Computer Architecture

The Instruction Set Architecture (ISA)
The ISA is an example of a successful parallel abstraction
The Instruction Set Architecture (ISA)
   The ISA is an example of a successful parallel abstraction

Today we’ll look at microprocessor trends over the years

What can we learn from this success story?
Parallelism in Hardware

Microprocessors are highly parallel.
Consider a block diagram of the Pentium processor.

Figure 1. Pentium block diagram.
Hardware Has Long Exploited Parallelism

Bit-serial ALUs
Hardware Has Long Exploited Parallelism

Bit-serial ALUs
Bit-parallel ALUs
Hardware Has Long Exploited Parallelism

Bit-serial ALUs
Bit-parallel ALUs
Pipelined microprocessors
Pipelined Microprocessors

Divide an instruction into stages

<table>
<thead>
<tr>
<th>Stage 1</th>
<th>Stage 2</th>
<th>Stage 3</th>
<th>Stage 4</th>
<th>Stage 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction Fetch</td>
<td>Instruction Decode &amp; Register Fetch</td>
<td>Execute</td>
<td>Memory Access</td>
<td>Register Write-back</td>
</tr>
</tbody>
</table>

IF \[\rightarrow\] ID/RF \[\rightarrow\] EX \[\rightarrow\] MEM \[\rightarrow\] WB

![Diagram showing pipeline stages with time and instructions](image-url)
Pipelined Microprocessors

Divide an instruction into stages

<table>
<thead>
<tr>
<th>Stage 1</th>
<th>Stage 2</th>
<th>Stage 3</th>
<th>Stage 4</th>
<th>Stage 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction</td>
<td>Instruction</td>
<td>Execute</td>
<td>Memory Access</td>
<td>Register</td>
</tr>
<tr>
<td>Fetch</td>
<td>Decode &amp; Register Fetch</td>
<td></td>
<td>Access</td>
<td>Write-back</td>
</tr>
</tbody>
</table>

IF ➞ ID/RF ➞ EX ➞ MEM ➞ WB

![Diagram showing stages of instruction processing](image)

Hardware Success Stories
Divide an instruction into stages

<table>
<thead>
<tr>
<th>Stage 1</th>
<th>Stage 2</th>
<th>Stage 3</th>
<th>Stage 4</th>
<th>Stage 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction Fetch</td>
<td>Instruction Decode &amp;</td>
<td>Execute</td>
<td>Memory Access</td>
<td>Register Write-back</td>
</tr>
<tr>
<td></td>
<td>Register Fetch</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF</td>
<td>ID/RF</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
</tbody>
</table>

A form of task parallelism

Increases **latency** of a single instruction
## Pipelined Microprocessors

Divide an instruction into stages

<table>
<thead>
<tr>
<th>Stage 1</th>
<th>Stage 2</th>
<th>Stage 3</th>
<th>Stage 4</th>
<th>Stage 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction Fetch</td>
<td>Instruction Decode &amp; Register Fetch</td>
<td>Execute</td>
<td>Memory Access</td>
<td>Register Write-back</td>
</tr>
<tr>
<td>IF</td>
<td>ID/RF</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
</tbody>
</table>

A form of task parallelism

Increases **latency** of a single instruction

Increases **throughput**—ideally completes one instruction per cycle

Hardware Success Stories
Hardware Has Long Exploited Parallelism

Bit-serial ALUs
Bit-parallel ALUs
Pipelined microprocessors
Out-of-order execution

Program order
Hardware Has Long Exploited Parallelism

Bit-serial ALUs
Bit-parallel ALUs
Pipelined microprocessors
Out-of-order execution

Program order

Issue order

Hardware Success Stories
Hardware Has Long Exploited Parallelism

Bit-serial ALUs
Bit-parallel ALUs
Pipelined microprocessors
Out-of-order execution

How does out-of-order execution help?
Hardware Has Long Exploited Parallelism

Bit-serial ALUs
Bit-parallel ALUs
Pipelined microprocessors
Out-of-order execution

