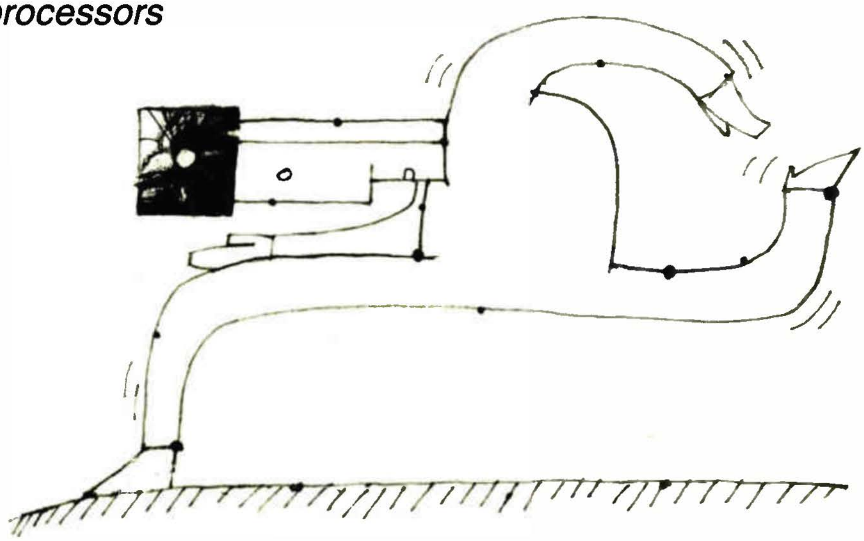
by the BYTE Editorial staff

High-Tech Horsepower

Benchmarking the computational speed and power of the Intel 80386 and Motorola 68020 microprocessors



When you're comparing the basic performance of automobiles, it's convenient to look at two measurements: How long does the car take to go from 0 to 60 mph, and what is its top speed? Of course, these measures of performance won't tell you much about the overall quality of a car, but they will let you place the vehicle in a general class: slow, fast, or faster.

In a similar way, computational benchmarks such as Dhrystones and Sieves—while not telling the whole story by themselves—can give you a common ground to compare different microprocessors and system architectures.

That's why, from time to time and under the collective heading of The New Generation, we'll present articles that analyze the computational performance of the major 32-bit processors (the Intel 80386 and the Motorola 68020) and the machines that use them. By measuring the *actual* (as opposed to the *projected*) performance of these systems, our tests will shed light on the conflicting claims made by the chip vendors as well as help you decide which machines to buy.

In addition to the 32-bit-specific articles in the Features section, we're adding 32-bit coverage to the Best of BIX, starting this month. It will include the best discussions from the BYTE Information Exchange as developers and users of 80386 and 68020 hardware and software share insights on current and future products, trends, and standards.

We'll also have major coverage of other items of importance in the world of 32-bit microcomputing. Next month, for example, we plan to present the first independent, in-depth electrical analysis of the Micro Channel bus found in IBM's PS/2 Models 50, 60, and 80. The analysis is being performed by Steve Ciarcia and his Circuit Cellar staff.

This month, we begin our coverage with a look at the performance of the Compaq Deskpro 386 and a Macintosh SE equipped with a 68020 add-in board.

A Can of Worms

It's important to note that our benchmarks test whole systems: complete, functioning machines using standard compilers. As in everyday applications, secondary factors such as disk I/O, memory-access times, bus architectures, and compiler differences influence our benchmarks.

We used the same source code to produce the executable code for both the 68020 and 80386 benchmarks. While it's possible to tweak a benchmark to improve its performance on a given processor (for example, using more register variables on a processor with a large register set), we've made no attempt to squeeze maximum performance from each processor. Instead, we've measured their relative performance using as uniform a yardstick as possible.

Finally, you'll notice that we've made no attempt to quantify the performance of the processors in terms of millions of instructions per second (MIPS). We don't think a comparison between Intel 80386 instructions and Motorola 68020 instructions would be valid. If both CPUs process 2 million instructions in 1 second but the first is 75 percent of the way through a problem while the second is only 40 percent finished with its code, then their performance is not equal.

