vfpmath: A Maths library for the VFP architecture

vfpmath is a library of mathematical functions targetting the ARM VFP (Vector Floating Point) architecture. At present it covers most of the functions normally included in the ANSI C math.h library. Although all the routines are implemented in assembly language, it is intended that it should be possible to use them from C, as well as from assembler. Although this could be a first step towards supporting the VFP architecture on RISC OS, other significant problems remain, such as the lack of support for the VFP architecture in the C compiler.

Calling convention

As far as possible vfpmath conforms to the APCS (ARM Procedure Call Standard) so that the routines can be called for any language which follows these conventions. As far as the VFP architecture is concerned, this means that the double precision registers d0 to d7 may be used as scratch registers. Conversely, registers d8 to d15 are reserved for register variables must be preserved across procedure calls.

There are a number of conventions for passing floating point variables as arguments to a procedure. The Hard VFP convention is that arguments are passed in the double precision registers, with the first floating point argument being in d0. A floating point result is also returned in d0. The Soft VFP convention assumes that floating point instructions are emulated in software, and so arguments are passed in the ARM core integer registers instead. The first argument is passed with the low order bits in r0, and the high order bits in r1. Results are similarly returned in r0 and r1.

Previous floating point systems on RISC OS, whether the FP Emulator, or in some cases the FPA hardware, support a different instruction set, use different registers, and stores the two words making up a double precision value in a different order. Supporting this alongside the two VFP conventions is less easy because the assembler will not allow VFP and FPE instructions to be mixed in a single source file. However, for compatibility with the Norcorft compiler, vfpmath now offers this option, and the necessary FPE instructions are assembled manually. When this convention is selected, floating point arguments are passed in a pair of consecutive integer registers, with the high order word first (the opposite way round to Soft VFP conventions). Floating point results are returned in the FPE register f0.

Building vfpmath

Building vfpmath will require the RISC OS Open development tools. Assuming that these have been installed on your system, then vfpmath can be built by changing to the directory where the source files are located, then from the command line run `amu`. This will build the library and a simple demonstration/test program.

By default the library will use the default RISC OS/FPE convention (see above). If you with to use a different calling convention, then, in the makefile, edit the macro asmopts. For HardVFP calling conventions remove the flag:

```
--predefine="FPE_POINTERS SETL {TRUE}"
```

For SoftVFP conventions, additionally change the `--fpu` option to:

```
--fpe=SoftVFP+VFPv3
```

No changes are necessary to the header file vfpmath.h.

Once the library has been built, you may decide you wish to install it alongside the other C libraries. To do this, you can use the command `amu install`. By default this will install the vfpmath in the same directory as the Shared C library, but this location can be changed by editing the `install` target of the makefile.

Compatibility

The vfpmath library is now developed on a Raspberri Pi 4 running RISC OS 5.29. This uses a Cortex-A72 processor, and supports the VFPv4 architecture. The library does use a few instructions introduced in the VFPv3 standard, most notably a VMOV.F64 instruction with an immediate constant to generate a limited range of common constants (like 0.5 or 1.0) in a VFP register. There is one routine (used internally for range reduction of the argument to trigonometric functions) which uses a VFPv4 instruction VRINT. If these were changed then it should be possible to use vfpmath on earlier versions of the VFP architecture with at least 16 double precision registers. In some cases it is necessary to work with the integer representation of a flating point value, where the ARMv6 instructions UBFX (Unsigned Bit Field eXtract), BFI (Bit Field Insert) are useful. This shouldn't cause any problems as all RISC OS hardware supporting the VFP instruction set should support ARMv6 or later.

