Aritmética Binária

Universidade Federal do Ceará - Campus Quixadá

Pedro Botelho
pedro.botelho@ufc.br

19 de Setembro de 2025

Arquitetura de Computadores

Seção 1

Operações Aritméticas

Operações aritméticas básicas

- Adição.
- Subtração.
- Multiplicação.
- Divisão.

As operações aritméticas nos sistemas binário, octal, decimal e hexadecimal obedecem a regras similares.

Operação de adição

- A operação soma de dois números na base 2 é efetuada de modo semelhante à soma decimal.
- Deve ser levado em conta que só há dois algarismos disponíveis (0 e 1)
- Assim, podemos criar uma tabela com todas as possibilidades:

$$0 + 0 = 0 \quad 0 + 1 = 1$$
$$1 + 0 = 1 \quad 1 + 1 = 0, \text{ com "vai 1" ou } 10_2$$

- Estouro (maior algarismo é ultrapassado) resulta em **carry** (transporte) ou vai um.

Exemplo de adição

Exemplo : $10101_2 + 10111_2 =$

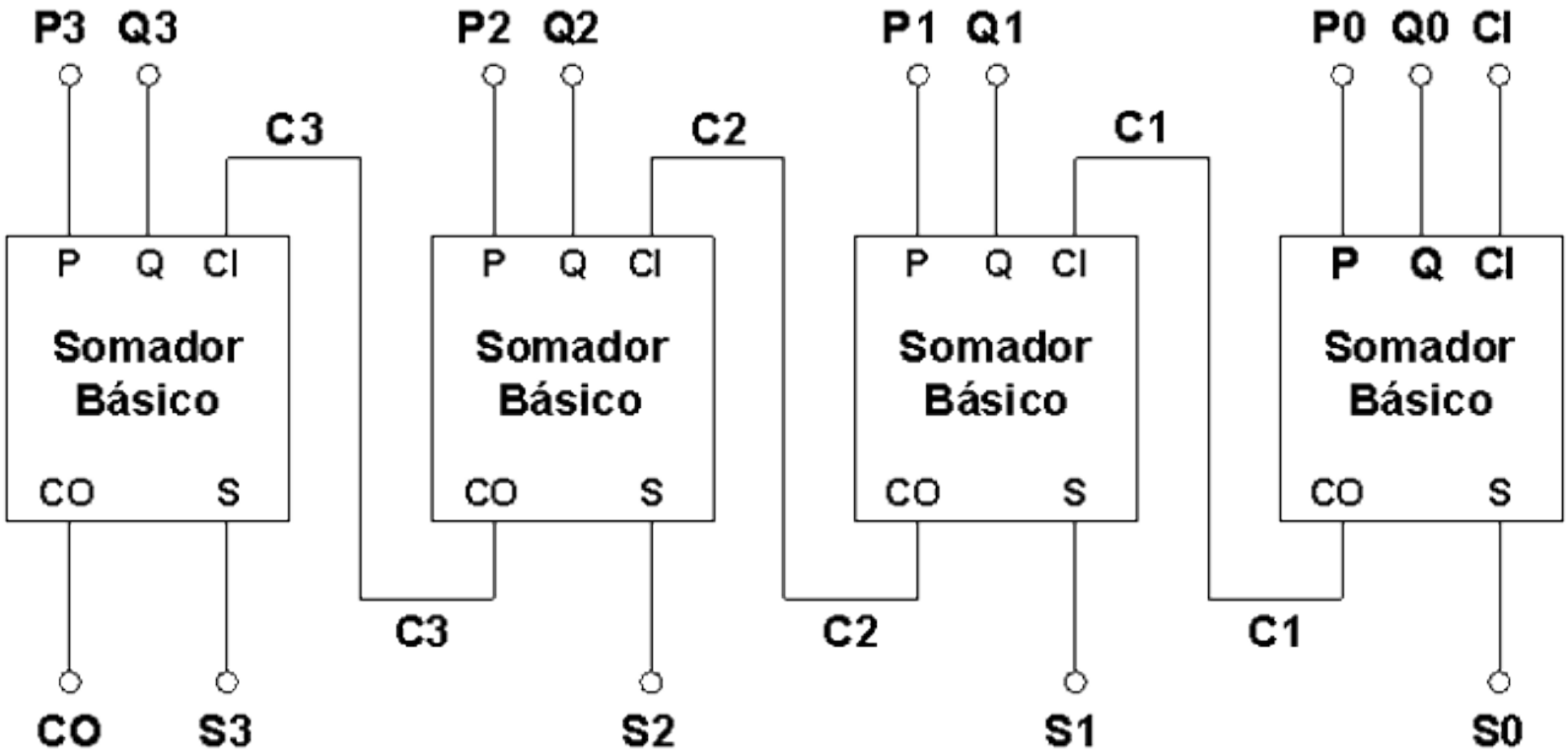
Solução:

$$\begin{array}{rr} & 10101_2 \\ + & 10111_2 \\ \hline & 101100_2 \end{array}$$

$$\begin{array}{rr} 0+0= & 0 \\ 0+1= & 1 \\ 1+0= & 1 \\ 1+1= & 10 \end{array}$$

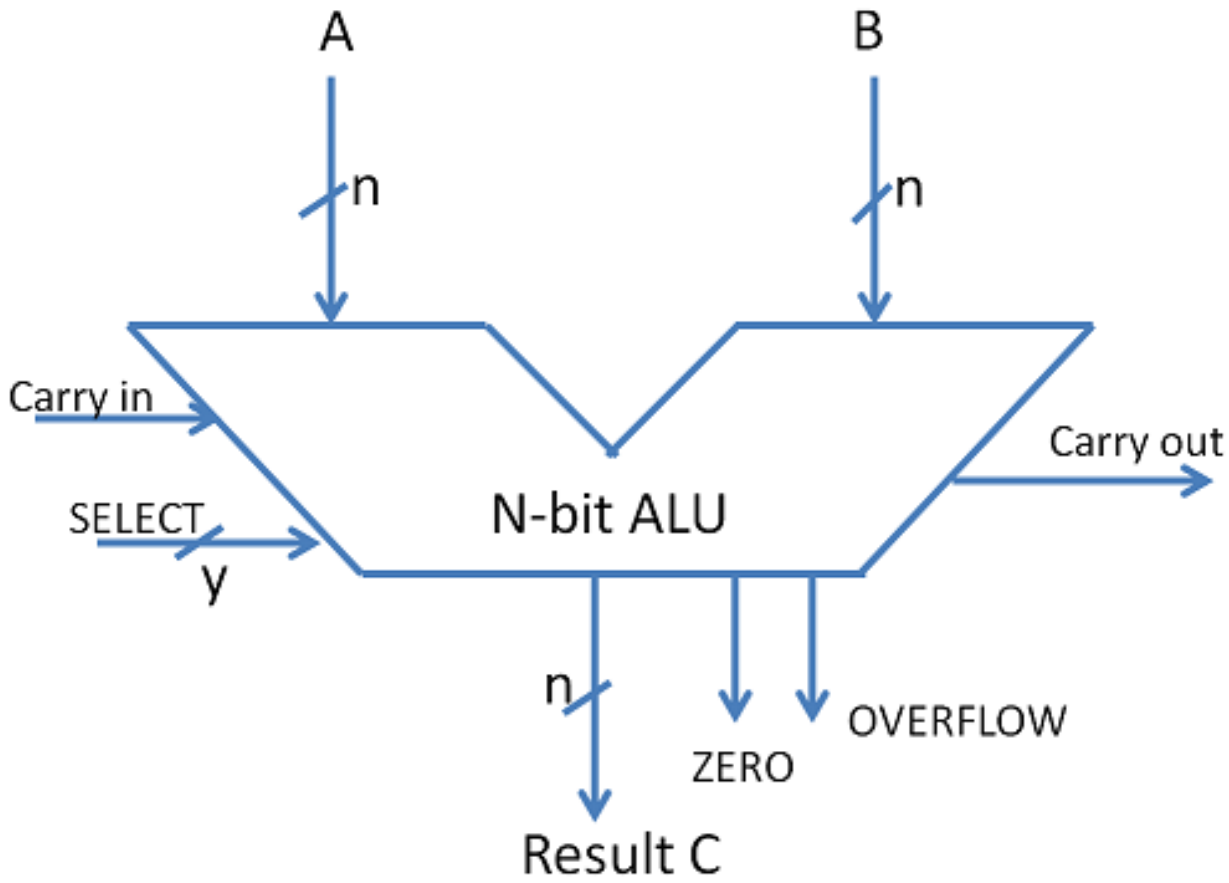
Circuito somador de 4 bits

Realiza a soma entre P e Q (de 4 bits) $\rightarrow$ Suporta subtração e sinalização



Unidade Lógica e Aritmética (ULA)

Realiza várias operações **lógicas** (AND, OR, XOR, NOT) e **aritméticas** (soma, subtração, multiplicação) entre A e B (de n bits)



Operação de subtração

- A subtração também é efetuada de forma semelhante à subtração decimal (minuendo - subtraendo = diferença).
- Como na adição, podemos criar uma tabela com todas as possibilidades:

$$0 - 0 = 0 \quad 0 - 1 = 1, \text{ e "vem 1 do próximo"}$$
$$1 - 0 = 1 \quad 1 - 1 = 0$$

- Como é impossível tirar 1 de 0, o artifício é "pedir emprestado" 1 da casa de ordem superior, ou seja, na realidade o que se faz é subtrair 1_2 de 10_2 e encontramos 1_2 como resultado, devendo então subtrair 1 do dígito de ordem superior.

Exemplo de subtração

$$
\begin{array}{r}
1\ 1^{0}1^{2}0\ 0 \\
-\ 0\ 1\ 0\ 1\ 0 \\
\hline
1\ 0\ 0\ 1\ 0
\end{array}
$$

Exemplos de subtração

exemplos:

- 10011_2 - $10010_2 = ?$
- 11100_2 - $00101_2 = ?$
- 11001_2 - $01101_2 = ?$
- 11110_2 - $11001_2 = ?$

Exemplos de subtração

exemplos:

- 10011_2 - $10010_2 = 00001_2$
- 11100_2 - $00101_2 = 10111_2$
- 11001_2 - $01101_2 = 01100_2$
- 11110_2 - $11001_2 = 00101_2$

