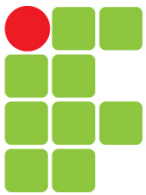
Microcontroladores

Configuração de pinos E/S

Prof. Renan Augusto Starke

Instituto Federal de Santa Catarina – IFSC
Campus Florianópolis
renan.starke@ifsc.edu.br

5 de março de 2020



INSTITUTO FEDERAL
SANTA CATARINA

Ministério da Educação
Secretaria de Educação Profissional e Tecnológica
INSTITUTO FEDERAL DE SANTA CATARINA

Linguagem C em microcontroladores

- Programação usando assembly:
  - Controle maior sobre o desempenho.
  - Útil na construção de rotinas críticas (otimização).
  - Aplicação **não** é portável.
  - Grande esforço de programação, exige conhecimento do microcontrolador e do núcleo.
- Programação usando Linguagem C:
  - Redução do tempo de desenvolvimento.
  - O reuso do código é facilitado.
  - Facilidade de manutenção.
  - Aplicação **mais** portável.

Linguagem C em microcontroladores

- Programação usando Linguagem C:
  - Redução do tempo de desenvolvimento.
    - Foco maior no microcontrolador e na aplicação.
  - O reuso do código é facilitado.
    - Algoritmos são portáveis.
- Porém:
  - Desempenho geral da aplicação depende do compilador e otimizações.
  - Fabricante deve disponibilizar arquivos de cabeçalhos para configuração dos periféricos (endereços de registradores e funções otimizadas).
  - Alguns microcontroladores são tão complexos que o fabricante disponibiliza uma **biblioteca** de acesso e configuração de periféricos.

Linguagem C em microcontroladores

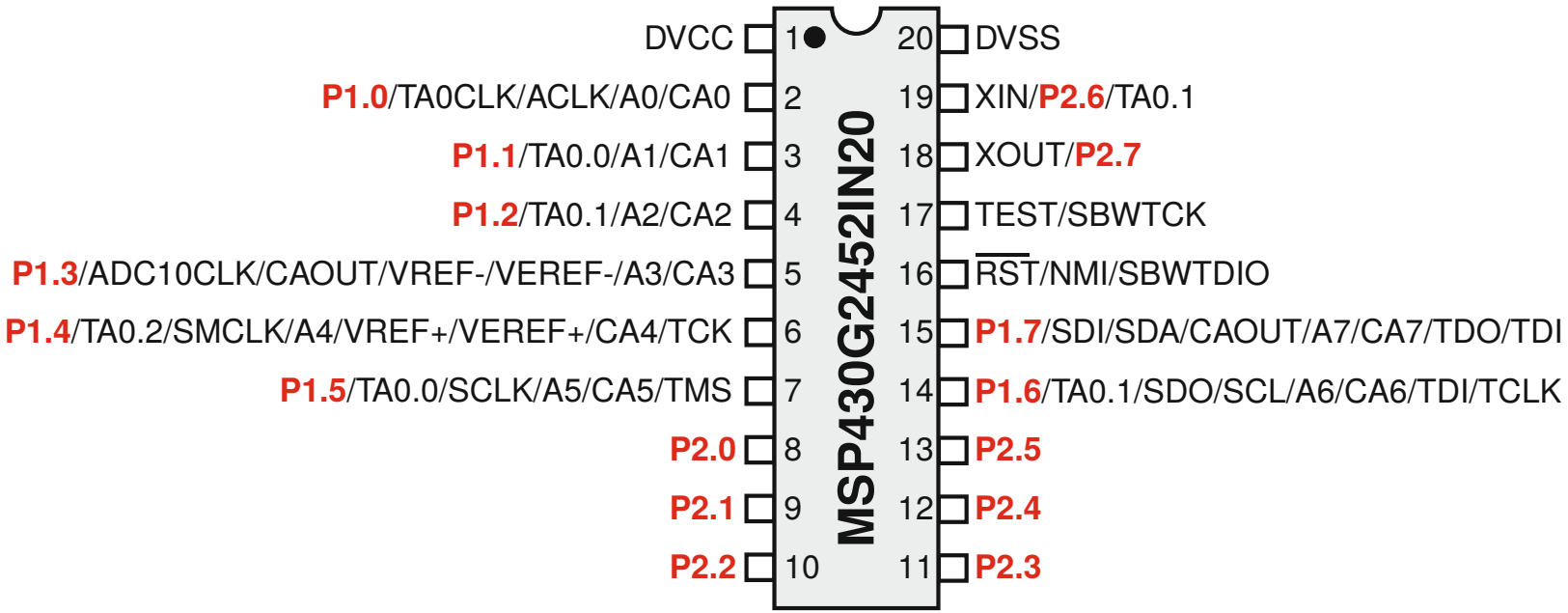
- Comparação código:

```c
#include <msp430.h>

#define LED   BIT0
#define DELAY 5000

int main(void)
{
    int i;

    /* Configuração de hardware */
    WDTCTL = WDTPW | WDTHOLD;
    P1DIR |= BIT0;

    /* main não pode retornar */
    while(1){
        /* Liga/Desliga LED */
        P1OUT = P1OUT ^ LED;

        /* Atraso */
        for (i=DELAY;i--; i > 0);
    }

    return 0;
}
```

```
#include "msp430g2231.h"

LED   EQU 01h
DELAY EQU 5000

#define COUNTER R15

ORG 0f800h

RESET:
        mov.w #0300h, SP
        mov.w #WDTPW+WDTHOLD, &WDTCTL
        bis.b #001h, &P1DIR

main:
        xor.b #LED, &P1OUT
        mov.w #DELAY, COUNTER
L1:
        dec.w COUNTER
        jnz L1
        jmp main


ORG 0FFFEh
DW RESET
END
```

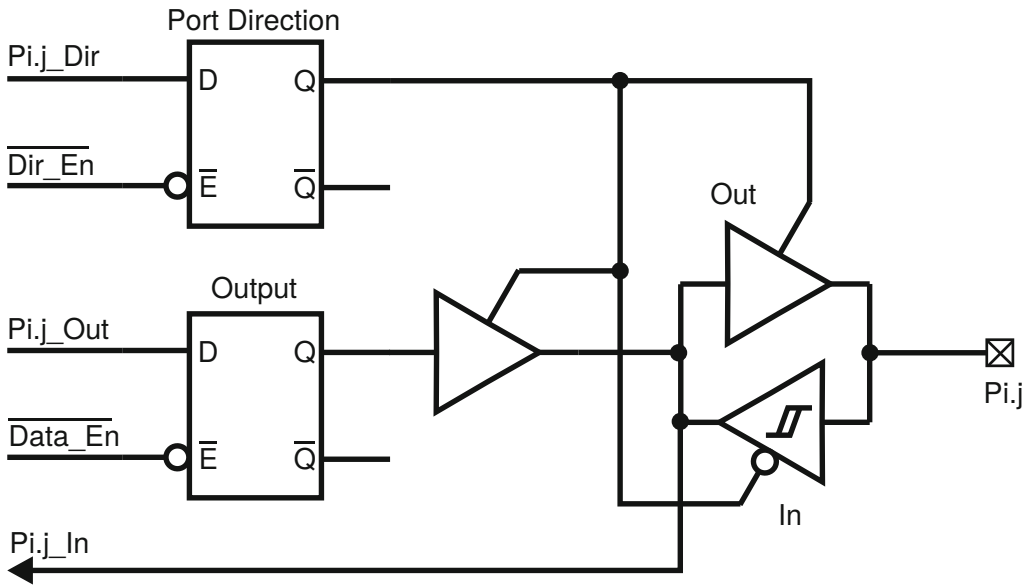
Pinos de entrada e saída

- Pinos de entrada e saída são responsáveis pela interação do microcontrolador com o mundo externo:
  - Acionar um LED.
  - Acionar um motor.
  - Ler um botão.
  - Acionar displays.
  - ...

