01 - (Re)Apresentação do curso e representação de dados

Igor Montagner

2020/2

Apresentação do curso

Visão geral

O curso tem duas grandes partes:

1. Arquitetura de computadores: assembly e engenharia reversa de programas em *C*
2. Sistemas operacionais: processos, entrada e saída e programação concorrente

Objetivo de aprendizagem

Ao final da disciplina o aluno será capaz de:

1. Explicar a representação binária de tipos numéricos essenciais e sua manipulação aritmética e lógica
2. Compreender programas em nível de máquina
3. Otimizar programas através da aplicação do conhecimento do funcionamento do hardware e do sistema operacional
4. Entender a hierarquia de memória e como se relaciona com o modelo de memória virtual
5. Explicar como um processo funciona e como tratar o fluxo de controle de exceções
6. Entender programação concorrente e mecanismos de sincronismo para a construção de programas concorrentes corretos e de alto desempenho.

Objetivos

Este curso;

- **Sistemas de Hardware e Software**: foca em como um programa se comunica com o SO do ponto de vista do programador e como isto impacta seu desempenho.

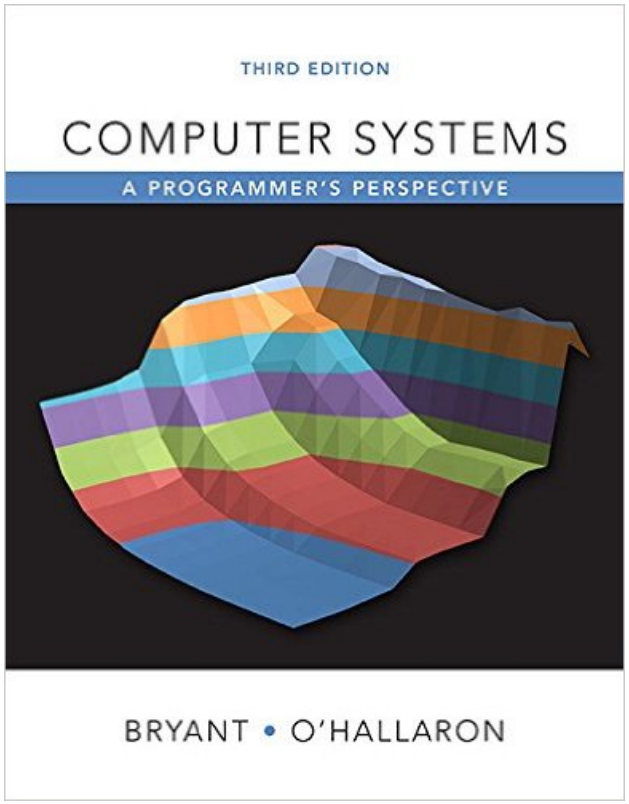

Figure 1: Computer Systems - A Programmers Perspective - Bryant

Horários

Aulas:

- SEG: 13:30 - 15:30
- QUI: 15:45 - 17:45

Atendimento:

- TER: 14:00 - 15:30

Avaliações

Duas provas nas semanas de provas. Atividades de acompanhamento durante o semestre.

Trabalhos práticos:

1. Bomblab - laboratório de engenharia reversa
2. POSIXlab - conceitos de sistemas operacionais
3. Threadlab - operações assíncronas e paralelas usando threads

Atividades de acompanhamento

- *Labs de C*: continuação do mutirão
- exercícios para entrega

Avaliações

Nota final

$$NF = P * 0.45 + L * 0.4 + AT * 0.05 + C * 0.1$$

Importante! Aprovação:

- média de provas (P) > 5
- média de labs (L) > 5
- conceito $\geq$ **D** em todos os labs
- Atividades e Labs de C não são obrigatórios

Ferramentas do curso

- Ubuntu Linux (18.04) **64bits**
- Compilador gcc8 e ferramentas relacionadas de build/debug (gdb, valgrind, objdump, string, etc)

