UNIVERSIDADE DE SÃO PAULO (USP)

INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO (ICMC)

DEPARTAMENTO DE CIÊNCIAS DE COMPUTAÇÃO (SCC)

Gabriel Costa - 14785489

Isabella Arão - 9265732

Marina Fagundes - 9265405

Torre de Hanói

São Carlos

2023

Gabriel Costa - 14785489

Isabella Arão - 9265732

Marina Fagundes - 9265405

Torre de Hanói

Relatório apresentado à disciplina Organização e Arquitetura de Computadores, como parte dos requisitos para aprovação na matéria oferecida pela Universidade de São Paulo, na área de Computação.

Prof.: Vanderlei Bonato

Disciplina: Organização e Arquitetura de Computadores

Turma: Turma SSC 0513

São Carlos

2023

1. Introdução

O presente relatório foi desenvolvido para a disciplina de Organização e Arquitetura de Computadores do curso de Sistema de Informação. O programa implementa o jogo Torre de Hanói, na linguagem *Assembly*, para ser executado em um simulador do processador MIPS, o *MARS MIPS Simulator*.

2. Torre de Hanói

A Torre de Hanói é um quebra-cabeça, foi inventado pelo matemático francês Édouard Lucas em 1883, inspirado em uma lenda sobre um templo Hindu que teria sido construído no centro do universo a partir da movimentação de discos sobre pinos.

O problema consiste em uma base que contém três pinos. Um desses pinos dispõe de alguns discos, que devem ser movidos para os pinos vizinhos.

O objetivo do jogo é reordenar todos os discos em um pino vizinho, mas seguindo algumas regras básicas:

1) Um disco maior não pode ser colocado sobre um menor;

2) A movimentação deve ser feita com apenas um disco por vez.

O quebra-cabeça, portanto, é encontrar a sequência de movimentos que permita reorganizar todos os discos do pino de origem para um pino de destino, utilizando-se de um pino intermediário.

3. Planejamento

Cronograma				
Produto	**Sub tarefa**	**Explicação/Questões a serem respondidas**	**Situação**	**Data limite**
Relatório	Introdução	Explicação breve do trabalho		15/11
	Torre de Hanoi	Explicar o que é Torre de Hanoi		15/11
	Implementação	Apresentar a implementação do código no relatório		15/11
	Conclusão	Dificuldades, aprendizado, imagens do programa executando		15/11
	Bibliografia	ABNT		15/11
Código	-	-		11/11

	Quais funções faltam?	**Situação**
Funções	Movimento	
	Torre Vazia	
	Torre Cheia	
	Movimento inválido	
	Main	
	Imprimir torre	
	Lê torre	
	Valida entrada	
	Entrada inválida	
	Fim	

Seguindo o planejamento que foi elaborado para a entrega intermediária (tabela 01), elencamos as funções que seriam necessárias para o desenvolvimento do jogo (tabela 02). Apesar das dificuldades, que serão listadas no tópico 4, conseguimos manter o cronograma original e terminar o código dentro do prazo estipulado desde o início.

4. Desenvolvimento

A princípio, começamos a desenvolver a estrutura das torres com uma pilha. Seguindo essa lógica, quando um disco era desempilhado da torre original, ele seria empilhado na torre de destino e assim por diante. Porém, encontramos muitas dificuldades no desenvolvimento dessa proposta e não conseguimos achar muitos materiais de suporte na internet.

A principal dificuldade foi porque precisaríamos criar três pilhas: de origem, auxiliar e de destino, mas existe apenas um registrador *stack pointer* ($sp). Dessa forma, precisaríamos simular $sp com outros registradores, o que fez com que a pilha já não fosse a melhor solução.

Apesar de ter muitas implementações de Torre de Hanói disponíveis na internet, nenhuma tinha o nível de complexidade necessária para o desenvolvimento do jogo com a interação do usuário.

Então, começamos o código do zero novamente, visando ter três vetores, que representariam cada uma das torres. Inicialmente, um vetor estaria cheio (torre de origem) e os outros dois vetores estariam vazios e poderiam receber os discos. Mesmo após muitas tentativas, não conseguimos validar com efeito os movimentos, então mudamos parcialmente nossa lógica de desenvolvimento.

Na versão final, o código conta com apenas um vetor, que contém as três torres, com 15 posições, sendo que as 5 primeiras estão ocupadas no início do jogo (representando a torre de origem) e as outras posições estão zeradas (representando as torres auxiliar e de destino, respectivamente). Assim, logo no início do código, é possível observar que a torre de origem ocupa as posições 0 a 16 bits do vetor, a torre auxiliar ocupa as posições 20 a 36 e, por último, a torre de destino ocupa as posições 40 a 56.

Além disso, as posições da torre de origem estão numeradas da seguinte forma ao iniciar o jogo: 5 a 1, indicando por 5 o maior disco (que está na base da torre de origem) e 1 o menor disco (está no topo da torre de origem).

