EC45C - Sistemas Microcontrolados
Turmas: C51A/C51B

# Prática L2 – Portas de Entrada e Saída.

## Objetivos:

- Apresentar as portas de I/O.
- Criar o esquemático de um circuito no Proteus.
- Criar um *firmware* com foco em leitura e escrita de portas de I/O.
- Simular o *firmware* no Proteus.
- Agrupar os arquivos gerados para envio pelo Moodle.

## Interface de Entrada e Saída Digital

Dos 40 pinos do PIC18F4550, 33 deles podem ser utilizados como entradas ou saídas digitais, conforme apresentado na Tabela 1. Nesta aula veremos como configurar e utilizar estes pinos como I/O.

***Tabela 1**: Registradores e tamanho em bits.*

| **Registradores** | **Tamanho** |
|---|---|
| PORTA, TRISA e LATA | 7 *bits* |
| PORTB, TRISB e LATB | 8 *bits* |
| PORTC, TRISC e LATC | 7 *bits* |
| PORTD, TRISD e LATD | 8 *bits* |
| PORTE, TRISE e LATE | 3 *bits* |

## Portas de Entrada e Saída (I/O)

Portas de I/O (input/output) são portas que permitem o fluxo de dados, cujos valores alternam entre níveis lógicos ‘0’ e ‘1’. Por essas portas o microcontrolador é capaz de obter informações do mundo exterior, como também atuar diretamente no controle (ON/OFF) de dispositivos, como, por exemplo, ligar/desligar uma lâmpada, válvula, motor, entre outros.

O PIC18F4550 tem cinco portas disponíveis (A, B, C, D e E). Os pinos associados a elas são multiplexados com diferentes funções de periféricos. Geralmente, quando um determinado periférico é habilitado, o pino relacionado a ele deixa de ser de propósito geral, e passa a desempenhar a função que lhe é concedida.

Cada porta tem três registradores associados à configuração como apresentado na Figura 1:

- **Registrador TRIS:** Configura o sentido do fluxo de dados de uma determinada porta.
- **Registrador PORT:** Escreve/lê o nível dos pinos associados à porta.
- **Registrador LAT:** Armazena o valor do último comando de escrita.

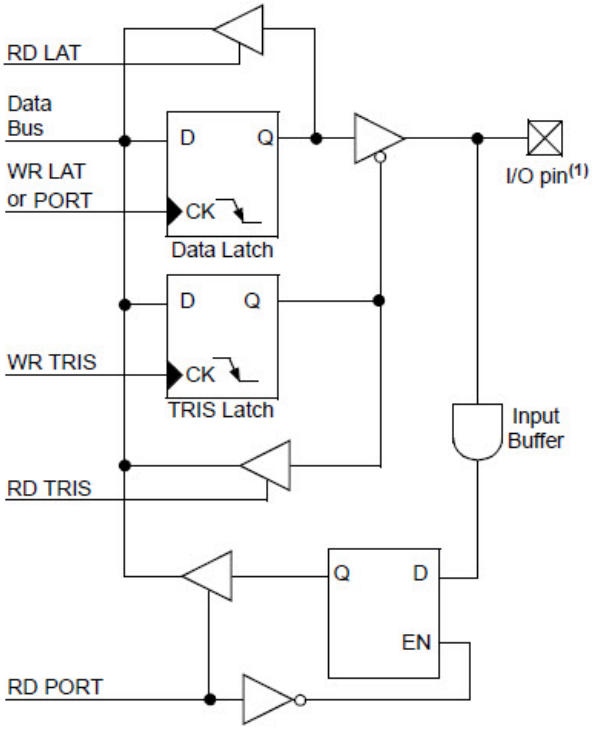


**Figura 1:** Diagrama de operação de uma porta I/O genérica.

Por meio do diagrama é possível verificar que o comando de escrita passa pelo Data Latch antes de realizar a modificação do estado de saída dos pinos.

O Registrador Data Latch (LAT) também é uma memória mapeada e armazena o valor da última operação de escrita, no entanto essa operação não afeta os pinos I/O que estejam configurados como entrada ou sendo utilizados por um determinado periférico. O LAT retém o valor escrito e quando o pino for setado como saída de propósito geral, ele é afetado de acordo com o valor do LAT.

