Curso em Linguagem C para microcontroladores PIC.
(Compilador CCS)
PIC16F877A e PIC16F887.

Unidade 01

ACEPIC Tecnologia e Treinamento LTDA.
Autor: Carlos Eduardo Sandrini Luz

Introdução

Este curso é baseado nos microcontroladores PIC16F877A e 16F887 da microchip (www.microchip.com), sendo utilizado o compilador em linguagem C CCS – PCWHD.

O compilador CCS módulo PCWHD é desenvolvido pela CCS Inc. (www.ccsinfo.com) e abrange toda a linha de microcontroladores PIC. Este compilador possui uma interface própria onde é possível criar projetos, editar o código fonte, compilar e, utilizando-se do Gravador e Depurador ICD-U64 (desenvolvido pela própria CCS), é possível a programação e depuração do microcontrolador diretamente pela interface.

Neste curso, utilizaremos para a programação e eventual depuração o gravador e depurador ACE ICD desenvolvido pela ACEPIC Tecnologia e Treinamento LTDA (www.acepic.com.br).

O compilador CCS também pode ser integrado ao MPLAB através de um software 'plug-in' que pode ser baixado sem custo através do link abaixo. Através desta integração é possível editar, compilar, programar e depurar os projetos diretamente na interface de desenvolvimento MPLAB, assim é possível a utilização de ferramentas que podem ser integradas a este software, como por exemplo, o gravador e depurador ACE ICD.

A CCS oferece uma versão de demonstração de seu compilador válida por um período de 45 dias sem limite no tamanho do programa e totalmente funcional para toda a linha de microcontroladores PIC. Para baixar a versão de demonstração basta seguir o link informado abaixo, lembrando que é necessário o preenchimento de um formulário de cadastro pelo usuário.

Neste curso, apresentaremos os exemplos feitos na interface PCWHD, porém em alguns casos utilizaremos a integração ao MPLAB.

Todos os exemplos de projetos criados, foram desenvolvidos e testados baseados nos Kits de desenvolvimento ACEPIC 40 V2.0 e ACEPIC PRO V2.1 também desenvolvidos pela ACEPIC Tecnologia e Treinamento

LTDA.

Apesar deste curso ter como base os PICs 16F877A e 16F887, as explicações e programas podem ser adaptadas facilmente para outros microcontroladores da microchip.

Plug-In de integração CCS – MPLAB:
http://www.ccsinfo.com/downloads.php
Versão de demonstração CCS – PCWHD:
http://www.ccsinfo.com/ccsfreedemo.php
MPLAB IDE:
http://www.microchip.com

Linguagem C

É uma linguagem de programação de propósito geral, estruturada, imperativa, procedural de alto e baixo nível, criada em 1972 por *Dennis Ritchie* no AT&T Bell Labs, para desenvolver o sistema operacional UNIX (que foi originalmente escrito em Assembly). Desde então, espalhou-se por muitos outros sistemas e tornou-se uma das linguagens de programação mais usadas influenciando muitas outras linguagens, especialmente o C++, que foi desenvolvida como uma extensão para C.

Fonte: WIKIPEDIA

Programação de Microcontroladores em linguagem C

Atualmente, a maioria dos microcontroladores existentes nos mercado, contam com compiladores em C para o desenvolvimento de software, pois a linguagem C permite a construção de programas e aplicações muito mais complexas do que o Assembly.
O compilador C tem a capacidade de "traduzir" com alto grau de inteligência e velocidade o código em C para o código de máquina, portanto podemos dizer que a linguagem C possui grande eficiência.
Essa eficiência da linguagem C faz com que o programador preocupe-se mais com a programação em si e o compilador assume responsabilidades como localização da memória, operações matemáticas e lógicas, verificação de bancos de memórias e outros..

Introdução à Linguagem C

Palavras Reservadas

Toda linguagem de programação possui um conjunto de palavras definidas para interpretação do próprio compilador, sendo assim, essas palavras não deverão ser utilizas pelo usuário para outras finalidades além das definidas pelo compilador.
Essas palavras são chamadas de Reservadas e em linguagem C, temos as seguintes:

auto	do	goto	signed	unsigned
break	double	if	sizeof	void
case	else	int	static	volatile
char	enum	long	struct	while
const	extern	register	switch	
continue	float	return	typedef	
default	for	short	union	

Identificadores

Os identificadores são nomes dados às funções, variáveis, constantes, etc e não devem conter caracteres acentuados, espaços, ‘ç’ e devem sempre começar com uma letra ou o símbolo ’_’ que é tratado como letra

Tipos de dados

Os tipos de dados suportados pela linguagem C e utilizados no compilador CCS são identificados pela palavras reservadas: **char, int, float** e **void.**

Tipo de dado	Tamanho
char	8 bits
Int	8 bits
float	32 bits
void	0

O tipo **char** representa caracteres ASCII de 8 bits, sendo que cada variável do tipo **char** pode representar somente um caracter ASCII.
O tipo **int** representa números inteiro de 8 bits.
O tipo **float** representa números fracionários de 32 bits, sendo que este tipo de dado deve ser evitado tendo em vista o seu tamanho.
O tipo **void** é normalmente utilizado em funções para declarar que ela não deve retornar nenhum valor.

Modificadores de Tipo

Além dos tipos de dados acima, podemos utilizar comandos especiais para modificá-los e são chamados de modificadores de tipo, sendo eles: **signed, unsigned, short** e **long.**

O modificador de tipo **signed** pode ser utilizado para representar um número positivo ou negativo, assim um tipo de dado **signed int** pode representar valores de -127 a 128.
O modificador **unsigned** define um tipo de dado sem sinal, ou seja, somente a parte positiva de uma variável, então o tipo de dados **unsigned int** representa valores de 0 à 255.
O modificador **short** é utilizada para definir um valor menor do que o

tipo modificado, ou seja, ao declaramos uma variável com o tipo **short int,** a variável tem seu tamanho reduzido para 1 bit.
O modificador **long** é o contrário, ou seja, amplia o tamanho de uma variável, então uma variável declarada com o tipo de dado long int passa a ter o tamanho de 16 bits.

Temos abaixo, uma tabela com todos os tipos de dados e seus modificadores disponíveis para o compilador CCS:

Tipo	Tamanho	Intervalo
short int, int1, boolean	1 bit	0 ou 1
char	8 bits	0 a 255
int, int8	8 bits	-128 a 127
signed int	8 bits	-128 a 127
unsigned int	8 bits	0 a 255
long int, int16	16 bits	-32768 a 32767
signed long int, signed int16	16 bits	-32768 a 32767
unsigned long int, unsigned int16	36 bits	0 a 65535
int32	32 bits	-2147483648 a 2147483647
signed int32	32 bits	-2147483648 a 2147483647
unsigned int32	32 bits	0 a 4294967295
float	32 bits	$1,5^{-45}$ a $3,4^{38}$

Variáveis

As variáveis são uma representação simbólica onde são armazenados dados do programa ou dados externos como uma tecla pressionada ou uma tensão lida, etc. As variáveis, como vimos acima, podem conter letras e números, sempre começando com letras e não devem ter nome de palavras reservadas pelo compilador como, por exemplo, for, do, int, etc.

Declaração de Variáveis

Declarar uma variável é simplesmente informar ao compilador que uma variável chamada "X" é do tipo "Y" e é declarada da seguinte forma:

<tipo> + <nome da variável>;

Podemos também declarar e inicializar uma variável da seguinte forma:

<tipo> + <nome da variável> = <valor da variável>;

Exemplos:
```
unsigned int x = 123;
int conta;
short x1;
```

Variáveis Globais

São declaradas no início de nosso código e que podem ser acessadas em qualquer ponto do programa:

Exemplo:

```
int conta;
unsigned int c;

void main()
{
conta = 10;
c = 2;

while(true);
}
```

Variáveis Locais

São declaradas dentro de uma função e somente existe durante a execução da função. Elas são descartadas depois de executada a função:

Exemplo:

```
void main()
{
int conta

conta = conta++;

while(true);
}
```

Operadores

Temos, na linguagem C, vários operadores e podemos verificá-los abaixo:

Operadores de Atribuição

São utilizados para atribuir valores às variáveis:

Operador	Descrição	Exemplo
=	Associa um valor à variável	a = 2

Aritméticos:

Operador	Descrição	Exemplo
+	Soma dos argumentos	a + b
-	Subtração dos argumentos	a - b
*	Multiplicação dos argumentos	a * b
/	Divisão dos argumentos	a / b
%	Resto da divisão (só pode ser utilizado com valores inteiros)	a % b
++	Soma 1 ao argumento	a++
--	Subtrai 1 ao argumento	a--

Relacionais:

São utilizados para comparação entre argumentos e retornam uma resposta verdadeira ou falsa. Como em linguagem C não existe uma variável booleana para um resultado verdadeiro ou falso, todo valor igual a 0 será considerado falso e todo valor diferente de 0 (qualquer valor) será considerado verdadeiro.

Operador	Descrição	Exemplo
==	Compara se igual a	a == 5
!=	Compara se diferente de	a != 5
>	Compara se maior que	a > 5
<	Compara se menor que	a < 5
>=	Compara se maior ou igual a	a >= 5
<=	Compara se menor ou igual a	a <= 5

Operadores lógicos bit-a-bit:

Operador	Descrição
&	E (AND)
|	OU (OR)
^	OU EXCLUSIVO (XOR)
~	Complemento (NOT)
>>	Deslocamento à direita
<<	Deslocamento à esquerda

Operadores lógicos relacionais:

Operador	Descrição
&&	Comparação lógica E (AND)
||	Comparação lógica OU (OR)
!	Comparação lógica Complemento (NOT)

Declarações de controle:

Comando *if*

O comando *if* é uma comando de decisão e é utilizado para avaliar uma determinada condição e determinar se ela é verdadeira, caso seja, executa o bloco contido dentro desta condição. Sua forma geral é:

```
if (exp) comando;
```

Se o resultado da condição referente a expressão (exp) for verdadeiro, o *comando* será executado, caso contrário, o programa segue sem executar o *comando*.

```
if (exp)
        {
         comando1;
         comando2;
         }
```

Para o caso acima, a mesma explicação anterior se encaixa, sendo que agora, se a condição da expressão for verdadeira, serão executados *comando1* e *comando2*.

