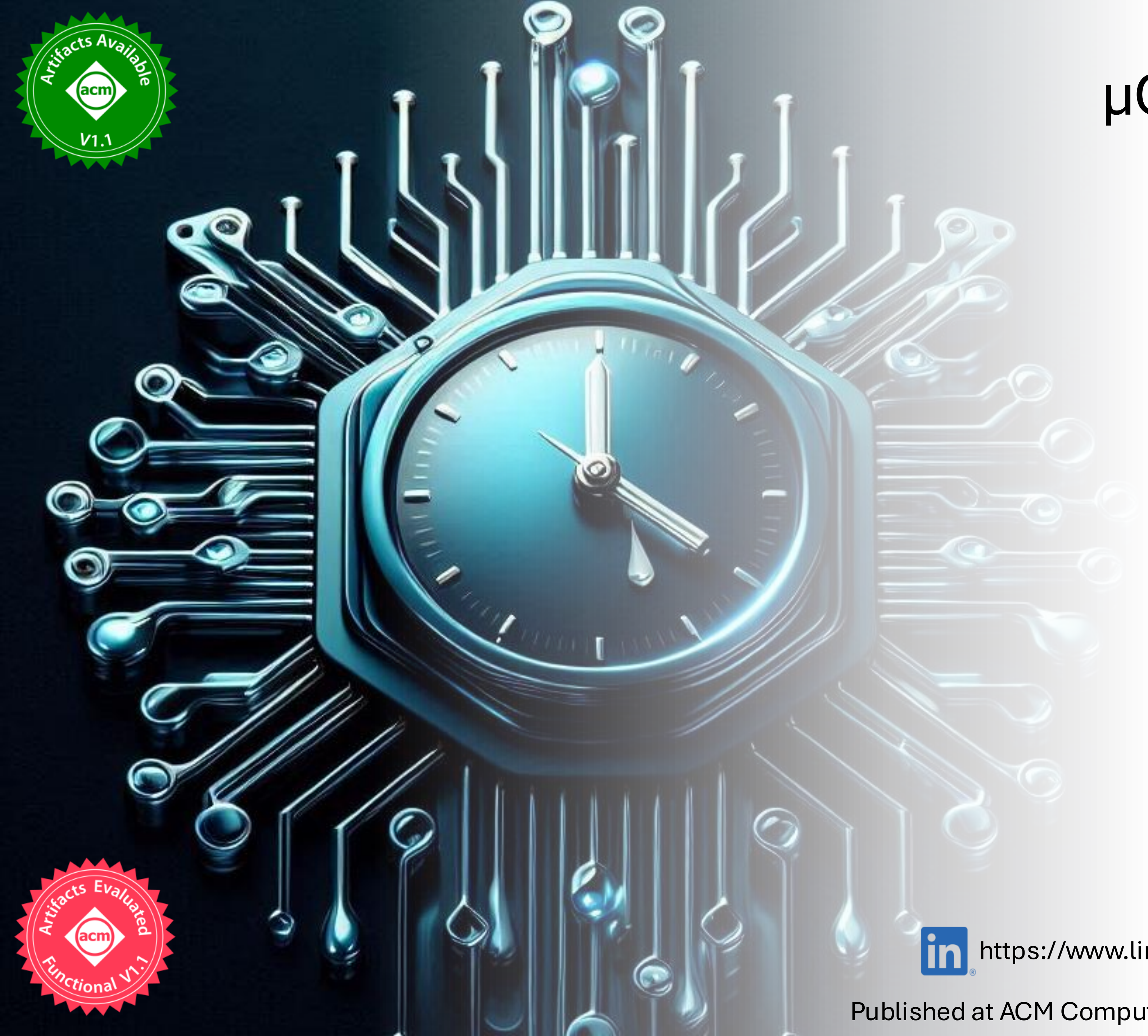
Artifacts Available acm V1.1



µCFI: Formal Verification of Microarchitectural Control-flow Integrity

Katharina Ceesay-Seitz, Flavien Solt,

Kaveh Razavi

COMSEC, Computer Security Group,

ETH Zurich

Verification Futures UK, 01.07.2025

COMSEC

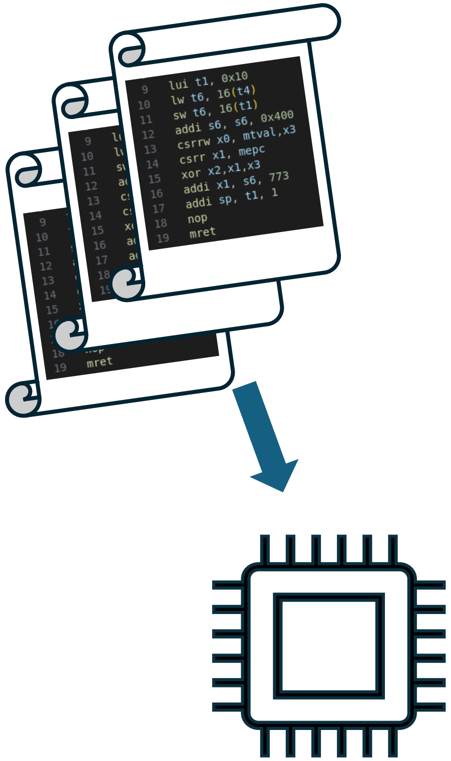
ETH zürich

https://www.linkedin.com/in/katharina-ceesay-seitz-ba521087/

Artifacts Evaluated acm Functional V1.1

Published at ACM Computer and Communications Security (CCS) 2024

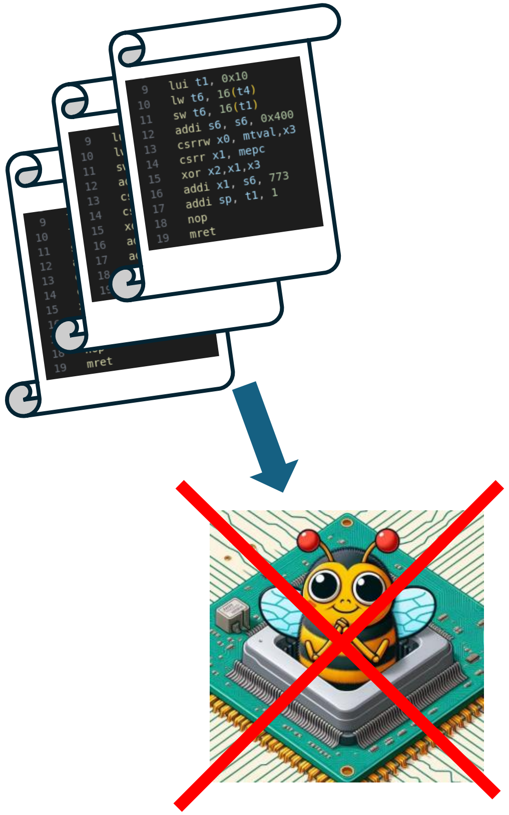
CPU Verification



```
lui t1, 0x10
lw t6, 16(t4)
sw t6, 16(t1)
addi s6, s6, 0x400
csrrw x0, mtval,x3
csrr x1, mepc
xor x2,x1,x3
addi x1, s6, 773
addi sp, t1, 1
nop
mret
```

Testing, e.g., fuzzing

CPU Verification

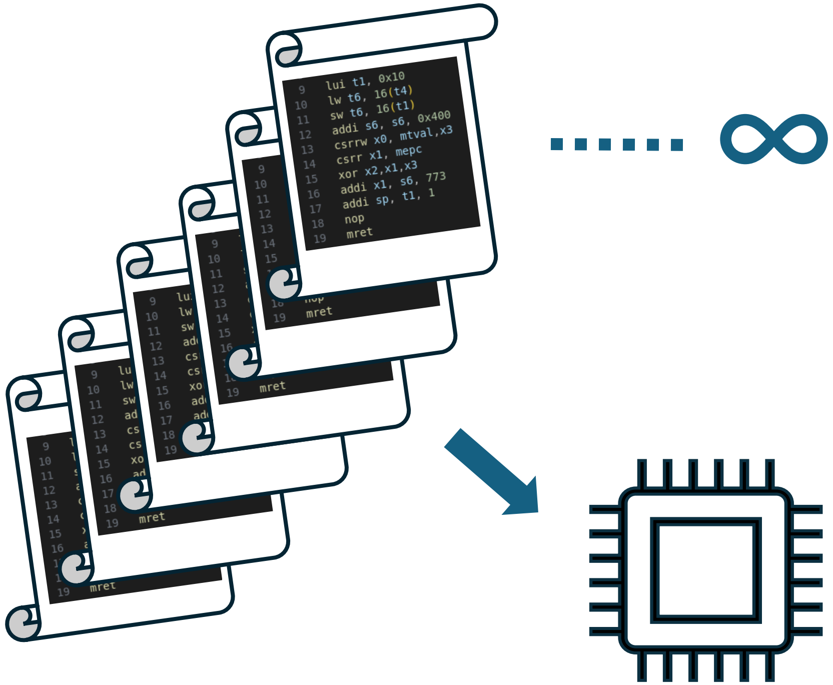


```
lui t1, 0x10
lw t6, 16(t4)
sw t6, 16(t1)
addi s6, s6, 0x400
csrrw x0, mtval,x3
csrr x1, mepc
xor x2,x1,x3
addi x1, s6, 773
addi sp, t1, 1
nop
mret
```

Testing, e.g., fuzzing, is incomplete

Security: need guarantee of absence of bugs

CPU Verification

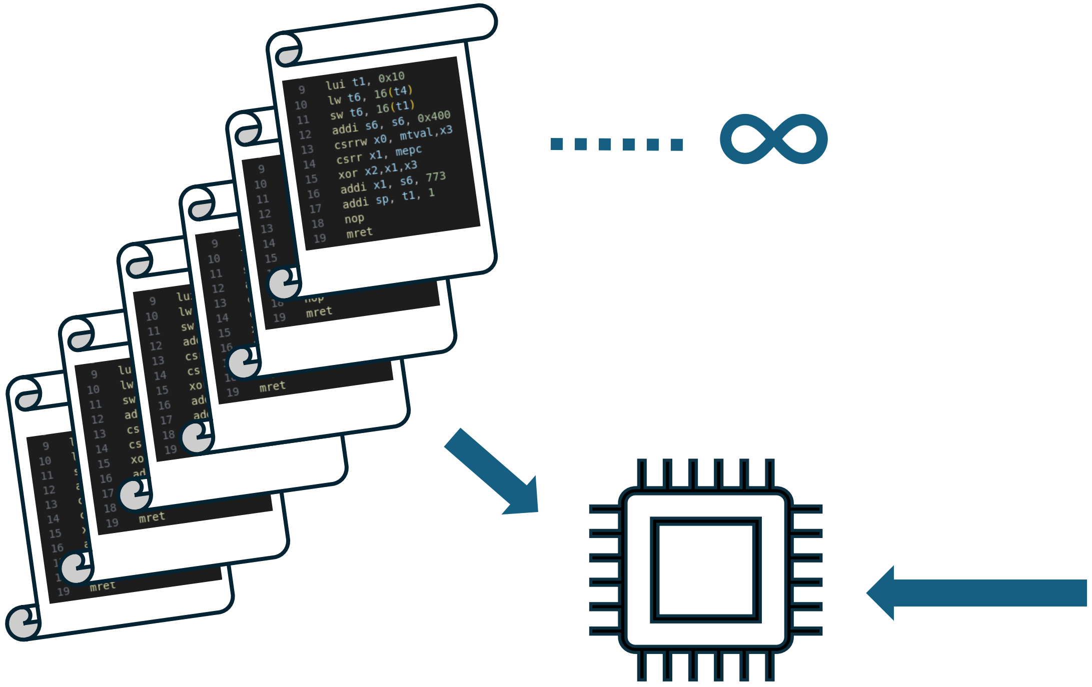


```
lui t1, 0x10
lw t6, 16(t4)
sw t6, 16(t1)
addi s6, s6, 0x400
csrrw x0, mtval,x3
csrr x1, mepc
xor x2,x1,x3
addi x1, s6, 773
addi sp, t1, 1
nop
mret
```

Formal verification:

- Provides formal guarantees for all inputs

CPU Verification



```
lui t1, 0x10
lw t6, 16(t4)
sw t6, 16(t1)
addi s6, s6, 0x400
csrrw x0, mtval,x3
csrr x1, mepc
xor x2,x1,x3
addi x1, s6, 773
addi sp, t1, 1
nop
mret
```

Formal verification:

- Provides formal guarantees for all inputs

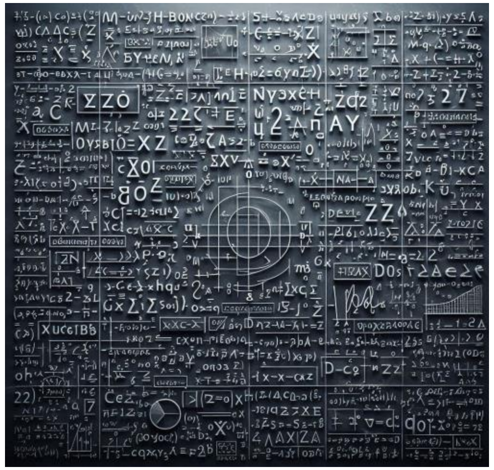
- Often a CPU-specific, manual effort

Formal Property Verification

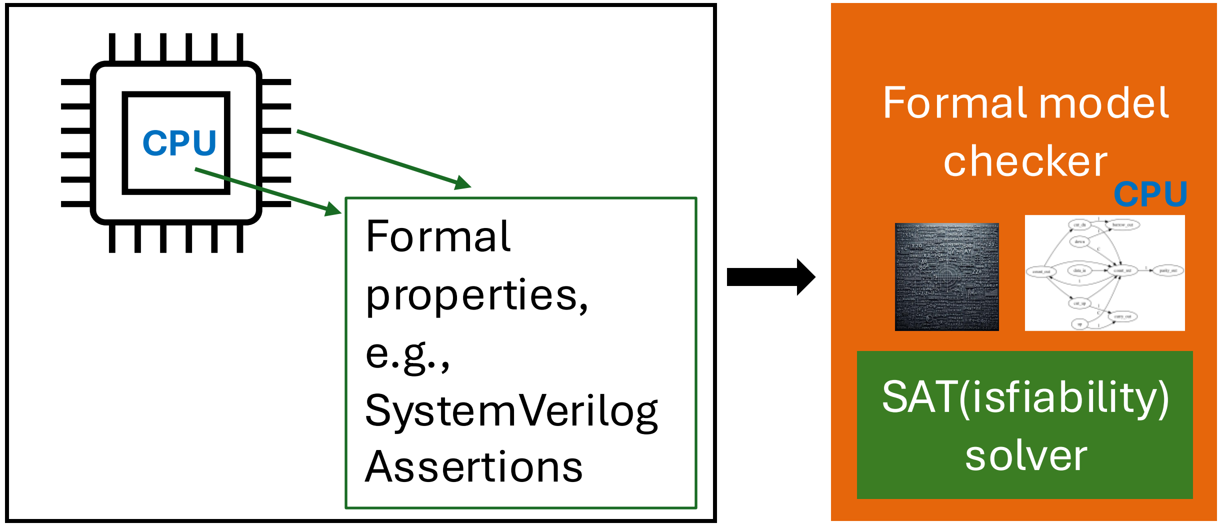
CPU

HDL (Hardware Description Language) Design

Formal properties, e.g., **SystemVerilog Assertions** describe desired behavior



Formal Property Verification



CPU

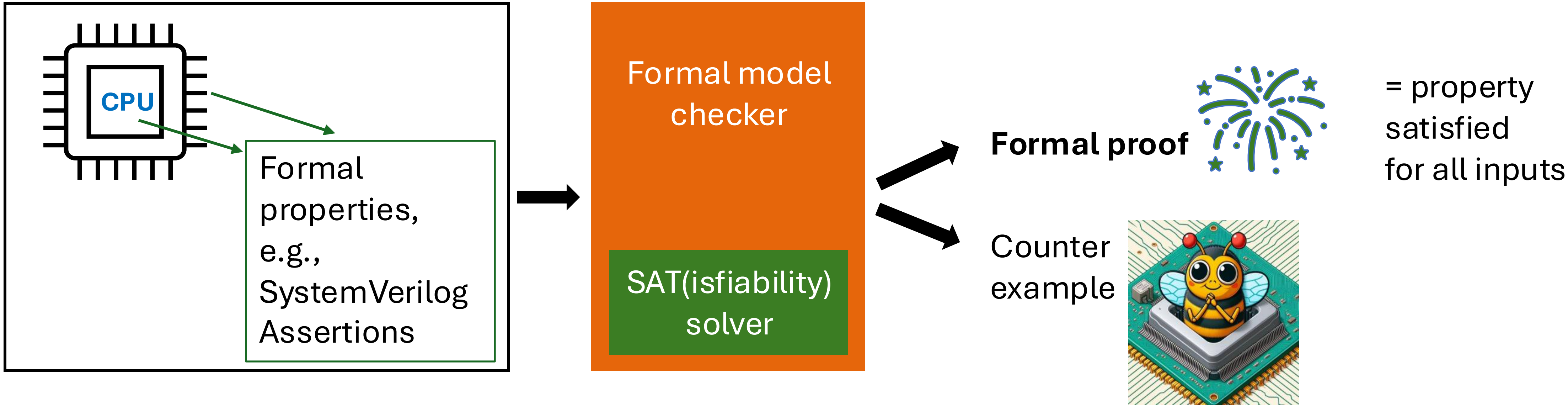
Formal properties, e.g., SystemVerilog Assertions

Formal model checker

CPU

SAT(isfiability) solver

Formal Property Verification



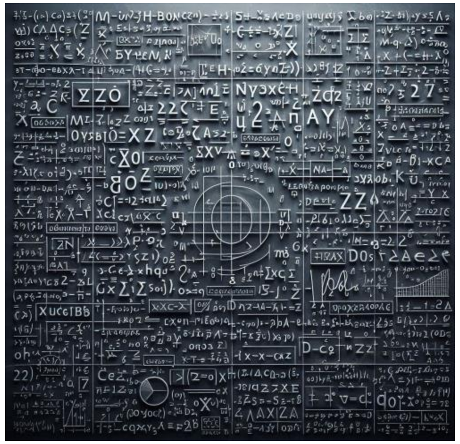
Formal properties, e.g., SystemVerilog Assertions

Formal model checker

SAT(isfiability) solver

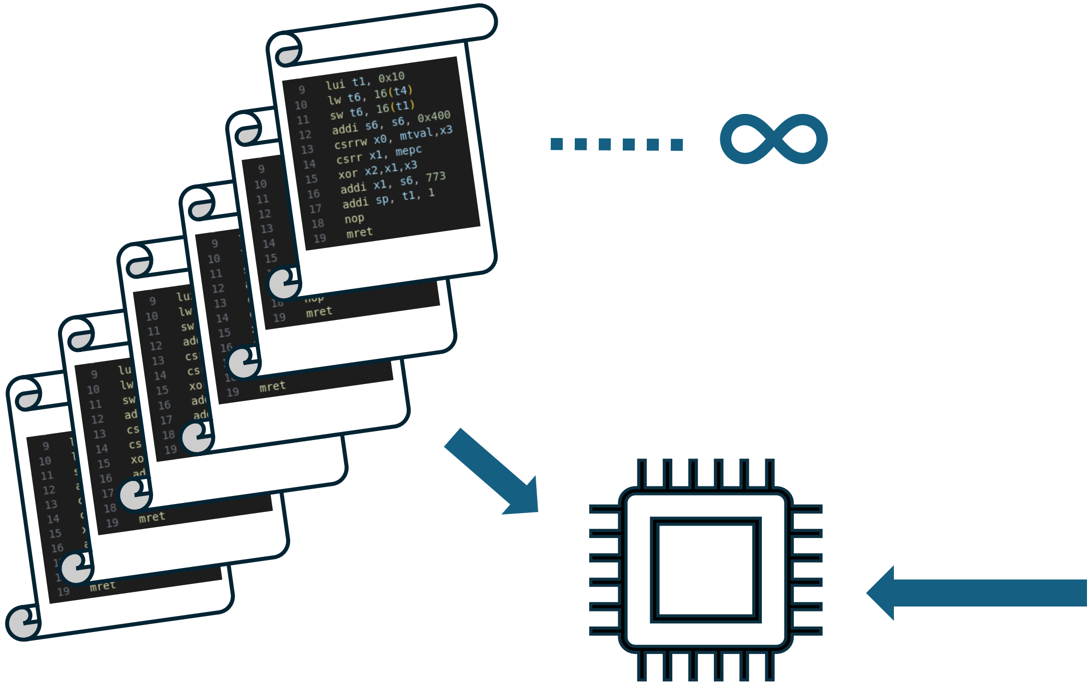
Formal proof = property satisfied for all inputs

Counter example



SIMPLER SOLUTION?

