Journée thématique sur les Attaques par Injection de Fautes
September 24th 2020

Bridging the Gap between RTL and Software Fault Injection: a Methodology for Accurate Fault Modeling

Johan Laurent[1], Vincent Beroulle[1], Christophe Deleuze[1], Florian Pebay-Peyroula[2]

[1] Univ. Grenoble Alpes, Grenoble INP, LCIS
26000 Valence, France
firstname.lastname@lcis.grenoble-inp.fr

[2] Univ. Grenoble Alpes, CEA, LETI
38000 Grenoble, France
firstname.lastname@cea.fr

This work was funded thanks to the French national program 'programme d'Investissements d'Avenir, IRT Nanoelec' ANR-10-AIRT-05

Summary

I. Introduction

- **Hardware fault injection can be countered at the hardware level, but also at the software level**

- **Software analyses are based on software fault models (defined by the Joint Interpretation Library for example [1])**
  - Instruction skip [2]
  - Control-flow corruption (test inversion, …) [3][4]
  - Register/memory corruptions [5][6]

- **Problem: There are fault effects that are not modelled in typical software fault models [7]**

- **These effects arise from the complexity of processor microarchitecture**

I. Introduction

- **Software fault modeling usually consider the processor as a black-box [8, 9, 10] and perform physical injections**

  → Realistic, but can only infer what happens in the processor from visible results

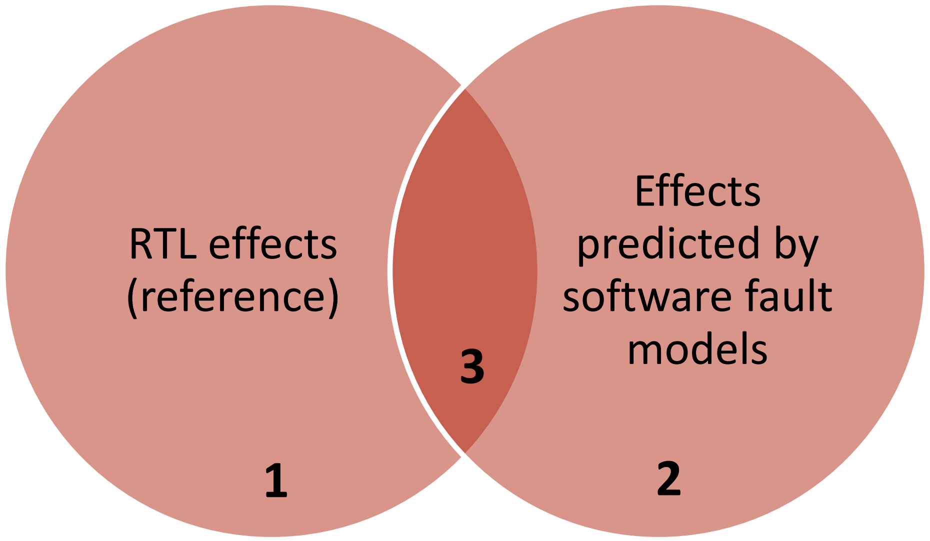
- **We consider the RTL description of the processor available and we perform RTL injections in simulation**

  → Need to rely on a hardware fault model, but better understanding of the faulty behaviors, earlier in the design flow

- **Effects obtained in RTL simulation in a LowRISC processor [11]:**
  - Replace an argument by the last computed value
  - Make an instruction "transient"
  - Commit a speculated instruction
  - …

I. Introduction

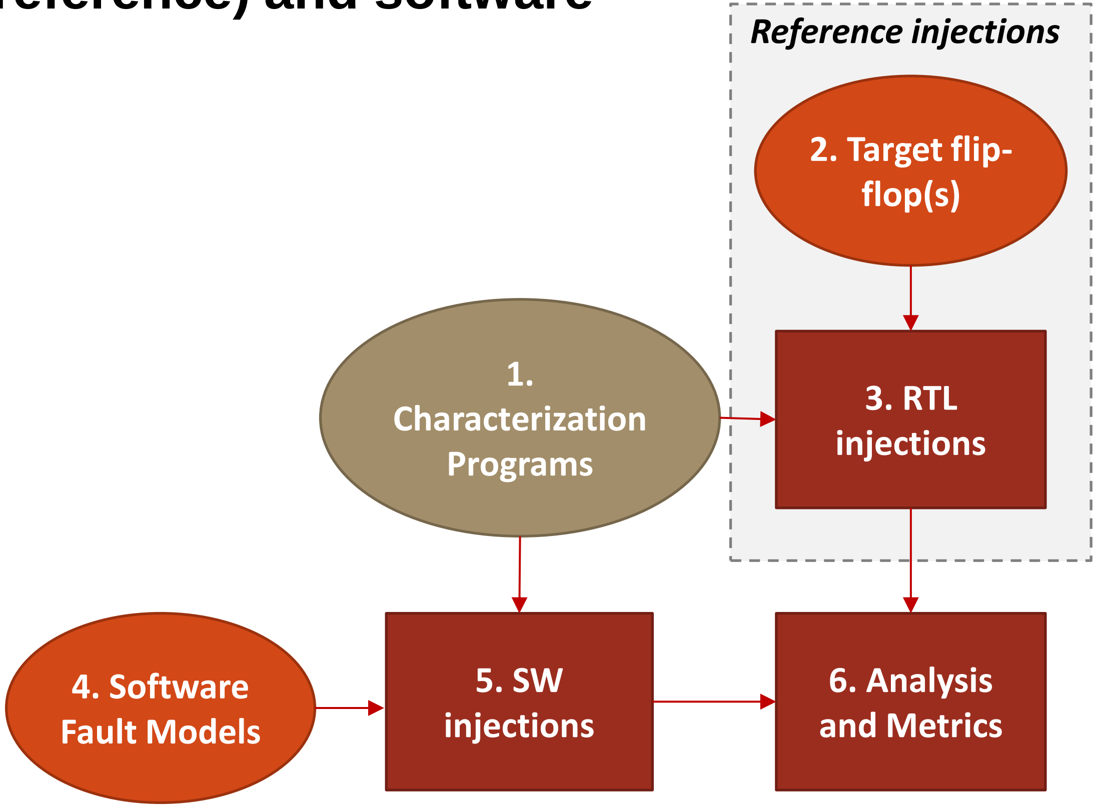
- **Our goal is to precisely model RTL faults at the software level, and evaluate these models.**



