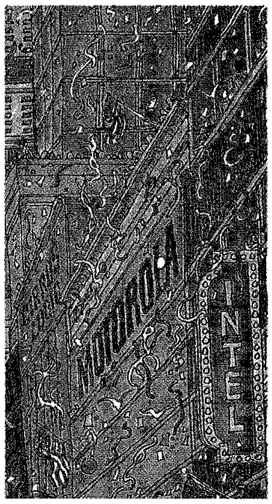
Tuning the Pentium Pro Microarchitecture

David B. Papworth

Intel Corporation



This inside look at a large microprocessor development project reveals some of the reasoning (for goals, changes, trade-offs, and performance simulation) that lay behind its final form.

Designing a wholly new microprocessor is difficult and expensive. To justify this effort, a major new microarchitecture must improve performance one and a half or two times over the previous-generation microarchitecture, when evaluated on equivalent process technology. In addition, semiconductor process technology continues to evolve while the processor design is in progress. The previous-generation microarchitecture increases in clock speed and performance due to compactions and conversion to newer technology. A new microarchitecture must "intercept" the process technology to achieve a compounding of process and microarchitectural speedups.

The process technology, degree of pipelining, and amount of effort a team is willing to spend on circuit and layout issues determine the clock speed of a microarchitecture. Typically, a microarchitecture will start with the same clock speed as a prior microarchitecture (adjusted for process technology scaling). This enables the maximum reuse of past designs and circuits, and fits the new design to the existing product development tools and methodology. Performance enhancements should come primarily from the microarchitecture and not from clock speed enhancements per se.

Often, a new processor's die area is close to the maximum that can be manufactured. This design choice stems from marketplace competitiveness and efforts to get as much performance as possible in the new microarchitecture. While making the die smaller and cheaper and improving performance are desirable, it is generally not possible to achieve a 1.5-to-2-times-better performance goal without using at least 1.5 to 2 times a prior design's transistors.

Finally, new processor designs often incorporate new features. As the performance of the core logic improves, designs must continue to enhance the bus and cache architecture to keep pace with the core. Further, as other technologies (such as multiprocessing) mature, there is a natural tendency to draw them into the processor design as a way of providing additional features and value for the end user.

Mass-market designs

The large installed base and broad range of applications for the Intel architecture place additional constraints on the design, constraints beyond the purely academic ones of performance and clock frequency. We do not have the flexibility to control software applications, compilers, or operating systems in the same way system vendors can. We cannot remove obsolete features and must cater to a wide variety of coding styles. The processor must run thousands of shrink-wrapped applications, rather than only those compiled by a vendor-provided compiler running on a vendor-provided operating system on a vendor-provided platform. These limitations leave fewer avenues for workarounds and the processor exposed to a much greater variety of instruction sequences and boundary conditions.

Intel's architecture has accumulated a great deal of history and features in 15 years. The product must deliver world-class performance and also successfully identify and resolve compatibility issues. The microprocessor may be an assemblage of pieces from many different vendors, yet must function reliably and be easy for the general public to use.

Since a new design needs to be manufacturable in high volume from the very beginning, designers cannot allow the design to expand to the limits of the technology. It also must meet stringent environmental and

0272-1732/96/$5.00 © 1996 IEEE

design-life limits. It must deliver high performance using a set of motherboard components costing less than a few hundred dollars.

Meeting these additional design constraints is critical for business success. They add to the complexity of the project and the total effort required, compared to a brand-new instruction set architecture. The additional complexity results in extra staffing and longer schedules. A vital ingredient in long-term success is proper planning and management of the demands of extra complexity; management must ensure that the complexity does not impact the product's long-term performance and availability.

Our first effort

After due consideration of the performance, area, and mass-market constraints, we knew we would have to implement out-of-order execution and register renaming to wring more instruction level parallelism out of existing code. Further, the modest register file of the Intel architecture constrains any compiler. That is, it limits the amount of instruction reordering that a compiler can do to increase superscalar parallelism and basic block size. Clearly, a new microarchitecture would have to provide some way to escape the constraints of false dependencies and provide a form of dynamic code motion in hardware.

