Relatório da proposta 6

Pontos Flutuantes

Augusto Barbosa Villar Silva - 11805130

Vinícius Barros Alvarenga - 11257564

PCS3616 - Sistemas de programação

22 de Abril de 2022

1 Introdução

No presente relatório foram implementadas rotinas de ponto flutuante, são elas "FLOAT", "SOMA", "SUBTRAI"e "ESCREVE".

A seguir, essas rotinas são descritas, entrando mais profundamente nos algoritmos utilizados e nos raciocínios pensados. Sempre ilustrando com exemplos. Por fim, temos algumas perguntas sobre o código e os algoritmos que foram respondidas para dar mais clareza ao funcionamento deste.

2 FLOAT

Na rotina FLOAT, recebemos um inteiro, guardado na variável VARIAVEL, e o transformamos para um float, de acordo com a fórmula da figura 1.

Esse float do resultado é guardado com suas partes em diferentes variáveis. No caso, SIGNAL, CARACTERISTICA1, CARACTERISTICA2, MANTISSA1, MANTISSA2, MANTISSA3 e MANTISSA4 que podem ser reaproveitadas em outras funções.

Primeiro, nós descobrimos o sinal do número, colocando o valor correspondente em uma variável. Fazemos isso através de um JN e de rótulos que direcionam para cada caso. Após descobrir o sinal passamos para a mantissa e a característica. Note que se o número for negativo, multiplicamos ele por menos um e prosseguimos conforme o caso positivo.

$$(-1)^s \cdot 2^{c-2^{C-1}+1} \cdot (1+f).$$

Figura 1: Fórmula da estrutura de um float

Para isso, aplicamos um algoritmo que calcula a característica e a mantissa de um número, porém, para isso devemos converter o número de hexadecimal para binário. Assim sendo, fizemos uma rotina de conversão.

Nesse método (HEX2BIN), convertemos um número hexa de 4 dígitos em um binário de 16 dígitos: 312A vira 0011 0001 0010 1010. Sendo que cada nibble (conjunto de 4 bits) em binário é guardado em uma variável. Essa função é chamada 4 vezes, uma para cada nibble, guardando o nibble correspondente em uma variável para cada chamada. Agora que temos o número em binário podemos calcular a mantissa e a característica de forma mais simples. Começamos pela característica.

Para o cálculo da característica seguimos o seguinte algoritmo. Analisamos o número de 32 bits do dígito mais significativo para o menos significativo e, ao achar o

primeiro um, guardamos a posição desse número e somamos com o bias, esse número é a característica.

Por exemplo, hex 312A = binario 0011 0001 0010 1010. O bit mais significativo que vale 1 é o bit 13, logo a característica vale 127+13 = 10001101.

No nosso código, fizemos isso com a rotina ENCONTRAPRIMEIROUM que usa de um contador e varre cada uma das variáveis BINARIO1, BINARIO2, BINARIO3 e BINARIO4 bit a bit até achar um dígito 1. Ao achar, ela retorna a sua posição, ou seja,o número que indica em qual digito está esse primeiro um no número completo (forma em binário do número original). Com isso, conseguimos a característica do número, guardando-a numa variável, nos resta encontrar a mantissa.

Para a mantissa, seguimos o mesmo algoritmo. Nele, trocamos o bit 1 mais significativo por um bit 0, o número resultante é a mantissa.

No exemplo, temos que 1 0001 0010 1010 é a mantissa. Sendo que o número em float será então: 0100 0110 1100 0100 1010 1000 0000 0000 = 46C4A800

Na implementação de código fizemos a rotina ENCONTRAMANTISSA que se vale da variável posição e varre as variáveis BINARIO1, BINARIO2, BINARIO3 e BINARIO4 bit a bit até achar o bit encontrado na rotina anterior, substitui-o por 0 e retorna. Nisso, copiamos as variáveis do tipo BINARIO para variáveis do tipo MANTISSA, retornando o número em float correspondente.

3 SOMA

A função soma recebe dois floats e retorna um float que é a soma desses dois floats. Para isso, optamos por transformar os floats de entrada que estão em binário para hexadecimais, somar os hexadecimais e depois converter para binário novamente, retornando o float nessa forma.

Assim sendo, primeiro lemos o float um, com todas suas partes (sinal, característica e mantissa) e o convertemos para a versão hexadecimal, com a função ANTIFLOAT. Fazemos o mesmo para o segundo float e guardamos esses dois em variáveis hexadecimais (PARC1 e PARC2). Somamos essas variáveis hexadecimais e, por fim, chamamos a rotina FLOAT que converte essa soma (PARC1+PARC2) em um número no formato float.

A subrotina ANTIFLOAT recebe um float em suas variáveis de entrada: o sinal, as duas da característica e as quatro da mantissa. O que ela faz é transformar cada conjunto de quatro bits do float em um numero hexadecimal e soma-los com os pesos de cada parte, resultando no valor final em hexadecimal.

