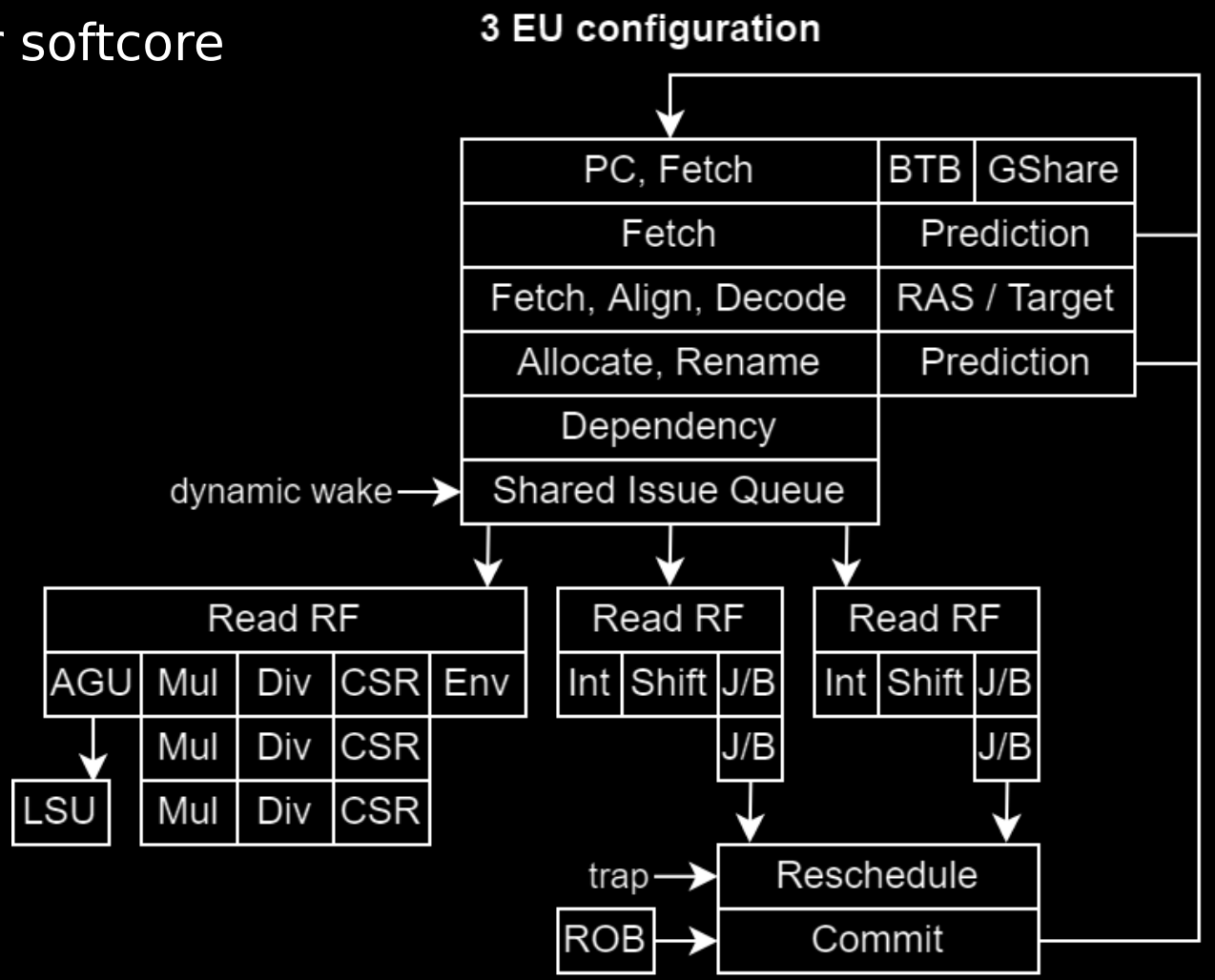
NaxRiscv

An open-source OoO superscalar softcore



Background / whoami

- Dolu1990 on github
- Active on
  - SpinalHDL
  - VexRiscv
  - NaxRiscv
- Software / Hardware background

NaxRiscv in short

- RV32/RV64 IMACSU (Linux ready)
- Out of order / superscalar
- Free / Open source (MIT)
- Many paradigm experimentation with alternative HDL
- Target FPGA with distributed RAM
- ~75 FPS Chocolate-doom (@100Mhz / linux)

https://twitter.com/i/status/1493996880593887235

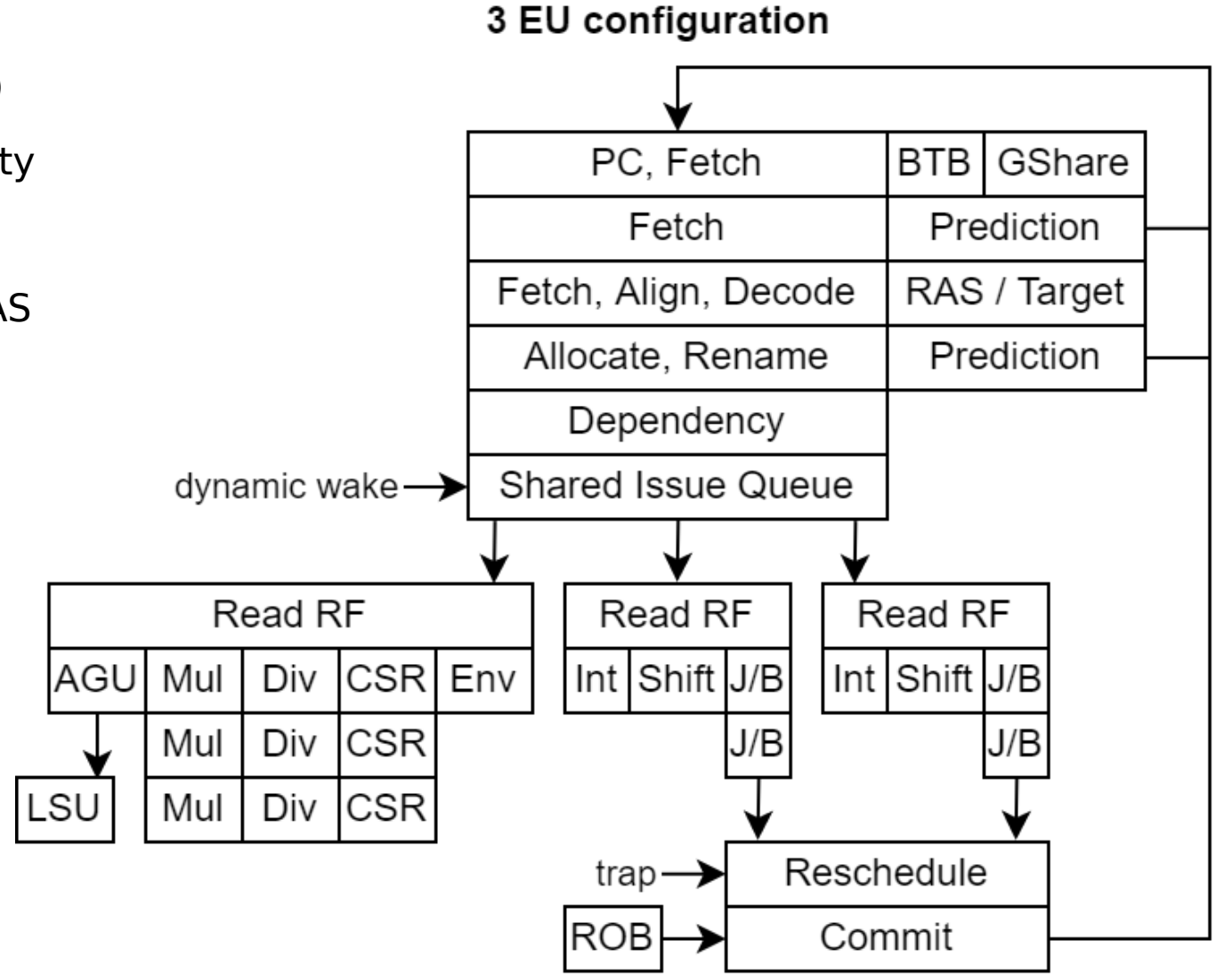
Performances / Area

- RV32IMASU, 2 decode, 3 issue
  - Dhrystone : 2.64 DMIPS/Mhz 1.48 IPC (-O3 -fno-common -fno-inline)
  - Coremark : 4.70 Coremark/Mhz 1.19 IPC
  - Embench-iot : 1.59 baseline 1.35 IPC (baseline = Cortex M4)
- On Artix 7 speed grade 3 :
  - 145 Mhz (360 Mhz on Kintex ultrascale+ grade 3)
  - 14.6 KLUT, 9.8 KFF, 12.5 BRAM, 4 DSP
- RV64IMASU, same FPGA/config
  - 140 Mhz
  - 19.5 KLUT, 12.0 KFF, 12.5 BRAM, 16 DSP

