UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2002

BEng Honours Degree in Computing Part III
MEng Honours Degree in Electrical Engineering Part IV
BEng Honours Degree in Information Systems Engineering Part III
MEng Honours Degree in Information Systems Engineering Part III
MSc in Advanced Computing
for Internal Students of the Imperial College of Science, Technology and Medicine

This paper is also taken for the relevant examinations for the Associateship of the City and Guilds of London Institute

PAPER C332=I3.26=E4.31

ADVANCED COMPUTER ARCHITECTURE

Thursday 2 May 2002, 14:30
Duration: 120 minutes

Answer THREE questions

Paper contains 4 questions
Calculators not required

1 This question concerns instruction caching and branch prediction in the Intel Pentium 4 processor, as described in the paper "The Architecture of the Pentium 4 Processor" (Hinton et al, Intel Technology Journal Q1 2001), which you should have available to you in the examination. **See, in particular, pages 4 and 5.** Where the paper is incomplete, you are invited to speculate using your understanding of the underlying architectural principles.

a Under what circumstances is the Frontend BTB used?

b How is the prediction from the Frontend BTB used?

c There could be two or more different trace cache blocks corresponding to the same IA-32 instruction address. Why?

Now, consider a loop such as the following:

```
j = 0
for (i=1; i<N; i++) {
  j = j+1;
  if (j<M) {
    A[i] = B[j];
  } else {
    j=0;
  }
}
```

d Suppose `M=2`. Assuming a one-bit branch predictor, what branch misprediction rate would you expect? Why?

e Suppose `M=2`. Assuming a two-bit bimodal branch predictor, what branch misprediction rate would you expect? What is the worst-case behaviour?

f Suppose `M=2`. Now assume a two-level correlating $(2,2)$ "gselect" branch predictor. What branch misprediction rate would you expect? Why?

(The six parts carry, respectively, 10%, 20%, 20%, 10%, 20% and 20% of the marks).

2 This question concerns dynamic instruction scheduling in the Intel Pentium 4 processor, as described in the paper "The Architecture of the Pentium 4 Processor" (Hinton et al, Intel Technology Journal Q1 2001), which you should have available to you in the examination. **See, in particular, pages 6 and 7.** Where the paper is incomplete, you are invited to speculate using your understanding of the underlying architectural principles.

a Explain how there could be several distinct instances of a given register (such as EAX) in the processor at the same time. Illustrate your answer with reference to a sample assembly code sequence (in an instruction set of your choice).

b At what point in instruction processing is the Front-end RAT (Register Alias Table) updated? What does the new entry mean?

c At what point in instruction processing is the Retirement RAT (Register Alias Table) read?

d Under what circumstances is a physical register de-allocated?

Now, consider the following code fragment (this is the inner loop of a matrix multiply, compiled using Microsoft Visual Studio 6.0):

```
$L170:
   fld     ST(0)              ; duplicate value at top of FP register stack
   add     ecx, 4
   fmul    DWORD PTR [ecx-4]  ; multiply top-of-stack by data at location [ecx-4]
   add     eax, 4
   dec     edx
   fadd    DWORD PTR [eax-4]  ; add top-of-stack and data at location [eax-4]
   fstp    DWORD PTR [eax-4]  ; store top-of-stack to location [eax-4]
   jne     SHORT $L170
```

e *Estimate* the number of clock cycles between the start of execution of the first uop of the loop, and execution of the final store (`fstp`). Assume cache hits and correct branch prediction wherever necessary. Note that floating-point loads have a 6-cycle latency. Assume that the floating-point multiply latency is 4 cycles, and the floating-point add latency is 2 cycles.

f A dynamically-scheduled processor like the Pentium 4 should be able to execute two or more iterations of this loop in parallel. List four architectural constraints which limit the amount of parallelism which can be exploited from this loop. Give a very brief explanation in each case.

(The six parts carry, respectively, 15%, 15%, 20%, 20%, 10% and 20% of the marks).

3a What are the names of the four cache line states in the Berkeley cache coherency protocol?

b In your answer above, you will have identified two "dirty" states. Under what circumstances would a cache line in a given processor P_i change from one dirty state to the other, and then back again?

c When a processor P_j suffers a read miss, where does it get the data from? There are two cases; explain what distinguishes the two situations.

d "Read snarfing" is a proposed modification to the Berkeley protocol. The idea is that, since each cache controller has to process every read miss request and reply (because they are all broadcast on the bus), it may as well update its cache with the resulting value.

(i) Under which precise circumstances might it make sense for a cache controller to do this?

(ii) For what kind of application program behaviour might read snarfing be a good idea?

(iii) What are the potential disadvantages of read snarfing?

(The four main parts carry, respectively, 20%, 20%, 20% and 40% of the marks).

4a Consider the following C code:

```
char A[2000000], B[1000000];  /* (assume no padding between A and B) */

void V1() {
  int i;
  for (i=0; i<1000000; i++)
    B[i] = A[i] + A[i+OFFSET] + A[i+OFFSET*2];
}

void V2() {
  int i;
  for (i=0; i<1000000; i++)
    B[i] = A[i] + A[i+OFFSET];

  for (i=0; i<1000000; i++)
    B[i] = B[i] + A[i+OFFSET*2];
}
```

Note that functions `V1` and `V2` have the same effect. On a certain laptop computer, the following run-times were observed (in milliseconds):

OFFSET value	Runtime for V1	Runtime for V2
1 bytes	122 ms	164 ms
2048 bytes	122 ms	164 ms
4096 bytes	1344 ms	177 ms
8192 bytes	1346 ms	176 ms

(i) Explain why `V2` is sometimes faster than `V1`.

(ii) What can you tell from this data about the size and associativity of this machine's cache?

b Consider the following loop:

```
     for (i=1; i<=4; i++) {
S1:    B[i] = B[i] + A[i];
S2:    A[i] = A[i-1] + A[i+1];
S3:    C[i] = A[i+1] + B[i];
     }
```

(i) Write down *all* the dependences present in this loop and indicate in each case whether the dependence is loop-carried.

(ii) Draw the iteration space graph for the loop and its three constituent statements. Show the dependences as arrows, and indicate the normal execution order using a dotted path.

(iii) Using this analysis, show how the loop can be split into separate loops executed one after the other.

(The five parts carry, respectively, 10%, 15%, 25%, 25% and 25% of the marks).

End of Paper

© University of London 2002

The Microarchitecture of the Pentium® 4 Processor

Glenn Hinton, Desktop Platforms Group, Intel Corp.
Dave Sager, Desktop Platforms Group, Intel Corp.
Mike Upton, Desktop Platforms Group, Intel Corp.
Darrell Boggs, Desktop Platforms Group, Intel Corp.
Doug Carmean, Desktop Platforms Group, Intel Corp.
Alan Kyker, Desktop Platforms Group, Intel Corp.
Patrice Roussel, Desktop Platforms Group, Intel Corp.

Index words: Pentium® 4 processor, NetBurst™ microarchitecture, Trace Cache, double-pumped ALU, deep pipelining

ABSTRACT

This paper describes the Intel® NetBurst™ microarchitecture of Intel's new flagship Pentium® 4 processor. This microarchitecture is the basis of a new family of processors from Intel starting with the Pentium 4 processor. The Pentium 4 processor provides a substantial performance gain for many key application areas where the end user can truly appreciate the difference.

