PCS 3115

Sistemas Digitais I

Módulo 02 – Números negativos

versão: 4.0 (Jan/2019)

Conteúdo

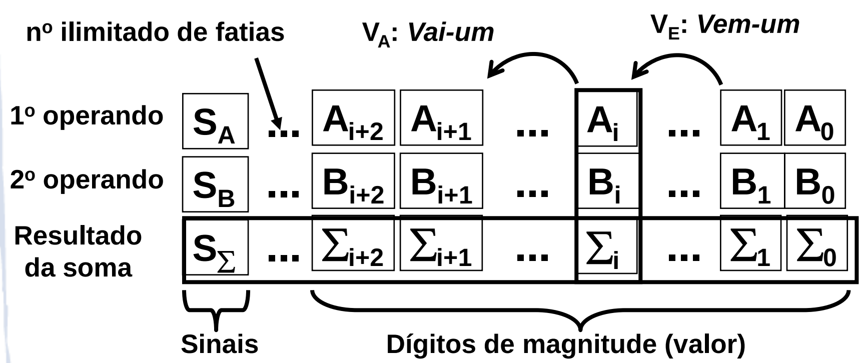
Aritmética Binária

- Soma e Subtração com Números Decimais e Binários
  - Aritmética Modular
- Representação de números negativos
  - Sinal e magnitude
  - Complemento de base ➔ complemento de 2
- Soma e Subtração com complemento de 2
  - Overflow

1. Soma e Subtração com Números Decimais e Binários

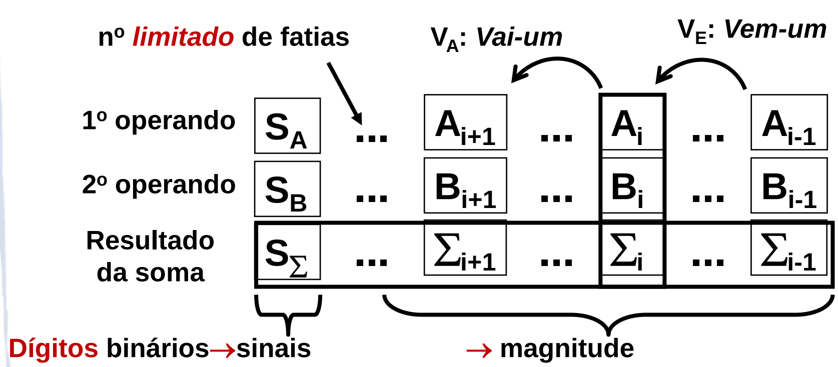
Operações com números decimais (*no papel*):

- O resultado da operação (ex.: **soma**) pode ser decidido calculando-se os resultados parciais em ***fatias*** individuais:



1. Soma e Subtração com Números Decimais e Binários

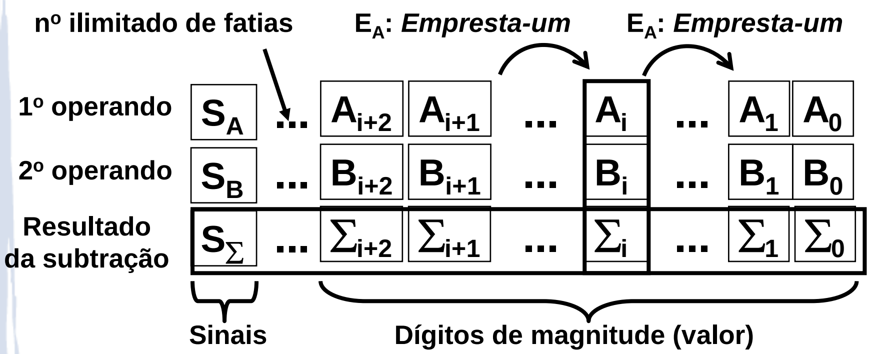
Operações com números binários (*no computador*): O resultado da operação (ex.: **soma**) ***também*** pode ser decidido calculando-se os resultados parciais em ***fatias*** individuais, ***porém*** ....:



1. Soma e Subtração com Números Decimais e Binários

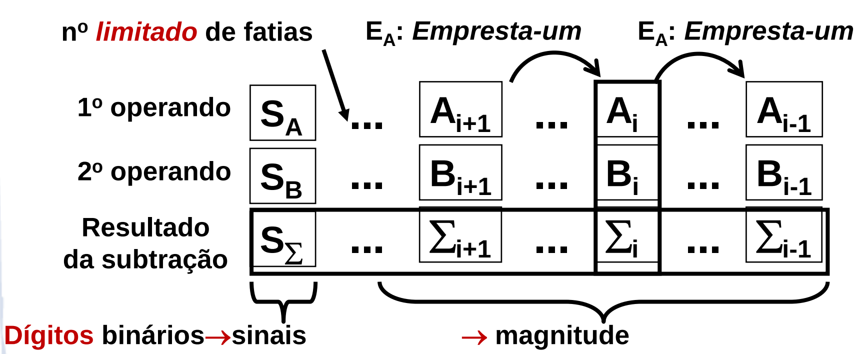
Operações com números decimais (*no papel*):

- O resultado da operação (ex.: **subtração**) pode ser decidido calculando-se os resultados parciais em ***fatias*** individuais:



1. Soma e Subtração com Números Decimais e Binários

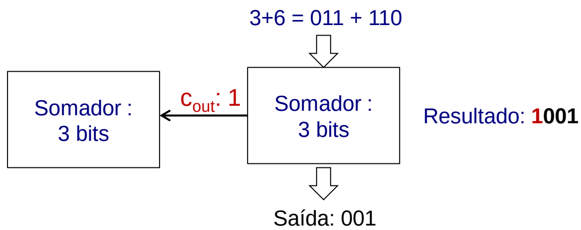
Operações com números binários (*no computador*): O resultado da operação (ex.: **subtração**) ***também*** pode ser decidido calculando-se os resultados parciais em ***fatias*** individuais, ***porém*** ....:



Desafio:

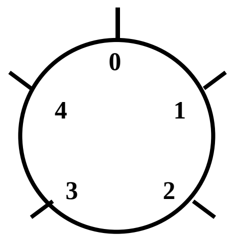
Desenvolver técnicas de **realização** de **operações**: elas fundamentalmente estarão **baseadas** na escolha do **tipo** de **representação** que se utilizará para os números binários, a fim de **resolver** os empecilhos e **problemas** das **operações de soma e subtração em binário**.

- **Problema 1: limitação no número de fatias**
  - Nem todos os números podem ser representados: existem **faixas de representações** possíveis
    - Ex.: 000 a 999 com 3 dígitos decimais;
    - Ex.: 00000000 a 11111111 com 8 dígitos binários (bits)
  - Podem ocorrer **overflows**: resultado de operação aritmética não cabe na representação
    - Ex.: 100 * 10 = 1000 (não representável com 3 dígitos decimais)
    - Problema deve ser tratado: geração de **carry**, ligado a outro somador ou a circuito que indica erro



- **Aritmética modular:**
  - Resultado natural de cenários com limitação do espaço de representação de números
  - Aritmética "módulo n": toma-se "o resto da divisão por n"

Aritmética Convencional
Dígitos = conjunto dos números Naturais = N
4 + 0 = 4
4 + 1 = 5
4 + 2 = 6
4 + 3 = 7
4 + 4 = 8
4 + 5 = 9
4 + 6 = 10
4 + 7 = 11
4 + 8 = 12
4 + ... = ...

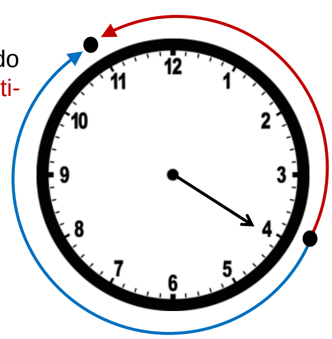


Aritmética modular (módulo 5)
Dígitos = {0,1,2,3,4}
4 + 0 = 4
4 + 1 = 0
4 + 2 = 1
4 + 3 = 2
4 + 4 = 3
(4 + 4) + 1 = 4
(4 + 4) + 2 = 0
(4 + 4) + 3 = 1
(4 + 4) + 4 = 2
((4 + 4) + 4) + ... = ...

- **Aritmética modular:**
  - Leva a equivalências entre algumas operações de adição e subtração: subtrair x é equivalente a somar (n – x)
    - Formalmente: (a – x) mod n = (a + n – x) mod n, pois (n mod n) = 0

Subtrair: movimento do ponteiro no sentido anti-horário

Somar: movimento do ponteiro no sentido horário



Subtrair 5 intervalos de 4 horas da tarde ...

