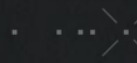
FIAP



FIAP

Engenharia de Software

EDGE COMPUTING & COMPUTER SYSTEMS

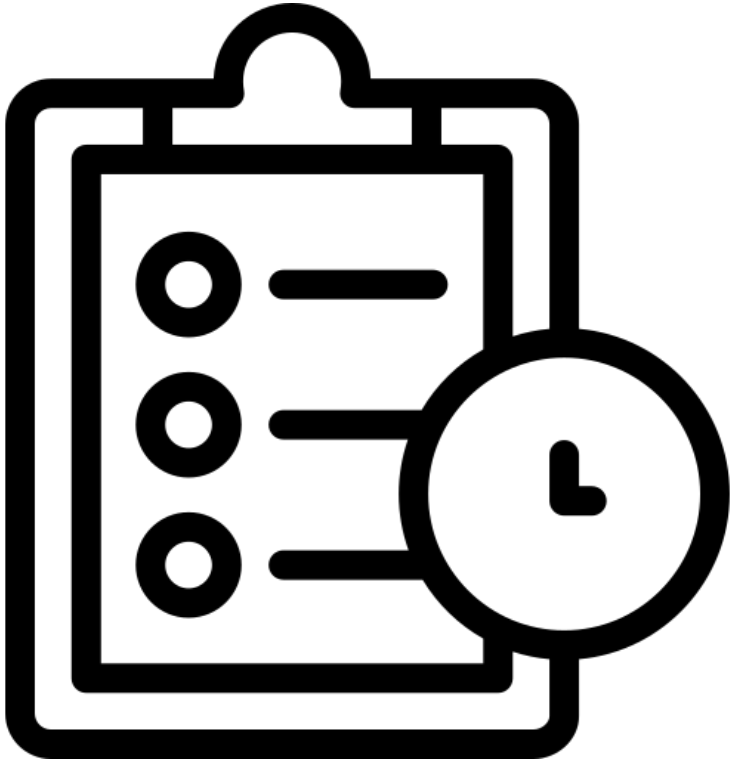
11 – Bootloader e Interrupções


Prof. Airton Y. C. Toyofuku


profairton.toyofuku@fiap.com.br

Agenda

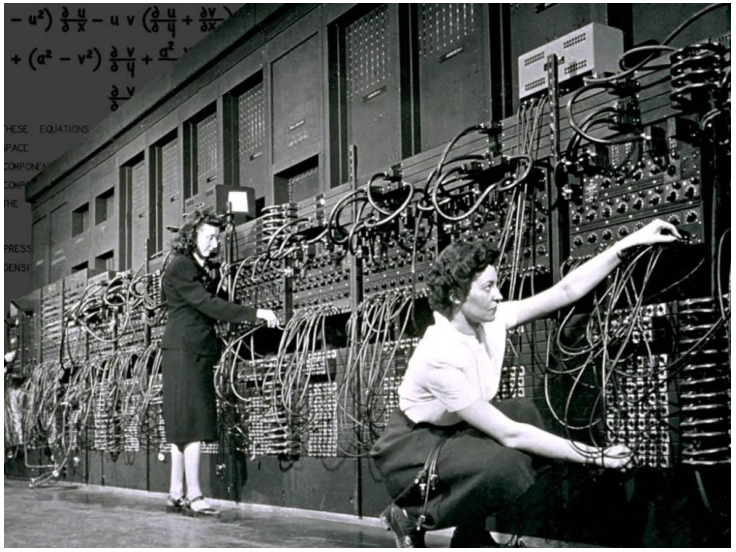
- O que é Bootloader?
- Como funciona?
- O que é uma Interrupção?
- Interrupções Externas no Arduino;
- Laboratório;
- Interrupções Internas no Arduino;
- Laboratório;
- Exercício;