To conform to projected Pentium processor goals, we initially targeted a 100-MHz clock speed using 0.6-micron technology. Such a clock speed would have resulted in roughly a 10-stage pipeline. It would have much the same structure as the Pentium processor with an extra decode stage added for more instruction decode bandwidth. It would also require extra stages to implement register renaming, runtime scheduling, and in-order retirement functions.

We expected a two-clock data cache access time (like the Pentium processor) and other core execution units that would strongly resemble the Pentium processor. The straw-man microarchitecture would have had the following components:

- a 100-MHz clock using 0.6-micron technology,
- a 10-stage pipeline,
- four-instruction decoding per clock cycle,
- four-micro-operation renaming and retiring per clock cycle,
- a 32-Kbyte level-1 instruction cache,
- a separate 32-Kbyte L1 data cache,
- two general load/store ports, and
- a total of 10 million transistors.

From the outset we planned to include a full-frequency, dedicated L2 cache, with some flavor of advanced packaging connecting the cache to the processor. Our intent was to enable effective shared-memory multiprocessing by removing the processor-to-L2 transactions from the traditional global interconnect, or front-side, bus, and to facilitate board and platform designs that could keep up with the high-speed processor. Remember that in 1990/1991 when we began the project, it was quite a struggle to build 50- and 66-MHz systems. It seemed prudent to provide for a package-level solution to this problem.

Terminology

This article uses "performance" to mean work done per unit time. We termed a time reduction of 1/3 (100 seconds to 67 seconds) to be a performance improvement of 50 percent. A clock-per-instruction degradation of 20 percent results in a program taking 20 percent longer at a constant clock rate, which amounts to a performance degradation of 20 percent over the baseline.

What we actually built

The actual Pentium Pro processor looks much different from our first straw man:

- a 150-MHz clock using 0.6-micron technology,
- a 14-stage pipeline,
- three-instruction decoding per clock cycle,
- three micro-operations (micro-ops) renamed and retired per clock cycle,
- an 8-Kbyte L1 instruction cache,
- an 8-Kbyte L1 data cache,
- one dedicated load port and one store port, and
- 5.5 million transistors.

The evolution process

Our first efforts centered on designing and simulating a high-performance dynamic-execution engine. We attacked the problems of renaming, scheduling, and dispatching, and designed core structures that implemented the desired functionality.

Circuit and layout studies overlapped this effort. We discovered that the basic out-of-order core and the functional units could run at a higher clock frequency than 100 MHz. In addition, instruction fetching and decoding in two pipeline stages and data cache access in two clock cycles were the main frequency limiters.

One of our first activities was to create a microarchitect's workbench. Basically, this was a performance simulator capable of modeling the general class of dynamic execution microarchitectures. We didn't base this simulator on silicon structures or detailed modeling of any particular implementation. Instead, it took an execution trace as input and applied various constraints to each instruction, modeling the functions of decoding, renaming, scheduling, dispatching, and retirement. It processed one micro-operation at a time, from decoding until retirement, and at each stage applied the limitations extant in the design being modeled.

This simulator was very flexible in allowing us to model any number of possible architectures. Modifying it was much faster than modifying a detailed, low-level implementation, since there was no need for functional correctness or routing signals from one block to another. We set up this simulator to model our initial straw-man microarchitecture and then performed a sensitivity analysis of the major microarchitectural areas that affect performance, clock speed, and die area.

We simulated each change or potential change against at least 2 billion instructions from more than 200 programs. We

Pentium Pro structure and features

The Pentium Pro processor is a 32-bit Intel architecture microprocessor. It implements a dynamic-execution microarchitecture, incorporating speculative and out-of-order execution. A micro-dataflow approach identifies instruction level parallelism and controls the dispatch of operations to a superscalar, superpipelined core.

The processor incorporates an 8-Kbyte instruction cache and an 8-Kbyte data cache. These nonblocking caches let multiple cache misses proceed in parallel; cache hits proceed during outstanding cache misses to other addresses.

We packaged the processor, along with a separate 256-Kbyte L2 cache, in a dual-cavity, ceramic pin grid array (Figure A). This pipelined and nonblocking cache connects to the processor via a 64-bit, full-frequency bus. The four-way set associative L2 cache employs 32-byte cache lines and contains 8 bits of error correcting code for each 64 bits of data.

