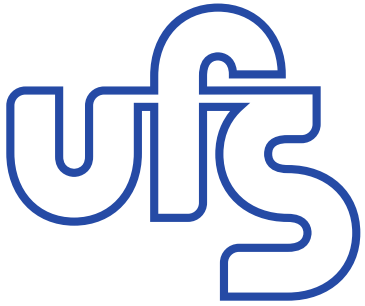
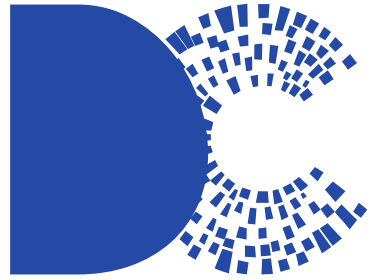




Aritmética binária

Arquitetura de Computadores

Bruno Prado

Departamento de Computação / UFS

Introdução

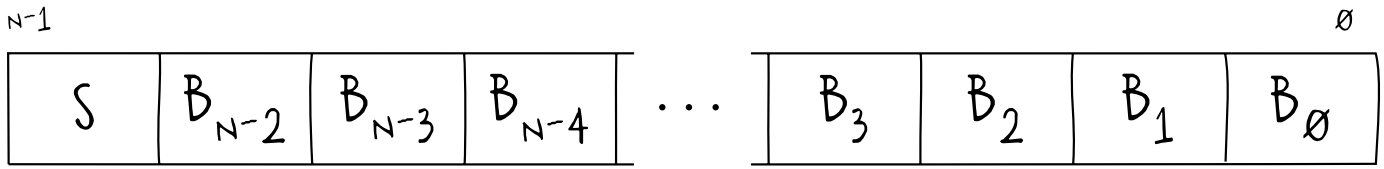
- Como pode ser feita a representação numérica?
  - Bases numéricas
    - 2 (binário)
    - 10 (decimal)
    - 16 (hexadecimal)

$$Número = \sum_{i=0}^{n-1} B_i \times N^i$$

$$\begin{aligned} 33_{10} &= 1 \times 2^5 + 1 \times 2^0 = 100001_2 \\ &= 3 \times 10^1 + 3 \times 10^0 = 33_{10} \\ &= 2 \times 16^1 + 1 \times 16^0 = 21_{16} \end{aligned}$$

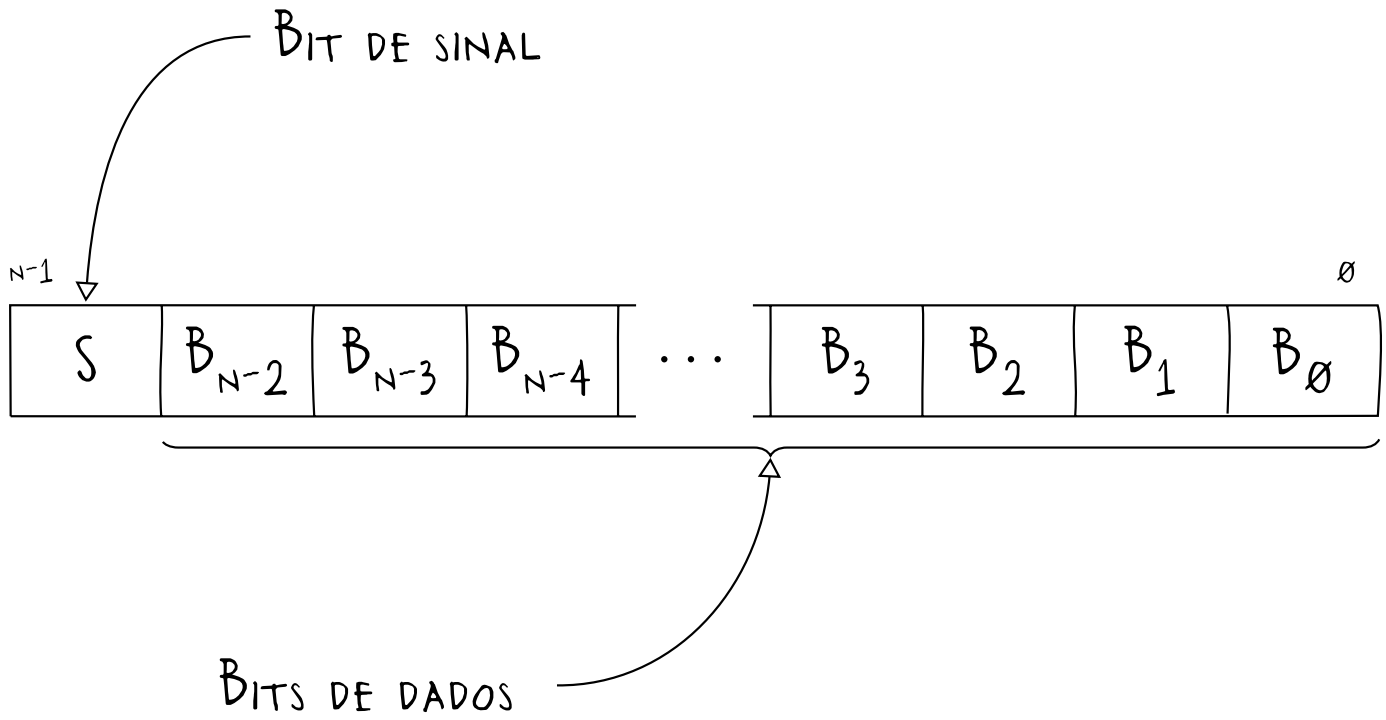
Introdução

- Como o sinal dos números binários é implementado?



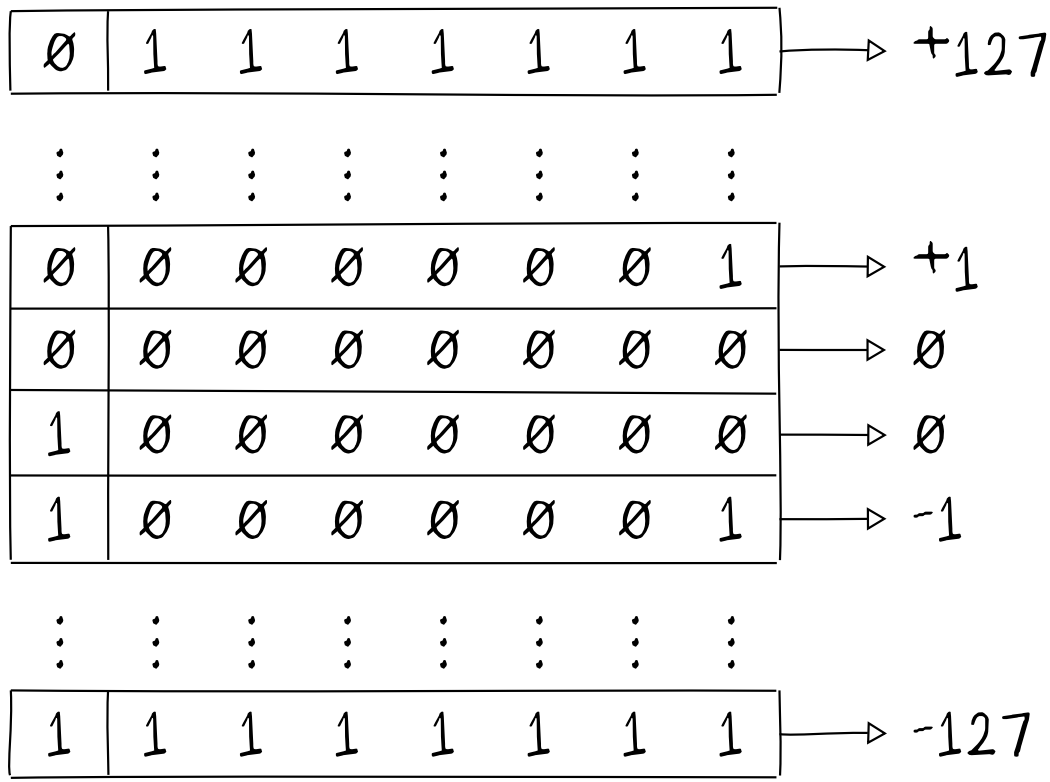
Introdução

- Como o sinal dos números binários é implementado?



