



Microcontroladores PIC

on-line **GRÁTIS!**

- ▶ [Índice](#)
- ▶ [Sistema de desenvolvimento](#)
- ▶ [Contacte-nos](#)



CAPÍTULO 6

Exemplos

[Introdução](#)

[6.1 Alimentando o microcontrolador](#)

[6.2 Utilização de macros em programas](#)

[6.3 Exemplos](#)

[Teclado](#)

[Optoacopladores](#)

[Optoacoplador numa linha de entrada](#)

[Optoacoplador numa linha de saída](#)

[O Relé](#)

[Produzindo um som](#)

[Registos de deslocamento](#)

[Registo de deslocamento de entrada 74HC597](#)

[Registo de deslocamento de saída paralela](#)

[Displays de 7-Segmentos \(multiplexagem\)](#)

[DISPLAY LCD](#)

[Macros para LCD](#)

[Conversor Analógico-Digital de 12 bits](#)

[Comunicação Série](#)

Displays de 7-Segmentos (multiplexagem)

Os segmentos num display de 7 segmentos podem ser seleccionados de modo a obtermos quaisquer dos caracteres hexadecimais de 0 a F, um de cada vez, conforme se pode ver na animação:



É possível o visionamento de números com vários dígitos, utilizando displays adicionais. Embora seja mais confortável trabalhar com displays LCD (displays de cristal líquido), os displays de 7 segmentos continuam a constituir um standard na indústria. Isto devido à sua robustez em relação à temperatura, visibilidade e amplo ângulo de visão. Os segmentos são representados pelas letras minúsculas: a, b, c, d, e, f, g, dp, em que dp representa o ponto decimal.

Os 8 LEDs contidos no display podem estar dispostos nas configurações de ânodo comum ou de cátodo comum. Nos displays de cátodo comum, o cátodo comum deve ser ligado à massa e para que os leds acendam, é preciso aplicar uma tensão positiva aos respectivos ânodos (1 lógico). Os displays de ânodo comum apresentam o ânodo comum ligado a +5V e acendem quando se aplica um nível lógico zero aos cátodos respectivos. O tamanho do display é medido em milímetros, que corresponde à altura do display propriamente dito (não do encapsulamento mas sim do dígito!). No mercado, estão disponíveis displays com tamanho de 7, 10, 13.5, 20 ou 25mm. Podem também aparecer em diversas cores como vermelho, laranja e verde.

A maneira mais simples de alimentar um display é utilizando um 'display driver'. Estes estão disponíveis para até 4 displays.

Alternativamente, os displays podem ser actuados por intermédio de um microcontrolador e, se necessitarmos de mais que um display, podemos utilizar o método de 'multiplexagem'.

A principal diferença entre estes dois métodos, consiste no número de linhas utilizadas para fazer as ligações aos displays. Um 'driver' especial, pode necessitar apenas de uma linha de "clock" e será o chipe que o contém que irá aceder aos segmentos e incrementar o display.

Se o microcontrolador for alimentar um único display, então apenas serão necessárias 7 linhas ou mais uma se utilizarmos o ponto decimal. Se utilizarmos vários displays, então precisamos de uma linha adicional para cada display.

Para construirmos displays de 4, 5 ou 6 dígitos, devemos ligar em paralelo todos os displays de 7 segmentos.

A linha de cátodo comum (no caso de displays de cátodo comum) é tratada separadamente e é posta a nível baixo durante um curto espaço de tempo para acender o display.

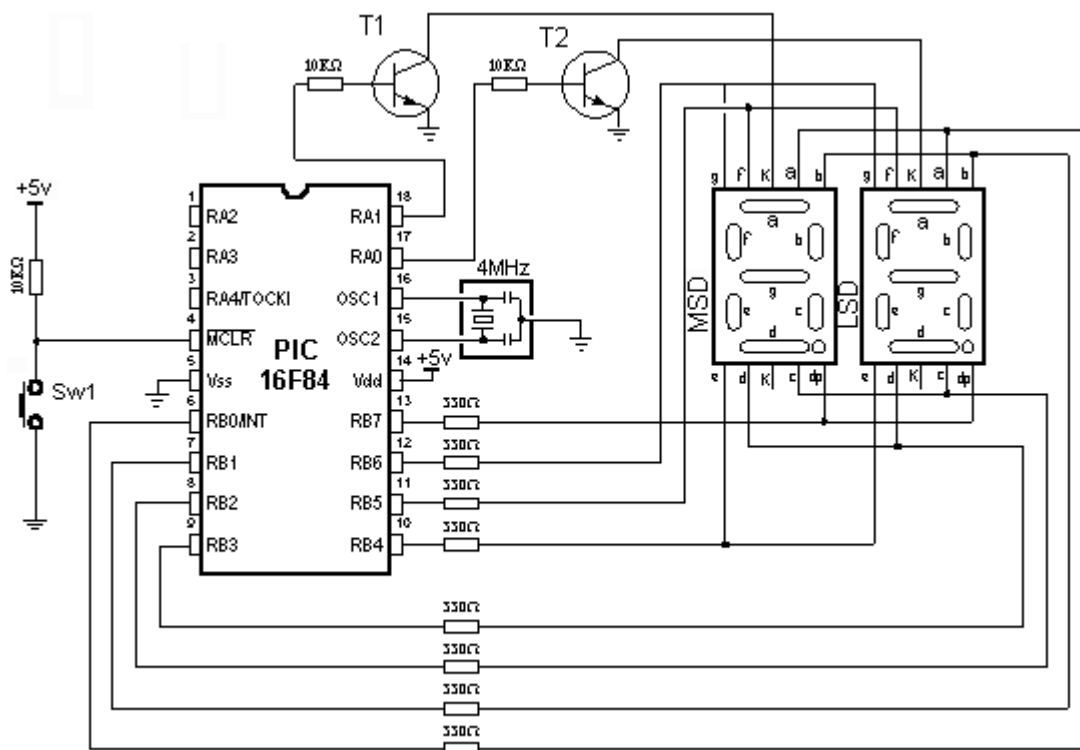
Todos os displays devem acender-se sucessivamente um após outro e, este processo, deve repetir-se cerca de 100 vezes por segundo, fazendo com que todos os displays acesos em simultâneo.

Sempre que um display é seleccionado, e para que a leitura seja correcta, o dado correspondente a esse display deve estar disponível nas linhas que vão ligar aos segmentos. Até 6 displays podem ser acedidos deste modo, sem que o brilho desses displays seja afectado. Cada display é activado durante um sexto do tempo com bastante intensidade e, a persistência da imagem nos nossos olhos, faz parecer que todos os displays estão todos acesos ao mesmo tempo.