CPU Verification



```
lui t1, 0x10
lw t6, 16(t4)
sw t6, 16(t1)
addi s6, s6, 0x400
csrrw x0, mtval,x3
csrr x1, mepc
xor x2,x1,x3
addi x1, s6, 773
addi sp, t1, 1
nop
mret
```

Formal verification:

- Provides formal guarantees for all inputs

µCFI - Generalized security property

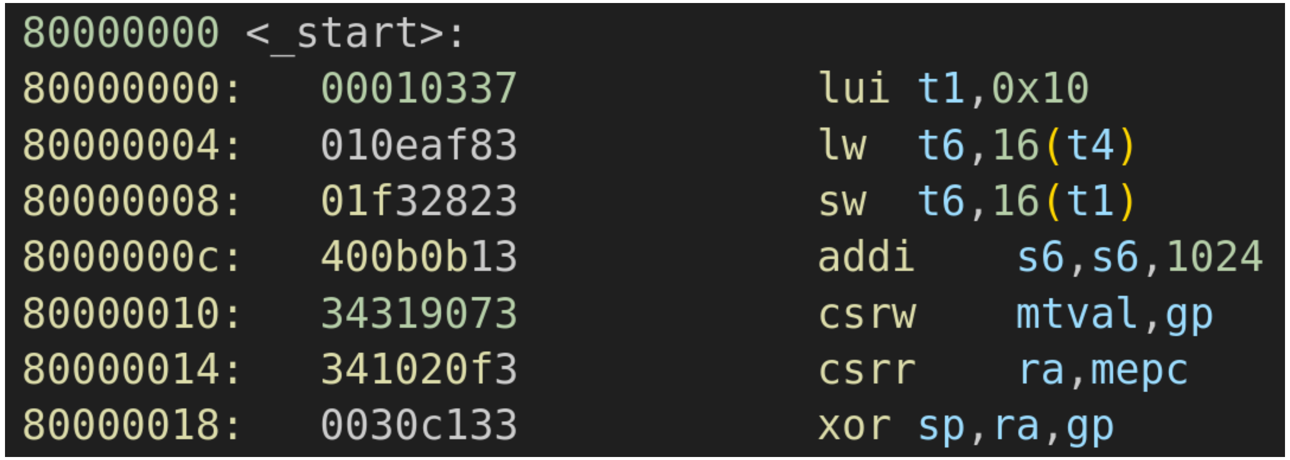
- Easy application and reuse
- Independent of CPU's verification state

  => apply it early in the design cycle
- Captures multiple threat models

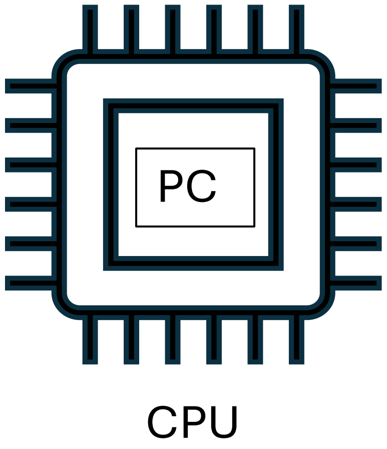
Definition: Architectural Control Flow

Software program (assembly instructions)

Architectural (software) Program Counter (PC) →



```
80000000 <_start>:
80000000:   00010337            lui t1,0x10
80000004:   010eaf83            lw  t6,16(t4)
80000008:   01f32823            sw  t6,16(t1)
8000000c:   400b0b13            addi    s6,s6,1024
80000010:   34319073            csrw    mtval,gp
80000014:   341020f3            csrr    ra,mepc
80000018:   0030c133            xor sp,ra,gp
```



Architectural PC decides the order of instructions

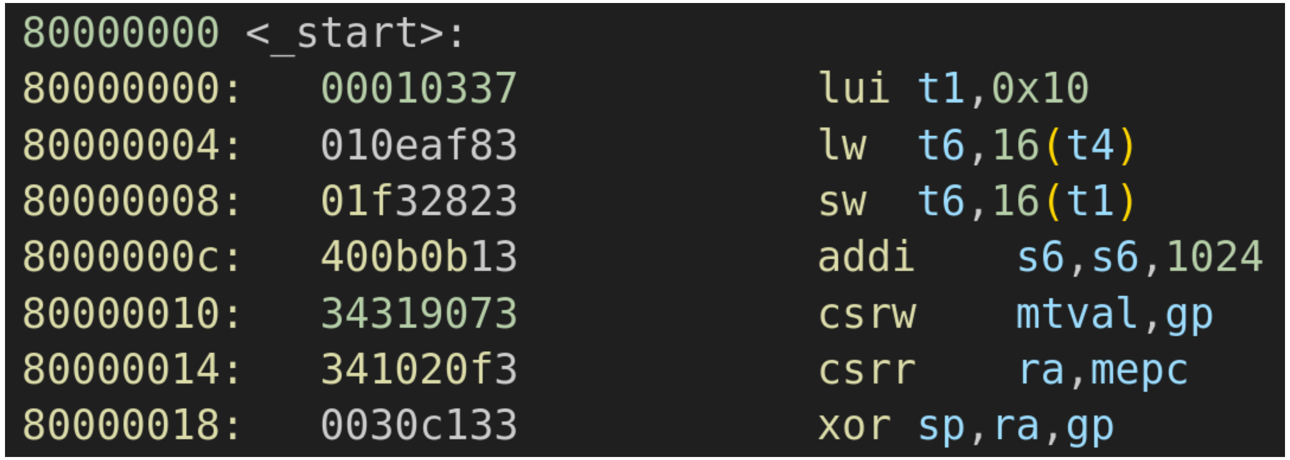
```
Software 'if'
      =
Branch instruction
```

```
If condition
     PC = Branch target = A
Else
     PC = Branch target = PC + 4
```

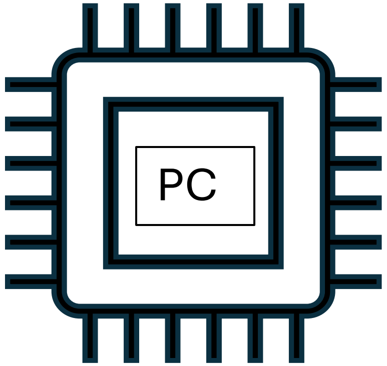
Definition: Microarchitectural Control Flow (μCF)

Architectural (software) Program Counter (PC) →

Software program (assembly instructions)

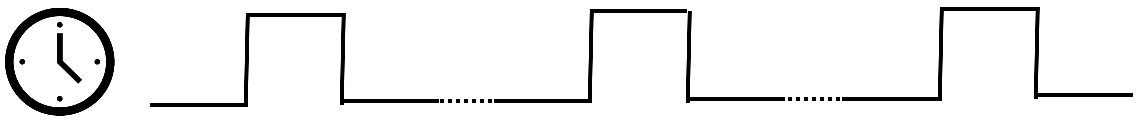


```
80000000 <_start>:
80000000:   00010337            lui t1,0x10
80000004:   010eaf83            lw  t6,16(t4)
80000008:   01f32823            sw  t6,16(t1)
8000000c:   400b0b13            addi    s6,s6,1024
80000010:   34319073            csrw    mtval,gp
80000014:   341020f3            csrr    ra,mepc
80000018:   0030c133            xor sp,ra,gp
```



CPU

Microarchitectural control flow (μCF)



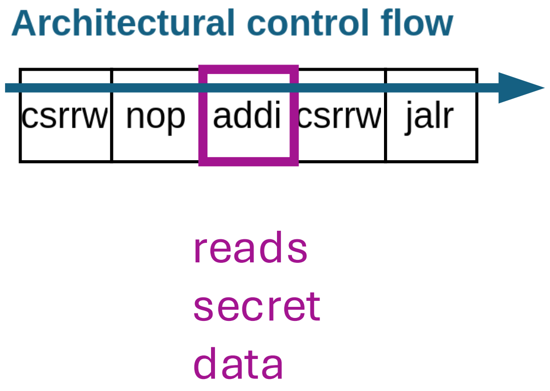
update time

Microarchitectural PC = a register inside the CPU

0x80000004 0x80000008 0x80001000 value

Microarchitectural Control Flow Violations

Constant Time (CT) RISC-V program



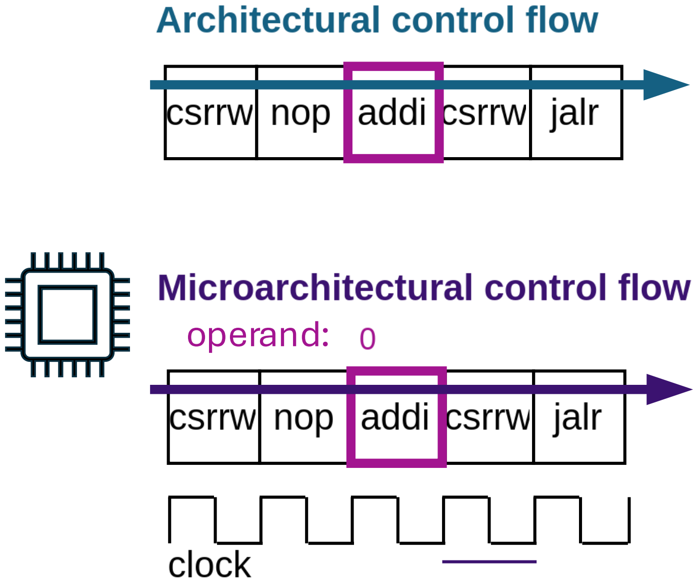
Architectural control flow

csrrw nop addi csrrw jalr

reads
secret
data

Microarchitectural Control Flow Violations

Constant Time (CT) RISC-V program

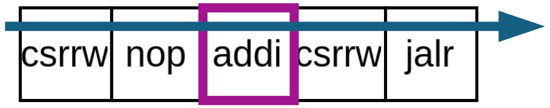


reads
secret
data

Microarchitectural Control Flow Violations

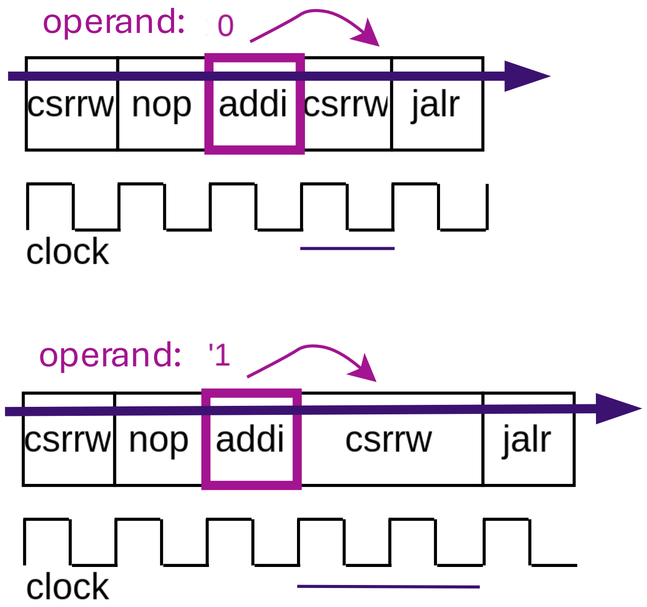
Constant Time (CT) program

Architectural control flow



csrrw nop addi csrrw jalr

Microarchitectural control flow



operand: 0

csrrw nop addi csrrw jalr

clock

operand: '1

csrrw nop addi csrrw jalr

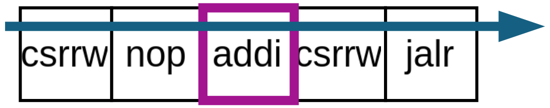
clock

PROBLEM?

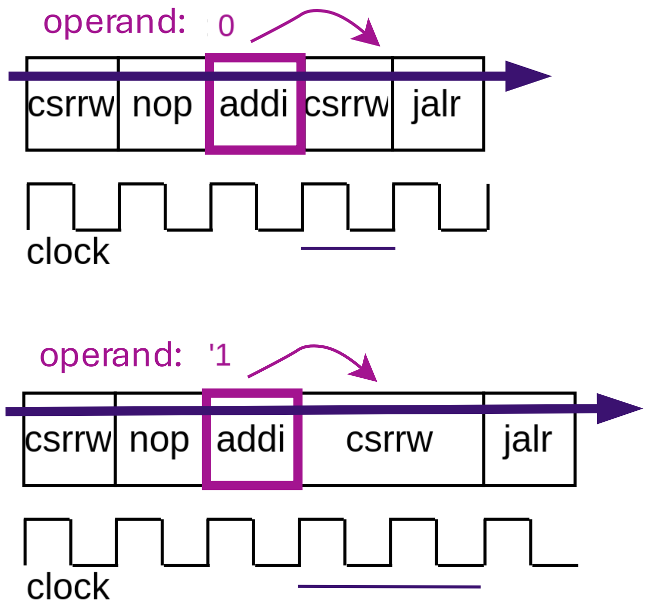
Microarchitectural Control Flow Violations

Constant Time (CT) program

Architectural control flow



Microarchitectural control flow



Secret influences µCF

Execution takes longer = timing side channel

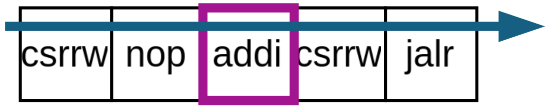
Delayed PC update

PC

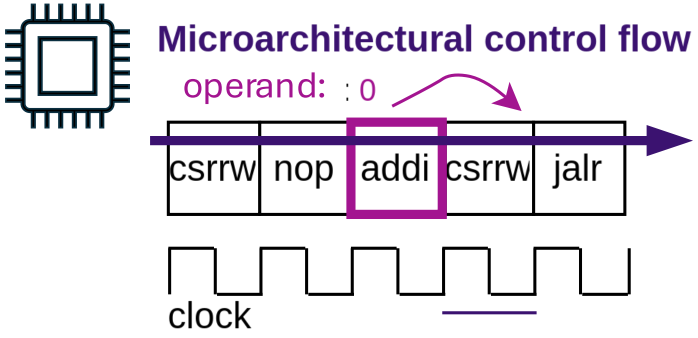
Microarchitectural Control Flow Violations

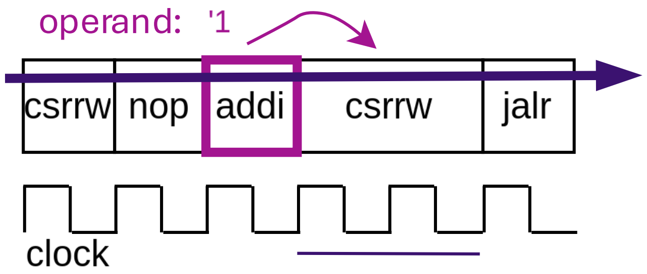
Constant Time (CT) program

Architectural control flow



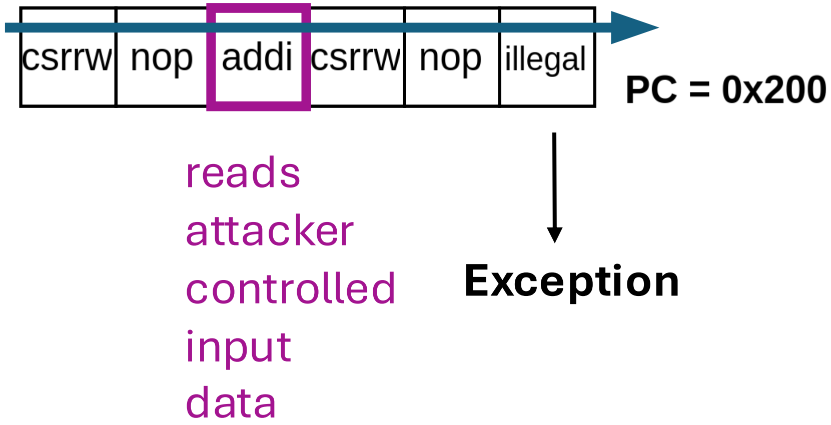
Microarchitectural control flow





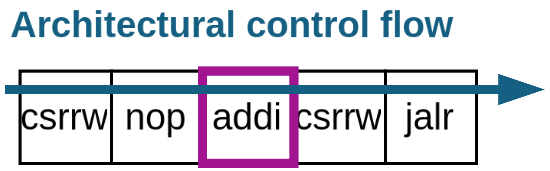
Control-flow integrity secure program

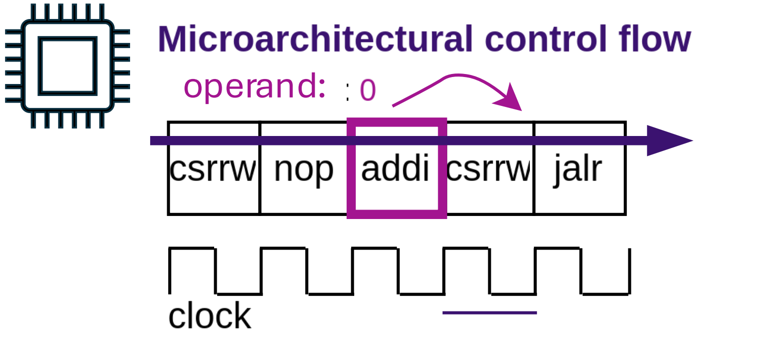
Architectural control flow

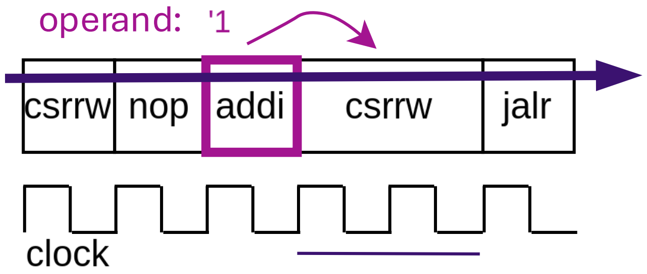


Microarchitectural Control Flow Violations

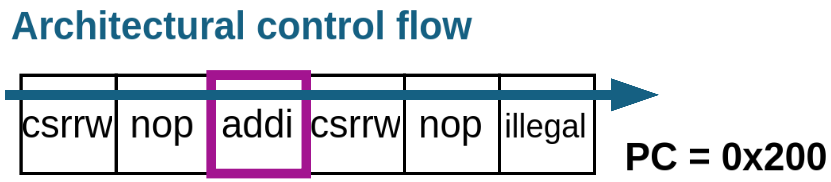
Constant Time (CT) program

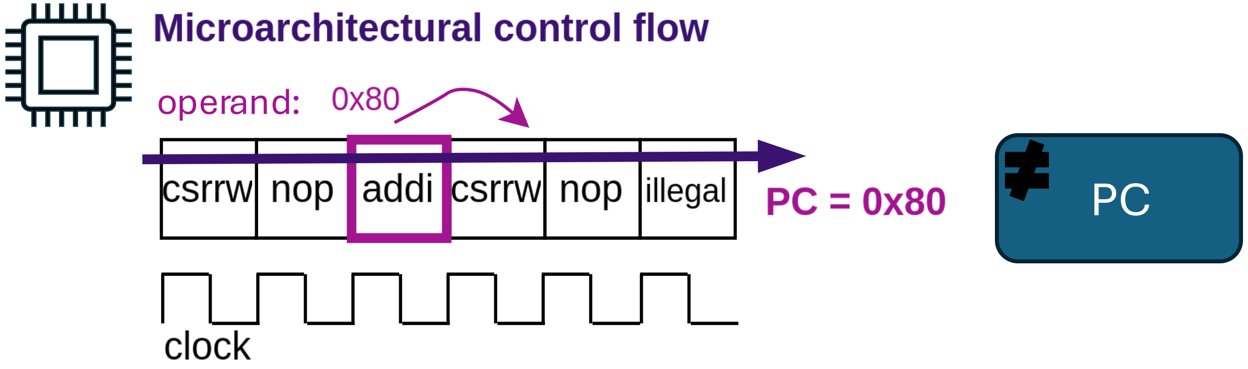






Control-flow integrity secure program

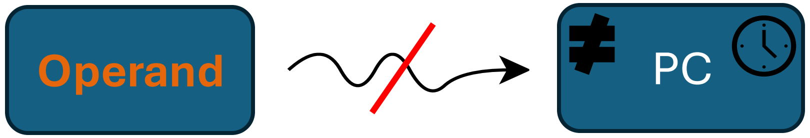




Input influences μCF
by changing PC value

µCFI - Microarchitectural Control-flow Integrity

- Proof that the µCF has only ISA-specified data dependencies
- Detect non-ISA specified flows



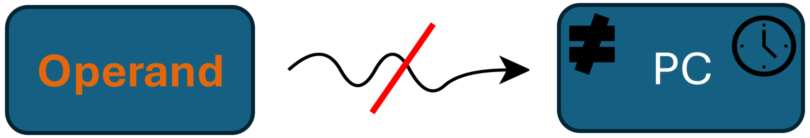
ISA = Instruction Set Architecture, PC = Program Counter

μCFI - Microarchitectural Control-flow Integrity

- Proof that the μCF has only ISA-specified data dependencies
- Detect non-ISA specified flows

One property

Two threat models captured



Operand

PC

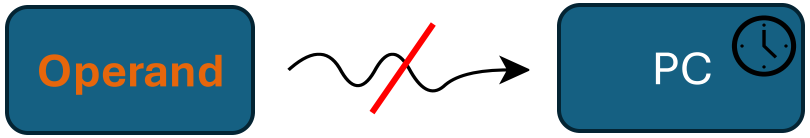
ISA = Instruction Set Architecture, PC = Program Counter

µCFI - Microarchitectural Control-flow Integrity

- Proof that the µCF has only ISA-specified data dependencies
- Detect non-ISA specified flows

One property

Two threat models captured



Operand

PC

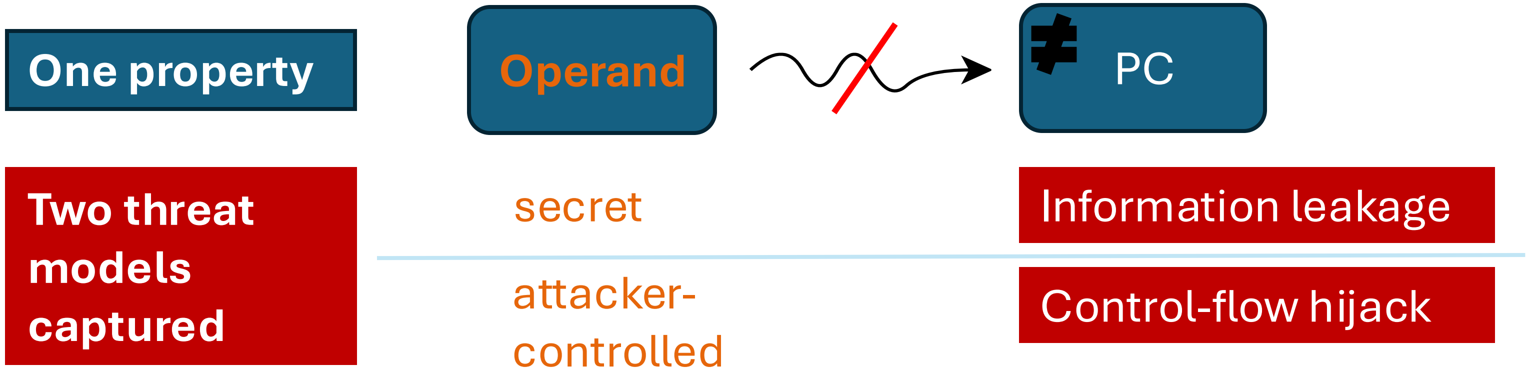
secret

Information leakage

ISA = Instruction Set Architecture, PC = Program Counter

µCFI - Microarchitectural Control-flow Integrity

- Proof that the µCF has only ISA-specified data dependencies
- Detect non-ISA specified flows