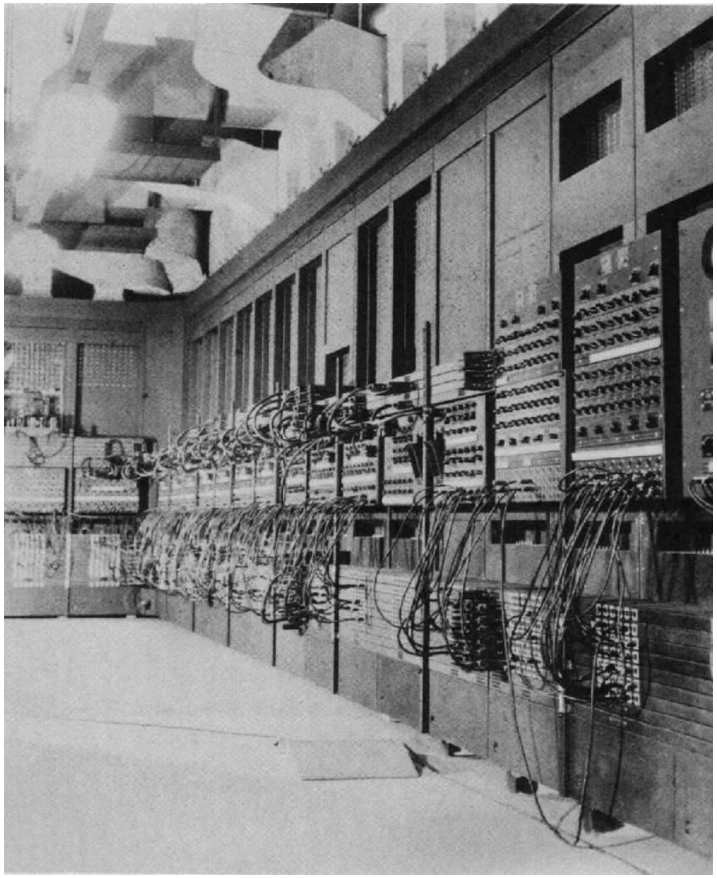


O que é Bootloader

Antigamente, os primeiros computadores eram programados mecanicamente, ou através de cabos ou por cartões perfurados.

Ou seja, alguém dizia para o computador o que deveria ser feito assim que fosse ligado.





Fonte: https://witchdoctor.co.nz/index.php/2021/03/the-worlds-first-digital-computer-is-76-years-old/

O que é Bootloader

Atualmente, com o avanço da tecnologia, temos computadores puramente eletrônicos, que “rodam” programas quase que sem intervenção humana.

Estamos familiarizados com esse tipo de computador. Que assim que ligamos, ele “boota” um Windows, um Linux ou mesmo um MAC OS.

Mas antes dele nos mostrar telinhas bonitas, o computador apresenta um negócio chamado “BIOS”

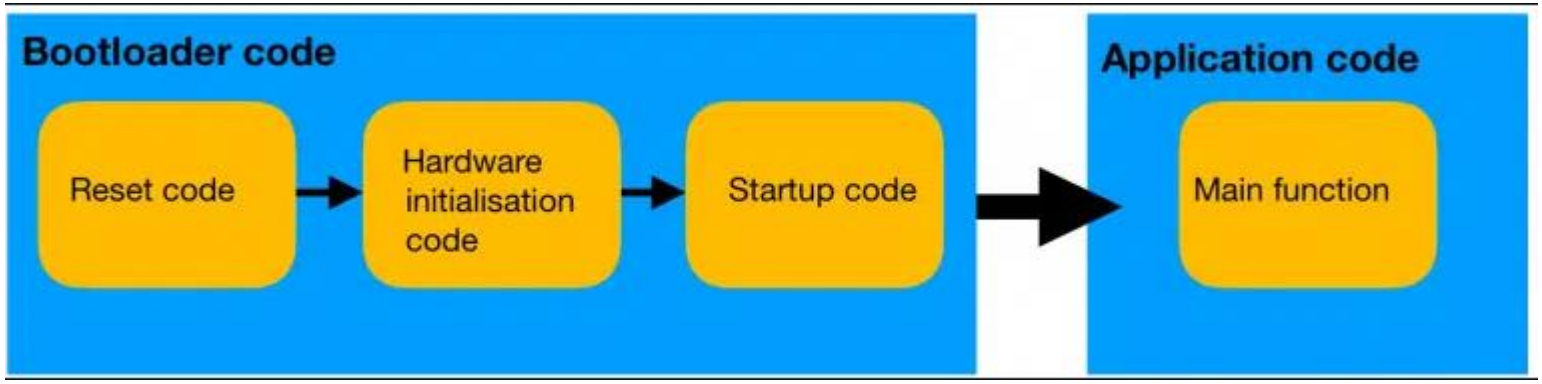
A BIOS, é o programa integrado ao processador ou placa mãe do computador, sendo responsável por verificar a integridade do hardware, carregar os drivers básicos e “chamar” o sistema operacional.

O que é Bootloader

Temos **BIOS** em **uControladores**? Sim e não...

Temos um software pequeno, seguro e confiável (em teoria), chamado **Bootloader**

O Bootloader tem a função de carregar na memória de programa o firmware definitivo que vai rodar no produto.

