Universidade Católica de Santos
Ciências da Computação

Professor: Walter S. Oliveira
walter.oliveira@unisantos.com
Disciplina: Organização de Computadores

Simulador de Computador RISC-V

Este documento contém as instruções para o desenvolvimento do projeto final da disciplina de Organização de Computadores, um **Simulador de Computador RISC-V**. O RISC-V [1, 2] é uma arquitetura de conjunto de instruções (ISA - *Instruction Set Architecture*) de padrão aberto, baseada nos princípios de computadores com conjunto de instruções reduzido (RISC - *Reduced Instruction Set Computer*). Diferente da maioria das outras arquiteturas (como x86 e ARM), o RISC-V é um **padrão livre e de código aberto**, o que significa que qualquer pessoa pode usá-lo, projetar, fabricar e vender processadores RISC-V sem **pagar taxas de licenciamento**.

Deste modo, será implementado simulador de um computador com as seguintes características:

(a) CPU que implemente a ISA RV32I

(b) Memória RAM/VRAM

(c) Periféricos de entrada e saída

A figura 1 ilustra o diagrama de blocos do simulador.

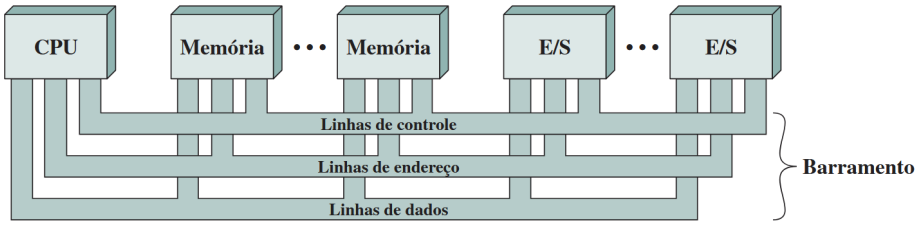

Figura 1: Diagrama de blocos das partes a serem implementadas do simulador

Pode-se utilizar **qualquer stack** para o desenvolvimento do simulador. Sugestão: utilizem uma linguagem fortemente tipada e/ou uma tecnologia nova que queiram aprender. Sugestões de linguagens para o núcleo principal do simulador: C++, Rust, Golang, Python com Ctypes, entre outros.

A seguir, os requisitos de cada um dos blocos a serem avaliados no simulador:

(1) CPU - 3 pontos

`TL;DR: Dada uma instrução do RV32i contida na memória, a CPU deve executar a operação e armazenar o valor no registrador/memória.`

A CPU do simulador deve dar suporte **a todas as instruções** e registradores do RV32I, ilustrado na figura 2, incluindo as instruções para operações aritméticas, lógicas, de desvio e carregamento/armazenamento, aceitando interrupções.

Instruções da Base de Números Inteiros: RV32I e RV64				
Categoria	Nome	Fmt	RV32I Base	+RV64I
Shifts	Shift Left Logical	R	SLL rd,rs1,rs2	SLLW rd,rs1,rs2
	Shift Left Log. Imm.	I	SLLI rd,rs1,shamt	SLLIW rd,rs1,shamt
	Shift Right Logical	R	SRL rd,rs1,rs2	SRLW rd,rs1,rs2
	Shift Right Log. Imm.	I	SRLI rd,rs1,shamt	SRLIW rd,rs1,shamt
	Shift Right Arithmetic	R	SRA rd,rs1,rs2	SRAW rd,rs1,rs2
	Shift Right Arith. Imm.	I	SRAI rd,rs1,shamt	SRAIW rd,rs1,shamt
Aritmética	ADD	R	ADD rd,rs1,rs2	ADDW rd,rs1,rs2
	ADD Immediate	I	ADDI rd,rs1,imm	ADDIW rd,rs1,imm
	SUBtract	R	SUB rd,rs1,rs2	SUBW rd,rs1,rs2
	Load Upper Imm	U	LUI rd,imm	
	Add Upper Imm to PC	U	AUIPC rd,imm	
Lógica	XOR	R	XOR rd,rs1,rs2	
	XOR Immediate	I	XORI rd,rs1,imm	
	OR	R	OR rd,rs1,rs2	
	OR Immediate	I	ORI rd,rs1,imm	
	AND	R	AND rd,rs1,rs2	
	AND Immediate	I	ANDI rd,rs1,imm	
Comparação	Set <	R	SLT rd,rs1,rs2	
	Set < Immediate	I	SLTI rd,rs1,imm	
	Set < Unsigned	R	SLTU rd,rs1,rs2	
	Set < Imm Unsigned	I	SLTIU rd,rs1,imm	
Desvios	Branch =	B	BEQ rs1,rs2,imm	
	Branch ≠	B	BNE rs1,rs2,imm	
	Branch <	B	BLT rs1,rs2,imm	
	Branch ≥	B	BGE rs1,rs2,imm	
	Branch < Unsigned	B	BLTU rs1,rs2,imm	
	Branch ≥ Unsigned	B	BGEU rs1,rs2,imm	
Salto & Link	J&L	J	JAL rd,imm	
	Jump & Link Register	I	JALR rd,rs1,imm	
Synch	Synch thread	I	FENCE	
	Synch Instr & Data	I	FENCE.I	
Environment	CALL	I	ECALL	
	BREAK	I	EBREAK	

