DINO CPU

A Teaching-Focused RISC-V Design in Chisel

UCDAVIS

https://github.com/jlpteaching/dinocpu

Jason Lowe-Power
@JasonLowePower
Christopher Nitta

DINO CPU

A suite of **RISC-V CPU designs**
- Single cycle
- Five stage pipeline
  - + Branch predictor

All designs can run rv32i code compiled with mainline GCC

A set of **assignments**
- One for each design

Tools for classroom use
- Chisel development
- Auto grading

Open source
https://github.com/jlpteaching/dinocpu

https://chisel.eecs.berkeley.edu/

Open source hardware construction language

Embedded in Scala

Main benefit: Parameterizable

Used in industry: SiFive, Google, IBM, others…

Why Chisel?

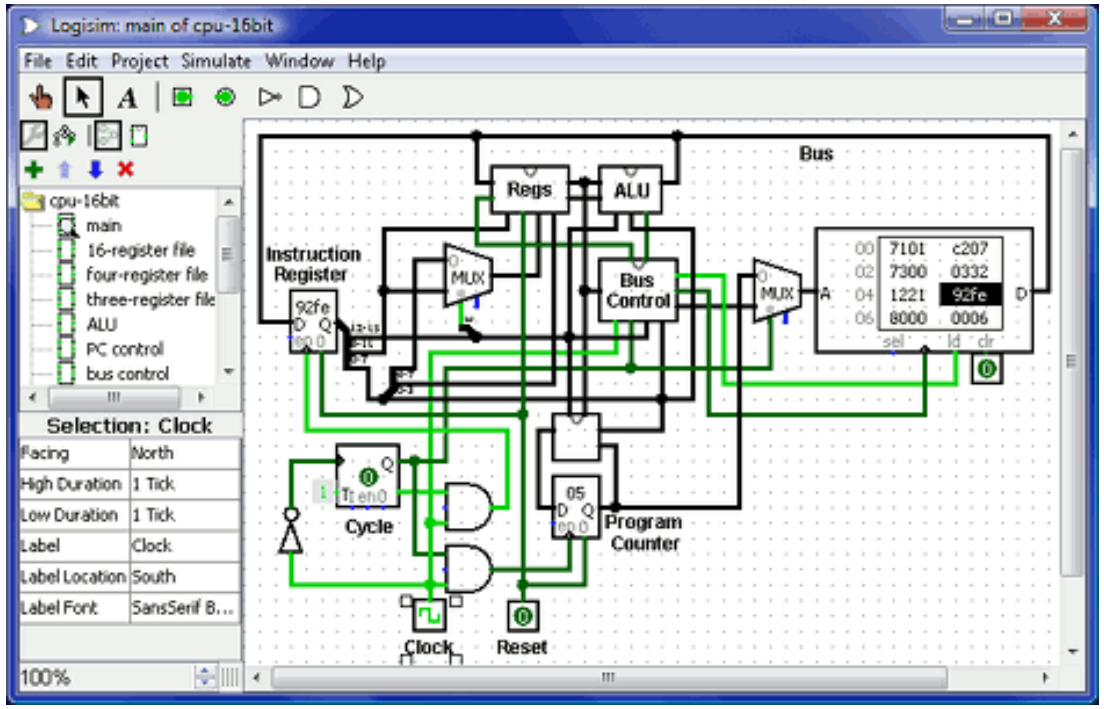
Vs Logisim

From an evaluation:

“I hate Logisim with a passion”

Scaling designs difficult

Grading time consuming



Why Chisel?

