CEFETES
Centro Federal de Educação Tecnológica do Espírito Santo

Ministério da Educação

CENTRO FEDERAL DE EDUCAÇÃO TECNOLOGICA DO ESPIRITO SANTO

UNED – SERRA / AUTOMAÇÃO INDUSTRIAL

APOSTILA DE:
PROGRAMAÇÃO DE
MICROCONTROLADORES PIC USANDO LINGUAGEM C

PROF: MARCO ANTONIO

VITORIA, AGOSTO 2006

INTRODUÇÃO

Esta pequena apostilha esta orientada para os profissionais técnicos que necessitam de conhecimentos básicos do PIC. Abordaremos noções da estrutura do PIC16F877A e depois nosso foco será o estudo da linguagem C para microcontroladores.

A linguagem C provavelmente é a linguagem mais conhecida e tem muitas vantagens sobre a linguagem assembler no nível técnico.

Primeiramente trataremos sobre a instalação dos programas necessários para poder trabalhar. O compilar CCS será o escolhido e o entorno ou editor será o MatLab V 7.4.

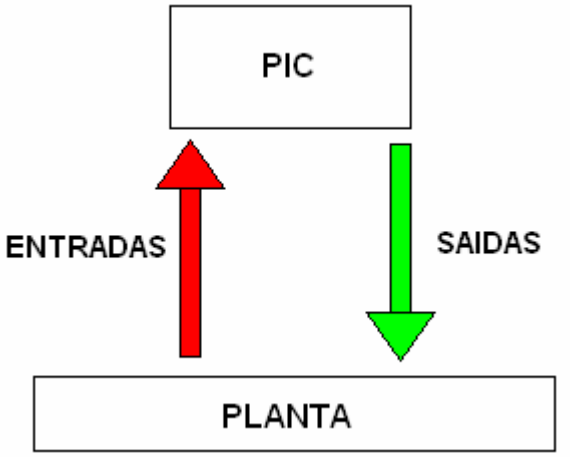
Uma vez com nosso ambiente instalado abordaremos as características básicas do PIC 16F877A. È importante mencionar que usaremos o módulo de desenvolvimento SD-1700 que atualmente esta descontinuado, mas é considerado pelo autor como uma excelente ferramenta.

O estudo da linguagem C será focado à compreensão das funções básicas e sua relação com o circuito de teste. Com esses espera-se que o leitor consiga entender a infinita utilizada destes magníficos dispositivos.

1. INTRODUÇÃO AOS MICROCONTROLADORES

1.1. O que é um Microcontrolador?

Podemos definir o microcontrolador como um "pequeno" componente eletrônico, dotado de uma "inteligência" programável, utilizado no controle de processos lógicos. O controle de processos deve ser entendido como o controle de periféricos, tais como: led´s, botões, display's de cristal líquido (LCD), resistências, relês, sensores diversos (pressão, temperatura, etc.) e muitos outros. São chamados de controles lógicos pois a operação do sistema baseia-se nas ações lógicas que devem ser executadas, dependendo do estado dos periféricos de entrada e/ou saída.



1.2. Qual a diferença do Microcontroladores e dos Microprocessadores?

Um microcontrolador difere de um microprocessador em vários aspectos. Primeiro e o mais importante, é a sua funcionalidade. Para que um microprocessador possa ser usado, outros componentes devem-lhe ser adicionados, tais como memória e componentes para receber e enviar dados. Em resumo, isso significa que o microprocessador é o verdadeiro coração do computador. Por outro lado, o microcontrolador foi projectado para ter tudo num só.
Nenhuns outros componentes externos são necessários nas aplicações, uma vez que todos os periféricos necessários já estão contidos nele. Assim, nós poupamos tempo e espaço na construção dos dispositivos.

1.3. O que significa PIC?

É o nome que a Microchip adotou para a sua família de microcontroladores, sendo que a sigla significa **Controlador Integrado de Periféricos**.

O **PIC** é um circuito integrado produzido pela Microchip Technology Inc., que pertence da categoria dos microcontroladores, ou seja, um componente integrado

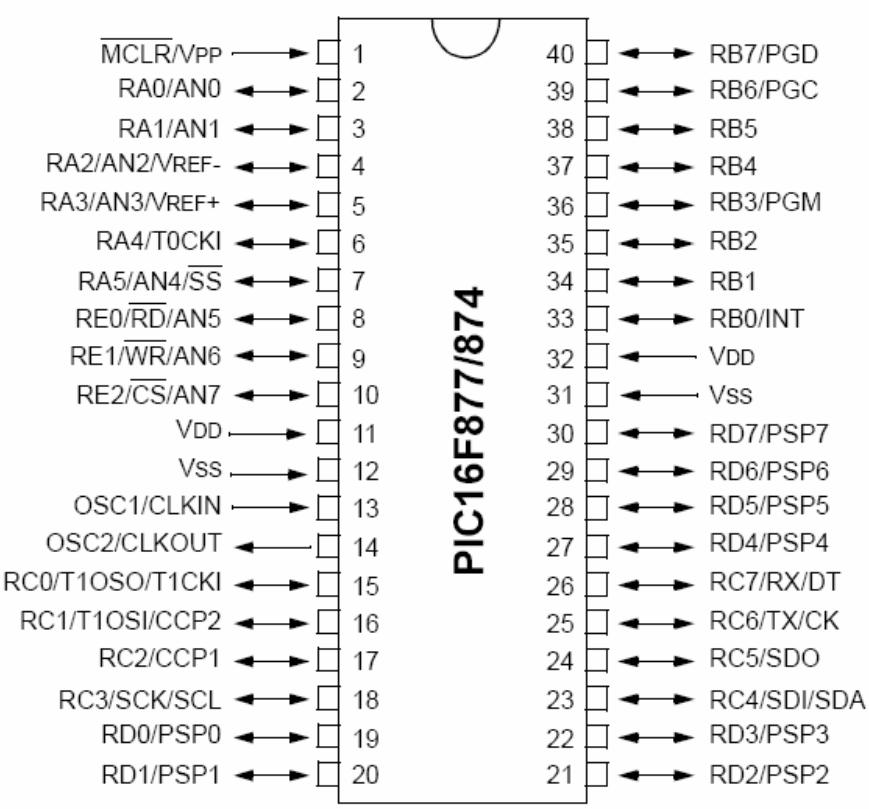
que em um único dispositivo contem todos os circuitos necessários para realizar um completo sistema digital programável.

Internamente dispõe de todos os dispositivos típicos de um sistema microprocessado, ou seja:
Uma **CPU** (**C**entral **P**rocessor **U**nit ou seja Unidade de Processamento Central) e sua finalidade é interpretar as instruções de programa.

Uma memória **PROM** (**P**rogrammable **R**ead **O**nly **M**emory ou Memória Programavel Somente para Leitura) na qual ira memorizar de maneira permanente as instruções do programa.

Uma memória **RAM** (**R**andom **A**ccess **M**emory ou Memoria de Accesso Aleatório) utilizada para memorizar as variáveis utilizadas pelo programa.

Uma serie de **LINHAS de I/O** para controlar dispositivos externos ou receber pulsos de sensores, chaves, etc.
Uma serie de dispositivos auxiliares ao funcionamento, ou seja gerador de clock, bus, contador, etc.



A presença de todos estes dispositivos em um espaço extremamente pequeno, da ao projetista ampla gama de trabalho e enorme vantagem em usar um sistema microprocessado, onde em pouco tempo e com poucos componentes externos podemos fazer o que seria oneroso fazer com circuitos tradicionais.
O **PIC** esta disponível em uma ampla gama de modelos para melhor adaptar-se as exigencias de projetos especificos, diferenciando-se pelo numero de linha de I/O e pelo conteudo do dispositivo. Inicia-se com modelo pequeno identificado pela sigla **PIC12Cxx** dotado de 8 pinos, até chegar a modelos maiores com sigla **PIC17Cxx** dotados de 40 pinos.
Uma descrição detalhada da tipologia do PIC é disponivel no site da Microchip acessavel via , onde conseguimos encontrar grandes e variadas quantidades de informações tecnicas, software de apoio, exemplos de aplicações e atualizações disponiveis.
Para o nosso curso usaremos um modelo de PIC o **PIC16F877**. Este é dotado de 40 pinos.

1.4. Programação do PIC

Como o PIC é um dispositivo programável, o programa tem como objetivo deixar instruções para que o PIC possa fazer atividades definidas pelo programador.
Um programa é constituído por um conjunto de instruções em seqüência, onde cada uma identificara precisamente a função básica que o PIC ira executar.
Um programa escrito em linguagem assembler ou em C pode ser escrito em qualquer PC utilizando-se qualquer processador de texto que possa gerar arquivos ASCII(Word, Notpad etc). Um arquivo de texto que contenha um programa em assembler é denominado de **source ou codigo assembler**.
Uma vez preparado o nosso código assembler ou C(veremos mais adiante), iremos precisar de um programa para traduzir as instruções mnemônicas e todas as outras formas convencionais com que escrevemos o nosso código em uma serie de números (o opcode) reconhecível diretamente pelo PIC. Este programa se chama **compilador assembler** ou **assemblador**, o compilador usado neste curso será o PCWH.
Na figura seguinte está esquematizado o fluxograma de operações e arquivos que devera ser realizado para passar um código assembler a um PIC a ser programado.

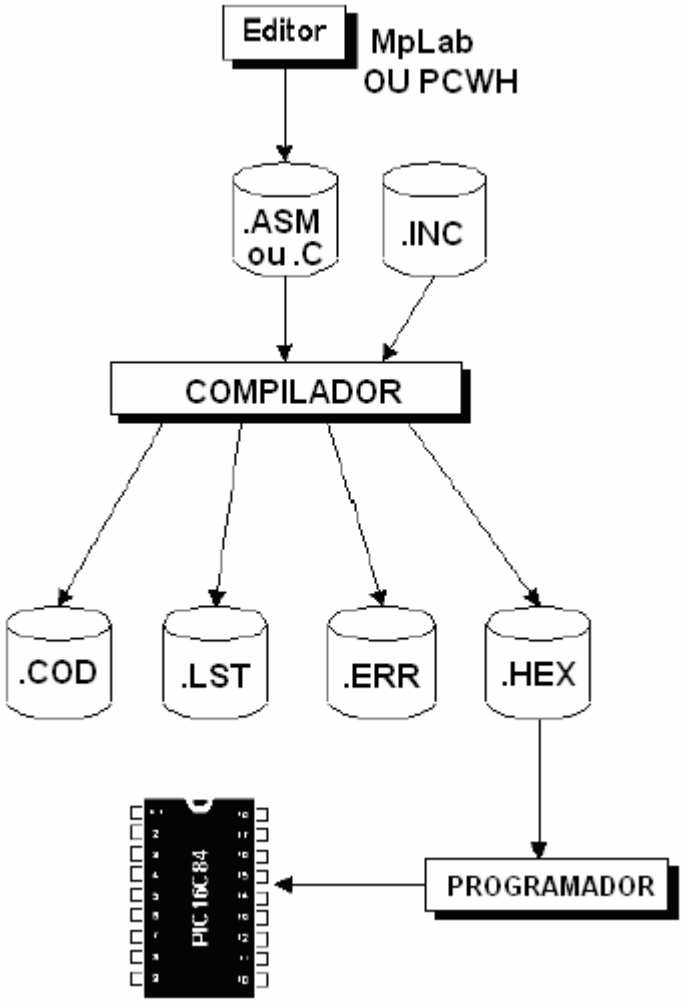


Ilustração 1: Fluxograma de compilação de um programa e gravação de um PIC

Para poder programar um PIC precisaremos de um Editor (Mplab ou o PCWH), um compilador (PCWH) e um programador (PICStar Plus).

1.5. O que é o MpLab?

O MpLab é um ambiente integrado de desenvolvimento (I.D.E.: Integrated Development Environment). No mesmo ambiente o usuário pode executar todos os procedimentos relativos ao desenvolvimento de um software para o PIC (edição, compilação, simulação, gravação), tornando o trabalho do projetista mais produtivo. Este programa é totalmente gratuito e pode ser pego no site www.microchip.com.

Edição
O MpLab possui um editor de texto para seus programas que possui diversas ferramentas de auxílio como localizar, substituir, recortar, copiar e colar.

Compilação
Compilar significa traduzir um programa escrito em assembly (mneumônicos) para linguagem de máquina (números). A compilação gera um arquivo com extensão .hex (hexadecimal) a partir dos arquivos de código fonte (.asm) e de projeto (.pjt). É o conteúdo do arquivo hexadecimal que é gravado na memória de programa do PIC.

Simulação
O MpLab possui ferramentas para simulação do programa no próprio computador, possibilitando a execução passo a passo, visualização e edição do conteúdo dos registradores, edição de estímulos (entradas), contagem de tempo de execução, etc.

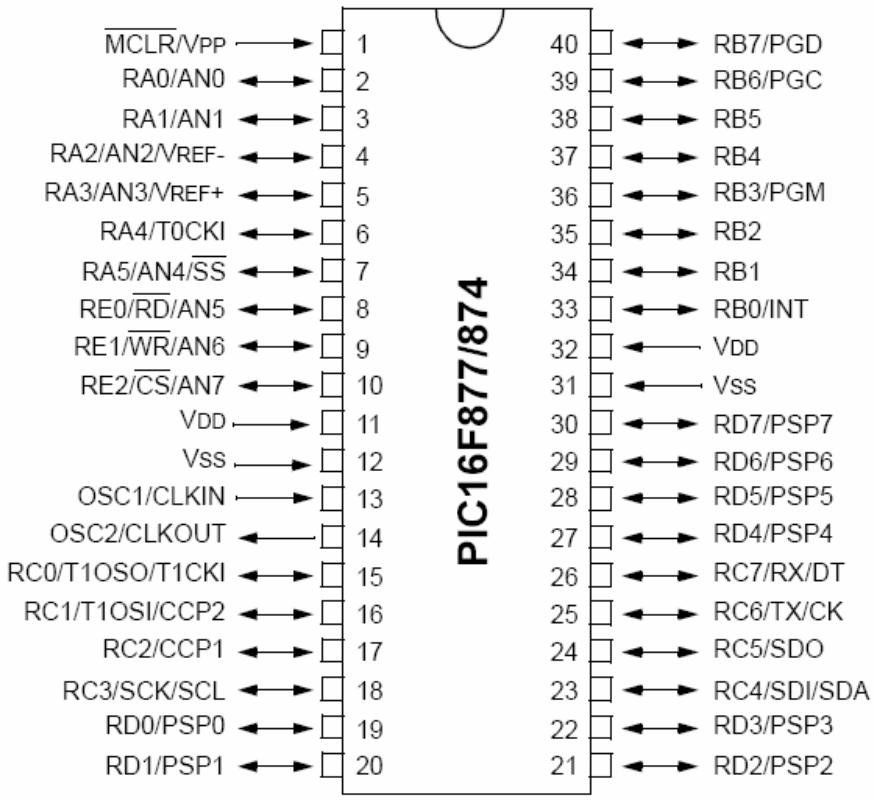
Gravação
Para que o programa seja executado no microcontrolador, o arquivo hexadecimal deve ser gravado no PIC. O MpLab oferece suporte aos gravadores fabricados pela Microchip.

Emulação
A Emulação é um recurso de desenvolvimento que possibilita testes em tempo real. O MpLab oferece suporte ao hardware necessário para estar emulando um determinado programa. Esta emulação é feita conectando-se (através do hardware mencionado) o computador ao sistema projetado, no lugar do PIC

2. O PIC 16F877A

Dentro de todos os PICs este é um dos que apresenta mais recursos e se adpata as necessidades de nosso curso, entre suas caraterísticas principais podemos citar:

- Microcontrolador de 40 pinos, o que possibilita a montagem de um hardware complexo e capaz de interagir com diversos recursos e funções ao mesmo tempo.
- 33 portas configuráveis como entrada ou saída.
- 15 interrupções disponíveis.
- Memória de programação E²PROM FLASH, que permite a gravação rápida do programa diversas vezes no mesmo chip, sem a necessidade de apagá-lo por meio de luz ultravioleta, como acontece nos microcontroladores de janela;



- Memória de programa com 8Kwoeds, com capacidade de escrita e leitura pelo própio código interno;
- Memória E²PROM (não volátil) interna com 256 bytes;
- Memória RAM com 368 bytes;
- Têes timers (2x8bits e 1x16 bits);
- Comunicações seriais: SPI, I²C e USART;Conversores analógicos de 10 bits (8x) e comparadores analógicos (2x);
- Conversores analógicos de 10 bits (8x) e comparadores analógicos (2x);
- Dois módulos CCP: Capture, Compare e PWM;

2.1. NOMENCLATURA DOS PINOS

Nome do Pino	Núm. Pino	I/O/P	TIPO	Descrição
OSC1/CLKIN	13	I	ST/CMOS	Entrada para cristal. Osciladores externos(RC)
OSC2/CLKOUT	14	O	-	Saída para cristal. Os cristais ou resonadores devem ser ligados aos pinos OSC1 e OSC2. Saída com onda quadrada em ¼ da frequencia imposta em OSC1 quando em modo RC. Essa frequencia equivale aos ciclos de máquina internos.
MCLR/Vpp	1	I/P	ST	Master Clear (reset) externo. O uC só funciona quando este pino encontra-se em nível alto. Entrada para tensão de programação 13V.
Vss	12/31	P	-	GND
Vdd	11/32	P	-	Alimentação positiva
				PORTA (I/O digitais bidirecionais e sistema analógico)
RA0/AN0	2	I/0	TTL	RA0: I/O digital ou entrada analógica AN0
RA1/AN1	3	I/0	TTL	RA1: I/O digital ou entrada analógica AN1
RA2/AN2/V_{REF-} /CV_{REF}	4 -	I/0 -	TTL -	RA2: I/O digital ou entrada analógica AN2 ou tensão negativa de referência analógica.
RA3/AN3 /V_{REF+}	5 -	I/0 -	TTL -	RA3: I/O digital ou entrada analógica AN3 ou tensão positiva de referência analógica.
RA4/TOCKI /C10OUT	6 -	I/0 -	ST -	RA4: I/O digital (*open drayn*), ou entrada externa do contador TMR0 ou saída do comparador 1.
RA5/SS/AN4 /C2OUT	7	I/0	TTL	RA5: I/O digital ou entrada analógica AN4 ou habilitação externa (slave select) para comunicação SPI ou saída do comparador 2.
				PORTB: (I/Os digitais bidirecionais). Todos os pinos deste PORT possuem *pull-up* que podem ser ligado ou desligados pelo software:
RB0/INT	33	I/O	TTL/ST(1)	RB0: I/O digital com interrupção externa.
RB1	34	I/O	TTL	RB1: I/O digital.
RB2	35	I/O	TTL	RB2: I/O digital.
RB3/PGM	36		TTL	RB3: I/O digital ou entrada para programação em baixa tensão (5v).
RB4	37		TTL	RB4: I/O digital com interrupção por mudança de estado
RB5	38		TTL	RB5: I/O digital com interrupção por mudança de estado
RB6/PGC	39		TTL/ST(2)	RB6: I/O digital com interrupção por mudança de estado ou clock da programação serial ou pino de *in-circuit debugger.*
RB7/PGD	40	I/O	TTL/ST(2)	RB7: I/O digital com interrupção por mudança de estado ou data da programação serial ou pino de *in-circuit debugger.*

				PORTC (I/Os digitais bidirecionais):
RC0/T1OSO /TICK1	15	I/O	ST	RC0: I/O digital ou saída do oscilador externo para TMR1 ou entrada de incremento para TMR1.
RC1/T1OSI /CCPS	16	I/O	ST	RC1: I/O digital ou saída do oscilador externo para TMR1 ou entrada de Capture2 ou saídas para Compare2/PWM2.
RC2/CCP1	17	I/O	ST	RC2: I/O digital ou entrada do Capture1 ou saídas para Compare1/PWM1.
RC3/SCK/SCL	18	I/O	ST	RC3: I/O digital ou entrada/saída de clock para comunicação serial SPI/I²C.
RC4/SDI/DAS	23	I/O	ST	RC4: I/O digital ou entrada de dados para SPI ou via de dados (entrada/saída) para I²C.
RC5/SDO	24	I/O	ST	RC5: I/O digital ou saída de dados para SPI.
RC6/TX/CK	25	I/O	ST	RC6: I/O digital ou TX (transmissão) para comunicação USART assíncrona ou data para comunicação síncrona.
RC7/RX/DT	26	I/O	ST	RC7: I/O digital ou RX (recepção) para comunicação USART assíncrona ou data para comunicação síncrona.
				PORTD (I/Os digitais bidirecionais) ou porta de comunicação paralela:
RD0/PSP0	19	I/O	TTL/ST(3)	RD0: I/O digital ou dado 0 (comunicação paralela).
RD1/PSP1	20	I/O	TTL/ST(3)	RD1: I/O digital ou dado 1 (comunicação paralela).
RD2/PSP2	21	I/O	TTL/ST(3)	RD2: I/O digital ou dado 2 (comunicação paralela).
RD3/PSP3	22	I/O	TTL/ST(3)	RD3: I/O digital ou dado 3 (comunicação paralela).
RD4/PSP4	27	I/O	TTL/ST(3)	RD4: I/O digital ou dado 4 (comunicação paralela).
RD5/PSP5	28	I/O	TTL/ST(3)	RD5: I/O digital ou dado 5 (comunicação paralela).
RD6/PSP6	29	I/O	TTL/ST(3)	RD6: I/O digital ou dado 6 (comunicação paralela).
RD7/PSP7	30	I/O	TTL/ST(3)	RD7: I/O digital ou dado 7 (comunicação paralela).
				PORTE (I/Os digitais bidirecionais e sistema analógico):
RE0/RD/AN5	8	I/O	TTL/ST(3)	RE0: I/O digital ou controle de leitura da porta paralela ou entrada analógica AN5.
RE0/RD/AN6	9	I/O	TTL/ST(3)	RE1: I/O digital ou controle de leitura da porta paralela ou entrada analógica AN6.
RE0/RD/AN7	10	I/O	TTL/ST(3)	RE2: I/O digital ou controle de leitura da porta paralela ou entrada analógica AN7.

Legenda:

I =*Input* (entrada)
I/O = *Input/OutPut* (entrada ou saída)
- = não utilizado
ST = Entrada tipo *Schmitt trigger*
O = *Output*(saída)
P = *Power*(Alimentação)
TTL = Entrada tipo TTL

Notas:

(1): Esta entrada é do tipo ST, somente quando configurado como interrupção externa.

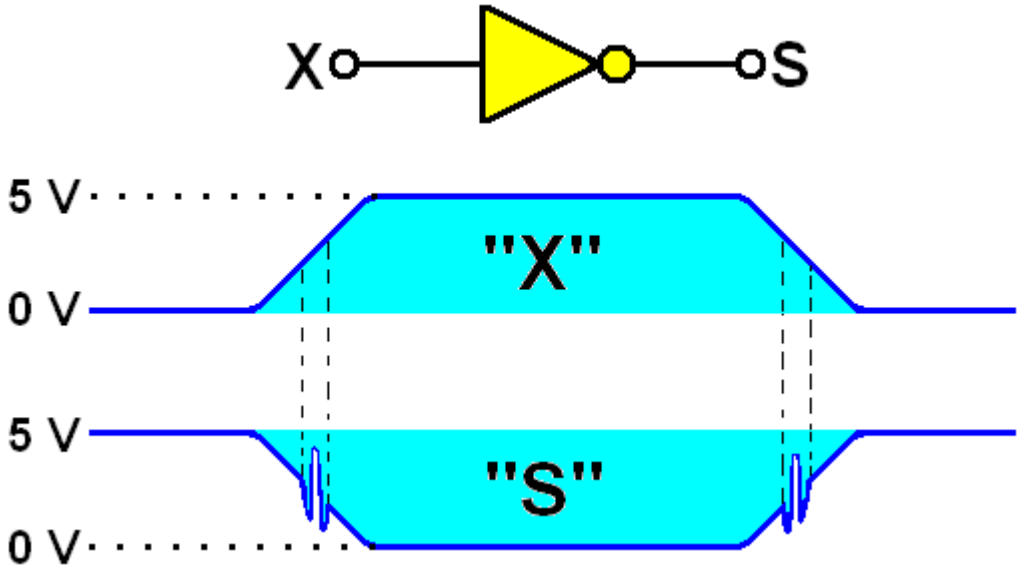
(2): Esta entrada é do tipo ST, somente durante o modo de programação serial.

(3): Esta entrada é do tipo ST, quando configurado como I/O de uso geral e TTL quando usado em modo de porta paralela.

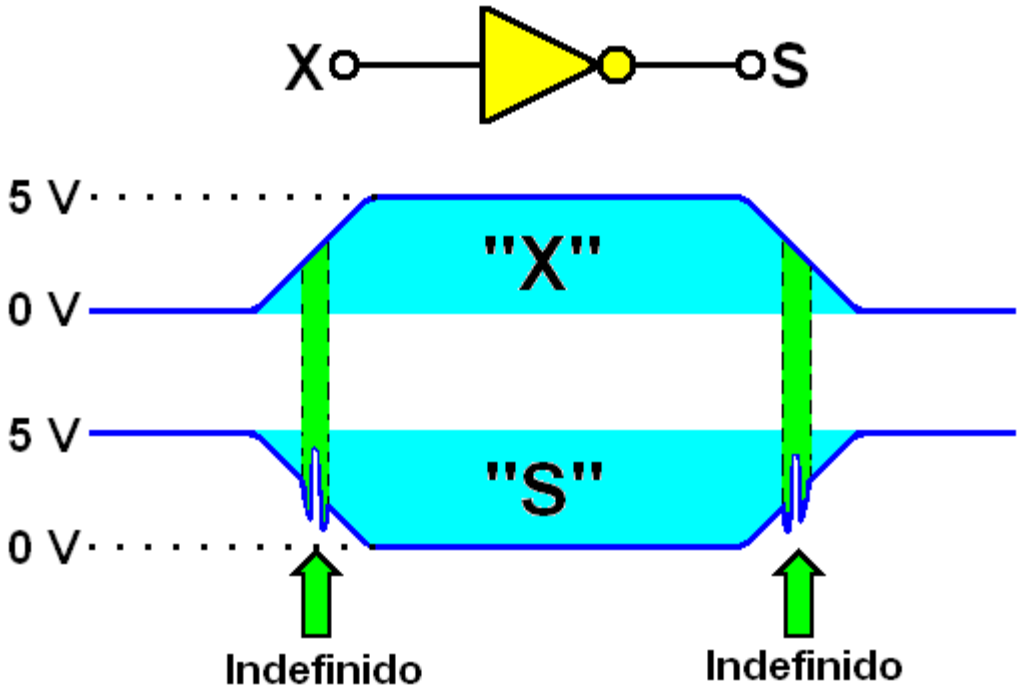
(4): Esta entrada é ST quando em modo RC e CMOS nos demais casos.

2.2. QUE É SCHMITT-TRIGGER?

Observe o gráfico de transição da porta lógica "NÃO" abaixo.



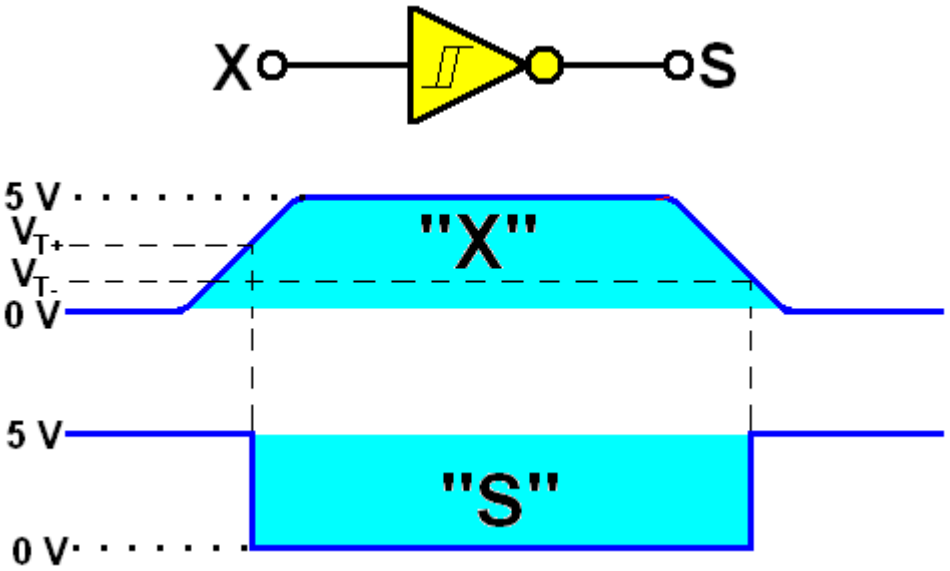
Verifica-se pelo gráfico que a transição de nível lógico baixo (*LOW*) para alto (*HIGH*) é muito longa, implicando na oscilação da saída da porta lógica "NÃO" à medida que o sinal de entrada passa pelo intervalo indefinido (1 ou 0), transportando para saída o mesmo efeito.



Um dispositivo interessante para a solução deste problema lógico, são as portas ***Schmitt-trigger.***

Portanto fica claro que não existe apenas porta lógica "NÃO" ***Schmitt-trigger***, existem outras portas com essa tecnologia.

Porta lógica ***Schmitt-trigger*** é projetada para aceitar sinais cuja a transição é lenta e garante na saída nível lógico livre de oscilações.



Analisando o gráfico da figura anterior observa-se que a saída muda de nível lógico alto (*HIGH*) para baixo (*LOW*) quando a tensão de transição é superior a $\mathbf{V_{T+}}$ (***Positive-going threshold*** - limiar de tenso positiva de transição).

2.3. GERADOR DE RELÓGIO – OSCILADOR

O circuito do oscilador é usado para fornecer um relógio (clock), ao microcontrolador. O clock é necessário para que o microcontrolador possa executar um programa ou as instruções de um programa.

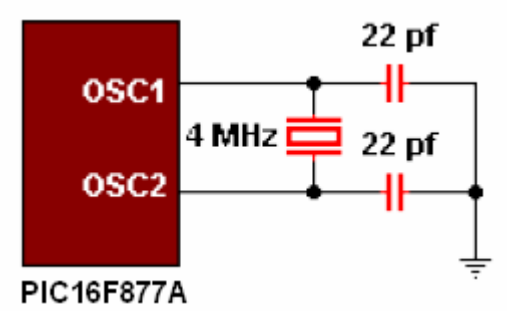
Tipos de osciladores

O PIC pode trabalhar com quatro configurações de oscilador. Uma vez que as configurações com um oscilador de cristal e resistência-condensador (RC) são aquelas mais frequentemente usadas, elas são as únicas que vamos mencionar aqui.

Quando o oscilador é de cristal, a designação da configuração é de XT, se o oscilador for uma resistência em série com um condensador, tem a designação RC. Isto é importante, porque há necessidade de optar entre os diversos tipos de oscilador, quando se escolhe um microcontrolador.

