Hérick Vitor Vieira Bittencourt, Lorina Zondervan e Luiz Augusto Inthurn

UNIVALI - Universidade do Vale de Itajaí

Ciências da Computação - Arquitetura e Organização de Processadores

Avaliação 3 - Programação em Linguagem de Montagem

Professor - Thiago Felski Pereira

Data *–* 22/05/2023

# 

[Herick@edu.univali.br](mailto:Herick@edu.univali.br), [zondervan@edu.univali.br](mailto:zondervan@edu.univali.br) e [luiz\_inthurn@edu.univali.br](mailto:luiz_inthurn@edu.univali.br)

# 

# 

# 

# 

# 

# 

# 

# Código fonte do programa 1

Texto

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Durante o inicio da execução do programa 1, ele faz um jump para o main, na qual pede o tamanho que será usado para o vetor, caso o valor seja menor que 2 ou maior que 100, a entrada é considerada invalida e o código pula devolta para o começo do main, repetindo o processo até que uma entrada valida seja fornecida, após o usuário fornecer um valor, o código prepara o endereço do vetor e o tamanho dele nos registradores de argumento e então entra no for loop obterValores, na qual utiliza o registrador s0 para manter a contagem do índice, o loop obtém o valor fornecido e então calcula o endereço onde ele deve ficar com base no endereço base + (index \* 4) e então armazena-o, isto se repete até que o vetor esteja cheio até o índice desejado, ao final do obterValores, o código confere se o valor de s0 está igual a zero, cujo era o valor inicial, caso não esteja, ele obtem o valor de s0 anterior que estava armazenado na Stack diminui ela até chegar devolta a origem, apenas então a função retorna para o main.

Após todos os valores serem fornecidos, é chamada a função somaValores, na qual armazena o índice atual no Stack pointer, seguido de calcular o endereço onde está o valor atual, obtém ele em t3 e então realiza o somatório no registrador t4, ao chegar no final do for loop, o índice vai sendo carregado com base no valor anterior do Stack pointer até chegar a 0, por fim o valor do registrador t4 é movido para o registrador de retorno, a função retorna e o main mostra o resultado do somatório.

Texto

Descrição gerada automaticamente

# Código fonte do programa 2

Texto

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente com confiança média

Texto

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

O código do programa 2 funciona de forma um pouco similar ao programa 1, as diferenças a destacar são que ambas as funções obterValores e somaValores utilizam recursividade e a cada índice, os endereços de retorno também são armazenados na Stack para retornar recursivamente, com as branches responsáveis por repetir o procedimento sendo trocadas por jump-and-link, uma mudança importante no comportamento de somaValores é que no programa 2, os valores carregados são colocados temporariamente na Stack ao invés de serem diretamente somados logo após serem carregados, quando a função começa o processo de retorno, ela carrega os valores na Stack e realiza o somatório enquanto retorna, assim, o comportamento recursivo do código se torna mais similar ao código exemplo em C++

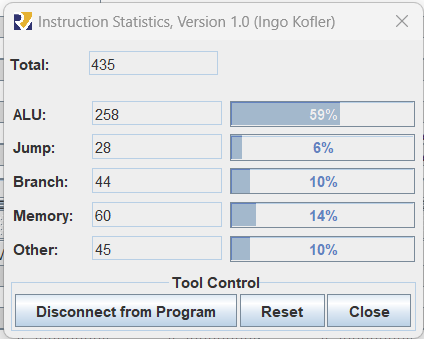
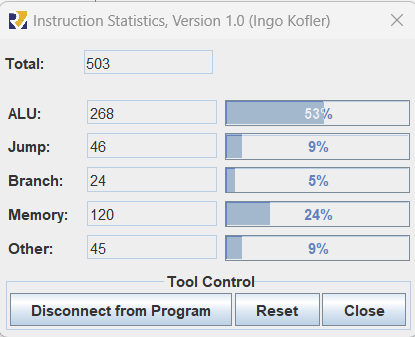
**Enunciado:**

A parte mais importante desse trabalho é a análise dos resultados obtidos nos 2 problemas. Para isso procure coletar estatísticas (instruction statistics) das duas soluções com diferentes tamanhos de vetor.

Recomenda-se o uso de gráficos para auxiliar na demonstração dos resultados. Tente responder perguntas como qual solução obtém o melhor desempenho, como os dados pioram ou melhoram as soluções e por que isso acontece.

