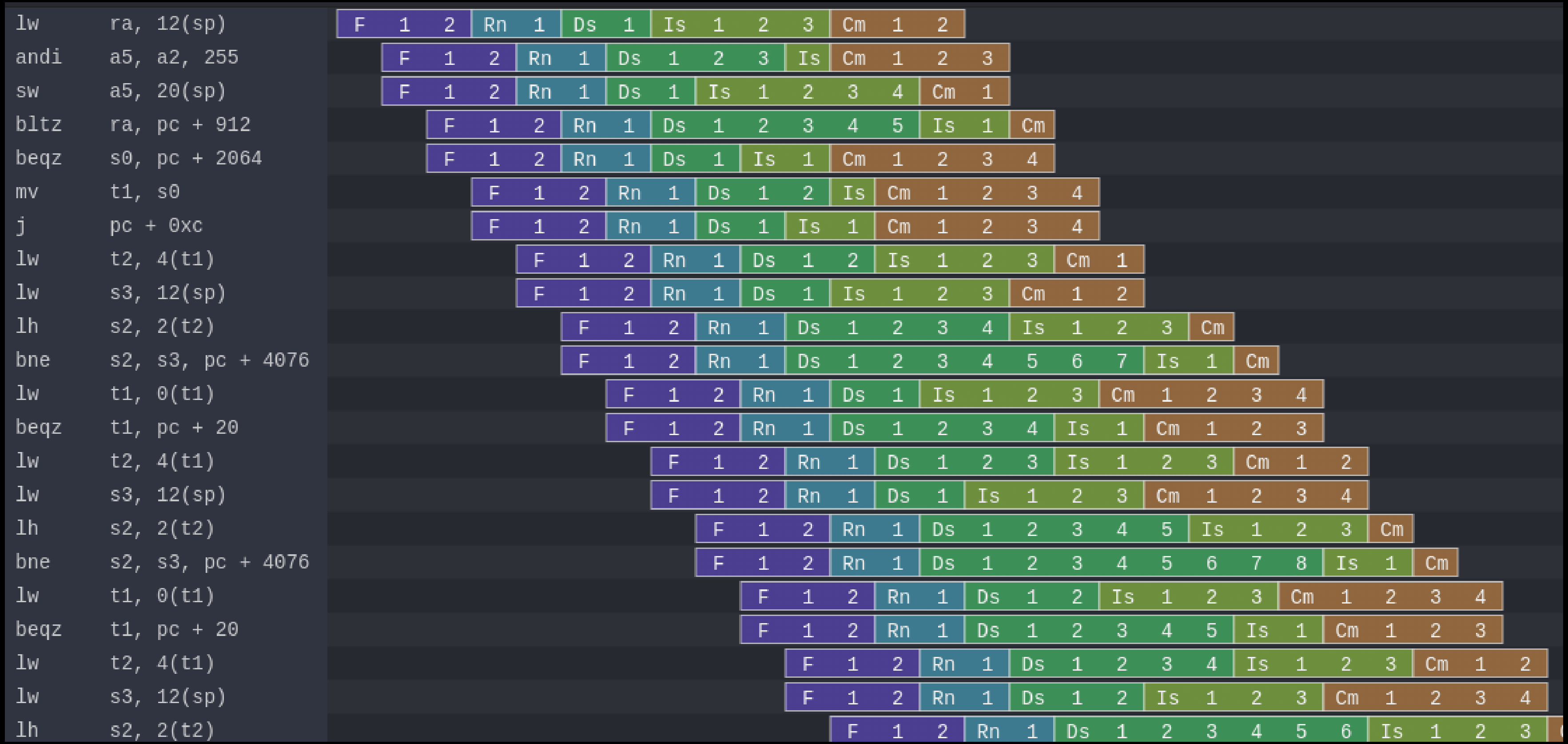
# NaxRiscv : A Pipeline / Plugins / SpinalHDL / Scala mix



```
lw    ra, 12(sp)          F 1 2 Rn 1 Ds 1 Is 1 2 3 Cm 1 2
andi  a5, a2, 255           F 1 2 Rn 1 Ds 1 2 3 Is Cm 1 2 3
sw    a5, 20(sp)            F 1 2 Rn 1 Ds 1 Is 1 2 3 4 Cm 1
bltz  ra, pc + 912            F 1 2 Rn 1 Ds 1 2 3 4 5 Is 1 Cm
beqz  s0, pc + 2064           F 1 2 Rn 1 Ds 1 Is 1 Cm 1 2 3 4
mv    t1, s0                    F 1 2 Rn 1 Ds 1 2 Is Cm 1 2 3 4
j     pc + 0xc                  F 1 2 Rn 1 Ds 1 Is 1 Cm 1 2 3 4
lw    t2, 4(t1)                   F 1 2 Rn 1 Ds 1 2 Is 1 2 3 Cm 1
lw    s3, 12(sp)                  F 1 2 Rn 1 Ds 1 Is 1 2 3 Cm 1 2
lh    s2, 2(t2)                     F 1 2 Rn 1 Ds 1 2 3 4 Is 1 2 3 Cm
bne   s2, s3, pc + 4076             F 1 2 Rn 1 Ds 1 2 3 4 5 6 7 Is 1 Cm