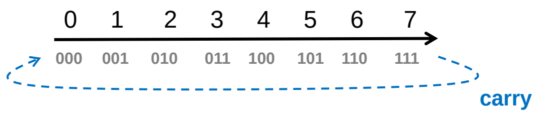
...é equivalente a **somar 7** intervalos de 4 horas da tarde

*Obs.: aritmética "**módulo 12**" (5+7 = **12**)*

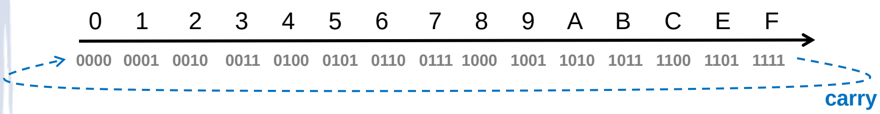
1. Soma e Subtração com Números Decimais e Binários

- **Aritmética modular:**
  - Módulos para soma binária operam com aritmética modular: somadores de n bits realizam somas módulo 2^n

Somador de 3 bits: somas módulo 8



Somador de 4 bits: somas módulo 16



1. Soma e Subtração com Números Decimais e Binários

- **Problema 1: limitação no número de fatias**
  - Nem todos os números podem ser representados: existem **faixas de representações** possíveis
  - Podem ocorrer **overflows**: resultado de operação aritmética não cabe na representação (gera-se carry)

- Problema 2: **números negativos**
  - Deve-se adotar alguma forma eficiente de **representá-los** e de fazer operações aritmética com eles
  - **Exercício:** propor uma solução para representação de números negativos em binário

2. Representação de números negativos

- Representação em **sinal e magnitude:**
  - 1 bit (mais significativo) para o sinal ➔ 0: positivo; 1: negativo
  - Bits restantes para a magnitude
  - Ex.: **01010101** = **+ 85** ; **11010101** = **– 85**
  - Faixa de representação com **n bits**: $[-(2^{n-1} - 1)\ ,\ +(2^{n-1} - 1)]$
- Simples para humanos entenderem, mas...
  - **Desperdício**: duas representações para o número zero
    - **00000000** = **+ 0** ; **10000000** = **– 0**
  - Operações de soma e subtração (= soma com inversão do sinal do segundo operando) **pouco eficientes** em hardware:
    - Ex. (decimal):

+ 123	– 123	+ 123	+ 123
+ 333	– 333	– 333	– 111
--------	-------	-------	-------
+456	– 456	– 210	+ 012

2. Representação de números negativos

- Representação em **sinal e magnitude:**
  - 1 bit (mais significativo) para o sinal ➔ 0: positivo; 1: negativo
  - Bits restantes para a magnitude
  - Ex.: **01010101** = **+ 85** ; **11010101** = **– 85**
  - Faixa de representação com **n bits**: $[-(2^{n-1} - 1)\ ,\ +(2^{n-1} - 1)]$
- Simples para humanos entenderem, mas...
  - **Desperdício**: duas representações para o número zero
    - **00000000** = **+ 0** ; **10000000** = **– 0**
  - Operações de soma e subtração (= soma com inversão do sinal do segundo operando) **pouco eficientes** em hardware:

Algoritmo da soma:
Comparar sinal dos operandos:
if(iguais) {mantém sinal}
else {**comparar** números e usar sinal do de maior magnitude}

Circuitos de lógica extras...
<u>É possível fazer melhor?</u>

2. Representação de números negativos

- Representação em **sinal e magnitude:**
  - Quando observada em binário, não é muito "natural": sinal inverte a sequência usual encontrada em números positivos

-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
1111	1110	1101	1100	1011	1010	1001	0000	0001	0010	0011	0100	0101	0110	0111
							1000							

- Algo um pouco mais "natural" (ainda com desperdício):

-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
1000	1001	1010	1011	1100	1101	1110	0000	0001	0010	0011	0100	0101	0110	0111
							1111							

"Complemento de 1": bits de x e –x são invertidos

- Algo mais "natural" e sem desperdício:

-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111

"Complemento de 2": complemento de 1, mais 1

2. Representação de números negativos

- Representação em **complemento de base:**
  - Aplica-se a ideia de aritmética modular: a representação de número negativo é dada pelo seu **complemento no espaço de valores possíveis**
- Mais formalmente:
  - Número D representado com n dígitos (notação posicional) :

$$D = d_{n-1}d_{n-2}...d_1d_0$$

  - Complemento na base r (do inglês, *radix*) do número D: obtido como $r^n – D$
    - Nota: r^n tem n+1 dígitos ➔ se D = 0, então exclui-se o dígito extra, de modo que 0 é representado simplesmente como n zeros
- Decimal: complemento de 10; binário: complemento de 2
  - Ex: Base r = 10 (decimal); n = 3 (3 dígitos); D = 345:
    - $r^n – D = 10^3 – 345 =$; $1000 – 345 = 655$;
    - Logo, o complemento na base 10 de 345 é 655!

