CS 152
Computer Architecture and Engineering
Lecture 20 -- Dynamic Scheduling III

2014-4-8
John Lazzaro
(not a prof - “John” is always OK)

TA: Eric Love

www-inst.eecs.berkeley.edu/~cs152/
Intel Inside: Dynamic Execution

The IA-32 ISA: How to adapt a complex instruction set to dynamic techniques.


Short Break

Sandy Bridge: Dynamic execution in today’s Core and Xeon product lines.

Limits to Parallelism: How fast can the perfect dynamic execution design go?
When configured to conceal “legacy” modes ...

Two differences between MIPS and IA-32 integer programmer’s model:

- Only 8 registers
  - Thus, register renaming.
- ALU instructions set bits in EFLAGS register. Branches are taken based on EFLAGS values.

Example:

Zero Flag (ZF): ZF = 1 if last ALU instruction result was zero, otherwise ZF = 0.
Even the simple instructions are complex:

```
ADC m32, r32:
```

Read the memory location m32, add the contents of register r32 to it, along with the carry bit in EFLAGS, and store the result back in m32.

Rewriting this instruction in MIPS takes at least four instructions: 1 load, 2 adds, 1 store. More for a complex m32 addressing mode.
The following addressing modes suggest uses for structures commonly used by programmers in high-level languages and assembly language.

The base, index, and displacement components can be used in any combination, and any of these components can be a register or memory operand.

- A base alone represents an indirect offset to the operand. Because the address of the beginning of the array is sometimes called an absolute or static base, a base register and a displacement can be used together for two distinct purposes:
  - To access a field of a record: the base register holds the address of the beginning of the record, while the displacement is a static offset to the field.
  - As an index into an array when the element size is 2, 4, or 8 bytes—The displacement component specifies the index into the array.

- A scale factor may be used only when an index is used.

- A displacement alone represents a direct (uncompressed) offset to the operand. When the ESP or EBP register is used as the base, the displacement is encoded in the instruction, this form of addressing is called stack-based addressing.

- Up to 2 adds and a shift for a memory operand ...

### Table: Addressing Modes

<table>
<thead>
<tr>
<th>Base</th>
<th>Index</th>
<th>Scale</th>
<th>Displacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAX</td>
<td>EAX</td>
<td>1</td>
<td>None</td>
</tr>
<tr>
<td>EBX</td>
<td>EBX</td>
<td>2</td>
<td>8-bit</td>
</tr>
<tr>
<td>ECX</td>
<td>ECX</td>
<td>4</td>
<td>16-bit</td>
</tr>
<tr>
<td>EDX</td>
<td>EDX</td>
<td>8</td>
<td>32-bit</td>
</tr>
<tr>
<td>ESP</td>
<td>EAX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EBP</td>
<td>EBX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ESI</td>
<td>ECX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EDI</td>
<td>EDX</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Offset = Base + (Index * Scale) + Displacement
And then there are complex instructions ... 

**PUSHA:** Push all 8 integer registers onto the stack

<table>
<thead>
<tr>
<th>Stack Growth</th>
<th>Before Pushing Registers</th>
<th>After Pushing Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td></td>
<td></td>
</tr>
<tr>
<td>n - 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>n - 8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>n - 12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>n - 16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>n - 20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>n - 24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>n - 28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>n - 32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>n - 36</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Rewrite PUSHA in MIPS? > 16 instructions.

Some IA-32 instructions require 100s of MIPS instructions ...
IA-32, instruction decode

IA-32 instructions are 1 to 17 bytes in length. To determine an instruction’s length? Parse it!

<table>
<thead>
<tr>
<th>Instruction Prefixes</th>
<th>Opcode</th>
<th>ModR/M</th>
<th>SIB</th>
<th>Displacement</th>
<th>Immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Up to four prefixes of 1 byte each (optional)</td>
<td>1-, 2-, or 3-byte opcode</td>
<td>1 byte (if required)</td>
<td>1 byte (if required)</td>
<td>Address displacement of 1, 2, or 4 bytes or none</td>
<td>Immediate data of 1, 2, or 4 bytes or none</td>
</tr>
</tbody>
</table>

Figure 2-1. Intel 64 and IA-32 Architectures Instruction Format

Creeping features: Prefix bytes for specifying atomicity, for hinting branch direction, for 8086 mode ...
Dynamic Execution?
Idea: Translate IA-32 opcodes to RISC

Micro-ops:

RISC-like instructions that reference the 8 architected registers, plus “temporary” registers.

Translate?

For IA-32 instructions that map into a small number of micro-ops, do translation after decode.

EFLAGS?

Treat each EFLAG bit as an architected register.
Translation example ...

