Computer Systems Design

Lesson 12

Dynamic branch prediction

Open-source superscalar OoO CPU cores

Alexander Antonov, Assoc. Prof., ITMO University

Hangzhou, 2025

ITMO

Outline the lesson

- Branch prediction goals
- Branch predictors
  - Branch target buffer
  - Return address stack
  - Adaptive branch predictors
  - TAGE
  - Neural branch predictors
  - Combining branch predictors
- Open-source superscalar OoO CPU cores

ITMO

Dynamic branch prediction

ITMO

Branch prediction: what and how to predict?

Data to predict:

1) branch outcome (taking/not taking branch)
2) branch target address

Typically used information:

- current instruction address (PC)
- cached branch target addresses
- branch history (shift register with previous taken/not taken outcomes)
- accumulated statistics for current PC/history combinations

To reduce resource requirements for statistics accumulation, instruction address and history are ***hashed***

Problem: ***interference*** between aliased branches

ITMO

Static branch prediction

Never branch

- effectively not making any predictions
- simplest to implement, but performs badly

Backwards taken/forwards not-taken (BTFNT)

- “backwards taken” - good for loops (>90% precision)
- other branches – insufficient (50% precision)

In total, static schemes have insufficient performance (70-80% precision)

Adaptation to runtime needed (>99% achievable)

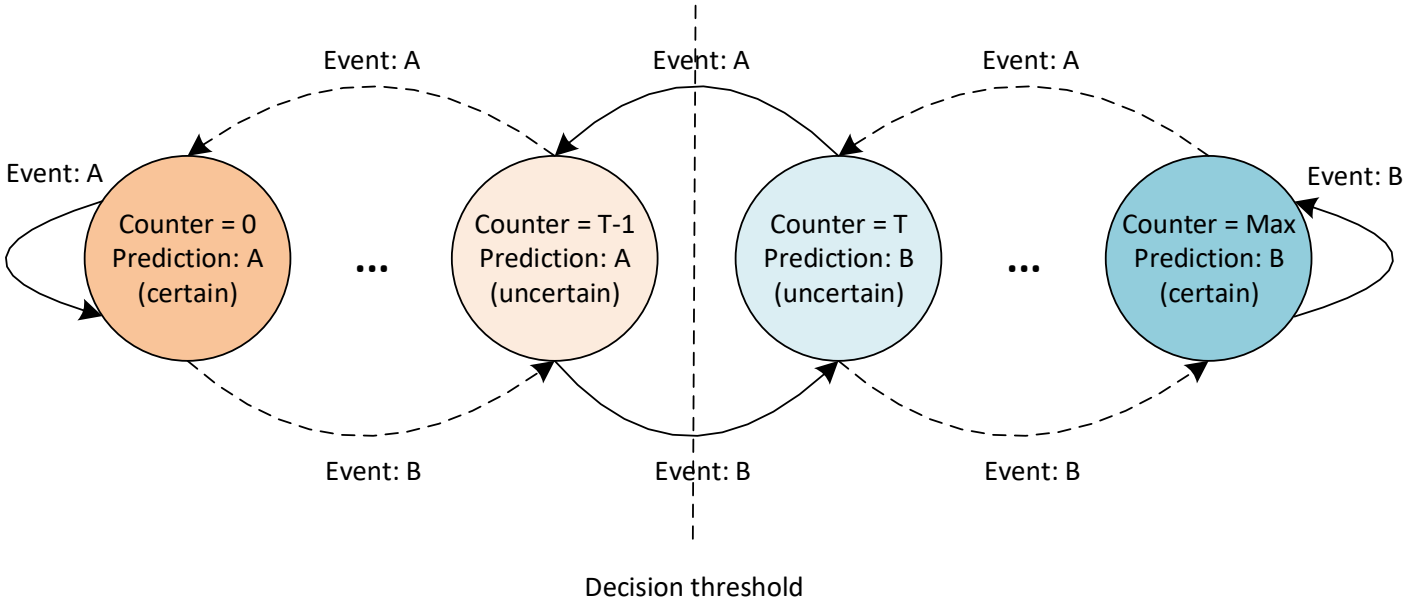
Statistics accumulation: saturation arithmetic

Principle: ***overflow*** is replaced with ***saturation***

min/max values preserved

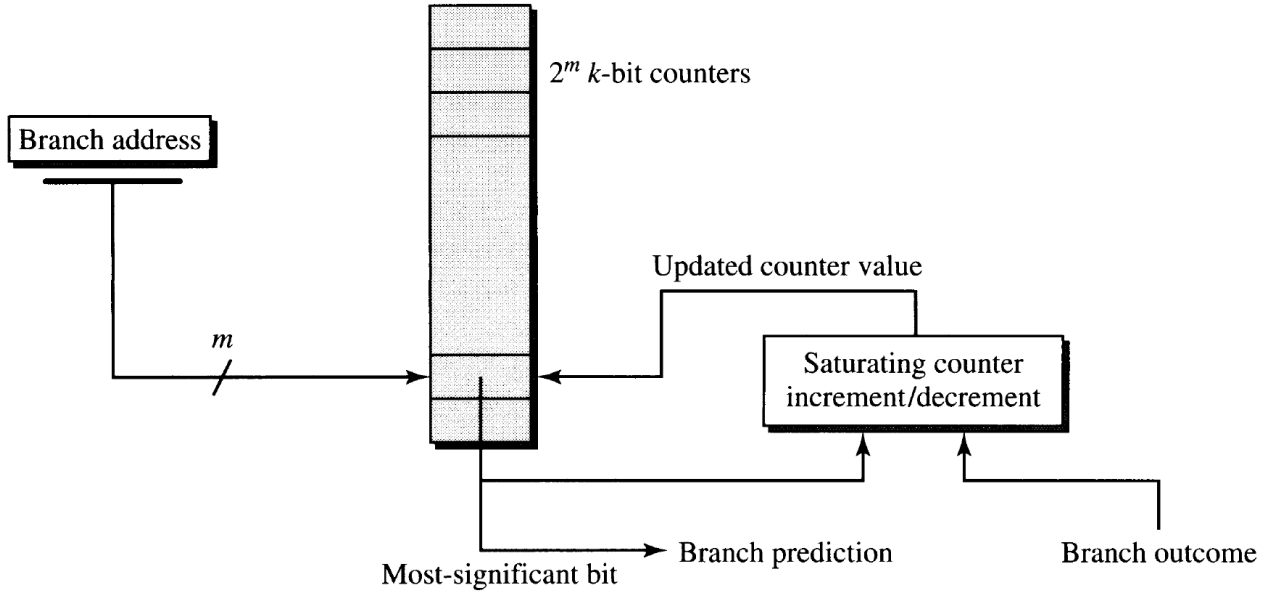
e.g. max value: `0x7F`*;* `0x70 + 0x20 = 0x7F` *(not* `0x10` *or* `0x90`*)*

Prediction is based on accumulated statistics with increasing confidence for monotonic events (hysteresis, inertia)



Basic branch outcome predictor: Smith's predictor (1981)

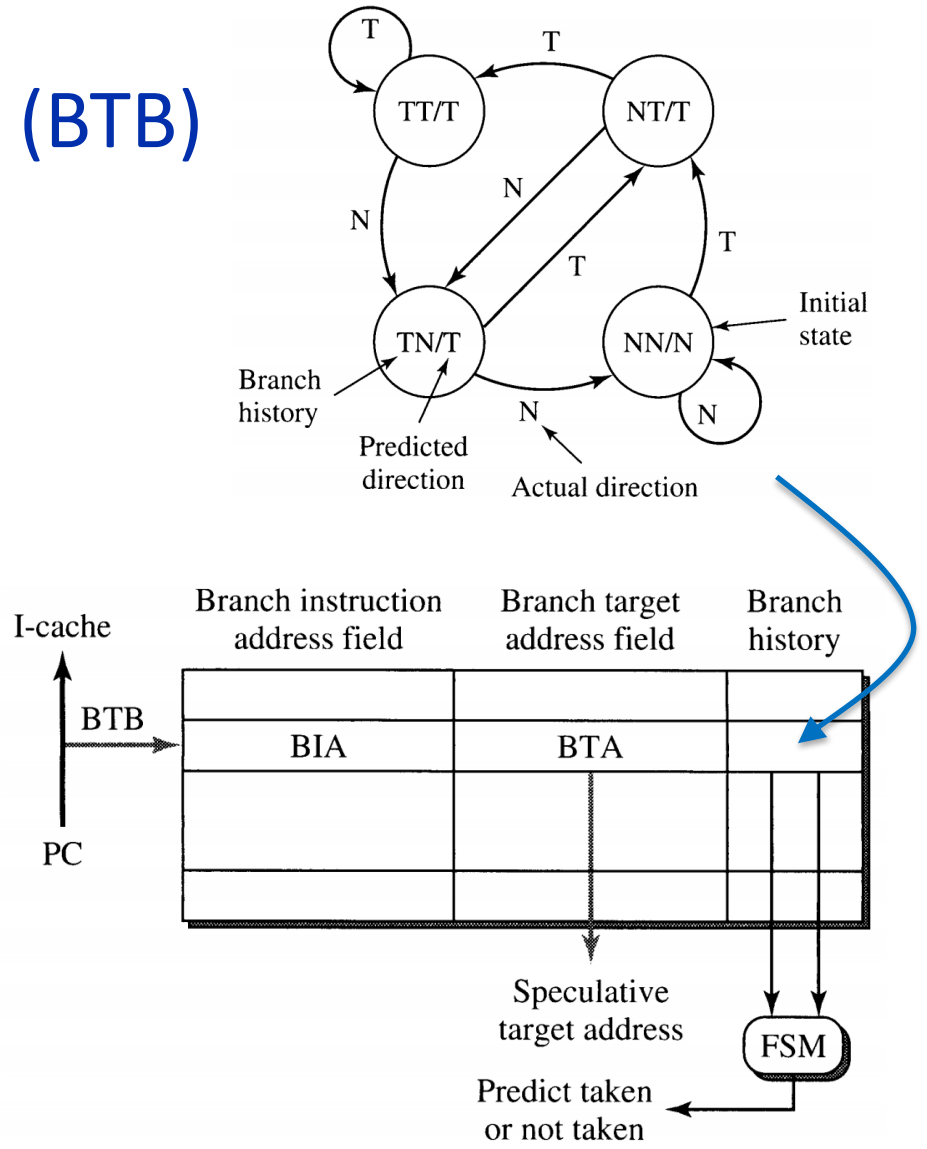
- branch history is implemented as saturating counters
- accessed by instruction identifier (PC)
- MSB of branch counter is used as prediction



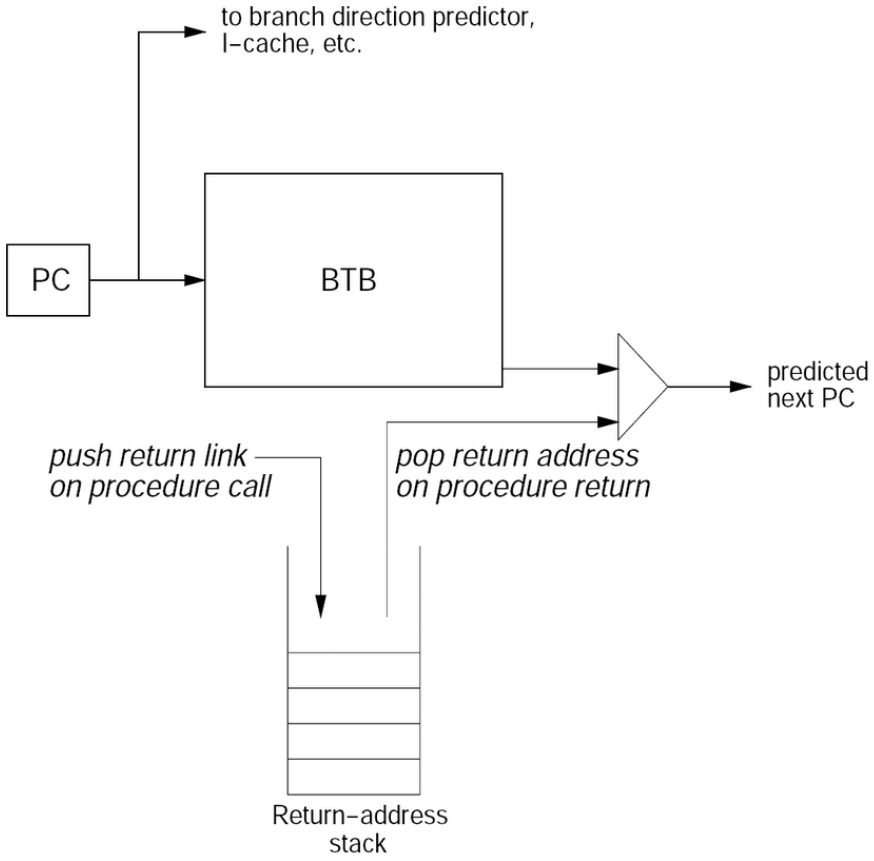
Picture: Shen, J. P., & Lipasti, M. H. Modern processor design: fundamentals of superscalar processors. Waveland Press, 2013

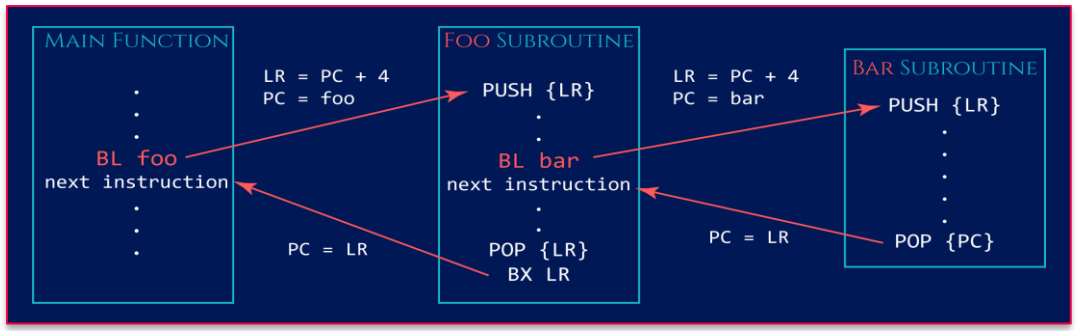
Target prediction: Branch Target Buffer (BTB)

- cache for taken branches
- branch history is implemented as saturating counter (usually biased towards taking)
- accessed by instruction identifier (PC)
- if branch predicted taken: branch target fetched and used as PC for next instruction



Return Address Stack (RAS)