Exemplos:

```
if (conta>50) conta = 0;
```

Neste caso, se a variável conta atingir um valor maior que 50, o comando *conta = 0* será executado e a variável conta será zerada.

```
if (conta>50)
                {
                conta = 0;
```

```
        conta1++;
        }
```

Neste caso, se a variável conta atingir um valor maior que 50, os comandos *conta = 0* e *conta1++* serão executados e assim a variável conta será zerada e a variável conta1 será incrementada em 1, respectivamente.

Comando *if-else*

Neste caso, a condição *if* é utilizada da mesma forma anterior, sendo que agora, se a condição da expressão for falsa a condição *else* será executada, ou seja, neste caso existe a possibilidade de escolha de uma entre duas opções. Sua forma é:

```
if (exp) comando1;
else comando2;
```

Caso a expressão seja verdadeira, o *comando1* será executado, caso seja falsa, o *comando2* será executado.

Exemplo:

```
if (conta>0)
        {
        conta = 0;
        conta1++;
        }
else conta++;
```

Para o exemplo, se o valor da variável conta for maior que 0, então os comandos *conta = 0* e *conta1++* serão executados, porém caso o valor da variável conta seja 0 ou menor que 0, então o comando *conta++* será executado.

Comando *switch-case*

Quando existem muitos valores para testar de uma só variável, o comando IF pode ficar meio confuso ou sem muita eficiência, para isso podemos utilizar o SWITCH-CASE. Segue sua forma:

```
switch(variável)
        {
        case valor1: comando1;
                        ....
                     break;
      case valor2: comando2;
                     ....
                     break;
                ....
                ....
      default:     comandoN;
                     ....
                     ....
      }
```

Neste caso, a *variável* será testada e se o valor dela for igual a *valor1*, o *comando1* será executado, se for igual ao *valor2*, o *comando2* será executado e assim por diante, agora se o valor for diferente de qualquer caso (case), o *comandoN* será executado.

Exemplo:

```
switch(conta)
      {
      case 10 : conta1++;
                break;
      case 15:  conta2++;
                break;
      case 20:  {
                conta1++;
                conta2++;
```

```
            }
            break;
default:  conta3++;
}
```

Neste caso, se o valor da variável conta for igual a 10, a variável *conta1* será incrementado, se o valor da variável conta for igual a 15, o valor da variável *conta2* será incrementado, caso o valor de conta seja igual a 20, tanto os valores de *conta1* quanto de *conta2* serão incrementados, para todos outros valores diferentes para a variável conta, o valor de *conta3* será incrementado.
Note que após cada comando, temos a cláusula *break*, cuja função é encerrar o teste da variável tendo em vista já ter sido satisfeita a condição, assim, por exemplo:

```
switch(conta)
        {
        case 10 : conta1++;
                  break;
                    .
                    .
                    .
```

Se a variável conta tem seu valor igual a 10, o comando de incremento da variável *conta1* será executado, ou seja, a condição já foi atendida e não é preciso testar mais vezes a variável conta. Então, a cláusula *break,* encerra os teste feitos por *case* e, assim, o programa continua na próxima instrução após a estrutura *switch.*

Laço *for*

Este é um dos comandos de laço (loop ou repetição) disponíveis na linguagem C, a seu formato é:

```
for(inicialização;condição(término);incremento)
    {
    comando1;
    comando2;
    }
```

onde:
inicialização: essa seção conterá uma inicialização para a variável;
condição: responsável por contar a condição de finalização do laço;
incremento: conterá a expressão para incremento da variável.

Exemplo:

```
int conta;
int a = 0;

for (conta=0;conta<10;conta++) a++;
```

Neste exemplo, a variável conta será iniciada com o valor 0, a variável *a* será incrementada e, como o resultado da expressão (*conta<10*) ainda é verdadeiro, a variável conta será incrementada (*conta++*), assim como novamente a variável *a*. Essa repetição ou laço se dará enquanto o resultado da expressão for verdadeiro.

Laço *while*

Neste laço, os comandos serão repetidos enquanto a expressão for verdadeira, sua forma é:

```
while (exp)
        {
        comando;
       }
```

Exemplo:

```
int x;
x = 0;

while(x<10) x++;
```

A programa ficará no laço de repetição *while,* enquanto a variável x for menor que 10 e o programa só continuará quando o valor de x for maior ou igual a 10.

Laço *do-while*

Este laço é uma variação do comando WHILE, sendo que neste caso o comando será executado antes de testar se a condição é verdadeira. Sua forma é:

```
do
   {
   comando;
  } while(exp);
```

O comando será executado pelo menos uma vez antes de verificar a condição da expressão.

Exemplo:

```
int x;
int y;

do
   {
   x++;
   } while(y!=1);
```

Notação numérica

Em linguagem C, podemos usar as 4 formas de representação numérica, decimal, binária, hexadecimal e octal, sendo as mais comuns somente as 3 primeiras.

Notação decimal: a representação desta notação é direta: **Ex.: x = 10;**

Notação binária: esta representação vem precedida de "*0b*" ou "*0B*", indicando a notação: **Ex.: x = 0B00001010;**

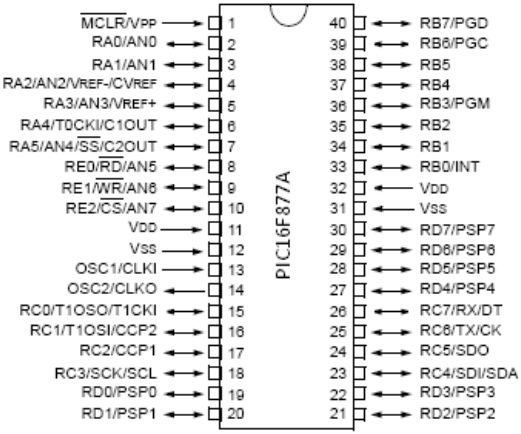
Notação Hexadecimal: esta representação vem precedida de "*0x*" ou "*0X*": **Ex.: x = 0x0A;**

Microcontroladores PIC – Família PIC16F

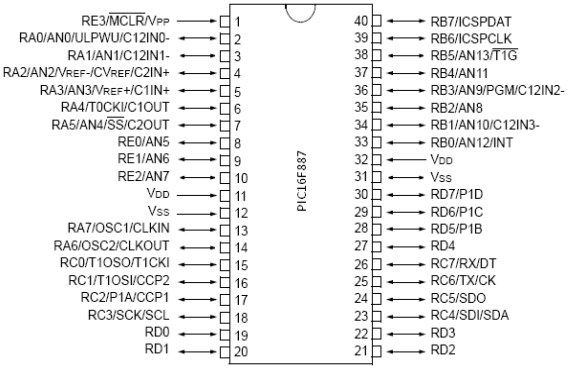
Características dos microcontroladores PIC16F877A e 16F887

Características	PIC16F877A	PIC16F887
Frequência de Operação	DC à 20MHz	DC à 20MHz
Memória de Programa	8192 bytes	8192 bytes
Memória de dados RAM	368 bytes	368bytes
Memória de dados EEPROM	256 bytes	256 bytes
Terminais de I/O	33	35
Timers	2 de 8 bits e 1 de 16 bits	2 de 8 bits e 1 de 16 bits
CCP	2	1
ECCP	0	1
Comunicação	SPI USART I2C	SPI EUSART I2C
Comparador	2	2
Conversor AD	8 canais de 10 bits cada	14 canais de 10 bits cada

Pinagem dos microcontroladores

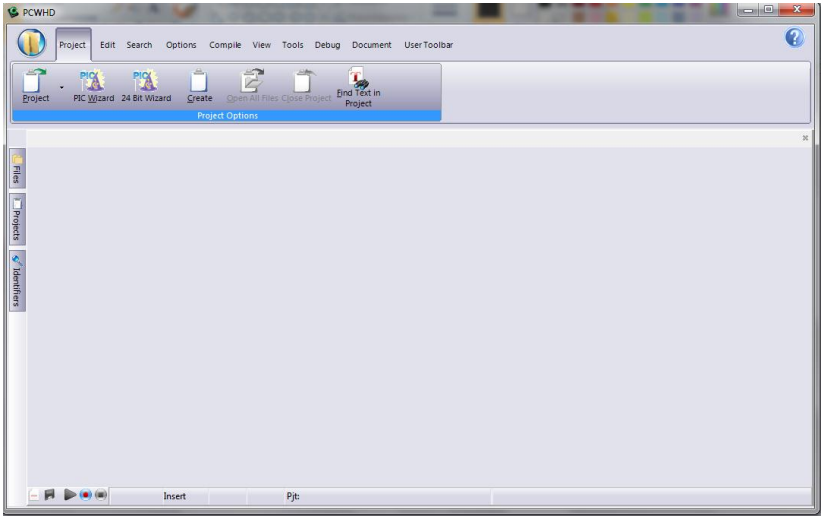


Pinagem do PIC16F877A



Pinagem do PIC16F887

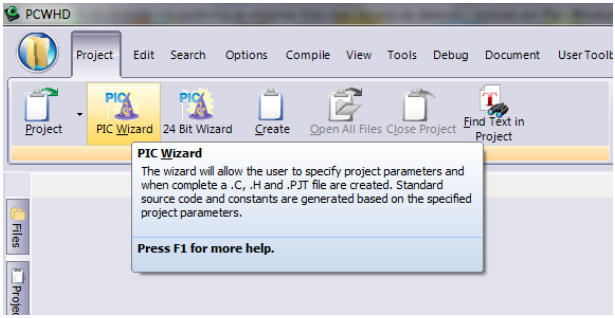
O compilador CCS – PCWHD



Interface de desenvolvimento do compilador CCS - PCWHD

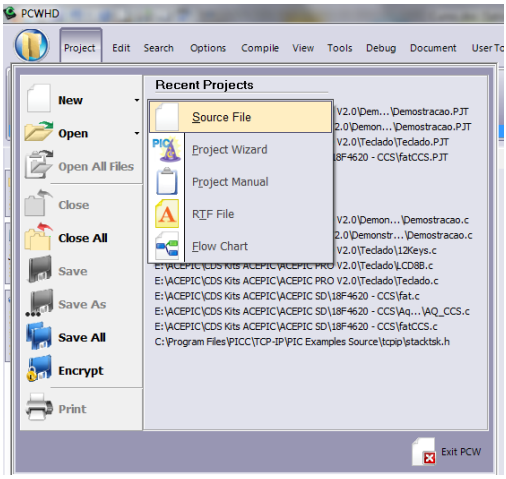
Criando um programa utilizando o compilador CCS