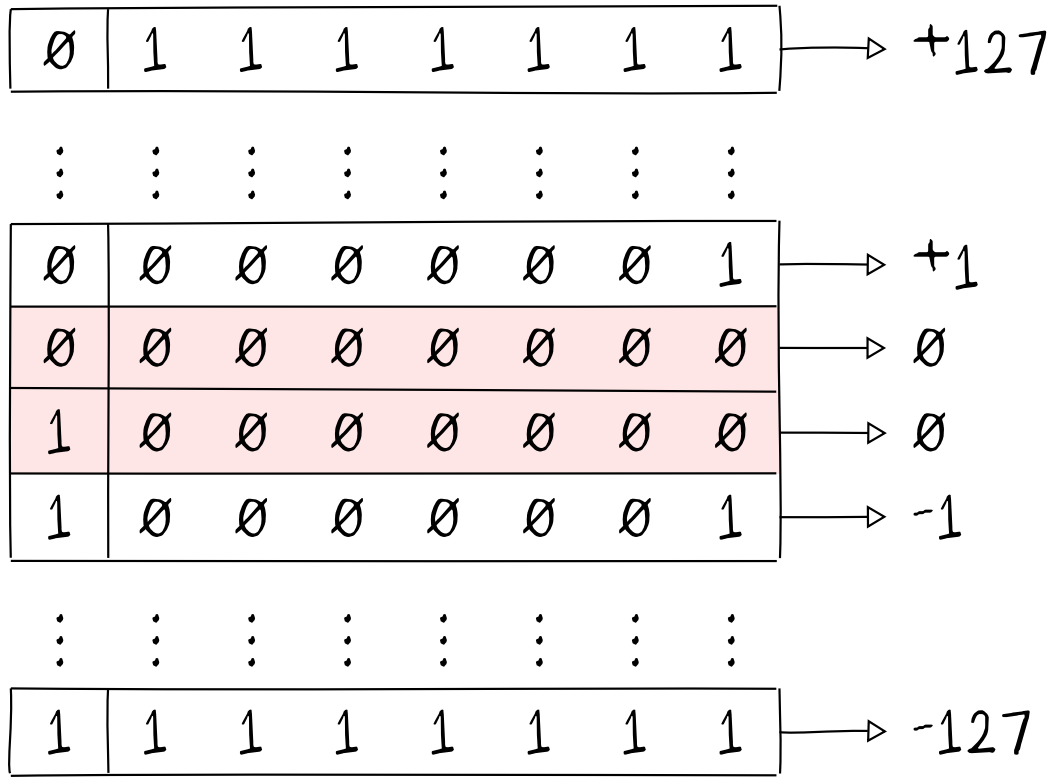
Introdução

- Método de sinal e magnitude (8 bits)
  - Primeiro bit indica o sinal e demais bits a magnitude



Introdução

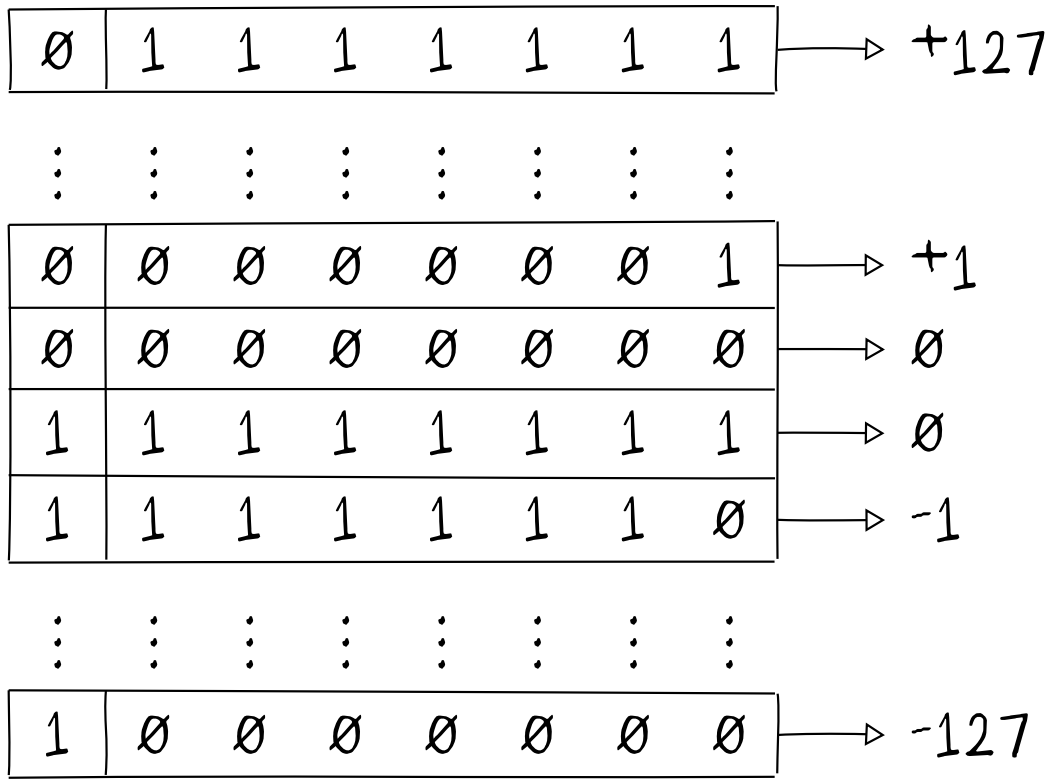
- Método de sinal e magnitude (8 bits)
  - Primeiro bit indica o sinal e demais bits a magnitude



Duas representações para o valor 0

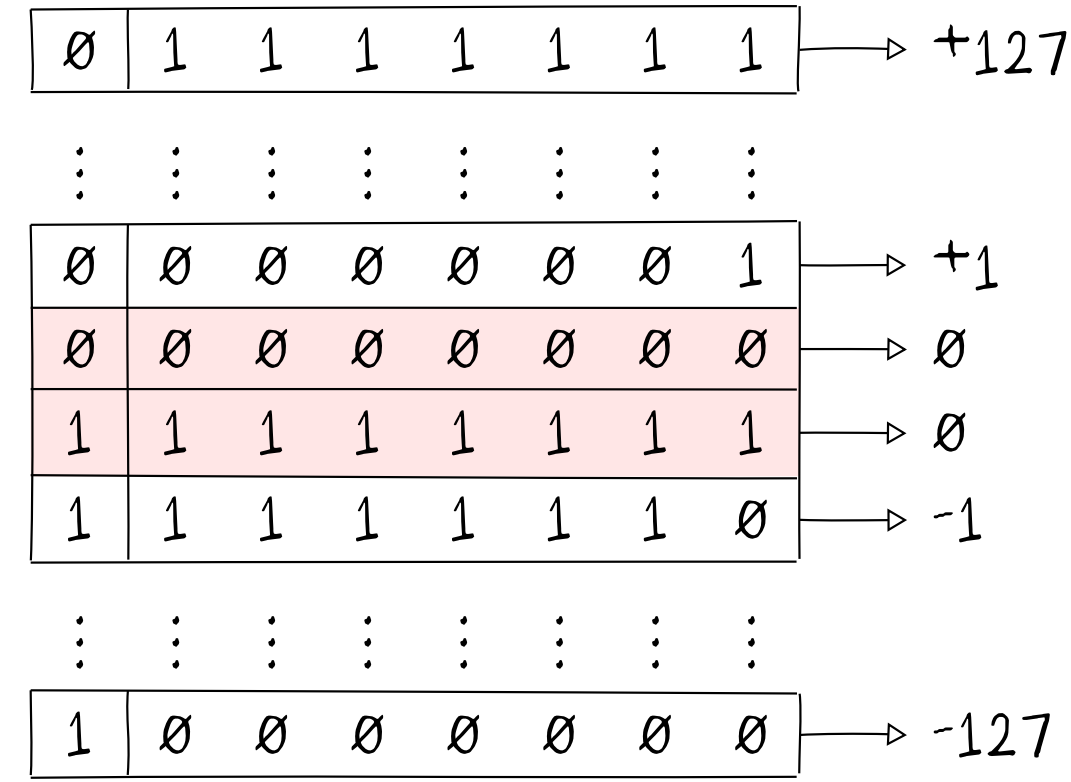
Introdução

- Método de complemento a 1 (8 bits)
  - O valor de sinal oposto é o complemento bit a bit



