Constructive Computer Architecture

Tutorial 3

RISC-V Processor

Sizhuo Zhang

6.175 TA

Course Content

Normal Register File

```
module mkRFile(RFile);
  Vector#(32,Reg#(Data)) rfile <- replicateM(mkReg(0));

  method Action wr(RIndx rindx, Data data);
    if(rindx!=0) rfile[rindx] <= data;
  endmethod
  method Data rd1(RIndx rindx) = rfile[rindx];
  method Data rd2(RIndx rindx) = rfile[rindx];
endmodule
```

{rd1, rd2} < wr

Bypass Register File using EHR

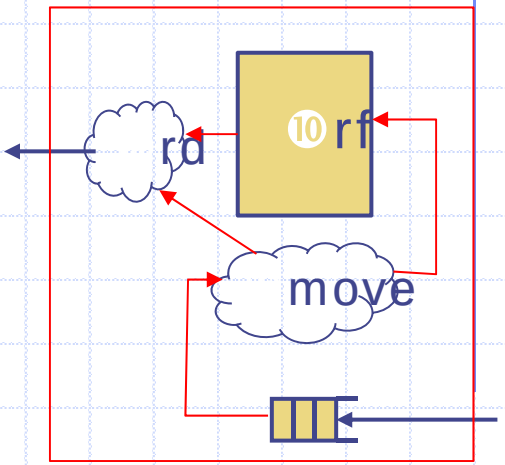
```
module mkBypassRFile(RFile);
  Vector#(32,Ehr#(2, Data)) rfile <-
                              replicateM(mkEhr(0));

  method Action wr(RIndx rindx, Data data);
    if(rindex!=0) (rfile[rindex])[0] <= data;
  endmethod
  method Data rd1(RIndx rindx) = (rfile[rindx])[1];
  method Data rd2(RIndx rindx) = (rfile[rindx])[1];
endmodule
```

wr < {

Bypass Register File

with external bypassing




```
module mkBypassRFile(BypassRFile);
  RFile              rf <- mkRFile;
  Fifo#(1, Tuple2#(RIndx, Data))
                  bypass <- mkBypassSFifo;
  rule move;
    begin rf.wr(bypass.first); bypass.deq end;
  endrule
  method Action wr(RIndx rindx, Data data);
    if(rindex!=0) bypass.enq(tuple2(rindx, data));
  endmethod
  method Data rd1(RIndx rindx) =
      return (!bypass.search1(rindx)) ? rf.rd1(rindx)
                : bypass.read1(rindx);
  method Data rd2(RIndx rindx) =
      return (!bypass.search2(rindx)) ? rf.rd2(rindx)
                : bypass.read2(rindx);
endmodule
```

wr < {rd1, rd2}

Scoreboard implementation
using searchable Fifos

```
function Bool isFound
         (Maybe#(RIndx) dst, Maybe#(RIndx) src);
  return isValid(dst) && isValid(src) &&
              (fromMaybe(?,dst)==fromMaybe(?,src));
endfunction

module mkCFScoreboard(Scoreboard#(size));
  SFifo#(size, Maybe#(RIndx), Maybe#(RIndx))
       f <- mkCFSFifo(isFound);
  method insert  = f.enq;
  method remove  = f.deq;
  method search1 = f.search1;
  method search2 = f.search2;
endmodule
```

