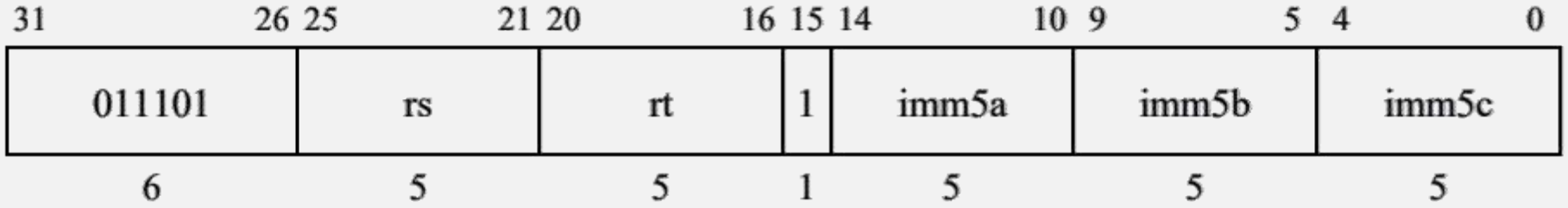
杭州電子科技大学
HANGZHOU DIANZI UNIVERSITY

NSCSCC 2023 PUA MIPS

叶剑豪 奚力丰 胡致尧

高效的添加一条指令

RLWINM



汇编格式： rlwinm　rt, rs, imm5a, imm5b, imm5c

指令功能： 带掩码的循环左移。

功能描述：

将 rs 寄存器里的 bit[31:0]循环左移 imm5a 位，随后根据 imm5b 和 imm5c 的值构建出由连续的 1 组成的掩码并与移位的结果按位与，将结果写入到 rt 寄存器。imm5b 和 imm5c 表示连续的 1 的起始位置与结束位置，例如：imm5b 与 imm5c 分别为 4 和 31，表示掩码从第 5 比特到第 31 比特是连续的 1，掩码为 0xFFFFFFE0；如果 imm5b 与 imm5c 分别为 30 和 4，则表示掩码从第 31 比特到第 4 比特是连续的 1,掩码为 0x8000001F。

高效的添加一条指令

1. Const.scala　　添加进指令表

```
// 比赛增加的指令
val EXE_RLWINM = 77.U(OP_WID.W)
```

2. Instruction.scala 添加指令识别码

```
def RLWINM      = BitPat("b011101???????????1???????????????")
```

3. Decoder.scala　　通过ListLookUp产生译码信号

```
RLWINM → List(INST_VALID, READ_ENABLE, READ_DISABLE, FU_ALU,
    EXE_RLWINM, WRITE_ENABLE, WRA_T2, IMM_LSE, SINGLE_ISSUE),
```

4. ALU.scala　　在运算单元中进行处理

高效的添加一条指令


```
val imm5a = src2(14, 10)
val imm5b = src2(9, 5)
val imm5c = src2(4, 0)
val mask  = Wire(Vec(32, Bool()))
for (i ← 0 until (32)) {
  when((imm5c ≥ i.U(5.W) && i.U(5.W) > imm5b) ||
       (i.U(5.W) > imm5b && imm5b > imm5c) ||
       (i.U(5.W) ≤ imm5c && imm5b > imm5c)) {
    mask(i) := true.B
  }.otherwise {
    mask(i) := false.B
  }
}
val rlwinmv = (src1 << imm5a) | (src1 >> (32.U - imm5a))
val rlwinmask = mask.asUInt

io.result := MuxLookup(
  op,
  0.U,
  Seq(
    EXE_RLWINM → (rlwinmv & rlwinmask),
```

不言自证

不证自明

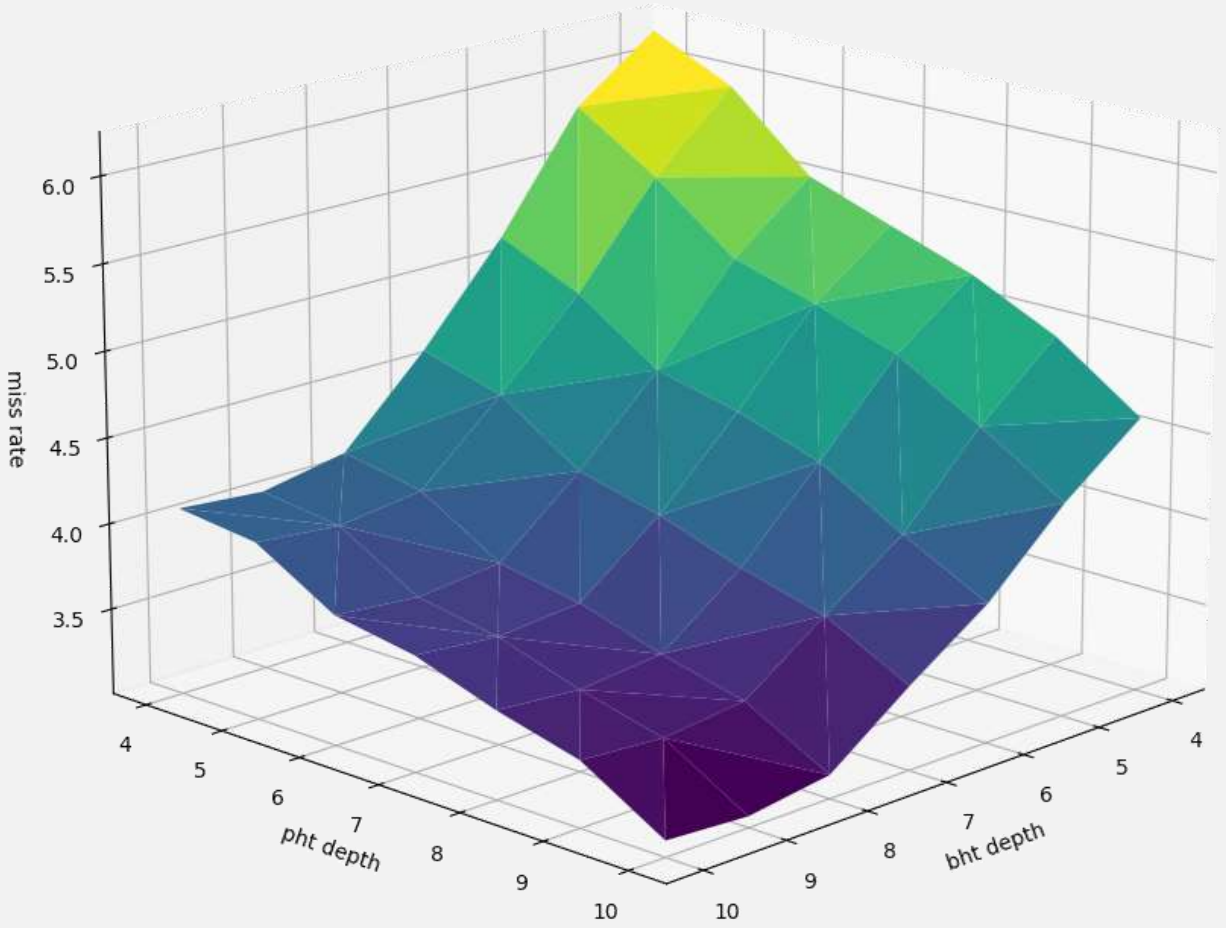
```
for i in [0..31]
    if (
        (imm5c>=i   and   i>imm5b) or
        (i>imm5b    and   imm5b>imm5c) or
        (i<=imm5c   and   imm5b>imm5c)
        ):
            mask[i]=1
    else
            mask[i]=0
GPR[rd] ← rotl(GPR[rj], imm5a) & mask
```