NaxRiscv => https://spinalhdl.github.io/NaxRiscv-Rtd/main/NaxRiscv/performance/index.html
RSD => https://www.rsg.ci.i.u-tokyo.ac.jp/members/shioya/pdfs/Mashimo-FPT'19.pdf

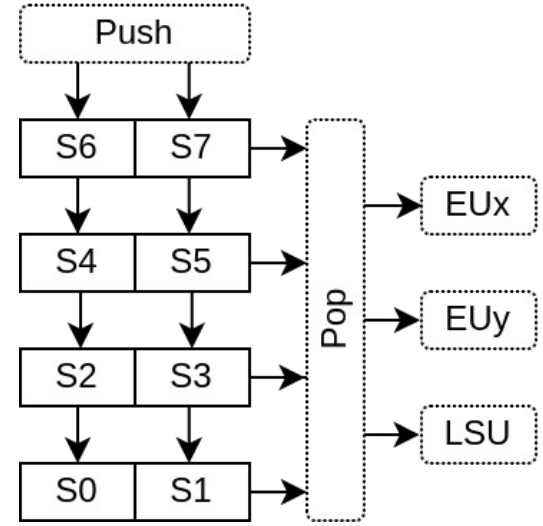
Architecture (as configured in the last slide)

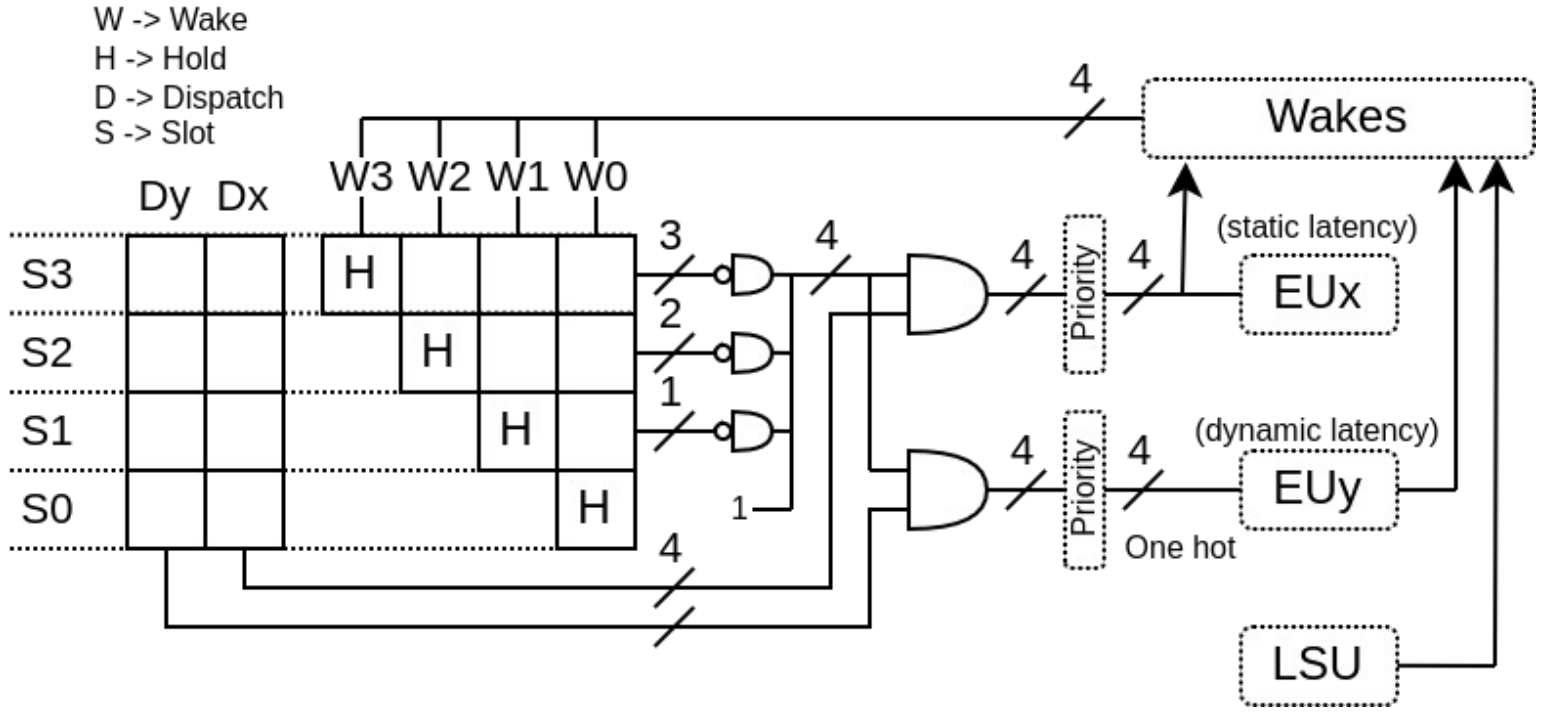
- 2 decode, 3 issue (2 ALU, 1 shared EU)
- 10 cycles miss predicted branch penalty
- 64 physical register, 64 entry ROB
- 4KB GShare, 4KB BTB (both 1 way), RAS
- Non-blocking D$
- I$ D$ each have 16 KB (4 ways)
- I$ D$ each have 192 TLB (2+4 ways)



