ECS 154B Discussion 1

January 7th, 2022

Goals

- An overview of Chisel.
- How to start assignment 1.

Offices Hours, Campuswire, Gradescope stuff

- Offices Hours: TBD
- For questions about homework assignments -> ideally, Campuswire
  - Public questions: might get help quicker.
  - Private questions: questions that expose part of your answer, or not.
- Submissions: via Gradescope for both diagram and code.
  - Need to setup grading stuff, will be available later.

Resources for Singularity, Chisel, and RISC-V

- Setting up singularity: https://github.com/jlpteaching/dinocpu-wq22/blob/main/documentation/singularity.md
- Chisel docs: https://www.chisel-lang.org/chisel3/docs/introduction.html
- RISC-V specs: https://riscv.org/technical/specifications/
- Quick RISC-V specs: http://riscvbook.com/
- Search Engine Usage: better results using “chisel lang”.

Chisel

- Hardware description language (HDL) for designing digital circuits.
  - Other HDLs are Verilog, VHDL, etc.
- Built on top of Scala.
- Uses the power of a modern programming language to describe complex and parameterized circuits.

Why Chisel?

- Design hardware via a high level programming language.
- Allow developing hardware as a test-driven development process.
- Allow fast development process.
- The industry (and academia) uses it.
  - Google Edge TPU
  - SiFive

Chisel Basics

- https://github.com/jlpteaching/dinocpu-wq22/blob/main/documentation/chisel-notes/cheat-sheet.md
- https://github.com/freechipsproject/chisel-cheatsheet/releases/latest/download/chisel_cheatsheet.pdf

Chisel Basics

- Boolean

```
val x = Bool() // declare a variable
x := true.B    // converting Scala's true to Chisel
x := false.B   // converting Scala's false to Chisel
```

- Integers

```
UInt(32.W)  // Chisel 32-bit unsigned integer
UInt()      // Chisel unsigned integer with an unknown width (will be inferred)
77.U        // converting Scala int literal (32 bits) to Chisel unsigned int
77L.U       // converting Scala long literal (64 bits) to Chisel unsigned int
3.S(2.W)    // Chisel 2-bit signed integer (e.g. -1)
"b01011".U  // converting a string of binary literal to Chisel unsigned int
```

Chisel Basics

- Arithmetics, e.g.

```
val x = UInt(3.W)
val y = x + 2.U // Chisel unsigned ints addition
val z = x - 2.U // Chisel unsigned ints subtraction
val t = x >> 2.U // Chisel unsigned int shift right
```

- Comparisons,

```
x === y   // equality between two Chisel objects
x =/= 3.U // inequality between two Chisel objects
```

- More complete cheat sheet: https://github.com/freechipsproject/chisel-cheatsheet/releases/latest/download/chisel_cheatsheet.pdf

Chisel Basics

- State elements (registers)

```
val x = Reg(UInt(64.W))     // x is a 64-bit register
val y = RegInit(1.U(32.W)) // y is a 32-bit register
                            // y has the value of 1 when the system starts
```

- Setting/Getting value of registers

```
// Suppose z is a wire, and x is a register
x := z // “assigning” value of the wire z to the register x
       // e.g. z is wired to input of x, value of x changes at the end of cycle
z := x // “assigning” value of register x to the wire z
       // e.g. z is wired to output of x, value of z is the value coming out of x
```

Chisel Basics

- Creating a new wire: `val x = Wire(UInt())`
- Connecting two wires: `y := x`
- Getting a bit from a wire:

```
x(31) // get bit 32 from wire x (bit 0 is the least significant bit)
```

- Getting multiple bits from a wire:

```
x(6, 0) // get bits from bit 6 to bit 0 (inclusive)
```

- Concatenate bits:

```
// Suppose y = "b01110001"
                    ^^^^  ^
                  y(6,3)  y(0)
val x = Cat(y(6, 3), y(0)) // x is "b11101"
```

Chisel Basics

- Branching: when-elsewhen-otherwise

```
when(value === 0.U) { x := “b001”.U }
.elsewhen(value > 0.S) { x := “b010”.U }
.otherwise { x := “b100”.U }
```

- Mux:

```
x := Mux(selector, true_value, false_value)
```

- Mux is similar to:

```
when(selector === 1.U) { x := true_value }
.otherwise { x := false_value }
```

