NetWinder Floating Point Notes

Scott Bambrough, scottb@netwinder.org $Revision: 1.1.1.1 $, $Date: 1999/11/02 15:25:12 $

This document describes the floating point subsystems available for ARM systems and the floating point subsystems in use on ARM Linux for the NetWinder.

1 The NetWinder Floating Point Hardware

There is none. The StrongARM unlike the Intel x86 series of CPU's has no integrated floating point hardware. ARM manufactures a coprocessor floating point unit; the FPA11, however it is only available on the ARM 7500FE and it is not compatible with the Intel StrongARM chips. Hence the Netwinder has no floating point hardware at all. It depends entirely on software that emulates the FPA11 floating point unit.

2 ARM Floating Point Software

ARM distributes a software development toolkit, that contains a set of floating point libraries that can be used to construct an emulator. These libraries I believe are licensed from Acorn by ARM for inclusion in their toolkit. It is however expensive, and source is not available without a source licence from ARM. Consequently open source development projects like Linux and NetBSD contain no source for a floating point emulator.

So how do various systems deal with the lack of floating point hardware? Acorn developed their own emulator from scratch for their RISC operating systems. The ARM port of NetBSD has a unique solution; the binary object code for the ARM floating pointing routines was converted into a text file of hexidecimal numbers. Some assembler glue is added to the file and the assembler is then used to convert the file back to binary form. Hence the NetBSD kernel, contains source of a sort, although it is unreadable and not maintainable. Russell King distributes a port of the Acorn floating point emulator that is compatible with ARM Linux kernels.

3 Handling Floating Point Without Hardware

What exactly is a floating point emulator? Well as I said before the StrongARM has no floating point hardware. Yet it is possible to write, compile and execute C code that uses the float, double and long double data types. How is this possible? There are a couple of possible ways of handling it. The compiler can handle floating point operations by generating calls to routines to handle the operations, or it can generate code for hardware that will handle the floating point operations.

Both methods are possible on the NetWinder. By default the compiler generates FPA11 assembler instructions to handle floating point operations. If code is compiled with the compiler flag -msoft-float then the compiler generates calls to software routines that will handle the floating point operations. Both methods cannot succeed on their own without help. A library implementing the routines the compiler calls when generating code with -msoft-float must be provided. Thanks to Phil Blundell and Neil Carson such a library exists. Something to handle the FPA11 opcodes is required for the default compilation case. Otherwise when the CPU executes an FPA11 opcode, it will fault with an undefined instruction trap.

This is where the floating point emulator comes in. It is software designed to mimic the FPA11 hardware. When the StrongARM CPU executes an FPA11 opcode, it faults with an undefined instruction trap. This is caught and handled by the ARM Linux kernel. The kernel saves the machine state and calls the floating point emulator's entry point with the offending opcode. The emulator decodes the opcode and performs the necessary floating point operations, then returns to the kernel. The kernel restores the machine state, and returns execution to the next executable instruction in the process that attempted the floating point operation. If the opcode is not a floating point opcode, control is returned to the kernel and the invalid instruction trap handler is executed, resulting in the process dumping core.

I should point out it this technique is not restricted to emulation of floating point operations. It is possible to emulate the Thumb instructions in a similar manner. It should also be possible to emulate another FPU, not just the FPA11. The choice of emulating the FPA11 has its merits, however, theoretically allowing ARM binaries to run on any ARM CPU with either the emulator present or the actual FPA11 hardware. One could even design a model FPU that has no hardware implemenation. It would be an interesting exercise to extend the FPA11 emulator to allow the rounding methods to be selected on the fly, rather than being encoded in the instruction. As it stands now, the compiler only ever generates code that uses the IEEE round to nearest rounding method.

4 Floating Point Emulators for ARM Linux

There are actually two floating point emulators available for ARM Linux. A version of the Acorn emulator was ported to ARM Linux by Russell King. He has a very specific licence which allows him to distribute this emulator as a module with most of the symbols stripped out from his web site. It is well established, in its third release. It is highly optimized, and written in ARM assembly language.

The licence restrictions on the Acorn FPE proved problematic for the NetWinder. It looked like it would be expensive to licence the emulator from Acorn. The ARM SDT route was also expensive. Neither route would satisfy one of our goals of a completely open source operating system for the NetWinder. Writing an emulator from scratch, was not really an option given our ship dates. After all, writing routines to implement the IEEE floating point algorithms in software is not what I would call a trivial short term programming exercise. The implementation as a kernel module also caused support headaches. If the module failed to load any program linked with the C libaries failed to run. Thus the shell would crash and the system would not run. (The __set_fpucw routine called from the C library startup code issues a WFS and RFS instruction).