- **We approach security in a global way: hardware & software → We want to define cross-layer analyses.**

II. Fault injection approach
1 - Overview

- Principle: Precisely comparing results of RTL injections (reference) and software injections.

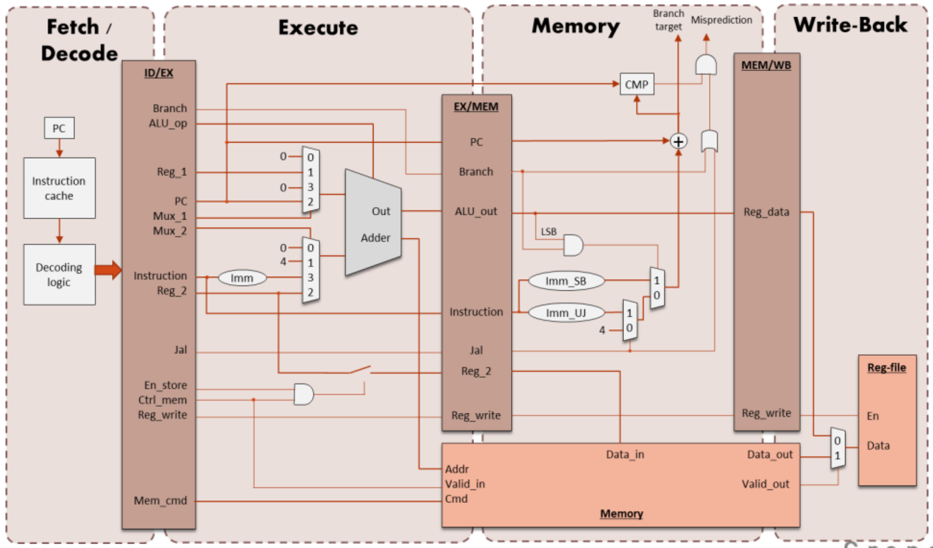


II. Fault injection approach
1 - Overview

- **Injections and observations are performed under the same circumstances for both abstraction levels**

- **Some structures are visible from both abstraction levels (register-file, memory) ; others, hidden registers, are only seen at the hardware level.**

- **Observation:**
  - Register-file
  - Memory



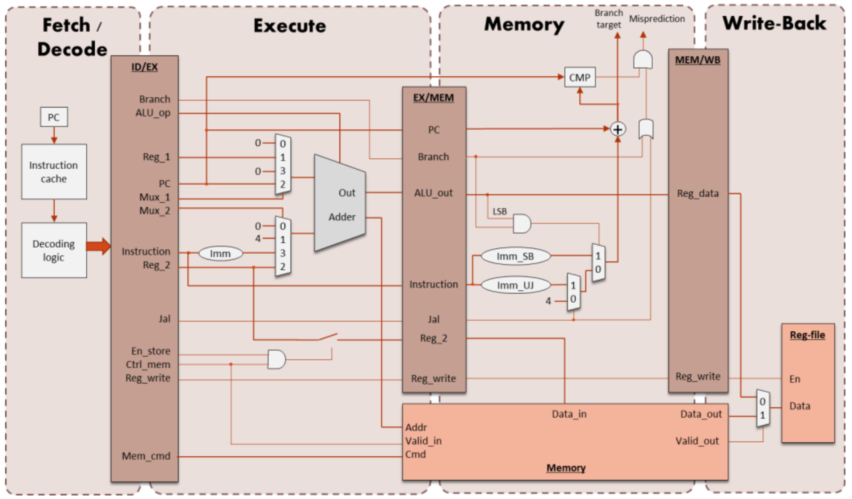
II. Fault injection approach
2 – Characterization programs

- **Fault effects can vary depending on the precise instruction sequence → Small assembly contexts that represent various situations:**

```
Prologue_instruction_1
Prologue_instruction_2
Target_instruction          // Injection + Observation immediate effects
Epilogue_instruction_1
Epilogue_instruction_2
Epilogue_instruction_3      // Observation propagation effects
```

II. Fault injection approach
3 – RTL fault injection

- Bit-flips injected through simulator commands (mostly single-bit, but also some multiple-bit injections)
- RISC-V LowRisc v0.4 processor (5-stage pipeline)
- Pipelined execution → exact injection instant depends on which flip-flop is faulted

