### Introduction: CPUs and Attacks

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Evolution</th>
<th>Attacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>We’ll use Intel CPUs as a specific architecture example.</td>
<td>Architecture needs (cache, speculative execution, security levels) have increased complexity tremendously as CPUs went from 4 to 8, 16, 32, and 64 bit data sizes.</td>
<td>Complexity in CPU needs and features has driven exploits far into the corners of modern CPUs. <em>Meltdown</em> and <em>Spectre</em> are two large classes affecting nearly all modern, large processors as of 2018, affecting CPUs going back decades.</td>
</tr>
</tbody>
</table>

->NEEDS  
->COMPLEX  
->BINGO!
### Early Intel CPUs

<table>
<thead>
<tr>
<th>CPU</th>
<th>Year</th>
<th>D/A bits</th>
<th>Speed</th>
<th>Inst</th>
<th>trans/Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>4004</td>
<td>1971</td>
<td>4/4</td>
<td>108 kHz</td>
<td>46</td>
<td>2300/10000nm, 1kb program, 4kb data, 16 4-bit regs, 12x4 stack</td>
</tr>
<tr>
<td>8008</td>
<td>1972</td>
<td>8/8</td>
<td>800 kHz</td>
<td>48</td>
<td>3500/10000nm, 16k address space, 6 8-bit regs, 17x7 stack</td>
</tr>
<tr>
<td>8080/8085</td>
<td>1974</td>
<td>8/8</td>
<td>2-3 Mhz</td>
<td>80</td>
<td>6500/3000nm, 64kb address space, 6-8 bit regs, IO ports, stack pointer</td>
</tr>
<tr>
<td>8086/8088</td>
<td>1978/79</td>
<td>16/20</td>
<td>5 Mhz</td>
<td>81</td>
<td>29000/3000 nm, All regs &amp; addr lines 16 bits, 6 byte prefetch queue</td>
</tr>
</tbody>
</table>

- Instructions 1-6 bytes
- 8086+: Segmented memory (ES:AX style code)
- **Prefetch queue** loaded bytes ahead of processor speed to address RAM/CPU timing mismatch
- Predecessor to adding L1/L2/L3 caches, which are central to Meltdown & Spectre
### 1980s Intel CPUs

<table>
<thead>
<tr>
<th>CPU</th>
<th>Year</th>
<th>D/A bits</th>
<th>Speed</th>
<th>Inst</th>
<th>trans/Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>80186/188</td>
<td>1982</td>
<td>16/20</td>
<td>6 MHZ</td>
<td>99</td>
<td>55000/1500 Added clock, timer, interrupts</td>
</tr>
<tr>
<td>80286</td>
<td>1982</td>
<td>16/24</td>
<td>6-8 MHZ</td>
<td>116</td>
<td>134K/1500 MMU, Global and Local Descriptor Tables, protected mode</td>
</tr>
</tbody>
</table>

- Memory management unit (MMU)
  - Allows paging
  - Virtual memory
  - Each process has its own memory space
Virtual Memory

- 286/386 allowed multiple processes
- Each needed own address space
- Memory Management Unit (MMU)
  - Divide physical memory into pages, assign to logical addresses per process
  - Allows swapping pages to disk if needed
- Page Table
  - Contains mapping, changed for each process
  - Page Table Entry (PTE): dirty bit, R/W, read only, security bits, etc.
Virtual Memory (x64)

<table>
<thead>
<tr>
<th>63–48</th>
<th>47–39</th>
<th>38–30</th>
<th>29–21</th>
<th>20–12</th>
<th>11–0</th>
</tr>
</thead>
<tbody>
<tr>
<td>unused</td>
<td>PML4</td>
<td>page</td>
<td>page</td>
<td>page</td>
<td>page</td>
</tr>
<tr>
<td>index</td>
<td>directory</td>
<td>index</td>
<td>directory</td>
<td>table</td>
<td>offset</td>
</tr>
<tr>
<td>pointer</td>
<td>index</td>
<td>index</td>
<td>index</td>
<td>index</td>
<td>index</td>
</tr>
</tbody>
</table>

