Performance of the HARRIS RTX 2000 Stack Architecture versus the Sun 4 SPARC and the Sun 3 M68020 Architectures

William F. Keown, Jr., Philip Koopman, Jr., Aaron Collins[1]

ABSTRACT

This study compares a stack machine, the Harris RTX 2000, a RISC machine, the Sun 4 /SPARC, and a CISC machine, the Sun3/M68020. An attempt is made to compare the generic features of each machine which are characteristic of their architectural classes as opposed to being characteristic of the individual machine only. Performance is compared based on execution of the Stanford Integer Benchmark series (12) and on interrupt response characteristics. The data indicates that, for these benchmarks, the RTX stack architecture approaches or exceeds the SPARC machine performance for such measures as total execution cycles required, clock cycles per instruction, native MIPS, static code size, and dynamic instruction count. The 68020 machine is by far the slowest of the three. When scaled to account for disparities in process technology, the RTX 2000 is as fast as (or faster than) the SPARC in actual program execution time, and it has a smaller code size.

Introduction

Although the Stack Architecture has been criticized as a general purpose computer architecture by some researchers (1,2,3), some of the arguments against this approach can be countered by some equally compelling arguments (4,5,6,7,8). For real-time applications, especially for interrupt intensive applications, the stack machine has some distinctive advantages. This study compares a stack machine, the Harris RTX 2000(9), a RISC machine, the Sun 4 /SPARC (10,11), and a CISC machine, the Sun3/M68020(14). An attempt is made to compare the generic features of each machine which are characteristic of their architectural classes as opposed to being characteristic of the individual machine only. The only stack architecture available to us for benchmarking is a 16-bit embedded controller which is not suitable for executing Unix or workstation applications. Therefore, we chose to use the Stanford Integer benchmarks (13) instead of the more comprehensive, but unsuitable for RTX execution, SPEC benchmarks.

Stack Computers vs. RISC vs. CISC

Stack Machines have many RISC characteristics. Of special note are their small instruction set and their ability to execute an instruction in a single clock cycle. Features that are uniquely found on stack machines include subroutine calls and interrupt context switches which have near zero cost in terms of execution time. This extremely low context switching time for servicing interrupts suggests that stack machines will also be especially effective for real-time applications, but that issue is addressed in a followup paper (). CISC machines are known to have features which are directly supportive of high level languages. The same can be said of stack machines in that a low context switch time supports of a high level of modularity in the form of "cheap" subprogram calls. In addition, their static code size is extremely small due to their extensive use of implied addressing. This study gathers data to compare RISC, CISC, and stack machines based on their ability to execute this set of benchmarks.

---

[1] Philip Koopman was with Harris Corporation during this study, and is now with United Technologies Research Center, M/S 48, East Hartford, CT 06108. William F. Keown and Aaron Collins are with Clemson Univ. ECE Dept., 102 Riggs Hall, Clemson, SC 29634.

Systems Used

RTX Original

The stack processor used is the Harris RTX 2000. It was programmed using the RTXDS, a development system provided by Harris for the RTX 2000. It is an IBM PC-based package which provides an interface between the host PC and the RTX target monitor. The RTX 2000 is a 16-bit microcontroller which operates at 10 MHz. It has on-chip a 256-word parameter stack and a 256-word return stack. The top two elements of the parameter stack and the top element of the return stack are stored in registers. Three on-chip 16-bit internal counters are decremented at each clock cycle. All instructions are executed in a single clock cycle except memory access instructions and long literal loads, which take two clock cycles. The RTX does not use cache memories, but instead uses system memory that is fast enough to guarantee single-cycle memory access. This includes the single-cycle 16-bit on-chip multiply instruction. The instruction set of the RTX 2000 is a superset of the primitives used by the Forth programming language. Thus, Forth can be used as an "assembly language" for the RTX. In some cases multiple Forth primitives can be peephole optimized into a single RTX instruction by exploiting parallel data paths inside the chip. Parallelism in the architecture allows subroutine returns to be executed at the same time as other instructions.

