LA-UR-77-2649

MASTER

TITLE: "A FLOATING POINT HARDWARE EMULATOR FOR RSX-11D"

CONF-771144--2

AUTHOR(S): Martin Kellogg, MP-1, LASL

Marshall Long, Yale University, New Haven, CT

SUBMITTED TO: PROCEEDINGS OF THE DIGITAL EQUIPMENT COMPUTER USERS SOCIETY (DECUS), SAN DIEGO, CA, NOVEMBER 28 - DECEMBER 1, 1977

los alamos
scientific laboratory
of the University of California
LOS ALAMOS, NEW MEXICO 87544

Form No. 836
St. No. 2629
1/75

UNITED STATES
ENERGY RESEARCH AND
DEVELOPMENT ADMINISTRATION
CONTRACT W-7405-ENG. 36

A FLOATING POINT HARDWARE EMULATOR FOR RSX-11D

Marshall Long
Engineering and Applied Science Department
Yale University
New Haven, CT

Martin Kellogg
Los Alamos Scientific Laboratory
Los Alamos, NM

ABSTRACT

An RSX-11D task has been written to simulate the FP-11 floating point hardware on systems that lack this hardware. The simulation is transparent to tasks using floating point instructions. All normal features of the hardware are simulated exactly, including its action on exception conditions.

The emulator is a privileged task occupying about 2.7K words of memory. When it is loaded and run, it sets up a linkage to intercept the reserved instruction trap before it reaches the executive, and route it to a service routine that can decode and simulate the floating point instruction set.

The results of a benchmark timing test are given, as are notes on converting the emulator to run under RSX-11M.

MOTIVATION

At the Clinton P. Anderson Meson Physics Facility (LAMPF) we maintain a large data-acquisition system that runs under the RSX-11D operating system on the PDP-11. This system is supported on over a dozen different machines around the site and must be usable in the hardware environments of all these machines.

The system was originally developed using the DEC FORTRAN IV Plus compiler and object time system. At that time all the machines it was to run on had FP-11 floating-point hardware, and FORTRAN IV Plus was able to use this hardware far more effectively than FORTRAN IV could. Later, a requirement arose to run the system on machines such as PDP-11/40s that lacked the FP-11 hardware. We first attempted to develop a version based on FORTRAN IV, but we found that language incompatibilities between the two systems caused difficulties. Although the FORTRAN IV version was actually used for a time, its quality was never as high as that of the FORTRAN IV Plus version. Furthermore, maintaining a large and complex application is a time-consuming task, and trying to maintain two compatible versions was costing more time than we felt we could afford.*

For these reasons we decided to put some effort into developing an emulator for the FP-11 hardware. We felt that the time spent in doing this would be quickly paid back by the time saved by not supporting a FORTRAN IV version of our application. The use of the emulator was to be transparent to tasks, so that a task that ran with floating-point hardware would also run with the emulator and require no modification or rebuilding.

HARDWARE REQUIREMENTS

The emulator is designed to run in a hardware environment that supports RSX-11D. Thus it uses the extended instruction set (hardware multiply/divide) on the PDP-11. It does not use the PDP-11/40 floating instruction set (FIS). It occupies about 2.7K words of memory.

GATHERING OF STATISTICS

The emulator provides a feature that is not present in the FP-11 hardware: a version can be assembled that will gather statistics on the various floating point instructions executed and addressing modes used. This can provide valuable information about the instruction mix in a floating point calculation. Detailed instructions for using the statistics feature are given in the emulator source code.