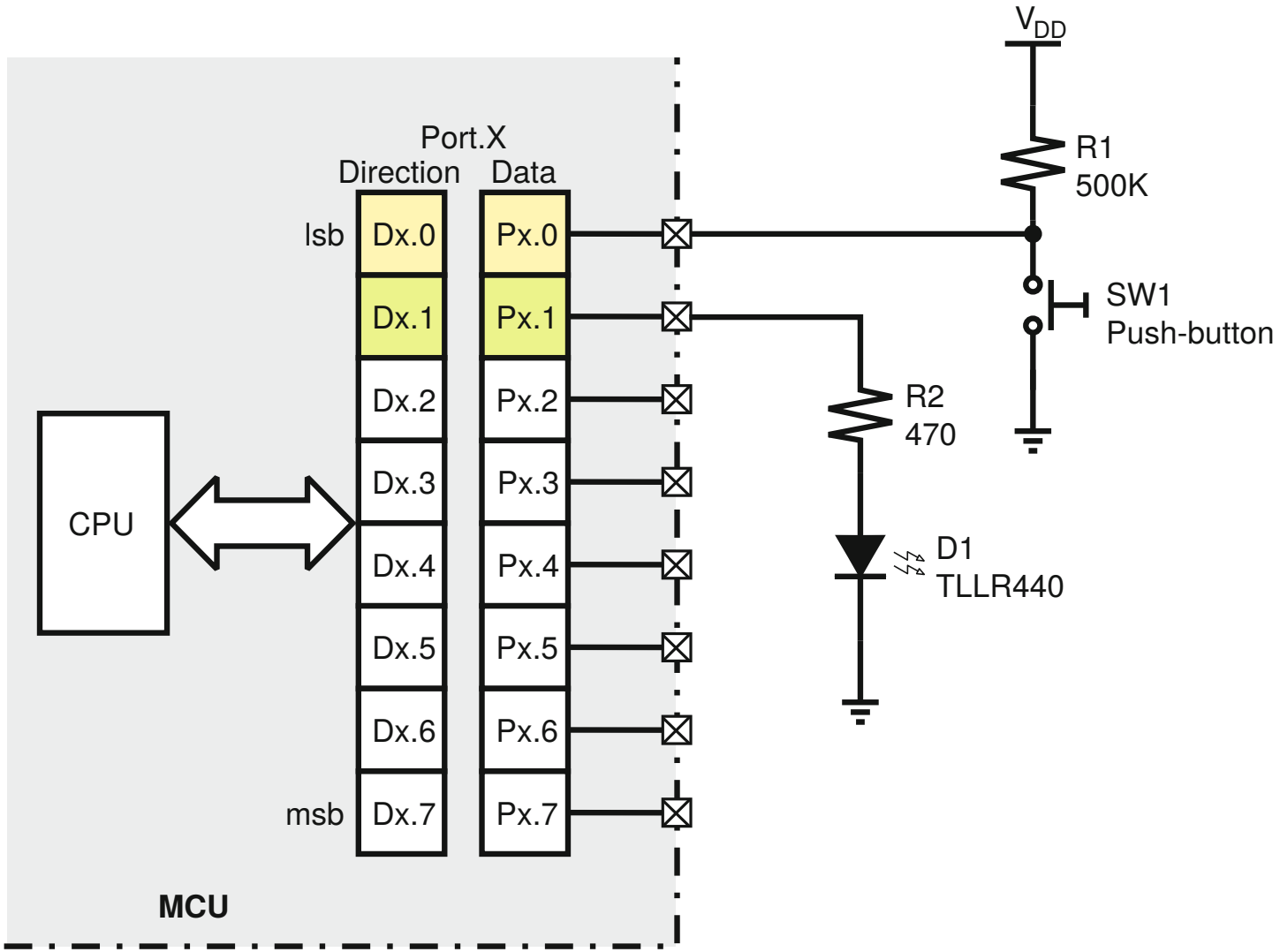


Entradas e saídas digitais: MSP430

- Todas os pinos são "Read-Modify-Write": mudança de configuração de um pino não altera outros.
- Os pinos podem ser configurados como entrada ou saída.
- É possível habilitar resistores internos de pull-up ou pull-down.
- Cada pino pode fornecer/drenar **até 6mA**. A **porta** pode fornecer no máximo **48mA**.

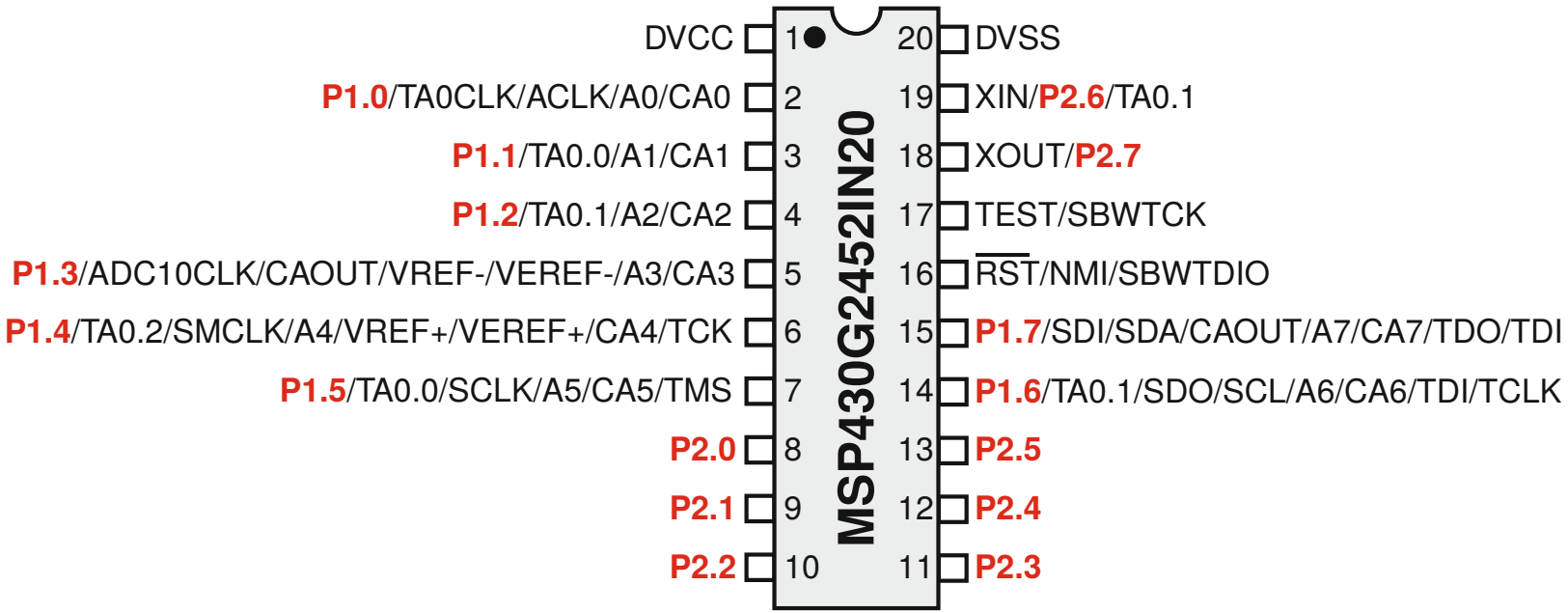


Entradas e saídas digitais: MSP430



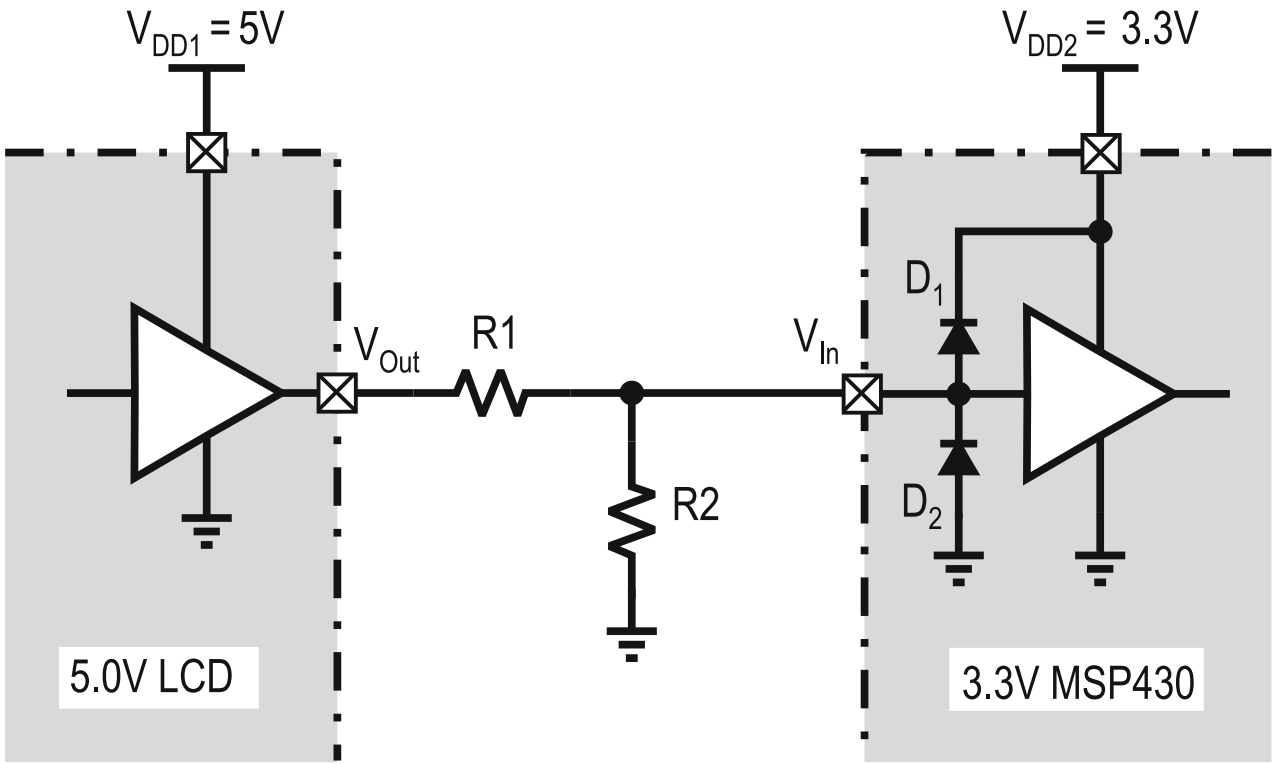
Configuração de Entradas e saídas digitais: MSP430

- Pinos são divididos em portas de 8 pinos (registradores de 8-bits):
  - P1: P1.0, P1.1 ..., P1.7
  - P2: P2.0, P2.1 ..., P2.7
  - Pi.j: número de portas depende do microcontrolador



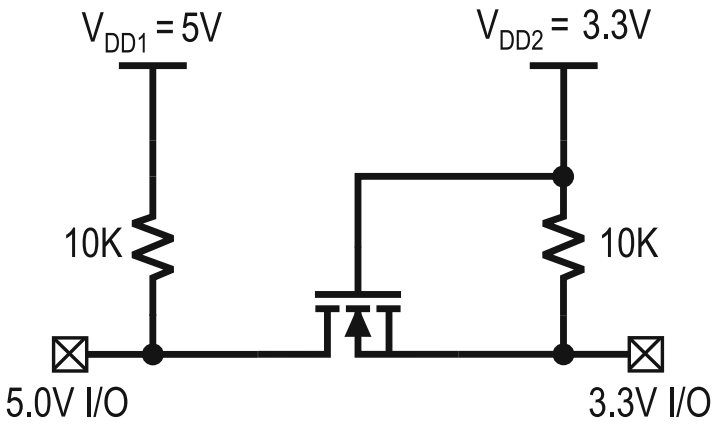
Entradas e saídas digitais: MSP430

- Conversor de nível: divisor resistivo.
- Sistemas unidirecionais e lentos.



