UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
CAMPUS DE CHAPECÓ
CURSO DE CIÊNCIA DA COMPUTAÇÃO

GUSTAVO DA FONSECA ROZA & HENRIQUE RIBEIRO RODRIGUES

BLACKJACK
EM ASSEMBLY!

CHAPECÓ
Junho de 2025

SUMÁRIO

1 INTRODUÇÃO

Este trabalho, desenvolvido pelos alunos: Gustavo da Fonseca Roza (2211100074) & Henrique Ribeiro Rodrigues (2211100005).

Tem como objetivo apresentar e descrever o código desenvolvido, utilizando assembly com o ISA do RISC-V (1).

O código consiste em uma implementação do jogo **BlackJack!**. Popularmente conhecido como **Vinte e Um**.

2 PROBLEMATIZAÇÃO

A proposta inicial era fazermos uma versão simplificada do jogo **BlackJack!** para testar nossos conhecimentos em: armazenamento de registradores, manipulação desses dados entre a memória principal e os registradores, controle de fluxo com a aplicação de funções e rótulos. Isso utilizando a arquitetura de instruções(ISA) do processador RISC-V, sendo a solução criada em um arquivo (.asm) executada no simulador (RARS). (2).

3 DESENVOLVIMENTO

REGISTRADORES

Registrador	Função Principal
`s0`	Pontuação do dealer
`s1`	Pontuação do jogador
`s2`	Total de cartas restantes no baralho
`s3`	Número de cartas do jogador
`s4`	Número de cartas do dealer
`s5`	Valor calculado da mão do dealer
`s6`	Valor calculado da mão do jogador
`t0-t5`	Registradores temporários para cálculos, contadores, offsets e manipulação de arrays
`a0-a1`	Argumentos para funções, valores de cartas, ponteiros para arrays, syscalls
`ra`	Endereço de retorno das funções (salvo/restaurado em chamadas de função)
`sp`	Ponteiro da pilha (stack pointer), usado para salvar/restaurar ra

Tabela 1 – Uso dos Registradores

FUNÇÕES:

1. **main:** É a função principal que inicia o programa. Inicializa as pontuações do jogador e do dealer em zero, exibe a mensagem de boas-vindas e entra no loop principal do jogo (`breckJacquiLoop`).

2. **exibePontuacao:** Esta função é responsável por mostrar no terminal a pontuação atual do jogador e do dealer. Carrega os valores dos registradores de pontuação (s0 para o dealer e s1 para o jogador) e os imprime na tela.

3. **exibePontuacaoFinal:** Tem básicamente a mesma lógida da função `exibePontuacao`, porém, é utilizada quando o jogador decide para de jogar, dessa forma, faz três comparações para abranger todos os três casos possíveis, (dealer ganha, empate ou jogador ganha) levando em consideração a pontuação total, e imprime o rótulo correspondente com o resultado.

4. **embaralha:** Zera o contador-cartas, efetivamente "resetando"o baralho. É chamada quando o número de cartas restantes no baralho é menor que 12, garantindo tenha cartas suficiente para as próximas rodadas.

5. **novaRodada:** Inicia uma nova rodada do jogo. Zera os contadores de cartas da rodada, distribui duas cartas para o jogador e duas para o dealer, e exibe as cartas iniciais (ambas do jogador e a primeira do dealer).

6. **distribuiCartaDealer:** Gera e distribui uma carta aleatória. Ela utiliza a chamada de sistema (***syscall***) 42 do RARS para gerar um número aleatório entre 0 e 12, o que é corrigido, fazendo uma soma de +1, para que fique compatível ao baralho, que é de 1 à 13. A função também verifica no contador-cartas se o limite de 4 cartas para aquele número já foi atingido. Se sim, ela gera um novo número.

7. **turnoJogador:** Gerencia as ações do jogador. Em um loop, a função exibe a mão atual do jogador, calcula a soma e pergunta se o jogador deseja "pedir mais uma carta"(**Hit**) ou "parar"(**Stand**). Se o jogador pedir uma carta e a soma ultrapassar 21 ("estourar"), a função encerra o turno e retorna um valor indicando a derrota.

8. **turnoDoDealer:** Executa a jogada automática do dealer. O dealer segue uma regra fixa: ele deve pedir cartas até que a soma de sua mão seja 17 ou mais. A função revela a carta oculta do dealer e continua pedindo cartas até atingir o limite.

