# Introdução

Seja bem vindo ao curso de programação de microcontroladores PIC!

Antes de iniciar, preciso lembrar que é muito importante que você tenha feito o curso de introdução a programação com linguagem C e que tenha algum conhecimento mínimo de eletrônica também. Durante o curso iremos utilizar softwares para a simulação do circuito e também circuitos reais na protoboard. É extremamente importante você reproduzir tudo o que falamos aqui para entender bem os aspectos de um sistema embarcado.

Neste curso utilizaremos o PIC18F45k22 como microcontrolador padrão, mas em um primeiro momento vamos trabalhar com um PIC16F84A que é um MCU mais simples e possível de ser simulado através do software gratuito Proteus. Primeiramente, vamos instalar os softwares necessários.

## Instalando Softwares

Como IDE, vamos utilizar o mplabX, que é um software que está a anos no mercado e apresenta muitas funcionalidades que ajudam na programação de um microcontrolador PIC.

Como compilador, utilizaremos o XC8, que é o compilador específico para microcontroladores PIC. O GCC não funcionará para o MCU pois ele não é feito para gerar o código assembly no formato que o PIC consiga ler. Já o XC8 foi feito para esta função.

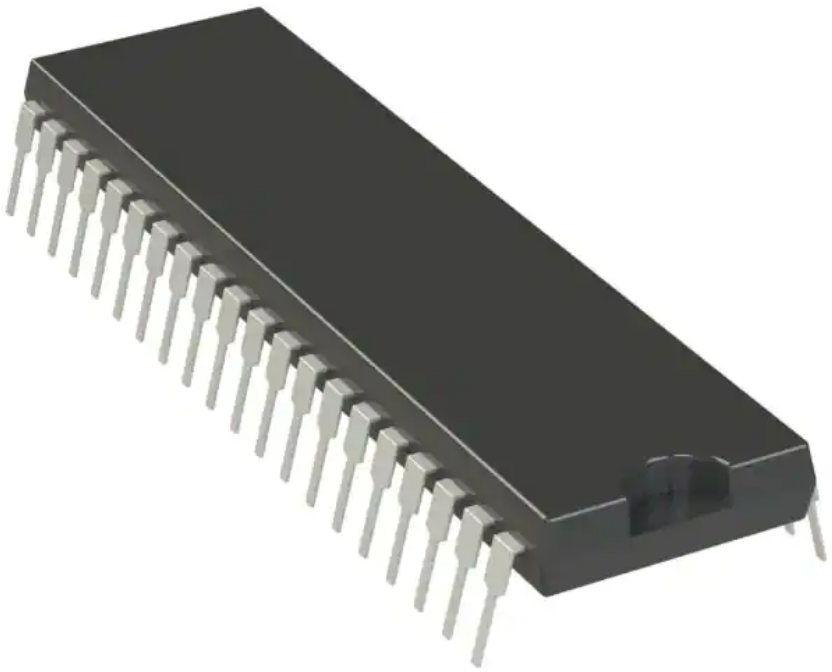
Para todos estes softwares existe sua versão gratuita, que muito provavelmente já atenderá qualquer projeto que você venha a fazer, e a versão paga que adicionará algumas funcionalidades de desempenho para sua aplicação.

Você também já deve ter ouvido falar do compilador MikroC, da mikroElektronica. É um software ainda mais completo, porém com grandes limitações em sua versão gratuita. Por isso, não o utilizaremos neste curso.

## Características do microcontrolador

Os microcontroladores (MCU’s) são formados por um conjunto de componentes integrados a um circuito. São eles:

- CPU
- Memória flash
- Memória RAM
- Memória EEPROM
- Conversores ADC
- Módulos PWM



No datasheet é possível encontrar também informações sobre tensão de operação, periféricos internos e outros. É muito importante saber utilizar o datasheet pois cabe ao desenvolvedor do sistema entender qual é o melhor microcontrolador para sua aplicação. Não existe um MCU ideal, mas existe o que vai apresentar o melhor custo benefício para você.

O PIC18F45k22 possui 36 pinos de propósito geral e 4 pinos dedicados a alimentação. Totalizando 40 pinos. A diferença entre e este microcontrolador e o PIC16F84A é enorme, possuindo conversores digitais/analógicos, comunicação UART, módulo CCP, entre outros. Enquanto o PIC16F se resume apenas ao acionamento de pinos GPIO's (input/output de propósito geral) e Timer.

