The 80386: A High Performance Workstation Microprocessor

Intel Corporation

June 1, 1986

The 80386 : A High Performance Workstation Microprocessor

Introduction

One of the most important issues in 32-bit system design is performance. Performance goals in such systems require that a microprocessor be able to deliver sustained high performance for both applications and operating systems.

Intel's new 32-bit microprocessor, the 80386, is specially optimized for use in high-performance systems. It establishes a new milestone in 32-bit system performance with features that include:

- Extensive execution pipelining,
- Integrated, on-chip, paged memory management,
- A high-throughput (32 Megabytes/second) bus, and
- High-performance floating-point coprocessor options.

This document first describes various techniques for measuring performance, followed by a discussion of the 80386 performance measurements and features responsible for its performance levels.

How is performance compared?

In the fiercely competitive 32-bit microprocessor market, claims of performance are often ambiguous. Objective comparison of performance is made difficult by the lack of a standard; a systematic evaluation of performance across different processors requires a common metric for measurement.

The ideal benchmarking method would be one that establishes a common basis for measurement — by having the same operating system and applications execute on all the systems being evaluated. In practice, however, this is hard to achieve. Even when the operating system is the same, implementations differ. As evidence, consider the plethora of UNIX[1]-like operating systems that are available for different microprocessor architectures.

---

[1] UNIX is a trademark of AT&T Bell Laboratories

Since absolute uniformity is impossible, the next best thing is to analyze the key elements that affect performance (once again, using a common metric for measurement). There are three main areas of interest:

1) Applications performance:

This should be a measure of general CPU performance that shows how well a processor supports high-level languages, such as C. As far as possible, the measure should be independent of operating system functions, in order to be a true indicator of performance available to applications.

2) Numerics performance:

This should be an operating-system-independent measure of floating-point performance. High-speed floating-point computations are crucial in scientific applications and high-speed graphics.

3) Operating system performance:

This should be an application-independent measure of performance of operating system functions, such as memory management, virtual-to-physical address translation, device I/O, interrupt response time, task switching, etc.

Which benchmarks should one choose to evaluate performance? Surely, this depends on how closely a benchmark resembles a typical application on the system. The closer a benchmark models the behavior of the target application, the greater is its authenticity in judging performance. The following sections will discuss the metrics used for comparison, which are the best to date that fulfill the above requirements.

What about MIPS?

It is common to rate a microprocessor in MIPS (millions of instructions per second). However, it is difficult and often misleading to compare the MIPS rating of one processor to that of another. The difficulty arises from differences in microprocessor architectures, and from the lack of general agreement about which data to use for computing MIPS.

For example, a RISC microprocessor will have a higher MIPS rating than a non-RISC microprocessor, simply because the former has a simpler instruction set. However, this does not imply that a RISC microprocessor delivers greater performance; since it typically requires more instructions to perform a given task. Clearly, MIPS by itself is not an accurate performance indicator. But, by quantifying it with the amount of work done, a MIPS rating can be put in proper perspective.

Given two processors A and B , and given the time it takes for each to do an equal amount of work, one can compare the MIPS rating of A to that of B. Consider the results of the Dhrystone benchmark (described in the next section). By equating the work (the Dhrystone benchmark), one can derive a MIPS rating for the 80386 realtive to other processors.

For example, the IBM RT PC, at 1880 Dhrystones per second, is rated as a 2.1 MIPS system (vendor claimed). In comparison, the 80386 computes 6133 Dhrystones per second. This gives the 80386 an effective rating of 6.85 (RT PC) MIPS.

As another example, the VAX[2] 11/780 using the UNIX V.2 C compiler, is measured at 1562 Dhrystones per second. It is generally acknowleged that the VAX 11/780 (with UNIX) is a 1 MIPS system. Again, a comparison with the 80386 Dhrystone performance yields a relative rating for the 80386 of 3.93 (VAX) MIPS.

However, when we consider that the VAX 11/780 using the UNIX 4.2 BSD C compiler executes 1662 Dhrystones per second, we are faced with two MIPS ratings for same processor. It should be obvious that MIPS alone is not sufficient to compare performance.

---

[2] VAX is a trademark of Digital Equipment Corporation

Application Performance: Dhrystone

The Dhrystone program is an independently derived synthetic benchmark originally published in CACM in ADA, and later transcribed to C (see Appendix A). The benchmark models programs written in high-level languages. It is balanced with respect to different kinds of high-level-language statements, data types, and operand accesses.

Table 1 shows that the instruction profile of the Dhrystone program, which represents over 28.3 million instructions executed dynamically. The profile is very similar to that obtained from a mix of 31 million instructions from 14 technical applications used to initially evaluate 386 performance. It is thus a valid measure for application-level performance.

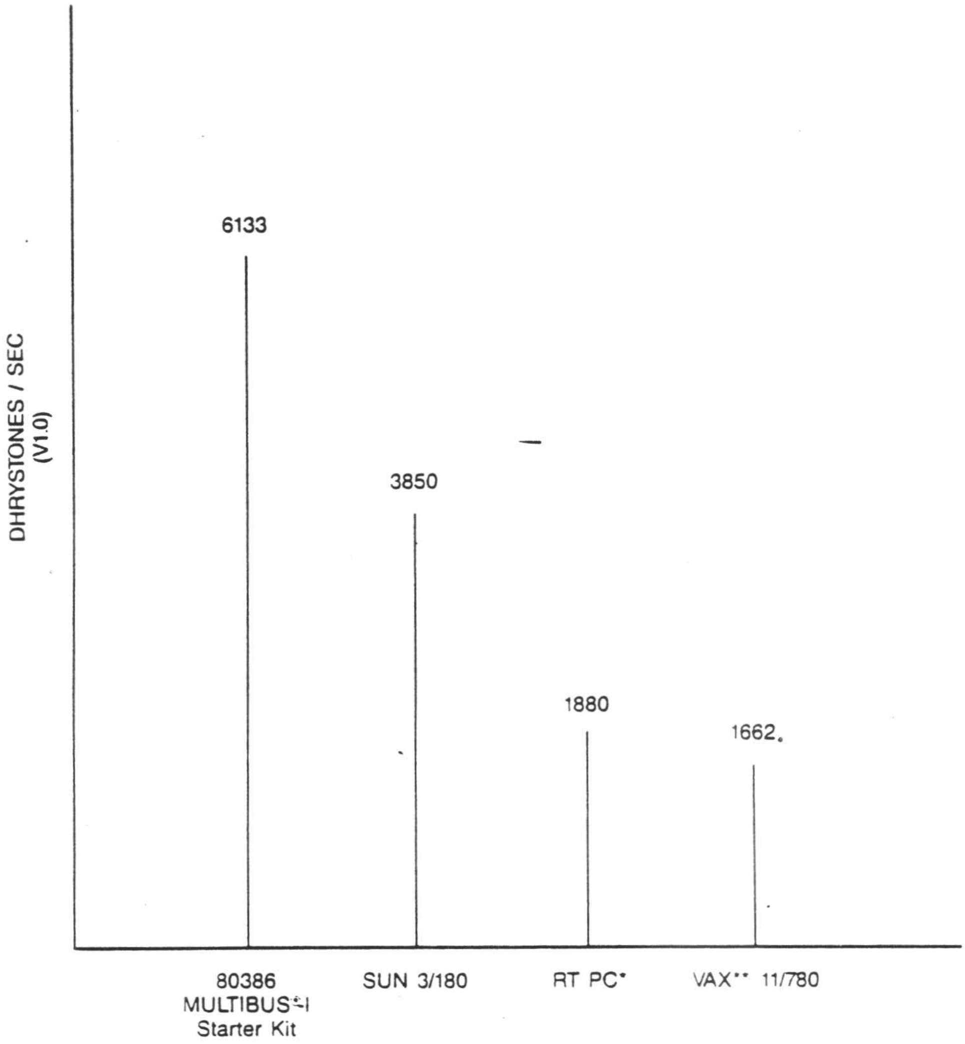
Table 1: Dhrystone instruction profile

Instruction Group	Dhrystone Frequency
MOVE (register/memory)	38%
ALU	24%
MOVE (register/register)	9%
BRANCH conditional	11%
BRANCH unconditional	11%
other	7%

Also, because the benchmark is written in C, it is representative of user-level applications that execute on most workstations. (Most workstations present the UNIX environment, which uses C as the programming language). The Dhrystone benchmark does not use any operating system functions, nor does it execute any floating-point operations. It measures a microprocessor's efficiency for common user-level applications, independent of the operating system and I/O implementation.