As temporizações de todos os sinais destinados aos displays são produzidas pelo programa, a grande vantagem de ser o microcontrolador a lidar com os displays, é a sua flexibilidade.

O programa do microcontrolador pode ser concebido para obtermos uma contagem crescente ou decrescente no display, ou para produzir um certo número de mensagens usando letras do alfabeto que são geradas facilmente.

O exemplo em baixo, mostra como activar dois displays.



Ligando um microcontrolador a displays de 7 segmentos no modo multiplexado

O ficheiro Led.inc contém duas macros: LED_Init e LED_Disp2. A primeira macro é usada para iniciação do display. É nela que o período de refrescamento do display é definido bem como quais os pinos do microcontrolador que vão ser ligados aos displays. A segunda macro é usada para visualizar os números de 0 a 99 nos dois displays.

A macro LED_Disp2 tem um argumento:

LED_Disp2 macro número

numero é o número de 0 a 99 que vai ser mostrado nos dígitos MSD e LSD.

Exemplo: LED_Disp2 0x34

Neste caso, vai aparecer o número 34 nos displays.

Neste caso, o número 34, vai aparecer nos dois displays

A implementação da macro mostra-se na listagem que se segue.



7-seg.inc

```

;**** Macros ****

LED_Init macro
    call InitPorts
    call InitTimers
endm

LED_Disp2 macro num
    movlw num
    movwf LO
    call UpdateDisplay
endm

;**** Subprogramas ****

InitPorts
    BANK1
    clrf LEDtrisA           ;Pinos RA0-4 de saída
    clrf LEDtrisB         ;Porto B de saída
    BANK0
    clrf LEDportA         ; Todos os bits a
    clrf LEDportB         ; nível lógico 0
    bsf  LEDportA,3       ; Activar display MSD

    RETURN

InitTimers
    BANK1
    movlw B'10000100'     ; Prescaler atribuído a TMR0
    movwf OPTION_REG     ; e igual a 32
    BANK0
    movlw B'00100000'     ; Habilitar interrupção por TMR0
    movwf INTCON
    movlw .96
    movwf TMR0           ; Iniciar o temporizador

    RETFIE

;****ISR - Rotina de Serviço de Interrupção****

ISR
    bcf  INTCON,GIE       ; Inibir todas as interrupções
    btfsc INTCON,GIE      ; Verificar se estão inibidas
    goto ISR

    movlw .96             ; Iniciar TMR0
    movwf TMR0
    bcf  INTCON,TOIF      ; Limpar a flag TOIF
    call UpdateDisplay    ; atualizar o display

    RETFIE

UpdateDisplay
    movf  LEDportA,W      ; Estado dos displays -> registo w
    clrf  LEDportA        ; Apagar todos os displays de 7 segmentos
    andlw 0x0f            ; Isolar os quatro bits menos significativos
    movwf TempC           ; Guardar o estado dos displays em TempC
    bsf  TempC,4          ; Estado inicial do display menos significativo
    rrf  TempC,F          ; Estabelecer estado do display seguinte
    btfss STATUS,C ; c=1 ?
    bcf  TempC,3          ; Se não, apagar display menos significativo
    btfsc TempC,0         ; Se sim, verificar estado do display + significativo
    goto UpdateMsd       ; Se display activado, mostrar o byte + significativo

UpdateLsd
    call  ChkMsdZero      ; dígito + significativo = 0 ?
    btfss STATUS,Z        ; sim, ignorar inst. seguinte
    movf  LO,W            ; dígito - significativo -> W
    andlw 0x0f            ; mascarar o que não interessa

```

O programa que se segue, exemplifica a utilização de macros num programa. Este programa faz aparecer o número '21' nos dois displays de 7 segmentos.

LED.asm

```

;***** Escolher e configurar o microcontrolador *****

PROCESSOR 16F84
#include "p16f84.inc"

    __CONFIG_CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC

;***** Declarar variáveis *****

Cblock 0x0C          ; Início da Ram
TempC              ; Pertence à macro "LED_Dis2"
LO
endc

;***** Declarar o hardware *****

LEDtrisA equ TRISA
LEDportA equ PORTA

LEDtrisB equ TRISB
LEDportB equ PORTB

;***** Estrutura da memória de programa *****

ORG 0x00           ; Vector de reset
goto Main

ORG 0x04           ; Vector de interrupção
goto ISR           ; A rotina de interrupção encontra-se
                  ; no ficheiro 7-seg.inc

#include "bank.inc" ; ficheiros auxiliares
#include "7-seg.inc"

Main
LED_Init          ; Início do programa

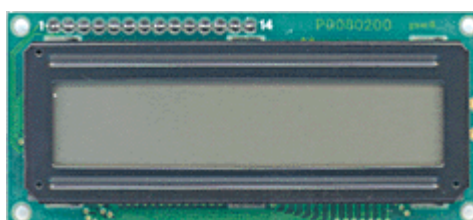
LED_Dis2 0x21     ; Visualizar nos dois displays de 7 segmentos
                  ; o número "21"
loop goto loop    ; Permanecer aqui

End               ; Fim do programa

```

DISPLAY LCD

Cada vez mais os microcontroladores estão a usar 'displays de cristal líquido - LCD' para visualizarem a saída de dados. A discussão que se segue diz respeito à ligação de um **display LCD Hitachi** a um microcontrolador PIC. Estes displays LCD são baseados no módulo de LCD HD44780 da Hitachi, são baratos, fáceis de usar e permitem utilizar os 8x80 pixels de display. Estes displays LCD contêm um conjunto de caracteres ASCII standard e ainda caracteres japoneses, gregos e símbolos matemáticos.



Display Hitachi HD44780 de duas linhas com 16 caracteres por linha

Cada um dos 640 pixels do display, podem ser acedidos individualmente, esta tarefa é executada por chips de controle montados em superfície, na parte detrás do display.

Isto permite-nos poupar uma enorme quantidade de fios e linhas de controle, de tal maneira que, através de poucas linhas é possível fazer a ligação do display ao mundo exterior. É possível comunicar com o exterior através de um bus de 8 bits ou mesmo através de um bus de dados de apenas 4 bits.

No caso de escolhermos um bus de dados de 8 bits, o display requer uma alimentação de +5V mais 11 linhas de entrada e saída. Se optarmos pelo bus de dados de 4 bits, apenas precisamos de 7 linhas mais a alimentação. Quando o display LCD não está habilitado, as linhas de dados tristate assumem o estado de alta impedância (como se estivessem desligadas do circuito) o que significa que não interferem com o funcionamento do microcontrolador.

O LCD também requer do microcontrolador mais 3 linhas de "controle".

A linha **Enable (E)** permite a activação do display e a utilização das linhas R/W e RS. Quando a linha de habilitar (Enable) está a nível baixo, o LCD fica inibido e ignora os sinais R/W e RS. Quando (E) está a nível alto, o LCD verifica os estados das duas linhas de controle e reage de acordo com estes.