Vs Verilog

More modular design

Built-in unit tests
+ easy to add auto grading

Scala-based
More familiar (to me)


```
module RISCVCPU (clock);
  // Instruction opcodes
  parameter LD = 7'b000_0011, SD = 7'b010_0011, BEQ = 7'b110_0011, NOP =
32'h0000_0013, ALUop = 7'b001_0011;
  input clock;

  reg [63:0] PC, Regs[0:31], IDEXA, IDEXB, EXMEMB, EXMEMALUOut,
MEMWBValue;
  reg [31:0] IMemory[0:1023], DMemory[0:1023], // separate memories
             IFIDIR, IDEXIR, EXMEMIR, MEMWBIR; // pipeline registers
  wire [4:0] IFIDrs1, IFIDrs2, MEMWBrd; // Access register fields
  wire [6:0] IDEXop, EXMEMop, MEMWBop; // Access opcodes
  wire [63:0] Ain, Bin; // the ALU inputs

  // These assignments define fields from the pipeline registers
  assign IFIDrs1  = IFIDIR[19:15];  // rs1 field
  assign IFIDrs2  = IFIDIR[24:20];  // rs2 field
  assign IDEXop   = IDEXIR[6:0];    // the opcode
  assign EXMEMop  = EXMEMIR[6:0];   // the opcode
  assign MEMWBop  = MEMWBIR[6:0];   // the opcode
  assign MEMWBrd  = MEMWBIR[11:7];  // rd field
  // Inputs to the ALU come directly from the ID/EX pipeline registers
  assign Ain = IDEXA;
  assign Bin = IDEXB;

  integer i; // used to initialize registers
  initial
  begin
    PC = 0;
    IFIDIR = NOP; IDEXIR = NOP; EXMEMIR = NOP; MEMWBIR = NOP; // put NOPs
in pipeline registers
    for (i=0;i<=31;i=i+1) Regs[i] = i; // initialize registers--just so
they aren't cares
  end

  // Remember that ALL these actions happen every pipe stage and with the
use of <= they happen in parallel!
  always @(posedge clock)
  begin
    // first instruction in the pipeline is being fetched
    // Fetch & increment PC
    IFIDIR <= IMemory[PC >> 2];
    PC <= PC + 4;

    // second instruction in pipeline is fetching registers
    IDEXA <= Regs[IFIDrs1]; IDEXB <= Regs[IFIDrs2]; // get two registers
    IDEXIR <= IFIDIR; // pass along IR--can happen anywhere, since this
affects next stage only!

    // third instruction is doing address calculation or ALU operation
    if (IDEXop == LD)
      EXMEMALUOut <= IDEXA + {{53{IDEXIR[31]}}, IDEXIR[30:20]};
    else if (IDEXop == SD)
      EXMEMALUOut <= IDEXA + {{53{IDEXIR[31]}}, IDEXIR [30:25],
IDEXIR[11:7]};
    else if (IDEXop == ALUop)
      case (IDEXIR[31:25]) // case for the various R-type instructions
        0: EXMEMALUOut <= Ain + Bin;  // add operation
```

FIGURE e4.13.1 A Verilog behavioral model for the RISC-V five-stage pipeline, ignoring