- Page 4K (or 2MB or newer 1GB)
- 4 levels of tables:
  - CR3 register points to root page table PML4
  - 16 bits of 64 unused (48 bits gives 262 TB)
  - Each page holds 512 Page Table Entries (2^9)
  - Address resolved by walking page tables
  - Transition Lookahead Buffer (TLB) is cache to speed up address resolution
- Meltdown fix causes performance hit because of need to rewrite these
# More 1980s Intel CPUs

<table>
<thead>
<tr>
<th>CPU</th>
<th>Year</th>
<th>D/A bits</th>
<th>Speed</th>
<th>Inst</th>
<th>trans/Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>80386</td>
<td>1985</td>
<td>32/32</td>
<td>12-88 MHZ</td>
<td>142</td>
<td>275K/1000nm Virtual modes for multitasking, external cache</td>
</tr>
<tr>
<td>i486</td>
<td>1989</td>
<td>32/32</td>
<td>50-100 MHZ</td>
<td>150</td>
<td>1.2M/600nm L1 cache, pipelining</td>
</tr>
</tbody>
</table>

- **386**: Real mode, protected mode, virtual mode
  - Real mode was a 4GB *flat* memory model
  - Virtual mode allowed running one or more 8086 mode programs in a *protected* environment
  - Added technical details to make it work well
- **i486**
  - Large transistor jump due to on chip cache and FPU
  - Naming change: Court ruling made number trademarks unenforceable
Process separation

• 286 added protected mode, not very usable
• 386 added technical details to make it work
• Originally three “rings” of protection: 0 to 3
  • Windows, Linux, more: Kernel is Ring 0, User is Ring 3
    • Others not used (Paging only knows ring 012 vs 3 for security)
  • More:
    • 1985: SMM (called “Ring -2” now), suspends normal CPU, runs special CPU
    • 2005: Intel VT-x (AMD-V) added virtualization hypervisor, “Ring -1”
    • 2008: Intel Management Engine (ME) (AMD PSP) added “Ring -3”
      • Significant exploit found in 2017
    • More?
• Invalid access (page fault) traps to kernel
  • Used for protection
  • Used to load swapped out memory
  • Used to grow stack, etc.
Process separation II

- Kernel memory stores valuable things
  - Usually mapped to high half of memory space
  - Process switching involves setting up user page tables

- User should not be able to read kernel memory
  - Meltdown is flaw bypassing this
RAM vs CPU performance
Cache

- Starts as single cache
  - Later split into data and instruction

- Originally external
  - Then internal, then Level 1 and 2
  - Now L1, L2, L3, cache/core, shared

<table>
<thead>
<tr>
<th>Feature</th>
<th>Dynamic RAM (DRAM)</th>
<th>Static RAM (SRAM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circuit</td>
<td>1 capacitor</td>
<td>Flip-flop (6 transistors)</td>
</tr>
<tr>
<td>Transfer</td>
<td>Slower than CPU</td>
<td>Fast as CPU</td>
</tr>
<tr>
<td>Latency</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Density</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Energy</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Cost</td>
<td>Cheap</td>
<td>Expensive</td>
</tr>
</tbody>
</table>
Cache details

• Each line: valid bit, dirty bit, tag, data
  • Tag: who is here?

• Associative
  • only some lines map to a fixed memory spot
  • faster lookup – don’t need to search all cache

• Coherence

• Modern: 64 byte lines
• Experiment: double loop order timing
SDRAM details

<table>
<thead>
<tr>
<th>CPU</th>
<th>Year</th>
<th>Bus clock MHZ</th>
<th>prefetch</th>
<th>Data Rate (MT/s)</th>
<th>Rate (GB/s)</th>
<th>Voltage (V)</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDR</td>
<td>1996</td>
<td>100-166</td>
<td>1n</td>
<td>100-166</td>
<td>0.8-1.3</td>
<td>3.3</td>
<td>Single data rate</td>
</tr>
<tr>
<td>DDR</td>
<td>133-200</td>
<td>2n</td>
<td>266-400</td>
<td>2.1-3.2</td>
<td>2.5/2.6</td>
<td>Double, rising + falling</td>
<td></td>
</tr>
<tr>
<td>DDR2</td>
<td>266-400</td>
<td>4n</td>
<td>533-800</td>
<td>4.2-6.4</td>
<td>1.6</td>
<td>Double clock speed</td>
<td></td>
</tr>
<tr>
<td>DDR3</td>
<td>2007</td>
<td>533-800</td>
<td>8n</td>
<td>1066-1600</td>
<td>8.5-14.9</td>
<td>1.35/1.5</td>
<td>60% power, temp varies</td>
</tr>
<tr>
<td>DDR4</td>
<td>2014</td>
<td>1066-1600</td>
<td>8n</td>
<td>2133-3200</td>
<td>17-21.3</td>
<td>1.2</td>
<td>4 transfers/clock</td>
</tr>
</tbody>
</table>

