ECE 454
Computer Systems Programming
CPU Architecture

Ding Yuan
ECE Dept., University of Toronto
http://www.eecg.toronto.edu/~yuan

Content

• Examine the tricks CPU plays to make life efficient
  • History of CPU architecture
  • Modern CPU Architecture basics
  • UG machines

• More details are covered in ECE 552
Before we start...

- Hey, isn’t the CPU speed merely driven by transistor density?
  - Transistor density increase $\rightarrow$ clock cycle increase $\rightarrow$ faster CPU

- A faster CPU requires
  - Faster clock cycle
  - smaller *Cycles Per Instruction (CPI)*
    - *CPI is the focus of this lecture!*

In the Beginning...

- 1961:
  - First commercially-available integrated circuits
  - By Fairchild Semiconductor and Texas Instruments

- 1965:
  - Gordon Moore’s observation: (director of Fairchild research)
    - number of transistors on chips was doubling annually
1971: Intel Releases the 4004

- First commercially available, stand-alone microprocessor
- 4 chips: CPU, ROM, RAM, I/O register,
- 108KHz; 2300 transistors
  - 4-bit processor for use in calculators

Designed by Federico Faggin
Intel 4004 (first microprocessor)
- 3 Stack registers (what does this mean)?
- 4-bit processor, but 4KB memory (how)?
- No Virtual Memory support
- No Interrupt
- No pipeline

The 1970’s (Intel): Increased Integration

- 4004: 1971: 108KHz; 2300 trans.; 4-bit processor for use in calculators
- 8008: 1972: 500KHz; 3500 trans.; 20 support chips
- 8080: 1974: 2MHz; 6k trans.; 6 support chips
- 8086: 1978: 10MHz; 29k trans.; Full 16-bit processor, start of x86
The 1980’s: RISC and Pipelining

- 1980: Patterson (Berkeley) coins term RISC
- 1982: Makes RISC-I pipelined processors (only 32 instructions)
- 1981: Hennessy (Stanford) develops MIPS
- 1984: Forms MIPS computers
- RISC Design Simplifies Implementation
  - Small number of instruction formats
  - Simple instruction processing
- RISC Leads Naturally to Pipelined Implementation
  - Partition activities into stages
  - Each stage simple computation
RISC pipeline

Instruction execution in 6-stage pipeline

Instruction 1: IF | ID | EX | MEM | WB
Instruction 2: IF | ID | EX | MEM | WB
Instruction 3: IF | ID | EX | MEM | WB
Instruction 4: IF | ID | EX | MEM | WB
Instruction 5: IF | ID | EX | MEM | WB
Instruction 6: IF | ID | EX | MEM | WB
Instruction 7: IF | ID | EX | MEM | WB

Reduce CPI from 5 → 1 (ideally)

1985: Pipelining: Intel 386

- 33MHz, 32-bit processor, cache → KBs
Pipelines and Branch Prediction

Instruction execution in 6-stage pipeline

- Must wait/stall fetching until branch direction known?
- Solutions?

Branch instructions: 15% - 25%
- Pipeline deeper: branch not resolved until much later
  - Cycles are smaller
  - More functionality btw. fetch & decode
  - Misprediction penalty larger!
- Multiple instruction issue (superscalar)
  - Flushing & refetching more instructions
  - Object-oriented programming
    - More indirect branches which are harder to predict by compiler
Branch Prediction: solution

- Solution: predict branch directions:
  - Intuition: predict the future based on history
  - Use a table to remember outcomes of previous branches

BP is important: 30K bits is the standard size of prediction tables on Intel P4!

1993: Intel Pentium
What do we have so far

- CPI:
  - Pipeline: reduce CPI from $n$ to $1$ (ideal case)
  - Branch instruction will cause stalls: effective CPI > 1
    - Branch prediction
  - But can we reduce CPI to <1?

