CMEmu: Synthesizing a Cycle-Exact Model of Program Execution on ARM Cortex-M from In-Code Timing Measurements

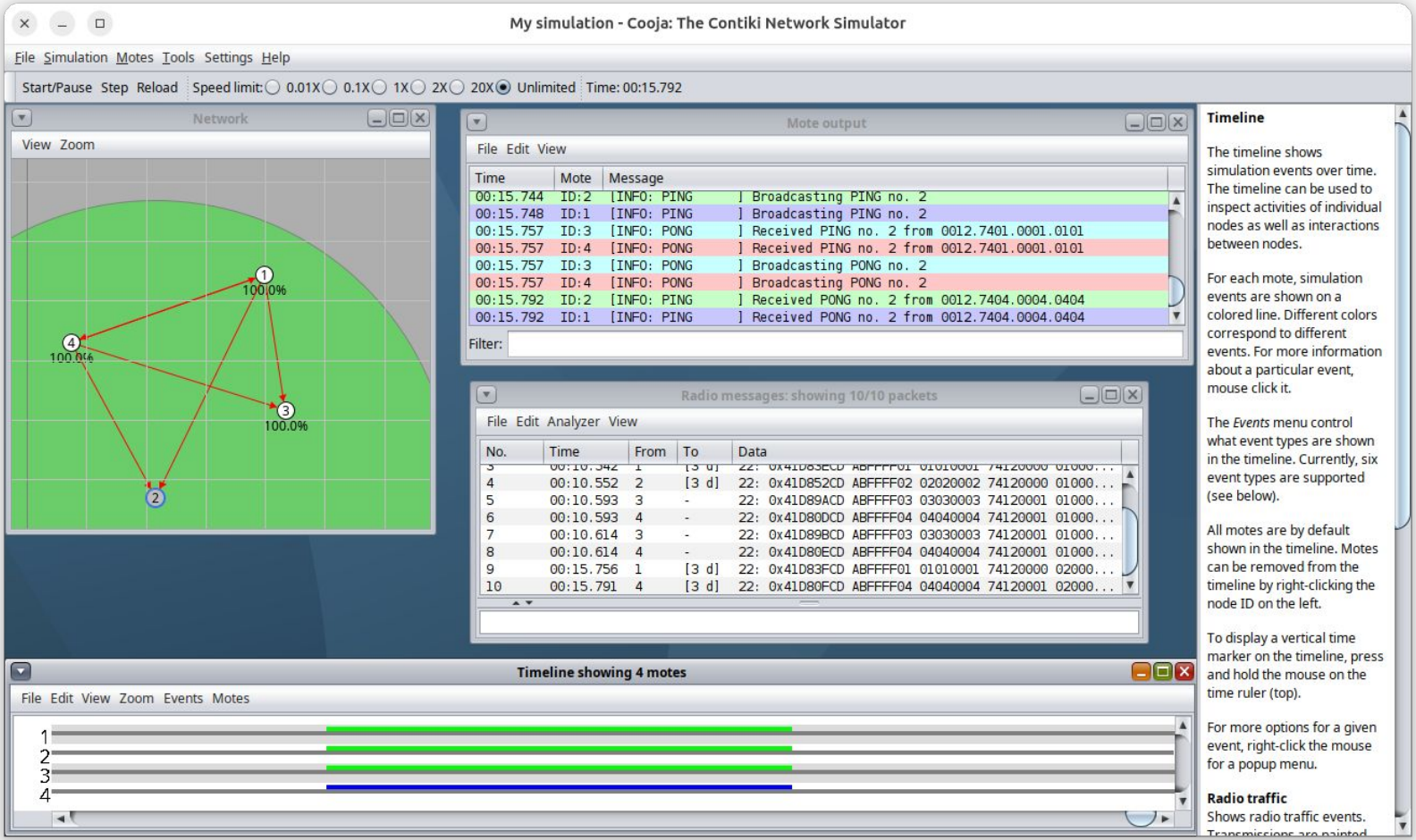
Maciej Matraszek, Mateusz Banaszek, Wojciech Ciszewski, Artur Jamro, Wojciech Kordalski, Daniel Gutowski, Michał Siwiński, Bartłomiej Dalak, and Konrad Iwanicki

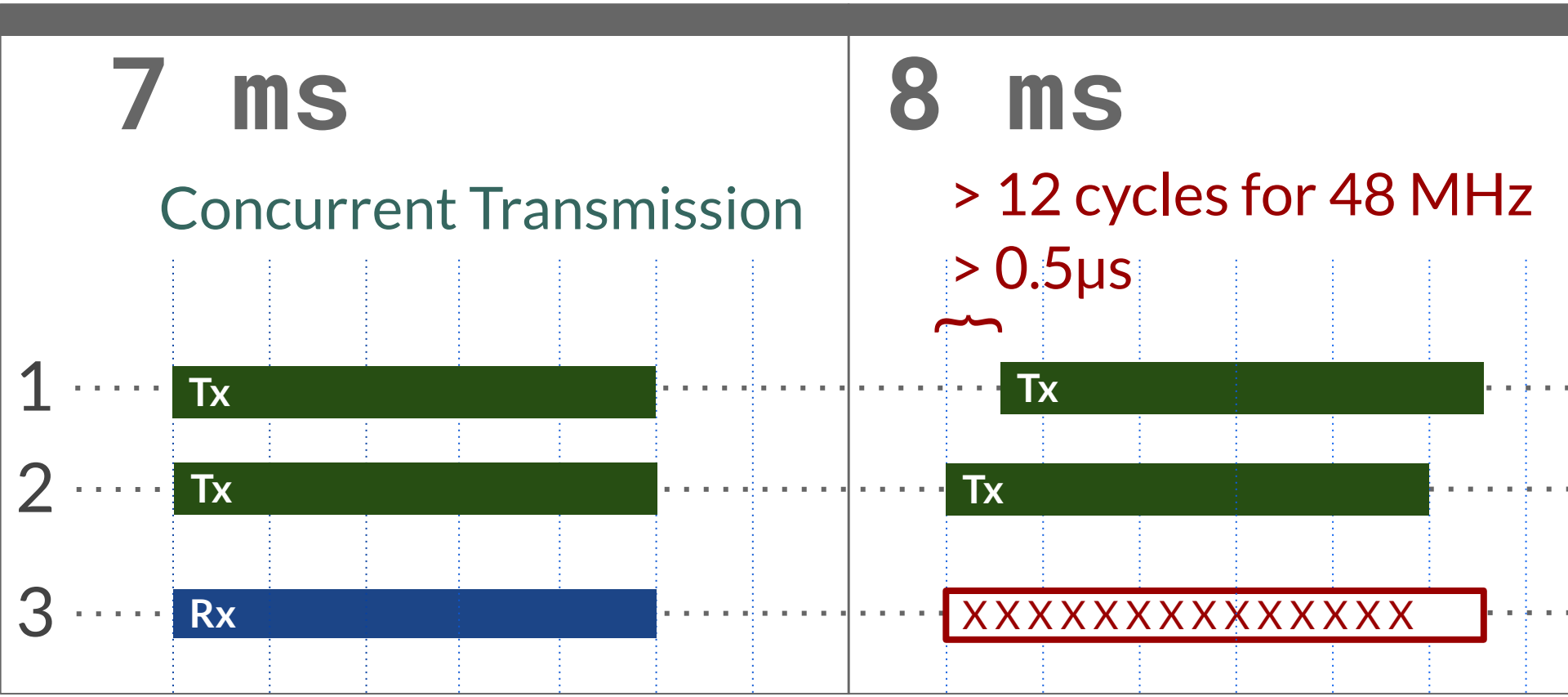
UNIVERSITY OF WARSAW

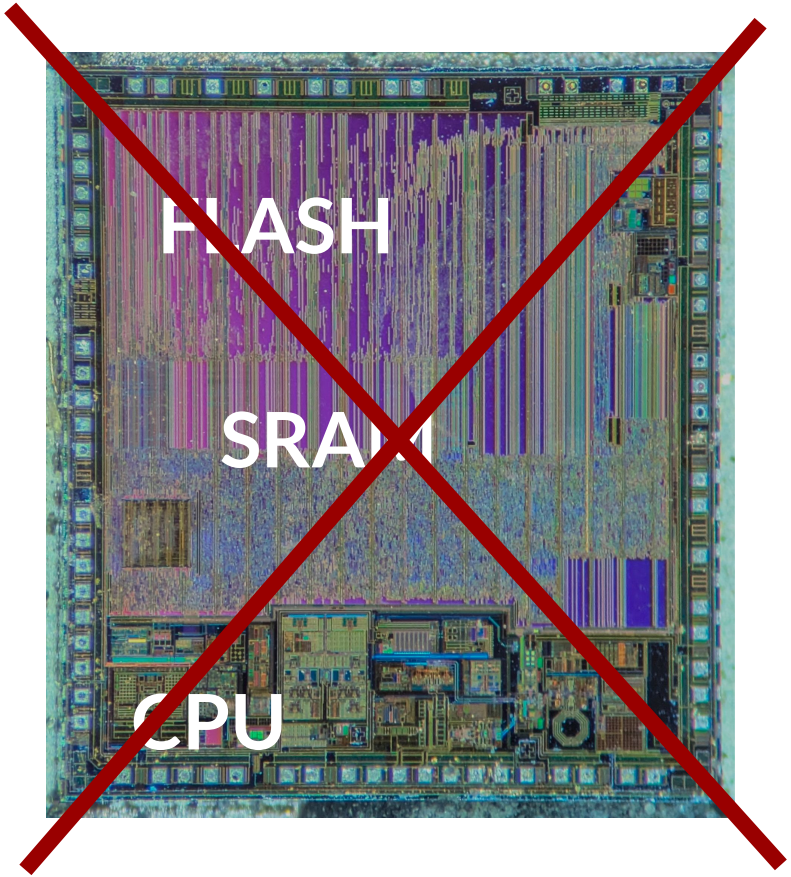
MSWiM 2025, Barcelona, Spain, October 30, 2025

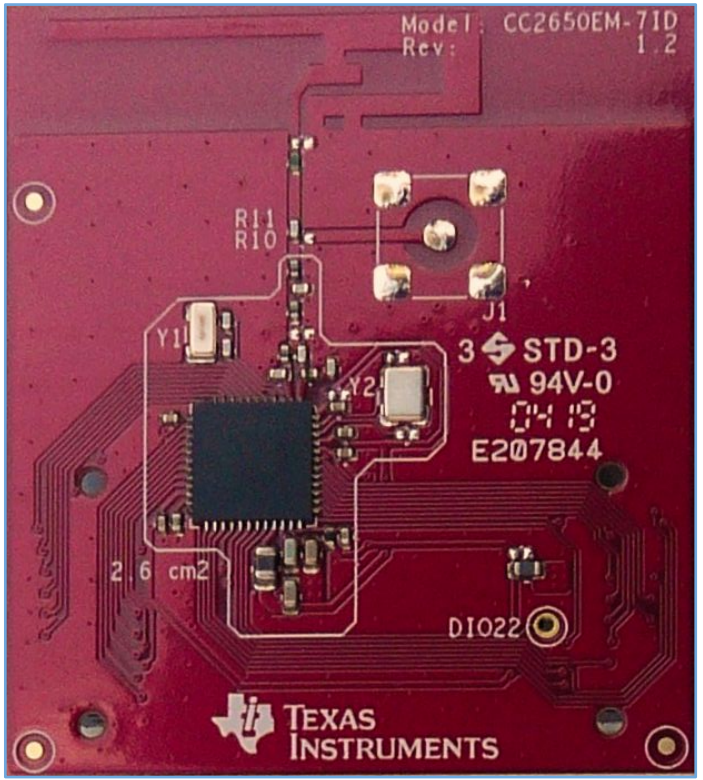

00000bd4
add.w r1, r1, r1
F:ID
F:IA
F:IA+
F:IA+
F:ID+
F:ID+
F:ID
D
X
00000bd8
ldr.w r2, [r0, #0x4]
F:IA
F:ID+
F:ID+
F:ID
F:IA
F:IA+
F:IA+
F:ID+
F:ID+
F:ID
D
D:AGU
X:DA
X:DD
00000bdc
add.w r8, r8, r9
F:IA
F:IA+
F:IA+
F:ID
D
X
00000be0
add.w r6, r6, r7
F:IA
F:ID+
F:ID+
F:ID
D
X
00000be4
ittet.n eq
F:IA
F:IA+
F:IA+
F:ID
D
X
00000be6
nop
00000be8
F:IA
F:ID+
F:ID+
00000bea
ldr.w r6, [r3, #0x0]
00000bec
F:IA
F:IA+
00000bee
ldr.n r6, [r3, #0x0]
00000bf0
add.n r1, r1, r1
00000bf2
ldr.w r3, [r0, #0x4]
00000bf4
00000bf6
bl.w [pc, #0x99ca]
00000bf8

