

Open CPU / SoC design, all the way up to Debian

lw	ra, 12(sp)	F	1	2	Rn	1	Ds	1	Is	1	2	3	Cm	1	2		
andi	a5, a2, 255	F	1	2	Rn	1	Ds	1	2	3	Is	Cm	1	2	3		
sw	a5, 20(sp)	F	1	2	Rn	1	Ds	1	Is	1	2	3	4	Cm	1		
bltz	ra, pc + 912	F	1	2	Rn	1	Ds	1	2	3	4	5	Is	1	Cm		
beqz	s0, pc + 2064	F	1	2	Rn	1	Ds	1	Is	1	Cm	1	2	3	4		
mv	t1, s0	F	1	2	Rn	1	Ds	1	2	Is	Cm	1	2	3	4		
j	pc + 0xc	F	1	2	Rn	1	Ds	1	Is	1	Cm	1	2	3	4		
lw	t2, 4(t1)	F	1	2	Rn	1	Ds	1	2	Is	1	2	3	Cm	1		
lw	s3, 12(sp)	F	1	2	Rn	1	Ds	1	Is	1	2	3	Cm	1	2		
lh	s2, 2(t2)	F	1	2	Rn	1	Ds	1	2	3	4	Is	1	2	3	Cm	
bne	s2, s3, pc + 4076	F	1	2	Rn	1	Ds	1	2	3	4	5	6	7	Is	1	Cm
lw	t1, 0(t1)	F	1	2	Rn	1	Ds	1	Is	1	2	3	Cm	1	2	3	4
beqz	t1, pc + 20	F	1	2	Rn	1	Ds	1	2	3	4	Is	1	Cm	1	2	
lw	t2, 4(t1)	F	1	2	Rn	1	Ds	1	2	3	Is	1	2	3	Cm		
lw	s3, 12(sp)	F	1	2	Rn	1	Ds	1	Is	1	2	3	Cm	1	2		
lh	s2, 2(t2)	F	1	2	Rn	1	Ds	1	2	3	4	5	Is	1			
bne	s2, s3, pc + 4076	F	1	2	Rn	1	Ds	1	2	3	4	5	6	7			
lw	t1, 0(t1)	F	1	2	Rn	1	Ds	1	2	Is	1	2	3				
beqz	t1, pc + 20	F	1	2	Rn	1	Ds	1	2	3	4	5	Is				
lw	t2, 4(t1)	F	1	2	Rn	1	Ds	1	2	3	4	Is					
lw	s3, 12(sp)	F	1	2	Rn	1	Ds	1	2	Is	1	2					
lh	s2, 2(t2)	F	1	2	Rn	1	Ds	1	2	3	4						

Applications: DOOM S... OpenTTD... VisualBo... Fri 23 Sep, 09:35 root



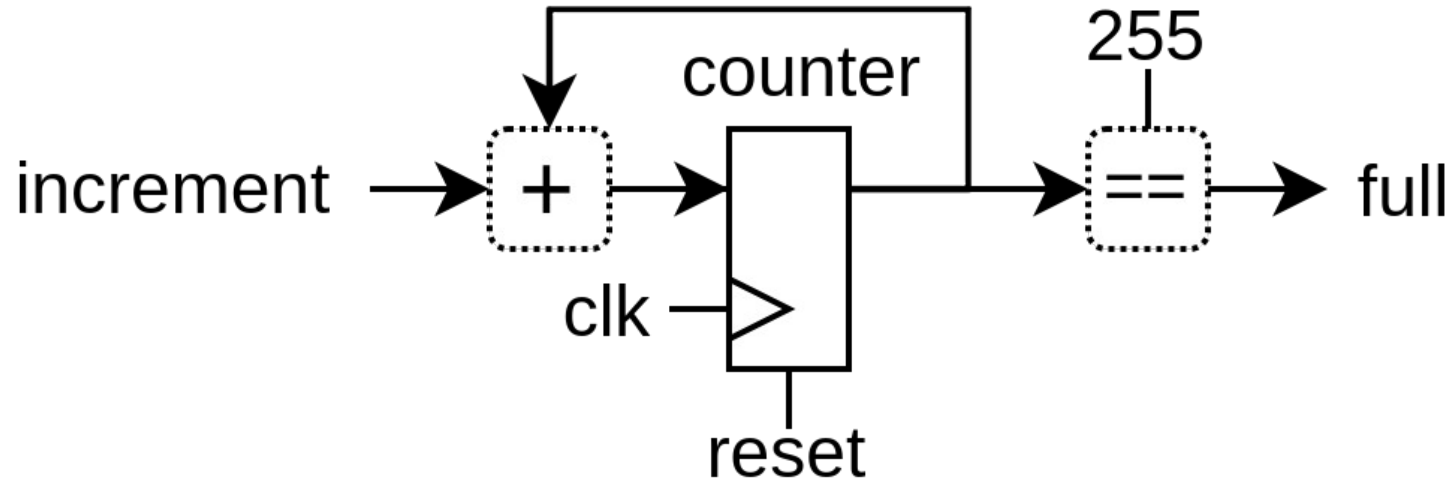
Background / whoami

- Dolu1990 on github
- Active on open/free project
 - SpinalHDL / VexRiscv / NaxRiscv
- Software / Hardware background
 - Without computer science degree
 - With Industrial system / Electronic degree

Introduction

- This talk will be based on NaxRiscv
 - Opensource <https://github.com/SpinalHDL/NaxRiscv>
 - RISC-V softcore
 - out of order, multi-issue, multi core, memory coherent
- What the talk will do
 - Share experience
 - Give tips / keywords

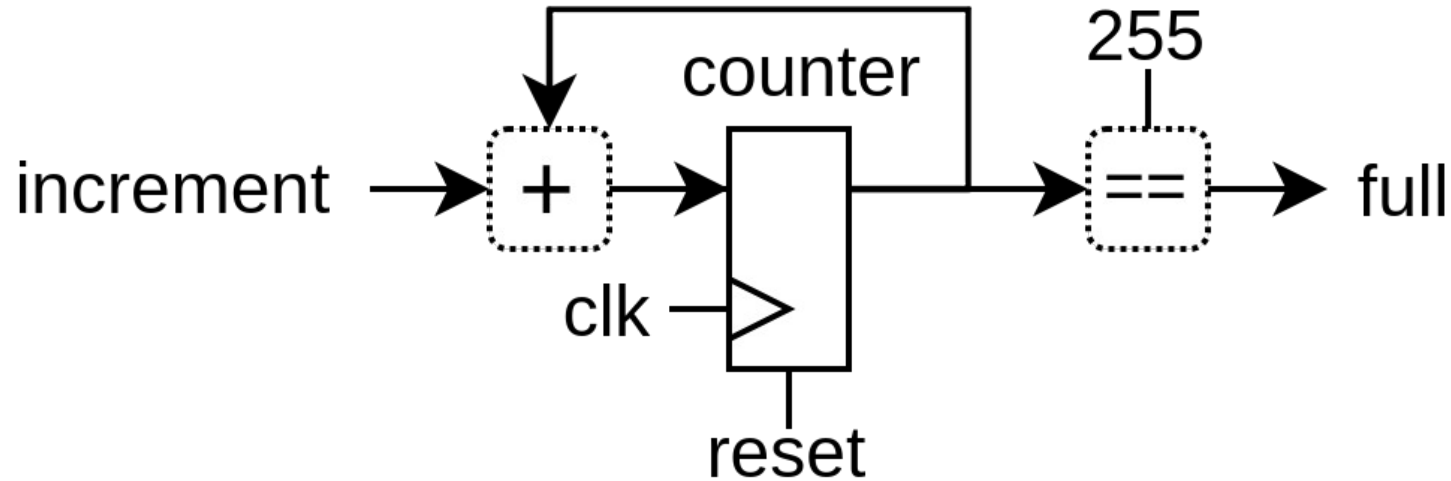
Digital Hardware description



- “There is no alternative” (VHDL / Verilog)
 - Scala based : SpinalHDL, Chisel
 - Python based : Migen, Amaranth
 - ...