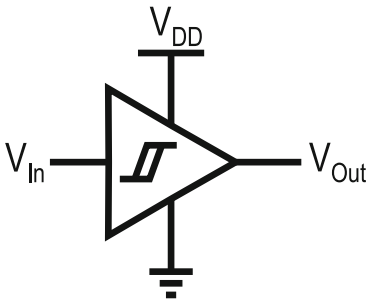
Entradas e saídas digitais: MSP430

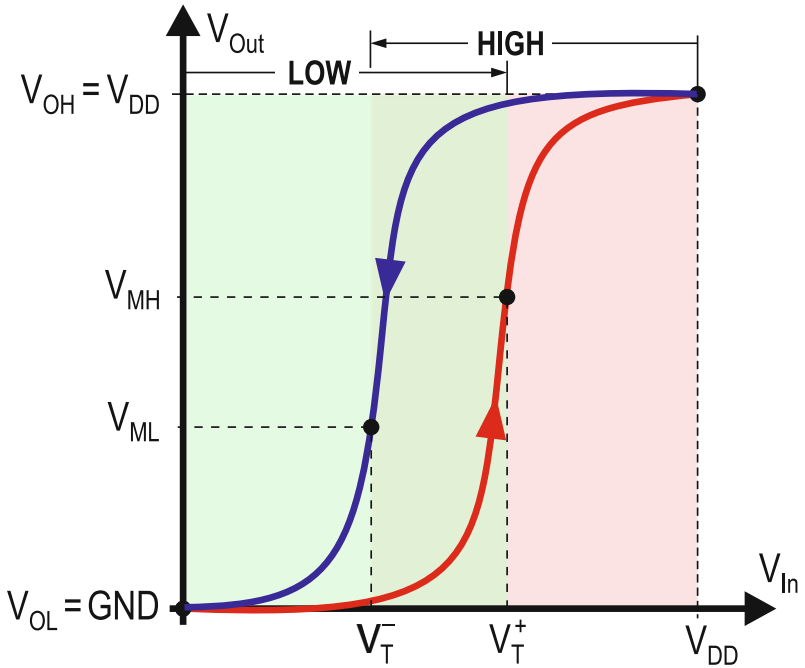
- Conversor de nível bidirecional.
- Sistemas rápidos.



Entradas e saídas digitais: MSP430

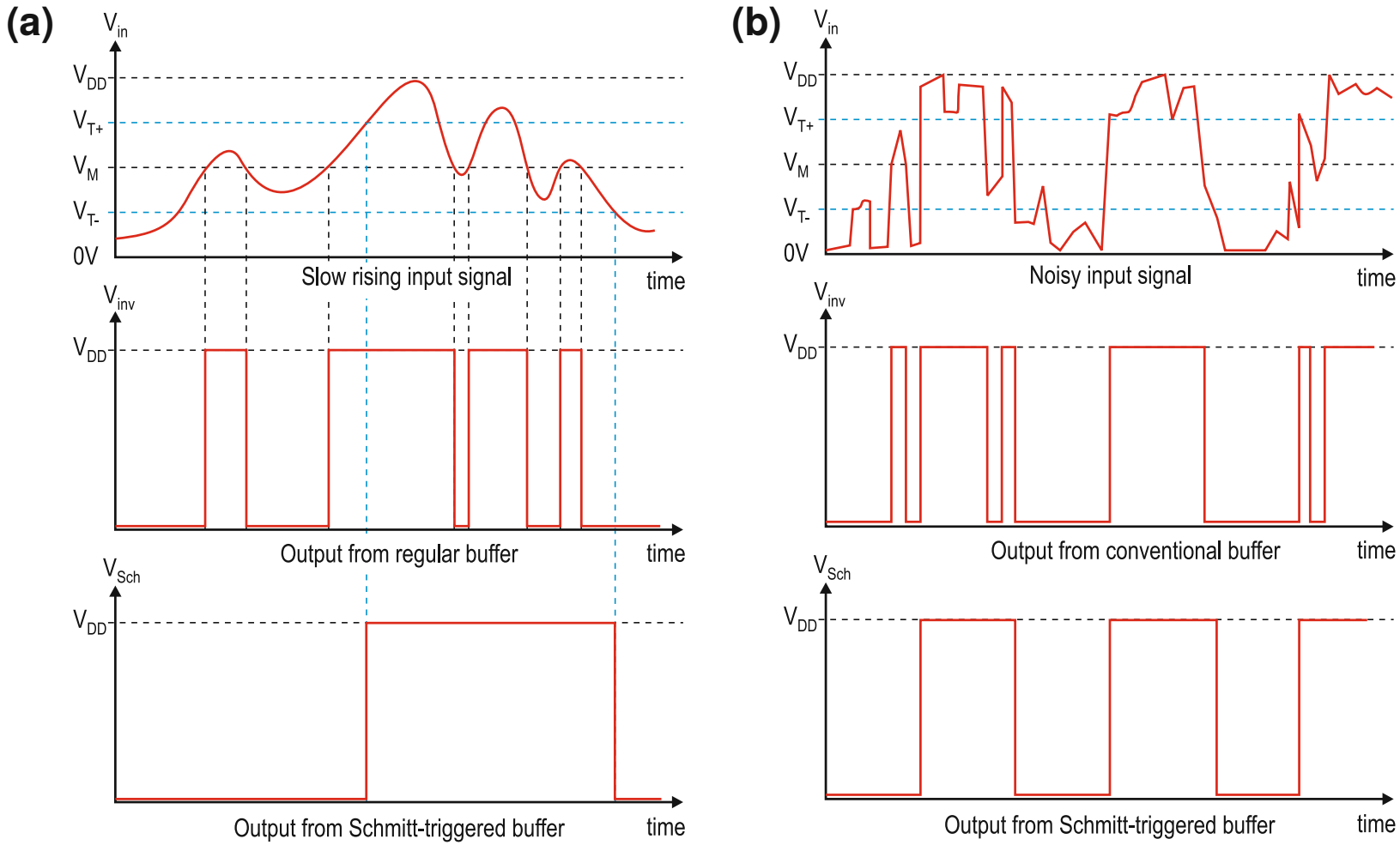
- Buffer com histerese: Schmitt-trigger.





Entradas e saídas digitais: MSP430

- Buffer com histerese: a Schmitt-trigger.

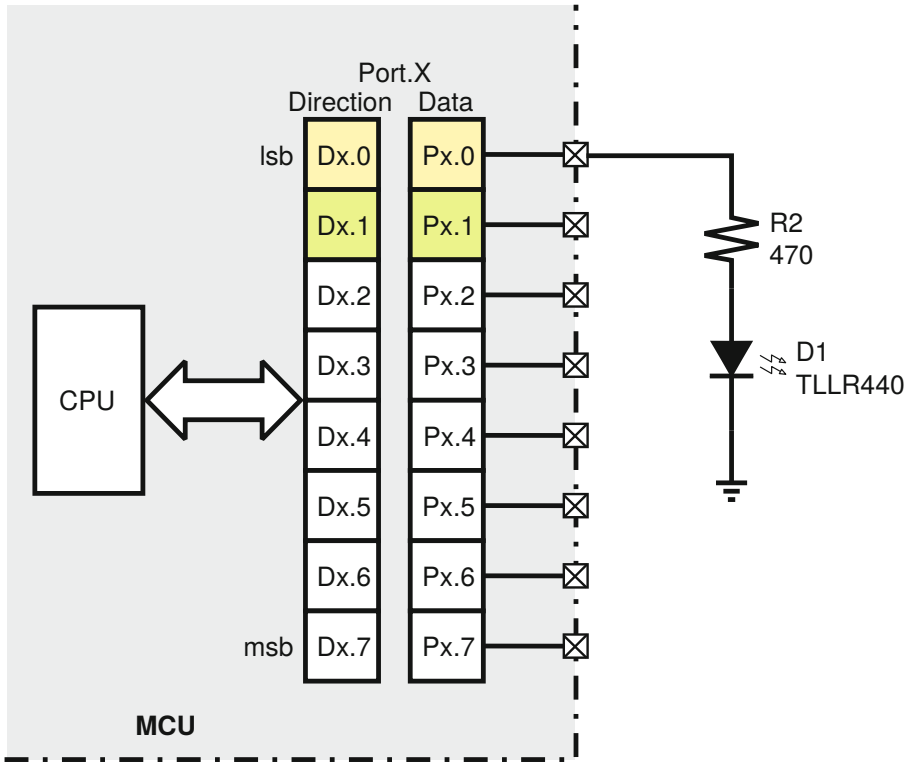


Configuração de entradas e saídas digitais

- Registradores básicos de configuração:
  - PxDIR com $x = (1, 2, 3, ...)$: direção. Cada bit em nível alto configura um pino como saída.
  - PxREN com $x = (1, 2, 3, ...)$: caba bit em nível alto habilita resistor de *pull-up/down* de um pino.
  - PxOUT com $x = (1, 2, 3, ...)$: se pino é saída, altera o nível lógic. Se entrada e pull-up/down habilitado, nível alto habilita pull-up, nível baixo habilita pull-down.
  - PxIN com $x = (1, 2, 3, ...)$: leitura de dados quando configurado como entrada.

