INSTITUTO FEDERAL DO ESPÍRITO SANTO
CAMPUS SERRA
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

ARTHUR RECCO, FERNANDO MONTENEGRO, ILANNA CARDOSO, LUCAS STEFANELLI, RHENNER SANTOS

RESOLVENDO PROBLEMAS EM C E ASSEMBLY

Serra
2025

ARTHUR RECCO, FERNANDO MONTENEGRO, ILANNA CARDOSO, LUCAS STEFANELLI, RHENNER SANTOS

RESOLVENDO PROBLEMAS EM C E ASSEMBLY

Relatório Técnico-Científico apresentado à disciplina de Arquitetura e Organização de Computadores do Curso de Bacharelado em Sistemas de Informação do Instituto Federal do Espírito Santo, como requisito parcial para avaliação.

Orientador: Prof. Flávio Giraldeli

Serra
2025

SUMÁRIO

1 INTRODUÇÃO

Este relatório tem como objetivo principal expor a experiência do grupo na realização das atividades da disciplina de Arquitetura e Organização de Computadores. O propósito do trabalho é a comparação entre as linguagens e suas principais diferenças. Inicialmente, foram apresentados três problemas a serem resolvidos em linguagens de alto e baixo nível; para isso, o grupo utilizou as linguagens C e Assembly, respectivamente. A implementação em Assembly foi feita utilizando o *assembler* EMU8086, enquanto para a codificação em C foi desenvolvida pelo *Visual Studio Code*, uma IDE (*Integrated Development Environment*), em conjunto com o compilador GCC (*GNU Compiler Collection*).

Os problemas a serem resolvidos consistem em programas simples, cujo objetivo da implementação é justamente reforçar conceitos básicos de programação, só que em Assembly.

O primeiro problema envolve as competências de: comparações, desvios condicionais, operações aritméticas e *input* / *output* de caracteres. Buscando facilitar, o problema deu liberdade ao grupo de escolher um outro problema dentre os que já haviam resolvido na disciplina de Programação I, matéria do 1º período do curso de Sistemas de Informação. Dessa forma, o código seria necessariamente curto e simples, utilizando poucas funções de cada linguagem.

Já o segundo problema envolve as competências do anterior, com o adicional de *loops* (iterações/comandos de repetição). Seu objetivo é testar a habilidade do grupo de utilizar os *jumps* condicionais do Assembly para alcançar a "mesma" funcionalidade das funções "`if`" e "`for`" da linguagem C, em busca de verificar se um número *n* é ou não primo e, se não for, imprimir seus divisores na tela, com: "`<N> eh primo`" ou "` <N> não eh primo e tem como divisores: <conjunto de divisores>`".

Por fim, o terceiro problema envolve todas as competências anteriores, com mais a utilização de procedimentos. Sua funcionalidade é realizar a soma dos termos de uma Progressão Aritmética (PA), através da fórmula:

$$S_n = \frac{(a_1 + a_n) * n}{2}$$

, onde a soma dos termos (S_n) é igual à soma do primeiro termo (a_1) com o último termo (a_n), multiplicando o resultado pela quantidade termos (n) e, por fim, dividindo tudo por 2. Em outras palavras, essa fórmula representa a conhecida soma de Gauss.

O cálculo do resultado da fórmula deve ser feito exclusivamente dentro de uma função (em C) / procedimento (em Assembly), retornando o valor para o código principal, que deverá imprimí-lo na tela.

2 PRINCIPAIS DIFERENÇAS ENTRE C E ASSEMBLY

Durante a construção dos códigos em C e Assembly, foi possível observar grandes diferenças na complexidade de cada algoritmo. O código em Assembly demonstrou ser consideravelmente mais difícil de desenvolver, exigindo um número significativamente maior de passos em comparação com o código em C. Dessa forma, o trabalho apresenta 93,8% de Assembly e apenas 6,2% foi feito em C.

