COL216   LOKESH ACHARYA   Assignment 11

The main components of the program are:

void **FETCH**(vector<pair<string,string>> v, int PC)
void **CMPR_EXP**(string n1, string n2, int &PC, int &cycles);
void **ADD**(string a, string b, int &exp, int &PC, int &cycles);
void **NRMLS**(string &r, int &exp, int &PC, int &cycles);
void **RNDOFF**(string &r, int &exp, int &PC, int &cycles);
Described below

While supporting components are:

string **shrgt**(string &s, int n) : for rigth shifting
string **badd**(string a, string b) : for binary add
string **bsub**(string a, string b) : for binary sub
int **loadfile**(vector<pair<string,string>> &inst) : for loading instructions

Steps -->

1. The main program calls the loadfile function which fills the add ooperands in a vector of pair<string, string>.

2. Call the FETCH function which loads the add operands and passes to the exponent comparing function CMPR_EXP

3. Check the exponents of the ooperands
if any of the operands takes the reserve value of INFINITY the program gives output as INFINITY , NEG INFINITY or NaN values depending on the operands
if the exopents the ordinary value the we compare the exponent and adjust the binary point of the operand with lower exponent and pass the fraction components of the operands in the ADD function after adding appropriate no. Of zeros in starting of the lower operand.

4. The add function adds the given two strings in the order of (larger, smaller) and passes in badd function if sign of bothe operant are same. Otherwise to the bsub function. Then passes on to the NRMLS function.

5. The normalization function normalises the value with adjusting the exponent of the result and passes the result to the RNDOFF function.

6. The RNDOFF function rounds of the string according to the “`round to the nearest; ties to even`” method described as:

The general rule when rounding binary fractions to the **n**-th place prescribes to check the digit following the **n**-th place in the number. If it’s **0**, then the number should always be rounded down. If, instead, the digit is **1** and any of the following digits are also **1**, then the number should be rounded up. If, however, all of the following digits are **0**’s, then a tie breaking rule must be applied and usually it’s the ‘*ties to even*’. This rule says that we should round to the number that has **0** at the **n**-th place.

To demonstrate those rules in action let’s round some numbers to **2** places after the radix point:

- **0.11001** — rounds down to **0.11**, because the digit at the **3**-rd place is **0**
- **0.11101** —rounds up to **1.00**, because the digit at the **3**-rd place is **1**and there are following digits of **1** (**5**-th place)
- **0.11100** — apply the ‘ties to even’ tie breaker rule and round up because the digit at **3**-rd place is **1** and the following digits are all **0**'s.

(SRC: https://indepth.dev/how-to-round-binary-numbers/)

If the rounded off value has string string size of 24 (‘1’ + fraction bit) then give the result of this otehrwise pass again to the Normalisation function.

Explaination of the test cases:

t0: In this file the first operand is same for all 00111111100110011000000000000000
with exp=127 and positive
In these five cases the exp of all the ooperands is same and are positive:

Inst1:
exp1: 01111111 exp2: 01111111
exp1: 127 exp2: 127
sign1: 0 sign2: 0
frac1: 00110011000000000000000 frac2: 00111100000000000000000
significand add: 1001101111000000000000000
exp: 128
Fraction of result: 00110111100000000000000

01000000000110111100000000000000
No of cycles taken: 4

Inst2:
exp1: 01111111 exp2: 01111111
exp1: 127 exp2: 127
sign1: 0 sign2: 0
frac1: 00110011000000000000000 frac2: 11000111000000000000000
significand add: 1011111010000000000000000
exp: 128
Fraction of result: 01111101000000000000000

01000000001111101000000000000000
No of cycles taken: 4

Inst3
exp1: 01111111 exp2: 01111111
exp1: 127 exp2: 127
sign1: 0 sign2: 0
frac1: 00110011000000000000000 frac2: 10101010000000000000000
significand add: 101101110100000000000000
exp: 128
Fraction of result: 01101110100000000000000