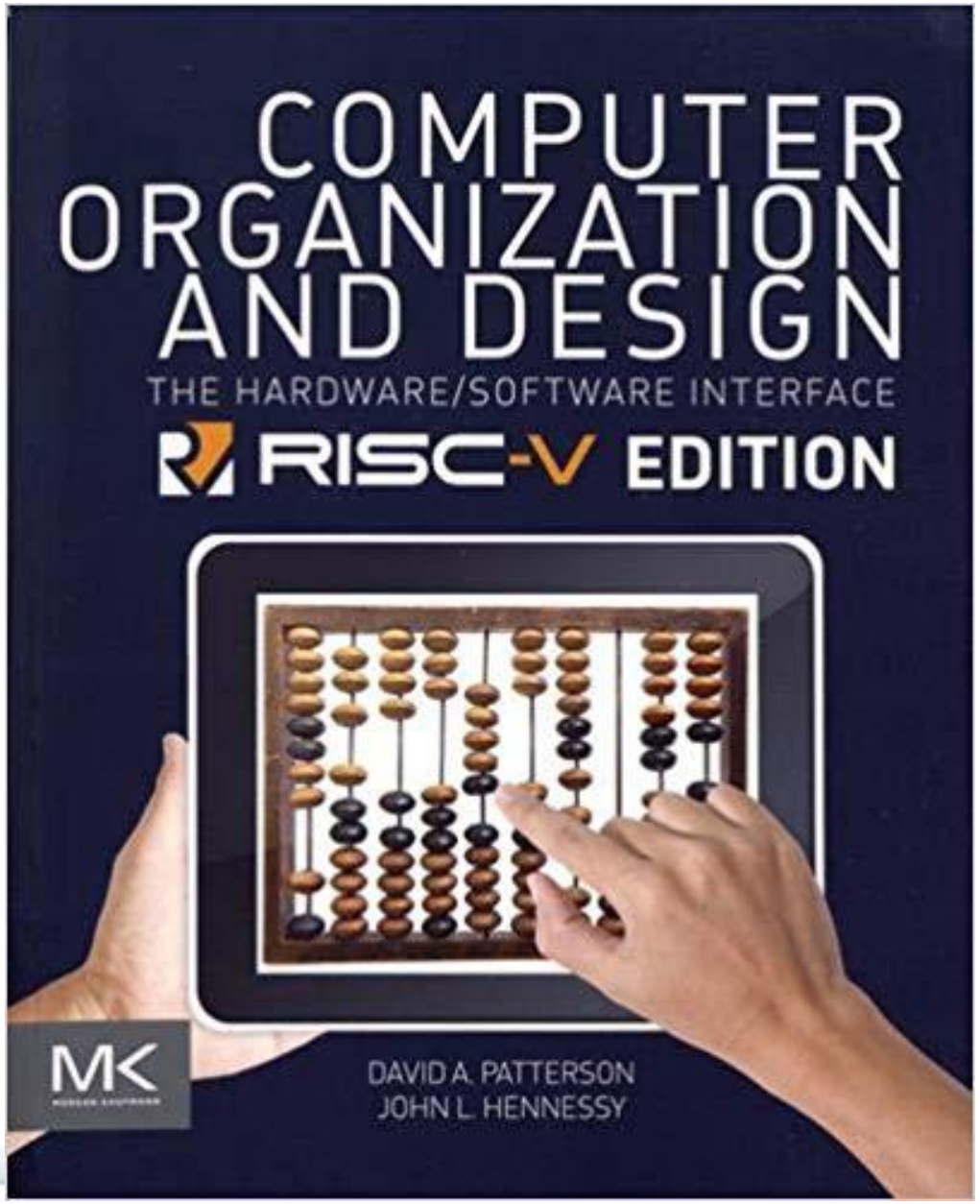
UCDAVIS

DINO CPU Design

Closely follows Patterson and Hennessy's textbook's design

Simple and modular
- Not fast, small, synthesizable

Complete-ish
- Hide complexity when possible



DINO CPU Assignment 1: R-types

Step 1: Logic for ALU control

```
/**
 * The ALU control unit
 *
 * Input:  add, if true, add no matter what the other bits are
 * Input:  immediate, if true, ignore funct7 when computing the operation
 * Input:  funct7, the most significant bits of the instruction
 * Input:  funct3, the middle three bits of the instruction (12-14)
 * Output: operation, What we want the ALU to do.
 *