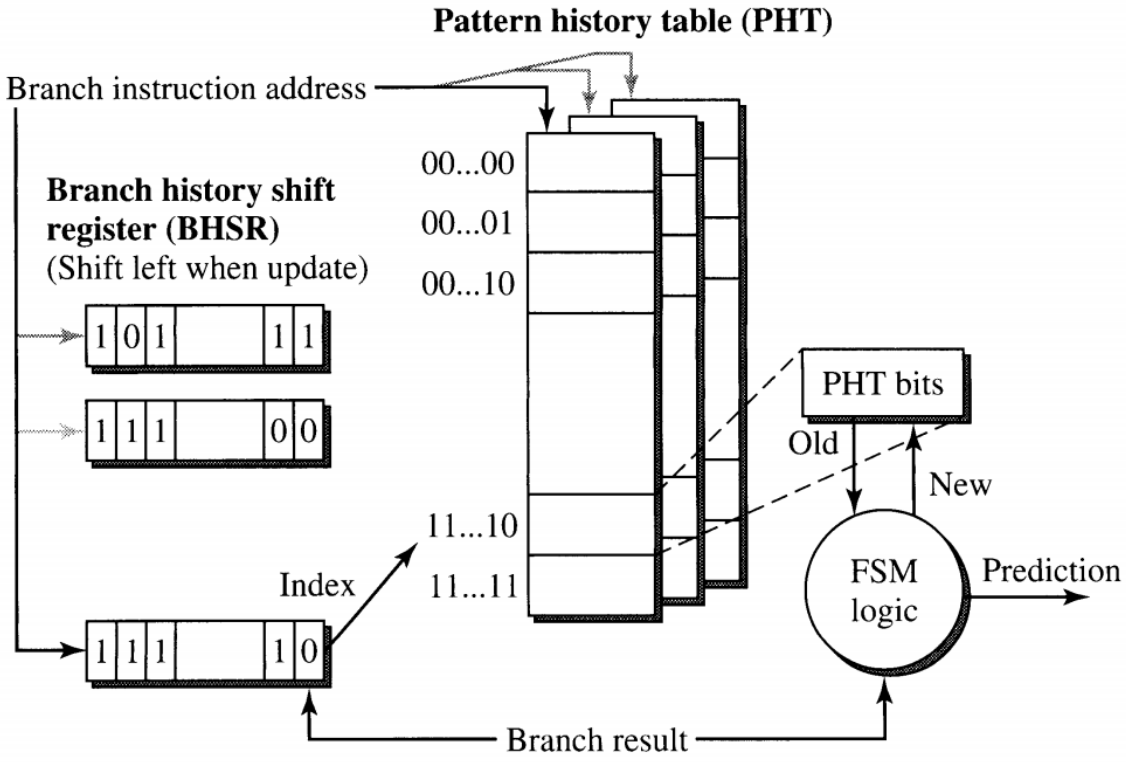
- Function: stack saving return link when calling procedures
- Activates: when returning from procedure
- Prediction: return from procedure will take place to instruction right after the call

ITMO

Two-level adaptive branch predictor (1991)

Observation: branch behavior correlates well with previous branch patterns

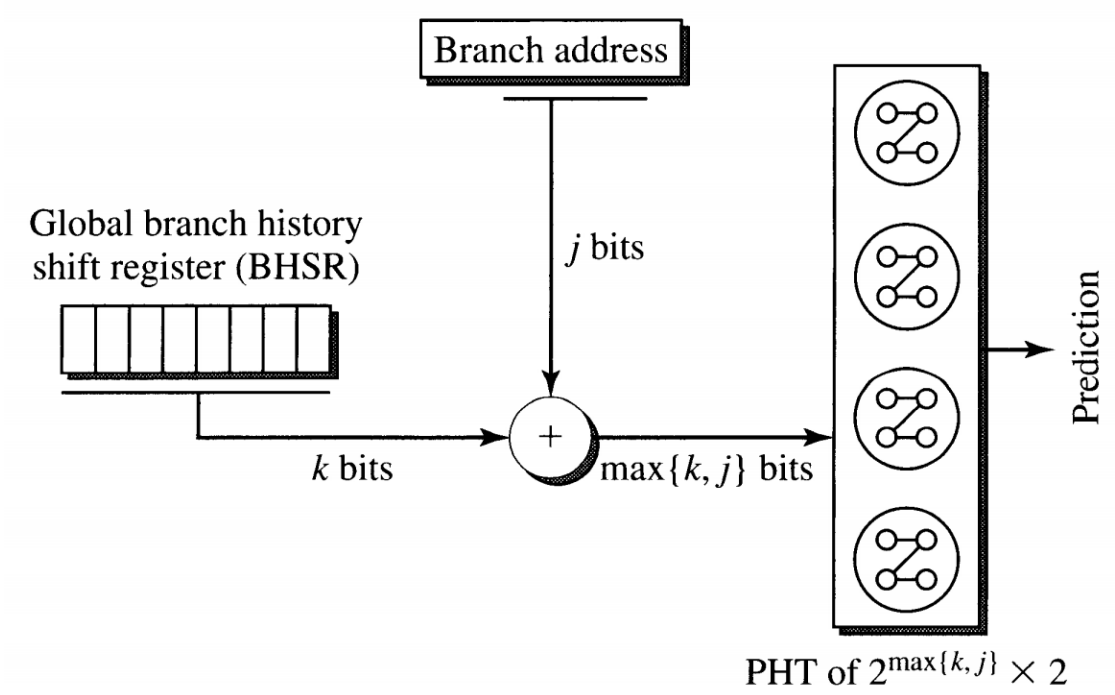
- **Idea**: ***selective*** statistics accumulation for ***branch instruction and branch history***
- PHT is a 2D table (branch instruction x branch history)
- BHSR can be global (one) or multiple individual for each branch (in picture)
- >95% prediction accuracy achievable



Gshare (McFarling, 1993)

Observation: correlated predictor can be ***simplified*** without significant loss of quality

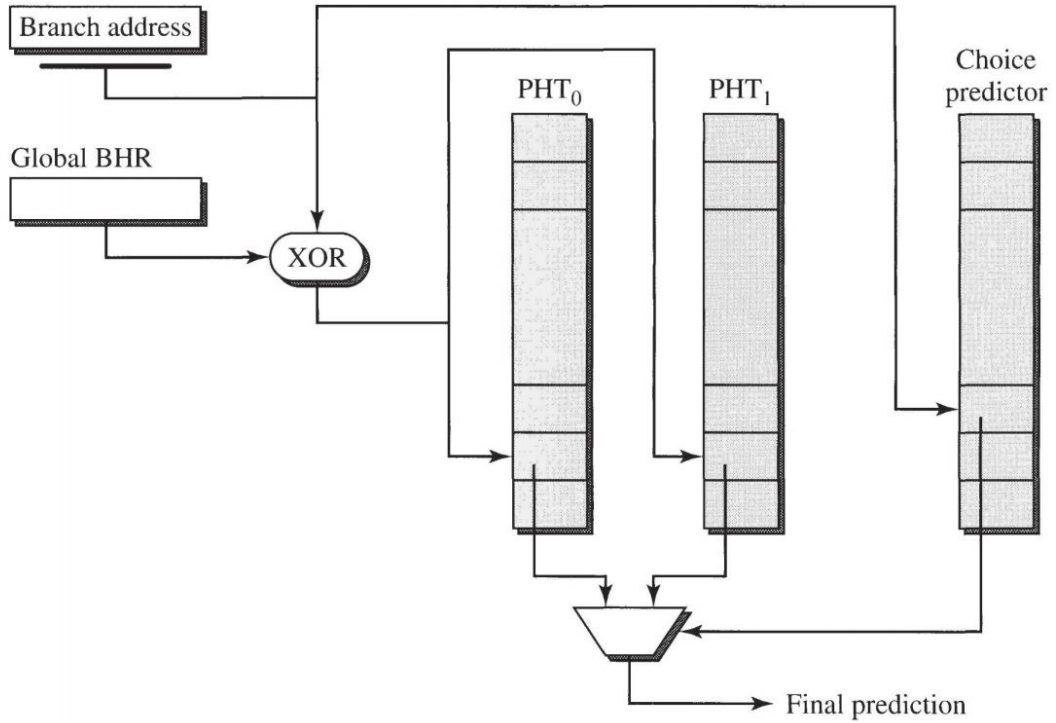
- **Idea**: global BHSR is ***hashed*** (XOR) with least significant bits of branch address
- Selectivity of branch address / BHSR is partially preserved
  issue: collisions (tolerable)
- Offers good complexity/ accuracy trade-off



Bi-mode predictor (1997)

Observation: some branches have strong “taken” bias, others – “non-taken”.
For these different branches aliases are destructive.

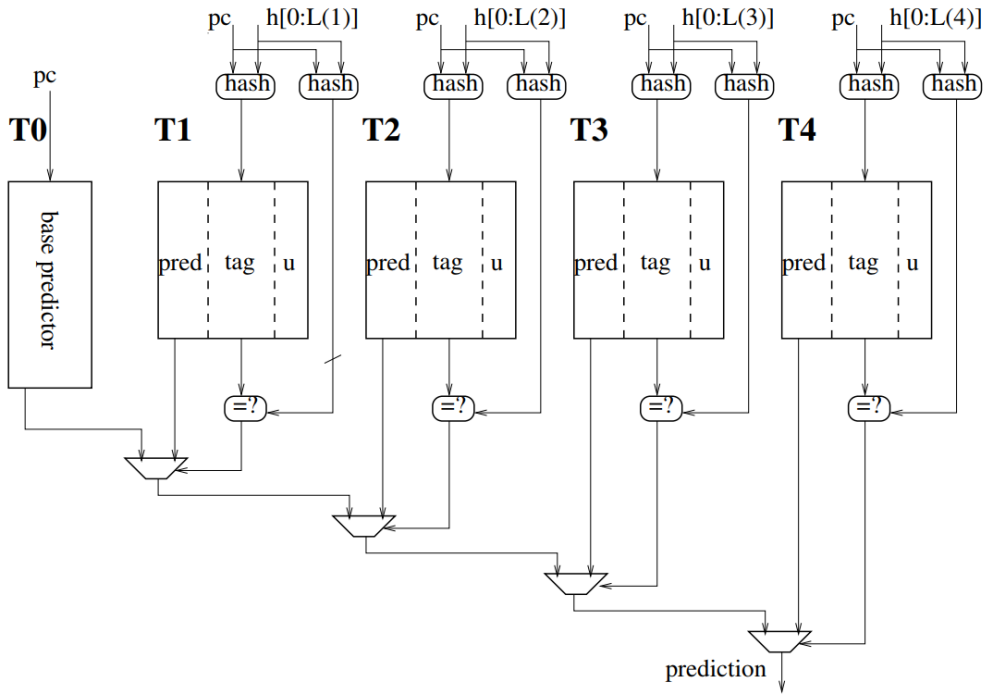
- Branches with “taken” and “non-taken” bias are placed to different PHTs
- Choice predictor (saturating counter) selects what PHT to use
- Destructive interference is reduced



TAGE (TAgged GEometric history length, 2006)

Observation: ***high-confidence*** predictions based on the ***longest history*** are the most reliable

- Series of predictors accumulating predictions for ***geometrically increasing history length***
- ***High-confidence*** prediction ***based on the longest history*** is used
- Considered one of the best available solutions

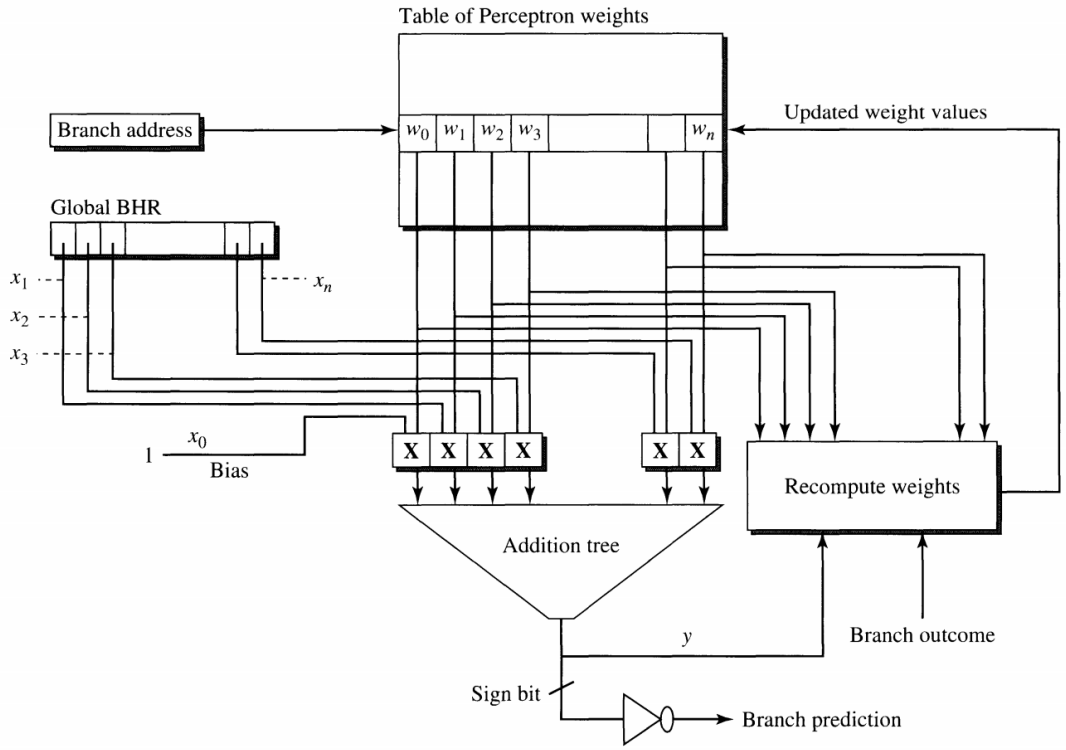


André Seznec. A New Case for the TAGE Branch Predictor. MICRO'11, 2011

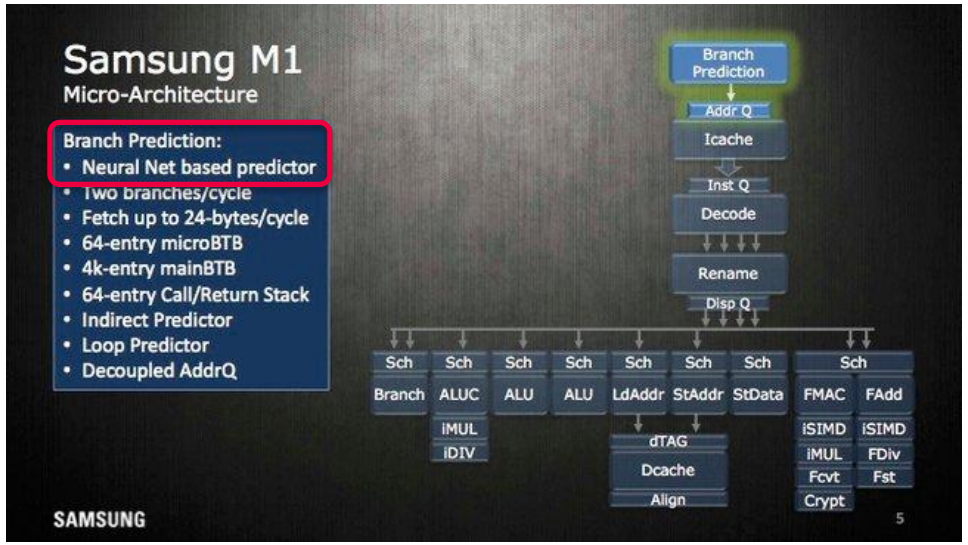
Neural predictor (2000s …)

