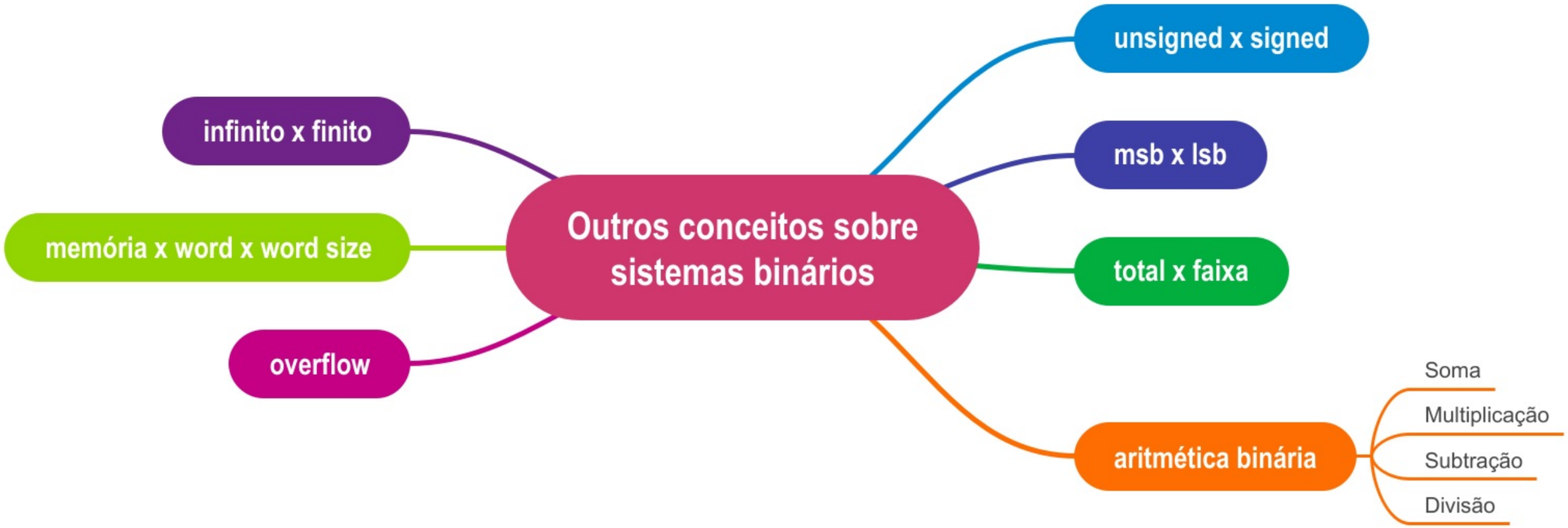
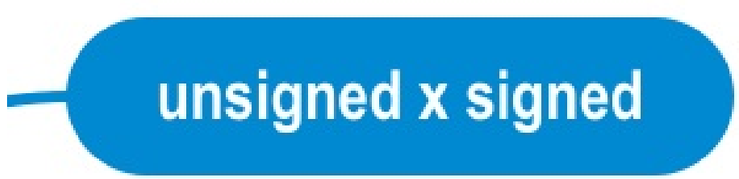
FUNDAMENTOS DA COMPUTAÇÃO

Outros conceitos fundamentais



Unsigned x Signed



Unsigned x Signed

Unsigned (sem sinal):

São números binários sem nenhuma informação sobre o sinal (+ ou -), ou seja, nenhum dos bits do número representa o sinal, todos os bits são utilizados para os números.

10110

10

10010011

11111111

00000001

10000000

A interpretação é simples:

são números não negativos.

Unsigned x Signed

Signed (com sinal):

São números binários com informação sobre o sinal (+ ou -), ou seja, um dos bits do número representa o sinal e os outros bits são utilizados para os números.

10110

10

10010011

11111111

00000001

10000000

A interpretação desses números depende da NOTAÇÃO utilizada. Sem saber quel a notação, não é possível identificar que número está sendo representado.