A linha **Read.Write (R/W)** determina o sentido dos dados entre o microcontrolador e o LCD. Quando está a nível baixo, os dados estão a ser escritos no LCD. Quando está a nível alto, os dados estão a ser lidos do LCD.

Com a ajuda da linha de **Seleção de registo (RS)**, o LCD interpreta o tipo de dados presentes nas linhas de dados. Quando está a nível baixo, está a ser escrita uma instrução no LCD. Quando está a nível alto é um caracter que está a ser escrito no LCD.

Estado lógico nas linhas de controle:

E	0 Acesso ao LCD inibido
	1 Acesso ao LCD habilitado
R/W	0 Escrever dados no LCD
	1 Ler dados do LCD
RS	0 Instrução
	1 Caracter

A escrita dos dados no LCD, é feita em várias etapas:

Pôr o bit R/W a nível baixo

Pôr o bit RS a nível lógico 0 (instrução) ou a nível lógico 1 (caracter)

Colocar o dado na linha de dados (se for uma operação de escrita)

Pôr a linha E a nível alto

Pôr a linha E a nível baixo

Ler o dado das linhas de dados (no caso de uma operação de leitura)

A leitura de dados do LCD é feita da mesma maneira, mas a linha de controle tem que estar a nível alto. Antes de enviarmos comandos ou dados para o módulo LCD, este tem que ser iniciado. Comandos típicos enviados depois de um reset podem ser: activar um display, visualizar um cursor e escrever os caracteres da esquerda para a direita.

Depois de iniciado o LCD, ele fica pronto para continuar a receber dados ou comandos. Se receber um caracter, ele escreve-o no display e move o cursor um espaço para a direita. O cursor marca o local onde o próximo caracter vai ser escrito. Quando queremos escrever uma cadeia de caracteres, primeiro necessitamos de estabelecer um endereço de início e depois enviar os caracteres, um de cada vez. Os caracteres que podem ser mostrados no display estão guardados na RAM de display de dados (DD). O tamanho da DDRAM é de 80 bytes.

O display LCD também possui 64 bytes de RAM Geradora de Caracteres (CG). Os dados na RAM CG

representam caracteres num mapa de 8 bits.

Cada caracter gasta até 8 bytes de RAM geradora de caracteres, assim, o número total de caracteres que podem ser definidos pelos utilizador pode ir até oito. De modo a ler o mapa de bits de caracteres no display LCD, temos primeiro que estabelecer o endereço de início na CGRAM (0 geralmente) e, a seguir, escrever dados no display. A definição de caracter 'especial' mostra-se na figura ao lado.

Endereços CG RAM	Mapa de bits	Dados
0000	<input type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/>	01010
0001	<input type="checkbox"/> <input type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	00100
0010	<input type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/>	01110
0011	<input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input checked="" type="checkbox"/>	10001
0100	<input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	10000
0101	<input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input checked="" type="checkbox"/>	10001
0110	<input type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/>	01110
0111	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	00000

Depois de definirmos um caracter especial e antes de acedermos à RAM DD, o programa tem que apontar para o endereço da RAM DD. Escrever e ler dados de ou para a memória LCD faz-se a partir do último endereço que é estabelecido usando a instrução set-address (definir endereço). Uma vez fixado o endereço da RAM DD, um novo caracter pode ser visualizado no local apropriado do écran.

Até agora, encarámos as operações de escrita ou leitura relativamente a um LCD como se estas incidissem sobre uma memória normal. Mas não é disto que se trata. O controlador do LCD precisa de 40 a 120 microsegundos (μs) para ler e escrever. Outras operações podem demorar até 5mS. Durante este período de tempo, o microcontrolador não pode aceder ao LCD, assim, num programa, é preciso saber quando o LCD está ocupado. Podemos resolver este problema de duas maneiras.

Apontar endereço DD RAM

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	A	A	A	A	A	A	A

Apontar endereço CG RAM

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	1	A	A	A	A	A	A

Escrever dado na RAM

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
1	0	D	D	D	D	D	D	D	D

Ler dado da RAM

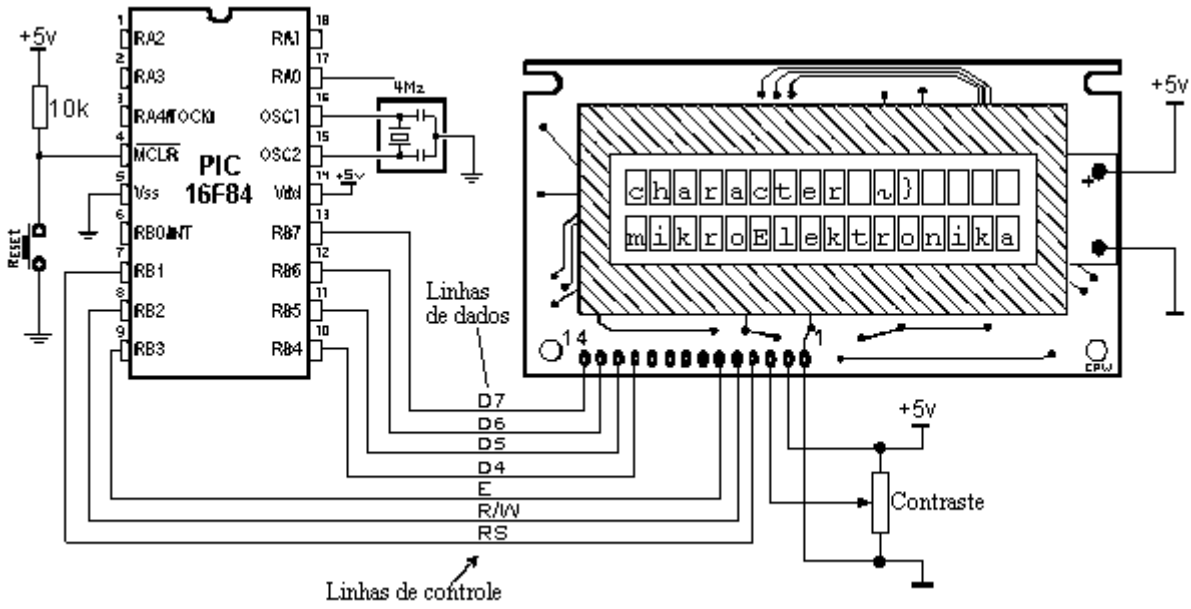
RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
1	1	D	D	D	D	D	D	D	D

A = Endereço

D = Dado

Uma maneira, é verificar o bit BUSY que coincide com a linha de dados D7. Este não é, contudo, o melhor método porque o LCD pode bloquear e o programa permanecer para sempre num loop de verificação do bit BUSY. A outra maneira, é introduzir um tempo de espera no programa. Este período de tempo deve ser suficientemente longo para permitir que o LCD termine a operação. As instruções destinadas a ler ou escrever na memória de um LCD, mostram-se na tabela anterior.