Observation: various bits in branch history register have ***various importance (“weights”)*** in decision making

- Coefficient vector defines weights of bits in BHR
- Perceptron makes prediction based on weight vector and continuously updates it (***“learns”***) based on actual branch outcomes

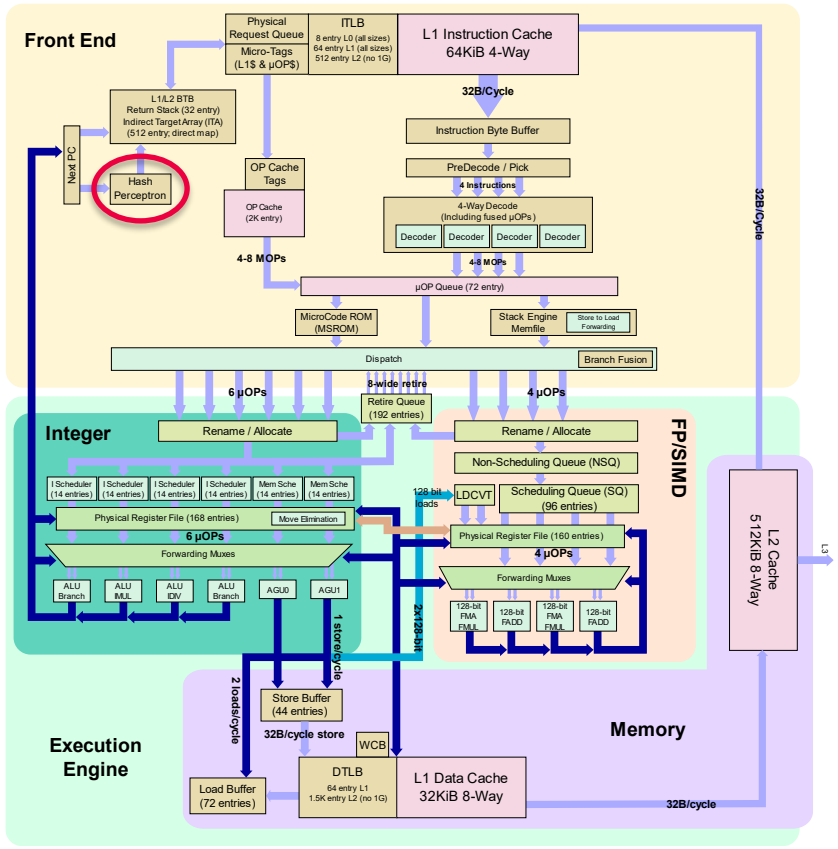


Neural predictors in modern CPU cores



Samsung Exynos

https://www.theregister.com/2016/08/22/samsung_m1_core/

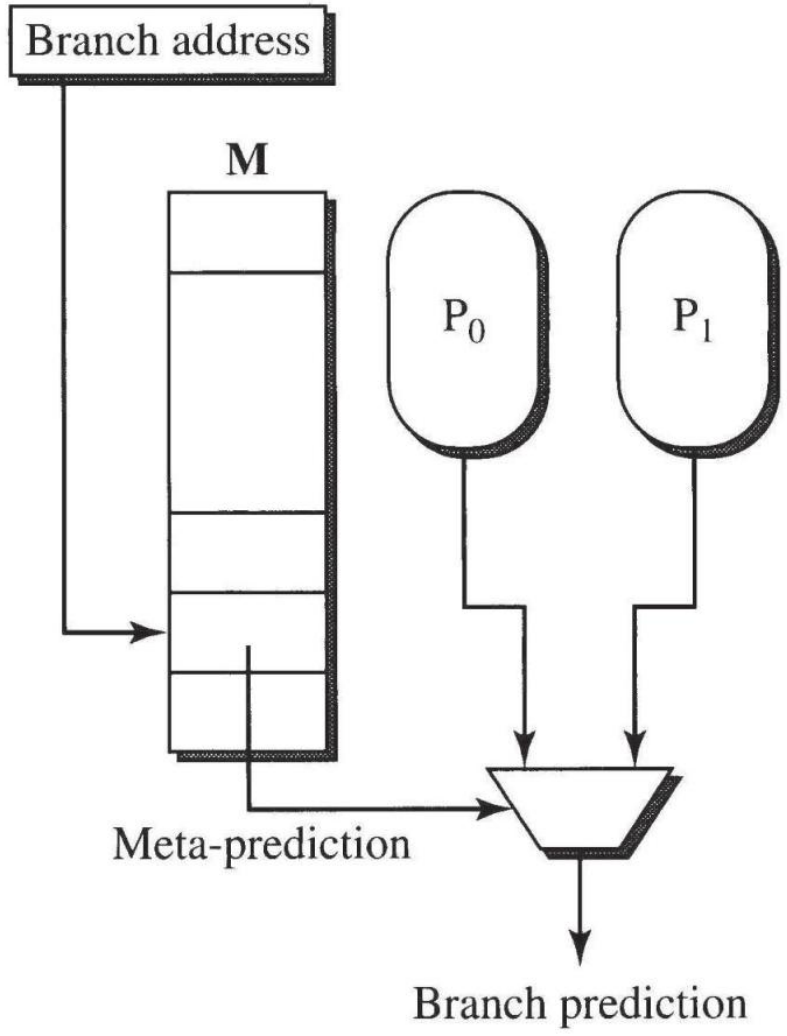


AMD Zen

https://en.wikichip.org/wiki/amd/microarchitectures/zen

Hybrid predictors

- several different predictors work in parallel
- effective predictor in current state is chosen based on ***meta-predictor***



Scott McFarling. Combining Branch Predictors. TN-36, Digital Equipment Corporation, Western Research Laboratory, June 1993

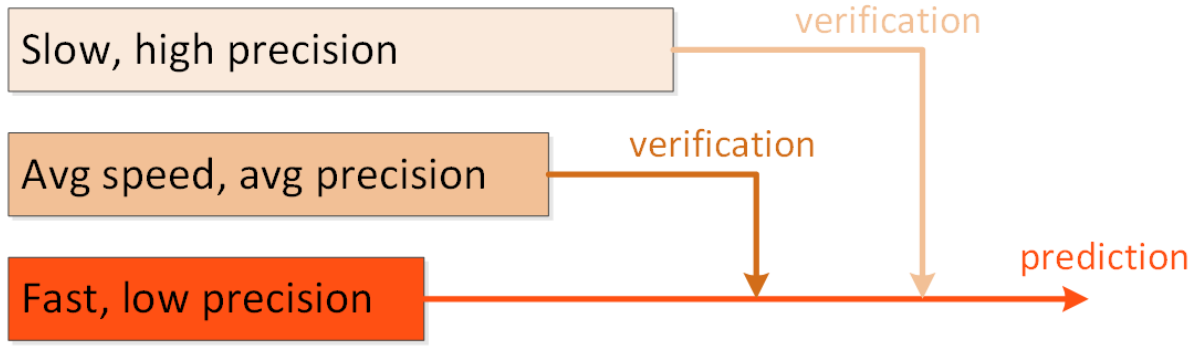
Hierarchical predictors

Observation: precise predictors typically have longer latencies

Solution: ***combine*** fast (but imprecise) and precise (but slow) predictors

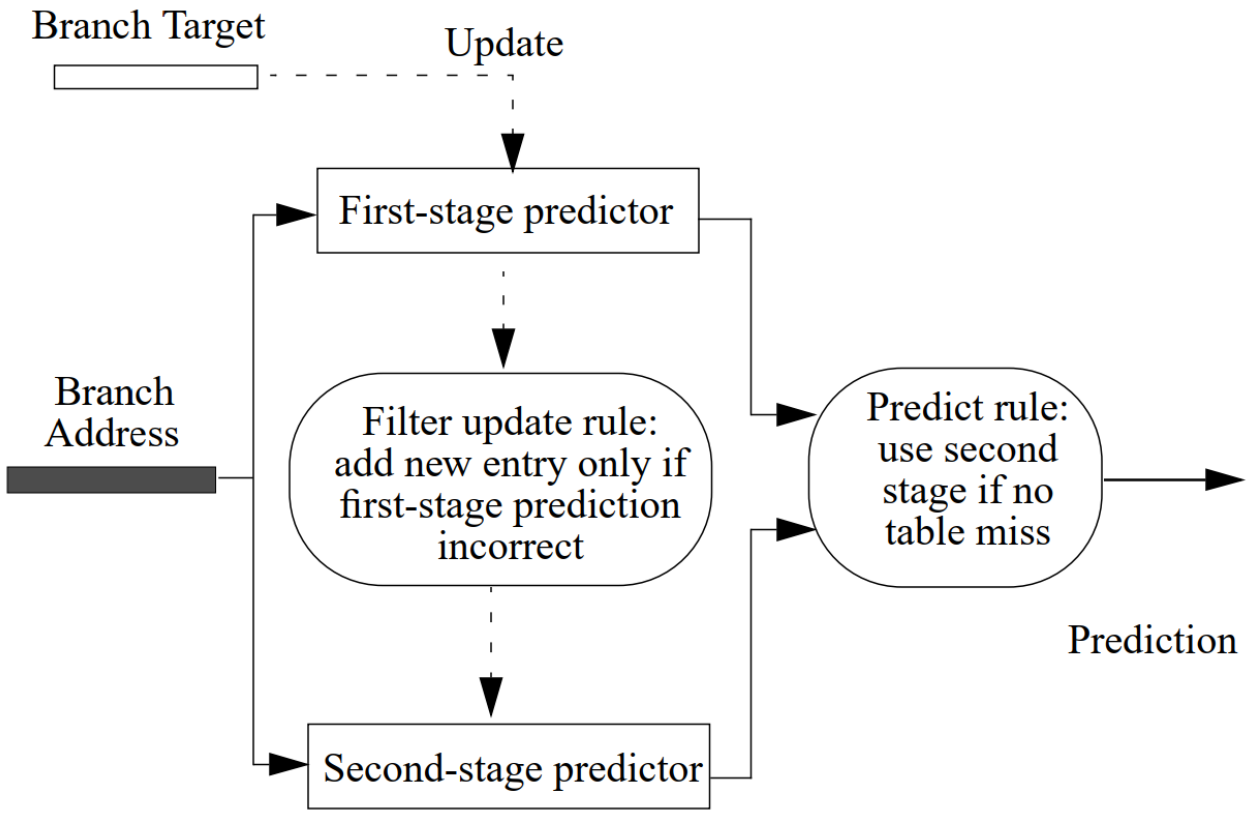
- Imprecise predictions output ***first***, precise ones – ***afterwards***
- If precise predictors contradict imprecise ones – ***flush***

  additional penalty is low (latency margin)



Cascaded predictors

- Combines simple and complex predictors
- Classifies branches according to their performance on simple predictor
- If simple predictor performs poorly – entry in complex predictor allocated
- ***Easily predicted branches are filtered – don't overload complex predictor***



K. Driesen and U. Hölzle, "The Cascaded Predictor: Economical and Adaptive Branch Target Prediction", MICRO-31, pp. 249-258, 1998.