Podemos criar um projeto utilizando o 'PIC Wizard' que está no menu 'Project' da interface, conforme abaixo, porém criaremos somente o arquivo com extensão '*.c' e no MPLAB, montaremos um projeto.



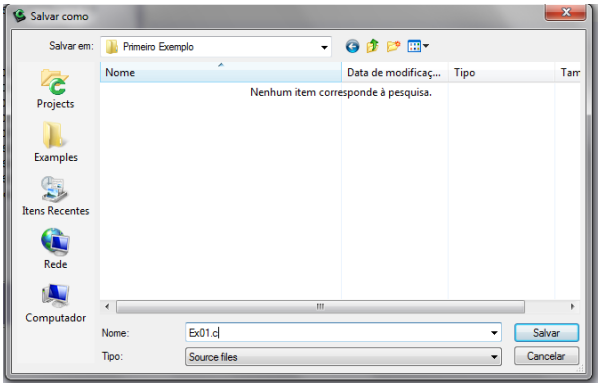
Utilizando o 'PIC Wizard' para criar um projeto no CCS

Para isso, clique sobre a pasta que se encontra ao lado esquerdo do menu 'Project', em seguida escolha a opção 'New' e na janela que se abre, escolha 'Source File', conforme segue:



Criando um arquivo '*.c'

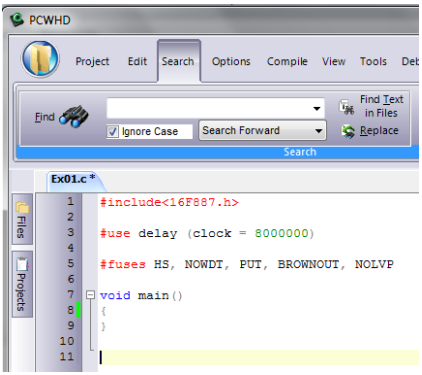
Escolha a pasta onde será armazenado o seu arquivo, dê um nome a ele com a extenção '.c', conforme abaixo e salve p arquivo.



Salvando o arquivo EX01.c

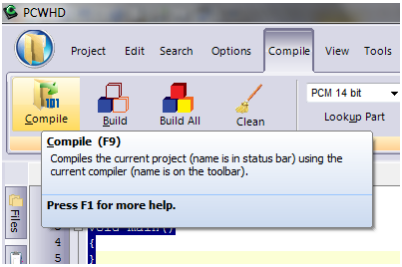
Após isso, teremos um editor de texto onde escreveremos o código do programa. Neste editor, digite o seguinte:

```
#include<16F887.h>

void main()
{
}
```



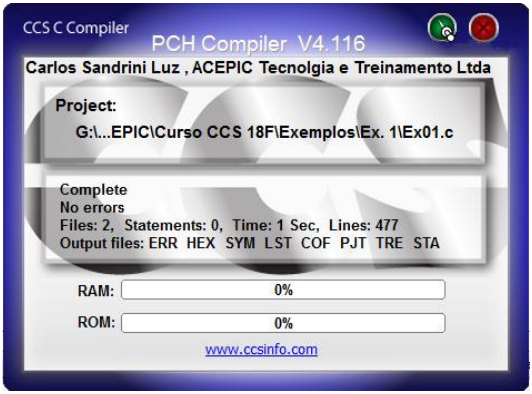
Escrevendo o código fonte

Após escrever o código acima, pressione o botão 'F9' do computador ou clique no menu 'Compiler' e em seguida no botão 'Compile' conforme a figura abaixo:



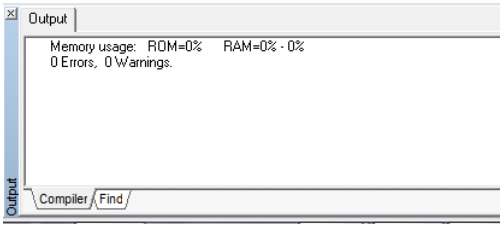
Compilando o programa

Com isso, o código será compilado:



Compilando o programa

Se não houver nenhum erro, a mensagem abaixo será mostrada na janela 'Output', de acordo com a figura abaixo:



Programa compilado sem erros

Note que foi utilizado apenas 2% da memória RAM e 0% da memória ROM.

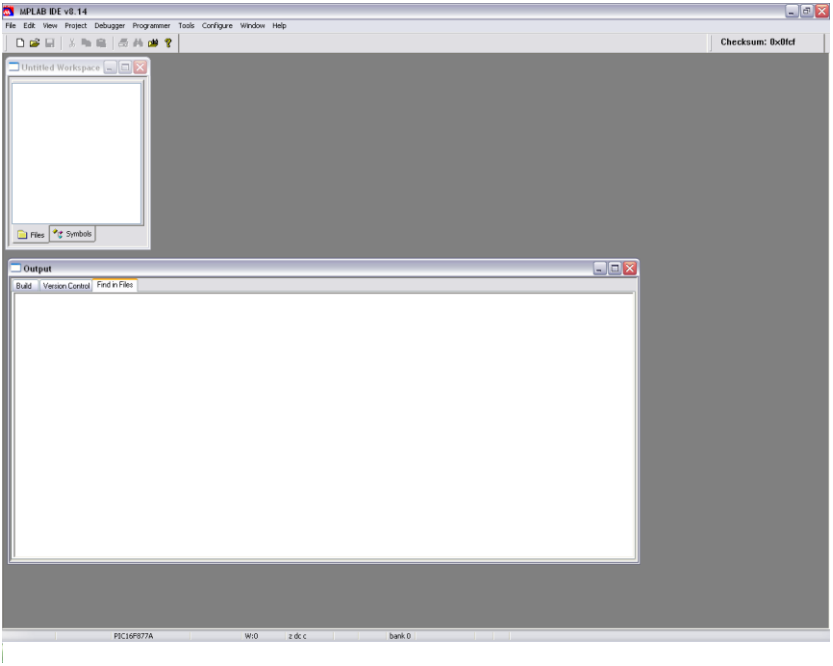
Bem, esse primeiro programa foi feito somente para verificarmos como podemos escrever um código fonte e compilá-lo diretamente na interface do compilador CCS – PCWHD.

Integrando o compilador CCS ao MPLAB

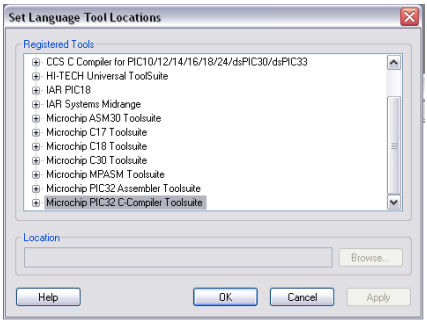
Após baixar e instalar o MPLAB IDE e após a instalação do compilador CCS, é necessário baixar e instalar o software 'Plug-In (MPLAB IDE PLUGIN)'.

Criando um projeto no MPLAB

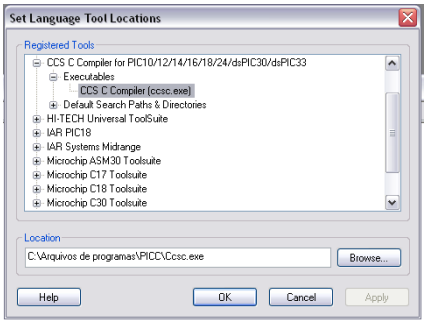
Ao abrirmos o programa MPLAB, esta será a tela de apresentação:



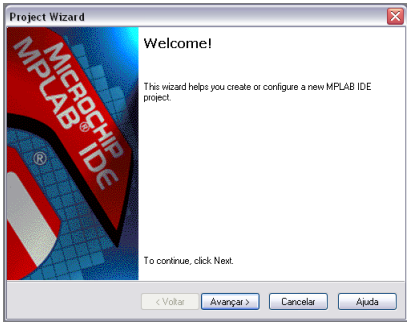
Primeiramente, vamos verificar se tudo está correto com a instalação do plug-in. Para isso, clique em 'Project' e em seguida em 'Set Language Tool Locations...'. A tela abaixo deverá ser mostrada:



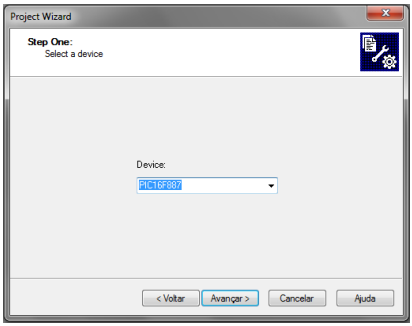
Clique no sinal '+' em CCS C Compiler for PIC10/12....., logo em seguida, clique no sinal '+' em 'Executables' e em 'CCS C Compiler(ccsc.exe)' e confira no campo 'Location' o caminho onde está instalado o compilador, conforme abaixo:



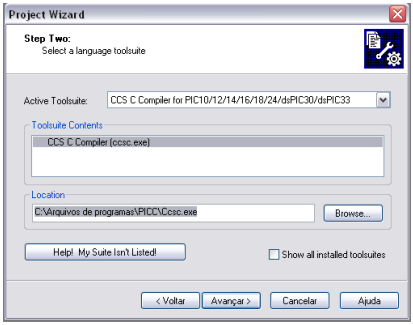
Estando tudo certo, vamos prosseguir com a criação do projeto.
Clique em 'Project' novamente e em seguida em 'Project Wizard' para que a próxima tela seja mostrada:



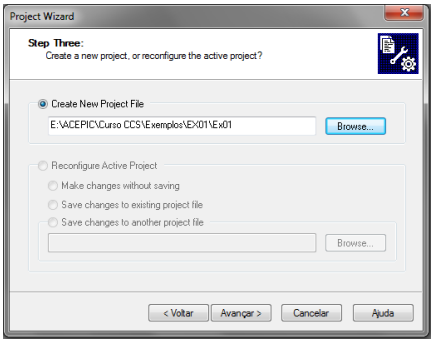
Nesta janela, clique em 'Avançar' e, na próxima tela, escolha em 'Device' o microcontrolador PIC16F887 e clique em 'Avançar'.



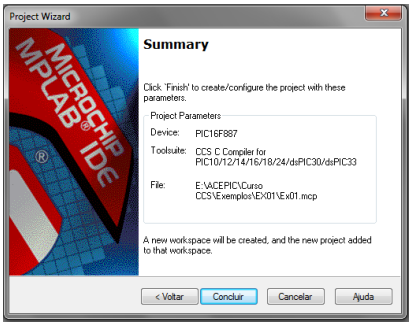
Na próxima tela, escolha, em 'Active toolsuite', a opção 'CCS C Compiler for.....', conforme mostradado na figura seguinte e clique em 'Avançar':



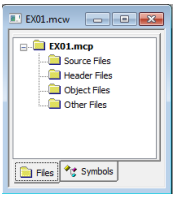
Na tela seguinte deverá ser informado o nome do novo projeto no campo 'Create New Project File'. Clique em 'Avançar'



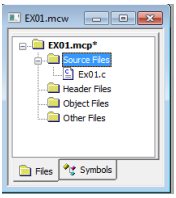
Na próxima tela (Step Four) não acrescentaremos nenhum arquivo ao projeto que estamos criando, então podemos clicar em 'Avançar'.
O 'Project Wizard' nos mostrará um resumo do que acabamos de configurar, conforme mostrado na próxima figura:



Basta clicar em concluir para que o projeto seja criado.
Agora, clique em 'View' e 'Project' para visualizarmos a janela do projeto, conforme abaixo:

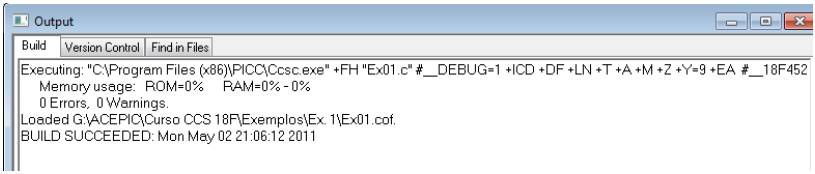


Agora, adcionaremos o arquivo Ex01.c ao projeto. Este arquivo é o mesmo criado anteriormente, na interface CCS. Para isto, clique com o botão direito do mouse em 'Source Files' na janela do projeto, no caso abaixo, 'EX01.mcp', e escolha a opção 'Add Files...'.
Na janela de busca que aparecerá, encontre o arquivo 'Ex01.c'. Veja na figura abaixo:



Se quiser, você pode salvar o projeto clicando em ‘Project’ e depois em ‘Save Project’...

Depois de criado o projeto e adicionado o arquivo a ele, vamos compilar este projeto clicando em ‘Project’ e depois em ‘Make’ ou simplesmente podemos pressionar a tecla ‘F10’. Toda vez que o projeto é compilado, os arquivos serão salvos.



Estrutura de um programa em C

Em linguagem C, os programas podem ter uma ou mais funções, isso faz com que tenhamos uma estrutura modular, ou seja, tenhamos blocos que podem ser acessados em qualquer parte do programa facilitando a visualização e o entendimento do programa.
Todo programa C tem uma função principal (main) e a partir dela o programa será inicializado.

Estrutura básica de um programa em C no compilador CCS:

```
#include<16F887.h>                    /*Inclusão do arquivo header (*.h)  para o
                                       microcontrolador utilizado*/
#use delay (clock=8000000)             //Definição da frequência do cristal para cálculo
                                       //..dos delays
#fuses HS, NOWDT, PUT,BROWNOUT, NOLVP  //Configuração dos bits de controle

void main()
{  //Inicio da função.

   //Este é o bloco principal do programa.

} //Final da função.
```

Note a função principal *void main()* e logo após, abrimos a inicialização da função com uma chave, logo após vem o corpo do programa (onde ele será escrito) e em seguida fechamos o programa com outra chave.
Note também as duas barras (//) antes das explicações. Essas barras iniciam um comentário e tudo que vier após estas barras não será 'entendido' pelo programa, ou seja, não será compilado.

Podemos também fazer comentários de outra maneira, veja a seguir:

```
/*Temos aqui uma outra forma de  comentar num programa. Esta maneira é indicada
quando precisamos escrever  algum comentário grande como este ou maior*/
```

Neste caso, o comentário tem seu início com '/*' e é finalizado com '*/'.

Antes de realizarmos nosso primeiro projeto, vamos verificar algumas diretivas:

Diretivas são comandos internos e que não são compilados, sendo estes dirigidos ao pré-processador e executados pelo compilador antes da execução do processo de compilação.

```
#include<16F887.h>                    //Inclusão do arquivo header (*.h)  para o
                                       //microcontrolador utilizado
#use delay (clock=8000000)             //Definição da frequência do cristal para cálculo
                                       //..dos delays
#fuses HS, NOWDT, PUT,BROWNOUT, NOLVP  //Configuração dos bits de controle
```

A primeira diretiva ***#include*** insere um arquivo ao programa ou ao projeto, no nosso caso, estamos incluindo o arquivo 16F887.h que é um arquivo tipo 'header' (*cabeçalho*) cuja finalidade é definir o endereço na memória de algumas funções e identificações para o microcontrolador PIC16F887. Este arquivo, geralmente, está localizado em '*C:\Arquivos de Programas\PICC\Devices*'.

A diretiva ***#use delay***, informa ao compilador a velocidade de clock do sistema, sendo necessária para as funções de atraso (**delay_us() e delay_ms()**). Este valor é especificado em Hertz .

Ex.: Para um cristal de 8MHz

```
#use delay(clock=8000000)
```

Bits de controle (fuses)

A diretiva ***#fuses***, informa a configuração dos bits de controle (conhecidos como fusíveis). Estes bits de controle são responsáveis por 'informar' ao microcontrolador como deverá ser o seu funcionamento, ou seja, que tipo de oscilador, se terá algum tipo de proteção ou não, algumas opções de 'reset' e temporização, etc.

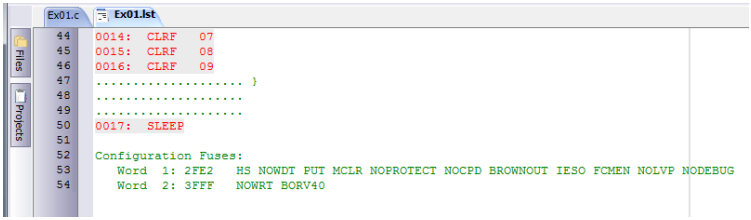
No exemplo acima, temos a seguinte configuração:

HS = Oscilador HS (oscilador a cristal >= 4MHz);
NOWDT = Watchdog desabilitado
PUT = Power Up Time habilitado
BROWNOUT = Brownout reset habilitado
NOLVP = Desabilitada programação por baixa tensão

Como podemos notar, temos somente 5 bits de controle configurados, porém existem mais bits de controle que precisam ser configurados para que o funcionamento do microcontrolador seja adequado.

Na realidade, o compilador CCS já possui uma pré configuração destes bits de controle e podemos vê-las clicando no menu 'Compile' e pressionando-se o botão 'C/ASM List'.

Esta opção nos mostrará o código que compilamos em 'Assembly' e no final deste arquivo, temos a informação 'Configuration Fuses', conforme a figura a seguir:



Configurações dos bits de controle – fuses

Verifica-se, através das configurações mostradas na figura que agora temos todos os bits de controle configurados e também que as configurações que fizemos pela diretiva *fuses* se mantém e as que não foram feitas pelo programador, serão feitas automaticamente pelo compilador mantendo um padrão inicial.
Para ver o padrão utilizado pelo compilador, apenas comente a linha da diretiva *fuses*, compile novamente o arquivo e verifique o código em 'assembly'.