Motivation


My simulation - Cooja: The Contiki Network Simulator
File Simulation Motes Tools Settings Help
Start/Pause Step Reload Speed limit: 0.01X 0.1X 1X 2X 20X Unlimited Time: 00:15.792
Network
View Zoom
1
100.0%
4
100.0%
3
100.0%
2
Mote output
File Edit View
Time Mote Message
00:15.744 ID:2 [INFO: PING] Broadcasting PING no. 2
00:15.748 ID:1 [INFO: PING] Broadcasting PING no. 2
00:15.757 ID:3 [INFO: PONG] Received PING no. 2 from 0012.7401.0001.0101
00:15.757 ID:4 [INFO: PONG] Received PING no. 2 from 0012.7401.0001.0101
00:15.757 ID:3 [INFO: PONG] Broadcasting PONG no. 2
00:15.757 ID:4 [INFO: PONG] Broadcasting PONG no. 2
00:15.792 ID:2 [INFO: PING] Received PONG no. 2 from 0012.7404.0004.0404
00:15.792 ID:1 [INFO: PING] Received PONG no. 2 from 0012.7404.0004.0404
Filter:
Radio messages: showing 10/10 packets
File Edit Analyzer View
No. Time From To Data
4 00:10.552 2 [3 d] 22: 0x41D852CD ABFFFF02 02020002 74120000 01000...
5 00:10.593 3 - 22: 0x41D89ACD ABFFFF03 03030003 74120001 01000...
6 00:10.593 4 - 22: 0x41D80DCD ABFFFF04 04040004 74120001 01000...
7 00:10.614 3 - 22: 0x41D89BCD ABFFFF03 03030003 74120001 01000...
8 00:10.614 4 - 22: 0x41D80ECD ABFFFF04 04040004 74120001 01000...
9 00:15.756 1 [3 d] 22: 0x41D83FCD ABFFFF01 01010001 74120000 02000...
10 00:15.791 4 [3 d] 22: 0x41D80FCD ABFFFF04 04040004 74120001 02000...
Timeline showing 4 motes
File Edit View Zoom Events Motes
1
2
3
4
Timeline
The timeline shows simulation events over time. The timeline can be used to inspect activities of individual nodes as well as interactions between nodes.
For each mote, simulation events are shown on a colored line. Different colors correspond to different events. For more information about a particular event, mouse click it.
The Events menu control what event types are shown in the timeline. Currently, six event types are supported (see below).
All motes are by default shown in the timeline. Motes can be removed from the timeline by right-clicking the node ID on the left.
To display a vertical time marker on the timeline, press and hold the mouse on the time ruler (top).
For more options for a given event, right-click the mouse for a popup menu.
Radio traffic
Shows radio traffic events.

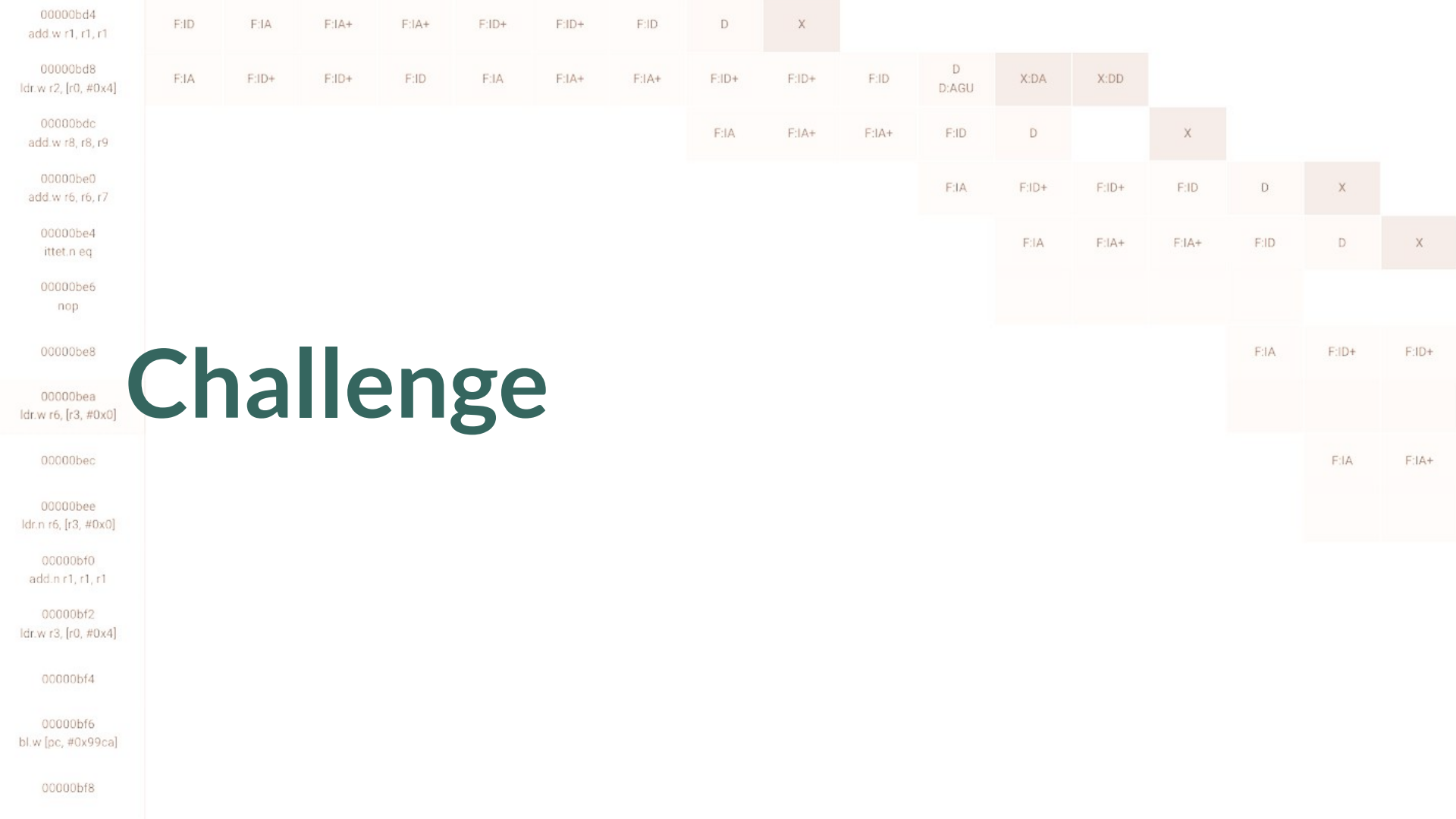

7 ms
Concurrent Transmission
1
Tx
2
Tx
3
Rx
8 ms
> 12 cycles for 48 MHz
> 0.5µs
Tx
Tx
XXXXXXXXXXXXXXX



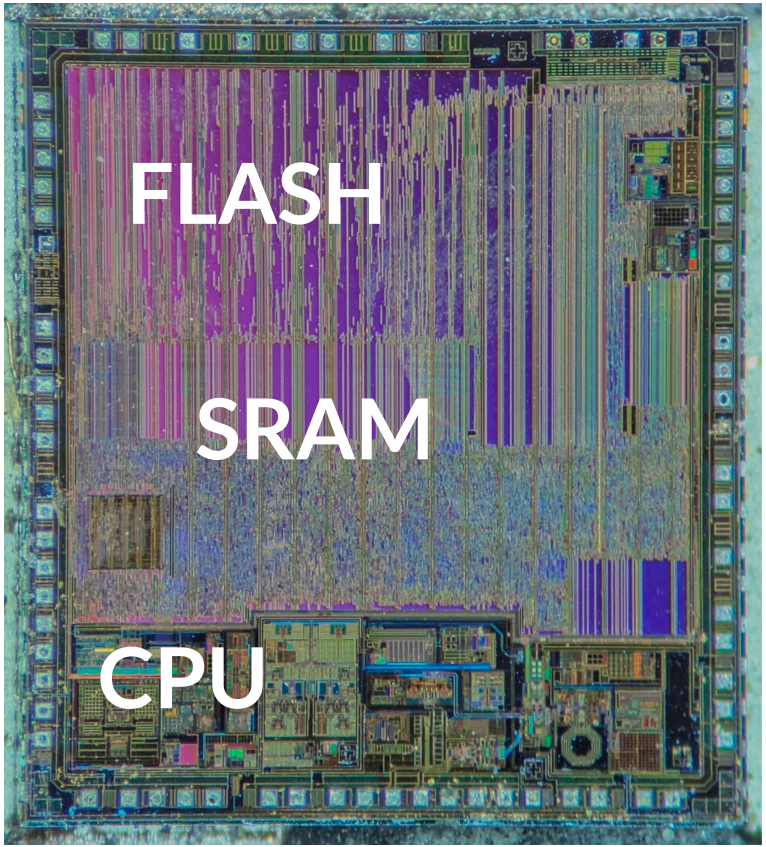


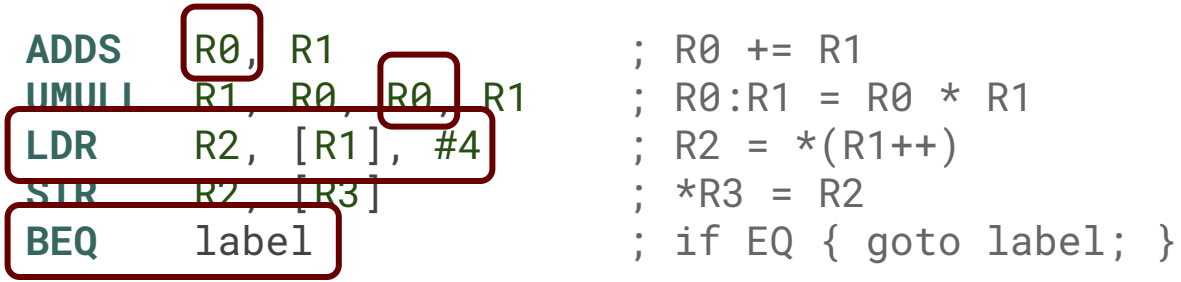
~~reverse engineerig~~

modeling execution time


00000bd4
add.w r1, r1, r1
F:ID
F:IA
F:IA+
F:IA+
F:ID+
F:ID+
F:ID
D
X
00000bd8
ldr.w r2, [r0, #0x4]
F:IA
F:ID+
F:ID+
F:ID
F:IA
F:IA+
F:IA+
F:ID+
F:ID+
F:ID
D
D:AGU
X:DA
X:DD
00000bdc
add.w r8, r8, r9
F:IA
F:IA+
F:IA+
F:ID
D
X
00000be0
add.w r6, r6, r7
F:IA
F:ID+
F:ID+
F:ID
D
X
00000be4
ittet.n eq
F:IA
F:IA+
F:IA+
F:ID
D
X
00000be6
nop
00000be8
F:IA
F:ID+
F:ID+
00000bea
ldr.w r6, [r3, #0x0]
00000bec
F:IA
F:IA+
00000bee
ldr.n r6, [r3, #0x0]
00000bf0
add.n r1, r1, r1
00000bf2
ldr.w r3, [r0, #0x4]
00000bf4
00000bf6
bl.w [pc, #0x99ca]
00000bf8