The processor connects to input/output and memory via a 64 bit bus operating at up to 66 MHz. The bus implements shared-memory multiprocessing and supports high bandwidth via deep pipelining and split address and data cycles.

Figure B is a simplified block diagram. The instruction fetch unit (IFU) fetches 16 bytes every clock cycle and delivers them to the instruction decoder (ID). The ID buffers multiple 16-byte fetches, rotates these bytes to the starting point of the next instruction, and decodes up to three instructions per clock. The decoder converts instructions into dataflow nodes (called micro-ops), buffering them in a six-entry micro-op queue.

The micro-ops then pass through a register-renaming stage (the register alias table, or RAT) for remapping of the architectural registers to a set of 40 physical registers. This stage removes false dependencies caused by a limited number of architectural registers while preserving the true data dependencies (reads after writes).

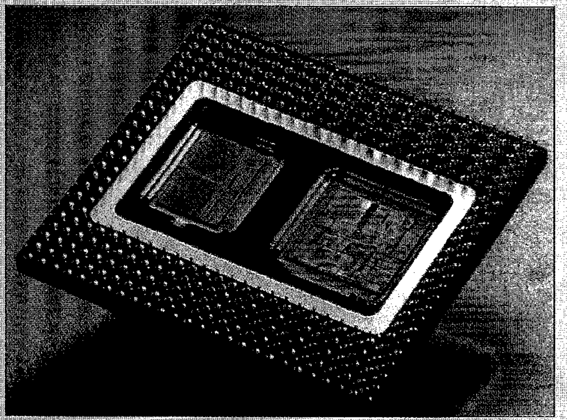
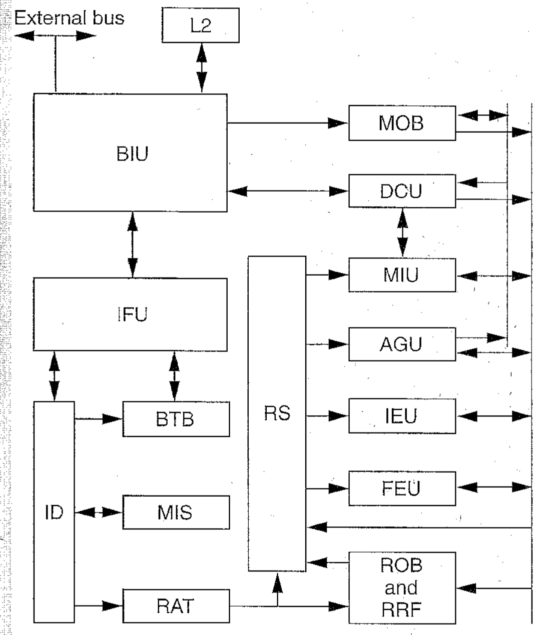


Figure A. Pentium Pro processor and L2 cache in a dual-cavity package.



AGU	Address generation unit
BIU	Bus interface unit
BTB	Branch target buffer
DCU	Data cache unit
FEU	Floating-point execution unit
ID	Instruction decoder
IEU	Integer execution unit
IFU	Instruction fetch unit (includes I-cache)
L2	Level-2 cache
MIS	Microinstruction sequencer
MIU	Memory interface unit
MOB	Memory reorder buffer
RAT	Register alias table
ROB	Reorder buffer
RRF	Retirement register file
RS	Reservation station

Figure B. Basic processor block diagram.

studied the effects of L1 cache size, pipeline depth, branch prediction effectiveness, renaming width, reservation station depth and organization, and reorder buffer depth. Quite often, we found that our initial intuition was wrong and that every assumption had to be tested and tuned to what was proven to work.

The trade-off

Based on our circuit studies, we explored what would happen if we boosted the core frequency by 1.5 times over our initial straw man. This required a few simple changes to the reservation station, but clearly we could build the basic core to operate at this frequency. It would allow us to retain

Pentium Pro (continued)

Following renaming, the operations wait in a 20-entry reservation station (RS) until all of their operands are data ready and a functional unit is available. As many as 5 micro-ops per clock can pass from the reservation station to the various execution units. These units perform the desired computation and send the result data back to data-dependent micro-ops in the reservation stations, as well as storing the result in the reorder buffer (ROB).

