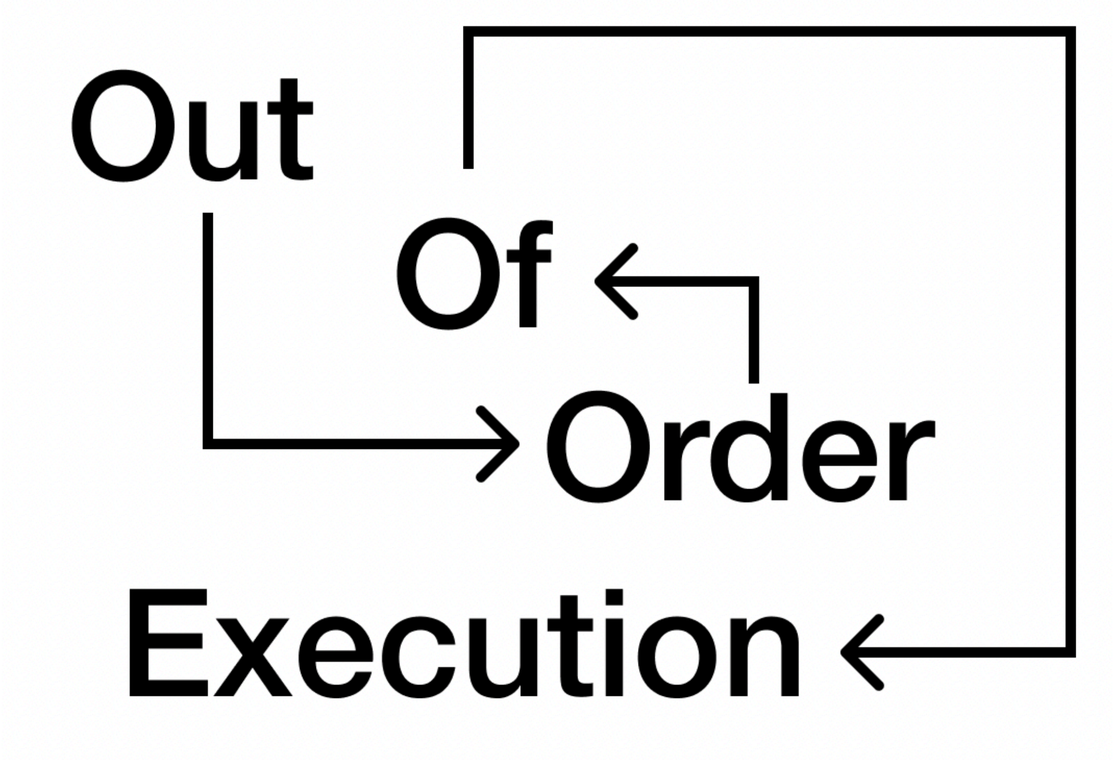
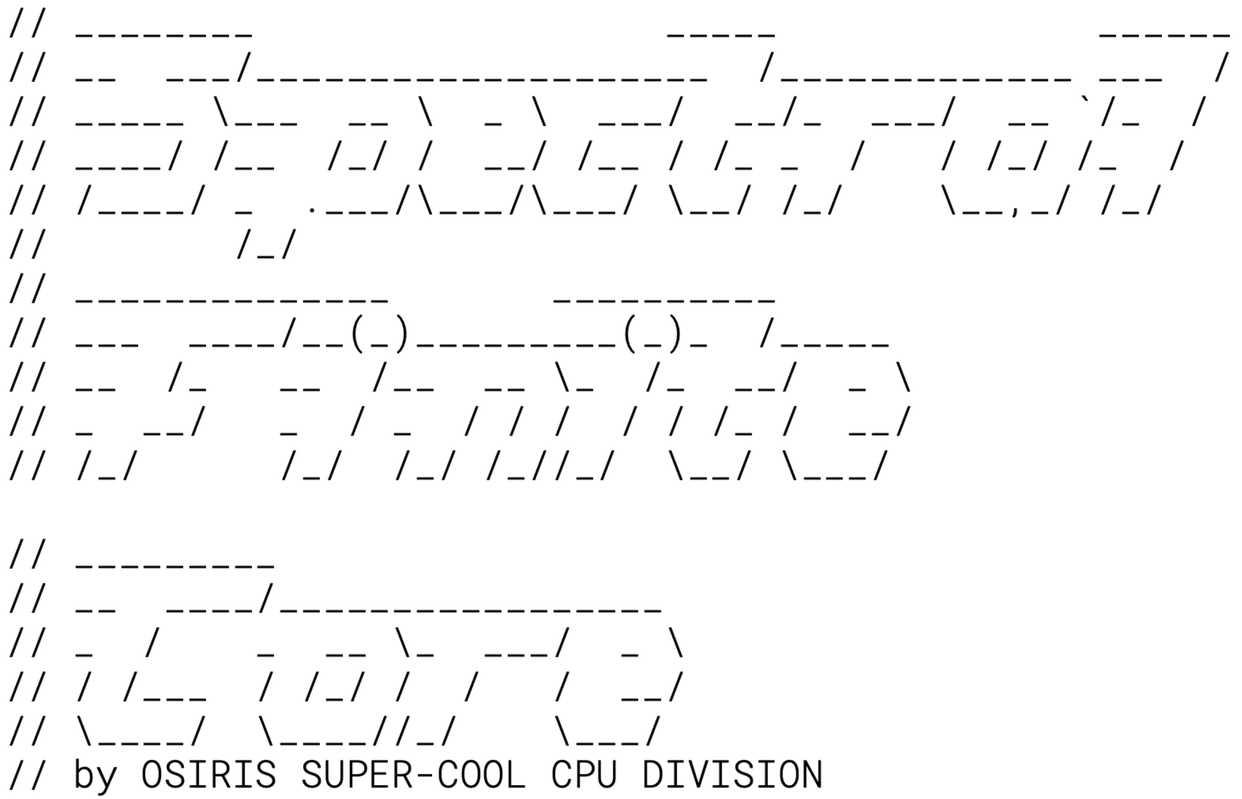


