Case Study: Alpha 21264
Digital Equipment Corporation

• One of the Big Old Computer companies (along with IBM)
  – Business-oriented computers
  – Check out Gordon Bell’s lecture in “History of Computing” class
• They produced a string of famous machines
• Sold to Compaq in 1998
• Sold to HP (and Intel) in 2002
The PDPs

- Most famous: PDP-11
  - Birthplace of UNIX
  - Elegant ISA
  - Designed by a small team in short order
    - In response to competitor
    - Formed by defecting engineers
  - 16 bits of virtual address
    - PDP-5 and PDP-8 were 12 bits
  - Chronically short of address bits
  - Sold until 1997
The VAX

• (In)famous and long-lived
  – for "Virtual Address Extension (to the PDP-11)"
  • LOTS of extensions
  – Very CISCy -- polynomial evaluate inst. Etc.
The Alpha

• Four processors
  – 21064, 21164, 21264, 21364, (21464)
  – 21 for “21st century”; 64 - for “64 bit”

• High-end workstations/servers
• Fast processors in the world at introduction
• Unix, VMS (old VAX OS), WindowsNT, Linux
• Alpha died when Intel bought the IP and the design team.
AlphaAXP

• **New ISA from scratch**
  – No legacy anything (almost)
    • VAX-style floating point mode
  – 64-bit
  – Very clean RISC ISA
    • Register-Register/Load-Store
    • No condition codes
    • Conditional moves -- reduced branching, but at what cost?
  – 32 GPRs and FPRs
• **OS support**
  – PALCode -- “firmware” control of low-level hardware
• **VAX compatibility provided in software**
  – VAX ISA -> Alpha via a compiler
Alpha 21064

- Introduced in 1991
- 100-300Mhz (blazingly fast at the time)
- 750nm/0.75micron (vs 45nm today)
- 234mm$^2$ die, 1.6M transistors
- 33 Watts
- Full custom design
Alpha 21064 (cont)

- **Pipeline**
  - Dual issue
  - 7 stage integer/10 stage FP
  - 4 cycle mis-prediction penalty.
  - 45 bypassing paths
  - 22 instructions “in flight”

- **Caches**
  - On-chip L1I + L1D. 8KB each
  - Off-chip L2

- **Branch prediction**
  - Static: forward taken/Back not taken
  - Simple dynamic prediction
  - 80% accuracy
Alpha 21164

- Introduced in 1995
- 500Mhz
- 500nm/0.5micron
- 299mm² die, 9.7M transistors
- 56W
Alpha 21164 (cont)

• Pipeline
  – Quad issue: 2 integer + 2 FP
  – 7 stage integer/10 stage FP

• Caches
  – On-chip L1I + L1D. 8KB each. Direct-mapped (fast!)
    • Hit under miss/miss under miss (21 outstanding at once)
  – On-chip 3-way 96KB L2.
  – Off-chip L3 (1-64MB)

• ISA changes
  – Native support for byte operations

• Branch prediction
  – 5 cycle mispredict penalty
  – History-based dynamic predictor. Bits stored per cache line.
Alpha 21264

- Introduced in 1998
- 600Mhz-1.2Ghz
- 0.35-0.18micron
- 314mm² die, 15.2M transistors
- 73W
Alpha 21264 (cont)

• Pipeline
  – 6-issue: 4 integer + 2 FP
  – 7 stage integer/longer for FP, depending on op.
  – 80 in-flight instructions

• Caches
  – On-chip L1I + L1D. 64KB each. 2-way
  – Off-chip L2
  – Compared to 21164 8x the L1 capacity, but no on-chip L2
Aggressive Speculation

- The 21264 executes instructions that may or may not be on the correct path.
- When it’s wrong, it has to undo those instructions
  - It stores backups of renaming tables, register file, etc.
  - It also must prevent changes to memory from occurring until the instructions “commit”
In Order Fetch and Commit

- Fetch is in-order
- Execution is out of order
  - Extract as much parallelism as possible
- Commit is in-order
  - Make the changes permanent in program order.
  - This is what is “visible” to the programmer.
  - This enables precise exceptions (mostly)
Alpha 21264 (cont)

• Fetch unit
  – Pre-decodes instructions in the Icache
  – next line and set predictors -- correct 80-100%
  – Tournament predictor
    • A local history predictor + A global history predictor
    • A third predictor to track which one is most effective
    • 2 cycle to make a prediction
Alpha 21264: I Cache/fetch