Veremos isso em outro vídeo.

msb x lsb

msb x lsb

msb: most significant bit (bit mais significativo)
É o bit mais à esquerda em um número binário,
pois tem o maior valor posicional.

lsb: least significant bit (bit menos significativo)
É o bit mais à direita em um número binário,
pois tem o menor valor posicional.

10110100

01010101

11110001

01001110

Não confunda as POSIÇÕES msb e lsb
com os ALGARISMOS 0 e 1.

Total x Faixa

Total x Faixa

O total de números que podem ser representados com n bits, e a faixa numérica (menor e maior) desses números, depende se estamos considerando binários unsigned ou signed.

Para binários unsigned com n bits, temos o seguinte:

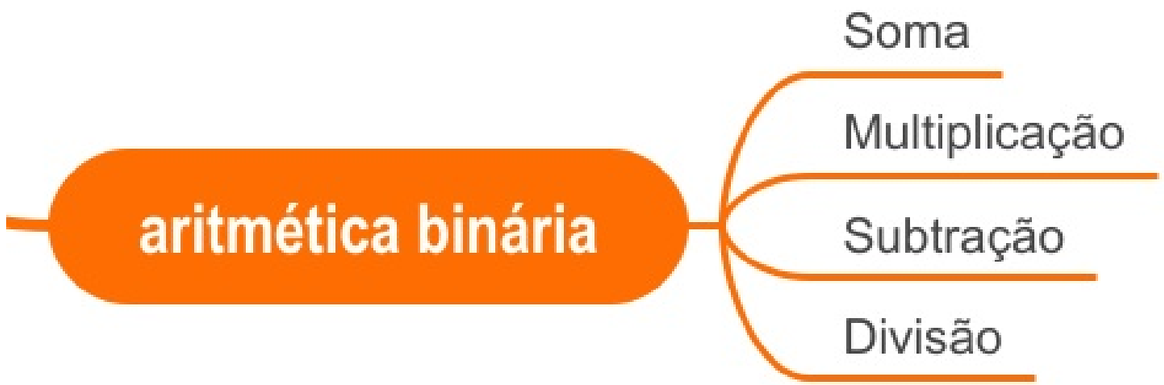
- Total de números: 2^n

- Faixa numérica: $[0; 2^n - 1]$

Para binários signed com n bits:

- Depende da NOTAÇÃO, veremos em outro vídeo.

Aritmética binária



Aritmética binária: soma

Parecida com a soma decimal:

- Da direita para esquerda

- Regras:

```
0 + 0       =  0
0 + 1       =  1
1 + 0       =  1
1 + 1       = 10 (0 e vai 1)
1 + 1 + 1   = 11 (1 e vai 1)
```

Binários unsigned: sem problema!

Binários signed: depende da notação (veremos em outro vídeo).

Aritmética binária: soma

Ex. 1: (8 + 5) = 13

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

Ex. 2: (9 + 5) = 14

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

Ex. 3: (12 + 13) = 25

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

Aritmética binária: multiplicação

Processo idêntico à multiplicação decimal (cuidado com a adição final).

Binários unsigned: sem problema!

Binários signed: depende da notação.

Aritmética binária: multiplicação

Ex. 1: (12 x 10) = 120

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

Ex. 2: (12 x 11) = 132

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

Ex. 3: (13 x 13) = 169

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

Aritmética binária: subtração

Processo parecido com a subtração decimal, com uma diferença: quando ocorre um empréstimo o valor emprestado é dois.

Binários unsigned: sem problema!

Binários signed: depende da notação (veremos em outro vídeo).

Aritmética binária: subtração

Ex. 1: (13 - 5) = 8

```
  1 1 0 1
- 1 0 1
---------
  1 0 0 0
```

Ex. 2: (14 - 11) = 3

```
  1 1 1 0
- 1 0 1 1
---------
    1 1
```

Ex. 3: (34 - 8) = 26

```
  1 0 0 0 1 0
-     1 0 0 0
-------------
  1 1 0 1 0
```

Aritmética binária: divisão

Trabalhosa. Duas técnicas principais:

- **subtração sucessiva**
- **deslocamento**

Binários unsigned: sem problema!