MIT 6.888 Secure Hardware Design Spring 2022
Joseph Ravichandran

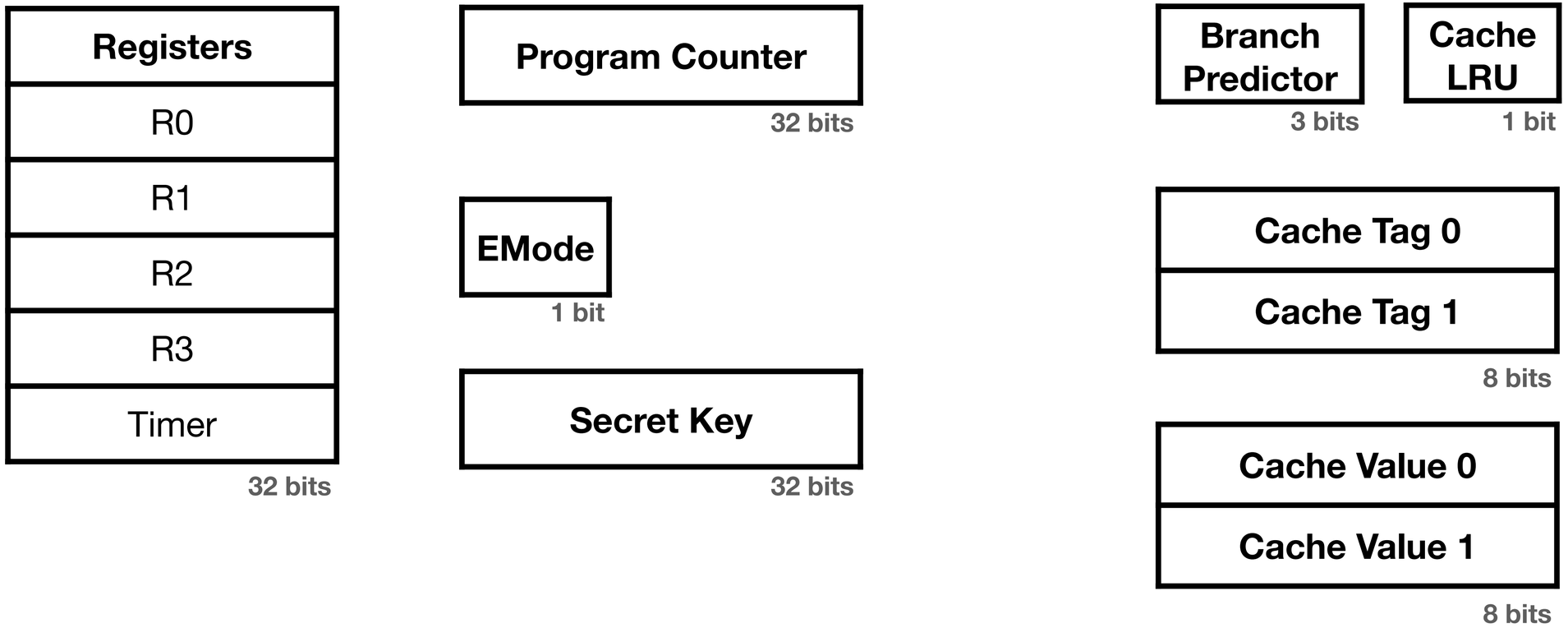
What could possibly go wrong?