É possível definir diretamente a funcionalidade de cada um dos pinos do MCU através de registradores que estudaremos a fundo quando formos programar. Mas certos pinos são dedicados a funcionalidades importantes como gravação do microcontrolador, comunicação serial, PWM, botão de reset e clock.

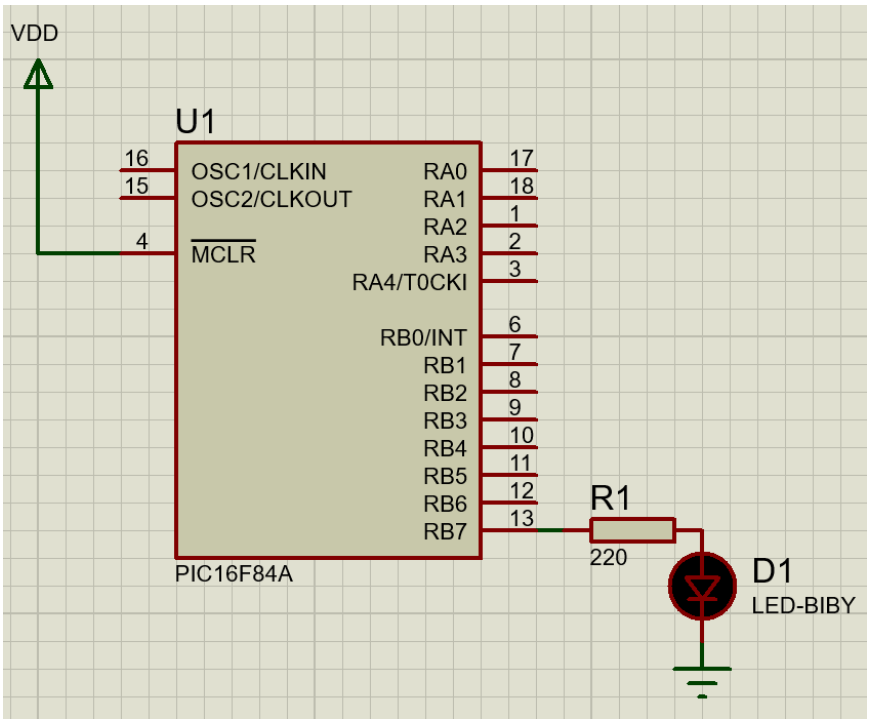
## Iniciando no proteus

Podemos simular um circuito através do proteus. Para isto basta fazer o download da versão gratuita no link https://www.labcenter.com/downloads/ que já será o suficiente. Na instalação também será necessário a instalação do compilador Visual C++, mas é um processo automático.

A versão gratuita do proteus terá limitações, dentre elas, você só pode fazer simulações com o PIC16F84A. Não é um problema pois o funcionamento é o mesmo. A diferença é a quantidade reduzida de pinos e algumas funcionalidades que não serão importantes agora.

A alimentação é realizada automaticamente. O pino de reset deve ficar em alto para o MCU funcionar.

Vamos colocar leds em todo o PORTB para entender como funciona os registradores




## Clock

O clock é extremanente importante para o funcionamento do microcontrolador. A arquitetura CMOS na qual o microcontrolador é feito necessita de pulsos de clock para executar as mudanças de estado nos MOSFET afim de realizar as instruções lógicas. Na verdade, qualquer tipo de computador hoje em dia é baseado em um clock.