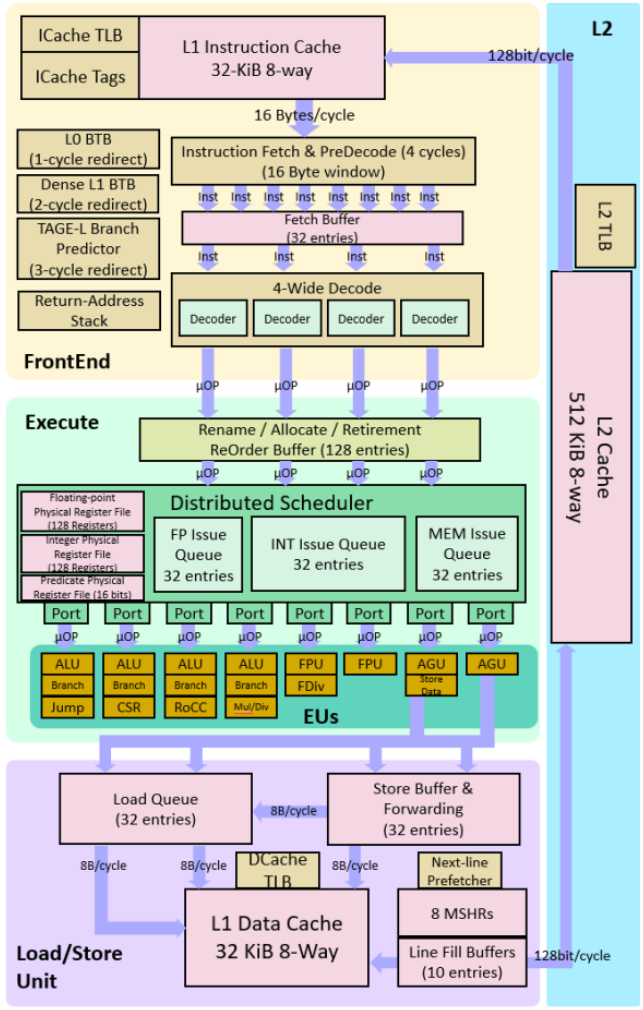
ITMO

Open-source superscalar OoO CPU cores

ITMO

SonicBOOM/Ocelot

SonicBOOM/Ocelot (Berkeley/Tenstorrent): design overview



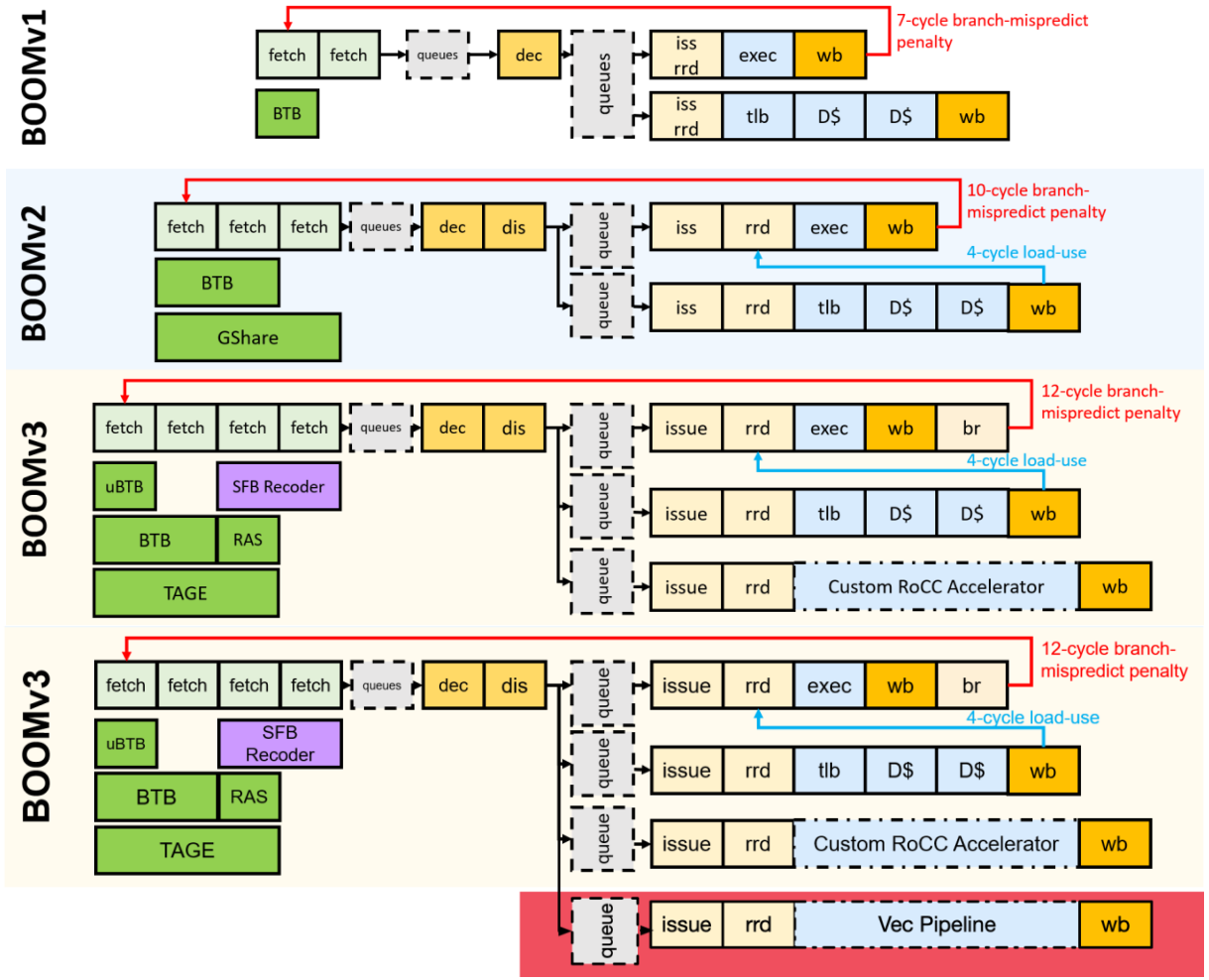
- Three main iterations implemented
- RISC-V architecture (RV64GC, Ocelot: RV64GC**V**)
- Boots Linux
- Written in Chisel high-level HDL
- Declared as the fastest open source core (x2, 2020), 6.2 CoreMark/MHz
- Demonstrates actual microarchitectural mechanisms
  superscalar, speculations, branch prediction, caching, etc.

Jerry Zhao et al. SonicBOOM: The 3rd Generation Berkeley Out-of-Order Machine. CARRV 2020
Repo: https://github.com/riscv-boom/riscv-boom, https://github.com/tenstorrent/riscv-ocelot
Documentation: https://docs.boom-core.org/en/latest/

BOOM v1-3 pipeline structures

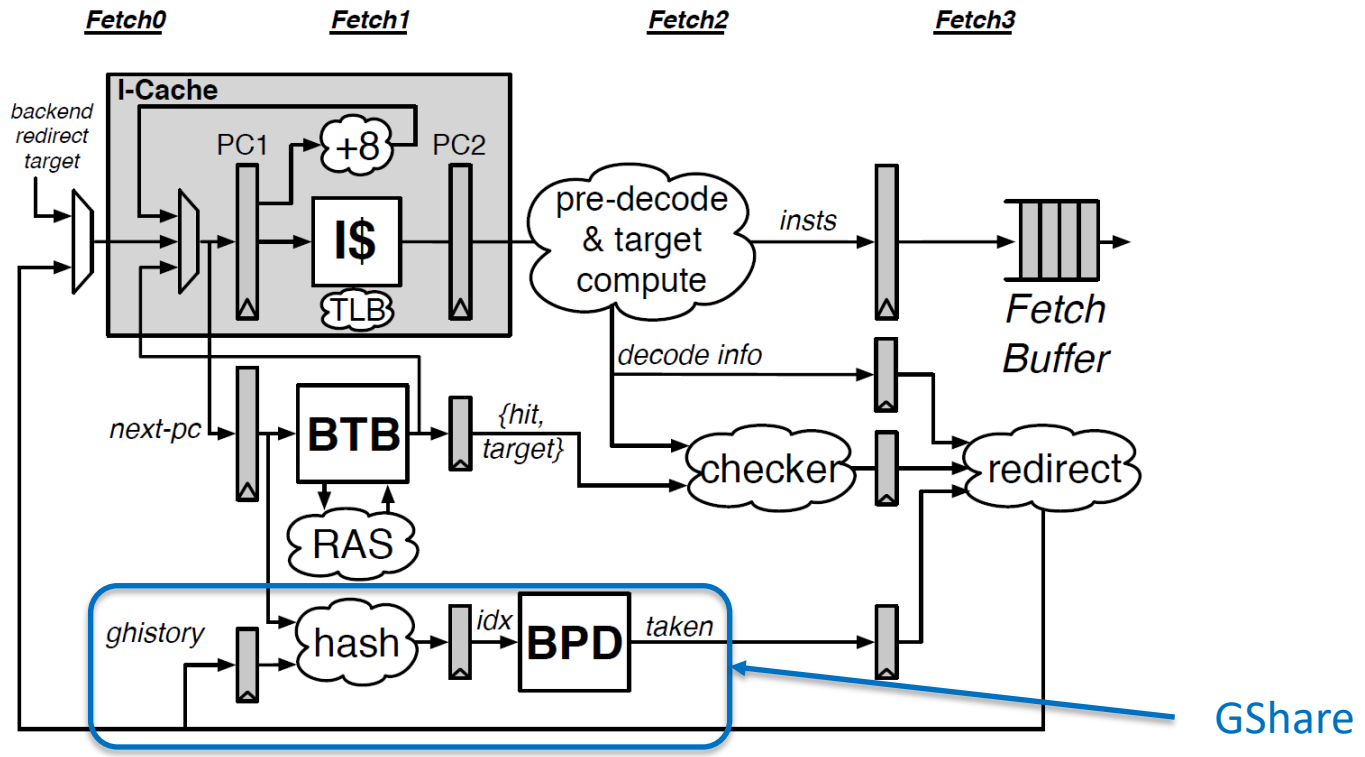
SonicBOOM

Ocelot



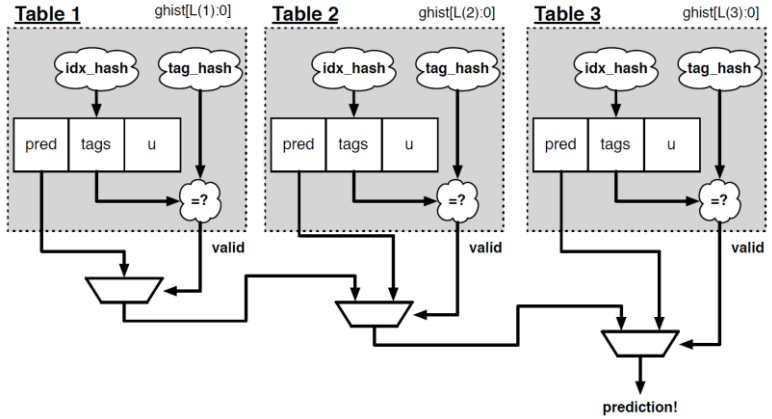
ITMO

Instruction fetch: overview (BP: L1 + L2)



Christopher Celio. A Highly Productive Implementation of an Out-of-Order Processor Generator. Technical Report No. UCB/EECS-2018-151. Berkeley, CA. December 2018.

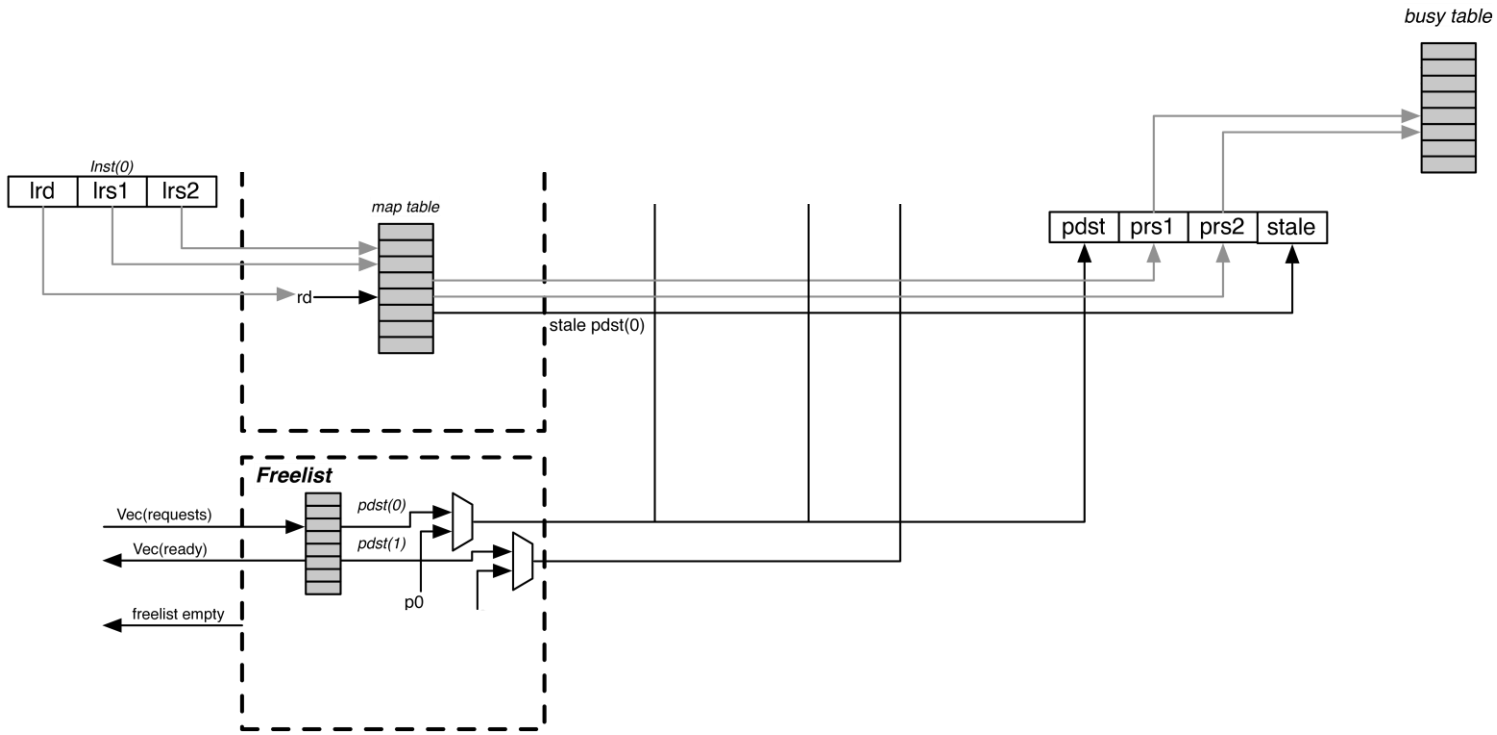
L3 branch predictor: TAGE




```scala
class TageTable(val nRows: Int, val tagSz: Int, val histLength: Int, val uBitPeriod: Int)
  (implicit p: Parameters) extends BoomModule()(p)
  with HasBoomFrontendParameters
{
  require(histLength <= globalHistoryLength)

  val nWrBypassEntries = 2
  val io = IO( new Bundle {
    val f1_req_valid = Input(Bool())
    val f1_req_pc    = Input(UInt(vaddrBitsExtended.W))
    val f1_req_ghist = Input(UInt(globalHistoryLength.W))

    val f3_resp = Output(Vec(bankWidth, Valid(new TageResp)))

    val update_mask    = Input(Vec(bankWidth, Bool()))
    val update_taken   = Input(Vec(bankWidth, Bool()))
    val update_alloc   = Input(Vec(bankWidth, Bool()))
    val update_old_ctr = Input(Vec(bankWidth, UInt(3.W)))

    val update_pc    = Input(UInt())
    val update_hist  = Input(UInt())

    val update_u_mask = Input(Vec(bankWidth, Bool()))
    val update_u = Input(Vec(bankWidth, UInt(2.W)))
  })

  def compute_folded_hist(hist: UInt, l: Int) = {
    val nChunks = (histLength + l - 1) / l
    val hist_chunks = (0 until nChunks) map {i =>
      hist(min((i+1)*l, histLength)-1, i*l)
    }
    hist_chunks.reduce(_^_)
  }

  def compute_tag_and_hash(unhashed_idx: UInt, hist: UInt) = {
    val idx_history = compute_folded_hist(hist, log2Ceil(nRows))
    val idx = (unhashed_idx ^ idx_history)(log2Ceil(nRows)-1,0)
```

