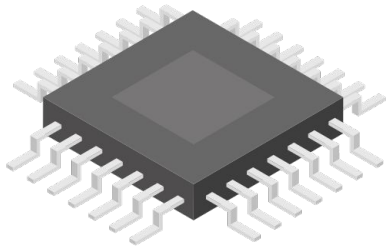
PADO

Labs

Microcontroladores



Prof.º: Pablo Jean Rozário

pablo.jean@padotec.com.br

/in/pablojeanrozario

https://github.com/Pablo-Jean

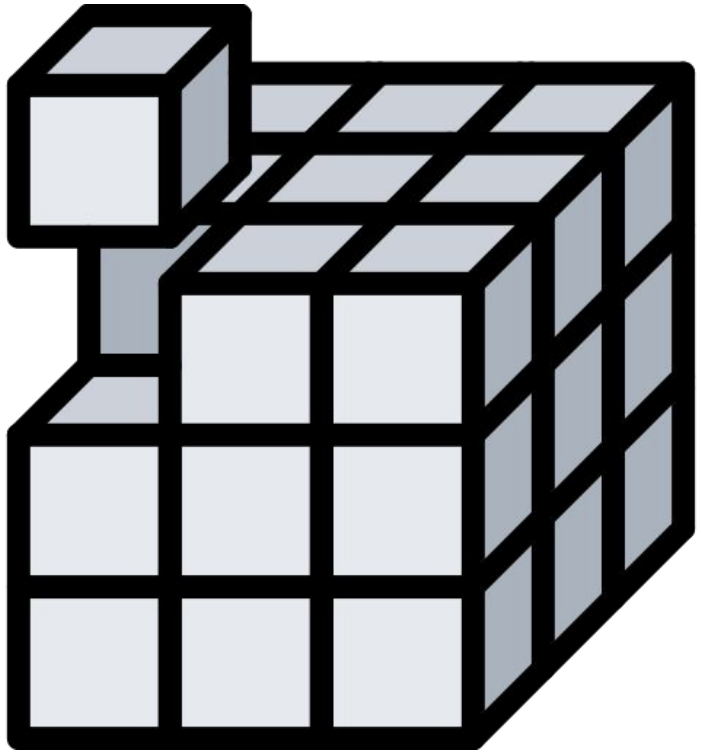
Estruturas em C e Interrupções

Índice da Aula #3

- Estruturas em C
- Typedefs, Enumerates, Structs e Unions
- Interrupções
- Fluxo de Interrupções
- NVIC
- Dicas
- Interrupções de GPIO no STM32
- Lista de Exercícios #3

PADO Labs

ESTRUTURAS EM C



Estruturas em C

Importante conhecermos pois auxilia na organização de códigos e legibilidade. Além de que, muitas SDKs utilizam estes recursos.

Iniciaremos pelo mais simples: **typedefs**

Typedefs

Typedefs são um recurso importante em C, que serve para criar novos tipos de variáveis, a partir de primitivos ou estruturas. Possui a seguinte declaração:

```
typedef antigo_nome novo_nome;
```

Com isso, podemos criar uma nova variável e instanciar como se fosse um tipo comum.

Typedefs

Como por exemplo, podemos chamar um variável *int* como *integer* utilizando o **typedef**.

```
typedef int integer;
```

```
integer value;
```

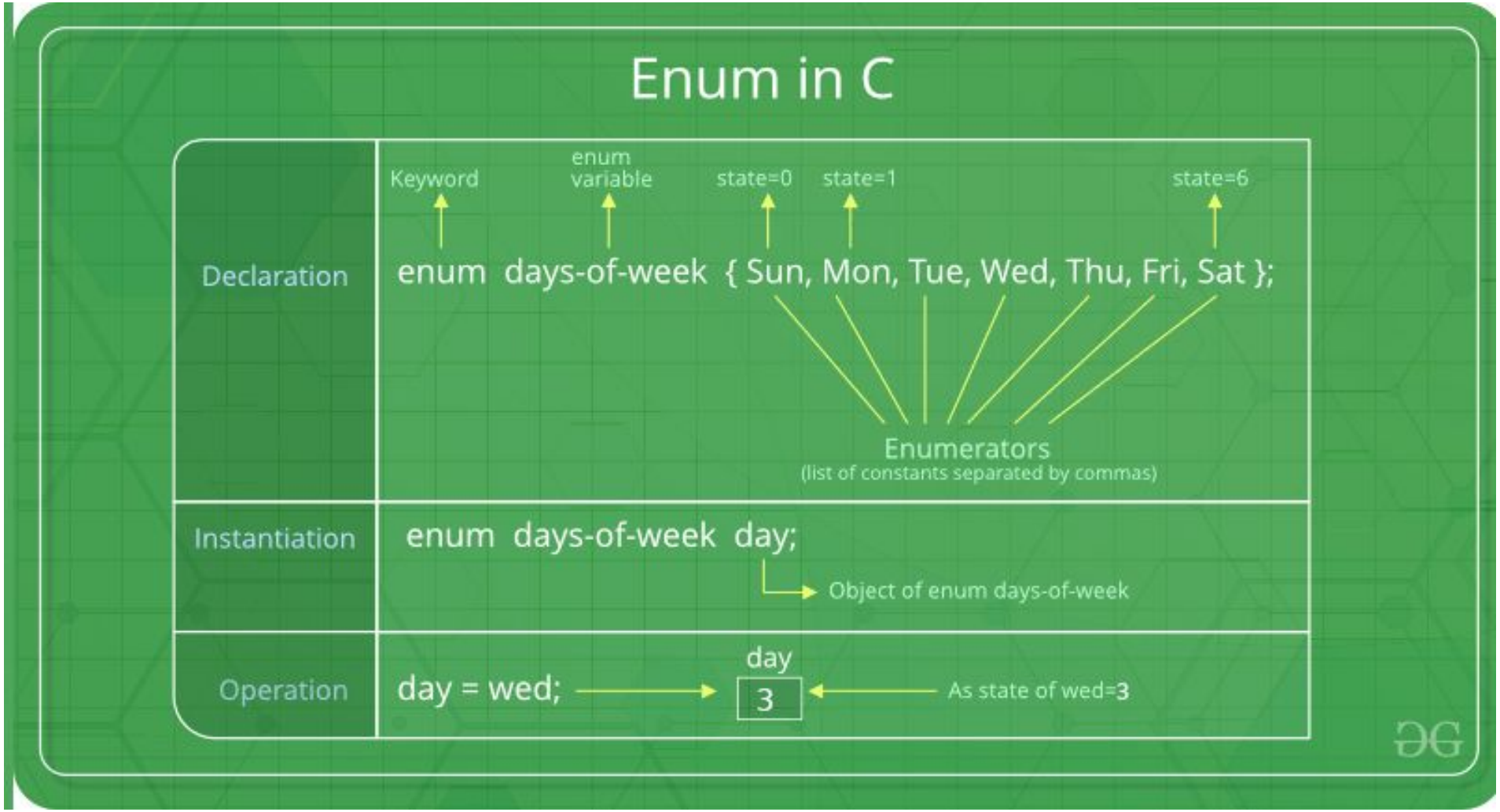
Podemos utilizar o typedef também em *structs*, *unions* e *enums*.

