Relatório EP OCD

Dupla de alunos:

Lucas Moura de Carvalho e Victor Reis.

Sobre a estrutura

Como compilar e rodar:

Dentro da pasta ep, vulgo a pasta que contém todas as classes, execute na linha de comando:

```
javac *.java (compila)
java Main    (executa)
```

Há cinco classes:

- Main - onde é feita a introdução do usuário ao programa e onde é feita a leitura e preparo da entrada do usuário.
- Alu – que implementa as operações de fato, inclui também métodos auxiliares.
- IntFloat – criada para facilitar as operações com inteiros, segurando as informações do mesmo e também metodos auxiliares.
- BinFloat - criada para facilitar as operações com float, segurando as informações do mesmo e também metodos auxiliares.
- Report – classe de testes.

Sobre os testes

Os testes foram feitos na classe Report, do arquivo de mesmo nome, que serviu para guardar muitos dos testes, principalmente os automatizados. Alguns testes podem ser re-feitos com as seguintes linhas:

```
Report.TestSum();
Report.TestSub();
Report.TestDiv();
Report.TestMult();
Report.TestFloatMult();
Report.TestFloatDiv(); ERR
```

Report.TestFloatSum();

Para os inteiros a maioria dos testes foram automatizados e dividido e quatro métodos:

- static void TestSum()

1. Teste de complemento de dois. Complemento(IntA(-i)) = i.
2. Teste de range de soma de inteiros. IntA + i : i = [-128, 127].
3. Testando complemento de dois. Complemento(IntA(i)) = -i.
4. Teste de range de soma de inteiros. IntA + IntB, com todas as combinacoes de IntA e IntB : IntA, IntB = [-128, 127].

   Isso dentro de um loop, e tambem foi testado o overflow, comparando quando poderiam ou não ter dado.

   Nessa categoria todos os testes passaram.

- static void TestSub()

1. Teste de range de soma de inteiros. IntA - IntB, com todas as combinacoes de IntA e IntB : IntA, IntB = [-128, 127].

   Isso dentro de um loop, e tambem foi testado o overflow, comparando quando poderiam ou não ter dado.

   Quase todos os testes passaram, exceto que para qualquer numero - valor minimo gera overflow nao pego. Problema que Complemento(valor min) = valor min, e.g, C(-128) = -128, logo as subtrações foram atrapalhadas. Isso poderia ser evitado ao impedir que se faça IntA – valor min.

- static void TestMult()

1. Teste de range de soma de inteiros. IntA * IntB, com todas as combinacoes de IntA e IntB : IntA, IntB = [-128, 127].

   Isso dentro de um loop, e tambem foi testado o overflow, comparando quando poderiam ou não ter dado.

Quase todos os testes passaram, exceto que para qualquer numero IntB, a multiplicação valor min * intB gera sinal errado. Isto é devido a uma falha na implementação do algoritmo, pois mais uma vez C(valor min) = valor min.

O artigo da wikipedia de onde o algoritmo foi descrito também apresenta um a lgoritmo alterntivo que corrige esse problema.

- static void TestDiv()

1. Teste range de soma de inteiros. IntA / IntB, com todas as combinacoes de IntA e IntB : IntA, IntB = [-128, 127].

   Isso dentro de um loop, e tambem foi testado o overflow, comparando quando poderiam ou não ter dado.

   Nessa categoria todos os testes passaram.

O numero de 8 bits foi escolhido por ser rapido de se executar, também não há motivos para suspeitar que não funcione com numero de bits diferentes.

Para os float, foram outros três métodos, além de alguns testes pre-eliminares:

- Padrão em memória

1. O teste de sinal foi trivial.

2. Os testes de expoente utilizaram do construtor padrão, que recebe um inteiro no range -127, 128 e guarda como expoente em forma de BinInt.

   Foi verificado se na memoria o numero aparecia acrescido de 127, como dita o excesso, tambem foram verifcados overflow e underflow de expoente em casos de ponta, i.e números menores que -127 e maiores que 128.

   Foram comparados com os mesmos floats guardados em memoria no java, com auxilio dos métodos:

   Integer.toBinaryString(Float.floatToRawIntBits(floatA)).

Estes testes passaram com sucesso.

- TestFloatSum()

    1. Foram testadas as somas (manualmente) de valores intermediarios, de borda, underflow e underflow tanto no expoente quanto na mantissa, na segunda utilizando também de combinações diferentes de sinais. Os resultados foram verificados olho a olho em suas representações científicas de entrada e saida.

       Foram testados primeiro a corretude das mantissas, que mantiveram valores correto mesmo trocando de soma para subtração, isto é, considerando a forma de arredondamento (truncando).

       Para a soma, foi verificado o aumento do expoente na hora de normalizar, e o mesmo para subtração, que gera valores sub normais.

       Exemplos:

       floatA = new BinFloat(0, 127, 2);
       floatB = new BinFloat(0, 127, 5);
       floatA = new BinFloat(0, 127, 5);
       floatB = new BinFloat(0, 127, 2);
       floatA = new BinFloat(0, 127, 0b11111111111111111111111);
       floatB = new BinFloat(0, 127, 0b11111111111111111111111);

       Passando em todos os realizados

- TestFloatMult()

    1. Os testes foram realizados da mesma maneira que para soma de float. Alguns exemplos de testes são valores:

    floatA = new BinFloat(0, 1, 1);
    floatB = new BinFloat(0, 2, 1);
     floatA = new BinFloat(0, -1, 1);
    floatB = new BinFloat(0, -2, 1);
    floatA = new BinFloat(0, 5, 1);
    floatB = new BinFloat(0, -2, 1);
    floatA = new BinFloat(0, 127, 1);
    floatB = new BinFloat(0, 1, 1);
    floatA = new BinFloat(0, -126, 1);
    floatB = new BinFloat(0, -1, 1);
    floatA = new BinFloat(0, -127, 1);
    floatB = new BinFloat(0, -1, 1);

floatA = new BinFloat(0, 127, 1);
floatB = new BinFloat(0, 2, 1);

floatA = new BinFloat(0, 1, 0b01);
floatB = new BinFloat(0, 1, 0b01);
floatA = new BinFloat(0, 1, 0b10000000000000000000000);
floatB = new BinFloat(0, 1, 0b10000000000000000000000);
floatA = new BinFloat(0, 1, 0b11111111111111111111111);
floatB = new BinFloat(0, 1, 0b11111111111111111111111);
floatA = new BinFloat(0, 1, 0b01000000000000000000000);
floatB = new BinFloat(0, 1, 0b01000000000000000000000);

Executando Alu.MultFloat(floatA, floatB);

Passando em todos os realizados.

- TestFloatDiv()

    1. Os testes de divisao foram feitos de maneira extremamente similar aos testes feitos para a multiplicação, utilizando os mesmos valores. Primeiro no expoente e no sinal, os quais passaram com sucesso.

       Na mantissa o método de divisão falha em mostrar um valores diferentes de 1 ou 0, providos da divisão de inteira.

       Foi tentado aumentar as casas, utilizando dos sucessivos restos das d ivisões inteiras e correspondente ao left shift do mesmo (acrescimo do zero a direita em uma divisao “de chave”), e então escrevendo 1 ou 0 e dando right shift na mantissa final.

       Devido a complicações na utilização do algoritmo de divisão inteira, além da necessidade de constante conversão de formatos intermediarios que se gerou o problema, que só se resolveria executando o procedimento descrito acima de forma correta.

Fontes:

Algoritmo de Booth
https://en.wikipedia.org/wiki/Booth%27s_multiplication_algorithm