Laboratório 4: Prática PIC “Hello World” e “ADC”

Alan Garcia, Ana Barbosa, Cesar Junior e Luan dos Santos

Resumo—O microcontrolador PIC16F877 é estudado e manipulado neste laboratório. As práticas ”Hello World”, onde uma programação básica do microcontrolador em duas linguagens, Assembly e C, é realizada, e ”ADC”, onde o conversor analógico-digital embutido é utilizado, são demonstradas. Circuitos para ambos os casos são simulados e fisicamente arranjados para visualização dos procedimentos do laboratório. O procedimento ”Hello World”já havia sido realizado préviamente na disciplina de Microprocessadores com outro microcontrolador, o AT89S51. Para essa programação, comparações são realizadas entre os códigos resultantes com ambos os microcontroladores e, no caso do laboratório 4, ambas as linguagens utilizadas na manipulação do PIC16F877.

Palavras-chave—PIC16F877, laboratórios, microcontroladores, microprocessadores.

1 Introdução

TRABALHAMOS, até agora, com o microcontrolador AT89S51 dentro dos laboratórios da disciplina de Microprocessadores, realizando práticas o suficiente para conseguirmos detalhar sua linguagem (programação do microcontrolador), sua inserção (motivação quanto ao uso) e comportamento (características da componente) em um circuito.

No laboratório 4, que apesar de propor uma simples programação “Hello World” instiga e exige mais de uma forma de manipulação quanto ao microcontrolador utilizado, a componente trabalhada é o PIC16F877 – que desempenha mesmo papel do AT89S51 durante seu próprio laboratório “Hello World”, primeiro dos projetados para a disciplina.

Diferenças entre as componenetes incluem o intervalo de operação com relação a tensão (4V a 5V no AT89S51 [1] e uma diferença significativa de aproximadamente 2V com o PIC16F877, que opera entre 2V a 5.5V [2]), o conversor analógico-digital já embutido (como é o caso do PIC16F877 que disponibiliza um *ADC* de 10 bits [2]) e o conjunto de instruções (o AT89S51 apresenta uma quantidade bem maior, como esperado; enquanto ele apresenta uma abordagem *CISC*, o PIC16F877 faz parte das abordagens *RISC*). Veremos que o modo de programação também se difere entre os microcontroladores (mesmo quando uma mesma linguagem é utilizada). O acumulador oferecido pelo AT89S51, por exemplo, é manipulado via o registrador especial `A` quando necessário; essa manipulação não é explicita no PIC16F877.

Características adicionais apresentadas pelo PIC16F877 incluem o espaço reservado quanto as memórias, o processamento veloz por conta da frequência de operação e o número de linguagens que reconhece.

Se encontram a seguir as especificações do laboratório. Como já mencionado, todo o desenvolvimento é apresentado na seção 3. Os resultados são encontrados na seção 4 e as considerações na seção 5.

• *A. Garcia, A. Barbosa, C. Junior e L. dos Santos são da Universidade Estadual de Ponta Grossa.*

2 Procedimento

Na prática ”Hello World”, o comumente piscar de um LED em um circuito físico foi proposto (o fluxograma de um sistema com esse propósito foi projetado e apresentado no laboratório 1) através da programação do PIC16F877 em duas linguagens: Assembly e C.

Após a validação de ambos os códigos, o microcontrolador e seu conversor analógico-digital são manipulados para que apresentem a tensão presente em um circuito físico arranjado, que é alterada a partir de um potenciômetro. O fluxograma para esse sistema é apresentado em Fig. 1. A programação do PIC16F877 na segunda parte do laboratório é obrigatóriamente realizada em linguagem C.

2.1 Componentes

As componentes utilizadas no laboratório são listadas a seguir. Os comentários sobre suas presenças no circuito são realizados na seção 3.

- Arduino;
- PIC16F877;
- Crystal (12MHz;)
- Capacitor (22pF, 10uF, 0.1uF);
- Resistência (330 a 10k);
- LED;
- Visor LCD (16x2);
- Potenciômetro (10k ou 1k).

2.2 Material auxiliar

As ferramentas utilizadas na programação do microcontrolador e na simulação do circuito prévia a montagem física são ArdPicProg [3] e Proteus [4], respectivamente. A IDE MPLAB X [5] é utilizada na manutenção do projeto. A IDE Arduino [6] é utilizada na manipulação da placa Arduino.

