Universidade Federal de Santa Maria
CT
UFSM00274

Relatório Final Cleops – Sofware POO

Aluno: Bruno Henrique Spies
Professor: Osmar Marchi dos Santos

novembro
2023

UFSM
CT
UFSM00274

Relatório Final Cleops – Sofware POO

Segundo Relatório da Disciplina de Programação Orientada a Objetos do Curso de Engenharia de Computação da Universidade Federal de Santa Maria, como requisito parcial para aprovação.

Aluno: Bruno Henrique Spies

Matrícula: 202220187

E-mail: bruno.spies@ecomp.ufsm.br

Data de realização: 18/11/2023

Professor: Osmar Marchi dos Santos

novembro
2023

Sumário

1 Objetivos

O objetivo principal deste relatório é descrever o plano e a abordagem para o desenvolvimento do *backend* de um simulador de um processador "Cleópatra"denominado "CleoSim"em linguagem Java. O objetivo final do projeto é a construção de uma interface gráfica amigável e intuitiva também em linguagem Java. O projeto visa atender às necessidades de alunos e entusiastas de hardware, fornecendo uma ferramenta de simulação de alto desempenho que permitirá a análise e experimentação com o funcionamento do processador Cleópatra de forma eficaz e visualmente atrativa, facilitando o entendimento dos conceitos de um processador básico.

Para alcançar esse objetivo, foram realizadas as seguintes etapas:

- Pesquisa e Compreensão do Processador Cleópatra: Inicialmente, foi realizada uma pesquisa abrangente para compreender completamente o funcionamento do processador Cleópatra, seus componentes, conjunto de instruções e características técnicas. Isso nos permitirá criar uma base sólida para o simulador.

- Desenvolvimento da organização do Simulador: Com base na pesquisa, projetamos a organização que comporte a arquitetura do processador em questão, definindo a estrutura dos componentes necessários para emular o Cleópatra. Isso inclui a modelagem dos registradores, unidade de controle, memória e outros elementos fundamentais.

- Desenvolvimento da Lógica de Simulação: Foi criada uma lógica de simulação que permite aos usuários carregar programas, executar instruções, monitorar o estado do processador e analisar os resultados e salvar arquivos em assembly que podem ser executados e editados quando o usuário desejar. Além disso, a simulação deve ser precisa e eficiente.

- Implementação da Interface Gráfica: Uma das partes essenciais deste projeto foi a criação de uma interface gráfica amigável e intuitiva. Com foco no design e na implementação da GUI, garantindo que os usuários possam interagir facilmente com o simulador.

- Testes e Depuração: Foram realizados vários testes do simulador para garantir que ele funcione corretamente e reproduza com precisão o comportamento do processador Cleópatra. Quaisquer erros ou problemas serão identificados e corrigidos.

- Documentação: Foi criada uma documentação abrangente para o simulador, incluindo um manual de usuário detalhado, explicando como usar a interface gráfica e aproveitar todas as funcionalidades do simulador.

- Entrega e Avaliação: Após a conclusão do projeto, o projeto do simulador será entregue para avaliação por parte do professor e posteriormente apresentado para os colegas de classe. (Etapa Final).