## Sentido do fluxo de Dados das Portas

O sentido do fluxo de dados de uma determinada porta é configurado pelo registrador TRIS. Ele possui 8 bits e cada elemento corresponde à configuração de um determinado pino de I/O. Sendo: ‘0’ – Saída e ‘1’ – Entrada (modo de alta impedância).

### Registradores: TRISA, TRISB, TRISC, TRISD e TRISE

Os nomes dos identificadores de acesso aos 8 bits dos registradores **TRIS** são os próprios nomes dos registradores associados a eles.

**Sintaxe**

**TRIS**x = *valor;*

*valor* = **TRIS**x;

Sendo:

- x: nome da porta (letra maiúscula – A, B, C, D e E);
- *valor:* valor de 8 bits. ‘0’ (Saída) ou ‘1’ (Entrada);

### TRISAbits, TRISBbits, TRISCbits, TRISDbits e TRISEbits

Essas estruturas permitem o acesso de apenas um bit do registrador TRIS.

**Sintaxe**

**TRIS**x**bits.TRIS**xy = *valor_bit;*

*valor_bit* = **TRIS**x**bits.TRIS**xy;

Sendo:

- x: nome da porta (letra maiúscula – A, B, C, D e E);
- y: nome do pino (número – 0 a 7);
- *valor_bit:* valor '0' (Saída) ou '1' (Entrada);

## Controle do estado dos Pinos da Porta

O status dos pinos de uma porta é armazenado no registrador **PORT**. Esse registrador possui um tamanho de 8 bits e é responsável pelas operações de escrita e leitura dos pinos relacionados à porta.

### Registradores: PORTA, PORTB, PORTC, PORTD e PORTE

Os nomes dos identificadores de acesso aos 8 bits dos registradores PORT são os próprios nomes dos registros associados a eles. Para um comando de leitura, o registrador PORT realiza a leitura dos status dos pinos e para um comando de escrita, o valor é enviado para a porta Latch (LAT) que vai modificar os níveis dos pinos I/O configurados como saída.

**Sintaxe:**

**PORT**x = *valor;*

*valor* = **PORT**x;

Sendo:

- x: nome da porta (letra maiúscula – A, B, C, D e E);
- *valor:* valor de 8 bits.

### PORTAbits, PORTBbits, PORTCbits, PORTDbits e PORTEbits

Essas estruturas permitem o acesso de apenas um bit do registrador PORT.

**Sintaxe:**

**PORT**x**bits.R**xy = *valor_bit;*

*valor_bit* = **PORT**x**bits.R**xy;

Sendo:

- x: nome da porta (letra maiúscula – A, B, C, D e E);
- y: nome do pino (número – 0 a 7);
- *valor_bit:* valor '0' ($V_{SS}$) ou '1' ($V_{DD}$).

### Registradores: LATA, LATB, LATC, LATD e LATE

Como visto anteriormente, o Data Latch (LAT) armazena o valor enviado pelo comando de escrita. No entanto, o comando afeta somente os pinos configurados como saída de propósito geral. Logo, o LAT retém o valor do último comando de escrita e modifica o estado do pino assim que for configurado como saída.

Os 8 bits do registrador LAT são acessados por identificadores associados ao mesmo nome do registrador. Para um comando de leitura o registrador LAT realiza a leitura do valor de saída do Data Latch e escreve o valor no registrador PORT; para um comando de escrita, o valor é carregado para o Data Latch que vai modificar os níveis dos pinos I/O configurados como saída.

**Sintaxe:**

**LAT**x = *valor;*

*valor* = **LAT**x;

Sendo:

- x: nome da porta (letra maiúscula – A, B, C, D e E);
- *valor:* valor de 8 bits.

LATAbits, LATBbits, LATCbits, LATDbits e LATEbits

Essas estruturas permitem o acesso de apenas um bit do registrador LAT.

**Sintaxe:**

**LAT**x**bits.R**xy = *valor_bit;*

*valor_bit* = **LAT**x**bits.R**xy;

Sendo:

- x: nome da porta (letra maiúscula – A, B, C, D e E);
- y: nome do pino (número – 0 a 7);
- *valor_bit:* valor '0' ($V_{SS}$) ou '1' ($V_{DD}$).