In this paper we describe the main features and functions of the NetBurst microarchitecture. We present the front-end of the machine, including its new form of instruction cache called the Execution Trace Cache. We also describe the out-of-order execution engine, including the extremely low latency double-pumped Arithmetic Logic Unit (ALU) that runs at 3GHz. We also discuss the memory subsystem, including the very low latency Level 1 data cache that is accessed in just two clock cycles. We then touch on some of the key features that allow the Pentium 4 processor to have outstanding floating-point and multi-media performance. We provide some key performance numbers for this processor, comparing it to the Pentium® III processor.

INTRODUCTION

The Pentium 4 processor is Intel's new flagship microprocessor that was introduced at 1.5GHz in November of 2000. It implements the new Intel NetBurst microarchitecture that features significantly higher clock rates and world-class performance. It includes several important new features and innovations that will allow the Intel Pentium 4 processor to deliver industry-leading performance for the next several years. This paper provides an in-depth examination of the features and functions of the Intel NetBurst microarchitecture.

The Pentium 4 processor is designed to deliver performance across applications where end users can truly appreciate and experience its performance. For example, it allows a much better user experience in areas such as Internet audio and streaming video, image processing, video content creation, speech recognition, 3D applications and games, multi-media, and multi-tasking user environments. The Pentium 4 processor enables real-time MPEG2 video encoding and near real-time MPEG4 encoding, allowing efficient video editing and video conferencing. It delivers world-class performance on 3D applications and games, such as Quake 3*, enabling a new level of realism and visual quality to 3D applications.

The Pentium 4 processor has 42 million transistors implemented on Intel's 0.18u CMOS process, with six levels of aluminum interconnect. It has a die size of 217 mm^2 and it consumes 55 watts of power at 1.5GHz. Its 3.2 GB/second system bus helps provide the high data bandwidths needed to supply data to today's and tomorrow's demanding applications. It adds 144 new 128-bit Single Instruction Multiple Data (SIMD) instructions called SSE2 (Streaming SIMD Extension 2) that improve performance for multi-media, content creation, scientific, and engineering applications.

*Other brands and names are the property of their respective owners.

OVERVIEW OF THE NETBURST™ MICROARCHITECTURE

A fast processor requires balancing and tuning of many microarchitectural features that compete for processor die cost and for design and validation efforts. Figure 1 shows the basic Intel NetBurst microarchitecture of the Pentium 4 processor. As you can see, there are four main sections: the in-order front end, the out-of-order execution engine, the integer and floating-point execution units, and the memory subsystem.

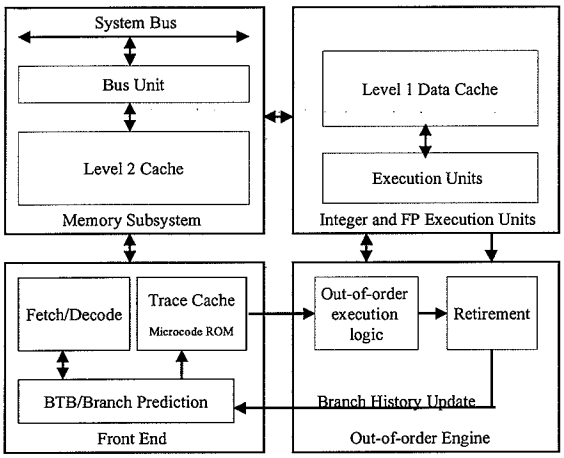

Figure 1: Basic block diagram

In-Order Front End

The in-order front end is the part of the machine that fetches the instructions to be executed next in the program and prepares them to be used later in the machine pipeline. Its job is to supply a high-bandwidth stream of decoded instructions to the out-of-order execution core, which will do the actual completion of the instructions. The front end has highly accurate branch prediction logic that uses the past history of program execution to speculate where the program is going to execute next. The predicted instruction address, from this front-end branch prediction logic, is used to fetch instruction bytes from the Level 2 (L2) cache. These IA-32 instruction bytes are then decoded into basic operations called uops (micro-operations) that the execution core is able to execute.

The NetBurst microarchitecture has an advanced form of a Level 1 (L1) instruction cache called the Execution Trace Cache. Unlike conventional instruction caches, the Trace Cache sits between the instruction decode logic and the execution core as shown in Figure 1. In this location the Trace Cache is able to store the already decoded IA-32 instructions or uops. Storing already decoded instructions removes the IA-32 decoding from the main execution loop. Typically the instructions are decoded once and placed in the Trace Cache and then used repeatedly from there like a normal instruction cache on previous machines. The IA-32 instruction decoder is only used when the machine misses the Trace Cache and needs to go to the L2 cache to get and decode new IA-32 instruction bytes.

Out-of-Order Execution Logic

The out-of-order execution engine is where the instructions are prepared for execution. The out-of-order execution logic has several buffers that it uses to smooth and re-order the flow of instructions to optimize performance as they go down the pipeline and get scheduled for execution. Instructions are aggressively re-ordered to allow them to execute as quickly as their input operands are ready. This out-of-order execution allows instructions in the program following delayed instructions to proceed around them as long as they do not depend on those delayed instructions. Out-of-order execution allows the execution resources such as the ALUs and the cache to be kept as busy as possible executing independent instructions that are ready to execute.

The retirement logic is what reorders the instructions, executed in an out-of-order manner, back to the original program order. This retirement logic receives the completion status of the executed instructions from the execution units and processes the results so that the proper architectural state is committed (or retired) according to the program order. The Pentium 4 processor can retire up to three uops per clock cycle. This retirement logic ensures that exceptions occur only if the operation causing the exception is the oldest, non-retired operation in the machine. This logic also reports branch history information to the branch predictors at the front end of the machine so they can train with the latest known-good branch-history information.

Integer and Floating-Point Execution Units

The execution units are where the instructions are actually executed. This section includes the register files that store the integer and floating-point data operand values that the instructions need to execute. The execution units include several types of integer and floating-point execution units that compute the results and also the L1 data cache that is used for most load and store operations.

Memory Subsystem

Figure 1 also shows the memory subsystem. This includes the L2 cache and the system bus. The L2 cache stores both instructions and data that cannot fit in the Execution Trace Cache and the L1 data cache. The external system bus is connected to the backside of the second-level cache and is used to access main memory when the L2 cache has a cache miss, and to access the system I/O resources.

CLOCK RATES

Processor microarchitectures can be pipelined to different degrees. The degree of pipelining is a microarchitectural decision. The final frequency of a specific processor pipeline on a given silicon process technology depends heavily on how deeply the processor is pipelined. When designing a new processor, a key design decision is the target design frequency of operation. The frequency target determines how many gates of logic can be included per pipeline stage in the design. This then helps determine how many pipeline stages there are in the machine.

There are tradeoffs when designing for higher clock rates. Higher clock rates need deeper pipelines so the efficiency at the same clock rate goes down. Deeper pipelines make many things take more clock cycles, such as mispredicted branches and cache misses, but usually more than make up for the lower per-clock efficiency by allowing the design to run at a much higher clock rate. For example, a 50% increase in frequency might buy only a 30% increase in net performance, but this frequency increase still provides a significant overall performance increase. High-frequency design also depends heavily on circuit design techniques, design methodology, design tools, silicon process technology, power and thermal constraints, etc. At higher frequencies, clock skew and jitter and latch delay become a much bigger percentage of the clock cycle, reducing the percentage of the clock cycle usable by actual logic. The deeper pipelines make the machine more complicated and require it to have deeper buffering to cover the longer pipelines.