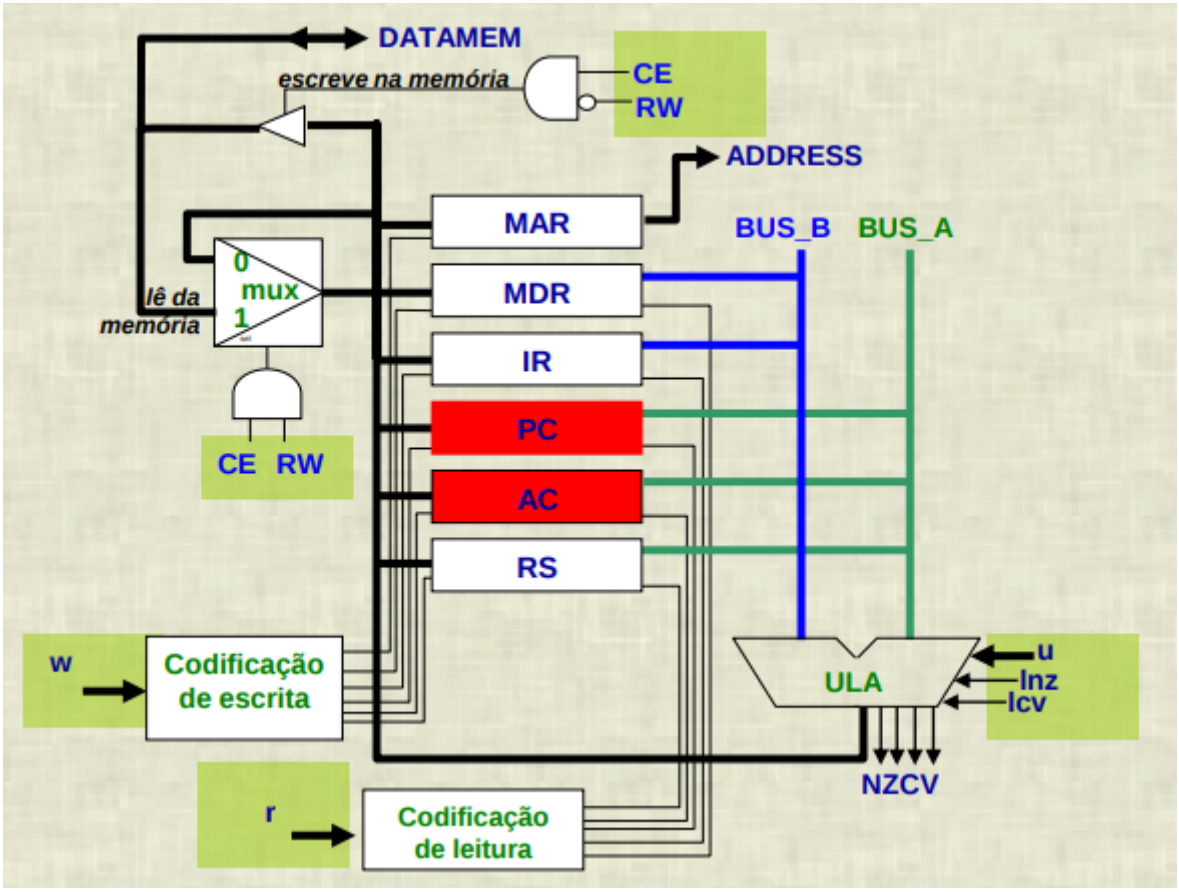
Este relatório servirá como um guia de referência para o desenvolvimento do simulador do processador Cleópatra com interface gráfica. Espero que o resultado final atenda às expectativas e contribua para o avanço no entendimento e na análise deste processador.

2 Introdução

O Cleópatra é um processador didático baseado em acumulador criado por Ney Calazans (PUCRS) e Fernando Moraes (PUCRS) de apenas 8 bits, ele endereça 2^8 posições ou seja 256 bytes de memória. Os conceitos desenvolvidos no Cleópatra são baseados no modelo de Von Neumann e podem ser observados em processadores comerciais como por exemplo 8051 (Intel) e 68HC11 (Motorola). Ele possui apenas 14 instruções e 4 modos de endereçamento. Apesar disso sua arquitetura é considerada CISC (*complex instruction set*) [AMORY A.; MORENO 2010].

A organização do processador que será utilizada para implementação possui 6 registradores de 8 bits e 4 *flipflops* que guardam os *flags* da ULA (Unidade Lógica e Aritmética). Além disso, uma memória de 256 posições. A versão do processador utilizada está ilustrada abaixo na Figura 1.

Figura 1: Organização utilizada do processádor Cleópatra



Fonte: Alexandre Amory e Edson Moreno, PUCRS

A Figura 2 possui o conjunto de instruções deste processador, o objetivo do software em desenvolvimento é reconhecer essas instruções digitadas pelo usuário na interface gráfica, gerando os códigos equivalentes a cada uma delas que serão gravados na memória. Após isso, o processador deve conseguir executar as instruções manipulando a memória e realizando as respectivas operações utilizando a ULA.