lw    t1, 0(t1)                       F 1 2 Rn 1 Ds 1 Is 1 2 3 Cm 1 2 3 4
beqz  t1, pc + 20                     F 1 2 Rn 1 Ds 1 2 3 4 Is 1 Cm 1 2 3
lw    t2, 4(t1)                         F 1 2 Rn 1 Ds 1 2 3 Is 1 2 3 Cm 1 2
lw    s3, 12(sp)                        F 1 2 Rn 1 Ds 1 Is 1 2 3 Cm 1 2 3 4
lh    s2, 2(t2)                           F 1 2 Rn 1 Ds 1 2 3 4 5 Is 1 2 3 Cm
bne   s2, s3, pc + 4076                   F 1 2 Rn 1 Ds 1 2 3 4 5 6 7 8 Is 1 Cm
lw    t1, 0(t1)                             F 1 2 Rn 1 Ds 1 2 Is 1 2 3 Cm 1 2 3 4
beqz  t1, pc + 20                           F 1 2 Rn 1 Ds 1 2 3 4 5 Is 1 Cm 1 2 3
lw    t2, 4(t1)                               F 1 2 Rn 1 Ds 1 2 3 4 Is 1 2 3 Cm 1 2
lw    s3, 12(sp)                              F 1 2 Rn 1 Ds 1 2 Is 1 2 3 Cm 1 2 3 4
lh    s2, 2(t2)                                 F 1 2 Rn 1 Ds 1 2 3 4 5 6 Is 1 2 3
```

