# EA871 – Laboratório de Programação Básica de Sistemas Digitais

Atividade 06 – 2° Semestre de 2023

**1. Objetivos**

- Trabalhar simultaneamente com o receptor e o transmissor de uma UART, usando interrupções para receber comandos de um computador e trocar mensagens.
- Consolidar o conceito de interrupção.
- Introduzir o uso de *buffers* para armazenamento temporário de dados ainda não processados.
- Exercitar o uso de máquinas de estado para modelagem e solução de problemas sequenciais.

**2. Buffer circular**

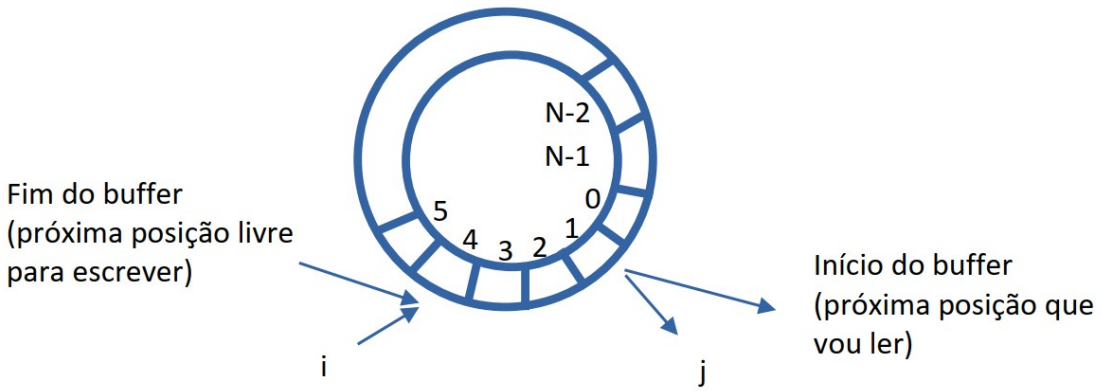
Vetores podem ser usados de muitas maneiras em programas de computador. Uma dessas maneiras se chama ***buffer* circular**. O *buffer* circular é uma estrutura de dados que é usada para manejar filas de dados que são recebidos de algum lugar no seu programa. Em sistemas microcontrolados, *buffers* circulares são usados para armazenar dados recebidos até que eles possam ser processados.

O *buffer* circular parte da ideia de que vamos chamar uma função "adicionar_buffer()" que adiciona um elemento no nosso vetor. Para isso, precisamos de um índice i, que aponta para a próxima posição livre do vetor, e que é incrementado a cada chamada de "adicionar_buffer()".

Outra possibilidade de ação sobre o *buffer* circular é "remover um elemento". Nesse caso, usamos outra função, "remover_buffer()", que retorna o primeiro elemento da fila – isto é, o elemento mais antigo que foi adicionado. Isso significa que precisamos também de um índice j, que aponta para o elemento mais antigo do *buffer* que ainda não foi processado. Este índice deve ser atualizado sempre que a função remover_buffer() for chamada.

Quando um elemento é removido do *buffer*, o espaço em que ele estava fica livre, isto é, pode ser usado para adicionar um novo elemento. Então, a função adicionar_buffer() deve ser capaz de manejar o índice i de forma a voltar ao começo do vetor e verificar se aquela posição está livre.

Uma possível maneira de pensar o *buffer* circular é imaginar que o vetor de dados está posicionado sobre um círculo. Para verificar se você entendeu bem, identifique na figura abaixo as posições do *buffer* circular que contém dados válidos, isto é, dados que foram recebidos, mas ainda não processados:




**2. Atividade de aula**

Resolva os exercícios a seguir usando as seguintes configurações da UART:

Especificações do protocolo:

- Baud rate igual a 115.200 bps.
- Número de bits de dados por frame igual a 8;
- Sem bit de paridade;
- Uso de dois bits de parada;

Especificações adicionais (particulares para a USART do ATmega328P):

- Modo double-speed ativado;
- Modo multi-processador desabilitado;
- Modo assíncrono de funcionamento da USART;

### Exercício 1: Transmissão usando interrupção USART Transmit Complete

Programe a placa de desenvolvimento de modo que o sistema repita ininterruptamente a seguinte sequência: 1) enviar a mensagem `msg_TXC0`; 2) aguardar 500ms.

```
char msg_TXC0 [] = "Transmissao serial utilizando a interrupcao USART Transmit Complete. \n\n";
```

A mensagem deve ser transmitida integralmente utilizando-se a interrupção associada ao evento "transmissão completa" (USART Transmit Complete) para garantir que o próximo caractere só seja enviado após o término do atual.

