Trabalho de Organização de Computadores

Integrante:
João Luís Almeida Santos – 20240002408

I. Introdução

Este relatório descreve o desenvolvimento de uma implementação do jogo Mancala em Assembly RISC-V, executada no simulador RARS, como parte das atividades da disciplina de Organização de Computadores. A proposta do trabalho foi de simular o jogo de tabuleiro Mancala em formato de terminal.

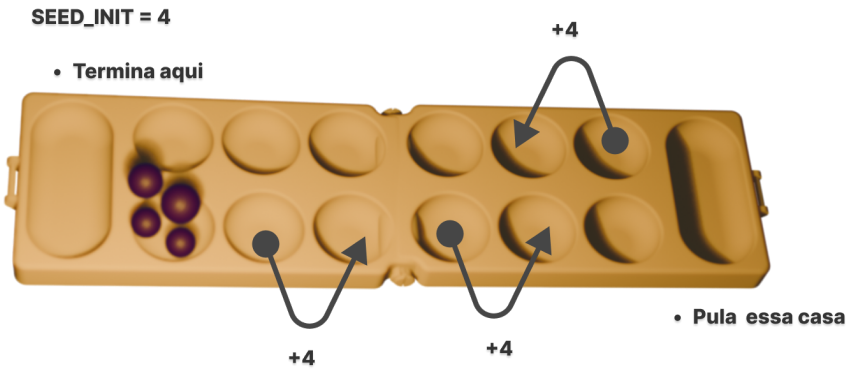

Figura 1: Representação do tabuleiro do Mancala

Este relatório descreve o desenvolvimento de uma implementação do jogo Mancala em Assembly RISC-V, executada no simulador RARS, como parte das atividades da disciplina de Organização de Computadores. A proposta do trabalho foi de simular o jogo de tabuleiro Mancala em formato de terminal.

Inicialização e Estrutura de Dados

Os primeiros passos do programa são dados na seção `.data`. Lá, são declaradas as variáveis, textos necessários para as funções de print, a vitória do jogador, o turno atual, etc. Além disso, todas as cavidades são inicializadas com o valor de 0, e a variável **SEED_INIT** é criada com o valor 4. Esta variável pode ser alterada para mudar a funcionalidade do jogo. Cada mensagem para o usuário/jogador foi colocada em asciz. Com essas linhas especificamente foi possível criar o formato formatado do tabuleiro. Vale mencionar que em vários pontos, eu coloquei esses textos dentro de rótulos no .text, onde poderiam ser acessados pela função **print** para printar valores como se fosse em um for loop. Isso me permitiu diminuir o

tamanho do arquivo no geral. O código no total deu 847 linhas, contando os comentários.

Função de Inicialização do Tabuleiro

A primeira função a ser chamada dentro do main é a de inicialização de tabuleiro. Ela é essencial para colocar os valores necessários dentro das cavidades para que o jogo efetivamente se inicie. Dentro dessa função, recebemos o número desejado em a0. Isso acontece apesar da existência do SEED_INIT. Significa que a função não lê diretamente o SEED_INIT. Ela recebe-o no início. Acredito que isso seja mais eficaz para caso queiramos mudar a lógica do tabuleiro de alguma forma.

Lógica do Loop e Armazenamento

De todo modo, a função prossegue. Ao receber o valor em a0, ela salva o valor em s0 para não perder em futuras chamadas de funções. Logo após, em **li s2, 5**, decidimos onde o loop vai parar enquanto estiver enchendo as cavidades. Não podemos chegar em 6 pois aí se localiza a cavidade de um dos jogadores. Iniciamos o Loop. Enquanto i não for igual a 5, continuamos. Chamamos a função auxiliar **armazena_cavidade** para armazenar o valor no endereço i. A função de armazenar cavidade foi útil para evitar ter que ficar repetindo endereço inicial + i * 4 para acessar endereços toda hora. Todo esse processo é demonstrado pela figura 1.

Uso da Memória

A memória foi dividida em duas seções principais. A **seção** `.data` contém variáveis globais, textos e vetores necessários para o jogo. A **seção** `.text` contém as funções principais e auxiliares. O código começa em `main`, inicializa o tabuleiro e executa o loop principal do jogo até o fim da partida.

Uso dos Registradores

Os registradores temporários (`t0-t6`) são usados para cálculos e comparações momentâneas. Os registradores salvos (`s0-s2`) armazenam valores persistentes entre chamadas de funções (ex.: valor inicial de sementes, contadores de loop). Os registradores de argumentos (`a0-a7`) seguem convenção padrão de chamadas. O registrador `sp` (stack pointer) é usado com as macros `startF` e `endF`, que salvam e restauram `ra`, `s0-s2`, garantindo integridade da pilha.

Funções Implementadas

- `main` — controla o fluxo principal do jogo, alternando entre os jogadores.
- `inicializar_tabuleiro` — preenche as cavidades com o valor de `SEED_INIT`.

- `mostra_tabuleiro` — imprime o estado atual do tabuleiro.
- `player_one_turn` / `player_two_turn` — controlam as jogadas, incluindo roubo e turno extra.
- `distribute_pellets` — distribui as sementes a partir da cavidade escolhida.
- `valida_escolha` — garante que a cavidade selecionada é válida.
- `carrega_cavidade`, `armazena_cavidade`, `adiciona_na_cavidade` — manipulam valores no vetor `cavidades`.
- `soma_valores_j1/j2` — somam o total de sementes de cada lado.
- `incrementa_vitoria_j1/j2` — atualizam o placar.
- `print`, `print_one_string`, `print_integer` — funções genéricas de saída via `ecall`.
- `read_integer` — leitura numérica do usuário.
- `verifica_vencedor` / `compara_valores` — determinam o resultado final da partida.