TRABALHO DE ASSEMBLY - RISC-V

DESENVOLVIMENTO DO JOGO DE CARTAS BLACKJACK

OBJETIVO:

Neste trabalho deve-se implementar, em Assembly para a arquitetura RISC-V (RV32), uma versão simplificada do jogo de cartas Blackjack (também conhecido como *21*). O objetivo do trabalho é a compreensão e aplicação de conceitos de manipulação de dados, controle de fluxo e interação com o usuário através do terminal.

DESCRIÇÃO DO JOGO:

O Blackjack é um jogo de cartas jogado contra um "*dealer*" (o computador). O objetivo do jogo é ter uma mão de cartas que somem o valor mais próximo possível de 21, sem ultrapassar esse valor. Cada carta tem um valor numérico: cartas numeradas (2 a 10) de acordo com o número na carta, cartas de figuras (Rei, Dama, Valete) têm valor 10. Por fim, o Ás pode valer 1 ou 11, sempre favorecendo o jogador, sem ultrapassar 21.

REQUISITOS DO JOGO:

1. DISTRIBUIÇÃO DAS CARTAS:

- O jogador e o *dealer* recebem inicialmente 2 cartas.
- As cartas são representadas por números de 1 a 13, onde:
  - 1 = Ás
  - 2 a 10 = Cartas numeradas
  - 11 = Valete
  - 12 = Dama
  - 13 = Rei

2. REGRAS DO JOGO:

- O *dealer* dá duas cartas ao jogador e recebe duas cartas.
- As cartas recebidas pelo jogador são visíveis.
- Apenas uma carta do *dealer* é visível.
- Inicia-se uma sequência de rodadas para o jogador. A cada rodada o jogador deve "pedir mais" (**Hit**) uma carta ou "parar" (**Stand)**, encerrando sua sequência de rodadas.
- Após o jogador dar o comando **Stand**, inicia-se a sequência de rodadas do *dealer*.
- O *dealer* segue uma regra automática:
  - Se a soma das cartas for menor que 17, o *dealer* deve pedir mais (**Hit**).
  - Se for 17 ou mais, o *dealer* deve parar (**Stand**).
- Se em algum momento durante as rodadas do jogador ou do *dealer* o valor da mão ultrapassar 21, o jogo é encerrado dando a vitória ao oponente.
- Caso contrário vence a mão com a maior pontuação.
- Decreta-se empate caso ambas as mãos somem o mesmo valor.

3. Exibição no Terminal:

- O jogo deve exibir o estado atual das mãos de ambas as partes (jogador e *dealer*) no terminal.
- Após cada jogada, deve-se exibir a opção de continuar jogando (pedir mais (**Hit**) cartas ou parar (**Stand**)).

Instruções de Funcionamento:

1. Início do Jogo:

- O terminal exibe uma mensagem de boas-vindas e a instrução para o jogador iniciar o jogo.
- O jogador pode optar por começar ou sair.

2. Jogada do Jogador:

- Após receber as duas primeiras cartas, o jogador deve decidir se deseja "pedir mais" ou "parar".
- Se "pedir mais" (**Hit**), uma nova carta é distribuída ao jogador e o total da mão é atualizado.
- Se "parar" (**Stand**), o turno do jogador termina e é a vez do *dealer* jogar.

3. Jogada do Dealer:

- O *dealer* joga automaticamente conforme as regras: continua "pedindo mais" (**Hit**) cartas até que sua mão tenha 17 ou mais.
- O *dealer* revela sua mão final e o vencedor é determinado.

4. Resultados:

- Após o *dealer* jogar, o resultado é exibido:
  - Se o jogador ultrapassou 21, o *dealer* vence.
  - Se o *dealer* ultrapassou 21, o jogador vence.
  - Caso ambos fiquem abaixo de 21, quem tiver o valor mais alto vence.
- O resultado é exibido e o jogador pode optar por **jogar novamente** ou sair.

Requisitos Técnicos:

1. Estrutura de Dados:

- Use registradores para armazenar valores temporários, como as cartas do jogador e do *dealer*, o total da mão de cada um e o número de cartas.
- Use a memória para armazenar as cartas restantes do baralho e para manter o estado do jogo.

2. Controle de Fluxo:

- O jogo deve ser baseado em loops que controlam as rodadas do jogador e do *dealer*.
- Use saltos condicionais para verificar as condições de vitória e derrota, além das decisões do jogador durante o jogo.

3. Interação com o Usuário:

- O programa deve interagir com o usuário via entrada e saída padrão no terminal. Use chamadas de sistema para ler entradas do jogador e exibir informações no terminal.

4. Geração das Cartas:

- Para que o jogo fique interessante, deve-se utilizar o gerador de números aleatórios para sortear as cartas.
- O RARS possui uma chamada de sistema que produz um número aleatório dentro de uma faixa:

RandIntRange	42	Get a random bounded integer	a0 = index of pseudorandom number generator a1 = upper bound for random number	a0 = uniformly selectect from [0,bound]

EXEMPLO DE SAÍDA:

```
Bem-vindo ao Blackjack!

O jogador recebe: 7 e 8
O dealer revela: 4 e uma carta oculta

Sua mão: 7 + 8 = 15
O que você deseja fazer? (1 - Hit, 2 - Stand): 1

O jogador recebe: 6
Sua mão: 7 + 8 + 6 = 21
O dealer revela sua mão: 4 + 10 = 14

O dealer deve continuar pedindo cartas...
O dealer recebe: 8
O dealer tem: 4 + 10 + 8 = 22

O dealer estourou! Você venceu!

Deseja jogar novamente? (1 - Sim, 2 - Não): 2
```

CRITÉRIOS DE AVALIAÇÃO:

- **Funcionalidade:** O jogo funciona corretamente no terminal, com interação completa (distribuição de cartas, jogadas, vitória/derrota).
- **Estrutura do Código:** O código está bem organizado, modular e documentado. Utiliza conceitos adequados de Assembly, como controle de fluxo e manipulação de registradores e memória.
- **Interatividade e Saída no Terminal:** A interação com o usuário é clara, e o jogo é exibido de forma legível no terminal.

ENTREGA:

- Este trabalho deve ser realizado em **grupos de até 3 pessoas**.
- Deve ser entregue, por apenas um dos integrantes do grupo, um arquivo **.zip** contendo:
    - O código em Assembly para a arquitetura RISC-V deve ser entregue em um arquivo *.asm* contendo os comentários necessários.
    - Um arquivo de documentação (**no máximo 2 páginas**) contendo o nome dos integrantes do grupo, explicando o uso dos registradores, enumerando e descrevendo as funções implementadas e de que forma foi construída a estrutura que guarda as cartas do jogador e do *dealer* na memória.