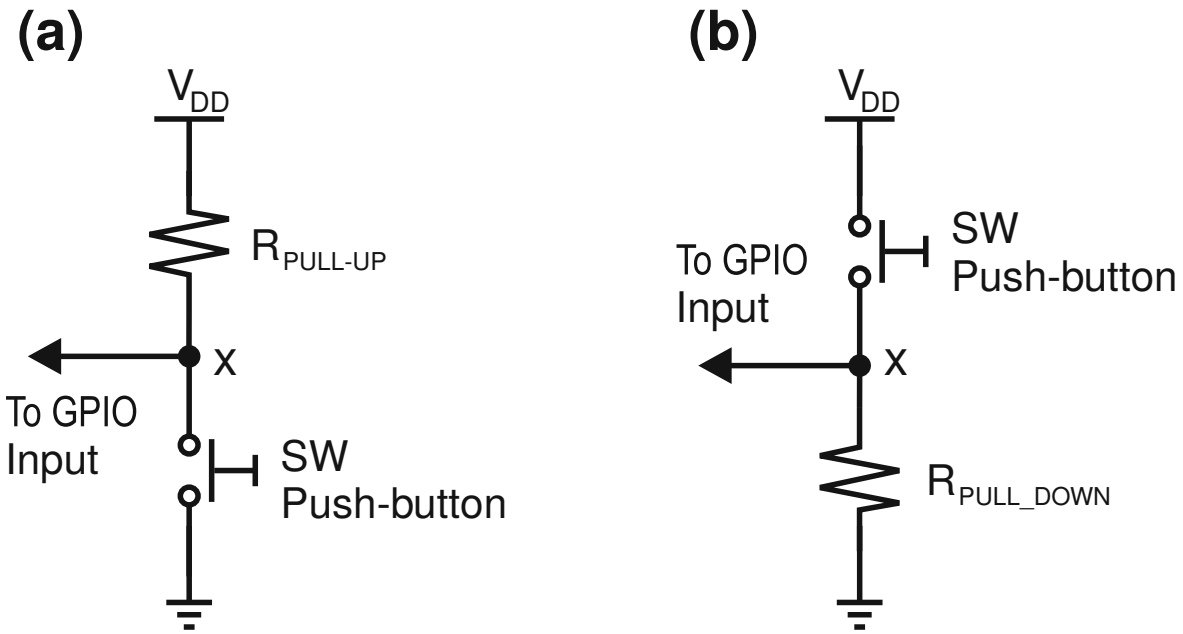
Configuração de saídas digitais

```c
#include <msp430.h>

#define LED   BIT0
#define DELAY 5000

int main(void)
{
    int i;

    /* Configuração de hardware */
    WDTCTL = WDTPW | WDTHOLD;

    /* Pino P1.0 como saída */
    P1DIR |= LED;

    /* main não pode retornar */
    while(1){
        /* Liga/Desliga LED */
        P1OUT = P1OUT ^ LED;

        /* Atraso */
        for (i=DELAY;i--; i > 0);
    }

    return 0;
}
```



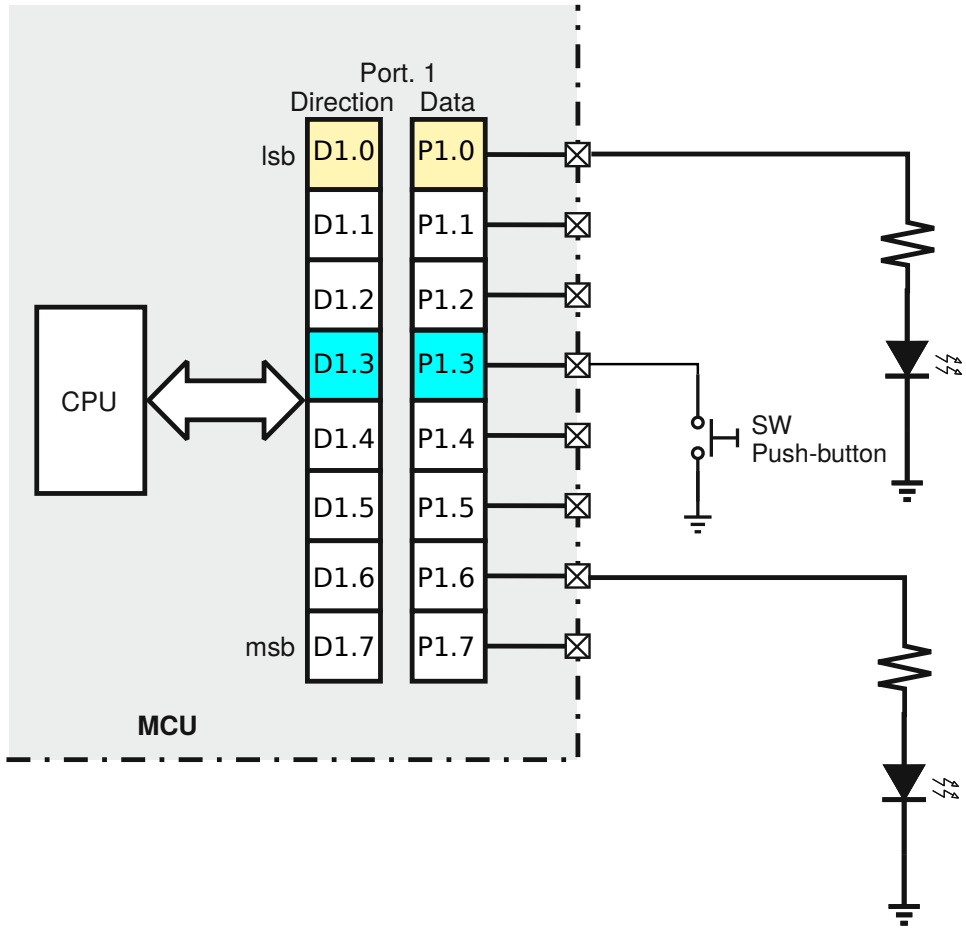
Configuração de entradas digitais: *push buttons*

- Entradas digitais possuem alta impedância.
- Inadequado deixá-las flutuando.



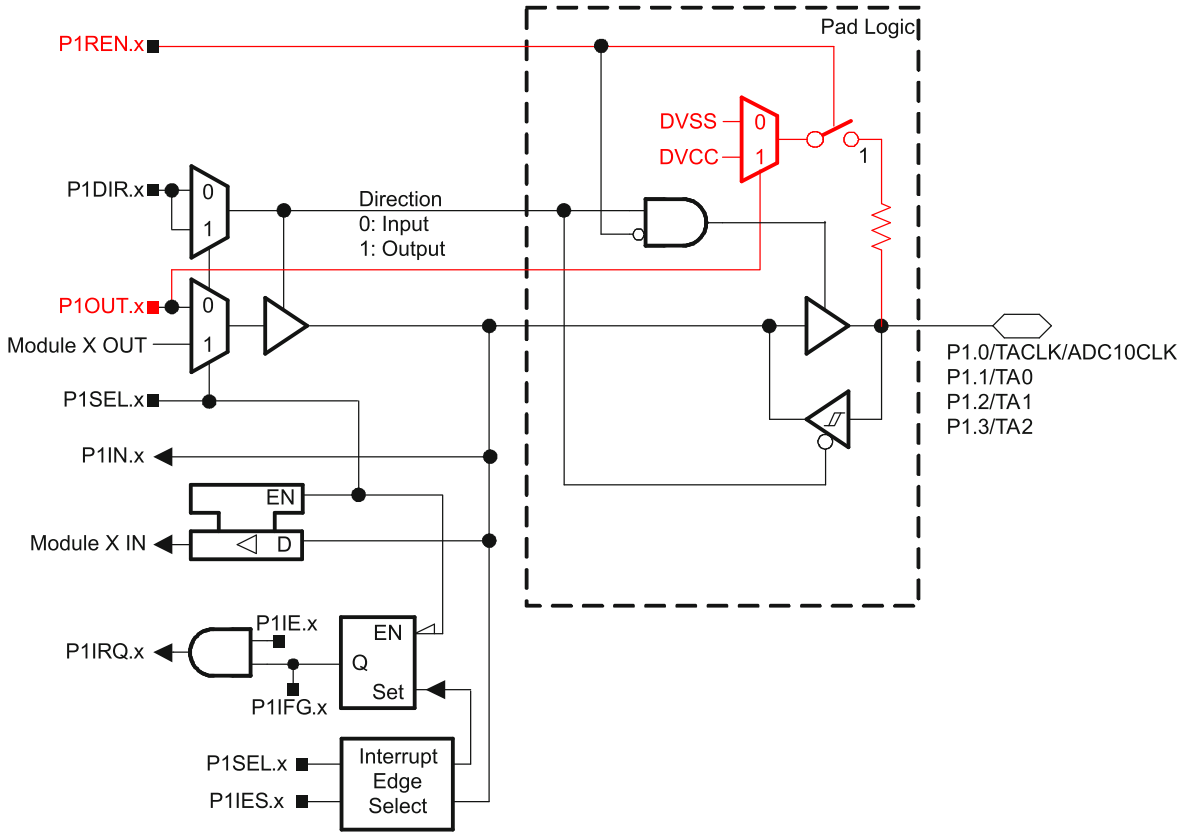
Configuração de saídas digitais

```c
#include <msp430.h>

#define LED_1   BIT0
#define LED_2   BIT6
#define BUTTON  BIT3

void hardware_init()
{
    /* Acesso direto: P1.0 e P1.6 como
     saídas. Demais como entrada      */
    P1DIR |= LED_1 | LED_2;

    /* Habilita resistor de pull up
     * ou pull down */
    P1REN |= BUTTON;
    /* Habilita resitor como pull up */
    P1OUT |= BUTTON;
}
```



