**Pontifícia Universidade Católica do Rio Grande do Sul**
**Escola Politécnica**
**Fundamentos de Sistemas Computacionais**
**98800-04**

Prof. Iaçanã Ianiski Weber

# *Trabalho II - Assembly RISC-V*
# Jogo de Cartas Blackjack (21)

**Prazo de Entrega: 01/06/2025 - 23:59 - via Moodle**

## 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, utilizando o simulador RARS.

## 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 um valor o mais próximo possível de 21, sem ultrapassar esse valor. Cada carta tem um valor numérico específico para o jogo:

- Cartas numeradas (2 a 10) valem seu próprio número.
- Cartas de figuras (Valete, Dama, Rei) valem 10 pontos cada.
- O Ás tem um valor flexível: vale 11 pontos, a menos que isso faça a mão ultrapassar 21, caso em que passa a valer 1 ponto. A escolha deve sempre favorecer o jogador (ou dealer), maximizando a pontuação sem "estourar" (bust).

O jogador compete apenas contra o dealer, não contra outros jogadores.

## Requisitos do Jogo

### Distribuição e Representação das Cartas

- O jogador e o dealer recebem inicialmente 2 cartas cada.
- Para simplificar a geração, as cartas podem ser representadas internamente por números de 1 a 13, onde:
  - 1 = Ás (A)
  - 2 a 10 = Cartas numeradas (2 a 10)
  - 11 = Valete (J)

  - 12 = Dama (Q)
  - 13 = Rei (K)

- **Importante:** Ao calcular a pontuação da mão, essa representação (1-13) deve ser convertida para o valor do Blackjack (A=1/11, 2-10=valor facial, J/Q/K=10).

- A geração de cartas deve usar o gerador de números aleatórios do RARS (ver Requisitos Técnicos). Para este trabalho, pode-se assumir um "baralho infinito", onde cada pedido de carta gera um novo número aleatório entre 1 e 13, sem se preocupar em remover cartas de um baralho finito.

## Regras e Fluxo

1. **Início da Rodada:** O dealer distribui duas cartas para o jogador (ambas visíveis) e duas para si mesmo (apenas a primeira visível, a segunda permanece oculta inicialmente).

2. **Turno do Jogador:**

   - A soma inicial da mão do jogador é calculada e exibida (considerando a regra do Ás).
   - O jogador decide entre:
     - **Pedir mais (Hit):** Recebe uma nova carta. A soma é recalculada. Se a soma ultrapassar 21 ("estourar"), o jogador perde imediatamente e o turno do dealer não ocorre. O jogador pode pedir quantas cartas quiser, desde que não estoure.
     - **Parar (Stand):** O jogador está satisfeito com sua mão e encerra seu turno.

3. **Turno do Dealer:**

   - Ocorre apenas se o jogador não estourou e escolheu "Parar".
   - O dealer revela sua carta oculta. A soma inicial de sua mão é calculada.
   - O dealer deve seguir uma regra fixa:
     - Se a soma for menor que 17, o dealer **obrigatoriamente** pede mais uma carta (Hit).
     - Se a soma for 17 ou mais, o dealer **obrigatoriamente** para (Stand).
   - Se o dealer pedir cartas e estourar (ultrapassar 21), o jogador vence imediatamente.

4. **Determinação do Vencedor (se ninguém estourou):**

   - Comparam-se as somas finais das mãos do jogador e do dealer.
   - Quem tiver a maior soma (sem ultrapassar 21) vence.
   - Em caso de somas iguais, é declarado empate.

# Interface e Funcionamento no Terminal

## Exibição

- O jogo deve exibir o estado atual das mãos e pontuações do jogador e do dealer de forma clara.
- A segunda carta inicial do dealer deve ser exibida como oculta (ex: usando "?", "X", ou "Oculta") até o início do turno do dealer.
- Mensagens informativas devem guiar o jogador (ex: "Sua vez", "Vez do Dealer", resultados).

## Fluxo Interativo

- **Início do Jogo:** Exibir mensagem de boas-vindas. Perguntar se o jogador deseja iniciar uma nova partida (ex: 1 para Sim, 2 para Não). Se não, encerrar o programa.
- **Jogada do Jogador:** Após receber as cartas iniciais, exibir a mão do jogador, sua soma, e a carta visível do dealer. Perguntar ao jogador se deseja "Pedir mais" (Hit) ou "Parar" (Stand) (ex: 1 para Hit, 2 para Stand). Repetir o pedido de carta e a decisão até o jogador "Parar" ou "Estourar".
- **Jogada do Dealer:** Após o jogador parar, exibir a mão completa inicial do dealer e sua soma. Anunciar as ações automáticas do dealer (pedindo cartas ou parando) conforme a regra (<17 Hit, >=17 Stand). Exibir cada carta adicional que o dealer recebe.
- **Resultados:** Exibir claramente o resultado final (Jogador Venceu, Dealer Venceu, Empate) e as pontuações finais de ambos. Perguntar novamente se deseja jogar outra partida.

# Requisitos Técnicos

## Estrutura de Dados

- Utilize registradores RISC-V para armazenar valores temporários importantes durante os cálculos (ex: valor da carta atual, somas parciais, contadores).
- Utilize a memória (segmento `.data` para constantes, mensagens, e para dados variáveis) para armazenar o estado do jogo, incluindo:
  - As cartas na mão do jogador (sugestão: um array/vetor de bytes/words).
  - As cartas na mão do dealer (similar ao jogador).
  - Contadores para o número de cartas em cada mão.
  - A soma atual de cada mão.
  - (Opcional) Um contador de Ases em cada mão para facilitar o cálculo do valor flexível (1 ou 11).
- A documentação entregue **deve** detalhar como as mãos foram armazenadas.