Figura 2: ISA RISC-V RV32I

As instruções possuem um formato fixo, ilustrado na figura 3.

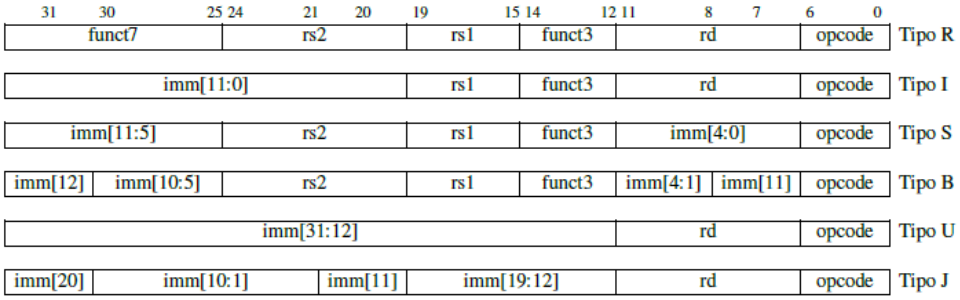

Figura 3: Formatos de instrução RV32I

Para isso, a CPU utiliza 32 registradores, conforme ilustrado na figura 4.

Name	Register Number	Use
zero	x0	Constant value 0
ra	x1	Return address
sp	x2	Stack pointer
gp	x3	Global pointer
tp	x4	Thread pointer
t0–2	x5–7	Temporary registers
s0/fp	x8	Saved register / Frame pointer
s1	x9	Saved register
a0–1	x10–11	Function arguments / Return values
a2–7	x12–17	Function arguments
s2–11	x18–27	Saved registers
t3-6	x28–31	Temporary registers

Figura 4: Os registradores do RV32I

A CPU deve implementar o ciclo de busca e execução passo-a-passo. Para isso, deve ser implementado todos os módulos constituintes de um processador de ciclo único, como na figura 5.

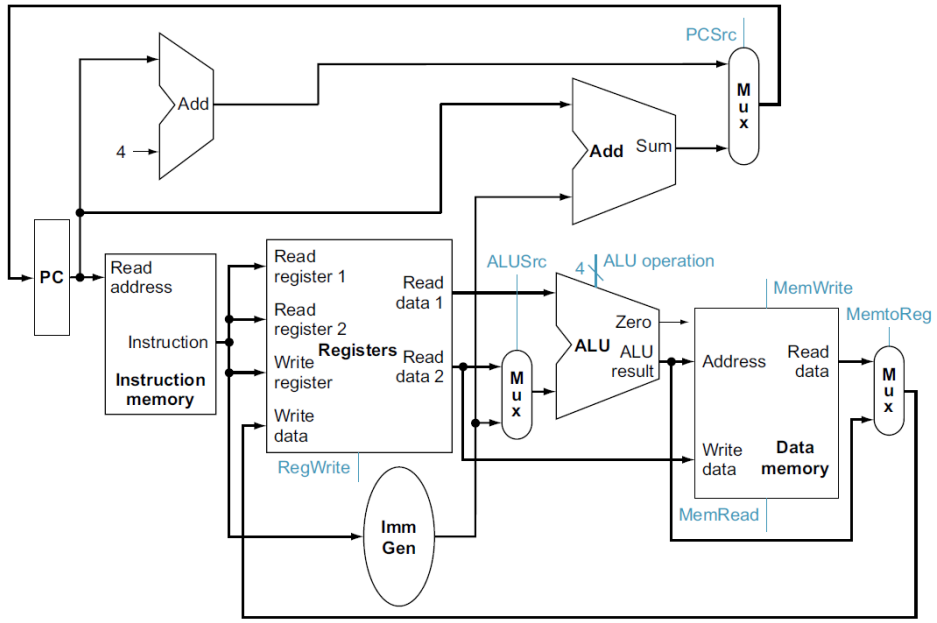

Figura 5: Processador de ciclo único RISC-V

Portanto, deve-se seguir as seguintes etapas:

1.) O contador de programa aponta para um endereço contido na memória principal

2.) O conteúdo (em binário/hexadecimal) é enviado para o processador

3.) O processador identifica qual instrução que deve ser executada

4.) Executa a instrução

(2) Barramento - 1 pontos

`TL;DR: Controle e conexão entre a CPU, Memória RAM e E/S.`

O barramento do computador deve realizar o controle dos blocos para gerenciar as interrupções e o fluxo de instruções/dados do sistema e dos dispositivos de entrada e saída.

Ele deve ser dividido em 3 partes:

1.) Barramento de dados: 32 bits, deve permitir o fluxo de entrada/saída dos dados entre os blocos

2.) Barramento de endereços: 32 bits, ajusta as posições de memória para leitura/escrita da informação contida no barramento de dados

3.) Barramento de controle: ajusta a permissão de leitura/escrita dos blocos

(3) Memória RAM - 2 pontos

`TL;DR: Matriz com 32 colunas em que cada setor corresponde a um trecho específico.`

A memória RAM deve seguir o seguinte mapeamento de memória:

Faixa de Endereço (Hex)	Descrição
0x00000 - 0x7FFFF	RAM Principal (Programa, Dados, Pilha, Heap)
0x80000 - 0x8FFFF	VRAM (Vídeo RAM)
0x90000 - 0x9FBFF	Área Reservada para Expansão Futura
0x9FC00 - 0x9FFFF	Periféricos de Hardware (E/S Mapeada)

Sendo assim, quando passado um endereço de memória pelo barramento, a RAM deve mandar no barramento de dados o valor correspondente ou realizar a escrita do valor contido no barramento no endereço correto.

(4) Utilização de E/S Programada - 2 pontos

`TL;DR: O trecho da memória VRAM deve ser transmitido via terminal quando a cada` n `instruções executadas.`

O setor de memória responsável pela VRAM deve ser carregado de informações, utilizando as instruções do de acesso a memória (`store word- sw`) de modo que, após a execução de uma quantidade finita de instruções (a definir pelo grupo), deve-se mostrar no terminal os caracteres ASCII correspondentes ao conteúdo presente na VRAM.

(5) Programa de teste - 2 pontos

`TL;DR: Conjunto de instruções para exemplificar funcionamento do simulador.`

Por fim, os alunos devem desenvolver um programa em RISC-V que utilize todas as funcionalidades do simulador, como por exemplo um algoritmo simples. Indica-se utilizar o Compile Explorer (https://godbolt.org/) para traduzir um programa em alto nível para as instruções RV32I equivalentes.

Método Avaliativo

1.) O projeto deve sem desenvolvido em grupo de, no máximo dois alunos.

2.) Semanalmente, será avaliado o desempenho do grupo pelo professor, gerando notas parciais. A nota final será composta da média das notas parciais em conjunto com o projeto final entregue

3.) A entrega do projeto será realizada com uma apresentação de demonstração do funcionamento do sistema (em vídeo ou presencialmente), assim como um arquivo de documentação detalhado e os códigos implementados em um repositório público, assim como todas as etapas necessárias para implementar o projeto (bibliotecas, pacotes, etc).

4.) Cada item presente neste documento pode possuir nota parcial ou total, de acordo com o cumprimento dos pré-requisitos apresentados

Sugere-se o seguinte cronograma para realização do projeto em 6 semanas:

- **Semana 1**: organização do grupo, definição de tecnologias e esboço inicial
- **Semana 2**: implementação da CPU
- **Semana 3**: implementação da CPU
- **Semana 4**: implementação da memória RAM
- **Semana 5**: implementação do barramento
- **Semana 6**: implementação do barramento com interrupção programada

Pontuação extra

Os seguintes itens contarão como pontuação extra para o projeto. Qualquer nota adicional nos itens abaixo serão incluídos no conceito obtido nos itens citados anteriormente e, caso o somatório seja superior a 10, na avaliação individual da disciplina.

① **Memória Cache - 2 pontos**

Deve ser implementado uma memória cache com **256 linhas** entre a CPU e o barramento de sistema, conforme ilustra a figura 6.