2. Representação de números negativos

- Representação em **complemento de base:**
  - Faixa de representação na base **r**: $\left[-\left(\left\lfloor \frac{r^n}{2} \right\rfloor_{CHÃO}\right), +\left(\left\lceil \frac{r^n}{2} \right\rceil^{TETO} -1\right)\right]$
  - Faixa de representação com **n bits**: $[-(2^{n-1}), +(2^{n-1}-1)]$
    - Perceba que há **um número negativo a mais** do que os números positivos
- Para calcular "–D" a partir de "D" (ou vice-versa)
  - Sinal e magnitude: basta trocar sinal
    - Números binários: inverter bit mais significativo
  - Complemento de base: calcular $(r^n - D)$
    - Números binários (r = 2): calcular $(2^n - D)$

2. Representação de números negativos

- Representação em **complemento de base:**
  - É necessária uma operação de subtração para calcular os **complementos da base** de um número D?
  - Resposta: Não ➔ basta calcular o **complemento de cada dígito** naquela base e somar 1 ao resultado
- Explicação:

$$D = d_{n-1}d_{n-2}...d_1d_0, \text{ onde } 0 \le d_i \le r-1$$

$$\text{Complemento de D} = (r^n - D) = [(r^n - 1 + 1) - D] = [(r^n - 1) - D] + 1$$

  - Número $(r^n - 1)$ é da forma $\underbrace{mm...mm}_{\text{n vezes}}$, onde m = r – 1.
    - Exemplo: r = 10; n = 3 ➔ $r^n = 1000 = 999 + 1 = (r^n - 1) + 1$.
  - Logo: $[(r^n - 1) - D] + 1 = \left(\begin{array}{cccc} m & m & ... & m & m \\ d_{n-1} & d_{n-2} & ... & d_1 & d_0 \end{array} -\right) + 1$

2. Representação de números negativos

- Representação em **complemento de base:**

Dígito	Complemento Binário	Complemento Octal	Complemento Decimal	Complemento Hexa
0	1	7	9	F
1	0	6	8	E
2	-	5	7	D
3	-	4	6	C
4	-	3	5	B
5	-	2	4	A
6	-	1	3	9
7	-	0	2	8
8	-	-	1	7
9	-	-	0	6
A	-	-	-	5
B	-	-	-	4
C	-	-	-	3
D	-	-	-	2
E	-	-	-	1
F	-	-	-	0

- Exemplos
  - Comp(1849_{10}) = 8150 + 1
    = 8151_{10}
  - Comp($0F36_{16}$) = F0C9+ 1
    = $F0CA_{16}$
  - Comp(1010_2) = 0101 + 1
    = 0110_2

Complemento de 2: inverter bits e somar 1 ao resultado

2.1. Números binários: Complemento de 2

- **Números positivos**: idem a notação sinal-módulo;
- Para **inverter** o **sinal**:

  1) Invertem-se todos os bits (equivale a complementar de 1 cada um dos bits) e então

  2) Soma-se 1 ao resultado.

Exemplo: **+ 27_{10} = 0 0 0 1 1 0 1 1**

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ bits são complementados

1 1 1 0 0 1 0 0

+ 1 soma-se 1 ao resultado

-27_{10} = 1 1 1 0 0 1 0 1

2.1. Números binários: Complemento de 2

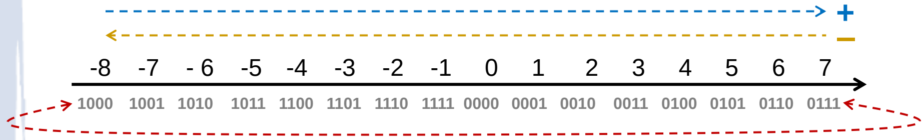
- **Extensão de sinal** (*sign extension*):
  - Ao aumentar o número de bits de D, deve-se tomar cuidado para manter o sinal correto!
  - Regra prática
    - Se D é **positivo**: adicionar **0s** à esquerda
    - Se D é **negativo**: adicionar **1s** à esquerda
- **Truncagem**
  - Diminuir o número de bits de D, cortando bits "sobrando" à esquerda: 0s se D é positivo; 1s se D é negativo
  - Resultado só é válido se o sinal do número se mantém