The reorder buffer stores the individual micro-ops in the original program order and retires as many as three per clock in the retirement register file (RRF). This file examines each completed micro-op for the presence of faults or branch mispredictions, and aborts further retirement upon detecting such a discontinuity. This reimposes the sequential fault model and the illusion of a microarchitecture that executes each instruction in strict sequential order.

The L1 data cache unit (DCU) acts as one of the execution units. It can accept a new load or store operation every clock and has a data latency of three clocks for loads. It contains an 8-Kbyte, two-way associative cache array plus fill buffers to buffer data and track the status of as many as four simultaneously outstanding data cache misses.

The bus interface unit (BIU) processes cache misses. This unit manages the L2 cache and its associated 64-bit, full-frequency bus, as well as the front-side system bus, which typically operates at a fraction of the core frequency, such as 66-MHz on a 200-MHz processor. Transactions on the front-side and dedicated buses are organized as 32-byte cache line transfers. The overall bus architecture permits multiple Pentium Pro processors to be interconnected on the front-side bus to form a glueless, symmetric, shared-memory multiprocessing system.

For a detailed discussion of the microarchitecture, see Colwell and Steck,[1] the Intel Web site,[2] and Gwennap.[3]

References

1. R. Colwell and R. Steck, "A 0.6-μm BiCMOS Microprocessor with Dynamic Execution," *Proc. Int'l Solid-State Circuits Conf.*, IEEE, Piscataway, N.J., 1995, pp. 176-177.
2. http://www.intel.com/procs/p6/p6white/index.html (Intel's World Wide Web site).
3. L. Gwennap, "Intel's P6 Uses Decoupled Superscalar Design," *Microprocessor Report*, Vol. 9, No. 2, Feb. 16, 1995, pp. 9-15.

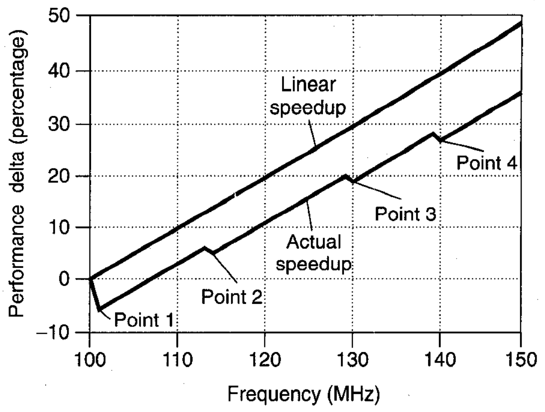


Figure 1. Delivered performance versus clock frequency.

single-cycle execution of basic arithmetic operations at a 50 percent higher frequency. The rest of the pipeline would then have to be retuned to use one-third fewer gates per clock than a comparable Pentium microarchitecture.

We first added a clock cycle to data cache lookup, changing it from two to three cycles. We used the performance simulator to model this change and discovered that it resulted in a 7 percent degradation (increase) in clock cycles per instruction.

The next change was to rework the instruction fetch/decode/rename pipeline, resulting in an additional two stages in the in-order fetch pipeline. Retirement also required one additional pipe stage. This lengthening resulted in a further clock-per-instruction (CPI) degradation of about 3 percent.

Finally, we pursued a series of logic simplifications to shorten critical speed paths. We applied less aggressive organizations to several microarchitecture areas and experienced another 1 percent in CPI loss.

The high-frequency microarchitecture completes instructions at a 50 percent higher rate than the lower frequency microarchitecture, but requires 11 percent more of the now-faster clocks per 100 instructions to enable this higher frequency. The net performance is $(1.5/1.0) * (1.0/1.11) = 1.35$, or a 35 percent performance improvement—a very significant gain compared to most microarchitecture enhancements.