Enumerates

Os enumerates são um tipo de dado definido pelo desenvolvedor que é utilizado para atribuir um **nome** a uma **constante**. Este recurso facilita a **leitura** e **manutenção** de códigos.

Utiliza-se muito para atribuir nomes a erros, a parâmetros de configurações e status.

Enumerates


Enum in C
Declaration
Keyword
enum variable
state=0
state=1
state=6
enum days-of-week { Sun, Mon, Tue, Wed, Thu, Fri, Sat };
Enumerators
(list of constants separated by commas)
Instantiation
enum days-of-week day;
Object of enum days-of-week
Operation
day = wed;
day
3
As state of wed=3

Enumerates - Declaração

Podemos definir os valores que desejarmos também.

```
enum error_e {ERROR_OK=1, ERROR_FAIL=0};
```

E ainda é possível iniciar a contagem a partir de um valor

```
enum error_e {BLE_OK=0x10, BLE_FAIL, BLE_CONN,
              AUDIO_OK=0x20, AUDIO_FAIL};
```

Enumerates - Instâncimamento

Pode-se também instanciar um enumerate como se fosse uma variável, da seguinte forma:

```
enum error_e {ERROR_OK, ERRO_FAIL};

enum error_e retError;
```

Note que é necessário utilizar a keyword **enum**, ocorre porque o compilador não vê o enumerate como uma variável.

Enumerates - Typedef

Se utilizarmos o **typedef**, podemos tratar nosso **enum** como uma variável comum.

```
typedef enum {ERROR_OK, ERROR_FAIL} error_e;

error_e retError;
```

Note que a declaração do nome do **enum** foi para o final!

Structs

Também chamada de *registros*, definem um tipo de dado novo que **agrupa** uma ou mais **variáveis** de outro **tipos**. Podemos utilizar como analogia uma caixa. Imagine um tipo de dado chamado **Pessoa**.

Structs

Isto auxilia na organização do código e em aplicações que utilizam busca de cadastro.
Para declarar uma **struct** utilizamo a seguinte estrutura:

```
struct{
   type1 var1;
   type2 var2;
}str;

struct str m;
```

Structs

Por exemplo a variável pessoa pode ser declarada como:

```
struct{
   char Nome[40];
   sexo_e sexo;
   int idade;
}Pessoa;

struct Pessoa povo[30];
```

Structs - Typedef

Se eu quiser omitir a *keyword* struct, posso declará como um novo formato utilizando o **typedef**.

```
typedef struct{
   char Nome[40];
   sexo_e sexo;
   int idade;
}Pessoa;

Pessoa povo[30];
```

Structs - Acesso

Para acessar uma struct utilizamos o ponto (.).

```
void main(){
   Pessoa eu;

   eu.idade = 25;
   eu.sexo = MASCULINO;
   strcpy(eu.Nome, “Pablo Jean”);
}
```

Structs - Acesso

Podemos passar uma struct como argumento de uma função, sendo o dado propriamente dito, ou ponteiro.

```
void atribui_pessoa(Pessoa  *p, char *n, sexo_e s, int idade){
   p->idade = idade;
   p->sexo = s;
   strcpy(p->Nome, n);
}
```

Structs - Dicas

As structs possuem recursividade, ou seja, posso declarar uma **struct** dentro da outra.

```
struct{
   int i;
   struct{
      float f;
   };
}st1;
```

Structs - Dicas

O compilador pode gerar otimização da **struct**, realizando alinhamento de memória que podem causar *bugs*, a depender das operações que serão realizadas. para isso utilizamos o ***__attribute__((packed))*** na declaração.

```
typedef struct __attribute__((packed)){
   int iValue;
   float fValue;
}data_t;
```

Unions

As **unions** são um tipo de estrutura em que os elementos declarados ocupem um **mesmo espaço de memória**. Permite que uma região memória seja alocada de forma **inteligente** para uso **múltiplo**.