RTX Improved Compiler:

The C compiler used for the RTX was a pre-release version which still had some optimization features under development at Harris Corporation at the time that this study was initially performed. An analysis of the code indicated that a substantial number of inefficient structures were used by this C compiler, and that some of these inefficiencies were in critical parts of the code such as in the implementation of conditional loops. Specifically, the loop index was saved to memory repeatedly instead of being kept on the stack in the CPU. Subsequently, additional data was gathered with an improved version of the C compiler, and as will be seen from data presented below, the improvement was significant. The optimization level of the RTX compiler is equivalent to that of the Sun 4 Unix C compiler with level 2 optimization as described below.

RTX Improved Instruction Set:

The RTX was designed primarily to execute the Forth programming language. Although this language has excellent primitives for implementing other languages, it is not optimal for C. Some instruction set changes were proposed by the RTX design team and their effects were simulated in order to better evaluate the potential of stack architectures as opposed to evaluation of this particular stack architecture. The only proposed change of significant consequence was an increment/decrement by N operation for the register used as the C frame pointer.

Sun 4 SPARC:

The only RISC-based machine easily available for this study was the SUN4/110, which uses a SPARC chip set. This machine operates at 14.28 MHz. It has separate integer and floating-point processing units which operate concurrently. There are separate 32-bit address and data busses. It has 128 registers divided into eight overlapping windows of 24 registers each, for quicker context switching. There are also eight global general-purpose registers and 32 global floating-point registers. The instruction set consists of 50 instructions. Each of the instructions can be executed in a single clock cycle with the exception of loads/stores (the only instructions for accessing memory) and floating-point instructions. Instructions flow through a pipeline, and conditional branches are handled using a one-instruction delayed-branching technique. Unlike the RTX, the SPARC has an off-chip cache memory. The Sun 4 UNIX C compiler has three levels of optimization. They are:

(1) assembly level local optimization,
(2) level 1 plus automatic register allocation and loop-invariant code motion, and
(3) level 2 plus optimization expressions using global variables or indirect memory references.

Sun 3 M68020:
Several CISC machines were available. The SUN3/160, which is also based on Motorola's 68020 was selected because of the system similarity to the RISC selection. The 68020 has separate 32-bit address and data busses. It has eight 32-bit data and eight 32-bit address registers. There are 110 instructions and 18 modes of addressing. Instructions are decoded in a 3-stage pipeline. The most recently accessed instructions are stored in an on-chip instruction cache. The cache (256 bytes) holds 64 long words and is accessed by direct mapping. The SUN3 operates in 16.67 MHz and uses the MC68881 floating-point coprocessor. The Sun 3 UNIX C compiler has only one level of optimization, which is equivalent to the SUN 4 level 2 optimization.

Benchmarks

Since bigger benchmarks such as SPEC (13) wouldn't fit on the RTX, a 16-bit controller, the Stanford Integer benchmarks were chosen for this evaluation. The Stanford Integer benchmarks (12) are a set of short C programs, gathered by John Hennessy and modified by Peter Nye. They include no I/O and use no floating-point numbers, therefore test only the central processor and not the entire system. Although they do not provide a reliable indication of a system's performance when running large code, they are useful. Most real-time programs for the RTX-class machines tend to have small inner loops for control operations, which corresponds to the structures of the Stanford benchmarks. The small code size with a large number of calls to subroutines is useful in evaluating real-time processors since this is the way many real-time systems are written. Also, no acceptable real-time embedded control benchmarks exist, or they would have been used instead.

The individual programs are described below:

Bubble.c:	Randomly generates an array of 500 integers which are then sorted using a bubble sort algorithm.
Intmm.c:	Multiplies two 40x40 integer matrices.
Perm.c:	Heavily recursive permutation program.
Queen.c:	Solves the 8 queens problem 50 times.
Quick.c:	Randomly generates an array of 5000 integers which are sorted using a quick sort algorithm.
Towers.c	Solves the 'Towers of Hanoi' problem using 14 disks.

Test Facilities and Methods

Dynamic Instruction Count and Instruction Mix:
Each C compiler used had an option to produce an assembly code listing of the source. From these, a count was made of the number of instructions in each basic block of code for every benchmark. Also tallied were the frequency of instructions in the different classes of instructions. These numbers were placed in the C code as counter increments. Each time a block of code, either a loop or a straight line segment, was executed, the appropriate counters were incremented. This produced a count of the total number of instructions executed for all configurations and benchmarks, Table 2 and Figure 3, and the instruction mix for the RTX Original and the Sun 4 SPARC, Figure 1.

