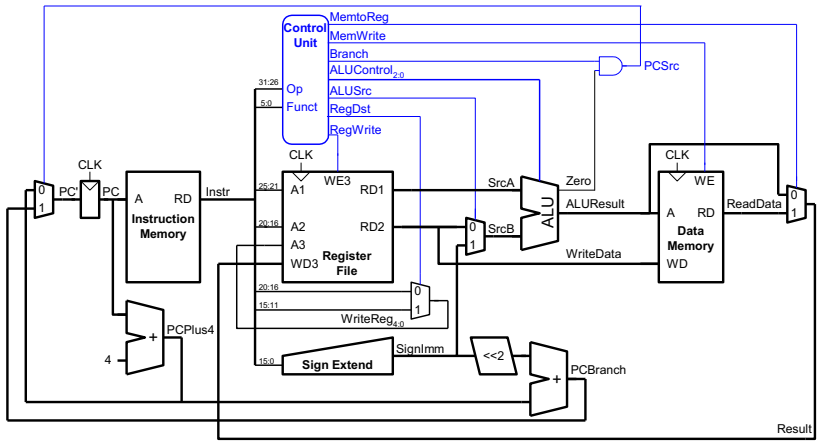
ISA, μArchitecture, and Bad News from the Real World

Jan Reineke
Universität des Saarlandes

Roadmap: Computer architecture



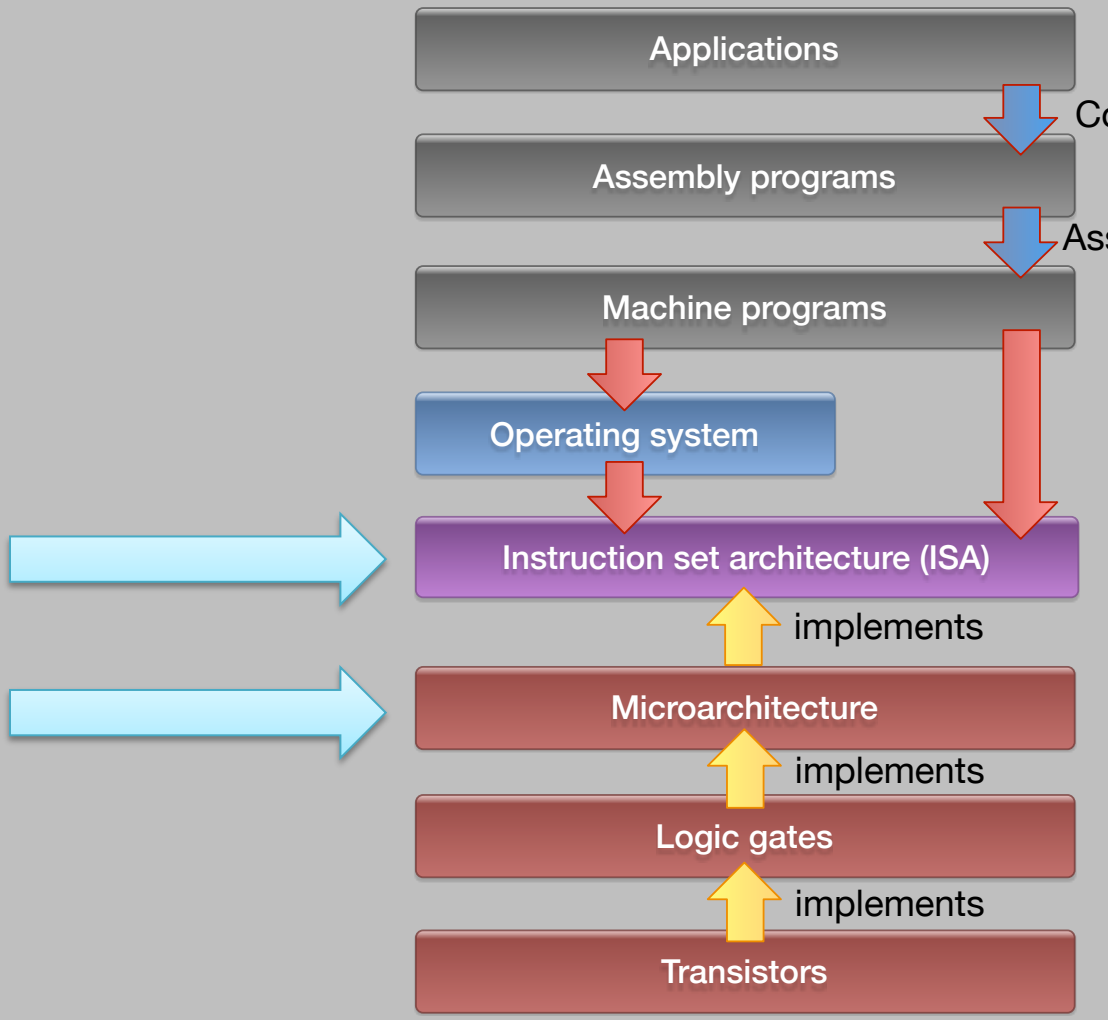
1. Combinatorial circuits: Boolean Algebra/Functions/Expressions/Synthesis
2. Number representations
3. Arithmetic Circuits: Addition, Multiplication, Division, ALU

