

CENTRO UNIVERSITÁRIO DO INSTITUTO MAUÁ DE TECNOLOGIA

ESCOLA DE ENGENHARIA MAUÁ

ENGENHARIA DE COMPUTAÇÃO

ANDRÉ SOLANO FERREIRA RODRIGUES MAIOLINI

RISC-V PROCESSOR

SÃO PAULO

2024

CENTRO UNIVERSITÁRIO DO INSTITUTO MAUÁ DE TECNOLOGIA

ESCOLA DE ENGENHARIA MAUÁ

ENGENHARIA DE COMPUTAÇÃO

ANDRÉ SOLANO FERREIRA RODRIGUES MAIOLINI

RISC-V PROCESSOR

Aplicando conceitos de Arquitetura de Computadores para o desenvolvimento de um processador RISC-V.

Orientador: Prof. Me. Núncio Perella

SÃO PAULO

2024

*Este trabalho é dedicado àqueles cuja curiosidade os
empurra sempre um pouco além do necessário.*



*“Fairy tales are more than true: not because they
tell us that dragons exist, but because they tell us
that dragons can be beaten.”
(Neil Gaiman)*

LISTA DE ILUSTRAÇÕES

- Figura 1 – Arquitetura geral de um núcleo RISC-V, destacando o *pipeline* de execução, o banco de registradores e as interfaces de memória. . . . 14

LISTA DE QUADROS

LISTA DE TABELAS

Tabela 1 – Instruções do Conjunto RV32I	15
---	----

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

ABNT Associação Brasileira de Normas Técnicas

abnTeX ABsurdas Normas para TeX

LISTA DE SÍMBOLOS

Γ	Letra grega Gama
Λ	Lambda
ζ	Letra grega minúscula zeta
\in	Pertence

SUMÁRIO

1	INTRODUÇÃO	11
1.1	Motivação	11
1.2	Objetivos do Projeto	12
1.3	Organização do Documento	13
2	FUNDAMENTAÇÃO	14
2.1	Arquitetura RISC-V	14
2.1.1	Conjunto de Instruções RV32I	15
2.1.2	Extensões relevantes	17
2.2	VHDL e Projeto de Hardware	18
2.3	Metodologias de Projeto	19
2.3.1	Desenvolvimento Modular e Hierárquico	19
2.3.2	Estratégia de Verificação e Simulação	19
3	MICROARQUITETURA	20
3.1	Visão Geral do Caminho de Dados	20
3.1.1	Unidade Lógica e Aritmética	20
3.1.2	Banco de Registradores	21
3.2	Implementação Single-Cycle (RV32I)	21
4	ORGANIZAÇÃO DE SISTEMAS E I/O	22
4.1	Comunicação entre CPU e Memória	22
4.2	Mapeamento de Memória (MMIO)	22
5	ECOSSISTEMA DE SOFTWARE (BARE-METAL)	23
5.1	Toolchain RISC-V e Compilação	23
5.2	Automação com Makefile	23
5.3	Integração com simulação e verificações	23
6	IMPLEMENTAÇÃO EM FPGA	24
6.1	Síntese	24
6.2	Pinagem e restrições	24
6.3	Testes em hardware	24
7	CONCLUSÃO	25
	REFERÊNCIAS	26

1 INTRODUÇÃO

A computação moderna é construída sobre camadas de abstração. Do clique de um mouse à execução de uma inteligência artificial, existem bilhões de transistores operando em uma sinfonia lógica precisa. No entanto, para a maioria dos profissionais da área, essa base opera como uma "caixa-preta". O hardware tornou-se algo que se usa, mas não necessariamente se compreende.

Este trabalho propõe romper com essa visão utilitária. O projeto aqui documentado não é apenas a construção de um microprocessador, mas, uma investigação prática sobre os fundamentos que sustentam toda a era digital. Ao projetar, implementar e validar um sistema computacional completo, buscamos resgatar o domínio sobre a camada de hardware subjacente.

