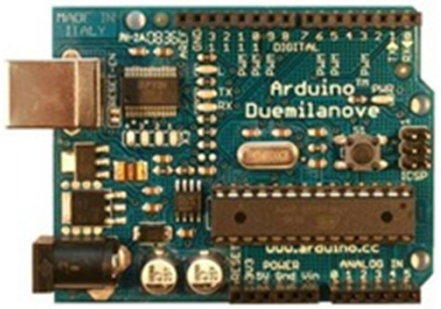
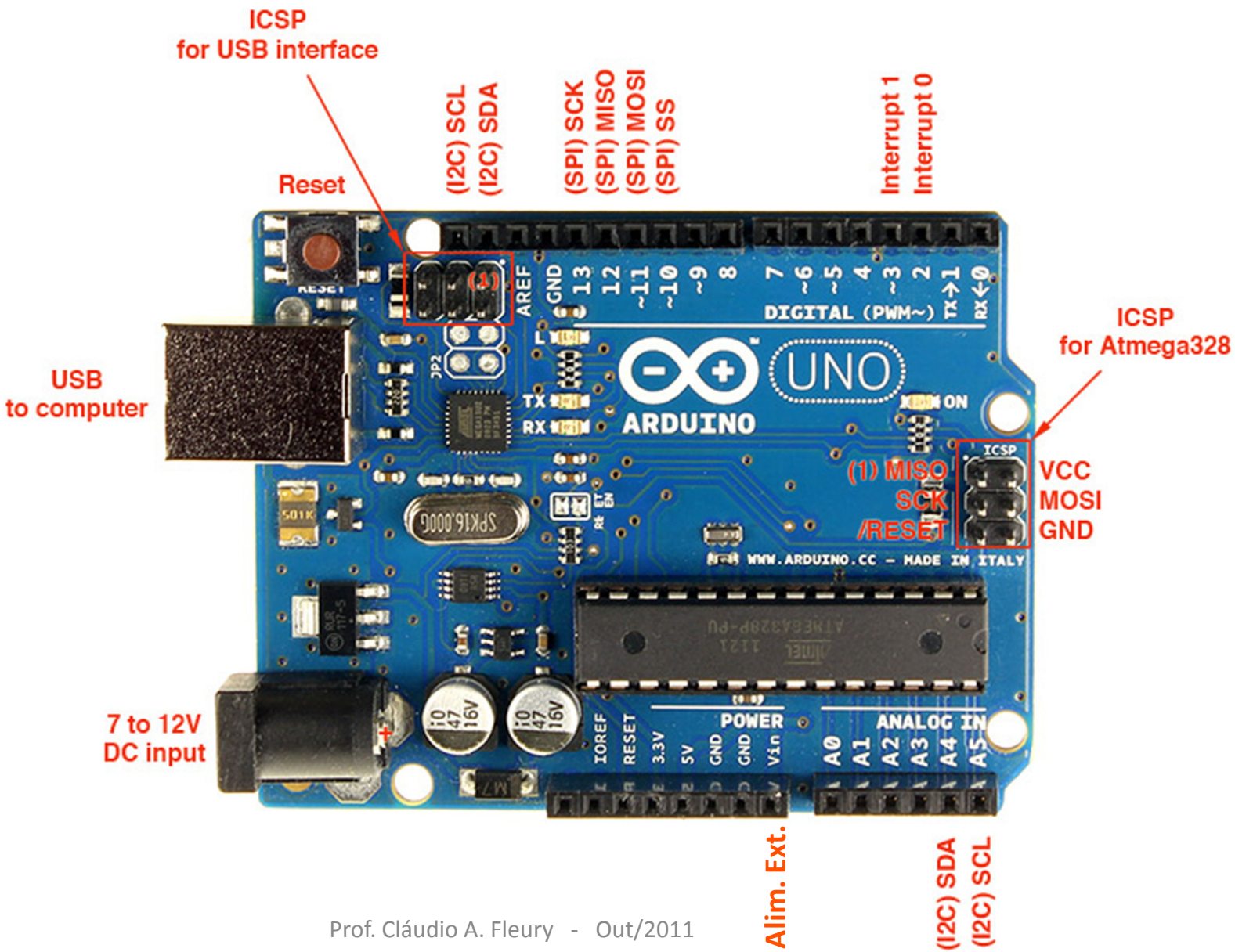


Interrupções de HW no Arduino

Prof. Cláudio A. Fleury – Out/2011

Fonte - By Dave Auld, 23 Feb 2010 www.codeproject.com/KB/system/Arduino_interrupts.aspx

Arduino UNO


ICSP
for USB interface
Reset
(I2C) SCL
(I2C) SDA
(SPI) SCK
(SPI) MISO
(SPI) MOSI
(SPI) SS
Interrupt 1
Interrupt 0
USB
to computer
ICSP
for Atmega328
VCC
MOSI
GND
(1) MISO
SCK
/RESET
7 to 12V
DC input
Alim. Ext.
(I2C) SDA
(I2C) SCL
UNO
ARDUINO
DIGITAL (PWM~)
POWER
ANALOG IN
WWW.ARDUINO.CC – MADE IN ITALY