Static Code Size:
The static code size of each benchmark was obtained for the RTX Original and SPARC, Figure 1. On the two Sun systems, the code size was obtained by producing the assembly code, assembling this code, and using the system function, size(). This gave an accurate measure, verified by physically counting assembler instructions in several of the programs. The RTX code size was plotted only for the original configuration, and had to be physically counted since its linker included many system library functions which were never called.

Execution Time:
Execution times were obtained in two ways. A theoretical execution time (Table I) was calculated based on known information about the execution times for the different instructions. The actual CPU times (Table I), were obtained by using the setitimer() and getitimer() functions on the Suns and the gettimer() and calctime() functions on the RTX. The theoretical time is useful in this study, since it reflects only the performance of the CPU, and not any coprocessors or other system-dependent extras. The actual execution times give some credibility to the theoretical values and to the cpu time measurement of the system. These system measurements will be needed when observing real-time performance.

In calculating execution times of the SUN4, it was assumed that the only instructions evaluated which required more than a single clock cycle to execute were load, store, mul, and div. Based on the results of experiments which timed multiple loads and stores to elements which should be in cache, it was determined that the best case load is in two clock cycles and best case store is in four cycles. The mul instruction takes 47-51 cycles, depending on the signs of the operands. The div instruction was estimated to take 120 cycles, based on a count of the code given in the SPARC Architecture Manual (10).

The SUN3 timing information is supplied by Motorola. Because of the unknown state of the cache and pipeline, exact numbers cannot be given. However, a best case (which assumes data is located in the cache and the pipeline is full) and a worst case (which assumes a cache miss and the pipeline is empty) can give maximum and minimum bounds.

The RTX timing was the simplest since all operations, including the 16-bit multiply, execute in a single clock cycle except memory loads/stores and long literal loads. These exceptions have opcodes whose most significant four bits are in the range of 0xC - 0xE and execute in two cycles.

Computational Results

Instruction Count:
The dynamic instruction count for each CPU system and for each benchmark is shown in Table 2, and the average over all six benchmarks for each CPU system is shown in Figure 3. Although the RTX original system executed an average of 197% as many instructions as the SPARC machine, the RTX machine with both an improved compiler and an improved instruction set executed an average of only 132% as many instructions as the SPARC machine. The RTX was expected to execute more instructions than a RISC machine, because it too has a simple single-cycle instruction set, and must execute additional instructions to manipulate stack values since it cannot randomly access its stack as a RISC machine can randomly access its register file. It is reasonable to expect that further efforts can further improve the performance of stack machines. As expected, the M68020 machine required fewer instructions than the RISC and Stack systems, although not for every benchmark. The difference in dynamic instruction count between this CISC machine and the RISC and stack machines is less than would normally be expected, possibly because this set of benchmarks and the compiler does not fully exercise the power of the M68020 instruction set.

Instruction Mix:
As shown in Figure 1, the RTX, when compared to the Sun 4 SPARC, performs a significantly larger number of loads from memory and register/stack moves, and about the same number of math and logic operations. The high number of loads will be discussed later, but it is attributed to the lack of registers which could hold frequently used variables. Instead, all variables must all be stored in memory unless the compiler can manage them on the stack.

The differences between the number of reg/stack moves and the math/logic operations may not be as significant as they appear on the chart. The RTX can perform FORTH operations in parallel, within a single RTX machine language instruction. For example, the RTX instruction 'LIT 01 AND' is classified as a stack move since it pushes 01 onto the top of the parameter stack. However, it also performs a logical AND operation with the value originally on top of the stack. This is counted as one reg/stack move instruction and zero arithmetic/logic operations, and occurs very frequently. This overlap in the stack moves and the math/logic operations categories indicate that the reg/stack moves count as charted is too high, and the arithmetic operations count is too low.