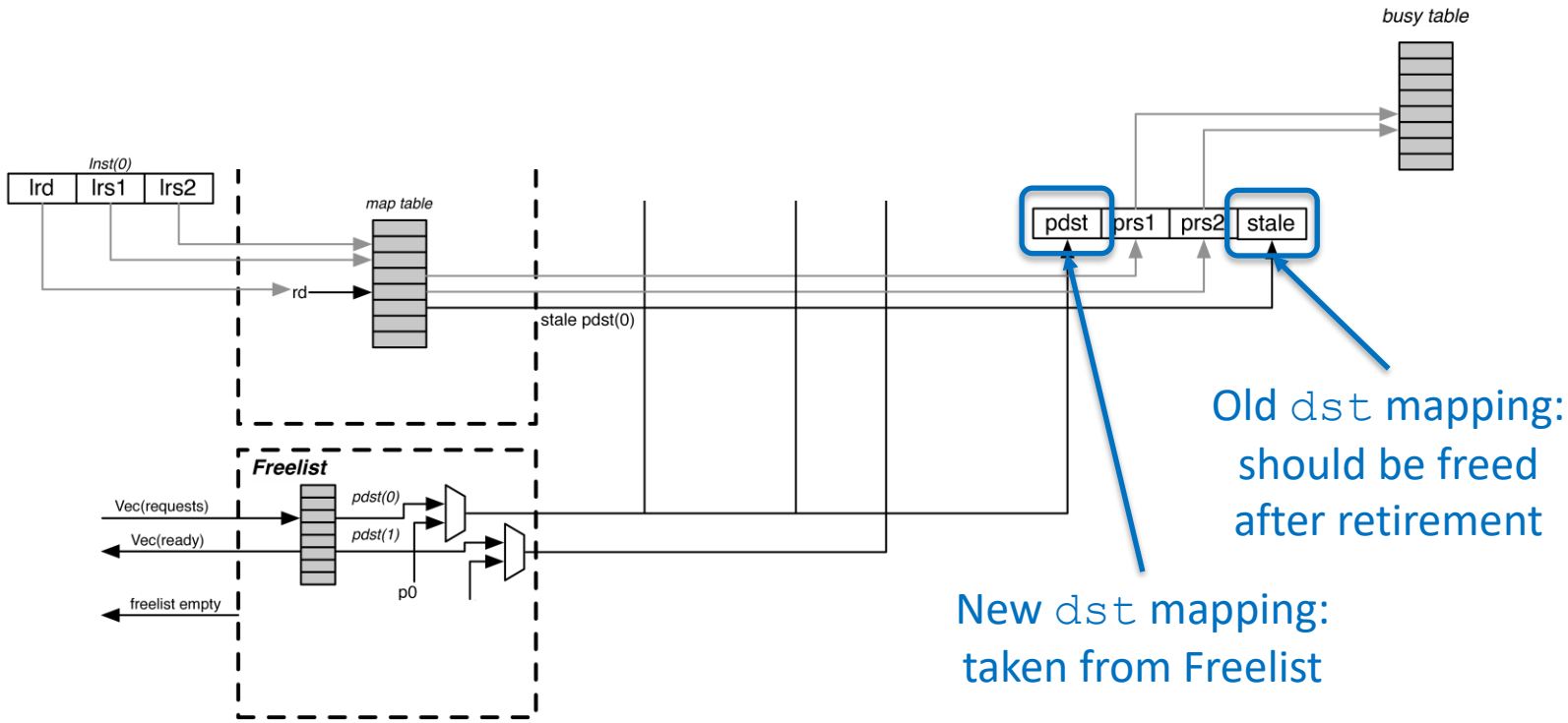
https://github.com/riscv-boom/riscv-boom/blob/master/src/main/scala/ifu/bpd/tage.scala

Register renaming (one channel)

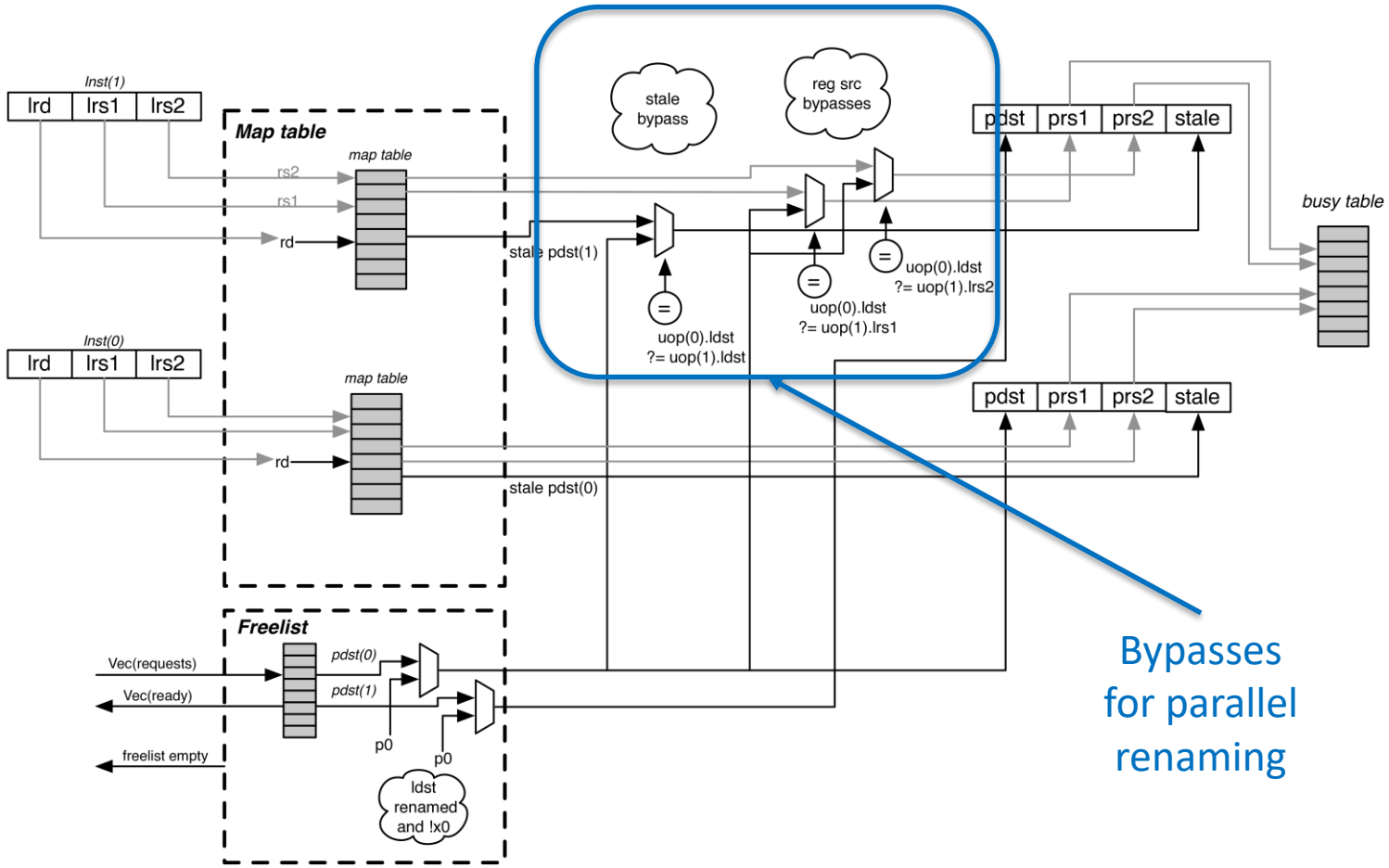


https://docs.boom-core.org/en/latest/

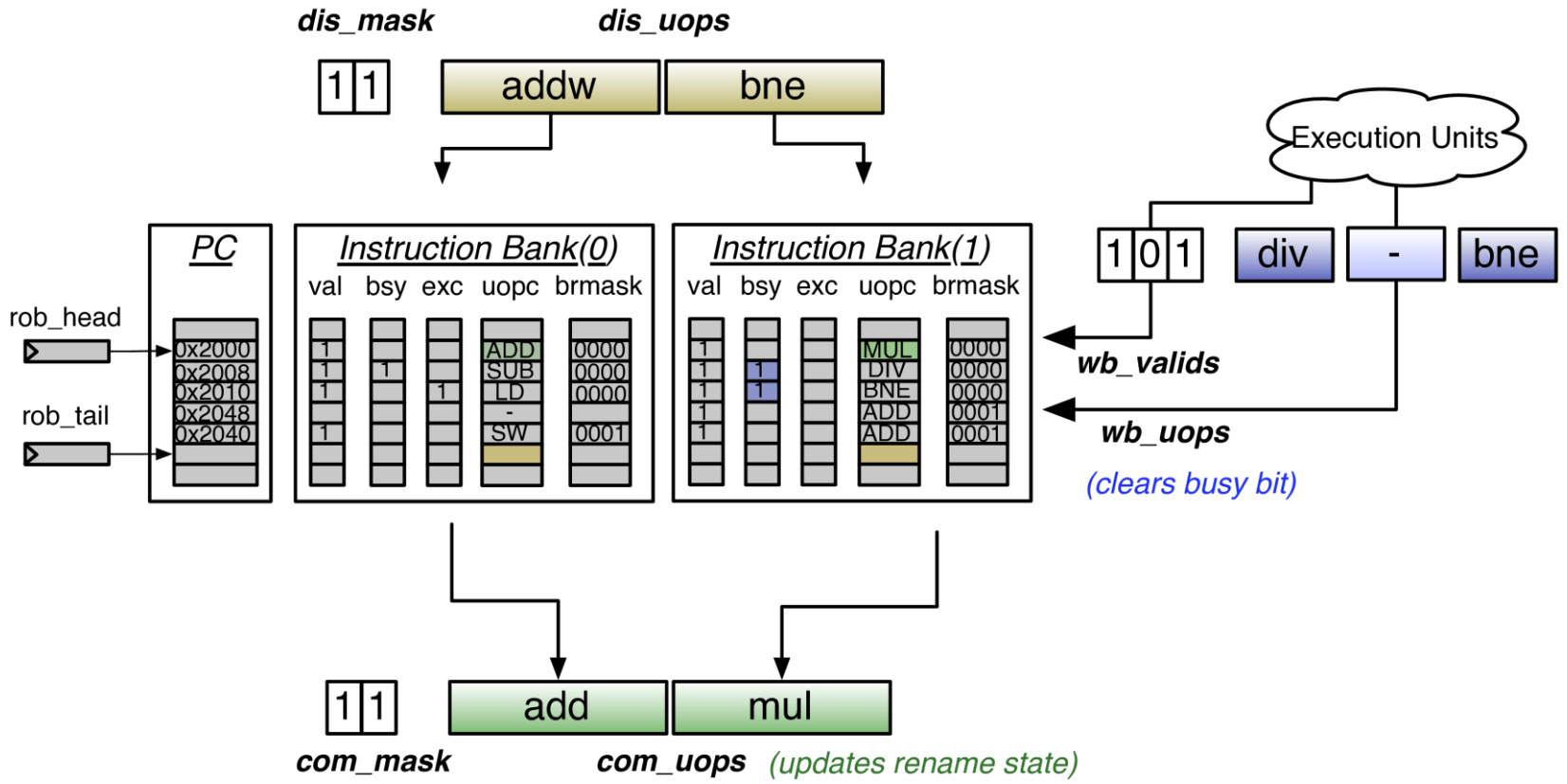
Register renaming (one channel)



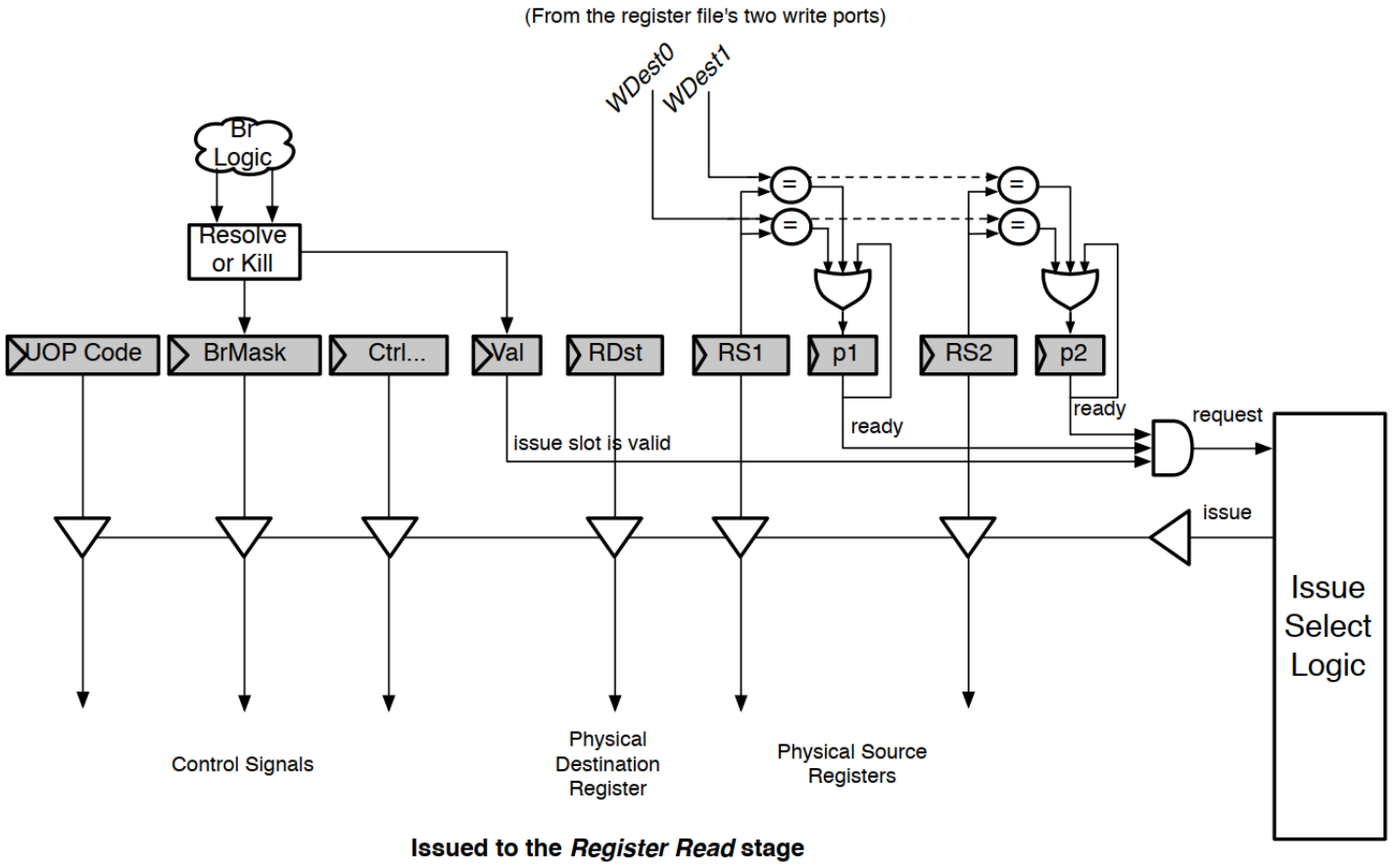
Register renaming (two parallel channels)



