FPU_I386

The FPU_I386 module is a shared library providing the implementation of the floating-point support routines listed in Appendix B4 of Kernighan & Ritchie. It also contans the support routines for floating-point input and output from section B5.

Caveat

The current implementation of the ROME core does not save and restore the floating-point state between processes. The main reason for this is that few processes use floating-point and supporting multi-process FPU use would add significantly to the overhead of a context switch. The usual workaround for this limitation is to ensure that only one process in the system uses floating-point operations. So far, this has not proved to be an issue for embedded applications.

Extending the core to support multiple floating-point contexts is not particularly difficult, but so far it has not been necessary.

Shared Library Macros and Routines

Full details of the behaviour of most of these routines is exactly as described in Appendix B of K&R. A number of these routines are implemented directly from the math library support provided in the *gcc* distribution as inlined assembler functions calling the appropriate machine instruction. However, some functions are implemented as separate routines. All functions use the default **double** type as defined in Appendix B. The compiler supports the full-precision **long double** type and a corresponding floating-point library could be provided, if needed.

Inlined assembler functions

The following functions are directly coded as inline assembler in the file *math.h*:
atan2, **ceil**, **cos**, **exp**, **fabs**, **floor**, **fmod**, **ldexp**, **log**, **log10**, **sin**, **sqrt**, **tan**

Standard Functions

The following functions have no direct assembler implementation, but are written as routines using the standard formulae:
asin, **acos**, **atan** using calls to *atan2* and *sqrt*
cosh, **sinh**, **tanh** using *exp*
modf using *ceil* or *floor*

atof

double *atof*(
const char **_s*)

The *atof* macro expands directly to *strtod*(*_s*, (**char** **)*NULL*).

fpu_pfloat

int *fpu_pfloat*(
char **buffer*,
int **rc*,
int *flags*,
double *x*,
char *format*,
int *width*,
int *precision*,
int *sign*)

The *fpu_pfloat* routine formats the number *x* into the output string *buffer* according to the supplied formatting controls. The routine is intended to be called from within the main C runtime library *printf* set of routines to process the 'e', 'f' and 'g' format effectors. The routine returns the number of characters written to the buffer, and it also increments the *rc* variable by that amount. The *format* character is one of 'e', 'E', 'f', 'g' or 'G' and controls the layout according to the conversion rules in table B-1 of K&R. The *width* and *precision* fields specify the total field width and number of digits following the decimal point. The *sign* flag forces a mandatory sign character. The *flags* are a bitfield of *printf* options for leading zeroes and plus or space as sign.

fpu_pi

double *fpu_pi*(**void**)

The *fpu_pi* routine returns the value of the constant π.

fpu_powint

double *fpu_powint*(
double *x*,
int *pow*)

The *fpu_powint* routine computes the value of x^{pow} for a (positive or negative) integer *pow* using the minimum number of multiplications.

frexp

double *frexp*(
double *x*,
int **exp*)

The *frexp* routine splits *x* into a fractional part and a power of 2. The routine operates directly on the IEEE bit representation of the number to yield maximum accuracy.

pow

double *pow*(
double *x*,
double *y*)

The *pow* routine computes x^y. In the case where *y* is an exact integer, the *fpu_powint* routine is used for maximum accuracy.

strtod

double *strtod*(
const char **s*,
char ***endp*)

The *strtod* routine converts the characters in *s* into a floating point number, ignoring any leading white spaces, and storing the residue in *endp* (if *endp* is not *NULL*). The accepted syntax is:

$$[sign]digits[.[digits]][\{e|E\}[sign]digits]$$

where $digits$ are zero or more decimal digits.