Challenge

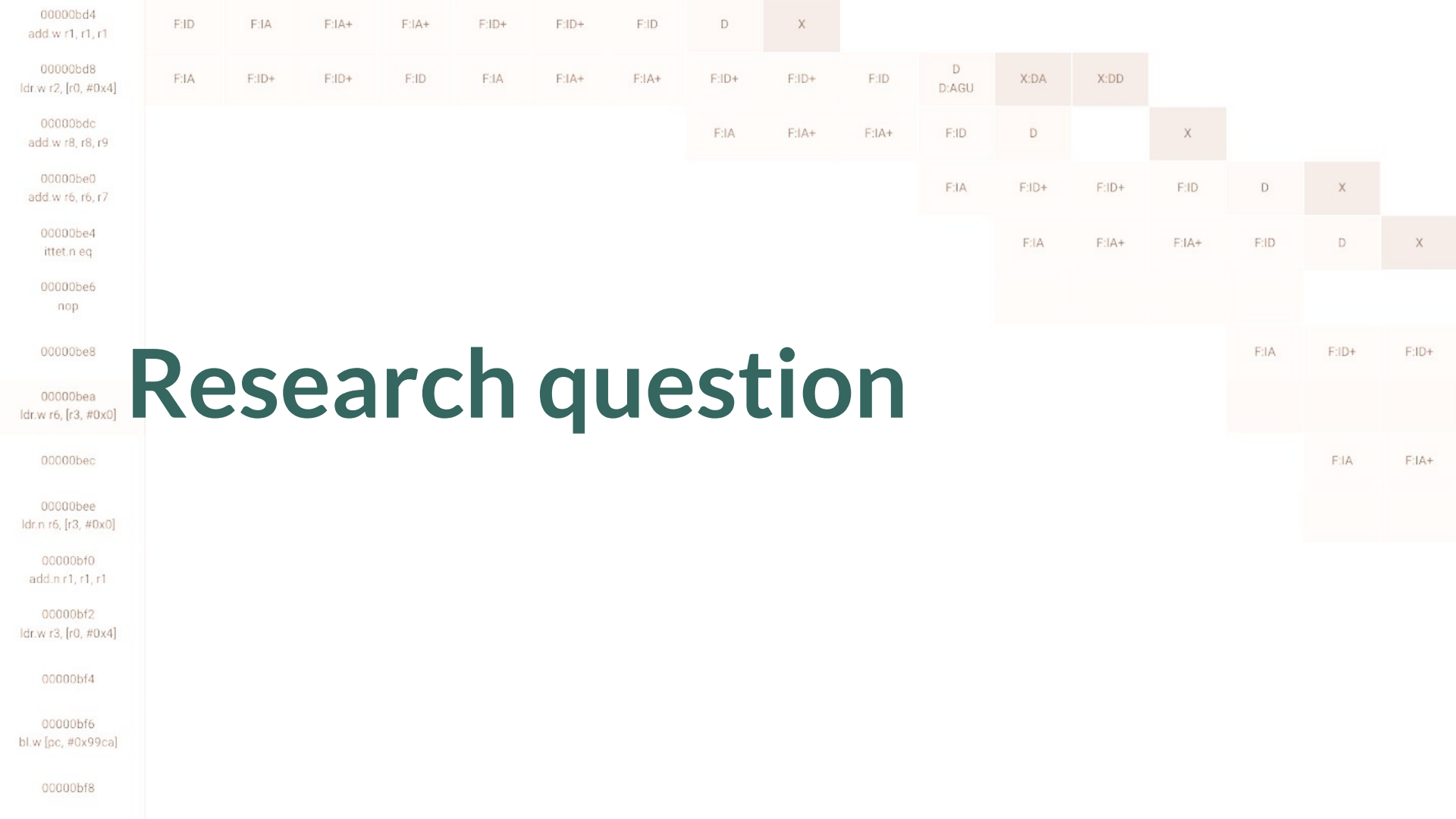

FLASH
SRAM
CPU



read-after-write (RAW) hazard

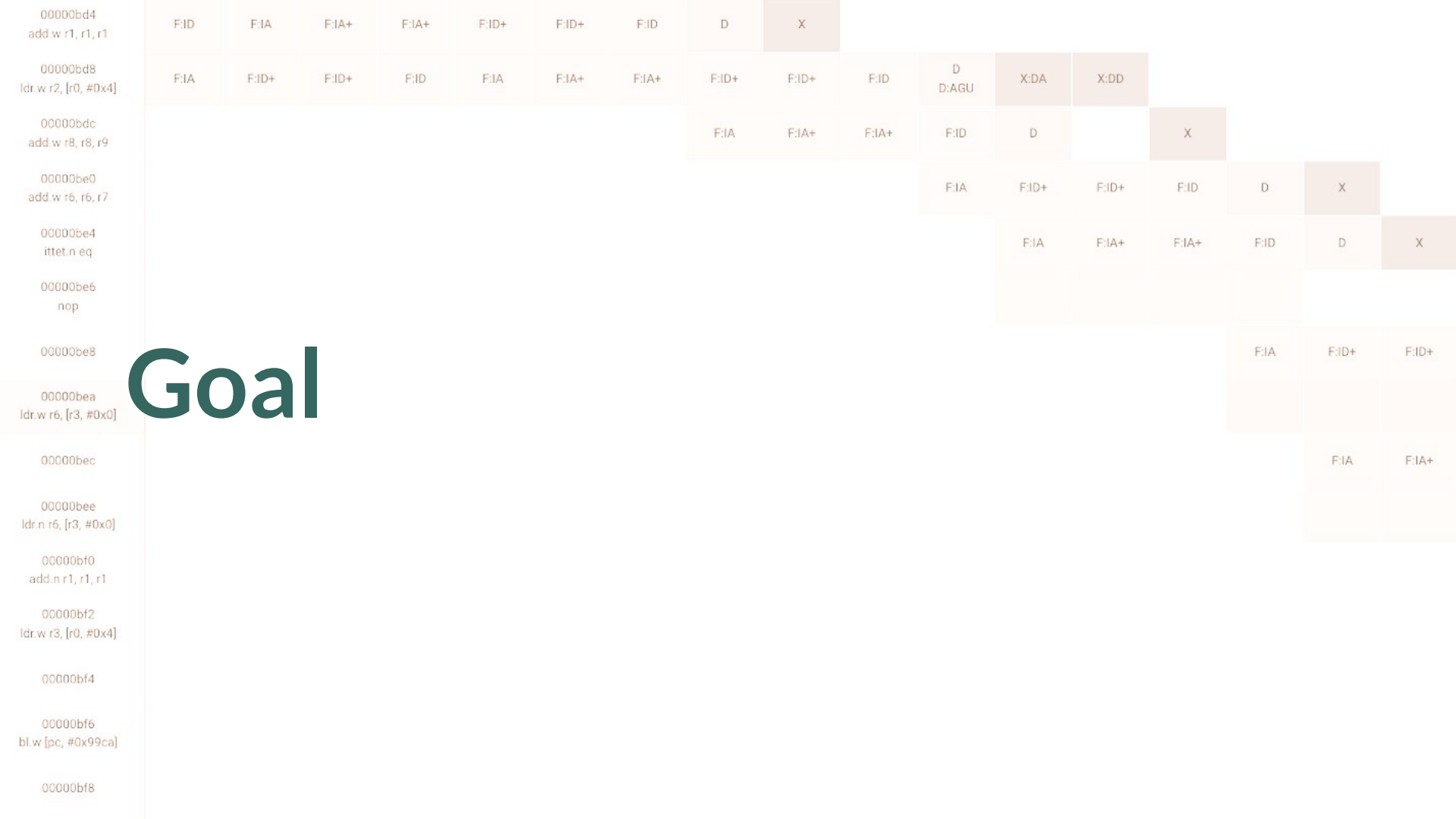
speculative loading of the target instruction

memory access (bus contention)

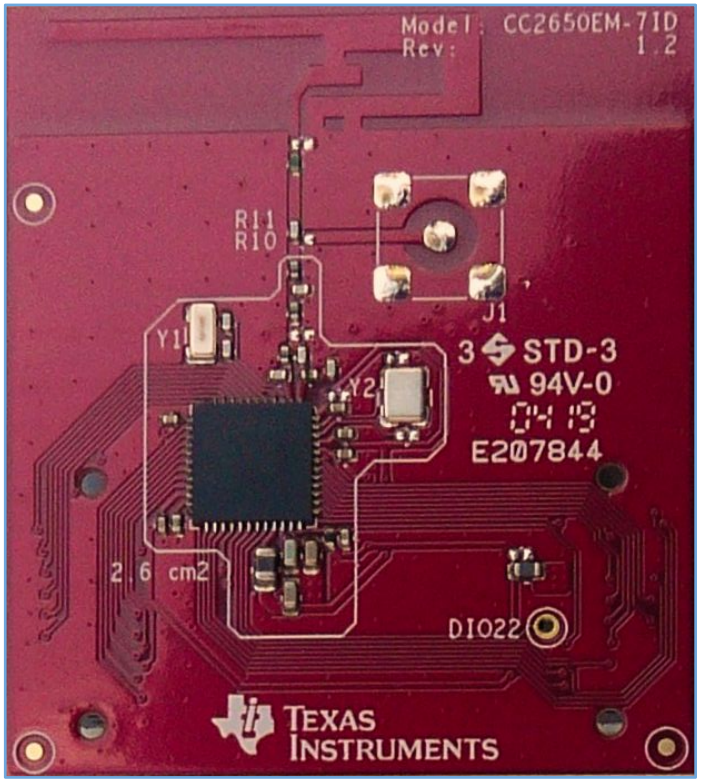


Research question

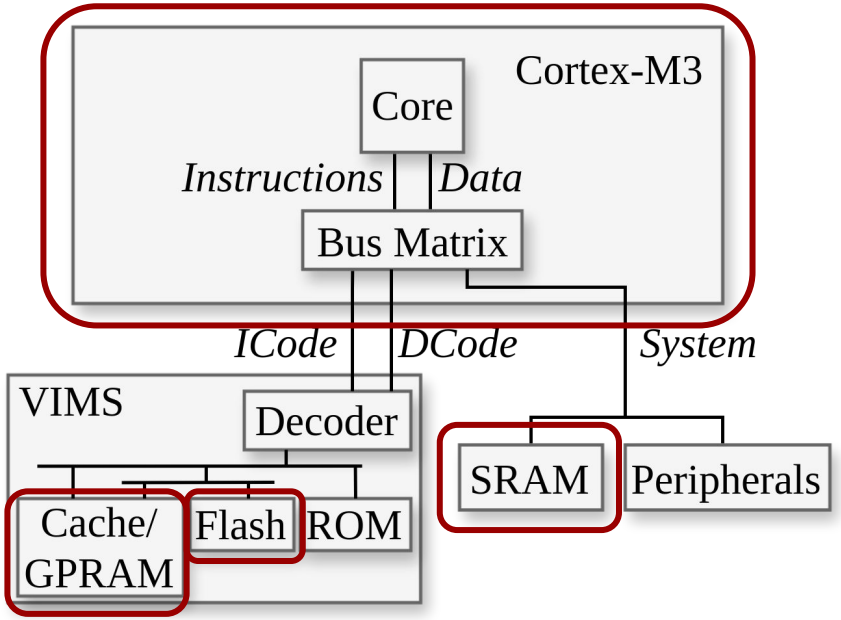
Can a cycle-exact model of a modern microcontroller be devised without access to its hardware sources?


00000bd4
add.w r1, r1, r1
F:ID
F:IA
F:IA+
F:IA+
F:ID+
F:ID+
F:ID
D
X
00000bd8
ldr.w r2, [r0, #0x4]
F:IA
F:ID+
F:ID+
F:ID
F:IA
F:IA+
F:IA+
F:ID+
F:ID+
F:ID
D
D:AGU
X:DA
X:DD
00000bdc
add.w r8, r8, r9
F:IA
F:IA+
F:IA+
F:ID
D
X
00000be0
add.w r6, r6, r7
F:IA
F:ID+
F:ID+
F:ID
D
X
00000be4
ittet.n eq
F:IA
F:IA+
F:IA+
F:ID
D
X
00000be6
nop
00000be8
F:IA
F:ID+
F:ID+
00000bea
ldr.w r6, [r3, #0x0]
00000bec
F:IA
F:IA+
00000bee
ldr.n r6, [r3, #0x0]
00000bf0
add.n r1, r1, r1
00000bf2
ldr.w r3, [r0, #0x4]
00000bf4
00000bf6
bl.w [pc, #0x99ca]
00000bf8