Configuração de saídas digitais

```c
#include <msp430.h>

#define LED_1   BIT0
#define LED_2   BIT6
#define BUTTON  BIT3

void hardware_init()
{
    /* Acesso direto: P1.0 e P1.6 como
     saídas. Demais como entrada      */
    P1DIR |= LED_1 | LED_2;

    /* Habilita resistor de pull up
     * ou pull down */
    P1REN |= BUTTON;
    /* Habilita resitor como pull up */
    P1OUT |= BUTTON;
}
```



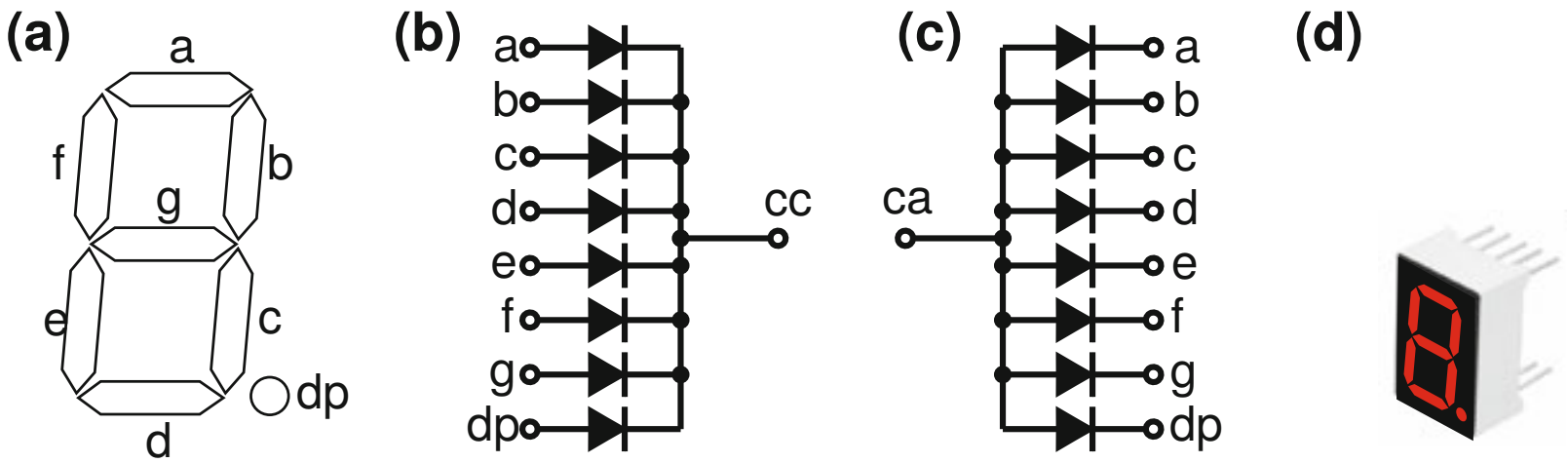

DANGER
MAN AT WORK

Displays de segmentos

- Útil em várias aplicações como interface humano máquina – IHMs.
- Menor custo em relação aos cristais líquidos.

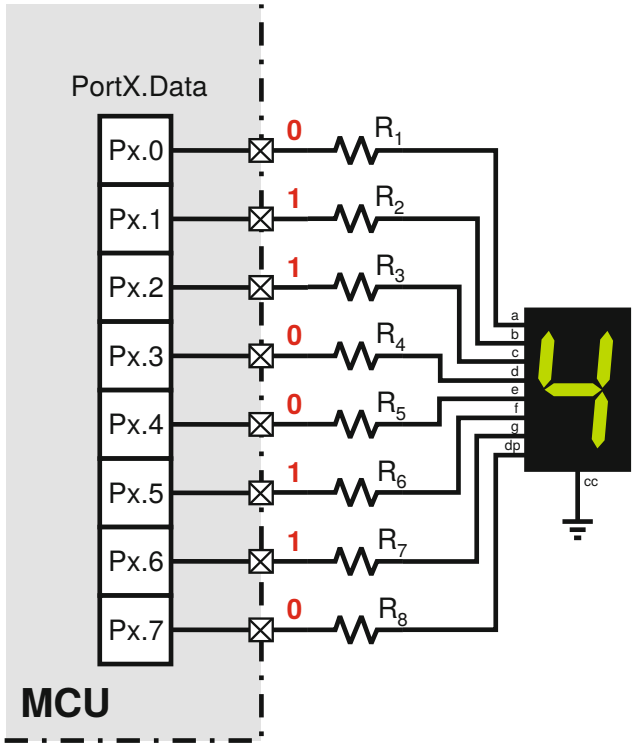
Displays de segmentos

- Displays podem ser anodo ou catodo comum.



Displays de segmentos

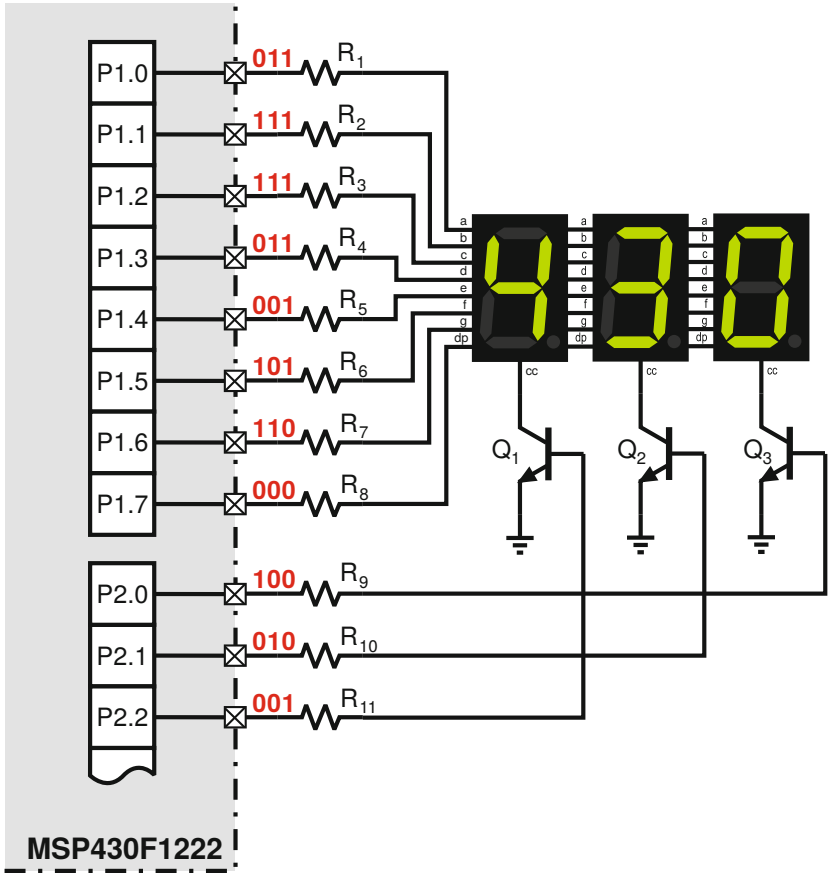
```
/* Tabela de conversão em flash: Anodo comum */
#ifdef COM_ANODO
const uint8_t convTable[] = {0x40, 0x79, 0x24, 0x30, 0x19,
        0x12, 0x02, 0x78, 0x00, 0x18, 0x08, 0x03, 0x46,
        0x21, 0x06, 0x0E};
#endif

/* Tabela de conversão em flash: Catodo comum */
#ifdef COM_CATODO
const uint8_t convTable[] = { ...  };
#endif

void display_init() {

    /* Configuração de portas */
    DISPLAY_PORT_DIR = 0xff;
    DISPLAY_PORT_OUT = 0;

}

void display_write(uint8_t data){

    /* Data não pode ser maior que 0x0f */
    data = data & 0x0f;
    /* Escreve no display */
    DISPLAY_PORT_OUT = convTable[data];

}
```



Display de segmentos

fazer em 24x/segundo - todos vão parecer ligados ao mesmo tempo..

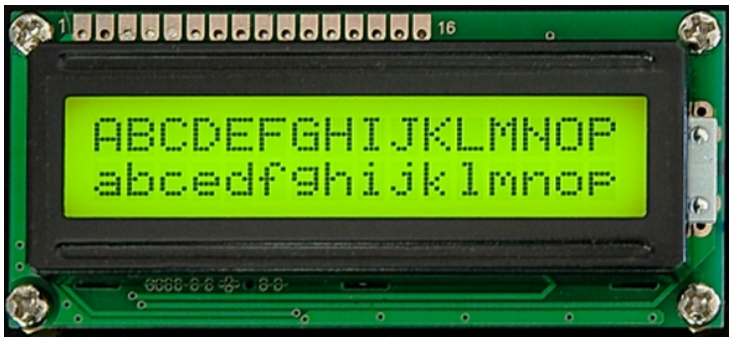
- Desliga displays
- Escreve dígito 1 em P1
- Liga display 1
- Mantém um tempo ligado
- Desliga display 1
- Escreve dígito 2 em P1
- Liga display 2
- Mantém um tempo ligado
- (...)




DANGER
MAN AT WORK

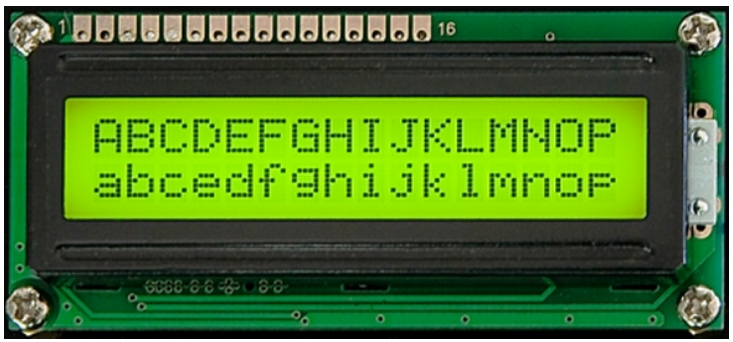
Display LCD

- Módulos LCD são interfaces muito utilizadas em produtos eletrônicos.
- Gráficos:
  - Suportam desenhos, textos, gráficos.
  - Tamanho em pixels (resolução).
- Caractere:
  - Geralmente suportam elementos alfanuméricos.
  - Tamanho em especificado em colunas e linhas.
  - 16 x 2, 16 x 1, 20 x 2, 20 x 4, 8 x 2.