Representação de dados na CPU

O bit

Toda informação é codificada como um *bit*, que pode ser *0* ou *1*

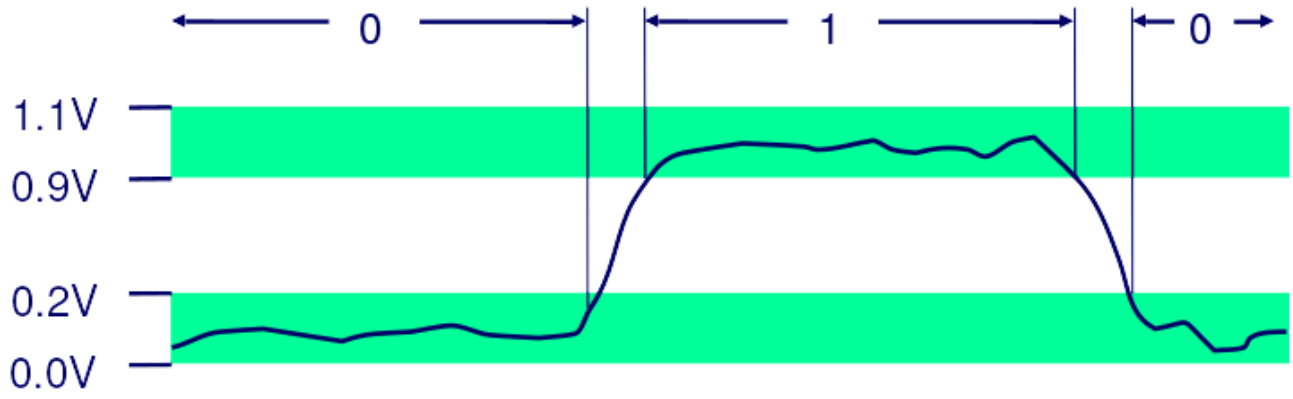

Figure 2: Codificação de sinal como bit

Menor unidade endereçável é o *byte* = *8 bits*.

O bit

Toda informação é codificada como um *bit*, que pode ser *0* ou *1*

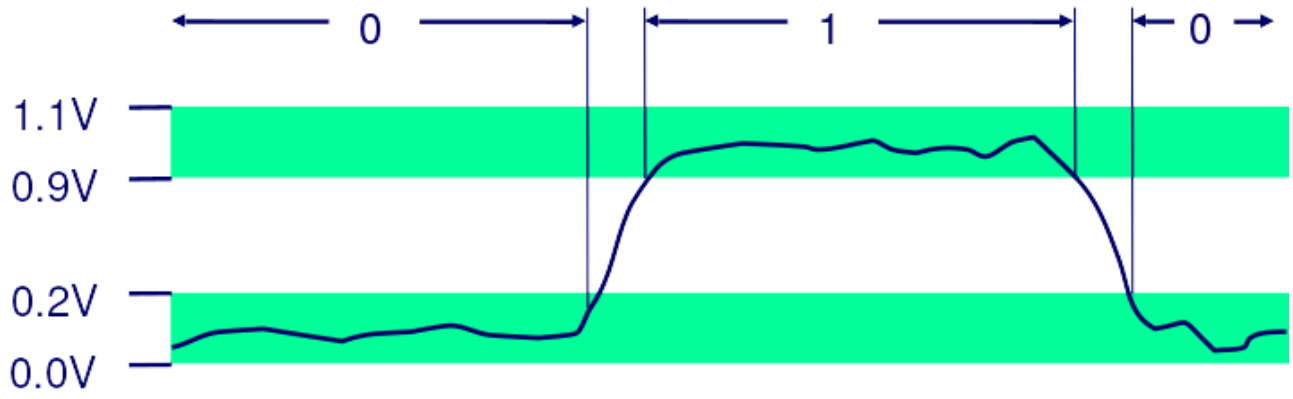

Figure 2: Codificação de sinal como bit

Menor unidade endereçável é o *byte* = *8 bits*.