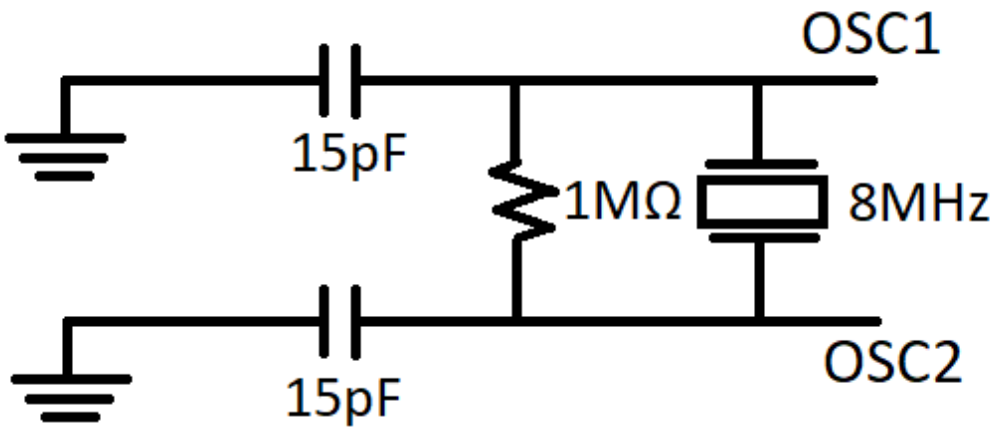
Clock pode ser traduzido como um relógio, ele gera pulsos em períodos fixos de tempo, como o ponteiro de um relógio. E as instruções lógicas são as mesmas instruções assembly que vimos no curso de lógica de programação.

```
START:  CLC         ; Limpa o registrador C
        LDA A_low   ; Carrega o valor de A
        ADC B_low   ; Soma com B
        STA B_low   ; Armazena o resultado em B

        LDA A_mid   ; Carrega o valor de A
        ADC B_mid   ; Soma com B
        STA B_mid   ; Armazena o resultado em B

        LDA A_high   ; Carrega o valor de A
        ADC B_high   ; Soma com B
        STA B_high   ; Armazena o resultado em B
```

Usaremos um cristal de clock de quartzo de 8MHz para nossos projetos. Ele estará conectado nos pinos RA6 e RA7 do microcontrolador.


O cristal acima é de 8Mhz, mas existem modelos de maior ou menor clock. Exemplo são os cristais de 32,768 kHz utilizados para circuitos de relógio alimentados por bateria. Um clock menor gera um menor aquecimento em circuitos integrados CMOS e, portanto, menor consumo de bateria. Praticamente qualquer placa mãe de computador possui um circuito assim.


A vantagem de utilizar cristais mais rápidos é que seu sistema executará mais rapidamente as instruções de leitura, escrita, etc. Há diversas aplicações onde isto é necessário, como a leitura de RPM de um motor de alta rotação. Porém, é muito incomum implementar cristais de clock muito mais altos que 20Mhz, por exemplo, pois isto gera uma interferência extremamente problemática no circuito, principalmente quando falamos de vias de comunicação. Na verdade, um cristal de 8Mhz já pode causar diversos problemas, por isso criamos um circuito capacitivo que serve como filtro para este tipo de interferência.




Não é necessário um cristal de quartzo para gerar clock, existem outros tipos de circuito capacitivos que fazem o trabalho, além de osciladores internos presentes nos

próprios microcontroladores. Mas para aplicações onde é necessário um clock preciso, o cristal de quartzo é o mais indicado.

Em circuitos robustos, como o de um computador ou smartphone, o clock da CPU pode se alto regular para atender a demanda de processamento. Por isso vemos tanta variação de temperatura nestes dispositivos, que se traduzem no aumento ou diminuição da velocidade de operação dos Coolers do sistema. E a forma como esses CPU's atingem clocks tão altos, na ordem de giga hertz, é com a utilização de PLL's.

PLL's são multiplicadores internos de clock que estão disponíveis também nos PIC. Com eles é possível multiplicar o clock sem alterar o circuito e sem utilizar cristais de clock muito elevado. Todos estes itens serão discutidos em um módulo específico mais a frente.

## Ciclo de máquina

Um cristal de 8Mhz executa 8 milhões de pulsos por segundo no microcontrolador. Mas isso não significa que o microcontrolador executará 8 milhões de instruções por segundo. Existe um outro conceito que é o **ciclo de máquina**.