## Controle de Fluxo

- O jogo deve ser estruturado usando laços (*loops*) para controlar as rodadas de "Pedir mais" do jogador e as ações automáticas do dealer.
- Use instruções de desvio condicional (`beq`, `bne`, `blt`, `bge`, etc.) extensivamente para:
  - Implementar a lógica de decisão do jogador (Hit/Stand).
  - Implementar a regra automática do dealer (<17 Hit, >=17 Stand).
  - Verificar condições de "estouro" (> 21).
  - Comparar as mãos e determinar o vencedor/empate.
  - Controlar o fluxo principal do jogo (iniciar, turno do jogador, turno do dealer, resultado, jogar novamente).

## Interação com o Usuário

- Utilize chamadas de sistema (`ecall`) do RARS para toda a interação:
  - Imprimir strings (mensagens, status): `ecall` código 4.
  - Imprimir inteiros (pontuações, valores de cartas): `ecall` código 1.
  - Ler inteiro do usuário (para decisões Hit/Stand, Jogar Novamente): `ecall` código 5.
  - Encerrar o programa: `ecall` código 10.

## Geração das Cartas (Aleatoriedade)

- Para gerar as cartas (representadas por 1-13), utilize o gerador de números aleatórios do RARS.
- A chamada de sistema para gerar um inteiro aleatório dentro de uma faixa é:
  - Colocar o código da *syscall* em `a7`: **42**
  - Argumento `a0`: ID do gerador (usar **0** para o gerador padrão).
  - Argumento `a1`: Limite superior **não-incluso**. Ou seja, para gerar números de 1 a 13, o limite superior deve ser 14.
  - O número aleatório gerado será retornado no registrador `a0`.
- Exemplo de uso para gerar um valor entre 1 e 13 (inclusive):

```
# Gera um numero aleatorio entre 1 e 13 usando syscall 42 (Random Int Range)
# RARS gera numeros no intervalo [0, a1-1], portanto usamos a1 = 13 e somamos 1

li a7, 42       # Syscall 42: Random Int Range
li a0, 0        # ID do gerador aleatorio (0 = padrao)
li a1, 13       # Limite superior exclusivo => gera valores de 0 ate 12
ecall           # Executa a chamada de sistema
                # Agora, a0 contem um valor aleatorio entre 0-12

addi a0, a0, 1  # Ajusta o intervalo para [1, 13]

# Agora a0 contem um numero aleatorio entre 1 e 13 (inclusive)
```

- Lembre-se de converter este valor (1-13) para o valor de jogo (1/11, 2-10, 10) ao calcular as somas.

## Exemplo de Saída no Terminal

```
Bem-vindo ao Blackjack!
Deseja jogar? (1 - Sim, 2 - Nao): 1

--- Nova Rodada ---
Sua mao: [ 7 ][ 8 ]  (Soma: 15)
Mao do Dealer: [ 4 ][ ? ]

O que voce deseja fazer? (1 - Hit, 2 - Stand): 1

Voce recebeu: [ 6 ]
Sua mao: [ 7 ][ 8 ][ 6 ]  (Soma: 21) - Blackjack!

--- Vez do Dealer ---
Dealer revela a carta oculta: [ 10 ]
Mao do Dealer: [ 4 ][ 10 ] (Soma: 14)

Dealer precisa pedir (< 17)...
Dealer recebe: [ 8 ]
Mao do Dealer: [ 4 ][ 10 ][ 8 ] (Soma: 22) - Estourou!

*** Voce venceu! ***

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

Obrigado por jogar!
```

## Critérios de Avaliação

- **Funcionalidade:** O jogo implementa corretamente as regras do Blackjack (distribuição, valores das cartas, regra do Ás, Hit/Stand, regra do dealer, verificação de estouro, determinação do vencedor/empate)? A interação via terminal está completa e funcional?
- **Estrutura do Código:** O código Assembly está bem organizado (uso de labels, funções/procedimentos se aplicável)? É legível e comentado adequadamente? O uso de registradores e memória é eficiente e correto?
- **Interatividade e Saída:** A interação com o usuário é clara e intuitiva? O estado do jogo (mãos, pontuações) é exibido de forma compreensível no terminal?
- **Documentação:** A documentação entregue atende aos requisitos (nomes, uso de registradores, descrição de funções/procedimentos, estrutura de dados das mãos)?

# Entrega

- Este trabalho deve ser realizado em **grupos de até 4 pessoas**.
- Deve ser entregue, **por apenas um dos integrantes do grupo**, via Moodle, um único arquivo `.zip` contendo:
  1. O código fonte completo em Assembly RISC-V (`.s` ou `.asm`), devidamente comentado.
  2. Um arquivo de documentação (**PDF** com no **máximo 2 páginas**) contendo:
     - Nome completo e matrícula de todos os integrantes do grupo.
     - Uma descrição clara de como as mãos do jogador e do dealer foram armazenadas na memória (ex: quais endereços/labels, qual o formato dos dados).
     - Uma explicação sobre o uso dos principais registradores no seu código (ex: quais registradores guardam as somas das mãos, ponteiros para as mãos, etc.).
     - Se você dividiu o código em funções/procedimentos (usando '`jal`'/'`jr ra`'), liste e descreva brevemente o propósito de cada função.
- **Plágio não será tolerado**. Trabalhos com trechos copiados entre grupos ou de outras fontes sem a devida citação terão **nota zero**.