Figure 1 shows the 80386 Dhrystone performance compared to several well-known systems. The results were taken from the list of performance measurements maintained on Unix-net. The performance of the 80386 results from its efficient support of high-level languages, including a complete set of addressing modes, high instruction throughput, and fast data-movement capability. The following points outline the major features in these areas:

FIGURE 1: DHRYSTONE PERFORMANCE



Source: UNIX†-net (net.arch)

Configurations:

System	Microprocessor	Operating System	Compiler
386/20 MULTIBUS-I[1]	Intel 80386, 16 MHz 0 Wait States	PMON386 (Debugger)	Intel C386 V0.2 (Beta Version)
SUN 3/180	Motorola 68020, 16.67 MHz, SUN Proprietary MMU	SUN 4.2	CC
IBM RT PC	Proprietary	AIX††	CC
VAX 11/780	Proprietary	UNIX 4.2 BSD	CC

[1] 386/20 Starter Kit modified for 0 wait state performance

* RT PC is a trademark of IBM Corp.

** VAX is a trademark of Digital Equipment Corp.

† UNIX is a trademark of AT&T

†† AIX is a trademark of IBM Corp.

Flexible Addressing Modes

The 80386 instruction set provides addressing modes that map efficiently to high-level language addressing techniques. Any operand in the four Gigabyte linear address space can be addressed in the following manner:

BASE + (INDEX * SCALE) + DISPLACEMENT

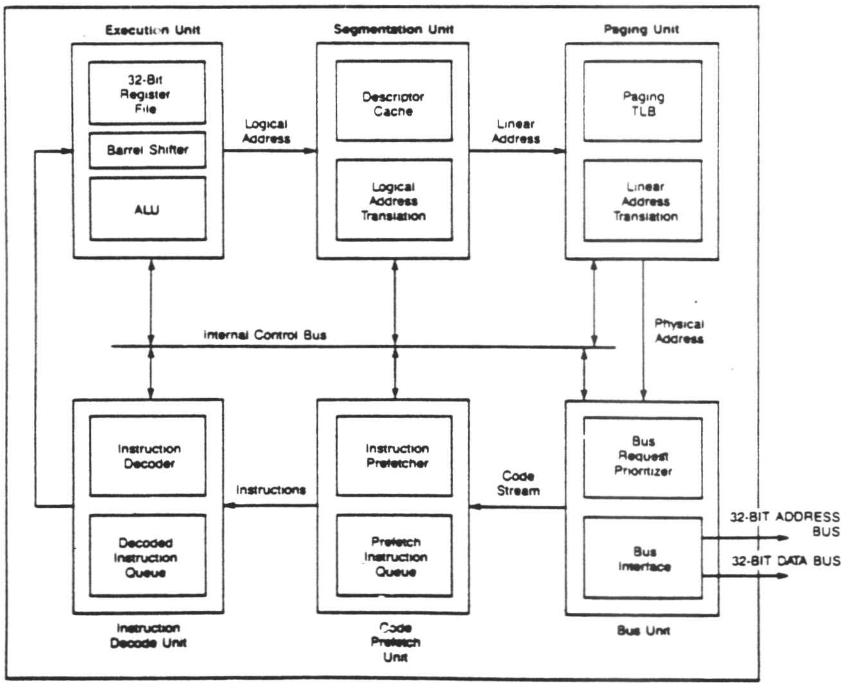
Any of the three components is optional. Any of the eight general purpose registers can specify the BASE, and, except for the stack pointer (ESP), any general purpose register can specify the INDEX. The DISPLACEMENT component (8 or 32 bits) is encoded within the instruction. The SCALE factor is either 1,2,4 or 8.

This flexibility of the addressing modes allows optimized code generation by compilers.

High Instruction Throughput

The 80386 uses six stages of pipelining to enhance its instruction processing throughput. Figure 2 shows the micro-architecture of the 80386. By overlapping various stages of instruction processing, the 80386 is able to process multiple instructions in parallel.

Figure 2: 80386 Microarchitecture



The Prefetch Unit continuously retrieves instructions from memory and deposits them into an internal queue, being careful not to interfere with operand fetches. The Decode Unit decodes instructions from the prefetch queue, and places decoded instructions within its own (decode) queue. With a steady stream of instructions passing through these two units, fetch and decode times totally overlapped with other instruction processing activities. When a branch is taken, the 80386 restarts pipelining from the target of the branch instruction. But, as seen from Table 1, this happens only about 15% of the time (assuming half of the conditional branches result in a transfer).

A substantial performance advantage of the 80386 comes from its speed in calculating effective addresses. Effective address computations, using two of the three components, are computed in one clock. In the worst case, it requires two clocks – this being when all three components are present. Due to the pipelined architecture, address calculations are overlapped with instruction execution, and do not add to the instruction execution time, except when the three component addressing mode is used, which adds one clock to the clock count of an instruction. Computing effective addresses quickly expedites accesses to memory operands.

Two-clock Data Bus

Data movement instructions are by far the most frequent instructions executed in programs. Typically, they account for 35% to 40% of all instructions executed. Table 1 shows the instruction profile of the Dhrystone benchmark; it places the frequency of memory-register data movement instructions at 38%. (Data-movement instructions are basic load/store type instructions such as: move between register and memory, stack pushes and pops, and string instructions. A significant portion of the Arithmetic and Logic (ALU) instructions also use memory operands, but they are included in the ALU group).

With a two-clock synchronous bus, the 80386 has a highly optimized ability to move data between itself and memory. As figure 3a illustrates, a memory-to-register transfer is performed in four clocks; while a register-to-memory transfer is performed in only two clocks (figure 3b). In the figures, instruction boundaries are indicated by solid vertical lines. Effective address computation and virtual address translation are performed in the first two clock periods, after which the data bus is accessed. At zero wait states the bus latency is two clocks. Note that for the register-to-memory move instruction, the memory write is performed by the Bus Unit in parallel with the next instruction.

Memory to memory operations – such as string moves – can also be performed at the full 32 Megabyte per second bus bandwidth, allowing high-performance block I/O transfers.

Figure 3a: Memory to Register move

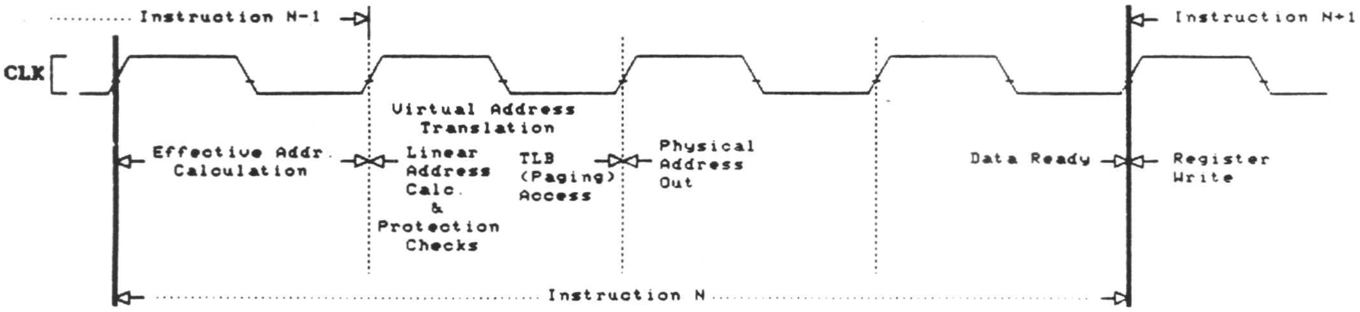
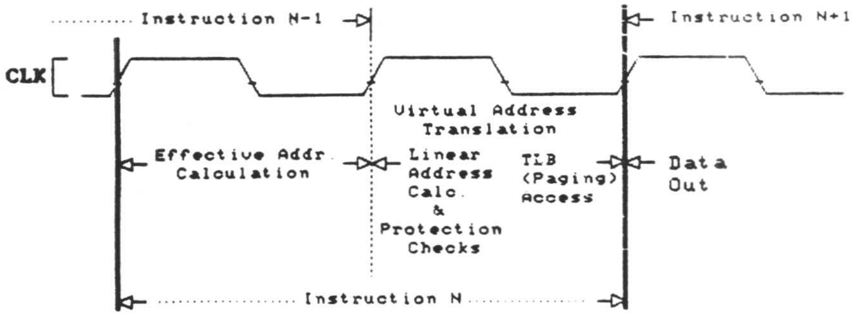


Figure 3b: Register to Memory move



Numerics Performance: Whetstone

Numerics performance is commonly measured with the Whetstone benchmark, with numeric speed quoted in Whetstones per second. Although the Whetstone program uses a good mix of floating-point operations, it also includes integer arithmetic, array indexing, function calls, conditional jumps and transcendental functions. Therefore, a Whetstone per second rating is a composite performance indicator, representing both the host processor and the floating-point coprocessor. As such, it is a reasonable approximation to a floating-point intensive scientific application. The Whetstone source listing is given in Appendix B.

