3_{dez}$
    - $11111111111111111111111111111110_{dois} = -2_{dez}$
    - $11111111111111111111111111111111_{dois} = -1_{dez}$

2 Números com sinal e números sem sinal

- **Representação em complemento de 2**
  - Pode-se afirmar que $X + \overline{X} = -1$, pois

    00001111_{dois}

    $+ \underline{11110000}_{dois}$

    $11111111_{dois} = -1_{dez}$

A representação em complemento de 2 vem sendo usada em virtualmente todos os computadores desde 1965.

  - Logo, para obter o valor negativo de um dado número pode-se fazer:

    $-X = \overline{X}+1$

  - Exemplo: obtendo a representação do número -2_{dez}

    $2_{dez} = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{dois}$

    $2_{dez} = 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{dois}$

    $-2_{dez} = 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{dois}$

    $+\underline{0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001}_{dois}$

    $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{dois}$

2 Números com sinal e números sem sinal

Extensão de sinal

- A carga (*load*) de uma inteiro de 32 bits em um registrador de 32 bits é feita da mesma maneira independente do fato do número ser representado com ou sem sinal

- Porém, o RISC-V oferece a possibilidade de se carregar um inteiro de apenas 8 bits (um byte) com ou sem sinal
  - *load byte* **`lb`** com sinal
  - *load byte unsigned* **`lbu`** sem sinal

- Ao carregar um inteiro de 8 bits com sinal para um registrador de 32 bits o processador precisa preencher os 24 bits mais significativos com o valor do sinal
  - 00001010_{dois} => $00000000000000000000000000001010_{dois}$
  - 11111010_{dois} => $11111111111111111111111111111010_{dois}$

2 Números com sinal e números sem sinal

Overflow em números sem sinal

- Ocorre quando soma-se dois números sem sinal e o resultado extrapola a capacidade de representação, ou seja,
  o transporte de saída (*carry out*) do bit mais significativo é igual a 1.

- Exemplo

$$
\begin{array}{rl}
 & 1111111111111111111111111111111 \leftarrow \text{Bits de transporte} \\
 & 11111111111111111111111111111111_{dois} \\
+ & \underline{00000000000000000000000000000001_{dois}} \\
 & 00000000000000000000000000000000_{dois}
\end{array}
$$

2 Números com sinal e números sem sinal

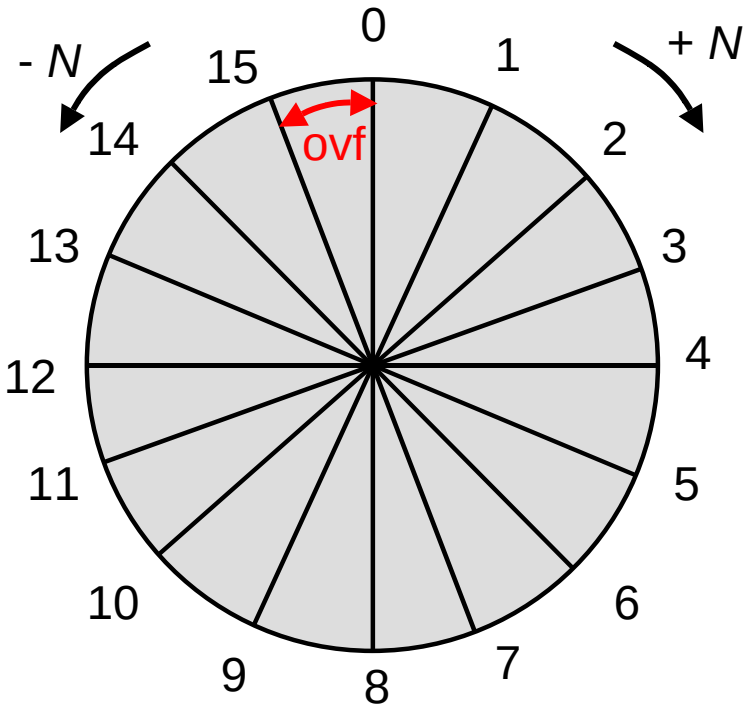
Overflow em números sem sinal

- Uma forma de entender o overflow consiste em usar um diagrama como o indicado abaixo. Ele apresenta todos os valores possíveis de se representar usando um número inteiro de 4 bits sem sinal (>0)

- Dado um número qualquer, somar um número *N* a esse número consiste em realizar *N* deslocamentos no sentido horário