In Figure 1 we see that performance generally improves as clock frequency increases. The improvement is not linear or monotonic, however. Designers must make a series of microarchitectural changes or trade-offs to enable the higher frequency. This in turn results in "jaggies" or a CPI degradation and performance loss at the point at which we make a change. The major drop shown at point 1 represents the 7 percent CPI loss due to the added data cache pipe stage. The series of minor deflections at points 2, 3, and 4 shows the effect of added front-end pipe stages. The overall trend does not continue indefinitely, as the CPI starts to roll off dramatically once the central core no longer maintains one-clock latency for simple operations.

The right of this graph shows a fairly clear performance win, assuming that one picks a point that is not in the valley of one of the CPI dips, and assuming that the project can absorb the additional effort and complexity required to hit higher clock speeds.

When we first laid out the straw man, we did not expect the CPI-clock frequency curve to have this shape. Our ini-

Our initial intuition suggested that the cost of an extra clock of latency on loads would be more severe than it actually is.

tial intuition suggested that the cost of an extra clock of latency on loads would be more severe than it actually is. Further, past industry experience suggests that high frequency at the expense of deep pipelines often results in a relative standstill in performance for general-purpose integer code. However, our performance simulator showed the graphed behavior and the performance win possible from higher clock speeds. Since we had carefully validated the simulator with micro-benchmarks (to ensure that it really modeled the effect in question), we were inclined to believe its results over our own intuition and went ahead with the modified design. We did, however, work to come up with qualitative explanations of the results, which follow.

Consider a program segment that takes 100 clock cycles at 100 MHz. The baseline microarchitecture takes 1 microsecond to execute this segment. We modify this baseline by adding an extra pipe stage to loads. If 30 percent of all operations are loads, this would add 30 clocks to the segment, and take 130 clocks to execute. If the extra pipe stage enables a 50 percent higher frequency, the total execution time becomes 130/150 or 0.867 microseconds. This amounts to a 15 percent performance improvement (1/0.867). This is certainly a higher performance microarchitecture but hardly the 50 percent improvement one might naively expect from clock rate alone. Such a result is typical of past experience with in-order pipelines when we seek the CPI-frequency balance.

The Pentium Pro microarchitecture does not suffer this amount of CPI degradation from increased load latency because it buffers multiple loads, dispatches them out of order, and completes them out of order. About 50 percent of loads are not critical from a dataflow perspective. These loads (typically from the stack or from global data) have their address operands available early. The 20-entry reservation station in the Pentium Pro processor can buffer a large pool of micro-ops, and these "data-ready" load micro-ops can bubble up and issue well ahead of the critical-path operations that need their results. For this class of loads, an extra clock of load latency does not impact performance.

The remaining 50 percent of the loads have a frequent overlap of multiple critical-path loads. For example, the code fragment a = b + c might compile into the sequence

```
load b => r1
load c => r2
r1 plus r2 => r3
```

Both b and c are critical-path loads, but even if each takes an extra clock of latency, only one extra clock is added for both, assuming loads are pipelined and nonblocking. This blocking-factor effect varies, depending upon the program mix. But a rule of thumb for the Pentium Pro processor is that additional clocks of load latency cost approximately half of what they do in a strict in-order architecture.

Thus the 15 percent of micro-ops that are critical-path loads take an extra clock, but the overlap factor results in one half of the equivalent in-order penalty, or about 7.5 percent. This is close to what we measured in the detailed simulation of actual code.

Now let's look at the effect of additional fetch pipeline stages. If branches are perfectly predicted, the fetch pipeline can be arbitrarily long, at no performance cost. The cost of extra pipe stages comes only on branch mispredictions. If 20 percent of the micro-ops are branches, and branch prediction is 90 percent accurate, then two micro-ops in 100 are mispredictions. Each additional clock in the fetch pipeline will add one clock per misprediction. If the base CPI is about 1, we'll see about two extra clocks per 100 micro-ops or and additional 2 percent per pipe stage. The actual penalty is somewhat less, because branch prediction is typically better than 90 percent, and there is a compound-interest effect. As the pipeline gets longer, the CPI degrades and the cost of yet another pipe stage diminishes.