- SDRAM = Synchronous Dynamic RAM, syncs to CPU timing, faster
- Internal rate 100-166 MHZ for SDRAM, 133-200 MHZ for rest
- Async timing
  - RAS - delay between row strobe and column strobe
  - CAS - delay between column strobe and data availability

- Sync timing
  - CL – cycles between column strobe to data access
  - T<sub>RCO</sub> – cycles between opening a row and accessing columns
  - T<sub>R</sub>P – cycles between issuing precharge and opening next row
  - T<sub>RAS</sub> – cycles between row active command and issuing precharge command
RAM and Architecture

- Old PC (1990ish)
  - Northbridge, Southbridge, Front Side Bus

- Over time more things moved into CPU
  - For speed
  - Memory controller now in CPU
  - Video decoder, driver in CPU
  - GPU built in
FPU aside

<table>
<thead>
<tr>
<th>CPU</th>
<th>Year</th>
<th>Speed</th>
<th>Inst</th>
<th>trans/Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>8087</td>
<td>1980</td>
<td>5 MHZ</td>
<td>83</td>
<td>45,000/3000 nm</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Not IEEE 754, 80 bit floating non std</td>
</tr>
<tr>
<td>80287</td>
<td>1983</td>
<td>6,8,10 MHZ</td>
<td>+1</td>
<td>1500 nm</td>
</tr>
<tr>
<td>80387</td>
<td>1986</td>
<td>16 MHZ</td>
<td>+12</td>
<td>Added SIN, COS</td>
</tr>
<tr>
<td>80487</td>
<td>1991</td>
<td>25 MHZ</td>
<td>+0</td>
<td>1.19M/1000 nm</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Contained full 486DX CPU, took over PC</td>
</tr>
</tbody>
</table>

- Added \( +\times*\sqrt \) 
- 8 level stack \( \text{st0 - st7} \) 
- 20\%-500\% faster than CPU 
- 50,000 FLOPS, 2.4 Watts 
- Led to IEEE 754, released 1985
## Floating point performance – cycle counts