No princípio, dissemos que eram precisas 11 linhas de entrada e saída para comunicar com um LCD. Contudo, também é possível comunicar com um LCD, através de um bus de dados de apenas 4 bits. Deste modo, é possível reduzir para sete o total de linhas de comunicação. Uma ligação, através de um bus de dados de 4 bits, mostra-se no diagrama em baixo. Neste exemplo, nós usamos um display LCD com 2x16 caracteres e que o fabricante japonês SHARP designa por LM16X212. A mensagem 'character' é escrita na primeira linha e dois caracteres especiais '~' e '}' são também mostrados. Na segunda linha, está a palavra 'mikroElektronika'.



Ligação de um display LCD a um microcontrolador

O ficheiro **LCD.inc** contém um grupo de macros para utilizarmos quando trabalhamos com displays LCD.



LCD.inc

```

;***** Declarar o hardware *****

RS    equ    1        ; Selecção de registo
RW    equ    2        ; Ler/Escriver
EN    equ    3        ; Habilitar saída/CLK'

;***** Comandos LCD *****

CONSTANT LCDEM8 = b'00110000' ; Modo 8 bits, 2 linhas
CONSTANT LCDDZ = b'10000000'  ; Escrever 0 na DDRAM
CONSTANT LCDEM4 = b'00100000' ; Modo 4 bits, 2 linhas

;***** Comandos standard para iniciação do LCD *****

CONSTANT LCD2L = b'00101000'  ; Função: 4 bits, 2 linhas
CONSTANT LCDCONT = b'00001100' ; Controle de display: Display ON
                                   ; Cursor OFF, piscar OFF
CONSTANT LCDSH = b'00101000'  ; Modo de display: deslocar cursor
                                   ; sem autodeslocamento do display

;***** Comandos LCD standard *****

;Para enviar um destes comandos para o LCD, usamos a macro LCDcmd
; ex. "LCDcmd LCDCLR"

CONSTANT LCDCLR = b'00000001'  ; limpar o display, repôr cursor
CONSTANT LCDCH = b'00000010'   ; cursor no início
CONSTANT LCDCR = b'00000110'   ; mover cursor p/ a direita
CONSTANT LCDCL = b'00000100'   ; mover cursor p/ a esquerda
CONSTANT LCDSL = b'00011000'   ; conteúdo do display p/ esquerda
CONSTANT LCDSR = b'00011100'   ; conteúdo do display p/ a direita
CONSTANT LCDL1 = b'10000000'   ; seleccionar linha 1
CONSTANT LCDL2 = b'11000000'   ; seleccionar linha 2

;***** Macros *****

LCDinit macro
    call LCD_init                ; iniciação do LCD
endm

LCDchar macro LCDarg                ; escrever caracter no LCD
    movlw LCDarg
    call LCDdata
endm

LCDw macro
    call LCDdata
endm

LCDcmd macro LCDcommand                ; enviar comando para o LCD
    movlw LCDcommand
    call LCDcmd
endm

LCDline macro line_num
    IF (line_num == 1)
        LCDcmd LCDL1        ; Começar macro com o comando "1ª linha"
    ELSE
        IF (line_num == 2)
            LCDcmd LCDL2    ; Começar macro com o comando "2ª linha"
        ELSE
            ENDIF
        ENDIF
    ENDIF
endm

LCD_DDAdr macro DDRamAddress
    Local value = DDRamAddress | b'10000000' ; Início da DDRAM
    IF (DDRamAddress > 0x67)
        EPPOR "Wrong DDRAM address in LCD_DDAdr"
    ENDIF
endm

```

```

;*****Subprogramas*****

LCDcomd clrf LCDbuf          ; Limpar flag de dados
        goto LCDwr

LCDdata clrf LCDbuf
        bsf LCDbuf,RS        ; Pôr a '1' flag de dados

LCDwr movwf LCDtemp          ; Comando/dado em Temp
        andlw b'11110000'    ; isolar os 4 bits + significativos
        iorwf LCDbuf,0       ; isolar flag de dados
        movwf LCDport        ; enviar os 4 bits + significativos p/ o porto do LCD
        call LCDclk
        clrf LCDport
        swapf LCDtemp,0      ; trocar os 4 bits + significativos c/ os 4 bits - sig.
                                ; novamente
        andlw b'11110000'    ; isolar os 4 bits - significativos
        iorwf LCDbuf,0       ; isolar flag de dados
        movwf LCDport        ; enviar os 4 bits - significativos p/ o porto do LCD
        call LCDclk
        clrf LCDport
        RETURN

LCDclk WAITX 0x02,0x00      ; Habilitar transferência de um dado
                                ; ou comando para o LCD

        bsf LCDport,EN
        bcf LCDport,EN
        WAIT 0x02
        RETURN

LCD_init
        clrf LCDport        ; preparar o porto do LCD
        BANK1
        clrf OPTION_REG
        movlw b'00000000'
        movwf LCDtris
        BANK0
        WAIT 0x02

        movlw LCDEM8        ; Iniciação no
        movwf LCDport        ; modo "8 bits"
        call LCDclk
        clrf LCDport
        WAIT 0x02

        movlw LCDDZ        ; escrever 0 na DDRAM
        movwf LCDport
        call LCDclk
        clrf LCDport

        movlw LCDEM4        ; ir para o modo '4 bits'
        movwf LCDport
        call LCDclk
        clrf LCDport

        LCDcmd LCD2L        ; modo 4 bits, 2 linhas
        LCDcmd LCDCONT      ; Display ligado, sem cursor
        LCDcmd LCDSH        ; Autoincrementar display sem autodeslocamento
        LCDcmd LCDCLR       ; Limpar display, apontar endereço 0
        call LCDspecialChars ; Caracteres definidos pelo utilizador
                                ; na CGRAM
        RETURN

LCDspecialChars              ; O utilizador pode definir um
                                ; máximo de 8 caracteres

; *** o primeiro byte do caracter especial está no ***
; *** endereço 0x00 da CGRAM ***

        LCD_CGAdr 0x00      ; Enviar endereço da CGRAM
        LCDchar b'00001010' ; Escrever dado no endereço da CGRAM

```

Macros para LCD

LCDinit macro usada para iniciar o porto a que o LCD está ligado. O LCD é configurado para trabalhar no modo de 4 bits.

Exemplo: LCDinit

LCDchar LCDarg Escrever caracter ASCII. O argumento é o caracter ASCII.