## Dispositivos conectados aos portais de entrada e saída.

O kit didático XM118 que utilizamos em nossas aulas práticas contém vários dispositivos que podem ser ligados às portas de I/O. As informações de como os pinos estão ligados a estes dispositivos encontram-se no esquemático do kit didático XM118, disponível no Moodle.

## Ligação de chaves e botões com o microcontrolador.

As interfaces de entrada do microcontrolador entendem os sinais de valor $V_{DD}$ como nível lógico "1" e $V_{SS}$ como nível lógico "0". Desta forma, para garantir que estas interfaces recebam os níveis lógicos esperados dos dispositivos externos, fazemos uso de um resistor em série com as chaves em configurações denominas *pull-up* e *pull-down*.

O resistor é chamado de resistor *pull-up* quando este está conectado à tensão de $V_{DD}$ e a chave faz o contato com a tensão comum.

Quando a chave faz contato com a tensão $V_{DD}$ e o resistor com a tensão comum, chamamos este resistor de *pull-down*.

De acordo com as figuras a seguir, é possível notar os níveis de tensão para as duas configurações:

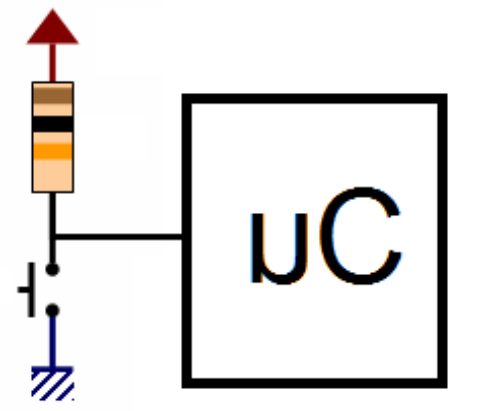


**Figura 2a:** *Pull-Up* com chave aberta
Nível lógico = 1

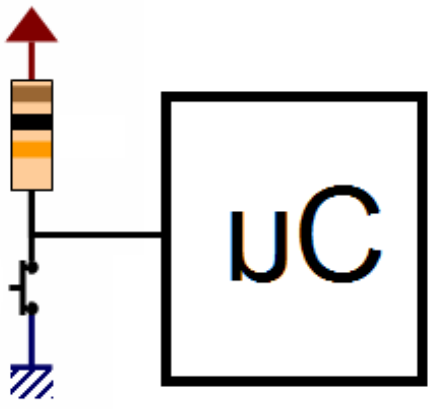


**Figura 2b:** *Pull-Up* com chave fechada
Nível lógico = 0

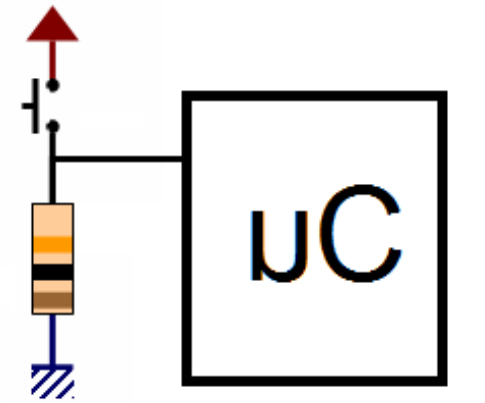


**Figura 2c:** *Pull-Down* com chave aberta
Nível lógico = 0

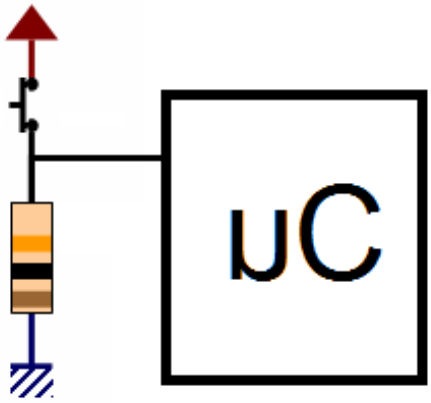


**Figura 2d:** *Pull-Down* com chave fechada
Nível lógico = 1

### O efeito "*bouncing*".

Pela figura 2a percebemos que a tensão na porta do microcontrolador é igual a $V_{DD}$ quando a chave está desligada, pois não há circulação de corrente no circuito, de modo que a queda de tensão em R1 é zero.

