Software Embarcado

05 – Interrupções

Francisco Sant'Anna
Sala 6020-B
francisco@ime.uerj.br

http://github.com/fsantanna-uerj/SE

Exercício 1

- Piscar o LED a cada 1 segundo
- Parar ao pressionar o botão, mantendo o LED aceso para sempre (mesmo após soltar o botão)

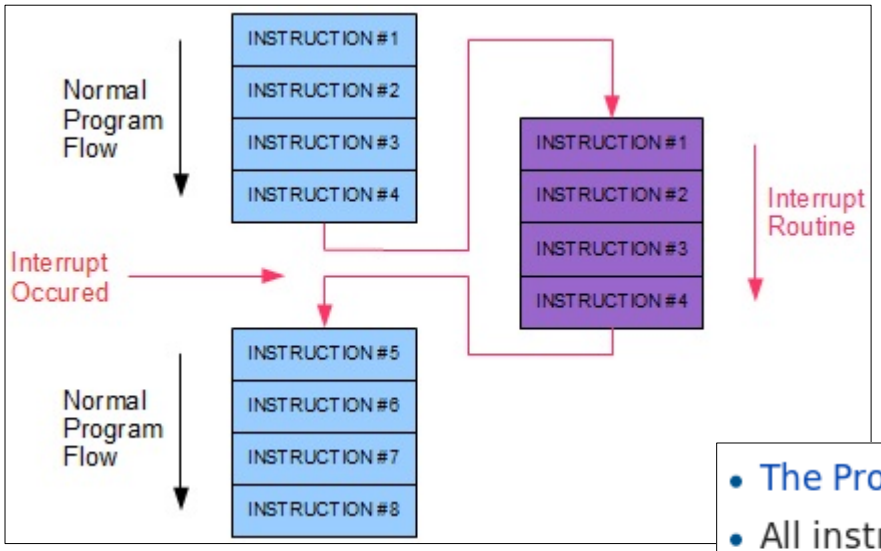
```
void loop () {
    digitalWrite(LED_PIN, HIGH);
    delay(1000);
    digitalWrite(LED_PIN, LOW);
    delay(1000);

    int but = digitalRead(BUT_PIN);
    if (but) {
        digitalWrite(LED_PIN, HIGH);
        while(1);
    }
}
```

- Programa não reativo!

Interrupções (ISRs)

In system programming, an **interrupt** is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention. An interrupt alerts the processor to a high-priority condition requiring the interruption of the current code the processor is executing. The processor responds by suspending its current activities, saving its state, and executing a function called an *interrupt handler* (or an interrupt service routine, ISR) to deal with the event. This interruption is temporary, and, after the interrupt handler finishes, the processor resumes normal activities.[1] There are two types of interrupts: hardware interrupts and software interrupts.



- The Program Counter (PC) is saved in a known place.
- All instructions before the one pointed to by the PC have fully executed.
- No instruction beyond the one pointed to by the PC has been executed, (
- The execution state of the instruction pointed to by the PC is known.

Características / Usos

- Alternativa ao *Polling*
- Reação Rápida
- Transmissão e Recepção em “Background”
- Temporizadores

