Universidade Federal de Ouro Preto - UFOP
Instituto de Ciências Exatas e Biológicas - ICEB
Departamento de Computação - DECOM
Ciencia da Computacao

Trabalho Prático 1

BCC 266 - Organização de computadores

Lucas Chagas, Pedro Morais, Nicolas Mendes
Professor: Pedro Henrique Lopes Silva

Ouro Preto
10 de janeiro de 2023

Implementação

No desenvolvimento deste projeto,foi implementado ao código-fonte que possuía 4 funções básicas como mover o valor da RAM,desligar a máquina e executar as operações de soma e subtração,as funções de multiplicação,tendo como base a operação de soma,a função de divisão,usando-se as funções de soma e subtração,a função de exponencial,utilizando-se a função de multiplicação como base e função que faz as operações de soma da sequência de Fibonacci e mostra o termo escolhido pelo usuário.

- Função de multiplicação [**generateMultiplicationInstructions()**]

```
127 //Primeiro fator na posição 0 da RAM; Segundo fator na posição 1; Resultado na posição 2;
128 Instruction *generateMultiplicationInstructions(int fator1, int fator2, int exponenciacao)
129 {
130     int qtdInstrucoes = 3;
131 
132     Instruction *instrucoes = (Instruction *)malloc(qtdInstrucoes * sizeof(Instruction));
133 
134     if (exponenciacao == 0) // Caso em que trata-se de uma multiplicação simples independente
135     {
136 
137         // Pegando o fator 1 e colocando na posição 0 do vetor da RAM
138         instrucoes[0].opcode = 0;
139         instrucoes[0].info1 = fator1;
140         instrucoes[0].info2 = 0;
141 
142         // Pegando o fator 2 e colocando na posição 1 do vetor da RAM
143         instrucoes[1].opcode = 0;
144         instrucoes[1].info1 = fator2;
145         instrucoes[1].info2 = 1;
146 
147         // Colocando o termo neutro da soma no lugar do resultado da multiplicação (posição 2 do vetor da RAM)
148         instrucoes[2].opcode = 0;
149         instrucoes[2].info1 = 0;
150         instrucoes[2].info2 = 2;
151 
152         // Operação de multiplicação em si
153         for (int i = 0; i < fator1; i++)
154         {
155             qtdInstrucoes++;
156             instrucoes = (Instruction *)realloc(instrucoes, qtdInstrucoes * sizeof(Instruction));
157             // Soma
158             instrucoes[qtdInstrucoes - 1].opcode = 1;
159             instrucoes[qtdInstrucoes - 1].info1 = 1; // Posição do fator2
160             instrucoes[qtdInstrucoes - 1].info2 = 2; // Posição do resultado
161             instrucoes[qtdInstrucoes - 1].info3 = 2; // Armazena a informação na posição do resultado
162         }
163 
164         // Criando a instrução para finalizar a máquina
165         instrucoes = realloc(instrucoes, (qtdInstrucoes + 1) * sizeof(Instruction));
166         instrucoes[qtdInstrucoes].opcode = -1;
167         instrucoes[qtdInstrucoes].info1 = -1;
168         instrucoes[qtdInstrucoes].info2 = -1;
169         instrucoes[qtdInstrucoes].info3 = -1;
170     }
```

Imagem 1 : Função de multiplicação

A função de multiplicação possui três parâmetros, sendo eles:

- fator1 (int): Um fator da multiplicação;
- fator2 (int): Outro fator da multiplicação;
- exponenciacao (int): Uma flag utilizada para sinalizar se a multiplicação é independente ou se a multiplicação é parte de uma exponenciação;

Inicialmente, o vetor principal (instruções), do tipo Instruction, é alocado dinamicamente com 3 posições.

A partir desse ponto há duas abordagens dentro da função: uma multiplicação independente e multiplicação dentro de uma exponenciação.

>Multiplicação independente:

O fator1 e o fator2 são movidos para as posições 0 e 1 da RAM, respectivamente. Posteriormente, o 0 (termo neutro da soma) é movido para a posição 2 da RAM.