Operação de multiplicação

- As regras para a multiplicação no sistema binário são exatamente as mesma do sistema decimal.
- Com uma vantagem, no binário temos apenas dois algoritmos.
- As regras são:

$$0 * 0 = 0 \quad 0 * 1 = 0$$
$$1 * 0 = 0 \quad 1 * 1 = 1$$

Exemplo de multiplicação

Exemplo: $1111_2 * 11110_2$, ou seja, $(15_{10} * 30_{10})$

30	11110
x15	x 1111
150	11110
+ 30	11110
450	11110
	+11110
	111000010

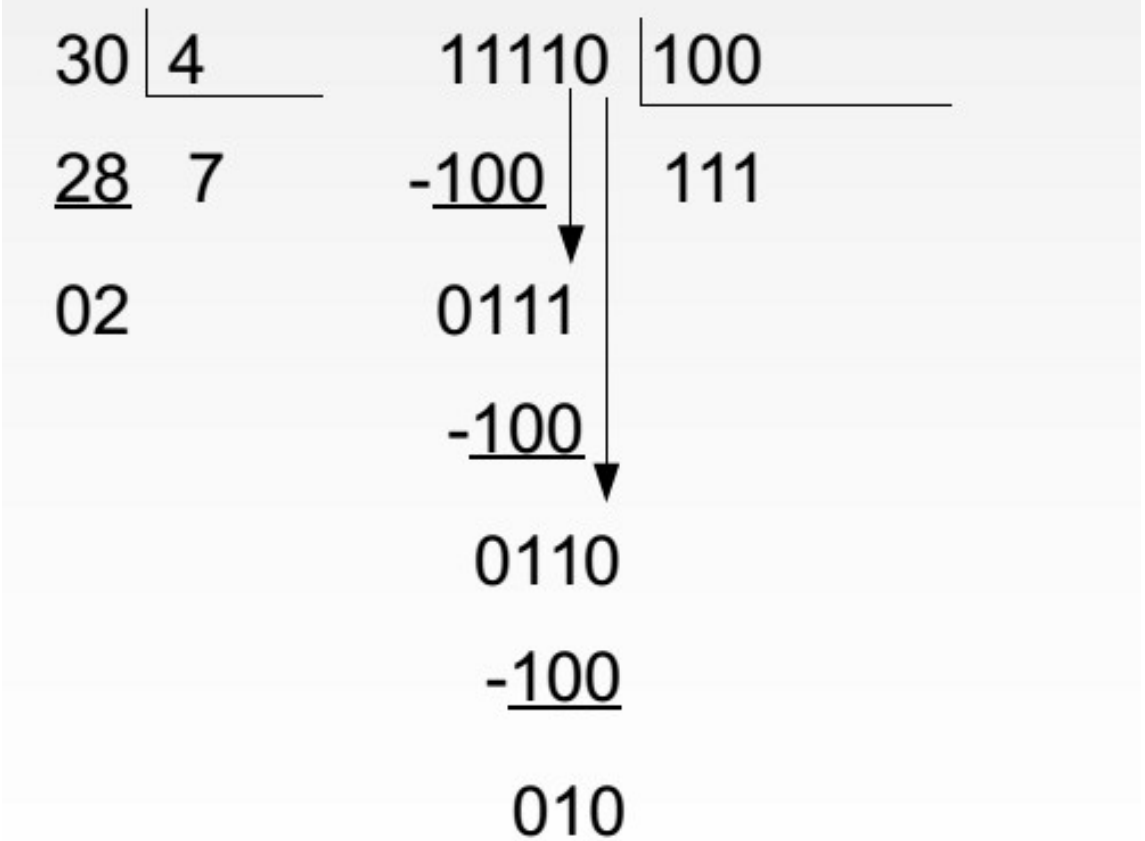
Operação de divisão

- A operação de divisão utiliza de forma conjunta as operações de multiplicação e subtração.
- O procedimento matemático para divisão com números binários é o mesmo com valores decimal.
- As regras da divisão são:

$$0/1 = 0 \quad 1/1 = 1$$

Exemplo de divisão

Exemplo: $11110_2 / 100_2$, ou seja, $(30_{10} / 4_{10})$



Seção 2

Representação Numérica

Representação dos números

- Como vimos em aulas anteriores, os computadores só conseguem representar zeros e uns.
- Os números positivos são armazenados em binário.
  - Ex.: $41 = 00101001$.
- Consequentemente, o computador não consegue armazenar sinais de **magnitude** e nem a **vírgula** dos números reais.

Representação dos números

- Naturais (já vimos).
- Inteiros (veremos em sequência).
- Vírgula (ou ponto) fixa (próxima aula).
- Vírgula (ou ponto) flutuante (próxima aula).
- Para todos, a quantidade de valores possíveis depende do número de bits (N).
  - 2^N valores.

Representação dos números

- Os números são infinitos.
- No computador eles são **finitos** $\rightarrow$ Limitados pela representação.
- O computador pode lidar com números até um certo tamanho.
- Representar números maiores que o possível resulta em *overflow*.
- No computador é preciso representar números com sinal.
  - Uma solução é usar um bit para representar o sinal de magnitude.
- Uma primeira tentativa é usar o **bit mais significativo** (MSB) para representar o sinal.