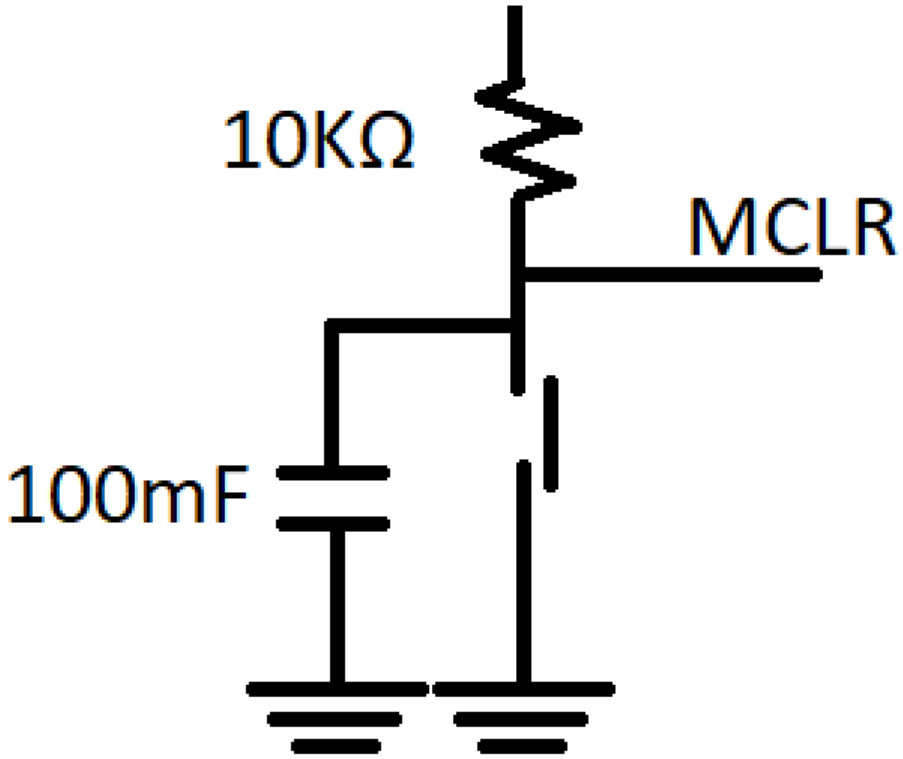
É importante ressaltar que isso varia de MCU para MCU. Mas para este caso, e de muitos outros PIC, 1 ciclo de máquina necessita de 4 pulsos de clock para acontecer. E é neste ciclo de máquina que o MCU executará uma instrução.

Então para esta aplicação, o MCU estará executando 2 milhões de instruções por segundo, ou 1 instrução a cada 0,5 microssegundo. Este é um valor importante para várias aplicações no futuro.

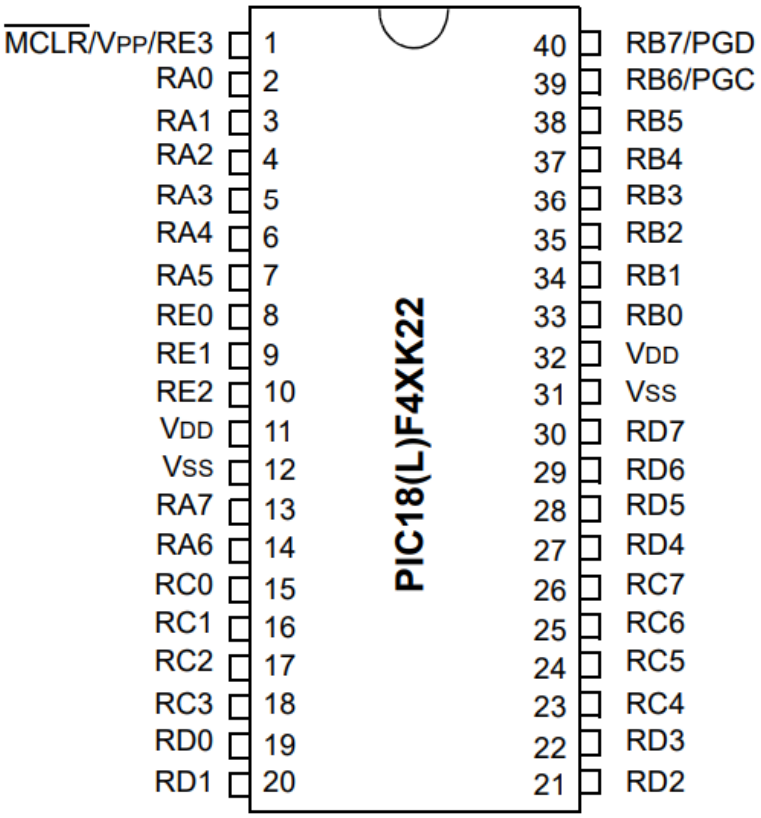
## Pino de RESET

Além do pino de alimentação, um pino que necessita de estar em nível lógico alto durante toda execução é o pino de MCLR (memory clear). Dizemos que ele realiza um reset pois ele limpa toda a memória RAM do microcontrolador, limpa a call stack de funções e retorna o ponteiro de execução para o inicio da memória, tudo isso quando vai para nível lógico zero. É basicamente um reset da aplicação. Nenhum dado da memória de programa ou da EEPROM é perdido.

