Testing Reveals x86 Core Differences

Design Tradeoffs Can Create Anomalies

by Brian Case

At the same clock rate, all the leading x86 cores—Pentium, Pentium Pro, Pentium II, and K6—have similar performance on most desktop applications. Application-level benchmarks like Winstone and synthetic benchmarks like WinBench typically do not reveal large differences among these cores.

But the broad scope of these benchmarks masks more significant differences underneath. Peeling the onion back one layer reveals that these cores have performance differences as large as 2× on specific CPU-intensive tasks. Even more surprising, the direction of the difference can be counterintuitive. We found, for example, cases where the venerable Pentium core takes fewer clocks than its more complex out-of-order cousin, Pentium II.

For those interested only in average performance on PC productivity applications, the traditional benchmark results provide reasonable guidelines. But for those looking to use a CPU for a specific task, and those interested in microarchitectural tradeoffs, it may be worthwhile to look deeper.

Tuning for One Application Domain Can Backfire

An effort to harness the computing power inherent in the servers and clients that form the Internet is being coordinated at *www.distributed.net*. The effort has discovered, through brute force, the secret keys to a couple of cryptography challenges. Available at the Web site are client key-checking programs that coordinate to check every possible key in the key space until the one correct key is found. The clients implement key-checking for the DES and RC5 encryption algorithms.

These client key-checkers are available for many types of processors, and the x86 clients allow the user to select core code that has been hand-tuned for the most popular x86 chips. While it is beyond the scope of this article to verify that a tuned version is optimal for a given x86 chip, a quick check indicates that the tuned versions do indeed yield good performance on the intended chip. Table 1 reports the results of running the respective hand-tuned versions.

Testing the performance of these key-checking clients on various processors gives surprising results. Table 1 shows that when compared at equal clock rates, the fastest DES key-checker is the Pentium/MMX. The complexity of the P6 core pays better dividends on the RC5 benchmark, running essentially twice as fast as the Pentium. Pentium Pro and Pentium II (Klamath, in this case) have essentially identical performance.

Tests with the L2 caches disabled and various tunings of the BIOS memory timing parameters produced identical results. This indicates the benchmark fits in the L1 caches, so the performance of these benchmarks depends only on the microprocessor core and L1 caches.

While it may be startling to see the P55C microarchitecture beating the dramatically more sophisticated P6 core, the real surprise is K6, which performs poorly on these tests.

Investigation indicates that the K6 performance problem on this test stems from a single design factor: the implementation of the rotate instructions. Since these encryption algorithms use a lot of shifting and rotating of long bit strings, a slow implementation of rotates can significantly reduce performance. The K6 implements the rotate-register instruction with a microcoded ROP sequence, while the P6 implements it with a single internal ROP.

Equalizing Clock Rate Can Be Misleading

The results shown in Table 1 are both revealing and fallacious. The test is revealing if the goal is to expose the relative efficiencies of different microarchitectures.

The fallacy arises because the P6 microarchitecture was designed with a very long pipeline to enable operation at higher clock rates, resulting in what appear to be inefficiencies relative to a shorter pipeline. Note that the same fabrication process that yields 233-MHz Pentiums produces 300-MHz Pentium II CPUs, and a A 300-MHz Pentium II will soundly trounce a 233-MHz Pentium/MMX.

Pentium Extensions Ease Performance Monitoring

With the Pentium processor, Intel introduced a host of performance monitoring and debugging capabilities. Intel added even more in the P6. Most measure specific events such as L1 cache misses or mispredicted branches.

While Intel's competitors have not copied all these capabilities, one very useful feature seems ubiquitous: the Time Stamp Counter (TSC). The TSC value advances at the processor core clock rate and is always running. The RDTSC

Microprocessor (all @ 233/66 MHz)	DES-II		RC5-64	
	Mkeys/s	% Max	Mkeys/s	% Max
Pentium II	1.09	91%	**0.656**	**100%**
Pentium Pro*	1.08	90%	**0.654**	**100%**
K6	0.82	68%	0.381	58%
Pentium/MMX	**1.20**	**100%**	0.334	51%

Table 1. The DES-II and RC5 decryption clients are CPU-bound programs that fit in the L1 caches; thus L2 cache, memory, I/O, and display performance are irrelevant. *Pentium Pro-200 overclocked to 233 MHz. (Tests run as "rc5des -c n -benchmark" using client version 2.7020.403 under Linux.) (Source: MDR)

instruction reads the value of the 64-bit TSC into EDX:EAX. By sampling the TSC with a pair of RDTSC instructions, a program can measure the elapsed number of cycles between two points in a program (see sidebar, next page).

Characterizing Rotate Performance

Table 2 shows the measurements obtained using RDTSC to count the execution times of several loops.