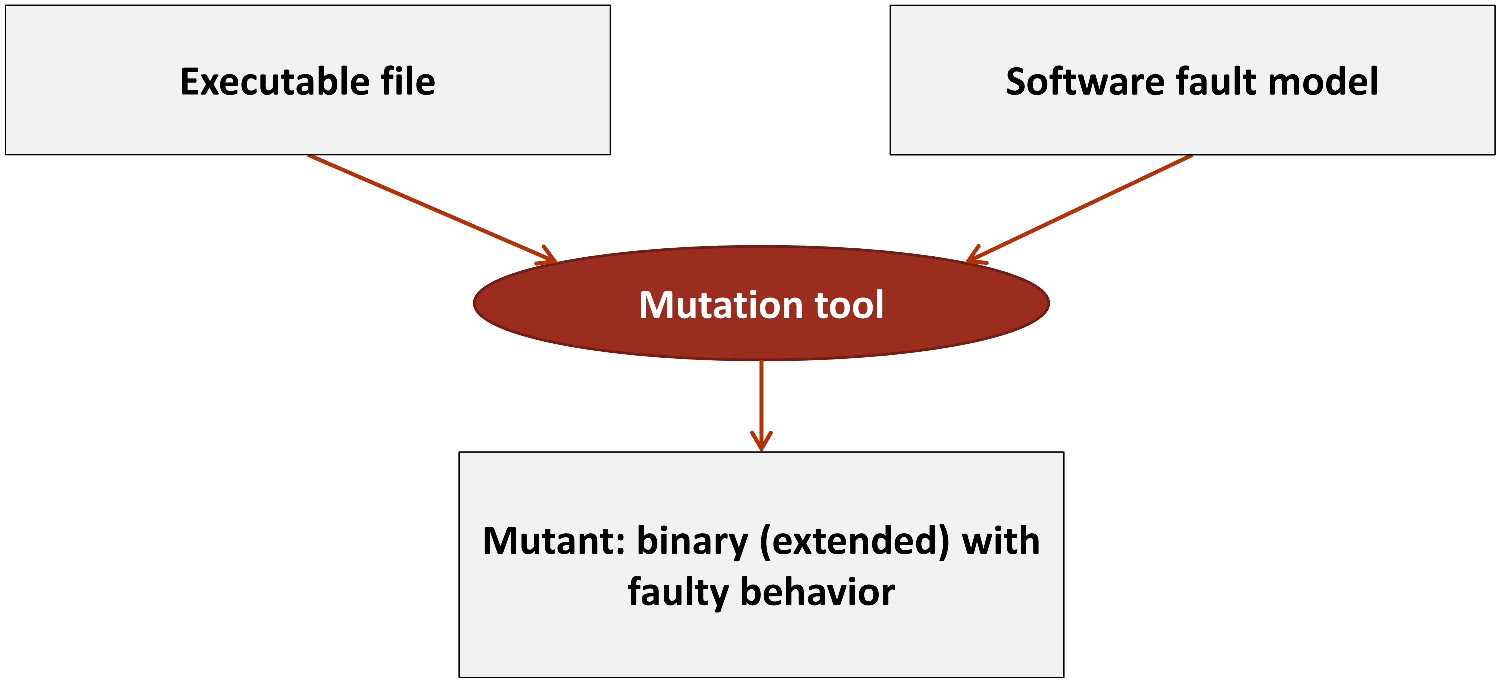


II. Fault injection approach
4 – Software fault injection

- **Software fault injection through a program mutation tool.**

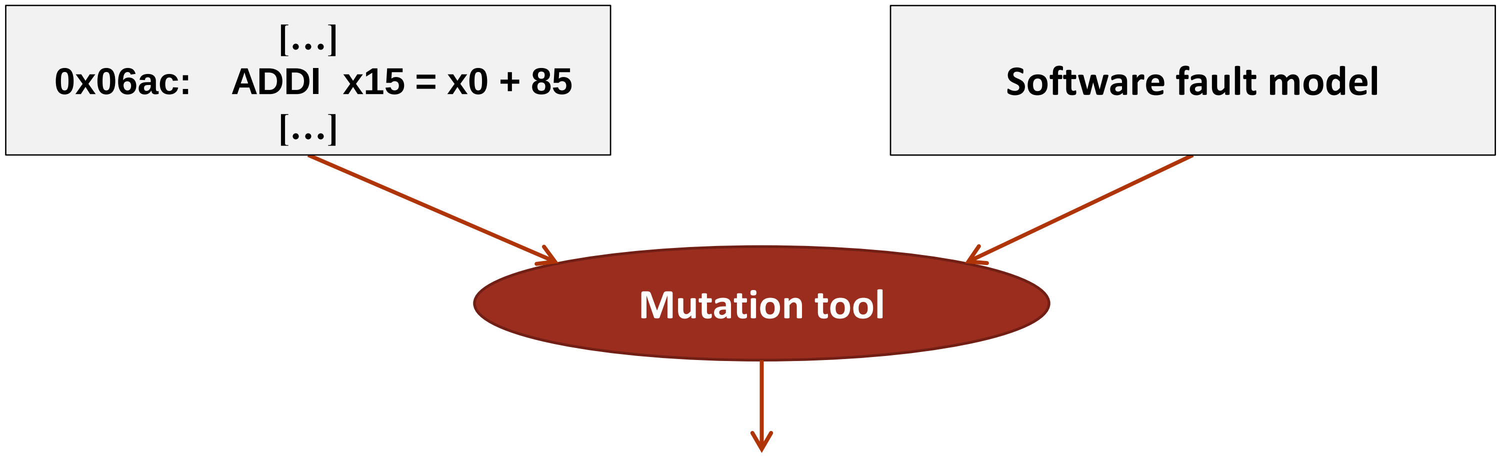
- **Constraints:**
  - Precise modeling → takes the binary as input to execute the exact same thing in hardware and software and to eliminate compiler influence
  - Effects are varied → need flexibility
  - Need to represent information not available at the binary level (like the value of some hidden registers) → representation of the execution at a higher level (in our case, we chose C)