A arquitetura escolhida para guiar esta jornada é o RISC-V, um conjunto de instruções (ISA) aberto que, por sua simplicidade e modularidade, serve como o veículo perfeito para o estudo de arquitetura e organização de computadores.

1.1 Motivação

Por que, na era da computação em nuvem e linguagens de altíssimo nível, um engenheiro deve despender tempo entendendo barramentos, datapaths e sinais de controle? Apesar do avanço das abstrações oferecidas por linguagens modernas, frameworks e plataformas em nuvem, os sistemas computacionais continuam fundamentados em princípios microarquiteturais que determinam seu desempenho, seu comportamento e suas limitações. Esses conhecimentos permanecem essenciais porque tais componentes constituem o nível de base sobre o qual todas as abstrações são construídas. Sem essa compreensão, o engenheiro passa a operar máquinas sem entender suas características - tais como: latência, gargalos, paralelismo, hierarquia de memória e restrições — que afetam diretamente a eficiência e a confiabilidade de qualquer solução de software ou hardware desenvolvida.

Do ponto de vista formativo, a familiaridade com os elementos internos do processador também capacita o engenheiro a compreender o funcionamento da cadeia completa

de execução — da especificação de uma instrução ao seu efeito no hardware físico. Essa visão abrangente é indispensável em áreas como sistemas embarcados, computação de alto desempenho, aceleradores especializados, dispositivos IoT, aplicações críticas e de tempo real, nas quais a interação direta com o hardware é frequente e decisiva.

Em suma, mesmo em um cenário em que as abstrações se tornam cada vez mais sofisticadas, a compreensão do nível fundamental da computação permite que o engenheiro avalie limitações, explore otimizações, antecipe comportamentos do sistema e desenvolva soluções mais robustas e eficientes. Assim, o estudo desses conceitos continua sendo um elemento central na formação e na prática da engenharia moderna.

1.2 Objetivos do Projeto

O objetivo central deste trabalho é suprir o abismo entre estudar diagramas de livros-texto e fazer um sinal elétrico acender um pixel na tela. Para tal, projetando um System-on-Chip (SoC) baseado na arquitetura RISC-V, validando os conceitos teóricos de organização de computadores através de uma aplicação prática.

Portanto, visando prover uma experiência prática acerca dos tópicos estudados, os objetivos específicos desse projeto são:

1. **Microarquitetura:** Implementar um núcleo de processamento funcional, detalhando o fluxo de dados e a lógica de controle.
2. **Organização de Sistemas:** Desenvolver uma infraestrutura de comunicação (Barra-mento) que demonstre na prática o conceito de Mapeamento de Memória (MMIO), permitindo que a CPU interaja com o mundo externo.
3. **Interface Hardware/Software:** Construir o ecossistema necessário para executar código C em um ambiente sem sistema operacional (*bare-metal*), desmistificando o funcionamento de bibliotecas padrão.

1.3 Organização do Documento

Este documento está organizado de forma a acompanhar evolução do projeto - abordando a concepção da arquitetura e desenvolvimento da microarquitetura, organização do sistema, até a execução de aplicações e simulações:

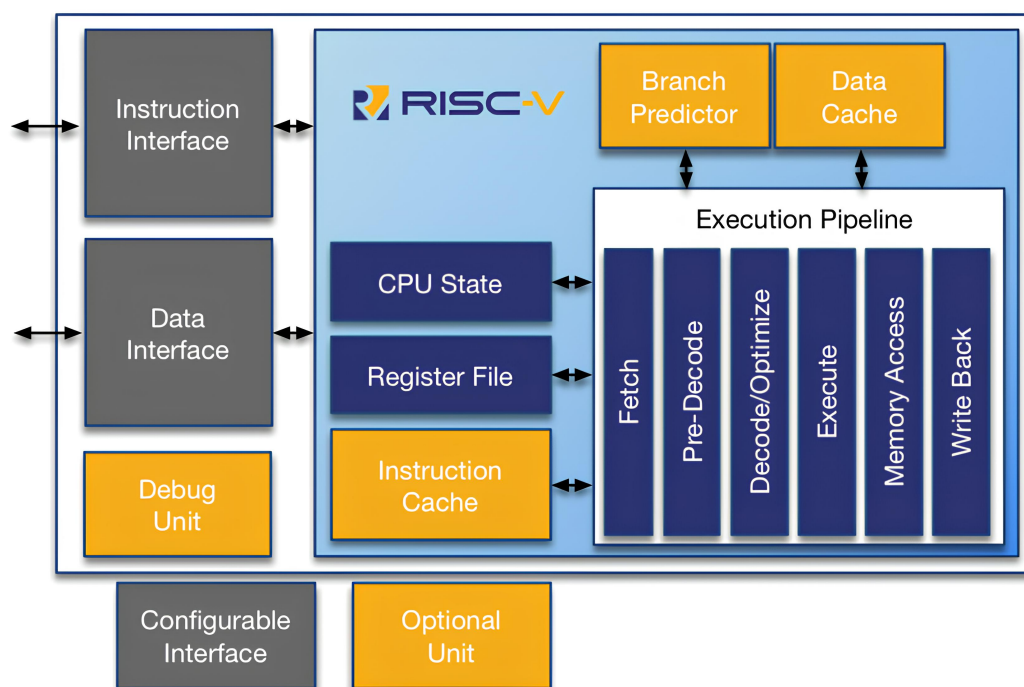
1. O **Capítulo 2** fundamenta as escolhas de arquitetura (RISC-V) e ferramentas.
2. O **Capítulo 3** descreve o coração do sistema: o processador e sua microarquitetura.
3. O **Capítulo 4** expande o horizonte para o sistema completo, detalhando o Barramento e a organização da memória.
4. O **Capítulo 5** aborda a camada invisível de software que torna o hardware utilizável.
5. O **Capítulo 6** apresenta a implementação em um hardware real, através de uma placa FPGA, consolidando todos os conceitos em uma aplicação real.

2 FUNDAMENTAÇÃO

2.1 Arquitetura RISC-V

O RISC-V é uma ISA (Instruction Set Architecture) baseada no conceito RISC (Reduced Instruction Set Computer). Diferente de arquiteturas proprietárias (como x86 e ARM), a RISC-V é aberta e livre de royalties, permitindo que universidades, empresas e desenvolvedores a utilizem e modifiquem.

Figura 1 – Arquitetura geral de um núcleo RISC-V, destacando o *pipeline* de execução, o banco de registradores e as interfaces de memória.



Fonte: O Autor (2017)

As principais características da arquitetura RISC-V incluem:

1. Modularidade: núcleo mínimo (base ISA) e extensões opcionais.
2. Simplicidade: instruções de tamanho fixo (geralmente 32 bits).

3. Portabilidade: projetada para uso desde microcontroladores até supercomputadores.
4. Ecossistema aberto: suporte crescente em compiladores, simuladores e hardware.

Apesar do domínio da ARM no mercado de embarcados e móveis, a flexibilidade, o custo zero de licenciamento e a capacidade de customização do RISC-V o posicionam como um competidor formidável e uma alternativa estratégica para o futuro da computação.

Assim como processadores de arquitetura ARM (*Advanced RISC Machine*), os RISC-V recebem esse nome por serem baseados nos princípios RISC - computação com conjunto de instruções reduzido (*Reduced Instruction Set Computing*).

Diferentemente da arquitetura CISC (*Complex Instruction Set Computing*), que se baseia em um vasto conjunto de instruções complexas e especializadas, o RISC-V adota a filosofia do "menos é mais". Ao simplificar o conjunto de instruções, a arquitetura permite um hardware mais eficiente e previsível.

2.1.1 Conjunto de Instruções RV32I

O conjunto de instruções **RV32I** é a especificação base da arquitetura RISC-V para processadores de 32 bits. O nome indica que é um processador RISC-V de 32 bits de largura da palavra de processamento (*word*) e utiliza a extensão de números inteiros (*Integer*) - que é a base para todas as outras. A Tabela 1 lista as instruções base suportadas pelo núcleo RV32I.

