The Intel® Pentium® 4 Processor

Doug Carmean
Principal Architect
Intel Architecture Group

Spring 2002
Agenda

- Review
- Pipeline Depth
- Execution Trace Cache
- Data Speculation
- Spec Performance
- Summary
Information in this document is provided in connection with Intel® products. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. Except as provided in Intel’s Terms and Conditions of Sale for such products, Intel assumes no liability whatsoever, and Intel disclaims any express or implied warranty, relating to sale and/or use of Intel® products including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright or other intellectual property right. Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

This document contains information on products in the design phase of development. Do not finalize a design with this information. Revised information will be published when the product is available. Verify with your local sales office that you have the latest datasheet before finalizing a design.

Intel processors may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Intel, Pentium, and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and foreign countries.

Copyright © (2001) Intel Corporation.
Deeper Pipelines enable higher frequency and performance.
Hyper Pipelined Technology

- **Introduction**
  - 60MHz
  - 166MHz

- **P5 Micro-Architecture**
  - 1.5GHz
  - 200MHz

- **P6 Micro-Architecture**
  - 1.8GHz
  - 233MHz

- **Netburst Micro-Architecture**
  - 2.2GHz
  - 1.2GHz

- **Today**
  - 2200MHz
Deeper Pipelines are Better

Performance Improvement

Pipeline Depth

Source: Average of 2000 application segments from performance simulations
Why not deeper pipelines?

- Increases complexity
  - Harder to balance
  - More challenges to architect around
  - More algorithms
  - Greater validation effort
  - Need to pipeline the wires

Overall Engineering Effort Increases Quickly as Pipeline depth increases
Performance

- High bandwidth front end
- Low latency core

High Bandwidth Front End
Higher Frequency increases requirements of front end

- Branch prediction is more important
  - So we improved it
- Need greater uop bandwidth
  - Branches constantly change the flow
  - Need to decode more instructions in parallel
Execution Trace Cache

1 cmp
2 br -> T1
   ...
   (unused code)
T1: 3 sub

4 br -> T2
   ...
   (unused code)
T2: 5 mov
   6 sub

7 br -> T3
   ...
   (unused code)
T3: 8 add
   9 sub

10 mul
11 cmp
12 br -> T4

Trace Cache Delivery

1 cmp  2 br T1  3 T1: sub
4 br T2  5 mov  6 sub
7 br T3  8 T3:add  9 sub
10 mul  11 cmp  12 br T4
### Execution Trace Cache

#### P6 Microarchitecture

<table>
<thead>
<tr>
<th>1</th>
<th>cmp</th>
<th>2</th>
<th>br T1</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>T1: sub</td>
<td>4</td>
<td>br T2</td>
</tr>
<tr>
<td>5</td>
<td>mov</td>
<td>6</td>
<td>sub</td>
</tr>
<tr>
<td>8</td>
<td>T3: add</td>
<td>9</td>
<td>sub</td>
</tr>
<tr>
<td>11</td>
<td>cmp</td>
<td>12</td>
<td>br T4</td>
</tr>
</tbody>
</table>

**BW = 1.5 uops/ns**

#### Trace Cache Delivery

<table>
<thead>
<tr>
<th>1</th>
<th>cmp</th>
<th>2</th>
<th>br T1</th>
<th>3</th>
<th>T1: sub</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>br T2</td>
<td>5</td>
<td>mov</td>
<td>6</td>
<td>sub</td>
</tr>
<tr>
<td>7</td>
<td>br T3</td>
<td>8</td>
<td>T3: add</td>
<td>9</td>
<td>sub</td>
</tr>
<tr>
<td>10</td>
<td>mul</td>
<td>11</td>
<td>cmp</td>
<td>12</td>
<td>br T4</td>
</tr>
</tbody>
</table>

**BW = 6 uops/ns**
Inside the Execution Trace Cache

Instruction Pointer: 0x0900

Set 0 Set 1 Set 2 Set 3 Set 4
Way 0 Way 1 Way 2 Way 3

- **head**: cmp, br T1, T1:sub, br T2, mov, sub
- **body 1**: br T3, T3:add, sub, mul, cmp, br T4
- **body 2**: T4:add, sub, mov, add, add, mov
- **tail**: add, sub, mov, add, add, mov
Self Modifying Code