基于Verilator仿真器进行设计空间探索

VERILATOR

+

简单Bash脚本



Verilator仿真器提高设计空间探索速率

VS

- 修改顶层仿真框架
- 添加处理器核端口
- 编写bash测试脚本
- 自动生成性能数据

- 使用Vivado进行仿真
- 手动查看信号数据

20 min vs 15 hour

Chisel带来强大的可配置性

多种参数任意组合

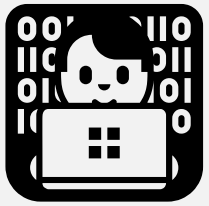
```
case class CpuConfig(
    val build: Boolean = true,                  // 是否为build模式
    val hasCommitBuffer: Boolean = false,       // 是否有提交缓存
    val decoderNum: Int = 2,                    // 同时访问寄存器的指令数
    val commitNum: Int = 2,                     // 同时提交的指令数
    val fuNum: Int = 2,                         // 功能单元数
    val instFetchNum: Int = 2,                  // iCache取列的指令数量
    val instFifoDepth: Int = 8,
    val writeBufferDepth: Int = 16,
    val mulClockNum: Int = 2,
    val divClockNum: Int = 8,
    val branchPredictor: String = "adapt
)
case class BranchPredictorConfig(
    val bhtDepth: Int = 8,
    val phtDepth: Int = 10,
)
```

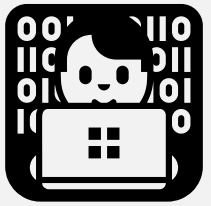
```
case class CacheConfig(
    nway: Int = 2,
    nbank: Int = 8,
    nset: Int,
    bankWidth: Int, // bytes per bank
) {
  val indexWidth      = log2Ceil(nset)
  val bankIndexWidth  = log2Ceil(nbank)
  val bankOffsetWidth = log2Ceil(bankWidth)
  val offsetWidth     = bankIndexWidth + bankOffsetWidth
  val tagWidth        = 32 - indexWidth - offsetWidth
  val tagvWidth       = tagWidth + 1
  val bankWidthBits   = bankWidth * 8
  val burstSize       = 16
  val ninst           = 4
  require(isPow2(nset))
  require(isPow2(nway))
  require(isPow2(nbank))
  require(isPow2(bankWidth))
```

指令队列解耦前后端


```
class InstFifo(implicit val config: CpuConfig) extends Module {
  val io = IO(new Bundle {
    val do_flush = Input(Bool())

    val icache = Flipped(Vec(config.instFetchNum, Decoupled(new FifoUnit())))
    val decoder = Vec(config.decoderNum, Decoupled(new FifoUnit()))
  })
```