One property

Operand

≠ PC

Two threat models captured

secret

Information leakage

attacker-controlled

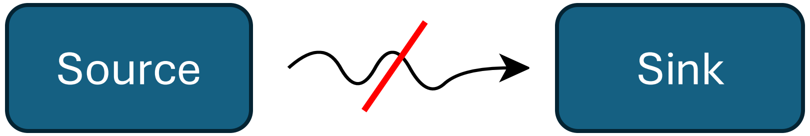
Control-flow hijack

ISA = Instruction Set Architecture, PC = Program Counter

µCFI - Microarchitectural Control-flow Integrity

- Proof that the µCF has only ISA-specified data dependencies
- Detect non-ISA specified flows

Information flow property

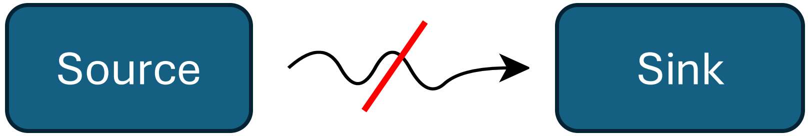


ISA = Instruction Set Architecture

µCFI - Microarchitectural Control-flow Integrity

- Proof that the µCF has only ISA-specified data dependencies
- Detect non-ISA specified flows

Information flow property



Information =

≠ data flows

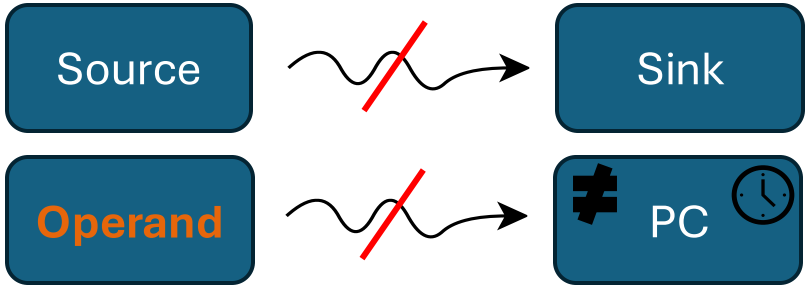
🕓 time & control flows

ISA = Instruction Set Architecture

µCFI - Microarchitectural Control-flow Integrity

- Proof that the µCF has only ISA-specified data dependencies
- Detect non-ISA specified flows

Information flow property



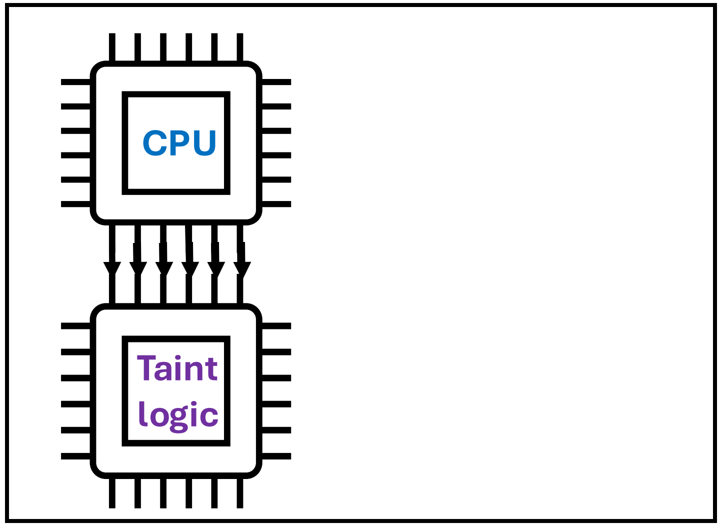
Information =

≠ data flows

🕓 time & control flows

ISA = Instruction Set Architecture, PC = Program Counter

Formal Verification of Information Flow

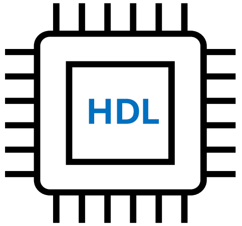


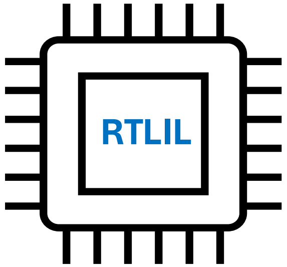
Information flow tracking with taint logic – CellIFT [1]

taint = **secret** or **attacker-controlled information**

[1] F. Solt, B. Gras, K. Razavi, "CELLIFT: Leveraging Cells for Scalable and Precise Dynamic Information Flow Tracking in RTL", USENIX Security 2022
https://github.com/comsec-group/cellift-yosys

CellIFT Yosys [1] pass

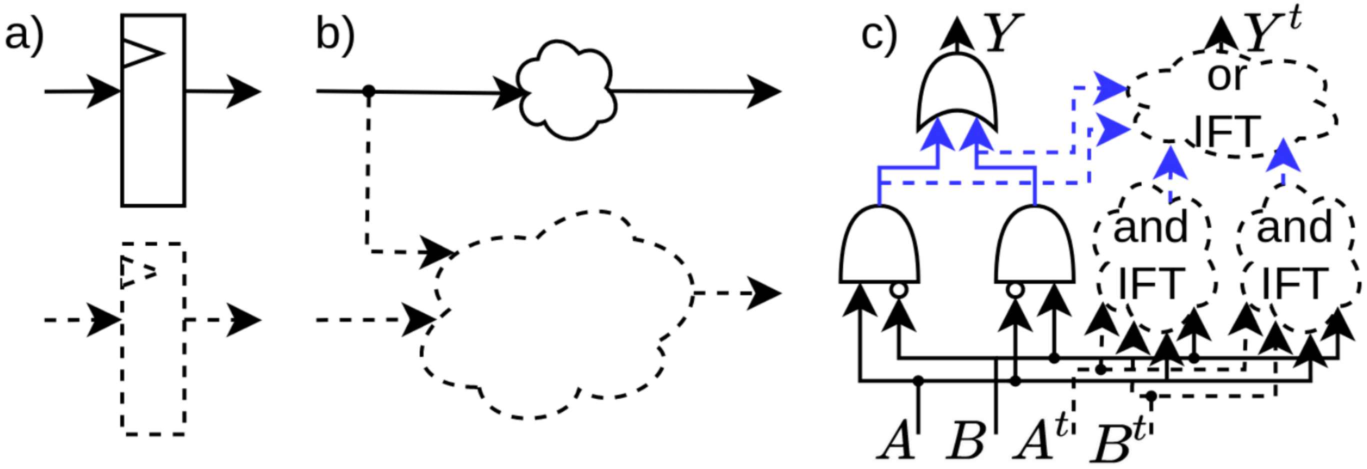


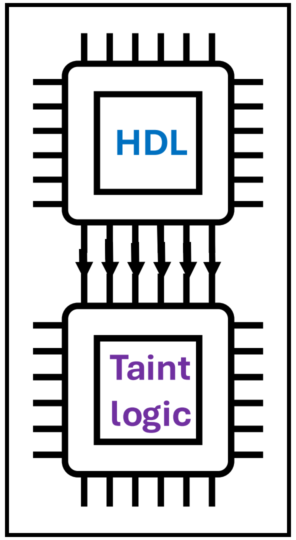


[2]

∀ cells (flip flops, logic cells, ...):

- Duplicate* in-/outputs for taint tracking
- Connect them with cell-type dependent taint tracking logic





a) State-holding cells

b) Combinational block

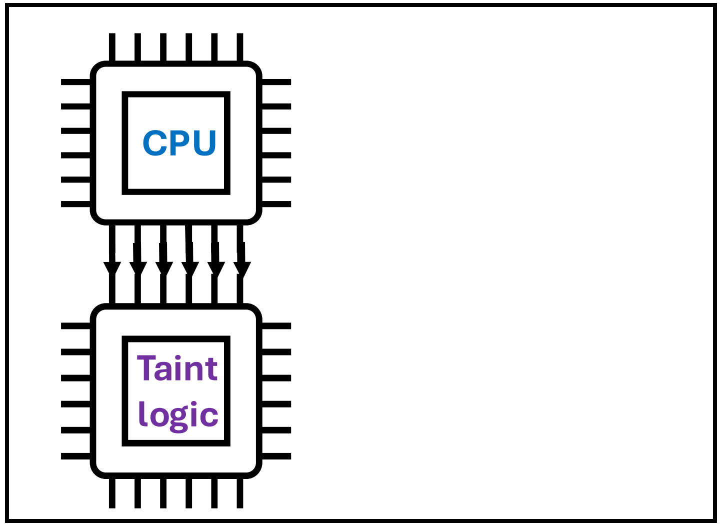
c) Gate-level output of Yosys

*it is possible to add multiple independent taint instrumentations. Each in-/output gets a taint representation per instrumentation.

[1] Yosys Open SYnthesis Suite - https://github.com/YosysHQ/yosys

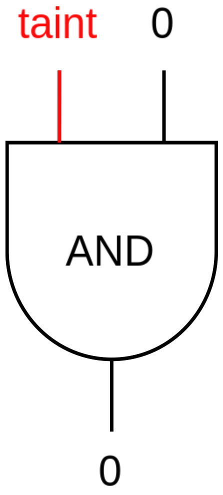
[2] F. Solt, B. Gras, K. Razavi, "CELLIFT: Leveraging Cells for Scalable and Precise Dynamic Information Flow Tracking in RTL", USENIX Security 2022

CellIFT



Taint logic (CellIFT [1]) tracks information flows

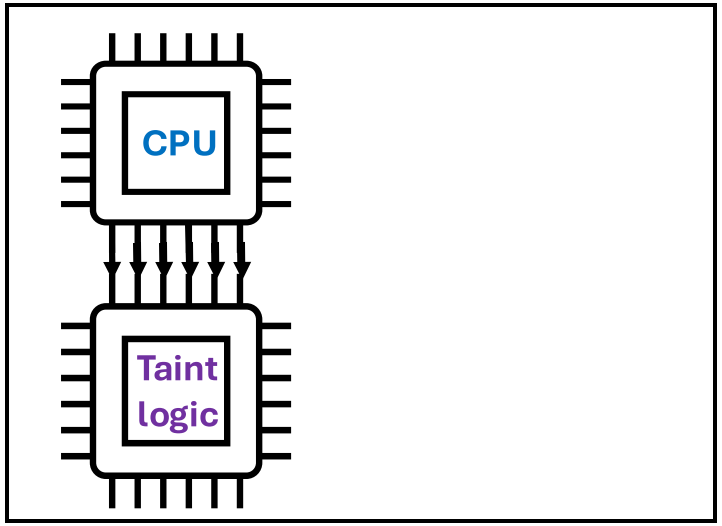
Information flow tracking with taint logic – CellIFT [1]



taint = **secret** or **attacker-controlled**

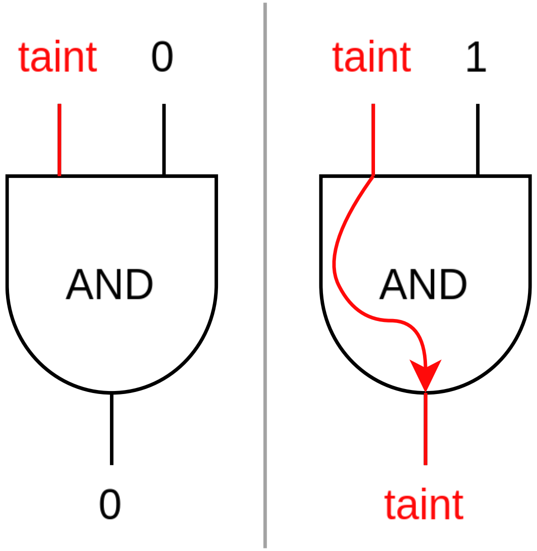
[1] F. Solt, B. Gras, K. Razavi, "CELLIFT: Leveraging Cells for Scalable and Precise Dynamic Information Flow Tracking in RTL", USENIX Security 2022

CellIFT



Taint logic (CellIFT [1]) tracks information flows

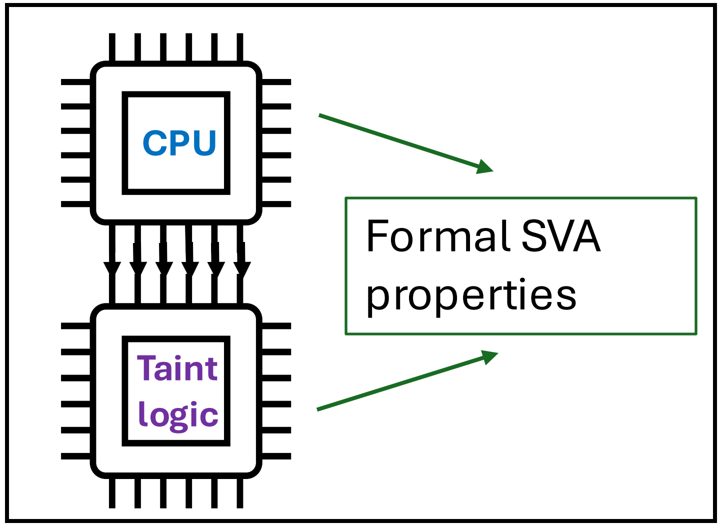
Information flow tracking with taint logic – CellIFT [1]



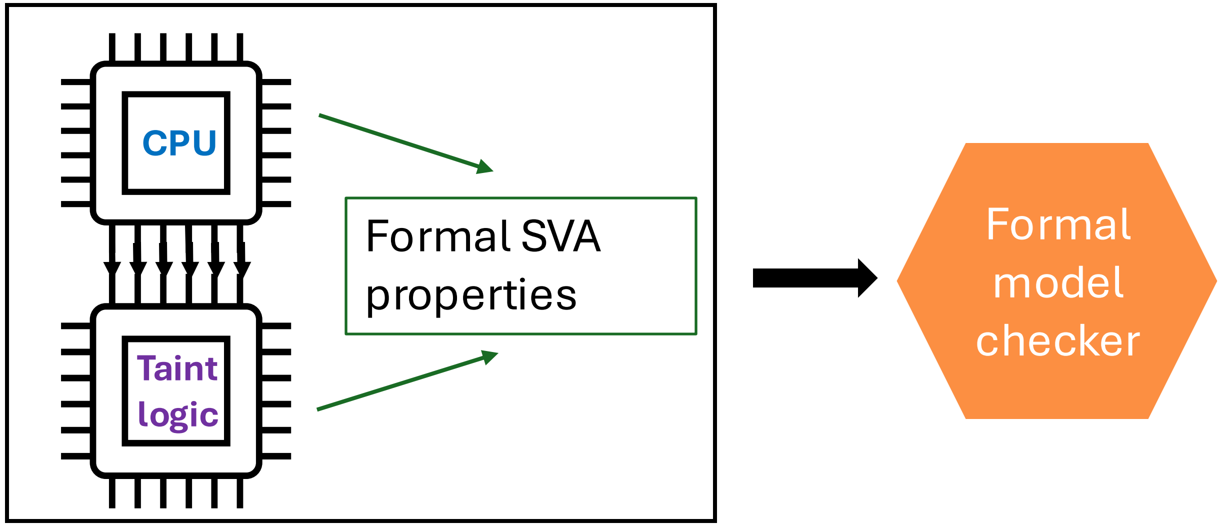
taint = **secret** or **attacker-controlled**

[1] F. Solt, B. Gras, K. Razavi, "CELLIFT: Leveraging Cells for Scalable and Precise Dynamic Information Flow Tracking in RTL", USENIX Security 2022