ADC m32, r32: // for a simple m32 address mode

Becomes:

LD T1 0(EBX);  // EBX register point to m32
ADD T1, T1, CF;  // CF is carry flag from EFLAGS
ADD T1, T1, r32; // Add the specified register
ST 0(EBX) T1;   // Store result back to m32

Instruction traces of IA-32 programs show most executed instructions require 4 or fewer micro-ops.

Translation for these ops are cast into logic gates, often over several pipeline cycles.
Complex instructions? Microcode ROM

Instructions like PUSHA are sent to a sequencer that clocks a long sequence of micro-ops out of a ROM.

ROM entry point of the micro-ops for an IA-32 instruction.

"The micro-op engine"
Power-hungry decode? Cache micro-ops!

Most IA-32 implementations cache decoded instructions, to avoid repetitive decoding of the complicated instruction format.

Implementations vary in the size of the caches, and how the micro-op cache interacts with the branch predictor and the normal instruction cache.

Macro-Fusion: Another idea to save power, by reducing the number of micro-ops in a translation. An implementation defines micro-ops that captures the semantics of IA-32 instruction pairs that occur in sequence, and maps these pairs to one micro-op.
Intel Pentium IV (2001)

1.5 GHz main clock

Execute loop has 2X clock

55 Watts

0.15um process

42M transistors
A. Front-End

The front-end of the Pentium® 4 processor consists of several units, as shown in the upper part of Fig. 3. It has the instruction translation lookaside buffer (ITLB), the front-end branch predictor (labeled here Front-End BTB), the IA-32 instruction decoder, the trace cache, and the microcode ROM.

1) Trace Cache:

The trace cache is the primary or Level 1 (L1) instruction cache of the Pentium® 4 processor and delivers up to three micro-operations (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 machine 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–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, 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 redecode 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, instructions bypass the instruction decoder most of the time, thereby reducing misprediction latency and allowing a simpler decoder that processes only one IA-32 instruction per clock cycle.

The 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 Intel’s Pentium IV (2001) Cyan blocks “fix” IA-32 issues. Green blocks are for OoO execution. Gold blocks are fast execute pipes.
Branch predictor steers instruction prefetch from L2. Maximum decoder rate is 1 IA-32 instruction per cycle.

**Trace cache**: 12K micro-ops: 2K lines hold 6 decoded micro-ops in executed (trace) order, that follow taken branches and function calls.

**Prediction**: Each line includes a prediction of next trace line to fetch. The front-end prefetcher takes over on a trace cache miss.

**Microcode ROM**: Trace cache jumps into ROM if IA-32 instruction maps to > 4 micro-ops.
Out of order logic

Fig. 6. Important logic loops.

Fig. 3. Pentium \textsuperscript{4} processor consists of several units and the L1 data cache. On the right of the diagram is the ALU adder is a propagate–generate–kill addition algorithm as well as sign extension and a full set of logic functions. The execution unit output.

Fig. 8. ALU inputs, loading, and bypass loop.

126 instructions in flight

128 integer physical registers
128 floating point physical registers

3 micro-ops per cycle from trace cache

6 operations scheduled per cycle

3 micro-ops retired per cycle

128 floating point physical registers

3 micro-ops per cycle from trace cache

Out of order logic

126 instructions in flight
Execution Unit:
Simple integer ALUs runs on both edges of the 1.5 Ghz clock.

Data cache speculate on L1 hit, and has a 2 cycle load-use delay. 48 load + 24 store buffers also within this critical loop.

2X Bypass Network:
Fast ALUs can use their own results on each edge.
Key trick: Staggered ALUs

Pipeline registers added to carry chain. A 32-bit adder is computing parts of 3 different operations at the same time.
In context: complete datapath

The logic loops used 90% of the time run at 3 GHz, but most of the chip runs at 1500 MHz.
The trace cache was too much of a departure from Pentium III, and the existing code base missed the cache too often. This was particularly bad because the Pentium IV pipeline had so many stages!
Recall: Limits to super-pipelining ...

FO4 Delays

Historical limit: about 12 FO4s

CPU Clock Periods 1985-2005

MIPS 2000
5 stages

Pentium Pro
10 stages

Pentium 4
20 stages

Power wall: Intel Core Duo has 14 stages

FO4: How many fanout-of-4 inverter delays in the clock period.

Thanks to Francois Labonte, Stanford
The Pentium IV was the chip that foreshadowed the "power wall".

Upper-management pressure for a high clock rate (for marketing) pushed the design team to use too many pipeline stages, and performance (and power) suffered.

Intel recovered by going back to their earlier Pentium Pro out-of-order design ... 

Many elements of the Pentium IV are innovative ... and were reintroduced in Sandy Bridge (2011-onward).
It's all in the timing ...

The dot-com tech stock "bubble"
Break
Sandy Bridge

Out-of-order core

LLC and ring stop

FP/SSE
AVX
unit

2MB L3
Sandy Bridge

Figure 3. Complete block diagram of Sandy Bridge CPU.