The Benchmarks

Listings 1 through 5 provide the source code for five of our six benchmarks. Because of its length, we're not printing the Dhrystone benchmark listing. (All the listings, including Dhrystone, are available on BIX, BYTEnet, on disk, and in print. See the insert card following page 224 for details.) We chose these six benchmarks for two reasons: They are well known and widely used, providing a historical frame of reference, and they test a variety of computational functions.

The Fibonacci test computes the first 24 numbers in the Fibonacci sequence (1,

continued

Illustration by Maciek Albrecht.

Clock rates do not tell the whole story on computational throughput, however.

2, 3, 5, 8, 13, . . .) and repeats the process for 100 iterations. The Float benchmark performs 14 double-precision multiplications and divisions (7 of each) and repeats the process 10,000 times. The Sieve finds 1899 primes using the Sieve of Eratosthenes algorithm. The Sort test performs the Quicksort algorithm 100 times on an array of 1000 long integers. Savage is a floating-point test using a nested sequence of trigonometric and transcendental functions in a loop of 25,000 iterations. Dhrystone is a general-purpose benchmark testing processor speed except floating-point operations.

Tables 1 and 2 summarize the findings of our benchmarks. The Compaq Deskpro 386 and the Macintosh SE (equipped with a HyperCharger 68020) were both driven at 16 MHz and their accompanying FPUs were driven at 8 MHz. Therefore, direct comparisons of the results seem justified. However, as we'll discuss later, direct comparisons can be misleading, so the results must be interpreted with caution.

Benchmarking the 386

To test the computational abilities of the 80386, we used a Deskpro 386 with an 80287 numeric coprocessor, 1 megabyte of RAM, a 1.2-megabyte floppy disk drive, a 360K-byte floppy disk drive, a 40-megabyte hard disk, and a 40-megabyte tape backup. The system had a Compaq enhanced color graphics board and a Compaq color monitor.

Clock rates do not tell the whole story on computation throughput, however. The processor must also have quick access to memory. According to the Deskpro 386 *Technical Reference Guide*, the system operates with less than one wait state for memory accesses. The figure is imprecise because it depends on the type of memory access involved. Paged accesses require zero wait states, and non-paged accesses, two. (A paged access is one in which the row address of the memory device remains constant and only the column address changes. On the Deskpro 386, the page size is 2048 bytes.) On the average, 60 percent of the memory accesses occur in page mode, hence the figure of "less than one wait state."

All the benchmark tests were compiled to native 80386 code using MetaWare's High C version 1.3. The object programs were then linked using Phar Lap Software's 386/Link version 1.1 and run with its RUN386 version 1.1. RUN386 is a utility that allows applications to run in

continued

Listing 1: *Fibonacci test.*

```
#define NTIMES 100 /* number of times to compute Fibonacci value */
#define NUMBER 24  /* biggest one we can compute with 16 bits */

main()                      /* compute Fibonacci value */
   {
   int i;
   unsigned value, fib();

   printf("%d iterations:  ", NTIMES);

   for (i = 1; i <= NTIMES; i++)
      value = fib(NUMBER);

   printf("Fibonacci(%d) = %u.\n", NUMBER, value);
   exit(0);
   }


unsigned fib(x)            /* compute Fibonacci number recursively */
int x;
   {
   if (x > 2)
      return (fib(x - 1) + fib(x - 2));
   else
      return (1);
   }
```

Listing 2: *Float test.*

```
/* simple benchmark for testing floating-point speed
  of C libraries; does repeated multiplications and
  divisions in a loop that is large enough to make the
  looping time insignificant */

#define CONST1  3.141597E0
#define CONST2  1.7839032E4
#define COUNT 10000

main()
   {
   double a, b, c;
   int i;

   a = CONST1;
   b = CONST2;
   for (i = 0; i < COUNT; ++i)
      {
      c = a * b;
      c = c / a;
      c = a * b;
      c = c / a;
      c = a * b;
      c = c / a;
      c = a * b;
      c = c / a;
      c = a * b;
      c = c / a;
      c = a * b;
      c = c / a;
      c = a * b;
      c = c / a;
      }
   printf ("Done \n");
   }
```