Tabela 1 – Instruções do Conjunto RV32I

Categoria	Mnemônico	Descrição
R-Type	ADD	Soma aritmética entre registradores.
	SUB	Subtração aritmética entre registradores.
	SLL	Deslocamento lógico à esquerda (<i>Shift Left Logical</i>).
	SLT	Definir se menor que (<i>Set Less Than</i> , com sinal).
	SLTU	Definir se menor que (<i>Set Less Than Unsigned</i> , sem sinal).
	XOR	Ou exclusivo bit a bit.

Categoria	Mnemônico	Descrição
	SRL	Deslocamento lógico à direita (<i>Shift Right Logical</i>).
	SRA	Deslocamento aritmético à direita (<i>Shift Right Arithmetic</i>).
	OR	Ou lógico bit a bit.
	AND	E lógico bit a bit.
I-Type	ADDI	Soma com valor imediato.
	SLTI	Definir se menor que imediato (com sinal).
	SLTIU	Definir se menor que imediato (sem sinal).
	XORI	Ou exclusivo imediato.
	ORI	Ou lógico imediato.
	ANDI	E lógico imediato.
	LLI	Deslocamento lógico à esquerda imediato.
	SRLI	Deslocamento lógico à direita imediato.
	SRAI	Deslocamento aritmético à direita imediato.
U-Type	LUI	Carregar Imediato Superior (<i>Load Upper Immediate</i>). Pre-enche os 20 bits mais significativos.
	AUIPC	Adicionar Imediato Superior ao PC (<i>Add Upper Immediate to PC</i>).
Jumps	JAL	Salto e Link (<i>Jump and Link</i>). Salto incondicional relativo ao PC.
	JALR	Salto e Link Registrador (<i>Jump and Link Register</i>). Salto incondicional via registrador.
Branches	BEQ	Desvio se Igual (<i>Branch if Equal</i>).
	BNE	Desvio se Diferente (<i>Branch if Not Equal</i>).
	BLT	Desvio se Menor (<i>Branch if Less Than</i> , com sinal).
	BGE	Desvio se Maior ou Igual (<i>Branch if Greater/Equal</i> , com sinal).
	BLTU	Desvio se Menor (<i>Branch if Less Than</i> , sem sinal).
	BGEU	Desvio se Maior ou Igual (<i>Branch if Greater/Equal</i> , sem sinal).

Categoria	Mnemônico	Descrição
Loads	LB	Carregar Byte (estende sinal).
	LH	Carregar Meia-Palavra (<i>half-word</i> , estende sinal).
	LW	Carregar Palavra (<i>word</i> , 32 bits).
	LBU	Carregar Byte (sem sinal, preenche com zeros).
	LHU	Carregar Meia-Palavra (sem sinal, preenche com zeros).
Stores	SB	Armazenar Byte.
	SH	Armazenar Meia-Palavra.
	SW	Armazenar Palavra.
System	FENCE	Sincronização de memória e I/O.
	ECALL	Chamada de sistema (<i>Environment Call</i>).
	EBREAK	Paragem para depuração (<i>Environment Break</i>).

É a ISA mais simples e fundamental, servindo como ponto de partida obrigatório. Sozinha, ela é suficiente para a criação de um processador de propósito geral e para a execução de um código compilado em C completo - usando a toolchain da RISC-V.

Um processador que se diz compatível com **RV32IMC**, por exemplo, implementa a base de inteiros (I), as operações de multiplicação/divisão (M) e instruções comprimidas de 16 bits (C).

2.1.2 Extensões relevantes

A arquitetura RISC-V distingue-se pela sua modularidade. O conjunto de instruções base (como o RV32I implementado neste trabalho) é a única parte obrigatória; sobre ela, projetistas podem acoplar módulos opcionais para otimizar o desempenho em domínios específicos.