$+1_{10} = 1_2 = 00000001_2$ $\qquad$ $-8_{10} = 1000_2 = 11111000_2$

$-7_{10} = 1001_2 = 11111001_2$ $\qquad$ $-9_{10} = 10111_2 = 11110111_2$

3. Aritmética: Adição e Subtração

- **Complemento de 2**: contagem "natural"
  - **Adição** $n + m$: equivale a contar m a partir de n no sentido da esquerda para a **direita**
    - Ex.: $(-5 + 6)_{10} = 1011_2 + 0110_2$ = "1011_2 + 6 p/ direita" = $0001_2 = 1_{10}$
  - **Subtração** $n - m$: equivale a contar m a partir de n no sentido da direita para a **esquerda**
    - Ex.: $(4 - 2)_{10} = 0100_2 - 0010_2$ = "0100_2 + 2 p/ esquerda" = $0010_2 = 2_{10}$
  - **Overflow**: operação ultrapassa fronteira entre +7 e –8,
    - Ex.: $(6 + 4)_{10} = 0110_2 + 0100_2$ = "0110_2 + 4 p/ direita" = 1010_2 = **–6**



Overflow se cruzar fronteira em qualquer direção

3. Aritmética: Adição e Subtração

- **Complemento de 2**: **adição**
  - Pode-se usar as regras usuais da soma, ignorando o "vai-um" no bit mais significativo, se houver
- **Exemplos:**

+3	0011	-2	1110
+ +4	+ 0100	+ -6	+ 1010
+7	0111	-8	1 1000
+6	0110	+4	0100
+ -3	+ 1101	+ -7	+1001
+3	1 0011	-3	1101

Ignorar "vai-um"

3. Aritmética: Adição e Subtração

- **Complemento de 2**: **overflow na adição**
  - Nunca ocorre se sinais dos operandos são diferentes
  - **Detecção** de ocorrência: duas regras **equivalentes**
    1. Soma de dois números com o mesmo sinal produz resultado de sinal diferente
    2. "Vem-um" (c_{IN}) que chega na posição de sinal é diferente do "*vai-um*" (c_{OUT}) que sai da posição do sinal.

entradas			saídas: +	
c_{IN}	x	y	Σ (soma)	c_{OUT}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

regra 1

regra 2

3. Aritmética: Adição e Subtração

- **Complemento de 2**: **subtração**
  - Pode ser feita como se fosse uma adição, e depois verificam-se os sinais para detectar overflow
  - **Na prática**: nega-se subtraendo e faz-se soma normal, verificando overflow usando as regras da adição
    - Operação $m - n$ equivale a: 1) Complementar bit-a-bit n
      2) Somar m e n com $c_{IN} = 1$
- **Exemplos:**

		$c_{IN} = 1$			$c_{IN} = 1$
+4	0100	0100	+3	0011	0011
– +3	– 0011	+ 1100	– +4	– 0100	+ 1011
+1		**1** 0001	–1		1111

		$c_{IN} = 1$			$c_{IN} = 1$
+4	0100	0100	+3	0011	0011
– –3	– 1101	+ 0010	– –4	– 1100	+ 0011
+7		0111	+7		0111

3. Aritmética: Adição e Subtração

- Exercícios: Soma e subtração em **complemento de 2**

–3		1101		+5		0101	
+ –6		+ 1010	overflow	+ +6		+ 0110	overflow
–9		**1** 0111	+7	+11		1011	–5

		$c_{IN} = 1$				$c_{IN} = 1$	
+4	0100	0100		–3	1101	1101	
– +4	– 0100	+ 1011	sem overflow	– –4	– 1100	+ 0011	sem overflow
0		**1** 0000		+1		**1** 0001	

		$c_{IN} = 1$					
–4	1100	1100		–2		1110	
– +5	– 0101	+ 1010	overflow	+ –6		+ 1010	sem overflow
–9		**1** 0111	+7	–8		**1** 1000	

3. Aritmética: Adição e Subtração

- **Exercício 10.8.1.** Faça as operações com 6 bits (inclui o bit de sinal) em **Complemento de 2**. Indique a ocorrência de Transbordo:

a) + 19 + (–12)
b) – 19 + (–12)
c) + 19 + (+12)
d) – 19 + (+12)
e) + 21 + (–11)
f) – 21 + (–11)
g) + 21 + (+11)
h) – 21 + (+11)

Lição de Casa

- Material no Moodle
- Leitura:
  - Seções 2.5 e 2.6 do Livro Texto.
  - Wakerly, John F. Digital Design: Principles and Practices, Pearson Prentice-Hall, 4a Edição, 2006.