## NaxRiscv in short

- RV32/RV64 IMAFDCSU (Linux/Debian 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

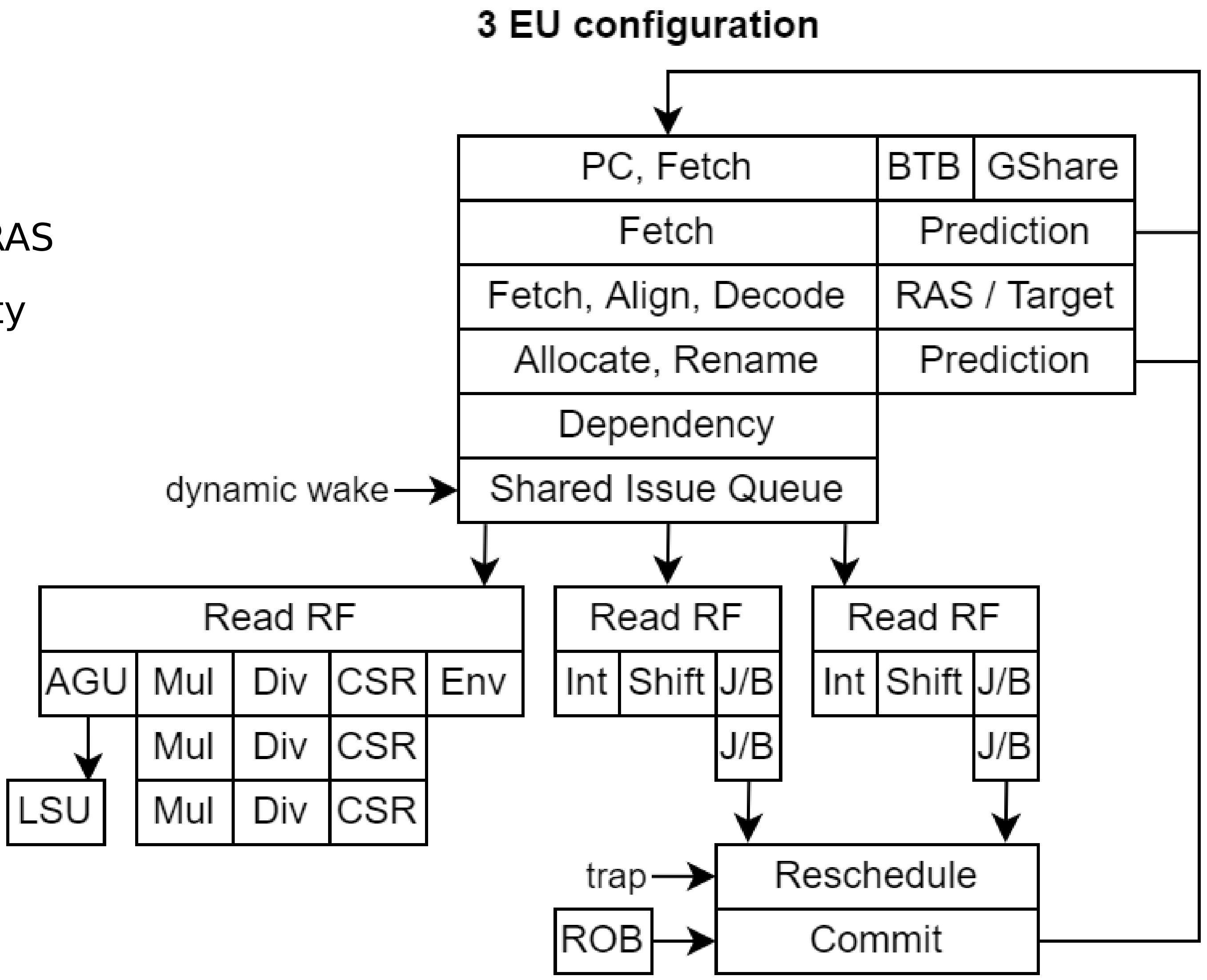
## Performances / Area

- RV32IMASU, 2 decode, 3 issue (Linux ready)
  - Dhrystone : 2.93 DMIPS/Mhz 1.65 IPC (-O3 -fno-common -fno-inline)
  - Coremark : 5.02 Coremark/Mhz 1.28 IPC
  - Embench-iot : 1.67 baseline 1.42 IPC (baseline = Cortex M4)
- On Artix 7 speed grade 3 :
  - 155 Mhz
  - 13.3 KLUT, 10.3 KFF, 12 BRAM, 4 DSP
- RV64IMASU, same FPGA/config
  - 137 Mhz
  - 17.9 KLUT, 12.5 KFF, 12 BRAM, 16 DSP

https://spinalhdl.github.io/NaxRiscv-Rtd/main/NaxRiscv/performance/index.html

# Architecture (from last slide)

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



## Main topic of the talk :

- Pipelining API
  - HardType
  - Naming
- Non usual toplevel
  - Plugins
  - Software interface
  - Multithreaded elaboration

## Pipelining API live demo

## Elaboration multithreading live demo

```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"),
)

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

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

Elaboration time list of Plugin

```scala
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"),
)

