```
typedef union{
        float   floating_value_in_32_bits;
        struct  sign_exp_mantissa{
                unsigned  mantissa:23;
                unsigned  exponent:8;
                unsigned      sign:1;
        } f_bits;
        struct single_bits{
                unsigned  b0 :1;
                unsigned  b1 :1;
                        :
                        :
                unsigned  b30:1;
                unsigned  b31:1;
        }bit;
}FLOAT_UN;
```

```
int main(int argc, char * argv[])
{

FLOAT_UN float_32_s1,float_32_s2,float_32_rslt, fun_arg;

/******** local helper variables ***********/

float       the_hardware_result;
int         mant_s1, mant_s2, mant_res, exp_s1, exp_s2;
int         i, j, k, shift_count;
int         de_norm_s1 = TRUE, de_norm_s2 = TRUE;

/******** request two floating point numbers ********/

printf("please enter your first  floating point number and new-line: ");
scanf("%g", &float_32_s1.floating_value_in_32_bits);

printf("please enter your second floating point number and new-line: ");
scanf("%g", &float_32_s2.floating_value_in_32_bits);

/****** generate the floating point hardware result ********/

the_hardware_result = float_32_s2.floating_value_in_32_bits +
                      float_32_s1.floating_value_in_32_bits;
```

```
/******* get the mantissa and exponent components ****/
/*******       into the helper variables        ****/

mant_s1 = float_32_s1.f_bits.mantissa;
mant_s2 = float_32_s2.f_bits.mantissa;
exp_s1  = float_32_s1.f_bits.exponent;
exp_s2  = float_32_s2.f_bits.exponent;

/******* check for normalization and slam in the *****/
/*******    hidden bit if normalized          *****/
if(exp_s1){
      mant_s1 |= (1 << 23);
      de_norm_s1 = FALSE;
}
if(exp_s2){
      mant_s2 |= (1 << 23);
      de_norm_s2 = FALSE;
}
```

```
/******* check exponent diff and who's the smallest ****/

if((shift_count = exp_s1 - exp_s2) < 0){
      shift_count = -(shift_count);    /* keep diff + */
      if(shift_count > 24)shift_count = 24;
      if(shift_count >= 1 && de_norm_s1){
         mant_s1 = (mant_s1 >> shift_count-1);
      }else{
         mant_s1 = mant_s1 >> shift_count;}
      float_32_rslt.f_bits.exponent = exp_s2;
}else{
      if(shift_count > 24)shift_count = 24;
      if(shift_count >= 1 && denom_s2){
        mant_s2 = (mant_s2 >> shift_count-1);
      }else{
        mant_s2 = mant_s2 >> shift_count;}
      float_32_rslt.f_bits.exponent = exp_s1;
}
```

```
/**** finally ready to add helper mantissa variables ****/

mant_res = mant_s1 + mant_s2;

/**** check if the addition overflowed 24 bits, since  ****/
/**** mantissa with hidden bit can only be 24 bits    ****/
/**** if we need to right shift, must increase the exp ****/
/**** finally clear the slammed hidden bit in the      ****/
/**** mantissa helper to get to 23 bits and put these ****/
/**** 23 bits into the mantissa bit field of the result  ****/

if(mant_res & (1<<24)){
      mant_res >>= 1;
      float_32_rslt.f_bits.exponent++;
      printf("\n2 HIDDEN BITS, MUST INCREMENT
EXPONENT\n");
      float_32_rslt.f_bits.mantissa = (mant_res & ~(1<<23));
}else{
      float_32_rslt.f_bits.mantissa = (mant_res & ~(1<<23));
}
/**** check for infinity exponent pattern (0xFF)          ****/
/**** cannot have NAN from addition, so clear mantissa ****/
```