Introdução

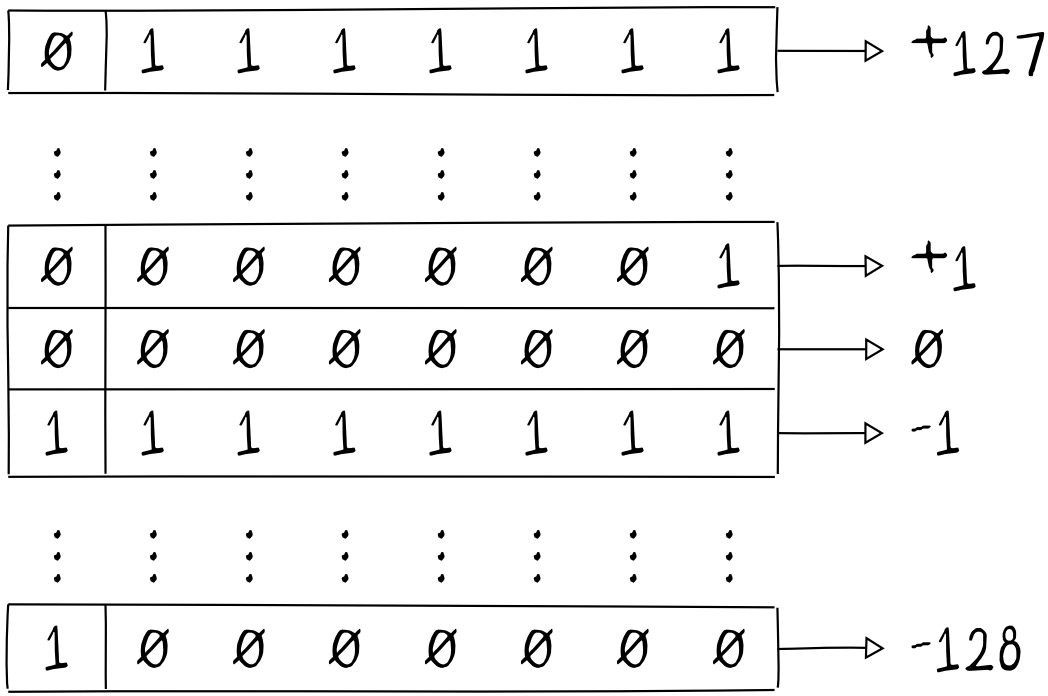
- Método de complemento a 1 (8 bits)
  - O valor de sinal oposto é o complemento bit a bit



DUAS REPRESENTAÇÕES PARA O VALOR 0

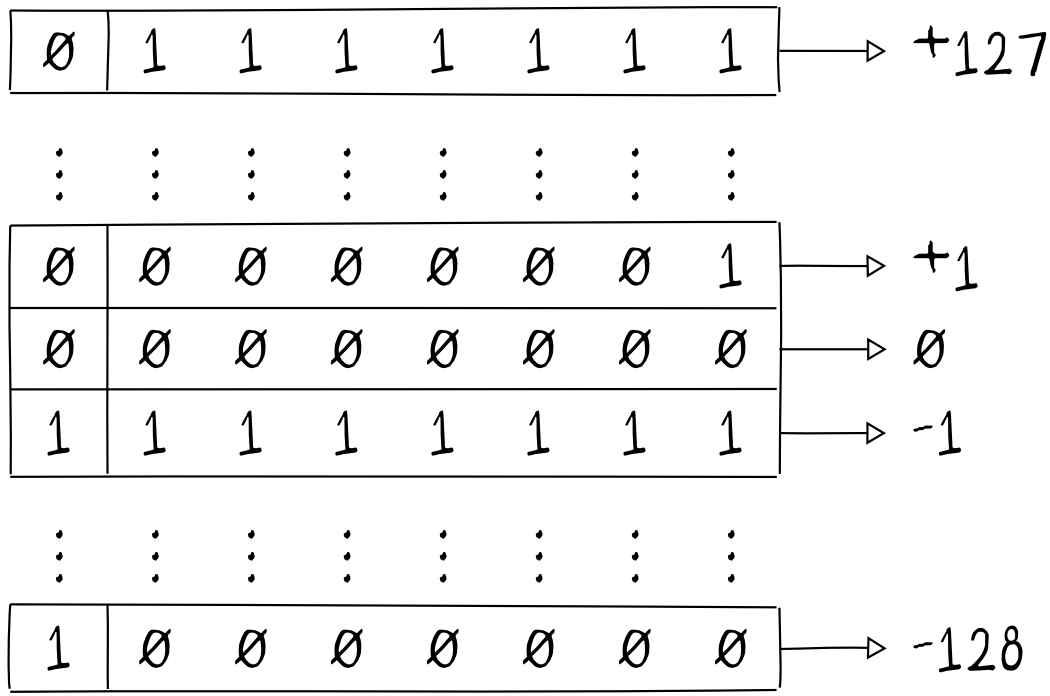
Introdução

- Método de complemento a 2 (8 bits)
  - O valor de sinal oposto é o complemento bit a bit + 1



Introdução

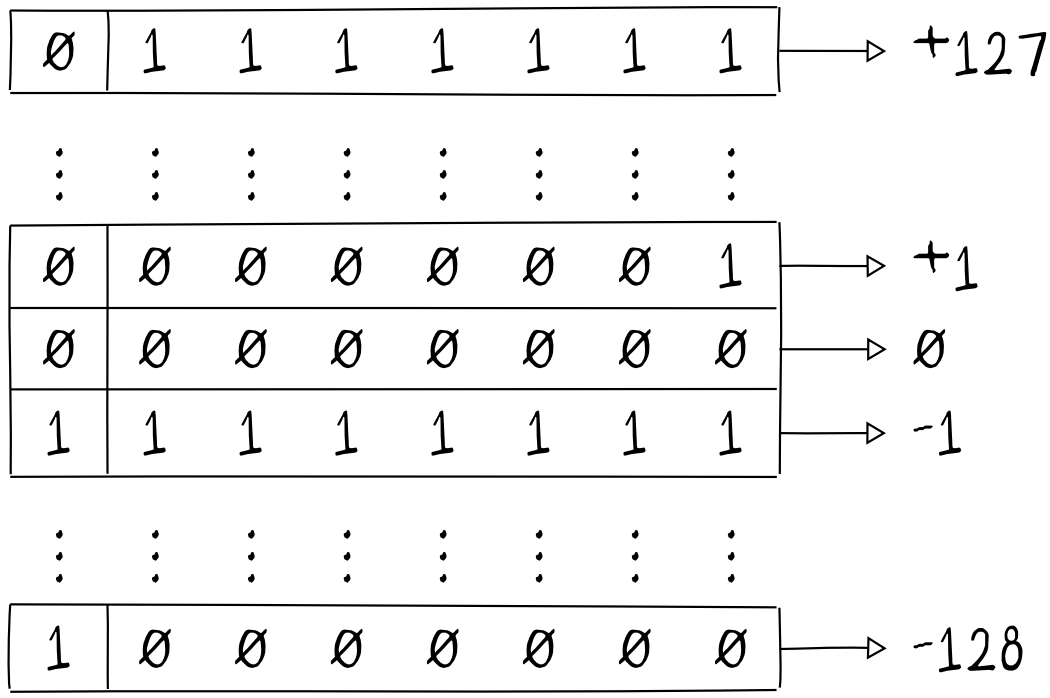
- Método de complemento a 2 (8 bits)
  - O valor de sinal oposto é o complemento bit a bit + 1



$2^7 = 128$ VALORES NEGATIVOS E POSITIVOS

Introdução

- Método de complemento a 2 (8 bits)
  - O valor de sinal oposto é o complemento bit a bit + 1

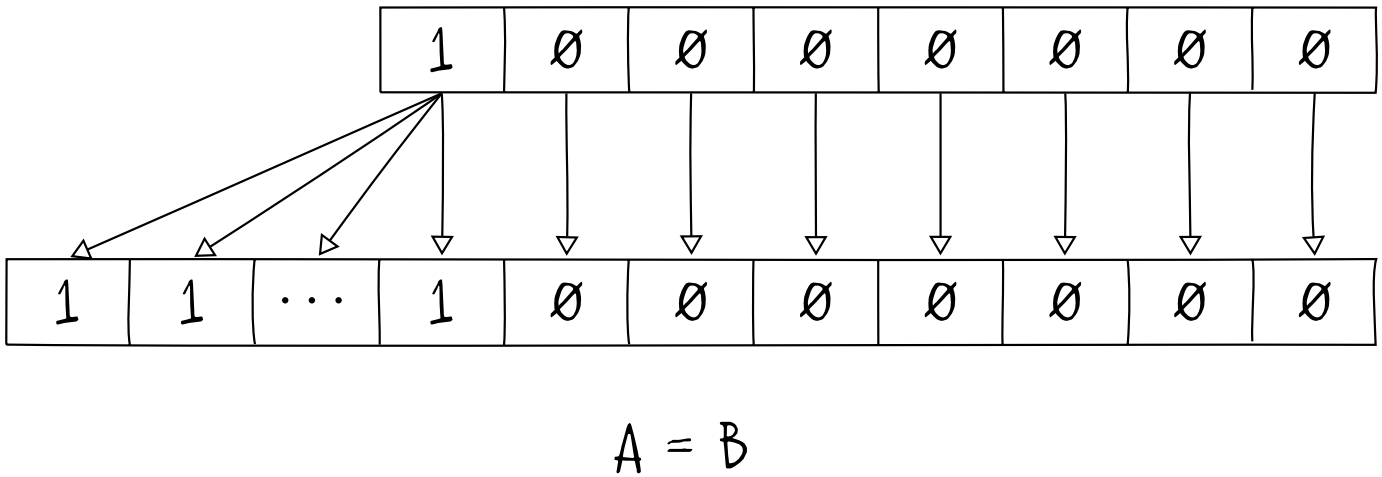


$2^7 = 128$ VALORES NEGATIVOS E POSITIVOS

COM UMA CAPACIDADE DE N BITS,
OS VALORES ESTÃO ENTRE -2^{N-1} E $+2^{N-1}-1$

Introdução

- Operação de extensão de sinal (*sign_extension*)
  - $A[8] = 10000000_2 = -128_{10}$
  - $B[32] = 1111 \cdots 111110000000_2 = -128_{10}$



