

A3. Mini-RISC-V-Processor

CSE261: Computer Architecture, Fall 2021

Hyungon Moon

Out: Nov 9, 2021

Due: Nov 18 2021 11:59pm

Goal

- Implement the mini-RISC-V processor that runs 5–7 instructions.
- Mandatory: five instructions
 - `addi`, `add`, `ld`, `sd`, `beq`
- Optional: two instructions
 - `jal`, `jalr`

Vagrant and VirtualBox

- Vagrant allows you to easily create and access a virtual machine.
- Install Vagrant from

```
https://www.vagrantup.com/
```

- Vagrant (in this class) need VirtualBox. Install from

```
https://www.virtualbox.org/
```

- Should work on Ubuntu, Mac or Windows.
- May not work inside another virtual machine.

Initializing VM (Mac, Ubuntu (Linux))

- Use terminal to enter the hw1 directory
- Make sure that you are seeing `Vagrantfile` in the directory
- Create and boot VM

```
vagrant up
```

- Connect to the VM

```
vagrant ssh
```

- From V2 of the assignment, the hw1 directory is automatically synced with the hw directory

Initializing VM (Windows)

- Use terminal to enter the hw1 directory
- Make sure that you are seeing `Vagrantfile` in the directory
- Create and boot VM

```
vagrant.exe up
```

- Connect to the VM

```
vagrant.exe ssh
```

- From V2 of the assignment, the hw1 directory is automatically synced with the hw directory

SBT

- We will use a complex set of tools enabling us to write in Chisel.
- We don't need to care about them: SBT does the dirty job.
 - Located in hw/sbt or sbt.
- You will use three commands, referring to the lecture slides.
 - To run the Main object (and test the core in this assignment)

```
sbt/bin/sbt run
```

- To run all tests,

```
sbt/bin/sbt test
```

- To run one test only (e.g., RegisterFileTest)

```
sbt/bin/sbt "testOnly RegisterFileTest"
```