Historical Trend of Processor Frequencies

Figure 2 shows the relative clock frequency of Intel's last six processor cores. The vertical axis shows the relative clock frequency, and the horizontal axis shows the various processors relative to each other.

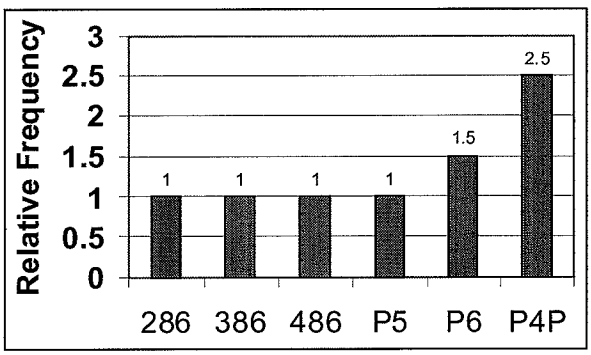

Figure 2: Relative frequencies of Intel's processors

Figure 2 shows that the 286, Intel386™, Intel486™ and Pentium® (P5) processors had similar pipeline depths–they would run at similar clock rates if they were all implemented on the same silicon process technology. They all have a similar number of gates of logic per clock cycle. The P6 microarchitecture lengthened the processor pipelines, allowing fewer gates of logic per pipeline stage, which delivered significantly higher frequency and performance. The P6 microarchitecture approximately doubled the number of pipeline stages compared to the earlier processors and was able to achieve about a 1.5 times higher frequency on the same process technology.

The NetBurst microarchitecture was designed to have an even deeper pipeline (about two times the P6 microarchitecture) with even fewer gates of logic per clock cycle to allow an industry-leading clock rate. Compared to the P6 family of processors, the Pentium 4 processor was designed with a greater than 1.6 times higher frequency target for its main clock rate, on the same process technology. This allows it to operate at a much higher frequency than the P6 family of processors on the same silicon process technology. At its introduction in November 2000, the Pentium 4 processor was at 1.5 times the frequency of the Pentium III processor. Over time this frequency delta will increase as the Pentium 4 processor design matures.

Different parts of the Pentium 4 processor run at different clock frequencies. The frequency of each section of logic is set to be appropriate for the performance it needs to achieve. The highest frequency section (fast clock) was set equal to the speed of the critical ALU-bypass execution loop that is used for most instructions in integer programs. Most other parts of the chip run at half of the 3GHz fast clock since this makes these parts much easier to design. A few sections of the chip run at a quarter of this fast-clock frequency making them also easier to design. The bus logic runs at 100MHz, to match the system bus needs.

As an example of the pipelining differences, Figure 3 shows a key pipeline in both the P6 and the Pentium 4 processors: the mispredicted branch pipeline. This pipeline covers the cycles it takes a processor to recover from a branch that went a different direction than the early fetch hardware predicted at the beginning of the machine pipeline. As shown, the Pentium 4 processor has a 20-stage misprediction pipeline while the P6 microarchitecture has a 10-stage misprediction pipeline. By dividing the pipeline into smaller pieces, doing less work during each pipeline stage (fewer gates of logic), the clock rate can be a lot higher.

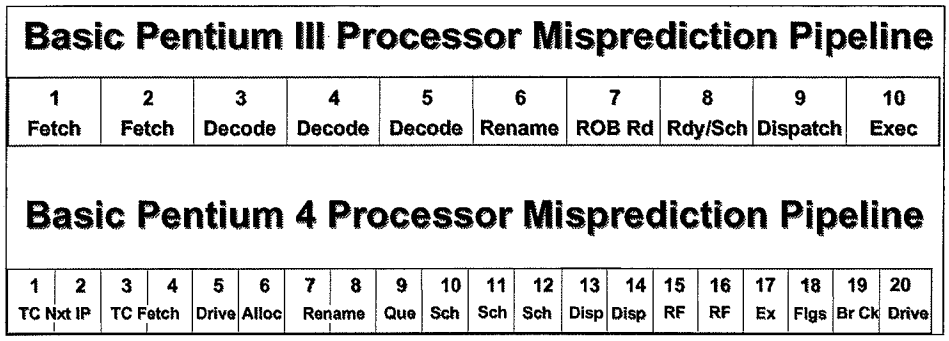

Figure 3: Misprediction Pipeline

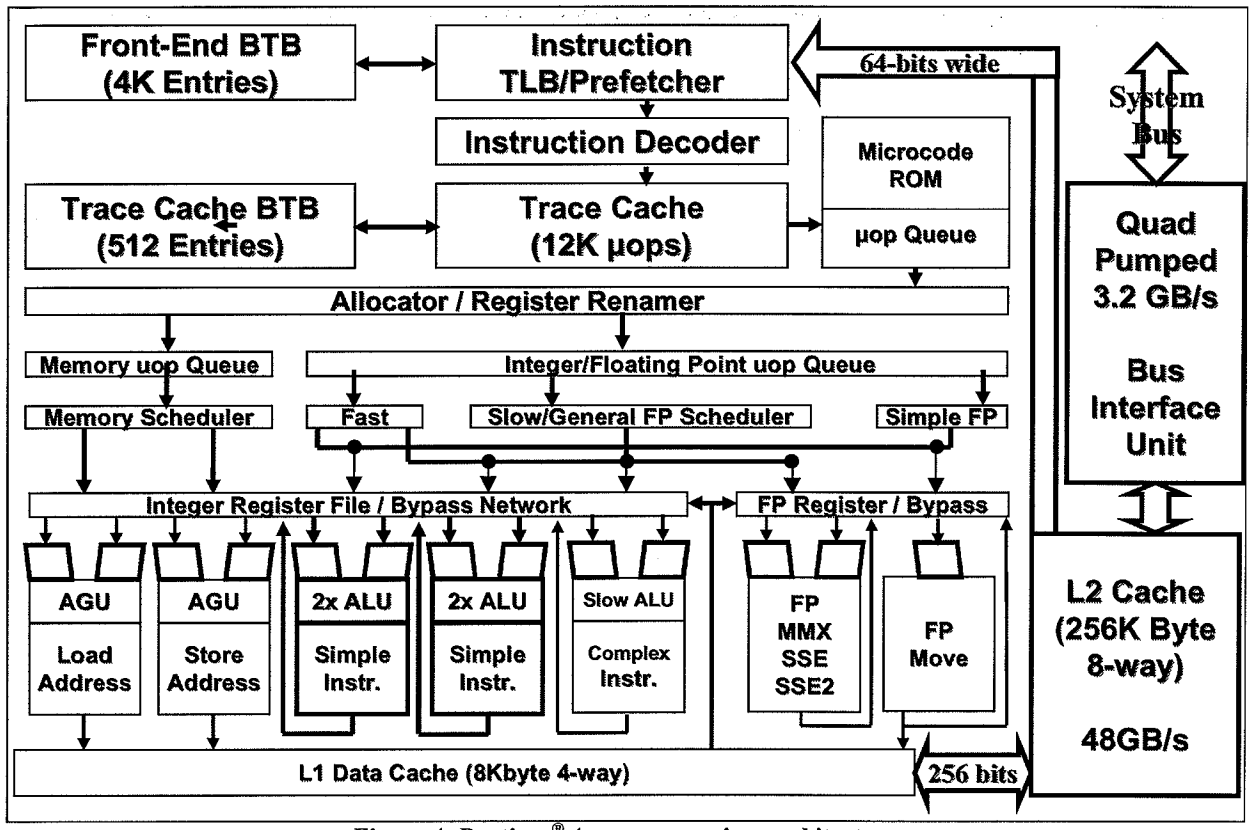

Figure 4: Pentium® 4 processor microarchitecture

NETBURST MICROARCHITECTURE

Figure 4 shows a more detailed block diagram of the NetBurst microarchitecture of the Pentium 4 processor. The top-left portion of the diagram shows the front end of the machine. The middle of the diagram illustrates the out-of-order buffering logic, and the bottom of the diagram shows the integer and floating-point execution units and the L1 data cache. On the right of the diagram is the memory subsystem.