Interrupção de HW

- **Definição**
  - Mecanismo de controle do fluxo de execução do programa
  - É um sinal (int./ext.) que interrompe a atividade corrente do microprocessador/microcontrolador (uP/uC)
    - Tipos de sinais
      - Externo: entrada digital (sinal gerado por HW)
      - Interno : temporizador (sinal gerado por SW)
  - Método de sinalizar ao uP/uC de que um determinado **evento assíncrono** ocorreu e precisa ser atendido
  - Ação a ser executada com base no estado de um pino digital
    - Programas maiores/complexos nem sempre conseguem checar continuamente o estado de um pino digital

Interrupção de HW

- Exemplos de Eventos Assíncronos
  - Pressionamento de uma tecla
  - Leitura de dados em uma porta serial (UART)
  - Armazenamento de uma amostra obtida por um conversor A/D
  - Finalização de uma operação de fechamento de embalagem
  - Contagem de peças em uma esteira
  - Alarme de pressão interna de uma caldeira
  - *Encoder* giratório (medição de ângulo ou veloc. angular)

Todos esses eventos precisam da atenção imediata do *uC*...

Interrupção de HW

- Mecanismos de Controle do Fluxo de Execução

1. **Verificação Frequente** (**Revezamento**[1])

- Leitura e comparação frequentes de uma entrada digital

(comando de leitura em um *loop*) → **CPU OCUPADA** !!!

- Algumas mudanças de estado da entrada (eventos) podem ser perdidas (não lidas em tempo) dependendo da complexidade da rotina em que se insere a **verificação frequente**

[1] Em inglês: *round-robin*

Interrupção de HW

- Mecanismos de Controle do Fluxo de Execução

2. **Alerta de ocorrência de evento** ou **Interrupção**

- A execução do programa principal (sub-rotina *loop()*) é interrompida e desviada para uma **rotina de serviço**, para atendimento ao evento, voltando-se ao fluxo normal, após a execução da **ISR**[1]

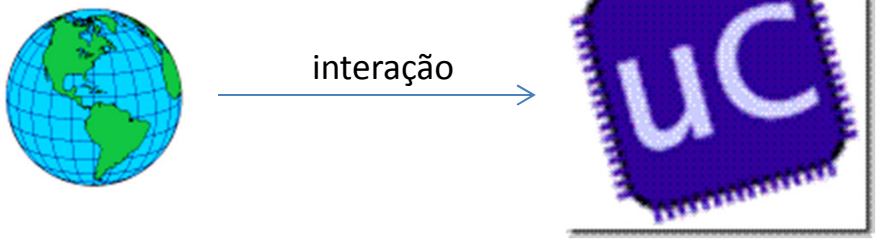
- Eventos não são perdidos/ignorados, como podem ocorrer com **verificações frequentes**

Interrupção
=
sub-rotina disparada por um evento assíncrono (externo ou interno)

[1] Em inglês: *Interrupt Service Routine*

Vantagens das Interrupções

- Resolve problemas de temporização que podem acontecer nos projetos - em boa parte dos projetos, os problemas de temporização só são detectados depois que estão em produção
  - Sintomas dos projetos problemáticos
    - Porque não está funcionando mais?
    - Funcionou até ontem!!! Eu não mudei nada!!!
    - Esse hardware/software/projeto está maluco!
- Diminui a complexidade da lógica do controle de fluxo



Mecanismo de Interrupção

- Sequência de funcionamento

I. Disparo da interrupção (sinal externo ou interno)

II. Programa principal é suspenso (pausado)

III. Execução é desviada para uma sub-rotina específica
   - A sub-rotina é conhecida como Manipuladora de Interrupção (*Interrupt Handler*) ou **Rotina de Serviço da Interrupção** (ISR)

IV. Ao terminar a execução da ISR o programa principal é retomado a partir do ponto em que foi suspenso

HW - ATmega168/328p

Interrupções:

INT0, **INT1**, **PCINT0**, **PCINT1**, **PCINT2**

externas

Vetores de Interrupções – endereços de memória da localização das ISRs.
Existe um vetor (ender.) para cada interrupção