3 Desenvolvimento

3.1 ”Hello World”

Apesar de requisitada em duas linguagens, o circuito físico resultante do procedimento ”Hello World”é o mesmo para ambas as programações do microcontrolador (a escolha de

entradas e saídas da componente é um fator facultativo; apesar de diferente neste laboratório - por pura facilitação na apresentação de resultados -, a mesma porta poderia ser utilizada em ambos os casos). Ele pode ser observado nas simulações do projeto (Fig. 4, Fig. 5, Fig. 6 e Fig. 7). A gravação é realizada pelos pinos PGC e PGD do microcontrolador, específicos para transmissão de dados. Os capacitores e resistores são utilizados para manter a frequência do cristal estável e reduzir a corrente gerada sobre as componentes, respectivamente, e podem variar de acordo com o arranjo do circuito. O Arduino é utilizado para alimentação e interfaceamento quanto a programação do microcontrolador.

A programação do piscar do LED com a linguagem Assembly é dividida, principalmente, entre a manipulação da porta onde a componente está conectada (B, nesse caso), que pode ser consultado dentro da função `repeat` do Apêndice A, e o ajuste dos tempos utilizados para uma frequência visível. A linguagem Assembly já havia sido utilizada na programação do AT89S51 para mesma finalidade, o piscar de um LED. Os códigos resultantes, porém, são bem diferentes (a comparação pode ser realizada a partir de Apêndice A e Apêndice E). Nessa programação, o LED fica ativo e inativo por um período em segundos de 1 (na prática, não é bem isso que acontece. O osciloscópio em Fig. 2 mostra que esse período varia. Acredita-se (e espera-se) que essa diferença esteja associada a alguma manipulação erronea na programação do microcontrolador).

A programação do microcontrolador em linguagem C se mostrou muito mais simples, como apresentado no Apêndice B. A porta onde o LED é conectado (D, nesse caso) é diretamente manipulada para que fique ativa por um determinado período de tempo, determinado pela função `delay`. Nessa programação, o LED fica ativo e inativo por um período em milissegundos de 100 (o ajuste na simulação do circuito para esse período é apresentado em Fig. 3).

3.2 "ADC"

Para a apresentação da tensão em um circuito físico proposto na prática "ADC", duas componentes essencias foram utilizadas: o conversor analógico-digital, oferecido pelo PIC16F877, e um visor LCD. A manipulação do visor LCD pode ser verificada no Apêndice D, que é um biblioteca espontâneamente desenvolvida para a prática. Nela, as funções responsáveis pela escrita no visor são encontradas.

A programação do microcontrolador, exigido em linguagem C, é apresentado no Apêndice C. Nela, a conversão analógica-digital de valores de tensão obtidos no circuito físico através do pino RA0 é realizada. Essa conversão é necessária para que o visor possa interpretar de forma correta a informação de tensão sendo recebida. A tensão pode ser modificada através de um potenciômetro. Sua utilização não afeta a leitura programada, porém. Ela é realizada a cada intervalo de tempo pré-determinado, independentemente. O visor LCD é atualizado a cada 2 ms, aproximadamente (tempo de descarga do capacitor utilizado; o tempo de conversão do microcontrolador não é considerado nesse intervalo). Além da tensão, o visor também apresenta as iniciais dos nomes dos autores.

A simulação dessa prática pode ser visualizada em Fig. 8 e seu circuito final em Fig. 9. O PIC16F877 é conectado ao visor LCD pelas portas B e D através dos conectores em amarelo. O potenciômetro adicional é utilizado para variação do contraste do visor LCD.

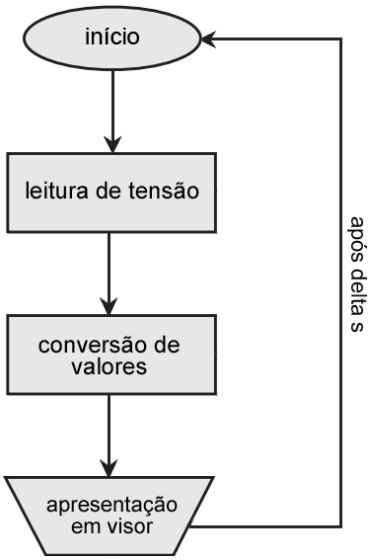

Fig. 1. Fluxograma "ADC". A tensão no circuito é verificada e apresentada a cada variação de tempo em milissegundos *delta* = 2.