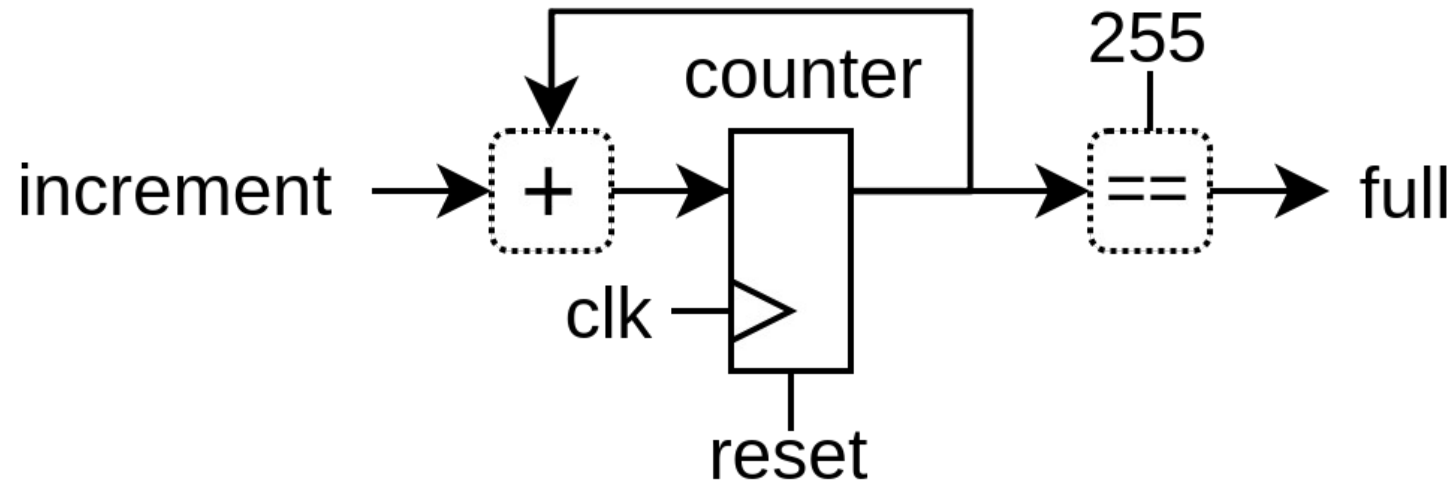


It can be pretty / explicit



```
class Timer extends Component {  
  val increment = in Bool()  
  val counter = Reg(UInt(8 bits)) init(0)  
  val full = counter === 255  
  
  when(increment) {  
    counter := counter + 1  
  }  
}
```

Using software to elaborate your hardware



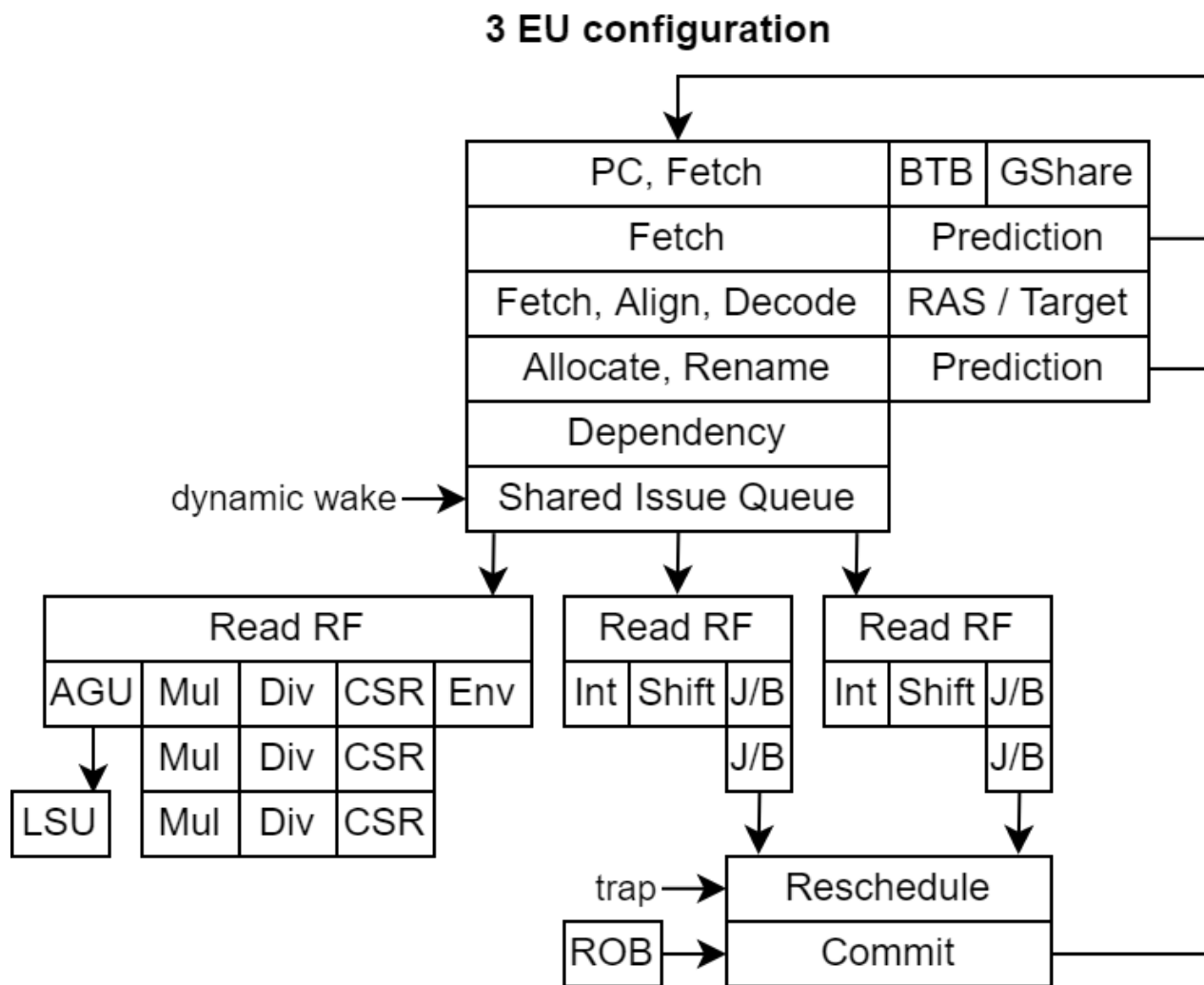
- Control flow : if / for
- Data structures : dynamic array / hash map / hash set
- Lambda function : reduce / fold / map / filter / ...
- OOP : class / software interface
- See <https://spinalhdl.github.io/NaxRiscv-Rtd/main/NaxRiscv/abstraction/index.html>