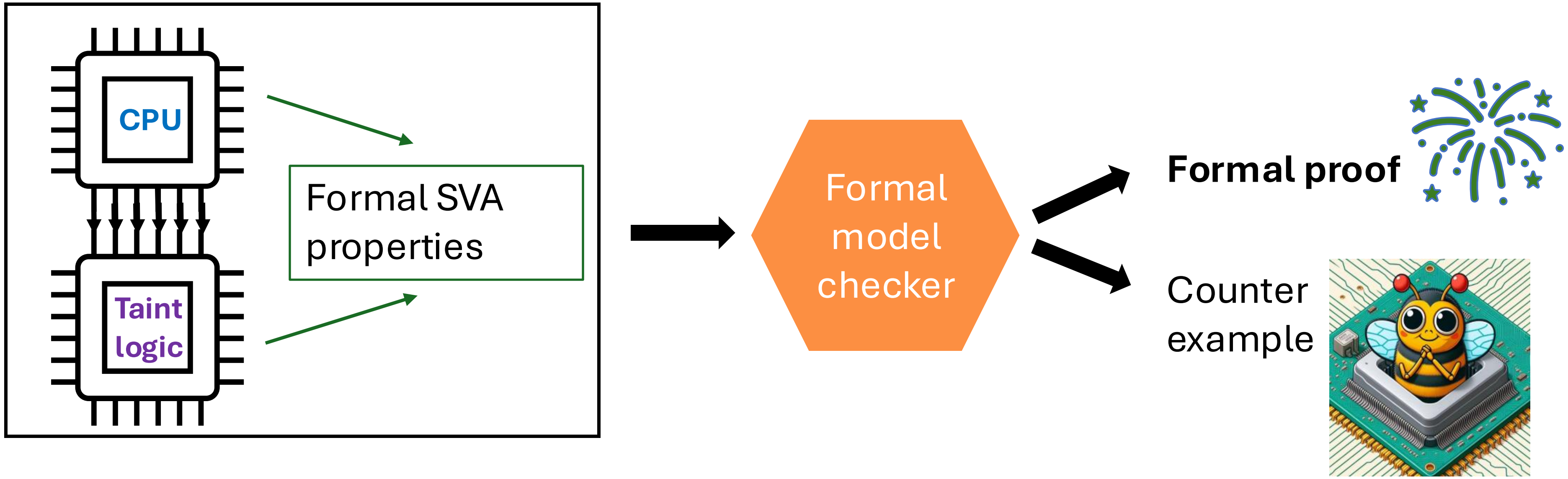
Formal Verification of Information Flow



Formal Verification of Information Flow



Formal Verification of Information Flow



CPU

Taint logic

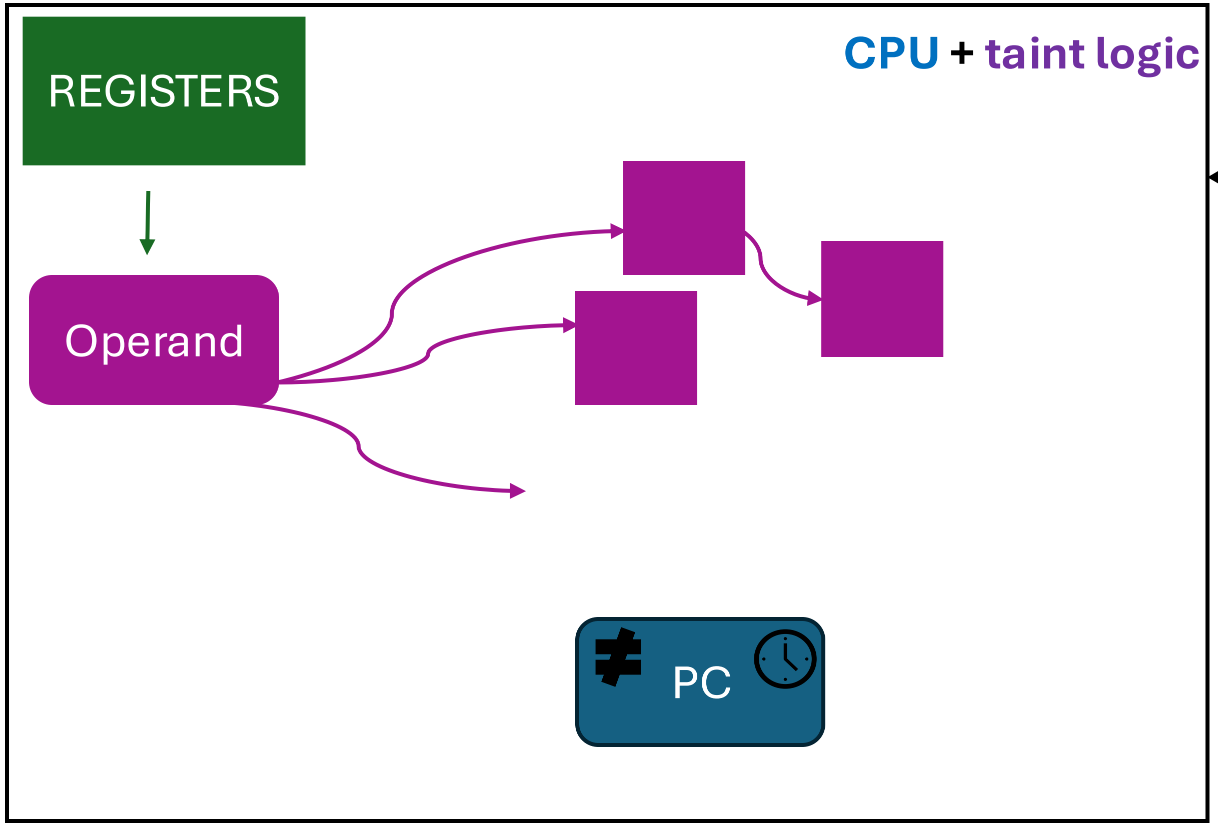
Formal SVA properties

Formal model checker

Formal proof

Counter example

Formally Verifying µCFI



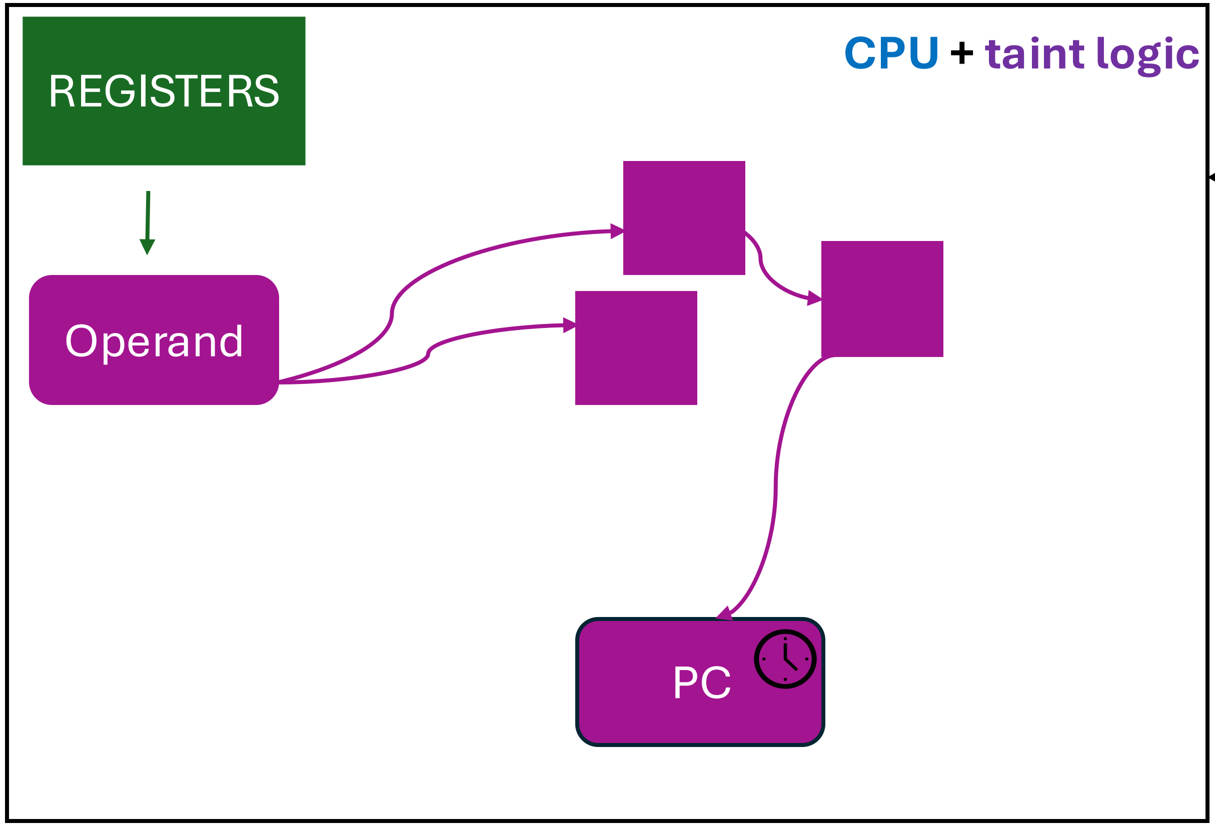
Instruction **ADD**

Taint does not reach the PC

taint = secret or attacker-controlled

PC = Program Counter

Formally Verifying µCFI



Instruction **BNE**

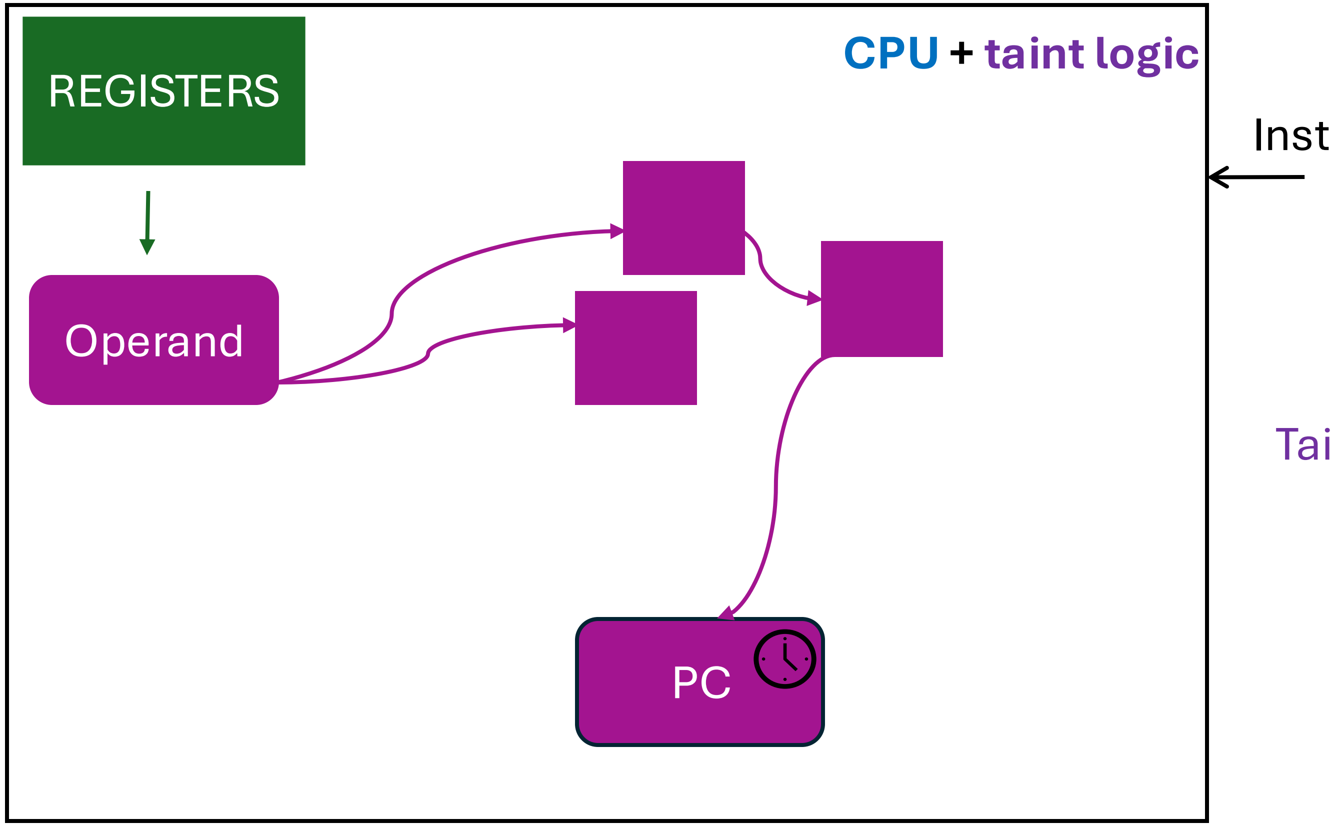
Branch Not Equal

Taint reaches the PC

taint = secret or attacker-controlled

PC = Program Counter

Formally Verifying μCFI



BNE

Branch Not Equal

Taint reaches the PC

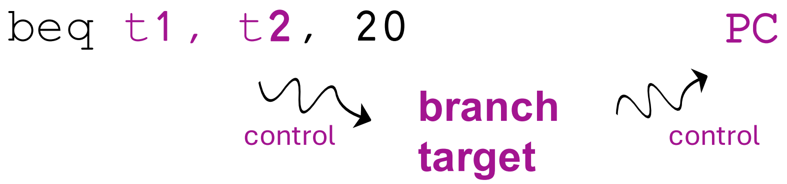
μCFI violated??

PC = Program Counter

Instruction Classification

Control-influencing:
direct branches,
instructions with
exceptions, …

are expected
to influence
the program counter



```
beq t1, t2, 20
```

control

branch target

control

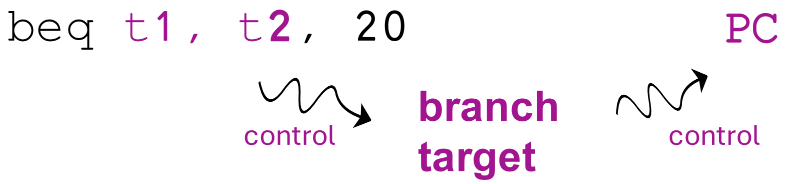
PC

```
If reg[t1] == reg[t2]
     Branch target = A
Else
     Branch target = PC + 4
```

Instruction Classification

Control-influencing:
branches,
instructions with
exceptions, …

are expected
to influence
the program counter



beq t1, t2, 20

control

branch target

control

PC

via implicit/control paths only

```
If reg[t1] == reg[t2]
     Branch target = A
Else
     Branch target = PC + 4
```

not via data paths

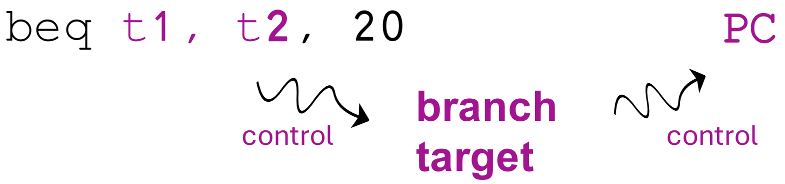
~~Program Counter = reg[t1]~~
~~Program Counter = reg[t2]~~

Instruction Classification

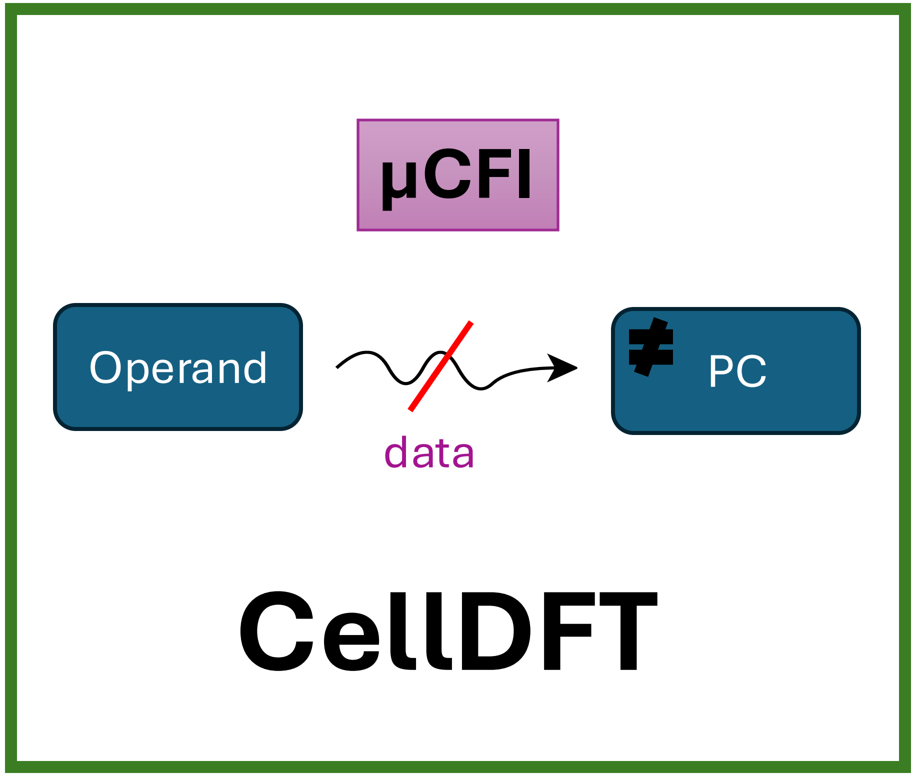
Control-influencing:
branches,
instructions with
exceptions, …

are expected
to influence
the program counter

via implicit/control paths only



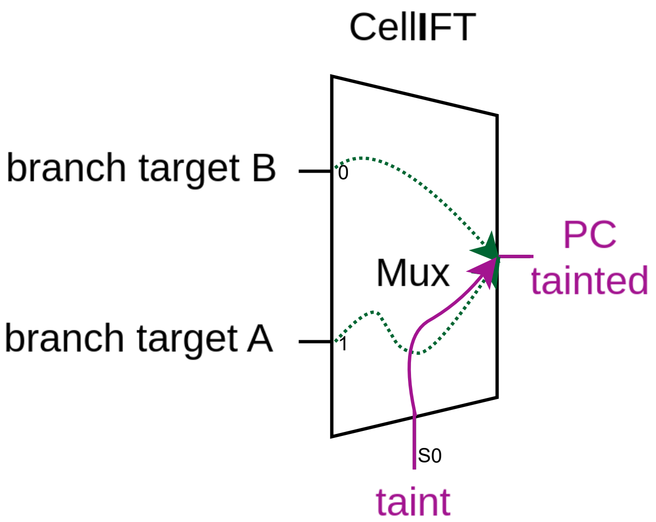
```
If reg[t1] == reg[t2]
      Branch target = A
Else
      Branch target = PC + 4
```



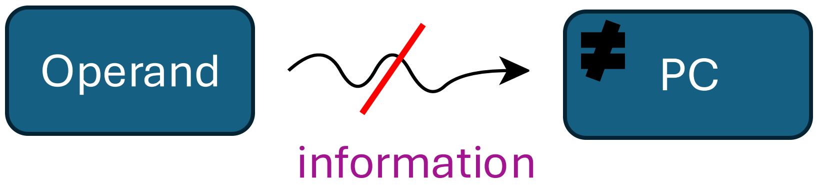
~~Program Counter = reg[t1]~~
~~Program Counter = reg[t2]~~

CellIFT

μCFI



if reg[t1] == reg[t2]



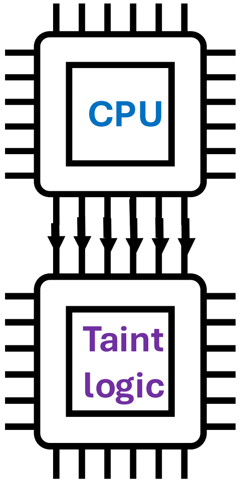
Non-influencing:
arithmetic, logic, …

CellIFT [1]

tracks information =

≠ data,

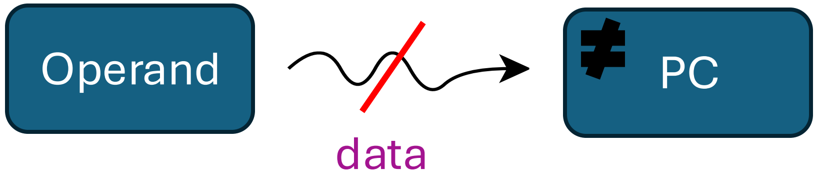
🕒 control & timing flows

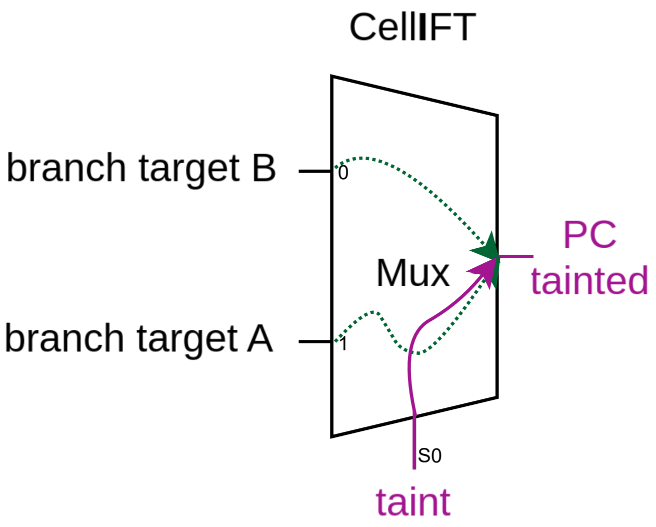


[1] F. Solt, B. Gras, K. Razavi, "CELLIFT: Leveraging Cells for Scalable and Precise Dynamic Information Flow Tracking in RTL", USENIX Security 2022

CellDFT – Data Flow Tracking

µCFI





reg[t1] == reg[t**2**]

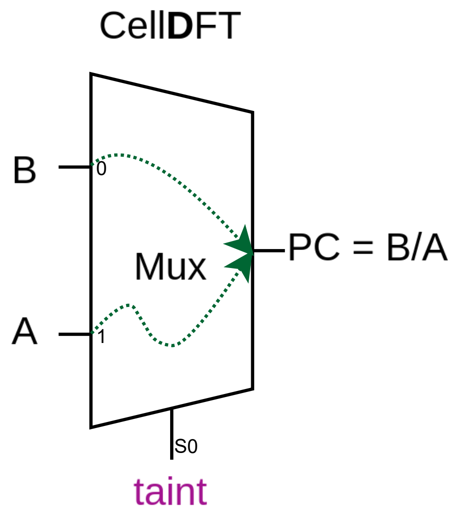
CellIFT [1]

tracks information =

≠ data,

🕓 control & timing flows

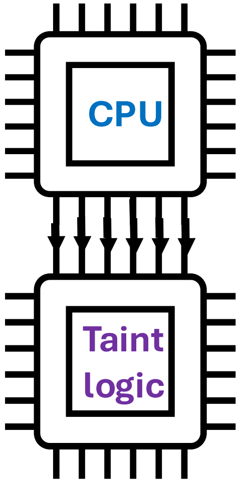
New:



CellDFT

only tracks

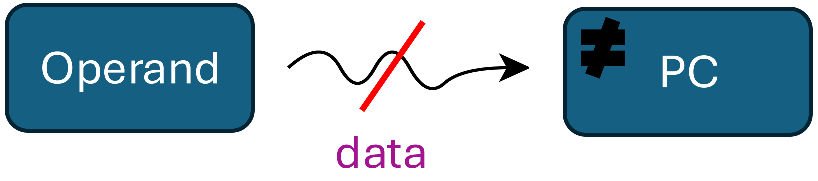
≠ **data flows**



[1] F. Solt, B. Gras, K. Razavi, "CELLIFT: Leveraging Cells for Scalable and Precise Dynamic Information Flow Tracking in RTL", USENIX Security 2022

CellDFT – Data Flow Tracking

μCFI



CellDFT

- Only tracks **data flows**
- Blocks control flows

Taint logic

CellDFT rules implemented in Yosys pass

Cell Name	Definition
State elems. with enable (EN)	$Q_n^t = (EN \wedge D^t) \vee (\neg EN \wedge Q_{n-1}^t)$
2-input mux, aldff cells [106]	$Y^t = (\neg S \wedge A^t) \vee (S \wedge B^t)$
pmux cells [106]	$Y^t = A[S]^t$
Comparison/reduction cells	$Y^t = 0$
Shift cells	$Y^t = A^t \circ B$

Data taint propagates

No taint propagates

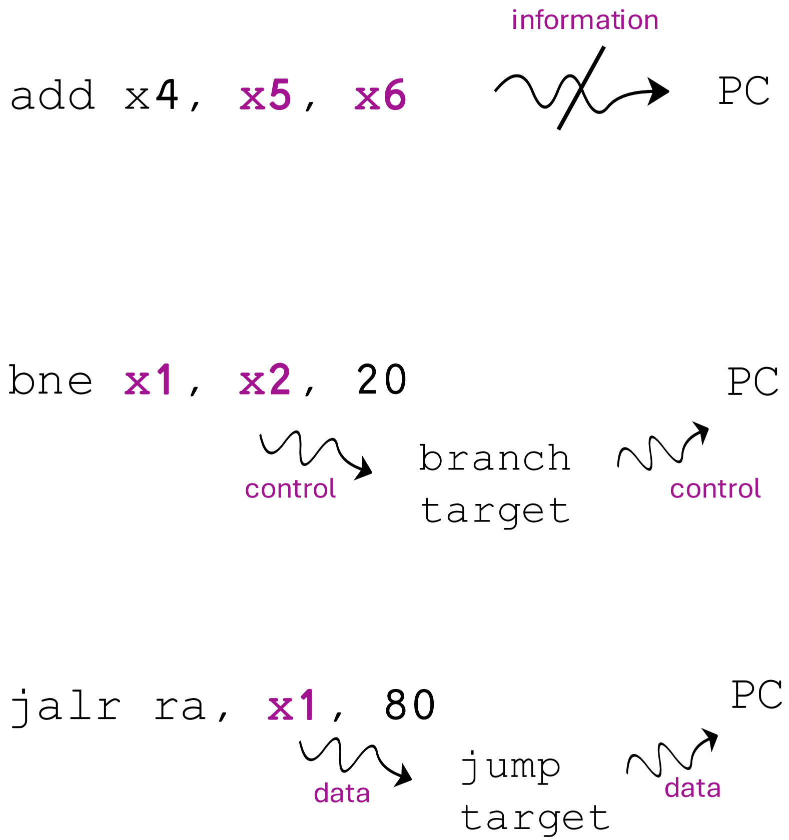
Data taint is shifted

Instruction classification

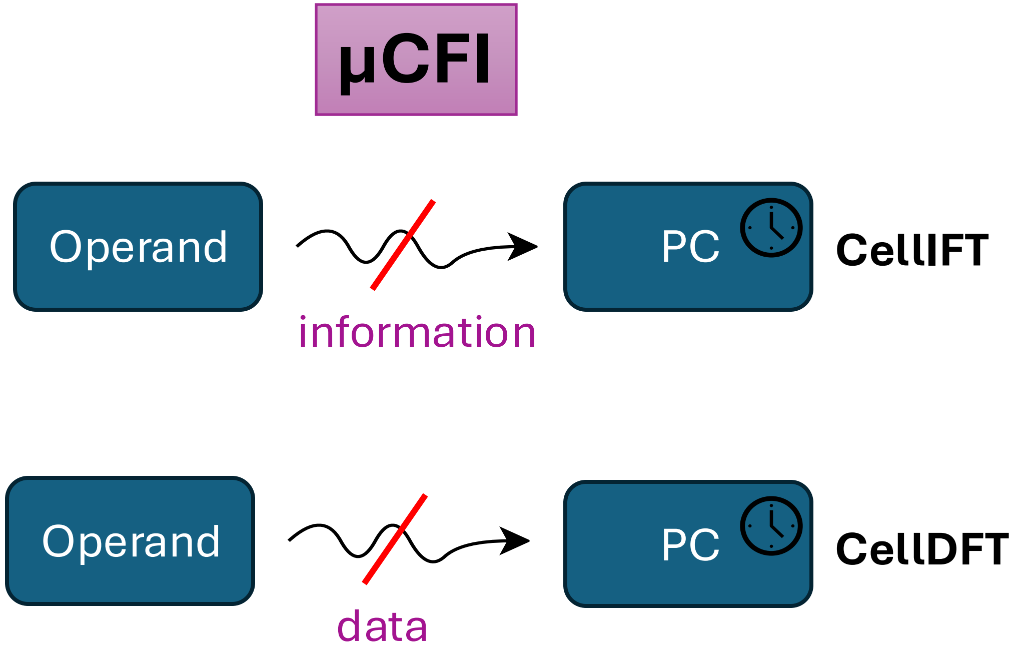
Non-influencing:
arithmetic, logic, …

Control-influencing:
branches, exceptions

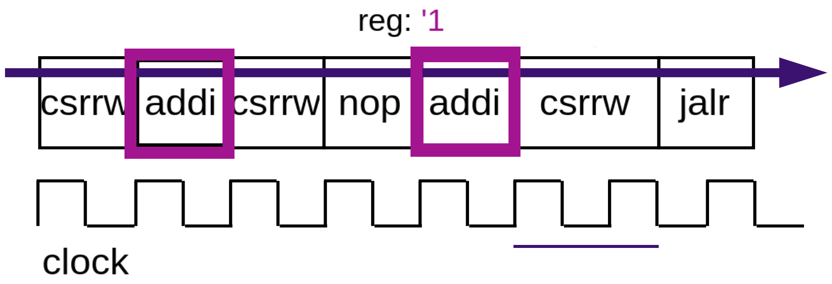
Value-influencing:
jumps



```
add x4, x5, x6
bne x1, x2, 20
jalr ra, x1, 80
```



µCFI - Verification Goals

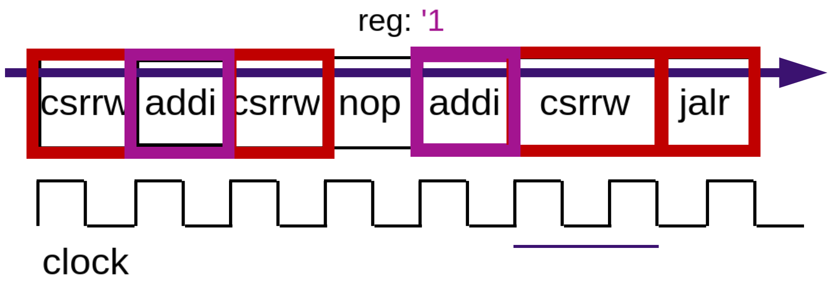


For communication with software engineers/tools:

- Security classification per instruction

Is a constant time (CT) software really CT on an actual CPU implementation?