The P6 core executes the empty decrement/jump-not-zero loop slower than the other processors. With branch prediction and the capability to retire two or more instructions per cycle, the natural assumption is that a two-instruction loop will execute at the rate of one iteration per cycle, as shown by the K6 and Pentium/MMX, but the P6 can decode branches only in its first decoder position. In real programs, empty loops are not very useful, so the suboptimal performance of the P6 in this case is unimportant.

As the table shows, the P6 can overlap up to two independent ROL instructions with the loop overhead. Although not shown, a third ROL adds a single cycle because the P6 has only one shift execution unit. The inclusion of two independent increment instructions adds only a single cycle to the count, since the P6 has two execution units capable of integer addition; P6 performance is unaffected by the static scheduling of the instruction sequence (columns 4 and 5).

Each ROL adds a cycle to the count for Pentium/MMX, because ROL is not a pairable instruction. Even though Pentium/MMX has two integer execution units, the static scheduling of the sequence with two ROLs and two INCs matters, since the in-order microengine has no capability to reorder instructions.

Each ROL adds two cycles to the total for the K6, because ROL is a microcoded instruction. Not only does the microcoded sequence of ROPs take longer, but it also monopolizes the decoders while the ROP sequence is generated, as illustrated by the difference in the cycle counts in columns 4 and 5. In the first sequence, the first ROL stalls the decode, but the two INCs get decoded together and executed in parallel. In the second sequence, with the ROL and INC instructions interleaved, all four ROL and INC instructions are decoded separately. The ROLs and INCs are independent, but this decoder bottleneck starves the execution units.

The last two columns in Table 2 show the effect of adding one and two more independent INCs to the sequence from column 4. For the K6 and Pentium/MMX, the additional INCs in columns 6 and 7 do not add to the execution time, because they are paired and can use otherwise idle decode and execution resources.

The oddity in the cycle count for the P6 core on the code in column 6 indicates the sequence executes in three cycles for half the iterations and in four cycles on the other half. This is probably caused by instruction pairing across the loop boundaries.

The IPC (instructions per cycle) columns in Table 2 are a measure of how effectively the execution hardware is used. For these code sequences, only the K6 and Pentium/MMX ever achieve their theoretical maximum of two instructions decoded, executed, and retired per cycle. The P6 core has a theoretical maximum of three IPC, but this peak is not achieved for these sequences. With two integer execution units and one load/store unit, P6 can only achieve three IPC when loads or stores are present. The P6 does, however, exhibit the most consistent performance, which illustrates the power of a general out-of-order microarchitecture.

Winstone results indicate that the K6's poor IPC numbers on these microbenchmarks are not representative of the chip's performance on common PC applications; rather, they are a direct result of a design tradeoff that sacrifices the performance of rotate instructions. For the vast majority of

Test Loop Instruction Sequence	1 L1: DEC EAX JNZ L1		2 L1: ROL EBX,3 DEC EAX JNZ L1		3 L1: ROL EBX,3 ROL ECX,3 DEC EAX JNZ L1		4 L1: ROL EBX,3 INC EDI INC ESI ROL ECX,3 DEC EAX JNZ L1		5 L1: ROL EBX,3 INC EDI ROL ECX,3 INC ESI DEC EAX JNZ L1		6 L1: ROL EBX,3 INC EDI INC ESI ROL ECX,3 INC EDX DEC EAX JNZ L1		7 L1: ROL EBX,3 INC EDI INC ESI ROL ECX,3 INC EDX INC EBP DEC EAX JNZ L1	
Processor	**Cycles**	**IPC**	**Cycles**	**IPC**	**Cycles**	**IPC**	**Cycles**	**IPC**	**Cycles**	**IPC**	**Cycles**	**IPC**	**Cycles**	**IPC**
Pentium Pro	2	1	2	1.5	2	2	3	2	3	2	3.5	2	4	2
Pentium II	2	1	2	1.5	2	2	3	2	3	2	3.5	2	4	2
Pentium/MMX	1	2	2	1.5	3	1.33	4	1.5	5	1.2	5	1.4	5	1.6
K6	1	2	3	1	5	0.8	6	1	7	0.86	7	1	7	1.14

Table 2. The empty loop executes faster on the K6 and Pentium/MMX than on the P6-based processors. The increasing cycle counts for the K6 show how a microcoded instruction reduces efficiency by stalling decode. The fourth and fifth code sequences illustrate the advantage of out-of-order execution: the P6 can reorder the instruction sequence to achieve maximum performance, whereas the Pentium/MMX cannot. (Source: MDR)

Cycle Counting With RDTSC

The Time Stamp Counter and the RDTSC instruction were added to the x86 architecture with the introduction of the Pentium processor. The TSC is a 64-bit counter that is initialized to zero following hardware reset. It then increments by one with each internal processor cycle even when the processor is halted by the HLT instruction or the external STPCLK pin.