Dada a liberdade de escolha do problema 1, o grupo decidiu fazer um cálculo da média de um aluno a partir de 3 notas fornecidas pelo usuário e imprimir a sua situação – aprovado ou reprovado. No começo do trabalho já foi possível observar as diferenças entre C e Assembly, pois foi necessário dedicar registradores para guardar as informações.

Quanto ao uso de variáveis, precisou-se declarar um tamanho para elas em *byte* ou *word*. Além disso, as mensagens em formato *string* não podiam ser escritas diretamente no "*print*", elas precisavam ser armazenadas em uma variável de antemão. Para passar a mensagem para o registrador, necessitou-se o uso do comando "`LEA`" que passa o endereço, semelhante a como um ponteiro funciona em C.

A parte mais difícil foi a de aceitar um *input* do usuário e imprimir o número da média na interface. Porém, para isso foi copiado do `inc\emu8086.inc` um procedimento para escanear números e outro para imprimir eles na tela (`SCAN_NUM`). Para pular a linha no terminal, foi utilizado uma *macro*. Na implementação das contas, surgiram algumas dúvidas sobre a manipulação das partes baixas e altas dos registradores, mas para simplificar priorizou-se utilizar o registrador todo nas operações aritméticas.

No problema 2 foram introduzidos o uso de estruturas de condicional e *loop*. Em C foi mais simples, pois podia-se utilizar o for e já especificar onde a variação da iteração começava e parava. Já em Assembly foi necessário usar um *loop*, um registrador para iterar e outro como número limite da iteração. Para sair do *loop*, foi utilizado um *jump* que verifica se o valor incrementado do registrador excedeu o valor limite.

O uso de procedimentos foi o diferencial na solução do problema 3. Conforme solicitado, o grupo criou um procedimento dedicado exclusivamente ao cálculo da soma da Progressão Aritmética (PA). A implementação desse procedimento em Assembly não apresentou grande complexidade, pois os valores dos registradores modificados dentro do procedimento se mantiveram válidos para o restante do código. Dessa forma, não houve necessidade de implementar a passagem de parâmetros ou o retorno de um valor pelo procedimento.

A análise das linguagens revela uma diferença significativa na complexidade das instruções utilizadas. Em C, uma linguagem de alto nível, as instruções essenciais foram `printf`, `scanf`, `if`/`else`, `for`, `return` e `func()`. Por outro lado, em Assembly, uma linguagem de baixo nível, a lista é extensa, incluindo: `org`, `lea`, `mov`, `int`, `call`, `xor`, `div`, `add`, `cmp`, `jge`, `jmp`, `push`, `je`, `jae`, `pop`, `jc`, `sub`, `neg`, `jnz`, `jz`, `jb`, `ja`, `macro`, `proc` e `ret`.

Essa disparidade evidencia que a implementação de um algoritmo simples exigiu um número consideravelmente maior de passos em Assembly quando comparada à sua equivalente em C.

3 EXPLICAÇÃO DOS PROCEDIMENTOS BÁSICOS DO INC\EMU8086.INC

Como já foi mencionado anteriormente, para facilitar o desenvolvimento dos programas, foi permitida a “importação” indireta (copiar e colar no código principal) da biblioteca `inc\emu8086.inc`, integrada ao EMU8086. A biblioteca contém diversos *macros* e procedimentos básicos que, caso houvesse a necessidade de serem implementados do zero em cada programa que deles necessita, dificultariam e muito o trabalho do programador de Assembly (Philadelphia University, 2006). Nesse tópico será descrito em detalhe o funcionamento de cada um dos *macros* e procedimentos que foram utilizados.

O macro PUTC tem a função de imprimir um único caractere na tela. Ele recebe o caractere desejado como argumento, move o valor para o registrador AL e configura o registrador AH com a função 0Eh. Em seguida, chama a interrupção da BIOS INT 10h para imprimir o caractere na tela, tudo isso enquanto preserva o valor original do registrador AX utilizando as instruções PUSH e POP.