Dispatching and reorder buffer

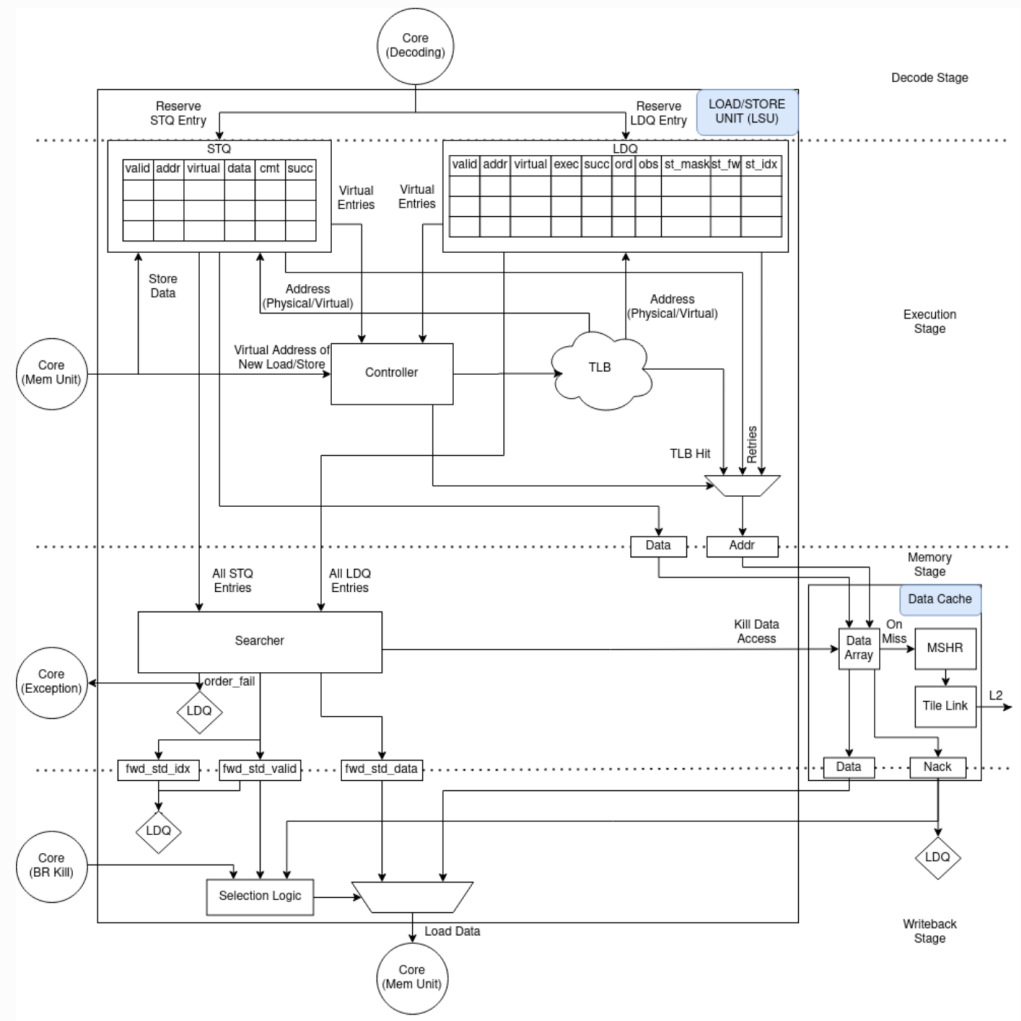


Issue Queue slot



Issued to the *Register Read* stage

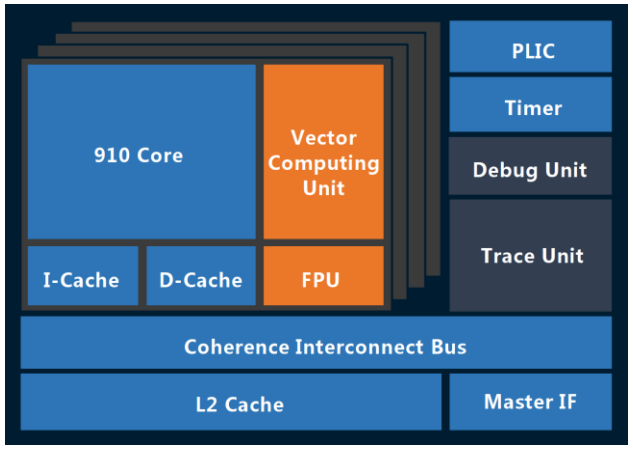
Load/Store Unit (LSU)

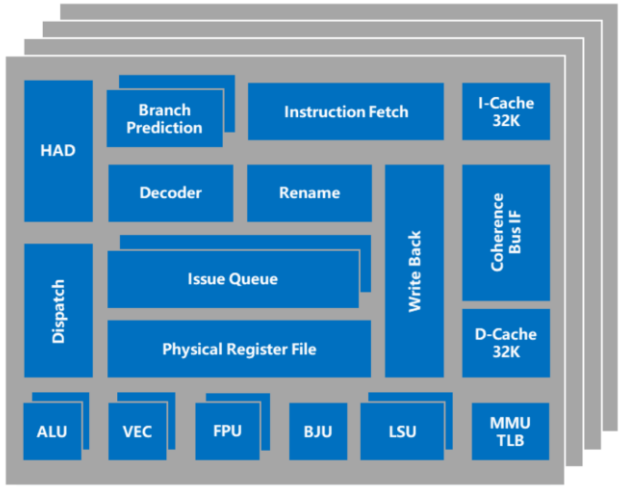


ITMO

OpenXuantie

OpenXuantie (Alibaba): design overview





- RISC-V (RV64GCV) architecture
- 4 CPU cores with various capabilities available:
  OpenE902, OpenE906, OpenC906, OpenC910
- 64-bit, 12-Stage, Out-of-Order
- Vector Processor Unit
- 50 Custom Instructions
- 32/64KB L1D/I Cache
- Each cluster has a shared, 8/16-way associated L2 cache, up to 8 MB
- Written in Verilog HDL
- Up to 2.5 GHz on TSMC 12 nm
- Fastest open core on release (7,1 CoreMark/MHz)

C. Chen et al., "Xuantie-910: A Commercial Multi-Core 12-Stage Pipeline Out-of-Order 64-bit High Performance RISC-V Processor with Vector Extension : Industrial Product," Proc. - Int. Symp. Comput. Archit., vol. 2020-May, pp. 52–64, 2020
Presentation: https://ieeexplore.ieee.org/document/9220630
Repo: https://github.com/T-head-Semi

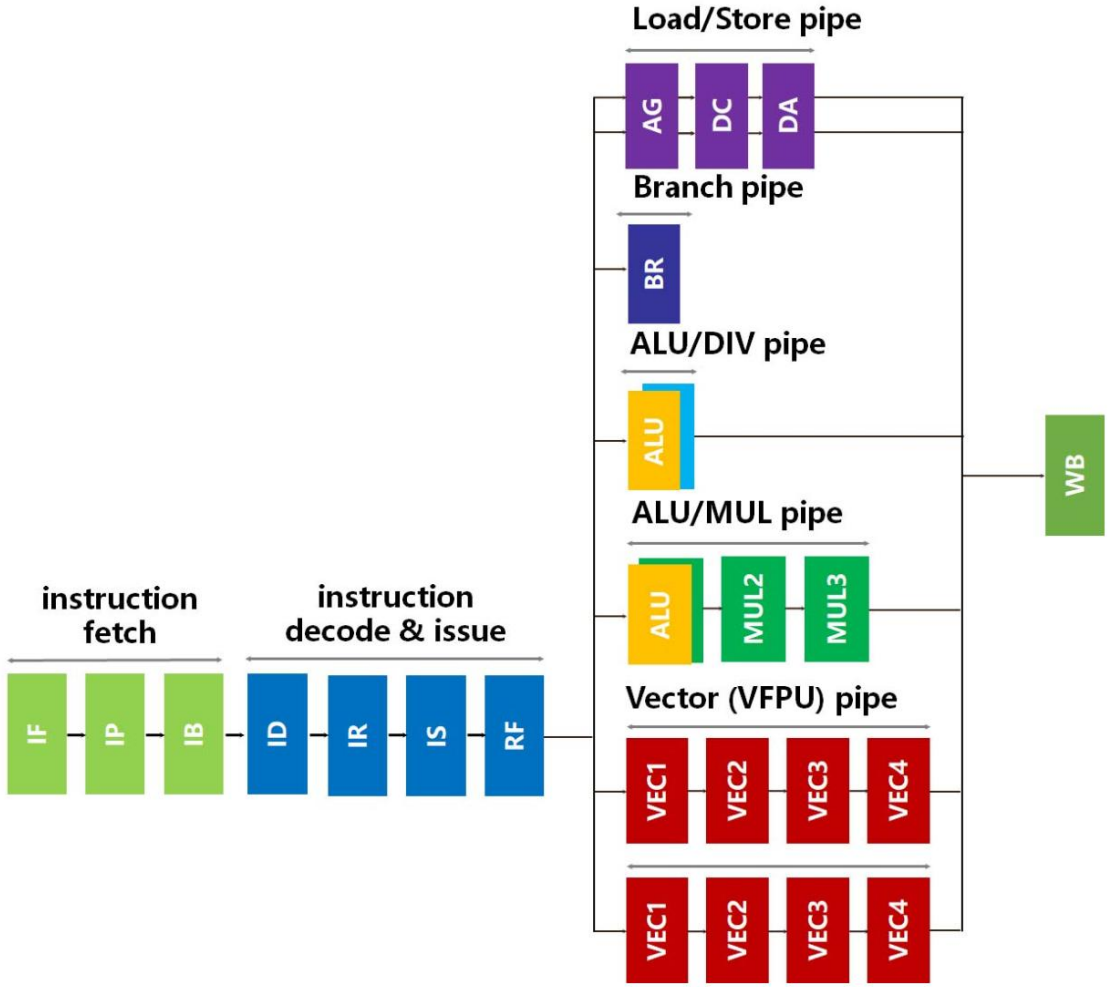
Pipeline structure

Frontend:

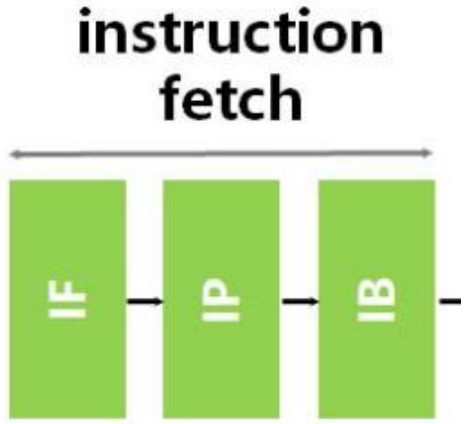
- Consists of 7 stages
- IFU uses a hybrid branch predictor
- IDU can decode 3 instructions simultaneously and can rename up to 4 instructions using physical registers
- OoO issue engine can issue up to 8 instructions

Backend:

- Multiple execution units (EUs):
  - 2x single-cycle ALUs
  - 1x single cycle branch jump unit
  - 1x dual-issue out-of-order load & store unit
  - 2x FPU and 2x VEC
  - 1x load/store unit (LSU)



Instruction Fetch Unit (IFU)



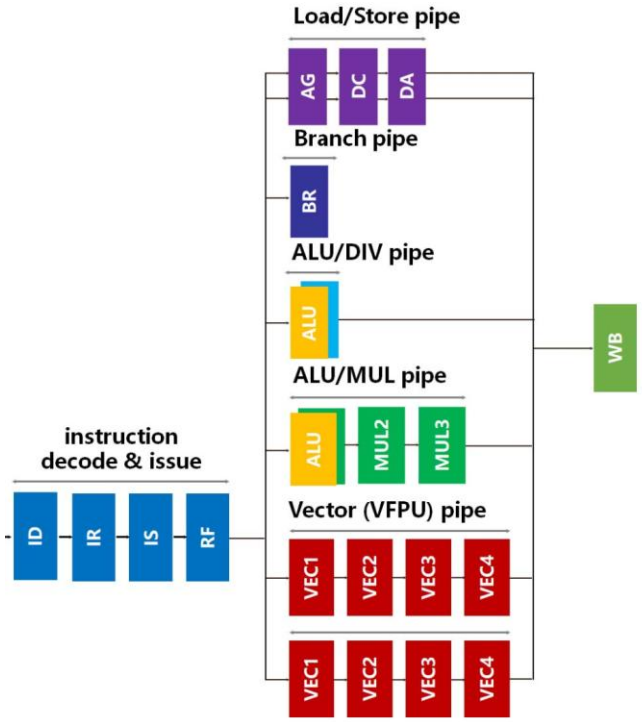
3 stages:

- Instruction Fetch (IF):
  - 128 bit fetch from the L1 cache in a single cycle
- Instruction Pack (IP):
  - pack and pre-process the instructions,
  - process the second-stage of branch jump
  - refill lines in case of cache miss
  - Up to 8 instructions can be packed
  - IDU can maximally decode 3 instructions, the throughput of IP will not become the performance bottleneck
- Instruction Buffer (IB):
  - the packed instruction is buffered
  - up to 3 instructions are sent to the ID pipeline stage
  - the third-stage of jump is processed.

