



PÓS-GRADUAÇÃO SENAI CIMATEC

Sistema FIEB

**SENAI
CIMATEC**

Professor Marco Câmara

Sistemas Multiprocessados em Automação

Curso de Especialização em Automação, Controle e Robótica

Campus Universitário
SENAI CIMATEC
2024-01

Quem é o professor?

Marco Antônio Chaves Câmara

Engenheiro Eletricista - UFBA '87;

Pós-graduado em Educação;

Mestrado (em curso) em Redes de Computadores

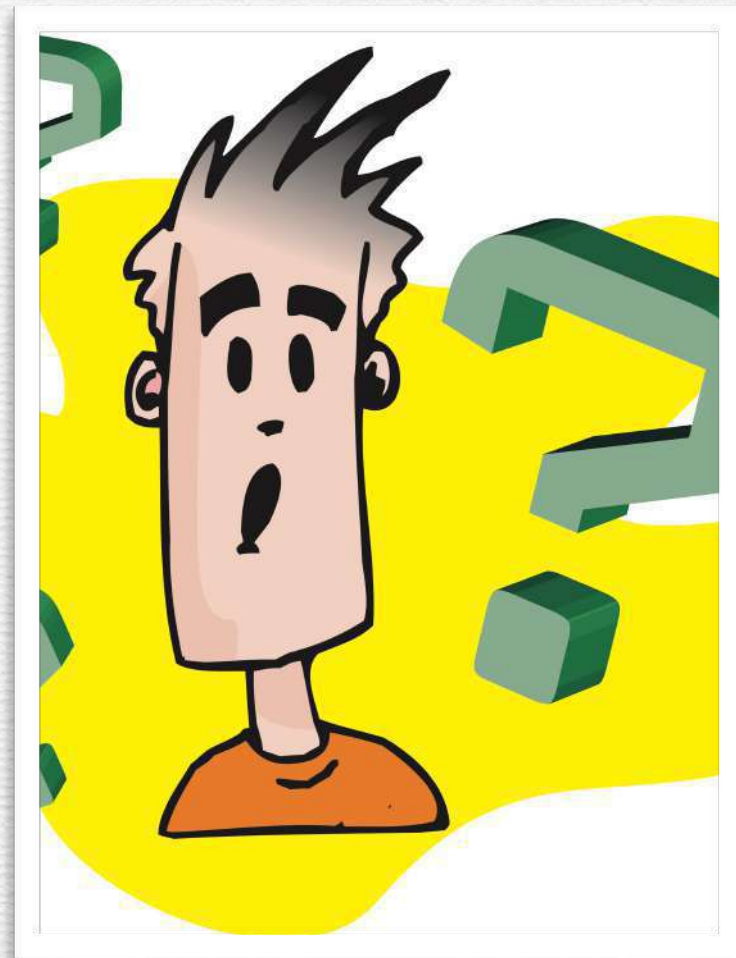
Professor desde 1992:

Graduação: UCSAL, Unifacs, UNEB (convidado).

Pós-Graduação: Area1, Ruy Barbosa e UFBA (convidado),

Extensão: Certificações CISCO, Furukawa, Microsoft, Novell.

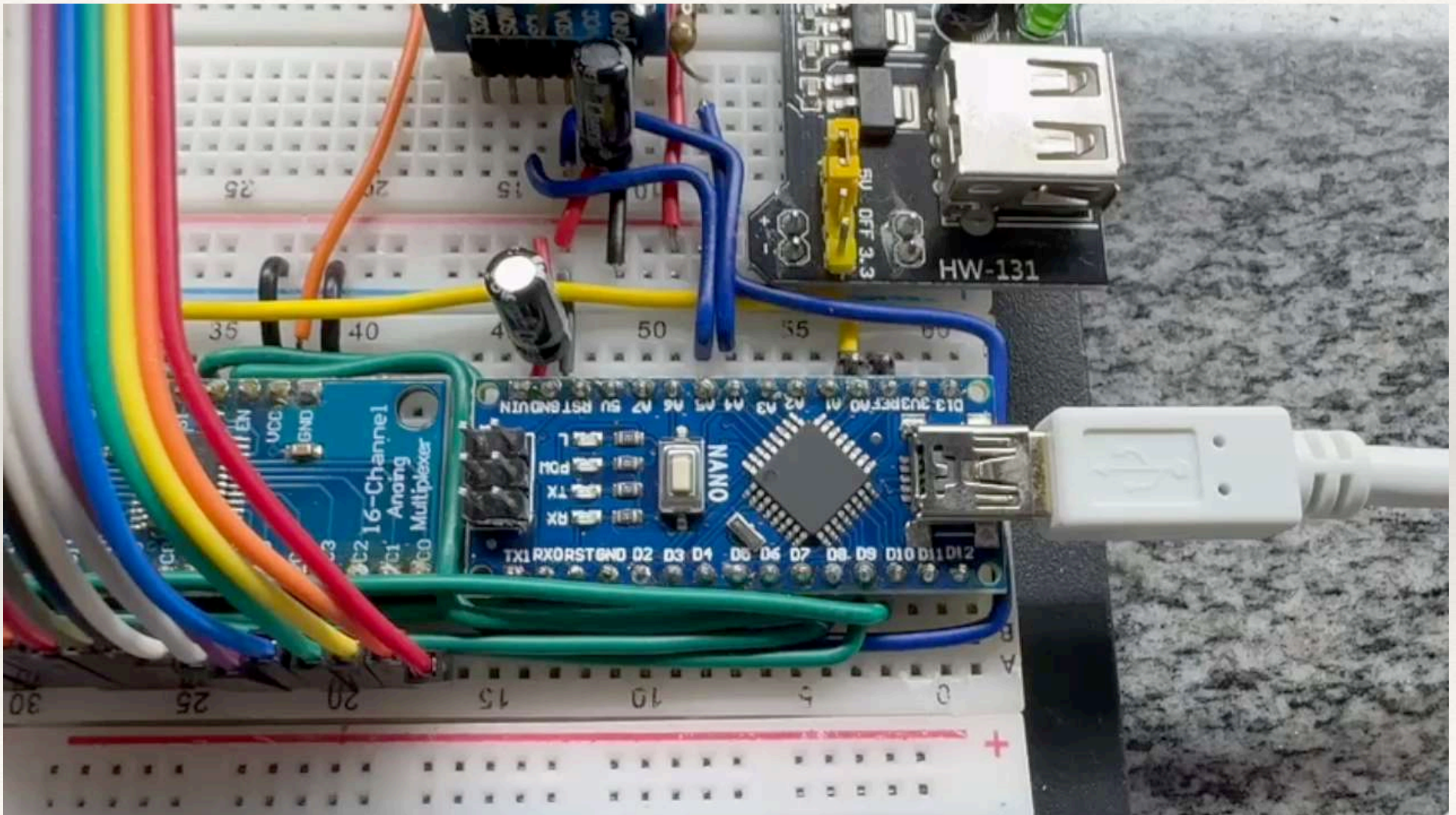
Palestras/Eventos: Eletrônica e Microcontroladores



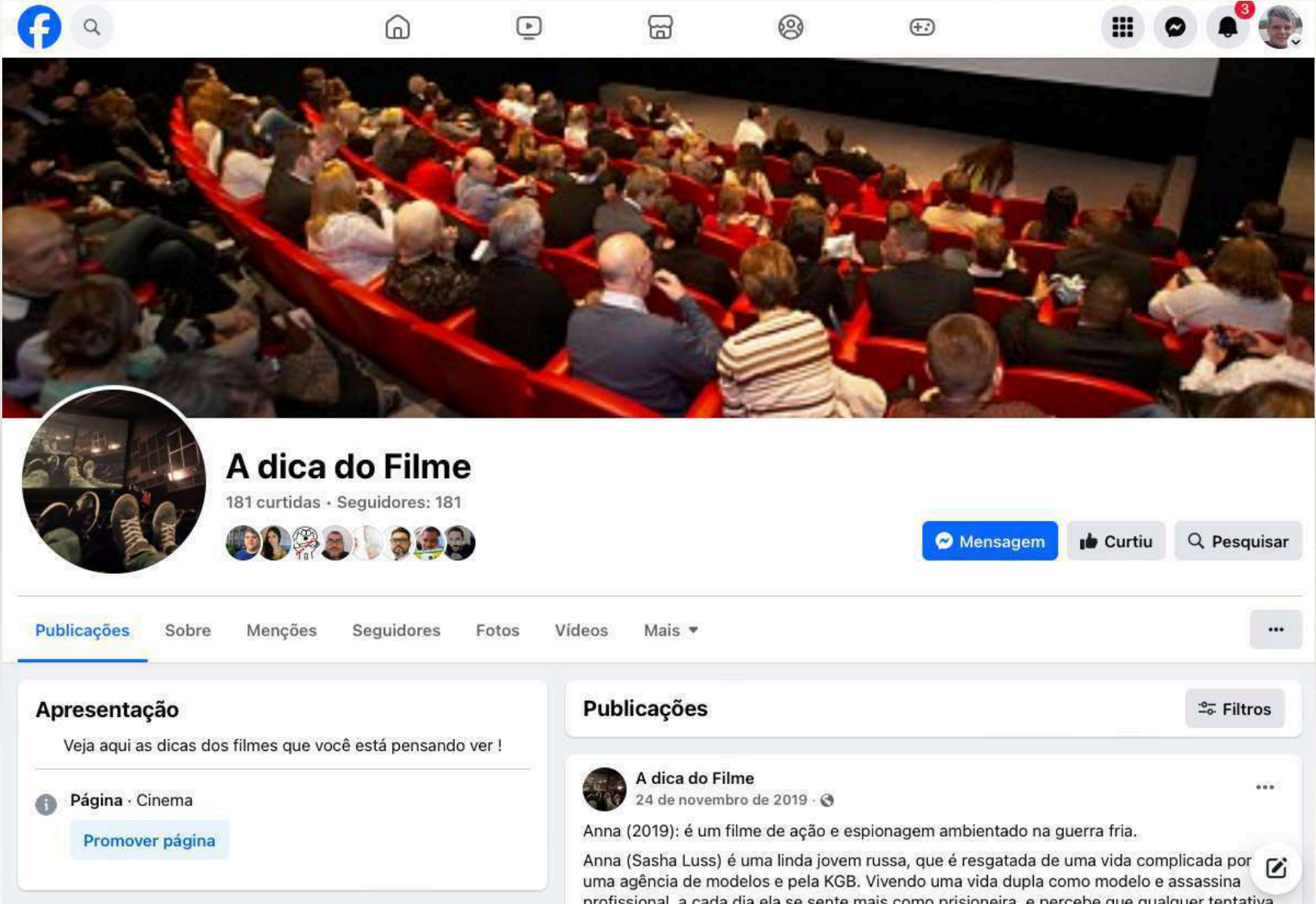
Meus *hobbies* (Eletrônica !)



Meus *hobbies* (Eletrônica !)



Meus *hobbies* (Cinema !)



The image shows a Facebook profile page for a page named "A dica do Filme". The profile picture is a large photograph of a cinema audience seated in red chairs, viewed from behind. The page header includes the Facebook logo, a search icon, and navigation icons for home, video, marketplace, profile, and groups. The profile name "A dica do Filme" is displayed with 181 likes and 181 followers. Below the name are several profile pictures of users who liked the page. To the right of the name are buttons for "Mensagem", "Curtiu", and "Pesquisar". The page tabs include "Publicações", "Sobre", "Menções", "Seguidores", "Fotos", "Vídeos", and "Mais". The "Publicações" tab is selected, showing a post from November 24, 2019, about the movie "Anna (2019)".

A dica do Filme
181 curtidas • Seguidores: 181

Mensagem Curtiu Pesquisar

Publicações Sobre Menções Seguidores Fotos Vídeos Mais

Apresentação
Veja aqui as dicas dos filmes que você está pensando ver !

Página · Cinema
Promover página

Publicações Filtros

A dica do Filme
24 de novembro de 2019

Anna (2019): é um filme de ação e espionagem ambientado na guerra fria.

Anna (Sasha Luss) é uma linda jovem russa, que é resgatada de uma vida complicada por uma agência de modelos e pela KGB. Vivendo uma vida dupla como modelo e assassina profissional, a cada dia ela se sente mais como prisioneira, e percebe que qualquer tentativa

Meus *hobbies* (Fotografia !)



Para me achar ...

Meu *site*:

www.logicengenharia.com.br/mcamara

... ou procure no Google ! (sou ± popular 😊)

Meu *e-mail*:

marco.camara@ba.docente.senai.br

Canal no Youtube:

Busque **@ProfMarcoCamara**

Celular / WhatsApp

71-9 9197-8976 (Vivo)

(*) Também estou nas redes sociais
(Facebook, X, LinkedIn...), mas não as utilizo
para assuntos acadêmicos!



Agora, vocês !

Área de Concentração

Eletrônica?

Sistemas Digitais, Microcontroladores?

Sistemas Embarcados, IoT?

Programação?

Fale um pouco de você ...



Método de Avaliação

Avaliações Diárias (30%)

Questões bem básicas para avaliar acompanhamento;

Devem ser respondidas no final de cada encontro;

Disponíveis por alguns minutos no Google Forms.

Avaliações das Práticas (20%)

Devem ser respondidas ao final de cada prática;

Simulações e Atividades extra-classe (10%)

Devem ser respondidas após realização das atividades extra-classe.

Avaliação Final (40%)

Será realizada no último encontro, abordando todo o assunto visto na disciplina.

Atividades Extras (?)

Sempre contam pontuação EXTRA, e podem ser resultado de participação durante a aula, resposta a desafios, apoio na localização de erros no material etc.



Metodologia

Aulas expositivas

Apresentação de slides, vídeos;

Referências externas (livros, sites, canais e blogs);

Atividades práticas

Laboratório;

Atividades simuladas

Uso de simulador de circuitos e *sketches*;

Tipicamente será utilizado como atividades extra-classe.



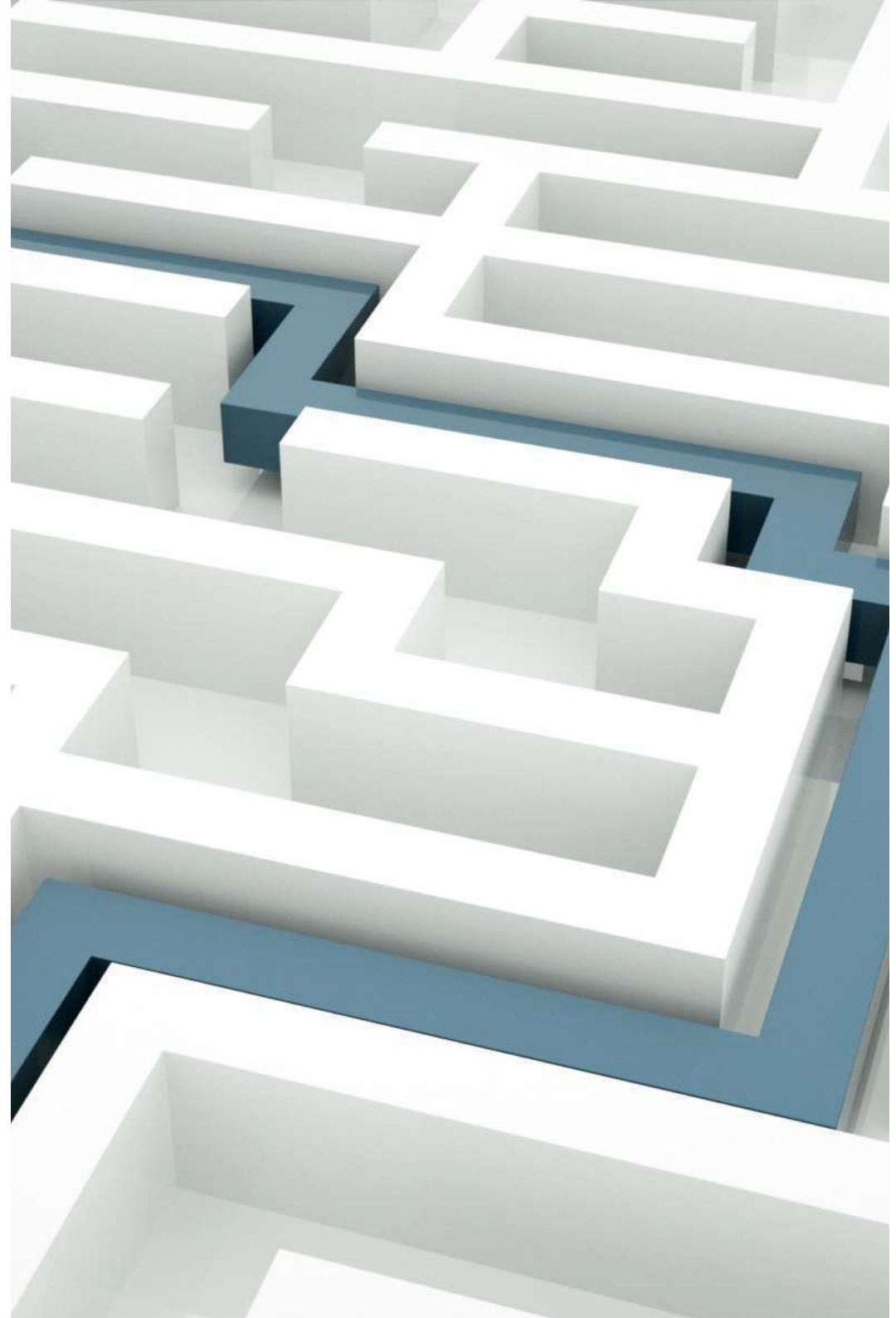
Algumas regrinhas ...

Teremos uma Avaliação Diária ao final de todas os encontros.

Ela é obrigatória e funciona como **chamada**. Se você não responder, será registrada a sua falta.

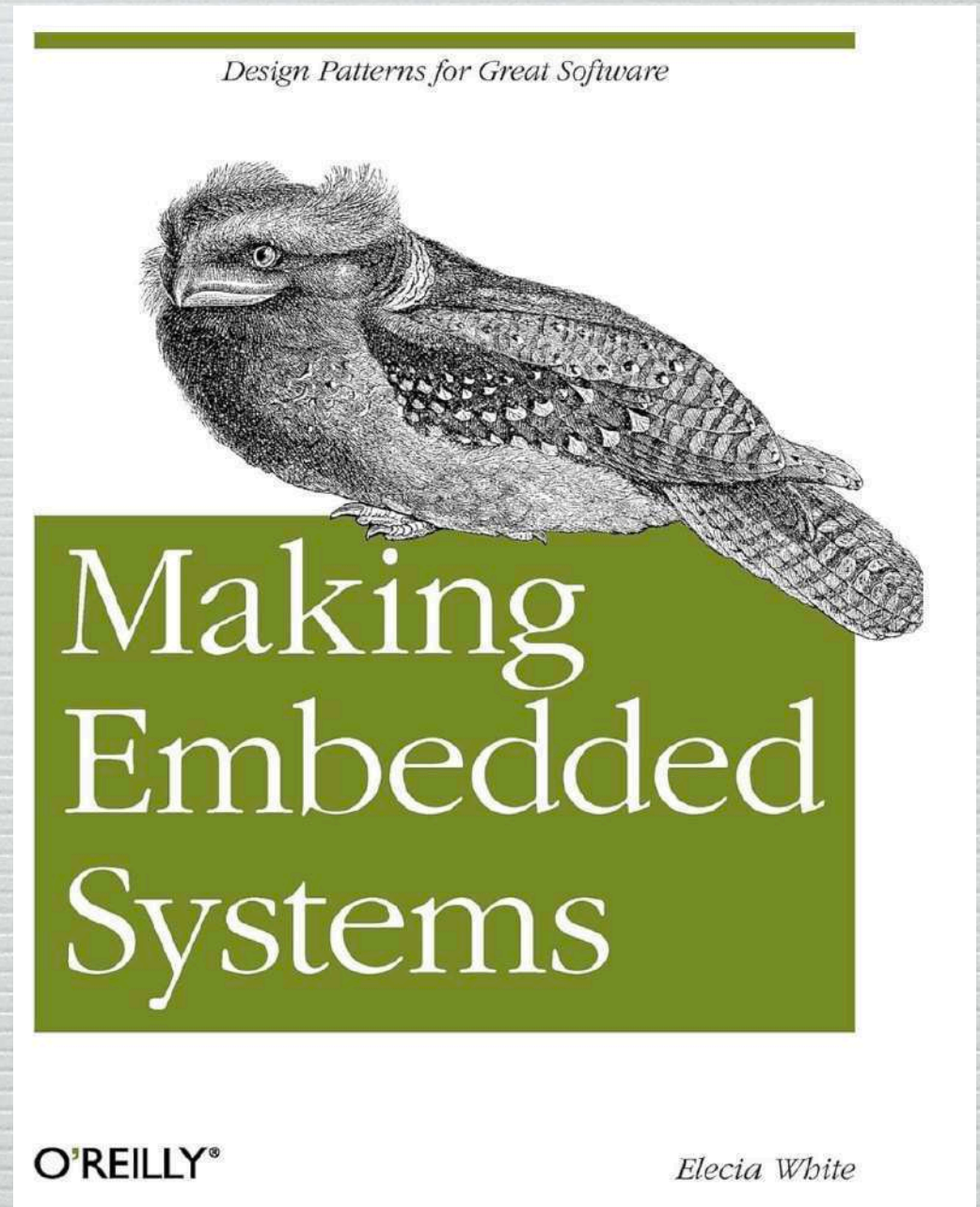
Negociaremos intervalos e horário de término.

Nas práticas, é FUNDAMENTAL seguir o roteiro indicado pelo professor, seguindo passo a passo no mesmo ritmo de toda a turma.



Bibliografia

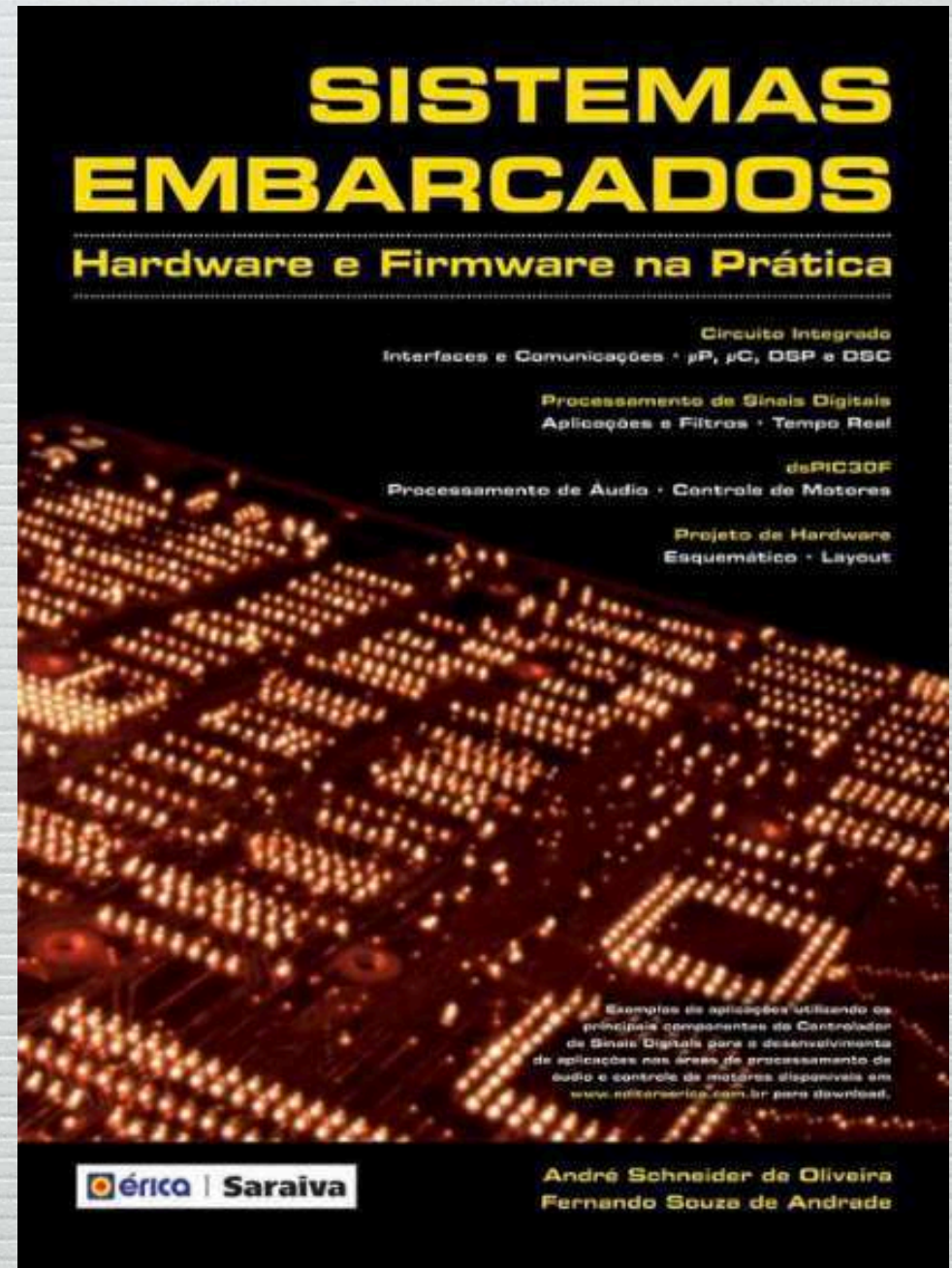
- ✓ Embora não tenha versão disponível em português, foi a melhor referência encontrada;
- ✓ Trata de aspectos de *hardware* e *software*.



Bibliografia

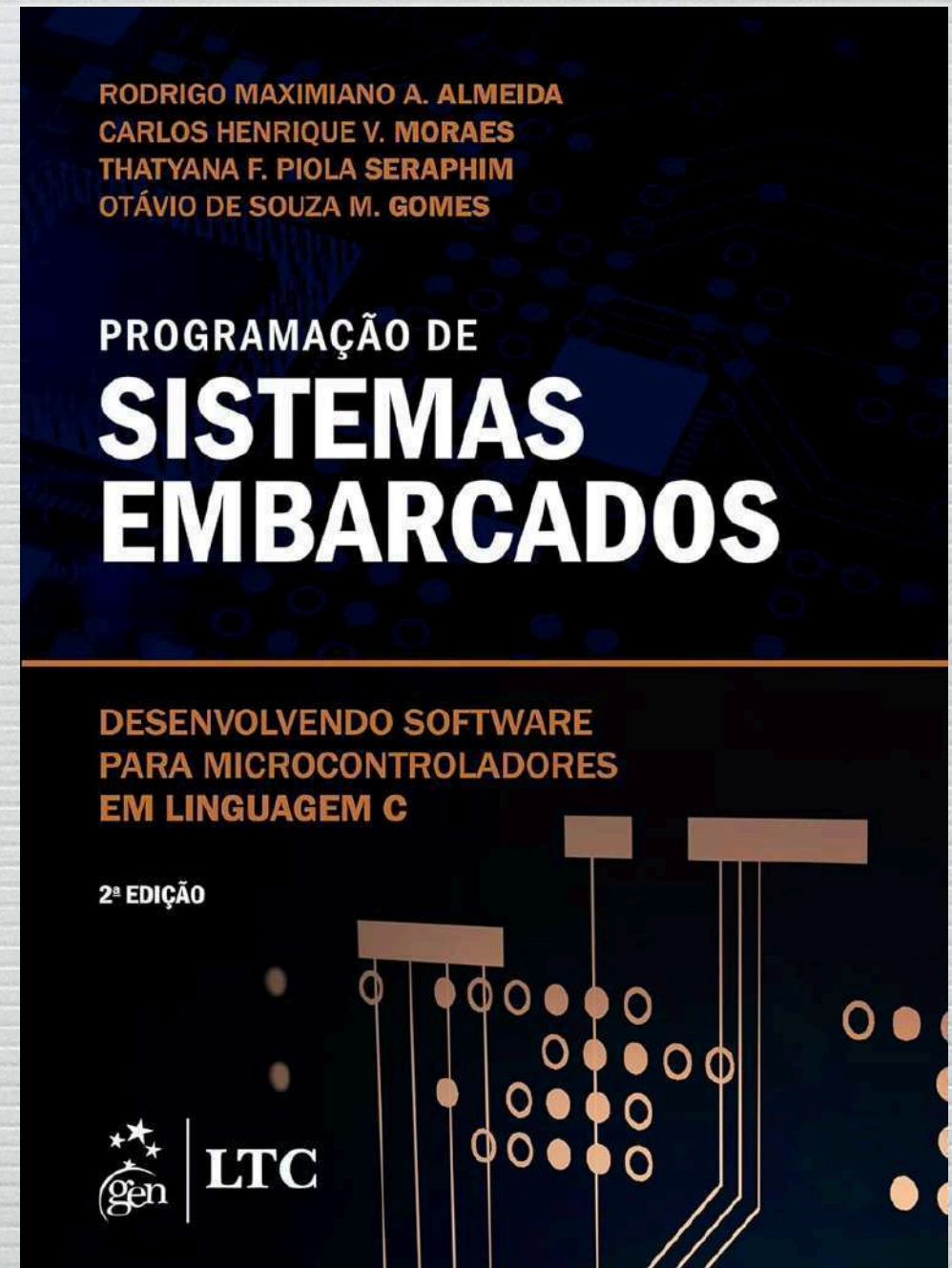
✓ Um livro bem completo e interessante;

✓ Adota uma linguagem simples e direta.



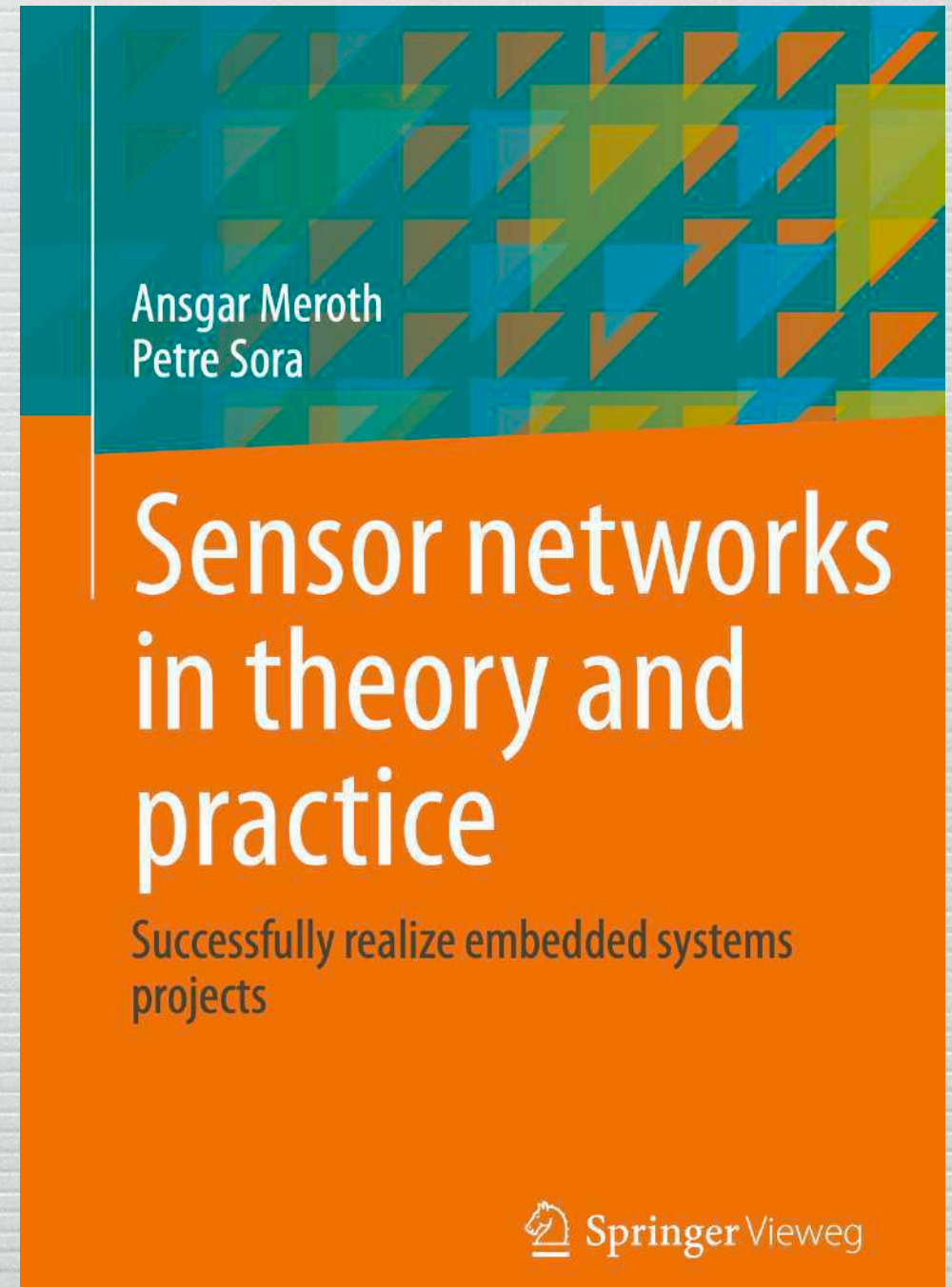
Bibliografia

✓ É uma boa referência a partir do capítulo 10 (os primeiros capítulos compõem uma revisão de temas básicos).



Bibliografia

✓ Referência muito interessante, especialmente para o tratamento dos componentes de entrada e saída.



Bibliografia

- ✓ Referência bem completa, embora mais superficial;
- ✓ Pode ser interessante para quem precisa rever os conceitos básicos do Arduíno, embora o primeiro livro da série seja ainda mais indicado.



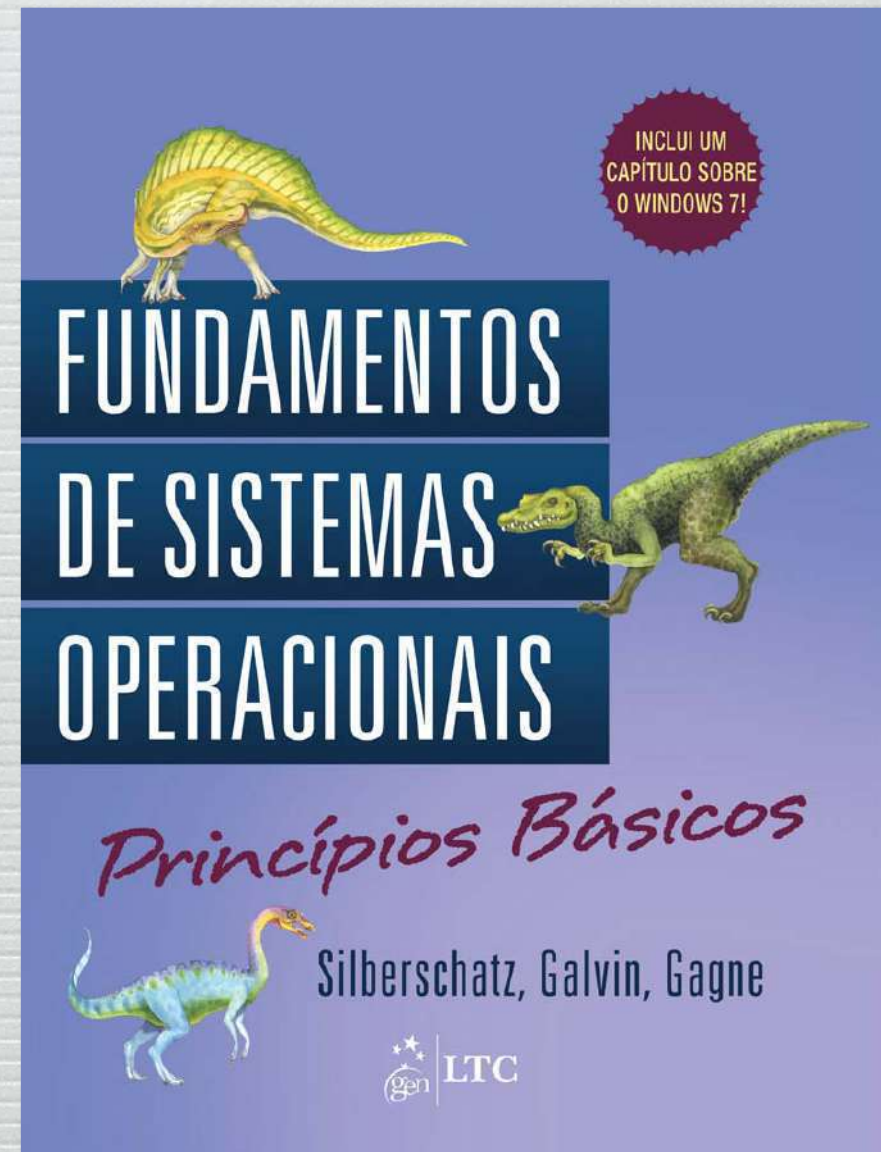
Bibliografia

- ✓ É um excelente livro, com conteúdo além do básico; pode ajudar no tema;
- ✓ Conteúdo bem organizado e didático.



Bibliografia

- ✓ Recomendável para rever alguns pontos básicos, mas ...
- ✓ Vai além do básico, podendo ajudar em alguns pontos.



Bibliografia



MARCO ANTÔNIO C. CÂMARA

[Home](#) [Tutoriais](#) [Palestras](#) [Mini-Cursos](#) [Outros Eventos](#) [Sites Legais](#) [Fale Comigo](#) [Fotos](#) [Cadastre-se](#)

Cursos / Disciplinas

[Introdução à Computação](#)
[Intr. à Seg. da Informação](#)
[Projetos de Redes](#)
[Redes de Computadores](#)
[Tecnologia e Sociedade](#)
[Sistemas Operacionais](#)
[Sistemas Distribuídos](#)
[Redes de Alta Velocidade Area1](#)
[Projeto de Infraestrutura para Redes Area1](#)
[Tópicos em informática \(Redes\)](#)
[Lógica de Programação](#)
[Arquitetura de Computadores](#)
[Processos de Negócio](#)
[Trabalho dos alunos](#)

Arquitetura e Organização de Computadores

Seguem informações e ferramentas para a disciplina Arquitetura e Organização de Computadores. Não é necessário instalar nenhuma aplicação no seu dispositivo, exceto no caso dos arquivos de texto, que estão em formato PDF, o que pode exigir o Acrobat Reader ([Baixe Aqui](#)).

Informações da Disciplina

[Programa](#) - Versão 2014-2 revisada em 2018-2 (29 KB) - [Programa completo da disciplina \(em revisão\)](#).

Transparencias de Apresentação da Disciplina - [Turma 01 \(1,8 MB\)](#); [Turma 02 \(1,8 MB\)](#)

Simuladores

[Circuitos Digitais](#) - Emula circuitos analógicos e digitais em um navegador (*on-line*). O simulador tem versão em Português. Recomendo usar o Google Chrome.

["Kit" de Construção de Circuitos](#) - Emula circuitos bem simples AC/DC. Exige JVM disponível na máquina

Planejamento

Carga-Horária

36 horas-aula de 50 min.

Encontros

Quintas e Sextas-feiras das 18:30 às 22:30 (200 min cada)

13,14,27 e 28/Junho - CIM1-2-LAB6 - ROBÓTICA

11 e 12/Julho - CIM2-1-LAB6 - LAB. ELETRÔNICA

Sábados das 09:00 às 13:00 (200 min)

15 e 29/Junho - CIM1-2-LAB6 - ROBÓTICA

13/Julho - CIM2-1-LAB6 - LAB. ELETRÔNICA

Intervalo quinzenal entre cada final de semana

Intervalo de 10 min entre cada hora-aula (pode-se, em comum acordo, definir a exclusão do intervalo para terminar a aula mais cedo);

Planejamento

Duração de cada encontro

200 min (3 horas e 20 minutos), com 4 horas-aula de 50 min.

Total de Encontros

9 (36 / 4)

Período de Aulas: 13/06 até 13/07

Objetivos da disciplina

Compreender a arquitetura de sistemas microcontrolados, considerando a aplicação destes em Sistemas Embarcados de automação, visando reduzir dimensões, consumo e custo;

Compreender os componentes complementares para leitura de variáveis e eventos externos, e escrita/controlado de atuadores e demais elementos de saída visando o controle do sistema de automação;

Compreender os conceitos de programação destes sistemas, incluindo as técnicas para aumento de performance, redução dos recursos exigidos, e estabilidade operacional.

Pré-requisitos desejáveis

Eletrônica Básica;

Eletrônica e Circuitos Digitais;

Conceitos Básicos de Arquitetura de Computadores;

Programação de Computadores (C++, C ou Java);

Inglês Técnico.

Aula 01

Motivação e história

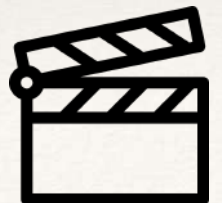
Sempre houve demanda pela automação de tarefas repetitivas, por diversos motivos. Entre eles:

Aumento da performance

Redução de Custos

Padronização

Segurança



Motivação e história

Sempre houve demanda pela automação de tarefas repetitivas, por diversos motivos. Entre eles:

Aumento da performance

Redução de Custos

Padronização

Segurança

A revolução industrial praticamente foi definida por esta demanda

Autômatos complexos serviram de inspiração para as máquinas de tear

Os autômatos durante algum tempo eram dispositivos mecânicos: roda d'água, relógio, semáforos, e até robôs !



Motivação e história

Sempre houve demanda pela automação de tarefas repetitivas, por diversos motivos. Entre eles:

Aumento da performance

Redução de Custos

Padronização

Segurança

A revolução industrial praticamente foi definida por esta demanda

Autômatos complexos serviram de inspiração para as máquinas de tear

Os autômatos durante algum tempo eram dispositivos mecânicos: roda d'água, relógio, semáforos, e até robôs !

Dispositivos mecânicos acabaram sendo substituídos por elétricos, eletrônicos, digitais, e finalmente microprocessados

Redução de custos pela escala de produção;

Maior simplicidade de projeto, fabricação e manutenção.

Motivação e história

Sempre houve demanda pela automação de tarefas repetitivas, por diversos motivos. Entre eles:

Aumento da performance

Padronização

A revolução industrial

Autômatos complexos

Os autômatos do relógio, semáforos

Dispositivos mecânicos

eletrônicos, digitais, e finalmente **microprocessados**

Redução de custos pela escala de produção;

Maior simplicidade de projeto, fabricação e manutenção.

Alguns autores adotam uma classificação dos sistemas embarcados em função da arquitetura dos processadores (pequena ou média escala, ou “complexa”). Outros usam a “geração” dos dispositivos (1ª a 4ª gerações). Como este tipo de classificação inevitavelmente ficará defasada à medida em que evoluem os dispositivos, não adotaremos esta estratégia.

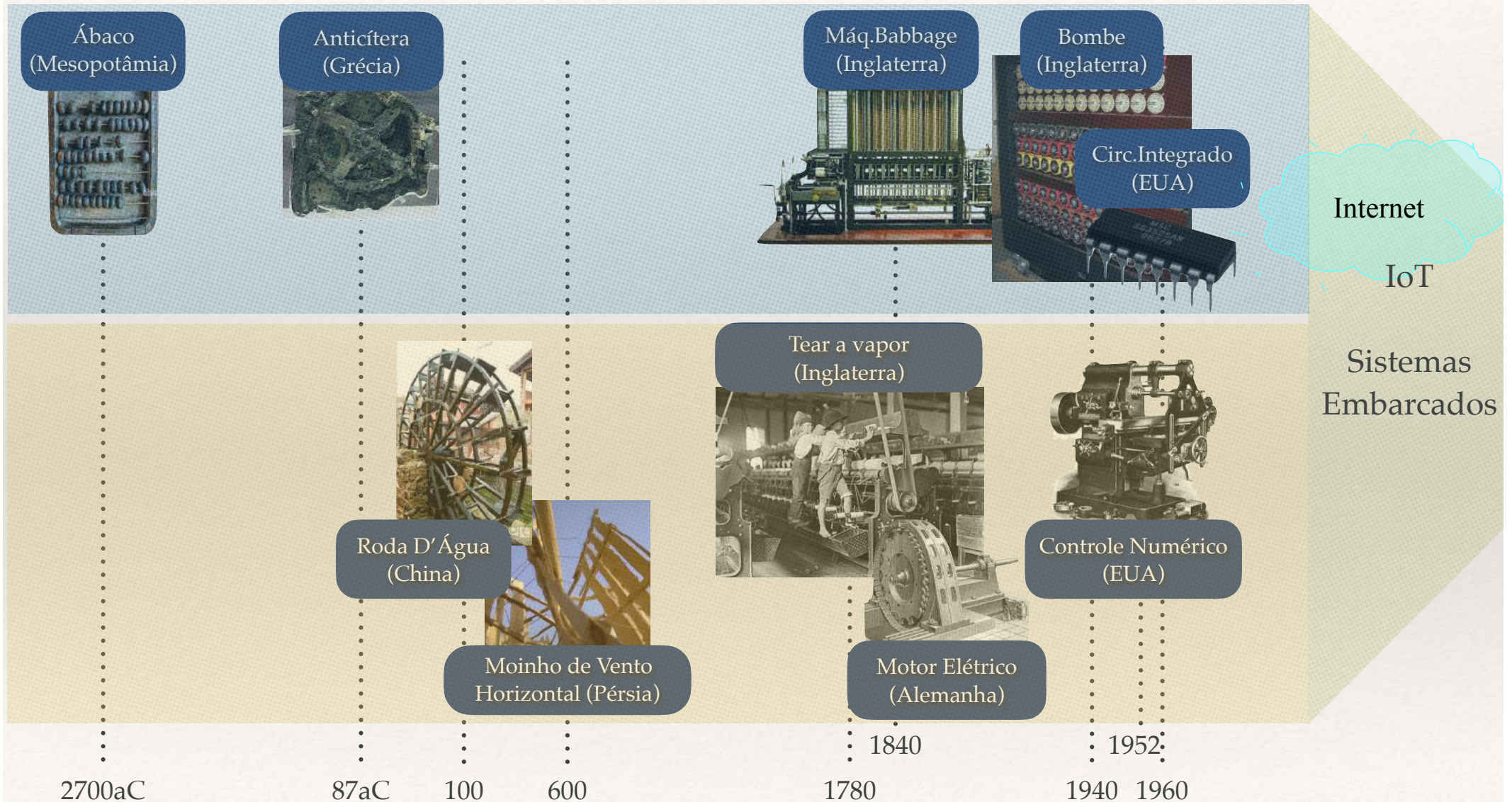
ta demanda

de tear

: roda d'água,

por elétricos,

Um pouco de história



Eletrônica na Automação

Os benefícios da eletricidade foram rapidamente aproveitados

- Motores, válvulas, solenóides;
- Botões, Relês, Chaves Magnéticas;
- Circuitos elétricos.

Circuitos eletrônicos analógicos foram o próximo passo

- Automação “inteligente”;
- Maior confiabilidade, menor ocupação de espaço etc.

Chegada da eletrônica digital e a Lógica Fixa

- Portas lógicas permitiam lógicas complexas, e até programação sequencial;
- Comportamento mais estável, obedecendo à lógica digital e binária;

Lógica Fixa

Funcionalidade Digital

É virtualmente possível implementar qualquer lógica específica através de projeto combinacional/sequencial;

Pode-se obter níveis elevados de performance.

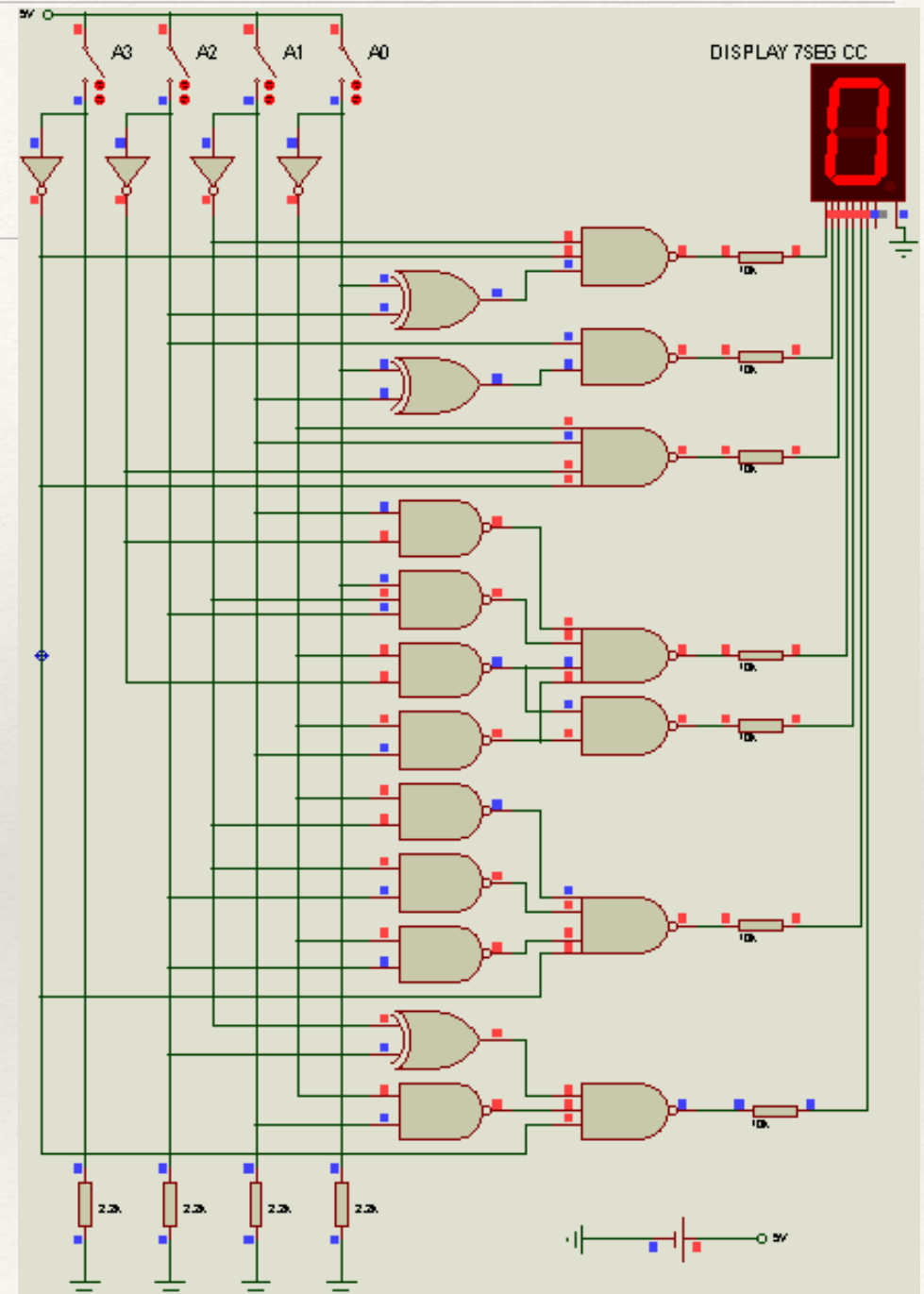
Pontos negativos

Baixa flexibilidade e portabilidade;

Complexidade gera custos elevados;

Projeto, Implementação e Manutenção

Tipicamente tem dimensões maiores.



Lógica Fixa (implementação)

Discreta

Operações lógicas básicas (E, OU, XOR etc);

Componentes eletrônicos analógicos complementares.

Integrada

Funcionalidade implementada totalmente, ou em grande parte, em circuito integrado específico;

Pode utilizar inclusive circuitos digitais programáveis, como FPGA (*Field Programmable Gate Array*).

Sistemas Microprocessados

Flexibilidade

Lógica implementada por SW;

Um único microcontrolador admite infinitos projetos;

Recursos embutidos permitem novas funcionalidades talvez inviáveis em sistemas de Lógica Fixa;

Permitem atualizações pós-venda.

Baixo custo

Não é raro que microcontroladores substituam dezenas de componentes discretos;

Consumo energético e espaço ocupado são reduzidos.

Sistemas Microprocessados $\Rightarrow$ SISTEMAS *ou "embutidos"* ~~EMBARCADOS~~.

A História mais recente ...

Microprocessadores

1968: surge a Intel.

4 bits: Intel 4004, 4040

8 bits: Intel 8008, 8080, 8085

16 bits: Intel 8086, 8088, 80186 & 80188, 80286

32 bits: 80386, 80486, Intel Pentium, Pentium Pro, II, II xeon, III, IV, Dual Core

64 bits: Intel Core 2, i3, i5, i7, i9

Microcontroladores

1971: TMS1000, da Texas

1976: Intel 8048

1980: Intel 8051

Anos 90: junto com a EPROM, surgiram diversos fornecedores

Definição

Um **sistema embarcado** é um sistema computadorizado construído para uma aplicação específica. Por isto mesmo, normalmente não suporta itens que não estejam relacionados à execução da aplicação. Seu *hardware* e *software* têm limitações.

O *hardware* normalmente é mais lento para reduzir o consumo de energia, e oferece apenas recursos mínimos essenciais para suporte ao seu conjunto específico de periféricos.

O *software* tipicamente precisa ter comportamento determinístico (sempre oferece o mesmo tempo de execução), e, em certos casos, em tempo real (reagindo sempre de forma imediata a eventos). Em alguns sistemas, o *software* precisa ser tolerante a falhas, com degradação controlada (considere por exemplo um satélite ou um marcador preso ao corpo de uma baleia). Em outros, o *software* precisa comunicar o erro imediatamente após o primeiro sinal de falha (considere um monitor cardíaco).

[[Making Embedded Systems - White, Elecia](#)]

Definição

Um **sistema embarcado** é um sistema eletrônico microprocessado que, após ser programado, possui uma função específica que geralmente não pode ser alterada. Uma impressora, por exemplo, mesmo possuindo um processador que poderia ser utilizado para qualquer tipo de atividade, tem sua funcionalidade restrita apenas à impressão de páginas. Um computador de propósito geral, no entanto, pode ser utilizado num instante como um ambiente de entretenimento, em outro como estação de trabalho, ou até mesmo um telefone.

Definição

Um **sistema embarcado** é um sistema eletrônico microprocessado, completamente encapsulado, dedicado ao dispositivo ou sistema que ele controla. Diferentemente de computadores de propósito geral, como o computador pessoal, um sistema embarcado realiza um conjunto de tarefas predefinidas, geralmente com requisitos específicos. Já que o sistema é dedicado a tarefas específicas, através de engenharia pode-se otimizar o projeto reduzindo tamanho, recursos e custo do produto.

Definição (+ algumas)

Interface com o usuário

Sistemas embarcados podem operar sem nenhuma ou muito pouca interação com seus usuários.

Omnipresentes

Praticamente todo equipamento elétrico, mecânico e químico atual é um sistema embarcado;

Carregadores de celular, eletrodomésticos, semáforos, automóveis, aviões, motores industriais, equipamentos médicos etc.

Diversos níveis de complexidade

Sistemas embarcados controlam desde um eletrodoméstico simples, como um liquidificador, até um foguete recuperável da SpaceX.

Classificação de Sistemas embarcados

Autônomos ou “isolados”

Em rede;

De tempo real;

Móveis.



* As classificações não são mutuamente exclusivas

Classificação de Sistemas embarcados

Autônomos ou “isolados”

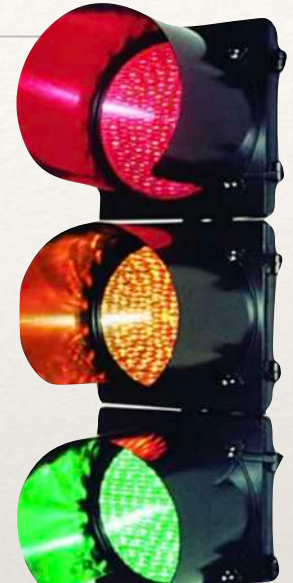
São independentes, não precisam de servidores ou acesso à internet;

Recursos de rede, quando existem, são locais, para acesso a sensores, atuadores, ou a sub-sistemas.

Em rede;

De tempo real;

Móveis.



Classificação de Sistemas embarcados

Autônomos ou “isolados”;

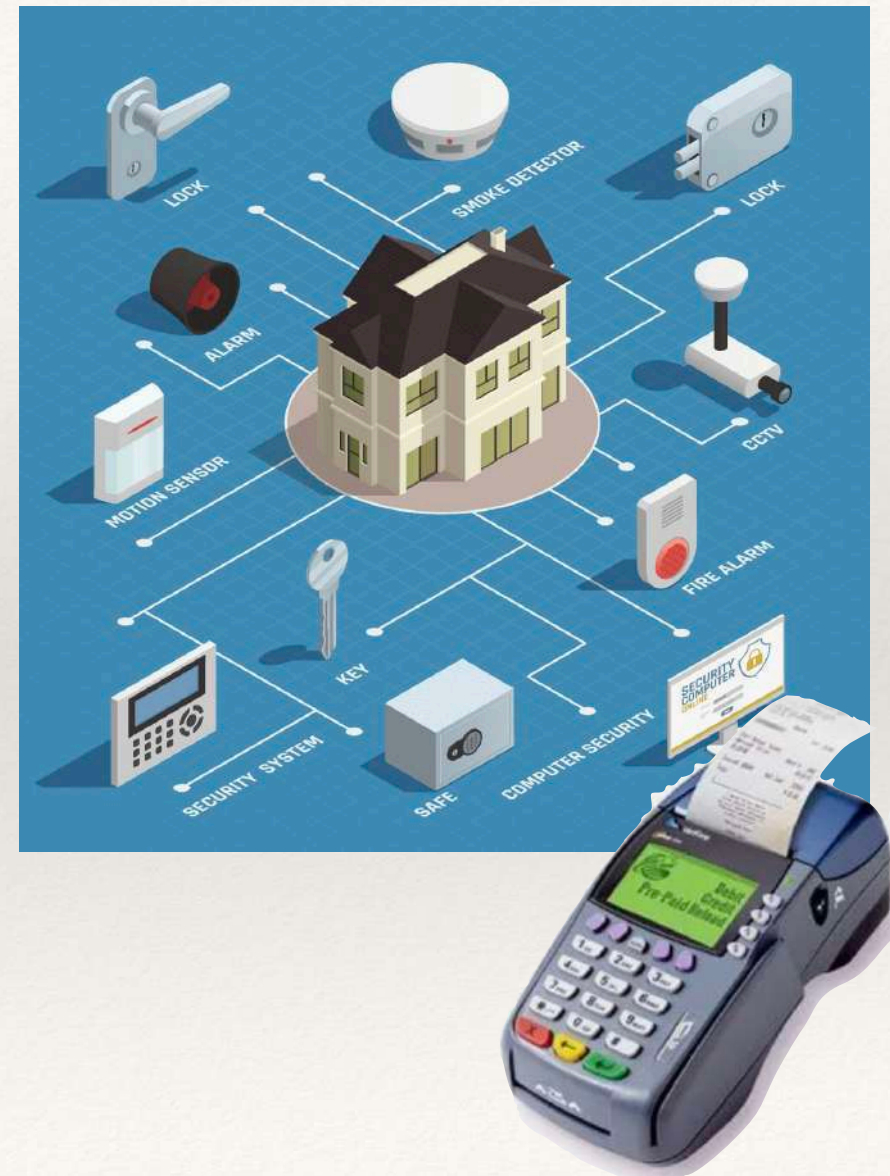
Em rede

Dependem de recursos externos, seja como servidores, ou mesmo para comunicação com outros sistemas independentes;

Incorporam a **IoT (Internet das Coisas)**, que devido à sua abrangência e diversidade, deve-se tratar como um tema à parte;

Representam um desafio para a segurança.

De tempo real;



Classificação de Sistemas embarcados

Autônomos ou “isolados”;

Em rede;

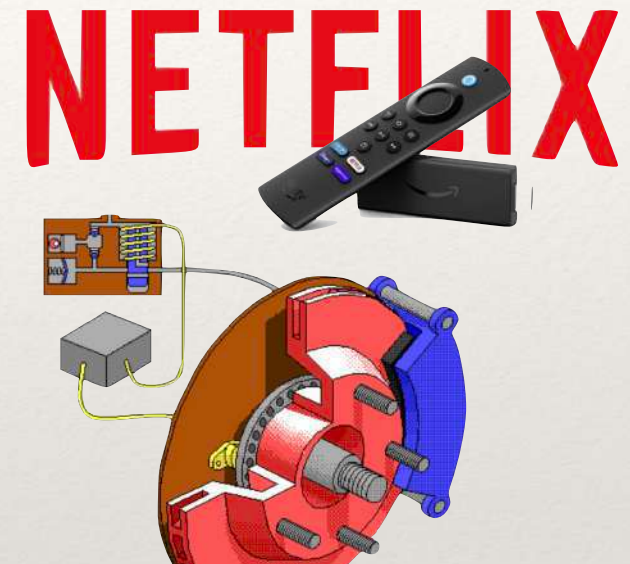
De tempo real

Precisam obedecer a requisitos de tempo de resposta:

Sistemas *Soft Real Time* operam mesmo diante de falhas na obediência a estes requisitos;

Sistemas *Hard Real Time* têm requisitos rígidos, que não podem ser desobedecidos.

Móveis.



Classificação de Sistemas embarcados

Autônomos ou “isolados”

Em rede;

De tempo real;

Móveis

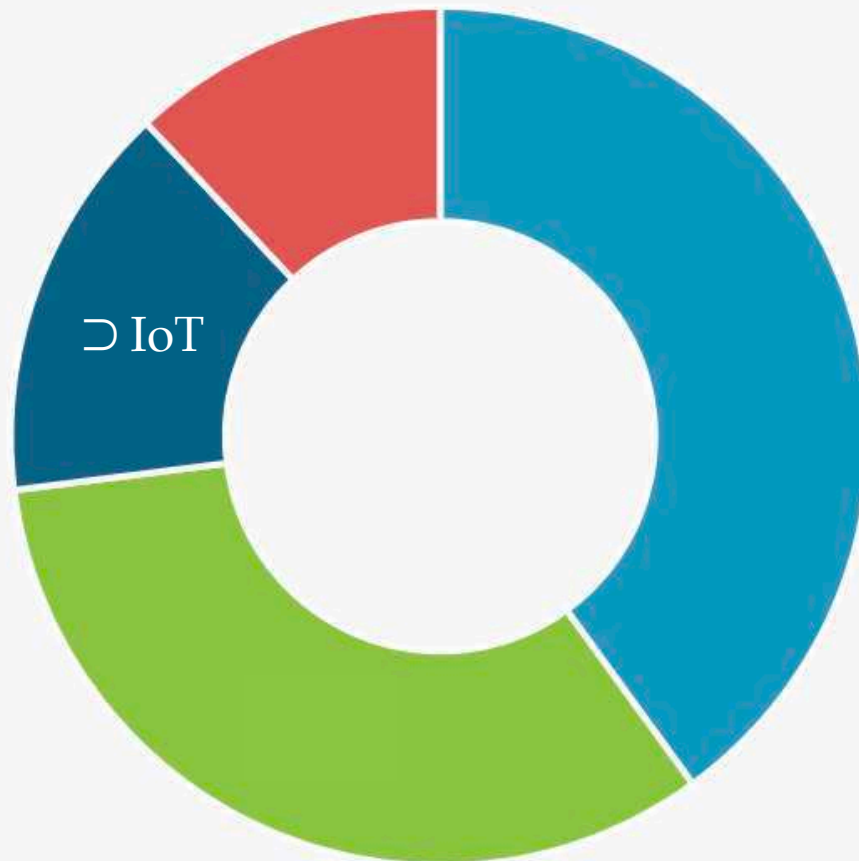
Tipicamente estão em dispositivos de pequeno porte, portáteis e com restrições severas de consumo energético;

Celular é um sistema embarcado !!?



Classificação de Sistemas embarcados

Global Embedded Systems Market Share, By Type, 2022

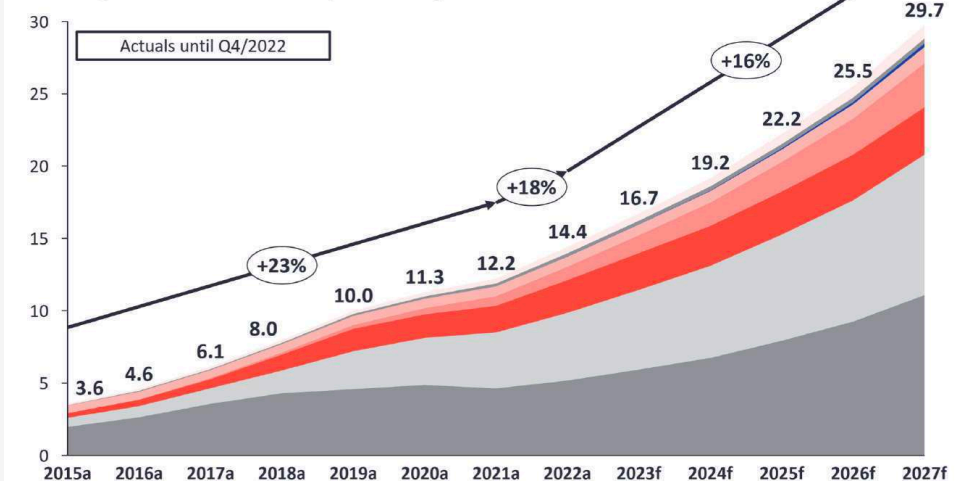


www.fortunebusinessinsights.com

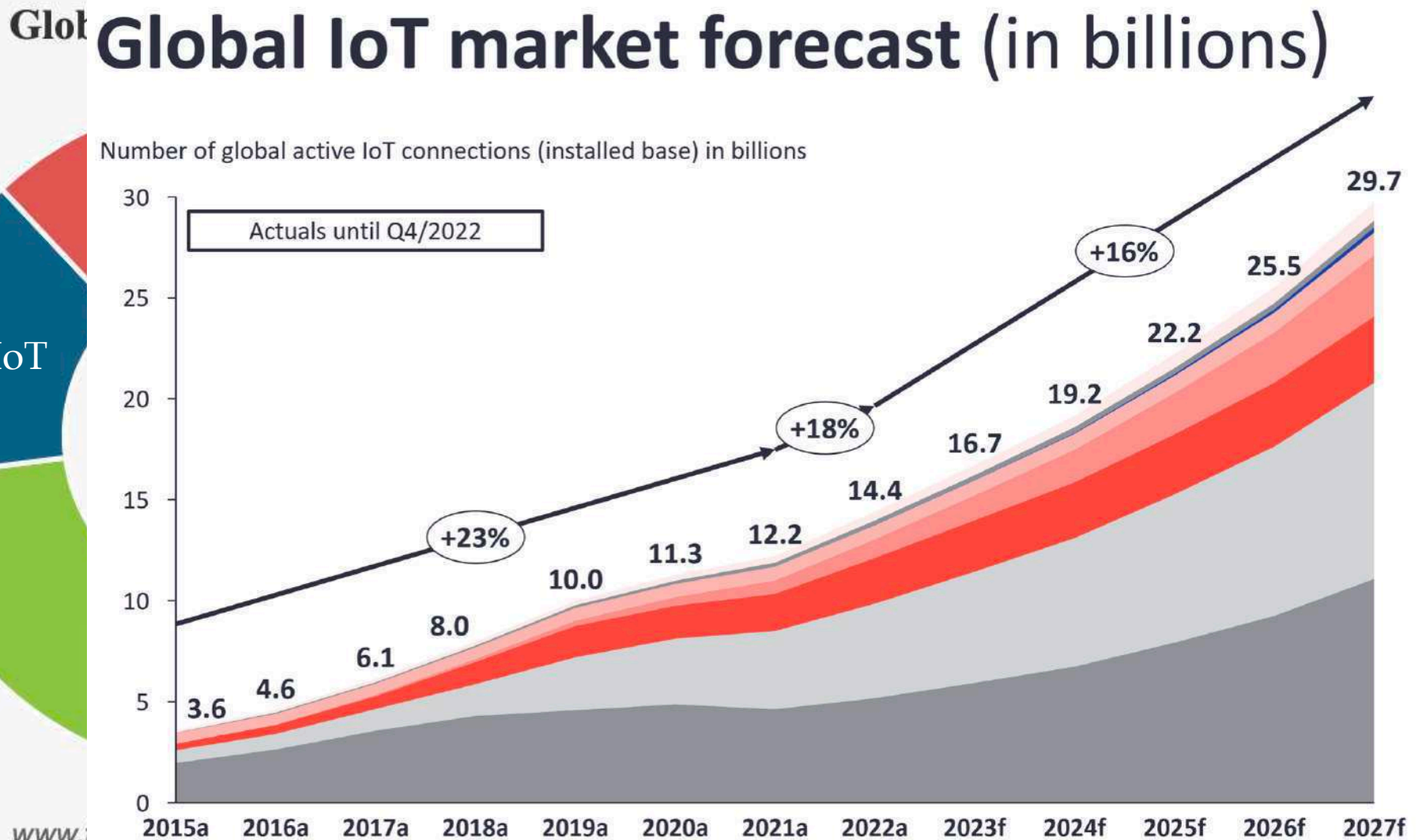
- Standalone Embedded Systems
- Real-Time Embedded Systems
- Network Embedded Systems
- Mobile Embedded Systems

Global IoT market forecast (in billions)

Number of global active IoT connections (installed base) in billions



Classificação de Sistemas embarcados



Hardware

No sistema embarcado, o HW é um conjunto de componentes

Dispositivo “processador”

Atuadores;

Fontes de alimentação;

Sensores diversos;

Interfaces e protocolos de comunicação;

Componentes eletrônicos diversos.

Escolher corretamente cada um dos componentes é essencial

A portabilidade pode ser inviável;

O *hardware* determina os parâmetros de projeto, inclusive de *software*.

Outro fator importante são as condições ambientais

Acesso, proteção, temperatura, vibração, EMI etc

Além de tudo isso, tem o SW !

Ele é a verdadeira “inteligência” da automação.

Hardware

No sistema embarcado, o HW é um conjunto de componentes

Dispositivo “processador”

Atuadores;

Fontes de alimentação;

Sensores diversos;

Interfaces e protocolos de comunicação;

Componentes eletrônicos diversos.

Escolher corretamente cada um dos componentes é essencial

A portabilidade pode ser inviável;

O *hardware* determina os parâmetros de projeto, inclusive de *software*.

Outro fator importante são as condições ambientais

Acesso, proteção, temperatura, vibração, EMI etc

Além de tudo isso, tem o SW !

Ele é a verdadeira “inteligência” da automação.

Processador

Um sistema embarcado de automação moderno tipicamente exige um processador. Temos algumas opções:



*CISC Intel i9-13900H
14 núcleos
20 threads
24MB cache
Clock 2.6GHz até 5.4GHz
Consumo de 45W
64 GB DDR5
SSD 1TB PCIe NVM M.2*

Processador

Um sistema embarcado de automação moderno tipicamente exige um processador. Temos algumas opções:



CISC Intel i9-13900T
14 núcleos
20 threads
24MB cache
Clock 2.6GHz até 5.4GHz
Consumo de 45W
64 GB DDR5
SSD 1TB PCIe NVMe M.2

Processador

Um sistema embarcado de automação moderno tipicamente exige um processador. Temos algumas opções:



*RISC Apple M3 Max
14 núcleos
40 núcleos GPUs
16 núcleos neurais
Clock 2.7GHz até 4.1GHz
Consumo de 78W
36 GB Memória Unificada
SSD 1TB*

Processador

Um sistema embarcado de automação moderno tipicamente exige um processador. Temos algumas opções:



*RISC Apple M3 Max
14 núcleos
40 núcleos GPUs
16 núcleos neurais
Clock 2.7GHz até 4.1GHz
Consumo de 78W
36 GB Memória Unificada
SSD 1TB*

Processador

Um sistema embarcado de automação moderno tipicamente exige um processador;

São máquinas maravilhosas, rápidas e modernas, MAS ...

Desempenho e capacidade extraordinária de armazenamento não são tão importantes neste tipo de aplicação !

Quais são os requisitos típicos de um processador para um sistema de automação ?

Consumo baixo de energia;

Dimensões reduzidas;

Custo acessível.

Processador

Não faz sentido aumentar complexidade, e com isto consumo, tamanho e custo ...



Processador

Microprocessadores

Tipicamente baseados na arquitetura Von Neumann;

Tipicamente divide-se em ULA, Unidade de Controle e registradores;

Exige componentes externos essenciais, como memórias, barramentos, *clock*, interfaces de E/S etc;

Projetados para múltiplos usos;

Devido a estas dependências, sistemas microprocessados ocupam mais espaço.

Microcontroladores

Tipicamente baseados na arquitetura de Harvard;

Além dos componentes presentes em um microprocessador, condensa periféricos como memória, *clock*, interfaces de E/S diversas, conversores A/D e D/A, e portas de acesso GPIO;

Projetados para uso específico;

Tipicamente são limitados em performance e capacidade para redução do consumo, tamanho e custo.

Arquitetura Básica

Von Neumann

Conceito desenvolvido pela equipe de John Von Neumann no IAS (*Institute for Advanced Studies*) da Universidade de Princeton em 1946;

Tornou-se um padrão de fato, inspirando projetos até os dias atuais (**"Esta máquina é Von Neumann ?"**)

Estabeleceu o padrão de armazenamento do código executável na memória do dispositivo;

Executa uma instrução em dois ciclos;

Possui um barramento de cada tipo (dados, endereços e controle).

