CS152 Discussion Section 1
Spring 2024

Jan 23, 2023

Agenda

- Admin
- Architecture vs Microarchitecture
- Microcoding Overview
- Lab 1 Overview
- Questions—OH

GSI Introduction

- **Coleman Hooper**
  - chooper@berkeley.edu
  - OH: Tues 10-11am
  - 2nd-yr PhD student advised by Sophia Shao, Kurt Keutzer
  - Research focus: Efficient ML Inference, Hardware for ML
  - Took CS252 last year
- **Eran Kohen Behar**
  - erankohen@berkeley.edu
  - OH: Friday 1-2pm
  - Senior in EECS
  - Research in ML for hardware
  - Took 152 last year

What to Expect from DIS 101/102

- **About 1-1.5 hour of “discussion”**
  - Review of lecture material from that week
  - Work through examples together
    - Blank worksheets linked on website
  - Please interrupt for questions anytime
  - Small break @ 1hr

- **Remaining time used for OH/anything**
  - Intro and help on labs
  - Review for exams
  - Office hours
  - For CS252 students (if they come): discussions on projects

Admin: Week 1

- **Lab1 out soon and due Wednesday Feb 7 @ Midnight**
- **Problem Set 1 is out soon and due Wednesday Feb 7 @ Midnight**
- **Check course webpage for schedule updates**
- **Find partners for lab (CS152) and project (CS252)**
  - Look for Ed “find-a-group” post
  - Optional but can help save your time

Review Part 1. Architecture vs. Microarchitecture

“The term architecture is used here to describe the attributes of a system as seen by the programmer, i.e., the **conceptual structure and functional behavior**, as distinct from the organization of the data flow and control, the logic design, and the physical implementation.”

- **G. M. Amdahl, G. A. Blaauw, and F. P. Brooks, “Architecture of the IBM System/360,” IBM Journal of Research and Development, vol. 8, no. 2, pp. 87–101, Apr. 1964**

Review Part 1. Architecture vs. Microarchitecture

Architecture: The instruction set

- What are some examples of ISAs?
- Why might a company create a closed-source architecture?

Microarchitecture: The implementation of an architecture

- What are some examples of implementations of RISC-V publically available?
- Why might a company create a closed-source microarchitecture?

Q1. Architecture vs Microarchitecture

What is *architecturally* visible?

Instructions modify architecturally-visible state to do work

Is the following exposed by the architecture? (True/False)

- Register file entries in a classic RISC machine **True**
- Stack in a stack architecture **True**
- Pipeline registers **False**
- Branch-delay slots **True**
- NOPs **True**
- Pipeline bubbles **False**
- Condition codes, status flags **True**
- Memory address width **True**
- Instruction/data caches **Usually false, cache management instructions, etc**

Q1. Architecture vs Microarchitecture

Classifying ISAs

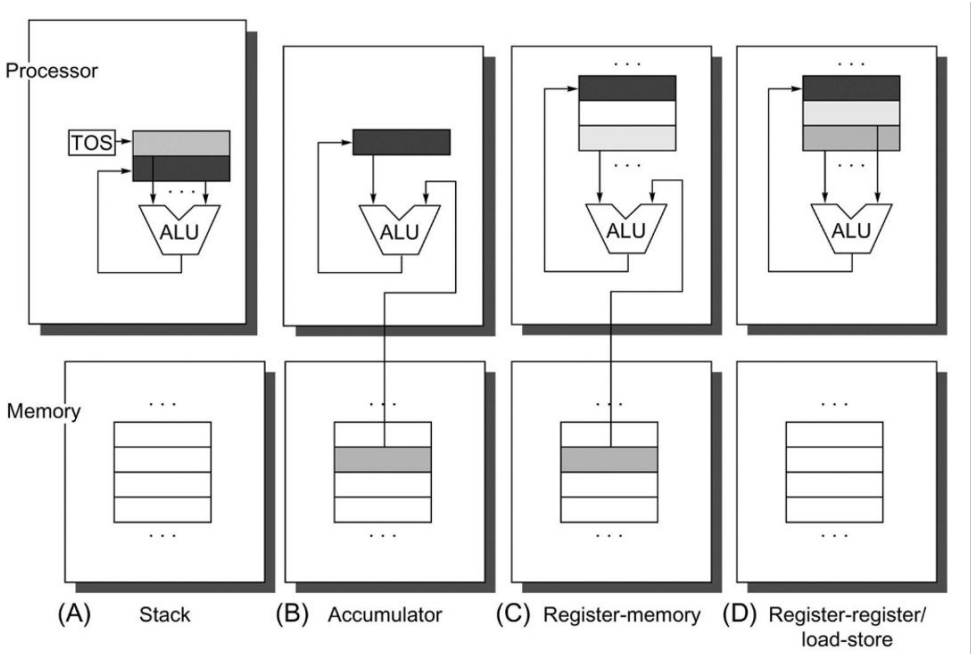
Addressing mode: How operands are identified in an instruction

