Relatório Técnico do Simulador MIC-1

Igor Ribeiro, Lucas Lyra, Matheus Pereira, Pedro Ávila e Pedro Passos

10 de julho de 2025

Conteúdo

1 Introdução

O presente documento descreve a arquitetura e a implementação técnica de um simulador para a arquitetura MIC-1, um microprocessador didático projetado por Andrew S. Tanenbaum. O simulador foi desenvolvido em Python, utilizando a biblioteca `tkinter` para a sua interface gráfica. O objetivo principal do projeto é fornecer uma ferramenta educacional interativa que permita aos estudantes de arquitetura de computadores visualizar o fluxo de dados (datapath), a execução de microinstruções e a operação dos componentes internos de uma CPU baseada em microprograma.

A MIC-1 é uma arquitetura de 16 bits com um microprograma que controla o datapath. A sua simplicidade a torna ideal para fins de ensino, abstraindo complexidades de processadores comerciais enquanto mantém os conceitos fundamentais de operação de uma CPU. Este simulador implementa fielmente essa arquitetura, adicionando uma camada de visualização gráfica que enriquece a experiência de aprendizado.

O software é composto por um montador (assembler) que traduz código assembly para o formato binário da MIC-1, uma simulação detalhada da CPU e seus componentes, e uma interface gráfica rica que exibe registradores, memória, sinais de controle e o datapath em tempo real.

2 Arquitetura do Software

O projeto é modular, dividido em múltiplos arquivos Python, cada um com uma responsabilidade clara, promovendo a manutenibilidade e a clareza do código.

- `simulador.py`: É o orquestrador principal da aplicação. Ele inicializa a CPU, o montador e a interface gráfica. É responsável por conectar os eventos da UI (cliques de botão) às funções lógicas do simulador, como carregar um arquivo, montar o código, executar micro ou macroinstruções e resetar o sistema.

- `cpu.py`: Simula a Unidade Central de Processamento (CPU). Esta classe contém os principais componentes do datapath, como a ULA, o Shifter, e os registradores. Ela gerencia a execução dos subciclos de clock e o fluxo de dados entre os componentes a cada passo.

- `control_unit.py`: Implementa a Unidade de Controle microprogramada. Contém a memória de controle (microstore), o Contador de Microprograma (MPC), o Registrador de Microinstrução (MIR) e a lógica para decodificar o MIR e gerar os 32 sinais de controle que governam o datapath.

- `memory.py`: Simula a memória principal (RAM) de 4096 palavras de 16 bits. Implementa a lógica de leitura e escrita, incluindo um contador de ciclos para simular o tempo de acesso à memória, uma característica importante da arquitetura MIC-1.

- `hardware_components.py`: Define as classes para os componentes de hardware fundamentais, como Registradores (`Register`), a Unidade Lógica e Aritmética (`ALU`), o Deslocador (`Shifter`), Multiplexadores (`MUX`) e a Lógica de Sequenciamento Mestre (`MSL`).

- `assembler.py`: Contém a classe `Assembler`, responsável por traduzir o código-fonte em assembly MIC-1 para o código de máquina binário que pode ser carregado na memória do simulador. Ele valida as instruções e seus operandos.

- `interface.py`: Um "UI Kit"completo que define todos os elementos visuais da aplicação. Ele encapsula a complexidade do `tkinter`, fornecendo widgets personalizados, um tema visual coeso e o layout principal da aplicação, incluindo a visualização do datapath.

- `helpers.py`: Um módulo utilitário que contém funções auxiliares, como a conversão de valores para inteiros de 16 bits com sinal (`to_short`), e dicionários que mapeiam códigos numéricos a nomes legíveis (e.g., nomes de registradores e operações da ULA).

3 Implementação dos Componentes Centrais

3.1 Unidade de Controle e Microprograma

A Unidade de Controle (`control_unit.py`) é o cérebro da CPU. Ela opera com base em um microprograma armazenado em uma memória de controle (uma lista de strings de 32 bits).

Ciclo de Execução da Microinstrução: A execução de uma microinstrução é dividida em quatro subciclos, implementados na função `advance_subcycle` da classe `CPU` e orquestrados pela `ControlUnit`.