- Já a subtração corresponde a um deslocamento no sentido anti-horário

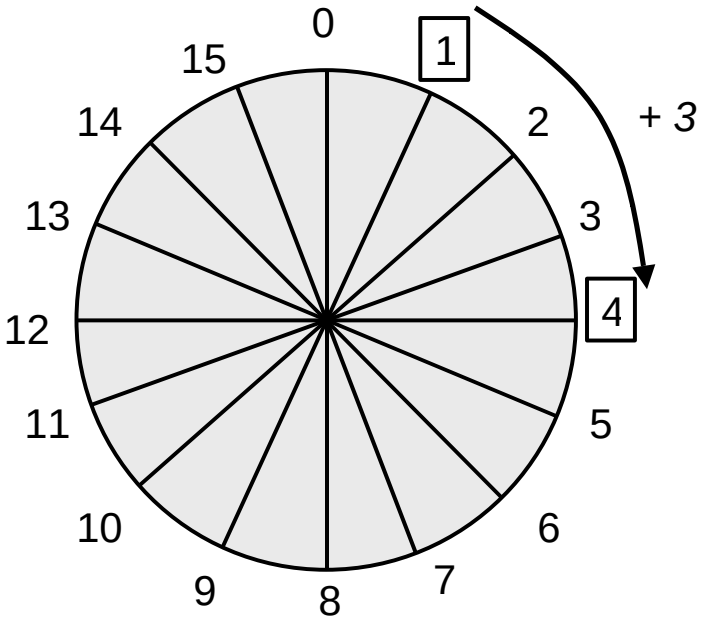
- Qualquer operação que produza um deslocamento do 0 para o 15 (ou vice-versa) produz um **overflow**



2 Números com sinal e números sem sinal

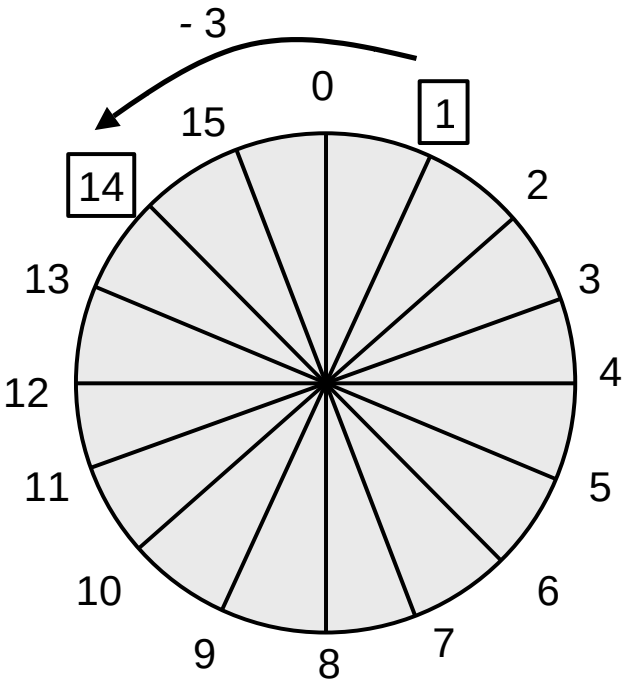
Overflow em números sem sinal

- Exemplos



1 + 3 = 4

sem overflow



1 - 3 = 14

com overflow

2 Números com sinal e números sem sinal

- **Overflow em números com sinal (representados em complemento de 2)**
  - Ocorre quando em uma operação de soma ou subtração o transporte de saída (carry out) do bit mais significativo é diferente do transporte de entrada do mesmo bit (ou seja o transporte de saída do bit anterior)
  - Exemplo

