Modern Dynamic Instruction Scheduling

Tomasulo Performance

- Observe at the EX stage, how many cycles to execute this code?
- LW R2, 45 (R3)
- ADD R6, R2, R4
- SUB R10, R0, R6
- ADD R10, R10, R12
- Assume load takes 1 cycle, ALU 1 cycle

Tomasulo vs MIPS Pipeline

- How many cycles on the 5-stage MIPS pipeline?

Why does the simple pipeline run faster?

- Still check
- Data forwarding

Review Tomasulo Inst Scheduling

- Both in RS, no contention on CDB or FU
- ADD R2, R2, 45  # R2 = tag p, result = A
- SUB R6, R2, R4  # R4 is ready, - B
- Cycle 1: ADD starts at FU, producing A
- Cycle 2: ADD broadcast p + A
- SUB matches on p and accepts A
- Cycle 3: SUB starts execution, FU calc A-B
- A is produced at cycle 1, but consumed at cycle 3 — unavoidable?

Review Data Forwarding

MIPS pipeline data forwarding:
- FU/MEM = FU
- Why not in Tomasulo?

REG/ROB

ROB bypass

Cycle 2: forward A from FU output to FU input...

But tag broadcasting has one cycle delay!!!

When is it known that A will be ready?

Cycle 1: A is to be ready

Cycle 2: A and the tag are broadcast

If the tag is broadcast one cycle earlier...

Revise Scheduling

- RO1 ADD RO0, RO1
- RO2 ADD RO4, RO5
- RO3 ADD RO2, RO3
- RO4 ADD RO5, RO6

- Add has been read and selected
- ADD to be broadcast, and forwarded to RO2, RO4
- Output is young and selected
- RO2 broadcasts A to RO2, RO4
- RO4 accepts RO2 output, and is selected
- RO4 is wake up and accepts RO2 output, and is selected
- RO2 is wake up and accepts RO4 output, and is selected
- RO3 is wake up and accepts RO4 output, and is selected
- RO1 is wake up and accepts RO4 output, and is selected
- RO0 is wake up and accepts RO4 output, and is selected
- RO0 is wake up and accepts RO4 output, and is selected

One cycle earlier

How to address CDB contention?
How to Handle Variable Latency?

- RS[RS[RS[RS[RS
   1 2 3 4 5
- SELECT
- Cycle n:
- Tag broadcast
- FU of k-cycle latency
- Control data bus
- Cycle n+k

One method: Use result shift register to track latency and control tag/data bus.

Revised Pipeline Stages

- As efficient as MIPS pipeline (instruction throughput)
- With data forwarding and bypassing

Rethink RS and ROB design

- Data broadcasting to RS stations:
  - Broadcasting saves reg-write to reg-read delay
  - Not all n child instructions can receive data simultaneously

However,
- RS and ROB may store duplicate values
- Not all n child instructions may execute next cycle

Rethink ROB Design

- Does every ROB entry store register output?
- Split ROB into ROB + Rename Register

Arch. Regs

Rename Regs

ROB (no reg value)

From FU

Alternative: Register Mapping and Issue Queue

- RS entry
  - op  |   Qj   |   Qk   | bus[k] | Vj | Vk
  - ROB entry
  - Physical: dest, PC, valid, result
- Arch. regs
  - Physical register: Central collection of all register values, architectural or temporary

Register Mapping Table

- Rename architectural register to physical register
- NO real architectural registers (now virtual registers)
- RS = issue queue
- Rename stage: allocate issue queue entry, allocate ROB, allocate physical register
- What is tag now?
**Mis-speculation Recovery**
- RS/RCB: no changes to arch. registers; sub-task clears pipeline and re-fetch
- Fundamental issue:
  - software does not see wrong register contents
- Recovery from mapping
  - map back to the mis-speculation point
- Architectural registers = virtual registers

**Scheduling with Issue Queue**

**Alpha 21264 Pipeline**

**Examples: Intel P6**

**Example: Intel Pentium 4**

**Generic Superscalar Processor Models**

Source: Parashar PhD thesis 1998
Dynamic Scheduling in P6
(Pentium Pro, II, III)

Q: How pipeline 1 to 17 byte 80x86 instructions?

