Introduction to Microprocessors

Yuri Baida
yuri.baida@gmail.com
yuriy.v.baida@intel.com

October 2, 2010
Agenda

- Background and History
  - What is a microprocessor?
  - What is the history of the development of the microprocessor?
  - How does transistor scaling affect processor design?

- PC Components
  - What are the major PC components and their functions?
  - What is memory hierarchy and how has it changed?

- Processor Architecture
  - What are processor architecture and microarchitecture?
  - How does microarchitecture affect performance?
  - How is performance measured?
Background and History
What is a Microprocessor?

- Microprocessor is a computer **Central Processing Unit (CPU)** on a single chip.
- It contains **millions of transistors** connected by wires.
Electrical Numerical Integrator and Calculator

- Designed for the *U.S. Army’s Ballistic Research Laboratory*
- Built out of
  - 17,468 vacuum tubes
  - 7,200 crystal diodes
  - 1,500 relays
  - 70,000 resistors
  - 10,000 capacitors
- Consumed *150 kW* of power
- Took up *72 m²*
- Weighted *27 tons*
- Suffered a failure on average every 6 hours
Electrical Numerical Integrator and Calculator

Glen Beck and Betty Snyder program the ENIAC in BRL building 328. (Picture: U.S. Army)
The First Transistor was Created in 1947

- Used germanium
- Created by a team lead by William Shockley at Bell Labs
- Shockley later shared the Noble prize in physics
- Shockley semiconductors was founded in Palo Alto in 1955
- In 1957 Bob Noyce, Gordon Moore, and 6 others ("Traitorous Eight") leave to found Fairchild semiconductor
The First Integrated Circuit was Created in 1959

- Proposed independently by Bob Noyce at Fairchild and Jack Kilby at Texas Instruments
- In 1968 Noyce and Moore leave Fairchild to found Intel
- Contained a single transistor and supporting components

The first working integrated circuit (1.6 × 11.1 mm)
Picture: TI
Intel Created the First Commercial Microprocessor

- Introduced the **4004** in 1971, contained **2,300** transistors
- Had roughly **the same processing power as ENIAC**

---

Intel 4004

Picture: Intel

Intel founders (circa 1978)

Picture: Intel
IBM Introduced its Original PC in 1981

- Used the **Intel 8088** processor containing **29,000** transistors
- Used operating system (MS-DOS) designed by Microsoft
Microprocessor Evolution

- 4004 transistors were 10 µm across
- Pentium 4 transistors are 0.13 µm across
- Human hair is about 100 µm across

- Smaller transistors allow
  - More transistors per chip
  - More processing per clock cycle
  - Faster clock rates
  - Smaller/cheaper chips
Microprocessor Evolution

Feature Size (microns)

100

10

1

0.1

0.01

'60 '65 '70 '75 '80 '85 '90 '95 '00 '05 '10

Intel [update 5/20/02]

ITRS [2001 edition]

Examples

Human hair, 100 μm

Amoeba, 15 μm

Red blood cell, 7 μm

AIDS virus, 0.1 μm

Buckyball, 0.001 μm

** Planar Transistor; remaining data points are ICs.
Source: Intel, post ’96 trend data provided by SIA
International Technology Roadmap for Semiconductors (ITRS)
* ITRS DRAM Half Pitch vs. Intel "Lithography"

Picture: Intel
Moore’s Law

- “The number of transistors incorporated in a chip will approximately double every 24 months.” (1965)
Computer Components
PC Components

- **Microprocessor** — performs all computations
- **Cache** — fast memory which holds current data and program
- **Main memory** — larger DRAM memory contains more data
- **Chipset** — controls communication between components
- **Motherboard** — circuit board which holds all the above components
- **Peripheral cards** — controls added computer accessories
Memory Hierarchy

- **Register File**: ~1kB, ~1ns
- **Level 1 Cache**: ~100kB, ~5ns
- **Level 2 Cache**: ~1MB, ~10ns
- **Level 3 Cache**: ~10MB, ~20ns
- **Main Memory**: ~10GB, ~100ns
- **Hard Drive**: ~1TB, ~10ms

**Capacity** vs **Access Time**
Processor-Memory Performance Gap

- **60% per year**
  - (Doubles every 1.5 year)

- **9% per year**
  - (Doubles every 10 year)

**Performance gap**
- (Grows 50% per year)

Source: David Patterson, UC Berkeley
Memory Hierarchy Evolution

386
No on-die cache.
Level 1 cache on motherboard

486
Level 1 cache on-die.
Level 2 cache on motherboard

Pentium
Separate Instruction and Data Caches
Memory Hierarchy Evolution

Pentium II
Separate bus to L2 cache in same package

Pentium III
L2 cache on-die

Core i7
L3 cache on-die
Microprocessor Architecture
What is Architecture?

- Computer architecture is defined by the instructions a processor can execute.
- Programs written for one processor can run on any other processor of the same architecture.
- Current architectures include:
  - IA32 (x86)
  - IA64
  - ARM
  - PowerPC
  - SPARC
What are Instructions?

- Instructions are the most basic actions the processor can take:
  - ADD AX, BX — Add value AX to BX and store in AX
  - CMP AX, 5 — Compare value in AX to 5
  - JE 16 — Jump ahead 16 bytes if comparison was equal

- High level programming languages (C, C++, Java) allow many processor instructions to be written simply:
  - if (A + B = 5) then... — Jump if sum of A and B is 5

- Every program must be converted to the processor instructions of the computer it will be run on
What is Microarchitecture?

