Projeto de Sistemas Microcontrolados

Aula 07 - Fundamentos de Linguagem C – Parte II

Apresentação

Nesta aula, serão complementados os fundamentos necessários para edição de programas em Linguagem C para os compiladores CCS e C18. Embora comum a várias linguagens, serão apresentadas, também, as principais estruturas de fluxo de controle de programa, bem como as principais funções de acesso à entrada e saída de dados. Encerrando a aula, teremos um estudo de como manipular matrizes em C e desenvolver dois *templates* para edição de programas em C para os compiladores CCS e C18.

Objetivos

Ao final desta aula você será capaz de:

- Definir quais as principais estruturas de controle de fluxo de programas em C.
- Usar algumas funções C para entrada/saída de dados em um projeto baseado em PIC.
- Manipular dados de matrizes.
- Escrever programas em C para os compiladores CCS e C18 a partir de dois *templates* básicos.

Declarações de controle de fluxo de programa em linguagem C

A seguir, iremos rever os conceitos vistos nas aulas de programação do modulo básico, os quais estão relacionados aos comandos de seleção, de repetição e de desvio de fluxo de programa em linguagem C.

Comandos de seleção

Os comandos de seleção são aplicados quando se deseja realizar um determinado tipo de ação, com base no resultado de uma expressão condicional. Os comandos empregados pelos compiladores CCS e C18 são os mesmos dos compiladores C padrões, neste caso o **if** e o **switch**.

Comando if

Esse comando executa um ou mais comandos C, presentes num laço **if**, se uma dada **expressão** for verdadeira (diferente de '0'). Caso contrário, executa o bloco de comandos presentes no laço **else**.

Sintaxe

```
1  if(expressão)
2  {
3     // Bloco de comandos C a serem executados
4  }
5  else
6  {
7     // Bloco de comandos C a serem executados
8  }
9
```

O uso das chaves, **{ }**, delimitam o início e o fim do comando **if, else**.

Exemplo

```
1   void main( ) // Função principal
2   {
3      int cont = 6;   // Declara uma variável do tipo int.
4      short int cond;    // Declara uma variável do tipo short.
5
6   if(cont<= 5)   // Verifica se a expressão é verdadeira.
7      { cond = 1; }   //  Indica uma condição verdadeira.
8   else
9      { cond = 0; } // Indica uma condição falsa.
10  }
11
```

Comando Switch

O comando **switch** verifica se o valor de uma dada **expressão** é verdadeiro, em uma lista de constantes. Se o valor for igual ao rótulo (constante_1, constante_2 e constante_n) de uma cláusula **case**, as instruções pertencentes a ele serão executadas. Mas, se nenhum **case** atender à condição, então, os comandos presentes na cláusula **default** serão executados.

Sintaxe

```
1   switch(expressão)
2   {
3      case constant_1:
4         // Bloco de comandos C a serem executados;
5         break;
6      case constant_2:
7         // Bloco de comandos C a serem executados;
8         break;
9      case constant_n:
10        / / Bloco de comandos C a serem executados;
11        break;
12     default:
13        // Bloco de comandos C a serem executados;
14  }
15
```

De forma idêntica,o uso das chaves, **{ }**, indica o início e o fim do comando **switch**.

Na sintaxe do comando **switch,** pode-se observar a existência da cláusula **break**, empregada para efetuar a saída imediata da cláusula **case**. Se a cláusula **break** não existir em algum dos **case**, as instruções serão executadas até que uma cláusula **break** seja encontrada ou o último **case** seja executado.

Exemplo

```
unsigned char int_para_char(int número) //Função secundária
{
   unsigned char caractere_pro; //declara uma variável do tipo unsigned char
  switch(numero)
  {
        case 0: caractere_pro = '0';
              break;
        case 1: caractere_pro = '1';
              break;
        case 2: caractere_pro = '2';
              break;
        case 3: caractere_pro = '3';
              break;
        default: caractere_pro = ' ';
              break;
  }
   return caractere_pro; // Sai da função e retorna o valor do caractere_pro
}

void main( )   // Função principal
{
   unsigned int num; // Declara uma variável do tipo unsigned int.
   unsigned char caractere; // Declara uma variável do tipo unsigned char.
   num = 2; // Atribui o valor 2 decimal a variável num.
   caractere = int_para_char(num); /* Chama a função int_para_char() e armazena o caractere re
}

```

Comandos de repetição

São estruturas construídas de forma a executar, repetidamente, um determinado comando ou bloco de comandos enquanto uma determinada condição for verdadeira. As estruturas de repetição existentes na linguagem C são compostas pelos comandos **for**, **while** e **do while**.