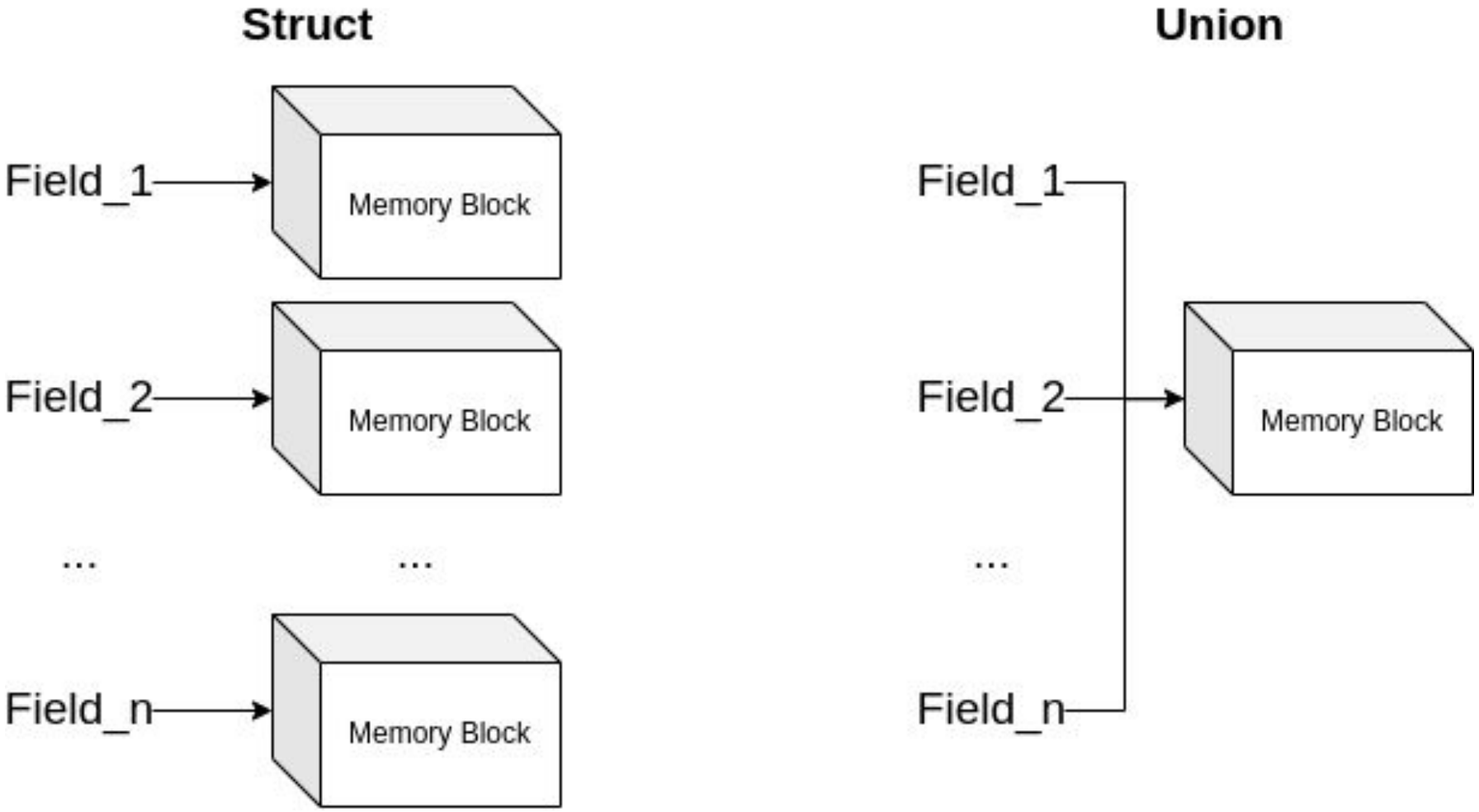
```
typedef union{
   int iV;
   float iF;
}num_u;
```

Unions - Aplicação

Um exemplo de aplicação é converter um valor de 32 bits em um array de 8 bits, de forma rápida e de baixo custo de CPU.

```
typedef union{
   uint8_t raw[4];
   uint32_t val32;
}conv32;
```

Unions - Diferença de Struct



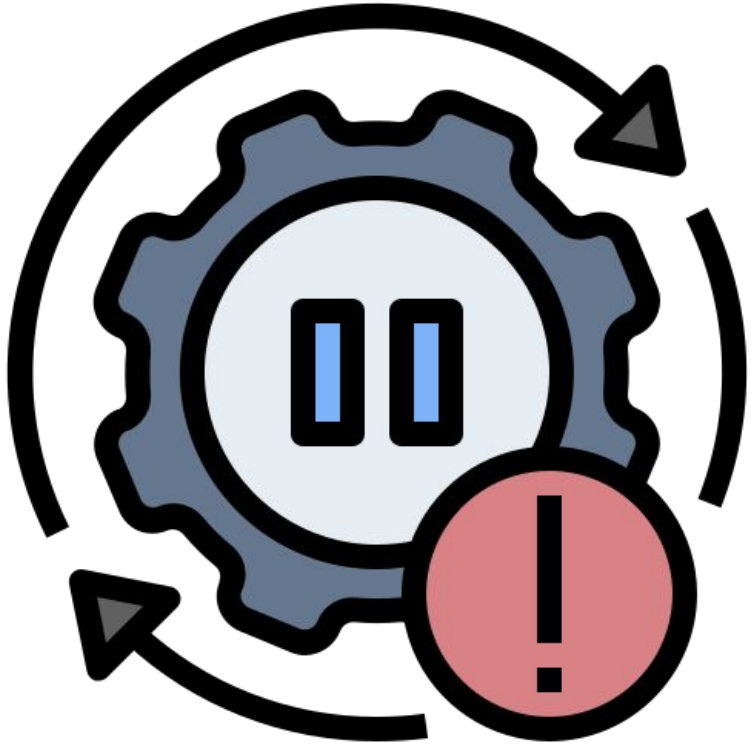
Boas Práticas

Quando declaram variáveis dos tipos que discorremos, utilizar os sufixos abaixo para facilitar a compreensão.