Goal



Texas Instruments CC2650

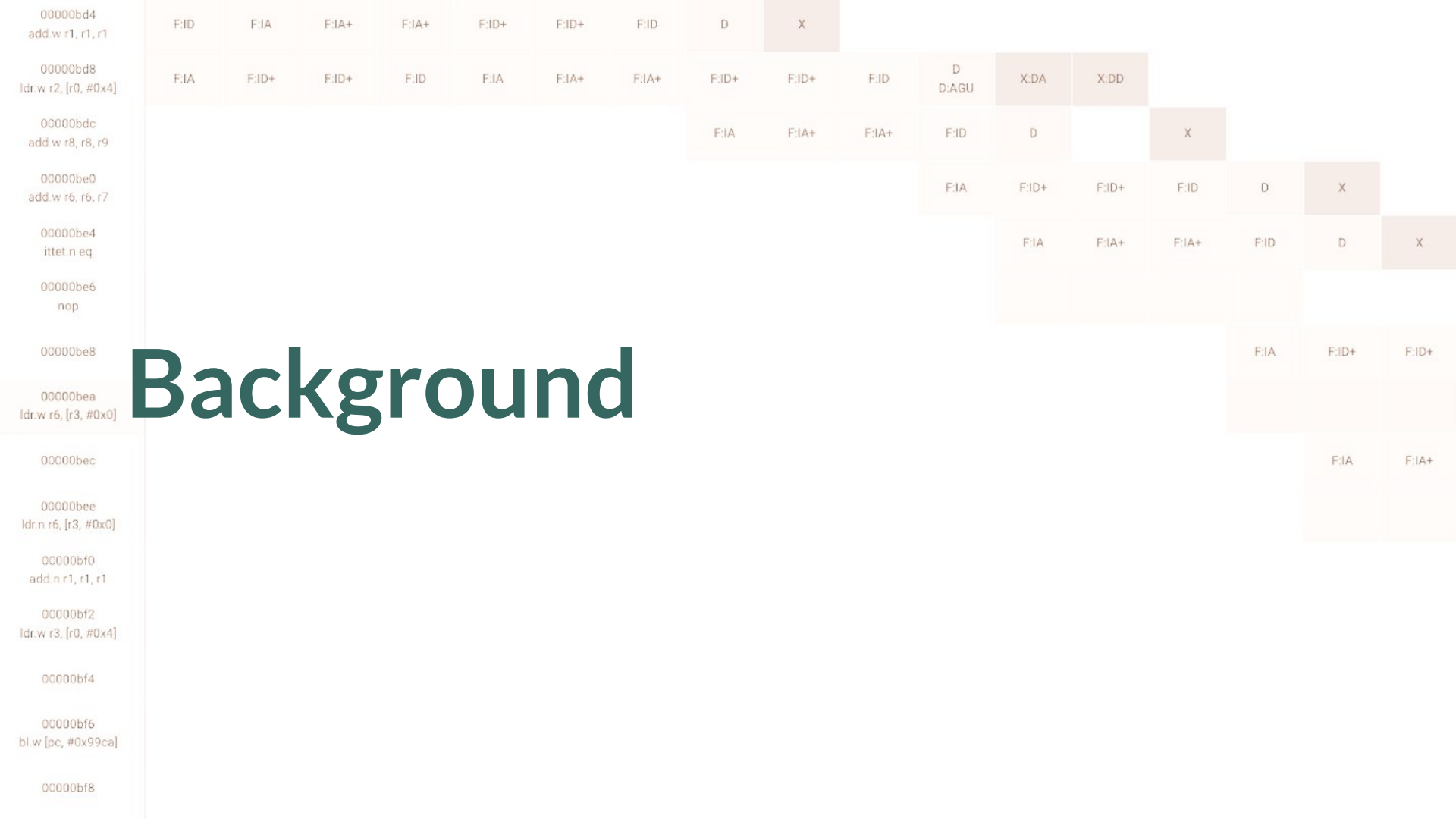


start()

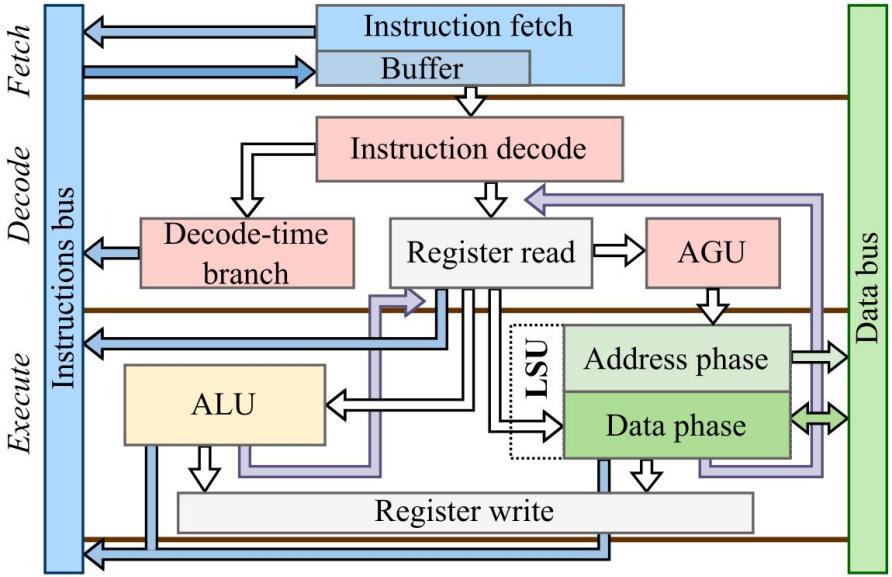
```
ADDS    R0, R1
UMULL   R1, R0, R0, R1
LDR     R2, [R1], #4
STR     R2, [R3]
BEQ     label
```

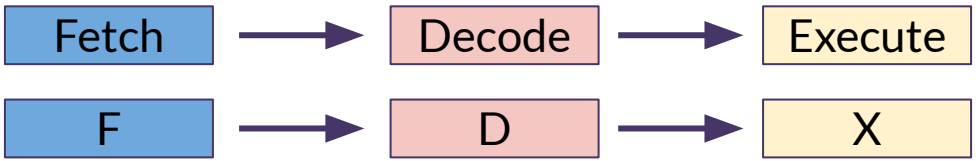
stop()

built-in counters: 

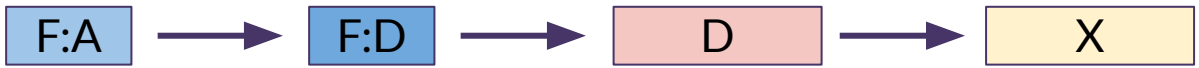

00000bd4
add.w r1, r1, r1
F:ID
F:IA
F:IA+
F:IA+
F:ID+
F:ID+
F:ID
D
X
00000bd8
ldr.w r2, [r0, #0x4]
F:IA
F:ID+
F:ID+
F:ID
F:IA
F:IA+
F:IA+
F:ID+
F:ID+
F:ID
D
D:AGU
X:DA
X:DD
00000bdc
add.w r8, r8, r9
F:IA
F:IA+
F:IA+
F:ID
D
X
00000be0
add.w r6, r6, r7
F:IA
F:ID+
F:ID+
F:ID
D
X
00000be4
ittet.n eq
F:IA
F:IA+
F:IA+
F:ID
D
X
00000be6
nop
00000be8
F:IA
F:ID+
F:ID+
00000bea
ldr.w r6, [r3, #0x0]
00000bec
F:IA
F:IA+
00000bee
ldr.n r6, [r3, #0x0]
00000bf0
add.n r1, r1, r1
00000bf2
ldr.w r3, [r0, #0x4]
00000bf4
00000bf6
bl.w [pc, #0x99ca]
00000bf8

Background


Fetch
Decode
Execute
Instructions bus
Instruction fetch
Buffer
Instruction decode
Decode-time branch
Register read
AGU
LSU
Address phase
Data phase
ALU
Register write
Data bus

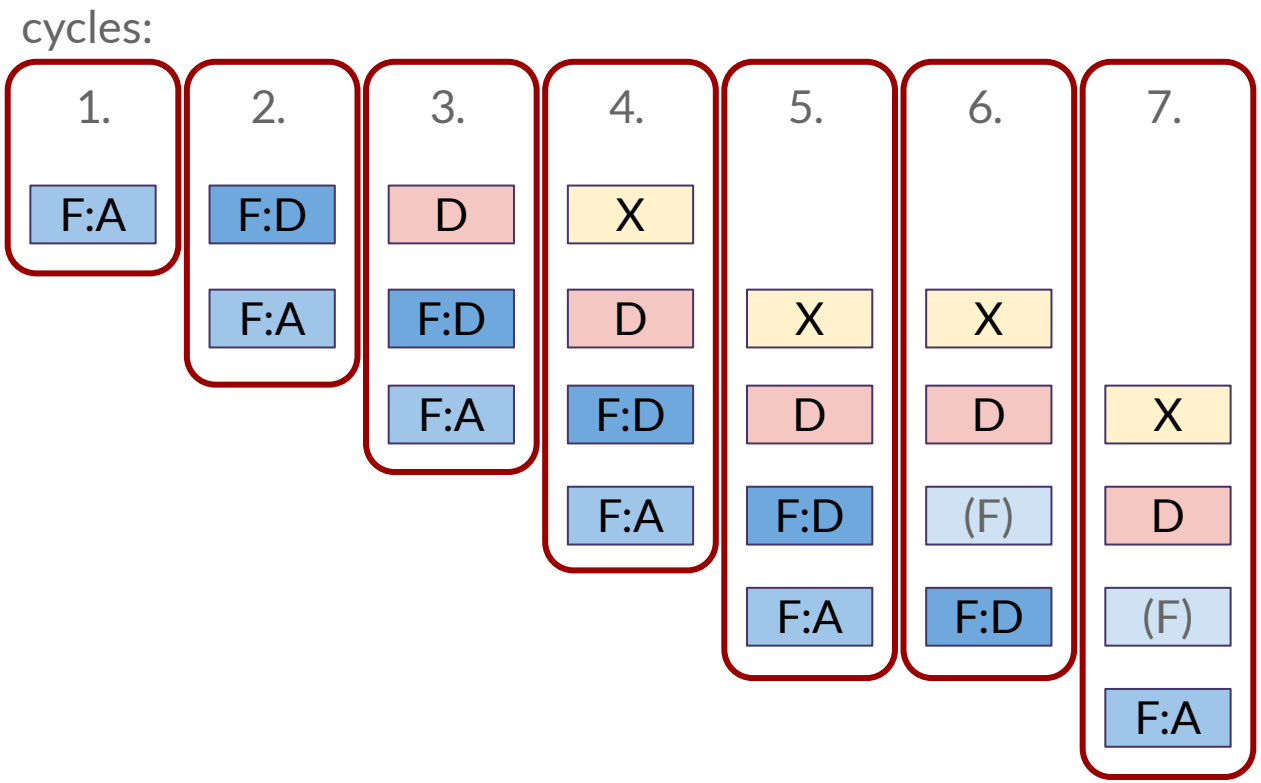

Fetch
Decode
Execute
F
D
X

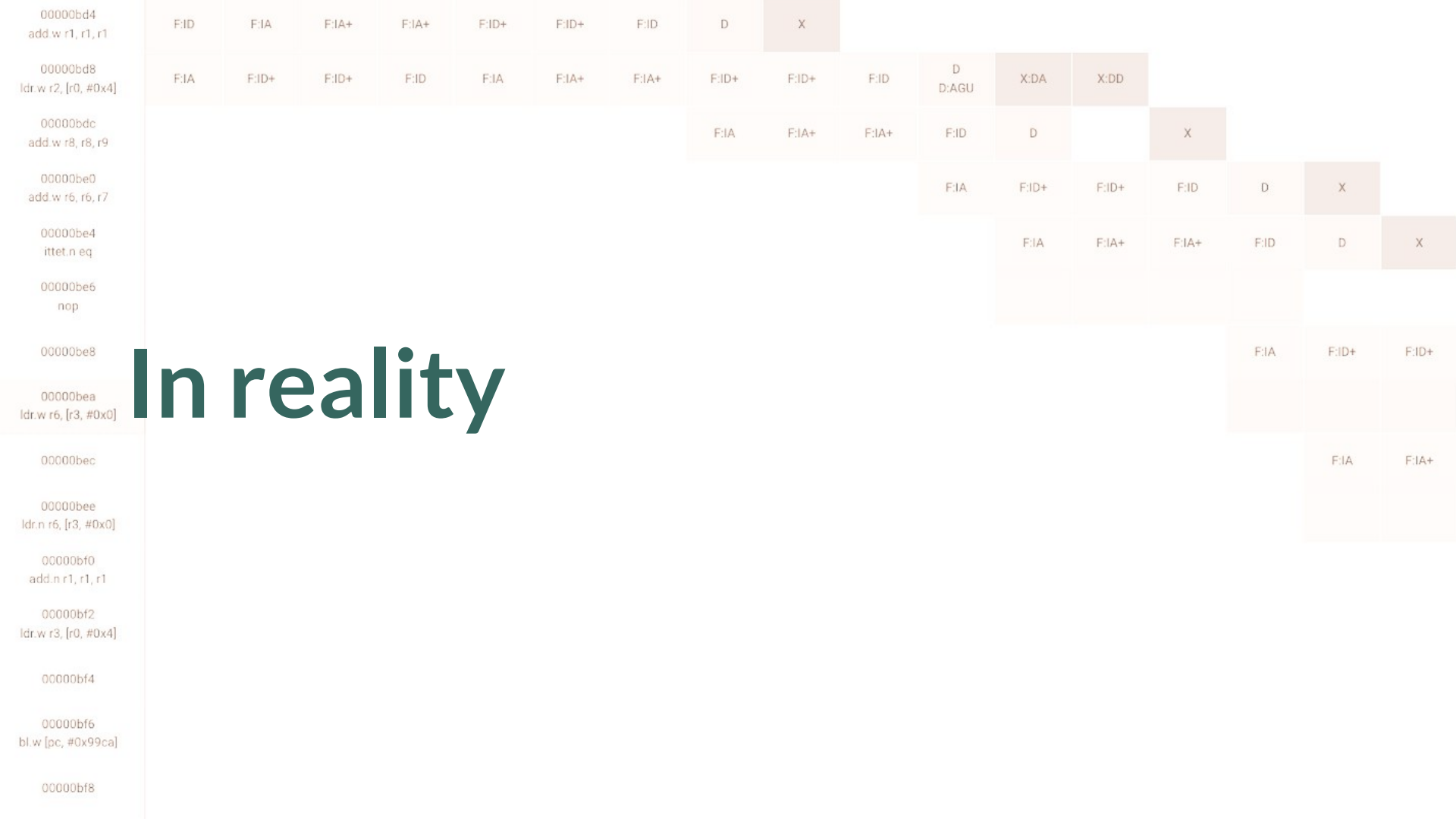
arithmetic-logical:



load-store:

```
ADD    R0, R1
MLA    R4, R3, R2
SUB    R6, R5
. . .
```




00000bd4
add.w r1, r1, r1
F:ID
F:IA
F:IA+
F:IA+
F:ID+
F:ID+
F:ID
D
X
00000bd8
ldr.w r2, [r0, #0x4]
F:IA
F:ID+
F:ID+
F:ID
F:IA
F:IA+
F:IA+
F:ID+
F:ID+
F:ID
D
D:AGU
X:DA
X:DD
00000bdc
add.w r8, r8, r9
F:IA
F:IA+
F:IA+
F:ID
D
X
00000be0
add.w r6, r6, r7
F:IA
F:ID+
F:ID+
F:ID
D
X
00000be4
ittet.n eq
F:IA
F:IA+
F:IA+
F:ID
D
X
00000be6
nop
00000be8
F:IA
F:ID+
F:ID+
00000bea
ldr.w r6, [r3, #0x0]
00000bec
F:IA
F:IA+
00000bee
ldr.n r6, [r3, #0x0]
00000bf0
add.n r1, r1, r1
00000bf2
ldr.w r3, [r0, #0x4]
00000bf4
00000bf6
bl.w [pc, #0x99ca]
00000bf8