Comando for

O comando **for** é, geralmente, utilizado quando se pretende repetir uma ou mais instruções uma quantidade de vezes predefinida.

Sintaxe

```
for (inicialização; expressão; incremento_decremento)
{
   // Bloco de commandos C a serem executados
}
```

Exemplo

```
int cont;                  // Declara uma variável do tipo int
int vet_num[3];         // Declara um vetor de três posições do tipo int
for(cont=0; cont<3; cont++)
{
   vet_num[cont] = cont;
}
```

Comando While

O comando **while** é empregado em situações em que o laço pode ser finalizado a qualquer momento, devido à ligação entre a expressão e as ações executadas dentro do laço. Para isso, inicialmente, a condição deve ser avaliada, caso seja verdadeira, então, o comando ou bloco de comandos associado deverá ser executado. Caso a condição seja falsa, o comando ou bloco de comandos não é executado e o programa tem sequência a partir da declaração seguinte ao bloco **while**.

Sintaxe:

```
while(expressão)
{
        // Bloco de comandos C a serem executados
}
```

Exemplo

```
#include <stdio.h> // Adiciona a biblioteca padrão de I/O.
void main()    // Função principal
{
   unsigned char contador = 0;
   while(contador<5)
   {
    printf("Valor do contador: %u\n", contador);     // Imprime o valor
                                                      // do contador
    contador++;
   }
}
```

No exemplo, as instruções no laço **while** serão executadas enquanto o valor da variável contador for inferior a 5.

do while

O comando **do while** é muito parecido com o comando **while**. A diferença é que o comando **do while** testa a condição no final de cada ciclo de iteração do laço de repetição, ao contrário do **while**, que realiza o teste da condição no início do laço.

Sintaxe

```
do
{
  // bloco de comandos C a serem executados
} while (expressão);
```

Exemplo

```
#include <stdio.h> // Adiciona a biblioteca padrão de I/O.
void main() // Função principal
{
   unsigned char contador = 0;
   do
   {
    printf("Valor do contador: %u\n", contador);     // Imprime o valor
                                                      // do contador
    contador++;
   } while(contador<5)
}
```

As instruções contidas na estrutura de repetição do comando **do while** serão executadas enquanto o valor da variável contador for inferior a 5.

Comandos de desvio

A linguagem C dispõe de comandos que permitem o desvio incondicional do programa, ou seja, por meio desses comandos é possível sair de um laço ou de um comando de seleção a qualquer momento, ignorando as expressões condicionais. Os comandos de desvio presentes nos compiladores CCS e C18 são **break**, **continue**, **goto** e **return**.

Comando break

O comando **break** é utilizado para finalizar a execução de um laço (**for**, **while**, ou **do while**) ou de um comando **switch**.

Exemplo

```
1 int cont = 0;    // Declara uma variável do tipo int com atribuição de valor.
2 while(1)        //Looping infinito
3 {
4    if (cont == 5)
5       { break; }      // Força a saída do laço while.
6    cont++;
7 }
```

No trecho de código acima, se observa que o laço **while** é realizado enquanto o valor da variável *cont* for diferente de 5, pois, em seu corpo de programa, há um comando **if**, cuja expressão é verdadeira, quando o valor da variável *cont* for igual a 5. Então, a instrução **break** executa a saída do laço.

Comando continue

O comando **continue** é muito semelhante ao comando **break**. A diferença é que o comando **continue** força a execução da próxima iteração de um laço (**for**, **while** e **do while**) em vez de forçar a saída do laço. Toda vez que esse comando for utilizado, as instruções abaixo dele são ignoradas e uma nova iteração é iniciada.

