

CONTEXT

The goal is to implement the instruction duplication technique as a countermeasure against Fault Attacks on an ARM 32-bit Microcontroller[1,2]. Operating inside a compiler allowed us to reduce the security overhead thanks to the flexibility and code transformations opportunities offered by compilers

WORKFLOW



The user delimits the portions of the program to protect

```
@__to_secure__("fault")
int foo(int a, int b){
    . . .
    return a * b + a;
}
```

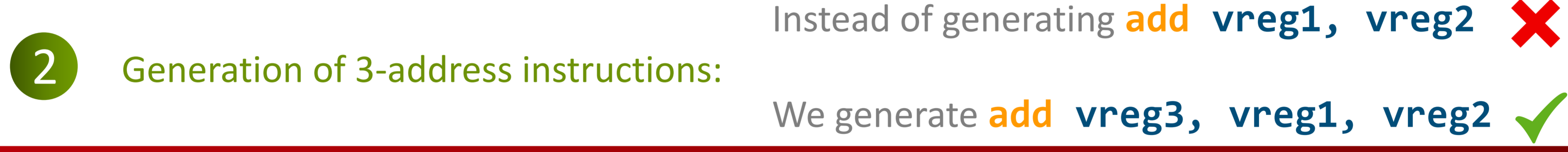
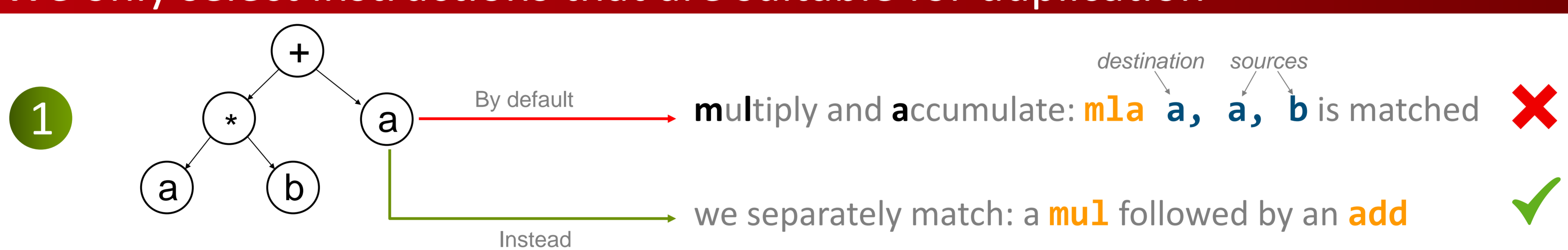
C source code

The user has a full control over parts of the code to protect

Instructions cannot be duplicated at the middle-end due to the SSA form



We only select instructions that are suitable for duplication

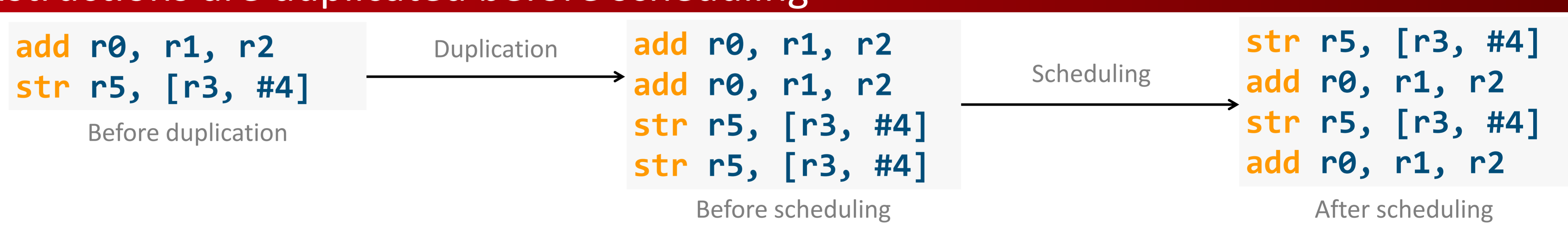


Registers are allocated in favor of duplication

The register allocator tends to reduce *register pressure*: Reusing the allocated registers as soon as possible
When the liveness intervals (L) of registers are disjoint: $\{L(vreg3)\} \cap \{L(vreg1), L(vreg2)\} = \emptyset$



Instructions are duplicated before scheduling



Comparison with assembly approach

	Instruction	Transformation	Duplication
Assembly approach	add r0, r0, r2	mov rx, r0 add r0, rx, r2	mov rx, r0 mov rx, r0 add r0, rx, r2 add r0, rx, r2 (X 4)
Our approach	add r0, r1, r2		add r0, r1, r2 add r0, r1, r2 (X 2)

AES 8-bit NIST on ARM Cortex-M3

Unprotected	Protected	Overhead
8541 cycles	17311 cycles	× 2.03

ONGOING WORK & REFERENCES

- ### FUTURE WORK
- Using code annotation for more flexibility when defining the code regions to protect
 - Automatic identification of the most vulnerable parts of the program

- ### REFERENCES
- [1] Barenghi et al. Countermeasures against fault attacks on software implemented AES
 - [2] Moro et al. Electromagnetic Fault Injection : Towards a Fault Model on a 32-bit Microcontroller

LEGEND
 Duplicable
 Not duplicable