O procedimento SCAN_NUM é o responsável por ler o que digitamos e armazenar no registrador CX. Ele fica em loop esperando pela entrada, conforme o usuário digita o procedimento vai multiplicando o valor acumulado no registrador CX por 10 e soma o novo dígito (é necessário para que funcione corretamente, pois o dígito anterior é empurrado para a esquerda e o novo, ao ser somado, entra no lugar correto). Ele trata o "Backspace" dividindo por 10 e isso “apaga” o último dígito, o resultado final fica guardado no registrador CX.

O procedimento PRINT_NUM serve para exibir na tela o valor numérico contido no registrador AX no formato decimal com sinal. Ele verifica primeiramente se o número é zero (se for, já imprime diretamente) ou se é negativo. Se for um número negativo, ele imprime o caractere de menos ('-'), torna o número em positivo e então chama o procedimento PRINT_NUM_UNS para imprimir o resultado sem sinal.

O procedimento PRINT_NUM_UNS imprime o valor contido no registrador AX como um número decimal sem sinal, ele funciona dividindo o número por potências decrescentes de 10 (ele começa em 10.000) para extrair os dígitos de um em um da esquerda para a direita. O procedimento converte cada dígito extraído para seu código ASCII (adicionando 30h) e imprime, também possui um flag para não imprimir os zeros à esquerda, garantindo que o número seja impresso corretamente.

4 AVALIAÇÃO DO TEMPO

No início do projeto, o grupo estipulou o tempo base para a escrita do código de cada tarefa em Assembly. Essa estimativa levou em conta o tempo consumido na implementação em C, notavelmente influenciado pelo fato de boa parte dos programas ter sido desenvolvida de forma mais informal (na "resenha"), o que resultou em um aumento no tempo usual de implementação.

A seguir, é apresentada a comparação entre o tempo gasto em C para cada problema e o tempo estimado inicialmente:

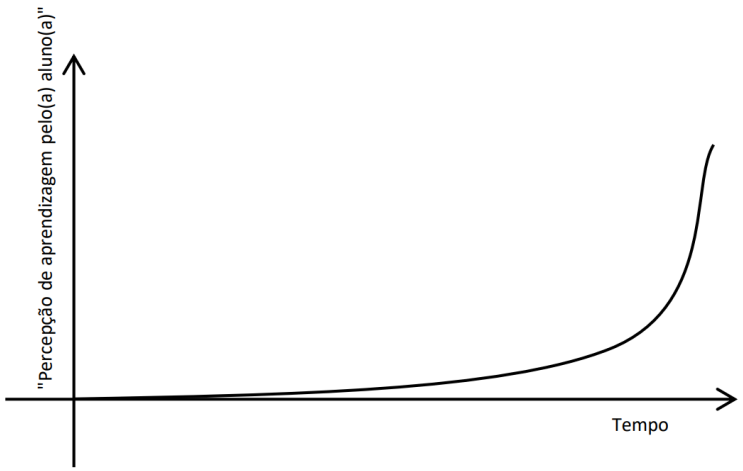
Tabela 1 - Comparação de tempo de escrita dos códigos

Linguagem	Problema 1	Problema 2	Problema 3
C (real)	20 min	20 min	10 min
Assembly (estimado)	1h 30min	4 h	2 h
Assembly (real)	4h 40min	2 h	30 min

Fonte: Elaborado pelo autor.