How does out-of-order execution help?
Initiate a slow instruction early
### Hardware Has Long Exploited Parallelism

- Bit-serial ALUs
- Bit-parallel ALUs
- Pipelined microprocessors
- Out-of-order execution

#### Superscalar execution

<table>
<thead>
<tr>
<th>time</th>
<th>IF</th>
<th>ID</th>
<th>EX</th>
<th>MM</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MM</td>
<td>WB</td>
</tr>
<tr>
<td></td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MM</td>
<td>WB</td>
</tr>
<tr>
<td></td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MM</td>
<td>WB</td>
</tr>
<tr>
<td></td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MM</td>
<td>WB</td>
</tr>
</tbody>
</table>

Issue multiple instructions per cycle
Hardware Has Long Exploited Parallelism

Bit-serial ALUs
Bit-parallel ALUs
Pipelined microprocessors
Out-of-order execution

Superscalar execution

Issue multiple instructions per cycle

Hardware Success Stories
Hardware Has Long Exploited Parallelism

Bit-serial ALUs
Bit-parallel ALUs
Pipelined microprocessors
Out-of-order execution

Superscalar execution

Issue multiple instructions per cycle
Hardware Has Long Exploited Parallelism

Bit-serial ALUs
Bit-parallel ALUs
Pipelined microprocessors
Out-of-order execution

Superscalar execution

Issue multiple instructions per cycle

Hardware Success Stories
Hardware Has Long Exploited Parallelism

Bit-serial ALUs
Bit-parallel ALUs
Pipelined microprocessors
Out-of-order execution

Superscalar execution

Issue multiple instructions per cycle
Hardware Has Long Exploited Parallelism

Bit-serial ALUs
Bit-parallel ALUs
Pipelined microprocessors
Out-of-order execution

Superscalar execution

Issue multiple instructions per cycle

Can’t always exploit full issue width
Hardware Has Long Exploited Parallelism

Bit-serial ALUs
Bit-parallel ALUs
Pipelined microprocessors
Out-of-order execution

Superscalar execution

Issue multiple instructions per cycle

Can’t always exploit full issue width
Hardware Has Long Exploited Parallelism

Bit-serial ALUs
Bit-parallel ALUs
Pipelined microprocessors
Out-of-order execution
Superscalar execution
Multithreading

Issue multiple instructions per cycle

Can’t always exploit full issue width

Hardware Success Stories
Multithreading

Bubbles in the pipeline represent performance loss
Fill bubbles with instructions from other threads

Issue slots
Thread A

Time
bubble

Hardware Success Stories
Multithreading

Bubbles in the pipeline represent performance loss
Fill bubbles with instructions from other threads
Multithreading

Bubbles in the pipeline represent performance loss

Fill bubbles with instructions from other threads
Multithreading

Bubbles in the pipeline represent performance loss
Fill bubbles with instructions from other threads

What is the cost of multithreading?
Multithreading

Bubbles in the pipeline represent performance loss

Fill bubbles with instructions from other threads

What is the cost of multithreading?
Is it ever not profitable?
Multithreading

Bubbles in the pipeline represent performance loss

Fill bubbles with instructions from other threads

What is the cost of multithreading?

Is it ever not profitable?

Problem
– Still have lots of unfilled issue slots

Hardware Success Stories
Improving Hardware Utilization

Limitation of Multithreading
At each cycle, issue instructions from any one thread
Limitation of Multithreading
At each cycle, issue instructions from any one thread

Simultaneous Multithreading (SMT)
[Tulsen, Eggers, Levy, ISCA95]
At each cycle, issue instructions from multiple threads

Improving Hardware Utilization

Hardware Success Stories
Limitation of Multithreading
At each cycle, issue instructions from any one thread

Simultaneous Multithreading (SMT)
[Tulsen, Eggers, Levy, ISCA95]
At each cycle, issue instructions from multiple threads

