CISP 440

Terrence Jackson

Homework 1.42

Floating Point Implementation

Description:

A program to perform “home made” floating point functionality using only integer operations.

Source Code:

addFloats

```
/*
Adds two 8-bit floating point numbers.
Puts them into 16 bit buffers, denormalizes them
Adds them as integers, normalizes the result.
Only works for positive numbers.
Based more or less kinda on:
http://pages.cs.wisc.edu/~smoler/x86text/lect.notes/arith.flpt.html

Example:        2.125  +          0.125  =          2.25
Binary:  1.0001 x 2^1  +  1.0000 x 2^-3  =  1.0010 x 2^1
Packed:    0 0001 101  +     0 0000 001  =    0 0010 101

Unpacked:          0000 0010.0010 0000
                 + 0000 0000.0010 0000
                 = 0000 0010.0100 0000
*/
unsigned char addFloats(unsigned char f1, unsigned char f2)
{
   unsigned char theFloat = 0;   // the answer to return

   /*
   ////////////// f1 //////////////
   repack the bits to a 'bit field' and de-normalize it
    - get the exponent
    - mask off everything except mantissa
    - put on the leading 1
    - scoot into high byte
    - de-normalize
```

```
Description from the Instructor Notes:
    unpack each 8 bit format to 16 bit buffers
    align the imaginary decimal points "fixed point denormalized form"
*/
int exponent1;
exponent1 = (f1 & 0x07) - 4;    // get the exponent
f1 &= 0x78;                     // mask off everything except mantissa
f1 |= 0x80;                     // put on the leading 1

// now pack the normalized bits to a 'bit field' so we can denormalize
unsigned short buffer1 = 0;
buffer1 = f1;
buffer1 <<= 8;                          // scoot into high byte
buffer1 >>= (7 - exponent1);            // de-normalize

// get the whole part
unsigned char i_whole1;                 // bits to left of decimal
i_whole1 = (buffer1 & 0xFF00) >> 8;

// get the fractional part
unsigned char b_fract1;                 // bits to right of decimal
b_fract1 = (buffer1 & 0x00FF);

printf("A expanded: ");
print16bits(buffer1);
printf("\n");

/*
////////////// f2 /////////////
DO AGAIN FOR SECOND NUMBER
*/
int exponent2;
exponent2 = (f2 & 0x07) - 4;     // get the exponent
f2 &= 0x78;                      // mask off everything except mantissa
f2 |= 0x80;                      // put on the leading 1
```

```
// now pack the normalized bits to a 'bit field' so we can denormalize
unsigned short buffer2 = 0;
buffer2 = f2;
buffer2 <<= 8;                          // scoot into high byte
buffer2 >>= (7 - exponent2);            // de-normalize

// get the whole part
unsigned char i_whole2;                 // bits to left of decimal
i_whole2 = (buffer2 & 0xFF00) >> 8;

// get the fractional part
unsigned char b_fract2;                 // bits to right of decimal
b_fract2 = (buffer2 & 0x00FF);

printf("B expanded: ");
print16bits(buffer2);
printf("\n");

// add them with ordinary integer addition!
unsigned short buffer3 = 0;
buffer3 = buffer1 + buffer2;
printf("Sum:        ");
print16bits(buffer3);
printf("\n");

/*
////////////// Packing the Sum /////////////
Find the first 1, then shift and extract m and e bits from the sum
Pack the sum's m and e into 8 bit format
*/

// normalize - find the first 1 from left to right
int exponent3 = 7;
unsigned short mask2 = 0x8000;
while(!(buffer3 & mask2)) {
    exponent3--;
    mask2 >>= 1;
}
```

```
    // overflow check - if exponent is greater than 3
    if(exponent3 > 3){
        printf("atof conversion Overflow of sum");
        exit(0);
    }

    // underflow check - if exponent is less than -4
    if(exponent3 < -4){
        printf("atof conversion Underflow of sum");
        exit(0);
    }

    // final packing
    buffer3 <<= (7 - exponent3);                // align mantissa
    buffer3 >>= 8;                              // scoot into low byte
    theFloat = buffer3;                         // pack the mantissa
    theFloat &= 0x78;                           // mask off the other stuff
    exponent3 += 4;                             // the excess 4 thing
    theFloat |= exponent3;                      // insert the exponent
    // not worrying about the sign bit for this implementation

    printf("Packed Sum: ");
    print8bits(theFloat);
    printf("\n");

    return theFloat;
}
```