Dispatch / Issue Queue

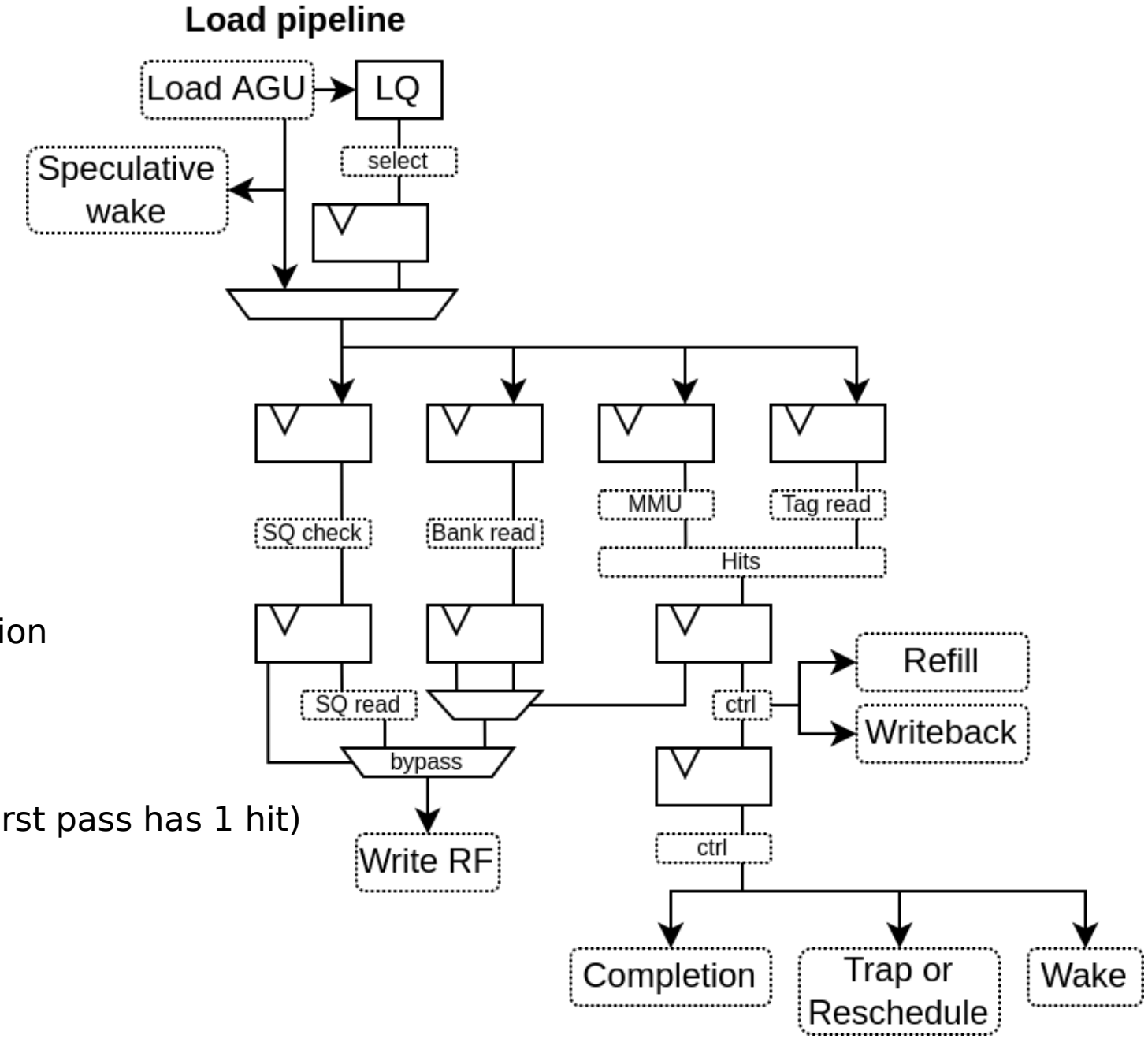
- Shared between execution units
- 32 entries, grouped by “wave”, (ex: 2x16)
- Older first, no compression
- Matrix design, removal on completion





LSU (load)

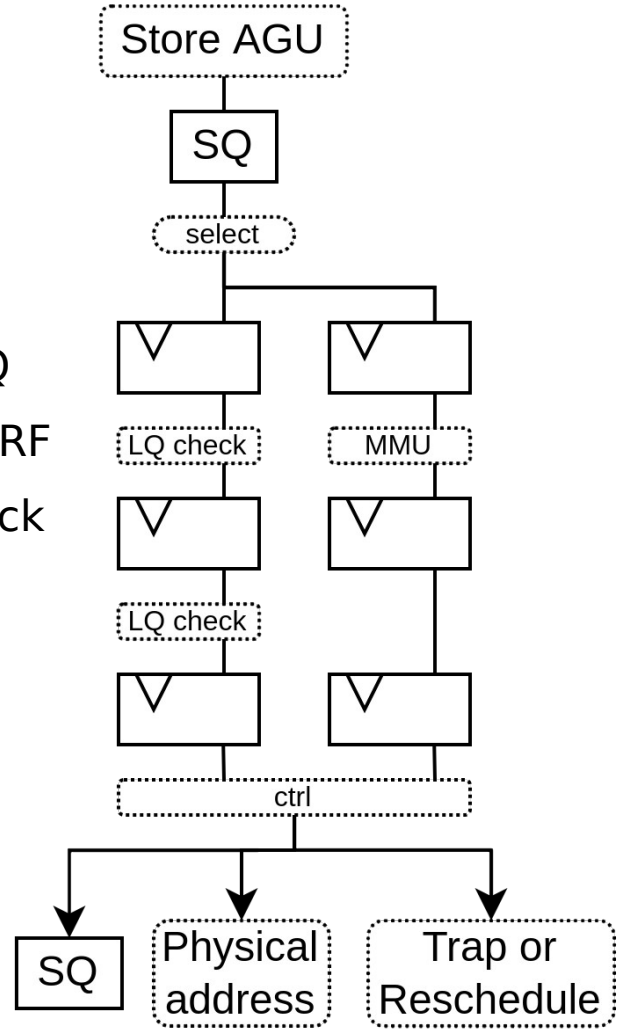
- Speculative wake (hit prediction)
- load to use latency :
  - 3 cycle (hit predicted OK)
  - 6 cycle (miss predicted)
  - Full flush (hit predicted KO)
- AGU to D$ bypass
- RF written 1 cycle before completion
- SQ address hazard check
  - 4KB virtual (first pass)
  - Full physical (second pass if first pass has 1 hit)
- 16 entries



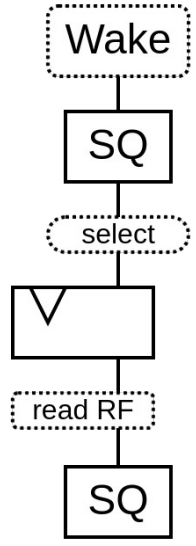
LSU (store)

- 4 pipelines
- Dependencies
  - Address through IQ
  - Data directly read RF
- LQ address hazard check
  - 4KB virtual
- 16 entries

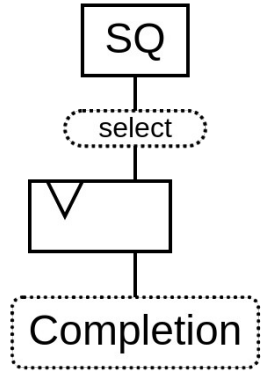
Store pipeline (address)



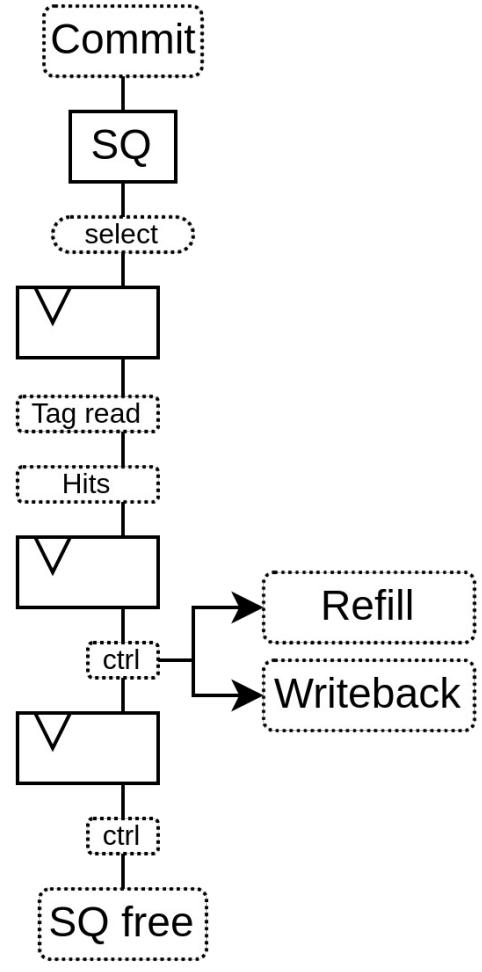
Store pipeline (data)



Store pipeline (completion)