4 Resultados

As simulações e circuitos físicos resultantes desse laboratório são apresentados a seguir. As codificações e comentários quanto a elas (Apêndice F) são encontradas como apêndices.

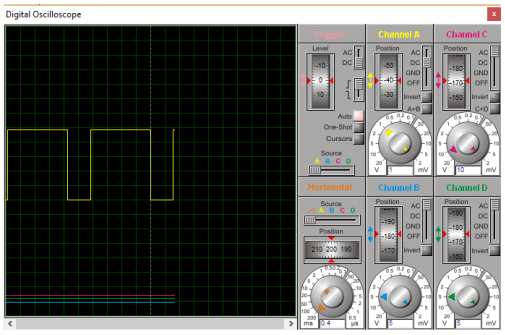

Fig. 2. Manipulação de tempo na prática "Hello World"em linguagem Assembly.

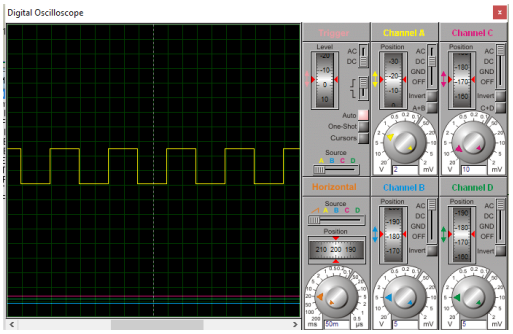

Fig. 3. Manipulação de tempo na prática "Hello World"em linguagem C.

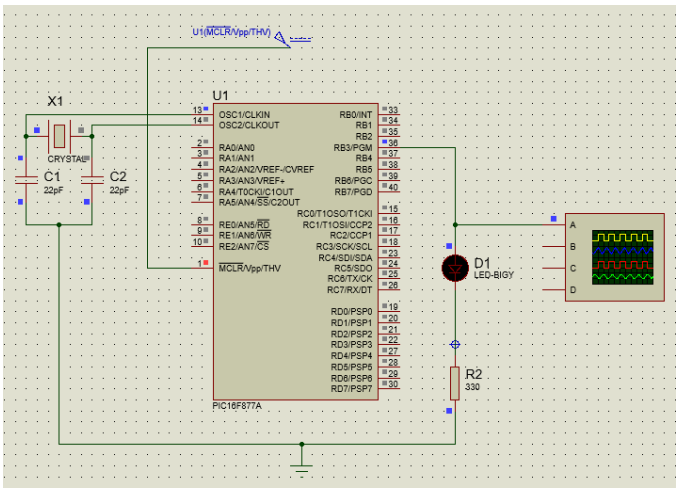

Fig. 4. Simulação do circuito ”Hello World”em linguagem Assembly. O LED fica inativo por um período de tempo em segundos de 1.

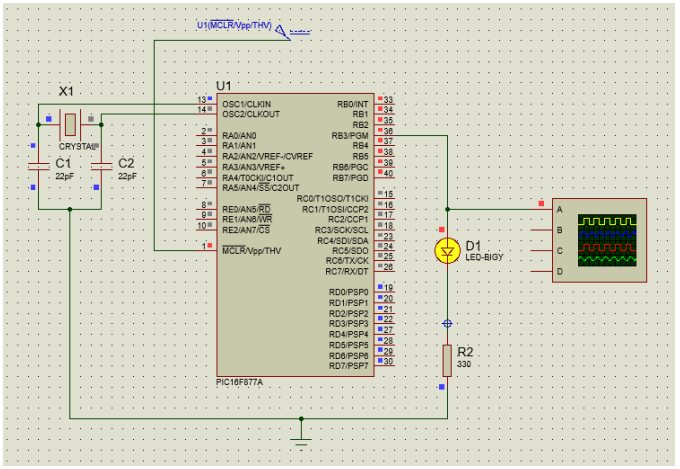

Fig. 5. Simulação do circuito ”Hello World”em linguagem Assembly. O LED fica ativo por um período de tempo em segundos de 1.