INTERFACE TO THE RSX-11D SYSTEM

The emulator is a privileged task that runs

*There were actually four versions, since at the time we were also trying to maintain compatible versions under RSX-11M.

under RSX-11D. The general strategy for interfacing it to the system is to intercept the reserved instruction trap before it reaches the RSX-11D executive. If the instruction causing the trap proves to be a floating point instruction issued by a task, the emulator simulates the action of the FP-11 hardware and returns control to the task. Otherwise, it forwards the trap to the executive to allow normal system action.*

If an error condition occurs during the simulated execution of the floating point instruction, the action taken depends on the nature of the error. If it is an addressing error (odd address or segment fault), the emulator simulates an odd address trap from the task. If it is a floating point exception (overflow, underflow, divide by zero, etc.), the emulator simulates a floating point exception trap from the task. In either case, the executive will take whatever action the task has requested -- cause an odd address AST, a floating point exception AST, or terminate the task.

The most straightforward way to emulate the FP-11 hardware would be to assign the registers to fixed locations in memory. If this were done, the registers would have to be swapped every time the system context was switched between running tasks. This would be done by emulating the load and store instructions issued by the executive to switch context, and would be a time-consuming operation that would greatly degrade the system's performance.

To avoid this large overhead in the system, the emulator uses the floating point context save area in the currently active task for its registers. Thus each task has an up-to-date set of floating point registers at all times, and it is not necessary to swap the registers during a context switch. The executive must be modified to prevent it from attempting to swap the registers, but the changes are very simple and can be handled by patching the executive in memory.

In order to intercept the reserved instruction trap, the emulator "steals" the trap from the system when it is loaded. The technique for doing this was developed by Sally Shlaer, and is described in the RSX-11 Special Interest Group Newsletter.[1]

Fig. 1 shows the overall logical flow of the emulator.

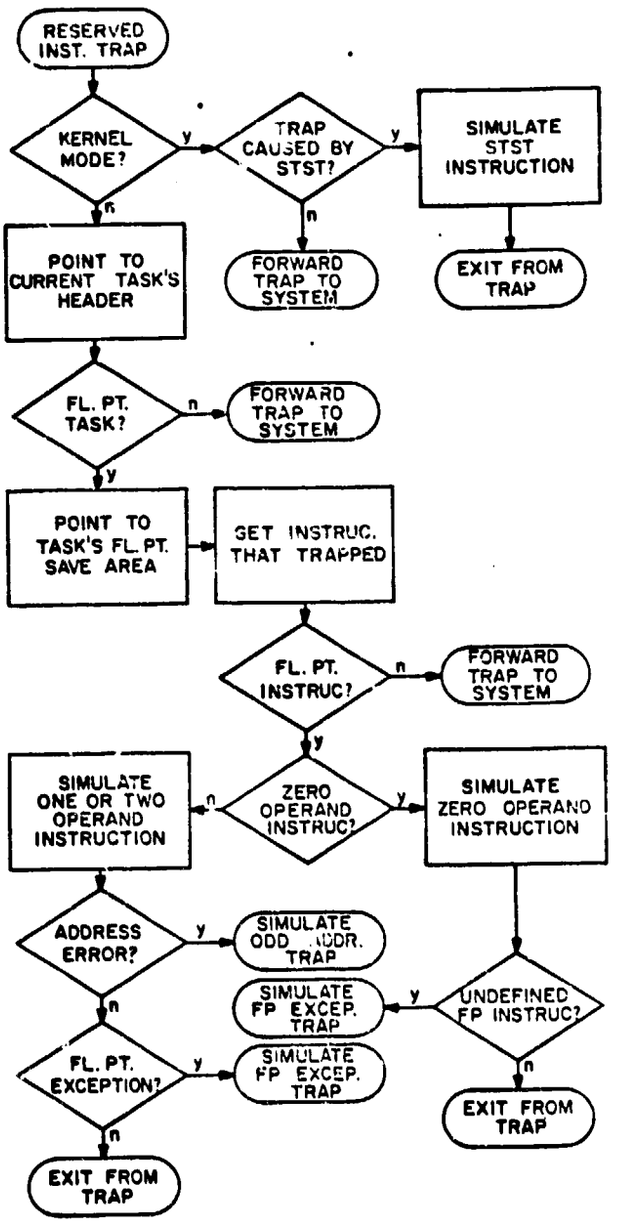

Figure 1. Logical flow of the emulator

INSTALLATION

To use the emulator an RSX-11D system must be generated that assumes there is floating point hardware on the machine. This is done by including the keyword 'FP' in the PDP-11 line of the SYSGEN phase 1 control file, for example:

```
PDP11 = 45,96K,FP
```