Harvard

Conceito desenvolvido para o computador Mark I, desenvolvido por Howard Aiken em 1937.

Separa memórias de uso específico (dados e instruções), que são acessíveis por barramentos diferentes;

Comum em microcontroladores, e em processadores DSP (*Digital Signal Processing*);

Executa uma instrução em um ciclo;

Permite a implementação do *pipelining*, técnica de processamento paralelo facilitada pelos barramentos independentes.

Computador "Von Neumann"

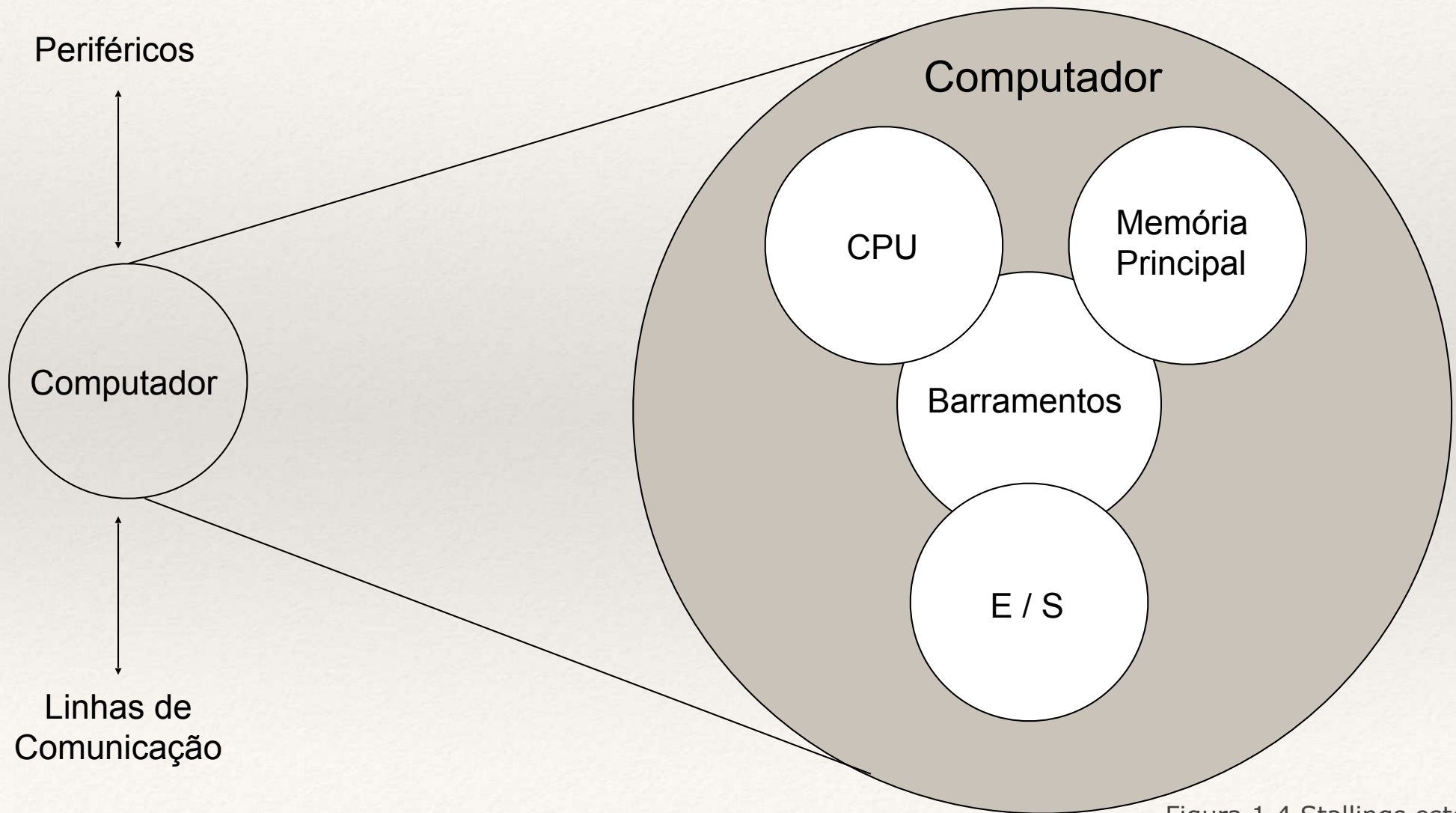


Figura 1.4 Stallings estendida

Microprocessador

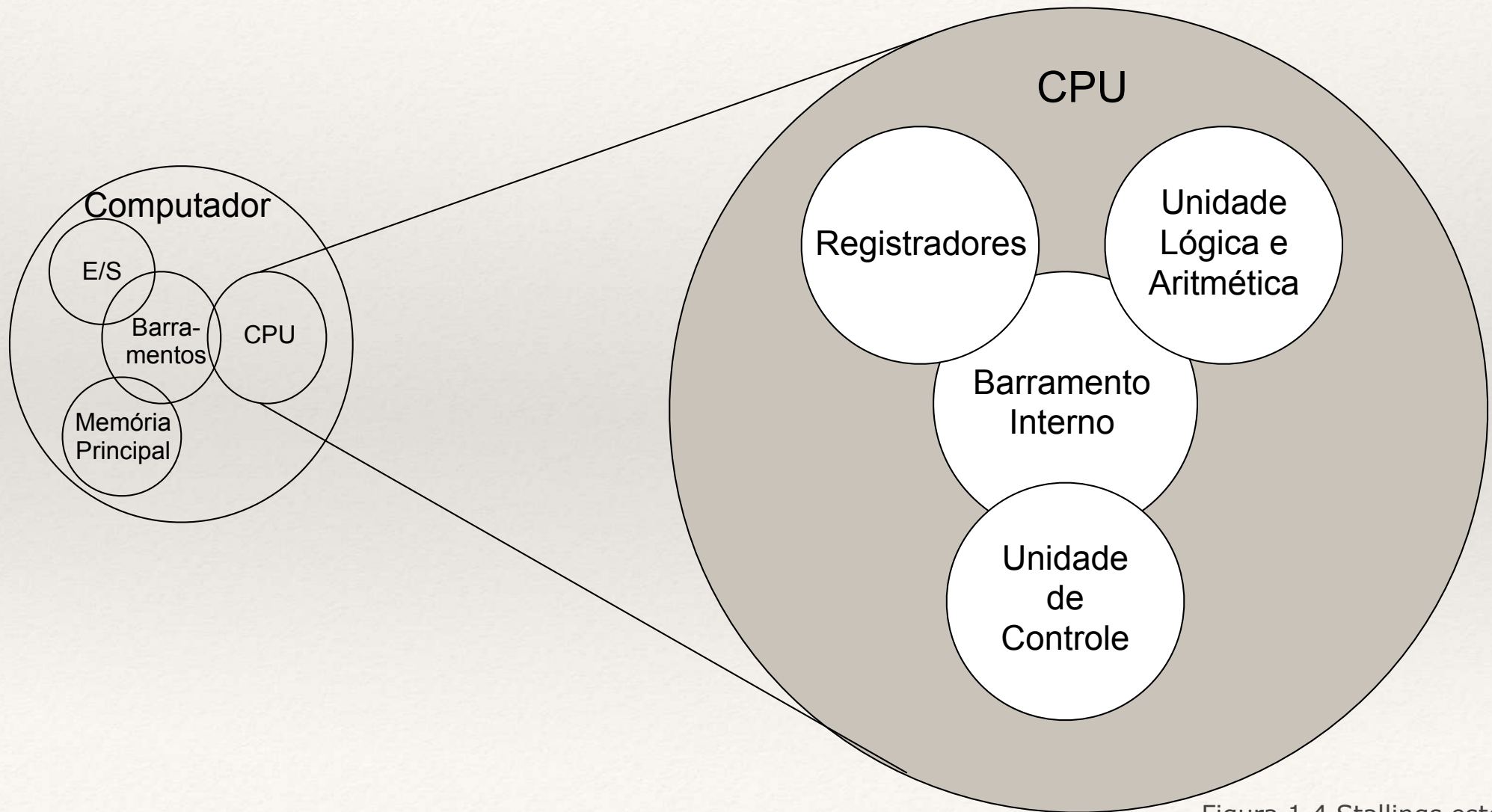


Figura 1.4 Stallings estendida

Unidade de Controle

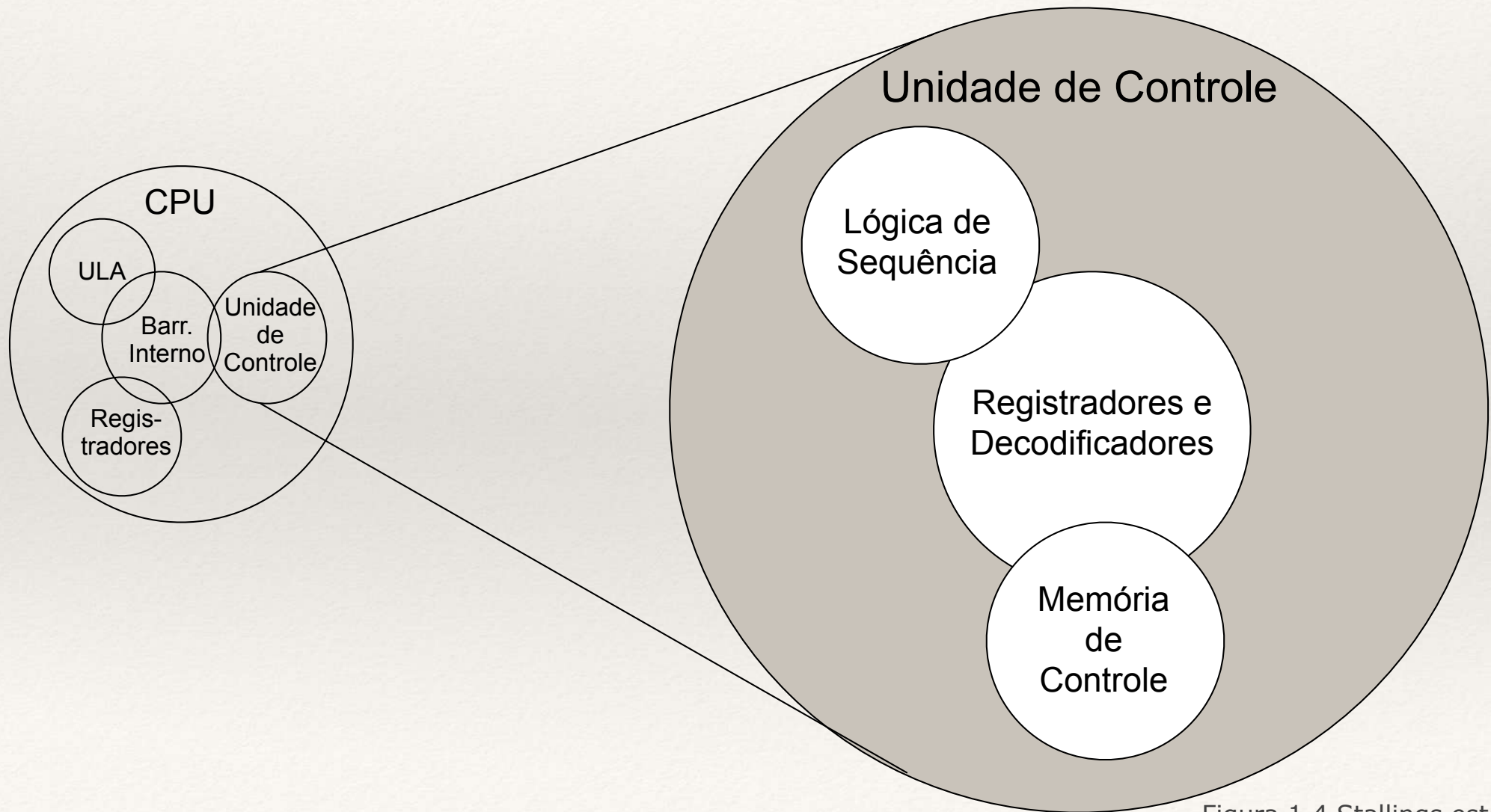


Figura 1.4 Stallings estendida

Estrutura

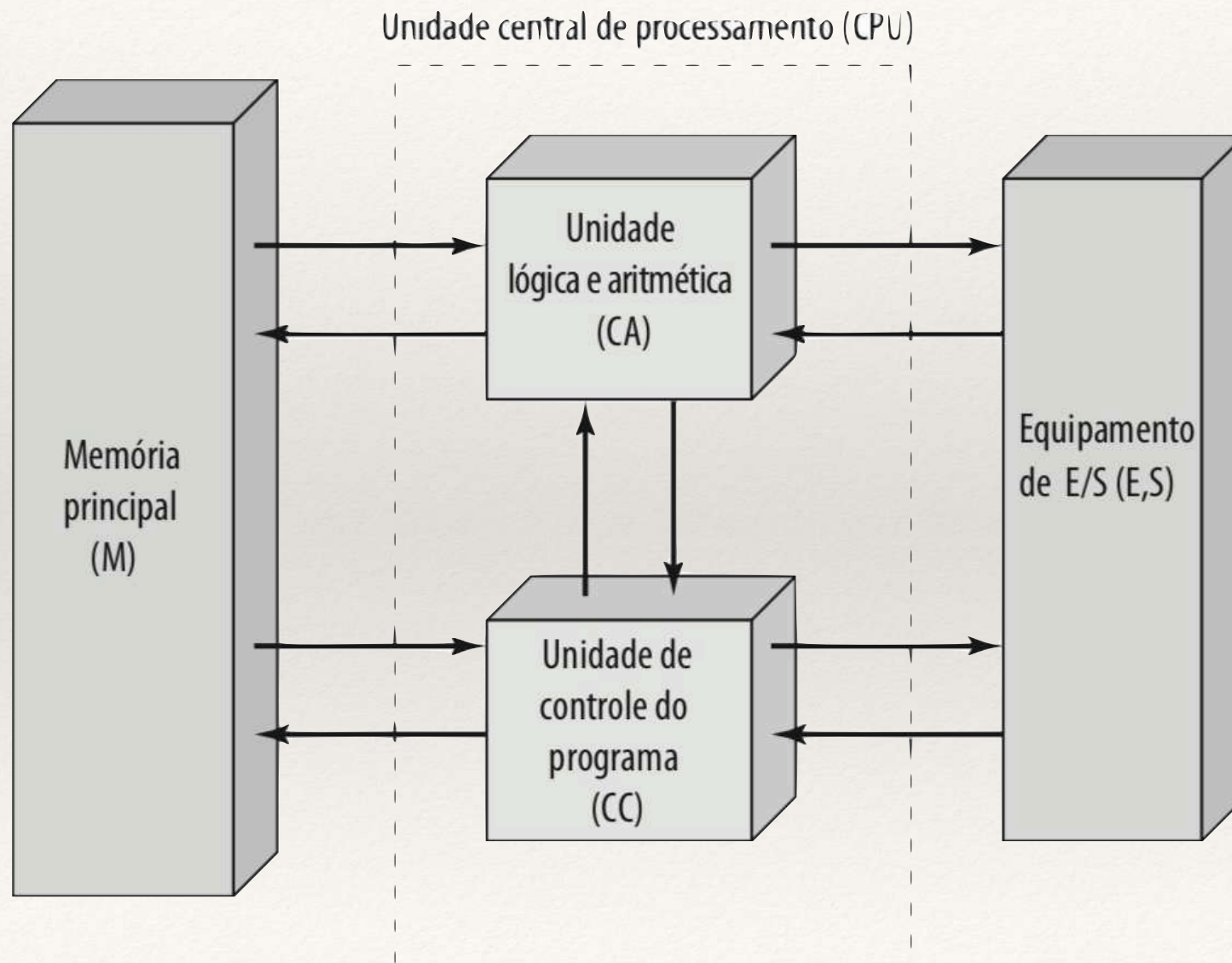


Figura 2.1 Stallings

Barramentos

Os componentes trocam informações;

Na arquitetura de Von Neumann são 3 os barramentos:

Endereços

Dados

Controle

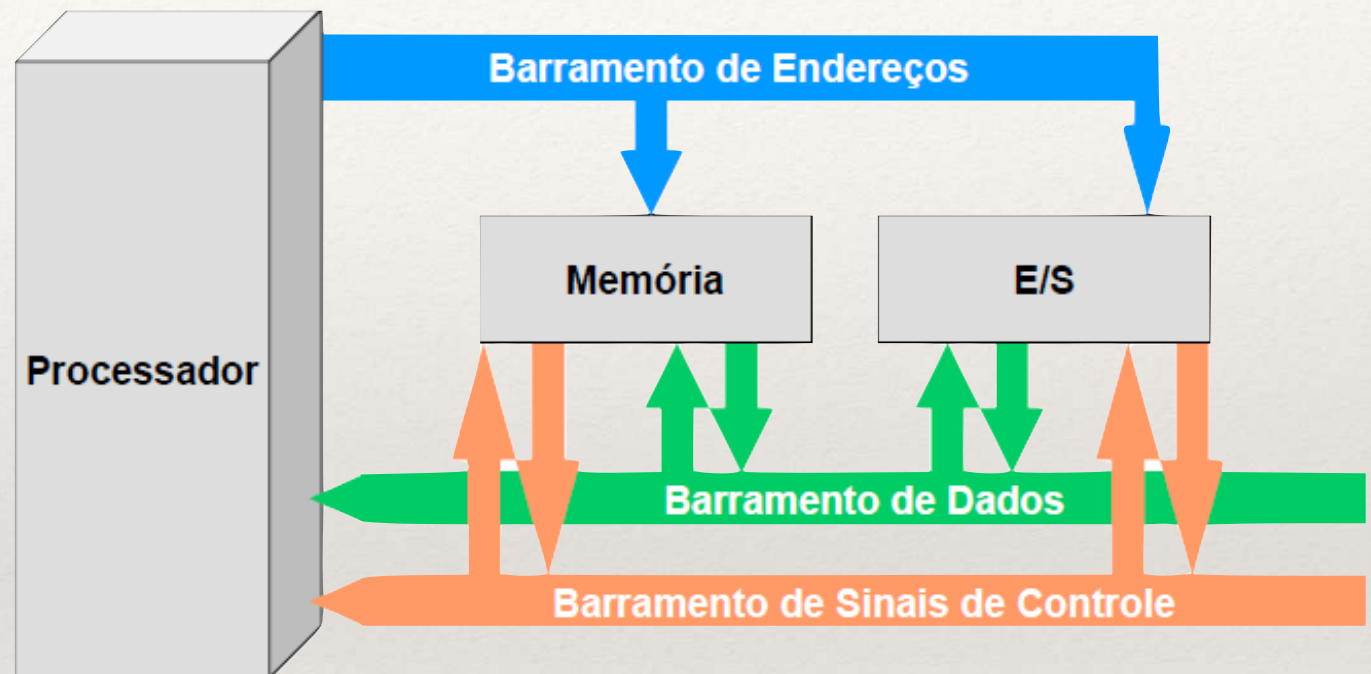
As setas indicam o fluxo de informações pelos barramentos:

Apenas a CPU escreve no barramento de endereços, mas lê e escreve no barramento de dados;

A Memória pode ser escrita ou lida (barramento de dados);

O mesmo acontece com os periféricos de Entrada/Saída;

Todos escrevem e lêem no barramento de controle, a depender do sinal.



Barramentos

Barramento de Endereços

Fan-out no barramento de endereços:

Supondo 16 GBytes de memória, e palavra de 64 bits, temos:

$$\frac{16 \times 1024 \times 1024 \times 1024 \times 8}{64} = 1024 \times 1024 \times 1024 \times 2 \approx 2 \text{ bilhões de palavras}$$

Cada uma das palavras precisa ler o barramento de endereços para identificar o destinatário.

Mesmo para correntes de entrada baixas (Ex. 10uA), teríamos um consumo de $2 \times 10^9 \times 10^{-6} = 2000A$!?!

O problema exige soluções específicas !

Barramentos

Barramento de Dados

Tri-state no barramento de dados:

Como duas saídas diferentes de memória podem compartilhar o mesmo barramento em uma leitura da CPU? (curto-circuito ???)

O terceiro estado (desconectado) é essencial;

Apenas dois dispositivos ficam “ativos”: UM emissor e um receptor.

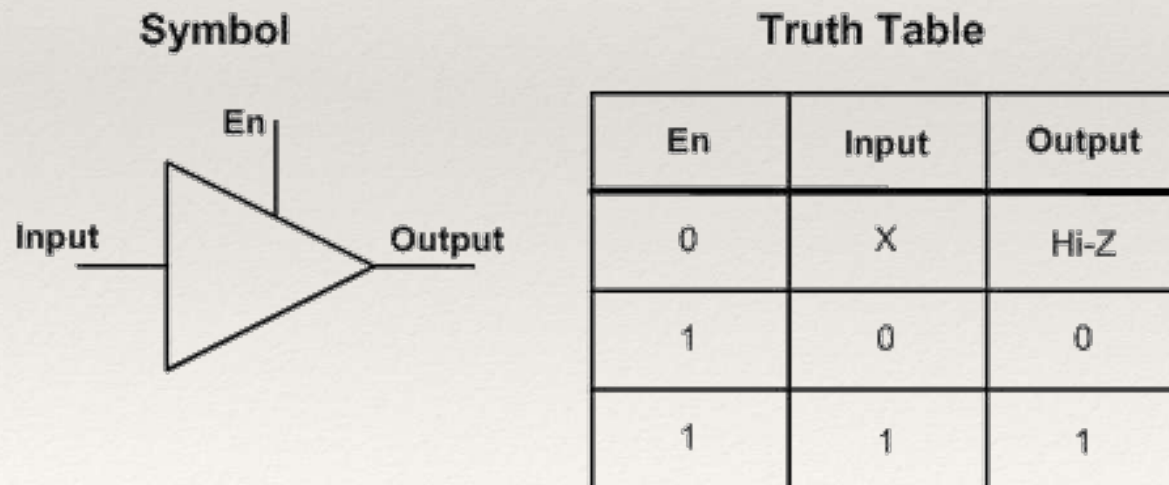


Fig. 3 Tri-state buffer symbol and truth table

Barramentos

Barramento de Sinais de Controle

Alguns sinais tipicamente presentes no barramento de Controle:

BREQ (*Bus Request*): requisição de barramentos por um periférico que passará a controlar o ambiente. Ex: DMA (*Direct Memory Access*)

CLK (Clock): sinal de *clock*;

INTR (*Interrupt Request*): solicitação de interrupção externa;

R/ $\overline{W}$ (*Read/ $\overline{Write}$*): identifica o tipo de operação que será feita na memória ou periférico. Em algumas arquiteturas, há pinos separados;

RDY (Ready): sinaliza a conclusão de alguns procedimentos;

RST (*Reset*): reinicialização dos componentes.

É comum termos sinais ativos em nível zero, o que é representado pela barra acima do sinal correspondente.

Barramentos

Vantagens

A ampliação do número de dispositivos e usuários é trivial;
Baixo custo.

Desvantagens

Compartilhamento pode provocar retenção de tráfego, com queda de desempenho, e possível *jitter* associado.

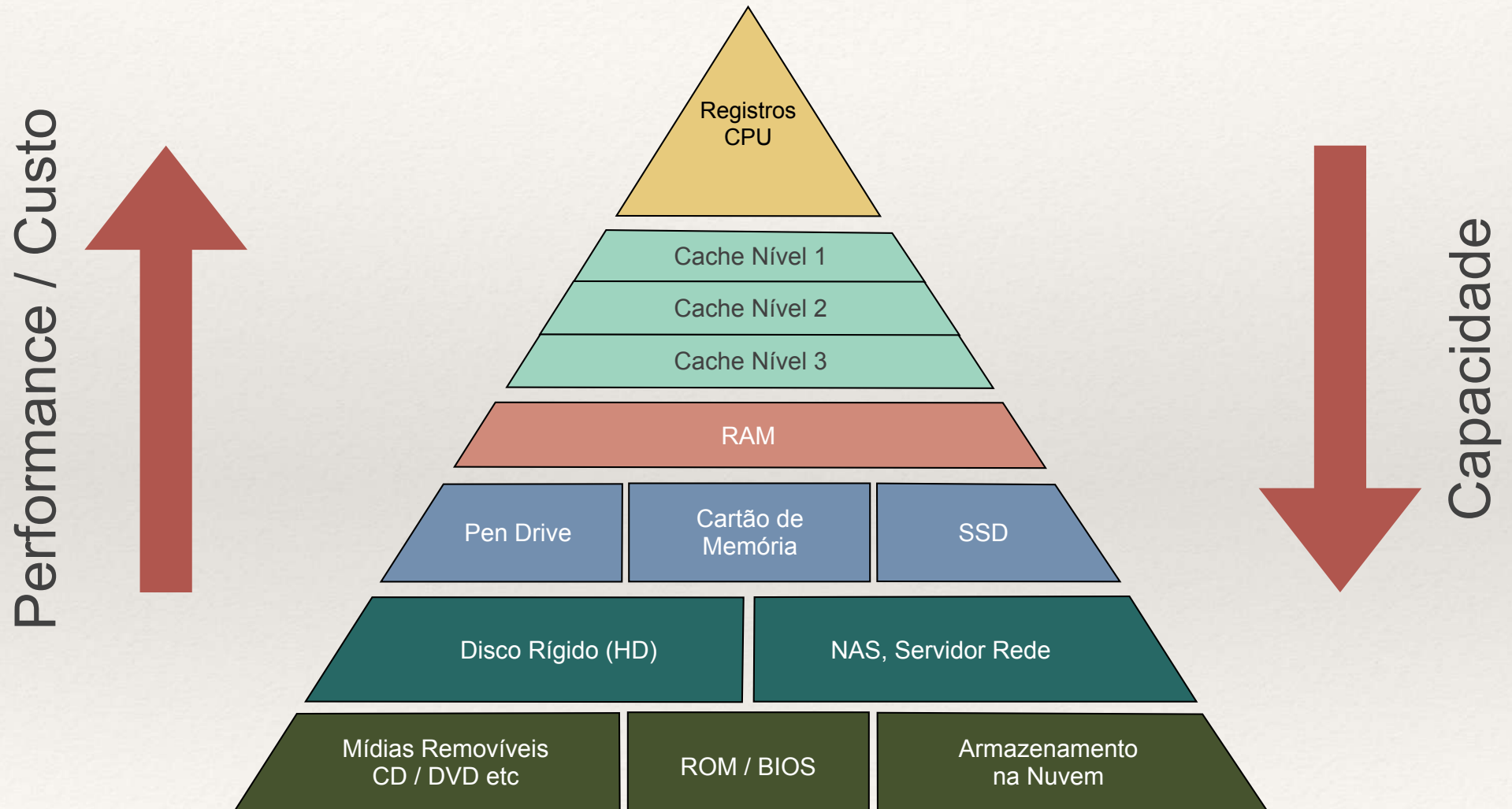
Aula 02

Memória - Características

Tabela 4.1 Principais características dos sistemas de memória do computador

Localização Interna (por exemplo, registradores do processador, memória principal, cache) Externa (por exemplo, discos ópticos, discos magnéticos, fitas)	Desempenho Tempo de acesso Tempo de ciclo Taxa de transferência
Capacidade Número de palavras Número de bytes	Tipo físico Semicondutor Magnético Óptico Magneto-óptico
Unidade de transferência Palavra Bloco	Características físicas Volátil/não volátil Apagável/não apagável
Método de acesso Sequencial Direto Aleatório Associativo	Organização Módulos de memória

Hierarquia de Memória



Comparativo de Performance

Nível	1	2	3	4	5
Nome	registradores	cache	memória principal	disco de estado sólido	disco magnético
Tamanho típico	< 1 KB	< 16 MB	< 64 GB	< 1 TB	< 10 TB
Tecnologia de implementação	memória personalizada com várias portas CMOS	CMOS SRAM em chip ou fora do chip	CMOS SRAM	memória flash	disco magnético
Tempo de acesso (ns)	0,25 – 0,5	0,5 – 25	80 – 250	25.000 – 50.000	5.000.000
Largura de banda (MB/s)	20000 – 100000	5000 – 10000	1000 – 5000	500	20 – 150

Conjunto de Instruções

Mnemonic, Operands	Description	Cycles	14-Bit Instruction Word				Status Affected	
			MSb		LSb			
BYTE-ORIENTED FILE REGISTER OPERATIONS								
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C,DC,Z
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z
CLRF	f	Clear f	1	00	0001	1fff	ffff	Z
CLRW	-	Clear W	1	00	0001	0xxx	xxxx	Z
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00	1011	dfff	ffff	
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00	1111	dfff	ffff	
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z
MOVWF	f	Move W to f	1	00	0000	1fff	ffff	
NOP	-	No Operation	1	00	0000	0xx0	0000	
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C,DC,Z
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff	
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z
BIT-ORIENTED FILE REGISTER OPERATIONS								
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff	
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff	
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01	10bb	bfff	ffff	
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01	11bb	bfff	ffff	

Conjunto de Instruções

Define instruções em “linguagem de máquina” reconhecidas pelo dispositivo

A programação tipicamente é realizada em “assembly”, que representa cada instrução com uma sigla (mneumônico);

Veremos outras formas mais simples de programar um microcontrolador ao longo da disciplina.

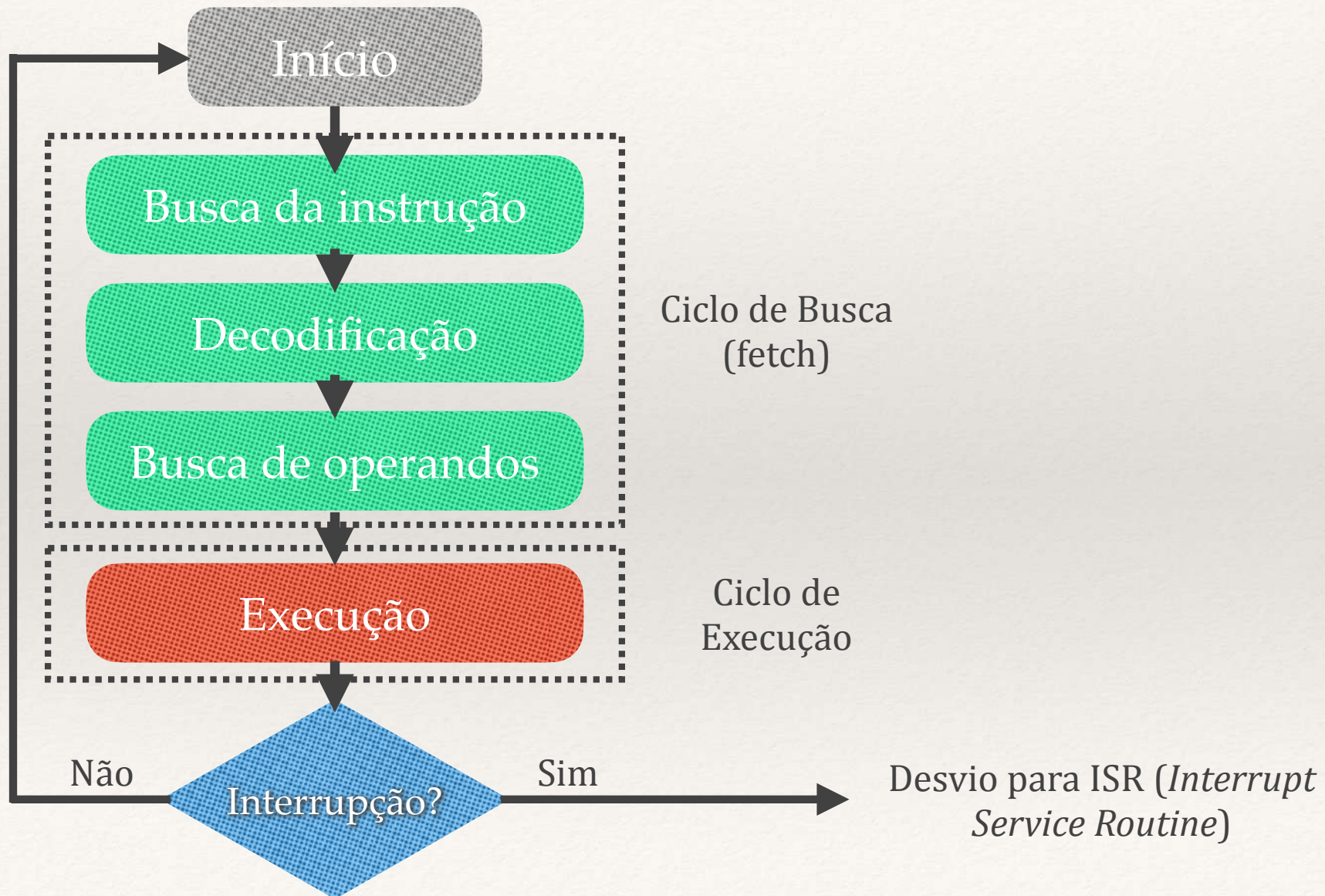
Arquiteturas RISC x CISC

(Reduced / Complete) Instruction Set Computer;

O PIC16F877, por exemplo, possui apenas 35 instruções (RISC), divididas em seis grupos;

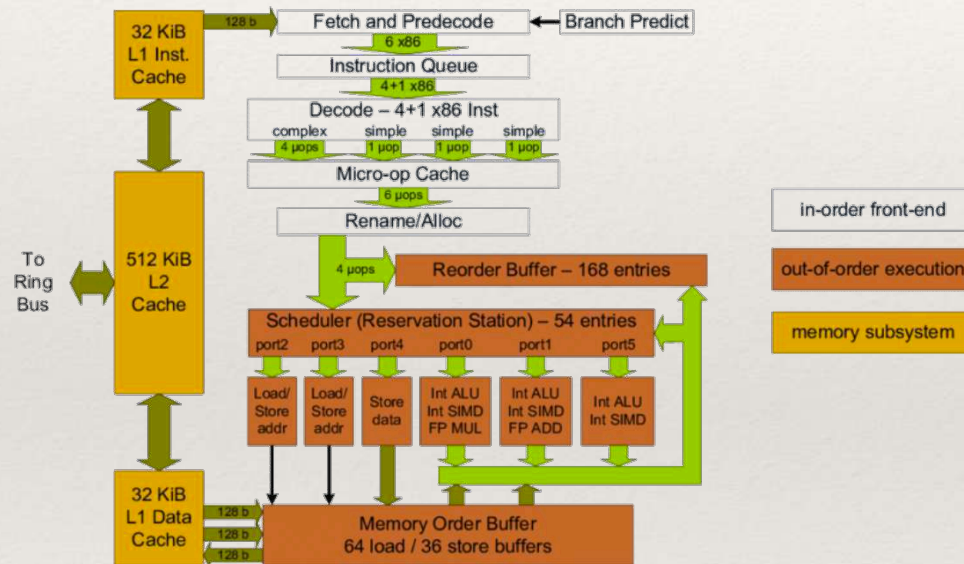
RISC tipicamente oferece performance maior, porém traz desafios para a programação.

Execução de Programa

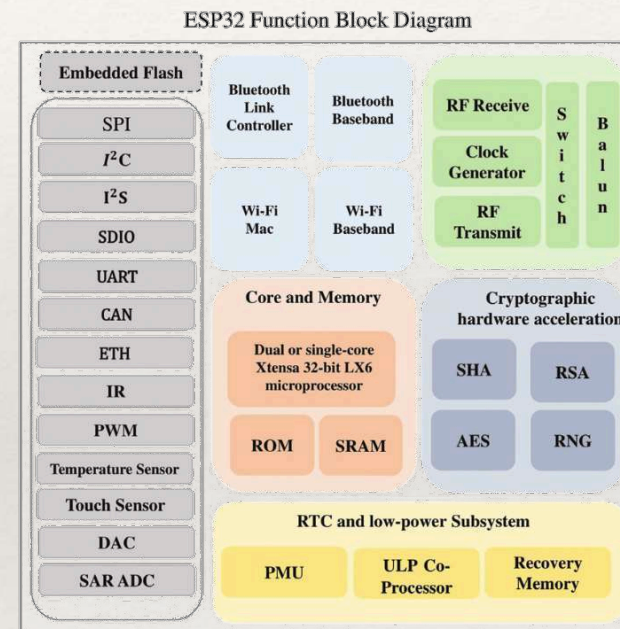


Opções de Processador

Microprocessadores



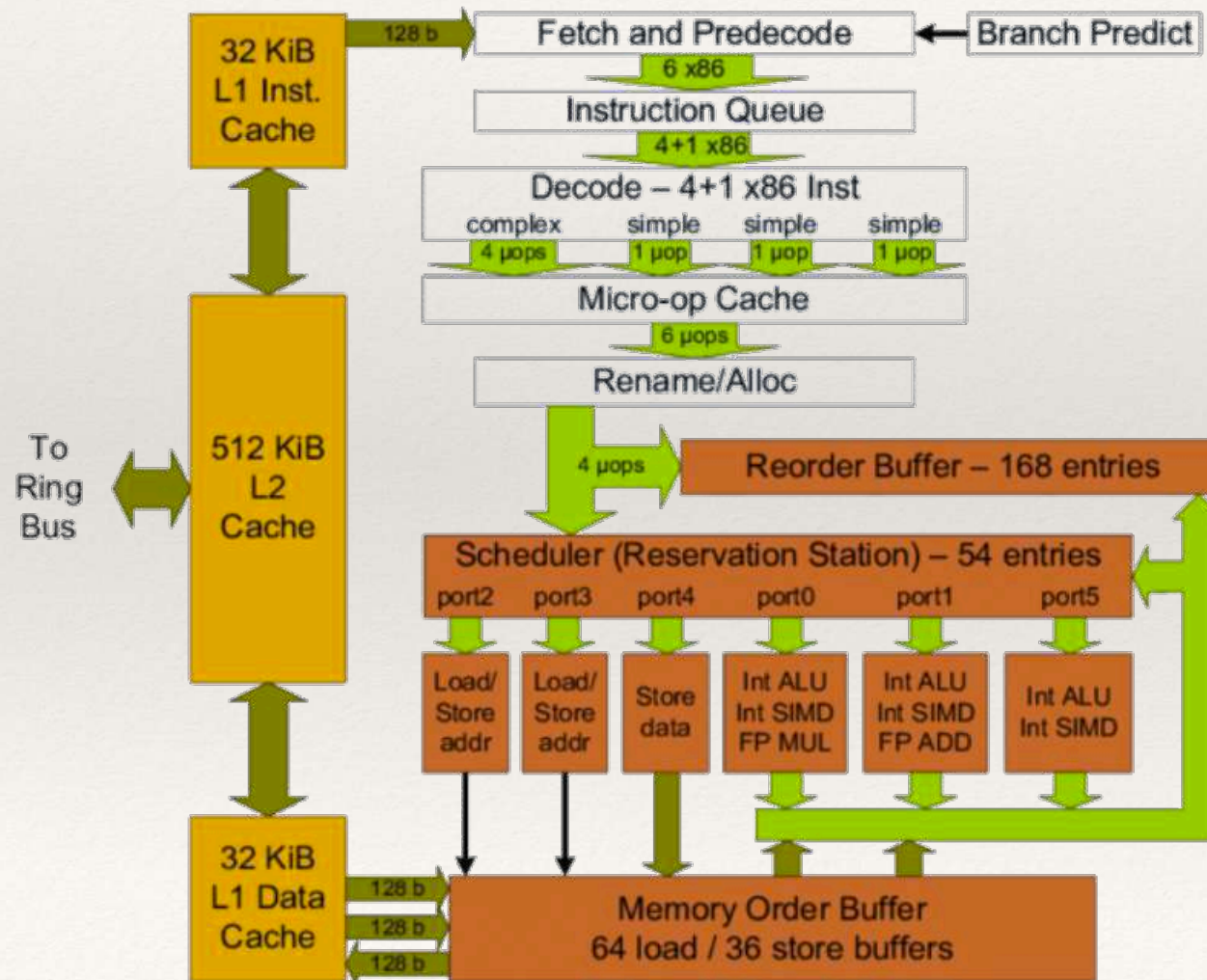
Microcontroladores



Microprocessadores

Microprocessadores

Diagrama funcional em blocos de um processador Intel (*Sandy Bridge*)

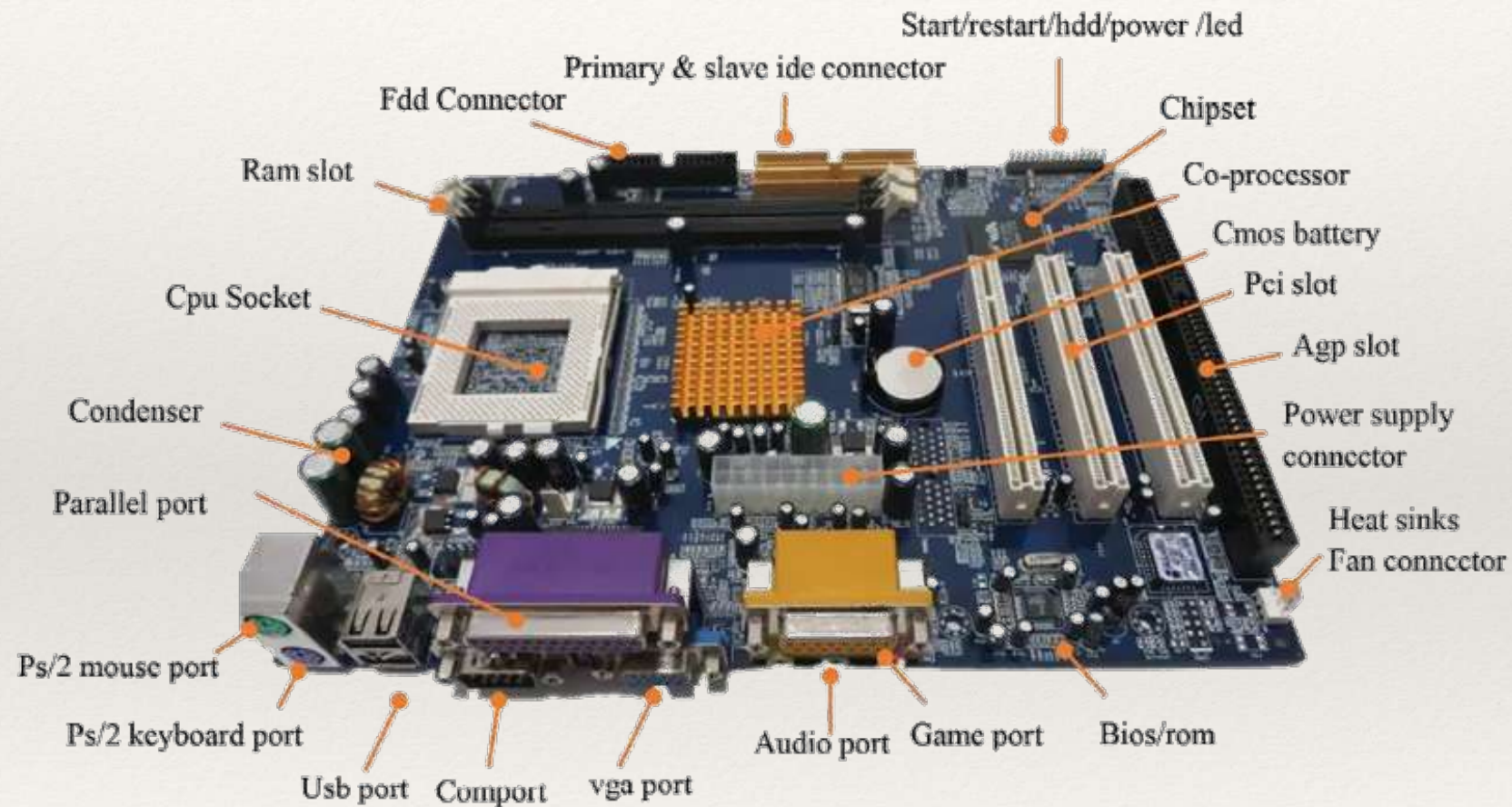


in-order front-end

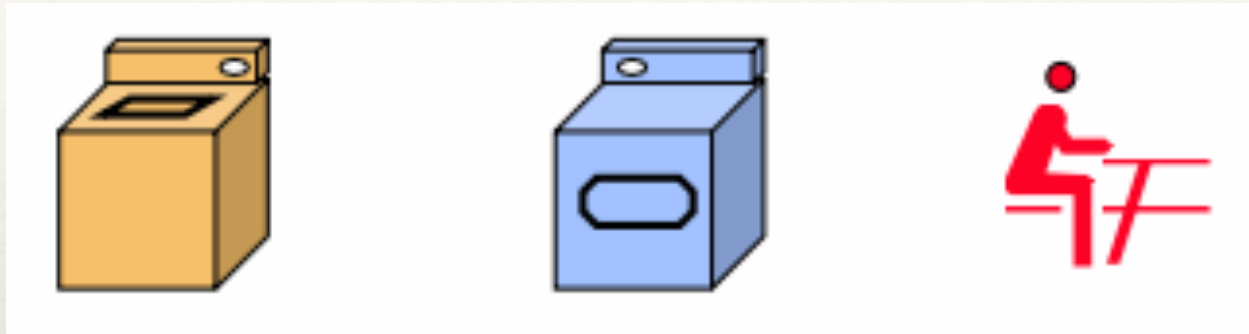
out-of-order execution

memory subsystem

Microprocessadores



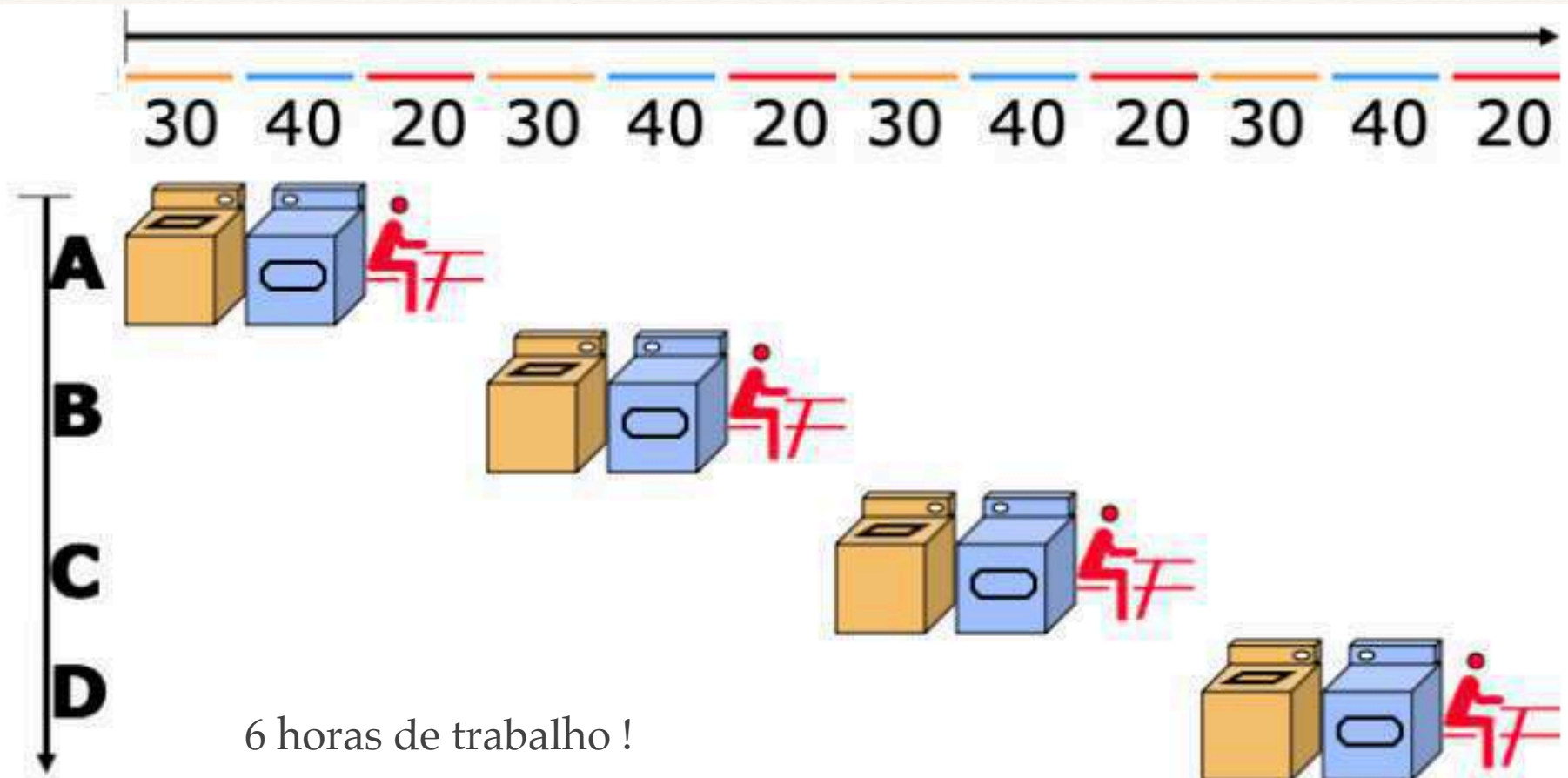
Pipeline - Uma analogia



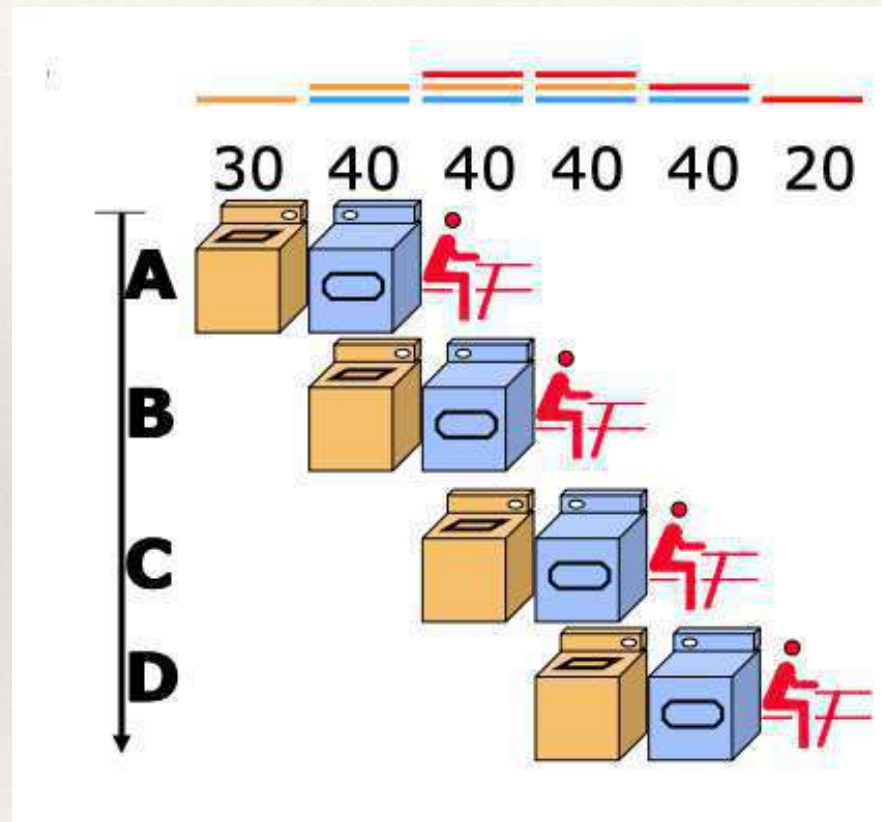
Ana, Beto, Carol e Damásio precisam lavar suas roupas na lavanderia do prédio, mas só existe uma lavadora, uma secadora e uma mesa para dobrar as roupas;

A lavagem demora 30 minutos, a secagem 40 minutos, e para dobrar são necessários mais 20 minutos. Ou seja, 1 hora e meia para tudo.

Sem *pipeline*

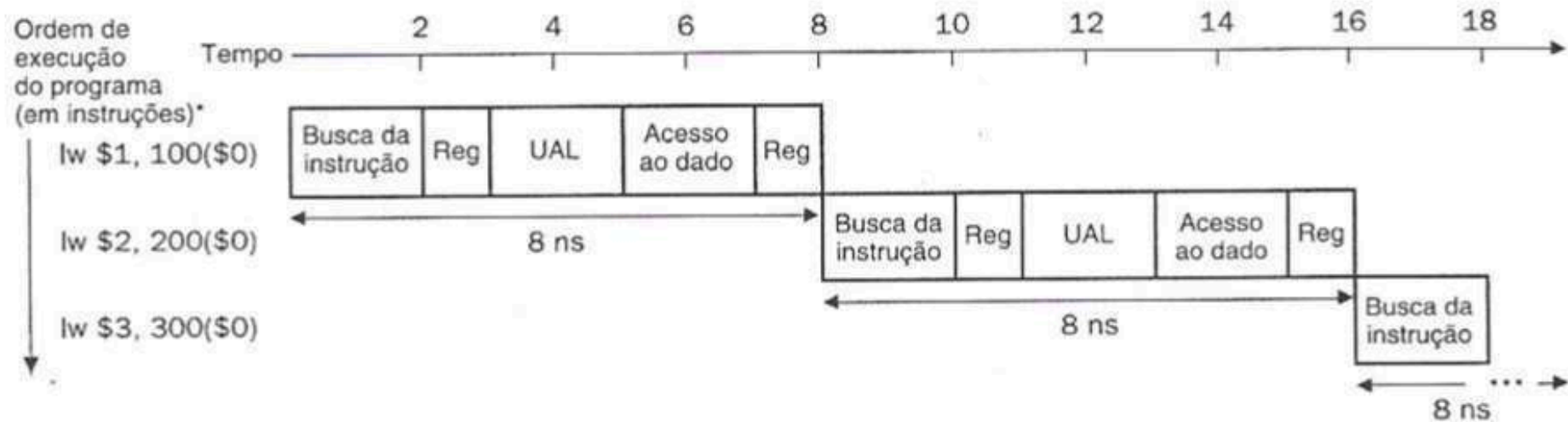


Com *pipeline*



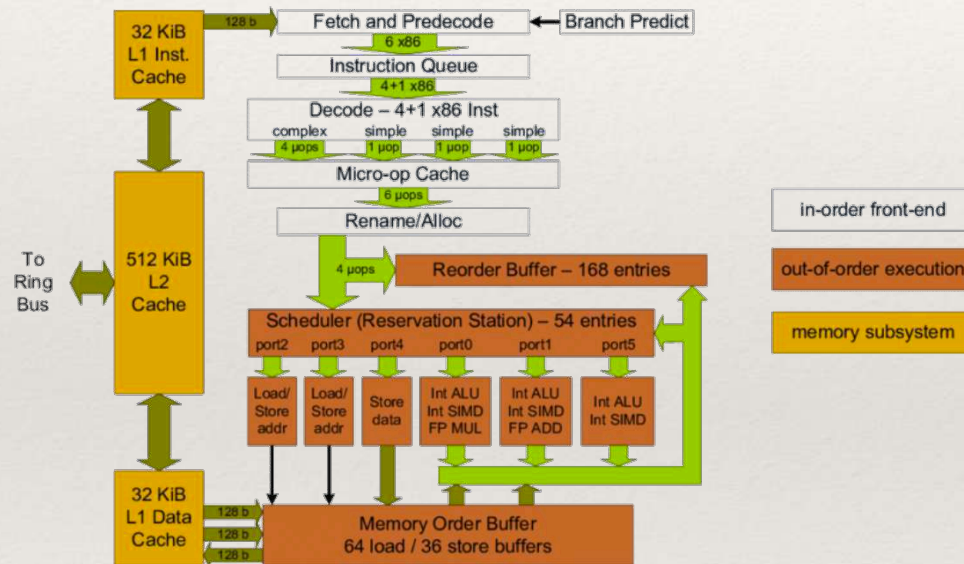
3:30h de trabalho !

Pipeline - Ciclos de instrução

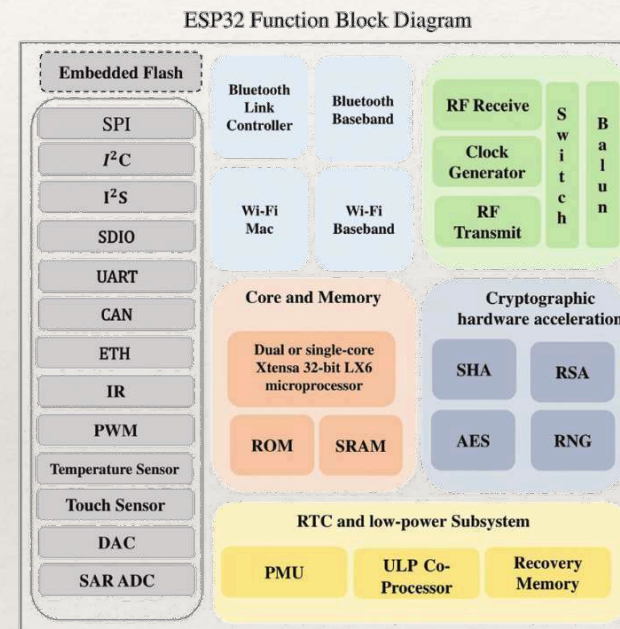


Opções de Processador

Microprocessadores



Microcontroladores



Microcontroladores

ESP32 Function Block Diagram

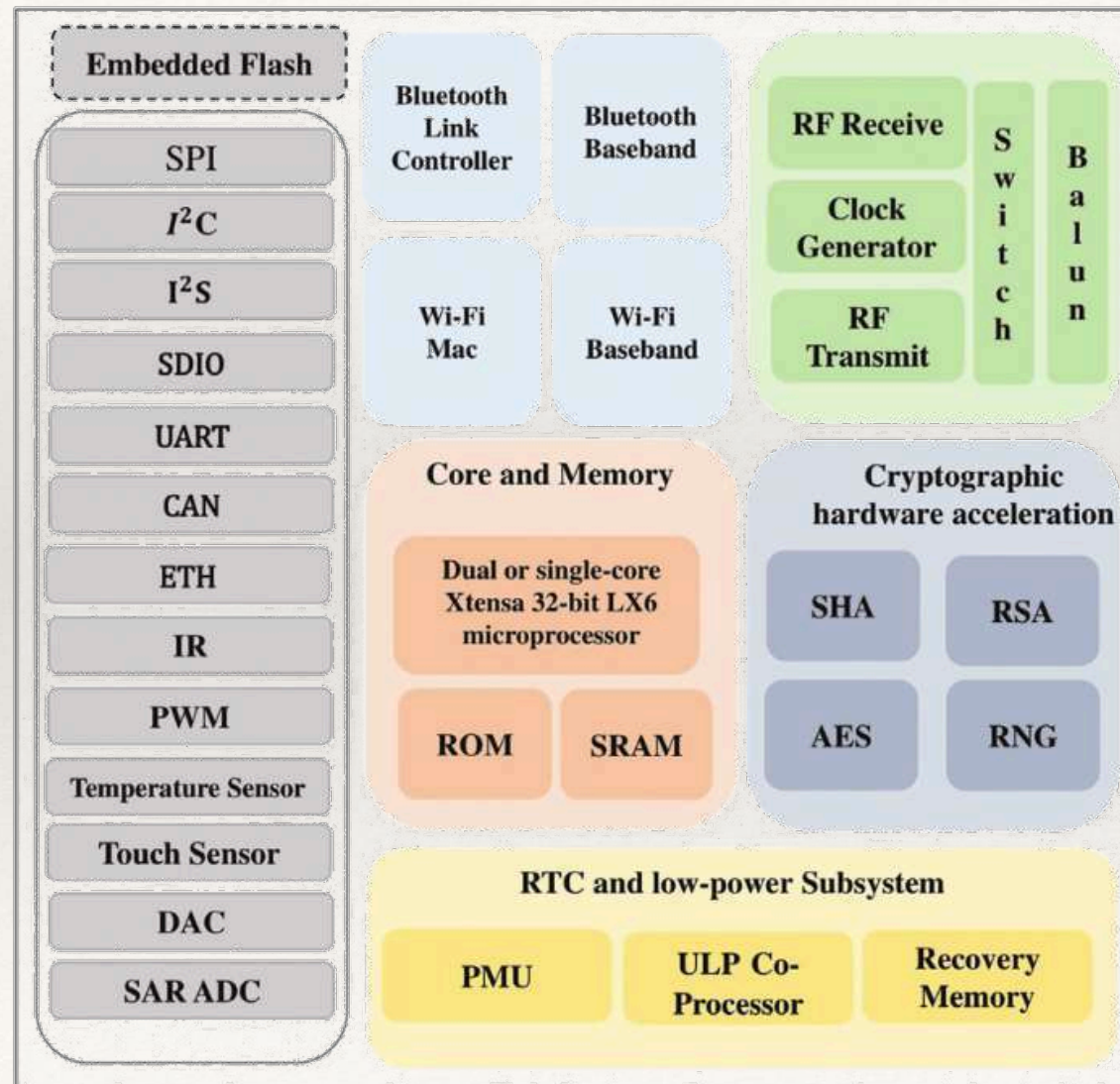


Diagrama funcional em blocos de um microcontrolador Expressif (ESP32)

Microcontroladores

ESP32 Function Block Diagram

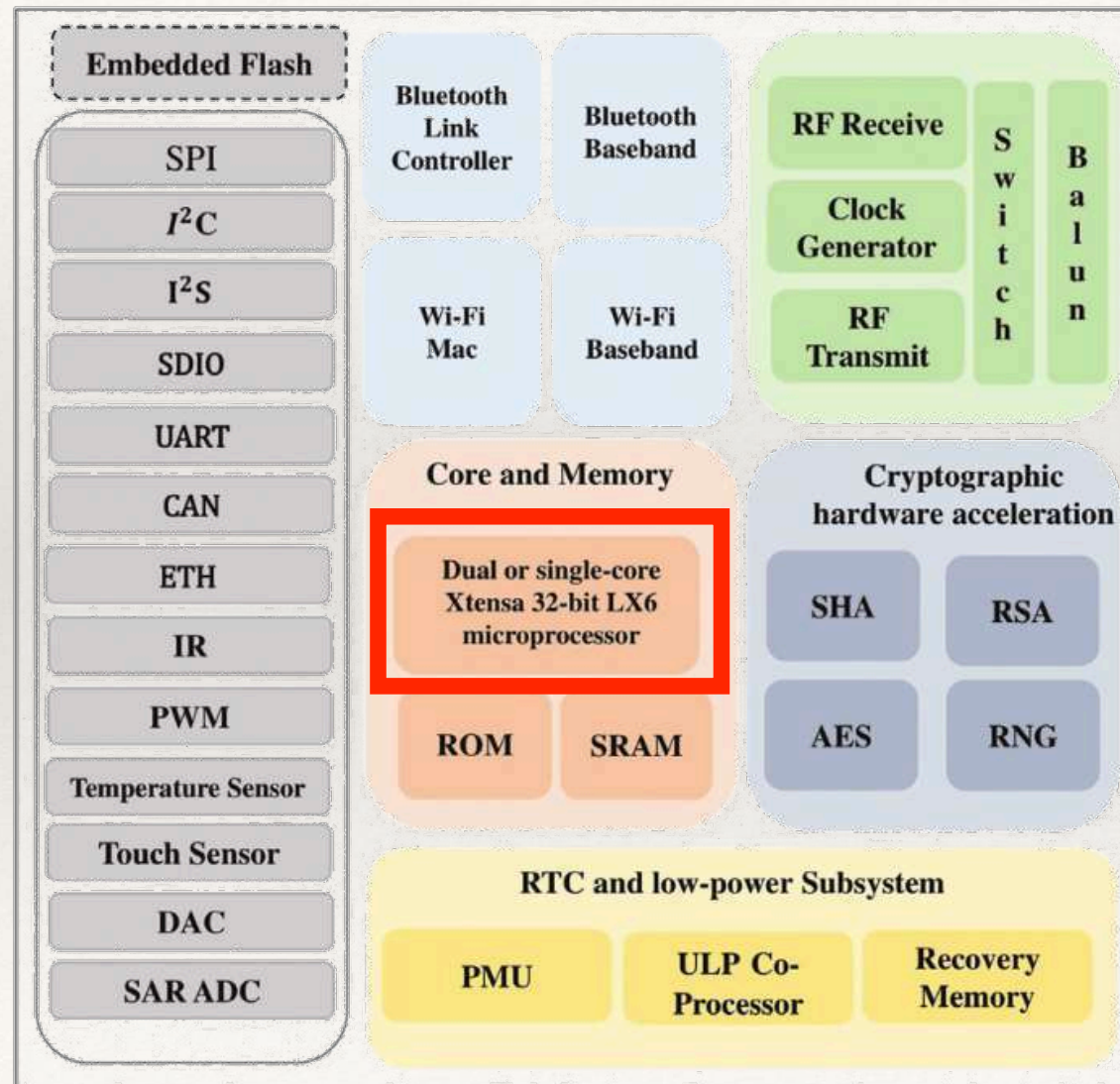
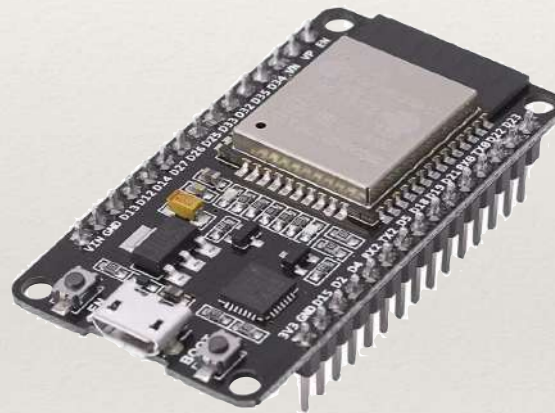


Diagrama funcional em blocos de um microcontrolador Expressif (ESP32)

Microcontroladores

Conceito de SoC
(*System on a Chip*)

* Na foto, uma placa de desenvolvimento do ESP32.



Tamanho (componentes)

Aplicabilidade

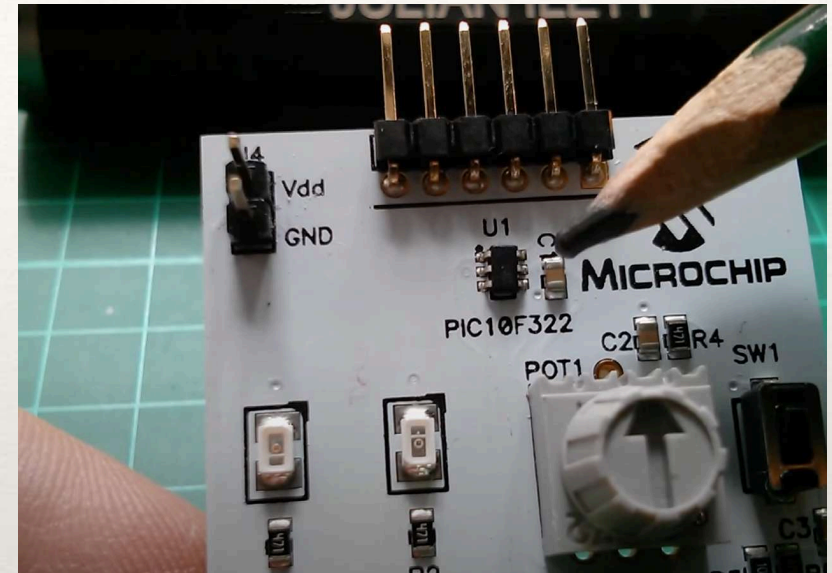
Estes são aspectos fundamentais em projetos embarcados, determinando inclusive a viabilidade, ou não, de algumas aplicações;

Efeitos

Miniaturização em larga escala não mais impõe limites das dimensões do CI associados ao número de componentes internos;

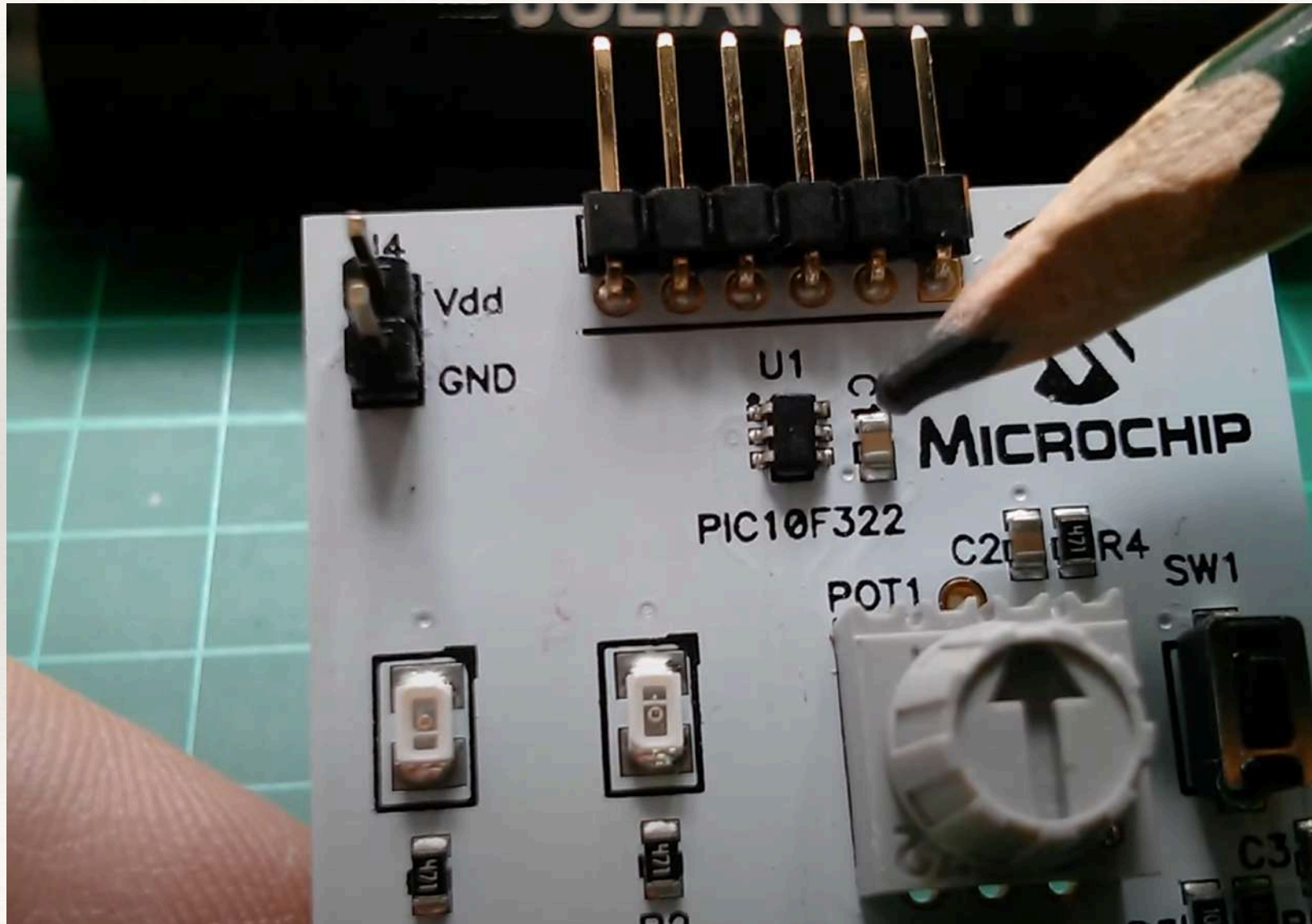
Componentes menores produzem menos calor, mas têm área menor de dissipação;

Reduzir o número de pinos afeta o tamanho significativamente, mas dificulta as interfaces, limitando as aplicações;



Placa de Protótipo PIC10F322

Tamanho (componentes)



Conexões

Outro fator importante é a forma de conexão ao circuito:

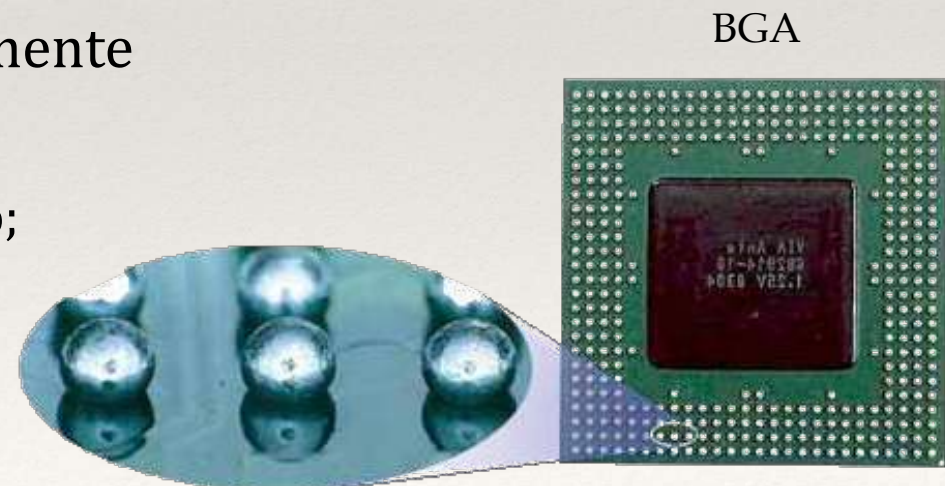
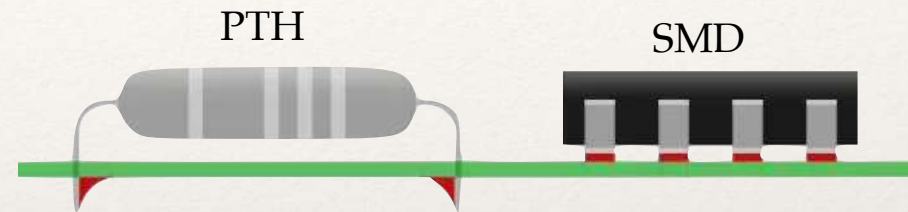
PTH - *Plated-Through Hole*

SMD - *Surface Mount Device*

Tamanho dos componentes SMD é muito menor, especialmente com encapsulamento BGA (*Ball Grid Array*) pela redução do espaço ocupado pelos pinos.