- 64KB, 2-way, 16byte lines (4 instructions)
- Each line also contains extra information:
  - Incorporates BTB and parts of instruction decode
  - BTB data is protected by 2-bits of hysteresis, trained by branch predictor.
- Branch prediction is aggressive to find parallelism and exploit speculative out-of-order execution.
  - We wants lots of instructions in flight.
- On a miss, it prefetched up to 64 instructions
Alpha 21264

Branch Predictor

L1I 64KB, 2-way

Next line/Set prediction

Fetch

Slot

Rename

Int reg rename

Int IQ 20 entries

Issue

Int IQ 20 entries

Int Reg File (80)

Reg Read

Int Reg File (80)

Execute

ALU

ALU

ALU

L1D 64KB 2-way

Memory

L2 96KB 3-way

Int reg rename

FP reg rename

FP IQ 15 entries

FP Reg File (72)

FP Add

FP Mult
Alpha 21264

Fetch

Branch Predictor

Slot

Rename

Int reg rename

Int IQ 20 entries

Issue

Int IQ 20 entries

Reg Read

Int Reg File (80)

Execute

ALU

ALU

ALU

L1D 64KB 2-way

Memory

L2 96KB 3-way

"enriched" L1 Icache

Next line/ Set prediction

L1I 64KB, 2-way

FP reg rename

FP IQ 15 entries

Reg File (72)

FP Mult

FP Add
Alpha 21264

Out-of-order

Fetch

Slot

Rename

Issue

Reg Read

Execute

Memory

Branch Predictor

Int reg rename

Int IQ 20 entries

Int Reg File (80)

ALU

L1D 64KB 2-way

L2 96KB 3-way

Next line/ Set prediction

L1 Icache

Int reg rename

FP reg rename

FP IQ 15 entries

FP Reg File (72)

FP Mult

FP Add

“Cluster”

“enriched” L1 Icache
Alpha 21264: Renaming

- Separate INT and FP
- Replaces “architectural registers” with “physical registers”
  - 80 integer physical registers
  - 72 FP physical registers
  - Eliminates WAW and WAR hazards
- Register map table maintains mapping between architectural and physical registers
  - One copy for each in-flight instruction (80 copies)
- Special handling for conditional moves.
Alpha 21264: Renaming

- Two parts
  - Content-addressable lookup to find physical register inputs
  - Register allocation to rename the output
- Four instructions can be renamed each cycle.
  - 8 ports on the lookup table
  - 4 allocations per cycle
- There is no fixed location for architectural register values!
  - How can we read architectural register r10?
Alpha 21264: Renaming

1: Add r3, r2, r3
2: Sub r2, r1, r3
3: Mult r1, r3, r1
4: Add r2, r3, r1
5: Add r2, r1, r3
Alpha 21264: Renaming

1: Add r3, r2, r3
2: Sub r2, r1, r3
3: Mult r1, r3, r1
4: Add r2, r3, r1
5: Add r2, r1, r3

\[ \begin{array}{ccc}
\text{r1} & \text{r2} & \text{r3} \\
\text{p1} & \text{p2} & \text{p3} \\
\text{p1} & \text{p2} & \text{p4} \\
\text{p4} & \text{p2} & \text{p3} \\
\text{p4} & \text{p2} & \text{p3} \\
\end{array} \]
Alpha 21264: Renaming

1: Add r3, r2, r3  
2: Sub r2, r1, r3  
3: Mult r1, r3, r1  
4: Add r2, r3, r1  
5: Add r2, r1, r3

p4, p2, p3  
p5, p1, p4

<table>
<thead>
<tr>
<th></th>
<th>r1</th>
<th>r2</th>
<th>r3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>p1</td>
<td>p2</td>
<td>p3</td>
</tr>
<tr>
<td>1</td>
<td>p1</td>
<td>p2</td>
<td>p4</td>
</tr>
<tr>
<td>2</td>
<td>p1</td>
<td>p5</td>
<td>p4</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

RAW ➔ WAW ➔ WAR
1: Add r3, r2, r3  p4, p2, p3
2: Sub r2, r1, r3  p5, p1, p4
3: Mult r1, r3, r1  p6, p4, p1
4: Add r2, r3, r1
5: Add r2, r1, r3