4. Sequential circuits: Flip-Flops, Registers, SRAM, Moore and Mealy automata
5. Verilog

6. **Instruction Set Architecture**
7. **Microarchitecture**

8. Performance: RISC vs. CISC, Pipelining, Memory Hierarchy

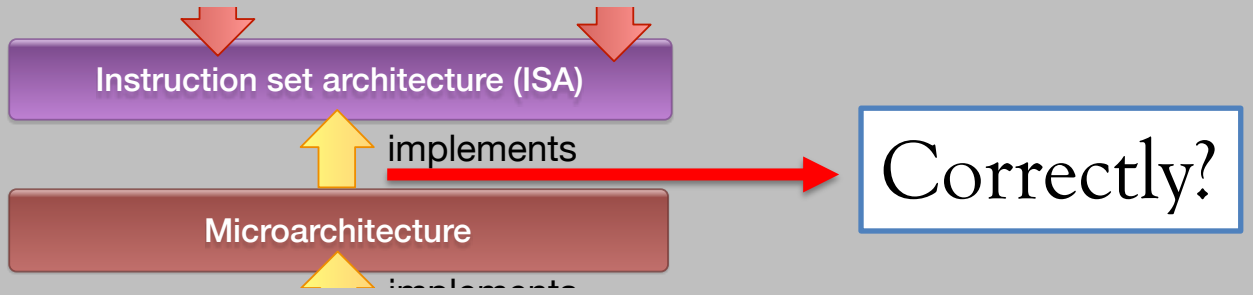
Abstraction layers in computer systems



"What" a computer computes
For example: x86, ARM

"How" a computer works
For example: Intel Skylake, AMD Zen 3 (both x86), Apple M1 (ARM)

What does it mean to "correctly implement" an ISA?



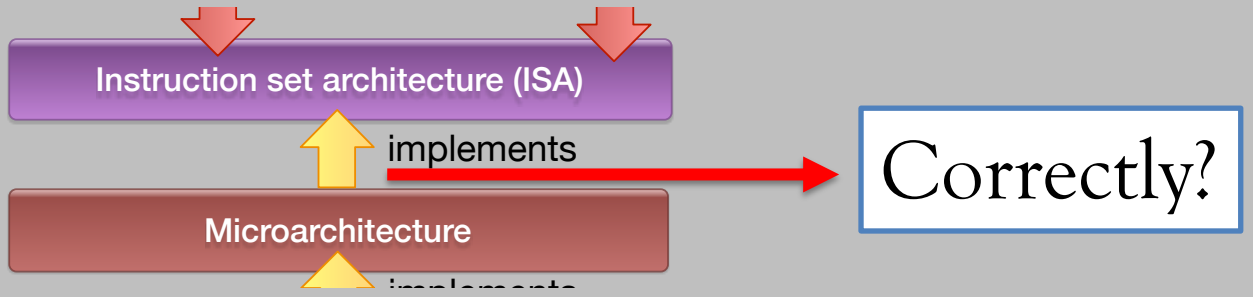
How to formalize ISA?

STATES Q (MIPS = SET OF REGISTERS, PC, AND MEMORY STATE)

NEXT : Q → Q

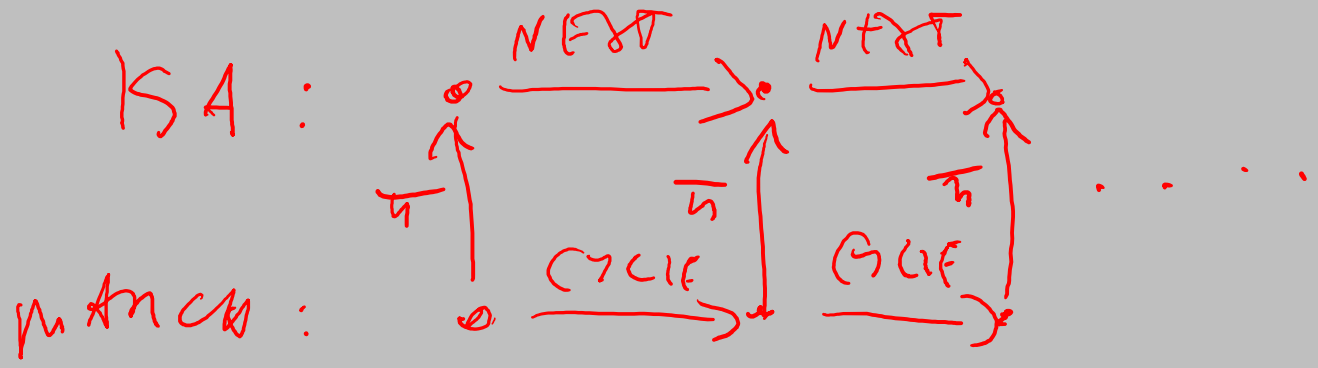
AT THE GRANULARITY OF INSTRUCTIONS

What does it mean to "correctly implement" an ISA?