Enumerates : **_e** (**error_e**, **types_e**)
Structs : **_t** (**tas2505_t**, **comm_t**)
Unions : **_t** ou **_u** (**conv32_u**)
Typedefs : sempre usar **_t** quando não for os tipos acima

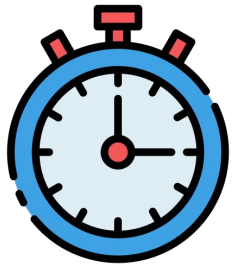
PADO Labs

INTERRUPÇÕES

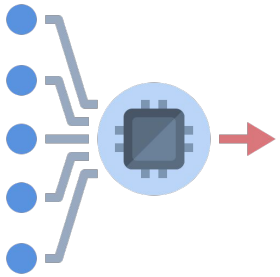


Interrupções

Interrupção é uma funcionalidade no microcontrolador que alterar o fluxo do programa na ocorrência de eventos específicos, que são configurados pelo desenvolvedor.
Podendo ter como fonte:



Timers

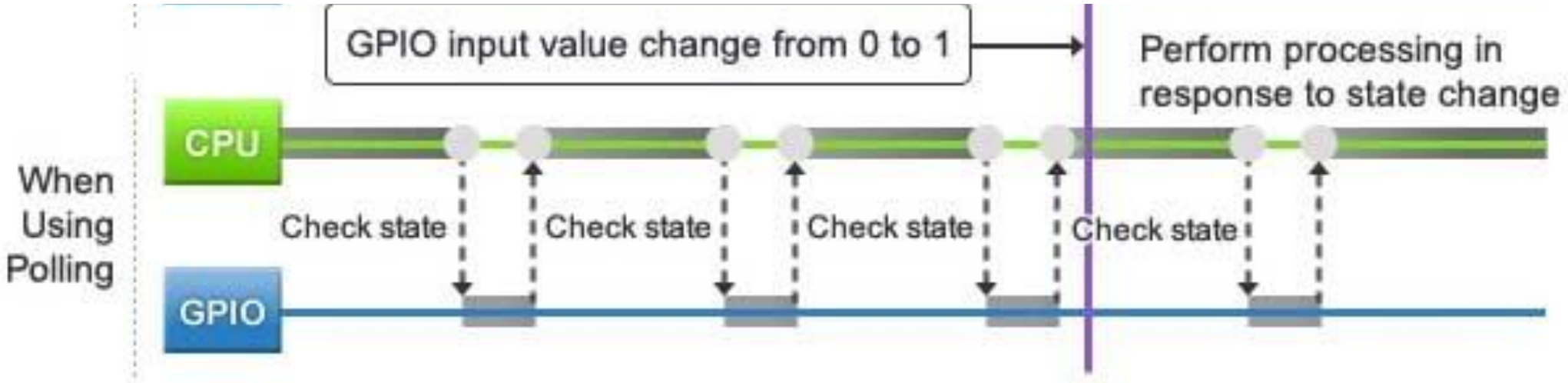


Inputs

Eventos

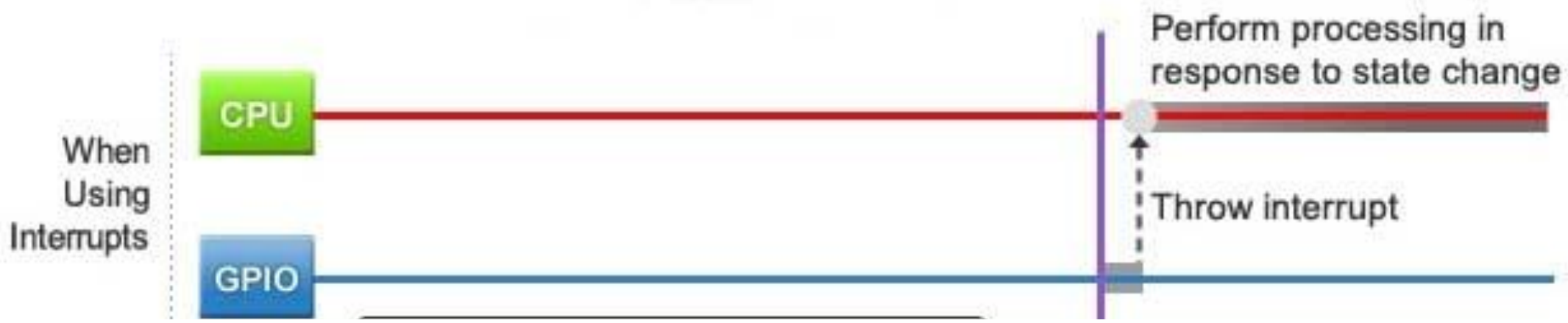
Interrupções - Fluxo

Quando **NÃO** utilizamos interrupção, a **latência** da resposta pode suficientemente alta, prejudicando a resposta do programa.