**Branch Pred**
- **Fetch Unit**
  - **Micro-code**
  - **Complex Decoder**
  - **Simple Decoder**
  - **x86 predecode**
  - **Rename/Dispatch**
  - **32KB Instruction Cache**
    - >20 bytes
    - 4 instructions
    - 4 micro-ops
  - **I-TLB**
  - **1.5K Micro-op (L0) Cache**
  - 4 micro-ops

**Retire Unit**
- **Load Unit**
- **Load/Store**
- **ALU/Branch**
- **ALU/Divide**
- **Integer ALU**
- **FP Multiply**
- **FP Add**
- **64-bit Integer Registers**
  - 64 micro-ops
  - 64 micro-ops
  - 64 micro-ops
  - 64 micro-ops
  - 64 micro-ops
  - 64 micro-ops
  - 64 micro-ops

**Shared L2 Cache**
- **D-TLB**
  - 32KB Data Cache
  - 256 micro-ops
  - 256 micro-ops
  - 256 micro-ops
  - 256 micro-ops
  - 256 micro-ops
  - 256 micro-ops
  - 256 micro-ops

**AVX**
- AVX instructions support one 256-bit load, 128-bit AVX load per cycle, 128-bit AVX store per cycle. Sandy Bridge can perform two loads and one store per cycle, doubling the load bandwidth. This change helps support the AVX instructions, which are 435GB/s.
- Sandy Bridge implements a unique design for its cache hierarchy.
- Instead of the single L3 cache used in Nehalem and Westmere, Sandy Bridge divides the L3 cache into four blocks, allowing each CPU to see a single, large L3 cache.
- As a result, the average L3 cache latency is 27 to 31 cycles compared with about 36 cycles in Nehalem.
- The three hops can take as little time as a single transfer on a PCI Express bus at 26 to 31 cycles.
- Each station in the ring connects only to the next station, the ring segments are short enough to operate at the full CPU speed.
- Furthermore, each CPU is physically adjacent to one of the cache blocks, there is no association of data between the CPU and its neighboring cache block.
- Thus, the new cache reduces the amount of data that was previously called the north bridge.

**Ring Around the Cores**
- Sandy Bridge contains a new component, the system agent, that takes on the role of the north bridge.
- Instead, each CPU simply sees a single, large L3 cache.
- The memory controller, PCI Express, display interfaces, and other components are in the south bridge.
- Instead of the single L3 cache used in Nehalem and Westmere, Sandy Bridge divides the L3 cache into four blocks, allowing each CPU to see a single, large L3 cache.
- Diodes use rings to interconnect instead of a ring, the ring itself must handle three hops.
- These extra hops add latency, but given the fast speed of the ring, two or three hops can take as little time as a single transfer on a PCI Express bus.
- Each station in the ring connects only to the next station, the ring segments are short enough to operate at the full CPU speed.
- Furthermore, each CPU is physically adjacent to one of the cache blocks, there is no association of data between the CPU and its neighboring cache block.
- Thus, the new cache reduces the amount of data that was previously called the north bridge.

**Intel's Larrabee Redefines GPUs**
- Larrabee redefines GPUs, but it has not previously used a ring in its mainstream processors.
- Other processors use rings to design complex interconnection networks.
Traditional L1 instruction cache does the "heavy lifting". Caches IA-32 instructions.

Decoders can generate 4 micro-ops/cycle. Microcode ROM still a part of decode.

Micro-op cache is 10% of the size of the Pentium IV trace cache. Purpose is power savings (80% of time, decode off).
instructions. The team want accurate branch prediction is critical to its performance. Enabled the team to increase the number of scheduler and the ROB would have to be able to hold a bits wide that file, so only the YMM rename registers need to be 256 bits wide. As a result, accuracy is improved entries and still have room left over. This extra space that the team was able to double the number of BTB placements, looping back or jumping forward several instructions whose micro code was not committed from the data cache. Figure 3 shows, this simplification also saves the ROB to the physical register file, saving power. The Sandy Bridge design also expands the size of the branch prediction unit.

Table 1. Nehalem v

- Nehalem
- Sandy Bridge

<table>
<thead>
<tr>
<th></th>
<th>Nehalem</th>
<th>Sandy Bridge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Buffers</td>
<td>48 entries</td>
<td>64 entries</td>
</tr>
<tr>
<td>Store Buffers</td>
<td>32 entries</td>
<td>36 entries</td>
</tr>
<tr>
<td>Scheduler Entries</td>
<td>36 micro-ops</td>
<td>54 micro-ops</td>
</tr>
<tr>
<td>Integer Rename File</td>
<td>Not applicable</td>
<td>160 registers</td>
</tr>
<tr>
<td>FP Rename File</td>
<td>Not applicable</td>
<td>144 registers</td>
</tr>
<tr>
<td>Reorder Buffer</td>
<td>128 micro-ops</td>
<td>168 micro-ops</td>
</tr>
</tbody>
</table>

Can retire up to 4 micro-ops/cycle. Only 25% more than Pentium IV. However, Sandy Bridge does so much more often ...