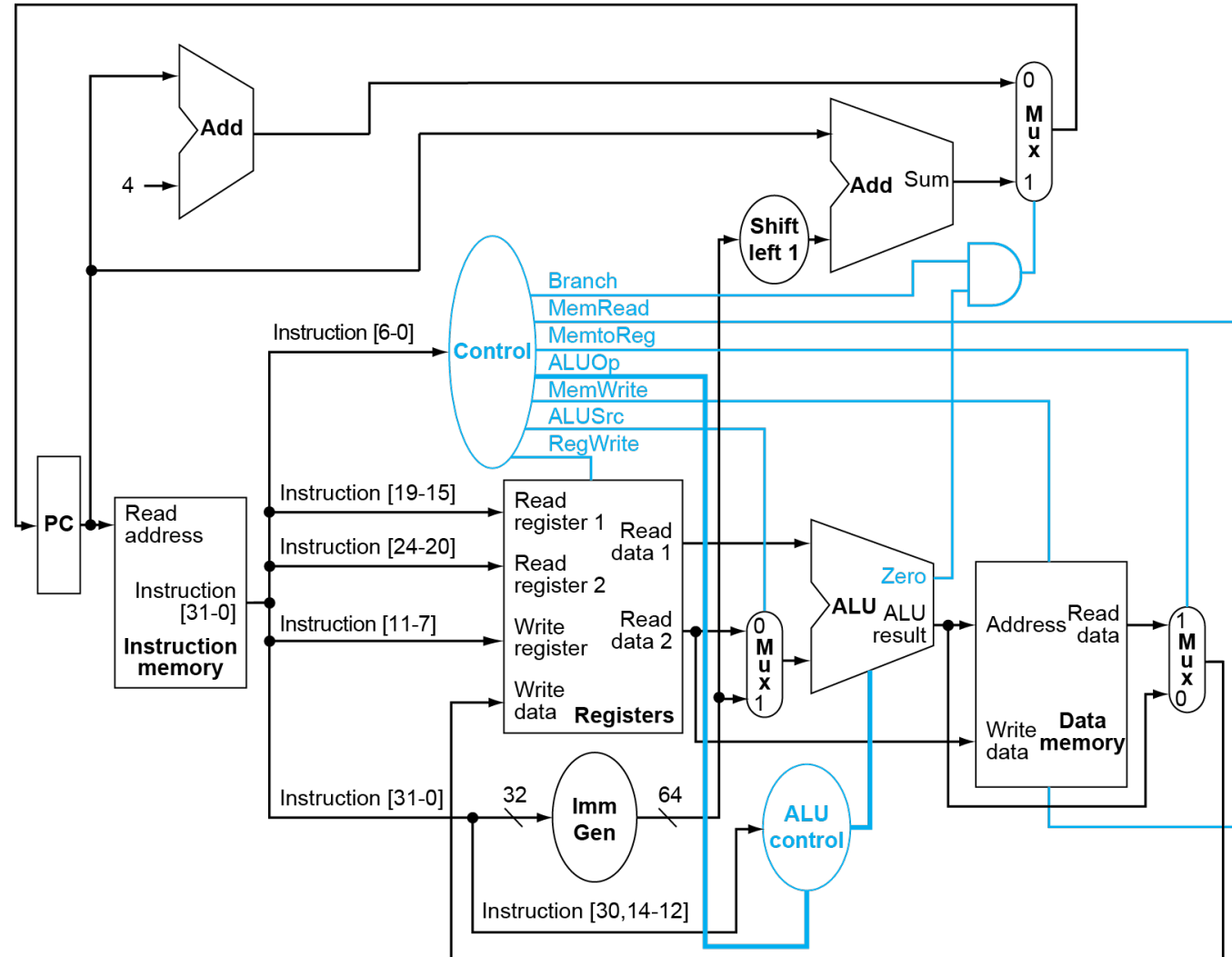
SBT trouble shooting

- When running sbt for the first time, you may see this message

```
java.io.IOException: org.scalasbt.ipcsocket.NativeErrorException:  
[1] Operation not permitted  
Create a new server? y/n (default y)
```

- Just press enter to go for the default value.