- RISC: Load-store register architecture
  - Only load/store instructions access memory
  - All other instructions are register-register or register-immediate
- CISC: Not RISC
  - Generally a highly orthogonal instruction set - many different addressing modes
  - i386 memory operand: reg + reg*scale + imm8/32 (scale can be 1, 2, 4, 8)
  - VAX: each operand specifier can be 1 to 17 (!) bytes long
- Stack: No explicit registers, operands are kept on a stack
- Accumulator: One implicit special register as both source and destination

Q1. Architecture vs Microarchitecture

C = A + B where A, B, C are values in memory

(Appendix A of the textbook)



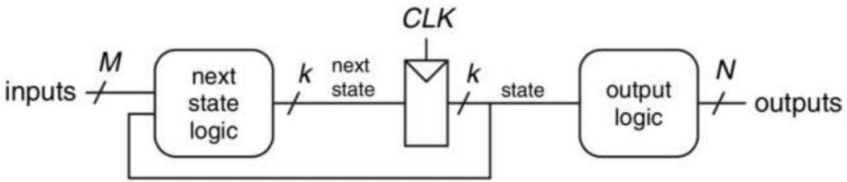
Stack	Accumulator	Register (register-memory)	Register (load-store)
Push A	Load A	Load R1,A	Load R1,A
Push B	Add B	Add R3,R1,B	Load R2,B
Add	Store C	Store R3,C	Add R3,R1,R2
Pop C			Store R3,C

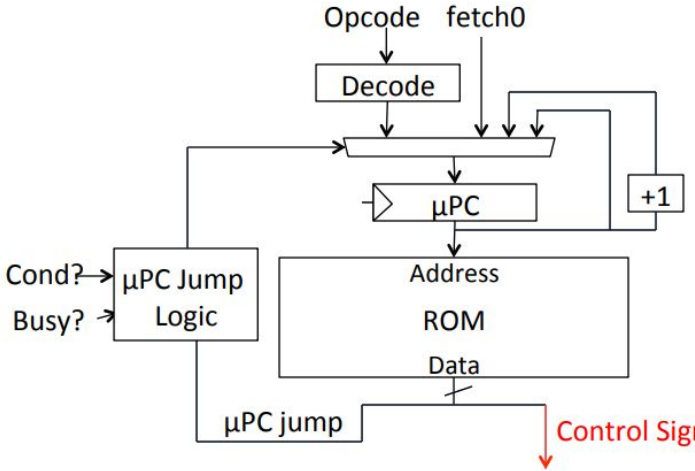
Q2: Microcoded vs Pipelined

Separation of **datapath** (arithmetic, data movement) from **control** (sequencing of operations on datapath)

Microcode program is essentially a description of a finite state machine for the ***control*** of a ***datapath***

- What is the state? uPC
- What are the outputs? Control signals
- What is the output logic? ucode ROM
- What is the input? Cond/busy, Opcode, fetch0, uBR
- What is the next state logic? uPC jump logic





µPC jump = next | spin | fetch | dispatch | ftrue | ffalse

Q2: Microcoded vs Pipelined

Microcode vs Pipelining

How does a microcoded single-bus machine differ from a classic RISC pipeline?

Why is a simpler microarchitecture generally possible with microcoding?

Q2: Microcoded vs Pipelined

Microcode vs Pipelining

How does a microcoded single-bus machine differ from a classic RISC pipeline?