Searchable FIFO Code

- Will see in Lab 6

RISC-V Processor

SCE-MI Infrastructure

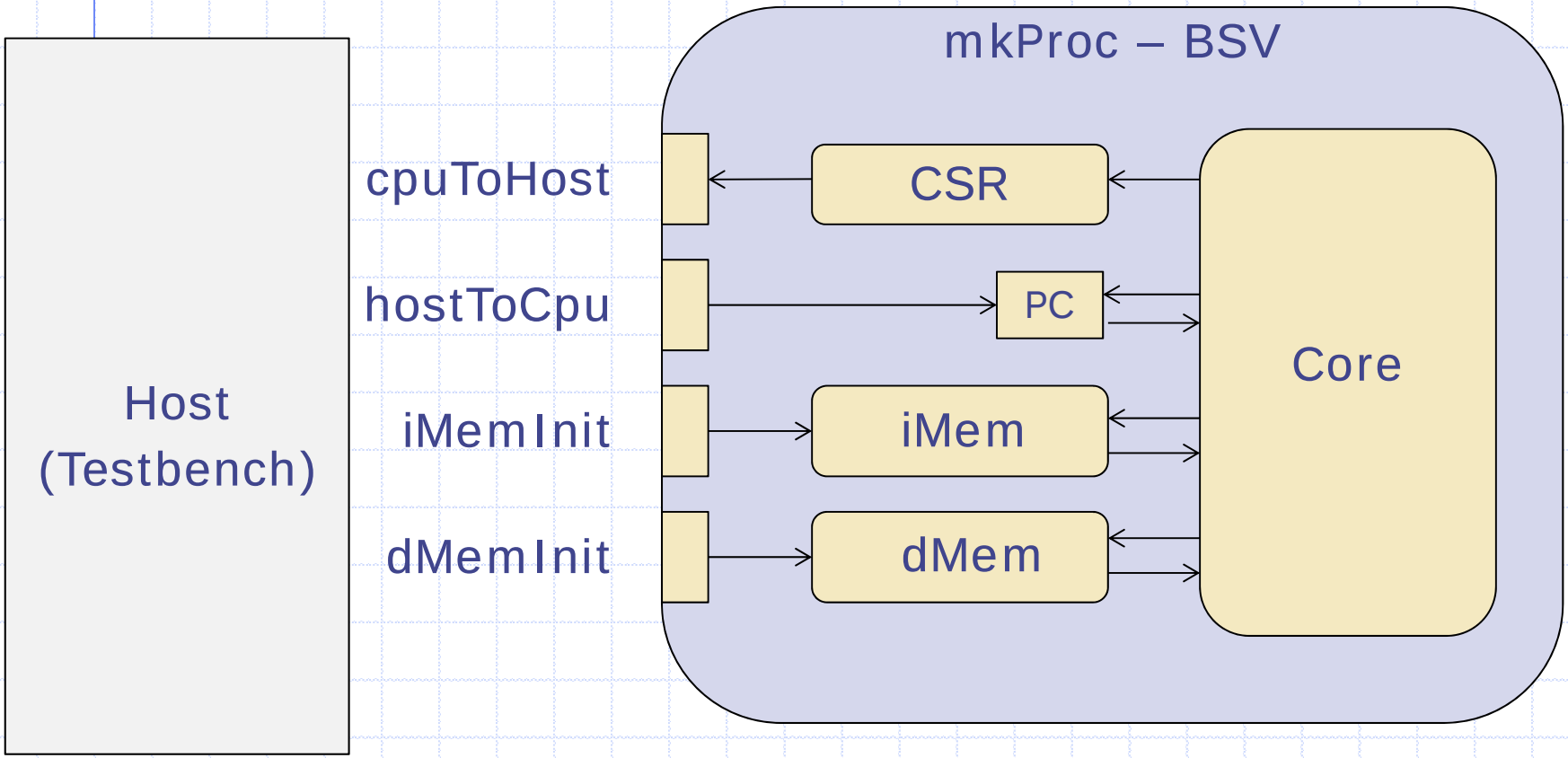
RISC-V Interface

```
interface Proc;
    method Action hostToCpu(Addr startpc);
    method ActionValue#(CpuToHostData) cpuToHost;
    interface MemInit iMemInit;
    interface MemInit dMemInit;
endinterface

typedef struct {
    CpuToHostType c2hType;
    Bit#(16) data;
} CpuToHostData deriving(Bits, Eq);

typedef enum {
    ExitCode, PrintChar, PrintIntLow, PrintIntHigh
} CpuToHostType deriving(Bits, Eq);
```

RISC-V Interface



RISC-V Interface: cpuToHost

- Write mtohost CSR: csrw mtohost, rs1
  - rs1[15:0]: data
    - 32-bit Integer needs two writes
  - rs1[17:16]: c2hType
    - 0: Exit code
    - 1: Print character
    - 2: Print int low 16 bits
    - 3: Print int high 16 bits

```
typedef struct {
    CpuToHostType c2hType;
    Bit#(16) data;
} CpuToHostData deriving(Bits, Eq);
```