2.4. Oscilador XT

O oscilador de cristal está contido num envólucro de metal com dois pinos onde foi escrita a freqüência a que o cristal oscila. Dois condensadores cerâmicos devem ligar cada um dos pinos do cristal à massa. Casos há em que cristal e condensadores estão contidos no mesmo encapsulamento, é também o caso do ressonador cerâmico ao lado representado. Este elemento tem três pinos com o pino central ligado à massa e os outros dois pinos ligados aos pinos OSC1 e OSC2 do microcontrolador. Quando projetamos um dispositivo, a regra é colocar o oscilador tão perto quanto possível do microcontrolador, de modo a evitar qualquer interferência nas linhas que ligam o oscilador ao microcontrolador.



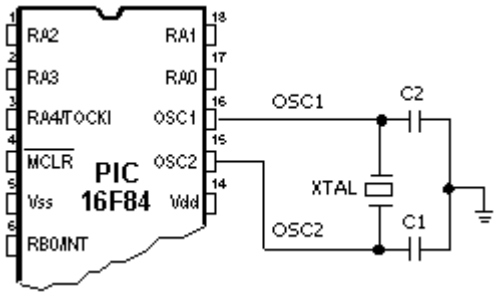


Ilustração 2: Clock de um microcontrolador a partir de um cristal de quartzo

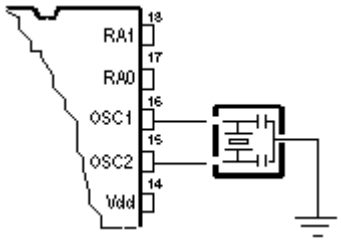


Ilustração 3: Clock de um microcontrolador com um ressonador

2.5. Oscilador RC

Em aplicações em que a precisão da temporização não é um fator crítico, o oscilador RC torna-se mais econômico. A freqüência de ressonância do oscilador RC depende da tensão de alimentação, da resistência R, capacidade C e da temperatura de funcionamento.

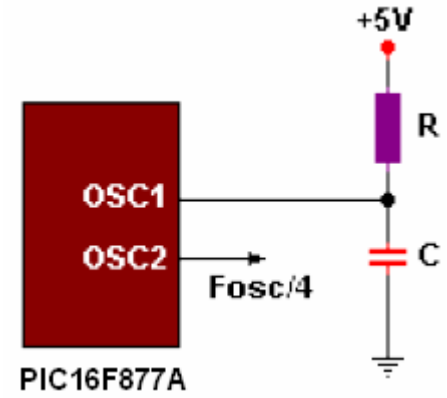


Ilustração 4: Ligação de umoscilador RC

O diagrama acima, mostra como um oscilador RC deve ser ligado a um PIC16F877A. Com um valor para a resistência R abaixo de 2,2 K, o oscilador pode tornar-se instável ou pode mesmo parar de oscilar. Para um valor muito grande R (1M, por exemplo), o oscilador torna-se muito sensível à umidade e ao ruído. É recomendado que o valor da resistência R esteja compreendido entre 3K e 100K. Apesar de o oscilador poder trabalhar sem condensador externo (C = 0 pF), é conveniente, ainda assim, usar um condensador acima de 20 pF para evitar o ruído e aumentar a estabilidade. Qualquer que seja o oscilador que se está a utilizar, a freqüência de trabalho do microcontrolador é a do oscilador dividida por quatro. A freqüência de oscilação dividida por 4 também é fornecida no pino OSC2/CLKOUT e, pode ser usada, para testar ou sincronizar outros circuitos lógicos pertencentes ao sistema.

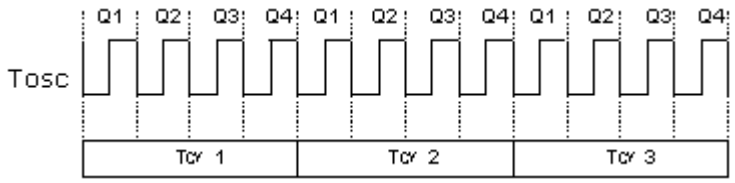


Ilustração 5:Relação entre o sinal de clock e os ciclos de instrução

Ao ligar a alimentação do circuito, o oscilador começa a oscilar. Primeiro com um período de oscilação e uma amplitude instável, mas, depois de algum tempo, tudo estabiliza.

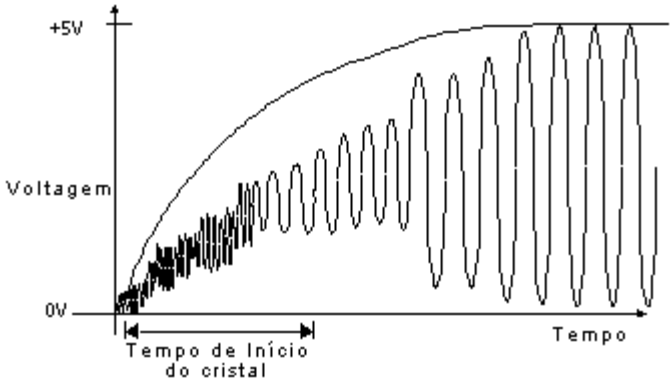


Ilustração 6:Sinal de clock do oscilador do microcontrolador depois de ser ligada a alimentação

Para evitar que esta instabilidade inicial do clock afete o funcionamento do microcontrolador, nós necessitamos de manter o microcontrolador no estado de reset enquanto o clock do oscilador não estabiliza. O diagrama em cima, mostra uma forma típica do sinal fornecido por um oscilador de cristal de quartzo ao microcontrolador quando se liga a alimentação.

Para ter um controle sobre o reset é conveniente instalar um botão como se mostra na figura.

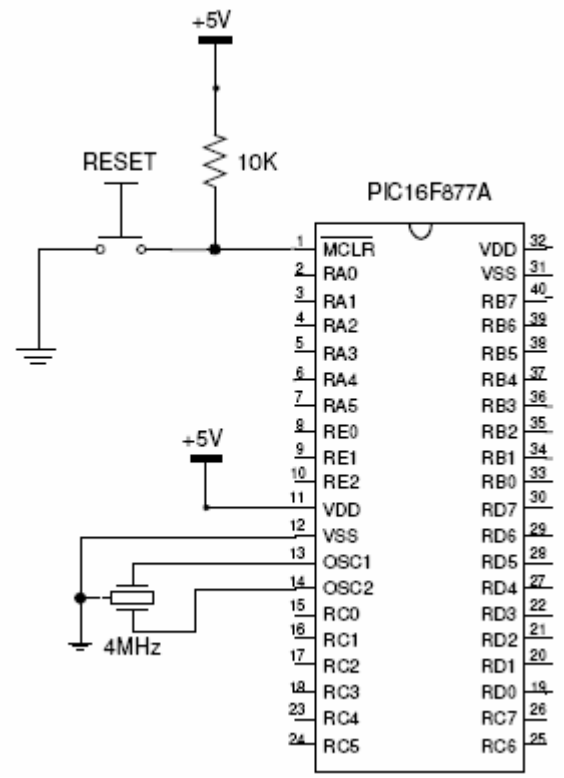


Ilustração 7: Circuito de Reset através de um botão

3. AMBIENTE INTEGRADO DE DESENVOLVIMENTO (IDE)

Como ambiente de desenvolvimentos será usado o compilador PCWH da CCS. Este é constituído de um IDE gráfico que pode ser executado em qualquer plataforma Windows. Este ambiente permitirá uma programação dos PICs em linguagem C. Existe a possibilidade de se fazer uma integração com o ambiente MPLAB da própria microchip. Como vantagem do uso desse compilador pode-se citar a grande eficiência do código gerado, compatibilidade com o padrão ANSI e ISSO salvo algumas exceções e a sua grande diversidade de funções e bibliotecas desenvolvidas em linguagem C.

O objetivo deste capítulo é a compreensão do ambiente de programação e da seqüência para poder criar um novo projeto usando o MpLab como editor e o CCS como compilador.

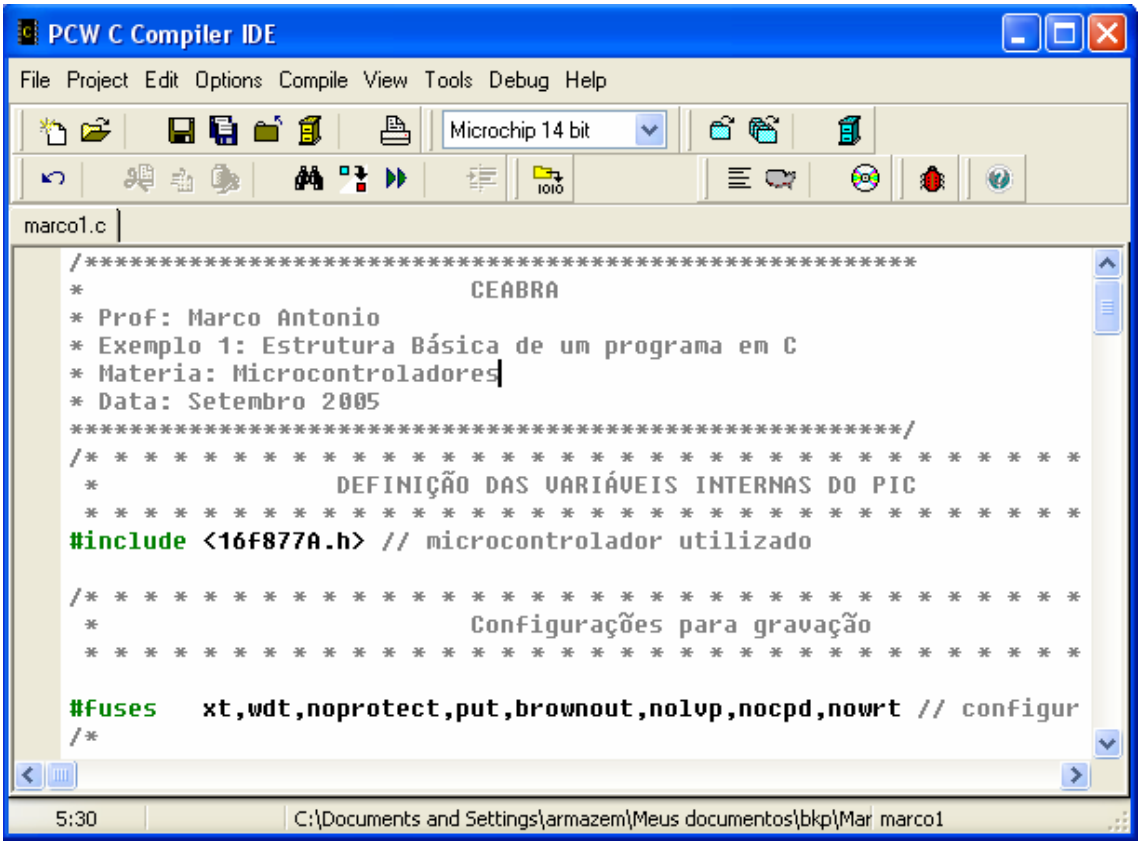


Ilustração 8: Ambiente de desenvolvimento PCWH

Atualmente, a maioria dos microcontroladores disponíveis no mercado conta com compiladores de linguagem C para o desenvolvimento de software. O desenvolvimento em C permite uma grande velocidade na criação de novos projetos permite que o programador preocupe-se mais com a programação da aplicação em si, já que o compilador asssume para si tarefas como o controle e localização das variáveis, operações matemáticas e lógicas, verificação de bancos de memória, etc.

Existe outra linguagem muito usada chamado *Assembly* que é bastante eficiente, mas o programador terá que se preocupar pela localização das variáveis,

pelos estados dos registros controladores de hardware, exigindo do programador o conhecimento do hardware usado.

Sendo nosso curso Técnico escolho-se como principal opção a linguagem C para que o aluno possa aproveitar ao máximo o seu tempo, ampliando desta forma suas habilidades na programação de PICs. Quando um programa se faça crítico no sentido de precisar mais eficiência o programador deverá escolher a linguagem Assembler.

È importante também sinalizar que este programa tem um excelente ambiente de programação (EDITOR), mas nosso curso usará como editor ou ambiente de escrita de programa o MPLAB, programa gratuito da MICROCHIP. O MPLAB como foi dito tem também um ambiente de simulação que será bastante usado. O compilador PCW é necessário na medida em que o MPLAB usará este.

3.1. CRIAÇÃO DE UM PROJETO:

Nesta sessão serão explicados os passos para poder criar um projeto:

- Abrir o MPLAB

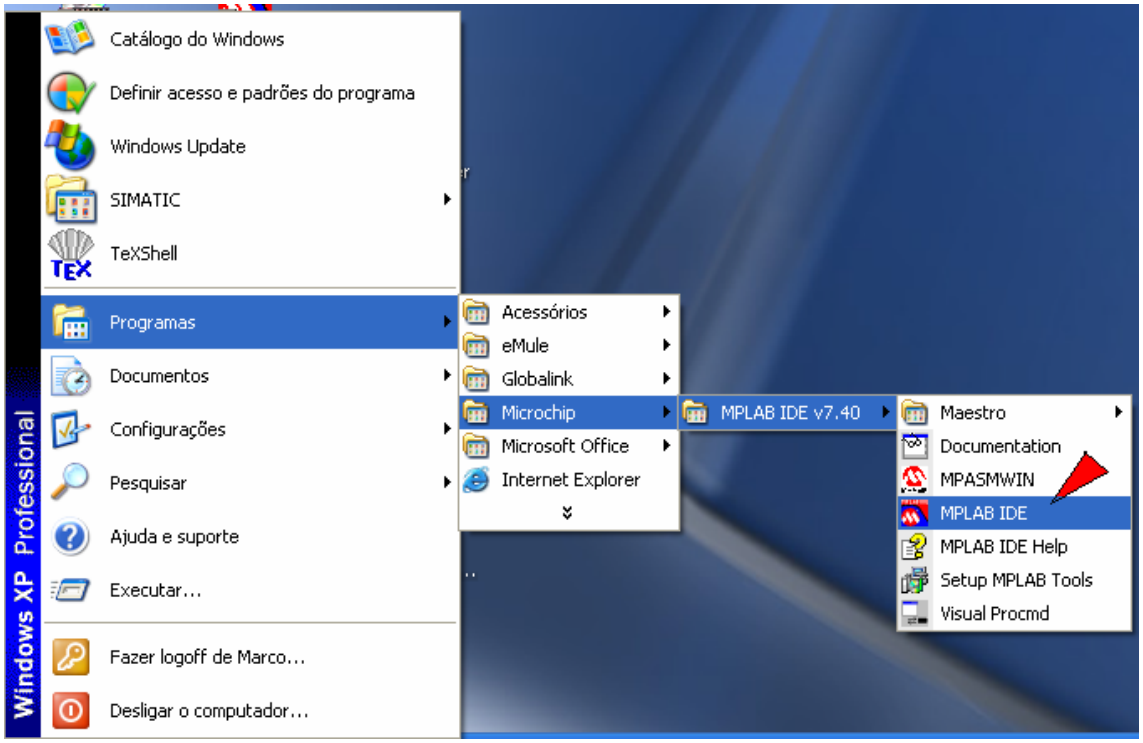


Ilustração 9: Abrindo o MPLAB

Também é possível abrir o MPLAB através do ícone no DESKOP.

- Criar um projeto novo usando o assistente (Wizard), para isto clicar no menu Project / Project Wizard .... como é mostrado na seguinte figura.

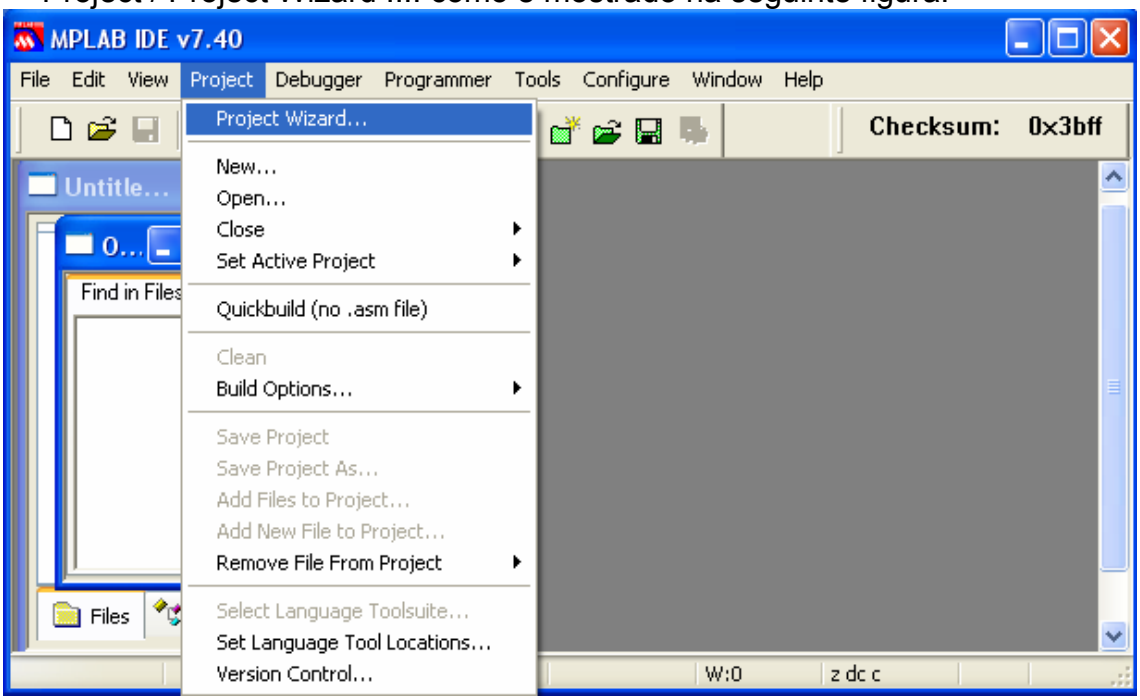


Ilustração 10: Criação de um projeto usando o Assistente (Wizard)

- Avançar

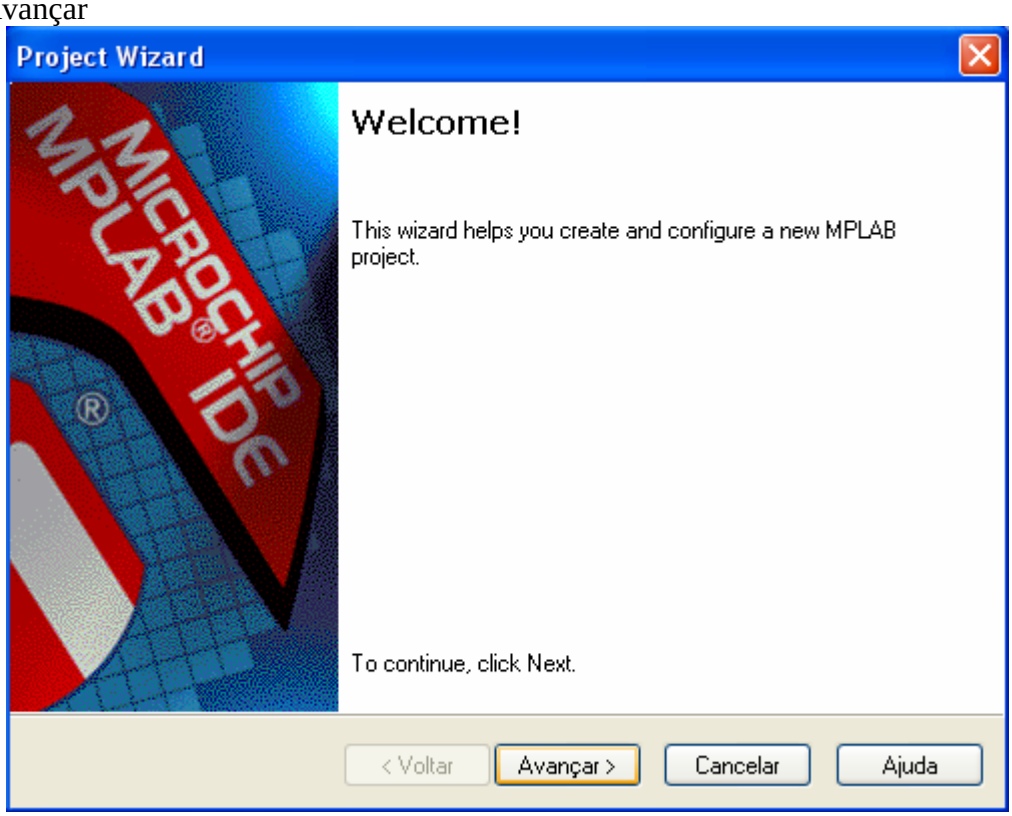


Ilustração 11: Bem vinda do assistente para criação de projeto

- Escolher o tipo de PIC e clicar em avançar

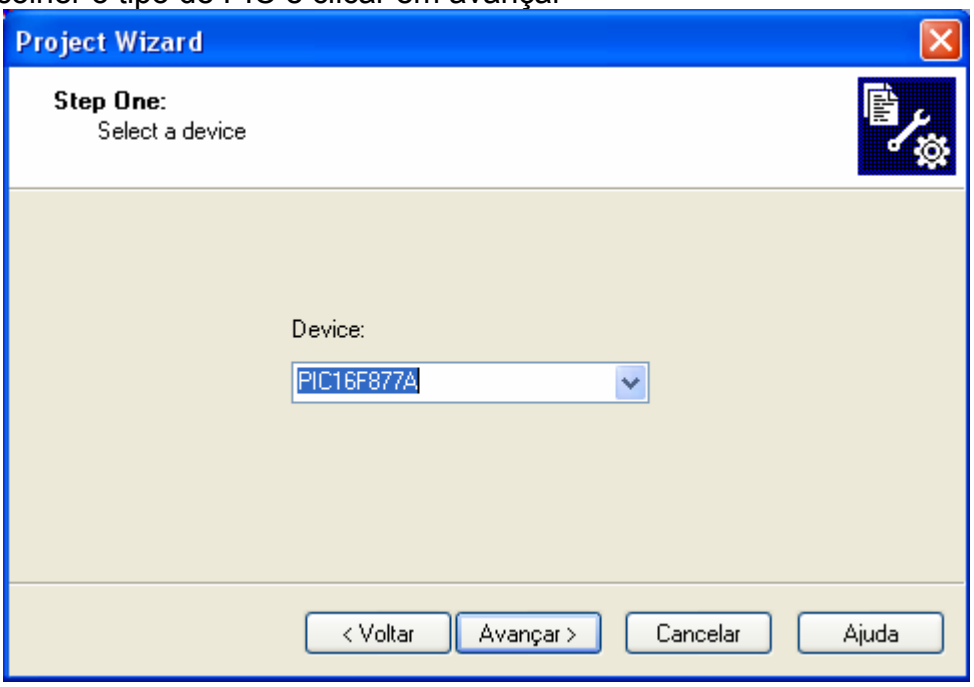


Ilustração 12: Seleção do PIC usando o assistente (wizard)

- Escolha o compilar *"CCS C compiler for PIC12/14/16/18"* e depois em avançar.

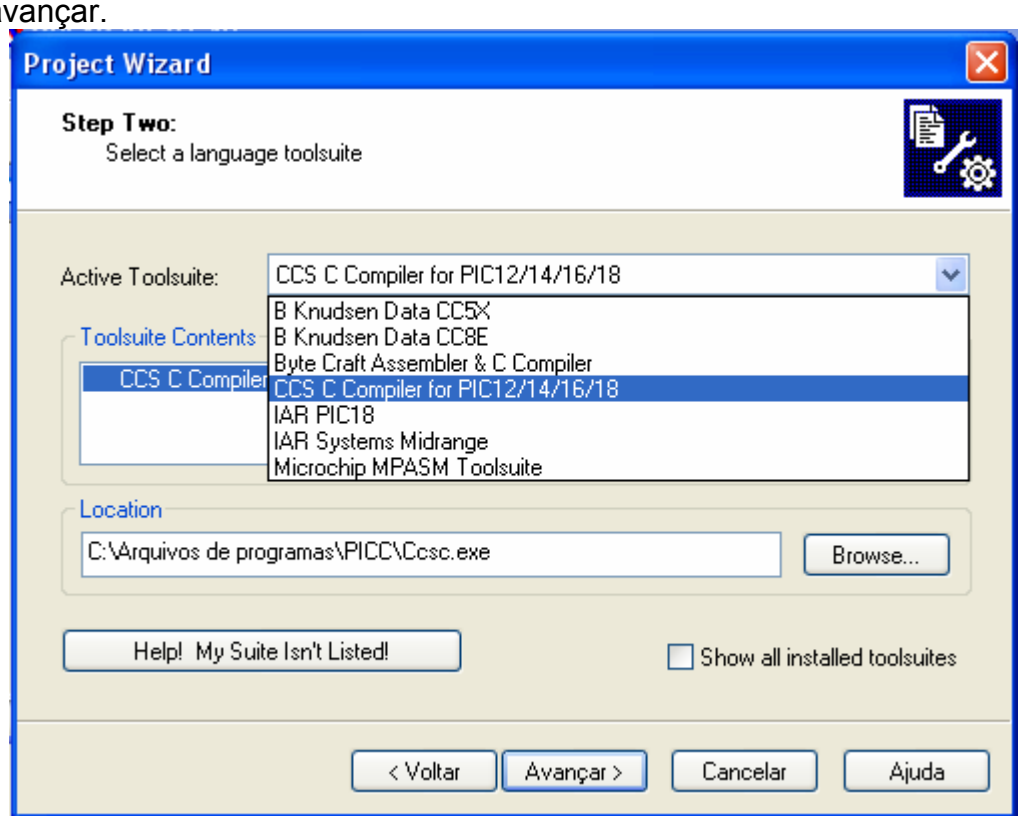


Ilustração 13: Escolha do Compilador CCS no assistente (wizard)

A localização do arquivo **Ccsc.exe** é muito importante. Neste caso esta localizado na pasta "C:\Arquivos de programas\PICC\Ccsc.exe".

- Escrever o nome do projeto e sua localização, logo clicar em avançar.

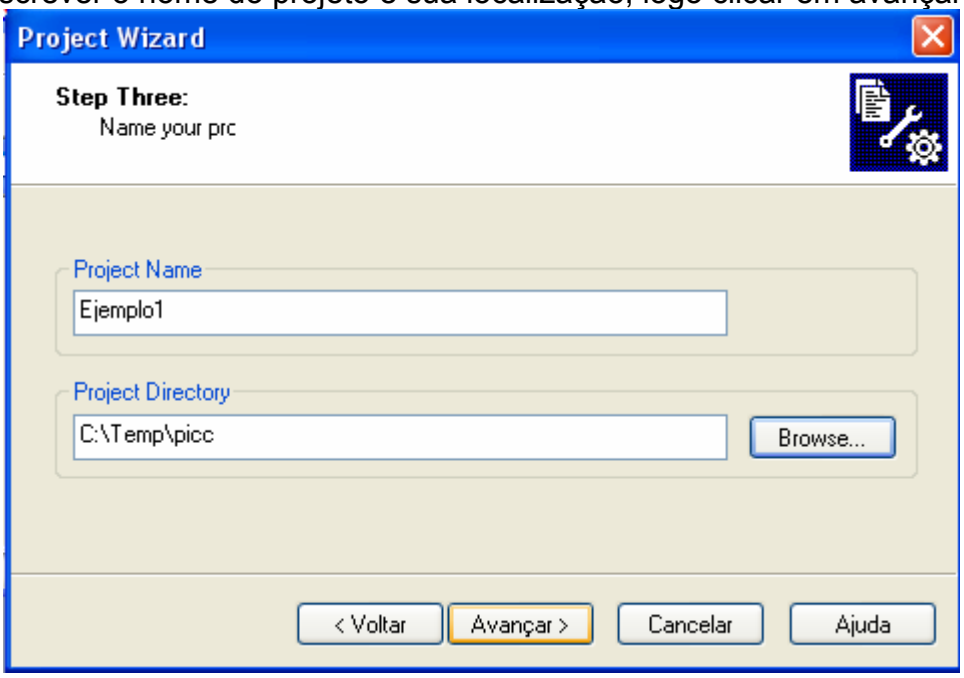


Ilustração 14: Nome do Projeto - Assistente (Wizard)

- Incrementar algum arquivo existente no projeto em caso de ter. Para este curso por enquanto não incrementaremos nenhum arquivo e clicaremos em avançar diretamente.

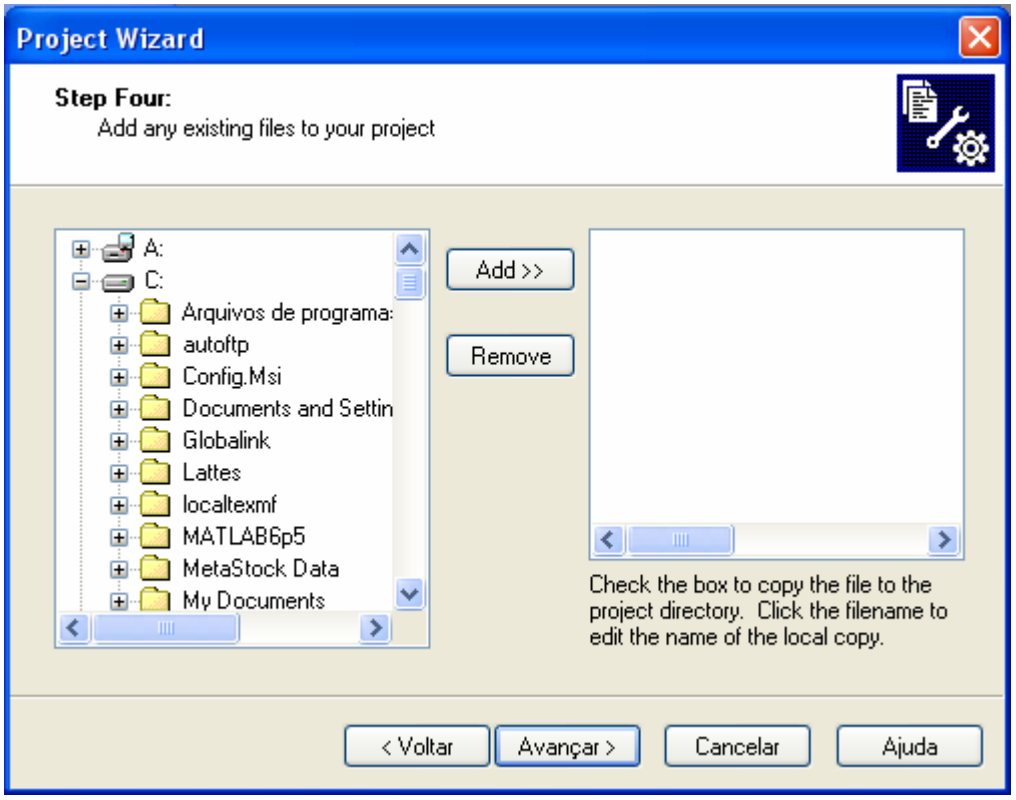


Ilustração 15: Incremento de Arquivos no Assistente - (Wizard)

- Clicar em concluir para criar o projeto com os parâmetros mostrados na tela.