Exemplo: LCDChar `d`

LCDw Escrever o caracter correspondente ao conteúdo do registo W.

Exemplo: movlw `p`

LCDw

LCDcmd LCDcommand Enviar comandos

Exemplo: LCDcmd LCDCH

LCD_DDAdr DDRamAddress Apontar o endereço da DDRAM

Exemplo: LCD_DDAdr .3


LCDline line_num Colocar o cursor no início da primeira ou da segunda linha

Exemplo: LCDline 2

Quando se trabalha com um microcontrolador os números são tratados na forma binária.

Como tal, é difícil apresentá-los num display. É por isso que é necessário converter esses números do sistema binário para o sistema decimal, de modo a que possam ser facilmente entendidos. A seguir, apresentam-se as listagens de duas macros **LCDval_08** e **LCDval_16**.

A macro **LCDval_08** converte um número binário de oito bits num número decimal entre 0 e 255 e mostra o resultado num display LCD. É necessário declarar as seguintes variáveis no programa principal: TEMP1, TEMP2, LO, LO_TEMP, Bcheck. O número binário de oito bits é guardado na variável LO. Quando a macro é executada, o equivalente decimal deste número vai ser mostrado no display LCD. Os zeros à esquerda não irão ser mostrados.



LCDv8.inc

```

;***** Macros *****

LCDval_08 macro
    call LCDval08
endm

;***** Subprogramas*****

LCDval08
    movfw LO           ;LO->LO_TEMP
    movwf LO_TEMP
    clrf Bcheck       ; iniciar indicador de zero à esquerda

    movlw d'100'      ;Determinar algarismo das centenas
    movwf TEMP2
    call VALcnv

    movlw d'10'       ;Determinar algarismo das dezenas
    movwf TEMP2
    call VALcnv

    movlw d'1'        ;Determinar algarismo das unidades
    movwf TEMP2
    bsf Bcheck,0      ;limpar indicador se não houver zeros à esquerda

    call VALcnv
    RETURN

VALcnv
    clrf TEMP1        ;Iniciação do contador
    movfw TEMP2

VALc01
    subwf LO_TEMP,0   ;Testar se LO > 99, c=0 se for maior
    skpc
    goto LCDval2      ;Sair, se for menor

    incf TEMP1,1      ;Incrementar contador
    movfw TEMP2
    subwf LO_TEMP,1   ; TEMP1=TEMP1-100
    bsf Bcheck,0
    goto VALc01

LCDval2 movlw 0'      ;A escrever um dígito no LCD
    addwf TEMP1,0     ;Adicionar ao contador
    btfss Bcheck,0
    movlw ' '         ; Omitir zero à esquerda
    LCDw              ;Escrever dígito no LCD
    RETURN

```

A macro **LCDval_16** converte um número binário de 16 bits num número decimal entre 0 e 65535 e mostrando-o no display LCD. As seguintes variáveis necessitam de ser declaradas no programa principal: TEMP1, TEMP2, TEMP3, LO, HI, LO_TEMP, HI_TEMP, Bcheck. O número binário de 16 bits, ocupa as variáveis LO e HI. Quando a macro for executada, o equivalente decimal do número será apresentado no display LCD. Os zeros à esquerda do número não são mostrados.



LCDv16.inc

```

;**** Macros ****

LCDval_16 macro
    call LCDval16
endm

;**** Subprogramas ****

LCDval16
    movfw LO           ;LO->LO_TEMP
    movwf LO_TEMP     ;SUB-LO
    movfw HI           ;SUB-HI
    movwf HI_TEMP     ;HI->HI_TEMP
    clrf Bcheck       ;iniciar indicador de zero à esquerda

    movlw b'00010000' ;Determinar dezenas de milhar
    movwf TEMP2       ;SUB-LO
    movlw b'00100111' ;SUB-HI
    movwf TEMP3
    call VALcnv

    movlw b'11101000' ;Determinar algarismo dos milhares
    movwf TEMP2       ;SUB-LO
    movlw b'00000011' ;SUB-HI
    movwf TEMP3
    call VALcnv

    movlw b'01100100' ;Determinar algarismo das centenas
    movwf TEMP2       ;SUB-LO
    clrf TEMP3        ;SUB-HI se for zero
    call VALcnv

    movlw b'00001010' ;Determinar algarismo das dezenas
    movwf TEMP2       ;SUB-LO
    clrf TEMP3
    call VALcnv

    movlw b'00000001' ;Determinar algarismo das unidades
    movwf TEMP2       ;SUB-LO
    clrf TEMP3
    bsf Bcheck,0     ;limpar indicador se não houver zeros à esquerda

    call VALcnv
    RETURN

VALcnv ;Iniciação do contador
    clrf TEMP1

Vcnv1
    movfw TEMP3
    subwf HI_TEMP,0
    skpc ;Ignorar instrução seguinte se HI>=0
    goto LCDval2 ;Sair se não for
    bnz Vcnv2

    movfw TEMP2
    subwf LO_TEMP,0
    skpc ; Ignorar instrução seguinte se LO>=0
    goto LCDval2 ; Sair se não for

Vcnv2
    movfw TEMP3
    subwf HI_TEMP,1 ;HI=HI-TEMP3
    movfw TEMP2
    subwf LO_TEMP,1 ;LO=LO-TEMP2
    skpc ;Ignorar instrução seguinte se LO>=0
    decf HI_TEMP,1 ;Decrementar HI
    incf TEMP1,1 ;Incrementar contador
    bsf Bcheck,0
    goto Vcnv1

```

O programa principal demonstra como usar o display LCD e gerar novos caracteres. No início do programa, nós necessitamos de declarar as variáveis **LCDbuf** e **LCDtemp** usadas pelos subprogramas para o LCD, bem como o porto do microcontrolador a que o LCD vai ser ligado.

O programa escreve a mensagem 'characters:' na primeira linha e apresenta também dois caracteres especiais '~' e '}'. Na segunda linha mostra-se 'mikroElektronika'.