Logo após, há um *for* executado ***fator1*** **vezes**. A cada iteração, o vetor instruções é realocado com uma posição a mais e o mesmo recebe uma instrução que soma o fator2 com o conteúdo na posição 2 da RAM, posição do resultado (acumulativo). Dessa forma, o fator2 é somado com ele mesmo ***fator1*** **vezes**, o que, na prática, seria fator1 vezes fator2.

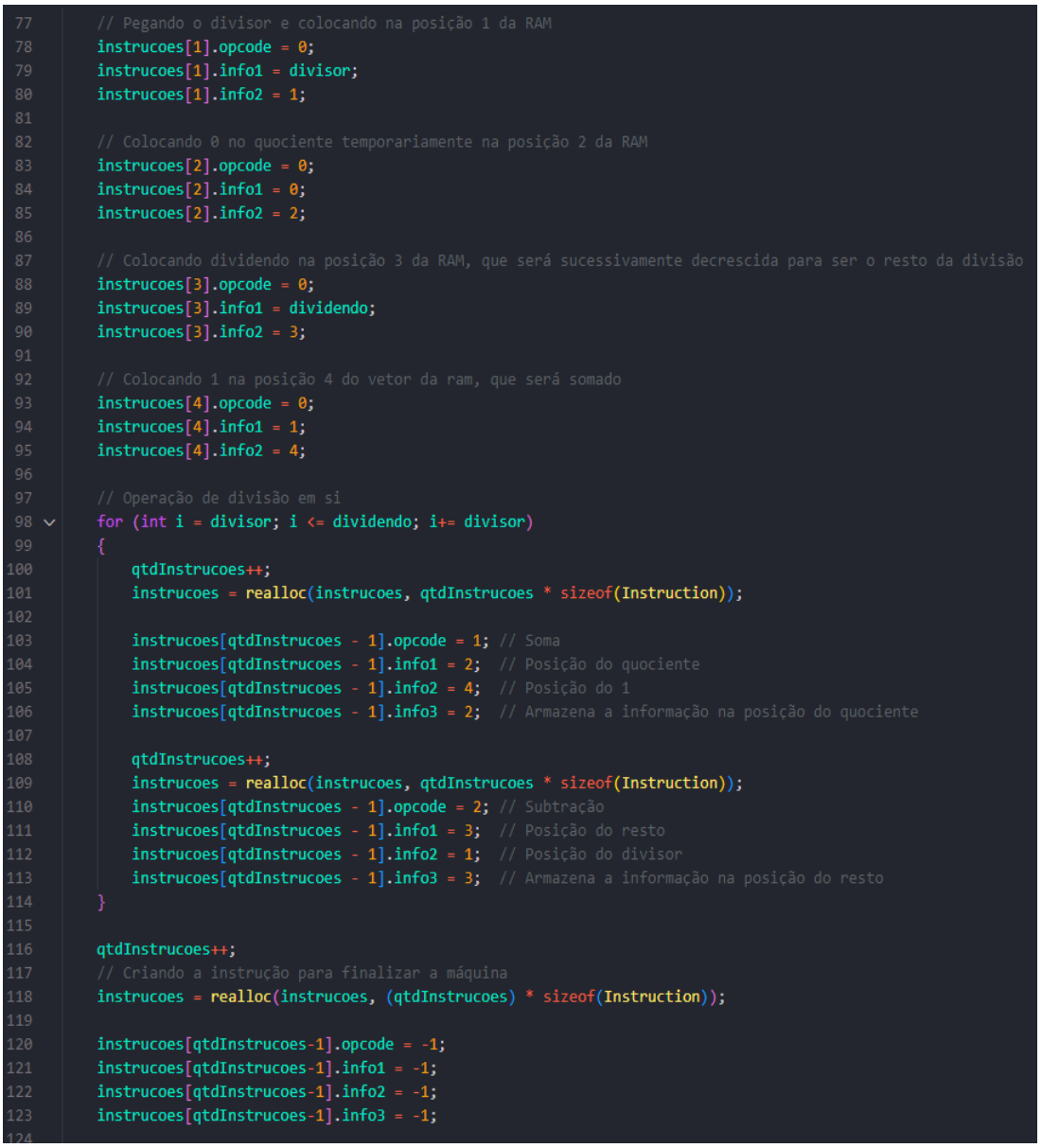
Posteriormente, é gerada a instrução de desligar a máquina e o vetor instruções é retornado.