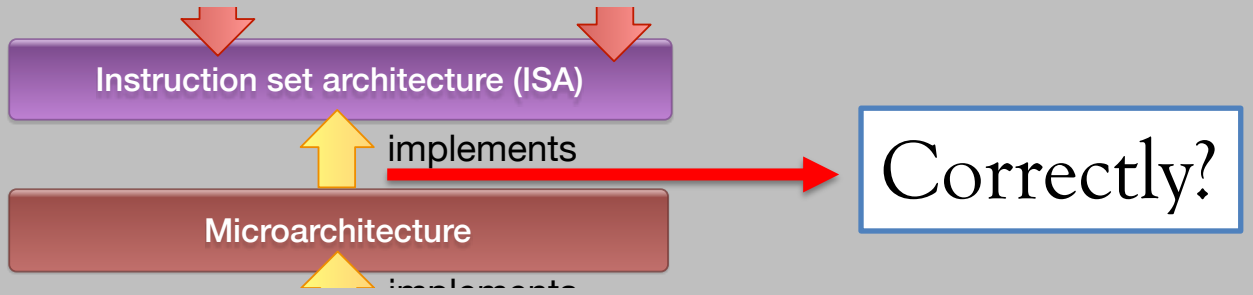


What does "correct implementation" mean?

For single-cycle processors?



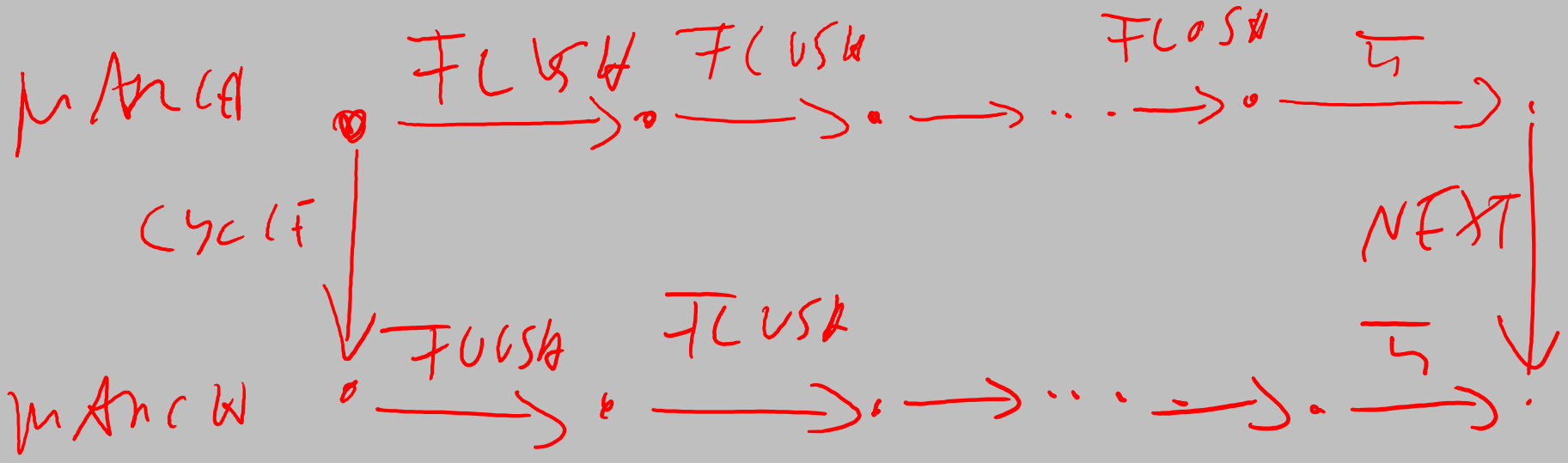
What does it mean to "correctly implement" an ISA?



What does "correct implementation" mean?

For a pipelined processors?

Bunch / Dill, CAV 94

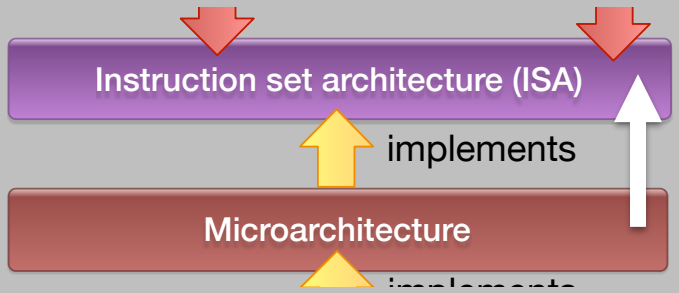


Back to formalization...

How to formalize ISA?