Figura 2: Conjunto de Instruções do processádor Cleópatra

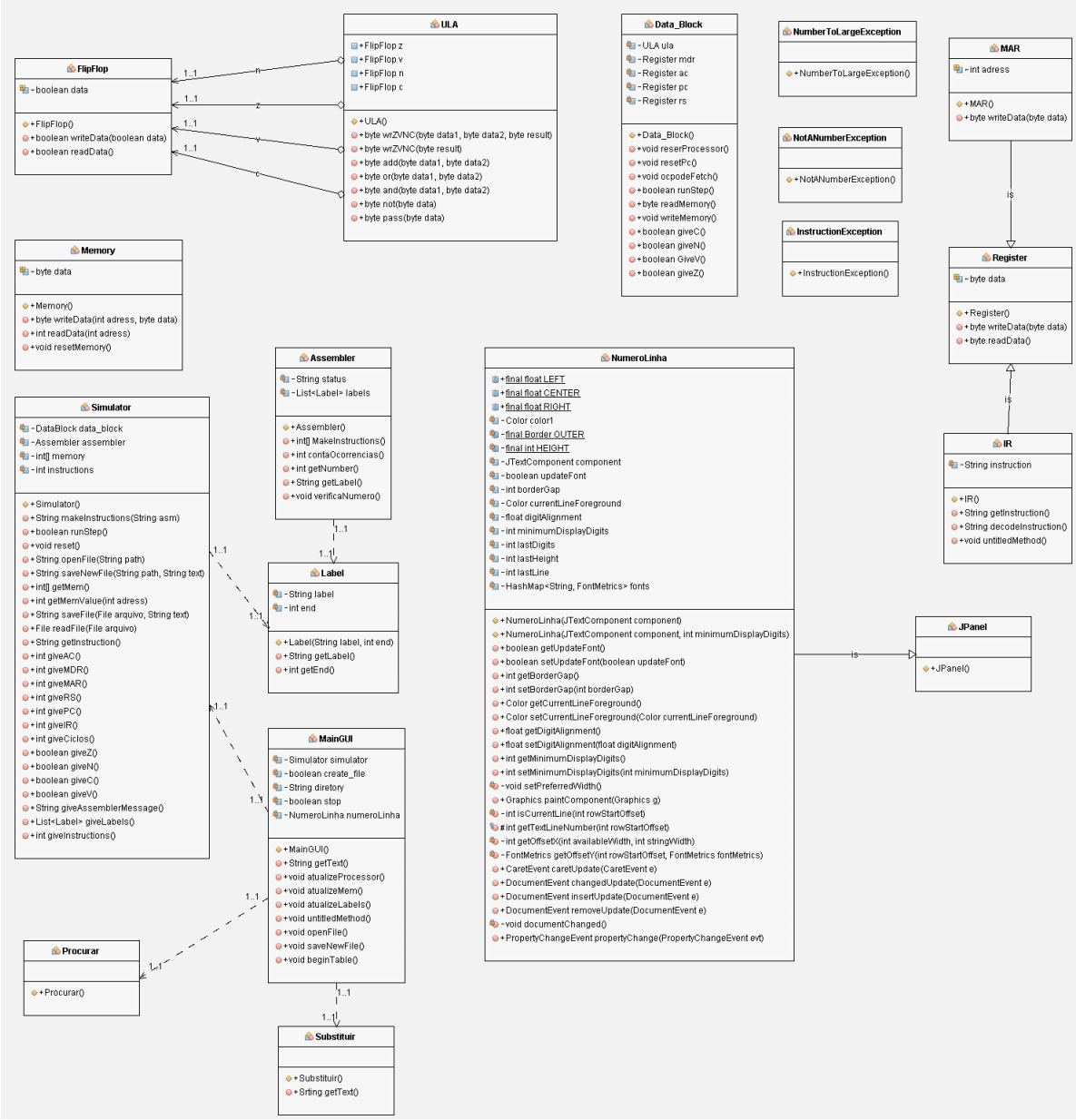
Mnemônico	Operação
NOT	Complementa (inverte) todos os bits de AC.
STA oper	Armazena AC na memória dada por oper.
LDA oper	Carrega AC com conteúdos de memória da posição dada por oper.
ADD oper	Adiciona AC ao conteúdo da memória dada por oper.
OR oper	Realiza OU lógico do AC com conteúdo da memória dada por oper.
AND oper	Realiza E lógico do AC com conteúdo da memória dada por oper.
JMP oper	PC recebe dado especificado por oper (desvio incondicional).
JC oper	Se C=1, então PC recebe valor dado por oper (desvio condicional).
JV oper	Se V=1, então PC recebe valor dado por oper (desvio condicional).
JN oper	Se N=1 então PC recebe valor dado por oper (desvio condicional).
JZ oper	Se Z=1, então PC recebe valor dado por oper (desvio condicional).
JSR oper	RS recebe conteúdo de PC e PC recebe dado de oper (subrotina).
RTS	PC recebe conteúdos de RS (retorno de subrotina).
HLT	Suspende processo de busca e execução de instruções.

Fonte: Alexandre Amory e Edson Moreno, PUCRS

3 Descrição das atividades realizadas

As atividades desempenhadas durante o periodo de confecção do trabalho se resumem na implementação do diagrama de classes contido na figura 3.