Veja todas as opções de configuração dos bits de controle para os microcontroladores PIC16F877A e PIC16F887 a seguir:

Bits de configuração para o PIC 16F877A

Fuse	Descrição
LP	Oscilador LP (< 200 KHz)
RC	Oscilador RC
XT	Oscilador XT (Oscilador a cristal <= 4MHz)
HS	Oscilador HS (oscilador a cristal >= 4MHz
NOWDT	Watchdog desabilitado
WDT	Watchdog habilitado
NOPUT	Temporizador Power-up desligado
PUT	Temporizador Power-up ligado
NOPROTECT	Proteção de código desabilitada
PROTECT	Proteção de código habilitada
NOBROWNOUT	Reset por queda de tensão desabilitado
BROWNOUT	Reset por queda de tensão habilitado
NOLVP	Programação em baixa tensão desabilitada

LVP	Programação em baixa tensão habilitada.
NOCPD	Proteção da EEPROM habilitada
CPD	Proteção da EEPROM desabilitada
WRT_5%	Proteção dos primeiros 255 bytes da memória de programa habilitada
WRT_25%	Proteção de ¼ dos bytes totais da memória de programa habilitada
WRT_50%	Proteção de ½ dos bytes totais da memória de programa habilitada
NOWRT	Proteção da memória de programa desabilitada

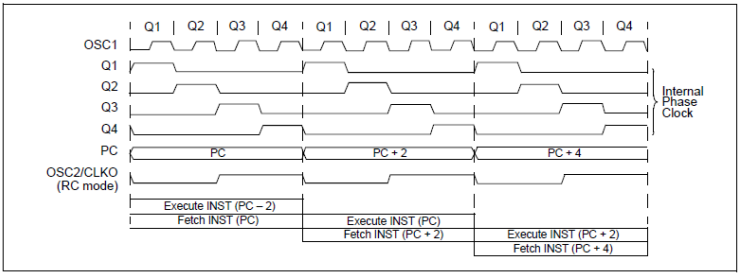
Bits de configuração para o PIC 16F887

Fuse	Descrição
LP	Oscilador LP (< 200 KHz)
XT	Oscilador XT (Oscilador a cristal <= 4MHz)
HS	Oscilador HS (oscilador a cristal >= 4MHz)
EC_IO	Clock externo
INTRC_IO	Oscilador RC interno, sem CLKOUT
INTRC	Oscilador RC interno
RC_IO	Oscilador RC com CLKOUT
RC	Oscilador RC (Resistor/Capacitor)
NOWDT	Watchdog desabilitado
WDT	Watchdog habilitado
NOPUT	Temporizador Power-up desligado
PUT	Temporizador Power-up ligado
NOMCLR	Master Clear desabilitado, pino 1 configurado como entrada (RE3)
MCLR	Master Clear habilitado
NOPROTECT	Proteção de código desabilitada
PROTECT	Proteção de código habilitada
NOBROWNOUT	Reset por queda de tensão desabilitado
BROWNOUT	Reset por queda de tensão habilitado
BROWNOUT_NOSL	Reset por queda de tensão habilitado durante a operação e desabilitado durante o modo SLEEP
BROWNOUT_SW	Reset por queda de tensão controlado via software
BORV21	Reset por queda de tensão abaixo de 2,1V
BORV40	Reset por queda de tensão abaixo de 4,0V
NOLVP	Programação em baixa tensão desabilitada
LVP	Programação em baixa tensão habilitada.
NOCPD	Proteção da EEPROM habilitada

CPD	Proteção da EEPROM desabilitada
NOIESO	Mudança do oscilador interno/externo desabilidada
IESO	Mudança do oscilador interno/externo habilitada
NOFCMEN	Desabilita o monitoramento do sinal de clock
FCMEN	Habilita o monitoramento do sinal de clock
NOCPD	Código na EEPROM não protegido
CPD	Código na EEPROM protegido
NOWRT	Proteção da memória de programa desabilitada
WRT	Proteção da memória de programa habilitada
DEBUG	Depuração pelo ICD habilitada

Ciclo de Máquina

A entrada de clock para o microcontrolador, seja ela interna ou externa, é dividida internamente por 4, gerando, assim, 4 fases (Q1, Q2, Q3 e Q4). Internamente, o contador de programa (program counter – PC) é incrementado na fase Q1 onde a instrução é localizada na memória de programa e carregada no registro de instrução na fase Q4, então a instrução é decodificada e executada durante os ciclos Q2, Q3 e Q4.



Conforme podemos ver acima, cada ciclo de máquina executa duas funções ao mesmo tempo, ou seja, ao mesmo tempo em que uma instrução é executada, a próxima instrução é localizada e carregada no registro de instrução. A essa tecnologia é dado o nome de PIPELINE e que resulta no aumento da velocidade de processamento.

Sendo assim, para obtermos a freqüência para o cálculo do ciclo, devemos então, dividir o sinal de entrada de clock interno ou externo por 4.

Por exemplo:
Supondo que contamos com um cristal de 20MHz como fonte de clock para o microcontrolador, então teremos:

Frequência de operação:

$$Freq = \frac{20MHz}{4} = 5MHz$$

Ciclo de máquina:

$$Ciclo = \frac{1}{Freq} = \frac{1}{5MHz} = 0,2us$$

Portas de I/O

Nos PICs 16F877A e 16F887 contamos com 5 portas (A, B, C, D e E) e alguns pinos dessas portas podem ter outras funções. De modo geral, quando o pino é utilizado para outra função, este não poderá ser utilizado com pino de I/O.
Cada porta tem 3 registradores para sua correta operação:
- **Registrador TRIS**, que é o responsável pelo direcionamento (entrada ou saída) das portas, onde se um bit estiver em 0, o pino referente será um pino de saída e se o bit estiver em 1, o pino referente será um pino de entrada;
- **Registrador PORT**, responsável por escrever ou ler o nível dos pinos associados à porta;

Registrador TRIS

O registrador TRIS, como já vimos, é o responsável pelo direcionamento no sentido do fluxo de dados de uma determinada porta. Esse registrador possui 8 bits, sendo cada bit correspondente a um determinado pino de I/O.
Podemos identificar os registradores de acesso como TRISA, TRISB, TRISC, TRISD e TRISE.

O compilador CCS, por padrão, faz automaticamente o direcionamento dos pinos dos registradores TRIS através das funções de saída ou de entrada, porém o programador também pode desabilitar esta forma de configuração e fazer, via programação, o direcionamento dos pinos.

Diretivas de direcionamento dos pinos

Para que possamos fazer a configuração dos registradores TRIS no compilador CCS, temos três possibilidades através das diretivas abaixo:

#USE STANDARD_IO(*port*)

Onde ***port*** pode ser A, B, C, D ou E.

Esta é a configuração padrão do compilador e não é necessária sua utilização a não ser que seja necessário voltar à configuração padrão após o uso de uma das diretivas de direcionamento abaixo.
Com esta diretiva, o direcionamento dos pinos serão de acordo com a utilização das funções de entrada e saída do próprio compilador.

#USE FAST_IO(*port*)

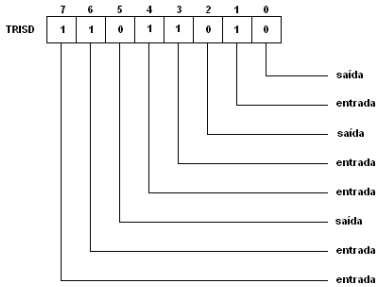
Onde ***port*** pode ser A, B, C, D ou E.

Com o uso desta diretiva, o programador faz a configuração dos pinos de entrada e saída de acordo com o desejado, ou seja, o direcionamento deixa de ser automático.
Para o direcionamento dos pinos através da programação deve ser utilizada uma função específica para este fim conforme de utilização abaixo:

```
#USE FAST_IO(D)          /*O direcionamento dos pinos da porta D serão via
                         ...programação*/

void main()
```

```
{

SET_TRIS_D(0B11011010);   /*os pinos 0, 2 e 5 da porta D serão configurados como
                          ... pinos de saída e os demais pinos serão configurados
                          ... como pinos de entrada*/

while(true);
}
```

Veja na próxima figura a representação dos bits deste registrador.



#USE FIXED_IO(*port_output=pin,pin?*)

Onde ***port*** pode ser A, B, C, D ou E e ***pin*** é o pino de entrada/saída a ser utilizado.

Esta diretiva faz com que o pino seja configurado como entrada ou saída de acordo com a utilização das funções OUTPUT_X(value) e INPUT_X(value), ou seja, cada vez que uma das funções for encontrada o direcionamento do(s) pino(s) informado(s) na diretiva será alterado para entrada ou saída.

Registrador PORT

O registrador PORT, assim como informado antes, é o responsável por informar o estado dos pinos de uma porta. Este registrador também é

de 8 bits e através das funções de entrada e saída do compilador é possível escrever ou ler os pinos de uma determinada porta.

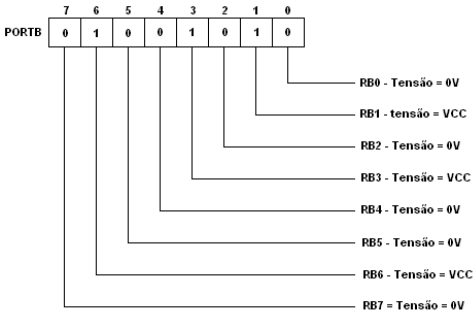
Função de entrada

```
x = INPUT_B();                    //Lê os pinos da porta B e armazena em x
```

Funções de saída

```
OUTPUT_B(0b01001010);             //Aciona os pinos da porta B
```

É possível verificarmos os níveis de tensão na porta D conforme mostra a figura abaixo:



Caso seja desejado alterar somente um bit de uma determinada porta para nível lógico ‘0’ ou ‘1’, podemos utilizar a seguinte função:

```
OUTPUT_BIT(PIN_RB0,0);    //O pino 0 da porta B terá nível lógico  ‘0’
```

Para levar um determinado pino de uma determinada porta ao nível lógico ‘1’, utilizamos a seguinte função:

```
OUTPUT_HIGH(PIN_B1);      //O pino 1 da porta B terá nível lógico ‘1’
```

A seguinte função tem a finalidade de levar para o nível lógio ‘0’, um somente um pino de uma determinada porta:

```
OUTPUT_LOW(PIN_B2);      //O pino 2 da porta B terá nível lógico ‘0’
```