Front End

The front end of the Pentium 4 processor consists of several units as shown in the upper part of Figure 4. It has the Instruction TLB (ITLB), the front-end branch predictor (labeled here Front-End BTB), the IA-32 Instruction Decoder, the Trace Cache, and the Microcode ROM.

Trace Cache

The Trace Cache is the primary or Level 1 (L1) instruction cache of the Pentium 4 processor and delivers up to three uops per clock to the out-of-order execution logic. Most instructions in a program are fetched and executed from the Trace Cache. Only when there is a Trace Cache miss does the NetBurst microarchitecture fetch and decode instructions from the Level 2 (L2) cache. This occurs about as often as previous processors miss their L1 instruction cache. The Trace Cache has a capacity to hold up to 12K uops. It has a similar hit rate to an 8K to 16K byte conventional instruction cache.

IA-32 instructions are cumbersome to decode. The instructions have a variable number of bytes and have many different options. The instruction decoding logic needs to sort this all out and convert these complex instructions into simple uops that the machine knows how to execute. This decoding is especially difficult when trying to decode several IA-32 instructions each clock cycle when running at the high clock frequency of the Pentium 4 processor. A high-bandwidth IA-32 decoder, that is capable of decoding several instructions per clock cycle, takes several pipeline stages to do its work. When a branch is mispredicted, the recovery time is much shorter if the machine does not have to re-decode the IA-32 instructions needed to resume execution at the corrected branch target location. By caching the uops of the previously decoded instructions in the Trace Cache, the NetBurst microarchitecture bypasses the instruction decoder most of the time thereby reducing misprediction latency and allowing the decoder to be simplified: it only needs to decode one IA-32 instruction per clock cycle.

The Execution Trace Cache takes the already-decoded uops from the IA-32 Instruction Decoder and assembles or builds them into program-ordered sequences of uops called traces. It packs the uops into groups of six uops per trace line. There can be many trace lines in a single trace. These traces consist of uops running sequentially down the predicted path of the IA-32 program execution. This allows the target of a branch to be included in the same trace cache line as the branch itself even if the branch and its target instructions are thousands of bytes apart in the program.

Conventional instruction caches typically provide instructions up to and including a taken branch instruction but none after it during that clock cycle. If the branch is the first instruction in a cache line, only the single branch instruction is delivered that clock cycle. Conventional instruction caches also often add a clock delay getting to the target of the taken branch, due to delays getting through the branch predictor and then accessing the new location in the instruction cache. The Trace Cache avoids both aspects of this instruction delivery delay for programs that fit well in the Trace Cache.

The Trace Cache has its own branch predictor that directs where instruction fetching needs to go next in the Trace Cache. This Trace Cache predictor (labeled Trace BTB in Figure 4) is smaller than the front-end predictor, since its main purpose is to predict the branches in the subset of the program that is currently in the Trace Cache. The branch prediction logic includes a 16-entry return address stack to efficiently predict return addresses, because often the same procedure is called from several different call sites. The Trace-Cache BTB, together with the front-end BTB, use a highly advanced branch prediction algorithm that reduces the branch misprediction rate by about 1/3 compared to the predictor in the P6 microarchitecture.

Microcode ROM

Near the Trace Cache is the microcode ROM. This ROM is used for complex IA-32 instructions, such as string move, and for fault and interrupt handling. When a complex instruction is encountered, the Trace Cache jumps into the microcode ROM which then issues the uops needed to complete the operation. After the microcode ROM finishes sequencing uops for the current IA-32 instruction, the front end of the machine resumes fetching uops from the Trace Cache.

The uops that come from the Trace Cache and the microcode ROM are buffered in a simple, in-order uop queue that helps smooth the flow of uops going to the out-of-order execution engine.

ITLB and Front-End BTB

The IA-32 Instruction TLB and front-end BTB, shown at the top of Figure 4, steer the front end when the machine misses the Trace Cache. The ITLB translates the linear instruction pointer addresses given to it into physical addresses needed to access the L2 cache. The ITLB also performs page-level protection checking.

Hardware instruction prefetching logic associated with the front-end BTB fetches IA-32 instruction bytes from the L2 cache that are predicted to be executed next. The fetch logic attempts to keep the instruction decoder fed with the next IA-32 instructions the program needs to execute. This instruction prefetcher is guided by the branch prediction logic (branch history table and branch target buffer listed here as the front-end BTB) to know what to fetch next. Branch prediction allows the processor to begin fetching and executing instructions long before the previous branch outcomes are certain. The front-end branch predictor is quite large–4K branch target entries–to capture most of the branch history information for the program. If a branch is not found in the BTB, the branch prediction hardware statically predicts the outcome of the branch based on the direction of the branch displacement (forward or backward). Backward branches are assumed

to be taken and forward branches are assumed to not be taken.

IA-32 Instruction Decoder

The instruction decoder receives IA-32 instruction bytes from the L2 cache 64-bits at a time and decodes them into primitives, called uops, that the machine knows how to execute. This single instruction decoder can decode at a maximum rate of one IA-32 instruction per clock cycle. Many IA-32 instructions are converted into a single uop, and others need several uops to complete the full operation. If more than four uops are needed to complete an IA-32 instruction, the decoder sends the machine into the microcode ROM to do the instruction. Most instructions do not need to jump to the microcode ROM to complete. An example of a many-uop instruction is string move, which could have thousands of uops.

Out-of-Order Execution Logic

The out-of-order execution engine consists of the allocation, renaming, and scheduling functions. This part of the machine re-orders instructions to allow them to execute as quickly as their input operands are ready.

The processor attempts to find as many instructions as possible to execute each clock cycle. The out-of-order execution engine will execute as many ready instructions as possible each clock cycle, even if they are not in the original program order. By looking at a larger number of instructions from the program at once, the out-of-order execution engine can usually find more ready-to-execute, independent instructions to begin. The NetBurst microarchitecture has much deeper buffering than the P6 microarchitecture to allow this. It can have up to 126 instructions in flight at a time and have up to 48 loads and 24 stores allocated in the machine at a time.

The Allocator

The out-of-order execution engine has several buffers to perform its re-ordering, tracking, and sequencing operations. The Allocator logic allocates many of the key machine buffers needed by each uop to execute. If a needed resource, such as a register file entry, is unavailable for one of the three uops coming to the Allocator this clock cycle, the Allocator will stall this part of the machine. When the resources become available the Allocator assigns them to the requesting uops and allows these satisfied uops to flow down the pipeline to be executed. The Allocator allocates a Reorder Buffer (ROB) entry, which tracks the completion status of one of the 126 uops that could be in flight simultaneously in the machine. The Allocator also allocates one of the 128 integer or floating-point register entries for the result data value of the uop, and possibly a load or store buffer used to track one of the 48 loads or 24 stores in the machine pipeline. In addition, the Allocator allocates an entry in one of the two uop queues in front of the instruction schedulers.