II. Fault injection approach
4 – Software fault injection



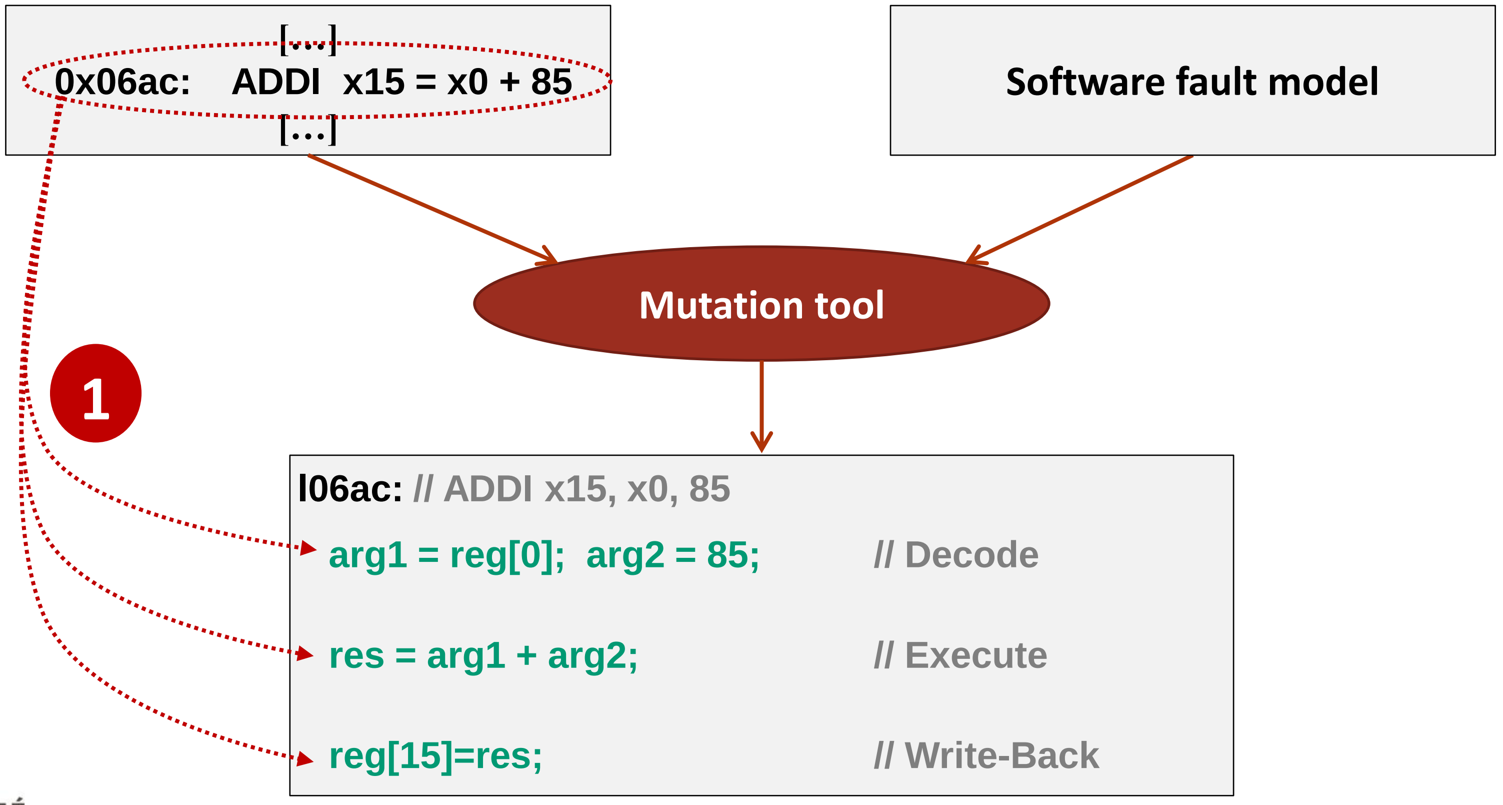
II. Fault injection approach

4 – Software fault injection



II. Fault injection approach

4 – Software fault injection

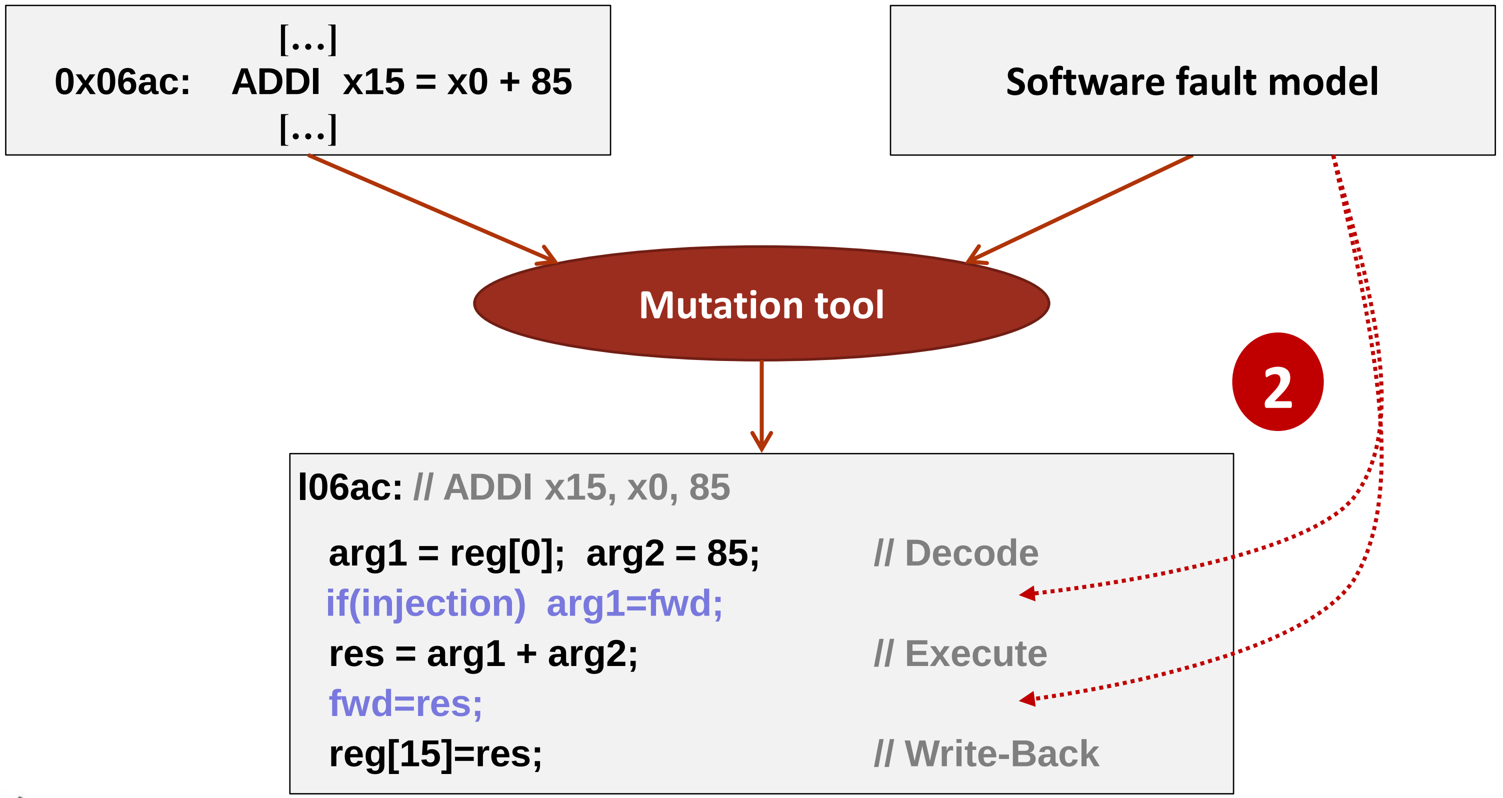


```
I06ac: // ADDI x15, x0, 85
    arg1 = reg[0];  arg2 = 85;        // Decode

    res = arg1 + arg2;                // Execute