Static Code Size:

Two counts were made for the SPARC code. One count looked at code which had not been optimized by the compiler, while the other count was for code which had been optimized at level 2 by the compiler. The SPARC calls to system multiply and divide operations (mul and div) were counted separately since they are not performed in one clock cycle. Two counts were also made for the 68020 code, optimized and not optimized. No instruction mix was obtained from the 68020, since it was more difficult to break the instructions into well-defined classes. The RTX 2000 by default does a limited amount of optimizing, combining instructions which can be performed in parallel into a single clock cycle. There is no option for further optimization. Therefore, only one count was taken for the RTX code.

A comparison of static code size is somewhat biased, since the RTX 2000 is a 16-bit processor while the others are 32-bits. A comparison between the static code size and dynamic instruction count may be useful. The static code size of the RTX was 0.9 that of the SPARC system while the dynamic instruction count of the RTX was almost twice that of the SPARC. The RTX code used in this comparison was for the original configuration. Data for the improved compiler and instruction set have not been plotted.

Clock Cycles per Instruction:

Figure 6 shows that the RTX requires fewer clock cycles per instruction than the SPARC machine for these benchmarks. This can be predicted by observing that the RTX executes all instructions in a single clock cycle except memory accesses and long literal loads. Subroutine calls are also especially cheap on the RTX. The M68020 machine, as a CISC machine, takes from five to nine clock cycles per instruction for most of the benchmarks.

MIPS:

Although native MIPS yield limited information about performance, it is interesting to see how these machines compare when native MIPS are measured. Figure 7 shows that although the SPARC has a much faster clock than the RTX, for most benchmarks it does not exceed the native MIPS rate of the RTX by very much. The M68020 does not look very good when native MIPS are used for comparisons. This measure is especially biased against CISC machines. It is interesting to note that even though the RTX performance improved as the compiler and instruction set were improved, the native MIPS rating actually went down. This is yet another indictment of native MIPS as a computer system comparison criterion.

Execution Time:

As can be seen from Figure 5, the original RTX system required more than twice as much execution time as the SPARC system. When both the improved compiler and the improved instruction set version of the RTX was used, execution time was only about sixty percent higher. The Sun3/M68020 required over three times as much execution time as the SPARC system. The SPARC, RTX, and M68020 were running at 14.28, 10, and 16.67 MHz, respectively.

Execution Cycles:

The RTX fabrication process (2.0 micron standard cell) places it at a significant disadvantage compared to both the M68020 and SPARC, so it seems reasonable to at least normalize the RTX performance to equal clock speeds of the other machines, and therefore use number of clock cycles executed to measure performance. The results shown in Table 2 and Figure 5 show that the RTX, especially the improved versions, requires only slightly more execution cycles than the SPARC. Specifically, the average number of execution cycles measured over all benchmarks required for the RTX with improved compiler and instruction set was 113% as many as were required for the SPARC. The M68020 machine required 397% as many execution cycles as the SPARC. For one benchmark, the integer matrix multiply, the RTX executes faster than the SPARC. This is attributed to the fast integer multiply which is built into the RTX. Given that the RTX was fabricated using an old process technology and speed-inefficient standard-cell design techniques, it is an open question whether the RTX or the SPARC would be faster given comparable levels of design effort and implementation technology.

Real-Time System Processing

Additional evaluation of the RTX was also performed. The significant results were derived from the observation that the RTX requires only four clock cycles to perform a context switch in response to an interrupt, whereas the other two machines require up to hundreds of clock cycles. The RTX, and most stack machines, have a near zero cost for interrupt switch time, therefore can handle a much higher interrupt rate. A followup report will explore these observations.

Conclusions

The performance observations which are drawn from the above information are:

Clock cycle count, Table 2 and Figure 5, and native MIPS, Figure 7, for the RTX were similar to, but not quite as good as, those of the SPARC machine. Figure 5 is a technology-neutral snapshot comparison of the architectures.

Static code size, Figure 2, clock cycles per instruction, Figure 6, and especially interrupt context switching time were all much better on the RTX than on the SPARC.

Count of instructions executed, Table 2 and Figure 3, and total execution time, Table 1 and Figure 4, were somewhat better for the SPARC than for the RTX.