Figure 4 shows the Whetstone performance of the 80386 in two configurations: with the 80387 numerics coprocessor, and with the Weitek 1167 floating-point chip set. The 1167 is comprised of three chips: the 1164 multiplier, the 1165 ALU, and the 1163 custom interface processor. The 1163 implements a 31x32-bit register file, the instruction decoder, exception handling, and the 80386 bus interface. At 16Mhz, the 80386/80387 pair performs at 1.8 million single precision Whetstones per second, while the 80386/1167 combination performs at 4 million single precision Whetstones per second.

The results for the 80387 and 1167 were computed from the numeric profile of the Whetstone benchmark. The profile lists each floating-point operation used in the Whetstone program, together with a count of how often that operation was executed. Given this information, and the clock counts for the 80387 and 1167 floating-point operations, the total time for executing the Whetstone benchmark was computed. Both the 80387 and 1167 will sample in the third quarter of 1986.

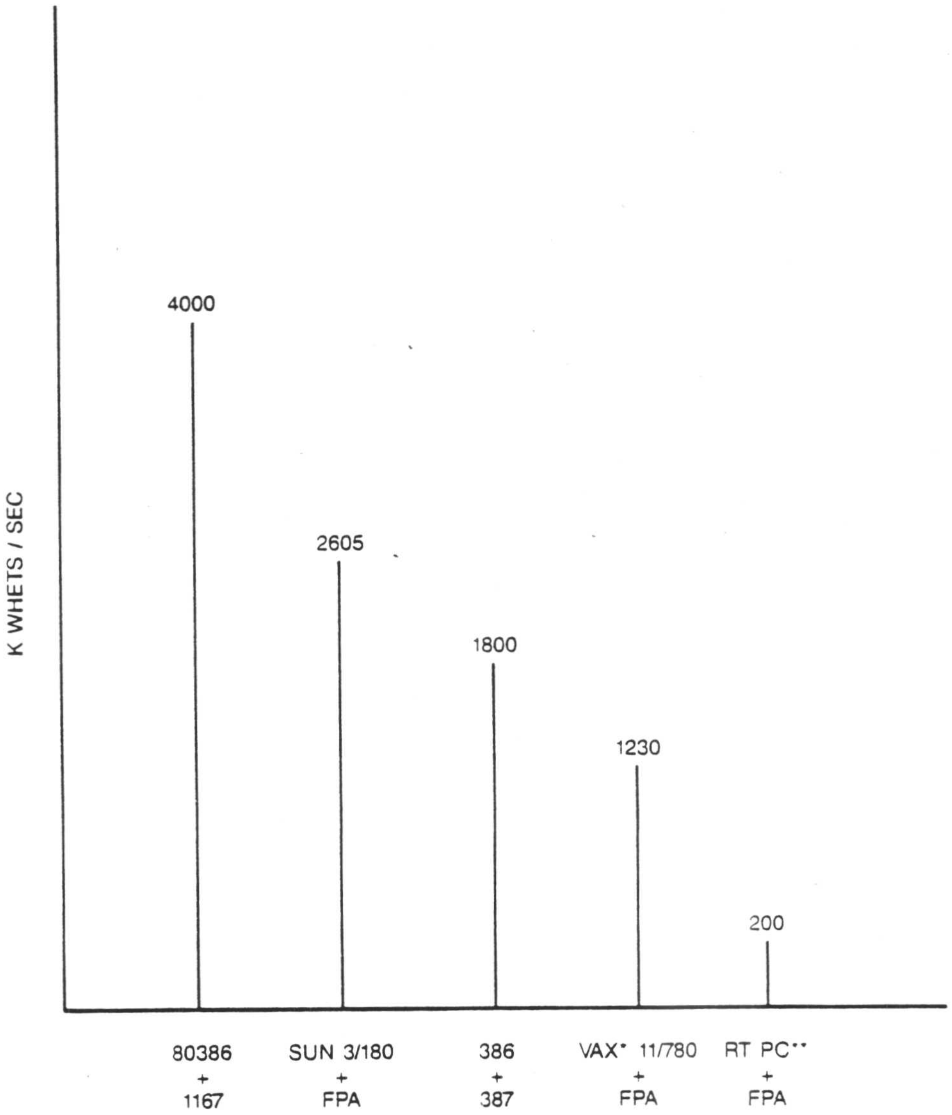
Optimized coprocessor interfaces

The 80386 and 80387 use a highly optimized, closely-coupled interface to communicate. This interface fully implements the IEEE 754 draft 10 floating-point standard, and provides 8087 and 80287 compatibility.

The 80386 uses a highly optimized, loosely-coupled interface to communicate with the 1167. The interface is memory-mapped; so, from the 80386's point of view, all floating-point operations are performed by simple MOV instructions.

The interface uses an elegant method to execute floating-point operations. The address specified by the 80386 MOV instruction encodes the floating-point instruction, while the 32-bit

FIGURE 4: WHETSTONE PERFORMANCE



Source:

80386 + 1167	— Derived from numeric profile of Whetstone (see text)
80386 + 80387	— Derived from numeric profile of Whetstone
SUN 3/180 + FPA	— UNIX†-net (net.arch); FPA uses Weitek 1164/1165
VAX 11/780 + FPA	— UNIX-net (net.arch); FPA proprietary
RT PC + FPA	— Vendor claimed (product announcement); FPA uses NS32081

* VAX is a trademark of Digital Equipment Corp.
** RT PC is a trademark of IBM Corp.
† UNIX is a trademark of AT&T

data bus provides an operand. (The floating-point instruction is encoded in 17 address bits, and only 64K of the 4Gbyte address space is used by the memory-mapped interface). In this manner, the 1167 obtains both the floating-point command and the operand in just two clocks. The 80386 bus bandwidth optimally matches the performance of the 1167 to provide greater floating-point performance than achievable with any other microprocessor.

In addition to fast operation/operand transfers, the 80386/1167 combination benefits greatly from concurrency. While the coprocessor is executing, the 80386 continues processing its own instruction stream. As long as the result of the previous floating-point operation is not needed, the 80386 does not need to wait. This overlap enhances the throughput for floating-point operations, and reduces the overall computation time for numeric applications.

The 4 MegaWhetones/sec performance level of the 80386/1167 combination allows over 60,000 vector transforms per second in systems requiring high-speed graphics.

Operating System Performance

The architectural features that effect high performance for high-level-languages also enhance performance in operating systems. For example, UNIX performance will benefit from C performance, because most of UNIX is written in C.

Other critical areas for operating system performance are: memory management, I/O, task-switching, and interrupt latency. In particular, virtual memory systems, such as demand-paged UNIX systems, require efficient implementations in all these areas. The 80386, with its on-chip operating systems support capabilities, provides the necessary elements for high-performance virtual memory designs.

A key component of virtual memory performance is the speed of mapping virtual addresses to physical addresses. By performing the translation on-chip, the 80386 is able to substantially improve address translation efficiency. On-chip memory management permits a two-fold optimization: it enables address translation to be pipelined along with other activities in the chip, and it allows the use of internal half-clock periods. When the translation information is present in on-chip caches, the virtual-to-physical translation is completed in a single clock; this includes both segment and page translations. (The paging-unit uses the latter half of the clock period). With pipelining, address translation is overlapped with other instruction execution. In contrast, off-chip memory management implementations cannot exploit internal pipelining. Off-chip approaches necessarily incur chip-access delays that makes address translation slower by one or more clocks.

To speed page translations, the 80386 paging unit contains a 32-entry translation look-aside buffer (TLB), or paging cache. The TLB contains the most recent page mappings. It is automatically updated by the processor from page tables in memory, whenever a miss occurs. The TLB spans a physical address space of 128 kilobytes, and has a hit rate of over 98%. With an on-chip TLB, address translation on the 80386 has no performance impact 98% of the time, while competitive (off-chip) approaches with similar hit rates have a one or two clock performance penalty 98% of the time.

Additionally, this one-clock address translation includes a robust set of protection checks, should a system designer choose to use them. Such hardware-enforced capabilities as preventing execution of data, writing into code space, user access of operating system data structures, and corruption of other address spaces are all performed within that same one-clock period. This allows high-performance virtual memory systems without compromising security and reliability.

Another important factor in virtual memory performance is the data transfer rate for swapping pages to and from memory. At 32 Megabytes per second, the 16 MHz 80386 data bus has ample I/O bandwidth to sustain high-speed DMA transfers. Table 2 compares the 80386 bus bandwidth with other 32-bit microprocessors (using the highest frequencies currently available).

Table 2: Bus Bandwidth (Megabytes/second)

	No Memory Management	With Memory Management
80386 (16 MHz)	32	32
MC68020 (20 MHz)	26.67	20 (with 1 clock MMU)
NS32032 (10 MHz)	10	8 (with 1 clock MMU)

The 80386 provides extensive hardware support for task management in multitasking systems. A hardware-assisted task switch can be completed within only 17 microseconds. A task switch entails saving the entire hardware context of the old task, and loading the hardware context of the new task. This includes comprehensive protection checks during the task switch.

