IEEE 754 Single precision format

TEORI MED EXEMPEL

Lasse Karagiannis

2016-12-19

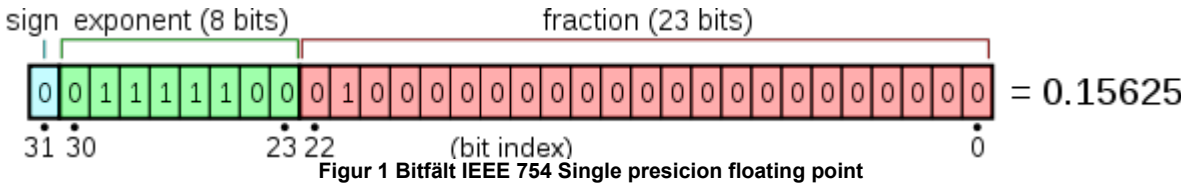
Sammanfattning Teknisk rapport om IEEE 754 Single precision floating point

IEEE 754 Single precision floating point format representation presenteras med teori och exempel.och teori med exempel presenteras presenteras i denna rapport som på ett pedagogiskt sätt presenterar. Ett C-program som skriver ut bitfältet som en textsträng presenteras. Programmet har skrivits av Jens Björnhager, lärare på en EC-utbildning i Inbyggda System.

Innehållsförteckning

1 IEEE 754 Single precision floating point

Kodningen av ett decimal tal görs med 32 bitar där bit 31 är teckenbiten, bitarna 22 till 0 är en muliplicerande faktor den s.k. mantissan, och exponenten utgörs av bitarna 30 till 23, Se Figur 1



Figur 1 Bitfält IEEE 754 Single presicion floating point

Exponenten skall ses som ett unsigned 8 bitars tal med s.k. *bias*, dvs. det ska subtraheras med 127 för att få en korrekt mening.
I exemplet från Figur så ska alltså exponenten 0111 1100 = 124, uppfattas som 124-127 = -3.

Mantissan (*fraction*) ska ses som 1.b22 b21....b0, om inte exponeneten lagrats med enbart nollor.

Flyttalets värde fås genom följande matematiska uttryck

$$(-1)^{b_{31}} * 1.b_{22}b_{21}...b_0 * 2^{b_{30}b_{29}...b_{23}-127}$$

I exemplet ovan så fås alltså talet genom att multiplicera följande tre faktorer:

$(-1)^0 = 1$
$1.b_{22}\ b_{21}...b_0 = 1.25$
$2^{(124-127)} = 2^{-3} = 1/8 = 0.125$

Således fås 1*1.25*0.125 = 0.15625

2 Fler exempel

0.25 kodas som

0 01111101 00000000000000000000000

Teckenbiten är 0
Exponenten är 125, men ska ses som 125 – 127 = -2
Mantissan är 0, men ska ses som 1.0

Det matematiska uttrycket blir $1.0*2^{-2} = 0.25$

3 Verifiering med C-program

Program från tidigare utbildning har används för att verifiera IEEE-754 standarden. Författaren är Jens Björnhager från MotionControl AB.

För att köra detta program på enklast möjliga sätt så rekommenderas *CodeBlocks*.
Välj codeblocks-16.01mingw-setup.exe , så fås nedladdning och automatisk installation av mingw kompilatorn.

Programmet väntar på användarens input av ett decimaltal, skrivet med decimalpunkt, och skriver ut bitfältsrepresentationen på skärmen.

Programmet använder scanf, som kräver att adressen till den variabel den ska läsa till har samma typ som formateringsflaggan, vilket i C-syntax åstadkommes såsom *&f*.

Adressen *&f* är av typen ”float-pekare” dvs. *float **, som måste typkastas till en ”int-pekare” *int ** för att kompilatorn ska tillåta bitfältsoperationer, vilket åstadkommes med uttrycket *(int *)&f*;

Detta har nu typen ”int-pekare” *int ** som måste derefereras, dvs. vi vill komma åt det som pekaren pekar på och tilldela detta till en int variabel, variablen i, så därför blir hela uttrycket för tilldelningen:
*i = *(int *) &f;*

När man har bitfältet som en int, så kan man således använda bitvisa operationer.
Mask är en 32-bitars *unsigned int* med värdet b1000..0, vars MSB skiftas åt höger med uttrycket mask >> = 1, vilket är en kompaktare form av uttrycket mask = mask >> 1.
Hade denna inte varit *unsigned* så hade ettor skiftats in under högerskift.

Därefter körs en select-sats där bitvis AND görs mellan variabeln i och mask. Om resultat resulterar i TRUE, så skrivs '1' ut på skärmen med funktionen *putchar* annars '0'. Resten är textformatering för att urskilja teckenbiten, exponenten och mantissan.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
    float f;
    int n,i;
    unsigned int mask;
    scanf("%f",&f);
    i = *(int *)&f;
    n = 0;
    for (mask = 0x80000000;mask != 0;mask>>= 1){
        if((n==1) || (n==9))
            putchar(' ');
        putchar((i & mask)? '1':'0');
        n++;
    }
    printf("\n");
    return 0;
}
```

4 Referenser

https://en.wikipedia.org/wiki/Single-precision_floating-point_format