multiplyFloats

```
/*
Multiplies two 8-bit floating point numbers.

Puts them into 16 bit buffers, NORMALIZED
Multiplies them as integers
Normalizes the result
Adds the exponents.

Based more or less kinda on:
http://pages.cs.wisc.edu/~smoler/x86text/lect.notes/arith.flpt.html

Example:          1.5  *           1.5  =        2.25
Binary:  1.1000 x 2^0  *  1.1000 x 2^0  =  1.0010 x 2^1
Packed:    0 1000 100  *    0 1000 100  =    0 0010 101

Unpacked NORMALIZED:          0000 0001.1000 0000  e0
                      *       0000 0001.1000 0000  e0
                      = 0010.0100 0000 0000 0000  e1 = e0 + e0 +
e1rollover

   normalize          =       0000 0001 0010 0000  e1
*/
unsigned char multiplyFloats(unsigned char f1, unsigned char f2)
{
   unsigned char theFloat = 0; // the answer to return
   int negative = 0;
   if((f1 & 0x80) xor (f2 & 0x80)) negative = 1;

   /*
   ////////////// f1 ////////////
   repack the bits to a 'bit field' and keep it normalized
    - get the exponent
    - mask off everything except mantissa
    - put on the leading 1
    - scoot into normal position
```

```
Description from the Instructor Notes:
    Unpack each 8 bit format to 32 bit buffers: A,B.
    Leave in "fixed point normal form"
*/
int exponent1;
exponent1 = (f1 & 0x07) - 4;     // get the exponent
f1 &= 0x78;                      // mask off everything except mantissa
f1 |= 0x80;                      // put on the leading 1

// now pack the normalized bits to a 'bit field' and move to normal
   position
unsigned short buffer1 = f1;
buffer1 <<= 1;

printf("A expanded: ");
print32bits(buffer1);
printf(" e1 = %d" , exponent1);
printf("\n");

/*
////////////// f2 /////////////
DO AGAIN FOR SECOND NUMBER
*/
int exponent2;
exponent2 = (f2 & 0x07) - 4;     // get the exponent
f2 &= 0x78;                      // mask off everything except mantissa
f2 |= 0x80;                      // put on the leading 1

// now pack the normalized bits to a 'bit field' and move to normal
   position
unsigned short buffer2 = f2;
buffer2 <<= 1;

printf("B expanded: ");
print32bits(buffer2);
printf(" e2 = %d" , exponent2);
printf("\n");
```

```
// multiply them with ordinary integer multiplication!
unsigned buffer3;
buffer3 = buffer1 * buffer2;

printf("Product:    ");
print32bits(buffer3);
printf("\n");

////////////// normalize the result /////////////
// check if answer is >= 2
// normalize and set rollover value
int rollover = 0;
if(buffer3 & 0x20000) rollover = 1;
buffer3 >>= (9 + rollover);

// calculate exponent for answer
int exponent3 = exponent1 + exponent2 + rollover;

printf("Norm Prod:  ");
print32bits((buffer3 << 1)); //buffer is currently ready to pack, but
                              normalized form is shifted back 1
printf(" e3 = %d" , exponent3);
printf("\n");

// overflow check - if exponent is greater than 3
if(exponent3 > 3){
    printf("atof conversion Overflow of sum");
    exit(0);
}

// underflow check - if exponent is less than -4
if(exponent3 < -4){
    printf("atof conversion Underflow of sum");
    exit(0);
}
```

```
    // final packing
    theFloat = buffer3;                     // pack the mantissa
    theFloat &= 0x78;                       // mask off the other stuff
    exponent3 += 4;                         // the excess 4 thing
    theFloat |= exponent3;                  // insert the exponent
    if(negative) theFloat | 0x80;           //sign bit

    printf("Packed Prod: "); print8bits(theFloat); printf("\n");
    return theFloat;
}
```