<table>
<thead>
<tr>
<th>Chip</th>
<th>FADD</th>
<th>FMUL</th>
<th>FDIV</th>
<th>FXCH</th>
<th>FCOM</th>
<th>FSQRT</th>
<th>Max Clock (MHz)</th>
<th>Peak FMUL (millions/sec)</th>
<th>FMUL rel. 5 MHz 8087</th>
</tr>
</thead>
<tbody>
<tr>
<td>8087</td>
<td>70...100</td>
<td>90...145</td>
<td>193...203</td>
<td>10...15</td>
<td>40...50</td>
<td>180...186</td>
<td>5 → 10</td>
<td>0.034...0.055 → 0.100...0.111</td>
<td>1 → 2× as fast</td>
</tr>
<tr>
<td>80287 (original)</td>
<td>70...100</td>
<td>90...145</td>
<td>193...203</td>
<td>10...15</td>
<td>40...50</td>
<td>180...186</td>
<td>6 → 12</td>
<td>0.041...0.066 → 0.083...0.133</td>
<td>1.2 → 2.4×</td>
</tr>
<tr>
<td>80387 (and later 287 models)</td>
<td>23...34</td>
<td>29...57</td>
<td>88...91</td>
<td>18</td>
<td>24</td>
<td>122...129</td>
<td>16 → 33</td>
<td>0.280...0.552 → 0.580...1.1</td>
<td>~10 → 20×</td>
</tr>
<tr>
<td>80486 (or 80487)</td>
<td>8...20</td>
<td>16</td>
<td>73</td>
<td>4</td>
<td>4</td>
<td>83...87</td>
<td>16 → 50</td>
<td>1.0 → 3.1</td>
<td>~18 → 56×</td>
</tr>
<tr>
<td>Cyrix 6x86, MII</td>
<td>4...7</td>
<td>4...6</td>
<td>24...34</td>
<td>2</td>
<td>4</td>
<td>59...60</td>
<td>66 → 300</td>
<td>11...16 → 50...75</td>
<td>~320 → 1400×</td>
</tr>
<tr>
<td>AMD K6(I,II,III)</td>
<td>2</td>
<td>2</td>
<td>21...41</td>
<td>2</td>
<td>3</td>
<td>21...41</td>
<td>166 → 550</td>
<td>83 → 275</td>
<td>~1500 → 5000×</td>
</tr>
<tr>
<td>Pentium, P MMX</td>
<td>1...3</td>
<td>1...3</td>
<td>39</td>
<td>1 (0*)</td>
<td>1...4</td>
<td>70</td>
<td>60 → 300</td>
<td>20...60 → 100...300</td>
<td>~1100 → 5400×</td>
</tr>
<tr>
<td>Pentium Pro</td>
<td>1...3</td>
<td>2...5</td>
<td>16...56</td>
<td>1 (0*)</td>
<td>1</td>
<td>28...68</td>
<td>150 → 200</td>
<td>30...75 → 40...100</td>
<td>~1400 → 1800×</td>
</tr>
<tr>
<td>Pentium II / III</td>
<td>1...3</td>
<td>2...5</td>
<td>17...38</td>
<td>1 (0*)</td>
<td>1</td>
<td>27...50</td>
<td>233 → 1400</td>
<td>47...116 → 280...700</td>
<td>~2100 → 13000×</td>
</tr>
<tr>
<td>Athlon (K7)</td>
<td>1...4</td>
<td>1...4</td>
<td>13...24</td>
<td>1 (0*)</td>
<td>1...2</td>
<td>16...35</td>
<td>500 → 2330</td>
<td>125...500 → 580...2330</td>
<td>~9000 → 42000×</td>
</tr>
<tr>
<td>Athlon 64 (K8)</td>
<td>1...4</td>
<td>1...4</td>
<td>13...24</td>
<td>1 (0*)</td>
<td>1...2</td>
<td>16...35</td>
<td>1000 → 3200</td>
<td>250...1000 → 800...3200</td>
<td>~18000 → 58000×</td>
</tr>
<tr>
<td>Pentium 4</td>
<td>1...5</td>
<td>2...7</td>
<td>20...43</td>
<td>multiple cycles</td>
<td>1</td>
<td>20...43</td>
<td>1300 → 3800</td>
<td>186...650 → 543...1900</td>
<td>~11000 → 34000×</td>
</tr>
</tbody>
</table>
### Other numerical additions

<table>
<thead>
<tr>
<th>CPU</th>
<th>Year</th>
<th>Inst</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMX</td>
<td>1997</td>
<td>~50</td>
<td>8 registers, overlays FPU registers, double performance, integer only</td>
</tr>
<tr>
<td>3DNOW! (AMD)</td>
<td>1998</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSE</td>
<td>~50</td>
<td>8(now 16) 128-bit registers, XMM0-15. Only FP32</td>
<td></td>
</tr>
<tr>
<td>SSE2</td>
<td>2001</td>
<td>144</td>
<td>Adds FP64, I64, I32, I16, I8</td>
</tr>
<tr>
<td>SSE3</td>
<td>2004</td>
<td>13</td>
<td>Horizontal and vertical register manipulation</td>
</tr>
<tr>
<td>SSSE3</td>
<td>2006</td>
<td>16</td>
<td>64 bit MMX or 128 bit XMM registers</td>
</tr>
<tr>
<td>SSE4 (4.1 / 4.2)</td>
<td>2006</td>
<td>47/7</td>
<td>POPCNT, LZCNT, CRC32, string and text</td>
</tr>
<tr>
<td>AVX</td>
<td>2008</td>
<td>13</td>
<td>16 YMM registers (8 FP32, or 4 FP64)</td>
</tr>
<tr>
<td>AVX2</td>
<td>2013</td>
<td>30</td>
<td>Integer command to 256 bits, 3 operand support, more</td>
</tr>
<tr>
<td>AVX-512</td>
<td>2015</td>
<td>512-bit register extensions, 4 operands</td>
<td></td>
</tr>
</tbody>
</table>
Pipelining