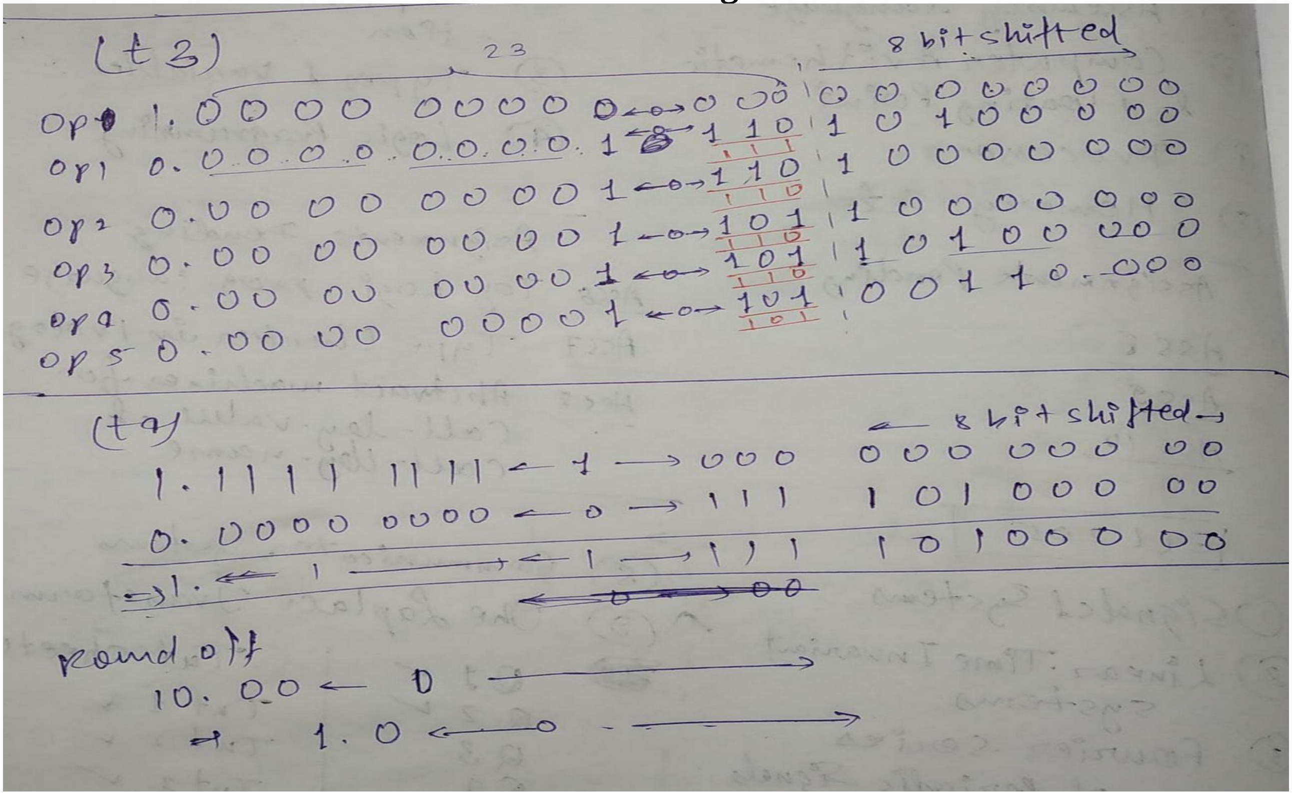
01000000001101110100000000000000
No of cycles taken: 4

t1: addditions involving infinity numbers

t2: additions involving zero

t3. round off without normalized condition changed

t4: round off with normalized condition changed


(t 3)
23
8 bit shifted
op0 1. 0 0 0 0 0 0 0 0 0 0←0→0 0 0 | 0 0 0 0 0 0 0 0
op1 0. 0 0 0 0 0 0 0 0 0 1←→1 1 0 | 1 0 1 0 0 0 0 0
op2 0. 0 0 0 0 0 0 0 0 0 1←0→1 1 0 | 1 0 0 0 0 0 0 0
op3 0. 0 0 0 0 0 0 0 0 0 1←0→1 0 1 | 1 0 0 0 0 0 0 0
op4 0. 0 0 0 0 0 0 0 0 0 1←0→1 0 1 | 1 0 1 0 0 0 0 0
op5 0. 0 0 0 0 0 0 0 0 1←0→1 0 1 | 0 0 1 1 0 . 0 0 0
111
110
110
110
101
(t4)
← 8 bit shifted →
1. 1 1 1 1 1 1 1 1 ← 1 → 0 0 0   0 0 0 0 0 0 0 0
0. 0 0 0 0 0 0 0 0 ← 0 → 1 1 1   1 0 1 0 0 0 0 0
→1. ← 1 → ← 1 → 1 1 1   1 0 1 0 0 0 0 0
← 0 → 0 0
round off
10. 0 0 ← 0 →
→ 1. 0 ← 0 →

t3 test file contains cases of rounding off.
t4 test file contains cases where there is need to normalise again.