MEMORIAL DESCRITIVO - Programando a Arquitetura MIPS

Objetivo: Transformar programas de diversos propósitos para linguagem de baixo nível, Assembly MIPS-32, detalhando as etapas, desafios e raciocínios envolvidos no processo

- ## Primeiro problema: Algoritmo da Série Harmônica

Entendendo o problema: É necessário escrever um programa que calcule a série harmônica de um dado número, esse cálculo é muito importante na matemática sendo usado para calcular a constante de Euler-Mascheroni (também chamada de constante de Euler).

Resolução do problema proposto: Foi feito um algoritmo que calcula a série harmônica de duas formas: utilizando precisão simples (float) e precisão dupla (double). Nos dois casos o algoritmo segue a mesma lógica, o valor de i é inicializado como 1 e a cada iteração do loop seu valor é atualizado e o resultado da divisão de 1/i é somado ao registrador que armazena o valor da soma. A diferença entre os algoritmos é a forma com que as variáveis são convertidas, utilizando os operadores “cvt.s.w” para float ou “cvt.d.w” para double e manipuladas nos registradores, utilizando os operadores específicos para esses tipos de dados.

- ## Segundo Problema: Série de Taylor da Função Exponencial

Entendendo o problema: É pedido que seja feito o somatório da série de Taylor que representa a função exponencial até que seja alcançado o erro 0.00001.

Resolução do problema proposto: Primeiramente, criamos um loop onde o valor de X é multiplicado por si mesmo a cada iteração (deste modo obtém-se as potências de x^n). Além disso, encontramos o sucessor do contador de iterações (n), para então multiplicá-lo pelo fatorial do contador anterior, a fim de determinar o fatorial (n!), que irá dividir a expressão, uma vez que o contador é inicializado com 0. Após encontrar o x^n e o n!, executamos a divisão e verificamos se o erro foi alcançado, se não adicionamos o resultado da divisão ao somatório iterativamente.

- ## Terceiro Problema: Avaliação Polinomial

Entendendo o problema: É proposto criar dois programas com mesmo propósito, calcular o valor de um polinômio dado um valor de X. O primeiro segue a lógica tradicional na qual o valor de X é elevado ao grau do polinômio e depois multiplicado pelo seu termo, essa sequência é mantida até que a potência de X seja 0. O somatório

destes valores parciais é a resposta. No modelo secundário, o Método de Horner, o X é posto em evidência repetidas vezes, não sendo necessário calcular suas potencias, cada valor parcial é somado de modo a obter a mesma resposta do primeiro método.

Resolução do problema proposto:

Polinômio 1: O método tradicional é implementado fazendo um loop em que o termo com o X^0 é somado de antemão. Posteriormente, o termo com X^1 é multiplicado pelo seu respectivo coeficiente e em seguida é somado ao resultado parcial, para que a cada termo somado, o loop seguinte inicie fazendo o cálculo do valor atual da potência de X vezes a potência do loop anterior, considerando o vetor de coeficientes e o X definidos pelo usuário. Deste modo, o cálculo anterior é reaproveitado a cada ciclo. Essa sequência é finalizada quando o n-ésimo termo do loop de um polinômio de grau N+1 é somado ao resultado parcial.

Polinômio 2: O Método de Horner, ao colocar os X em evidência, despensa a necessidade de calcularmos potências. O valor que teria maior potência, caso as distribuições fossem realizadas, é o primeiro a ser somado no registrador de resposta. Dentro do loop esse valor se multiplica a X e soma com o próximo termo do polinômio, o ciclo se repete, multiplicando a nova “resposta parcial” por X e somando com o próximo termo até que o polinômio chegue ao seu fim. O resultado deste método é igual ao tradicional.

- Quarto problema: Regressão linear

Entendendo o problema: Escrever um programa em código Assembly para o IAS, que, pelo método da regressão linear, permite determinar os coeficientes da reta Y = A*X+B que se aproxima melhor do conjunto de pontos fornecidos.

Resolução do problema proposto: Primeiramente, o usuário determina a quantidades de pontos no plano cartesiano que serão utilizados para os cálculos dos coeficientes. Em seguida, preenchemos aleatoriamente, dois vetores, um contendo as abscissas e outro as ordenadas dos pontos. Posteriormente, inicializamos o loop que irá executar os somatórios de Sx, Sy, Sxx e Sxy, utilizando os elementos dos vetores. Ao finalizar esse processo, realizamos as operações determinadas no enunciado multiplicando e dividindo os somatórios, a fim de obtermos os coeficientes A e B.

- Quinto Problema: Quicksort

Entendendo o problema: É necessário traduzir a função quicksort, escrita em C, que ordena um vetor recursivamente, para MIPS Assembly.

Resolução do problema proposto: A princípio, os valores de low e high são definidos com o início e fim do vetor a ser ordenado respectivamente. Em seguida, a função quicksort é chamada e seu endereço de retorno para a main, bem como seus parâmetros, são armazenados na stack a fim de não serem perdidos no processamento dos dados. Na função, é usado constantemente o valor na posição média do vetor, chamado de pivô; são comparados os valores armazenados antes e depois com o pivô. São percorridos os elementos anteriores ao pivô até que seja encontrado um de valor maior ou igual, ao mesmo tempo que são percorridos os elementos posteriores em busca de um com valor menor ou igual, ao encontrar tais valores realiza-se a troca desses elementos. Com base nisso, a função é chamada recursivamente, alterando os parâmetros high e low, a deender do caso; antes de cada chamada o valor de $ra e dos parâmetros são salvos usando o ponteiro $sp da stack, a cada retorno, os valores são recuperados e passados para a função anterior a fim de terminar o processamento. Deste modo, o quicksort é concluído e retorna para a main, onde é impresso o resultado: um vetor ordenado.

- Sexto problema: Multiplicação de matrizes

Entendendo o problema: É necessário escrever duas funções de multiplicação de matrizes C = A*B, usando duas estratégias de indexação diferentes ikj e ijk.

Resolução do problema proposto: Primeiramente, realizamos uma função chamada rmo (row-major-order) que utiliza um loop para preencher as matrizes A e B com elementos do tipo float. Posteriormente, nas duas funções propostas foram implementadas usando 3 loops e dentro do terceiro loop, é efetivada a multiplicação das matrizes A e B; o offset de cada matriz é calculado e o valor armazenado em A é obtido, multiplicado pelo da matriz B, em seguida somado com todas as outras multiplicações do loop mais interno. Após finalizar o somatório, o resultado é armazenado na devida posição da matriz C. A diferença de implementação das duas funções consiste na mudança no contador do loop mais interno com o do intermediário.

Análise das funções: a função que usa da configuração ijk possui uma taxa de acertos por volta de 86%, superior à função de configuração ikj que possui por volta de 65%. Ao usarmos 3 matrizes de 512 x 512 foram feitos mais de 269 milhões de acessos ao cache.

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