Constructive Computer Architecture:

Lab Comments & RISCV

Notes

- Inelastic VS Elastic
- The FIFO lab has been graded
- Good job on the conflicting, pipeline, and bypass FIFOs
- Difficulties in the Conflict-Free FIFO with clear.

What is a Conflict-Free FIFO?

- Is it a FIFO with a conflict matrix filled with CFs?
  - Not really...
- Is it a FIFO that, in its typical use, doesn’t impose any more conflicts than necessary?
  - Yes!

Typical FIFO use

- Enqueuing:
  - check notFull and call enq(x)
- Dequeuing:
  - check notEmpty and call first and deq
- Clearing:
  - call clear

- These are the methods that commonly appear in the same rule
  - A CF FIFO says enqueuing and dequeuing rules will have no scheduling conflict imposed by the FIFO.
  - {notFull, enq} CF {notEmpty, first, deq}

Sequential FIFO Execution

Assume FIFO is empty initially

1. enq(1)
2. first
3. deq
4. enq(2)
5. clear
6. enq(3)
7. enq(4)
8. first
9. deq

How many clock cycles does this take?
Should that change the answer to these questions?

$\leftarrow$ What is the state of the fifo here?

What does first return?

How many elements are in the FIFO at the end?

Concurrent FIFO Execution

With Clock Cycles

Clock cycle boundary

```
1. enq(1)
- - - - - - - - - - - -
2. first
3. deq
- - - - - - - - - - - -
4. enq(2)
5. clear
- - - - - - - - - - - -
6. enq(3)
- - - - - - - - - - - -
7. enq(4)
8. first
9. deq
```

If these are both valid executions for a given FIFO, they should produce the same results.

```
1. enq(1)
2. first
3. deq
- - - - - - - - - - - -
4. enq(2)
- - - - - - - - - - - -
5. clear
- - - - - - - - - - - -
6. enq(3)
- - - - - - - - - - - -
7. enq(4)
8. first
9. deq
```

Concurrent FIFO Execution

Clear-Enq Interactions

1. enq(1)
2. first
3. deq
4. enq(2)
5. clear
6. enq(3)
7. enq(4)
8. first
9. deq

What should clear and enq do when they happen in the same cycle?

If both executions are valid, then the action has to be dynamic based on the order.

1. enq(1)
2. first
3. deq
4. enq(2)
5. clear
6. enq(3)
7. enq(4)
8. first
9. deq

Can we detect the order of methods firing?

- In order for a method to know if it fired after another method fired, it has to always be scheduled after that method
- If you want two methods to be able to tell if either of them came after the other, they have to conflict
  - They will never fire in the same cycle making this problem of who fired first trivial...

Concurrent FIFO Execution

CF FIFO solution

1. enq(1)
2. first
3. deq
4. enq(2)
5. clear
6. enq(3)
7. enq(4)
8. first
9. deq

Force enq < clear

Invalid Execution

1. ~~enq(1)~~
2. ~~first~~
3. ~~deq~~
4. ~~enq(2)~~
5. ~~clear~~
6. ~~enq(3)~~
7. ~~enq(4)~~
8. ~~first~~
9. ~~deq~~

What does a CF FIFO do?

- It allows the most flexible scheduling constraints possible while still requiring all valid concurrent executions to match the expected result from sequential execution.
  - That is why {enq, deq} < clear and not {enq, deq} CF clear

What about the canonicalize rule?

- What happens if you have:
  - {enq, deq} < canonicalize < clear
    - If deq and clear occur in the same rule, that rule conflicts with canonicalize. This may or may not be a problem depending on the exact use.
  - {enq, deq} < clear < canonicalize
    - canonicalize can always fire at the end of the cycle independent of how enq, deq, and clear are used in rules