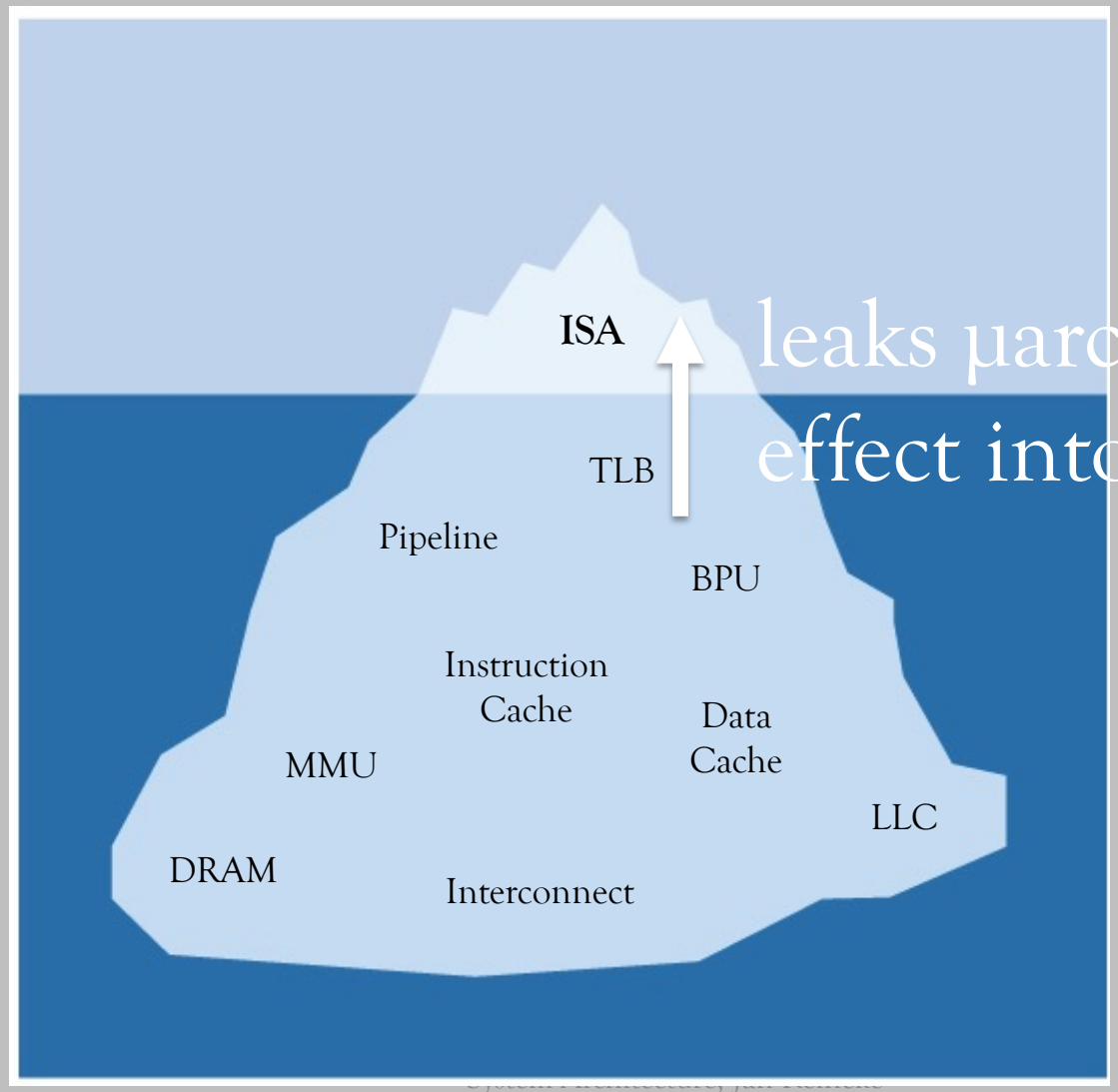
A weird instruction:
`rdtsc`:"read time-stamp counter"

"count number of CPU cycles since its reset"



leaks μarchitectural effect into ISA state!