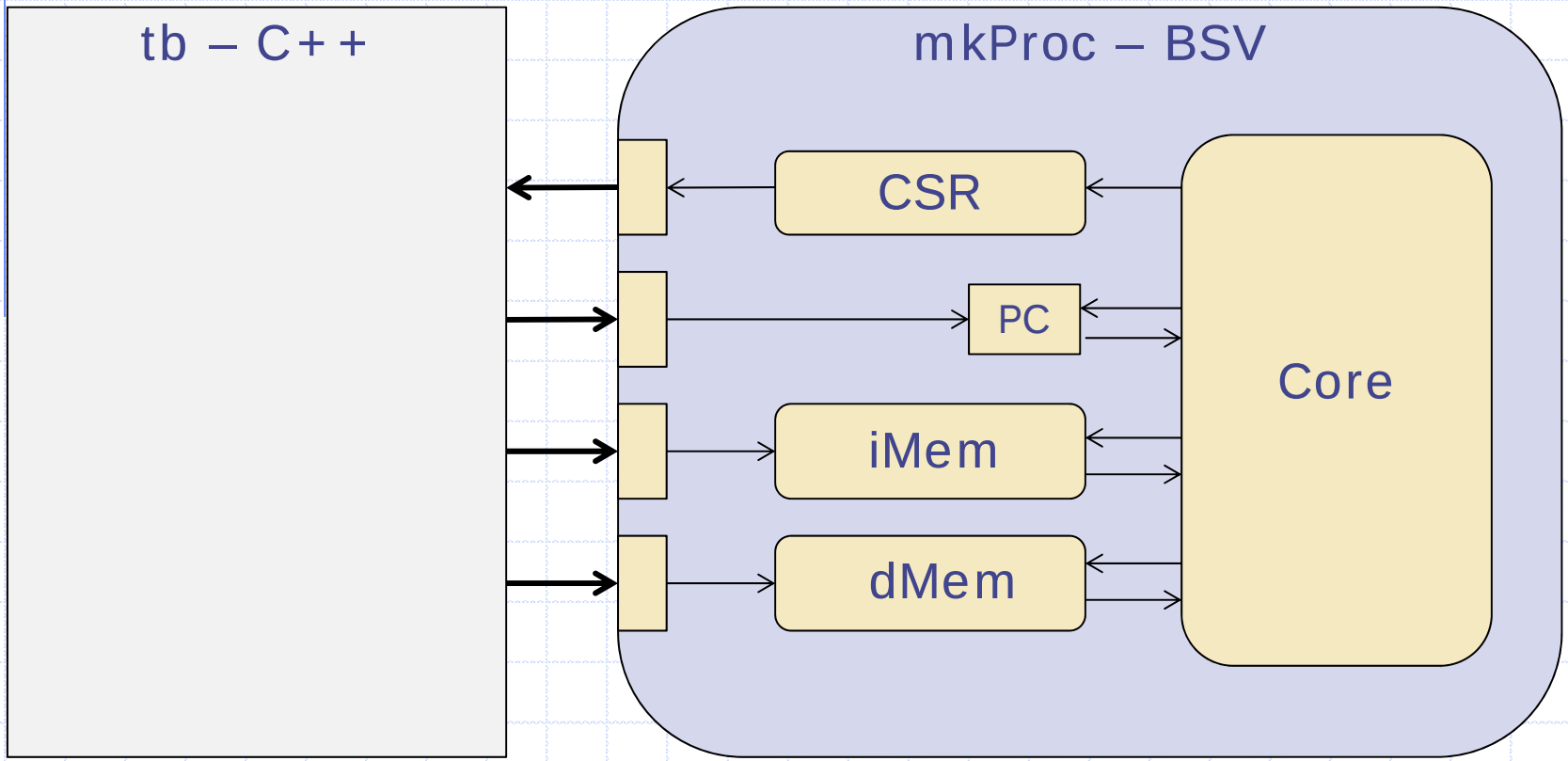
RISC-V Interface: Others

- hostToCpu
  - Tells the processor to start running from the given address
- iMemInit/dMemInit
  - Used to initialize iMem and dMem
  - Can also be used to check when initialization is done
  - Defined in MemInit.bsv