Interrupts, on the 80386, can be handled by either procedures or tasks. This gives an operating system the flexibility to choose the context in which an interrupt is to be handled. An interrupt procedure runs in the context of the interupted task. With an interrupt latency of 3.7 microseconds, the 80386 allows fast response to events such as page faults and external interrupts. When a task is used to handle an interrupt, the 80386 automatically dispatches the task (on interrupt). This establishes an isolated context for handling the interrupt, and therefore promotes security in the operating system without impacting performance.

Summary

The 80386 provides the performance levels needed in demanding next-generation 32-bit systems. Its 6133 Dhrystone/second integer performance, 4 MegaWhetstone/second floating-point performance, 32 Megabytes/second bus bandwidth, and on-chip operating system performance features provide the capabilities needed in high-end UNIX workstations.

To equal the performance of a 16 MHz 80386, the SUN 3/180 (currently based on a 16.67 MHz 68020) needs a clock speed of 26.5 MHz. At 20 MHz, the 80386 will be in the 7500+ Dhrystones/second, 5 Million+ Whetstones/second performance class. An equivalent performance SUN system would require a clock speed of 33.2 Mhz, assuming the 68020 memory management and memory access speeds would be able to scale.

In addition, the memory management capabilities of the 80386 are extremely flexible, allowing any of the 4 Gigabyte direct physical, 4 Gigabyte flat demand-paged, segmented, and the 64 Terabyte segmented demand-paged environments to be implemented — all on-chip. Added to the 80386's capability to provide virtual 8086 machines as tasks in 32-bit operating systems, the 80386 has the performance, features, and flexibility to allow an OEM to create a wide range of differential advantages in his market.

TRADEMARKS

UNIX is a trademark of AT&T Bell Laboratories.

VAX is a trademark of Digital Equipment Corporation.

RT PC is a trademark of IBM Corporation.

APPENDIX A: Dhrystone Benchmark Listing

```
/* EVERBODY: Please read "APOLOGY" below. -rick 01/06/86
 *
 * "DHRYSTONE" Benchmark Program
 *
 * Version:     C/1.1, 12/01/84
 *
 * Date:        PROGRAM updated 01/06/86, RESULTS updated 02/17/86
 *
 * Author:              Reinhold P. Weicker, CACM Vol 27, No 10, 10/84 pg. 1013
 *                      Translated from ADA by Rick Richardson
 *                      Every method to preserve ADA-likeness has been used,
 *                      at the expense of C-ness.
 *
 *      Compile:        cc -O dry.c -o drynr                    : No registers
 *                      cc -O -DREG=register dry.c -o dryr      : Registers
 *
 *      Defines:        Defines are provided for old C compiler's
 *                      which don't have enums, and can't assign structures.
 *                      The time(2) function is library dependant; Most
 *                      return the time in seconds, but beware of some, like
 *                      Aztec C, which return other units.
 *                      The LOOPS define is initially set for 50000 loops.
 *                      If you have a machine with large integers and is
 *                      very fast, please change this number to 500000 to
 *                      get better accuracy.  Please select the way to
 *                      measure the execution time using the TIME define.
 *                      For single user machines, time(2) is adequate. For
 *                      multi-user machines where you cannot get single-user
 *                      access, use the times(2) function.  If you have
 *                      neither, use a stopwatch in the dead of night.
 *                      Use a "printf" at the point marked "start timer"
 *                      to begin your timings. DO NOT use the UNIX "time(1)"
 *                      command, as this will measure the total time to
 *                      run this program, which will (erroneously) include
 *                      the time to malloc(3) storage and to compute the
 *                      time it takes to do nothing.
 *
 *      Run:            drynr; dryr
 *
 *      Results:        If you get any new machine/OS results, please send to:
 *
 *                              {ihnp4,vax135,..}!houxm!castor!pcrat!rick
 *
 *                      and thanks to all that do.  Space prevents listing
 *                      the names of those who have provided some of these
 *                      results.  I'll be forwarding these results to
 *                      Rheinhold Weicker.
 *
 *      Note:           I order the list in increasing performance of the
 *                      "with registers" benchmark.  If the compiler doesn't
 *                      provide register variables, then the benchmark
 *                      is the same for both REG and NOREG.
 *
 *      PLEASE:         Send complete information about the machine type,
 *                      clock speed, OS and C manufacturer/version.  If
 *                      the machine is modified, tell me what was done.
 *                      On UNIX, execute uname -a and cc -V to get this info.
 *
```

```
 *      80x8x NOTE:     80x8x benchers: please try to do all memory models
 *                      for a particular compiler.
 *
 *      APOLOGY (1/30/86):
 *              Well, I goofed things up!  As pointed out by Haakon Bugge,
 *              the line of code marked "GOOF" below was missing from the
 *              Dhrystone distribution for the last several months.  It
 *              *WAS* in a backup copy I made last winter, so no doubt it
 *              was victimized by sleepy fingers operating vi!
 *
 *              The effect of the line missing is that the reported benchmarks
 *              are 15% too fast (at least on a 80286).  Now, this creates
 *              a dilema - do I throw out ALL the data so far collected
 *              and use only results from this (corrected) version, or
 *              do I just keep collecting data for the old version?
 *
 *              Since the data collected so far *is* valid as long as it
 *              is compared with like data, I have decided to keep
 *              TWO lists- one for the old benchmark, and one for the
 *              new.  This also gives me an opportunity to correct one
 *              other error I made in the instructions for this benchmark.
 *              My experience with C compilers has been mostly with
 *              UNIX 'pcc' derived compilers, where the 'optimizer' simply
 *              fixes sloppy code generation (peephole optimization).
 *              But today, there exist C compiler optimizers that will actually
 *              perform optimization in the Computer Science sense of the word,
 *              by removing, for example, assignments to a variable whose
 *              value is never used.  Dhrystone, unfortunately, provides
 *              lots of opportunities for this sort of optimization.
 *
 *              I request that benchmarkers re-run this new, corrected
 *              version of Dhrystone, turning off or bypassing optimizers
 *              which perform more than peephole optimization.  Please
 *              indicate the version of Dhrystone used when reporting the
 *              results to me.
 *
 * RESULTS BEGIN HERE
 *
 *----------------DHRYSTONE VERSION 1.0 RESULTS BEGIN--------------------------
 *
 * MACHINE      MICROPROCESSOR  OPERATING
 *                                              COMPILER        DHRYSTONES/SEC.
 * TYPE                         SYSTEM                          NO REG  REGS
 * --------------------------   ------------    -----------     ---------------
 * Commodore 64 6510-1MHz       C64 ROM         C Power 2.8       36      36
 * HP-110       8086-5.33Mhz    MSDOS 2.11      Lattice 2.14     284     284
 * IBM PC/XT    8088-4.77Mhz    PC/IX           cc               271     294
 * CCC 3205     ?               Xelos(SVR2)     cc               279     296
 * Perq-II      2901 bitslice   Accent S5c      cc (CMU)         301     301
 * IBM PC/XT    8088-4.77Mhz    COHERENT 2.3.43 MarkWilliams cc  296     317
 * Cosmos       68000-8Mhz      UniSoft         cc               305     322
 * IBM PC/XT    8088-4.77Mhz    Venix/86 2.0    cc               297     324
 * DEC PRO 350  11/23           Venix/PRO SVR2  cc               299     325
 * IBM PC       8088-4.77Mhz    MSDOS 2.0       b16cc 2.0        310     340
 * PDP11/23     11/23           Venix (V7)      cc               320     358
 * Commodore Amiga              ?               Lattice 3.02     368     371
 * PC/XT        8088-4.77Mhz    Venix/86 SYS V  cc               339     377
 * IBM PC       8088-4.77Mhz    MSDOS 2.0       CI-C86 2.20M     390     390
```