Binários signed: depende da notação.

Aritmética binária: divisão

Divisão por subtração sucessiva:

- **A cada subtração, incrementa-se o quociente.**
- **O processo continua enquanto o dividendo for maior do que o divisor.**
- **O resto é o resultado da última subtração feita.**

Aritmética binária: divisão

Ex. 1: (8 / 2) = 4

```
    1 0 0 0
/       1 0
-----------
```

Ex. 2: (13 / 4) = 3 r1

```
    1 1 0 1
/     1 0 0
-----------
```

Ex. 3: (17 / 3) = 5 r2

```
    1 0 0 0 1
/         1 1
-------------
```

1 0 0

1 1 r 1

1 0 1 r 1 0

Aritmética binária: divisão

Divisão por deslocamento:

- **O divisor é deslocado para a esquerda, alinhando-se os msb.**
- **O deslocamento d será o valor parcial do quociente, calculado como 2^d.**
- **Realiza-se a subtração.**
- **Repetir o processo.**
- **Continuar até zerar a diferença ou até o dividendo ser menor do que o divisor.**
- **O valor final do quociente será a somatória dos quocientes parciais.**
- **O resto será o binário da última subtração.**

Aritmética binária: divisão

Ex. 1: (210 / 5) = 42

```
  1 1 0 1 0 0 1 0
/               1 0 1
-----------------------
```

```
1 0 1 0 1 0
```

Ex. 2: (210 / 8) = 26 r2

```
    1 1 0 1 0 0 1 0
/               1 0 0 0
-----------------------
```

```
1 1 0 1 0 r 1 0
```

Infinito x finito

Infinito x finito

É possível representar um conjunto infinito de números inteiros em um dispositivo eletrônico com memória finita?

$\mathbb{Z} \Longrightarrow$

(https://commons.wikimedia.org/wiki/File:Swissbit_2GB_PC2-5300U-555.jpg)

Infinito x finito

Solução:

Aceitar a limitação e escolher um determinado número de bits para representar uma faixa de números inteiros que seja boa o suficiente para a maioria dos cálculos do dia a dia.

Novo problema:

Quantos bits devemos usar para representar um número inteiro?

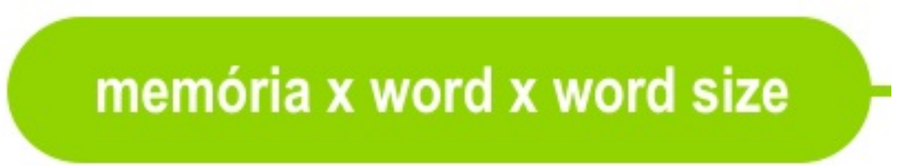
Infinito x finito

inteiros.c - GNU Emacs at cosmos

```c
#include <inttypes.h>
#include <math.h>
#include <stdint.h>
#include <stdio.h>

int main(void)
{
    uint8_t  x = (uint8_t)  (pow(2,  8) - 1.0);
    uint16_t y = (uint16_t) (pow(2, 16) - 1.0);
    uint32_t z = (uint32_t) (pow(2, 32) - 1.0);
    uint64_t w = (uint64_t) (pow(2, 64) - 1.0);

    printf("Maior número inteiro com  8 bits: %20" PRIu8  "\n", x);
    printf("Maior número inteiro com 16 bits: %20" PRIu16 "\n", y);
    printf("Maior número inteiro com 32 bits: %20" PRIu32 "\n", z);
    printf("Maior número inteiro com 64 bits: %20" PRIu64 "\n", w);

    return 0;
}
```

```
[abrantesasf@cosmos ~/cr6100b]$ ./inteiros
Maior número inteiro com  8 bits:                  255
Maior número inteiro com 16 bits:                65535
Maior número inteiro com 32 bits:           4294967295
Maior número inteiro com 64 bits: 18446744073709551615
```