Pinos Arduino	PORTA
Digital 0-7	D
Digital 8-13	B
Analog. 0-5	C

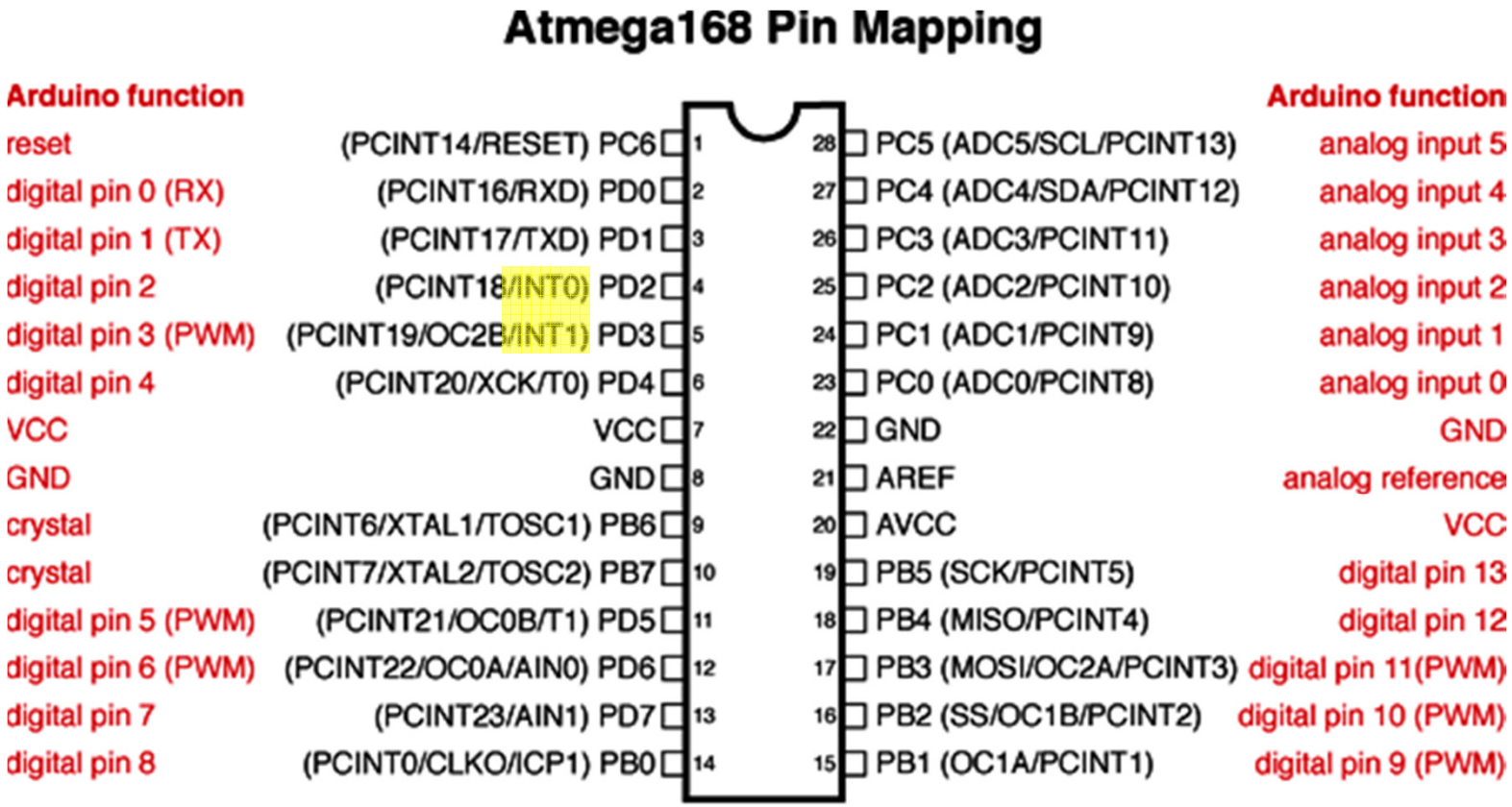
Tipos de Interrupções:
- **Externa:**
  INT0 e **INT1**
- **Mudança de Estado do Pino:**
  PCINT0, PCINT1 e **PCINT2**

Prioridades: 1 (maior prior.) → 6 (menor prior.)

Vetor No.	Endereço Programa	Fonte	Definição da Interrupção
1	0x000	RESET	Pino Externo, *Power-on Reset*, *Brown-out Reset* e *Watchdog*
2	0x001	INT0	Pedido Externo de Interrupção 0
3	0x002	INT1	Pedido Externo de Interrupção 1
4	0x003	PCINT0	Pedido de Inter. por Mudança Est. de Pino 0 (PORT B)
5	0x004	PCINT1	Pedido de Inter. por Mudança Est. de Pino 1 (PORT C)
6	0x005	PCINT2	Pedido de Inter. por Mudança Est. de Pino 2 (PORT D)

Fonte:
code.google.com/p/arduino-pinchangeint

HW - ATmega168/328p



Fonte:
code.google.com/p/arduino-pinchangeint

Uso das Interrupções

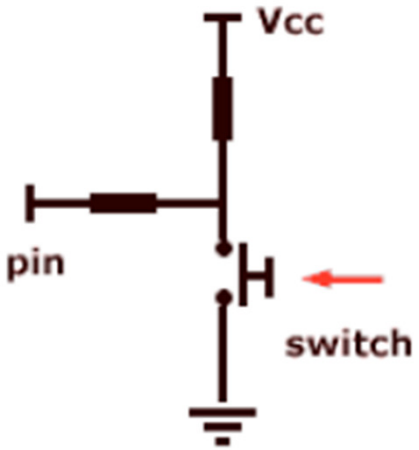
- Programa controlado por interrupção
  - Laço principal aguarda ocorrência das interrupções (não precisa "vigiar" mudanças)