    reg[15]=res;                      // Write-Back
```

II. Fault injection approach
4 – Software fault injection

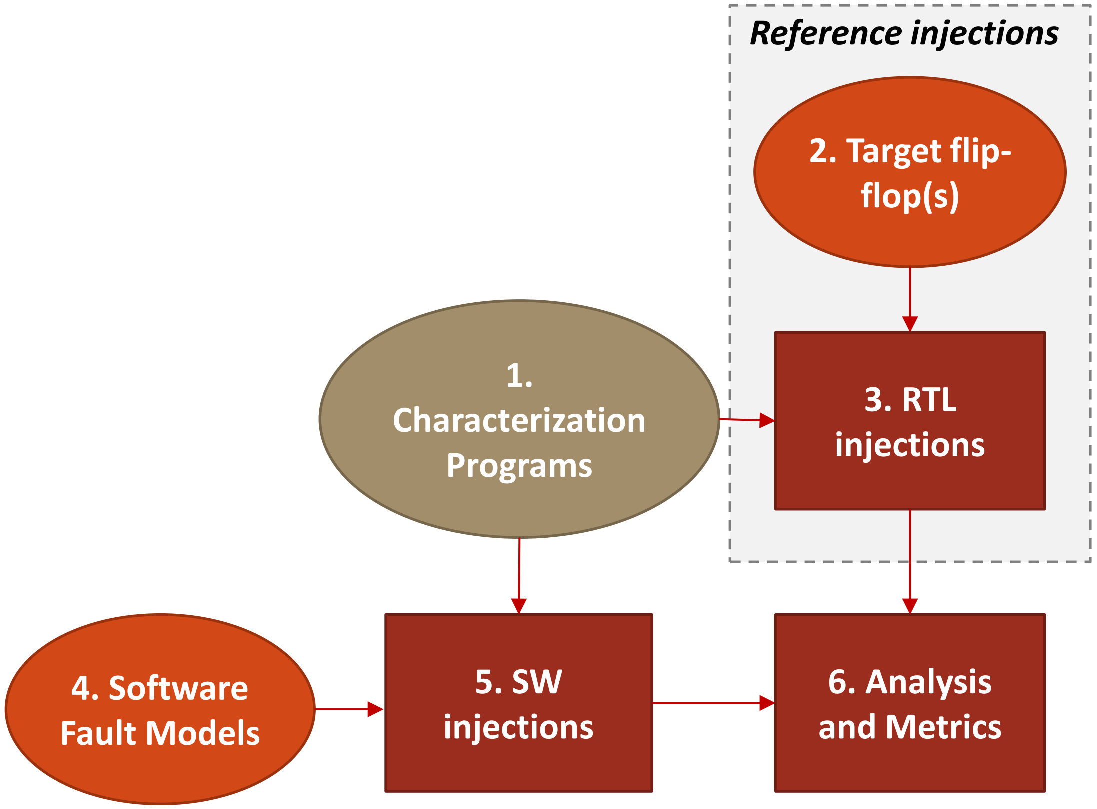


```
         […]
0x06ac:    ADDI  x15 = x0 + 85
         […]
```

```
I06ac: // ADDI x15, x0, 85
  arg1 = reg[0];  arg2 = 85;          // Decode
  if(injection)  arg1=fwd;
  res = arg1 + arg2;                  // Execute
  fwd=res;
  reg[15]=res;                        // Write-Back
```