Spectral Finite Core

CSAW 2021

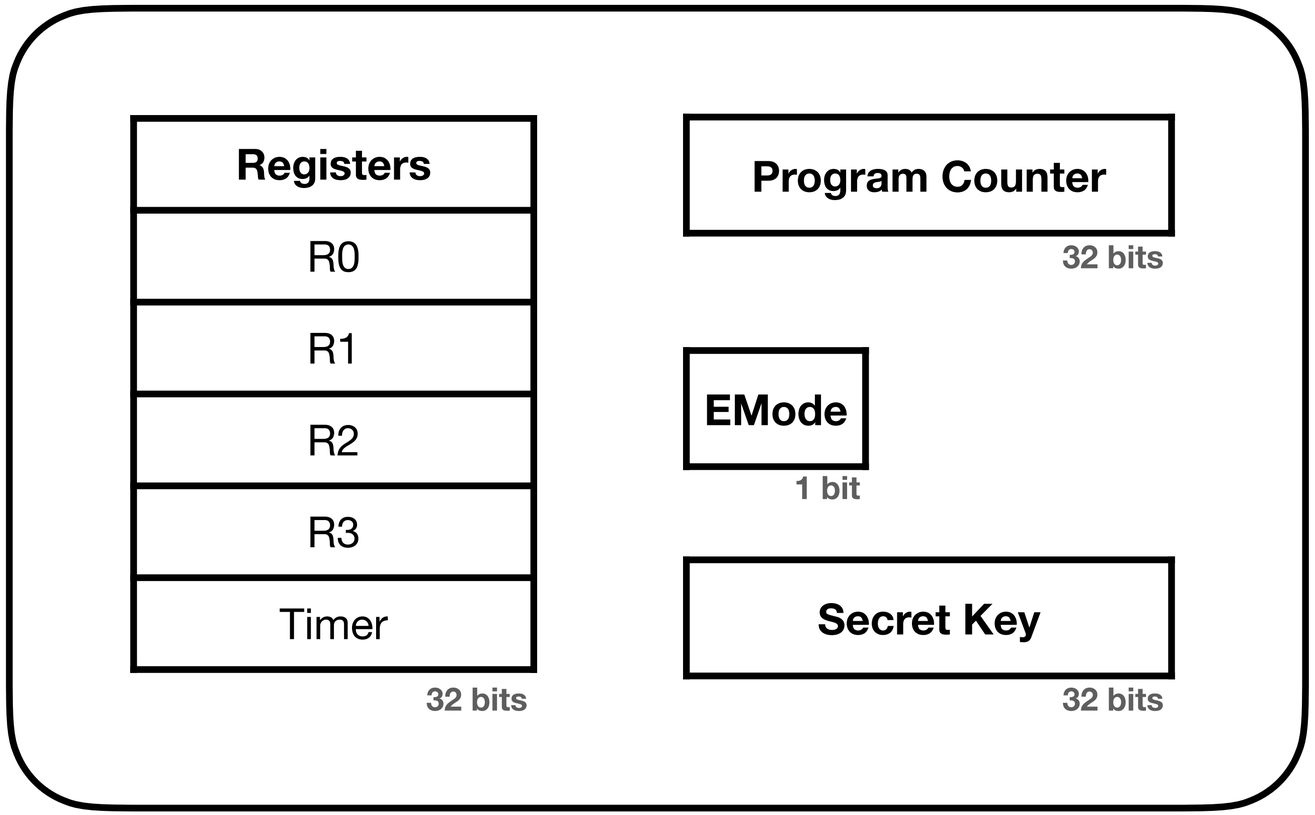


Overview

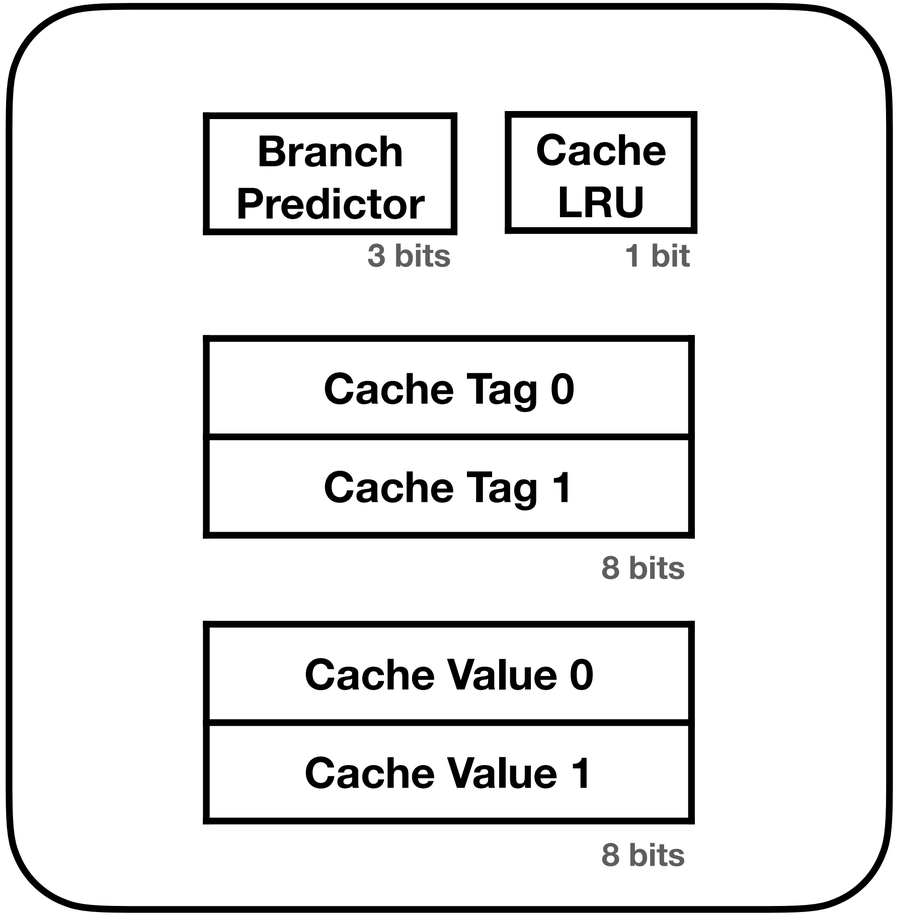

Registers
R0
R1
R2
R3
Timer
32 bits
Program Counter
32 bits
EMode
1 bit
Secret Key
32 bits
Branch Predictor
3 bits
Cache LRU
1 bit
Cache Tag 0
Cache Tag 1
8 bits
Cache Value 0
Cache Value 1
8 bits