Library functions

The vfpmath library includes most of the functions normally included in the ANSI C standard, and a growing number of additional functions from the C99 standard. To avoid any conflicts over function names, all names in the vfpmath library have been prefixed with vfp_, and the corresponding header file is similarly called vfpmath.h. The table below summarises the available functions:

vfp_sin	sine of x	$\sin x$
vfp_cos	cosine of x	$\cos x$
vfp_tan	tangent of x	$\tan x$
vfp_arcsin	inverse sine of x	$\sin^{-1} x$
vfp_arccos	inverse cosine of x	$\cos^{-1} x$
vfp_arctan	inverse tangent of x	$\tan^{-1} x$
vfp_atan2	angle of point x,y from positive x axis	$\tan^{-1} y/x$
vfp_exp	e to the power of x	e^x
vfp_exp2	2 to the power of x	2^x
vfp_pow	x to the power of y	x^y
vfp_sqrt	square root of x	$\sqrt{x}$
vfp_cbrt	cube root of x	$\sqrt[3]{x}$
vfp_hypot	hypotinuse with sides x and y	$\sqrt{x^2+y^2}$
vfp_ln	base e, natural logarithm of x	$\ln x$
vfp_log	base 10 logarithm of x	$\log x$
vfp_log2	base 2 logarithm of x	$\log_2 x$
vfp_sinh	hyperbolic sine of x	$\sinh x$
vfp_cosh	hyperbolic cosine of x	$\cosh x$
vfp_tanh	hyperbolic tangent of x	$\tanh x$
vfp_arcsinh	inverse hyperbolic sine of x	$\sinh^{-1} x$
vfp_arccosh	inverse hyperbolic cosine of x	$\cosh^{-1} x$
vfp_arctanh	inverse hyperbolic tangent of x	$\tanh^{-1} x$
vfp_floor	largest integer less than or equal to x	$\lfloor x \rfloor$

vfp_ceil	smallest integer greater than or equal to x	$\lceil x \rceil$
vfp_rint	round x to the nearest integer	int(x)
vfp_frexp	return the fraction and exponent given x	$m \times 2^p = x$
vfp_ldexp	return x given its fraction and exponent	$x = m \times 2^p$
vfp_gamma	gamma function of x	$\Gamma(x)$
vfp_gammln	logarithm of gamma function	$\ln \Gamma(x)$
vfp_erf	error function	erf(x)
vfp_erfc	complementary error function	erfc(x)
vfp_igamp	incomplete gamma function of the first kind	P(a, x)
vfp_igamq	incomplete gamma function of the second kind	Q(a, x)
bessj0, bessj1	Bessel function of the first kind of orders 0 and 1	$J_0(x)$ and $J_1(x)$
bessy0, bessy1	Bessel function of the second kind of orders 0 and 1	$Y_0(x)$ and $Y_1(x)$

By default, the basic prototype for the functions in the C programming language is:

```
extern double fn_name(double x);
```

A few functions are exceptions to this rule and take two floating point arguments, or have an additional integer argument. The C prototypes for these are:

```
extern double vfp_pow(double x, double y);
extern double vfp_atan2(double x, double y);
extern double vfp_hypot(double x, double y);
extern double vfp_frexp(double x, int *p);
extern double vfp_ldexp(double m, int p);
```

Using vfpmath from C

The vfpmath library is intended to be called from C, and the default calling convention allows it to be used with code produced by the Norcroft compiler with the default options.

To begin with, the prototypes for the vfpmath functions are all found in the vfpmath.h header file which will need to be included at the begining of your program:

```
#include "vfpmath.h"
```

Also, before any VFP instructions can be used you will need a current VFP context. The Shared C library does not create this for you, so you will need to use the SWIs provided by the VFPSupport module. The demonstration program included with vfpmath defines two functions to simplify this, or you can use VFPSupport_CreateContext (SWI &58EC1) to do this.

Once a VFP context has been created, we can use the vfpmath routines like the usual mathematical functions in the C library. For example:

```
double x=1.0, y;
y=vfp_exp(x); /* returns e^x in y */
```

If you have built the library with the correct conventions for your system, then it will be possible to use it anywhere which expects a floating point value. This includes other procedures, futher calculations and input/output:

```
printf("The value of e is %12.10f\n", y);
```

To facilitate testing you may wish to produce a table of function values for regularly spaced values of the argument. The utility function `fntable` will do this. Create an array with the values of the argument, then call fntable with a pointer to the array, the number of values and a pointer to the

function to tabulate:

```
double x[10];
for(i=1; i<=10; i++)
    x[i]=(double) i; /* x=1...10 */
fntable(x, 10, &vfp_log); /* table of values of log */
```

There is another function `linrange` to create a linearly spaced range of values between a given start and end point. Combining these functions makes it easy to compute a table of function values:

```
double x[7];
const pi=3.142;
linrange(x, 7, 0.0, pi);
fntable(x, 7, &vfp_sin);
```

Finally, remember to release any allocated memory using `free()`, restore the previous VFP context and destroy your previously created context using VFPSupport_DestroyContext (SWI &58EC2).