Limitação dos números inteiros

- Positivos: Mesmo valor, limitado ao número de bits.
  - Exemplo: $6_{10} = 110_2$
- Negativos: limitado pelo número de bits e pode ser representado de diferentes formas:
  - Módulo e sinal.
  - Complemento de 1 (C-1)
  - Complemento de 2 (C-2)

Módulo e sinal (MS)

- O bit mais à esquerda representa o sinal
  - O valor 0: sinal +; o valor 1: sinal -
- Os (N-1)bits restantes são o módulo do número.
- Quantidade: $-2^{N-1}+1 \leq X \leq 2^{N-1}-1$.
  - N = 8, $-127 \leq X \leq 127$
- Exemplo para N=8:
  - $00101010_2 = +42_{10}$
  - $10101010_2 = -42_{10}$

Complemento de 1 (C-1)

- O bit mais à esquerda representa o sinal
  - O valor 0: sinal +; o valor 1: sinal -
- Os (N-1)bits restantes são o módulo do número.
- Quantidade: $-2^{N-1}+1 \leq X \leq 2^{N-1}-1$.
- O complemento de 1 de um número binário é determinado pela troca de todos os 1s por 0s e vice-versa.
- Exemplo para N=8:
  - $00101010_2 = +42_{10}$
  - $11010101_2 = -42_{10}$

Complemento de 2 (C-2)

- O bit mais à esquerda representa o sinal
  - O valor 0: sinal +; o valor 1: sinal -
- Os (N-1)bits restantes são o módulo do número.
- Quantidade: $-2^{N-1} \leq X \leq 2^{N-1} - 1$.
- O complemento de dois é simétrico em dois passos:
  - Passo 1: Calsula-se C-1.
  - Passo 2: Soma-se 1 ao resultado do passo 1.
  - **Obs.:** O *overflow* é desprezado, caso exista.
- Exemplo para N=8:
  - $00101010_2 = +42_{10}$
  - $11010110_2 = -42_{10}$

Intervalo de valores

- **Intervalo** → Faixa de Valores Representáveis
  - Complemento de Dois → -2^{n-1} a $2^{n-1}-1$
  - Exemplo: 4 bits -8_{10} a 7_{10}→ 1111_2 a 0111_2
- Expansão de Bits → Armazenar n bits em m bits, m > n
  - **Extensão de Sinal** → Acrescentar o bit de sinal a esquerda
  - **Extensão de Zeros** → Acrescentar zeros a esquerda
- Exemplos de Sinal-Magnitude:
  - $+18_{10} = 00010010_2$ (sinal-magnitude, 8 bits)
  - $+18_{10} = 0000000000010010_2$ (sinal-magnitude, 16 bits)
  - -$18_{10} = 10010010_2$ (sinal-magnitude, 8 bits)
  - -$18_{10} = 1000000000010010_2$ (sinal-magnitude, 16 bits)
- Exemplos de Complemento de Dois:
  - $+18_{10} = 00010010_2$ (complemento de dois, 8 bits)
  - $+18_{10} = 0000000000010010_2$ (complemento de dois, 16 bits)
  - -$18_{10} = 11101110_2$ (complemento de dois, 8 bits)
  - -$32.658_{10} = 1000000001101110_2$ (complemento de dois, 16 bits)
  - -$18_{10} = 1111111111101110_2$ (complemento de dois, 16 bits)

Exemplos

1. Determine o complemento de 1 e cada número binário a seguir:
   - $00011010_2 =$
   - $11110111_2 =$
   - $10001101_2 =$

2. Determine o complemento de 2 e cada número binário a seguir:
   - $00010110_2 =$
   - $11111100_2 =$
   - $10010001_2 =$

Exemplos

1. Determine o complemento de 1 e cada número binário a seguir:
   - $00011010_2 = 11100101_2$
   - $11110111_2 = 00001000_2$
   - $10001101_2 = 01110010_2$

2. Determine o complemento de 2 e cada número binário a seguir:
   - $00010110_2 = 11101010_2$
   - $11111100_2 = 00000100_2$
   - $10010001_2 = 01101111_2$

Seção 3

Aritmética em Sinal-Magnitude

Aritmética em sinal de magnitude

Algoritmo para operação aritmética de adição:

- Verificam-se os sinais dos números e efetua-se uma comparação entre eles.
- Se ambos os números têm o mesmo sinal, somam-se as magnitudes; o sinal do resultado é o mesmo das parcelas.
- Se os números têm sinais diferentes:
  - identifica-se a maior das magnitudes e registra-se o seu sinal;
  - subtrai-se a magnitude menor da maior (apenas as magnitudes);
  - sinal do resultado é igual ao sinal da maior magnitude.

