INSTITUTO FEDERAL DO ESPÍRITO SANTO
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

Gustavo Firme Fiorot
Lucas Mariani Gomes
Rafael Almeida Deps Caldeira
Rodrigo Lyrio Rodrigues
Ronnald Willian Silva de Santana
Vivian Moreira Gomes de Lacerda

Resolvendo problemas em C e Assembly

ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES
PROF. FLÁVIO GIRALDELI BIANCA

SERRA
2025

1. INTRODUÇÃO

O presente relatório tem como objetivo comparar as implementações em C e Assembly de três problemas que serão apresentados a seguir. A partir deles, pôde-se destacar os aspectos da linguagem de baixo nível que permanecem ocultos na linguagem C, bem como a diferença na quantidade de instruções de cada implementação e o funcionamento básico dos procedimentos/macros provenientes do arquivo **inc\emu8086.inc**.

2. DESENVOLVIMENTO

2.1 Problema 1

No problema 1, o objetivo foi implementar um programa que lê três valores distintos e calcula a soma dos dois maiores valores, utilizando as linguagens C e Assembly.
Na implementação em C, foram utilizadas estruturas condicionais para identificar o menor valor entre os três números fornecidos pelo usuário. Após a determinação do menor valor, os dois maiores são somados e o resultado é exibido. Ou seja, a lógica foi baseada apenas em comparações diretas entre as variáveis.
Já na implementação em Assembly, os valores foram armazenados em registradores específicos e comparações foram feitas utilizando instruções como ‘CMP’ e saltos condicionais (‘JG’, ‘JL’) para identificar o menor valor. Após a determinação dos dois maiores valores, a soma foi realizada utilizando-se a instrução ‘ADD’, cujo resultado foi armazenado em uma variável. Assim, utilizou-se o ‘Print_num’ para exibir o resultado, convertendo os números para formato ASCII.

2.2 Problema 2

No problema 2, o objetivo consistiu em simular o movimento de dois carros que partem de posições iniciais distintas, com velocidades diferentes, deslocando-se na mesma direção. Durante a simulação, as posições dos carros são exibidas a cada hora até que um deles ultrapasse o outro.

Na implementação em linguagem C, utilizou-se um laço *while*, juntamente com operações aritméticas simples, para atualizar as posições dos veículos. Já na implementação em linguagem Assembly, foi necessário armazenar os dados do carro que está à frente em registradores específicos e os dados do carro que está atrás em outros registradores distintos. Isso permitiu a realização de um laço utilizando a instrução *JMP*. Adicionalmente, foi implementado um verificador após cada operação

de soma para identificar possíveis erros de *overflow*. Outro aspecto verificado foi se o carro que estava atrás efetivamente ultrapassava o carro à frente durante a simulação.

2.3 Problema 3

O Problema 3 consistiu em receber três valores correspondentes aos lados de um triângulo e determinar, por meio de processamento, se é possível formar um triângulo com tais medidas. Na implementação em C, foi utilizada uma função simples que emprega uma única estrutura condicional (if-else) para retornar 1 ou 0. Em contrapartida, na solução em Assembly, foi necessário utilizar diversos parâmetros para armazenar as variáveis relativas aos lados e suas somas, visto que a validade de um triângulo depende de a medida de cada lado ser menor que a soma dos outros dois lados. Adicionalmente, foram implementadas verificações após cada operação de soma para identificar possíveis erros de *overflow*, bem como para assegurar que os valores atribuídos aos lados sejam maiores que 0.

2.4 Processamentos Utilizados

PRINT_NUM_UNS: responsável por converter e imprimir um número contido no registrador AX em sua representação decimal. Ele utiliza um algoritmo de divisão sucessiva para extrair os dígitos do número dividindo o valor por 10 e obtendo o resto, que corresponde a um dígito na ordem inversa. Cada dígito é então convertido para o seu código ASCII (por meio da adição de 30h) e enviado para a saída utilizando o macro PUTC (explicado mais pra frente). O procedimento também trata casos especiais, como quando o número é zero ou negativo, garantindo que a saída esteja formatada corretamente.

