Intel Reveals Pentium Implementation Details

Architectural Enhancements Remain Shrouded by NDA

By Brian Case

Recently, Intel revealed many of Pentium's microarchitectural details to the Microprocessor Report staff. The company was, at last, very forthcoming about much of the design, except for some details on architectural extensions described below. This article gives an overview of the microarchitecture; for a guide to the complete spectrum of our Pentium coverage, see: ***061201.PDF****,* ***061405.PDF****,* ***070502.PDF****,* ***070503.PDF****.*

Pentium Overview

Figure 1 shows a block diagram of the Pentium design. The most important enhancements over the 486 are the separate instruction and data caches, the dual integer pipelines (the U-pipeline and the V-pipeline, as Intel calls them), branch prediction using the branch target buffer (BTB), the pipelined floating-point unit, and the 64-bit external data bus. Even-parity checking is implemented for the data bus and the internal RAM arrays (caches and TLBs).

As for new functions, there are only a few; nearly all the enhancements in Pentium are included to improve performance, and there are only a handful of new instructions. Pentium is the first high-performance microprocessor to include a system management mode like those found on power-miserly processors for notebooks and other battery-based applications; Intel is holding to its promise to include SMM on all new CPUs.

Pentium uses about 3 million transistors on a huge 294 mm^2 (456k mils^2) die. As is evident from the die photo in Figure 2, the caches plus TLBs use only about 30% of the die. At about 17 mm on a side, Pentium is one of the largest microprocessors ever fabricated and probably pushes Intel's production equipment to its limits.

The integer data path is in the middle, while the floating-point data path is on the side opposite the data cache. In contrast to other superscalar designs, such as SuperSPARC, Pentium's integer data path is actually bigger than its FP data path. This is an indication of the extra logic associated with complex instruction support.

Intel estimates about 30% of the transistors were devoted to compatibility with the x86 architecture. Much of this overhead is probably in the microcode ROM, instruction decode and control unit, and the adders in the two address generators, but there are other effects of the complex instruction set. For example, the higher frequency of memory references in x86 programs compared to RISC code led to the implementation of the dual-access data cache.

Architecture Extensions

While Pentium incorporates several architectural changes from the 486, there are only a few significant ones. For Intel, it makes little sense to change the instruction set of the most successful general-purpose microprocessor architecture in existence. Rumors are rampant, however, that one of Intel's next-generation x86 family members—either P6 or P7—will have a second, RISC-like instruction set.

Many of the architectural changes are either partially or wholly described in Appendix H of the *Pentium Processor User's Manual: Volume 3*; this volume, by itself, is over 1000 pages long. Unfortunately, Appendix H contains only a three-sentence explanation that the information is considered Intel confidential and proprietary and is provided in the *Supplement to the Pentium Processor User's Manual* only under appropriate non-disclosure.

The supplement is supplied only to selected operat-

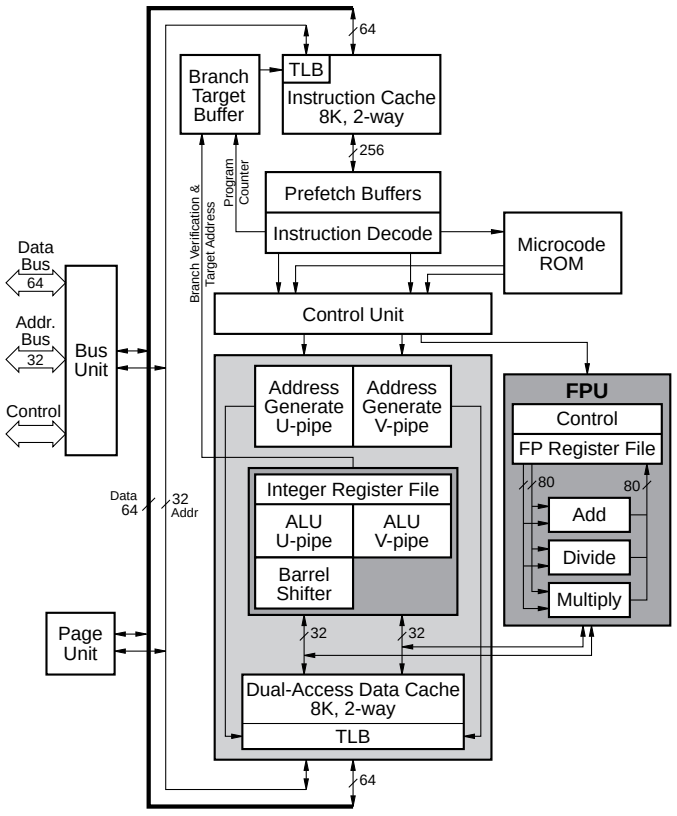

Figure 1. Pentium block diagram.

