Universidade Tecnológica Federal do Paraná – UTFPR
Bacharelado em Ciência da Computação

BCC34C – Sistemas Microcontrolados

Prof. Frank Helbert Borsato

Portas de Entrada e Saída

- **Portas de Entrada e Saída ATmega328**
  - O Atmega328 possui 3 conjuntos de pinos de entrada e saída (I/Os):
    - PORTB, PORTC e PORTD
    - Todos com a função Lê – Modifica – Escreve
- **Rotinas Simples de Atraso**
  - São realizadas fazendo-se a CPU gastar ciclos de máquina na repetição de instruções
- **Ligando um LED**
  - Como o microcontrolador trabalha de acordo com um programa, por mais simples que pareça ligar um LED, existe uma infinidade de possibilidades:
    - O LED pode ser piscado;
    - A frequência pode ser alterada;
    - O número de vezes que se liga e desliga pode ser ajustada;
    - O tempo de acionamento também pode ser ajustado.

Portas de Entrada e Saída

- **Lendo um Botão (Chave Táctil)**
  - Além de se ligar um LED, um dos primeiros programas é ligá-lo ao se pressionar um botão
  - O problema é que na prática, botões apresentam o chamado *bounce*, um ruído que pode ocorrer ao se pressionar ou soltar o botão

Lendo um Botão (Chave Táctil)

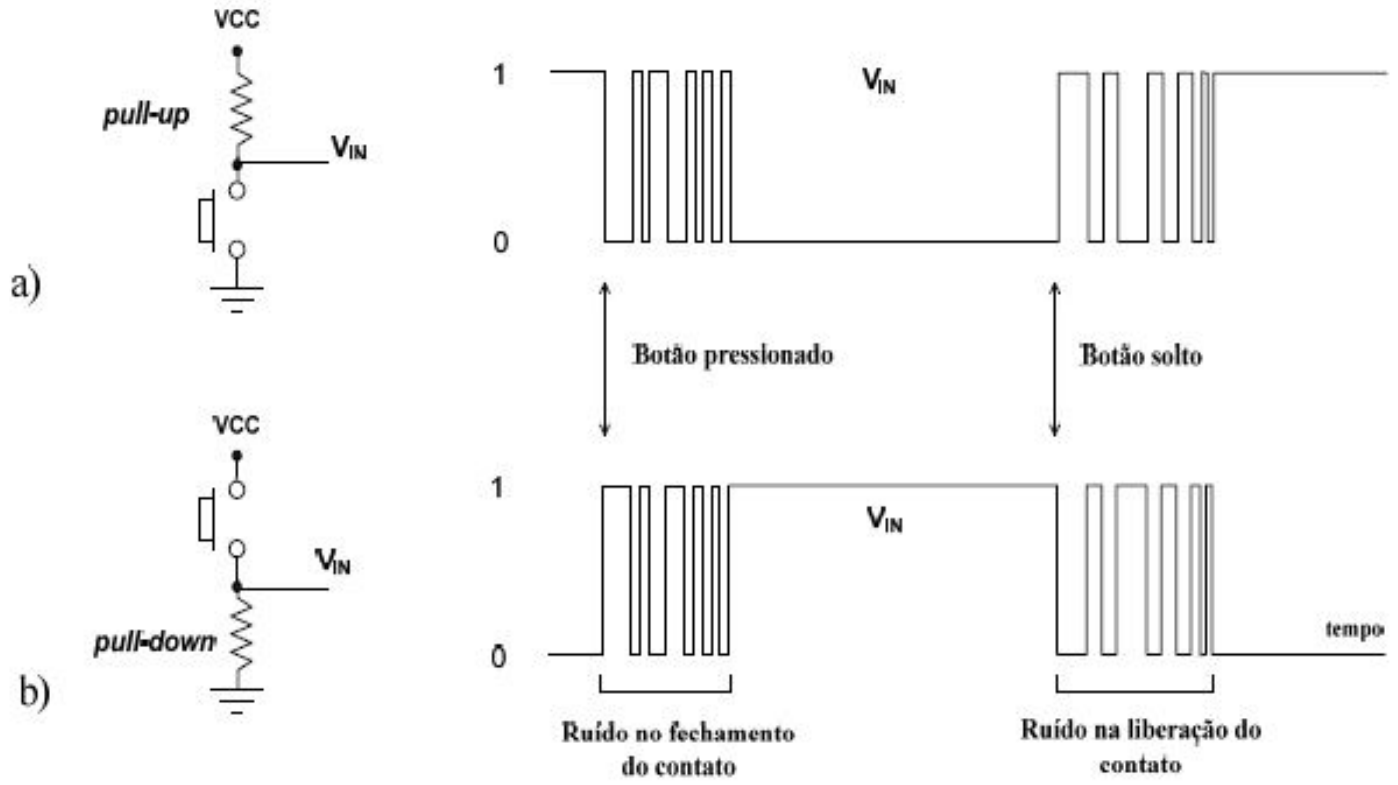
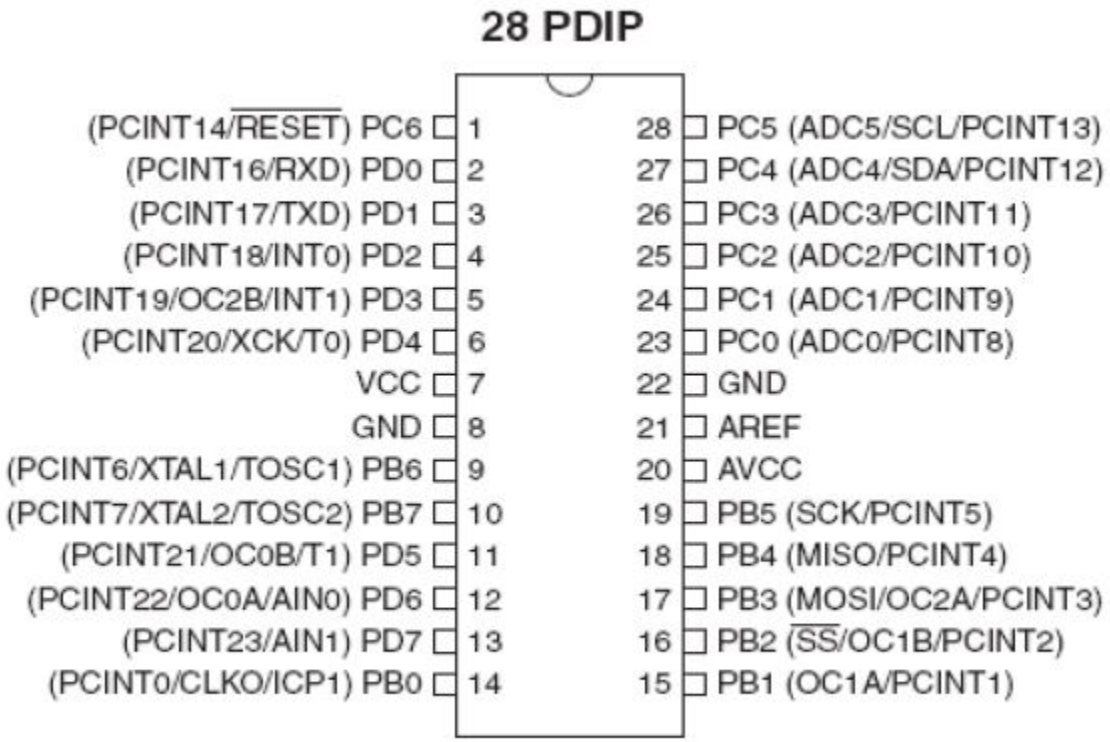