- Inteiros, Ponto Flutuante, Caracteres (Dados do usuário);
- Código de máquina, Endereços de memória;

O bit

Toda informação é codificada como um *bit*, que pode ser *0* ou *1*

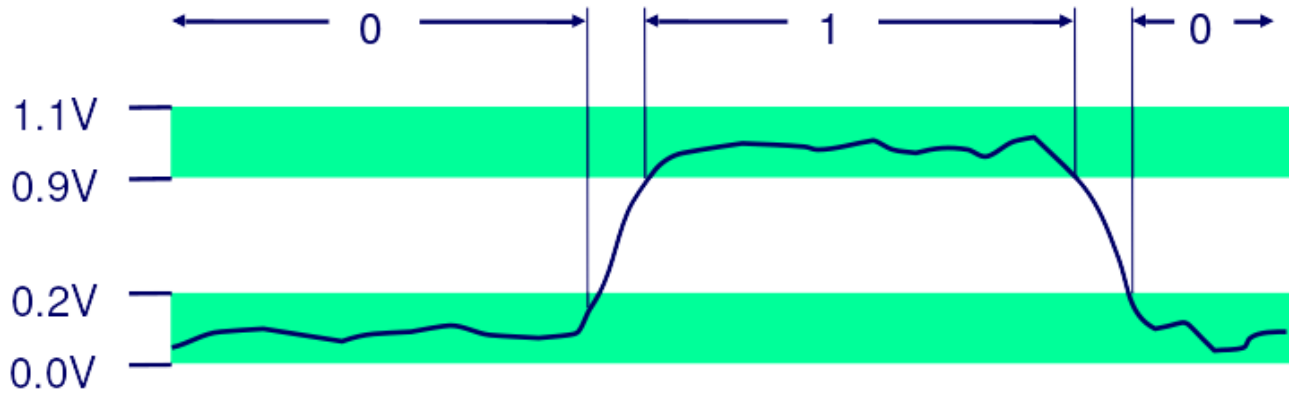

Figure 2: Codificação de sinal como bit

Menor unidade endereçável é o *byte* = *8 bits*.

- Inteiros, Ponto Flutuante, Caracteres (Dados do usuário);
- Código de máquina, Endereços de memória;

Não é possível distinguir o tipo de dado a partir da memória!

Números binários

Precisamos representar números usando somente 2 dígitos!

Base decimal:

$$1234 = 4 \times 10^0 + 3 \times 10^1 + 2 \times 10^2 + 1 \times 10^3$$

Cada dígito multiplica uma potência de 10. O dígito mais significativo é o 1. O menos significativo é o 4.

Números binários

Precisamos representar números usando somente 2 dígitos!

Base decimal:

$$1234 = 4 \times 10^0 + 3 \times 10^1 + 2 \times 10^2 + 1 \times 10^3$$

Cada dígito multiplica uma potência de 10. O dígito mais significativo é o 1. O menos significativo é o 4.

Base binária:

$$11001 = 1 \times 2^0 + 0 \times 2^1 + 0 \times 2^2 + 1 \times 2^3 + 1 \times 2^4$$

Cada dígito multiplica uma potência de 2! Tudo continua igual!

Números binários - Exercício

Converta o número abaixo para decimal

11001110

Números binários - Exercício

Converta o número abaixo para decimal

$$11001110$$

Resposta: $\$206 = 2 + 4 + 8 + 64 + 128\ \$$

Números binários

A conversão decimal $\Rightarrow$ é feita usando divisões sucessivas! Cada dígito é o resto de uma divisão por 2.

