INTEL PRESENTS P6 MICROARCHITECTURE DETAILS

Technical Paper Highlights Dynamic Execution Design

SAN FRANCISCO, Calif., Feb. 16, 1995 Intel Corporation today disclosed details of the first fruit of a parallel engineering effort, the next-generation P6 microprocessor, at an engineering conference here. The presentation of technical details follows the delivery of first working samples to OEMs.

The 5.5-million transistor chip will deliver the highest level of processor performance for the Intel Architecture when systems using the chip begin to ship in the second half of this year. P6 will achieve this performance using a unique combination of technologies known as Dynamic Execution.

P6 microarchitecture details were presented by Intel at the IEEE International Solid State Circuits Conference (ISSCC), an annual industry gathering where technical innovations are showcased and discussed. Details on P6 s unique approach to high-performance processing, described collectively as Dynamic Execution, were presented by Dr. Robert Colwell, P6 architecture manager, at ISSCC.

Colwell explained that this architectural enhancement is the next step beyond the superscalar advance implemented in the Pentium(R) processor. Dynamic Execution is a combination of technologies multiple branch prediction, data flow analysis and speculative execution that is constantly feeding P6 s data-crunching units. Intel engineers were able to implement Dynamic Execution by analyzing how billions of lines of code in software programs are typically executed by processors. Collectively, these technologies allow the P6 to operate as an efficient information factory.

Multiple branch prediction increases the amount of work available for the microprocessor to execute. Data flow analysis schedules the instructions to be executed when ready, independent of the original program order. Speculative execution allows the P6 to keep its superscalar engine as busy as possible by executing instructions that are likely to be needed.

With these technologies, the P6 can efficiently analyze much larger sections of incoming program flow than any previous PC processor, swiftly allocate internal resources, and intelligently optimize work that can be done in parallel. Consequently, more data can be processed in a given time period.

Parallel Design Teams Learn From Each Other

The concept of P6 s Dynamic Execution engine began in 1990, when today s mainstream Pentium(R) processor was still just a software simulation.

Intel s use of parallel engineering teams for chip design has compressed delivery cycles of new generations of chips, cutting the time about in half, said Albert Yu, senior vice president and general manager, Microprocessor Products Group. As a result, computer users will have some of the most powerful, low-cost engines at hand to enrich the desktop with software and other capabilities we only imagined five years ago, he said.

Yu said the Oregon-based P6 design team, building on the knowledge gained from the Pentium processor design, embarked on an innovative system-level solution to the next-generation processor involving the processor, cache (high-speed supporting memory), and bus (the transport mechanism that keeps data flowing into and out of the processor). This approach will ensure that computers built around P6 will be able to take advantage of the chip s processing power when it is introduced as a commercial product later this year, he said.

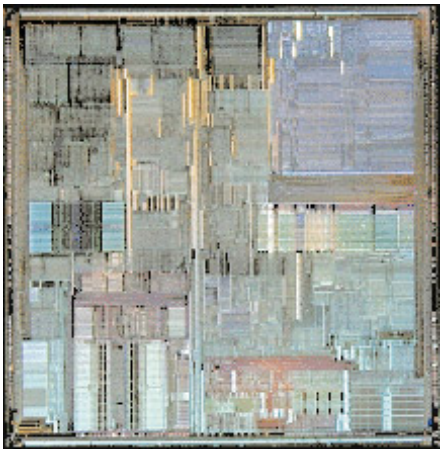
The system-level approach means the P6 will be the first high-volume microprocessor with two die in a single package. A dual-cavity, standard PGA package contains a P6 die and a companion level two (L2) cache die. The two chips communicate using a highly-optimized bus which contributes to high performance by tightly-coupling the processor to its primary data source.

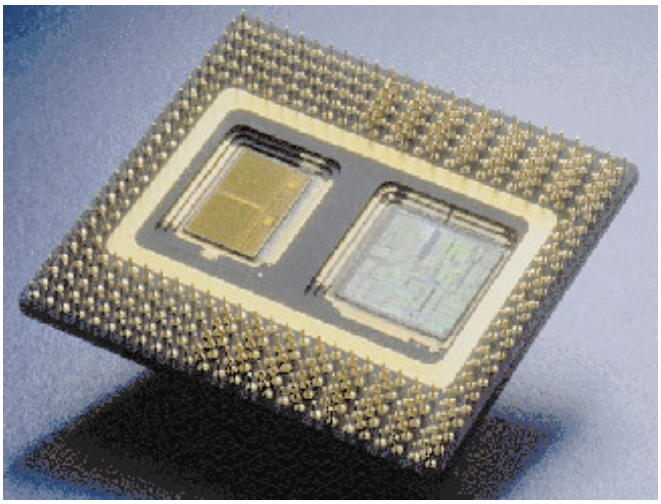
Additional Features

In addition to providing new levels of performance, the P6 will contain new features which will greatly simplify the design of multiprocessor systems and improve overall system reliability. Among applications that will benefit greatly from this processing power are: desktop applications such as image processing, speech recognition, software-only videoconferencing and multimedia authoring, and server applications such as transaction and database processing.

At introduction in the second half of this year, the P6 processor will operate at 133 MHz and will use a power supply of 2.9 volts. The low voltage also contributes to low power dissipation, which is expected to be only about 14 watts, typically, for the processor and L2 cache combined. Complete performance and power dissipation information will also be available at that time, although estimated performance has been measured at more than 200 SPECint92 on a prototype system, twice the performance of today s fastest Pentium processor.

Intel, the world s largest chip maker, is also a leading manufacturer of personal computer, networking and communications products.





P6 Processor Overview
Intel s P6 family of processors...