LCD.asm

```

;***** Escolher e configurar o microcontrolador *****

PROCESSOR 16F84
#include "p16f84.inc"

    _CONFIG_CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC

;***** Declarar variáveis *****

    Cblock 0x0C          ; Início da RAM
    LCDbuf              ; Pertence às macros 'LCDxxx'
    LCDtemp
    WCYCLE              ; Pertence à macro 'WAITX'
    PRESCwait
    Pointer             ; Ponteiro para os caracteres da mensagem
    endc

;***** Declarar o hardware *****

    LCDtris equ    TRISB
    LCDport equ   PORTB

;***** Estrutura da memória de programa *****

    ORG    0x00    ; Vector de reset
    goto  Main

    ORG    0x04    ; Vector de interrupção
    goto  Main    ; Sem rotina de interrupção

Messages          ; Início das mensagens

    movwf PCL

                ; Mostrar mensagens

Message1 dt "mikRoEleKtrOnlkA"
Message2 dt "bla, bla"
Message3 dt "example"

END_messages          ; Fim das mensagens

#include "bank.inc"    ; Ficheiros auxiliares
#include "wait.inc"
#include "lcd.inc"
#include "print.inc"

Main                ; Início do programa

    bcf PORTB,2

    LCDinit          ; Iniciação do LCD

    LCDchar 'C'      ; Mostrar os caracteres no LCD
    LCDchar 'a'
    LCDchar 'Y'
    LCDchar 'a'
    LCDchar 'c'
    LCDchar 't'
    LCDchar 'e'
    LCDchar 'Y'
    LCDchar 's'
    LCDchar ':'
    LCDchar ''

    LCDchar    0x00    ; Mostrar caracteres especiais
    LCDchar    0x01
    LCDline 2

```

Conversor Analógico-Digital de 12 bits

Se tudo no mundo dos microcontroladores é representado por "0's" e "1's", como é que chegamos a um sinal igual a por exemplo 0,5 ou 0,77?

Parte do mundo exterior a um computador consiste em sinais áudio. Além da fala e da música, existem muitas outras grandezas que necessitam de ser introduzidas num computador. Humidade, temperatura, pressão atmosférica, cor e níveis de metanos são outros exemplos.

A resposta é usar um conjunto de linhas digitais e juntá-las de modo a que elas possam "ler" um valor analógico. Um valor analógico é qualquer valor **entre** 0 e 1. Também se lhe pode chamar um "valor fraccionário". Todas as grandezas necessitam de ser convertidas em valores entre 0 e 1 de modo a poderem entrar num computador.

Trata-se de um conceito lato. Que se torna um pouco mais complexo quando tem que ser aplicado.

Se tomarmos 8 linhas e fizermos com que estas aceitem valores binários, a contagem total será 256 (o que corresponde à contagem até 255 mais o valor 0).

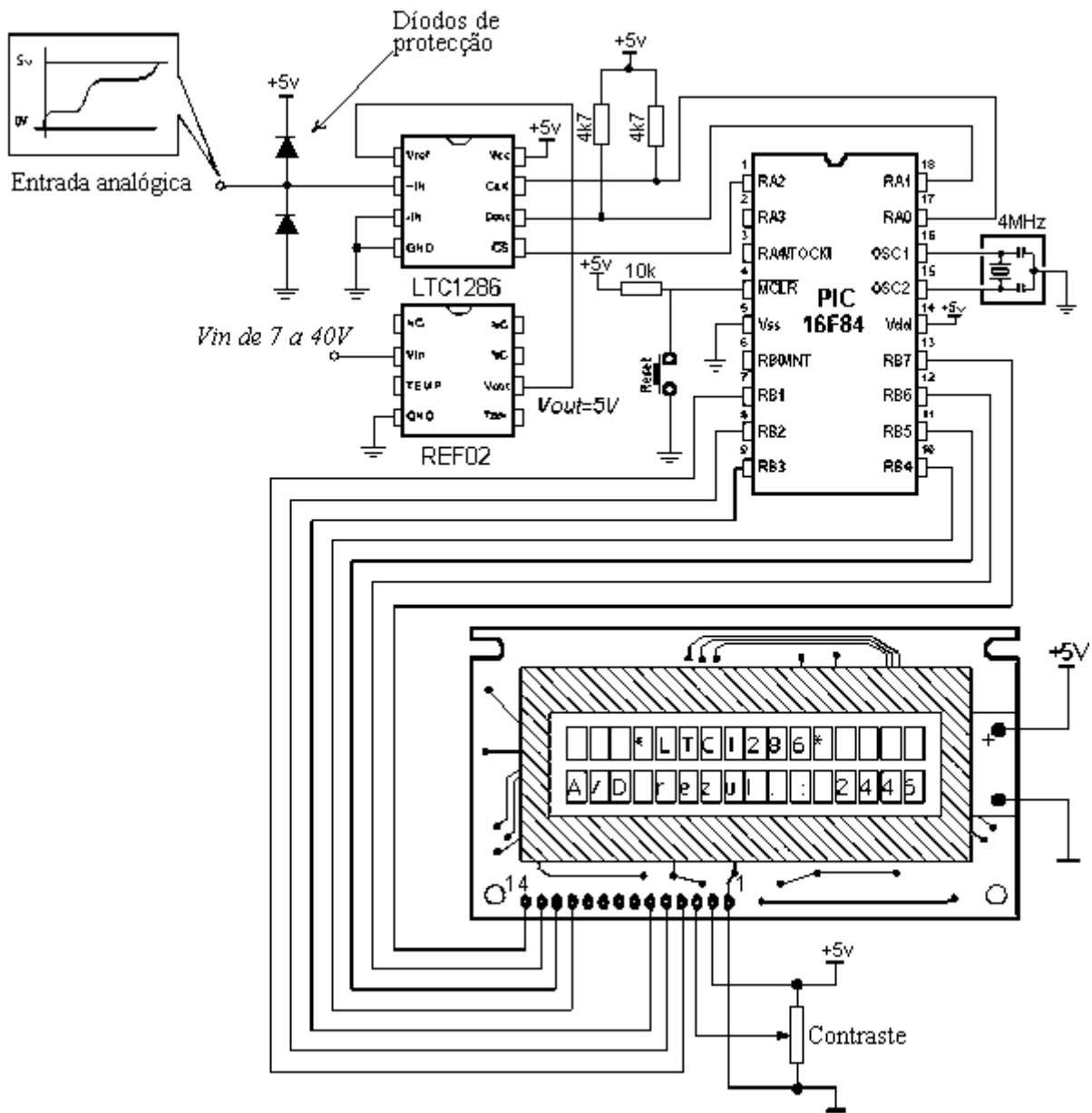
Se juntarmos estas 8 linhas numa "caixa preta", elas passarão a ser designadas como linhas de saída e, assim, temos que arranjar uma linha de entrada. Com esta configuração, nós podemos detectar 255 incrementos entre zero e "1". Esta caixa preta é designada por CONVERSOR e, como estamos a converter um valor **A**nalógico num **D**igital, o conversor é designado por **conversor analógico-digital** ou **ADC (Analog Digital Converter)**.

Os conversores analógicos - digitais podem ser classificados de acordo com diferentes parâmetros. Os parâmetros mais importantes são a **resolução** e o **modo de transferir dados**. Quando falamos de resolução, encontramos conversores de 8 bits, 10 bits, 12 bits, 14 bits e 16 bits. Como os conversores de 12 bits constituem um standard na indústria, o exemplo que vamos analisar diz respeito a um ADC de 12 bits. O outro parâmetro importante é o modo como os dados são transferidos para um microcontrolador. A transferência pode fazer-se em série ou em paralelo. A transmissão em paralelo é mais rápida. Contudo, estes conversores são normalmente mais caros. A transmissão série é mais lenta, mas é mais barata e ocupa menos linhas do microcontrolador, por isso, é a favorita em muitas aplicações.