147

Números binários

A conversão decimal $\Rightarrow$ é feita usando divisões sucessivas! Cada dígito é o resto de uma divisão por 2.

$$\begin{array}{r} 147 \\ 73 \end{array}$$

Números binários

A conversão decimal $\Rightarrow$ é feita usando divisões sucessivas! Cada dígito é o resto de uma divisão por 2.

$$\begin{array}{rr} 147 & 1 \\ 73 & \end{array}$$

Números binários

A conversão decimal $\Rightarrow$ é feita usando divisões sucessivas! Cada dígito é o resto de uma divisão por 2.

147	1
73	
36	

Números binários

A conversão decimal $\Rightarrow$ é feita usando divisões sucessivas! Cada dígito é o resto de uma divisão por 2.

147	1
73	1
36	

Números binários

A conversão decimal $\Rightarrow$ é feita usando divisões sucessivas! Cada dígito é o resto de uma divisão por 2.

147	1
73	1
36	
18	

Números binários

A conversão decimal $\Rightarrow$ é feita usando divisões sucessivas! Cada dígito é o resto de uma divisão por 2.

147	1
73	1
36	0
18	

Números binários

A conversão decimal $\Rightarrow$ é feita usando divisões sucessivas! Cada dígito é o resto de uma divisão por 2.

147	1
73	1
36	0
18	
9	

Números binários

A conversão decimal $\Rightarrow$ é feita usando divisões sucessivas! Cada dígito é o resto de uma divisão por 2.

147	1
73	1
36	0
18	0
9	

Números binários

A conversão decimal $\Rightarrow$ é feita usando divisões sucessivas! Cada dígito é o resto de uma divisão por 2.

147	1
73	1
36	0
18	0
9	
4	

Números binários

A conversão decimal $\Rightarrow$ é feita usando divisões sucessivas! Cada dígito é o resto de uma divisão por 2.

147	1
73	1
36	0
18	0
9	1
4	

Números binários

A conversão decimal $\Rightarrow$ é feita usando divisões sucessivas! Cada dígito é o resto de uma divisão por 2.

147	1
73	1
36	0
18	0
9	1
4	
2	

Números binários

A conversão decimal $\Rightarrow$ é feita usando divisões sucessivas! Cada dígito é o resto de uma divisão por 2.

147	1
73	1
36	0
18	0
9	1
4	0
2	

Números binários

A conversão decimal $\Rightarrow$ é feita usando divisões sucessivas! Cada dígito é o resto de uma divisão por 2.

147	1
73	1
36	0
18	0
9	1
4	0
2	
1	

Números binários

A conversão decimal $\Rightarrow$ é feita usando divisões sucessivas! Cada dígito é o resto de uma divisão por 2.

147	1
73	1
36	0
18	0
9	1
4	0
2	0
1	

Números binários

A conversão decimal $\Rightarrow$ é feita usando divisões sucessivas! Cada dígito é o resto de uma divisão por 2.

147	1
73	1
36	0
18	0
9	1
4	0
2	0
1	1

Resultado final: $1001\ \ 0011_2$

Números binários

Converta para binário: 211

Números binários

Converta para binário: 211

211	1
105	1
52	0
26	0
13	1
6	0
3	1
1	1

Resposta: 1101 0011

Arquitetura de computadores - 0

CPUs operam em inteiros de tamanho **fixo**. Abaixo os tamanhos para a arquitetura **x64** (ou **x86_64**).

Tamanho	Tipo
1 byte	char
2 bytes	short
4 bytes	int
8 bytes	long

Arquitetura de computadores - 0

CPUs operam em inteiros de tamanho **fixo**. Abaixo os tamanhos para a arquitetura **x64** (ou **x86_64**).

Tamanho	Tipo	Qtd números
8 bits	char	
16 bits	short	
32 bits	int	
64 bits	long	

Arquitetura de computadores - 0

CPUs operam em inteiros de tamanho **fixo**. Abaixo os tamanhos para a arquitetura **x64** (ou **x86_64**).