A partir dessa lógica, conseguimos implementar uma série de funções, que fazem as verificações necessárias para validar os movimentos e atualizar os “topos” de cada torre (posições específicas no vetor). As funções fazem as seguintes análises:

1) leitura de cada uma das torres;

2) validação da entrada digitada pelo usuário (se o valor digitado é 1, 2 ou 3, indicando uma das torres);

3) verificação dos topos de cada uma das torres (se estão vazias ou cheias);

4) verificação para descobrir se o movimento é inválido (tentativa de empilhar um disco maior acima de um menor que já está na torre de destino);

- **a)** consideramos que movimentar um disco de uma torre para a mesma torre também se configura como movimento inválido;

5) se o movimento não é inválido, realiza o movimento e muda os discos da torre de origem para a torre de destino;

6) impressão de mensagens de erro para auxiliar o usuário;

7) impressão na tela das torres e seus respectivos discos para permitir que o usuário visualize o andamento do jogo;

Na *main*, temos:

1) definição dos topos de cada torre para início do jogo;

2) *loop* principal, referente à partida:

- **a)** *loop* secundário para leitura da torre de origem (realiza as verificações necessárias);
- **b)** *loop* secundário para leitura da torre de destino (realiza as verificações necessárias);

c) movimentação dos discos entre as torres;

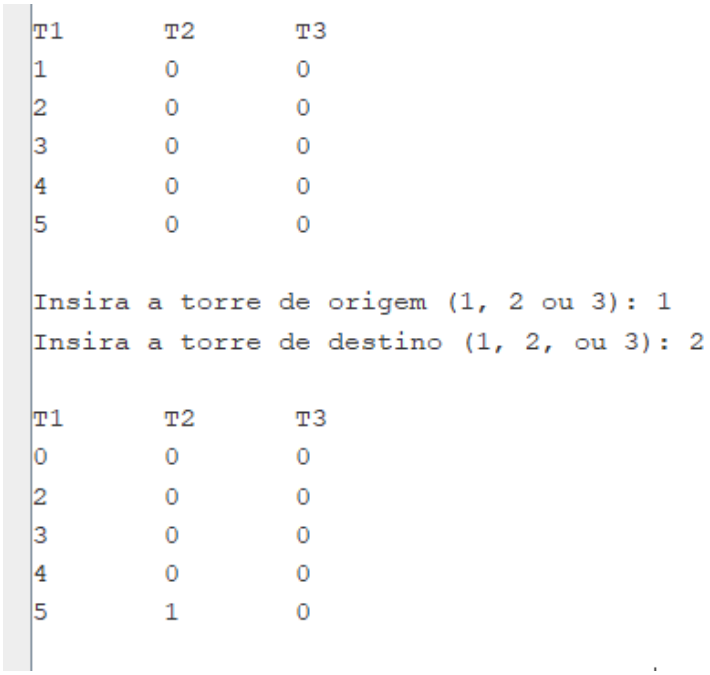
d) atualização dos topos de cada torre após cada movimento;

Com essa nova lógica de implementação conseguimos seguir o código e fazer com que o usuário consiga fazer as jogadas continuamente até terminar o jogo.

Em anexo, disponibilizamos o código com os comentários necessários para melhor entendimento do que foi desenvolvido.

A seguir, apresentamos algumas imagens do jogo em execução. Na apresentação, demonstraremos uma partida.

Figura 01 - Primeira jogada e atualização das torres.



Fonte: Elaborada pelos autores

Figura 02 - Caso teste (entrada inválida).

```
Insira a torre de origem (1, 2 ou 3): 4

A entrada deve ser 1, 2 ou 3

Insira a torre de origem (1, 2 ou 3):
```

Fonte: Elaborada pelos autores

Figura 03 - Caso teste (torres iguais).

```
Insira a torre de origem (1, 2 ou 3): 1
Insira a torre de destino (1, 2, ou 3): 1

Movimento invalido. Escolha as torres novamente
```

Fonte: Elaborada pelos autores

Figura 04 - Caso teste (disco maior sobre menor).

```
T1      T2      T3
0       0       0
2       0       0
3       0       0
4       0       0
5       1       0

Insira a torre de origem (1, 2 ou 3): 1
Insira a torre de destino (1, 2, ou 3): 2

Movimento invalido. Escolha as torres novamente
```

Fonte: Elaborada pelos autores

5. Conclusão

Em suma, tivemos muitas dificuldades ao longo do processo de desenvolvimento do código, principalmente no início, pois despendemos bastante tempo ao tentar implementar a pilha. Após isso, tivemos que recomeçar do zero, nos guiando pelos poucos materiais que encontramos na internet, para nos auxiliar no processo de construção do código na linguagem *Assembly*, à qual não estávamos acostumados.

O processo foi de muita aprendizagem para todos os integrantes do grupo. E, apesar das dificuldades, ficamos satisfeitos com o resultado final.

6. Referências bibliográficas

BONATO, V. Material didático apresentado na disciplina SCC0513 - Organização e Arquitetura de Computadores. São Carlos/SP. s.d.;

TORRE de Hanói, jun. 2017. Disponível em: <https://pt.wikipedia.org/wiki/Torre_de_Han%C3%B3i>. Acesso em: 09 out. 2023.