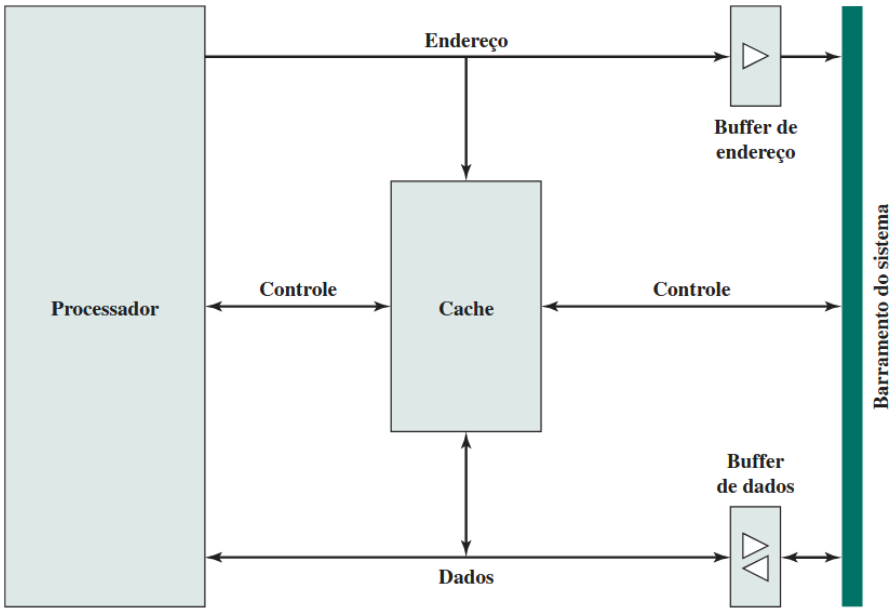

Figura 6: Organização com memória cache

Portanto, deve-se também definir um módulo de controle de memória (MMU), que deve gerenciar o preenchimento da cache dada uma política de substituição e o tipo de mapeamento do bloco da memória principal para a cache.

(2) Interrupção externa via teclado - 1 pontos

Adicionar uma verificação de interrupção caso uma tecla específica seja acionada, como ilustrado na figura 7. Portanto, um trecho pré-definido da memória RAM deve possuir as instruções a serem executadas caso ocorra interrupção e, assim que tratada, devolver o fluxo de execução ao programa principal.

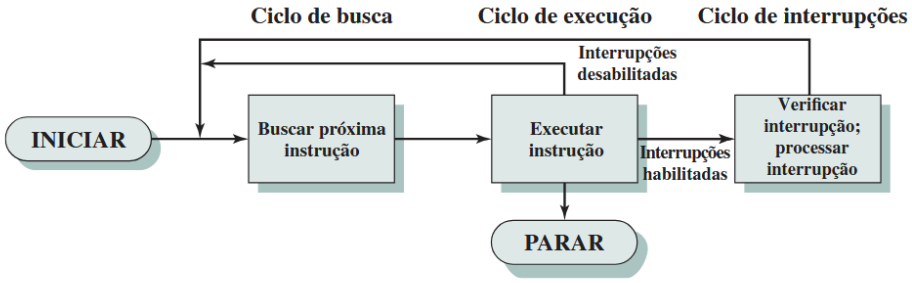

Figura 7: Ciclo de busca do processador com interrupção

(3) Interface gráfica/interatividade - 2 pontos

Grupos que realizarem alguma maneira iterativa de ilustrar o funcionamento do simulador ou de interagir com o simulador (por exemplo, alterando posições de memória, atualizando instruções carregadas no processador) poderão ganhar até dois pontos adicionais

(4) Performance - 2 pontos

Dado um programa benchmark que será disponibilizado pelo professor, o simulador que executar o mesmo:

1.) de maneira mais rápida

2.) com menos cache misses (caso tenham implementado a cache)

Ganharão até dois pontos adicionais

(5) Processador Pipeline - 3 pontos

Grupos que alterarem a microarquitetura do processador de ciclo único para pipeline e, demonstrarem um ganho de velocidade na execução das instruções, ganharão até 3 pontos adicionais.

Referências Bibliográficas

[1] David Patterson e Andrew Waterman. *Guia Prático RISC-V: Atlas de uma Arquitetura aberta*. Strawberry Canyon LLC, 2019.

[2] RISC-V International. https://riscv.org/. [Acesso 13-07-2025].