**Single Precision Representation**

**EXP NO: 35**

**AIM:** To write a C program to implement Single Precision Representation

.

**ALGORITHM:**

1. \*\*Represent the Number in Binary:\*\*

- If the number is not an integer, convert the fractional part to binary as well.

2. \*\*Normalize the Binary Representation:\*\*

- Express the binary representation in the form \(1.xxxxx... \times 2^n\).

3. \*\*Determine the Sign Bit:\*\*

- If the number is negative, set the sign bit to 1; otherwise, set it to 0.

4. \*\*Express the Exponent:\*\*

- Determine the biased exponent. For single precision, the bias is 127. Add the bias to the exponent obtained from the normalization step.

5. \*\*Encode the Exponent:\*\*

- Represent the biased exponent in binary using 8 bits.

6. \*\*Encode the Mantissa:\*\*

- Take the fractional part obtained after normalization and pad it with zeros to the right if necessary.

- Take the first 23 bits of the mantissa.

7. \*\*Combine the Sign Bit, Exponent, and Mantissa:\*\*

- Concatenate the sign bit**,** exponent, and mantissa to get the 32-bit single-precision floating-point representation.

**PROGRAM:**

#include<stdio.h>

void printbinary(int n,int i){

int k;

for(k=i-1;k>=0;k--){

if((n>>k)&1){

printf("1");

}else{

printf("0");

}

}

}

typedef union{

float f;

struct{

unsigned int mantessa:23;

unsigned int exponent:8 ;

unsigned int sign:1;

}raw;

}myfloat;

void printieee(myfloat var){

printf("%d |",var.raw.sign);

printbinary(var.raw.exponent,8);

printf("|");

printbinary(var.raw.mantessa,23);

printf("\n");

}

int main(){

myfloat var;

var.f=1259.125;

printf("ieee representation of 784 %f is \n",var.f);

printieee(var);

return 0;

}

**INPUT:**

**Var.f=**1259.125

**OUTPUT:**

**A screenshot of a computer

Description automatically generated**

**RESULT:** Thus the program was executed successfully using DevC++.