Números inteiros

- Adição binária (8 bits)
  - $A = 200_{10} = 1 \times 2^7 + 1 \times 2^6 + 1 \times 2^3 = 11001000_2$
  - $B = 25_{10} = 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^0 = 00011001_2$
  - $R = A + B = 11001000_2 + 00011001_2 = 11100001_2\,(225_{10})$

A_i	B_i	R_i	C_i
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$
\begin{array}{cccccccc}
 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
+ & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
\hline
 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1
\end{array}
$$

Números inteiros

- Adição binária (8 bits)
  - $A = 200_{10} = 1 \times 2^7 + 1 \times 2^6 + 1 \times 2^3 = 11001000_2$
  - $B = 25_{10} = 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^0 = 00011001_2$
  - $R = A + B = 11001000_2 + 00011001_2 = 11100001_2\,(225_{10})$

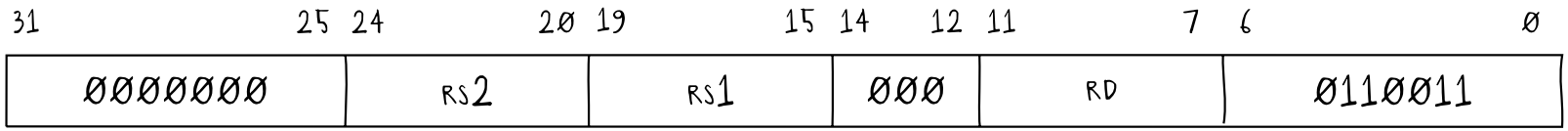
A_i	B_i	R_i	C_i
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$
\begin{array}{cccccccc}
 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
+ & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
\hline
 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1
\end{array}
$$

A ADIÇÃO DE NÚMEROS COM N BITS PODE GERAR UM RESULTADO COM ATÉ N + 1 BITS

Números inteiros

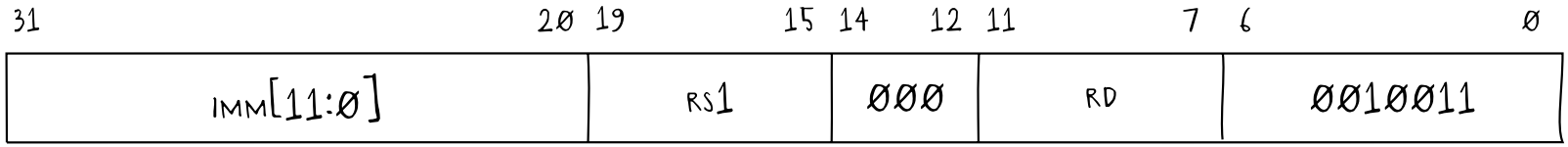
- Adição (*add*)




```
add rd, rs1, rs2:
    rd = rs1 + rs2
```

Números inteiros

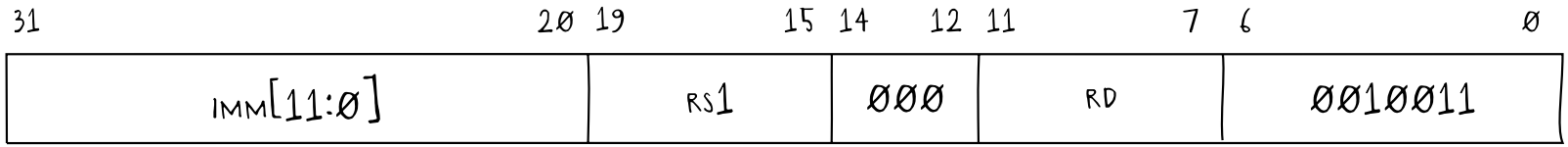
- Adição imediata (*add immediate*)




```
addi rd, rs1, imm:
    rd = rs1 + sign_extension(imm)

nop:
    addi x0, x0, 0

li rd, imm:
    addi rd, x0, imm

li rd, imm32:
    lui rd, get_31_12(imm32)
    addi rd, rd, get_11_0(imm32)

...
```

Números inteiros

- Adição imediata (*add immediate*)




```
addi rd, rs1, imm:
    rd = rs1 + sign_extension(imm)

...

mv rd, rs1:
    addi rd, rs1, 0

la rd, imm32:
    auipc rd, get_31_12(imm32)
    addi rd, rd, get_11_0(imm32)
```

Números inteiros

- Subtração binária (8 bits)
  - $A = 200_{10} = 1 \times 2^7 + 1 \times 2^6 + 1 \times 2^3 = 11001000_2$
  - $B = 25_{10} = 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^0 = 00011001_2$
  - $R = A - B = 11001000_2 + 11100111_2 = 10101111_2\,(175_{10})$

A_i	B_i	R_i	C_i
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$
\begin{array}{cccccccc}
 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
+ & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\
\hline
 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1
\end{array}
$$

Números inteiros

- Subtração binária (8 bits)
  - $A = 200_{10} = 1 \times 2^7 + 1 \times 2^6 + 1 \times 2^3 = 11001000_2$
  - $B = 25_{10} = 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^0 = 00011001_2$
  - $R = A - B = 11001000_2 + 11100111_2 = 10101111_2\,(175_{10})$

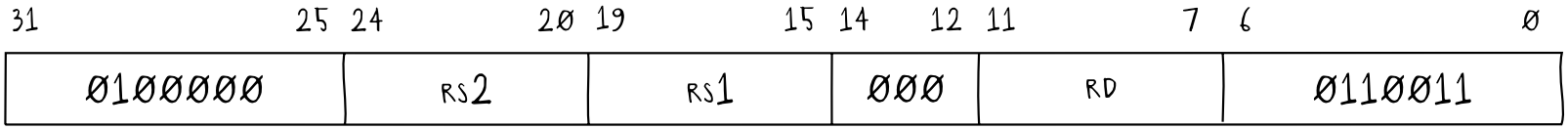
A_i	B_i	R_i	C_i
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$
\begin{array}{cccccccc}
 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
+ & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\
\hline
 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1
\end{array}
$$

A SUBTRAÇÃO DE NÚMEROS COM N BITS PODE GERAR UM RESULTADO COM ATÉ N + 1 BITS

Números inteiros

- Subtração (*sub*)




```
sub rd, rs1, rs2:
    rd = rs1 - rs2

neg rd, rs2:
    sub rd, x0, rs2
```

Números inteiros

- Multiplicação binária (8 bits)
  - $A = 11_{10} = 1 \times 2^3 + 1 \times 2^1 + 1 \times 2^0 = 00001011_2$
  - $B = 13_{10} = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^0 = 00001101_2$
  - $R = A \times B = 143_{10} = 10001111_2$

A_i	B_i	R_i
0	0	0
0	1	0
1	0	0
1	1	1

```
                    1 0 1 1
                x   1 1 0 1
                ---------
                    1 0 1 1
                0 0 0 0
            + 1 0 1 1
        1 0 1 1
        ---------------
1 0 0 0 1 1 1 1
```

Números inteiros

- Multiplicação binária (8 bits)
  - $A = 11_{10} = 1 \times 2^3 + 1 \times 2^1 + 1 \times 2^0 = 00001011_2$
  - $B = 13_{10} = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^0 = 00001101_2$
  - $R = A \times B = 143_{10} = 10001111_2$

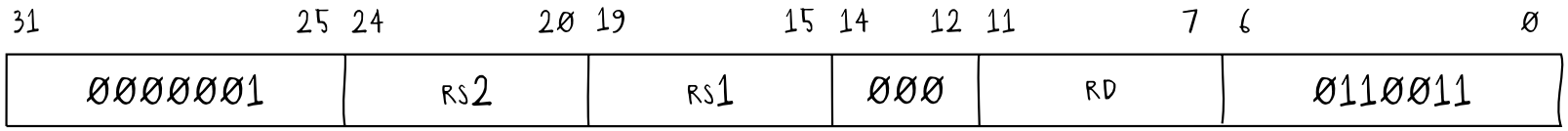
A_i	B_i	R_i
0	0	0
0	1	0
1	0	0
1	1	1

```
                    1 0 1 1
                x   1 1 0 1
                ---------
                    1 0 1 1
                0 0 0 0
            + 1 0 1 1
        1 0 1 1
        ---------------
1 0 0 0 1 1 1 1
```

A multiplicação de números com n bits necessita de até 2n bits para armazenar o resultado