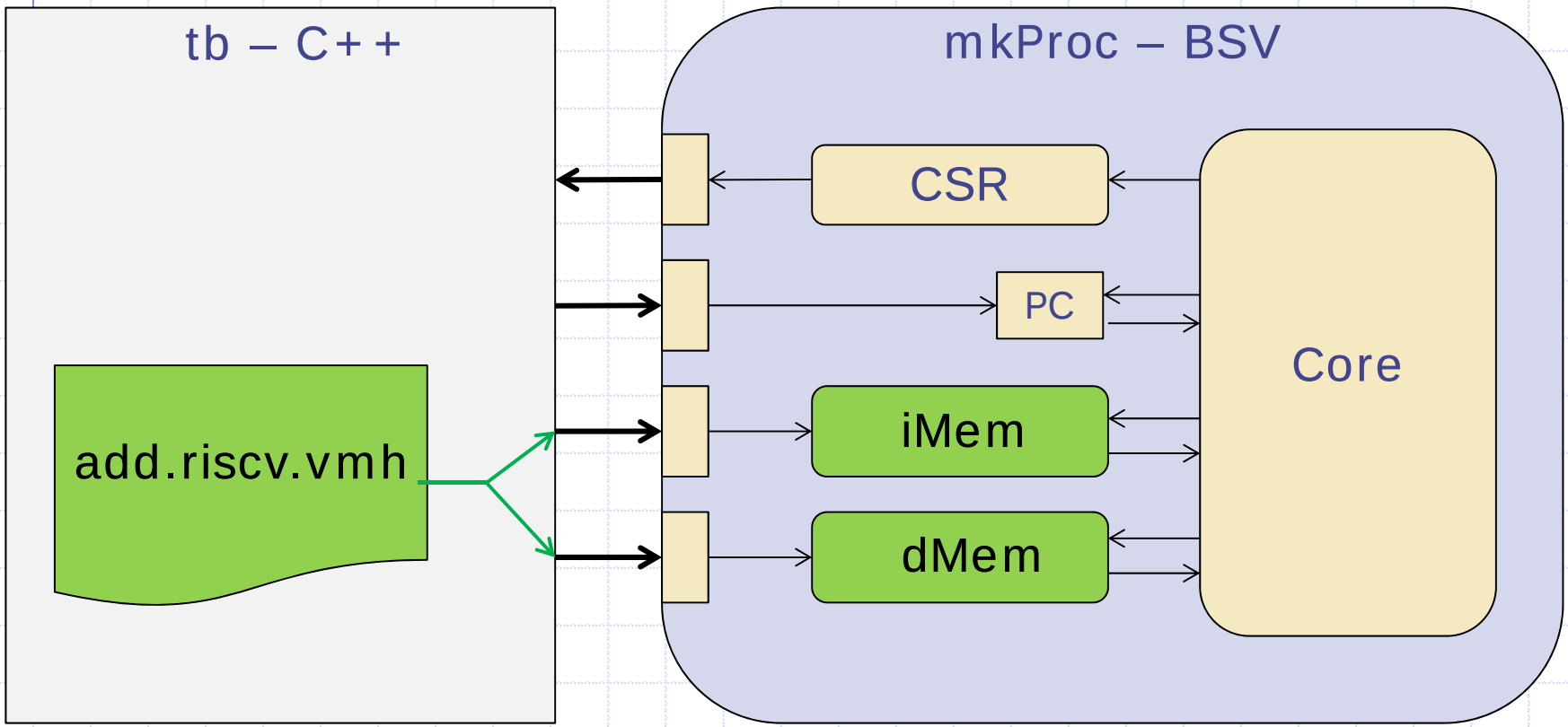
Connecting RISC-V Interface

- Previous labs have used testbenches written in BSV to connect with modules we wanted to test.
- Now we want a more advanced program testing the processor
  - Want to be able to load multiple files from the user and display printed output
- How do we do this?
  - Use a SceMi interface to connect a testbench written in C++ with a module written in BSV