Register Renaming

The register renaming logic renames the logical IA-32 registers such as EAX onto the processors 128-entry physical register file. This allows the small, 8-entry, architecturally defined IA-32 register file to be dynamically expanded to use the 128 physical registers in the Pentium 4 processor. This renaming process removes false conflicts caused by multiple instructions creating their simultaneous but unique versions of a register such as EAX. There could be dozens of unique instances of EAX in the machine pipeline at one time. The renaming logic remembers the most current version of each register, such as EAX, in the Register Alias Table (RAT) so that a new instruction coming down the pipeline can know where to get the correct current instance of each of its input operand registers.

As shown in Figure 5 the NetBurst microarchitecture allocates and renames the registers somewhat differently than the P6 microarchitecture. On the left of Figure 5, the P6 scheme is shown. It allocates the data result registers and the ROB entries as a single, wide entity with a data and a status field. The ROB data field is used to store the data result value of the uop, and the ROB status field is used to track the status of the uop as it is executing in the machine. These ROB entries are allocated and deallocated sequentially and are pointed to by a sequence number that indicates the relative age of these entries. Upon retirement, the result data is physically copied from the ROB data result field into the separate Retirement Register File (RRF). The RAT points to the current version of each of the architectural registers such as EAX. This current register could be in the ROB or in the RRF.

The NetBurst microarchitecture allocation scheme is shown on the right of Figure 5. It allocates the ROB entries and the result data Register File (RF) entries separately.

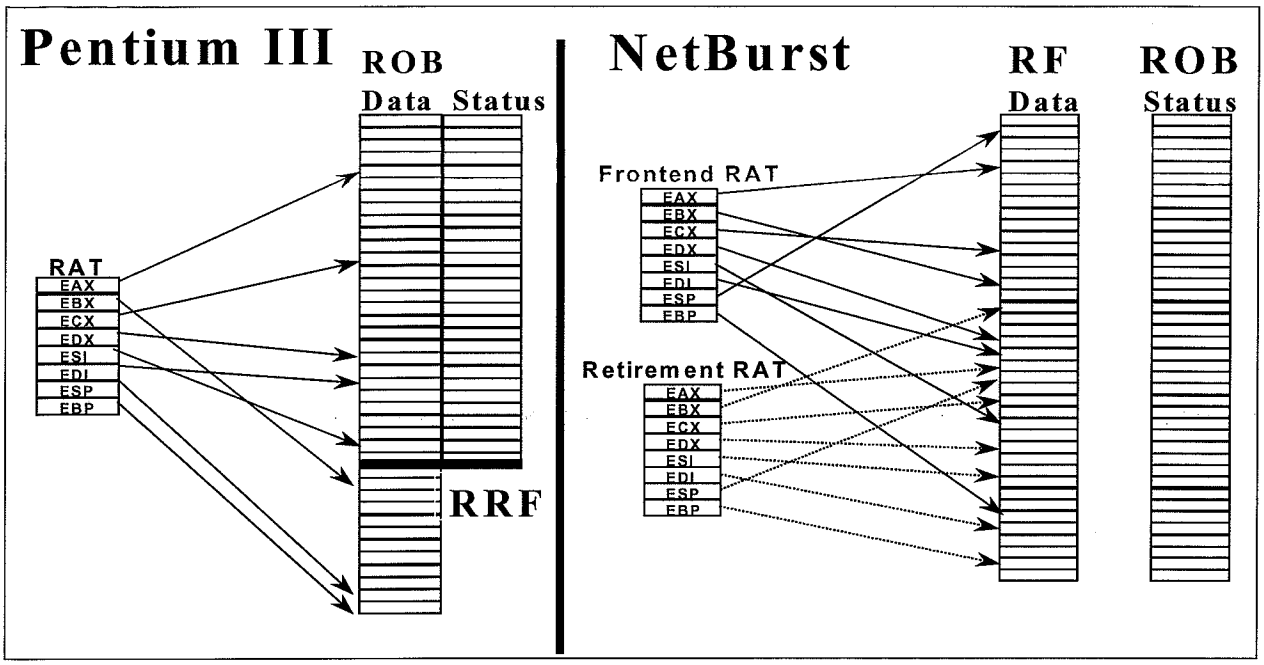

Figure 5: Pentium® III vs. Pentium® 4 processor register allocation

The ROB entries, which track uop status, consist only of the status field and are allocated and deallocated sequentially. A sequence number assigned to each uop indicates its relative age. The sequence number points to the uop's entry in the ROB array, which is similar to the P6 microarchitecture. The Register File entry is allocated from a list of available registers in the 128-entry RF–not sequentially like the ROB entries. Upon retirement, no result data values are actually moved from one physical structure to another.

Uop Scheduling

The uop schedulers determine when a uop is ready to execute by tracking its input register operands. This is the heart of the out-of-order execution engine. The uop schedulers are what allow the instructions to be reordered to execute as soon as they are ready, while still maintaining the correct dependencies from the original program. The NetBurst microarchitecture has two sets of structures to aid in uop scheduling: the uop queues and the actual uop schedulers.

There are two uop queues–one for memory operations (loads and stores) and one for non-memory operations. Each of these queues stores the uops in strict FIFO (first-in, first-out) order with respect to the uops in its own queue, but each queue is allowed to be read out-of-order with respect to the other queue. This allows the dynamic out-of-order scheduling window to be larger than just having the uop schedulers do all the reordering work.

There are several individual uop schedulers that are used to schedule different types of uops for the various execution units on the Pentium 4 processor as shown in Figure 6. These schedulers determine when uops are ready to execute based on the readiness of their dependent input register operand sources and the availability of the execution resources the uops need to complete their operation.

These schedulers are tied to four different dispatch ports. There are two execution unit dispatch ports labeled port 0 and port 1 in Figure 6. These ports are fast: they can dispatch up to two operations each main processor clock cycle. Multiple schedulers share each of these two dispatch ports. The fast ALU schedulers can schedule on each half of the main clock cycle while the other schedulers can only schedule once per main processor clock cycle. They arbitrate for the dispatch port when multiple schedulers have ready operations at once. There is also a load and a store dispatch port that can dispatch a ready load and store each clock cycle. Collectively, these uop dispatch ports can dispatch up to six uops each main clock cycle. This dispatch bandwidth exceeds the front-end and retirement bandwidth, of three uops per clock, to allow for peak bursts of greater than 3 uops per clock and to allow higher flexibility in issuing uops to different dispatch ports. Figure 6 also shows the types of operations that can be dispatched to each port each clock cycle.