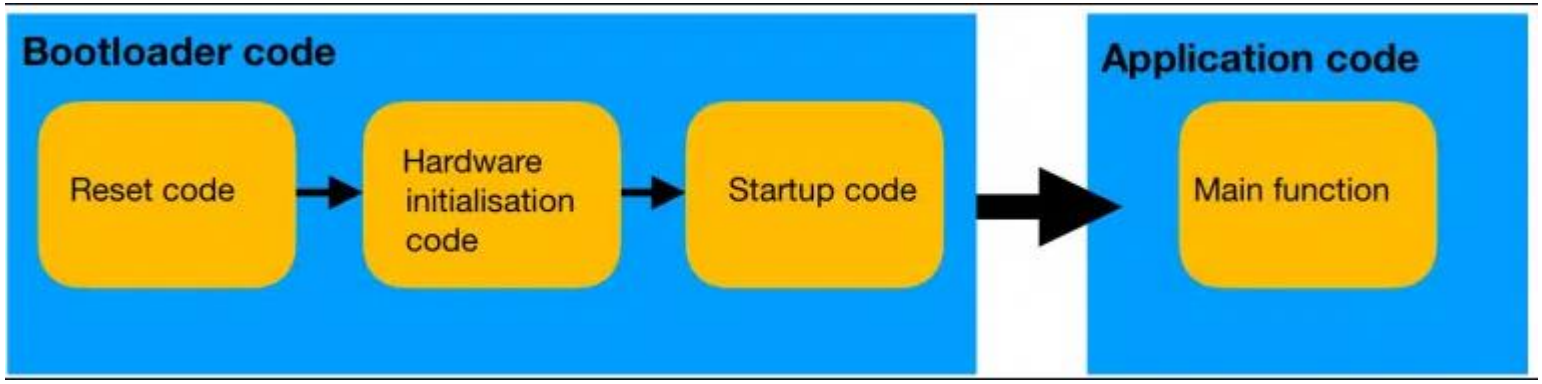


O que é Bootloader

Temos **BIOS** em **uControladores**? Sim e não...

Temos um software pequeno, seguro e confiável (em teoria), chamado **Bootloader**

O Bootloader tem a função de Inicializar os periféricos básicos do uControlador como a Bios, mas também tem o papel de atualizar o software do dispositivo.

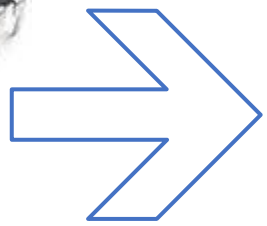


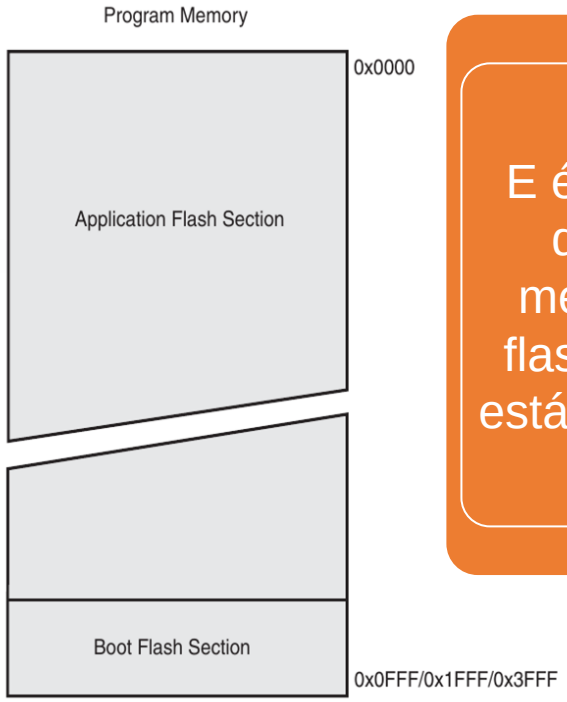
E como funciona?

MAS COMO ASSIM ?

O Bootloader também tem o papel de atualizar o software do dispositivo????

Vamos tentar simplificar: Esse é o noso uControlador usado no Arduino

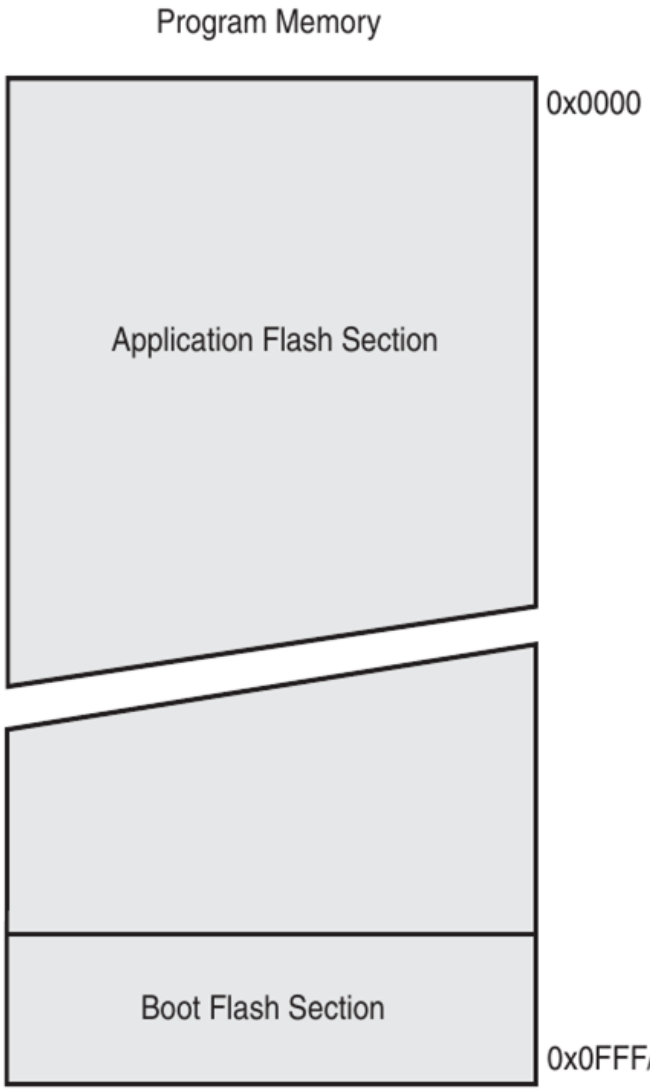




E é assim que a memória flash (HD) está dividida

Fonte: http://www.rjhcoding.com/avr-asm-memory-types.php

E como funciona?

