PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL

ESCOLA POLITÉCNICA - CURSO DE ENGENHARIA DE SOFTWARE

FUNDAMENTOS DE SISTEMAS COMPUTACIONAIS - PROF. IAÇANÃ IANISKI WEBER

TRABALHO II – JOGO DE CARTAS *BLACKJACK* EM *ASSEMBLY* DO *RISC-V*

Luca Wolffenbüttel Bohnenberger, Lucas da Paz Oliveira e Rodrigo Miotto Slongo

I. APRESENTAÇÃO

Este trabalho consiste na implementação de uma versão simplificada do jogo de cartas *Blackjack* (também conhecido como 21) em *Assembly* para a arquitetura *RISC-V* (RV32). Seu objetivo foi a compreensão e aplicação de conceitos como: manipulação de dados, chamadas de sistema, controle de fluxo e interação com o usuário através do terminal

II. SOBRE O JOGO

Nesta implementação, o jogador joga contra um "*dealer*" (computador) e o objetivo é ter uma mão de cartas que somem o valor mais próximo de 21, sem ultrapassar este valor. As cartas numeradas de 2 a 10 valem seu respectivo número, as cartas Valete, Dama e Rei (J, Q e K ou 11, 12 e 13, respectivamente) valem 10 pontos e o Às (1) pode valer 1 ou 11 pontos, sempre favorecendo o jogador.

No início, o *dealer* e o jogador recebem duas cartas cada, sendo que apenas uma carta do *dealer* é revelada; o jogador pode pedir mais cartas (*hit*) ou parar (*stand*), sendo que sua rodada é encerrada se somar 21, ultrapassar este valor ou parar. Após o fim da rodada do jogador, o *dealer* joga seguindo a regra definida pelo programa: enquanto a soma de suas cartas for menor do que 17, ele pede mais cartas.

O vencedor é quem tiver a pontuação mais próxima (ou igual a) 21, sem ultrapassar este valor. Se o jogador "estourar" — i.e., ultrapassar 21 pontos — o *dealer* apenas revela suas cartas e é o vitorioso; caso o *dealer* ultrapasse os 21 pontos, então o jogador é vitorioso. Se ambos encerrarem as rodadas com a mesma pontuação, o jogo empata. Após o final do jogo, o jogador pode optar por jogar novamente ou encerrar o programa.

III. SOBRE A IMPLEMENTAÇÃO

Para facilitar a compreensão e a escrita do código, dados e *labels* foram prefixados de forma a tornar mais claro seu conteúdo/significado. No que se refere aos dados declarados na seção `.data`, *strings* foram prefixadas com `s_` e vetores com `v_`; referente às *labels*, funções foram prefixadas com `f_` e procedimentos com `p_`.

Sabendo que o valor das cartas, em inteiros, varia de 1 a 13, foram utilizados vetores de *half words* (16 *bits*) para armazená-las. Visando garantir que cada carta possa ser "comprada" apenas 4 vezes, como em um jogo real utilizando um baralho, foi criado um vetor de 13 posições (`v_DrawnCards`), em que cada posição representa uma carta e seu valor é a quantidade de vezes que a carta já foi comprada no jogo (e.g.: a posição 0 representa a carta Ás; se o valor nesta posição for 3 significa que já foram compradas, pelos jogadores, 3 cartas Ás, i.e., existe apenas mais uma carta Ás disponível). Para representar as cartas de cada jogador durante o jogo, foram criados vetores de tamanho 9 nos quais cada posição armazena o valor de uma carta (de 1 a 13); o tamanho 9 justifica-se pois é, em um cenário extremo, a quantidade máxima de cartas que um jogador consegue ter em sua mão até atingir ou ultrapassar os 21 pontos e ter sua rodada encerrada (e.g.: quatro cartas 2, quatro cartas 3 e um Ás).

Para ser possível exibir na tela os símbolos das cartas Ás, Valete, Dama e Rei, foi utilizado um vetor do tipo `.ascii`; O motivo desta escolha é que, utilizando `.asciiz` é necessário informar o *null termination character* (`\0`) explicitamente, o que torna claro que o deslocamento para acessar cada posição do vetor é de 2 *bytes*.

Com a ciência de que os registradores do tipo S, se modificados dentro de uma função, devem ser restaurados antes de retornar para o *caller*, foram utilizados, na `main`, os registradores de *s1* a *s5* para salvar, nesta ordem, a opção de *hit*/*play* (1), a opção de *stand*/*stop* (2), o endereço de `v_DrawnCards` e os endereços dos vetores de cartas do jogador e do *dealer* (`v_PlayerCards` e `v_DealerCards`, respectivamente). Dentro de procedimentos e funções, sempre que possível são utilizados os registradores T; se necessário salvar algum valor, são utilizados os registradores S partindo de *s11* para trás. Para passagem de argumentos e retornos, são utilizados os registradores A partindo de *a0*.