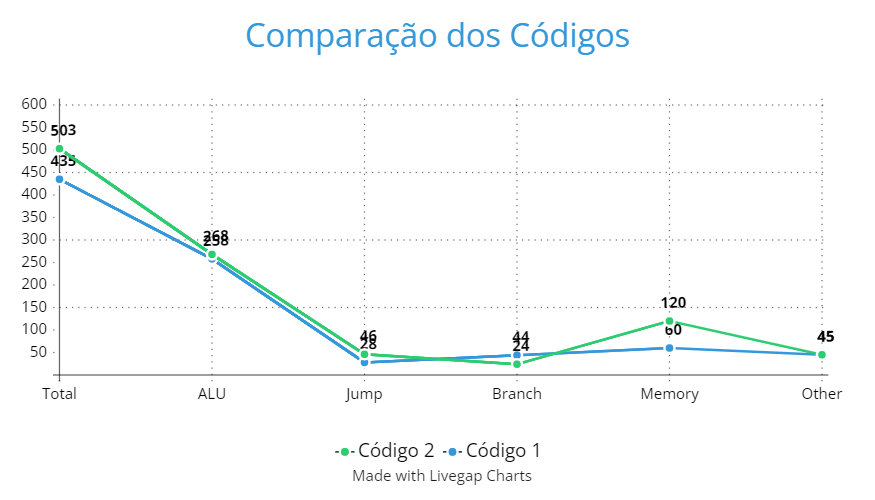
Agora iremos comparar os dois programas com o mesmo vetor de 10 posições e os mesmo números e veremos qual dos códigos irá se sair melhor que o outros.

**Código 1:**   **Código 2:**

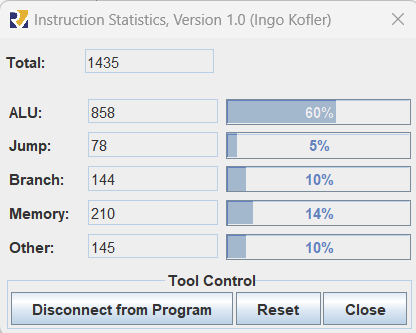
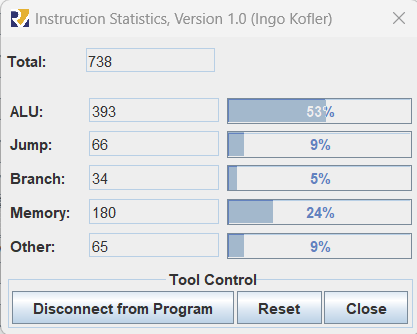
Nesse exemplo de comparação, como foi dito usamos o mesmo tamanho de vetor com 10 posições com os seguintes números 1, 30, 250, 80, 8, 38, 540, 12, 100 e 0. E com isso podemos notar que o código 1 se saiu muito melhor do que o código 2, com uma diferença de 68 operações no total.

Agora podemos ver um gráfico, para auxiliar a comparação dos dois códigos:

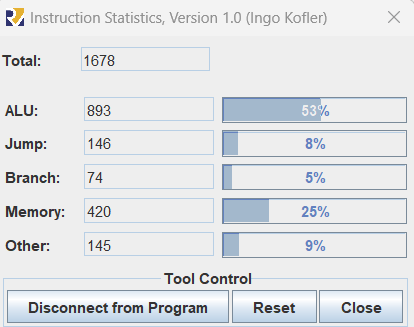


Agora iremos ver como os dois códigos se comportam com tamanhos diferentes entre vetores e seus valores aleatórios para cada um, verificando por fim qual deles se dá melhor para a aplicação. Como o resultado passado nos mostrou que o código 1 é mais rápido, iremos forçar ele um pouco, colocando nele com um vetor de 30 de posição e no código 2 iremos utilizar o vetor com 15 de posição, comparando os dois e vendo qual deles melhor se comporta em situações diferentes umas das outras.

