Niet gehele binaire getallen

De voorstelling van een kommagetal is eigenlijk zoals een gewoon getal want het getal meest links van de komma is de ½ bit daarna de ¼ bit en zo verder 1/8 1/16 1/32 dus eingenlijk net het zlefde alleen dan dat het getal in een breuk staat met een 1 er boven

13,125

13 => 8 +4 +1

1101,001

Als je een groter getal heb bv 116,46484375 dan kan je ook proberen dat uit te tellen maar dat is bijna onmogelijk daarom is er een simpelere methode je pakt je kommagetal en je doet het maal 2 als de uitkomst meer is dan een 1 dan schrijf je links een één en recht de rest van de uitkomst als je een nul uitkomt doe je hetzelfde maar schrijf je een 0. Dat doe je dan tot het rechte getal een 0 is en dan lees je het linkse getal van boven naar onder(dus tegenovergestelde van de regel voor een geheel getal.)

Tweede manier die meestal sneller is , is je pakt het kommagetal en doet dat \*16 hierbij doe je het zelfde als bij 2 je schrijf je geheel getal links op en het kommagetal schrijf je rechts. Op het einde bekom je een getal dat getal lees je ook van boven naar onder en omdat het een hexadecimaal getal is kan je het simpel omzetten van een getal naar binair dus in dit voorbeeld is dat 77 dus pak je de eerste 7 en zet je die om naar binair dat is dan 0111 en met de tweede 7 het zelfde en dan bekom je 01110111 en dat is dan 0.46484375 in binair en zet je het achter de 116

116.46484375== 0111 0100,01110111

116 = 64 +32+16+4 = 0111 0100

\*2 46484375

-----------------------------

0 9296875

1 859375

1 71875

1 4375

0 875

1 75

1 5

1 0

\*16 46484375

-----------------------------

7 4375

7 0

77 => 0111 0111

(1+2+4+16+32+64)/256 => 46484375

Om van een binair getal naar een decimaal getal te gaan moet je het getal moet je het binair getal van rechts naar links lezen alsof het een geheel getal is en dan delen door de volgende bit die niet in het bereik is

Floating point getallen

Als er een floating point notatie is het altijd dezelfde notatie en is die altijd volgends de zelfde regels in een 8 bit notatie is die redelijk eenvoudig

0 11 11111

De eerste 0 staat voor dat het getal een positief getal is als die bit een 1 zou zijn dan zou het een getatief getal zijn. De volgende twee getallen geven de weteschappelijke schrijfwijze van het getal weer in dit getwal staart er 3 maar in een 8 bit bereik is het geschreven in een +2 notatie en wilt dat getal zeggen 10^1 . De overige 5 bits zijn voor het getal zelf dus in dit geval zout dat willen zeggen dat het een negatief komma getal is 11.111 dus => 3.875

1+2+4/8

Maar als het getal 0 11 11.011 zou zijn dan moet je het zelfde doen maar doen moet je tellen1+2/8 dus je moet altijd delen door het grootste getal links.

Dus ook al heb je een negatief getal bv 10110101 dan wilt zeggen dat je een negtief getal hebt met 10 ^-1 dus 0.110101 maar omdat dit een kommagetal is moet je het niet omzetten en kan je heb gewoon op de zelfde manier omzetten als een positief getal dus 1+4+16+32 /64 = -0.828125

IEEE- standaard

De enigste die je op dit moment moet kennen is de 32 bit notatie van een kommagetal

Hier is je eerste bit links de tekenbit als het een 1 is, is het een negatief getal als het een 0 is is het een positief getal dan daarna komen de 8 bits die bepalen de plaats van de komma dit is een plus 127 notatie dus als het getal van die 8 bits onder de 0 is dan is het 10 ^-.. en verschuift de komma naar links. als het hoger is dan is 127 dan is het een verschuift de komma naar rechts. Daarna komen nog 23 bits die bepalen de cijfers waar je dus ook de eerste 1 weg laat als hidden bit dus voor een “eenvoudig” voorbeeld

128.03125

\*16 03125

0 5

8 0

80 1000 => 0001

1000 000, 00001 => dus is het getal 1.000000000001\*2^7

Het getal is positief dus is het eerste getal een 0

De plus 127 notatie is 127+7 = 134 => 10000110 want het is tot de 7de

Het getal zonder de hidden bit is 000000000001

Dus is de float notatie

0100001100000000000100000000000

Enkele uitzonderingen

+ ∞ = 01111111100000000000000000000000 => eerste bit op 0 volgende 8 bits op 1 en de volgende 23 bits op 0

-∞ = 11111111100000000000000000000000 => eerste bit op 1 volgende 8 bits op 1 en volgenden 23 bits op 0

NaN(not a number) = eerste bit is bijde mogelijk zowel 0 of 1 de volgende 8 bits allemaal op 1 en de volgende 23 bits kan elke combinatie behalve allemaal op 0 dus bv

1 1111111100000000000000000000100

+0 = 00000000000000000000000000000000 = alle 32 bits op 0

-0 = 10000000000000000000000000000000= eerste rechte bit op 1 en al de rest (31) op 0

Gedenormalisserd(een getal dat zo klein is dat gelijk gesteld word aan 0) eerste bit kan zowel 0 als 1 zijn de volgende 8 bits zijn 8 0en en daar na gaat elke combinatie dus bv

10000000010101010101011110000101

Dit is omdat het getal dat je gaat krijgen 1.0\*2^-127 en dat getal is zo klein dat er is afgsproken dat het een gedenormaliseerd getal is

BCD getallen

Hier worden de getallen per getal gepakt dus 28 in bcd 0010 1000 of 4197 0100 0001 1001 0111

Er zijn 3 waarde die geen getal zijn:

1111 = het getal is niet gekend

1100 = + teken dus het getal is positief

1101 = -teken het getal is negatief

Dus 256/

Is 0010 0101 0110 1111

256-

0010 0101 0110 1101

Maar zonder een zoned decimaal formaat is het

11110010 11110101 11000110

Conclusie je moet elk getal apart bekijken en dan omzetten naar een 4 bit notatie behalve als je zit met een zoned decimaal formaat dan moet je het in groepjes van 8 bits zetten voor teken er bij te hebben