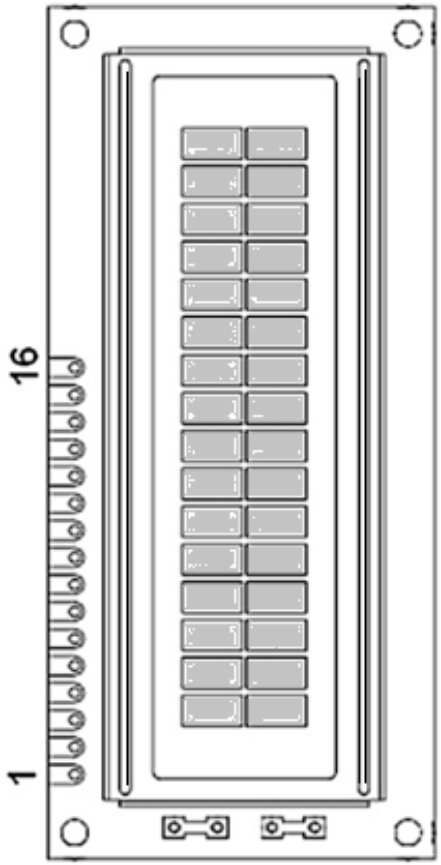


Display LCD

- Controladores específicos responsáveis por ligar/desligar os pixels.
  - Controlador mais comum é o HD44780 da Hitachi.
- Comunicação entre microcontrolador e módulo por barramento paralelo:
  - 8 bits de dados e 3 de controle.
  - 4 bits de dados e 3 de controle.
- Comunicação entre microcontrolador e módulo por de interfaces seriais:
  - Interface SPI.
  - Interface I^2C.



Pinos



Pino	Função	Descrição
1	Alimentação	VSS (GND)
2	Alimentação	VCC
3	VEE	Tensão para ajuste do contraste do LCD
4	RS	*Register Select*: 1 = dado, 0 = instrução
5	R/W	*Read/Write*: 1 = leitura, 0 = escrita
6	E	*Enable*: 1 = habilita, 0 = desabilita
7	DB0	Barramento de dados
8	DB1	
9	DB2	
10	DB3	
11	DB4	
12	DB5	
13	DB6	
14	DB7	
15	LED+ (A)	Anodo do LED de iluminação de fundo
16	LED - (K)	Catodo do LED de iluminação de fundo

Comandos

Instruction	Code RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	Execution Time (max) (when f_{cp} or f_{osc} is 270 kHz)
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter.	
Return home	0	0	0	0	0	0	0	0	1	—	Sets DDRAM address 0 in address counter. Also returns display from being shifted to original position. DDRAM contents remain unchanged.	1.52 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	37 µs
Display on/off control	0	0	0	0	0	0	1	D	C	B	Sets entire display (D) on/off, cursor on/off (C), and blinking of cursor position character (B).	37 µs
Cursor or display shift	0	0	0	0	0	1	S/C	R/L	—	—	Moves cursor and shifts display without changing DDRAM contents.	37 µs
Function set	0	0	0	0	1	DL	N	F	—	—	Sets interface data length (DL), number of display lines (N), and character font (F).	37 µs
Set CGRAM address	0	0	0	1	ACG	ACG	ACG	ACG	ACG	ACG	Sets CGRAM address. CGRAM data is sent and received after this setting.	37 µs
Set DDRAM address	0	0	1	ADD	ADD	ADD	ADD	ADD	ADD	ADD	Sets DDRAM address. DDRAM data is sent and received after this setting.	37 µs
Read busy flag & address	0	1	BF	AC	AC	AC	AC	AC	AC	AC	Reads busy flag (BF) indicating internal operation is being performed and reads address counter contents.	0 µs

Mapeamento de memória

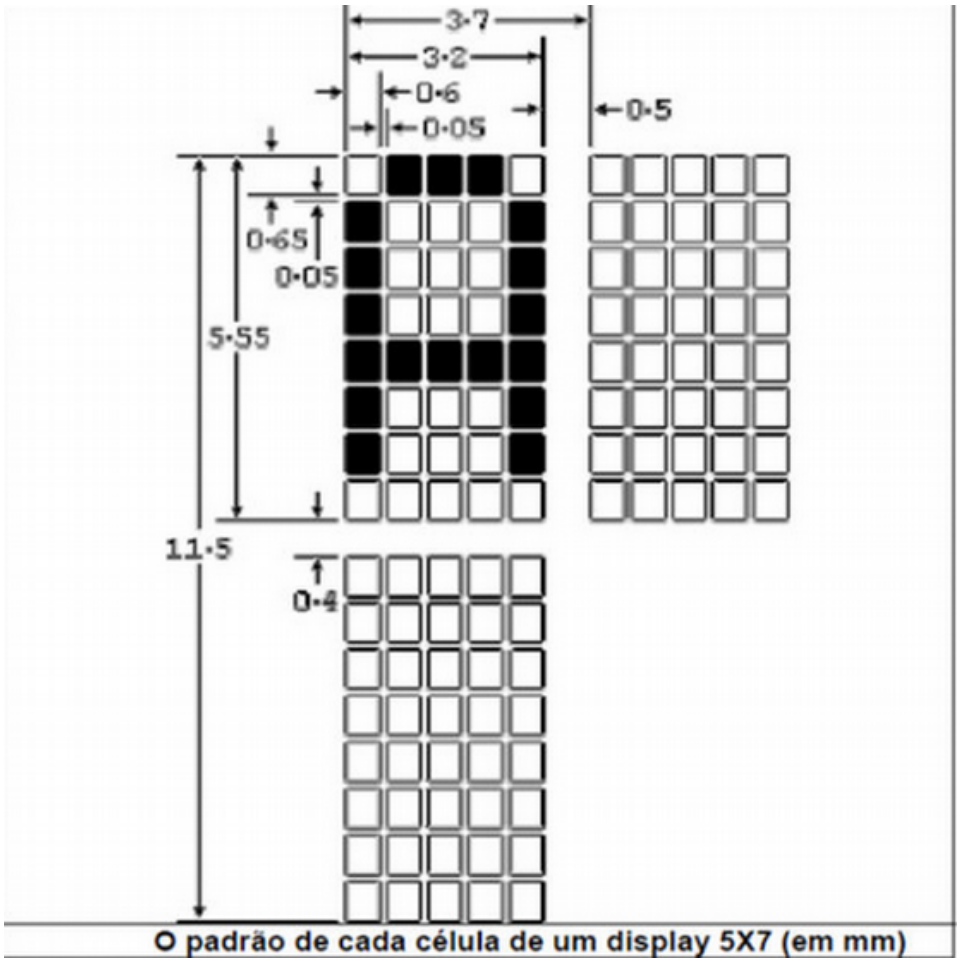
- Caracteres são exibidos escrevendo-os na DDRAM (*Display Data RAM*).
- São escritos caracteres ASCII.
  - O controlador converte o caractere para o desenho em pixels.
- Cada endereço é mapeado em linha e coluna.
- Utiliza-se o comando e a tabela abaixo para mudar a posição dos caracteres.

Display position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
DDRAM address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

Instruction	Code RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	Execution Time (max) (when f_{cp} or f_{osc} is 270 kHz)
Set DDRAM address	0	0	1	ADD	ADD	ADD	ADD	ADD	ADD	ADD	Sets DDRAM address. DDRAM data is sent and received after this setting.	37 µs

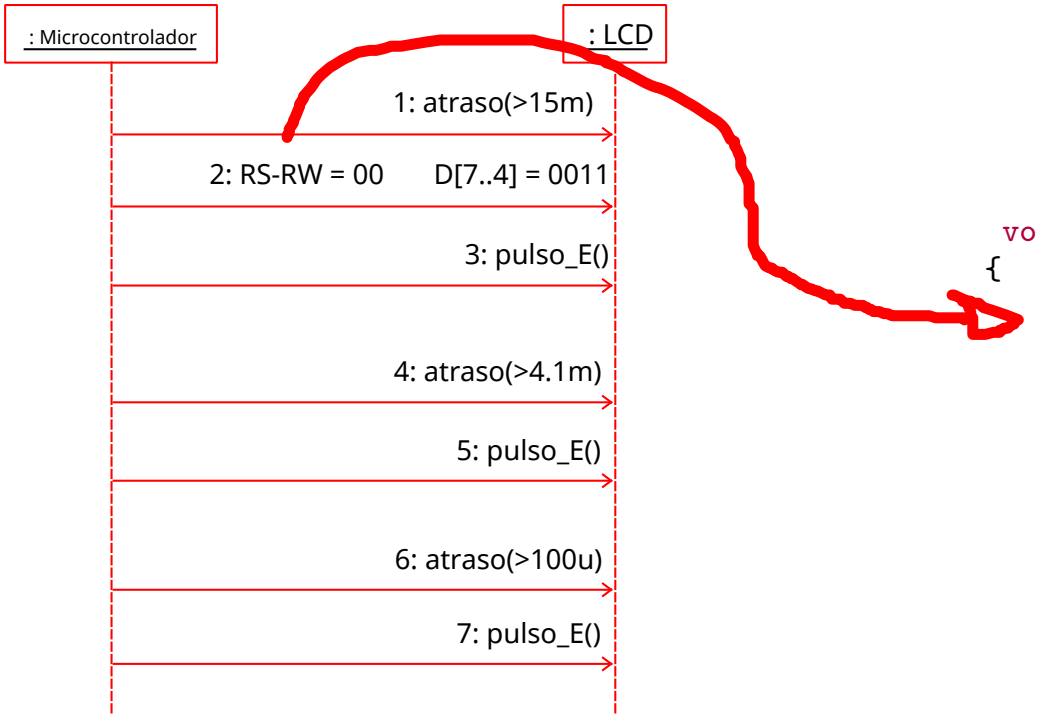
Conversão ASCII para pixels

- Usuário pode desenhar seus próprios caracteres.
  - CGRAM Memory (Character Generator RAM)



Procedimento de inicialização

- LCD sempre está em 8-bits no *power on*.