Figura 3: Diagrama de Classes



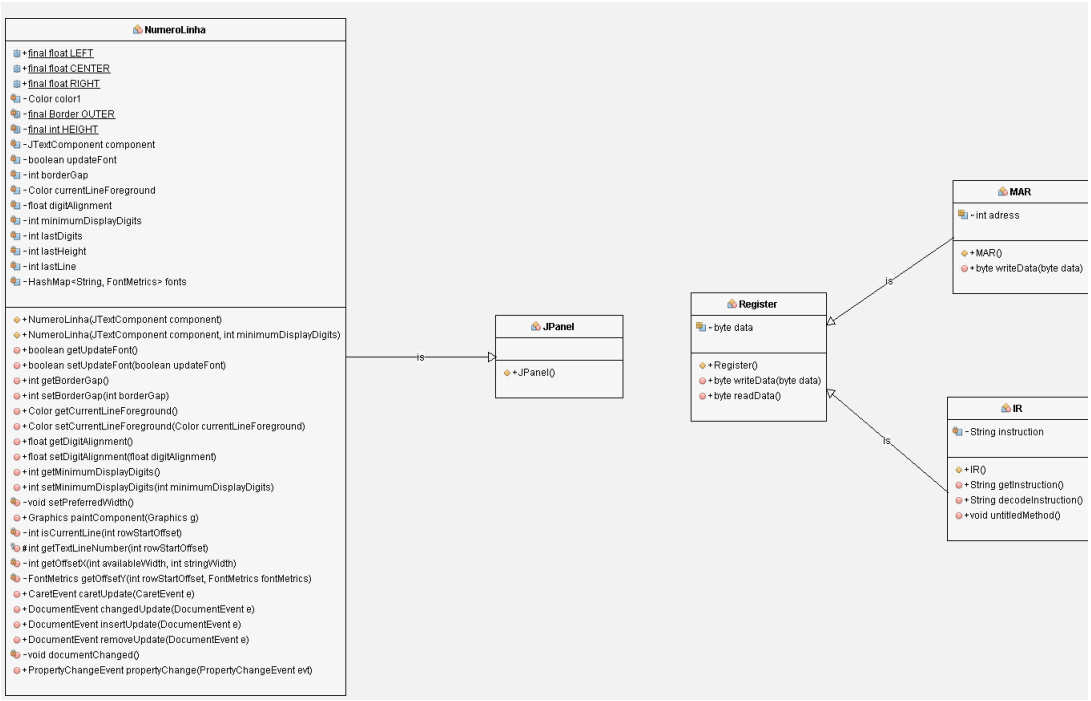
Fonte: Autor, 2023

3.1 Cumprimento dos Requisitos

3.1.1 Henrança

As três classes de herança implementadas estão contidas na Figura 4 abaixo.

Figura 4: Diagrama de Classes de Herança



Fonte: Autor, 2023

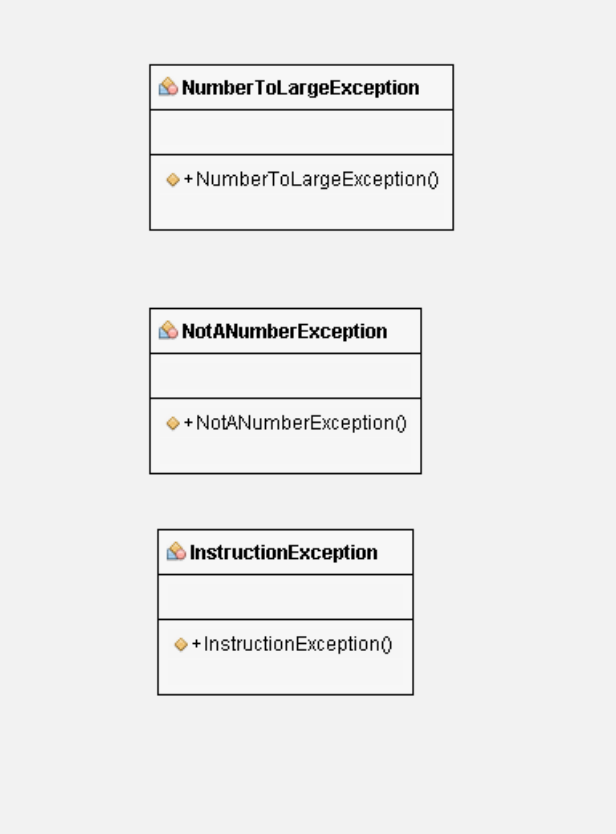
MAR e IR herdam a classe Register, MAR se especifica por possuir um atributo chamado adress do tipo inteiro e por sobrepor o método de *writeData()*. IR possuí um atributo chamado instruction do tipo String que guarda a instrução atual armazenada neste registrador, o método *decodeInstruction()* decodifica a instrução recebida pelo método *writeData()*.