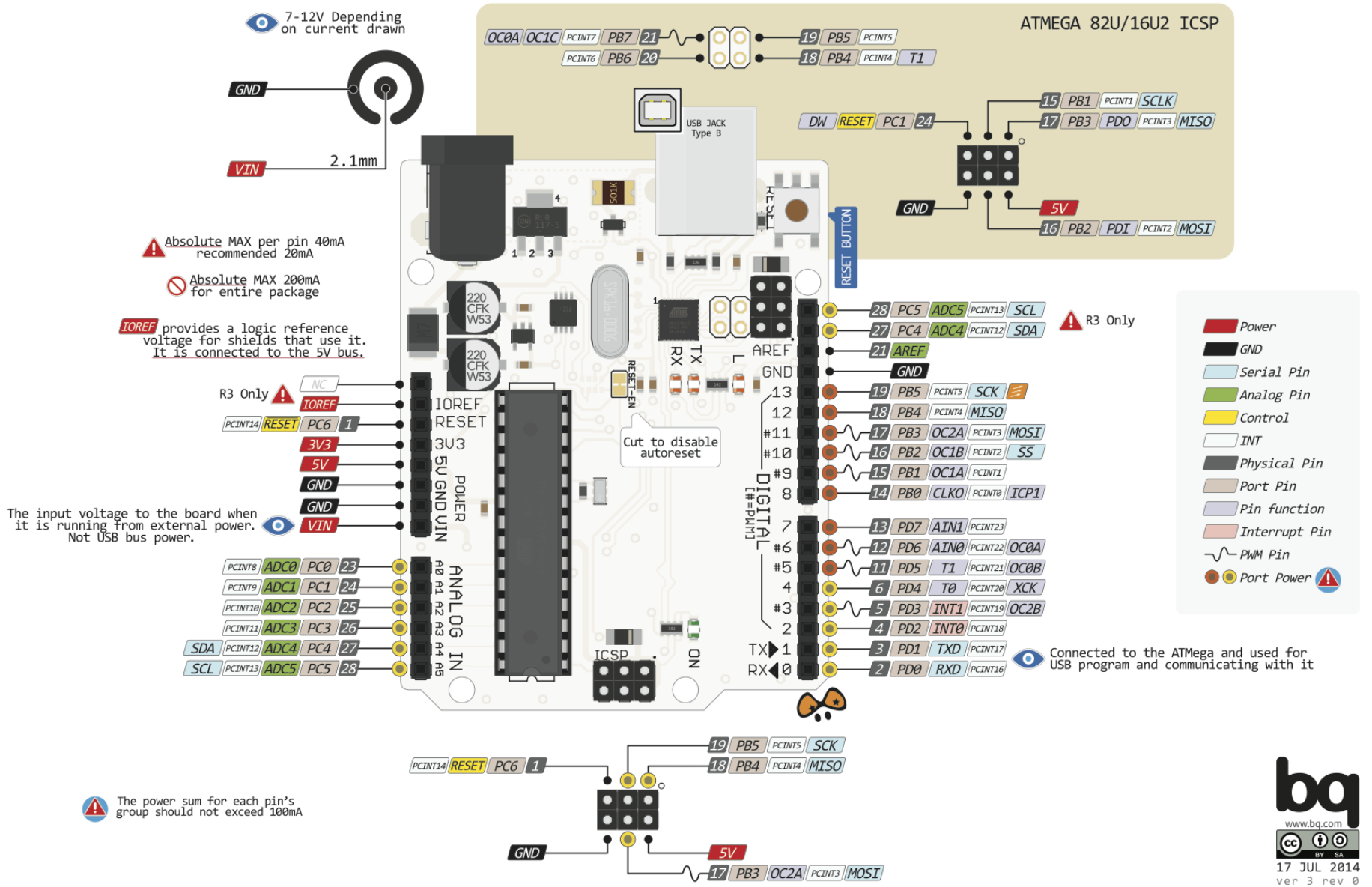
Fig. 5.5 – Exemplo do ruído que pode ser gerado ao se pressionar e soltar um botão: a) usando um resistor de *pull-up* e b) usando um resistor de *pull-down*.

Portas de Entrada e Saída ATmega328

- **PORTB**
  - Pinos PB7...PB0
- **PORTC**
  - Pinos PC6...PC0
- **PORTD**
  - Pinos PD7...PD0



UNO PINOUT



17 JUL 2014
ver 3 rev 0

Portas de Entrada e Saída ATmega328

- **Cada PORT possui um registrador de saída com características simétricas**
- **Com capacidade de drenar ou suprir corrente, suficiente para alimentar LEDs diretamente (20 mA por pino)**
  - O microcontrolador suporta 200 mA
  - Cada PORT suporta 100 mA

Portas de Entrada e Saída ATmega328

- **Todos os pinos tem resistores de pull-up internamente e diodos de proteção entre o VCC e o terra.**

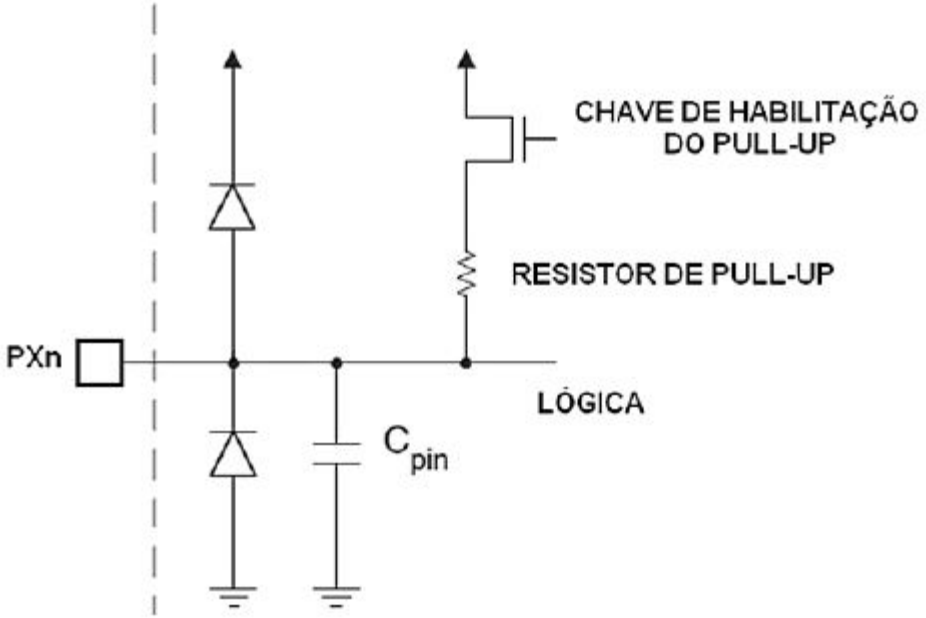


Fig. 5.1 – Esquema geral dos pinos de I/O (PXn) do ATmega.

Portas de Entrada e Saída ATmega328

- **Diferença entre pull-up e pull-down externo:**

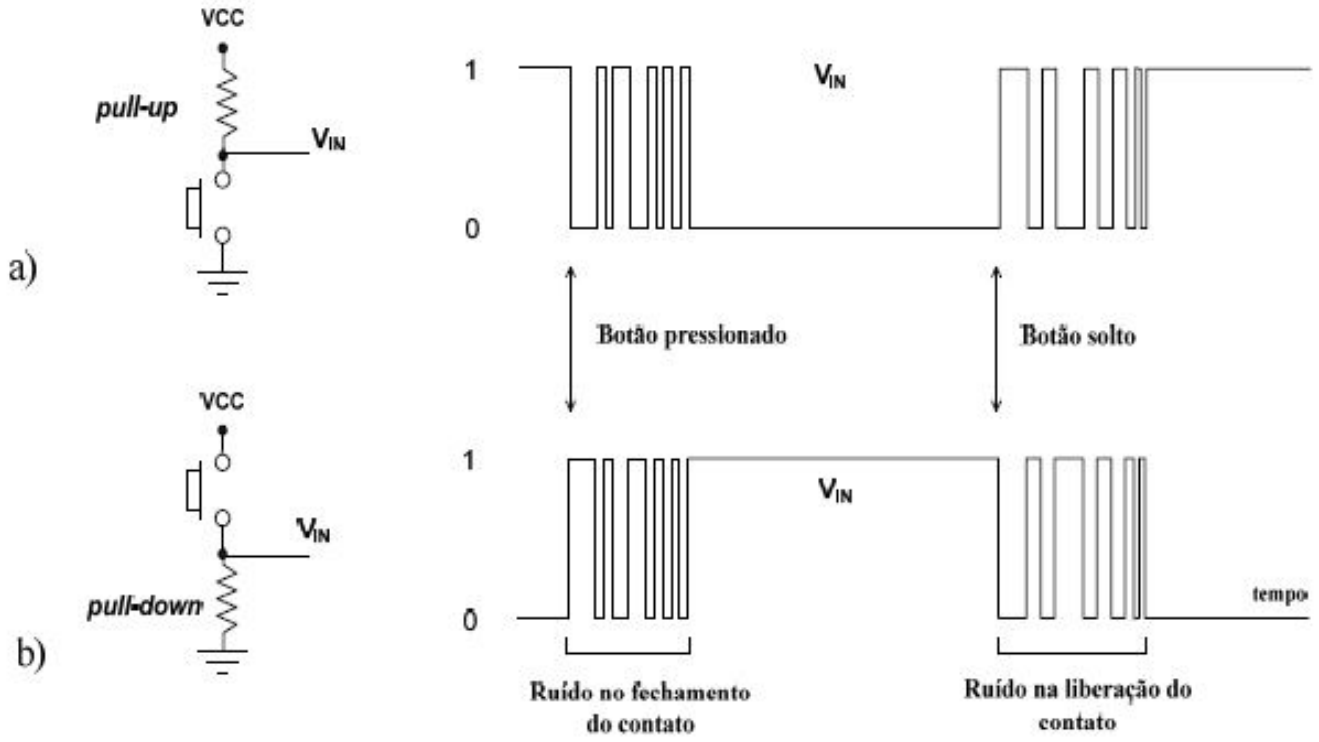


Fig. 5.5 – Exemplo do ruído que pode ser gerado ao se pressionar e soltar um botão: a) usando um resistor de *pull-up* e b) usando um resistor de *pull-down*.