- Preparação para uso
  - Escreva a ISR (sub-rotina de serviço para atendimento ao evento causador da interrupção)
  - Estabeleça o vetor de interrupção (endereço da ISR)
  - Habilite as interrupções desejadas (ajuste de máscara)
  - Habilite as interrupções globalmente

Interrupções no Duemilanove

- Duas interrupções **externas** disponíveis
  - **INT0** e **INT1** (pinos digitais 2 e 3, respectivamente)

- Interrupção por **mudança de estado do pino** (borda: RISING, FALLING ou nível: LOW)
  - Qualquer um dos 20 pinos do Arduino
  - Só existem 3 vetores disponíveis (ISR's): PCINT0, PCINT1, PCINT2

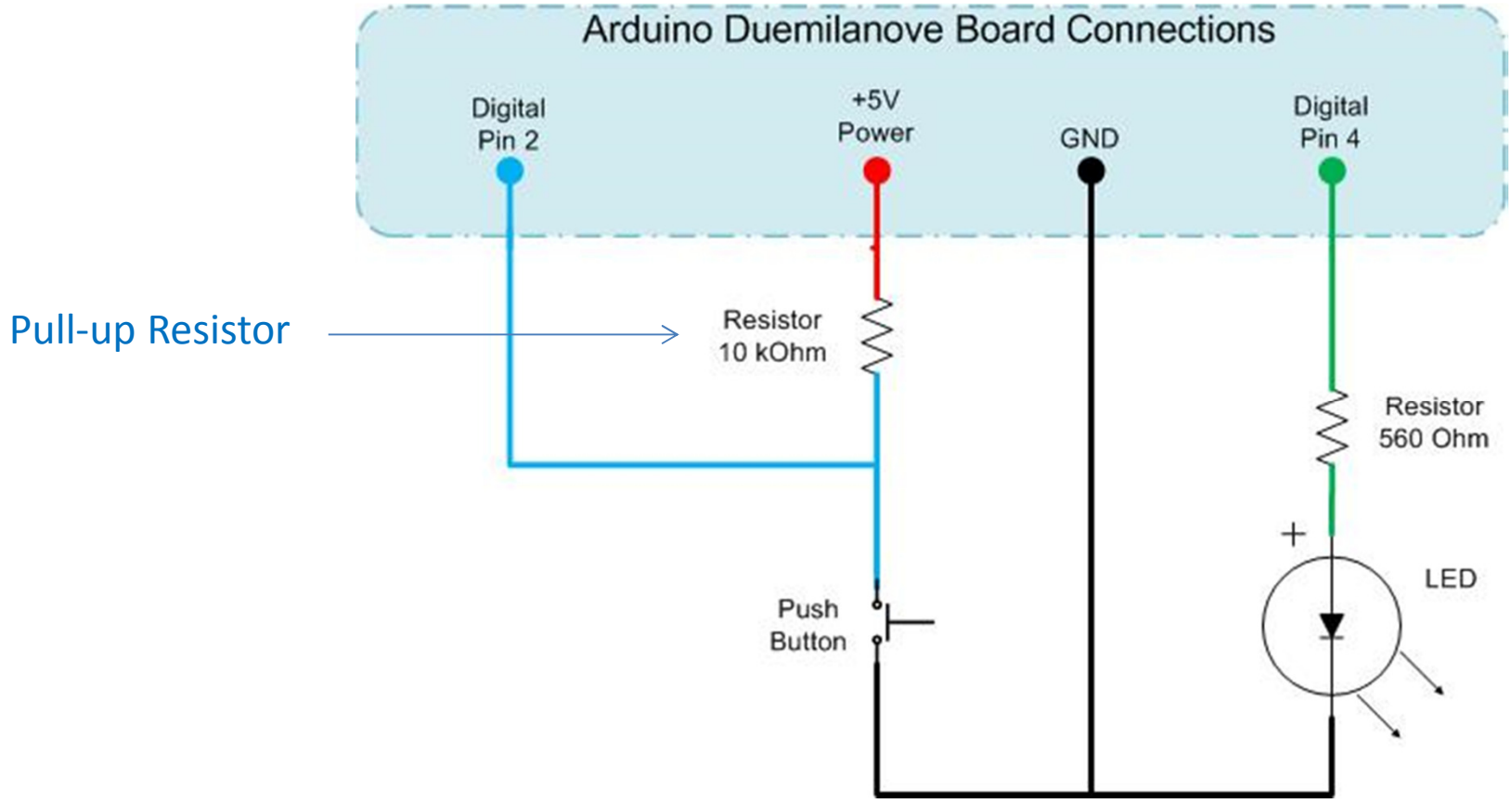


Biblioteca **EnableInterrupt** facilita o trabalho!
Podemos usar essa biblioteca para criar Interrupções em qualquer pino do Arduino...

https://github.com/GreyGnome/EnableInterrupt

Interrupções no Duemilanove

- **Exemplo** - Acendimento de um LED conectado ao pino 4, usando a interrupção externa no pino 2 (INT0)