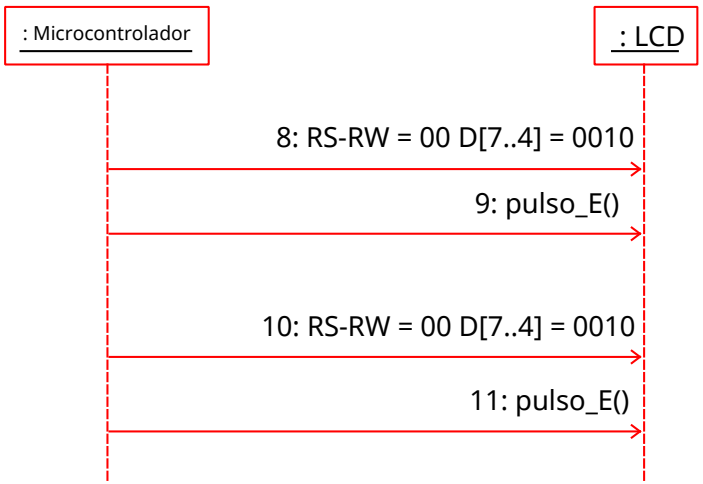


- Comandos abaixo preparam o controlador do módulo.

```
void lcd_init_4bits()
{
    CLR_BIT(LCD_DATA_PORT, RS_PIN | E_PIN);
    __delay_cycles(10000);

    LCD_DATA_PORT |= 0x30;

    enable_pulse();
    __delay_cycles(100);
    enable_pulse();
    __delay_cycles(10000);
    enable_pulse();
```

Procedimento de inicialização

- Após esse ponto, módulo está pronto para executar comandos em 4-bits.

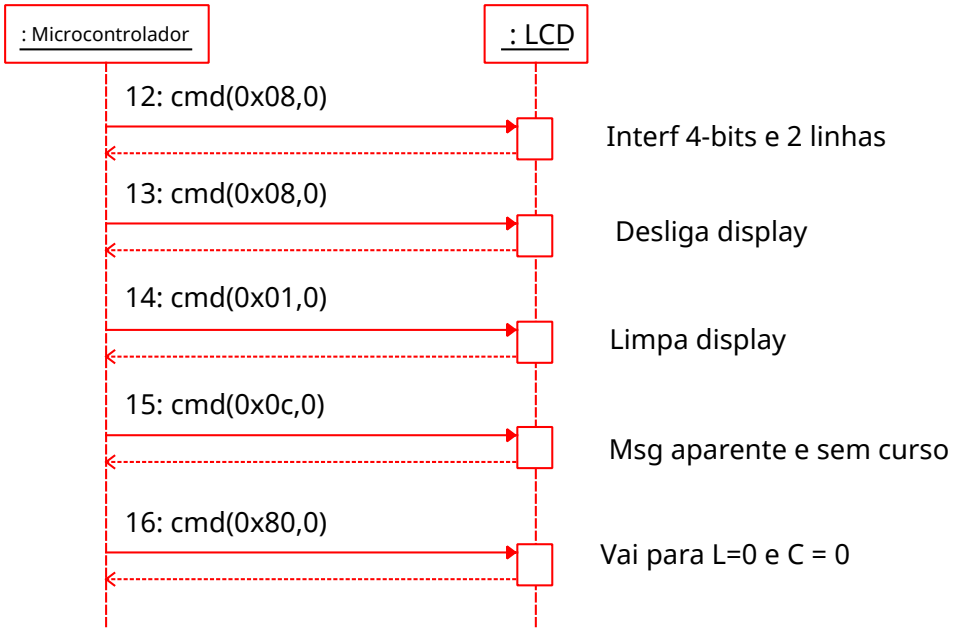



```
void inic_LCD_4bits(){

  (...)


  LCD_DATA_PORT = 0x02;
  pulso_enable();

  LCD_DATA_PORT = 0x02;
  pulso_enable();
```

- Outros métodos de inicialização e maiores detalhes:
  - **datasheet** do módulo!

Procedimento de inicialização

- Após esse ponto, módulo está pronto para executar comandos em 4-bits.



Interf 4-bits e 2 linhas

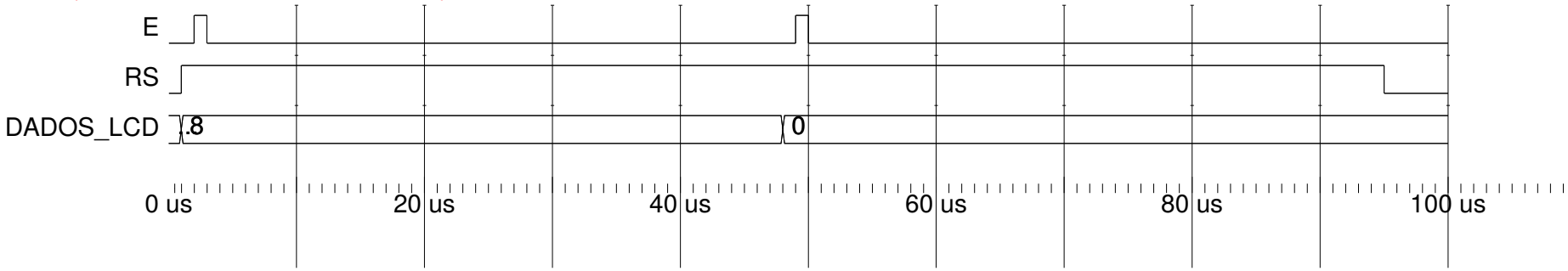
Desliga display

Limpa display

Msg aparente e sem curso

Vai para L=0 e C = 0

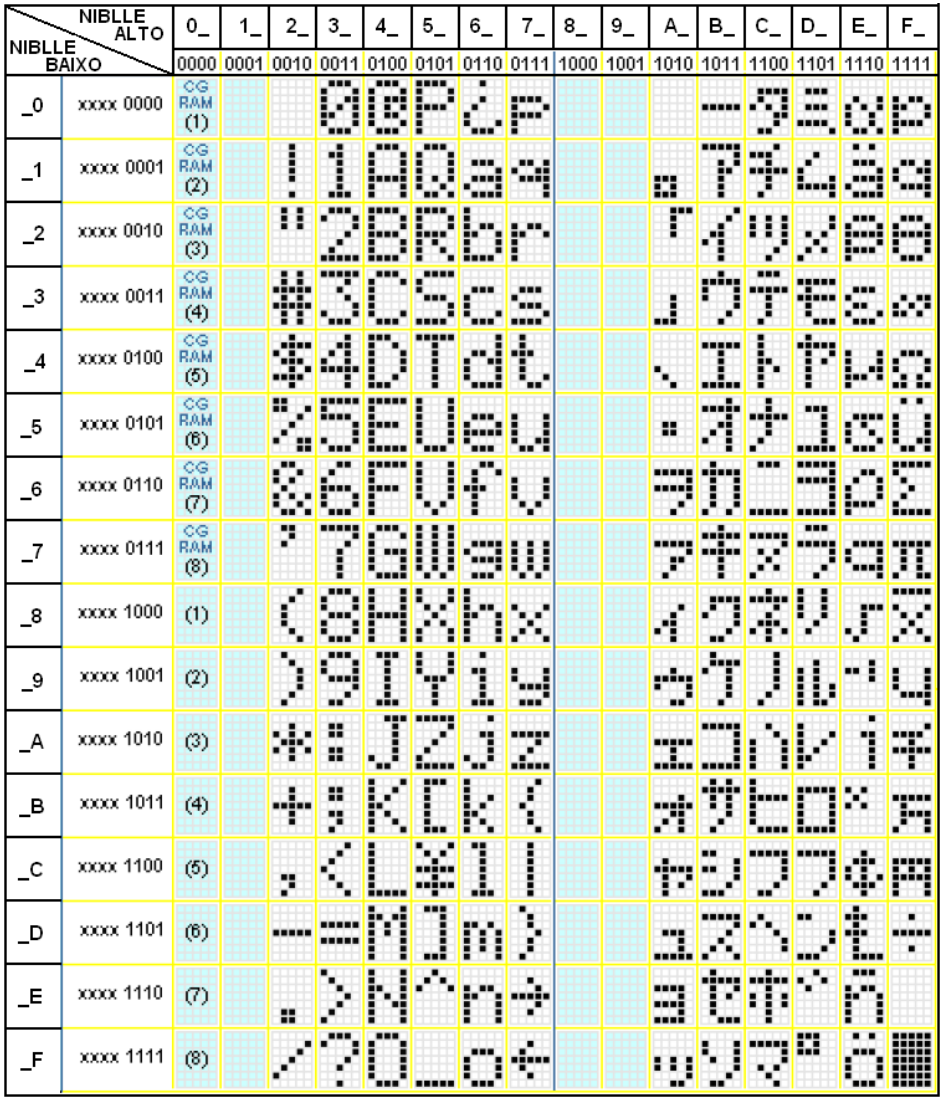
```
void lcd_init_4bits(){

  (...)

  /* 2 linhas: deve ser enviado    *
  após a configuração da interface */
  lcd_send_data(0x28,0);
  /* Desliga o display */
  lcd_send_data(0x08,0);
  /* Limpa todo o display */
  lcd_send_data(0x01,0);
  /* Mensagem aparente cursor  *
   * inativo não piscando      */
  lcd_send_data(0x0C,0);
  /* Inicializa cursor na primeira *
   * posição a esquerda - 1a linha */
  lcd_send_data(0x80,0);
```