III. Cross-layer analyses

1 - Inputs



III. Cross-layer analyses

1 - Inputs

- **Characterization programs:**
  - 105 small assembly contexts
  - VerifyPIN [12] version 6
  - LittleXorkey

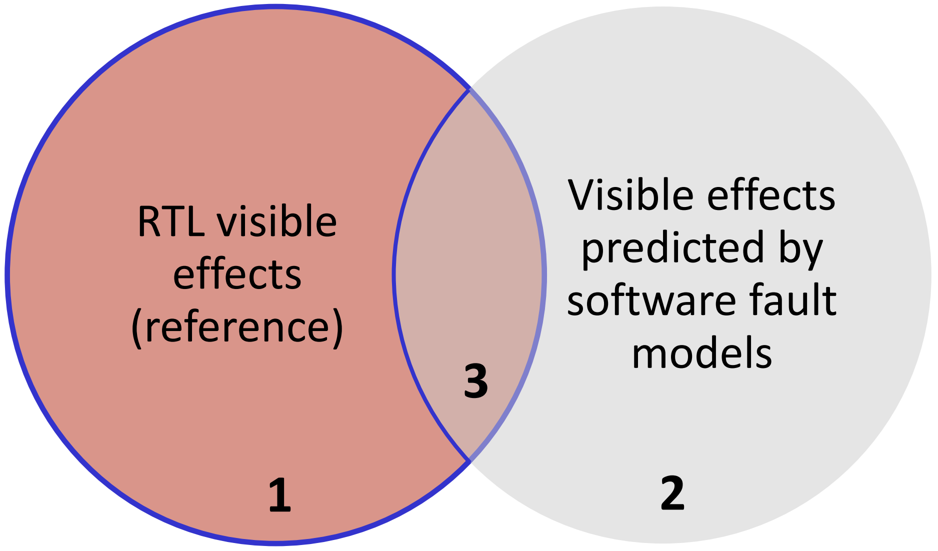
- **Target flip-flops:**
  - All flip-flops from the last three pipeline stages. Total: 1308 flip-flops. Excludes the register-file and memory.

- **Software fault models: 49 models built mainly to cover single-bit injections**
  - 32 simple bit-flips in the result of the instruction
  - 6 typical (skip, test inversion…)
  - 11 non-typical (related to forwarding for example)

III. Cross-layer analyses

2 - Coverage metric

- **Among the RTL faults, what proportion are correctly predicted by software fault models ?**
- $Coverage = \dfrac{|area\ 3|}{|area\ 1| + |area\ 3|}$


RTL visible effects (reference)
Visible effects predicted by software fault models
1
2
3

III. Cross-layer analyses

2 - Coverage metric

- **Results – exhaustive single-bit campaigns**

	Silent	Exception	Unknown	Analyzable	Coverage
Assembly contexts	80.6%	3.3%	0.6%	15.4%	24.0%
VerifyPIN	65.5%	2.3%	22.4%	9.8%	28.7%
LittleXorKey	76.6%	2.7%	2.8%	17.9%	29.0%

- **Results – statistical multiple-bit campaigns on asm contexts**

	Silent	Exception	Unknown	Analyzable	Coverage
1-bit	80.6%	3.3%	0.6%	15.4%	24.04%
2-bit	66.2 ± 0.3%	6.4 ± 0.2%	1.2 ± 0.1%	26.1 ± 0.3%	~ 22.51%
3-bit	54.8 ± 0.4%	9.4 ± 0.2%	1.7 ± 0.1%	34.0 ± 0.3%	~ 20.21%
4-bit	46.6 ± 0.5%	12.0 ± 0.3%	2.3 ± 0.2%	39.2 ± 0.5%	~ 18.41%
5-bit	40.0 ± 0.5%	14.4 ± 0.4%	2.5 ± 0.2%	43.1 ± 0.5%	~ 17.68%

