**Instituto Federal Sudeste de Minas Gerais – Campus Barbacena**
**Curso de Tecnologia em Sistemas para Internet**
**Disciplina: Lógica de Programação**
**Prof.: Wender Magno Cota**

**Terceira Atividade Avaliativa**

Como os computadores operam com elementos binários, a forma mais eficiente de representar números é utilizando o sistema binário, convertendo diretamente do formato decimal para o binário correspondente.

Ao lidar com números binários na computação, é fundamental considerar três aspectos que podem afetar o projeto e o funcionamento do sistema:

1. Representação do sinal do número
2. Representação da vírgula (ou ponto) que separa a parte inteira da parte fracionária
3. Limite de bits disponíveis para a representação numérica

### Representação do Sinal

Para indicar o sinal de um número, utiliza-se um bit adicional, geralmente o mais à esquerda. Por convenção:

- `0` representa um número **positivo**
- `1` representa um número **negativo**

### Representação da Parte Fracionária

Como os computadores não representam diretamente a vírgula decimal, utiliza-se uma das duas abordagens:

- **Ponto fixo**
- **Ponto flutuante**

### Limitações dos Sistemas Computacionais

Na matemática, os números são infinitos, mas computadores, por serem sistemas finitos, só conseguem representar uma quantidade limitada de valores.

## Tipos de Representação de Números Inteiros

1. Sinal e magnitude
2. Complemento à base - 1
3. Complemento à base (Complemento a 2)

### Sinal e Magnitude

Utiliza-se `n` bits, sendo o bit mais à esquerda para o **sinal** e os `n-1` bits restantes para a **magnitude**. A magnitude é igual para valores positivos e negativos.

**Faixa de representação:**

$-(2^{n-1} - 1)$ a $+(2^{n-1} - 1)$

**Exemplo com 10 bits:**

- `0000110110` = +54
- `1000110110` = -54

**Desvantagens:**

- Duas representações para o zero
- Requer dois circuitos somadores

## Aritmética com Complemento a 2

Essa representação é amplamente utilizada por sua eficiência, especialmente por permitir a utilização de um único circuito somador.

### Conceito de Complemento

Para a base B, o complemento à base de um número N com n dígitos é:

$B^n - N$

**Exemplo com base 2 e 5 bits:**

- N = `00110` (decimal 6)
- 25=322^5 = 3225=32
- Complemento de 2: `100000` - `00110` = `11010`

**Faixa de representação:**

$-2^{n-1}$ a $+(2^{n-1}-1)$

### Algoritmo de Adição com Complemento a 2

1. Somar os dois números, inclusive o bit de sinal.
2. Desprezar o "vai 1" final, se houver.
3. Se ambos os "vai 1" (para o bit de sinal e para fora do número) ocorrem ou ambos não ocorrem, o resultado é válido.
4. Se apenas um ocorre, há overflow.

**Nota:** Overflow só acontece se os dois operandos têm o mesmo sinal e o resultado tem sinal oposto.

### Algoritmo de Subtração com Complemento a 2

1. Obter o complemento de 2 do subtraendo.
2. Somar ao minuendo, como em uma operação de adição.

**Exemplos (4 bits):**

a) Somar `1100` com `1101` (valores negativos):

- `1100` = -4
- `1101` = -3

- Soma: `1100 + 1101` = `10001` (despreza o vai 1 → resultado: `0001`, que representa -7)

b) Somar 13 + 15 (com 6 bits):

- 13 = `001101`
- 15 = `001111`
- Soma: `011100` (28)

c) Somar 23 + 20:

- 23 = `010111`
- 20 = `010100`
- Soma: `111001` → **Overflow detectado**

## Complemento à Base - 1

Este complemento é calculado por:

$(2^{n-1})-N$

**Exemplo com 5 bits:**

- Complemento de 1 de `01110` = `10001`

## Exercício Prático

Desenvolver um programa em C para somar ou subtrair números inteiros usando **complemento de 2**, com as seguintes características:

- Utilizar 16 bits para representar os números;
- Ler vários pares de números em decimal;
- Converter cada número para complemento de 2;
- Permitir ao usuário escolher entre soma ou subtração;
- Exibir o resultado em binário e decimal;
- Detectar overflow;
- Operações devem ser feitas diretamente em binário;
- Modularização adequada com uso correto de funções;
- Utilizar apenas funções da biblioteca padrão ANSI C;
- O programa deve compilar sem warnings.