Once the emulator task has been assembled and built, installation in the system consists merely of installing and running the task. The emulator will steal the reserved instruction trap from the system, suspend execution, and subsequently process all res-

*An exception is the STST instruction, which the emulator allows to be issued by the executive, since the executive uses it in its floating point exception service.

erved instruction traps caused by floating point instructions.

At the same time, the executive should be patched to disable the context switching of floating point registers. This is done by using the OPEN command to change two locations, as follows:

RSX-11D version 6A:

```
MCR>OPE 7232
007232/001010 240<ESC>
MCR>OPE 7342
007342/001025 240<ESC>
```

RSX-11D version 6.2:

```
MCR>OPE 11766
011766/001010 240<ESC>
MCR>OPE 12076
012076/001025 240<ESC>
```

We do not know the patch locations for RSX-11D version 6B or for IAS at present.

Once the emulator is running and the executive has been patched, the system can be saved using the SAVE command, making the emulator a permanent part of the system.

The emulator source gives complete instructions for assembly, task building, and installation.

STUDIES OF TIMING AND STATISTICS

We checked out the emulator by running the Whetstone benchmarks,[2] FORTRAN programs that spend most of their time doing floating point computation. Table 1 shows the benchmark results, and compares them with results for some other hardware and software configurations.

Table 2 shows the results of running the Whetstone benchmarks with the version of the emulator that gathers statistics on the instructions executed. It is interesting to note that in the single precision case, about 40% of the instructions do floating point arithmetic, and 60% do data movement and testing. The results are similar for the double precision case.

The large proportion of data movement and test instructions may account for the great difference in speed between the emulator and the FORTRAN IV software (see Table 1). The software can do these operations very quickly, whereas the emulator incurs a large overhead for trap service and instruction decoding no matter how simple the actual instruction simulation is. We have done some tuning of the emulator by trying to speed up the execution of the most frequently used instructions and addressing modes, but the gain in speed has been only about 25%. It has been suggested that more speed might be gained by looking ahead to the next instruction and not doing the trap again if it is a floating point operation. We will investigate this as time permits.

The emulator should be especially suitable for use with a FORTRAN program that does mostly integer arithmetic, with a small fraction of floating point computation. In this type of program, the gain in speed for integer operations of FORTRAN IV Plus over FORTRAN IV would more than offset the loss in speed for floating point operations.

PUTTING THE EMULATOR IN RSX-11M

Although we have no requirement to run the emulator under RSX-11M at this time, we did some studies of the work that would be involved in making it run under this system. A preliminary version actually ran but contained no code for processing error conditions. Some notes on the conversion effort are given below.

Three ways to put the emulator into the system suggest themselves:

1. Build it into the executive.
2. Make it a loadable driver.
3. Make it a privileged task that steals the reserved instruction trap and interfaces to the executive in the same general way that the RSX-11D version does.

Method 1 is the most elegant and involves the least trap servicing overhead. By conditionalizing the code, the emulator could be included in the executive or left out at system generation time. Since the emulator code is about 2K words long, there might be a problem with the executive size if too many other options were included.

Method 2 has the problem that the emulator is not really a driver, and an excessive amount of kludging might be required to convince the system that it was.

Method 3 was used to implement the preliminary version. It seems to work well but does not run as fast as Method 1 would. The preliminary version runs slightly slower than the RSX-11D version.

The emulator interfaces to the system at four points. We have summarized these below and given references to the relevant parts of the RSX-11M executive. Line numbers are for version 3.