• Process instructions in series
  • Overlapping work in core to use all pieces simultaneously
• Classic stages from 1956-61 IBM project
  • Fetch
  • Decode + register fetch
  • Execute
  • Memory access
  • Register and memory writeback
• AVR and PIC have 2 stages
• Intel has up to 31 stages
• Xelerated X10q has over 1000 stages!
• Bubbles: caused by data dependencies, cache misses…
486 Pipelining details

- 486 had two decode cycles, allowing more complex decoding
- Desired to get one instruction per clock
- Needed cache to get data prepared
- Pipeline:
  - Fetch – when needed, get entire cache line
    Averages 5 instructions / 16 byte line
  - Decode1 – process up to 3 instruction bytes
  - Decode2 – finish decoding, compute addresses
    5% of instructions need D2
  - Execute – ROM microprogram executes instruction
  - WriteBack – writeback to register file or RAM
Modern pipelines

- Pipelines getting longer (10-30 stages)

- **Branch prediction** tries to keep pipeline full by *speculatively* executing code

- Misprediction costly
  - Must scrap work, refill pipeline
Branch prediction

• To keep pipelines full, speculative execution via branch prediction
• Branch Target Buffer (BTB) predicts where branch will go
• Intel does not publish, people do reverse engineer via testing
• One example:

• Spectre attack tricks the branch predictor into leaking info
## 1990s Intel CPUs

<table>
<thead>
<tr>
<th>CPU</th>
<th>Year</th>
<th>D/A bits</th>
<th>Speed</th>
<th>trans/Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pentium</td>
<td>1993</td>
<td>32/32</td>
<td>60-300 M</td>
<td>3.1M (4.5 MMX)/800nm, 1&lt;sup&gt;st&lt;/sup&gt; <strong>superscalar</strong> design, <strong>dual</strong> integer pipelines, RDTSC, MSR, CUPID</td>
</tr>
<tr>
<td>Pentium Pro</td>
<td>1995</td>
<td>32/36</td>
<td>200 MHZ</td>
<td>5.5M / 350nm, Out of Order (OoO), 14 stage pipeline, 256KB L2 cache, conditional moves, PAE (64 GB RAM), microcode updatable, register renaming</td>
</tr>
<tr>
<td>Pentium II</td>
<td>1997</td>
<td>32/36</td>
<td>450 MHZ</td>
<td>7.5M / 350-180nm, 14 stage pipeline, MMX, Xeon, Celeron, 512 KB L2</td>
</tr>
<tr>
<td>Pentium III</td>
<td>1999</td>
<td>32/36</td>
<td>1GHZ</td>
<td>9.5-22M / 180nm, 10 stage pipeline, L2 on die</td>
</tr>
</tbody>
</table>

- Single pipeline is scalar
- **Superscalar** sends instructions through multiple parallel execution units
- **Out of order** (OoO) – reorders (micro) ops on the fly to remove bubble adding dependencies from pipeline
  - Both add to cost and complexity
- Pentium FDIV bug led to Intel **microcode update** ability in following processors
- Dual pipeline P5 had “U” and “V” integer pipelines. Certain instructions could only go in one, and there were rules for pairing. Handwritten assembly was much faster than compilers.
## 2000s Intel CPUs

<table>
<thead>
<tr>
<th>CPU</th>
<th>Year</th>
<th>D/A bits</th>
<th>Speed</th>
<th>trans/Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pentium 4</td>
<td>2000</td>
<td>32/36</td>
<td>2-3.2 GHZ</td>
<td>42-188M / 180-90nm</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>20 stages, <strong>Hyperthreading</strong></td>
</tr>
<tr>
<td>Prescott (P4 variant)</td>
<td>2004</td>
<td>64/48</td>
<td>125M/90nm</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>64 bit</strong>, 31 stages</td>
</tr>
<tr>
<td>Pentium D</td>
<td>2005</td>
<td>3.2 GHZ</td>
<td>230M/65 nm</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>First <strong>dual core</strong></td>
</tr>
</tbody>
</table>