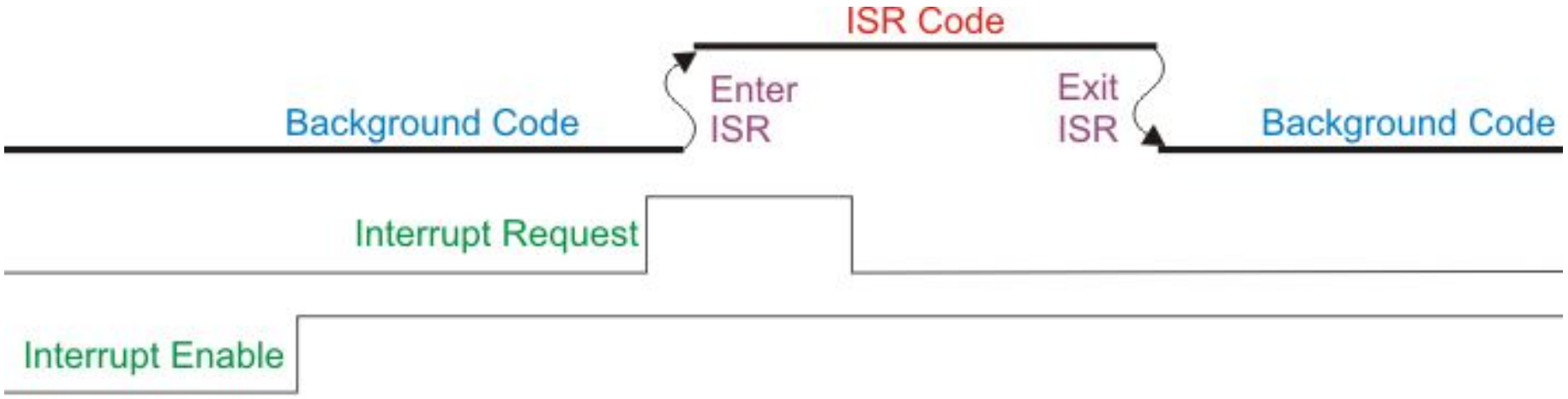
Interrupções - Fluxo

Quando **utilizamos** interrupções, obtemos respostas mais **rápidas**, melhorando a eficiência do programa.



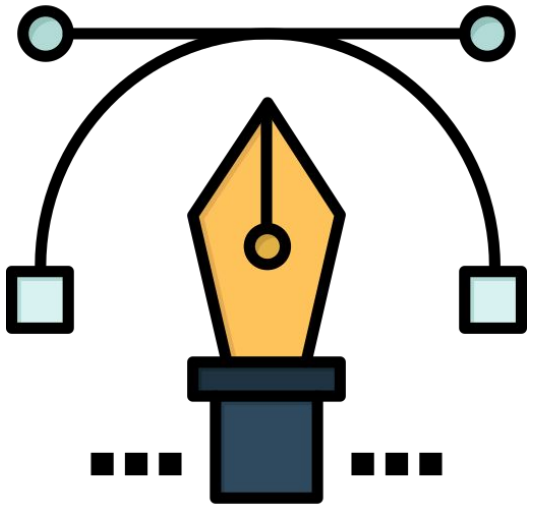
Interrupções - ISR

Interrupt Service Routine trata-se da rotina de processamento da interrupção.
Está função é chamada através de um ***Interrupt Vector***, que é um endereço de memória para onde o programa é desviado e realiza a chamada da **ISR**.



Interrupções - Fluxo

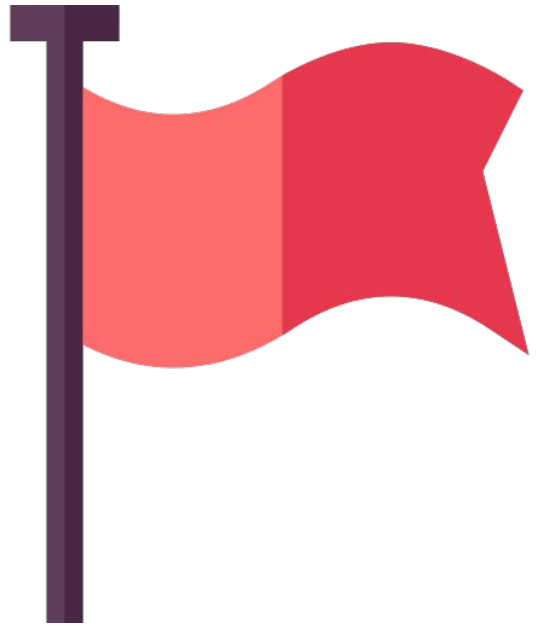
- Interrupção é habilitada
- Ocorre a *Interrupt Request (IR)* : Sinal que informa a CPU
  - Tempo entre ocorrência da IR e entrada da ISR é chamado de *Interrupt Latency*
- O microcontrolador realiza o **push** na *stack*
- Em seguida, desvia o código para a ISR e executa a rotina de interrupção
- Após realizar a ISR, a CPU executa um **pop** na *stack* e retorna ao fluxo anterior ao da interrupção.



Interrupções - *Flag*

Sempre que ocorrer uma interrupção, é gerado um *flag bit*.

Este *flag bit* indica qual a **fonte de interrupção** e dependendo da arquitetura é necessário ser limpo pelo programa!