Tamanho	Tipo	Qtd números
8 bits	char	$256 = 2^8$
16 bits	short	
32 bits	int	
64 bits	long	

Arquitetura de computadores - 0

CPUs operam em inteiros de tamanho **fixo**. Abaixo os tamanhos para a arquitetura **x64** (ou **x86_64**).

Tamanho	Tipo	Qtd números
8 bits	char	$256 = 2^8$
16 bits	short	$65536 = 2^{16}$
32 bits	int	
64 bits	long	

Arquitetura de computadores - 0

CPUs operam em inteiros de tamanho **fixo**. Abaixo os tamanhos para a arquitetura **x64** (ou **x86_64**).

Tamanho	Tipo	Qtd números
8 bits	char	$256 = 2^8$
16 bits	short	$65536 = 2^{16}$
32 bits	int	$4294967296 = 2^{32}$
64 bits	long	

Arquitetura de computadores - 0

CPUs operam em inteiros de tamanho **fixo**. Abaixo os tamanhos para a arquitetura **x64** (ou **x86_64**).

Tamanho	Tipo	Qtd números
8 bits	char	$256 = 2^8$
16 bits	short	$65536 = 2^{16}$
32 bits	int	$4294967296 = 2^{32}$
64 bits	long	$18446744073709551616 = 2^{64}$

Números hexadecimais

Não é prático visualizar um número com muitos bits...

1001110011101110

1001110111101110

Números hexadecimais

E agora?

$0x9CEE$

$0x9DEE$

Base hexadecimal: cada dígito multiplica uma potência de 16.

Conversões de/para decimal: mesma técnica usada em binário, mas multiplicando/dividindo por 16.

Números hexadecimais $\Leftrightarrow$ binário

Agrupamos grupos de 4 bits em um só dígito (da direita para a esquerda)

Binário	Hexa	Binário	Hexa
0000	0x0	1000	0x8
0001	0x1	1001	0x9
0010	0x2	1010	0xA
0011	0x3	1011	0xB
0100	0x4	1100	0xC
0101	0x5	1101	0xD
0110	0x6	1110	0xE
0111	0x7	1111	0xF

Números hexadecimais $\Leftrightarrow$ binário

Agrupamos grupos de 4 bits em um só dígito (da direita para a esquerda)

Binário	Hexa	Binário	Hexa
0000	0x0	1000	0x8
0001	0x1	1001	0x9
0010	0x2	1010	0xA
0011	0x3	1011	0xB
0100	0x4	1100	0xC
0101	0x5	1101	0xD
0110	0x6	1110	0xE
0111	0x7	1111	0xF

Quantos dígitos hexadecimais tem um `short`?

Números hexadecimais $\Leftrightarrow$ binário

Agrupamos grupos de 4 bits em um só dígito (da direita para a esquerda)

Binário	Hexa	Binário	Hexa
0000	0x0	1000	0x8
0001	0x1	1001	0x9
0010	0x2	1010	0xA
0011	0x3	1011	0xB
0100	0x4	1100	0xC
0101	0x5	1101	0xD
0110	0x6	1110	0xE
0111	0x7	1111	0xF

Quantos dígitos hexadecimais tem um `short`? $4 = 16/4$.

Números hexadecimais - Exercício

Converta para decimal:

`0xB9`:

Números hexadecimais - Exercício

Converta para decimal:

`0xB9`: 185

Números hexadecimais - Exercício

Converta para binário:

`0xB9`:

Números hexadecimais - Exercício

Converta para binário:

`0xB9`: 1011 1001

Números hexadecimais - Exercício

Converta para hexadecimal

1. 189 :
2. 1001 1110:

Use a mesma técnica da conversão para binário, mas divida por 16.

Números hexadecimais - Exercício

Converta para hexadecimal

1. 189 : ‘0xBD‘
2. 1001 1110:

Use a mesma técnica da conversão para binário, mas divida por 16.