Overview

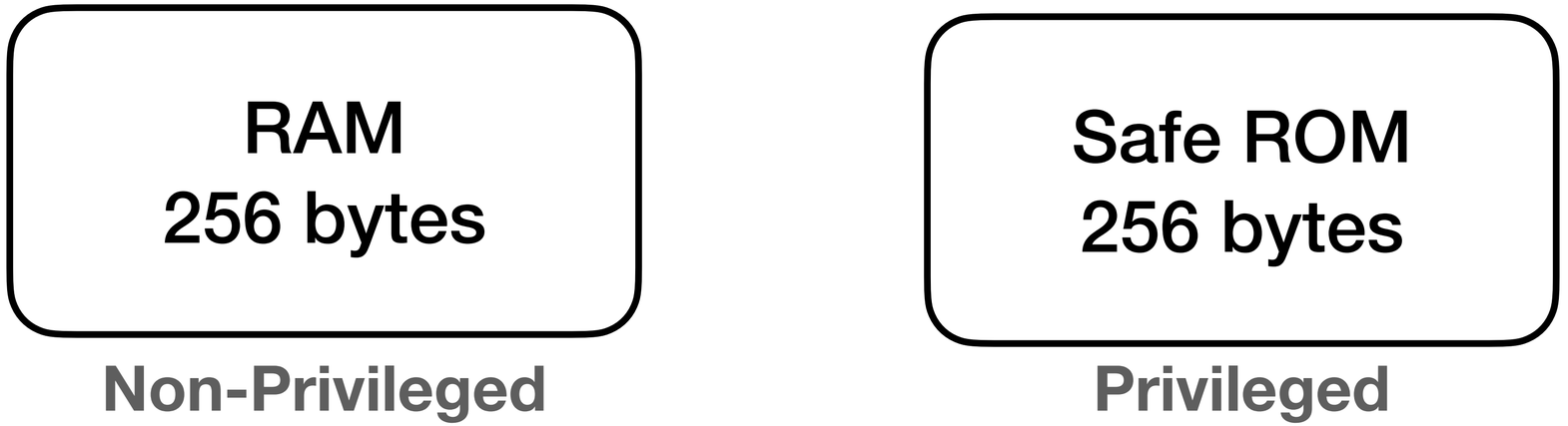
ISA



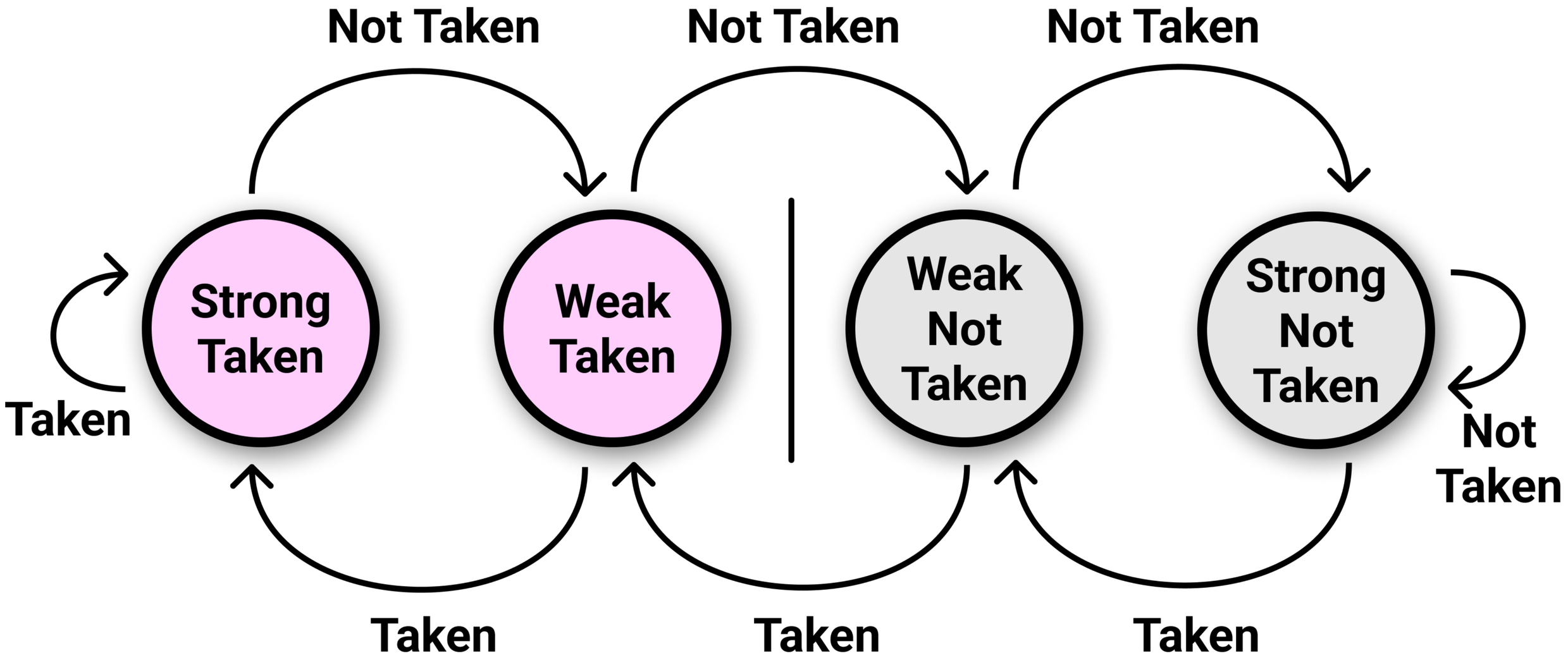
Micro-Arch



Address Space

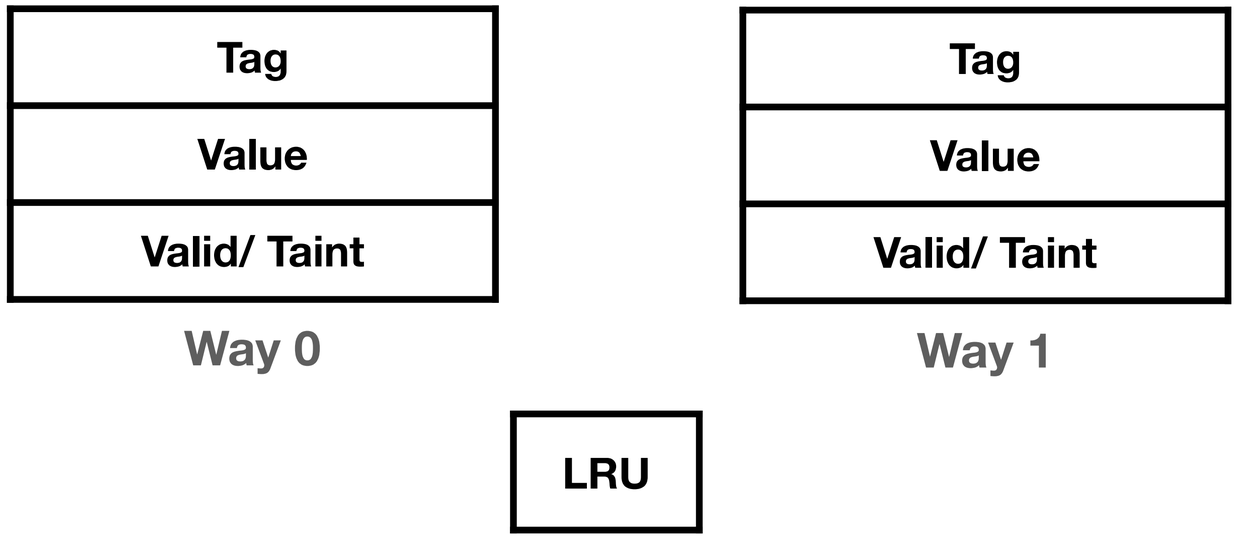


Branch Predictor



Caching Policy

2-Way Fully Associative



Caching Policy

2-Way Fully Associative

Tag	Address of Line
Value	Contents of Line
Valid/ Taint	0 = invalid 1 = RAM 2 = safe ROM

Instruction Encoding

```
# Each instruction is 2 bytes:
################################
# rB (2) | rA (2) | opcode (4) # +0
################################
#        8 bit immediate       # +1
################################
```

Memory Instructions

movf rd, imm8	rd <- ram[imm8]
movt rs, imm8	ram[imm8] <- rs1
movfs rd, imm8	rd <- secure rom[imm8]
movfi rd, rs1, imm8	rd <- ram[rs1 + imm8]
movfsi rd, rs1, imm8	rd <- secure rom[rs1 + imm8]
movfu rd, imm8	rd <- imm8 (no caching!)
flush	Flush the cache

Arithmetic Instructions

add rd, rs1, rs2	rd <- rs1 + rs2
sub rd, rs1, rs2	rd <- rs1 - rs2
inc rd	rd <- rd + 1
rdtm rd	rd <- timer

Control Flow Instructions

jgt target rs1, rs2	pc <- target if rs1 > rs2
jeq target, rs1, rs2	pc <- target if rs1 == rs2
jmp imm8	pc <- imm8