 * For more information, see Section 4.4 and A.5 of Patterson and Hennessy
 * This follows figure 4.12
 */
class ALUControl extends Module {
  val io = IO(new Bundle {
    val add       = Input(Bool())
    val immediate = Input(Bool())
    val funct7    = Input(UInt(7.W))
    val funct3    = Input(UInt(3.W))

    val operation = Output(UInt(4.W))
  })

  io.operation := 15.U // invalid operation

  // Your code goes here
}
```

The following table details the `operation` input and which values produce which results.

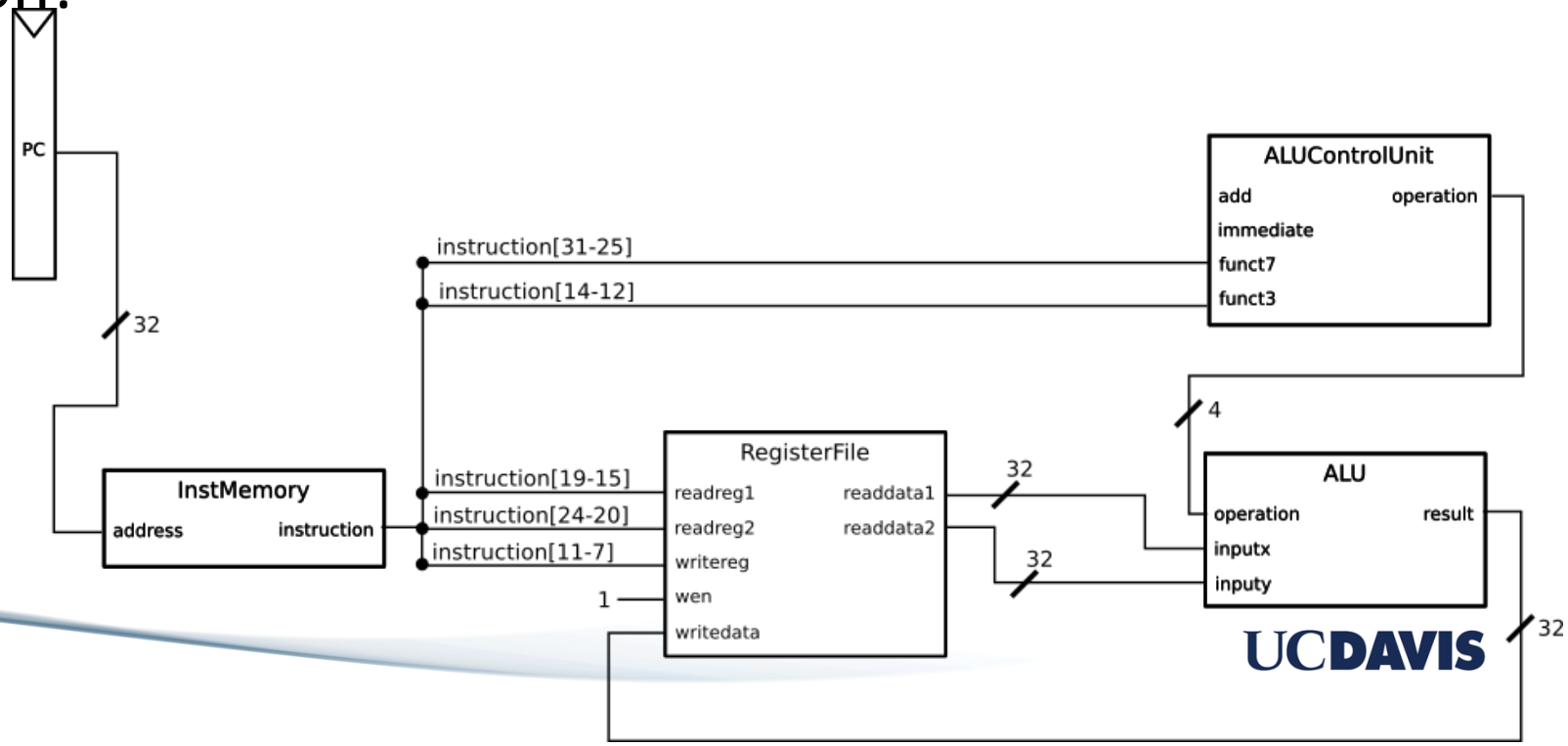
0000	and
0001	or
0010	add
0011	sub
0100	slt
0101	sltu
0110	sll
0111	srl
1000	sra
1001	xor

