ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

Departamento de Engenharia de Computação e Sistemas Digitais

PCS3732– Laboratório de Processadores

VirtualFuc*

Integrantes:	João Felipe de Souza Melo	13682913
	Lucas Suzin Bertan	13682548
	Ian Andrade de Magalhães	12551449

Introdução

O objetivo principal deste projeto foi aplicar os conceitos abordados na disciplina de Laboratório de Processadores (PCS3732), focando no desenvolvimento e na execução de software em uma arquitetura de processador específica. O projeto consiste na criação de um interpretador para a linguagem de programação Brainfuck, que executa dentro de um emulador de sistema, o QEMU, em uma arquitetura ARM.

O projeto foi estruturado para demonstrar a aplicação prática de conhecimentos sobre a arquitetura de microprocessadores e sistemas microprocessados, incluindo a manipulação de memória, a comunicação com periféricos (como a UART) e a criação de ferramentas de depuração. O interpretador não só executa programas em Brainfuck, mas também inclui um modo de depuração para análise do estado da execução em tempo real.

Fundamentos

Interpretador

Um interpretador é um programa de computador que executa diretamente instruções escritas em uma linguagem de programação, sem a necessidade de compilá-las previamente para código de máquina. Ele lê o código-fonte, traduz e executa cada instrução sequencialmente. Isso contrasta com um compilador, que traduz o código-fonte inteiro para um arquivo executável que será rodado posteriormente.

Brainfuck

Brainfuck é uma linguagem de programação esotérica, conhecida por sua extrema simplicidade e minimalismo. Ela é composta por apenas oito comandos, que manipulam um array de bytes, uma espécie de "fita de memória", e um ponteiro para essa fita.

Os comandos são:

- >: Incrementa o ponteiro da memória.
- <: Decrementa o ponteiro da memória.

- +: Incrementa o byte no qual o ponteiro está.
- -:Decrementa o byte no qual o ponteiro está.
- .: Imprime o byte no qual o ponteiro está como um caractere.
- ,: Lê um caractere e armazena seu valor no byte do ponteiro.
- [: Se o byte no ponteiro for zero, pula para o comando] correspondente.
-]: Se o byte no ponteiro não for zero, volta para o comando [correspondente.

Código para printar “Hello Word” no terminal:

```
++++++++++[>++++++++>++++++++++++>++
++++++++>++++>+++>++++++++>++++++++
++++>++++++++++>++++++++++>+++++++
++++>+++>+<<<<<<<<<<<<<-]>-.>--.>---
.>++++.>++.>---.>---.>.>.>+.>+++.>.
```

Arquitetura e Implementação

Estrutura Geral do Código

O projeto é composto por vários arquivos, cada um com uma função específica na construção do interpretador:

- startup.s: Arquivo de *assembly* que define o ponto de entrada (_start) do programa e salta para a função main em C.
- linker.ld: Define a organização da memória do programa. Ele especifica a seção
- .text (código), .data (dados inicializados) e .bss (dados não inicializados). O endereço de início do código é definido em 0x10000
- main.c: Contém a lógica principal do interpretador Brainfuck. É aqui que o laço de execução percorre o código do programa em Brainfuck e realiza a interpretação de cada comando.
- Makefile: O arquivo Makefile automatiza o processo de compilação. Ele usa o compilador arm-none-eabi-gcc para gerar o arquivo kernel.elf a partir dos

arquivos startup.s, main.c e linker.ld. Em seguida, usa arm-none-eabi-objcopy para converter o kernel.elf em kernel.bin.

O interpretador funciona como um laço de repetição que percorre o programa Brainfuck caractere por caractere. Uma static char memory[MAX_MEMORY_SIZE] atua como a fita de memória, e a variável pos_memoria atua como o ponteiro de memória.

O fluxo de execução é o seguinte:

1. A função main inicia o programa Brainfuck a ser interpretado.
2. Um for loop percorre a string do programa.
3. Dentro do loop, uma estrutura switch compara o caractere atual com os comandos do Brainfuck (>, <, +, -, ., ,, [,]).
4. Para cada comando, a ação correspondente é executada, como mover o ponteiro (pos_memoria), alterar o valor na memória, ler (uart_getchar_2) ou escrever (uart_putchar_2) dados.
5. Os colchetes [e] são tratados com loops while para garantir o fluxo correto do programa, pulando o código quando o valor do ponteiro é zero ou retornando ao início do loop.
6. O programa para em um loop while(1); ao final da execução.