new NaxRiscv(plugins)
```

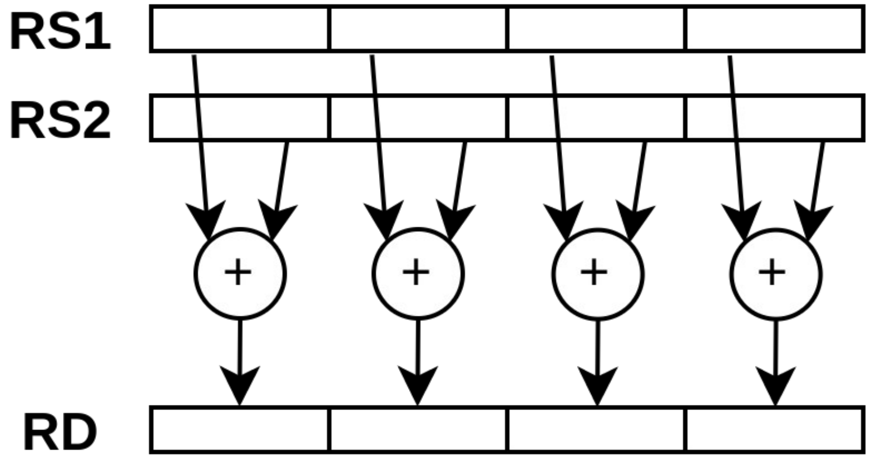
Wanna a second execution unit ?

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

## Plugin / SIMD ADD8

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

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

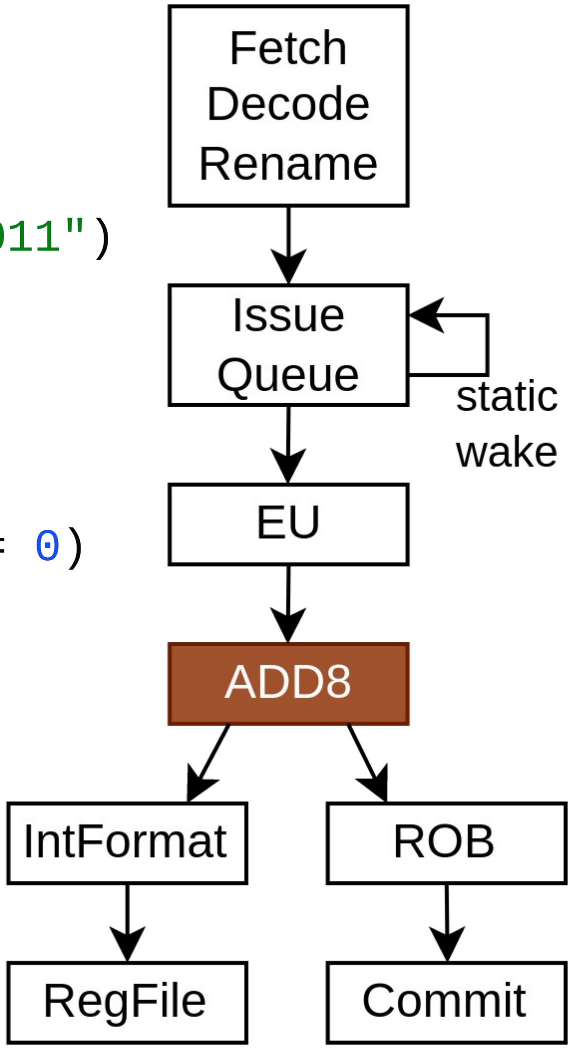


```scala
val setup = create early new Area{
    val eu = findService[ExecutionUnitBase](_.euId == euId)
    eu.retain()

    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)
}
```

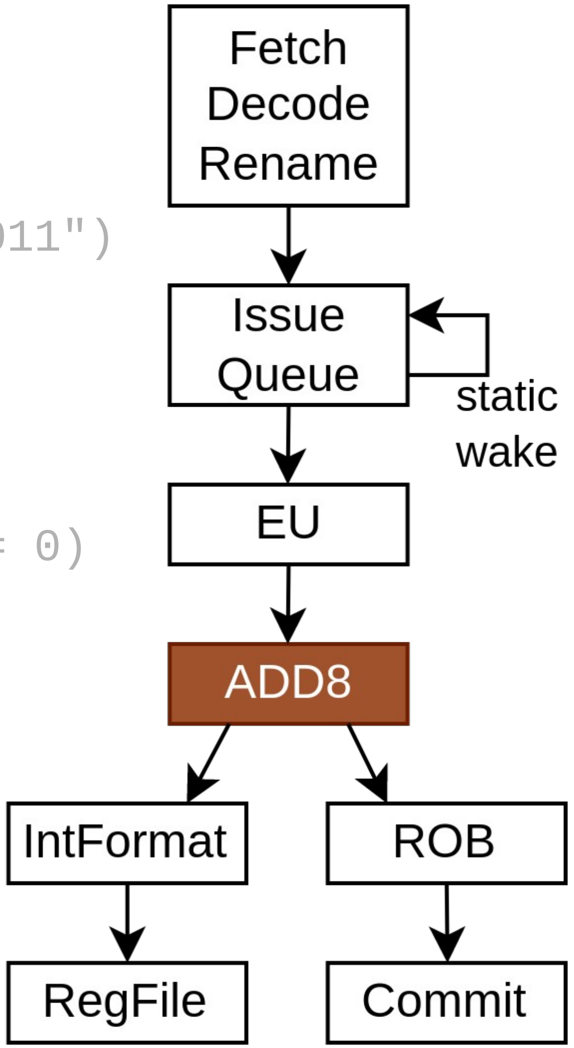