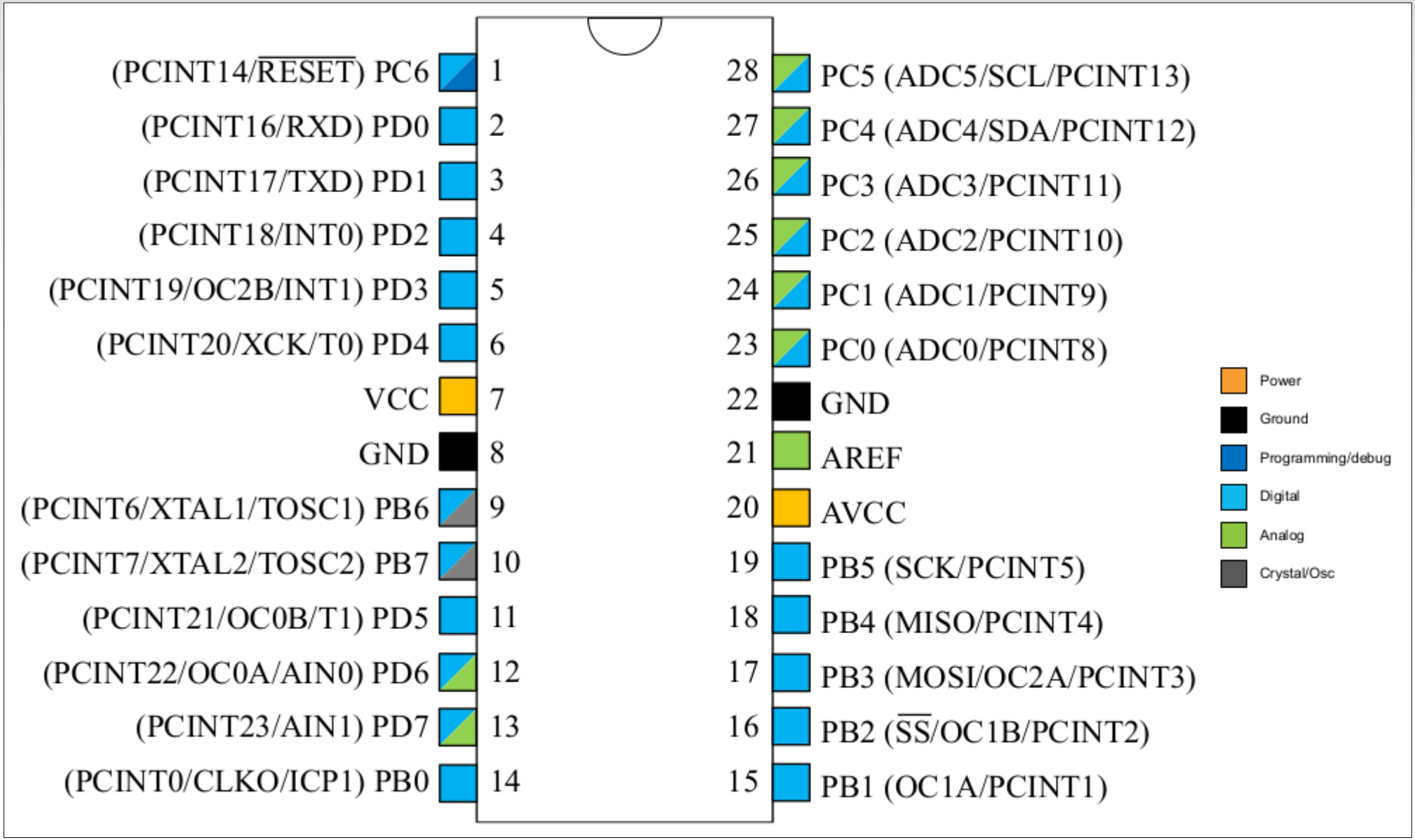
Observações

- Programa principal
  - para de executar (e outras ISRs também)
    - *starvation / deadlock*
  - em ponto não especificado
    - *race conditions*
- Cuidados
  - ISRs devem executar rápido
  - Regiões críticas devem desligar ISRs temporariamente