Once you have written your program, you will need to link it with the vfp_math library. In general, assuming the same filenames as used by the distributed Zip file, it should be possible to compile and link your program using the following command:

```
*cc -c -IC: c.VFPTest -o o.VFPTest
*link -AIF o.VFPTest o.harness o.vfpmath C:o.stubs -Output VFPTest
```

Remember to change the name of the C source file and the associated object file to that of your own code. If you do not use any of the utility functions supplied as part of the test 'harness', then you can omit the `o.harness` object at the link stage. Note that when linking you may see warning messages about attribute differences. These are generated because the C compiler currently generates code for the FPE system, and the linker detects that the calling convention does not match that used by the vfpmath library. Assuming that you have used the correct conventions for your system, and built the library using the correct options then these warnings can be ignored.

Currently it is not straightforward to use the options for HardVFP or SoftVFP calling conventions from C. The Norcroft compiler currently only supports the FPE system, passes arguments in two integer registers, and expects results to be returned in the FPE register f0. One solution to this is to supply 'dummy' routines which simply copy the arguments into the correct registers, call the appropriate vfpmath routine, and then put the result into the FPE register f0. This could be done using the following assember sequence (the FPE instruction has to be assembled manually because the assembler will not accept both VFP and FPE instructions in a single source file):

```
__sin       mov ip,sp
            stmdb sp!,{fp,ip,lr,pc}
            sub fp,ip,#4
            vmov.F64 d0,r1,r0    ; put first argument in d0
            bl vfp_sin           ; call library routine
            vmov.F64 r1,r0,d0
            stmdb r13!,{r0,r1}   ; store result on stack and load into f0
            DCD 0xECBD8102       ; ldfd f0,[r13],#8
            ldmdb fp,{fp,sp,pc}
```

Using vfpmath with gcc

It is now possible to use vfpmath with the gcc compiler, although there are some limitations. In particular, gcc will always use the SoftVFP conventions when targetting the VFP architecture, even when using Unixlib. The assembler source files will have to be converted to the syntax expected by the `gas` assembler. The supplied perl script `convasm/pl` will do this. Although this is not a

completely general conversion script, it now manages to convert the vfpmath sources without the need for manual editing. There is an additional target in the Makefile to convert and assemble one of the sources. You can examine this and change the commands to suit your requirements.

When writing C code, make sure that you are using the correct version of vfpmath.h, namely the one using arguments declared as double, not the default using pointers. Then, compile your program and link it with one of the library object files using:

```
gcc -O2 -mfpu=vfp -o test c.test o.trig
```

Accuracy

Although coverage is far from exhaustive, the range of available tests covers most of the core mathematical functions. From these it is possible to give an estimate of the accuracy of these functions. Preliminary results are as follows:

Function	Test	Reported absolute error
cos x	Principal range, known values	9.99E-16
	Range reduction	1.33E-15
sin x	Principal range, known values	1.11E-16
	Range reduction	5.66E-15
arccos x	Known values	2.22E-16
arcsin x	Known values	4.44E-16
exp x	Principal range	3.33E-16
	Range reduction	8.27E-15
	Reflection formula	1.77E-15

Future developments

Possible future development of the vfpmath library includes:

- Standardise on one calling convention, or:
- Extend the utility functions for testing to support different calling conventions.
- Additionally convert the assembly language source files to the format required by the `gas` assember and provide an ELF format version of the library for use with GCC.
- Develop a thorough test suite to check the accuracy of all the maths routines.
- Add error checking to the vfpmath routines.
- Improve coverage of the library, particularly by adding the extra functions in the ISO C99 standard.
- Supply routines to format VFP variables for display, similar to those required by `printf`.
- Improve comments in the source code and document the mathemtical techniques used in calculating the functions.