What is the cost of SMT?
Hardware Has Long Exploited Parallelism

Bit-serial ALUs
Bit-parallel ALUs
Pipelined microprocessors
Out-of-order execution
Superscalar execution
Multithreading
SMT
VLIW
VLIW

Very Long Instruction Word
- Wide instructions for controlling multiple functional units
- ISA explicitly encodes parallelism
- Statically scheduled
- Reduces power by devoting less hardware to control

Transmeta TM8000 Core
Vector Processors

Goal:

Reduce instruction fetch bandwidth

Apply instructions to all elements of vectors simultaneously

eg. **VectorAdd**
Language:en
Usage:educational
Topic:Computer Architecture
Subtopic:Vector Processors

**Goal:**

- Reduce instruction fetch bandwidth
- Apply instructions to all elements of vectors simultaneously
  
  eg. **VectorAdd**

Alleviates the **von Neumann Bottleneck:**

Instruction fetch and data fetch contend for memory
Goal:

- Reduce instruction fetch bandwidth
- Apply instructions to all elements of vectors simultaneously
  eg. VectorAdd

Alleviates the von Neumann Bottleneck:

- Instruction fetch and data fetch contend for memory

Have compilers convert existing programs to vector programs
Vectorizing compilers have been quite successful
Vector Processors: A Success Story?

Vectorizing compilers have been quite successful
But initially, many programs could not be vectorized
Vector Processors: A Success Story?

Vectorizing compilers have been quite successful
But initially, many programs could not be vectorized

Did compilers get better?
Vector Processors: A Success Story?

Vectorizing compilers have been quite successful
But initially, many programs could not be vectorized

Did compilers get better?
No. Through compiler feedback and training, programmers
learned how to write code that could be vectorized
Vector Processors: A Success Story?

Vectorizing compilers have been quite successful
But initially, many programs could not be vectorized

Did compilers get better?
No. Through compiler feedback and training, programmers learned how to write code that could be vectorized

Can we follow a similar path for parallelizing compilers?
Vector Processors: A Success Story?

Vectorizing compilers have been quite successful
But initially, many programs could not be vectorized

Did compilers get better?
No. Through compiler feedback and training, programmers learned how to write code that could be vectorized

Can we follow a similar path for parallelizing compilers?
Parallelization is more difficult.
The question is not just “is a loop parallelizable?” There are many more tradeoffs to consider.
The Big Picture

- Bit-serial ALUs
- Bit-parallel ALUs
- Pipelined microprocessors
- Out-of-order execution
- Superscalar execution
- Multithreading
- SMT
- VLIW
- Vector processors
- Multi-cores

Increasing granularity of parallelism
The Big Picture

Bit-serial ALUs
Bit-parallel ALUs
Pipelined microprocessors
Out-of-order execution
Superscalar execution
Multithreading
SMT
VLIW
Vector processors
Multi-cores

Increasing granularity of parallelism
Which of these expose parallelism to through the ISA?
The Big Picture

Bit-serial ALUs
Bit-parallel ALUs
Pipelined microprocessors
Out-of-order execution
Superscalar execution
Multithreading
SMT
VLIW
Vector processors
Multi-cores

Increasing granularity of parallelism
Which of these expose parallelism to through the ISA?
The Big Picture

Bit-serial ALUs
Bit-parallel ALUs
Pipelined microprocessors
Out-of-order execution
Superscalar execution
Multithreading
SMT
VLIW
Vector processors
Multi-cores

Increasing granularity of parallelism
Which of these expose parallelism to through the ISA?
Conclusions

For years, the ISA was able to hide hardware parallelism, so programmers could just write sequential code.

As the granularity of parallelism has increased, it’s become difficult to hide the parallelism.

Why is the granularity increasing?
Conclusions

For years, the ISA was able to hide hardware parallelism, so programmers could just write sequential code.

As the granularity of parallelism has increased, it’s become difficult to hide the parallelism.

Why is the granularity increasing?

Is hidden complexity always good?