Solução

a) $(+13)_{10} + (+12)_{10}$

+13	001101
+12	001100
+25	**011001**

b) $(+18)_{10} + (-11)_{10}$

+18	010010
-11	101011
+ 7	**000111**

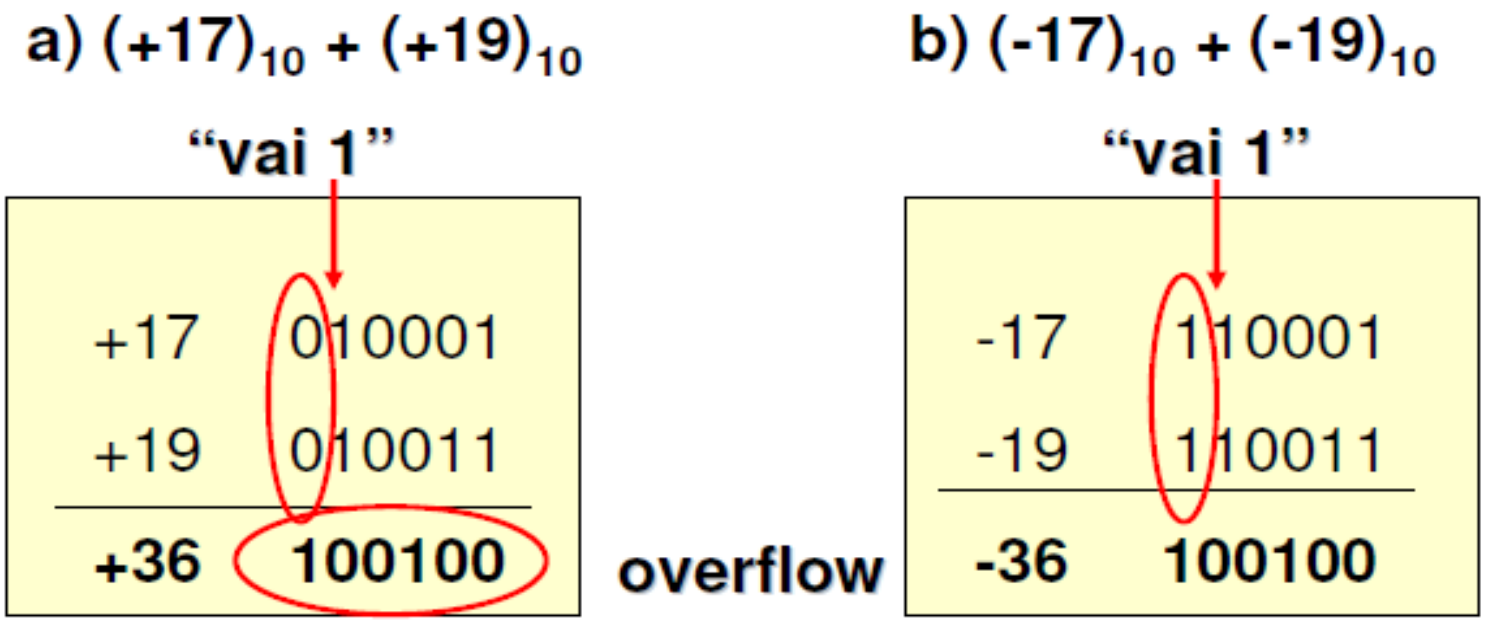
c) $(-21)_{10} + (+10)_{10}$

-21	110101
+10	001010
- 11	**101011**

d) $(-17)_{10} + (-9)_{10}$

-17	110001
-9	101001
-26	**111010**

Solução



Estouro (***overflow***) - existência de um "vai 1" para o bit de sinal.

Faixa de representação de valores para 6 bits (em sinal e magnitude) $\Rightarrow$ -31 a + 31.

Aritmética em sinal de magnitude

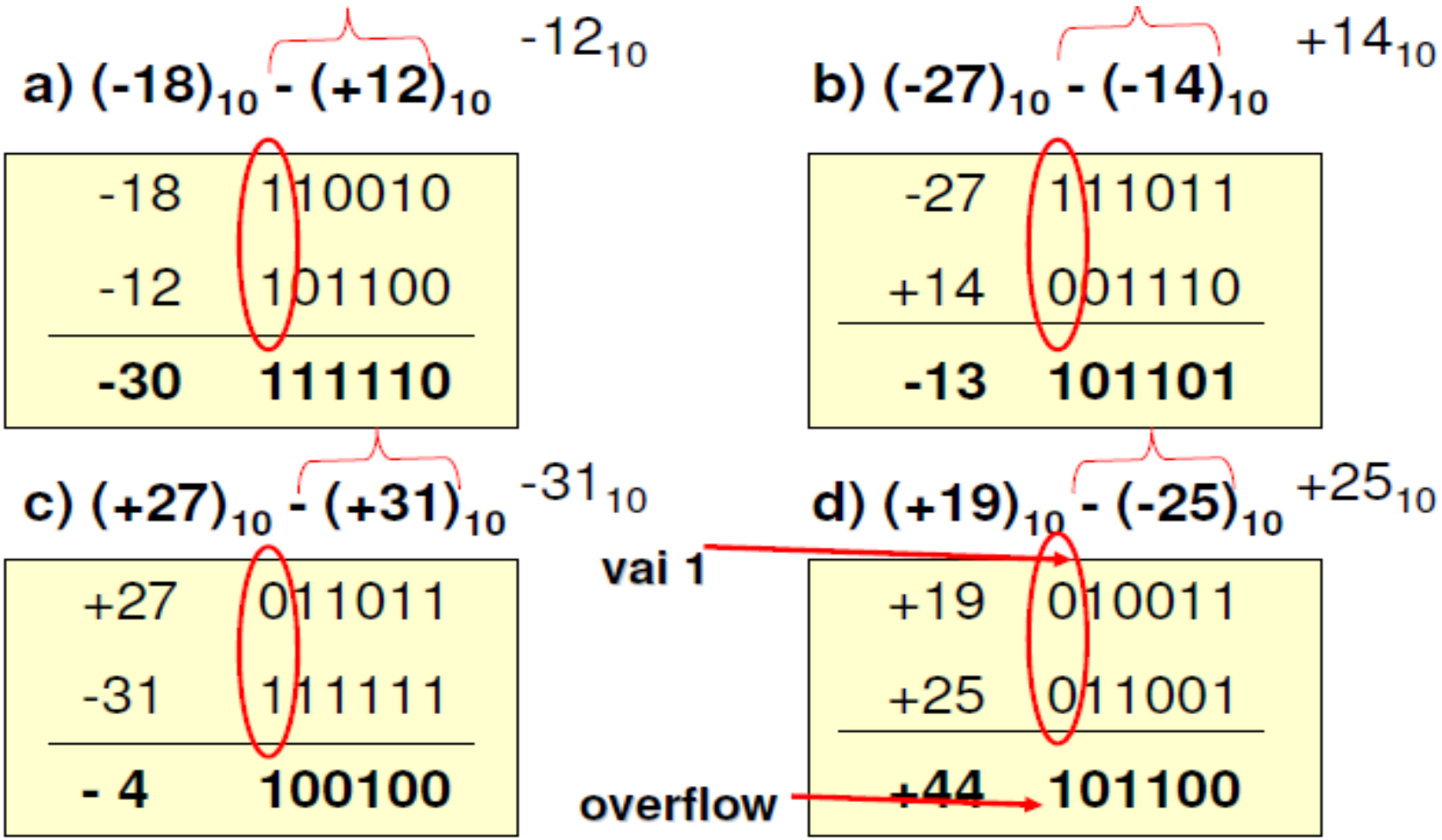
Subtração (Minuendo - Subtraendo = Resultado)