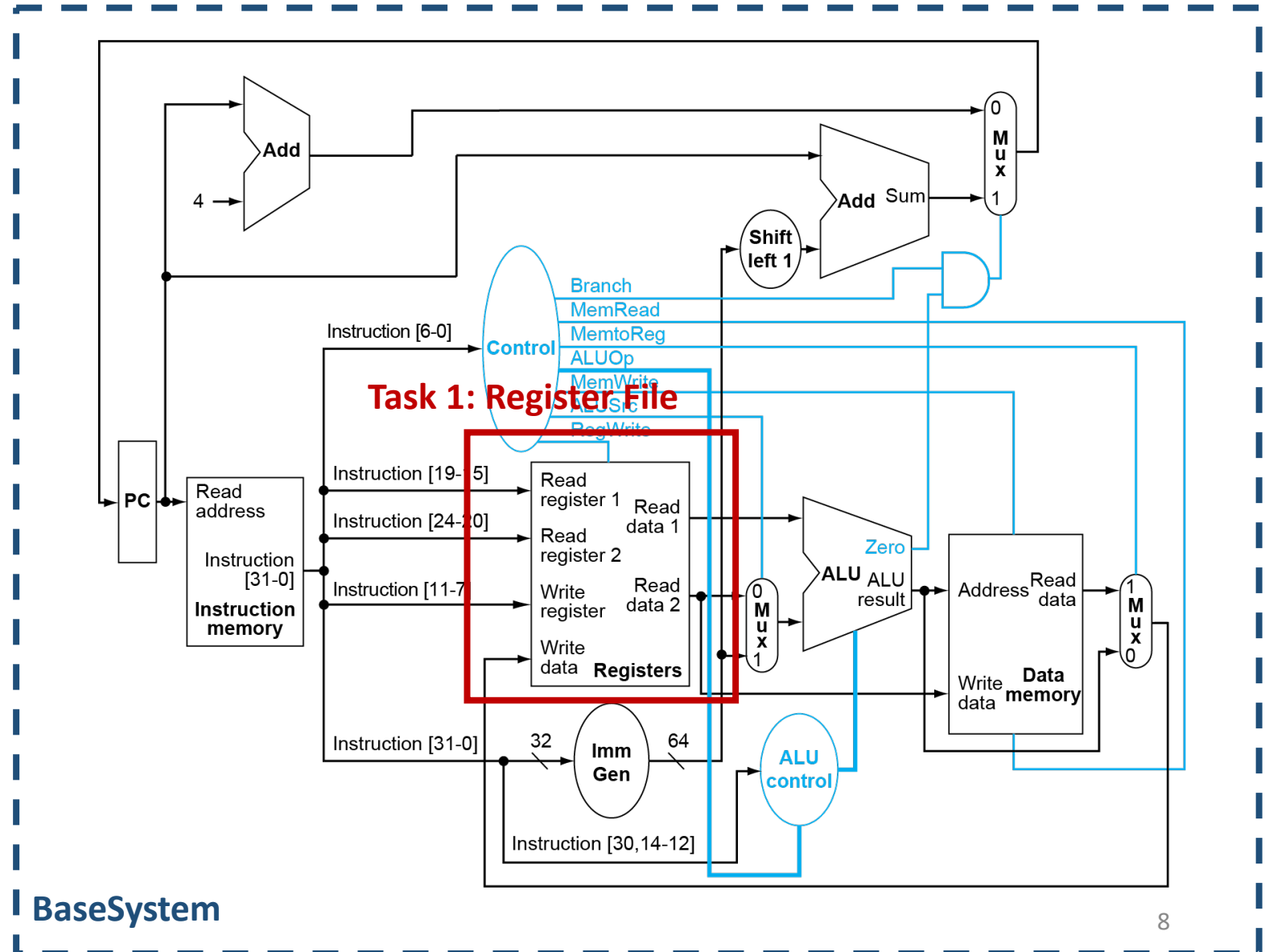
Grand goal: implementing a processor



HW3: Register File and the remainders

- Execute

- Load: ld
- Store: sd
- Branch: beq
- Register ops
 - addi, add
- Jumps
 - jal, jalr



Task 0.1: Incrementing Counter (1/2)

- An example module (also appeared in Lecture)

```
class IncrementingCounter extends Module {  
  val io = IO(new Bundle {  
    val inc = Input(Bool())  
    val output = Output(UInt(32.W))  
  })  
  
  val counter = RegInit(0.U(32.W))  
  counter := Mux(io.inc, counter + 1.U, counter)  
  io.output := counter  
  
}
```

Task 0.1: Incrementing Counter (2/2)

- Test and results (src/test/ExampleTest.scala)

```
poke(c.io.inc, 1)
poke(c.io.reset, 1)
step(1)
println("1: " ++ peek(c.io.output).toString(16))
step(1)
poke(c.io.inc, 0)
println("2: " ++ peek(c.io.output).toString(16))
poke(c.io.inc, 1)
poke(c.io.reset, 0)
step(1)
println("3: " ++ peek(c.io.output).toString(16))
step(1)
println("4: " ++ peek(c.io.output).toString(16))
poke(c.io.inc, 0)
step(1)
println("5: " ++ peek(c.io.output).toString(16))
poke(c.io.inc, 0)
poke(c.io.reset, 1)
step(1)
println("6: " ++ peek(c.io.output).toString(16))
```

```
sbt "testOnly IncrementingCounterTest"
```

```
info] [0.001] SEED 1636269605466
[info] [0.004] 1: 0
[info] [0.006] 2: 0
[info] [0.007] 3: 1
[info] [0.008] 4: 2
[info] [0.009] 5: 2
[info] [0.011] 6: 0
test IncrementingCounter Success:
6 tests passed in 11 cycles taking
0.017777 seconds
```

Task 0.1: Simple Register (1/2)

- An example module (also appeared in Lecture)

```
class RegisterModule extends Module {  
  val io = IO(new Bundle {  
    val in  = Input(UInt(12.W))  
    val output = Output(UInt(12.W))  
  })  
  
  val register = Reg(UInt(12.W))  
  register := io.in + 1.U  
  io.output := register  
}
```

Task 0.2: Simple Register (2/2)

- Test and results (src/test/ExampleTest.scala)

```
poke(c.io.in, 0)
step(1)
println("1: " ++ peek(c.io.output).toString(16))
step(1)
println("2: " ++ peek(c.io.output).toString(16))
poke(c.io.in, 5)
step(1)
println("3: " ++ peek(c.io.output).toString(16))
step(1)
println("4: " ++ peek(c.io.output).toString(16))
poke(c.io.in, 0)
step(1)
println("5: " ++ peek(c.io.output).toString(16))
poke(c.io.in, 0)
step(1)
println("6: " ++ peek(c.io.output).toString(16))
```

```
sbt "testOnly RegisterModuleTest"
```

```
[info] [0.000] SEED 1636270058332
[info] [0.004] 1: 1
[info] [0.005] 2: 1
[info] [0.006] 3: 6
[info] [0.007] 4: 6
[info] [0.008] 5: 1
[info] [0.009] 6: 1
```