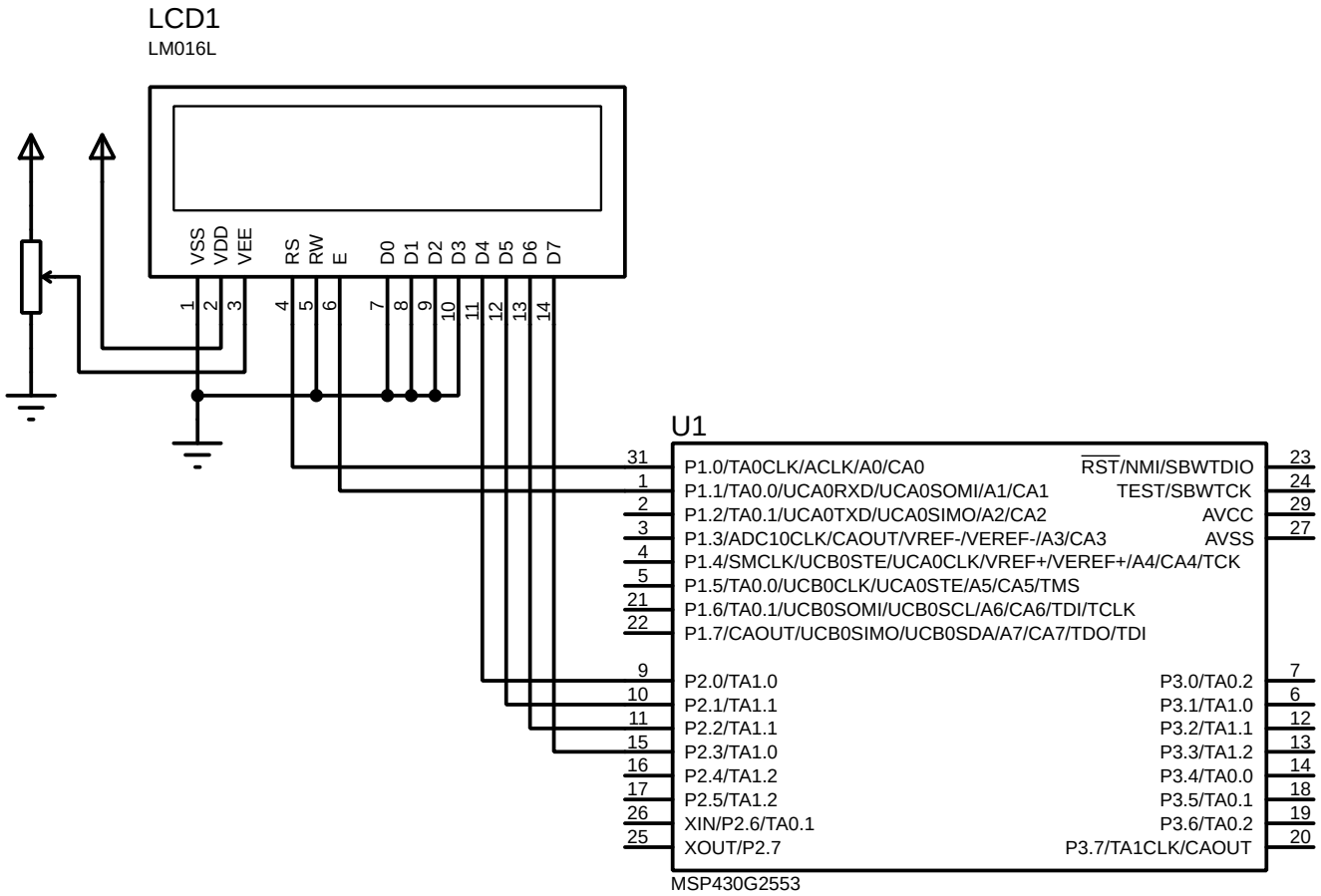
Comandos: interface 4-bits

```
void lcd_send_data(uint8_t data, lcd_data_t data_type)
{
    if (data_type == LCD_CMD)
        CLR_BIT(PORT_OUT(LCD_CTRL_PORT),RS_PIN);
    else
        SET_BIT(PORT_OUT(LCD_CTRL_PORT),RS_PIN);

    /* Envia 4 MSB primeiramente */
#if (NIBBLE_DADOS)
    PORT_OUT(LCD_DATA_PORT) = (PORT_OUT(LCD_DATA_PORT) & 0x0F)|(0xF0 & data);
#else
    PORT_OUT(LCD_DATA_PORT) = (PORT_OUT(LCD_DATA_PORT) & 0xF0) | (data >> 4);
#endif

    enable_pulse();

    /* Envia 4 LSB restantes do byte */
#if (NIBBLE_DADOS)
    PORT_OUT(LCD_DATA_PORT) = (PORT_OUT(LCD_DATA_PORT) & 0x0F) | (0xF0 & (data << 4));
#else
    PORT_OUT(LCD_DATA_PORT) = (PORT_OUT(LCD_DATA_PORT) & 0xF0) | (0x0F & data);
#endif

    enable_pulse();

    /* Delay adicional em caso de instruções lentas: limpeza, etc */
    if ( (data == 0) && ( data_type < 4))
        __delay_cycles(10000);// _delay_ms(2);
```

Caracteres ASCII



Exemplo de ligação eletrônica



Conversão de caracteres

- Display LCD apenas exibe caracteres.
- Para enviar números, deve-se convertê-los para ASCII antes.
  - Se não a *libc* for muito grande:

```
void int2string(int data, char *buffer, int b_size)
{
        int         i = 0;
        int div, j;
        char temp;

        do
        {
                div = data % 10;
                buffer[i] = '0' + div;
                data /= 10;
                i++;
        } while (data > 10 && i < (b_size -1));

        buffer[i] = '0' + data;
        buffer[i+1] = '\0';

        for (j=0; j < i; j++){
                temp = buffer[j];
                buffer[j] = buffer[i];
                buffer[i] = temp;
                i--;
        }
}
```

Conversão de caracteres

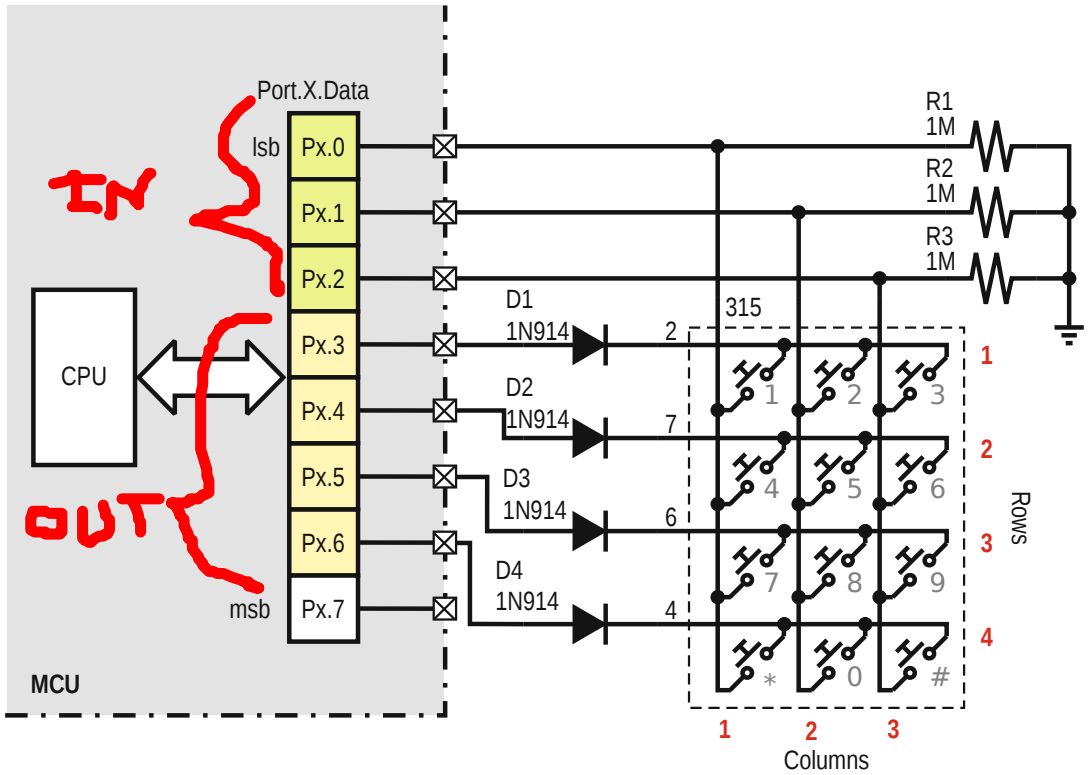
- Funções de conversão *libc*:
  - stdio.h
  - string.h
  - stdlib.h

```c
/* Qualquer dado para string */
int sprintf(char *str, const char *format, ...);

/* String para int */
int atoi(const char *nptr);
```


DANGER
MAN AT WORK

Teclado matricial

- Para poucos botões na aplicação:
  - Um pino de E/S por pino.
- Para vários botões:
  - Multiplexação/varredura.



configuração hw

```
p2dir= 0b01111000;
//direção entrada e saida
p2ren =0;
//entrada em pullup
ou
p2dir = bit6 | bit5 |....;
```


DANGER
MAN AT WORK