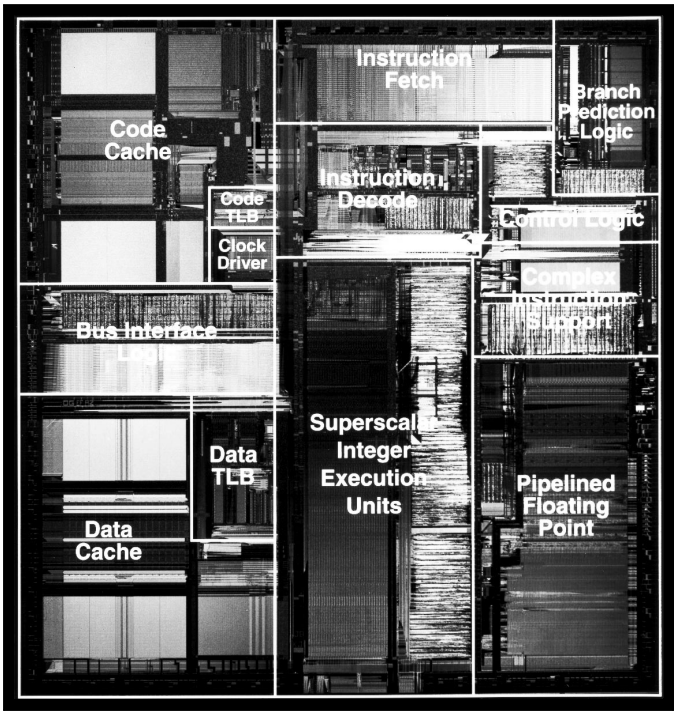

Figure 2. Die photo of Pentium, which incorporates 3.1 million transistors on a 16.7 mm × 17.6 mm die.

ing-system vendors and so was not made available to *Microprocessor Report*. This policy allows Intel to keep Pentium-specific details secret from its competitors. It remains to be seen whether operating systems will come in two versions—one for the 486, one for Pentium—or whether a single version that checks processor type will be delivered.

Only three new instructions have been added to the user-mode instruction set: CMPXCHG8B, CPUID, and RDTSC. CMPSXCHG8B is an eight-byte version of the compare-and-exchange instruction that was introduced on the 486. When used with the LOCK prefix, this instruction can be used as a mutual-exclusion primitive in multiprocessor algorithms.

CPUID is a new instruction that allows a program to directly learn the vendor, family, model, and stepping of the microprocessor on which it is executing. This instruction will return a different piece of information depending on the index value in the 32-bit EAX register. With EAX set to zero, the instruction returns the string "GenuineIntel" as three, four-character ASCII strings in EBX, EDX, and ECX (here we see the influence of marketing). For Pentium, the only other EAX index valid for CPUID is 1, and when run with EAX equal to 1, CPUID returns the stepping, model, family, and feature flags in EAX and EDX. Three of the feature flag bits tell whether there is an on-chip FPU, whether the machine-check exception is implemented, and whether the CMPXCHG8B instruction is implemented. The other six bits are described only in the mysterious Appendix H.

Instruction		Description
MOV	CR4, r32	Move to control register 4
MOV	r32,CR4	Move from control register 4
RDMSR		Read model-specific register
WRMSR		Write mode-specific register
RSM		Return from system-management mode

Table 1. The five new system-mode Pentium instructions.

The third new user-mode instruction is RDTSC, and is described only in Appendix H.

Five new system instructions are implemented to serve new Pentium features and are legal only in privileged execution mode (see Table 1). The MOV instructions access Pentium's control register number 4, which is not implemented in the 486. This control register implements six bits: MCE (enable machine-check exceptions), PSE (documented in Appendix H), DE (enable debugging extensions), TSD (documented in Appendix H), PVI (documented in Appendix H), and VME (documented in Appendix H). The machine-check exception is used to report parity errors, so trapping on parity errors can be turned off by disabling this exception (but parity checking on the bus is always enabled; this will be covered in more detail next issue).

The RDMSR and WRMSR instructions are used to read and write model-specific registers, respectively. The forms of the MOV instruction that were used in the 486 to access the test registers have been removed in Pentium. A new set of test registers has been defined for the caches, TLBs, and the BTB, and these "model-specific" registers—documented in Appendix H—are accessed with RDMSR and WRMSR.

The RSM instruction is used to return from system management mode to the interrupted processor operating mode. System management mode is discussed in detail below.

The 32-bit EFLAGS register has three new Pentium-specific bits. The ID bit allows a program to determine if the processor on which it is running supports the CPUID instruction. If ID can be set and cleared under program control, CPUID is supported. The VIP (virtual interrupt pending) and VIF (virtual interrupt flag) bits support changes to the way virtual-86 mode is implemented on Pentium; unfortunately, information beyond that is contained in Appendix H.

Three new extensions to the exception model are implemented in Pentium. Exception #13, the general protection fault, is triggered by trying to write a 1 into a reserved bit in a special register. Exception #14, the page-fault exception, is triggered on Pentium in the case of a page fault or when a 1 is detected in any reserved bit position in a page table entry, a page directory entry, or the page directory pointer during address translation. Exception #18, the machine check exception, is used to report parity and other hardware errors.