SceMi Interface

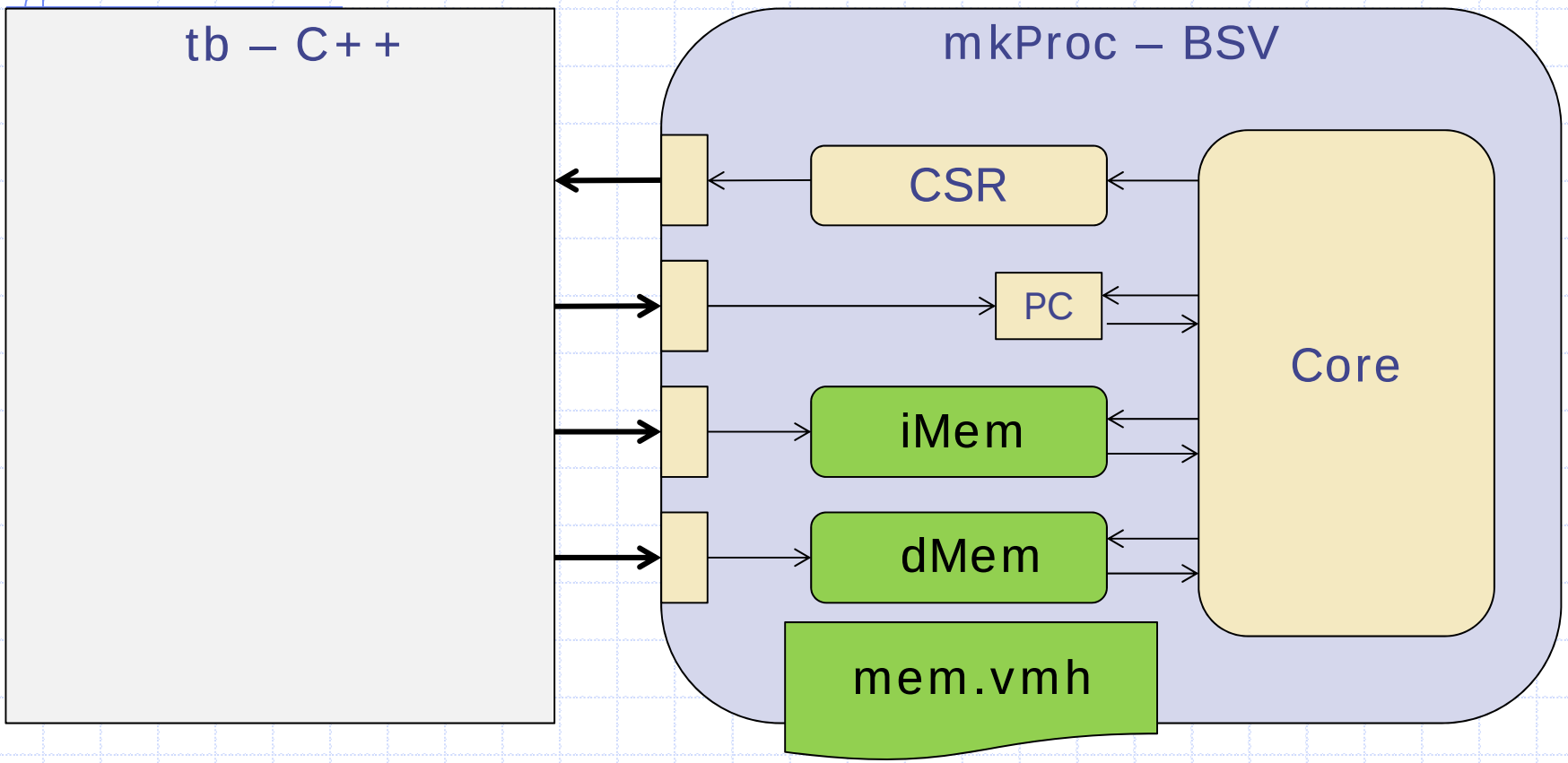


Load Program



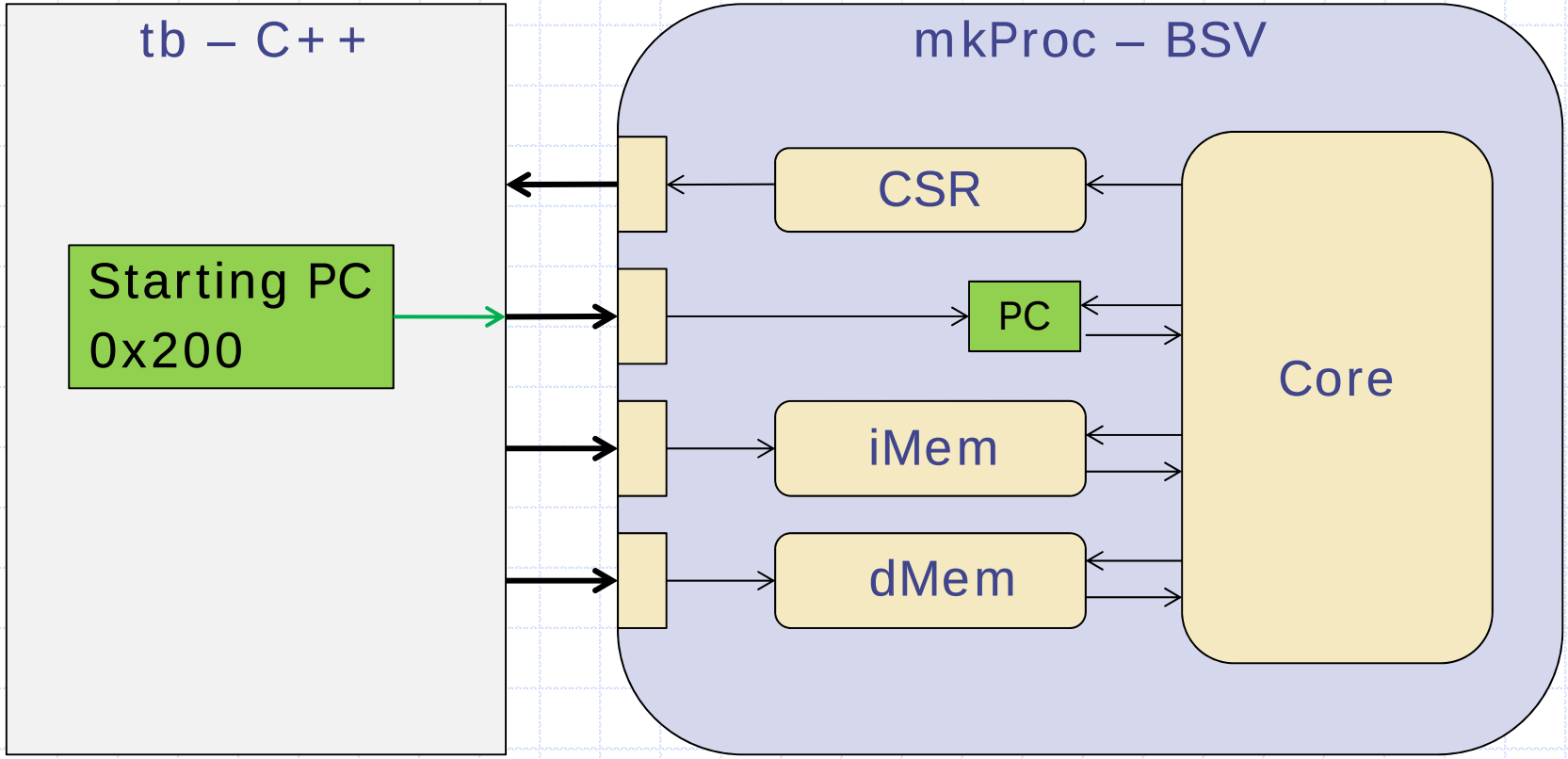
- Bypass this step in simulation

Load Program