As extensões padronizadas são identificadas por letras sufixas à nomenclatura do processador. As principais são:

- **M (*Integer Multiplication and Division*)**: Adiciona hardware dedicado para multiplicação e divisão de inteiros. Sem esta extensão, estas operações devem ser

realizadas via software (rotinas de emulação), o que é significativamente mais lento.

- **A (*Atomic*)**: Fornece instruções de leitura-modificação-escrita atômicas. Estas são essenciais para a sincronização de memória e gerenciamento de exclusão mútua em sistemas operacionais e arquiteturas *multicore*.
- **F/D (*Single/Double Precision Floating-Point*)**: Adiciona um banco de registradores separado e unidades de cálculo para operações com ponto flutuante, seguindo o padrão IEEE 754 (32 bits para F e 64 bits para D).
- **C (*Compressed*)**: Introduz um conjunto de instruções de 16 bits que mapeiam para as operações mais comuns de 32 bits. Isso aumenta a densidade de código, reduzindo o tamanho dos binários e o uso de largura de banda de memória.
- **V (*Vector*)**: Habilita operações vetoriais (SIMD - *Single Instruction, Multiple Data*), permitindo o processamento paralelo de grandes volumes de dados, comum em aplicações de IA e processamento de sinal.
- **B (*Bit Manipulation*)**: Oferece instruções eficientes para manipulação de bits individuais (rotações, contagens, extrações), úteis em criptografia e drivers de dispositivos.

É comum na literatura encontrar a designação **RV32G** ou **RV64G**, onde 'G' (*General*) representa um agrupamento das extensões mais utilizadas em sistemas de propósito geral.

Além do padrão, a ISA reserva espaço de opcodes para **extensões customizadas (X)**. Esta característica permite a criação de aceleradores de hardware proprietários para tarefas especializadas (domain-specific architectures) sem quebrar a compatibilidade com o ecossistema de software base.

2.2 VHDL e Projeto de Hardware

A descrição de hardware deste projeto foi realizada utilizando a linguagem **VHDL** (*VHSIC Hardware Description Language*). Diferente de linguagens de programação

de software, que descrevem sequências de instruções, o VHDL é uma linguagem de descrição de hardware utilizada para modelar o comportamento concorrente e a estrutura de circuitos digitais.

Para este trabalho, adotou-se especificamente o padrão **VHDL-2008** (IEEE 1076-2008). Esta revisão da linguagem introduz construções modernas que aumentam a legibilidade e a capacidade de síntese do código, permitindo uma descrição mais concisa de lógicas complexas sem sacrificar o rigor tipológico característico da linguagem.

2.3 Metodologias de Projeto

A complexidade inerente ao desenvolvimento de um microprocessador exige uma metodologia de projeto rigorosa e estruturada. Este trabalho seguiu uma abordagem de desenvolvimento **Bottom-Up** (de baixo para cima) combinada com **Prototipagem Evolutiva**.

2.3.1 Desenvolvimento Modular e Hierárquico

O sistema foi decomposto em módulos fundamentais de menor complexidade. O desenvolvimento iniciou-se pelos componentes base do *datapath* (Multiplexadores, ALU, Banco de Registradores), progredindo para a Unidade de Controle e, finalmente, para a integração no nível de topo (*Top Level*). Cada módulo foi projetado como uma entidade independente, com interfaces (*ports*) claramente definidas, facilitando o isolamento de falhas.

2.3.2 Estratégia de Verificação e Simulação

Dada a dificuldade de depurar erros diretamente no hardware físico, adotou-se a abordagem de testes através de simulações (*textbenches*). Nenhum componente foi integrado ao sistema principal sem antes ser validado individualmente.

O ambiente de simulação utilizado foi o **GHDL**, um simulador *open-source* de alto desempenho, integrado a um fluxo de automação via **GNU Make**. Isso garante que todo o processo de compilação, simulação e visualização de ondas (via GTKWave) seja reproduzível e automatizado.