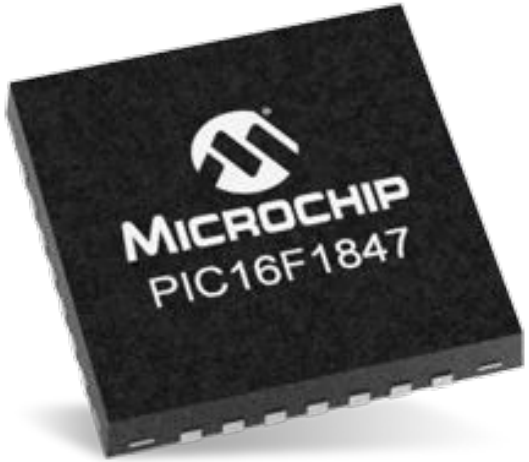


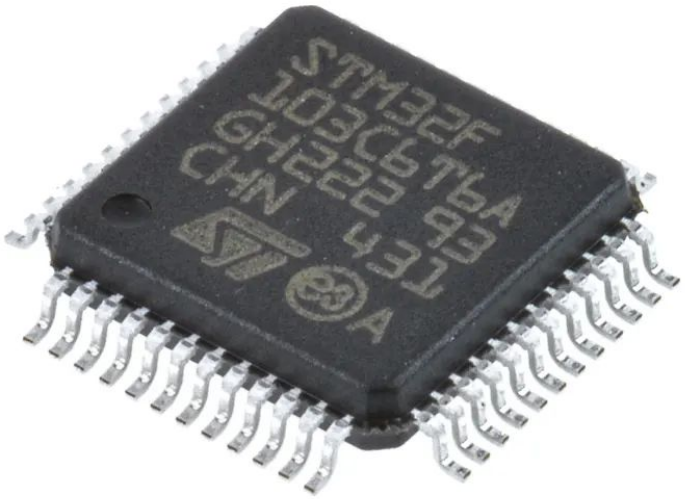
Obs: Em geral, as bibliotecas já fazem esse trabalho de limpar a flag.

Caso a flag não seja zerada, a interrupção entra em *loop* infinito.

Interrupções - Vetores

Cada arquitetura de microcontrolador pode ter vetores diferentes de interrupção. Algumas arquiteturas tem somente um único vetor, outras tem mais de um, filtrando por prioridade, e outras podem ter vários a depender da fonte de interrupção.





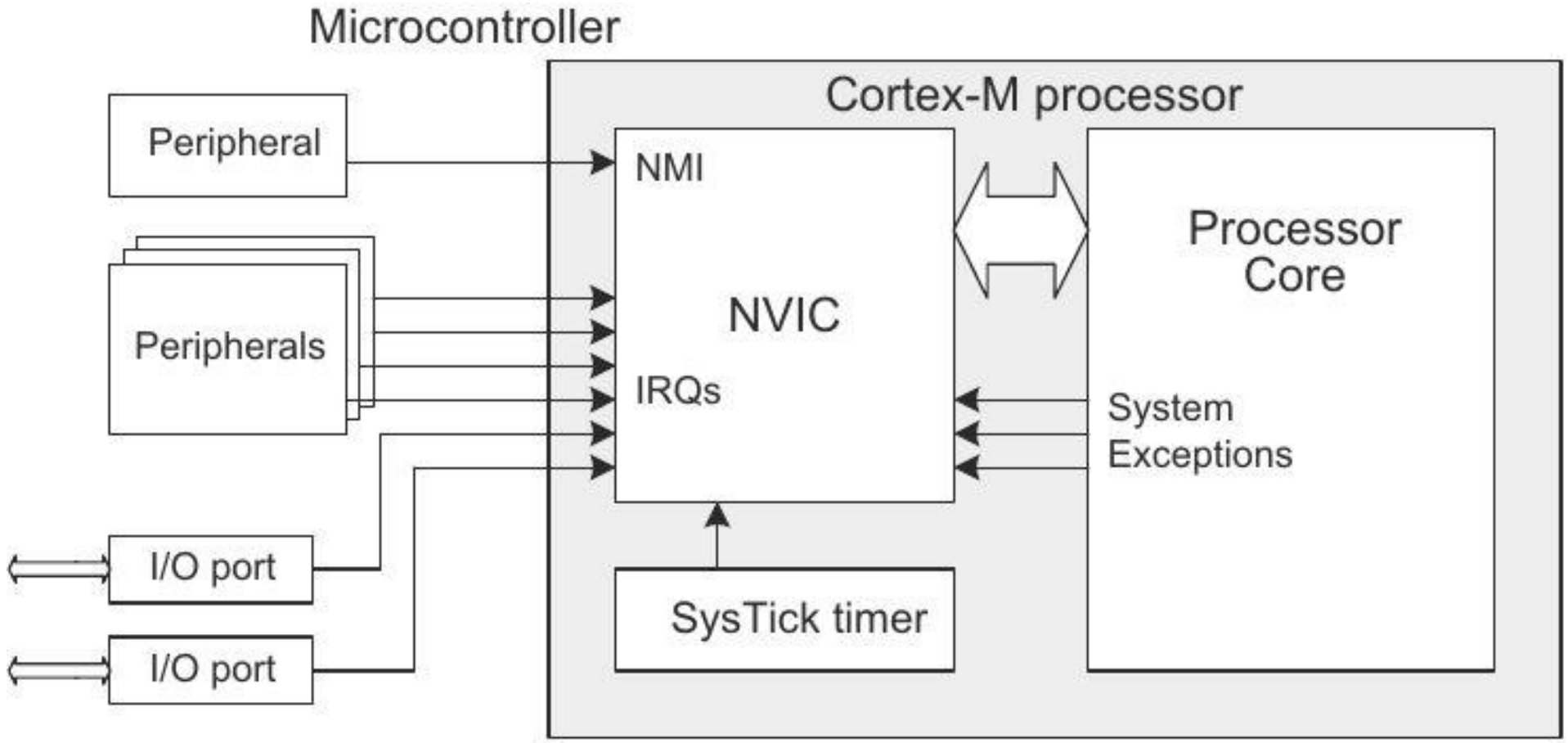
Interrupções - ATENÇÃO

O que deve ser feito numa **ISR**? O **Mínimo**!

Pois o programa processa as interrupções com prioridade acima do *programa normal*.