Portas de Entrada e Saída ATmega328

- **Os registradores responsáveis pelos pinos de I/O são:**
  - **PORTx:** (**Port**x Data Register)
    - registrador de dados, usado para escrever nos pinos do PORTx
  - **DDRx:** (**D**ata **D**irection **R**egister)
    - registrador de direção, usado para definir se os pinos do PORTx são entrada ou saída
      - Bit 1: saída (analogia “uma flecha sai”)
      - Bit 0: entrada (analogia “um alvo”)
  - **PINx:** (**P**ortx **In**put Pins Address)
    - registrador de entrada (INput), usado para ler o conteúdo dos pinos do PORTx

Portas de Entrada e Saída ATmega328

PORTs DE I/O

PORTB - PORT B *Data Register*

Bit		7	6	5	4	3	2	1	0
	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
Lê/Escreve		L/E	L/E	L/E	L/E	L/E	L/E	L/E	L/E
Valor Inicial		0	0	0	0	0	0	0	0

DDRB - PORT B *Data Direction Register*

Bit		7	6	5	4	3	2	1	0
	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
Lê/Escreve		L/E	L/E	L/E	L/E	L/E	L/E	L/E	L/E
Valor Inicial		0	0	0	0	0	0	0	0

PINB - PORT B *Input Pins Address*

Bit		7	6	5	4	3	2	1	0
	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
Lê/Escreve		L/E	L/E	L/E	L/E	L/E	L/E	L/E	L/E
Valor Inicial		0	0	0	0	0	0	0	0

Rotinas Simples de Atraso

```
//------------------------------------------------------------------------ //
//    AVR e Arduino: Técnicas de Projeto, 2a ed. - 2012.                   //
//------------------------------------------------------------------------ //


.include "m328pdef.inc" //arquivo com as definições dos nomes
.equ LED   = PB5        //LED é o substituto de PB5 na programação

.ORG 0x000              //endereço de início de escrita do código

INICIO:
   LDI R16,0xFF         //carrega R16 com o valor 0xFF
   OUT DDRB,R16         //configura todos os pinos do PORTB como saída

PRINCIPAL:
    SBI PORTB, LED       //coloca o pino PB5 em 5V
    RCALL ATRASO        //chama a sub-rotina de atraso
    CBI PORTB, LED      //coloca o pino PB5 em 0V
    RCALL ATRASO        //chama a sub-rotina de atraso
    RJMP PRINCIPAL      //volta para PRINCIPAL


Atraso:
    DEC R3              //decrementa R3, começa com o valor 0x00
    BRNE Atraso         //enquanto R3 > 0 fica decrementando R3,desvio para Atraso
    DEC R2              //decrementa R2, começa com o valor 0x00
    BRNE Atraso         //enquanto R2 > 0 volta a decrementar R3
    RET                 //retorno da sub-rotina
//------------------------------------------------------------------------
```

Rotinas Simples de Atraso

- **Para calcular o exato número de ciclos de máquina gastos em uma sub-rotina de atraso é necessário saber quantos ciclos cada instrução consome.**

  Exemplo:

```
Atraso:
    DEC R3          //decrementa R3, começa com o valor 0x00
    BRNE Atraso     //enquanto R3 > 0 fica decrementando R3,
                    //desvio para Atraso
    DEC R2          //decrementa R2, começa com o valor 0x00
    BRNE Atraso     //enquanto R2 > 0 volta a decrementar R3
    RET             //retorno da sub-rotina
```

Rotinas Simples de Atraso

Atraso:

DEC R3
BRNE Atraso $\begin{bmatrix} 1 \text{ ciclo} \\ + 2 \text{ ciclos} \end{bmatrix} \times 255 + \begin{bmatrix} 1 \text{ ciclo} \\ 1 \text{ ciclo} \end{bmatrix} = \begin{bmatrix} 767 \text{ ciclos} \\ + 1 \text{ ciclo} \\ + 2 \text{ ciclos} \end{bmatrix} \times 255 + \begin{bmatrix} 767 \text{ ciclos} \\ + 1 \text{ ciclo} \\ + 1 \text{ ciclo} \end{bmatrix} = 197119 \text{ ciclos}$
DEC R2
BRNE Atraso

Fig. 5.2 – Cálculo preciso da sub-rotina de atraso.

Rotinas Simples de Atraso

Mnemônico	Operandos	Descrição	Operação	Flags	Clocks
DEC	Rd	Decrementa registrador	Rd ← Rd -1	Z, N, V	1
BRNE	k	Desvia se diferente	if(Z=0) PC ← PC + k + 1	Nenhum	1/2
RCALL	k	Chamada de sub-rotina	PC ← PC + k + 1	Nenhum	3
RET		Retorno de sub-rotina	PC ← STACK	Nenhum	4

Apêndice A – ASSEMBLY DO ATMEGA, Livro texto (página 537)

O ATmega328

Conditional Branch Summary

Test	Boolean	Mnemonic	Complementary	Boolean	Mnemonic	Comment
Rd > Rr	Z•(N ⊕ V) = 0	BRLT(1)	Rd ≤ Rr	Z+(N ⊕ V) = 1	BRGE*	Signed
Rd ≥ Rr	(N ⊕ V) = 0	BRGE	Rd < Rr	(N ⊕ V) = 1	BRLT	Signed
Rd = Rr	Z = 1	BREQ	Rd ≠ Rr	Z = 0	BRNE	Signed
Rd ≤ Rr	Z+(N ⊕ V) = 1	BRGE(1)	Rd > Rr	Z•(N ⊕ V) = 0	BRLT*	Signed
Rd < Rr	(N ⊕ V) = 1	BRLT	Rd ≥ Rr	(N ⊕ V) = 0	BRGE	Signed
Rd > Rr	C + Z = 0	BRLO(1)	Rd ≤ Rr	C + Z = 1	BRSH*	Unsigned
Rd ≥ Rr	C = 0	BRSH/BRCC	Rd < Rr	C = 1	BRLO/BRCS	Unsigned
Rd = Rr	Z = 1	BREQ	Rd ≠ Rr	Z = 0	BRNE	Unsigned
Rd ≤ Rr	C + Z = 1	BRSH(1)	Rd > Rr	C + Z = 0	BRLO*	Unsigned
Rd < Rr	C = 1	BRLO/BRCS	Rd ≥ Rr	C = 0	BRSH/BRCC	Unsigned
Carry	C = 1	BRCS	No carry	C = 0	BRCC	Simple
Negative	N = 1	BRMI	Positive	N = 0	BRPL	Simple
Overflow	V = 1	BRVS	No overflow	V = 0	BRVC	Simple
Zero	Z = 1	BREQ	Not zero	Z = 0	BRNE	Simple

Note: 1. Interchange Rd and Rr in the operation before the test, i.e., CP Rd,Rr → CP Rr,Rd.

27. BRNE – Branch if Not Equal

Operation:

(i) If Rd ≠ Rr (Z = 0) then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRNE k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	01kk	kkkk	k001

27.2 Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
            eor     r27,r27   ; Clear r27
loop:       inc     r27       ; Increase r27
            ...
            cpi     r27,5     ; Compare r27 to 5
            brne    loop      ; Branch if r27<>5
            nop               ; Loop exit (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false
2 if condition is true

atmel-0856-avr-instruction-set-manual.pdf (página 37)

Rotinas Simples de Atraso

- DEC consome 1 ciclo
- BRNE consome 2 ciclos e na última vez, quando não desvia mais, consome 1 ciclo
- Como os registradores R3 e R2 possuem o valor zero inicialmente (o primeiro decremento os leva ao valor 255) e o decremento de R3 é repetido dentro do laço de R2
  - Espera-se que haja 256 (255 + 1) decrementos de R3 vezes 256(255 + 1) decrementos de R2

Atraso:

DEC R3
BRNE Atraso
DEC R2
BRNE Atraso

$$\begin{bmatrix} 1 \text{ ciclo} \\ 2 \text{ ciclos} \end{bmatrix} \times 255 + \begin{bmatrix} 1 \text{ ciclo} \\ 1 \text{ ciclo} \end{bmatrix} = \begin{bmatrix} 767 \text{ ciclos} \\ + \ 1 \text{ ciclo} \\ + \ 2 \text{ ciclos} \end{bmatrix} \times 255 + \begin{bmatrix} 767 \text{ ciclos} \\ + \ 1 \text{ ciclo} \\ + \ 1 \text{ ciclo} \end{bmatrix} = 197119 \text{ ciclos}$$

Fig. 5.2 – Cálculo preciso da sub-rotina de atraso.