9. **calcularMao:** Calcula o valor total de uma mão (seja do jogador ou do dealer). A função percorre as cartas da mão, somando seus valores. Ela trata as cartas de figura (Valete, Dama, Rei) como 10 e tem uma lógica especial para o Ás, que pode valer 1 ou 11, sempre da maneira mais benéfica para não ultrapassar 21.

10. **comparaMao:** Após os turnos do jogador e do dealer, compara as pontuações finais para determinar o vencedor. Verifica quem tem a maior pontuação sem estourar 21 ou se houve um empate, e então atualiza a pontuação geral do jogo.

11. **finaliza:** Realiza um JumpAndLink para a função `exibePontuacaoFinal` e após isso, encerra a execução do programa utilizando a chamada de sistema (***syscall***) 10.

SUB-ROTINAS E LOOPS IMPORTANTES

1. **breckJacquiLoop:** O loop principal do jogo, localizado na função `main`. A cada iteração, ele exibe as pontuações, pergunta se o jogador deseja iniciar uma nova rodada e verifica se o baralho precisa ser reembaralhado.

2. **dealerEstouraLoop:** Dentro do `turnoDoDealer`, este loop continua a solicitar cartas para o dealer enquanto a sua pontuação for inferior a 17, seguindo a regra padrão do jogo.

3. **turnoJogadorLoop:** O loop central do turno do jogador. A cada passagem, exibe a mão atual do jogador, calcula os pontos e aguarda a próxima ação: "**Hit**"(pedir) ou "**Stand**"(parar).

4. **jogadorEstourou:** Um bloco de código específico que é acionado quando a pontuação do jogador ultrapassa 21, tratando a condição de derrota imediata.

5. **jogadorGanha / dealerGanha:** Blocos de código dentro da função `comparaMao` que são executados para declarar o vencedor da rodada e atualizar o placar geral.

6. **LoopParaCalcularCarta:** O loop principal dentro da função `calcularMao`. Ele itera sobre cada carta na mão de um participante para somar o valor total.

7. **calculaAses:** Uma rotina lógica especializada dentro de `calcularMao`, responsável por determinar o valor do Ás (1 ou 11) de forma estratégica, após a soma das outras cartas.

8. **pescaCarta:** Um loop de segurança dentro de `distribuiCartaDealer`. Ele garante que uma carta válida seja gerada, tentando um novo sorteio caso a carta selecionada já tenha sido distribuída quatro vezes.

9. **fimRodada:** Bloco de código executado ao final de cada rodada (após uma vitória, derrota ou empate), responsável por restaurar os registradores da pilha e preparar o retorno ao loop principal do jogo.