P6 doesn't pipeline 80x86 instructions
- P6 decode unit translates the Intel instructions into
  72-bit micro-operations (+ MIPS)
- Sends micro-operations to reorder buffer & reservation stations
- Many instructions translate to 1 to 4 micro-operations
- Complex 80x86 instructions are executed by a
  conventional microprogram (8K x 72 bits) that issues long
  sequences of micro-operations
- 14 clocks in total pipeline (= 3 state machines)

Parameter 80x86 microops
Max. instructions issued/clock 3 6
Max. instr. complete exec/clock 5
Max. instr. committed/clock 3
Window (Instrs in reorder buffer) 40
Number of reservations stations 20
Number of rename registers 40
No. integer functional units (FUs) 2
No. floating point FUs 1
No. SIMD FPU FUs 1
No. memory FUs 1 load + 1 store

P6 Pipeline
- 14 clocks in total (=3 state machines)
- 8 stages are used for in-order instruction
  fetch, decode, and issue
  - Takes 1 clock cycle to determine length of
    80x86 instructions + 2 more to create the
    micro-operations (uops)
- 3 stages are used for out-of-order execution in
  one of 5 separate functional units
- 3 stages are used for instruction commit

P6 Block Diagram

AMD Athlon
- Similar to P6 microarchitecture
  (Pentium III), but more resources
- Transistors: PIII 24M v. Athlon 37M
- Die Size: 106 mm² v. 117 mm²
- Power: 30W v. 76W
- Cache: 16K/16K/256K v. 64K/64K/256K
- Window size: 40 vs. 72 uops
- Rename registers: 40 v. 36 int +36 fl. Pr.
- BTB: 512 x 2 x 4096 x 2
- Pipeline: 10-12 stages v. 9-11 stages
- Clock rate: 1.0 GHz v. 1.2 GHz
- Memory bandwidth: 1.06 GB/s v. 2.12 GB/s

Pentium III Die Photo
- EBL/EBL - Blue logic, Front, Back
- MAC - Memory Access Buffer
- FPU - Floating Point Unit (4x)
- IUUM - Integer Unit Microprocessor
- VLIU - Vector Logic Interface Unit
- PA - Page Address Block
- DTB - Data TLB
- BAC - Branch Address Calculator
- RST - Register Store Table
- SIMD - Simultaneous ISS
- RS - Reservation Station
- BTB - Branch Target Buffer
- ITU - Instruction Tuple Unit (+3)
- ID - Instruction Decoder
- ROB - Reorder Buffer
- MMU - Memory Management Unit

1st Pentium III,retail 933 MHz; 128 KB 154 mm/in 0.55 mm with 8 layers of aluminum

AMD Athlon
- Similar to P6 microarchitecture
  (Pentium III), but more resources
- Transistors: PIII 24M v. Athlon 37M
- Die Size: 106 mm² v. 117 mm²
- Power: 30W v. 76W
- Cache: 16K/16K/256K v. 64K/64K/256K
- Window size: 40 vs. 72 uops
- Rename registers: 40 v. 36 int +36 fl. Pr.
- BTB: 512 x 2 x 4096 x 2
- Pipeline: 10-12 stages v. 9-11 stages
- Clock rate: 1.0 GHz v. 1.2 GHz
- Memory bandwidth: 1.06 GB/s v. 2.12 GB/s
Pentium 4
- Still translate from 80x86 to micro-ops
- P4 has better branch predictor, more FUs
- Instruction Cache holds micro-operations vs. 80x86 instructions
  - No decode stages of 80x86 as cache hit
  - Called “trace cache” (TC)
- Faster memory bus 400 MHz vs. 133 MHz
- Caches
  - Pentium III: L1I 16KB, L1D 16KB, L2 256 KB
  - Pentium 4: L1I 12K ops, L1D 8 KB, L2 256 KB
  - Block size: PIII 32B vs. P4 128B, 128B vs. 256 bits/dock
- Clock rates
  - Pentium III 1 GHz vs. Pentium IV 1.5 GHz