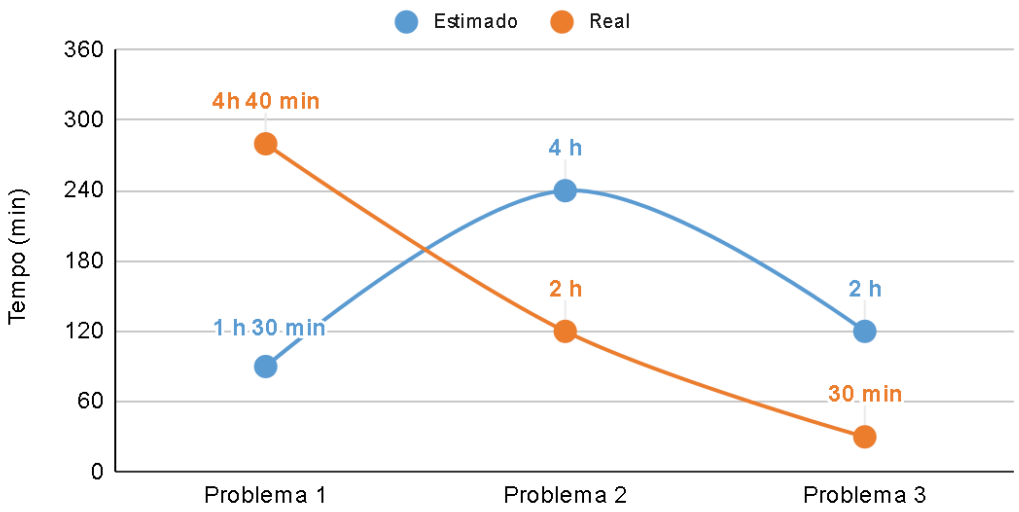
A curva de aprendizado, prevista pelo professor durante a explicação do conteúdo (Figura 1), manifestou-se neste trabalho, como pode ser visto na Figura 2 (em ordem invertida). Enquanto o primeiro desafio levou quase 5 horas para ser implementado em Assembly, após a conclusão bem-sucedida do primeiro problema, notou-se uma significativa redução no tempo de execução dos problemas subsequentes, superando até mesmo as estimativas de tempo.

Figura 1 - Curva da percepção de aprendizagem em Assembly



Fonte: Giraldeli (2025).

Figura 2 - Estimativa versus realidade do tempo de implementação em Assembly



Fonte: Elaborado pelo autor.

O problema 1 englobou todas as possíveis dificuldades, desde a manipulação de entradas (inputs) até as operações e condições lógicas. A experiência adquirida ao finalizar esta primeira etapa foi crucial, facilitando a progressão para os desafios seguintes. Dessa forma, os dois problemas subsequentes foram resolvidos com relativa "facilidade".

5 CONCLUSÃO

O presente trabalho possibilitou o estudo e a aplicação dos conceitos de Assembly, em consonância com o conteúdo da disciplina de Arquitetura e Organização de Computadores. Houve um aprendizado significativo, integrando os aspectos teóricos e práticos e consolidando o conhecimento prévio.

Inicialmente, a falta de familiaridade com Assembly representou um desafio, especialmente em operações como entrada de dados (*input*), impressão e divisão. Entretanto, esses obstáculos foram superados por meio do estudo de vídeos, slides e da documentação disponível no Ambiente Virtual de Aprendizagem (AVA).

Como ponto central, o trabalho consistiu na resolução de três problemas propostos, utilizando tanto Assembly quanto a linguagem C. Essa abordagem permitiu uma comparação direta entre linguagens de baixo e alto nível, evidenciando como o alto nível abstrai complexidades para otimizar o processo de programação.

REFERÊNCIAS

GIRALDELI, Flávio. [AOC 2025-2] Slides (Cap. 6): slides de aula de Arquitetura e Organização de Computadores. Instituto Federal do Espírito Santo, 2025. Disponível em: https://ava3.cefor.ifes.edu.br/pluginfile.php/1605287/mod_resource/content/15/%5BAOC%202025-2%5D%20Slides%20%28Cap.%206%29.pdf. Acesso em: 7 dez. 2025.

PHILADELPHIA UNIVERSITY. Help for Emu8086. Universidade da Filadélfia, 2006. Disponível em: https://www.philadelphia.edu.jo/academics/qhamarsheh/uploads/emu8086.pdf. Acesso em: 9 dez. 2025.