Números inteiros

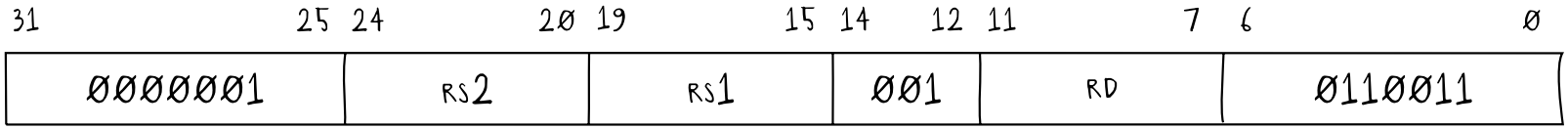
- Multiplicação (*mul*)




```
mul rd, rs1, rs2:
    rd = get_31_0(rs1 * rs2)
```

Números inteiros

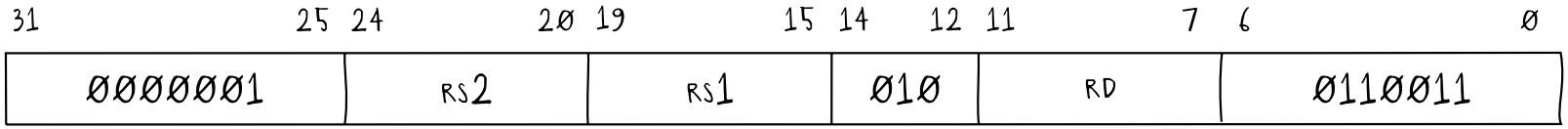
- Multiplicação (sinal-sinal) retornando os 32 bits mais significativos (*mulh*)




```
mulh rd, rs1, rs2:
    rd = get_63_32(rs1 * rs2)
```

Números inteiros

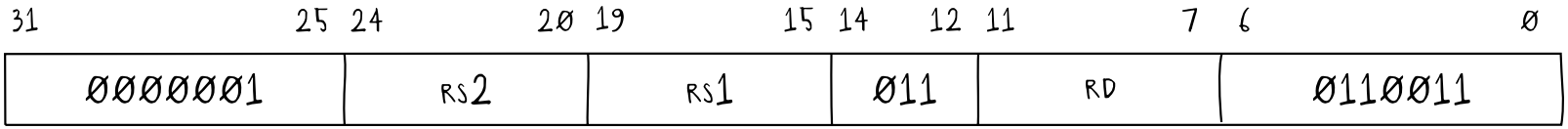
- Multiplicação (sinal-sem sinal) retornando os 32 bits mais significativos (*mulhsu*)




```
mulhsu rd, rs1, rs2:
    rd = get_63_32(rs1 * rs2)
```

Números inteiros

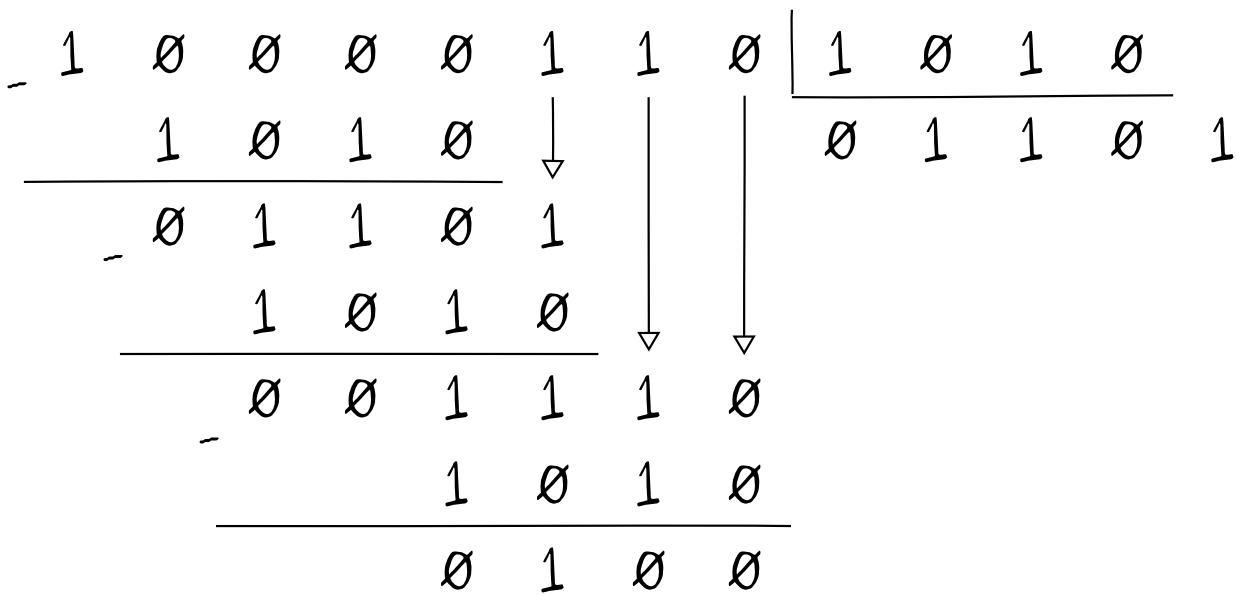
- Multiplicação (sem sinal-sem sinal) retornando os 32 bits mais significativos (*mulhu*)




```
mulhu rd, rs1, rs2:
    rd = get_63_32(rs1 * rs2)
```

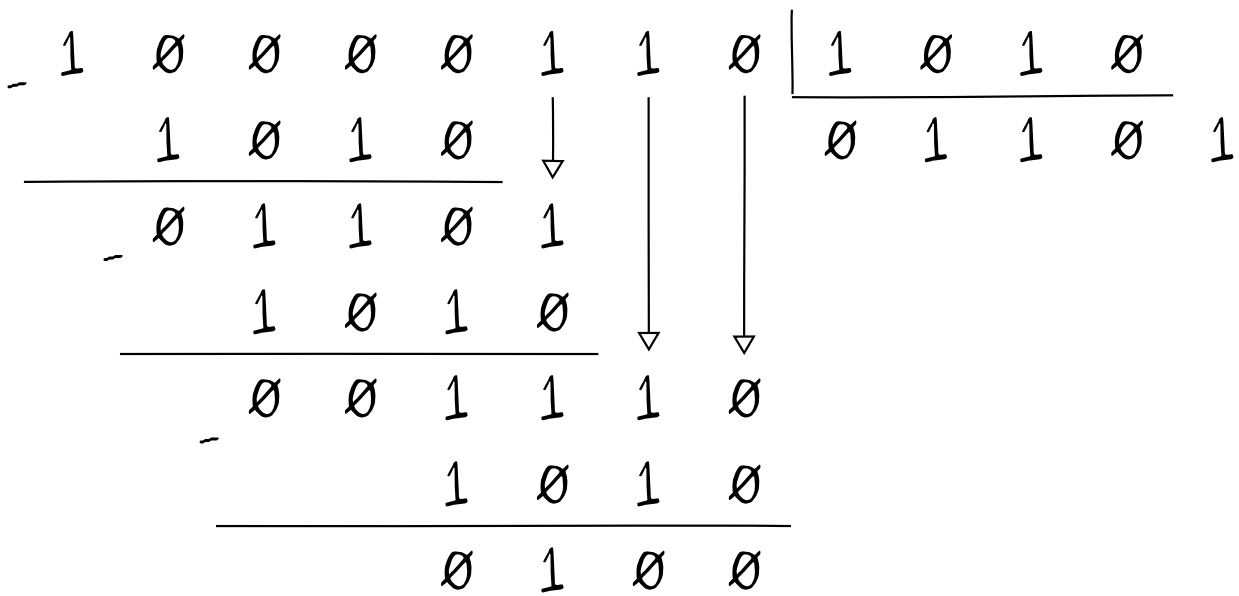
Números inteiros

- Divisão binária (8 bits)
  - $A = 134_{10} = 1 \times 2^7 + 1 \times 2^2 + 1 \times 2^1 = 10000110_2$
  - $B = 10_{10} = 1 \times 2^3 + 1 \times 2^1 = 00001010_2$
  - $Q = A \div B = 13_{10} = 00001101_2$
  - $R = A \bmod B = 4_{10} = 00000100_2$



Números inteiros

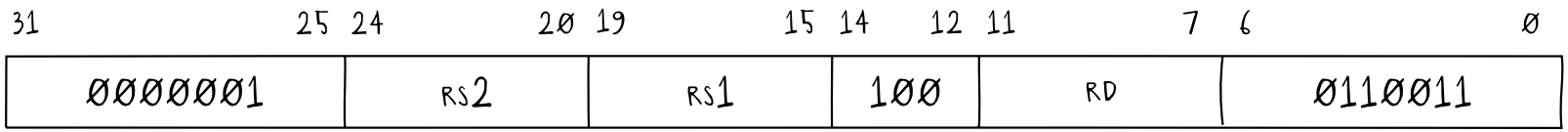
- Divisão binária (8 bits)
  - $A = 134_{10} = 1 \times 2^7 + 1 \times 2^2 + 1 \times 2^1 = 10000110_2$
  - $B = 10_{10} = 1 \times 2^3 + 1 \times 2^1 = 00001010_2$
  - $Q = A \div B = 13_{10} = 00001101_2$
  - $R = A \bmod B = 4_{10} = 00000100_2$