前端取指 → 指令队列 → 后端执行





集中式控制 与 分布式控制并存

- 大框架上通过整体暂停实现集中控制
- 小框架中通过握手信号实现分布控制

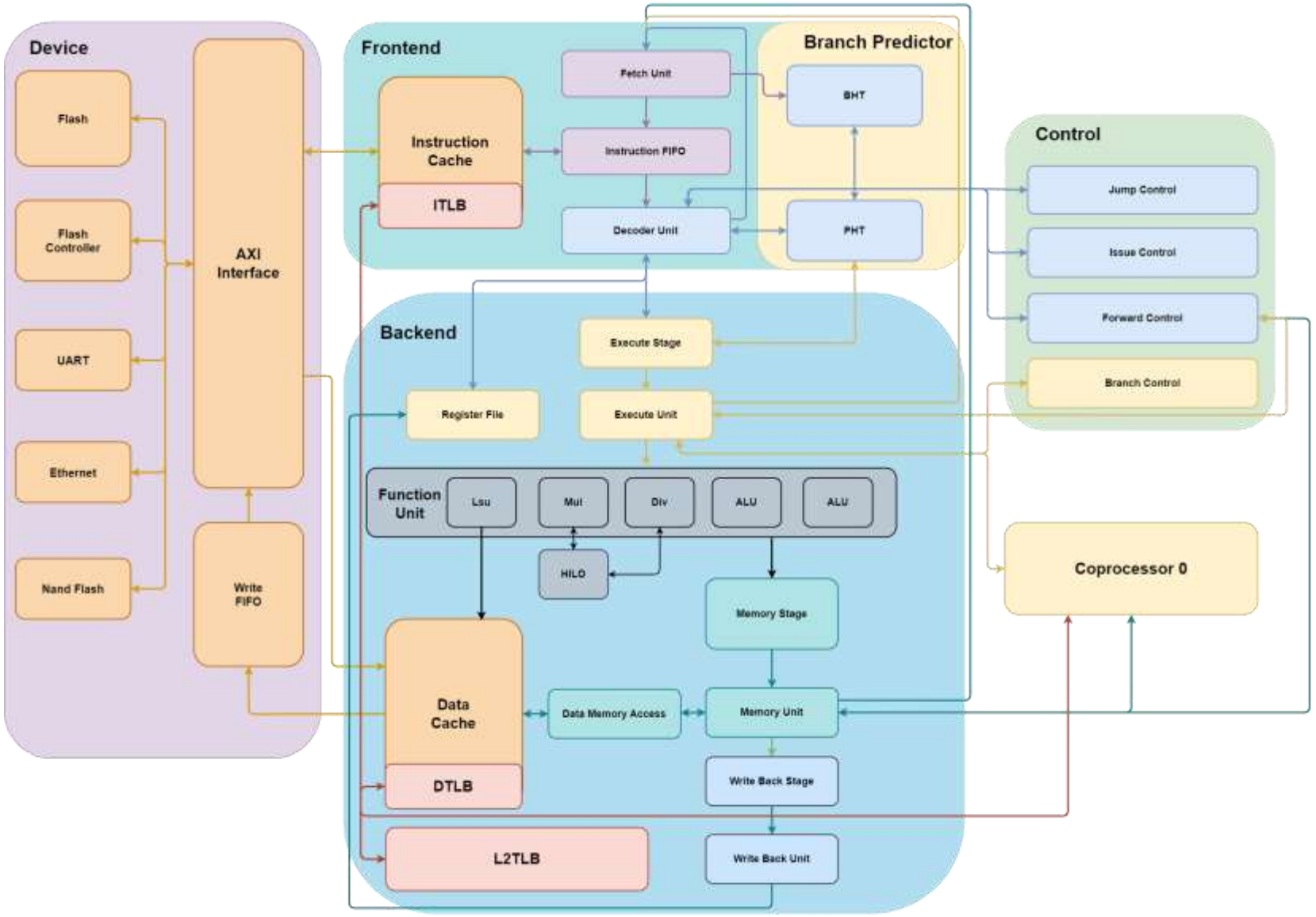
```
class Mul(implicit val config: CpuConfig) extends Module {
  val io = IO(new Bundle {
    val src1          = Input(UInt(DATA_WID.W))
    val src2          = Input(UInt(DATA_WID.W))
    val signed        = Input(Bool())
    val valid         = Input(Bool())

    val ready  = Output(Bool())
    val result = Output(UInt(HILO_WID.W))
  })
```