1. **Subciclo 1:** A microinstrução apontada pelo MPC é buscada da memória de controle e carregada no MIR (`run_first_subcycle`).
2. **Subciclo 2:** O MPC é incrementado. Os sinais que determinam quais registradores serão colocados nos barramentos A e B são decodificados e ativados (`run_second_subcycle`).
3. **Subciclo 3:** Os sinais para a operação da ULA, do Shifter e o carregamento do MAR são ativados. A ULA e o Shifter executam suas operações.
4. **Subciclo 4:** O resultado do barramento C é escrito em um dos registradores. A lógica de desvio condicional (MSL e MMUX) calcula o endereço da próxima microinstrução, que pode ser 'MPC+1' ou um endereço de desvio. Os sinais de leitura/escrita da memória são ativados.

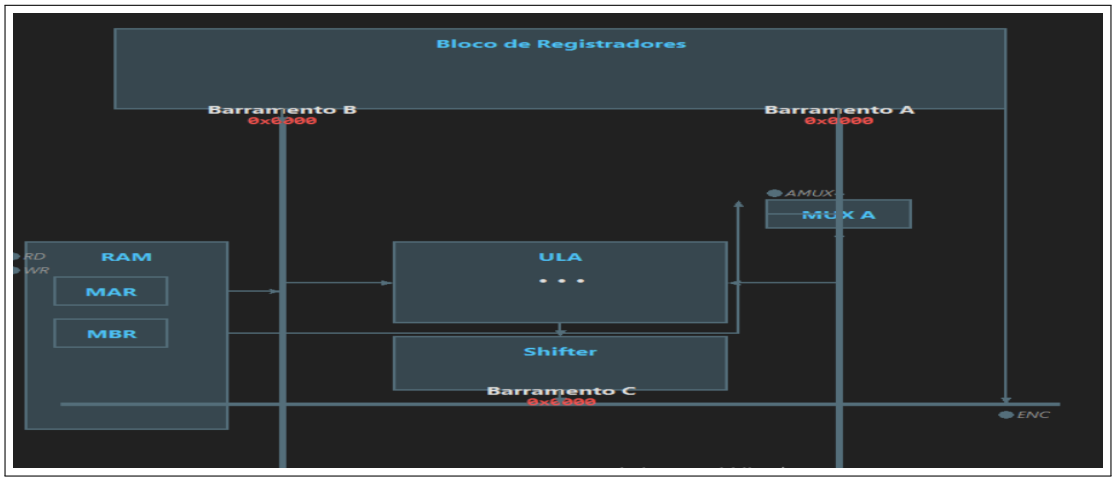
A classe `ControlUnit` gerencia o MPC e decodifica o MIR de 32 bits a cada ciclo para gerar os sinais de controle, como `amux_control`, `alu_control`, `rd_control`, etc.

3.2 Datapath e CPU

A classe `CPU` (`cpu.py`) instancia e conecta todos os componentes do datapath. A sua principal função, `execute_datapath`, é chamada a cada subciclo e executa as seguintes ações com base nos sinais de controle ativos:

1. Verifica se a memória está pronta para uma operação de leitura ou escrita.
2. Coloca os valores dos registradores de origem nos barramentos A e B.
3. O MUX de entrada da ULA (AMUX) seleciona entre o barramento A ou o MBR.
4. A ULA executa a operação definida pelos sinais `alu_control` e atualiza as flags N (negativo) Z (zero).
5. O Shifter opera sobre a saída da ULA.
6. O resultado do Shifter (barramento C) pode ser usado para carregar o MBR ou ser escrito de volta em um registrador de destino.
7. O valor do barramento B pode ser usado para carregar o MAR.

Figura 1: Representação conceitual do fluxo no datapath.



Esta figura é um placeholder. A UI real do simulador renderiza um datapath dinâmico e detalhado.

3.3 Montador (Assembler)

O montador (`assembler.py`) é uma peça crucial que torna o simulador utilizável. Ele converte código assembly em código de máquina de 16 bits.