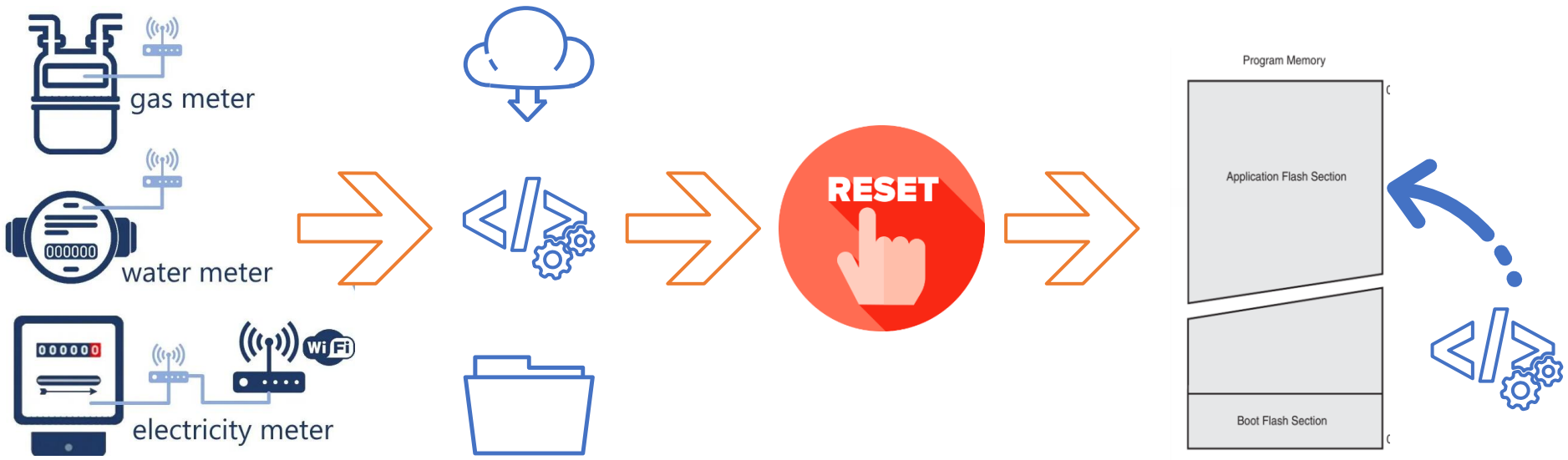


Fonte: http://www.rjhcoding.com/avr-asm-memory-types.php

Application Flash Section é onde está o nosso programa principal.
É onde roda o software que escrevemos no IDE do Arduino.

O Boot Flash Section é onde está o bootloader, um software que normalmente não temos acesso e tem a função de pegar o software que escrevemos na IDE e colocar no Application Flash Section

E como funciona?



Fonte: https://www.freva.com/wp-content/uploads/2019/10/smart-metering.jpg

01

Temos diversos dispositivos em campo e precisamos atualizar o software deles.

02

O nosso sistema vai fazer o download do novo software e armazenar numa “pasta” de download

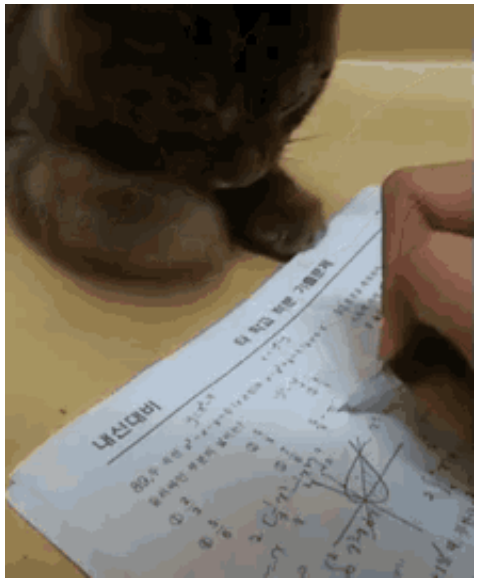
03

Assim como quando atualizamos o Win, o sistema faz um reset

04

No reset, o bootloader assume a tarefa de pegar o software da pasta de download e colocar na memória principal de programa

O que é uma interrupção?



Imagine que você está tentando estudar para uma prova muito difícil, e o seu gato fica sempre te interrompendo.

O gato é um animal persistente. Ele vai fazer você parar qualquer coisa que estiver fazendo para dar atenção para ele.