```
val setup = create early new Area{
    val eu = findService[ExecutionUnitBase](_.euId == euId)
    eu.retain()

    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)
}
```

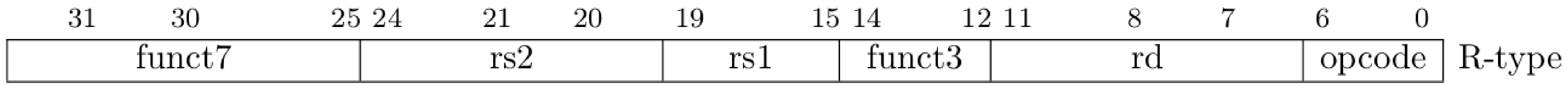


```
val setup = create early new Area{
    val eu = findService[ExecutionUnitBase](_.euId == euId)
    eu.retain()

    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)
}
```

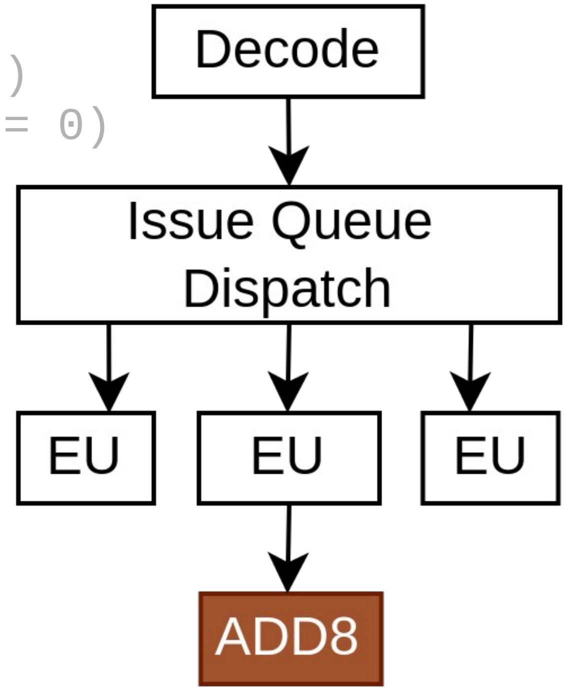


```
val setup = create early new Area{
    val eu = findService[ExecutionUnitBase](_.euId == euId)
    eu.retain()

    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)
}
```

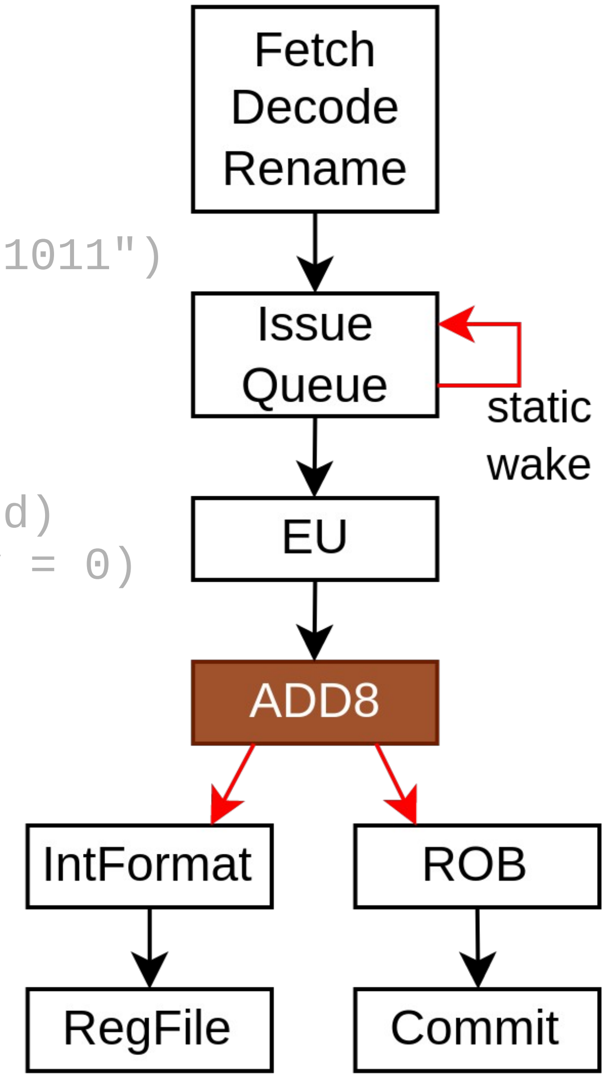


