Trabalho 01

Membros:

Cauê Sordi Paulino - 14564520

Gustavo Curado Ribeiro - 14576732

Lucien Rodrigues Franzen - 14554835

Luís Filipe Silva Forti - 14592348

Esse código em Assembly para arquitetura RISC-V implementa um jogo de pedra, papel e tesoura, no qual o jogador faz sua jogada e o computador responde com uma jogada aleatória. O código também mantém uma lista das jogadas anteriores feitas pelo computador e exibe essa lista quando o jogador decide finalizar o jogo.

Funcionamento

1. **seções principais**:
    - `ler_jogada`: Exibe as instruções ao jogador e lê a jogada digitada.
    - `verificar_entrada`: Valida a jogada do jogador, verificando se é um número entre 1 e 4.
    - `verificar_resultado`: Gera uma jogada aleatória do computador e compara com a jogada do jogador, determinando o resultado (vitória, empate ou derrota). Adiciona a jogada do computador à lista.
    - **Resultado**: Dependendo da jogada, o programa imprime uma das três mensagens: vitória, empate ou derrota.
    - `aviso_erro`: Exibe uma mensagem de erro se a entrada for inválida.
2. **Seção de Dados**:
    - **Variáveis**: São usadas para armazenar entradas e referências para o começo e o fim da lista de jogadas.

- **Strings**: Mensagens para instruções, erros e resultados do jogo (vitória, empate, derrota) são armazenadas na memória.

3. **Manipulação de Lista**:
   - `adiciona_elemento`: Adiciona a jogada do computador à lista, se a lista estiver vazia, cria a lista, caso contrário, insere o novo elemento no fim.
   - `printa_Lista`: Quando o jogador opta por finalizar, imprime todas as jogadas feitas pelo computador, uma a uma, na ordem.
4. **Serviços do Sistema (ecall)**:
   - **Serviço 4**: Para print de strings
   - **Serviço 5**: Para leitura de ints.
   - **Serviço 9:** Para alocação de memória
   - **Serviço 42**: Para gerar um número aleatório.
   - **Serviço 10**: Para finalizar o programa.

Etapas de Desenvolvimento e Dificuldades

O programa permitiu entender como funciona a assembly RISC-V, e teve algumas dificuldades. Além disso, através do desenvolvimento foi necessário testar várias coisas paralelamente devido a nosssa inexperência com assembly.

As etapas no processo de desenvolvimento foram as seguintes:

1. Implementação da leitura de entradaPrecisamos, antes de tudo, receber a jogada do usuário para comparação.

- Utilizamos a seção `ler_jogada` para exibir instruções e capturar entrada. Então
  precisamos verificar de algum jeito que a entrada era válida, e fizemos `verificar_entrada` para definir 1 e 4 como válidos. 1 a 3 representam as jogadas, e 4 representa a saída. Caso estivesse errado, a seção `aviso_erro` foi
  feita para imprimir a mensagem de erro, começando no primeiro byte dela. Essa função precisou então executar
  `ecall` para chamar uma função do sistema operacional, como nesse caso

imprimir a string na tela. O programa também pula de volta para o início da próxima jogada porque se isso não acontecia, ocorreriam instruções sem sentido no
programa.

2. Desenvolvimento da geração de valor aleatório para a jogada do computador

- Integração na seção `verificar_resultado`
- Utilização do serviço do sistema 42 (ecall) para obtenção de número aleatório

Utilizamos o `ecall 42` para realizar uma chamada de sistema que gera um número pseudoaleatório. No código, o registrador `a0` é inicializado com uma semente arbitrária (por exemplo, o valor `719`), enquanto o registrador `a1` define o limite superior não inclusivo, ou seja, o número aleatório gerado estará no intervalo de 0 até `a1-1`. Após a execução do `ecall`, o valor gerado é armazenado no registrador `a0`, que posteriormente é ajustado na função para representar a jogada do computador.

3. Criação da lógica para verificação do resultado

- Comparação das jogadas do usuário e do computador
- Exibição do resultado utilizando mensagens pré-definidas

A lógica de verificação compara a jogada do usuário com a jogada aleatória gerada pelo computador. Após a geração da jogada do computador, ela é comparada à jogada do jogador, armazenada previamente. Se ambos fizerem a mesma jogada, o jogo resulta em empate. Caso contrário, as regras do jogo "Pedra, Papel e Tesoura" são aplicadas: o jogador vence se sua jogada for uma posição acima da jogada do computador (por exemplo, papel vence pedra), ou se o jogador escolher tesoura e o computador pedra (coberto pela lógica de subtração). Se nenhuma dessas condições for atendida, o jogador perde. O resultado final é exibido através de mensagens pré-definidas que indicam vitória, empate ou derrota.

4. Implementação da atualização da lista de jogadas

- Uso da função `adiciona_elemento` para registrar jogadas do computador
- Manutenção de um histórico completo da sessão do jogo

A função `adiciona_elemento` é responsável por armazenar cada jogada aleatória do computador na lista de jogadas. A estrutura da lista é dinâmica e é criada conforme as jogadas vão sendo registradas. Se a lista ainda estiver vazia, o primeiro elemento é salvo tanto em `lista_comeco` quanto em `lista_fim`. Para cada nova jogada, o endereço do último elemento da lista é atualizado para apontar para o novo elemento, garantindo que todas as jogadas anteriores sejam preservadas. Esse processo permite manter um histórico completo das jogadas do computador durante a sessão de jogo, que pode ser exibido ao final do programa para o usuário.

Desafios e Aprendizados

Desafios

1. **Adaptação à Arquitetura RISC-V**:
   - Um dos maiores desafios foi trabalhar com listas encadeadas, já que a arquitetura RISC-V não fornece instruções prontas como `push` ou `pop`, comuns em outras arquiteturas.
   - O desenvolvimento da lógica para adicionar e manipular elementos na lista demandou soluções criativas.
2. **Manipulação de Entrada e Saída**:
   - Implementar a leitura e validação de entradas com Assembly RISC-V exigiu um controle rigoroso sobre o fluxo do programa, especialmente no manuseio de valores inválidos.

Aprendizados

1. **Compreensão Aprofundada do Assembly RISC-V**:
   - O projeto proporcionou uma compreensão prática da arquitetura RISC-V, principalmente em relação ao uso de registradores e chamadas de sistema (syscalls).
2. **Resolução de Problemas em Ambientes de Baixo Nível**:
   - Trabalhar com Assembly requer uma abordagem detalhista e paciente, com foco em depuração e controle preciso sobre as operações realizadas.

3. **Trabalho em Equipe**:
    - A colaboração foi essencial para superar as dificuldades, e a divisão de responsabilidades tornou o processo de desenvolvimento mais fluido.

Conclusão

Este projeto em Assembly RISC-V demonstrou a capacidade de desenvolver um jogo simples com a utilização de estruturas de dados como listas encadeadas e o uso de chamadas de sistema para realizar operações de entrada e saída. As dificuldades encontradas durante o desenvolvimento trouxeram lições valiosas sobre a arquitetura RISC-V e a programação em baixo nível. O trabalho em equipe e a constante busca por soluções criativas permitiram o sucesso deste projeto.