A DIVISÃO DE NÚMEROS COM N BITS NECESSITA DE ATÉ N BITS PARA O QUOCIENTE E O RESTO

Números inteiros

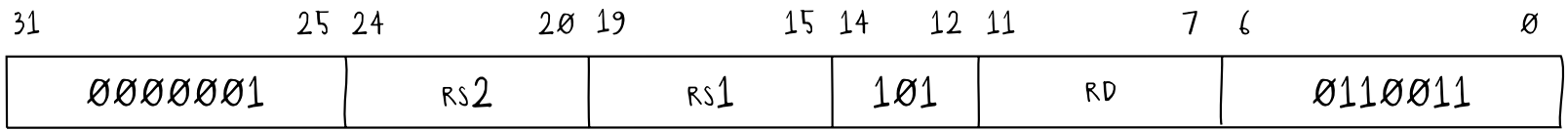
- Divisão com sinal (*div*)




```
div rd, rs1, rs2:
    rd = rs1 / rs2
```

Números inteiros

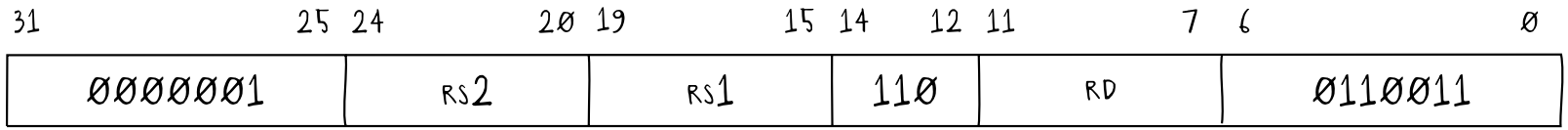
- Divisão sem sinal (*divu*)




```
divu rd, rs1, rs2:
    rd = rs1 / rs2
```

Números inteiros

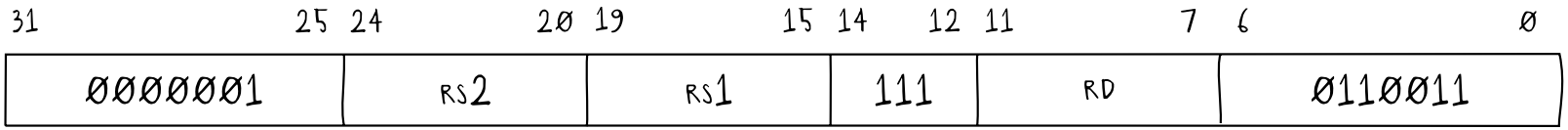
- Resto da divisão com sinal (*rem*)




```
rem rd, rs1, rs2:
    rd = rs1 % rs2
```

Números inteiros

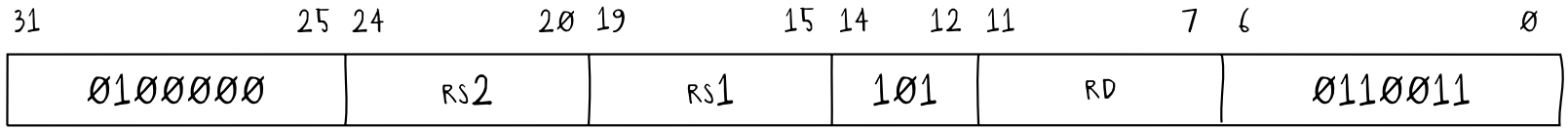
- Resto da divisão sem sinal (*remu*)




```
remu rd, rs1, rs2:
    rd = rs1 % rs2
```

Números inteiros

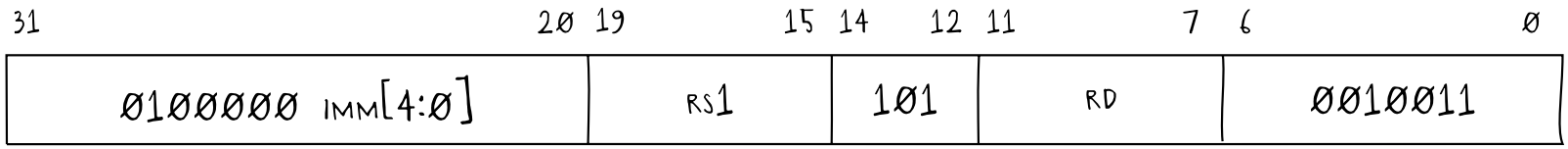
- Deslocamento para direita aritmético (*shift right arithmetic*)




```
sra rd, rs1, rs2:
    rd = rs1 >> get_4_0(rs2)
```

Números inteiros

- Deslocamento para direita aritmético imediato (*shift right arithmetic immediate*)

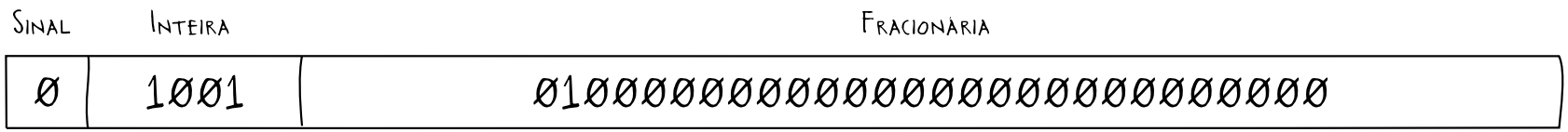



```
srai rd, rs1, imm:
    rd = rs1 >> imm
```

Números reais

- Aritmética de ponto fixo (notação Q)
  - Definição da parte inteira e fracionária (32 bits)

$$\begin{aligned} Q4.27 &= 9,25_{10} \\ &= 9_{10} + 0,25_{10} \\ &= 1001_2 + 0,01_2 \end{aligned}$$



Números reais

- Aritmética de ponto fixo (notação Q)
  - Entendendo a codificação binária fracionária

$$9,25_{10} = 9_{10} + 0,25_{10}$$

Números reais

- Aritmética de ponto fixo (notação Q)
  - Entendendo a codificação binária fracionária

$$9,25_{10} = 9_{10} + 0,25_{10}$$

$$9,25_{10} = \left(2^3 + \underline{1}\right) + 0,25_{10}$$

Números reais

- Aritmética de ponto fixo (notação Q)
  - Entendendo a codificação binária fracionária

$$9,25_{10} = 9_{10} + 0,25_{10}$$

$$9,25_{10} = \left(2^3 + 2^0\right) + 0,25_{10}$$

Números reais

- Aritmética de ponto fixo (notação Q)
  - Entendendo a codificação binária fracionária

$$9,25_{10} = 9_{10} + 0,25_{10}$$

$$9,25_{10} = 1001_2 + 0,25_{10}$$

Números reais

- Aritmética de ponto fixo (notação Q)
  - Entendendo a codificação binária fracionária

$$9,25_{10} = 9_{10} + 0,25_{10}$$

$$9,25_{10} = 1001_2 + \left(2^{-2} + \underline{0}\right)$$

Números reais

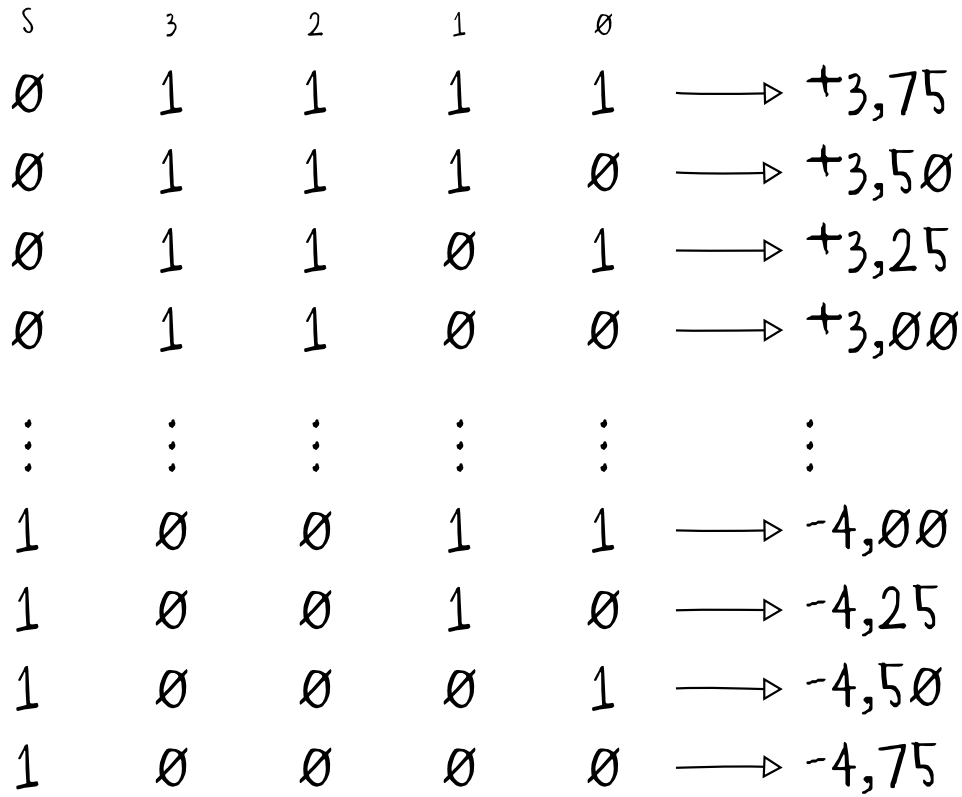
- Aritmética de ponto fixo (notação Q)
  - Entendendo a codificação binária fracionária