CPU instruction set

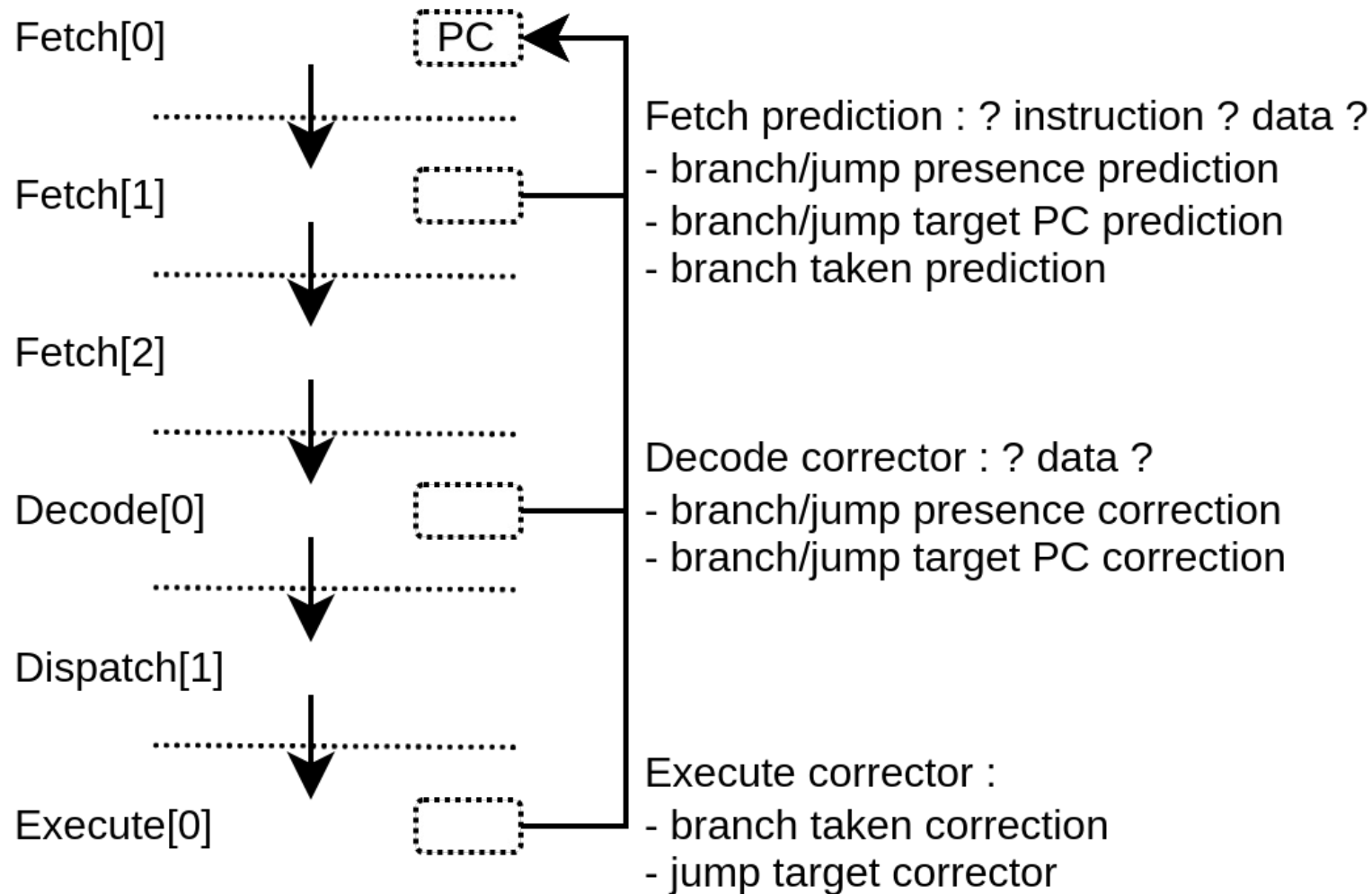


- A few open / free ISA (OpenRISC, RISC-V , ...)
- RISC-V
 - <https://riscv.org/technical/specifications/>
 - Secretly being embedded in ASICs
 - Bloat free (Mostly, from a FPGA perspective)
 - GCC / LLVM
 - Qemu support
 - Linux / Debian port
 - ...

Basics for an out of order CPU (NaxRiscv)



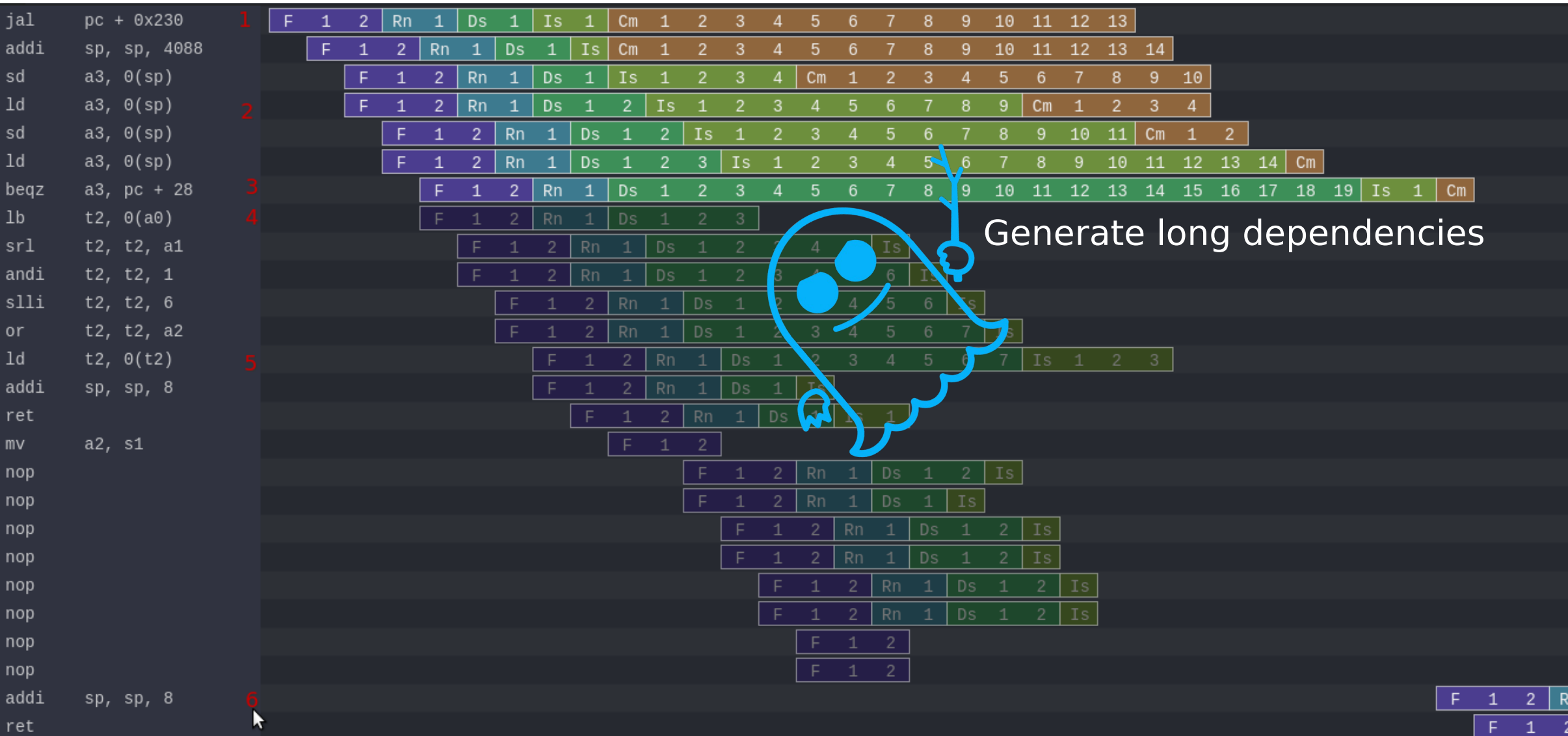
“Branch prediction”