Input e output

A comunicação é baseada em dois registradores de hardware, acessados através de ponteiros volatile:

- UART0DR (0x101f1000): O **Data Register** (Registrador de Dados). É usado para ler dados recebidos ou escrever dados para serem transmitidos.

- UART0FR (0x101f1018): O **Flag Register** (Registrador de Status). Contém bits que indicam o estado atual da UART, como se o buffer de transmissão está cheio ou se o buffer de recepção está vazio.

As funções implementadas são:

- **uart_putchar(char c)**: Envia um único caractere pela UART. A função primeiro entra em um loop
- **while** que verifica o bit 5 do UART0FR ((1 << 5)), que corresponde ao TXFF (Transmission FIFO Full). Ela espera até que o buffer de transmissão não esteja mais cheio antes de escrever o caractere no UART0DR.
- **uart_getchar()**: Recebe um caractere da UART. A função espera em um loop
- **UART0DR**, retornando apenas os 8 bits menos significativos (& 0xFF)
- **uart_puts(const char *s)**: Envia uma string completa. A função percorre a string caractere por caractere, chamando uart_putchar para enviar cada um até encontrar o terminador nulo (\0).

A função main demonstra o uso dessas rotinas. Ela exibe a mensagem "Digite algo:\n" e, em seguida, entra em um loop infinito. Dentro desse loop, o programa lê um caractere da entrada serial usando uart_getchar() e imediatamente o retransmite usando uart_putchar(), criando um "eco" do que foi digitado.

Funcionalidades do Debug Mode

O projeto inclui um modo de depuração que pode ser ativado e desativado com o comando d. As funcionalidades de depuração permitem um controle mais preciso da execução do programa:

- **Ver conteúdo da memória**: O comando m permite visualizar o conteúdo da memória, especificando um intervalo. Por exemplo, m 0 12 mostraria a memória das posições 0 a 12.
- **Adicionar breakpoints**: É possível adicionar pontos de parada (breakpoints) para que a execução seja interrompida em pontos específicos do código.

- **Execução passo a passo**: O comando n executa o próximo caractere do programa Brainfuck e para.
- **Execução até o próximo breakpoint**: O comando c continua a execução do programa até que ele encontre o próximo breakpoint.
- **Ver código atual**: É possível visualizar a posição atual no código-fonte, indicando com um v e b o caractere que está sendo executado.
- **Identificar loops**: O depurador tem a capacidade de sinalizar quando o programa está em um loop e mostrar o trecho de código correspondente.

Configuração e Execução no QEMU ARM

Para rodar o interpretador, é necessário um ambiente com o compilador arm-none-eabi-gcc e o QEMU configurado para emular um processador ARM.

Para instalar o QEMU no WSL do windows basta rodar os seguintes comandos no terminal:

- sudo apt update
- sudo apt install qemu-system-arm qemu-user-static gcc-arm-linux-gnueabihf build-essential

O processo de execução envolve os seguintes passos:

1. Compilação

Utilize o Makefile para compilar o código-fonte. Basta executar:

- make

Esse comando gera o arquivo kernel.bin, que é o binário executável para a arquitetura ARM, e o kernel.elf, utilizado na execução no QEMU.

2. Execução no QEMU

Carregue o binário no emulador com:

- qemu-system-arm -M versatilepb -cpu arm926 -nographic -kernel kernel.elf

Esse comando inicializa o sistema no modelo de placa VersatilePB com processador ARM926, desativa a interface gráfica (-nographic) e carrega o interpretador para execução direta no terminal.

Interação no terminal

Ao iniciar, o programa exibe o menu de comandos disponíveis. No **modo normal**, é possível carregar e executar programas Brainfuck. No **modo debug**, os comandos adicionais permitem inspeção e controle detalhado da execução, conforme descrito na seção anterior.

Conclusão

Resultados alcançados

O projeto alcançou seu objetivo de desenvolver um interpretador Brainfuck funcional que roda em um ambiente emulado ARM, aplicando os conhecimentos adquiridos na disciplina. A implementação do interpretador mostrou a compreensão da lógica da linguagem e a manipulação de estruturas de dados e ponteiros em C. A inclusão de um modo de depuração permite a análise detalhada do fluxo de execução. A utilização do QEMU e da ferramenta **Makefile** também validou a capacidade de configurar e automatizar o processo de compilação e execução para uma arquitetura de destino específica.

Limitações e Melhorias

O projeto, em sua forma atual, tem algumas limitações que podem ser abordadas em futuras melhorias como por exemplo o gerenciamento de erros, o interpretador não lida com erros de forma robusta, como colchetes desbalanceados ([e]) ou acesso a endereços de memória inválidos.