Trabalho 1: Implementação de uma Máquina Virtual Chip-8

Prof. Dr. Juliano H. Foleis

1. Introdução

O que é Chip-8?

O Chip-8 é uma linguagem de programação interpretada, desenvolvida originalmente por Joseph Weisbecker nos anos 70 para ser usada em microcomputadores de 8 bits, como o COSMAC VIP e o Telmac 1800. Embora simples, o Chip-8 pode ser considerado uma espécie de "console de videogame virtual" ou uma máquina virtual simplificada. Sua especificação inclui um conjunto de instruções (opcodes), uma organização de memória, registradores, e mecanismos de entrada e saída.

Por sua simplicidade e arquitetura bem definida, a implementação de um interpretador Chip-8 tornou-se um rito de passagem popular para estudantes e entusiastas interessados em aprender sobre emulação, interpretação de bytecode e o funcionamento de máquinas virtuais.

Objetivo do Trabalho

O objetivo principal deste trabalho é compreender, na prática, os conceitos fundamentais por trás da implementação de uma máquina virtual. Ao desenvolver um interpretador para a plataforma Chip-8, vocês irão:

- Aprender sobre a arquitetura básica de um sistema computacional (memória, CPU, registradores).
- Entender o ciclo de busca, decodificação e execução de instruções (ciclo de fetch-decode-execute).
- Implementar uma CPU virtual com seu conjunto de instruções específico.
- Gerenciar memória, pilha (stack) e registradores.
- Lidar com periféricos virtuais, como tela, teclado e temporizadores.
- Utilizar a biblioteca SDL (Simple DirectMedia Layer) para renderização gráfica, manipulação de entrada (teclado) e áudio.

Ao final, vocês terão uma máquina virtual funcional, capaz de carregar e executar programas e jogos clássicos desenvolvidos para a plataforma Chip-8.

2. A Especificação Chip-8

Para construir a máquina virtual, é crucial seguir a especificação de *hardware* da plataforma. Abaixo estão detalhados os componentes que devem ser implementados.

Memória

- **Tamanho:** A VM Chip-8 possui 4KB (4096 bytes) de memória RAM.
- **Mapeamento:**
  - `0x000` a `0x1FF` (512 bytes): Reservado para o interpretador. Historicamente, continha todo o código do interpretador Chip-8! Nesta implementação, você pode deixar essa área vazia ou usá-la para armazenar os sprites dos dígitos hexadecimais (0-F), que são comumente pré-carregados aqui.
  - `0x200` a `0xFFF` (3584 bytes): Espaço disponível para o programa (ROM) a ser carregado e para o restante das variáveis do sistema. A maioria dos programas começa a ser carregada no endereço `0x200`.

Registradores

- **Registradores de Propósito Geral (V0 a VF):**
  - 16 registradores de 8 bits, denominados `V0`, `V1`, ..., `VE`, `VF`.
  - O registrador `VF` é de uso especial, servindo como uma *flag* para certas operações. Por exemplo, ele pode indicar um estouro ("carry") em uma soma ou uma colisão de pixels na tela. **Não o utilize para outros fins.**
- **Registrador de Endereço (I):**
  - Um registrador de 16 bits, chamado `I`, usado principalmente para armazenar endereços de memória.
- **Program Counter (PC):**
  - Um registrador de 16 bits, o `PC` (Contador de Programa), que aponta para a próxima instrução a ser executada na memória. Ele deve ser inicializado com o valor especificado na linha de comando (por padrão, `0x200`).
- **Stack Pointer (SP):**
  - Um registrador de 8 bits, o `SP` (Ponteiro de Pilha), que aponta para o topo da pilha.

Pilha

- A pilha é uma estrutura usada para armazenar os endereços de retorno de sub-rotinas.
- A especificação original permitia até 16 níveis de aninhamento. Portanto, uma pilha com capacidade para 16 endereços de 16 bits é suficiente.