NumeroLinha herda a classe JPanel e serve para numeral as linhas de um *TextArea* utilizado na Interface.

3.1.2 Tratamento de Exceções

As três classes de exceções implementadas estão contidas na Figura 5 abaixo.

Figura 5: Diagrama de Classes de Exceção



Fonte: Autor, 2023

NumberToLargeException gera uma mensagem de exceção de "O número digitado é inválido"e trata a exceção que ocorre quando algum valor maior que 255 é digitado na área de código.

NotANumberException gera uma mensagem de exceção de "Caracter não numérico!"e trata a exceção que ocorre quando algum valor não reconhecido como número é digitado na área de código.

InstructionException gera uma mensagem de exceção de "Instrução não Reconhecida"e trata a exceção que ocorre quando alguma instrução da memória não é reconhecida.

3.1.3 Uso da Coleção

Foi criada uma coleção do tipo *List<Label>* na classe Assembler, a mesa armazena o nome do Label em formato *String* e o endereço do mesmo em formato Inteiro. Após a montagem todos os Labels são exibidos em uma tabela.

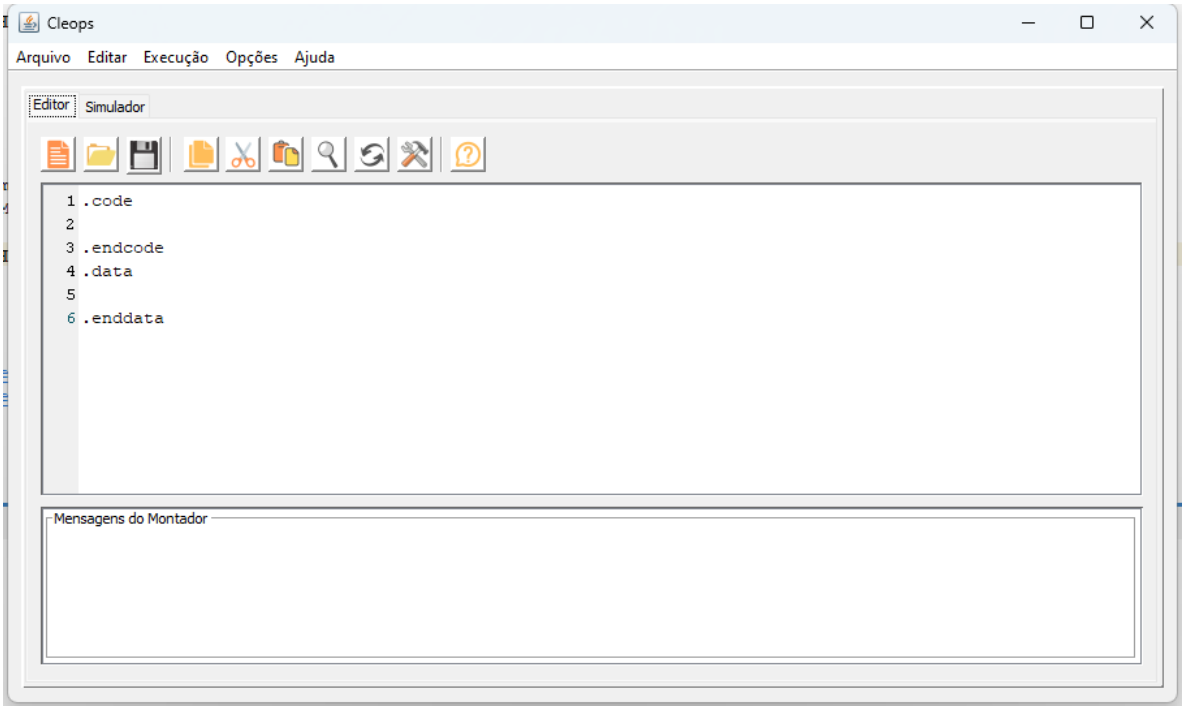
3.1.4 Manipulação de Arquivos

A classe *simulator* possuí os métodos de *saveNewFile* e *readFile* que criam e leem respectivamente um arquivo .asm através de uma janela criada com a classe *JFileChooser*.

3.1.5 Interface Gráfica

Foi implementada uma interface gráfica atrativa e funcional utilizando a biblioteca Swing e o NetBeans.