Números hexadecimais - Exercício

Converta para hexadecimal

1. 189 : ‘0xBD‘
2. 1001 1110: ‘0x9E‘

Use a mesma técnica da conversão para binário, mas divida por 16.

Inteiros de tamanho fixo

Limites

Vimos anteriormente quantos inteiros podemos representar em cada tipo.

Tamanho	Tipo	Qtd números
8 bits	char	$256 = 2^8$
16 bits	short	$65536 = 2^{16}$
32 bits	int	$4294967296 = 2^{32}$
64 bits	long	$18446744073709551616 = 2^{64}$

1. **Como representar números negativos?**
2. **É possível converter de um tipo para outro? Se sim, como?**

Inteiros sem sinal

Números representáveis usando os tipos inteiros sem sinal!

Tipo	Menor(Umin)	Maior(Umax)
char	0	255
short	0	65535
int	0	$2^{32} - 1$
long	0	$2^{64} - 1$

- Em *C* são representados pelos tipo `unsigned char/short/int/long`.
- Casting de um tipo maior para um menor ignora os bits mais significativos
- Casting de um tipo menor para um maior preenche os bits "novos" com 0

Inteiros sem sinal

Qual é a saída do código abaixo?

```
unsigned char c;
unsigned int i = 347;
unsigned long l;

c = (unsigned char) i;
l = (unsigned long) i;

printf("%u %u %ul\n", c, i, l);
```

Inteiros sem sinal

Qual é a saída do código abaixo?

```
unsigned char c;
unsigned int i = 347;
unsigned long l;

c = (unsigned char) i;
l = (unsigned long) i;

printf("%u %u %ul\n", c, i, l);
```

91 347 347

Inteiros com sinal

Two's complement: o bit mais significativo é uma potência negativa! Dado número binário b com n dígitos,

$$dec(b) = \sum_{i=0}^{n-2} b_i 2^i - b_{n-1} 2^{n-1}$$

- Em *C* são representados pelos tipo `char/short/int/long`.
- Sinal é uma convenção. Cast de signed $\Leftrightarrow$ unsigned não muda os bits!

Inteiros com sinal

Números representáveis usando os tipos inteiros sem sinal!

Tipo	Menor(Tmin)	Maior(Tmax)
char	-128	127
short	-32768	32767
int	-2^{31}	$2^{31} - 1$
long	-2^{63}	$2^{63} - 1$

Temos mais negativos que positivos!

Inteiros com sinal

Exemplos:

- Converter para decimal: $1001\ \ 1100 \rightarrow$

Inteiros com sinal

Exemplos:

- Converter para decimal: 1001 1100 $\rightarrow$

$$4 + 8 + 16 - 128 = -100$$

- Converter para binário (8bits) -49

Inteiros com sinal

Exemplos:

- Converter para decimal: 1001 1100 $\rightarrow$

$$4 + 8 + 16 - 128 = -100$$

- Converter para binário (8bits) -49 $\rightarrow$ 1100 1111

$$-128 + x = -49 \rightarrow x = 128 - 49 \rightarrow x = 79$$

0100 1111 = 79 em binário, como o número é negativo o bit mais significativo é 1.

Inteiros com sinal - Exercícios

- Converta para decimal: 1101 1101 $\Rightarrow$
- Converta para binário (8bits): -13

Inteiros com sinal - Exercícios

- Converta para decimal: 1101 1101 $\Rightarrow -35$
- Converta para binário (8bits): -13

$$-128 + x = -13 \rightarrow x = 115$$

115 em binário: 0111 0011, **Resposta**: 1111 0011

Inteiros com sinal

- Casting de tipo maior para tipo menor ignora os bits mais significativos

- Casting de tipo menor para tipo maior preenche os bits "novos" com o bit mais significativo (0 se positivo, 1 se negativo)

Número com 8 bits: 1001 1100 (-100)

Número com 16 bits: 1111 1111 1001 1100 (-100)