Pentium extends the virtual address translation

model of the 486. Only 4K pages are implemented by 486 address translation, but Pentium also implements 4M pages. The documentation for the 4M page-table entries is contained in Appendix H, but it seems likely that the page directory entry, which normally points to a table of 1024 4K page table entries, is used alone to describe a single, 4M page.

Pentium implements some extensions to the virtual-86 processor mode, which allows a program written for the 8086 to be run in a virtual machine environment as a separate, protected task. The extensions, such as the VIP and VIF bits in the EFLAGS register, are documented only in Appendix H. These extensions are rumored to dramatically speed interrupt handling in virtual-86 mode.

System Management Mode

Pentium is the first high-performance microprocessor to implement a system management mode (SMM). Ordinarily, an SMM capability is provided in processors for portable applications to implement power-saving functions, such as powering down peripherals and then restarting them only when they are accessed. Until now, Intel has implemented SMM only in its 386SL and 486SL.

SMM is a unique processor operating mode beyond all other normal processor modes. This mode is implemented in hardware with a separate SMM interrupt request pin and a status pin that indicates the processor is in SMM. This pin can be used externally to enable a special SMM memory area. Just as interrupts and traps allow an operating system to transparently add functions to application software, SMM allows software functions to be added to a system without making changes to the operating system.

Pentium support for SMM consists of the SMI# interrupt input pin, the SMIACT# status output pin, and the RSM instruction. Triggering SMI# is the only way to enter SMM.

When SMI# is triggered, Pentium automatically saves its register state in an area of SMRAM (SMM memory) and disables further interrupts. Interrupts can be re-enabled in SMM, but only after taking special care to set up correct interrupt vectors.

By default, SMM begins execution at 0x8000 in the CS segment. The SMRAM address space is essentially a real-address mode, flat, 4G linear address space. The default operand and address sizes are set to 16 bits, but operand-size and address-size override prefixes can be used to access data and code anywhere in the 4G SMRAM space. When the SMM routine is finished, SMM is exited with the special RSM instruction.

Besides implementing procedures to save power, SMM can be used to implement security options and other features. While SMM may not be used by some

Superscalar Instruction Pairing Rules

Pentium can issue two integer instructions per clock cycle so long as they satisfy the following constraints:

- Both instructions must be "simple."
- There must be no read-after-write or write-after-write register dependencies.
- Neither instruction may contain both a displacement and an immediate value.
- Instructions with prefixes (other than jump-conditional with 16/32-bit prefix) can occur only in the U-pipeline.

For the purposes of these rules, simple instructions are:

- MOV register←register/memory/immediate
- MOV memory←register/immediate
- ALU-op register←register/memory/immediate
- ALU-op memory←register/immediate
- INC register/memory
- DEC register/memory
- PUSH register/memory
- POP register
- LEA register/memory
- JUMP/CALL/Jcc near
- NOP

These simple instructions are hardwired and execute in a single clock cycle except for "ALU-op register←memory," which takes two clocks, and "ALU-op memory←register/immediate," which takes three. Another exception to the pairing rules occurs for shifts: they can be executed only in the U-pipeline, so they must be the first instruction in a pair.

Implicit register dependencies (usually based on the condition codes) can also prevent dual-instruction issue. For example, an ALU instruction that sets the carry flag cannot be paired together with an ALU instruction that reads the carry flag.

There are, however, two important exceptions which allow dependent instructions to be paired. The first exception allows a compare and conditional branch that tests the result of the compare to be paired, while the second allows pairs of pushes or pops to be paired. Branch prediction helps the compare/conditional-branch case, and special hardware is included to resolve the dependency on the stack pointer for pushes and pops.

In general, an integer and floating-point instruction pair, or a pair of floating-point instructions, cannot be simultaneously issued. There is one exception: a simple floating-point load, arithmetic, or compare can be paired with an FXCH (floating-point exchange) instruction. The FXCH must be the second instruction in the pair. If an integer instruction immediately follows the FXCH, it will stall for one or four clocks depending on the operands to the pair of floating-point instructions. Simple floating-point instructions are:

- FLD single/double, FLD ST(i),
- all forms of FADD, FSUB, FMUL, FDIV,
- all forms of FCOM, FUCOM, FTST, FABS, and FCHS.