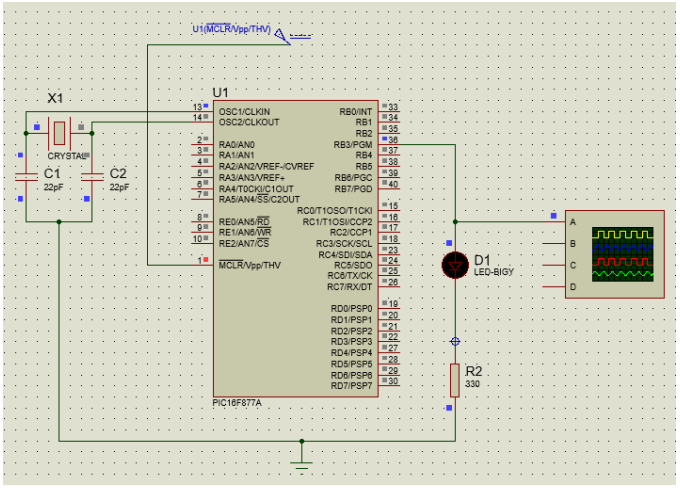

Fig. 6. Simulação do circuito ”Hello World”em linguagem C. O LED fica ativo por um período de tempo em milissegundos de 100.

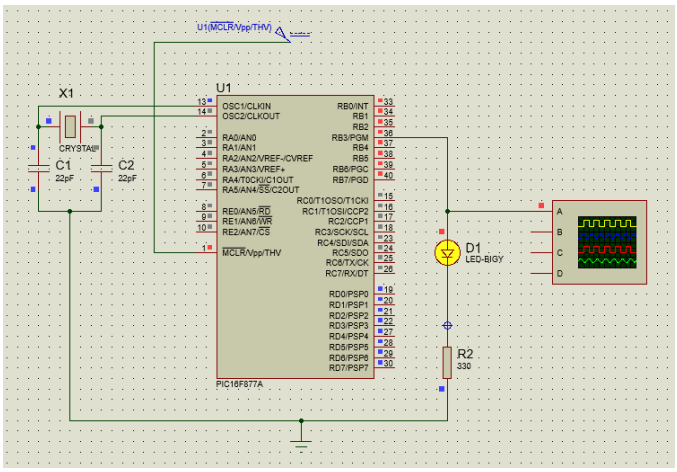

Fig. 7. Simulação do circuito ”Hello World”em linguagem C. O LED fica inativo por um período de tempo em milissegundos de 100.

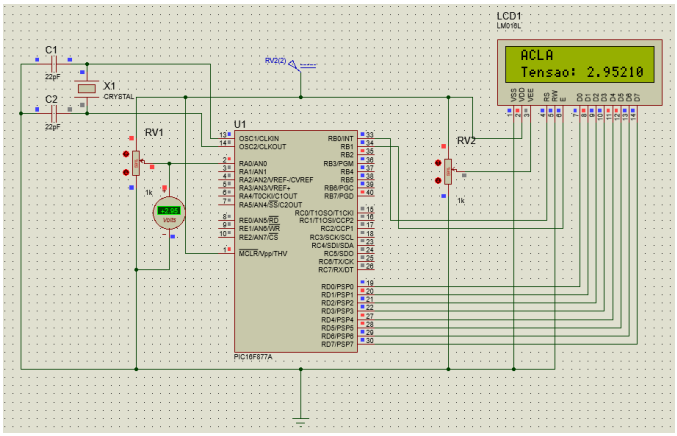

Fig. 8. Simulação do circuito ”ADC”. O visor apresenta a tensão no circuito e as iniciais dos nomes dos autores.

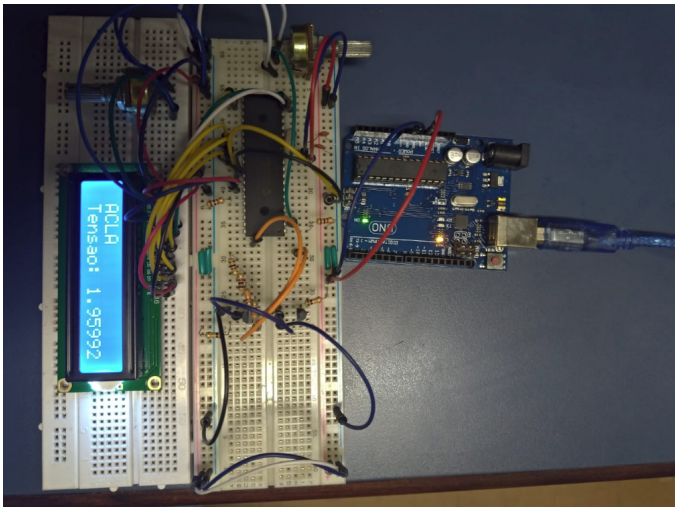

Fig. 9. Circuito físico para ”ADC”. A tensão no circuito é de aproximadamente 2 Volts.