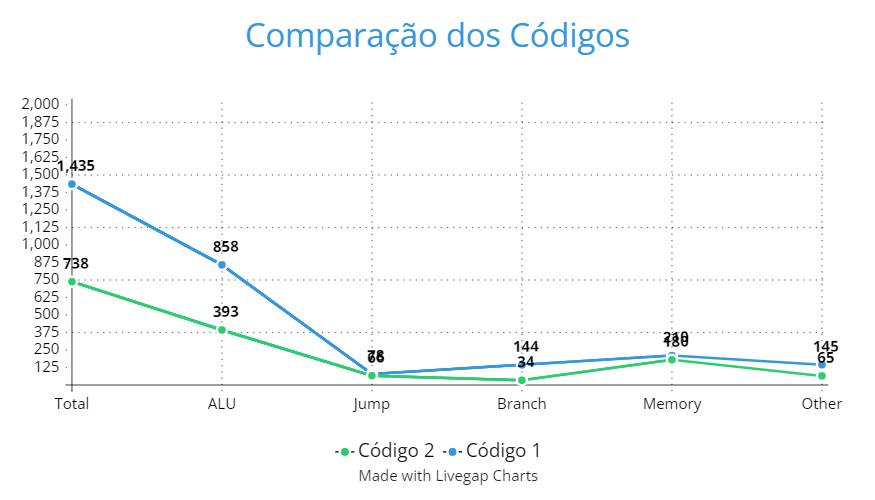
**Código 1**  **Código 2**

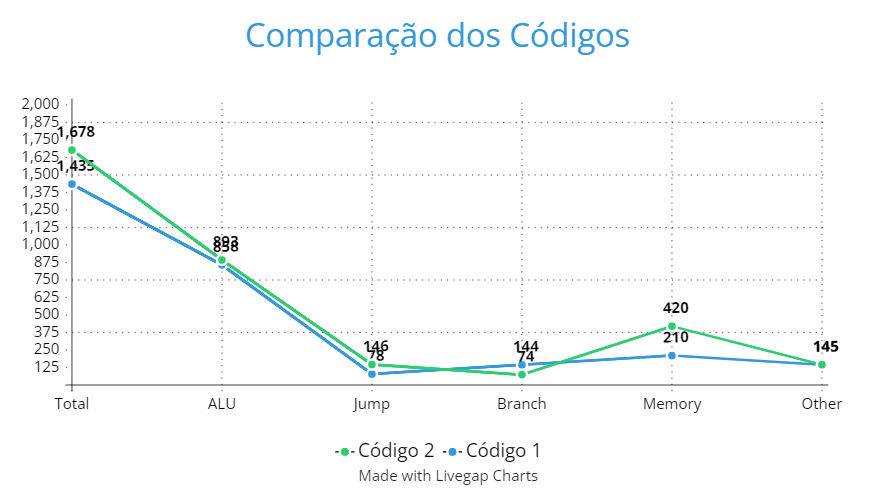
Agora podemos ver que em situações diferentes onde o código 1 é mais forçado com maiores quantidades de dados, que são eles, um vetor com 35 espaços que foram ocupados pelos seguintes números 20, 5, 38, 108, 5, 80, 78, 60, 802, 388, 88, 0, 38, 437, 827, 12, 2, 0, 38, 50, 68, 78, 88, 1000, 2, 6, 60, 888, 29, 30, 111, 280, 33, 31 e 0. E por conta disso em alguns requisitos sofre mais do que o código que não foi forçado, no caso o código 2, sendo que o código utilizou um vetor com 15 de espaço e com os seguintes números 80, 2000, 12, 0, 66, 802, 120, 100, 8, 12, 58, 12, 60 90 e 1. Não tendo uma grande quantidade de número dentro do seu vetor, mas se colocamos os mesmos valores utilizados no código 1 no código 2, e criar mais uma comparação,



Podemos por fim notar que ele se sai bem pior que o código 1, então podemos concluir que o código irá sempre se sair melhor em diversas situações, e o motivo disso ocorrer é por conta de como ele faz a soma dos vetores, que ele utiliza um for loop enquanto o código 2 durante a soma dos valores respeitas as ordem que foram dadas e as armazenadas os valores dentro de uma stack. Por fim, para concluir iremos ver um gráfico para tirar os resultados da primeira comparação, onde o código 1 recebeu mais resultados e o código 2 recebeu menos.



E por fim iremos ver o gráfico comparando os dois códigos com as mesmas quantidades, sendo elas do vetor 35, que fizemos logo após, para ver qual iria se sair melhor com a mesma quantidade de valores, mas com eles altos.



Em suma, podemos notar que o código 1, irá sempre se sair melhor do que o código 2, tendo isso em mente vemos que os resultados só são diferentes e melhores para o código 2 quando o código 1 recebe valores a mais do que o 2. Então quando os valores que são entregues para o programa 1 e o 2 são da mesma quantidade de números depende de quais sejam o código 1 sempre tem vantagem, trazendo resultados melhores em suas operações.