Você pode ligá-lo diretamente à fonte com um resistor de 10kΩ ou criar um botão de reset para seu circuito.




**Ports do microcontrolador**




Ao olhar novamente o esquemático do MCU, vemos que ele possui conjuntos de pinos RA, RB, RC, RD e RE. Cada conjunto (exceto o RE) possuem 8 pinos. Isto está diretamente relacionado a arquitetura de 8 bits do MCU, significando que a comunicação do microcontrolador ocorre em 8 bits.

O conjunto de pinos formam 1 Porta do microcontrolador. Então temos as portas A, B, C e D, cada uma com um barramento de 8 bits para comunicação. Lembrando das aulas de programação, sabemos que 1 byte é um pacote de dado de 8 bits, portanto, podemos comunicar através destas portas de byte em byte. Para realizar essa comunicação, vamos conhecer os registradores que dão acesso aos pinos.

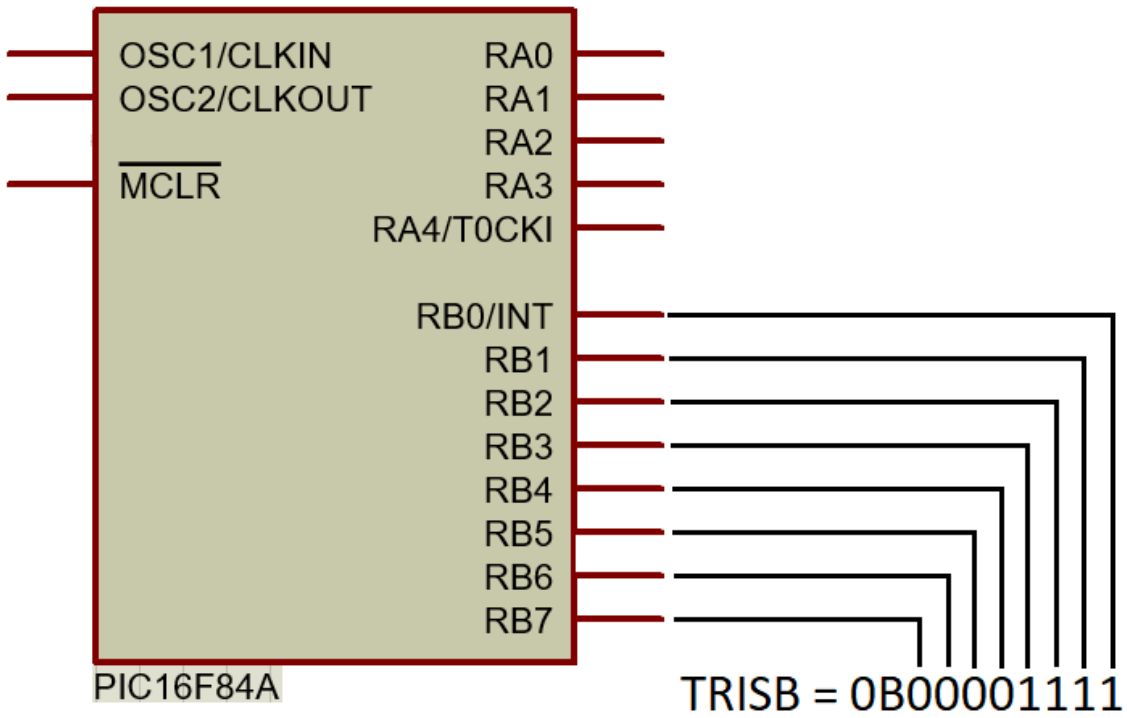
## Registradores TRIS E PORT

Estes serão os primeiros registradores que iremos trabalhar. São os registradores que dão acesso direto a leitura ou escrita de níveis lógicos nas Portas.