RAW → WAW → WAR
Alpha 21264: Renaming

1: Add \( r3, r2, r3 \)  \( \rightarrow \)  \( p4, p2, p3 \)
2: Sub \( r2, r1, r3 \)  \( \rightarrow \)  \( p5, p1, p4 \)
3: Mult \( r1, r3, r1 \)  \( \rightarrow \)  \( p6, p4, p1 \)
4: Add \( r2, r3, r1 \)  \( \rightarrow \)  \( p7, p4, p6 \)
5: Add \( r2, r1, r3 \)

<table>
<thead>
<tr>
<th></th>
<th>r1</th>
<th>r2</th>
<th>r3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>p1</td>
<td>p2</td>
<td>p3</td>
</tr>
<tr>
<td>1</td>
<td>p1</td>
<td>p2</td>
<td>p4</td>
</tr>
<tr>
<td>2</td>
<td>p1</td>
<td>p5</td>
<td>p4</td>
</tr>
<tr>
<td>3</td>
<td>p6</td>
<td>p5</td>
<td>p4</td>
</tr>
<tr>
<td>4</td>
<td>p6</td>
<td>p7</td>
<td>p4</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

RAW  \( \rightarrow \)  WAW  \( \rightarrow \)  WAR
Alpha 21264: Renaming

1: Add r3, r2, r3  \( \text{p4, p2, p3} \)
2: Sub r2, r1, r3  \( \text{p5, p1, p4} \)
3: Mult r1, r3, r1  \( \text{p6, p4, p1} \)
4: Add r2, r3, r1  \( \text{p7, p4, p6} \)
5: Add r2, r1, r3  \( \text{p8, p6, p4} \)

\[
\begin{array}{|c|c|c|c|}
\hline
& r1 & r2 & r3 \\
\hline
0 : & p1 & p2 & p3 \\
1 : & p1 & p2 & p4 \\
2 : & p1 & p5 & p4 \\
3 : & p6 & p5 & p4 \\
4 : & p6 & p7 & p4 \\
5 : & p6 & p8 & p4 \\
\hline
\end{array}
\]
Alpha 21264: Renaming

1: Add r3, r2, r3
2: Sub r2, r1, r3
3: Mult r1, r3, r1
4: Add r2, r3, r1
5: Add r2, r1, r3

<table>
<thead>
<tr>
<th></th>
<th>r1</th>
<th>r2</th>
<th>r3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>p1</td>
<td>p2</td>
<td>p3</td>
</tr>
<tr>
<td>1</td>
<td>p1</td>
<td>p2</td>
<td>p4</td>
</tr>
<tr>
<td>2</td>
<td>p1</td>
<td>p5</td>
<td>p4</td>
</tr>
<tr>
<td>3</td>
<td>p6</td>
<td>p5</td>
<td>p4</td>
</tr>
<tr>
<td>4</td>
<td>p6</td>
<td>p7</td>
<td>p4</td>
</tr>
<tr>
<td>5</td>
<td>p6</td>
<td>p8</td>
<td>p4</td>
</tr>
</tbody>
</table>

RAW → WAW → WAR
Alpha 21264: Issue Queue

- Separate Int and FP
- Decouple front and back ends
- Dynamically track dependences
  - Instructions can issue once their input registers are written
  - Track register status in “register scoreboard”
  - Issue instructions “around” long-latency operations
  - Exploit cross-loop parallelism
- Issue up to 4 instructions/cycle (2 floating point)
  - Issue oldest first
  - Compact the queue (the free slots are always mostly at the top)
Alpha 21264: Issue Queue

1: Add p4, p2, p3
2: Sub p5, p1, p4
3: Mult p6, p4, p1
4: Add p7, p4, p6
5: Add p8, p6, p4
Alpha 21264: Issue Queue

1: Add p4, p2, p3
2: Sub p5, p1, p4
3: Mult p6, p4, p1
4: Add p7, p4, p6
5: Add p8, p6, p4
Alpha 21264: Issue Queue

1: Add p4, p2, p3
2: Sub p5, p1, p4
3: Mul p6, p4, p1
4: Add p7, p4, p6
5: Add p8, p6, p4
Alpha 21264: Issue Queue

1: Add p4, p2, p3
2: Sub p5, p1, p4
3: Mult p6, p4, p1
4: Add p7, p4, p6
5: Add p8, p6, p4

Register scoreboard:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>p5</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>p6</td>
<td></td>
</tr>
<tr>
<td>p3</td>
<td>p7</td>
<td></td>
</tr>
<tr>
<td>p4</td>
<td>p8</td>
<td></td>
</tr>
</tbody>
</table>