Rotinas Simples de Atraso

- Se forem considerados:
  - Os ciclos gastos para a chamada da sub-rotina a instrução RCALL (3 ciclos)
  - Os ciclos para o retorno, com a instrução RET (4 ciclos)
- Tem-se um gasto total da chamada da sub-rotina até seu retorno de 197.126 ciclos

Rotinas Simples de Atraso

- **O tempo gasto pelo microcontrolador dependerá da frequência de trabalho utilizada**
- **Como no AVR um ciclo de máquina equivale ao inverso da frequência do clock (período), o tempo gasto será dado por:**

Período = 1/Freq. de trabalho

Período = 1/16000000 Hz (16 MHz)

Período = 0,0000000625 segundos

Período = 62,5 nanossegundos

Rotinas Simples de Atraso

- **O tempo gasto pelo microcontrolador dependerá da frequência de trabalho utilizada**
- **Como no AVR um ciclo de máquina equivale ao inverso da frequência do clock (período), o tempo gasto será dado por:**

Tempo Gasto = Nº de ciclos x 1/Freq. de trabalho ou

Tempo Gasto = Nº de ciclos x Período

- **Para o exemplo anterior, com um clock de 16 Mhz (período de 62,5 ns), da chamada da sub-rotina até seu retorno, resulta em:**

Tempo Gasto = 197.126 ciclos x 62,5 ns = 12,32 ms

```
//------------------------------------------------------------------------ //
//              AVR e Arduino: Técnicas de Projeto, 2a ed. - 2012.         //
//------------------------------------------------------------------------ //
.include "m328pdef.inc"          //arquivo com as definições dos nomes dos bits
.equ LED   = PB5                 //LED é o substituto de PB5 na programação
.ORG 0x000                       //endereço de início de escrita do código
INICIO:
        LDI R16,0xFF             //carrega R16 com o valor 0xFF
        OUT DDRB,R16             //configura todos os pinos do PORTB como saída
PRINCIPAL:
     SBI PORTB, LED              //coloca o pino PB5 em 5V
         RCALL ATRASO            //chama a sub-rotina de atraso
         CBI PORTB, LED          //coloca o pino PB5 em 0V
         RCALL ATRASO            //chama a sub-rotina de atraso
         RJMP PRINCIPAL          //volta para PRINCIPAL
ATRASO:                          //atraso de aprox. 200ms
        LDI R19,16
 volta:
        DEC  R17                 //decrementa R17, começa com 0x00
        BRNE volta               //enquanto R17 > 0 fica decrementando R17
        DEC  R18                 //decrementa R18, começa com 0x00
        BRNE volta               //enquanto R18 > 0 volta decrementar R18
        DEC  R19                 //decrementa R19
        BRNE volta               //enquanto R19 > 0 vai para volta
        RET
//------------------------------------------------------------------------
```

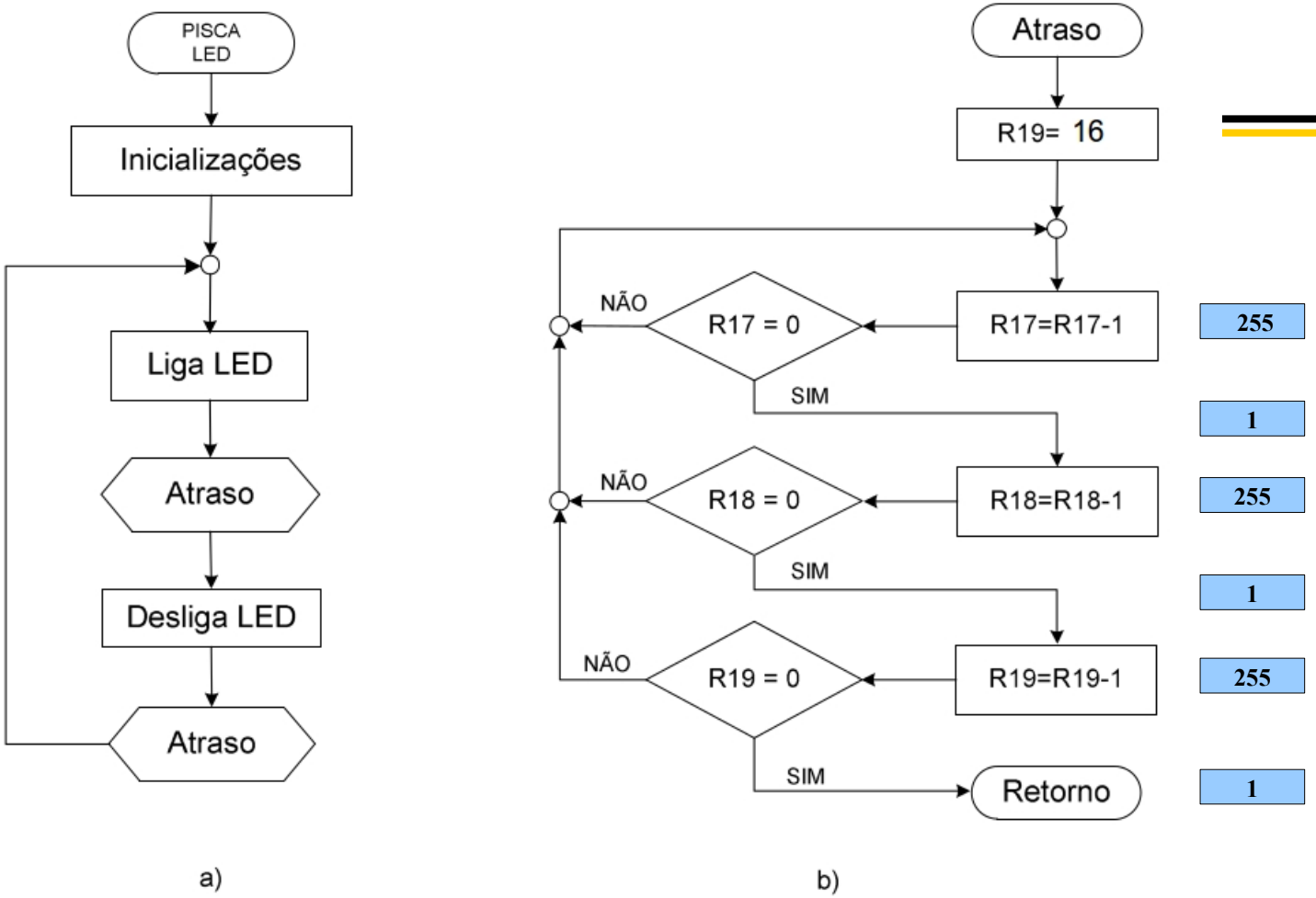


Fig. 4.4 – Fluxograma para o programa que pisca um LED: a) principal e b) sub-rotina.

ATRASO:

```
        LDI R19,16
volta:
        DEC  R17
        BRNE volta
        DEC  R18
        BRNE volta

        DEC  R19
        BRNE volta

        RET
```

$$\begin{bmatrix} 1 \text{ ciclo} \\ +2 \text{ ciclos} \end{bmatrix} \times 255 + \begin{bmatrix} 1 \text{ ciclo} \\ 1 \text{ ciclo} \end{bmatrix} = \begin{bmatrix} 767 \text{ ciclos} \\ +1 \text{ ciclo} \\ +2 \text{ ciclos} \end{bmatrix} \times 255 + \begin{bmatrix} 767 \text{ ciclos} \\ +1 \text{ ciclo} \\ +1 \text{ ciclo} \end{bmatrix}$$

= 197119 ciclos + 1 ciclo + 2 ciclos → **x 15 +** → **= 197119 ciclos + 1 ciclo + 1 ciclos**

= 197122 x 15 = 2956830 + 197121 = 3153951

= 3153951 + 7 = 3153958

= 3153958 x 0,0000000625 = 0,197121938 s = 197 ms

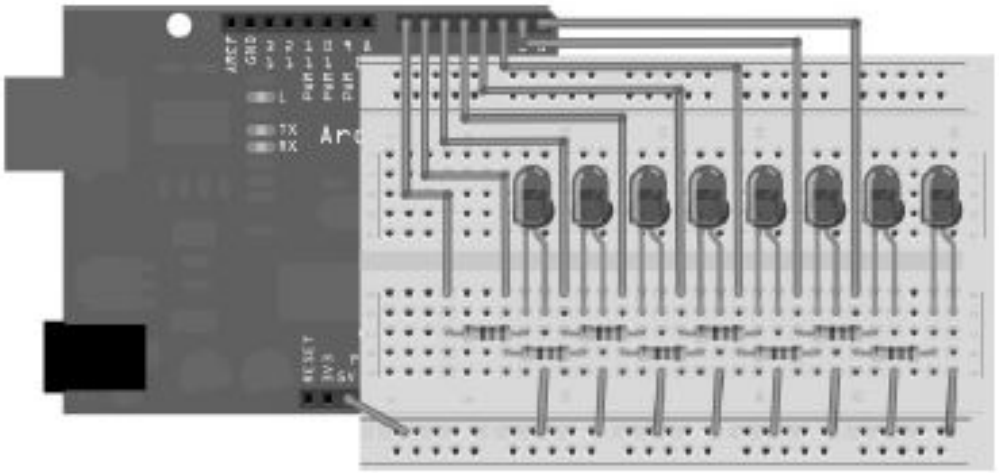
```
#include <avr/io.h>          //definições do componente especificado
#include <util/delay.h>      //biblioteca para o uso das rotinas de _delay_ms() e _delay_us()

//Definições de macros - empregadas para o trabalho com os bits de uma variável ou registrador

#define set_bit(Y,bit_x) (Y|=(1<<bit_x))    //ativa o bit x da variável Y (coloca em 1)
#define clr_bit(Y,bit_x) (Y&=~(1<<bit_x))   //limpa o bit x da variável Y (coloca em 0)
#define tst_bit(Y,bit_x) (Y&(1<<bit_x))     //testa o bit x da variável Y (retorna 0 ou 1)
#define cpl_bit(Y,bit_x) (Y^=(1<<bit_x))    //troca o estado do bit x da variável Y (complementa)

#define LED PB5              //LED é o substituto de PB5 na programação

//-------------------------------------------------------------------------------
int main( )
{
    DDRB = 0xFF;             //configura todos os pinos do PORTB como saídas

    while(1)                 //laço infinito
    {
        set_bit(PORTB,LED); //liga LED
        _delay_ms(200);      //atraso de 200 ms
        clr_bit(PORTB,LED); //desliga LED
        _delay_ms(200);      //atraso de 200 ms
    }
}
```