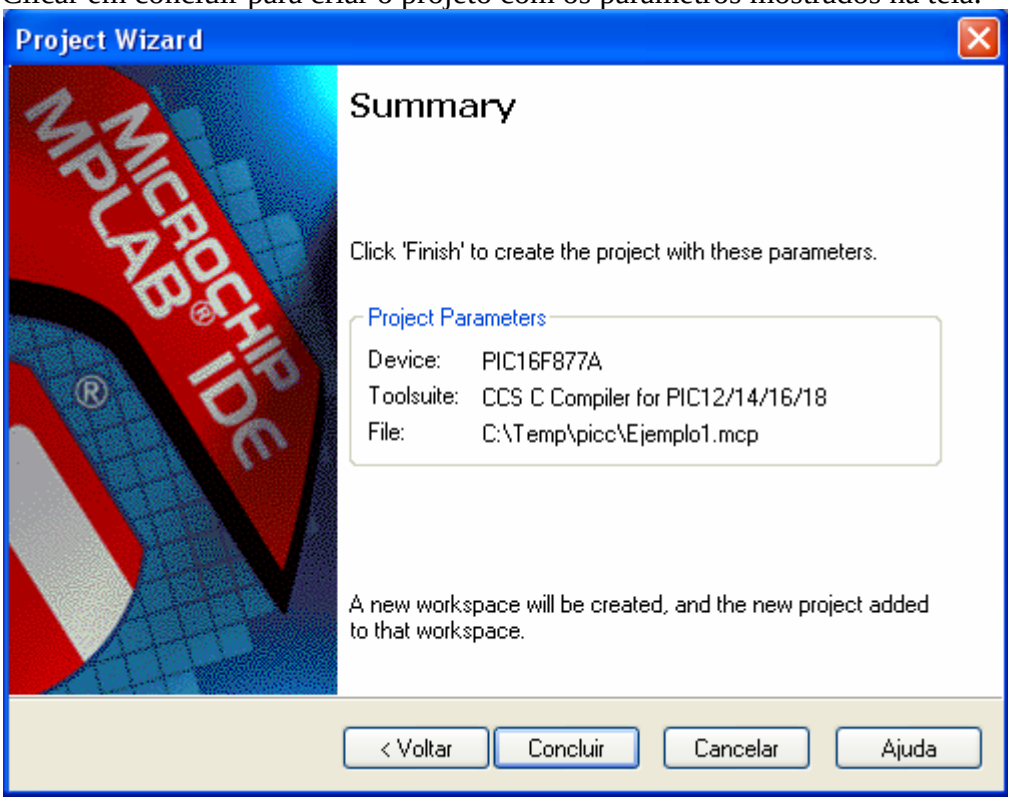


Ilustração 16: Sumário do Projeto - Fim do Assistente (Wizard)

- Finalmente temos o projeto, mas precisamos de um arquivo onde possamos escrever o código, para isto clicamos no menu File/New

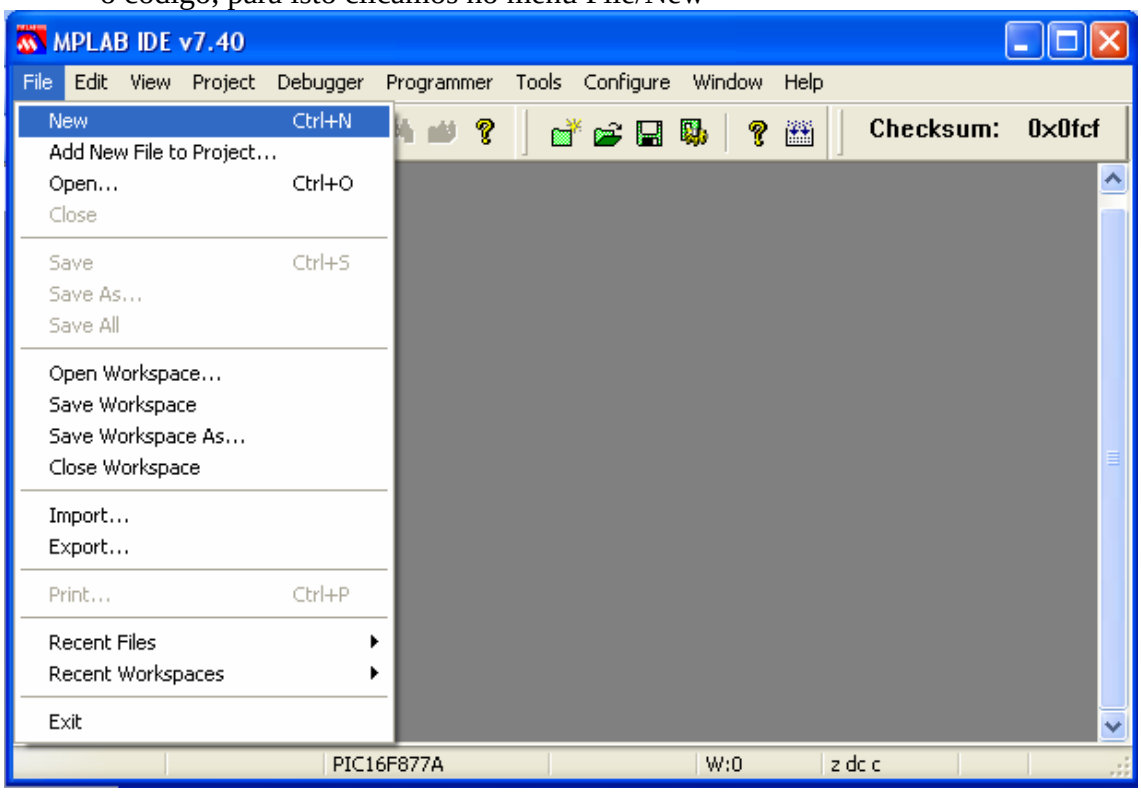


Ilustração 17: Criação de um novo arquivo C

- Escrevemos o programa em código C

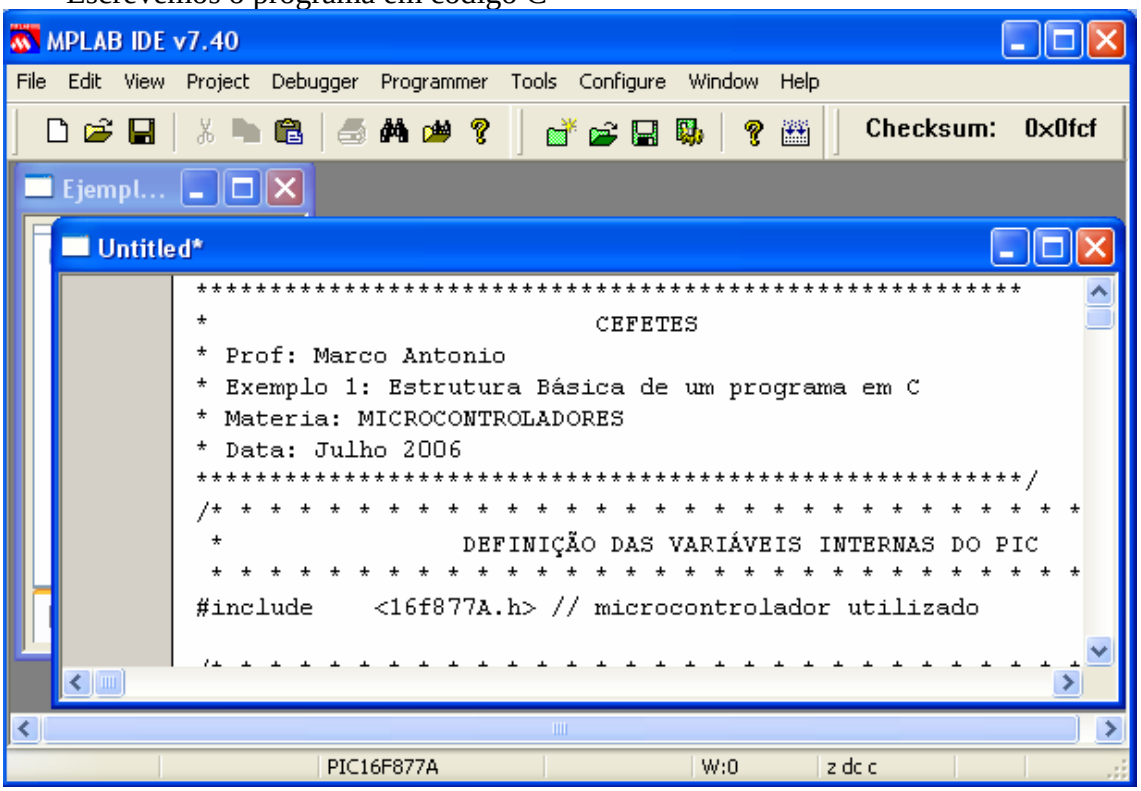


Ilustração 18: Escrita do programe em C

- Depois de escrever o programa deverá salvar este. Clicar em salvar e escrever em nome do arquivo “Ejemplo1.c” e clicar no botão Salvar.

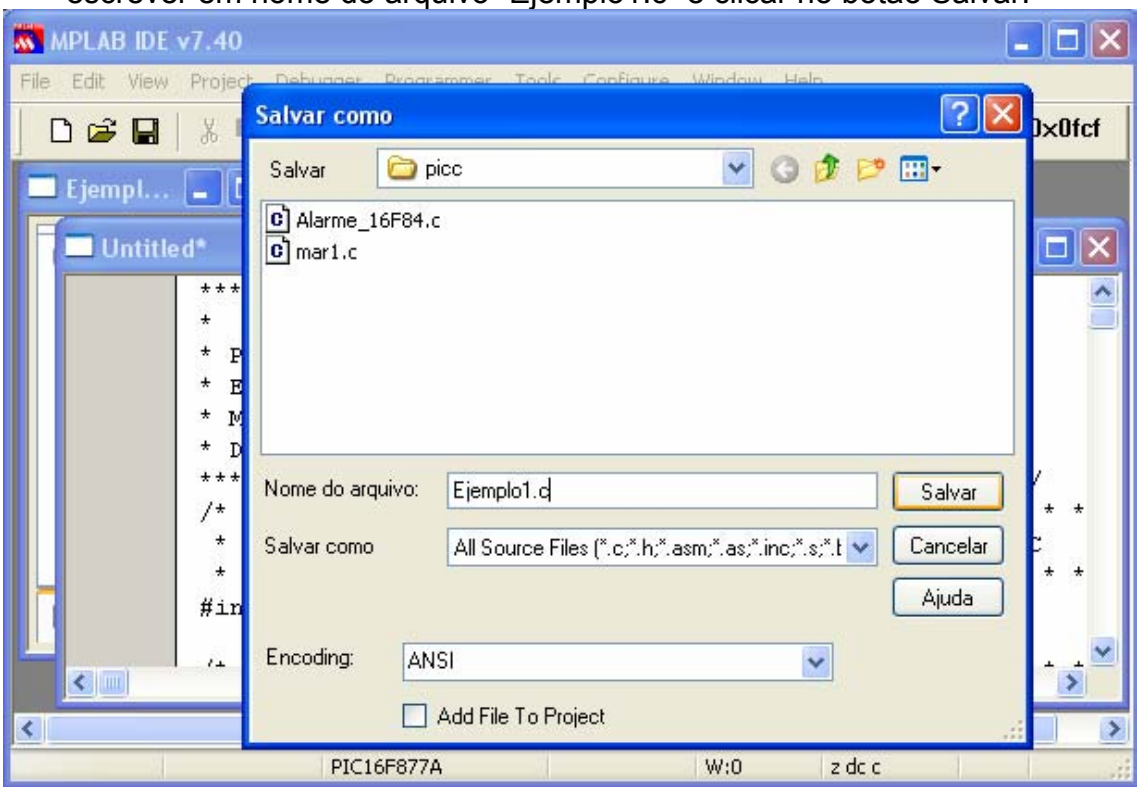


Ilustração 19: Gravação do arquivo "C"

Até aqui o seu projeto esta vazio sem nenhum programa. O programa criado ainda não esta associado ao projeto.

- Com o mouse sobre *Source Files* clicar no botão direito do *mouse*, depois clicar na opção *"Add Files"*.

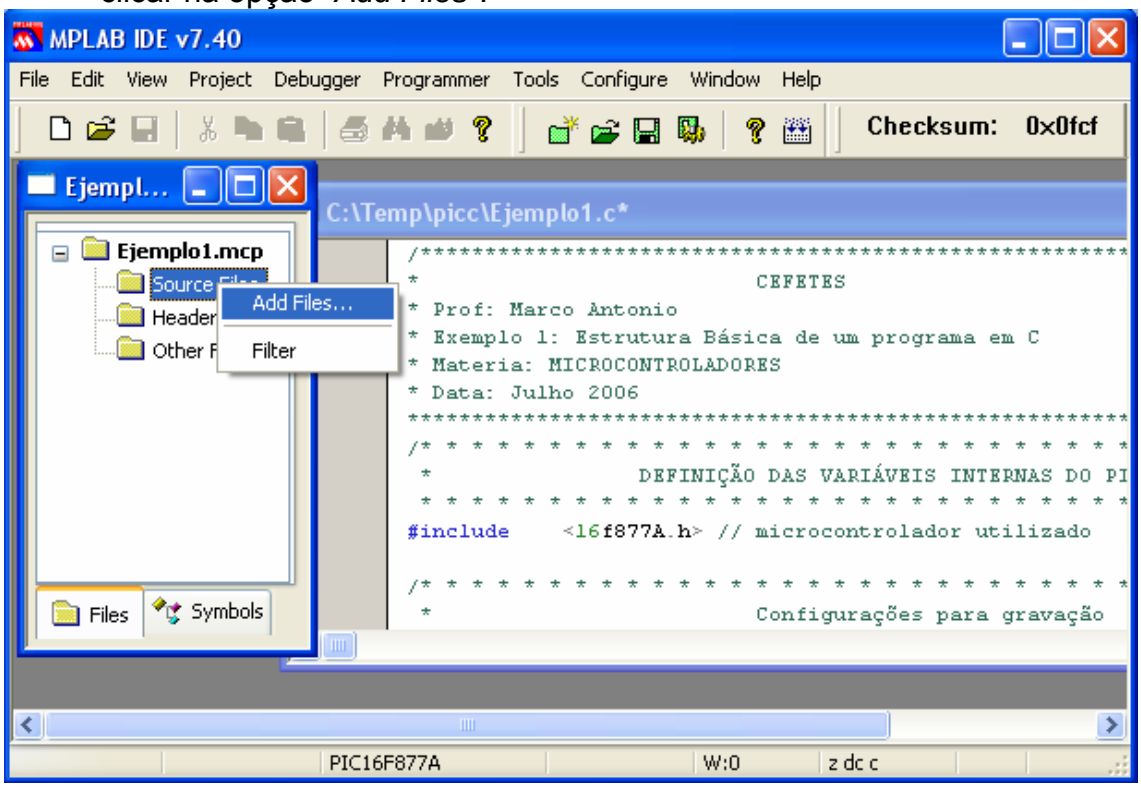


Ilustração 20: Incremento de arquivos ao projeto

- Em examinar escolha o caminho aonde foi gravado o arquivo anteriormente, neste caso C:\Temp\picc. Escolha o arquivo e clicar no botão Abrir.

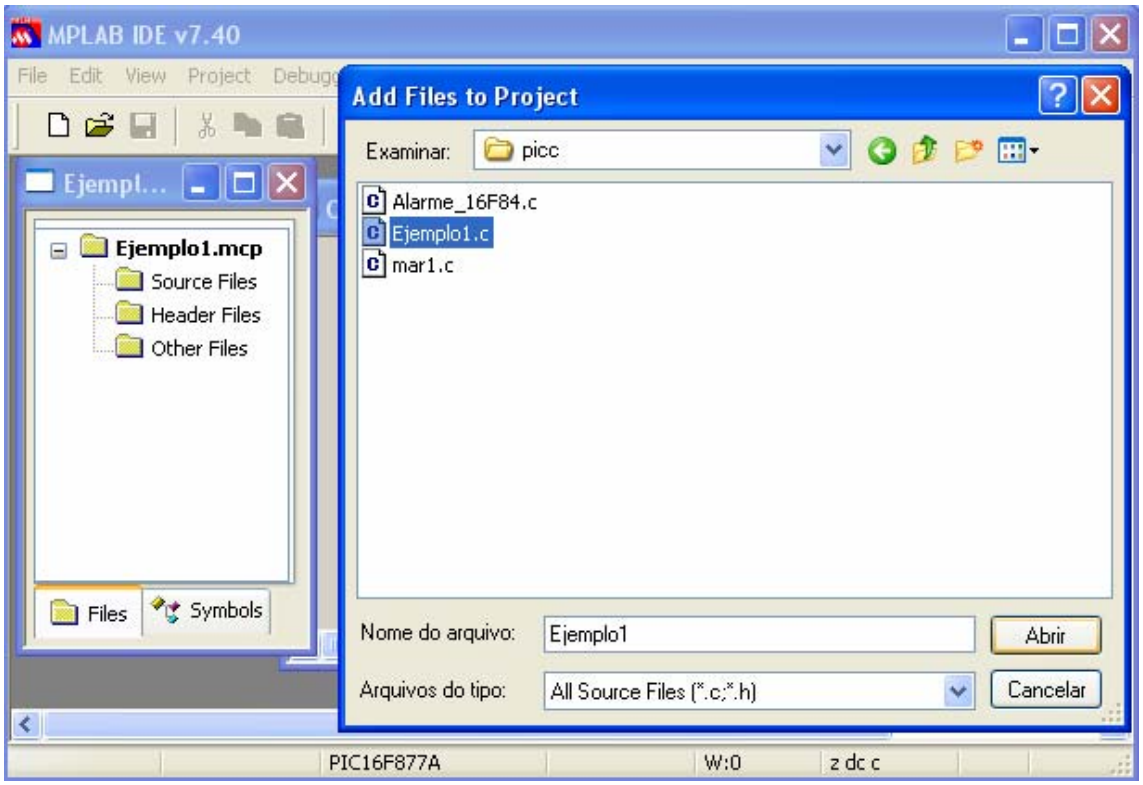


Ilustração 21:Seleção do arquivo.C para ser incrementado no projeto

4. INTRODUÇÃO À LINGUAGEM C – O PRIMEIRO PROGRAMA

Agora estamos em condições de poder começar a escrever um programa no ambiente de desenvolvimento, chegou a hora de estudar os fundamentos da linguagem C.

No projeto criado, abrimos, salvamos e incrementamos um arquivo chamado Ejemplo1.C, neste arquivo podemos escrever o seguinte programa:

EXEMPLO 1: Neste exemplo iremos desenvolver um exercício e explicar a primeira parte do programa e a funcionalidade das funções. Desta forma não será necessário explicar nos exemplos posteriores os itens explicados neste exemplo.

```
#include        <16f877A.h>
#use    delay(clock=4000000, RESTART_WDT)
#fuses  xt,nowdt,noprotect,put,brownout,nolvp,nocpd,nowrt

#use    fast_io(a)
#use    fast_io(b)
#use    fast_io(c)
#use    fast_io(d)
#use    fast_io(e)

#byte    porta = 0x05
#byte    portb = 0x06
#byte    portc = 0x07
#byte    portd = 0x08
#byte    porte = 0x09

#bit     botao = portb.0
#bit     led = portb.1

void main ()
{

        set_tris_a(0b11111111);
        set_tris_b(0b11111001);
        set_tris_c(0b11111111);
        set_tris_d(0b11111111);
        set_tris_e(0b00000111);

        porta=0x00;
        portb=0x00;
        portc=0x00;
        portd=0x00;
        porte=0x00;

        while(TRUE)
        {
                RESTART_WDT();
                if(!botao)
                        led = 1;
                else
                        led=0;
        }

}
```

Vejamos o significado de cada linha do programa.

```
#include        <16f877A.h>
```

È uma diretiva do compilador. Neste caso está determinando ao compilador que anexe ao programa o arquivo especificado **16f877A.h.** Arquivos .h são chamados de arquivos de cabeçalho e são utilizados em C para definir variáveis, tipos, símbolos e funções úteis ao programa.

O **16f877A.h.** é um arquivo com as definições relativas ao processador lavo, para o qual o programa será compilado.

```
#use    delay(clock=4000000, RESTART_WDT)
```

A seqüência #use especifica uma diretiva interna do compilador. Aqui é determinado o valor de 4 MHz para a freqüência do clock.

```
#fuses  xt,wdt,noprotect,put,brownout,nolvp,nocpd,nowrt
```

Configuração dos fusíveis.

XT: O oscilador, da maioria dos PIC, é externo, sem ele nada funciona. Existem vários tipos de osciladores como o RC composto por uma resistência e um capacitor e o cristal XT que é mais preciso sendo também os mais caros.

NO**WDT:** Watchdog timer ou cão de guarda é um recurso poderosíssimo que deve ser utilizado. É um contador automático incrementado por meio de um oscilador próprio, independente do oscilador principal. Caso o WDT estoure, um reset do sistema irá ocorrer imediatamente.

noprotect: permite reprogramar o PIC e enxergar o código.

put: prevê que o programa não comece antes que a alimentação seja inicializada. Esta opção irá fazer com que o PIC comece a operar cerca de 72 ms após o pino /MCLR seja colocado em nível alto.

Brown-Out: é utilizado para forçar um reset quando a tensão de alimentação sofre uma pequena queda. Ele é extremamente recomendado em projetos que possibilitam ao usuário desligar e religar rapidamente a alimentação.

NOLVP Low Voltage Programming disabled

Trabalho 4.1: Pesquisar para a próxima aula as outras diretivas não comentadas.

```
#use    fast_io(a)
#use    fast_io(b)
#use    fast_io(c)
#use    fast_io(d)
#use    fast_io(e)
```

Afeta como o compilador gerará código para as instruções de entrada ou saída que seguem. Esta diretiva tem efeito até que uma nova diretiva #use xxxx_IO seja encontrado. Este método rápido de configuração de configuração de I/O a configurar corretamente a direção de I/O usando a instrução set_tris_X().

```
#byte   porta = 0x05
#byte   portb = 0x06
```

```
#byte   portc = 0x07
#byte   portd = 0x08
#byte   porte = 0x09
```

No data sheet do PIC 16F877A podemos observar que as portas a,b,c,d, e estão associados a um endereço de memória. Depois destas linhas as portas terão como identificador as variáveis porta, portb até porte.

```
#bit    botao = portb.0
#bit    led = portb.1
```

As entradas devem ser associadas a nomes para facilitar a programação e futuras alterações do hardware. De aqui em diante podemos nomear o pino portb.0 como botão por exemplo.

```
void main ()
```

A declaração **main()** especifica o nome de uma função. No caso, a função main() é padronizada na linguagem C e é utilizada para definir a função principal, ou corpo principal do programa.

O sinal de abertura de chave "{" é utilizado para delimitar o início da função e o sinal de fechamento de chave "}" indica o final da função. Na verdade, as chaves delimitam o que chamamos de bloco de programa, ou bloco de código.

```
set_tris_a(0b11111111);
set_tris_b(0b11111001);
set_tris_c(0b11111111);
set_tris_d(0b11111111);
set_tris_e(0b00000111);
```

Esta função permite direcionar as I/O. Deve de ser usado com o comando "fast_io" e quando as portas de I/O estão acessíveis, para isto deve ser usada a diretiva #BYTE para o acesso da porta. Cada bit no valor representa um pino. O "1"(um) representa entrada e o zero saída.

Exemplo: SET_TRIS_B(0x0F); similar a SET_TRIS_B(0b00001111)
// B7,B6,B5,B4 are outputs
// B3,B2,B1,B0 are inputs

```
porta=0x00;
portb=0x00;
portc=0x00;
portd=0x00;
porte=0x00;
```

Inicializando as portas, todos os pinos das portas na execução destas linhas tomarão o valor de zero ou desligado.

```
while(TRUE)
```

Este é um comando de controle utilizado na repetição de um determinado bloco de instruções. Esse bloco será repetido enquanto a avaliação da condição especificada entre parênteses for verdadeira. No caso, a avaliação é explicitamente verdadeira (true).

O bloco de instruções que será repetido é aquele especificado dentro das chaves que seguem o comando **while.**

```
if(!botao)
        led = 1;
else
```

```
                    led=0;
```

O comando if (em português “se”) é utilizado para avaliar a condição que esta entre parênteses. Em caso de que esta condição seja verdadeira o comando seguinte será executado. Em caso de ser falsa (else) o comando depois da instrução “else” será executado. Neste exemplo o a variável booleana botão esta negada (!botao) quer disser se o botão estivesse em estado zero a condição será falsa e o led será igual a 1. Em caso que o botão seja 1 a condição será falsa e o led será igual a zero.

Trabalho 4.2: Responda as seguintes perguntas:

- No programa do exemplo 1 disser como esta configurado o pino portb.4?. è uma entrada ou uma saída e por quê?
- A variável led esta associada a que pino do PIC?. Responder com o número do pino.
- Porque no exemplo 1 foram consideradas 5 portas?. Explicar olhando o desenho do PIC.
- No programa em que linha decide o tipo de PIC a ser usado?
- Que mudaria no programa para que quando botão = 1 o led seja 1?
- Incremente no exemplo1 a variável botao1 e led1. Quando botão1 seja igual a 1 led1 deverá de ser zero e quando botão1 seja zero, led1 será 1.
- Esquematize o circuito eletrônico para testar este programa. Utilize um oscilador de cristal e lembre que o limite de corrente oscila entre os 20mA por cada pino de saída. A tensão de alimentação do PIC é de 5 v. Instale um botão que aoi ser pressionado reiniciará o PIC (reset).

Até aqui se espera que o leitor consiga associar o pino do PIC às portas mencionadas no programa, e que possa entender este exemplo que mesmo sendo básico é o início do aprendizado de um recurso sem limites. Até a próxima aula onde aprenderemos a simular programas sem a necessidade de um hardware.

5. USO DO MPSIM PARA SIMULAÇÃO

Neste capítulo o objetivo é que o leitor consiga aprender a simular o seu projeto. Em outras palavras sem necessidade do circuito eletrônico podemos conferir se o programa pode compilar, (esta sem erros de sintaxe) e responde da forma que se quer (sem erros lógicos).

O MpSim é um utilitário do MpLab e tem a finalidade de simular o projeto. Para poder fazer uso deste deverá de seguir os seguintes passos:

1. Escolher a ferramenta MpLab SIM no menu Debugger/Select Tool/MpLab SIM.

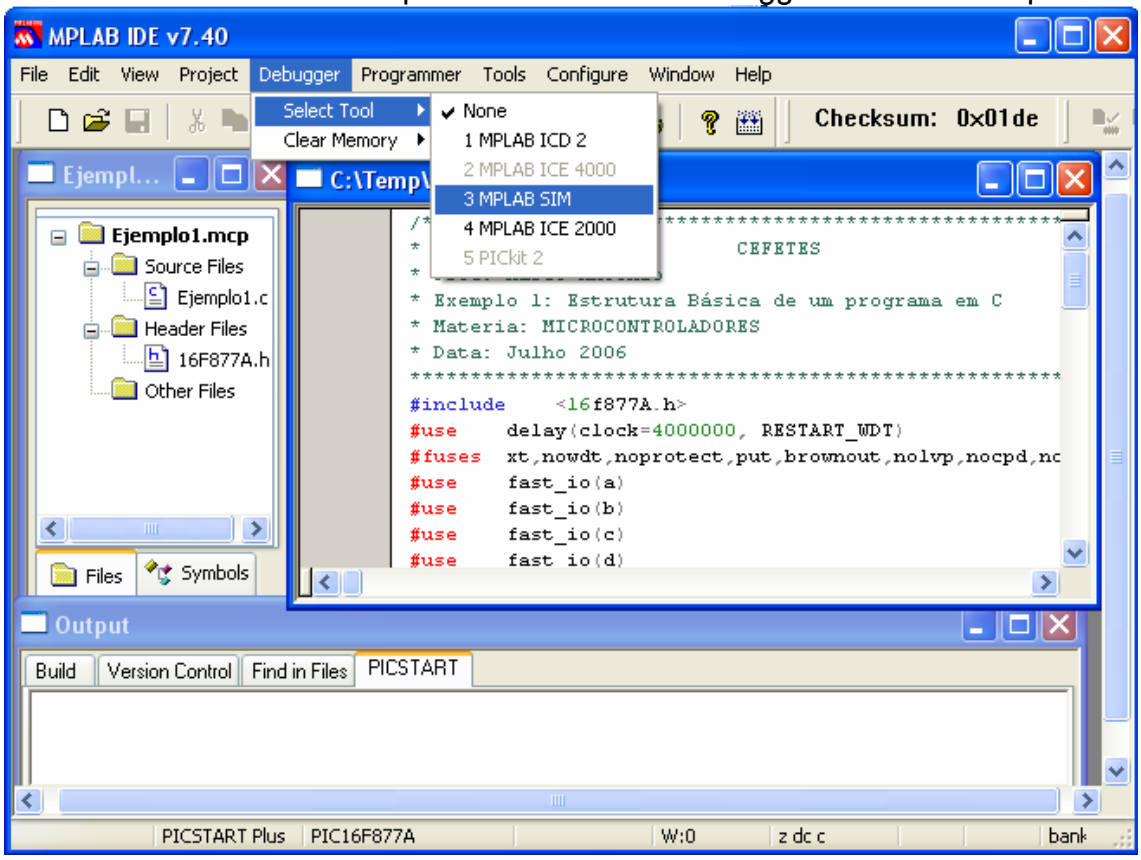


Ilustração 22: Escolha da ferramenta MpLab SIM

2. Compilar o seu projeto, clicando no ícone mostrado no gráfico ou pressionando F10.

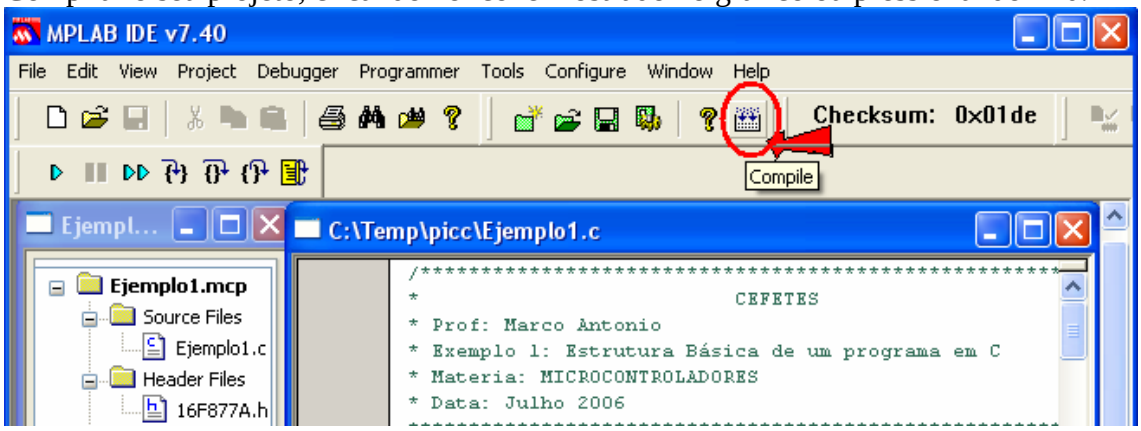


Ilustração 23: Compilação do projeto

3. Observe na janela OutPut o resultado da compilação, neste caso a compilação foi bem sucedida (build succeeded), mostrando o caminho aonde foram copiados os arquivos produto desta compilação. Também mostra o número de erros da compilação e de avisos (warnings). Em caso de erro este será mostrado, e indicado em que linha se encontra. Se sua compilação contiver algum erro, proceda a sua localização e correção.