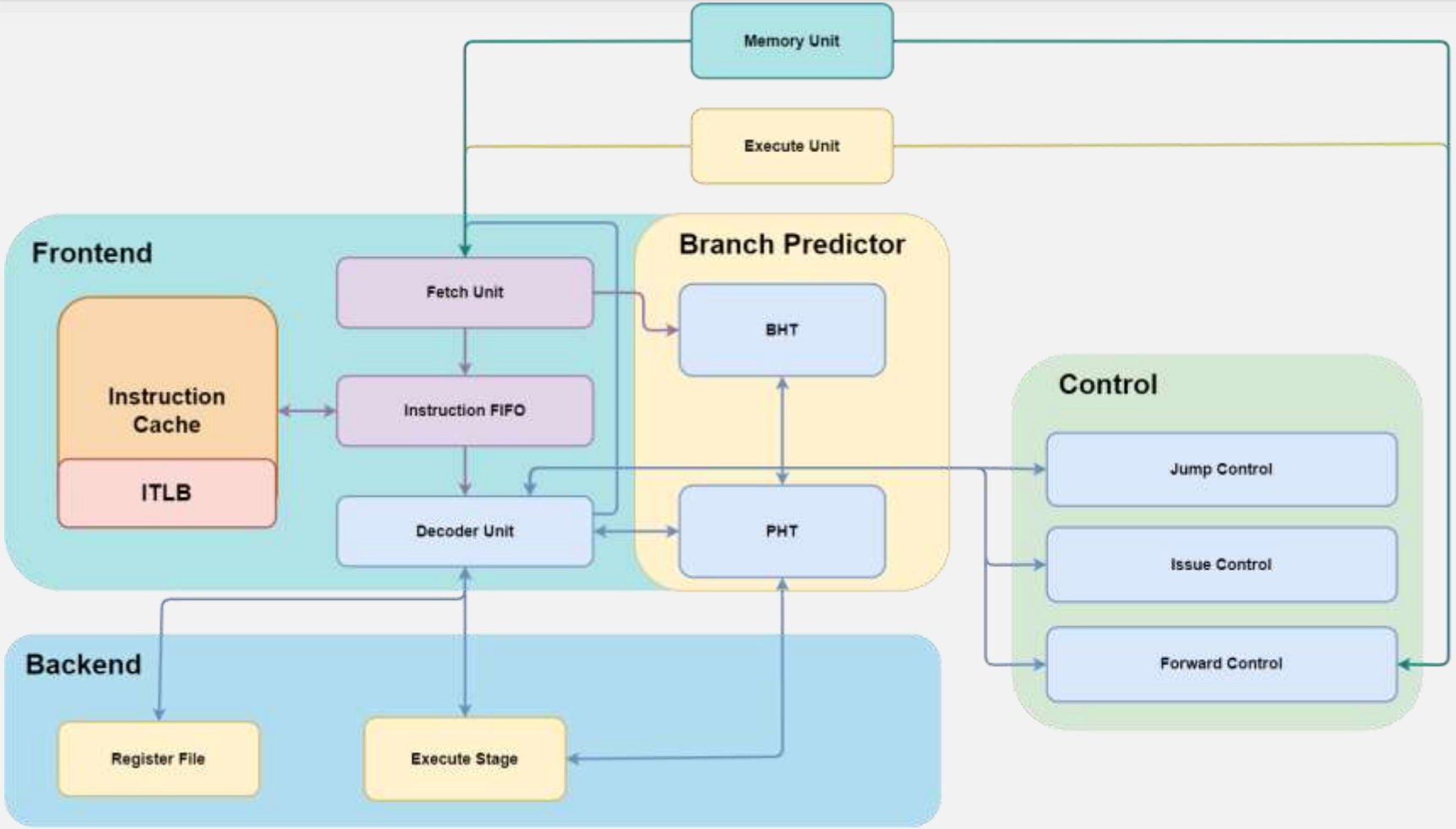
取指

写回

运算

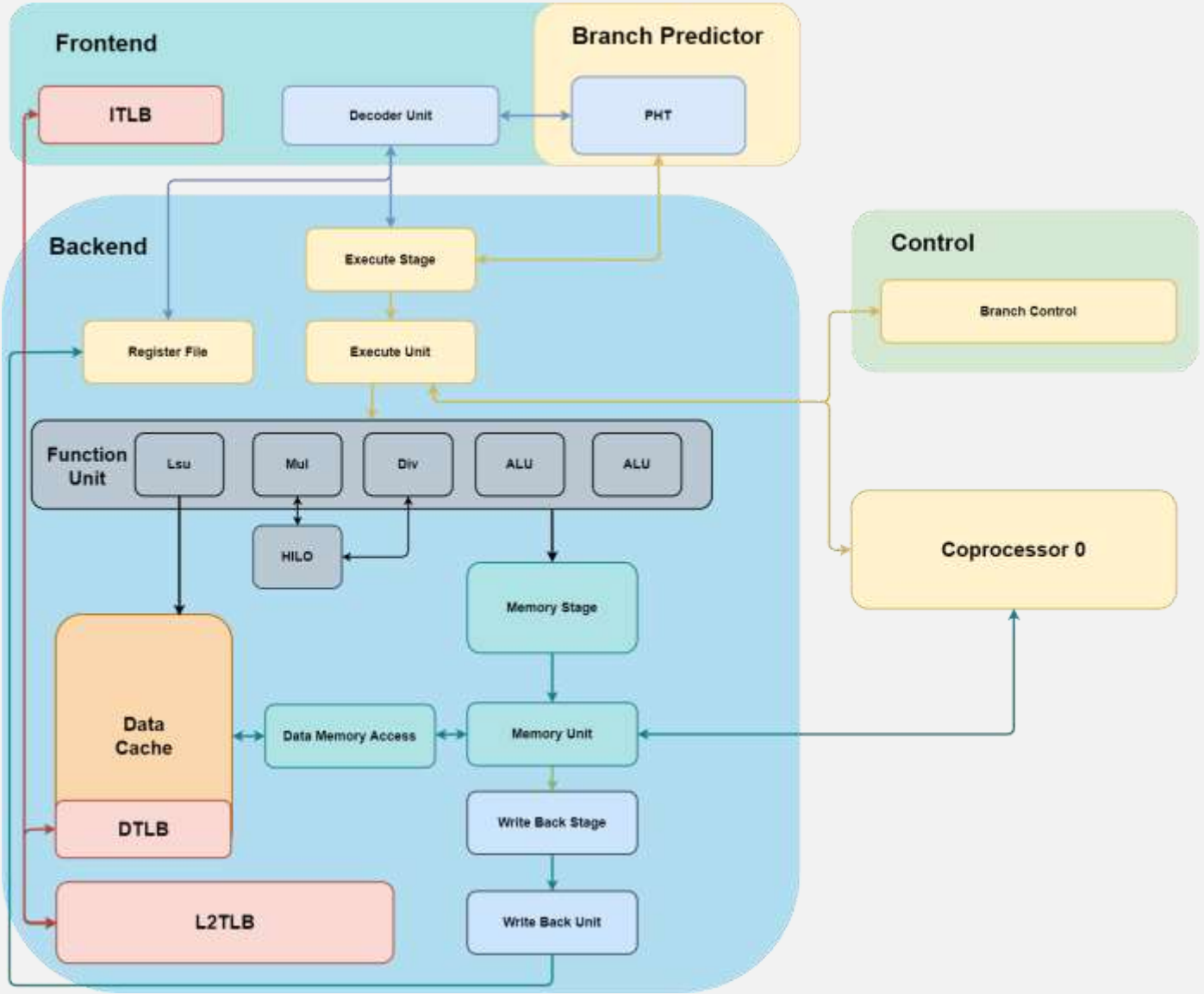
处理器总体架构


Device
Flash
Flash Controller
UART
Ethernet
Nand Flash
AXI Interface
Write FIFO
Frontend
Fetch Unit
Instruction Cache
ITLB
Instruction FIFO
Decoder Unit
Branch Predictor
BHT
PHT
Control
Jump Control
Issue Control
Forward Control
Branch Control
Backend
Execute Stage
Register File
Execute Unit
Function Unit
Lsu
Mul
Div
ALU
ALU
HILO
Memory Stage
Coprocessor 0
Data Cache
DTLB
Data Memory Access
Memory Unit
Write Back Stage
L2TLB
Write Back Unit