```
val setup = create early new Area{
    val eu = findService[ExecutionUnitBase](_.euId == euId)
    eu.retain()

    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)
}
```

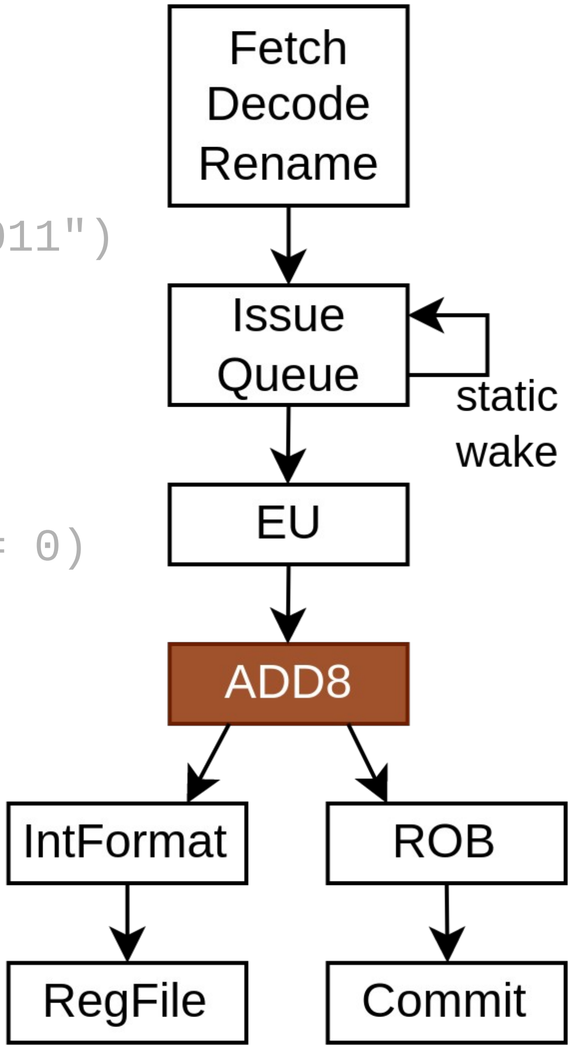


```
val setup = create early new Area{
    val eu = findService[ExecutionUnitBase](_.euId == euId)
    eu.retain()

    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)
}
```



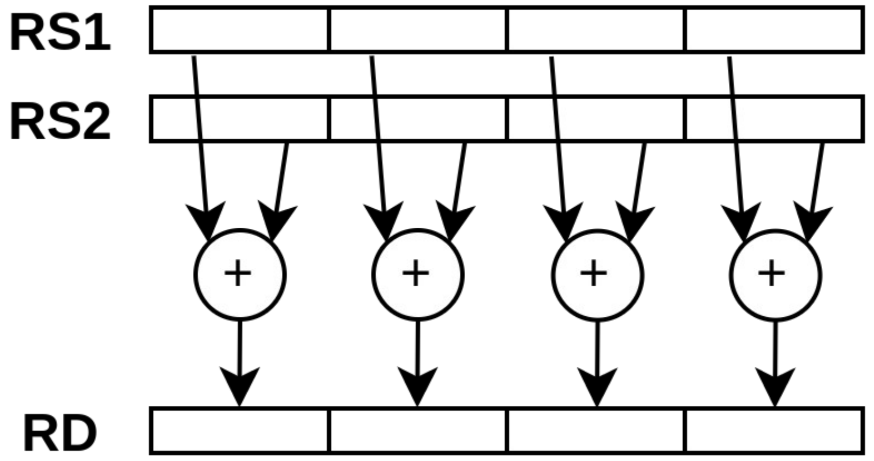
```
val logic = create late new Area{
    val eu        = setup.eu
    val writeback = setup.writeback
    val stage     = eu.getExecute(stageId = 0)