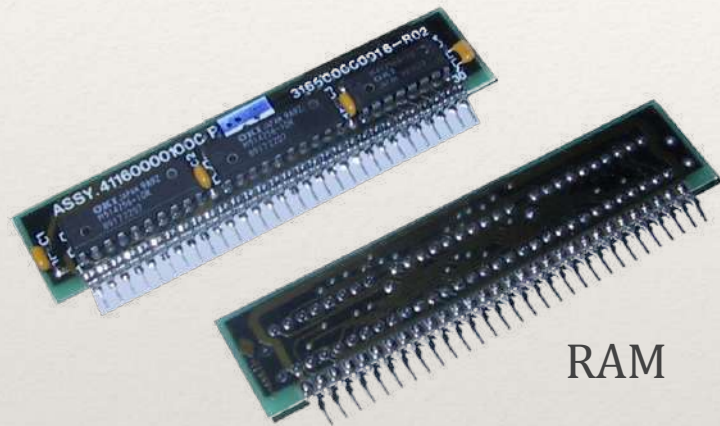
Tamanho afeta positiva ou negativamente diversos aspectos:

Dissipação de calor;	Custo de Produção;
Interferência;	Confiabilidade;
Manutenção.	



Encapsulamentos (PTH)

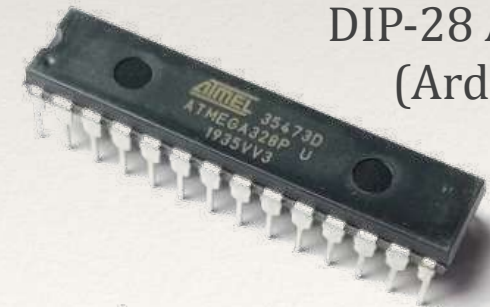
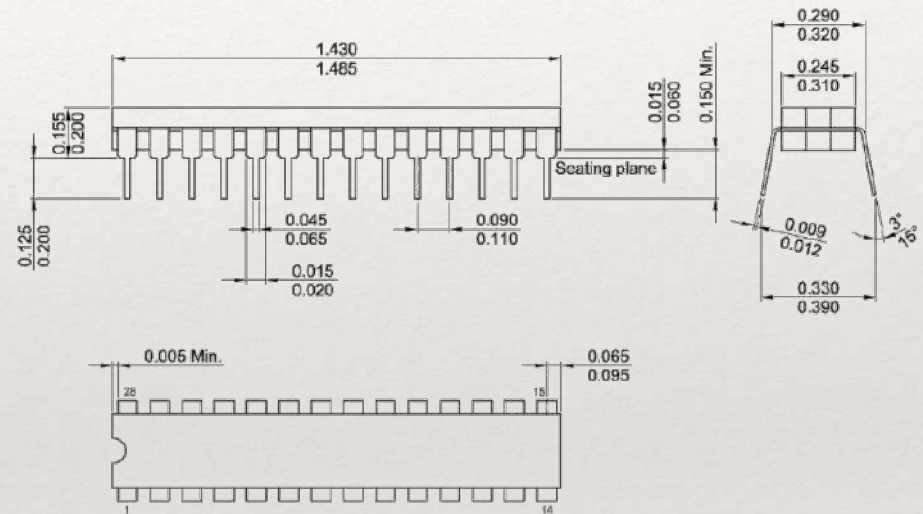
SIP - Single In-Line Package



RAM

Resistor Network

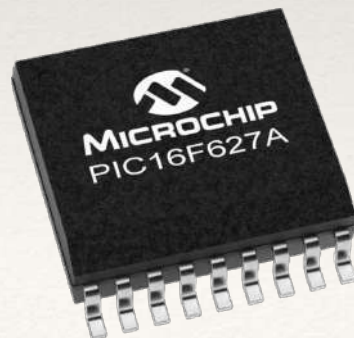
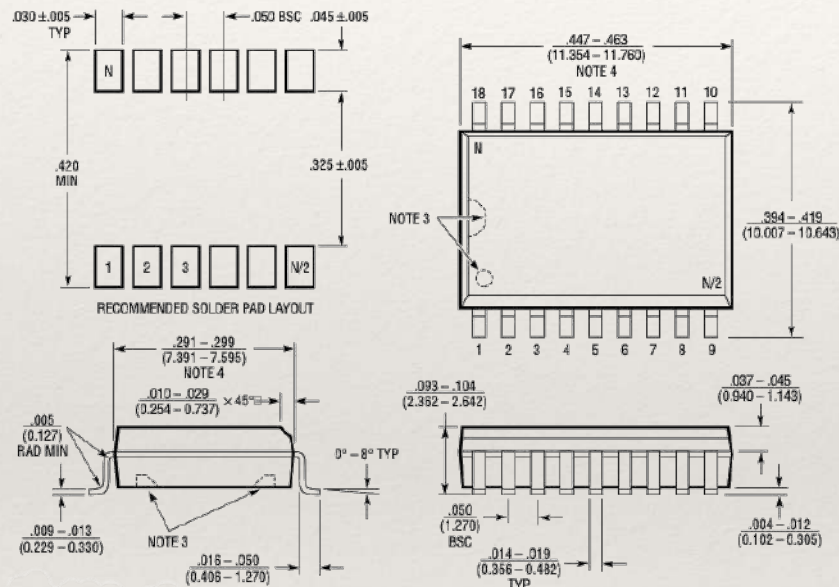
DIP - Dual In-Line Package



DIP-28 ATmega328P
(Arduíno Uno)

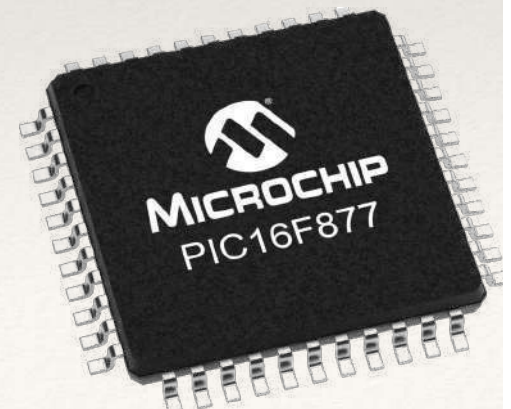
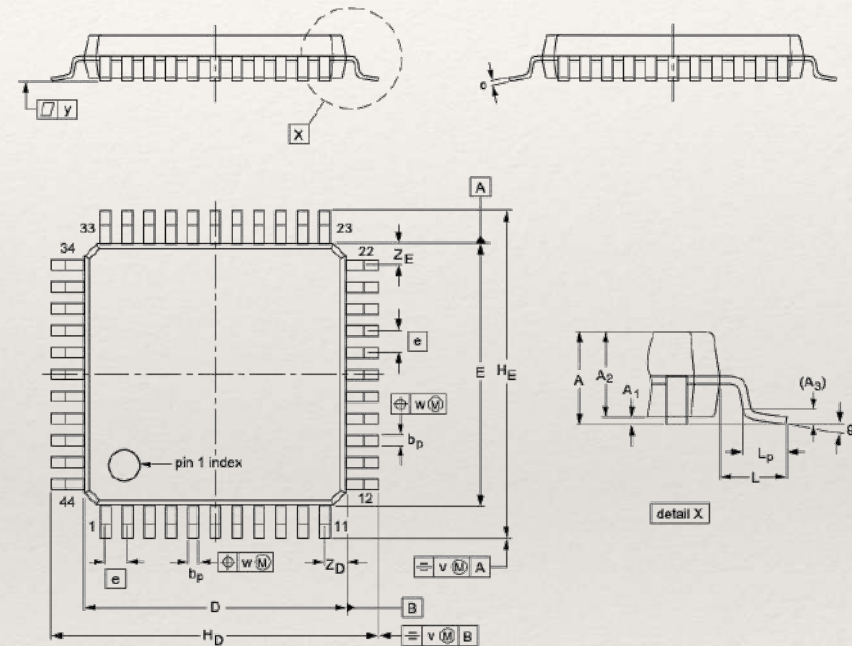
Encapsulamentos (SMD)

SOIC - Small Outline Integrated Circuit



SOIC-18
PIC16F627A

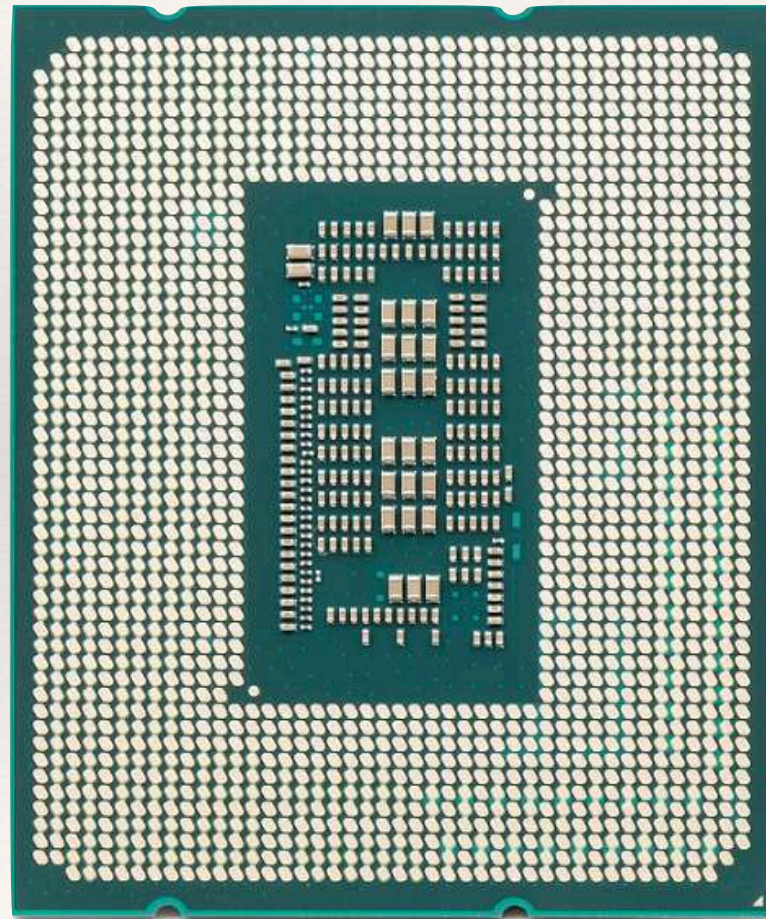
QFP - Quad Flat Package



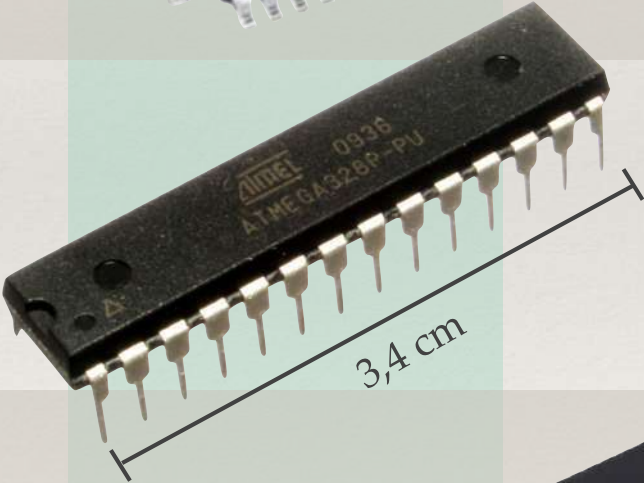
QFP-44
PIC16F877

Encapsulamentos (SMD)

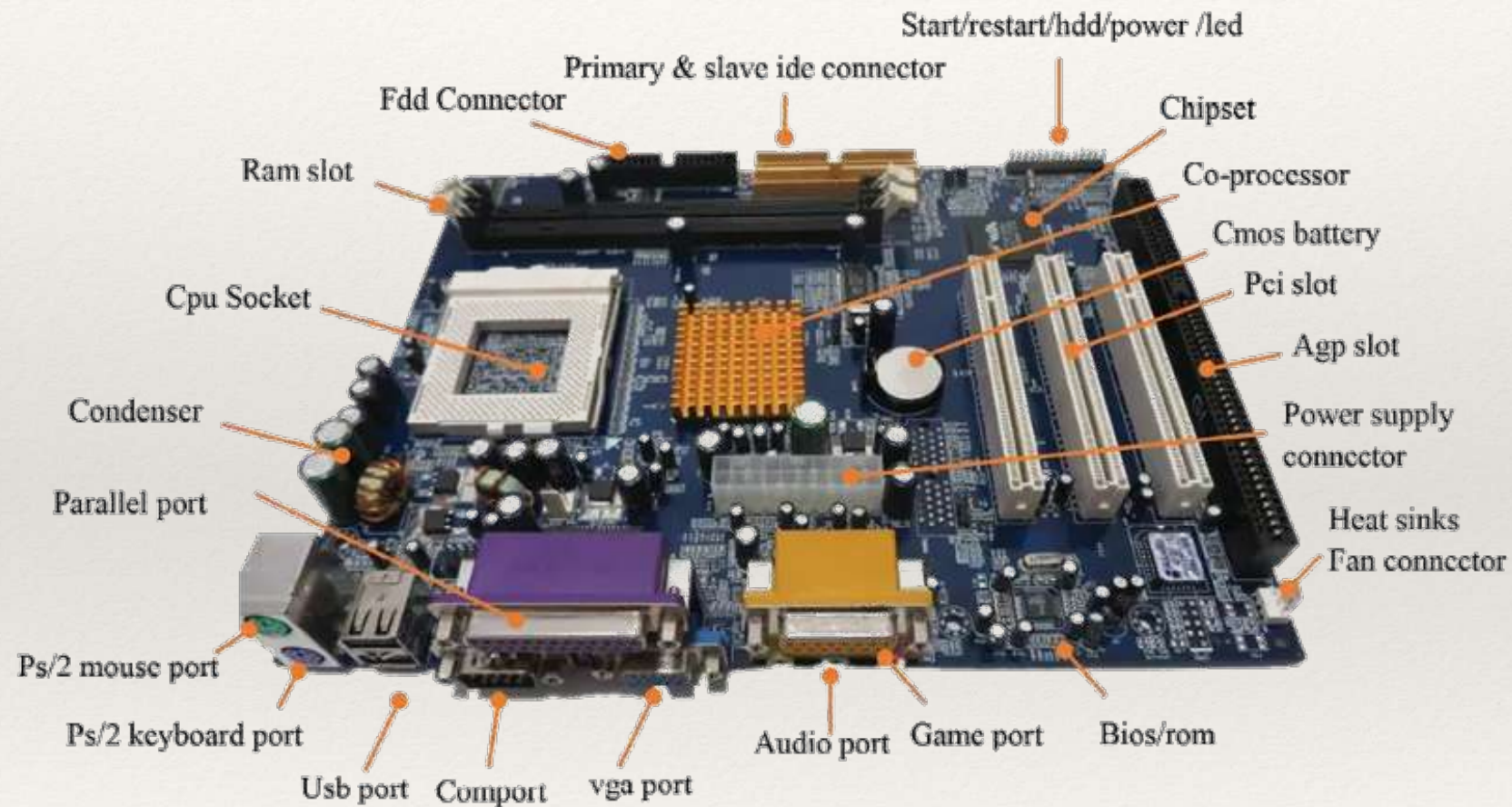
BGA - Ball Grid Array



Comparativo Dimensional

	<u>PIC10F322</u> 6 pinos	<u>ATTiny85</u> 8 pinos	<u>ATMega328P</u> 28 pinos	<u>ESP32 SoC</u> 48 pinos
SMD				
PTH				
QFN				

Dispositivos Adicionais



Dispositivos Adicionais

Um processador exige companhia !

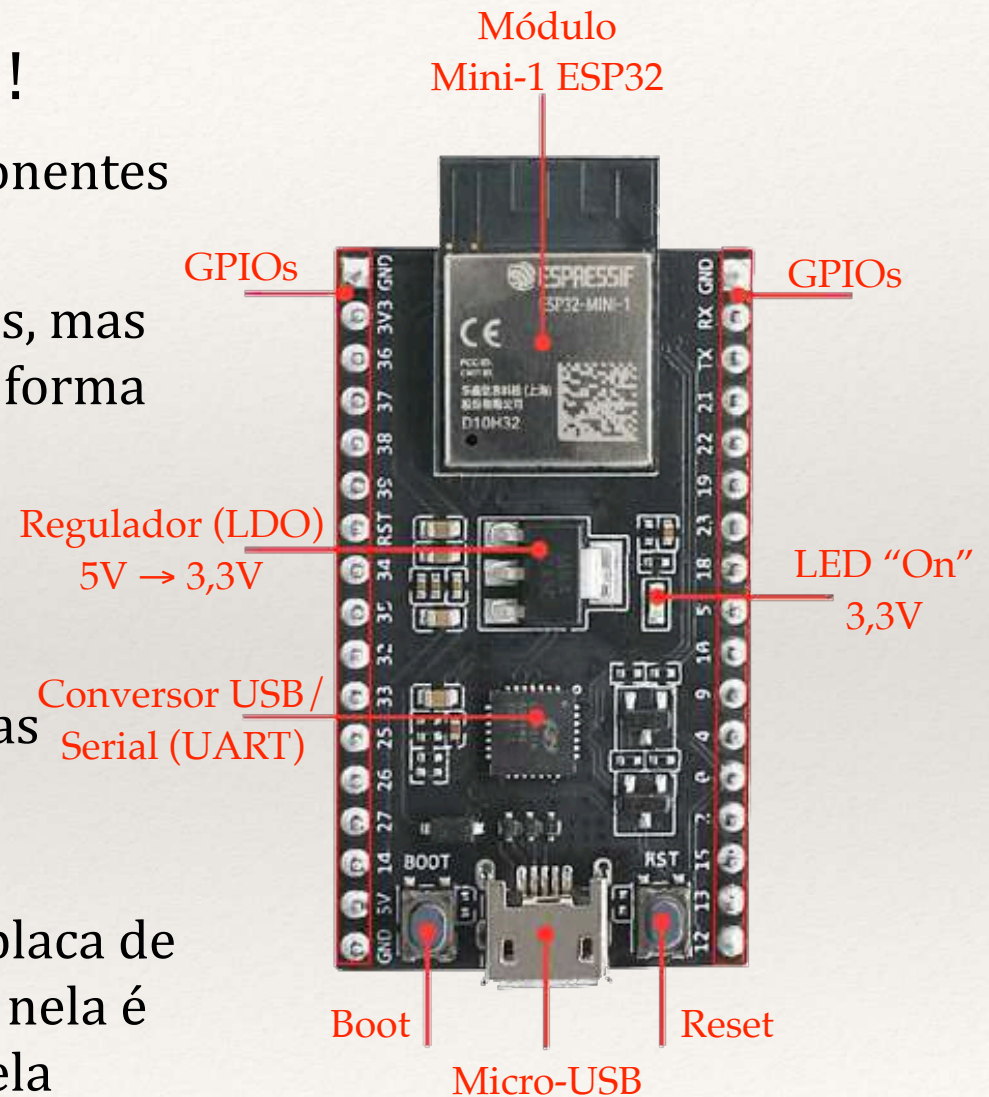
Microprocessadores precisam de componentes externos essenciais à sua operação;

Microcontroladores são menos exigentes, mas mesmo assim não podem ser usados de forma isolada.

Placas de Protótipo

Também chamadas de “kit de desenvolvimento” (development kit), elas permitem testar protótipos com menor esforço;

O Arduíno Uno v3, por exemplo, é uma placa de protótipo. O microcontrolador utilizado nela é o ATmega328P (da ATMEI, comprada pela Microchip Technology em 2016).



Placa de Protótipo EXP32

Dispositivos Adicionais

Um processador exige companhia !

Microprocessadores precisam de componentes externos essenciais à sua operação;

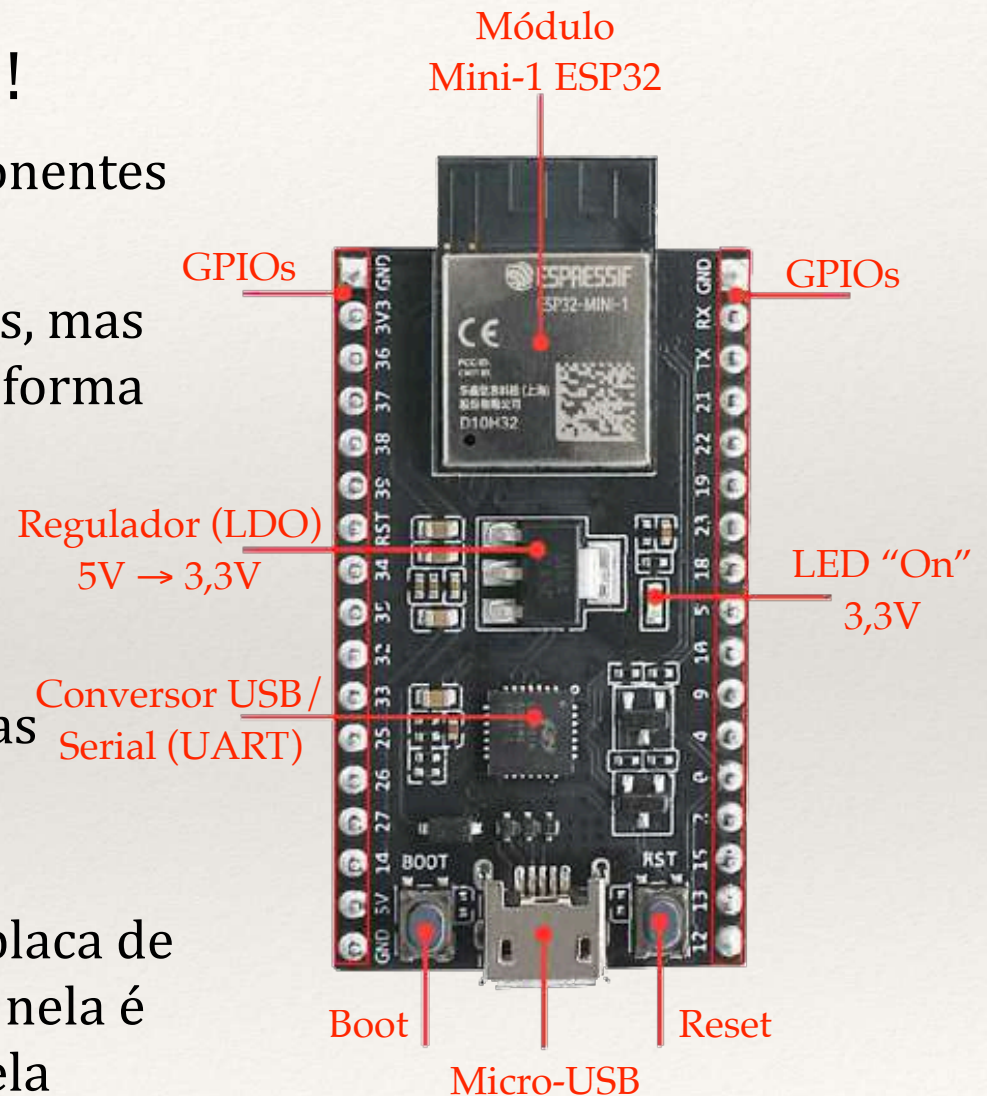
Microcontroladores são menos exigentes, mas mesmo assim não podem ser usados de forma isolada.

Placas de Protótipo

Também chamadas de “kit de desenvolvimento” (development kit), elas permitem testar protótipos com menor esforço;

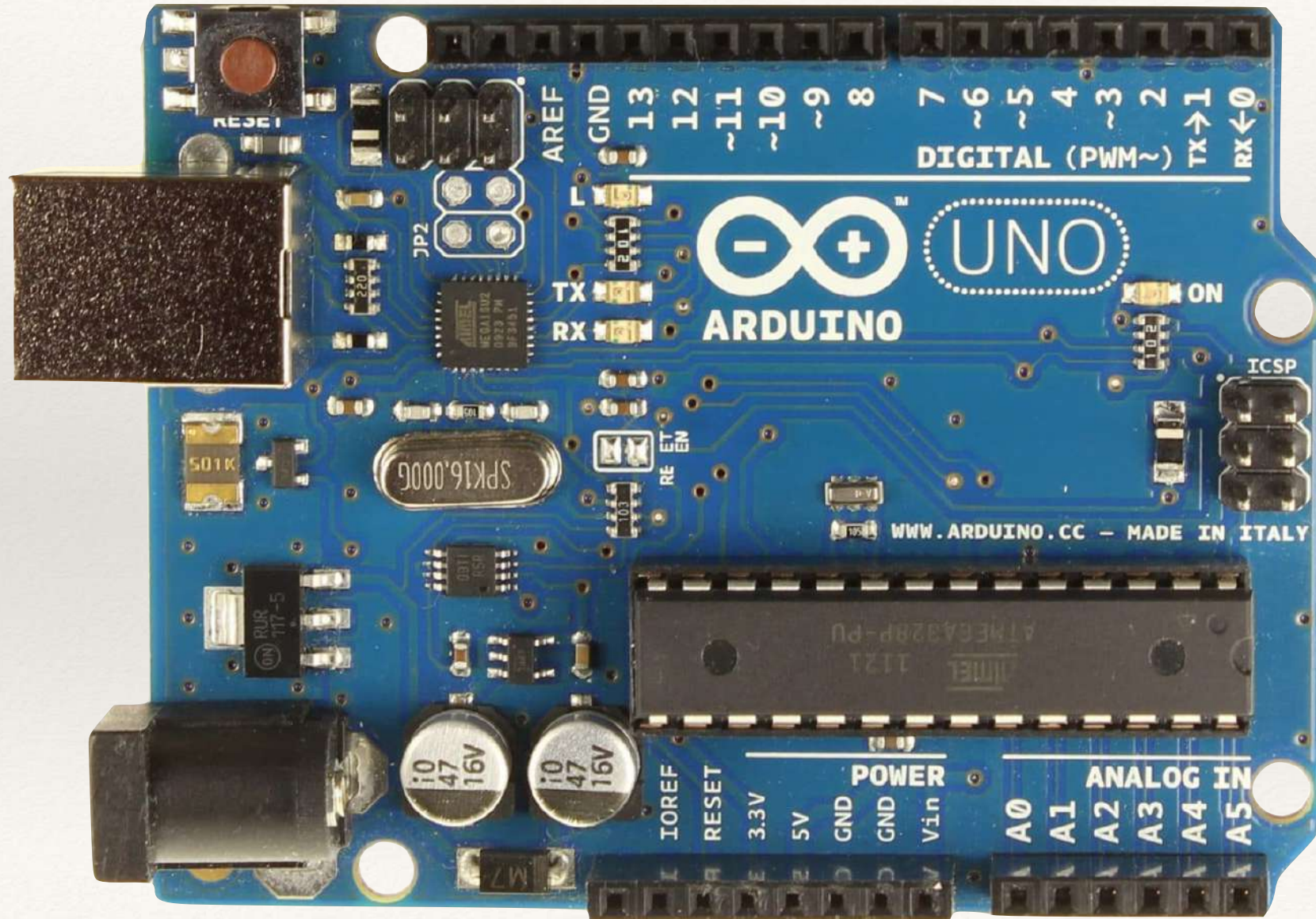
a versão 4 usa outro
O Arduino Uno ~~v3~~, por exemplo, é uma placa de protótipo. O microcontrolador utilizado nela é o ATmega328P (da ATMEI, comprada pela Microchip Technology em 2016).

Renesas RA4M1 32-bit ARM® Cortex®

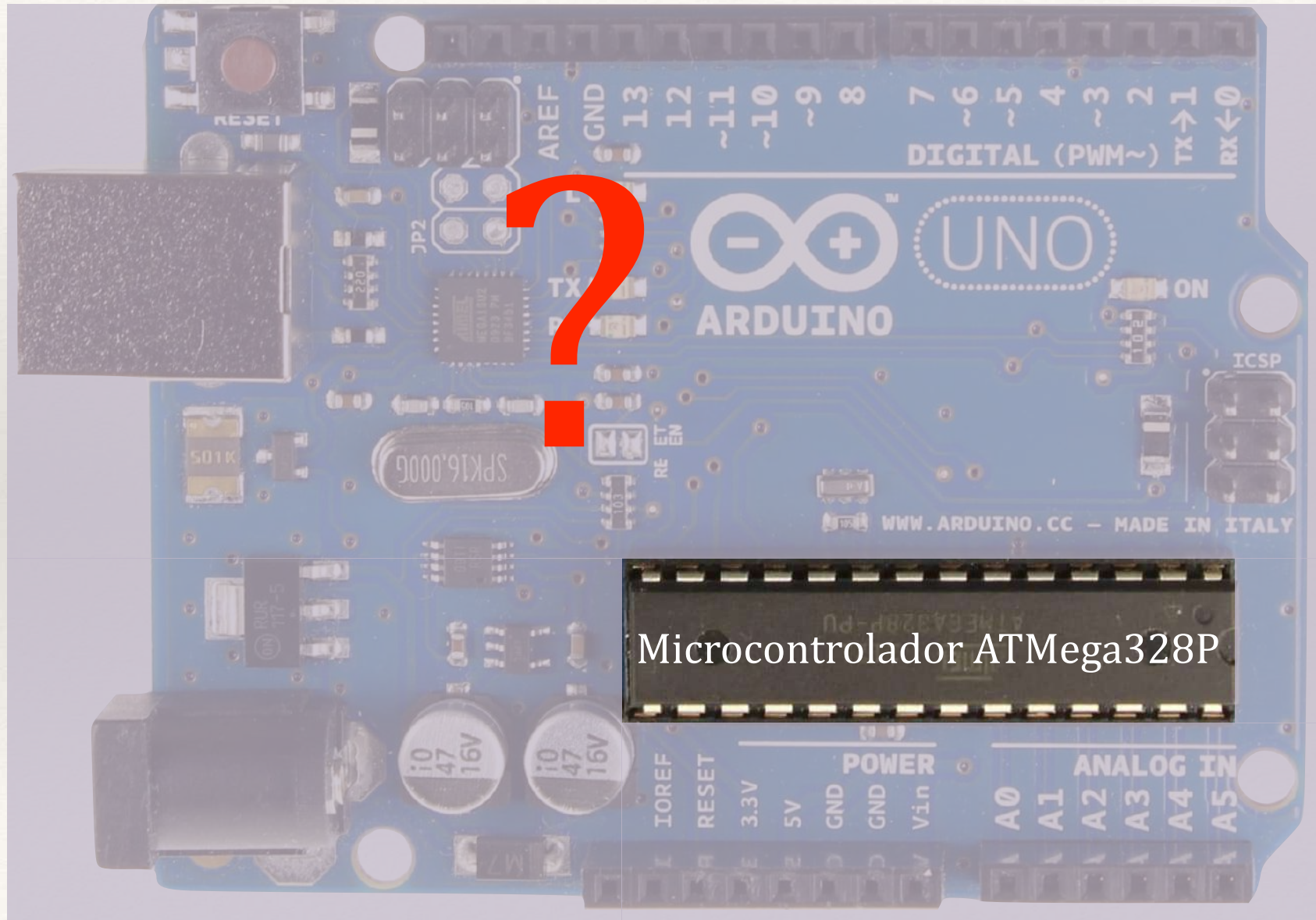


Placa de Protótipo EXP32

Conhecendo o Arduino Uno R3

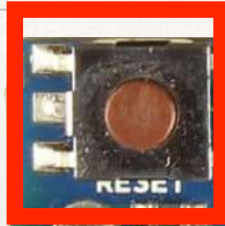


Conhecendo o Arduino Uno R3

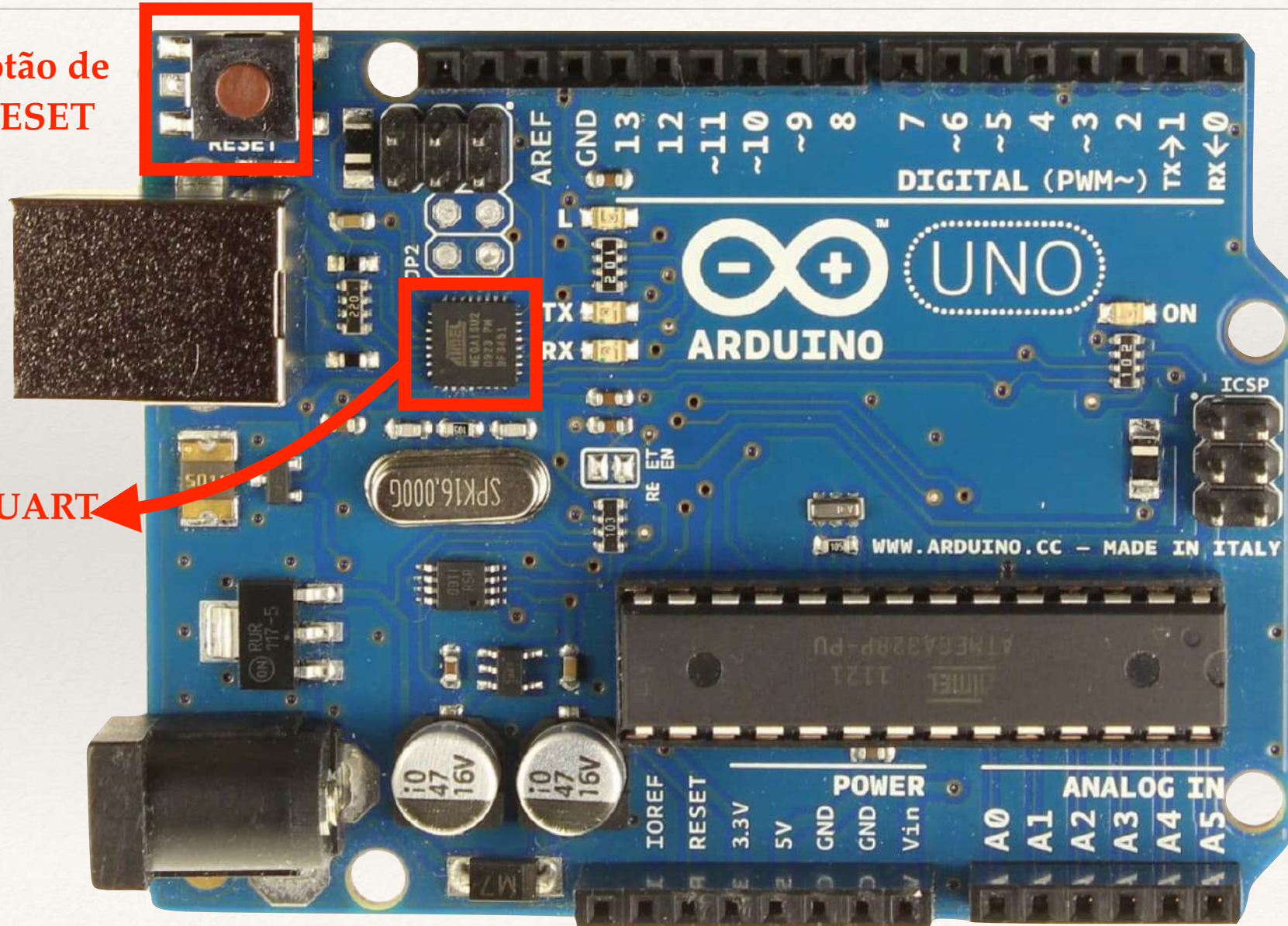


Conhecendo o Arduino Uno R3

Botão de
RESET



USB -> UART



Conhecendo o Arduino Uno R3



Cristal de Clock

O *clock* sincroniza processador, barramentos e periféricos;

Performance:

Se for baixa, pode tornar a operação lenta demais - porém reduz consumo;

Se for alta, pode impor tempos de resposta impossíveis para os componentes - porém é mais ágil;

Estabilidade:

Se não existirem operações síncronas, as exigências são baixas (1 ~ 5 %);

★ Um relógio atrasa até 14 minutos por dia com 1% de estabilidade.

Para aplicações síncronas, é um item crítico;

★ Ex: ADC, DAC e comunicação de dados.

Conhecendo o Arduino Uno R3

Existem algumas opções para implementação do *clock*:

- Oscilador RC interno;
- Oscilador RC externo;
- Oscilador com cristal de quartzo.

As opções diferem na estabilidade, tamanho, consumo e custo;

Fatores externos (ou internos) afetam a estabilidade:

Temperatura, alimentação, interferências elétricas e vibração;

O Arduino Uno R3 usa um cristal de 16MHz em seu oscilador.



Cristal de Clock

Conhecendo o Arduino Uno R3

A alimentação é crítica:

Queda de tensão por descarga da bateria, ou falhas de projeto, podem provocar alteração de comportamento;

Ruído injetado na alimentação é uma das fontes de instabilidade mais difíceis de diagnosticar. O próprio circuito digital é uma fonte de ruído;

Um surto de tensão, mesmo pequeno, pode danificar diversos componentes.

Como evitar problemas?

Escolha cuidadosa das fontes de alimentação;

Capacitores de desacoplamento individuais para cada CI, instalados bem próximos dos mesmos.

Regulador
de Tensão
(5V)



Capacitor de
Desacoplamento



Aula 03

Dispositivos Adicionais

Circuitos/*chips* específicos podem ser incluídos em um projeto

Buffers

Permite a interconexão de circuitos com performances diferentes;

Podem ser unidirecionais ou bidirecionais.

Latches

Drivers

Clock externo

Memórias

Periféricos

Dispositivos Adicionais

Circuitos/*chips* específicos podem ser incluídos em um projeto

Buffers

Latches

Agrega a capacidade de “memorizar” temporariamente o valor de saída;

São apenas unidirecionais.

Drivers

Clock externo

Memórias

Periféricos

Dispositivos Adicionais

Circuitos/*chips* específicos podem ser incluídos em um projeto

Buffers

Latches

Drivers

Compatibiliza saídas digitais de baixa potência com cargas elevadas;

Em algumas situações, agregam circuitos analógicos especializados para controle de cargas específicas, como motores *brushless*, de passo, sistemas trifásicos etc.

Clock externo

Memórias

Periféricos

Dispositivos Adicionais

Circuitos/*chips* específicos podem ser incluídos em um projeto

Buffers

Latches

Drivers

Clock externo

É muito comum a existência de circuitos internos de *clock* nos microcontroladores, mas a demanda por maior exatidão e precisão pode exigir circuitos externos de maior qualidade.

Memórias

Periféricos

Dispositivos Adicionais

Circuitos/*chips* específicos podem ser incluídos em um projeto

Buffers

Latches

Drivers

Clock externo

Memórias:

Esta é uma demanda essencial nos microprocessadores;

Não é suportada por microcontroladores, e é complexa até em SBCs como o Raspberry Pi.

Periféricos

Dispositivos Adicionais

Circuitos/*chips* específicos podem ser incluídos em um projeto

Buffers

Latches

Drivers

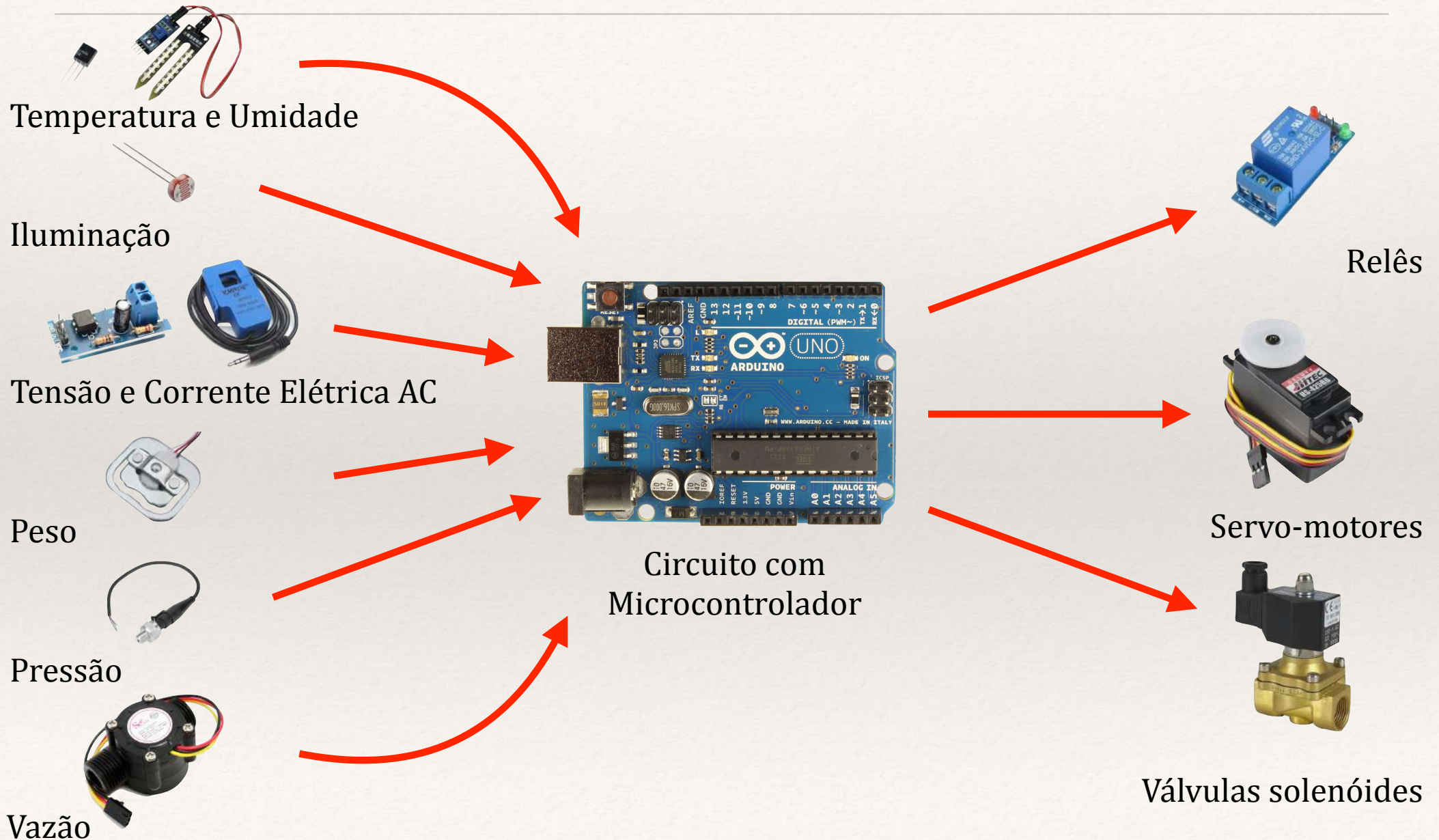
Clock externo

Memórias

Periféricos:

Até mesmo em placas de protótipo às vezes são oferecidos RPCs, interfaces de rede, sensores de temperatura, transmissores de rádio, displays etc.

Sensores e Atuadores



Computação Física

A tensão elétrica dos sinais trocados com o mundo externo codificam a informação transferida

Portas digitais **ESCREVEM E LEEM** sinais codificados digitalmente

Tensão alta (HIGH) = 1;

Tensão baixa (LOW) = 0;

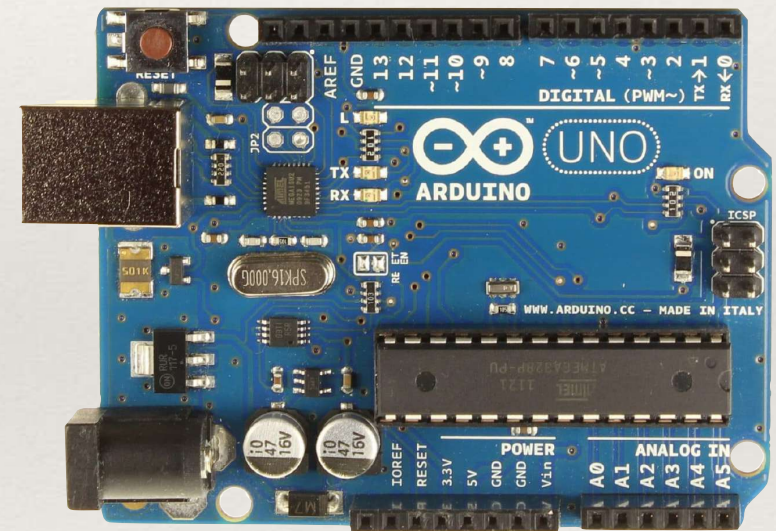
Portas analógicas **LEEM** sinais elétricos codificados analogicamente

O valor da tensão indica o valor analógico;

Teoricamente, existem "infinitos" valores;

Na prática não ! Por quê?

Sinais Elétricos
(valores de tensão digitais)



Sinais Elétricos
(valores de tensão analógicos)

E/S Analógicas X Digitais

Grandezas Digitais e Analógicas precisam ser processadas constantemente;

Para manipular informações analógicas, há dois desafios:

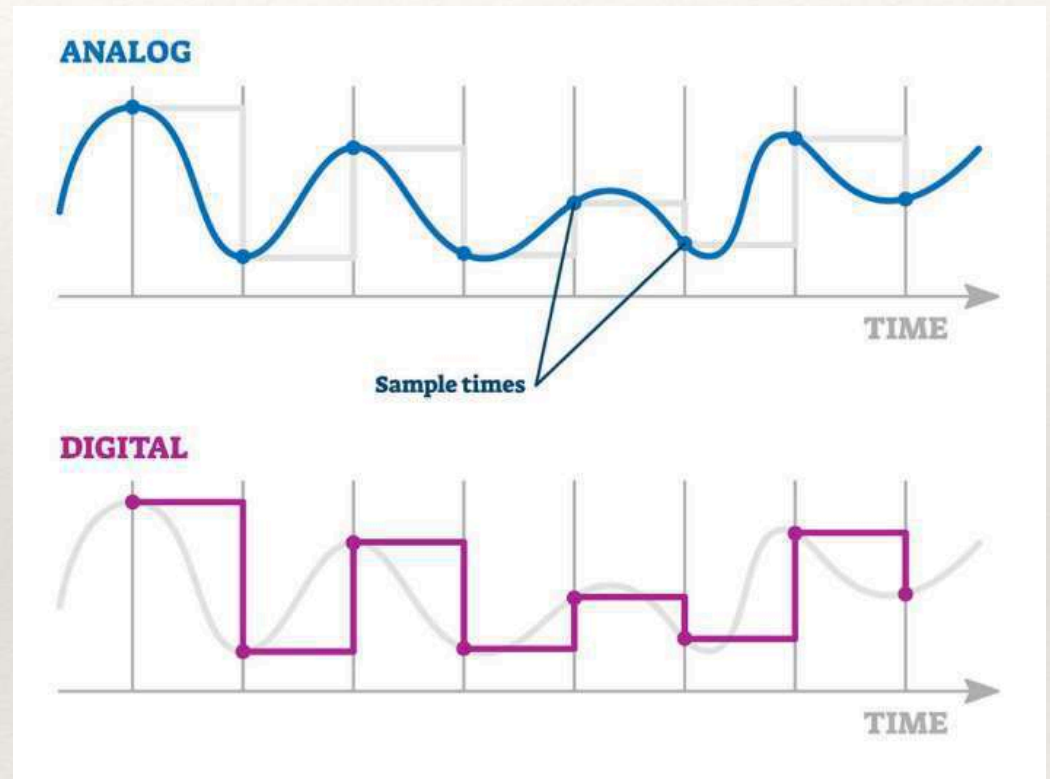
Leitura:

ADCs (*Analog to Digital Converters*);

Escrita:

DACs (*Digital to Analog Converters*);

PWM (*Pulse Width Modulation*).



ADC: como funciona?

Arquitetura Paralela

Um divisor resistivo divide a tensão de referência máxima em intervalos;

As saídas do divisor resistivo são comparadas com o valor da entrada analógica através de “n” comparadores, gerando uma tabela verdade com “n” saídas;

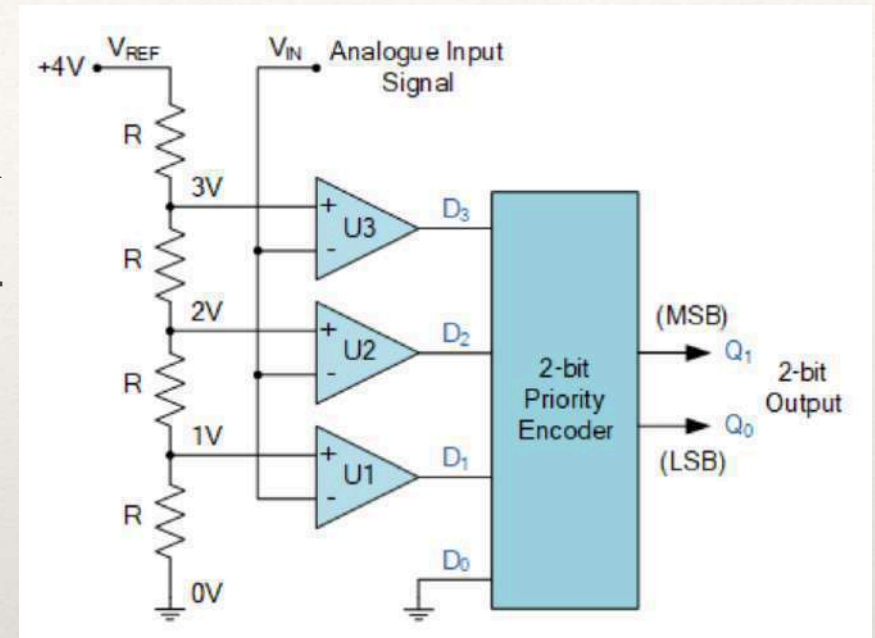
As “n” saídas são convertidas para o número binário correspondente.

A arquitetura é rápida (tempo real), mas sua complexidade é muito grande para resoluções maiores;

Outras arquiteturas são mais adequadas para resoluções mais altas

Pipeline, SAR, Sigma-Delta

Estas arquiteturas geram retardo, e são muito dependentes da linearidade, ruído, exatidão e precisão.



Analogue Input Voltage (V_{IN})	Comparator Outputs				Digital Outputs	
	D ₃	D ₂	D ₁	D ₀	Q ₁	Q ₀
0 to 1 V	0	0	0	0	0	0
1 to 2 V	0	0	1	X	0	1
2 to 3 V	0	1	X	X	1	0
3 to 4 V	1	X	X	X	1	1

ADC: como funciona?

Arquitetura Pa

Um divisor res
em intervalos;

As saídas do di
da entrada ana
gerando uma t

As "n" saídas s
correspondent

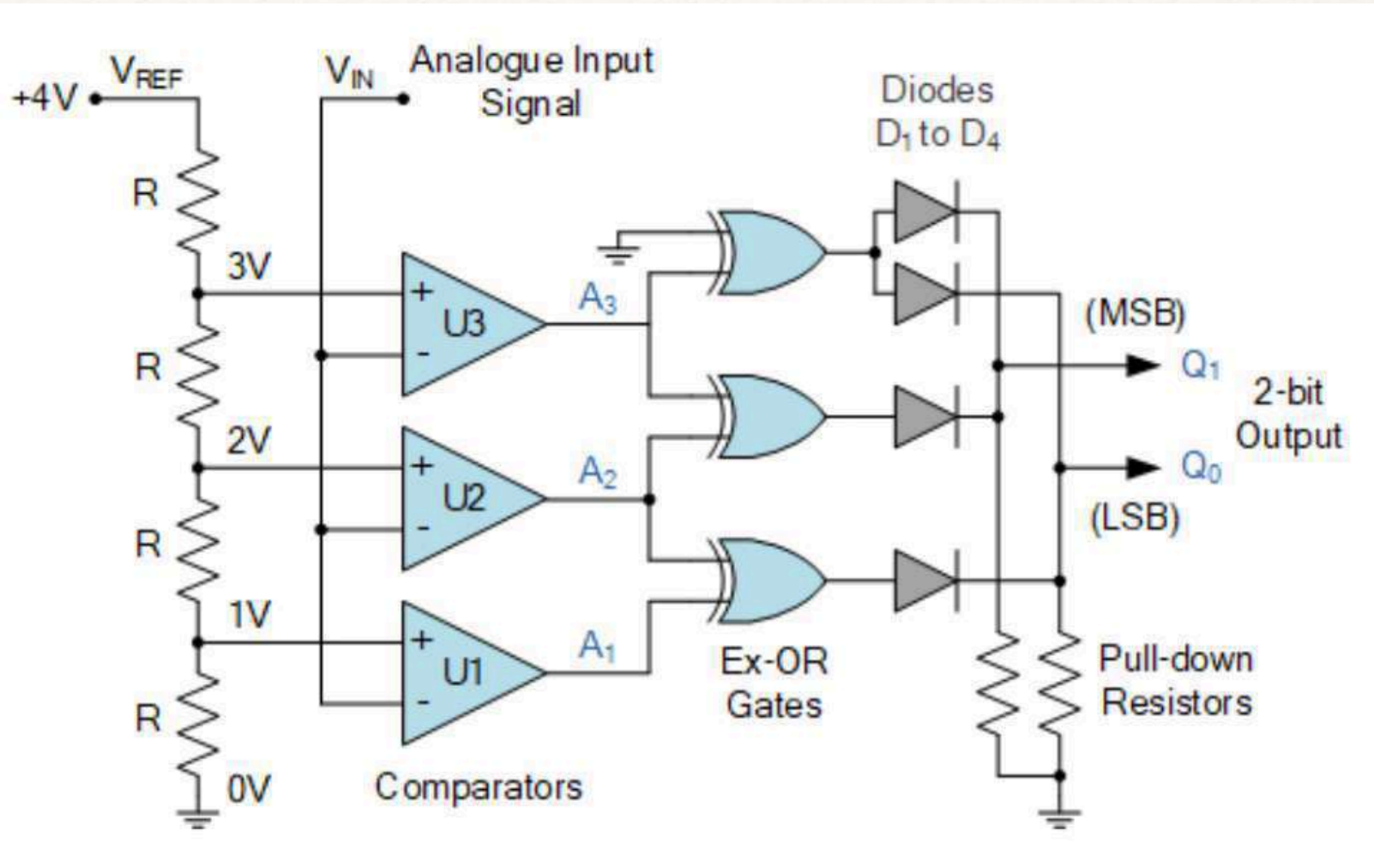
A arquitetura é
complexidade
maiores;

Outras arquite
resoluções ma

Pipeline, SAR, S

Estas arquite

dependentes da linearidade, ruído, exatidão e precisão.



ADC: como funciona?

Arquitetura Paralela

Um divisor resistivo divide a tensão de referência máxima em intervalos;

As saídas do divisor resistivo são comparadas com o valor da entrada analógica através de “n” comparadores, gerando uma tabela verdade com “n” saídas;

As “n” saídas são convertidas para o número binário correspondente.

A arquitetura é rápida (tempo real), mas sua complexidade é muito grande para resoluções maiores;

Outras arquiteturas são mais adequadas para resoluções mais altas

Pipeline, SAR, Sigma-Delta

Estas arquiteturas geram retardo, e são muito dependentes da linearidade, ruído, exatidão e precisão.

Amostragem

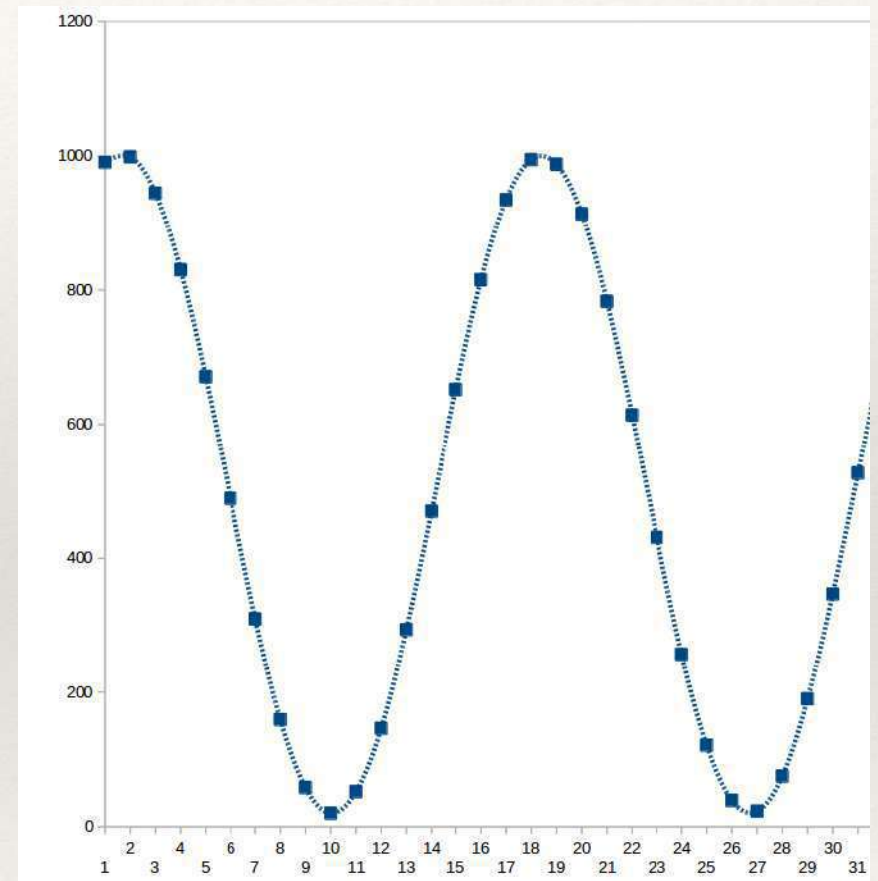
As medições são realizadas em um determinado momento, onde o valor medido é registrado;

A quantidade de medições em um determinado período é chamada de taxa de amostragem

É necessária uma frequência de amostragem mínima para conseguirmos efetivamente representar um determinado sinal, e suas variações com o tempo;

O teorema de Nyquist diz que, para uma amostragem infinita, uma frequência de amostragem pelo menos 2 vezes superior à maior componente de frequência do sinal é capaz de representar adequadamente o sinal original;

Sinais digitais, devido às suas harmônicas, podem ter frequências de amostragem ideais muito elevadas.



A frequência de amostragem está relacionada à performance do ADC.

Amostragem

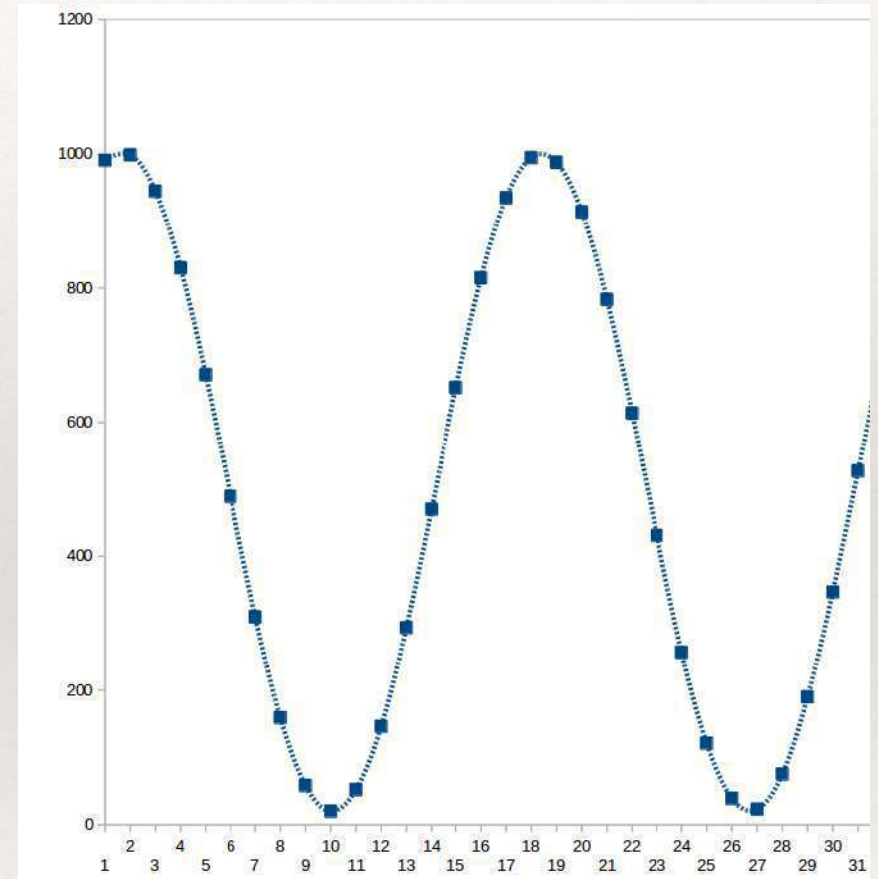
A taxa de amostragem está associada à aplicação

Se você quer ler um valor fixo, ou com variação lenta, a taxa pode ser menor;

Se você deseja analisar o comportamento de um sinal que varia rapidamente, a taxa precisa ser maior;

Um voltímetro tem taxa típica de 4 SA/s (sem considerar cálculo de média);

Um osciloscópio tem taxa típica de 1 GSA/s (você não vai conseguir fazer isto com um ATmega328P, por exemplo).



O ADC do ATmega328P

Hardware do ADC

Uma amostragem é, na verdade, uma comparação;

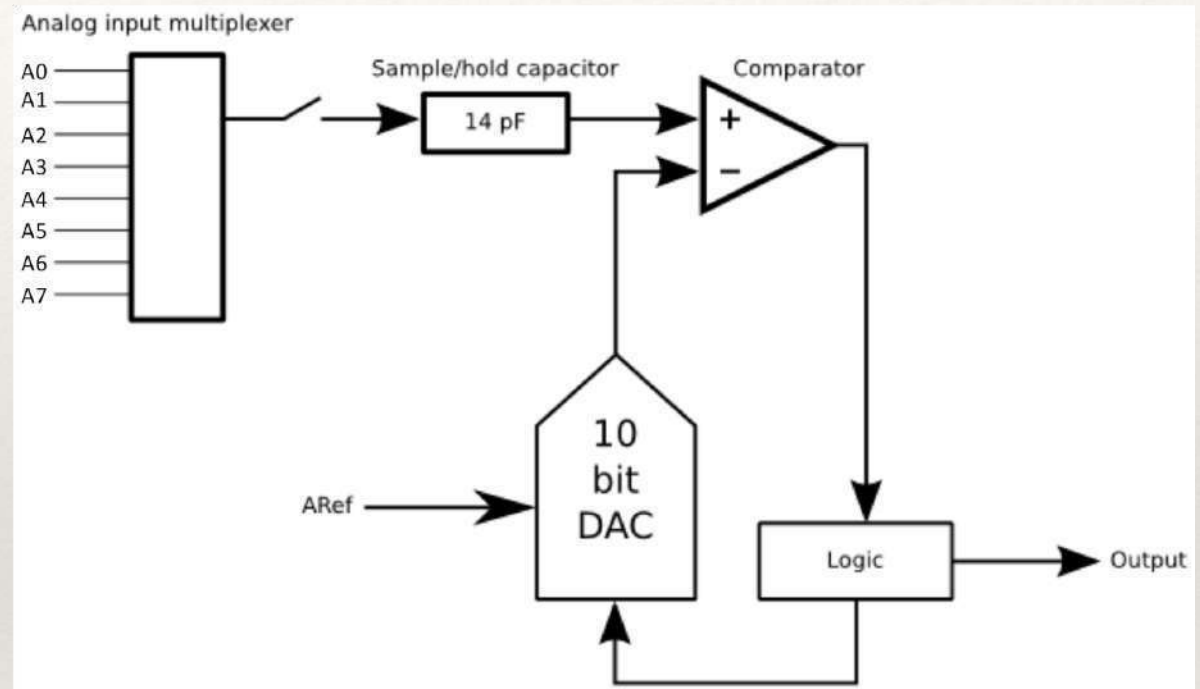
Capacitor para "filtro de ruído analógico";

O filtro é mais eficiente para ruídos de frequência mais alta.

Multiplexação de 8 diferentes entradas analógicas

Poderíamos ter mais?

Tudo é uma questão de performance e aplicação.

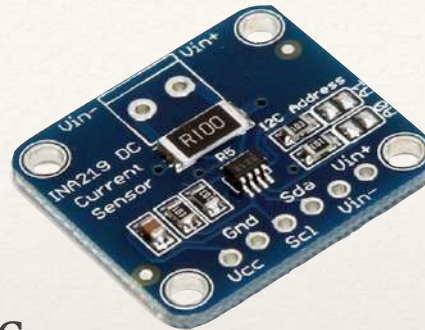


ADC externo

INA219 - 12 bits com interface i2C

Até 128 SA/s;

Leitor de Tensão e Corrente.



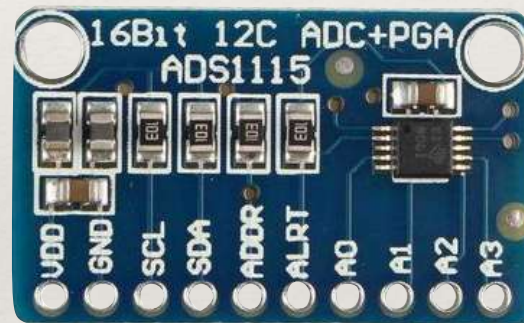
ADS1115 - 16 bits com interface i2C

Até 860 SA/s;

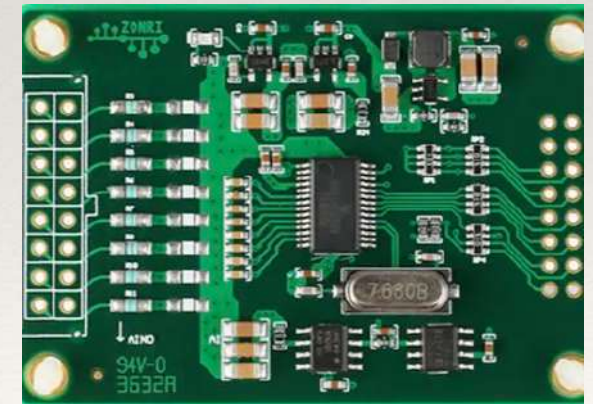
4 canais de entrada ou ...

2 canais diferenciais;

Amplificador de ganho programável.