- Microarchitecture is the steps a processor takes to execute a particular set of instructions
- Processors of the same architecture have the same instructions but may carry them out in different ways
- Microarchitecture Features:
  - Cache memory
  - Pipelining
  - Out-of-Order Execution
  - Superscalar Issue
**Simplified Microprocessor**

Fetch Unit gets the next instruction from the cache.

Decode Unit determines type of instruction.

Instruction and data sent to Execution Unit.

Write Unit stores result.
### Sequential Processing (386)

<table>
<thead>
<tr>
<th>Cycle</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instr_1</td>
<td>Fetch</td>
<td>Decode</td>
<td>Execute</td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instr_2</td>
<td></td>
<td></td>
<td>Fetch</td>
<td>Decode</td>
<td>Execute</td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instr_3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Fetch</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Sequential processing works on one instruction at a time**
## Pipelined Processing (486)

<table>
<thead>
<tr>
<th>Cycle</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instr(_1)</td>
<td>Fetch</td>
<td>Decode</td>
<td>Execute</td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instr(_2)</td>
<td>Fetch</td>
<td>Decode</td>
<td>Execute</td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instr(_3)</td>
<td>Fetch</td>
<td>Decode</td>
<td>Execute</td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instr(_4)</td>
<td>Fetch</td>
<td>Decode</td>
<td>Execute</td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instr(_5)</td>
<td></td>
<td></td>
<td>Fetch</td>
<td>Decode</td>
<td>Execute</td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instr(_6)</td>
<td></td>
<td></td>
<td>Fetch</td>
<td>Decode</td>
<td>Execute</td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Latency** — elapsed time from start to completion of a particular task
- **Throughput** — how many tasks can be completed per unit of time
- **Pipelining only improves throughput**
  - Each job still takes 4 cycles to complete
- **Real life analogy: Henry Ford’s automobile assembly line**
### In-Order Pipeline (486)

<table>
<thead>
<tr>
<th>Cycle</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instr₁</td>
<td>Fetch</td>
<td>Decode</td>
<td>Execute</td>
<td></td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instr₂</td>
<td>Fetch</td>
<td>Decode</td>
<td></td>
<td>Wait</td>
<td>Execute</td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instr₃</td>
<td>Fetch</td>
<td>Decode</td>
<td></td>
<td>Wait</td>
<td>Execute</td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instr₄</td>
<td>Fetch</td>
<td>Decode</td>
<td></td>
<td>Wait</td>
<td>Execute</td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instr₅</td>
<td>Fetch</td>
<td>Decode</td>
<td></td>
<td>Wait</td>
<td>Execute</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instr₆</td>
<td>Fetch</td>
<td>Decode</td>
<td></td>
<td>Wait</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- In-Order execution requires instructions to be executed in the original program order
Out-of-Order Execution (Pentium II)

<table>
<thead>
<tr>
<th>Cycle</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instr₁</td>
<td>Fetch</td>
<td>Decode</td>
<td>Execute</td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instr₂</td>
<td>Fetch</td>
<td>Decode</td>
<td>Wait</td>
<td>Execute</td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instr₃</td>
<td>Fetch</td>
<td>Decode</td>
<td>Execute</td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instr₄</td>
<td>Fetch</td>
<td>Decode</td>
<td>Wait</td>
<td>Execute</td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instr₅</td>
<td>Fetch</td>
<td>Decode</td>
<td>Execute</td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instr₆</td>
<td>Fetch</td>
<td>Decode</td>
<td>Execute</td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Program Order vs Dataflow Order**
- **Dataflow**: data-driven scheduling of events
  - The start of an event should be enabled by the availability of its required input (data dependency)
- **Real life analogy**: taking tests — work on the questions you know first
### Superscalar Issue (Pentium)

<table>
<thead>
<tr>
<th>Cycle</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instr(_1)</td>
<td>Fetch</td>
<td>Decode</td>
<td>Execute</td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instr(_2)</td>
<td>Fetch</td>
<td>Decode</td>
<td>Wait</td>
<td>Execute</td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instr(_3)</td>
<td>Fetch</td>
<td>Decode</td>
<td>Execute</td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instr(_4)</td>
<td>Fetch</td>
<td>Decode</td>
<td>Wait</td>
<td>Execute</td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instr(_5)</td>
<td>Fetch</td>
<td>Decode</td>
<td>Execute</td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instr(_6)</td>
<td>Fetch</td>
<td>Decode</td>
<td>Execute</td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instr(_7)</td>
<td>Fetch</td>
<td>Decode</td>
<td>Execute</td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instr(_8)</td>
<td>Fetch</td>
<td>Decode</td>
<td>Execute</td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Superscalar issue allows multiple instructions to be issued at the same time
Microarchitecture and Performance

- Performance is measured by how long a processor takes to run a program
- Time is reduced by increasing Instructions Per Cycle (IPC) and clock rate
- Microarchitecture affects IPC and clock rate:
  - More pipe stages
    - Less work in each cycle means better clock rate
    - More dependencies means worse IPC
  - Superscalar Issue and Out-of-Order Execution
    - Parallel work means better IPC
    - More complexity can mean worse clock rate
Measuring Processor Performance

- **Clock Rate**
  - Simplest but not especially accurate

- **Instructions Per Cycle (IPC)**
  - Not meaningful without clock rate
  - Varies from program to program

- **SPEC Performance**
  - Standard Performance Evaluation Corporation
  - Tests PCs using benchmark suite of programs
    - SPECint: Integer intensive programs
    - SPECfp: Floating point intensive programs

- **TPC Performance**
  - Transaction Performance Council
  - Tests servers and workstations
Acknowledgements

- These slides contain material developed and copyright by:
  - Grant McFarland (Intel)
  - Mark Louis (Intel)
  - Arvind (MIT)
  - Joel Emer (Intel/MIT)