Ações que permeiam todo o código foram extraídas para procedimentos ou funções. E.g.: `p_PrintString` recebe no registrador *a0* o endereço de uma *string* e se encarrega de carregar o registrador *a7* corretamente e chamar o sistema operacional com `ecall`; de modo similar, `p_PrintInt` recebe no registrador *a0* o inteiro que deve ser impresso no console. A função `f_ReadInt` retorna, no registrador *a0*, o inteiro digitado pelo usuário.

O procedimento `p_Run` é responsável pela execução do programa; é ele quem inicia o jogo chamando o procedimento `p_Blackjack` e, após sua execução, solicita ao usuário se deseja jogar novamente. Quando a opção do usuário for por não jogar novamente, o procedimento `p_Run` será finalizado, o que finalizará, posteriormente, o programa.

O procedimento `p_Blackjack` é o responsável pelo jogo em si; ele atribui duas cartas para cada jogador, chama a função `f_PlayerRound` para executar a rodada do usuário e após chama, se

necessário, f_DealerRound para executar a rodada do *dealer*. Após o fim das rodadas, imprime o vencedor e reseta o jogo (i.e., zera os vetores de cartas).

IV. PROCEDIMENTOS E FUNÇÕES

Abaixo, é possível conferir a lista completa de procedimentos e funções com suas descrições, parâmetros e valores de retorno (se houver):

4.1 main

É onde ocorre a execução do programa; carrega nos registradores S de 1 a 5 a opção de *hit*/*play*, a opção de *stand*/*stop*, o endereço do vetor das cartas já compradas e os endereços dos vetores de cartas do jogador e do *dealer*. Chama o procedimento p_Run e, após sua execução, pula para halt para encerrar o programa.

4.2 p_Run

Procedimento: imprime o cabeçalho do jogo e inicia um *loop* onde é chamado o procedimento p_Blackjack e, após sua execução, é solicitado ao jogador se ele deseja continuar jogando. Se a opção do jogador for para continuar, o *loop* é reiniciado; se a opção do jogador for para não continuar, o *loop* termina e o procedimento se encaminha para o fim da execução. Se a opção digitada for inválida, o usuário é informado e é solicitada uma nova entrada.

4.3 f_DrawCard

Parâmetros: *a0* – endereço do vetor de cartas do jogador que está recebendo a carta.

Retorna: *a0* – a carta sorteada.

Função: sorteia um número entre 0 e 12, representando uma carta; verifica no vetor de cartas compradas se a carta está disponível (i.e., se a posição equivalente neste vetor é menor do que 4). Se não estiver disponível, sorteia outra carta; repete esta ação até que a carta sorteada esteja disponível. Após, adiciona 1 ao valor da carta (para estar entre 1 e 13) e adiciona na primeira posição vazia do vetor passado como argumento (i.e., primeira posição com valor 0).

4.4 p_Deal

Procedimento: um *loop* que vai de 0 a 3 (incluso). Chama f_DrawCard passando, quando o valor do contador for ímpar, o endereço do vetor de cartas do jogador, e quando for par o endereço do vetor de cartas do *dealer*. Ao final da execução, ambos os vetores devem ter duas cartas.

4.5 p_Blackjack

Procedimento: inicializa o jogo chamando p_Deal e p_PrintGameStart; chama f_PlayerRound para a rodada do jogador e, caso necessário, f_DealerRound para a rodada do

dealer. Após o fim das rodadas, imprime o resultado chamando `p_PrintWinner` e volta o jogo para o estado inicial chamando `p_ResetGame`.

4.6 p_PrintGameStart

Procedimento: Mostra o estado inicial do jogo, revelando as cartas recebidas pelo jogador e uma das cartas do *dealer*.

4.7 f_PlayerRound

Retorna: *a0* – quantidade de pontos do jogador após o fim da sua rodada.

Função: soma os pontos do jogador utilizando `f_SumPoints`. Enquanto a quantidade de pontos for menor do que 21, são chamados `p_PrintPlayerMove` e `f_ReadInt` para o jogador escolher se quer *hit* ou *stand*; sempre que escolher a primeira opção, é chamada `f_DrawCard` para que o jogador recebe uma nova carta e `p_PrintDrawnCardPlayer` e `p_PrintPlayerHand` para exibir o estado atual na tela. A execução se repete até que o usuário escolha *stand* ou a sua pontuação exceda o limite.

4.8 f_DealerRound

Retorna: *a0* – quantidade de pontos do *dealer* após o fim da sua rodada.

Função: soma os pontos do *dealer* utilizando `f_SumPoints` e revela sua carta oculta chamando `p_RevealDealerHand`. Enquanto a quantidade de pontos for menor do que 17 — i.e., a regra definida na especificação —, o *dealer* solicita um *hit* e é chamada `f_DrawCard` para que ele receba uma nova carta e `p_PrintDrawnCardDealer` e `p_PrintDealerHand` para exibir o estado atual na tela. A execução se repete até que a pontuação atinja ou exceda o limite.