* IBM PC/XT	8088-4.77Mhz	PCDOS 2.1	Wizard 2.1	367	403	
* IBM PC/XT	8088-4.77Mhz	PCDOS 3.1	Lattice 2.15	403	403	@
* Colex DM-6	68010-8Mhz	Unisoft SYSV	cc	378	410	
* IBM PC	8088-4.77Mhz	PCDOS 3.1	Datalight 1.10	416	416	
* IBM PC	NEC V20-4.77Mhz	MSDOS 3.1	MS 3.1	387	420	
* IBM PC/XT	8088-4.77Mhz	PCDOS 2.1	Microsoft 3.0	390	427	
* IBM PC	NEC V20-4.77Mhz	MSDOS 3.1	MS 3.1 (186)	393	427	
* PDP-11/34	-	UNIX V7M	cc	387	438	
* IBM PC	8088, 4.77mhz	PC-DOS 2.1	Aztec C v3.2d	423	454	
* Tandy 1000	V20, 4.77mhz	MS-DOS 2.11	Aztec C v3.2d	423	458	
* Tandy TRS-16B	68000-6Mhz	Xenix 1.3.5	cc	438	458	
* PDP-11/34	-	RSTS/E	decus c	438	495	
* Onyx C8002	Z8000-4Mhz	IS/1 1.1 (V7)	cc	476	511	
* CCC 3230		Xelos (SysV.2)	cc	507	565	
* Tandy TRS-16B	68000-6Mhz	Xenix 1.3.5	Green Hills	609	617	
* DEC PRO 380	11/73	Venix/PRO SVR2	cc	577	628	
* FHL QT+	68000-10Mhz	Os9/68000	version 1.3	603	649	FH
* Apollo DN550	68010-?Mhz	AegisSR9/IX	cc 3.12	666	666	
* HP-110	8086-5.33Mhz	MSDOS 2.11	Aztec-C	641	676	
* ATT PC6300	8086-8Mhz	MSDOS 2.11	b16cc 2.0	632	684	
* IBM PC/AT	80286-6Mhz	PCDOS 3.0	CI-C86 2.1	666	684	
* Tandy 6000	68000-8Mhz	Xenix 3.0	cc	694	694	
* IBM PC/AT	80286-6Mhz	Xenix 3.0	cc	684	704	MM
* Macintosh	68000-7.8Mhz 2M	Mac Rom	Mac C 32 bit int	694	704	
* Macintosh	68000-7.7Mhz	-	MegaMax C 2.0	661	709	
* IBM PC/AT	80286-6Mhz	Xenix 3.0	cc	704	714	LM
* Codata 3300	68000-8Mhz	UniPlus+ (v7)	cc	678	725	
* WICAT MB	68000-8Mhz	System V	WICAT C 4.1	585	731	~
* Cadmus 9000	68010-10Mhz	UNIX	cc	714	735	
* AT&T 6300	8086-8Mhz	Venix/86 SVR2	cc	668	743	
* Cadmus 9790	68010-10Mhz 1MB	SVR0,Cadmus3.7	cc	720	747	
* NEC PC9801F	8086-8Mhz	PCDOS 2.11	Lattice 2.15	768	-	@
* ATT PC6300	8086-8Mhz	MSDOS 2.11	CI-C86 2.20M	769	769	
* Burroughs XE550	68010-10Mhz	Centix 2.10	cc	769	769	CT1
* EAGLE/TURBO	8086-8Mhz	Venix/86 SVR2	cc	696	779	
* ALTOS 586	8086-10Mhz	Xenix 3.0b	cc	724	793	
* DEC 11/73	J-11 micro	Ultrix-11 V3.0	System V	735	793	
* ATT 3B2/300	WE32000-?Mhz	UNIX 5.0.2	cc	735	806	
* Apollo DN320	68010-?Mhz	AegisSR9/IX	cc 3.12	806	806	
* IRIS-2400	68010-10Mhz	UNIX System V	cc	772	829	
* Atari 520ST	68000-8Mhz	TOS	DigResearch	839	846	
* IBM PC/AT	80286-6Mhz	PCDOS 3.0	MS 3.0(large)	833	847	LM
* WICAT MB	68000-8Mhz	System V	WICAT C 4.1	675	853	S~
* VAX 11/750	-	Ultrix 1.1	4.2BSD cc	781	862	
* CCC 7350A	68000-8MHz	UniSoft V.2	cc	821	875	
* VAX 11/750	-	UNIX 4.2bsd	cc	862	877	
* Fast Mac	68000-7.7Mhz	-	MegaMax C 2.0	839	904	+
* IBM PC/XT	8086-9.54Mhz	PCDOS 3.1	Microsoft 3.0	833	909	C1
* DEC 11/44		Ultrix-11 V3.0	System V	862	909	
* Macintosh	68000-7.8Mhz 2M	Mac Rom	Mac C 16 bit int	877	909	S
* CCC 3210	?	Xelos R01(SVR2)	cc	849	924	
* CCC 3220	?	Ed. 7 v2.3	cc	892	925	
* IBM PC/AT	80286-6Mhz	Xenix 3.0	cc -i	909	925	
* AT&T 6300	8086, 8mhz	MS-DOS 2.11	Aztec C v3.2d	862	943	
* IBM PC/AT	80286-6Mhz	Xenix 3.0	cc	892	961	
* VAX 11/750	w/FPA	Eunice 3.2	cc	914	976	
* IBM PC/XT	8086-9.54Mhz	PCDOS 3.1	Wizard 2.1	892	980	C1
* IBM PC/XT	8086-9.54Mhz	PCDOS 3.1	Lattice 2.15	980	980	C1