ADS1252 - 24 bits com interface SPI

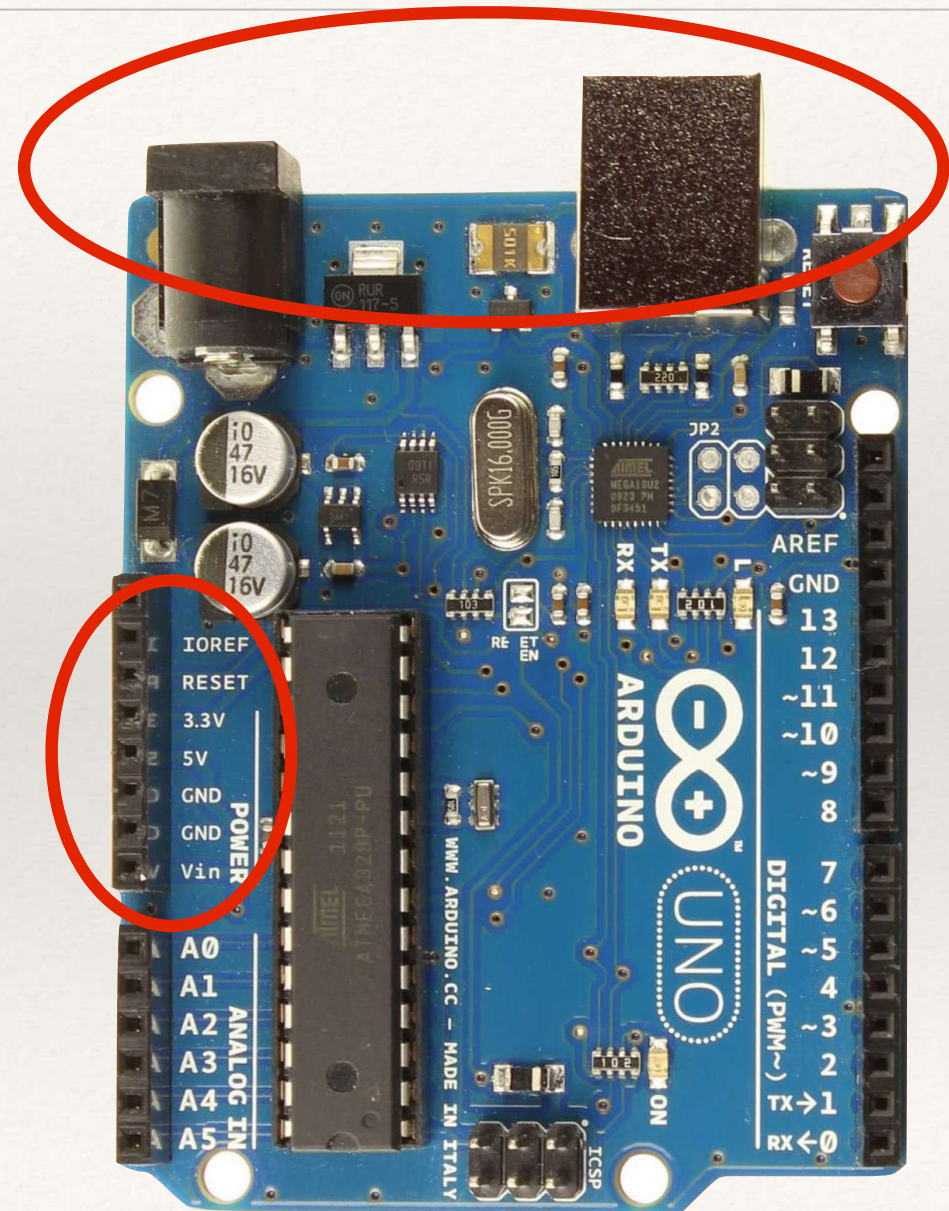


Portas do Arduino Uno

Alimentação

Portas Analógicas

Portas Digitais

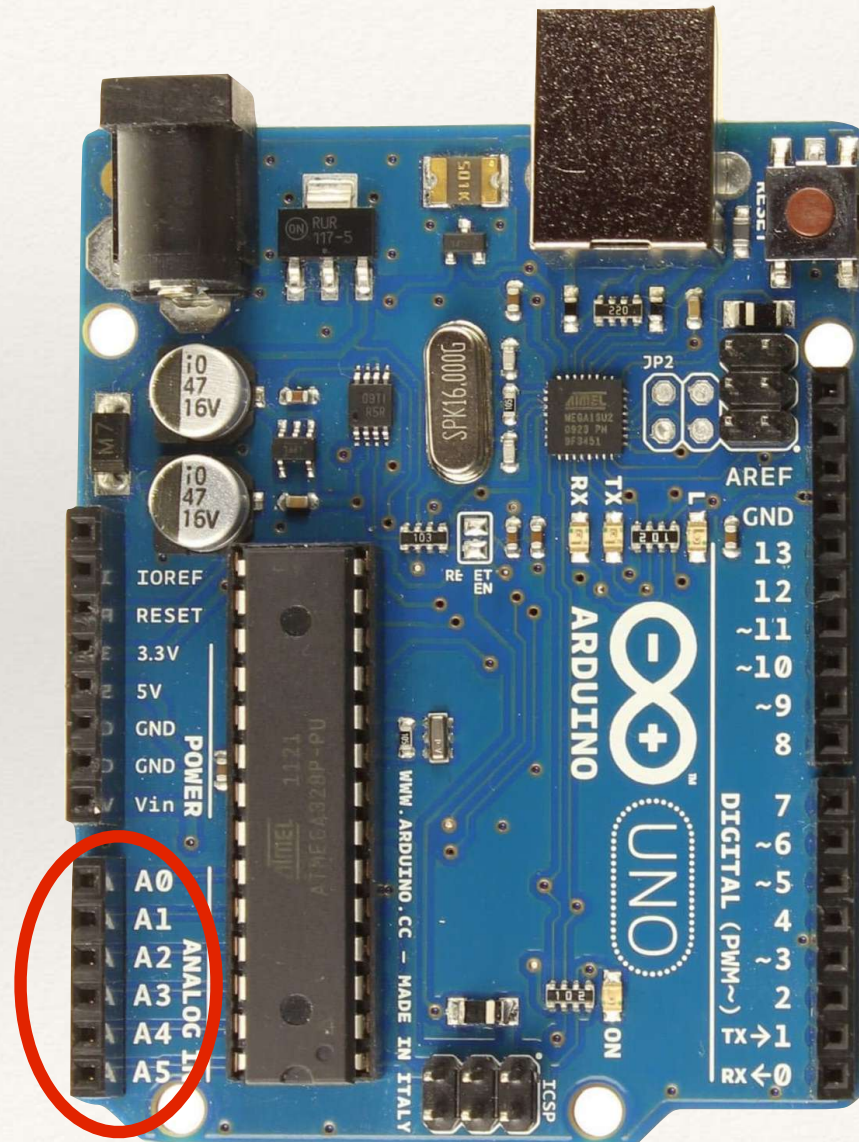


Portas do Arduino Uno

Alimentação

Portas Analógicas

Portas Digitais

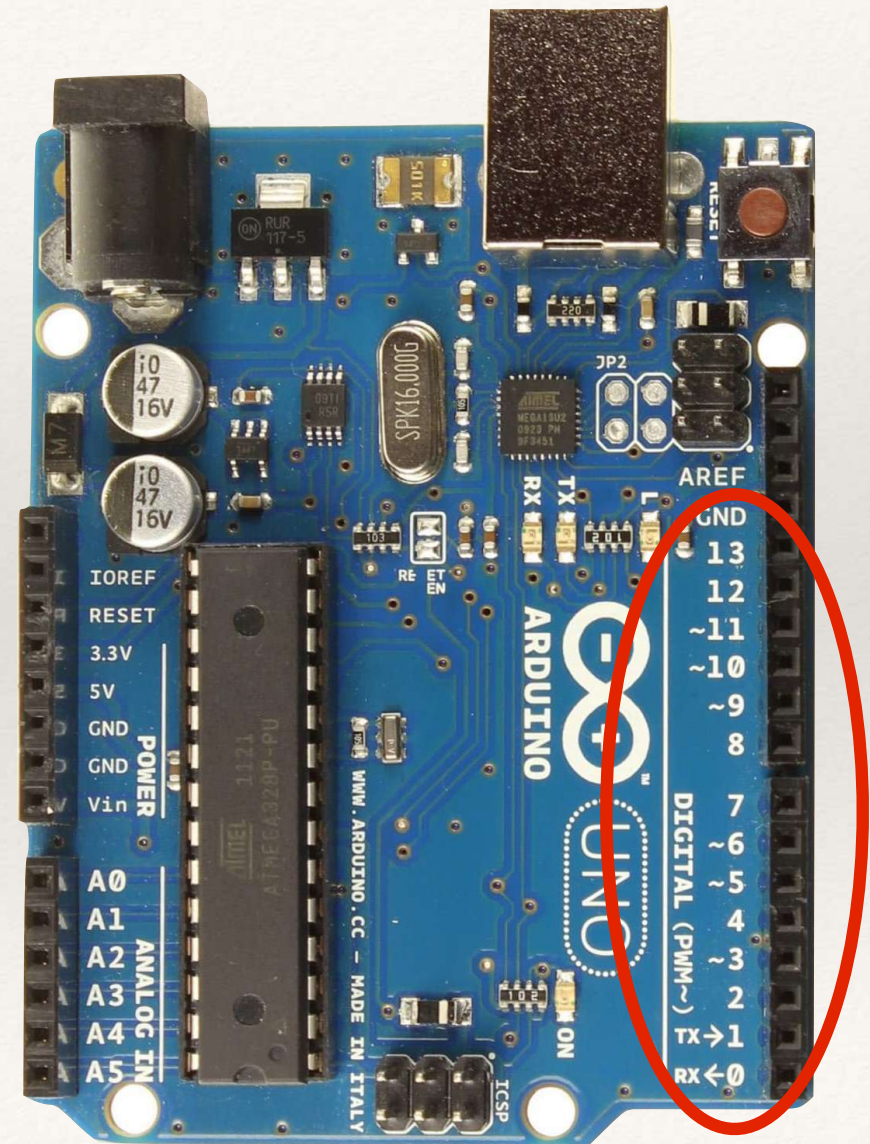


Portas do Arduino Uno

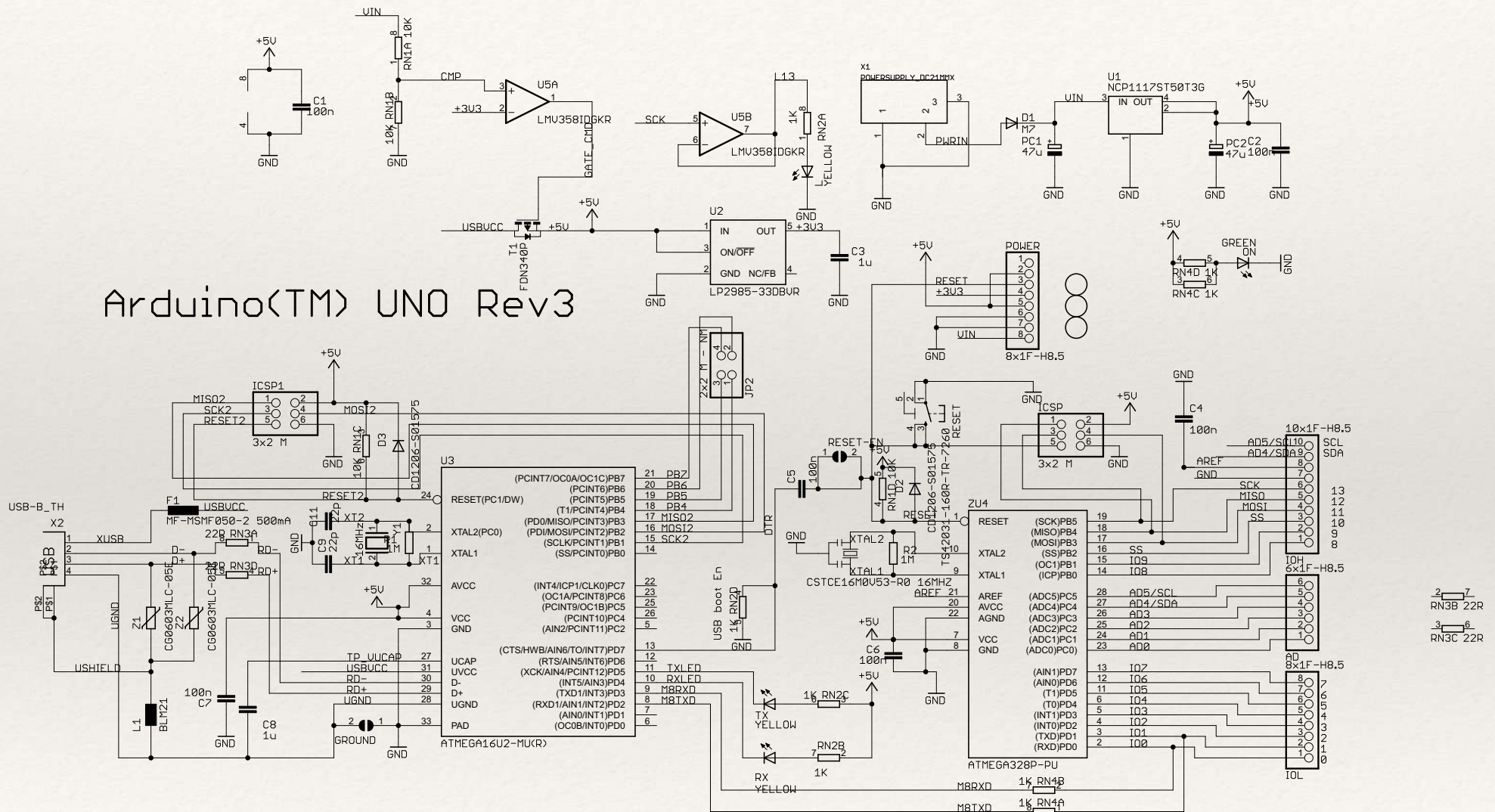
Alimentação

Portas Analógicas

Portas Digitais



Portas do Arduino Uno



Conexões do Arduino Uno R3

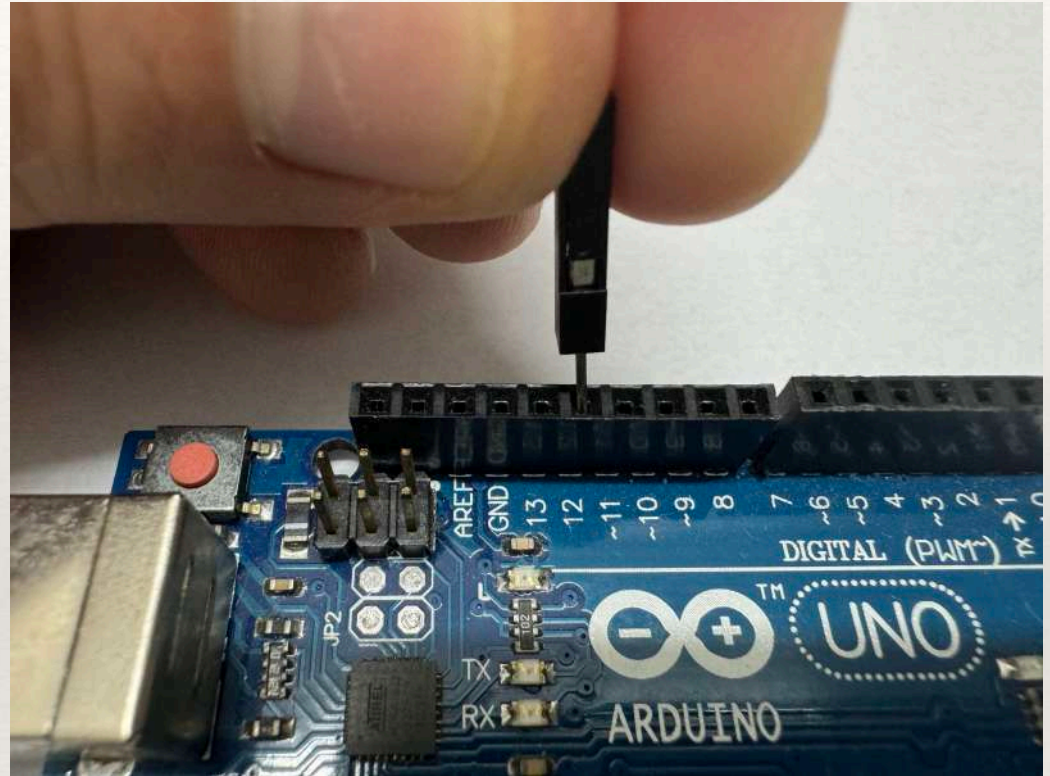
O Uno oferece conectores fêmea para fios *jumper Dupont*, sem uso de solda ou mesmo uma *protoboard*.

Embora simples, estas conexões têm alguns problemas:

- Contatos instáveis, especialmente para sistemas mais complexos;

- Baixa condutividade;

- Suportam correntes pequenas.



Por outro lado, outras placas de protótipo podem exigir o uso de *protoboards*. Projetos complexos podem exigir inclusive uma placa de circuito impresso, mesmo para protótipos.

Portas de Alimentação

Alim. / Comunicação

Entrada USB

Provê comunicação com o computador (e alimentação).

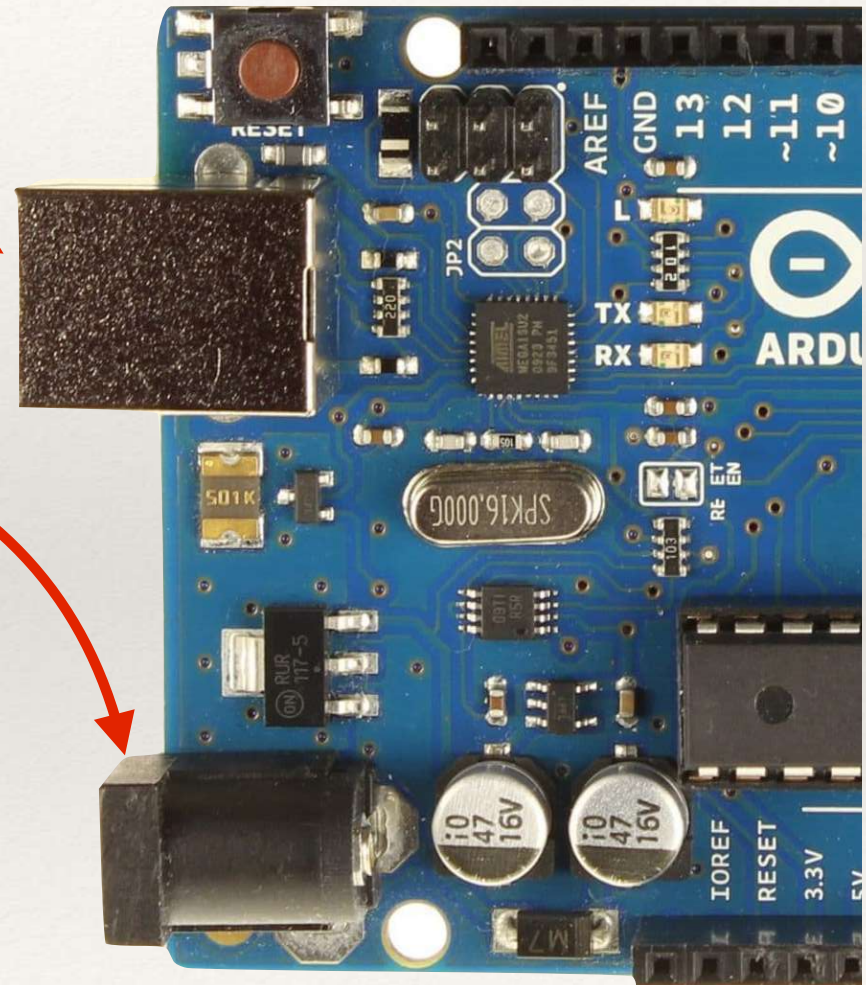
Entrada Fonte DC Externa

Provê alimentação a partir de fonte de 5 a 17V, <100mA;

Não se recomenda >12V (aquecimento componentes).

Portas Analógicas

Portas Digitais



Portas de Alimentação

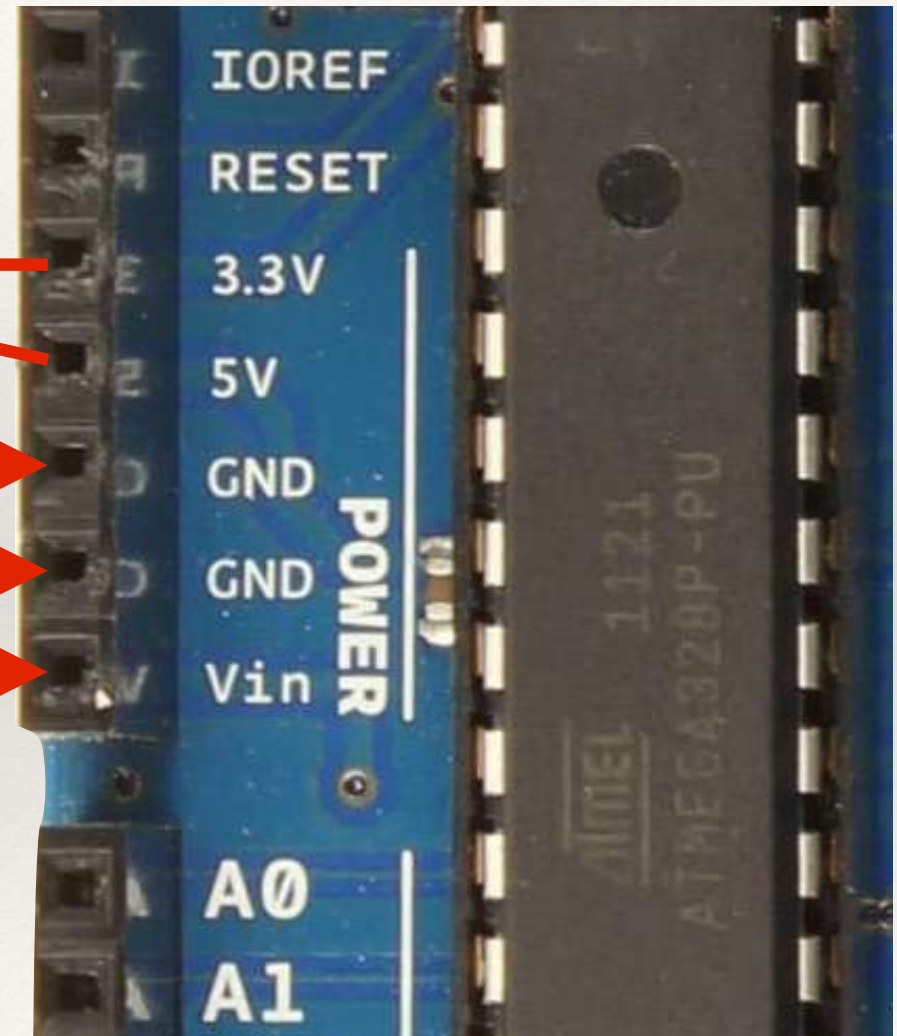
Alimentação (pinos)

Saídas de 5V e 3,3V
Corrente máxima de 40mA

Entrada Vin
Alimentação de 5V $\geq 100\text{mA}$

Portas Analógicas

Portas Digitais



Portas Analógicas

Alimentação

Portas Analógicas

Entradas com conversor A/D

Valores de 10 bits (0~1023)

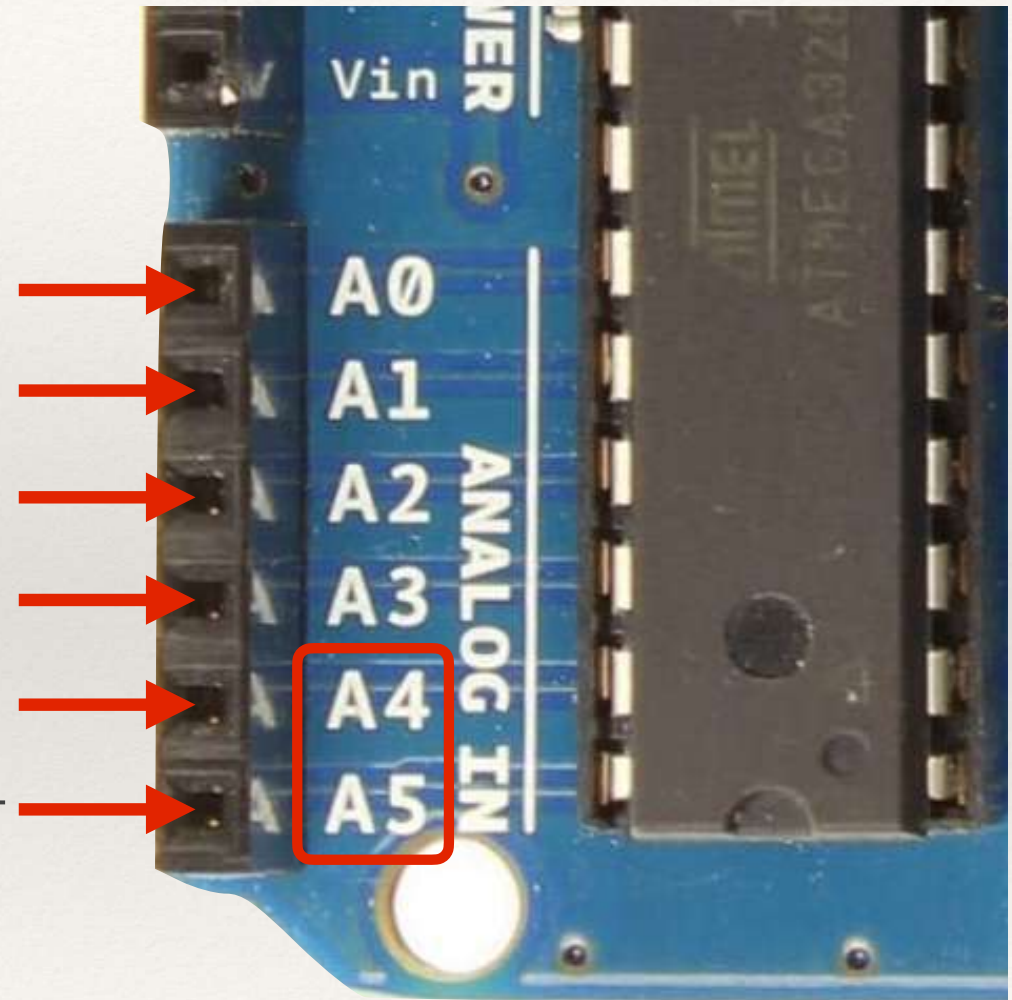
0 = **0V**

1023 = **5V** (IOREF reduz valor máximo)

Funções Especiais

I2C (SCL e SDA): portas A5 e A4

Portas Digitais



Lendo uma tensão elétrica

A tensão elétrica deve ser injetada em uma porta analógica do microcontrolador

Tensão elétrica precisa estar dentro dos limites do microcontrolador;

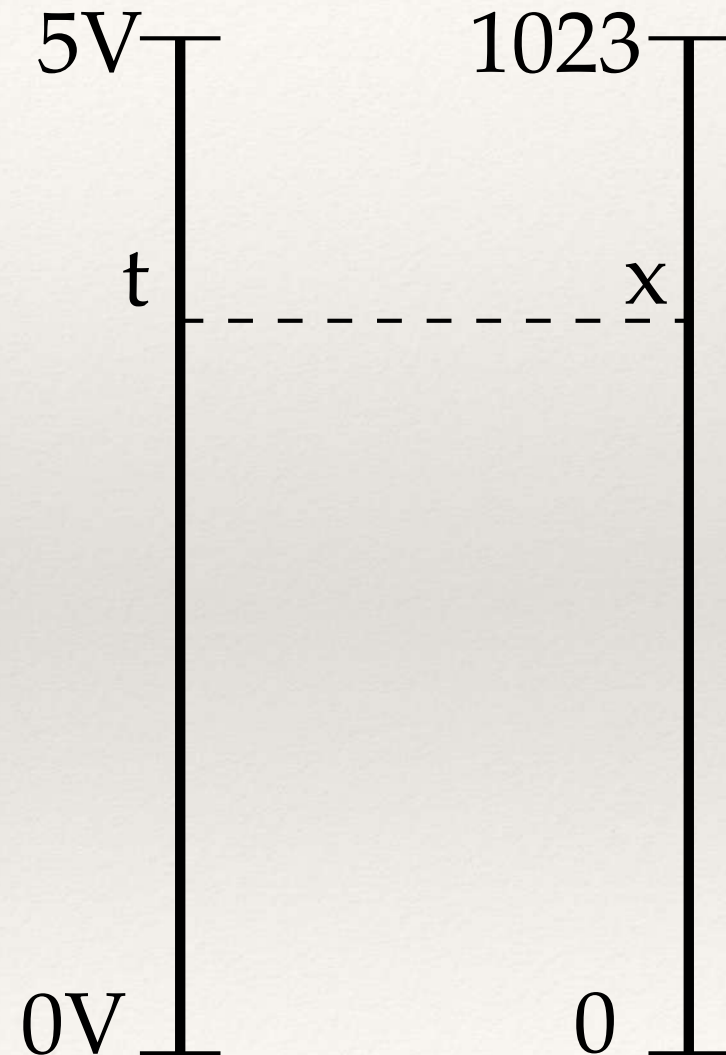
No Arduíno, pode estar entre 0 e 5V.

Na porta analógica, a tensão é convertida para um número inteiro pelo conversor analógico/digital interno;

Número é determinado pela quantidade de bits do conversor analógico digital

No Arduíno, o conversor é de 10bits

$$2^{10} = 1024$$



Tensão → Valor Binário

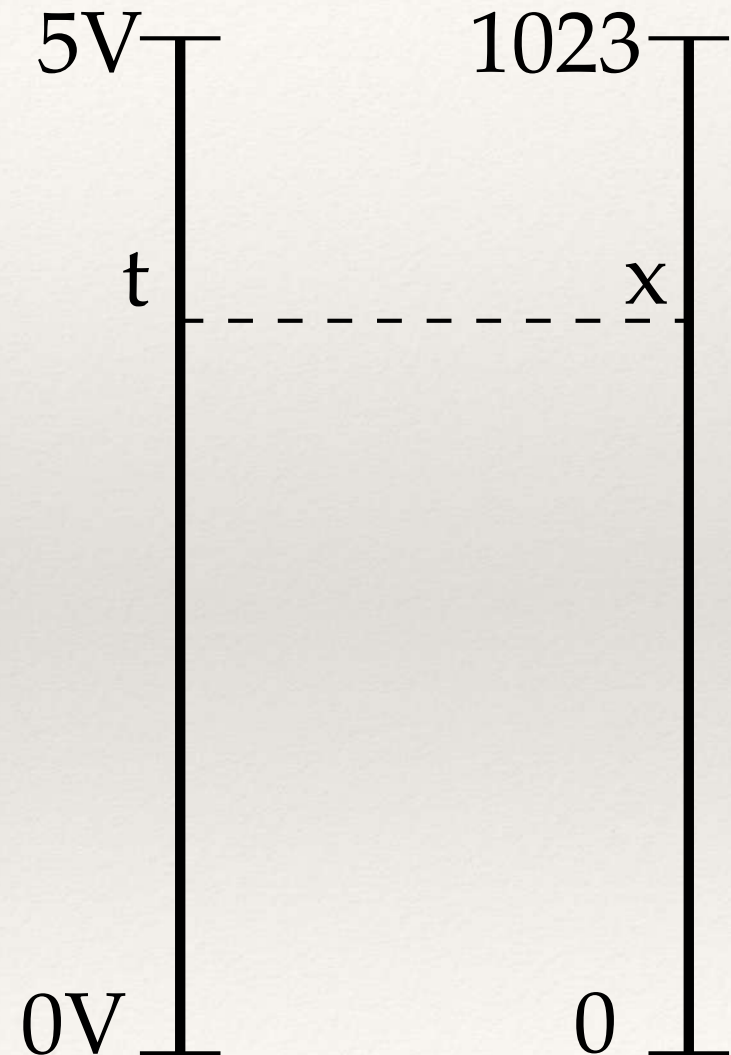
Medições são proporcionais graças à linearidade do sensor;

Conversão é básica, por regra de 3

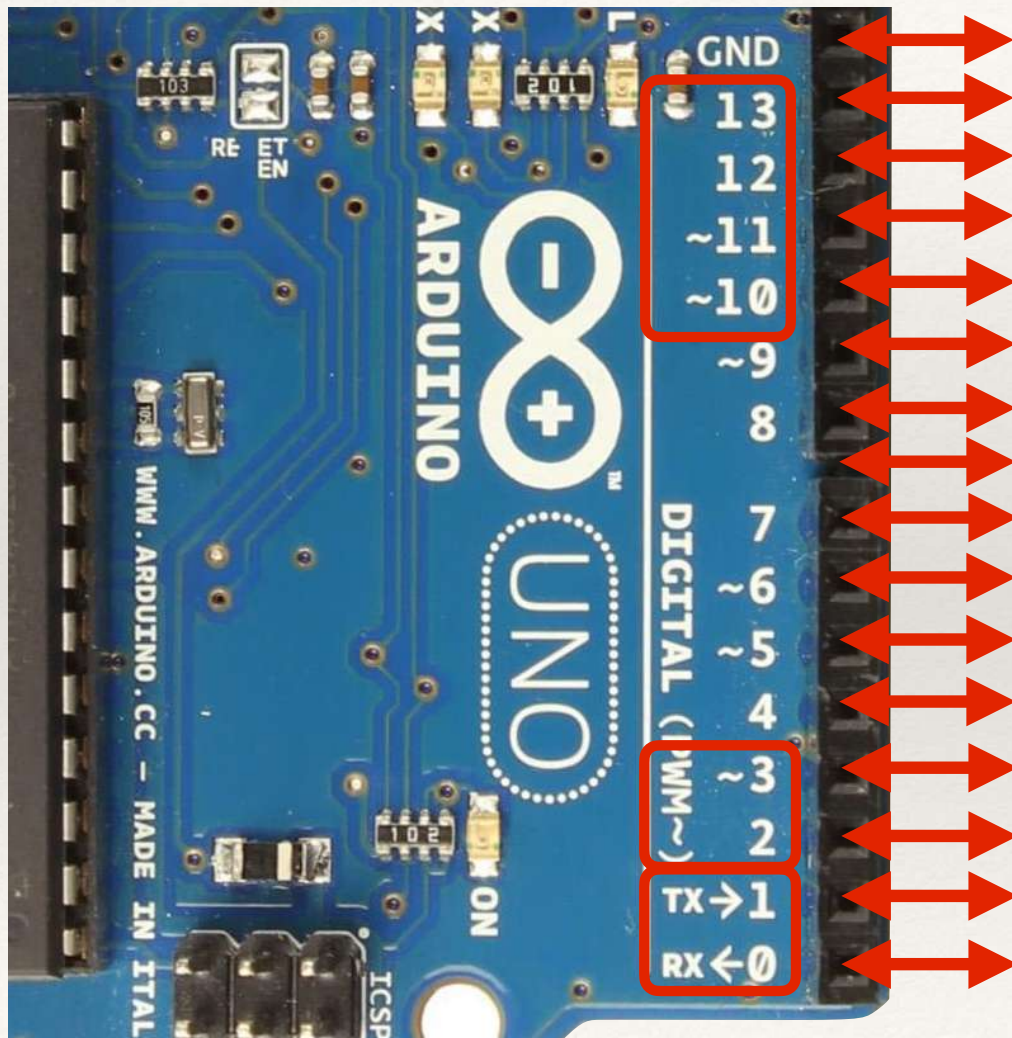
$$\frac{t - 0}{5 - 0} = \frac{x - 0}{1023 - 0}$$

$$\frac{t}{5} = \frac{x}{1023}$$

$$t = \frac{5 \cdot x}{1023}$$



Portas Digitais



Alimentação

Portas Analógicas

Portas Digitais

Entrada ou Saída

Configurada via SW

Saídas marcadas com ~ são PWM
(*Pulse Width Modulation*)

< 20mA (máx.40 mA); total 200mA

Funções especiais

UART: Rx/Tx (portas 0 e 1)

Interrupções: INT0 e 1 (portas 2 e 3)

SPI: SS,MOSI,MISO,SCK(p. 10~13)

Pino 13: LED interno

Prática 1 - Leitura Analógica

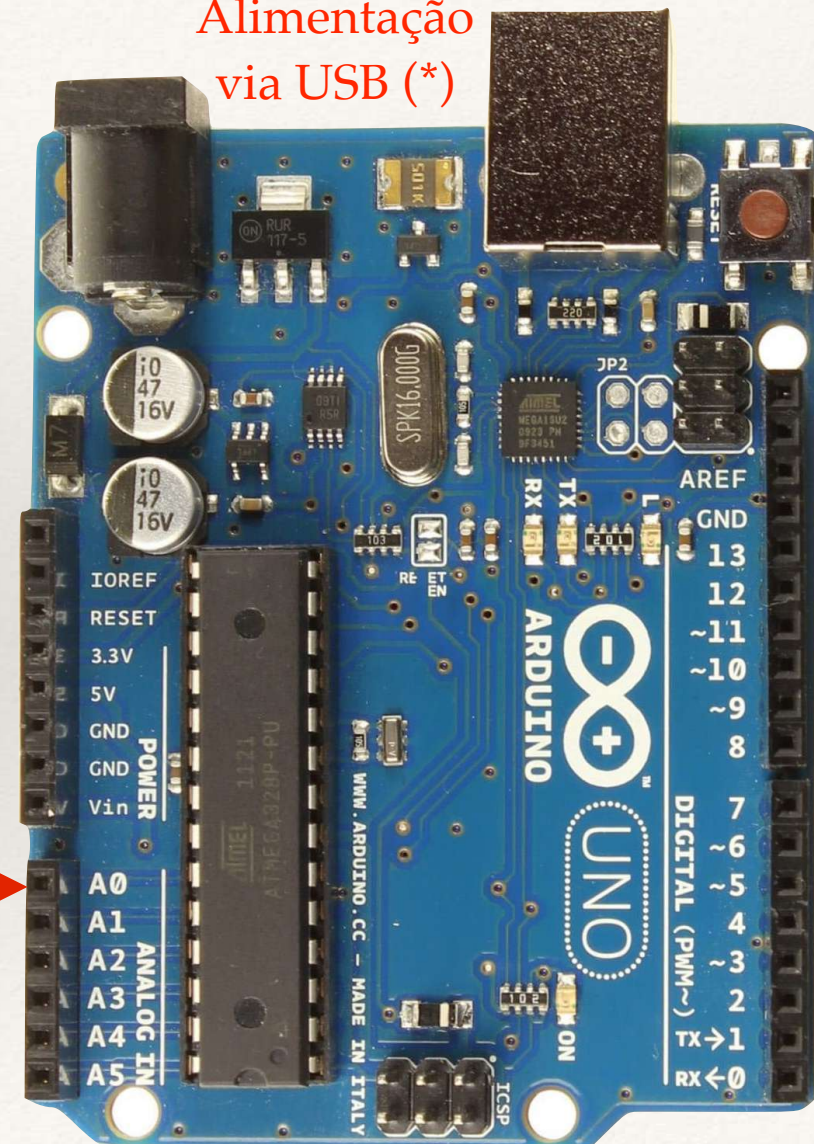
Medição de Tensões Disponíveis no próprio Arduino

Sem componentes externos;

Avaliar as medições com base no valor numérico medido pelo ADC;

Entrada a ser medida →

Alimentação
via USB (*)



(*) Sempre tome MUITO cuidado !

Prática 1 - Leitura Analógica

Circuito básico, com apenas um Arduino e um cabo

Simplicidade, sem equip. externos.

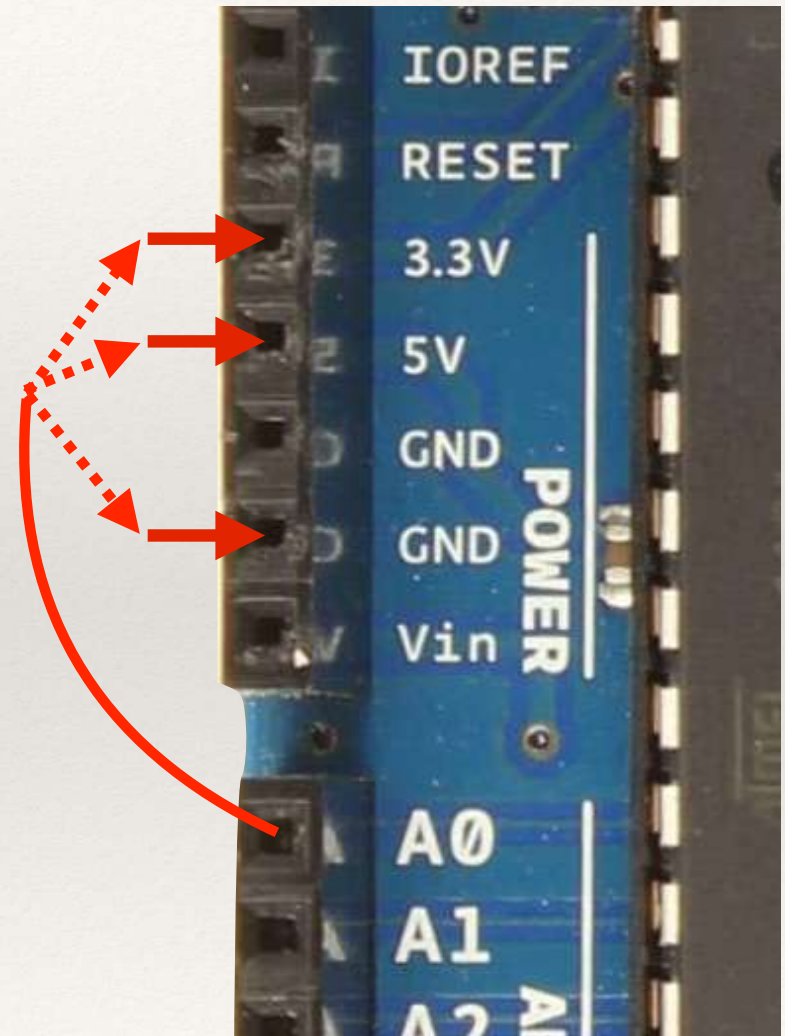
Escolha da tensão de entrada:

5V (Vcc)

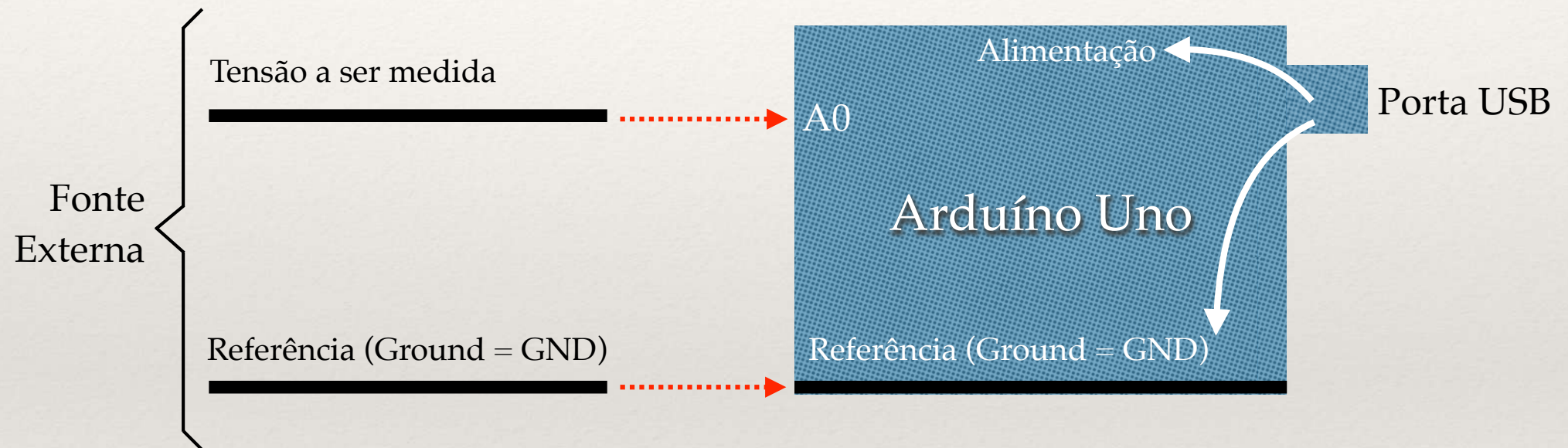
3,3V (saída de alimentação)

0V (GND)

Vamos fazer um voltímetro?



Prática 1 - Leitura Analógica



O compartilhamento da referência de terra (GND) é essencial.

Prática 1 - Leitura Analógica

Os resultados serão vistos no Monitor Serial
Utilizaremos a entrada A0 do Arduíno

Salvaremos o valor lido, e o anterior

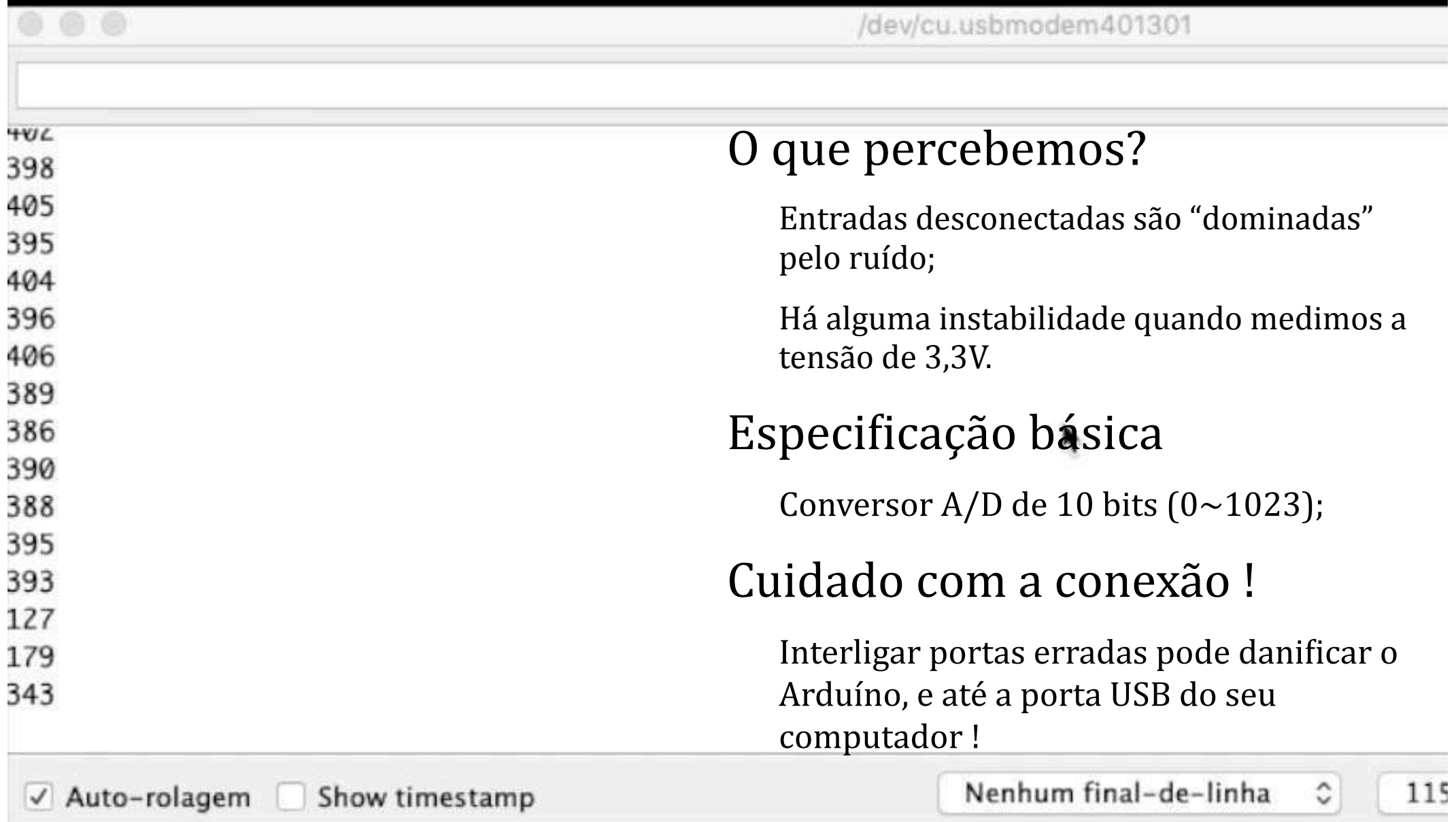
Leitura analógica da entrada

Mostra apenas se houver alteração

Número de amostragens < 10 por segundo

```
void setup() {  
    Serial.begin(115200);  
    pinMode(A0, INPUT);  
}  
  
int entrada, entradaAnterior;  
  
void loop() {  
    entrada = analogRead(A0);  
  
    if( entrada != entradaAnterior ) {  
        Serial.println( entrada );  
        entradaAnterior = entrada;  
    }  
  
    delay(100);  
}
```

Prática 1 - Leitura Analógica



The screenshot shows a serial monitor window with the title bar "/dev/cu.usbmodem401301". The main area displays a list of numbers: 402, 398, 405, 395, 404, 396, 406, 389, 386, 390, 388, 395, 393, 127, 179, and 343. At the bottom, there are checkboxes for "Auto-rolagem" (checked) and "Show timestamp" (unchecked). On the right, there is a dropdown menu set to "Nenhum final-de-linha" and a page number "115".

O que percebemos?

- Entradas desconectadas são “dominadas” pelo ruído;
- Há alguma instabilidade quando medimos a tensão de 3,3V.

Especificação básica

- Conversor A/D de 10 bits (0~1023);

Cuidado com a conexão !

- Interligar portas erradas pode danificar o Arduino, e até a porta USB do seu computador !

Entradas Desconectadas

Entradas desconectadas “flutuam” com os sinais presentes no ambiente

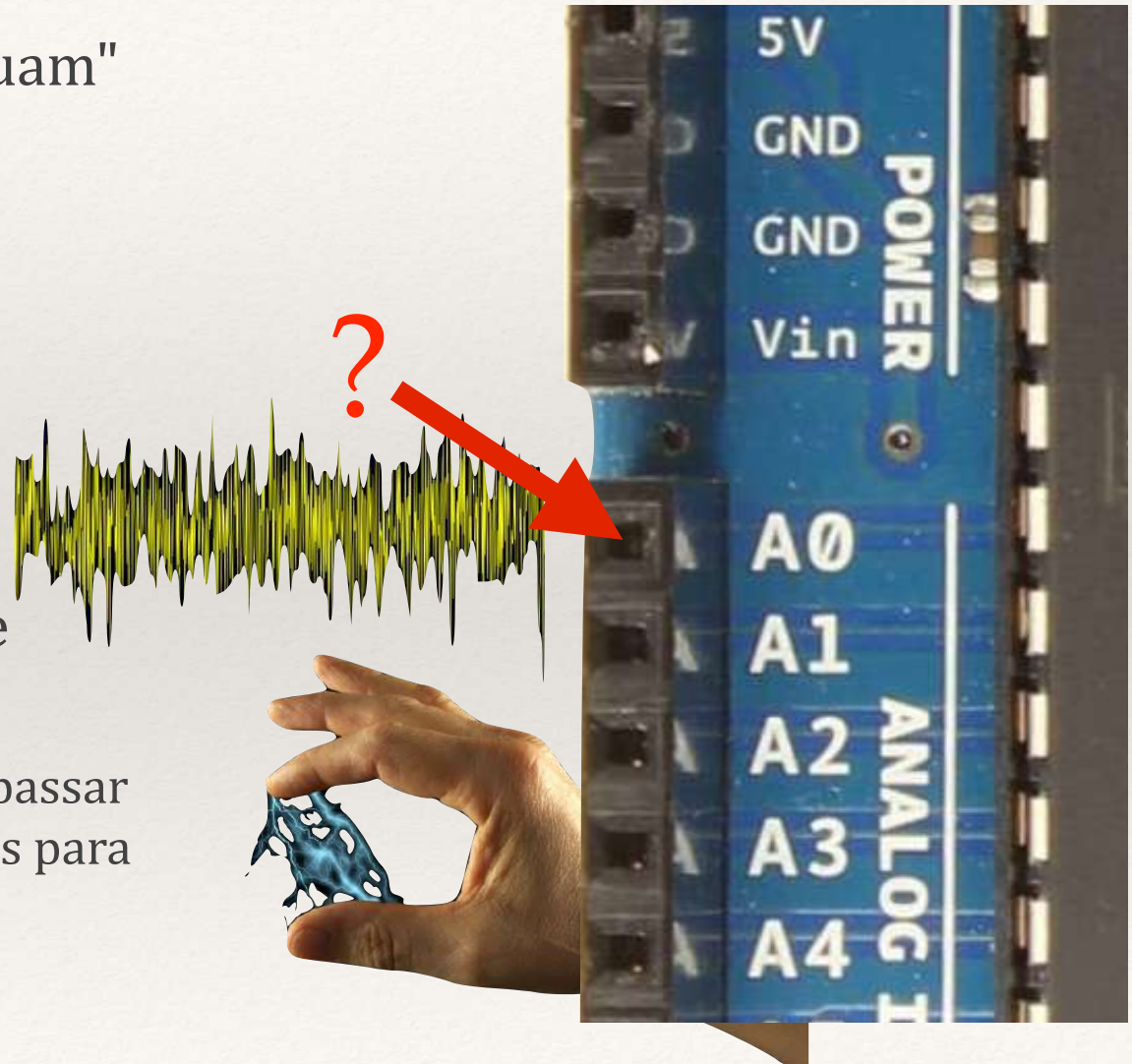
Ruídos elétricos;

Contato manual do operador ou usuário;

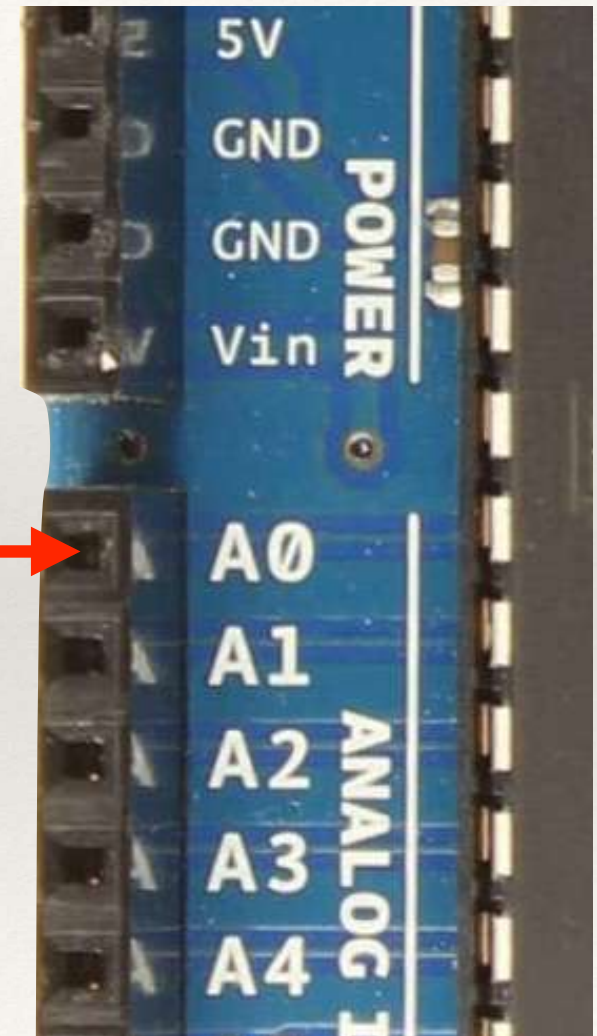
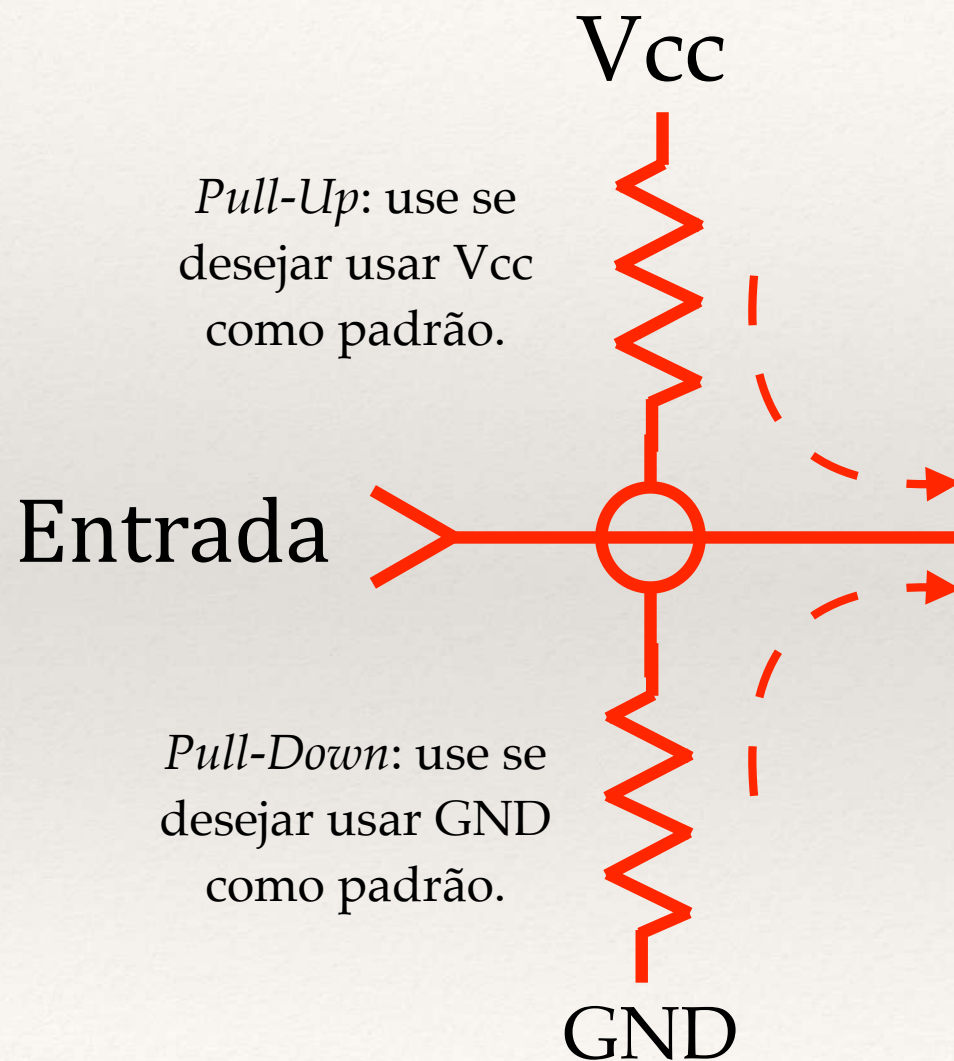
Outros sinais indesejados.

Alguns sinais podem inclusive danificar a porta de entrada

Sinais indesejados podem ultrapassar os níveis de tensão estabelecidos para a porta.



Entradas Desconectadas



Instabilidade

A instabilidade é um sintoma da falta de precisão;

A precisão, exatidão e a resolução são parâmetros importantes, e serão vistos posteriormente;

A imprecisão costuma afetar mais os valores intermediários de uma escala (no caso do experimento, ela só apareceu com 3,3V).

Desafio de Programação

Vamos modificar o código para mostrar o valor da tensão analógica, ao invés do valor lido pelo ADC?

A idéia é mostrar algo como:

3.3V

```
void setup() {  
  
    Serial.begin(115200);  
    pinMode(A0, INPUT);  
  
}  
  
int entrada, entradaAnterior;  
  
void loop() {  
  
    entrada = analogRead(A0);  
  
    if( entrada != entradaAnterior ) {  
        Serial.println( entrada );  
        entradaAnterior = entrada;  
    }  
  
    delay(100);  
  
}
```


Sugestão de Resposta

O valor da tensão não é inteiro

Cálculo da tensão

(usamos a casa decimal para garantir o processamento das constantes como "float")

O valor da tensão terá 2 casas decimais

```
void setup() {  
  
    Serial.begin(115200);  
    pinMode(A0, INPUT);  
  
}  
  
int entrada, entAnt;  
float tensao;  
  
void loop() {  
  
    entrada = analogRead(A0);  
    tensao = 5.0 * entrada / 1023.0;  
  
    if( entrada != entAnt ) {  
        Serial.print( entrada );  
        Serial.print( ": ");  
        Serial.print( tensao, 2);  
        Serial.println("V");  
        entAnt = entrada;  
    }  
  
    delay(100);  
  
}
```

Casas Decimais

O número de casas decimais não é só uma “escolha”; ele depende diretamente da resolução;

Outro parâmetro também importante é o “número de contagens”;

Vamos ver estes parâmetros ?

Precisão e Exatidão

A informação é instável e imprecisa

Nota-se também um erro de medição, mesmo que façamos uma média;

Vamos entender cada um dos parâmetros?



Pouco preciso,
pouco exato

Precisão e Exatidão



Muito preciso,
pouco exato

Precisão não tem relação
com a obtenção do valor
exato

Se o valor medido for fixo, medições
precisas retornarão valores sempre
muito próximos;

Ou seja, para valores fixos, a medição
será estável, mesmo que errada.



Pouco preciso,
pouco exato

Precisão e Exatidão

Exatidão não tem relação com precisão (ou estabilidade)

Muitas vezes medições exatas perdem precisão devido à presença de erros aleatórios (ruído branco, ou aleatório);

Nestes casos, a média dos valores medidos é muito próxima do valor real.



Pouco preciso,
pouco exato



Pouco preciso,
muito exato

Precisão e Exatidão

Exatidão não tem relação com precisão (ou estabilidade)

Muitas vezes medições exatas perdem precisão devido à presença de erros aleatórios (ruído branco, ou aleatório);

Nestes casos, a média dos valores medidos é muito próxima do valor real.



Pouco preciso,
pouco exato



Muito preciso
muito exato

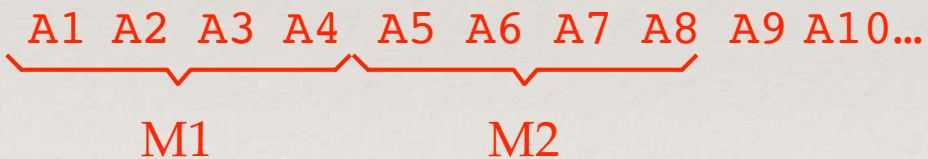
Precisão

Se a causa for um ruído com características aleatórias

Cálculo da média pode resolver;

Solução Típica: média simples;

Número de amostragens depende do desvio padrão



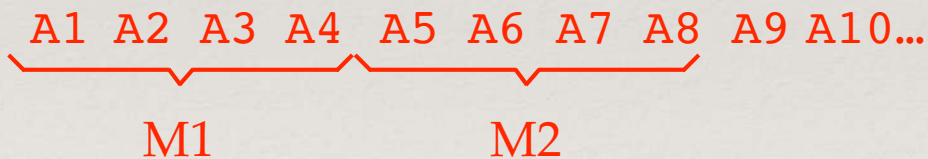
Precisão

Se a causa for um ruído com características aleatórias

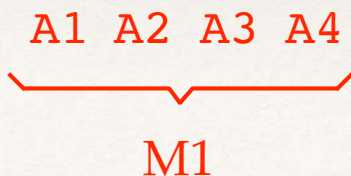
Cálculo da média pode resolver;

Solução Típica: média simples;

Número de amostragens depende do desvio padrão



Médias móveis normalmente são melhores devido à performance



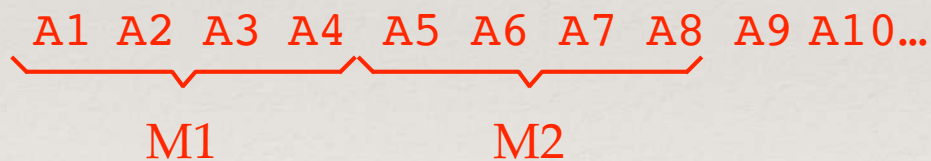
Precisão

Se a causa for um ruído com características aleatórias

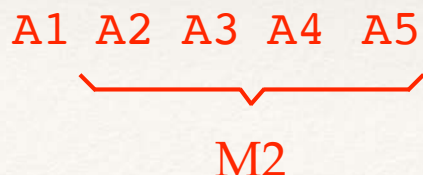
Cálculo da média pode resolver;

Solução Típica: média simples;

Número de amostragens depende do desvio padrão



Médias móveis normalmente são melhores devido à performance



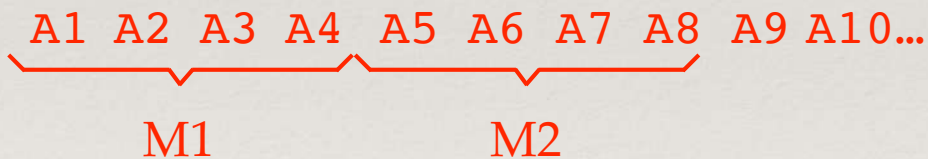
Precisão

Se a causa for um ruído com características aleatórias

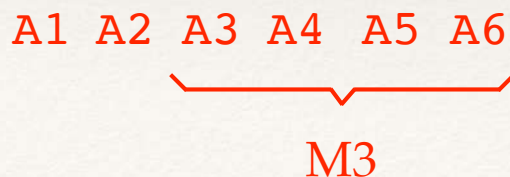
Cálculo da média pode resolver;

Solução Típica: média simples;

Número de amostragens depende do desvio padrão



Médias móveis normalmente são melhores devido à performance



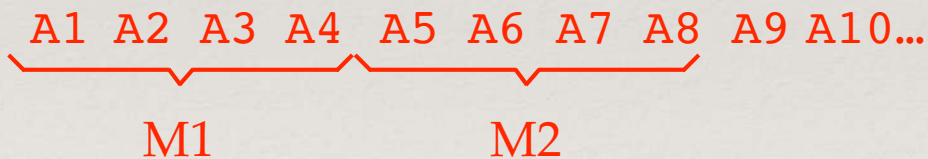
Precisão

Se a causa for um ruído com características aleatórias

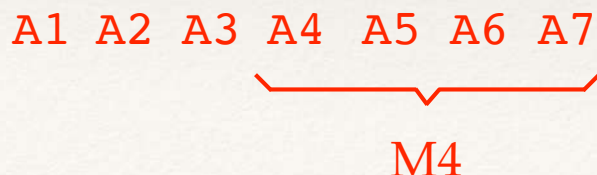
Cálculo da média pode resolver;

Solução Típica: média simples;

Número de amostragens depende do desvio padrão



Médias móveis normalmente são melhores devido à performance



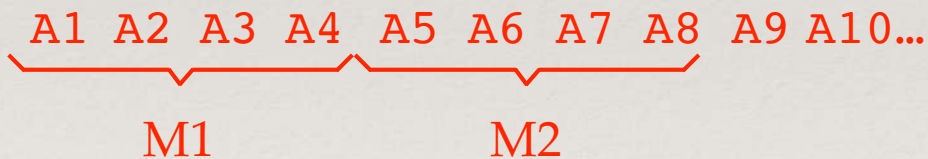
Precisão

Se a causa for um ruído com características aleatórias

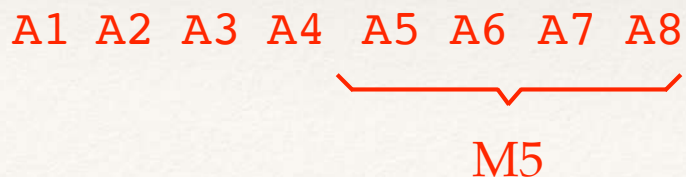
Cálculo da média pode resolver;

Solução Típica: média simples;

Número de amostragens depende do desvio padrão



Médias móveis normalmente são melhores devido à performance



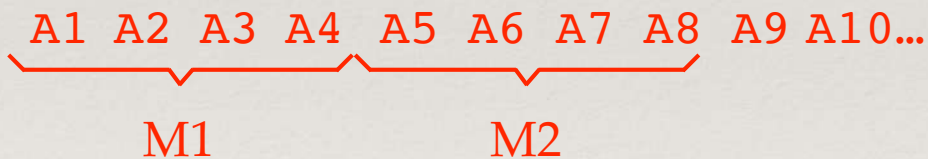
Precisão

Se a causa for um ruído com características aleatórias

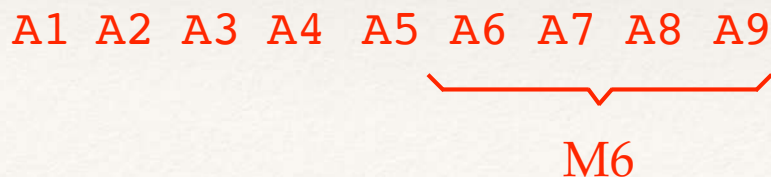
Cálculo da média pode resolver;

Solução Típica: média simples;

Número de amostragens depende do desvio padrão



Médias móveis normalmente são melhores devido à performance



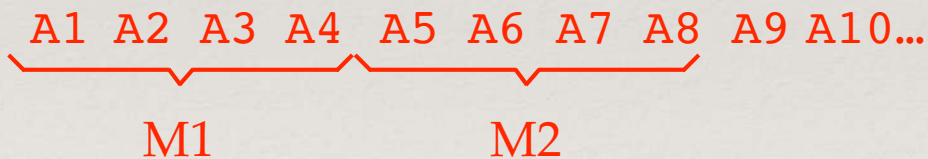
Precisão

Se a causa for um ruído com características aleatórias

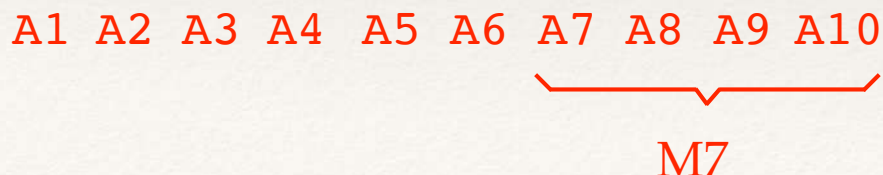
Cálculo da média pode resolver;

Solução Típica: média simples;

Número de amostragens depende do desvio padrão



Médias móveis normalmente são melhores devido à performance



Exatidão

Normalmente a inexatidão é um problema mais grave

A informação correta não está disponível;

Se houver precisão, no entanto, a comparação com outra medição mais exata pode determinar um padrão para correção do erro;

A exatidão é um percentual

Ex.: $\pm (2\% + 2)$ significa 2% de exatidão + 2. Se estiver lendo 10,0, o valor pode estar entre 9,78 e 10,22.

Linearidade

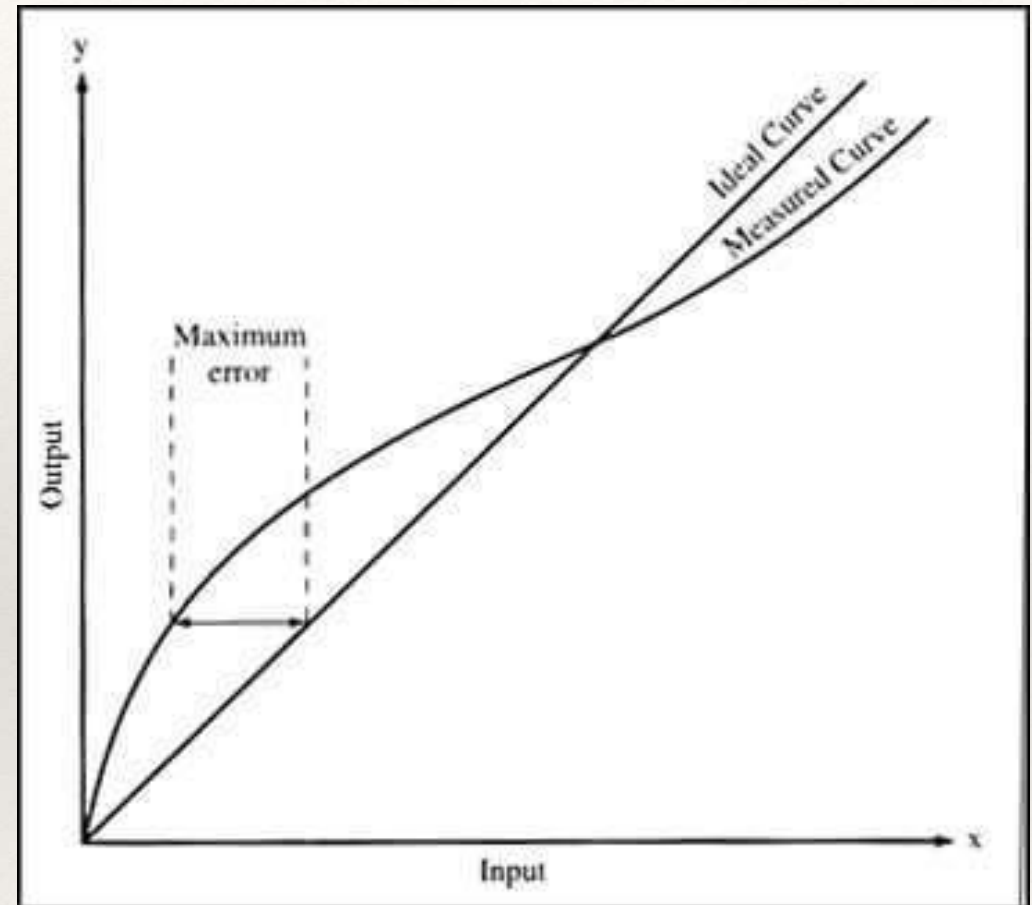
O comportamento da leitura nem sempre corresponde ao comportamento real do valor medido

A não-linearidade provoca erros de medição em parte dos pontos avaliados;

É comum encontrar a não-linearidade nos extremos da faixa de valores mensuráveis;

Em algumas situações, pode valer a pena limitar a faixa de valores medidos, e com isso a resolução;

Em casos específicos, pode-se corrigir a não-linearidade matematicamente.



Precisão, Exatidão e Linearidade

Nenhum destes parâmetros tem relação obrigatória com medições digitais !

Qualquer tipo de medida, inclusive com instrumentos analógicos ou manuais, é afetada pela precisão, exatidão e linearidade;

No entanto, medições digitais em sistemas automatizados também precisam ser avaliadas nestes aspectos.

Fatores Universais

Referência

Ruídos

Escala

Fatores em Medições Digitais

Resolução

Taxa de Amostragem

Referência

Qualquer medição baseia-se em uma referência;

Microcontroladores e sensores podem usar tipicamente uma das seguintes fontes de referência:

- Tensão de Alimentação;

- Tensão de Referência Interna;

- Tensão de Referência Externa.

Referências possuem exatidão e precisão pré-definidas

- Quanto melhor a referência, melhor a medição (e maior o custo);

- Fatores externos, como a temperatura, muitas vezes afetam as referências.

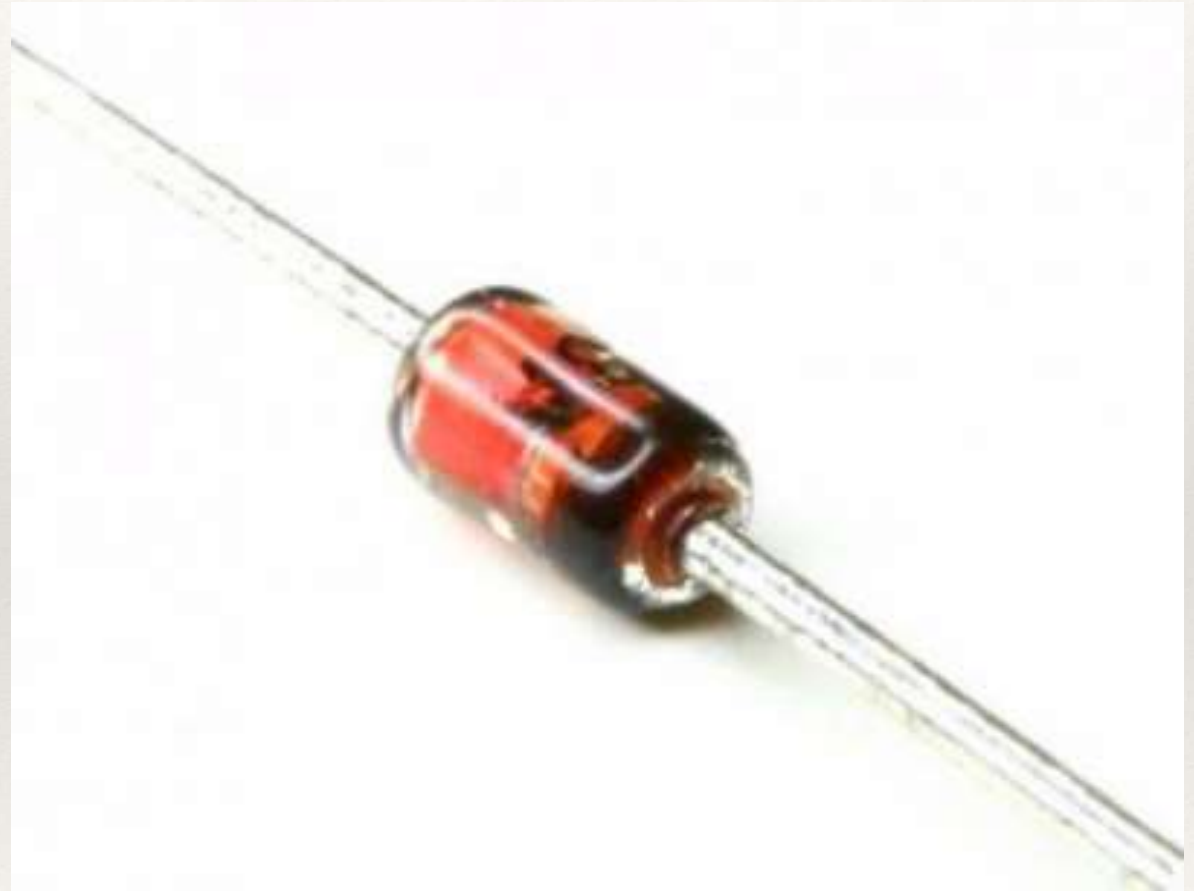
Fontes de Referência

Diodo Zener

Tensões diversas;

Tomar cuidado com a
linearidade;

Afetado pela temperatura
(700ppm/°C)



Fontes de Referência

Referências Programáveis de Precisão

Ex.: TL431 (Texas)

Tensão ajustável de 2,5 a 36V;

0,5% de Exatidão;

Afetado pela temperatura
(600ppm/°C).