Can retire up to 4 micro-ops/cycle. Only 25% more than Pentium IV. However, Sandy Bridge does so much more often ...
Single-thread benchmark: Diminishing returns ...

3D Rendering Performance - Cinebench R11.5
Single Threaded Benchmark - Higher is Better

<table>
<thead>
<tr>
<th>Model</th>
<th>Clock Speed</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>MacBook Pro with Retina Display - Core i7 2.6GHz</td>
<td></td>
<td>1.43</td>
</tr>
<tr>
<td>13-inch MacBook Pro (Early 2011) - Core i7 2.7GHz</td>
<td></td>
<td>1.3</td>
</tr>
<tr>
<td>15-inch MacBook Pro (Early 2011) - Core i7 2.3GHz</td>
<td></td>
<td>1.29</td>
</tr>
<tr>
<td>13-inch MacBook Air (Mid 2012) - Core i7 2.0GHz</td>
<td></td>
<td>1.27</td>
</tr>
<tr>
<td>Mac Pro (Early 2009) - 8-core Xeon 3.00GHz</td>
<td></td>
<td>1.11</td>
</tr>
<tr>
<td>13-inch MacBook Air (Mid 2013) - Core i5 1.3GHz</td>
<td></td>
<td>1.11</td>
</tr>
<tr>
<td>13-inch MacBook Air (Mid 2012) - Core i5 1.8GHz</td>
<td></td>
<td>1.11</td>
</tr>
<tr>
<td>15-inch MacBook Pro (Mid 2010) - Core i7 2.66GHz</td>
<td></td>
<td>1.06</td>
</tr>
<tr>
<td>11-inch MacBook Air (Mid 2012) - Core i5 1.7GHz</td>
<td></td>
<td>1.04</td>
</tr>
<tr>
<td>13-inch MacBook Air (Mid 2011) - Core i5 1.7GHz</td>
<td></td>
<td>1.01</td>
</tr>
<tr>
<td>Mac Pro (Mid 2010) - 8-core Xeon 2.40GHz</td>
<td></td>
<td>0.89</td>
</tr>
<tr>
<td>11-inch MacBook Air (Mid 2011) - Core i5 1.6GHz</td>
<td></td>
<td>0.87</td>
</tr>
<tr>
<td>Mac Pro (Early 2008) - 8-core Xeon 3.00GHz</td>
<td></td>
<td>0.84</td>
</tr>
<tr>
<td>13-inch MacBook Air (Late 2010) - Core 2 Duo 1.86GHz</td>
<td></td>
<td>0.57</td>
</tr>
<tr>
<td>MacBook Air (Late 2008) - Core 2 Duo 1.86GHz</td>
<td></td>
<td>0.49</td>
</tr>
<tr>
<td>11-inch MacBook Air (Late 2010) - Core 2 Duo 1.6GHz/4GB</td>
<td></td>
<td>0.48</td>
</tr>
<tr>
<td>11-inch MacBook Air (Late 2010) - Core 2 Duo 1.4GHz</td>
<td></td>
<td>0.42</td>
</tr>
</tbody>
</table>
Limits to Instruction-Level Parallelism
Recall: Most execution units lie idle

Big Question
Is there instruction level parallelism in single threads yet to be attained?
Or are we near fundamental limits?

For an 8-way superscalar.

Instruction level parallelism available for selected SPEC benchmarks, given a perfect architecture.

Perfect? Branch predictor accuracy of 100%, caches never miss, infinite out-of-order resources ...
Add minor real-world constraints

Reorder buffer windows as shown.

64 issues/cycle max

64 physical registers for int and for float.

Branch predictors near the current state-of-the-art.

Infinite load/store buffers.

Perfect caches
Figure 3.43 The CPI for the 19 SPECCPU2006 benchmarks shows an average CPI for 0.83 for both the FP and integer benchmarks, although the behavior is quite different. In the integer case, the CPI values range from 0.44 to 2.66 with a standard deviation of 0.77, while the variation in the FP case is from 0.62 to 1.38 with a standard deviation of 0.25. The data in this section were collected by Professor Lu Peng and Ph.D. student Ying Zhang, both of Louisiana State University.

GCC: Perfect IPC was 55.

“Real-world” machines were 8-10.

Intel Nehalem was slightly under 1.

Maybe new ideas are needed ... maybe incremental improvements will eventually get us there ... or maybe we should focus on other goals than single-threaded performance.
On Thursday

Beyond Von Neumann ...

| Th 4/10 | Dataflow |

Have fun in section!