Interrupções - ANALOGIA

Interrupções - NVIC

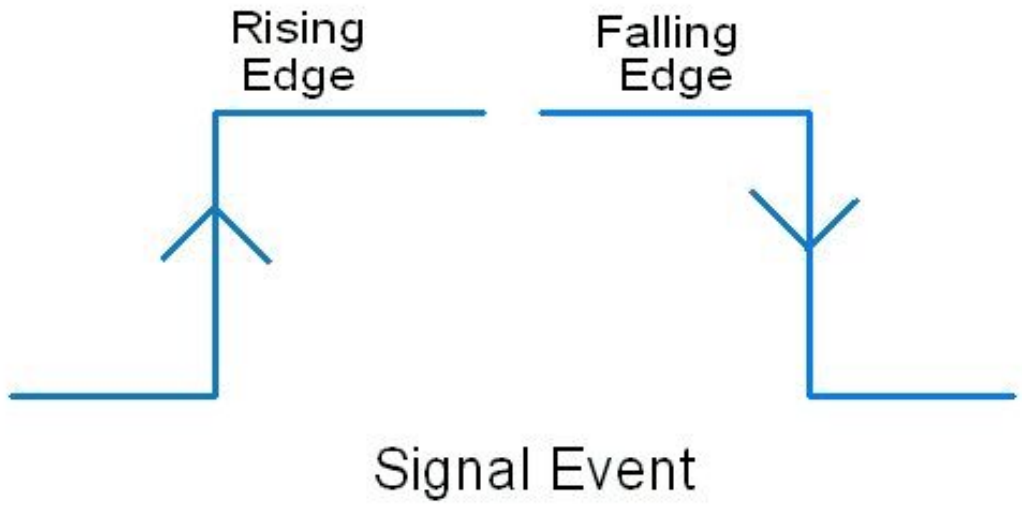


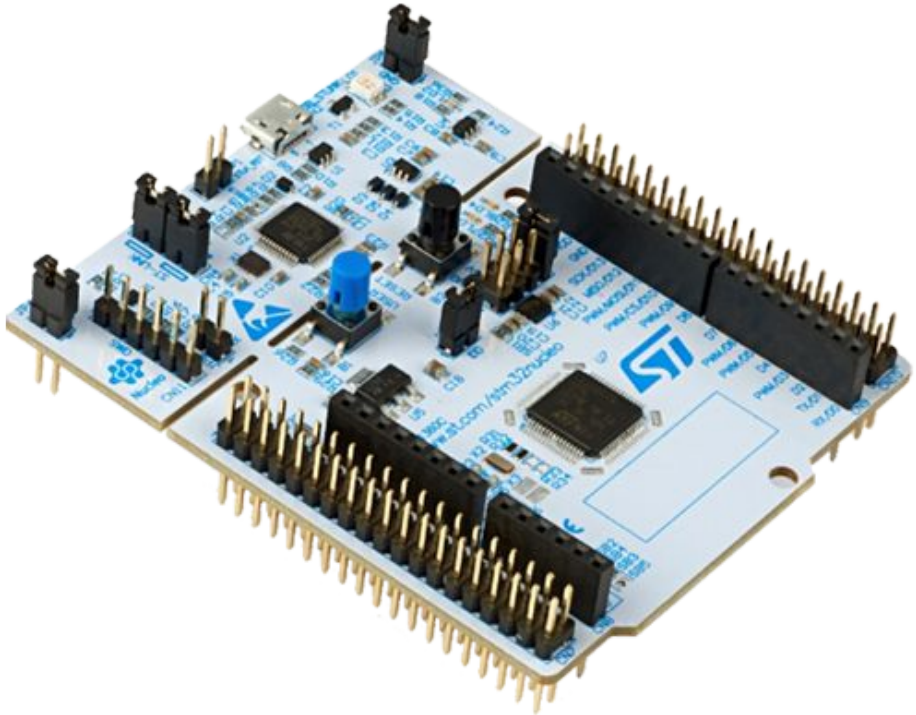
Mão na Massa

PADO Labs

NUCLEO-G0B1RE

Vamos, neste primeiro momento, habilitar uma interrupção na nossa placa de desenvolvimento!
Nosso microcontrolador
trabalha com **pulso** na GPIO.





Funções

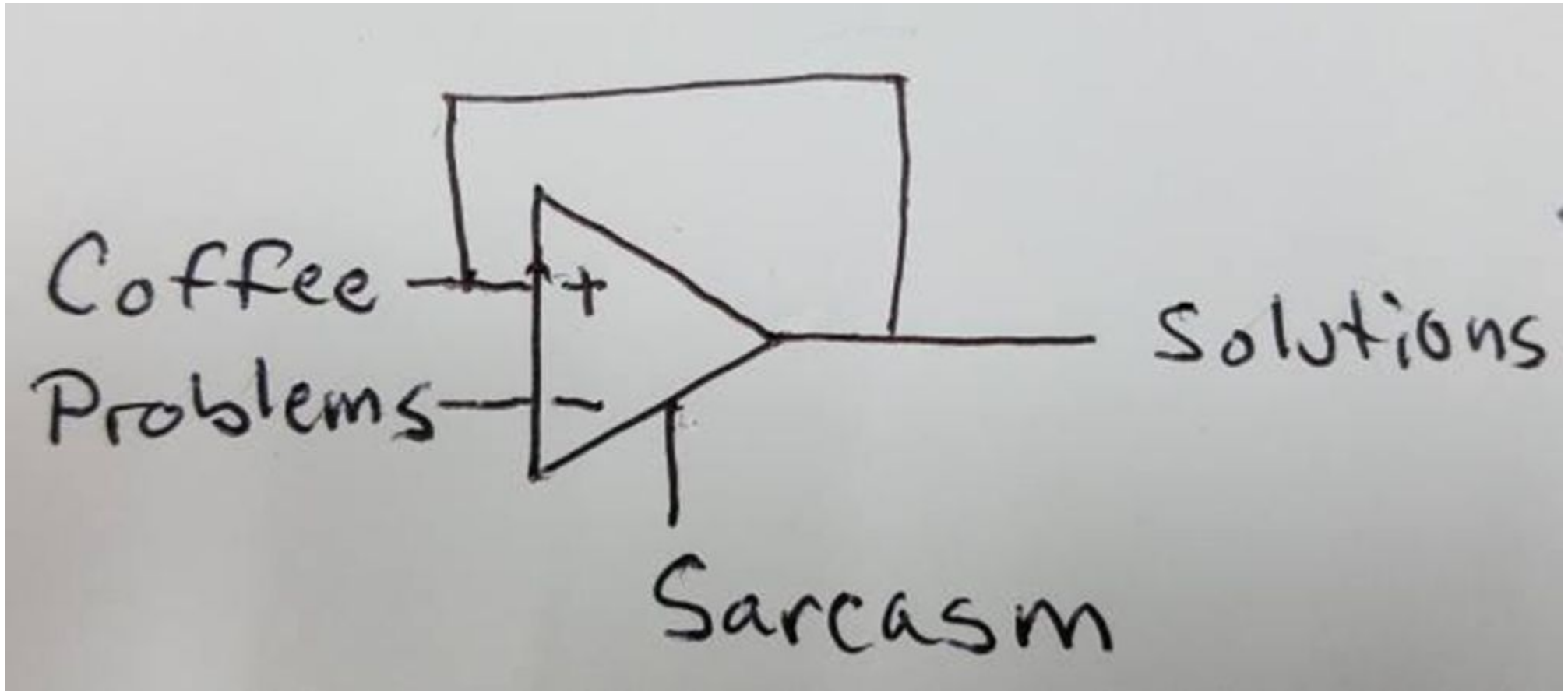
Utilizamos:

```
// Callback de interrupcao que ocorre em rising edge
void HAL_GPIO_EXTI_Rising_Callback(uint16_t GPIO_Pin)
// Callback de interrupcao que ocorre em falling edge
void HAL_GPIO_EXTI_Falling_Callback(uint16_t GPIO_Pin)

//GPIO_Pin = GPIO_PIN0, GPIO_PIN1,.., GPIO_PIN15
```