Execution subsystem

Execution frontend – 4 stages:

- Instruction Decode (ID):
  - decomposes and decodes instructions
  - splits instruction into micro-instructions
  - zero-latency decoupling of scalar and vector instructions is supported
- Instruction Rename (IR):
  - renames operands in GPR, FGPR and VGPR using physical registers
  - eliminates the use of the costly move instruction
- Instruction Schedule (IS):
  - performs out-of-order instruction scheduling
  - The processor has 8 instruction slots
  - use an Age-Vector based scheduling algorithm
  - support dynamic load balancing by monitoring the workload in the pipeline
- Register File access (RF):
  - reads register file for operands



Execution backend:

- The EX stage (8 pipes)
- The RT stage:
  - takes care of instruction write-back and retirement
  - The ROB can hold up to 192 instructions

Memory subsystem: Load/Store Unit (LSU)

- LSU supports:
  - unaligned memory accesses
  - dual Issue OoO memory interface
  - multi-stream prefetching
- Can process one load instruction and one store instruction in parallel
- Requires dedicated load pipe and store pipe
- Each pipe contains 4 pipeline stages
  - address generation (AG)
  - data cache (DC)
  - data alignment (DA)
  - write-back (WB)
- Executions on the load and store pipes access their own RAMs
- Load Queue (LQ) and the Store Queue (SQ) keep the order of instructions

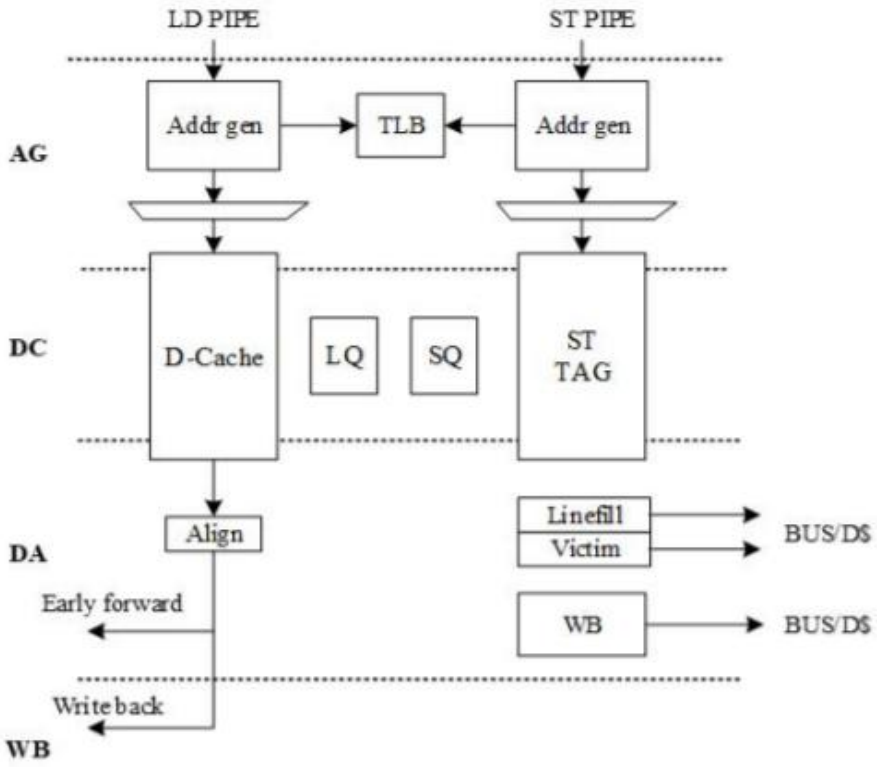
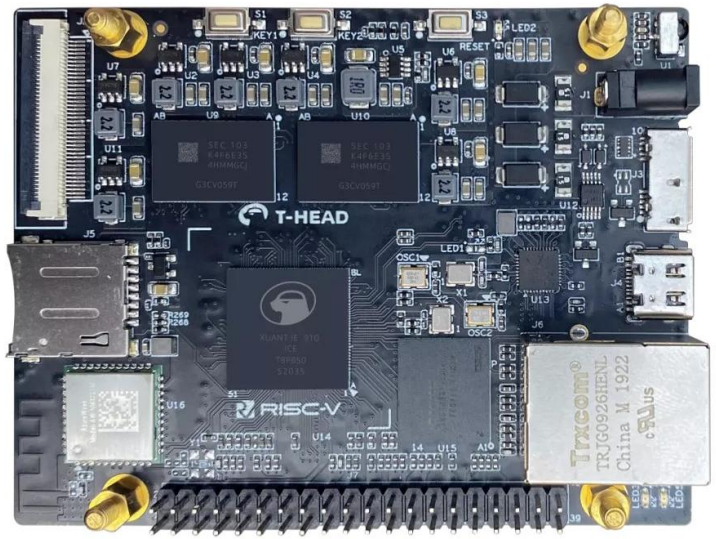


Fig. 9: The pipeline stages in the LSU.

Xuantie-based products

Lichee RV dev board
(Sipeed)
Allwinner D1 SoC
CPU core: XuanTie C906



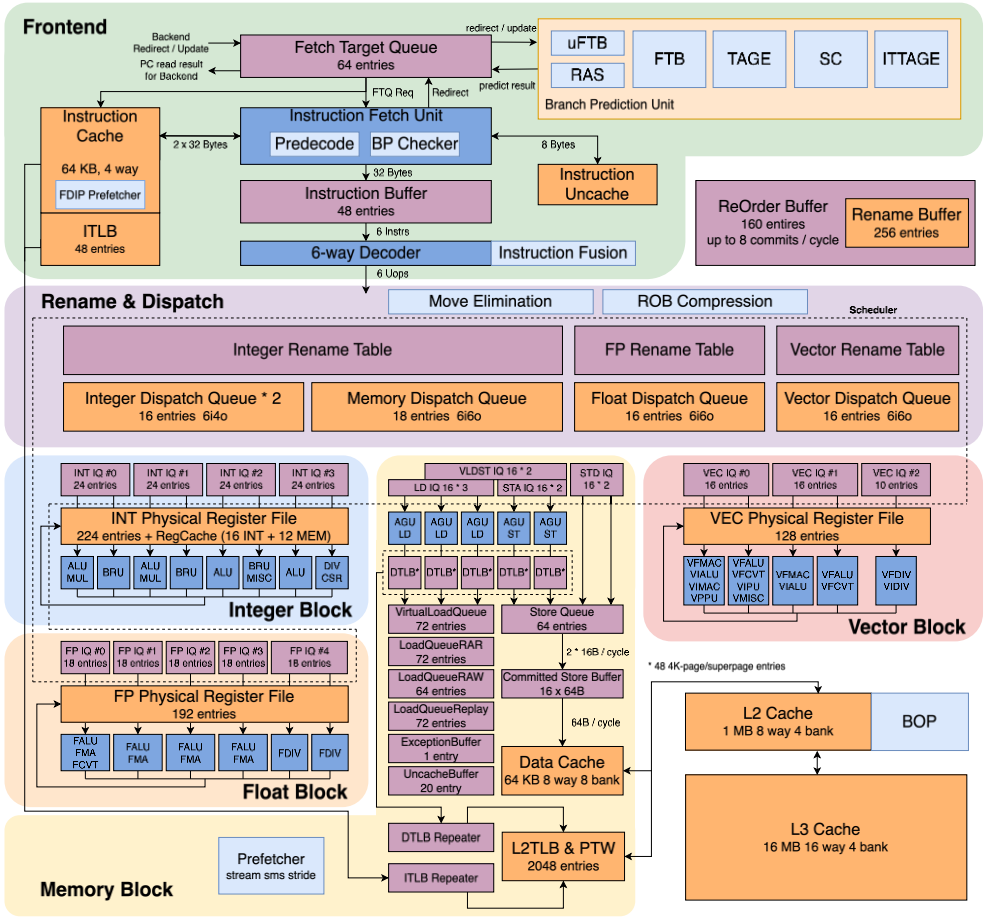
RVB-ICE dev board
(T-Head)
XuanTie C910 ICE SoC
CPU core: XuanTie C910

ROMA dev laptop
(DeepComputing, Xcalibyte)
Alibaba TH1520 SoC
CPU core: XuanTie C910

ITMO

XiangShan

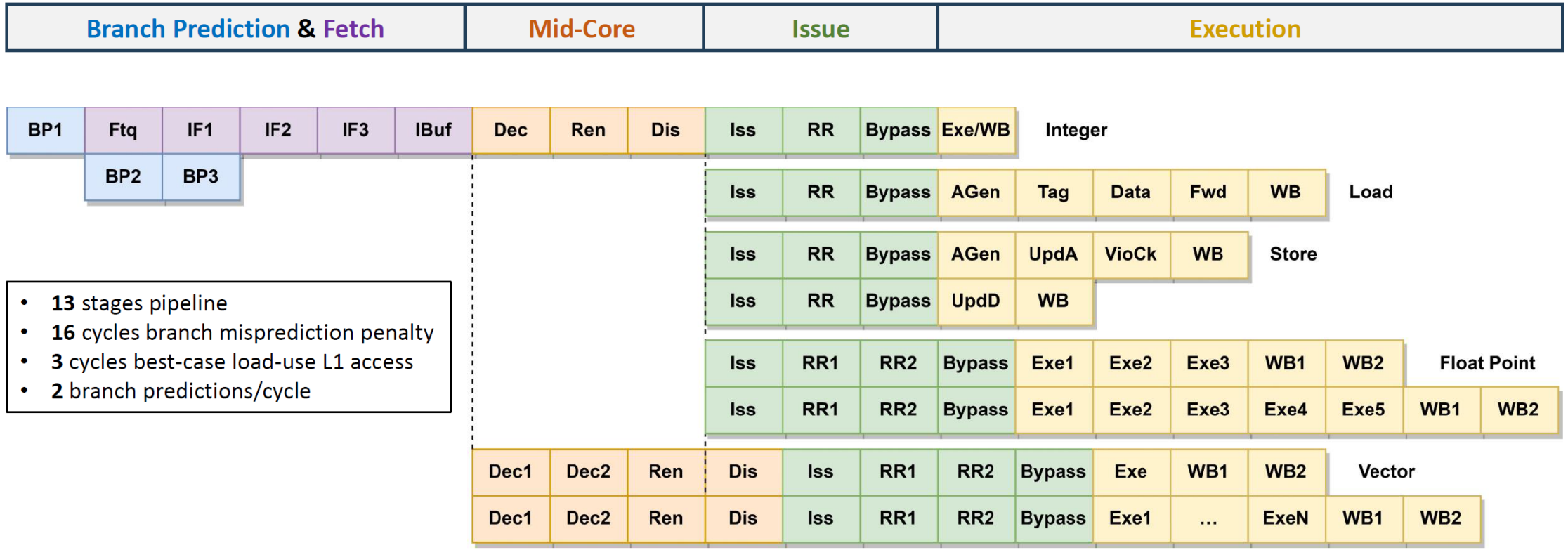
XiangShan (Chinese Academy of Science): design overview



- RISC-V (RV64GCBK) architecture
- Project contains 3x microarchitectures:
  Yanqihu (stable), Nanhu (stable), Kunminghu (in development)
- 13-stage 6-issue OoO pipeline
- Boots Debian Linux
- Award-winning project by RISC-V International
- >4100 stars and >550 forks on GitHub
- Written in Chisel HCL, uses components from BOOM
- Had several tape-outs:
  - Yanqihu: TSMC 28nm, 1.3GHz (2022)
  - Nanhu: SMIC(?) 14nm, 2GHz (2023)
- Fastest open core on release (Nanhu: 7,81 CoreMark/MHz)
- Published under Mulan Permissive Software Licence 2 (MulanPSL2)

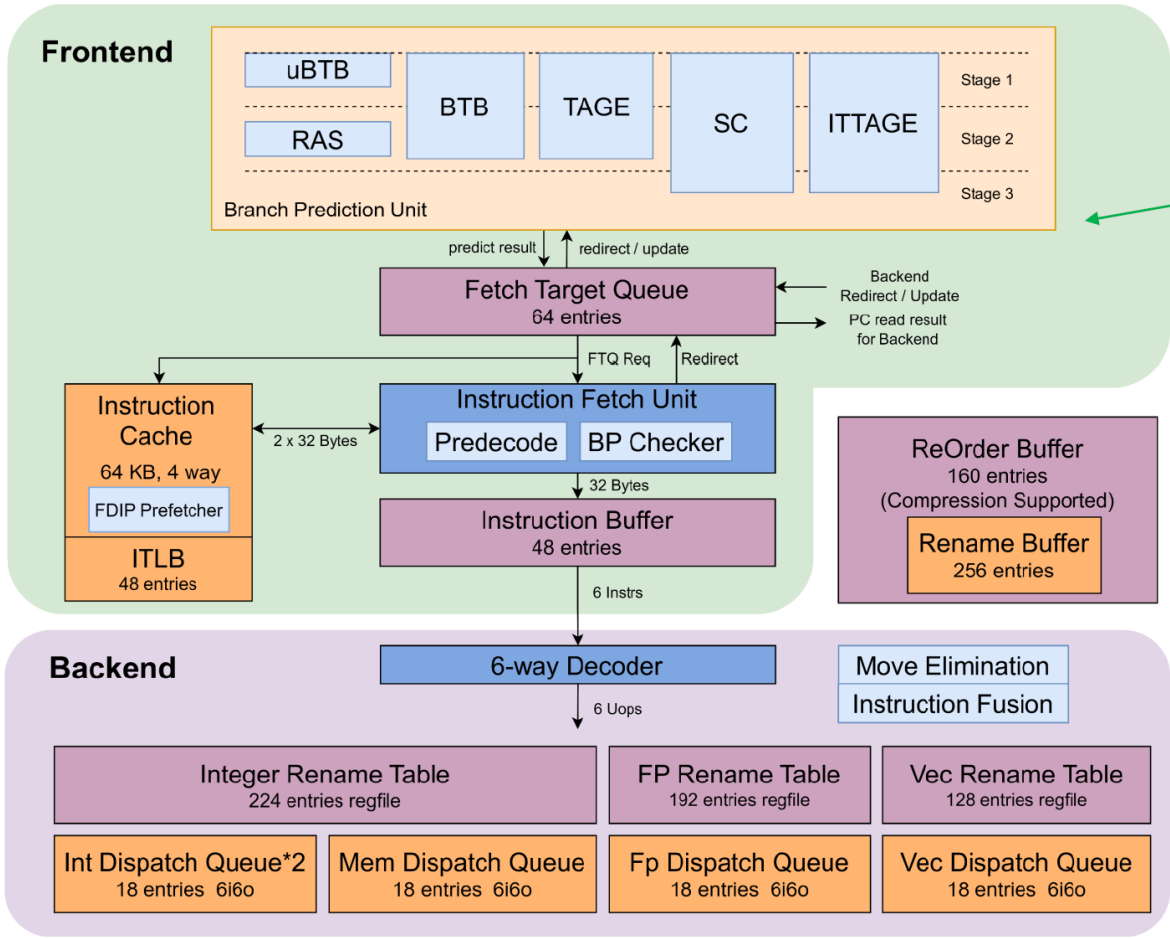
Documentation: https://xiangshan.cc
Repo: https://github.com/OpenXiangShan/XiangShan

Pipeline structure (Kunminghu)



Kaifan Wang et al. XiangShan: An Open-Source Project for High-Performance RISC-V Processors Meeting Industrial-Grade Standards. Hot Chips 2024.

