Relatório Técnico: Emulador SAP-1 com Interface Gráfica e Animação

Autor: Marcus Meleiro
Disciplina: Arquitetura de Computadores I
Professor: Claudio
Data: 30/06/2025

1. Introdução

Este relatório descreve o desenvolvimento de um software aplicativo que emula o microcomputador didático SAP-1 (Simple-As-Possible 1). O projeto teve como objetivo principal não apenas simular a arquitetura e o conjunto de instruções do SAP-1, conforme detalhado no livro "Digital Computer Electronics" de Albert Paul Malvino (Capítulo 10), mas também fornecer uma interface gráfica intuitiva com animações dos blocos funcionais, um editor de código Assembly e um montador integrado.
Minha intenção ao criar este emulador foi proporcionar uma ferramenta visual para o estudo da arquitetura de computadores, permitindo que usuários escrevam e executem programas simples em Assembly, observando de forma clara e interativa o fluxo de dados e o estado interno da CPU em cada ciclo de instrução. A interface animada é fundamental para solidificar a compreensão dos conceitos de busca, decodificação e execução de instruções.

2. Arquitetura do SAP-1 (Malvino)

O SAP-1 é uma arquitetura de computador simplificada, de 8 bits, projetada para fins didáticos. Sua estrutura é baseada em um barramento único (o Barramento W), que conecta os principais componentes da CPU.
Os blocos funcionais essenciais do SAP-1, que foram emulados e visualizados na minha implementação, incluem (com referências ao Capítulo 10 do livro de Malvino):

- **Contador de Programa (PC)**: Um registrador de 4 bits que armazena o endereço da próxima instrução a ser buscada. (Seção 10.1)
- **Registrador de Endereço de Memória (MAR/REM)**: Um registrador de 4 bits que recebe o endereço do PC e o utiliza para acessar a memória RAM. (Seção 10.1)
- **Memória RAM (16x8)**: Uma memória de acesso aleatório de 16 bytes (cada byte com 8 bits), onde são armazenados tanto o programa quanto os dados. (Seção 10.1)
- **Registrador de Instruções (IR)**: Um registrador de 8 bits que armazena a instrução atualmente sendo executada, dividindo-a em opcode (4 bits MSB) e operando/endereço (4 bits LSB). (Seção 10.1)
- **Acumulador (ACC / Registrador A)**: Um registrador de 8 bits central para operações aritméticas e lógicas, armazenando resultados intermediários. (Seção 10.1)
- **Registrador B**: Um registrador auxiliar de 8 bits, usado para armazenar um operando em operações da ULA. (Seção 10.1)

- **Unidade Lógica Aritmética (ULA / Somador-Subtrator)**: Realiza operações de adição e subtração com os conteúdos do Acumulador e do Registrador B. (Seção 10.1)
- **Registrador de Saída**: Um registrador de 8 bits que recebe o resultado final do Acumulador para ser exibido. (Seção 10.1)
- **Indicador Visual em Binário (LEDs)**: Uma representação visual dos bits do Registrador de Saída. (Seção 10.1)
- **Barramento W**: O barramento de 8 bits que permite a transferência de dados entre a maioria dos componentes. (Fig. 10-1)
- **Controlador/Sequenciador (Unidade de Controle)**: Embora não seja um bloco animado diretamente, sua lógica é implementada no código para gerar os sinais de controle que orquestram o ciclo de busca-execução. (Seção 10.1, Seção 10.6)

O conjunto de instruções do SAP-1 é bastante reduzido, consistindo em cinco macroinstruções principais (Tabela 10-1 e Tabela 10-2):

- LDA <endereço> (Load Accumulator): Carrega um valor da memória para o ACC.
- ADD <endereço>: Soma um valor da memória ao ACC.
- SUB <endereço>: Subtrai um valor da memória do ACC.
- OUT: Transfere o conteúdo do ACC para o Registrador de Saída.
- HLT: Interrompe a execução do programa.

3. Implementação do Software

Desenvolvi o emulador utilizando **Python** pela sua simplicidade e agilidade no desenvolvimento, e a biblioteca **Tkinter** para a criação da interface gráfica. A estrutura do código foi modularizada para separar as responsabilidades de emulação da CPU, montagem do código Assembly e gerenciamento da interface/animações.

3.1. Visão Geral da Arquitetura do Software

O software é dividido logicamente em:

- **Módulo da CPU (Core)**: Contém a lógica interna do SAP-1, como registradores, memória e a ULA.
- **Módulo do Montador (Assembler)**: Responsável por traduzir o código Assembly para a linguagem de máquina.
- **Módulo da Interface Gráfica (GUI)**: Gerencia a interação do usuário, a exibição dos componentes da CPU e as animações.

