What makes processors fail and how to prevent it

Alastair Reid

alastair.reid@arm.com

@alastair_d_reid
Processors always work

More technical version: https://alastairreid.github.io/papers/CAV_16/
Related talk: Mate Soos “Hacking using SAT and SMT solvers”
2 Related workshop: Jonny Austin “Be a Computer! Demystifying assembly language and CPUs”
Processors almost always work

More technical version: https://alastairreid.github.io/papers/CAV_16/

Related talk: Mate Soos “Hacking using SAT and SMT solvers”

Related workshop: Jonny Austin “Be a Computer! Demystifying assembly language and CPUs”
Correct  Fast

More technical version: https://alastairreid.github.io/papers/CAV_16/
Related talk: Mate Soon “Hacking using SAT and SMT solvers”
Related workshop: Jonny Austin “Be a Computer! Demystifying assembly language and CPUs”
Correct  
\[\rightarrow\]  
Designer  
\[\rightarrow\]  
Fast

More technical version: https://alastairreid.github.io/papers/CAV_16/
Related talk: Mate Soon “Hacking using SAT and SMT solvers”
Related workshop: Jonny Austin “Be a Computer! Demystifying assembly language and CPUs”
Correct \[\xrightarrow{\text{Designer}}\] Fast

More technical version: https://alastairreid.github.io/papers/CAV_16/
Related talk: Mate Soon “Hacking using SAT and SMT solvers”
Related workshop: Jonny Austin “Be a Computer! Demystifying assembly language and CPUs”
Correct → Designer → Fast
Verifier
Using SAT solvers
And Bounded Model Checkers

More technical version: https://alastairreid.github.io/papers/CAV_16/
Related talk: Mate Soon “Hacking using SAT and SMT solvers”
Related workshop: Jonny Austin “Be a Computer! Demystifying assembly language and CPUs”

Similar CPUs, see OPC project (https://revaldinho.github.io/opc/)
What tests to run?

Every instruction $\times$ Every corner case $\times$ Fudge Factor

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Big</th>
<th>Small</th>
<th>Equal</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDR</td>
<td>$\times$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BNE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fudge Factor:

- 1?
- 50?
- 1000?
Logical Equivalence Check (LEC) / SAT
Correct but slow

Fast but broken

Make it faster

Make it work
https://en.wikipedia.org/wiki/Instruction_pipelining
https://en.wikipedia.org/wiki/Instruction_pipelining
BR    PC+100
ADD   R0, R0, #1
BR  PC+100
ADD  R0, R0, #1
BR  PC+100
ADD  R0, R0, #1

Fetch BR
Fetch ADD
Execute BR
Execute ADD
What tests to run?

All the single-stage tests

Branches

All “interesting” pairs of instructions
Correct but slow

Fast but broken

Make it faster

Make it work
<table>
<thead>
<tr>
<th>Fetch 1</th>
<th>Decode 1</th>
<th>Exec 1</th>
<th>Mem 1</th>
<th>Write 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch 2</td>
<td>Decode 2</td>
<td>Exec 2</td>
<td>Mem 1</td>
<td>Write 1</td>
</tr>
<tr>
<td>Fetch 3</td>
<td>Decode 3</td>
<td>Exec 3</td>
<td>Mem 2</td>
<td>Write 2</td>
</tr>
<tr>
<td>Fetch 4</td>
<td>Decode 4</td>
<td>Exec 4</td>
<td>Mem 3</td>
<td>Write 3</td>
</tr>
<tr>
<td>Fetch 5</td>
<td>Decode 5</td>
<td>Exec 5</td>
<td>Mem 4</td>
<td>Write 4</td>
</tr>
<tr>
<td>Fetch 6</td>
<td>Decode 6</td>
<td>Exec 6</td>
<td>Mem 5</td>
<td>Write 5</td>
</tr>
<tr>
<td>Fetch 7</td>
<td>Decode 7</td>
<td>Exec 7</td>
<td>Mem 6</td>
<td>Write 6</td>
</tr>
<tr>
<td>Fetch 8</td>
<td>Decode 8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fetch 9</td>
<td>Decode 9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fetch 10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Bounded Model Checking

Tools: Yosys [http://www.clifford.at/yosys/](http://www.clifford.at/yosys/)
Blogs: [https://zipcpu.com/blog/2017/10/19/formal-intro.html](https://zipcpu.com/blog/2017/10/19/formal-intro.html)

Bounded Model Checking


Tools: Yosys http://www.clifford.at/yosys/
Blogs: https://zipcpu.com/blog/2017/10/19/formal-intro.html
Bounded Model Checking


Tools: Yosys http://www.clifford.at/yosys/
Blogs: https://zipcpu.com/blog/2017/10/19/formal-intro.html

alastair.reid@arm.com @alastair_d_reid
LDR R0, [R1]
STR R0, [R2]
LDR  R0, [R1]
STR  R0, [R2]

LDR  R0, [R1]
STR  R0, [R2]

The diagram illustrates a pipelined MIPS architecture. Each stage in the pipeline is labeled as follows:

- **Instruction Fetch (IF)**
- **Instruction Decode Register Fetch (ID)**
- **Execute Address Calculation (EX)**
- **Memory Access (MEM)**
- **Write Back (WB)**

The pipeline stages are connected with data flow arrows, indicating the sequence of operations. The diagram includes components such as the PC (Program Counter), Memory, Register File, ALU (Arithmetic Logic Unit), and other relevant circuits for instruction execution and data processing.

What tests to run?

All the single-stage tests

All the two-stage tests

Load followed by <anything>

All “interesting” sequences of 5 instructions
Correct but slow → Make it faster → Fast but broken → Make it work
Formal verification is breadth first
Formal verification is breadth first
Formal verification is breadth first
Formal verification is breadth first
Formal verification is breadth first
Formal verification is breadth first
Testing is depth-first
Testing is depth-first
Testing is depth-first
Testing is depth-first
Testing is depth-first
Testing is depth-first
Testing is depth-first
Testing is depth-first
Mixed Mode Verification
Mixed Mode Verification

Idle

BNE

STR

ADD

CMP

LDR
Mixed Mode Verification

Idle

BNE

CMP

ADD

STR

LDR
Mixed Mode Verification

Idle

BNE

CMP

ADD

LDR

STR
Mixed Mode Verification
Mixed Mode Verification

- Idle
- ADD
- CMP
- BNE
- LDR
- STR
Mixed Mode Verification

Idle

ADD

CMP

BNE

LDR

STR
Mixed Mode Verification

- Idle
- ADD
- CMP
- BNE
- LDR
- STR
Processors almost always work
Correct $\rightarrow$ Designer $\rightarrow$ Fast $\leftarrow$ Verifier

Using SAT solvers
And Bounded Model Checkers
Correct  Designer  Fast

Verifier

Using SAT solvers
And Bounded Model Checkers