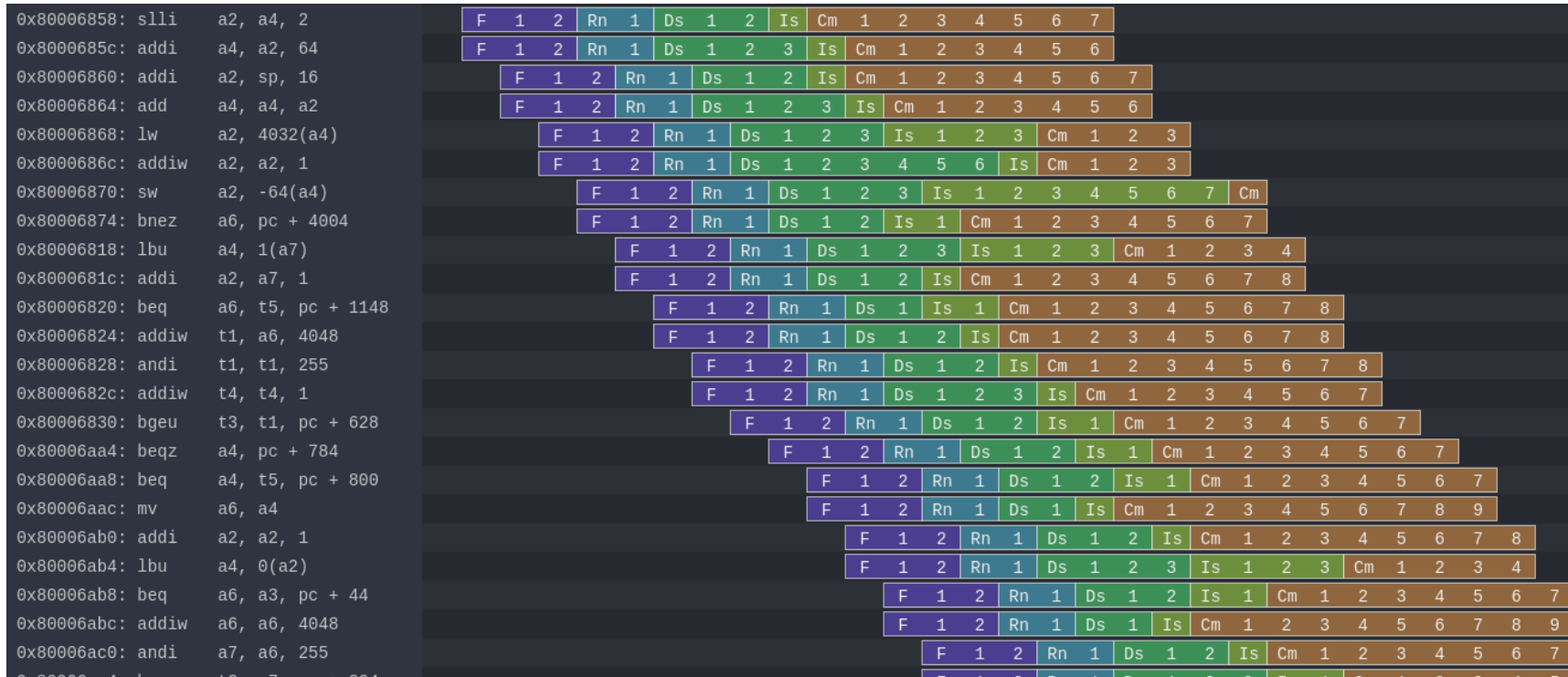


Writeback pipeline



Simulation

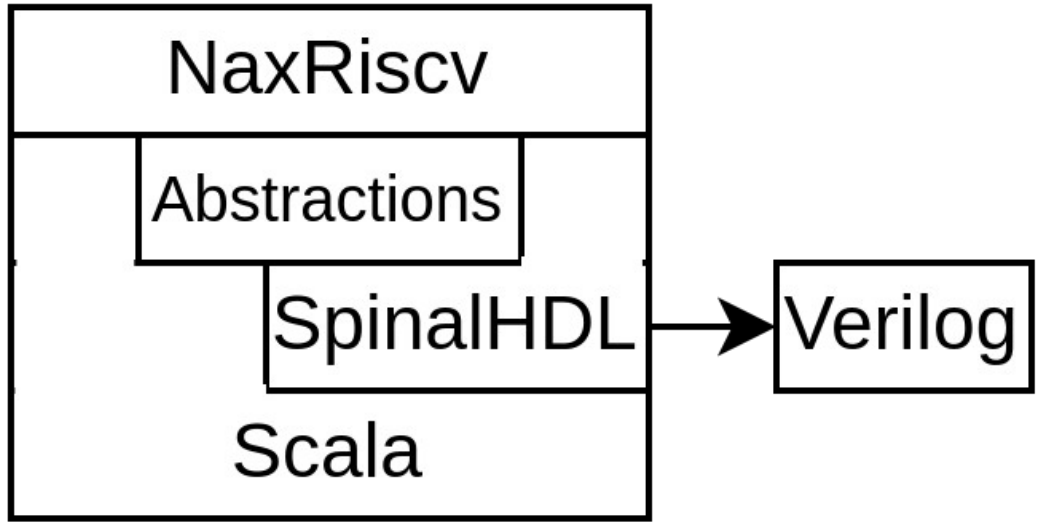
- Based on Verilator (~150-200 Khz on Ryzen 3900)
- Lock-step checks against Spike (life saver)
- Can emit Gem5-O3 / Konata traces



Exploring hardware description paradigms

- NaxRiscv <> Scala + SpinalHDL <> Verilog
  - SpinalHDL ~= Chisel / Migen / Amaranth HDL / ...

- 4 key concepts in NaxRiscv
  - Composable toplevel
  - Multithreaded elaboration
  - Pipelining framework
  - Software engineering



```
val plugins = ArrayBuffer[Plugin]()

plugins ++= List(
    new PcPlugin(),
    new FetchCachePlugin(16 KB),
    new DecoderPlugin(),
    new DispatchPlugin(slotCount = 32),
    new CommitPlugin()
    ...
)

plugins ++= List(
    new ExecutionUnitBase("ALU0"),
    new IntAluPlugin("ALU0"),
    new ShiftPlugin("ALU0" ),
    new BranchPlugin("ALU0")
    new IntFormatPlugin("ALU0"),
    new SrcPlugin("ALU0"),
)

new NaxRiscv(plugins)
```

```
val plugins = ArrayBuffer[Plugin]()

plugins ++= List(
    new PcPlugin(),
    new FetchCachePlugin(16 KB),
    new DecoderPlugin(),
    new DispatchPlugin(slotCount = 32),
    new CommitPlugin()
    ...
)

plugins ++= List(
    new ExecutionUnitBase("ALU0"),
    new IntAluPlugin("ALU0"),
    new ShiftPlugin("ALU0" ),
    new BranchPlugin("ALU0")
    new IntFormatPlugin("ALU0"),
    new SrcPlugin("ALU0"),
)

new NaxRiscv(plugins)
```

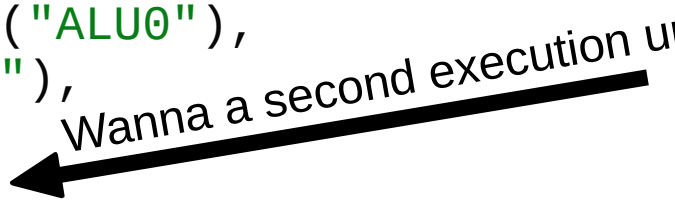
Create NaxRiscv

```
val plugins = ArrayBuffer[Plugin]()
```