Inteiros com sinal

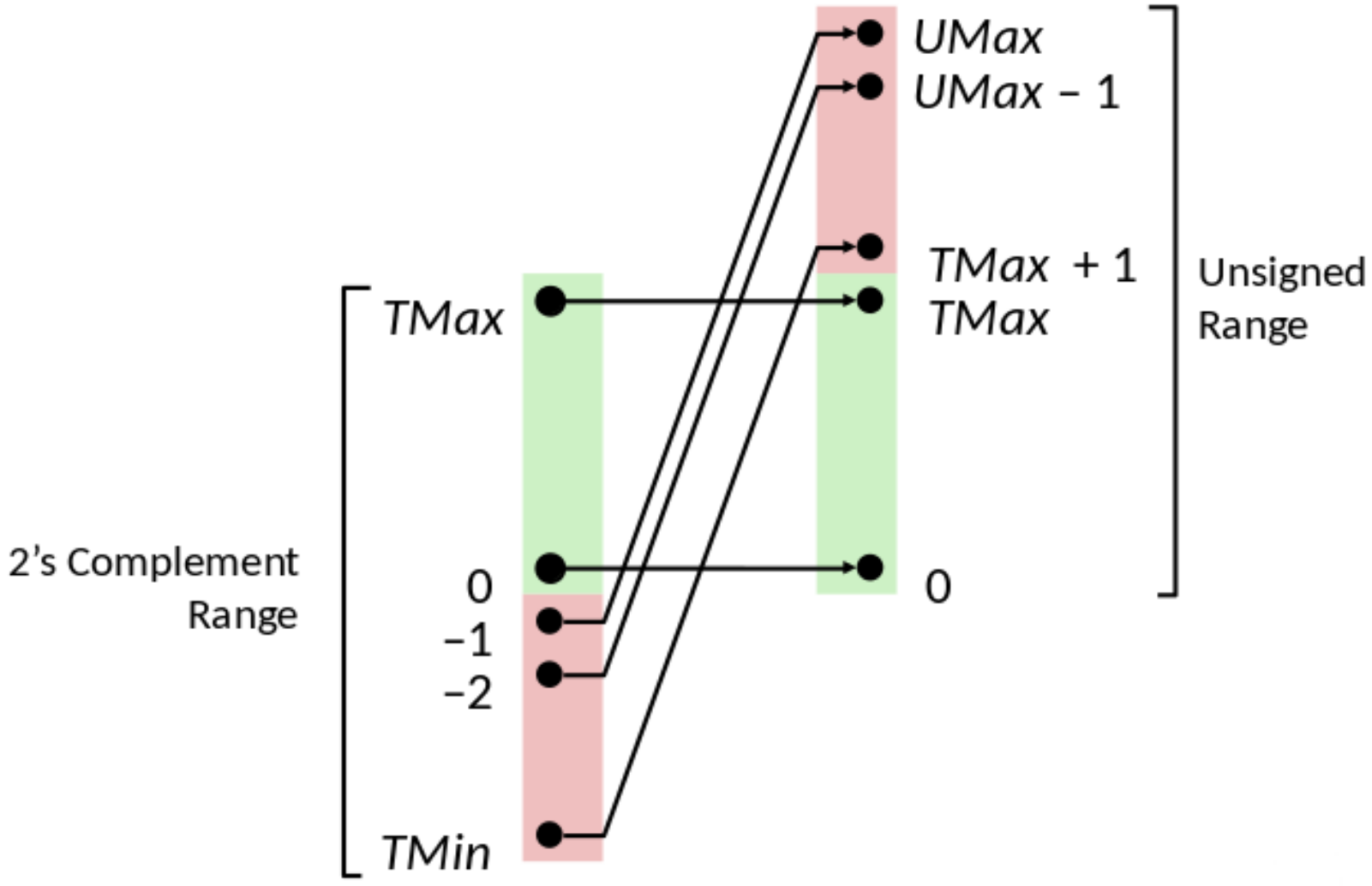

Figure 3: Conversão signed $\Leftrightarrow$ unsigned é segura somente em parte dos intervalo

