Notes on Computer Project #6

Comments about the assignment and responses to frequently asked questions will be added to this file as necessary.

***** comments added on 10/09/18 *****

1) Please note the following statement from the assignment handout:

> The deliverables for this assignment are:
>
> proj06.makefile -- the makefile which produces "proj06"
> proj06.support.c -- the source code for your support module
> proj06.driver.c -- the source code for your driver module
>
> Be sure to use the specified file names, and to submit your files for grading via the "handin" program.

It is possible to submit your solution files multiple times: the last files that you submit will be graded.

2) Please note that you are required to create a makefile which controls the translation of your program, and that the name of your executable file must be "proj06".

3) For your convenience, I have supplied the following interface file:

> ~cse320/Projects/project06.support.h

Since that file may not be modified, there is no reason to copy it into your account. Instead, use its absolute pathname in the appropriate preprocessor directive. For example:

```
#include "/user/cse320/Projects/project06.support.h"
```

Any program which uses those functions should contain the appropriate preprocessor directive.

The interface file contains the declaration of function "add", the declaration of "union double_precision", and the definition of two symbolic constants:

```
#define INFINITY  (__builtin_inf())
#define NAN       (__builtin_nan(""))
```

That will allow you to use the symbolic constants INFINITY and NAN in your source code (such as "proj06.driver.c").

4) Consider the following example (an extension of the example on the handout):

```
#include <stdio.h>

union double_precision
{
  double drep;
  unsigned long long int irep;
};
```

```
  int main()
  {
    union double_precision num;
    signed true_exp;

    num.drep = 24.5;

    printf( "Real: %f  Hex integer: %016llx  ", num.drep, num.irep );

    true_exp = ((num.irep >> 52) & 0x7ff) - 0x3ff;

    printf( "True exponent: %d \n\n", true_exp );
  }
```

When compiled and executed, that program produces the following:

```
  Real: 24.500000  Hex integer: 4038800000000000  True exponent: 4
```

Note that shifting and masking (as well as other integer operations) will be quite useful.

5) Please note that it is perfectly acceptable to use data objects of type "union double_precision" in your driver module. Using that approach, you can create test cases which have specific bit patterns by assigning hexadecimal constants to the "irep" version of an object. The "drep" version of that object can then be passed as an argument to one or more of the functions in your support module.

6) Please note the following statement from the assignment handout:

> Function "add" (and any associated "helper" functions which you develop) will constitute the support module. The functions in that support module will not call any C library functions, and they will not use any floating point operations. There is one exception: the functions may use the assignment operation to copy a floating point value from one variable to another.

You may use the floating point assignment operation to copy an object of type "float", but no other floating point operations.

You may wish to use "printf" to display intermediate results as you develop your support module, but be sure to remove those calls (or turn them into comments) before submitting your work.

In most cases, you should print in hexadecimal (using "%016llx") in order to check the bit patterns of the intermediate results. In some cases, it might be useful to print an integer value in decimal (using "%+d") or a floating point value (using "%+14.7e").

Alternatively, you could use "gdb" to display intermediate results.

7) Be sure to think carefully about the special cases that can arise, such as the exponent which represents Infinity (and NaN) and the exponent which represents zero (and denormal values).

As noted on the handout, any arguments which are denormal will be processed as the value zero for this assignment.

Also, any operation where one of the operands is Not-a-Number produces Not-a-Number as the result; otherwise, any floating point operation where one of the operands is Infinity produces Infinity as the result.

8) I have posted two diagrams that might be helpful for Project #6:

  ~cse320/Projects/project06.add1.pdf
  ~cse320/Projects/project06.add2.pdf

Those diagrams give the algorithm and circuits for floating point addition in hardware. Obviously, the steps in software would be similar.

9) I have posted three C programs that demonstrate bitwise operations:

  ~cse320/Projects/project06.unpack.c
  ~cse320/Projects/project06.pack.c
  ~cse320/Projects/project06.union.c

The comments at the beginning of each program show how to compile and execute the program.

--M. McCullen