3.2. Evolução da Interface de Entrada

A interface do emulador passou por uma evolução significativa em resposta à necessidade de tornar a interação mais amigável.

- **Primeira Versão**: Inicialmente, a única forma de inserir programas era através do **Editor Assembly** (área de texto), onde o usuário digitava manualmente todo o código em mnemônicos (e.g., LDA 0E, ADD 0F). Esta abordagem exigia um conhecimento prévio mais aprofundado da sintaxe e da arquitetura de memória do SAP-1, incluindo o manuseio de endereços para dados e instruções.

- **Segunda Versão (Atual)**: Inspirado por um exemplo prático apresentado em aula pelo **Professor Cláudio**, introduzi uma nova área de **"Entrada de Valores da Expressão"**. Esta funcionalidade permite ao usuário digitar expressões matemáticas simples (e.g., "5+3", "10-2") utilizando botões numéricos e de operação. Ao clicar em "Entrar", o emulador automaticamente processa a expressão, **gera o código Assembly** correspondente e o insere no editor principal. Essa mudança simplificou drasticamente a criação de programas para cálculos básicos, tornando o emulador mais acessível para usuários iniciantes.

Essa adição demonstra um avanço na usabilidade, transformando a entrada de dados de uma tarefa puramente baseada em Assembly para uma interação mais intuitiva, sem abrir mão da capacidade de programar diretamente via Assembly para casos mais complexos.

3.3. Módulo da CPU (Classe SAP1Emulator)

Dentro da classe principal SAP1Emulator, mantenho o estado da CPU em um dicionário self.cpu, que inclui:

- "PC": Contador de Programa.
- "ACC": Acumulador.
- "MAR": Registrador de Endereço de Memória.
- "IR": Registrador de Instruções.
- "B": Registrador B.
- "memory": Uma lista Python de 16 elementos representando a RAM.
- "output": Registrador de Saída.
- "flags": (Para futura expansão, caso o SAP-1 fosse estendido para incluir flags de Zero/Carry mais explícitas na ULA).

O método initialize_cpu() é responsável por resetar todos esses componentes para seus estados iniciais.

Exemplo de Ligação Código-Artigo (Inicialização da CPU):

A inicialização dos registradores e da memória no meu código Python reflete diretamente a arquitetura descrita na **Seção 10.1 do artigo de Malvino**, "Arquitetura". Nela, Malvino detalha os principais registradores como o Contador de Programa, Acumulador, Registrador de Endereço de Memória (REM), Registrador de Instruções e Registrador B, além da memória RAM de 16x8.

No código, isso é representado da seguinte forma:

```
  def initialize_cpu(self):
    """
    Inicializa o estado dos componentes da CPU SAP-1, definindo os valores iniciais
    para os registradores e limpando a memória.
    A referência para isso é a Seção 10.1 do artigo (Arquitetura).
    """
    self.cpu = {
      "PC": 0,     # Contador de Programa (4 bits) - Inicia em 0000 para a primeira
instrução.
      "ACC": 0,    # Acumulador (8 bits)
```

```
        "MAR": 0,     # Registrador de Endereço de Memória (4 bits)
        "IR": 0,      # Registrador de Instruções (8 bits)
        "B": 0,       # Registrador B (8 bits)
        "memory": [0] * MEMORY_SIZE,  # A memória é de 16 byte (16x8 RAM - Seção 10.1 do artigo)
        "output": 0,  # Registrador de Saída (8 bits)
        "flags": {"Z": 0, "C": 0}
    }
    # ... restante do código ...
```

Este trecho do meu código demonstra como os elementos teóricos do artigo são transpostos para a estrutura de dados do emulador, garantindo a fidelidade à arquitetura do SAP-1.