Fontes de Referência

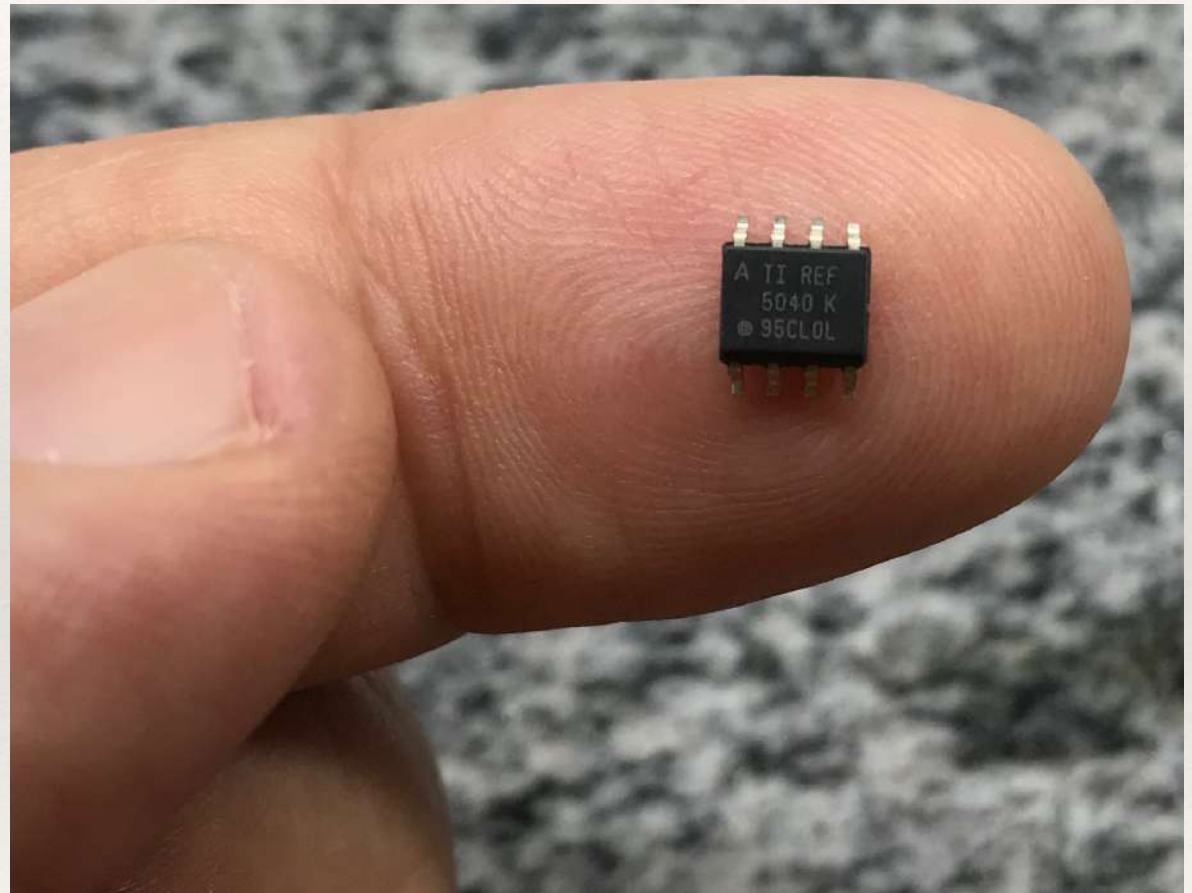
Chips especializados

Ex.: 5040AIDG4 (Texas)

4,096 Volts

0,05% de exatidão;

3 ppm/°C

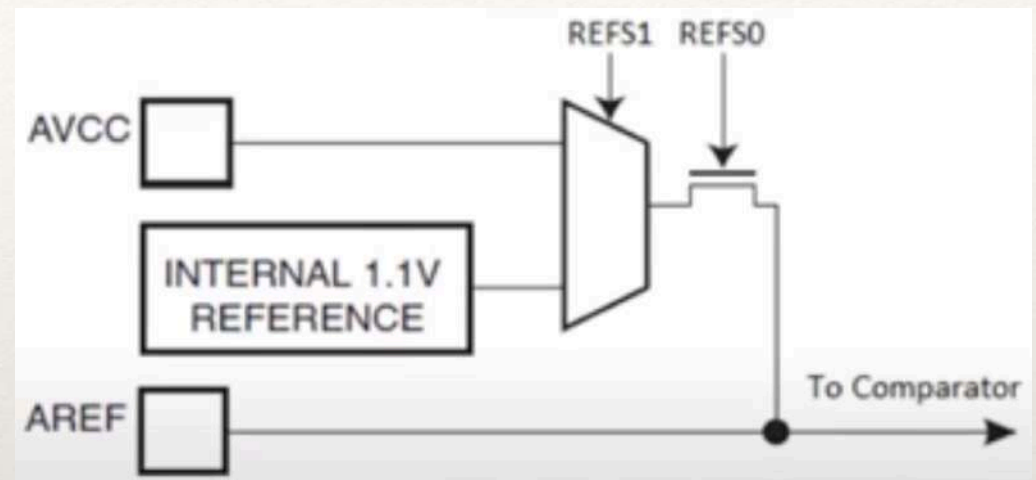


Ex: Referência no Arduino

Por padrão, o Arduino utiliza a tensão de alimentação (5V) como referência;

Outras referências exigem *software* e até *hardware* específico;

Após mudança, as primeiras medições podem ser imprecisas.



analogReference(DEFAULT) : utiliza Vcc, e varia com ela.

analogReference(INTERNAL) : utiliza referência interna de 1,1V

Medir valor efetivo (medir pino AREF durante operação normal).

analogReference(EXTERNAL) : utiliza tensão de entrada no pino AREF

0V < Referência <= 5V (a qualidade da referência determinará a exatidão das medidas);

Não usar analogRead() antes de mudar a referência, pois isso pode danificar o Arduino.

Tratando o ruído

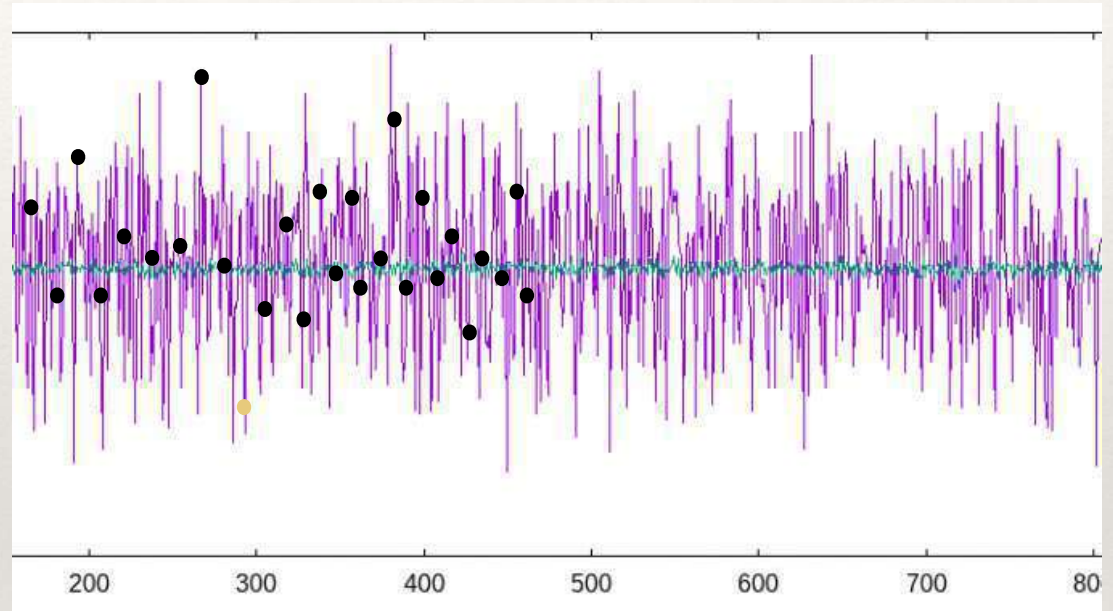
A melhor opção é buscar a redução do ruído no sensor, circuito ou meio ambiente

Qualidade do sensor e componentes;

Proximidade de Fontes de ruído;

Conexão e cabos;

Alimentação elétrica;



Ruídos tipicamente afetam a precisão

Quando aleatórios, ruídos podem ser reduzidos por algoritmos de média;

Se o desvio padrão for elevado, o número de amostragens necessárias pode ser grande.

Aula 04

Ruído causa problemas graves

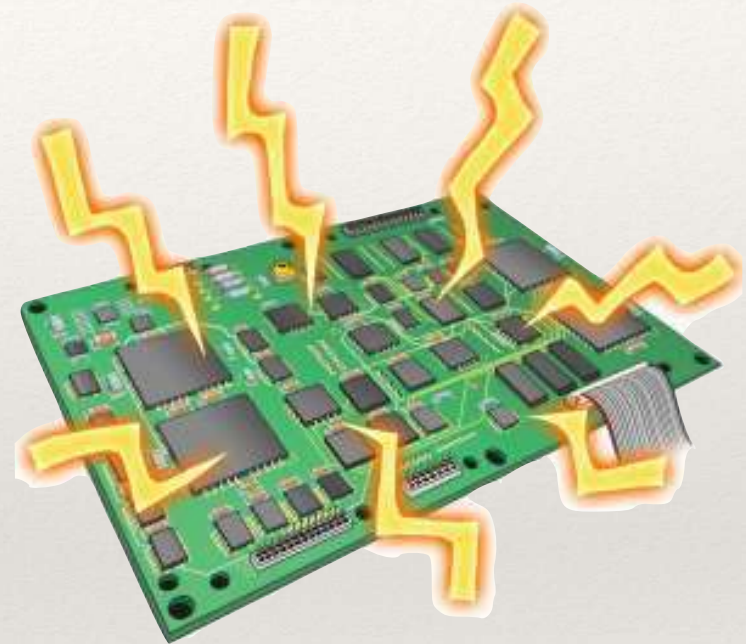
Fontes de ruído de alta intensidade podem provocar problemas ainda mais sérios

Comum em indústrias, e outra aplicações, como sistemas automotivos embarcados;

Ruídos podem impedir a operação de ADCs;

Ruídos podem alterar registros internos do MCU;

Ruídos podem provocar “travamento”.



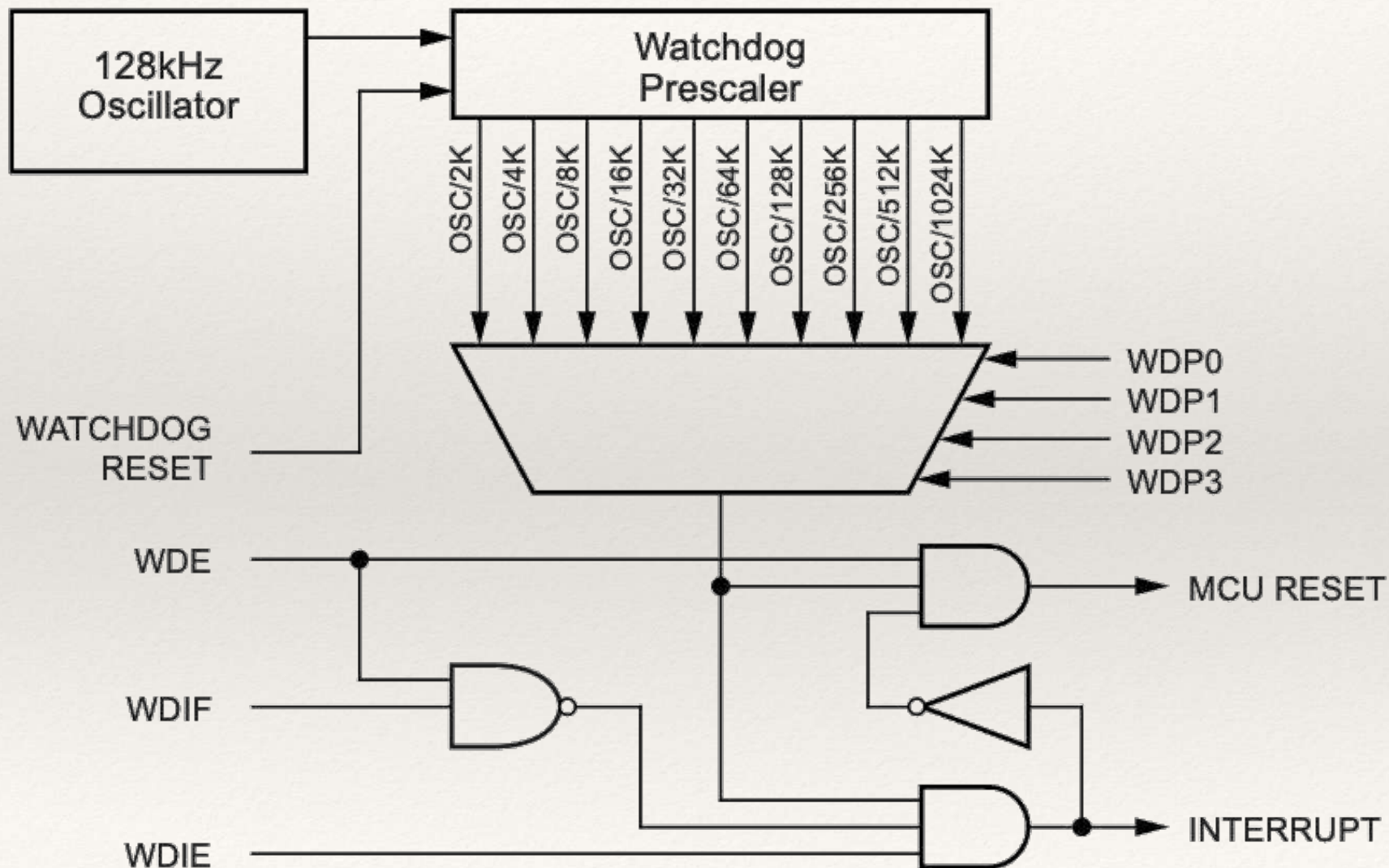
Design e projeto eletrônico podem reduzir impactos

Blindagem e aterramento;

Layout da PCB e outras questões;

Como evitar o travamento em aplicações embarcadas de automação?

WDT (*Watch Dog Timer*)



WDT (*Watch Dog Timer*)

MCUs sujeitas a "travamento"

Espera por dispositivo que não responde;

Endereço da instrução corrompido, etc.

Operação básica do WDT:

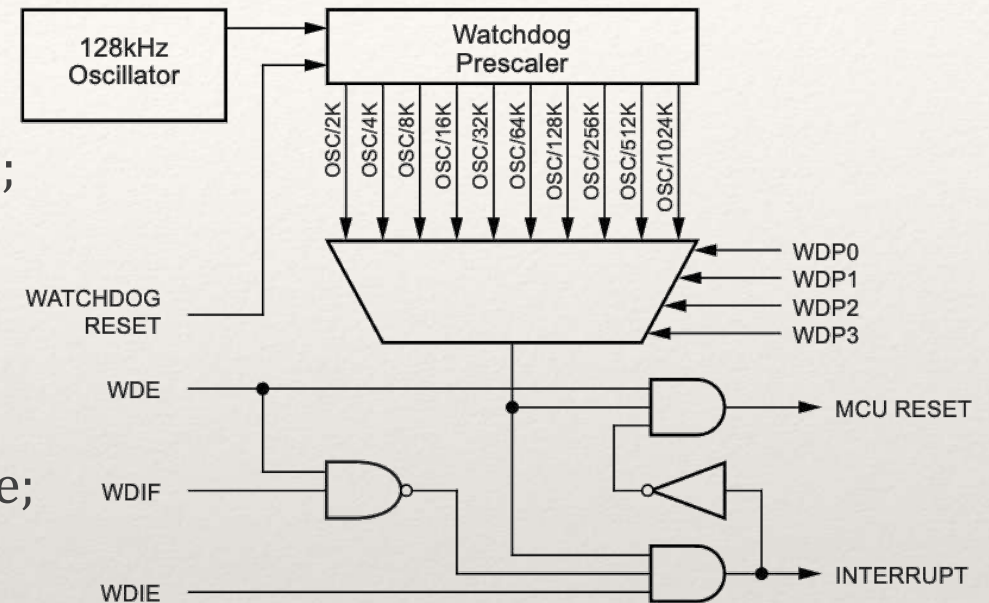
Contador baseado em *clock* independente;

Ao atingir contagem determinada, envia sinal de RESET ao MCU;

MCU evita o RESET enviando ele mesmo um sinal de RESET para o WDT;

Enquanto o MCU enviar periodicamente o sinal, tudo funciona bem;

Diante da eventual falha no envio do sinal (por um travamento), o WDT reinicializa o MCU, permitindo que tudo volte à operação.



Informações Digitais

Entradas analógicas nem sempre estão disponíveis

Nem todo MCU oferece portas ADC, e menos ainda oferecem DAC;

As interfaces para periféricos externos também são tipicamente digitais.

Todo MCU oferece portas de entrada e saída digitais;

Vamos novamente usar o Arduino Uno como exemplo;

Interface na Prática

A tensão elétrica dos sinais trocados com o mundo externo codificam a informação transferida

Portas analógicas LEEM sinais codificados analogicamente

O valor da tensão indica o valor analógico;

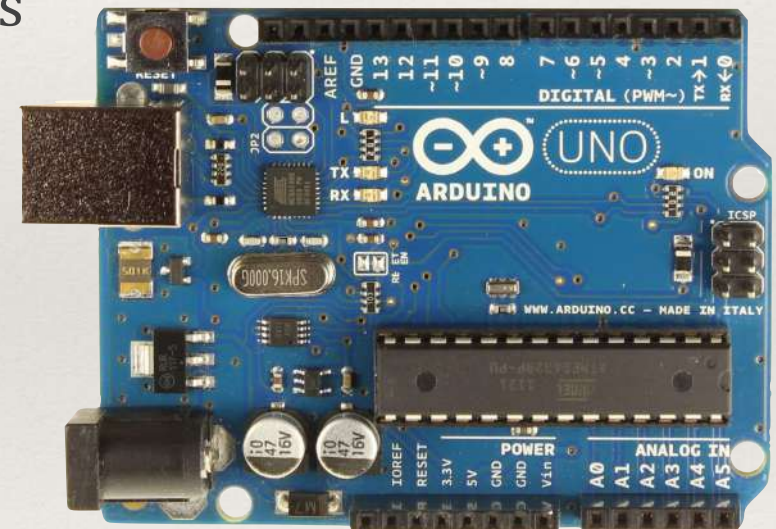
Teoricamente, existem “infinitos” valores (na prática não. Por quê?)

Portas digitais ESCRIVEM E LEEM sinais codificados digitalmente

Tensão alta (HIGH) = 1;

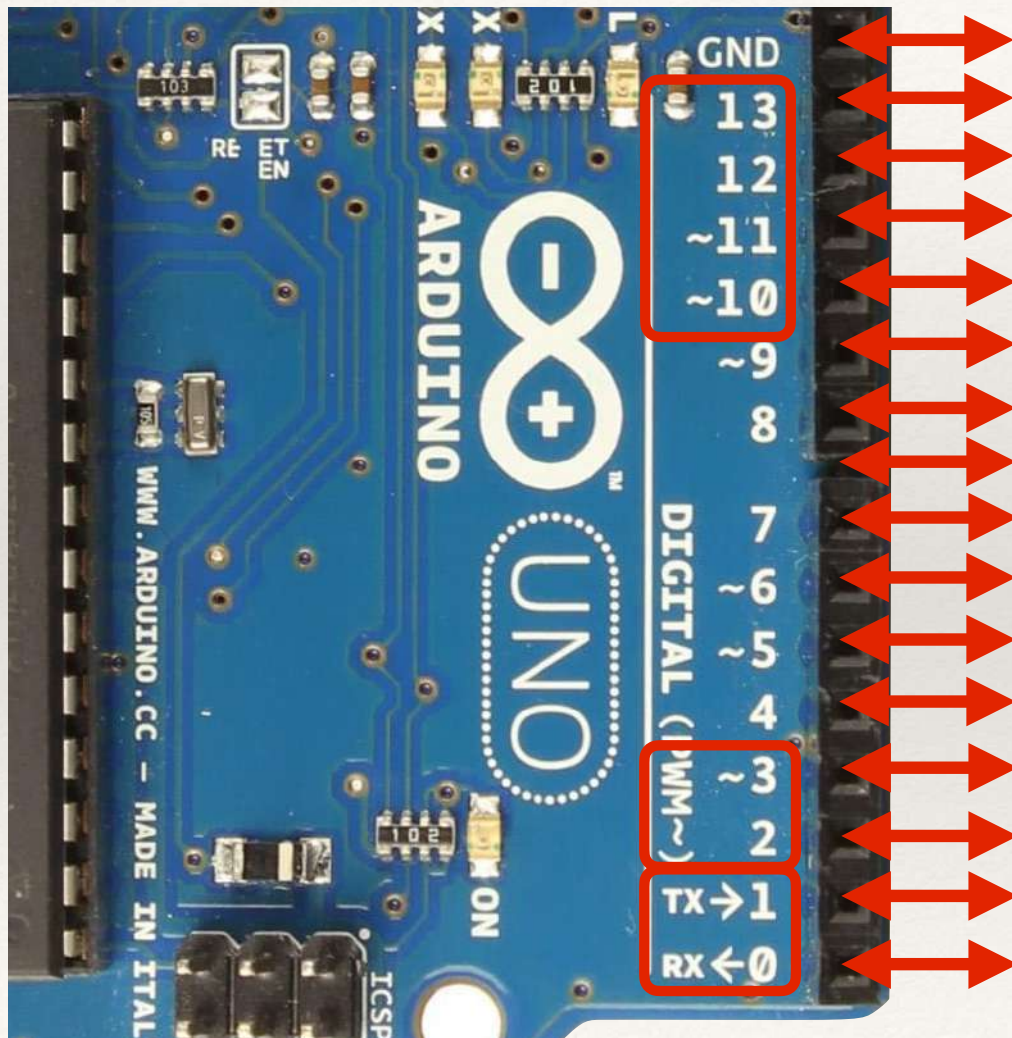
Tensão baixa (LOW) = 0;

Sinais Elétricos
(valores de tensão digitais)



Sinais Elétricos
(valores de tensão analógicos)

Portas Digitais



Alimentação

Portas Analógicas

Portas Digitais

Entrada ou Saída

Configurada via SW

Saídas marcadas com ~ são PWM
(*Pulse Width Modulation*)

< 20mA (máx.40 mA); total 200mA

Funções especiais

UART: Rx/Tx (portas 0 e 1)

Interrupções: INT0 e 1 (portas 2 e 3)

SPI: SS,MOSI,MISO,SCK(p. 10~13)

Pino 13: LED interno

Informações Digitais

Entradas analógicas nem sempre estão disponíveis

Nem todo MCU oferece portas ADC, e menos ainda oferecem DAC;

As interfaces para periféricos externos também são tipicamente digitais.

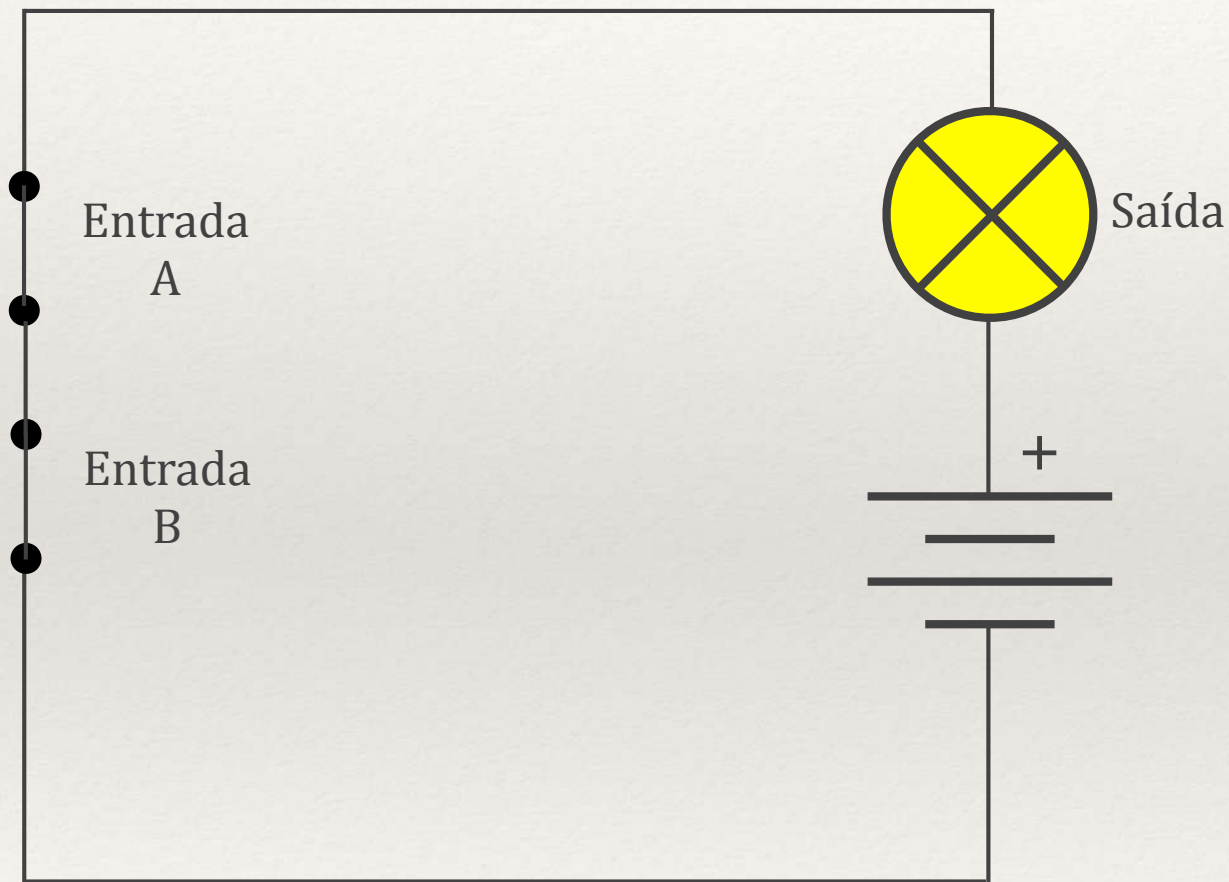
Todo MCU oferece portas de entrada e saída digitais;

Vamos novamente usar o Arduíno Uno como exemplo;

O processamento de informações digitais ocorre através de funções lógicas;

O armazenamento nos MCUs ocorre em dispositivos de memória específicos, normalmente internos.

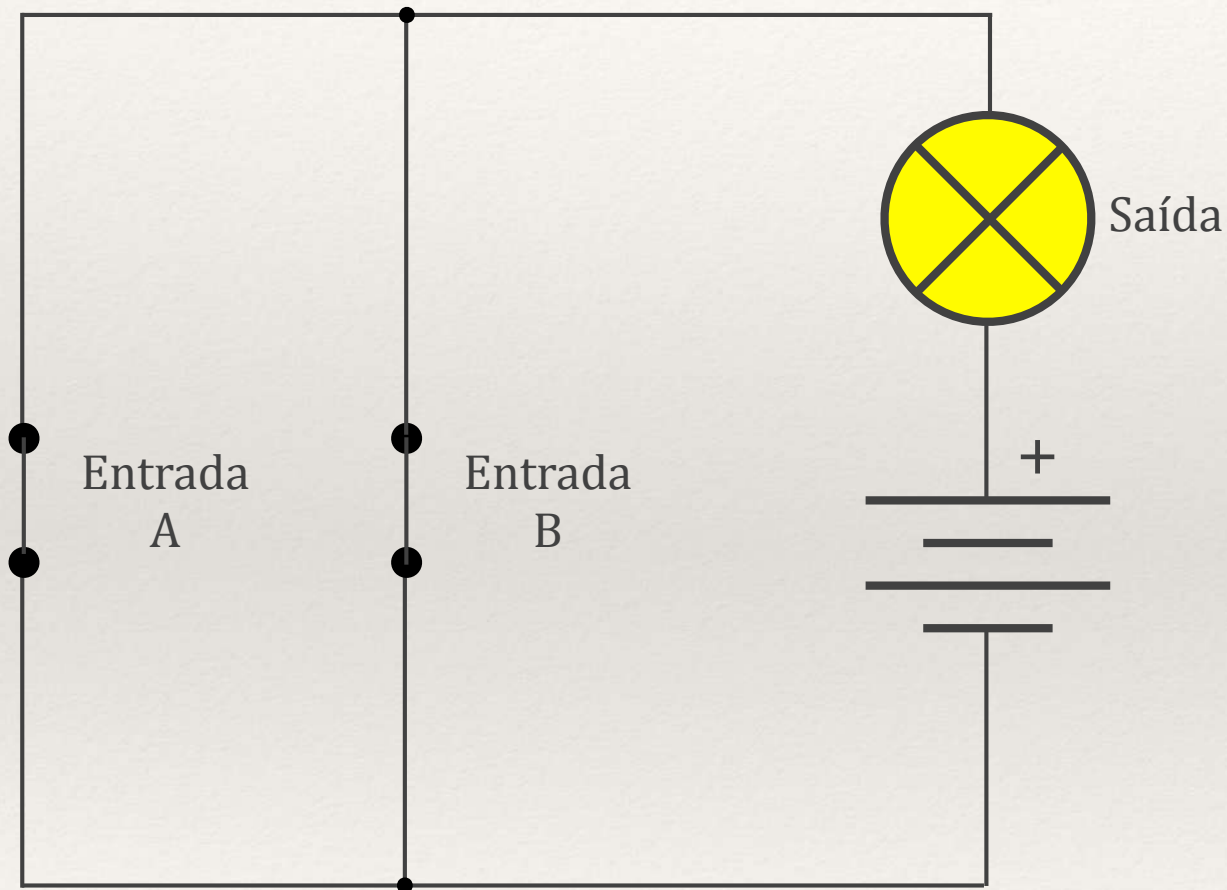
Blocos Funcionais Básicos



Lógica "E"
(AND)

A	B	S
0	0	0
0	1	0
1	0	0
1	1	1

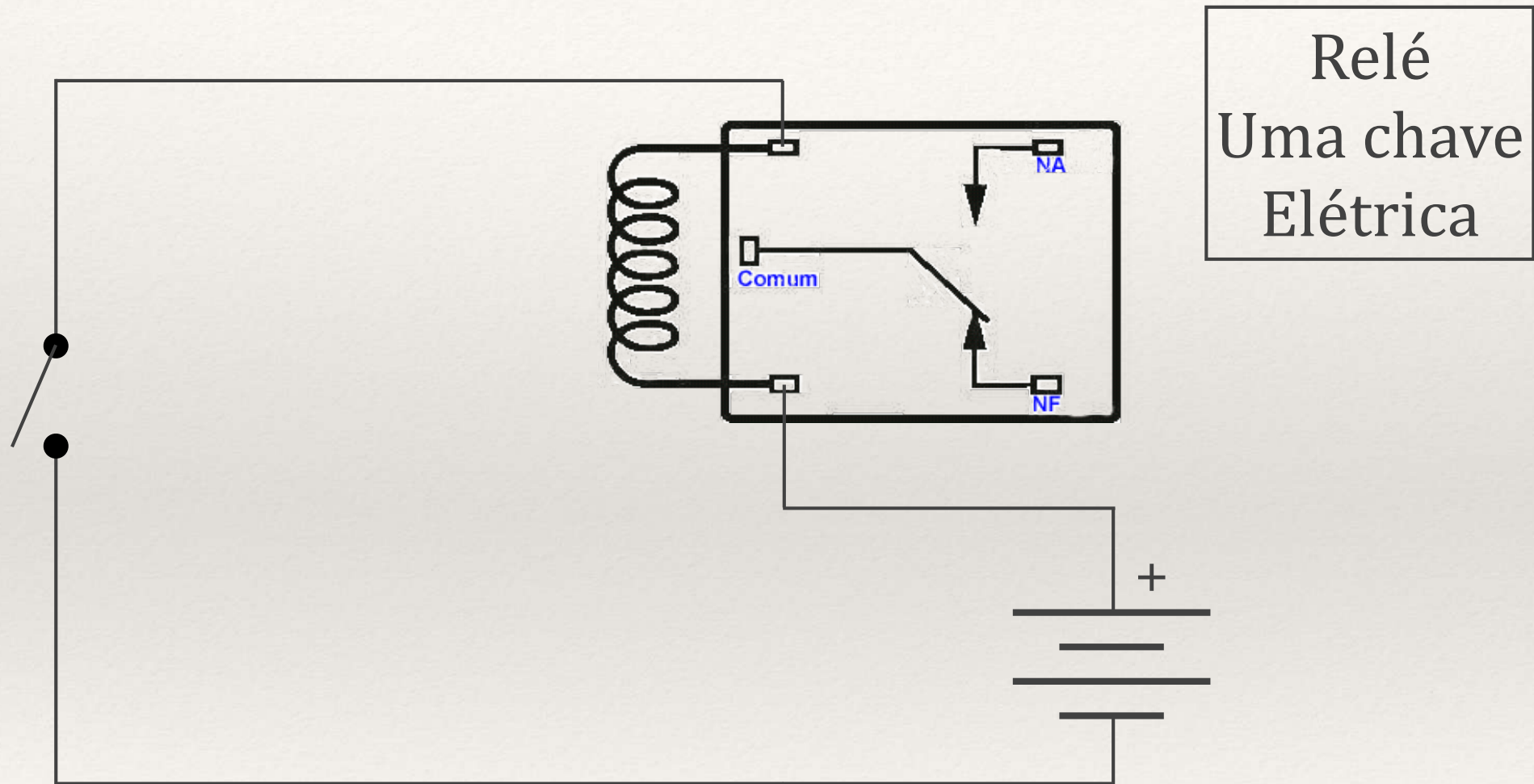
Blocos Funcionais Básicos



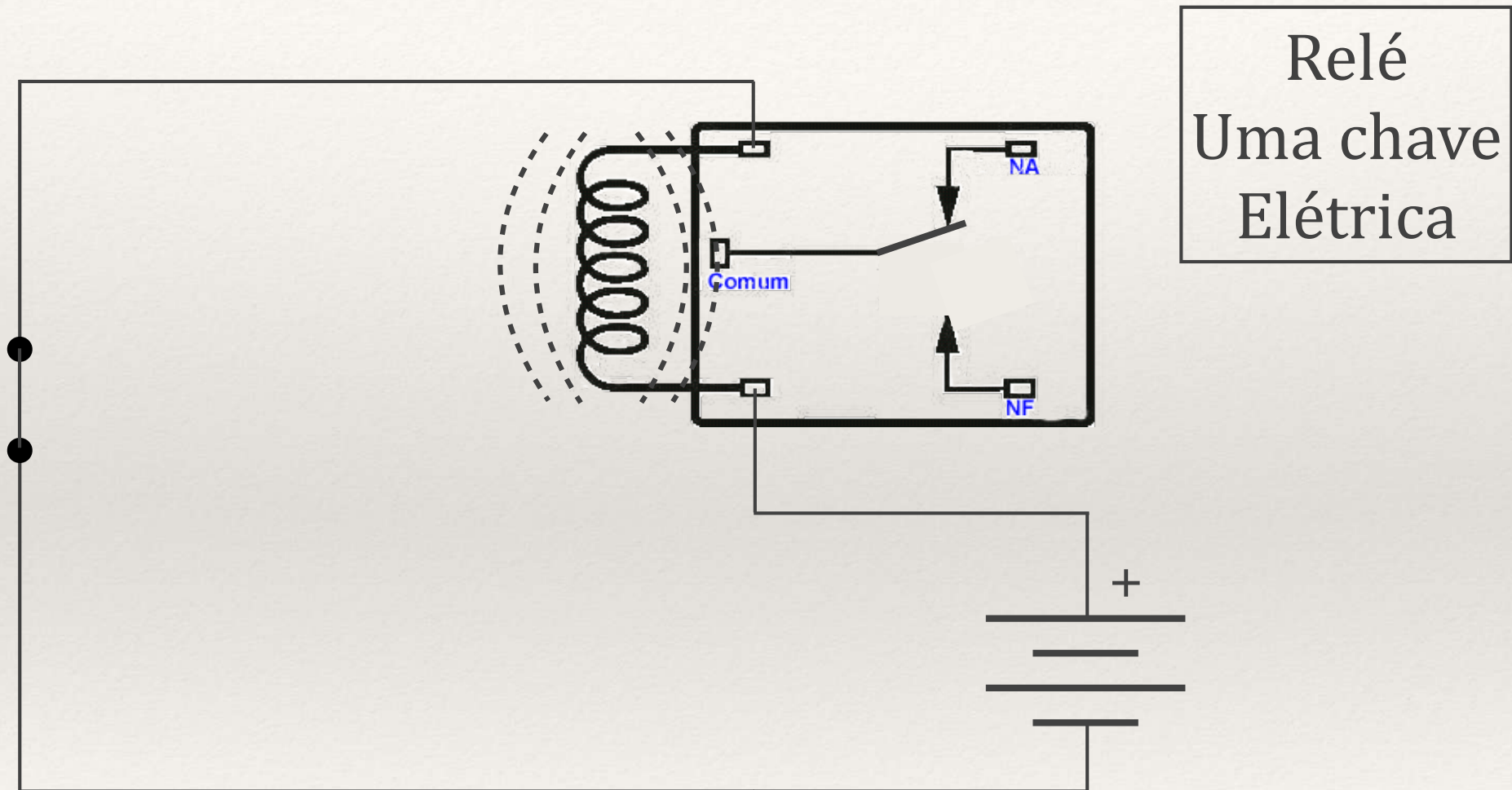
Lógica "OU"
(OR)

A	B	S
0	0	0
0	1	1
1	0	1
1	1	1

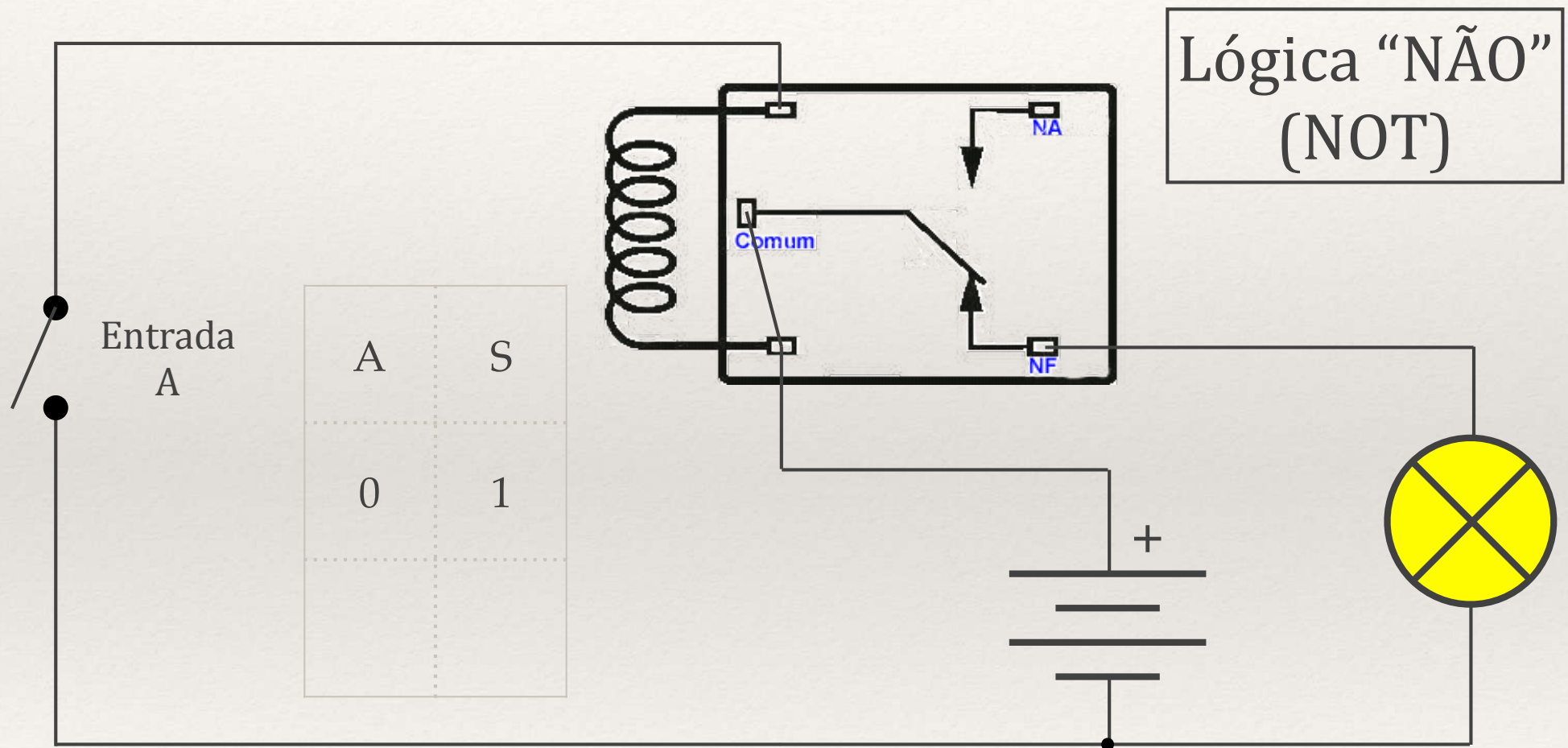
Blocos Funcionais Básicos



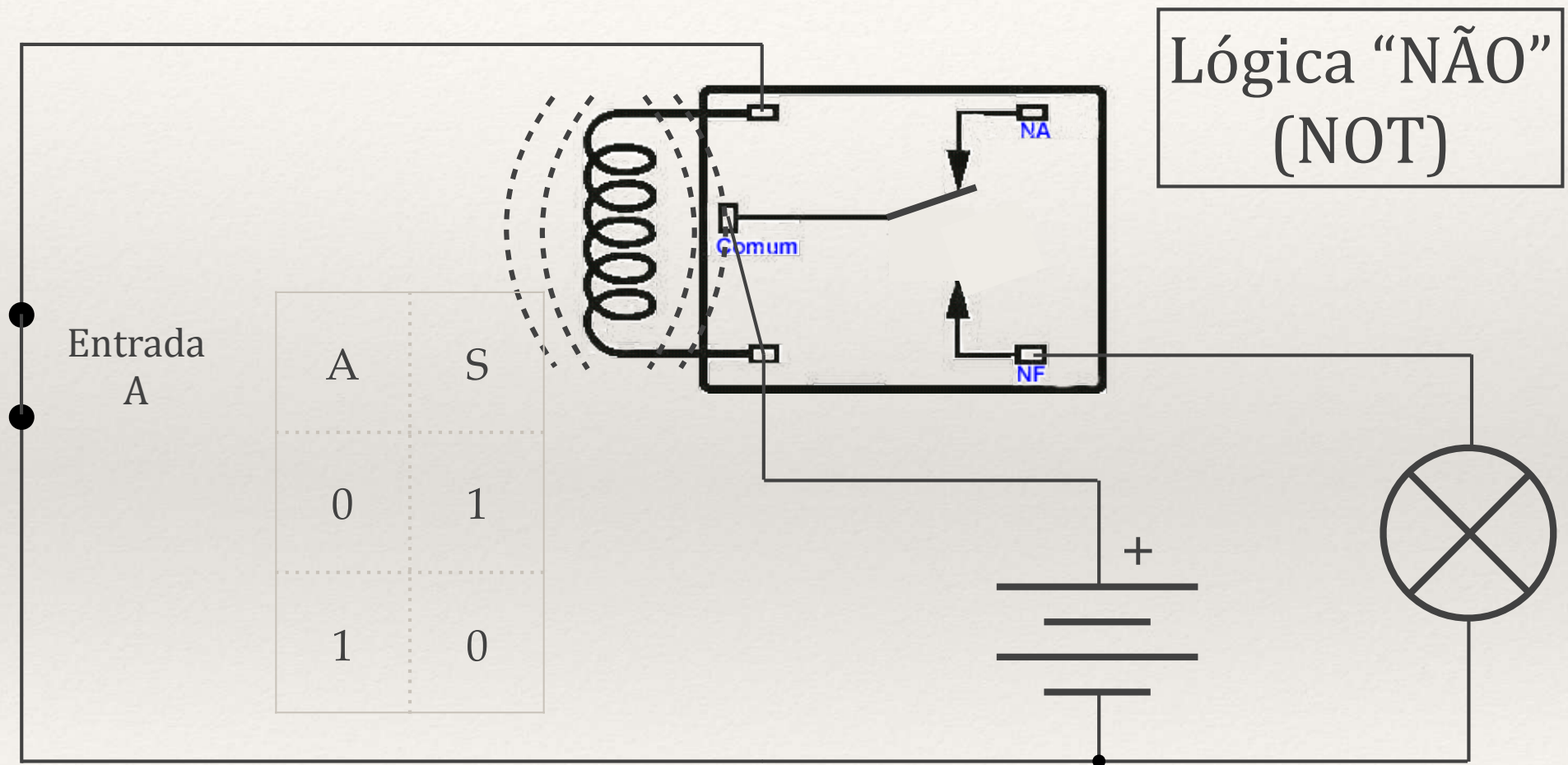
Blocos Funcionais Básicos



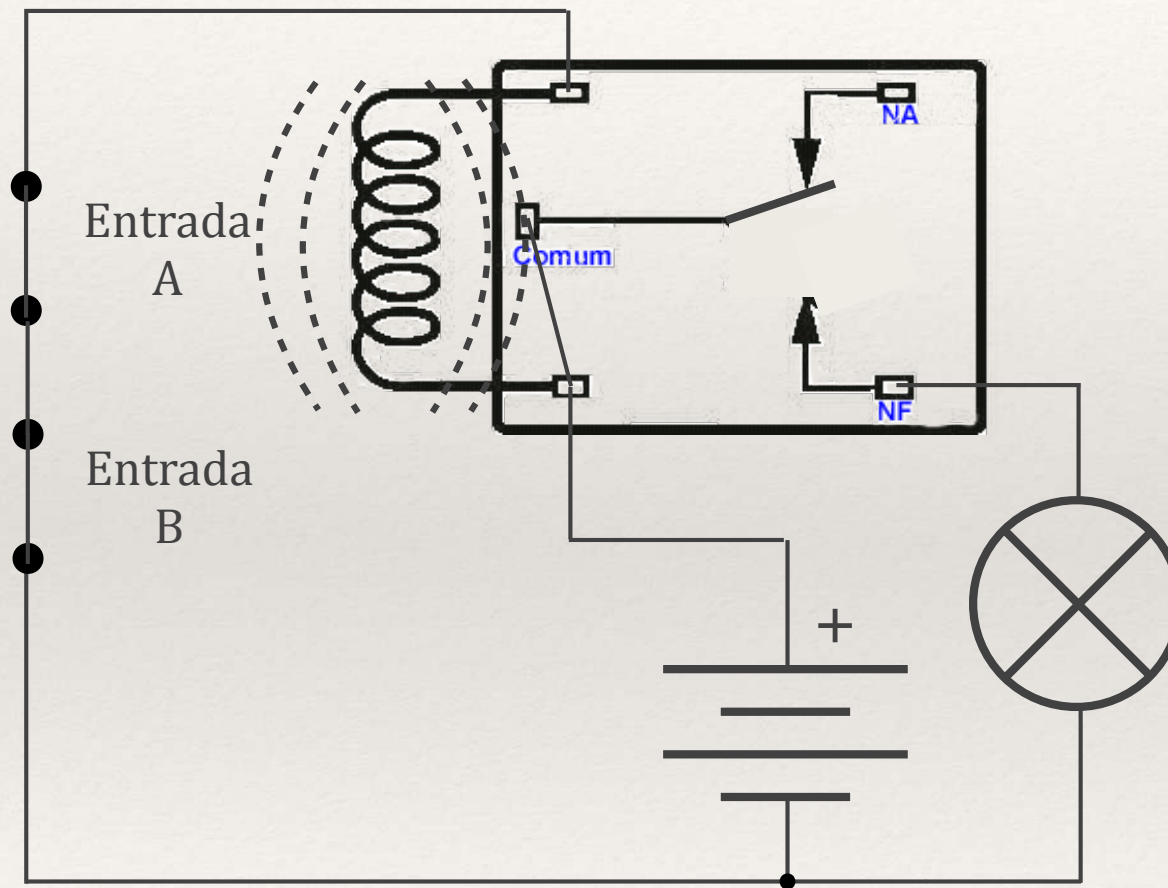
Blocos Funcionais Básicos



Blocos Funcionais Básicos



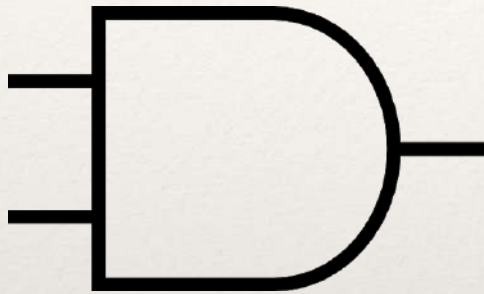
Blocos combinados



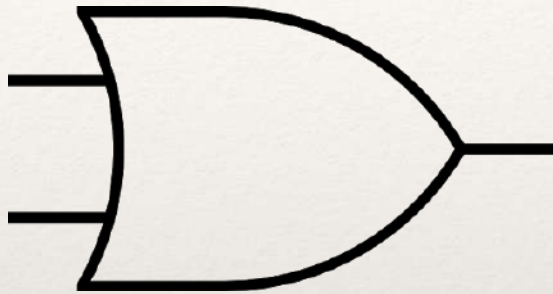
Lógica “Não E”
(NAND)

A	B	S
0	0	1
0	1	1
1	0	1
1	1	0

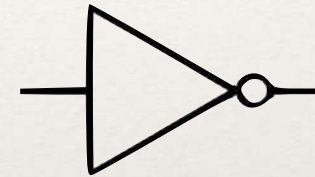
Portas Lógicas



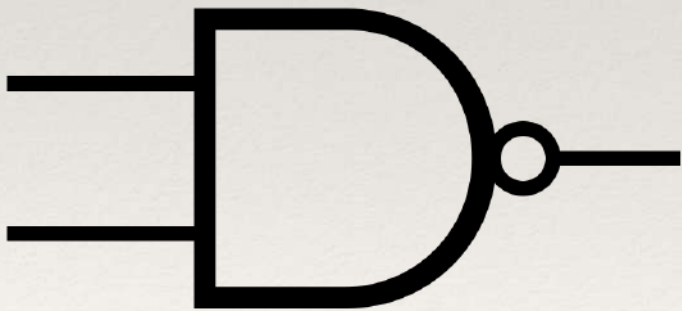
AND



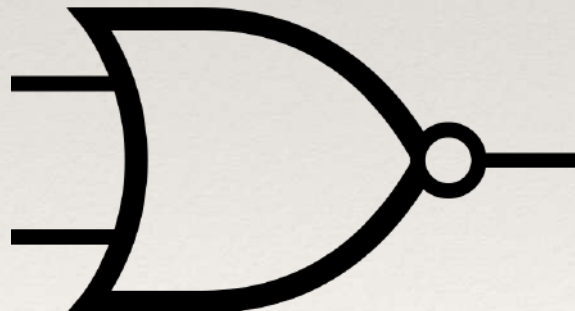
OR



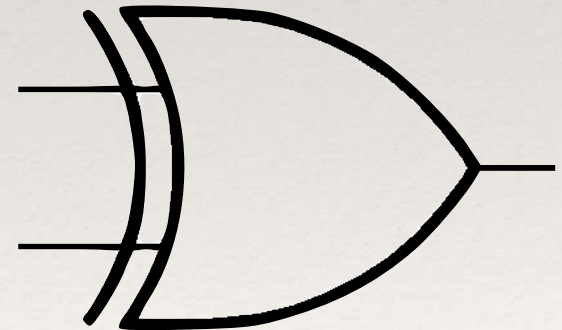
NOT



NAND

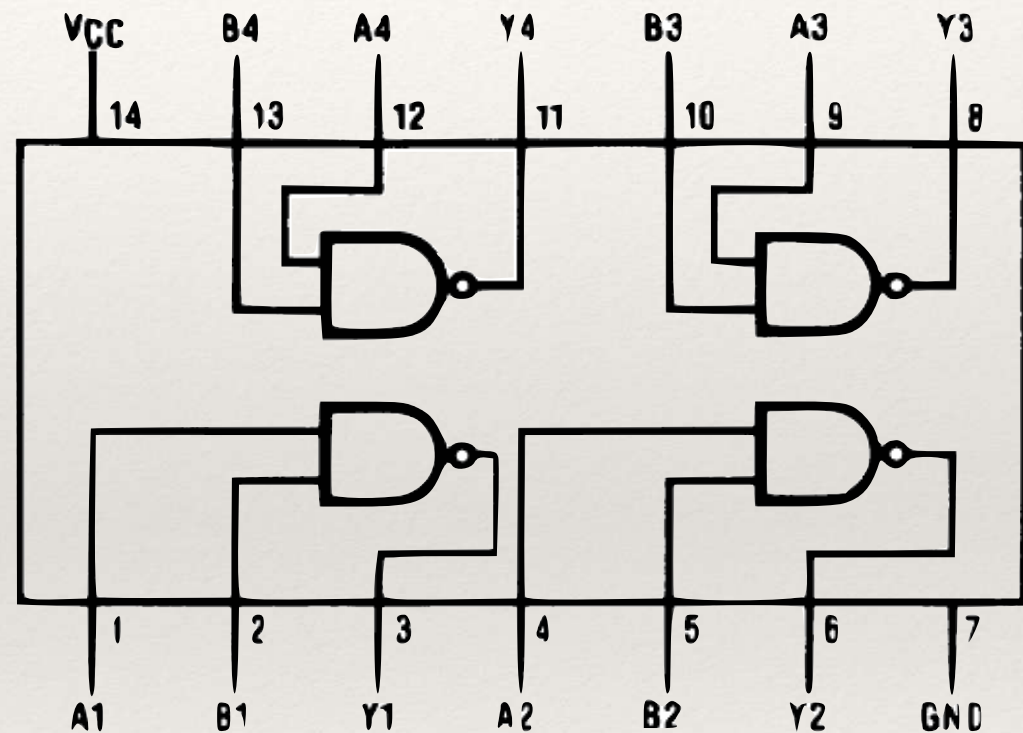


NOR



XOR

Circuito Integrado 74HC00



Top View

Operadores Lógicos em C

Convencionais	
Símbolo	Operação
&&	E (AND)
 	OU (OR)
!	NÃO (NOT)

Bit a bit	
Símbolo	Operação
&	E (AND)
 	OU (OR)
^	OU Exclusivo (XOR)
~	NÃO (NOT)
<<	Desloc.Esquerda
>>	Desloc.Direita

Usos Típicos dos Operadores

Alguns operadores são utilizados, especialmente em Assembly e C, para manipular bits específicos

“Setar” um bit específico

0000	0	1	00	}OR	("setar" o bit 2) (não interessa o valor)
????	???	???			
????	?	1	??		

“Resetar” um bit específico

1111	1	0	11	}AND	("resetar" o bit 2) (não interessa o valor)
????	???	???			
????	?	0	??		

Além disto, o XOR também é utilizado para “zerar” uma variável

Se $X = (???? ????)_2$, então $X \wedge X = (0000 0000)_2$

Operação é mais eficiente do que carregar o valor zero.

Usos Típicos dos Operadores

Para definir uma constante com valores específicos em um bit, utiliza-se tipicamente a operação de deslocamento

```
X = 0000 0100      (bit 2 "setado")
```

```
X = (1 << 2)
```

```
X = 1111 1011      (bit 2 "resetado")
```

```
X = ~(1 << 2)
```


Memória nos MCUs

Tipos mais comuns

RAM (estática X dinâmica)

EPROM e Flash

RAM Estática X Dinâmica (voláteis)

Espaço ocupado X Performance;

SRAM usa 8 MOSFETS; DRAM utiliza 1;

DRAM exige refresh;

Computadores pessoais utilizam muita DRAM (8~16 GB típicos)

MCUs usam SRAM (Ex: 2KB no ATmega, 520K no ESP32)

EPROM e Flash (não voláteis)

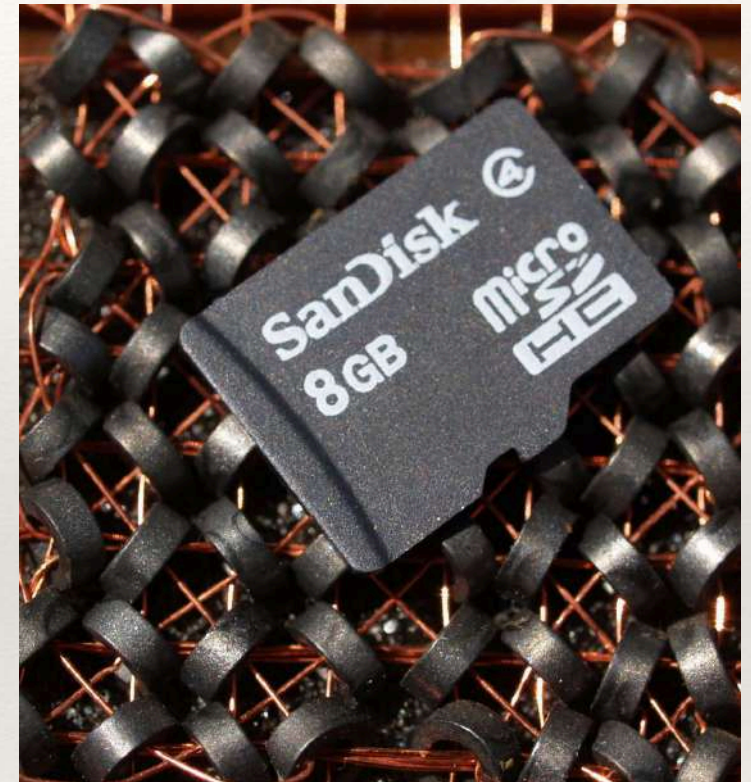
EPROM permite acesso ao nível de palavra, Flash a blocos;

EPROM tem menor número de ciclos (100K), mas é rápida;

Ambas utilizadas para armazenar código (*sketches/firmware*).

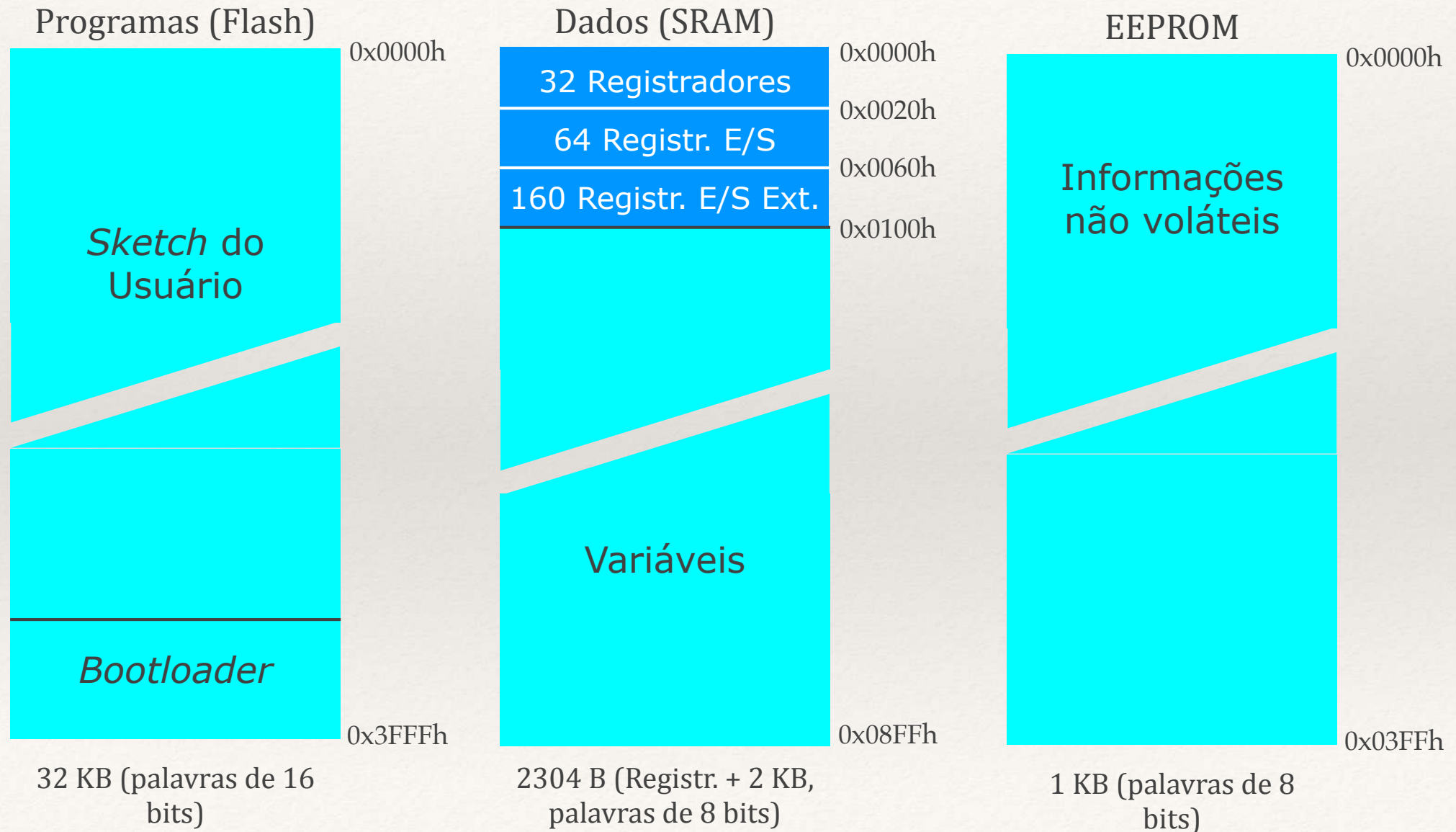
Memória externa e mídias

Cuidado com o uso adequado e seus impactos.



8 GBytes x 8 Bytes

Memória no ATmega328P



Registradores

Parte especial da memória

RAM Estática;

Acumulam apenas 1 palavra, onde às vezes o que interessa está em um bit específico;

Classificados em registradores “de Função Geral” e “de Funções Especiais”.

Registradores de Função Geral

Utilizados pelos programadores, diretamente ou pelos compiladores;

Armazenam operadores para cálculos ou variáveis específicas de um código.

Registradores de Função Especial (SFT - *Special Function Register*)

Controlam aspectos internos do MCU. Eis alguns exemplos:

PC (*Program Counter*): ponteiro para a memória onde está a próxima instrução. Em alguns MCUs não é acessível ao programador (ATMega328P, PIC etc);

AC (*Accumulator*): armazena resultados intermediários de cálculos, e funciona como operando para muitas instruções. Em algumas CPUs e MCUs, o programador pode escolher o registrador para esta função;

Registradores

Registradores para funções especiais (cont.)

Controlam aspectos internos do MCU. Eis alguns exemplos:

SP (*Stack Pointer*): ponteiro para a posição da pilha que onde está o último elemento armazenado;

Os registradores, e seus nomes, variam de acordo com o modelo e fabricante do MCU ou CPU.

Registradores de controle

Diversas funcionalidades de um MCU são controladas através de registradores

Timers, Contadores, Registros de *status*, Controle de portas, Interrupções etc.

Operações sobre registradores podem ser unidirecionais e bidirecionais;

Registradores podem ser manipulados pelo programador diretamente, bit a bit, ou indiretamente através de instruções em linguagem de alto nível

Também pode ocorrer acesso DMA (*Direct Memory Access*) por dispositivos externos;

É importante ficar atento para evitar conflitos no uso.

ATMega328P em blocos

Os registradores controlam toda a operação do ATMega328P

Muitos são acessados indiretamente quando usamos o conjunto de instruções padrão, ou uma biblioteca externa;

Alguns recursos exigem acesso direto

Interrupções de Relógio;

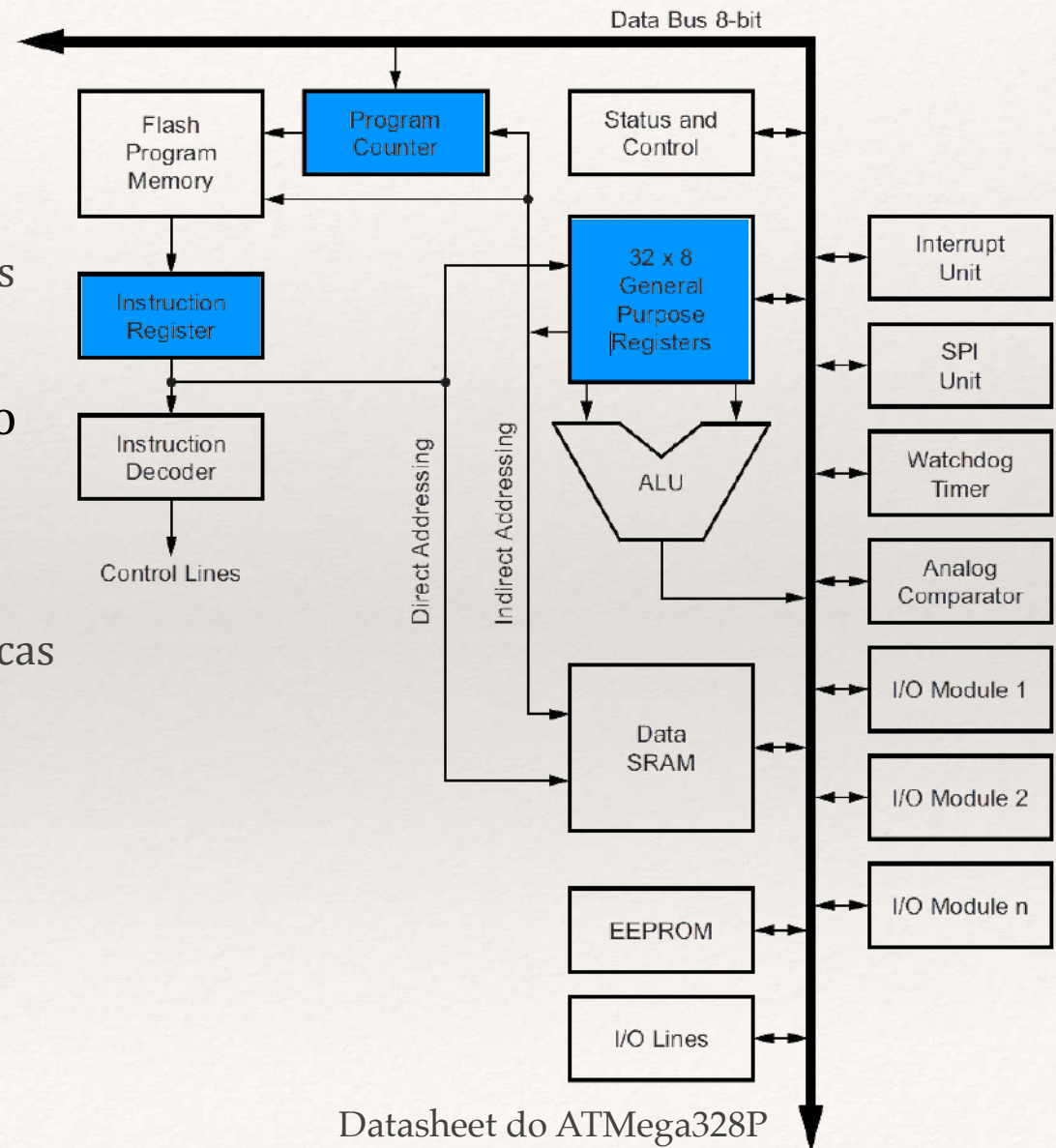
Alteração de frequência PWM;

Manipulação "criativa" de portas em placas de protótipo.

Acesso via registradores

Códigos menores, e mais rápidos;

Manipulação e consulta binária mais simples.



Acesso Binário

Bits dos registradores agregam funções na mesma palavra

Alterações e consultas bit-a-bit;

Operações lógicas binárias: E, OU, NÃO, deslocamento.

Registradores de 8 ou 16 bits

Registradores de 16 bits tipicamente são tratados byte a byte.

Ativando bits (*set bits*)

Diretamente pela carga da palavra;

Carga de bit com operação OU (*OR*).

Desativando bits (*reset bits*)

Diretamente pela carga da palavra;

Desativação de bit com operação E (*AND*).

Prática 2 - Acesso Binário

Projeto básico com Arduíno Uno

Ler botão;

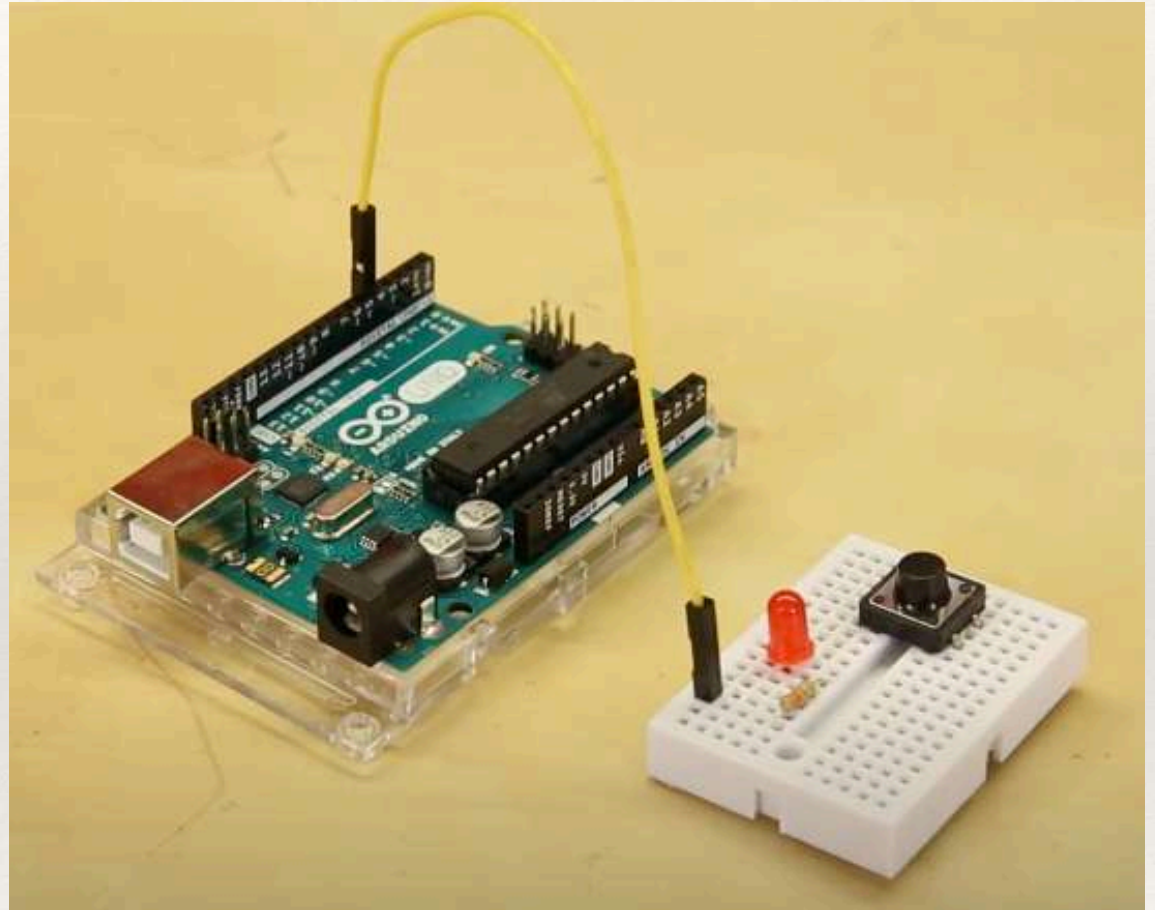
Se apertado, acende LED; caso contrário, apaga.

Desenvolvimento do teste

Teste inicial com código em C++ padrão do Arduíno;

Migração de parte do código, desta vez manipulando diretamente os registradores;

Observação dos resultados.



Circuito de Teste

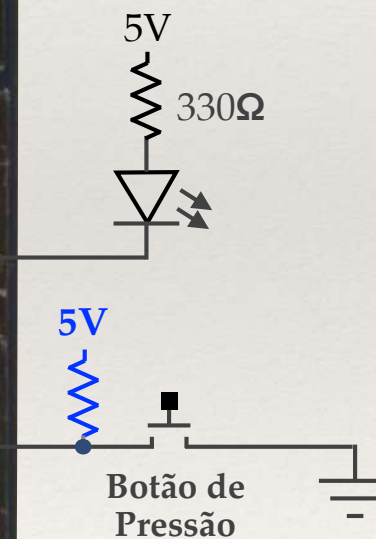


Componentes alimentados pelo Arduíno Uno

LED (saída digital 5);

Botão de Leitura (leitura digital 2)

Pull-Up interno



Sketch de Teste

Sketch define entrada e saída

Pino 2 utilizado para leitura do botão,
com uso de *PullUp* interno;

Pino 5 utilizado para ativar o LED.

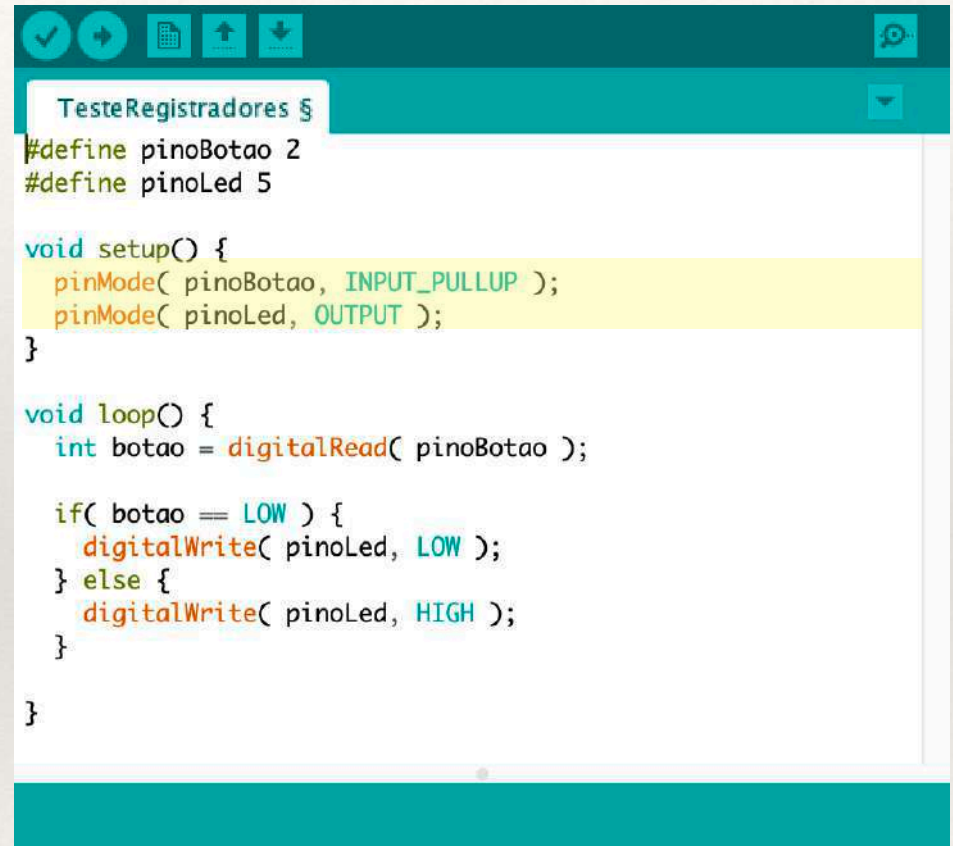
Função básica Arduino

`pinMode ( ) ;`

Opera indiretamente com registradores.