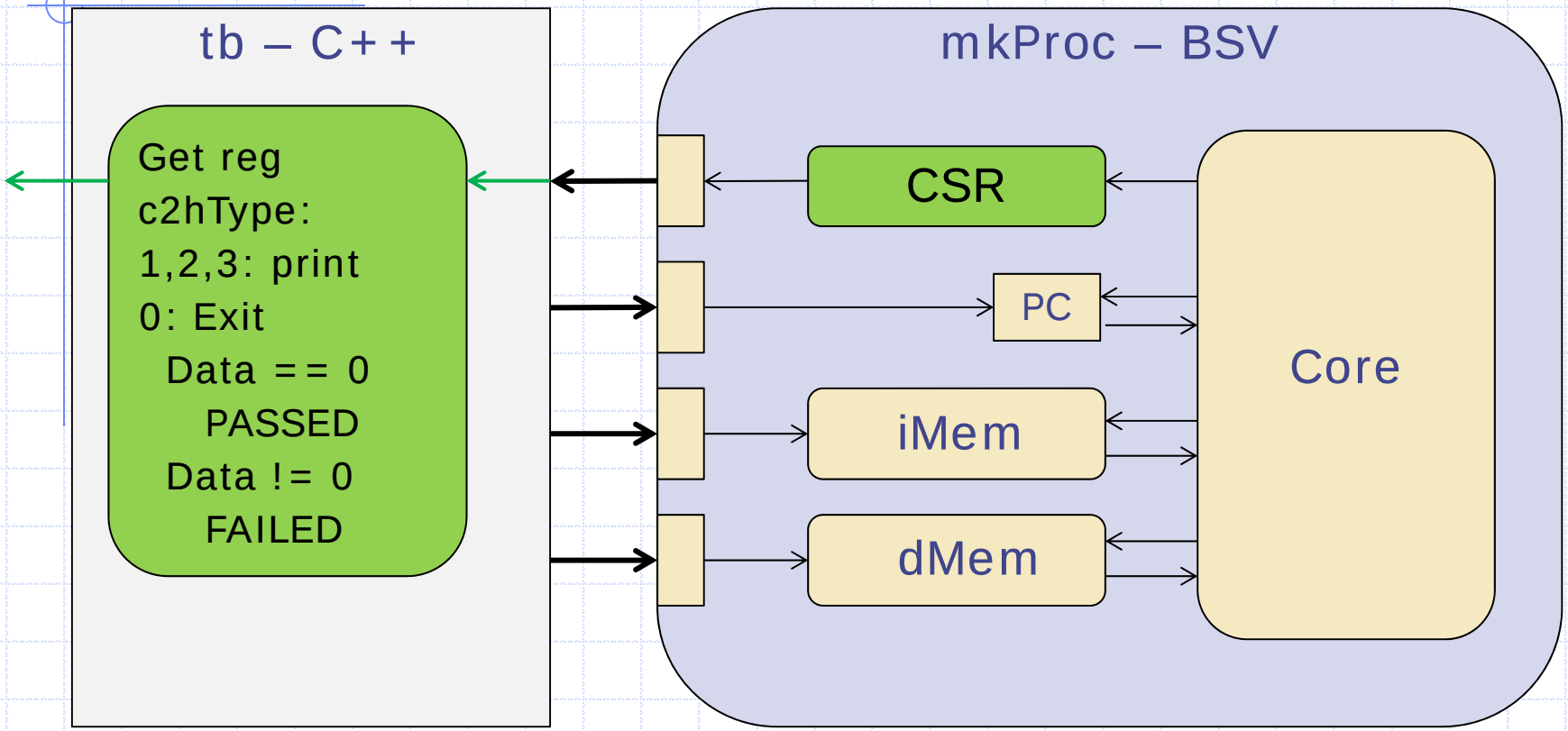


- Simulation: load with mem.vmh (fixed file name)
  - Copy <test>.riscv.vmh to mem.vmh

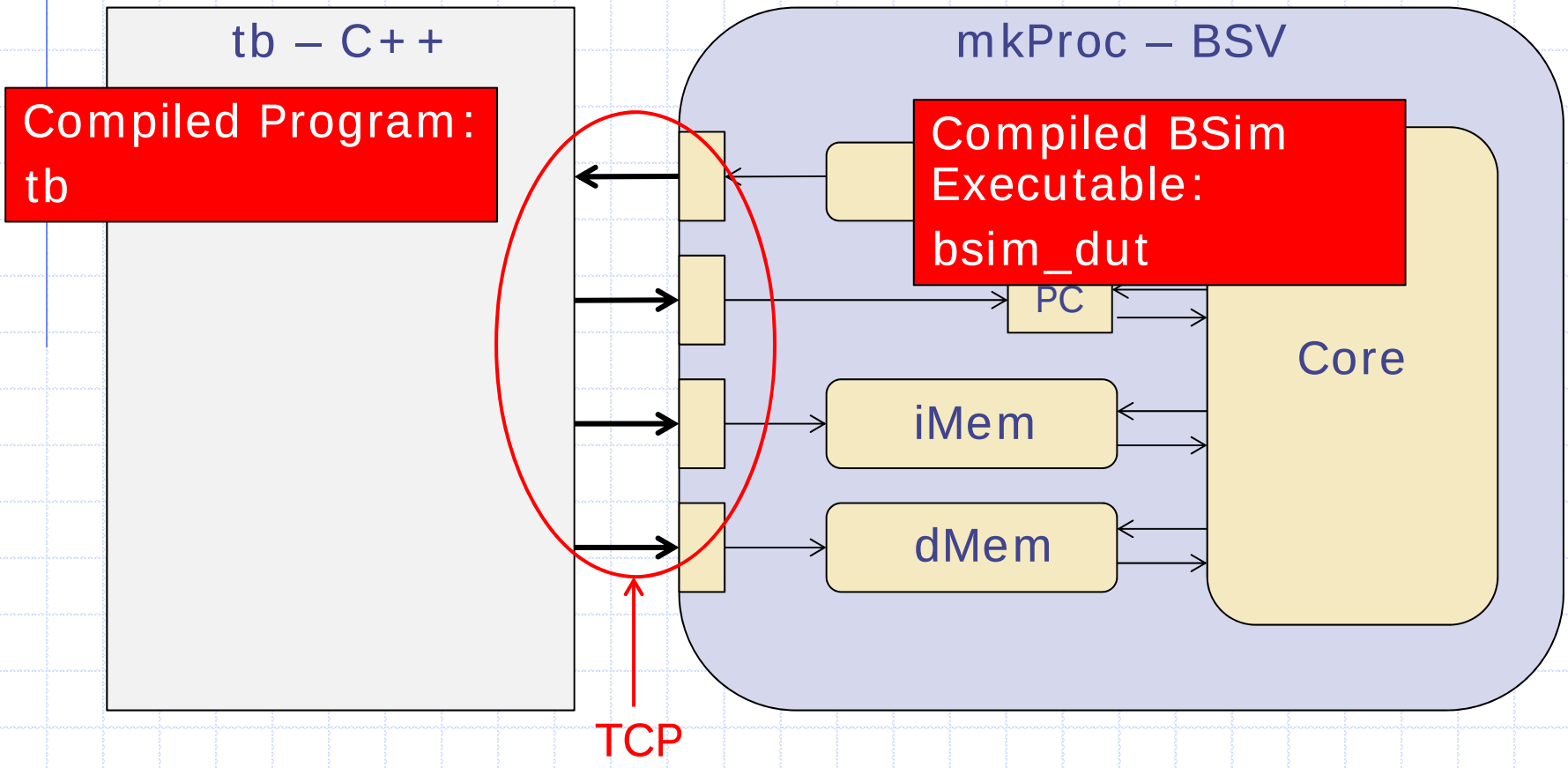
Start Processor



Print & Exit


tb – C++
Get reg
c2hType:
1,2,3: print
0: Exit
Data == 0
PASSED
Data != 0
FAILED
mkProc – BSV
CSR
PC
iMem
dMem
Core