Elaboration time list of Plugin

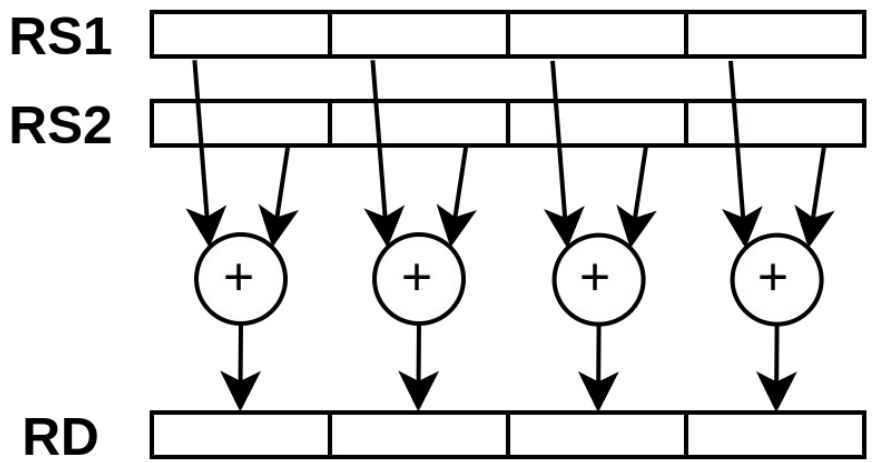
```
plugins ++= List(
    new PcPlugin(),
    new FetchCachePlugin(16 KB),
    new DecoderPlugin(),
    new DispatchPlugin(slotCount = 32),
    new CommitPlugin()
    ...
)

plugins ++= List(
    new ExecutionUnitBase("ALU0"),
    new IntAluPlugin("ALU0"),
    new ShiftPlugin("ALU0" ),
    new BranchPlugin("ALU0")
    new IntFormatPlugin("ALU0"),
    new SrcPlugin("ALU0"),
)

new NaxRiscv(plugins)
```

```scala
val plugins = ArrayBuffer[Plugin]()

plugins ++= List(
    new PcPlugin(),
    new FetchCachePlugin(16 KB),
    new DecoderPlugin(),
    new DispatchPlugin(slotCount = 32),
    new CommitPlugin()
    ...
)

plugins ++= List(
    new ExecutionUnitBase("ALU0"),
    new IntAluPlugin("ALU0"),
    new ShiftPlugin("ALU0" ),
    new BranchPlugin("ALU0")
    new IntFormatPlugin("ALU0"),
    new SrcPlugin("ALU0"),
)

plugins ++= List(
    new ExecutionUnitBase("ALU1"),
    new IntAluPlugin("ALU1"),
    new ShiftPlugin("ALU1" ),
    new BranchPlugin("ALU1")
    new IntFormatPlugin("ALU1"),
    new SrcPlugin("ALU1"),
)

new NaxRiscv(plugins)
```

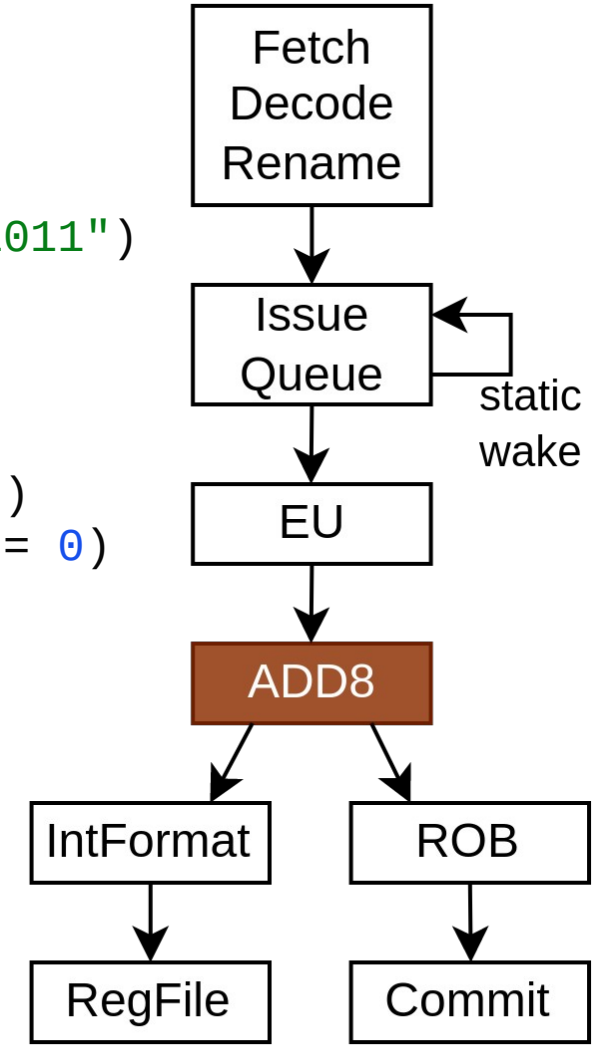


Plugin / SIMD ADD8

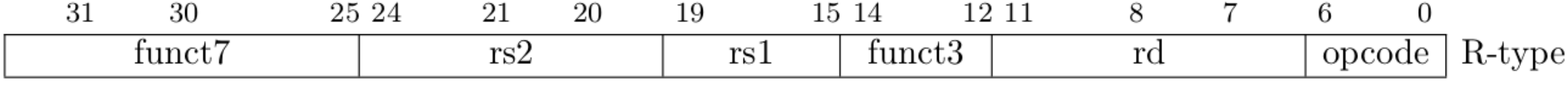
```
class SimdAddPlugin(euId : String) extends Plugin {
    val setup = create early new Area{
        // ...
    }

    val logic = create late new Area{
        // ...
    }
}
```



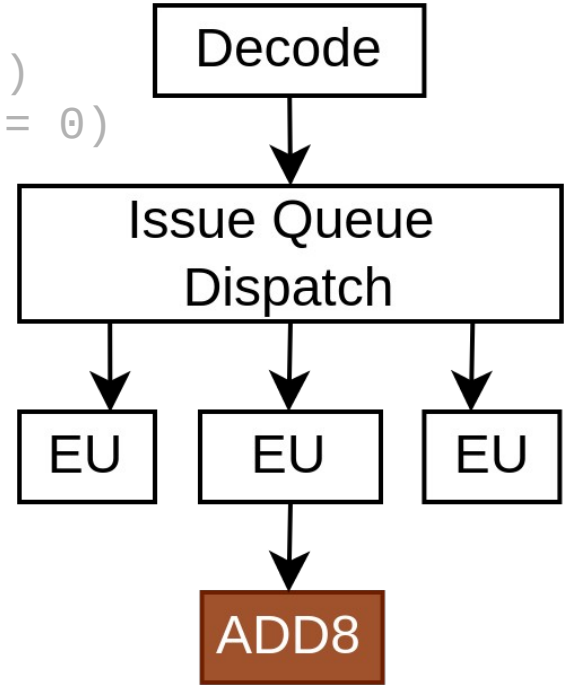
```scala
val setup = create early new Area{
    val eu = findService[ExecutionUnitBase](_.euId == euId)
    eu.retain()

    val ADD8 = IntRegFile.TypeR(M"0000000----------000-----0001011")
    eu.addMicroOp(ADD8)
    eu.setCompletion(ADD8, stageId = 0)
    eu.setStaticWake(ADD8, stageId = 0)

    val intFormat = findService[IntFormatPlugin](_.euId == euId)
    val writeback = intFormat.access(stageId = 0, writeLatency = 0)

    val SEL = Stageable(Bool())
    eu.setDecodingDefault(SEL, False)
    eu.addDecoding(ADD8, SEL, True)
}
```