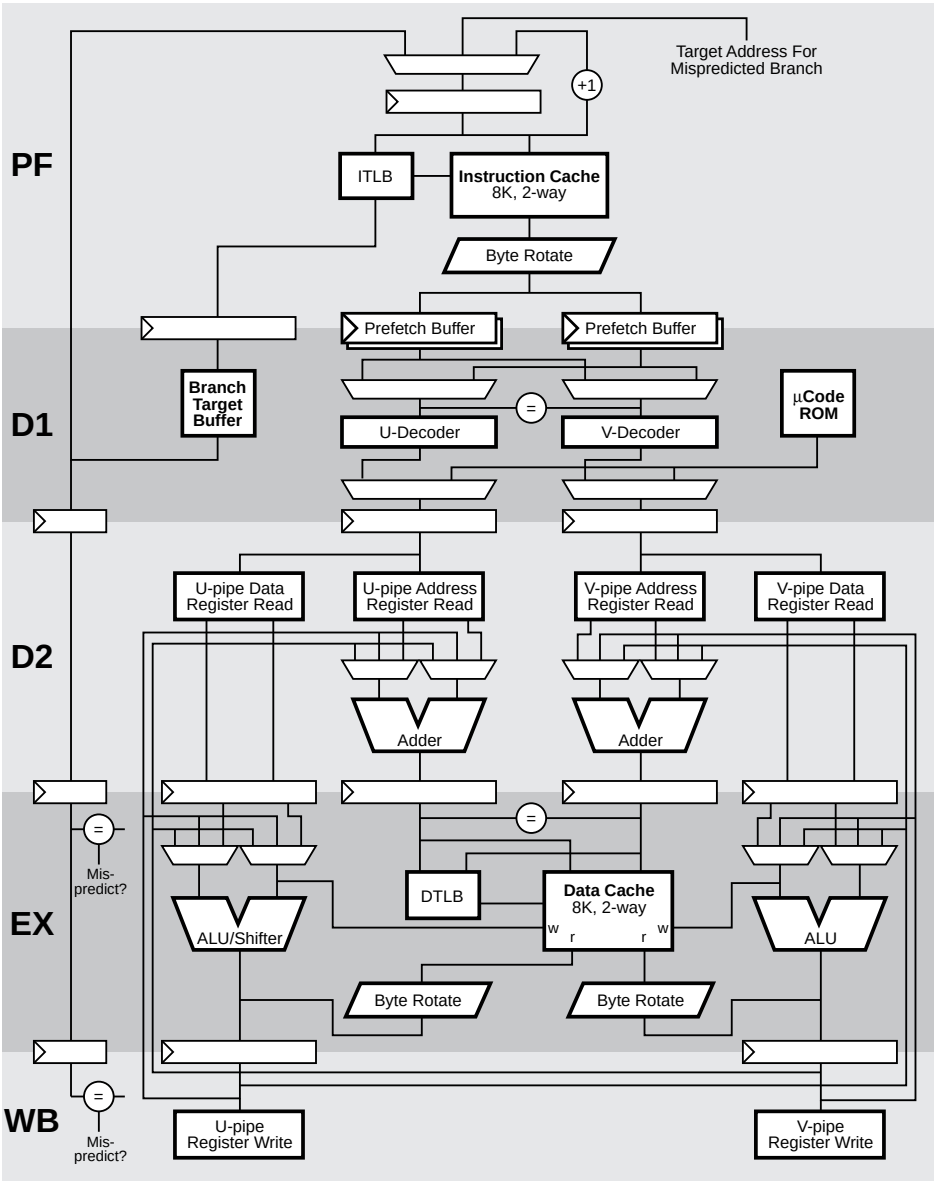

FIgure 3. Block diagram of the major Pentium integer pipeline resources.

of the first Pentium systems, its inclusion in Pentium shows that Intel has made good on its commitment to include SMM as a part of its mainstream x86 processors. A future 3.3V version of Pentium will raise the importance of SMM.

Microarchitecture Overview

Superficially, Pentium's microarchitecture looks like a superscalar version of the 486. Even though Pentium has two integer pipelines, the basic five-stage pipeline structure is unchanged from the 486, as shown in Figure 3. (Note that Figure 3 is a functional diagram, not a timing diagram; thus, a multiplexer may be shown where one is not actually present.) U is the "default" pipeline when two instructions cannot be issued simultaneously, and the U pipe is slightly more powerful since it has a barrel shifter.

The pipelines are similar to the 486's: each pipeline begins with instruction prefetching, which loads prefetch buffers, and instruction decoding is spread out over two pipeline stages to accommodate some of the more semantically rich (i.e., complex) instructions. The last two stages are the traditional execute and writeback pipeline phases.

While Pentium uses the high-level structure of the 486 pipeline, there are many subtle implementation differences. For example, the total prefetch capacity has been increased by a factor of four, and the address adders in the D2 stage have four instead of three inputs to reduce by one the number of cycles for some complex addressing modes.

Prefetch Stage

The prefetch stage incorporates one of the most significant enhancements of Pentium over the 486: a separate, 8K instruction cache. The cache has a two-way set-associative organization with LRU (least-recently used) replacement and a line size of 32 bytes. Intel chose two-way set associativity as a compromise between performance and implementation constraints. Also, note that Pentium has two two-way set-associative caches vs. the 486's single four-way cache.

Full coherency is maintained between the on-chip caches and external memory with hardware snooping. The instruction cache tags are triple-ported: one port is for snooping operations while the other two are used for the split fetch capability (described below). This means the snooping hardware and the processor can access the cache simultaneously with no contention. The cache implements parity, one bit per eight bytes of data and one bit per tag.

Of course, having a separate instruction cache improves instruction fetch efficiency because data and instruction accesses do not compete for a single cache resource, but Pentium further improves instruction fetching by implementing a "split fetch" capability not present in the 486. (Split fetching was first implemented in the 960CA.) Split fetching gives Pentium the ability to fetch

a contiguous block of instruction bytes even if the block is split across two instruction-cache lines. As shown by the worst-case alignment scenario in Figure 4, this allows a minimum of 17 bytes to be fetched from the cache because a fetch can straddle the boundary between two consecutive half-lines. According to Intel's measurements, the split fetch capability improves Pentium performance by a few percent.