    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

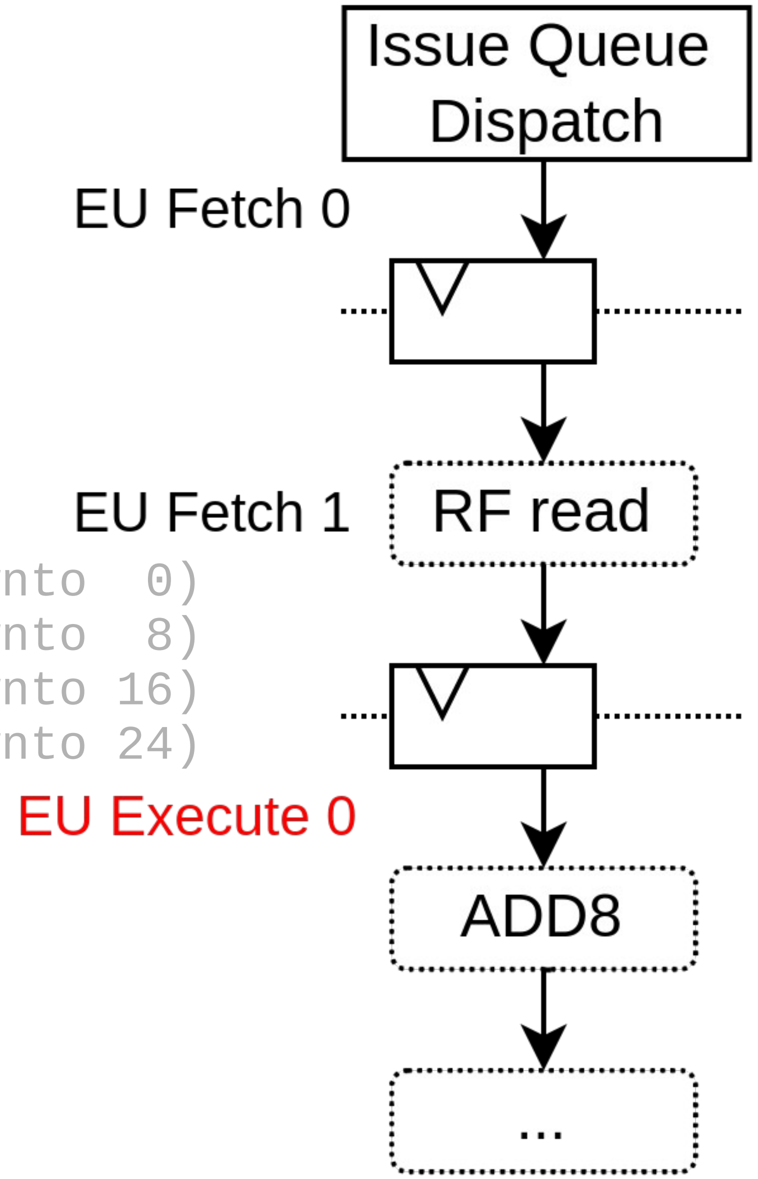
```
val logic = create late new Area{
    val eu        = setup.eu
    val writeback = setup.writeback
    val stage     = eu.getExecute(stageId = 0)

    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

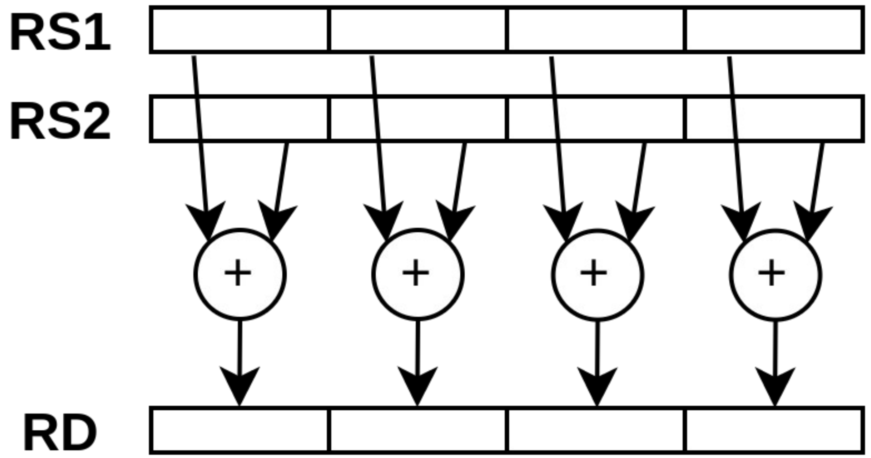
```
val logic = create late new Area{
    val eu        = setup.eu
    val writeback = setup.writeback
    val stage     = eu.getExecute(stageId = 0)