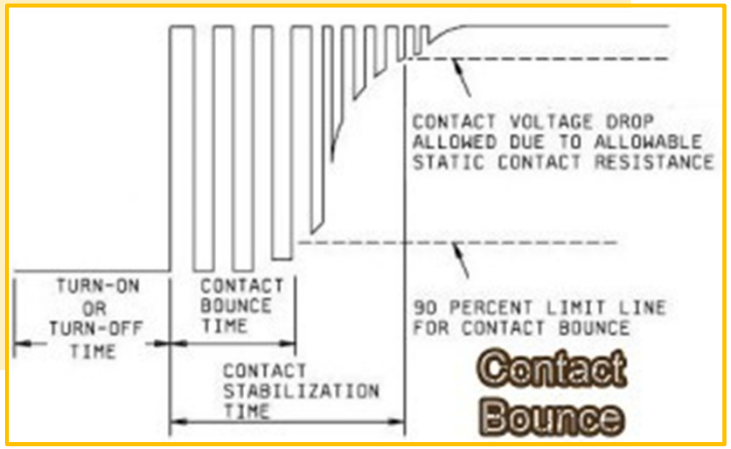
Sketch sem uso de Interrupção

```
int tecla = 2;           // entrada digital no pino 2
int led = 4;             // saída digital no pino 4
int estado = LOW;        // estado da entrada

void setup() {
  // configura Pino.2 como entrada e Pino.4 como saída
  pinMode(tecla,INPUT);
  pinMode(led,OUTPUT);
}

void loop() {
  estado = digitalRead(tecla);  // lê o estado da tecla
  digitalWrite(led,estado);     // escreve o estado no led
  // Simulando um longo processo ou tarefa complexa em execução
  for (int i = 0; i < 100; i++) {
    // não faz nada além de gastar tempo (CPU ocupada)
    delay(10);
  }
}
```

Sketch **com** uso de Interrupção

```
#include <avr/interrupt.h>
#define  TECLA        0                              // interrupção INT0 (pino 2)
#define  LED          4                              // LED no pino 4
volatile byte estado = LOW;                          // estado da entrada

void setup() {
  pinMode(LED,OUTPUT);                               // pino 4 como saída (LED)
  attachInterrupt(TECLA, trocaEstado, CHANGE);       // monitora eventos (mud. de estado)
}

void loop() {                  // Simula um longo processo/tarefa complexa em execução
  for (int i = 0; i < 100; i++)
    delay(10);                 // não faz nada além de gastar tempo (CPU ocupada)
}

void trocaEstado() {           // ISR para interrupção externa 0 (pino 2)
  estado = !estado;            // inverte o estado da tecla
  digitalWrite(LED,estado);    // mostra o estado no LED
}
```

Versão **sem** tratamento da trepidação dos contatos (*bounce* ou *chatter*)



Sketch **com** uso de Interrupção

- Comentários sobre o programa
  - ***`volatile`*** é um qualificador de variável (usado na declaração de ***estado***) que diz ao compilador para alocá-la na **memória RAM**, e não em registradores do µC
  - Registradores podem ser alterados por outras linhas de execução (***threads***) além da principal, ou seja, pelas ISR's disparadas por interrupções
  - Função: `attachInterrupt(param1,param2,param3)` faz a amarração entre a interrupção e a ISR a ser executada no atendimento à interrupção
    - param1 – número da interrupção a ser monitorada: 0 ou 1 (UNO/Duemilanove)
    - param2 - nome da rotina (ISR) a ser chamada na ocorrência do evento.
      Essa rotina não recebe parâmetro(s) e não retorna valor → sub-rotina
    - param3 - indica o tipo da condição a ser monitorada para disparar a interrupção:
      - **LOW**: a entrada estiver em nível baixo
      - **RISING**: a entrada mudar do nível baixo para o nível alto
      - **FALLING**: a entrada mudar do nível alto para o nível baixo
      - **CHANGE**: a entrada mudar de estado (baixo para alto ou alto para baixo)

Sketch com uso da Interrupção

```
#include <avr/interrupt.h>
int tecla = 0;                                          // interrupção número 0
int led = 4;                                            // saída digital no pino 4
volatile int estado = LOW, contador = LOW;              // estado da entrada
volatile long debounce = millis();

void setup() {
  // configura Pino.4 como saída
  pinMode(led,OUTPUT);
  // adiciona a interrupção ao pino da entrada digital e programa
  // para eventos de qualquer mudança no estado do pino
  attachInterrupt(tecla,trocaEstado,CHANGE);
  Serial.begin(9600);
}
void loop() {
  // Simulando um longo processo ou tarefa complexa em execução
  for (int i = 0; i < 100; i++) {
    // não faz nada além de gastar tempo (CPU ocupada)
    delay(10); }
  Serial.println(contador); contador = 0;
}

void trocaEstado() {
  if(millis() - debounce > 200) {
    estado = !estado;            // inverte o estado da tecla
    digitalWrite(led,estado);              // escreve o estado no led
    debounce = millis();
  }
  contador++;
}
```

Versão **com** tratamento da trepidação dos contatos (*bounce* ou *chatter*)