Mapeamento de Opcodes: A classe `Assembler` utiliza um dicionário estático, `OPCODE_MAP`, para mapear mnemônicos de instruções para seus opcodes binários e para indicar se a instrução requer um operando.

```
OPCODE_MAP = {
    'LODD': ('0000', True), 'STOD': ('0001', True),
    'ADDD': ('0010', True), 'SUBD': ('0011', True),
    'JPOS': ('0100', True), 'JZER': ('0101', True),
    'JUMP': ('0110', True), 'LOCO': ('0111', True),
    # ... outras instrucoes
    'PSHI': ('1111000000000000', False),
    'RETN': ('1111100000000000', False),
    'INSP': ('11111100', True),
}
```

Listing 1: Exemplo do OPCODE_MAP no Assembler.

O processo de montagem percorre o código-fonte linha por linha, identifica o mnemônico, valida a presença e o tipo do operando (se necessário), e combina o opcode com o operando (formatado para o número correto de bits) para formar a instrução binária de 16 bits. O montador também realiza verificação de erros, como instruções desconhecidas ou operandos inválidos.

4 Interface Gráfica (UI)

A interface do usuário, definida em `interface.py`, é um dos pontos mais fortes do projeto. Ela foi projetada para ser profissional, informativa e reutilizável, seguindo boas práticas de desenvolvimento de UI.

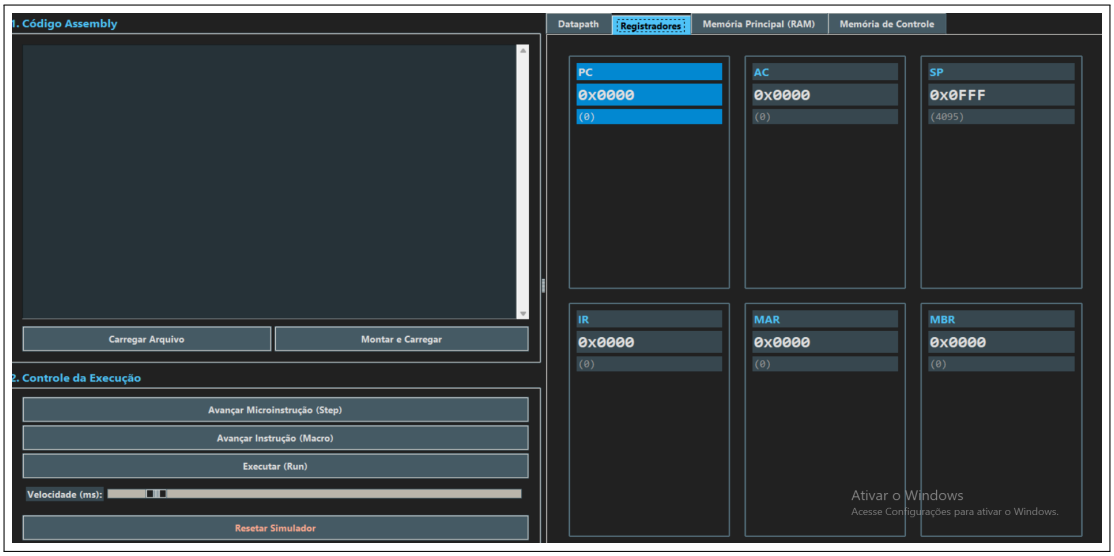
4.1 Estrutura e Tema

A UI é organizada em um painel esquerdo para controle e um conjunto de abas à direita para visualização.

- **Painel de Controle (`CodeControlPanel`):** Contém a área de texto para o código assembly, botões para carregar, montar, executar (passo a passo ou contínuo) e resetar, além de um controle de velocidade para a execução automática.
- **Abas de Visualização (`ttk.Notebook`):**
  1. **Datapath:** Um canvas dinâmico (`DatapathCanvas`) que desenha os componentes da CPU e as conexões. As linhas de barramento e os componentes mudam de cor para indicar atividade, fornecendo feedback visual imediato sobre o fluxo de dados e os sinais de controle.
  2. **Registradores:** Exibe os principais registradores da CPU (PC, AC, SP, etc.) usando um widget personalizado (`RegisterDisplay`). Os registradores são destacados com cores diferentes para indicar operações de leitura (fonte de dados para os barramentos) e escrita (destino do barramento C).
  3. **Memória Principal (RAM):** Mostra uma janela de visualização da memória, centralizada no ponteiro de instrução (PC). Endereços importantes como PC, MAR e SP são destacados com cores distintas para fácil identificação.
  4. **Memória de Controle:** Exibe todo o microprograma, destacando a microinstrução que está sendo executada (apontada pelo MPC).

