UNIVERSIDADE CATÓLICA DE SANTOS

Centro de Ciências Exatas, Arquitetura e Engenharias

Graduação em Ciência da Computação/Sistemas de Informação

ARTHUR IWANKIU CASTRO
ENZO DA SILVA PASSOS

Documentação do Projeto: Simulador de Computador RISC-V (RV32I)

Santos - SP

2025

Sumário

1. VISÃO GERAL

Este projeto consiste na implementação de um simulador funcional da arquitetura RISC-V, especificamente o conjunto de instruções base de inteiros de 32 bits (RV32I). O simulador foi desenvolvido em Java, utilizando conceitos de Orientação a Objetos para modularizar os componentes de hardware (CPU, Memória, Barramento, ALU) e a biblioteca Java Swing para a interface gráfica interativa.

O sistema implementa o ciclo de busca e execução (Fetch-Decode-Execute), mapeamento de memória para VRAM/IO e suporte a interrupções de hardware.

2. ARQUITETURA DO SISTEMA

O simulador foi estruturado em módulos independentes que se comunicam através de interfaces bem definidas, replicando o comportamento físico de um computador.

2.1. Estrutura de Classes

- **CPU (Unidade Central de Processamento):** Responsável pelo controle do fluxo de execução. Mantém o Program Counter (PC), decodifica as instruções binárias de 32 bits e orquestra a operação entre ALU e Registradores.

- **ALU (Unidade Lógica e Aritmética):** Realiza operações matemáticas (soma, subtração) e lógicas (AND, OR, XOR, Shifts) puras, sem efeitos colaterais no estado do sistema.

- **Registers (Banco de Registradores):** Simula os 32 registradores de propósito geral (x0-x31), garantindo que o registrador x0 permaneça constante em zero.

- **Memory (Memória Principal):** Uma abstração de armazenamento linear endereçável a byte (Little Endian).

- **Bus (Barramento do Sistema):** Interconecta a CPU à Memória e aos dispositivos de E/S. É responsável pelo Mapeamento de Memória, roteando leituras e escritas para a RAM, VRAM ou Periféricos baseados no endereço.

- **SimulatorGUI (Interface Gráfica):** Camada de visualização que permite observar o estado interno da máquina (registradores, VRAM e PC) em tempo real..

3. DETALHES DE IMPLEMENTAÇÃO

3.1. CONJUNTO DE INSTRUÇÕES E CICLO DE EXECUÇÃO

A CPU implementa o suporte completo ao conjunto de instruções base RV32I, incluindo:

- **Aritmética/Lógica:** ADD, SUB, SLT, AND, OR, XOR, SLL, SRL, SRA (e suas variações imediatas).
- **Acesso à Memória:** Suporte a Bytes, Half-words e Words (LB, LH, LW, LBU, LHU, SB, SH, SW).
- **Controle de Fluxo:** Desvios condicionais (BEQ, BNE, BLT, BGE, etc.) e incondicionais (JAL, JALR).
- **Manipulação de Imediatos:** Extração correta de imediatos dos tipos I, S, B, U e J, incluindo extensão de sinal conforme especificação RISC-V.

O ciclo de instrução segue os passos:

1. **Fetch:** Busca a instrução no endereço apontado pelo PC via Barramento.
2. **Decode:** Extrai opcode, funct3, funct7, registradores e imediatos.
3. **Execute:** Aciona a ALU ou realiza transferências de dados.

3.2. Mapeamento de Memória e Barramento

O Barramento (Bus.java) implementa a lógica de roteamento de endereços conforme especificado:

Faixa de Endereço (Hex)	Dispositivo	Descrição Técnica
0x00000 - 0x7FFFF	**RAM Principal**	Área destinada ao armazenamento de instruções (o programa) e dados (variáveis, pilha). O acesso é direto à matriz de memória.
0x80000 - 0x8FFFF	**VRAM (Vídeo)**	Memória de Vídeo mapeada. Escritas nesta região são armazenadas na memória principal, mas também interpretadas pela Interface Gráfica como caracteres a serem exibidos no "Monitor".
0x90000 - 0x9FBFF	**Reservado**	Área reservada para expansão futura (acessos aqui podem ser tratados como falha ou ignorados).
0x9FC00 - 0x9FFFF	**E/S Mapeada**	Mapeamento de Periféricos (MMIO). Escritas nesta região são interceptadas e enviadas para a saída padrão (`System.out`), simulando uma porta serial ou terminal de texto.

Nota: O acesso à memória é seguro, com verificações de limites para evitar falhas de segmentação.

3.3. E/S Programada e VRAM

Para atender ao requisito de E/S programada, a CPU monitora o ciclo de execução. A cada **5 instruções** (configurável), o conteúdo da VRAM é varrido e atualizado na Interface Gráfica e no Terminal, simulando a taxa de atualização de um monitor

4. FUNCIONALIDADES EXTRAS E IMPLEMENTAÇÃO

Visando a pontuação extra e uma melhor experiência de uso, foram adicionados:

4.1. Interface Gráfica (GUI)

Desenvolvida em Java Swing, a interface permite:
Visualização tabular dos 32 registradores (Hexadecimal e Decimal).
Monitoramento do Program Counter (PC).
Tela de VRAM simulada (Console visual).
Controles de execução: "Step" (passo-a-passo) e "Fast Run" (execução em lote).

4.2. Interrupção de Hardware

Foi implementado um mecanismo de verificação de interrupções (checkInterrupt) no início de cada ciclo de clock. O sistema monitora a entrada padrão (System.in) e, ao detectar uma tecla pressionada, sinaliza uma Interrupção de Hardware no console, simulando o comportamento de um controlador de interrupções externo.

5. TESTES REALIZADOS E VALIDAÇÕES

Para validar o funcionamento integrado de todos os módulos (CPU, ALU, Barramento e Memória), foi utilizado um programa de teste escrito em linguagem de máquina e carregado na inicialização do sistema.
O Programa de Teste executa as seguintes ações:

1. **Carga de Imediatos (LUI):** Carrega o endereço base da VRAM (0x80000) no registrador x1.

2. **Operações Aritméticas (ADDI):** Calcula os valores ASCII dos caracteres 'R', 'I', 'S', 'C', '-', 'V'.

3. **Acesso à Memória (SW):** Armazena os valores calculados nos endereços de memória correspondentes à VRAM.

4. **Verificação:** O sucesso do teste é confirmado visualmente quando a string "RISC-V" aparece na tela da Interface Gráfica, provando que o fluxo de dados passou corretamente por: Decodificação -> ALU -> Barramento -> Mapeamento VRAM -> GUI.

6. INSTRUÇÕES DE COMPILAÇÃO E EXECUÇÃO

O projeto não possui dependências externas complexas e pode ser executado em qualquer ambiente com Java (JDK 8+).

Passos:

1. Assegure-se de que os arquivos **.java (CPU.java, Bus.java, Memory.java, ALU.java, Registers.java, SimulatorGUI.java, RiscVSimulator.java)** estão no mesmo diretório.

2. **Compile o projeto via terminal**: javac *.java

3. **Execute o simulador**: java RiscVSimulator.

7. REFERÊNCIAS BIBLIOGRÁFICAS

1. RISC-V International. The RISC-V Instruction Set Manual, Volume I: Unprivileged ISA. Document Version 20191213.

2. Patterson, David A.; Waterman, Andrew. The RISC-V Reader: An Open Architecture Atlas. Strawberry Canyon LLC, 2017.

3. Oracle. Java Platform, Standard Edition 8 API Specification. Disponível em: https://docs.oracle.com/javase/8/docs/api/. Acesso em: 22/09/2025.