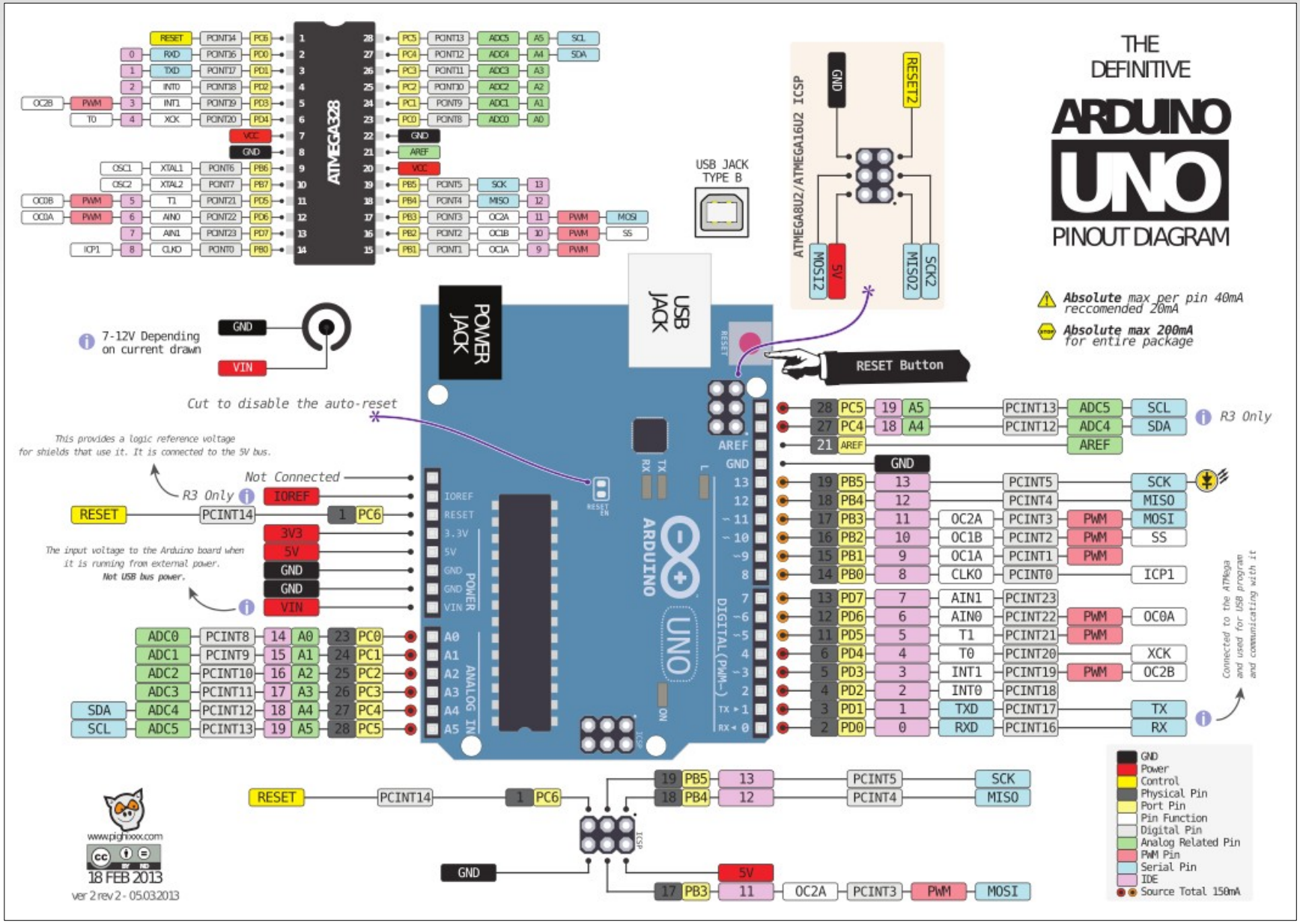
Sinais

Vector No	Program Address(2)	Source	Interrupts definition
1	0x0000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 0
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
7	0x000C	WDT	Watchdog Time-out Interrupt
8	0x000E	TIMER2_COMPA	Timer/Counter2 Compare Match A
9	0x0010	TIMER2_COMPB	Timer/Coutner2 Compare Match B
10	0x0012	TIMER2_OVF	Timer/Counter2 Overflow
11	0x0014	TIMER1_CAPT	Timer/Counter1 Capture Event
12	0x0016	TIMER1_COMPA	Timer/Counter1 Compare Match A
13	0x0018	TIMER1_COMPB	Timer/Coutner1 Compare Match B
14	0x001A	TIMER1_OVF	Timer/Counter1 Overflow
15	0x001C	TIMER0_COMPA	Timer/Counter0 Compare Match A
16	0x001E	TIMER0_COMPB	Timer/Coutner0 Compare Match B
17	0x0020	TIMER0_OVF	Timer/Counter0 Overflow
18	0x0022	SPI STC	SPI Serial Transfer Complete
19	0x0024	USART_RX	USART Rx Complete
20	0x0026	USART_UDRE	USART Data Register Empty
21	0x0028	USART_TX	USART Tx Complete
22	0x002A	ADC	ADC Conversion Complete
23	0x002C	EE READY	EEPROM Ready
24	0x002E	ANALOG COMP	Analog Comparator
25	0x0030	TWI	2-wire Serial Interface (I^2C)
26	0x0032	SPM READY	Store Program Memory Ready

Pinagem do Atmega328p



ATmega328p / Arduino UNO



External Interrupts

External Interrupts

- Associadas aos GPIOs (int0, int1)
- *Falling/Rising Edge*, *Low Level*
- Detecção é assíncrona
  - Funciona com o main clock desligado
    - (http://gammon.com.au/interrupts)
- Registradores: `SREG`, `EICRA`, `EMSK`, `EIFR`
- ISRs: `INT0_vect`, `INT1_vect`

SREG

11.3.1. Status Register

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

Name: SREG
Offset: 0x5F
Reset: 0x00
Property: When addressing as I/O Register: address offset is 0x3F

Bit	7	6	5	4	3	2	1	0
	I	T	H	S	V	N	Z	C
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit 7 – I: Global Interrupt Enable

The Global Interrupt Enable bit must be set for the interrupts to be enabled. The individual interrupt enable control is then performed in separate control registers. If the Global Interrupt Enable Register is cleared, none of the interrupts are enabled independent of the individual interrupt enable settings. The I-bit is cleared by hardware after an interrupt has occurred, and is set by the RETI instruction to enable subsequent interrupts. The I-bit can also be set and cleared by the application with the SEI and CLI instructions, as described in the instruction set reference.

EICRA

17.2.1. External Interrupt Control Register A

The External Interrupt Control Register A contains control bits for interrupt sense control.