Instruction-Level Parallelism

- Application
  - Execution Time
- Instructions
- Single-issue
- Superscalar
1995: Intel Pentium Pro

DIV F0, F2, F4 // F0 = F2/F4
ADD F10, F0, F8 // F10 = F0 + F8
SUB F12, F8, F14 // F12 = F8 - F14

Data hazard: obstacle to perfect pipeline

DIV F0, F2, F4 // F0 = F2/F4
ADD F10, F0, F8 // F10 = F0 + F8
SUB F12, F8, F14 // F12 = F8 - F14

STALL: Waiting for F0 to be written
STALL: Waiting for F0 to be written
Necessary?
Out-of-order execution: solving data-hazard

DIV F0, F2, F4 // F0 = F2/F4
ADD F10, F0, F8 // F10 = F0 + F8
SUB F12, F8, F14 // F12 = F8 - F14

DIV F0, F2, F4
ADD F10, F0, F8
SUB F12, F8, F14

STALL: Waiting for F0 to be written
Not wait (as long as it's safe)

Out-of-Order exe. to mask cache miss delay

IN-ORDER:
inst1
inst2
inst3
inst4
load (misses cache)

load (misses cache)
Cache miss latency
inst5 (must wait for load value)
inst6

OUT-OF-ORDER:
inst1
load (misses cache)
inst2
inst3
inst4

Cache miss latency
inst5 (must wait for load value)
inst6
Out-of-order execution

- In practice, much more complicated
  - Detect dependency
  - Introduce additional hazard
    - e.g., what if I write to a register too early?

Instruction-Level Parallelism

<table>
<thead>
<tr>
<th>Application</th>
<th>Instructions</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>single-issue</td>
<td></td>
<td></td>
</tr>
<tr>
<td>superscalar</td>
<td></td>
<td></td>
</tr>
<tr>
<td>out-of-order super-scalar</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Ding Yuan, ECE454
1999: Pentium III

Deep Pipelines

Pentium III’s Pipeline: 10 stages

Pentium IV’s Pipeline (deep pipeline):
The Limits of Instruction-Level Parallelism

---

2000: Pentium IV
Multithreading The “Old Fashioned” Way

Simultaneous Multithreading (SMT) (aka Hyperthreading)

SMT: 20-30% faster than context switching
Putting it all together: Intel

<table>
<thead>
<tr>
<th>Year</th>
<th>Processor</th>
<th>Tech.</th>
<th>CPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>1971</td>
<td>4004</td>
<td>no pipeline</td>
<td>n</td>
</tr>
<tr>
<td>1985</td>
<td>386</td>
<td>pipeline</td>
<td>close to 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>branch prediction</td>
<td>closer to 1</td>
</tr>
<tr>
<td>1993</td>
<td>Pentium</td>
<td>Superscalar</td>
<td>&lt; 1</td>
</tr>
<tr>
<td>1995</td>
<td>PentiumPro</td>
<td>Out-of-Order exe.</td>
<td>&lt;&lt; 1</td>
</tr>
<tr>
<td>1999</td>
<td>Pentium III</td>
<td>Deep pipeline</td>
<td>shorter cycle</td>
</tr>
<tr>
<td>2000</td>
<td>Pentium IV</td>
<td>SMT</td>
<td>&lt;&lt;&lt;1</td>
</tr>
</tbody>
</table>

32-bit to 64-bit Computing

- Why 64 bit?
  - 32b addr space: 4GB; 64b addr space: 18M * 1TB
  - Benefits large databases and media processing
  - OS’s and counters
    - 64bit counter will not overflow (if doing ++)
  - Math and Cryptography
    - Better performance for large/precise value math

- Drawbacks:
  - Pointers now take 64 bits instead of 32
  - Ie., code size increases

☞ unlikely to go to 128bit
Core2 Architecture (2006): UG machines!

Summary (UG Machines CPU Core Arch. Features)

- 64-bit instructions
- Deeply pipelined
  - 14 stages
  - Branches are predicted
- Superscalar
  - Can issue multiple instructions at the same time
  - Can issue instructions out-of-order