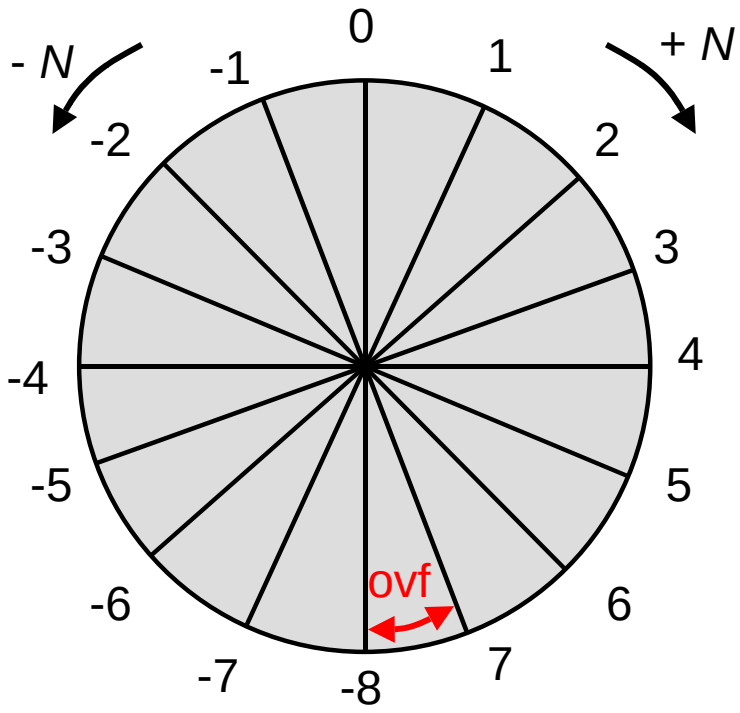
$$
\begin{array}{l}
01111111111111111111111111111111 \\
\quad 01111111111111111111111111111111_{dois} \\
+\ \underline{00000000000000000000000000000001_{dois}} \\
\quad 10000000000000000000000000000000_{dois}
\end{array}
$$

← Bits de transporte

A soma de dois números positivos resultou em um valor negativo

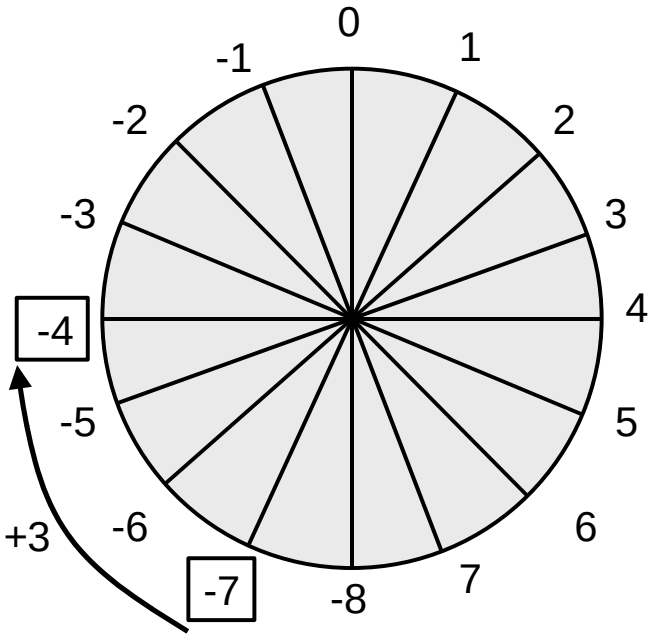
2 Números com sinal e números sem sinal

- **Overflow em números com sinal (representados em complemento de 2)**
  - No caso de números com sinal de 4 bits com sinal, o overflow ocorre quando uma operação produz um deslocamento do 7 para o –8 (e vice-versa), conforme ilustrado no diagrama



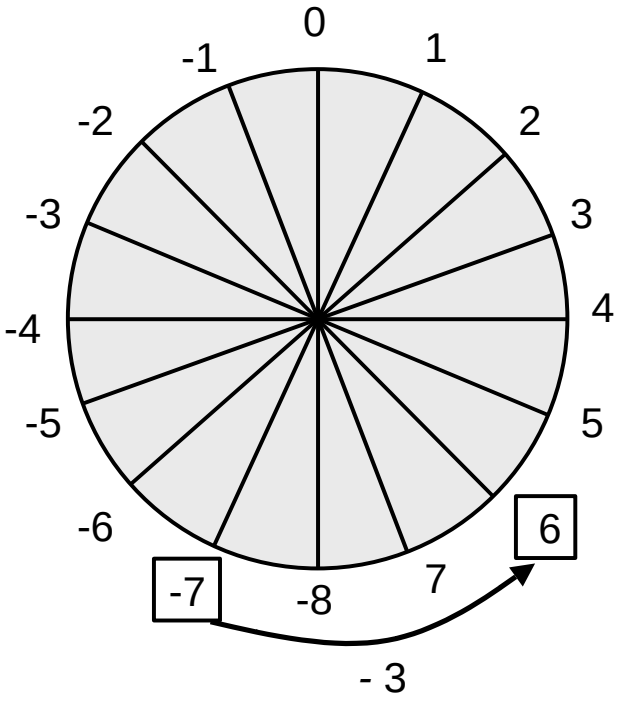
2 Números com sinal e números sem sinal

- **Overflow em números com sinal (representados em complemento de 2)**
  - Exemplos