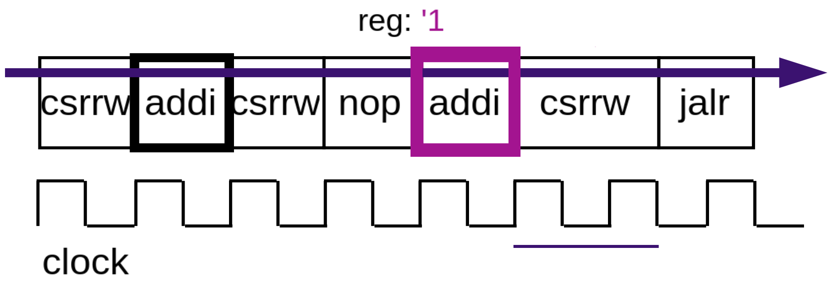
µCFI - Verification Goals



For communication with software engineers/tools:

- Security classification per instruction,
- surrounded by arbitrary, potentially insecure, instructions

µCFI - Verification Goals



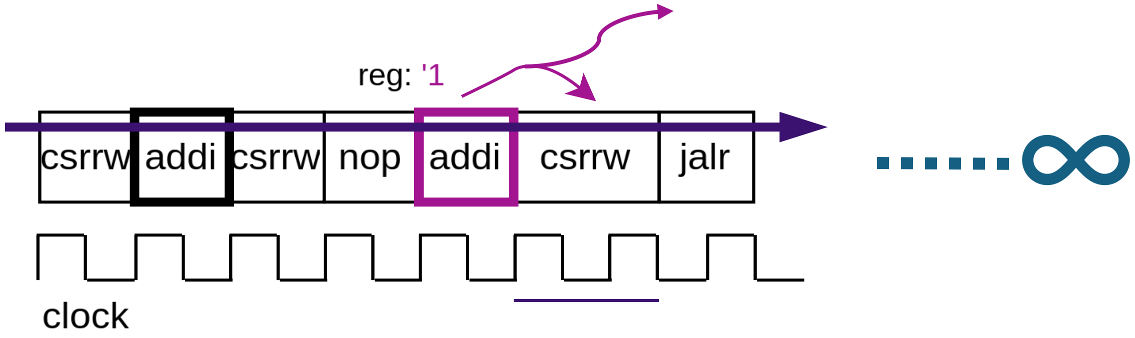
For communication with software:

- Security classification per instruction

To ease debugging:

- Identify the specific instruction that leaks

µCFI - Verification Goals



For communication with software:

- Security classification per instruction

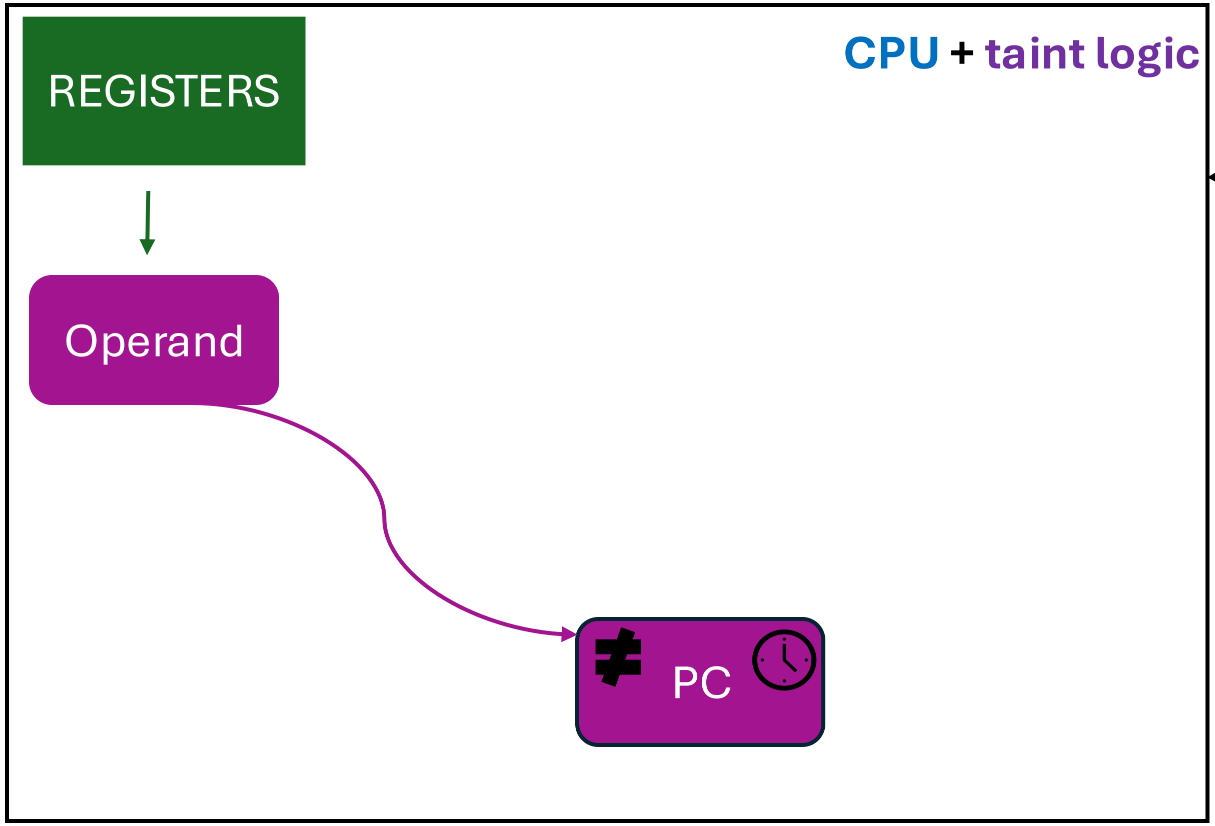
To ease debugging:

- Identify the specific instruction that leaks

For strong security guarantees:

- consider influences on younger instructions
- over arbitrary, infinitely long programs

Identifying Insecure Instructions



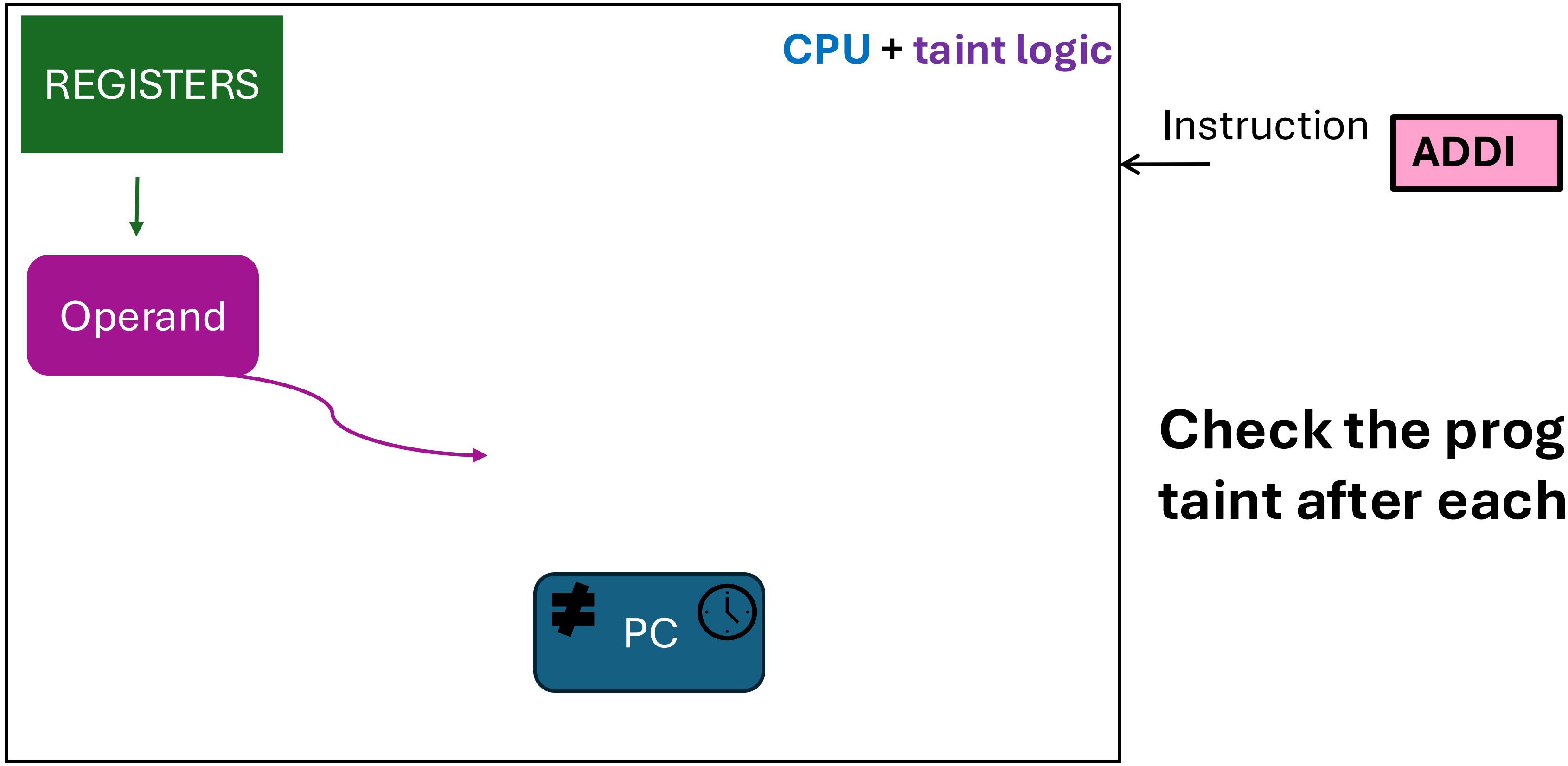
Instruction

ADDI	CSRRW

Control and Status Register
Read Write

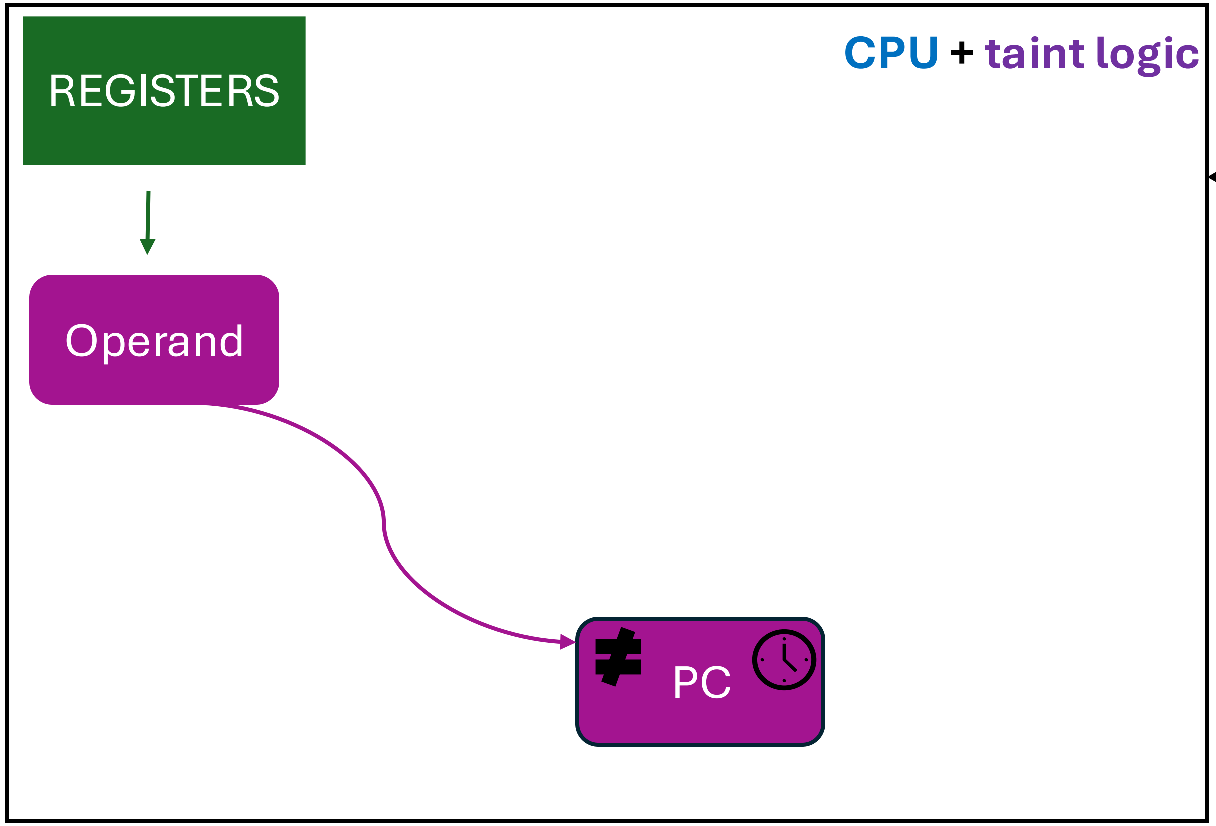
Which instruction tainted the PC?

Identifying Insecure Instructions



Check the program counter taint after each instruction?

Identifying Insecure Instructions



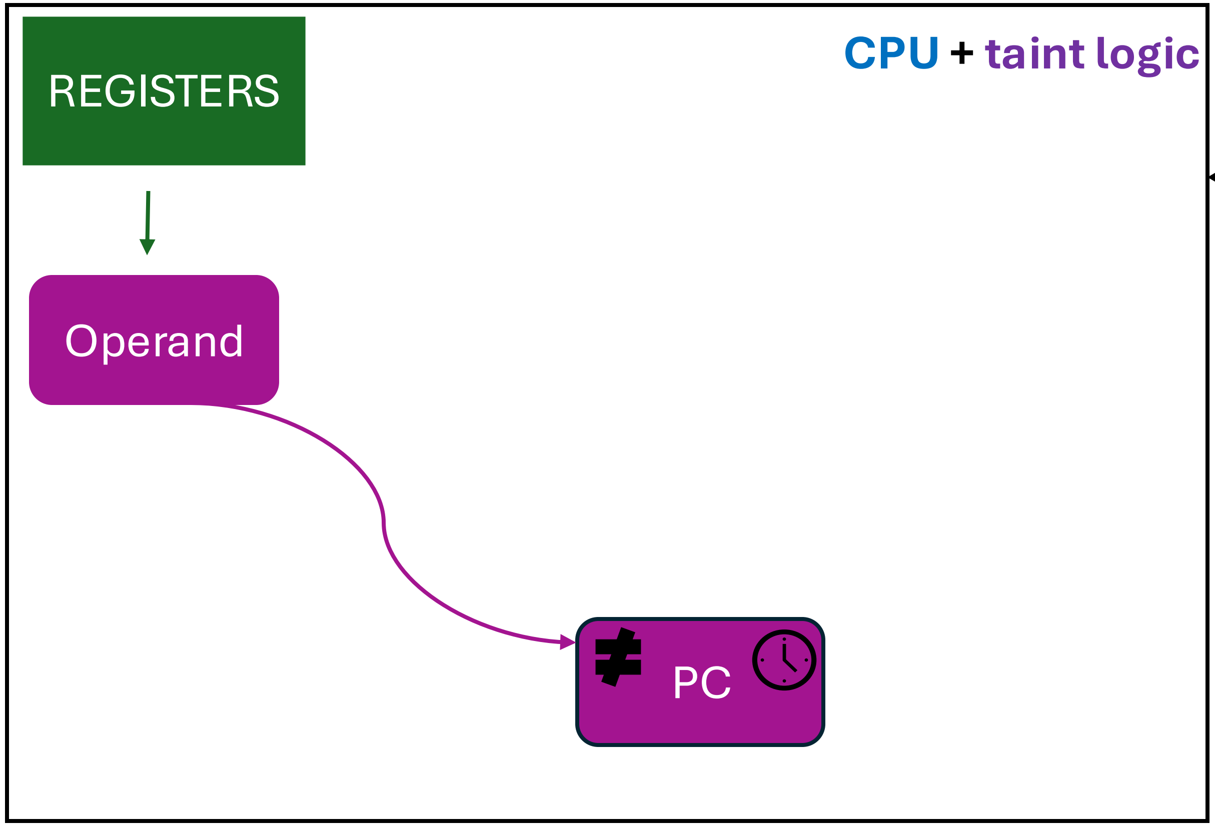
Instruction

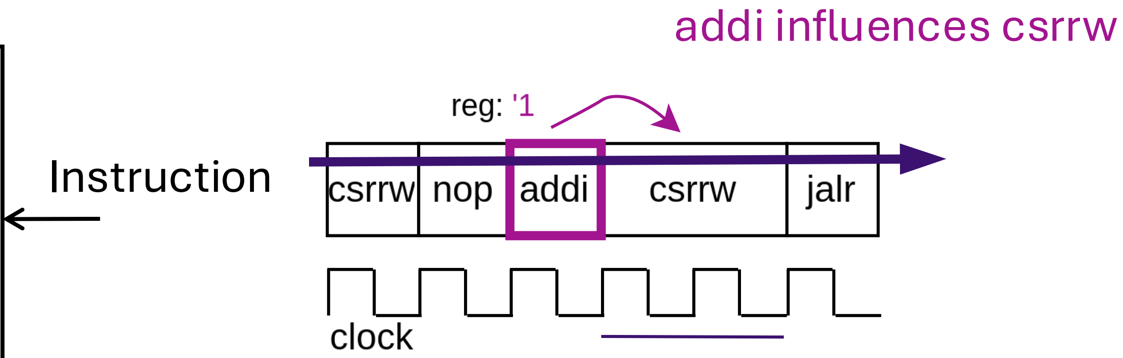
ADDI	CSRRW

Control and Status Register Read Write

Check the program counter taint after each instruction?

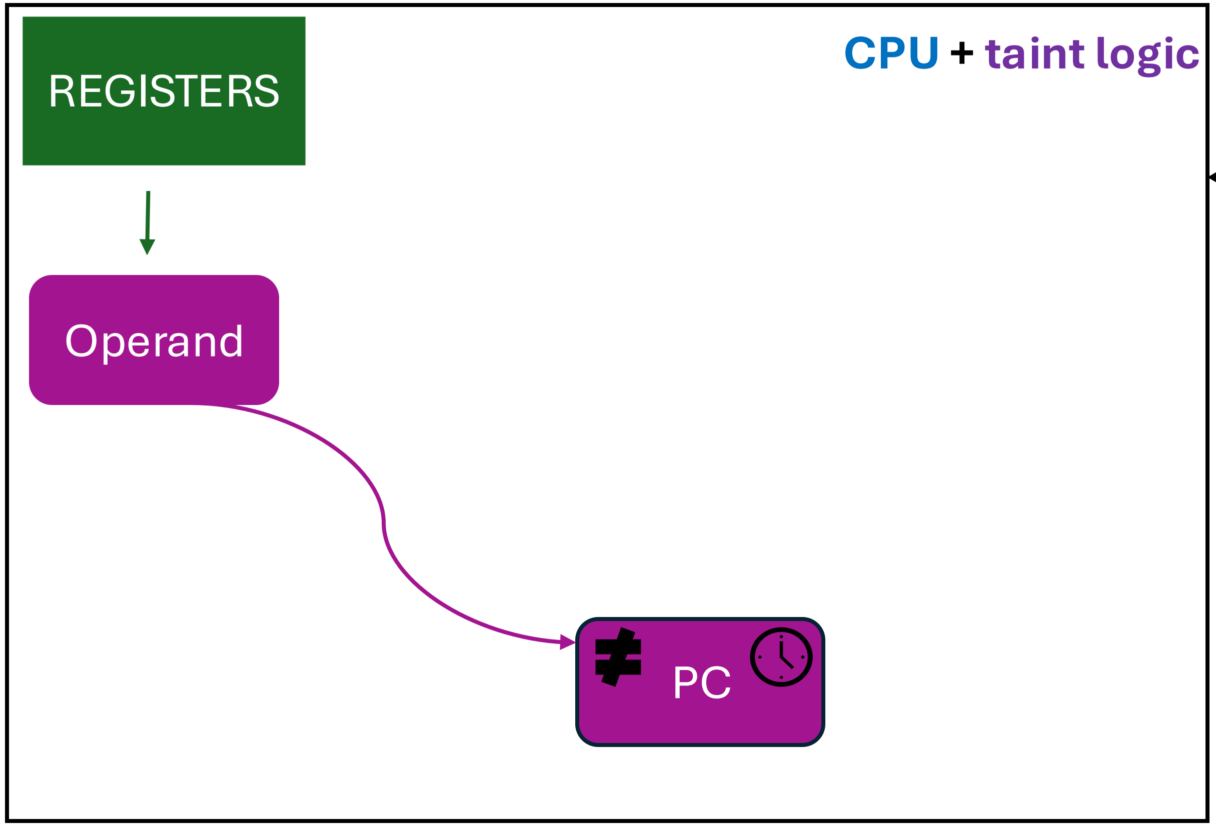
Identifying Insecure Instructions

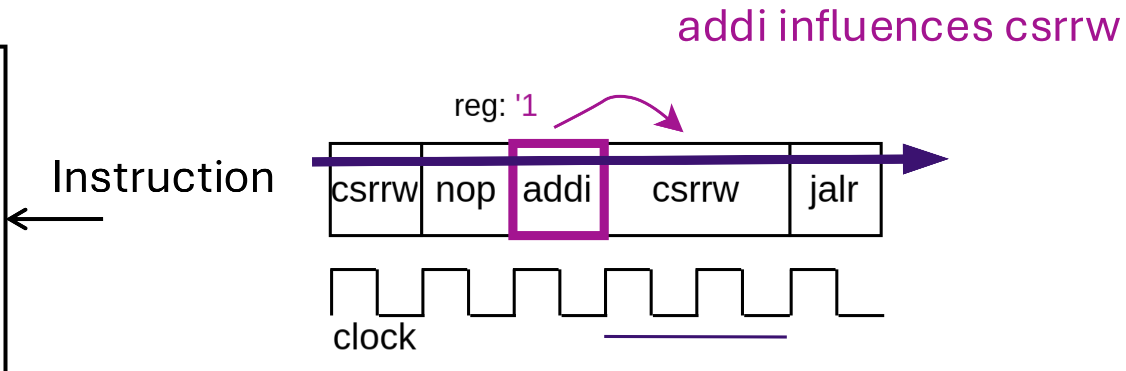




- **addi's** operand leaks, but violation is associated with csrrw
- Checking program counter taint after each instruction is **imprecise**