>Multiplicação que está dentro de uma exponenciação:

Há um *for* que é executado ***fator1*** **vezes** e, a cada iteração, o vetor instruções é realocado em mais uma posição e é atribuído uma instrução que soma o conteúdo da posição 3 (parcela fixa de cada soma) com o conteúdo da posição 4 (parcela acumulativa de cada soma) e armazena o resultado na posição 4.

Posteriormente, o vetor instruções é realocado mais duas posições. Logo após, há uma instrução que soma o elemento da posição 4 (parcela acumulativa) com o elemento da posição 2 (termo neutro da soma) e armazena na posição 3 (parcela fixa) - Na prática, o elemento acumulativo passa a ser o elemento fixo para as próximas sucessivas somas. Seguidamente, há uma instrução que coloca o número zero na posição 4 (parcela acumulativa), zerando a mesma para as próximas somas. Por fim, o vetor instruções é retornado.

- Função de divisão[**generateDivisionInstructions()**]



```
    // Pegando o divisor e colocando na posição 1 da RAM
    instrucoes[1].opcode = 0;
    instrucoes[1].info1 = divisor;
    instrucoes[1].info2 = 1;

    // Colocando 0 no quociente temporariamente na posição 2 da RAM
    instrucoes[2].opcode = 0;
    instrucoes[2].info1 = 0;
    instrucoes[2].info2 = 2;

    // Colocando dividendo na posição 3 da RAM, que será sucessivamente decrescida para ser o resto da divisão
    instrucoes[3].opcode = 0;
    instrucoes[3].info1 = dividendo;
    instrucoes[3].info2 = 3;

    // Colocando 1 na posição 4 do vetor da ram, que será somado
    instrucoes[4].opcode = 0;
    instrucoes[4].info1 = 1;
    instrucoes[4].info2 = 4;

    // Operação de divisão em si
    for (int i = divisor; i <= dividendo; i+= divisor)
    {
        qtdInstrucoes++;
        instrucoes = realloc(instrucoes, qtdInstrucoes * sizeof(Instruction));

        instrucoes[qtdInstrucoes - 1].opcode = 1; // Soma
        instrucoes[qtdInstrucoes - 1].info1 = 2;  // Posição do quociente
        instrucoes[qtdInstrucoes - 1].info2 = 4;  // Posição do 1
        instrucoes[qtdInstrucoes - 1].info3 = 2;  // Armazena a informação na posição do quociente

        qtdInstrucoes++;
        instrucoes = realloc(instrucoes, qtdInstrucoes * sizeof(Instruction));
        instrucoes[qtdInstrucoes - 1].opcode = 2; // Subtração
        instrucoes[qtdInstrucoes - 1].info1 = 3;  // Posição do resto
        instrucoes[qtdInstrucoes - 1].info2 = 1;  // Posição do divisor
        instrucoes[qtdInstrucoes - 1].info3 = 3;  // Armazena a informação na posição do resto
    }

    qtdInstrucoes++;
    // Criando a instrução para finalizar a máquina
    instrucoes = realloc(instrucoes, (qtdInstrucoes) * sizeof(Instruction));

    instrucoes[qtdInstrucoes-1].opcode = -1;
    instrucoes[qtdInstrucoes-1].info1 = -1;
    instrucoes[qtdInstrucoes-1].info2 = -1;
    instrucoes[qtdInstrucoes-1].info3 = -1;
```

Imagem 2 : Função de divisão

A função de divisão possui dois parâmetros, sendo eles:

- divisor (int): Divisor ;
- dividendo (int): Dividendo;

Inicialmente, o vetor principal (instruções), do tipo Instruction, é alocado dinamicamente com 5 posições.

A partir daí o dividendo é movido para a Ram na posição 0, o divisor para a posição 1 na Ram, na posição 3 é colocado um dividendo com a função de exibir o “resto” da divisão , já na posição 2(quociente) é colocado um “0” e na 4 é colocado “1”.