E ele só vai sossegar quando estiver satisfeito.

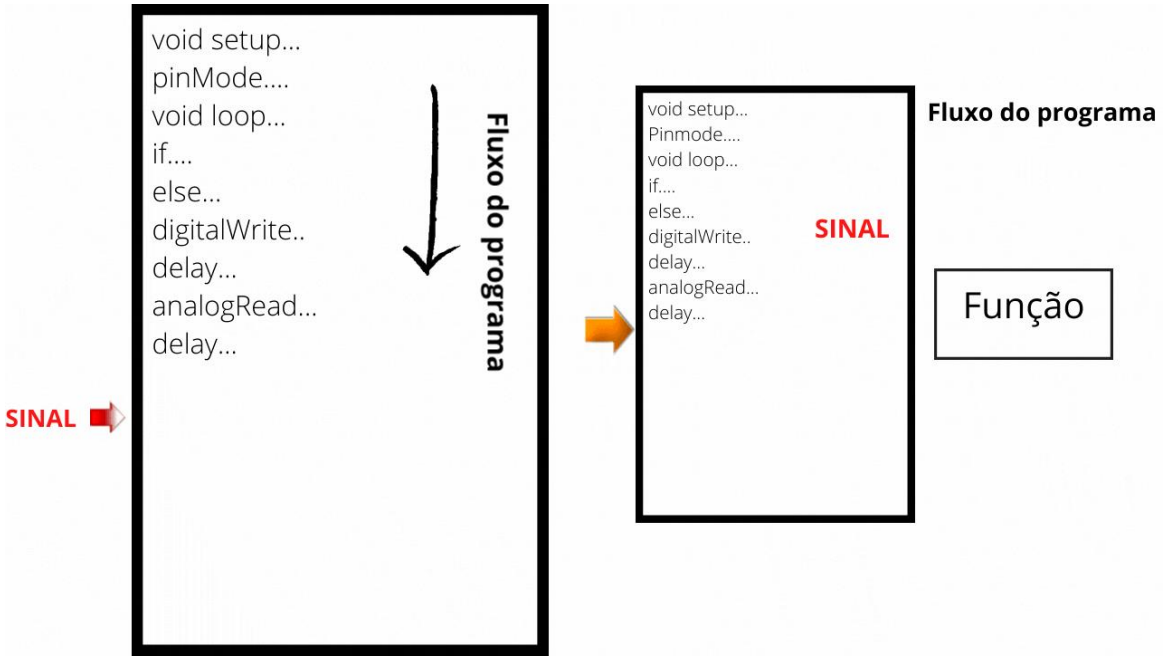
Agora vamos trocar **Você** por um **Computador**.

A tarefa de **estudar** por uma tarefa executada por um **software** qualquer

E o **Gato** por um evento externo, seja um **botão**, um **alarme**, ou uma **situação que acontece de vez em quando**.

O que é uma interrupção?

- Ela interrompe o fluxo normal do código ao receber um sinal **interno** ou **externo** que pode acontecer a **qualquer momento**.
- É chamada de **ISR** (Interrupt Service Routine)
- Ela não recebe **parâmetros** e nem tem **retornos**. Deve ser o mais **curta** possível e **todas** as **variáveis globais** utilizadas devem ser do tipo **volatile**.
- **Volatile** significa que a variável pode ser modificada **sem** que o programa principal saiba, desta forma o compilador não tenta otimizar o uso desta variável.



Fonte: https://blog.eletrogate.com/interrupcao-o-que-e-como-utilizar-arduin/

Interrupções Externas no Arduino

FIAP

- Imagine que você está estudando em casa, mas ao invés do gato, alguém toca a campainha. Você para o que está fazendo para atender a porta, certo?
- Essa é a ideia da interrupção externa, que pode ser um sensor, um botão ou outra placa
- No Arduino Uno, só dois pinos podem ser usado como interrupção externa, o pino 2 e 3.
- Cada placa tem pinos diferentes, por isso sempre verifique a documentação disponível: https://www.arduino.cc/reference/pt/language/functions/external-interrupts/attachinterrupt/

Primeiro, precisamos colocar o pino que vamos utilizar como interrupção no modo input: **pinMode(InterruptPin, Input);**

Em seguida usamos a função **attachInterrupt().** Essa função recebe três parâmetros, sendo:

1 – A função **digitalPinToInterrupt()** para converter o sinal de Input em interrupção
2 – A função que vai tratar a interrupção
3 – O modo em que a interrupção vai ocorrer;

Os modos podem ser:

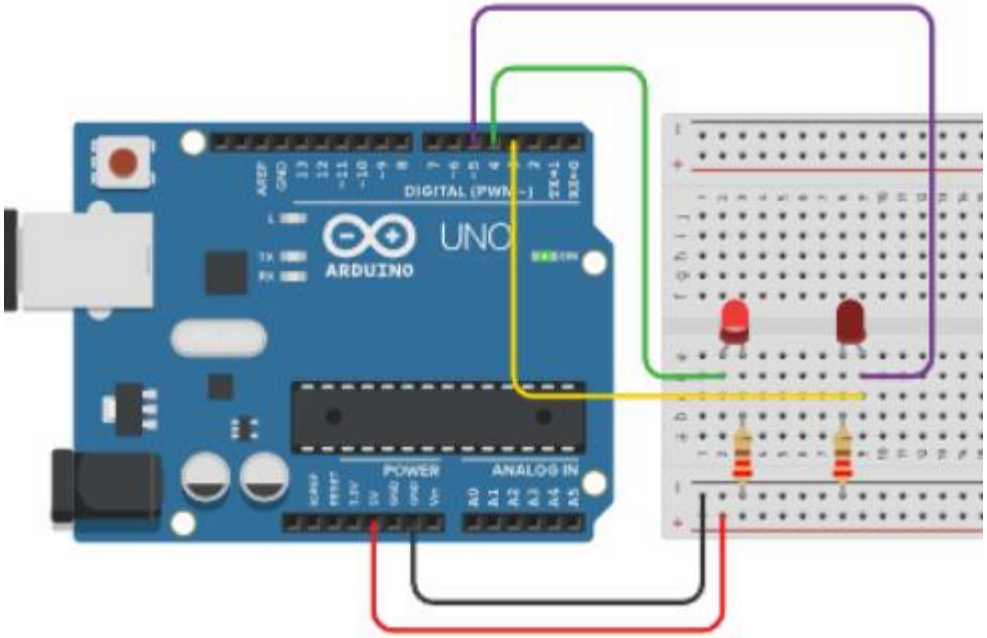
- **LOW** acionar a interrupção quando o estado do pino for LOW.
- **CHANGE** acionar a interrupção quando o sempre estado do pino mudar.
- **RISING** acionar a interrupção quando o estado do pino for de LOW para HIGH apenas.
- **FALLING** acionar a interrupção quando o estado do pino for de HIGH para LOW apenas.

Laboratório – Interrupções Externas

Neste laboratório, vamos explorar o conceito de interrupção e verificar como o software se comporta.

Vamos usar o LED 01 para acender o LED 02.

Mesmo fora do loop o primeiro led pisca, isso ocorre pois a cada vez que o segundo led muda de estado (de HIGH para LOW e vice-versa) ele chama a ISR “inverte_led”, saindo do loop, executando a função e retornando logo em seguida.



Material necessário:

- 1 Arduino;
- 2 LEDs;
- 2 Resistores de 220
- Jumpers

Link: Projeto 18 – Interrupção Externa

Interrupções Internas no Arduino

- Imagine que você está fazendo um bolo em casa e coloca um alarme para avisar que o bolo está pronto. Enquanto isso você vai fazer as suas tarefas. O que acontece quando o alarme toca?
- Essa é a ideia da interrupção interna, que são alarmes ou eventos dentro do uControlador.
- No Arduino Uno, possui 26 tipos de interrupções, sendo 6 externas e 19 internas.

Vector No.	Program Address	Source	Interrupt Definition
1	0x0000	RESET	External pin, power-on reset, brown-out reset and watchdog system reset
2	0x002	INT0	External interrupt request 0
3	0x0004	INT1	External interrupt request 1
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/Counter2 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/Counter1 compare match B
14	0x001A	TIMER1 OVF	Timer/Counter1 overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 compare match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 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
26	0x0032	SPM READY	Store program memory ready

(Fonte: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf, Pag 49)

Interrupções Internas no Arduino

- Vamos ver a interrupção relacionada aos Timers.
- Timers são contadores que a cada pulso do clock incrementam de uma unidade, ou decrementam de uma unidade.
- Quando o Timer atinge o seu valor máximo (Overflow) ou mínimo (0), dizemos que o timer estourou e gerou uma interrupção.
- O IDE Arduino não disponibiliza funções para acessar interrupções internas diretamente, por isso, vamos ter de mexer com registradores internos.

Registradores

São células de memória, normalmente de 8 bits, em que cada bit contém um parâmetro de configuração. Também podem ser de um tipo especial, em que seu conteúdo é uma informação ou ponteiro de controle

1 1 0 1 0 0 0 1 0

Fonte: https://blog.eletrogate.com/interrupcao-o-que-e-como-utilizar-arduin/

1 representa o nível lógico HIGH (ligado) e 0 (zero) representa o nível lógico LOW (desligado). Assim como uma máquina de lavar roupa ou um micro-ondas, a sequência de botões define sua funcionalidade

Interrupções Internas no Arduino

- O uControlador do Arduino Uno tem 3 registradores para controlar os Timers: TCCRxA, TCCRxB e TIMSKx, sendo “x” a identificação do Timer
- Podemos ter informações sobre esses registradores no datasheet do Chip, facilmente encontrado no site do fabricante.