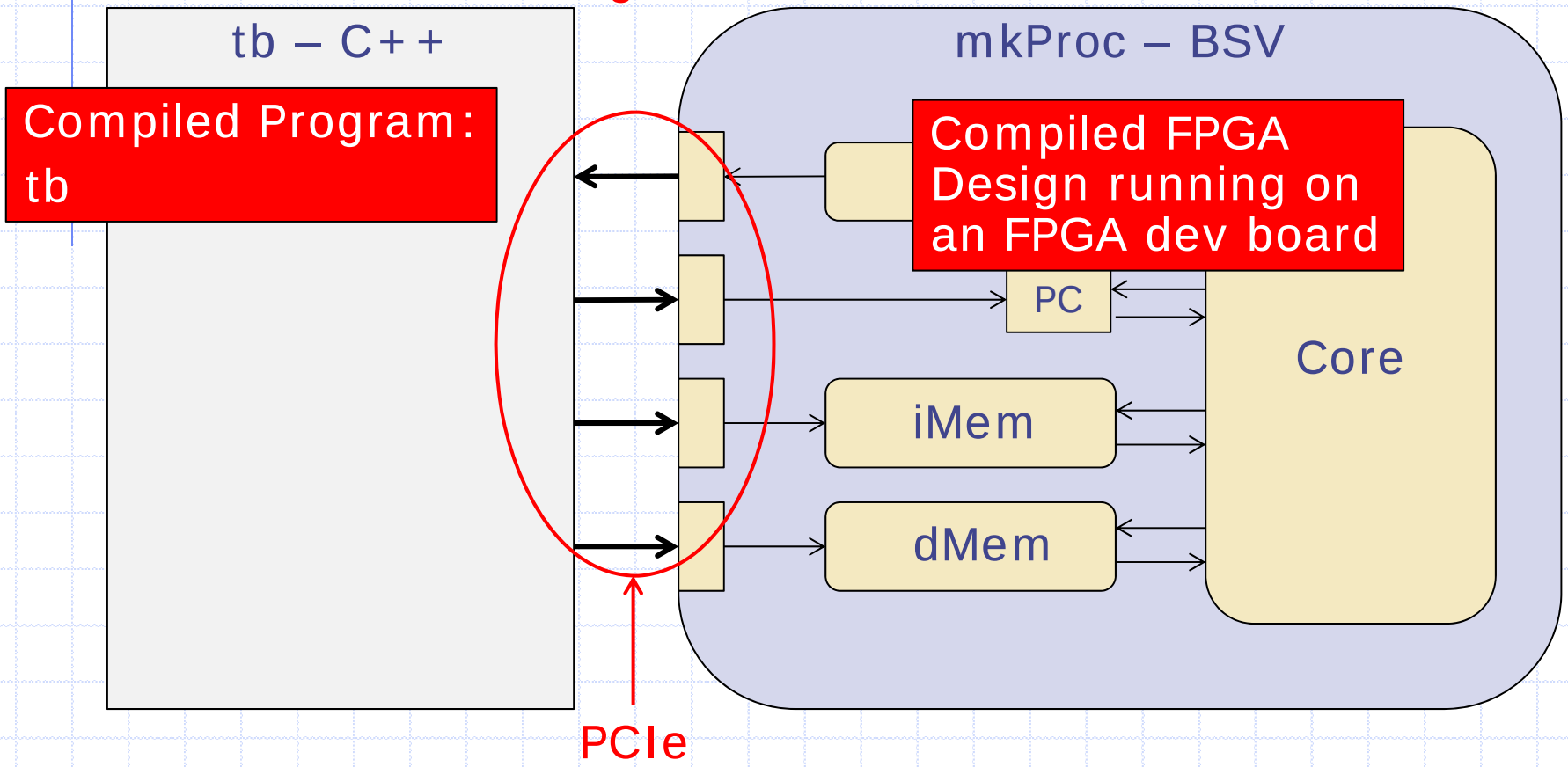
SceMi Interface: Simulation



Build and Run Processor

SceMi Interface: FPGA

The same SceMi Testbench can be used to test hardware FPGA designs too!



SceMi Code

RISC-V Assembly