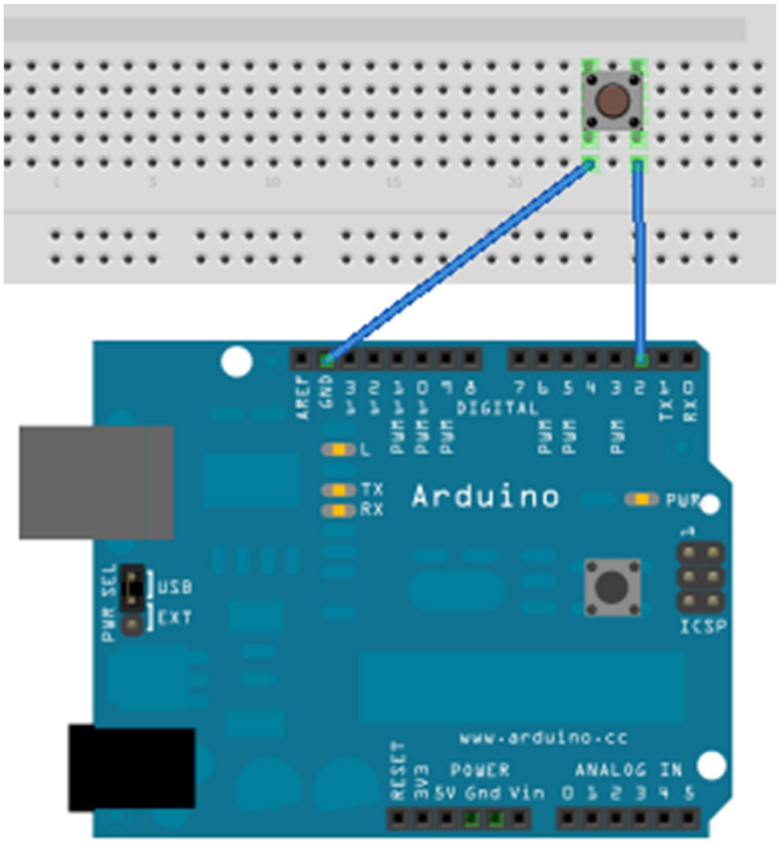
Manipulação de Registradores

```
int analogRead(uint8_t pin)     // faz a leitura de uma amostra de tensão no ADC
{
    uint8_t low, high;

    if (pin >= 14) pin -= 14;
    ADMUX = (analog_reference << 6) | (pin & 0x07);
    sbi(ADCSRA, ADSC);
    while (bit_is_set(ADCSRA, ADSC));
    low  = ADCL;
    high = ADCH;

    return (high << 8) | low;
}
```

Prática 1

Prática 1

- Detecção assíncrona de um tecla *push-button*



Prática 1

```
#include <avr/interrupt.h>

void setup(void)
{
    pinMode(2, INPUT);
    digitalWrite(2, HIGH);    // habilita resistor interno de pullup
    sei();                    // habilita interrupções globais
    EIMSK |= (1 << INT0);     // habilita interrupção externa INT0
    EICRA |= (1 << ISC01);    // dispara INT0 na borda de descida
    // attachInterrupt(0, pin2ISR, FALLING);  // só p/ inter.s ext.s
}

void loop(void)
{
}

// ISR programada no vetor INT0
ISR(EXT_INT0_vect)            // void pin2ISR(void)
{
    digitalWrite(13, !digitalRead(13));    // inverte LED no pino 13
}
```

Prática 1

```
int INT0 = 0;                    // Interrupt0 está no pino 2
int LED = 4;                     // pino do LED
volatile int state = LOW;        // estado do LED

void setup() {
  // configura o pino 4 como saída no LED
  pinMode(LED, OUTPUT);
  // amarra a interrupção à ISR e monitora mudança de estado no pino 2
  attachInterrupt(INT0, stateChange, CHANGE);
}

void loop() {
  // Simula um longo processo em execução
  for (int i = 0; i < 100; i++) {
    delay(10);          // não faz qualquer coisa a não ser gastar tempo
  }
}

void stateChange() {
  state = !state;
  digitalWrite(ledOut, state);
}
```

Exercício

- Mostre no LCD e no monitor Serial a quantidade de vezes que uma tecla (push-button) conectada ao pino 4 é pressionada, ao mesmo tempo em que se toca uma melodia em um buzzer conectado ao pino 3.

Mais sobre Interrupções

- Mudando ISR's Interrupções
  - As interrupções podem ter suas ISR's mudadas em qualquer ponto do sketch, usando o método `attachInterrupt()`
- Parando e Iniciando Interrupções
  - Em alguns casos uma parte do código pode precisar de execução sem interrupções, então use `noInterrupts()` para desativar as interrupções, e `interrupts()` para reativá-las
- Removendo Interrupções
  - Use o método `detachInterrup(número_da_interrupção)`

