UNIVERSIDADE ESTADUAL DE MARINGÁ
CENTRO DE TECNOLOGIA - DEPARTAMENTO DE INFORMÁTICA
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Relatório do Resolutor de Sistemas Lineares de Três Variáveis em GNU Assembly

6894 - Programação para Interfaceamento de Hardware e Software

Ronaldo Augusto de Lara Gonçalves

Ricardo Henrique Brunetto RA: 94182

Maringá
2017

Introdução

O presente documento contém relatório do primeiro trabalho da matéria ministrada pelo Ronaldo Augusto de Lara Gonçalves na disciplina de Programação para Interfaceamento de Hardware e Software para a turma de Bacharelado em Ciência da Computação de 2015. Tal documento apresenta uma estrutura de forma a expor o funcionamento dos principais módulos do programa, bem como salientar as limitações e possíveis exceções.

O conteúdo aqui citado é advindo, além das anotações em aula e materiais disponibilizados pelo professor, da gama de referências bibliográficas por ele recomendadas e encontradas. Além disso, a fundamentação teórica é baseada em um contexto não abrangido pelo escopo da disciplina.

1 Fundamentação Teórica

Tal seção busca apresentar a teoria na qual a implementação do Resolutor de Sistemas Lineares de Três variáveis se baseia. Não serão abordados aspectos referentes à linguagem de programação, tendo foco específico na técnica utilizada para resolver o problema em questão. Neste ínterim, utilizou-se o **Teorema de Cramer** para a resolução do sistema linear de três variáveis e o **Teorema de Sarrus** para os cálculos dos determinantes.

1.1 Teorema de Cramer

Existem diversos métodos de encontrar a solução de um sistema linear de n variáveis, caso exista, ou mostrar sua inexistência. Um dos métodos mais eficientes é a **Regra de Cramer**, mas só pode ser utilizado quando o número de equações e de incógnitas são iguais.

Assim, dadas n variáveis, deve-se ter n equações para utilizar o método. Será apresentado o funcionamento do método, mas sua prova formal será omitida por questões de complexidade.

Dado um sistema de equações lineares de n variáveis com n equações, o seguinte é valido:

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix}$$

onde cada a_ij é um coeficiente e forma matriz dos coeficientes C, x_k é uma variável e forma a matriz das variáveis X, e z_m é um termo independente e forma a matriz dos termos independentes Z. Assim, tem-se:

$$CX = Z$$

Denomina-se *matriz ampliada* a matriz $A = CZ$, ou seja, a matriz concatenada de C e Z, sendo $n + 1 \times n$.

O **Teorema de Cramer** diz que

$$x_i = \frac{det(C_i)}{det(C)}$$

onde C_i é uma matriz obtida através da substituição da coluna i pela matriz dos termos independentes Z.

1.2 Teorema de Sarrus

O Teorema (ou Regra) de Sarrus consiste de uma técnica que define uma fórmula fechada para o cálculo de determinantes de matrizes de ordem 3 através do conhecimento imediato de seus elementos.

A técnica prática consiste em repetir as duas primeiras colunas ao lado da matriz original e: somar os produtos das diagonais principais (serão três) e então subtrair de tal resultado a somatória dos produtos das diagonais secundárias (também serão três), conforme ilustra a Figura 1.

Contudo, em termos matemáticos, pode-se definir uma fórmula fixa que corresponda a este procedimento de duplicação de colunas. Tal fórmula é baseada no **Teorema de Laplace**. O

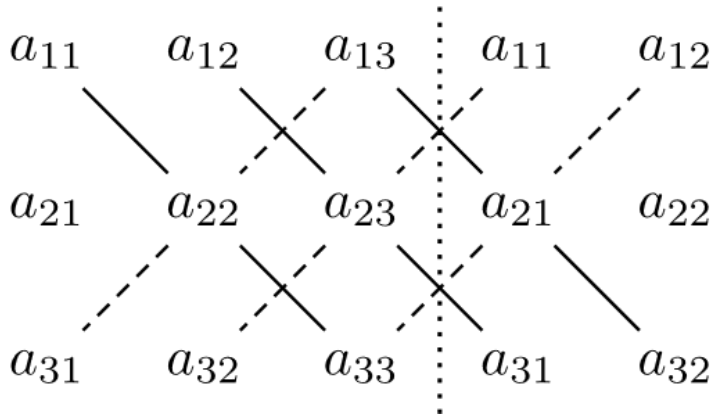

Figura 1: Representação da aplicação da Regra de Sarrus

Teorema de Laplace pode ser enunciado da seguinte maneira:

Dada uma matriz A de ordem n, fixa-se uma linha i de A e o determinante de A é dado por:

$$det(A) = \sum_{j=1}^{n} a_{ij}(-1)^{i+j} det(A_{ij})$$

onde A_{ij} é a **submatriz**, de ordem $n-1$, obtida removendo-se a linha i e coluna j de A.