Começando pelo registrador TRIS, ele define se a porta será para escrita ou leitura. Existe um TRIS para cada porta, ou seja, TRISA, TRISB, TRISC, TRISD.

```
// Define todos os pinos da porta C como entrada
TRISC = 0b11111111;
TRISC = 0xFF;
TRISC = 255;

// Define todos os pinos da porta C como saída
TRISC = 0b00000000;
TRISC = 0x00;
TRISC = 0;

// Define somente o pino 1 da porta C como entrada e o resto como saída
TRISC = 0b00000001;
TRISC = 0x01;
TRISC = 1;
```

O byte escrito nestes registradores funciona como uma chave para seu respectivo pino




Ou também é possível acessar diretamente o pino

```
TRISBbits.TRISB7 = 0;  // Pino RB7 definido como output
PORTBbits.RB7 = 1;     // Nível lógico alto para o pino RB7
```

O registrador PORT funciona de forma parecida. Existe o PORTA, PORTB, PORTC, PORTD. Ele pode indicar o nível lógico do pino (para leitura) ou definir um nível lógico para o pino (para escrita). Os registradores TRIS e PORT andam juntos no código.

## Bits de configuração (Fusíveis)

Os bits de configuração são chaves físicas dentro do microcontrolador. Elas atuam alterando alguns parâmetros internos dentro do microcontrolador.

O PIC16F84A possui um conjunto de fusíveis bem reduzido se comparado ao PIC18F45K22, são eles:

```
#pragma config FOSC = HS        // Oscillator Selection bits (HS oscillator)
#pragma config WDTE = OFF       // Watchdog Timer (WDT disabled)
#pragma config PWRTE = ON       // Power-up Timer Enable bit (Power-up Timer is enabled)
#pragma config CP = OFF         // Code Protection bit (Code protection disabled)
```

As diretivas pragma são muito parecidas com as define. Elas dizem ao compilador que aquelas configurações devem ser tratadas antes do início da execução do programa em si.

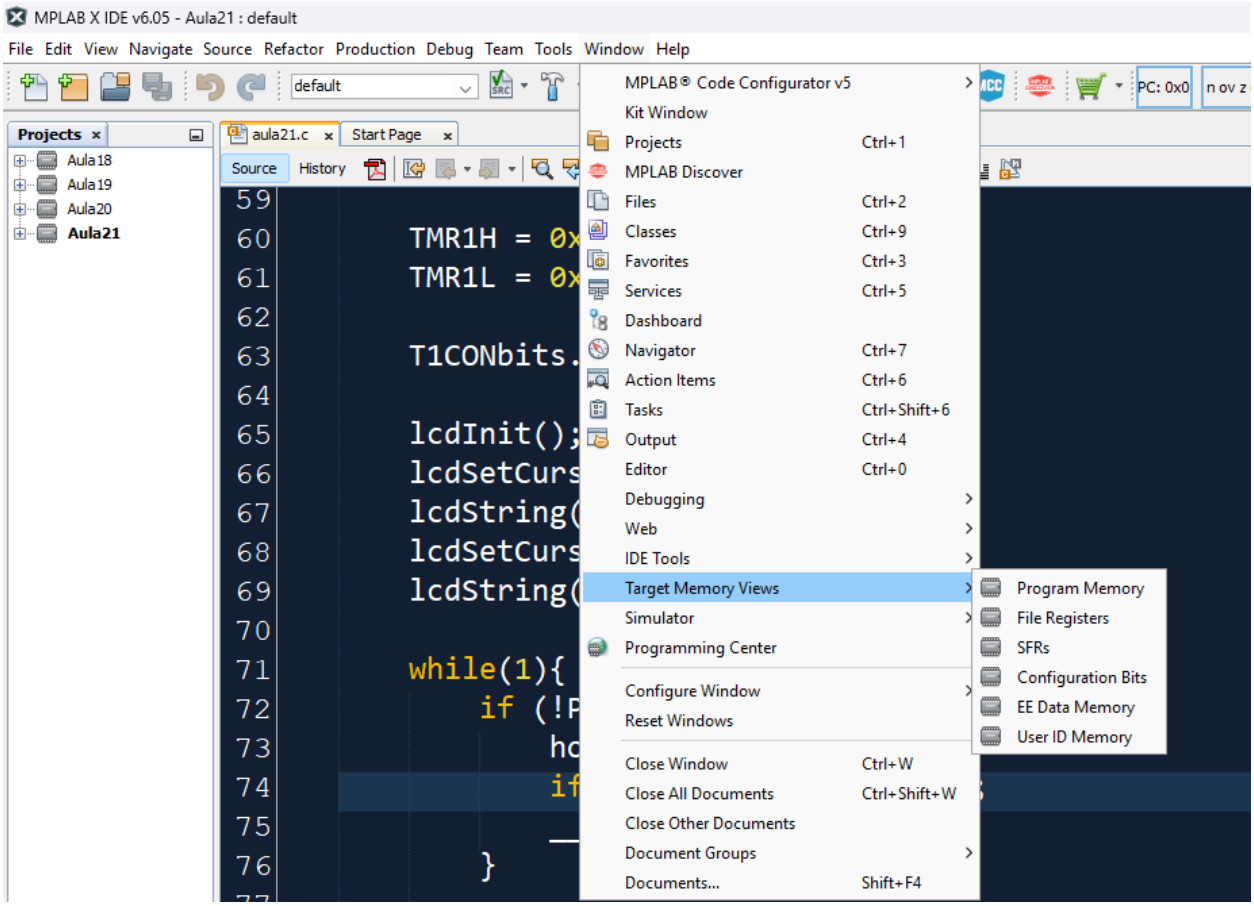
É aqui que encontraríamos também o PLL citado no assunto “Clock”. O PIC18 contém este bit que multiplica o clock da CPU 4x, mas o PIC16 que estamos utilizando não possui este recurso.