Tela

- **Resolução:** A tela é monocromática, com uma resolução de 64x32 pixels.
- **Renderização:** Os gráficos são desenhados através de *sprites*. Um sprite é uma sequência de bytes, onde cada byte representa uma linha de 8 pixels. O bit mais à esquerda do byte corresponde ao pixel mais à esquerda na linha. Os sprites podem ter entre 1 e 15 bytes de altura.
- **Dígitos Hexadecimais:** Os sprites para os dígitos hexadecimais (0-F) devem ser pré-carregados na memória, começando no endereço `0x000`. Cada dígito é representado por um sprite de 5 bytes (5 linhas de 8 pixels). Abaixo estão os sprites para os dígitos 0-F:

```
0: 0xF0, 0x90, 0x90, 0x90, 0xF0  // 0
1: 0x20, 0x60, 0x20, 0x20, 0x70  // 1
2: 0xF0, 0x10, 0xF0, 0x80, 0xF0  // 2
3: 0xF0, 0x10, 0xF0, 0x10, 0xF0  // 3
4: 0x90, 0x90, 0xF0, 0x10, 0x10  // 4
5: 0xF0, 0x80, 0xF0, 0x10, 0xF0  // 5
6: 0xF0, 0x80, 0xF0, 0x90, 0xF0  // 6
7: 0xF0, 0x10, 0x20, 0x40, 0x40  // 7
8: 0xF0, 0x90, 0xF0, 0x90, 0xF0  // 8
9: 0xF0, 0x90, 0xF0, 0x10, 0xF0  // 9
A: 0xF0, 0x90, 0xF0, 0x90, 0x90  // A
B: 0xE0, 0x90, 0xE0, 0x90, 0xE0  // B
C: 0xF0, 0x80, 0x80, 0x80, 0xF0  // C
D: 0xE0, 0x90, 0x90, 0x90, 0xE0  // D
E: 0xF0, 0x80, 0xF0, 0x80, 0xF0  // E
F: 0xF0, 0x80, 0xF0, 0x80, 0x80  // F
```

Teclado

- O sistema de entrada é um teclado hexadecimal com 16 teclas:

  ```
  1 2 3 C
  4 5 6 D
  7 8 9 E
  A 0 B F
  ```

- A VM precisa ser capaz de detectar quando uma tecla está pressionada e quando é liberada.

- A implementação deve mapear essas teclas para as teclas do teclado físico do computador onde a VM está rodando. A escolha do mapeamento fica a critério do desenvolvedor, mas deve ser documentada no arquivo `README_USO.pdf`.
- O padrão de mapeamento sugerido é:

Teclado Chip-8	Teclado Físico
1	1
2	2
3	3
C	4
4	Q
5	W
6	E
D	R
7	A
8	S
9	D
E	F
A	Z
0	X
B	C
F	V

Temporizadores

Existem dois temporizadores, ambos de 8 bits, que decrementam a uma taxa de 60Hz quando seus valores são maiores que zero.

- **Delay Timer (DT):** Usado para eventos de tempo em jogos. Pode ser lido e escrito pelo programa.
- **Sound Timer (ST):** Quando seu vahttps://tonisagrista.com/blog/2021/chip8-speclor é maior que zero, um som (um "beep") deve ser produzido. O som continua enquanto `ST > 0`. O som é um tom puro, cuja frequência é à escolha do desenvolvedor. A forma de onda pode ser simples, como uma onda quadrada.

Conjunto de Instruções

Todas as instruções do Chip-8 têm 2 bytes (16 bits) de comprimento. Há 36 instruções diferentes na especificação original do Chip-8. As instruções representam manipulações de registradores, controle de fluxo, operações aritméticas, manipulação de memória, desenho na tela e controle de entrada. Cada instrução deve estar armazenada em posições de memória alinhadas a 2 bytes (endereços pares). As instruções são armazenadas em ordem *big-endian* (o byte mais significativo vem primeiro).

As instruções, bem como suas descrições e efeitos, podem ser encontradas em diversas fontes *online*:

- Cowgod's Chip-8 Technical Reference
- Guide to Making a CHIP-8 Emulator
- CHIP-8 virtual machine specification

Programas para Testes

Para testar seu interpretador, você pode usar vários programas disponíveis na internet. Alguns úteis são:

- Chip-8 splash screen
- Logo da IBM
- chip8-test-suite
- Chip-8 Games Pack

3. Regras e Requisitos

Funcionalidade Esperada

- A máquina virtual deve ser capaz de carregar e executar corretamente os programas (ROMs) Chip-8 disponíveis publicamente. ROMS de domínio público serão fornecidas para testes.
- A biblioteca SDL deve ser utilizada para gerenciar a janela, a renderização dos gráficos, a captura de eventos de teclado e a reprodução de som.
- O programa deve ser executado via linha de comando, sem a necessidade de uma interface gráfica (GUI) para carregar a ROM.
- A implementação deve ser feita em `C++`.
- O código deve ser modular, organizado e bem documentado.
- O programa deve incluir tratamento de erros, como tentativas de carregar arquivos inexistentes ou inválidos.
- A velocidade de execução da CPU deve ser configurável via linha de comando (ex: 500Hz, 1000Hz). Consideramos que cada ciclo de CPU corresponde a uma instrução executada. Portanto, uma CPU rodando a 500Hz executa 500 instruções por segundo.
- A escala da janela (fator de zoom) também deve ser configurável via linha de comando. Por exemplo, um fator de escala de 10 resultaria em uma janela de 640x320 pixels (64*10 x 32*10).
- O endereço de carga do programa deve ser configurável via linha de comando, com o padrão sendo `0x200`.
- A tela deve ser atualizada a uma taxa de 60Hz, independentemente da velocidade da CPU.
- Os temporizadores (Delay Timer e Sound Timer) devem decrementar a uma taxa de 60Hz.
- O programa deve ser capaz de lidar com a entrada do teclado de forma responsiva, permitindo que o usuário interaja com os jogos em tempo real.
- O som deve ser reproduzido corretamente quando o *Sound Timer* estiver ativo.
- O programa deve ser capaz de rodar em sistemas operacionais comuns, como Linux e Windows. Para isso, utilize apenas bibliotecas multiplataforma.
- O código deve ser compilável utilizando compiladores gratuitos (`g++` ou `clang++`). A compilação deve ser dirigida por meio de um sistema de compilação simples, preferencialmente `CMake` ou `Make`.
- O código deve ser portável e não deve depender de funcionalidades específicas de um sistema operacional. Em outras palavras, o mesmo código deve compilar e rodar em diferentes sistemas operacionais (Linux e Windows) sem modificações. A avaliação será realizada em um sistema Linux.
- Em caso de dúvidas sobre a especificação, consulte o professor.
- Seu interpretador deve ser capaz de executar corretamente o splash screen, logo da IBM, Maze, Tank e Pong.

Critérios de Avaliação

A avaliação do trabalho será baseada nos seguintes critérios:

- **Correção Funcional (80%):** A máquina virtual deve executar corretamente os programas Chip-8, seguindo a especificação detalhada. A funcionalidade de cada opcode deve ser implementada conforme descrito.
- **Qualidade do Código (20%):** O código deve ser bem estruturado, modular e documentado. A legibilidade e a organização do código serão consideradas.

Plágio e Uso de Código de Terceiros

Em caso de plágio, ambos os membros da dupla receberão nota zero. Plágio inclui, mas não se limita a:

- Cópia direta de código-fonte de terceiros sem a devida atribuição.
- Uso de código-fonte de terceiros sem permissão, mesmo que modificado.
- Compartilhamento de código entre duplas.

Só é permitido o uso de bibliotecas externas para funcionalidades específicas, como a SDL para gráficos e som. O uso de código-fonte terceiros para a implementação da máquina virtual não é permitido.

Caso haja dúvidas sobre o que constitui plágio, consulte o professor antes de iniciar o trabalho.

IA Generativa

O uso de ferramentas de IA generativa para geração de código é altamente desencorajado. Haverá uma avaliação sobre o funcionamento do código e das decisões de projeto, e perguntas específicas poderão ser feitas para verificar a compreensão do aluno sobre o código submetido. Esta avaliação será feita individualmente. A nota da avaliação individual será usada para ajustar a nota individual do aluno.

Entrega do Trabalho

O trabalho deve ser realizado em **duplas**. A entrega final deverá ser um arquivo compactado (`.zip` ou `.tar.gz`) contendo os seguintes itens:

1. **Código-Fonte:**
   - Todos os arquivos-fonte (`.cpp`, `.h`) do projeto.
   - O código deve ser bem organizado e comentado para facilitar a compreensão.
2. **Instruções de Compilação (`README_COMPILAR.pdf`):**
   - Um arquivo em formato PDF explicando passo a passo como compilar e gerar o executável do projeto.
   - Liste todas as dependências necessárias (ex: compilador C++, SDL2, etc.)
   - Inclua os comandos exatos a serem executados no terminal.
3. **Instruções de Uso (`README_USO.pdf`):**
   - Um arquivo em formato PDF explicando todas as opções de linha de comando disponíveis para executar a máquina virtual.
   - Deve detalhar, no mínimo, como carregar um programa (ROM).
   - Exemplo de como documentar as opções (em `markdown`):

````
# Como Usar o Emulador Chip-8

A sintaxe básica para executar o programa é:

```sh
./chip8_emulator [opções] caminho/para/a/rom.ch8
```

## Opções de Linha de Comando

`--clock <velocidade>`: Define a velocidade do clock da CPU em Hz. Padrão: 500Hz.

**Exemplo:** `./chip8_emulator --clock 700 roms/tetris.ch8`

`--scale <fator>`: Define o fator de escala da janela.
    Um fator de 10 resulta em uma janela de 640x320. Padrão: 10.

**Exemplo:** `./chip8_emulator --scale 15 roms/pong.ch8`

`--help`: Exibe esta mensagem de ajuda.
````

Bom trabalho!