- 指令Fifo解耦取指
- 双发策略
  - 例外
  - RAW冲突
  - 访存冲突
  - 延迟槽处理

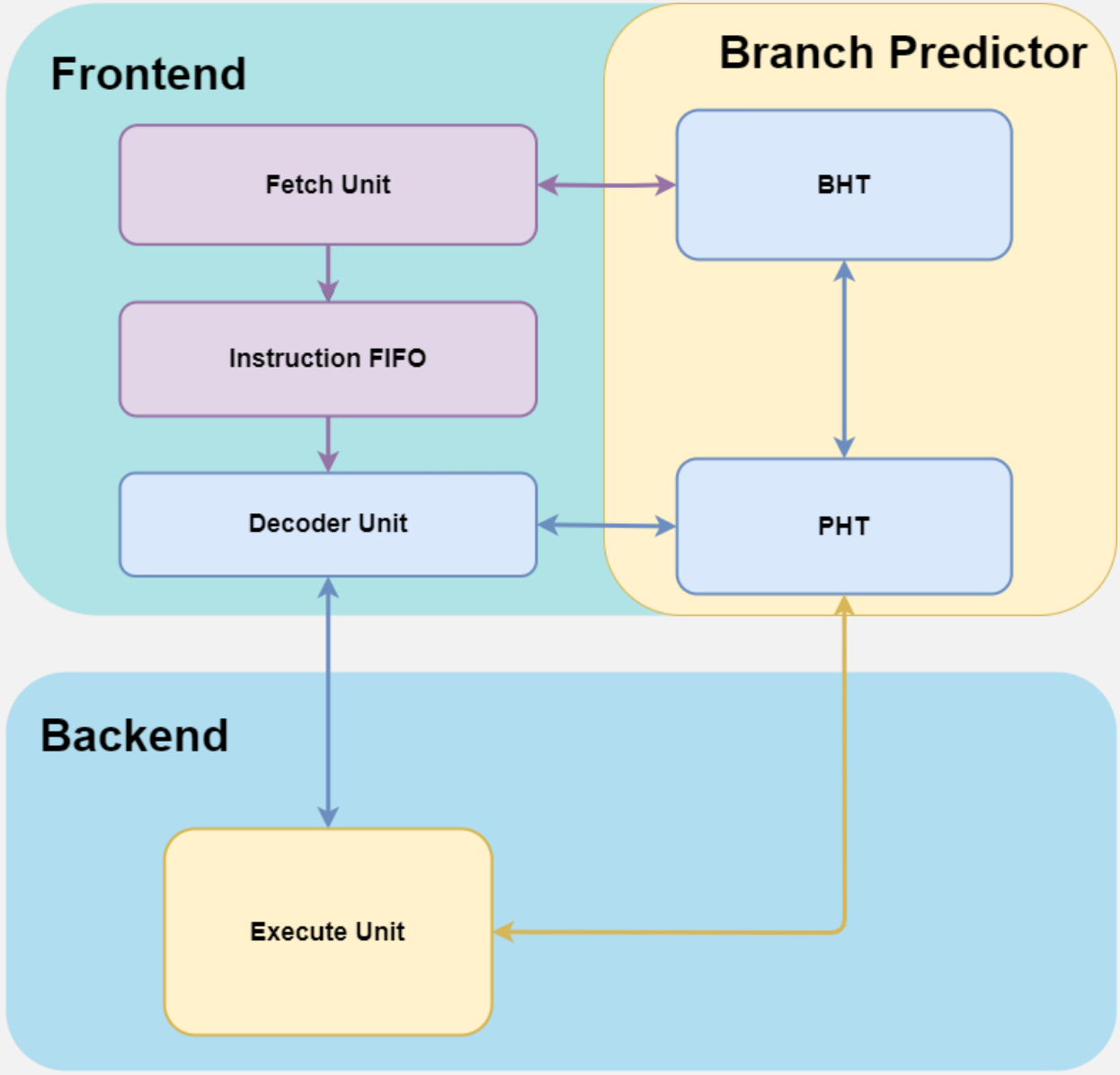
后端



- 两级访存
  - 执行级发出访存请求
  - 访存级收到访存数据
- 两级TLB
  - 访问缓存时访问TLB1
  - TLB1发生缺失找TLB2
  - TLB2处理缺失与例外
  - 频率提高13%