- o Ensures complete binary compatibility with previous generations of the Intel Architecture.
- o Delivers superior performance through an innovation called Dynamic Execution*.
- o Provides support for enhanced data integrity and reliability feaures:
  ECC (Error Checking and Correction), Fault Analysis & Recovery, and Functional Redundancy Checking.
- o Includes features that will greatly simplify the design of multiprocessing systems.

The first member of the P6 processor family...

- o Arrives in desktops and servers in 1995.
- o Integrates about 5.5 million transistors on the chip, compared to approximately 3.1 million transistors on the Pentium processor.
- o Will operate at 133 MHz with estimated performance at more than 200 SPECint92.
- o Will initially be produced on the same high volume 0.6 micron process currently used for the 90 and 100 MHz versions of the Pentium processor, and will then move to a 0.35 micron process.
- o Delivers performance that will scale up to 1000 MIPS with four processors.

*** What is Dynamic Execution?**
Dynamic Execution is the unique combination of three processing techniques the P6 uses to speed up software:

- o Multiple Branch prediction:
  First, the processor looks multiple steps ahead in the software and predicts which branches, or groups of instructions, are likely to be processed next. This increases the amount of work fed to the processor.

- o Dataflow analysis:
  Next, the P6 analyzes which instructions are dependent on each other s results, or data, to create an optimized schedule of instructions.

- o Speculative Execution:
  Instructions are then carried out speculatively based on this optimized schedule, keeping all the chip s superscalar processing power busy, and boosting overall software performance.

A Tour of the P6 Microarchitecture

Introduction

Achieving twice the performance of a Pentium® processor while being manufactured on the same semiconductor process was one of the P6's primary goals. Using the same process as a volume production processor practically assured that the P6 would be manufactureable, but it meant that Intel had to focus on an improved microarchitecture for ALL of the performance gains. This guided tour describes how multiple architectural techniques - some proven in mainframe computers, some proposed in academia and some we innovated ourselves - were carefully interwoven, modified, enhanced, tuned and implemented to produce the P6 microprocessor. This unique combination of architectural features, which Intel describes as Dynamic Execution, enabled the first P6 silicon to exceed the original performance goal.

Building from an already high platform

The Pentium processor set an impressive performance standard with its pipelined, superscalar microarchitecture. The Pentium processor's pipelined implementation uses five stages to extract high throughput from the silicon - the P6 moves to a decoupled, 12-stage, superpipelined implementation, trading less work per pipestage for more stages. The P6 reduced its pipestage time by 33 percent, compared with a Pentium processor, which means that from a semiconductor manufacturing process (i.e., transistor speed) perspective a 133MHz P6 and a 100MHz Pentium processor are equivalent

The Pentium processor's superscalar microarchitecture, with its ability to execute two instructions per clock, would be difficult to exceed without a new approach. The new approach used by the P6 removes the constraint of linear instruction sequencing between the traditional "fetch" and "execute" phases, and opens up a wide instruction window using an instruction pool. This approach allows the "execute" phase of the P6 to have much more visibility into the program's instruction stream so that better scheduling may take place. It requires the instruction "fetch/decode" phase of the P6 to be much more intelligent in terms of predicting program flow. Optimized scheduling requires the fundamental "execute" phase to be replaced by decoupled "dispatch/execute" and "retire" phases. This allows instructions to be started in any order but always be completed in the original program order. The P6 is implemented as three independent engines coupled with an instruction pool as shown in Figure 1.

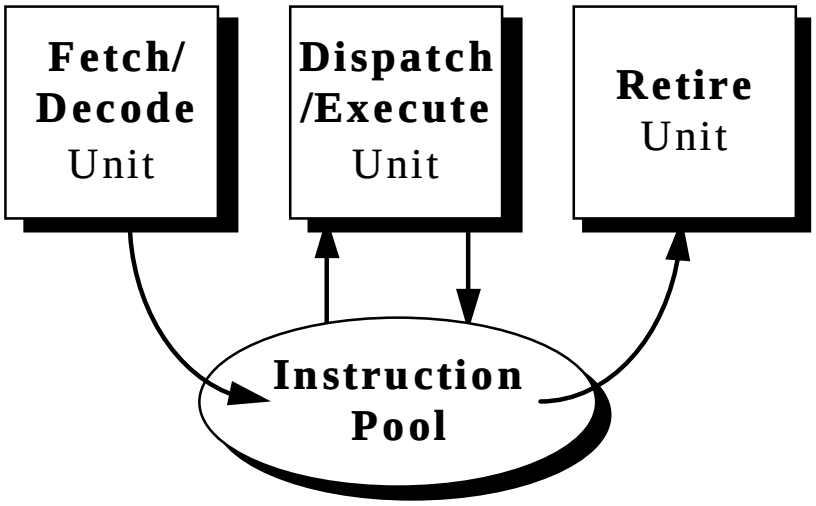


Figure 1. The P6 is implemented as three independent engines that communicate using an instruction pool.

What is the fundamental problem to solve?

Before starting our tour on how the P6 achieves its high performance it is important to note why this three-independent-engine approach was taken. A fundamental fact of today's microprocessor implementations must be appreciated: most CPU cores are not fully utilized. Consider the code fragment in Figure 2:

```
r1 <=  mem [r0]        /* Instruction 1 */
r2 <=  r1 + r2         /* Instruction 2 */
r5 <=  r5 + 1          /* Instruction 3 */
r6 <=  r6 - r3         /* Instruction 4 */
```

Figure 2. A typical code fragment.

The first instruction in this example is a load of r1 that, at run time, causes a cache miss. A traditional CPU core must wait for its bus interface unit to read this data from main memory and return it before moving on to instruction 2. This CPU stalls while waiting for this data and is thus being under-utilized.