Overview

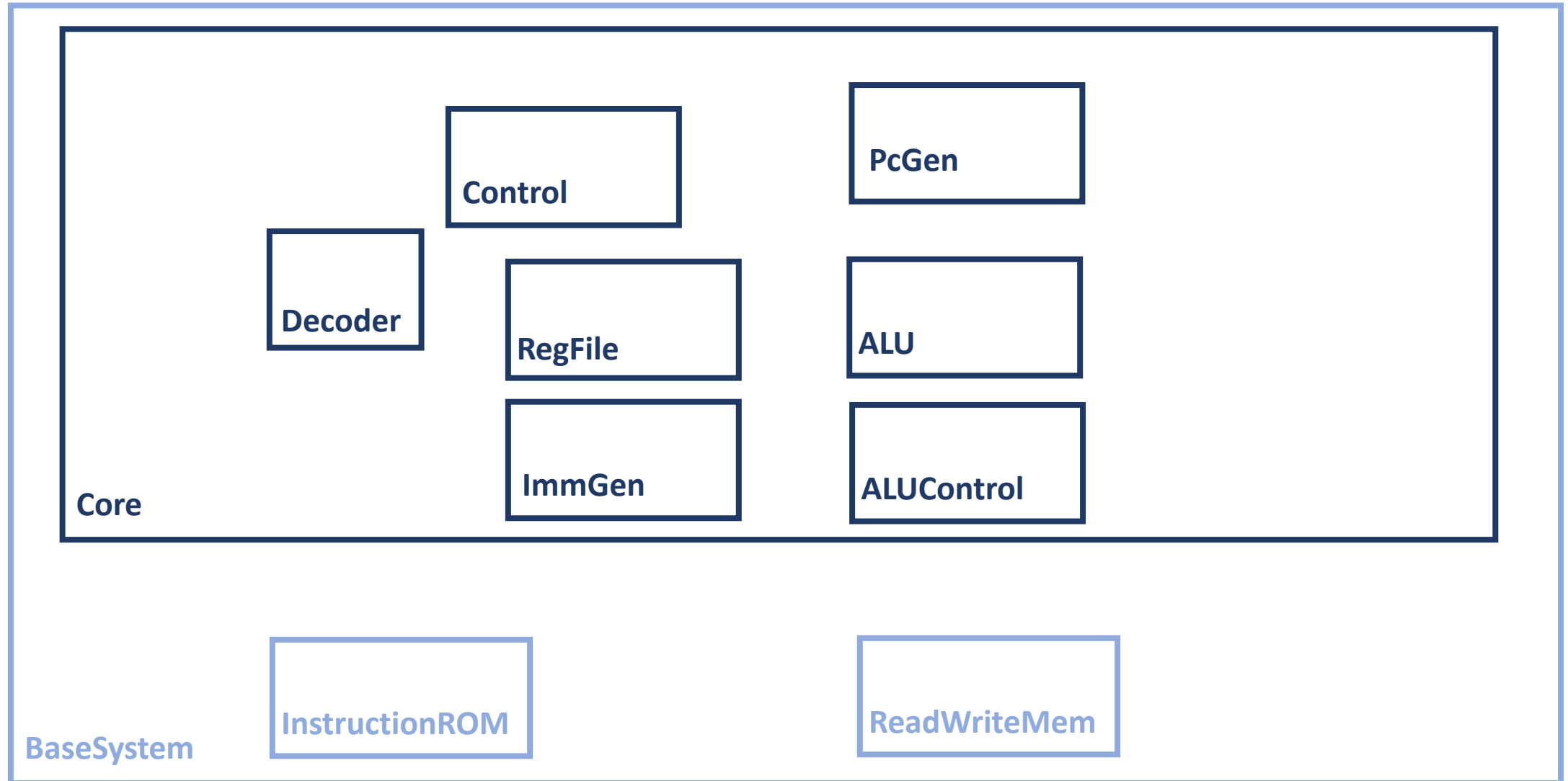
- The test driver compiles test programs and runs on the processor.
- Tests in `src/main/scala/Tests.scala` defines five tests.
- Goal is to implement the register file and a processor running the five tests correctly.
 - Task 1: Register file
 - Task 2: addi
 - Task 3: add
 - Task 4: load and store (ldsd)
 - Task 5: branch (beq)
 - Task 6 (Optional): jump and link (jal, jalr)
 - 20% more score for Hw3, not affecting the other students.