Clock frequency versus effort

This all sounds like a fine theoretical basis for building a faster processor, but it comes at a nontrivial cost. Using a higher clock frequency reduces the margin for error in any one pipe stage. The consequence of needing one too many gates to get the required functionality into a given pipe stage is a 9 to 10 percent performance loss, rather than a 4 to 5 percent loss. So the design team must make a number of small microarchitecture changes as the design matures, since it is impossible to perfectly anticipate every critical path and design an ideal pipeline. This results in rework and a longer project schedule. Further, with short pipe stages, many paths cannot absorb the overhead of logic synthesis, increasing the proportion of the chip for which we must hand-design circuits.

Higher clock speeds require much more hand layout and careful routing. The densities achievable by automatic placement and routing are often inadequate to keep parasitic delays within what will fit in a clock cycle. Beyond that, the processor spends a bigger fraction of each clock period on latched delay, set-up time, clock skew, and parasitics than with a slower, wider pipeline. This puts even more pressure on designers to limit the number of gates per pipe stage.

The higher performance that results from higher clock speeds places more pressure on the clock and power distribution budget. The shorter clock period is less able to absorb clock jitter and localized voltage sags, requiring very careful and detailed speed path simulations.

As long as a design team expects, manages, and supports this extra effort, clock speedups provide an excellent path to higher performance. Even if this comes at some CPI degradation, the end result is both a higher performance product and one that hits production earlier than one that attempts to retrofit higher clock frequency into a pipeline not designed for it.

The terms "architectural efficiency" or "performance at the same clock" are sometimes taken as metrics of goodness in and of themselves. Perhaps this is one way of apologizing for low clock rates or a way to imply higher performance when the microarchitecture "someday" reaches a clock rate that is in fact unobtainable for that design with that process technology. Performance at the same clock is not a good microarchitectural goal, if it means building bottlenecks into the pipeline that will forever impact clock frequency. Similarly, low latency by itself is not an important goal. Designers must consider the balance between latency, available parallelism in the application, and the impact on clock speed of forcing a lot of functionality into a short clock period.

It is equally meaningless to brag about high clock frequency without considering the CPI and other significant performance trade-offs made to achieve it. In designing the Pentium Pro microarchitecture, we balanced our efforts on increasing frequency and reducing CPI. As architects, we spent the same time working on clock frequency and layout issues as on refining parallel-execution techniques. The true measure of an architecture is delivered performance, which is clock speed/CPI and not optimal CPI with low clock speed or great clock speed but poor CPI.

One final interesting result was that the dynamic-execution microarchitecture was actually a major enabler of higher clock frequency. In 1990, many pundits claimed that the complexity of out-of-order techniques would ultimately lead to a clock speed degradation, due to the second-order effects of larger die size and bigger projects with more players. In the case of the Pentium Pro processor, we often found that dynamic execution enabled us to add pipe stages to reduce the number of critical paths. We did not pay the kind of CPI penalty that an in-order microarchitecture would have suffered for the same change. By alleviating some of the datapath barriers to higher clock frequency, we could focus our efforts on the second-order effects that remained.

Tuning area and performance

Another critical tuning parameter is the trade-off between silicon area and CPU performance. As designers of a new microarchitecture, we are always tempted to add more capability and more features to try to hit as high a performance as possible. We try to guesstimate what will fit in a given level of silicon technology, but our early estimates are generally optimistic and the process is not particularly accurate.

As we continued to refine the Pentium Pro microarchitecture, we discovered that, by and large, most applications do not perform as well as possible, being unable to keep all of the functional units busy all of the time. At the same time, better understanding of layout issues revealed that the die size of the original microarchitecture was uncomfortably large for high-volume manufacturing.

We found that the deep buffering provided by the large, uniform reservation station allowed a time-averaging of functional-unit demand. Most program parallelism is somewhat bursty (that is, it occurs in nonuniform clumps spread through the application). The dynamic-execution architecture can average out bursty demands for functional units; it draws micro-ops from a large range of the program and dispatches them whenever they and a functional unit become ready. No particular harm comes from delaying any one micro-op, since a micro-op can execute in several different clocks without affecting the critical path through the flow graph. This contrasts with in-order superscalar approaches, which offer only one opportunity to execute an operation that will not result in adding a clock or more to the execution time. The in-order architecture runs in feast-or-famine mode, its multiple functional units idle much of the time and only coming into play when parallelism is instantaneously available to fit its available templates.