4.9 p_PrintDrawnCardDealer

Parâmetros: *a0* – A carta recebida pelo *dealer*.

Procedimento: chama `p_PrintString` e `p_PrintCard` para imprimir na tela a carta recebida pelo *dealer* com a mensagem correspondente.

4.10 p_PrintDrawnCardPlayer

Parâmetros: *a0* – A carta recebida pelo jogador.

Procedimento: chama `p_PrintString` e `p_PrintCard` para imprimir na tela a carta recebida pelo jogador com a mensagem correspondente.

4.11 p_PrintPlayerMove

Procedimento: chama `p_PrintString` para imprimir na tela a mensagem informando que o jogador deve escolher uma opção (*hit* ou *stand*).

4.12 p_PrintPlayerHand

Parâmetros: *a0* – quantidade de pontos do jogador.

Procedimento: chama `p_PrintString` e `p_PrintHand` passando o endereço do vetor de cartas do jogador para imprimir na tela as cartas do jogador e a quantidade de pontos correspondente.

4.13 p_PrintDealerHand

Parâmetros: *a0* – quantidade de pontos do *dealer*.

Procedimento: chama `p_PrintString` e `p_PrintHand` passando o endereço do vetor de cartas do *dealer* para imprimir na tela as cartas do *dealer* e a quantidade de pontos correspondente.

4.14 p_RevealDealerHand

Parâmetros: *a0* – quantidade de pontos do *dealer*.

Procedimento: chama `p_PrintString` e `p_PrintHand` passando o endereço do vetor de cartas do *dealer* para imprimir na tela a mensagem que revela as cartas do *dealer* e a quantidade de pontos correspondente.

4.15 p_PrintHand

Parâmetros: *a0* – quantidade de pontos. *a1* – endereço do vetor de cartas.

Procedimento: enquanto a próxima posição do vetor de cartas não for nula (i.e., 0), imprime a carta na posição chamando `p_PrintCard` seguida do sinal "+"; após, imprime o sinal "=" e a quantidade de pontos recebida como argumento.

4.16 p_PrintWinner

Parâmetros: *a0* – quantidade de pontos do jogador. *a1* – quantidade de pontos do *dealer*.

Procedimento: compara as quantidades de pontos recebidas como argumento e imprime a mensagem correspondente informando o vencedor da partida (ou se houve empate).

4.17 f_SumPoints

Parâmetros: *a0* – endereço do vetor de cartas.

Retorna: *a0* – quantidade de pontos correspondente às cartas.

Função: realiza a contagem dos pontos referentes às cartas presentes no vetor passado como argumento, seguindo as regras estipuladas: as cartas numeradas de 2 a 10 valem seu respectivo número, as cartas Valete, Dama e Rei (J, Q e K ou 11, 12 e 13, respectivamente) valem 10 pontos e o Às (1) pode valer 1 ou 11 pontos, sempre favorecendo o jogador.

4.18 p_ResetGame

Procedimento: chama `p_ClearVector` para restaurar todos os vetores de cartas.

4.19 p_ClearVector

Parâmetros: *a0* – endereço de um vetor de cartas. *a1* – tamanho do vetor de cartas.

Procedimento: percorre o vetor de cartas passado como argumento até o atingir o tamanho especificado, definindo o valor em cada posição para `0`.

4.20 p_PrintHeader

Procedimento: chama `p_PrintString` para imprimir o cabeçalho do programa.

4.21 p_PrintEndMenu

Procedimento: chama `p_PrintString` para imprimir o menu de final de jogo.

4.22 p_PrintCard

Parâmetros: *a0* – valor da carta.

Procedimento: imprime o símbolo correspondente ao valor da carta passado como argumento.

4.23 p_PrintInt

Parâmetros: *a0* – valor inteiro para exibir na tela.

Procedimento: carrega o registrador *a7* com o código correspondente e utiliza `ecall` para imprimir o valor em *a0*.

4.24 p_PrintString

Parâmetros: *a0* – endereço da *string* para exibir na tela.

Procedimento: carrega o registrador *a7* com o código correspondente e utiliza `ecall` para imprimir a *string* a partir do endereço em *a0*.

4.25 f_ReadInt

Retorna: *a0* – inteiro digitado pelo usuário.

Função: carrega o registrador *a7* com o código correspondente e utiliza `ecall` para ler um inteiro inserido pelo usuário e armazená-lo em *a0*.

4.26 Halt

Carrega os registradores *a0* com o *status* de saída (`0`) e *a7* com o código correspondente e utiliza `ecall` para chamar o sistema operacional e encerrar o programa.