- Consider how instruction execution phases are implemented
  - Instruction Fetch
  - Decode
  - Register Read
  - ALU operations
  - Memory operations
  - Register writebacks
  - Next PC calculation

Why is a simpler microarchitecture generally possible with microcoding?

-> Reusing datapath many purposes, ROM replaces HW

Q2: Microcoded vs Pipelined

Historical context: why did microcoding make sense historically?

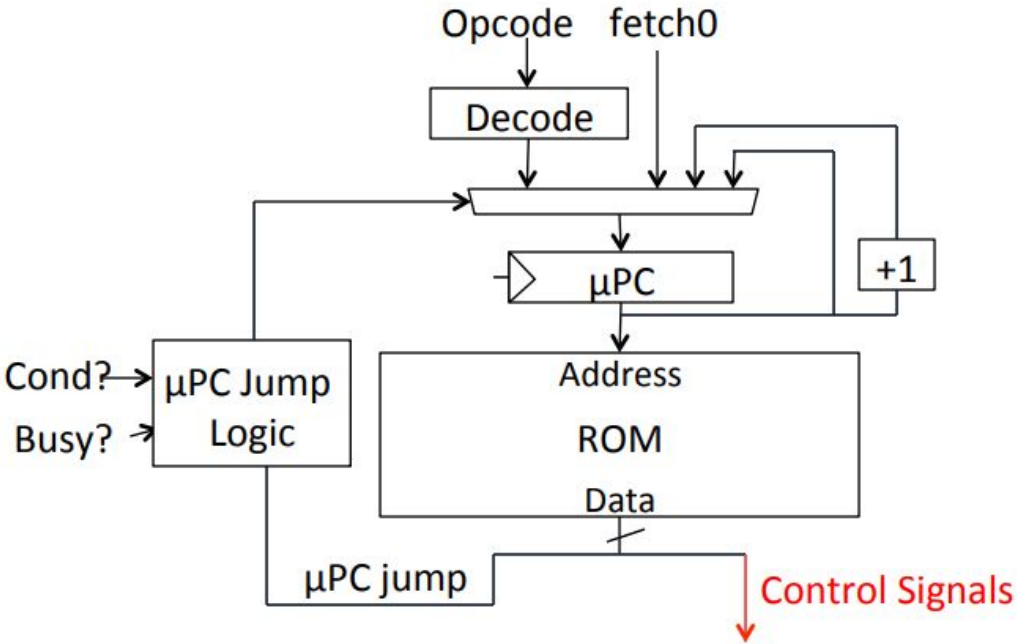
- ROM was cheaper and faster than logic
- Simplified control complexity (helps verification)
- CISC yielded better code density
- (Do these still hold true today?)

Recurring theme in architecture:

- Design choices are influenced by the limitations of available circuit technology
- Re-evaluation becomes necessary as technology changes