Quando a chave é pressionada (figura 2b) uma corrente flui de $V_{DD}$ para o comum, passando pelo resistor. Como não existe nenhuma outra resistência no circuito e a impedância da porta de entrada é alta, toda a tensão fica em cima do resistor. Deste modo a tensão de saída passa a ser zero.

Apesar do funcionamento aparentemente simples, este tipo de circuito apresenta um problema de oscilação do sinal no momento em que a tecla é pressionada. Esta oscilação é conhecida como *bouncing*. A Figura 3 apresenta esse efeito.

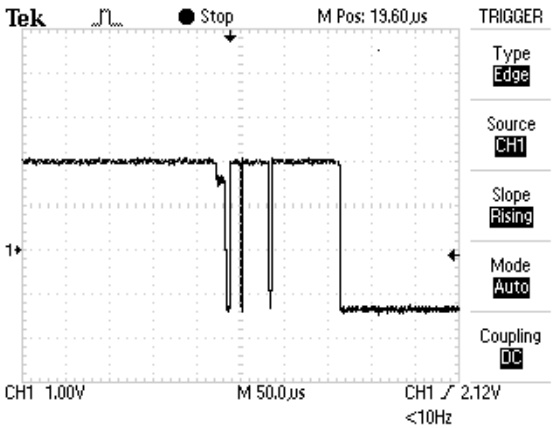


**Figura 3:** Oscilação do sinal no momento do chaveamento.
Fonte: http://www.labbookpages.co.uk/electronics/debounce.html

Estas oscilações indevidas podem gerar acionamentos acidentais, causando mau funcionamento do programa. Para evitar isso podemos utilizar técnicas de *debounce*, por hardware ou software.

Em *software* é possível atribuir um atraso de processamento após o reconhecimento do acionamento da chave e assim aguardar a estabilidade da chave.

As funções da biblioteca <delays.h> podem ser usadas para gerar este atraso. Por exemplo, a função: `Delay1KTCYx(T)` realiza o atraso de 1000 ciclos de clock internos para cada unidade de T. Considerando o uso de um cristal de 20 MHz, temos o clock interno em 5 MHz e o período de um ciclo interno igual 200ns. Desta forma, para T = 1, o atraso é de 1000 ciclos internos, que é igual a 200µs ou 0,2ms.

Em *hardware* é comum incluir um capacitor no circuito, criando assim um filtro RC passa-baixa.

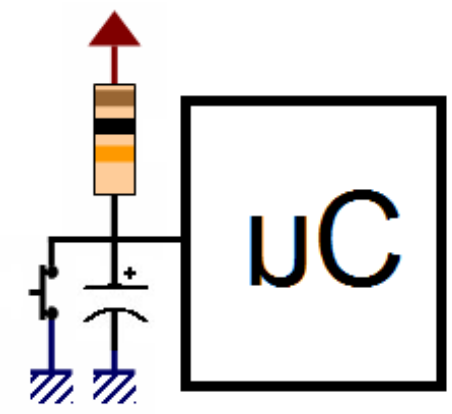


**Figura 4a:** Circuito de *debounce*

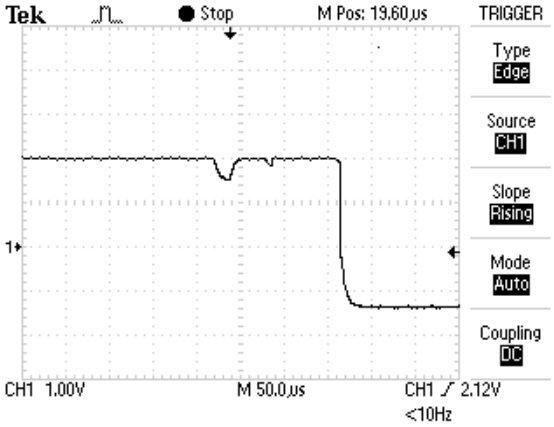


**Figura 4b:** Oscilação do sinal com circuito de *debounce* RC

Mais informações sobre técnicas de *debounce* podem ser acessadas em:
http://www.eng.utah.edu/~cs5780/debouncing.pdf

## Ligação dos LEDs com o microcontrolador.

Da mesma forma que a interface de entrada entende as tensões $V_{DD}$ e $V_{SS}$ como os níveis lógicos 1 e 0, a interface de saída apresenta em suas portas os valores de tensão $V_{DD}$ e $V_{SS}$ para os níveis lógicos 1 e 0 respectivamente.