Após isso, o loop é iniciado, e executado **(dividendo/divisor) vezes**, e durante o laço de repetição, a cada iteração, o quociente é somado com a Ram na posição 4 (1) e armazenado no quociente. Além disso, é realizada uma instrução de subtração entre a Ram posição 1 e a Ram posição 3, tendo o valor desta subtração armazenado na própria Ram posição 1.(a cada iteração o vetor de instruções sofre um realloc aumentando seu tamanho em 1)

E por fim, o vetor de instruções recebe mais um realloc aumentando seu tamanho em 1, e é inserindo a instrução de parada de máquina nesta última posição

- Função de exponenciação[**generateExponentiationInstructions()**]

```
    // Operação de exponenciação em si
    // Caso em que o expoente é 0, resultado igual a 1
    if (expoente == 0)
    {
        qtdInstrucoes++;
        instrucoes = (Instruction *)realloc(instrucoes, qtdInstrucoes * sizeof(Instruction));
        instrucoes[qtdInstrucoes - 1].opcode = 0; // Operação de mover para RAM
        instrucoes[qtdInstrucoes - 1].info1 = 1;
        instrucoes[qtdInstrucoes - 1].info2 = 2; // Coloca 1 no lugar do resultado

        // Criando a instrução para finalizar a máquina
        instrucoes = realloc(instrucoes, (qtdInstrucoes + 1) * sizeof(Instruction));
        instrucoes[qtdInstrucoes].opcode = -1;
        instrucoes[qtdInstrucoes].info1 = -1;
        instrucoes[qtdInstrucoes].info2 = -1;
        instrucoes[qtdInstrucoes].info3 = -1;
        return instrucoes;
    }

    // Caso em que o expoente não é 0
    // Vetor do tipo Instruction para armazenar cada vetor Instruction de cada multiplicação do loop

    int qtdInstrucoesTemp = 2 + base; // Numero de instrucoes geradas pela função generateMultiplicationInstructions() no caso da exponenciacao

    for (int i = 1; i < expoente; i++)
    {
        Instruction *instrucoesTemp;
        instrucoesTemp = generateMultiplicationInstructions(base, base, 1);
        instrucoes = realloc(instrucoes, (qtdInstrucoes + qtdInstrucoesTemp) * sizeof(Instruction));
        // Passando as instruções para o vetor que será retornado pela funcão de exponenciação
        for (int j = 0; j < qtdInstrucoesTemp; j++)
        {
            instrucoes[qtdInstrucoes + j] = instrucoesTemp[j];
        }
        qtdInstrucoes += qtdInstrucoesTemp;
        free(instrucoesTemp);
    }

    // Pegando resultado da exponenciação (posição 3), somando com 0 e colocando na posição 2 (posição do resultado). Na prática, alterando a posição do elemento
    qtdInstrucoes++;
    instrucoes = realloc(instrucoes, qtdInstrucoes * sizeof(Instruction));
    instrucoes[qtdInstrucoes - 1].opcode = 1;
    instrucoes[qtdInstrucoes - 1].info1 = 3;
    instrucoes[qtdInstrucoes - 1].info2 = 2;
    instrucoes[qtdInstrucoes - 1].info3 = 2;
```

Imagem 3 : Função de exponenciação

```
171     else // Caso em que trata-se de uma multiplicação que faz parte do processo da função de exponenciação
172     {
173         qtdInstrucoes = 0;
174         for (int i = 0; i < fator1; i++) // For para realizar as sucessivas somas
175         {
176             qtdInstrucoes++;
177             instrucoes = (Instruction *)realloc(instrucoes, qtdInstrucoes * sizeof(Instruction));
178             // Soma
179             instrucoes[qtdInstrucoes - 1].opcode = 1;
180             instrucoes[qtdInstrucoes - 1].info1 = 3; // Posição do elemento fixo; Parcela fixa de cada soma realizada
181             instrucoes[qtdInstrucoes - 1].info2 = 4; // Posição do elemento acumulativo; Acumula as somas anteriores
182             instrucoes[qtdInstrucoes - 1].info3 = 4; // Armazena a informação na posição do elemento acumulativo
183         }
184 
185         qtdInstrucoes++;
186         instrucoes = (Instruction *)realloc(instrucoes, qtdInstrucoes * sizeof(Instruction));
187 
188         // O elemento acumulativo (posição 4) passa a ser o elemento fixo (posição 2) das próximas somas
189         instrucoes[qtdInstrucoes - 1].opcode = 1;
190         instrucoes[qtdInstrucoes - 1].info1 = 4; // Posição do elemento acumulativo
191         instrucoes[qtdInstrucoes - 1].info2 = 2; // Posição do elemento neutro da soma(0)
192         instrucoes[qtdInstrucoes - 1].info3 = 3; // Armazena a informação na posição do elemento fixo
193 
194         qtdInstrucoes++;
195         instrucoes = (Instruction *)realloc(instrucoes, qtdInstrucoes * sizeof(Instruction));
196 
197         // Zera o elemento acumulativo
198         instrucoes[qtdInstrucoes - 1].opcode = 0;
199         instrucoes[qtdInstrucoes - 1].info1 = 0;
200         instrucoes[qtdInstrucoes - 1].info2 = 4;
201     }
202 
```