The execution speed of the RTX, when normalized for clock speed, was surprisingly close to the SPARC RISC chip, and well above the performance range of the CISC machine. Preliminary investigations indicate that the RTX has superior context switching times and interrupt response. This leads to a speculation that the RTX may be superior to the SPARC for its intended application area of real-time embedded control. A future publication will investigate this issue more thoroughly.

References

1. Amdahl, G.M., G.A. Blaauw, and F.P.Brooks, Jr., "Architecture of the IBM System 360," IBM J. Research and Development 8:2, p87-101, April 1964.
2. Bell, G., R. Cady, H. McFarland, B Delagi, J.O'Laughlin, R. Noonan, and W. Wulf, "A new architecture for mini-computers: The DEC PDP-11," Proc. AFIPS SJCC, 657-675.Bell '70
3. Myers, G.J., "The evaluation of expressions in a storage-to-storage architecture," Computer Architecture News 7:3, p20-23, October 1978.
4. Koopman, P.J., Stack Computers: The New Wave, Halsted Press, New York, 1989.

5. Keedy, J.L. "On Evaluation of Expressions Using Accumulators, Stacks, and Store-to-Store Instructions," Computer Architecture News, 7:24-27, Dec 1978.
6. Keedy, J.L. "More on the Use of Stacks in the Evaluation of Expressions," Computer Architecture News,7:18-21, June 1979.
7. Tanenbaum, A.S. Implications of Structured Programming for Machine Architecture," Comm ACM, 21:237-243, March 1978.
8. Miller, D.L. "Stack Machines and Compiler Design, " Byte, 12:177-185, April 1987.
9. RTX Reference Manual. Harris Corp. 1988.
10. The SPARC Architecture Manual. Version 7. Sun Microsystems. 1987.
11. SPARC Release Notes for 4.0. Sun Microsystems. 1988.
12. Hennesey, John and Peter Nye, "Stanford Integer Benchmarks," Stanford University.
13. "SPEC Benchmark Suite Release 1.0," System Performance Evaluation Cooperative, October 2, 1989.
14. Sun 3 Reference Manuals, Sun Microsystems, 1987.

	RTX (10 MHz)	SPARC (14.28 MHz)	M68020 (16.7 MHz)
Bubble	0.88/0.76	0.21/0.22	(0.69 - 1.42)/0.64
Intmm	0.47/0.47	0.25/0.42	(0.61 - 1.17)/0.94
Perm	0.43/0.44	0.14/0.15	(0.36 - 0.93)/0.72
Queen	0.29/0.34	0.10/0.12	(0.27 - 0.61)/0.50
Quick	0.39/0.41	0.16/0.19	(0.35 - 0.74)/0.56
Tower	0.52/0.52	0.18/0.25	(0.53 - 1.06)/1.00

Table 1. Theoretical/Actual Execution Time (Sec)