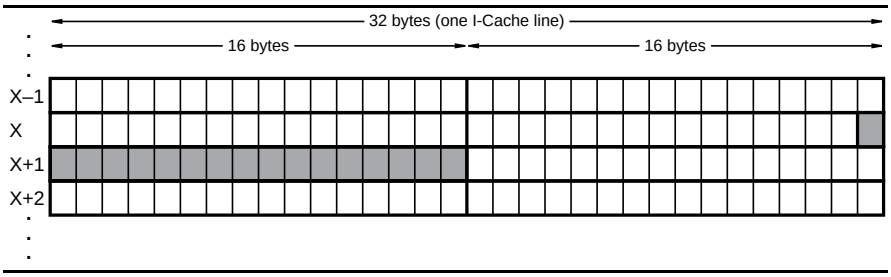

Figure 4. The instruction cache allows "split fetching" across the boundary from one cache line to the next. The worst-case situation, as shown, still delivers 17 bytes in a single cache access.

Pentium's split fetching is an example of an important technique for superscalar processors: eliminating instruction-fetch alignment restrictions. In a superscalar processor, the goal is to simultaneously issue and execute the maximum allowable number of instructions as often as possible. Other superscalar processors also implement some form of split fetching—although different names are used—to make sure that instruction fetching is not the limiting factor. All other existing superscalar processors, however, are RISCs. The word-alignment of RISC instructions results in less complex logic to eliminate alignment restrictions. The split-fetching logic, which must take care of byte-aligned x86 instructions, is one place where Pentium pays a price for the complex x86 architecture.

The instruction TLB is four-way set-associative, has 32 entries, and uses a pseudo-LRU replacement algorithm; ITLB misses are handled in hardware. The dedicated ITLB allows the I-cache to be physically tagged, which reduces the frequency of I-cache flushes. The 486 also indexes its cache with physical addresses.

Instruction bytes that are fetched from the I-cache are aligned, if necessary, and stored in one of the four prefetch buffers. Each buffer is the length of one cache line (32 bytes) for a total of 128 bytes. In contrast, the 486 has only 32 bytes of prefetch buffer.

Coupled with the dedicated instruction cache, the prefetch buffers should virtually guarantee that Pentium never waits for instruction bytes, except in the case of cache misses and mis-predicted branches. In situations where the 486 would stall waiting to fill its prefetch buffer, Pentium will continue executing.

First Decode Stage

The major function of the D1 stage is instruction decoding. Of course, Pentium is designed to decode in hardware as many of the most frequently occurring instructions as possible. Even the rather complex—at least by RISC standards—memory-to-register and register-to-memory arithmetic operations do not require microcode assistance for their processing. Instead, a single, internal microword is generated by the D1 decoding logic that triggers a simple hardware state machine in the EX stage. Thus, while memory/register operations do not require microcode, they do still require sequencing and multiple cycles.

For instructions that are complex enough to require a microcode routine, the first microword is always generated by the D1 decoding logic. In contrast, the D1 decoding logic in the 486 generates the first microcode ROM address. Thus, Pentium achieves at least some speedup over the 486 for microcoded instructions by directly generating the first microword.

For microcoded instructions, the first microword proceeds to the D2 stage, where the microcode engine takes over the Pentium execution resources. As shown in Figure 3, microwords from the microcode ROM control both integer pipelines; consequently, the pipelines operate independently only for pairs of instructions that use hardwired control. Intel has, of course, written the microcode routines to take maximum advantage of the dual pipelines.

This allows Pentium to reduce the number of cycles needed for many of the complex x86 instructions. For example, repeated string move instructions execute at one clock per iteration, compared to three clocks on the 486. The Pentium microcode actually contains an unrolled loop that writes the element of the destination string in the U pipeline in parallel with the reading of the next source string element in the V pipeline.

Pentium microwords are 92 bits long, and the microcode ROM contains about 4K microwords. Since microcoded routines take over all the execution resources, it is not possible for Pentium to pair microinstructions with regular, x86 instructions. Thus, instruction fetching and dispatch are stalled during the execution of a complex, microcoded instruction.

In Figure 3, the circle containing the equal sign between the two inputs to the decoder blocks represents logic that detects resource conflicts. Situations such as register dependencies that require serial execution are detected here. When a conflict is detected, the instruction at the head of the U pipeline gets priority.

Branch prediction, also a major function of the D1 stage, is covered below.

Cycle	EX-Stage Activity	
	U Pipeline	V Pipeline
n	**load**	*-idle-*
n + 1	**ALU**	*-idle-*
n + 2	**store**	**load**
n + 3	*-idle-*	**ALU**
n + 4	*-idle-*	**store**

Table 2. Execute-stage activity for two register-to-memory instructions.