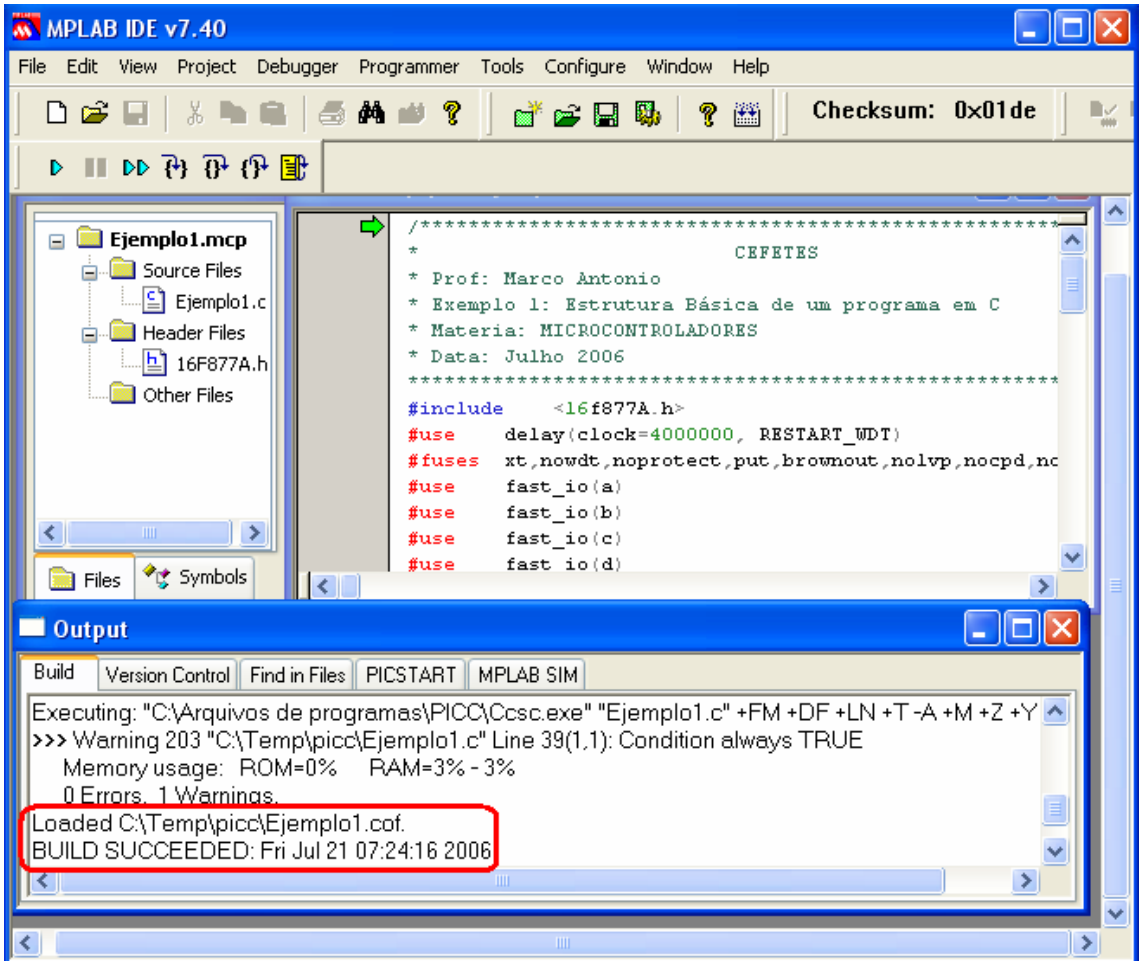


Ilustração 24: Visualização do resultado da compilação

Exercícios 5.1:

1. Incremente a palavra Brasil depois da linha de programa “while(TRUE)” e compile o projeto.
2. Em que linha se apresentou o erro?. Verifique a barra de indicação inferior na janela do Mplab que mostra a linha e a coluna na qual se encontra o cursor.
3. Porque ao incrementar esta palavra provocou ao compilador o erro?

Nos programas é comum o uso de comentários para poder incrementar um comentário é necessário escrever while(TRUE) **//**Brasil. Estes caracteres “//” antecedem a um comentário de linha simples. São utilizados para descrever o funcionamento ao final de cada linha de código.

4. Agora é hora de aprender a usar as ferramentas deste utilitário, no seguinte gráfico podemos observar que temos 7 botões que controlam a simulação.

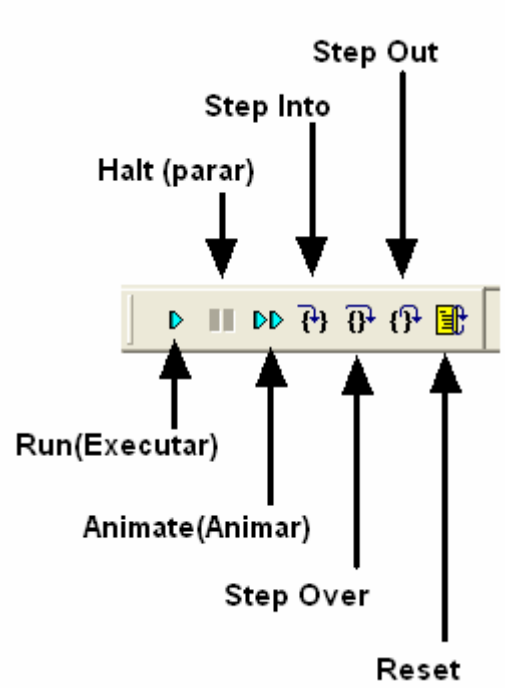


Ilustração 25: Botões de controle da simulação no MPSIM

Run: Executa o programa, começa a simulação.
Halt: Para a simulação.
Animate: Anima a simulação, executa linha por linha o programa e é indicado por uma zeta.
Step Into: Executa linha, e se houver uma função entra nela.
Step Over: Executa a linha, e se houver uma função executa o bloco todo e pula para a seguinte linha.
Step out: Saí da função
Reset: reinicia a simulação a partir da primeira linha.

Exercícios 5.2:

1. Pressionar o botão ***Animate*** e anotar suas observações. Para parar a animação clicar no botão ***halt.***
2. Clicar no botão ***Run*** e anotar suas observações, que esta acontecendo?
3. Clicar várias vezes o botão ***Step over*** e anotar suas observações?

5. Não tem sentido visualizar a simulação do programa se não conseguir estimular as entradas para poder observar o comportamento das saídas. Para isto existe uma ferramenta no MPSIM, clicar na opção do menu Debugger/Stimulus Controller/New Scenario como se mostra na ilustração seguinte.

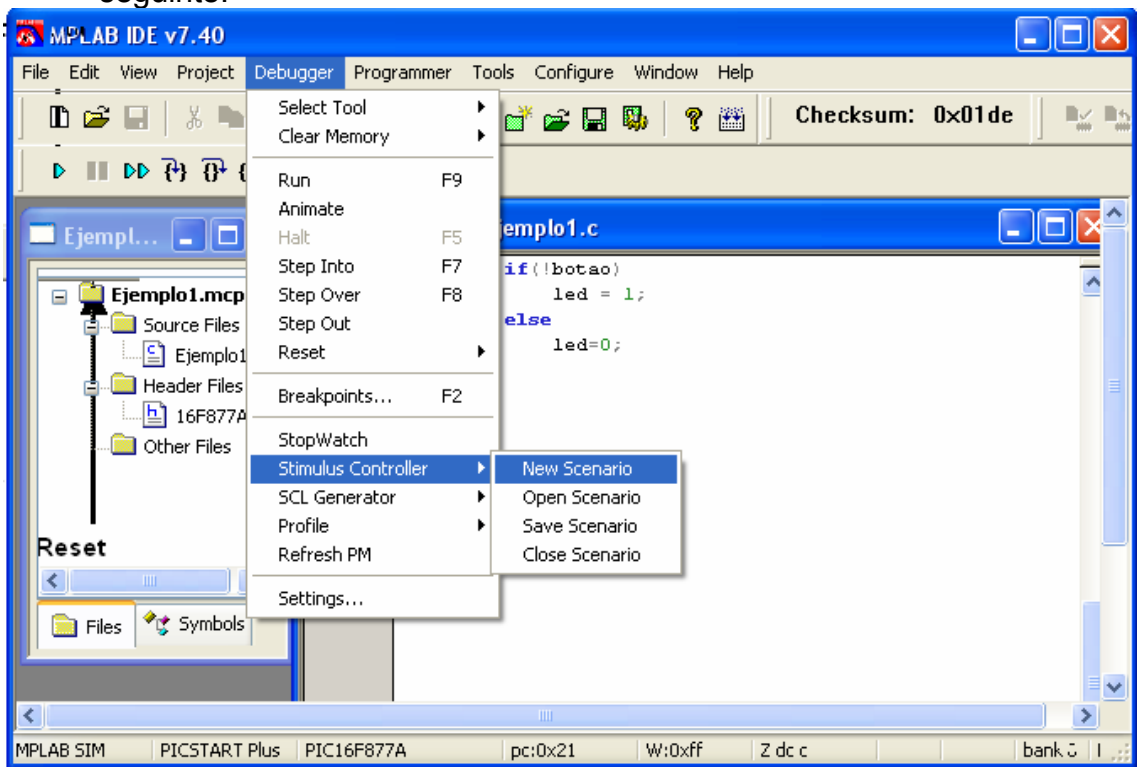


Ilustração 26: Criação de um controle de estimulo

6. Como as entradas têm que ser simuladas neste ambiente, criamos as entradas que serão estimuladas ou controladas nesta janela.

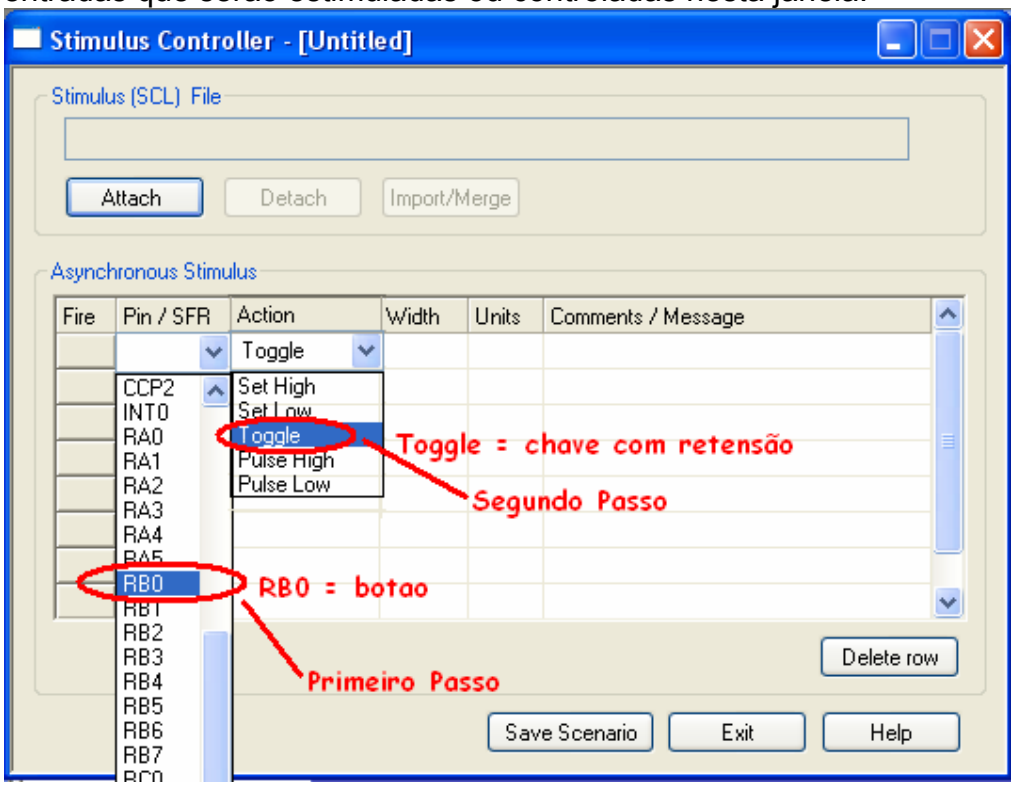


Ilustração 27: Parametrização de um estimulo ou entrada digital no simulador MPSIM

Escolhemos RB0 porque no exemplo 1 este representa a variável botão, e o tipo de ação do estimulo terá retenção. Em outras palavras botão será uma chave com retenção.

Este ambiente pode ser salvo e aberto. Na ilustração anterior observa-se que existem 5 tipos de ação:

- *Set High:* O Pino de referência tomará o valor de 1. Para que seja novamente zero necessariamente terá que ser executada a ação *Set Low.*
- *Set Low:* O Pino de referência tomará o valor de 0. Para que seja novamente um(1) necessariamente terá que ser executada a ação *Set High.*
- *Toggle:* Ao ser executada esta ação o pino de referência mudará de estado, se estiver em zero este irá para um, se estiver em um irá para zero. Esta ação atua como uma chave com retenção.
- *Pulse high:* Ação que manterá ligado o pino respectivo só um instante. Em outras palavras enviará um pulso.
- *Pulse low:* Ação que manterá desligado o pino respectivo só um instante. Em outras palavras enviará um pulso em sentido contrário a *Pulse high*.

Como já foi percebido na coluna PIN/SFR são selecionados os pinos do PIC que serão estimulados.

As colunas *Width e units*, são usados em caso de que a ação selecionada seja o *pulse high* ou *low*, na coluna *width* (largura) é especificada a largura do pulso, e na coluna *units* a unidade respectiva.

Para que a ação seja executada o usuário deverá clicar o botão da coluna *fire*.

As vezes a desorganização faz perder as pessoas um valioso tempo, na programação a organização esta relacionada entre outras coisas com os comentários, A coluna *comments* serve para escrever comentários ou até para simular entradas de mensagens pela porta serial.

Agora o leitor deve estar preparado para simular o seu projeto com estímulos de entrada.

Exercícios 5.3:

1) Pressionar o botão ***Step Over*** nos botões de controle de simulação e verifique se a linha de comando depois da instrução ***if*** é executada. Na seguinte barredura estimule o variável botão e verifique novamente a execução desta linha. Anote suas observações, que esta acontecendo?

2) Incremente na janela de controle de estímulos ações de *Set Low e Set high* para o botão e visualize este funcionamento.

7. O fato de poder estimular as entradas não ajuda muito se não se visualiza com maior clareza a reação das saídas. Existem várias formas de visualizar o comportamento das saídas, mas por enquanto se usará o opção localizada no menu View/Watch.

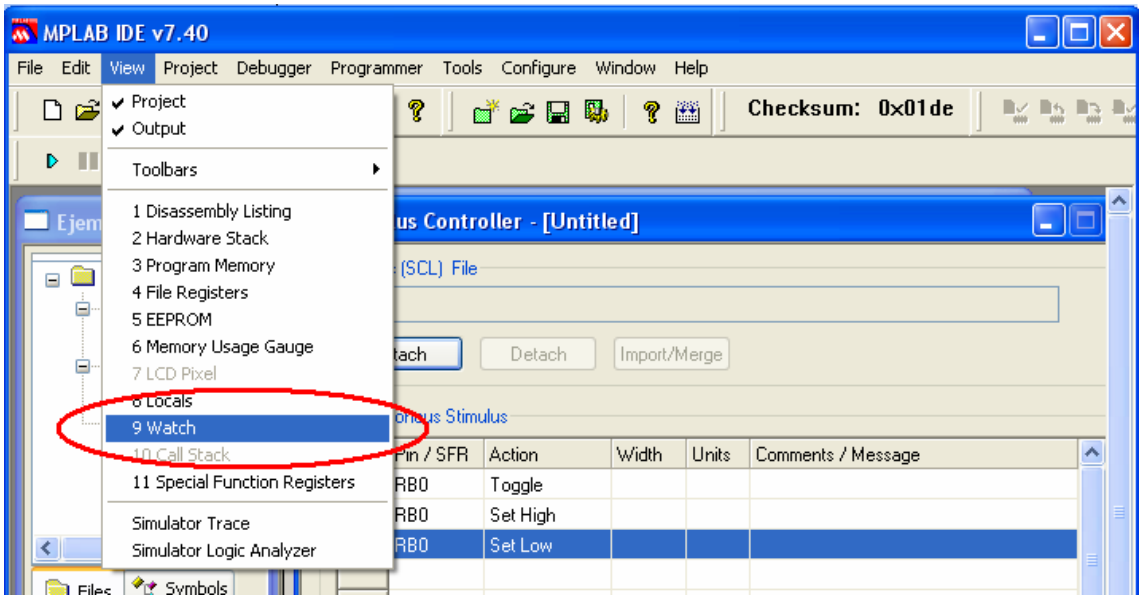


Ilustração 28: Abrindo um visualizador de registros – *watch*

Watch, permite você digitar uma variável, ou qualquer expressão importante e **visualizar** o resultado. Neste caso escolhemos a variável led e clicamos no botão *Add Symbol*, para poder mudar as características de visualização clicar com o botão direito sobre a variável em questão e selecionar *properties* e na janela emergente selecione o sistema de numeração binário, feche a janela e simule usando o botão ***Step Over*.** O watch Permite visualizar tanto saídas como entradas.

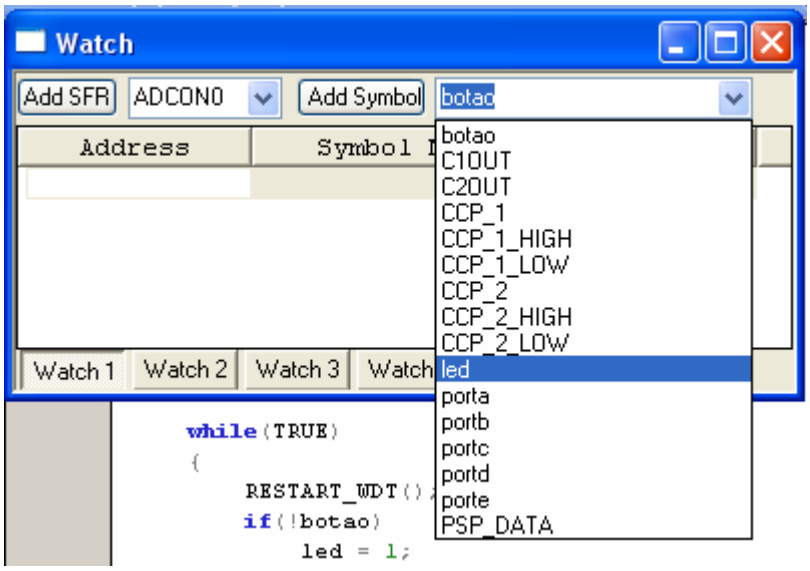


Ilustração 29: Janela da opção Watch

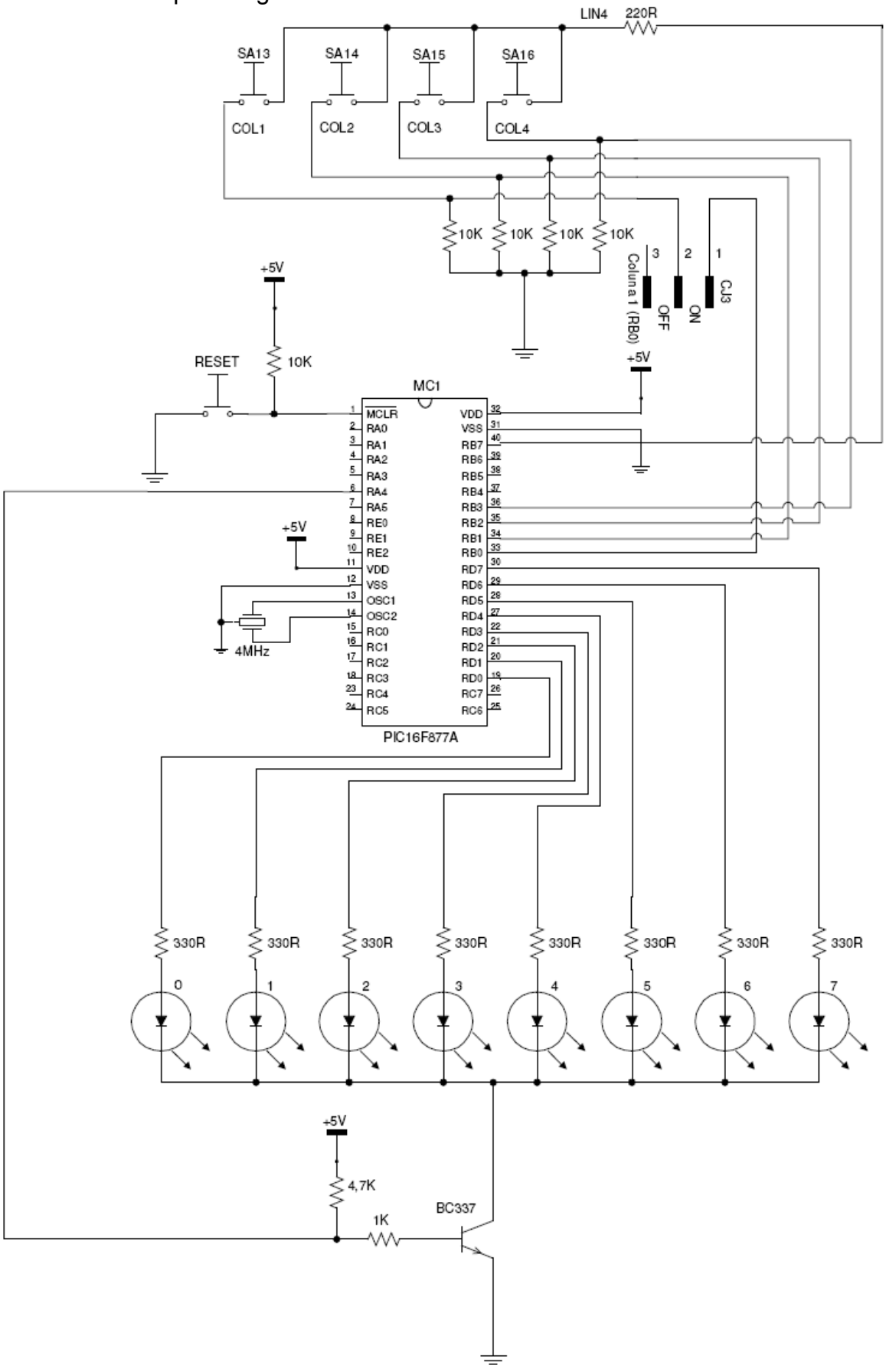
Exercícios 5.4:

1. Inserir na janela watch o byte portb em binário e simular novamente o programa observando detalhadamente os resultados.

2. Simular o seguinte programa e descrever o seu funcionamento. Existe um erro neste programa identifique-lo.

```
/************************************************************
*                    CEFETES
* Prof: Marco Antonio
* Exemplo 2: Função IF - reforço
* Materia: MICROCONTROLADORES
* Data: Julho 2006 /Linhares
************************************************************/
#include    <16f877A.h>
#use        delay(clock=4000000, RESTART_WDT)
#fuses      xt,nowdt,noprotect,put,brownout,nolvp,nocpd,nowrt
#use        fast_io(a)
#use        fast_io(b)
#use        fast_io(c)
#use        fast_io(d)
#use        fast_io(e)

#byte       porta = 0x05
#byte       portb = 0x06
#byte       portc = 0x07
#byte       portd = 0x08
#byte       porte = 0x09

#bit BotaoLiga = portb.0
#bit BotaoDesliga = portb.1
#bit Motor1 = portb.2
#bit Lampada = portb.3

void main ()
{
    set_tris_a(0b11111111);
    set_tris_b(0b11111011);
    set_tris_c(0b11111111);
    set_tris_d(0b11111111);
    set_tris_e(0b00000111);

    porta=0x00;
    portb=0x00;
    portc=0x00;
    portd=0x00;
    porte=0x00;

    while(TRUE)
    {
            if(!BotaoLiga)
            {
                    Motor1 = 1;
                    Lampada=1;
            }
            if(!BotaoDesliga)
            {
                    Motor1 = 0;
                    Lampada=0;
            }

    }
}
```

3. Avalia o circuito seguinte e faz um programa para que apertando o botão SA14 possa ligar o led 0.

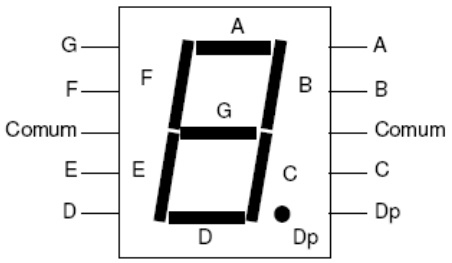


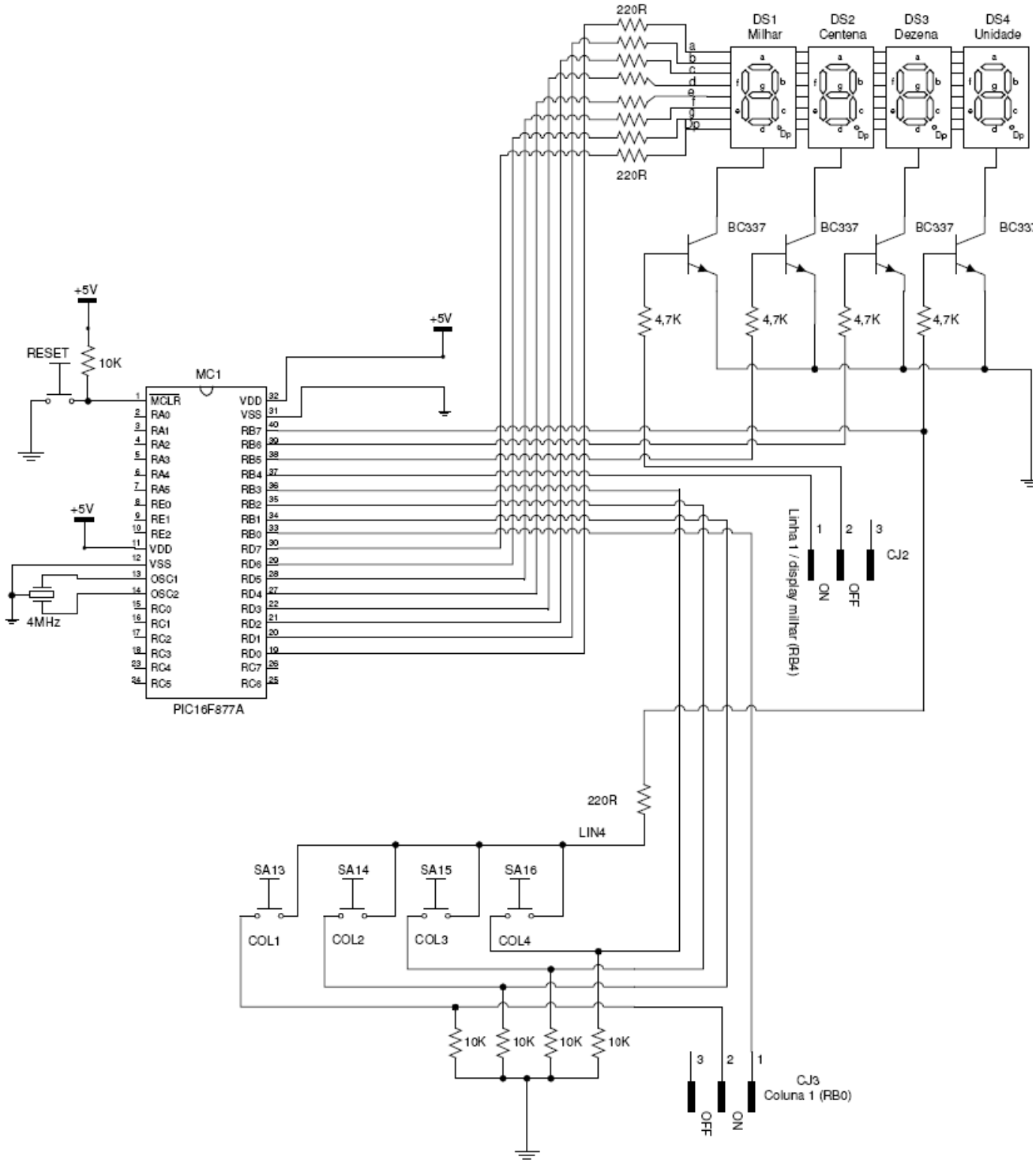
```
/*******************************************************
*                    CEFETES
* Prof: Marco Antonio
* Exemplo 1: Estrutura Básica de um programa em C
* Materia: Microcontroladores
* Data: Agosto 2006
*******************************************************/
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *            DEFINIÇÃO DAS VARIÁVEIS INTERNAS DO PIC            *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
#include     <16f877A.h>     // microcontrolador utilizado

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                  Configurações para gravação                 *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#fuses       xt,wdt,noprotect,put,brownout,nolvp,nocpd,nowrt          // configuração dos fusíveis

#use         delay(clock=4000000, RESTART_WDT)
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *            Definição e inicialização das variáveis             *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
//Neste bloco estão definidas as variáveis globais do programa.
//Este programa não utiliza nenhuma variável de usuário

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                   Constantes internas                        *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
//A definição de constantes facilita a programação e a manutenção.
//Este programa não utiliza nenhuma constante de usuário

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *              Declaração dos flags de software                *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
//A definição de flags ajuda na programação e economiza memória RAM.
//Este programa não utiliza nenhum flag de usuário

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *             Definição e inicialização dos port's             *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#use         fast_io(a)
#use         fast_io(b)
#use         fast_io(c)
#use         fast_io(d)
#use         fast_io(e)

#byte        porta = 0x05
#byte        portb = 0x06
#byte        portc = 0x07
#byte        portd = 0x08
#byte        porte = 0x09

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                      ENTRADAS                                *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
#bit BotaoSA14_b1 = portb.1

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                      SAÍDAS                                  *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
```

```
#bit HabLed = porta.4
#bit HabBotao = portb.7

#bit         Led0 = portd.0

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                Configurações do Microcontrolador             *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

void main ()
    {
    // configura CONFIG
    setup_counters(RTCC_INTERNAL, WDT_2304MS);


    // configura os TRIS
    set_tris_a(0b00100000);
    set_tris_b(0b00001111);
    set_tris_c(0b10011001);
    set_tris_d(0b00000000);
    set_tris_e(0b00000000);

    // inicializa os ports
    porta=0x00;                 // limpa porta
    portb=0x00;                 // limpa portb
    portc=0x00;                 // limpa portc
    portd=0x00;                 // limpa portd
    porte=0x00;                 // limpa porte

    HabLed =1;
    HabBotao=1;
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                     Loop principal                    *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
    while(TRUE)
            {
            RESTART_WDT();
            if(BotaoSA14_b1)            // testa botão
                    Led0 = 1;       // Se botão = 0, então led = 1
            else
                    Led0=0;             // caso contrário, led = 0

         }
                              // FIM DO PROGRAMA
    }
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                 Fim do Programa                              *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
```

4. Lembrando do seguinte circuito:

Faz um programa para que apertando o botão SA13 mostre no display DS4 o número 7 e com o botão SA14 o número 9. Escolhe outro botão para desligar todos os leds do display.





```
/* ************************************
 *          DEFINIÇÃO DAS VARIÁVEIS INTERNAS DO PIC          *
 ************************************/
#include          <16f877A.h>     // microcontrolador utilizado

/* ************************************
 *                Configurações para gravação              *
 ************************************/

#fuses  xt,wdt,noprotect,put,brownout,nolvp,nocpd,nowrt          // configuração dos fusíveis

#use     delay(clock=4000000, RESTART_WDT)
/* ************************************
 *            Definição e inicialização das variáveis            *
 ************************************/
//Neste bloco estão definidas as variáveis globais do programa.
//Este programa não utiliza nenhuma variável de usuário

/* ************************************
 *                  Constantes internas                  *
 ************************************/


/* ************************************
 *              Declaração dos flags de software              *
 ************************************/
//A definição de flags ajuda na programação e economiza memória RAM.
//Este programa não utiliza nenhum flag de usuário

/* ************************************
 *            Definição e inicialização dos port's            *
 ************************************/

#use     fast_io(a)
#use     fast_io(b)
#use     fast_io(c)
#use     fast_io(d)
#use     fast_io(e)

#byte    porta = 0x05
#byte    portb = 0x06
#byte    portc = 0x07
#byte    portd = 0x08
#byte    porte = 0x09

/* ************************************
 *                      ENTRADAS                      *
 ************************************/
#bit      Botao7 = portb.1
#bit      BotaoDesLiga = portb.2
#bit      Botao9 = portb.3

/* ************************************
 *                      SAÍDAS                      *
 ************************************/
#bit      HabLed = porta.4
#bit      HabBotao = portb.6

#bit      Led0 = portd.0
#byte     Display = portd
```

```
/* ************************************
 *              Configurações do Microcontrolador          *
 ************************************ */

void main ()
        {
        // configura CONFIG
        setup_counters(RTCC_INTERNAL, WDT_2304MS);


        // configura os TRIS
        set_tris_a(0b00100000);
        set_tris_b(0b00001111);
        set_tris_c(0b10011001);
        set_tris_d(0b00000000);
        set_tris_e(0b00000000);

        // inicializa os ports
        porta=0x00;                     // limpa porta
        portb=0x00;                     // limpa portb
        portc=0x00;                     // limpa portc
        portd=0x00;                     // limpa portd
        porte=0x00;                     // limpa porte

        HabLed =1;
        HabBotao=1;//rb7
/* ************************************
 *                  Loop principal                  *
 ************************************ */
        while(TRUE)
                {
                RESTART_WDT();
                if(Botao9)              // testa botão
                        portd = 0b01101111;
                if(Botao7)              // testa botão
                        portd = 0b00000111;
                if(BotaoDesLiga)
                        portd = 0b00000000;             // caso contrário, led = 0

            }
                                // FIM DO PROGRAMA
        }
/* ************************************
 *                Fim do Programa                          *
 ************************************ */
```

6. AS VARIÁVEIS NO COMPILADOR CCS

6.1. O que são Variáveis?

Para você poder manipular dados dos mais diversos tipos, é necessário poder armazená-los na memória e poder referenciá-los quando for preciso. É para isso que existem as **variáveis**, que nada mais são do que um espaço reservado na memória, e que possuem um nome para facilitar a referência. As variáveis podem ser de qualquer tipo visto nesta aula: int, char, float, double, etc.