A diretivas pragma são específicas da linguagem C

Finalmente, depois de entender todos os detalhes, podemos finalmente acender o LED do conectado ao microcontrolador.

## Bits de configuração no PIC18F45k22 e mplabX

O MplabX possui uma ferramenta que auxilia na definição dos bits de configuração. Para acessá-la, vá no canto superior do software na aba **Window**, arraste o mouse no botão **Target Memory Views** e clique em **Configurations Bits**



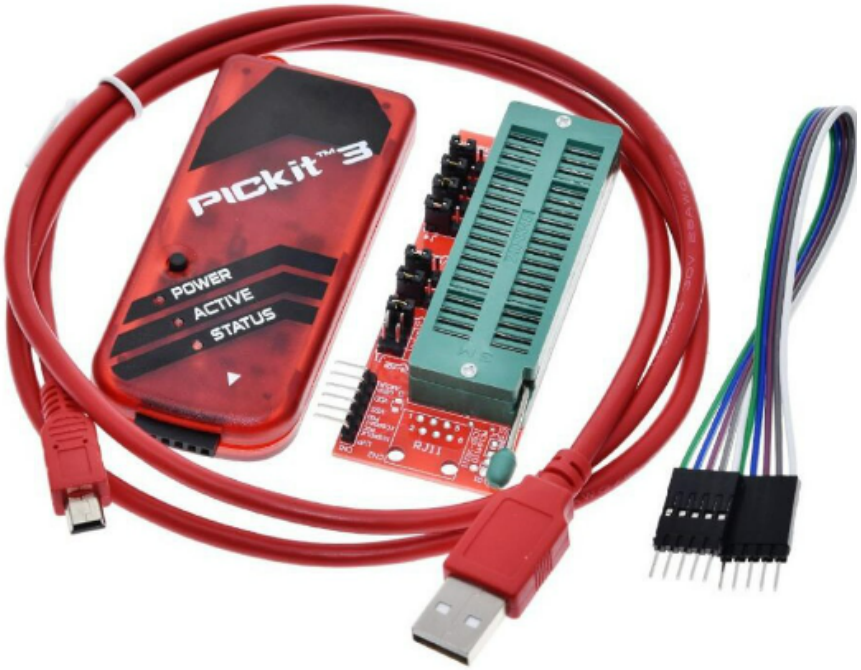

Abrirá uma tela com todos os bits de configuração específicos do microcontrolador que você selecionou no seu projeto. Geralmente, as configurações padrão são:

- Selecionar cristal de clock HS
- Habilitar power-up timer
- Desabilitar o WatchDog timer
- Desabilitar a entrada analógica do PORTB

Depois clique em **Generate source code**, copie o código e cole no topo de seu projeto.

## Gravação do microcontrolador

A escrita e compilação do código do será feita através de um computador com softwares da microchip que instalaremos mais a frente. A compilação, como sabemos, gerará um código binário que deverá ser carregado no microcontrolador. Para carregar utilizaremos o PicKit3


PICkit™3
POWER
ACTIVE
STATUS
RJ11