* Plexus P35	68000-10Mhz	UNIX System III	cc	984	980	
* PDP-11/73	KDJ11-AA 15Mhz	UNIX V7M 2.1	cc	862	981	
* VAX 11/750	w/FPA	UNIX 4.3bsd	cc	994	997	
* IRIS-1400	68010-10Mhz	UNIX System V	cc	909	1000	
* IBM PC/AT	80286-6Mhz	Venix/86 2.1	cc	961	1000	
* IBM PC/AT	80286-6Mhz	PCDOS 3.0	b16cc 2.0	943	1063	
* Zilog S8000/11	Z8001-5.5Mhz	Zeus 3.2	cc	1011	1084	
* NSC ICM-3216	NSC 32016-10Mhz	UNIX SVR2	cc	1041	1084	
* IBM PC/AT	80286-6Mhz	PCDOS 3.0	MS 3.0(small)	1063	1086	
* VAX 11/750	w/FPA	VMS	VAX-11 C 2.0	958	1091	
* Stride	68000-10Mhz	System-V/68	cc	1041	1111	
* Plexus P/60	MC68000-12.5Mhz	UNIX SYSIII	Plexus	1111	1111	
* ATT PC7300	68010-10Mhz	UNIX 5.2	cc	1041	1111	
* CCC 3230	?	Xelos R01(SVR2)	cc	1040	1126	
* Stride	68000-12Mhz	System-V/68	cc	1063	1136	
* IBM PC/AT	80286-6Mhz	Venix/286 SVR2	cc	1056	1149	
* Plexus P/60	MC68000-12.5Mhz	UNIX SYSIII	Plexus	1111	1163	T
* IBM PC/AT	80286-6Mhz	PCDOS 3.0	Datalight 1.10	1190	1190	
* ATT PC6300+	80286-6Mhz	MSDOS 3.1	b16cc 2.0	1111	1219	
* IBM PC/AT	80286-6Mhz	PCDOS 3.1	Wizard 2.1	1136	1219	
* Sun2/120	68010-10Mhz	Sun 4.2BSD	cc	1136	1219	
* IBM PC/AT	80286-6Mhz	PCDOS 3.0	CI-C86 2.20M	1219	1219	
* WICAT PB	68000-8Mhz	System V	WICAT C 4.1	998	1226	~
* MASSCOMP 500	68010-10MHz	RTU V3.0	cc (V3.2)	1156	1238	
* Alliant FX/8	IP (68012-12Mhz)	Concentrix	cc -ip;exec -i	1170	1243	FX
* Cyb DataMate	68010-12.5Mhz	Uniplus 5.0	Unisoft cc	1162	1250	
* PDP 11/70	-	UNIX 5.2	cc	1162	1250	
* IBM PC/AT	80286-6Mhz	PCDOS 3.1	Lattice 2.15	1250	1250	
* IBM PC/AT	80286-7.5Mhz	Venix/86 2.1	cc	1190	1315	*15
* Sun2/120	68010-10Mhz	Standalone	cc	1219	1315	
* Intel 380	80286-8Mhz	Xenix R3.0up1	cc	1250	1315	*16
* Sequent Balance 8000	NS32032-10MHz	Dynix 2.0	cc	1250	1315	N12
* IBM PC/DSI-32	32032-10Mhz	MSDOS 3.1	GreenHills 2.14	1282	1315	C3
* ATT 3B2/400	WE32100-?Mhz	UNIX 5.2	cc	1315	1315	
* CCC 3250XP	-	Xelos R01(SVR2)	cc	1215	1318	
* IBM PC/RT 032	RISC(801?)?Mhz	BSD 4.2	cc	1248	1333	RT
* DG MV4000	-	AOS/VS 5.00	cc	1333	1333	
* IBM PC/AT	80286-8Mhz	Venix/86 2.1	cc	1275	1380	*16
* IBM PC/AT	80286-6Mhz	MSDOS 3.0	Microsoft 3.0	1250	1388	
* ATT PC6300+	80286-6Mhz	MSDOS 3.1	CI-C86 2.20M	1428	1428	
* COMPAQ/286	80286-8Mhz	Venix/286 SVR2	cc	1326	1443	
* IBM PC/AT	80286-7.5Mhz	Venix/286 SVR2	cc	1333	1449	*15
* WICAT PB	68000-8Mhz	System V	WICAT C 4.1	1169	1464	S~
* Tandy II/6000	68000-8Mhz	Xenix 3.0	cc	1384	1477	
* WICAT MB	68000-12.5Mhz	System V	WICAT C 4.1	1246	1537	~
* IBM PC/AT	80286-9Mhz	SCO Xenix V	cc	1540	1556	*18
* Cyb DataMate	68010-12.5Mhz	Uniplus 5.0	Unisoft cc	1470	1562	S
* VAX 11/780	-	UNIX 5.2	cc	1515	1562	
* MicroVAX-II	-	-	-	1562	1612	
* VAX 11/780	-	UNIX 4.3bsd	cc	1646	1662	
* Apollo DN660	-	AegisSR9/IX	cc 3.12	1666	1666	
* ATT 3B20	-	UNIX 5.2	cc	1515	1724	
* NEC PC-98XA	80286-8Mhz	PCDOS 3.1	Lattice 2.15	1724	1724	@
* HP9000-500	B series CPU	HP-UX 4.02	cc	1724	-	
* IBM PC/STD	80286-8Mhz	MSDOS 3.0	Microsoft 3.0	1724	1785	C2
* WICAT MB	68000-12.5Mhz	System V	WICAT C 4.1	1450	1814	S~

```
 * WICAT PB      68000-12.5Mhz  System V          WICAT C 4.1      1530    1898 ~
 * DEC-2065      KL10-Model B   TOPS-20 6.1FT5    Port. C Comp.    1937    1946
 * Gould PN6005  -              UTX 1.1(4.2BSD)   cc               1675    1964
 * DEC2060       KL-10          TOPS-20           cc               2000    2000 &
 * VAX 11/785    -              UNIX 5.2          cc               2083    2083
 * VAX 11/785    -              VMS               VAX-11 C 2.0     2083    2083
 * VAX 11/785    -              UNIX SVR2         cc               2123    2083
 * VAX 11/785    -              ULTRIX-32 1.1     cc               2083    2091
 * VAX 11/785    -              UNIX 4.3bsd       cc               2135    2136
 * WICAT PB      68000-12.5Mhz  System V          WICAT C 4.1      1780    2233 S~
 * Pyramid 90x   -              OSx 2.3           cc               2272    2272
 * Pyramid 90x   FPA,cache,4Mb  OSx 2.5           cc no -O         2777    2777
 * Pyramid 90x   w/cache        OSx 2.5           cc w/-O          3333    3333
 * IBM-4341-II   -              VM/SP3            Waterloo C 1.2   3333    3333
 * IRIS-2400T    68020-16.67Mhz UNIX System V     cc               3105    3401
 * Celerity C-1200 ?            UNIX 4.2BSD       cc               3485    3468
 * SUN 3/75      68020-16.67Mhz SUN 4.2 V3        cc               3333    3571
 * IBM-4341      Model 12       UTS 5.0           ?                3685    3685
 * SUN-3/160     68020-16.67Mhz Sun 4.2 V3.0A     cc               3381    3764
 * Sun 3/180     68020-16.67Mhz Sun 4.2           cc               3333    3846
 * IBM-4341      Model 12       UTS 5.0           ?                3910    3910 MN
 * MC 5400       68020-16.67MHz RTU V3.0          cc (V4.0)        3952    4054
 * NCR Tower32   68020-16.67Mhz SYS 5.0 Rel 2.0 cc                 3846    4545
 * Gould PN9080 -               UTX-32 1.1c       cc               -       4629
 * MC 5600/5700 68020-16.67MHz  RTU V3.0          cc (V4.0)        4504    4746 %
 * Gould 1460-342 ECL proc      UTX/32 1.1/c      cc               5342    5677 G1
 * VAX 8600      -              UNIX 4.3bsd       cc               7024    7088
 * VAX 8600      -              VMS               VAX-11 C 2.0     7142    7142
 * Alliant FX/8 CE              Concentrix        cc -ce;exec -c 6952      7655 FX
 * CCI POWER 6/32               COS(SV+4.2)       cc               7500    7800
 * CCI POWER 6/32               POWER 6 UNIX/V  cc                 8236    8498
 * CCI POWER 6/32               4.2 Rel. 1.2b     cc               8963    9544
 * Sperry (CCI Power 6)         4.2BSD            cc               9345   10000
 * CRAY-X-MP/12  105Mhz         COS 1.14          Cray C          10204   10204
 * IBM-3083      -              UTS 5.0 Rel 1     cc              16666   12500
 * CRAY-1A             80Mhz    CTSS              Cray C 2.0      12100   13888
 * IBM-3083      -              VM/CMS HPO 3.4  Waterloo C 1.2 13889      13889
 * Amdahl 470 V/8               UTS/V 5.2         cc v1.23        15560   15560
 * CRAY-X-MP/48  105Mhz         CTSS              Cray C 2.0      15625   17857
 * Amdahl 580    -              UTS 5.0 Rel 1.2 cc v1.5           23076   23076
 * Amdahl 5860                  UTS/V 5.2         cc v1.23        28970   28970
 *
 * NOTE
 *   *   Crystal changed from 'stock' to listed value.
 *   +   This Macintosh was upgraded from 128K to 512K in such a way that
 *       the new 384K of memory is not slowed down by video generator accesses.
 *   %   Single processor; MC == MASSCOMP
 *   &   A version 7 C compiler written at New Mexico Tech.
 *   @    vanilla Lattice compiler used with MicroPro standard library
 *   S   Shorts used instead of ints
 *   T   with Chris Torek's patches (whatever they are).
 *   ~   For WICAT Systems: MB=MultiBus, PB=Proprietary Bus
 *   LM  Large Memory Model. (Otherwise, all 80x8x results are small model)
 *   MM  Medium Memory Model. (Otherwise, all 80x8x results are small model)
 *   C1  Univation PC TURBO Co-processor; 9.54Mhz 8086, 640K RAM
 *   C2  Seattle Telecom STD-286 board
 *   C3  Definicon DSI-32 coprocessor
 *   C?  Unknown co-processor board?
```

```
 *   CT1 Convergent Technologies MegaFrame, 1 processor.
 *   MN  Using Mike Newtons 'optimizer' (see net.sources).
 *   G1  This Gould machine has 2 processors and was able to run 2 dhrystone
 *       Benchmarks in parallel with no slowdown.
 *   FH  FHC == Frank Hogg Labs (Hazelwood Uniquad 2 in an FHL box).
 *   FX  The Alliant FX/8 is a system consisting of 1-8 CEs (computation
 *       engines) and 1-12 IPs (interactive processors). Note N8 applies.
 *   RT  This is one of the RT's that CMU has been using for awhile.  I'm
 *       not sure that this is identical to the machine that IBM is selling
 *       to the public.
 *   Nnn This machine has multiple processors, allowing "nn" copies of the
 *       benchmark to run in the same time as 1 copy.
 *   ?   I don't trust results marked with '?'.  These were sent to me with
 *       either incomplete info, or with times that just don't make sense.
 *       ?? means I think the performance is too poor, ?! means too good.
 *       If anybody can confirm these figures, please respond.
 *
 *  ABBREVIATIONS
 *      CCC     Concurrent Computer Corp. (was Perkin-Elmer)
 *      MC      Masscomp
 *
 *--------------------------------RESULTS END----------------------------------
 *
 *      The following program contains statements of a high-level programming
 *      language (C) in a distribution considered representative:
 *
 *      assignments                     53%
 *      control statements              32%
 *      procedure, function calls       15%
 *
 *      100 statements are dynamically executed.  The program is balanced with
 *      respect to the three aspects:
 *              - statement type
 *              - operand type (for simple data types)
 *              - operand access
 *                      operand global, local, parameter, or constant.
 *
 *      The combination of these three aspects is balanced only approximately.
 *
 *      The program does not compute anything meaningfull, but it is
 *      syntactically and semantically correct.
 *
 */

/* Accuracy of timings and human fatigue controlled by next two lines */
#define LOOPS   50000           /* Use this for slow or 16 bit machines */
/*#define LOOPS 500000          /* Use this for faster machines */

/* Compiler dependent options */
#undef  NOENUM                  /* Define if compiler has no enum's */
#undef  NOSTRUCTASSIGN          /* Define if compiler can't assign structures */

/* define only one of the next two defines */
#define TIMES                   /* Use times(2) time function */
/*#define TIME                  /* Use time(2) time function */

/* define the granularity of your times(2) function (when used) */
#define HZ      60              /* times(2) returns 1/60 second (most) */