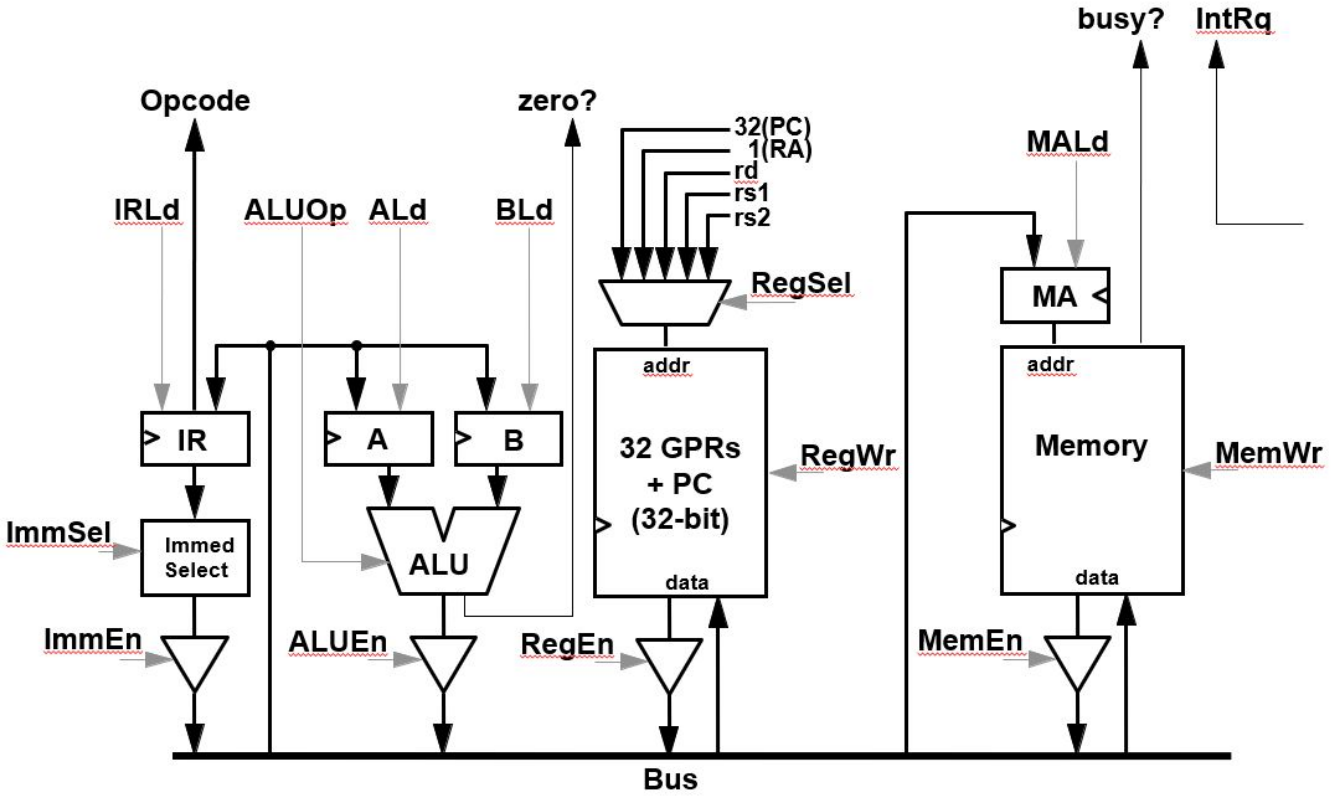
Avoid repeating past mistakes. . . but also know when old lessons may not apply

A Simple Microcoded Control Scheme



μPC jump = next | spin | fetch | dispatch | ftrue | ffalse

A Single-Bus Microcoded Datapath



Q3: Microprogramming

Practice:

Implement a conditional memory-to-memory move instruction in microcode for the single-bus RISC-V machine described in Handout #1. The instruction has the following format:

CMOVM (rd), (rs1), rs2

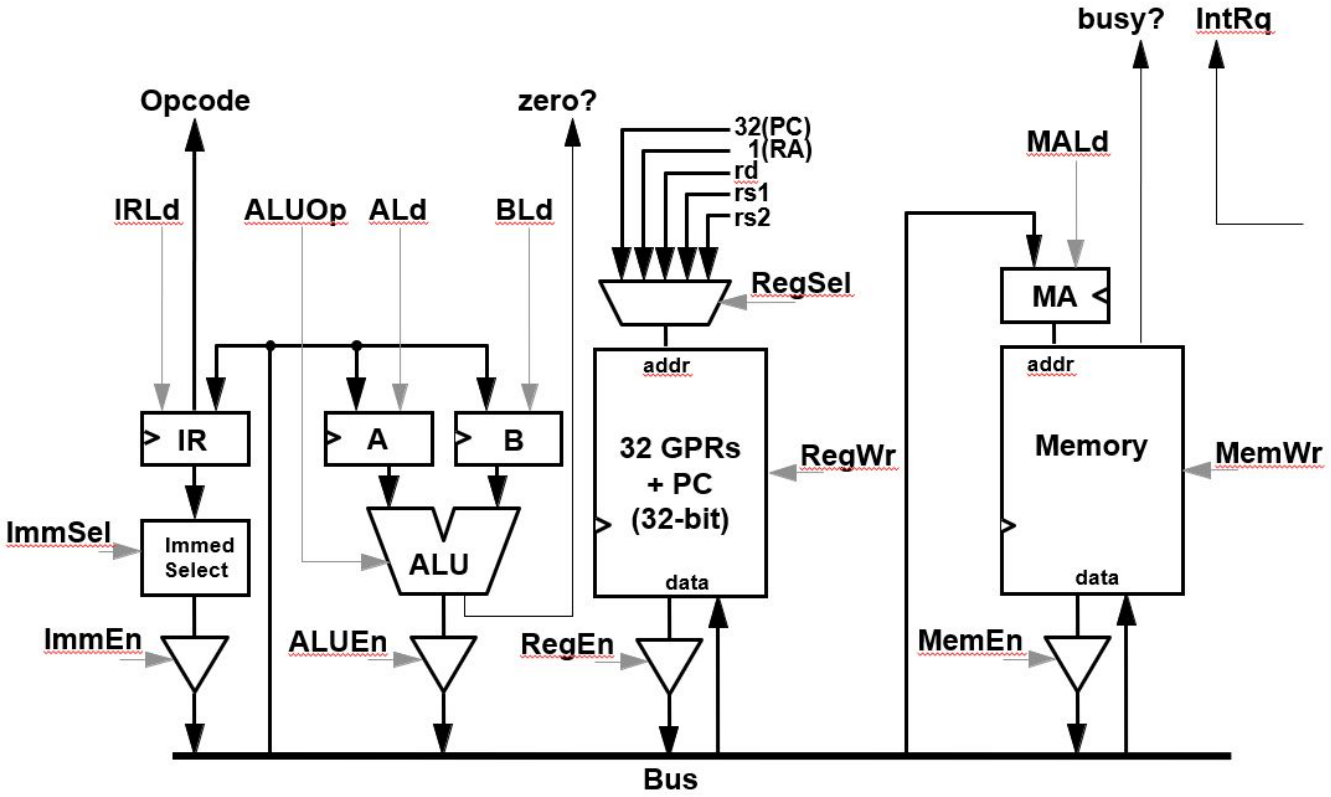
CMOVM performs the following operation: If the value in *rs2* is true (non-zero), then the memory word loaded from the address in *rs1* is stored to the address in *rd*.

```
if R[rs2] != 0
        M[R[rd]] := M[R[rs1]]
```

State	Pseudocode	ldIR	Reg Sel	RegWr	en Reg	ldA	ldB	ALU Op	en ALU	ld MA	Mem Wr	en Mem	Imm Sel	en Imm	µBr	Next State
FETCH0	MA := PC; A := PC	*	PC	0	1	1	*	*	0	1	0	0	*	0	N	*
	IR := Mem	1	*	0	0	0	*	*	0	0	0	1	*	0	S	*
	PC := A + 4	0	PC	1	0	0	*	INC_A_4	1	*	0	0	*	0	D	*
...																
NOP0	µBr to FETCH0	*	*	0	0	*	*	*	0	*	0	0	*	0	J	FETCH0
CMOVM0:																

State	Pseudocode	ldIR	Reg Sel	RegWr	en Reg	ldA	ldB	ALU Op	en ALU	ld MA	Mem Wr	en Mem	Imm Sel	en Imm	µBr	Next State
FETCH0	MA := PC; A := PC	*	PC	0	1	1	*	*	0	1	0	0	*	0	N	*
	IR := Mem	1	*	0	0	0	*	*	0	0	0	1	*	0	S	*
	PC := A + 4	0	PC	1	0	0	*	INC_A_4	1	*	0	0	*	0	D	*
...																
NOP0	µBr to FETCH0	*	*	0	0	*	*	*	0	*	0	0	*	0	J	FETCH0
CMOVM0:	A:= R[rs2]	0	rs2	0	1	1	*	*	0	*	0	0	*	0	N	
	MA := R[rs1] If (A==0) uBr to FETCH0	0	rs1	0	1	*	*	CopyA	0	1	0	0	*	0	EZ	FETCH0
	A := Mem	0	*	0	0	1	*	*	0	0	0	1	*	0	S	
	MA := R[rd]	*	rd	0	1	0	*	*	0	1	0	0	*	0	N	
	Mem := A	*	*	0	0	*	*	CopyA	1	0	1	0	*	0	S	
	uBr to FETCH0	*	*	0	0	*	*	*	0	*	0	0	*	0	J	FETCH0