5 CONCLUSÃO

Existem duas observações a serem feitas quanto a codificação desse laboratório. A primeira é que a

programação do PIC16F877 em linguagem Assembly se mostra complexa e longa quando comparada com a mesma programação (e linguagem) do AT89S51. Fica evidente através de ambos os códigos que a programação do microcontrolador usado nos laboratórios anteriores é simplificada e intuitiva. A segunda, porém, é que a programação do PIC16F877 em linguagem C é demasiadamente trivial. Mesmo na implementação da biblioteca de manipulação do visor LCD, que contém um conjunto complexo de operações, essa abordagem se mostrou menos exigente. É possível que a experiência obtida com o AT89S51 e a familiaridade com a linguagem C infuenciem essa facilidade na manipulação do PIC16F877. Todas as operações necessárias para os laboratórios "Hello World" 1 e 4 são realizadas de forma semelhante; apesar de diferentes nomes, existem instruções que realizam o mesmo tipo de operação entre os microcontroladores (especialmente quanto ao grupo básico, como o deslocamento). De qualquer forma, a programação nessa linguagem foi preferível no desenvolvimento do laboratório.

O desenvolvimento da biblioteca para manipulação do visor LCD fez valer dois aspectos: a otimização e independência de código na programação. Dispositivos limitados podem ser sobrecarregados quanto ao que se é utilizado em sua programação. Se torna interessante o aprofundamento desses quesitos tanto na carreira profissional (confecção de drivers e programas controladores de dispositivos, por exemplo) como na carreira acadêmica (alcance de limite inferior de complexidade de tempo de um algoritmo, por exemplo!).

As características apresentadas pelo PIC16F877 aparentam maior flexibilidade e adaptabilidade quando comparadas as apresentadas pelo AT89S51, mas apenas quando a programação é realizada em linguagem C. Quando utilizando a linguagem Assembly, é mais seguro e desejável o manuseio do AT89S51, que apresenta um maior conjunto de comandos de forma mais clara.

REFERÊNCIAS

[1] Microchip Technology Inc. (2003) *PIC16F87XA Data Sheet: 28/40/44-Pin Enhanced Flash Microcontrollers*.

[2] Atmel Corporation. (2008) *8-bit Microcontroller with 4K Bytes In-System Programmable Flash: AT89S51*.

[3] R. Weatherley. (Julho, 2013) *Ardpicprog*. Disponível em: rweather.github.io/ardpicprog/.

[4] Labcenter Electronics Ltd. (2018) *Proteus Design Suite*. Disponível em: labcenter.com/.

[5] Microchip Technology Inc. (1998-2018) *MPLAB® X Integrated Development Environment (IDE)*. Disponível em: microchip.com/mplab/mplab-x-ide/.

[6] Arduino. (2018) *Arduino (IDE)*. Disponível em: www.arduino.cc/en/Main/Software/.

APÊNDICE A

```
1.  #include "p16f877a.inc"
2.
3.   __CONFIG _FOSC_HS & _WDTE_ON &
4.  _PWRTE_OFF & _BOREN_ON & _LVP_ON
5.   & _CPD_OFF & _WRT_OFF & _CP_OFF
6.
7.  list P = 16F877A
8.  include P16F877A.inc
9.
10. STATUS EQU 0x03
11. PORTB EQU 0x06
12. TRISB EQU 0x86
13. PORTD EQU 0x08
14. TRISD EQU 0x88
15.
16. CBLOCK 0x20
17.       Kount100us
18.       Kount10ms
19.       Kount1s
20. ENDC
21.
22. org 0x0000
23.       goto START
24. org 0x05
25. START
26.
27.       banksel TRISD
28.       movlw 0x00
29.       movwf TRISD
30.       movwf TRISB
31.       banksel PORTB
32.       clrf PORTB
33.       clrf PORTD
34.
35. repeat
36.       movlw 0xFF
37.       movwf PORTB
38.       Call Delay1s
39.       movlw 0x00
40.       movwf PORTB
41.       Call Delay1s
42.       goto repeat
43.
44. Delay100us
45.       banksel Kount100us
46.       movlw H'A4'
47.       movwf Kount100us
48. R100us
49.       decfsz Kount100us
50.       goto R100us
51.       return
52.
53. Delay10ms
54.       banksel Kount10ms
55.       movlw H'64'
56.       movwf Kount10ms
57. R10ms
58.       call Delay100us
59.       decfsz Kount10ms
60.       goto R10ms
61.       return
62.
63. Delay1s
64.       banksel Kount1s
65.       movlw H'64'
66.       movwf Kount1s
67. R1s
68.       call Delay10ms
69.       decfsz Kount1s
```

```
70.         goto R1s
71.         return
72.  END
```