Chisel Basics

- Bundles : grouping a set of wires (e.g., grouping I/O).
- E.g.,

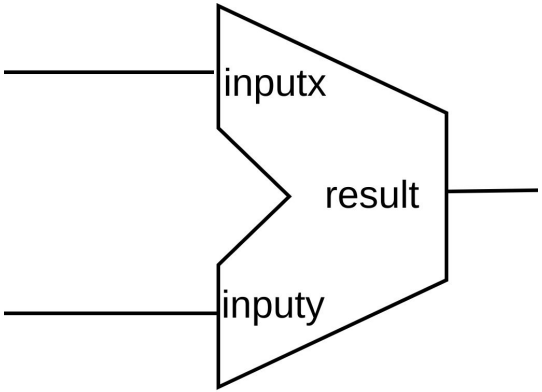
```
val io = IO(new Bundle{
    val inputx = Input(UInt(64.W))
    val inputy = Input(UInt(64.W))
    val result = Output(UInt(64.W))
  })
```

- Getting the output:

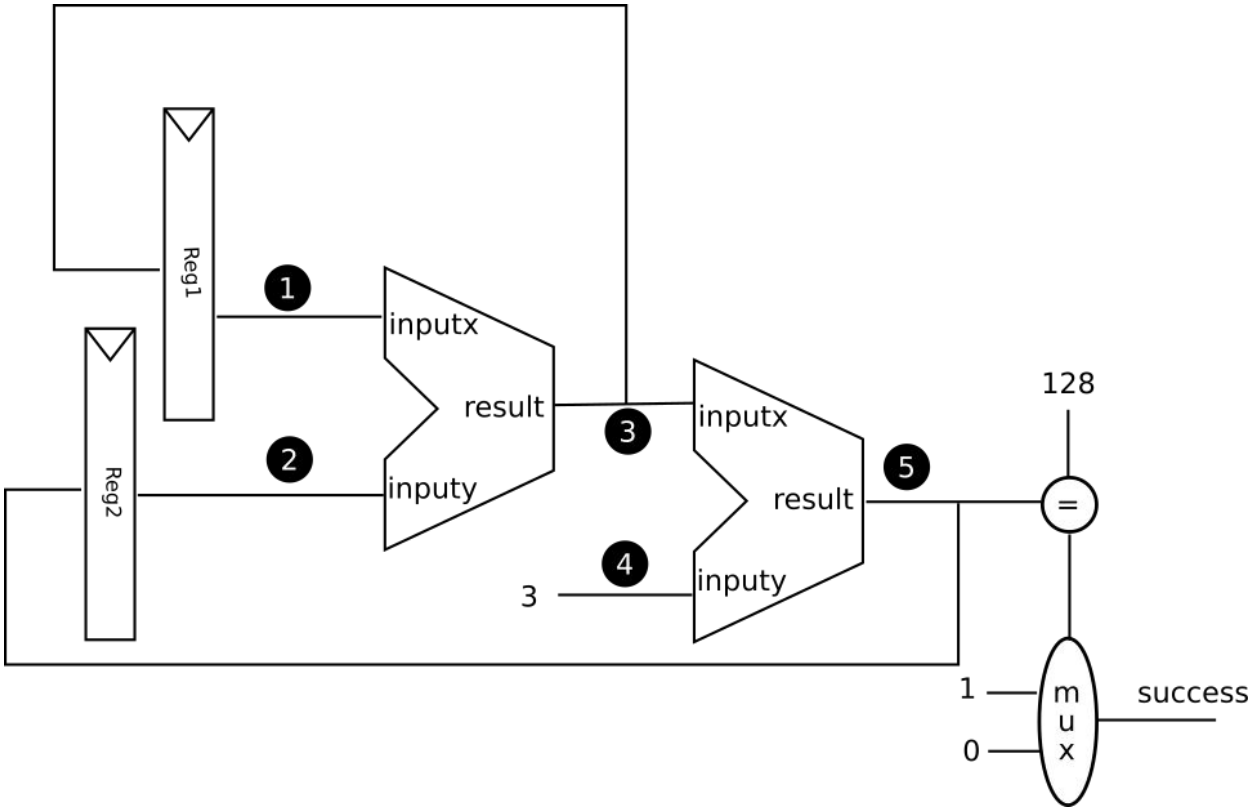
```
x := io.result
```

- Setting the inputs:

```
io.inputx := y
```



Simple Example

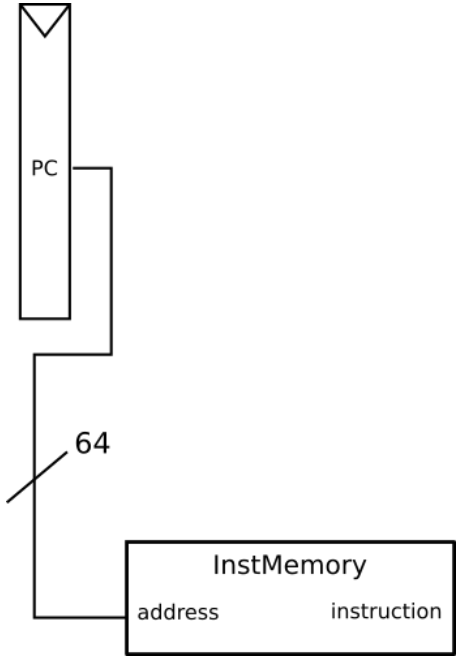


Notes

- For all assignments,
  - RISC-V instructions are 32 bits wide.
  - Registers are 64 bits wide.
  - Addresses are 64 bits wide.

