MEMORIAL DESCRITIVO - Programando a Arquitetura IAS Machine

Objetivo: Transformar programas de diversos propósitos escritos em C para linguagem de baixo nível, Assembly da IAS Machine, detalhando as etapas, desafios e raciocínios envolvidos no processo.

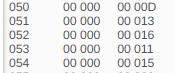
* Primeiro problema: Optmization

Entendendo o problema: Um grupo de colegas decide usar de vetores para saber a soma de suas idades exceto a própria. A lógica é que deveria haver um vetor que contenha as idades dos alunos e um segundo que recebesse em sua posição i, a soma das idades presentes no vetor anterior, exceto os da posição i.

Resolução do problema proposto: Foram feitos dois algoritmos que resolvem o problema descrito, o primeiro é mais rápido e eficiente, o segundo é mais lento e faz mais comparações e modificações em variáveis.

Código Otimizado: Esse algoritmo consiste em fazer um somatório único das idades, posteriormente o vetor somas é criado e a cada iteração o valor da idade na posição i é subtraído da soma total de idades. Esse processo garante uma maior agilidade e fluidez para o programa já que o vetor idades é iterado apenas uma vez.

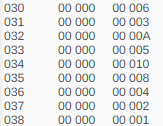
Código Não Otimizado: Esse algoritmo é mais simples, contudo, tem um tempo de processamento superior ao supra referido, são feitas n² (n ao quadrado) comparações sendo no tamanho do vetor de idades. Aqui, o vetor de idades é iterado e somado diversas vezes armazenando o resultado no vetor de somas. Limitações: Tanto o programa codificado em Assembly como o em hexadecimal possuem limitações; o código é feito para funcionar com os números do exercício e o vetor de idades deve ter tamanho 5. Contudo o programa pode ser atualizado para trabalhar com vetores maiores.

Outputs das versões “Boa” e “Ruim”: 

* Segundo Problema: Program Challenge

Entendendo o problema: É necessário fazer uma implementação da conjectura de Collatz na linguagem de Assembly das IAS Machine, a qual, a partir de um número n, divide esse valor por 2, caso ele seja par, ou multiplica por 3 e adiciona 1 caso seja ímpar, o processo continua até que o valor de n se torne 1.

Resolução do problema proposto: Primeiramente, foi criado um apontador para a primeira posição onde os valores de cada iteração da conjectura serão armazenados (a partir da posição de memória 030), variáveis que guardam os valores constantes 0,1,2 e o n. Em seguida, são feitas comparações a fim de verificar a paridade de n, caso seja par, n =/ 2, caso contrário, n = 3n +1, o programa é encerrado quando o critério de parada n = 1 é atingido.

Output do programa: 

* Terceiro Problema: Optimization

Entendendo o problema: Foi proposto a elaboração de dois códigos Assembly que implementassem e executassem a operação matemática de potenciação (x^n), devido a sua importância e recorrência nos algoritmos atuais, nessa operação um dado número x é multiplicado por ele mesmo, uma quantidade n de vezes.

Resolução do problema proposto: Foram escritos dois programas, embora os dois tenham tempos de execução parecidos dadas entradas pequenas, a diferença de velocidade e eficiência é notável para entradas suficientemente grandes. A operação realizada em ambos os casos foi 23, e os resultados foram armazenados, respectivamente, nas posições de memória 022 e 020.

Fragmento 1: Essa função é mais simples e intuitiva, contudo, o número de multiplicações é igual ao número n da entrada, o que faz seu custo computacional ser da ordem de O(n), o que pode causar um custo computacional elevado para entradas grandes.

Fragmento 2: A exponenciação binária (ou quadrádica) é uma forma mais eficiente de fazer o cálculo, aqui para cada caso de n é feita uma abordagem diferente. Se n for um valor par será feito (X2)n/2 e se n for um valor ímpar: X \* Xn-1 . É um algoritmo mais elaborado e eficiente, seu custo é O(log n), que em comparação com o algoritmo do fragmento 1 representa uma vantagem em desempenho.

Output do programa:

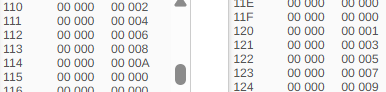
Fragmento 1:  Fragmento 2: 

* Quarto problema: Program Challenge

Entendendo o problema: Escrever um programa em código Assembly para o IAS, que verifica se um elemento do vetor é par ou ímpar, em seguida armazena o elemento em um vetor correspondente à sua paridade.