Exemplo

```
#include <stdio.h> //Adiciona a biblioteca padrão de I/O.
void main( ) // Função principal
{
   unsigned int cont = 0; /* Declara uma variável do tipo unsigned char (8 bits) com atribuição de va
      for (cont=0; cont< 6; cont++)
      {
         if (cont == 4)
            { continue; }     //Pula para a próxima iteração
            printf ("\nteste");
        }
}
```

O laço **for** envia a *string* teste pela *stream* de saída, exceto quando a variável *cont* for igual a 4 ou maior que 5. No momento em que a variável *cont* for igual a 4, a condição **if** será verdadeira e a instrução **continue** será executada, dessa forma, o controlador salta a função **printf( )** e uma nova iteração é iniciada.

Comando goto

Esse comando força o desvio do controle para um determinado ponto dentro da função, identificado pelo rótulo.

Sintaxe

```
goto rótulo;
   ...
rótulo:                  // Um identificador deve vir acompanhado de ' : '
   ...
```

Exemplo

```
int cont;         // Declara uma variável do tipo int.
for (cont=0; cont< 6; cont++)
{
   if (cont == 4)
   { goto fim; } // Desvia o contador para o rótulo fim.
}
fim:
// Ponto de comandos relacionados ao rótulo fim.
```

Após a execução do laço **for**, a variável *cont* passa a ser incrementada a cada iteração. Ao atingir um valor 4, a condição **if** passa a ser verdadeira e o comando **goto** desvia o contador para a posição do rótulo indicado (fim), e o programa continua a ser executado a partir desse ponto.

Comando return

Esse comando retorna o valor de uma função. Ele pode ser empregado em qualquer parte da função a qual foi chamada e, sempre que for executado, a função será finalizada e um valor será retornado.

Sintaxe

```
return expressão;
```

Um ponto importante que deve ser observado é que esse comando pode ser empregado com qualquer tipo de função, exceto a **void,** pois funções desse tipo não retornam valores.

Exemplo

```
long operação (int num_opcao, int a, int b) // Função secundária
{
   switch (num_opcao)
   {
      case 1: return (a + b); // Retorna o resultado da soma dos elementos a e b.
      case 2: return (a - b); // Retorna o resultado da subtração dos elementos a e b.
      case 3: return (a * b); // Retorna o resultado da multiplicação dos elementos.
      case 4: return (a / b); // Retorna o resultado da divisão dos elementos a e b.
      default: return 0; // Retorna 0.
   }
}

void main( ) // Função principal.
{
   unsigned char x = 50, y  = 10; /* Declara duas variáveis do tipo unsigned char com atribuição de v
   long resultado = 0; // Declara uma variável do tipo long.
   // Suponha que as opções são: 1-soma, 2-subtração, 3-multiplicação, 4-divisão.
   resultado = operação(1, x, y);
   resultado = operação(2, x, y);
   resultado = operação(3, x, y);
   resultado= operação(4, x, y);
}
```

Matrizes

Matriz é um agrupamento de variáveis do mesmo tipo, associadas a um nome. Ela pode ter uma única dimensão (unidimensional), como mostrado na Figura 1, ou mais de uma dimensão (multidimensional), como mostrado na Figura 2, onde cada posição corresponde a uma variável do tipo especificado, acessível por meio de um índice.

Matriz unidimensional

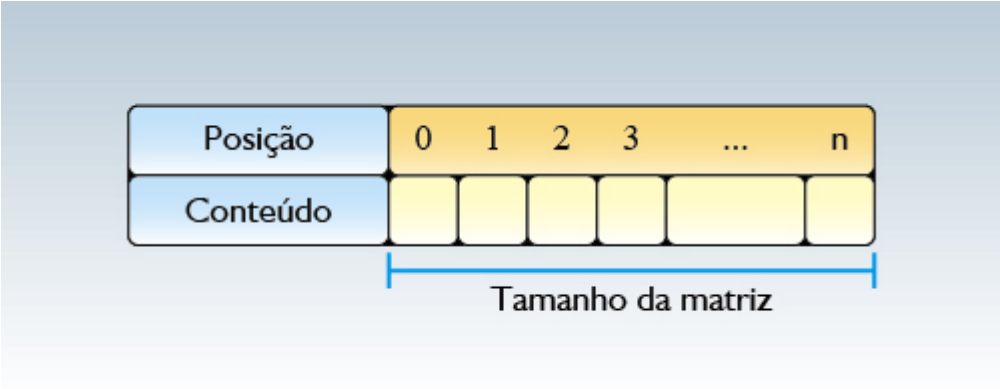
É um tipo de matriz que armazena os elementos em uma única dimensão. Nesse caso, a primeira posição (ou índice) da matriz é sempre o 0. Por exemplo, numa matriz de cinco elementos, os índices variam de 0 a 4, totalizando as cinco posições.