Imagem 4 : Função de exponenciação na multiplicação

A função de exponenciação possui dois parâmetros, sendo eles:

- base (int): Base da exponenciação;
- expoente (int): Expoente da exponenciação;

Inicialmente, o vetor principal (instruções), do tipo Instruction, é alocado dinamicamente com 5 posições.

Em seguida, a base e o expoente são movidos para a posição 0 e 1 da RAM, respectivamente. Posteriormente, a base é movida para a posição 3 da RAM (parcela fixa da soma) e o 0 (termo neutro da soma) para a posição 2 da RAM. Seguidamente, o valor 0 é colocado na posição 4 da RAM (parcela acumulativa da soma).

A partir daí, duas abordagens possíveis: expoente igual a 0 e expoente diferente de 0.

>Expoente igual a 0

Caso o expoente seja 0, o vetor instruções é realocado com uma posição a mais e o valor 1 é movido para a posição 2 da RAM (resultado). Em seguida, é gerada a instrução de desligar a máquina e o vetor instruções é retornado pela função.

>Expoente diferente de 0

Caso o expoente seja diferente de 0, há um *for*, executado **expoente - 1 vezes**, que aloca um vetor instruçõesTemp, do tipo Instruction, que recebe as instruções oriundas da função de multiplicação. Posteriormente, há um outro *for*, dentro do *for*, executado **base + 2 vezes** - número de instruções geradas pela função de exponenciação. Dentro desse segundo *for*, as instruções do vetor instruçõesTemp são passadas, item a item, para o vetor principal instruções. Ao final do primeiro for, o vetor instruçõesTemp é liberado.

Após o término do *for*, o vetor instruções é realocado em duas posições a mais e o resultado da exponenciação, que, neste momento, está na posição 3 da RAM (parcela fixa de cada soma) é passado para a posição 2 da RAM (resultado). Por fim, a instrução de desligar a máquina é gerada.

- Função da sequência de Fibonacci[**generateFibonacciInstructions()**]

```
311
312     if ((posicaoTermo == 1) || (posicaoTermo == 2)) //Caso em que a posição do termo requerido é 1 ou 2
313     {
314
315         qtdInstrucoes += 2;
316         instrucoes = realloc(instrucoes, qtdInstrucoes * sizeof(Instruction));
317         // Resposta para as posições 1 e 2
318         instrucoes[2].opcode = 0;
319         instrucoes[2].info1 = 1;
320         instrucoes[2].info2 = 2;
321         //Instrução para desligar a máquina
322         instrucoes[qtdInstrucoes - 1].opcode = -1;
323         instrucoes[qtdInstrucoes - 1].info1 = -1;
324         instrucoes[qtdInstrucoes - 1].info2 = -1;
325         instrucoes[qtdInstrucoes - 1].info3 = -1;
326
327         return instrucoes;
328     }
329
330     qtdInstrucoes++;
331     instrucoes = realloc(instrucoes, qtdInstrucoes * sizeof(Instruction));
332     //Colocando 0 (termo neutro da soma) na posição 3
333     instrucoes[2].opcode = 0;
334     instrucoes[2].info1 = 0;
335     instrucoes[2].info2 = 3;
336
337     for (int i = 0; i < posicaoTermo - 2; i++)
338     {
339         qtdInstrucoes++;
340         instrucoes = realloc(instrucoes, qtdInstrucoes * sizeof(Instruction));
341         //Soma os dois termos anteriores e armazena na posição 2
342         instrucoes[qtdInstrucoes - 1].opcode = 1;
343         instrucoes[qtdInstrucoes - 1].info1 = 0;
344         instrucoes[qtdInstrucoes - 1].info2 = 1;
345         instrucoes[qtdInstrucoes - 1].info3 = 2;
346
347         qtdInstrucoes += 2;
348         instrucoes = realloc(instrucoes, qtdInstrucoes * sizeof(Instruction));
349         //Pega o segundo termo e coloca na posição do primeiro
350         instrucoes[qtdInstrucoes - 2].opcode = 1;
351         instrucoes[qtdInstrucoes - 2].info1 = 1;
352         instrucoes[qtdInstrucoes - 2].info2 = 3;
353         instrucoes[qtdInstrucoes - 2].info3 = 0;
354         //O resultado produzido pela soma passa a ser o novo segundo termo
355         instrucoes[qtdInstrucoes - 1].opcode = 1;
356         instrucoes[qtdInstrucoes - 1].info1 = 2;
357         instrucoes[qtdInstrucoes - 1].info2 = 3;
358         instrucoes[qtdInstrucoes - 1].info3 = 1;
```