31--25	24--20	19--15	14--12	11--7	6--0	
funct7	rs2	rs1	funct3	rd	opcode	R-type
-------	------	------	------	-----	---------	--
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

DINO CPU Assignment 1: R-types

Step 1: Logic for ALU control

Step 2: Draw R-type circuit
Before implementation!

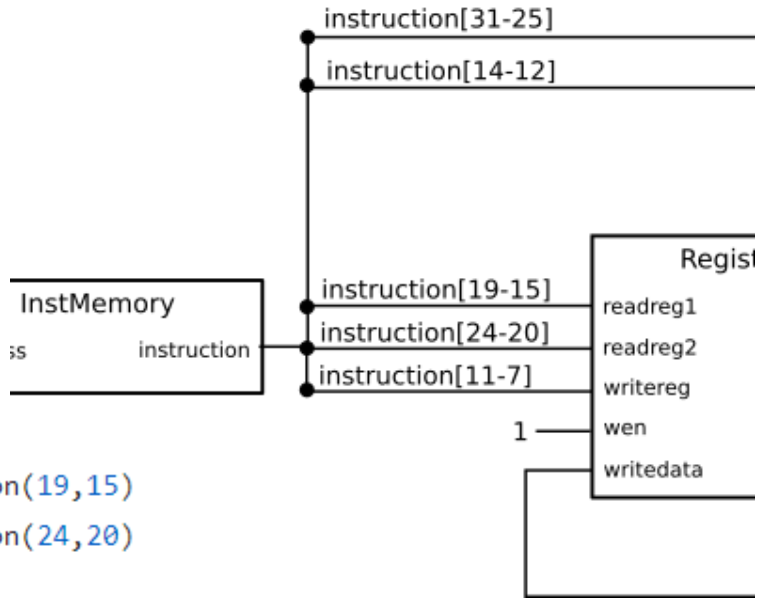


DINO CPU Assignment 1: R-types

Step 1: Logic for ALU control

Step 2: Draw R-type circuit
Before implementation!

