

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

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

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.

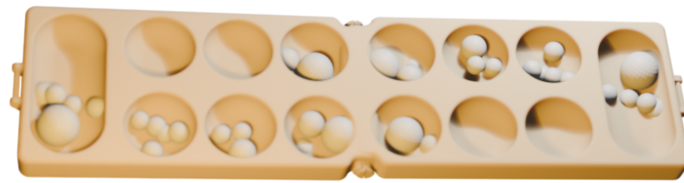


Figura 1: Representação do tabuleiro do Mancala (CrazyGames)

Demanda do Trabalho

O enunciado disponibilizado exige uma versão do Mancala com doze cavidades 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, 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 vitoria_jogador_1:
12     .asciz "Jogador 1 venceu!\n"
13
14 vitoria_jogador_2:
15     .asciz "Jogador 2 venceu!\n"
16
17 empate:
18     .asciz "Empate!\n"
19
20 texto_quer_jogar:
21     .asciz "Quer jogar novamente? 1 para sim, 0 para não... \n"
22
23 mensagem_valor_invalido:
24     .asciz "Por favor escolha um valor válido!\n"
25
26 mensagem_roubo:
27     .asciz "Roubou as pedras do adversário!\n"
28
29 mensagem_turno_extra:
30     .asciz "Caiu na vala! Jogue de novo!\n"
31
32 mensagem_fim_jogo:
33     .asciz "Fim de jogo! Um lado ficou vazio.\n"
34
35 # Textos do tabuleiro
36 titulo_acima_jogador_1:
37     .asciz
38 ↪ "                                0 <-- JOGADOR 1    5                                \n"
39
40 titulo_acima_jogador_2:
41     .asciz
42 ↪ "                                12 <-- JOGADOR 2    7                                \n"
43
44 linha_horizontaal:
45     .asciz
46 ↪ "+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---\n"
47
48 linha_horizontaal_meio:
49     .asciz
50 ↪ "----+---+---+---+---+---+---+---+---+---+---+---+---+"
51
52 quadrado_vazio:
53     .asciz "|      |"
54
55 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 imprimir 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 queiramos 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 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`.

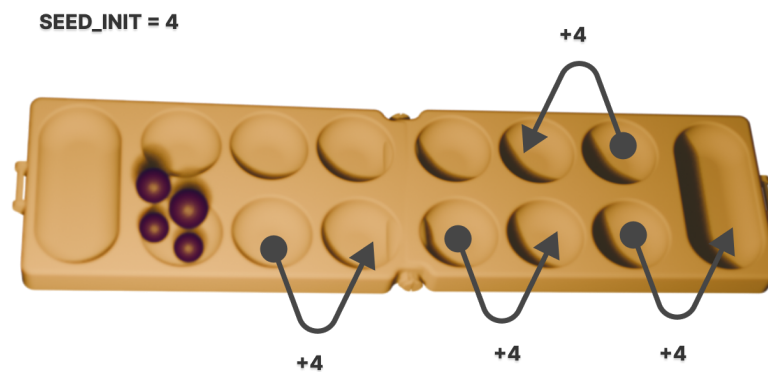


Figura 2: Demonstração visual da lógica de inicialização do tabuleiro

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

```

```

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

```

III. Conclusão