Exemplo

```
#include <avr/io.h>          // Definição dos nomes de interrução
#include <avr/interrupt.h>   // ISR - Interrupt Service Routine

int ledPin = 13;             // LED conectado ao pino digital 13
int sensePin = 2;            // pino da interrupção INT0 no ATMega8/168/328
volatile int valor = 0;      // variável na RAM

ISR(INT0_vect) {             // Instala a rotina de serviço da interrupção
  valor = digitalRead(sensePin);
}

void setup() {
  Serial.begin(9600);
  Serial.println("Iniciando a ISR...");
  pinMode(ledPin, OUTPUT);  // seta o pino digital como saída (output)
  pinMode(sensePin, INPUT); // seta o pino digital como entrada (input)
  GICR |= ( 1 < < INT0);    // habilitação global da interrupção INT0
  MCUCR |= ( 1 << ISC00);   // mudança de estado dispara a interrupção
  MCUCR |= ( 0 << ISC01);
  Serial.println("Iniciação encerrada.");
}

void loop() {
  if (valor) {
    Serial.println("Valor alto!");
    digitalWrite(ledPin, HIGH); }
  else {
    Serial.println("Valor baixo!");
    digitalWrite(ledPin, LOW); }
  delay(100);
}
```

Temporização

- delayMicroseconds(unsigned us)
  - Usa o Timer0 e não desabilita interrupções depois da Vs. 0018.
  - Acuidade para valores acima de 2 µs
  - Função embutida em wiring.c

```
/* Delay for the given number of microseconds. Assumes a 8 or 16 MHz clock. */
void delayMicroseconds(unsigned int us)
{
 // Calling avrlib's delay_Microseconds() with low values (1 or 2 us) gives delays longer than desired
 // for the 16 MHz clock on most Arduino boards.
 // For a one-microsecond delay, simply return. The overhead of the function call yields a delay of
 // approximately 1 1/8 us.

 if (--us == 0) return;

 // the following loop takes a quarter of a microsecond (4 cycles) per iteration, so execute it
 // four times for each microsecond of delay requested.
 us <<= 2;
 us -= 2;     // account for the time taken in the preceeding commands.

 // busy wait
 __asm__ __volatile__ (
         "1: sbiw %0,1" "\n\t"                    // 2 cycles
         "brne 1b" : "=w" (us) : "0" (us)         // 2 cycles
 );
}
```

Interrupção de **Timer**

- Timer1 é um temporizador de 16 bits
- Timer2 é um temporizador de 8 bits

Interrupção de Timer

```
// automaticamente inverte os pinos nas frequências dadas (400 e 415 Hz)
// gerando ondas quadradas que vc pode passa-las por um filtro RC para suaviza-las
// a frequência e' dada por: Fosc/(2*prescaler*TCCRn)
// o código seguinte funciona para usar timer1 ou timer2 no modo CTC no AtMega8

#include <avr/interrupt.h>

void setup() {
  pinMode(9, OUTPUT);    // OC1A = saída do modo CTC para timer1
  pinMode(11, OUTPUT);   // OC2 = mesmo para timer2
  TIMSK = 0x00;          // todas interrupções desligadas
  TCCR1A = 0x40;         // 0100 0000
  TCCR1B = 0x09;         // 0000 1001 essa linha e a anterior ==> modo CTC,
                         // inverte na comparação, prescaler = 1
  OCR1A = 19275;         // ~415Hz
  TCCR2 = 0x1E;          // 0001 1110 = modo CTC, inverte na comp., prescaler = 256
  OCR2 = 77;             // ~400Hz
  TIMSK = 0x90;          // 1001 0000 = 0CIE1 ON, OCIE2 ON
}

void loop() {
}
```

Interrupção de Timer

```
// automaticamente inverte os pinos nas frequências dadas (400 e 415 Hz)
// gerando ondas quadradas que vc pode passa-las por um filtro RC para suaviza-las
// a frequência e' dada por: Fosc/(2*prescaler*TCCRn)
// o código seguinte funciona para usar timer1 ou timer2 no modo CTC no Atmega168
// Alternativa: FrequencyTimer2 lib

#include <avr/interrupt.h>
#include <avr/io.h>

// manipulador do vetor de interrupcao overflow do Timer2, chamada
    16.000.000/(2*128*256) vezes por  seg.
ISR(TIMER2_OVF_vect) {
  // pino 10 indica o disparo da interrupcao
  digitalWrite(10,true);
}

void setup() {
  pinMode(9,OUTPUT); pinMode(10,OUTPUT);
  // configuracao do Timer2: Timer Prescaler/256, WGM modo 0
  TCCR2A = 0;
  TCCR2B = 1 << CS22 | 1 << CS21;
  TIMSK2 = 1 << TOIE2;            // habilita interrupcao Overflow do Timer2
  TCNT2  = 0;                     // reseta Timer2
  digitalWrite(9,true);           // led no pino 9 indica configuracao pronta
}

void loop() { }
```