Identifying Insecure Instructions

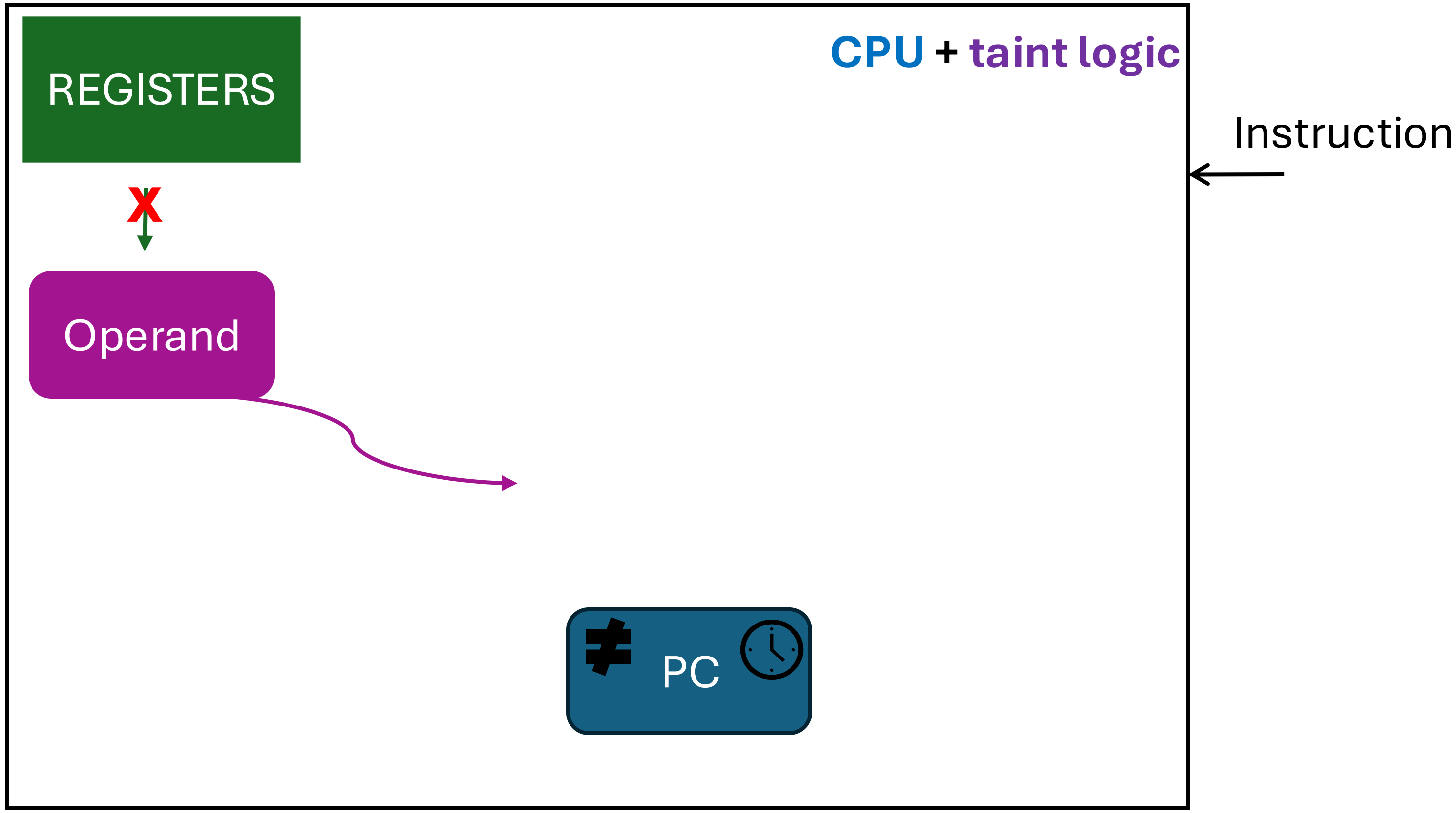




- **addi's** operand leaks, but violation is associated with csrrw
- Checking program counter taint after each instruction is **imprecise**
- A bug may be hidden by csrrw's specified information flow

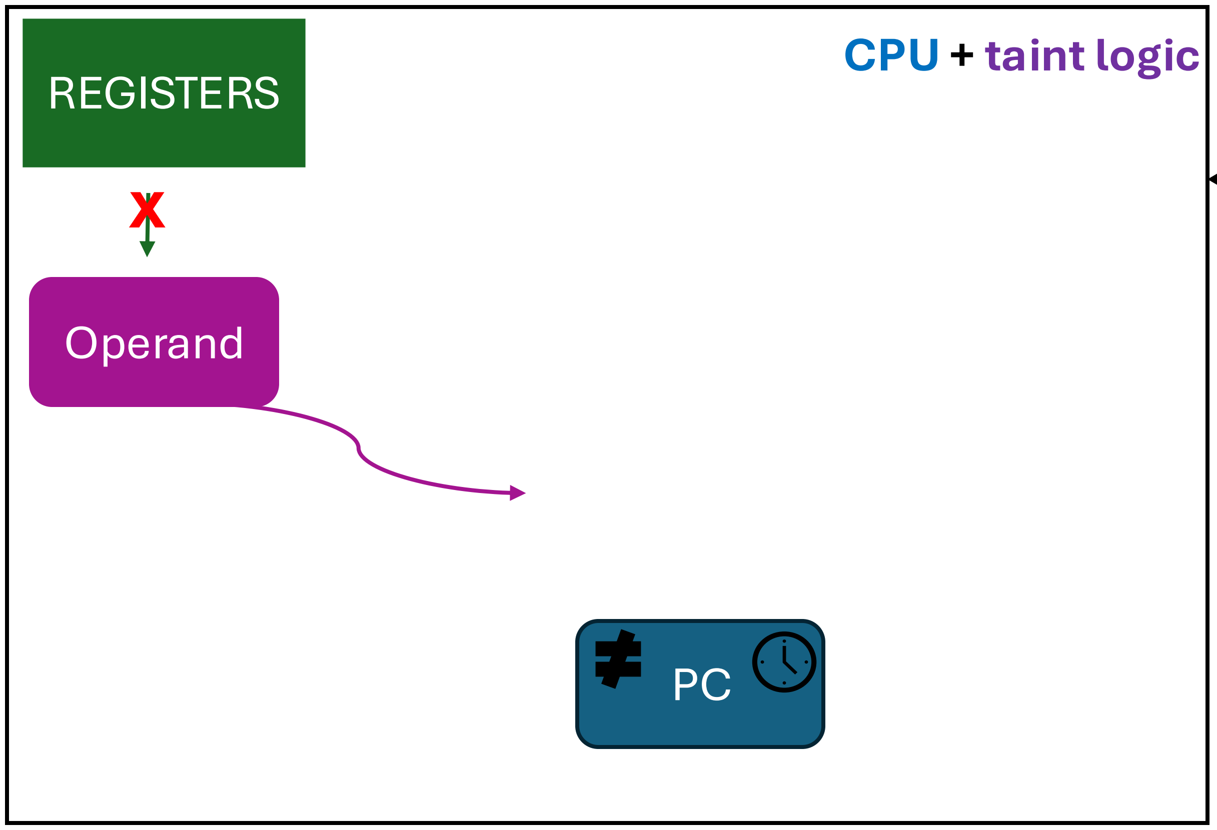
Precise Taint Injection

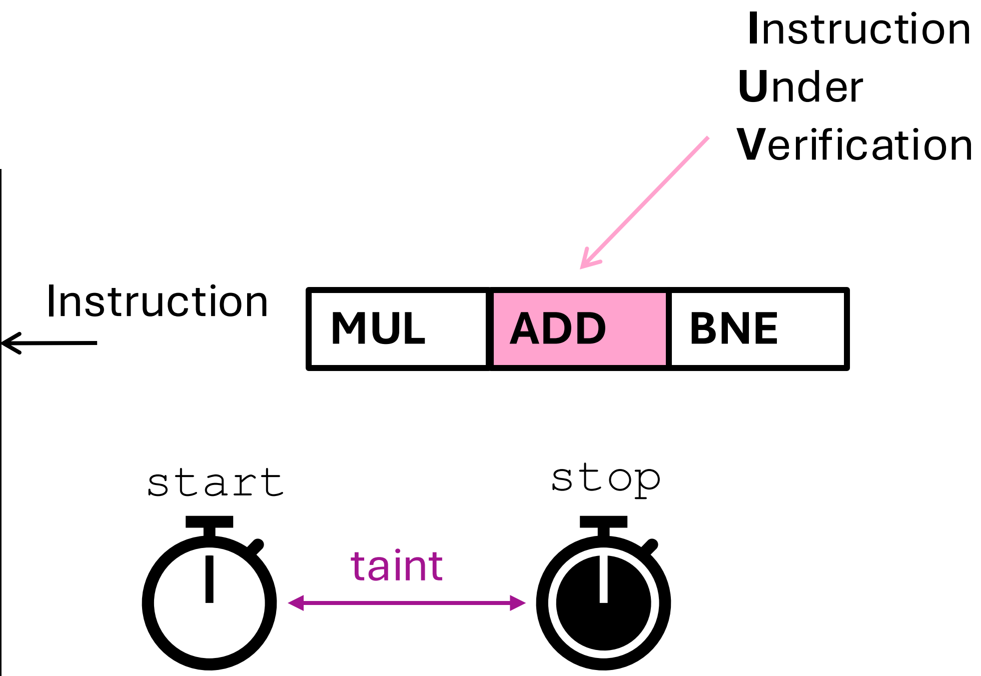
x = (taint) logic abstraction



Precise Taint Injection

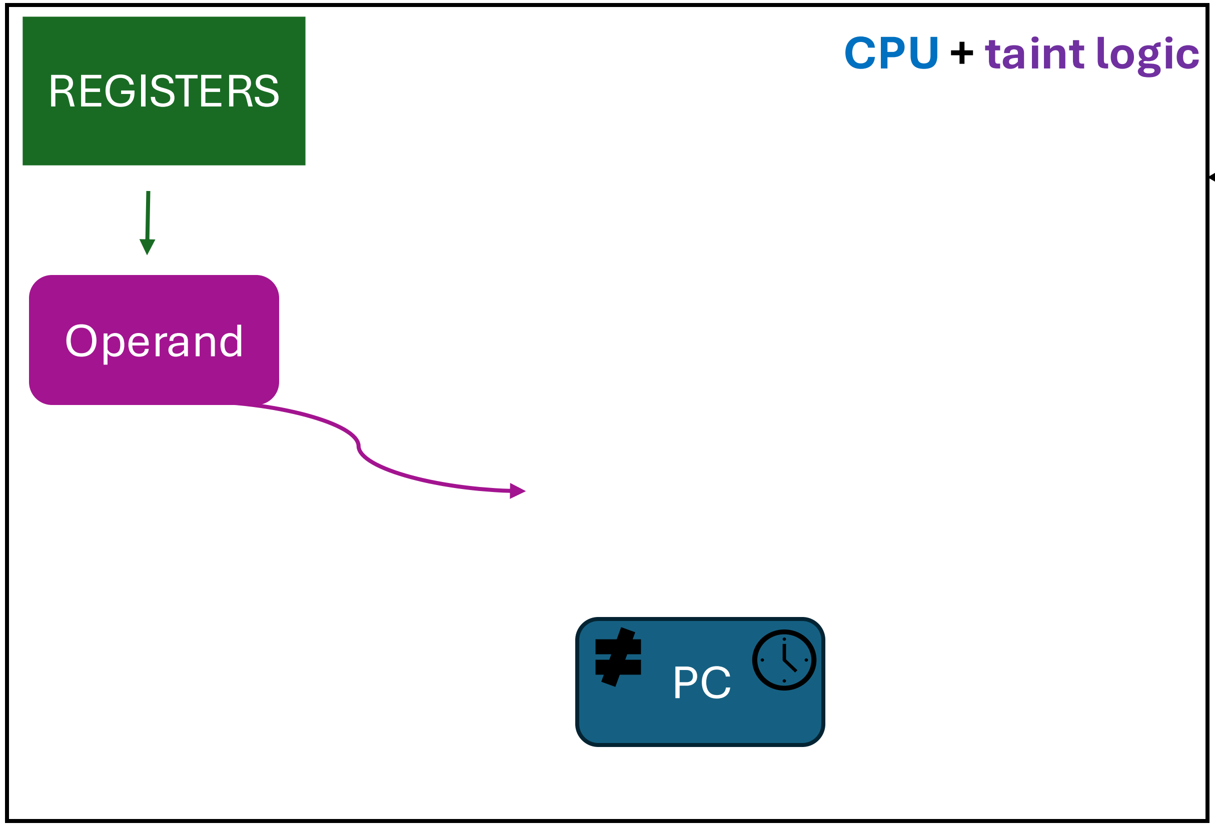
x = (taint) logic abstraction

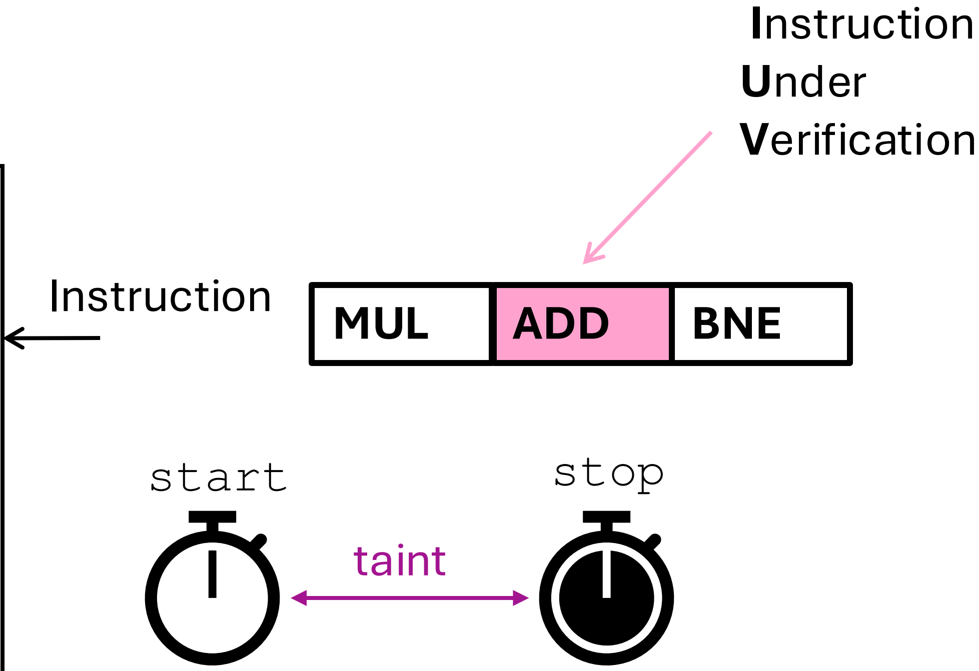




- Controlled **taint injection** per instruction
- Via SystemVerilog Assumptions

Precise Taint Injection

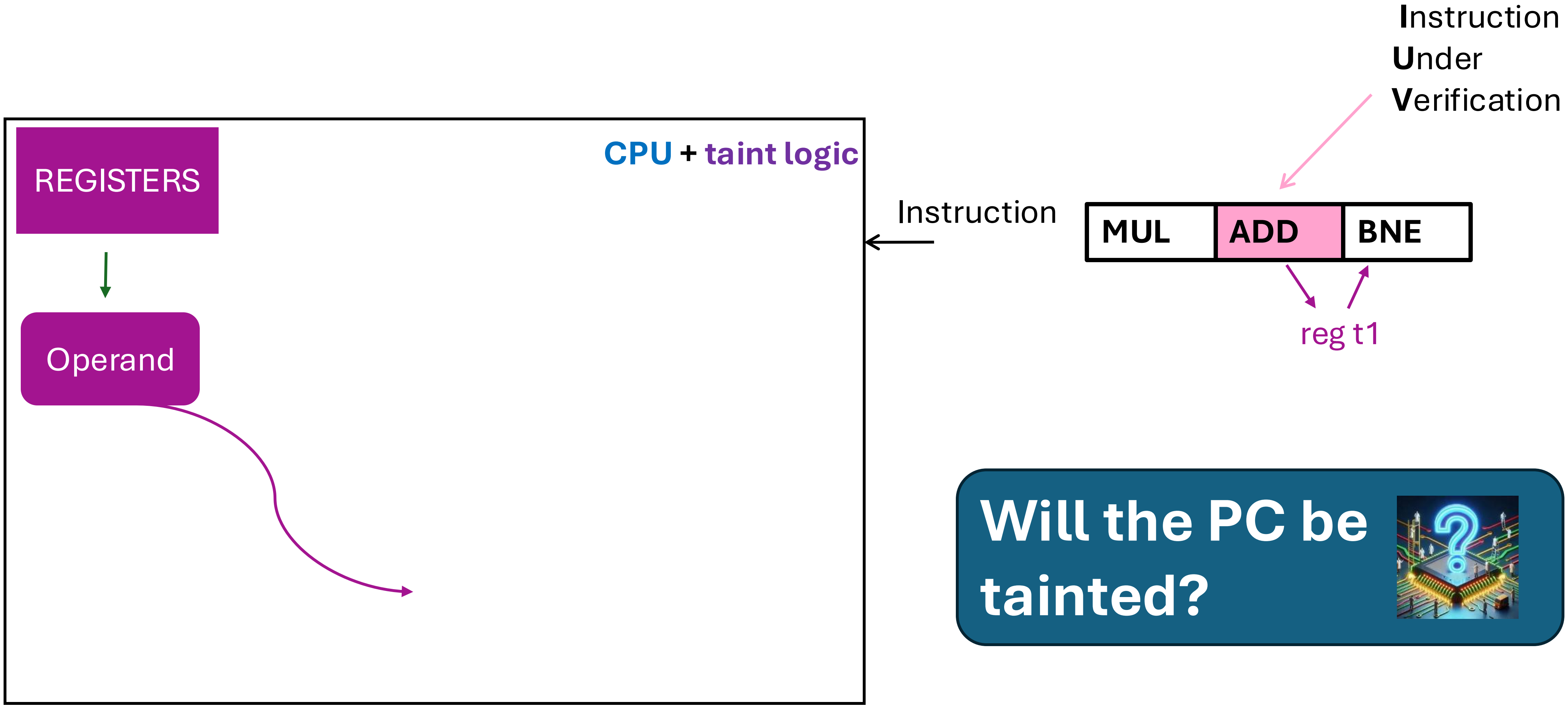




- Controlled **taint injection** per instruction
- Operand **reading conditions** automatically generated via static design analysis (custom Yosys[1] pass)

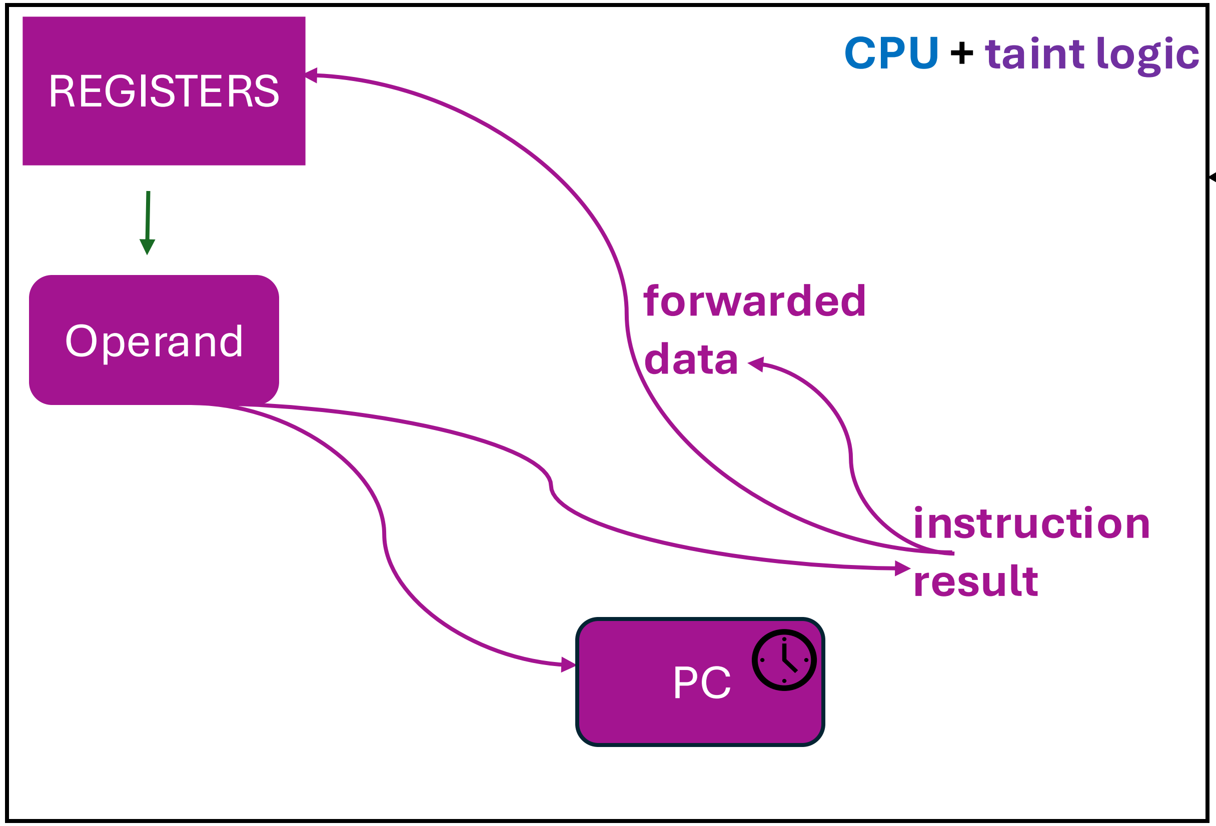
[1]https://github.com/YosysHQ/yosys

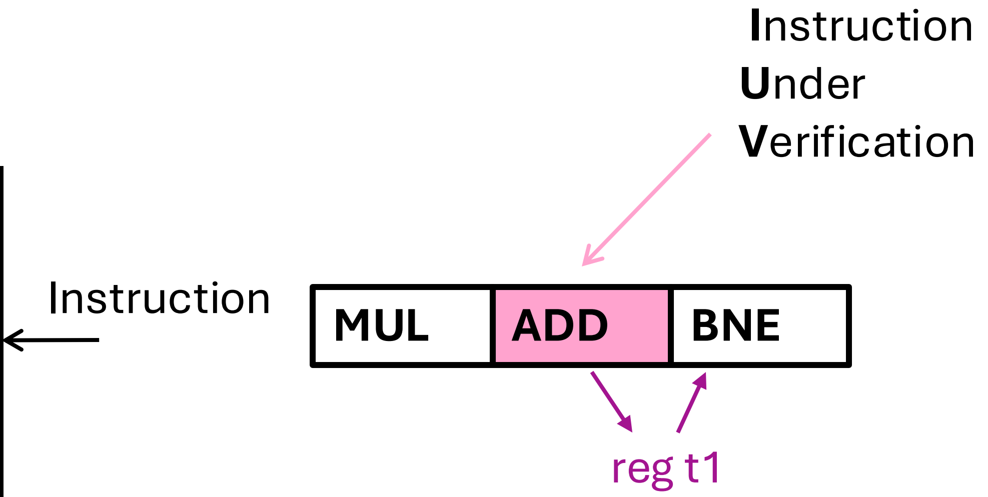
Declassification of Architectural Paths



Will the PC be tainted?