Inteiros com sinal

Qual a saída do programa abaixo?

```
unsigned char uc = 248;
char c;

char c2 = 100
unsigned char uc2;

c = uc;
uc2 = c2;

printf("%d %u\n", c, uc2);
```

Inteiros com sinal

Qual a saída do programa abaixo?

```
unsigned char uc = 248;
char c;

char c2 = 100
unsigned char uc2;

c = uc;
uc2 = c2;

printf("%d %u\n", c, uc2);
```

-8 100

Inteiros com e sem sinal

Qual a saída do programa abaixo?

```
unsigned int size = 2;
int counter = 10;

while (counter - size >= 0) {
    printf("%d\n", counter);
    counter -= size;
}
```

Inteiros com e sem sinal

Qual a saída do programa abaixo?

```
unsigned int size = 2;
int counter = 10;

while (counter - size >= 0) {
    printf("%d\n", counter);
    counter -= size;
}
```

loop infinito!

Operações bit a bit

Operações bit a bit entre dois inteiros

Operações entre dois inteiros (`a OP b`)

- `&` faz o `E` lógico entre os bits
- `|` faz o `OU` lógico entre os bits
- `^` bit i vale `1` se $a_i \neq b_i$.

Operações unárias

- `~` inverte o valor de cada bit

Operações booleanas entre inteiros

Cada inteiro é considerado como um booleano (True se $\neq 0$ e False se $= 0$).

- `&&` faz E entre os valores booleanos
- `||` faz OU entre os valores booleanos
- `!` nega o valor booleano

Importante!! Qual é a saída do código abaixo?

```
int v = 10;
printf("%d %d\n", !!v, ~~v);
```

Operações booleanas entre inteiros

Cada inteiro é considerado como um booleano (True se $\neq 0$ e False se $= 0$).

- `&&` faz E entre os valores booleanos
- `||` faz OU entre os valores booleanos
- `!` nega o valor booleano

Importante!! Qual é a saída do código abaixo?

```
int v = 10;
printf("%d %d\n", !!v, ~~v);
```

1 10

Operações bit a bit

- `a << b` desloca os bits de `a` à esquerda `b` vezes

Operações bit a bit

- `a << b` desloca os bits de `a` à esquerda `b` vezes ($a \times 2^b$)

Operações bit a bit

- `a << b` desloca os bits de `a` à esquerda `b` vezes ($a \times 2^b$)
- `a >> b` deslocal os bits de `a` à esquerda `b` vezes

Operações bit a bit

- `a << b` desloca os bits de `a` à esquerda `b` vezes ($a \times 2^b$)

- `a >> b` deslocal os bits de `a` à esquerda `b` vezes ($a/2^b$)
  - Shift lógico: completa os bits mais significativos com `0`
  - Shift aritmético: copia o bits mais significativo (mantém sinal!)

Importante em números negativos o arredondamento é feito para baixo!

$$(1001\ \ 1011) >> 1 = \quad 1100\ \ 1101$$
$$-101 \qquad\qquad\qquad\qquad -51$$

Operações bit a bit

- `a << b` desloca os bits de `a` à esquerda `b` vezes ($a \times 2^b$)

- `a >> b` deslocal os bits de `a` à esquerda `b` vezes ($a/2^b$)
  - unsigned: completa os bits mais significativos com `0`
  - signed: copia o bits mais significativo (mantém sinal!)

Cuidado! Em números negativos o arredondamento é feito para baixo!

$$(1001\ \ 1011) >> 1 = \begin{matrix} 1100\ \ 1101 \\ -101 \end{matrix} \qquad \begin{matrix} \\ -51 \end{matrix}$$

Exercícios

- Entrega *Números na CPU* disponível no repositório da disciplina.
- Data de entrega *02/03*.