Sintaxe

```
tipo identificador [quantidade_posições];
```

Onde, **tipo** é um tipo de dado válido, **identificador** é o nome da matriz e **quantidade_posições** é o número de elementos da matriz.

Figura 01 - Matriz unidimensional



Exemplo

```
void main() //Função principal
{
   unsigned char erro[5] = {12,33,45,66,70};  /* Declara e atribui valores a uma matriz de cinco elem
   unsigned char numero_erro;                 /* Declara uma variável do tipo unsigned char para armaz
   numero_erro = erro[3];                     /* A variável numero_erro recebe o valor (66) presente na

   erro[0] = 192;                            //Sobrescreve o valor da posição 0 da matriz.
   numero_erro = erro[0];                     /*A variável numero_erro recebe o valor (192) presente na
}
```

Pode-se constatar que a matriz declarada no exemplo anterior tem valores atribuídos às suas respectivas posições. Caso não fossem atribuídos valores iniciais à matriz, as correspondentes posições iriam conter lixo de memória, por essa razão, todas as posições devem ser preenchidas adequadamente durante a codificação do programa a fim de se evitar erros de inconsistência de dados.

Matriz multidimensional

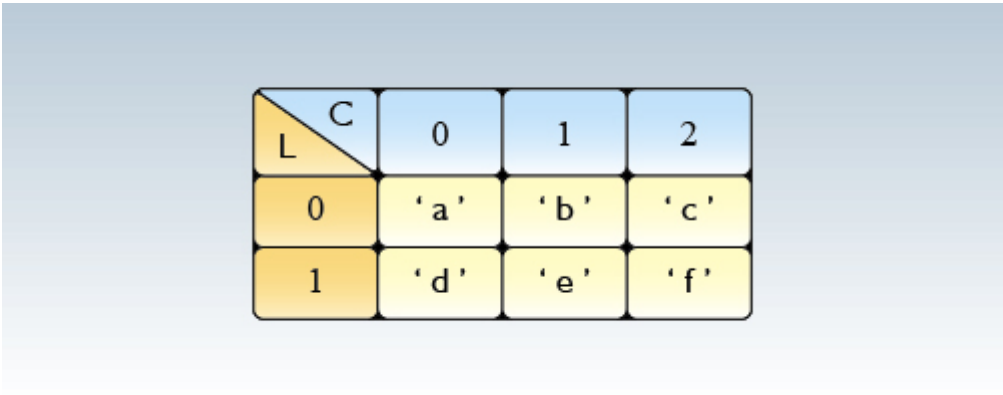
Esse tipo de matriz armazena os valores dos dados em mais de uma dimensão.

Sintaxe:

```
1 tipo  identificador [quantidade_pos_x][quantidade_pos_y] ... [quantidade_pos_z];
```

Sendo: **tipo**: a definição de tipo de dado válido; **identificador**: o nome da matriz; **quantidade_pos**: o número de elementos da matriz.

Figura 02 - Matriz bidimensional



Exemplo

```
1 char mat[2][3] = {'a','b','c','d','e','f'};    // Declara uma matriz 2x3 do tipo char.
2 char caractere;      /* Declara uma variável do tipo char para armazenar o caractere da matriz.*/
3 caractere = mat[0][1];      /* A variável recebe o caractere ('n') presente na posição especificada.*/
```

Para facilitar a compreensão de matrizes multidimensionais, exemplificamos uma matriz bidimensional 2x3, ou seja, com duas linhas e três colunas, como mostrado na Figura 2. Nesse caso, os índices irão variar de 0 a 1 para as linhas e de 0 a 2 para as colunas.