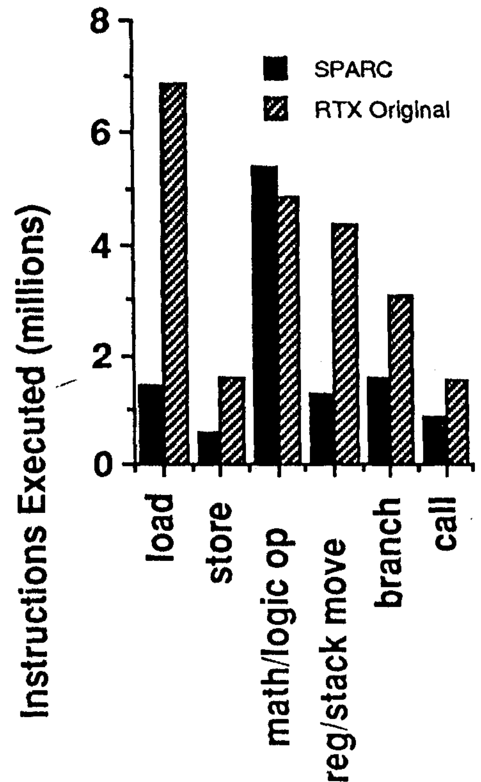

Figure 1. Instruction Mix

	RTX Original Compiler	RTX Improved Instr.Set	RTX Improv	SPARC	M68020
Clock Freq	10 MHz	10 MHz	10 MHz	14.28 MHz	16.7 MHz
total instr					
bubble	6373189	4235553	3354824	2011322	2295079
intmm	3448232	2675950	2456426	3040957	2127280
perm	2988694	2663996	2090781	1211872	1856092
queen	2118566	1503708	1438208	864054	1021059
quick	2805376	2608537	2122637	2086459	1191378
towers	3610429	3004180	2415018	1808020	1875104
clock cyc.					
bubble	7600000	6300000	5400000	3141600	10688000
intmm	4700000	3700000	3500000	5997600	15698000
perm	4400000	3700000	2900000	2142000	12024000
queen	2900000	2300000	2200000	1713600	8350000
quick	4100000	3600000	3100000	2713200	9352000
towers	5200000	4500000	3700000	3570000	16700000

Table 2. Dynamic Instruction Count and Total Clock Cycles for Stanford Integer Benchmarks