In reality

cycles:

start( )

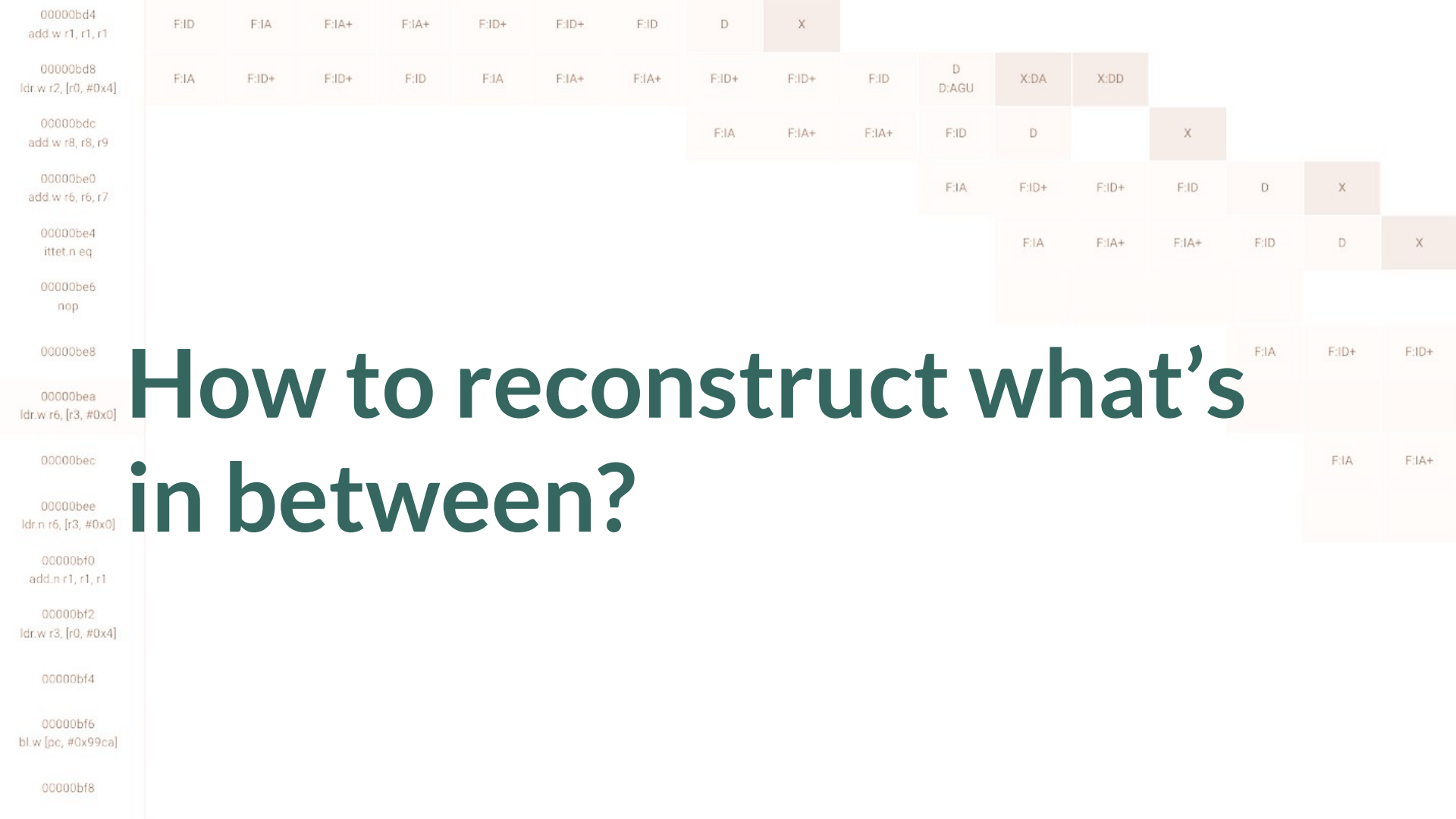
X:D

```
ADDS    R0, R1
UMULL   R1, R0, R0, R1
LDR     R2, [R1], #4
STR     R2, [R3]
BEQ     label
```

?

stop( )

X:D



How to reconstruct what's in between?

(`ADD RA, RB` from 0-wait-state GPRAM) = **1** cycle

F:D

Conclusion: X

(`ADD RA, RB` from 2-wait-state Flash) = **3** cycle

F:D F:D F:D

Conclusion: X X X ?

Let's use a ***gadget***

instruction	min. cycles
`ADD`	1
`UDIV`	2
`UMULL`	3
`UMULL`	4
`UDIV`	5
`UDIV`	6
`UDIV`	7
`...`	...

instruction	min. cycles
`ADD`	1

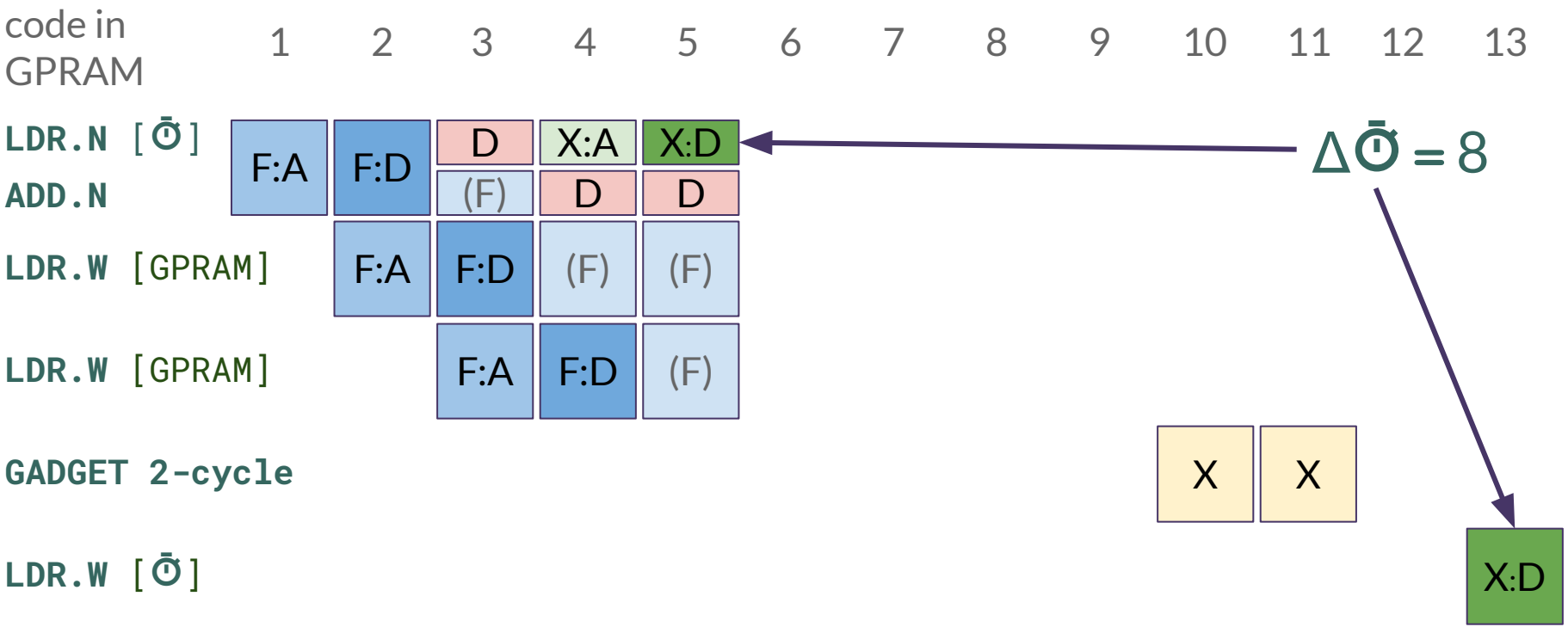
instructions	cycles in total
	6
	6
	6
`GADGET; ADD`	6
	6
	7
	8
	...

7 + 1 = 8 6 + 1 = 7 5 + 1 = 6 4 + 1 + 1 = 6

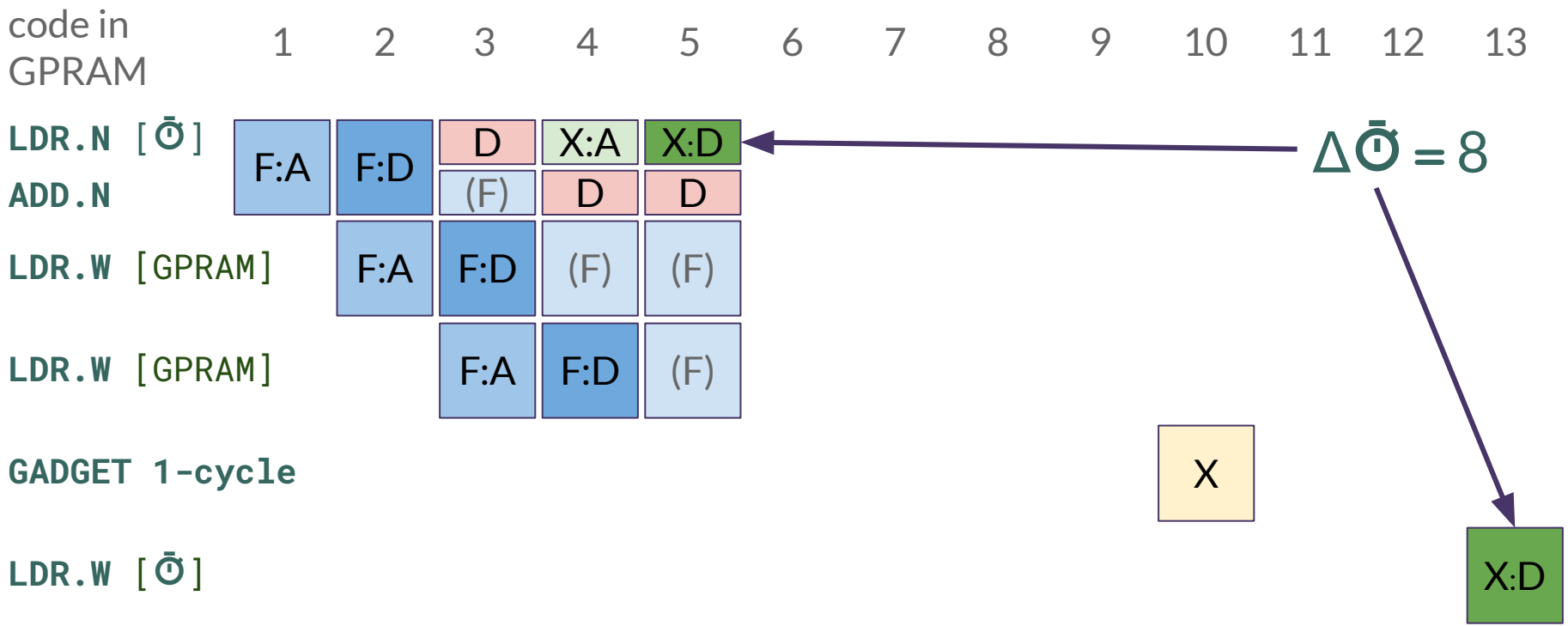
Conclusion: `ADD` X + something

What else is there besides X ? When?