A Single-Bus Microcoded Datapath



State	Pseudocode	ldIR	Reg Sel	RegWr	en Reg	ldA	ldB	ALU Op	en ALU	ld MA	Mem Wr	en Mem	Imm Sel	en Imm	μBr	Next State
FETCH0	MA := PC; A := PC	*	PC	0	1	1	*	*	0	1	0	0	*	0	N	*
	IR := Mem	1	*	0	0	0	*	*	0	0	0	1	*	0	S	*
	PC := A + 4	0	PC	1	0	0	*	INC_A _4	1	*	0	0	*	0	D	*
...																
NOP0	μBr to FETCH0	*	*	0	0	*	*	*	0	*	0	0	*	0	J	FETCH0
CMOVM0:	A := R[rs2]	0	RS2	0	1	1	*	*	0	*	0	0	*	0	N	
	MA := R[rs1] if (A == 0): goto FETCH0	0	RS1	0	1	*	*	COPY _A	0	1	0	0	*	0	EZ	FETCH0
	A := Mem	0	*	0	0	1	*	*	0	0	0	1	*	0	S	
	MA := R[rd]	*	RD	0	1	0	*	*	0	1	0	0	*	0	N	
	Mem := A	*	*	0	0	0	*	COPY _A	1	0	1	0	*	0	S	
	goto FETCH0	*	*	0	*	*	*	*	*	*	0	*	*	*	J	FETCH0

```
if R[rs2] != 0
    M[R[rd]] := M[R[rs1]]
```

Lab Logistics

CS152 partitioned into 5 modules

- 1 problem set per module (1.5 weeks per problem set)
- 1 lab per module (2.5 weeks per lab)

Labs are “take-home” assignments…

- No dedicated lab sessions (sometimes we will use end of discussion)
- Lab infrastructure (mostly) pre-installed on instructional Linux EDA machines
- Log in through ssh

TODO for you:

- **Request EECS instructional account**
- https://acropolis.cs.berkeley.edu/~account/webacct/

Lab Reports

Lab reports must be **typed in readable English** - no raw dumps!

Directed portion

- Complete independently
- Submit one report per person on Gradescope

Open-ended portion

- Pick *one* problem to do
- Work in groups of 1-3
- Submit one report on Gradescope (<= 2 pages for individual, 3 for group)

Lab Feedback

Write as much or as little as desired.

- **Point deduction if omitted**
- Answer team feedback questions
- Append feedback to individual report

We care about your feedback!

- Scope of class precludes design projects
- We want labs to be interesting, but not tedious

Lab 1

Due on Wednesday Feb 7 @ Midnight

- **Start early**! Some steps may take longer than expected
- Gentle introduction to simulation and software tools
- Given source code for simple processor designs
- Build cycle-accurate RTL simulators
- Run benchmarks, gather data, analyze data
- Make recommendations, write adversarial code, propose new designs

Lab Infrastructure Overview

CS152 labs are built on current research infrastructure (designs, tools, flows)

Developed from scratch at UC Berkeley over past 10 years

We will strive to use real hardware implementations wherever possible

- More interesting than a contrived pure-software model
- Enables more realistic experiments
- Actual systems sometimes exhibit counter-intuitive behavior