Exercícios

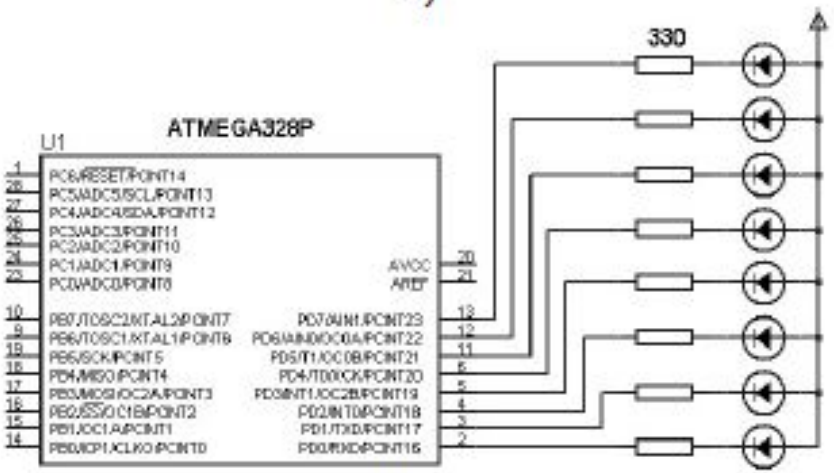
- **5.8 – Utilizando o deslocamento de bits crie um programa em C que ligue 8 LEDs (ver fig. 5.4a), da seguinte forma:**
  - **a) Ligue sequencialmente 1 LED da direita para a esquerda (o LED deve permanecer ligado até que todos os 8 estejam ligados, depois eles devem ser desligados e o processo repetido).**
  - **b) Ligue sequencialmente 1 LED da esquerda para a direita, mesma lógica da letra a).**
  - **c) Ligue sequencialmente 1 LED da direita para a esquerda, desta vez somente um LED deve ser ligado por vez.**
  - **d) Ligue sequencialmente 1 LED da esquerda par a direita e vice-versa (vai e volta), só um LED deve ser ligado por vez.**
  - **e) Ligue todos os LEDs e apague somente um LED de cada vez, da direita para a esquerda e vice-versa (vai e volta), somente um LED deve ser apagado por vez.**

Exercícios

- **5.8 – Utilizando o deslocamento de bits crie um programa em C que ligue 8 LEDs (ver fig. 5.4a), da seguinte forma:**
  - **f) Mostre uma contagem binária crescente (0-255) com passo de 250 ms.**
  - **g) Mostre uma contagem binária decrescente (255-0) com passo de 250 ms.**



a)



b)

Fig. 5.4 – Sequencial com 8 LEDs: a) montagem para o Arduino e b) esquemático.

Portas de Entrada e Saída ATmega328

- **Para o uso de um pino de I/O, deve-se:**
  - **Definir se ele será entrada ou saída escrevendo-se no registrador DDRx**
  - **Se for definido como saída (1):**
    - A escrita no registrador PORTx alterará o estado lógico do pino
  - **Se for definido como entrada (0):**
    - A escrita no registrador PORTx poderá habilitar o pull-up interno

```
//=============================================================================================  //
//      LIGANDO E DESLIGANDO UM LED QUANDO UM BOTÃO É PRESSIONADO                               //
//=============================================================================================  //

//DEFINIÇÕES
.include "m328pdef.inc" //arquivo com as definições dos nomes
.equ LED   = PD2  //LED é o substituto de PD2 na programação
.equ BOTAO = PD7  //BOTAO é o substituto de PD7 na programação
.def AUX   = R16  /*R16 tem agora o nome de AUX (nem todos os 32 registradores
              de uso geral podem ser empregados em todas as instruções) */
//---------------------------------------------------------------------------------------------
.ORG 0x000     //endereço de início de escrita do código na memória flash
               //após o reset o contador do programa aponta para cá
Inicializacoes:

   LDI  AUX,0b00000100  //carrega AUX com o valor 0x04 (1 = saída e 0 = entrada)
   OUT  DDRD,AUX        //configura PORTD, PD2 saída e demais pinos entradas
   LDI  AUX,0b11111111  //habilita o pull-up para o botão e apaga o LED (pull-up em todas as entradas)
   OUT  PORTD,AUX
```

Portas de Entrada e Saída ATmega328

- **Quando o bit PUD (Pull-Up Disable) no registrador MCUCR (MCU Control Register) está em 1 lógico**
  - Os pull-ups em todos os PORTs são desabilitados, mesmo que os bits DDXn e PORTXn estejam configurados para habilitá-los.

Tab. 5.1 - Bits de controle dos pinos dos PORTs.

DDXn*	PORTXn	PUD (no MCUCR)	I/O	Pull-up	Comentário
0	0	x	Entrada	Não	Alta impedância (Hi-Z).
0	1	0	Entrada	Sim	PXn irá fornecer corrente se externamente for colocado em nível lógico 0.
0	1	1	Entrada	Não	Alta impedância (Hi-Z).
1	0	x	Saída	Não	Saída em zero (drena corrente).
1	1	x	Saída	Não	Saída em nível alto (fornece corrente).

*X = B, C ou D; n = 0, 1, ... ou 7.

14.4 Register Description

14.4.1 MCUCR – MCU Control Register

Bit	7	6	5	4	3	2	1	0	
0x35 (0x55)	–	BODS[1]	BODSE[1]	PUD	–	–	IVSEL	IVCE	MCUCR
Read/Write	R	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Notes: 1. BODS and BODSE only available for picoPower devices ATmega48PA/88PA/168PA/328P

- **Bit 4 – PUD: Pull-up Disable**

When this bit is written to one, the pull-ups in the I/O ports are disabled even if the DDxn and PORTxn Registers are configured to enable the pull-ups ({DDxn, PORTxn} = 0b01). See "Configuring the Pin" on page 77 for more details about this feature.

14.4.2 PORTB – The Port B Data Register

Bit	7	6	5	4	3	2	1	0	
0x05 (0x25)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

14.4.3 DDRB – The Port B Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

14.4.4 PINB – The Port B Input Pins Address[1]

Bit	7	6	5	4	3	2	1	0	
0x03 (0x23)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

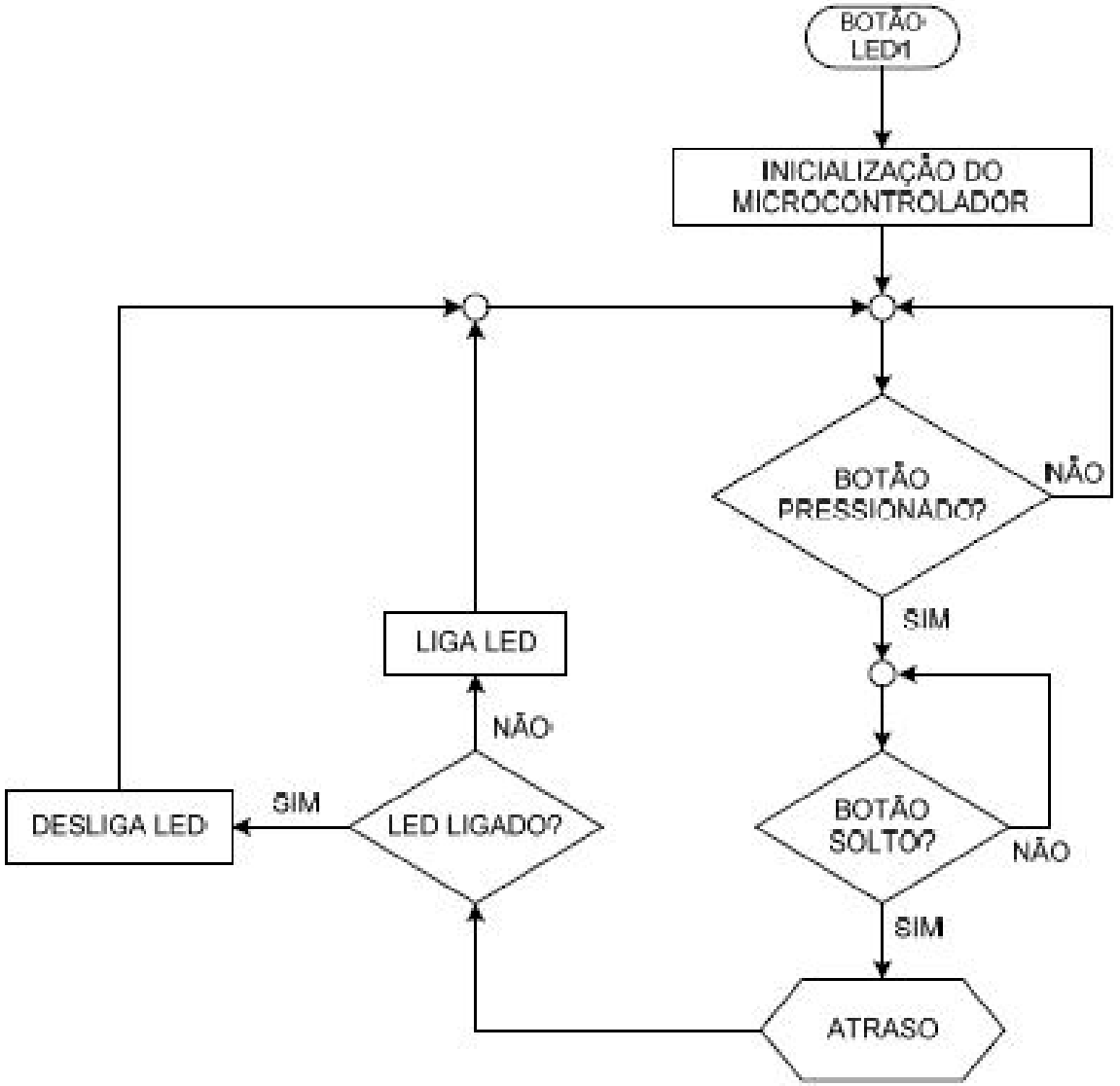


Fig. 5.7 – Fluxograma do programa para ligar e apagar um LED com um botão.