A classe `Theme` centraliza toda a paleta de cores e fontes, facilitando a manutenção do estilo visual e garantindo uma aparência consistente e profissional em toda a aplicação.

Figura 2: Layout principal da Interface do Simulador.



A UI é construída com `tkinter` e widgets `ttk`, com um painel de controle à esquerda e abas de visualização à direita.

4.2 Atualização da UI

A função `update_all_displays` na classe `Mic1Simulator` é chamada após cada ação que altera o estado da CPU (e.g., `next_micro`, `next_macro`). Esta função orquestra a atualização de todos os componentes visuais:

- `datapath_canvas.update_datapath(cpu)`: Redesenha o datapath, atualizando as cores dos barramentos e dos sinais de controle com base no estado atual da `ControlUnit`.

- `_update_register_displays()`: Atualiza os valores (hexadecimal e decimal) de cada registrador e aplica os destaques de leitura/escrita.
- `_update_ram_display()`: Limpa e reescreve o conteúdo da área de texto da RAM, aplicando as tags de destaque para PC, MAR e SP.
- `_update_ucode_display()`: Atualiza a visualização da memória de controle, destacando a linha correspondente ao valor atual do MPC.

5 Fluxo de Execução e Uso

O fluxo de trabalho típico de um usuário com o simulador é o seguinte:

1. O usuário escreve ou carrega um arquivo de código assembly (`.txt`) na área de texto usando o botão "Carregar Arquivo".
2. Ao clicar em "Montar e Carregar", o `Assembler` é invocado. O código assembly é convertido em código de máquina e carregado na `Memory` simulada. O programa fonte original é mantido para exibição na visualização da RAM.
3. O usuário pode então executar o programa de três maneiras:
   - **Avançar Microinstrução (Step):** Executa um único subciclo de clock, permitindo uma análise extremamente detalhada do funcionamento interno da CPU. O estado de todos os componentes é atualizado após cada passo.
   - **Avançar Instrução (Macro):** Executa todos os subciclos necessários para completar uma única instrução assembly (do fetch ao fim da execução), parando quando o MPC retorna a 0.
   - **Executar (Run):** Inicia a execução automática do programa, chamando repetidamente a função `next_macro`. A velocidade da execução pode ser controlada por um slider. A execução para quando uma instrução `HALT` é encontrada ou se o PC aponta para fora da área do programa.
4. A qualquer momento, o botão "Resetar Simulador"pode ser usado para limpar a memória, resetar a CPU para seu estado inicial e limpar a área de código.

6 Conclusão

O Simulador MIC-1 descrito é uma ferramenta de software robusta e bem projetada que cumpre com sucesso seu objetivo educacional. A sua arquitetura modular, a simulação fiel dos componentes de hardware e, principalmente, a sua interface gráfica informativa e interativa, o tornam um recurso valioso para o ensino e aprendizado de arquitetura de computadores.

Os principais destaques técnicos do projeto são:

- **Separação de Responsabilidades:** A clara distinção entre a lógica do simulador (backend) e a sua apresentação (frontend) é um exemplo de boas práticas de engenharia de software.
- **Visualização Dinâmica:** A capacidade de visualizar o datapath em tempo real, com destaque para os barramentos ativos e sinais de controle, é o recurso pedagógico mais poderoso da ferramenta.
- **Detalhamento da Simulação:** A implementação correta dos quatro subciclos de clock por microinstrução permite uma análise profunda e precisa do funcionamento da MIC-1.

O projeto não apenas simula uma arquitetura de computador, mas também serve como um excelente exemplo de aplicação de princípios de design de software para criar uma ferramenta complexa, funcional e de alta qualidade.