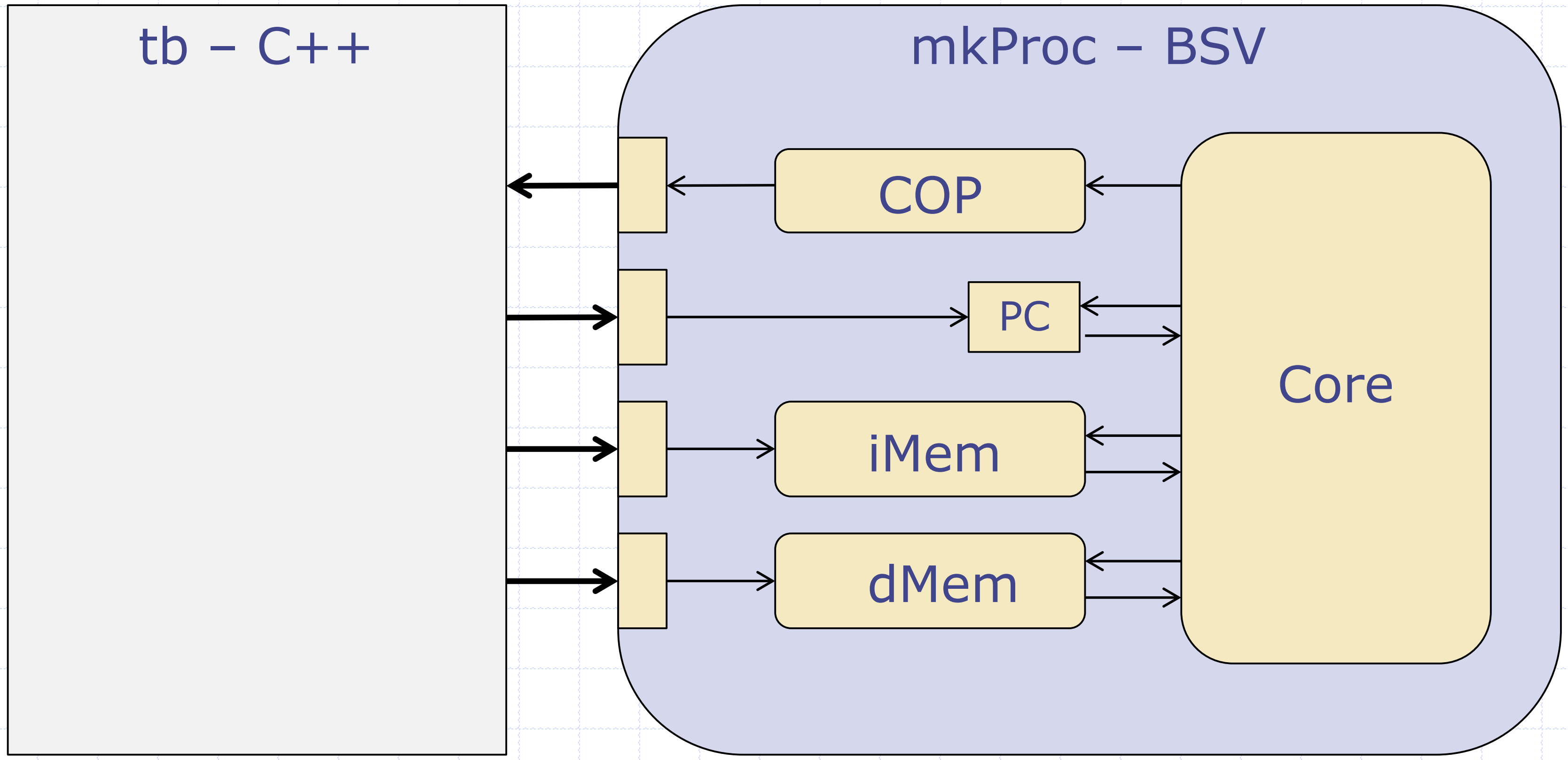
RISCV tutorial

Introduction

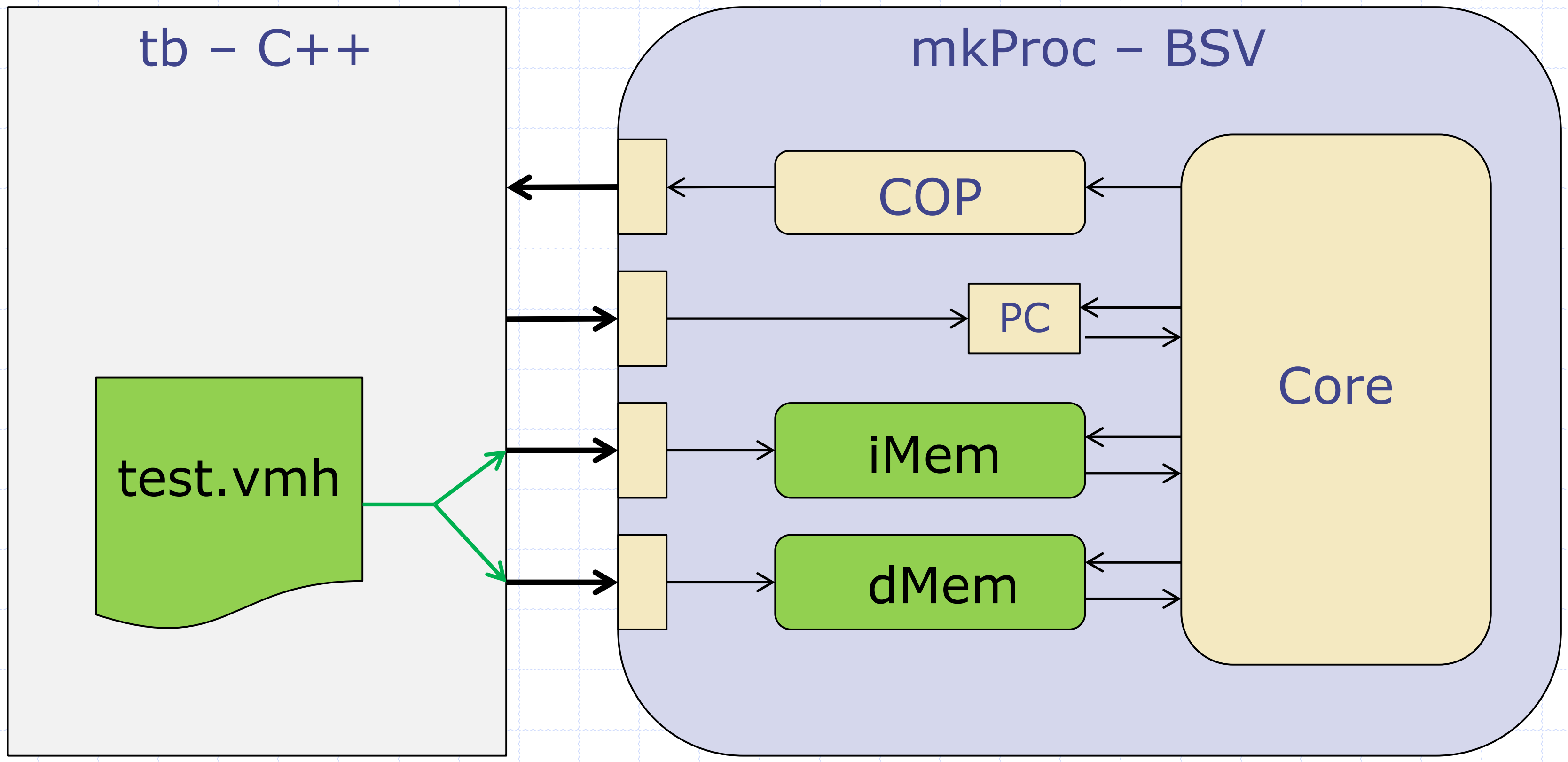
- In lab 5, you will be making modifications to an existing, functional, RISCV processor
- How do you know if your processor is working?
  - You will run an existing suite of C and assembly software test benches on your processor
- What could go wrong?
  - Software and Hardware
- How will you debug this?