Aplicando Laplace à uma matriz A de ordem 3 e fixando a primeira linha, tem-se:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$$det(A) = a_{11}(-1)^{1+1} \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} + a_{12}(-1)^{1+2} \begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{13}(-1)^{1+3} \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix}$$

Aplicando a regra de determinantes de matriz de ordem 2 (que também advém de Laplace), tem-se:

$$det(A) = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{33} + a_{13}a_{21}a_{32} - a_{11}a_{23}a_{32} - a_{12}a_{23}a_{31} - a_{13}a_{22}a_{31} \qquad (1)$$

Assim, define-se a **Regra de Sarrus** como a equação 1.

2 Estrutura e Funcionamento

Para que se pudesse trabalhar com melhor aproveitamento de código, desempenho e legibilidade, foram criadas funcionalidades específicas para que fossem chamadas durante a execução do programa. Cada uma das funcionalidades possui uma descrição baseada em:

- **Propósito**, que descreve o que, de fato, a funcionalidade faz.
- **Pré-Condição**, que descreve o estado do programa que a funcionalidade espera para que cumpra seu propósito.
- **Pós-Condição**, que descreve o estado do programa após a execução da funcionalidade.
- **Registradores alterados**, que lista quais sofreram alterações, para que se torne mais fácil administrar o fluxo de execução.

Evidentemente, existem outras *labels* no código. Contudo, as listadas abaixo representam funcionalidades específicas que são chamadas no decorrer da execução. As seguintes funcionalidades são implementadas, a saber:

matricial_linear

- **Propósito**: Ir para o elemento de coordenadas matriciais através da conversão em um deslocamento linear.
- **Pré-Condição**:

  A linha em `%ebx`

  A coluna em `%ecx`

  Endereço inicial da matriz no topo da pilha
- **Pós-Condição**: Avança para o elemento [`%ebx`][`%ecx`] da matriz.
- **Registradores alterados**: `%edi`, `%eax` e `%ecx`

proximo_campo

- **Propósito**: Avançar um campo de tamanho fixo em um endereço de memória.

- **Pré-Condição**: Endereço de memória no topo da pilha
- **Pós-Condição**: Endereço de memória retorna à pilha deslocado em 4 *bytes*.
- **Registradores alterados**: `%edi`

pular

- **Propósito**: Pular uma quantidade de campos de tamanho fixo em um endereço de memória.
- **Pré-Condição**:

  Quantidade de *bytes* em `%ebx`

  Endereço de memória no topo da pilha
- **Pós-Condição**: Avança `%eax` *bytes* no endereço de memória e o empilha.
- **Registradores alterados**: `%edi` e `%eax`

ler_dados

- **Propósito**: Ler as entradas do sistema e preencher a matriz ampliada.
- **Pré-Condição**: Endereço da matriz principal já alocado
- **Pós-Condição**: A matriz ampliada preenchida (coeficientes e termos independentes)
- **Registradores alterados**: `%edi` e `%ecx`

alocar_matriz

- **Propósito**: Aloca um bloco de memória com base em valores matriciais.
- **Pré-Condição**:

  Quantidade de linhas da matriz em `%eax`

  Quantidade de colunas da matriz em `%ebx`
- **Pós-Condição**: Endereço do primeiro elemento da matriz de inteiros alocada está em `%edi`
- **Registradores alterados**: `%edi`, `%eax` e `%ecx`

mostrar_sistema

- **Propósito**: Exibe o sistema linear na tela.
- **Pré-Condição**: Endereço do primeiro elemento da matriz ampliada em `%edi`
- **Pós-Condição**: -
- **Registradores alterados**: `%edi` e `%ecx`

determinante

- **Propósito**: Calcula o determinante de uma matriz através da Regra de Sarrus.
- **Pré-Condição**: Endereço do primeiro elemento da matriz em `%edi`
- **Pós-Condição**: A variável `det_valor` possui o determinante da matriz
- **Registradores alterados**: `%edi`, `%eax`, `%ebx`, `%ecx` e `%edx`