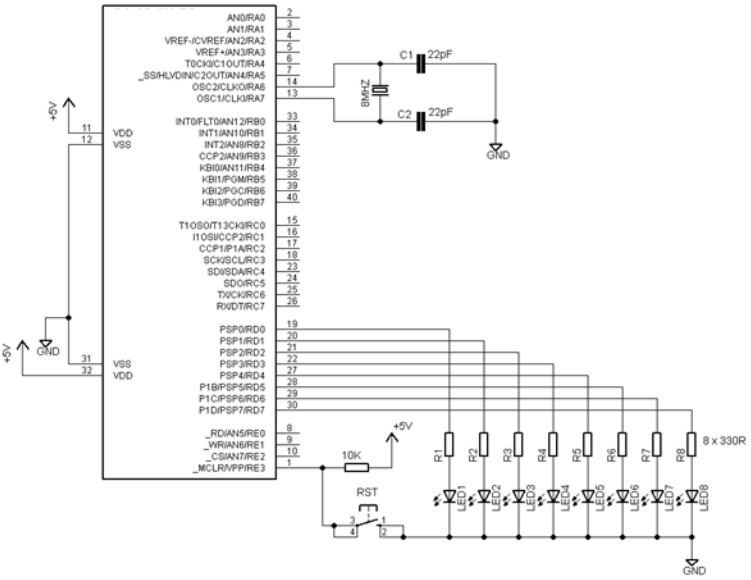
Nesta próxima função, será invertido o nível lógico de um pino de uma determinada porta, ou seja, o pino estando em ‘1’ vai para ‘0’ e estando em ‘0’ vai para ‘1’.

```
OUTPUT_TOGGLE(PIN_B3); /*Inverte o estado do pino 3 da porta B, ou seja, se tiver
                       ...em nível lógico ‘1’ muda para nível lógico ‘0’ e vice-
                       ...versa*/
```

Primeiro Programa – Acionamento de LEDS

Para o primeiro programa, vamos utilizar o circuito abaixo, onde faremos os leds conectados à PORTA D piscarem em intervalo de 500ms.

Circuito:



Circuito piscaled

Código fonte do arquivo EX02.c:

Antes de digitarmos o código, precisamos fechar todos o programas anteriores através da opção 'Close all' na pastinha da interface CCS. Abra um novo código fonte, conforme explicado anteriormente.

```
#include<16F887.h>                    //Inclusão do arquivo header (*.h)  para o
                                      //microcontrolador utilizado
#use delay (clock=8000000)            //Definição da frequência do cristal para cálculo
                                      //..dos delays
#fuses HS, NOWDT, PUT,BROWNOUT, NOLVP  //Configuração dos bits de controle

#use fast_io(D)                 /*Direcionamento dos pinos da porta D serão feitos
                                pelo programador*/

void main()
{
set_tris_d(0b00000000);         //Direciona todos os pinos da porta D como saída
output_D(0b00000000);           //Apaga todos os Led's da Porta D

while (true)                            //Loop principal
     {
          output_D(0b00000000);         //Apaga todos os Led's da Porta D
          delay_ms(500);                //Atraso de 500ms
          output_D(0b11111111);         //Acende todos os Led's da Porta D
          delay_ms(500);                //Atraso de 500ms
     }
}
```

Digitando o código acima no editor da interface CCS, podemos agora compilar para programar o microcontrolador e verificar o funcionamento.

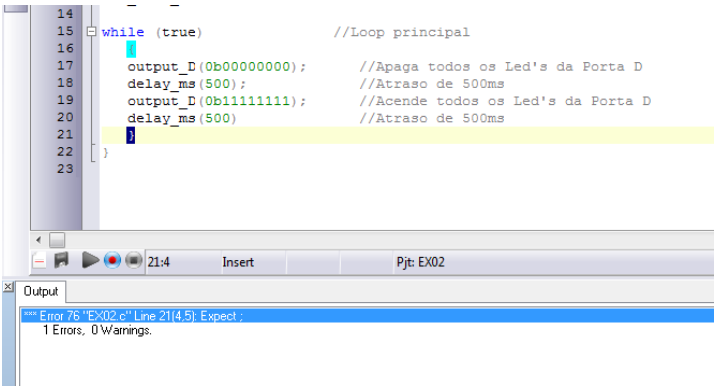
Para compilar o código, clique em 'Compile' no menu principal e logo em seguida no botão 'Compile', conforme mostrado na figura a seguir. Podemos também utilizar a tecla 'F9' do computador que é a tecla de atalho para o botão 'Compile'.

Ex02.c

```
1  #include<16F887.h>                //Inclusão do arquivo header (*.h)  para o
2                                    //microcontrolador utilizado
3  #use delay (clock=8M)             //Definição da frequência do cristal para cálculo
4                                    //..dos delays
5  #fuses HS, NOWDT, PUT,BROWNOUT, NOLVP  //Configuração dos bits de controle
6
7
8  #use fast_io(D)
9
10 void main()
11 {
12 set_tris_d(0b00000000); //Direciona todos os pinos da Porta D como saída
13 output_d(0b00000000);   //Apaga todos os leds conectados à porta D
14
15 while(true)
16    {
17    output_d(0b11111111); //Acendemos todos os leds da porta D
18    delay_ms(500);        //Atraso de 500ms
19    output_d(0b00000000); //Apagamos todos os leds da porta D
20    delay_ms(500);
21    }
22 }
```

Digitação do código fonte

Uma mensagem de erro será mostrada na janela 'Output' se algum erro acontecer, tal como uma palavra errada ou uma variável que não foi declarada ou até mesmo a falta do ';'.
Veja abaixo um exemplo onde é mostrado uma mensagem de erro e também informado qual o erro ocorrido, neste caso, faltou fechar com ';' a última linha do programa (delay_ms(500)). Basta-nos corrigir o problema e compilar novamente o código.

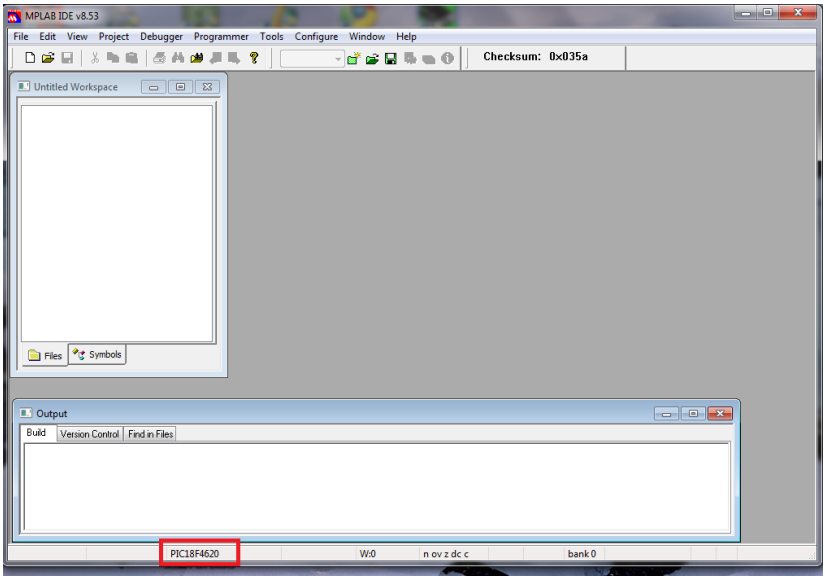


Apresentação de mensagens de erro

Gravando o microcontrolador

Quando o programa é compilado, será gerado um arquivo chamado 'hexa' com a extensão '*.hex' (no caso do código acima, será gerado o arquivo EX02.hex) e é este arquivo que utilizaremos para programar o microcontrolador.
Existem vários esquemas de gravadores e softwares de programadores livres na internet, os próprios kits que utilizamos possuem gravadores 'on-board' e funcionam pela porta serial do computador. Porém neste curso, como dito anteriormente, faremos uso da ferramenta de gravação e depuração ACE ICD que tem as mesmas funções do gravador e depurador ICD2 da microchip, sendo assim este será integrado ao MPLAB e assim facilitará a programação do microcontrolador.

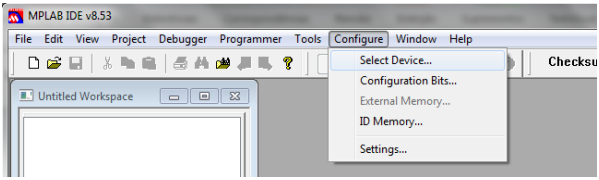
Abriremos a interface MPLAB e primeiramente, devemos verificar se o microcontrolador selecionado e apresentado na barra de status localizada na parte de baixo da interface é o mesmo microcontrolador que vamos programar. Veja na figura abaixo o microcontrolador informado em destaque



Microcontrolador PIC18F4620 selecionado

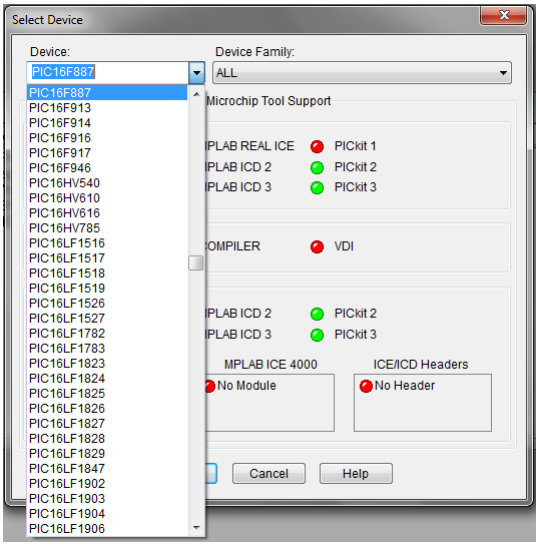
Caso não esteja selecionado o microcontrolador que desejamos, então será necessário fazer esta seleção.