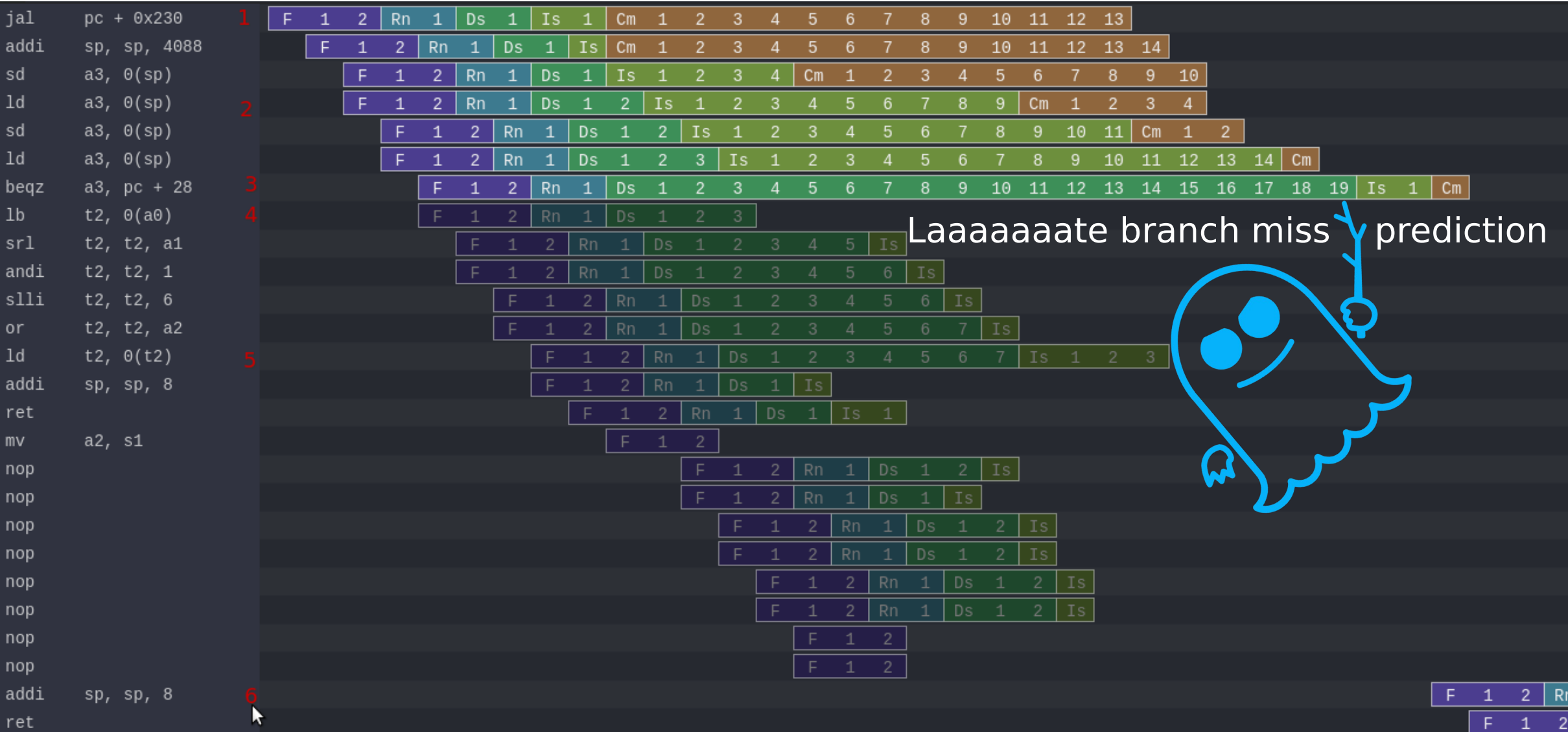
The diagram illustrates the execution of MIPS instructions in a pipeline with 6 stages. The instructions are: jal, addi, sd, ld, sd, ld, beqz, lb, srl, andi, slli, or, ld, addi, ret, mv, and seven nop instructions, followed by addi and ret. Each instruction is shown as a horizontal bar divided into segments representing pipeline stages (F, Rn, Ds, Is, Cm). The diagram illustrates the flow of instructions through the pipeline stages over time.

Instruction	Stage 1 (F)	Stage 2 (Rn)	Stage 3 (Ds)	Stage 4 (Is)	Stage 5 (Cm)
jal	1	2	1	1	1-13
addi	1	2	1	1	1-14
sd	1	2	1	1-4	1-10
ld	1	2	1	1-9	1-4
sd	1	2	1	1-11	1-2
ld	1	2	1	1-14	1
beqz	1	2	1	1-19	1
lb	1	2	1	1-3	
srl	1	2	1	1-5	1
andi	1	2	1	1-6	1
slli	1	2	1	1-6	1
or	1	2	1	1-7	1
ld	1	2	1	1-7	1-3
addi	1	2	1	1	1
ret	1	2	1	1	1
mv	1	2			
nop		1	2	1	1
nop		1	2	1	1
nop		1	2	1	1
nop		1	2	1	1
nop		1	2	1	1
nop		1	2		
nop		1	2		
addi		1	2		
ret		1	2		