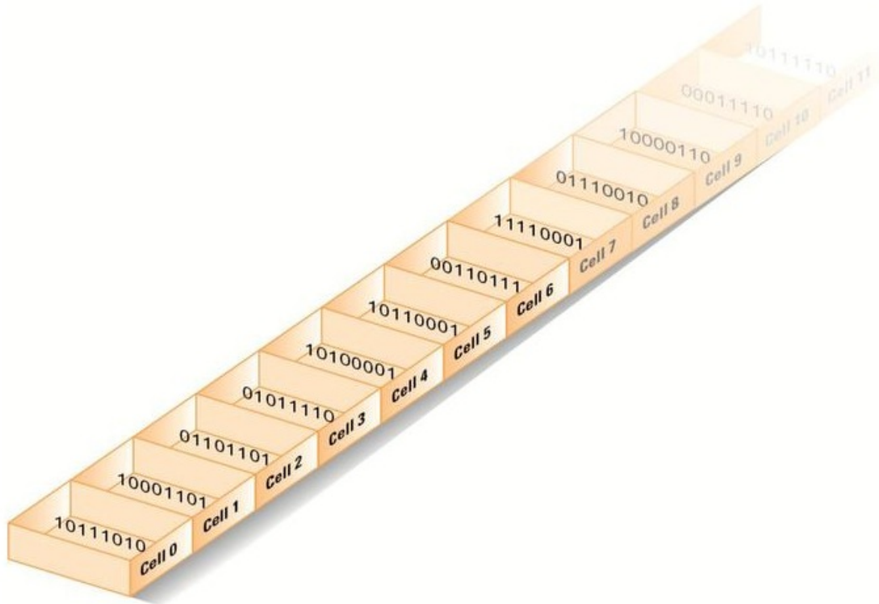
Memória x word x word size



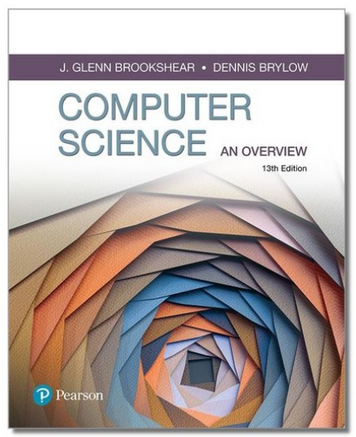
Memória x word x word size

A memória pode ser entendida como uma grande fila ordenada de"caixas de armazenamento de bits" chamadas de células, com as seguintes características:

- **cada célula armazena 8 bits (1 Byte)**
- **cada célula tem um endereço (começa em 0)**
- **dentro de cada célula os bits estão sempre ordenados (msb à esquerda; lsb à direita)**
- **como as células estão ordenadas, e os bits dentro das células também, todos os bits na memória estão organizados e ordenados em uma longa fila de bits**
- **devido à característica anterior, podemos armazenar padrões com mais de 8 bits utilizando células de memória consecutivas**
- **o acesso é "aleatório"**

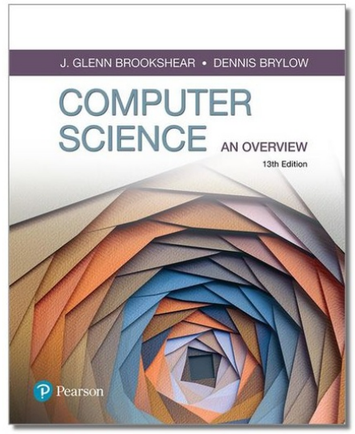


Brookshear & Brylow: Computer Science, an overview. 13ª ed, Pearson, 2019.



Memória x word x word size

Brookshear & Brylow: Computer Science, an overview. 13ª ed, Pearson, 2019.