Step 3: Write Chisel



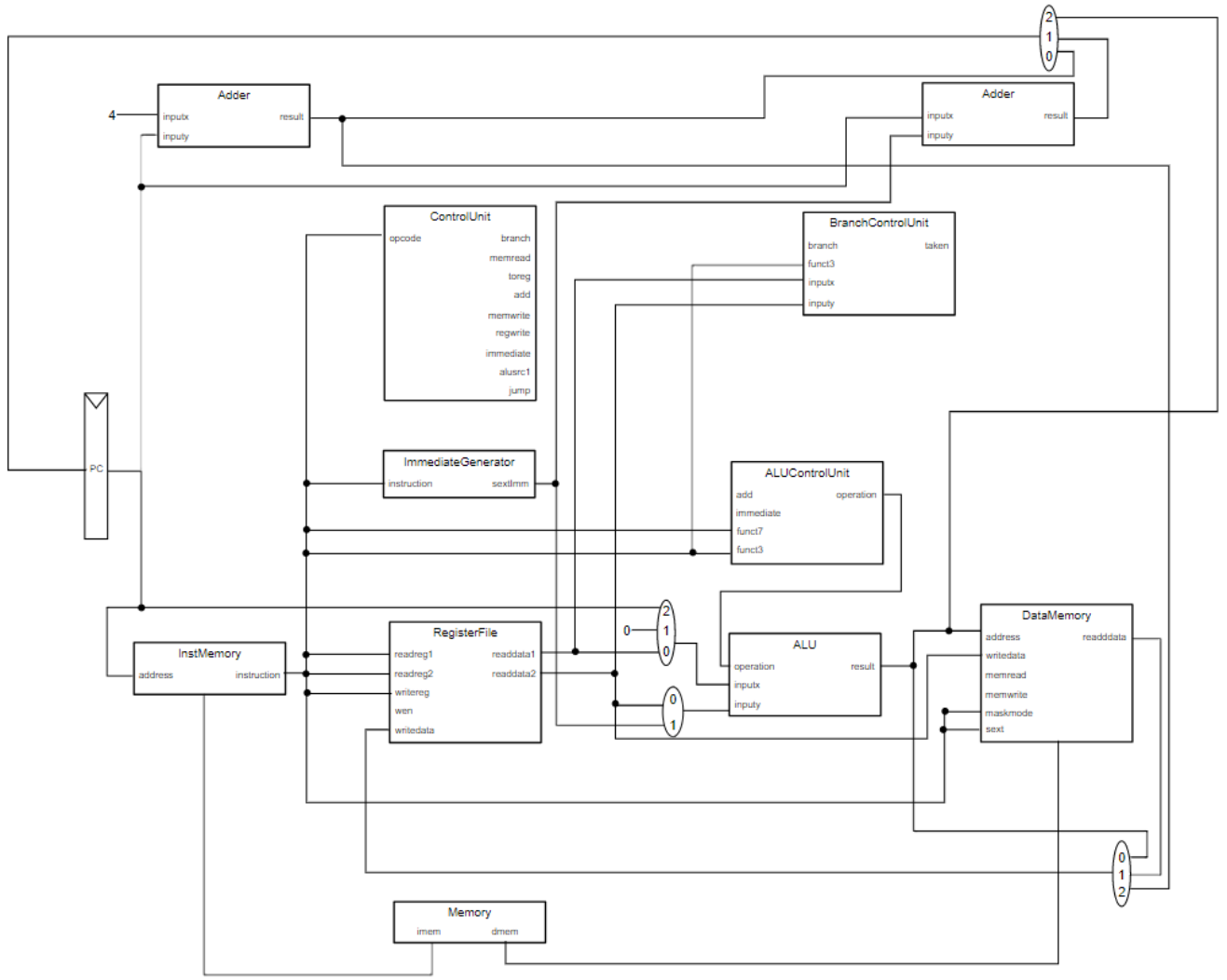

```
registers.io.readreg1 := instruction(19,15)
registers.io.readreg2 := instruction(24,20)

val writereg = instruction(11,7)
registers.io.writereg := writereg
registers.io.wen := true.B

aluControl.io.add       := false.B
aluControl.io.immediate := false.B
aluControl.io.funct7    := instruction(31,25)
aluControl.io.funct3    := instruction(14,12)
```

DINO CPU Assignment 2: Single-cycle

Given diagram:

- Implement control
- Wire control lines
- Wire whole datapath

Assignment takes one instruction type at a time

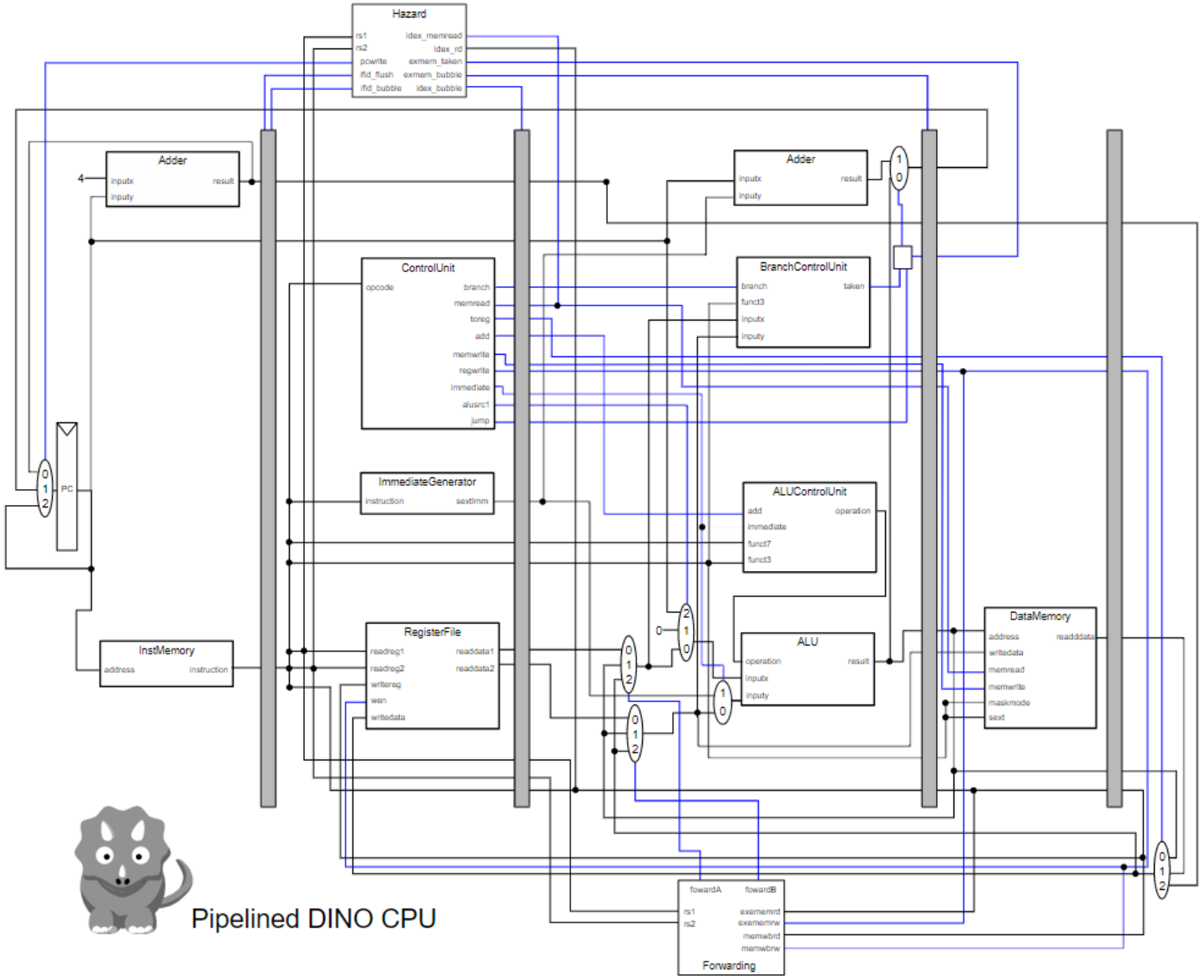


DINO CPU Assignment 3: Pipelined

Significant increase in complexity

Define all pipeline registers

Implement hazard and forwarding logic



Pipelined DINO CPU

DINO CPU Assignment 4: Branch prediction

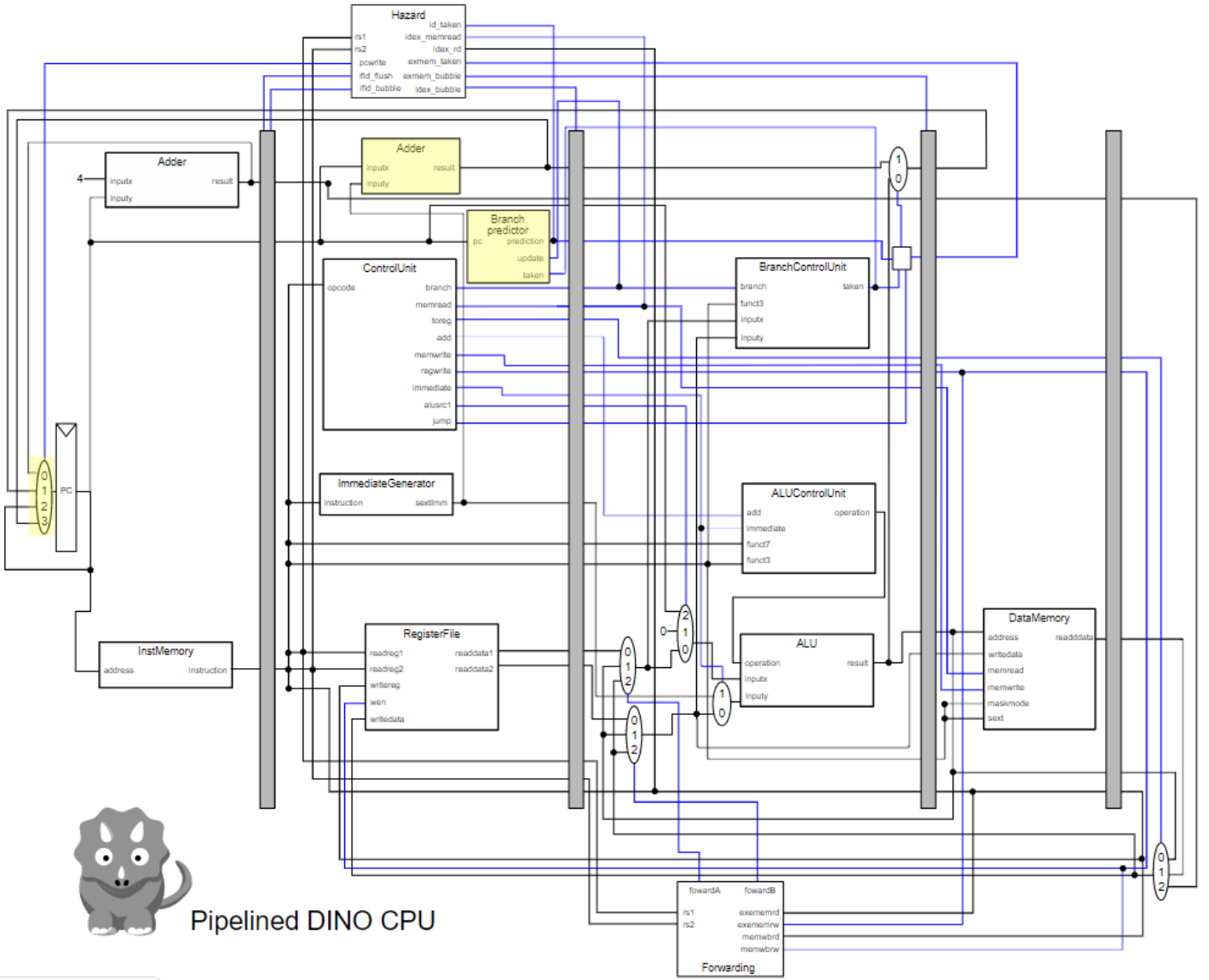
Extensions!