Running RISCV

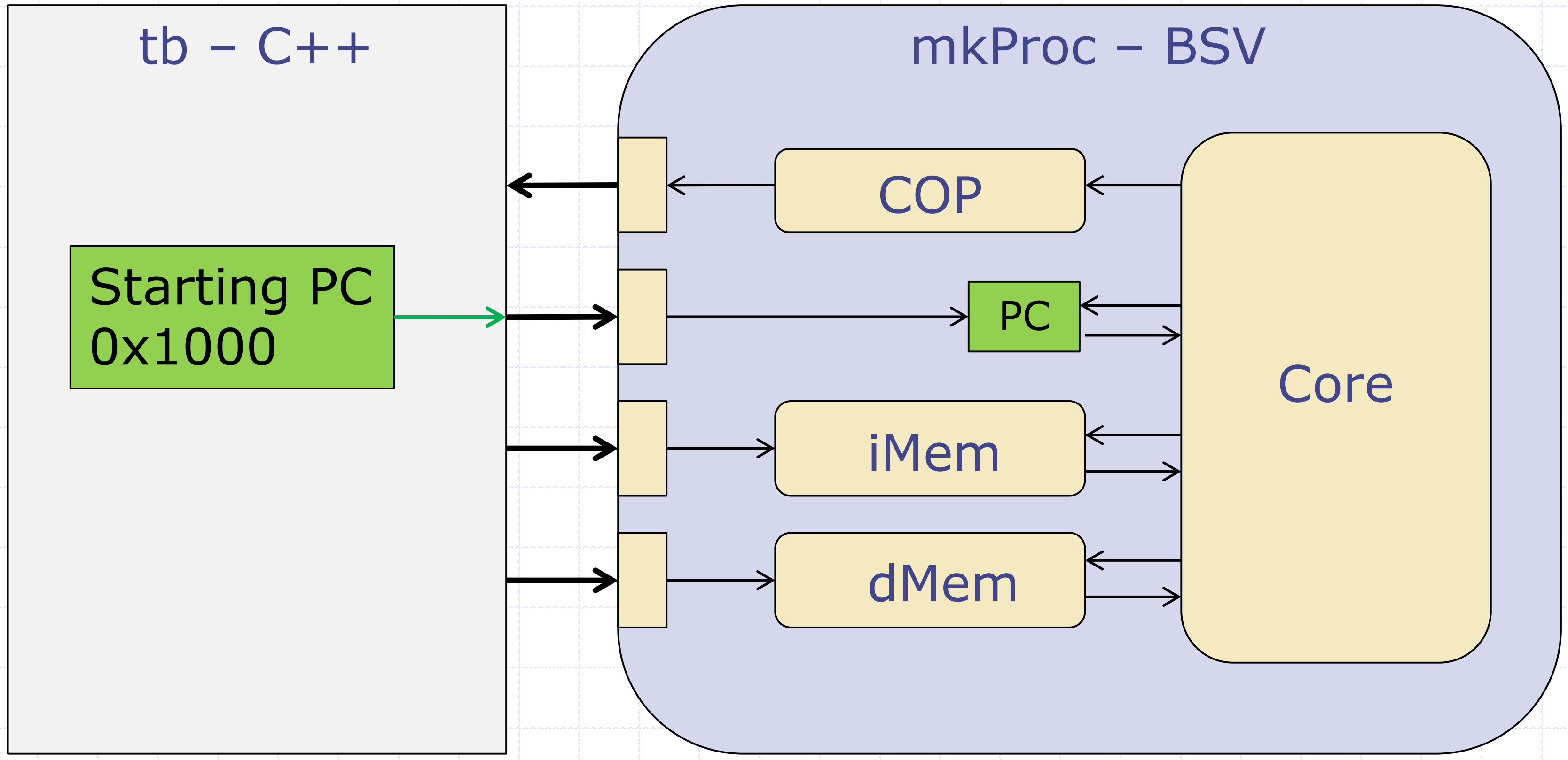
Connectal Testbench Interface



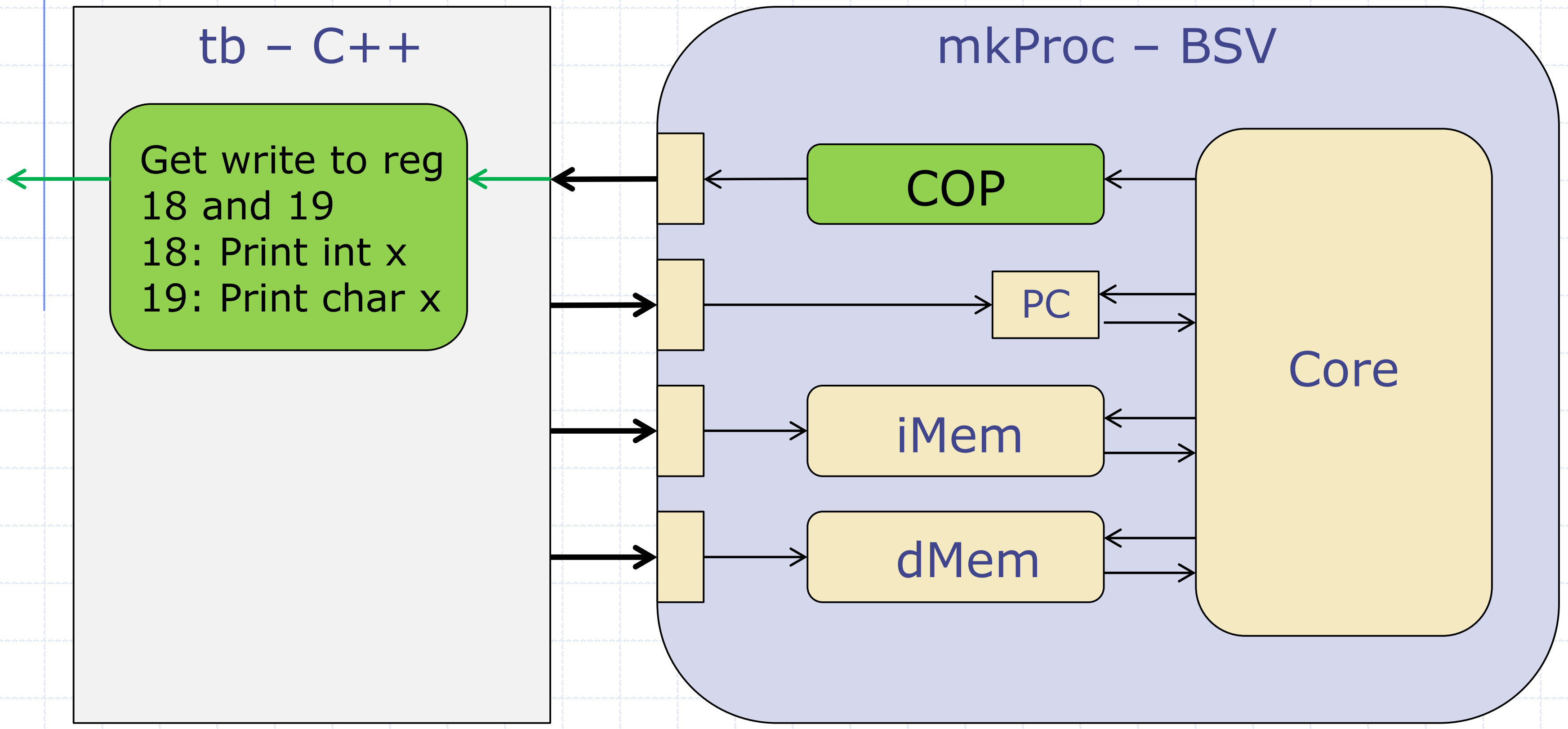
Loading Programs



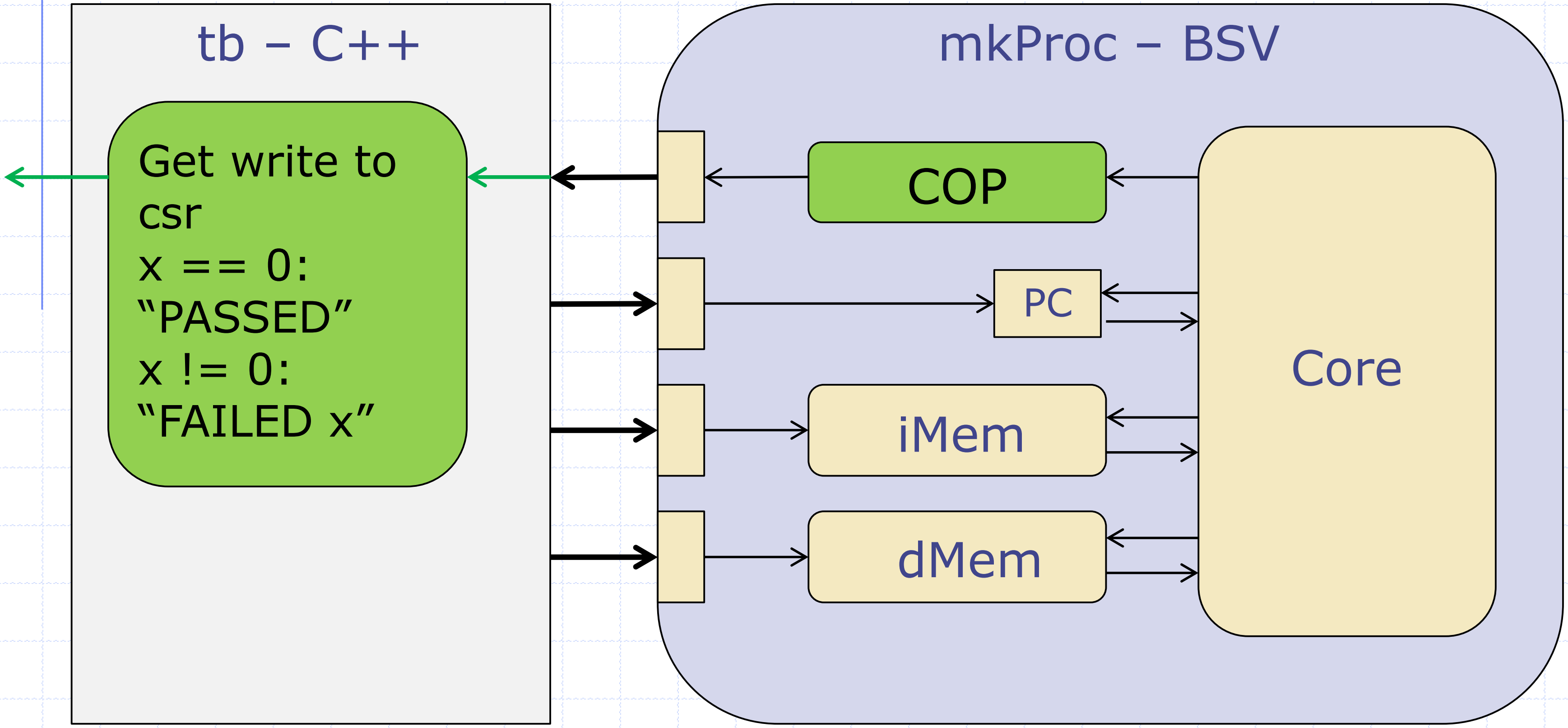
Starting the Processor



Printing to Console



Finishing Program



Get write to csr
x == 0: "PASSED"
x != 0: "FAILED x"

- Ballad in the code
  - Compilation of software
  - Life of a print statement
    - Software side
    - Hardware side
  - Example of debugging

Programming RISCV

Tooling for RISCV software

- How to compile your own C
- How to assemble
- How to disassemble

C Programs

- Start code
  - Jumps to main.
- Print library
  - Can print chars, ints, and strings
- No malloc
  - You can write one!

Functional Programming

Functional Programming

- Basics of functional programming emphasizes pure functions
  - pure functions have no side effects
- Avoids mutable data (mutable = mutate + able)
  - Example: Instead of sorting a list in place, create a function called sort that takes in a list and returns a new list that is sorted
- Functions are typically first-class objects
  - Functions can take in functions as arguments and return them

Intro to Haskell
A functional programming language

- Function type definition:
  - `addFive :: Integer -> Integer`
    - The function addFive takes in an integer and produces an integer
- Function definition:
  - `addFive x = x + 5`
    - The function addFive applied to a variable x produces x + 5

Intro to Haskell
A functional programming language

- Function type definition:
  - `add :: Integer -> Integer -> Integer`
    - The function add takes in two integers and produces an integer
- Function definition:
  - `add x y = x + y`
    - The function add applied to variable x and y produces x + y

```
addFive = add 5
```

Intro to Haskell
A functional programming language

- The add function is actually *curried*
  - It is a function on one Integer that returns (a function on one Integer that returns an Integer)
  - `add :: Integer -> (Integer -> Integer)`
  - `add x ::            Integer -> Integer`
  - Like add(x)(y) instead of add(x,y)
- BSV supports currying too