III. Cross-layer analyses

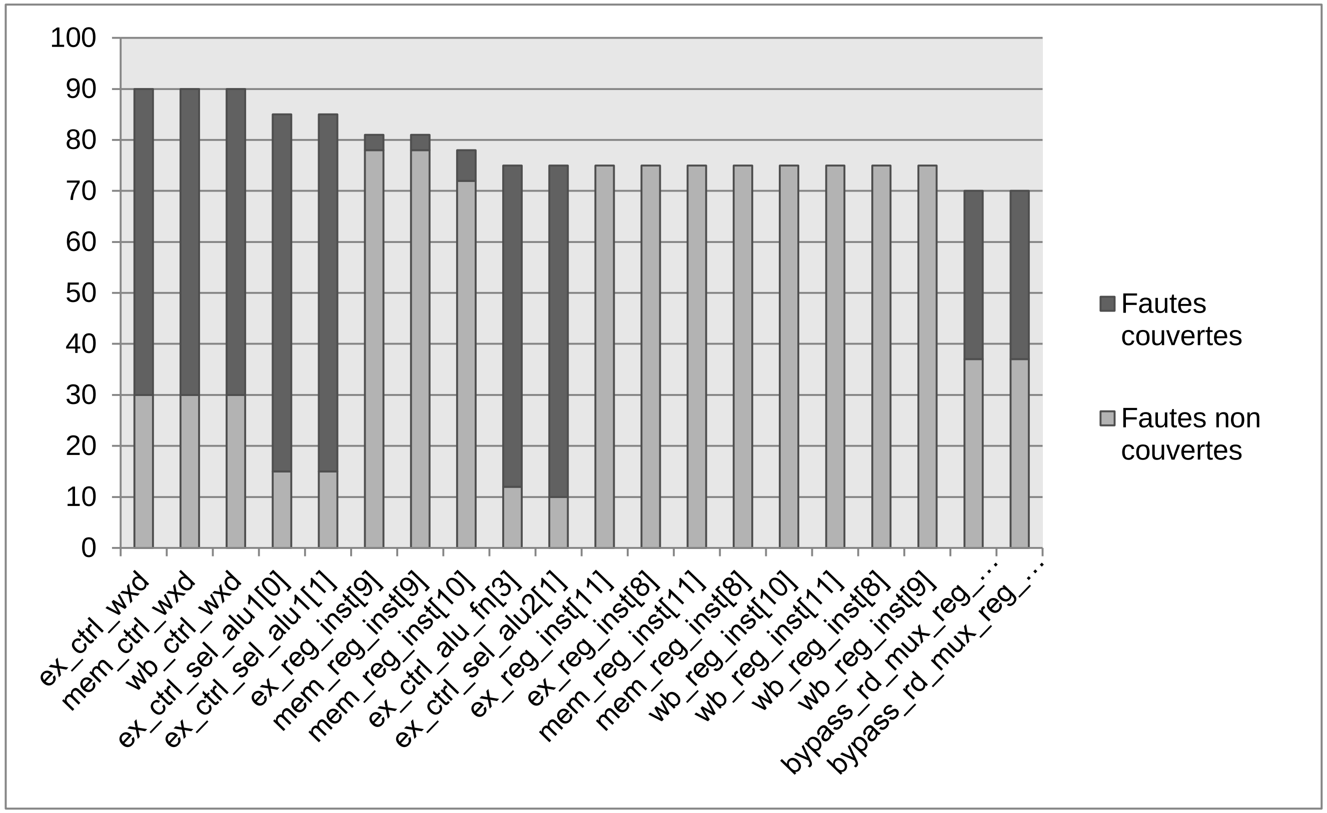
2 - Coverage metric

- **Low coverage shows the difficulty to model faults.**
- **Reasons to explain this low coverage:**
  - Injection in many hidden registers
  - Strict comparisons (without considering the instantaneous effect, coverages increase to 24.7%, 44,2% and 49,4%)
- **Among the best models:**
  - Extended instruction skip (a little bit better than typical instruction skip)

III. Cross-layer analyses

3 – Uncovered flip-flops

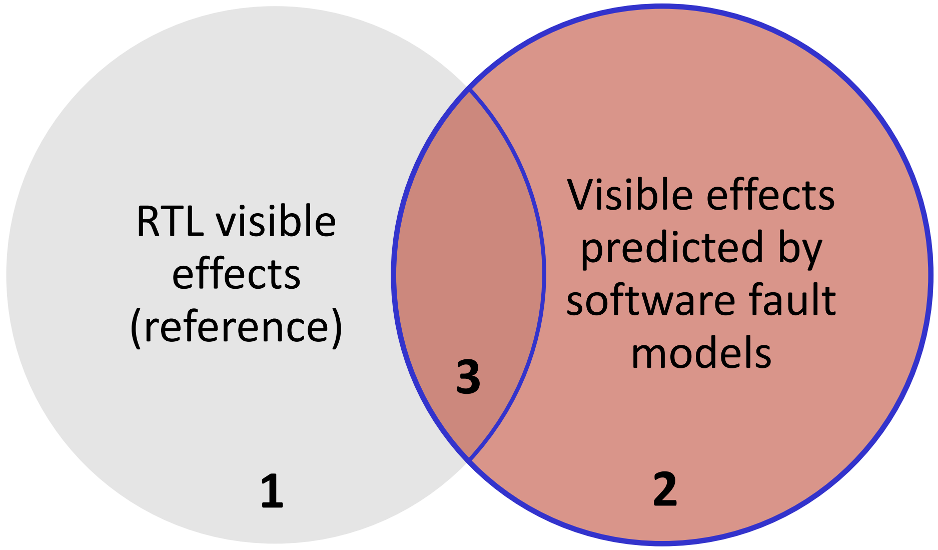
- Flip-flops creating the most faulty behaviors



III. Cross-layer analyses

4 – Fidelity metric

- **Among software faults, what proportion correctly predicts actual RTL faults ?**
- $$\boldsymbol{Fidelity} = \frac{|\boldsymbol{area}\ 3|}{|\boldsymbol{area}\ 2| + |\boldsymbol{area}\ 3|}$$


RTL visible effects (reference)
Visible effects predicted by software fault models
3
1
2

III. Cross-layer analyses

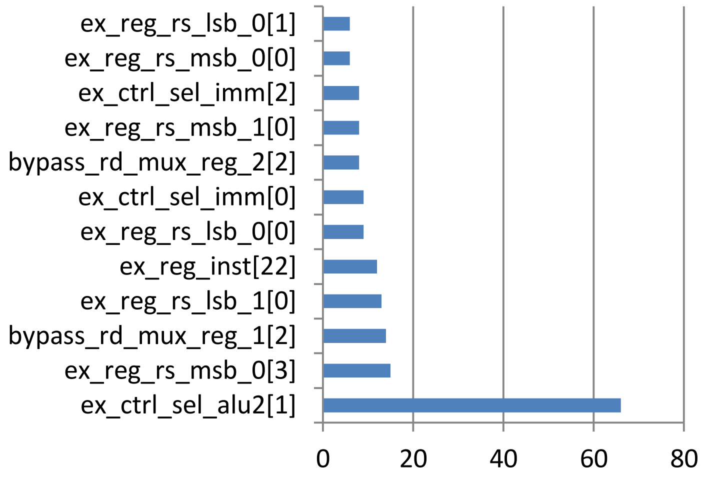
4 – Fidelity metric

- Globally, fidelity = 76.5%
- Very good models: replacing an argument or the result of an operation by 0 ~100%
- Good models: instruction skip ~80%
- Bad models: two models have a fidelity of 60.7% and 41.4%. They could be enhanced.