Figura 6: Interface

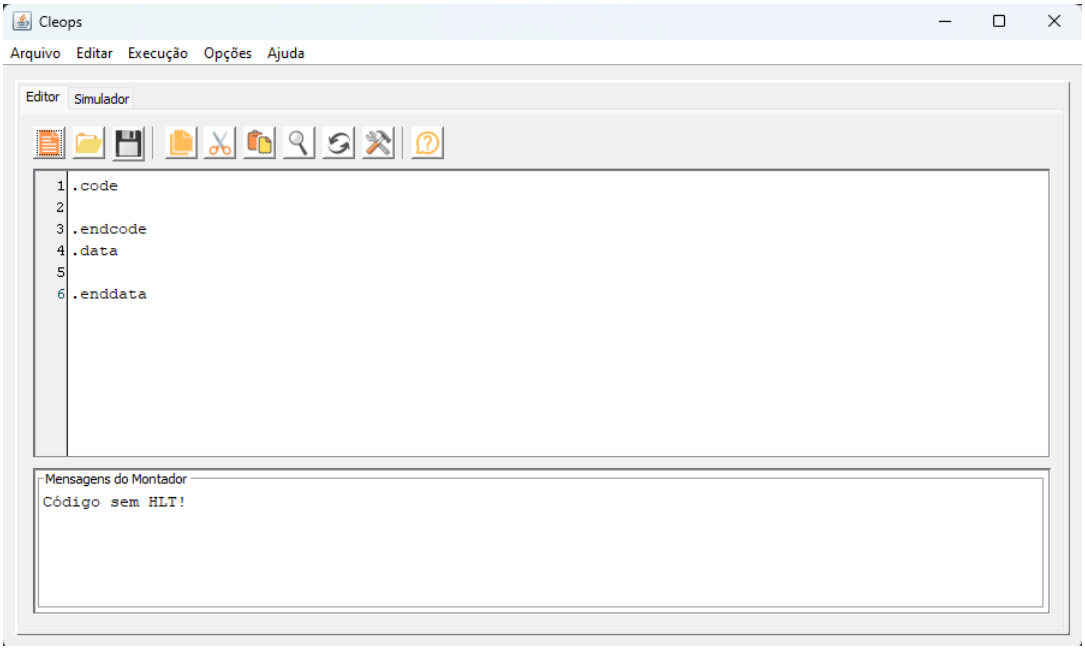


Fonte: Autor, 2023

4 Apresentação dos resultados obtidos

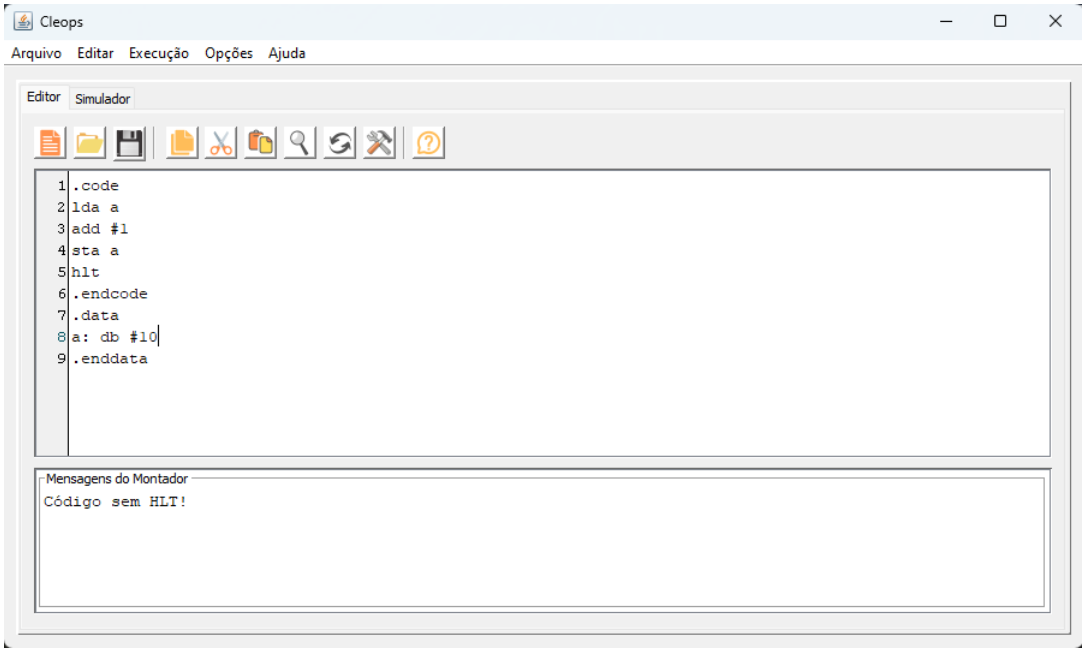
Como apresentação dos resultados obtidos vou demostrar o processo de impelmentação, salvamento e execução de um código em assembly.