It is possible to logically divide the construction of the emulator into two logically distinct parts; a library of IEEE floating point algorithms, and code to emulate the FPA11 floating point hardware. As luck would have it Phil Blundell had brought to my attention a free library of code implementing IEEE floating point algorithms. SoftFloat is a complete IEEE 754 floating point library written by John Hauser. Neil Carson had ported this software to run on the ARM platform, and Phil had modified it to create a library for use with the compiler when the -msoft-float compiler flag was used. With SoftFloat in hand, it was possible to concentrate on building just the emulation code for the floating point hardware. Thus the NWFPE was born.

5 The SoftFloat Library

SoftFloat is a software implementation of floating point that conforms to the IEC/IEEE standard for binary floating point arithmetic. As many as four formats are supported: single precision, double precision, extended double precision, and quadruple precision. All operations required by the standard are implemented, except for conversions to and from decimal. SoftFloat implements the following arithmetic operations:

```
1) Conversions among all the floating point formats, and also between 32-bit
integers and any of the floating point formats.
2) The add, subtract, multiply, divide, and square root operations for all
floating point formats.
3) The floating point remainder operation defined by the IEC/IEEE standard
for all floating point formats.
4) For each floating-point format, a ''round to integer'' operation that
rounds to the nearest integer value in the same format.
5) Comparisons between two values in the same floating-point format.
```

All four rounding modes prescribed by the IEC/IEEE standard are implemented for all operations that require rounding.

All five exception flags (INEXACT, UNDERFLOW, OVERFLOW, DIVBYZERO, INVALID) required by the IEC/IEEE standard are implemented. In the terminology of the IEC/IEEE standard, SoftFloat can detect tininess for underflow either before or after rounding. Detecting tininess after rounding is better because it results in fewer spurious underflow signals. Like most systems, SoftFloat always detects loss of accuracy for underflow as an inexact result.

At the time of this writing, the most up-to-date information about SoftFloat and the latest release can be found at the Web page `http://HTTP.CS.Berkeley.EDU/~jhauser/arithmetic/softfloat.html` .

6 libFloat and the compiler option -msoft-float

The compiler usually generates FPA11 floating point instructions for floating point math operations with float, double and long double types. It assumes an FPA11 floating point unit or an emulator is available.

It is possible to force the compiler to assume no floating point hardware or an emulator exists. To do this compile with the -msoft-float compiler flag. The compiler then generates function calls instead of floating point instructions. No changes are needed to program code, just recompilation. This means a library is needed to satisfy the references to the function calls generated by the compiler. This library is not provided by the compiler, it only defines the interface; each port is expected to provide its own implementation. On the NetWinder, this library is libfloat. This was ported to ARM Linux by Phil Blundell, based on work Neil Carson did for a similar library for NetBSD. libfloat depends on SoftFloat for the implmentation of the IEEE floating point algorithms.

The latest version of libfloat can be found at:

`ftp://ftp.netwinder.org/users/s/scottb/libfloat/libfloat-990616.tar.gz` .

Code compiled with -msoft-float should not linked with code that wasn't. The compiler follows APCS (ARM Procedure Call Standard) rules when generating floating point code. This uses the registers r0-r3 for the first four arguements (integer and floating point), and the stack for the rest. A floating point value can end up entirely in a register or multiple registers. It can even be split between the registers and the stack. A floating point return value is placed in f0. If the -msoft-float flag is used, then the compiler uses r0-r2, to return floating point values. Which registers are used depends on the size of the floating point numbers. Mixing object modules built with -msoft-float with those built is not really possible due to this difference in calling conventions.

7 Miscellaneous

7.1 Author

The author and maintainer of the NetWinder Floating Point Notes is Scott Bambrough (scottb@netwinder.org). Please send any comments, additions, or corrections so they can be included in the next release. The latest version of this document may be obtained from `http://www.netwinder.org/~scottb/notes/FP-Notes.html` .

7.2 History

The original version of these notes was in the form of an html page.

October 21, 1999 (version 1.0): Converted html page to SGML, and cleaned up the content.

7.3 Copyright Notice