Quando compilar, anote o status
da memória !

Informação será utilizada para
comparação após migração do código.

A screenshot of the Arduino IDE interface. The title bar shows the sketch name 'TesteRegistadores'. The code editor contains the following C++ code:

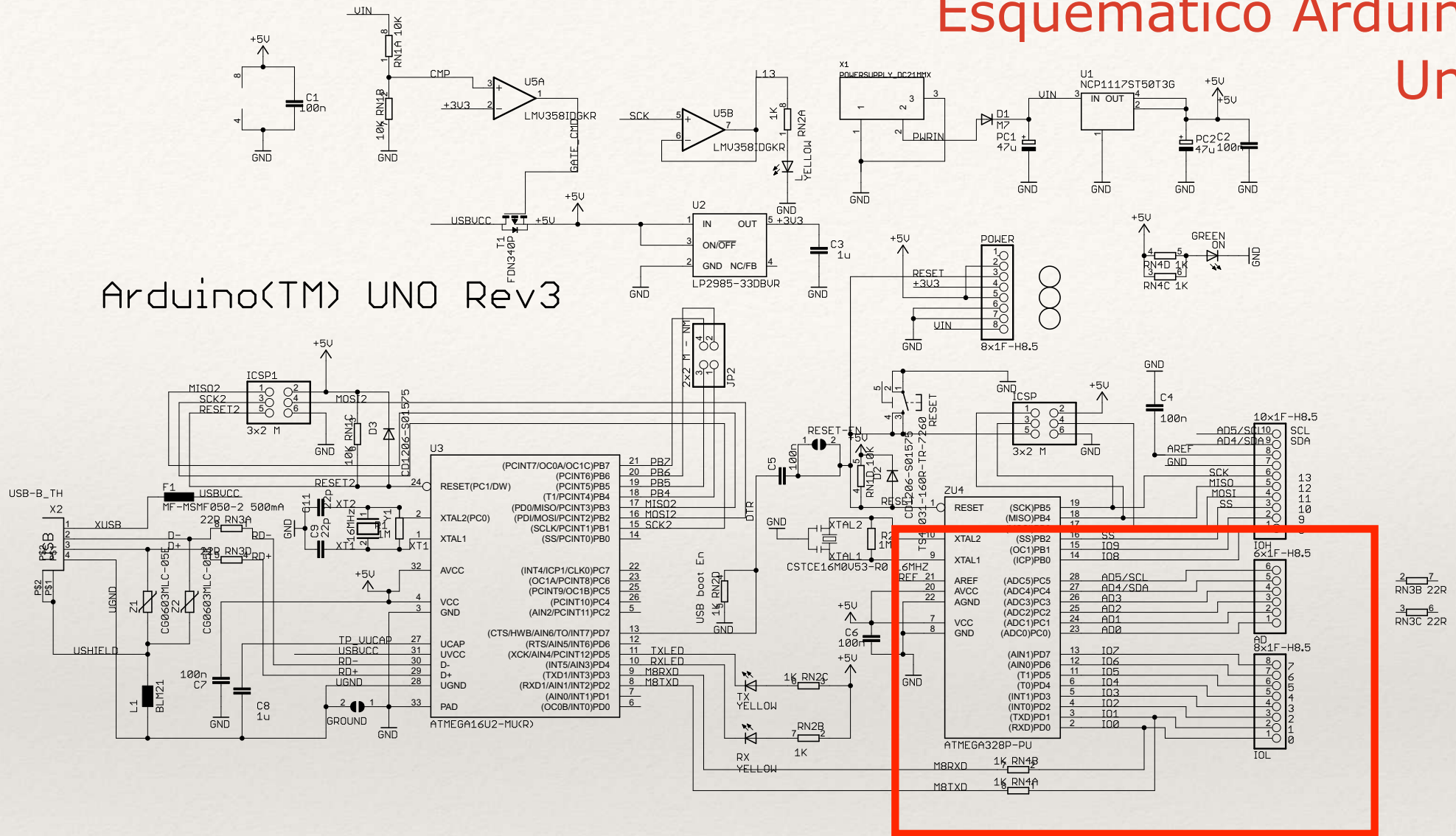
```
#define pinoBotao 2
#define pinoLed 5

void setup() {
  pinMode( pinoBotao, INPUT_PULLUP );
  pinMode( pinoLed, OUTPUT );
}

void loop() {
  int botao = digitalRead( pinoBotao );

  if( botao == LOW ) {
    digitalWrite( pinoLed, LOW );
  } else {
    digitalWrite( pinoLed, HIGH );
  }
}
```


Esquemático Arduino Uno

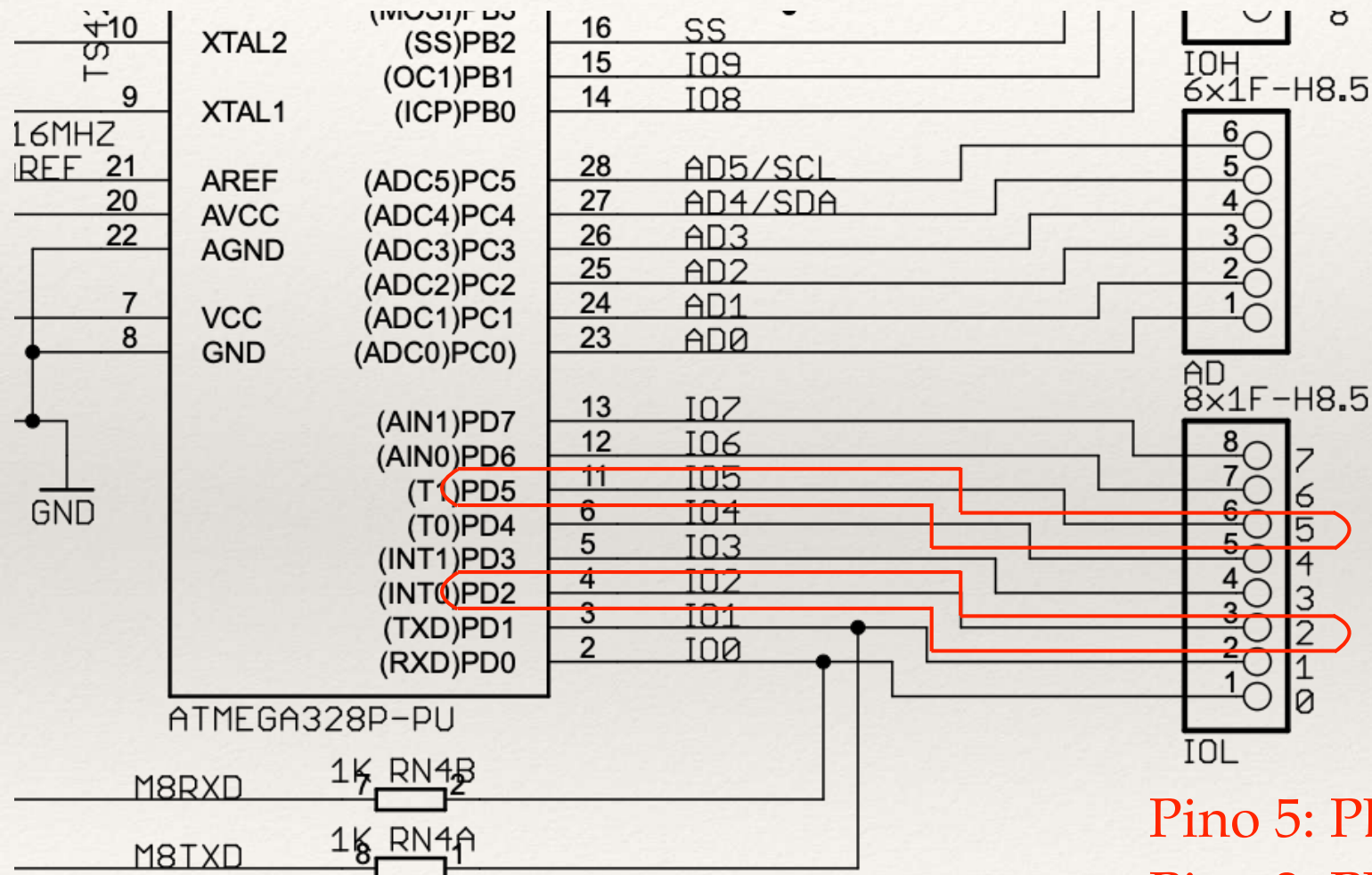


Reference Designs ARE PROVIDED "AS IS" AND "WITH ALL FAULTS. Arduino DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, REGARDING PRODUCTS, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Arduino may make changes to specifications and product descriptions at any time, without notice. The Customer must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Arduino reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The product information on the Web Site or Materials is subject to change without notice. Do not finalize a design with this information.

ARDUINO is a registered trademark.

Use of the ARDUINO name must be compliant with <http://www.arduino.cc/en/Main/Policy>

Registradores pinos 2, 5 ?

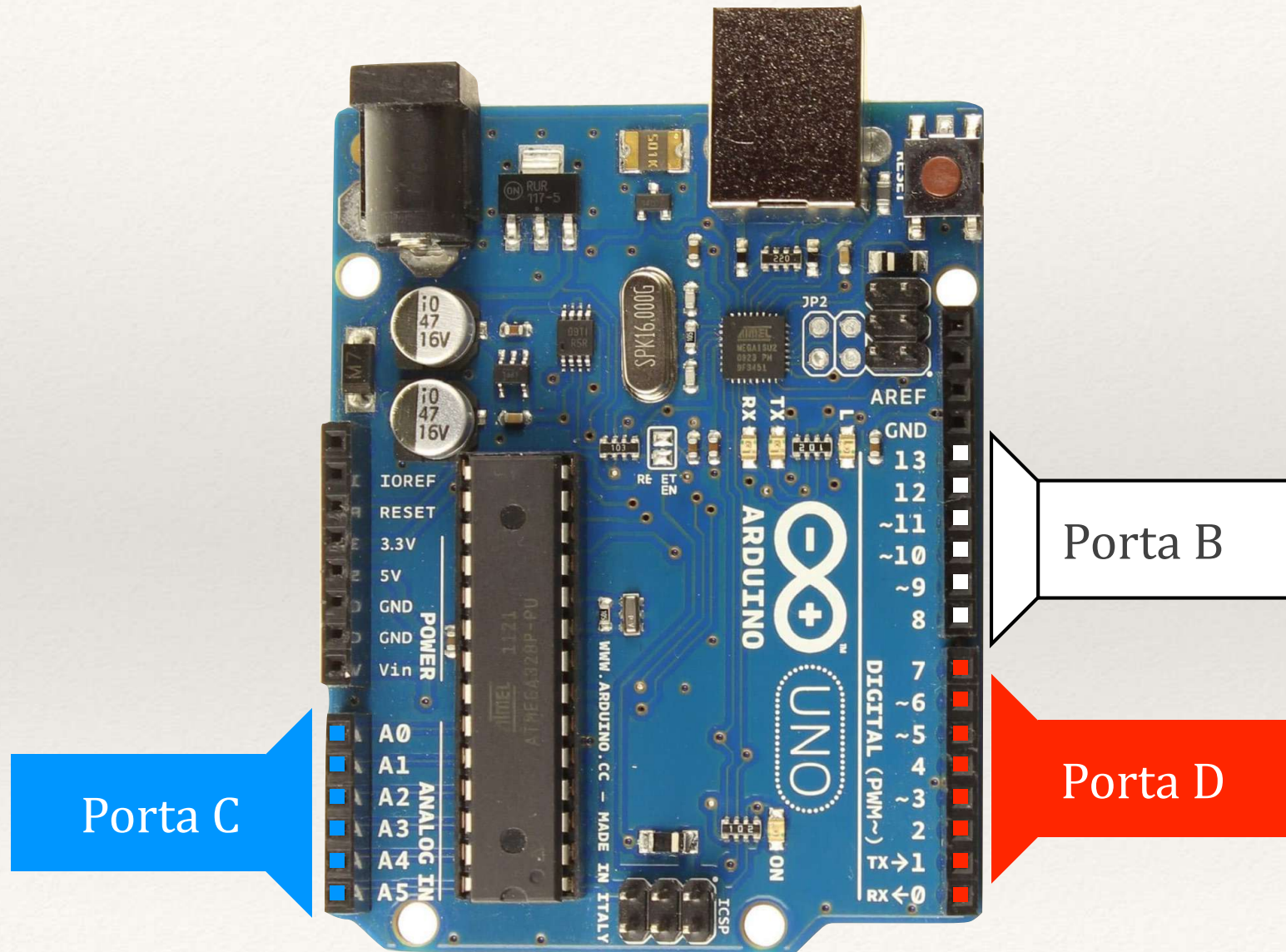


Pino 5: PD5

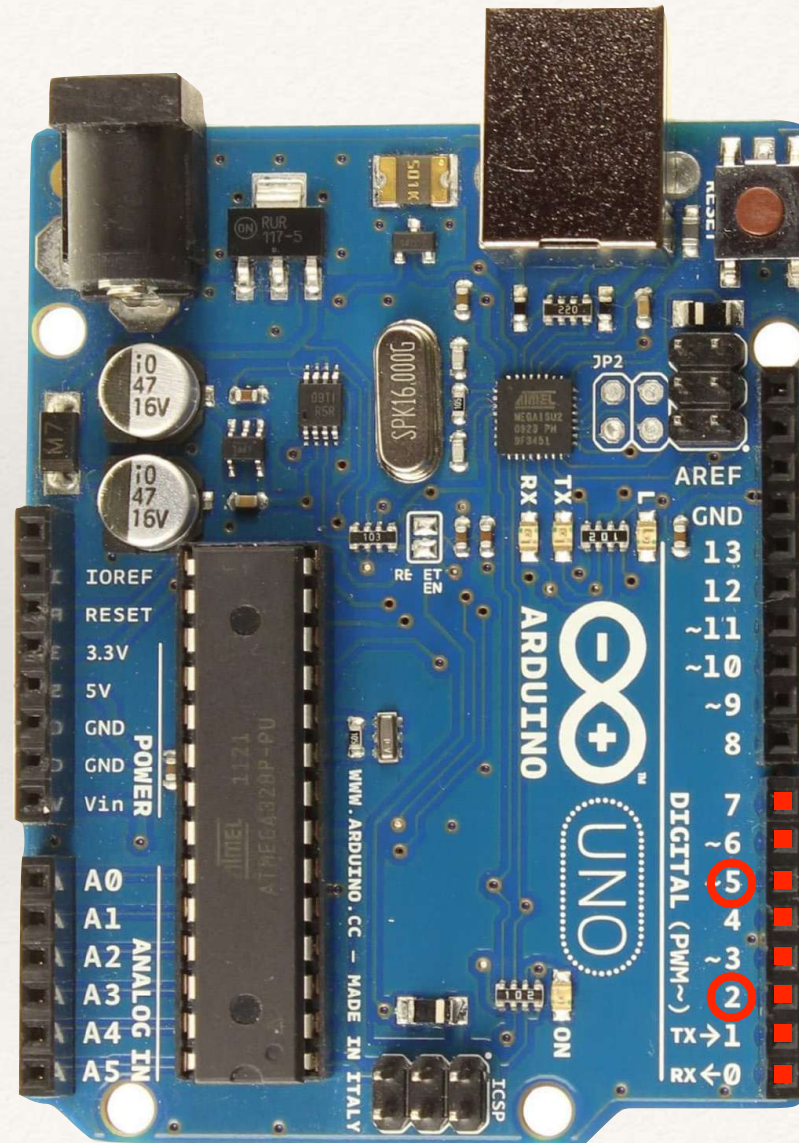
Pino 2: PD2

(ambos na porta D)

Arduíno: portas B, C e D



Arduíno: portas B, C e D



Porta D

Registadores - Porta D

PORTD – The Port D Data Register

Bit	7	6	5	4	3	2	1	0	
0x0B (0x2B)	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	PORTD
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	

DDRD – The Port D Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x0A (0x2A)	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	DDRD
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	

PIND – The Port D Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x09 (0x29)	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	PIND
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Registadores - Porta D

PORTD – The Port D Data Register

Bit	7	6	5	4	3	2	1	0	
0x0B (0x2B)	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	PORTD
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	

DDRD – The Port D Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x0A (0x2A)	DDRD7	DDRD6	DDRD5	DDRD4	DDRD3	DDRD2	DDRD1	DDRD0	DDRD
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	

O registrador DDR determina quais são os pinos que serão utilizados como entrada (valor 0) ou saída (valor 1)

PIND – The Port D Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x09 (0x29)	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	PIND
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Registadores - Porta D

PORTD - The Port D Data Register

Bit
0x0B (0x0B)
Read/Write
Initial Value

O registrador PORT determina, no caso de SAÍDA, qual o valor a ser apresentado; no caso de entrada, habilita o resistor interno de *PullUp* (1), ou desabilita (0)

DDRD - The Port D Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x0A (0x2A)	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	DDRD
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	

PIND - The Port D Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x09 (0x29)	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	PIND
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Registadores - Porta D

PORTD – The Port D Data Register

Bit	7	6	5	4	3	2	1	0	
0x0B (0x2B)	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	PORTD
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	

DDRD – The Port D Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x0A (0x2A)	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	DDRD
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	

PIND – The Port D Input Pins Address

	1	0	
0x0C (0x2C)	PIND1	PIND0	PIND
Read/Write	R	R	
Initial Value	N/A	N/A	

O registrador PIN, que deve ser tratado apenas para leitura, possui o status atual de cada pino

Acesso Binário - exemplo

Sketch define pino 5 como saída e pino 2 como entrada

1º passo: carregar 1 no bit 5 do registrador DDRD;

2º passo: carregar 1 no bit 2 do registrador PORTD;

Atribuindo o valor da palavra

Em binário:

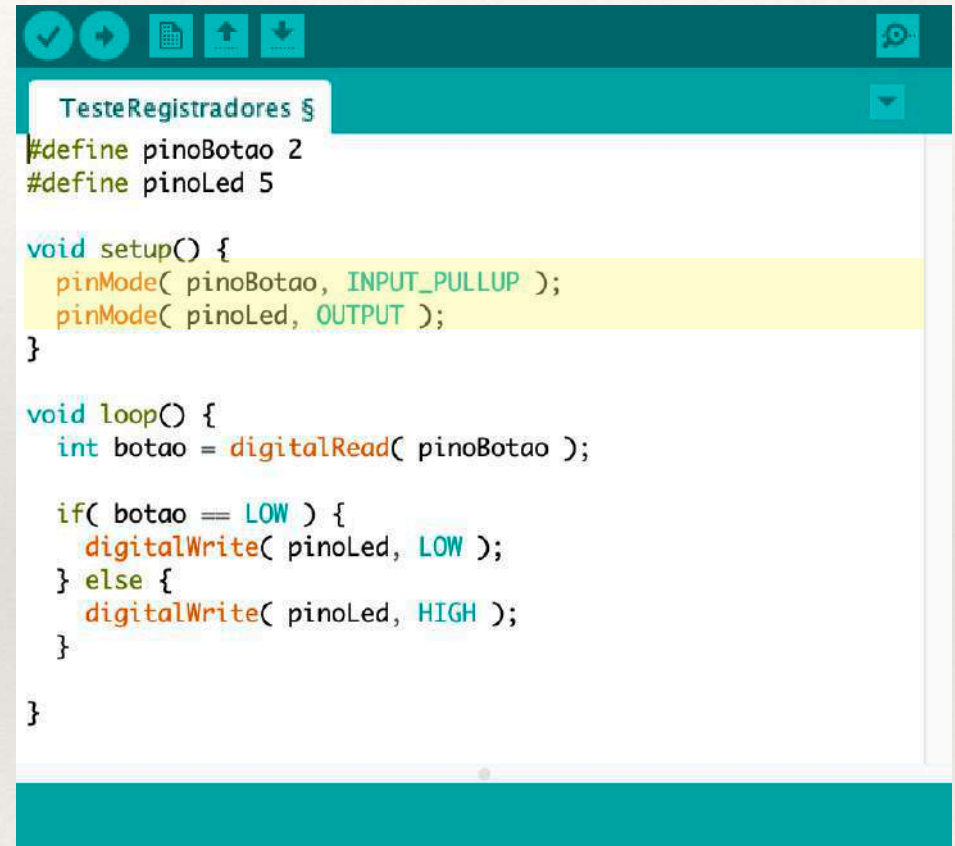
DDRD = B00100000;

PORTD = B00000100;

Ou, em decimal:

DDRD = 32;

PORTD = 4;

A screenshot of the Arduino IDE interface. The title bar shows 'TesteRegistradores \$'. The code editor contains the following C++ code:

```
#define pinoBotao 2
#define pinoLed 5

void setup() {
  pinMode( pinoBotao, INPUT_PULLUP );
  pinMode( pinoLed, OUTPUT );
}

void loop() {
  int botao = digitalRead( pinoBotao );

  if( botao == LOW ) {
    digitalWrite( pinoLed, LOW );
  } else {
    digitalWrite( pinoLed, HIGH );
  }
}
```


Acesso Binário - exemplo

Atribuir o valor à palavra pode invalidar atribuições feitas anteriormente a outros pinos !

O correto é atribuir valor ao BIT

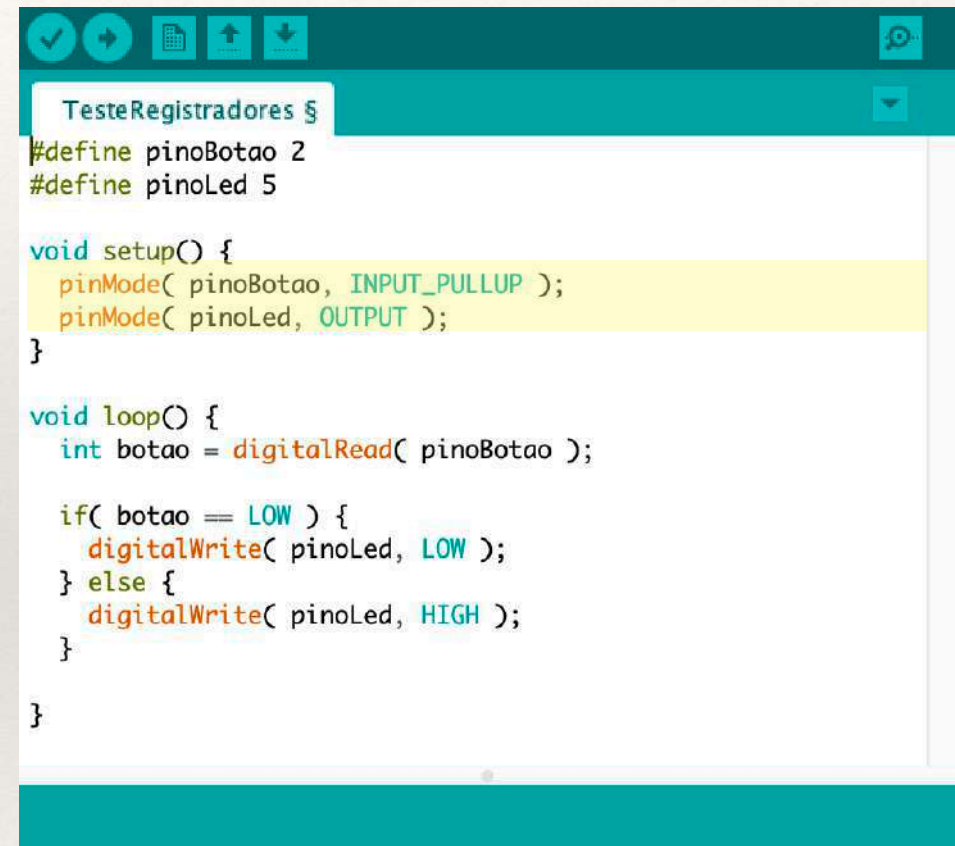
Operação OU com conteúdo anterior:

```
DDRD  |= B00100000;
```

```
PORTD |= B00000100;
```

Embora funcione, este procedimento força a criação de uma constante, e de uma variável provisória (pouco eficiente em memória e performance);

A outra forma, como vimos, utiliza a operação de deslocamento de bit.



```
TesteRegistradores $
#define pinoBotao 2
#define pinoLed 5

void setup() {
  pinMode( pinoBotao, INPUT_PULLUP );
  pinMode( pinoLed, OUTPUT );
}

void loop() {
  int botao = digitalRead( pinoBotao );

  if( botao == LOW ) {
    digitalWrite( pinoLed, LOW );
  } else {
    digitalWrite( pinoLed, HIGH );
  }
}
```

Acesso Binário - exemplo

Ativando o bit 5 em DDRD:

Criamos uma palavra com o bit desejado ativo:

```
( 1 << pinoLed )
```

0	0	0	0	0	0	0	1

Acesso Binário - exemplo

Ativando o bit 5 em DDRD:

Criamos uma palavra com o bit desejado ativo:

`(1<<pinoLed)`

Fazemos uma operação OU (|) com o valor atual de DDRD:

`DDRD =( 1 << pinoLed ) | DDRD;`

0	0	1	0	0	0	0	0

OU

?	?	?	?	?	?	?	?

Acesso Binário - exemplo

Ativando o bit 5 em DDRD:

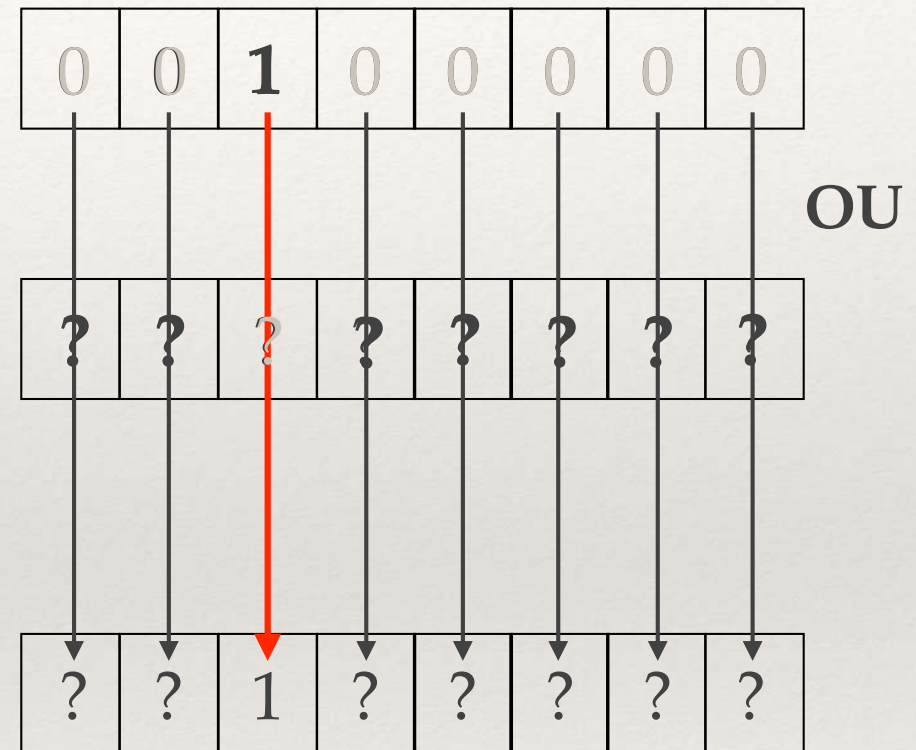
Criamos uma palavra com o bit desejado ativo:

```
(1<<pinoLed)
```

Fazemos uma operação OU (|) com o valor atual de DDRD:

```
DDRD =( 1 << pinoLed ) | DDRD;
```

Todos os bits de DDRD são mantidos, exceto o bit 5, que assume o valor 1.



Acesso Binário - exemplo

E se fosse ao contrário?

Desativando o bit 5 em DDRD:

Criamos uma palavra com o bit desejado ativo:
(1<<pinoLed)

0	0	1	0	0	0	0	0

?	?	?	?	?	?	?	?

Acesso Binário - exemplo

E se fosse ao contrário?

Desativando o bit 5 em DDRD:

Criamos uma palavra com o bit desejado ativo:

`(1<<pinoLed)`

Invertemos a palavra bit-a-bit (`~`);

1	1	0	1	1	1	1	1

?	?	?	?	?	?	?	?

Acesso Binário - exemplo

E se fosse ao contrário?

Desativando o bit 5 em DDRD:

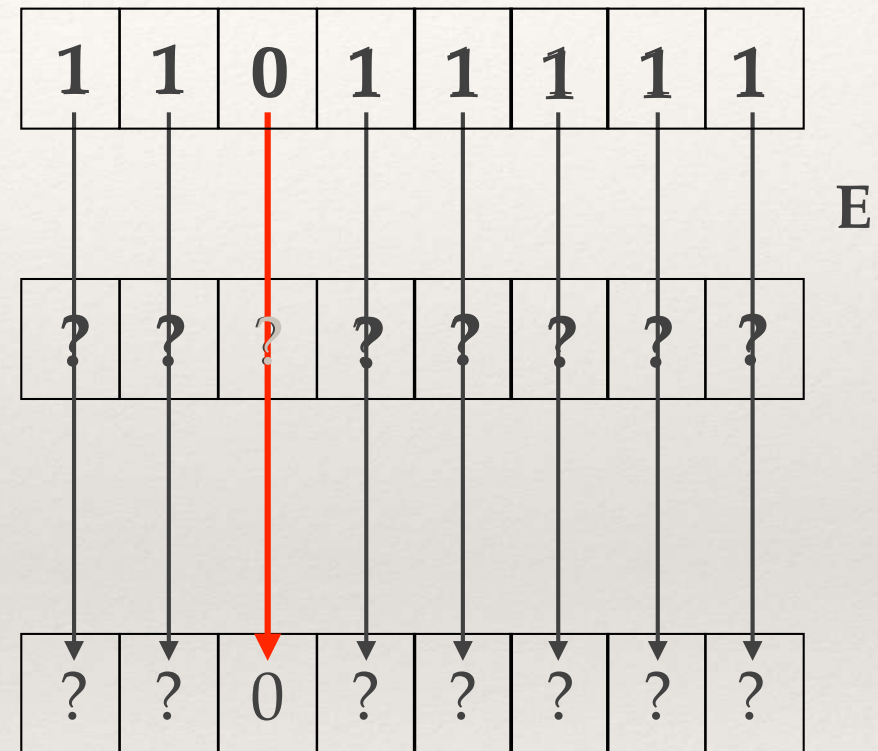
Criamos uma palavra com o bit desejado ativo:
(1<<pinoLed)

Invertemos a palavra bit-a-bit (~);

Fazemos uma operação E (&) com o valor atual de DDRD:

```
DDRD = ~( 1 << pinoLed ) & DDRD;
```

Todos os bits de DDRD são mantidos, exceto o bit 5, que assume o valor 0.



Acesso Binário - exemplo

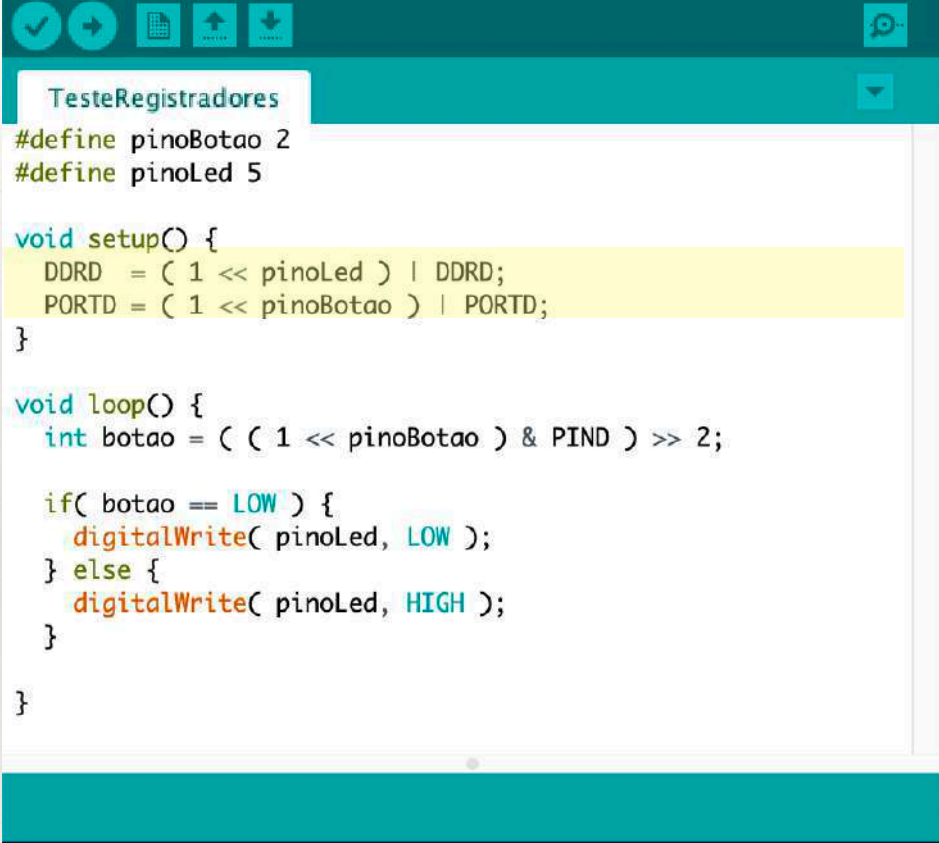
Para ativar o bit 2 de PORTD em DDRD, o processo é similar

Criamos uma palavra com o bit desejado ativo:

$(1 \ll \text{pinoBotao})$

Fazemos uma operação OU (|) com o valor atual de PORTD:

$\text{PORTD} = (1 \ll \text{pinoBotao}) \mid \text{PORTD};$



```
TesteRegistradores
#define pinoBotao 2
#define pinoLed 5

void setup() {
  DDRD = ( 1 << pinoLed ) | DDRD;
  PORTD = ( 1 << pinoBotao ) | PORTD;
}

void loop() {
  int botao = ( ( 1 << pinoBotao ) & PIND ) >> 2;

  if( botao == LOW ) {
    digitalWrite( pinoLed, LOW );
  } else {
    digitalWrite( pinoLed, HIGH );
  }
}
```

Acesso Binário - exemplo

Para leitura do Botão, utilizamos o registro PIND

Criamos uma palavra colocando 1 no bit a ser lido:

```
( 1 << pinoBotao )
```

Fazemos uma operação E (&) com o valor atual de PIND:

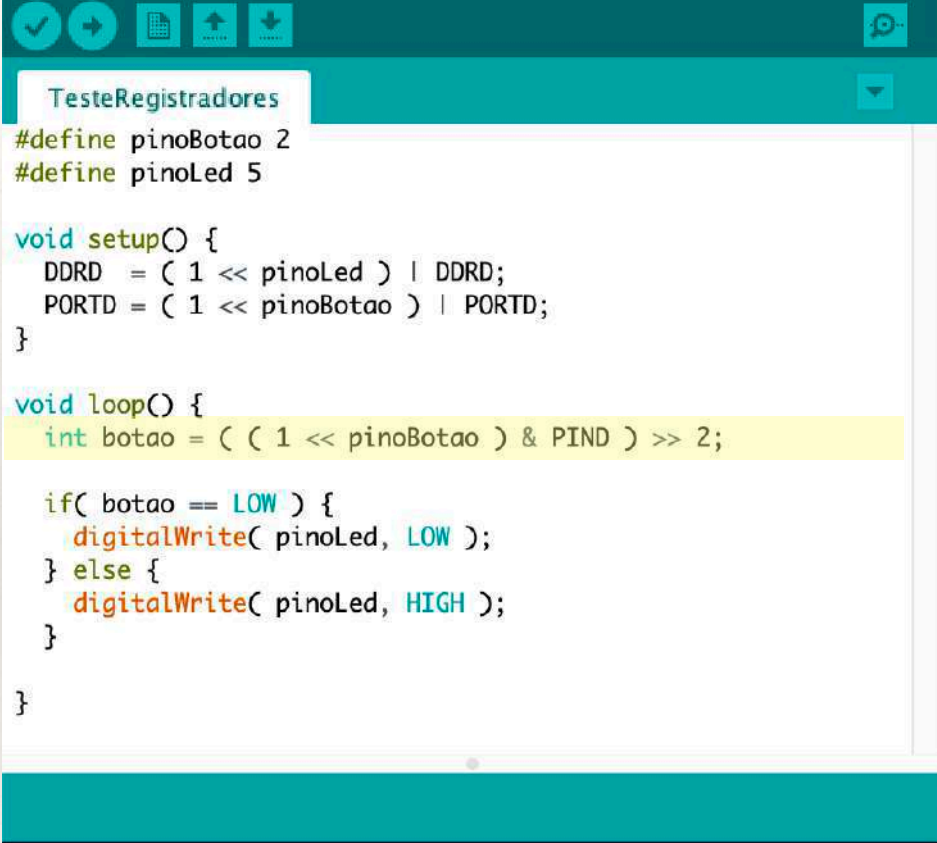
```
( 1 << pinoBotao ) & PIND;
```

Revertemos o deslocamento para colocar o bit testado na posição zero:

```
botao = ( ( 1 << pinoBotao ) & PIND ) >> 2;
```

O valor será 1 (verdadeiro) ou 0 (falso), a depender do valor lido na posição desejada do PIND;

A reversão é necessária? (O que é "verdadeiro" ?)



```
TesteRegistradores

#define pinoBotao 2
#define pinoLed 5

void setup() {
  DDRD = ( 1 << pinoLed ) | DDRD;
  PORTD = ( 1 << pinoBotao ) | PORTD;
}

void loop() {
  int botao = ( ( 1 << pinoBotao ) & PIND ) >> 2;

  if( botao == LOW ) {
    digitalWrite( pinoLed, LOW );
  } else {
    digitalWrite( pinoLed, HIGH );
  }
}
```


Acesso Binário - exemplo

Para acender ou apagar o LED, os processos são bem similares

Criamos uma palavra com o bit desejado ativo:

```
( 1 << pinoLed )
```

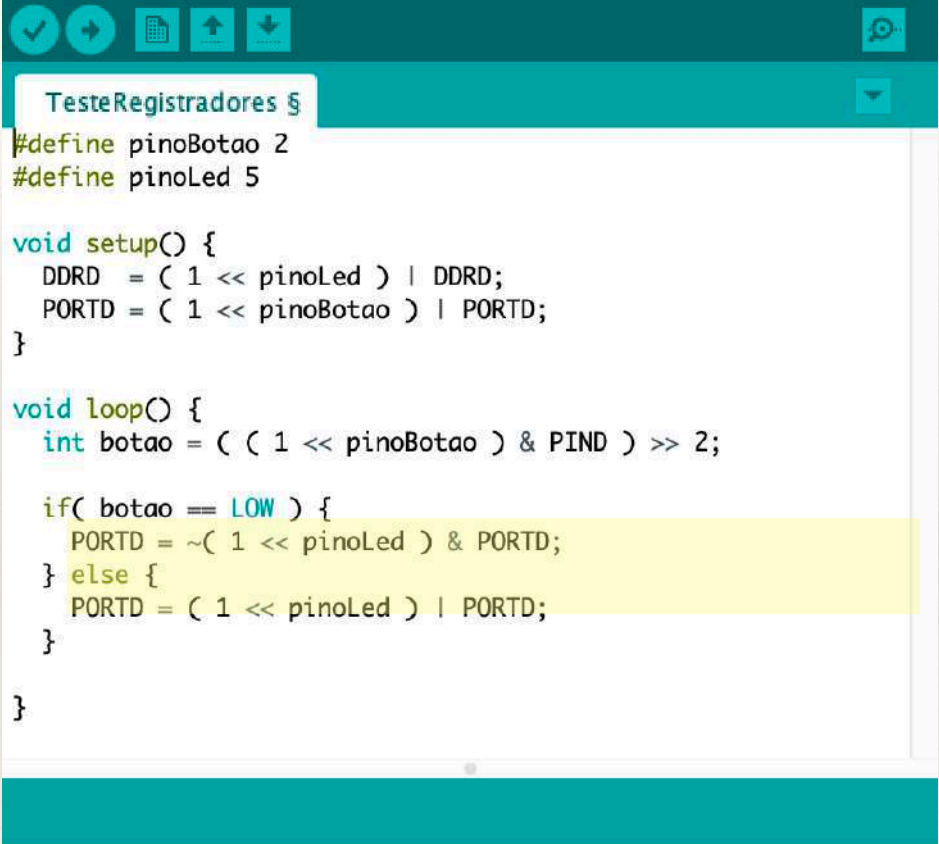
Fazemos uma operação OU (|) com o valor atual de PORTD:

```
if( botão == LOW ) {  
    PORTD = ~( 1 << pinoLed ) & PORTD;  
} else {  
    PORTD = ( 1 << pinoLed ) | PORTD;  
}
```

Após compilar, o que você notou?

Impressionante, não?

Vale a pena? (Lembrar da portabilidade)



```
TesteRegistradores 5  
#define pinoBotao 2  
#define pinoLed 5  
  
void setup() {  
    DDRD = ( 1 << pinoLed ) | DDRD;  
    PORTD = ( 1 << pinoBotao ) | PORTD;  
}  
  
void loop() {  
    int botao = ( ( 1 << pinoBotao ) & PIND ) >> 2;  
  
    if( botao == LOW ) {  
        PORTD = ~( 1 << pinoLed ) & PORTD;  
    } else {  
        PORTD = ( 1 << pinoLed ) | PORTD;  
    }  
}
```

Eventos Externos/Internos

Sistemas embarcados tipicamente precisam responder a eventos;

Eventos são sinais elétricos;

Eventos exigem código específico;

Tratamento:

Rotina verifica ocorrência do evento → **I/O programado !**

Rotina é disparada pelo evento → **Interrupções**

Rotina acessa registradores modificados pelo evento → **DMA (*Direct Memory Access*)**

O que é uma “Interrupção” ?



Interrupções

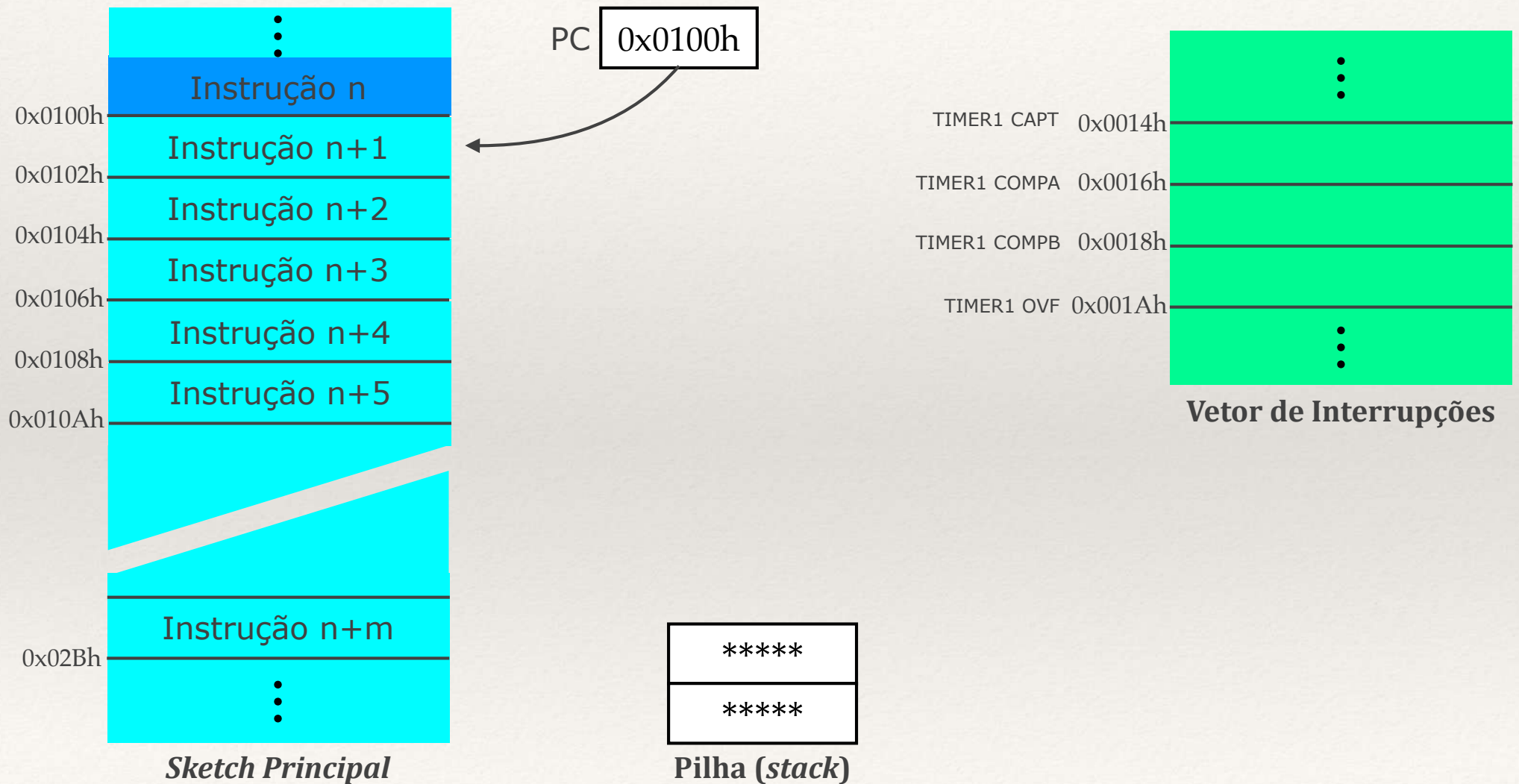
Uma das formas mais eficazes de tratar eventos;

Podem ser acionadas por eventos internos (ex. *timers*) ou externos (ex. sinais elétricos de entrada);

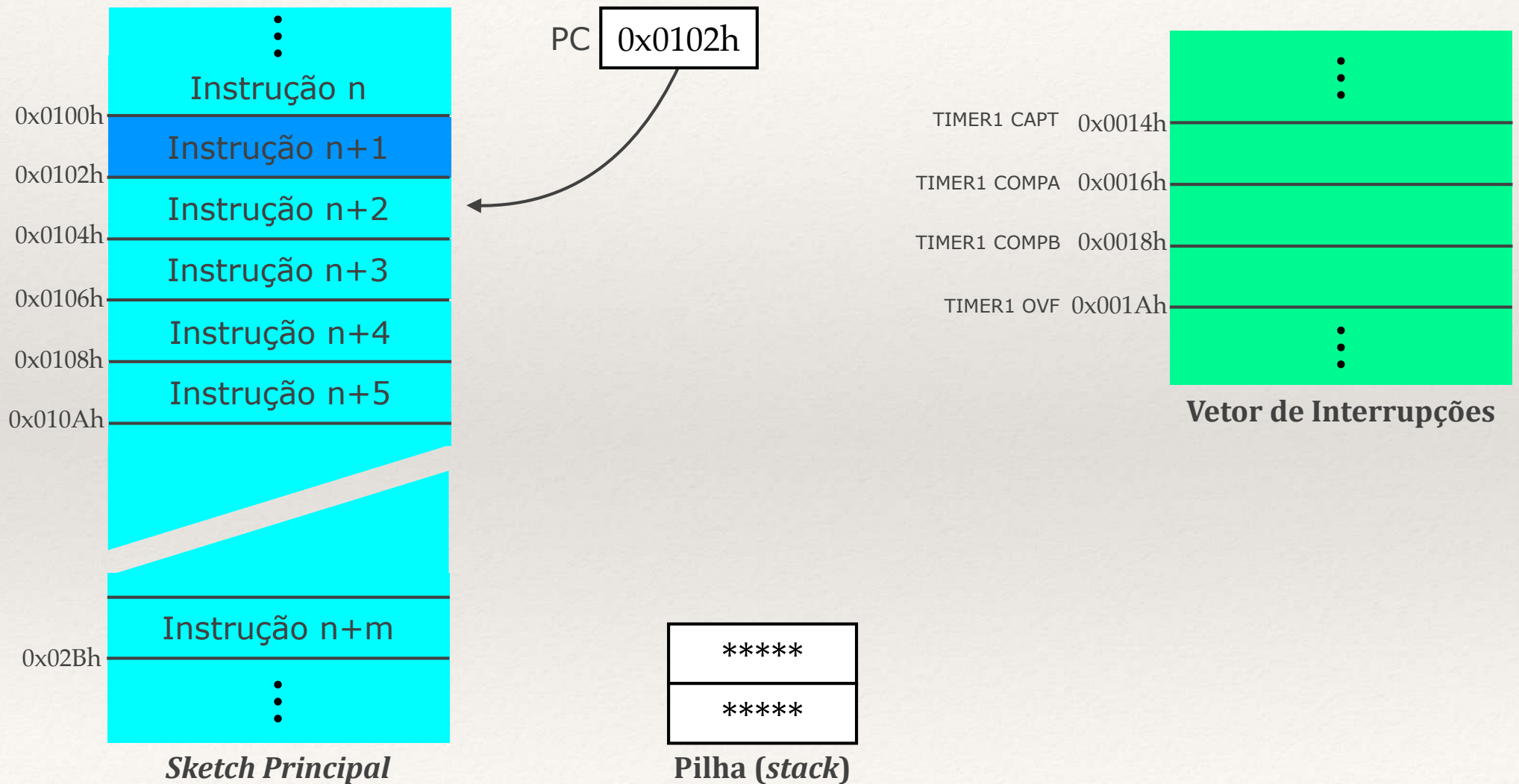
Provocam o desvio de execução do *sketch* para uma *Interrupt Service Routine* (ISR) específica. Ao término da ISR, a execução retorna ao ponto original;

A ISR deve ser construída com alguns cuidados especiais.

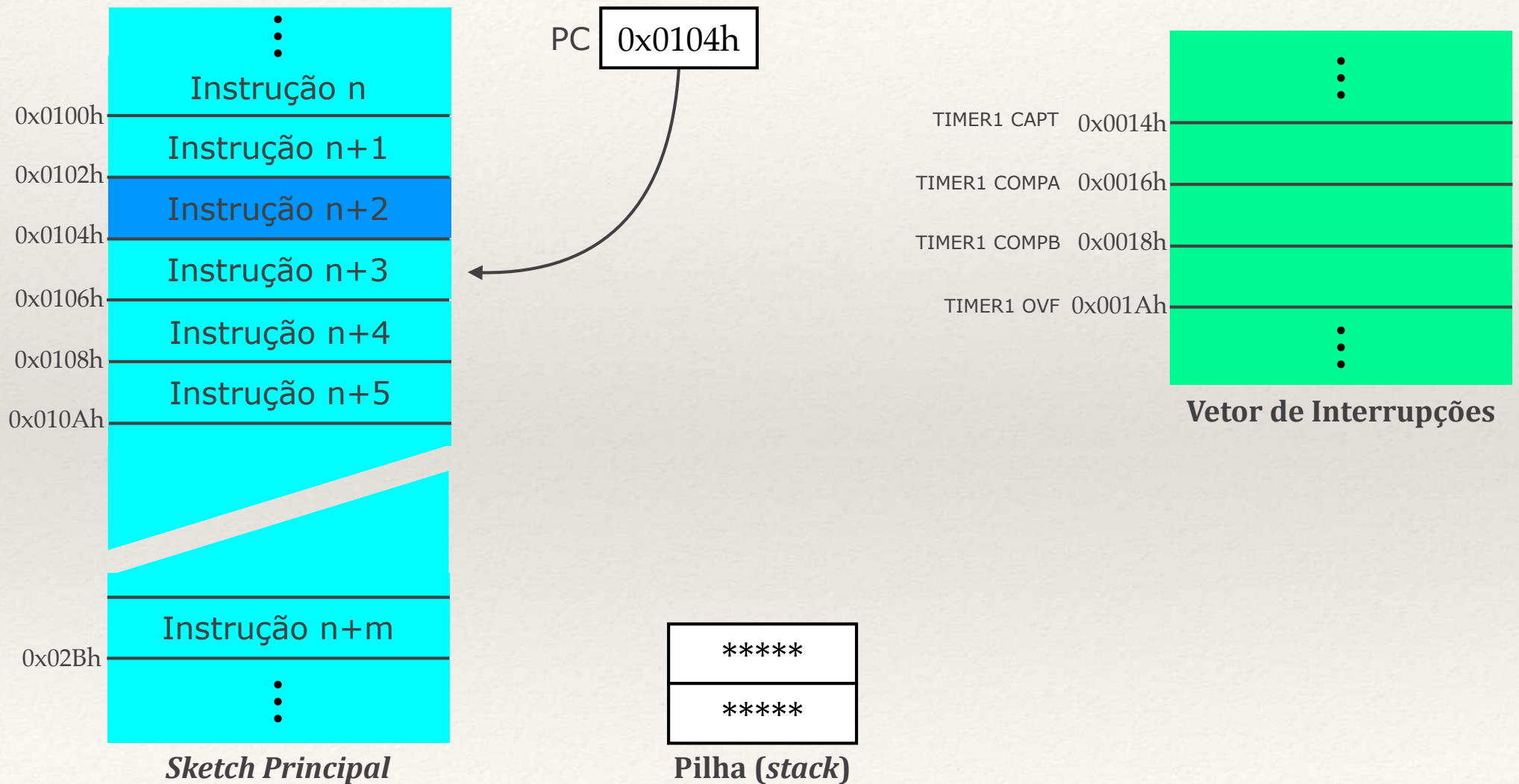
Interrupções



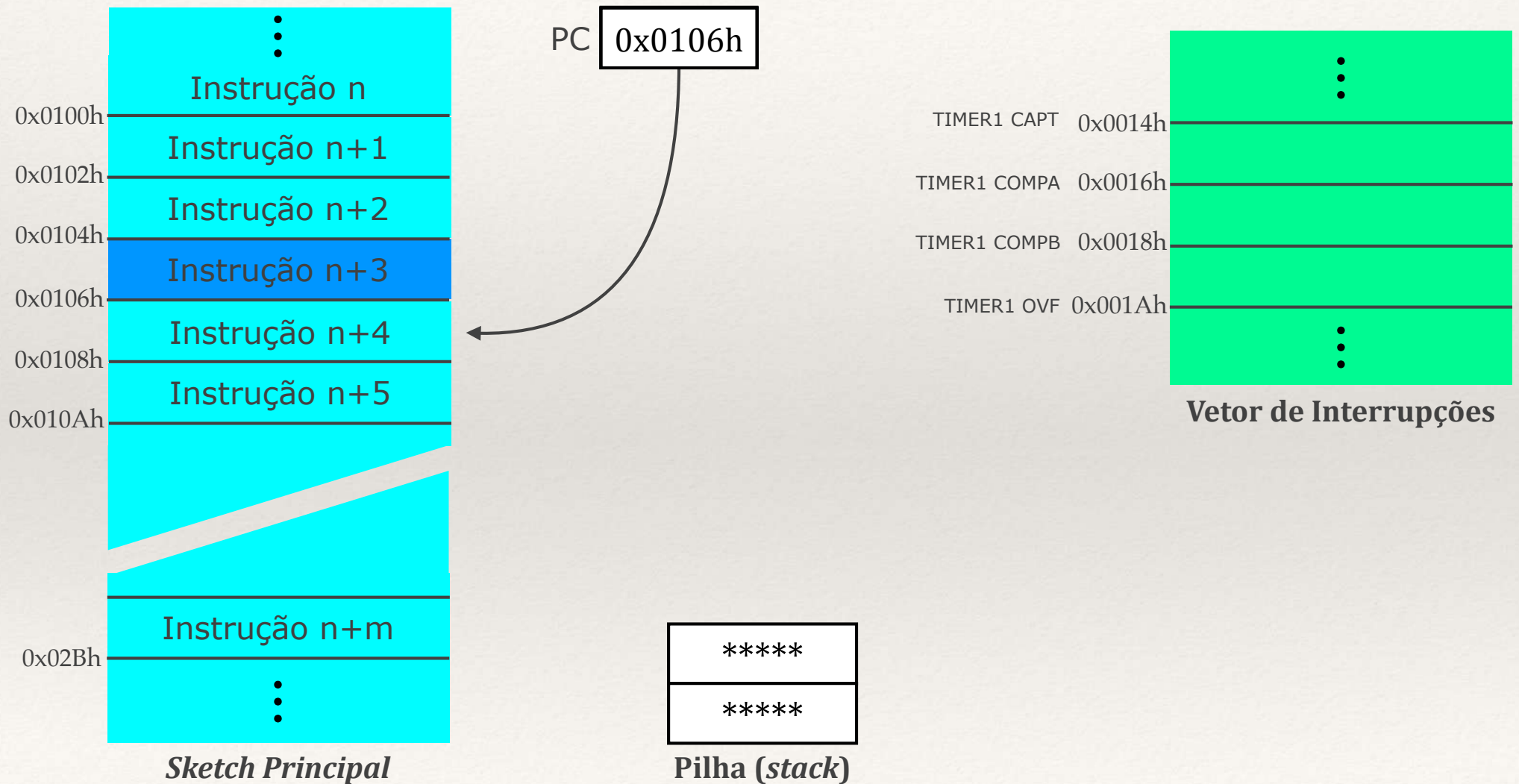
Interrupções



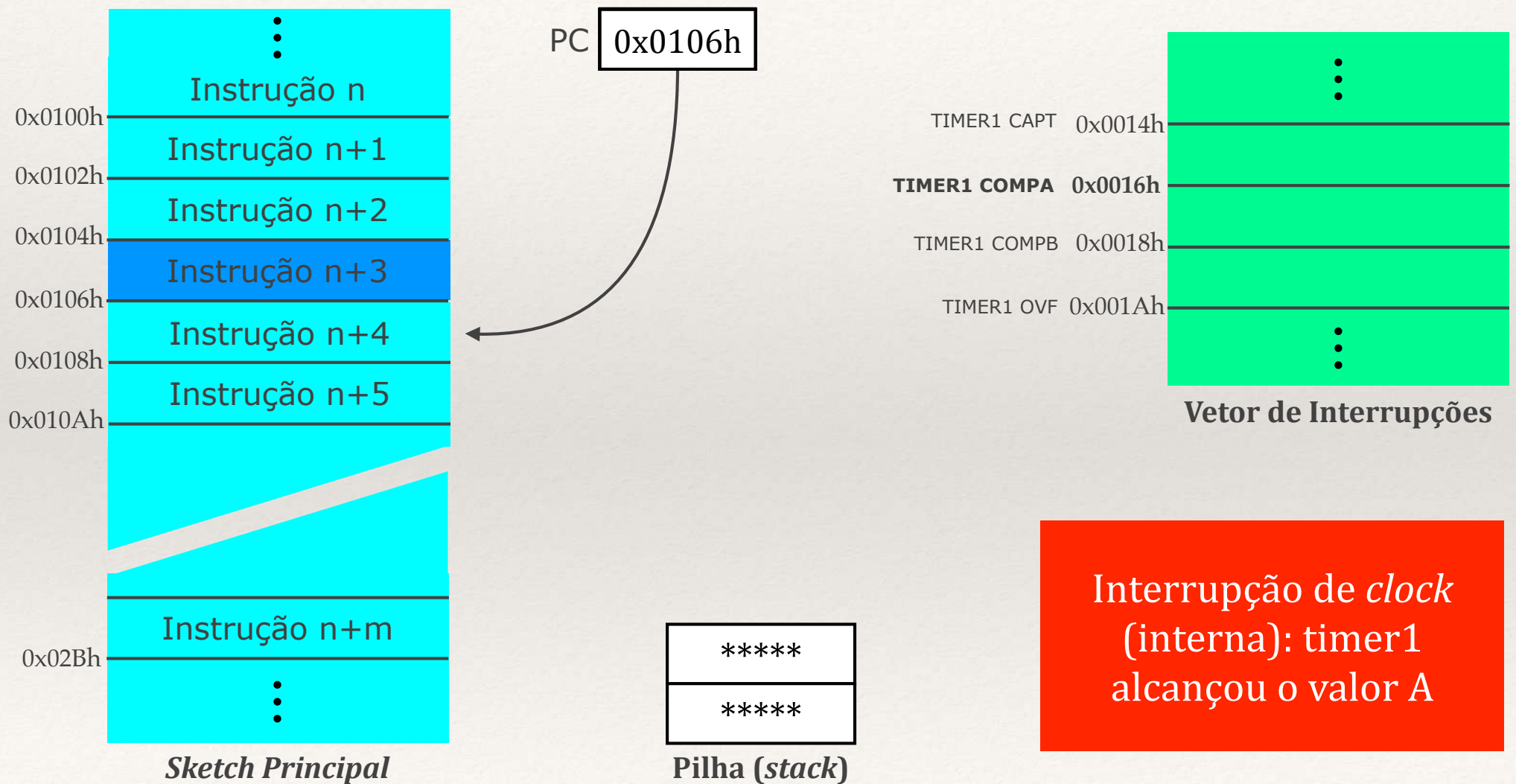
Interrupções



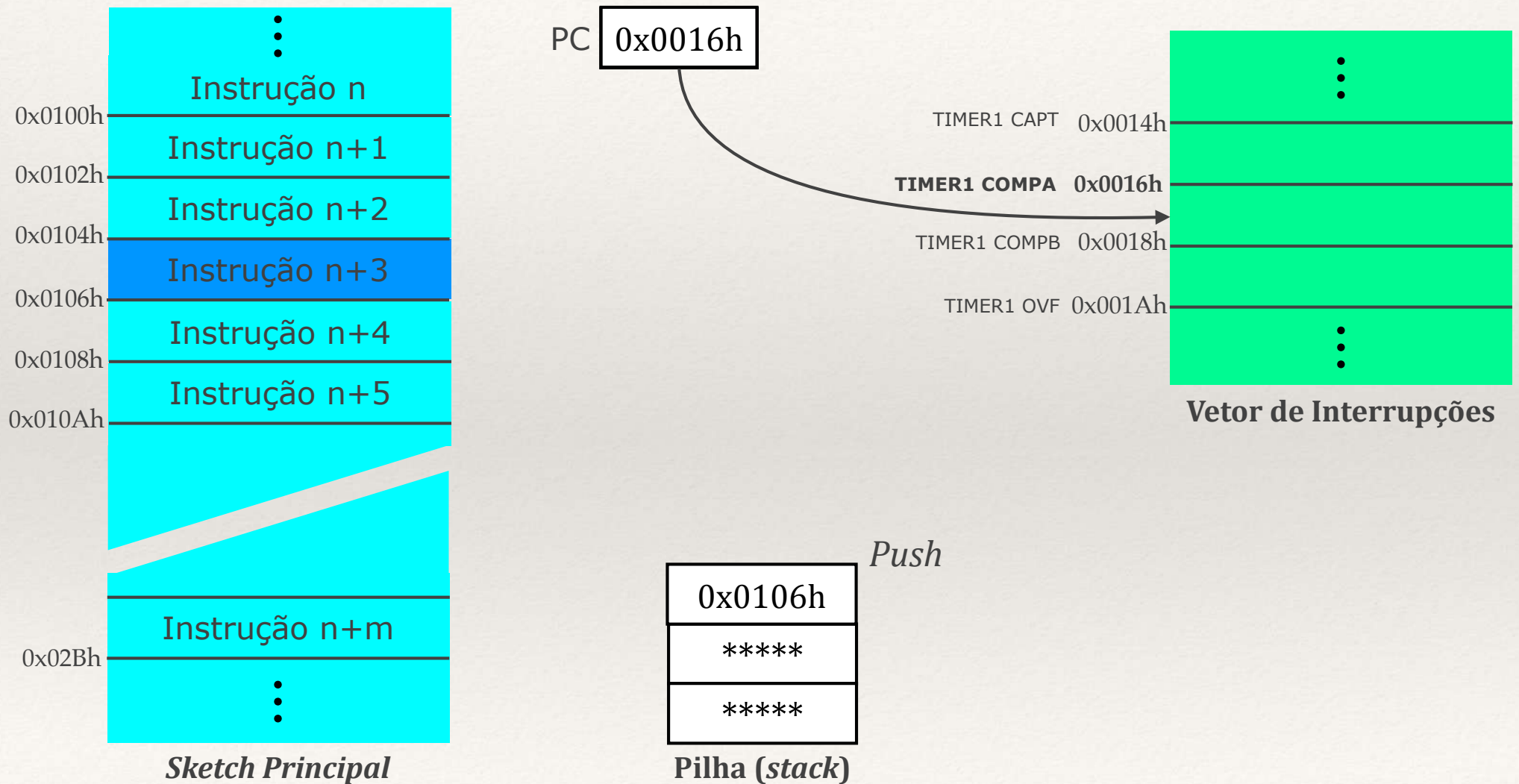
Interrupções



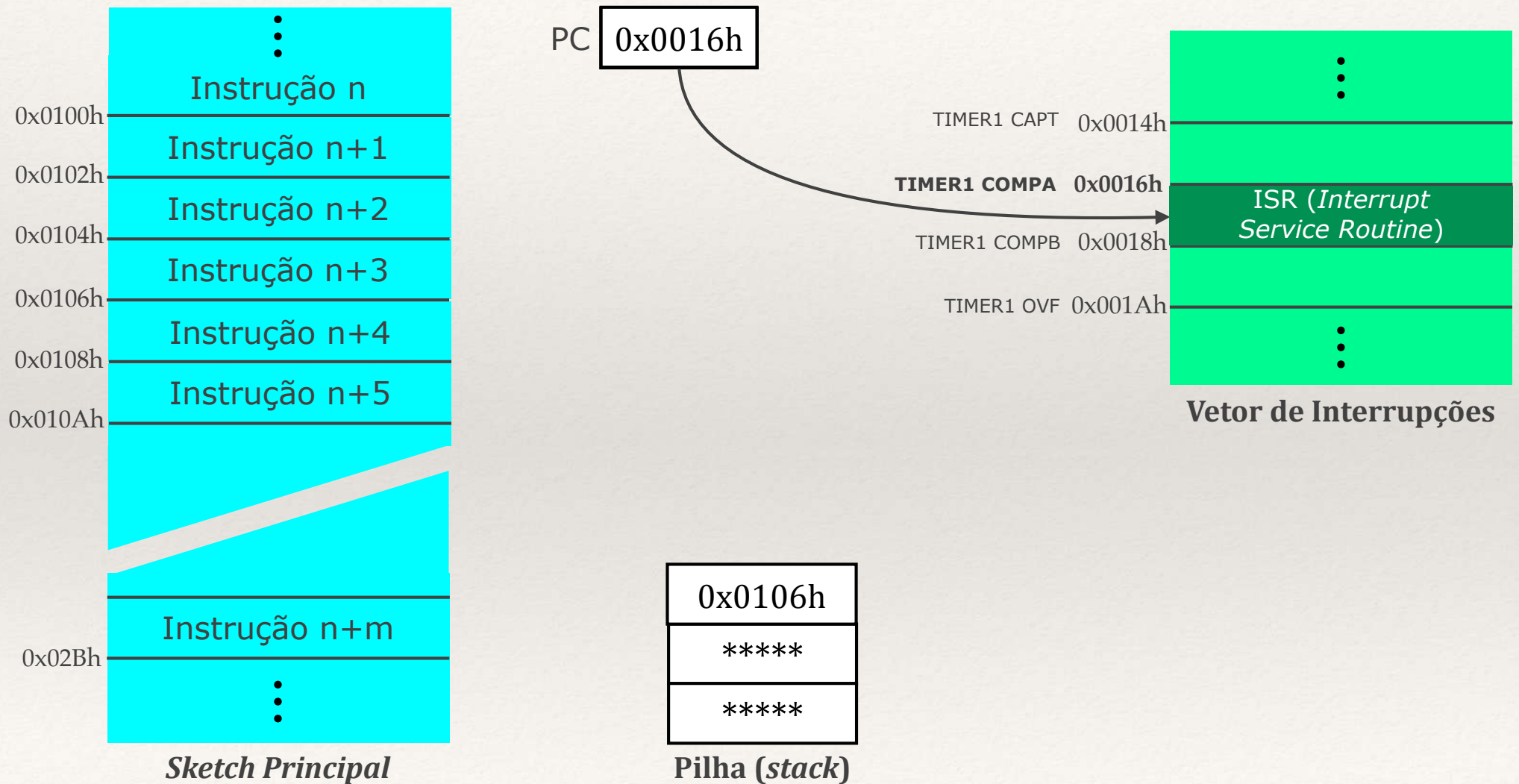
Interrupções



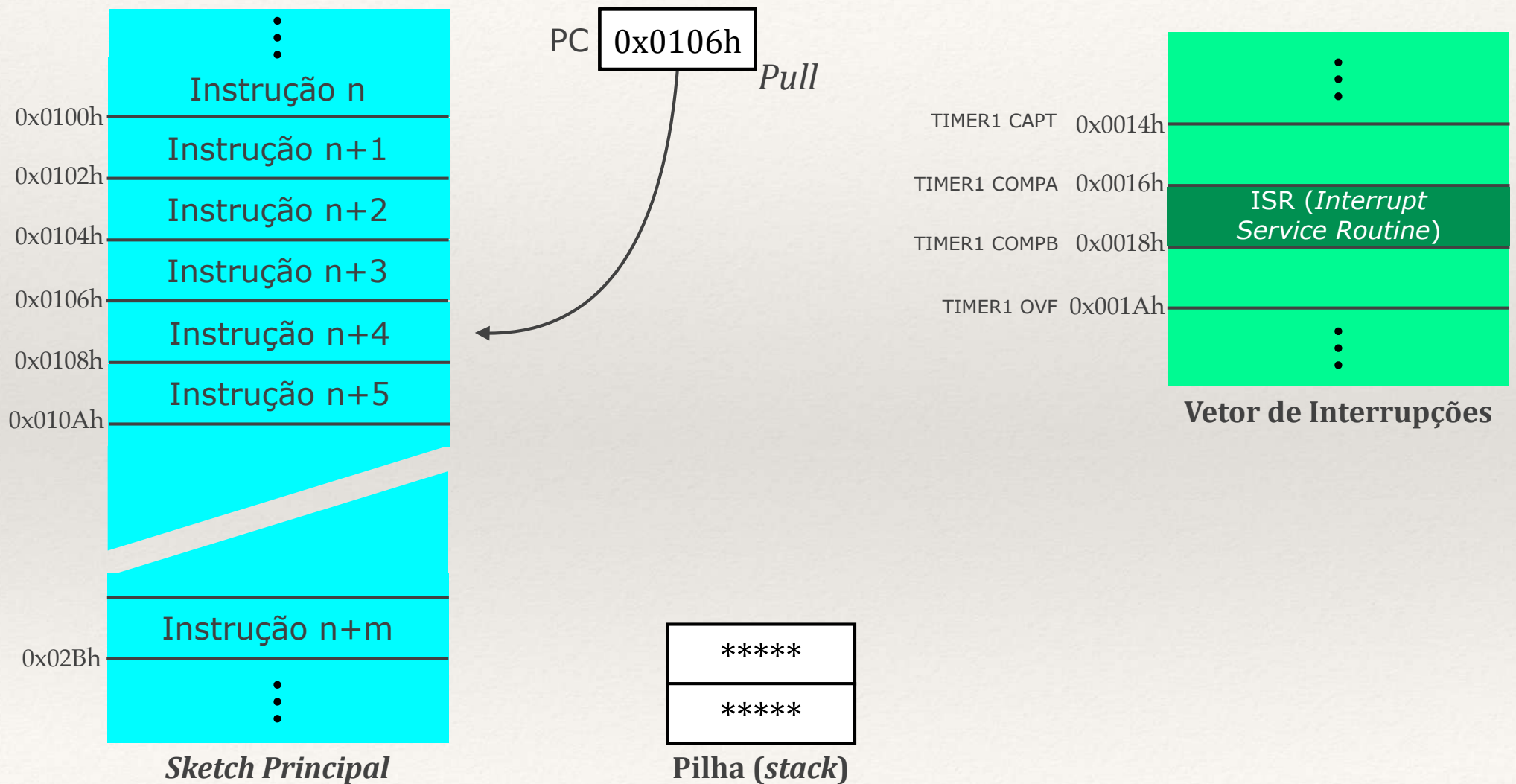
Interrupções



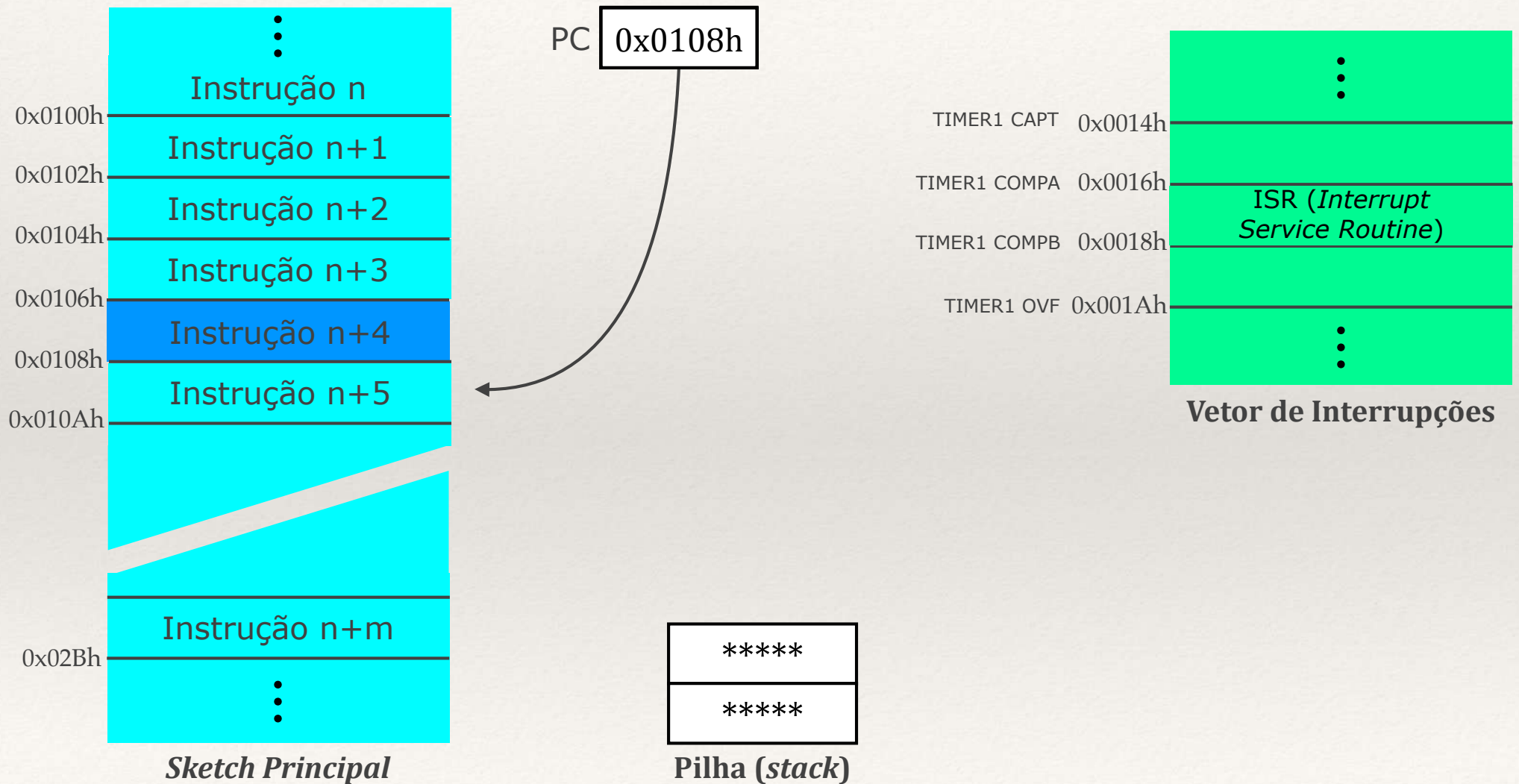
Interrupções



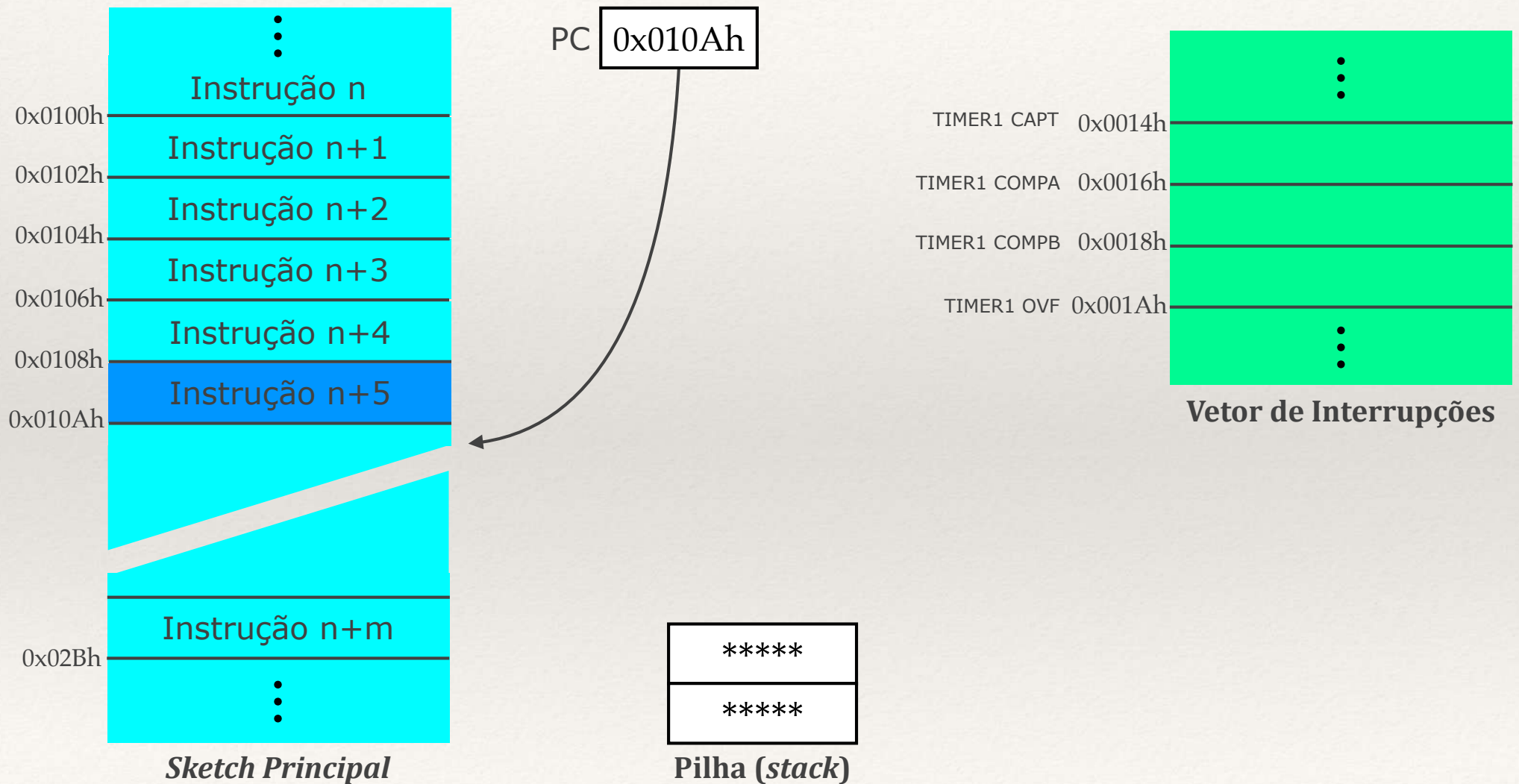
Interrupções



Interrupções



Interrupções



Interrupções nos MCUs

Cada fabricante pode implementar o tratamento de interrupções de forma específica;

Características das Interrupções

Vetorizada: cada tipo de evento está associado a um endereço específico

Normalmente mais eficiente,;

Nas não vetorizadas, é necessário consultar e tratar os *falas* (PIC);

Priorizável: trata ocorrências simultâneas;

Mascarável: programador pode desabilitar

Podem haver diferentes tipos. NMI= *Non-Masked Interrupts* (RESET ?)