- Hyperthreading – Intel proprietary simultaneous multithreading
  - OS sees two CPU cores where there is only one
  - CPU core parallelizes performance via superscalar architecture
- 2005, Percival shows cache timing attacks between threads
- First 64 bit x86 Intel chips
  - Addresses 48 bit for now
  - 31 stages!
- Instructions now 1 to 15 bytes in length! (well, technically, up to infinity length)
  - Prefixes such as 0x66 (operand size override) can be repeated
  - 286 imposed 10 byte limit, 386 and higher impose a 15 byte limit

### Correct

$$\frac{4,195,855}{\sqrt{1,797,797}} = 1.333820449136241002$$

### FDIV bug

$$\frac{4,195,855}{\sqrt{1,797,797}} = 1.333739068902037589$$

4 programs, 2 sending instructions, core executing them as needed in multiple virtual cores
## Intel Core i7

<table>
<thead>
<tr>
<th>CPU</th>
<th>Year</th>
<th>D/A bits</th>
<th>Speed</th>
<th>trans/Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nehalem</td>
<td>2008</td>
<td>64/48</td>
<td>731M/45nm</td>
<td>First i7, 20-24 stage, DDR3, 4-12 MB L3 cache</td>
</tr>
<tr>
<td>Sandy Bridge</td>
<td>2011</td>
<td>64/48</td>
<td>504M/32nm</td>
<td>14-19 stages, 1500 decoded <strong>micro-op cache</strong> (instructions pass 5 stages if cached), added <strong>ring bus</strong>, most significant leap in 7 years</td>
</tr>
<tr>
<td>Skylake</td>
<td>2015</td>
<td>64/48</td>
<td>Billion?/14 nm TriGate</td>
<td>1st to use DDR4</td>
</tr>
<tr>
<td>Coffee Lake</td>
<td>2017</td>
<td>64/48</td>
<td>3.7 G(4.7 Turbo)</td>
<td>Billions+/14 nm TriGate</td>
</tr>
</tbody>
</table>

- Instructions broken into micro-ops (uops) for reordering, execution
- Ring bus: interconnect between Cores, Graphics, Last Level Cache, System Agent
  - 32 byte data ring, request ring, acknowledge ring, snoop ring
  - Fully pipelined at core frequency
  - Sophisticated distributed arbitration to handle coherency, ordering
  - Each component has a “stop” on the ring bus
- System Agent (houses traditional NorthBridge)
  - PCI express, DMI, memory controller, Power control, thermal control
- L3 cache 96 GB/s, ~30 cycles latency
Oct 2017: Intel 8th gen 8700K

- 6 cores, 12 threads
- MMX, AES-NI, CLMUL, FMA3, SSE, SSE2, SSE3, SSSE3, SSE4, SSE 4.1, SSE 4.2, AVX, AVX2, TXT, TSX, SGX, MPX, VT-x, VT-d
- 3.7 GHZ clock (4.7 G Turbo mode)
- GPU 630 (1.2 GHZ, 24 units, 4K 60fps, H265, VP9, DP 1.2, HDMI 2.0, HDCP 2.2, DirectX 12, OpenGL 4.5, 3 displays)
- PCI Express 3.0, 1x16 or 2x8 or 1x8 + 2x4 lanes
- L1 64 kB/core (32 Data, 32 Instruction), 8 way set assoc
- L2 256 kB/core, 4 way set assoc
- L3 12 MB shared, up to 16 way set assoc
- L4 128 MiB (Iris Pro GPU only)
- Bus 8 GT/s DMI3
- 95W
- 37.5x37.5 mm die size
- DDR4-2666, up to 64 GB, 2 channels
8700K
Microprocessor Transistor Counts 1971-2011 & Moore's Law

The graph shows the trend of transistor count in microprocessors over the years, doubling every two years, as predicted by Moore's Law. The X-axis represents the date of introduction, while the Y-axis represents the transistor count. Key models and their corresponding transistor counts are marked on the graph, illustrating the exponential growth in processor technology from 1971 to 2011.
End of Moore’s law?

• Plenty of good, accurate articles
  • Scientific American, Wired

• Physics:
  • 1985: 1000 nm feature
  • 2018: 14 nm feature
  • Silicon atom: 0.2 nm
  • Quantum mechanics: smaller features “fuzzier”
  • 5 GHz: light travels 6 cm per clock tick, electricity ~80% of this in wire.
    • Cannot have RAM too far from CPU
  • CPU Die size ~ 400 mm^2, so light speed an issue for higher clocks
  • Heat hard to remove, so lower voltage, moving to diamond substrates, etc...