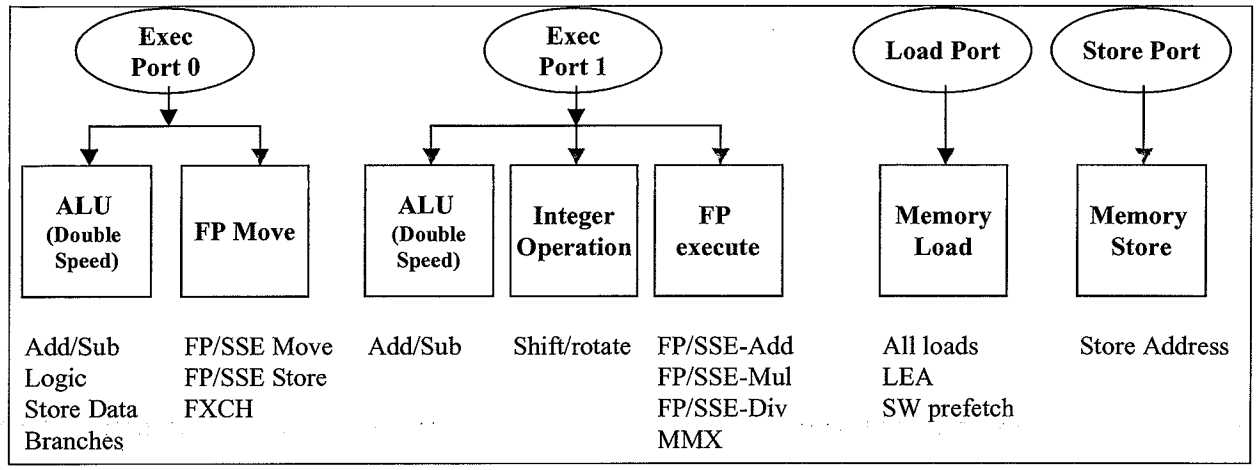

Figure 6: Dispatch ports in the Pentium® 4 processor

Integer and Floating-Point Execution Units

The execution units are where the instructions are actually executed. The execution units are designed to optimize overall performance by handling the most common cases as fast as possible. There are several different execution units in the NetBurst microarchitecture. The units used to execute integer operations include the low-latency integer ALUs, the complex integer instruction unit, the load and store address generation units, and the L1 data cache.

Floating-Point (x87), MMX, SSE (Streaming SIMD Extension), and SSE2 (Streaming SIMD Extension 2) operations are executed by the two floating-point execution blocks. MMX instructions are 64-bit packed integer SIMD operations that operate on 8, 16, or 32-bit operands. The SSE instructions are 128-bit packed IEEE single-precision floating-point operations. The Pentium 4 processor adds new forms of 128-bit SIMD instructions called SSE2. The SSE2 instructions support 128-bit packed IEEE double-precision SIMD floating-point operations and 128-bit packed integer SIMD operations. The packed integer operations support 8, 16, 32, and 64-bit operands. See IA-32 Intel Architecture Software Developer's Manual Volume 1: Basic Architecture [3] for more detail on these SIMD operations.

The Integer and floating-point register files sit between the schedulers and the execution units. There is a separate 128-entry register file for both the integer and the floating-point/SSE operations. Each register file also has a multi-clock bypass network that bypasses or forwards just-completed results, which have not yet been written into the register file, to the new dependent uops. This multi-clock bypass network is needed because of the very high frequency of the design.

Low Latency Integer ALU

The Pentium 4 processor execution units are designed to optimize overall performance by handling the most common cases as fast as possible. The Pentium 4 processor can do fully dependent ALU operations at twice the main clock rate. The ALU-bypass loop is a key closed loop in the processor pipeline. Approximately 60-70% of all uops in typical integer programs use this key integer ALU loop. Executing these operations at ½ the latency of the main clock helps speed up program execution for most programs. Doing the ALU operations in one half a clock cycle does not buy a 2x performance increase, but it does improve the performance for most integer applications.

This high-speed ALU core is kept as small as possible to minimize the metal length and loading. Only the essential hardware necessary to perform the frequent ALU operations is included in this high-speed ALU execution loop. Functions that are not used very frequently, for most integer programs, are not put in this key low-latency ALU loop but are put elsewhere. Some examples of integer execution hardware put elsewhere are the multiplier, shifts, flag logic, and branch processing.

The processor does ALU operations with an effective latency of one-half of a clock cycle. It does this operation in a sequence of three fast clock cycles (the fast clock runs at 2x the main clock rate) as shown in Figure 7. In the first fast clock cycle, the low order 16-bits are computed and are immediately available to feed the low 16-bits of a dependent operation the very next fast clock cycle. The high-order 16 bits are processed in the next fast cycle, using the carry out just generated by the low 16-bit operation. This upper 16-bit result will be available to the next dependent operation exactly when needed. This is called a staggered add. The ALU flags

are processed in the third fast cycle. This staggered add means that only a 16-bit adder and its input muxes need to be completed in a fast clock cycle. The low order 16 bits are needed at one time in order to begin the access of the L1 data cache when used as an address input.

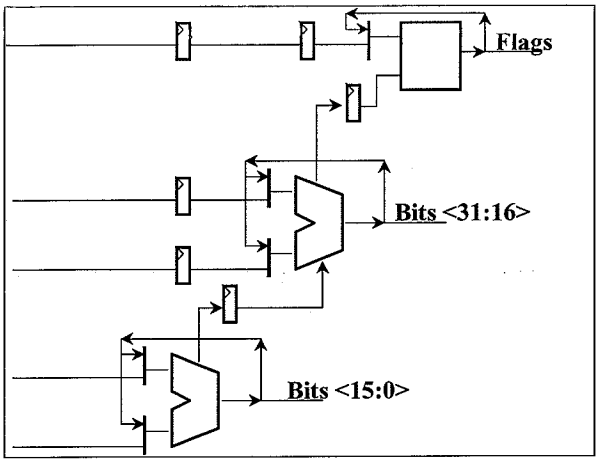

Figure 7: Staggered ALU add

Complex Integer Operations

The simple, very frequent ALU operations go to the high-speed integer ALU execution units described above. Integer operations that are more complex go to separate hardware for completion. Most integer shift or rotate operations go to the complex integer dispatch port. These shift operations have a latency of four clocks. Integer multiply and divide operations also have a long latency. Typical forms of multiply and divide have a latency of about 14 and 60 clocks, respectively.

Low Latency Level 1 (L1) Data Cache

The Level 1 (LI) data cache is an 8K-byte cache that is used for both integer and floating-point/SSE loads and stores. It is organized as a 4-way set-associative cache that has 64 bytes per cache line. It is a write-through cache, which means that writes to it are always copied into the L2 cache. It can do one load and one store per clock cycle.

The latency of load operations is a key aspect of processor performance. This is especially true for IA-32 programs that have a lot of loads and stores because of the limited number of registers in the instruction set. The NetBurst microarchitecture optimizes for the lowest overall load-access latency with a small, very low latency 8K byte cache backed up by a large, high-bandwidth second-level cache with medium latency. For most IA-32 programs this configuration of a small, but very low latency, L1 data cache followed by a large medium-latency L2 cache gives lower net load-access latency and therefore higher performance than a bigger, slower L1 cache. The L1 data cache operates with a 2-clock load-use latency for integer loads and a 6-clock load-use latency for floating-point/SSE loads.

This 2-clock load latency is hard to achieve with the very high clock rates of the Pentium 4 processor. This cache uses new access algorithms to enable this very low load-access latency. The new algorithm leverages the fact that almost all accesses hit the first-level data cache and the data TLB (DTLB).