Funções da Linguagem C

Os compiladores CSS e C18 adotam algumas funções básicas da linguagem C, algumas vezes modificando-as, como também ampliando esse leque de funções, como forma de adequação ao microcontrolador a ser programado. Essas funções são utilizadas, principalmente, para estabelecer a comunicação entre o usuário e o *chip* destino, via compilador. A seguir serão citadas e trabalhadas as mais comuns.

Função printf()

Essa é uma função de **I/O** ou de **E/S** (entrada/saída) e permite escrever em um dispositivo padrão, como um display, por exemplo.

Sintaxe

```
1 printf (“string”, variável);
```

Exemplo

```
1 printf(“valor = %d”, x); // Imprime valor pela saída padrão (stdout).
```

Função puts( )

Escreve uma *string* na saída serial padrão (stdout).

Sintaxe

```
1 puts (string);
```

Exemplo

```
1 puts(“testando ...”); // Imprime: testando ... pela saída padrão (stdout).
```

Função getc( )

Aguarda a chegada de um caractere pela porta serial padrão e retorna o seu valor.

Sintaxe

```
1 getc( );
```

Exemplo

```
int x;
x = getc();      // Aguarda a recepção de um caractere pela interface serial.
```

Função abs( )

Retorna o valor absoluto (sem sinal) de um número.

Sintaxe

```
valor = abs( x );
```

Exemplo

```
signed int x;
x = -5;
x = abs(x);     // o novo valor de x será positivo.
```

Função exp( )

Calcula o valor de e^x.

Sintaxe

```
exp(x);
```

Exemplo

```
float numero;
numero = exp(2);     /*A variável número irá armazenar o valor da exponencial de 2 que é 7.3891.*/
```

Função atoi( )

Converte uma *string* em um valor inteiro de 8 *bits*.

Sintaxe

```
1 atoi(x);
```

Exemplo

```
1 char teste[ ] = {"32"};
2 int valor;
3 valor = atoi(teste); /* A variável valor irá armazenar um valor inteiro correspondente a string 32, arr
```

Como veem, as funções são simples e de fácil manipulação. Como neste curso o objetivo não é ensinar a linguagem C, mas, praticar o seu uso em microcontroladores PIC, vamos encerrar a explanação da Lingagem C por aqui. Você terá muito ainda a aprender. Isso poderá ser alcançado nas nossas aulas práticas, que vêm por aí.

Tornando o estudo mais objetivo, vamos agora utilizar o programa de controle do semáforo, escrito em *assembly*, na Aula 5, e desenvolver *templates* C, voltados para os compiladores CCS e C18.

Ambiente de compilação CCS

Na Aula 5, foi visto como escrever um programa em a*ssembly*, utilizando o ambiente MPLAB, que permitia controlar um semáforo de dois tempos. Na Aula 6, e complementado nesta, foram vistos os conceitos da Linguagem C, sob a ótica de dois compiladores: o CCS e o C18. Vamos, agora, como uma primeira orientação prática em Linguagem C, escrever, num *template* básico para o CCS, o mesmo programa que permitiu controlar o semáforo de dois tempos da aula citada. Num segundo momento, será pedido que escreva o mesmo *template*, só que voltado para o compilador C18. Lembre-se que, embora o C seja uma linguagem de mais alto nível que o a*ssembly* e que, de uma forma geral, apresente um código com bem menos linhas que o mesmo em *assembly*, após o processo de compilação, cada

linha em linguagem C é transladada e equivale a várias linhas de código em *assembly*, mas, nem sempre, as mais otimizadas. Aproveitaremos o desenvolvimento desses *templates* para, também, conhecer um pouco dos dois ambientes de programação: o do CCS e o do C18.

Iniciando com o primeiro *template*, temos, na Figura 3, a tela do CCS PIC C *Compiler* com o exemplo do programa que permite controlar o semáforo de dois tempos.