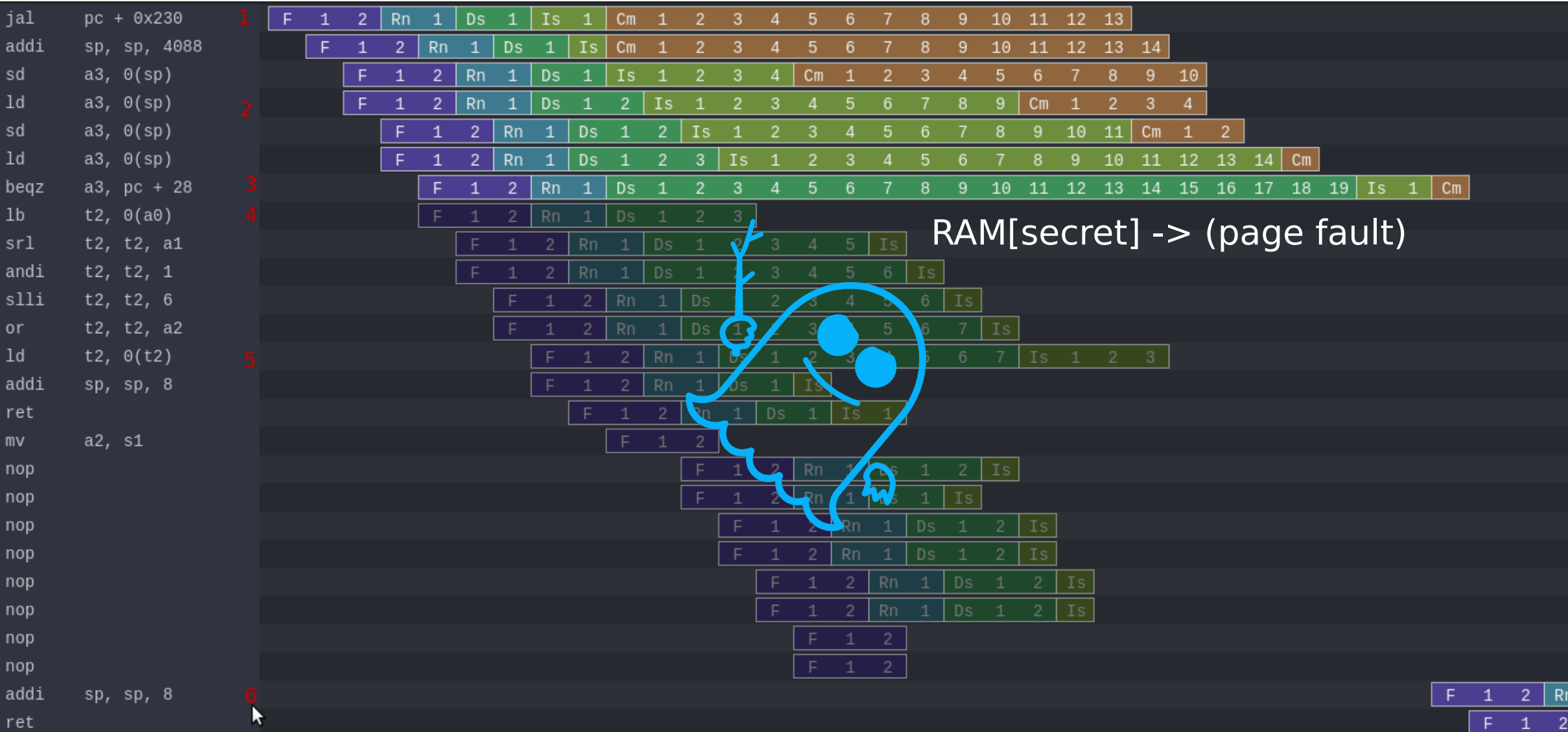
Secret leak



Secret leak



Secret leak

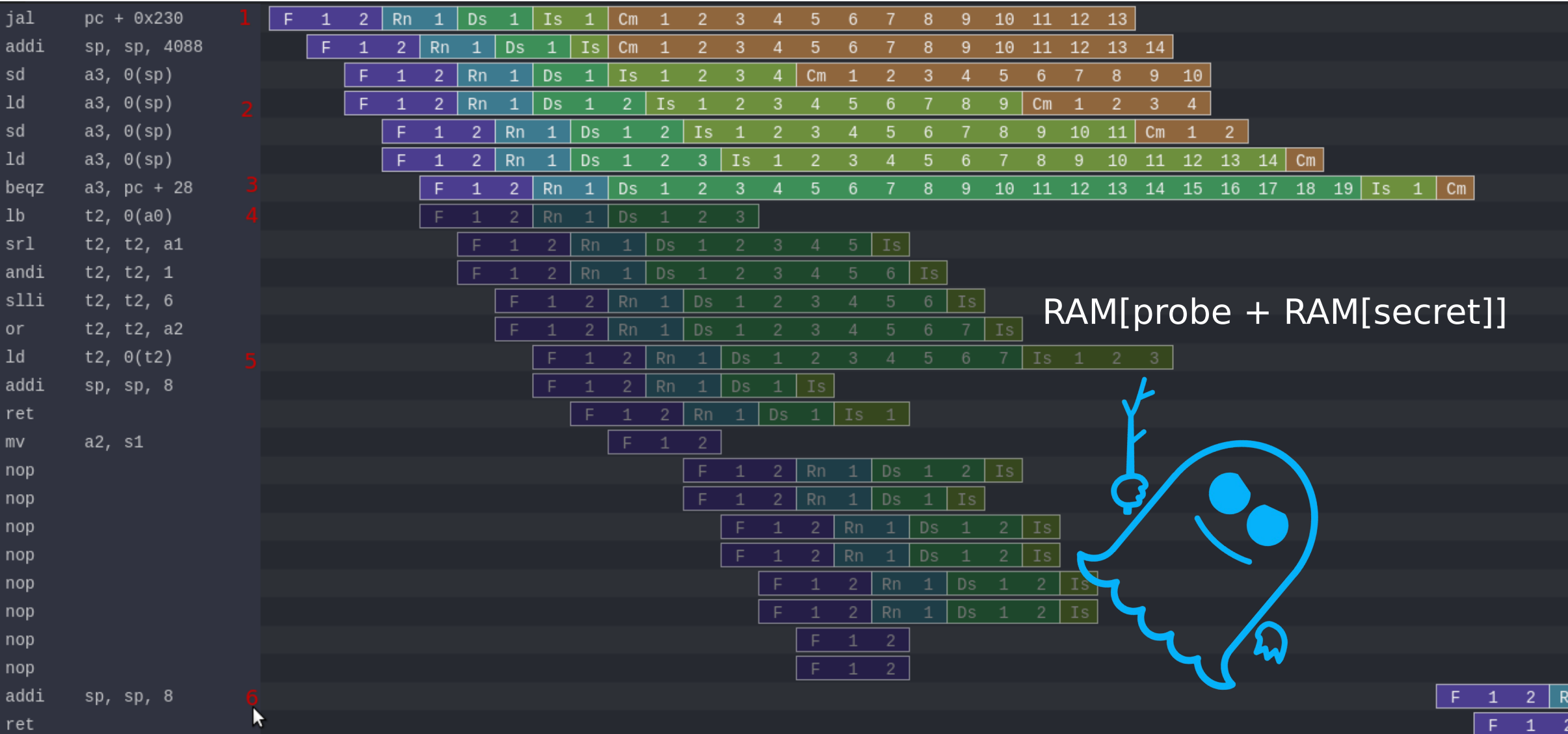


Secret leak

The diagram shows RISC-V assembly code with corresponding instruction format breakdowns. A blue brain icon is overlaid on the code, with a red arrow pointing to the instruction `ld t2, 0(t2)`. The text "probe + RAM[secret]" is written next to the brain icon.

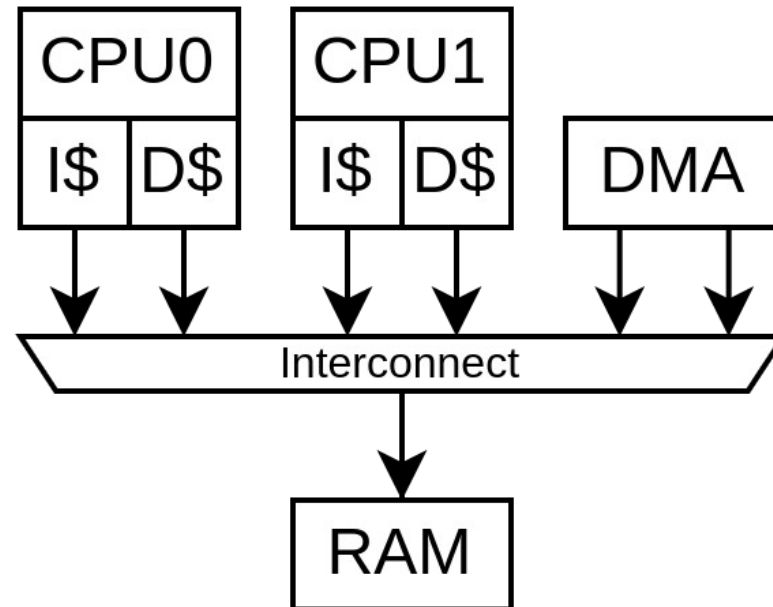
Instruction	Format Breakdown
<code>jal pc + 0x230</code>	F 1 2 Rn 1 Ds 1 Is 1 Cm 1 2 3 4 5 6 7 8 9 10 11 12 13
<code>addi sp, sp, 4088</code>	F 1 2 Rn 1 Ds 1 Is Cm 1 2 3 4 5 6 7 8 9 10 11 12 13 14
<code>sd a3, 0(sp)</code>	F 1 2 Rn 1 Ds 1 Is 1 2 3 4 Cm 1 2 3 4 5 6 7 8 9 10
<code>ld a3, 0(sp)</code>	F 1 2 Rn 1 Ds 1 2 Is 1 2 3 4 5 6 7 8 9 Cm 1 2 3 4
<code>sd a3, 0(sp)</code>	F 1 2 Rn 1 Ds 1 2 Is 1 2 3 4 5 6 7 8 9 10 11 Cm 1 2
<code>ld a3, 0(sp)</code>	F 1 2 Rn 1 Ds 1 2 3 Is 1 2 3 4 5 6 7 8 9 10 11 12 13 14 Cm
<code>beqz a3, pc + 28</code>	F 1 2 Rn 1 Ds 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 Is 1 Cm
<code>lb t2, 0(a0)</code>	F 1 2 Rn 1 Ds 1 2 3
<code>srl t2, t2, a1</code>	F 1 2 Rn 1 Ds 1 2 3 4 5 Is
<code>andi t2, t2, 1</code>	F 1 2 Rn 1 Ds 1 2 3 4 5 6 Is
<code>slli t2, t2, 6</code>	F 1 2 Rn 1 Ds 1 2 3 4 5 6 Is
<code>or t2, t2, a2</code>	F 1 2 Rn 1 Ds 1 2 3 4 5 6 Is
<code>ld t2, 0(t2)</code>	F 1 2 Rn 1 Ds 1 2 3 4 5 6 7 Is 1 2 3
<code>addi sp, sp, 8</code>	F 1 2 Rn 1 Ds 1 Is
<code>ret</code>	F 1 2 Rn 1 Ds 1 Is 1
<code>mv a2, s1</code>	F 1 2
<code>nop</code>	F 1 2 Rn 1 Ds 1 2 Is
<code>nop</code>	F 1 2 Rn 1 Ds 1 Is
<code>nop</code>	F 1 2 Rn 1 Ds 1 2 Is
<code>nop</code>	F 1 2 Rn 1 Ds 2 Is
<code>nop</code>	F 1 2 Rn 1 Ds 1 2 Is
<code>nop</code>	F 1 2 Rn 1 Ds 1 2 Is
<code>nop</code>	F 1 2
<code>nop</code>	F 1 2
<code>addi sp, sp, 8</code>	F 1 2 Rn 1 Ds 1 Is
<code>ret</code>	F 1 2 Rn 1 Ds 1 Is