Ao fazer a ligação do LED de modo que o catodo esteja em contato com o ponto de tensão comum ($V_{SS}$) e o anodo receba a tensão da porta de saída, temos uma ligação que acende o LED quando a tensão da porta de saída tiver nível lógico 1 e apaga o LED quando tiver nível lógico 0.

Ao fazer a ligação do LED de modo que o anodo esteja em contato com o ponto de tensão de 5V ($V_{DD}$) e o catodo receba a tensão da porta de saída, temos uma ligação que acende o LED quando a tensão da porta de saída tiver nível lógico 0 e apaga o LED quando tiver nível lógico 1.

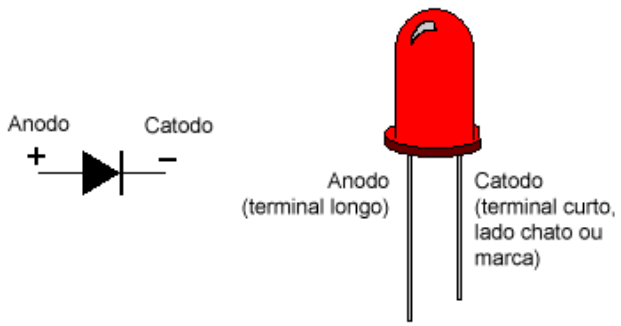


**Figura 5:** Representação do LED.

Obs:
Informações sobre como os LEDs estão ligados ao microcontrolador no kit didático XM118 estão disponíveis no Moodle.
Documentação: Esquema elétrico do kit didático XM118.

## 1- Criar o circuito usando a ferramenta ISIS.

Crie o circuito abaixo e salve no diretório de trabalho com o seguinte nome:
{Disciplina}_{Turma}_{Grupo}_{Número do roteiro}
ex: EC45C_C51A_G1_L2