- Algoritmo para operação aritmética de subtração:
  - Troca-se o sinal do subtraendo.
  - Procede-se como no algoritmo da adição.

Exemplos

1. Exemplo: realize as operações aritméticas a seguir (em sinal e magnitude). Considere a palavra de dados com 6 bits.
   - a. $(-18)_{10} - (+12)_{10}$
   - b. $(-27)_{10} - (-14)_{10}$
   - c. $(+27)_{10} - (+31)_{10}$
   - d. $(+19)_{10} - (-25)_{10}$

Solução


a) (-18)10 - (+12)10
-12 10
-18 110010
-12 101100
-30 111110
b) (-27)10 - (-14)10
+14 10
-27 111011
+14 001110
-13 101101
c) (+27)10 - (+31)10
-31 10
+27 011011
-31 111111
- 4 100100
d) (+19)10 - (-25)10
+25 10
vai 1
+19 010011
+25 011001
+44 101100
overflow

Aritmética em sinal de magnitude

Desvantagens de efetuar operações aritméticas com valores representados em sinal e magnitude :

- Custo - necessidade de construção de dois elementos, um para efetuar somas e outro para efetuar subtração (dois componentes eletrônicos).
- Velocidade - ocasionada pela perda de tempo gasto na manipulação dos sinais, de modo a determinar o tipo de operação e o sinal do resultado.
- Outro inconveniente: dupla representação para o zero , o que requer um circuito lógico específico para evitar erros de má interpretação.

Seção 4

Aritmética em Complemento de 2

Algoritmo da adição em C-2

- Somar os dois números, bit a bit, inclusive o bit de sinal.
- Desprezar o último "vai " (para fora do número), se houver.
- Se, simultaneamente, ocorrer "vai 1" para o bit de sinal e "vai 1" para fora do número, ou se ambos não ocorrerem, o resultado está **correto**.
- Se ocorrer apenas um dos dois "vai 1", ou para o bit de sinal (*overflow*), ou para fora, (*carry-out*), ao final, o resultado está **incorreto**.
  - O *overflow*/*carry-out* somente pode ocorrer se ambos os números tiverem o mesmo sinal e, nesse caso, se o sinal do resultado for **oposto** ao dos números.

Algoritmo de subtração em C-2

- Aplica o complemento de 2 no **subtraendo**, independentemente se é um valor positivo ou negativo.
- E então, somam-se os números utilizando o algoritmo de adição já mostrado anteriormente.

Exemplos

1. Exemplo: realize as operações aritméticas a seguir (em complemento de 2). Considere a palavra de dados com 6 bits.
   - a. $(+13)_{10} + (+15)_{10}$
   - b. $(+23)_{10} + (+20)_{10}$
   - c. $(+15)_{10} + (-13)_{10}$
   - d. $(+20)_{10} - (+17)_{10}$
   - e. $(-24)_{10} - (-15)_{10}$
   - f. $(-24)_{10} - (+15)_{10}$

Solução (1/3)

a) $(+13)_{10} + (+15)_{10}$

	001111
+13	001101
+15	001111
+28	**011100**

Resultado correto - não houve "vai 1" nem para o bit de sinal nem para fora do número.

b) $(+23)_{10} + (+20)_{10}$

	010100
+23	010111
+20	010100
+43	**101011**

Resultado incorreto - houve "vai 1" apenas para o bit de sinal.