- Programs that modify the instruction stream that is being executed
- Very common in Java* code from JITs
- Requires hardware mechanisms to maintain consistency

*Other names and brands may be claimed as the property of others.
Self Modifying Code

• The hardware needs to handle two basic cases:
  – Stores that write to instructions in the Trace Cache
  – Instruction fetches that hit pending stores
    – Speculative
    – Committed
Case 1: Stores to cached instructions

Execution Core

Data TLB

Store’s Physical Address

Instruction TLB (128 entries)

Trace Cache

“in use” bits
Case 2: Fetches to pending stores

Instruction Pointer

Instruction TLB (128 entries)

Addr

Addr

Addr

Execution Core

Write Combining Buffer

Speculative Store Buffer

Committed

Please Re-Fetch

Please Re-Fetch

Please Flush Pipeline

Copyright © 2002 Intel Corporation.
Execution Trace Cache

- Provides higher bandwidth for higher frequency core
- Reduces fetch latency
- Requires new fundamentally new algorithms
Performance

- High bandwidth front end
- Low latency core

Low Latency Core
Data Speculation

- Use data before we are sure it is valid
  - Lowers effective LD latency
  - Fast ALUs in Pentium 4 want fast LDs
  - Ratio of LD latency to ADD latency is important if 1 in 5 uops is a LD

- As pipelines get deeper, data speculation gets more important
  - Number of cycles saved /w data speculation increases as pipeline depth increases
L1 Cache is >3x Faster

- P6:
  - 3 clocks @ 1GHz

- P4:
  - 2 clocks @ 2GHz

Lower Latency is Higher Performance
L1 Data Cache

Pipeline Stages

VA 15:0

VA 31:16

2x Clock

Fast SB

data

tag

Data

TLB

TAG

Slow SB

replay

replay

replay

Copyright © 2002 Intel Corporation.
L1 Data Cache

Way Predictor (Tag array)

Data array

Way select

Hit (Replay)

Copyright © 2002 Intel Corporation.
A Digression on Stores

- Two components to a store:
  - STA: address computation
  - STD: data piece

- Hybrid uOP
  - Single uOP in the front, back ends
  - Two uOPs in the middle
Memory Disambiguation

- If the store is older
  - And AddrA = AddrB
  - Then the load must get DataB

- Dependencies can not be resolved until execution
Memory Disambiguation

Option 1

Loads wait for:
- All older STAs AND
- All older STDs
- No Recovery

K7?

Option 2

Loads wait for:
- All older STAs
- STD Recovery

P4

Option 3

Predict
- Specific older STAs
- Complex Recovery

EV8

Copyright © 2002 Intel Corporation.
Example

ML=0

- ST writes new value

ML=1

- LD2 forwards
- Branch resolves based on new value

JMP if ML=0

STA, STD

LD1

XOR

LD2

BR
Example

ML=0

LD misses, replays

ML=1

STA dependent, replays

LD hits, gets old value

BR mispredicts

JMP if ML=0
Example

- If LD2 depends on the STA, they are usually part of the same dataflow graph.
- If the STA replays, the LD usually has an address dependence.
Cautious Mode

- Normally, aggressively schedule
- If a large number of problems occur enter Cautious Mode
- In Cautious Mode, branches wait for data to be non-speculative
  - Increases branch misprediction latency
  - Completely eliminates problems
Cautious Mode: Implementation

- A simple state machine cleans up the outliers
  - Out of 2200 traces, 3 traces speedup >20%
  - The other traces are unaffected
  - Average performance improvement < 0.1%
Performance

- High bandwidth front end
- Low latency core
- Lower memory latency
Reducing Latency

- As frequency increases, it is important to improve the performance of the memory subsystem
- Data Prefetch Logic
  - Watches processor memory traffic
  - Looks for patterns
  - Initiates accesses
Prefetch logic first checks L2 cache and then fetches lines from memory that miss L2 cache.
Data Prefetch Logic

- Watches for streaming memory access patterns
  - Can track 8 multiple independent streams
  - Loads, Stores or Instruction
  - Forward or Backward

- Analysis on 32 byte cache line granularity

- Looks for “mostly” complete streams:
  - Access to cache lines 1,2,3,4,5,6 will prefetch
  - Access to cache lines 1,2,4,5,6 will prefetch
  - 1,3,6,9 will not prefetch
Performance

- High bandwidth front end
- Low latency core
- Lower memory latency