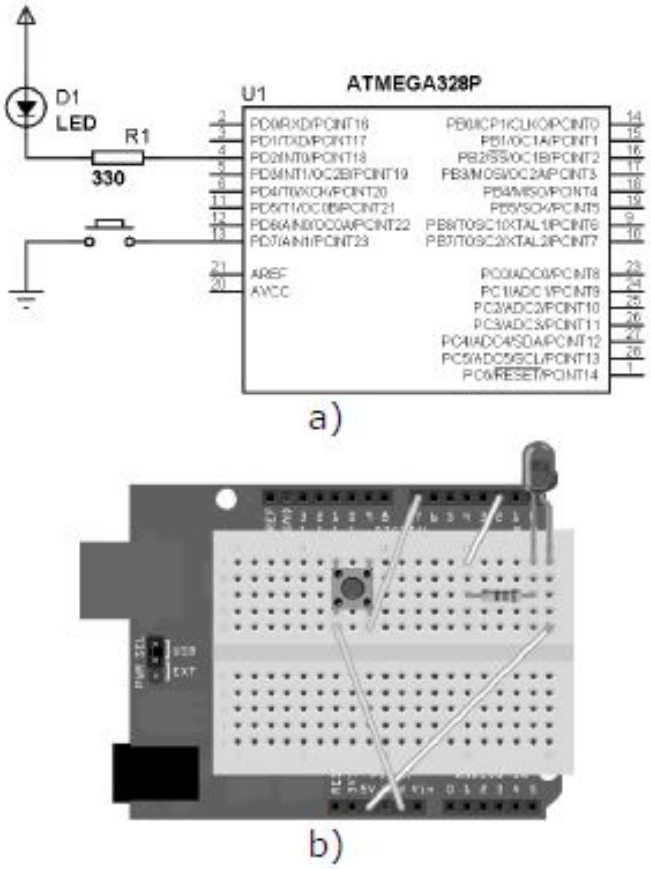


Fig. 5.8 – Circuito para ligar e apagar um LED com um botão: a) esquemático e b) montagem para o Arduino.

C- Trabalho com Bits

- **Código Botao_LED.c**

```
//------------------------------------------------------------------------------ //
//    AVR e Arduino: Técnicas de Projeto, 2a ed. - 2012.                        //
//------------------------------------------------------------------------------ //
//==============================================================================
//    LIGANDO E DESLIGANDO UM LED QUANDO UM BOTÃO É PRESSIONADO                  //
//============================================================================== //
#define F_CPU 16000000UL   /*define a frequência do microcontrolador 16MHz (necessário
                     para usar as rotinas de atraso)*/
#include <avr/io.h>      //definições do componente especificado
#include <util/delay.h>    //biblioteca para as rotinas de _delay_ms() e delay_us()

//Definições de macros - para o trabalho com os bits de uma variável

#define set_bit(Y,bit_x)(Y|=(1<<bit_x))      //ativa o bit x da variável Y (coloca em 1)
#define clr_bit(Y,bit_x)(Y&=~(1<<bit_x))  //limpa o bit x da variável Y (coloca em 0)
#define cpl_bit(Y,bit_x)(Y^=(1<<bit_x))      //troca o estado do bit x da variável Y
#define tst_bit(Y,bit_x)(Y&(1<<bit_x))    //testa o bit x da variável Y (retorna 0 ou 1)

#define LED   PD2   //LED é o substituto de PD2 na programação
#define BOTAO PD7   //BOTAO é o substituto de PD7 na programação
//------------------------------------------------------------------------------
```

C- Trabalho com Bits

```
//-------------------------------------------------------------------------
int main()
{

   DDRD = 0b00000100;   //configura o PORTD, PD2 saída, os demais pinos entradas
   PORTD= 0b11111111;   /*habilita o pull-up para o botão e apaga o LED (todas as
                  entradas com pull-ups habilitados)*/

   while(1)                        //laço infinito
   {
      if(!tst_bit(PIND,BOTAO))         //se o botão for pressionado executa o if
      {
         while(!tst_bit(PIND,BOTAO));  //fica preso até soltar o botão

         //_delay_ms(10);              //atraso de 10 ms para eliminar o ruído do botão

         if(tst_bit(PORTD,LED))        //se o LED estiver apagado, liga o LED
            clr_bit(PORTD,LED);
         else                          //se não apaga o LED
            set_bit(PORTD,LED);

         //o comando cpl_bit(PORTD,LED) pode substituir este laço if-else

      }//if do botão pressionado

   }//laço infinito
}
//=========================================================================
```

Display 7 Segmentos

- **Displays são compostos por LEDs arranjados adequadamente em encapsulamento**
  - **Produzindo os dígitos numéricos**

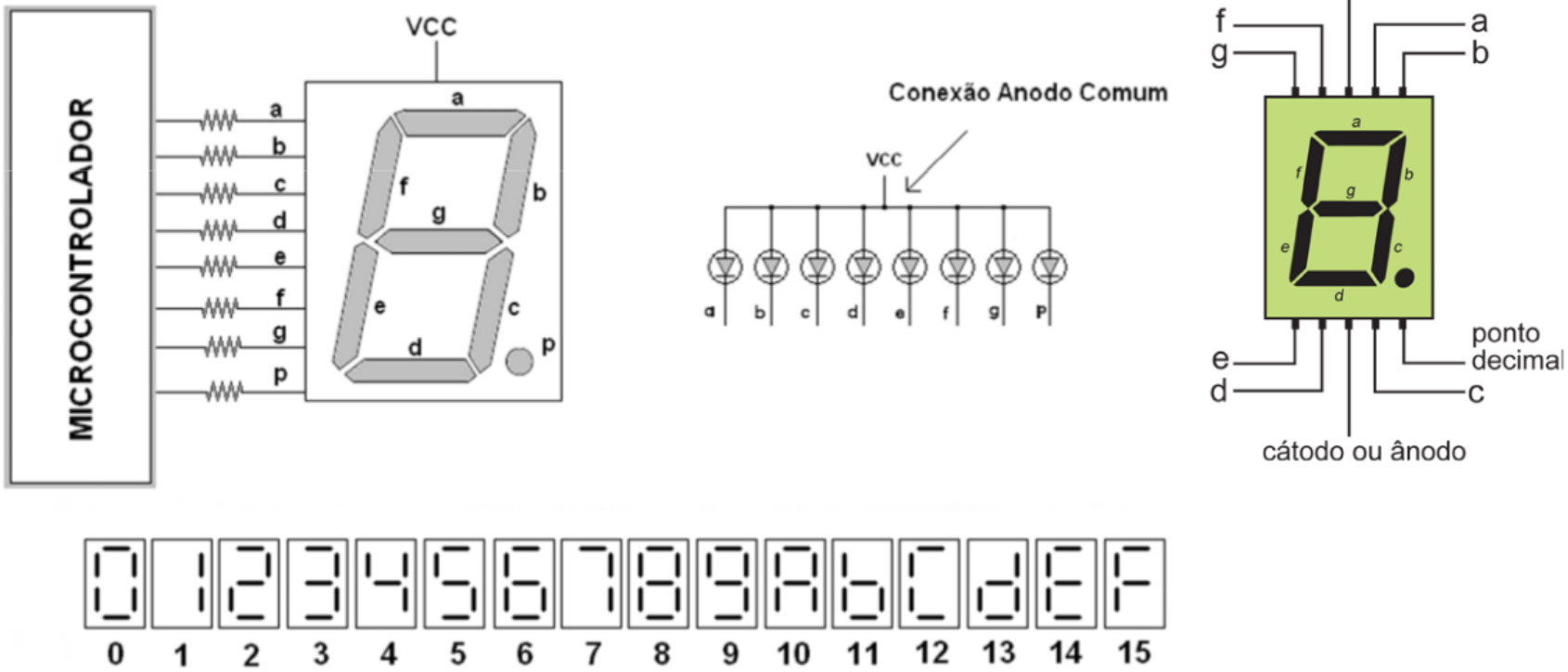


Fig. 5.12 – *Display* de 7 segmentos anodo comum.

Display 7 Segmentos

Tab. 5.2 - Valores para a decodificação de *displays* de 7 segmentos.