Overflow - faixa de representação para 6 bits (-32 a 31)

Solução (2/3)

c) $(+15)_{10} + (-13)_{10}$

	111111
+15	001111
-13	110011
+2	**000010**

d) $(+20)_{10} - (+17)_{10}$

	1**1**1100
+20	010100
-17	101111
+3	**000011**

Resultados corretos - houve "vai 1" para o bit de sinal e para fora do número; este é desprezado.

Solução (3/3)

e) $(-24)_{10} - (-15)_{10}$

	001000
-24	101000
+15	001111
-9	**110111**

Resultado correto - não houve "vai 1" para o bit de sinal nem para fora do número.

f) $(-24)_{10} - (+15)_{10}$

	100000
-24	101000
-15	110001
-39	**011001**

Resultado incorreto - houve "vai 1" apenas para fora do número.

Overflow - faixa de representação para 6 bits (-32 a 31)

Resumo da aritmética em C-2

Resumindo, é importante lembrar que:

- As operações de adição e subtração são normalmente realizadas como **adição** (mesmo *hardware*).
- As operações de subtração são realizadas como **soma de complemento** (minuendo mais o complemento do subtraendo).
- Se o resultado encontrado é um valor **positivo**:
  - ...o valor decimal correspondente da magnitude é obtido por pura conversão de base 2 para base 10.
- Se o resultado encontrado é um valor **negativo**:
  - ...deve-se primeiro converter esse valor para representação de sinal e magnitude (consistirá em trocar o valor dos bits da magnitude e somar 1 ao resultado) e, em seguida, converter a magnitude de base 2 para base 10.

Somador como elemento fundamental

- A aritmética em complemento de 2 requer apenas um componente (somador) para somar dois números e um componente que realize a operação de complementação.
- O algoritmo básico refere-se, então, à soma dos números, considerando-se que os números negativos estejam representados em complemento de 2; ele acusa, também, se o resultado ultrapassar a quantidade de bits representáveis na ULA (*overflow*).
- **Pode-se efetuar a multiplicação através de sucessivas somas e a divisão através de sucessivas subtrações (processo lento).**
  - *Software* para processadores sem instruções de multiplicação e divisão podem usar **funções** que as implementam e.g. ARM Cortex-A8

Outras operações aritméticas em C-2

Observação:

- A multiplicação em computadores pode ser feita por um artifício:
  - para multiplicar um número A por n, basta somar A com A, n vezes. Ex.: $4 \times 3 = 4 + 4 + 4$.
- A divisão também pode ser feita por subtrações sucessivas!
- O que concluímos?
- **As operações aritméticas podem ser realizadas em computadores apenas com somas.**

Exemplos

1. Exemplo: realize as operações aritméticas a seguir (em complemento de dois). Considere a palavra de dados com 6 bits.
   - a. $(+13)_{10} + (+12)_{10}$
   - b. $(+18)_{10} + (-11)_{10}$
   - c. $(-21)_{10} + (+10)_{10}$
   - d. $(-17)_{10} + (-9)_{10}$

Seção 5

Overflow Aritmético

O que vimos até agora?

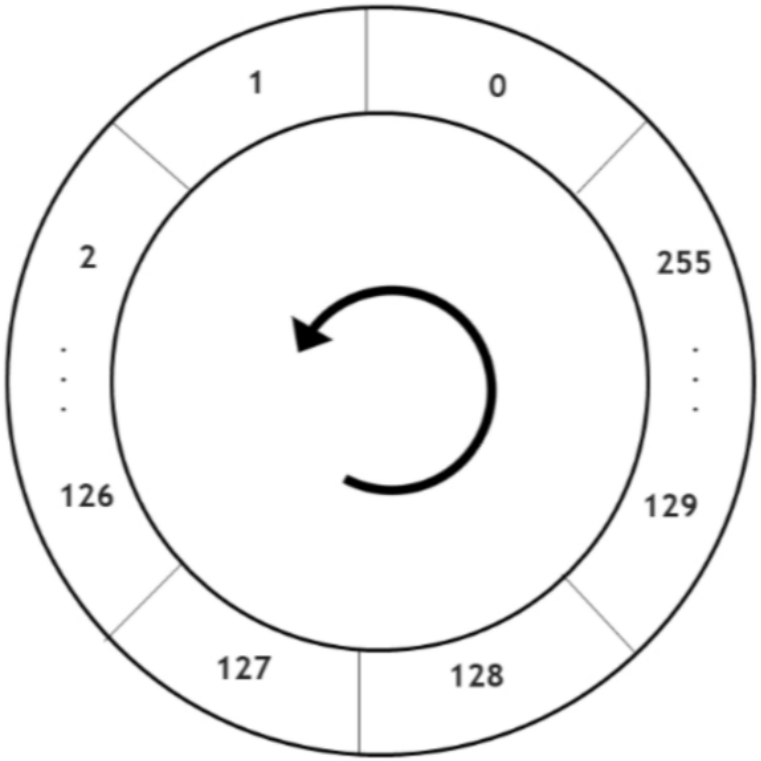
- **Adição** $\rightarrow$ Ocorre da mesma forma que a adição sem sinal
  - A diferença é que um dos dois números pode estar em C2
- **Subtração** $\rightarrow$ Utilizar complemento de dois no subtraendo e somar
  - Logo, para fazer A – B $\rightarrow$ A + (-B)
  - Facilita o projeto de *hardware*
  - Exemplo: 34_{10} - 19_{10}
- O que acontece se...
  - A + B = C, A e B com 4-bits e C com 5-bits (?)
  - (-A) + (-B) = C, A e B com 4-bits e C com 5-bits (?)