Second Decode Stage

The D2 stage is an artifact of the x86 architecture. Since so many of the instructions specify a multi-component address computation, it makes sense to have dedicated resources and a separate pipeline stage in which to perform the address addition.

Each of the two integer pipelines has a dedicated, four-input address adder. Four inputs are needed because x86 operand addresses can consist of a segment descriptor base, a base address from a general register, an index from a general register (possibly scaled), and a displacement from the instruction. The 486 address adder has only three inputs; thus, some instructions that spend only one cycle in D2 on Pentium will spend two cycles in the D2 stage on the 486. (In Figure 3, the address adders are drawn with only two inputs simply to save space.)

What is not shown in the D2 stage in Figure 3 is the separate, four-input segment limit-check adder. Architecturally, x86 addressing requires that all segment accesses be checked against the limit stored in the segment descriptor. This check requires a separate four-component addition, and Pentium has yet two more four-input adders to perform this check in parallel. As with the address adders, the 486 limit-check adders have only three inputs. While the need for this hardware probably has little or no effect on the cycle time of the Pentium implementation, it certainly requires significant area and power. This is another way that Pentium pays for the complexity of the x86 architecture.

The other major function of the D2 stage is reading operands from the register file for use by the ALUs in EX.

Execute Stage

The execute stage contains the ALUs and the data cache. The U-pipe has a full ALU and a barrel shifter, while the V-pipe has only a full ALU. Thus, all shift instructions must be processed in the U-pipe, and the logic in the D1 stage that detects resource requirements takes care of enforcing this rule.

The data cache is one of Pentium's most interesting features. Like the instruction cache, it is a two-way set-associative, 8K cache with a 32-byte line size. A MESI coherency protocol is used to keep caches coherent in a multiprocessor system. As mentioned earlier, the cache tags are triple-ported to allow concurrent snooping and dual access by the pipelines. The cache has a parity bit for each tag and each byte of data.

As explained in ***061201.PDF***, this dual-access capability, which lets both pipelines access the data cache simultaneously, is implemented by interleaving the data array into eight banks (four-byte granularity within a 32-byte cache line). As long as the data accesses from each pipe are to separate banks, both accesses can be processed simultaneously by the cache in a single cycle. This capability is not provided by any other existing microprocessor. (The circle containing the equal sign between the two inputs to the cache and DTLB represents the bank conflict detection logic.)

Since the cache stores physical tags, it is also necessary that the data TLB be able to perform two address translations simultaneously. This capability is provided by the dual-ported, 64-entry, four-way set-associative DTLB.

The DTLB stores translations for the standard 4K pages of the 386 architecture. There is a separate eight-entry, four-way set-associative DTLB for 4M pages that is also dual ported. Large-page mapping is standard on all high-end processors and is useful because mapping graphics frame buffers and operating-system segments can be done with only one 4M translation entry instead of many 4K entries. This keeps frame-buffer references from "polluting" the main TLB.

Most instruction dependencies are resolved in D1, but there is one important case that is resolved in EX: two register/memory operations. In this case, the two instructions are simultaneously issued into the U and V pipelines, and they proceed concurrently to the EX stage. Once there, however, Pentium forces serialized execution, as shown in Table 2. All pairs of register/memory instructions are serialized in the EX stage to avoid the complexity of checking for dependencies. Even though the instructions are serialized, the overlap of the store of the first and the load of the second at cycle n+2 saves one clock.

In general, the U and V pipes will be simultaneously executing separate instructions only if the instructions they contain are independent. The exceptions are register/memory operations (which get sequenced and serialized in hardware as just described), stack operations (any combination of push and pop), and compare/conditional-branch.

The compare/conditional-branch situation is allowed because branch prediction will likely provide the branch target anyway. If branch prediction is correct, a cycle is saved by pairing the compare and the conditional branch. Since most compare/conditional-branch pairs that occur during program execution will be in loops, and since most loops execute many times, branch prediction should perform very well for this situation.

Note that if the U-pipe contains any kind of branch,

the V-pipe will be idle.

Writeback Stage

The major function of the WB stage is to provide a time slot for writing results of computations and loads into the register file. This is shown conceptually in Figure 3 with separate boxes in the WB stage, but actually there is, of course, only a single register file.

Branch Prediction

Pentium uses a BTB (branch target buffer) for its branch-prediction algorithm. All taken branches are buffered. As shown in Figure 3, the BTB is accessed in stage D1 with the linear address (with the segment calculations done but not translated by TLB) of the branch instruction itself. The BTB stores a single predicted target for a branch. As shown in Figure 5, the BTB cache stores 256 branch predictions with a four-way set-associative organization. Note that this is different from the branch target cache in the 29000, which stores the first few instructions at the branch target. Pentium's BTB stores target addresses only.

Intel simulated several branch prediction algorithms, finally settling on the method described in a paper from the University of Wisconsin (J. Lee and A. J. Smith, "Branch Prediction Strategies and Branch Target Buffer Design," IEEE Computer, January 1984, pp. 6–22.). This algorithm uses two bits to hold the prediction state, with transitions between the four states occurring as necessary when a branch is encountered.