    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

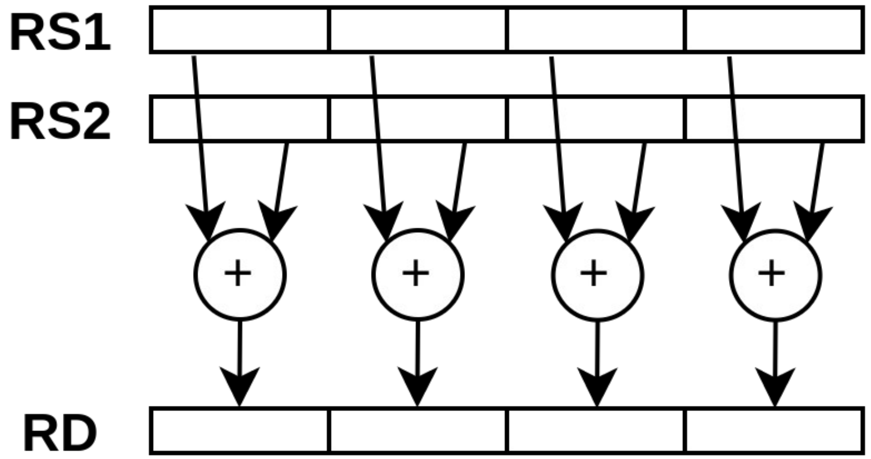
```
val logic = create late new Area{
    val eu        = setup.eu
    val writeback = setup.writeback
    val stage     = eu.getExecute(stageId = 0)

    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

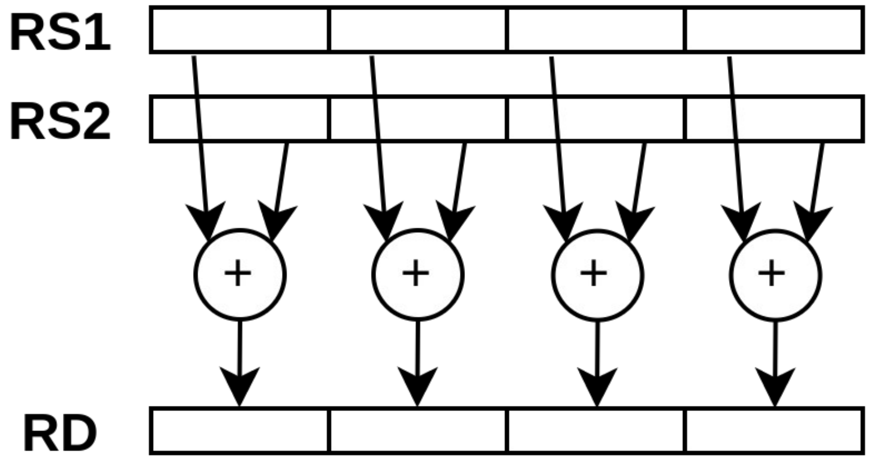
```
val logic = create late new Area{
    val eu        = setup.eu
    val writeback = setup.writeback
    val stage     = eu.getExecute(stageId = 0)

    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 plugins = ArrayBuffer[Plugin]()

...

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"),
)
```

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

new NaxRiscv(plugins)
```

## Status / Future

- JTAG / OpenOCD / GDB supported (RISCV External Debug Support 0.13.2)
- Planning :
  - 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