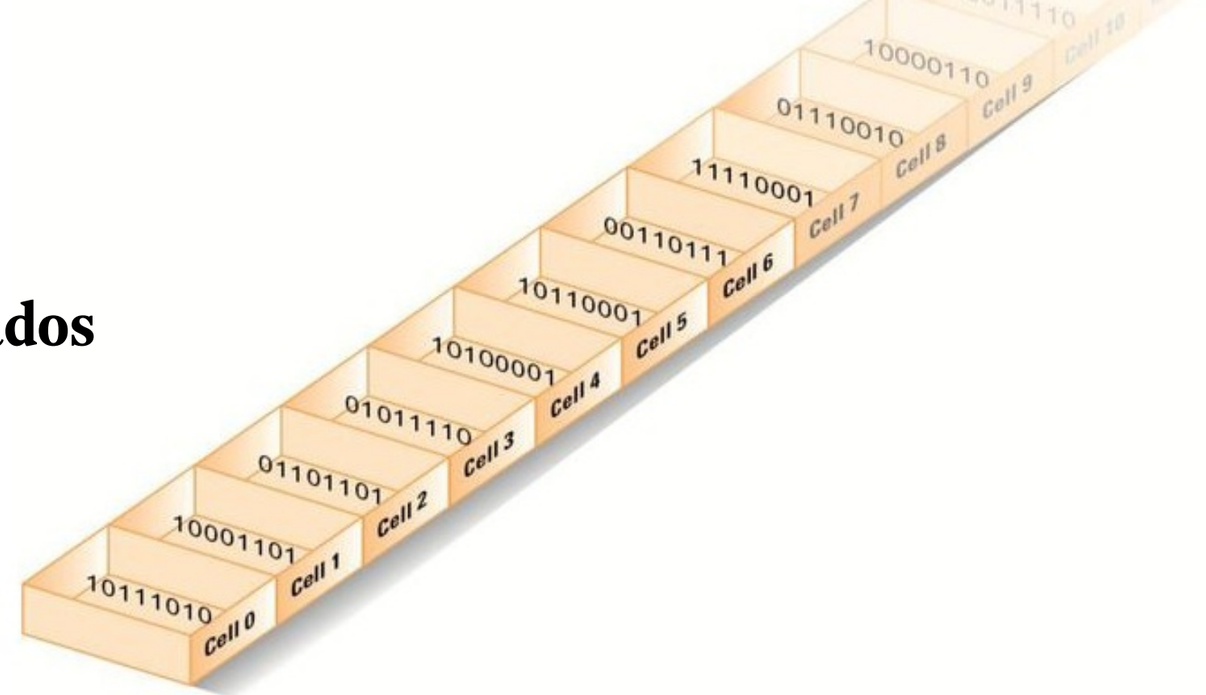


A word (palavra) é a quantidade de bits que o computador consegue ler, escrever e processar de uma única vez.

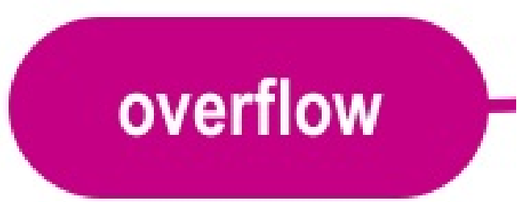
Cada computador tem uma word size (tamanho de palavra) específico, dependendo de sua arquitetura. As principais:

- **8 bits (1 Byte)**
- **16 bits (2 Bytes)**
- **32 bits (4 Bytes)**
- **64 bits (8 Bytes)**

O processador "sabe" como ler e trabalhar com inteiros representados com tamanhos diferentes de bits.



Overflow



Overflow

O overflow (transbordamento) é uma situação na qual o número que está sendo armazenado não cabe na quantidade de bits alocada para representá-lo.

Pode ocorrer ao armazenarmos um número ou como resultado de uma operação aritmética.

Se o programador não estiver atendo, introduz erros sutis e de difícil detecção no programa.