3 MICROARQUITETURA

A microarquitetura define a organização lógica e física dos componentes internos do processador para implementar o conjunto de instruções (ISA) alvo. Neste capítulo, detalha-se a construção do núcleo RISC-V, focando na separação clássica entre o **Caminho de Dados** (*Datapath*) — responsável pelo armazenamento e manipulação das informações — e a **Unidade de Controle** (*Control*) — responsável por orquestrar o fluxo de dados através de sinais de controle.

3.1 Visão Geral do Caminho de Dados

O Caminho de Dados (*Datapath*) é o 'músculo' do processador. Ele é composto por elementos de estado (memórias e registradores) e elementos combinacionais (ALU, somadores, multiplexadores).

O fluxo principal de execução segue a seguinte sequência lógica dentro do hardware:

1. **Busca (Fetch):** O Contador de Programa (PC) endereça a memória de instruções.
2. **Decodificação (Decode):** A instrução é quebrada em campos; os registradores fonte são lidos e valores imediatos são estendidos.
3. **Execução (Execute):** A ALU realiza operações aritméticas/lógicas ou calcula endereços de memória.
4. **Memória (Memory):** Dados são lidos ou escritos na Memória de Dados (se aplicável).
5. **Escrita (Write-Back):** O resultado é escrito de volta no banco de registradores.

3.1.1 Unidade Lógica e Aritmética

A Unidade Lógica e Aritmética (ALU) é um circuito puramente combinacional responsável por todas as operações matemáticas e lógicas do processador. A implementa-

ção descrita no arquivo `alu.vhd` opera com duas entradas de 32 bits e gera um resultado de 32 bits, além de *flags* de estado (Zero e Negativo).

As operações suportadas foram mapeadas através de um sinal de controle de 4 bits (`ALUControl`), permitindo:

3.1.2 Banco de Registradores

3.2 Implementação Single-Cycle (RV32I)

4 ORGANIZAÇÃO DE SISTEMAS E I/O

4.1 Comunicação entre CPU e Memória

4.2 Mapeamento de Memória (MMIO)

5 ECOSSISTEMA DE SOFTWARE (BARE-METAL)

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

5.1 Toolchain RISC-V e Compilação

5.2 Automação com Makefile

5.3 Integração com simulação e verificações

6 IMPLEMENTAÇÃO EM FPGA

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

6.1 Síntese

6.2 Pinagem e restrições

6.3 Testes em hardware

7 CONCLUSÃO

Sed consequat tellus et tortor. Ut tempor laoreet quam. Nullam id wisi a libero tristique semper. Nullam nisl massa, rutrum ut, egestas semper, mollis id, leo. Nulla ac massa eu risus blandit mattis. Mauris ut nunc. In hac habitasse platea dictumst. Aliquam eget tortor. Quisque dapibus pede in erat. Nunc enim. In dui nulla, commodo at, consectetur nec, malesuada nec, elit. Aliquam ornare tellus eu urna. Sed nec metus. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

Phasellus id magna. Duis malesuada interdum arcu. Integer metus. Morbi pulvinar pellentesque mi. Suspendisse sed est eu magna molestie egestas. Quisque mi lorem, pulvinar eget, egestas quis, luctus at, ante. Proin auctor vehicula purus. Fusce ac nisl aliquam ante hendrerit pellentesque. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Morbi wisi. Etiam arcu mauris, facilisis sed, eleifend non, nonummy ut, pede. Cras ut lacus tempor metus mollis placerat. Vivamus eu tortor vel metus interdum malesuada.

Sed eleifend, eros sit amet faucibus elementum, urna sapien consectetur mauris, quis egestas leo justo non risus. Morbi non felis ac libero vulputate fringilla. Mauris libero eros, lacinia non, sodales quis, dapibus porttitor, pede. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Morbi dapibus mauris condimentum nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Etiam sit amet erat. Nulla varius. Etiam tincidunt dui vitae turpis. Donec leo. Morbi vulputate convallis est. Integer aliquet. Pellentesque aliquet sodales urna.

REFERÊNCIAS