Test Infra

- Unit Tests
- Running programs (use `sbt run`)
 - `src/main/scala/Hw3Driver.scala`: compiles and runs program
 - No need to understand.
 - `sw/test` contains the five tests.
 - `addi.s`, `add.s`, `ldsd.s`, `beq.s`, `jal.s`
 - `src/main/scala/Tests.scala` contains the reference outputs
- Correct output

```
test results:
tested addi (addi.s): passed
tested add (add.s): passed
tested ldsd (ldsd.s): passed
tested beq (beq.s): passed
tested jal (jal.s): passed
```

Modules



Core

- Use imem and dmem interfaces to access memory.
- halted is used for testing.

```
class Core extends Module {  
  val io = IO(new Bundle {  
    //val reset = Input(Bool())  
    val imem_addr = Output(UInt(64.W))  
    val imem_insn = Input(UInt(32.W))  
    val dmem_addr = Output(UInt(64.W))  
    val dmem_write = Output(Bool())  
    val dmem_read = Output(Bool())  
    val dmem_wdata = Output(UInt(64.W))  
    val dmem_rdata = Input(UInt(64.W))  
    val halted = Output(Bool())  
  
  })  
}
```

Task 1: Register File (1/2)

- Implement the register file for our processor.

```
class RegFile extends Module {  
  val io = IO(new Bundle {  
    val rd = Input(UInt(5.W))  
    val rs1 = Input(UInt(5.W))  
    val rs2 = Input(UInt(5.W))  
    val write = Input(Bool())  
    val wdata = Input(UInt(64.W))  
    val rs1_out = Output(UInt(64.W))  
    val rs2_out = Output(UInt(64.W))  
  })  
  
  /* Your code starts here */  
  io.rs1_out := 0.U  
  io.rs2_out := 0.U  
  /* Your code ends here */  
}
```

Task 1: Register File (2/2)

- The test is included in `src/test/scala/Hw3Test.scala`
- Run by

```
sbt "testOnly RegFileTest"
```

- Expected result

```
[info] [0.001] SEED 1636282849175  
test RegFile Success: 64 tests passed  
in 37 cycles taking 0.096864 seconds
```

```
[info] [0.081] RAN 32 CYCLES PASSED
```

- The test will fill out the registers with random values and read through `rs1` and `rs2`.

Task2: addi

- Test program

```
.global _start
_start:
    addi    x1, x1, 0x10
    ld      x1, 0(x1)
    csrrwi  x0, 0, 0
```

This terminates the simulation (already implemented)

- Expected result

```
[0] pc: 0, insn: 1008093, write: 0, wdata: 0, addr: 32, read: 0, rdata: 0, halted: 0
[1] pc: 4, insn: b083, write: 0, wdata: 0, addr: 32, read: 1, rdata: 0, halted: 0
[2] pc: 8, insn: 5073, write: 0, wdata: 0, addr: 0, read: 0, rdata: 0, halted: 0
[3] pc: c, insn: 53, write: 0, wdata: 0, addr: 0, read: 0, rdata: 0, halted: 1
```

Task3: add

- Test program

```
.global _start
_start:
    addi    x1, x0, 0x10
    addi    t2, x0, 0x20
    add     x3, x1, x2
    ld      x1, 0(x3)
    csrrwi  x0, 0, 0
```

- Expected result

```
[0] pc: 0, insn: 1000093, write: 0, wdata: 0, addr: 16, read: 0, rdata: 0, halted: 0
[1] pc: 4, insn: 2000393, write: 0, wdata: 0, addr: 32, read: 0, rdata: 0, halted: 0
[2] pc: 8, insn: 2081b3, write: 0, wdata: 0, addr: 16, read: 0, rdata: 0, halted: 0
[3] pc: c, insn: 1b083, write: 0, wdata: 0, addr: 16, read: 1, rdata: 0, halted: 0
[4] pc: 10, insn: 5073, write: 0, wdata: 0, addr: 0, read: 0, rdata: 0, halted: 0
[5] pc: 14, insn: 53, write: 0, wdata: 0, addr: 0, read: 0, rdata: 0, halted: 1
```

Task 4: load and store

- Test program

```
.global _start
_start:
    addi    x1, x0, 0x204
    sd      x1, 8(x0)
    ld      x2, 8(x0)
    ld      x3, 8(x1)
    csrrwi  x0, 0, 0
```