15.11 Register Description

15.11.1 TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:6 – COM1A1:0: Compare Output Mode for Channel A**
- **Bit 5:4 – COM1B1:0: Compare Output Mode for Channel B**

The COM1A1:0 and COM1B1:0 control the output compare pins (OC1A and OC1B respectively) behavior. If one or both of the COM1A1:0 bits are written to one, the OC1A output overrides the normal port functionality of the I/O pin it is connected to. If one or both of the COM1B1:0 bit are written to one, the OC1B output overrides the normal port functionality of the I/O pin it is connected to. However, note that the data direction register (DDR) bit corresponding to the OC1A or OC1B pin must be set in order to enable the output driver.

When the OC1A or OC1B is connected to the pin, the function of the COM1x1:0 bits is dependent of the WGM13:0 bits setting. Table 15-2 shows the COM1x1:0 bit functionality when the WGM13:0 bits are set to a normal or a CTC mode (non-PWM).

15.11.2 TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – ICNC1: Input Capture Noise Canceler**

Setting this bit (to one) activates the input capture noise canceler. When the noise canceler is activated, the input from the input capture pin (ICP1) is filtered. The filter function requires four successive equal valued samples of the ICP1 pin for changing its output. The input capture is therefore delayed by four oscillator cycles when the noise canceler is enabled.

- **Bit 6 – ICES1: Input Capture Edge Select**

This bit selects which edge on the input capture pin (ICP1) that is used to trigger a capture event. When the ICES1 bit is written to zero, a falling (negative) edge is used as trigger, and when the ICES1 bit is written to one, a rising (positive) edge will trigger the capture.

When a capture is triggered according to the ICES1 setting, the counter value is copied into the input capture register (ICR1). The event will also set the input capture flag (ICF1), and this can be used to cause an input capture interrupt, if this interrupt is enabled.

When the ICR1 is used as TOP value (see description of the WGM13:0 bits located in the TCCR1A and the TCCR1B register), the ICP1 is disconnected and consequently the input capture function is disabled.

- **Bit 5 – Reserved Bit**

This bit is reserved for future use. For ensuring compatibility with future devices, this bit must be written to zero when TCCR1B is written.

15.11.8 TIMSK1 – Timer/Counter1 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6F)	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7, 6 – Res: Reserved Bits**

These bits are unused bits in the Atmel® ATmega328P, and will always read as zero.

- **Bit 5 – ICIE1: Timer/Counter1, Input Capture Interrupt Enable**

When this bit is written to one, and the I-flag in the status register is set (interrupts globally enabled), the Timer/Counter1 input capture interrupt is enabled. The corresponding interrupt vector (see Section 11. “Interrupts” on page 49) is executed when the ICF1 flag, located in TIFR1, is set.

- **Bit 4, 3 – Res: Reserved Bits**

These bits are unused bits in the Atmel ATmega328P, and will always read as zero.

- **Bit 2 – OCIE1B: Timer/Counter1, Output Compare B Match Interrupt Enable**

When this bit is written to one, and the I-flag in the status register is set (interrupts globally enabled), the Timer/Counter1 output compare B match interrupt is enabled. The corresponding interrupt vector (see Section 11. “Interrupts” on page 49) is executed when the OCF1B flag, located in TIFR1, is set.

- **Bit 1 – OCIE1A: Timer/Counter1, Output Compare A Match Interrupt Enable**

When this bit is written to one, and the I-flag in the status register is set (interrupts globally enabled), the Timer/Counter1 output compare A match interrupt is enabled. The corresponding interrupt vector (see Section 11. “Interrupts” on page 49) is executed when the OCF1A flag, located in TIFR1, is set.

Interrupções Internas no Arduino

FIAP

- Nosso objetivo agora será implementar um contador no modo normal. O datasheet do uControlador nos diz como deve ser a configuração dos bits dos registradores

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, phase correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, phase correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, phase correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, phase and frequency correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, phase and frequency correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, phase correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, phase correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

Note: 1. The CTC1 and PWM11:0 bit definition names are obsolete. Use the WGM12:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.

- Como podemos ver pela tabela, o modo normal é atingido quando os bits **WGM13, WGM12, WGM11 e WGM10** estão em zero.

15.11.1 TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Agora precisamos definir o quão rápido esse contador vai contar. Para isso usamos os bits **CS12, CS11 e CS10,** referentes ao **Prescaler**

15.11.2 TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Interrupções Internas no Arduino

- O **PRESCALER** é um fator de divisão do clock usado para aumentar ou diminuir a velocidade do timer, e é dado pela fórmula

$$T = \frac{(max + 1) \times Prescaler}{Frequencia}$$

- *T* é o tempo em microssegundos;
- *max* é o valor máximo que o timer atinge (255, para um Timer de 8 bits);
- *Prescaler* é o fator de divisão definido no último tópico;
- *frequência* é a frequência do nosso microcontrolador, que, no Arduino Uno, é de aproximadamente 16Mhz.