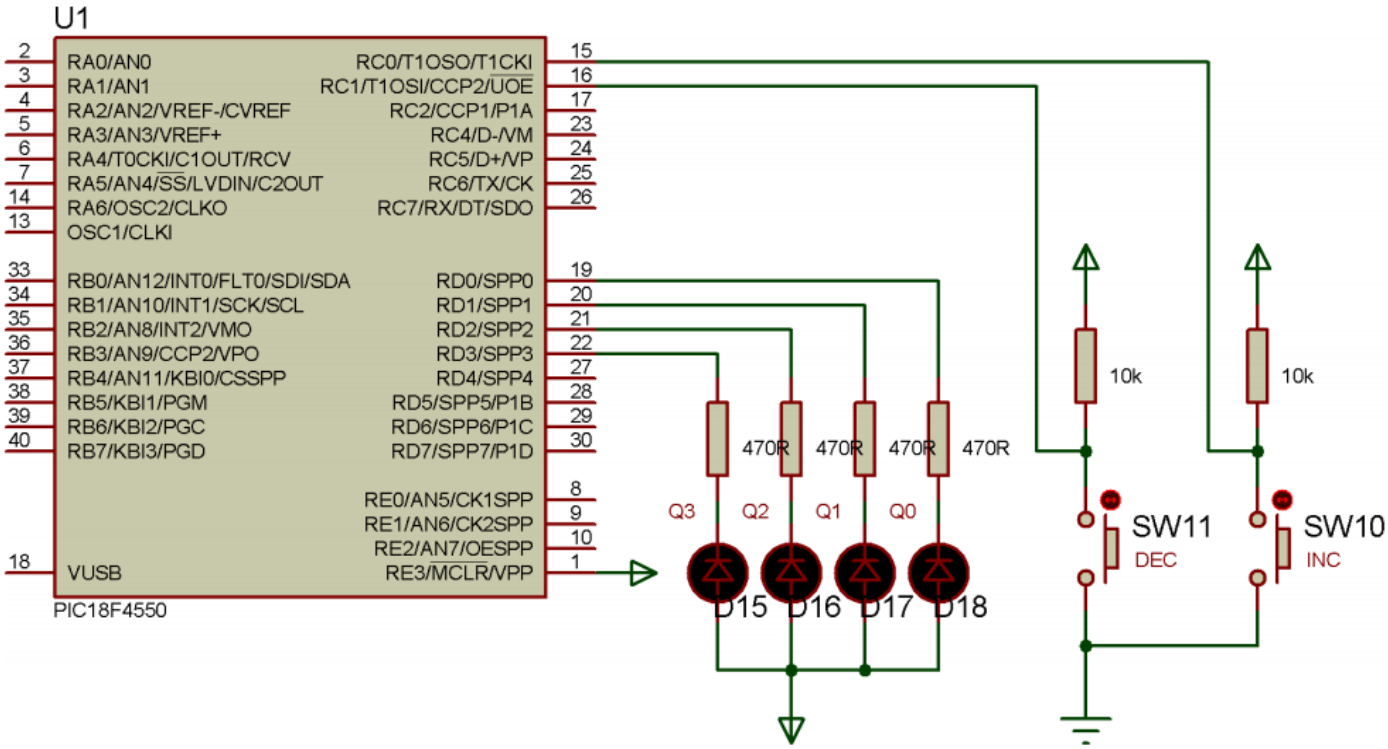


**Figura 6:** Circuito da atividade L2.

## 2- Desenvolvimento do *firmware*.

Crie um novo projeto o MPLAB X salvando-o no diretório de trabalho criado na atividade anterior.

Referências para a criação do projeto:

Nome do projeto: L2
Microcontrolador: PIC18F4550
Compilador: XC8
Ferramenta de programação: PicKIT 2

Implemente um sistema que acende os LEDs ligados nas portas RD0, RD1, RD2 e RD3 em função do acionamento dos botões SW10 e SW11 que estão ligados nas portas RC0 e RC1, respectivamente.

Inicialmente os LEDs devem estar apagados. O botão SW10 tem a função de acender os LEDs de forma progressiva e o botão SW11 tem a função de apagar os LEDs de forma regressiva. Ou seja, ao pressionar o botão SW10 pela primeira vez, o LED D18 (que está ligado na porta RD0) deve acender. Ao pressionar o botão SW10 novamente, o LED D17 (que está ligado na porta RD1) deve acender, e assim por diante. De forma semelhante, ao pressionar o botão SW11 pela primeira vez, o LED mais significativo que estiver acesso deve apagar.

Os botões realizam suas funções sempre que forem acionados e liberados. Utilizar a técnica de *debouncing* por *software* para os botões para garantir que seja possível acender os LEDs de forma controlada.

Observações sobre o kit de desenvolvimento XM118:

Possui um cristal de 20 MHz.
Os LEDs D15, D16, D17 e D18 são controlados pelos pinos de I/O RD3, RD2, RD1 e RD0, respectivamente. Nível lógico 1 apaga os LEDs e nível lógico 0 acende os mesmos.
Os botões SW10 e SW11 controlam os pinos de I/O RC0 e RC1 respectivamente. Se pressionados impõem nível lógico 0.

Utilize o seguinte código para iniciar seu projeto:

```
#include <xc.h>

// Configurações
#pragma config PLLDIV = 5          // PLL para 20MHz
#pragma config CPUDIV = OSC1_PLL2  // PLL desligado
#pragma config FOSC = HS           // Fosc = 20MHz; Tcy = 200ns
#pragma config WDT = OFF           // Watchdog timer desativado
#pragma config PBADEN = OFF        // Pinos do PORTB começam como digitais
#pragma config LVP = OFF           // Desabilita gravação em baixa tensão
#pragma config DEBUG = ON          // Habilita debug
#pragma config MCLRE = ON          // Habilita MCLR e desabilita RE3 como I/O

#define _XTAL_FREQ 20000000
```

### 3- Simulação do *firmware* gerado no Proteus.

Simule o código de máquina no Proteus e apresente seu funcionamento ao professor.

### 4- Gravação e execução do código de máquina no microcontrolador

Grave o código de máquina no microcontrolador e apresente o funcionamento para o professor.

### 5- Envio dos resultados para plataforma Moodle.

Compacte o diretório de trabalho em um arquivo .zip.

Renomeie o arquivo obedecendo o seguinte formato:
{Disciplina}_{Turma}_{Grupo}_{Número do roteiro}.zip
ex: EC45C_C51A_G1_L2.zip

Envie o arquivo compactado acessando no Moodle a atividade “L2”.