At this high frequency and with this deep machine pipeline, the distance in clocks, from the load scheduler to execution, is longer than the load execution latency itself. The uop schedulers dispatch dependent operations before the parent load has finished executing. In most cases, the scheduler assumes that the load will hit the L1 data cache. If the load misses the L1 data cache, there will be dependent operations in flight in the pipeline. These dependent operations that have left the scheduler will get temporarily incorrect data. This is a form of data speculation. Using a mechanism known as *replay*, logic tracks and re-executes instructions that use incorrect data. Only the dependent operations are replayed: the independent ones are allowed to complete.

There can be up to four outstanding load misses from the L1 data cache pending at any one time in the memory subsystem.

Store-to-Load Forwarding

In an out-of-order-execution processor, stores are not allowed to be committed to permanent machine state (the L1 data cache, etc.) until after the store has retired. Waiting until retirement means that all other preceding operations have completely finished. All faults, interrupts, mispredicted branches, etc. must have been signaled beforehand to make sure this store is safe to perform. With the very deep pipeline of the Pentium 4 processor it takes many clock cycles for a store to make it to retirement. Also, stores that are at retirement often have to wait for previous stores to complete their update of the data cache. This machine can have up to 24 stores in the pipeline at a time. Sometimes many of them have retired but have not yet committed their state into the L1 data cache. Other stores may have completed, but have not yet retired, so their results are also not yet in the L1 data cache. Often loads must use the result of one of these pending stores, especially for IA-32 programs, due to the limited number of registers available. To enable this use of pending stores, modern out-of-order execution processors have a pending store buffer that allows loads to use the pending store results before the stores have been

written into the L1 data cache. This process is called store-to-load forwarding.

To make this store-to-load-forwarding process efficient, this pending store buffer is optimized to allow efficient and quick forwarding of data to dependent loads from the pending stores. The Pentium 4 processor has a 24-entry store-forwarding buffer to match the number of stores that can be in flight at once. This forwarding is allowed if a load hits the same address as a proceeding, completed, pending store that is still in the store-forwarding buffer. The load must also be the same size or smaller than the pending store and have the same beginning physical address as the store, for the forwarding to take place. This is by far the most common forwarding case. If the bytes requested by a load only partially overlap a pending store or need to have some bytes come simultaneously from more than one pending store, this store-to-load forwarding is not allowed. The load must get its data from the cache and cannot complete until the store has committed its state to the cache.

This disallowed store-to-load forwarding case can be quite costly, in terms of performance loss, if it happens very often. When it occurs, it tends to happen on older P5-core optimized applications that have not been optimized for modern, out-of-order execution microarchitectures. The newer versions of the IA-32 compilers remove most or all of these bad store-to-load forwarding cases but they are still found in many old legacy P5 optimized applications and benchmarks. This bad store-forwarding case is a big performance issue for P6-based processors and other modern processors, but due to the even deeper pipeline of the Pentium 4 processor, these cases are even more costly in performance.

FP/SSE Execution Units

The Floating-Point (FP) execution cluster of the Pentium 4 processor is where the floating-point, MMX, SSE, and SSE2 instructions are executed. These instructions typically have operands from 64 to 128 bits in width. The FP/SSE register file has 128 entries and each register is 128 bits wide. This execution cluster has two 128-bit execution ports that can each begin a new operation every clock cycle. One execution port is for 128-bit general execution and one is for 128-bit register-to-register moves and memory stores. The FP/SSE engine can also complete a full 128-bit load each clock cycle.

Early in the development cycle of the Pentium 4 processor, we had two full FP/SSE execution units, but this cost a lot of hardware and did not buy very much performance for most FP/SSE applications. Instead, we optimized the cost/performance tradeoff with a simple second port that does FP/SSE moves and FP/SSE store data primitives. This tradeoff was shown to buy most of the performance of a second full-featured port with much less die size and power cost.

Many FP/multi-media applications have a fairly balanced set of multiplies and adds. The machine can usually keep busy interleaving a multiply and an add every two clock cycles at much less cost than fully pipelining all the FP/SSE execution hardware. In the Pentium 4 processor, the FP adder can execute one Extended-Precision (EP) addition, one Double-Precision (DP) addition, or two Single-Precision (SP) additions every clock cycle. This allows it to complete a 128-bit SSE/SSE2 packed SP or DP add uop every two clock cycles. The FP multiplier can execute either one EP multiply every two clocks, or it can execute one DP multiply or two SP multiplies every clock. This allows it to complete a 128-bit IEEE SSE/SSE2 packed SP or DP multiply uop every two clock cycles giving a peak 6 GFLOPS for single precision or 3 GFLOPS for double precision floating-point at 1.5GHz.

Many multi-media applications interleave adds, multiplies, and pack/unpack/shuffle operations. For integer SIMD operations, which are the 64-bit wide MMX or 128-bit wide SSE2 instructions, there are three execution units that can run in parallel. The SIMD integer ALU execution hardware can process 64 SIMD integer bits per clock cycle. This allows the unit to do a new 128-bit SSE2 packed integer add uop every two clock cycles. A separate shuffle/unpack execution unit can also process 64 SIMD integer bits per clock cycle allowing it to do a full 128-bit shuffle/unpack uop operation each two clock cycles. MMX/SSE2 SIMD integer multiply instructions use the FP multiply hardware mentioned above to also do a 128-bit packed integer multiply uop every two clock cycles.

The FP divider executes all divide, square root, and remainder uops. It is based on a double-pumped SRT radix-2 algorithm, producing two bits of quotient (or square root) every clock cycle.

Achieving significantly higher floating-point and multi-media performance requires much more than just fast execution units. It requires a balanced set of capabilities that work together. These programs often have many long latency operations in their inner loops. The very deep buffering of the Pentium 4 processor (126 uops and 48 loads in flight) allows the machine to examine a large section of the program at once. The out-of-order-execution hardware often unrolls the inner execution loop of these programs numerous times in its execution window. This dynamic unrolling allows the Pentium 4 processor to overlap the long-latency FP/SSE and memory instructions by finding many independent instructions to work on simultaneously. This deep window buys a lot more performance for most FP/multi-media applications than more execution units would.

FP/multi-media applications usually need a very high bandwidth memory subsystem. Sometimes FP and multi-media applications do not fit well in the L1 data cache but do fit in the L2 cache. To optimize these applications the Pentium 4 processor has a high bandwidth path from the L2 data cache to the L1 data. Some FP/multi-media applications stream data from memory–no practical cache size will hold the data. They need a high bandwidth path to main memory to perform well. The long 128-byte L2 cache lines together with the hardware prefetcher described below help to prefetch the data that the application will soon need, effectively hiding the long memory latency. The high bandwidth system bus of the Pentium 4 processor allows this prefetching to help keep the execution engine well fed with streaming data.

Memory Subsystem

The Pentium 4 processor has a highly capable memory subsystem to enable the new, emerging, high-bandwidth stream-oriented applications such as 3D, video, and content creation. The memory subsystem includes the Level 2 (L2) cache and the system bus. The L2 cache stores data that cannot fit in the Level 1 (L1) caches. The external system bus is used to access main memory when the L2 cache has a cache miss and also to access the system I/O devices.

Level 2 Instruction and Data Cache

The L2 cache is a 256K-byte cache that holds both instructions that miss the Trace Cache and data that miss the L1 data cache. The L2 cache is organized as an 8-way set-associative cache with 128 bytes per cache line. These 128-byte cache lines consist of two 64-byte sectors. A miss in the L2 cache typically initiates two 64-byte access requests to the system bus to fill both halves of the cache line. The L2 cache is a write-back cache that allocates new cache lines on load or store misses. It has a net load-use access latency of seven clock cycles. A new cache operation can begin every two processor clock cycles for a peak bandwidth of 48Gbytes per second, when running at 1.5GHz.