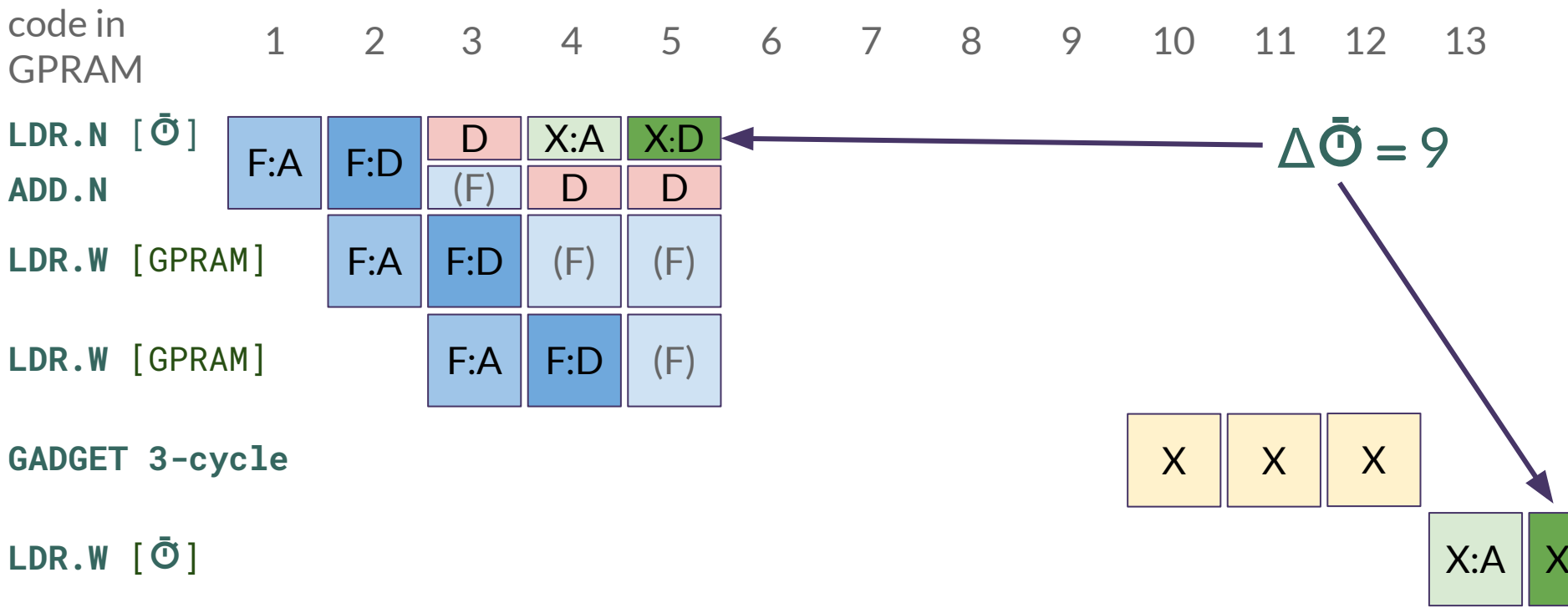
Let's use a ***Sudoku-like reconstruction***



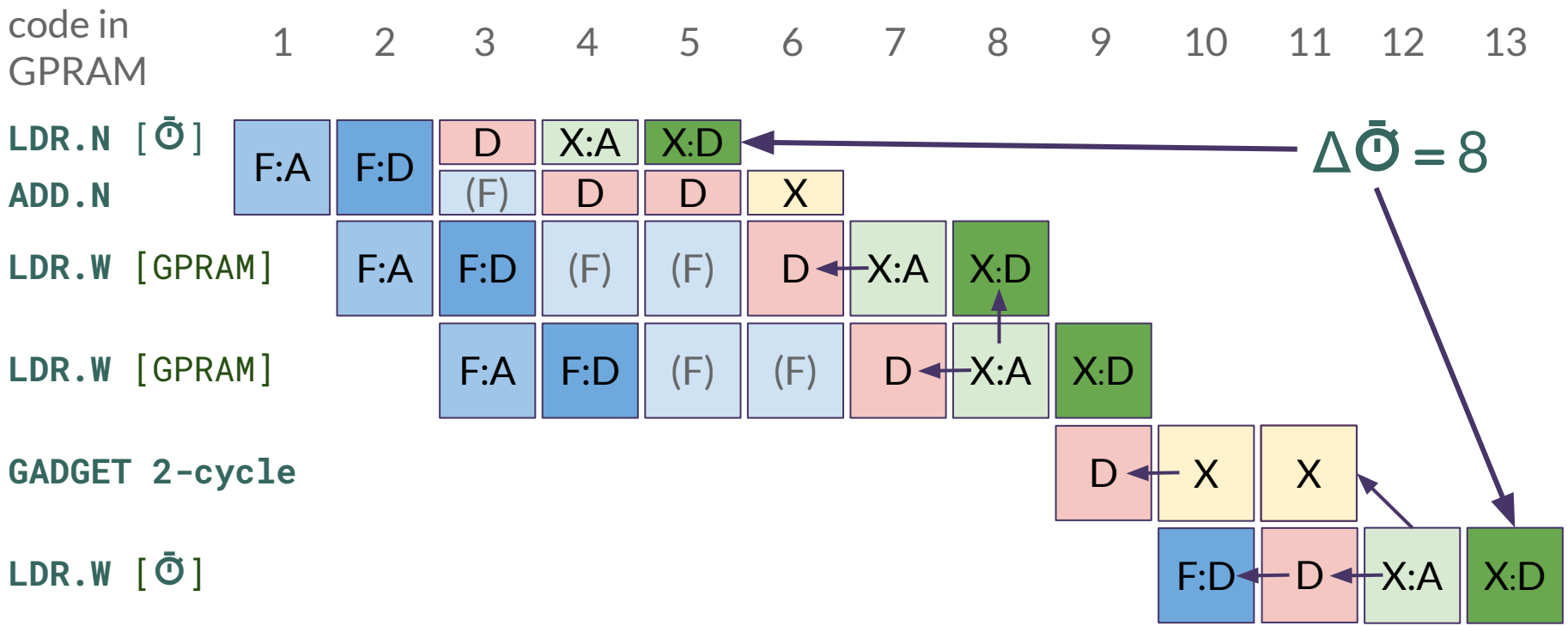
Let's use a ***Sudoku-like reconstruction***



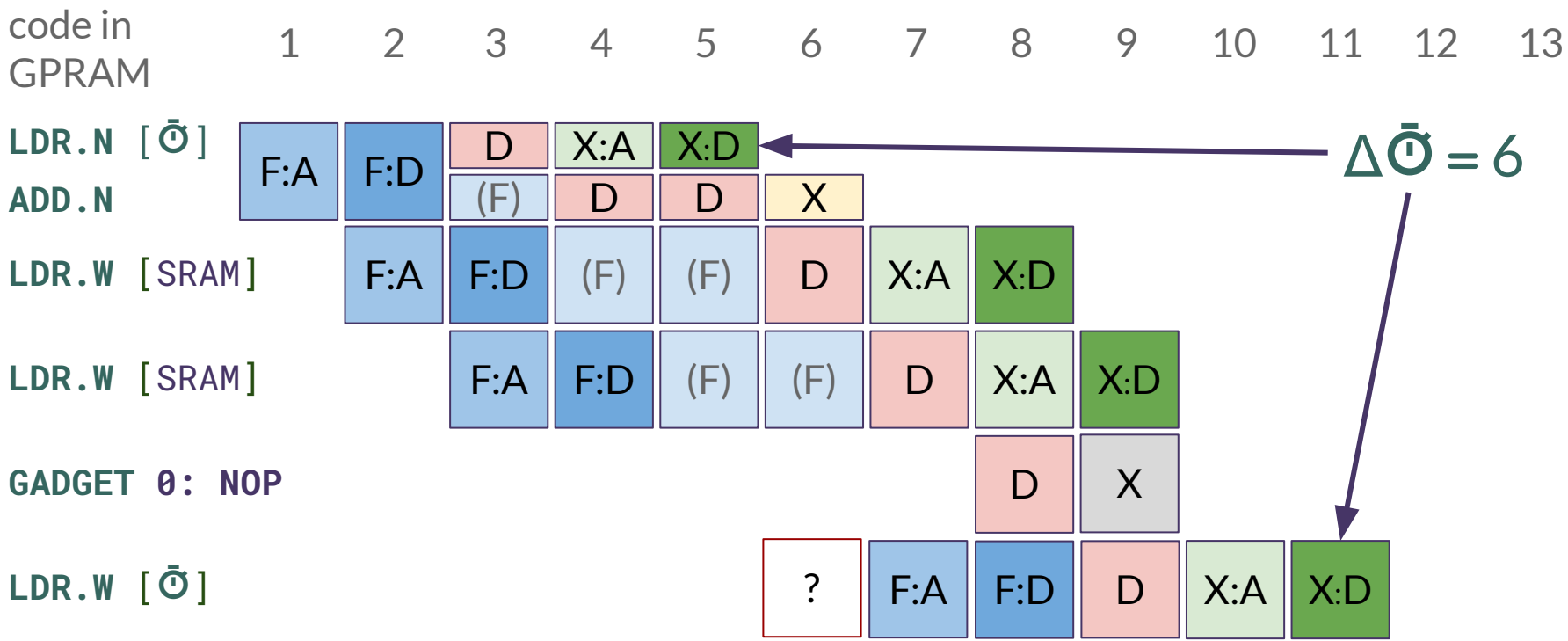
Let's use a ***Sudoku-like reconstruction***



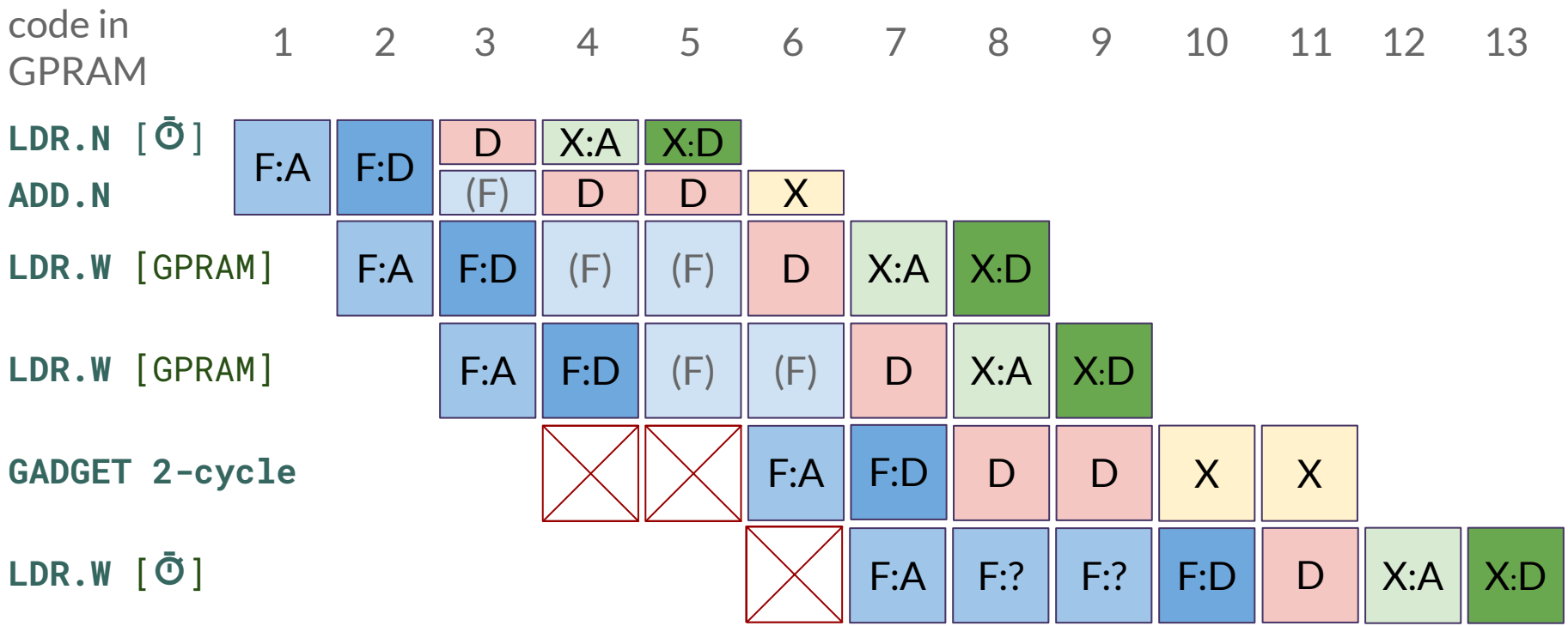
Let's use a ***Sudoku-like reconstruction***