Listing 3: *Sieve of Eratosthenes.*

```
/*
Eratosthenes Sieve Prime-Number Program from BYTE
 January 1983
*/

#define TRUE          1
#define FALSE         0
#define size       8190

   char flags [size + 1];
main()
   {
   int i, prime, k, count, iter;
   printf ("100 iterations \n");
   for (iter = 1; iter <= 100; iter++)  /* do program
                                          100 times */
   {

   count = 0;                    /* prime counter */
   for (i = 0; i <= size; i++)   /* set all flags true */
      flags [i] = TRUE;
   for (i = 0; i <= size; i++)
   {
      if (flags [i])             /* found a prime */
      {
      prime = i + i + 3;         /* twice index + 3 */
/*    printf ("\n%d", prime); */
```

```
      for (k = i + prime; k <= size;
       k+= prime)

         flags [k] = FALSE;      /* kill all multiple */
      count++;                  /* primes found */
      }
    }
  }
  printf ("%d primes. \n", count); /* primes found on
                                          100th pass */
}
```

Listing 4: *Quicksort test.*

```
  /*
  sorting benchmark - calls randomly the number of
  times specified by MAXNUM to create an array of long
  integers, then does a quicksort on the array of
  longs. The program does this for the number of times
  specified by COUNT.
*/

#define MAXNUM 1000
#define COUNT 100
#define MODULUS ((long) 0x20000)
#define C 13849L
#define A 25173L
long seed = 7L;
long random();
long buffer [MAXNUM] = {0};

main()
   {
   int i, j;
   long temp;
 /*
#include "startup.c"
*/
   printf ("Filling array and sorting %d times \n",
COUNT);
   for (i = 0; i < COUNT; ++i) {
      for (j = 0; j < MAXNUM; ++j) {
         temp = random (MODULUS);
         if (temp < 0L)
            temp = (-temp);
         buffer[j] = temp;
      }
      printf ("Buffer full, iteration %d \n", i);
      quick (0, MAXNUM - 1, buffer);
   }
 /*
#include "done.c"
*/
   }

quick (lo, hi, base)
      int lo, hi;
      long base [];
      {
      int i, j;
      long pivot, temp;

      if (lo < hi)
    {
    for (i = lo, j = hi-1, pivot = base [hi]; i < j; )
```

continued

```
            {
            while (i < hi && base [i] <= pivot)
                ++i;
            while (j > lo && base [j] >= pivot)
                --j;
            if (i < j)
                {
                temp = base [i];
                base [i] = base [j];
                base [j] = temp;
                }
            }
        temp = base [i];
        base [i] = base [hi];
        base [hi] = temp;
        quick (lo, i - 1, base);
        quick (i + 1, hi, base);
        }
    }

long random (size)
    long size;
    {
    seed = seed * A + C;
    return (seed % size);
    }
```

Listing 5: *Savage benchmark.*

```
/*
** savage.c - floating-point speed and accuracy test. C version
** derived from BASIC version which appeared in Dr. Dobb's Journal,
** Sep. 1983, pp. 120-122.
*/

#define  ILOOP  25000
extern  double  tan(), atan(), exp(), log(), sqrt();

main()
{
int i;
double a;

    printf("start\n");
    a = 1.0;
    for (i = 1; i <= (ILOOP - 1); i++)
        a = tan(atan(exp(log(sqrt(a*a))))) + 1.0;
    printf("a = %20.14e\n", a);
    printf("done\n");
}
```

protected mode; and it fully exploits the 80386's 32-bit capabilities while still letting the application make most kinds of MS-DOS system calls.