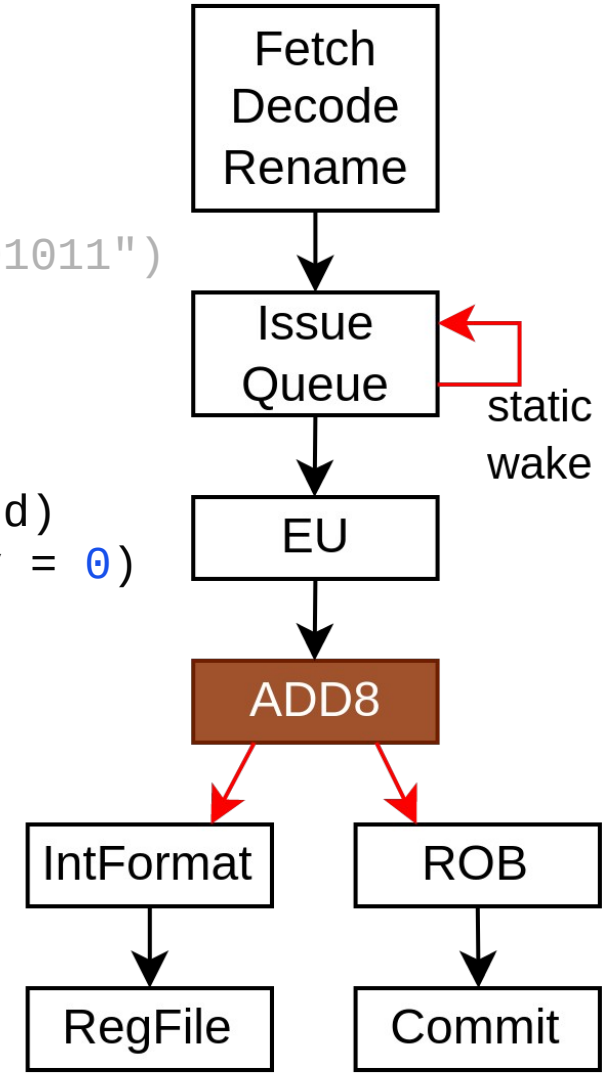
```
val setup = create early new Area{
    val eu = findService[ExecutionUnitBase](_.euId == euId)
    eu.retain()

    val ADD8 = IntRegFile.TypeR(M"0000000----------000-----0001011")
    eu.addMicroOp(ADD8)
    eu.setCompletion(ADD8, stageId = 0)
    eu.setStaticWake(ADD8, stageId = 0)

    val intFormat = findService[IntFormatPlugin](_.euId == euId)
    val writeback = intFormat.access(stageId = 0, writeLatency = 0)

    val SEL = Stageable(Bool())
    eu.setDecodingDefault(SEL, False)
    eu.addDecoding(ADD8, SEL, True)
}
```



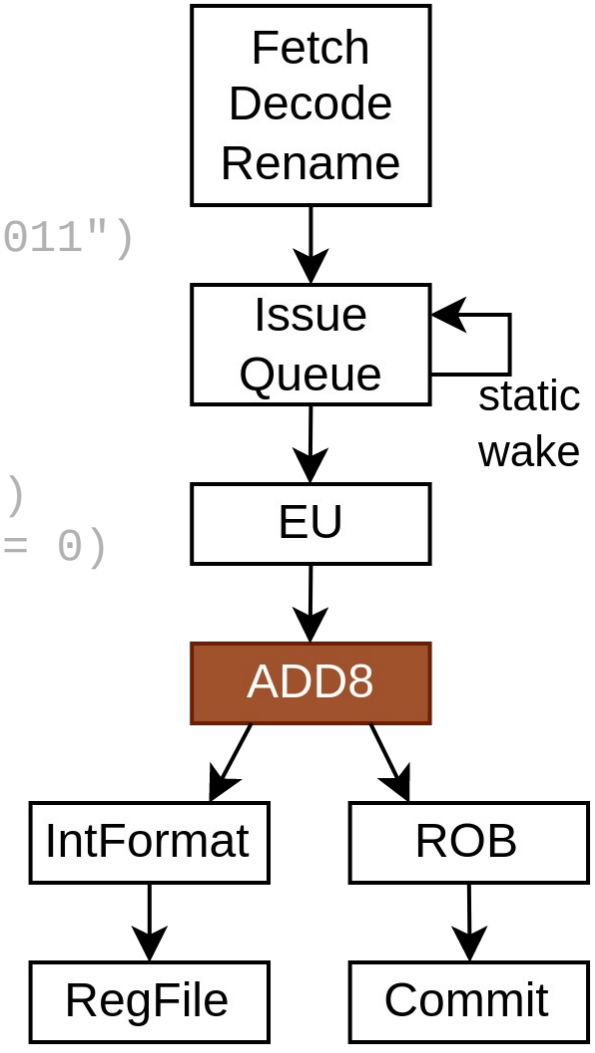
```
val setup = create early new Area{
    val eu = findService[ExecutionUnitBase](_.euId == euId)
    eu.retain()

    val ADD8 = IntRegFile.TypeR(M"0000000----------000-----0001011")
    eu.addMicroOp(ADD8)
    eu.setCompletion(ADD8, stageId = 0)
    eu.setStaticWake(ADD8, stageId = 0)

    val intFormat = findService[IntFormatPlugin](_.euId == euId)
    val writeback = intFormat.access(stageId = 0, writeLatency = 0)

    val SEL = Stageable(Bool())
    eu.setDecodingDefault(SEL, False)
    eu.addDecoding(ADD8, SEL, True)
}
```



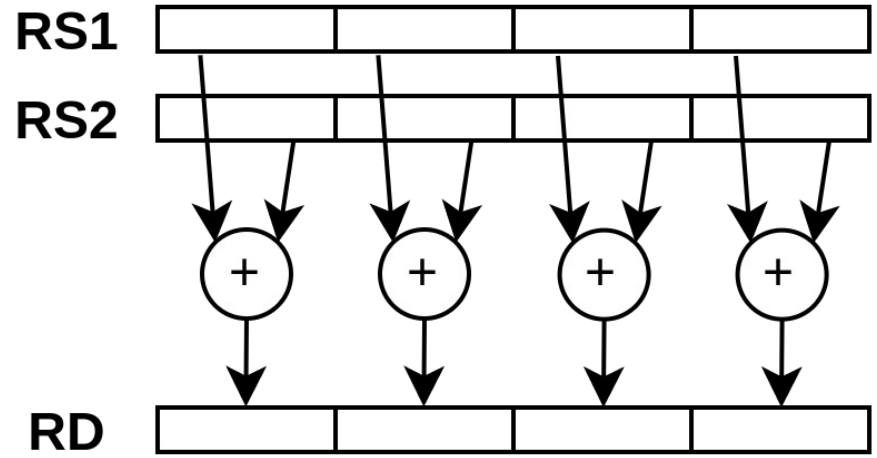
```
val setup = create early new Area{
    val eu = findService[ExecutionUnitBase](_.euId == euId)
    eu.retain()

    val ADD8 = IntRegFile.TypeR(M"0000000----------000-----0001011")
    eu.addMicroOp(ADD8)
    eu.setCompletion(ADD8, stageId = 0)
    eu.setStaticWake(ADD8, stageId = 0)

    val intFormat = findService[IntFormatPlugin](_.euId == euId)
    val writeback = intFormat.access(stageId = 0, writeLatency = 0)

    val SEL = Stageable(Bool())
    eu.setDecodingDefault(SEL, False)
    eu.addDecoding(ADD8, SEL, True)
}
```



```
val setup = create early new Area{
    val eu = findService[ExecutionUnitBase](_.euId == euId)
    eu.retain()

    val ADD8 = IntRegFile.TypeR(M"0000000----------000-----0001011")
    eu.addMicroOp(ADD8)
    eu.setCompletion(ADD8, stageId = 0)
    eu.setStaticWake(ADD8, stageId = 0)

    val intFormat = findService[IntFormatPlugin](_.euId == euId)
    val writeback = intFormat.access(stageId = 0, writeLatency = 0)

    val SEL = Stageable(Bool())
    eu.setDecodingDefault(SEL, False)
    eu.addDecoding(ADD8, SEL, True)
}
```

