TODO LIST

```
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.

There are a couple of ways to approach the implementation of these. One method would be to use accurate table methods for these routines. I have a couple of papers by S. Gal from IBM's research labs in Haifa, Israel that seem to promise extreme accuracy (in the order of 99.8%) and reasonable speed. These methods are used in GLIBC for some of the transcendental functions.

Another approach, which I know little about is CORDIC. This stands for Coordinate Rotation Digital Computer, and is a method of computing transcendental functions using mostly shifts and adds and a few multiplications and divisions. The ARM excels at shifts and adds, so such a method could be promising, but requires more research to determine if it is feasible.

Rounding Methods

The IEEE standard defines 4 rounding modes. Round to nearest is the default, but rounding to + or - infinity or round to zero are also allowed. Many architectures allow the rounding mode to be specified by modifying bits in a control register. Not so with the ARM FPA11 architecture. To change the rounding mode one must specify it with each instruction.

This has made porting some benchmarks difficult. It is possible to introduce such a capability into the emulator. The FPCR contains bits describing the rounding mode. The emulator could be altered to examine a flag, which if set forced it to ignore the rounding mode in the instruction, and use the mode specified in the bits in the FPCR.

This would require a method of getting/setting the flag, and the bits in the FPCR. This requires a kernel call in ArmLinux, as WFC/RFC are supervisor only instructions. If anyone has any ideas or comments I would like to hear them.

NOTE:

> pulled out from some docs on ARM floating point, specifically for the Acorn FPE, but not limited to it:
>
> The floating point control register (FPCR) may only be present in some implementations: it is there to control the hardware in an implementation- specific manner, for example to disable the floating point system. The user mode of the ARM is not permitted to use this register (since the right is reserved to alter it between implementations) and the WFC and RFC instructions will trap if tried in user mode.
>
> Hence, the answer is yes, you could do this, but then you will run a high risk of becoming isolated if and when hardware FP emulation comes out
>
> -- Russell.