It is equally meaningless to brag about high clock frequency without considering the CPI and other significant performance trade-offs made to achieve it.

The same phenomenon occurs in the instruction decoder. A decoder for a complex instruction set will typically have restrictions (termed "templates" here) placed upon it. This refers to the number and type of instructions that can be decoded in any one clock period. The Pentium Pro's decoder operates to a 4-1-1 template. It decodes up to three instructions each clock, with the first decoder able to handle most instructions and the other two restricted to single dataflow nodes (micro-ops) such as loads and register-to-register. A hypothetical 4-2 template could decode up to two instructions per clock, with the second decoder processing stores and memory-to-register instructions as well as single micro-op instructions.

The Pentium Pro's instruction decoder has a six-micro-op queue on its output, and the reservation station provides a substantial amount of additional buffering. If a template restriction forces a realignment, and only two micro-ops are decoded in a clock, opportunities exist to catch up in subsequent clocks. At an average CPI of about 1, there is no long-term need to sustain a decode rate of three instructions per clock. Given adequate buffering and some overcapacity, the decoder can stay well ahead of the execution dataflow. The disparity between CPI and the maximum decode rate reduce the template restrictions to a negligible impact.

After observing these generic effects, we performed sensitivity studies on other microarchitecture aspects. We trimmed each area of the preliminary microarchitecture until we noted a small performance loss.

For example, we observed that each of the two load/store ports were used about 20 percent of the time. We surmised that changing to one dedicated load port and one dedicated store port should not have a large effect on performance.

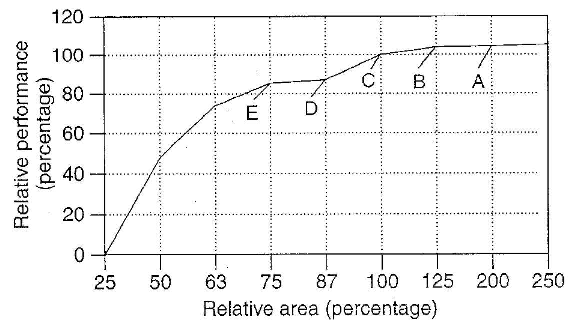


Figure 2. Performance versus die area for different decoder designs.

The load port would operate about 30 percent of the time and the store port at about 10 percent of the time. This proved to be the case, with less than a 1 percent performance loss for this change.

Changing from a 4-2-2-2 decode template (four instructions per clock) to a 4-2-2 template (three instructions per clock) also was a no-brainer, with no detectable performance change on 95 percent of programs examined.

We also changed the renaming and retirement ports from four micro-ops per clock to three, which resulted in a slightly larger, but still manageable 2 percent performance loss.

Finally, we reduced the L1 cache size from 16 to 8 Kbytes. In doing this, we took advantage of the full-frequency dedicated bus we had already chosen. Since the L1 cache is backstopped by a full bandwidth, three-clock L2 cache, the extra L1 misses that result from cutting the L1 cache size cause a relatively minor 1.5 percent performance loss.

The reduction from four- to three-way superscalar operation and the reduction in L1 cache size had some negative impact on chest-thumping bragging rights, but we could not justify the extra capacity by the delivered performance. Further, tenaciously holding on to extra logic would have resulted in significant negative consequences in die area, clock speed, and project schedule.

As the design progressed, we eventually found that even the first round of trimming was not enough. We had to make further reductions to keep die size in a comfortable range, and, as it turned out later, maintain clock frequency. This required making further cuts, which resulted in detectable performance loss, rather than the truly negligible losses from the earlier changes.

We made two major changes. We cut back the decoder to a 4-1-1 template from a 4-2-2 template. This amounted to about a 3 percent performance loss. We also cut back the branch target buffer from 1,024 to 512 entries, which barely affected SPECint92 results (1 percent) but did hurt transaction processing (5 percent). It was emotionally difficult (at the time) for the microarchitecture team to accept the resulting performance losses, but these results turned out to be critical to keeping the die area reasonable and obtaining a high clock frequency. This kind of careful tuning and flexibility in product goals was essential to the ultimate success of the program.