While CPU speeds have increased 10-fold over the past 10 years, the speed of main memory devices has only increased by 60 percent. This increasing memory latency, relative to the CPU core speed, is a fundamental problem that the P6 set out to solve. One approach would be to place the burdon of this problem onto the chipset but a high-performance CPU that needs very high speed, specialized, support components is not a good solution for a volume production system.

A brute-force approach to this problem is, of course, increasing the size of the L2 cache to reduce the miss ratio. While effective, this is another expensive solution, especially considering the speed requirements of today's L2 cache SRAM components. Instead, the P6 is designed from an overall system implementation perspective which will allow higher performance systems to be designed with cheaper memory subsystem designs.

P6 takes an innovative approach

To avoid this memory latency problem the P6 "looks-ahead" into its instruction pool at subsequent instructions and will do useful work rather than be stalled. In the example in Figure 2, instruction 2 is not executable since it depends upon the result of instruction 1; however both instructions 3 and 4 are executable. The P6 speculatively executes instructions 3 and 4. We cannot commit the results of this speculative execution to permanent machine state (i.e., the programmer-visible registers) since we must maintain the original program order, so the results are instead stored back in the instruction pool awaiting in-order retirement. The core executes instructions depending upon their readiness to execute and not on their original program order (it is a true dataflow engine). This approach has the side effect that instructions are typically executed out-of-order.

The cache miss on instruction 1 will take many internal clocks, so the P6 core continues to look ahead for other instructions that could be speculatively executed and is typically looking 20 to 30 instructions in front of the program counter. Within this 20- to 30- instruction window there will be, on average, five branches that the fetch/decode unit must correctly predict if the dispatch/execute unit is to do useful work. The sparse register set of an Intel Architecture (IA) processor will create many false dependencies on registers so the dispatch/execute unit will rename the IA registers to enable additional forward progress. The retire unit owns the physical IA register set and results are only committed to permanent machine state when it removes completed instructions from the pool in original program order.

Dynamic Execution technology can be summarized as optimally adjusting instruction execution by predicting program flow, analysing the program's dataflow graph to choose the best order to execute the instructions, then having the ability to speculatively execute instructions in the preferred order. The P6 dynamically adjusts its work, as defined by the incoming instruction stream, to minimize overall execution time.

Overview of the stops on the tour

We have previewed how the P6 takes an innovative approach to overcome a key system constraint. Now let's take a closer look inside the P6 to understand how it implements Dynamic Execution. Figure 3 extends the basic block diagram to include the cache and memory interfaces - these will also be stops on our tour. We shall travel down the P6 pipeline to understand the role of each unit:

The FETCH/DECODE unit: An in-order unit that takes as input the user program instruction stream from the instruction cache, and decodes them into a series of micro-operations (uops) that represent the dataflow of that instruction stream. The program pre-fetch is itself speculative.

The DISPATCH/EXECUTE unit: An out-of-order unit that accepts the dataflow stream, schedules execution of the uops subject to data dependencies and resource availability and temporarily stores the results of these speculative executions.

The RETIRE unit: An in-order unit that knows how and when to commit ("retire") the temporary, speculative results to permanent architectural state.

The BUS INTERFACE unit: A partially ordered unit responsible for connecting the three internal units to the real world. The bus interface unit communicates directly with the L2 cache supporting up to four concurrent cache accesses. The bus interface unit also controls a transaction bus, with MESI snooping protocol, to system memory.

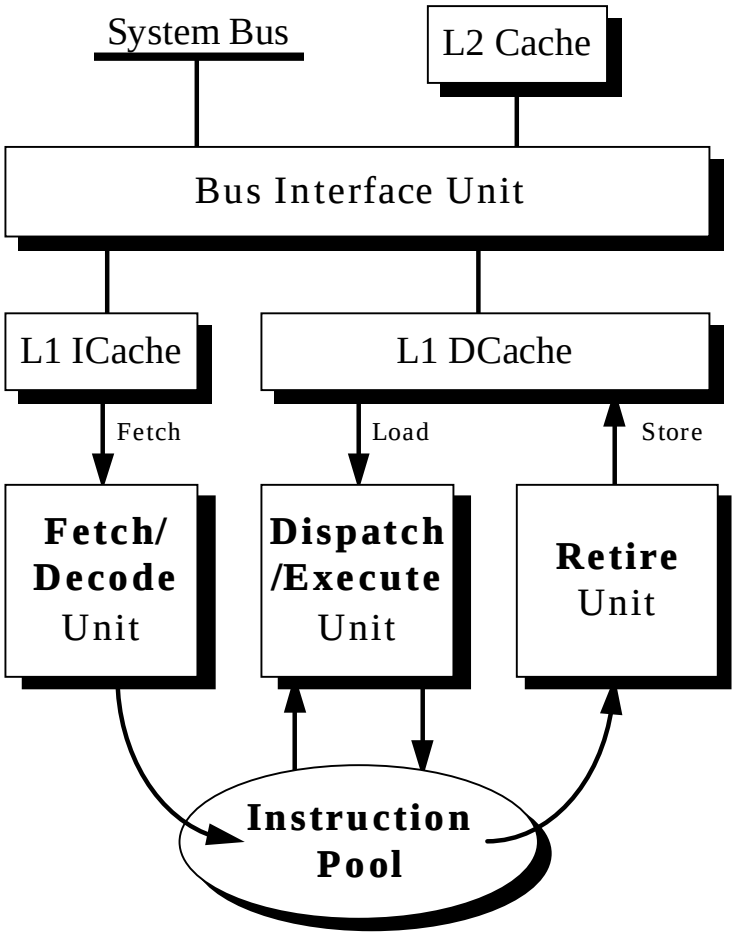


Figure 3. The three core engines interface with the memory subsystem using 8K/8K unified caches.