Figura 7: Inicio



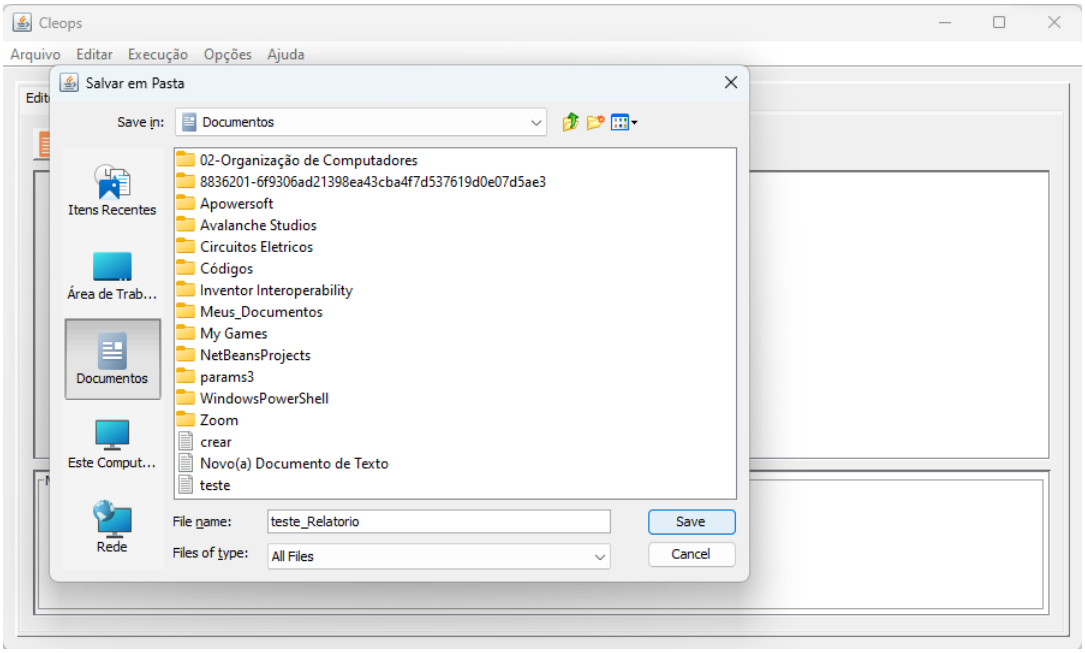
Fonte: Autor, 2023

Figura 8: Programando em assembly



Fonte: Autor, 2023

Figura 9: Salvando arquivo "teste_Relatorio.asm"