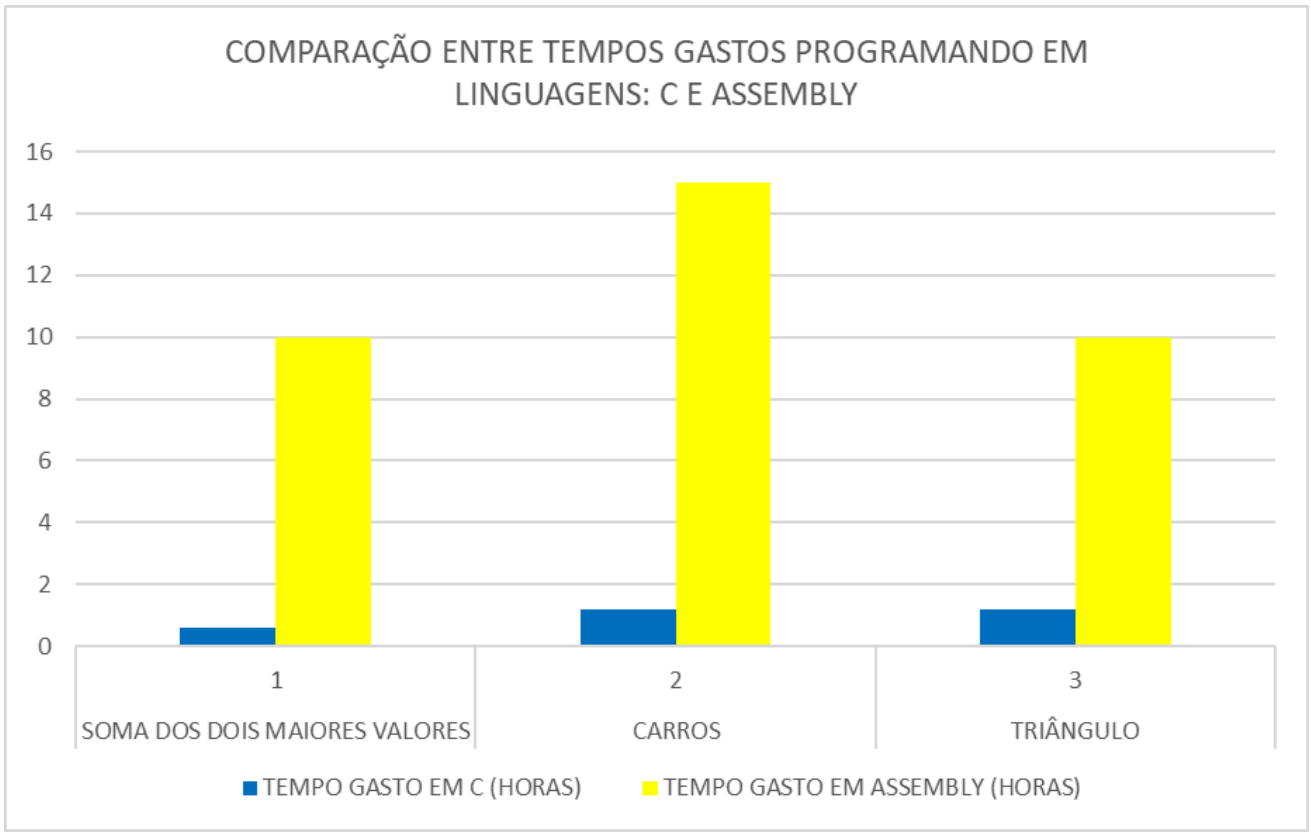
SCAN_NUM: É outro procedimento frequentemente copiado do inc\emu8086.inc e utilizado para a leitura de números a partir do teclado. Ele implementa a lógica de leitura dos caracteres digitados pelo usuário, utilizando a interrupção de teclado (INT 16h) para capturar cada entrada, convertendo os caracteres ASCII em seus valores numéricos e acumulando-os num valor inteiro por meio de multiplicação e soma.