Secret leak



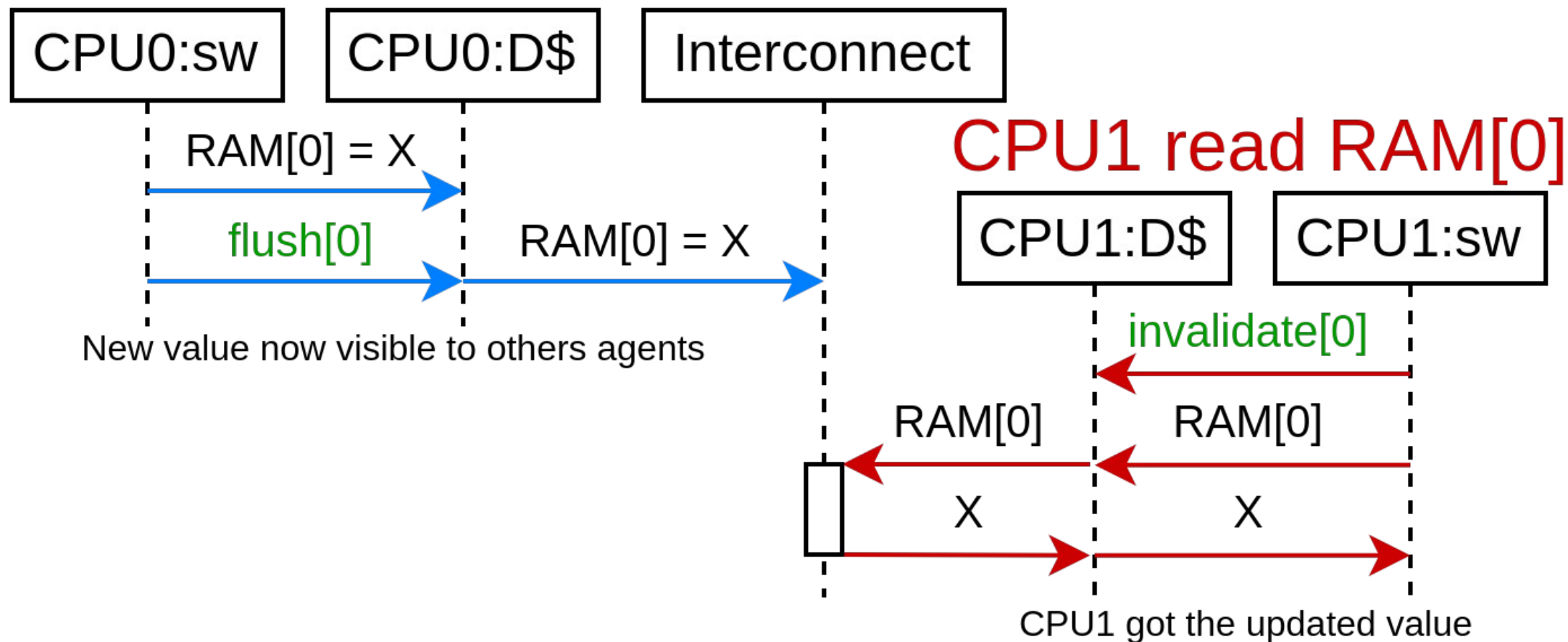
Multi core requirements (RISC-V linux)

- Hardware cache coherency
- Inter-CPU software interrupts



Software cache coherency

CPU0 write RAM[0]

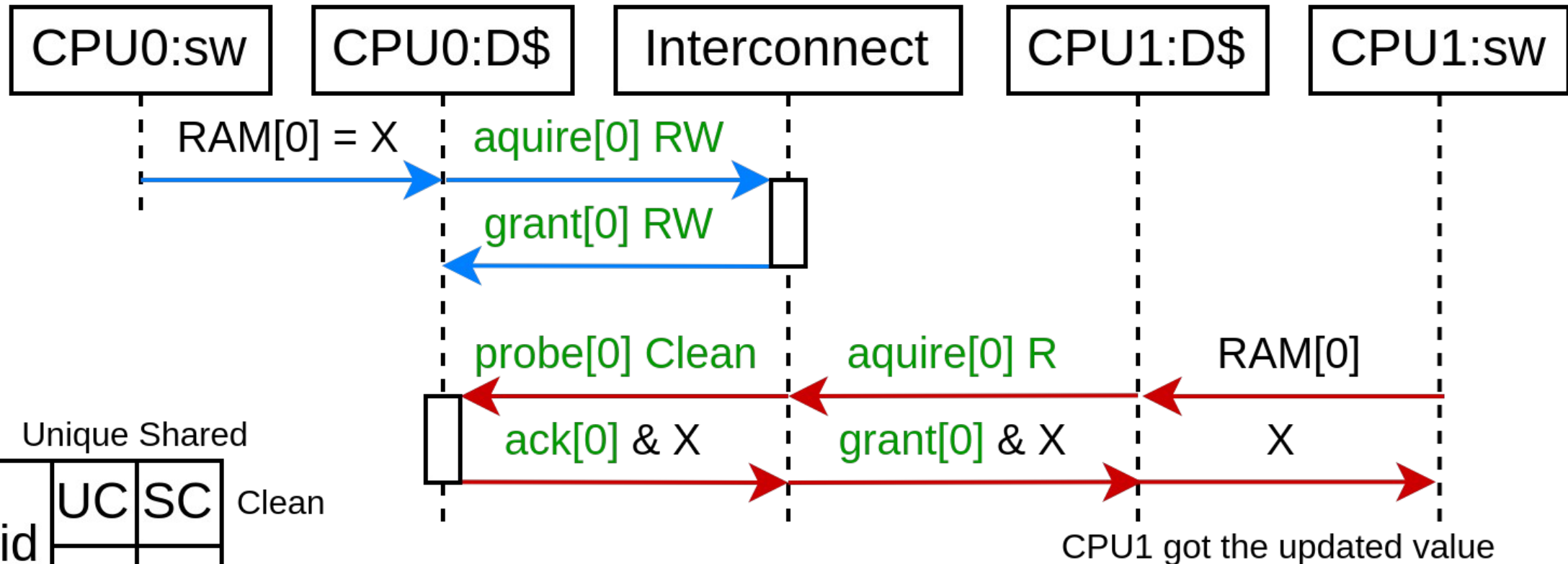


Hardware cache coherency

with memory block copy and permissions

CPU0 write RAM[0]

CPU1 read RAM[0]



Hardware cache coherency

And its side effects

```
volatile int array[2];
```

```
void thread_0() {  
    while(1) {  
        array[0] += 1;  
    }  
}
```

```
void thread_1() {  
    while(1) {  
        array[1] += 1;  
    }  
}
```

Same memory block (64 bytes): 18486 ps/iteration
Different memory blocks : 902 ps/iteration

(on a 2 years old CPU)

Cache coherency

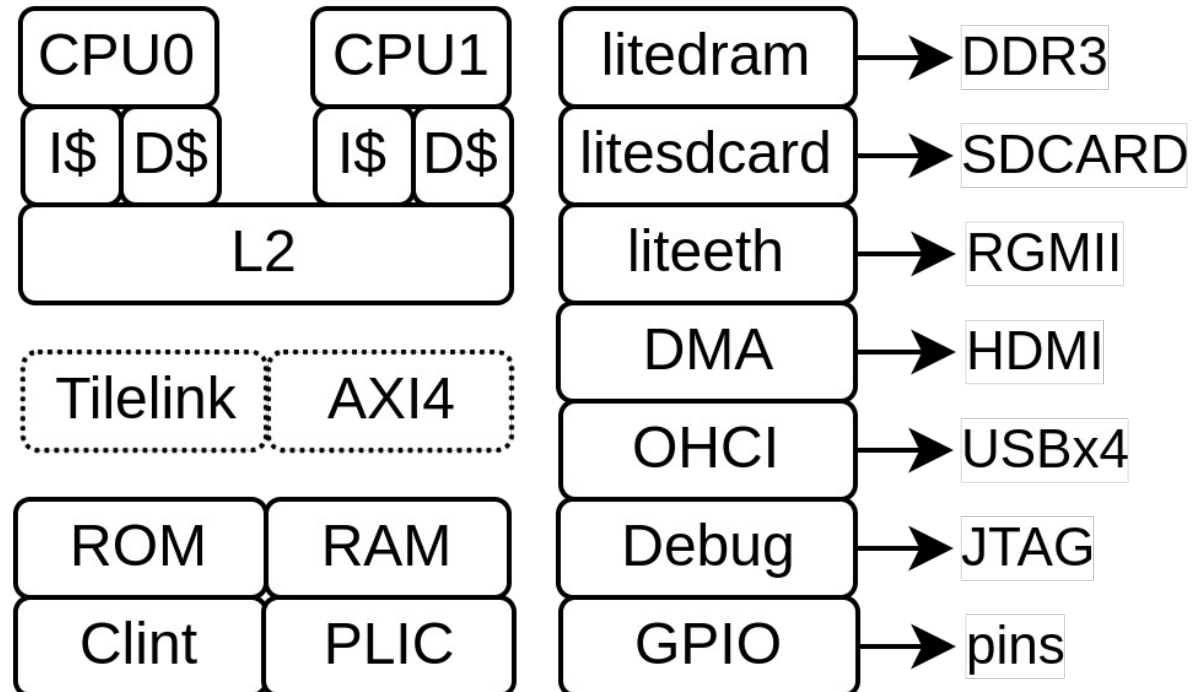
- Open specification : Tilelink
- Multiple open source implementations, ex :
 - <https://github.com/SpinalHDL/SpinalHDL/tree/dev/lib/src/main/scala/spinal/lib/bus/tilelink>
- Coherent L2 design
 - <https://github.com/SpinalHDL/SpinalHDL/blob/dev/lib/src/main/scala/spinal/lib/bus/tilelink/coherent/Cache.scala>
 - Exposes a lot of performance tricks
 - 1 pending transaction per 64 bytes block
 - 64 bytes per burst max
 - Keep bursts aligned