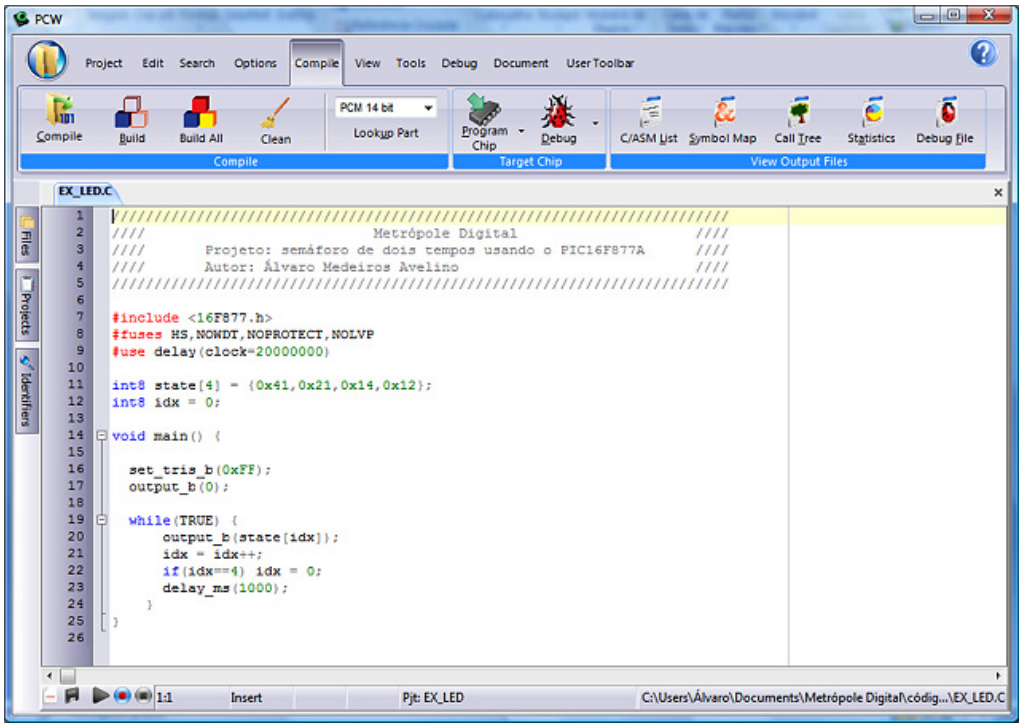
As linhas 2, 3, 4 e 5 possuem comentários simples sobre o código escrito.

A linha 7 é a primeira linha de código propriamente dito. Consiste em informar ao compilador que modelo de microcontrolador será utilizado. Isso é feito por meio da diretiva **#include <modelo_do_microcontrolador.h>**. Observe que a extensão do arquivo é **.h**. Em linguagem C, os arquivos com essa extensão possuem definições. Nesse caso, o arquivo possui definições tais como nomes de cada um dos pinos, periféricos (conversor analógico-digital, interface serial, interface paralela etc), bem como funções de inicialização dos periféricos.

A linha 8 contém a diretiva **#fuses**. Essa diretiva configura os **fuse bits** do PIC (ou *bits* de configuração do microcontrolador em uso).

A linha 9 contém a diretiva **#use delay(valor_do_clock)**. Essa diretiva é usada para indicar o valor do oscilador à cristal, utilizado no projeto. Nesse caso, está sendo utilizado um cristal de 20 MHz. O valor deve ser escrito por extenso (20 MHz = 20000000).

Figura 03 - Tela de edição do software CCS com controlador do semáforo de dois tempos.



Uma vez configurado o PIC, o próximo passo é a declaração das variáveis que serão utilizadas. No Quadro 3, da Aula 5, reeditado aqui como Quadro 1, é possível ver todos os estados que o semáforo de dois tempos pode assumir.

	S1	**S2**
Verde	RB6	RB2
Amarelo	RB5	RB1
Vermelho	RB4	RB0

Quadro 1 – Mapeamento de *bits* usado para os dois semáforos

Para criar (ou atribuir) os quatro estados em linguagem C, foi criado um vetor com quatro posições usando a sintaxe:

```
tipo_do vetor nome_do_vetor[tamanho_do_vetor] = {valor1,valor2,valorn}
```

No exemplo foi criado um vetor do tipo inteiro (**int8**), cujo nome é **state**, que apresenta quatro posições, cujos valores são, respectivamente: **0x41** (o **x** indica representação hexadecimal), **0x21**, **0x14**, **0x12**. Além do vetor, foi criada uma variável chamada **idx**, do tipo **int8**, com valor zero.