### Exercício 2: Transmissão usando interrupção USART Data Register Empty

Repita o exercício anterior substituindo a mensagem transmitida por:

```
char msg_UDRE0 [] = "Transmissao serial utilizando a interrupcao USART Data Register Empty. \n\
n";
```

Utilize a interrupção associada ao evento "*buffer* de transmissão vazio" (USART Data Register Empty) para garantir que o próximo caractere só seja enviado após o término do atual. Ao concluirmos esse exercício, teremos exercitado o uso das duas interrupções associadas a eventos de transmissão da UART do ATmega328P.

### Exercício 3: Transmissão e recepção usando interrupção

Faça um programa que transmita de volta para o computador um caractere recebido que tenha sido digitado no monitor serial, ou seja, um esquema de eco via serial. Utilize interrupções para lidar com os eventos referentes à recepção e à transmissão.

**3. Resumo da atividade para entrega**

O desafio proposto nesta atividade é desenvolver um programa que utilize a UART para criar um sistema de acionamento de um conjunto de 3 LEDs a partir de comandos fornecidos pelo teclado do computador ao qual a placa de desenvolvimento está conectada. Além disso, também vamos utilizar o transmissor da UART para encaminhar mensagens de confirmação e *status*.

**Recepção**

A interrupção do tipo "recepção completa" será explorada nesta atividade para que o microcontrolador ATMega328P receba comandos de controle através do teclado do computador. Uma vez que o microcontrolador pode estar realizando operações que consomem tempo enquanto novos comandos são enviados pelo usuário, vamos utilizar a estrutura de um *buffer* circular para armazenar temporariamente a

sequência de comandos recebidos. O processamento dos comandos, então, se dará a partir do primeiro elemento da fila (ou seja, do comando mais antigo armazenado no *buffer*). Por simplicidade, vamos trabalhar com um *buffer* circular relativamente pequeno, de tamanho igual a 5 (cinco). A tabela abaixo mostra os possíveis comandos e os respectivos efeitos sobre o conjunto de LEDs.

| Operação | Comando Enviado |
|---|---|
| **Pisca todos os LEDs ininterruptamente segundo a sequência: acende todos LEDs por 500ms; apaga todos os LEDs por 500ms** | 0 |
| **Varredura com um LED aceso (vai e volta), repetindo ininterruptamente a sequência: 1-0-0; 500ms; 0-1-0; 500ms; 0-0-1; 500ms; 0-1-0; 500ms. (PC3-PC4-PC5; 0, LED apagado; 1, LED aceso)** | 1 |
| **Varredura com um LED apagado, repetindo ininterruptamente a sequência: 0-1-1; 500ms; 1-0-1; 500ms; 1-1-0; 500ms; 1-0-1; 500ms;** | 2 |

A rotina de serviço associada à recepção deve, portanto, inserir cada novo comando na próxima posição livre do *buffer* circular, caso haja alguma.
Com isso, sempre que houver comandos disponíveis no *buffer* circular, o programa deve processar o comando mais antigo. Além, disso, ao executar um comando, o sistema deve enviar uma mensagem de confirmação pela porta serial, **uma única vez**, imediatamente após o início da execução. Se o comando não for válido, ou seja, se o caractere recebido for diferente de 0, 1 ou 2, o programa deve encaminhar uma mensagem de erro e permanecer repetindo o último comando válido. A tabela abaixo apresenta as mensagens referentes a cada comando recebido.

| Comando | Mensagem |
|---|---|
| **0** | `"Comando: Todos os LEDs piscando\n"` |
| **1** | `"Comando: Varredura com um LED aceso\n"` |
| **2** | `"Comando: Varredura com um LED apagado\n"` |
| **Qualquer outro caractere** | `"Comando incorreto\n"` |

## Transmissão

Conforme já descrito, cada vez que um comando for processado, uma mensagem de texto deve ser retornada pela porta serial, segundo a tabela já apresentada. Para enviar cada mensagem, *i.e.*, cada sequência de caracteres, vamos novamente utilizar o mecanismo de interrupção. No caso da transmissão, há duas opções de eventos que disparam interrupções: (1) "transmissão completa" ou (2) "*buffer* de transmissão vazio".
Além disso, sempre que não houver comandos a processar no *buffer* circular (ou seja, o *buffer* está vazio), a seguinte mensagem `"Vazio!\n"` deve ser enviada a cada 500ms.