Tour stop #1: The FETCH/DECODE unit.

Figure 4 shows a more detailed view of the fetch/decode unit:

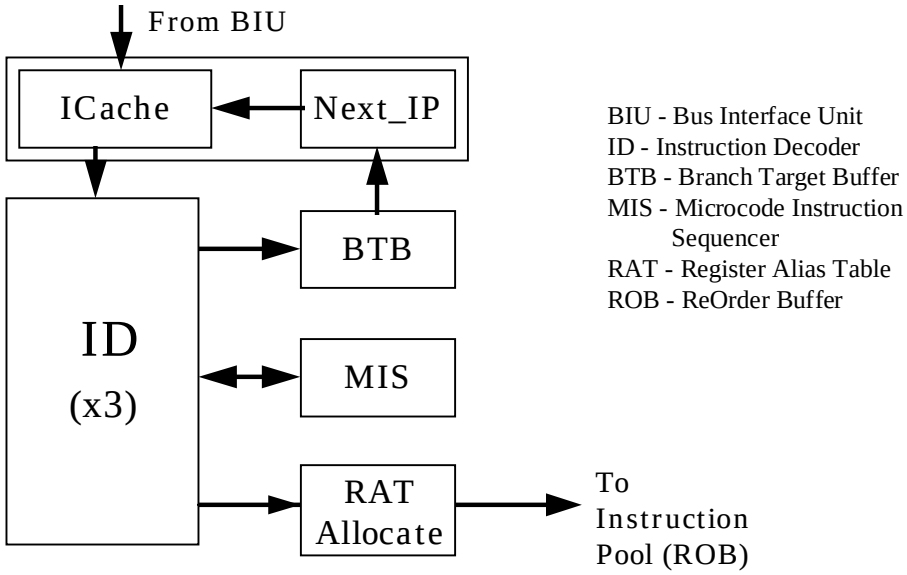


Figure 4: Looking inside the Fetch/Decode Unit

Let's start the tour at the ICache, a nearby place for instructions to reside so that they can be looked up quickly when the CPU needs them. The Next_IP unit provides the ICache index, based on inputs from the Branch Target Buffer (BTB), trap/interrupt status, and branch-misprediction indications from the integer execution section. The 512 entry BTB uses an extension of Yeh's algorithm to provide greater than 90 percent prediction accuracy. For now, let's assume that nothing exceptional is happening, and that the BTB is correct in its predictions. (The P6 integrates features that allow for the rapid recovery from a mis-prediction, but more of that later.)

The ICache fetches the cache line corresponding to the index from the Next_IP, and the next line, and presents 16 aligned bytes to the decoder. Two lines are read because the IA instruction stream is byte-aligned, and code often branches to the middle or end of a cache line. This part of the pipeline takes three clocks, including the time to rotate the prefetched bytes so that they are justified for the instruction decoders (ID). The beginning and end of the IA instructions are marked.

Three parallel decoders accept this stream of marked bytes, and proceed to find and decode the IA instructions contained therein. The decoder converts the IA instructions into triadic uops (two logical sources, one logical destination per uop). Most IA instructions are converted directly into single uops, some instructions are decoded into one-to-four uops and the complex instructions require microcode (the box labeled MIS in Figure 4, this microcode is just a set of preprogrammed sequences of normal uops). Some instructions, called prefix bytes, modify the following instruction giving the decoder a lot of work to do. The uops are enqueued, and sent to the Register Alias Table (RAT) unit, where the logical IA-based register references are converted into P6 physical register references, and to the Allocator stage, which adds status information to the uops and enters them into the instruction pool. The instruction pool is implemented as an array of Content Addressable Memory called the ReOrder Buffer (ROB).

We have now reached the end of the in-order pipe.

Tour stop #2: The DISPATCH/EXECUTE unit

The dispatch unit selects uops from the instruction pool depending upon their status. If the status indicates that a uop has all of its operands then the dispatch unit checks to see if the execution resource needed by that uop is also available. If both are true, it removes that uop and sends it to the resource where it is executed. The results of the uop are later returned to the pool. There are five ports on the Reservation Station and the multiple resources are accessed as shown in Figure 5:

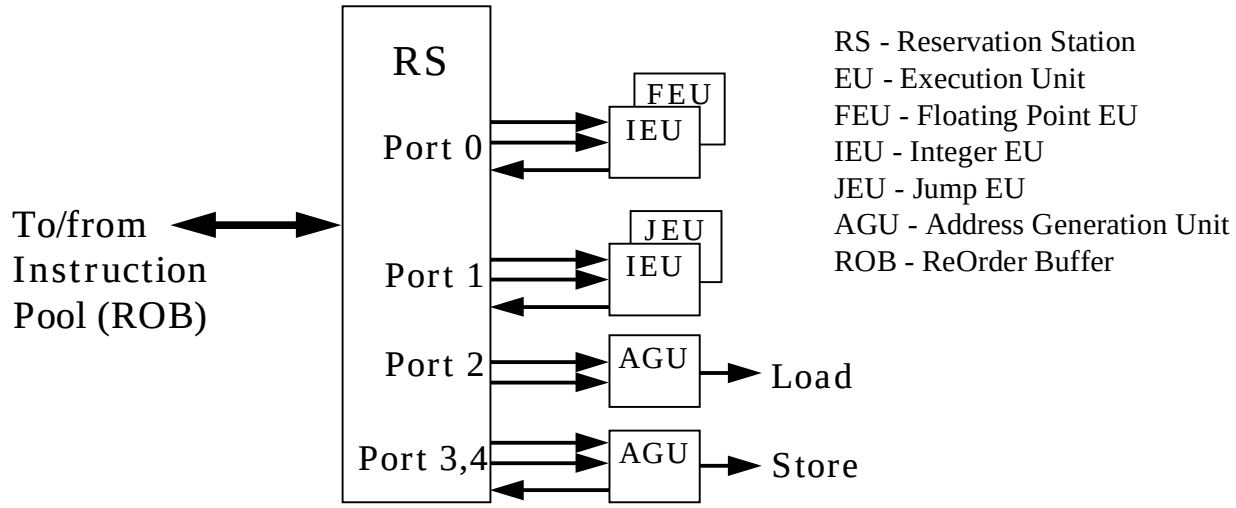


Figure 5: Looking inside the Dispatch/Execute Unit

The P6 can schedule at a peak rate of 5 uops per clock, one to each resource port, but a sustained rate of 3 uops per clock is typical. The activity of this scheduling process is the quintessential out-of-order process; uops are dispatched to the execution resources strictly according to dataflow constraints and resource availability, without regard to the original ordering of the program.

Note that the actual algorithm employed by this execution-scheduling process is vitally important to performance. If only one uop per resource becomes data-ready per clock cycle, then there is no choice. But if several are available, which should it choose? It could choose randomly, or first-come-first-served. Ideally it would choose whichever uop would shorten the overall dataflow graph of the program being run. Since there is no way to really know that at run-time, it approximates by using a pseudo FIFO scheduling algorithm favoring back-to-back uops.

Note that many of the uops are branches, because many IA instructions are branches. The Branch Target Buffer will correctly predict most of these branches but it can't correctly predict them all. Consider a BTB that's correctly predicting the backward branch at the bottom of a loop: eventually that loop is going to terminate, and when it does, that branch will be mispredicted. Branch uops are tagged (in the in-order pipeline) with their fallthrough address and the destination that was predicted for them. When the branch executes, what the branch actually did is compared against what the prediction hardware said it would do. If those coincide, then the branch eventually retires, and most of the speculatively executed work behind it in the instruction pool is good.

But if they do not coincide (a branch was predicted as taken but fell through, or was predicted as not taken and it actually did take the branch) then the Jump Execution Unit (JEU) changes the status of all of the uops behind the branch to remove them from the instruction pool. In that case the proper branch destination is provided to the BTB which restarts the whole pipeline from the new target address.

Tour stop #3: The RETIRE unit

Figure 6 shows a more detailed view of the retire unit:

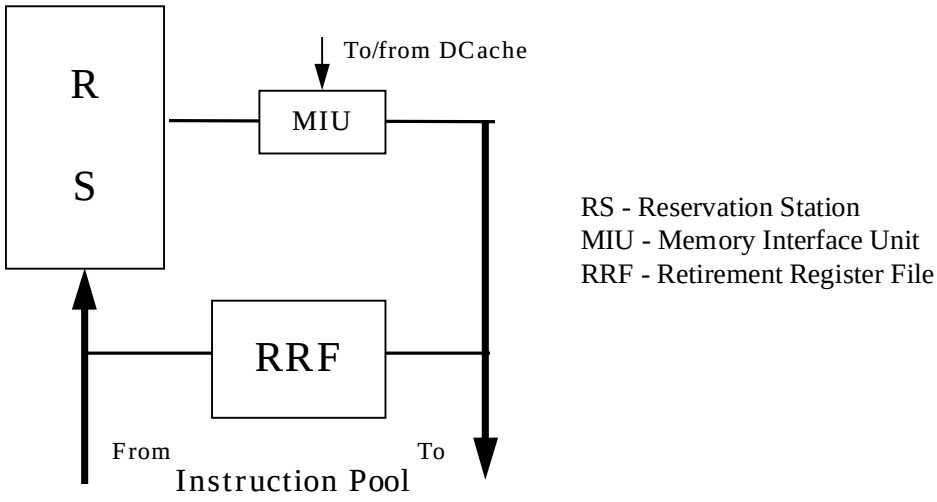


Figure 6: Looking inside the Retire Unit

The retire unit is also checking the status of uops in the instruction pool - it is looking for uops that have executed and can be removed from the pool. Once removed, the uops' original architectural target is written as per the original IA instruction. The retirement unit must not only notice which uops are complete, it must also re-impose the original program order on them. It must also do this in the face of interrupts, traps, faults, breakpoints and mis-predictions.

There are two clock cycles devoted to the retirement process. The retirement unit must first read the instruction pool to find the potential candidates for retirement and determine which of these candidates are next in the original program order. Then it writes the results of this cycle's retirements to both the Instruction Pool and the RRF. The retirement unit is capable of retiring 3 uops per clock.

Tour stop #4: BUS INTERFACE unit

Figure 7 shows a more detailed view of the bus interface unit:

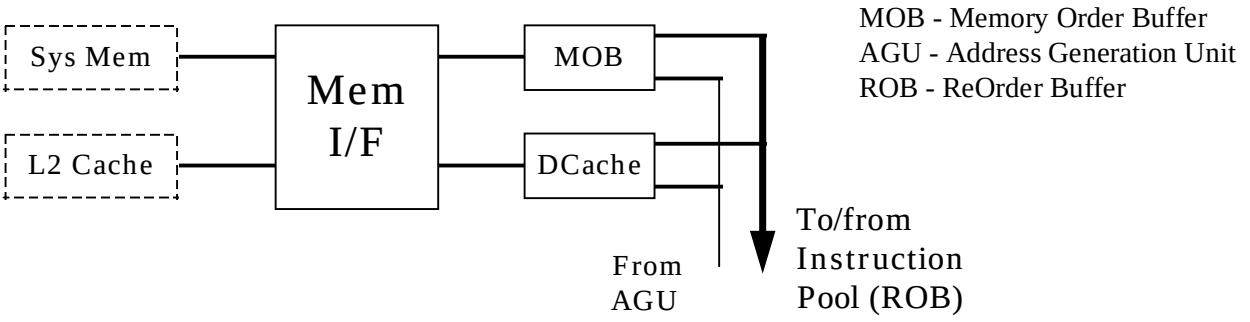


Figure 7: Looking inside the Bus Interface Unit

There are two types of memory access: loads and stores. Loads only need to specify the memory address to be accessed, the width of the data being retrieved, and the destination register. Loads are encoded into a single uop.