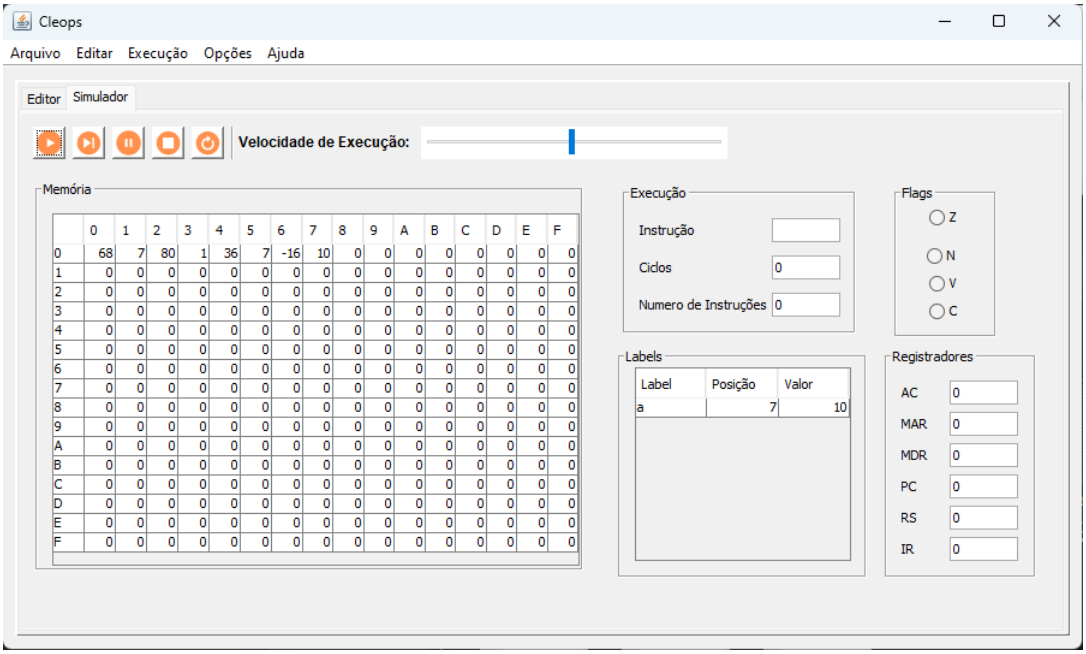


Fonte: Autor, 2023

Figura 10: Montando

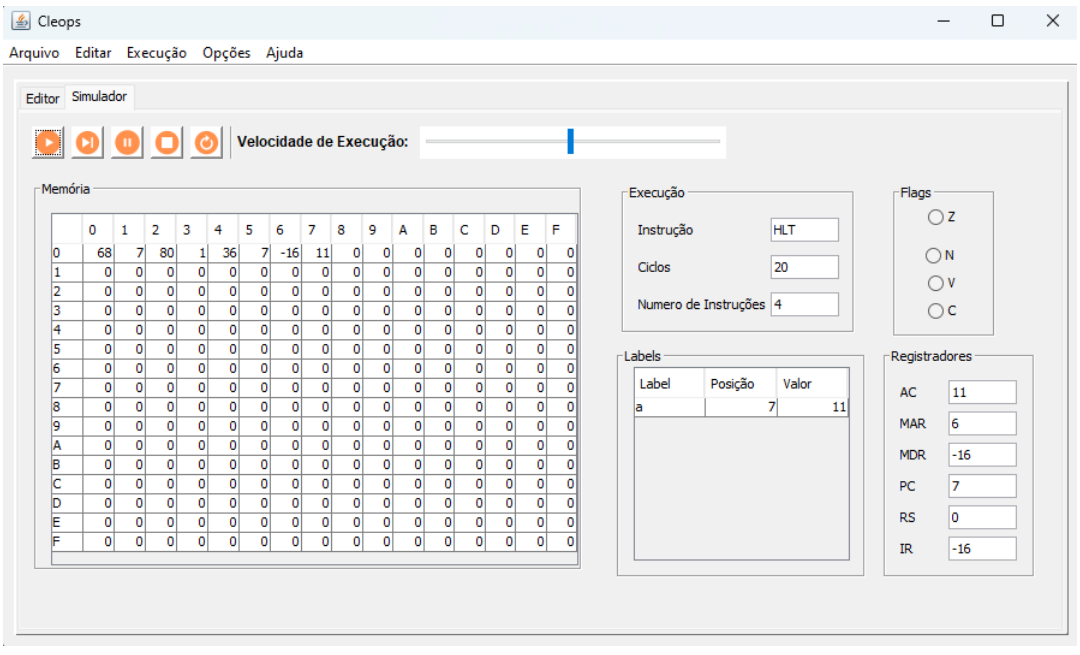
Fonte: Autor, 2023

Figura 11: Antes da Simulação



Fonte: Autor, 2023

Figura 12: Após a Simulação



Fonte: Autor, 2023

5 Discussão, considerações finais e conclusão

Neste relatório, foi apresentado o processo de desenvolvimento de software de um simulador do processador Cleópatra. Os passos seguidos foram:

- Pesquisa e Compreensão do Processador Cleópatra: Realizamos uma pesquisa detalhada para entender a arquitetura e o funcionamento interno do processador Cleópatra. Isso proporcionou uma base sólida para o desenvolvimento do simulador.
- Desenvolvimento da Arquitetura do Simulador: Foi criado um esquema claro para a estrutura do simulador, delineando os componentes-chave necessários para emular com precisão o Cleópatra.
- Implementação do Backend: A etapa de desenvolvimento do backend do simulador foi bem-sucedida, e agora tem-se um ambiente funcional para a simulação do processador. Isso permite carregar programas na memória, executar instruções e monitorar o estado interno do Cleópatra.
- Implementação da Interface Gráfica: Uma das partes essenciais deste projeto foi a criação de uma interface gráfica amigável e intuitiva. Com foco no design e na implementação da GUI, garantindo que os usuários possam interagir facilmente com o simulador.

Em conclusão, o desenvolvimento do software simulador do processador Cleopatra foi um sucesso. Todos os objetivos propostos foram alcançados, resultando em um simulador funcional e eficiente.

Referências

AMORY A.; MORENO, E. *Computador Cleópatra*: Organização de computadores. Porto Alegre: PUCRS, 2010.