Pentium 4 features
- Multimedia instructions 128 bits wide vs. 64 bits wide
  - 144 new instructions
- When used by programs?
  - Faster Floating Point: execute 2 64-bit FP per clock
  - Memory FU: 1 128-bit load, 1 128-store / clock to MAX regs
- Using RAMBUS DRAM
  - Bandwidth faster, latency same as SDRAM
  - Cost 2X-3X vs. SDRAM
- ALUs operate at 2X clock rate for many ops
- Pipeline doesn’t stall at this clock rate: ups replay
- Rename registers: 40 vs. 128; Window: 40 vs. 126
- BTB: 512 vs. 4096 entries (45% 1/3 improvement)

Basic Pentium 4 Pipeline

<table>
<thead>
<tr>
<th>TCNet</th>
<th>IP</th>
<th>TC Fetch</th>
<th>Drive</th>
<th>Alloc</th>
<th>Rename</th>
<th>Queue</th>
<th>Schd</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Schd</td>
<td>Schd</td>
<td>Schd</td>
<td>Schd</td>
<td>Schd</td>
<td>Schd</td>
<td></td>
</tr>
<tr>
<td>1-2 trace cache next instruction pointer</td>
<td>10-12 write ups to scheduler</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3-4 fetch ups from Trace Cache</td>
<td>13-14 move up to 6 ups to FU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 drive ups to alloc</td>
<td>15-16 read registers</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 alloc resources (ROB, reg...)</td>
<td>17 FU execution</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7-8 rename arch, reg to 128 physical reg</td>
<td>18 computer flaps e.g. for branch instructions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9 put renamed ups into queue</td>
<td>19 check branch output</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>20 drive branch check result to transient</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Block Diagram of Pentium 4 Microarchitecture

- BTB = Branch Target Buffer (branch prediction)
- ITLB = Instruction TLB, Trace Cache = Instruction cache
- RF = Register File, AGU = Address Generation Unit
- “Double jumped ALU means ALU clock rate 2X = 2X ALUs

From “Pentium 4 (Partially) Revealed,” Microprocessor Report, 8/28/00

Pentium 4 Die Photo
- 42M Xbars
- PIII: 26M
- 217 mm²
- PIII: 106 mm²
- L1 Execution Cache
  - Buffer 12,000
  - Micro-ops
  - 8KB data cache
  - 256KB L2

Workstation Microprocessors 3/2001

- Max issue: 4 instructions (many CPUs)
- Max rename registers: 128 (Pentium 4)
- Max BH-TI 4K x 9 (Alpha 21264B), 16K x 2 (Ultra III)
- Max Windows size (CEO): 126 instructions (Pentium 4)
- Max Pipeline: 22/24 stages (Pentium 4)

Benchmarks Pentium 4 v. PIII v. Athlon

- SPECbase2000
  - Intel P4 @ 1.5 GHz: 524, PIII @ 1 GHz: 454, AMD Athlon 1 GHz: 379
  - FP: P4 @ 1.5 GHz: 349, PIII @ 1 GHz: 299, AMD Athlon 1 GHz: 204
- WorldBench 2000 benchmark (business) PC World magazine, Nov. 20, 2000 (bigger is better)
  - P4 @ 1.66, PIII @ 1.67, AMD Athlon 1.8 GHz
- Quake 3 Arena: P4, 1.72, Athlon 1.5 GHz
- SYSmark 2000 composite: P4, 2.09, Athlon 2.2 GHz
- Office productivity: P4, 1.67, Athlon 2.2 GHz
- S.F. Chronicle 11/20/00: "...the challenge for AMD now will be to argue that frequency is not the most important thing—precisely the position Intel has argued while its Pentium III lapped behind the Athlon in clock speed."

Summary of Dynamic Scheduling

- Pipeline stages
  - Decoding (in-order)
  - Decode
  - Commit (in-order)
- Two organizations
  - Ways: table + shuffle + reg = table + reg + shuffle + reg
  - Reg and shuffle = reg + shuffle + reg
- Regular buffer: provides in-order commit
- Real 100 processors
  - Very complicated (like a vehicle)
  - Interleaved variants
  - But all root in those basic designs

- GCC 6.02: introduces scoreboard
- Terascale introduces renaming and tag broadcasting
- Register buffer: provides in-order commit
- 100 processors
  - Very complicated (like a vehicle)
  - Interleaved variants
  - But all root in those basic designs