Para isso, clique no menu superior em 'Configure' e em seguida em 'Select Device', conforme mostra a próxima figura.



Selecionando o microcontrolador

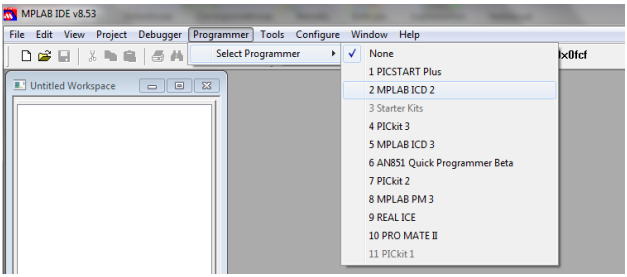
Na janela de seleção que se abrirá, selecione o microcontrolador PIC 16F887.



Selecionando o PIC 16F887

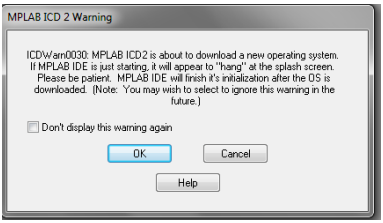
Após, a seleção, clique sobre o botão ‘OK’.
Note que agora, na barra de status, o microcontrolador PIC16F887 será mostrado.

Agora, conectaremos o gravador e depurador à uma porta USB no computador e, através do cabo ICSP, conectaremos o gravador ao kit.
Após isso, selecionaremos no MPLAB o programador ICD2, selecionando no menu ‘Programmer’ e em seguida clicando em ‘MPLAB ICD2’, conforme a próxima figura.



Selecionando o programador ICD2

Se o programador foi utilizado anteriormente para programar ou depurar um microcontrolador de outra família ou com características muito diferentes do PIC 16F887, pode acontecer a mensagem mostrada.

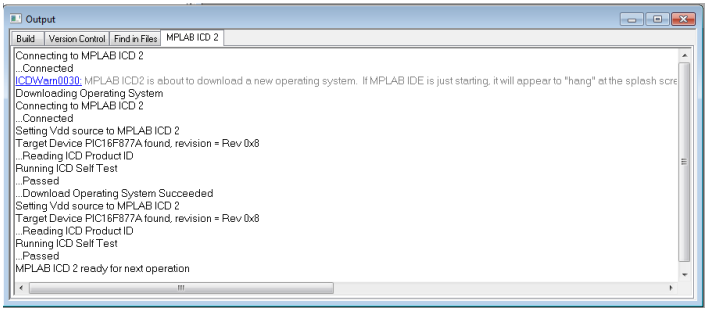


Mensagem de informação de download de novo sistema de operação

Esta mensagem informa que será feito um download no programador de um novo sistema de operação e que isso poderá levar alguns segundos, portanto, basta clicar no botão ‘OK’ e aguardar o término da operação.

Ao finalizar, a janela ‘Output’ mostrará as informações do microcontrolador selecionado e também a mensagem que o programador está pronto para próxima operação.

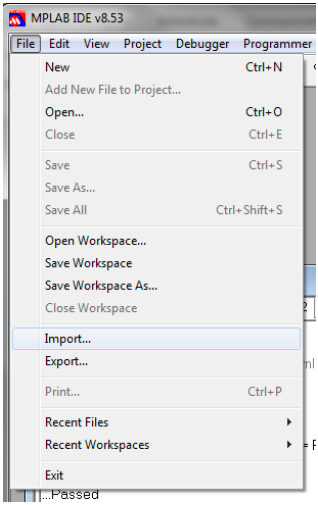
Veja na figura a seguir a informação mostrada.



Microcontrolador reconhecido e programador pronto para próxima operação

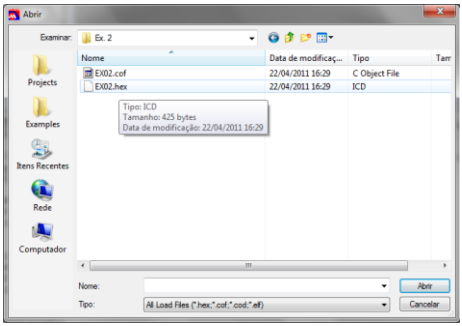
Agora, importaremos o arquivo hexa que o compilador gerou, programaremos o microcontrolador e verificaremos o funcionamento.

No menu, clique em 'File' e logo em seguida escolha a opção 'Import...'.



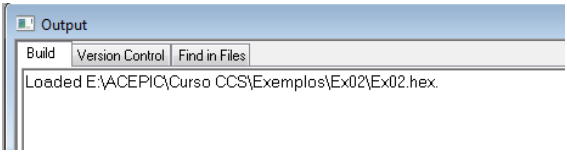
Importando o arquivo hexa

Procure o arquivo compilado e clique em 'Abrir', conforme a figura abaixo:



Abrindo o arquivo hexa

Note que o caminho do arquivo hexa aberto será mostrado na janela 'Output' do MPLAB.



Janela 'Output' com o caminho do arquivo hexa

Note também que serão habilitados os botões para programação do microcontrolador, conforme seguem:

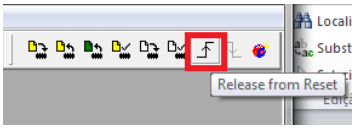
Botões de programação

Basta-nos programar o microcontrolador pressionando o botão 'Program target device', conforme mostrado em destaque na figura a seguir:

Botão de programação do microcontrolador

O resultado da programação também será mostrado na janela 'Output'.

Para que o programa 'rode' sem a necessidade de desconectar o programador, basta pressionar o botão 'Release from reset', conforme a próxima figura:



Resetando o microcontrolador

Entendendo o código fonte

Primeiramente, no código fonte, temos o cabeçalho, ou seja, a inclusão do arquivo 'header', as informações de frequência, as configurações dos bits de controle para o PIC 16F887 e finalmente a diretiva de direcionamento da porta D.

```
#include<16F887.h>                    //Inclusão do arquivo header (*.h)  para o
                                      //microcontrolador utilizado
#use delay (clock=8000000)            //Definição da frequência do cristal para cálculo
                                      //..dos delays
#fuses HS, NOWDT, PUT,BROWNOUT, NOLVP  //Configuração dos bits de controle

#use fast_io(D)              /*Direcionamento dos pinos da porta D serão feitos
                             pelo programador*/
```

Na função principal (main), logo ao iniciarmos, temos a função '*set_tris_d(0b00000000);*' que fará o direcionamento dos pinos da porta D, neste caso, todos os pinos serão configurados como saída. Seguindo, temos o laço 'while' e devido a este laço, conforme visto na introdução à linguagem C, o programa ficará repetindo os comandos dentro do seu bloco infinitamente.

```
void main()
{
set_tris_d(0b00000000);      //Direciona todos os pinos da porta D como saída

while (true)                          //Loop principal
     {
          output_D(0b00000000);       //Apaga todos os Led's da Porta D
          delay_ms(500);              //Atraso de 1000ms (1 segundo)
          output_D(0b11111111);       //Acende todos os Led's da Porta D
          delay_ms(500);              //Atraso de 1000ms (1 segundo)
     }
}
```

Neste caso, através da função ‘output_d(0b00000000)’, faremos com que toda a porta D tenha nível lógico ‘0’, ou seja, de acordo com o esquema, todos os Leds estarão apagados.
Em seguida, utilizamos a função de atraso ‘delay_ms(500)’ que fará com que seja gerado um atraso de 500ms no programa para que a próxima função, ‘output_D(0b11111111)’, faça com que todos os pinos da porta D passem a ter nível lógico ‘1’, assim acendendo todos os Leds.
Após essa função, temos mais uma função de atraso de 500ms e fechamos o laço ‘while’ fechando a chave do bloco e assim sendo, o programa executará estas 4 funções infinitamente.

Funções de atraso (delay)

São 3 as funções de atraso disponíveis no compilador CCS.

delay_ms(*tempo*);

Esta função gera um atraso no programa de 1 milisegundo vezes a variável tempo, onde ***tempo*** é uma variável do tipo **int16**, e assim sendo, pode assumir valores de 0 a 65535.

Ex.:

```
delay_ms(1000);    //Gera um atraso de 1 x 1000 milisegundos, ou seja, 1 segundo
```

delay_us(*tempo*);

Esta função gera um atraso no programa de 1 microsegundo vezes a variável tempo, onde ***tempo*** é uma variável do tipo **int16**, e assim sendo, pode assumir valores de 0 a 65535.

Ex.:

```
delay_us(1000);          //Gera um atraso de 1 x 1000 microsegundos, ou seja, 1
                         //milisegundo
```

delay_cycles(*valor*);

Esta função gera um atraso de ciclos de máquina de acordo com a variável ***valor***, sendo esta variável do tipo **constante**, sendo que pode assumir valores de 1 a 255.