Register File

2,3

ALU 1

ALU
1: Add   p4, p2, p3
2: Sub   p5, p1, p4
3: Mul   p6, p4, p1
4: Add   p7, p4, p6
5: Add   p8, p6, p4

Register scoreboard

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>p5</td>
</tr>
<tr>
<td>p2</td>
<td>p6</td>
</tr>
<tr>
<td>p3</td>
<td>p7</td>
</tr>
<tr>
<td>p4</td>
<td>p8</td>
</tr>
</tbody>
</table>

Register File

ALU 2

ALU 3
1: Add p4, p2, p3
2: Sub p5, p1, p4
3: Mult p6, p4, p1
4: Add p7, p4, p6
5: Add p8, p6, p4
1: Add  p4, p2, p3
2: Sub  p5, p1, p4
3: Mult p6, p4, p1
4: Add  p7, p4, p6
5: Add  p8, p6, p4
1: Add p4, p2, p3
2: Sub p5, p1, p4
3: Mul p6, p4, p1
4: Add p7, p4, p6
5: Add p8, p6, p4
Alpha 21264: Execution

- Integer ALUs are clustered
- Two ALUs share a complete replica of the Int register file
- 1 cycle extra latency for cross-cluster updates
  - Not a big performance hit
  - Issue queue can issue any instruction to either cluster
  - Critical paths tend to stay in one cluster
- Area savings
  - Register file size is quadratic in # of ports
  - Each replica needs 4 read, 4 write ports (2 local writes, 2 remote)
  - Unclustered -> 8 read, 4 write ports
  - $O(2*8^2) \text{ vs } O(12^2)$
- Simpler too.
- This is the beginning of the “slow wires problem”
Alpha 21264: Memory Interface

• Memory is king!!!
  – One of Alpha’s niche markets was large, memory-intensive applications
  – They went 64-bits for the physical address space as much as for the virtual.

• Lots of outstanding requests
  – 32 loads, 32 stores (D only)
  – 8 cache misses (I + D)

• Big caches (64KB, 2-way)
  – What does Patterson’s thumb say?
  – 2 loads/stores per cycle
  – Double-pumped instead of multi-ported. (area vs clock rate)
  – Virtually-index, physically tagged

• 8-entry victim buffer shared between L1I and L1D
Alpha 21264: Memory interface

- Memory ordering
  - Renames memory locations

- LDQ/STQ
  - 32 entries each.
  - Sorted in fetch order (but arrive out-of-order)
  - Instruction remain in the queues until retirement
  - Load watch for younger stores to the same address
    - Squash the load and subsequent instructions if a match occurs
  - Stores watch for younger stores
  - Speculative loads get speculative data from “speculative store data buffer”
Alpha 21264: Retirement

- Instructions retire in-order
- At retirement
  - Stores write to memory
  - Renamed registers are released
    - Each instruction carries the physical register number that held the previous value for the instruction’s architectural destination register.
    - Since retirement is in-order, that register is dead.
- On exceptions,
  - All younger instructions are squashed
  - Register map reverts to state before the exception.
Alpha 21264: Memory interface

• Ordering violations

Source:  
ST r0, 0(r10)
LD r1, 0(r11)

Execution:  
LD r1, 0(r11)
... 
ST r0, 0(r10)

R11 == r10 => violation, pipe flush

• Mark the Load as “delayed”
  – In the future, it will wait for all previous stores
  – Clear the “delayed” flag ever 16,384 cycles
Alpha 21264: Memory Interface

• Speculative cache hits (integer only)
• The instruction queue assumes loads hit the L1
• When they don’t hit, do a mini-restart
  – Up to 8 instructions are “pulled back into the issue queue to be reissued”
  – Results in a 2 cycle bubble
• A single 4-bit predictor tracks the miss behavior.
Alpha 21364

- Introduced 2003
- 1.3Ghz
- 0.18micron, 130M transistors
- 400mm$^2$
- 125 Watts
- 21264 + 1.75MB on-chip L2
- Essentially a 21264 with an on-chip cache.
Alpha Family SpecINT 95 Performance

300MHz
1.7x improvement
600MHz 1.8x improvement
300MHz 1.7x improvement
**Alpha Family SpecINT 95 Performance**

- **600MHz**: 1.8x improvement
- **300MHz**: 1.7x improvement
- **27.8x improvement**
- **8.3x cycle time improvement**
- **3.5x from architecture**