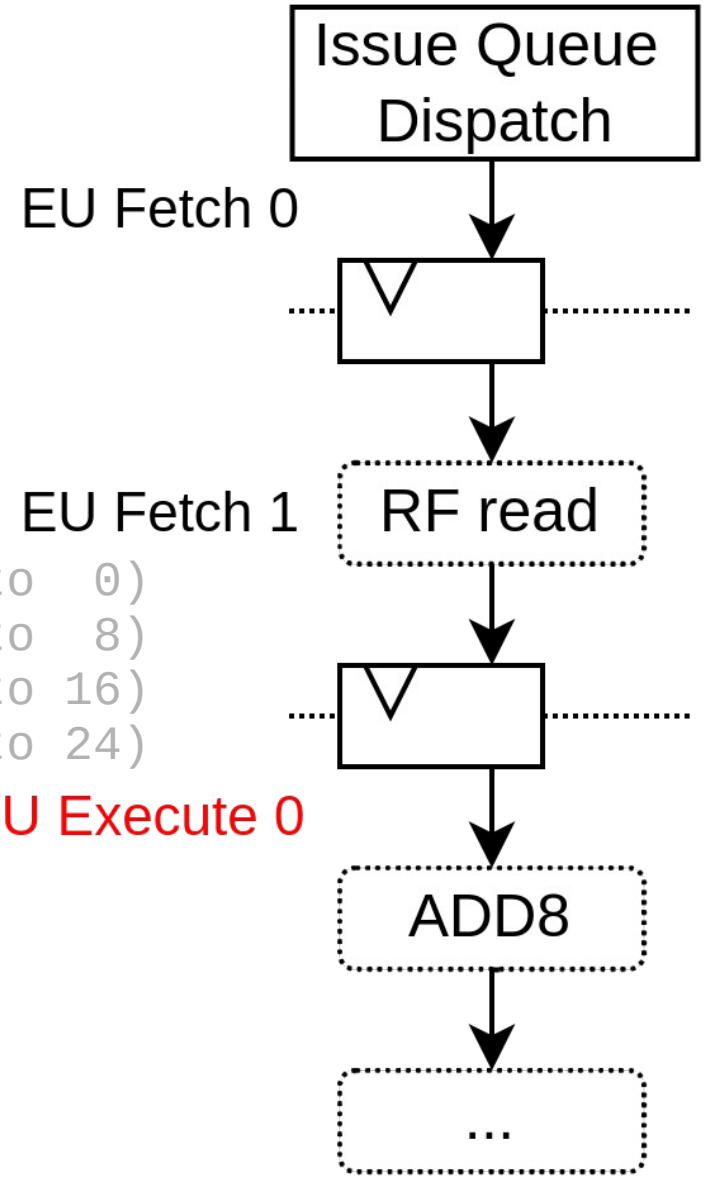


```scala
val logic = create late new Area{
    val eu        = setup.eu
    val writeback = setup.writeback
    val stage     = eu.getExecute(stageId = 0)

    val rs1 = stage(eu(IntRegFile, RS1)).asUInt
    val rs2 = stage(eu(IntRegFile, RS2)).asUInt

    val rd = UInt(32 bits)
    rd( 7 downto  0) := rs1( 7 downto  0) + rs2( 7 downto  0)
    rd(16 downto  8) := rs1(16 downto  8) + rs2(16 downto  8)
    rd(23 downto 16) := rs1(23 downto 16) + rs2(23 downto 16)
    rd(31 downto 24) := rs1(31 downto 24) + rs2(31 downto 24)

    writeback.valid   := stage(setup.SEL)
    writeback.payload := rd.asBits

    eu.release()
}
```



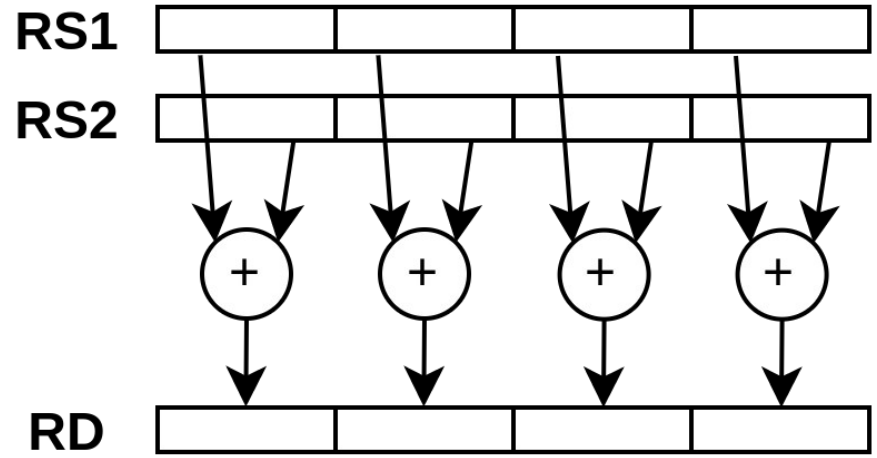
https://spinalhdl.github.io/NaxRiscv-Rtd/main/NaxRiscv/execution_units/index.html#custom-instruction

```
val logic = create late new Area{
    val eu        = setup.eu
    val writeback = setup.writeback
    val stage     = eu.getExecute(stageId = 0)

    val rs1 = stage(eu(IntRegFile, RS1)).asUInt
    val rs2 = stage(eu(IntRegFile, RS2)).asUInt

    val rd = UInt(32 bits)
    rd( 7 downto  0) := rs1( 7 downto  0) + rs2( 7 downto  0)
    rd(16 downto  8) := rs1(16 downto  8) + rs2(16 downto  8)
    rd(23 downto 16) := rs1(23 downto 16) + rs2(23 downto 16)
    rd(31 downto 24) := rs1(31 downto 24) + rs2(31 downto 24)

    writeback.valid   := stage(setup.SEL)
    writeback.payload := rd.asBits

    eu.release()
}
```



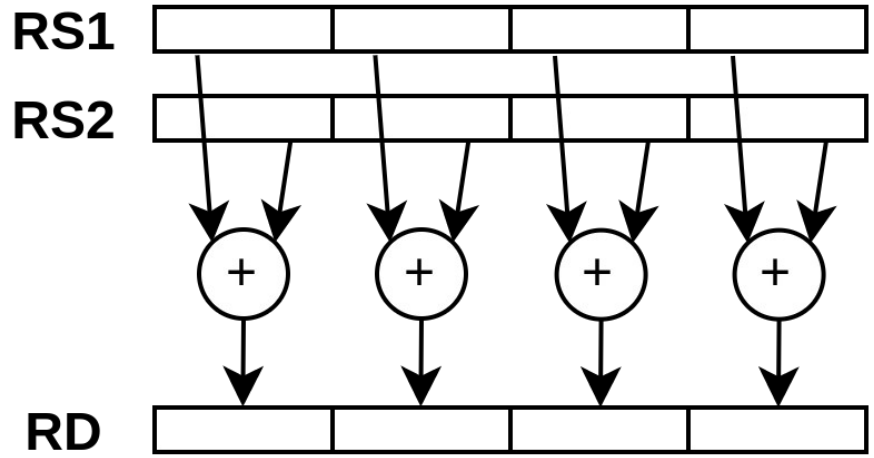
https://spinalhdl.github.io/NaxRiscv-Rtd/main/NaxRiscv/execution_units/index.html#custom-instruction

```
val logic = create late new Area{
    val eu        = setup.eu
    val writeback = setup.writeback
    val stage     = eu.getExecute(stageId = 0)

    val rs1 = stage(eu(IntRegFile, RS1)).asUInt
    val rs2 = stage(eu(IntRegFile, RS2)).asUInt

    val rd = UInt(32 bits)
    rd( 7 downto  0) := rs1( 7 downto  0) + rs2( 7 downto  0)
    rd(16 downto  8) := rs1(16 downto  8) + rs2(16 downto  8)
    rd(23 downto 16) := rs1(23 downto 16) + rs2(23 downto 16)
    rd(31 downto 24) := rs1(31 downto 24) + rs2(31 downto 24)

    writeback.valid   := stage(setup.SEL)
    writeback.payload := rd.asBits

    eu.release()
}
```



https://spinalhdl.github.io/NaxRiscv-Rtd/main/NaxRiscv/execution_units/index.html#custom-instruction

```
val logic = create late new Area{
    val eu        = setup.eu
    val writeback = setup.writeback
    val stage     = eu.getExecute(stageId = 0)

    val rs1 = stage(eu(IntRegFile, RS1)).asUInt
    val rs2 = stage(eu(IntRegFile, RS2)).asUInt

    val rd = UInt(32 bits)
    rd( 7 downto  0) := rs1( 7 downto  0) + rs2( 7 downto  0)
    rd(16 downto  8) := rs1(16 downto  8) + rs2(16 downto  8)
    rd(23 downto 16) := rs1(23 downto 16) + rs2(23 downto 16)
    rd(31 downto 24) := rs1(31 downto 24) + rs2(31 downto 24)

    writeback.valid   := stage(setup.SEL)
    writeback.payload := rd.asBits

    eu.release()
}
```



https://spinalhdl.github.io/NaxRiscv-Rtd/main/NaxRiscv/execution_units/index.html#custom-instruction