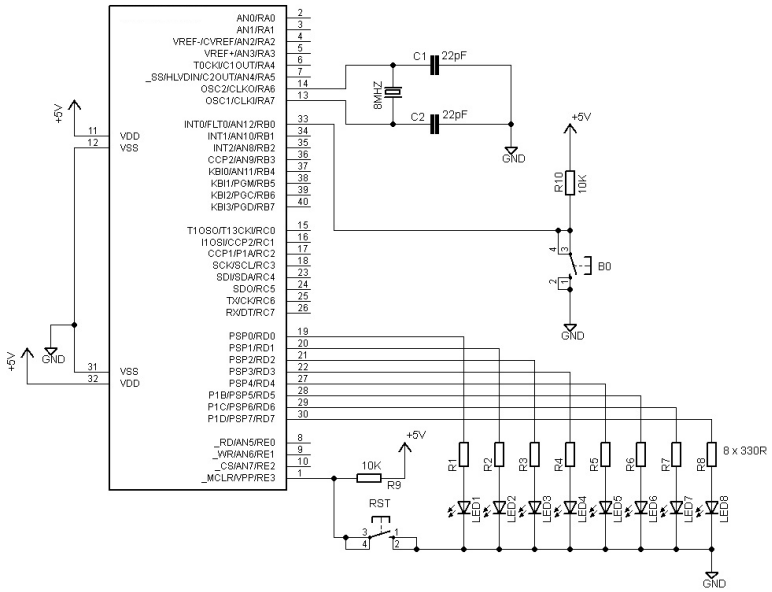
Ex.:

```
delay_cycles(10);    //Gera um atraso de 10 ciclos de máquina
```

Acionamento de Botões

Neste próximo projeto, faremos a leitura do pino 0 da porta B (RB0) e quando este for
igual a 0, acionaremos o bit 0 da porta D fazendo acender o LED L1 conectado a este pino e, caso o nível lógico no pino RB0 seja 0 novamente, apagaremos o LED L1. Assim, cada vez que pressionarmos o botão B0, mudaremos o estado do LED L1.
Veja no circuito apresentado que o pino RB0 está inicialmente em nível lógico 1 e, ao pressionarmos o botão B0, este pino deverá ser levado para o nível 0.

Circuito:



Circuito acionamento de botões

Código fonte arquivo Ex03.c

```
#include<16F887.h>          /*Inclusão do arquivo header (*.h)  para o
                            microcontrolador utilizado*/
#use delay (clock=8000000)  /*Definição da frequência do cristal para cálculo
                            ..dos delays*/
#fuses HS, NOWDT, PUT,BROWNOUT, NOLVP  //Configuração dos bits de controle


#use fast_io(B)             /*Direcionamento dos pinos da porta B serão feitos
                            pelo programador*/
#use fast_io(D)             /*Direcionamento dos pinos da porta D serão feitos
                            pelo programador*/

void main()
{
set_tris_b(0b00000001);     /*Somente o pino 0 da porta B será direcionado como
                            entrada*/

set_tris_d(0b00000000);     //Todos os pinos da porta D serão saída

output_D(0b00000000);       //Coloca em 0 toda a porta D

while(true)
      {
      if (input(PIN_B0)==0)             //Se o botão for pressionado
        {
        output_toggle(PIN_D0);          //Muda o estado do pino D0 (LED L1)
        delay_ms(300);                  //Atraso de 300ms
        }
      }
}
```

Note que neste código, somente verificamos o pino 0 da porta B através da declaração de controle ***if*** e, quando o nível de sinal no pino for '0' a condição (expressão ***input(PIN_B0)==0***) será verdadeira, assim serão executados os comandos (funções) contidos no bloco da declaração *if*.

A primeira função no bloco da declaração *if* (***output_Toggle(PIN_D0);***

), fará a inversão lógica de ‘1’ para ‘0’ e vice-versa somente do pino 0 da porta D e logo em seguida temos a função de atraso de 300ms para evitar o chamando ‘debounce’ (ruído elétrico da chave).

Agora, basta compilar o projeto e programar o microcontrolador com o arquivo hexadecimal gerado para verificar o funcionamento.

Diretiva *#define*

Podemos atribuir um nome a uma constante ou a um registrador usados com frequência num programa através da diretiva *#define* conforme segue abaixo:

Exemplos:

```
#define B0 PIN_B0          /*Informa ao compilador que o valor B0 será substituído
                           ...por PIN_B0 toda vez que este for utilizado */

#define MAX 125            /*Informa ao compilador que a constante MAX, toda vez
                           ...que for utilizada terá o valor 125 atribuído a ela*/
```

Abaixo podemos ver um exemplo de código que faz a mesam função do exemplo anterior, porém utilizaremos a diretiva *#define* para nomear o LED L1 e o Botão B0.

Código fonte arquivo Ex04.c

```
#include<16F887.h>          /*Inclusão do arquivo header (*.h)  para o
                            microcontrolador utilizado*/
#use delay (clock=8000000)  /*Definição da frequência do cristal para cálculo
                            ..dos delays*/
#fuses HS, NOWDT, PUT,BROWNOUT, NOLVP  //Configuração dos bits de controle

#use fast_io(B)             /*Direcionamento dos pinos da porta B serão feitos
                            pelo programador*/
```

```
#use fast_io(D)              /*Direcionamento dos pinos da porta D serão feitos
                             pelo programador*/

#define B0 PIN_B0            /*Informa ao compilador que o valor B0 será substituído
                             por PIN_B0 toda vez que este for utilizado */

#define L1 PIN_D0            /*Informa ao compilador que o valor L1 será substituído
                             por PIN_D0 toda vez que este for utilizado */

void main()
{
set_tris_b(0b00000001);      /*Somente o pino 0 da porta B será direcionado como
                             entrada*/

set_tris_d(0b00000000);      //Todos os pinos da porta D serão saída

output_D(0b00000000);        //Coloca em 0 toda a porta D


while(true)
      {
      if (input(B0)==0)      //Se o botão for pressionado
        {
        output_Toggle(L1);   //Muda o estado do pino 0 da porta D (LED L1)
        delay_ms(300);       //Atraso de 300ms
        }
      }
}
```

Funções

As funções em linguagem C são muito parecidas com as sub-rotinas da linguagem assembly. O seu uso permite executar uma seqüência de comandos sempre que necessitarmos, sem a necessidade de repetí-los.

O formato geral de uma função é:

```
{tipo} nome_da_função ({parâmetros})
{
 Comando_1;
 Comando_2;
 ....
}
```

Onde:
tipo: Especifica o tipo de dado que a função retornará para onde ela foi chamada.

nome_da_função: Identifica a função, ou seja, como ela será chamada pelo programa. Este nome não pode ter o mesmo nome utilizado por funções próprias do compilador (palavras reservadas).

parâmetros: Utilizados para enviar valores para a função de modo que estes sejam utilizados pela função para cálculos, atribuições, controle, etc.

Esses valores consistem em tipos de variáveis separados por vírgulas e são opcionais, portanto, podemos escrever funções sem qualquer parâmetro.
Mesmo sem a existência dos valores, os parênteses devem ser utilizados.

Exemplo:

```
#include<16F887.h>          /*Inclusão do arquivo header (*.h)  para o
                            microcontrolador utilizado*/
#use delay (clock=8000000)  /*Definição da frequência do cristal para cálculo
                            ..dos delays*/
#fuses HS, NOWDT, PUT,BROWNOUT, NOLVP  //Configuração dos bits de controle

int soma (int a, int b)
{
return a + b;
}

void main ()
{
int c;
c = soma (2,3);
while(true);
}
```

Neste exemplo, definimos uma função soma que retornará o resultado da operação dos parâmetros “a” + o parâmetro “b”. Note que o a função é do tipo short, isso então especifica que a função retornará um valor do tipo short de 8 bits.
No corpo da função, encontramos somente o código ‘**return** a + b’, onde o comando “**return**” é utilizado para retornar o valor da operação dos parâmetros “a + b”.
No bloco principal, encontramos a chamada da função através de ‘c = soma (2,3)’, onde a variável “**c**” receberá o valor do retorno da função (note que a variável também foi declarada com o mesmo tipo da função) e os parâmetros são os valores “**2**” e “**3**”, separados pela vírgula e dentro dos parênteses.

Então neste caso, a função soma recebe os valores “2” e “3” e estes serão atribuídos às variáveis “**a**” e “**b**”, respectivamente e a função retornará o valor da soma entre eles, portanto o valor “**5**” que será

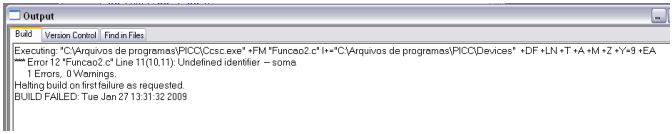
repassado para a variável “c” no corpo principal do programa.

Protótipo de função

Um programa em linguagem C pode ficar muito grande e muito complexo e, às vezes, uma função pode ser chamada antes desta ter sido definida. Neste caso, o compilador gera um erro, veja o exemplo abaixo:

```
#include<16F887.h>          /*Inclusão do arquivo header (*.h)  para o
                            microcontrolador utilizado*/
#use delay (clock=8000000)  /*Definição da frequência do cristal para cálculo
                            ..dos delays*/
#fuses HS, NOWDT, PUT,BROWNOUT, NOLVP  //Configuração dos bits de controle

void main ()
{
int c;
c = soma (2,3);
while(true);
}

int soma (int a, int b)
{
return a + b;
}
```

Veja que a função ‘soma’ está definida após a função principal (main). Neste caso, como a função ‘soma’ foi definida após a sua chamada, o compilador retornará um erro dizendo que o identificador ‘soma’ não foi declarado, veja a mensagens de erro na próxima figura:



Erro na função soma

Podemos solucionar este tipo de problema ao declararmos previamente a função, ou seja, informaremos ao compilador que existe uma função com aquele "nome" que estamos "chamando". Isto é conhecido como prototipagem de função.

O protótipo de função deve ser declarado obedecendo aos mesmos tipos e parâmetros da função seguido do ponto e vírgula (;) e, normalmente, são declarados logo no início do programa.

Exemplo:

```
#include<16F887.h>          /*Inclusão do arquivo header (*.h)  para o
                            microcontrolador utilizado*/
#use delay (clock=8000000)  /*Definição da frequência do cristal para cálculo
                            ..dos delays*/
#fuses HS, NOWDT, PUT,BROWNOUT, NOLVP  //Configuração dos bits de controle

int soma (int a, int b);

void main ()
{
int c;
c = soma (2,3);
while(true);
}

int soma (int a, int b)
{
return a + b;
}
```


PIC
TECNOLOGIA E TREINAMENTO