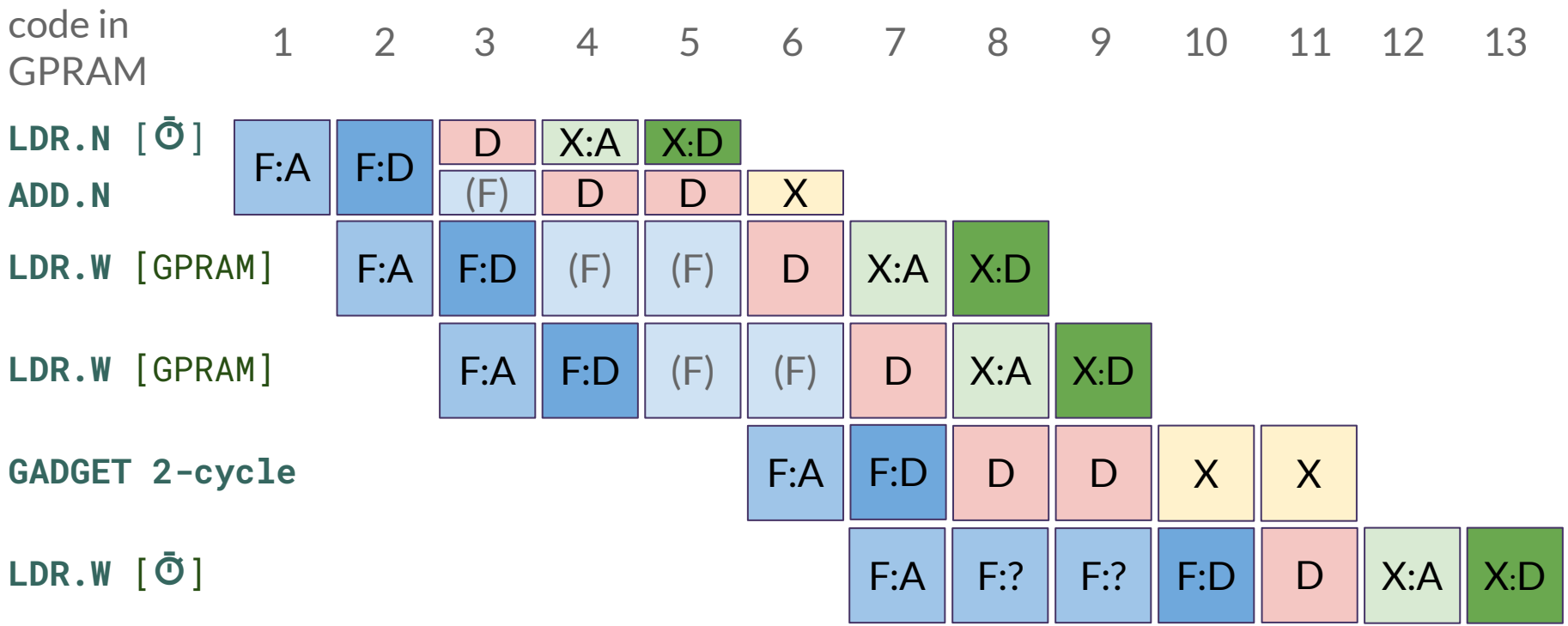
Let's use a ***Sudoku-like reconstruction***



Let's use a ***Sudoku-like reconstruction***



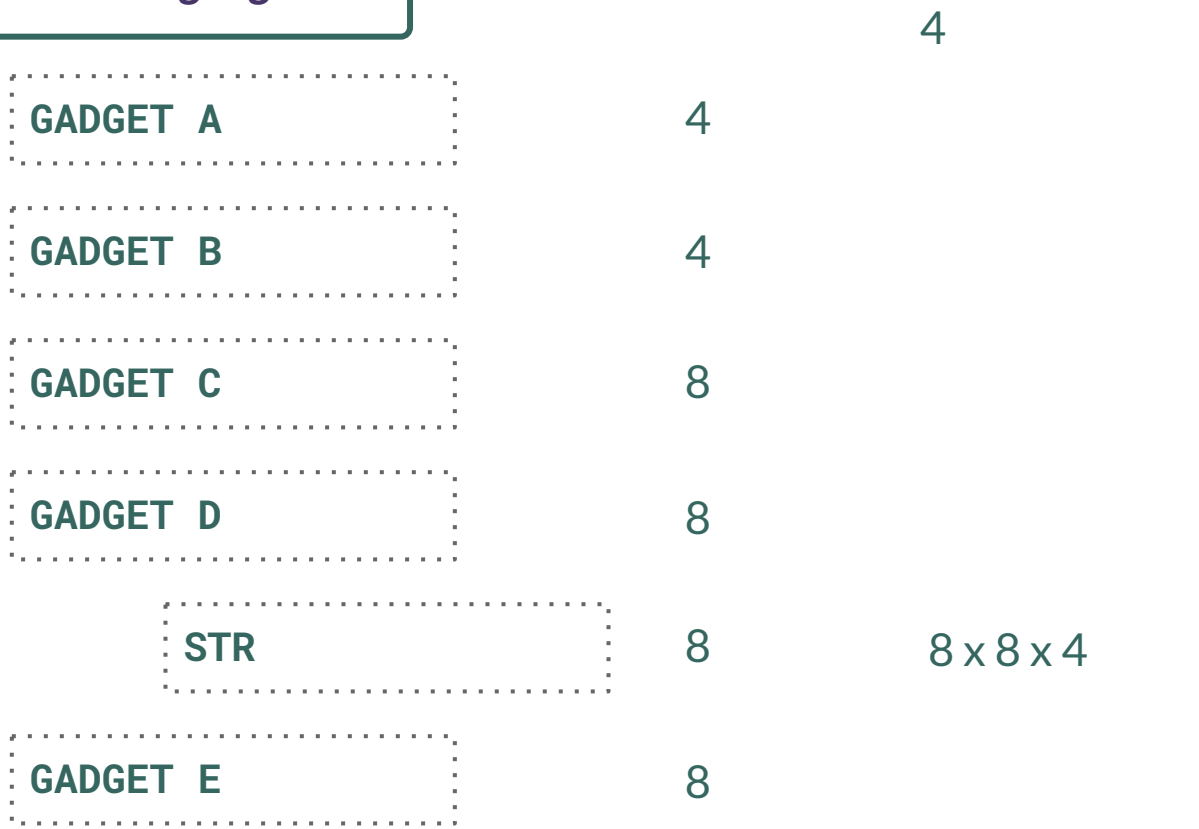
Let's use a ***Sudoku-like reconstruction***



How to validate interactions?

```
 B label

label:

 NOP.N

   UMULL R1, R0, R0, R1
                                LDR [R1], R0   ; from Flash
LDR [R1], R0   ; from GPRAM                                  LDR [R1], R0   ; from SRAM


                                                          STR R4, [R3], #4
                              STR.N [R2], R3
            STR [R2], R3                                  STR R4, [R3, #4]
                              STR.W [R2], R3
                                                          STR R4, [R3, #4]!


 LDR [R1], R0
```

Let's use ***extensive tests*** with ***gadgets***



4 x 4 x 4 x 8 x 8 x 8 x 8 x 8 x 8 x 4 = ~67M

Let's use an ***automatic characterization***

hardware	model	STR	Gadget A	Gadget B	
343	342	`.N [R, R]`	`NOP.N`	`ADD`	
434	434	`.N [R], R`	`–`	`UDIV`	
645	643	`.W [R, R]`	`NOP.N`	`UMULL`	...
454	454	`.W [R, #]!`	`–`	`UMULL`	
453	453	`.N [R], #`	`NOP.W`	`UDIV`	
...	...	...	...	...	

manual feature engineering

agglomerative clustering

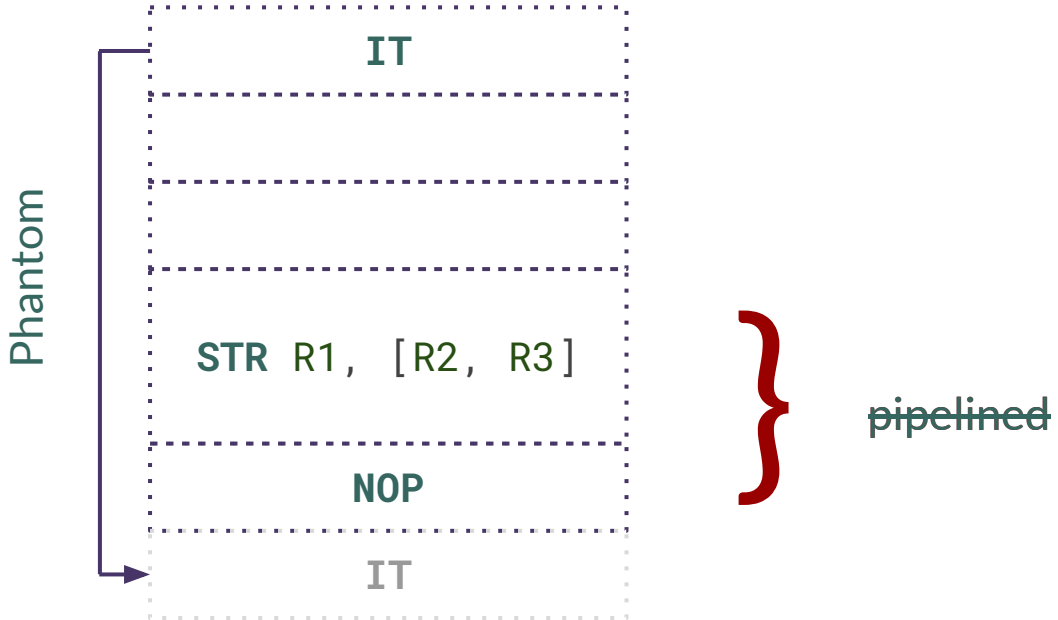
gradient boosted trees

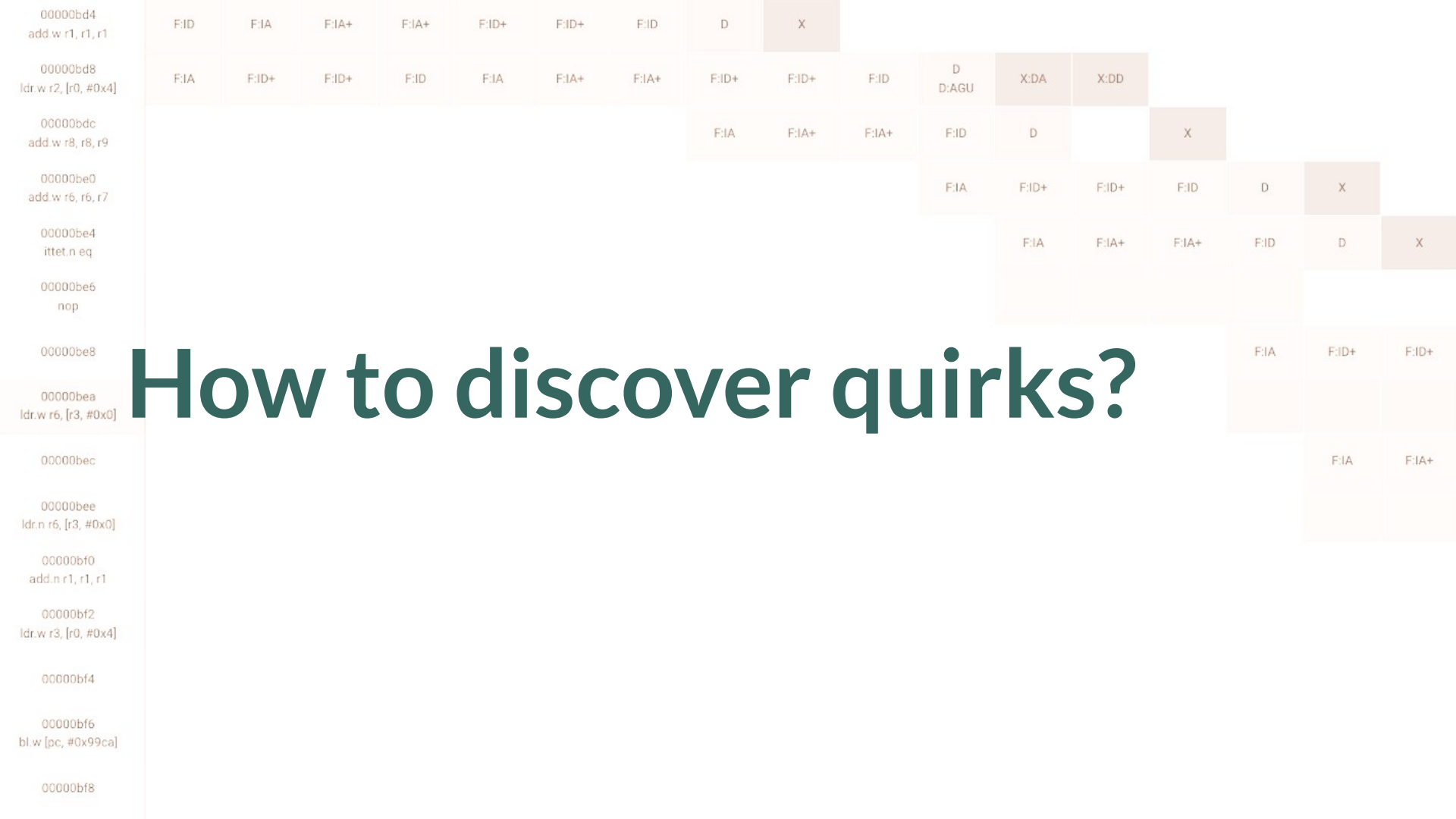
TE2Rules

logic- and data-based simplifications

Fetch's buffer:

empty





How to discover quirks?