Por exemplo, para o float 2^5.1,1110 1000 1010 1111 temos que cada conjunto de quatro bits será tal que, após a conversão, 1110 = E, 1000 = 8, 1010 = A, 1111 = F. Assim sendo, para obter o float em hexadecimal correspondente devemos multiplicar cada número pelo seu peso equivalente, tal qual:

FloatHexa = $2^5.(2^{-4}E + 2^{-8}8 + 2^{-12}A + 2^{-16}F)$ = 1C+1= 1D

Obtendo o número float em hexadecimal, verificamos, por fim, o sinal do número de acordo com a variável de entrada. Se for negativo, multiplicamos por menos um.

A sub-rotina ANTIFLOAT chama a sub-rotina BIN2HEX que converte nibbles binários em números hexadecimais, valendo-se de uma tabela de conversão entre as notações. Por sua vez, a sub-rotina BIN2HEX também se vale de outra sub-rotina, a POT2 que calcula uma determinada potência de 2 a partir de um expoente passado como argumento, essa função é muito útil nas duas anteriores.

4 SUBTRAI

Para a rotina SUBTRAI, basicamente chamamos a rotina SOMA, porém invertemos o valor do segundo termo. Por exemplo: 1-2 = 1+ (-2). Assim, se o sinal do segundo termo for positivo, trocamos por negativo, se for positivo, trocamos por negativo. Em seguida, chamamos pela rotina SOMA e o resultado dela é a subtração entre os dois floats, devolvido como um float.

5 ESCREVE

No desafio, foi implementada a rotina ESCREVE, ela escreve um ponto flutuante no monitor. Para isso, ela escreve o sinal, os 5 caracteres da característica e os 4 caracteres de cada uma das 4 variáveis usadas para guardar o valor da mantissa. Como são escritos em pares, o código junta cada par de caracteres e os imprime em sequência. Assim, seja a forma XXYY, XX se refere ao primeiro caractere YY se refere ao segundo caractere, podendo assumir valores de 30 (zero) e 31 (um) em ASCII, por exemplo. Assim, roda-se para os 11 vezes para a escrita dos 22 bits que utilizamos.

Nessa rotina, nós convertemos cada número hexadecimal no seu correspondente ASCII, para isso, usamos da constante ZEROASCII que representa o valor de 0 na tabela ASCII.Ao somar esse valor ao do caractere desejado obtemos o valor dele na tabela. Após isso, alocamos cada caractere na forma XXYY.

6 PERGUNTAS

6.1 Os algoritmos propostos podem ser mais eficientes? Como? Se sim, por que a forma menos eficiente foi escolhida?

Os algoritmos podem ser mais eficientes. Pode-se perceber, depois que o código estava pronto, que a mantissa, na função FLOAT, poderia ser calculada juntamente com

a característica, mas por conta da funcionalidade e para facilitar a visualização, foi optado por deixar os dois separados.

6.2 Os códigos escritos podem ser mais eficientes? Como? Se sim, por que a forma menos eficiente foi escolhida?

O código poderia ser mais eficiente. Podemos melhorar a rapidez e a quantidade de código implementando alguns looping nele, entretanto para facilitar o entendimento e a própria implementação foi escolhido colocar repetidamente algumas implementações, facilitando a visualização de problemas.

6.3 Qual foi a maior dificuldade em implementar a biblioteca?

A maior dificuldade de implementar a biblioteca foi a parte de conseguir entender o método de como transformar um número em hexadecimal inteiro para a representação de ponto flutuante. Tivemos que concertar muitos bugs durante o período de testagem, uma vez que resolvemos fazer todas as partes primeiro para depois testar.

6.4 O que teria que ser mudado no seu código para poder suportar definição de pontos flutuantes de tipo signed e unsigned?

Se existissem pontos flutuantes unisgned eles iriam de 0 até o número limite da precisão. Logo, não teríamos que tratar os casos com s=1, portanto, não precisaríamos de uma variável para guardar o s. Com isso, muitos casos de teste nos métodos seriam mudados e simplificados, evitando alguns jumps que antes desviavam caso os números fosse negativos. Também algumas multiplicações seriam evitadas, já que elas transformam um número negativo num positivo para ser tratado pelos métodos, como por exemplo o método SUBTRAI, que seria mais enxuto porque não precisaria testar os dois casos dos valores de s para fazer a subtração, mas apenas mudar o sinal do número externamente.

6.5 Se fossem precisos pontos flutuantes com maior precisão, qual seria a organização dessas estruturas e o que teria que ser adicionado no código?

Para isso, seria necessário aumentar o número de iterações do método FLOAT por exemplo, uma vez que ele não foi feito com um loop, mas com código reutilizado. Além disso, seriam necessárias mais variáveis para guardar a mantissa e a característica, ou seja, mais memória.