CHIP-8

Descrição e Arquitetura

O **CHIP-8** é uma máquina virtual criada em 1977 por Joseph Weisbecker (RCA) para facilitar o desenvolvimento de jogos em sistemas de 8 bits.
Ele foi projetado para rodar em computadores como o RCA COSMAC VIP e outros modelos domésticos da época.

Seu conjunto de instruções é simples (35 opcodes), e o hardware simulado é composto por:

- **Memória**: 4 KB, com programas geralmente carregados a partir do endereço `0x200`.
- **Registradores**: 16 registradores de 8 bits (`V0` a `VF`, sendo `VF` usado como flag).
- **Registrador de índice**: `I` (16 bits) para endereçamento de memória.
- **Program Counter (PC)**: 16 bits, aponta para a próxima instrução.
- **Stack**: até 16 níveis, para chamadas de subrotinas.
- **Timers**: `delay_timer` e `sound_timer`, decrementados a 60 Hz.
- **Display**: matriz monocromática de 64×32 pixels.
- **Entrada**: teclado hexadecimal com 16 teclas (`0x0` a `0xF`).

Conjunto de intruções

Abaixo está a tabela com o conjunto de instruções reconhecidas pelo interpretador do CHIP-8:

Opcode	Instrução	Descrição	Tipo
`0NNN`	`SYS addr`	Chama sub-rotina em endereço `NNN`	Sistema
`00E0`	`CLS`	Limpa a tela	Tela
`00EE`	`RET`	Retorna da sub-rotina	Fluxo
`1NNN`	`JP addr`	Salta para o endereço `NNN`	Fluxo
`2NNN`	`CALL addr`	Chama sub-rotina em `NNN`	Fluxo
`3XKK`	`SE Vx, byte`	Pula a próxima instrução se `Vx == KK`	Condicional
`4XKK`	`SNE Vx, byte`	Pula a próxima instrução se `Vx != KK`	Condicional

Opcode	Instrução	Descrição	Tipo
5XY0	SE Vx, Vy	Pula a próxima instrução se Vx == Vy	Condicional
6XKK	LD Vx, byte	Atribui KK ao registrador Vx	Dados
7XKK	ADD Vx, byte	Soma KK a Vx, sem carry	Aritmética
8XY0	LD Vx, Vy	Atribui Vy a Vx	Dados
8XY1	OR Vx, Vy	Vx = Vx OR Vy	Lógica
8XY2	AND Vx, Vy	Vx = Vx AND Vy	Lógica
8XY3	XOR Vx, Vy	Vx = Vx XOR Vy	Lógica
8XY4	ADD Vx, Vy	Soma Vy a Vx, define VF = 1 se overflow, senão 0	Aritmética
8XY5	SUB Vx, Vy	Vx = Vx - Vy, VF = 0 se houve borrow, 1 caso contrário	Aritmética
8XY6	SHR Vx	Desloca Vx para a direita 1 bit, VF = bit0 original	Bitwise
8XY7	SUBN Vx, Vy	Vx = Vy - Vx, VF = 0 se houve borrow, 1 caso contrário	Aritmética
8XYE	SHL Vx	Desloca Vx para a esquerda 1 bit, VF = bit7 original	Bitwise
9XY0	SNE Vx, Vy	Pula a próxima instrução se Vx != Vy	Condicional
ANNN	LD I, addr	Atribui NNN ao registrador I	Dados
BNNN	JP V0, addr	Salta para NNN + V0	Fluxo
CXKK	RND Vx, byte	Vx = random_byte() & KK	Aleatório
DXYN	DRW Vx, Vy, N	Desenha sprite na tela em (Vx, Vy) com altura N, VF = colisão	Tela
EX9E	SKP Vx	Pula próxima instrução se tecla Vx estiver pressionada	Entrada
EXA1	SKNP Vx	Pula próxima instrução se tecla Vx **não** estiver pressionada	Entrada
FX07	LD Vx, DT	Atribui o valor do delay timer a Vx	Timer
FX0A	LD Vx, K	Espera por uma tecla pressionada, armazena em Vx	Entrada
FX15	LD DT, Vx	Define delay timer = Vx	Timer
FX18	LD ST, Vx	Define sound timer = Vx	Som

Opcode	Instrução	Descrição	Tipo
FX1E	ADD I, Vx	Soma Vx a I	Aritmética
FX29	LD F, Vx	Define I para o endereço do sprite do caractere em Vx	Fonte (Gráfico)
FX33	LD B, Vx	Armazena BCD de Vx em I , I+1 , I+2	Conversão
FX55	LD [I], Vx	Armazena V0 até Vx na memória a partir de I	Memória
FX65	LD Vx, [I]	Lê da memória a partir de I para V0 até Vx	Memória

Compilando e executando

Para a compilar e executar uma rom, execute os comandos abaixo

```
make
./play <rom>
```

Estrutura do processador

O processador está dividido em 4 módulos principais, cada um com responsabilidade bem definida:

Inicialização (init.c)

Responsável por preparar o estado inicial da máquina CHIP-8:

- Zera memória, registradores, pilha, display e teclado.
- Carrega o fontset padrão na memória (endereços começando em 0x00).
- Inicializa os registradores especiais: PC é definido para o início dos programas (0x200), I , SP , timers e sinalizadores.
- Semente do gerador de números aleatórios é inicializada com srand(time(NULL)) .