2.5 Macros Utilizados

PUTC (Print Character): Foi desenvolvido para imprimir um único caractere na tela. Ele recebe como parâmetro um valor que representa o código ASCII do caractere a ser exibido e, ao ser expandido durante a compilação, gera as instruções necessárias para colocar esse valor em um registrador e invocar a interrupção de saída (geralmente INT 21h com a função correspondente) para exibir o carácter no cursor atual.

2.6 Tempo Gasto

Um fator de grande diferenciação entre a experiência de programar em Assembly comparado com a programação em C, foi o tempo gasto, essa divergência ocorre tanto pelo fato de estarmos acostumados a programar na linguagem C quanto ao fato de essa ser a nossa primeira experiência prática em uma linguagem de baixo nível, isso fica evidente principalmente na segunda atividade, pois ela foi a primeira que tivemos que montar o código do zero em ambas as linguagens (tendo em vista que na primeira nós possuímos uma versão do python para usar de referência), sendo que o tempo gasto para o programa em C foi de aproximadamente 1 hora enquanto para o programa em Assembly foram gastos 15 horas , o tempo gasto no primeiro problema em C foi de 40 minutos e em Assembly foi de 10 horas, e na última tarefa foi gasto aproximadamente 1 hora em C enquanto na linguagem de baixo nível foram 10 horas.
É relevante se notar que o nosso tempo entre o segundo e o terceiro código em assembly foi menor (diferença de 5 horas), isso se deve ao fato de que com o passar do tempo o nosso “estranhamento” inicial com uma nova linguagem acaba passando, além de que acontece o ganho de uma certa experiência assim evitando erros e com isso conseguindo resolver os problemas de maneira mais rápida.



2.7 Diferenças entre C e assembly

Em C, são utilizadas funções de entrada e saída padrão (printf/scanf) da biblioteca (“<stdio.h>”) e estruturas de controle (if/else). Essa abordagem de alto nível abstrai detalhes do hardware, como manipulação de registradores, comparações e gerenciamento da pilha. O compilador converte cada operação declarada em C em uma ou mais instruções de Assembly, de modo que um simples if (a < b) oculta diversas operações de movimentação de dados (MOV) e comparações que, em Assembly, precisam ser explicitamente escritas.
Já em Assembly, cada operação deve ser detalhadamente implementada, incluindo leitura de dados (scan_num), exibição de mensagens (sprint e print_num) e comparações (cmp, jg, jl). Assim, uma estrutura condicional que em C é expressa em uma única linha pode se traduzir em diversas instruções Assembly, exigindo maior atenção ao uso de registradores e endereçamento de memória.
A diferença na quantidade de instruções é notável: enquanto a versão em C o programa 1 ocupa 24 linhas, a implementação em Assembly requer 375 linhas para realizar a mesma tarefa. Isso evidencia como C permite a construção de um código mais compacto e legível, enquanto Assembly demanda uma escrita detalhada e minuciosa, tornando a leitura e compreensão do código mais complexas.

3 CONCLUSÃO

O presente trabalho nos permitiu realizar uma análise entre os códigos em C e Assembly, possibilitando notar as principais diferenças entre ambas as linguagens. Dessa forma, os resultados evidenciaram que, enquanto em C é possível escrever códigos mais compactos e legíveis, Assembly exige uma atenção muito maior, principalmente no que tange ao gerenciamento dos registradores, memória e instruções específicas. Essa diferença ficou clara ao compararmos, as quantidades de linhas e quantidade de tempo que em assembly demoraram múltiplas vezes mais que reflete a curva de aprendizado à linguagem.
A aprendizagem apesar de difícil e complexa nos permitiu aprender mais sobre como o computador entende os códigos de nível mais alto, principalmente como registradores são usados, como comparações são feitas e como jumps servem para os loops. Além disso, foi possível pôr em prática os conhecimentos de Arquitetura e Organização de Computadores.