- We chose branch prediction

We updated pipeline

Students implemented two predictors

Ran benchmarks and compared results



Pipelined DINO CPU

Tools included

Singularity container
- Many dependencies
- Safe and secure

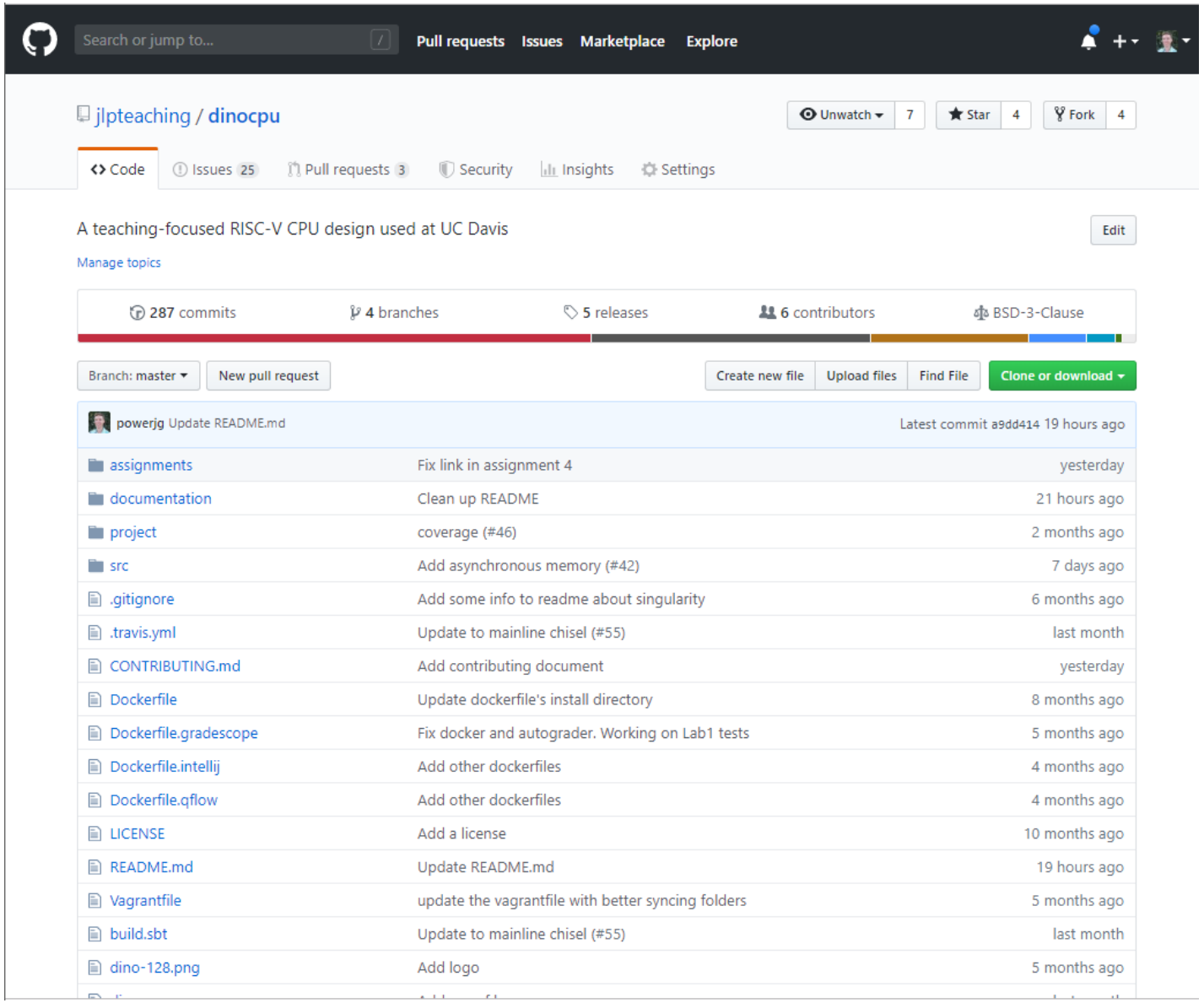
Gradescope scripts
- Autograder

Open source

Everything on GitHub

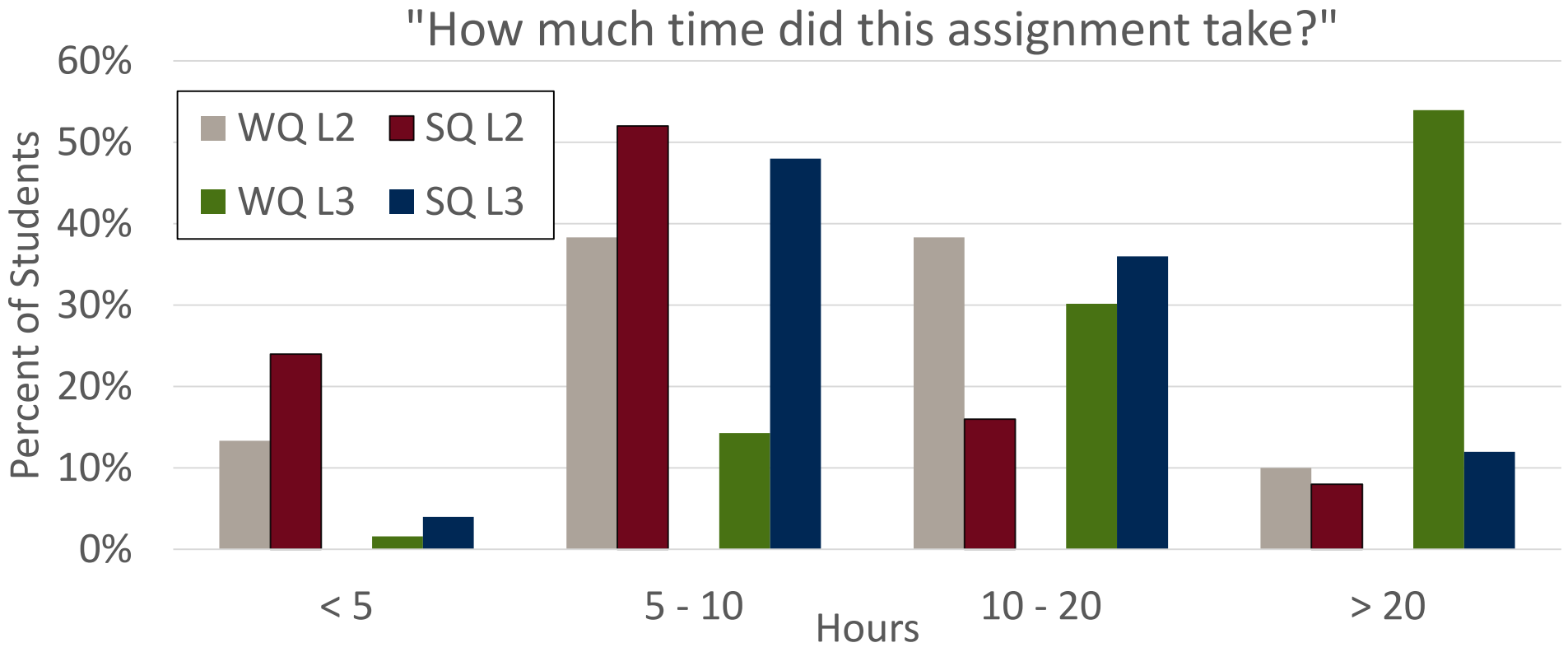
https://github.com/jlpteaching/dinocpu

- Assignments
- Documentation
- Source code
- Tools



Future Improvements

Main feedback: Need better debugging



Future Improvements

Main feedback: Need better debugging

More RISC-V support
- Privileged ISA for machine-mode rv32i (e.g., for embedded)

More assignments
- Non combinational memory + cache
- Multi-issue?

Open source community!

Questions/Comments?

Thanks to:

Students of ECS154B WQ19

Jared Barocsi, Filipe Eduardo Borges, Nima Ganjehloo, Daniel Grau, Markus Hankins, and Justin Perona

DINO CPU

A TEACHING-FOCUSED RISC-V DESIGN IN CHISEL

UCDAVIS

https://github.com/jlpteaching/dinocpu

Jason Lowe-Power

@JasonLowePower

Christopher Nitta