Configurável: endereços das ISC podem ser alterados (ATMega).

Vamos usar o ATMega328P como exemplo

Aula 05

Interrupções ATmega328P

O ATmega possui um vetor de interrupções

Cada linha associa uma interrupção à sua ISR;

Temos opções para tratamento de interrupções internas e externas;

A tabela ao lado identifica alguns que estão associados a 3 diferentes *timers* (0, 1 e 2);

Em todos há 2 funções de comparação, e uma de *overflow*.

Vector	Program Address	Source	Interrupt Definition
1	0	RESET	External pin, power-on reset, brown-out reset and watchdog system reset
2	2	INT0	External interrupt request 0
3	4	INT1	External interrupt request 1
4	6	PCINT0	Pin change interrupt request 0
5	8	PCINT1	Pin change interrupt request 1
6	A	PCINT2	Pin change interrupt request 2
7	C	WDT	Watchdog time-out interrupt
8	E	TIMER2 COMPA	Timer/Counter2 compare match A
9	10	TIMER2 COMPB	Timer/Counter2 compare match B
10	12	TIMER2 OVF	Timer/Counter2 overflow
11	14	TIMER1 CAPT	Timer/Counter1 capture event
12	16	TIMER1 COMPA	Timer/Counter1 compare match A
13	18	TIMER1 COMPB	Timer/Counter1 compare match B
14	1A	TIMER1 OVF	Timer/Counter1 overflow
15	1C	TIMER0 COMPA	Timer/Counter0 compare match A
16	1E	TIMER0 COMPB	Timer/Counter0 compare match B
17	20	TIMER0 OVF	Timer/Counter0 overflow
18	22	SPI, STC	SPI serial transfer complete
19	24	USART, RX	USART Rx complete
20	26	USART, UDRE	USART, data register empty
21	28	USART, TX	USART, Tx complete
22	2A	ADC	ADC conversion complete
23	2C	EE READY	EEPROM ready
24	2E	ANALOG COMP	Analog comparator
25	30	TWI	2-wire serial interface
26	32	SPM READY	Store program memory ready

Interrupções ATmega328P

Os 3 *timers* também estão associados a registradores específicos

Registradores são modificados automaticamente, sem a necessidade de código específico;

Modificações dependem direta ou indiretamente do Clock do sistema.

30. Register Summary (Continued)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
(0xB4)	OCR2B	Timer/Counter2 output compare register B								131
(0xB3)	OCR2A	Timer/Counter2 output compare register A								131
(0xB2)	TCNT2	Timer/Counter2 (8-bit)								131
0x28 (0x48)	OCR0B	Timer/Counter0 output compare register B								
0x27 (0x47)	OCR0A	Timer/Counter0 output compare register A								
0x26 (0x46)	TCNT0	Timer/Counter0 (8-bit)								
(0x8B)	OCR1BH	Timer/Counter1 - Output compare register B high byte								111
(0x8A)	OCR1BL	Timer/Counter1 - Output compare register B low byte								111
(0x89)	OCR1AH	Timer/Counter1 - Output compare register A high byte								111
(0x88)	OCR1AL	Timer/Counter1 - Output compare register A low byte								111
(0x87)	ICR1H	Timer/Counter1 - Input capture register high byte								112
(0x86)	ICR1L	Timer/Counter1 - Input capture register low byte								112
(0x85)	TCNT1H	Timer/Counter1 - Counter register high byte								111
(0x84)	TCNT1L	Timer/Counter1 - Counter register low byte								111

} 16 bits

Interrupções ATmega328P

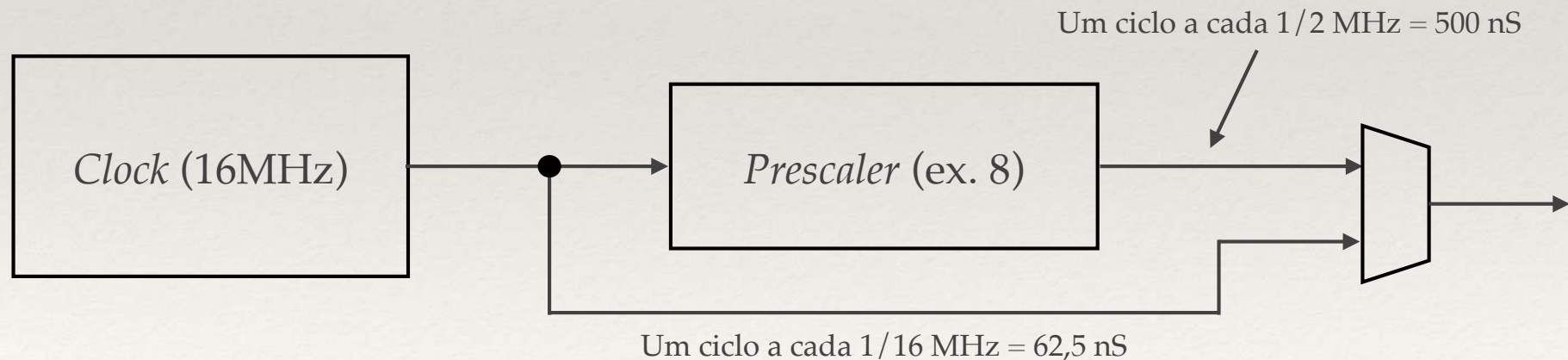
No Arduíno Uno, o clock é de 16 MHz

Cada microcontrolador tem sua própria taxa de clock;

O clock pode ser dividido por valores pré-definidos (*prescaler*);

O clock pode ter seu valor comparado para gerar interrupções .

Em algumas situações, os contadores podem ser configurados para contar eventos externos.



Interrupções ATmega328P

No Arduíno Uno, o clock é de 16 MHz

Cada microcontrolador tem sua própria taxa de clock;

O clock pode ser dividido por valores pré-definidos (*prescaler*);

O clock pode ter seu valor comparado para gerar interrupções .

Timer	Tamanho	Interrupções Possíveis	Uso no Arduíno Uno
0	8 bits (0 ~ 255)	Comparação <i>Overflow</i>	delay(), millis(), micros() analogWrite() pinos 5 e 6
1	16 bits (0 ~ 65535)	Comparação <i>Overflow</i> <i>Captura de Entrada</i>	Funções para servos analogWrite() pinos 9 e 10
2	8 bits (0 ~ 255)	Comparação <i>Overflow</i>	tone() analogWrite() pinos 3 e 11

Interrupções ATmega328P

Definindo o *prescaler* (ex. timer 1)

O timer 1 não afeta millis();

Processo é similar nos demais timers.

(*) Datasheet do ATmega328P

TCCR1B – Timer/Counter1 Control Register B

Bit (0x81)	7	6	5	4	3	2	1	0	
	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	

Table 15-6. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$\text{clk}_{\text{IO}}/1$ (no prescaling)
0	1	0	$\text{clk}_{\text{IO}}/8$ (from prescaler)
0	1	1	$\text{clk}_{\text{IO}}/64$ (from prescaler)
1	0	0	$\text{clk}_{\text{IO}}/256$ (from prescaler)
1	0	1	$\text{clk}_{\text{IO}}/1024$ (from prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

Interrupções (prática)

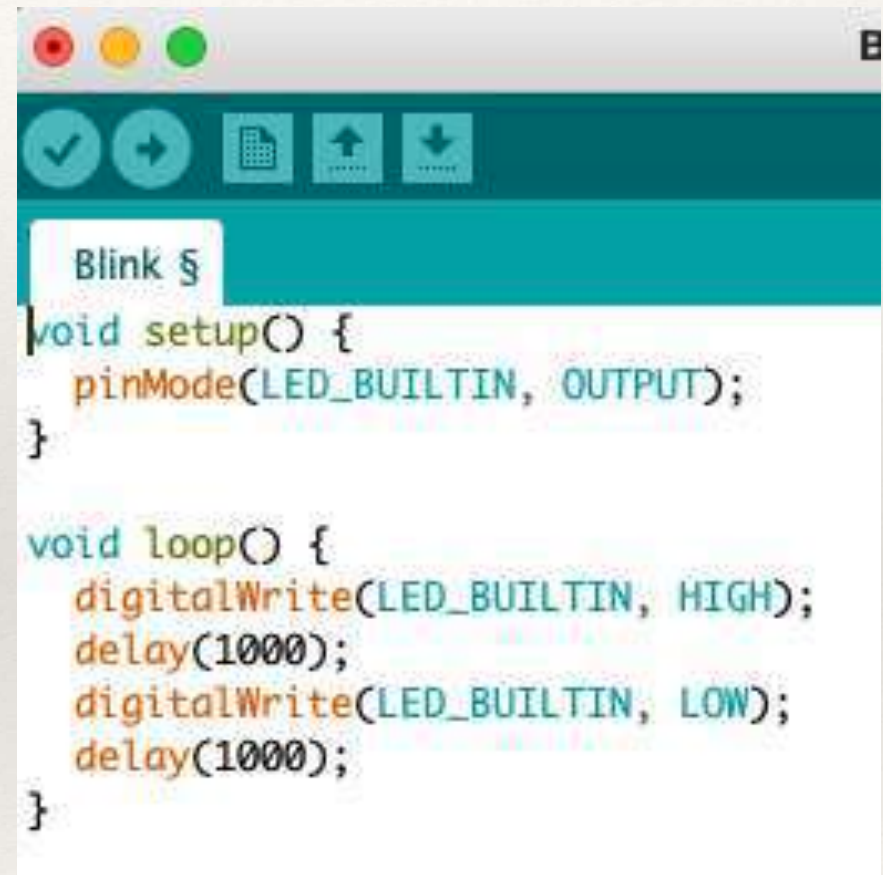
Para testar as interrupções com *timers*, vamos modificar o clássico “Blink”;

O código original pode ser obtido em

[Arquivos→Exemplos→Basics→Blink](#)

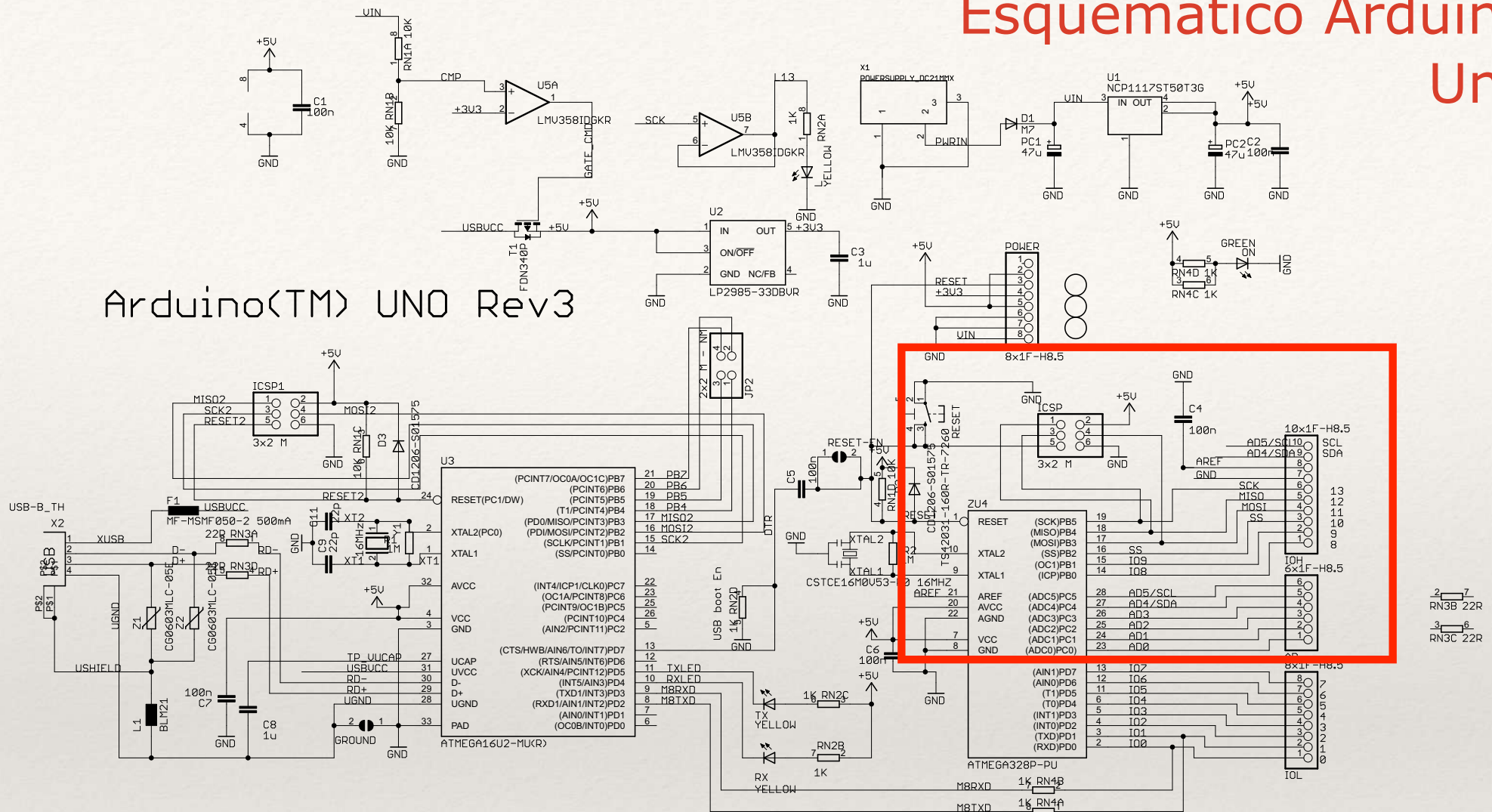
Vamos iniciar o processo testando o código original

LED_BUILTIN = 13

A screenshot of the Arduino IDE interface. The top toolbar shows icons for checking, running, and saving. Below the toolbar, a dropdown menu shows 'Blink' as the selected sketch. The main text area displays the C++ code for the Blink sketch. The code defines a setup function to initialize the LED_BUILTIN pin as an output and a loop function that toggles the LED state with 1000ms delays.

```
void setup() {  
  pinMode(LED_BUILTIN, OUTPUT);  
}  
  
void loop() {  
  digitalWrite(LED_BUILTIN, HIGH);  
  delay(1000);  
  digitalWrite(LED_BUILTIN, LOW);  
  delay(1000);  
}
```

Esquemático Arduino Uno

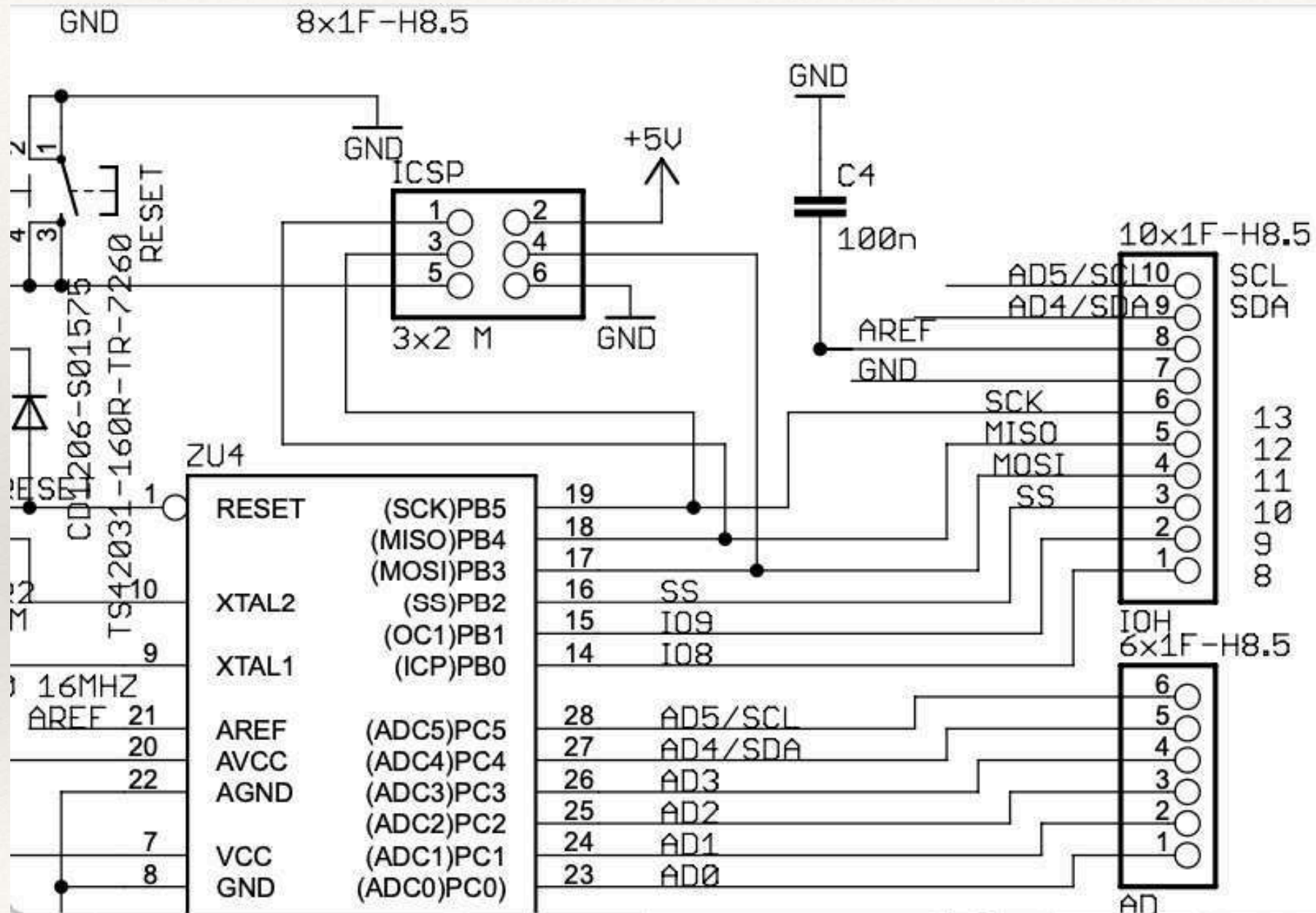


Reference Designs ARE PROVIDED "AS IS" AND "WITH ALL FAULTS. Arduino DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, REGARDING PRODUCTS, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Arduino may make changes to specifications and product descriptions at any time, without notice. The Customer must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Arduino reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The product information on the Web Site or Materials is subject to change without notice. Do not finalize a design with this information.

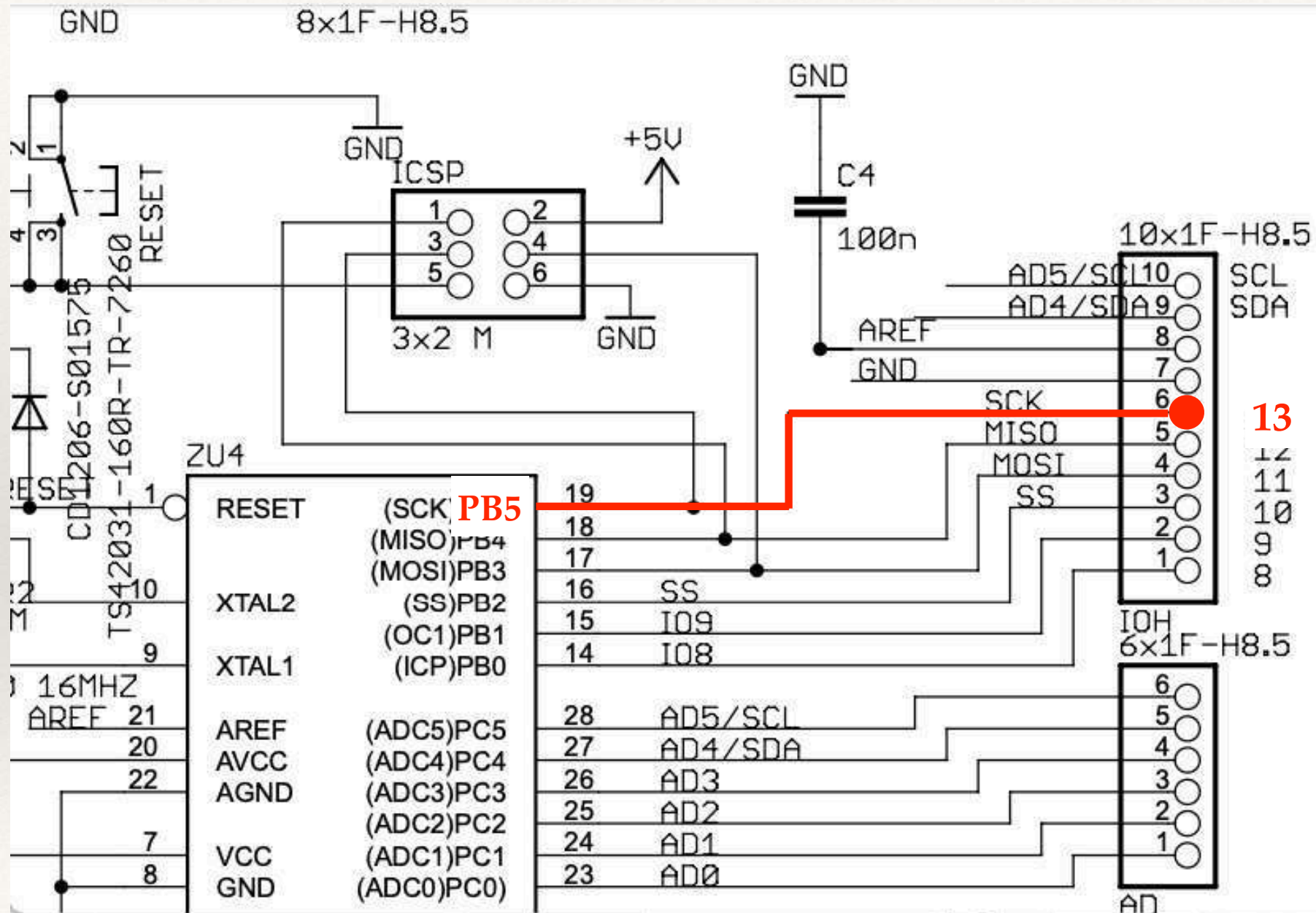
ARDUINO is a registered trademark.

Use of the ARDUINO name must be compliant with <http://www.arduino.cc/en/Main/Policy>

Esquemático Arduino Uno



Esquemático Arduino Uno



Interrupções (prática)

Temos o mesmo código, mas agora com registradores;

Outra opção é apenas inverter o valor do registrador

`PORTB ^= (1 << PB5 );`

A operação OU-Exclusivo (^) inverte o valor do bit.

Função *delay(1000)* mantém o processador “ocupado” por 1s;

Um segundo é muito tempo !

Temos que usar o *timer1* ! (2 Bytes)



```
BlinkComRegistradores §  
void setup() {  
  // Define o pino 13 como OUTPUT  
  DDRB |= (1 << PB5 );  
}  
  
void loop() {  
  
  // Ativa a porta PB5  
  PORTB |= (1 << PB5 );  
  delay(1000);  
  
  // Deativa a porta PB5  
  PORTB &= ~(1 << PB5 );  
  delay(1000);  
}
```

Interrupções (prática)

Temos o mesmo código, mas agora com registradores;

Outra opção é apenas inverter o valor do registrador

`PORTB ^= (1 << PB5 );`

A operação OU-Exclusivo (^) inverte o valor do bit.

Função *delay(1000)* mantém o processador “ocupado” por 1s;

Um segundo é muito tempo !

Temos que usar o *timer1* ! (2 Bytes)



```
BlinkComRegistradores
void setup() {
  // Define o pino 13 como OUTPUT
  DDRB |= (1 << PB5 );
}

void loop() {

  // Ativa a porta PB5
  PORTB ^= (1 << PB5 );
  delay(1000);
}
```


Preparando o *timer1*

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	

O primeiro registrador importante é o TCCR1A.

Utilizado principalmente para o PWM (que veremos depois);

Valores gravados para uso do `analogWrite()` podem gerar problemas;

Basta resetar todos os bits: **TCCR1A = 0;**

Preparando o *timer1*

TCCR1B – Timer/Counter1 Control Register B

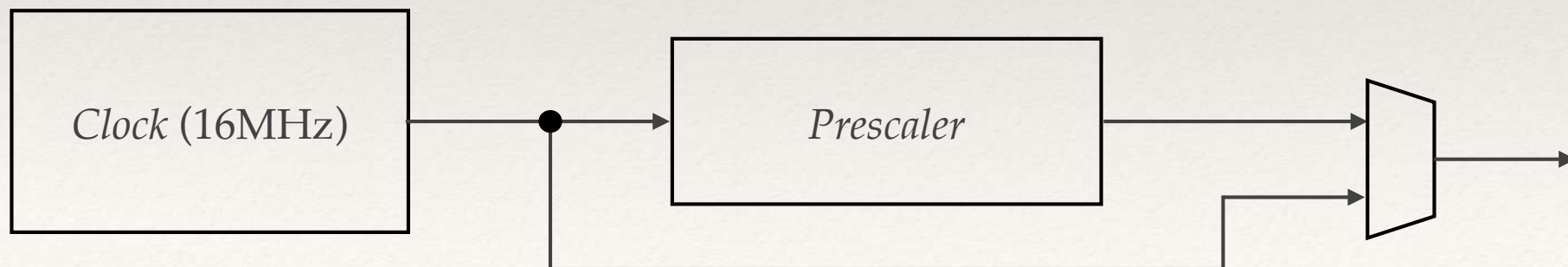
Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

O TCCR1B define, entre outras coisas, o prescaler

No momento, os bits 3 ~ 7 podem ser deixados de lado (outras funções);

Os bits 0 ~ 2 definem o tratamento do clock para o *timer1*;

Sem *clock* ? *Clock* externo? *Prescaler* com qual valor?



Um ciclo a cada $1/16 \text{ MHz} = 62,5 \text{ nS}$

Preparando o *timer1*

Uma mudança do LED a cada segundo

Clock de 16MHz = 16 milhões de ciclos por segundo;

Contagem máxima do *timer1* é 65.536 (16 bits);

Sem divisão, LED piscaria pelo menos 244 vezes por segundo ($16.000.000 / 65.536$)

$16.000.000 / 64 = 250.000 \Rightarrow$ inválido ($>$ contagem máxima)

$16.000.000 / 256 = 62.500 \Rightarrow$ válido (escolhido) !

$16.000.000 / 1.024 = 15.625 \Rightarrow$ válido !

Prescaler = 256; Contador = 62.500;

Table 15-6. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$\text{clk}_{\text{IO}}/1$ (no prescaling)
0	1	0	$\text{clk}_{\text{IO}}/8$ (from prescaler)
0	1	1	$\text{clk}_{\text{IO}}/64$ (from prescaler)
1	0	0	$\text{clk}_{\text{IO}}/256$ (from prescaler)
1	0	1	$\text{clk}_{\text{IO}}/1024$ (from prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

Preparando o *timer1*

Definindo o prescaler em 256

```
TCCR1B |= (1 << CS12 );
```

```
TCCR1B &= ~(1 << CS11 );
```

```
TCCR1B &= ~(1 << CS10 );
```

Para usar o contador

Definimos a interrupção pela comparação com o registro A;

```
TIMSK1 = (1 << OCIE1A );
```

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	

Input Capture

Overflow

Preparando o *timer1*

Carga do registro A

Definimos constantes para facilitar o acesso aos registradores de 16 bits:

```
const uint16_t t1_carga = 0;  
const uint16_t t1_comparador = 62500;
```

Registradores são carregados dentro da ISR:

```
ISR( TIMER1_COMPA_vect ) {  
    TCNT1 = t1_carga;  
    PORTB ^= ( 1 << PB5 );  
}
```

Também é importante habilitar as interrupções

```
sei();
```

Código Final

Estrutura loop() só possui delay()

Nenhum código afetará a operação do blink();

ISR() pode ser ainda menor

Modo CTC pode ser habilitado;

timer1 zera ao alcançar registro A;

Objetivo é fazer ISR() o menor possível.

(*) Código inspirado no material da SparkFun “Level Up Your Arduino Code: Timer Interrupts”

```
BlinkComInterrERegistr
const uint16_t t1_carga = 0;
const uint16_t t1_comparador = 62500;

void setup() {

    // Define o pino 13 como OUTPUT
    DDRB |= (1 << PB5 );

    // Impõe RESET ao registro A de controle do timer1
    TCCR1A = 0;

    // Define o prescaler em 256
    TCCR1B |= (1 << CS12 );
    TCCR1B &= ~(1 << CS11 );
    TCCR1B &= ~(1 << CS10 );

    // Zera o timer1, e define o valor para comparação
    TCNT1 = t1_carga;
    OCR1A = t1_comparador;

    // Define a interrupção por comparação com o registro A
    TIMSK1 = (1 << OCIE1A );

    sei();

}

void loop() {

    delay(1000);

}

ISR( TIMER1_COMPA_vect ) {

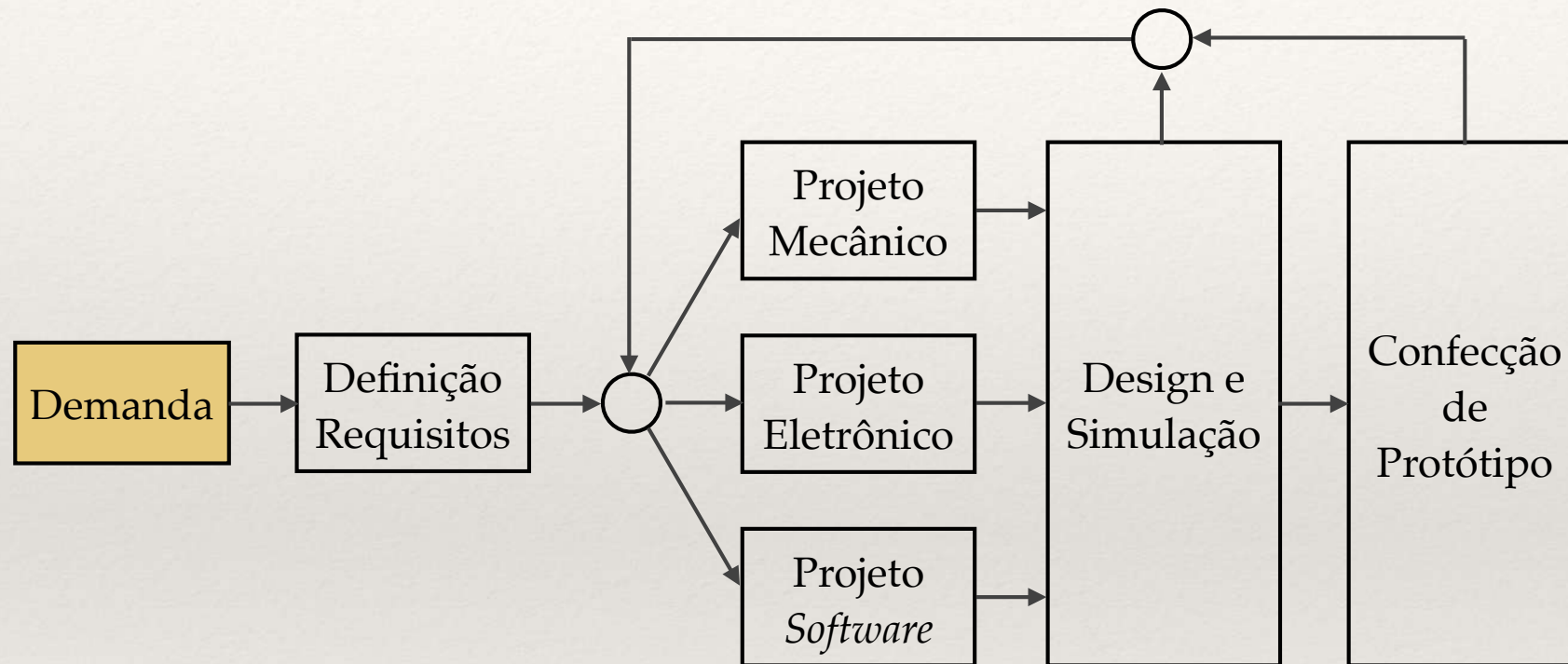
    TCNT1 = t1_carga;

    // Ativa a porta PB5
    PORTB ^= (1 << PB5 );

}
```


Aula 07

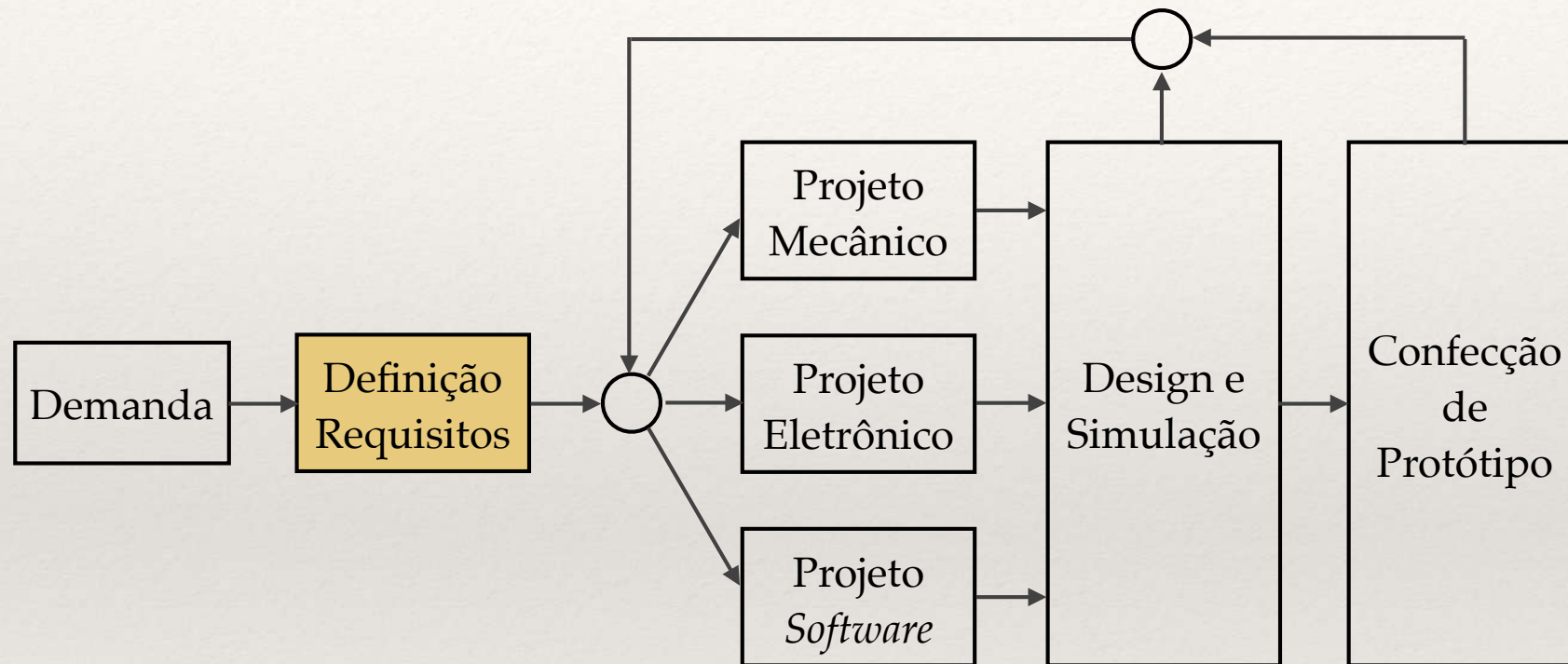
Fases do Projeto de SE



A demanda por SE hoje está em todas as áreas

Esta ficando difícil encontrar um equipamento meramente mecânico ou eletrônico;
Sistemas embarcados melhoram as funcionalidades, aumentam a vida útil, reduzem custo;
Caminho similar ao da informatização, da internet, e agora da IA.

Fases do Projeto de SE



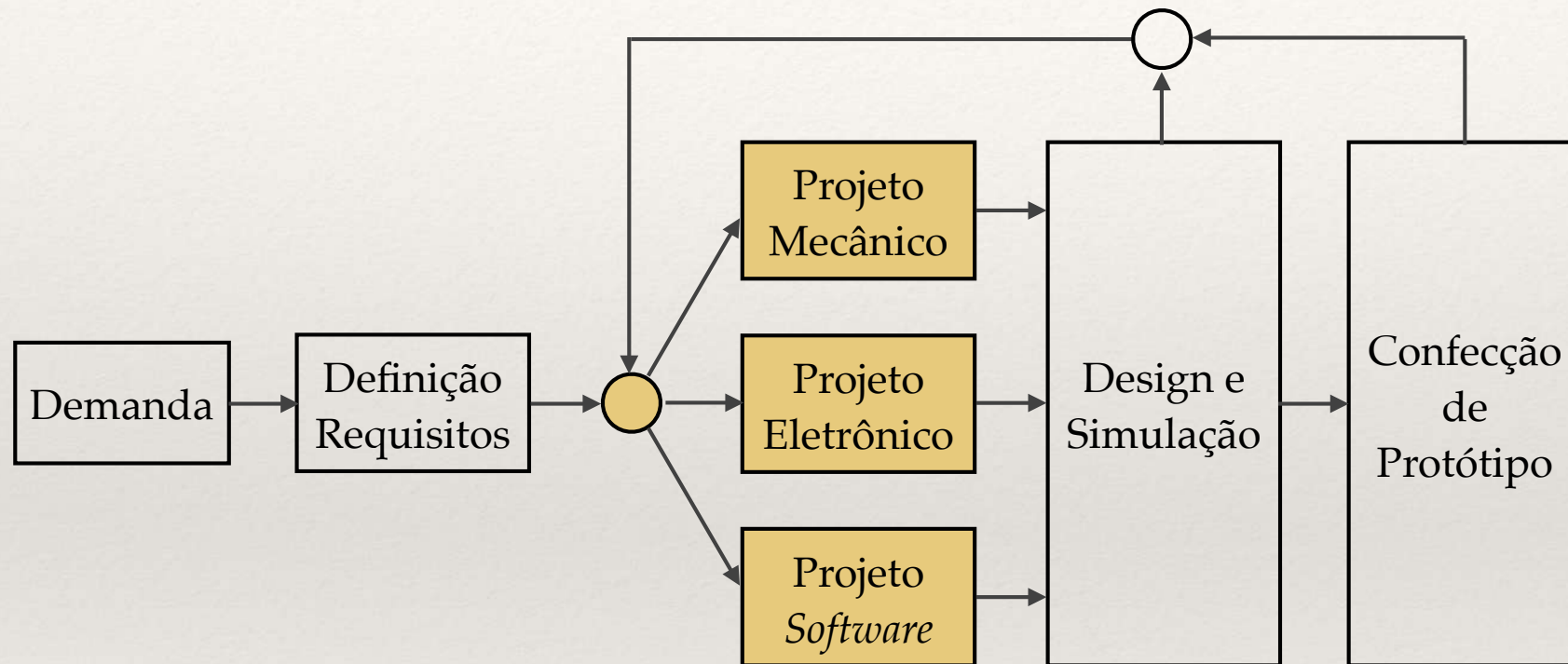
Definição de requisitos

Envolve diversas disciplinas;

Exige um trabalho bem cuidadoso, para garantir longevidade ao equipamento;

Pode mudar inclusive durante o ciclo de vida do produto (atualização de *software*).

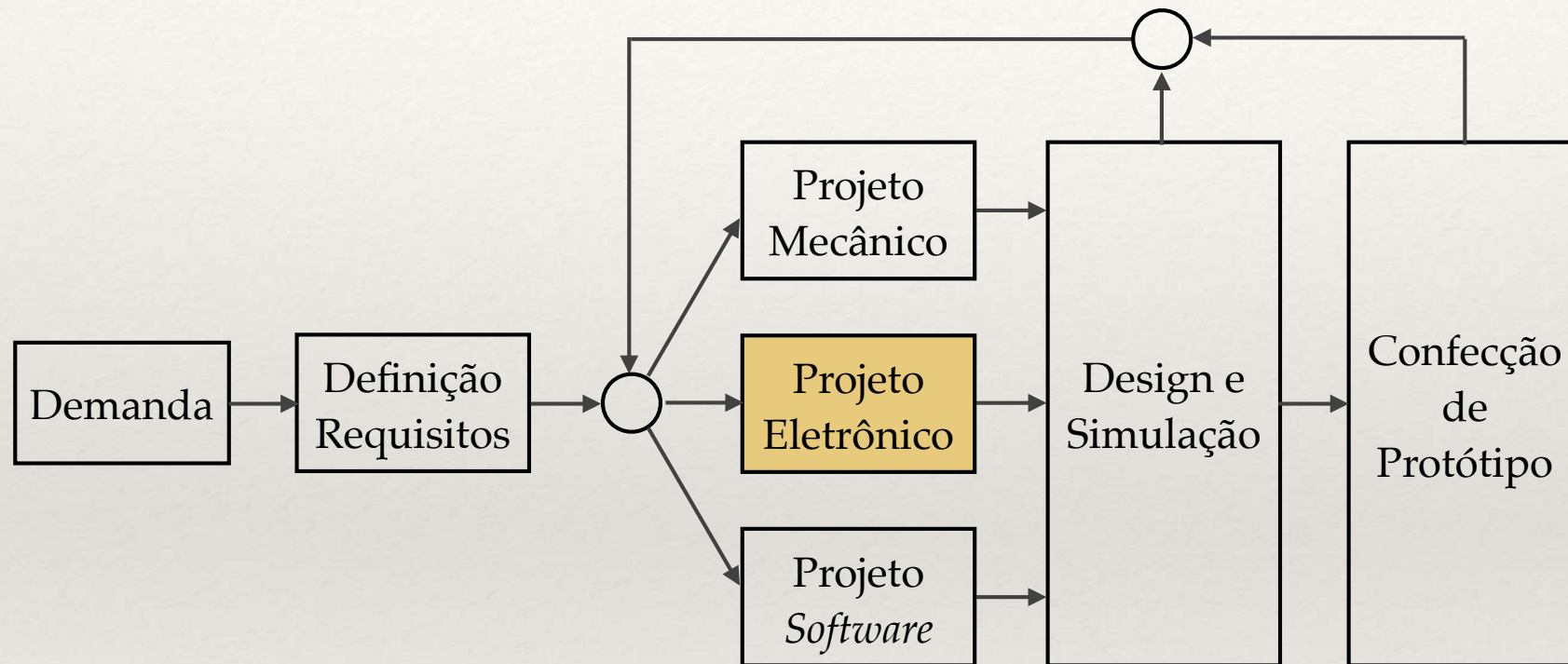
Fases do Projeto de SE



Projetos multidisciplinares

Podem inclusive envolver outras áreas (química, biofísica, agronomia, medicina etc);
Desenvolvidos de forma virtual com auxílio de ferramentas CAx (CAD, CAM, CAE etc);
Há forte interação entre projetos (espaço ocupado, peso, consumo de energia etc).

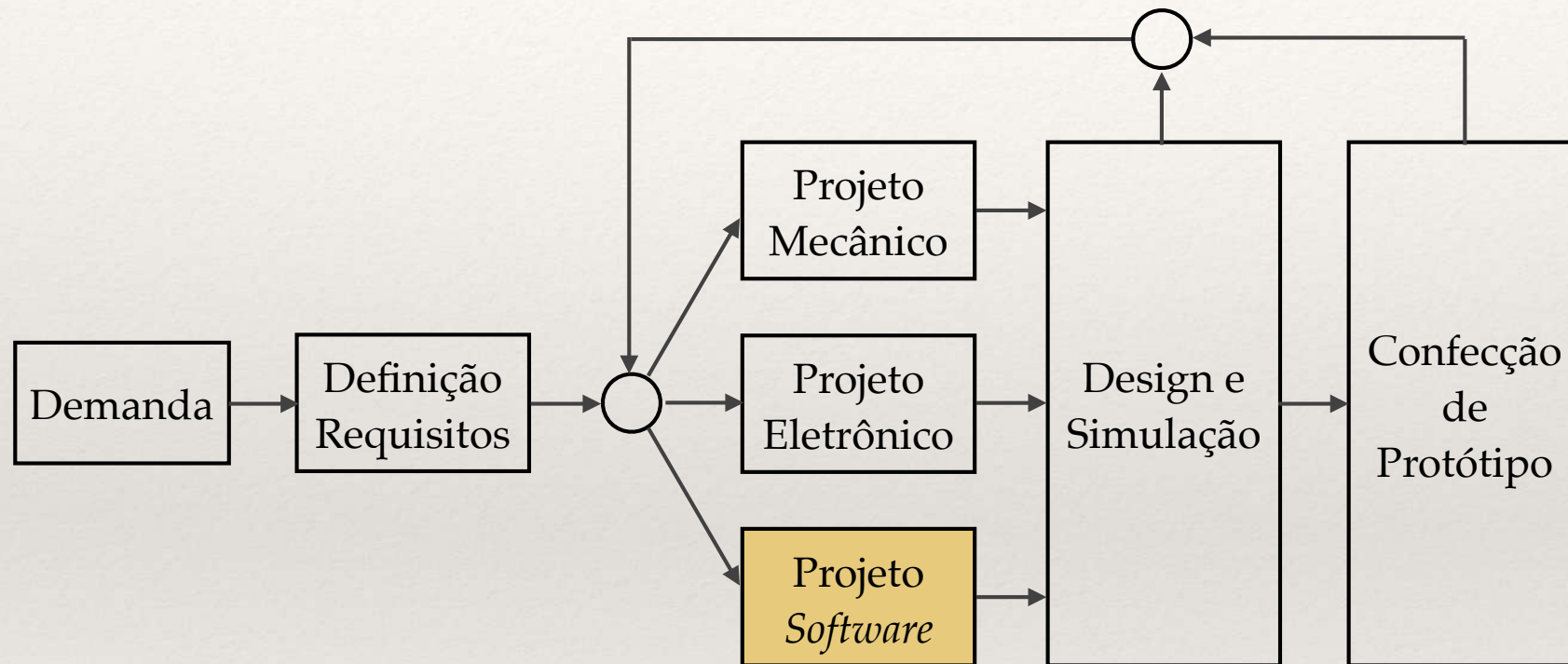
Fases do Projeto de SE



Projeto Eletrônico

Também é multidisciplinar (elétrica, eletrônica, microcontroladores, sensores, controle etc); Envolve definição da plataforma (microcontrolador?), sensores, atuadores e componentes; Integração forte com o projeto de *software*.

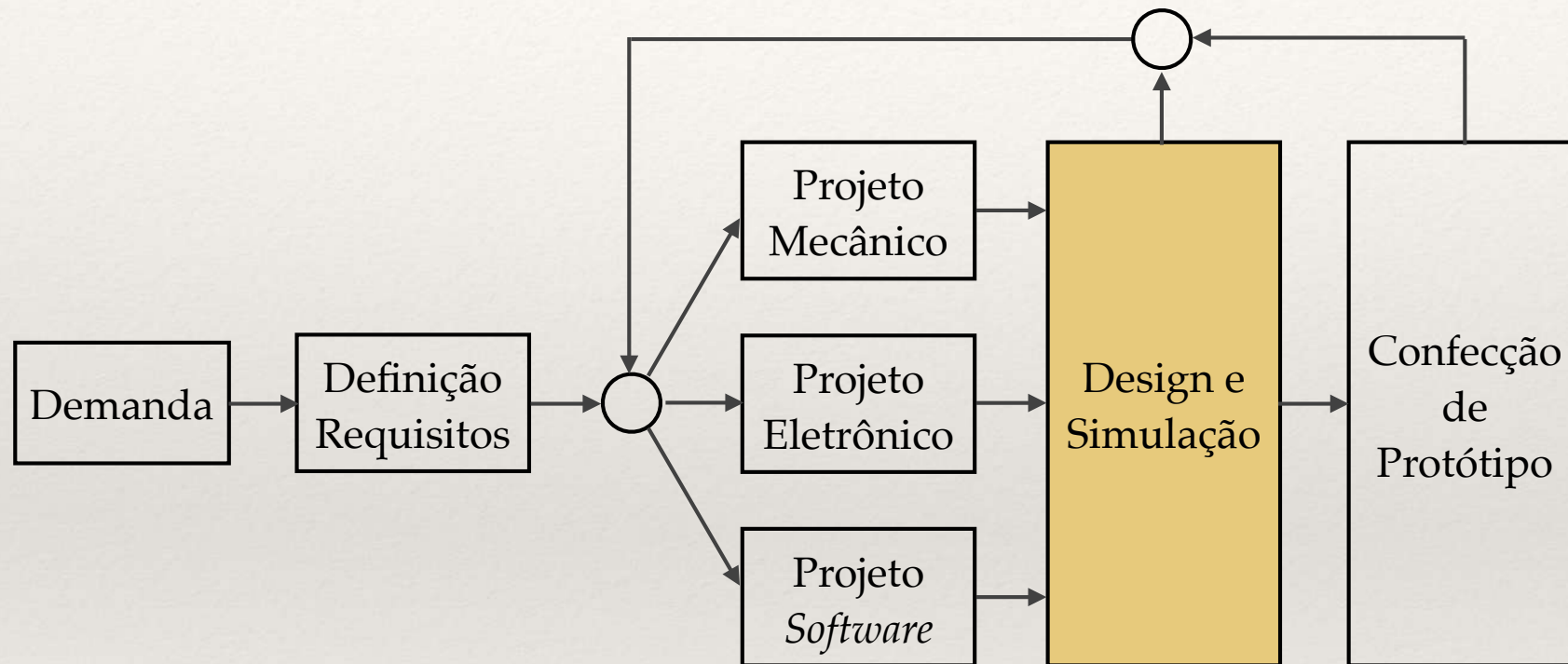
Fases do Projeto de SE



Projeto *Software*

Fortemente afetado pelos requisitos do projeto, e escolhas no projeto eletrônico;
Exige definições e sofre efeitos bem diferentes dos demais projetos;
É o nosso objeto de estudo nesta aula.

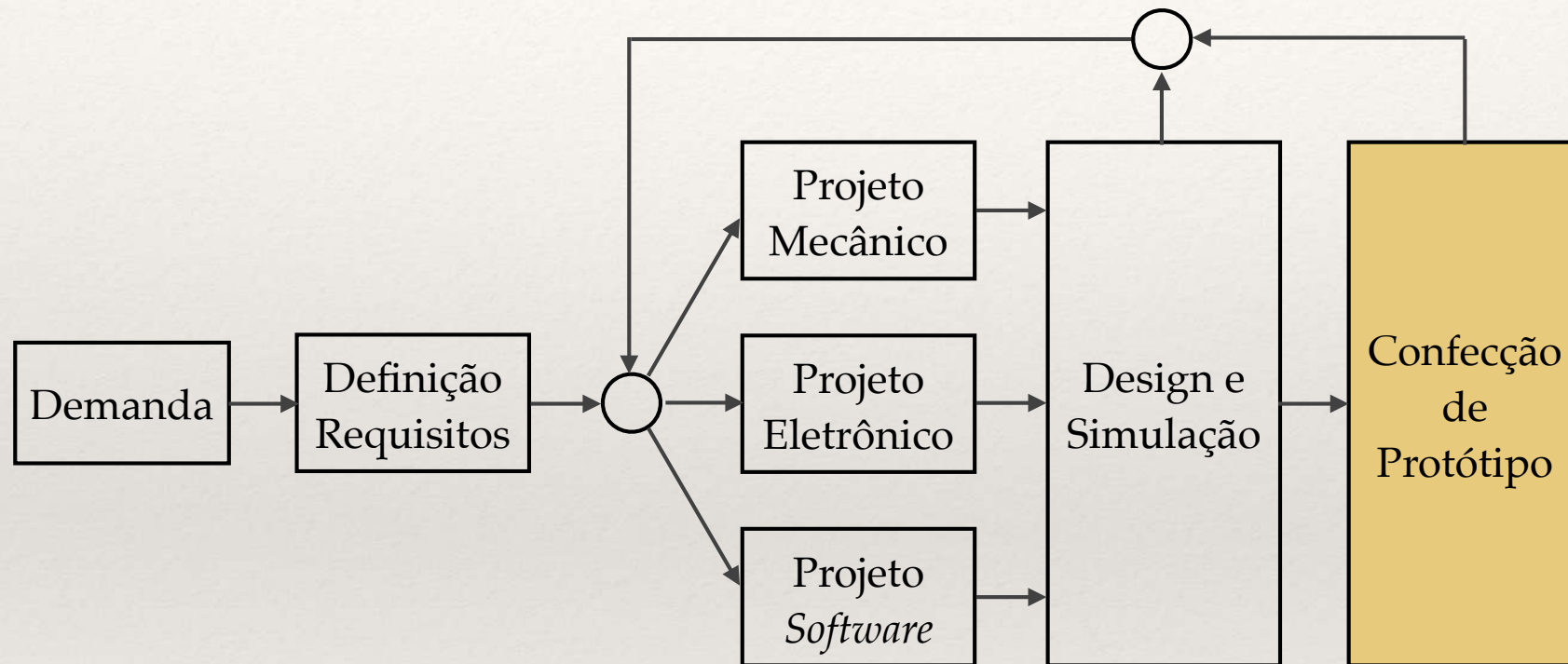
Fases do Projeto de SE



Design e Simulação

Há riqueza de ferramentas de *software* para apoio nesta etapa de projeto;
Tipicamente é possível simular a maior parte dos componentes do projeto;
Comportamentos observados na simulação podem afetar o projeto, exigindo ajustes.

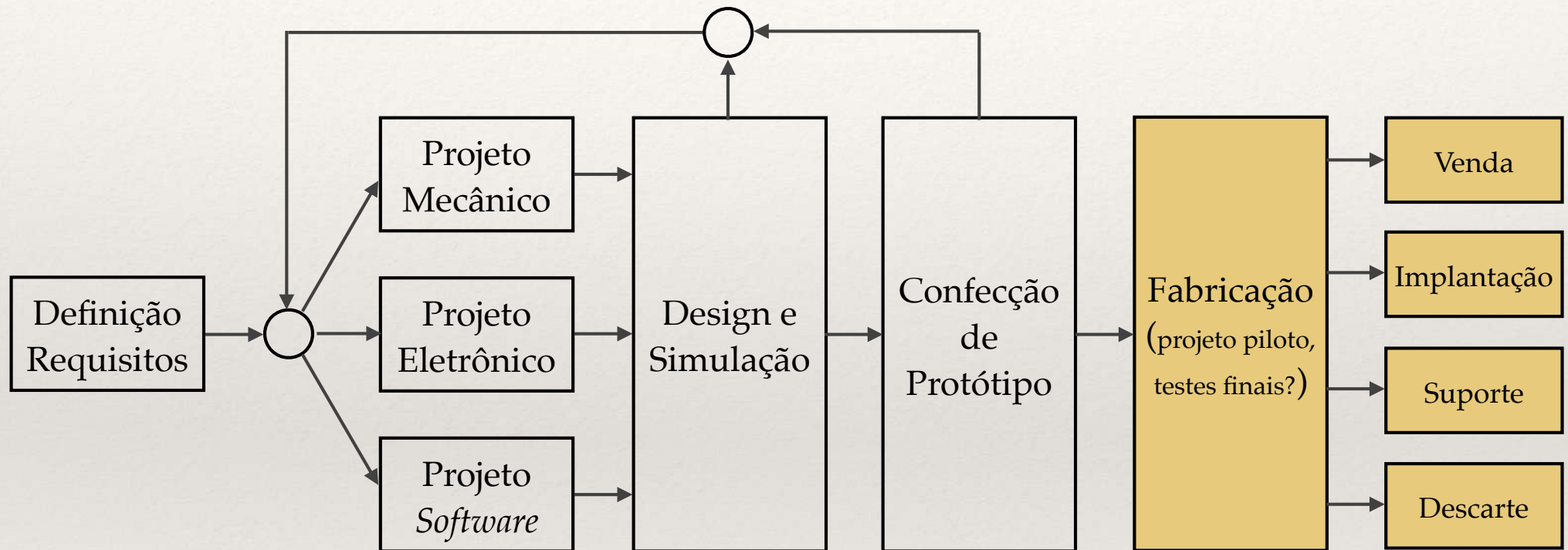
Fases do Projeto de SE



Confecção de Protótipo

Tipicamente, a confecção de um modelo físico e operacional do projeto é essencial; O comportamento do modelo ainda pode afetar o projeto, exigindo modificações.

Fases do Projeto de SE



Etapas posteriores

Obviamente o trabalho não acaba no protótipo;
Um projeto exige muitas outras preocupações, que podem inclusive afetar as primeiras etapas

Documentação

Treinamento

Seleção Fornecedores

Software em SE

Quem executará o *software*?

Definição tipicamente ocorre durante o projeto de *hardware*;
Soluções proprietárias (PLCs);



Software em SE

Quem executará o *software*?

Definição tipicamente ocorre durante o projeto de *hardware*;
Os dez maiores fabricantes:

	Fabricantes	Marca
1	Siemens	Simatic
2	Rockwell Automation	Allen Bradley
3	Mitsubishi Electric	Melsec
4	Schneider Electric	Modicon
5	Omron	Sysmac
6	Emerson Electric (GE)	RX3i & VersaMax (GE Fanuc)
7	Keyence	KV & V-8000
8	ABB (B&R Automation)	AC500 X20 & X90
9	Bosch	Rexroth ICL
10	Hitachi	EH & H

Software em SE

Norma IEC 61131-3 (1993 ~ 2013)

Promover interoperabilidade entre diferentes fabricantes de PLCs;
Define, entre outras coisas, 5 linguagens de programação padrão

LD (*Ladder Diagram*) - gráfica;

FBD (*Function Block Diagram*) - gráfica;

ST (*Structured Text*) - textual;

IL (*Instruction List*) - textual (descontinuada);

SFC (*Sequential Function Chart*) - gráfica.

Tecnologia, no entanto, continua fechada

Não há acesso ao *software*, nem à arquitetura de *hardware*;

Restrições levaram à busca por plataformas abertas;

Motivação para estudo dos microcontroladores em sistemas de automação, ou seja, nossa disciplina !

Software em SE

Microcontroladores utilizados em PLCs

Em sistemas menores, linha PIC e ARM Cortes M3 ou M4, ou FPGAs;

Em sistemas de grande porte, ARM Cortes A8 e A9, e microprocessadores.

Tecnologia aberta

Promove a redução significativa de custos;

Garante flexibilidade, por não depender de soluções proprietárias.

OpenPLC - uma iniciativa de tecnologia aberta

Iniciativa do projeto de doutorado de Thiago Rodrigues Alves

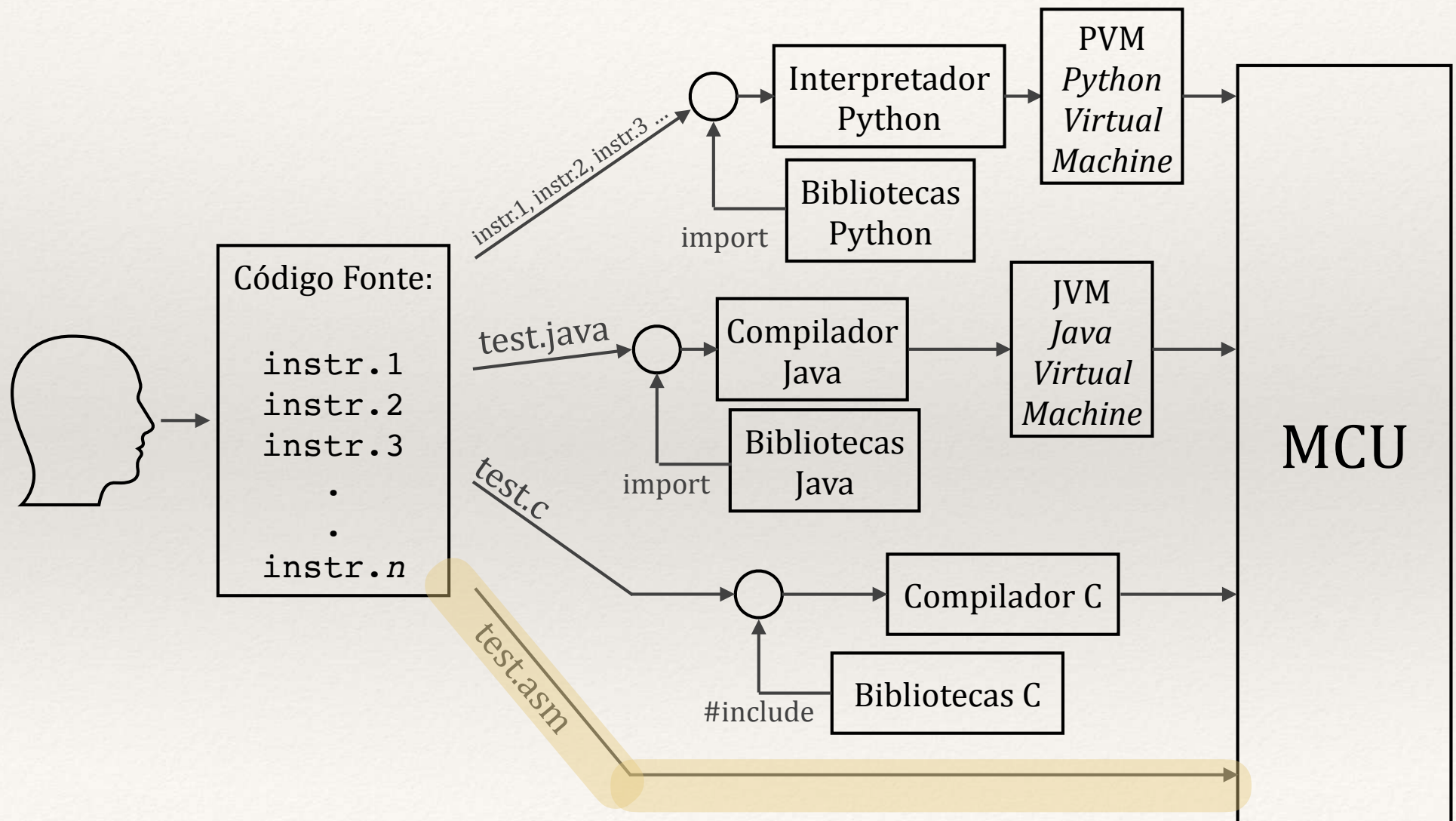
Suporta as 5 linguagem padrão definidas pela IEC 61131-3

Pode ser executada nas plataformas PIC, Arduíno, ESP, Raspberry Pi etc.