Associated with the L2 cache is a hardware prefetcher that monitors data access patterns and prefetches data automatically into the L2 cache. It attempts to stay 256 bytes ahead of the current data access locations. This prefetcher remembers the history of cache misses to detect concurrent, independent streams of data that it tries to prefetch ahead of use in the program. The prefetcher also tries to minimize prefetching unwanted data that can cause over utilization of the memory system and delay the real accesses the program needs.

400MHz System Bus

The Pentium 4 processor has a system bus with 3.2 Gbytes per second of bandwidth. This high bandwidth is a key enabler for applications that stream data from memory. This bandwidth is achieved with a 64-bit wide bus capable of transferring data at a rate of 400MHz. It uses a source-synchronous protocol that quad-pumps the 100MHz bus to give 400 million data transfers per second. It has a split-transaction, deeply pipelined protocol to allow the memory subsystem to overlap many simultaneous requests to actually deliver high memory bandwidths in a real system. The bus protocol has a 64-byte access length.

PERFORMANCE

The Pentium 4 processor delivers the highest SPECint_base performance of any processor in the world. It also delivers world-class SPECfp2000 performance. These are industry standard benchmarks that evaluate general integer and floating-point application performance.

Figure 8 shows the performance comparison of a Pentium 4 processor at 1.5GHz compared to a Pentium III processor at 1GHz for various applications. The integer applications are in the 15-20% performance gain while the FP and multi-media applications are in the 30-70% performance advantage range. For FSPEC 2000 the new SSE/SSE2 instructions buy about 5% performance gain compared to an x87-only version. As the compiler improves over time the gain from these new instructions will increase. Also, as the relative frequency of the Pentium 4 processor increases over time (as its design matures), all these performance deltas will increase.

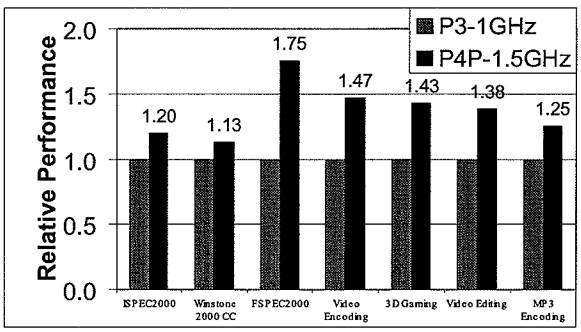

Figure 8: Performance comparison

For a more complete performance brief covering many application performance areas on the Pentium 4 processor, go to http://www.intel.com/procs/perf/pentium4/.

CONCLUSION

The Pentium 4 processor is a new, state-of-the-art

processor microarchitecture and design. It is the beginning of a new family of processors that utilize the new Intel NetBurst microarchitecture. Its deeply pipelined design delivers world-leading frequencies and performance. It uses many novel microarchitectural ideas including a Trace Cache, double-clocked ALU, new low-latency L1 data cache algorithms, and a new high bandwidth system bus. It delivers world-class performance in the areas where added performance makes a difference including media rich environments (video, sound, and speech), 3D applications, workstation applications, and content creation.

ACKNOWLEDGMENTS

The authors thank all the architects, designers, and validators who contributed to making this processor into a real product.

REFERENCES

1. D. Sager, G. Hinton, M. Upton, T. Chappell, T. Fletcher, S. Samaan, and R. Murray, "A 0.18um CMOS IA32 Microprocessor with a 4GHz Integer Execution Unit," International Solid State Circuits Conference, Feb 2001.
2. Doug Carmean, "Inside the High-Performance Intel® Pentium® 4 Processor Micro-architecture" Intel Developer Forum, Fall 2000 at ftp://download.intel.com/design/idf/fall2000/presentations/pda/pda_s01_cd.pdf
3. IA-32 Intel Architecture Software Developer's Manual Volume 1: Basic Architecture at http://developer.intel.com/design/pentium4/manuals/245470.htm.
4. Intel® Pentium® 4 Processor Optimization Reference Manual at http://developer.intel.com/design/pentium4/manuals/248966.htm.

AUTHORS' BIOGRAPHIES

Glenn Hinton is an Intel Fellow and Director of IA-32 Microarchitecture Development in the Intel Architecture Group. Hinton joined Intel in 1983. He was one of three senior architects in 1990 responsible for the P6 processor microarchitecture, which became the Pentium® Pro, Pentium® II, Pentium® III, and Celeron™ processors. He was responsible for the microarchitecture development of the Pentium® 4 processor. Hinton received a master's degree in Electrical Engineering from Brigham Young University in 1983. His e-mail address is glenn.hinton@intel.com.

Dave Sager is a Principal Engineer/Architect in Intel's Desktop Platforms Group, and is one of the overall architects of the Intel® Pentium 4 processor. He joined Intel in 1995. Dave also worked for 17 years at Digital Equipment Corporation in their processor research labs. He graduated from Princeton University with a Ph.D. in Physics in 1973. His e-mail address is dave.sager@intel.com.

Michael Upton is a Principal Engineer/Architect in Intel's Desktop Platforms Group, and is one of the architects of the Intel® Pentium 4 processor. He completed B.S. and M.S. degrees in Electrical Engineering from the University of Washington in 1985 and 1990. After a number of years in IC design and CAD tool development, he entered the University of Michigan to study computer architecture. Upon completion of his Ph.D degree in 1994, he joined Intel to work on the Pentium® Pro and Pentium 4 processors. His e-mail address is mike.upton@intel.com.

Darrell Boggs is a Principal Engineer/Architect with Intel Corporation and has been working as a microarchitect for nearly 10 years. He graduated from Brigham Young University with a M.S. in Electrical Engineering. Darrell played a key role on the Pentium® Pro Processor design, and was one of the key architects of the Pentium 4 Processor. Darrell holds many patents in the areas of register renaming; instruction decoding; events and state recovery mechanisms. His e-mail address is darrell.boggs@intel.com.

Douglas M. Carmean is a Principal Engineer/Architect with Intel's Desktop Products Group in Oregon. Doug was one of the key architects, responsible for definition of the Intel Pentium® 4 processor. He has been with Intel for 12 years, working on IA-32 processors from the 80486 to the Intel Pentium 4 processor and beyond. Prior to joining Intel, Doug worked at ROSS Technology, Sun Microsystems, Cypress Semiconductor and Lattice Semiconductor. Doug enjoys fast cars and scary, Italian motorcycles. His e-mail address is douglas.m.carmean@intel.com.

Patrice Roussel graduated from the University of Rennes in 1980 and L'Ecole Superieure d'Electricite in 1982 with a M.S. degree in signal processing and VLSI design. Upon graduation, he worked at Cimatel, an Intel/Matra Harris joint design center. He moved to the USA in 1988 to join Intel in Arizona and worked on the 960CA chip. In late 1991, he moved to Intel in Oregon to work on the P6 processors. Since 1995, he has been the floating-point architect of the Pentium® 4 processor. His e-mail address is patrice.roussel@intel.com.

Copyright © Intel Corporation 2001. This publication was downloaded from http://developer.intel.com/.

Legal notices at http://developer.intel.com/sites/developer/tradmarx.htm