Dígito	Anodo comum		Catodo comum	
	gfedcba		gfedcba	
0	0b1000000	0x40	0b0111111	0x3F
1	0b1111001	0x79	0b0000110	0x06
2	0b0100100	0x24	0b1011011	0x5B
3	0b0110000	0x30	0b1001111	0x4F
4	0b0011001	0x19	0b1100110	0x66
5	0b0010010	0x12	0b1101101	0x6D
6	0b0000010	0x02	0b1111101	0x7D
7	0b1111000	0x78	0b0000111	0x07
8	0b0000000	0x00	0b1111111	0x7F
9	0b0011000	0x18	0b1100111	0x67
A	0b0001000	0x08	0b1110111	0x77
B	0b0000011	0x03	0b1111100	0x7C
C	0b1000110	0x46	0b0111001	0x39
D	0b0100001	0x21	0b1011110	0x5E
E	0b0000110	0x06	0b1111001	0x79
F	0b0001110	0x0E	0b1110001	0x71

Display 7 Segmentos

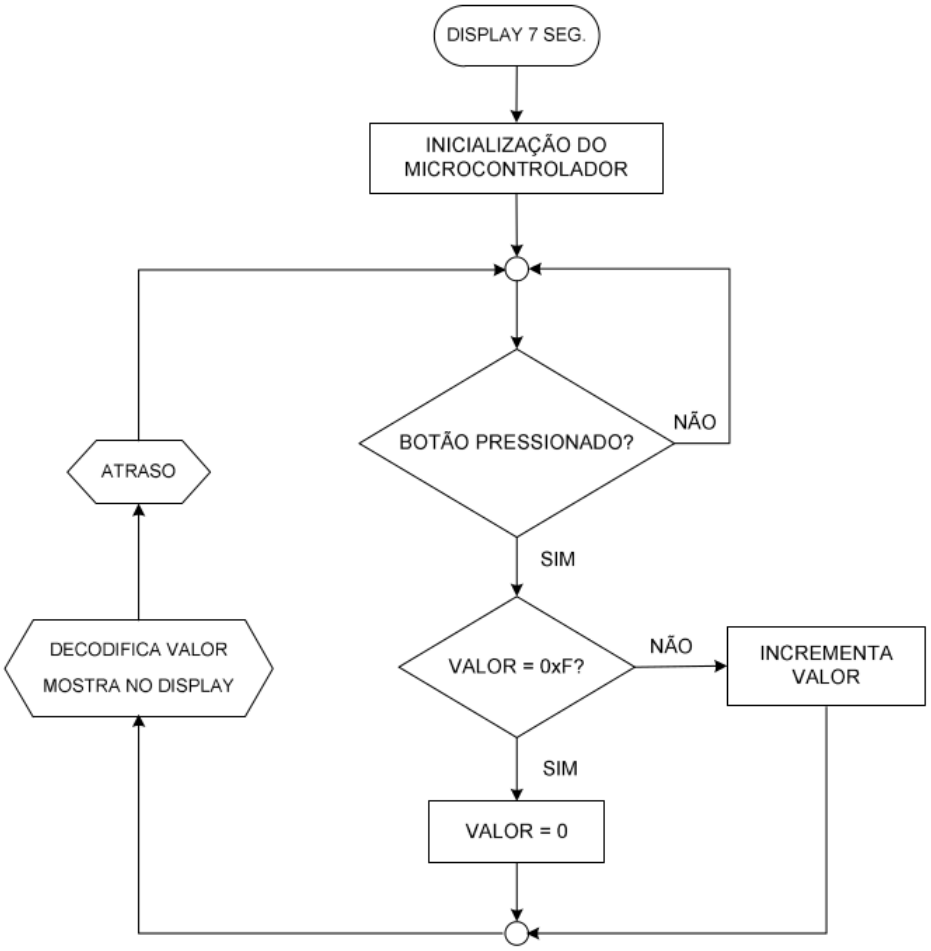


Fig. 5.13 – Fluxograma para apresentar um número hexadecimal de 0 até F quando um botão é pressionado.

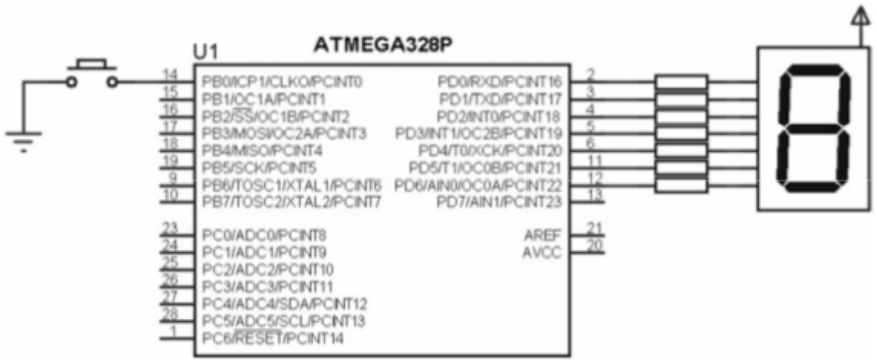
Display 7 Segmentos

```
//=============================================================================  //
//          ESCREVENDO EM UM DISPLAY DE 7 SEGMENTOS ANODO COMUM                  //
//=============================================================================  //
#define F_CPU 16000000UL   //define a frequência do microcontrolador em 16MHz

#include <avr/io.h>      //definições do componente especificado
#include <util/delay.h>    //biblioteca para o uso das rotinas de _delay_
#include <avr/pgmspace.h>  //biblioteca para poder gravar dados na memória flash

//Definições de macros - para o trabalho com os bits de uma variável
#define tst_bit(Y,bit_x) (Y&(1<<bit_x)) //testa o bit x da variável Y (retorna 0 ou 1)

#define DISPLAY    PORTD      //define um nome auxiliar para o display
#define BOTAO   PB0     //define PB0 com o nome de BOTAO

//variável gravada na memória flash
const unsigned char Tabela[] PROGMEM = {0x40, 0x79, 0x24, 0x30, 0x19, 0x12, 0x02, 0x78,
 0x00, 0x18, 0x08, 0x03, 0x46, 0x21, 0x06, 0x0E};
//-----------------------------------------------------------------------------
```

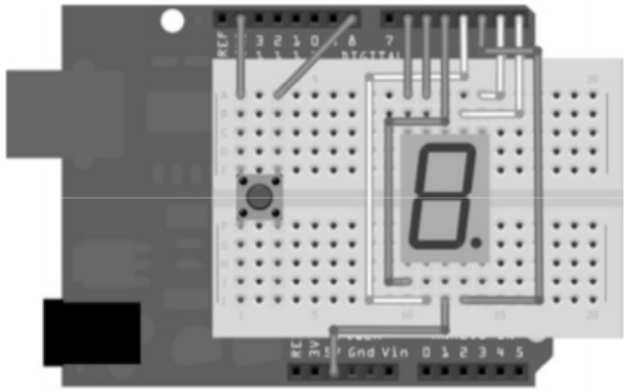
Display 7 Segmentos

```
//-------------------------------------------------------------------------

int main()
{
   unsigned char valor = 0;    //declara variável local

   DDRB = 0b11111110;          //PB0 como pino de entrada, os demais pinos como saída
   PORTB= 0x01;              //habilita o pull-up do PB0
   DDRD = 0xFF;              //PORTD como saída (display)
   PORTD= 0xFF;              //desliga o display
   UCSR0B = 0x00;            //PD0 e PD1 como I/O genérico, para uso no Arduino

   while(1)                  //laço infinito
   {
      if(!tst_bit(PINB,BOTAO))//se o botão for pressionado executa
      {
         if(valor==0x0F)      //se o valor for igual a 0xF, zera o valor,
            valor=0;
         else           //se não o incrementa
            valor++;

         //decodifica o valor e mostra no display, busca o valor na Tabela.
         DISPLAY = pgm_read_byte(&Tabela[valor]);

         _delay_ms(200);      //atraso para incremento automático do nr. no display

      }//if botão

   }//laço infinito
}
//=========================================================================
```

Display 7 Segmentos



a)