Após a declaração das variáveis, vem a função principal, que em linguagem C é chamada **main**. A declaração dessa função aparece na linha 14.

Na linha 16, é utilizada uma função que declara alguns pinos do microcontrolador como saída (para acionar os LEDs do semáforo). Para isso, é utilizada a função **set_tris_b(0x00)**. Essa função declara todos os pinos do PORTB do PIC como saída.

Uma vez determinada a direção (entrada ou saída) dos pinos a serem utilizados, o próximo passo é dizer o valor inicial da saída (alto ou baixo). Para isso, usou-se a função **output_b(valor_da_saida)**. Na linha 17, por exemplo, todos os pinos do PORTB do PIC estão, inicialmente, com valor baixo.

Por último, na linha 19, pode-se ver uma função chamada **while(TRUE)**. Trata-se de uma função que está sempre em repetição (também conhecida como laço infinito). Nesse exemplo, a função possui quatro linhas de código.

A linha 20 utiliza, novamente, a função **output_b**. Desta vez o argumento da função é um índice do vetor **state** na posição **idx**. Ou seja, a primeira vez em que essa linha for executada, o argumento da função **output_b** será o índice zero do vetor **state** (**0x41**).

Na linha 21, o índice é incrementado.

Na linha 22, a função **if** é utilizada para testar se o índice é o último elemento do vetor. Se for o índice é zerado.

Na linha 23, é utilizada a função **delay_ms(valor_do_delay)**. Essa função consiste em um atraso (em milissegundos) do valor contido no argumento da função. Nesse exemplo, é utilizado um atraso de 1000ms = 1s. O atraso torna-se necessário para que seja possível visualizar a mudança no estado do semáforo de dois tempos.

Após a execução da linha 23, o programa volta a executar do início do **while(TRUE)**, na linha 19.

Atividade 01

Como única atividade desta aula:

1. Reveja o ambiente de programação MPLab e veja como incluir a suíte de programação para o compilador C18.
2. Tomando como base o template escrito para o CCS, desenvolva um também para o compilador C18.
3. Reescreva para o C18 o programa de controle para o semáforo de dois tempos apresentado.
4. Admita que o microprocessador a ser usado no projeto do semáforo de dois tempos será o 18F45k20.

Resumo

Nesta aula, foram complementados alguns conceitos básicos da Linguagem C, tanto para uso com o compilador CCS como para o compilador C18. Encerrando, foi apresentado um *template* para edição de edição de programas em C para o compilador CCS, ficando, como atividade, a apresentação de um *template* para o compilador C18.

Autoavaliação

1. Quais os principais comandos de seleção usados na estruturação de fluxo de programas em C?
2. Quais os principais comandos de repetição usados na estruturação de fluxo de programas em C?
3. Quais os principais comandos de desvio usados na estruturação de fluxo de programas em C?
4. Quais são as definições possíveis para matrizes?
5. Cite algumas funções usadas pelos compiladores CCS e C18.

Referências

DEITEL, H. M.; DEITEL, P. J. **Como programar em C**. Rio de Janeiro: LTC, 1999.

MIYADAIRA, Alberto Noboru. **Microcontroladores PIC18**: aprenda e programe em linguagem C. São Paulo: Érica, 2009.

PEREIRA, Fábio. **Microcontroladores PIC**: técnicas avançadas. São Paulo: Érica, 2002.

_____. **Microcontroladores PIC**: Programação em C. Fábio Pereira. São Paulo: Érica, 2005.

_____. **Microcontroladores PIC 18 Detalhado**: Hardware e Software. São Paulo: Érica, 2010.

SOUZA, David José de. **Desbravando o PIC**. São Paulo: Editora Érica, 2000.

SOUZA, David J. LAVINIA, Nicolas C. **Conectando o PIC**: Explorando recursos avançados. São Paulo: Érica, 2003.