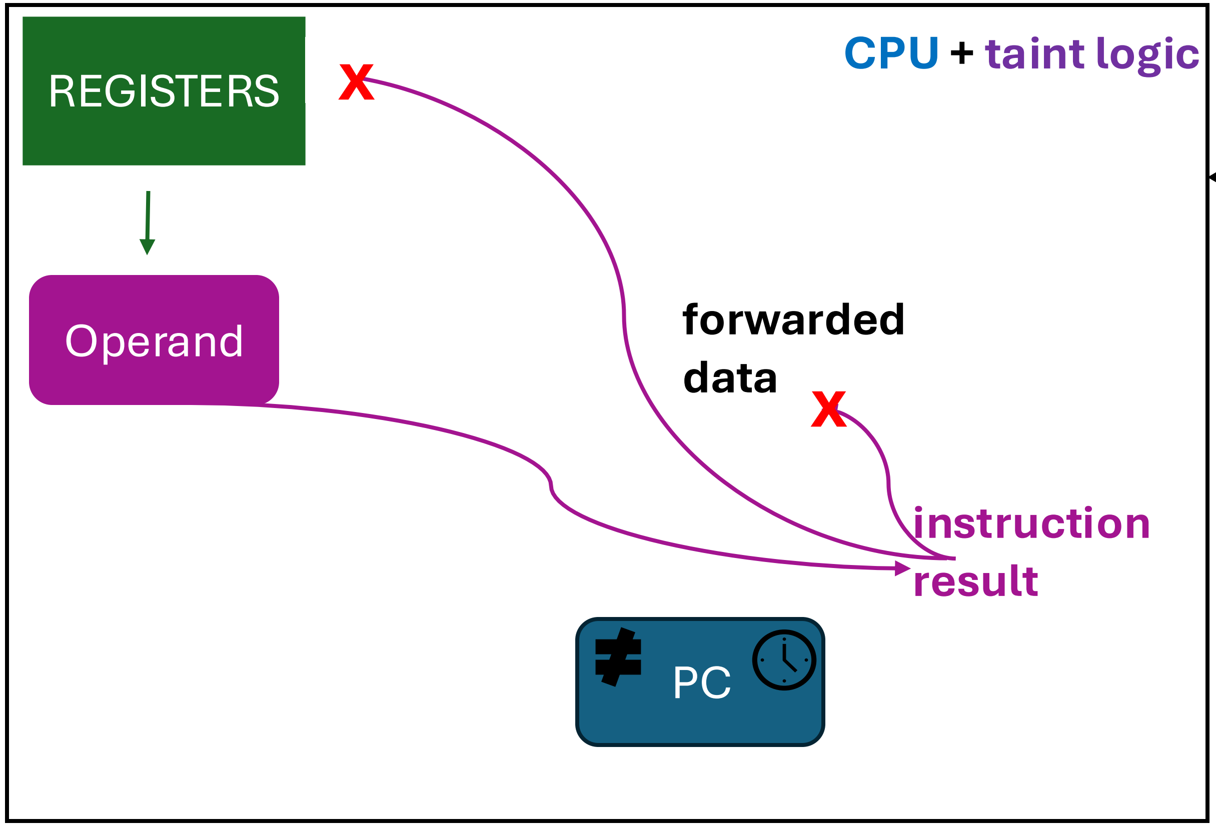
Declassification of Architectural Paths

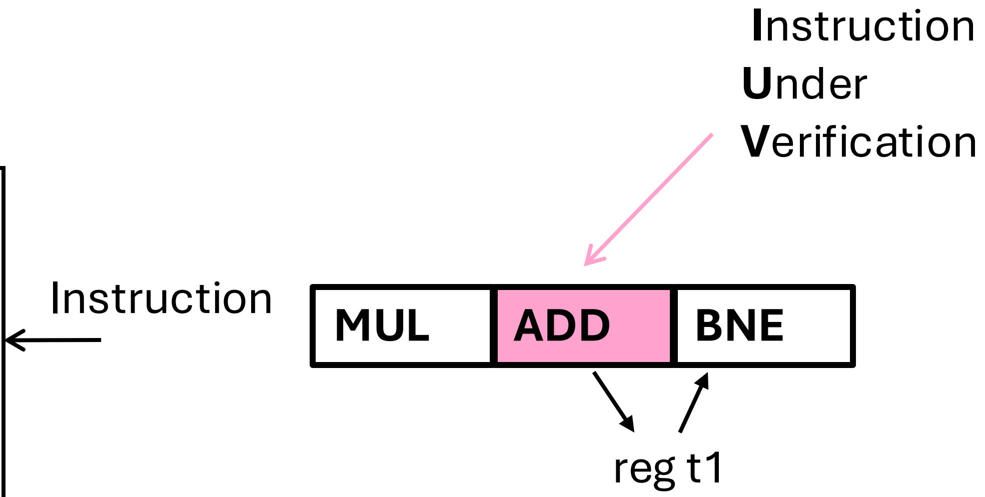




- Instruction result of 'add' forwarded to a 'branch' taints the PC.

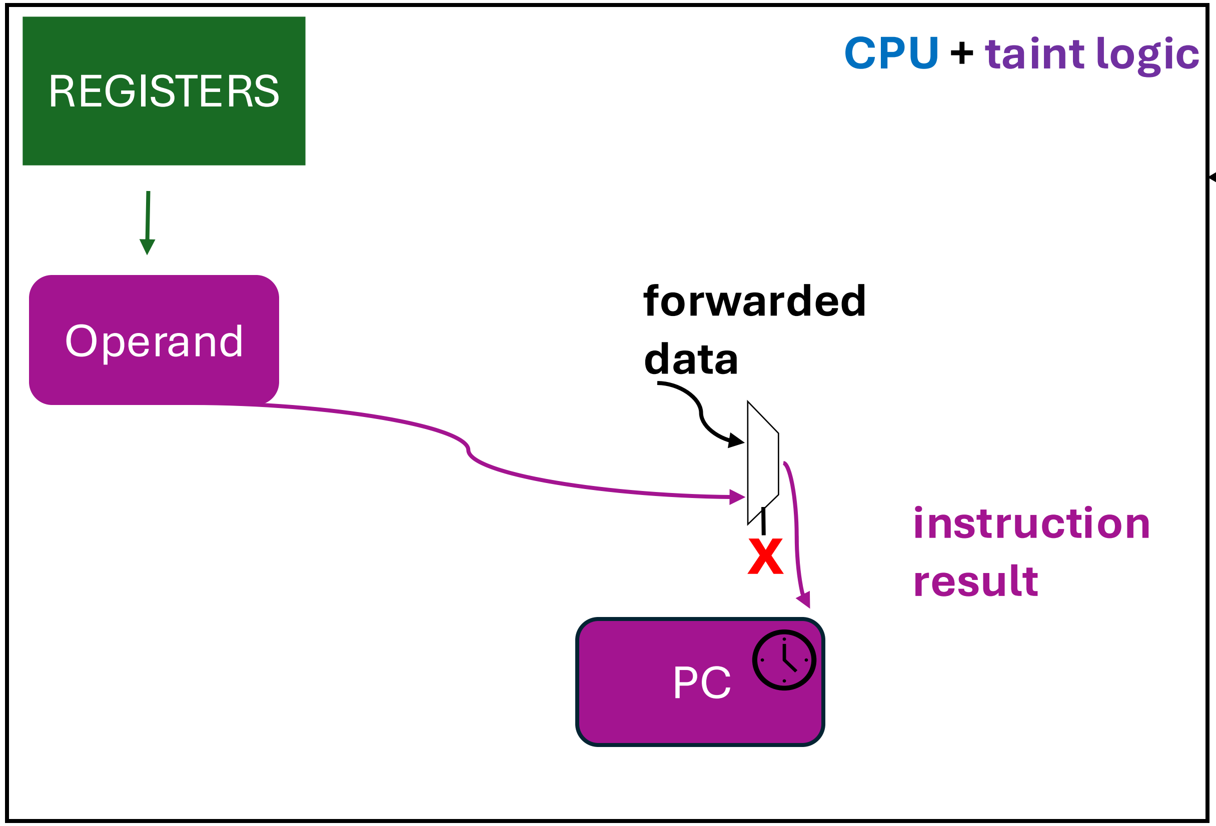
Declassification of Architectural Paths

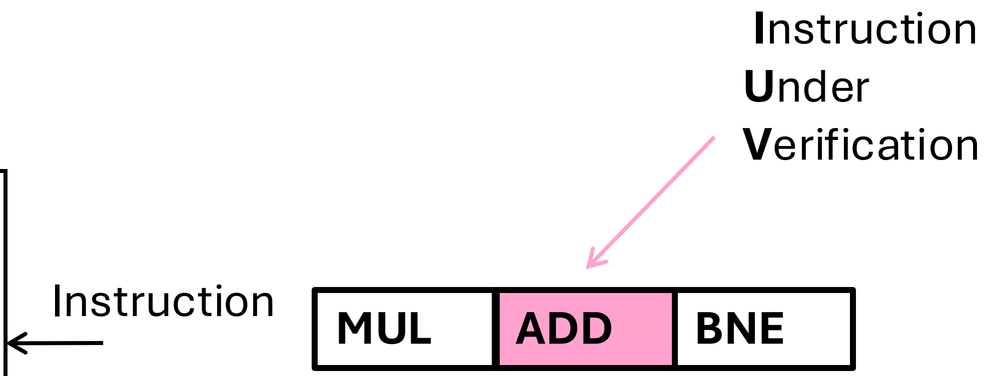




- Declassification:
  Block taint propagation via architectural (forwarding and register writeback) paths
- Forwarded data considered as instruction input
- Yosys pass checks that declassified paths do not reach the program counter

Tracking Forwarded Data

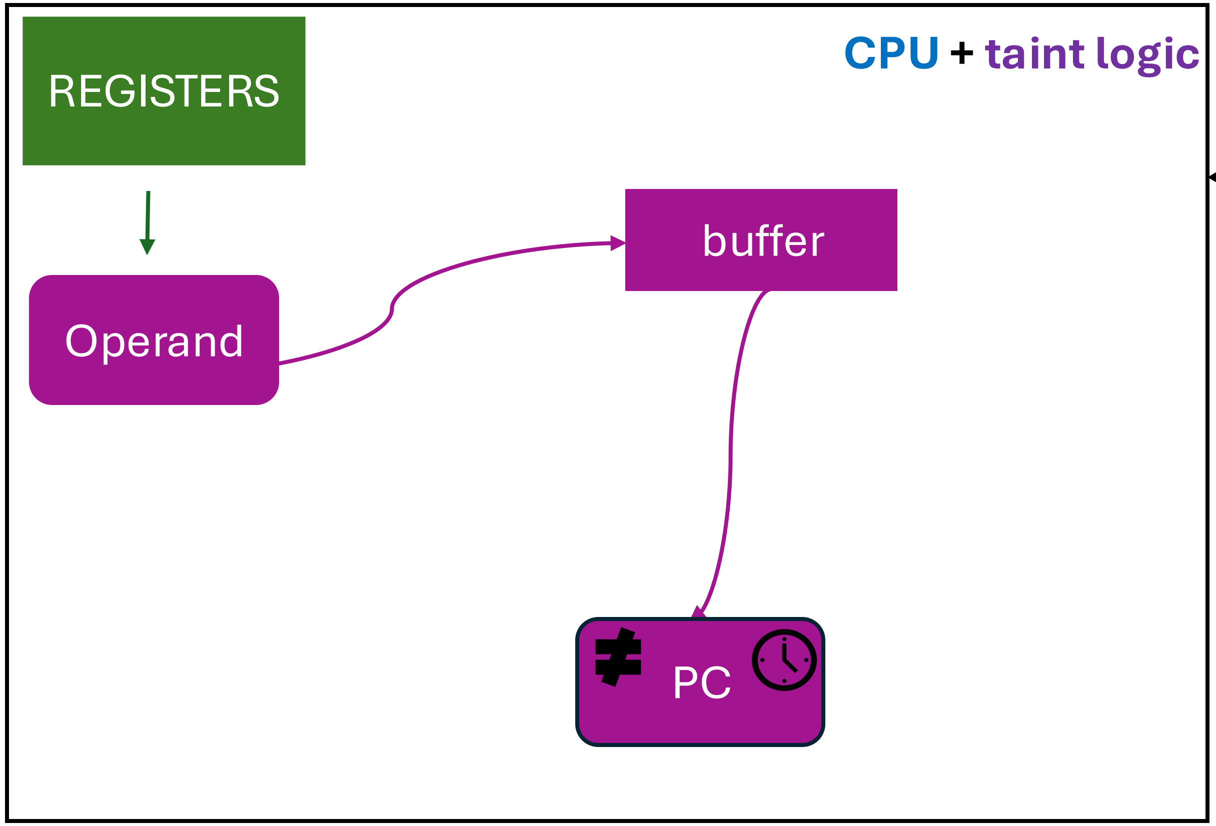


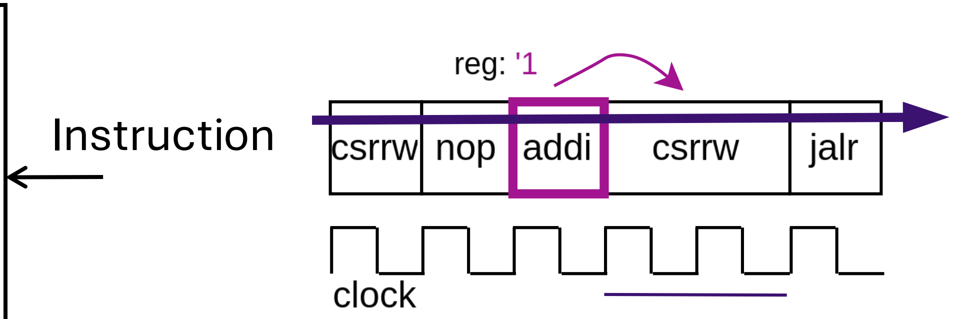


Forwarded data considered as instruction input:

- Allow operand taint to propagate if instruction reads from forwarded data

Declassification of Architectural Paths





No other microarchitectural flows are blocked

Verified RISC-V CPUs

Microcontroller-class, in-order CPUs

PicoRV32

Kronos

lowRISC Ibex

used in opentitan Root-of-Trust

Scarv

Zk scalar crypto extensions

State bits	3.2k	2.0k	2.5k	2.3k
Net bits	1.6k	1.4k	4.6k	6.7k

Verified RISC-V CPUs

Microcontroller-class, in-order CPUs

PicoRV32

Kronos

lowRISC Ibex

used in opentitan Root-of-Trust

Scarv

Zk scalar crypto extensions

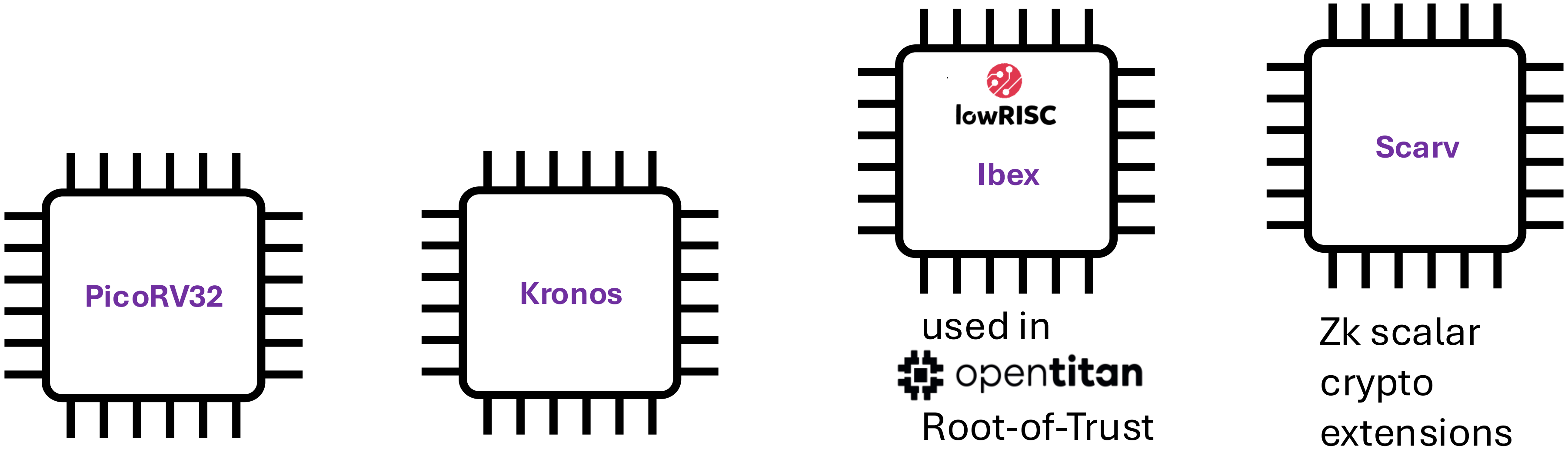
State bits	3.2k	2.0k	2.5k	2.3k
Net bits	1.6k	1.4k	4.6k	6.7k

Cell-	IFT / DFT	IFT / DFT	IFT / DFT	IFT / DFT
⌀ time to PROVE	17 h / 8m	16m / 30s	9h / 10m	14.5h / 50 m
⌀ time to FAIL	1h / 8m	37s / 15s	2h / 3m	11m / 34m

Model checker: Cadence Jasper Formal Property Verification App

Verified RISC-V CPUs

Microcontroller-class, in-order CPUs



used in opentitan Root-of-Trust

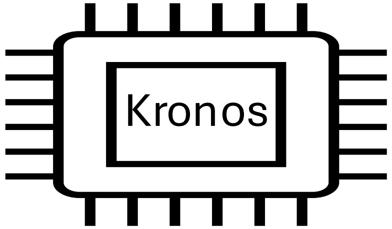
Zk scalar crypto extensions

State bits	3.2k	2.0k	2.5k	2.3k
Net bits	1.6k	1.4k	4.6k	6.7k

Cell-	IFT / DFT	IFT / DFT	IFT / DFT	IFT / DFT
⌀ time to PROVE	17 h / 8m	16m / 30s	9h / 10m	14.5h / 50 m
⌀ time to FAIL	1h / 8m	37s / 15s	2h / 3m	11m / 34m

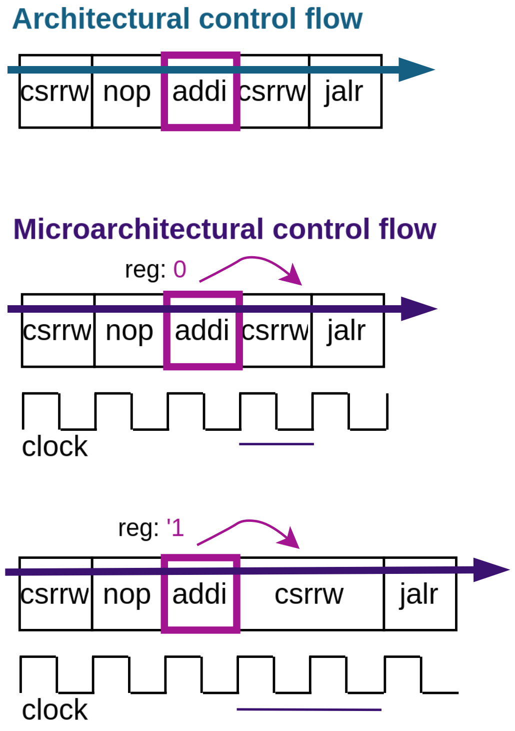
PROVEN instructions	38	25	27	38 + all crypto instr.
VULNERABLE instructions	3 (documented)	8	14	3 (known)

New Discovered Security Vulnerabilities



Constant time violation:

CVE-2023-51974



Two control-flow hijacks:

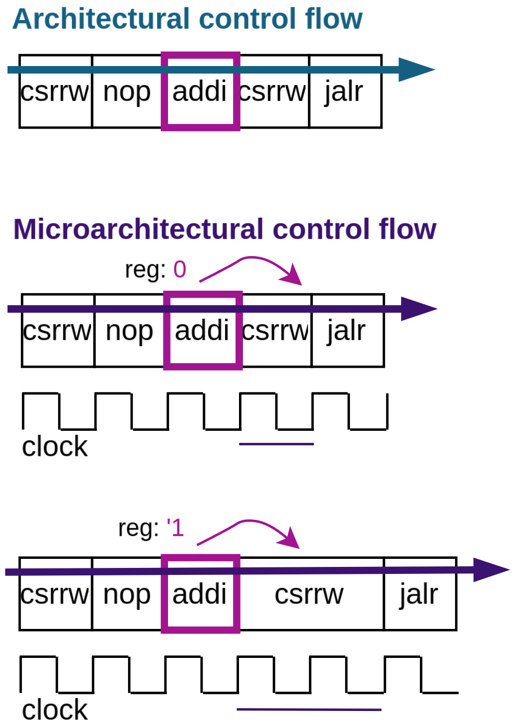
CVE-2023-51973

CVE-2024-44927

New Discovered Security Vulnerabilities

Constant time violation:

CVE-2023-51974



Architectural control flow

csrrw nop addi csrrw jalr

Microarchitectural control flow

reg: 0

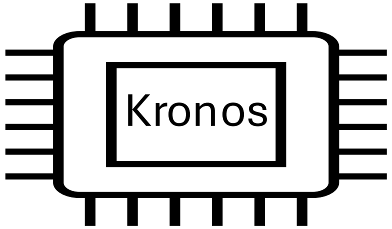
csrrw nop addi csrrw jalr

clock

reg: '1

csrrw nop addi csrrw jalr

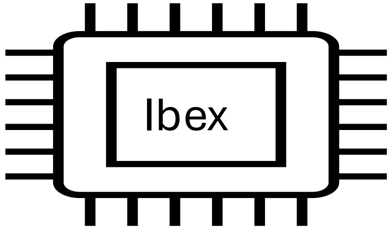
clock



Two control-flow hijacks:

CVE-2023-51973

CVE-2024-44927



Constant time violation + data leakage:

CVE-2024-28365

Control-flow hijack

Conclusion

- Introduced and formalized a generalized CPU security property

µCFI - Microarchitectural Control-flow Integrity

Conclusion

- Introduced and formalized a generalized CPU security property

µCFI - Microarchitectural Control-flow Integrity

- Automated verification method & implementation
- 4 open-source RISC-V CPUs verified
- Discovered 5 new vulnerabilities - 4 CVEs

Conclusion

- Introduced and formalized a generalized CPU security property

µCFI - Microarchitectural Control-flow Integrity

- Automated verification method & implementation
- 4 open-source RISC-V CPUs verified
- Discovered 5 new vulnerabilities - 4 CVEs

Video:
https://www.youtube.com/watch?v=Kxp-5kNMt40&t

Thank you! Questions?