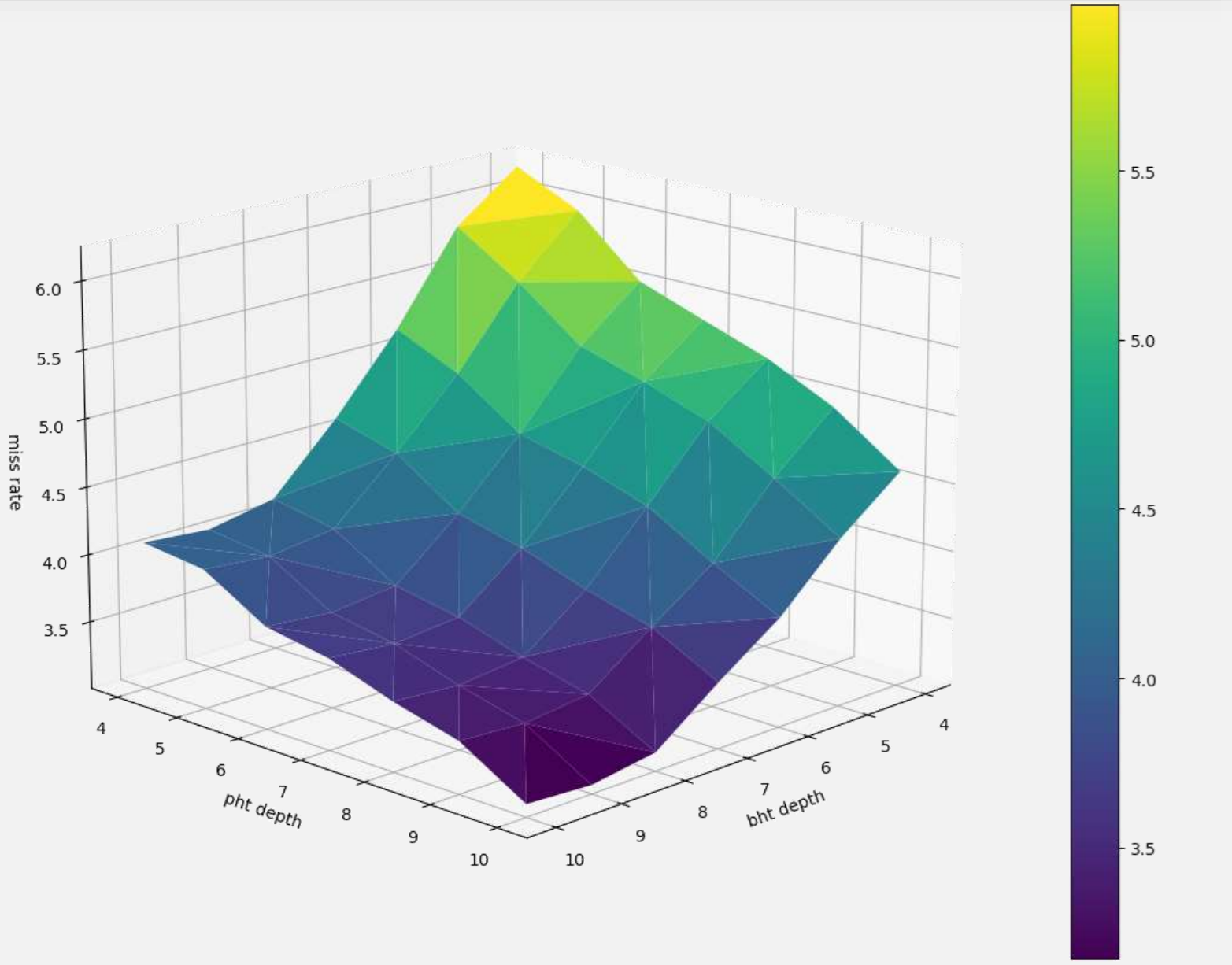
分支预测

- 局部历史分支预测
- 2bit饱和更新
- 两级预测不降频
  - 取指阶段获得分支历史
  - 译码阶段进行分支预测
  - 执行阶段进行分支更新
- 命中率 94.48%
- 设计空间探索



分支预测

- 局部历史分支预测
- 2bit饱和更新
- 两级预测不降频
  - 取指阶段获得分支历史
  - 译码阶段进行分支预测
  - 执行阶段进行分支更新
- 命中率 94.48%
- 设计空间探索