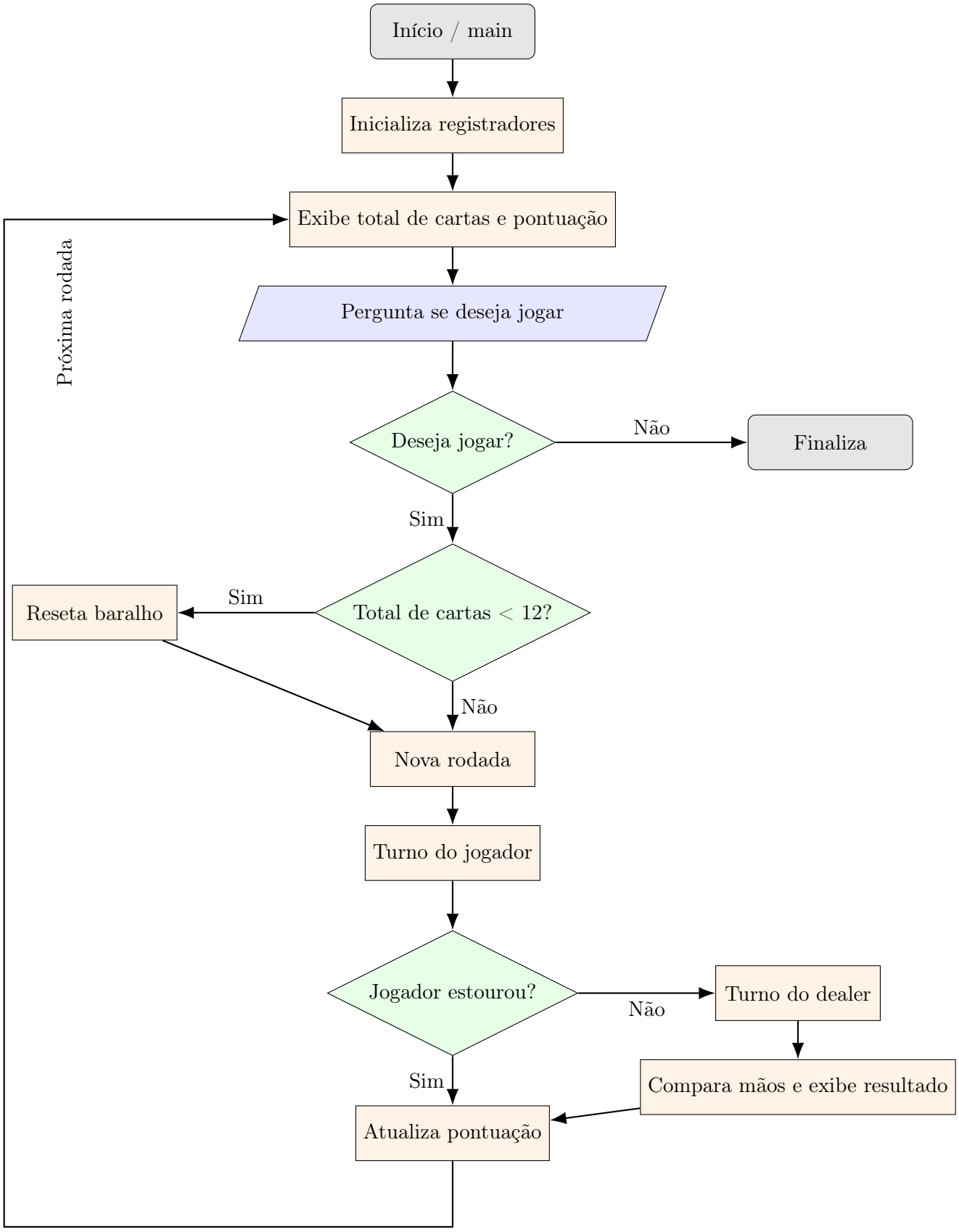
10. **empateFinal / jogadorGanhaFinal / dealerGanhaFinal:** Funções auxiliares, usadas pela função `exibePontuacaoFinal`, exibem respectivamente, o resultado final do jogo.

ESTRURURAS DE ARMAZENAMENTO

Em resumo, a estrutura para guardar as cartas do **jogador** e do **dealer** foi implementada como um array de bytes de tamanho fixo (40 bytes). Este espaço é reservado na seção **.data** usando a diretiva **.space**. As cartas são armazenadas sequencialmente dentro deste array, um byte por carta, utilizando as instruções **la (load address)** para apontar para o array e **sb (store byte)** para inserir cada nova carta na posição correta, controlada por um registrador que funciona como um contador.

Além disso, foi implementado o gerenciamento manual da pilha para preservar o contexto de execução entre chamadas de funções. Utilizando o registrador `sp`, instruções como `sw` e `lw` foram aplicadas para salvar e restaurar o endereço de retorno (`ra`). Essa prática foi especialmente importante nas funções que fazem uso de chamadas internas, como `novaRodada`, `turnoJogador` e `distribuiCartaDealer`, garantindo o retorno correto ao ponto de origem após a execução da sub-rotina.

4 FLUXOGRAMA



5 CONCLUSÃO

Com a finalização deste trabalho, conclui-se que sua realização foi um grande desafio, pois exigiu um tempo considerável de aprendizado e concentração no domínio de uma linguagem de baixo nível. Em suma, este aprendizado contribuiu muito para nosso percurso acadêmico, além de nos proporcionar uma nova visão sobre esta área da computação.

REFERÊNCIAS

1 ORG, RISC-V. **Site oficial do RISC-V**. acessado em: 20/06/2025. 2015. Disponível em: <`https://riscv.org/`>.

2 RARS. **Fonte Oficial do simulador RARS**. acessado em: 26/05/2025. 2023. Disponível em: <`https://github.com/TheThirdOne/rars`>.