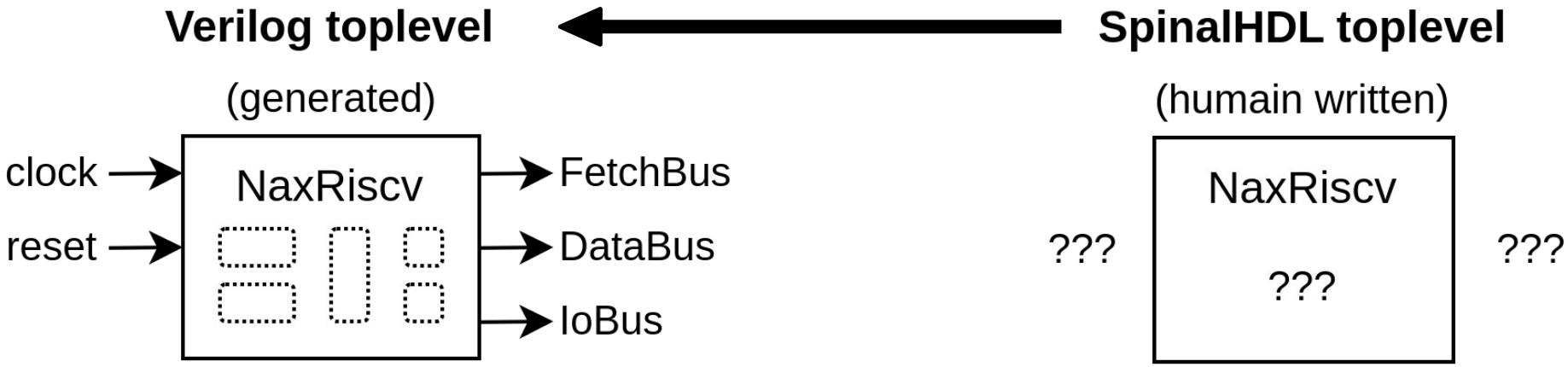
```scala
val plugins = ArrayBuffer[Plugin]()

...

plugins ++= List(
    new ExecutionUnitBase("ALU0"),
    new IntAluPlugin("ALU0"),
    new ShiftPlugin("ALU0" ),
    new BranchPlugin("ALU0")
    new IntFormatPlugin("ALU0"),
    new SrcPlugin("ALU0"),
)

plugins += new SimdAddPlugin("ALU0")
plugins += new SimdAddPlugin("ALU1")

new NaxRiscv(plugins)
```

```scala
plugins ++= List(
    new ExecutionUnitBase("ALU1"),
    new IntAluPlugin("ALU1"),
    new ShiftPlugin("ALU1" ),
    new BranchPlugin("ALU1")
    new IntFormatPlugin("ALU1"),
    new SrcPlugin("ALU1"),
)
```

Status / Future

- RV32/RV64 IMACSU is stable
  - Linux / FreeRTOS / riscv-tests / riscv-arch-test OK
- JTAG / OpenOCD / GDB supported (RISCV External Debug Support 0.13.2)
- Planning :
  - FPU → Debian support
  - Multicore / Memory coherency
- GIT : https://github.com/SpinalHDL/NaxRiscv
- Integrated in Litex for FPGA deployments
  - python3 -m litex_boards.targets.digilent_nexys_video --cpu-type=naxriscv --with-video-framebuffer --with-spi-sdcard --with-ethernet --build –load
- Thanks Nlnet for the funding

Extra slides not included in the talk




```
class NaxRiscv(plugins : Seq[Plugin]) extends Component{
   val database = new DataBase
   val framework = NaxScope(database) on new Framework(plugins)
}
```

```scala
class SimdAddPlugin(val euId : String) extends ExecutionUnitElementSimple(euId, staticLatency = true) {
  override def euWritebackAt = 0
  override val setup = create early new Setup{
    add(SimdAddPlugin.ADD4)
  }

  override val logic = create late new Logic{
    val process = new ExecuteArea(stageId = 0) {
      val rs1 = stage(eu(IntRegFile, RS1)).asUInt
      val rs2 = stage(eu(IntRegFile, RS2)).asUInt

      val rd = UInt(32 bits)
      rd( 7 downto  0) := rs1( 7 downto  0) + rs2( 7 downto  0)
      rd(16 downto  8) := rs1(16 downto  8) + rs2(16 downto  8)
      rd(23 downto 16) := rs1(23 downto 16) + rs2(23 downto 16)
      rd(31 downto 24) := rs1(31 downto 24) + rs2(31 downto 24)

      wb.payload := rd.asBits
    }
  }
}
```

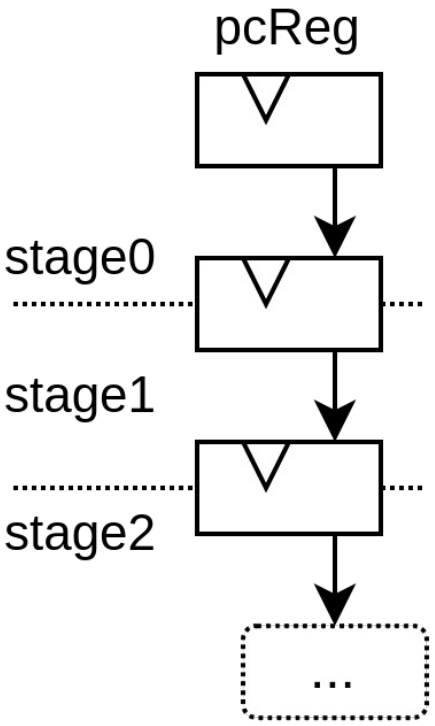
```
class CommitPlugin extends Plugin {
    val setup = create early new Area {
        val jump = getService[JumpService].newJumpPort(10)
    }
    val logic = create late new Area {
        setup.jump.valid := ???
        setup.jump.payload := ???
    }
}
```

```
trait JumpService extends Service{
    def newJumpPort(priority : Int) : Flow[UInt]
}
```

```
class PcPlugin extends Plugin with JumpService{
    override def newJumpPort(priority: Int) = {...}
    val logic = create late new Area {
        val pc = Reg(UInt(32 bits))
        ...
    }
}
```

```
class PcPlugin extends Plugin {
    val logic = ...{
        val pcReg = Reg(UInt(32 bits))
        getService[FetchPlugin].pipeline.stages(0)(PC) := pcReg
    }
}

class FetchCachePlugin extends Plugin{
    val logic = ...{
        getService[FetchPlugin].pipeline.stages(2)(PC) === wayTag
    }
}
```

```
object PC extends Stageable(UInt(32 bits))
```



```
class FetchCachePlugin extends Plugin{
    val logic = ...{
        getService[FetchPlugin].pipeline.stages(2)(PC) === wayTag
    }
}
```

```
class FetchPlugin() extends Plugin {
    val pipeline = create early new Pipeline {
        val stages = Array.fill(3)(newStage())
        ...
    }

    val lock = Lock()
    val builder = create late new Area{
        lock.await()
        pipeline.build()
    }
}
```