$$9,25_{10} = 9_{10} + 0,25_{10}$$

$$9,25_{10} = 1001_2 + 0,01_2$$

Números reais

- Aritmética de ponto fixo (notação Q)
  - Representatividade um número Q2.2

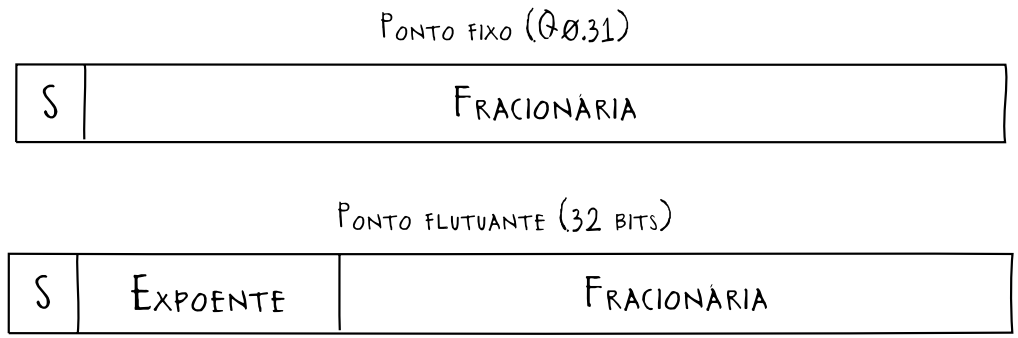


Números reais

- Aritmética de ponto fixo (notação *Q*)
  ✓ São utilizados componentes de aritmética inteira que permitem operações mais rápidas e simples

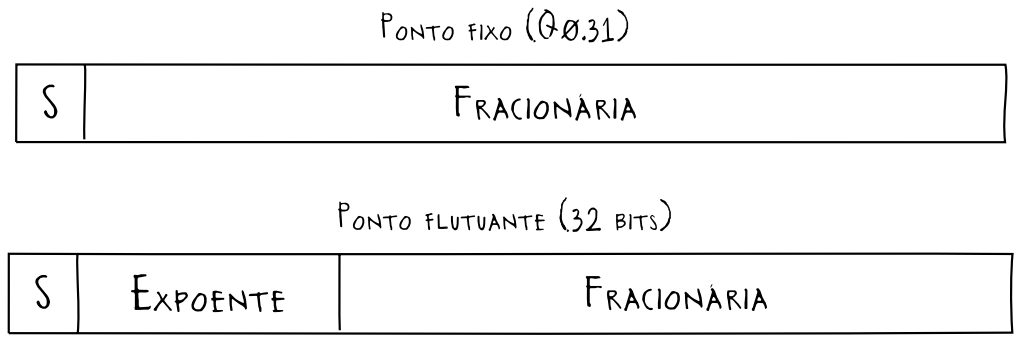
Números reais

- Aritmética de ponto fixo (notação Q)
  ✓ São utilizados componentes de aritmética inteira que permitem operações mais rápidas e simples
  ✓ Maior precisão da parte fracionária com mesma quantidade de bits



Números reais

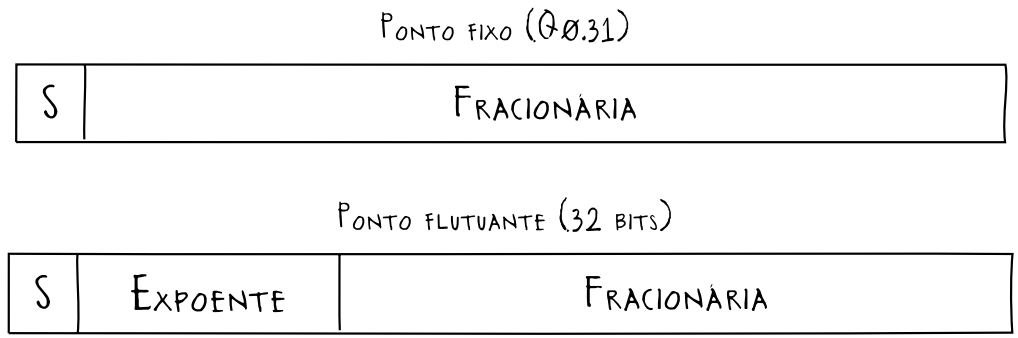
- Aritmética de ponto fixo (notação Q)
  ✓ São utilizados componentes de aritmética inteira que permitem operações mais rápidas e simples
  ✓ Maior precisão da parte fracionária com mesma quantidade de bits



✗ Representatividade limitada

Números reais

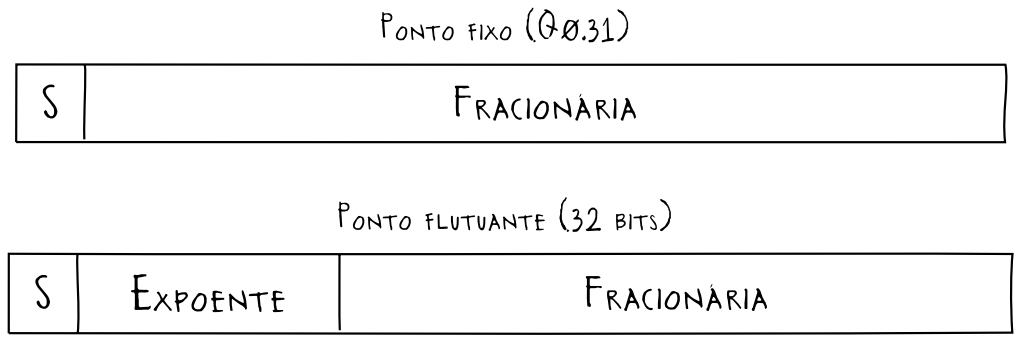
- Aritmética de ponto fixo (notação *Q*)
  ✓ São utilizados componentes de aritmética inteira que permitem operações mais rápidas e simples
  ✓ Maior precisão da parte fracionária com mesma quantidade de bits



  ✗ Representatividade limitada
  ✗ Problemas com arredondamento e *overflow*

Números reais

- Aritmética de ponto fixo (notação *Q*)
  ✓ São utilizados componentes de aritmética inteira que permitem operações mais rápidas e simples
  ✓ Maior precisão da parte fracionária com mesma quantidade de bits

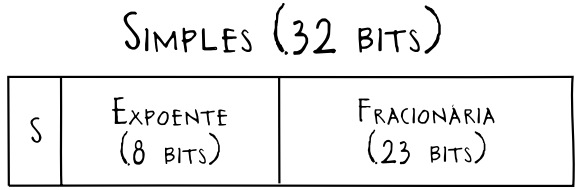


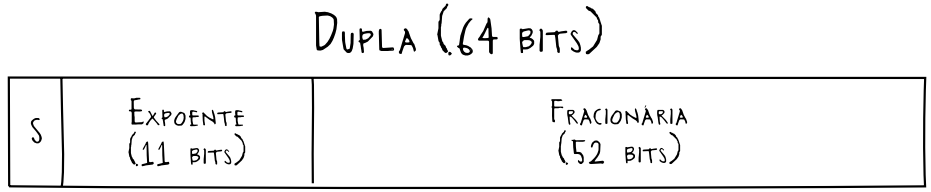
- ✗ Representatividade limitada
- ✗ Problemas com arredondamento e *overflow*