APÊNDICE B

```
1.  #include <xc.h>
2.  #define _XTAL_FREQ 12000000UL
3.
4.  #pragma config FOSC = HS
5.  #pragma config WDTE = ON
6.  #pragma config PWRTE = OFF
7.  #pragma config BOREN = ON
8.  #pragma config LVP = ON
9.  #pragma config CPD = OFF
10. #pragma config WRT = OFF
11. #pragma config CP = OFF
12.  /*
13.  *
14.  */
15.  int main() {
16.
17.  TRISD = 0;
18.  PORTD = 0;
19.
20.     while(1)
21.    {
22.          PORTD = 0XFF;
23.          __delay_ms(100);
24.          PORTD = 0X00;
25.         __delay_ms(100);
26.     }
27.
28.     return (0);
29. }
```

APÊNDICE C

```
1.  #define _XTAL_FREQ 12000000UL
2.
3.  #include <xc.h>
4.  #include <stdio.h>
5.  #include "lcd_luan.h";
6.
7.  #pragma config FOSC = HS
8.  #pragma config WDTE = OFF
9.  #pragma config PWRTE = OFF
10. #pragma config BOREN = ON
11. #pragma config LVP = OFF
12. #pragma config CPD = OFF
13. #pragma config WRT = OFF
14. #pragma config CP = OFF
15.
16.  void ADC_Init()
17.  {
18.    ADCON0 = 0x41;
19.    ADCON1 = 0xC0;
20.  }
21.
22.  unsigned int ADC_Read(unsigned char
23.  channel)
24.  {
25.    if(channel > 7)
26.   return 0;
27.   ADCON0 &= 0xC5;
28.   ADCON0 |= channel<<3;
29.   __delay_ms(2);
30.   GO_nDONE = 1;
31.   while(GO_nDONE);
32.   return ((ADRESH<<8)+ADRESL);
33.  }
34.
35.  int main()
36.  {
37.     unsigned int ad;
38.     float t;
39.     char s[20];
40.     TRISD = 0x00;
41.     TRISB = 0x00;
42.     inicia_lcd();
43.     ADC_Init();
44.
45.    while(1)
46.     {
47.      ad = ADC_Read(0);
48.      t = (float)(5.0/1023)*ad;
49.      seta_cursor(1,1);
50.      escreve_string_lcd("ACLA");
51.      seta_cursor(2,1);
52.      sprintf(s,"Tensao: %f",t);
53.      escreve_string_lcd(s);
54.      __delay_ms(1000);
55.      limpa_lcd();
56.    }
57.    return 0;
58.  }
```