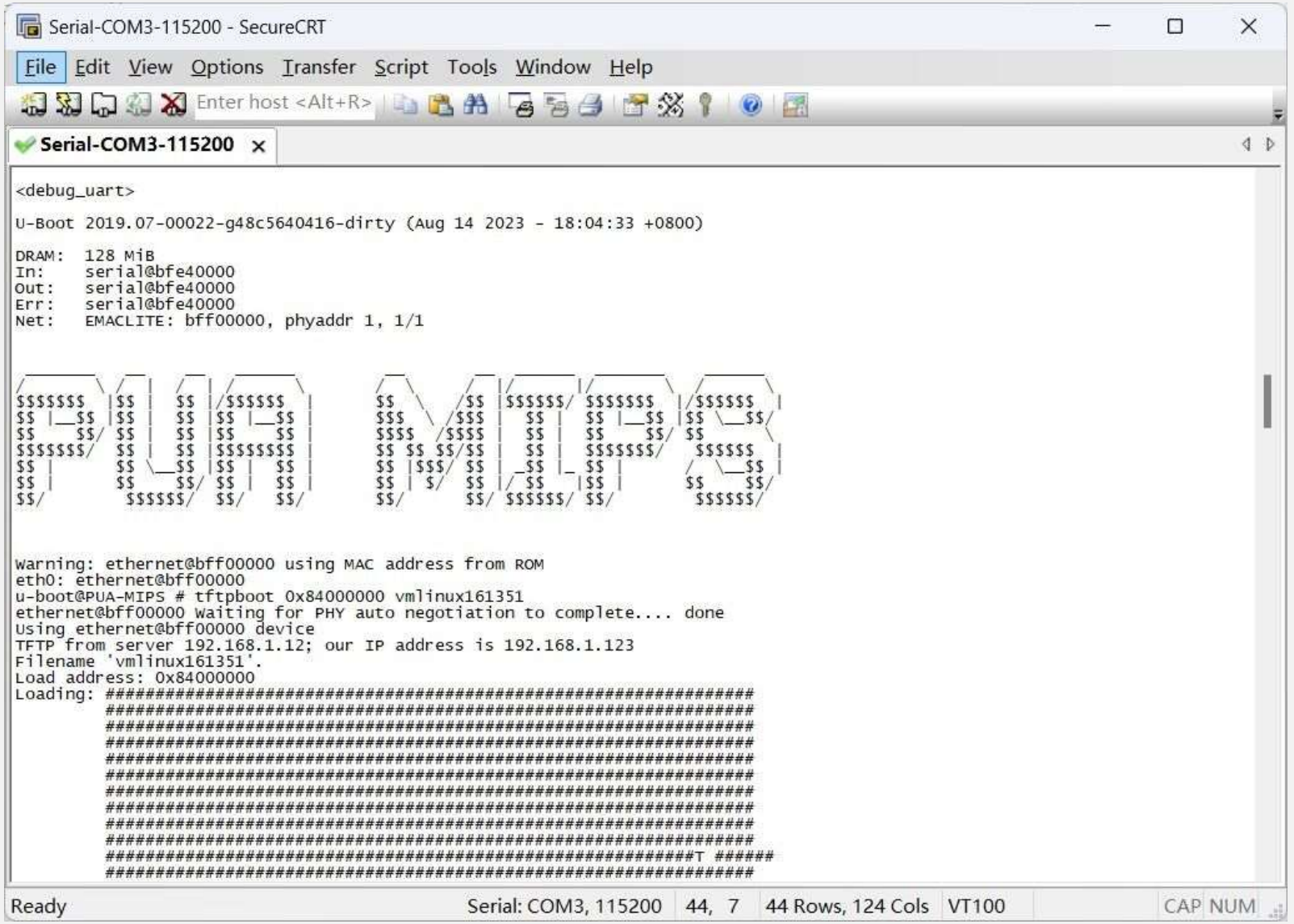


缓存

测试集	ICache hit	DCache hit
bitcount	99.83%	96.91%
bubble sort	99.97%	99.79%
coremark	99.36%	95.59%
crc32	99.98%	97.85%
dhrystone	99.89%	81.88%
quick sort	99.96%	97.85%
select sort	99.98%	99.33%
sha	99.94%	98.89%
stream copy	99.65%	93.37%
stringsearch	99.96%	90.52%
几何平均值	99.72%	95.00%

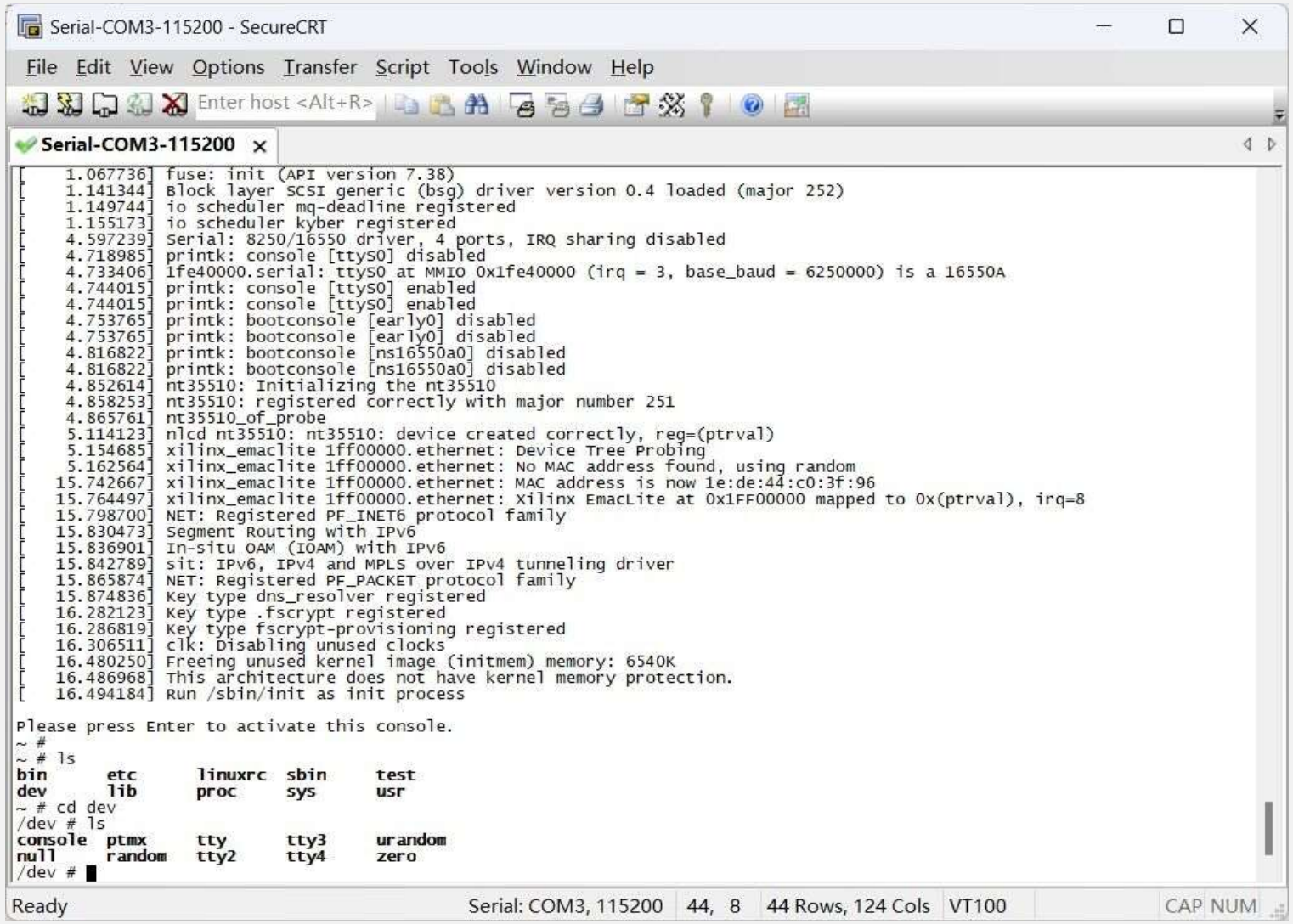
- 指令缓存
  - 命中率 99.72%
- 数据缓存
  - 命中率 95.00%
  - Victim Cache
    - 缺失率 ↓14.32%
  - Write Fifo
    - 阻塞率 ↓21.71%