Architecturally, RDTSC is guaranteed to return a monotonically increasing unique value on each execution, except in the case when the counter wraps around, which will not occur for years after processor reset.

On an in-order processor, the RDTSC instruction could be used to obtain very accurate measurements of instruction sequences. The architectural definition of RDTSC, however, specifies that RDTSC is not serializing and can be executed out of order with respect to other instructions. Thus, a given RDTSC may be executed before preceding instructions or after following instructions. Theoretically, the uncertainty of the execution order of a RDTSC is determined by the size of the window an out-of-order processor uses to buffer waiting instructions. In practical situations, execution of a RDTSC is fairly predictable due to dependencies.

To filter out uncertainties in the execution order of a pair of RDTSC instructions and to stabilize cache behavior, instruction sequences investigated in this article were incorporated into the following framework:

```
            RDTSC
            PUSH    EAX
            MOV     EAX,1000

            .align  32
L1:         Instruction 1
            Instruction 2
            ...
            Instruction n
            DEC     EAX
            JNZ     L1

            RDTSC
            POP     EBX
            SUB     EAX,EBX
```

An instruction sequence to be measured is inserted in a loop that executes for 1,000 iterations. The RDTSC instructions precede and follow the loop. This strategy has the property that each cycle attributable to an instruction in the loop will cause an increment of 1,000 in the cycle count; any transient cycles—due to the uncertainties in the execution scheduling of the RDTSCs and the overhead of saving the TSC count (PUSH EAX) and setting up the loop count—will appear in the low three digits of the measured cycle count. No uncertainty or transient will amount to more than a few tens of cycles at most.

The execution time for the empty loop (consisting of only DEC/JNZ) is one or two cycles on all processors tested, as Table 2 shows, so loop overhead can be factored out of the final cycle count.

popular PC applications, this tradeoff causes little measurable performance degradation.

Different Caches Perform Differently

Especially for the x86, a fast instruction-execution engine is of little use without a fast data cache to serve the high density of load and store operations in x86 programs. To test the performance of the three levels of a PC memory hierarchy, we designed a tiny loop and a linked-list data structure to allow an arbitrarily long sequence of memory references with controlled cachability.

Figure 1 shows the code sequence, which consists of a load-indirect MOV instruction surrounded by our familiar DEC/JNZ loop. The MOV loads a value from a base register directly into that same base register, so the new value will be used as the memory address on the next iteration. This causes the MOV to be dependent on itself, which ensures that the cycle count for the loop will include the entire execution time of the MOV. It can at most be overlapped with the DEC/JNZ, which has known execution time.

Setting the values of a sequence of memory locations to form a ring, causes this simple code sequence to follow the chain of pointers around in a circle until the count in EAX is exhausted. With the correct number of nodes and addresses of the nodes in the linked list, a cache of a given size and associativity can be overwhelmed, thus causing it to miss on each execution of the MOV instruction.

Table 3 shows the results of using this technique to determine the speed of x86 caches. All processors were running at 233 MHz, and the bus speed of the K6 and Pentium

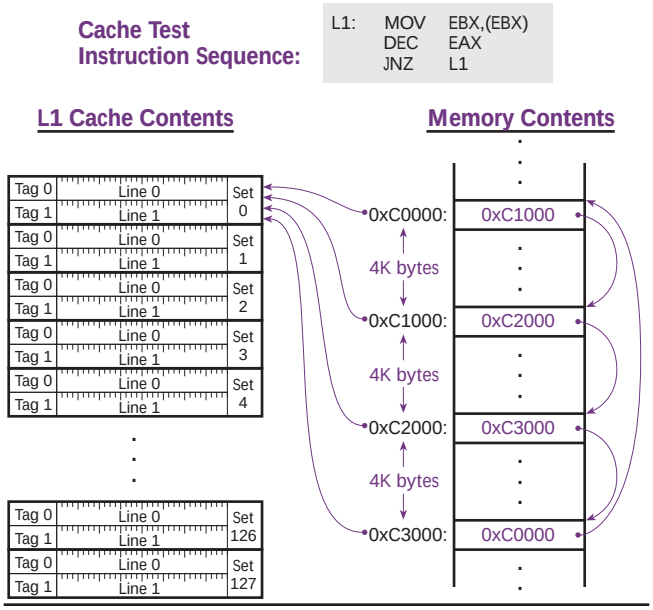

Figure 1. The linked-list data structure is set up such that when the cache test loop runs, the number of elements in the list will exceed the associativity of the cache in which misses are desired. Also, the addresses of the elements in the linked list are chosen so that they map into the same cache set.

MMX was 66 MHz. The cycle counts in the table are for an actual instruction sequence and reflect the true cost of executing worst-case code that delivers an operand to a waiting processor register, not just an isolated parameter such as SRAM access time. In real applications, similar pathological access patterns do arise, but they would not dominate the run time of most nontrivial programs. Database programs, however, may be a significant exception.