Neste ponto, aborda-se uma questão inerente ao cálculo do determinante, que se baseia na Regra de Sarrus (vide 1.2). Cada uma das *labels* que compõem a funcionalidade representam o cálculo de um termo da equação 1, definida pela Regra de Sarrus. Desta forma,

- `calcular_primeiro_termo` **produz** $a_{11}a_{22}a_{33}$
- `calcular_segundo_termo` **produz** $a_{12}a_{23}a_{33}$
- `calcular_terceiro_termo` **produz** $a_{13}a_{21}a_{32}$
- `calcular_quarto_termo` **produz** $-(a_{11}a_{23}a_{32})$
- `calcular_quinto_termo` **produz** $-(a_{12}a_{23}a_{31})$
- `calcular_sexto_termo` **produz** $-(a_{13}a_{22}a_{31})$

e cada um destes valores é somado à variável `det_valor` que, inicialmente possui o valor 0. Desta forma, `determinante` calcula corretamente o determinante da matriz.

gerar_matriz_sem_z

- **Propósito**: Gera a matriz sem a última coluna. No caso, sem a matriz dos termos independentes. Em outras palavras, isola a matriz dos coeficientes (matriz quadrada) da matriz

ampliada.

- **Pré-Condição**: Endereço de memória da matriz auxiliar alocado e em `%esi`
- **Pós-Condição**: Copiada a matriz quadrada dos coeficientes em `%esi`
- **Registradores alterados**: `%edi`, `%esi` e `%ecx`

copiar_ultima_coluna

- **Propósito**: Substitui uma coluna da matriz pela última coluna de outra.
- **Pré-Condição**:

  Endereço da matriz principal (em geral, a matriz ampliada) em `%edi`

  Endereço da matriz auxiliar (que receberá a coluna) em `%esi`

  Índice da coluna da matriz auxiliar que será substituída em `%ebx`
- **Pós-Condição**: Avança `%eax` *bytes* no endereço de memória e o empilha.
- **Registradores alterados**: -

resolver_sistema

- **Propósito**: Resolver o sistema linear através da aplicação da Regra de Cramer.
- **Pré-Condição**:

  Endereço da matriz principal (matriz ampliada) já preenchida em `%edi`

  Determinante da matriz dos coeficientes já deve ter sido calculado e armazenado em `det_D`
- **Pós-Condição**: Resolve o sistema e exibe os resultados à medida que são calculados.
- **Registradores alterados**: `%eax`, `%ebx`, `%ecx`, `%edx`, `%edi` e `%esi`

Neste ponto, tal funcionalidade é responsável por:

- Alocar uma matriz auxiliar
- Gerar, nessa matriz recém-alocada, a matriz dos coeficientes (matriz ampliada sem a matriz dos termos independentes)
- Substituir uma das colunas (iterativo, começa substituindo a primeira e se segue até a última) pela matriz dos termos independentes (a última coluna da matriz ampliada)

- Calcular o determinante de tal matriz
- Dividir pelo determinante da matriz principal dos coeficientes (`det_valor`)
- Exibir o resultado

Dessa forma, a funcionalidade `resolver_sistema` resolve corretamente o sistema linear através da Regra de Cramer.

inicio_resolucao

- **Propósito**: Prepara o que é necessário para a execução do programa.
- **Pré-Condição**: -
- **Pós-Condição**:

  Matriz ampliada alocada e preenchida

  Determinante da matriz principal dos coeficientes calculado e armazenado em `det_D`
- **Registradores alterados**: *N/A*

Neste ponto que, caso o determinante principal (`det_D`) seja 0, o programa interrompe a execução classificando o sistema como **impossível** ou **possível e indeterminado**.

3 Limitações e Exceções

Tal programa possui as seguintes limitações:

- Calcula **unicamente** soluções para sistemas de três variáveis com três equações.
- Opera **apenas** sobre números inteiros e não faz verificações das entradas.
- Realiza divisões produzindo resultados **inteiros**.

  Dessa forma, os determinantes são arredondados, **considerando apenas suas partes inteiras**.
- Os ponteiros das matrizes auxiliares são alocados e desalocados a cada iteração da Regra de Cramer e a matriz principal é desalocada no final da resolução do sistema. Contudo, resíduos podem acabar sendo deixados na pilha a cada execução, o que, a longo prazo (bem longo), pode ocasionar estouro de memória.