Name: EICRA
Offset: 0x69
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
					ISC11	ISC10	ISC01	ISC00
Access					R/W	R/W	R/W	R/W
Reset					0	0	0	0

Bits 3:2 – ISC1n: Interrupt Sense Control 1 [n = 1:0]

The External Interrupt 1 is activated by the external pin INT1 if the SREG I-flag and the corresponding interrupt mask are set. The level and edges on the external INT1 pin that activate the interrupt are defined in the table below. The value on the INT1 pin is sampled before detecting edges. If edge or toggle interrupt is selected, pulses that last longer than one clock period will generate an interrupt. Shorter pulses are not guaranteed to generate an interrupt. If low level interrupt is selected, the low level must be held until the completion of the currently executing instruction to generate an interrupt.

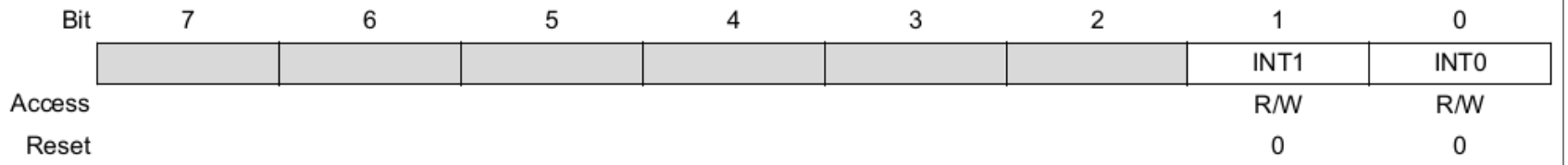
Value	Description
00	The low level of INT1 generates an interrupt request.
01	Any logical change on INT1 generates an interrupt request.
10	The falling edge of INT1 generates an interrupt request.
11	The rising edge of INT1 generates an interrupt request.

EIMSK

17.2.2. External Interrupt Mask Register

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

Name: EIMSK
Offset: 0x3D
Reset: 0x00
Property: When addressing as I/O Register: address offset is 0x1D



Bit 1 – INT1: External Interrupt Request 1 Enable

When the INT1 bit is set and the I-bit in the Status Register (SREG) is set, the external pin interrupt is enabled. The Interrupt Sense Control1 bits 1/0 (ISC11 and ISC10) in the External Interrupt Control Register A (EICRA) define whether the external interrupt is activated on rising and/or falling edge of the INT1 pin or level sensed. Activity on the pin will cause an interrupt request even if INT1 is configured as an output. The corresponding interrupt of External Interrupt Request 1 is executed from the INT1 Interrupt Vector.

EIFR

17.2.3. External Interrupt Flag Register

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

Name: EIFR
Offset: 0x3C
Reset: 0x00
Property: When addressing as I/O Register: address offset is 0x1C

Bit	7	6	5	4	3	2	1	0
							INTF1	INTF0
Access							R/W	R/W
Reset							0	0

Bit 1 – INTF1: External Interrupt Flag 1

When an edge or logic change on the INT1 pin triggers an interrupt request, INTF1 will be set. If the I-bit in SREG and the INT1 bit in EIMSK are set, the MCU will jump to the corresponding Interrupt Vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing '1' to it. This flag is always cleared when INT1 is configured as a level interrupt.

Exemplo

```
void setup() {
    pinMode( 2, INPUT_PULLUP);
    pinMode(12, OUTPUT);
    pinMode(13, OUTPUT);

    EICRA = (1 << ISC10) | (0 << ISC11);
    EIMSK = (1 << INT1);
}

void loop () {
    digitalWrite(13, HIGH);
    delay(1000);
    digitalWrite(13, LOW);
    delay(1000);
}

ISR(INT1_vect) {
    digitalWrite(12, digitalRead(3));
}
```

AVR-GCC

https://www.nongnu.org/avr-libc/user-manual/group__avr__interrupts.html

Introduction to avr-libc's interrupt handling

It's nearly impossible to find compilers that agree on how to handle interrupt code. Since the C language tries to stay away from machine dependent details, each compiler writer is forced to design their method of support.

In the AVR-GCC environment, the vector table is predefined to point to interrupt routines with predetermined names. By using the appropriate name, your routine will be called when the corresponding interrupt occurs. The device library provides a set of default interrupt routines, which will get used if you don't define your own.

Patching into the vector table is only one part of the problem. The compiler uses, by convention, a set of registers when it's normally executing compiler-generated code. It's important that these registers, as well as the status register, get saved and restored. The extra code needed to do this is enabled by tagging the interrupt function with `__attribute__((signal))`.

These details seem to make interrupt routines a little messy, but all these details are handled by the Interrupt API. An interrupt routine is defined with **ISR()**. This macro register and mark the routine as an interrupt handler for the specified peripheral. The following is an example definition of a handler for the ADC interrupt.

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

ISR(ADC_vect)
{
    // user code here
}
```

Refer to the chapter explaining **assembler programming** for an explanation about interrupt routines written solely in assembler language.