The following describes the current state of the NetWinder's floating point emulator.

In the following nomenclature is used to describe the floating point instructions. It follows the conventions in the ARM manual.

```
<S|D|E> = <single|double|extended>, no default
{P|M|Z} = {round to +infinity,round to -infinity,round to zero},
          default = round to nearest
```

Note: items enclosed in {} are optional.

Floating Point Coprocessor Data Transfer Instructions (CPDT)

LDF/STF - load and store floating

```
<LDF|STF>{cond}<S|D|E> Fd, Rn
<LDF|STF>{cond}<S|D|E> Fd, [Rn, #<expression>]{!}
<LDF|STF>{cond}<S|D|E> Fd, [Rn], #<expression>
```

These instructions are fully implemented.

LFM/SFM - load and store multiple floating

Form 1 syntax:

```
<LFM|SFM>{cond}<S|D|E> Fd, <count>, [Rn]
<LFM|SFM>{cond}<S|D|E> Fd, <count>, [Rn, #<expression>]{!}
<LFM|SFM>{cond}<S|D|E> Fd, <count>, [Rn], #<expression>
```

Form 2 syntax:

```
<LFM|SFM>{cond}<FD,EA> Fd, <count>, [Rn]{!}
```

These instructions are fully implemented. They store/load three words for each floating point register into the memory location given in the instruction. The format in memory is unlikely to be compatible with other implementations, in particular the actual hardware. Specific mention of this is made in the ARM manuals.

Floating Point Coprocessor Register Transfer Instructions (CPRT)

Conversions, read/write status/control register instructions

```
FLT{cond}<S,D,E>{P,M,Z} Fn, Rd          Convert integer to floating point
FIX{cond}{P,M,Z} Rd, Fn                 Convert floating point to integer
WFS{cond} Rd                            Write floating point status register
RFS{cond} Rd                            Read floating point status register
WFC{cond} Rd                            Write floating point control register
RFC{cond} Rd                            Read floating point control register
```

FLT/FIX are fully implemented.

RFS/WFS are fully implemented.

RFC/WFC are fully implemented. RFC/WFC are supervisor only instructions, and presently check the CPU mode, and do an invalid instruction trap if not called from supervisor mode.

Compare instructions

```
CMF{cond} Fn, Fm        Compare floating
CMFE{cond} Fn, Fm       Compare floating with exception
CNF{cond} Fn, Fm        Compare negated floating
CNFE{cond} Fn, Fm       Compare negated floating with exception
```

These are fully implemented.

Floating Point Coprocessor Data Instructions (CPDT)

Dyadic operations:

```
ADF{cond}<S|D|E>{P,M,Z} Fd, Fn, <Fm,#value> - add
SUF{cond}<S|D|E>{P,M,Z} Fd, Fn, <Fm,#value> - subtract
RSF{cond}<S|D|E>{P,M,Z} Fd, Fn, <Fm,#value> - reverse subtract
MUF{cond}<S|D|E>{P,M,Z} Fd, Fn, <Fm,#value> - multiply
DVF{cond}<S|D|E>{P,M,Z} Fd, Fn, <Fm,#value> - divide
RDV{cond}<S|D|E>{P,M,Z} Fd, Fn, <Fm,#value> - reverse divide
```

These are fully implemented.

```
FML{cond}<S|D|E>{P,M,Z} Fd, Fn, <Fm,#value> - fast multiply
FDV{cond}<S|D|E>{P,M,Z} Fd, Fn, <Fm,#value> - fast divide
FRD{cond}<S|D|E>{P,M,Z} Fd, Fn, <Fm,#value> - fast reverse divide
```

These are fully implemented as well. They use the same algorithm as the non-fast versions. Hence, in this implementation their performance is equivalent to the MUF/DVF/RDV instructions. This is acceptable according to the ARM manual. The manual notes these are defined only for single operands, on the actual FPA11 hardware they do not work for double or extended precision operands. The emulator currently does not check the requested permissions conditions, and performs the requested operation.

```
RMF{cond}<S|D|E>{P,M,Z} Fd, Fn, <Fm,#value> - IEEE remainder
```

This is fully implemented.

Monadic operations:

```
MVF{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - move
MNF{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - move negated
```

These are fully implemented.

```
ABS{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - absolute value
SQT{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - square root
RND{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - round
```

These are fully implemented.

```
URD{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - unnormalized round
```

```
NRM{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - normalize
```

These are implemented. URD is implemented using the same code as the RND instruction. Since URD cannot return a unnormalized number, NRM becomes a NOP.

Library calls:

```
POW{cond}<S|D|E>{P,M,Z} Fd, Fn, <Fm,#value> - power
RPW{cond}<S|D|E>{P,M,Z} Fd, Fn, <Fm,#value> - reverse power
POL{cond}<S|D|E>{P,M,Z} Fd, Fn, <Fm,#value> - polar angle (arctan2)

LOG{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - logarithm to base 10
LGN{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - logarithm to base e
EXP{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - exponent
SIN{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - sine
COS{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - cosine
TAN{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - tangent
ASN{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - arcsine
ACS{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - arccosine
ATN{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - arctangent
```

These are not implemented. They are not currently issued by the compiler, and are handled by routines in libc. These are not implemented by the FPA11 hardware, but are handled by the floating point support code. They should be implemented in future versions.

Signalling:

Signals are implemented. However current ELF kernels produced by Rebel.com have a bug in them that prevents the module from generating a SIGFPE. This is caused by a failure to alias fp_current to the kernel variable current_set[0] correctly.

The kernel provided with this distribution (vmlinux-nwfpe-0.93) contains a fix for this problem and also incorporates the current version of the emulator directly. It is possible to run with no floating point module loaded with this kernel. It is provided as a demonstration of the technology and for those who want to do floating point work that depends on signals. It is not strictly necessary to use the module.

A module (either the one provided by Russell King, or the one in this distribution) can be loaded to replace the functionality of the emulator built into the kernel.