Leaky abstractions

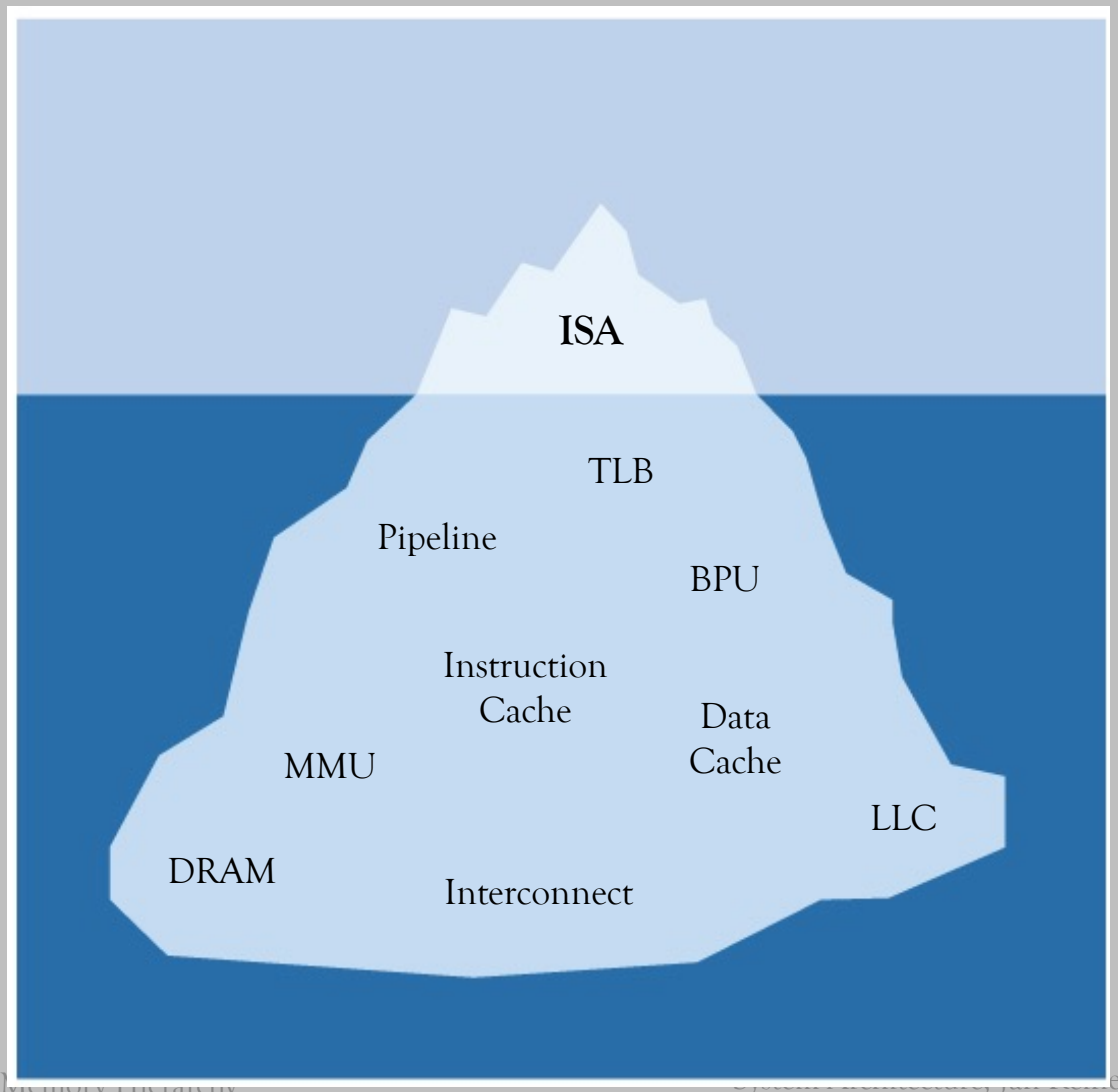


leaks μarchitectural effect into ISA state!

Based on slides kindly provided by Yuval Yarom, The University of Adelaide

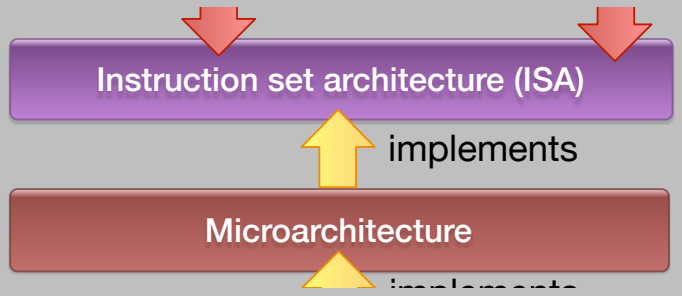
Excursion: Microarchitectural Attacks

Leaky abstractions – How to fix it?



Can we capture the leakage at the ISA level?

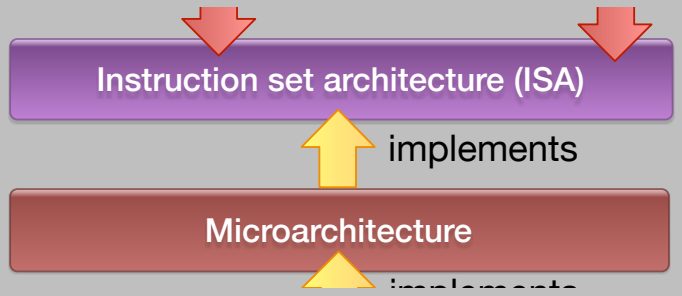
How to capture "microarchitectural leakage" at ISA level?



Idea:

- Associate **observations** with instruction executions
- Two program executions are **indistinguishable** if they generate the same sequence of observations

Example: Capturing Memory Hierarchy



Captures leakage via instruction cache

Introduce two types of observations:

- Addresses of executed instructions
- Addresses accessed by memory instructions

Captures leakage via data cache

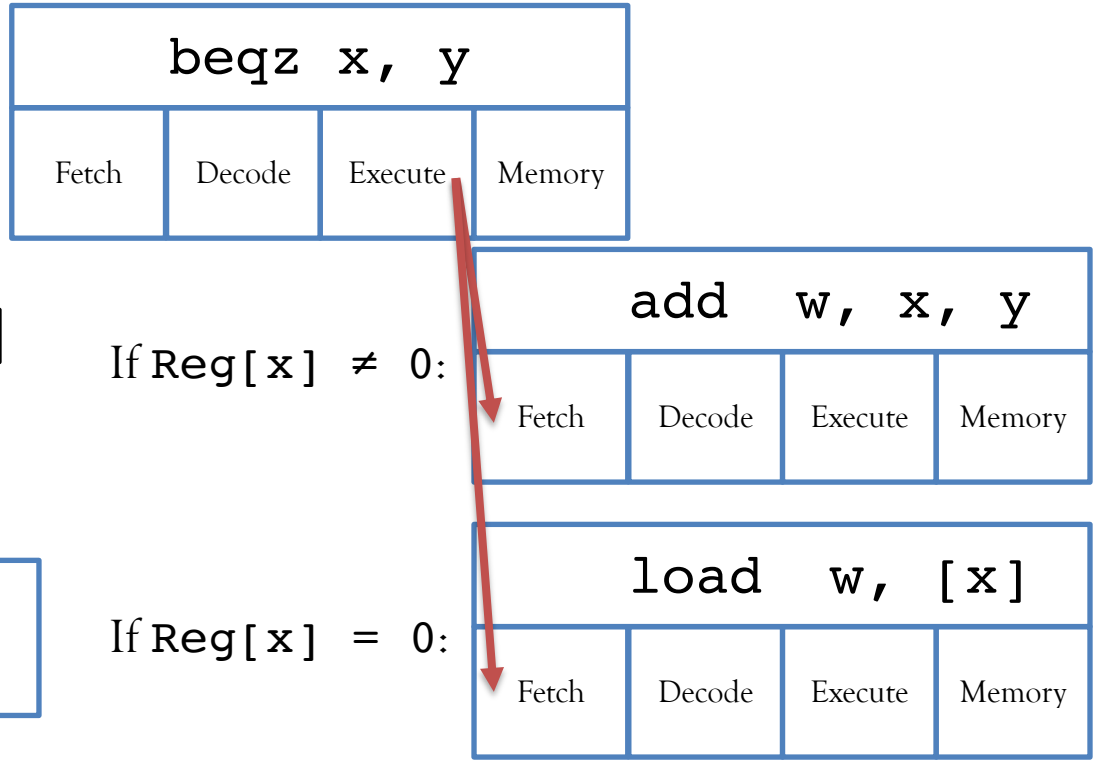
Revisiting the GnuPG example

```
x ⟵1
for i ⟵|d|-1 downto 0 do
    x ⟵x^2 mod n
    if (d_i = 1) then
        x = xC mod n
    endif
done
return x
```

Operation	x	i	d_i
	1	2	101
Square	1	2	101
reduce	1	2	101
Multiply	11	2	101
reduce	11	2	101
Square	121	1	101
reduce	21	1	101
Square	441	0	101
reduce	41	0	101
Multiply	451	0	101
reduce	51	0	101

Pipelining and dependencies: **Control dependencies**

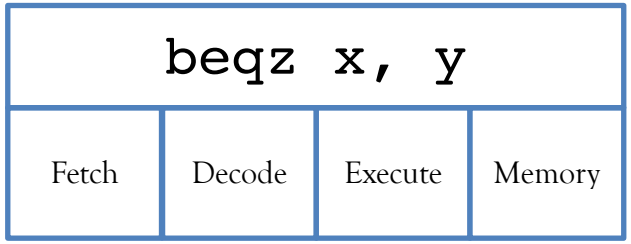

```
beqz x, y
add  w, x,
...
load w, [x]
```



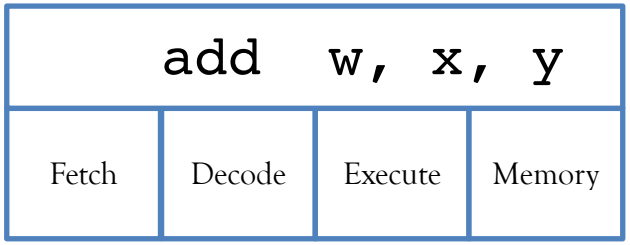
What to do?
→ **Speculative Execution**

Pipelining and dependencies: **Speculative execution**


```
beqz x, y
add  w, x,
...
load w, [x]
```