Variáveis globais definidas aqui e expostas via extern :

- opcode
- memory
- V (registradores V0..VF)
- I , PC
- gfx (tela)

- `delay_timer`, `sound_timer`
- `stack`, `SP`
- `key`
- `chip8_draw_flag`

Função principal:

```
void chip8_initialize(void);
```

Também contém o `chip8_fontset` e o helper `randbyte()` usado por instruções aleatórias.

Carregamento de ROM (load.c)

Responsável por abrir e ler a ROM do jogo para a memória em `0x200`:

Função:

```
void chip8_loadgame(char *game);
```

Se não conseguir abrir o arquivo, exibe erro e finaliza. Lê até `MAX_GAME_SIZE` bytes (tamanho máximo possível) em `memory[0x200]`.

Ciclo de Emulação e Timers (cycle.c)

Contém a lógica central de busca, decodificação e execução de opcodes, além da atualização dos timers:

Funções:

```
void chip8_emulatecycle(void);
void chip8_tick(void);
```

- `chip8_emulatecycle`: busca o opcode atual, decodifica campos (`x`, `y`, `n`, `kk`, `nnn`) e executa de acordo com a tabela completa de instruções do CHIP-8 (incluindo desenho de sprites, controle de fluxo, operações aritméticas, manipulação de registradores, eventos de tecla, etc.).
    - Usa `draw_sprite` para renderizar gráficos com detecção de colisão.
    - Atualiza o program counter (`PC`) conforme cada instrução.
    - Sinaliza desenho com `chip8_draw_flag` quando necessário.

- `chip8_tick` : decrementa os timers `delay_timer` e `sound_timer` . Emite "BEEP!" quando `sound_timer` chega a zero.

Arquivo de cabeçalho comum (chip8.h)

Define:

- Constantes (tamanhos de memória, display, pilha, etc.).
- Tipos e macros auxiliares.
- Declarações `extern` dos estados globais.
- Assinaturas das funções públicas:
  - `chip8_initialize`
  - `chip8_loadgame`
  - `chip8_emulatecycle`
  - `chip8_tick`
  - `draw_sprite` (usada internamente)

Frontend gráfico e main (main.c)

Implementa a interface de saída e entrada usando OpenGL/GLUT:

- Inicializa a janela com as dimensões baseadas em `GFX_ROWS` e `GFX_COLS` , escaladas por `PIXEL_SIZE` .
- Traduz o estado `gfx` para pixels e desenha via `glDrawPixels` .
- Mapeia teclado convencional para o teclado hexadecimal do CHIP-8.
- Loop principal chama `chip8_emulatecycle()` constantemente e atualiza timers em taxa fixa (`CLOCK_HZ`).
- Usa `chip8_draw_flag` para atualizar a tela apenas quando necessário.

Mapeamento de teclas

O teclado físico é mapeado para o teclado hexadecimal do CHIP-8 da seguinte forma:

```
1 2 3 4    -> 1 2 3 C
q w e r    -> 4 5 6 D
a s d f    -> 7 8 9 E
z x c v    -> A 0 B F
```

Pressionar a tecla correspondente define o índice em `key[]` para 1; liberá-la zera-o.

Temporização

- A emulação usa um timer de 60Hz (`CLOCK_HZ`) para atualizar os timers de delay e som.
- O ciclo de execução (`chip8_emulatecycle`) roda continuamente via `glutIdleFunc` .
- A função `chip8_tick` é chamada apenas quando o tempo acumulado ultrapassa o intervalo de 1/60 segundo.

Layout de memória importante

- Instruções são carregadas a partir de `0x200` .
- `chip8_fontset` está em `0x00` .
- Registrador de índice `I` é usado para endereçamento de fontes/sprites.
- `V[0xF]` é usado como flag de carry/collision em várias instruções.

Erros e tratamento

- OpCodes desconhecidos disparam mensagem de erro e encerram o programa.
- Falha ao abrir a ROM também encerra com mensagem.

OpenGL para Emular a Tela e Visualização Gráfica

Para a renderização da tela e interação visual, foi utilizada a biblioteca **OpenGL** em conjunto com a **GLUT** (OpenGL Utility Toolkit).
Essa escolha foi feita por três motivos principais:

1. **Portabilidade** – OpenGL/GLUT está disponível em múltiplas plataformas, permitindo que o emulador rode em diferentes sistemas operacionais sem alterações significativas no código.
2. **Controle direto de pixels** – Usar `glDrawPixels` possibilita escrever diretamente na área gráfica com base no estado do array `gfx` do CHIP-8, mantendo fidelidade ao funcionamento original.
3. **Integração simples com eventos de teclado e loop principal** – A GLUT oferece callbacks para eventos como pressionamento/soltura de teclas e funções de desenho, que se encaixam perfeitamente no ciclo de emulação.

O funcionamento conjunto é simples:

- O emulador atualiza o buffer `gfx` conforme a execução das instruções do CHIP-8.
- Quando há necessidade de redesenhar a tela, a função `draw()` traduz cada pixel lógico (64×32) em blocos maiores (`PIXEL_SIZE`) para visualização.
- Esse buffer expandido é enviado ao OpenGL, que o renderiza na janela criada pela GLUT, garantindo baixa latência e atualização suave.