Figure 6 shows the state-transition diagram. The four states are ST (strongly taken), WT (weakly taken), WNT (weakly not taken), and SNT (strongly not taken). Each time there is a hit in the BTB (though not necessarily a correct prediction), the state bits are updated. When the state bits are either ST or WT, the next prediction for the given branch will be "taken," and WNT and SNT mean the next prediction will be "not taken."

The two middle states provide a degree of misprediction hysteresis to avoid thrashing in certain cases. The hysteresis is provided by the fact that it takes two consecutive incorrect predictions to change the prediction polarity. For example, a branch that has been correctly predicted as not-taken many times in a row will continue to be predicted as not-taken even if the branch is occasionally taken.

The BTB allocation policy is that an uncached branch allocates an entry in the cache only if it is a taken branch (i.e., no allocate on miss). As a result, the state bits are always initialized to ST for a newly allocated branch. Branches that cause a miss in the BTB are initially assumed (predicted) to be not-taken.

As an example of the prediction state transition operation, if this newly allocated branch is not taken the next time it is encountered, its state bits will make a transition to WT. The next prediction will thus be "taken," but if this is also a misprediction, the prediction state will make the transition to WNT. The next prediction will be "not taken," and so on.

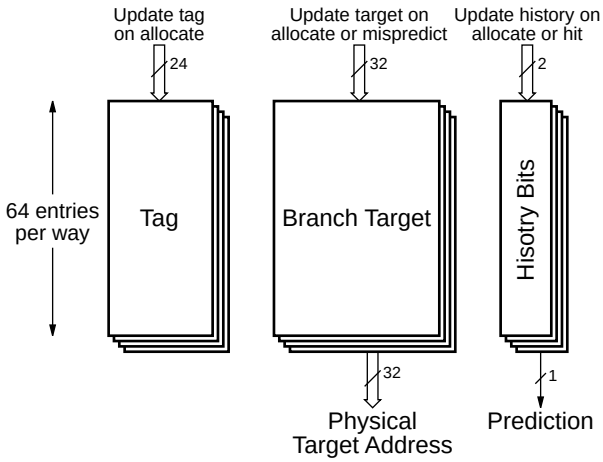

Figure 5. Pentium branch target buffer (BTB) structure.

Down the left side of Figure 3 is a (very simplified) pipeline path that is used to verify branch prediction. The predicted direction for the branch is carried along with the branch instruction as it moves through the pipeline. As soon as possible, the prediction and the actual direction taken are compared. For unconditional branches in the V pipeline and all branches in the U pipeline, the comparator (circle with equal sign) in the EX stage does the check. For conditionals in V, the check is made by the comparator in WB to allow resolution of a possible paired "compare" in the U pipe.

When an incorrect prediction is discovered or when the predicted target is wrong, the pipelines are flushed and the correct target fetched. Thus, based on the stage in which the misprediction is discovered, mispredicted unconditionals and U-pipeline conditionals incur a three-clock delay, while V-pipeline conditionals incur a four-clock delay.

Intel has made some measurements of branch behavior on Pentium. For the programs in the SPEC89 suite, the percent of dynamic branches correctly predicted is between 75% and 85%, including not-taken branches that miss. The branch distribution between pipelines appears to be balanced at about 50% for each pipeline on code produced by both 486-optimized and Pentium-optimized compilers.

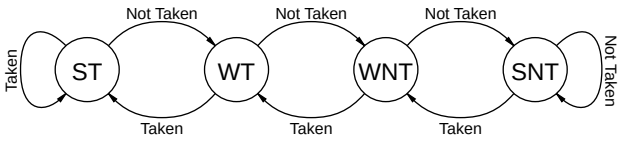

Figure 6. Prediction history bit state transition diagram.

Processor	FP Add	FP Sub	FP Mult	FP Div
Pentium	3/1	3/1	3/2*	39/39
486	8-20/8-20	8-20/8-20	16/16	73/73
R4000	4/3	4/3	8/4	36/36
Alpha	4/1	4/1	4/1	61/61
PowerPC 601	4/1	4/1	4/2	31/29

Table 3. Floating-point latencies/throughputs for some modern microprocessors. Times are for double-precision operations except for Pentium, which supports an 80-bit internal format.
*For pairs of back-to-back multiplies and adds, Pentium has a throughput of one instead of two.

Fast Floating-Point

In any benchmark comparison of high-performance processors, the 486 stands up reasonably well in integer results but trails dramatically in floating point performance *(see **070401.PDF**)*.

Pentium's floating-point performance is vastly improved over the 486 because the simple, serial floating-point unit of the 486 is replaced with fully pipelined, parallel execution units. The FPU pipeline is eight stages, where the first five are shared with the integer pipeline:

- PF (prefetch)
- D1 (instruction decode)
- D2 (address generation)
- EX (memory and register read, memory write if FP store instruction)
- X1 (FP execute first stage, write operand to FP register file if FP load)
- X2 (FP execute second stage)
- WF (rounding and write result to FP register file)
- ER (error reporting, update status word)

This pipeline structure is similar to that of other high-performance processors. For example, the PowerPC 601 has a six-stage floating-point pipeline. As shown in Table 3, Pentium has floating-point operation latency and throughput that is comparable to other processors for basic arithmetic operations.