APÊNDICE D

```
1.  #ifndef LCD_LUAN_H
2.  #define LCD_LUAN_H
3.
4.  #ifdef __cplusplus
5.  extern "C" {
6.  #endif
7.
8.  #ifdef __cplusplus
9.  }
10.  #endif
11.
12.  #endif /* LCD_LUAN_H */
13.
14.  #define L_RS RB0
15.  #define L_EN RB1
16.  #define L_PORT PORTD
17.
18.  void escreve_lcd(char s)
19.  {
20.    L_RS = 1;
21.    L_PORT = s;
22.    L_EN = 1;
23.    __delay_ms(5);
```

```
24.    L_EN = 0;
25.    L_RS = 0;
26.  }
27.
28.  void escreve_string_lcd(char *s
29.  {
30.    int i;
31.    for(i=0; s[i] != '\0'; i++){
32.      escreve_lcd(s[i]);
33.    }
34.  }
35.
36.  void limpa_lcd()
37.  {
38.  L_PORT = 0x01;
39.  L_EN = 1;
40.  __delay_ms(5);
41.  L_EN = 0;
42.  }
43.
44.  void retorna_carro()
45.  {
46.    L_PORT = 0x02;
47.    L_EN = 1;
48.    __delay_ms(5);
49.    L_EN = 0;
50.  }
51.
52.  void gira_direita()
53.  {
54.    L_PORT = 0x18;
55.   L_EN = 1;
56.   __delay_ms(5);
57.   L_EN = 0;
58.  }
59.
60.  void gira_esquerda()
61.  {
62.    L_PORT = 0x1C;
63.    L_EN = 1;
64.    __delay_ms(5);
65.    L_EN = 0;
66.  }
67.
68.  void seta_cursor(int y, int x)
69.  {
70.    int i=0;
71.    retorna_carro();
72.    if(y == 2)
73.    L_PORT = 0xC0;
74.    L_EN = 1;
75.    __delay_ms(5);
76.    L_EN = 0;
77.    for(i=0;i<x;i++){
78.      L_PORT = 0x14;
79.      L_EN = 1;
80.      __delay_ms(5);
81.      L_EN = 0;
82.    }
83.  }
84.  void inicia_lcd()
85.  {
86.     L_PORT = 0x00;
87.     __delay_ms(20);
88.     L_RS = 0;
89.     L_PORT = 0x38;
90.    L_EN  = 1;
91.    __delay_ms(4);
92.    L_EN  = 0;
93.
94.    __delay_ms(5);
95.
96.    L_RS = 0;
97.    L_PORT = 0x03;
98.    L_EN  = 1;
99.    __delay_ms(8);
100.   L_EN  = 0;
101.
102.   __delay_ms(11);
103.
104.   L_RS = 0;
105.   L_PORT = 0x08;
106.   L_EN  = 1;
107.   __delay_ms(8);
108.   L_EN  = 0;
109.
110.   L_RS = 0;
111.   L_PORT = 0x00;
112.   L_EN  = 1;
113.   __delay_ms(4);
114.   L_EN  = 0;
115.
116.   L_RS = 0;
117.   L_PORT = 0x0C;
118.
119.   L_EN  = 1;
120.   __delay_ms(8);
121.   L_EN  = 0;
122.
123.   L_RS = 0;
124.   L_PORT = 0x00;
125.   L_EN  = 1;
126.   __delay_ms(4);
127.   L_EN  = 0;
128.
129.   L_RS = 0;
130.   L_PORT = 0x06;
131.
132.   L_EN  = 1;
133.   __delay_ms(4);
134.   L_EN  = 0;
135. }
```

APÊNDICE E

```
1.  org 0;
2.  inicio:;
3.       mov R7, #0FFh;
4.       setb P1.0;
5.       mov A, R7;
6.       mov R0, A;
7.       loop1:;
```

```
8.            mov A, R7;
9.            mov R1, A;
10.           loop2:;
11.                   mov A, R1;
12.                   subb A, #01h;
14.                   mov R1, A;
15.                   jnz loop2;
16.           mov A, R0;
17.           subb A, #01h;
18.           mov R0, A;
19.           jnz loop1;
20.      clr P1.0;
21.      mov A, R7;
22.      mov R0, A;
23.      loop3:;
24.           mov A, R7;
25.           mov R1, A;
26.           loop4:;
27.                   mov A, R1;
28.                   subb A, #01h;
29.                   mov R1, A;
30.                   jnz loop4;
31.           mov A, R0;
32.           subb A, #01h;
33.           mov R0, A;
34.           jnz loop3;
35.      sjmp inicio;
36.  END
```

APÊNDICE F
COMENTÁRIOS

Apêndice A

3. 4. 5. – Configuração de dispositivo
10. – Ativa endereço de registradores
23. – Inicia programa
27. – Seleciona TRISD
28. – PORTB e PORTD com 0000 como saída
32. 33. – Limpa PORTB e PORTD
36. – PORTB = ’0b11111111’
37. – PORTB ativo
38. – Atraso em s de 1
39. – PORTB = ’0b00000000’
40. – PORTB inativo
42. – Repetição de função
44. – Rotina para atraso em ms de 100
45. – Seleciona banco da variável
47. – Move valor para variável
49. – Decrementa a variável
50. – Repete laço quando diferente de 0
53. – Rotina para atraso em ms de 10
63. – Rotina para atraso em s de 1

Apêndice B

2. – Frequência do clock
4. 5. 6. 7. 8. 9. 10. 11. – Configuração de dispositivo
17. – Ativa PORTD como saída
18. – Ativa PORTD em nível lógico baixo
22. – Ativa LED
23. – Atraso em ms de 100
24. – Desliga LED
25. – Atraso em ms de 100