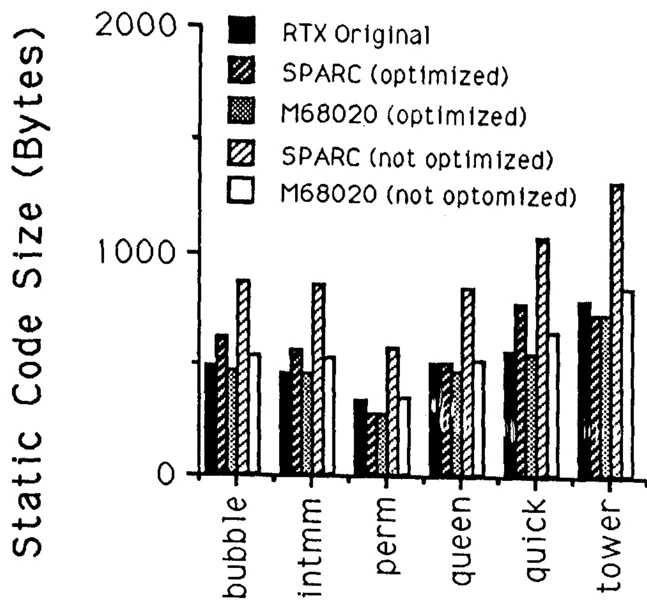

Figure 2. Static Code Size

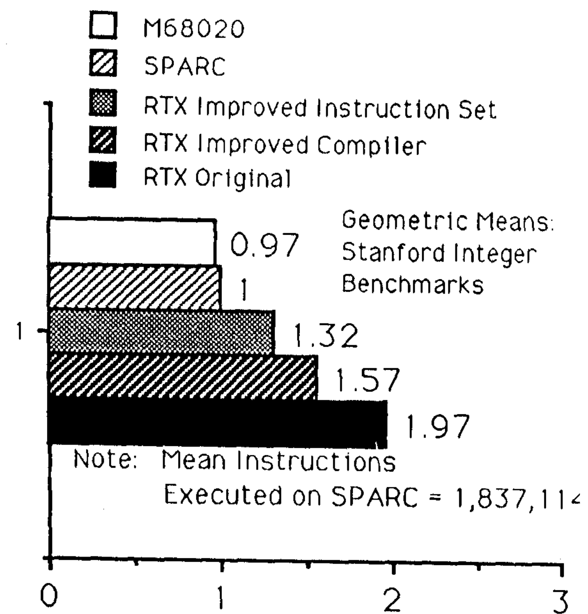

Fig 3. Instruct Count Ratio

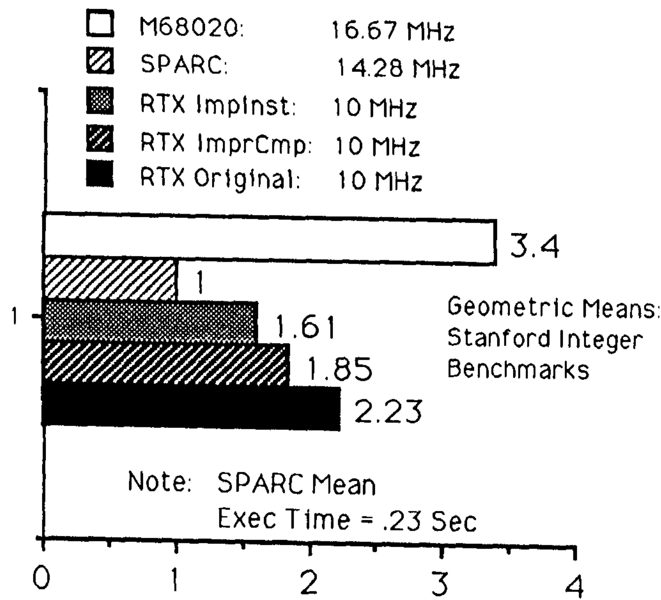

Fig 4. Exec Time Ratio

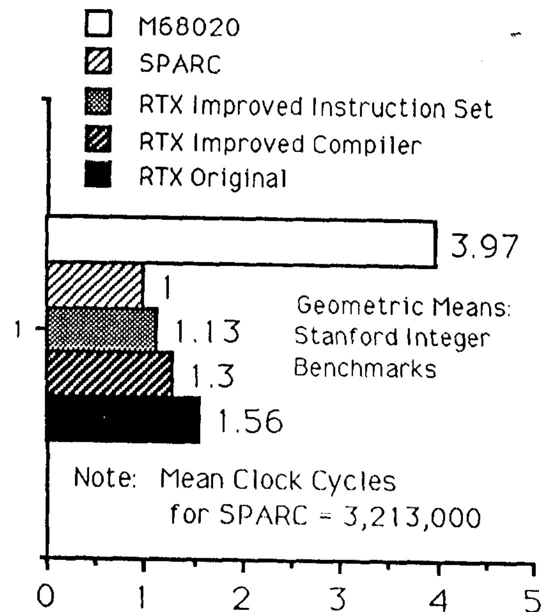

Fig 5. Clock Cycle Count Ratio

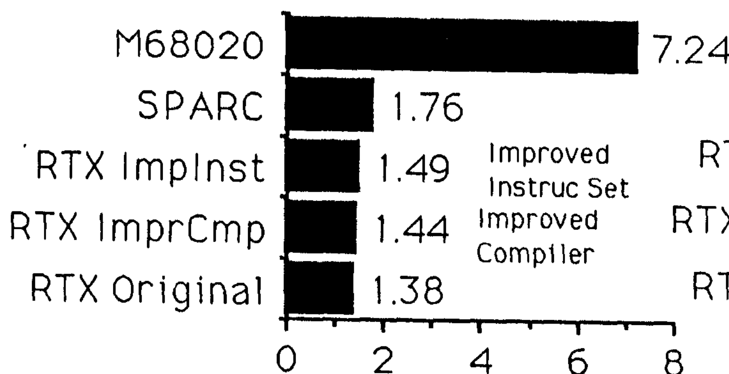

Fig 6. Clock Cycles / Instruction

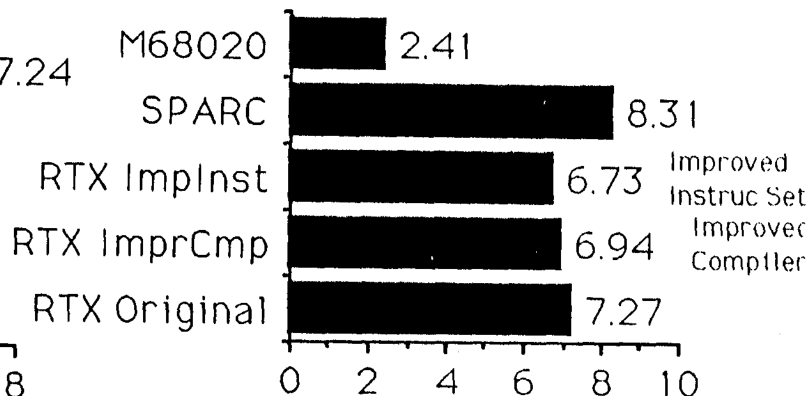

Fig. 7. Average Native MIPS