1. Service of the reserved instruction trap (line 274 of module SSTSR; label $ILINS).
2. Service of floating point exceptions (line 220 of module SSTSR; label $FLTRP).
3. Service of odd address trap. This is a special problem because the RSX-11M executive does not seem to have any general code for dealing with a faulting instruction in a trap service routine (RSX-11D specifies a fault recovery sequence that can be used for instructions that might cause traps). The code at SSTXT (line 429 of module SSTSR) handles a number of system faults as special cases, and a case for a fault in the emulator could probably be added here.
4. Return to the system. This is done by a return (RTS PC) to the co-routine $INTSV (line 320 of module SYSXT) that was called at the beginning of the trap service.

Other parts of the executive of interest to one

wishing to put the emulator into RSX-11M are the context switching code (line 584 of module SYSXT) and the system communications area (module SYSCM).

TABLE 1: WHETSTONE BENCHMARK RESULTS

Configuration	Single precision speed (thousands of Whetstone instructions per second)	Double precision speed (thousands of Whetstone instructions per second)
1	5.2	3.2
2	23.3	6.1
3	210	153

1. PDP-11/34 running FORTRAN IV Plus and the emulator.

2. PDP-11/34 running FORTRAN IV.

3. PDP-11/45 equipped with FP-11B running FORTRAN IV Plus.

In all configurations, trace code was generated (/TR:BLOCKS for FORTRAN IV Plus, /SN for FORTRAN IV).

TABLE 2: WHETSTONE BENCHMARK STATISTICS

Instruction type or addressing mode	Fraction of instructions executed in single precision benchmark	Fraction of instructions executed in double precision benchmark
no operand	11.17%	9.17%
one operand	3.30	3.25
two operand	85.53	87.58
mode 0 Rn	47.78%	47.85%
mode 1 (Rn)	0.25	0.23
mode 2 #X or (Rn)+	13.27	16.27
mode 3 @(Rn)+	0.	0.
mode 4 -(Rn)	0.51	0.47
mode 5 @-(Rn)	0.	0.
mode 6 X or X(Rn)	31.97	28.53
mode 7 @X or @X(Rn)	7.22	6.65
single prec mode	89.38%	2.52%
double prec mode	10.62	97.48
LDF(D)	28.56%	27.67%
ADDF(D)	15.31	18.12
STF(D)	13.85	13.05
MULF(D)	13.01	15.95
CFCC	4.66	4.39
DIVF(D)	4.12	4.09
CMPF(D)	2.41	2.27
SUBF(D)	2.18	2.06
LDCDF(FD)	1.66	0.21
STCFI(DI)	1.05	0.99
MODF(D)	1.05	0.99
ABSF(D)	1.05	0.99
LDCIF(D)	1.00	0.94
STEXP	0.66	0.62
LDEXP	0.66	0.62
TSTF(D)	0.59	0.70
NEGF(D)	0.53	0.50
CLRF(D)	0.46	0.43
LDFPS	0.44	0.42
STFPS	0.22	0.21
SETF	4.63	0.00
SETD	0.83	3.79
SETI	1.05	0.99
SETL	0.00	0.00

ACKNOWLEDGMENTS

We should like to thank Dennis Perry and Sally Shlaer for the support and encouragement they gave us in this project. We also want to thank Brooks Shera, who made his PDP-11/34 available for checkout of the code, and who helped a great deal with the benchmarking studies.

Special thanks are due to Lorrie Voorhees, Elvira Martinez, and Janis Builta for their help with the preparation of this paper.

REFERENCES

1. The Multi-Tasker, Newsletter of the RSX-11D Special Interest Group, vol. 5, no. 5, p. 74 (May 1976).

2. Nicholas Benwell, ed., Benchmarking (Hemisphere Publishing Corp., Washington, 1975), article by Harry J. Curnow, pp. 99-114.