Resolução do problema proposto: São reservados 10 espaços de memória para cada vetor (pares e ímpares), já que no pior caso, haveria apenas uma paridade no vetor V. Isso garante que haverá espaço para todos os valores computados. Sendo que, o vetor de pares e ímpares iniciam-se na posição de memória 110 e 120, respectivamente.

O vetor V é iterado e testado a cada elemento com a operação: %2, se o resto da divisão do número pelo valor dois for nulo, o vetor dos pares recebe o valor de V, se não, ele é armazenado no vetor de valores ímpares.

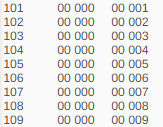
Output do programa: 

* Quinto Problema: Sorting

Entendendo o problema: É necessário escrever um programa que ordene em ordem crescente, números inteiros de um vetor V, sem a criação de um vetor auxiliar. Para que essa ordenação seja feita é necessário a utilização do método Selection Sort que foi apresentado no enunciado do exercício.

Resolução do problema proposto: O algoritmo itera sobre o vetor V (na posição de memória 100) usando de 2 laços “for”, de modo que o algoritmo seleciona V[i] como primeiro elemento, enquanto V[j] percorre o restante do vetor até encontrar o menor valor da porção não ordenada. Em seguida insere o elemento de menor valor no final porção ordenada (usando de uma variável auxiliar), e esse processo se repete até que todo o vetor esteja ordenado. A condição de parada é decidida pelo contador MAX, que contém o tamanho do vetor.

Output do programa:

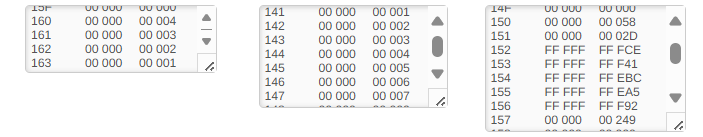


* Sexto problema: Polynomial Function

Entendendo o problema: É necessário calcular o valor de um dado polinômio em K pontos distintos dados os valores dos coeficientes e um vetor com os valores de X, além de guardar cada resultado no vetor Px.

Resolução do problema proposto: O programa começa inicializando apontadores para o vetor X (que guarda os pontos que serão calculados no polinômio), para o vetor dos expoentes e vetor dos coeficientes. Após a série de inicializações, verifica-se o valor do expoente j, para que seja possível calcular o valor de Xj, por meio de multiplicações sucessivas com critério de parada j = 1. Em seguida, multiplica-se o resultado da potência pelo coeficiente atual da iteração e este valor é armazenado na variável “Somas Parciais”. Feito isso, o programa retorna até a linha do vetor dos expoentes para que esses e os coeficientes sejam incrementados, esse laço se repete até que as operações sejam feitas com todos os coeficientes.

Por fim, o programa armazena o resultado acumulado em “Somas Parciais” no espaço de memória indicado pelo apontador Px, os apontadores de coeficientes e expoentes são resetados para seu conteúdo inicial, volta para a linha inicial e incrementa a posição do vetor que contém os valores de x, o ciclo continuará até que o vetor Px seja preenchido com os resultados das operações. Dessa maneira, o programa finaliza e o resultado é obtido em função do Polinômio: 88-27x- 7x2- 11x3 + 2x4, nos pontos de teste: (0, 1, 2, 3, 4, 5, 6, 7), e é armazenado a partir da posição de memória 0x150.

Output do programa: 

Os Testes No Simulador Do IAS: Todos os programas propostos foram resolvidos em equipe usando a extensão Live Share do VS Code para que todos possam editar o mesmo arquivo ao mesmo tempo. Após discussões da lógica a ser usada, o programa era escrito em linguagem de montagem Assembly e sua memória era montada e simulações eram feitas ainda no editor do VS Code para que pudéssemos imaginar como seria o funcionamento do programa. Quando a equipe concordava que o programa parecia correto o hexadecimal era escrito e conferido pelos integrantes.

Após esse processo tanto o hexadecimal quanto a memória eram carregados para o simulador do IAS Machine do Instituto de Computação da Faculdade Unicamp. Após rodarmos o programa observávamos a memória na região onde deveria haver o resultado. Comumente tínhamos que rodar o programa passo a passo até identificarmos onde estava o erro, seja de lógica, de memória ou de tradução para o hexadecimal.

Após vários testes, encontrávamos a combinação certa entre os dados na memória e a lógica escolhida, nesse momento elegíamos a versão final do programa que melhor resolvia o problema proposto.

Integrantes: Daniel Borges Gonçalves - 12311BCC005

Gustavo Costa Miranda - 12311BCC034

Luana Rodrigues Borges - 12311BCC028

Sophia Ladir Pereira Vieira de Morais - 12311BCC004

Raquel Emillene FreireFreire Thomé - 12311BCC026