A grandeza de um sinal analógico pode, muitas vezes, ultrapassar o limite permitido num conversor ADC. Isto pode danificar o conversor. Para proteger a entrada, dois díodos estão ligados como se mostra no diagrama. Esta montagem, vai proteger a entrada do conversor, de tensões acima de 5v e abaixo de 0v.


No nosso exemplo, nós usamos um conversor ADC de 12 bits que é o LTC1286 (Linear Technology). O conversor está ligado ao microcontrolador através de três linhas: data, clock e CS (chip select - selecção de chip). A linha CS é usada para seleccionar um dispositivo de entrada, que é possível seleccionar outros dispositivos (tais como: registo de deslocamento de entrada, registo de deslocamento de saída, conversor analógico digital série) para ligar ao microcontrolador e permitir usar as mesmas linhas de dados.

O circuito em baixo, mostra como ligar um conversor ADC, uma referência e um display LCD a um microcontrolador. O display LCD foi adicionado para mostrar o resultado da conversão AD.



Ligação de um conversor AD com voltagem de referência a um microcontrolador

A Macro usada neste exemplo, chama-se LTC86 e encontra-se no ficheiro LTC1286.inc .



LTC1286.inc

```

LTC86 macro Var_LO, Var_HI, Var

    Local Loop           ;Rótulo local
    Local Loop1

    clrf Var_LO           ; Limpar buffer de dados
    clrf Var_HI

    movlw .4
    movwf Var             ; Iniciação do contador

    bcf CS                ; Habilitar conversor ADC

    call CLK              ; Activar linha Dout
    call CLK              ; manter activa
    call CLK              ; bit de separação

Loop   rlf Var_HI,f      ; Rodar Var_HI um bit para a esquerda

    btfss Data            ; linha Dout = '1' ?
    bcf Var_HI,0          ; não, limpar bit '0' na variável Var_HI
    btfsc Data            ; linha Dout = '0' ?
    bsf Var_HI,0         ; não, pôr a 'um' o bit '0' na variável Var_HI

    call CLK

    decfsz Var,f         ; Recebidos os 4 bits?
    goto Loop            ; não, repetir

    movlw .8
    movwf Var            ; Iniciação do contador

Loop1  rlf Var_LO,f     ; Rodar 'Var_LO', um bit para a esquerda

    btfss Data            ; linha Dout = '1' ?
    bcf Var_LO,0         ; não, limpar bit 0 em Var_LO
    btfsc Data            ; linha Dout = '0' ?
    bsf Var_LO,0         ; não, pôr a 'um' o bit '0' de Var_LO

    call CLK

    decfsz Var,f         ; Recebidos os oito bits?
    goto Loop1          ; Não, repetir

    bsf CS                ; Inibir o conversor AD

endm

CLK
    bsf Clock            ; tempo de clock
    nop
    nop
    nop
    bcf Clock

    return

```

A Macro LTC86 tem três argumentos:

LTC macro Var_LO, Var_HI, Var

A variável **Var_LO** é onde o **byte menos significativo da conversão** é guardado

A variável **Var_HI** é onde o **byte mais significativo da conversão** é guardado

Var contador de ciclos

Exemplo: LTC86 LO, HI, Count

Os quatro bits do valor mais alto estão na variável **HI** e os oito bits menos significativos da conversão estão na variável **LO**. **Count** é uma variável auxiliar para contar o número de passagens no ciclo.

O exemplo que se segue, mostra como as macros são usadas no programa. O programa lê o valor de um conversor ADC e mostra-o num display LCD. O resultado é dado em degraus. Isto é, para 0V, o resultado é 0 e para 5V é 4095.



LTC1286.asm

```

;***** Escolher e configurar o microcontrolador *****

PROCESSOR 16f84
#include "p16f84.inc"

    __CONFIG_CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC

;***** Declarar variáveis *****

Cblock 0x0C          ; Início da RAM
LCDbuf             ; Pertencem às macros 'LCDxxxx'
LCDtemp
WCYCLE            ; Pertence à macro 'WAITX'
PRESCwait

TEMP1              ; Pertencem à macro 'LCDval_16'
TEMP2
TEMP3
LO                  ; bits menos significativos
HI                  ; bits mais significativos
LO_TEMP            ; buffer para LO
HI_TEMP            ; buffer para HI
Bcheck

Count              ; Pertence à macro 'LTC86'
Pointer            ; Ponteiro para os caracteres na mensagem
endc

;***** Declarar o hardware *****

#define Data    PORTA,0
#define Clock   PORTA,1
#define CS      PORTA,2

LCDtris equ    TRISB
LCDport equ    PORTB

;***** Estrutura da memória de programa *****

ORG 0x00          ; Vector de reset
goto Main

ORG 0x04          ; Vector de interrupção
goto Main         ; Não há rotina de interrupção

Messages          ; Início das mensagens

    mowf PCL

                                ; Mostrar mensagens

Message0 dt "** LTC1286 *"
Message1 dt "A/D resul.:"

END_messages     ; Fim de mensagens

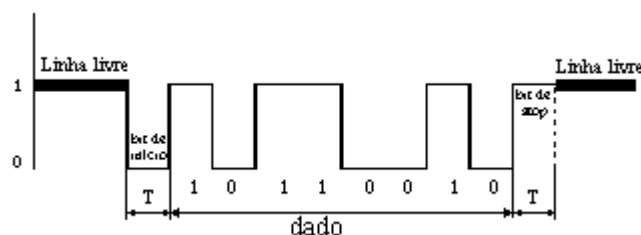
#include "bank.inc"          ; Ficheiros auxiliares
#include "ltc1286.inc"
#include "wait.inc"
#include "lcd.inc"
#include "lcdv16.inc"
#include "print.inc"

Main              ; Início do programa
BANK1
movlw 0xf1        ; Iniciação do Porto A
mowf TRISA        ; TRISA <- 0xf1
BANK0

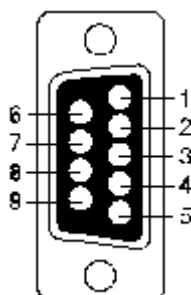
```

Comunicação Série

SCI é a abreviatura para Serial Communication Interface (Interface de Comunicação Série) e existe na maioria dos microcontroladores. No caso do PIC16F84 o SCI não está disponível em hardware, mas pode ser implementado por software.