- Se definirmos um prescaler de 256, então o tempo para o contador chegar ao máxmio será de:

$$T = \frac{(255 + 1) \times 256}{16} = 4096uS$$

- Depois de configurar o prescaler, precisamos habilitar a interrupção, definindo o bit TOIEx no registrador TIMSKx

15.11.8 TIMSK1 – Timer/Counter1 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6F)	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Por fim, habilitamos as interrupções globais chamando a função **sei()** e criamos a função que vai tratar a interrupção através do nome **ISR(TIMER1_OVF_vect);**

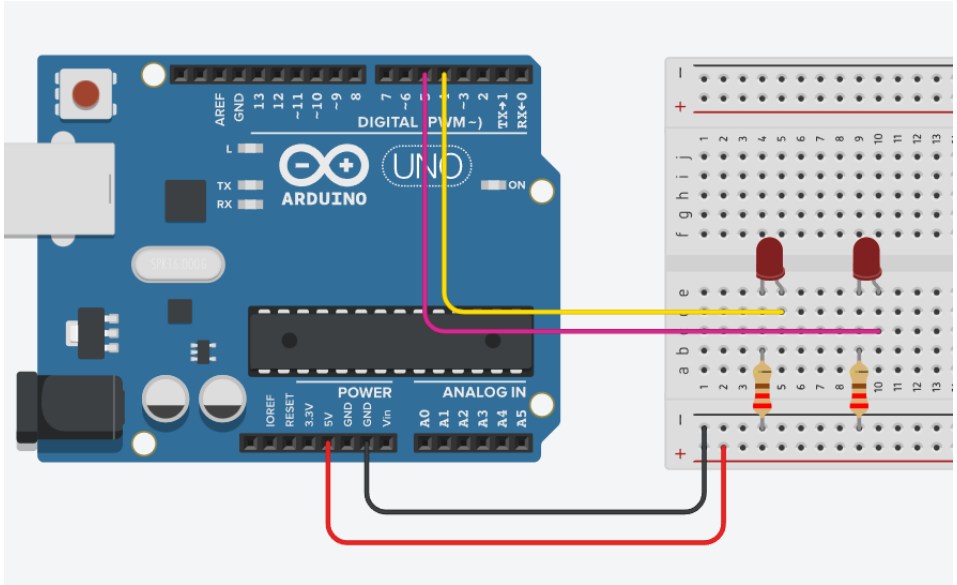
Laboratório – Interrupções Internas

Neste laboratório, vamos explorar o conceito de interrupção externa e verificar como o software se comporta.

Vamos usar o LED 01 para acender o LED 02.

A cada 1 segundo o segundo led pisca, sem interferir no primeiro led. Isso ocorre pois demora cerca de 1 segundo para o timer 1 contar de 0 a 65536 com um prescaler de 256 e a cada vez que ele atinge o máximo ele envia um sinal de interrupção, que chama nossa ISR, e volta a contagem para 0.

Link: Projeto 19 – Interrupção Interna

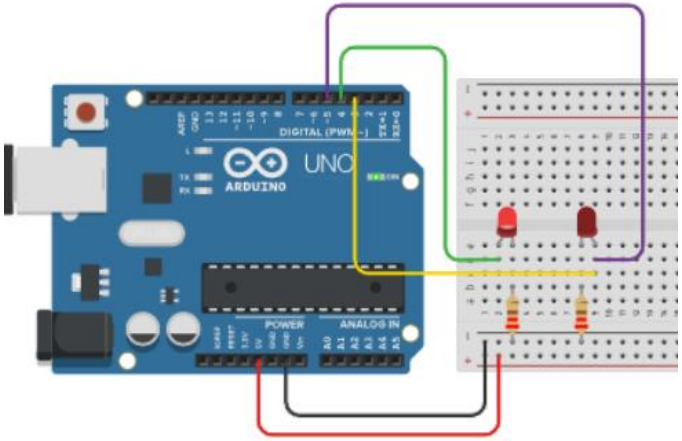


Material necessário:

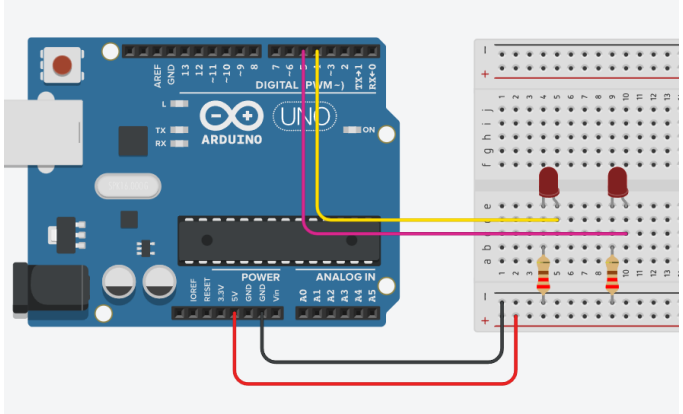
- 1 Arduino;
- 2 LEDs;
- 2 Resistores de 220
- Jumpers

Exercícios

1. Altere o laboratório de interrupções externas para utilizar um botão ao invés do sinal do LED.



1. Altere o laboratório de interrupções internas para valores de tempo de 0.25, 0.5, 1.5, 2.0, 2.5 e 3 segundos.



This presentation has been designed using images from Flaticon.com