Imagem 5 : Função da sequência de Fibonacci

A função de Fibonacci possui 1 parâmetro:

- posicaoTermo (int): Posição do termo requerido na sequência de Fibonacci;

Inicialmente, o vetor principal (instruções), do tipo Instruction, é alocado dinamicamente.

Tendo em vista a função acima,observa-se que a variável “posicaoTermo”,refere-se a posição do termo que o usuário deseja na sequência de Fibonacci.

>Primeiro e segundo termo da sequência de Fibonacci:

Quando (posicaoTermo == 1) e (posicaoTermo == 2) há um aumento na quantidade de instruções e uma realocação do vetor (intruções) para gerar o valor 1 na RAM, já que esses dois termos possui o valor 1 e logo a seguir,gera as instruções para o desligamento da máquina e retorna-las.

>A partir do segundo termo da sequência de Fibonacci:

A partir do segundo termo da sequência,há um aumento da quantidade de instruções e a realocação do vetor (instruçoes) para colocar o termo neutro da soma (0) na RAM.A seguir há um bloco de repetição que irá realizar a somas sucessivas para chegar ao valor desejado e ser somado ao termo neutro e ser o novo valor para a próxima soma e a seguir,gerar as instruções para desligar a máquina e retorná-las.

Impressões gerais

De modo geral, a implementação das instruções requisitadas deu-se, inicial e idealmente, de forma “individual”, com cada membro da equipe ficando responsável pelo desenvolvimento de uma parte do código. Entretanto, o fato das instruções dependerem umas das outras, acabou estimulando o processo de integração do time, algo que era esperado, resultando em um harmonioso e eficiente trabalho em equipe para construção do código.

Com relação às questões do trabalho que mais agradaram os membros do grupo, destaca-se o aprimoramento dos conceitos trabalhados em sala de aula,

sobre como a manipulação e processamento de informações se dão dentro da CPU, ainda que a simulação contida no algoritmo seja abstraída e em alto nível.

No tocante às questões do trabalho que mais desagradaram os membros do grupo, ressalta-se os empecilhos relacionados à alocação dinâmica de vetores, como o processo de obtenção do valor correto para realizar a alocação ou o acesso incorreto a posições que não foram alocadas ou puladas. Entretanto, com o uso de ferramentas como o Valgrind, tais empecilhos foram sendo superados ao longo do desenvolvimento do TP.

Análise

Com todas as funções implementadas(divisão, multiplicação, Fibonacci e exponenciação), que por sua vez proporcionaram o resultado esperado, puderam proporcionar para nosso grupo uma perspectiva mais ampla de como os processos matemáticos são realizados em uma CPU real obter uma aproximação mais adequada de como ocorre o processamento em uma CPU.

Conclusão

Durante e após a realização do projetos,percebe-se diversos conhecimentos e conceitos que são adquiridos e reforçados,tendo em vista que no projeto utiliza-se muitos tópicos da linguagem C que são de extrema importância.

Com relação às dificuldades encontradas,foi identificado uma dificuldade em implementar a função de exponenciação,já que era obrigatório o uso da função de multiplicação em sua implementação e havia um entrave quanto à aplicação da lógica.