Stores need to provide a memory address, a data width, and the data to be written. Stores therefore require two uops, one to generate the address, one to generate the data. These uops are scheduled independently to maximize their concurrency, but must re-combine in the store buffer for the store to complete.

Stores are never performed speculatively, there being no transparent way to undo them. Stores are also never re-ordered among themselves. The Store Buffer dispatches a store only when the store has both its address and its data, and there are no older stores awaiting dispatch.

What impact will a speculative core have on the real world? Early in the P6 project, we studied the importance of memory access reordering. The basic conclusions were as follows:

- Stores must be constrained from passing other stores, for only a small impact on performance.
- Stores can be constrained from passing loads, for an inconsequential performance loss.
- Constraining loads from passing other loads or from passing stores creates a significant impact on performance.

So what we need is a memory subsystem architecture that allows loads to pass stores. And we need to make it possible for loads to pass loads. The Memory Order Buffer (MOB) accomplishes this task by acting like a reservation station and ReOrder Buffer, in that it holds suspended loads and stores, redispatching them when the blocking condition (dependency or resource) disappears.

Tour Summary

It is the unique combination of improved branch prediction (to offer the core many instructions), data flow analysis (choosing the best instructions to operate upon), and speculative execution (executing instructions in the preferred order) that enables the P6 to deliver twice the performance of a Pentium processor on the same semiconductor manufacturing process. This unique combination is called Dynamic Execution and it is similar in impact as “Superscalar” was to previous generation Intel Architecture processors.

And while our architects have been honing the P6 microarchitecture, our silicon technologists have been working on the next Intel process - this 0.35 micron process will enable future P6 CPU core speeds in excess of 200MHz.

A 0.6um BiCMOS Processor with Dynamic Execution

Robert P. Colwell, Randy L. Steck

A next generation, Intel Architecture compatible microprocessor with dynamic execution has been implemented with a 0.6um 4 layer metal BiCMOS process [1]. Performance is achieved through the use of a large, full-speed cache accessed through a dedicated bus interface feeding a generalized dynamic execution microengine. A primary 64-bit processor bus includes additional pipelining features to provide high throughput to this CPU and cache. These and other techniques result in a projected performance of greater than 200 Ispec92. Testability features built into the design allow complete access to all structures without the overhead of a full LSSD implementation. Included in this paper are a microarchitecture block diagram, implementation details and a die photo.

This processor implements dynamic execution using an out-of-order, speculative execution engine, with register renaming of integer [2], floating point and flags variables, multiprocessing bus support, and carefully controlled memory access reordering. The flow of Intel Architecture instructions is predicted and these instructions are decoded into micro-operations (uops), or series of uops, and these uops are register-renamed, placed into an out-of-order speculative pool of pending operations, executed in dataflow order (when operands are ready), and retired to permanent machine state in source program order. This is accomplished with one general mechanism to handle unexpected asynchronous events such as mispredicted branches, instruction faults and traps, and external interrupts. Dynamic execution, or the combination of branch prediction, speculation and micro-dataflow, is the key to the high performance.

The basic operation of the microarchitecture (Figure 1) is as follows:

1. The 512 entry Branch Target Buffer (BTB) helps the Instruction Fetch Unit (IFU) choose an instruction cache line for the next instruction fetch. ICache line fetches are pipelined with a new instruction line fetch commencing on every CPU clock cycle.
2. Three parallel decoders (ID) convert multiple Intel Architecture instructions into multiple sets of micro-ops (uops) each clock.
3. The sources and destinations of these uops are renamed by the Register Alias Table (RAT), which eliminates register re-use artifacts, and are forwarded to the Reservation Station (RS) and to the ReOrder Buffer (ROB).
4. The renamed uops are queued in the RS where they wait for their source data - this can come from several places, including immediates, data bypassed from just-executed uops, data present in a ROB entry, and data residing in architectural registers (such as EAX).
5. The queued uops are dynamically executed according to their true data dependencies and execution unit availability (IEU, FEU, AGU). The order in which any given uops execute in time has no particular relationship to the order implied by the source program.
6. Memory operations are dispatched from the RS to the Address Generation Unit (AGU) and to the Memory Ordering Buffer (MOB). The MOB ensures that the proper memory access ordering rules are observed.
7. Once a uop has executed, and its destination data has been produced, that result data is forwarded to subsequent uops that need it, and the uop becomes a candidate for “retirement”.
8. Retirement hardware in the ROB uses uop timestamps to reimpose the original program order on the uops as their results are committed to permanent architectural machine state in the Retirement Register File (RRF). This retirement process must observe not only the original program order, it must correctly handle interrupts and faults, and flush all or part of its state on detection of a mispredicted branch. When a uop is retired, the ROB writes that uop’s result into the appropriate RRF entry and notifies the RAT of that retirement so that subsequent register renaming can be activated.

The component includes separate data and instruction L1 caches (each of which is 8KB), and a unified L2 cache. The L1 Data Cache is dual-ported, non-blocking, supporting one load and one store per cycle. The L2 cache interface runs at the full CPU clock speed, and can transfer 64 bits per cycle. The external bus is also 64-bits and can sustain a data transfer every bus-cycle. This external bus operates at 1/2, 1/3, or 1/4 of the CPU clock speed.