-7 + 3 = -4

sem overflow



-7 - 3 = 6

com overflow

2 Números com sinal e números sem sinal

- **Instruções para comparação de inteiros com sinal (*signed int*)**
  - *set on less than* **slt**

    ```
    slt  rd, rs, rt
    ```

  - *set on less than immediate* **slti**

    ```
    slti rt, rs, imediato
    ```

- **Instruções para comparação de inteiros sem sinal (un*signed int*)**
  - *set on less than unsigned* **sltu**

    ```
    sltu  rd, rs, rt
    ```

  - *set on less than immediate unsigned* **sltiu**

    ```
    sltiu rt, rs, imediato
    ```

2 Números com sinal e números sem sinal

- **Exemplo de uso das instruções de comparação com e sem sinal**
  - Supondo que
    - `$s0` = $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{dois}$
    - `$s1` = $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{dois}$
  - Quais serão os valores armazenados em `$t0` e em `$t1` após a execução das duas instruções seguintes?
    - `slt  $t0, $s0, $s1  #comparação com sinal`
    - `sltu $t1, $s0, $s1  #comparação sem sinal`
  - Solução
    - `$t0` = 1_{dez} pois $-1_{dez} < 0_{dez}$
    - `$t1` = 0_{dez} pois $4.294.967.295_{dez} > 0_{dez}$

2 Números com sinal e números sem sinal

- **Regra prática para extensão de sinal**
  - Replicar o bit mais significativo do número original tantas vezes quantos forem os bits a serem preenchidos para se chegar ao tamanho de número desejado
  - Exemplos
    - 8 bits para 32 bits
      - 2_{dez} = 0000 0010$_{dois}$ => 0000 0000 0000 0000 0000 0000 0000 0010$_{dois}$
      - -2_{dez} = 1111 1110$_{dois}$ => 1111 1111 1111 1111 1111 1111 1111 1110$_{dois}$

3 Adição e subtração

- Condições para ocorrência de overflow nas operações de soma e subtração

Operação	Operando A	Operando B	Resultado
A + B	≥ 0	≥ 0	< 0
A + B	< 0	< 0	≥ 0
A – B	≥ 0	< 0	< 0
A – B	< 0	≥ 0	≥ 0

- Algumas instruções sinalizam a ocorrência de overflow: `add`, `addi` e `sub`

- Já as instruções que ignoram o sinal (unsigned – sem sinal), não indicam a ocorrência de overflow `addu`, `addiu` e `subu`

3 Adição e subtração

- **Exceção de overflow**
  - Diferente de arquiteturas como o MIPS/ARM, o RISC-V não tem bits de carry para detecção de overflow e underflow, isso ocorre porque essas flags complicariam implementações superescalares.

3 Adição e subtração

Exemplo de código que gera overflow

- Código assembly

```
.text
    addi $s0, $zero, 0x7FFFFFFF  # Maior inteiro sem sinal
    addi $s0, $s0, 1             # Soma um e gera overflow
```

- Código executado (com endereços)

```
0x00400000 : lui   $at, 0x00007FFF
0x00400004 : ori   $at, $at, 0x00000FFFF
0x00400008 : add   $s0, $zero, $at
0x0040000C : add   $s0, $s0, 0x00000001
```

As três primeiras instruções constroem uma constante de 32 bits (0x7FFFFFFF) e a carregam no registrador $s0 ($16)

A última instrução faz o incremento e gera o overflow

- Registradores do CP0 após overflow
  - `Status = 0x00000F13`
  - `Cause  = 0x00000030` (bits[6..2] = 01100) = OVF
  - `EPC    = 0x0400000C` (endereço da instrução que gerou o overflow)

3 Adição e subtração

- **Exemplo de código que não gera overflow**
  - Código assembly

```
.text
    addiu $s0, $zero, 0x7FFFFFFF  # Maior inteiro sem sinal
    addiu $s0, $s0, 1             # Soma um, mas não gera
                                    overflow
```

  - As instruções unsigned (`addiu`) não sinalizam a ocorrência de overflow