As with most other high-performance processors, Pentium allows concurrency between the floating-point and integer units. Thus, the issue and execution of integer instructions can proceed in parallel with a long-latency floating-point operation.

One area where Pentium may actually feel some competition is in the Windows NT market *(see **0704ED.PDF**)*. From Table 3, it is tempting to conclude that Pentium could approximately match the floating-point performance of low-end implementations of its Windows NT competitors *(see benchmark results in **070401.PDF**)*. Pentium is hampered, however, by its stack-oriented floating-point register file architecture and by the need to transfer floating-point condition codes to the integer unit before a conditional branch can be executed.

For floating-point operands, Pentium maintains backward compatibility with previous x86 FPUs: there is a file of eight, 80-bit operand registers that are conceptually a stack and only marginally directly addressable. Since most floating-point instructions implicitly use the top of this register stack as one operand, there is a "top-of-stack bottleneck." To circumvent this, programs use the FXCH (floating-point register exchange) instruction to swap the top of stack with an operand deeper in the register file.

Pentium's designers added logic to allow superscalar issue and execution for a simple floating-point operation followed by an FXCH (see sidebar above). This is the only case of superscalar issue for floating-point instructions and is subject to the restriction that the first instruction must be "simple" and the FXCH must be the second instruction in the pair.

Even with the rapid execution of an FP-operation/FXCH pair, Pentium will be hampered by the small, eight-register file. In addition, an FP-operation/FXCH pair followed immediately by an integer instruction will incur a one-cycle penalty.

Another performance problem for Pentium is presented by branching on floating-point conditions. Most microprocessor architectures allow the results of a floating-point comparison to be tested directly, but the x86 architecture requires that the floating-point condition codes be transferred to the integer condition-code register, where a normal integer conditional branch can test them.

To effect a floating-point conditional branch requires four instructions:

1. An FP operation that sets the condition codes
2. FSTSW AX (move FP status word to AX register)
3. SAHF (transfer to upper half of EFLAGS)
4. Jcc (integer jump conditional)

This sequence takes nine clock cycles to execute on Pentium because the floating-point condition codes are updated late in the floating-point pipeline. Four of these clocks can be recovered by inserting integer instructions between instructions 1 and 2.

Although many floating-point loops iterate based on an integer condition, such as a loop count equal to the number of elements in an array, the need to transfer condition codes from the FPU to the integer unit creates a significant penalty for the case of loops with a floating-point termination condition, and for if-then statements with floating-point conditions.

In the final analysis, Pentium will bring a new level of floating-point performance to the PC market. It will not, however, out-perform its Windows NT competitors because of the weaknesses of its floating-point architecture and because the R4000 and Alpha processors will be operating at much higher raw clock speeds.

Conclusions

Pentium solidifies Intel's position as the premier supplier of advanced microprocessors for the PC market. While it will be expensive and difficult to manufacture in volume at first, Pentium uses advanced processor implementation techniques while maintaining full compatibility with the installed base of x86 application software. The superscalar integer unit, separate caches, branch prediction, and pipelined floating-point unit are all significant performance enhancements to the 486. Pentium's snooping-based MESI cache-coherency protocol makes it appealing for multiprocessor implementations.

Like many of the current generation of high-end microprocessors, Pentium integrates a huge number of transistors. At three million, the only other processors in this transistor-count league are SuperSPARC (3.1 million) and the PowerPC 601 (2.8 million). Those processors, however, have at least twice as much total cache and are more aggressive in other ways. It appears that many of Pentium's "extra" transistors are spent on things like internal parity, the triple-ported cache tag arrays, dual-ported TLBs, and adders for multi-component addressing modes and segment limit checking.

Certainly, a significant amount of Pentium's complexity is the result of the complex x86 instruction set. The four-input address adders, microcode ROM, extra decode pipeline stage, and register/memory sequencing logic in the execute stage are all extra complexity not present in RISC processors.

While any x86 program will benefit from Pentium's performance features, the full performance potential will be realized only for programs that are structured to take maximum advantage of Pentium's capabilities. Instruction sequences must be carefully selected to use the instructions that can be dual-issued and, as shown in the floating-point conditional-branch example above, scheduled to fill all available execution slots.

Pentium is a significant microprocessor milestone. It implements sophisticated caching, multiprocessor support, and branch prediction. It is also the first superscalar CISC microprocessor and the first high-end microprocessor to implement a system-management mode. The Pentium core will be around for many years to come because Intel will be able to exploit it by offering an array of microprocessors with varied cache sizes, bus widths, and bus speeds. As for Pentium's technological position in the marketplace, some RISCs will be faster or cheaper or both, but with x86 compatibility, multiprocessor support, and significant performance gains over the 486, Pentium will satisfy most users' needs.♦

*Next issue, we'll examine Pentium's new bus structure (see **070502.PDF**).*