Let's use ***errata notices***

Let's use a ***programs generator***

```
LDR   LR, [R1]
STR   R2, [R3]
BX    LR
```

+

```
UMULL R0,R1,R2,R3
MLA   R4,R5,R6,R0
MLA   R7,R8,R9,R0
```

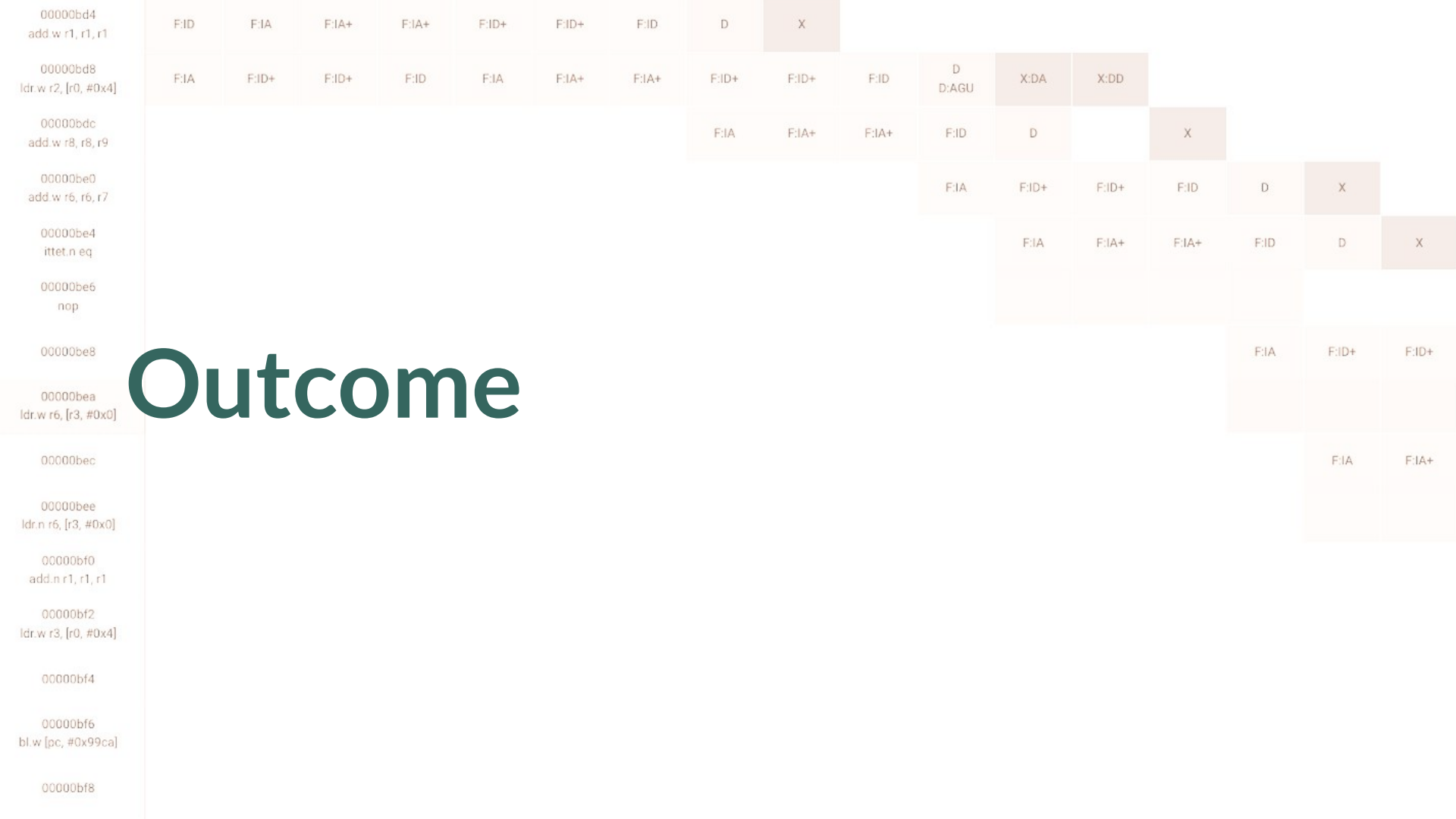
=

```
UMULL LR,R1,R2,R3
MLA   R4,R5,R6,LR
BX    LR
```

pipelined Execute but a *hack*

pipelined Execute but undocumented

New CPU bug!


00000bd4
add.w r1, r1, r1
F:ID
F:IA
F:IA+
F:IA+
F:ID+
F:ID+
F:ID
D
X
00000bd8
ldr.w r2, [r0, #0x4]
F:IA
F:ID+
F:ID+
F:ID
F:IA
F:IA+
F:IA+
F:ID+
F:ID+
F:ID
D
D:AGU
X:DA
X:DD
00000bdc
add.w r8, r8, r9
F:IA
F:IA+
F:IA+
F:ID
D
X
00000be0
add.w r6, r6, r7
F:IA
F:ID+
F:ID+
F:ID
D
X
00000be4
ittet.n eq
F:IA
F:IA+
F:IA+
F:ID
D
X
00000be6
nop
00000be8
F:IA
F:ID+
F:ID+
00000bea
ldr.w r6, [r3, #0x0]
00000bec
F:IA
F:IA+
00000bee
ldr.n r6, [r3, #0x0]
00000bf0
add.n r1, r1, r1
00000bf2
ldr.w r3, [r0, #0x4]
00000bf4
00000bf6
bl.w [pc, #0x99ca]
00000bf8

Outcome

CMEmu

```
_start:
    bl min_max

min_max:
     @ ...

    @ The example
    .align 4
    adds.w r0, r1
    umull.w r1, r0, r0, r1
    ldr.w r2, [r1], #4
    str.w r2, [r3]
    beq.w label
    add.w r4, r2
    add.w r5, r2
    add.w r4, r2
    label:
    add.w r4, r5

     @ ...
```

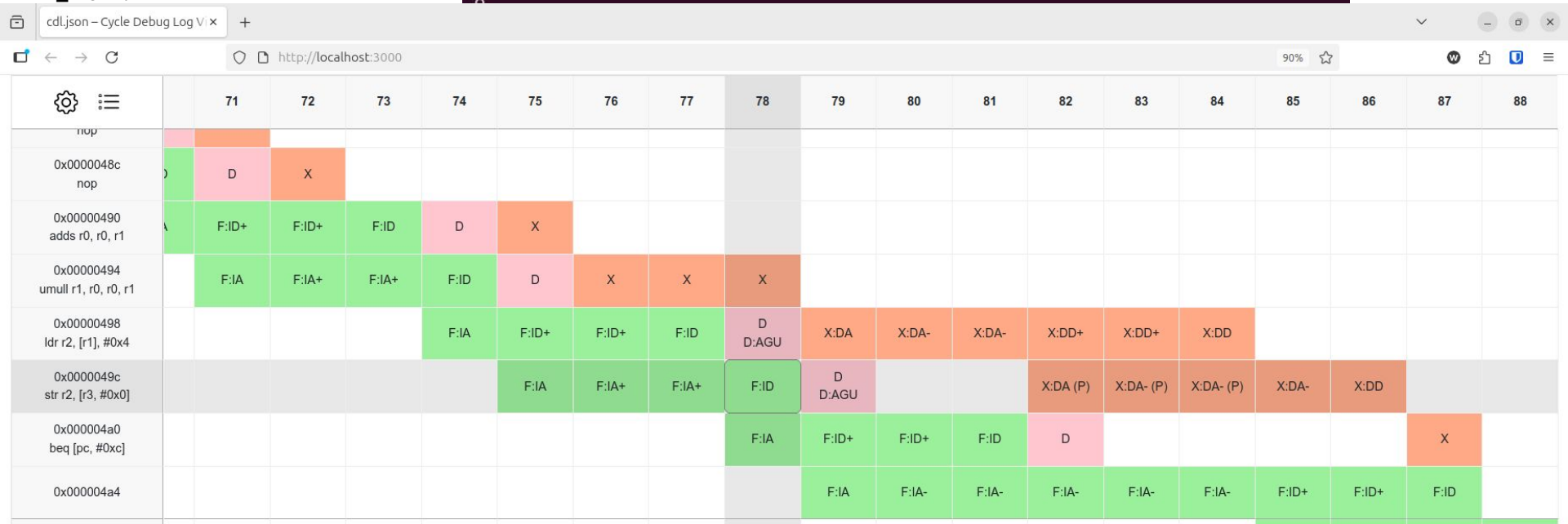
```
$ make
arm-none-eabi-gcc -mcpu=cortex-m3 -mthumb -mlittle-endian -fno-builtin-printf -f
no-builtin-sprintf -fno-builtin-snprintf -fno-builtin-vprintf -fno-exceptions -I
../../../../cmemu-elf-loader/cmemu-target/ -fshort-enums -fomit-frame-pointer -f
no-strict-aliasing -Wall -Werror -std=c99 -Og -ggdb3 -gz -c min_max.S -o min_max
.o
arm-none-eabi-gcc -mcpu=cortex-m3 -mthumb -mlittle-endian -fno-builtin-printf -f
no-builtin-sprintf -fno-builtin-snprintf -fno-builtin-vprintf -fno-exceptions -I
../../../../cmemu-elf-loader/cmemu-target/ -fshort-enums -fomit-frame-pointer -f
no-strict-aliasing -Wall -Werror -std=c99 -Og -ggdb3 -gz -T ../../../../cmemu-el
f-loader/cmemu-target/cc26xx.lds -Wl,--gc-sections -nostdlib -nostdinc -Wl,-\( m
in_max.o ../../../../cmemu-elf-loader/cmemu-target/cc26xx.o  -Wl,-\) -o min_max.
elf
rm min_max.o
$
```

```
$ cargo run --release -p cmemu --features cycle-debug-logger -- --cycle-debug-lo
g-file min_max.json min_max.elf
    Finished `release` profile [optimized] target(s) in 0.21s
     Running `target/release/cmemu --cycle-debug-log-file min_max.json min_max.e
lf`
$
```

CMEmu

```
_start:
    bl min_max

min_max:
```

```
$ make
arm-none-eabi-gcc -mcpu=cortex-m3 -mthumb -mlittle-endian -fno-builtin-printf -f
no-builtin-sprintf -fno-builtin-snprintf -fno-builtin-vprintf -fno-exceptions -I
../../../../cmemu-elf-loader/cmemu-target/ -fshort-enums -fomit-frame-pointer -f
no-strict-aliasing -Wall -Werror -std=c99 -Og -ggdb3 -gz -c min_max.S -o min_max
```



Cycle-exactness

Development:

- ☑ 500,000 hand-crafted tests
- ☑ 900,000 generated programs
- ☑ 500,000,000 extensive tests

Testing:

- ☑ Benchmarks:
  Embench, STREAM, BenchCouncil IoTBench, TI MSP430 microbenchmarks
- ☑ Real-world code:
  libc tests, Mbed TLS, OpenSSL, TweetNaCl, wolfSSL, built-in ROM
- ☑ Tools:
  CMGen, Examiner

In total:
10^{12} cycles

0 cycles error

Contributions, Next Steps & Future Goals

- ➔ Novel techniques
- ➔ CMEmu

- ➔ Integration with Cooja
- ➔ Adapting to more MCUs
- ➔ Supporting M0, M4, & co-processors

- ➔ Code optimization?
- ➔ Security validation?
- ➔ We're looking for collaborations!

Thank you! Time for Q&A

m.matraszek@mimuw.edu.pl

https://mimuw-distributed-systems-group.github.io/cmemu/

This work was supported by the National Science Center (NCN) in Poland under grant no. 2019/33/B/ST6/00448. The evaluation was partially carried out with the support of the Interdisciplinary Center for Mathematical and Computational Modeling at the University of Warsaw (ICM UW).