Clock distribution was carefully designed to minimize skew across the entire die by generating a master clock with a PLL which has been delay synchronized to output signals. Global clocks distributed to 80 different units across the die are then buffered for the specific load in each clock sub-branch. Tuning of these drivers, and careful routing of the global clocks, results in a worst-case global clock skew of 250 pS.

BiCMOS circuits were used extensively thoughout the design, providing lower delay for higher loads, and increasing overall performance by approximately 15%. Figure 2 shows the relative delay to a comparable CMOS inverter and Figure 3 shows the most common implementation of a BiCMOS gate.

A Vdd of 2.9v was selected as an optimal point for CMOS and BiCMOS gate performance while reducing overall power.

Delayed precharge domino logic, shown in Figure 4, was also used for speed-critical paths, particularly in the instruction decode logic, and resulted in lower power than standard domino logic during transitions with fewer race conditions on outputs.

Test structures were defined allowing full access to all processor logic while requiring only 4% of the full die area and incurring no speed penalty. This is accomplished through the use of test registers, mode bits to provide specific logic access, and the use of serial Scanout (scan chains). Scanout in particular provides observability of virtually all important signals within the design with no speed impact, and it is used extensively in test and debug. On-chip BIST is implemented to complement the Scanout observability. The processor is a fully static design accommodating IDDQ testing.

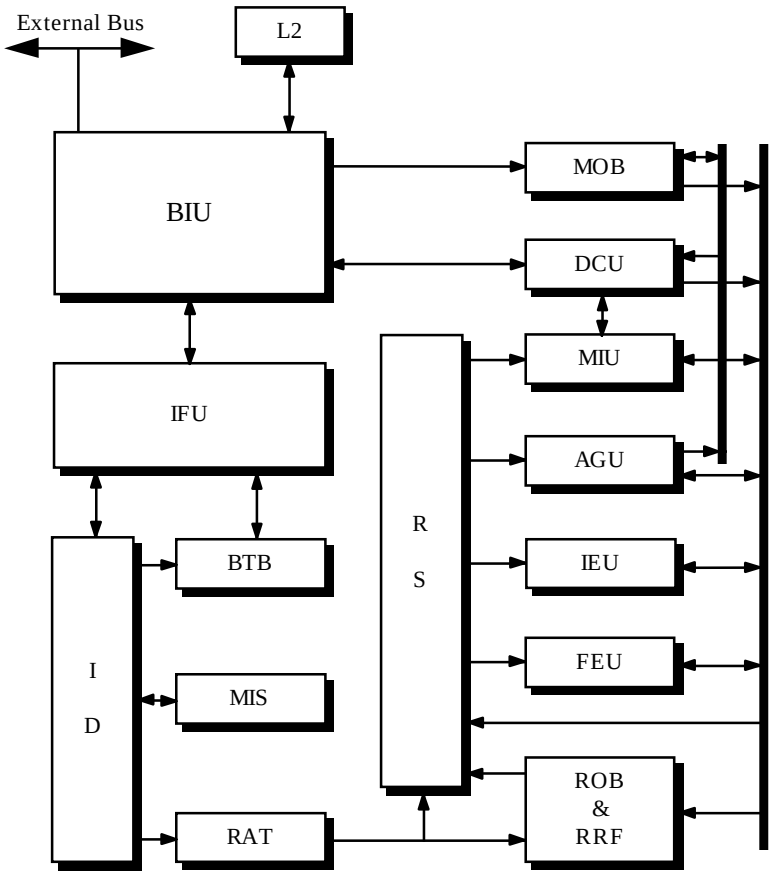
Features for lower power operation, (such as StopClock and standby mechanisms) and features intended to improve system management and RAS (Reliability, Availability, Serviceability) are included. An extensive Machine Check Architecture has been incorporated, with facilities to detect errors in hardware and allow those errors to be handled in software.

Acknowledgements
We are fortunate to represent the work of many talented, dedicated professionals and it is mainly their efforts that are presented here.

References

[1] Schutz, J., “A 3.3V 0.6um BiCMOS SuperScalar Microprocessor”, ISSCC Proceedings, pp. 202-203, 1994

[2] Hennessy, J. et al, Computer Architecture A Quantitative Approach, Morgan Kaufman Publishers Inc., 1990



- BIU: Bus Interface Unit
- IFU: Instruction Fetch Unit (includes ICache)
- BTB: Branch Target Buffer
- ID: Instruction Decoder
- MIS: Microinstruction Sequencer
- RAT: Register Alias Table
- ROB: ReOrder Buffer
- RRF: Retirement Register File
- RS: Reservation Station
- IEU: Integer Execution Unit
- FEU: Floating point Execution Unit
- AGU: Address Generation Unit
- MIU: Memory Interface Unit
- DCU: Data Cache Unit (includes DCache)
- MOB: Memory ReOrder Buffer
- L2: Level 2 Cache