Timings for every test except the Dhrystone were made with a stopwatch. The tests were run a number of times. To make the tests long enough for accurate timing, we increased the iterations by a factor of 10 beyond the count we normally use to benchmark 16-bit and 8-bit systems.

The floating-point-intensive tests in this group are the Savage and the Float benchmark.

Benchmarking the 68020

The Macintosh SE contained 1 megabyte of RAM, a 7.83-MHz 68000 processor, an 800K-byte 3½-inch floppy disk drive, and an internal 20-megabyte SCSI hard disk.

Another Macintosh SE was configured the same as above with a General Computer HyperCharger 68020 board with a 16-MHz 68020 processor, 1 megabyte of 32-bit one-wait-state memory, and a 12-MHz 68881 that was actually driven at 8 MHz. To force the 68020 to run out of its

continued

Table 1: *Benchmark results of 680x0 machines.*

Test	Number of iterations	Mac SE	Mac SE w/Hyper-Charger	Arete 1100
Fibonacci	100	264.00*	71.60	70.20
Float	10,000	229.98	4.16	2.90
Sieve	100	64.70*	14.94	12.80
Sort	100	111.30*	19.82	12.60
Savage	25,000	1884.30**	8.78	24.80
Dhrystones/sec	500,000	574.00***	2114.00	2702.00

* We ran 10 iterations and scaled the results accordingly.
** We ran 2500 iterations and scaled the results accordingly.
*** We ran 50,000 iterations and scaled the results accordingly.

Table 2: *Benchmark results for the Intel 80x86 machines.*

Test	Number of iterations	Deskpro 386 with FPU	IBM PC AT (8 MHz) without FPU	IBM PC AT (8 MHz) with FPU
Fibonacci	100	3.10	950.00	120.96
Float	10,000	5.41	116.36	9.70
Sieve	100	5.98	26.71	25.29
Sort	100	9.67	46.53	45.73
Savage	25,000	35.10	1103.00	38.28
Dhrystones/sec	500,000	3703.70	1567.90	1748.90

32-bit memory, we set the RAM cache on the Macintosh to 1024K bytes.

The Arete supermicro computer contained a 12.5-MHz 68020, 8K bytes of high-speed zero-wait-state cache, 68881 floating-point coprocessor, 2 megabytes of RAM, a 168-megabyte hard disk, and a cartridge tape drive.

Both sets of Macintosh benchmarks were compiled using the Consulair Mac C compiler. The 68000 benchmarks used version 5.01. The 68020 benchmarks used the initial release of the Consulair 68020/68881 Mac C compiler.

The Arete benchmarks used the standard Arete 68020 C compiler.

Analyzing the Results

Benchmarks tend to test compilers as much as they test processors. To get an idea of how much the compiler contributed to the benchmark times, we ran the Dhrystone benchmark on the Macintosh/HyperCharger compiled with an early beta version of the Macintosh Programmer's Workshop C compiler in addition to the Consulair compiler. The MPW C compiler yielded 2522 Dhrystones per second, a difference of 16.2 percent (see table 1). Since the MPW C compiler allocates register variables automatically, the time for register versus nonregister benchmarks was the same.

The discrepancy between the results of these benchmarks is perplexing and demonstrates the difficulty in testing whole machines. There are many variables to consider: the static RAM in the Deskpro 386, the 1024K-byte cache on the Arete, the differences among compilers, the differing levels of function in floating-point coprocessors, the different clock rates in the machines, 32-bit versus 16-bit buses, and many other factors.

Another factor that confuses the analysis of these benchmarks is the built-in support of functions on the floating-point coprocessors. The Motorola 68881 has built-in transcendentals and so can easily outperform the 80287 on the Savage test, for instance. Also, data exchanges between the CPU and FPU may account for execution-time discrepancies more than the raw performance of either part alone.

So what *do* these tests reveal? Just this: A Deskpro 386 runs these benchmarks faster than the HyperCharger-boosted Macintosh SE. ■