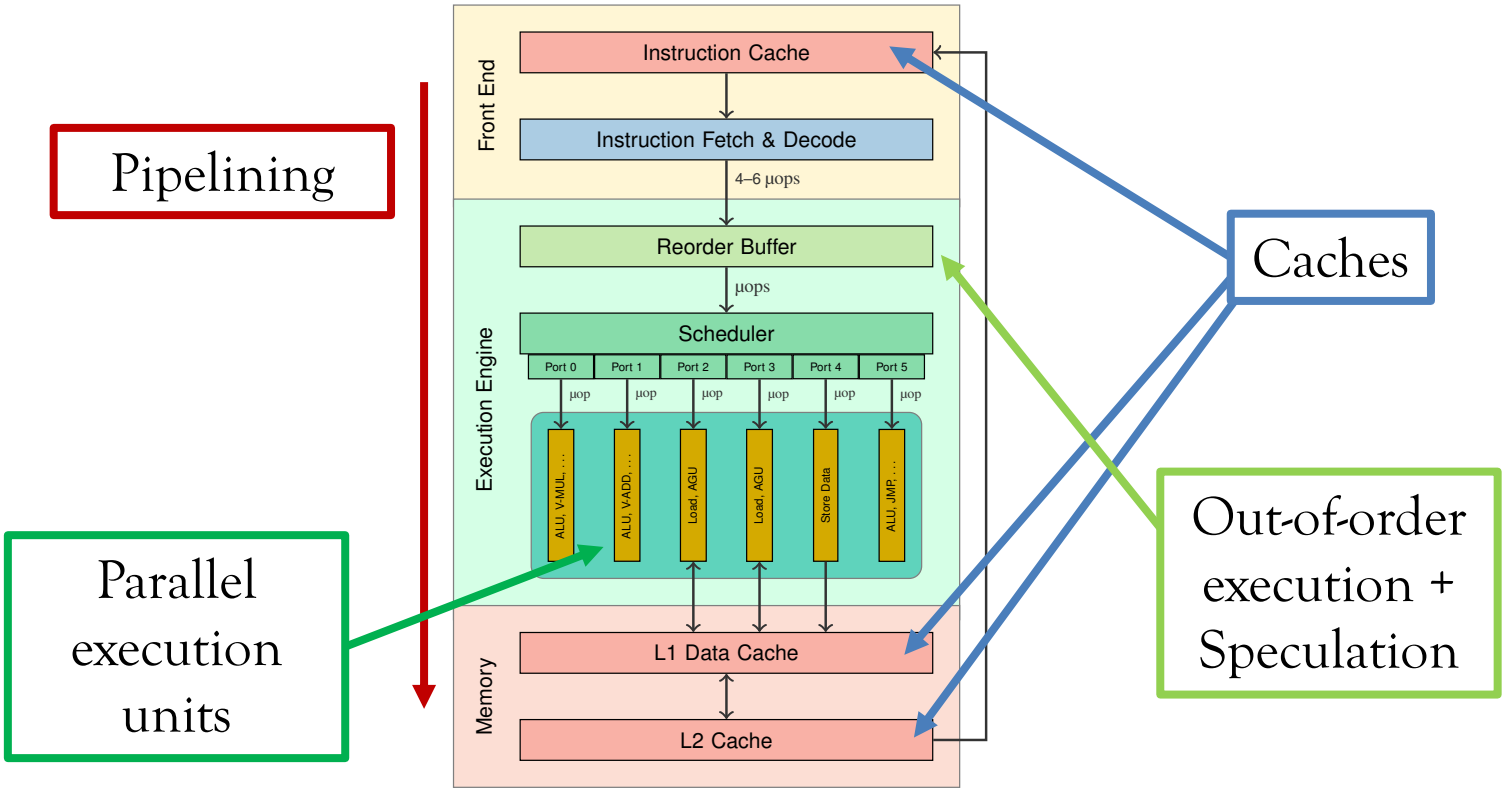


If `Reg[x] ≠ 0`:



1. Guess branch target (via branch predictor)
2. Execute the following instructions speculatively
3. Throw away intermediate results in case of mis-speculation

High-level structure of modern microarchitectures

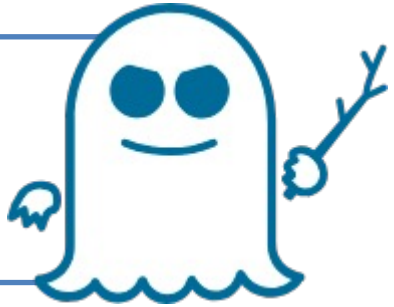


2. Spectre attack

Goal: Access to data of other processes or the operating system, (which should not be accessible)

Approach:

1. Perform illegal memory access **speculatively**
2. **Extract** data via „covert channel“



Spectre Attacks: Exploiting Speculative Execution
Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, Yuval Yarom

Variants of Spectre attacks

1. Meltdown: User process attacks OS 

2. User process attacks other user process

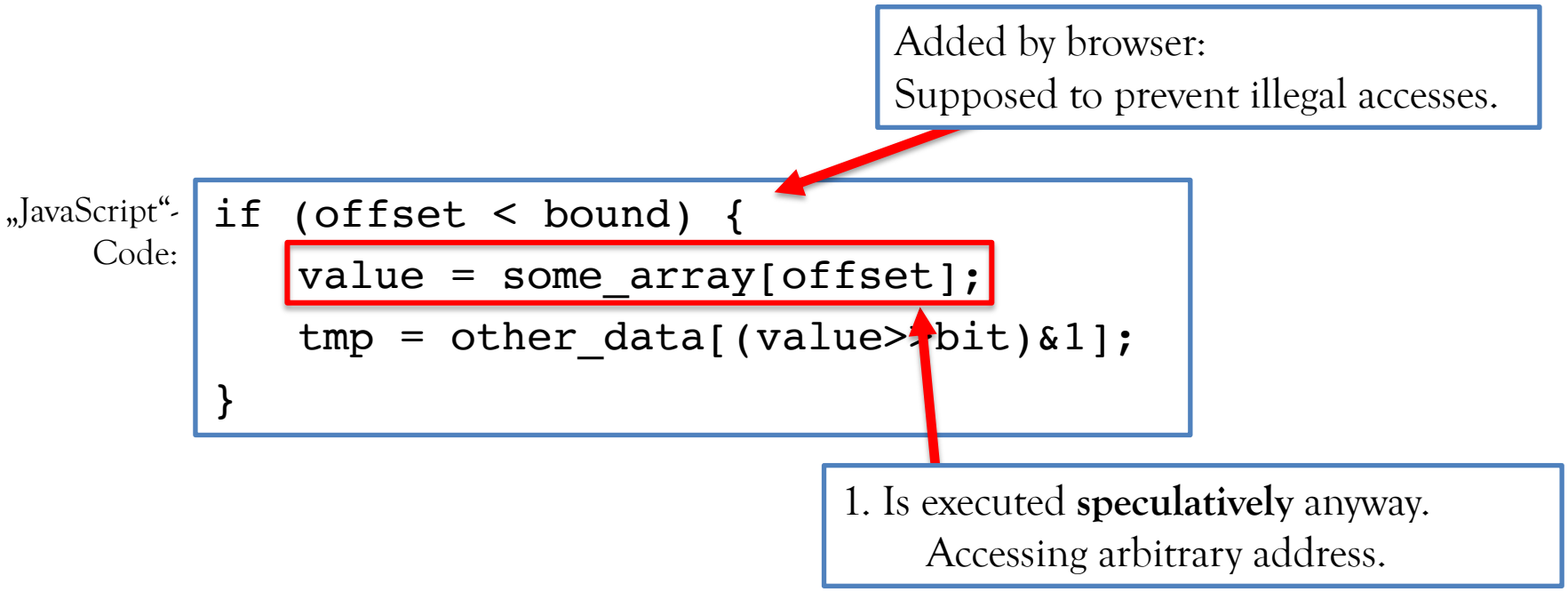
   Requires “Gadget code“ in attacked process.