Figure 1 - Basic CPU Block Diagram

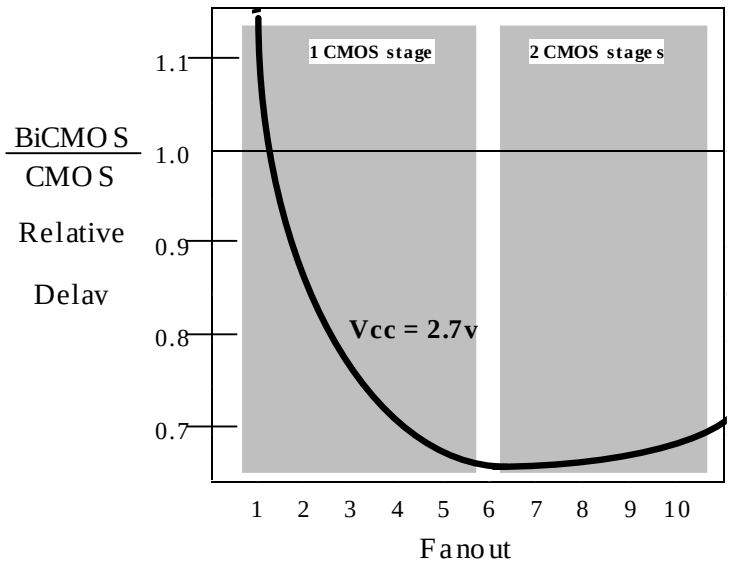


Figure 2 - BiCMOS vs CMOS relative inverter delay

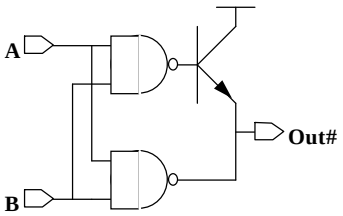


Figure 3 - BiCMOS Gate

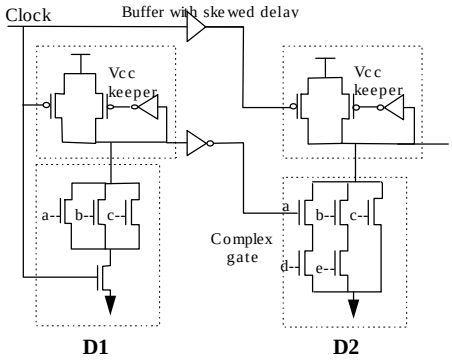


Figure 4 - Delayed Precharge Domino

P6 Technical Glossary

Architectural State. The value of registers, flags and memory as viewed by the programmer.

Associative Memory. A table that is accessed not via an explicit index, but by the data it contains. If no entries of the associative memory match the input data, a "miss" signal is asserted. If any entries of the associative memory match the input data, the associative memory indicates the match, and produces any related data that was stored with that entry. This is often termed Content Addressable Memory.

Branch Prediction. Pipelined machines must fetch the next instruction before they have completely executed the previous instruction. If the previous instruction was a branch, then the next-instruction fetch could have been from the wrong place. Branch prediction is a technique that attempts to infer the proper next instruction address, knowing only the current one, typically using an associative memory called a BTB.

Branch Recovery. When a branch is mispredicted, the speculative state of the machine must be flushed and fetching restarted from the correct target address. We call this activity branch recovery.

BTB. Branch Target Buffer. A small (typically 128-512 entry) associative memory that watches the ICache index and tries to predict which ICache index should be accessed next, based on branch history. Optimizing the actual algorithm used in retaining the history of each entry is an area of ongoing research. P6 uses a variant of Yeh's algorithm (IEEE Micro-24 conference proceedings, 1991.)

CAM. Content Addressable Memory. Synonym for Associative Memory. A table that is accessed not via an explicit index, but by the data it contains. If no entries of the associative memory match the input data, a "miss" signal is asserted. If any entries of the associative memory match the input data, the associative memory indicates the match, and produces any related data that was stored with that entry.

Dynamic Execution. Dynamic Execution involves optimally adjusting instruction execution by prediciting program flow, analysing the program's dataflow graph to choose the best order to execute the instructions, then speculatively executing instructions in the preferred order.

ICache. Instruction Cache. A fast local memory that holds the instructions to be executed. When a program tries to access an instruction that is not yet (or no longer) in the cache, the CPU must wait until hardware fetches the desired instructions from another cache (or memory itself) downstream. These stalls in the fetch/decode unit of the P6 are typically overlapped by the other units that are processing independently.

Instruction Pool. A metaphor used to describe the mechanism used by three independent P6 units to communicate. The pool is implemented as a CAM called the ReOrder Buffer (ROB).

L2 Cache. Caches exist in a "memory hierarchy." There is a small but very fast L1 cache; if that misses, then the access is passed on to the bigger but slower L2 cache, and if that misses, the access goes to main memory (or L3 cache if the system has one).

Pipelining. A microarchitecture design technique that divides the execution of an instruction into sequential steps, using different microarchitectural resources at each step. Pipelined machines have multiple IA instructions executing at the same time, but at different stages in the machine.

RAT. Register Alias Table. Renames programmer visible register references to internal physical registers. This mapping is done at run time.

Reservation Station. A generalized mechanism where uops wait for their dependent component parts. Once a uop has all of its operands, the uop is dispatched to a resource unit for execution.

Resource Unit. The p6 ahs multiple resources that are scheduled by the Reservation Station: they include 2 integer units, a full floating point arithmetic unit, a floating point multiplier, divider, and shifter, and two address generation units.

Retirement. A generalized mechanism that removes a completed uop from the ROB and commits its state to whatever permanent architectural state was designated by the Intel architecture instruction.

ROB. ReOrder Buffer. The P6 functional unit where initial uops wait, speculative results are collected, and then are retired.

Speculative Execution. A generalized mechanism that permits instructions to be started "early," i.e., ahead of their normal execution sequence. Results of this speculation are stored temporarily (in the ROB) since they may be discarded due to a change in program flow.

Store Buffer. A queue that receives write requests from the CPU and sends them to the memory subsystem. This store buffer is snooped by pending loads.

Superscalar. The ability to process more than one instruction per clock. The Pentium processor has two execution pipes (U and V) so it is superscalar level 2. The P6 can dispatch and retire 3 instructions per clock so it is superscalar lever 3.

UOP. A micro operation. The three decoders translate Intel architecture instructions into fixed length uops that are easier to schedule by the dispatch/execute unit. Most IA instructions translate to single uops, some need up to four, and the complex instructions (e.g., Enter, Leave) need microcode support.