It is important to consider the actual shape of the area-performance curve. Most high-end CPU designs operate well past the knee of this curve. Efficiency is not a particularly critical goal. For example, the market demands as much performance as possible from a given technology, even when that means using a great deal of silicon area for relatively modest incremental gains

Figure 2 illustrates this effect. This graph charts the performance of various instruction decoder schemes coupled to a fixed execution core. All of the architectures discussed earlier are clearly well past the knee of the performance curve. Moving from point A (a 4-2-2-2 template) to point B (4-2-2) is clearly the right choice, since the performance curve is almost flat. Moving down to point C (4-1-1) shows a detectable performance loss, but it is hardly disastrous in the grand scheme of things. Point D (4-2—one we actually considered at one time) occupies an awkward valley in this curve, barely improved over point E (4-1) for significantly more area and noticeably lower performance than point C.

As we converted the microarchitecture to transistors and then to layout, execution quality became critical. All members of the project team participated in holding the line on clock frequency and area. Some acted as firefighters, handling the hundreds of minor emergencies that arose.

This phase is very critical in any major project. If a project takes shortcuts and slips clock frequency or validation to achieve earlier silicon, the design often contains bugs and suffers unrecoverable performance losses. This design phase determines a project's ultimate success. The best planned and most elegant microarchitecture will fail if the design team does not execute implementations well. As CPU architects, we were very fortunate to work with a strong design and layout team that could realize our shared vision in the resulting silicon.

THE PENTIUM PRO PROCESSOR ran DOS, Windows, and Unix within one week of receiving first silicon. We had most major operating systems running within one month. We made a series of small metal fixes to correct minor bugs and speed paths. The A2 material, manufactured using a 0.6-micron process, ran at 133 MHz with a production-quality test program, including 85 degree case temperature and 5 percent voltage margins).

The B0 stepping incorporated several microcode bug and speed path fixes for problems discovered on the A-step silicon, and added frequency ratios to the front-side bus. Our success with early Pentium Pro processor silicon, plus positive early data on the 0.35-micron process, encouraged us to retune the L2 access pipeline. Retuning allowed for dedicated bus frequencies in excess of 200 MHz. We added one clock to the L2 pipeline, splitting the extra time between address delivery and path while retaining the full-core clock frequency and the pipelined/nonblocking data access capability. The 0.6-micron B0 silicon became a 150-MHz, production-worthy part and met our goals for performance and frequency using 0.6-micron Pentium processor technology.

We optically shrank the design to the newly available 0.35-

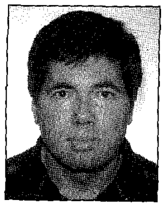
Table 1. Pentium Pro performance.

Processor (0.6 micron, 150 MHz, 256-Kbyte L2 cache)	Processor (0.35 micron, 200 MHz, 256-Kbyte L2 cache)
6.08 SPECint95	8.09 SPECint95
5.42 SPECfp95	6.75 SPECfp95

micron process, which allowed Intel to add a 200-MHz processor to the product line. Table 1 shows the delivered performance on some industry-standard benchmarks.

These results are competitive with every processor built today on any instruction set architecture.

The Pentium Pro processor was formally unveiled on November 1, 1995, 10 months after first silicon. Since then, more than 40 systems vendors have announced the availability of computer systems based on the processor. In the future, we will enhance the basic microarchitecture with multimedia features and ever higher clock speeds that will become available as the microarchitecture moves to 0.25-micron process technology and beyond.



David B. Papworth is a principal processor architect for Intel Corporation and one of the senior architects of the Pentium Pro processor. Earlier, he was director of engineering for Multiflow Computer, Inc., and one of the architects of the first commercial VLIW processor. He holds a BSE degree from the University of Michigan.

Direct comments regarding this article to the author at Intel Corporation, JF1-19, 5200 N.E. Elam Young Parkway, Hillsboro, OR 97124; papworth@ichips.intel.com.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 159 Medium 160 High 161