Memory / Peripherals / Deployment

- LiteX
 - Open-source SoC framework (in Python)
 - Integrate many peripherals
 - DDRx controller
 - SDCARD
 - Ethernet, USB host
 - SATA, PCIe, ...

Memory / Peripherals / Deployment

- `python3 -m litex_boards.targets.digilent_nexys_video --cpu-type=naxriscv --bus-standard axi-lite --with-video-framebuffer --with-coherent-dma --with-sdcard --with-ethernet --xlen=64 --scala-args='rvc=true,rvf=true,rvd=true,alu-count=2,decode-count=2' --with-jtag-tap --sys-clk-freq 100000000 --cpu-count 2 --build -load`
- + minor for USB / JTAG



USB host support

- USB host support =>
 - Keyboard / mouse / audio / flash drive / sdcard / uart / ethernet / bluetooth
- Open specification :
 - Open Host Controller Interface Specification for USB (OHCI)
 - USB 12 Mb/s
 - Up to 15 ports, 2 FPGA GPIO per port
- Open implementation :
 - https://spinalhdl.github.io/SpinalDoc-RTD/master/SpinalHDL/Libraries/Com/usb_ohci.html
 - <https://github.com/SpinalHDL/SpinalHDL/blob/dev/lib/src/main/scala/spinal/lib/com/usb/ohci/UsbOhci.scala>
 - https://github.com/litex-hub/pythondata-misc-usb_ohci/tree/master/pythondata_misc_usb_ohci/verilog

Synthesis / Place / Route

- Some open source tools for
 - Xilinx 7-Series
 - Lattice ice40 / ecp5
 - ...
- Synthesis : Yosys
- Place and route : Nextpnr
- ...

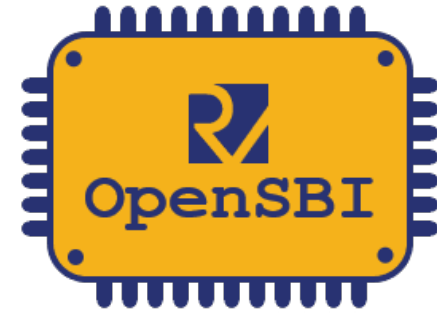
Running Linux



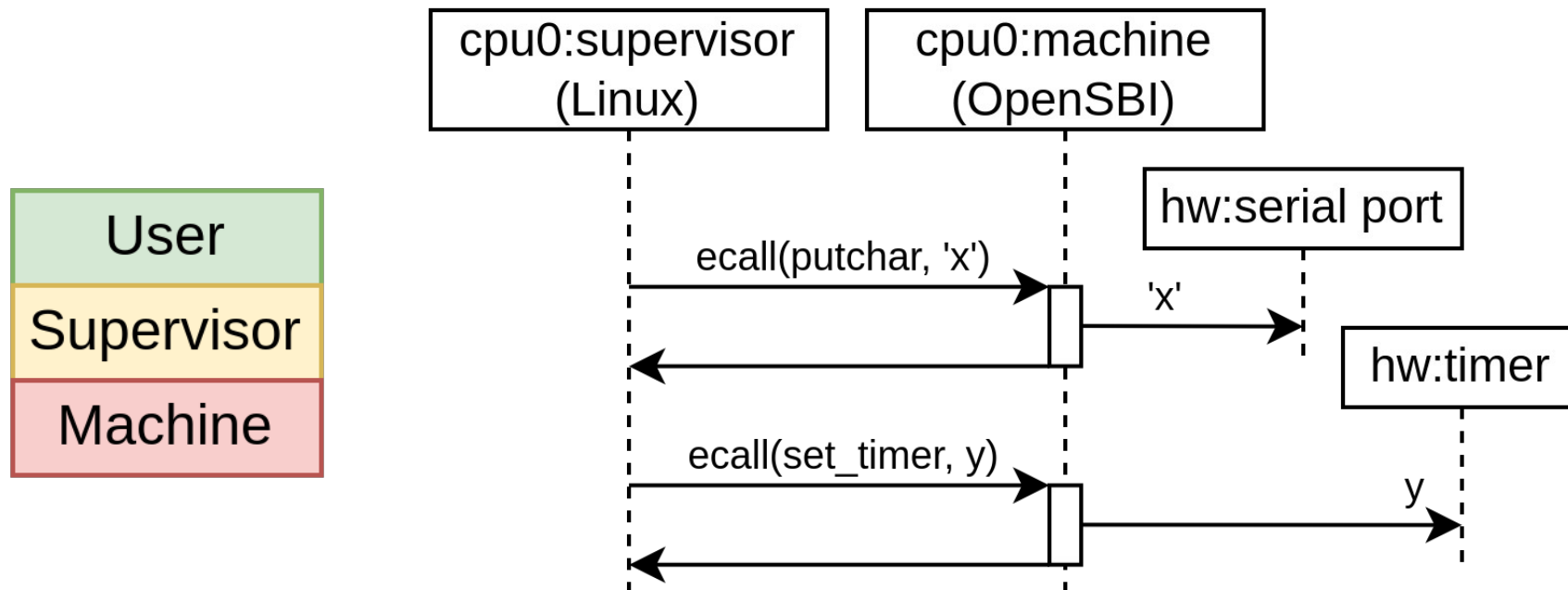
- Requirements on RISC-V
 - Hardware cache coherency (between CPUs + DMA)
 - Machine/Supervisor/User modes (riscv-privileged spec)
 - SBI (Supervisor Binary Interface, kinda like BIOS)
 - DTB (Device Tree Binary)



SBI (Supervisor Binary Interface)



- <https://github.com/riscv-non-isa/riscv-sbi-doc>
- <https://github.com/riscv-software-src/opensbi>



Running Debian

- Requirements on RISC-V
 - RV64IMAFDC
 - Linux kernel
 - Some storage (ex: SDCARD)
- Doc : https://github.com/SpinalHDL/NaxSoftware/tree/main/debian_litex
- Video : <https://x.com/dolu1990/status/1712848731349889108>

```
0[*] 0.6%] Tasks: 28, 18 thr, 61 kthr; 1 running
1[####*] 13.8%] Load average: 3.48 1.37 0.50
Mem[||||##*@@@@@@] 46.2M/472M] Uptime: 00:01:43
Swp[ 0K/4.00G]

[Main] [I/O]
  PID USER      PRI  NI  VIRT   RES   SHR S  CPU%MEM%  TIME+  Command
  512 root        20   0  3784   2776   2220 R   11.6  0.6   0:01.42 htop
  405 root        20   0 25000   7160   5568 S    0.6  1.5   0:00.89 /usr/sbin/cup
    1 root        20   0 16768 10876   7524 S    0.0  2.2   0:28.04 /sbin/init
```

524 M
Volum



Trash



File Sys



Home

OpenTTD 12.2

OPENTTD

New Game

Load Game

Play

Scenario

Game

AI/Game S

Check O

VisualBoy

LEVEL 9

HIGH 2

ghtmap

layer

DOOM Shareware - Chocolate

PICKED UP 4 SHOTGUN SHELLS.