Interrupção de Timer

```
#include <avr/interrupt.h>
#include <avr/io.h>
#define  INIT_TIMER_COUNT   6
int ledPin=13, contador=0, segAnt=0; volatile int segundo=0; long starttime=0;
// Arduino em 16 MHz: teremos 1000 overflows por segundo ((16000000/64)/250) = 1000
ISR(TIMER2_OVF_vect) {
  TCNT2 = INIT_TIMER_COUNT;                    // reseta o Timer2
  if (++contador == 1000) {
    second++; contador = 0; }
}
void setup() {
  Serial.begin(9600); Serial.println("Iniciando interrupção do Timer");
  TCCR2 |= (1 << CS22);                        // liga o bit CS22
  TCCR2 &= ~((1 << CS21) | (1 << CS20));       // desliga os bits CS21 e CS20
  // Usa o modo normal
  TCCR2 &= ~((1 << WGM21) | (1 << WGM20));     // desliga os bits WGM21 e WGM20
  // Usa o clock interno (clock externo nao e usado no Arduino)
  ASSR  |= (0 << AS2);
  TIMSK |= (1 << TOIE2) | (0 << OCIE2);        // habilita inter. de overflow Timer2
  TCNT2 = INIT_TIMER_COUNT;                    // reseta o Timer2
  sei();
  starttime = millis();
}
void loop() {
  if (segAnt!= second) {
    Serial.print(second); Serial.print(". ->");
    Serial.print(millis() - starttime); Serial.println(".");
    digitalWrite(ledPin, HIGH); delay(100); digitalWrite(ledPin, LOW); segAnt= segundo;
    }
}
```

Interrupção do **Timer2**: contador de 8 bits, auto-incrementável , aciona a ISR TIMER2_OVF_vect sempre que ocorrer um estouro de contagem (overflow)

Interrupção de Timer

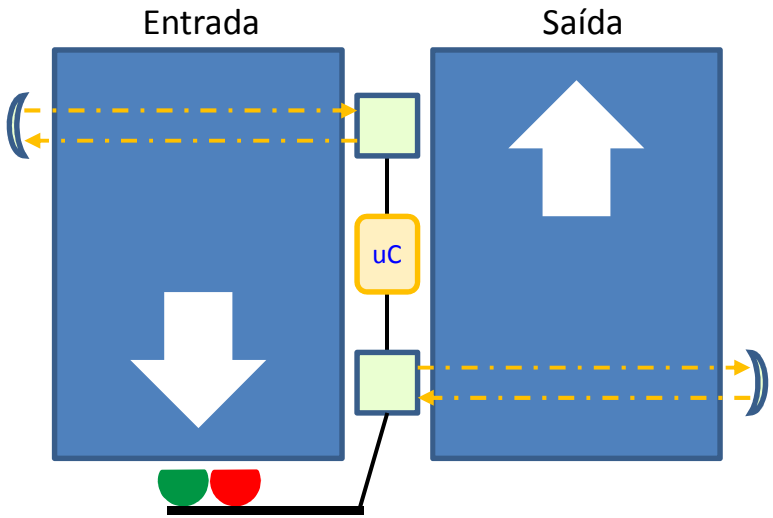
```
#include <avr/interrupt.h>
#include <avr/io.h>

#define INIT_TIMER_COUNT    6
#define RESET_TIMER2        TCNT2 = INIT_TIMER_COUNT

int ledPin = 13, int_counter = 0;
volatile int second = 0;
int oldSecond = 0;
long starttime = 0;

// Arduino runs at 16 Mhz, so we have 1000 Overflows per second...
// 1/ ((16000000 / 64) / 256) = 1 / 1000

ISR(TIMER2_OVF_vect)
{
  RESET_TIMER2;
  int_counter += 1;
  if (int_counter == 1000) {
    second++;
    int_counter = 0;
  }
}

// continua no próximo slide...
```

Interrupção de Timer

```
// continuação

void setup() {
  Serial.begin(9600); Serial.println("Configurando a interrupção do Timer2");
  // configuração do Timer2:   TimerPrescaler/64
  TCCR2 |= (1<<CS22);                    // liga o bit CS22
  TCCR2 &= ~((1<<CS21) | (1<<CS20));     // desliga os bits CS21 e CS20
  // Usa o modo normal
  TCCR2 &= ~((1<<WGM21) | (1<<WGM20));   // desliga os bits WGM21 e WGM20
  // usa o clock interno (clock externo não é usado no Arduino)
  ASSR  |= (0<<AS2);
  TIMSK |= (1<<TOIE2) | (0<<OCIE2);      // habilita interrupção por Overflow do Timer2
  RESET_TIMER2;
  sei();                                 // habilita todas interrupções
  starttime = millis();
}

void loop() {
  if (oldSecond != second) {
    Serial.print(second);
    Serial.print(" -> ");
    Serial.print(millis() - starttime);
    digitalWrite(ledPin, HIGH);
    delay(100);
    digitalWrite(ledPin, LOW);
    oldSecond = second;
  }
}
```

Projeto

- Contagem de estímulos externos (interrupções)
  - Controle de vagas de um estacionamento com semáforo
    - As vias de acesso e saída do estacionamento comportam apenas um veículo por vez, e o estacionamento contém N vagas. Faça a contagem de veículos entrantes/saintes. Se a quantidade de veículos que adentrarem o estacionamento for maior que N então acenda a luz vermelha. Caso contrário acenda a luz verde.



Fontes

- http://www.engblaze.com/we-interrupt-this-program-to-bring-you-a-tutorial-on-arduino-interrupts/
- http://arduino.cc/playground/Main/PinChangeIntExample