# Assembly RISC-V
# Simulação de execução

**Thales L. Menezes, 17/0045919**

[1]Dep. Ciência da Computação – Universidade de Brasília (UnB)
CiC 116394 - Organização e Arquitetura de Computadores - Turma C

thalesmenezes13@gmail.com

***Resumo.*** *Nesse relatório é retratado o processo de implementação, em linguagem C++, da arquitetura (parcial) da linguagem assembler RISC-V.*

## 1. Introdução

Podemos dividir em duas categorias as linguagens de programção: alto e baixo nível, categorias relativas à proximidade com o hardware. C#, Java, C++, C são consideradas linguagens de alto nível, pelas suas abstrações de dados, paradigmas associados e comandos complexos; enquanto que *linguagens assembler* são baseadas em comandos executados diretamente pelo processador, sendo assim consideradas linguagens de baixo nível.

A linguagem analisada ao longo desse projeto é uma linguagem RISC (*Reduced Instruction Set Computer*), ou seja, uma linguagem formada de operações consideradas essenciais ao computador.

## 2. Objetivos

Implementar um ambiente de simulação das instruções da linguagem *assembler* RISC-V fazendo uso da linguagem C++.

## 3. Procedimentos

Tomando por base as instruções de acesso à memória previamente implementadas, foram construídos dois cabeçalhos (*word.hpp e memory.hpp*) e testes de unidade (utilizando a biblioteca Catch2) foram propostos; foi utilizada a metodologia de desenvolvimento baseada em testes (TDD), assegurando grande robustez e confiabilidade no código construído, possibilitando segurança ao acrescentar novos módulos.

O simulador foi planejado de acordo com os seguintes módulos:

- word.hpp
  - Relacionada à manipulação de *bits*.
- memory.hpp
  - Relacionada à manipulação da memória RAM simulada.
- control.hpp
  - Encapsula as memórias RAM e banco de registradores.
- decode.hpp
  - Responsável pela decodificação das instruções.

- execute.hpp
  - Responsável pela execução das instruções decodificadas.

O projeto pode ser compilado através do arquivo makefile, contido na pasta raiz, o qual gera um arquivo executável de nome ”rars”.

A suíte de testes pode ser compilada através do mesmo makefile usando o comando ”make test”, o qual gera o executável ”suit”, que é um conjunto de todos os testes unitários do projeto (acessíveis na pasta tests).

## 4. Programas simulados

Além dos testes unitários, o simulador foi submetido a alguns códigos disponibilizados pelo professor, dos quais destaco.

- Oito Primos:

```
Os oito primeiros numeros primos sao : 1 3 5 7 11 13 17 19
-- program is finished running --
```

- Fibonacci:

```
The Fibonacci numbers are:
1 1 2 3 5 8 13 21 34 55 89 144
-- program is finished running --
```

- Decimal para Binário Invertido:

```
Decimal: 96

Inverted Binary: 0000011

-- program is finished running --
```

## 5. Análise dos Resultados

Foi um grande desafio implementar a grande quantidade de recursos que a linguagem oferece, apesar de ser uma de suas versões mais básicas, entretanto acredito que ainda exista espaço para melhorias e otimizações, tanto para a execução do código de baixo nível quanto para a utilização dos módulos implementados no simulador.

## 6. Conclusão

Estudar uma linguagem RISC nos apresenta a uma nova perpectiva dos recursos computacionais que utilizamos e da beleza de suas implementações; também nos possibilita uma visão muito clara de funcionamento do computador, especialmente em relação à custo de execução e memória, o que torna esse estudo uma experiência altamente enriquecedora ao programador como profissional.