- Expected result

```
[0] pc: 0, insn: 20400093, write: 0, wdata: 0, addr: 516, read: 0, rdata: 204, halted: 0
[1] pc: 4, insn: 103423, write: 1, wdata: 204, addr: 8, read: 0, rdata: 0, halted: 0
[2] pc: 8, insn: 803103, write: 0, wdata: 0, addr: 8, read: 1, rdata: 204, halted: 0
[3] pc: c, insn: 80b183, write: 0, wdata: 0, addr: 524, read: 1, rdata: 20c, halted: 0
[4] pc: 10, insn: 5073, write: 0, wdata: 0, addr: 0, read: 0, rdata: 0, halted: 0
[5] pc: 14, insn: 53, write: 0, wdata: 0, addr: 0, read: 0, rdata: 0, halted: 1
```

Task 5: branch

- Test program

```
.global _start
_start:
    addi    x1, x0, 0x204
    addi    x2, x0, 0x204
    ld      x1, 0(x1)
    ld      x2, 0(x2)
    add     x3, x1, x2
    beq     x1, x2, target
    csrrwi  x0, 0, 0
    sd      x1, 0(x0)
target:
    sd      x1, 0(x0)
    beq     x1, x0, target2
    csrrwi  x0, 0, 0
target2:
    sd      x0, 0(x0)
    csrrwi  x0, 0, 0
```

Task 5: branch

- Expected result

```
[0] pc: 0, insn: 20400093, write: 0, wdata: 0, addr: 516, read: 0, rdata: 204, halted: 0
[1] pc: 4, insn: 20400113, write: 0, wdata: 0, addr: 516, read: 0, rdata: 204, halted: 0
[2] pc: 8, insn: b083, write: 0, wdata: 0, addr: 516, read: 1, rdata: 204, halted: 0
[3] pc: c, insn: 13103, write: 0, wdata: 0, addr: 516, read: 1, rdata: 204, halted: 0
[4] pc: 10, insn: 2081b3, write: 0, wdata: 204, addr: 1032, read: 0, rdata: 0, halted: 0
[5] pc: 14, insn: 208663, write: 0, wdata: 204, addr: 0, read: 0, rdata: 0, halted: 0
[6] pc: 20, insn: 103023, write: 1, wdata: 204, addr: 0, read: 0, rdata: 204, halted: 0
[7] pc: 24, insn: 8463, write: 0, wdata: 0, addr: 516, read: 0, rdata: 204, halted: 0
[8] pc: 28, insn: 5073, write: 0, wdata: 0, addr: 0, read: 0, rdata: 204, halted: 0
[9] pc: 2c, insn: 3023, write: 1, wdata: 0, addr: 0, read: 0, rdata: 0, halted: 1
```


Task 6: jump and link (Optional)

- Test program

➤ You should implement both `jal` and `jalr`

```
.global _start
_start:
    addi    x10, x0, 0x10
    jal     x1, foo
    sd      x10, 0(x0)
    csrrwi  x0, 0, 0
foo:
    addi    x10, x0, 0x20
    ld      x2, 0(x1)
    jalr    x0, 0(x1)
    csrrwi  x0, 0, 0
```

Task 6: jump and link (Optional)

- Expected result

```
[0] pc: 0, insn: 1000513, write: 0, wdata: 0, addr: 16, read: 0, rdata: 0, halted: 0
[1] pc: 4, insn: c000ef, write: 0, wdata: 0, addr: 8, read: 0, rdata: 0, halted: 0
[2] pc: 10, insn: 2000513, write: 0, wdata: 0, addr: 32, read: 0, rdata: 0, halted: 0
[3] pc: 14, insn: b103, write: 0, wdata: 0, addr: 8, read: 1, rdata: 0, halted: 0
[4] pc: 18, insn: 8067, write: 0, wdata: 0, addr: 28, read: 0, rdata: 0, halted: 0
[5] pc: 8, insn: a03023, write: 1, wdata: 20, addr: 0, read: 0, rdata: 0, halted: 0
[6] pc: c, insn: 5073, write: 0, wdata: 0, addr: 0, read: 0, rdata: 20, halted: 0
[7] pc: 10, insn: 2000513, write: 0, wdata: 0, addr: 32, read: 0, rdata: 0, halted: 1
```