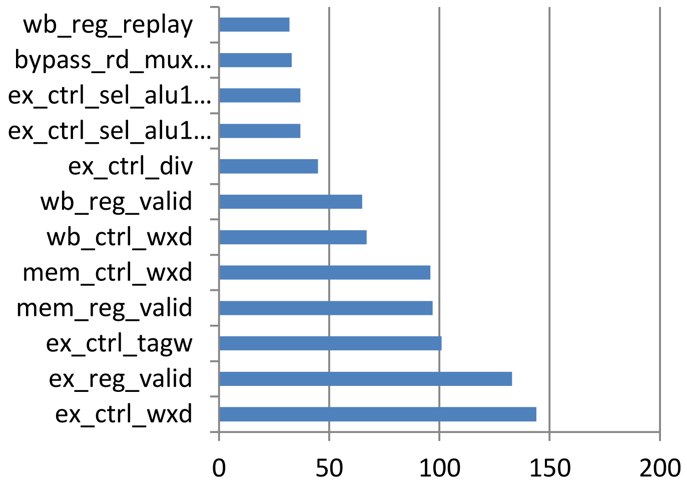
III. Cross-layer analyses

5 – Model profiles

- How to reproduce in an RTL simulation behaviors predicted by a software fault model?
- Model profiles show the most likely flip-flops to target.



Number of times a flip-flop produces the same effect as the **arg2_4** fault model.



Number of times a flip-flop produces the same effect as the **skip** fault model.

IV. Conclusion

- Presented a cross-layer approach to study fault injection, based on precise comparisons between RTL and software injections
- Multiple analyses to think security globally hardware + software

- Perspective: using other hardware fault models (considering limited attacker power)

Thanks for your attention !

Questions ?

References

[1] Joint Interpretation Library, “Application of Attack Potential to Smartcards.” Jan-2013.

[2] N. Moro, K. Heydemann, E. Encrenaz, and B. Robisson, “Formal verification of a software countermeasure against instruction skip attacks,” presented at the PROOFS 2013, 2013.

[3] M. L. Potet, L. Mounier, M. Puys, and L. Dureuil, “Lazart: A Symbolic Approach for Evaluation the Robustness of Secured Codes against Control Flow Injections,” in *Verification and Validation 2014 IEEE Seventh International Conference on Software Testing*, 2014, pp. 213–222.

[4] J. Vankeirsbilck, N. Penneman, H. Hallez, and J. Boydens, “Random Additive Signature Monitoring for Control Flow Error Detection,” *IEEE Trans. Reliab.*, vol. 66, no. 4, pp. 1178–1192, Dec. 2017.

[5] M. Christofi, B. Chetali, L. Goubin, and D. Vigilant, “Formal verification of an implementation of CRT-RSA algorithm,” presented at the Security Proofs for Embedded Systems (PROOFS), 2012, pp. 28–48.

[6] A. Höller, A. Krieg, T. Rauter, J. Iber, and C. Kreiner, “QEMUBased Fault Injection for a System-Level Analysis of Software Countermeasures Against Fault Attacks,” in *2015 Euromicro Conference on Digital System Design*, 2015, pp. 530–533.

[7] H. Cho, S. Mirkhani, C. Y. Cher, J. A. Abraham, and S. Mitra, “Quantitative evaluation of soft error injection techniques for robust system design,” in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2013, pp. 1–10.

[8] M. S. Kelly, K. Mayes, and J. F. Walker. 2017. Characterising a CPU fault attack model via run-time data analysis. In *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 79–84. DOI:https://doi.org/10.1109/HST.2017.7951802

[9] Louis Dureuil. 2016. Analyse de code et processus d’évaluation des composants sécurisés contre l’injection de faute. phdthesis. Communauté Université Grenoble Alpes. Retrieved October 16, 2017 from https://tel.archives-ouvertes.fr/tel-01403749/document

[10] Julien Proy, Karine Heydemann, Fabien Majéric, Albert Cohen, and Alexandre Berzati. 2019. Studying EM Pulse Effects on Superscalar Microarchitectures at ISA Level. *arXiv:1903.02623 [cs]* (March 2019). Retrieved March 12, 2019 from http://arxiv.org/abs/1903.02623

[11] J. Laurent, V. Beroulle, C. Deleuze, F. Pebay-Peyroula, and A. Papadimitriou, “Cross-layer analysis of software fault models and countermeasures against hardware fault attacks in a RISC-V processor,” Microprocessors and Microsystems, vol. 71, p. 102862, Nov. 2019, doi: 10.1016/j.micpro.2019.102862.

[12] L. Dureuil, G. Petiot, M.-L. Potet, T.-H. Le, A. Crohen, and P. de Choudens, “FISSC: A Fault Injection and Simulation Secure Collection,” 2016, pp. 3–11.

- **Coverage of the instruction skip model, and of 2 more advanced skip models for various injection campaigns**

Skip/skip_mem/skip_wb	1-bit	2-bit	3-bit	4-bit	5-bit
Assembly contexts	2.8/3.3/3.3	2.9/3.6/3.5	3.1/3.6/3.5	3.0/3.5/3.4	3.2/4.2/4.0
VerifyPIN	8.0/8.6/8.4	8.2/8.7/8.5	8.1/8.8/8.5	8.5/9.1/8.8	8.3/8.8/8.5
LittleXorKey	3.2/3.5/4.2	3.0/3.4/4.5	3.0/3.3/4.1	2.9/3.2/4.0	2.7/3.0/3.9