Ana Luíza Moreira Silva
Eric Rhaysllier Silva
Jean Luc Girvent Deu
Pedro Otávio Setulim Silva
Matheus Martins de Resende

MEMORIAL DESCRITIVO
Prática 01 – Laboratório MIPS Assembly

Uberlândia, 2025

Sumário

1.Introdução

Esta atividade prática avaliativa tem como objetivo incentivar o aprendizado de como funciona uma linguagem *low-level* usando da ferramenta MARS e, subsequentemente, a linguagem Assembly. O principal desafio é o de entender e escrever lógica e programar códigos em um ambiente mais difícil do que a maioria das linguagens apresentam e, por isso, acreditamos que será a maior dificuldade em si de se acostumar com os comandos do Assembly.

2.Estrutura do Grupo

O grupo é formado pelos seguintes membros:

- Ana Luíza Moreira Silva – 12411BCC053
- Eric Rhaysllier Silva – 12411BCC107
- Jean Luc Girvent Deu – 12411BCC101059
- Pedro Otávio Setulim Silva – 12411BCC
- Matheus Martins de Resende – 12411BCC019

Apesar de não haver separação de tarefas, todos nos mantemos em comunicação, por canais como *Discord* ou *Whatsapp*, e sempre ajudando uns aos outros com o que aprendíamos fazendo os códigos. Todos tiveram participação em cada um dos códigos anexados.

3.Soluções Propostas

Dado o desafio que é escrever uma linguagem *low-level* como Assembly, escolhemos a estratégia de usar uma outra linguagem que temos mais conhecimento e prática que, apesar de ser considerada de nível alto, é bem próxima de um baixo que é a Linguagem C.

3.1.Prática 1

Dada a prática 1 sendo para fazer uma cifra de César, o código foi feito inicialmente em C, o que não demorou muito para fazer na lógica dada na explicação. Ao passar para Assembly, lê a string, isto é, o *array* de caracteres e ao encontrar o símbolo “/0”, para a leitura e codifica a string dada no 1º Loop e decodifica quando passa pelo 2º Loop.

3.2.Prática 2

Para o código da prática 2, o cálculo da série harmônica foi feito com um Loop e, dentro dele, convertia o valor do incrementador para ponto flutuante e realizava a soma em diante, após isso incrementa o contador e testa se é menor que o valor de n. O que foi um código menor e mais simples que o anterior.

3.3.Prática 3

Na prática 3, teve a questão de que, para fazer o cálculo de Phi de Euler, precisávamos primeiro fazer uma função de MDC para continuar, o que foi relativamente não tão complexo ao implementar uma função recursiva para definir o MDC dos números. Após isso, utilizou um Loop para implementar o código dado como exemplo na própria questão, somente traduzindo ele de C para Assembly com um contador e a função MDC que agora já tínhamos.

3.4.Prática 4

O código da prática 4 já foi um pouco mais complexo. A partir do exemplo doado, ele foi feito em C e depois traduzido para Assembly. A ideia foi de fazer uma função para calcular a média e depois outra função para fazer a correlação dos elementos. Na função de média, tem um loop que pega cada um dos elementos e que faz também a soma deles para retornar a conta da média. Na função de correlação, calcula a média de cada uma das listas, depois faz os somadores e por fim divide os resultados e se der:

- 1 : correlação perfeita positiva,
- -1: perfeita negativa,
- 0 : sem correlação linear.

3.5.Prática 5

A prática 5, apesar de meio confusa, foi feita com o pensamento do movimento que o texto tem que fazer dentro do vetor, que no caso é uma matriz mas é usado como um vetor que as linhas são estruturadas em row-major-order, e da tela. No caso, isso foi feito com o auxílio de diversos loops, com a maioria aninhados, assim pegando cada parte da matriz e fazendo o movimento em espiral. Esses loops consistem em pegar o tamanho da String dada e separar em vetores menores para caber no tamanho da espiral.

3.6.Prática 6

Igualmente a prática 5, a sexta foi feita usando um código parecido, porém mudando alguns loops para caber no formato do zig-zag. A estratégia foi a mesma de usar a matriz inicial e ir moldando ela no novo formato, agora mais facilmente levando em conta que a palavra termina em uma linha e começa em outra para fazer o formato desejado.

3.7.Prática 7

Para a prática 7, a conta por trás de Bhaskara foi implementado inicialmente em C e depois fomos estruturar em Assembly como ficaria. Dessa vez não foi utilizado Loops, já que a conta é mais linear. Após pegar os números, é calculado o delta, que no caso é uma função a parte que somente retorna o resultado, e, em seguida, calcula as duas raízes e devolvem elas para o usuário.

3.8.Prática 8

Os códigos das próximas práticas são mais complexos, dito isso, a prática 8 foi feito da mesma forma, isto é, fazer em Linguagem C e depois passar para Assembly, mesmo assim foi um código complexo mesmo em C. O código em si foi feito com dois Loops, um para gerar os 10 elementos nas listas *x* e *y* e outro para fazer a soma dos elementos da lista (os pontos no plano), com isso fazendo as somatórias dadas na questão e resultando nos valores que forem apresentados na tela.

3.9.Prática 9

Na prática 9, para calcular o resultado do e^x, tivemos que ter cuidado ao mostrar o resultado e não nos depararmos com overflow, dito isso o código resultante faz um somatório igual ao que foi dado no enunciado e calculando o termo anterior ao invés dos fatoriais e, com o resultado, entrava num Loop e testava se o resultado do termo, que é a precisão, menor que 0,00001, assim definindo se saí do Loop. No final é mostrado o resultado do somatório.

3.10.Prática 10

Primeiramente, o código lê um valor em Radianos e converte para Graus para poder fazer as contas. Em seguida, o código é parecido com o do anterior (prática 9), logo é usado ele para fazer as contas do seno e do cosseno com a mesma precisão de 0,00001, assim dando o resultado aproximado de seno e cosseno de x com a série de Taylor.

4.Estratégias

A metodologia seguida na prática, assim como foi dito nos tópicos anteriores, foi de usar a Linguagem C como base para entender a lógica por trás dos enunciados e depois desenvolver o código em Assembly. Além de também, ao fazer em grupo, nos ajudamos a fazer o trabalho, o que fez conseguirmos terminar mais rápido.

5.Testes Realizados

Muitos dos testes realizados foram com base nos exemplos dados em cada um dos enunciados. Os que não tinham, fazíamos um exemplo fácil e com casos que poderiam gerar erros, como overflow, e testávamos logo em seguida.

Outros testes triviais foram quanto tentávamos testar funções auxiliares, como a do MDC ou Delta, para vermos se o resultado era o esperado. Assim não nos deparávamos com overflows ou bugs que passassem despercebidos.

6.Experiência Vivenciada e Conclusão

Tendo finalizado esta prática, percebemos que muitas das dificuldades compartilhadas foram de se acostumar com a escrita e comandos do Assembly. Como temos maior experiência com linguagens *high-level*, então escrever códigos numa linguagem como essa foi bem desafiador. Porém, com a ajuda de nossas estratégias de usar a Linguagem C e marcar diversos encontros para discutir sobre a codificação, foi muito mais fácil e eficiente a escrita das práticas.

Ainda, foi muito bom passarmos por esses desafios. Isso nos mostra o quanto as linguagens de programação evoluíram nos últimos tempos e também nos ajuda a entender como um sistema é arquitetado e como seus processos ocorrem no nível baixo da máquina.