3.4. Módulo do Montador (Método assemble())

O método assemble() é o coração do meu montador de Assembly. Ele:

1. **Lê o Código Fonte**: Obtém o texto do editor Assembly.
2. **Processamento de Linha**: Itera sobre cada linha, ignorando linhas vazias e removendo comentários em linha (tudo após um ;). Este foi um ponto de atenção, pois comentários em linha eram indevidamente interpretados como operandos, gerando erros de montagem ("instrução OUT não aceita operando"). A correção envolveu uma etapa explícita de remoção de comentários antes da análise da linha.
3. **Processamento de Diretivas**:
    - ORG <endereço>: Define o endereço inicial na memória para as próximas diretivas DB ou instruções. (Seção 10.3)
    - DB <valor>: Define um byte de dado e o armazena no endereço atual da memória. (Seção 10.3)
4. **Tradução de Instruções**: Para cada instrução Assembly (LDA, ADD, SUB, OUT, HLT):
    - Identifica o mnemônico.
    - Converte o mnemônico para seu opcode binário correspondente (4 bits mais significativos). (Tabela 10-2)
    - Extrai o operando (endereço de 4 bits) para instruções que o requerem.
    - Combina o opcode e o operando em um único byte de 8 bits.
    - Armazena este byte na self.cpu['memory'] no instruction_ptr atual.
5. **Controle de Memória**: Um ponteiro (instruction_ptr ou data_ptr) garante que o código e os dados não excedam o limite de 16 bytes da memória do SAP-1. Um erro comum ("memória insuficiente") ocorria quando o último byte (0xF) era preenchido, devido a uma condição de verificação (>= MEMORY_SIZE) que impedia esse endereço. Ajustei a lógica para > MEMORY_SIZE - 1, permitindo o uso completo dos 16 bytes.
6. **Tratamento de Erros**: Qualquer erro de sintaxe, endereço fora do limite ou operando inválido é capturado e exibido ao usuário via messagebox e na barra de status, com destaque da linha no editor.

3.5. Ciclo Fetch-Execute (Método step())

O método step() simula um único ciclo de instrução do SAP-1, que é dividido em 6 estados de tempo (T-states): 3 para o ciclo de busca (Fetch) e 3 para o ciclo de execução. (Seção 10.4 e 10.5 do artigo).

1. **Animação do Clock**: Inicia cada estado T com um pulso de clock animado. (Fig. 10-2b)
2. **Ciclo de Busca (Fetch) - (Seção 10.4)**:
    - **T1 (Estado de Endereço)**: Conteúdo do PC é transferido para o MAR. (PC -> MAR). (Fig. 10-3a)
    - **T2 (Estado de Incremento)**: O PC é incrementado para apontar para a próxima instrução. (PC++). (Fig. 10-3b)
    - **T3 (Estado de Memória)**: A instrução no endereço do MAR é lida da memória e transferida para o IR. (Mem[MAR] -> IR). (Fig. 10-3c)
3. **Ciclo de Execução - (Seção 10.5)**:
    - A instrução no IR é decodificada (opcode e operando extraídos).
    - Com base no opcode, a função Python correspondente à instrução (ex: self.lda, self.add) é chamada.
    - As funções de instrução implementam os estados T4, T5 e T6 específicos para aquela macroinstrução, incluindo as transferências de dados relevantes.
    - A execução é interrompida se a instrução HLT for encontrada.

3.6. Implementação das Instruções (Métodos lda(), add(), sub(), out(), hlt())

Cada método de instrução detalha as micro-operações (transferências de dados e operações da ULA) que ocorrem nos estados T4, T5 e T6 para aquela instrução específica, sempre com referências às seções e figuras do artigo de Malvino. Por exemplo:

- **lda(operand) (Load Accumulator)**: Envolve IR (operando) -> MAR (T4) e Mem[MAR] -> ACC (T5). (Seção 10.5, Fig. 10-4)
- **add(operand) (Add)**: Mais complexa, envolve IR (operando) -> MAR (T4), Mem[MAR] -> Reg B (T5), e depois ACC + Reg B -> ULA -> ACC (T6). Uma melhoria visual implementada foi a exibição temporária do resultado da soma na ULA antes de ser transferido para o ACC. (Seção 10.5, Fig. 10-6)
- **sub(operand) (Subtract)**: Similar ao ADD, com a diferença de que a ULA realiza a subtração. Também exibe o resultado temporário na ULA. (Seção 10.5, similar a ADD, Fig. 10-6)
- **out() (Output)**: Apenas transfere o ACC para o Registrador de Saída. (Seção 10.5, Fig. 10-8)
- **hlt() (Halt)**: Sinaliza o fim da execução. (Seção 10.5)

3.7. Animação e Visualização (Métodos update_visualization(), animate_...(), highlight_component())

Esta foi uma parte crucial do projeto para tornar a simulação didática.