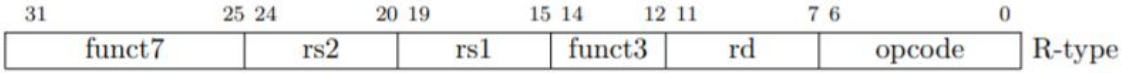
Assignment 1 Notes

- Width of each wire must be specified.



Assignment 1 Notes

- R-type instruction format



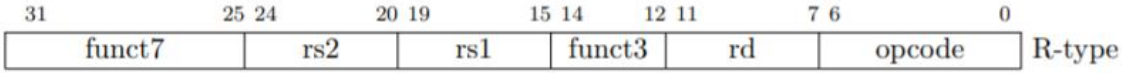
- Meaning,

  reg[rd] = reg[rs1] <op> reg[rs2]

- <op> is determined by funct3 and funct7

Assignment 1 Notes

- R-type instruction format



- Meaning,

  reg[rd] = reg[rs1] <op> reg[rs2]

- <op> is determined by funct3 and funct7

Assignment 1 Notes: diagram

- Each component has its inputs and outputs described in its the scala file.
  - https://github.com/jlpteaching/dinocpu-wq22/blob/main/src/main/scala/components/alucontrol.scala
- You don’t need to modify the Control Unit, and you also don’t need to wire more wires from/to Control Unit.

Assignment 1 Notes: implementing

- The diagram should be useful.
- If the CSIF machine you are working on does not have singularity, you can use this command,
  - /home/jlp/singularity-ce/bin/singularity run library://jlowepower/default/dinocpu

Assignment 1 Notes: testing

- There's a command for testing each part at the end of each part.
  - E.g., part 1: `Lab1 / testOnly dinocpu.SingleCycleAddTesterLab1`
- The risc-v binaries (and their assembly) used for testing can be found at: https://github.com/jlpteaching/dinocpu-wq22/tree/main/src/test/resources/risc-v
  - E.g. assembly of add1: https://github.com/jlpteaching/dinocpu-wq22/blob/main/src/test/resources/risc-v/add1.riscv
- Inputs and expected outputs for each test: https://github.com/jlpteaching/dinocpu-wq22/blob/main/src/main/scala/testing/InstTests.scala

Assignment 1 Notes: testing

- Understanding RISC-V assembly:
  - Register names: figure on the right
  - Instructions:
    - Format: `<op> rd, rs1, rs2`
    - E.g.: `add t0, t1, t2`
    - Meaning: t0 = t1 + t2
    - All t0, t1, t2 are treated as 64-bit integers.
    - The values of t1 and t2 do not change.

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5	t0	Temporary/alternate link register	Caller
x6–7	t1–2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function arguments/return values	Caller
x12–17	a2–7	Function arguments	Caller
x18–27	s2–11	Saved registers	Callee
x28–31	t3–6	Temporaries	Caller
f0–7	ft0–7	FP temporaries	Caller
f8–9	fs0–1	FP saved registers	Callee
f10–11	fa0–1	FP arguments/return values	Caller
f12–17	fa2–7	FP arguments	Caller
f18–27	fs2–11	FP saved registers	Callee
f28–31	ft8–11	FP temporaries	Caller

Assignment 1 Notes: testing

- RISC-V 64-bit word instructions:
  - Has suffix of “`w`”; e.g. `addw`
  - Format: `<op> rd, rs1, rs2`
  - E.g.: `addw t0, t1, t2`
  - Meaning: t0 = t1 + t2
  - t1 and t2 are treated as 32-bit integers (i.e. only bit 31 -> bit 0 are used for both registers).
  - The 32-bit result of (t1+t2) sign extended to 64-bit then assigned to t0.
  - The values of t1 and t2 do not change.
  - The ALU has already implemented this.

Assignment 1 Notes: debugging

- `dinocpu.singlestep` debugger: https://github.com/jlpteaching/dinocpu-wq22/blob/main/documentation/single-stepping.md
- `printf` statements: https://github.com/jlpteaching/dinocpu-wq22/blob/main/documentation/chisel-notes/printf-debugging.md

Assignment 1 Notes

- Start early.
- Campuswire.