```

```
/*#define HZ      100                /* times(2) returns 1/100 second (WECo) */

/* for compatibility with goofed up version */
/*#define GOOF                       /* Define if you want the goofed up version */

#ifdef GOOF
char      Version[] = "1.0";
#else
char      Version[] = "1.1";
#endif

#ifdef  NOSTRUCTASSIGN
#define structassign(d, s)           memcpy(&(d), &(s), sizeof(d))
#else
#define structassign(d, s)           d = s
#endif

#ifdef  NOENUM
#define Ident1  1
#define Ident2  2
#define Ident3  3
#define Ident4  4
#define Ident5  5
typedef int         Enumeration;
#else
typedef enum        {Ident1, Ident2, Ident3, Ident4, Ident5} Enumeration;
#endif

typedef int         OneToThirty;
typedef int         OneToFifty;
typedef char        CapitalLetter;
typedef char        String30[31];
typedef int         Array1Dim[51];
typedef int         Array2Dim[51][51];

struct  Record
{
          struct Record              *PtrComp;
          Enumeration                Discr;
          Enumeration                EnumComp;
          OneToFifty                 IntComp;
          String30                   StringComp;
};

typedef struct Record        RecordType;
typedef RecordType *         RecordPtr;
typedef int                  boolean;

#define NULL                 0
#define TRUE                 1
#define FALSE                0

#ifndef REG
#define REG
#endif

extern Enumeration           Func1();
extern boolean               Func2();
```

```
#ifdef TIMES
#include <sys/types.h>
#include <sys/times.h>
#endif

main()
{
        Proc0();
        exit(0);
}

/*
 * Package 1
 */
int                 IntGlob;
boolean             BoolGlob;
char                Char1Glob;
char                Char2Glob;
Array1Dim           Array1Glob;
Array2Dim           Array2Glob;
RecordPtr           PtrGlb;
RecordPtr           PtrGlbNext;

Proc0()
{
        OneToFifty              IntLoc1;
        REG OneToFifty          IntLoc2;
        OneToFifty              IntLoc3;
        REG char                CharLoc;
        REG char                CharIndex;
        Enumeration             EnumLoc;
        String30                String1Loc;
        String30                String2Loc;
        extern char             *malloc();

#ifdef TIME
        long                    time();
        long                    starttime;
        long                    benchtime;
        long                    nulltime;
        register unsigned int   i;

        starttime = time( (long *) 0);
        for (i = 0; i < LOOPS; ++i);
        nulltime = time( (long *) 0) - starttime; /* Computes o'head of loop */
#endif
#ifdef TIMES
        time_t                  starttime;
        time_t                  benchtime;
        time_t                  nulltime;
        struct tms              tms;
        register unsigned int   i;

        times(&tms); starttime = tms.tms_utime;
        for (i = 0; i < LOOPS; ++i);
        times(&tms);
        nulltime = tms.tms_utime - starttime; /* Computes overhead of looping */
#endif
```

```
        PtrGlbNext = (RecordPtr) malloc(sizeof(RecordType));
        PtrGlb = (RecordPtr) malloc(sizeof(RecordType));
        PtrGlb->PtrComp = PtrGlbNext;
        PtrGlb->Discr = Ident1;
        PtrGlb->EnumComp = Ident3;
        PtrGlb->IntComp = 40;
        strcpy(PtrGlb->StringComp, "DHRYSTONE PROGRAM, SOME STRING");
#ifndef GOOF
        strcpy(String1Loc, "DHRYSTONE PROGRAM, 1'ST STRING");
                                                                        /*GOOF*/
#endif
        Array2Glob[8][7] = 10;  /* Was missing in published program */

/******************
-- Start Timer --
******************/
#ifdef TIME
        starttime = time( (long *) 0);
#endif
#ifdef TIMES
        times(&tms); starttime = tms.tms_utime;
#endif
        for (i = 0; i < LOOPS; ++i)
        {

                Proc5();
                Proc4();
                IntLoc1 = 2;
                IntLoc2 = 3;
                strcpy(String2Loc, "DHRYSTONE PROGRAM, 2'ND STRING");
                EnumLoc = Ident2;
                BoolGlob = ! Func2(String1Loc, String2Loc);
                while (IntLoc1 < IntLoc2)
                {
                        IntLoc3 = 5 * IntLoc1 - IntLoc2;
                        Proc7(IntLoc1, IntLoc2, &IntLoc3);
                        ++IntLoc1;
                }
                Proc8(Array1Glob, Array2Glob, IntLoc1, IntLoc3);
                Proc1(PtrGlb);
                for (CharIndex = 'A'; CharIndex <= Char2Glob; ++CharIndex)
                        if (EnumLoc == Func1(CharIndex, 'C'))
                                Proc6(Ident1, &EnumLoc);
                IntLoc3 = IntLoc2 * IntLoc1;
                IntLoc2 = IntLoc3 / IntLoc1;
                IntLoc2 = 7 * (IntLoc3 - IntLoc2) - IntLoc1;
                Proc2(&IntLoc1);
        }

/******************
-- Stop Timer --
******************/

#ifdef TIME
        benchtime = time( (long *) 0) - starttime - nulltime;
        printf("Dhrystone(%s) time for %ld passes = %ld\n",
                Version,
                (long) LOOPS, benchtime);
```

```
        printf("This machine benchmarks at %ld dhrystones/second\n",
                ((long) LOOPS) / benchtime);
#endif
#ifdef TIMES
        times(&tms);
        benchtime = tms.tms_utime - starttime - nulltime;
        printf("Dhrystone(%s) time for %ld passes = %ld\n",
                Version,
                (long) LOOPS, benchtime/HZ);
        printf("This machine benchmarks at %ld dhrystones/second\n",
                ((long) LOOPS) * HZ / benchtime);
#endif

}

Proc1(PtrParIn)
REG RecordPtr   PtrParIn;
{
#define NextRecord      (*(PtrParIn->PtrComp))

        structassign(NextRecord, *PtrGlb);
        PtrParIn->IntComp = 5;
        NextRecord.IntComp = PtrParIn->IntComp;
        NextRecord.PtrComp = PtrParIn->PtrComp;
        Proc3(NextRecord.PtrComp);
        if (NextRecord.Discr == Ident1)
        {
                NextRecord.IntComp = 6;
                Proc6(PtrParIn->EnumComp, &NextRecord.EnumComp);
                NextRecord.PtrComp = PtrGlb->PtrComp;
                Proc7(NextRecord.IntComp, 10, &NextRecord.IntComp);
        }
        else
                structassign(*PtrParIn, NextRecord);

#undef  NextRecord
}

Proc2(IntParIO)
OneToFifty      *IntParIO;
{
        REG OneToFifty          IntLoc;
        REG Enumeration         EnumLoc;

        IntLoc = *IntParIO + 10;
        for(;;)
        {
                if (Char1Glob == 'A')
                {
                        --IntLoc;
                        *IntParIO = IntLoc - IntGlob;
                        EnumLoc = Ident1;
                }
                if (EnumLoc == Ident1)
                        break;
        }
}
```

```
Proc3(PtrParOut)
RecordPtr         *PtrParOut;
{
        if (PtrGlb != NULL)
                *PtrParOut = PtrGlb->PtrComp;
        else
                IntGlob = 100;
        Proc7(10, IntGlob, &PtrGlb->IntComp);
}

Proc4()
{
        REG boolean      BoolLoc;

        BoolLoc = Char1Glob == 'A';
        BoolLoc |= BoolGlob;
        Char2Glob = 'B';
}

Proc5()
{
        Char1Glob = 'A';
        BoolGlob = FALSE;
}

extern boolean Func3();

Proc6(EnumParIn, EnumParOut)
REG Enumeration EnumParIn;
REG Enumeration *EnumParOut;
{
        *EnumParOut = EnumParIn;
        if (! Func3(EnumParIn) )
                *EnumParOut = Ident4;
        switch (EnumParIn)
        {
        case Ident1:    *EnumParOut = Ident1; break;
        case Ident2:    if (IntGlob > 100) *EnumParOut = Ident1;
                        else *EnumParOut = Ident4;
                        break;
        case Ident3:    *EnumParOut = Ident2; break;
        case Ident4:    break;
        case Ident5:    *EnumParOut = Ident3;
        }
}

Proc7(IntParI1, IntParI2, IntParOut)
OneToFifty        IntParI1;
OneToFifty        IntParI2;
OneToFifty        *IntParOut;
{
        REG OneToFifty  IntLoc;

        IntLoc = IntParI1 + 2;
        *IntParOut = IntParI2 + IntLoc;
}

Proc8(Array1Par, Array2Par, IntParI1, IntParI2)
```

```
Array1Dim       Array1Par;
Array2Dim       Array2Par;
OneToFifty      IntParI1;
OneToFifty      IntParI2;
{
        REG OneToFifty  IntLoc;
        REG OneToFifty  IntIndex;

        IntLoc = IntParI1 + 5;
        Array1Par[IntLoc] = IntParI2;
        Array1Par[IntLoc+1] = Array1Par[IntLoc];
        Array1Par[IntLoc+30] = IntLoc;
        for (IntIndex = IntLoc; IntIndex <= (IntLoc+1); ++IntIndex)
                Array2Par[IntLoc][IntIndex] = IntLoc;
        ++Array2Par[IntLoc][IntLoc-1];
        Array2Par[IntLoc+20][IntLoc] = Array1Par[IntLoc];
        IntGlob = 5;
}

Enumeration Func1(CharPar1, CharPar2)
CapitalLetter   CharPar1;
CapitalLetter   CharPar2;
{
        REG CapitalLetter       CharLoc1;
        REG CapitalLetter       CharLoc2;

        CharLoc1 = CharPar1;
        CharLoc2 = CharLoc1;
        if (CharLoc2 != CharPar2)
                return (Ident1);
        else
                return (Ident2);
}

boolean Func2(StrParI1, StrParI2)
String30        StrParI1;
String30        StrParI2;
{
        REG OneToThirty         IntLoc;
        REG CapitalLetter       CharLoc;

        IntLoc = 1;
        while (IntLoc <= 1)
                if (Func1(StrParI1[IntLoc], StrParI2[IntLoc+1]) == Ident1)
                {
                        CharLoc = 'A';
                        ++IntLoc;
                }
        if (CharLoc >= 'W' && CharLoc <= 'Z')
                IntLoc = 7;
        if (CharLoc == 'X')
                return(TRUE);
        else
```

```
	{
		if (strcmp(StrParI1, StrParI2) > 0)
		{
			IntLoc += 7;
			return (TRUE);
		}
		else
			return (FALSE);
	}
}

boolean Func3(EnumParIn)
REG Enumeration EnumParIn;
{
	REG Enumeration EnumLoc;

	EnumLoc = EnumParIn;
	if (EnumLoc == Ident3) return (TRUE);
	return (FALSE);
}

#ifdef  NOSTRUCTASSIGN
memcpy(d, s, l)
register char	*d;
register char	*s;
register int	l;
{
	while (l--) *d++ = *s++;
}
#endif
```