- **update_visualization()**: Este método é chamado após cada micro-operação para atualizar os valores exibidos nos registradores, na memória e nos LEDs de saída. Ele

também gerencia o destaque visual da célula de memória atualmente acessada pelo MAR.
- **animate_main_bus_transfer()**: Simula o fluxo de dados pelo Barramento W. Ele muda a cor do componente de origem, da linha de conexão ao barramento, do próprio barramento, da linha de conexão do destino e do componente de destino para um tom vibrante (vermelho claro), e depois os reseta para suas cores originais, criando um efeito de "onda de dados".
- **animate_direct_transfer()**: Usado para transferências que não envolvem o barramento principal, como os dados de ACC e Reg B indo para a ULA.
- **highlight_component()**: Simplesmente muda a cor de fundo e do texto de um componente por um breve período para indicar que ele está ativo ou sendo manipulado.
- **highlight_assembly_line()**: Uma melhoria de usabilidade, que destaca a linha de código Assembly que está sendo executada no editor, facilitando o acompanhamento do fluxo do programa.
- **Layout Visual**: O Canvas foi cuidadosamente projetado para refletir a disposição dos blocos funcionais do SAP-1 (Fig. 10-1). Os registradores e a ULA são maiores, e a memória RAM é exibida em uma grade 4x4 com endereços e valores claros. Linhas espessas representam o barramento, e pequenas bolinhas (LEDs) visualizam a saída binária.

3.8. Melhorias Visuais e de Usabilidade

Para aprimorar a experiência do usuário, implementei as seguintes melhorias visuais:
- **Tema Moderno**: Utilizei o tema clam do ttk.Style para conferir uma aparência mais plana e contemporânea à interface.
- **Estilo de Botões**: Personalizei os botões com bordas suaves e preenchimento, dando-lhes um visual mais limpo e responsivo.
- **Sombras Sutis**: Adicionei uma simulação de sombra aos blocos principais da CPU (registradores, ULA, memória) desenhando um retângulo levemente deslocado e mais escuro por trás de cada componente. Isso proporciona uma sensação de profundidade e um visual mais polido.
- **Organização dos Componentes**: Aumentei a área do canvas para acomodar componentes maiores e mais espaçados, com um layout que prioriza a clareza do fluxo de dados.
- **Legenda de Cores**: Incluí uma legenda no canto inferior do canvas que explica o significado das cores de destaque (componente ativo, memória acessada, LED ligado/desligado), facilitando a compreensão das animações.
- **Redimensionamento do Editor**: A caixa de texto do editor Assembly foi ajustada para um tamanho mais compacto, otimizando o espaço da janela.

4. Como Utilizar o Emulador

Para executar o emulador, siga estes passos:
1. **Pré-requisitos**: Tenha Python 3 instalado no seu sistema. É altamente recomendado

usar um ambiente virtual para instalar as dependências.
Na pasta do seu projeto:
python3 -m venv venv
source venv/bin/activate
pip install pyinstaller # PyInstaller já está instalado no seu venv

Obs: Caso encontre erros de python3.x-venv não encontrado, use sudo apt install python3.x-venv (substitua x pela sua versão de Python, ex: python3.12-venv).

2. **Clonar o Repositório**: Se ainda não o fez, clone o repositório do GitHub:
   git clone https://github.com/marcusmelleiro/emulador_sap.git
   cd emulador_sap

3. **Executar o Script**: Com o ambiente virtual ativo, execute o arquivo principal:
   python main.py

Uma vez que a janela do emulador esteja aberta:

- **Entrada de Expressão**: Digite uma expressão numérica (e.g., "5+3", "10-2") usando os botões e clique em "Entrar" para gerar o código Assembly automaticamente.
- **Carregar Exemplo**: Clique para preencher o editor com um programa Assembly de soma pré-definido.
- **Montar**: Compile o código Assembly para a memória da CPU.
- **Executar**: Inicia a simulação contínua do programa.
- **Passo a Passo**: Executa uma instrução por vez, ideal para depuração e observação detalhada.
- **Reset**: Limpa o estado da CPU para iniciar um novo programa.
- **Slider de Velocidade**: Ajusta a velocidade das animações e da execução.

5. Conclusão

O desenvolvimento deste emulador SAP-1 foi um desafio gratificante e uma experiência de aprendizado profunda em arquitetura de computadores e programação de interfaces gráficas. Acredito que a combinação de um emulador funcional, um montador integrado e uma visualização animada e aprimorada graficamente oferece uma ferramenta didática poderosa para estudantes interessados em entender o funcionamento interno de um processador simples. O projeto me permitiu aplicar conceitos teóricos de ciclos de máquina, registradores e lógica de controle em uma implementação prática e interativa.