System Instructions

ent	Enter emode if r0 == key
ext	Exit emode

rasm: Our Custom Assembler

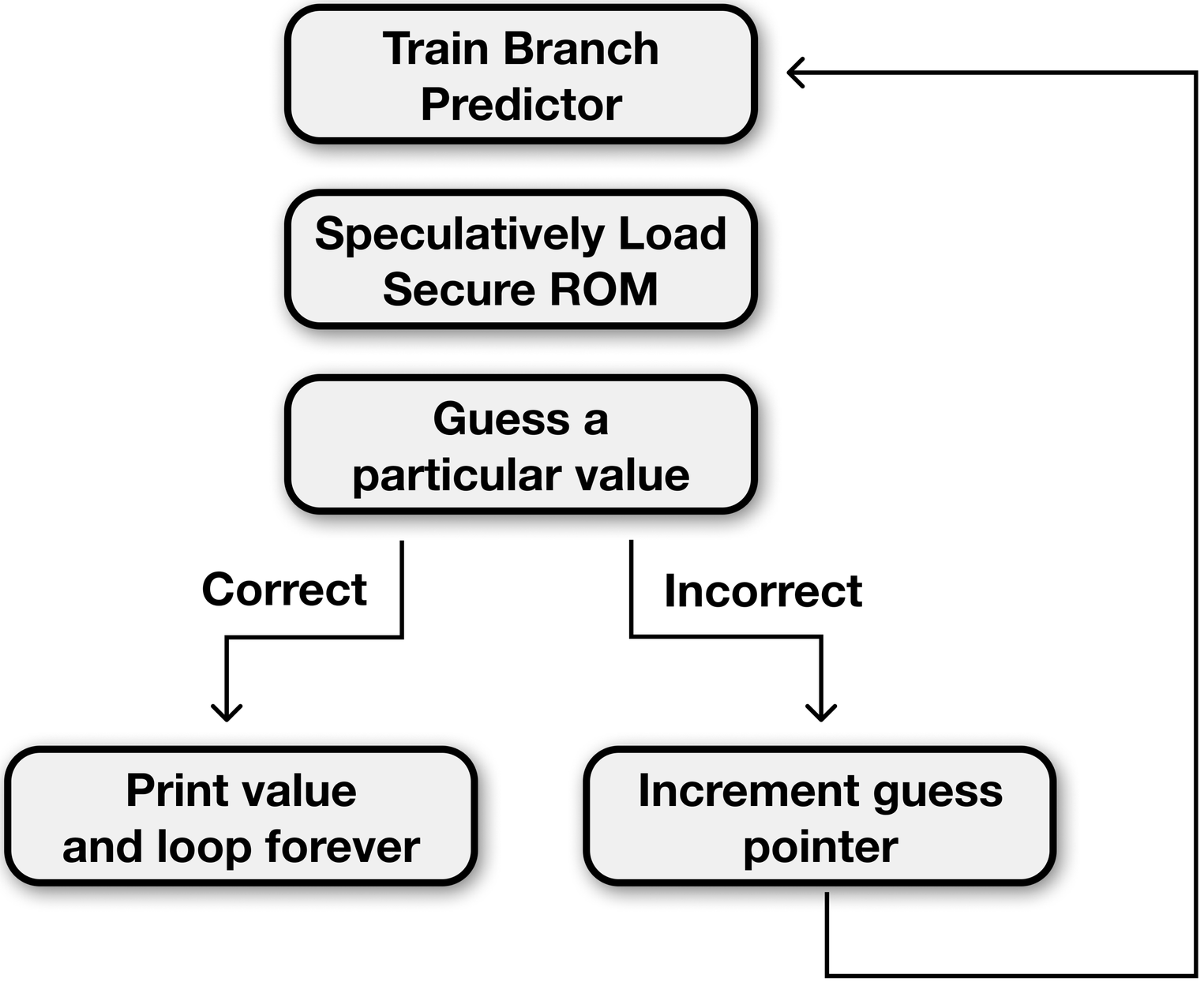
`input.asm`
Your first SFC Program

```
; This is a sample SFC assembly file
; This is created for the "rasm" assembler

; flush the cache:
flush

; Let r3 point to the string
movfu r3, zero

; This is a label:
loop:

; Load a byte from ram
; movfi = move indirect
movfi r1, r3, string_to_print
```

```
; Continue loop
inc r3
movfu r0, zero
jeq loop_exit r0, r1
```

```
; Store the loaded constant in 0xFF
movt r1, 0xFF

jmp loop

loop_exit:
jmp loop_exit

; Constant
zero:
dd 0x00

; NULL terminated tring to output ('ABCD')
; We should see 'D' at the end (last char of the string)
string_to_print:
dd 0x41
dd 0x42
dd 0x43
dd 0x44
dd 0x00
```

`demo_cache.asm`
How to time the cache in SFC


Train Branch Predictor
Speculatively Load Secure ROM
Guess a particular value
Correct
Incorrect
Print value and loop forever
Increment guess pointer

Speculatively Load Secure ROM

```
; Set initial conditions
movfu r0, zero
movfu r2, zero
flush

; r2 holds our current guess
test_guess:

; Reset cache and load an address into the cache
flush
movf r0, 0x41
; @TODO: Use spectre to load from secure ROM here instead!
```

Guess a particular value

Increment guess pointer

Print value and loop forever

```
; Now we will try to figure out which line was just loaded

; Load r1 <- mem[r2]
; If r2 is equal to the address in the cache this will be fast
; Otherwise this will be slow
rdtm r0
movfi r1, r2, 0x00
rdtm r1

sub r1, r1, r0
movfu r0, threshold
jgt success, r0, r1

inc r2
jmp test_guess

; Print the byte to the last line of memory
success:
movt r2, 0xff
jmp success
```

Demo

Commands

make	Rebuild the SFC machine
./build.sh	Assemble input.asm with rasm, store result in ram.hex
./nco	Execute the SFC machine

Your Turn!

Find a speculative execution bug in SFC
by reverse engineering the Verilog.

See if you can use it to bypass the cache access domain policy and
speculatively load from secure ROM!