Como o próprio nome diz, as variáveis podem ter o seu conteúdo alterado durante a execução do programa, ou seja, o programador tem a liberdade de atribuir valores ao decorrer da execução.

6.2. Tipos de variáveis

O C tem 5 tipos básicos: **char**, **int**, **float**, **void, double**. Destes não todos os tipos fazem parte do compilador CCS. O **double** é o ponto flutuante duplo e pode ser visto como um ponto flutuante com muito mais precisão. O void é o tipo vazio, ou um "tipo sem tipo". A aplicação deste "tipo" será vista posteriormente. Na seguinte tabela observaremos os tipos de dados do compilador CCS.

Type-Specifier	Num bits	Coment
int1	1	Define a 1 bit number
int8	8	Defines an 8 bit number
int16	16	Defines a 16 bit number
int32	32	Defines a 32 bit number
char	8	Defines a 8 bit character
float	32	Defines a 32 bit floating point number
short	1	By default the same as int1
Int	8	By default the same as int8
long	16	By default the same as int16
byte	8	
bit	1	
void	0	Indicates no specific type

6.3. OS MODIFICADORES

Para cada um dos tipos de variáveis existem os modificadores de tipo. Os modificadores de tipo do C são quatro: **signed**, **unsigned**, **long** e **short**. Ao **float** não se pode aplicar nenhum.

Os quatro modificadores podem ser aplicados a inteiros. A intenção é que **short** e **long** devam prover tamanhos diferentes de números inteiros, onde isto for prático. Inteiros menores (**short**) ou maiores (**long**). Assim, se especificamos uma variável como sendo do tipo **short int**, este será uma versão reduzida do tipo int, o

que no caso do compilador CCS cria uma variável de apenas um bit. Uma especificação long int criará uma variável de 16 bits

O modificador **unsigned** serve para especificar variáveis sem sinal. Um **unsigned int** será um inteiro que assumirá apenas valores positivos. A representação de números negativos é feita tomando o bit MSB (*Most Significant Bit* ou bit mais significativo) da variável para representar o sinal: bit MSB =1, sinal negativo, bit MSB = 0, sinal positivo. Note que devido ao fato de utilizar um bit para representação do sinal, a magnitude absoluta de representação do tipo modificado será metade da magnitude do tipo não modificado.

Assim, um tipo de dados signed int pode representar valores entre -128 e +127, em vez de 0 a 255.

Todos os tipos, exceto o *Float*, por defeito são *unsigned*; porem pode estar precedido por *unsigned* ou *signed*. Na seguinte tabela mostramos alguns exemplos de combinações possíveis dos tipos de dados do compilador CCS.

Tipo	Num de bits	Intervalo	
		Inicio	Fim
char	8	0	255
unsigned char	8	0	255
signed char	8	-128	127
int, int8, byte	8	0	255
unsigned int, unsignet byte	8	0	255
signed int, signed int	8	-128	127
long int	16	0	65.535
signed long int	16	-32.768	32.767
unsigned long int	16	0	65.535
Int32, unsigned int32	32	0	4.294.967.295
Signed int32	32	-2,147.483.648	2,147.483.647
float	32	3,4E-38	3.4E+38

6.4. Declaração de Variáveis

Todas as variáveis em C devem ser declaradas antes de serem usadas. A forma geral de declaração de uma variável é a seguinte:

tipo nome_da_variavel;

Exemplos:

Uma variável de cada vez:

```
int id;
```

```
float aux;
unsigned int i;
char carac;
```

Ou várias do mesmo tipo em uma única linha:

```
int id, obj, n, t;
char c1, c2, c3;
```

Onde *tipo* pode ser qualquer tipo válido da linguagem C, e *nome_da_variavel* deve ser um nome dado pelo programador, sendo o primeiro caractere uma letra ou um "_", e os demais, letras, números ou "_".

```
num          _carac
peso         Id_obj
aluno_1              AUX
```

Lembrando que C diferencia maiúsculas de minúsculas.

Não são nomes válidos de variáveis:

```
1num         -idade
$aux         id@al
```

6.5. Inicializando Variáveis

Inicializar significa atribuir um valor inicial a uma variável. Em C, podemos inicializar variáveis na declaração:

```
int n=12;
```

Neste exemplo, estamos declarando a variável inteira *n* e atribuindo o valor 12 à ela. O sinal de igual (=) em C é o operador de atribuição. Ele serve para colocar o valor (do lado direito) na variável (lado esquerdo).

Podemos também inicializar uma variável no corpo do programa, após sua declaração:

```
int n;
n=12;
```

Pode-se atribuir também variáveis a variáveis:

```
num=i;
```

Neste caso, a variável *num* está recebendo o valor da variável *i*.

6.6. Variáveis Locais e Globais

As variáveis podem ser locais ou globais. As **variáveis locais** são aquelas declaradas dentro de uma função, sendo somente "vistas" dentro desta função. Se você tentar acessar uma variável local de fora da função onde ela está, acontecerá um erro. Após a execução desta função, as variáveis locais ali contidas são destruídas. Na aula sobre **funções** estudaremos mais sobre variáveis locais, com exemplos.

As **variáveis globais** são aquelas declaradas fora de qualquer função, até da função main(). São reconhecidas pelo programa inteiro e podem ser usadas em qualquer lugar do código, mesmo dentro de funções. Seus valores ficam guardados durante toda a execução do programa. Veja um exemplo do uso de variáveis globais:

```
#include *****

int num;    /* num é uma variável global */

int main()
{
        int contador=0;  //contador é uma variável local
        num= 10;
        printf("%d", n);
        return 0;
}
```

Assim como as variáveis locais, veremos as variáveis globais com mais detalhes na aula sobre funções.

6.7. Constantes

Constantes em C são valores fixos, que não mudam durante a execução. Elas podem ser de qualquer tipo básico: inteiras, ponto flutuante, caractere, string.

De maneira geral, qualquer tipo de dado pode ser utilizado para definir uma constante, cabendo apenas ao programador representar o valor dela adequadamente.

```
const valor1 = 10
const valor1 = -3
const valor1 = 13.24
const valor1 = ‘c’
const valor1 = “CEFETES-Linhares”
```

As constantes são também utilizadas para impedir que uma função altere um parâmetro passado a ela.

Bem, nesta aula você aprendeu sobre os tipos de dados que a linguagem C pode manipular. Também viu a forma de armazenar temporariamente estes dados, com o

uso de variáveis, e como expressar constantes de vários tipos em C. Na próxima aula estudaremos os operadores da linguagem C (aritméticos, relacionais, lógicos, etc.). Não percam!

Exercícios 6.1:

1. Qual a faixa numérica de uma variável **int**? E de um **unsigned int**?

2. Qual a diferença de um **signed int** para um **int**?

3. Por que a declaração de uma variável como **unsigned** é redundante?

4. O que são variáveis e qual é sua utilidade?

5. Qual a diferença entre as **variáveis locais** e **globais**?

8. O que são constantes? Dê exemplos.

9. Assinale a alternativa que possui um nome de variável CORRETO:

- $num
- -temp
- _carac_
- 3aux

7. OPERADORES EM C

Já aprendemos sobre a estrutura básica de um programa em C, sobre os tipos de dados que C pode manipular, sobre o que são variáveis e constantes. Na aula de hoje você aprenderá a criar expressões em C com o uso de **operadores**, sejam eles aritméticos, lógicos, relacionais, etc. Mostraremos vários exemplos e finalizaremos a aula com exercícios.

Então, pé na tábua!

7.1. O Operador de Atribuição

Na aula sobre variáveis já foi falado sobre o operador de atribuição (o símbolo de igual "="). O que ele faz é colocar o valor de uma expressão (do lado direito) em uma variável (do lado esquerdo). Uma expressão neste caso pode ser um valor constante, uma variável ou uma expressão matemática mesmo.

É um operador binário, ou seja, trabalha com dois operandos. Exemplos:

Atribuição de uma constante a uma variável:

```
n= 10;
ch= 'a';
fp= 2.51;
```

Atribuição do valor de uma variável a outra variável:

```
n= num;
```

Atribuição do valor de uma expressão a uma variável:

```
n= (5+2)/4;
```

Atribuições múltiplas:

```
x = y = z = 20;
```

Em uma atribuição, primeiro é processado o lado direito. Depois de processado, então, o valor é atribuído a variável.

Como você viu no último exemplo acima, C também permite atribuições múltiplas (como x = y = z = 20;). Neste caso, todas as variáveis da atribuição (x, y e z) recebem o valor mais à direita (20).

7.2. Os Operadores Aritméticos

Estes são, de longe, os mais usados. Os operadores aritméticos em C trabalham praticamente da mesma forma que em outras linguagens. São os operadores **+**

(adição), **-** (subtração), * (multiplicação), **/** (divisão) e **%** (módulo ou resto da divisão inteira), todos estes binários (de dois operandos). Temos também o **- unário**, que muda o sinal de uma variável ou expressão para negativo. Veja a tabela a seguir:

Operador	**Descrição**	**Exemplo**
- unário	Inverte o sinal de uma expressão	-10, -n, -(5*3+8)
*	Multiplicação	3*5, num*i
/	Divisão	2/6, n/(2+5)
%	Módulo da divisão inteira (resto)	5%2, n%k
+	Adição	8+10, exp+num
-	Subtração	3-6, n-p

A precedência dos operadores aritméticos é a seguinte:

Mais alta

- unário

* / %

+ -

Mais baixa

Uma expressão deste tipo:

```
5 + 2 * 3 - 8 / 4
```

É avaliada assim: primeiro a multiplicação (2*3), depois a divisão (8/4). Os resultados obtidos destas duas operações são utilizados para resolver as duas últimas operações:

```
5 + 6 - 2
```

adição e subtração. Igualzinho à matemática aprendida no primário...

Tudo isso porque as operações de multiplicação e divisão têm maior precedência e estas são resolvidas primeiro em uma expressão. Para mudar a ordem de operação, devem-se usar parênteses. Deve-se tomar cuidado ao construir expressões, pois a falta de parênteses pode causar erros no resultado.

O Operador **%** é equivalente ao **mod** em Pascal, e é útil em várias situações. Ele dá como resultado o resto da divisão inteira de dois operandos. Assim, fica fácil, por exemplo, saber se um número é múltiplo de outro:

```
if ((num%3)==0)   /* se o resto da divisão entre num e 3 for igual a 0 ... */
        printf("Múltiplo de 3\n");
```

Este é apenas um de vários problemas que podem ser resolvidos com o uso do operador **%**.

7.3. Operadores Relacionais e Lógicos

Os operadores relacionais e lógicos são usados em testes e comparações, principalmente nos comandos de controle e nos laços.

Para entender melhor esses operadores, temos que entender o conceito de **verdadeiro** e **falso**. Em C, **verdadeiro é qualquer valor diferente de zero**, e **falso é zero**. As expressões que usam operadores relacionais e lógicos retornam 0 para falso e 1 para verdadeiro.

Os operadores relacionais são 6:

Operador	Ação
<	Menor que
<=	Menor que ou igual
>	Maior que
>=	Maior que ou igual
==	Igual
!=	Diferente

Veja um exemplo do uso de operadores relacionais:

```
 #include <stdio.h>                                    /* Inclusão de stdio.h (necessário para printf e scanf) */

 int main()
 {
        int n;                                         /* Declaração de uma variável inteira */

        printf("Digite um número: ");
        scanf("%d", &n);                    /* Lê o número e armazena na variável n */
        if (n < 0)                          /* Se n for MENOR QUE 0... */
                printf("Número negativo\n");    /* ... escreve isto. */
        else                                /* Senão... */
                printf("Número positivo\n");    /* ... escreve isto. */
        return 0;                           /* Retorna 0 para o sistema (sucesso) */
 }
```

Tente fazer alguns testes com os outros operadores relacionais. Sei lá, seja criativo! Você é o programador... :)

Os operadores lógicos são 3:

Operador	Ação	Formato da expressão
&&	and (**e** lógico)	p && q
||	or (**ou** lógico)	p || q
!	not (**não** lógico)	!p

Veja um exemplo, só do bloco *if*:

```
  if ((n > 0) && (n < 100))                    /* se n for maior que 0 E n for
menor que 100... */
          printf("Número positivo menor que 100\n"); /* ... imprime isto */
```

Outro exemplo:

```
  if ((n == 0) || (n == 1))                     /* se n for IGUAL a 0 OU n for igual a 1
... */
          printf("zero ou um\n");              /* ... imprime isto. */
```

A tabela seguinte mostra a precedência dos operadores relacionais e lógicos:

```
  Maior
  !
  >, >=, <, <=
  ==, !=
  &&
  ||
  Menor
```

Os parênteses também podem ser usados para mudar a ordem de avaliação das expressões, como no caso das expressões aritméticas.

7.4. Operadores de Incremento e Decremento

A linguagem C possui dois operadores que geralmente não são encontrados em outras linguagens, mas que facilita bastante a codificação: os operadores de incremento(++) e decremento(--).

O operador **++** soma 1 ao seu operando, similar a operação de *variavel= variavel+1;*, mas muito mais resumido.

O operador **--** subtrai 1 de seu operando, também similar a *variavel= variavel-1;*.

Estes operadores são unários, e podem ser usados antes da variável:

```
  ++n;
```

ou depois da variável:

```
n++;
```

A diferença é que, se o operador precede o operando (++n), o incremento ou decremento é realizado antes do valor da variável ser usado. E se o operador vem depois do operando (n++), o valor da variável poderá ser usado antes de acontecer a operação de incremento ou decremento. Veja estes dois exemplos:

Exemplo 1:

```
n= 5;
p= ++n;
printf("%d ",p);  /* imprime na tela: 6 */
```

Exemplo 2:

```
n= 5;
p= n++;
printf("%d ",p);  /* imprime na tela: 5 */
```

No exemplo 1, a variável n é incrementada de 1 ANTES de seu valor ser atribuído a p. No exemplo 2, o valor de n é atribuído a p antes de acontecer a operação de incremento. Essa é a diferença de se colocar esses operadores antes ou depois da variável.

7.5. Operadores Aritméticos de Atribuição

Algumas operações de atribuição podem ser feitas de forma resumida. Por exemplo, a atribuição:

```
x= x+10;
```

pode ser escrita:

```
x+=10;
```

A forma geral desse tipo de atribuição é:

```
variável [operador]= [expressão];
```

que é igual a:

```
variável= variável [operador] [expressão]
```

Veja essa tabela com exemplos:

Forma longa	Forma resumida
x= x+10	x+=10
x= x-10	x-=10
x= x*10	x*=10
x= x/10	x/=10
x= x%10	x%=10

Se familiarize com essa notação, pois é um estilo largamente adotado pelos profissionais que trabalham com C.

7.6. Operadores Bit a Bit

A linguagem C, ao contrário de outras linguagens, suporta um amplo conjunto de operadores bit a bit. Como C precisava substituir a linguagem Assembly na maioria das tarefas, era de vital importância que pudesse realizar essas operações.

As operações bit a bit manipulam diretamente os bits de um byte ou uma palavra, e essas operações só podem ser feitas em variáveis **int** e **char** (e suas variantes). Os tipos **void**, **float**, **double** e **long double** NÃO podem ser usados.

Esses operadores são usados com mais freqüência em drivers de dispositivo, como rotinas de impressoras, modems, operações com arquivos em disco, rotinas da porta serial e paralela, etc.

Veja os operadores bit a bit:

Operador	Ação
&	and ("e" lógico)
|	or ("ou" lógico)
^	exclusive or ("ou exclusivo")
~	complemento de 1
>>	deslocamento à direita
<<	deslocamento à esquerda

Obs: Não confunda os operadores bit a bit com os operadores lógicos.

Para saber como se comportam os operadores bit a bit, segue abaixo uma tabela verdade dos operadores, com exceção dos operadores de deslocamento:

p	q	p&q	p|q	p^q	~p
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

O operador **&** compara os dois operandos bit a bit e cada bit do resultado é 1 somente quando os dois bits do operando for 1. Caso contrário, 0.

O bits do resultado de uma operação **|** bit a bit só será 1 quando um dos dois bits, ou os dois, forem 1. Na operação **^** só será 1 quando os bits dos operandos forem diferentes. Se forem iguais (0^0 ou 1^1) será 0.

O operador de complemento (~) é um operador unário que inverte os bits do operando. 1 vira 0 e 0 vira 1.

Vejamos um exemplo mais claro. Digamos que as variáveis inteiras x e y recebem os valores 8 e 9, respectivamente:

```
x= 8;    /* 8 em binário é 1000 */
y= 9;    /* 9 em binário é 1001 */
```

Vamos fazer algumas operações bit a bit nesses números:

```
x & y:
        1000
        1001
        ----
        1000     => O resultado desta operação é 8 (1000).

x | y:
        1000
        1001
        ----
        1001     => O resultado desta operação é 9 (1001).

x ^ y:
        1000
        1001
        ----
        0001     => O resultado desta operação é 1.

~x:     1000
        ----
```

```
        0111    => O resultado desta operação é 7 (111).

~y:     1001
        ----
        0110    => O resultado desta operação é 6 (110).
```

Aí você me pergunta: e os operadores de deslocamento? Esses operadores deslocam **n** bits para a esquerda ou direita:

```
x>>1   0100    => resultado: 4
x>>2   0010    => resultado: 2

y<<1   10010   => resultado: 18
x<<1   10000   => resultado: 16
```

Você deve ter notado que deslocar um bit à direita é o mesmo que dividir o número por 2. Deslocar 2 bits à direita é o mesmo que dividir o número por 4, e assim por diante.

Deslocar um bit à esquerda, porém, é o mesmo que multiplicar o número por 2. Deslocar 2 bits à esquerda significa multiplicar por 4, e assim por diante.

Em um determinado ponto do curso, criaremos uma calculadora bit a bit, onde vocês poderão estudar melhor essas operações.

7.7. Interface com os interruptores

Provavelmente este tema foge um pouco ao tema de operadores, mas é importante compreender as formas de poder ligar um interruptor no PIC. Este conteúdo foi localizado nesta parte do texto com fins didáticos para poder complementar os conhecimentos necessários para desenvolver o próximo exemplo.

A figura seguinte mostra dois circuitos possíveis para ligar um interruptor de pressão ao microcontrolador. Na figura 1 a) o pino de E/S é ligado a +5V através de uma resistência *pull-up* de valor elevado, ficando a nível lógico 1. Quando o interruptor é fechado, o pino é ligado à massa através de uma resistência menor, passando a nível lógico 0. O circuito da figura 1 b) tem um princípio de funcionamento simétrico ao que acabamos de descrever. Inicialmente o pino de E/S está ligado à massa através de uma resistência *pull-down* de valor elevado, ficando a nível lógico 0. Quando o interruptor é fechado, o pino é ligado a +5V através de uma resistência menor, passando a nível lógico 1.

Ambos os circuitos funcionam igualmente bem tanto para interruptores normalmente abertos, como para interruptores normalmente fechados.

Notar que é indispensável a existência de resistências *pull-up* e *pull-down*. Imaginemos o caso da figura 1 (a). Se o interruptor estiver fechado, o pino de E/S é ligado à massa e não é necessário nada mais. Mas quando o interruptor estiver aberto, se não houver resistência *pull-up*, a entrada digital está “ao ar”, o que para

portas de alta impedância é especialmente problemático, uma vez que o valor lógico da porta poderá oscilar continuamente entre nível lógico alto e baixo.

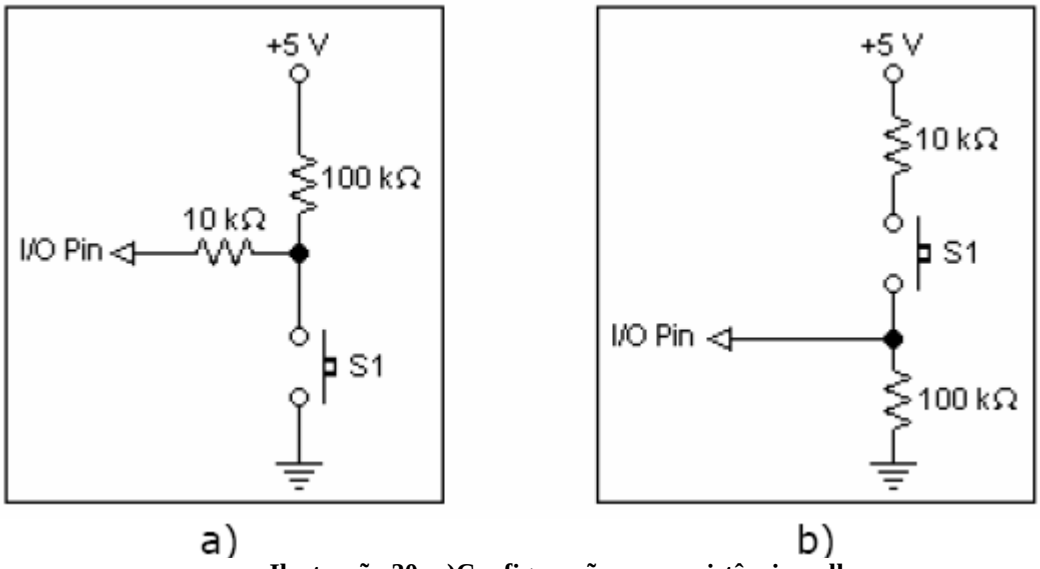
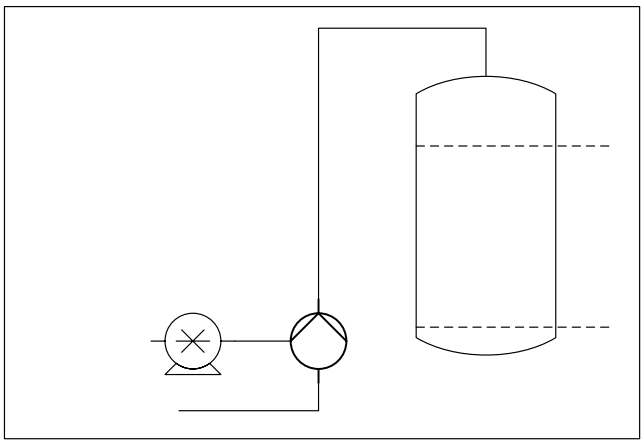


Ilustração 30: a)Configuração com resistência pull-up;
b) Configuração com resistência pull-down

Estudem bastante e façam os exercícios e exemplos.

Exemplos:

1. Resolver o seguinte problema: Fazer o Projeto, para o controle da Bomba B1, sendo que o nível sempre estará entre S1 e S2 (sensores bóia). Quando o nível atinge o motor S2 o seu contato fecha, e quando o nível atinge S1 este também fecha, sendo que S2, já esta fechado porque o nível esta sobre S2.



Para poder ter uma noção, de como poderia ser o circuito eletrônico, é apresentado o esquemático deste. O motor não pode ser ligado diretamente pelo PIC, por enquanto consideraremos que um Led (LMOTOR)representando o Motor.

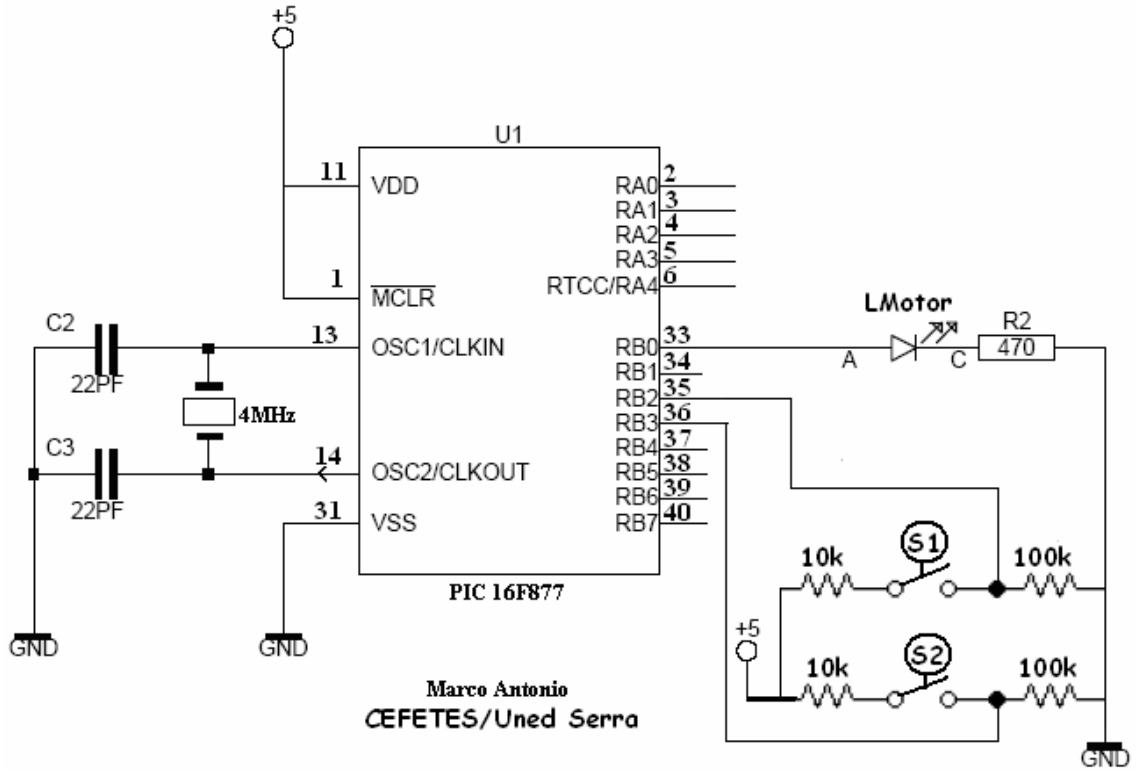


Ilustração 31: Circuito elétrico do exemplo 7.1

Simular o seguinte programa escrito. Depois descrever o seu funcionamento.

```
/*******************************************************
*                      CEFETES
* Prof: Marco Antonio
* Exemplo 2: Controle de Nível num tanque de água
* Materia: MICROCONTROLADORES
* Data: Julho 2006 /Linhares
*******************************************************/
#include          <16f877A.h>
#use     delay(clock=4000000, RESTART_WDT)
#fuses  xt,nowdt,noprotect,put,brownout,nolvp,nocpd,nowrt
#use     fast_io(a)
#use     fast_io(b)
#use     fast_io(c)
#use     fast_io(d)
#use     fast_io(e)

#byte    porta = 0x05
#byte    portb = 0x06
#byte    portc = 0x07
#byte    portd = 0x08
#byte    porte = 0x09

#bit      LMotor = portb.0
#bit      S1 = portb.2
#bit      S2 = portb.3


void main ()
{
          set_tris_a(0b11111111);
          set_tris_b(0b11110110);
          set_tris_c(0b11111111);
          set_tris_d(0b11111111);
          set_tris_e(0b00000111);

          porta=0x00;
          portb=0x00;
          portc=0x00;
          portd=0x00;
          porte=0x00;

          while(TRUE)
          {

                    if((!S1)&&(!S2))
                              LMotor =1;
                    if((S1)&&(S2))
                              LMotor = 0;
          }
}
```

Exercícios 7.1:

1) Para que serve o operador de atribuição?

