Universidade Federal da Fronteira Sul - *Campus* Chapecó
Curso de Ciência da Computação
Disciplina: Organização de Computadores
Professor: Luciano Lores Caimi
Aluno: Luan Bortoli

Implementação do jogo de cartas Blackjack em Assembly

Introdução

O presente trabalho foi desenvolvido e implementado pelo aluno Luan Bortoli, sob matrícula 2121101061 para obtenção de nota na disciplina de Organização de Computadores.

O principal objetivo deste trabalho é implementar de forma simplificada o jogo de cartas Blackjack, também chamado de 21, utilizando a linguagem Assembly para a arquitetura RISC-V. O jogo foi executado no simulador RARS.

O desenvolvimento deste trabalho proporcionou ao aluno a oportunidade de explorar a manipulação de registradores e memória, o controle de fluxo e a interação com o terminal, utilizando a arquitetura do conjunto de instruções (ISA) do processador RISC-V.

Nas seções a seguir, serão apresentados o uso dos registradores, as funções implementadas, a estrutura de armazenamento, a lógica do jogo e um fluxograma que representa o funcionamento geral do programa.

Uso dos Registradores

Registrador	Principal finalidade
t0 - t6	Registradores temporários para cálculos, controle de fluxo, contadores e manipulação de arrays
a0 - a1	Argumentos para syscalls (chamadas de sistema) e retorno de valores
a7	Indica qual syscall (chamada de sistema) deve ser executada, syscall utilizadas: 1 - Imprime um número inteiro; 4 - Imprime uma string; 5 - Lê um número inteiro; 10 - Encerra o programa; 11 - Imprime um caractere; 42 - Gera um número aleatório
s1	Ponteiro para a mão do jogador
s2	Ponteiro para a mão do dealer

Funções implementadas

Para execução das etapas do jogo foram implementadas funções (rótulos), sendo as seguintes funções e sua aplicação:

- **main:** Exibe mensagem inicial, total de cartas, pontuações e inicia a rodada.
- **restaurar_cartas_iniciais e redefinir_cartas_disponiveis:** Restaura o baralho atribuindo 4 cartas para cada tipo, ficando ao todo 52 cartas.
- **iniciar_rodada:** Como o próprio nome diz, ele inicia a rodada, zerando o acumulador da mão do jogador e da mão do dealer, e utilizando o ponteiro para o registrador s1 para a mão do jogador e o registrador s2 para a mão do dealer.
- **iniciar_limpeza_mao_jogador, iniciar_limpeza_mao_dealer, zerar_mao_jogador e zerar_mao_dealer:** Asseguram que a mão do jogador/dealer ficará sem nenhuma carta para iniciar a rodada.
- **pedir_inicio_jogo:** Pergunta ao jogador se ele deseja jogar ou não.
- **iniciar_jogo_jogador e iniciar_jogada_dealer:** Responsável por iniciar a rodada, e por dar duas cartas ao jogador/dealer e mostrar o somatório de cartas que estão em sua mão.
- **comprar_mais_cartas_jogador:** Responsável por verificar se o usuário pode comprar mais cartas ou se já excedeu o valor máximo de 21, ou se ele ainda deseja continuar comprando cartas.
- **comprar_carta_jogador e comprar_carta_dealer:** Responsável pelo sorteio das cartas, verificar se a carta sorteada está disponível para ser utilizada, atualização da quantidade de cartas, e soma ao valor total de pontos do jogador ou dealer.
- **decrementa_valor_as_jogador e decrementa_valor_as_dealer:** Responsável por atualizar o valor do ‘às’ quando o jogador ou dealer está próximo de estourar o valor máximo de 21, o ‘às’ passa a valer 1 para evitar a derrota.
- **processar_carta_jogador e processar_carta_dealer:** Exibe a carta que o jogador ou dealer recebeu.
- **exibir_msg_mao_jogador, exibir_msg_mao_dealer, cartas_mao_jogador, cartas_mao_dealer, soma_valor_cartas_mao_jogador e soma_valor_cartas_mao_dealer:** Responsavél pela exibição da mensagem da mão do jogador/dealer, as cartas que estão na mão do jogador/dealer, e a somatória das cartas da mão do jogador/dealer em cada rodada, respectivamente.
- **comprar_mais_cartas_dealer:** Responsável por verificar se o dealer pode comprar mais cartas ou se já excedeu o valor máximo de 17, caso exceda, ele para de comprar cartas.
- **fim_dealer_jogo:** Finaliza a rodada após o jogador e dealer finalizarem as jogadas, e

chama a função para definir o vencedor.

- **eh_vencedor:** Define o vencedor da rodada, e soma 1 ponto a pontuação geral do jogo ao vencedor, em caso de empate, não é contabilizado ponto para o jogador e dealer.
- **fim_rodada, exibir_total_cartas, verificar_necessidade_restaurar_baralho e contagem_cartas_disponiveis**: São responsáveis por finalizar a rodada, restaurar o baralho, quando já for utilizada ao menos 40 cartas durante todas as rodadas, e verificar a quantidade total de cartas utilizadas e disponíveis.
- **exibir_status_jogo:** Exibir o total de cartas disponíveis em cada rodada, a pontuação geral do jogador e do dealer, e chama a função que pode iniciar uma nova rodada.

Estrutura de armazenamento

As cartas de cada jogador são armazenadas em arrays de 52 bytes na memória, com acesso e modificação por instruções de byte. O controle da quantidade de cartas restantes é feito por um vetor de inteiros representando o baralho. Durante cada rodada, a soma das cartas do jogador e do dealer é mantida nos registradores temporários t6 e t5, respectivamente. As pontuações acumuladas ao longo das rodadas são armazenadas em variáveis de memória nomeadas pontos_jogador e pontos_dealer.

Lógica do jogo

O jogador inicia cada rodada com duas cartas e pode optar por continuar comprando ou encerrar sua vez. Caso ultrapasse 21 pontos, perde automaticamente. O dealer, por sua vez, recebe inicialmente duas cartas, e se o jogador não ultrapassou 21 pontos, joga de forma automática até atingir pelo menos 17 pontos, se ultrapassar 21 pontos o jogador vence. Ao final de cada rodada, as pontuações são comparadas e o vencedor é determinado. Se mais de 40 cartas forem utilizadas, o baralho é restaurado antes da próxima rodada. A pontuação geral é atualizada ao longo do jogo. A lógica do jogo pode ser visualizada no fluxograma em anexo.

Conclusão

A implementação do jogo de cartas Blackjack em Assembly para a arquitetura RISC-V exigiu uma compreensão do funcionamento da arquitetura e do controle detalhado de registradores e memória. A lógica do jogo foi respeitada conforme o enunciado, com tratamento especial para o Ás, controle de baralho, e interação com o usuário via terminal. O trabalho proporcionou um excelente aprendizado sobre programação de baixo nível, modularidade e boas práticas em Assembly.

Anexo - Fluxograma com a lógica do jogo