49 100%

2 2 9

5 5 2

0%

BULL 48 200

SHELL 4 50

ROCK 0 50

CELL 0 300

AMMO

HEALTH

ARMS

ARMOR

Verification / Debugging

- May create rituals
- May exhaust your sanity
- May never end
- May crush your soul

Hello darkness my old friend

Debugging – example (a real one)

- Dual core Debian freeze (RCU error) after ~1h (360'000'000'000 cycles)
 - Impossible to reproduced in simulation (~40 days of runtime @ 100 Khz)
- Hope you get clues
 - Core was still alive on JTAG => OpenOCD
 - PC was in linux “_raw_spin_lock”
 - Forcing it to unlock => Everything back on track
- What it was :
 - CPU D\$ doing bad memory coherency
 - (releasing data permissions once again after being probed out)

Simulation



VERILATOR

- A few open source simulation tools :
 - GHDL, IVerilog, Verilator, ...
- Based on Verilator (~150-200 Khz on a 4 years old CPU)
- Lock-step checks against Spike / RVLS
- Konata traces

0x80006858: slli	a2, a4, 2	F	1	2	Rn	1	Ds	1	2	Is	Cm	1	2	3	4	5	6	7		
0x8000685c: addi	a4, a2, 64	F	1	2	Rn	1	Ds	1	2	3	Is	Cm	1	2	3	4	5	6		
0x80006860: addi	a2, sp, 16	F	1	2	Rn	1	Ds	1	2	Is	Cm	1	2	3	4	5	6	7		
0x80006864: add	a4, a4, a2	F	1	2	Rn	1	Ds	1	2	3	Is	Cm	1	2	3	4	5	6		
0x80006868: lw	a2, 4032(a4)	F	1	2	Rn	1	Ds	1	2	3	Is	1	2	3	Cm	1	2	3		
0x8000686c: addiw	a2, a2, 1	F	1	2	Rn	1	Ds	1	2	3	4	5	6	Is	Cm	1	2	3		
0x80006870: sw	a2, -64(a4)	F	1	2	Rn	1	Ds	1	2	3	Is	1	2	3	4	5	6	7	Cm	
0x80006874: bnez	a6, pc + 4004	F	1	2	Rn	1	Ds	1	2	Is	1	Cm	1	2	3	4	5	6	7	
0x80006818: lbu	a4, 1(a7)	F	1	2	Rn	1	Ds	1	2	3	Is	1	2	3	Cm	1	2	3	4	
0x8000681c: addi	a2, a7, 1	F	1	2	Rn	1	Ds	1	2	Is	Cm	1	2	3	4	5	6	7	8	
0x80006820: beq	a6, t5, pc + 1148	F	1	2	Rn	1	Ds	1	Is	1	Cm	1	2	3	4	5	6	7	8	
0x80006824: addiw	t1, a6, 4048	F	1	2	Rn	1	Ds	1	2	Is	Cm	1	2	3	4	5	6	7	8	
0x80006828: andi	t1, t1, 255	F	1	2	Rn	1	Ds	1	2	Is	Cm	1	2	3	4	5	6	7	8	
0x8000682c: addiw	t4, t4, 1	F	1	2	Rn	1	Ds	1	2	3	Is	Cm	1	2	3	4	5	6	7	
0x80006830: bgeu	t3, t1, pc + 628	F	1	2	Rn	1	Ds	1	2	Is	1	Cm	1	2	3	4	5	6	7	
0x80006aa4: beqz	a4, pc + 784	F	1	2	Rn	1	Ds	1	2	Is	1	Cm	1	2	3	4	5	6	7	
0x80006aa8: beq	a4, t5, pc + 800	F	1	2	Rn	1	Ds	1	2	Is	1	Cm	1	2	3	4	5	6	7	
0x80006aac: mv	a6, a4	F	1	2	Rn	1	Ds	1	Is	1	Cm	1	2	3	4	5	6	7	8	9
0x80006ab0: addi	a2, a2, 1	F	1	2	Rn	1	Ds	1	2	Is	Cm	1	2	3	4	5	6	7	8	
0x80006ab4: lbu	a4, 0(a2)	F	1	2	Rn	1	Ds	1	2	3	Is	1	2	3	Cm	1	2	3	4	
0x80006ab8: beq	a6, a3, pc + 44	F	1	2	Rn	1	Ds	1	2	Is	1	Cm	1	2	3	4	5	6	7	
0x80006abc: addiw	a6, a6, 4048	F	1	2	Rn	1	Ds	1	Is	1	Cm	1	2	3	4	5	6	7	8	9
0x80006ac0: andi	a7, a6, 255	F	1	2	Rn	1	Ds	1	2	Is	Cm	1	2	3	4	5	6	7	8	9

0xAA55

- I hope you had a good time
- If you are looking for open-source project funding :
 - NLnet Foundation (they helped me a lot)

Speculative execution

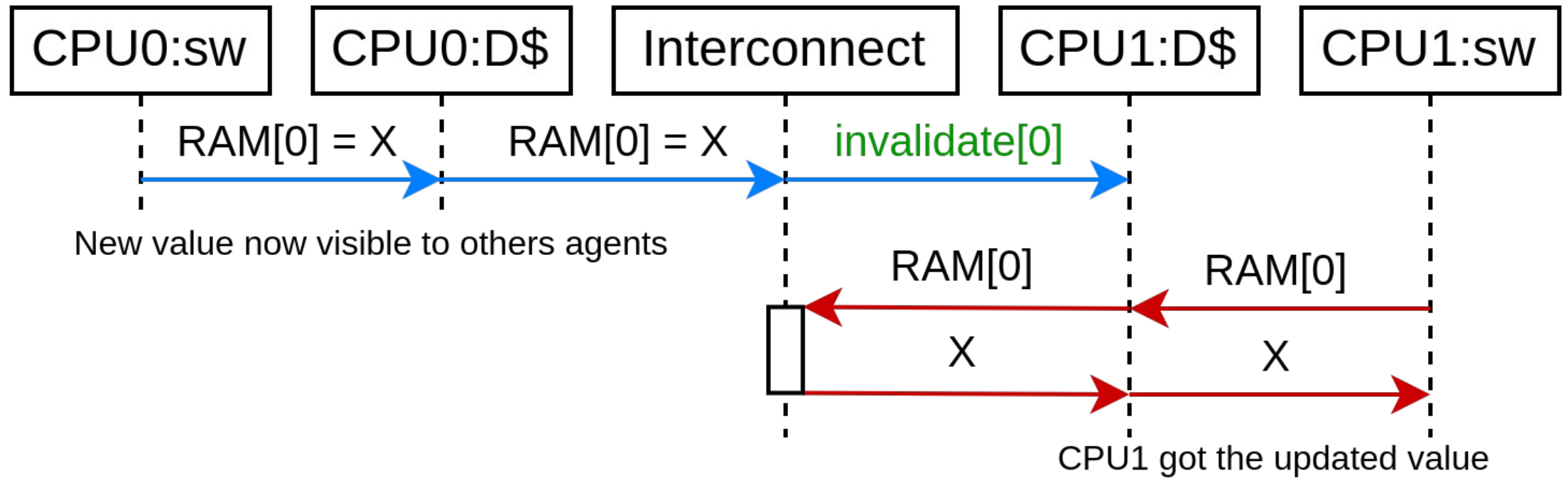
- Execute as much as you can in advance
- Bad speculation → no side effect on the **ISA states**
 - x1, x2, .. , x31
 - CSR
 - Global memory (RAM)
- But what's about all the others **hardware states**
 - Branch predictors (BTB, RAS, GShare)
 - I\$ D\$ tags (is address X in the cache or not)
 - Those are easy to probe

Hardware cache coherency

without memory block ownership

CPU0 write RAM[0]

CPU1 read RAM[0]



Hardware cache coherency

with memory block ownership

- Caches have to acquire memory block permissions
 - read [write] permissions
 - 64 bytes per block (typically), aligned
- Cache line states

- Valid / Invalid
- Clean / Dirty
- Unique / Shared

		Unique	Shared		
Invalid		UC	SC	Clean	
		UD	SD	Dirty	

Boot sequence

- FPGA bitstream
- Litex bios
- OpenSBI
- [Uboot]
- Linux
- Buildroot / Debian / ...