Tal como no caso da comunicação implementada por hardware, nós vamos usar o standard NRZ (Não Retorno a Zero) e o formato conhecido por 8 (9)-N-1, ou seja, 8 ou 9 bits de dados, sem bit de paridade e com um bit de stop. No caso de **linha livre** (sem estar a transmitir dados) o estado é de **nível lógico 'um'**. O início de transmissão ou **bit de início** (Start Bit), tem o **nível lógico zero**. Os bits que se seguem ao bit de início são os bits de dados (o primeiro bit é o menos significativo) e, finalmente, aparece o **bit de stop** que tem o **nível lógico 'um'**. A duração do bit de stop 'T' depende da velocidade de transmissão e é ajustada de acordo com as necessidades da transmissão. Para uma transmissão à velocidade de 9600 bauds, T tem o valor de 104µS.



Designações dos pinos no conector RS232

Para podermos ligar um microcontrolador a um porto série de um computador PC, nós precisamos de ajustar o nível dos sinais, só assim a comunicação poderá ter lugar. O nível do sinal num PC, é de -10V para o nível lógico um e +10V para nível lógico zero. Como os níveis lógicos num microcontrolador são de +5V para o nível lógico um e 0V para o nível lógico zero, nós precisamos de um andar intermédio para converter estes níveis. Um circuito integrado projectado especialmente para executar este trabalho, é o MAX232. Este circuito integrado recebe sinais de -10 e +10V e converte-os em 5V e 0 V, respectivamente.

O circuito para este interface, mostra-se no diagrama em baixo:



RS232.inc

```

;***** Declarar o hardware *****

        #define RXport PORTB,0
        #define RXtris TRISB,0

;***** Declarar constantes *****

        CONSTANT LF =      d'10'   ; Linha seguinte
        CONSTANT CR =      d'13'   ; Posição de início de linha
        CONSTANT TAB =     d'9'    ; Tabular
        CONSTANT BS =      d'8'    ; Um espaço atrás

;***** Macros *****

RS232init macro
        call    RS_init
        endm

SEND macro S_string          ; "send'x'", enviar caracteres ASCII
        movlw  S_string
        call   SENDsub
        endm

SENDw macro
        call   SENDsub
        endm

RECEIVE macro
        call   RECsub
        endm

;***** Subprogramas *****

RS_init bcf    TXport
        BANK1
        clrf  OPTION_REG
        bcf   TXtris           ; pino Tx
        bsf   RXtris          ; entrada Rx com pull-up
        BANK0
        bsf   TXport          ; O estado inicial na linha Tx é "1" lógico
        movlw b'10010000'
        movwf INTCON          ; Habilitar a interrupção em RB0
        RETURN

SENDsub movwf  TXD             ; Registo de dados associado a Tx
        bcf   TXport          ; bit de início
        movlw 0x08
        movwf RS_TEMP1       ; O número de bits a enviar 9600-8-N-1
        call  S_Wait

SENDa btfsc  TXD,0           ; enviar primeiro o bit menos significativo
        goto SENDb
        bcf   TXport
        goto SENDc

SENDb bsf   TXport
SENDc rrf   TXD,1
        call  S_Wait
        decfsz RS_TEMP1,1
        goto SENDa
        goto SENDd

SENDd bsf   TXport          ; bit de paragem (stop)
        call  S_Wait
        call  S_Wait        ; Re-sincronização
        RETURN

S_Wait movlw  0x1E           ; Pausa entre dois bits
        movwf RS_TEMP2      ; 9600 baud => 104uS tempo de bit de envio
        goto  X_Wait

```

Utilização das macros:

RS232init Macro para iniciar o pino RB0 como linha de transmissão de dados (pino – TX).

Exemplo: RS232init

SEND S_string Para enviar um caracter ASCII. O argumento é o caracter ASCII

Exemplo: SEND `g`

SENDw Enviar o dado contido no registo W.

Exemplo:

```
movlw `t`
```

```
SENDw
```

RECEIVE macro na rotina de interrupção, recebe dados pelo interface RS232 e guarda-os no registo RXD

Exemplo:

```

ORG    0x04
      goto  ISR
ISR    bcf  INTCON,GIE
      btfsc INTCON,GIE
      goto  ISR
      RECEIVE
      :
      :
ISRend bcf  INTCON,INTF
      RETFIE

```

No início do programa principal, nós necessitamos de declarar as variáveis RS_TEMP1, RE_TEMP2, TXD, RXD e o pino TX no microcontrolador. Depois de fazer o reset do microcontrolador o programa envia uma mensagens de boas-vindas para o computador PC: **\$ PIV16F84 na linha \$** e está pronto para receber dados através da linha RX.

Nós podemos enviar e receber dados de e para o computador PC através de um programa de comunicações. Quando o microcontrolador recebe um dado, ele devolve uma mensagem para o monitor: Caracter recebido do PIC16F84: x, confirmando que a recepção teve sucesso.

Programa principal:



RS232.asm

```

;***** Escolher e configurar o microcontrolador *****

PROCESSOR 16f84
#include "p16f84.inc"

    __CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC

;***** Declarar variáveis *****

    Cblock 0x0C      ; Início da RAM
    RS_TEMP1      ; Pertencem às funções 'RS232'
    RS_TEMP2
    TXD
    RXD
    Pointer      ; Ponteiro para os caracteres da mensagem
    endc

;***** Declarar hardware *****

    #define TXport PORTA,3
    #define TXtris TRISA,3

    LCDtris equ TRISB
    LCDport equ PORTB

;***** Estrutura da memória de programa *****

    ORG 0x00      ; Vector de reset
    goto Main

    ORG 0x04      ; Vector de interrupção
    goto ISR      ; Saltar para a rotina de interrupção

Messages

    mowwf PCL      ; Mensagens a mostrar no monitor
                  ; e enviadas pela porta RS232

Message0 dt "Received character from PIC16F84: "
Message1 dt "$ PIC16F84 on line $"

END_messages      ; Fim de mensagens

    #include "bank.inc"      ; Ficheiros auxiliares
    #include "rs232.inc"
    #include "print.inc"

;***** Rotina de interrupção *****

ISR    bcf INTCON,GIE      ; Inibir todas as interrupções
        btfs INTCON,GIE      ; Verificar se estão inibidas
        goto ISR

        RECEIVE      ; Guardar o dado recebido, na variável RX

        SEND TAB      ; Enviar Message0 através da ligação RS232

        PRINT Messages, Message0, Message1, Pointer, SENDw

        movfw RXD      ; Devolver dado recebido para confirmar que
        SENDw      ; a recepção teve êxito

        SEND CR      ; Encostar à esquerda
        SEND LF      ; linha seguinte
        SEND LF

```

© Copyright 2003. i-magazine e mikroElektronika. All Rights Reserved.