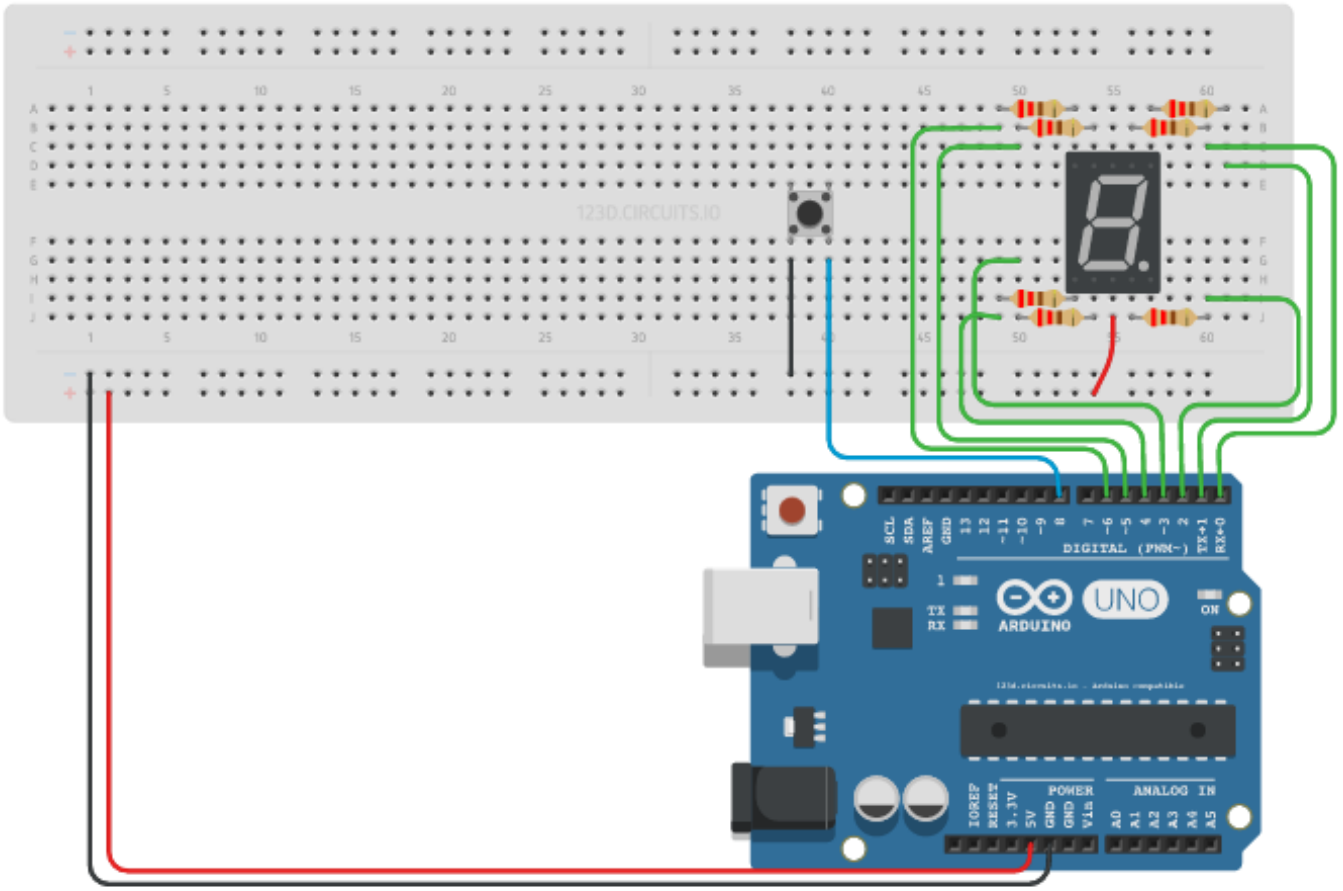


b)

Fig. 5.14 – Circuito para acionamento de um *display* de 7 segmentos anodo comum: a) esquemático e b) montagem no Arduino.

Display 7 Segmentos

https://tinkercad.com/things/33MlvKvihhK



Display 7 Segmentos

Exercícios:

5.13 – Elaborar um programa para apresentar em um *display* de 7 segmentos um número aleatório[25] entre 1 e 6 quando um botão for pressionado, ou seja, crie um dado eletrônico. Empregue o mesmo circuito da fig. 5.14.

5.14 – Elaborar um programa para apresentar nos LEDs da fig. 5.15 um número aleatório entre 1 e 6, formando os números de um dado (mesma lógica do exercício acima).

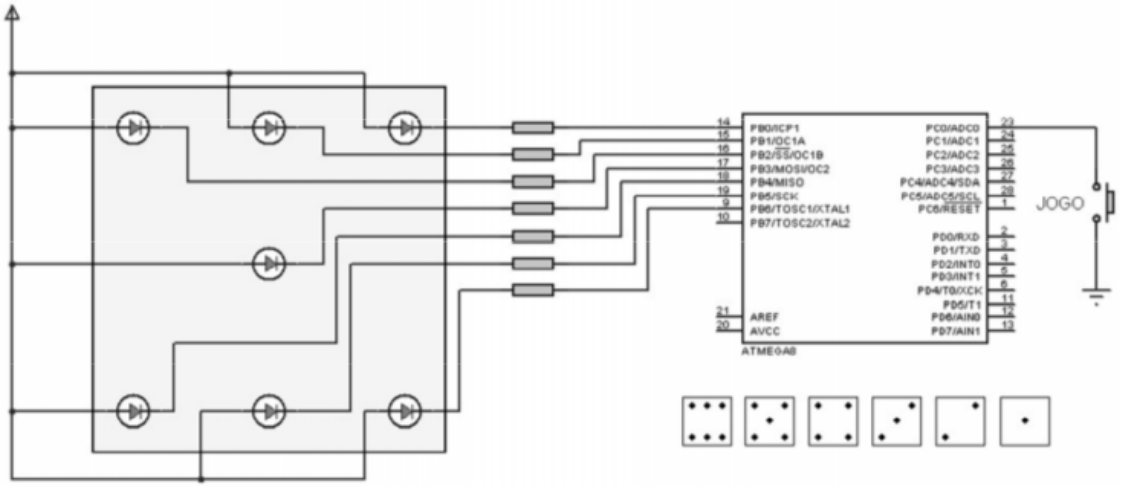


Fig. 5.15 – Dado eletrônico com LEDs.

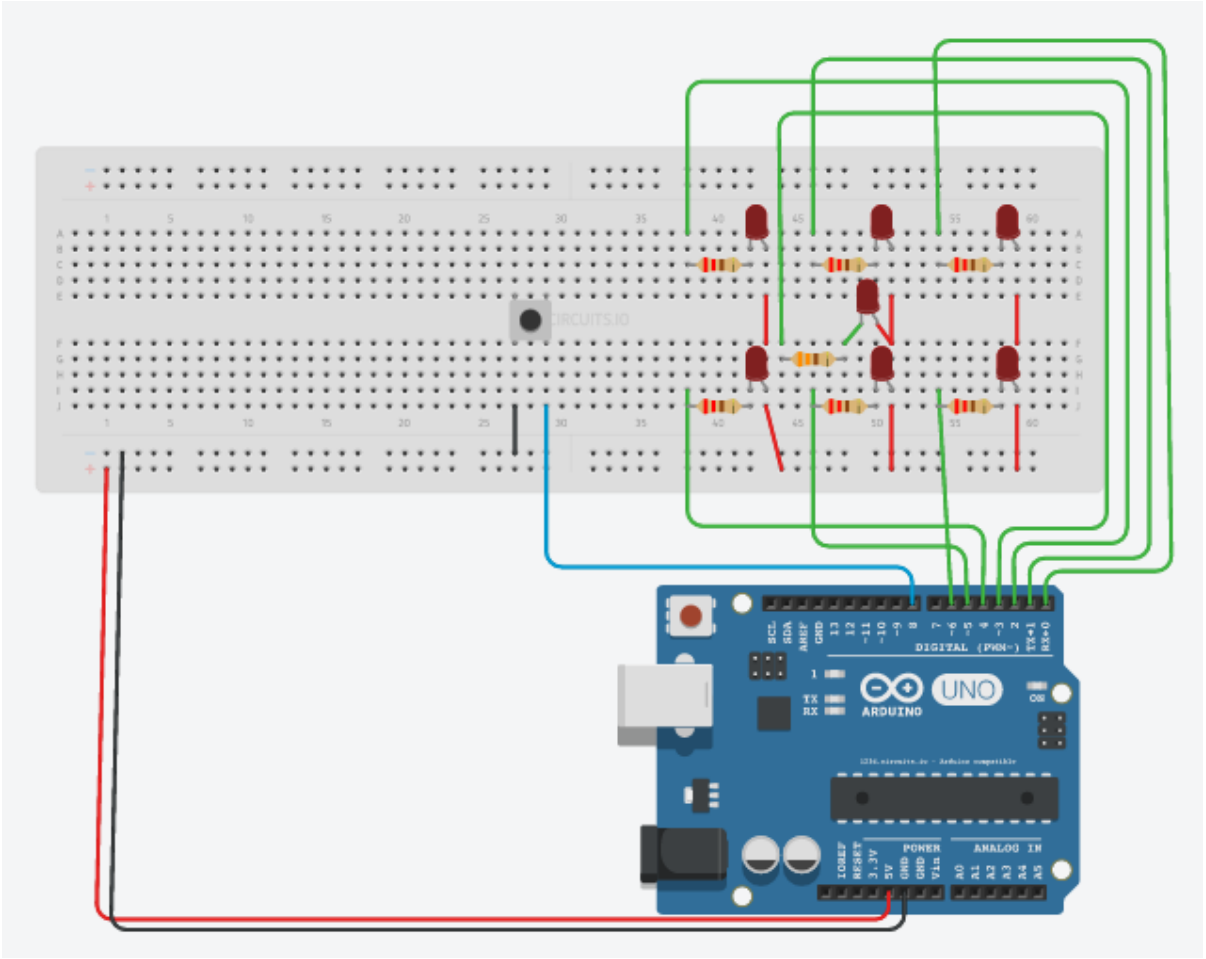
Obs.: a pinagem do ATmega8 é igual a do ATmega328.

Display 7 Segmentos

- Na verdade, criar um número puramente aleatório é difícil, o mais fácil é um pseudoaleatório
- Neste exercício, o objetivo é não empregar as bibliotecas padrão do C. A ideia é utilizar o botão para gerar o evento de sorteio do número
- Dessa forma, um contador pode ficar contando continuamente de 1 até 6 e, quando o botão for pressionado, um número da contagem será selecionado

Display 7 Segmentos

https://tinkercad.com/things/dqU6zk8AhtK



Referências

- **AVR e Arduino – Técnicas de Projeto.**
  - Capítulo 5. Portas de Entrada e Saída (I/Os)