3. JavaScript attacks browser

   Overcomes "browser sandboxing" protection mechanisms.

4. ...

Spectre:
1. Speculative access



Spectre:
2. Secret-dependent memory access

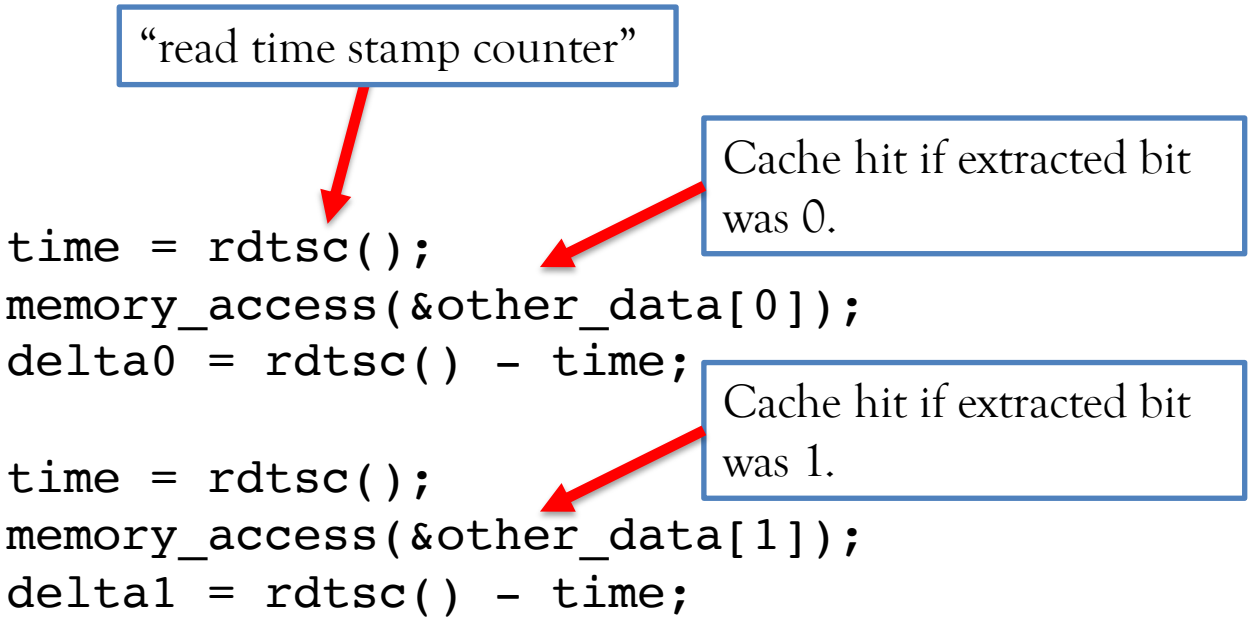
„JavaScript“-Code:

```
if (offset < bound) {
    value = some_array[offset];
    tmp = other_data[(value>>bit)&1];
}
```

Extracts a bit of “value”

2. Secret-dependent memory access

Spectre:
3. Reading out the transmitted data



Challenges

- How to capture speculative execution effects at ISA-level?
- How to prove ISA-level specification is correctly implemented?
- How to test an ISA-level specification?

Further reading:

Marco Guarnieri, Boris Köpf, Jan Reineke, Pepe Vila:
Hardware-Software Contracts for Secure Speculation
IEEE Symposium on Security and Privacy, 2021 (best paper)
Preprint: https://arxiv.org/abs/2006.03841

Zilong Wang, Gideon Mohr, Klaus von Gleissenthall,
Jan Reineke, Marco Guarnieri:
Specification and Verification of Side-channel Security for Open-source Processors via Leakage Contracts
Under submission
Preprint: https://arxiv.org/abs/2305.06979