Information:

https://comsec.ethz.ch/

Code:

comsec-group/mucfi

Contact:

https://www.linkedin.com/in/katharina-ceesay-seitz-ba521087/

@K_Ceesay-Seitz, @FlavienSolt

kceesay@ethz.ch, flavien.solt@eecs.berkeley.edu

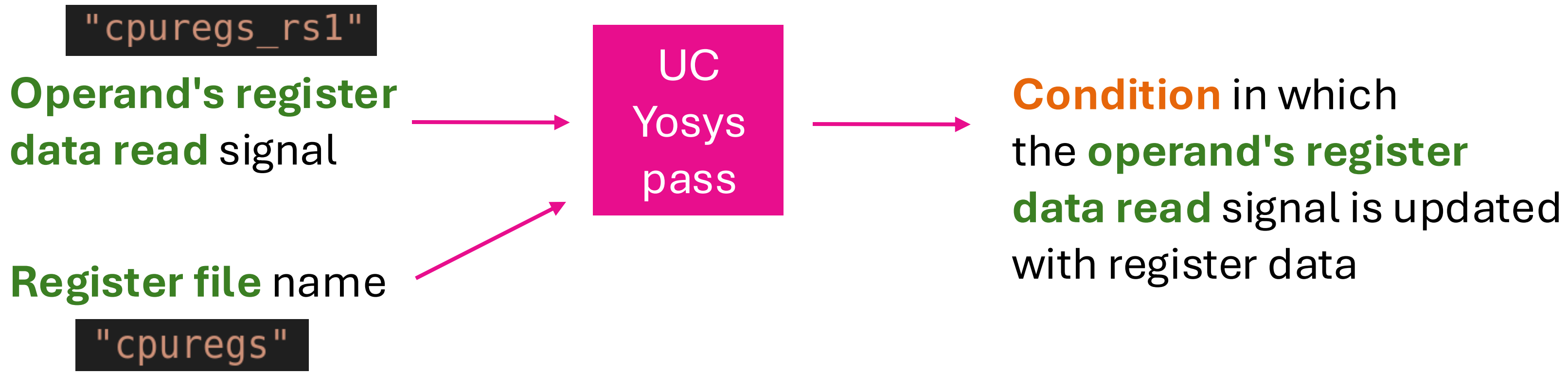
BACKUP

Taint Start Condition

Update Condition Yosys Pass

Read-from Condition = the condition in which a signal is updated with a chosen signal's value.

Taint start



"cpuregs_rs1"

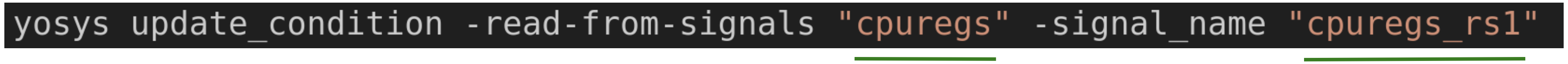
Operand's register data read signal

Register file name

"cpuregs"

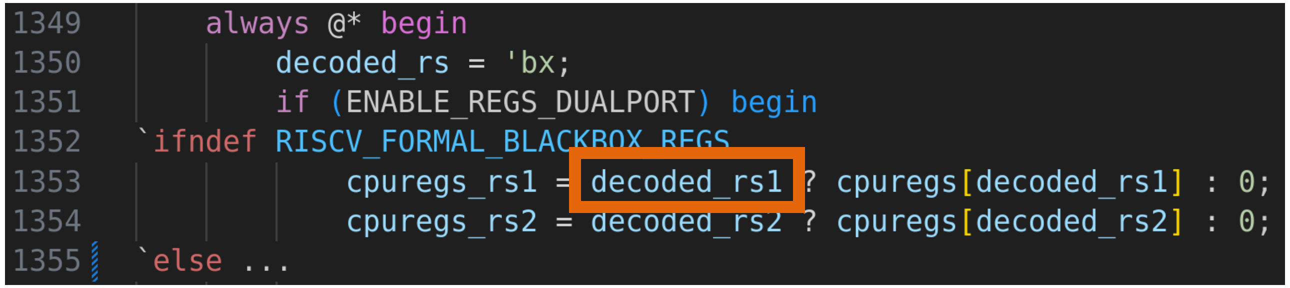
UC Yosys pass

Condition in which the **operand's register data read** signal is updated with register data



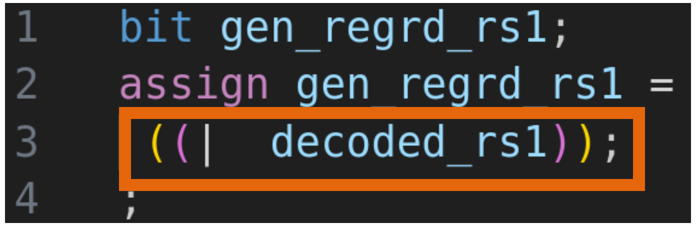
```
yosys update_condition -read-from-signals "cpuregs" -signal_name "cpuregs_rs1"
```

CPU code (PicoRV32):



```
    always @* begin
        decoded_rs = 'bx;
        if (ENABLE_REGS_DUALPORT) begin
`ifndef RISCV_FORMAL_BLACKBOX_REGS
            cpuregs_rs1 = decoded_rs1 ? cpuregs[decoded_rs1] : 0;
            cpuregs_rs2 = decoded_rs2 ? cpuregs[decoded_rs2] : 0;
`else ...
```

Generated Read-from Condition:



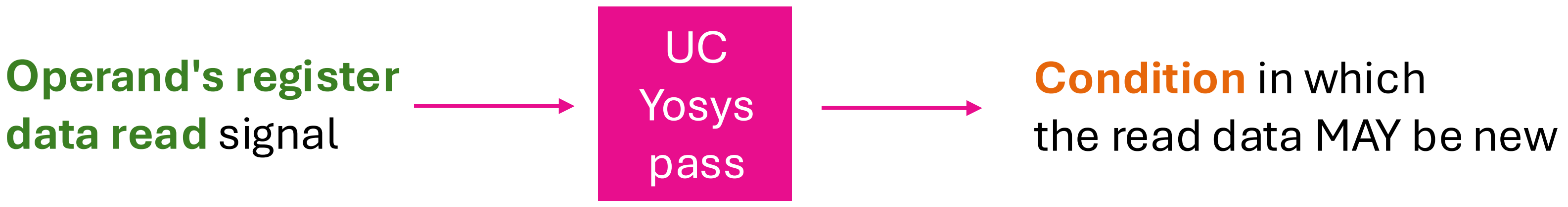
```
bit gen_regrd_rs1;
assign gen_regrd_rs1 =
 ((|  decoded_rs1));
;
```

Taint Stop Condition

Update Condition Yosys Pass

Update Condition (UC) = the condition in which a signal is updated with another value than its own previous value.

Taint stop



Operand's register data read signal

UC Yosys pass

Condition in which the read data MAY be new

For example:

- **enable condition of a flip flop**
- **'1' (True) for continuous assignments**

Precise Taint Injection Conditions

Sample Instruction Word (IW) in formal setup

start

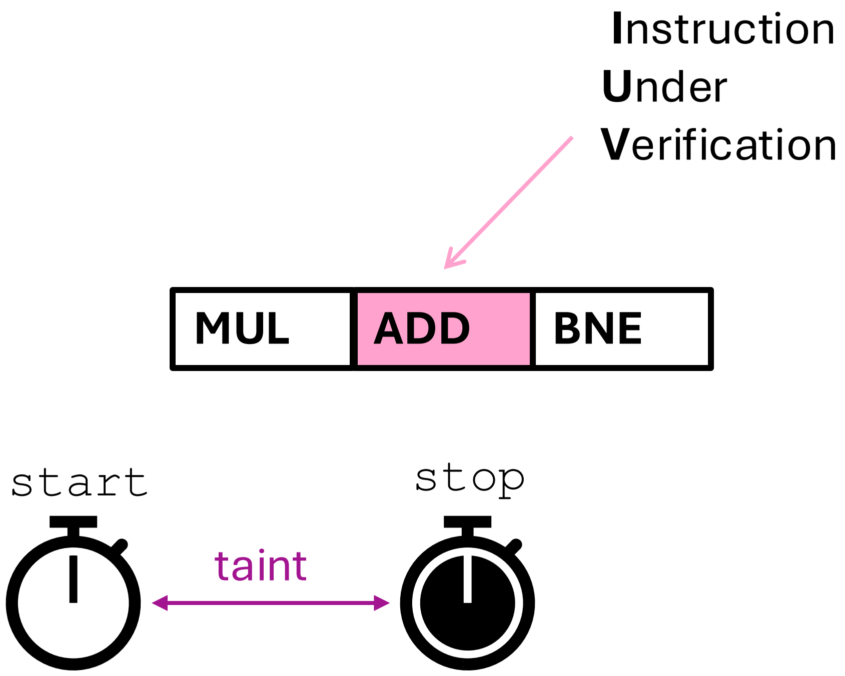
IW == IUV and taint start condition

stop

Taint stop condition

Potentially taint multiple times per instruction

Simple & precise counter examples

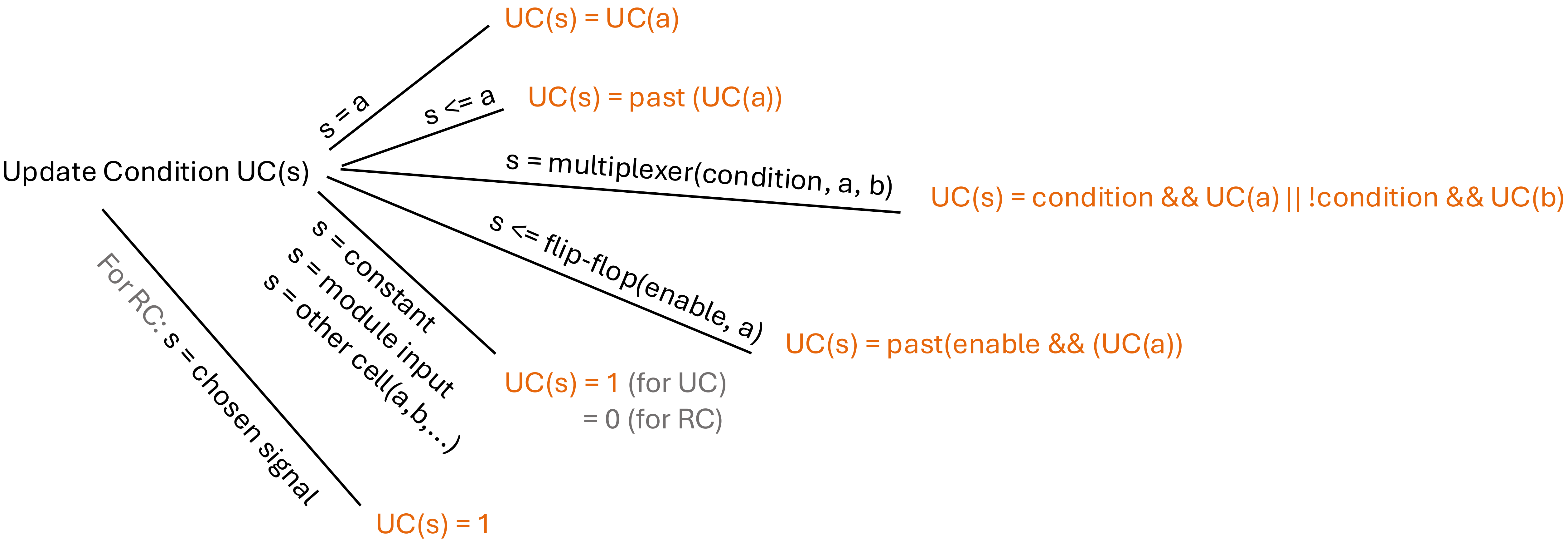


- Controlled **taint injection** per instruction

Update Condition (UC) / Read-from Condition (RC) Yosys Pass

s ... signal

a,b ... other internal signals
'past' = custom attribute



Update Condition UC(s)

s = a
UC(s) = UC(a)

s <= a
UC(s) = past (UC(a))

s = multiplexer(condition, a, b)
UC(s) = condition && UC(a) || !condition && UC(b)

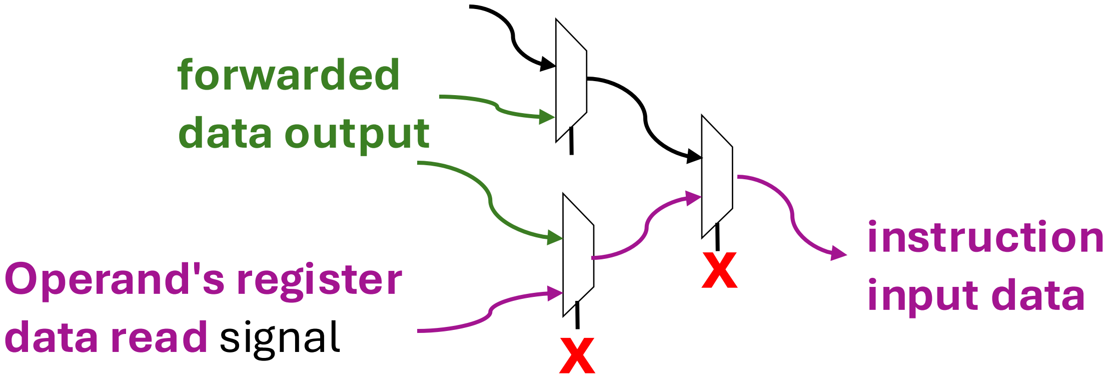
s <= flip-flop(enable, a)
UC(s) = past(enable && (UC(a))

s = constant
s = module input
s = other cell(a,b,...)
UC(s) = 1 (for UC)
= 0 (for RC)

For RC: s = chosen signal
UC(s) = 1

Find Forwarding Multiplexer Yosys Pass

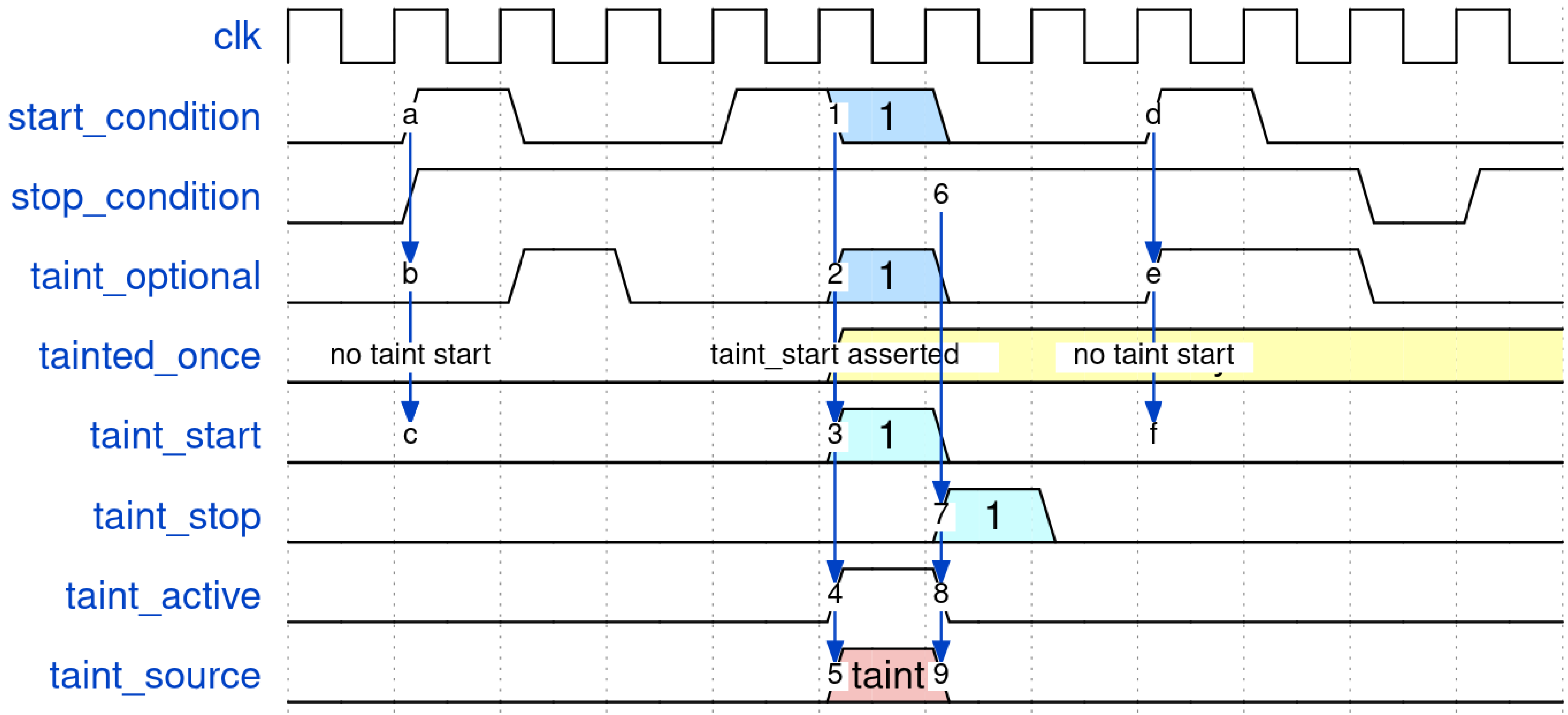
- Automatically identifies forwarding multiplexers
- Checks <u>declassification precondition</u>: all outgoing paths of declassified signals reach another declassified signal or data source without passing PC



1. Traverse outgoing paths of forwarded data output and check declassification precondition
2. If a mux uses forwarded data output, back-traverse multiplexers' other input's driving logic.
3. Is it directly assigned with operand's register data read signal?
   - No: continue at mux output
   - Yes: Forwarding mux found X --> return mux select signal

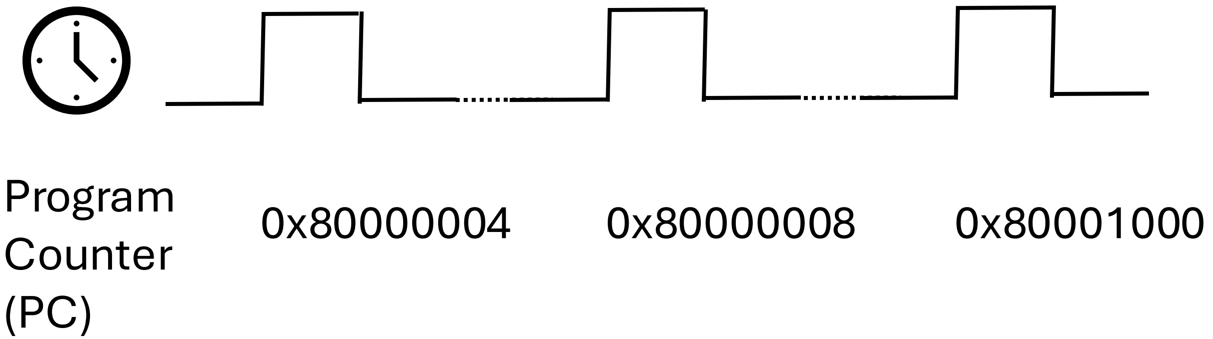
mux = multiplexer

Taint Injection Assumptions



Introducing
µCFI - Microarchitectural Control-flow Integrity

Microarchitectural control flow (µCF)



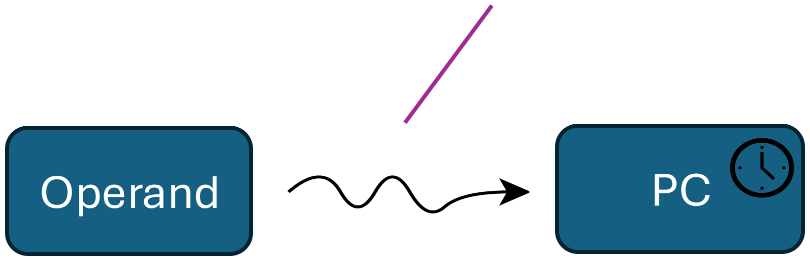
Program Counter (PC)

0x80000004

0x80000008

0x80001000

µCFI only allows explicitly ISA specified data dependencies of the µCF



Operand

PC

ISA = Instruction Set Architecture