Ex.: em inteiros de 3 bits, realizar as seguintes somas:

4 + 3

5 + 4

Overflow

overflow.c - GNU Emacs at cosmos

```c
#include <inttypes.h>
#include <math.h>
#include <stdint.h>
#include <stdio.h>

int main(void)
{
    uint8_t x = 200;
    uint8_t y = 10;
    uint8_t z = x + y;

    printf("%" PRIu8 " + %" PRIu8 " = %" PRIu8 "\n", x, y, z);

    return 0;
}
```

```
[abrantesasf@cosmos ~/cr6100b]$ ./overflow
200 + 10 = 210
```

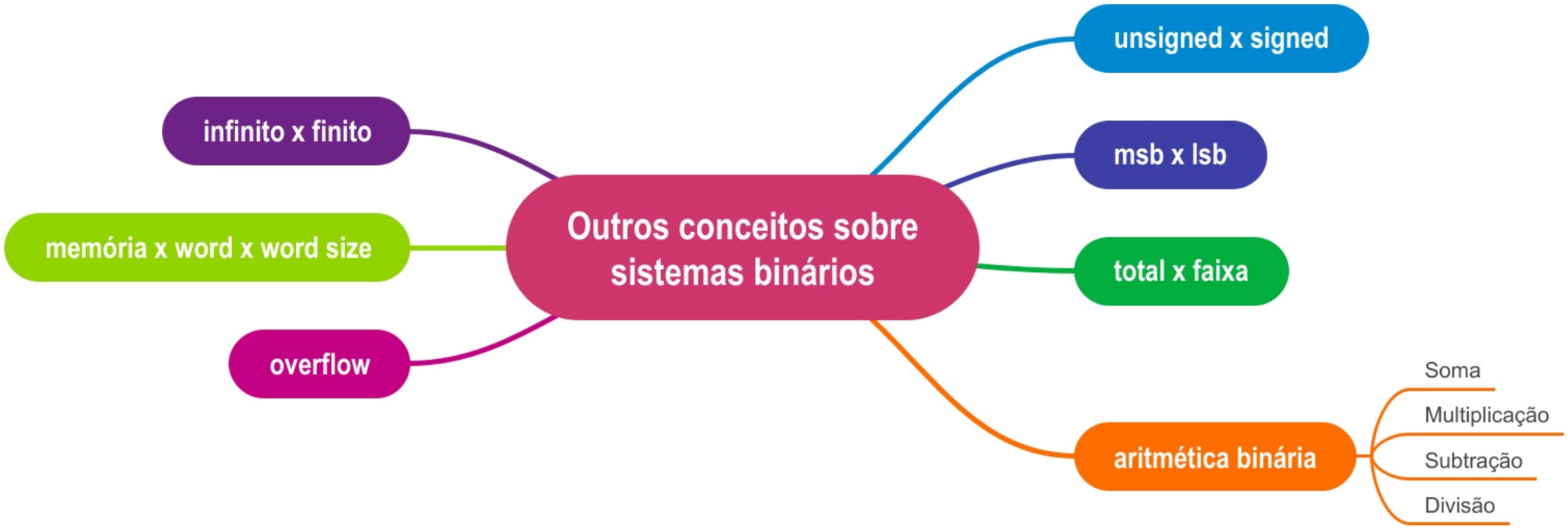
overflow.c - GNU Emacs at cosmos

```c
#include <inttypes.h>
#include <math.h>
#include <stdint.h>
#include <stdio.h>

int main(void)
{
    uint8_t x = 200;
    uint8_t y = 100;
    uint8_t z = x + y;

    printf("%" PRIu8 " + %" PRIu8 " = %" PRIu8 "\n", x, y, z);

    return 0;
}
```

```
[abrantesasf@cosmos ~/cr6100b]$ ./overflow
200 + 100 = 44
```

$$\begin{array}{rll} & 11001000 & (200) \\ + & 01100100 & (100) \\ \hline \end{array}$$

Em resumo


Outros conceitos sobre sistemas binários
unsigned x signed
msb x lsb
total x faixa
aritmética binária
Soma
Multiplicação
Subtração
Divisão
infinito x finito
memória x word x word size
overflow