• Economics
  • $8B for current fab plant.
Meltdown and Spectre

- 20 year old flaws in many, many chips,
  - Intel, ARM, AMD, IBM, more

- Found by researchers from four teams during 2017
  - Intel, Google, Amazon, AMD, Microsoft, Apple, others working behind closed doors to fix
  - Publicly disclosed early Jan 2018

- Biggest chip security flaw probably ever
- Will likely cost billions in liability for many companies
- Technical read: https://googleprojectzero.blogspot.com/
Meltdown

- “Melts” boundaries between hardware enforced security layers
- User mode programs can read kernel mode data.
- Affects:
  - Software: Windows, Linux, iOS, macOS, cloud services, Android, others
  - Hardware: Intel x86/64, most ARM, IBM Power, others
- Not affected: CPUs not doing speculative execution
  - Many Qualcomm, some ARM, Raspberry Pi
- Not hard to do.
- CVE-2017-5754 : rogue data cache load
Meltdown details

- **Idea:**
  - Speculatively execute instruction reading protected memory
  - Use speculative data to load cache line of unprotected memory
  - Read fails, cache line still touched
  - Look at cache timing to leak protected data

```plaintext
; rcx = protected pointer
; rbx = user accessible pointer
; flush cache before calling
meltdown:
    mov al, byte [rcx]
    shl rax, 12 ; 2^12 = 4096, page size
    jz meltdown ; read till crash
; read user space, trigger cache update
    mov rbx, qword [rbx+rax]
; now read cache timing to determine al
```
Meltdown fix

- Requires kernel changes, isolating kernel memory from user, KPTI
- Context switch requires changing page tables, incurs performance hit
- Recent patches to Windows, Linux, iOS, others
- Intel microcode patch to help mitigate
- Speculation on performance
  - Mostly negligible, depends on workload
Spectre

• Breaks the isolation between applications
  • Attacker can trick error-free programs into leaking their secrets
• Affects:
  • Software: Windows, Linux, iOS, macOS, cloud services, Android, others
  • Hardware: Intel x86/64, AMD, most ARM, IBM Power, others
• Not affected: CPUs not doing speculative execution
  • Many Qualcomm, some ARM, Raspberry Pi
• Hard to exploit, but also much harder to fix than meltdown.
  • Requires careful code analysis and recompilation
• CVE-2017-5753 : bounds check bypass
• CVE-2017-5715 : branch target injection
Spectre details

• Spectre: name from “Speculative Execution”, note branch
• Tricks branch predictor to mispredict safety check
• Use mispredicted value to speculatively read data, taint cache
• **Leaks data**
  • Ironically, security checks in code made this easier to do
  • Proof-of-concept JavaScript exploit allowed browser to read user process data
  • Google patched Chrome to address it, other vendors patched also

```c
if (untrusted_index < array1_length) {
    unsigned char value = array1[untrusted_index];
    unsigned char value2 = array2[value * 64];
}
```
Spectre “fix”

- Require code changes to insert “barriers” between certain actions
- Hardened browsers against JavaScript
- LLVM patch, ARM speculation barrier header
- MSVC has a fix, using /Qspectre switch
Vendor Fixes

• Intel
  • Working on mitigations via microcode updates

• Apple
  • iOS, macOS High Sierra, tvOS, Safari on Sierra and El Capitan
  • Addresses Meltdown, Spectre to some extent

• Microsoft
  • X64 only?
  • Meltdown and Spectre variant 1 only, not variant 2

• Google
  • Pushing “retpoline” binary modification to others to mitigate Spectre

• Amazon
  • Rolled out new, slower virtual machine infrastructure.

• Linux
  • Patches to 4.4 and 4.9 trees
  • Addresses Meltdowm only

• Many others
Meltdown demo

Steps:
Compute cached/uncached timing threshold
Prepare user memory
For each byte to test
  do ~1000 times:
    Flush cache
    Speculate
    Tally cache timing
  Pick most likely value for this byte

Code on GitHub: https://github.com/ChrisLomont/Meltdown
Questions?