Apêndice C

1. – Frequência do clock
5. – Inclusão de biblioteca espotaneamente desenvolvida
16. – Módulo ADC iniciado
18. – Seleção de clock
19. – Pinagem em entrada analógica
25. 26. – Pára para canal não válido
27. – Limpa bits do canal
28. – Ativa bits requeridos
29. – Tempo de estabilização
30. – Início de conversão A/D
31. – Aguarda conversão
32. – Retorno de resultado final
37. – Variável de leitura A/D
38. – Variável de tensão
39. – Variável de escrita
40. – Ativa PORTD como saída
41. – Ativa PORTB como saída
42. 43. – Inicia LCD e ADC
47. – Leitura de ADC
48. – Conversão binário-decimal real
49. – Alocação de cursor: primeira linha
50. – Escrita em tela: iniciais dos autores (Obs.: a escrita poderia ser alterada para que a palavra apresentada fosse "JUAN")
51. – Alocação de cursor: segunda linha
52. – Conversão do valor de tensão para string
53. – Escrita em tela: tensão
54. – Atraso em s de 1
55. – Limpa o visor

Apêndice D

14. – Pino de gravação
15. – Pino de ativação
16. – Pino de dados
18. – Procedimento de escrita (char) em visor
20. – Ativa RS para envio de 1 byte de dados
21. – Ativa PORT de dados com byte de char
22. – Ativa a leitura do visor
23. – Aguardo em ms de 5 (cômputo do LCD)
24. – Desativa a leitura do visor
25. – Desativa a leitura em modo de dados
28. – Procedimento de escrita (string) em visor
31. 32. – Escrita individual de caracter no LCD
36. – Procedimento de limpeza do visor
38. – PORT de envio = 0000 0001 (comando de limpeza de tela)
39. – Ativa leitura de LCD
40. – Aguardo em ms de 5 (cômputo do LCD)

41. – Desativa leitura de LCD
46. – PORT de envio = 0000 0010 (comando de limpeza de tela)
47. – Ativa leitura de LCD
48. – Aguardo em ms de 5 (cômputo do LCD)
49. – Desativa leitura de LCD
54. – PORT de envio = 0001 1000 (comando de limpeza de tela)
55. – Ativa leitura de LCD
56. – Aguardo em ms de 5 (cômputo do LCD)
57. – Desativa leitura de LCD
62. – PORT de envio = 0001 1100 (comando de limpeza de tela)
63. – Ativa leitura de LCD
64. – Aguardo em ms de 5 (cômputo do LCD)
65. – Desativa leitura de LCD
72. – Condicional = posição de cursor é na segunda linha
73. – PORT de envio = 1100 0000 (comando de limpeza de tela)
74. – Ativa leitura de LCD
75. – Aguardo em ms de 5 (cômputo do LCD)
76. – Desativa leitura de LCD
77. – Giro de cursor x vezes para a direita
78. – PORT de envio = 0001 0100 (comando de limpeza de tela)
79. – Ativa leitura de LCD
80. – Aguardo em ms de 5 (cômputo do LCD)
81. – Desativa leitura de LCD
86. – Inicia PORT de envio como 0
87. – Atraso em ms de 20 para inicializações
88. – RS = 0
89. – Envia definições de interface e comunicação
0011 1000 comunicação de 8 bits e 2 linhas de texto
90. – E = 1
91. – Aguardo em ms de 4 (leitura de LCD)
92. – E = 0
94. – Estabilização
97. – Posição de cursor no início
99. – Aguardo em ms de 8 (leitura de LCD)
102. – Estabilização
105. – Comando de opções de cursor com visor desligado
107. – Aguardo em ms de 8 (leitura de LCD)
111. – Limpeza de PORT
113. – Aguardo em ms de 4 (leitura de LCD)
117. – Comando de ativação de visor sem cursor
120. – Aguardo em ms de 8 (leitura de LCD)
124. – Limpeza de PORT
126. – Aguardo em ms de 4 (leitura de LCD)
130. – Comando de propriedades de escrita do cursor
133. – Aguardo em ms de 4 (leitura de LCD)

Apêndice E

4. – Ativa LED em PINO1
6. – Adiciona o valor teto ao contador
7. – Início de laço para espera em s de 1
9. – Contador para laço aninhado
10. – Início de loop aninhado
15. 19. – Condicional
20. – Desativa LED
22. – Adiciona o valor teto ao contador
23. – Início de laço para espera em s de 1
25. – Contador para laço aninhado
26. – Início de laço aninhado
30. – Condicional
34. – Condicional
35. – Retorno ao início de programa