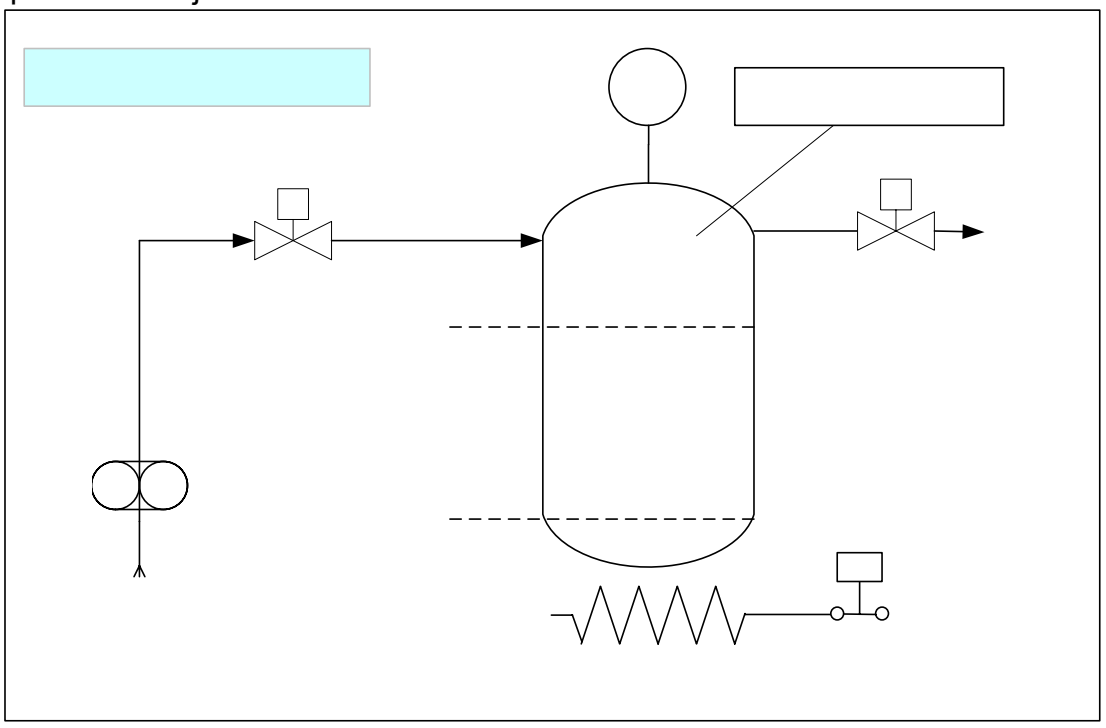
2) Assinale a expressão aritmética com resultado CORRETO:

a) 2+3*5 = 30
b) 5*2/2 = 5
c) 4+6-8/2 = 6
d) (-3)*4 = 12

3) As expressões a seguir têm como resultado VERDADEIRO (1) ou FALSO (0)?

a) 1 > 2
b) 2 >= 1
c) 3 != 3
d) 'c' == 'c'

4) Fazer um projeto para que a pressão do vapor sempre permaneça entre uma faixa específica (histerese), sabendo que a histerese do pressostato é calibrada no instrumento. Os níveis do tanque sempre devem de estar entre S1 e S2, sabendo que a bomba 1 tem uma partida simples. V2 só abrira quando a pressão esteja entre os níveis de histereses determinado.



.

Ilustração 32: Exercício 7.4, controle de pressão

5) Um misturador permite a seleção entre 2 materiais através de um seletor S2. Na posição 1 (S2 = 0), o material A passa para o tanque de mistura se o botão S1 esta atuado simultaneamente. Com o seletor S2 em posição 2 (S2=1) e S1 atuado o material B passa para o tanque de mistura. As Válvulas solenóides VA e VB permitem a passagem dos materiais. Faça o projeto e simule.

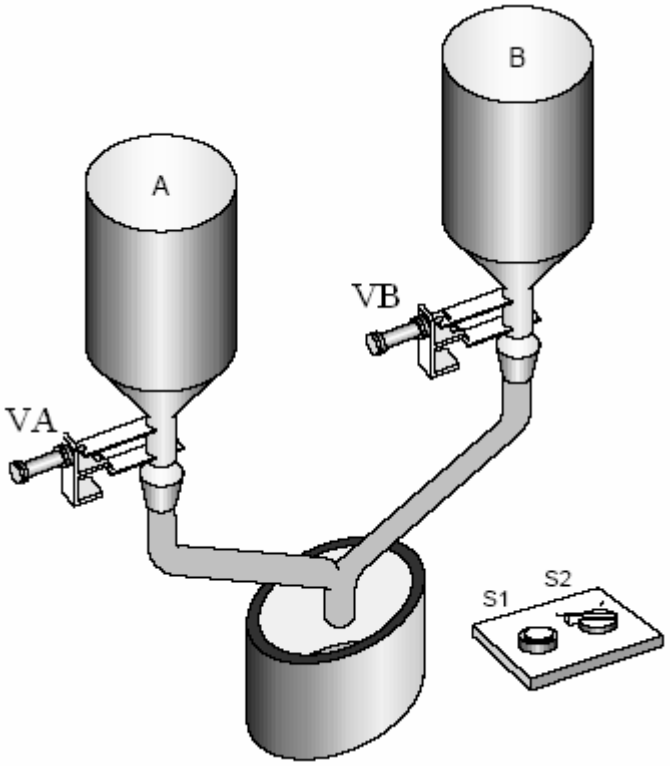


Ilustração 33: Exercício 7.5, misturador, combinacionais

8. TEMPORIZAÇÃO NO PIC

Los “temporizadores” los encontramos en muchos lugares de nuestra vida diaria, generalmente integrados en aparatos domésticos. Por ejemplo en los hornos de microondas para controlar el tiempo de calentamiento de nuestros alimentos, o en las lavadoras de ropa para seleccionar el tiempo lavado.

También a escala industrial las aplicaciones de los temporizadores son múltiples. Los hay para controlar el tiempo de arranque de algún proceso, se usan en máquinas herramientas, en dosificadores, fotografía, etc. La lista es interminable.

Neste capítulo aprenderemos a usar temporizadores, são duas formas básicas: usando a função *delay* e interrupções. A função *delay* não permite a execução de outras funções no decorrer do tempo, já usando interrupções isto não acontece. Desta forma, as interrupções têm muitas vantagens sobre a função *Delay*.

8.1. A função *Delay:*

Esta função criará código para executar uma demora do tempo especificado. O tempo é especificado em milisegundos. Esta função trabalha executando um número preciso de instruções para causar a demora pedida.

O tempo de demora pode ser mais longo do que o pedido se uma interrupção for chamada durante a espera do tempo. O tempo gastado na interrupção não conta para o tempo de demora.

Sintaxes da função: **delay_ms(time);**

Parâmetros: **time –** se for variável de 0 a 255 se for constante de 0 a 65635.

Exemplo 8.1:

È conveniente observar que neste exemplo o *watch dog time*r esta ativado, neste caso será necessário resetar o WDT para que o PIC não seja resetado e comece a ser executada a primeira linha do programa. Para isto é usada a função **RESTART_WDT()**.

Alem da função Delay, existem outras funções que ainda não foram vistas. Trata-se das funções **output_high(PIN_B0)** e **output_low(PIN_B0**).

A função **output_high(PIN_B0),** é utilizada para setar (ou seja, colocar em nível lógico ‘1’) um pino do microcontrolador. Isto significa que o pino RB0 (da porta B) será setado.

Note que “PIN_B0” é um símbolo predefinido para especificar o pino RB0. Este símbolo esta localizado no arquivo de cabeçalho do processador “16f877A.h”.

Da mesma forma a função **output_high(PIN_B0),** é utilizada para resetar (ou seja, colocar em nível lógico '0') um pino do microcontrolador. Isto significa que o pino RB0 (da porta B) será resetado.

```
/*******************************************************
*                    CEABRA
* Prof: Marco Antonio
* Exemplo 1: Pisca – Pisca  (Estrutura básica de um programa)
* Matéria: Microcontroladores
* Data: Julho 2006
*******************************************************/
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *            DEFINIÇÃO DAS VARIÁVEIS INTERNAS DO PIC            *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
#include          <16f877A.h>     // microcontrolador utilizado

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                 Configurações para gravação                 *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#fuses  xt,wdt,noprotect,put,brownout,nolvp,nocpd,nowrt          // configuração dos fusíve


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *            Definições para uso de Rotinas de Delay                    *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#use     delay(clock=4000000, RESTART_WDT)
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *            Definição e inicialização das variáveis            *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
//Neste bloco estão definidas as variáveis globais do programa.
//Este programa não utiliza nenhuma variável de usuário

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                   Constantes internas                       *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
//A definição de constantes facilita a programação e a manutenção.
//Este programa não utiliza nenhuma constante de usuário

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *               Declaração dos flags de software              *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
//A definição de flags ajuda na programação e economiza memória RAM.
//Este programa não utiliza nenhum flag de usuário

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *            Definição e inicialização dos port's             *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#use     fast_io(a)
#use     fast_io(b)
#use     fast_io(c)
#use     fast_io(d)
#use     fast_io(e)

#byte    porta = 0x05
#byte    portb = 0x06
#byte    portc = 0x07
```

```
#byte   portd = 0x08
#byte   porte = 0x09

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                         ENTRADAS                            *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
// As entradas devem ser associadas a nomes para facilitar a programação e
//futuras alterações do hardware.

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                         SAÍDAS                          *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
// AS SAÍDAS DEVEM SER ASSOCIADAS A NOMES PARA FACILITAR A PROGRAMAÇÃO E
//FUTURAS ALTERAÇÕES DO HARDWARE.


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                 Configurações do Microcontrolador             *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

void main ()
        {
        // configura CONFIG
        setup_counters(RTCC_INTERNAL, WDT_2304MS);


        // configura os TRIS
        set_tris_a(0b11111111);             // configuração dos pinos de I/O
        set_tris_b(0b11111000);
        set_tris_c(0b11111111);
        set_tris_d(0b11111111);
        set_tris_e(0b00000111);

        // inicializa os ports
        porta=0x00;                  // limpa porta
        portb=0x00;                  // limpa portb
        portc=0x00;                  // limpa portc
        portd=0x00;                  // limpa portd
        porte=0x00;                  // limpa porte

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                        Loop principal                    *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
        while(TRUE)
                {
                RESTART_WDT();

    output_high(PIN_B0);
    delay_ms(1000);
    output_low(PIN_B0);
    delay_ms(1000);
    output_high(PIN_B1);
    delay_ms(1000);
    output_low(PIN_B1);
    delay_ms(1000);
             }
                                        // FIM DO PROGRAMA
        }
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                  Fim do Programa                              *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

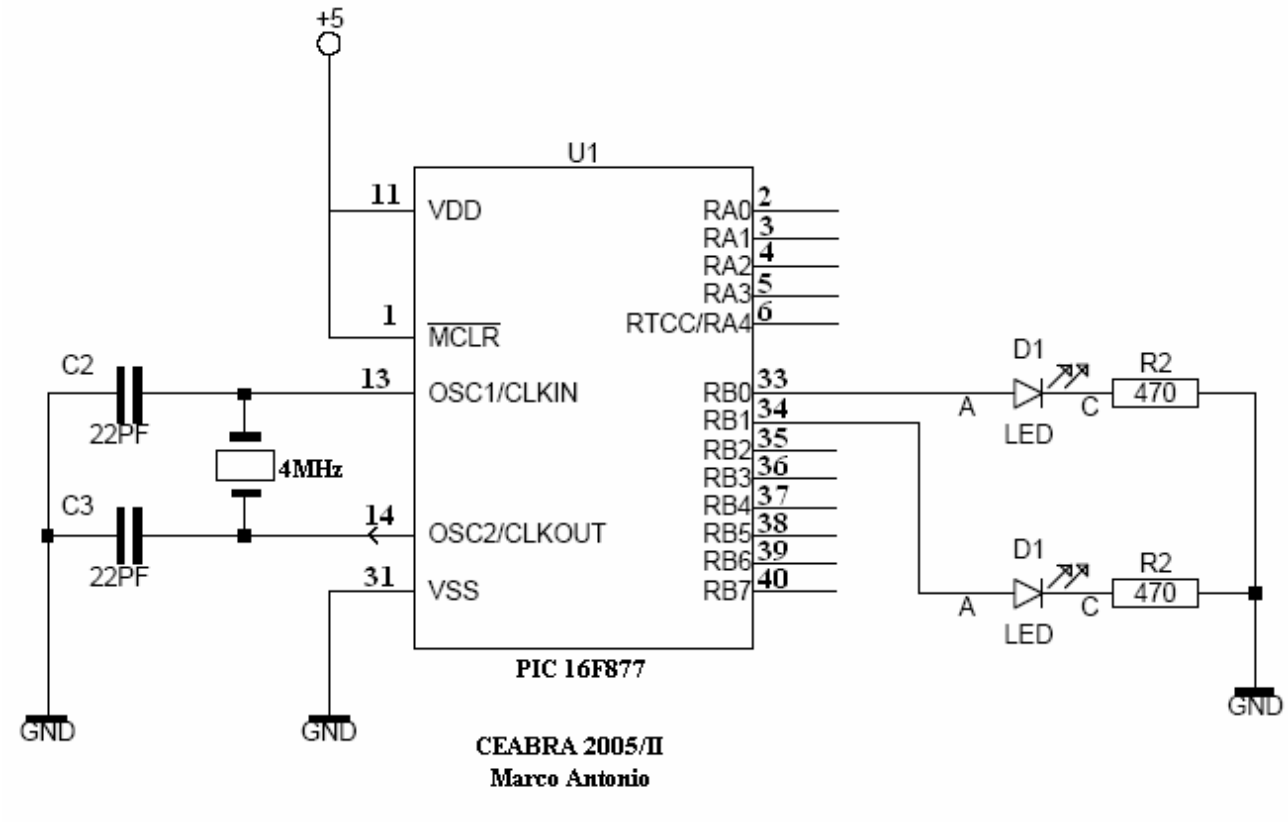


Ilustração 34: Circuito para o Exemplo 2

8.2. Interrupção Temporizador:

As interrupções são mecanismos que o microcontrolador possui e que torna possível responder a alguns acontecimentos no momento em que eles ocorrem, qualquer que seja a tarefa que o microcontrolador esteja a executar no momento. Esta é uma parte muito importante, porque fornece a ligação entre um microcontrolador e o mundo real que nos rodeia. Geralmente, cada interrupção muda a direção de execução do programa, suspendendo a sua execução, enquanto o microcontrolador corre um subprograma que é a rotina de atendimento de interrupção. Depois de este subprograma ter sido executado, o microcontrolador continua com o programa principal, a partir do local em que o tinha abandonado.

8.3. O temporizador TIMER 0

Os temporizadores são normalmente as partes mais complicadas de um microcontrolador, assim, é necessário gastar mais tempo a explicá-los. Servindo-nos deles, é possível relacionar uma dimensão real que é o tempo, com uma variável que representa o estado de um temporizador dentro de um microcontrolador. Fisicamente, o temporizador é um registro cujo valor está continuamente a ser incrementado até 255, chegado a este número, ele começa outra vez de novo: 0, 1, 2, 3, 4, ...,255, 0,1, 2, 3,..., etc.

Este registro é chamado de TMR0 que é incrementado a cada ciclo de máquina (sem pré-escala: Clock/4). Se ocorrer uma escrita em TMR0, o incremento será inibido pelos dois ciclos de máquina seguintes. A interrupção do Timer 0 é gerada quando o conteúdo do registrador TMR0 passar de FFH para 00H.

Exemplo 8.2:

Vejamos neste exemplo como configurar o módulo timer 0 para que uma saída pisque a cada segundo. Neste caso o clock é de 4 MHz e é utilizado um prescaler de 32, então teremos uma freqüência de entrada no timer 0 de:

Fosc/(4*preEscala) = 4 000 000/(4*32) = 31250

Deste resultado podemos tirar uma conclusão: a cada vez que o registro TMR0 se incrementa em 1, passa na verdade um tempo de 1/31250 seg.

Agora a pergunta é a cada quanto tempo é chamada a função de interrupção? A resposta é simples como a função de interrupção é chamada depois do estouro do registrador TMR0 e esta estoura depois de 255 vezes o tempo para cada interrupção será de:

255*1/31250 = 0,00816 seg

O tempo desejado é de 1 seg, então:

#Interrup*0,00816 seg = 1 seg ➔ #Interrup = 122,549

Como o número de interrupções é inteiro, temos que encontrar outro médio de ter maior exatidão. Para isto, podemos iniciar o registro TMR0 com 13, de forma que a primeira interrupção aconteça depois de 125 ciclos (256-131) de TMR0, e para fazer isto repetitivo é reiniciado o registro TMR0 com 131 na propria sub-rotina de interrupção. Cada interrupção teria 125 ciclos, como cada ciclo tem um tempo de 1/31250 ➔ cada interrupção terá um tempo de 125*1/31250 = 0,004 seg.

O tempo desejado é de 1 seg, então:

#Interrup*0,004 seg = 1 seg ➔ #Interrup = 250

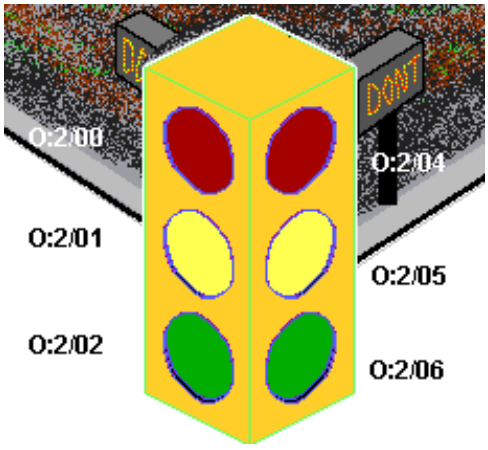
```
/*********************************************************
*                    CEFETES
* Prof: Marco Antonio
* Exemplo 8.2: Pisca - Pisca, usando interrupção Timer 0
* Materia: Eletrônica Digital
* Data: Setembro 2005
*********************************************************/
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *            DEFINIÇÃO DAS VARIÁVEIS INTERNAS DO PIC            *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
#include      <16f877A.h>     // microcontrolador utilizado

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                  Configurações para gravação               *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
```

```
#fuses        xt,wdt,noprotect,put,brownout,nolvp,nocpd,nowrt // configuração dos fusíveis

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *              Definições para uso de Rotinas de Delay                        *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#use          delay(clock=4000000, RESTART_WDT)
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *              Definição e inicialização das variáveis              *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
//Neste bloco estão definidas as variáveis globais do programa.
//Este programa não utiliza nenhuma variável de usuário

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                      Constantes internas                          *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
//A definição de constantes facilita a programação e a manutenção.
//Este programa não utiliza nenhuma constante de usuário

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                Declaração dos flags de software                  *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
//A definição de flags ajuda na programação e economiza memória RAM.
int conta;

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *               Definição e inicialização dos port's               *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#use          fast_io(a)
#use          fast_io(b)
#use          fast_io(c)
#use          fast_io(d)
#use          fast_io(e)

#byte         porta = 0x05
#byte         portb = 0x06
#byte         portc = 0x07
#byte         portd = 0x08
#byte         porte = 0x09

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                          ENTRADAS                                *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
// As entradas devem ser associadas a nomes para facilitar a programação e
//futuras alterações do hardware.

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                          SAÍDAS                              *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
// AS SAÍDAS DEVEM SER ASSOCIADAS A NOMES PARA FACILITAR A PROGRAMAÇÃO E
//FUTURAS ALTERAÇÕES DO HARDWARE.

#bit  led = portb.0               // Led correspondente ao botão 0

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                Configurações do Microcontrolador                  *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
#int_timer0
void trata_t0()
{
```

```
   //reinicia o timer 0 em 131 menos a contagem que já passou
   set_timer0 (131-get_timer0());
   conta++;
   //se já ocorreram 125 interrupções
   if (conta ==125)
   {
     conta=0;
     led = !led; //inverte o led
    }
}
void main ()
      {
      // configura CONFIG
      setup_counters(RTCC_INTERNAL, WDT_2304MS);
//configura o timer 0 para o clock interno e prescaler dividindo por 32
   setup_timer_0 (RTCC_INTERNAL | RTCC_DIV_32 );
   SET_TIMER0(131); //INICIA O TIMER 0 EM 131

   //HABILITA INTERRUPÇÕES
   ENABLE_INTERRUPTS (GLOBAL | INT_TIMER0);

      // configura os TRIS
      set_tris_a(0b11111111);          // configuração dos pinos de I/O
      set_tris_b(0b11111000);
      set_tris_c(0b11111111);
      set_tris_d(0b11111111);
      set_tris_e(0b00000111);

      // inicializa os ports
      porta=0x00;                      // limpa porta
      portb=0x00;                      // limpa portb
      portc=0x00;                      // limpa portc
      portd=0x00;                      // limpa portd
      porte=0x00;                      // limpa porte

/*************************************
 *                Loop principal     *
 *************************************/

      while(TRUE)
            {
            RESTART_WDT();
         }
                        // FIM DO PROGRAMA
      }
/*************************************
 *              Fim do Programa                    *
 *************************************/
```

Exercícios 8.1:

1) Qual é a diferença entre usar a função Delay e a interrupção do TMR0?
2) Fazer um semáforo, primeiro usando a função Delay e depois usando a interrupção TMR0. Simular e testar na bancada.
3) Utilizando seu conhecimento em temporizadores, automatizar o sinal de trânsito. Simular e testar na bancada.



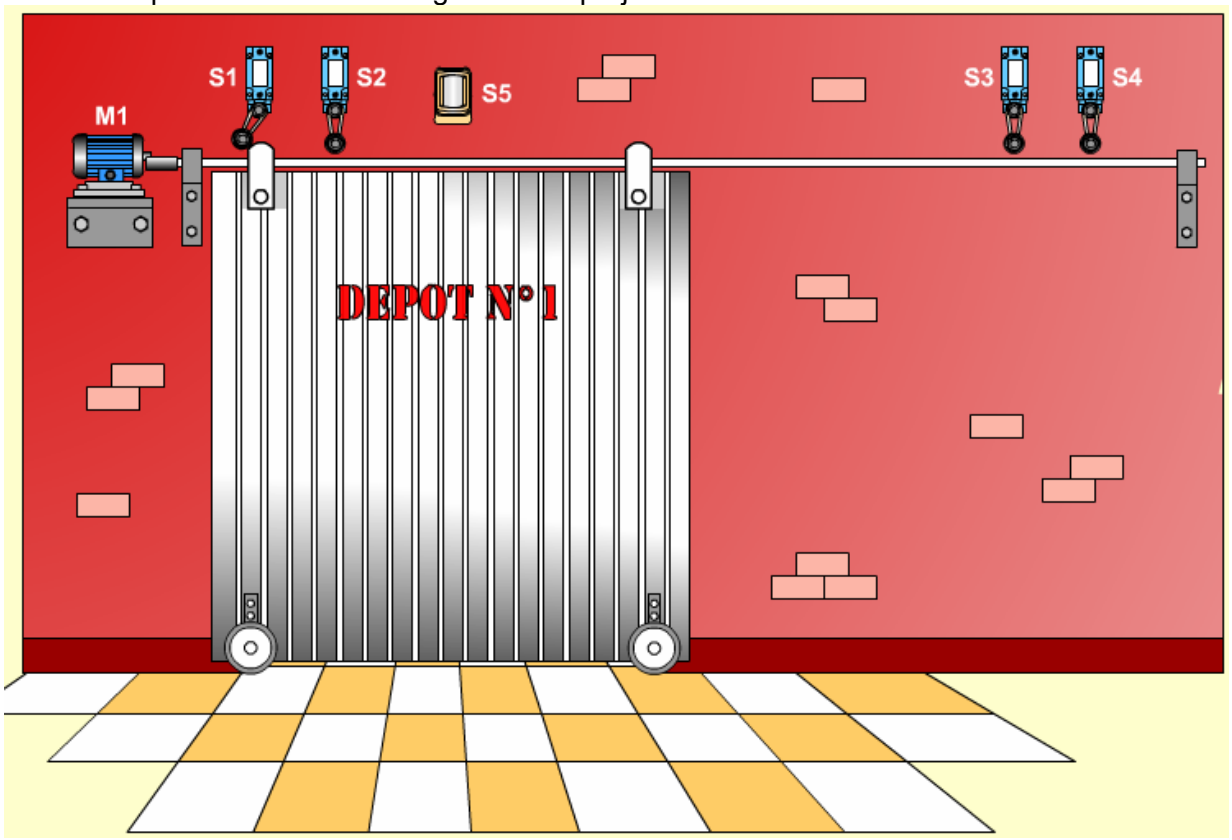
Red = O:2/00			Green = O:2/02	Amber = O:2/01	R
Green = O:2/06	Amber = O:2/05	Red = O:2/04			
8 sec.	4 sec.	1s	8 sec.	4 sec.	1s

<---------- Tempo em segundos ---------->

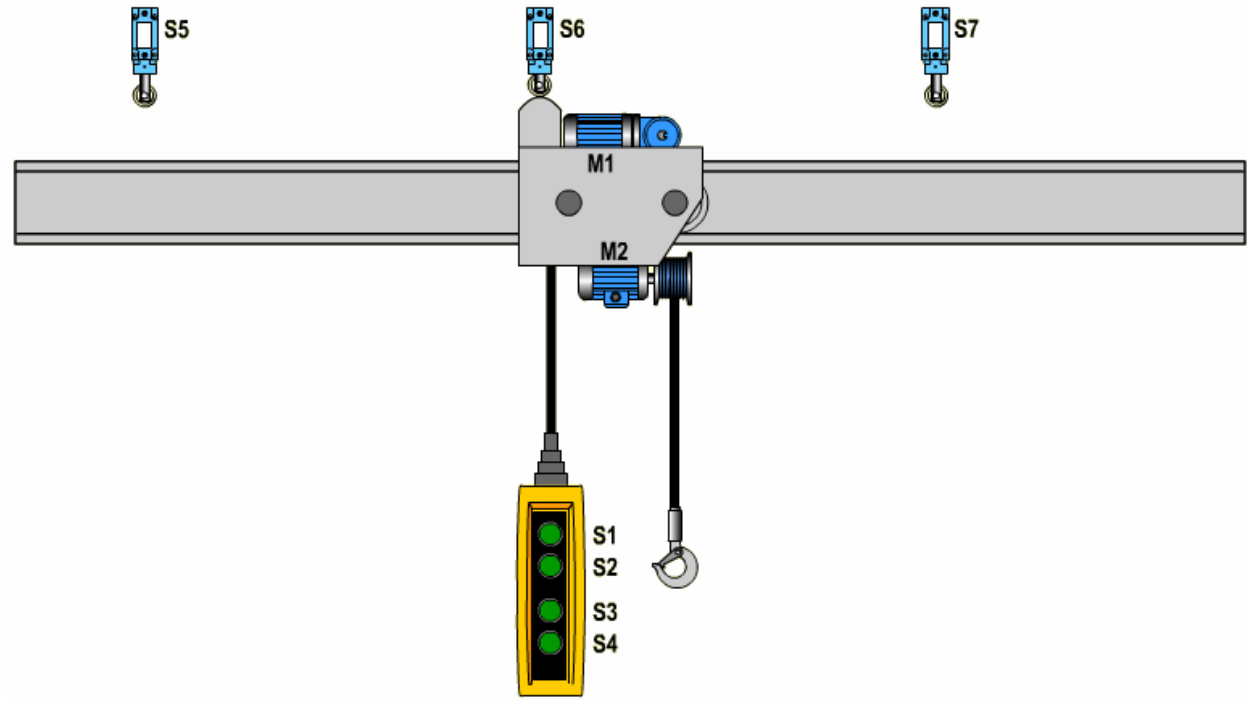
4) O proprietário de uma casa está cansado porque muitas vezes os ladrões entram no quintal e decidiu pedir ajuda aos alunos do CEFETES-Linhares. Pediu uma forma barata de fazer um alarme. O professor orientou aos alunos da seguinte forma:

   - Como o quintal era grande pediu para usar um diodo lazer para construir um sensor de barreira, só que o lazer não podia estar ligado constantemente, devia ficar ligado o 10% do período a uma freqüência de 1 KHz.
   - Para o receptor oriento trabalhar com uma resistência LDR que tem maior área sensora.
   - O alarme não poderia disparar imediatamente ao ser cortada a barreira, deverá esperar ½ segundo para ter certeza que se trata de um ladrão.
   - Boa sorte galera. Vocês terão que montar o projeto.

5) Quando o sensor S5 detecte a presença de uma pessoa a porta abrirá rapidamente até que o sensor S3 seja atingido, onde a velocidade diminui quando chegue a S4 este para, espera 15 segundos e o portão fecha. Chegando a S2 a velocidade diminui e em S1 o portão para. Considera a velocidade lenta como uma saída VL e a rápida como VR. Faz o grafcet e o projeto.



6) Pressionando o botão S1 o guindaste vai para a izquerda até o fim de curso S5 onde para, só o botão S3 faz ao guindaste retornar até a posição S6. O botão S2 envia o guindaste para a direita até S7 e para retornar só o botão S4 faz este retornar até S6. Sem grafcet.



9. SINAIS ANALÓGICOS NO PIC

Como os sinais dos periféricos são substancialmente diferentes daqueles que o microcontrolador pode entender (zero e um), eles devem ser convertidos num formato que possa ser compreendido pelo microcontrolador.

9.1. CONCEITOS BÁSICOS DOS CONVERSORES

Uma entrada analógica de tensão para que possa entrar no CLP, PIC ou computador precisa ser amostrada e convertida num valor numérico através de um conversor A/D.

A seguinte figura mostra como muda a tensão contínua sobre o tempo. Existem três amostras indicadas na figura. O processo de amostragem dos dados não é instantâneo, existe um tempo inicial e final. O tempo requerido para a amostra é chamado de tempo de amostragem.

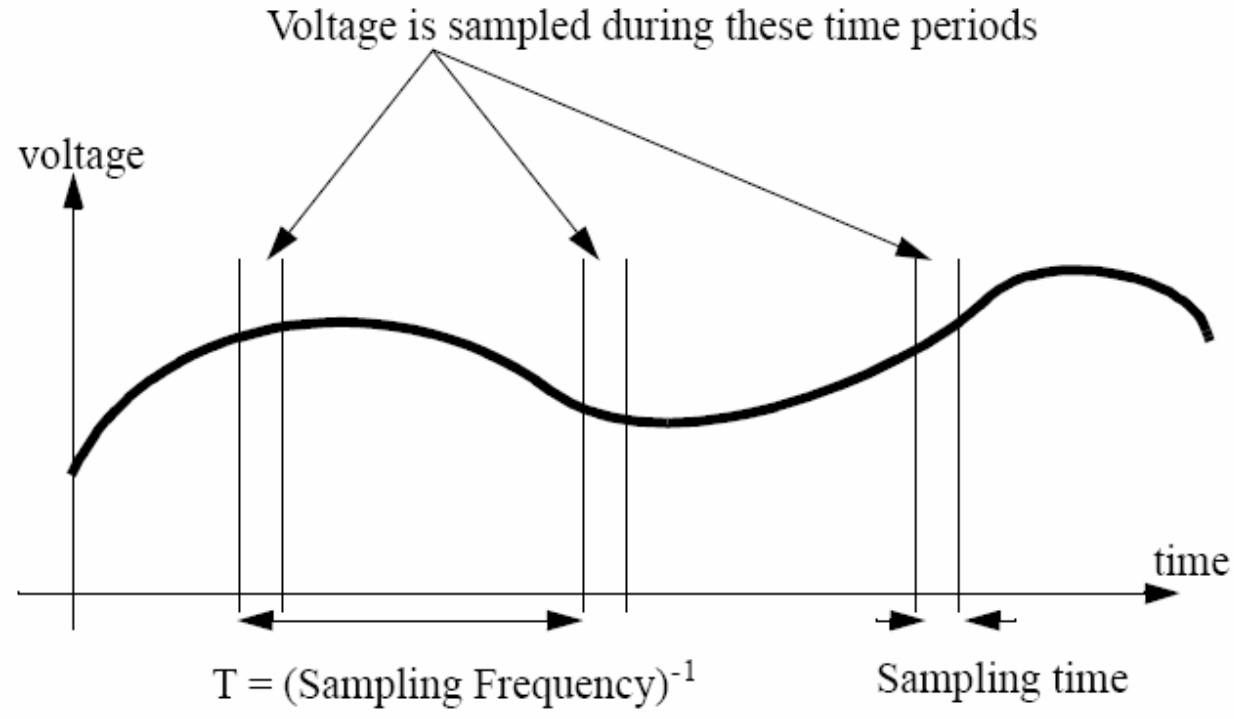


Ilustração 35: Amostragem de um sinal analógico

O conversor A/D pode só adquirir um número limitado de amostras por segundo. O tempo entre cada amostra é chamado de período de amostragem T e a inversa desse período é a freqüência de amostragem. A freqüência de amostragem dependerá de alguns fatores como do programa e do Clock.

Existem vários ranges de entradas analógicas padronizadas como de 0 a 5V, de 0 a10V, de -5 a 5V, de -10 a 10V, de 0 a 20 mA, de 4 a 20 mA. Quando o tipo do sinal analógico for corrente, é necessário colocar em serie uma resistência de para transformar este sinal em tensão. A resistência recomendada é de 250 ohms, desta forma ranges de 0 a 20ma e de 4 a 20 mA transformam-se em faixas de 0 a 5V e de 1 a 5V respectivamente.

O número de bits do conversor A/D é o número de bits da palavra resultado. Se o conversor A/D fosse de 8 bits, então o resultado ter 256 níveis. O PIC que estamos usando tem conversores de 10 bits.

No exemplo da figura seguinte vemos que se a freqüência de amostragem fosse menor que a freqüência do sinal o instrumento estaria perdendo informação. Para isto é recomendado que a freqüência do CLP deve de ser pelo menos duas vezes a do sinal.

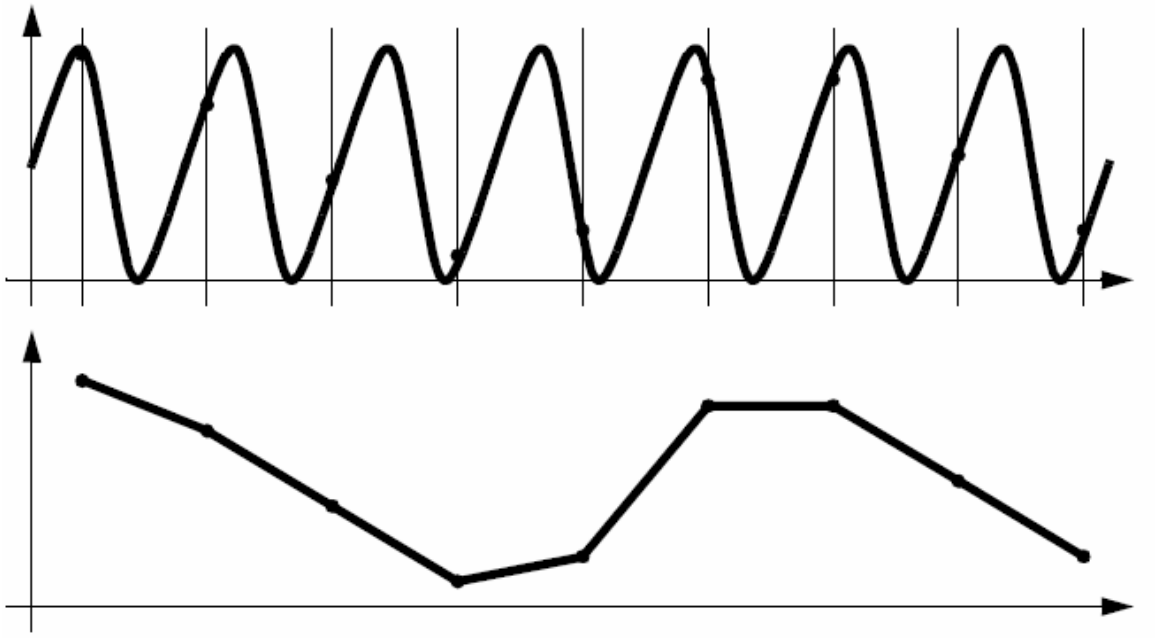
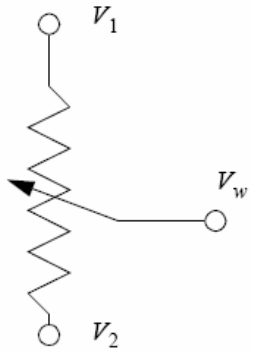


Ilustração 36: Amostragem de um sinal com freqüência muito maior a frequencia de amostragem

Exercícios 9.1:

a) Sendo que o range de uma entrada analógica de um CLP é de 0 a10V, e o conversor tivesse uma resolução de 10 bits. Qual seria o valor do número resultante para uma entrada de 5 volts?

b) No desenho temos um potenciômetro que atua como sensor do braço Mecânico construído por alunos do CEFETES/AN3 2003, este potenciômetro representa a posição do braço, sendo que a posição do braço é de 0 a 100%.
E o potenciômetro na posição 0% tem uma tensão de saída de 1V e na de 100% uma tensão de 4.5 V, e esta tensão representa a entrada analógica do PIC na variável SenAnalog1 (0/4095-10bits e de 0 a 5V), disser qual será a variação de SenAnalog1.



9.2. Tratamento de Entradas Analógicas no PIC

As funções necessárias para poder desenvolver uma aplicação usando sinais analógicos são as seguinte:

#device adc=10

Especifica ao compilar o número de bits da conversão analógico digital

setup_adc_ports (RA0_RA1_RA3_analog);//Configura Vdd e Vss como Vref

Configura os pinos para serem entradas analógicas, digital ou uma combinação destes. O parâmetro escrito dentro do parênteses é uma constante e são usadas constantes diferentes para cada tipo de PIC. Checar o arquivo de cabeçalho para visualizar a lista completa disponível destas constantes. As constantes ALL_ANALOG e NO_ANALOGS estão disponíveis em todos os PICs.

Exemplos:

setup_adc_ports (RA0_RA1_RA3_analog); se configuram as entradas RA0, RA1 e RA3 como entradas analógicas e as rederencias de tensão são 5v e 0v. Todos os outros pinos são digitais

setup_adc_ports(ALL_ANALOG); todos os pinos são entradas analógicas

setup_adc_ports(RA0_RA1_ANALOGRA3_REF); RA0 e RA1 são analógicos e RA3 é usado como referencia. Todos os outros pinos são digitais.

No arquivo cabeçalho 16F877A.h encontramos as constantes:

```
// Constants used in SETUP_ADC_PORTS() are:
#define NO_ANALOGS                      7    // None
#define ALL_ANALOG                      0    // A0 A1 A2 A3 A5 E0 E1 E2 Ref=Vdd
#define AN0_AN1_AN2_AN4_AN5_AN6_AN7_VSS_VREF 1    // A0 A1 A2 A5 E0 E1 E2 Ref=A3
#define AN0_AN1_AN2_AN3_AN4             2    // A0 A1 A2 A3 A5 Ref=Vdd
#define AN0_AN1_AN2_AN4_VSS_VREF        3    // A0 A1 A2 A5 Ref=A3
#define AN0_AN1_AN3                     4    // A0 A1 A3 Ref=Vdd
#define AN0_AN1_VSS_VREF                5    // A0 A1 Ref=A3
#define AN0_AN1_AN4_AN5_AN6_AN7_VREF_VREF 0x08    // A0 A1 A5 E0 E1 E2 Ref=A2,A3
#define AN0_AN1_AN2_AN3_AN4_AN5         0x09    // A0 A1 A2 A3 A5 E0 Ref=Vdd
#define AN0_AN1_AN2_AN4_AN5_VSS_VREF    0x0A    // A0 A1 A2 A5 E0 Ref=A3
#define AN0_AN1_AN4_AN5_VREF_VREF       0x0B    // A0 A1 A5 E0 Ref=A2,A3
```

```
#define AN0_AN1_AN4_VREF_VREF           0x0C    // A0 A1 A5 Ref=A2,A3
#define AN0_AN1_VREF_VREF               0x0D    // A0 A1 Ref=A2,A3
#define AN0                             0x0E    // A0
#define AN0_VREF_VREF                   0x0F    // A0 Ref=A2,A3
#define ANALOG_RA3_REF          0x1         //!old only provided for compatibility
#define A_ANALOG                0x2         //!old only provided for compatibility
#define A_ANALOG_RA3_REF        0x3         //!old only provided for compatibility
#define RA0_RA1_RA3_ANALOG      0x4         //!old only provided for compatibility
#define RA0_RA1_ANALOG_RA3_REF 0x5          //!old only provided for compatibility
#define ANALOG_RA3_RA2_REF              0x8   //!old only provided for compatibility
#define ANALOG_NOT_RE1_RE2              0x9   //!old only provided for compatibility
#define ANALOG_NOT_RE1_RE2_REF_RA3      0xA   //!old only provided for compatibility
#define ANALOG_NOT_RE1_RE2_REF_RA3_RA2  0xB   //!old only provided for compatibility
#define A_ANALOG_RA3_RA2_REF            0xC   //!old only provided for compatibility
#define RA0_RA1_ANALOG_RA3_RA2_REF      0xD   //!old only provided for compatibility
#define RA0_ANALOG                      0xE   //!old only provided for compatibility
#define RA0_ANALOG_RA3_RA2_REF          0xF   //!old only provided for compatibility
```

setup_adc (opções);

Esta função configura o relojo para o conversor A/D. As opções são variáveis ou constantes inteiras de 8 bits. As opções variam de acordo com cada PIC e podem ser encontradas no arquivo de cabeçalho de cada dispositivo. No caso do Pic 16F877A encontramos:

- ADC_OFF
- ADC_CLOCK_DIV_2
- ADC_CLOCK_DIV_4
- ADC_CLOCK_DIV_8
- ADC_CLOCK_DIV_16
- ADC_CLOCK_DIV_32
- ADC_CLOCK_DIV_64
- ADC_CLOCK_INTERNAL

Ejemplo:

setup_adc(ADC_CLOCK_INTERNAL);

set_adc_channel (1);

Sintaxe: set_adc_channel (***canal***)

Parâmetros: ***canal*** é o número do canal selecionado. O número do canal começa com zero referente ao pino AN0.

Função: Especifica o canal que será usado para executar o próximo comando READ_ADC. Verifique que passe um pequeno tempo antes de trocar de canal para que possa ter maior exatidão. O tempo varia em função do valor da impedância da fonte de entrada. Em geral 10us são suficientes. Não será necessário trocar de canal antes de cada leitura se o canal não muda. Esta função esta disponível em PICs com conversores A/D.

Exemplo:

```
set_adc_channel(2);
delay_us(10);
value = read_adc();
```

read_adc();

Sintaxe: value = read_adc ([***mode***])

Parâmetros: ***mode*** é um parâmetro opcional. Se usado estes podem ser:

ADC_START_AND_READ (this is the default)
ADC_START_ONLY (starts the conversion and returns)
ADC_READ_ONLY (reads last conversion result)

Retorna: o valor da conversão analógica digital para o caso do PIC 16F877A, retorna um valor de 16 bits.
Função: Esta função faz a leitura do valor digital do conversor analógico digital.

Examples:

```
setup_adc( ADC_CLOCK_INTERNAL );
setup_adc_ports( ALL_ANALOG );
set_adc_channel(1);
value = read_adc();
```

Exemplo 9.2:

Neste exemplo existe um sinal analógico variando de 0 a 5v entrando no pino RA0. Se a tensão for menor que 1v um led deverá indicar que a tensão esta baixa, se a tensão estiver entre 1 e 4 volts um outro led indicará que a tensão esta média, e se a tensão estiver entre 4 e 5v um outro led indicará que a tensão esta alta.

Este programa foi feito com a intenção de servir como modelo para o desenvolvimento de outros projetos.

Como o PIC16F877A tem uma resolução de 10 bits para os sinais analógicos então na escala da conversão variará de 0 a 1023.

Quando se trata de conversão analógico-digital é comum usar escalonamentos, existe uma forma de transformar valores a outras escalas. Relembremos um pouco de instrumentação:

$$\frac{EscNova - OffsetEscNova}{SpanEscNova} = \frac{EscVelha - OffsetEscVelha}{SpanEscVelha}$$

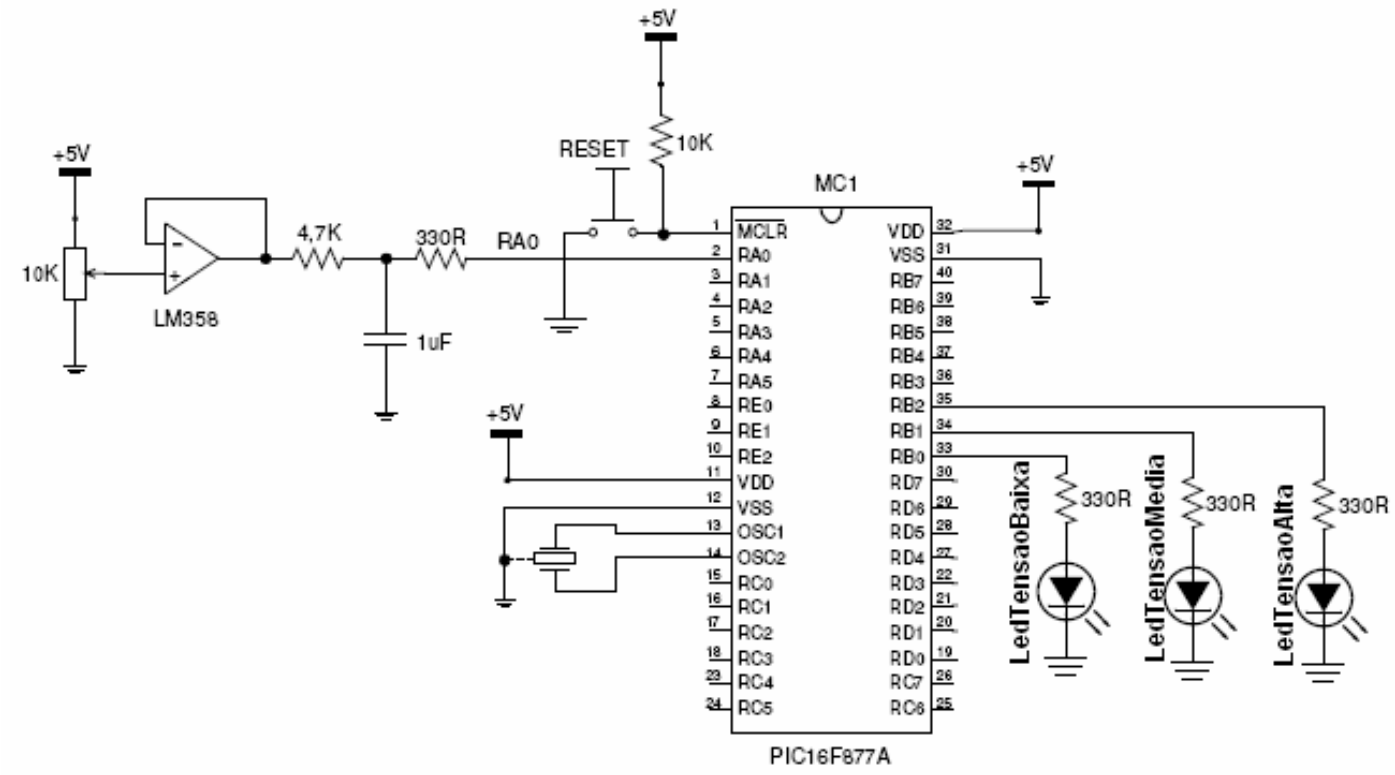
No caso deste exemplo não temos offset. A escala nova vária de 0 a 5 v, a velha de 0 a 1023, para transformar da escala velha à nova:

$$EscNova = 5 * Conversão/1023$$

```
/*********************************************************
*                      CEFETES
* Prof: Marco Antonio
* Exemplo 4: Conversão analógica digital
* Materia: Eletrônica Digital
* Data: Julho 2006
*********************************************************/

#include          <16f877A.h>     // microcontrolador utilizado
#device adc=10
```

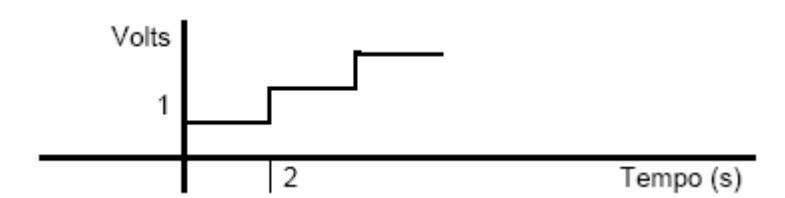
```
#fuses  xt,wdt,noprotect,put,brownout,nolvp,nocpd,nowrt          // configuração dos fusíveis

#use    delay(clock=4000000, RESTART_WDT)

#use    fast_io(a)
#use    fast_io(b)
#use    fast_io(c)
#use    fast_io(d)
#use    fast_io(e)

#byte   porta = 0x05
#byte   portb = 0x06
#byte   portc = 0x07
#byte   portd = 0x08
#byte   porte = 0x09


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                      ENTRADAS                                     *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
  float conversao = 0;              // armazena o resultado da conversão AD
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                       SAÍDAS                                      *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
        #bit LedTensaoBaixa = portb.0
        #bit LedTensaoMedia = portb.1
        #bit LedTensaoAlta = portb.2


void main()
{
                // configura microcontrolador
                setup_adc_ports (RA0_RA1_RA3_analog);//Configura Vdd e Vss como Vref
                setup_adc (adc_clock_div_32);
                setup_counters (rtcc_internal, WDT_2304MS);
                set_adc_channel (1);

                // configura os tris
                set_tris_a(0b11011111);                         // configuração da direção dos pinos
de I/O
                set_tris_b(0b00000000);
                set_tris_c(0b11111101);
                set_tris_d(0b00000000);
                set_tris_e(0b00000100);

                // inicializa os ports
                porta=0x00;                                     // limpa porta
                portb=0x00;                                     // limpa portb
                portc=0x00;                                     // limpa portc
                portd=0x00;                                     // limpa portd
                porte=0x00;                                     // limpa porte


        while(TRUE)                             // rotina principal
        {
        RESTART_WDT();                                  // incia o watch-dog timer

                conversao = read_adc();                 // inicia conversão AD
                conversao = (conversao * 5);    // faz regra de 3 para converter o valor,
                conversao = (conversao / 1023); // das unidades de AD em Volts.
```

```
                if (conversao<=1)
                        LedTensaoBaixa = 1;
                else
                        LedTensaoBaixa = 0;

                if ((conversao<=4)&&(conversao>=1))
                        LedTensaoMedia = 1;
                else
                        LedTensaoMedia = 0;

                if ((conversao<=5)&&(conversao>=4))
                        LedTensaoAlta = 1;
                else
                        LedTensaoAlta = 0;

        }
}
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                    Fim do Programa                                    *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
```



Exercícios:

1. Construir uma rampa de aceleração, para acionamento da bomba do problema 2, mostrada no gráfico abaixo:

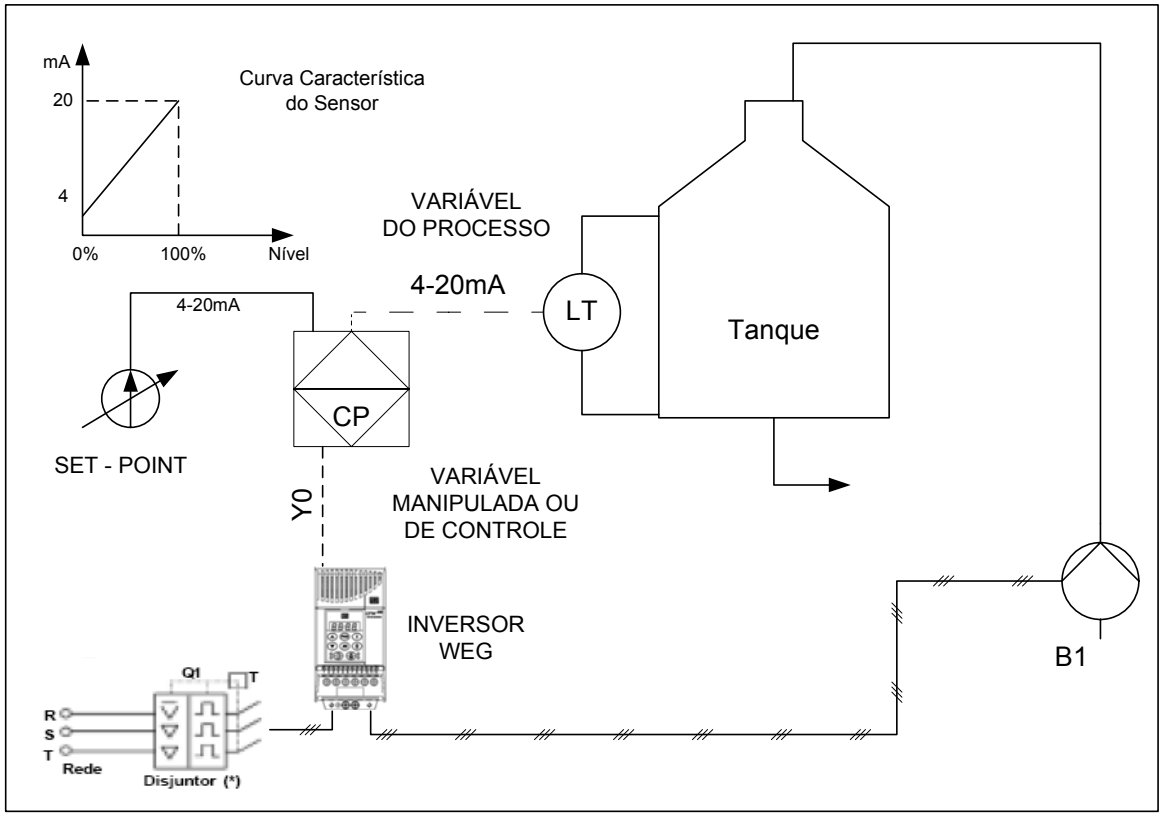


A cada 2 s o uC deve adicionar à sua saída analógica 1 volt, até alcançar 10 volts. Isto será iniciado quando o operador pressionar o botão start, e quando for pressionado o botão stop a rampa deverá ser decrescente até o motor parar. Para este problema assuma que a saída analógica é a porta B e que através de um DAC é conectada à entrada analógica do inversor.

2. Em uma linha de produção temos 3 produtos. O primeiro produto foi embalado em uma caixa de 10 cm de altura, o segundo com 8 cm de altura e o terceiro com 5 cm de altura. Temos que contar a quantidade de cada produto que passa na esteira pelos sensores infravermelhos. Existe um sensor ultra-som para medir a altura das caixas e um sensor capacitivo para determinar que as caixas estão baixo o sensor Ultra-som. Depois que passem 5, 7 e 8 caixas de cada produto 1,2 e 3 respectivamente as lâmpadas L1, L2 e L3 deverão ligar por um tempo de 5 segundos cada e a contagem deverá ser reiniciada. Quer disser se passam 5 caixas do produto 1 a lâmpada 1 liga e depois de 5 segundos esta desliga.

3. Fazer um programa para representar através da variável NívelT1 o nível do tanque 1 em unidades de engenharia. Sabe-se que o LT usado para medir o nível é um Ultra-som sendo seu range de 0-20 mA na saída e de 0-10 m na entrada. Mostrar em 3 displays de 7 segmentos o valor de 0 a 100% do nível.

4. Fazer um controlador ON-OFF, para a automatização da planta mostrada na seguinte figura.



Assuma que a fonte de corrente variável esta instalada na entrada AN0 através de uma resistência de 250 ohms e que a sinal do PDT esta instalado na entrada AN1 também com uma resistência de 250 ohms. A velocidade da bomba não é variada pelo uC, mas o controle de liga e desliga sim.

10.COMUNICAÇÃO SERIAL

Em muitas aplicações microcontroladas, pode ser necessário realizar a comunicação entre o microcontrolador e um ou mais dispositivos externos.

Esses dispositivos podem estar localizados tanto na mesma placa do circuito, como fora dela, a metros ou mesmo dezenas de quilômetros de distância.

A escolha do sistema de comunicação mais adequado para realizar a tarefa depende de diversos fatores, como: velocidade, imunidade a ruídos, custo, etc. As duas formas conhecidas são a comunicações seriais e a paralela.

Na comunicação paralela enviamos uma palavra por vez ao longo de um barramento composto por vários sinais. A comunicação com impressoras é tipicamente uma comunicação paralela. A velocidade é alta, porém as distâncias são curtas. Além do envio dos dados deve-se também enviar sinais de controle.

Na comunicação serial o dado é enviado bit por bit. O cabo que conecta os dispositivos pode ser mais longo, em virtude de características especiais do sinal que é transmitido.

10.1. Comunicação Serial Síncrona x Comunicação Serial Assíncrona

Diversos protocolos de comunicação operam sobre comunicação serial: redes de campo, comunicação com modens, etc...
Existem dois modos de comunicação serial: Síncrono e Assíncrona.

Comunicação Serial Síncrona
Neste modo de comunicação o transmissor e o receptor devem ser sincronizados para a troca de comunicação de dados. Geralmente uma palavra de SINCRONISMO é utilizada para que ambos ajustem o relógio interno. Após a sincronização os bits são enviados seqüencialmente, até uma quantidade pré-combinada entre os dispositivos.

Comunicação Serial Assíncrona
Esta é a forma mais usual de transmissão de dados. Não existe a necessidade de sincronização entre os dispositivos, uma vez que os caracteres são transmitidos individualmente e não em blocos como na comunicação síncrona. A transmissão de cada caractere é precedida de um bit de start e terminada por 1 (1/2 ou 2) bit(s) de stop.

O PIC 16F877A suporta duas formas de comunicação serial :

- Comunicação assíncrona universal (formato RS232) SCI ou USART;
- Comunicação síncrona entre dispositivos SPI;

Ilustração 37: Composição típica de um frame assíncrono

Notem que neste frame existe um bit de start, 8 bits de dados, um bit de paridade e um bit de parada. A paridade pode ser par ou ímpar. Quando a paridade é par o bit de paridade é gerado de modo que o número de 1s resultante na palavra mais o bit de paridade sejam pares.

Por exemplo, se a palavra 10001010 está sendo transmitida, ou recebida, o bit de paridade deve ser 1, para que o conjunto palavra + bit de paridade tenha sempre um número par de 1s. Se a paridade usada for ímpar o bit de paridade no exemplo anterior será zero.

No processo de transmissão assíncrona, os dispositivos envolvidos no processo de comunicação devem ter a mesma taxa de transmissão e recepção.

10.2. O RS232 no PIC

Para realizar a interface de comunicação serial entre o PC e o circuito controlador, utiliza-se o conversor de nível de tensão padrão RS232 para TTL/CMOS (MAX232 – CI2) acrescido de quatro capacitores (1 µF / +50 VCC) em conjunto com o conector DB-9 (X1). O circuito integrado dedicado MAX232 estabelece a conversão dos níveis de tensão do microcontrolador PIC (0 VCC e +5 VCC) para os níveis de tensão da porta serial do computador padrão RS232 (-12 VCC e +12 VCC) e vice-versa.

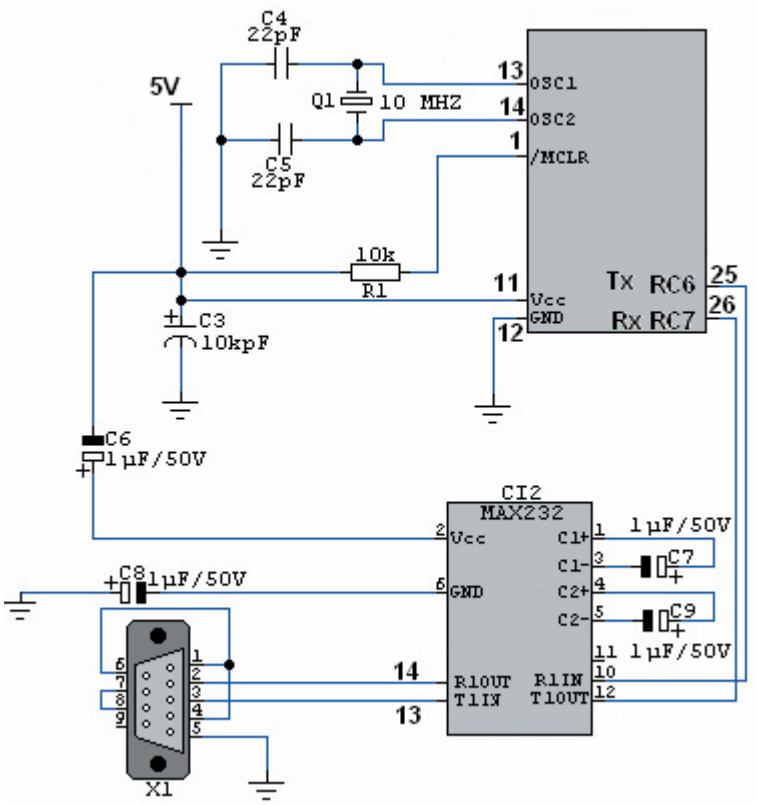


Ilustração 38: Circuito básico de ligação do RS232 do PIC

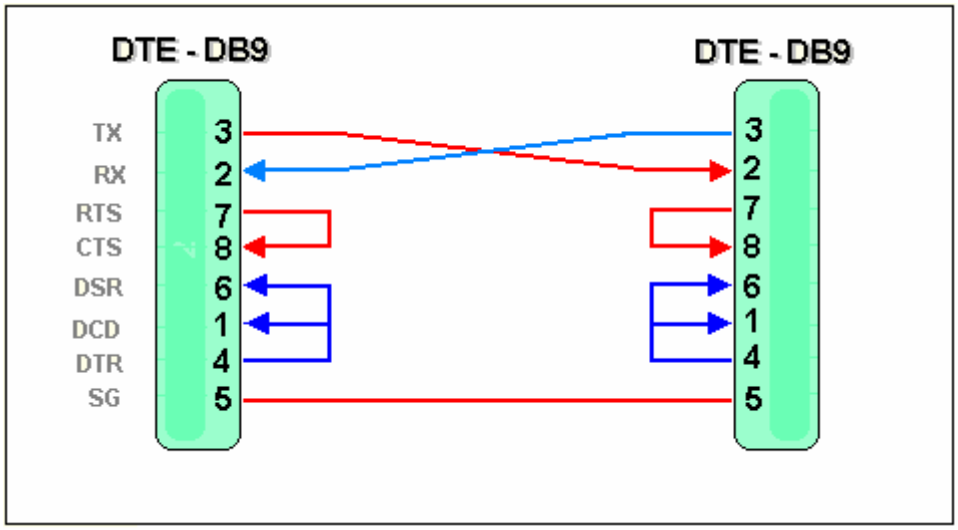


Ilustração 39: Cabo serial para conectar DB9 com DB9

EXEMPLO:

Neste exemplo esta sendo usada uma entrada analógica de 0 a 5V e o seu valor é transferido através do RS232 a um computador. No computador pode-se desenvolver um programa para poder trocar dados com o programa do PIC. Mas também podemos usar o hiperTerminal que é um acessório do Windows para comunicação serial. Basta ir a:

Iniciar\Programas\Acessórios\Comunicações\Hiperterminal

O programa exige que o operador digite uma letra “d” para que o PIC possa transmitir a informação.