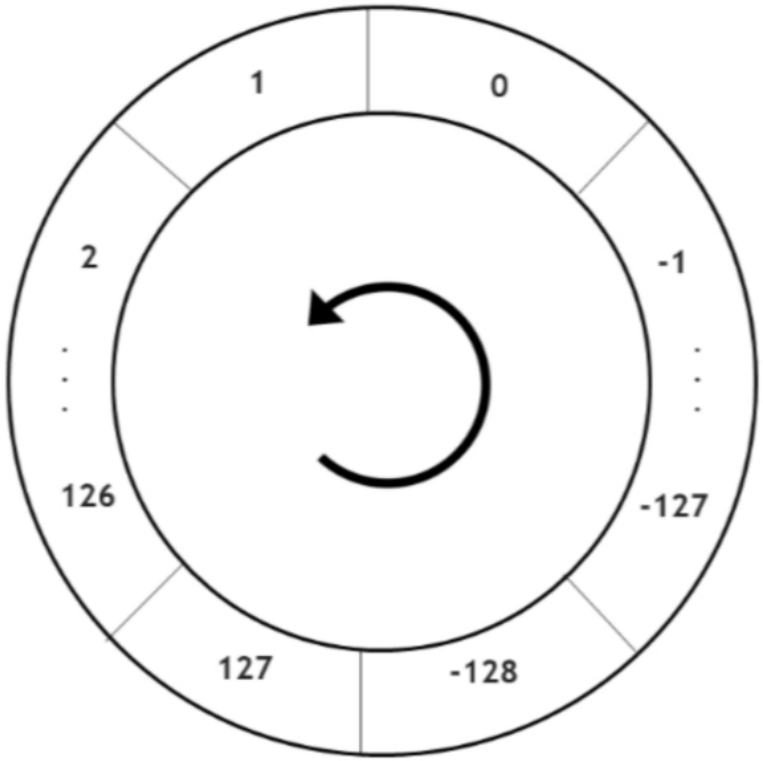
Overflow aritmético

- O que acontece se a magnitude invadir o espaço do sinal?
  - Ocorre o *overflow*!
  - Quando se "estoura" a quantidade de bits disponível
- *Overflow* $\rightarrow$ Resultado com sinal oposto ao dos operandos
  - $(+) + (+) = (-)$
  - $(-) + (-) = (+)$
- Overflow em números sem sinal (*unsigned*) $\rightarrow 1111_2 + 1_2 = 10000_2$
- Overflow em números com sinal (*signed*) $\rightarrow 0111_2 + 1_2 = 1000_2$
- Exemplos:
  - 4 bits em C-2 $\rightarrow 7_{10} + 7_{10}$
  - 8 bits em C-2 $\rightarrow 127_{10} + 1_{10}$

Faixa de valores circular

- Comparação da faixa de valores com sinal e sem sinal de 8-bit
- Perspectiva Circular → Duas Faixas de Valores
  - Orientações → Soma (normal) e Subtração (contrário)
  - Com sinal → $01111111_2 + 1_2 = 11111111_2 = \text{-}128_{10}$
  - Sem sinal → $11111111_2 + 1_2 = 00000000_2 = 0$





Exemplos

1. Exemplo: realize as operações aritméticas a seguir (em complemento de dois). Considere a palavra de dados com 6 bits.
   - a. $(+17)_{10} + (+19)_{10}$
   - b. $(-17)_{10} + (-19)_{10}$

Faixa de valores das variáveis em C

Tipo	Tamanho em Bytes	Faixa Mínima
`char`	1	-127 a 127
`unsigned char`	1	0 a 255
`signed char`	1	-127 a 127
`int`	4	-2.147.483.648 a 2.147.483.647
`unsigned int`	4	0 a 4.294.967.295
`signed int`	4	-2.147.483.648 a 2.147.483.647
`short int`	2	-32.768 a 32.767
`unsigned short int`	2	0 a 65.535
`signed short int`	2	-32.768 a 32.767
`long int`	4	-2.147.483.648 a 2.147.483.647
`signed long int`	4	-2.147.483.648 a 2.147.483.647
`unsigned long int`	4	0 a 4.294.967.295
`float`	4	Seis digitos de precisão
`double`	8	Dez digitos de precisão
`long double`	10	Dez digitos de precisão

Overflow em C

- Variáveis *signed* possuem menor faixa de valores que variáveis *unsigned*

```
signed char var1 = 127; // ok
signed char var2 = 128; // overflow
signed char var3 = 255; // overflow
unsigned char var4 = 255; // ok!
signed int var5 = 4000000000; // overflow
unsigned int var6 = 4000000000; // ok!
```

- *Overflow* (*signed*) e *carry-out* (*unsigned*) podem acontecer em operações aritméticas:

```
signed char res1 = var1 + 1; // overflow
unsigned char res1 = var4 + 1; // carry-out
```

Aritmética Binária

Universidade Federal do Ceará - Campus Quixadá

Pedro Botelho
pedro.botelho@ufc.br

19 de Setembro de 2025

Arquitetura de Computadores