main

```
int main()
{
   char strIn1[40] = "0.125";
   char strIn2[40] = "4.5";
   char strOut[40];
   unsigned char f1;
   unsigned char f2;
   unsigned char f3;
   f1 = my_atof(strIn1);
   f2 = my_atof(strIn2);
   printf("A: %s  B: %s \n", strIn1, strIn2);
   printf("A Packed:   ");
   print8bits(f1);
   printf("\n");
   printf("B Packed:   ");
   print8bits(f2);
   printf("\n");

   // multiply some stuff
   printf("A times B\n");
   f3 = multiplyFloats(f1, f2);
   my_ftoa(f3, strOut);
   printf("%s * %s = %s\n", strIn1, strIn2, strOut);

   // add some stuff
   printf("A plus B\n");
   f3 = addFloats(f1, f2);
   my_ftoa(f3, strOut);
   printf("%s + %s = %s\n", strIn1, strIn2, strOut);
   printf("\n");
```

```
/////////example 2/////////
char strIn3[40] = "1.5";
char strIn4[40] = "1.5";
char strOut1[40];

f1 = my_atof(strIn3);
f2 = my_atof(strIn4);
printf("A: %s  B: %s \n", strIn3, strIn4);
printf("A Packed:   ");
print8bits(f1);
printf("\n");
printf("B Packed:   ");
print8bits(f2);
printf("\n");

// multiply some stuff
printf("A times B\n");
f3 = multiplyFloats(f1, f2);
my_ftoa(f3, strOut1);
printf("%s * %s = %s\n", strIn3, strIn4, strOut1);

// add some stuff
printf("A plus B\n");
f3 = addFloats(f1, f2);
my_ftoa(f3, strOut1);
printf("%s + %s = %s\n", strIn3, strIn4, strOut1);
printf("\n");
```

```
    /////////example 3/////////
    char strIn5[40] = "2.625";
    char strIn6[40] = "5.0";
    char strOut2[40];

    f1 = my_atof(strIn5);
    f2 = my_atof(strIn6);
    printf("A: %s  B: %s \n", strIn5, strIn6);
    printf("A Packed:   ");
    print8bits(f1);
    printf("\n");
    printf("B Packed:   ");
    print8bits(f2);
    printf("\n");

    // multiply some stuff
    printf("A times B\n");
    f3 = multiplyFloats(f1, f2);
    my_ftoa(f3, strOut2);
    printf("%s * %s = %s\n", strIn5, strIn6, strOut2);

    // add some stuff
    printf("A plus B\n");
    f3 = addFloats(f1, f2);
    my_ftoa(f3, strOut2);
    printf("%s + %s = %s\n", strIn5, strIn6, strOut2);
    printf("\n");

    //for the excel plot
    char str[40];
    unsigned char f = 0;
    for(int i = 0; i <= 255; i++) {
        my_ftoa(f, str);
        cout << str << endl;
        f++;
    }
    return 0;
}
```

Output:

```
blue@marigold-pride:~/Desktop/School/CISP_440$ ./floats
A: 0.125  B: 4.5
A Packed:   0000 0001
B Packed:   0001 0110
A times B
A expanded: 0000 0000 0000 0000 0000 0001 0000 0000  e1 = -3
B expanded: 0000 0000 0000 0000 0000 0001 0010 0000  e2 = 2
Product:    0000 0000 0000 0001 0010 0000 0000 0000
Norm Prod:  0000 0000 0000 0000 0000 0001 0010 0000  e3 = -1
Packed Prod: 0001 0011
0.125 * 4.5 = 00.56250000
A plus B
A expanded: 0000 0000 0010 0000
B expanded: 0000 0100 1000 0000
Sum:        0000 0100 1010 0000
Packed Sum: 0001 0110
0.125 + 4.5 = 04.50000000

A: 1.5  B: 1.5
A Packed:   0100 0100
B Packed:   0100 0100
A times B
A expanded: 0000 0000 0000 0000 0000 0001 1000 0000  e1 = 0
B expanded: 0000 0000 0000 0000 0000 0001 1000 0000  e2 = 0
Product:    0000 0000 0000 0010 0100 0000 0000 0000
Norm Prod:  0000 0000 0000 0000 0000 0001 0010 0000  e3 = 1
Packed Prod: 0001 0101
1.5 * 1.5 = 02.25000000
A plus B
A expanded: 0000 0001 1000 0000
B expanded: 0000 0001 1000 0000
Sum:        0000 0011 0000 0000
Packed Sum: 0100 0101
1.5 + 1.5 = 03.00000000

A: 2.625  B: 5.0
A Packed:   0010 1101
B Packed:   0010 0110
A times B
A expanded: 0000 0000 0000 0000 0000 0001 0101 0000  e1 = 1
B expanded: 0000 0000 0000 0000 0000 0001 0100 0000  e2 = 2
Product:    0000 0000 0000 0001 1010 0100 0000 0000
Norm Prod:  0000 0000 0000 0000 0000 0001 1010 0100  e3 = 3
Packed Prod: 0101 0111
2.625 * 5.0 = 13.00000000
A plus B
A expanded: 0000 0010 1010 0000
B expanded: 0000 0101 0000 0000
Sum:        0000 0111 1010 0000
Packed Sum: 0111 0110
2.625 + 5.0 = 07.50000000
```

Excel Plot

Description:

Every possible base ten floating point value in our 8 bit format, generated in C and plotted on a number line in Excel.