Dúvidas ??

PADO Labs



Referências

ARAÚJO, ALDROVANDO LUÍS AZEREDO. O Comando typedef. 1999. http://mtm.ufsc.br/~azeredo/cursoC/aulas/cb60.html. Acesso em 7 de Janeiro de 2022.

COLLINS, DANIELLE. What is nested vector interrupt control (NVIC). 2019. https://www.motioncontroltips.com/what-is-nested-vector-interrupt-control-nvic. Acesso em 8 de Janeiro de 2022.

GASPAR, Wagner. Como criar novos tipos de dados em C com TYPEDEF STRUCT. 2021. https://wagnergaspar.com/como-criar-novos-tipos-de-dados-em-c-com-typedef-struct. Acesso em 7 de Janeiro de 2022.

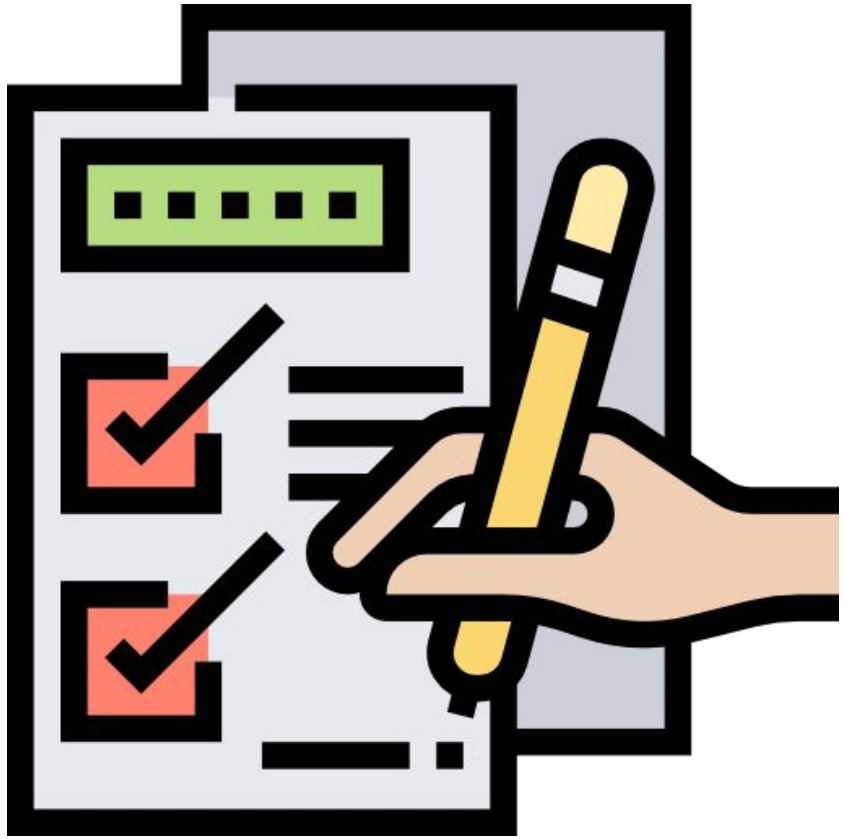
GEEKS, Geeks for. Enumeration (or enum) in C. 2021. https://www.geeksforgeeks.org/enumeration-enum-c/. Acesso em 7 de Janeiro de 2022.

PINHO, Márcio Sarroglia. Uso de Estruturas de Dados Heterogêneas. S.I. https://www.inf.pucrs.br/~pinho/LaproI/Structs/Structs.htm. Acesso em 7 de Janeiro de 2022.

POINT, Tutorials. C - Unions. 2022.

https://www.tutorialspoint.com/cprogramming/c_unions.htm. Acesso em 7 de Janeiro de 2022.

Lista de Exercícios #3



Estruturas em C e Interrupções

PADO

Labs