<https://autonomylogic.com/>

<https://www.youtube.com/watch?v=cNg-tXZynJE>

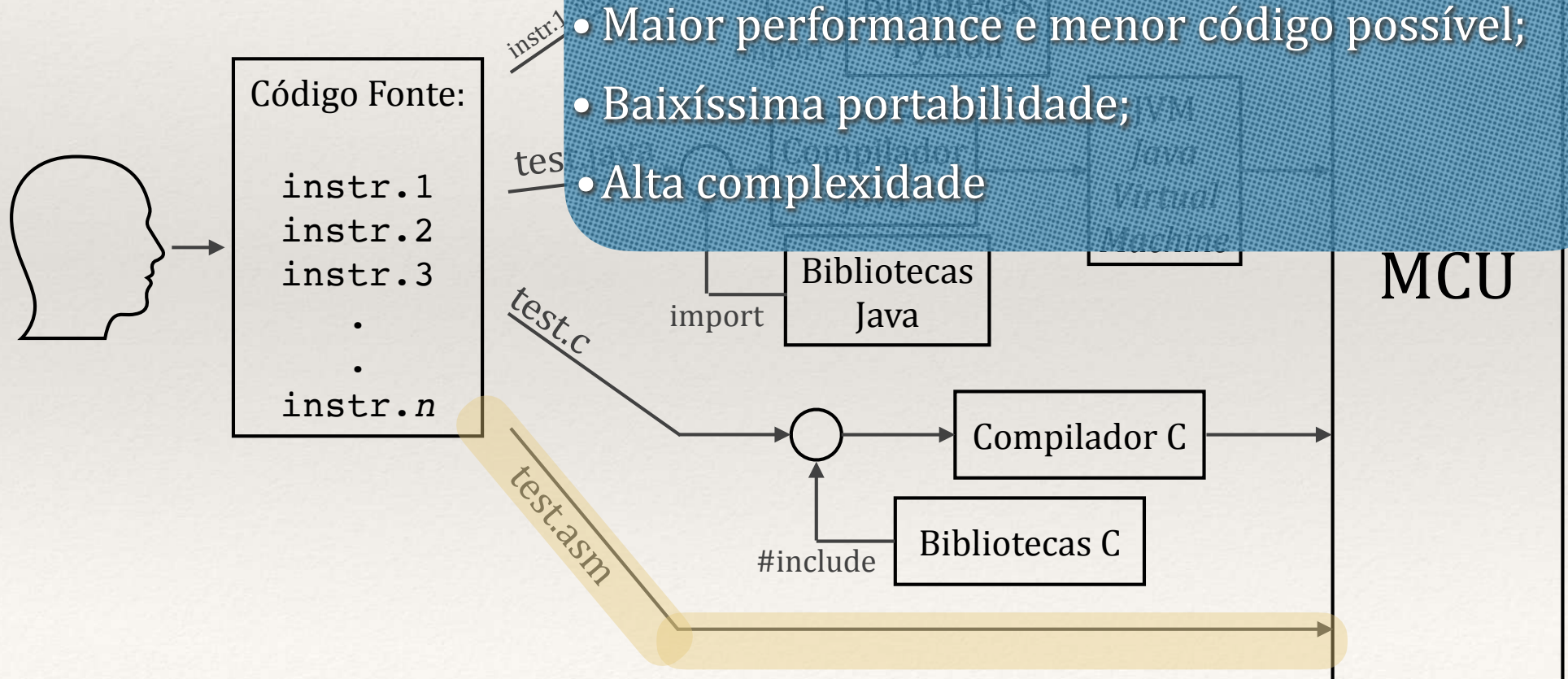
Programa "aberto" MCUs



Programa "aberto" MCUs

Programação *Assembly*

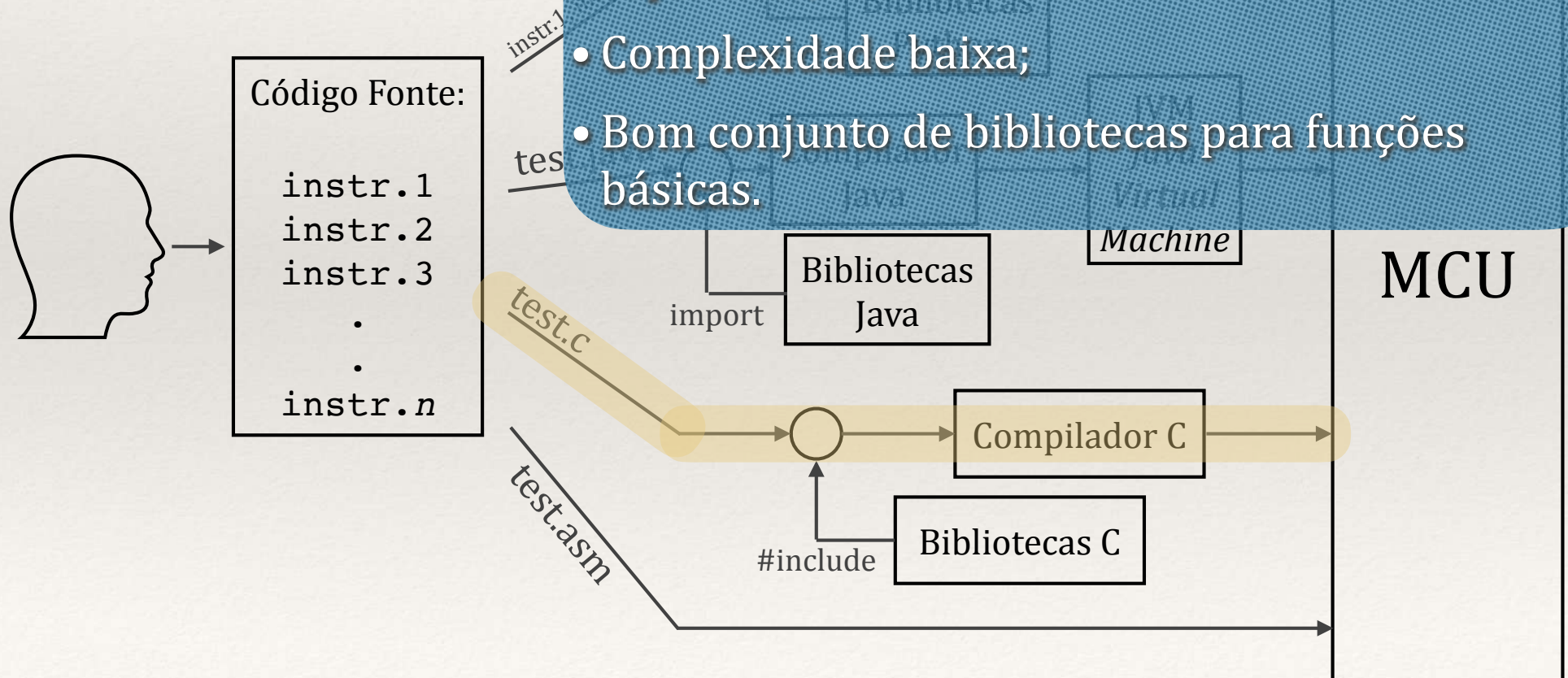
- Não há necessidade de compilação, exceto para conversão dos mnemônicos em instruções de linguagem de máquina com o "Assembler";
- Maior performance e menor código possível;
- Baixíssima portabilidade;
- Alta complexidade



Programa "aberto" MCUs

Programação C, C++

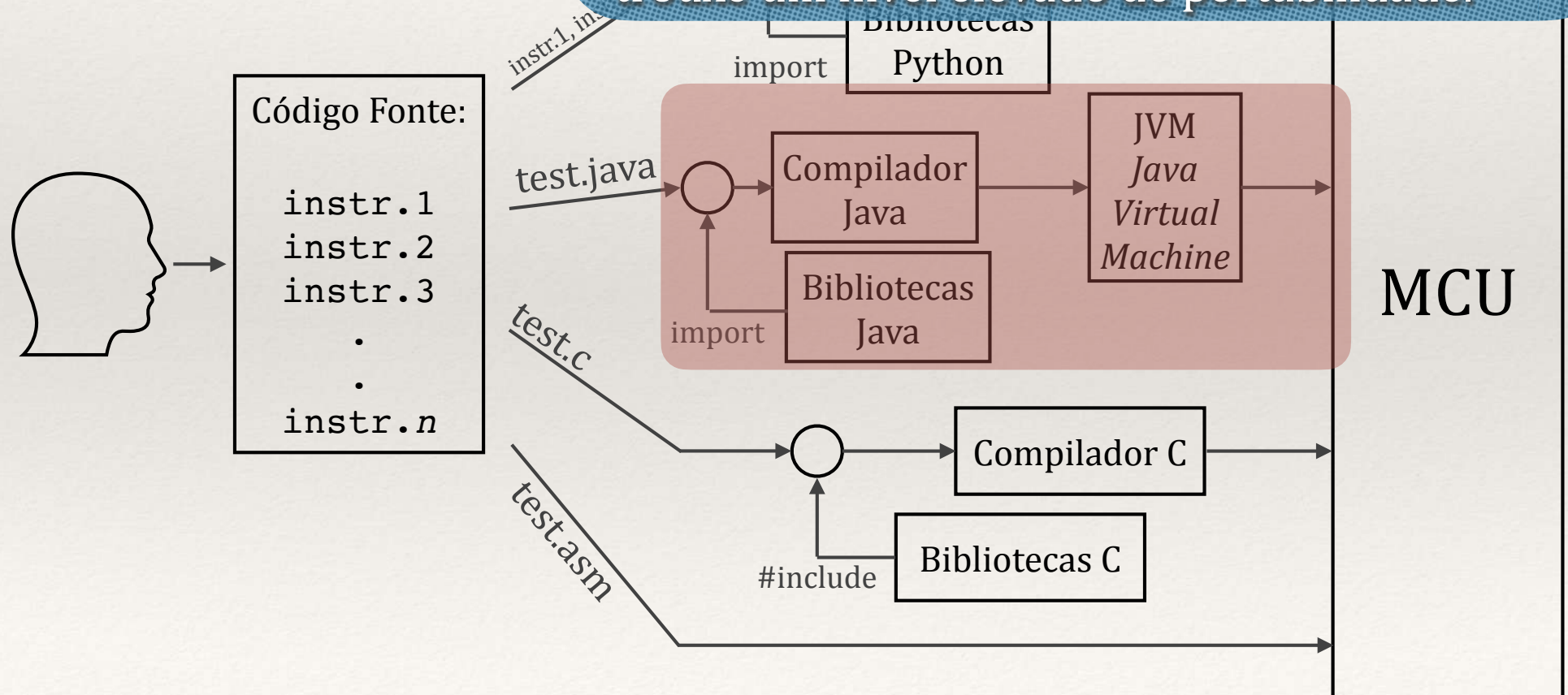
- Boa performance e código reduzido;
- Boa portabilidade, embora aceite código específico;
- Complexidade baixa;
- Bom conjunto de bibliotecas para funções básicas.



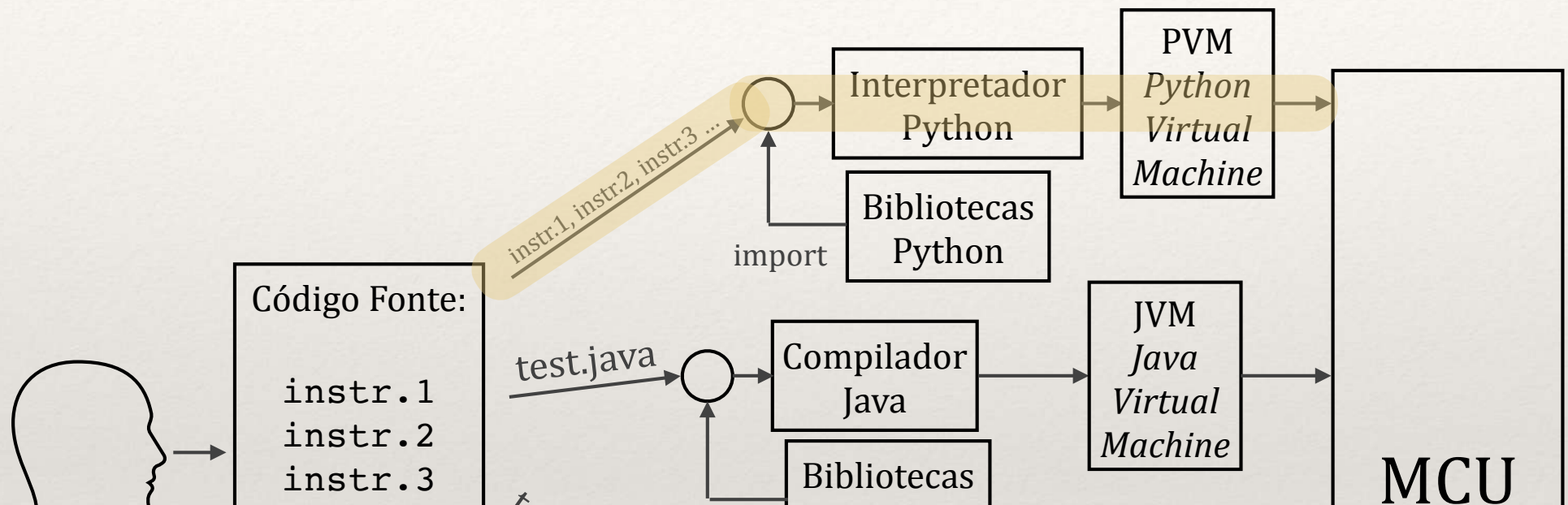
Programa "aberto" MCUs

Programação Java

- Incomum na programação de SE;
- Incorpora o conceito de Máquina Virtual, que trouxe um nível elevado de portabilidade.



Programa "aberto" MCUs



Programação Python

- Tipicamente é interpretada, e não compilada. Opera inclusive em modo interativo;
- Possui um conjunto robusto de bibliotecas com funcionalidades de IA, aprendizado de máquina, reconhecimento de imagens etc.

Compiladores especiais

Cross Compilers

Criam códigos executáveis para dispositivos diferentes de onde o compilador foi executado;

Tipicamente utilizado na programação de MCUs e outros dispositivos que não suportam adequadamente a operação de um compilador;

Também são utilizados na programação de celulares, *smartwatches* etc.

Source-to-source Compilers

Transforma um código fonte escrito em uma linguagem de programação para outra linguagem;

Existem exemplos para converter LD (*Ladder Diagram*) para C++, por exemplo.

Software em SE

Parâmetros importantes a serem avaliados

Performance: quanto mais baixo o nível, mais rápido;

Ocupação de Memória: quanto mais baixo o nível, menor ocupação;

Grau de abstração: quanto mais alto o nível, maior a abstração;

Portabilidade: linguagens de alto nível e Máquinas Virtuais ajudam;

Reuso & Bibliotecas: maior a participação de mercado ajuda;

Documentação e recursos de apoio: idem acima.

Cabe ao projetista a análise

Custo e tempo de desenvolvimento/testes;

Direitos autorais.

Outros componentes de SW

Software Básico

Bootloader

Nos computadores, carrega o SO da unidade de disco;

Software de inicialização, tipicamente associado a uma placa de protótipo;

Configuração Básica de módulos internos (Wi-Fi, UART etc)

Permite a carga de sketches;

Sistema Operacional

Tipicamente ausente nos MCUs, pode ser encontrado nos SBC (*Single Board Computers*);

Incorpora funcionalidades importantes, porém consome recursos para sua execução.

RTOS (*Real Time Operational System*)

Para microcontroladores, especialmente em aplicações multitarefa e em tempo real;

Trata problemas de compartilhamento de recursos, sincronismo e priorização.

Integração com recursos na nuvem

Diversos recursos são oferecidos na nuvem, especialmente para IoT.

Aula 08

IoT e as Redes

É inegável que a tecnologia IoT depende da qualidade das redes de comunicação, embora existam outros aspectos também importantes, como os Sistemas Operacionais e aplicações suportadas;

A tecnologia IoT adota múltiplos protocolos, alguns já existentes antes do seu surgimento, como o Wi-Fi, e outros bem ajustados às suas características:

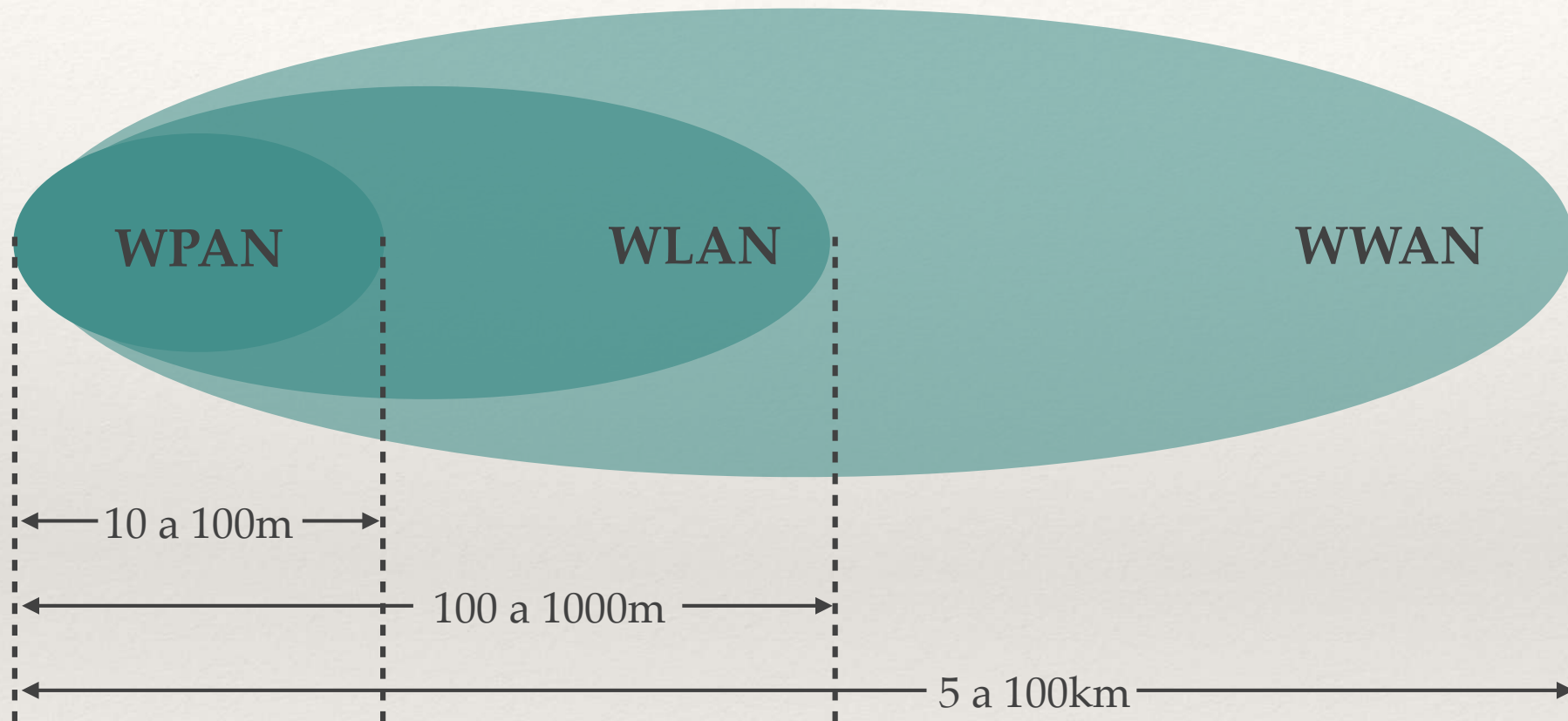
Baixa demanda de largura de banda;

Baixo consumo energético;

Acesso típico sem fio.

Tecnologias como Bluetooth, RFID, ZigBee, NFC e LoWPAN estão entre elas.

IoT e as Redes (alcance)



A IoT normalmente está associada a redes com baixo alcance. No entanto, alguns protocolos mais recentes têm expandido seu alcance, como a LoRaWAN, por exemplo;

Novas aplicações como telemetria, medição de consumo residencial de água, luz e gás, gestão de trânsito e agropecuária podem ser possíveis.

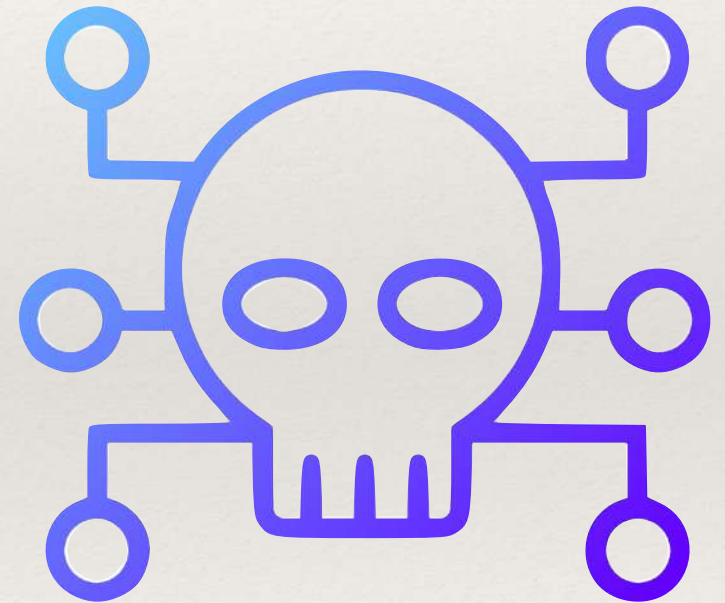
IoT X segurança

O conceito da IoT é conectar “tudo”, e quanto mais acessos à rede existirem, maiores são os riscos de segurança;

Métodos criptográficos seguros exigem processamento rápido e vigilância contínua, o que atenta contra as características dos dispositivos IoT !

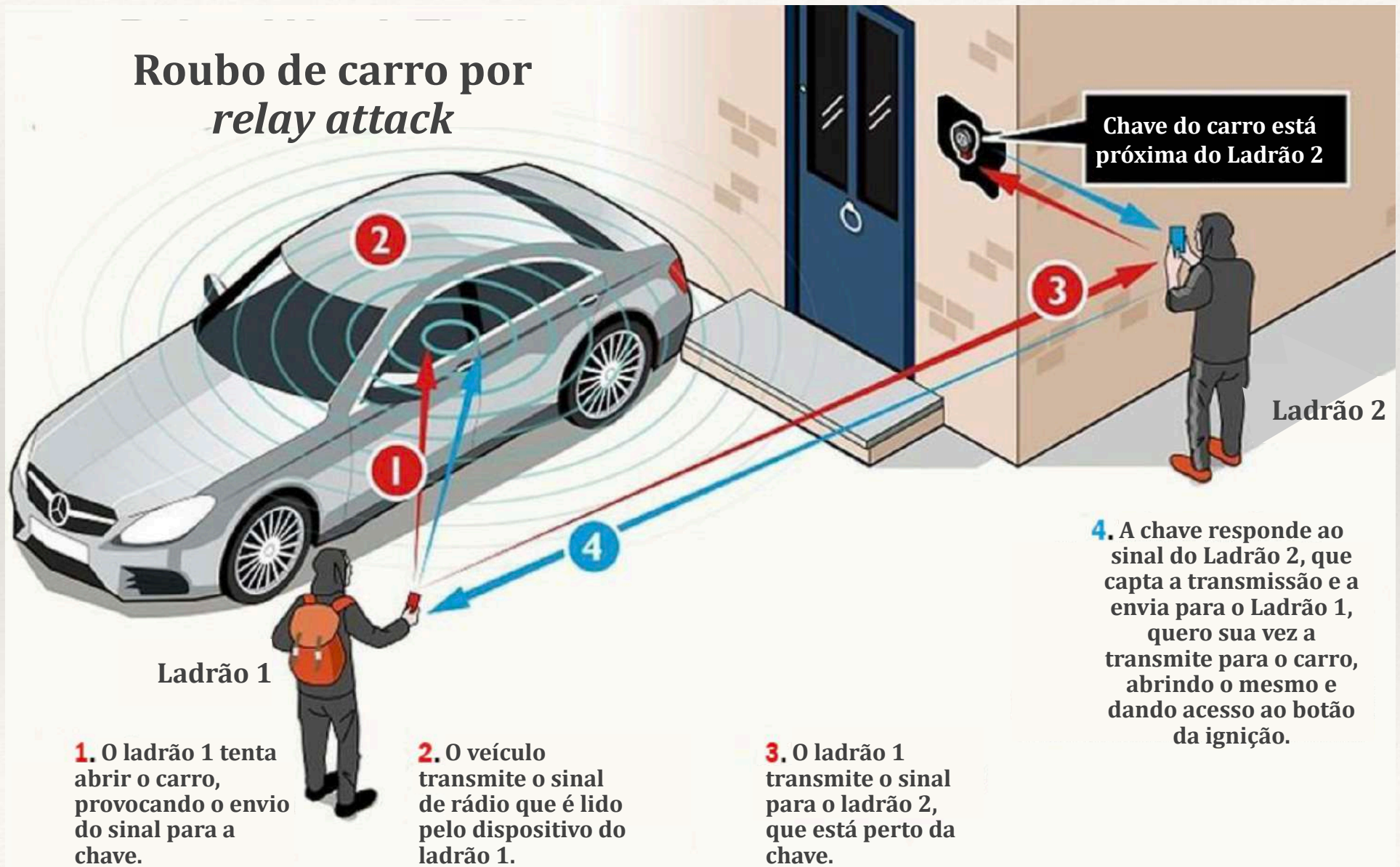
- Capacidade limitada de processamento;
- Baixo consumo energético.

São inúmeros os casos de crimes que exploram as vulnerabilidades dos sistemas IoT.



IoT X segurança

Roubo de carro por *relay attack*



Furto de carros “sem chave” está se tornando comum no Brasil

Por [Felipe Demartini](#) | Editado por [Claudio Yuge](#) 08 de Abril de 2022 às 19h49



Imagem: Felipe Ribeiro/Canaltech

Uma brecha de segurança em modelos de veículos que utilizam a [tecnologia](#) keyless para permitir a abertura de portas e partida do motor sem a necessidade de uso da chave,

1. O ladrão 1 tenta abrir o carro, provocando o envio do sinal para a chave.



2. O veículo transmite o sinal de rádio que é lido pelo dispositivo do ladrão 1.

3. O ladrão 1 transmite o sinal para o ladrão 2, que está perto da chave.

quando o ladrão 2 acessa o botão da ignição.

Reserve Agora

Voos para África

by flytap.com

Skip Ad ►

o está
drão 2

Ladrão 2

onde ao
rão 2, que
missão e a
Ladrão 1,
vez a
a o carro,
mesmo e

IoT X segurança



*Stalking com
o AirTag*

Atualização do Apple AirTag traz recurso 'anti-stalking' ao dispositivo de rastreo

22 de dezembro de 2022 4

ma

COM



O **Apple Airtag** recebeu uma nova **atualização de firmware** com a chegada de um recurso anti-stalking. Ou seja, uma característica que pode ajudar as pessoas a não serem perseguidas ou vigiadas. Esse update é o **2.0.24 que chegou em novembro**, mas só em dezembro é que finalmente atingiu 100% dos aparelhos da maçã.

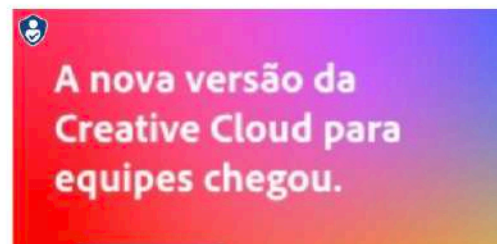


Imagem: Ge



Notícias

> Gadgets



Android ganha proteção contra stalkers que usam o Apple AirTag

Google libera ferramenta nativa que avisa sobre rastreadores por perto. Usuário também encontra dicas de como desativar o equipamento oculto.



Por [Thássius Veloso](#)
10/08/2023 às 12:53

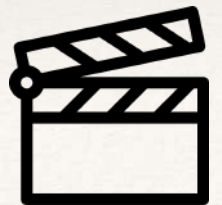


IoT X segurança

Outras implementações também foram alvos de ataque, embora não sejam tão bem documentadas na imprensa. Quais seriam os próximos alvos?

Sistemas Médicos (assassinato)?

Sistemas Aviônicos (terrorismo)?



Comunicação em SE

Em Sistemas Embarcados,
microcontroladores e sensores
precisam se comunicar

Sinais elétricos (portas analog./digitais);

Interfaces cabeadas;

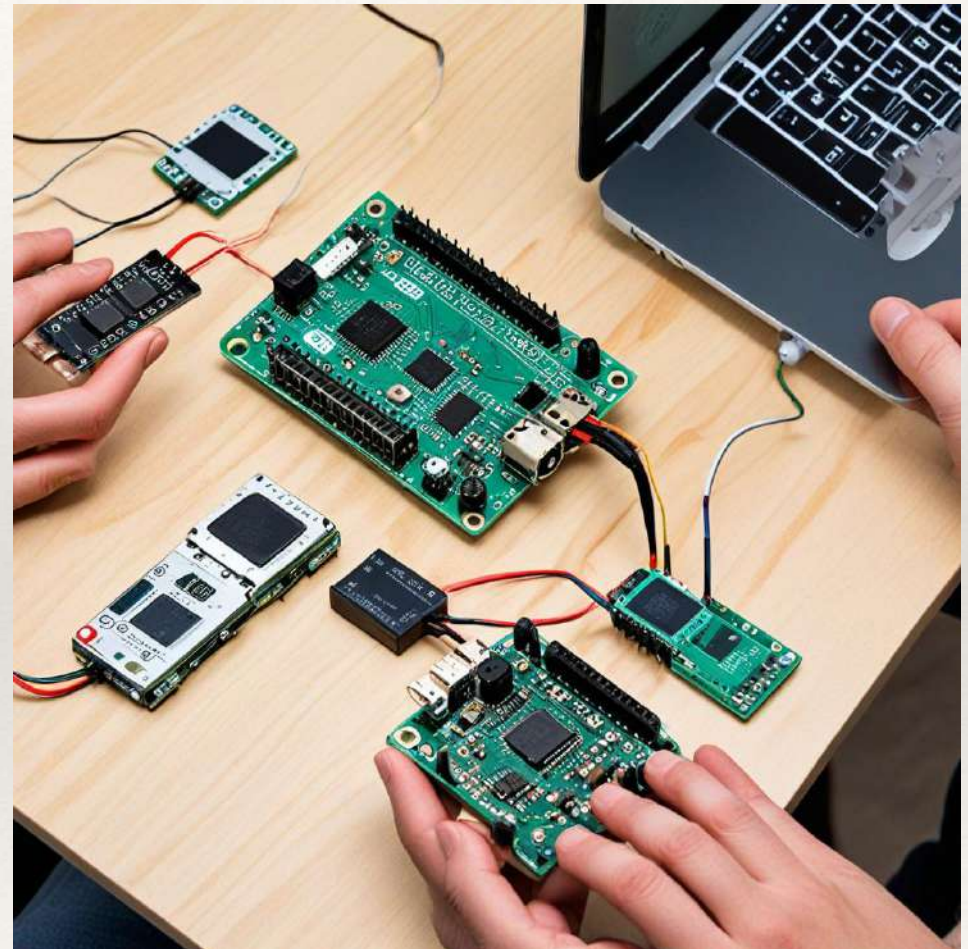
Interfaces sem fio.

Há diversos padrões internacionais

UART	ZigBee	Bluetooth
i2C	Ethernet	CAN
SPI	Wi-Fi	LoRa

Também há soluções proprietárias

Ex. ESP-Now



Comunicação - Protocolos

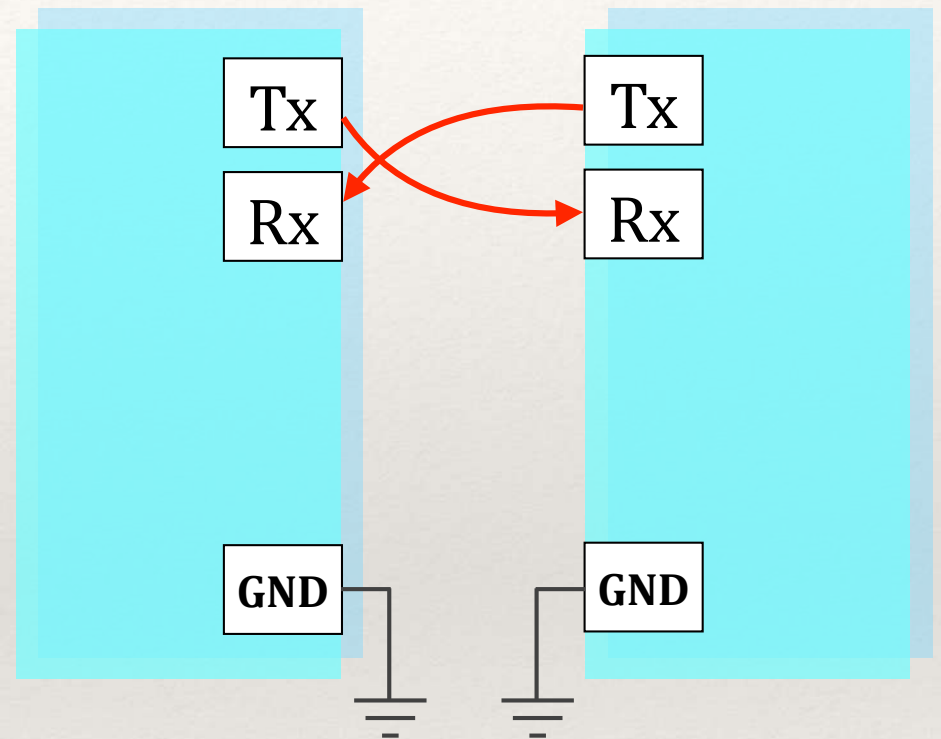
Protocolo	Mídia	Taxa	Alcance	Aplicação Típica
UART	Cabeada	< 5Mbps	1 Km	Muito simples, 2 fios, baixa performance, dois pontos, assíncrono, full-duplex, compatibilidade universal.
i2C	Cabeada	< 5Mbps	100 m	Simples, barramento 2 fios, baixa performance, até 1008 pontos, síncrono, Half-duplex, múltiplos controladores.
SPI	Cabeada	< 65Mbps	10 m	Complex.Média, barramento 4 fios, alta performance, "n" pontos (?), síncrono, full-duplex, 1 controlador.
ZigBee	Sem fio	< 250 Kbps	100 m	Simples, topologias estrela, árvore ou <i>mesh</i> , baixa performance, n ^o pontos (topologia ?) e baixo consumo.
CAN	Cabeada	< 1 Mbps	1 Km	Complex.Média, barramento 2 fios, média performance, "n" pontos, assíncrono, full-duplex, 1 controlador.
LoRa	Sem fio	< 5 Kbps	10 Km	Simples, topologia estrela ou <i>mesh</i> , baixíssima performance, "n" pontos, baixo consumo.
ESP-Now	Sem fio	< 1 Mbps	480 m	Simples, topologia estrela ou ponto a ponto, alta performance, 20 pontos, baixo consumo.

A opção UART

É um protocolo clássico, base de todas as comunicações seriais, como o RS-232, por exemplo;

É assíncrono (não utiliza *clock*);

Simplex, Half ou Full-duplex;



A opção UART

É um protocolo clássico, base de todas as comunicações seriais, como o RS-232, por exemplo;

É assíncrono (não utiliza *clock*);

Simplex, Half ou Full-duplex;

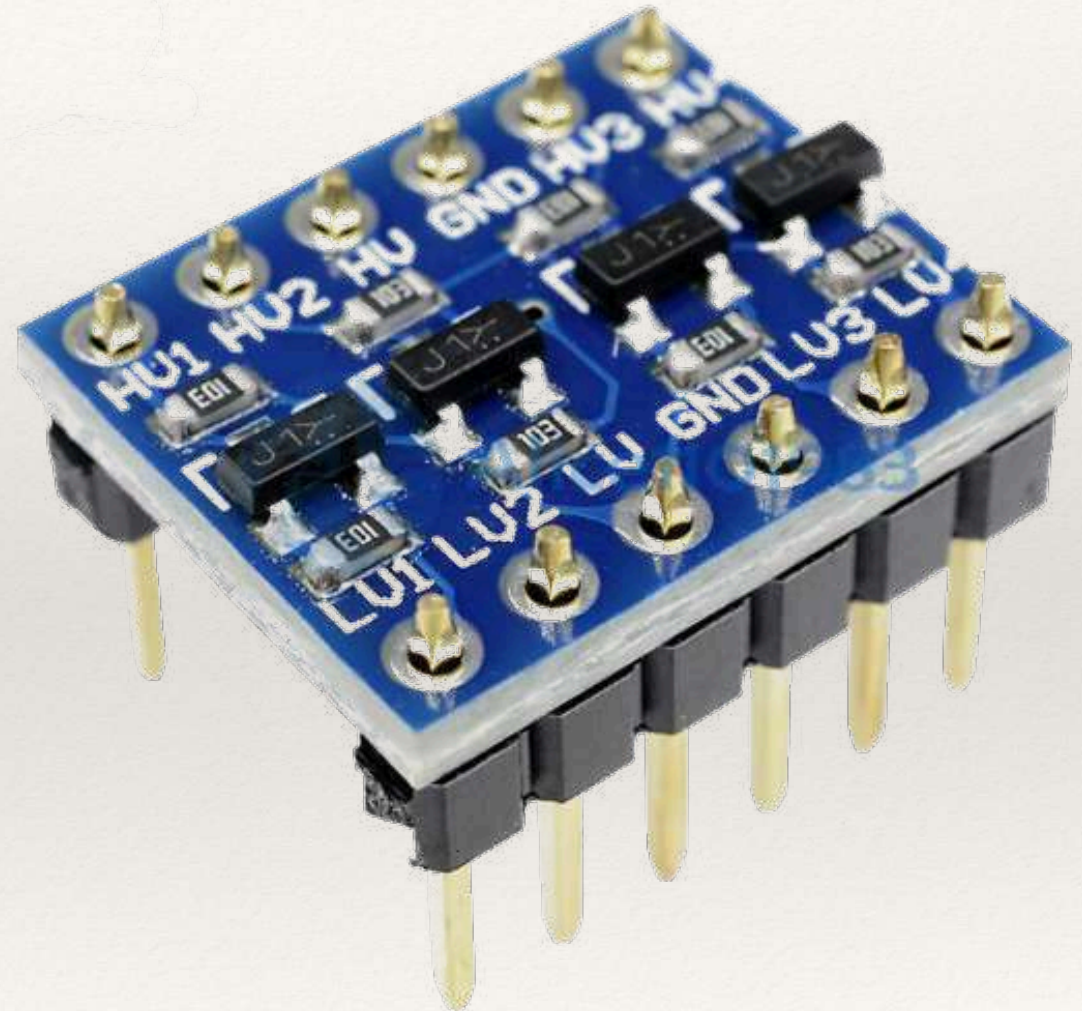
Compatibilizar níveis de tensão

Microcontroladores e sensores nem sempre usam o mesmo nível de tensão;

3,3 V $\rightarrow$ 5V : ok ?

5V $\rightarrow$ 3,3V : **não interligar !**

Divisores de tensão ou conversores de nível (foto) podem ser utilizados.



A opção UART

É um protocolo clássico, base de todas as comunicações seriais, como o RS-232, por exemplo;

É assíncrono (não utiliza *clock*);

Simplex, Half ou *Full-duplex*;

Compatibilizar níveis de tensão;

São 3 as informações de configuração:

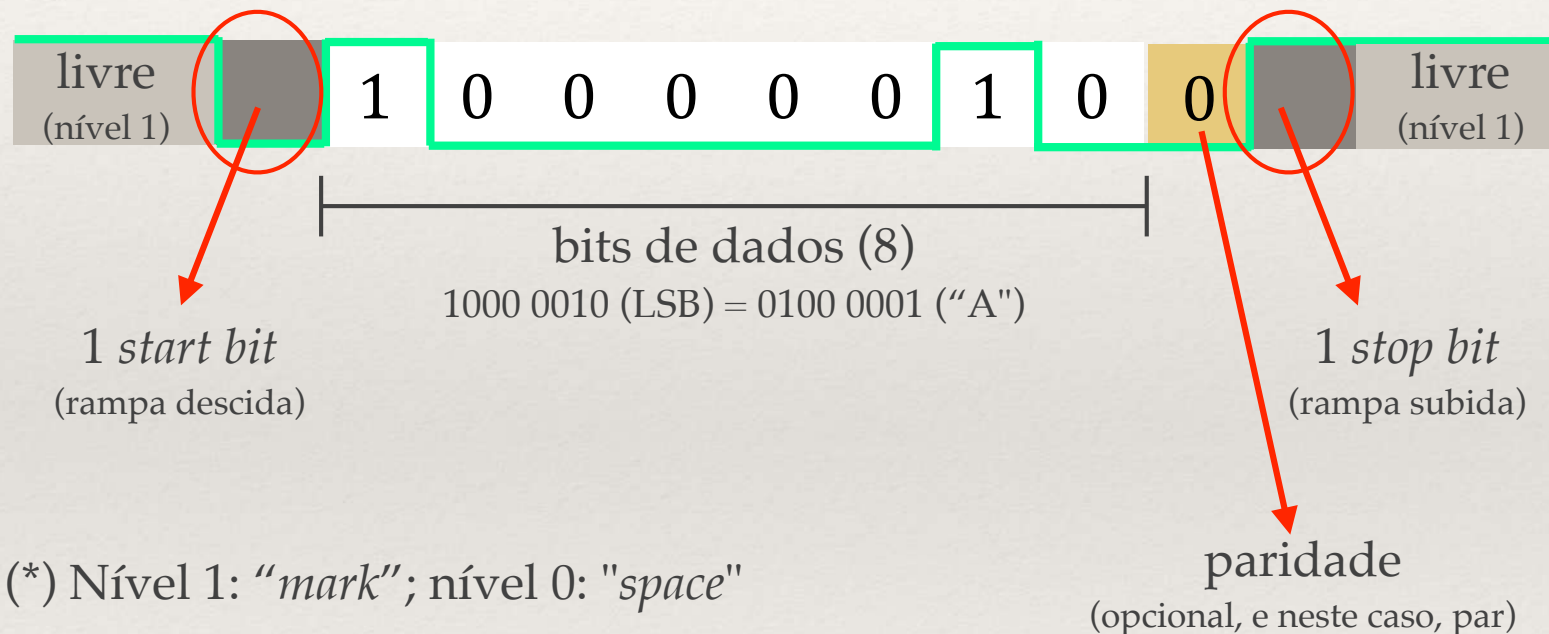
- Taxa de transferência;

- Comprimento em bits (5 a 9, típico 7-8);

- start* e *stop* bits;

Taxas Típicas
4.800 bps
9.600 bps
19.200 bps
57.600 bps
115.200 bps

UART Formato de quadro



UART Programação

Tomando como exemplo o Arduíno, temos os seguintes procedimentos para transmissão e recepção:

Transmissão:

Funciona como uma impressão serial;

O processo envolve:

```
Serial.begin( taxa,[configuração] )
```

Função **Serial.print** p/ transmissão.

Recepção:

Varredura frequente do *buffer* de entrada, ou por interrupção.

O processo envolve:

```
Serial.begin( taxa,[configuração] )
```

Verificar **Serial.available()**

Armazenar mensagem em variável

```
inChar = Serial.read()
```

A opção I²C

Protocolo criado pela Philips em 1982

O I²C não define todos os seus aspectos;

Nome, conector, e até mesmo tensão de operação são variáveis;

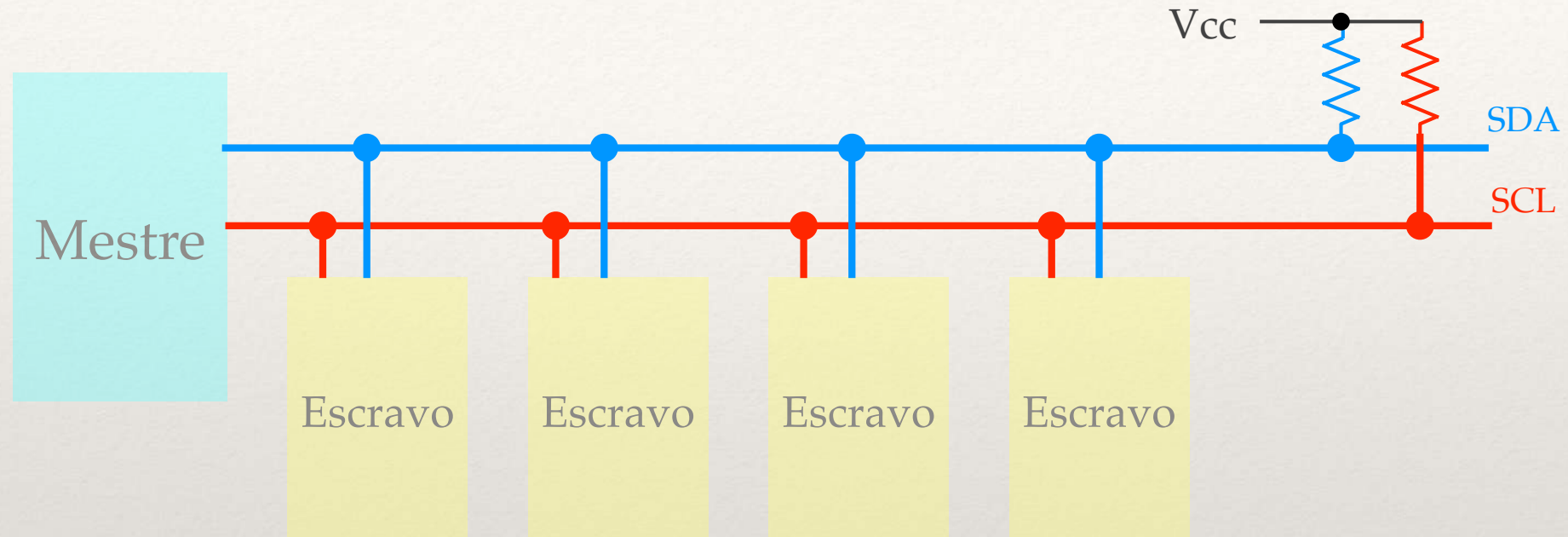
É muito comum internamente em celulares e outros dispositivos.

Opera em diversos modos

Standard (Sm)	100Kbps	High-Speed (Hs)	1,7Mbps
Fast (Fm)	400Kbps	High-Speed (Hs)	3,4Mbps
Fast plus (Fm+)	1Mbps	Ultra-Fast (UFm)	5Mbps (Unidirecional)

Utiliza 2 fios: SCL (clock) e SDA (serial data)

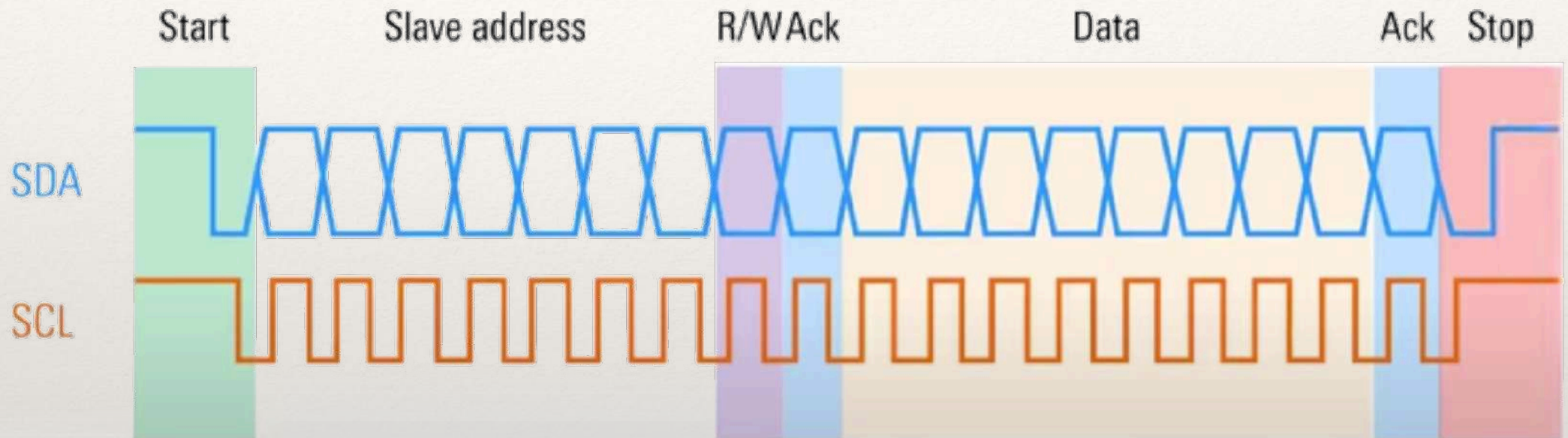
I²C - Topologia



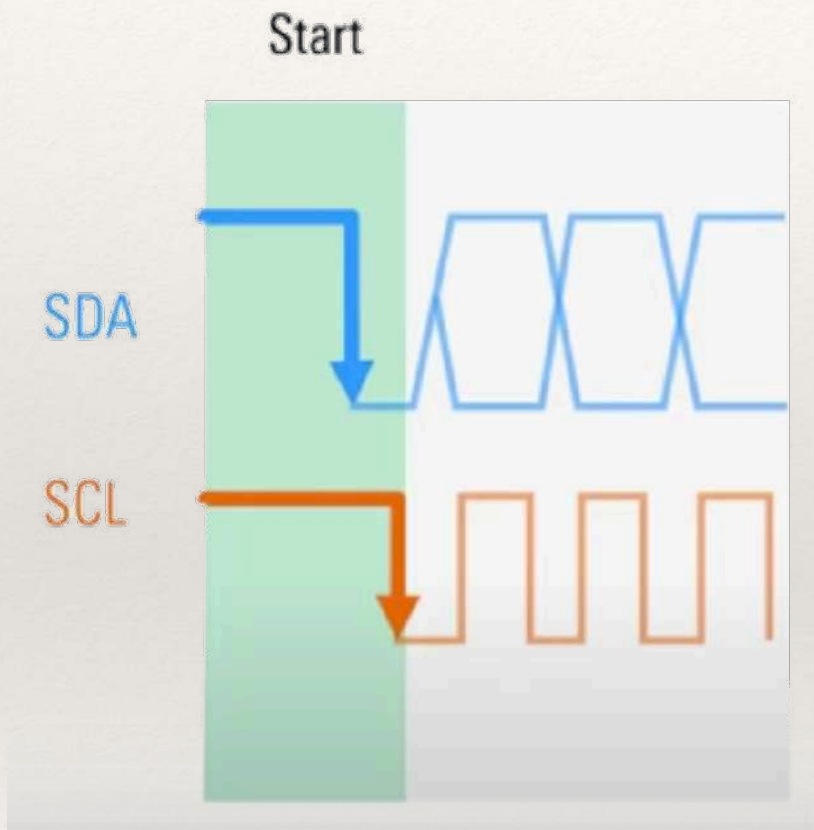
Barramento único

Admite entrada e saída de dispositivos a qualquer momento;
Cada dispositivo possui um endereço que o identifica.

I²C - Formato de Quadro



I²C - Formato de Quadro



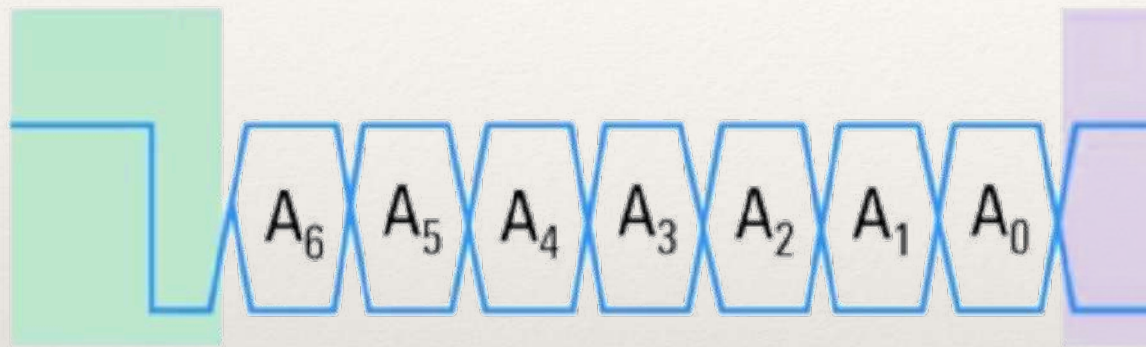
O valor "1" identifica que o barramento está livre;

Sequência determina *star bit*

SDA desce, seguido por SCL;

I²C - Formato de Quadro

Slave address



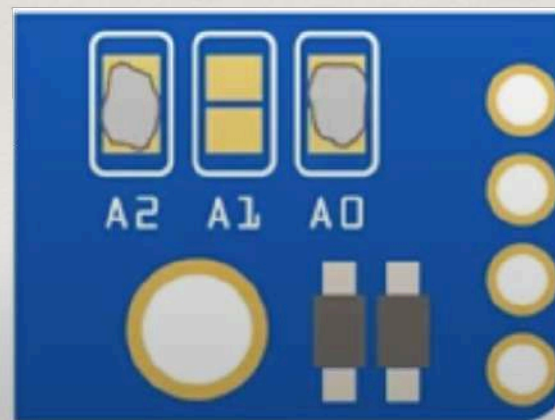
Cada dispositivo tem um endereço de 7 bits

Começa pelo MSB;

Endereço pode ser fixo, ou modificado por alterações na placa de circuito impresso do dispositivo.

Address: 0101A₂A₁A₀

Address: 0101101



I²C - Formato de Quadro

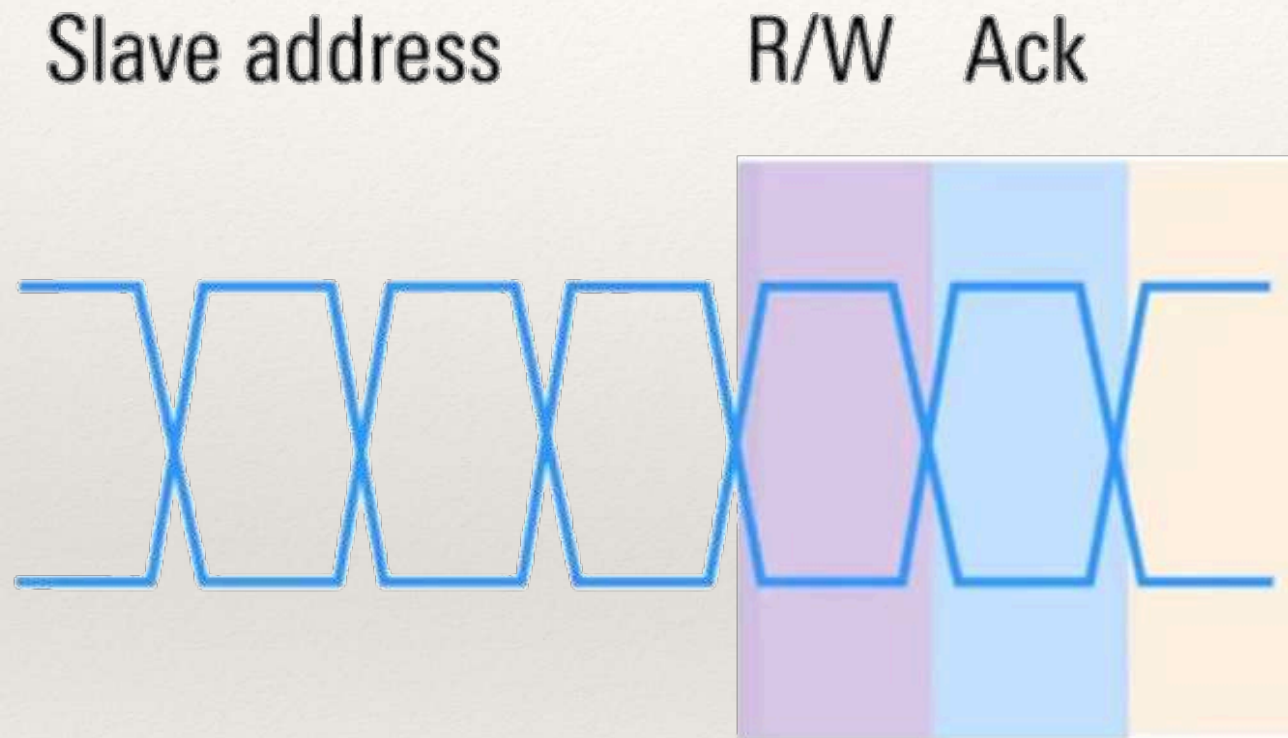


O bit R/W vem logo após o endereço, e é definido pelo Master

0 → Mestre escreve no escravo;

1 → Mestre lê o escravo;

I²C - Formato de Quadro



Definido pelo receptor, vai para zero para confirmação

1 → Condição livre do barramento = NACK;

ACK após dados

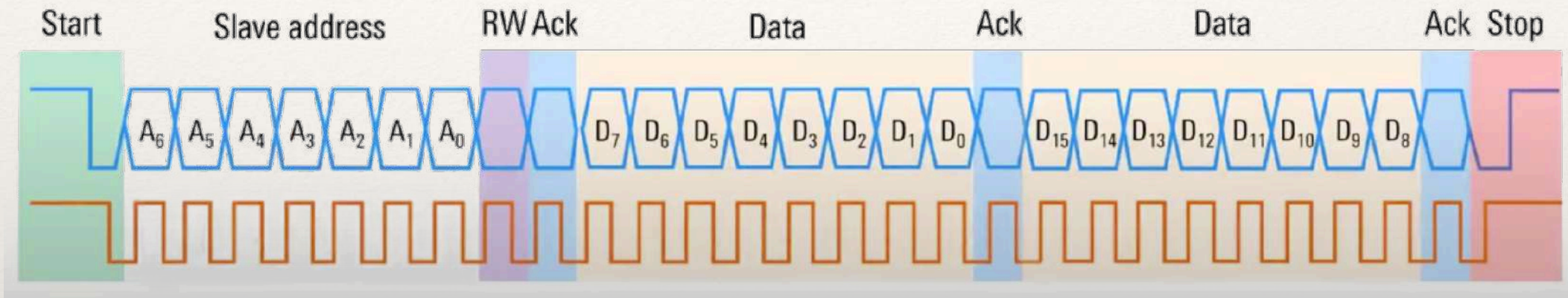
Confirme recebimento dos dados;

ACK após endereço do escravo

Escravo com aquele endereço existe;

Pronto para ler / escrever, de acordo com R/W.

I²C - Topologia



É comum encaminhar mais de um bloco de dados

Cada bloco individual tem sua confirmação (Ack) específica;

O I2C não define conteúdos, qualquer bloco pode transmitir qualquer coisa, mas ...

... é comum, na escrita de registradores de periféricos, que o primeiro bloco contenha o encereço, e o segundo os dados a serem escritos, por exemplo.

I²C - Formato de Quadro



Tudo ocorre de forma similar ao *start bit*;

A sequência inversa determina *stop bit*

SCL sobe, seguido por SDA;

I²C Programação

Tomando como exemplo o Arduíno, temos os seguintes procedimentos para transmissão e recepção:

Transmissão:

O processo envolve:

```
#include wire.h  
  
wire.begin( endereço )  
  
wire.write( mensagem )
```

Recepção:

O processo envolve:

```
#include wire.h  
  
wire.begin( endereço )  
  
Char = wire.read
```

A opção



Protocolo proprietário da Expressif, fabricante do ESP8266 e ESP32

- Utiliza o transceptor de 2,4 GHz já existente para Bluetooth e Wi-Fi;

- Utiliza um padrão de transmissão similar aos teclados e mouses sem fio;

- Não exige infraestrutura Wi-Fi (roteador, por exemplo);

- Aceita comunicação uni e bidirecional com conexões persistentes;

Algumas limitações

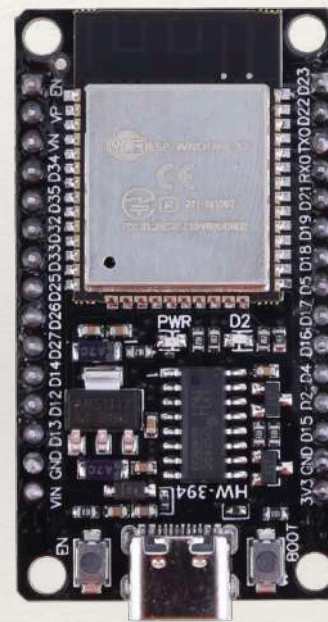
- Maior pacote deve ter 250 Bytes (mas permanece útil para SE);

- Podem haver entre 6 e 20 estações, a depender do modo de operação, e do uso de criptografia.

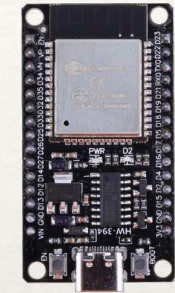
ESP-NOW Unidirecional

Um emissor (*iniciator*) e um receptor (*responder*);

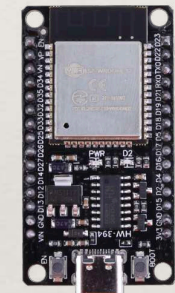
Um emissor e vários receptores individualmente, ou em *broadcast* (para todos);



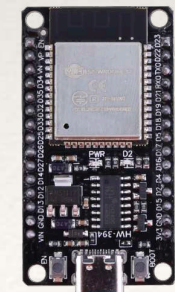
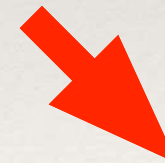
Initiator



Responder



Responder



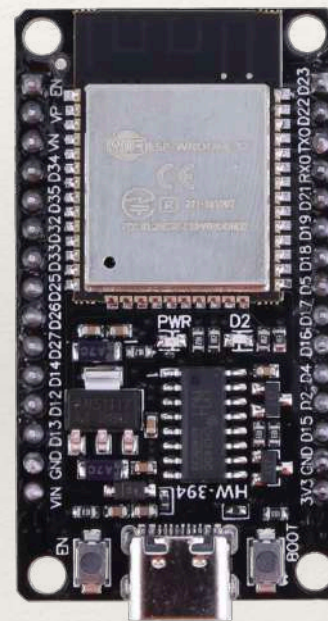
Responder

ESP-NOW Unidirecional

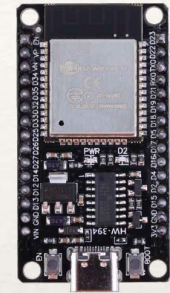
Um emissor (*iniciator*) e um receptor (*responder*);

Um emissor e vários receptores individualmente, ou em *broadcast* (para todos);

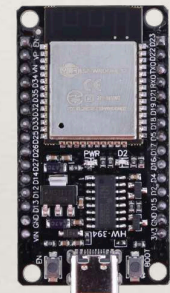
Vários emissores individualmente ou simultaneamente;



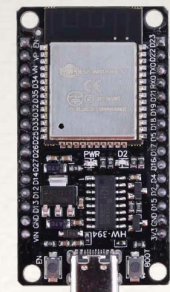
Responder



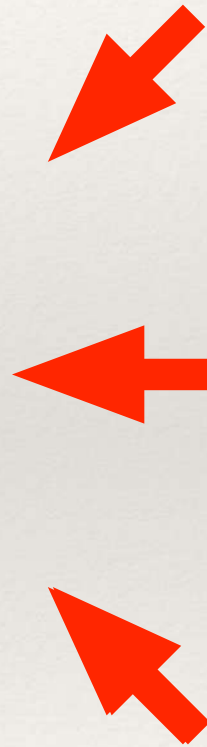
Iniciator



Iniciator

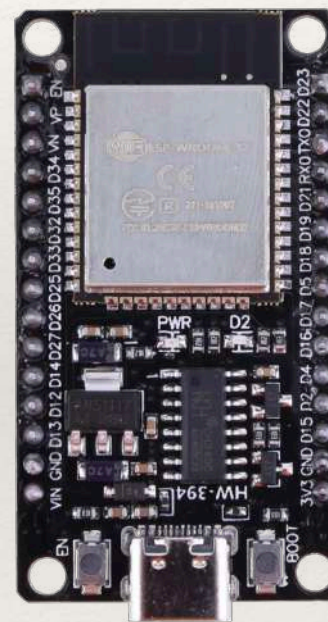


Iniciator

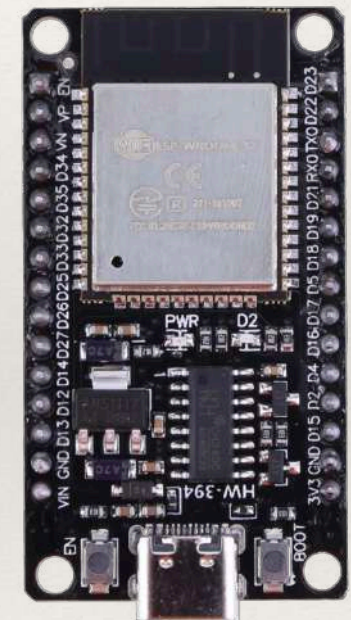
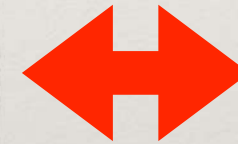


ESP-NOW Bidirecional

Dois dispositivos podem transmitir e ler ao mesmo tempo, atuando simultaneamente como emissor e receptor;



**Iniciator /
Responder**

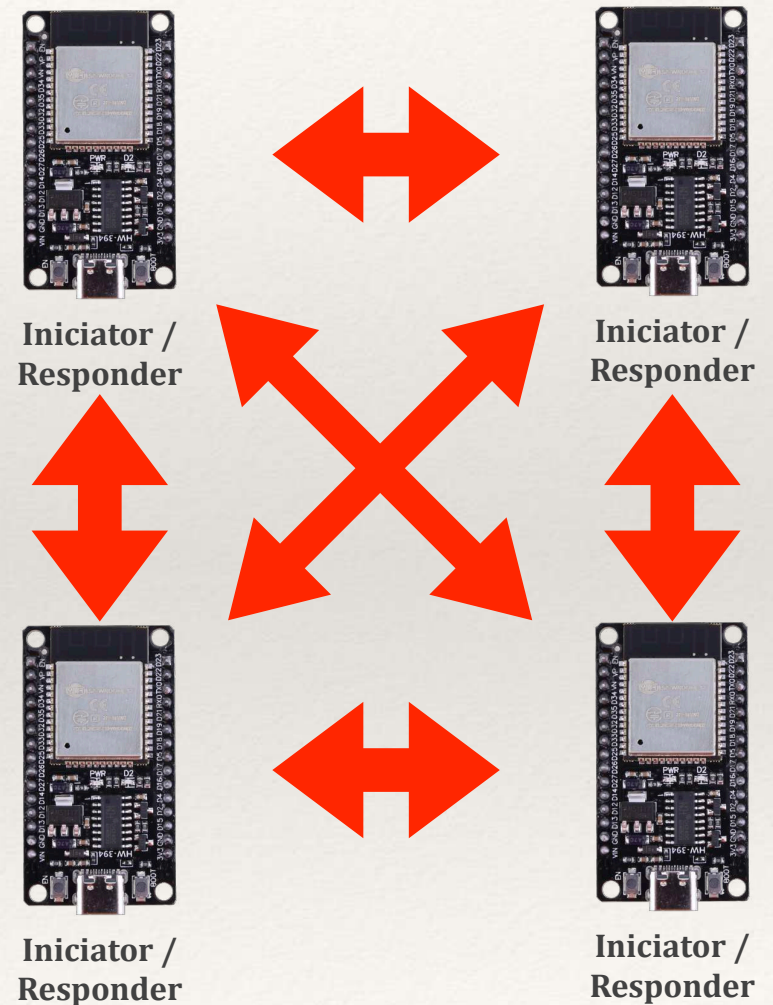


**Iniciator /
Responder**

ESP-NOW Bidirecional

Dois dispositivos podem transmitir e ler ao mesmo tempo, atuando simultaneamente como emissor e receptor;

Todos os dispositivos podem se comunicar com qualquer outro nas duas direções.



ESP-NOW Endereçamento

Utilizar o *MAC Address* do dispositivo para identificá-lo como destinatário;

O *MAC Address* é um número hexadecimal de 6 dígitos único gravado pelo fabricante;

(*) O ESP-NOW não suporta endereços de *broadcast* (FF:FF:FF:FF:FF:FF), mas existe forma de fazer um *pseudo-broadcast*.

Para obter o endereço, pode-se usar o *sketch* ao lado, ou mesmo verificar as mensagens de inicialização do dispositivo.



```
MostraMACAddress

/*
  ESP32 MAC Address printout
  MostraMACAddress.ino
  Mostra o endereço MAC do ESP32 no Terminal Serial

  Sketch obtido em:
  DroneBot Workshop 2022 ©
  https://dronebotworkshop.com
*/

// Inclui biblioteca WiFi
#include "WiFi.h"

void setup() {

  // Configura o terminal serial com taxa de 115200 bps
  // Lembrar de configurar a mesma taxa no Terminal Serial da IDE do Arduino
  Serial.begin(115200);

  // Coloca o ESP32 no modo "Station"
  WiFi.mode(WIFI_MODE_STA);

  // Mostra o endereço MAC no Terminal Serial
  Serial.print("MAC Address: ");
  Serial.println(WiFi.macAddress());
}

void loop() {

}
```

Funções de *callback*

Para a comunicação usando o ESP-NOW, utilizamos duas funções, chamadas na transmissão e recepção:

`esp_now_register_send_cb()`

Utilizada na transmissão, retorna *status* de envio e recepção;

O processo envolve:

- Iniciar o ESP-NOW

- Registrar a função de *callback*

- Adicionar um receptor (MAC Address)

- Enviar a mensagem

`esp_now_register_rcb_cb()`

Utilizada na recepção, retorna os dados recebidos;

O processo envolve:

- Iniciar o ESP-NOW

- Registrar a função de *callback*

- Salvar a mensagem recebida

Receptor

```
/*
ESP-NOW - RecebeMedicoesTemperatura
Teste do ESP-NOW com recepção múltipla
Recebe informações de temperatura de sensores conectados a outras ESP32 via ESP-NOW

Código inspirado na publicação de:
DroneBot Workshop 2022
https://dronebotworkshop.com
*/

// Inclui as Bibliotecas necessárias
#include <WiFi.h>
#include <esp_now.h>

// Define estrutura de dados (campo "a" do tipo "float" não foi utilizado)
typedef struct struct_message {
    //float a;
    float b;
    int c;
} struct_message;

// Cria objeto com base na estrutura
struct_message myData;

// função de "Callback"
void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len)
{
    // Lê os dados recebidos
    memcpy(&myData, incomingData, sizeof(myData));

    // Mostra dados no Monitor Serial
    Serial.print("Sensor ");
    Serial.print(myData.c);
    Serial.print(": ");
    Serial.println(myData.b);
    Serial.println("");
}
```

```
void setup() {
    // Configura taxa do Monitor Serial
    Serial.begin(115200);

    // Inicializa o Wi-Fi do ESP32 no modo "Estação"
    WiFi.mode(WIFI_STA);

    // Inicializa o ESP-NOW
    if (esp_now_init() != 0) {
        Serial.println("Erro inicializando o ESP-NOW");
        return;
    }

    // Registra a função de "callback"
    esp_now_register_recv_cb(OnDataRecv);
}

void loop() {
}
```

Transmissor

```
/*
 ESP-NOW EnviaTemperatura
 Teste do ESP-NOW com transmissão múltipla
 Envia valor da temperatura para um ESP32 central via ESP-NOW

 Código inspirado na publicação de:
 DroneBot Workshop 2022
 https://dronebotworkshop.com
*/

// Inclui as Bibliotecas necessárias
#include <WiFi.h>
#include <esp_now.h>

// Defini constante indicando o pino onde o sensor LM35 está conectado,
// ou seja, ao GPIO 34 (Analog ADC1_CH6)
const int pinoLM35 = 34;

// Identificador do transmissor (deve ser alterado para cada transmissor)
int ident = 1;

// Define o endereço MAC do "Responder"
uint8_t broadcastAddress[] = {0xE8, 0x6B, 0xEA, 0xD0, 0xE2, 0x18};

// Define estrutura de dados (campo "a" do tipo "float" não foi utilizado)
typedef struct struct_message {
    //float a;
    float b;
    int c;
} struct_message;

// Cria objeto com base na estrutura
struct_message myData;

// Register peer
esp_now_peer_info_t peerInfo;

// função de "Callback"
void OnDataSent(const uint8_t *macAddr, esp_now_send_status_t status)
{
    Serial.print("Status de envio do último pacote: ");
    Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Entrega bem sucedida" : "Entrega falhou");
}
```

```
void setup() {

    // Configura taxa do Monitor serial
    Serial.begin(115200);
    delay(100); // Aguarda estabilização (testar necessidade posteriormente)

    // Inicializa o Wi-Fi o ESP32 em modo "Estação"
    WiFi.mode(WIFI_STA);

    // Inicializa o ESP-NOW
    if (esp_now_init() != 0) {
        Serial.println("Erro inicializando o ESP-NOW");
        return;
    }

    // Registra a função de "callback"
    esp_now_register_send_cb(OnDataSent);

    memcpy(peerInfo.peer_addr, broadcastAddress, 6);
    peerInfo.channel = 0;
    peerInfo.encrypt = false;

    if (esp_now_add_peer(&peerInfo) != ESP_OK) {
        Serial.println("Falha ao adicionar destinatário");
        return;
    }
}

void loop() {

    int i, leituraADC;

    leituraADC = analogRead( pinoLM35 );

    myData.b = leituraADC * (3300.0 / 4096);
    myData.c = ident;

    // Envia dados
    esp_now_send(broadcastAddress, (uint8_t *) &myData, sizeof(myData));

    // Aguarda 2 seg para nova leitura do sensor
    delay(2000);
}
```

HTTP (*Hiper Text Transfer Protocol*)

O protocolo utiliza o TCP, e vem sendo aproveitado para outras aplicações diferentes do objetivo inicial (navegadores WEB)

- Simplicidade de especificação de protocolo

- Estabilidade operacional

Tradicionalmente, o HTTP operava com requisições e respostas

- Cada acesso envolvia o estabelecimento de uma conexão, transferência das informações e encerramento da conexão;

- Para garantir aplicações em tempo real / interativas, surgiu o *HTTP Websockets* e o *HTTP Push*;

HTTP (*Hiper Text Transfer Protocol*)

HTTP Websockets

Permite conexões persistentes e bidirecionais;

HTTP Push

Também chamado de HTTP Streaming, permite que servidores enviem respostas aos seus clientes independentes de requisições;

Permite atualizações em tempo real e comunicações orientadas a eventos;

Aplicação

O uso do HTTP é particularmente interessante quando a aplicação tem interação via WEB;

MQQT *(Message Queuing Telemetry Transport)*

Criado em 1999 por Andy Standor-Clark (IBM) e Arlen Nipper (Cirrus Link)

Baseado no modelo de Publicação / Assinatura

Define dois componentes: os **clientes** e os **brokers**;

Clientes podem publicar ou acessar informações

O acesso tipicamente se dá através de uma estrutura de tópicos pré-definida, onde clientes são “assinantes” ou através de filtros específicos;

Brokers atuam como intermediários, responsáveis por receber informações “publicadas”, e entregar estas informações para os clientes “assinantes”

Disponíveis em plataformas abertas e gratuitas (Ex. Mosquitto);

Este modelo “desacoplado” habilita aplicações eficientes e escaláveis

Suporte a ambientes com comunicação instável ?

Independência dos processos de envio e recepção;

Não é necessário um POST para cada cliente, como no HTTP.

MQQT (*Message Queuing Telemetry Transport*)

Utiliza o TCP, com recursos de QoS

Níveis: 0 - "at most once"; 1 - "at least once"; 2 - "exactly once"

Conexões persistentes

Protocolo leve, com *payload* binário

Uma mensagem MQQT pode ter apenas 2 Bytes !

Clientes/dispositivos de baixo consumo;

Taxas de transferência reduzidas (Ex.LoRa);

Habilita aplicações pervasivas;

HTTP x MQTT

	HTTP WEBSOCKETS	MQTT
Modelo	Full-duplex, Bidirecional	Publicação / Assinatura
Endereçamento	URIs	Tópicos
Protocolo de Transporte	TCP	TCP, UDP
Acompanhamento Cliente	N/D	Mensagens
Retenção de Mensagens	N/D	Sim
Modo de envio de mensagens	Bidirecional	Assíncrono
Arquitetura	Cliente/Servidor	Cliente/Servidor
Confiabilidade das Mensagens	WebSockets	QoS níveis 1 e 2

Aula 09