启动U-Boot



- 移植处理
  - 去除Branch-likely指令
  - 编写设备树

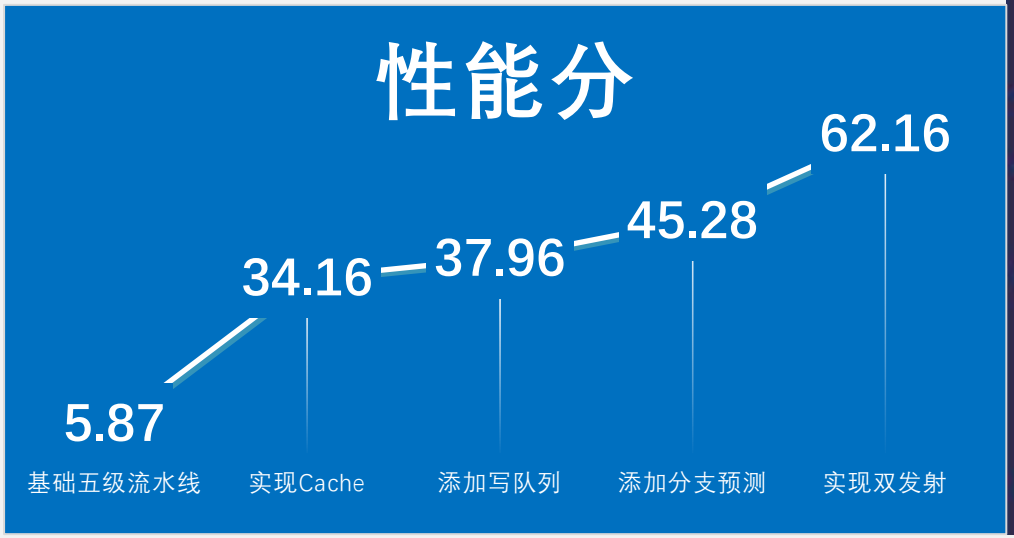
启动Linux

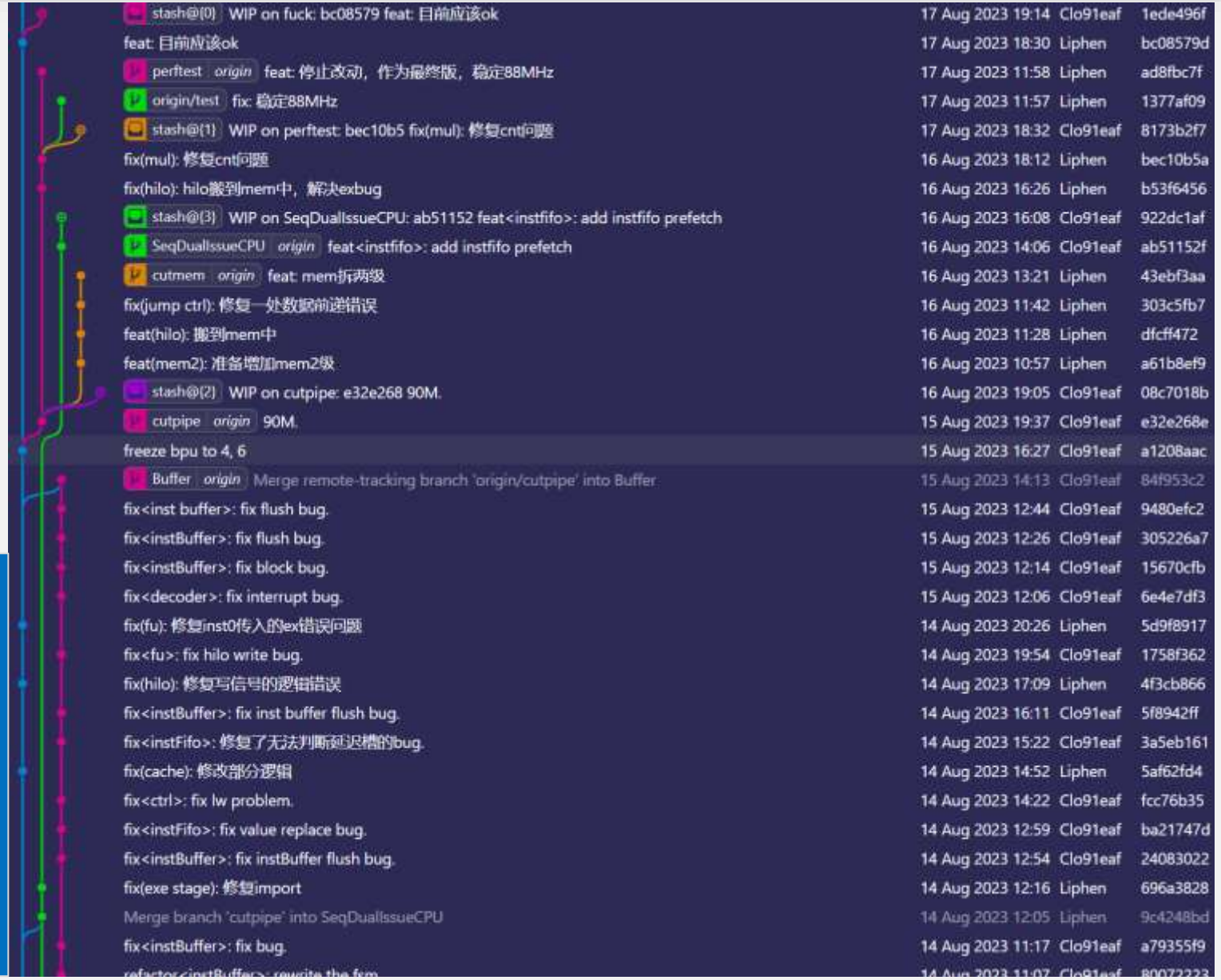


- 移植处理
  - 去除Branch-likely指令
  - 编译时关闭浮点支持
  - 编写设备树

设计历程

- 20个历史分支
- 513次提交
- 130次问题修复
- 80次代码重构
- 20次性能优化





谢谢！