APPENDIX B: Whetstone Benchmark Listing

```
/*  Whetstone benchmark. This program has a long history and is well
     described in "A Synthetic Benchmark" by H.J. Curnow and B.A. Wichman
     in Computer Journal, Vol 19 #1, February 1976.

     Time the compiles with and without the optimizer as follows:

     time cc -o whetstone whetstone.c -lm
     time cc -O -o whetstone.opt whetstone.c -lm

     Then time the runs of both versions as follows:

     time whetstone
     time whetstone.opt
*/

#define ITERATIONS  2
#define PNT5MINUS 0.499975
#define PNT5PLUS    0.50025
#define TWO      2.0

extern double sin();
extern double cos();
extern double atan();
extern double log();
extern double sqrt();
extern double exp();

main()
{
static int  mod1freq, mod2freq, mod3freq, mod4freq, mod6freq;
static int  mod7freq, mod8freq, mod9freq, mod10freq, mod11freq;
static float          ary[4];
static float          real1, real2, real3, real4;
register int          cntr;
register int          int1, int2, int3;

/* Establish execution frequencies */

mod1freq = 0 * ITERATIONS;
mod2freq = 12 * ITERATIONS;
mod3freq = 14 * ITERATIONS;
mod4freq = 345 * ITERATIONS;
mod6freq = 210 * ITERATIONS;
mod7freq = 32 * ITERATIONS;
mod8freq = 899 * ITERATIONS;
mod9freq = 616 * ITERATIONS;
mod10freq = 0 * ITERATIONS;
mod11freq = 93 * ITERATIONS;

/* MODULE 1: simple identifiers */

real1 =  1.0;
real2 = -1.0;
real3 = -1.0;
real4 = -1.0;
for(cntr = 1; cntr <= mod1freq; cntr += 1)
     {
     real1 = (real1 + real2 + real3 - real4) * PNT5MINUS;
```

```
    real2 = (real1 + real2 - real3 - real4) * PNT5MINUS;
    real3 = (real1 - real2 + real3 + real4) * PNT5MINUS;
    real4 = (real2 - real1 + real3 + real4) * PNT5MINUS;
    }       /* for */

/* MODULE 2: array elements */

ary[0] =  1.0;
ary[1] = -1.0;
ary[2] = -1.0;
ary[3] = -1.0;
for(cntr = 1; cntr <= mod2freq; cntr +=1)
    {
    ary[0] = (ary[0] + ary[1] + ary[2] - ary[3]) * PNT5MINUS;
    ary[1] = (ary[0] + ary[1] - ary[2] + ary[3]) * PNT5MINUS;
    ary[2] = (ary[0] - ary[1] + ary[2] + ary[3]) * PNT5MINUS;
    ary[3] = (ary[1] - ary[0] + ary[2] + ary[3]) * PNT5MINUS;
    }       /* for */

/* MODULE 3: array as parameter (see program at end) */

for(cntr = 1; cntr <= mod3freq; cntr += 1)
    mod3(ary);

/* MODULE 4: conditional jumps */

int1 = 1;
for (cntr = 1; cntr <= mod4freq; cntr += 1)
    {
    if (int1 == 1)
        int1 = 2;
    else
        int1 = 3;

    if (int1 > 2)
        int1 = 0;
    else
        int1 = 1;

    if (int1 < 1)
        int1 = 1;
    else
        int1 = 0;
    }       /* for */

/* MODULE 6: integer arithmetic using arrays */

int1 = 1;
int2 = 2;
int3 = 3;
for(cntr = 1; cntr <= mod6freq; cntr += 1)
    {
    int1 = int1 * (int2 - int1) * (int3 -int2);
    int2 = int3 * int2 - (int3 - int1) * int2;
    int3 = (int3 - int2) * (int2 + int1);

    ary[int3 - 1] = int1 + int2 + int3;
    ary[int2 - 1] = int1 * int2 * int3;
```

```
    }

/* MODULE 7: trigonometric functions */

real1 = 0.5;
real2 = 0.5;
for(cntr = 1; cntr <= mod7freq; cntr += 1)
    {
    real1 = atan(TWO * sin(real1) * cos(real1) / (cos(real1 + real2) +
        cos(real1 - real2) - 1.0)) * PNT5MINUS;
    real2 = atan(TWO * sin(real2) * cos(real2) / (cos(real1 + real2) +
        cos(real1 - real2) - 1.0)) * PNT5MINUS;
    }

/* MODULE 8: procedure calls */

real1 = real2 = real3 = 1.0;
for(cntr = 1; cntr <= mod8freq; cntr += 1)
    mod8(real1, real2, &real3);

/* MODULE 9; array references */

int1 = 1;
int2 = 2;
int3 = 3;
ary[1] = 1.0;
ary[2] = 2.0;
ary[3] = 3.0;
for(cntr = 1; cntr <= mod9freq; cntr += 1)
    {
    ary[int1] = ary[int2];
    ary[int2] = ary[int3];
    ary[int3] = ary[int1];
    }

/* MODULE 10: integer arithmetic */

int1 = 2;
int2 = 3;
for(cntr = 1; cntr <= mod10freq; cntr +=1)
    {
    int1 = int1 + int2;
    int2 = int1 + int2;
    int1 = int2 - int1;
    int2 = int2 - int1 - int1;
    }

/* MODULE 11: standard functions */

real1 = 0.75;
for(cntr = 1; cntr <= mod11freq; cntr +=1)
    real1 = sqrt( exp( log(real1) / PNT5PLUS));

/* end of main program */

}
```

```
/* Module 3 routine */

mod3(a)
float a[4];

{
int cntr;
for(cntr = 0; cntr <= 6; cntr += 1)
     {
     a[1] = (a[1] + a[2] + a[3] - a[4]) * PNT5MINUS;
     a[2] = (a[1] + a[2] - a[3] + a[4]) * PNT5MINUS;
     a[3] = (a[1] - a[2] + a[3] + a[4]) * PNT5MINUS;
     a[4] = (-a[1] + a[2] + a[3] + a[4]) / TWO;
     }
}

/* Module 8 routine */

mod8(r1, r2, r3)
float r1, r2, *r3;
{
     float tmp1, tmp2;

     tmp1=r1;
     tmp2=r2;

     tmp1 = PNT5MINUS * (tmp1 + tmp2);
     tmp2 = PNT5MINUS * (tmp1 + tmp2);
     *r3 = (tmp1 + tmp2) / TWO;
}
```