Berkeley Architecture Research

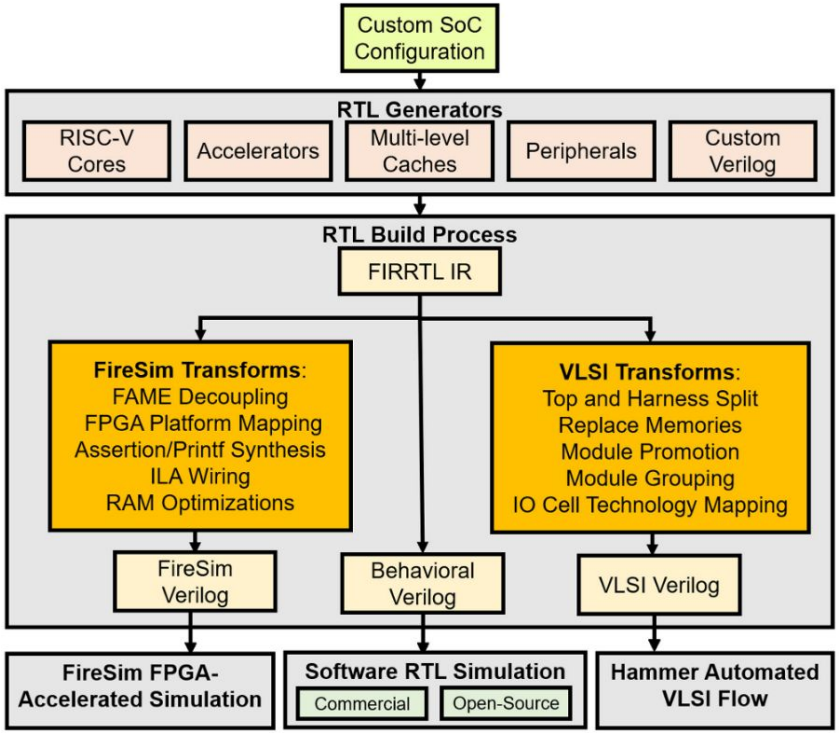
Chipyard

Integrated environment for

designing system-on-chips (SoCs)

- Educational Sodor cores
- In-order, out-of-order cores
- Accelerators, caches
- Multicore

Your labs will all run in Chipyard



Chisel

New hardware description language designed at UC Berkeley

Embedded in Scala

- Designers write Scala programs that generate hardware
- Compiler framework elaborates Scala source into Verilog RTL
- Elegant, concise, powerful compared to Verilog

NOTE: Students will NOT be required to implement hardware in Chisel!

- Processor designs will be provided, you will need to parameterize
- Can choose to write Chisel RTL for some lab open-ended questions
- Students are encouraged to try to read the Chisel source

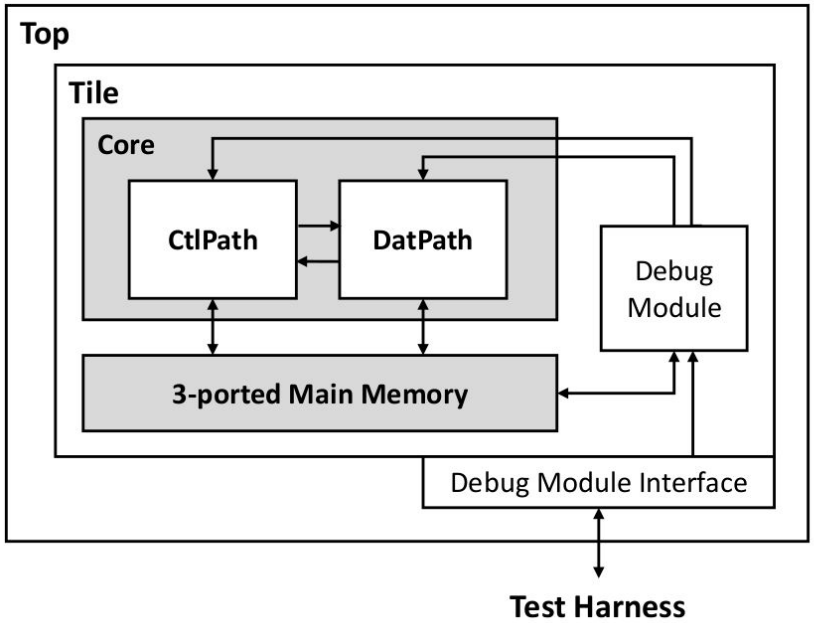
Sodor

Collection of educational RV32I cores

- 1-stage, 2-stage, 3-stage, 5-stage
- Microcoded bus-based design

Educational, favors clarity over power/perf/area

- ~1000 LoC per core



Tips: Use `tmux` for Labs

Labs will involve running simulations of real SoCs for millions of cycles

These simulations are slow, don't let your simulation get killed by a bad internet connection

Use TMUX!

- Create tmux session: `tmux`
- Detach from tmux session: `ctrl-b d`
- Attach to tmux session: `tmux a -t <id>`
  - Ex: `tmux a -t 0`