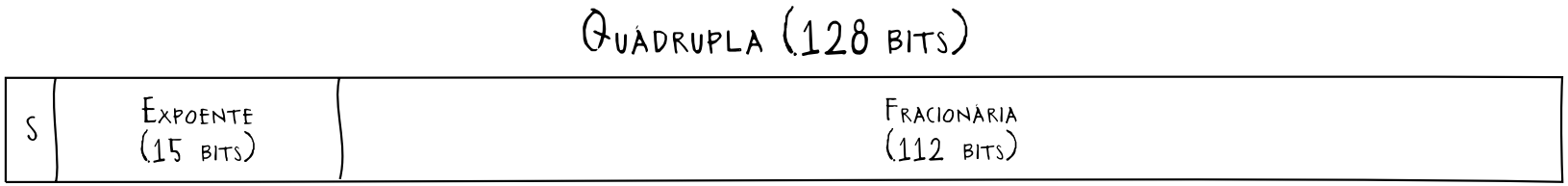
Aplicação principal: sistemas de baixo custo sem unidade de ponto flutuante (FPU)

Números reais

- Aritmética de ponto flutuante (IEEE 754)
  - Capacidades de representação



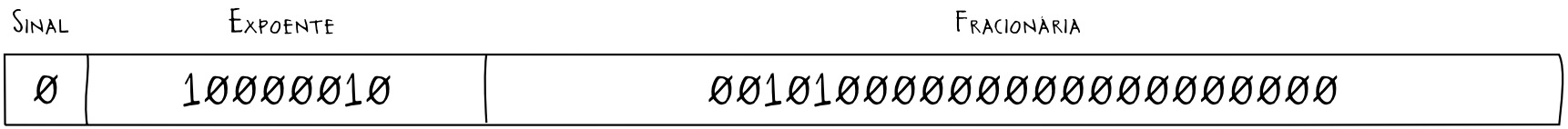




Números reais

- Aritmética de ponto flutuante (IEEE 754)
  - Representação com 32 bits (*float*)

$$\textbf{float} = (-1)^{Sinal} \left(1 + \sum_{i=0}^{22} B_{22-i} 2^{-i}\right) \times 2^{(Expoente - 127)}$$



$$\begin{aligned}
9,25_{10} &= 9_{10} + 0,25_{10} \\
&= 1001_2 + 0,01000000000000000000_2 \\
&= 1,00101000000000000000000_2 \times 2^3 \\
&= (-1)^0 (1_2 + 0,00101000000000000000000_2) \times 2^{(130_{10} - 127_{10})} \\
&= (-1)^0 (1_2 + 0,00101000000000000000000_2) \times 2^{(10000010_2 - 127_{10})}
\end{aligned}$$

Números reais

- Aritmética de ponto flutuante
  - ✓ Maior representatividade de valores

Números reais

- Aritmética de ponto flutuante
  - ✓ Maior representatividade de valores
  - ✓ Mecanismos de arredondamento e de precisão

Números reais

- Aritmética de ponto flutuante
  - ✓ Maior representatividade de valores
  - ✓ Mecanismos de arredondamento e de precisão
  - ✗ Hardware dedicado que aumenta o custo e o consumo de potência do sistema

Números reais

- Aritmética de ponto flutuante
  - ✓ Maior representatividade de valores
  - ✓ Mecanismos de arredondamento e de precisão
  - ✗ Hardware dedicado que aumenta o custo e o consumo de potência do sistema
  - ✗ As operações são mais complexas e demoradas

Números reais

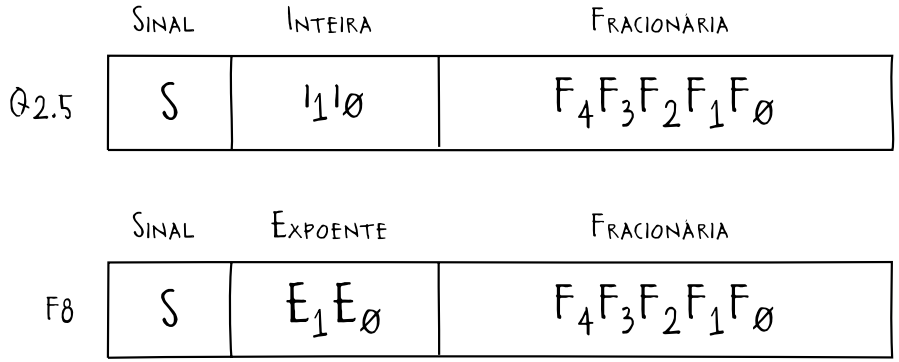
- Aritmética de ponto flutuante
  - ✓ Maior representatividade de valores
  - ✓ Mecanismos de arredondamento e de precisão
  - ✗ Hardware dedicado que aumenta o custo e o consumo de potência do sistema
  - ✗ As operações são mais complexas e demoradas

Aplicação principal: redução do tempo de projeto em sistemas com unidade de ponto flutuante (FPU)

Exercício

- Considerando os métodos de complemento a 2, de ponto fixo $Q2.5$ e de ponto flutuante $F8$ descritas abaixo, converta os números reais $A = 2,71$ e $B = 3,14$ para estas representações numéricas

$$F8 = (-1)^S \left(1 + \sum_{i=0}^{4} B_{4-i} 2^{-i}\right) \times 2^{Expoente}$$



  - Realize a operação $A - B$ para cada representação e compare erro dos resultados obtidos
- Para entender na prática como as operações de ponto flutuante podem ser implementadas, estude as extensões F e D da arquitetura RISC-V

Exercício

- Implemente um simulador do Poxim-V, utilizando as linguagens de programação suportadas e obtendo os argumentos de entrada e de saída pela linha de comando
  1. Realize o carregamento da programação na memória
  2. Execute cada instrução passo a passo
  3. Gere o fluxo de execução da aplicação

```
@80000000
6F 00 00 0A 6F 00 40 08 6F 00 00 08 6F 00 C0 07
6F 00 80 07 6F 00 40 07 6F 00 00 07 6F 00 C0 06
6F 00 80 06 6F 00 40 06 6F 00 00 06 6F 00 C0 05
6F 00 80 05 00 00 00 00 00 00 00 00 00 00 00 00
17 05 00 00 13 05 05 08 97 05 00 00 93 85 85 07
23 20 05 00 13 05 45 00 E3 4C B5 FE 67 80 00 00
13 01 01 FF 23 22 A1 00 13 05 00 02 B7 05 02 00
93 85 65 02 23 20 B1 00 B3 05 20 00 EF 00 40 01
13 01 01 01 6F 00 00 00 13 05 10 00 6F F0 5F FD
13 10 F0 01 73 00 10 00 13 50 70 40 67 80 00 00
17 81 00 00 13 01 01 F6 EF F0 9F F9 EF 00 C0 00
6F F0 1F FB 00 00 00 00 13 05 00 00 67 80 00 00
```

Entrada hexadecimal (.hex)

```
0x80000000:jal    zero,0x00050       pc=0x800000a0,zero=0x80000004
0x800000a0:auipc  sp,0x00008         sp=0x00008000+0x800000a0=0x800080a0
0x800000a4:addi   sp,sp,0xf60        sp=0x800080a0+0xfffff60=0x80008000
0x800000a8:jal    ra,0xfffcc         pc=0x80000040,ra=0x800000ac
0x80000040:auipc  a0,0x00000         a0=0x00000000+0x80000040=0x80000040
0x80000044:addi   a0,a0,0x080        a0=0x80000040+0x00000080=0x800000c0
0x80000048:auipc  a1,0x00000         a1=0x00000000+0x80000048=0x80000048
0x8000004c:addi   a1,a1,0x078        a1=0x80000048+0x00000078=0x800000c0
0x80000050:sw     zero,0x000(a0)     mem[0x800000c0]=0x00000000
0x80000054:addi   a0,a0,0x004        a0=0x800000c0+0x00000004=0x800000c4
0x80000058:blt    a0,a1,0xffc        (0x800000c4<0x800000c0)=0->pc=0x8000005c
0x8000005c:jalr   zero,ra,0x000      pc=0x800000ac+0x00000000,zero=0x80000060
0x800000ac:jal    ra,0x00006         pc=0x800000b8,ra=0x800000b0
0x800000b8:addi   a0,zero,0x000      a0=0x00000000+0x00000000=0x00000000
0x800000bc:jalr   zero,ra,0x000      pc=0x800000b0+0x00000000,zero=0x800000c0
0x800000b0:jal    zero,0xfffd8       pc=0x80000060,zero=0x800000b4
0x80000060:addi   sp,sp,0xff0        sp=0x80008000+0xffffff0=0x80007ff0
0x80000064:sw     a0,0x004(sp)       mem[0x80007ff4]=0x00000000
0x80000068:addi   a0,zero,0x020      a0=0x00000000+0x00000020=0x00000020
0x8000006c:lui    a1,0x00020         a1=0x00020000
0x80000070:addi   a1,a1,0x026        a1=0x00020000+0x00000026=0x00020026
0x80000074:sw     a1,0x000(sp)       mem[0x80007ff0]=0x00020026
0x8000007c:jal    ra,0x0000a         pc=0x80000090,ra=0x80000080
0x80000090:slli   zero,zero,31       zero=0x00000000<<31=0x00000000
0x80000094:ebreak
```

Saída formatada (.out)