**Cuidado:**

- A interrupção associada ao evento "*buffer* de transmissão vazio" é disparada continuamente caso o *buffer* esteja limpo. Sendo assim, é preciso desligar esta interrupção ao terminar de transmitir uma sequência completa de caracteres; caso contrário, uma nova interrupção será gerada mesmo se não quisermos transmitir outra mensagem.

**Lembrete:** caso as rotinas de serviço de interrupção tenham de acessar ou modificar conteúdos de variáveis do programa principal, é importante declará-las como variáveis globais com o qualificador `volatile` para assegurar o correto funcionamento.

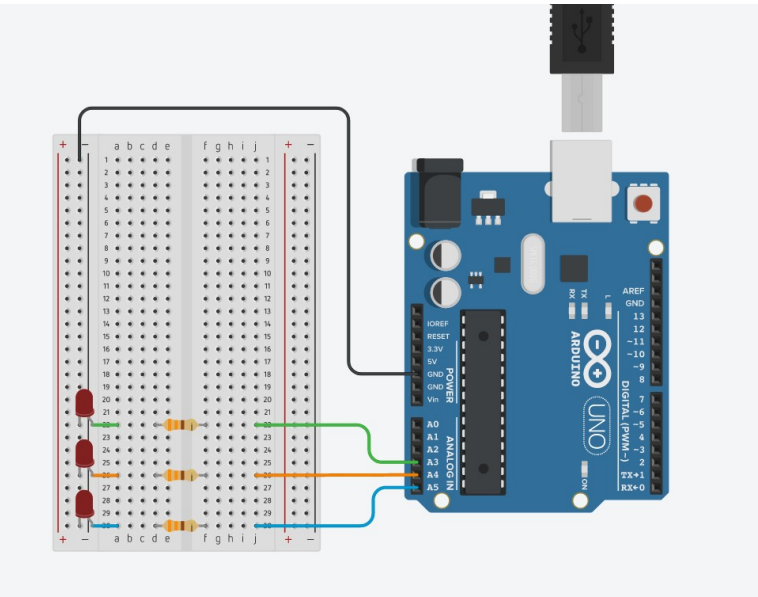
**Especificações da USART:**

- Velocidade de transmissão normal (i.e., modo *double-speed* desativado);

- Modo de transmissão multi-processador desabilitado;
- Número de bits de dados por *frame* igual a 8;
- Modo assíncrono de funcionamento da USART;
- Sem bits de paridade;
- Uso de um bit de parada;
- *Baud rate* igual a 19.200 bps.

## Circuito

Por convenção, vamos empregar os terminais PC3, PC4 e PC5 do microcontrolador para acionar os LEDs. A montagem pode ser acessada no link: **https://www.tinkercad.com/things/hJmsrahXvPS**. Notem que no programa da montagem do link já existem as funções que implementam o *buffer* circular. Podem usá-las e modificá-las do modo que lhes for mais conveniente.



## Instruções para a submissão do trabalho

1) Nos comentários do código-fonte, explique o funcionamento geral do programa e justifique as operações e os valores carregados em todos os registradores. É fundamental explicar o mecanismo de interrupção adotado para transmissão e recepção dos pacotes. Lembre-se que seus comentários fazem parte do relatório. Programas sem comentários terão nota máxima 4. Justificativas de configuração incorretas, mesmo que a configuração esteja certa, serão penalizadas fortemente, principalmente se tiverem sido discutidas em exemplos.
2) Ao final da atividade, salve o código-fonte com nome "seu_ra.txt" (Exemplo: 025304.txt).
3) Faça o upload da sua solução da atividade 6 no Google Classroom.