

Universidade Federal da Fronteira Sul

GEX1213 – Organização de Computadores

## Relatório Técnico

Jogo Mancala em Assembly RISC-V

**Integrante:**

João Luís Almeida Santos – 20240002408

Chapecó – SC

6 de novembro de 2025

# **Conteúdo**

<b>I. Introdução</b>	<b>2</b>
Demanda do Trabalho . . . . .	2
<b>II. Explicação do Código</b>	<b>3</b>
Inicialização . . . . .	3
Função: Inicializa tabuleiro . . . . .	5
<b>III. Conclusão</b>	<b>7</b>

## 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.

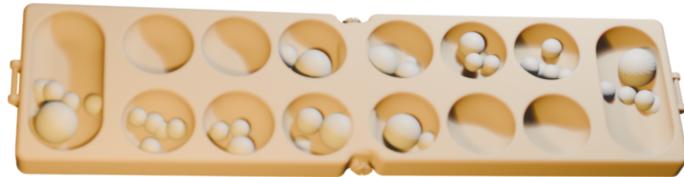


Figura 1: Representação do tabuleiro do Mancala (CrazyGames)

## Demanda do Trabalho

O enunciado disponibilizado exige uma versão do Mancala com doze casas e dois poços, quatro sementes por casa no estado inicial, suporte a turnos extras, captura de sementes e detecção do fim de jogo. Dentro do código, procurei modularizar e abstrair o máximo da lógica, dado o uso de funções de Macro e a criação de funções que vaziam processos simples, como printar, printar em loop, ler inteiro, etc.

## II. Explicação do Código

### Inicialização

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.

```
1      SEED_INIT:  
2          .word      4  
3  
4      vitorias_j1:  
5          .word 0  
6      vitorias_j2:  
7          .word 0  
8      turno_atual:  
9          .word 0  
10     cavidades:  
11         .word      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0  
12     SEED_INIT = 4;
```

Como visto na linha abaixo, cada mensagem para o usuário/jogador foi colocada em asciz. Com essas linhas especificamente foi possível criar o formato formatado do tabuleiro.

```
1  
2 # Informações dos jogadores  
3 titulo_jogador_1:  
4     .asciz      "Jogador 1\n"  
5 titulo_jogador_2:  
6     .asciz      "Jogador 2\n"  
7 texto_jogador_1:  
8     .asciz      "Escolha a cavidade [0-5]\n"  
9 texto_jogador_2:  
10    .asciz      "Escolha a cavidade [7-12]\n"
```

```

11
12 vitoria_jogador_1:
13     .asciz "Jogador 1 venceu!\n"
14 vitoria_jogador_2:
15     .asciz "Jogador 2 venceu!\n"
16 empate:
17     .asciz "Empate!\n"
18
19 texto_quer_jogar:
20     .asciz "Quer jogar novamente? 1 para sim, 0 para não... \n"
21
22
23 mensagem_valor_invalido:
24     .asciz      "Por favor escolha um valor válido!\n"
25 mensagem_roubo:
26     .asciz      "Roubou as pedras do adversário!\n"
27 mensagem_turno_extra:
28     .asciz      "Caiu na vala! Jogue de novo!\n"
29 mensagem_fim_jogo:
30     .asciz      "Fim de jogo! Um lado ficou vazio.\n"
31 # Textos do tabuleiro
32 titulo_acima_jogador_1:
33     .asciz
34     ↳   "                                0 <-- JOGADOR 1    5          \n"
35 titulo_acima_jogador_2:
36     .asciz
37     ↳   "                                12 <-- JOGADOR 2    7          \n"
38 linha_horizontal:
39     .asciz
40     ↳   "+----+----+----+----+----+----+----+----+----+----+----+----+\n"
41 linha_horizontal_meio:
42     .asciz
43     ↳   "----+----+----+----+----+----+----+----+----+----+----+----+"
44 quadrado_vazio:
45     .asciz      "|      | "
46 quadrado_esquerda:

```

```

43     .asciz    " | "
44     quadrado_direita:
45     .asciz    "   | "
46     quebra_linha:
47     .asciz    "\n"
48
49     .align    2
50
51

```

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: Inicializa 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 queríamos mudar a lógica do tabuleiro de alguma forma.

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 ai 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.

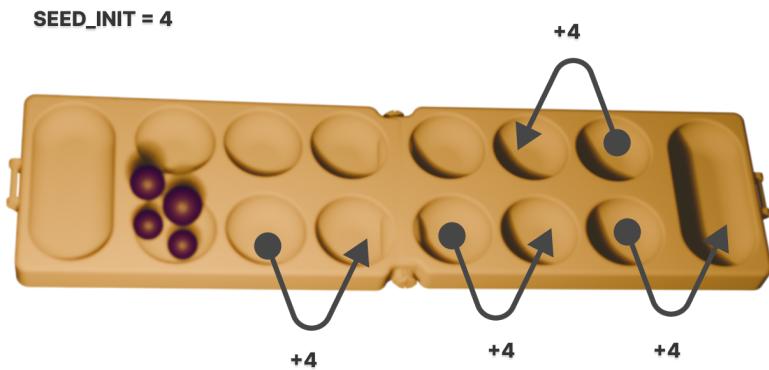


Figura 2: Demonstração visual da lógica de inicialização do tabuleiro

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 2.

```

1  inicializar_tabuleiro:
2    startF
3    # Supõe-se que o numero esteja em a0
4    # Isso pra caso queiramos tirar o SEED_INIT
5    mv      s0, a0                      # salva o valor inicial
6    ↪  (SEED_INIT)
7    li      s1, 0                      # contador/índice (s1 é
8    ↪  salvo)
9    li      s2, 5                      # max j1 (USA S2, que é
10   ↪  salvo, em vez de t0)
11   # li      t1, 12                  # Tente não usar em t1, deu
    ↪  problema
12
13  inicializar_tabuleiro_loop_j1:
14  # começa de 0 vai até 5

```

```

12      bgt      s1, s2, inicializar_tabuleiro_skip_cavidade # Compara
13          ↵ com s2
14      mv       a0, s1                      # índice
15      mv       a1, s0                      # valor a armazenar
16      call     armazena_cavidade
17      addi    s1, s1, 1                   # incrementa contador
18      j       inicializar_tabuleiro_loop_j1
19
20      inicializar_tabuleiro_skip_cavidade:
21          li      s1, 7                  # reinicia contador para j2
22          j       inicializar_tabuleiro_loop_j2
23      inicializar_tabuleiro_loop_j2:
24          li      t1, 12
25          bgt    s1, t1, end_inicializar_tabuleiro
26          mv       a0, s1                      # índice
27          mv       a1, s0                      # valor a armazenar
28          call     armazena_cavidade
29          addi    s1, s1, 1                   # incrementa contador
30          j       inicializar_tabuleiro_loop_j2
31      end_inicializar_tabuleiro:
32          endF
33          ret

```

### III. Conclusão