```
/*********************************************************
*                      CEABRA
* Prof: Marco Antonio
* Exemplo 4: Conversão analógica digital e comunicação via RS232
* Matéria: Eletrônica Digital
* Data: Outubro 2005
*********************************************************/

#include          <16f877A.h>     // microcontrolador utilizado
#device adc=10

#fuses  xt,wdt,noprotect,put,brownout,nolvp,nocpd,nowrt          // configuração dos fusíveis

#use     delay(clock=4000000, RESTART_WDT)
#use     rs232(baud=9600,xmit=pin_c6,rcv=pin_c7)

#define t_filtro 500                                  // tamanho do filtro

         float conversao = 0;               // armazena o resultado da conversão AD
   char Comando;

#use     fast_io(a)
#use     fast_io(b)
#use     fast_io(c)
#use     fast_io(d)
#use     fast_io(e)

#byte    porta = 0x05
#byte    portb = 0x06
```

```
#byte    portc = 0x07
#byte    portd = 0x08
#byte    porte = 0x09

        void main()
{
                // configura microcontrolador
                setup_adc_ports (RA0_RA1_RA3_analog);//Configura Vdd e Vss como Vref
                setup_adc (adc_clock_div_32);
                setup_counters (rtcc_internal, WDT_2304MS);
                set_adc_channel (1);

                setup_vref(VREF_LOW|6);

                // configura os tris
                set_tris_a(0b11011111);                 // configuração da direção dos pinos
de I/O
                set_tris_b(0b00000011);
                set_tris_c(0b11111101);
                set_tris_d(0b00000000);
                set_tris_e(0b00000100);

                // inicializa os ports
                porta=0x00;                             // limpa porta
                portb=0x00;                             // limpa portb
                portc=0x00;                             // limpa portc
                portd=0x00;                             // limpa portd
                porte=0x00;                             // limpa porte


        while(TRUE)                             // rotina principal
        {
        RESTART_WDT();                                  // incia o watch-dog timer
    Comando=getc();
    if (Comando=='d')
    {
       conversao = read_adc();          // inicia conversão AD
                    conversao = (conversao * 5); // faz regra de 3 para converter o valor,
                    conversao = (conversao / 1023); // das unidades de AD em Volts.
       printf("Voltimetro: %f\r\n", conversao);
       delay_ms(1000);
     }
    // printf("Voltimetro: %c\r\n", Comando);
    // delay_ms(3000);
        }

}
```

Exercícios:

1) Olhando o help do CCS explicar detalhadamente o programa feito no exemplo.
2) Pesquisar e mostrar a diferencia entre o RS485 e RS232.
3) Montar o circuito do exemplo explicado neste capítulo.

```
/*******************************************************
*                      CEABRA
* Prof: Marco Antonio
* Exemplo 3: Controle de nível num tanque
*           Uma bomba é acionada quando o tanque esta vazio
*          e quando este esteja cheio, a bomba desliga. Dois sensores
*          são instalados, sensor de tanque cheio e tanque vazio.
* Materia: Eletrônica Digital
* Data: Outubro 2005
*******************************************************/

#include       <16f877A.h> // microcontrolador utilizado

/*#fuses       xt,wdt,noprotect,put,brownout,nolvp,nocpd,nowrt       // configuração
dos fusíve*/
#fuses xt,WDT,NOPROTECT,put, NOLVP

#use   delay(clock=24000000, RESTART_WDT)

#use   fast_io(a)
#use   fast_io(b)
#use   fast_io(c)
#use   fast_io(d)
#use   fast_io(e)

#byte  porta = 0x05
#byte  portb = 0x06
#byte  portc = 0x07
#byte  portd = 0x08
#byte  porte = 0x09

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                      ENTRADAS                                     *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
// As entradas devem ser associadas a nomes para facilitar a programação e
//futuras alterações do hardware.

#bit    Tanque_Cheio = portb.0    // 1 -> Cheio

#bit    Tanque_Vazio = portb.1    //0 -> Vazio

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                      SAÍDAS                                       *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
// AS SAÍDAS DEVEM SER ASSOCIADAS A NOMES PARA FACILITAR A
PROGRAMAÇÃO E
//FUTURAS ALTERAÇÕES DO HARDWARE.

#bit    Bomba = portb.2              // 1 -> Bomba ligada
```

```
void main ()
        {
        // configura CONFIG
        setup_counters(RTCC_INTERNAL, WDT_2304MS);

        // configura os TRIS
        set_tris_a(0b11111111);          // configuração dos pinos de I/O
        set_tris_b(0b00000011);
        set_tris_c(0b11111111);
        set_tris_d(0b11111111);
        set_tris_e(0b00000111);

        // inicializa os ports
        porta=0x00;                 // limpa porta
        portb=0x00;                 // limpa portb
        portc=0x00;                 // limpa portc
        portd=0x00;                 // limpa portd
        porte=0x00;                 // limpa porte

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                    Loop principal                    *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

        while(TRUE)
                {
                RESTART_WDT();

    IF ((Tanque_Vazio==0) && (Tanque_Cheio==0))
      Bomba=1;
    IF ((Tanque_Vazio==1) && (Tanque_Cheio==1))
      Bomba=0;
     }
                                // FIM DO PROGRAMA
        }
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                  Fim do Programa                                    *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
```

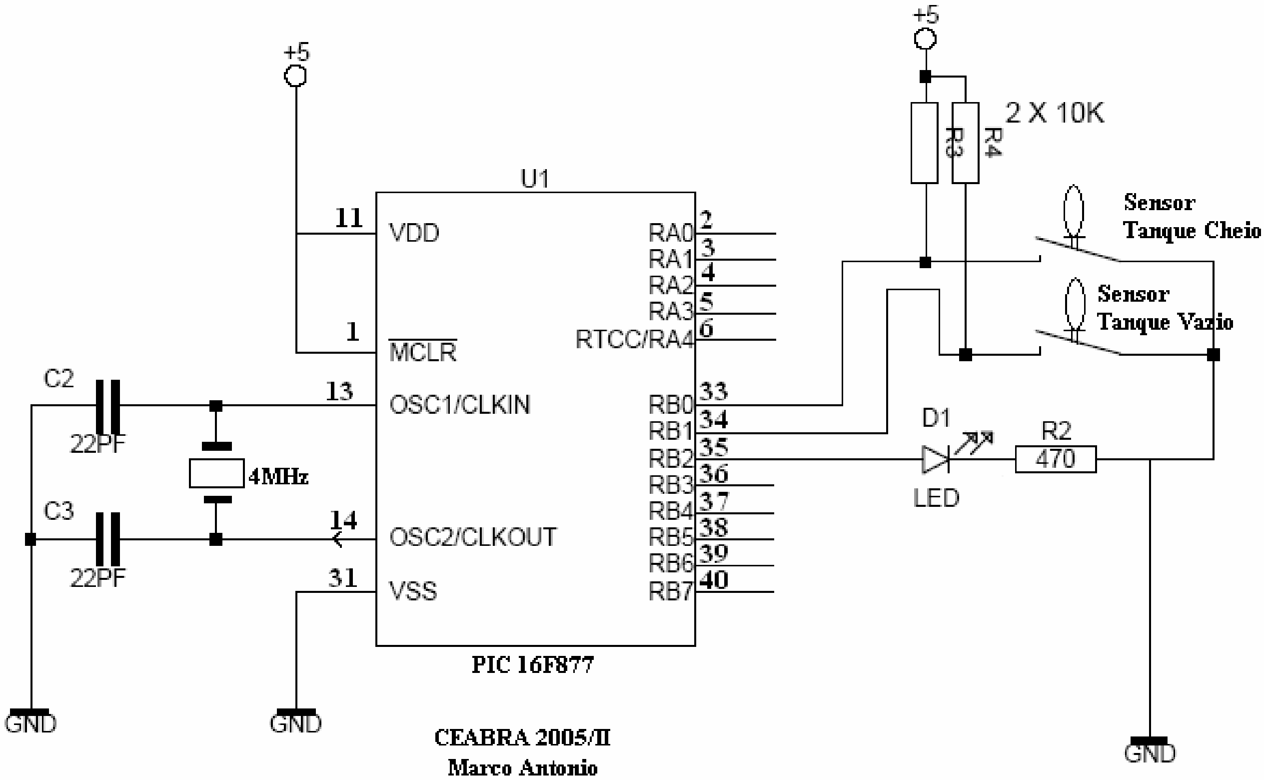


Ilustração 40: Circuito para o Exemplo 3

Exemplo4:

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *          Programação em C - Recursos Básicos de programação         *
 *                          Exemplo 4                          *
 *                                                          *
 *            CENTRO DE TREINAMENTO - MOSAICO ENGENHARIA              *
 *                                                          *
 *  TEL: (0XX11) 4992-8775             SITE: www.mosaico-eng.com.br     *
 *                         E-MAIL: mosaico@mosaico-eng.com.br  *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *   VERSÃO : 1.0                                           *
 *     DATA : 05/06/2003                                       *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                         Descrição geral                          *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
```

```
// Este software está preparado para ler quatro botões e tocar o buzzer com
//duração variavel conforme a tecla pressionada, além de acender o led
//indicando a última tecla pressionada.

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *            DEFINIÇÃO DAS VARIÁVEIS INTERNAS DO PIC              *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include       <16f877A.h> // microcontrolador utilizado

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                  Configurações para gravação                  *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#fusesxt,wdt,noprotect,put,brownout,nolvp,nocpd,nowrt       // configuração dos
fusíveis

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *            Definições para uso de Rotinas de Delay               *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#use   delay(clock=4000000, RESTART_WDT)

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                    Constantes internas                        *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
//A definição de constantes facilita a programação e a manutenção.
#define        t_filtro 20              // tamanho do filtro

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *            Definição e inicialização das variáveis             *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
//Neste bloco estão definidas as variáveis globais do programa.

        int status_botoes = 0;                    // armazena o estado lógico dos
botões
        int status_leds = 0;              // atualiza leds conforme o botão
pressionado
        int per = 0;

        int filtro1 = t_filtro;           // inicia filtro do botao1
        int filtro2 = t_filtro;           // inicia filtro do botao2
        int filtro3 = t_filtro;           // inicia filtro do botao3
        int filtro4 = t_filtro;           // inicia filtro do botao4

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *              Definição e inicialização dos port's              *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#use   fast_io(a)
```

```
#use   fast_io(b)
#use   fast_io(c)
#use   fast_io(d)
#use   fast_io(e)

#byte  porta = 0x05
#byte  portb = 0x06
#byte  portc = 0x07
#byte  portd = 0x08
#byte  porte = 0x09

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                Declaração dos flags de software                 *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
//A definição de flags ajuda na programação e economiza memória RAM.

#bit    botao1_press = status_leds.0
#bit    botao2_press = status_leds.1
#bit    botao3_press = status_leds.2
#bit    botao4_press = status_leds.3

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                         ENTRADAS                                *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
// As entradas devem ser associadas a nomes para facilitar a programação e
//futuras alterações do hardware.

#bit    botao1 = status_botoes.0   // Estado do botão 1
                                   // 1 -> Liberado
                                       // 0 -> Pressionado

#bit    botao2 = status_botoes.1   // Estado do botão 2
                                   // 1 -> Liberado
                                       // 0 -> Pressionado

#bit    botao3 = status_botoes.2           // Estado do botão 3
                                   // 1 -> Liberado
                                   // 0 -> Pressionado

#bit    botao4 = status_botoes.3   // Estado do botão 4
                                    // 1 -> Liberado
                                  // 0 -> Pressionado

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                         SAÍDAS                                  *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
// AS SAÍDAS DEVEM SER ASSOCIADAS A NOMES PARA FACILITAR A
PROGRAMAÇÃO E
//FUTURAS ALTERAÇÕES DO HARDWARE.
```

```
#bit    buzzer = porta.5

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                    Tabela de Valores                                    *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
byte const tabela[16] =
        {16,32,32,48,64,80,96,112,128,144,160,176,192,208,224,240};

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                Configuração do Microcontrolador                         *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
// A rotina principal simplesmente limpa o WDT, pois toda a lógica do
// programa é tratada dentro das interrupções.
        void main()
        {
                // configura microcontrolador
                setup_adc_ports (no_analogs);
                setup_counters(rtcc_internal , rtcc_div_8);
                setup_timer_2 (t2_div_by_16,per,1);

                // configura os tris
                set_tris_a(0b11011111);
                set_tris_b(0b11110000);
                set_tris_c(0b11111111);
                set_tris_d(0b11111111);
                set_tris_e(0b00000111);

                // inicializa os ports
                porta=(0b00000000);
                portb=(0b00000000);
                portc=(0b00000000);
                portd=(0b00000000);
                porte=(0b00000000);

                // habilita a interrupção de TMR0
                enable_interrupts(GLOBAL|INT_RTCC);

                #priority timer0,timer2              // prioridade para a int de trm0

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                      Rotina principal                                   *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

        while(TRUE)                                  // rotina principal
                {
        RESTART_WDT();                                       // incia o watch-dog timer
                }
        }

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

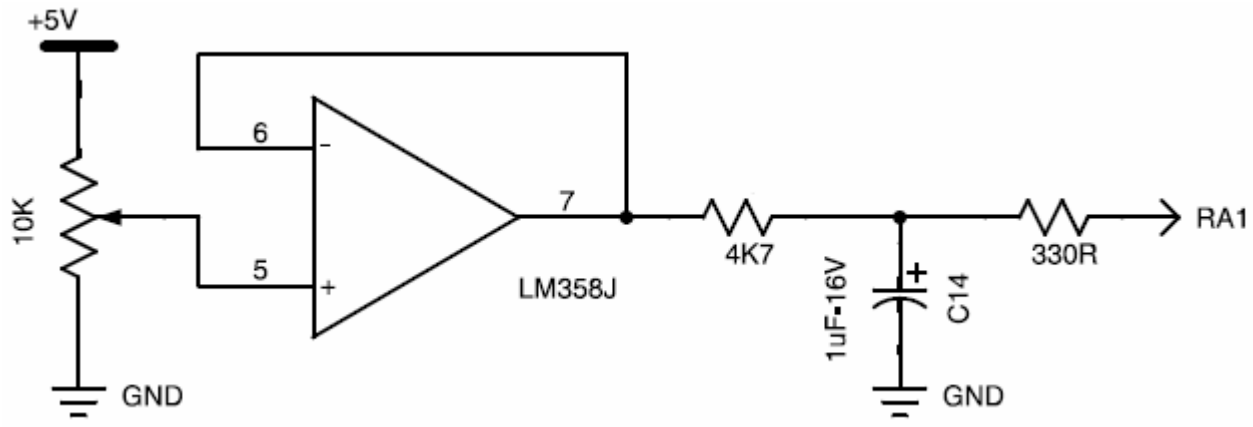
```
 *              Rotina de Tratamento de interrupção de TMR0       *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
// Esta interrupção ocorrerá a cada 2048us.
// O filtro dos botões tem duração de 40ms (2048us x 20) e são decrementados a
cada interrupção.
// Esta interrupção é responsável por converter os pinos dos botões em entrada,
salvar a condição
// dos botões em variável temporária e converter os pinos em saída novamente.

#int_rtcc
void trata_int_tmr0(void)
        {
                set_tris_b(0b00001111);                 // configura os pinos como
entrada para testar os botões
                delay_cycles(4);                // aguarda 4 ciclos de máquina para a
estabilização do portb
                status_botoes = portb;                  // lê o status dos botoes, salva
em variável temporária
                status_botoes = (status_botoes & 0x0f); // zera a parte alta do
registrador
                set_tris_b(0b00000000);                 // configura os pinos como
saída para controle dos leds

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                  Tratamento do Botão 1                  *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

                if (!botao1)                    // testa botão 1
                {                               // botão 1 está pressionado ?
                filtro1--;                      // Sim, então decrementa o filtro
                        if (filtro1 == 0)       // acabou o filtro do botão 1 ?
                        {
                        botao1_press = 1;       // marca que o botão está
pressionado
                        }

                }
                else
                {                               // botão 1 liberado
                        filtro1 = t_filtro;     // inicia o filtro do botão 1
                        botao1_press = 0;       // marca que o botão foi liberado
                }

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                  Tratamento do Botão 2                  *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

                if (!botao2)                    // testa botão 2
                {                               // botão 2 está pressionado ?
                filtro2--;                      // Sim, então decrementa o filtro
```

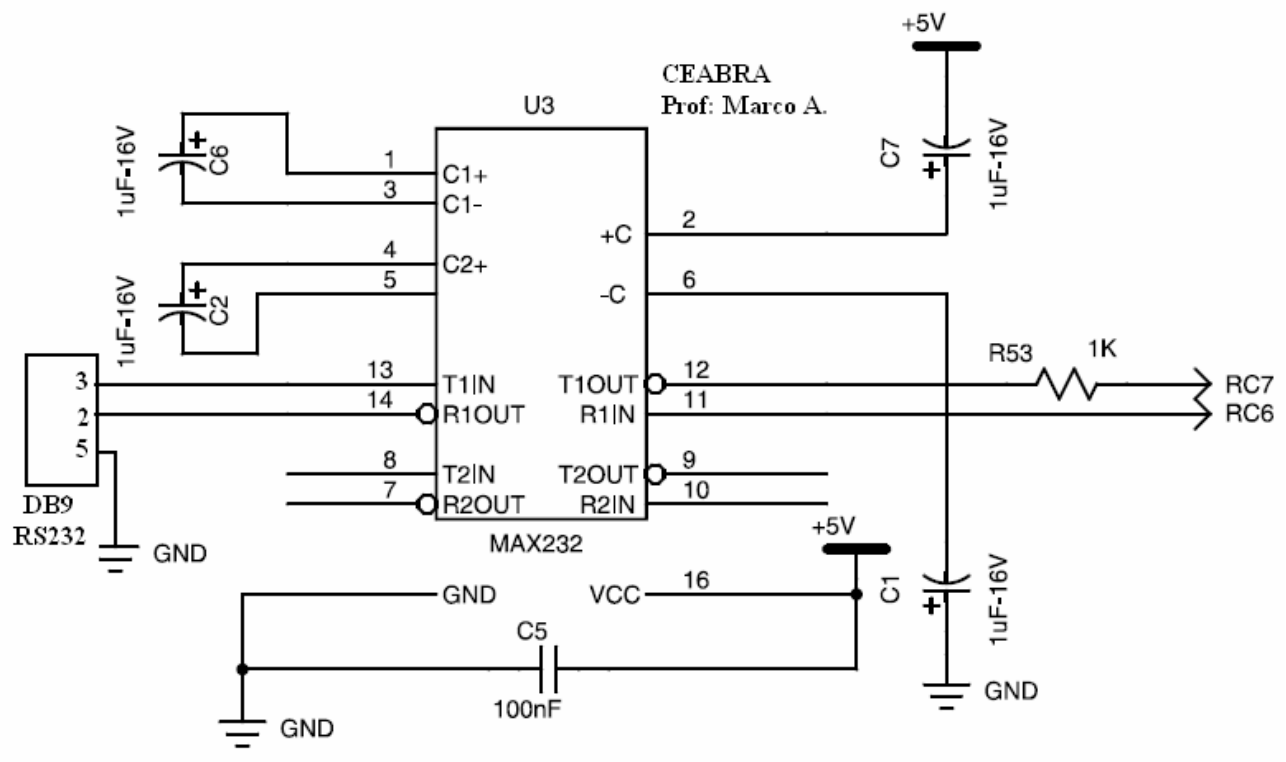
```
                if (filtro2 == 0)           // fim do filtro do botão 2 ?
                {
                botao2_press = 1;           // marca que o botão está
pressionado
                }

        }
        else
        {                                   // botão 2 liberado
                filtro2 = t_filtro;         // inicia o filtro do botão 2
                botao2_press = 0;           // marca que o botão foi liberado
        }

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                 Tratamento do Botão 3                         *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

        if (!botao3)                        // testa botão 3
        {                                   // botão 3 está pressionado ?
        filtro3--;                          // Sim, então decrementa o filtro
                if (filtro3 == 0)           // fim do filtro do botão 3 ?
                {
                botao3_press = 1;           // marca que o botão está
pressionado
                }

        }
        else
        {                                   // botão 3 liberado
                filtro3 = t_filtro;         // inicia o filtro do botão 3
                botao3_press = 0;           // marca que o botão foi liberado
        }

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                 Tratamento do Botão 4                         *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

        if (!botao4)                        // testa botão 4
        {                                   // botão 4 está pressionado ?
        filtro4--;                          // Sim, então decrementa o filtro
                if (filtro4 == 0)           // fim do filtro do botão 4 ?
                {
                botao4_press = 1;           // marca que o botão está
pressionado
                }

        }
        else
                {                           // botão 4 liberado
                filtro4 = t_filtro;         // inicia o filtro do botão 4
```

```
                botao4_press = 0;           // marca que o botão foi liberado
                }

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *           Atualiza Leds conforme botões pressionados          *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
                portb = status_leds;        // atualiza os leds

                if (status_leds == 0)
                {
                        per = 0xff;
                        setup_timer_2 (t2_div_by_16,per,1);
                        disable_interrupts (INT_TIMER2);
                        buzzer = 0;
                }
                else
                {
                        per = (tabela[status_leds]);        // consulta tabela e inicializa
timer2.
                        setup_timer_2 (t2_div_by_16,per,1);
                        enable_interrupts (INT_TIMER2);         // habilita interrupção de
timer2.
                }
        }

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *           Rotina de Tratamento de interrupção de TMR2          *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
// Está interrupção só irá ocorrer quando alguma tecla estiver pressionada,
// o periodo de ocorrência depende do botão ou da conbinação de botões
pressionados,
// ela irá inverter o pino de I/O do buzzer a cada interrupção.

#int_timer2
void trata_int_tmr2(void)
                {
                        if (buzzer != 0)            // o buzzer está ligado ?
                                {
                                buzzer = 0;         // sim, então desliga
                                }
                        else
                                {
                                buzzer = 1;         // não, então liga
                                }
                }
```

```
/*******************************************************
*                    CEABRA
* Prof: Marco Antonio
* Exemplo 4: Conversaõ analógica digital e comunicação vía RS232
* Materia: Eletrônica Digital
* Data: Outubro 2005
*******************************************************/

#include       <16f877A.h> // microcontrolador utilizado
#device adc=10

#fusesxt,wdt,noprotect,put,brownout,nolvp,nocpd,nowrt       // configuração dos
fusíveis

#use   delay(clock=4000000, RESTART_WDT)
#use   rs232(baud=9600,xmit=pin_c6,rcv=pin_c7)

#define        t_filtro 500                          // tamanho do filtro

       float conversao = 0;          // armazena o resultado da conversão AD
  char Comando;

#use   fast_io(a)
#use   fast_io(b)
#use   fast_io(c)
#use   fast_io(d)
#use   fast_io(e)

#byte  porta = 0x05
#byte  portb = 0x06
#byte  portc = 0x07
#byte  portd = 0x08
#byte  porte = 0x09

       void main()
{
              // configura microcontrolador
              setup_adc_ports (RA0_RA1_RA3_analog);//Configura Vdd e Vss como
Vref
              setup_adc (adc_clock_div_32);
              setup_counters (rtcc_internal, WDT_2304MS);
              set_adc_channel (1);

              // configura os tris
              set_tris_a(0b11011111);                  // configuração da direção dos
pinos de I/O
              set_tris_b(0b00000011);
              set_tris_c(0b11111101);
              set_tris_d(0b00000000);
              set_tris_e(0b00000100);
```

```
                // inicializa os ports
                porta=0x00;                     // limpa porta
                portb=0x00;                     // limpa portb
                portc=0x00;                     // limpa portc
                portd=0x00;                     // limpa portd
                porte=0x00;                     // limpa porte


        while(TRUE)                     // rotina principal
        {
        RESTART_WDT();                          // incia o watch-dog timer
    Comando=getc();
    if (Comando=='d')
    {
       conversao = read_adc();          // inicia conversão AD
                conversao = (conversao * 5);   // faz regra de 3 para converter o
valor,
                conversao = (conversao / 1023); // das unidades de AD em Volts.
       printf("Voltimetro: %f\r\n", conversao);
       delay_ms(1000);
     }
    // printf("Voltimetro: %c\r\n", Comando);
    // delay_ms(3000);
        }

}
```




CEABRA
Prof: Marco A.
U3
MAX232
C1+
C1-
C2+
C2-
+C
-C
T1IN
R1OUT
T2IN
R2OUT
T1OUT
R1IN
T2OUT
R2IN
VCC
GND
+5V
1uF-16V
C6
C2
C7
C1
C5
100nF
R53
1K
RC7
RC6
DB9
RS232

```
/*****************************************************
*                    CEABRA
* Prof: Marco Antonio
* Exemplo 5: Uso dos LCDs
* Materia: Eletrônica Digital
* Data: Setembro 2005
*****************************************************/

#include       <16f877A.h> // microcontrolador utilizado

#fusesxt,wdt,noprotect,put,brownout,nolvp,nocpd,nowrt       // configuração dos
fusíveis

#use   delay(clock=4000000, RESTART_WDT)

#zero_ram

#use   fast_io(a)
#use   fast_io(b)
#use   fast_io(c)
#use   fast_io(d)
#use   fast_io(e)

#byte  porta = 0x05
#byte  portb = 0x06
#byte  portc = 0x07
#byte  portd = 0x08
#byte  porte = 0x09

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                      SAÍDAS                                         *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
#bit rs = porte.0                    // via do lcd que sinaliza recepção de dados ou
comando
#bit enable = porte.1                      // enable do lcd


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *              Rotina que envia um COMANDO para o LCD                   *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
void comando_lcd(int caracter)
{
        rs = 0;                            // seleciona o envio de um comando
        portd = caracter;                  // carrega o portd com o caracter
        enable = 1 ;                       // gera pulso no enable
        delay_us(1);                       // espera 3 microsegundos
        enable = 0;                        // desce o pino de enable

        delay_us(40);                             // espera mínimo 40 microsegundos
```

```
        return;                         // retorna
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *              Rotina que envia um DADO a ser escrito no LCD                 *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
void escreve_lcd(int caracter)
{
        rs = 1;                         // seleciona o envio de um comando
        portd = caracter;               // carrega o portd com o caracter
        enable = 1;                     // gera pulso no enable
        delay_us(1);                    // espera 3 microsegundos
        enable = 0;                     // desce o pino de enable

        delay_us(40);                           // espera mínimo 40 microsegundos

        return;                         // retorna
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                      Função para limpar o LCD                  *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
void limpa_lcd()
{
        comando_lcd(0x01);                      // limpa lcd
        delay_ms (2);
        return;
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                  Inicialização do Display de LCD                   *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

void inicializa_lcd()
{
        comando_lcd(0x30);                      // envia comando para inicializar
display
        delay_ms(4);                    // espera 4 milisengundos

        comando_lcd(0x30);                      // envia comando para inicializar
display
        delay_us(100);                          // espera 100 microsengundos

        comando_lcd(0x30);                      // envia comando para inicializar
display

        comando_lcd(0x38);                      // configura LCD, 8 bits, matriz de
7x5, 2 linhas

        limpa_lcd();                    // limpa lcd
```

```
        comando_lcd(0x0c);                      // display sem cursor

        comando_lcd(0x06);                      // desloca cursor para a direita

        return;                         // retorna
}


void main()
{
                // configura microcontrolador
                setup_adc_ports (no_analogs);
                setup_counters (rtcc_internal, WDT_2304MS);

                // configura os tris
                set_tris_a(0b11011111);                 // configuração da direção dos
pinos de I/O
                set_tris_b(0b00000011);
                set_tris_c(0b11111101);
                set_tris_d(0b00000000);
                set_tris_e(0b00000100);

                // inicializa os ports
                                                        // inicializa os ports
                porta=0x00;                             // limpa porta
                portb=0x00;                             // limpa portb
                portc=0x00;                             // limpa portc
                portd=0x00;                             // limpa portd
                porte=0x00;                             // limpa porte

                inicializa_lcd();                       // configura o lcd

        while(TRUE)                             // rotina principal
        {
      RESTART_WDT();                                    // incia o watch-dog timer
         comando_lcd(0x80);                             // posiciona o cursor na linha 0,
coluna 0
            printf (escreve_lcd, " Marco Antonio");     // imprime mensagem no lcd

      comando_lcd(0xC2);                        // posiciona o cursor na linha 1,
coluna 2
            printf (escreve_lcd, "  CEFETES ");         // imprime mensagem no lcd

        }
}
```