The long latencies shown for L2 misses result in a full cache line being loaded, and so, in real programs, the cycle counts in Table 3 are typically amortized over at least a series of sequential accesses to the same cache line.

There are several observations of interest. First, note that the K6 and Pentium/MMX have the fastest L1 caches for 32-bit aligned loads. Any misalignment adds only a single cycle to the execution time for the K6, but Pentium/MMX pays a three-cycle penalty.

The execution time for various alignments highlights one of the improvements Intel made in Pentium II compared with Pentium Pro. Pentium II tolerates misalignment on all accesses except those that cross a 32-byte cache line, but Pentium Pro takes a five-cycle hit when crossing an 8- or 16-byte boundary within a cache line. This Pentium II refinement was probably made possible because its L1 caches are twice the size of Pentium Pro's: the extra 8K of cache can be accessed in parallel, and a multiplexer/shifter can then select the proper group of four bytes. Both processors pay a nine-cycle penalty for accesses that cross a cache line boundary.

The penalty for an L1-cache miss varies greatly, depending on how the L2 is implemented and connected to the processor. The best by far is Pentium Pro, which can deliver a 32-bit operand with less than half the latency of the Pentium II in most cases. Both Pentium II and Pentium Pro processor chips have a dedicated L2 cache bus, which provides fast access to the L2, but Pentium II uses commodity L2 SRAMs, which run at half the speed of the custom Pentium Pro L2 SRAMs. Because its L2 cache is so fast, L1 cache hits for some alignments in Pentium Pro are slower than L1 misses for other alignments.

Once again, examining these processors at equal clock rates may lead to a distortion. The advantage of the Pentium Pro L2 over the Pentium II L2 is probably slightly less than shown in Table 3, because the initial latency of the Pentium Pro custom cache chips is tuned for 200-MHz operation, while the initial latency of the Pentium II cache chips is tuned for 266-MHz (or even 300-MHz) operation. Thus, while the forthcoming Pentium II Xeon CPUs will have a much faster L2 cache than that of the Pentium II, the advantage may be slightly less than Table 3 indicates. For example, an aligned L1 miss/L2 hit may take eight or nine cycles instead of Pentium Pro's seven.

Beware of Benchmarks

Benchmarks like Winstone and Winbench are reasonable indicators of the relative performance of different CPUs and how well they will perform overall. But such benchmarks are designed to test all aspects of PC performance, which makes them poor indicators of how well a processor will perform on a specific task.

For example, as we showed in this article, the K6 may not be the best choice for cryptography applications that depend heavily on rotate operations. On the other hand, the K6 could perform significantly better than the others on applications where misaligned data is prevalent, such as applications that access legacy data structures in network transmission packets.

The results shown here compare cores at the same clock rate. In practice, however, these cores do not all run at the same maximum clock rate. So while Pentium may have better per-clock performance than Pentium II in some cases, Pentium II's deeper pipeline allows it to run significantly faster. This, in fact, may be the more significant contributor to Pentium II's bottom-line performance advantage than its out-of-order microarchitecture.

If you have a specific job in mind for a processor, you may not want to rely on traditional benchmark results to make your decision. By performing a clock-accurate analysis of the processor running code that is representative of your application, you may be able to select a processor that gives you better performance or saves you money. M

Interpretation	Address Alignment (bytes)	K6		Pentium II		Pentium Pro		P55C
		L1 Hit	L1 Miss L2 Hit	L1 Hit	L1 Miss L2 Hit	L1 Hit	L1 Miss L2 Hit	L1 Hit
Naturally aligned on four-byte boundary	0, 4, 8, 12, 16, 20, 24, 28	2	28	3	16	3	7	2
Misaligned, but does not cross 8-byte or 16-byte boundary	1, 2, 3 9, 10, 11 17, 18, 19 25, 26, 27	3	28	3	16	3	7	5
Crosses 8-byte but not 16-byte boundary	5, 6, 7 21, 22, 23	3	28	3	24	8	12	5
Crosses 16-byte boundary	13, 14, 15	3	28	3	28	8	12	5
Crosses 32-byte cache-line boundary	29, 30, 31	3	56	12	54	12	34	5

Table 3. Cycle counts for the 32-bit memory load instruction in the loop shown in Figure 1. All processors are running at 233 MHz; the K6 used a 66-MHz external bus. In some cases, cache performance varies greatly depending on the alignment of the load address. The half-speed cache of the Pentium II more than doubles the L1 miss penalty compared with Pentium Pro. Pentium II Xeon is likely to enjoy a similar advantage over Pentium II. The P55C (Pentium/MMX) results for L1 miss/L2 hit were not reproducible and so are not reported. (Source: MDR)