Frontend

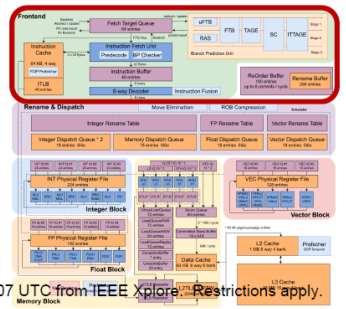


Multi-level branch predictors

- **256**-entry µBTB
- **2K**-entry BTB, optional L2 BTB
- **16K** TAGE-SC direction predictor
- **2K** ITTAGE indirection predictor
- **48**-entry return address stack

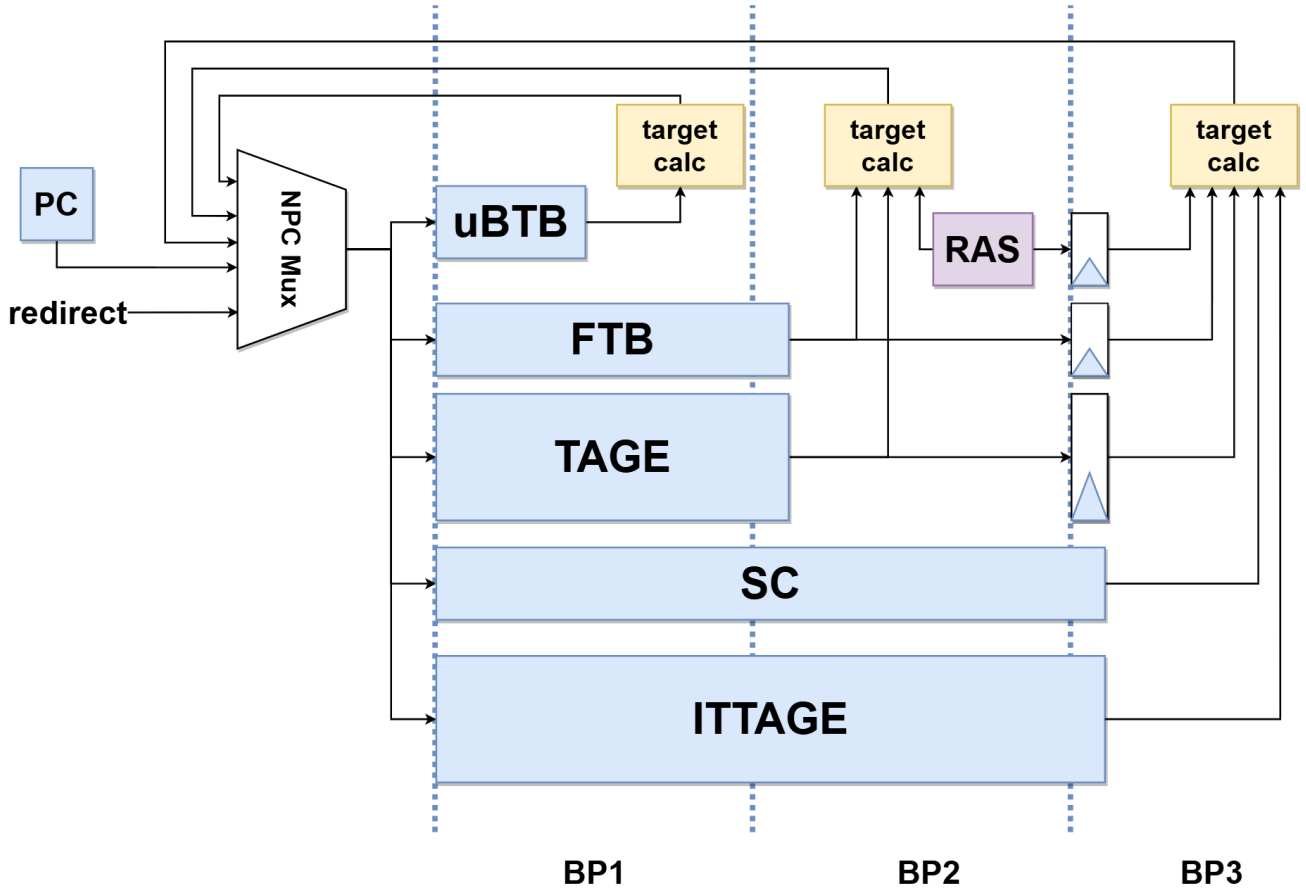
Instruction cache and TLB

- **64KB 4-way** ICache
- **48**-entry ITLB
- Fetch-directed instruction prefetcher

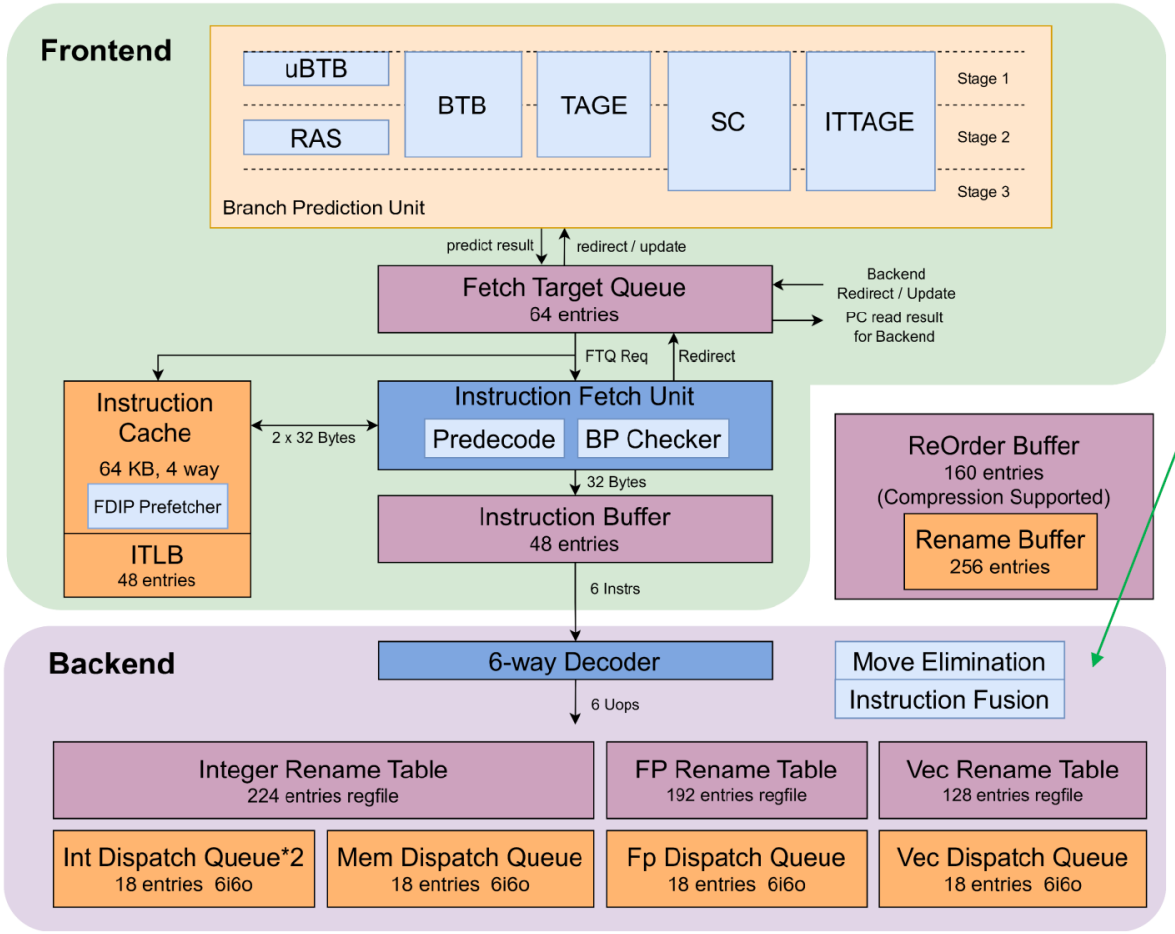


ITMO

Branch Prediction Unit



Backend



6-wide decode/rename/dispatch

Register rename

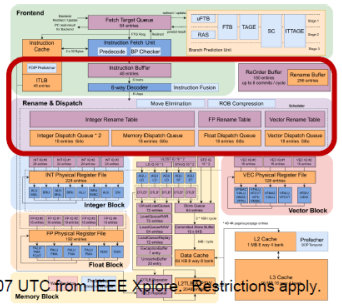
- **224**-entry Int regfile
- **192**-entry FP regfile
- **128**-entry Vec regfile
- Move elimination
- Instruction fusion

160 entry ROB

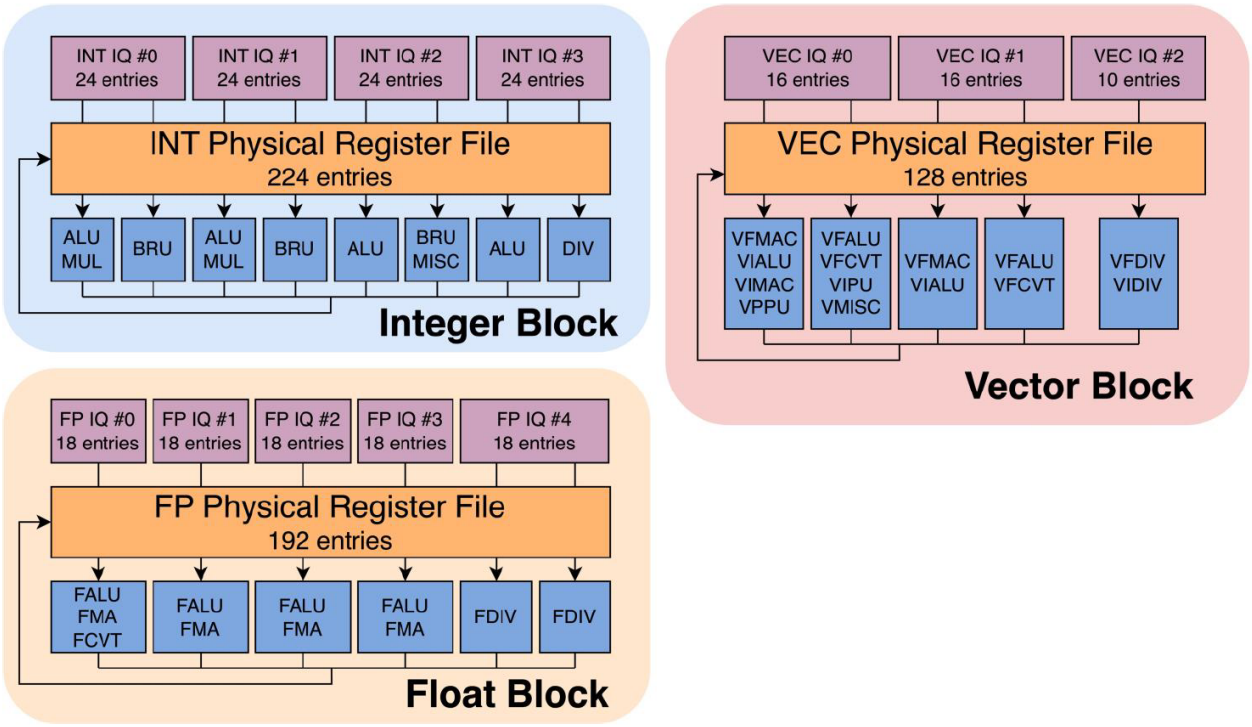
- Support compression (up to **6**-µop/entry)
- **8**-entry retire/cycle
- Recovery via checkpoint + rollback

256 entry Rename Buffer

- Bridge the gap between commit & rename table update



Execution units



Integer block

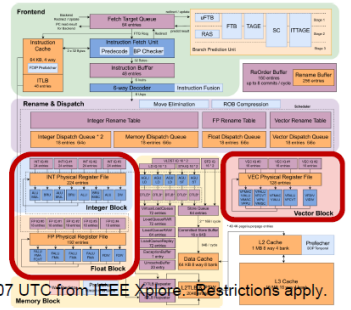
- **4** ALU (2 with 3-stage multiplier)
- **3** BRU for branch resolution
- **1** SRT radix-16 divider

Float-point block

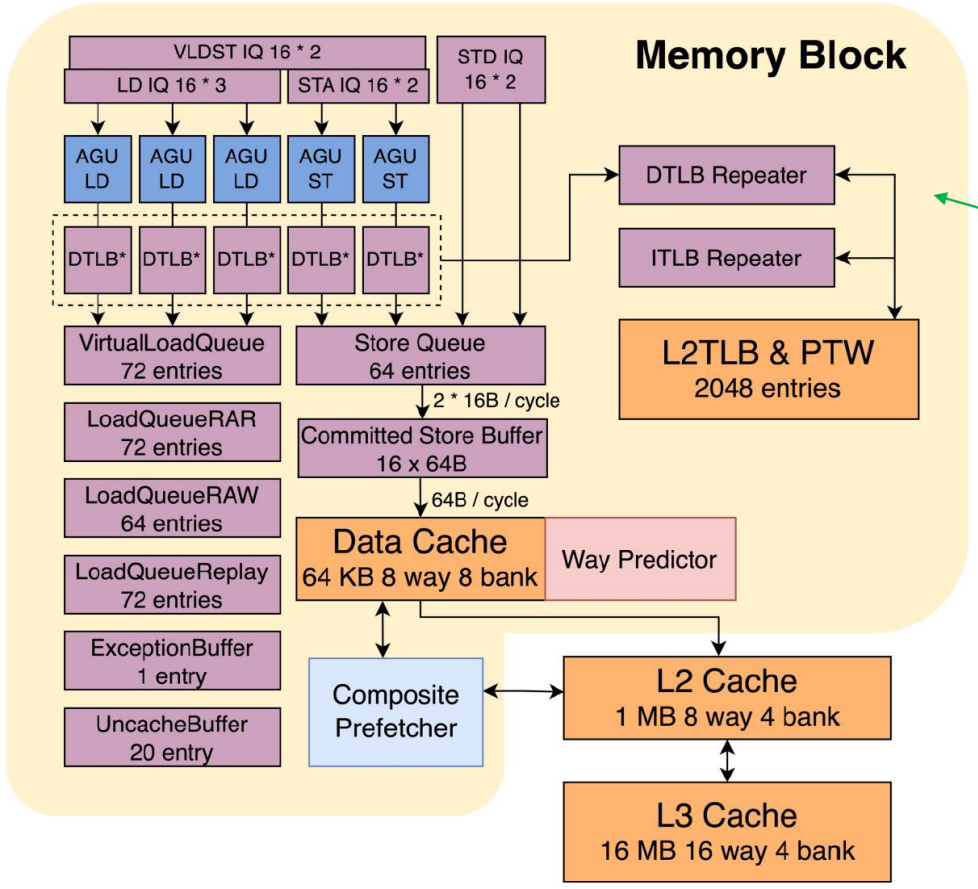
- **4** FPU + **2** FP Div

Vector block

- **4** VPU + **1** Vec Div
- **V1.0** SPEC (VLEN = **128**)



Load/store unit



Load/Store pipeline & queue

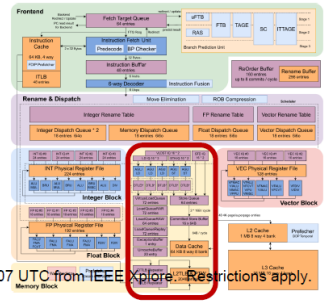
- **3** load pipes, **2** store pipes
- **72** inflight load, **64** inflight store

MMU

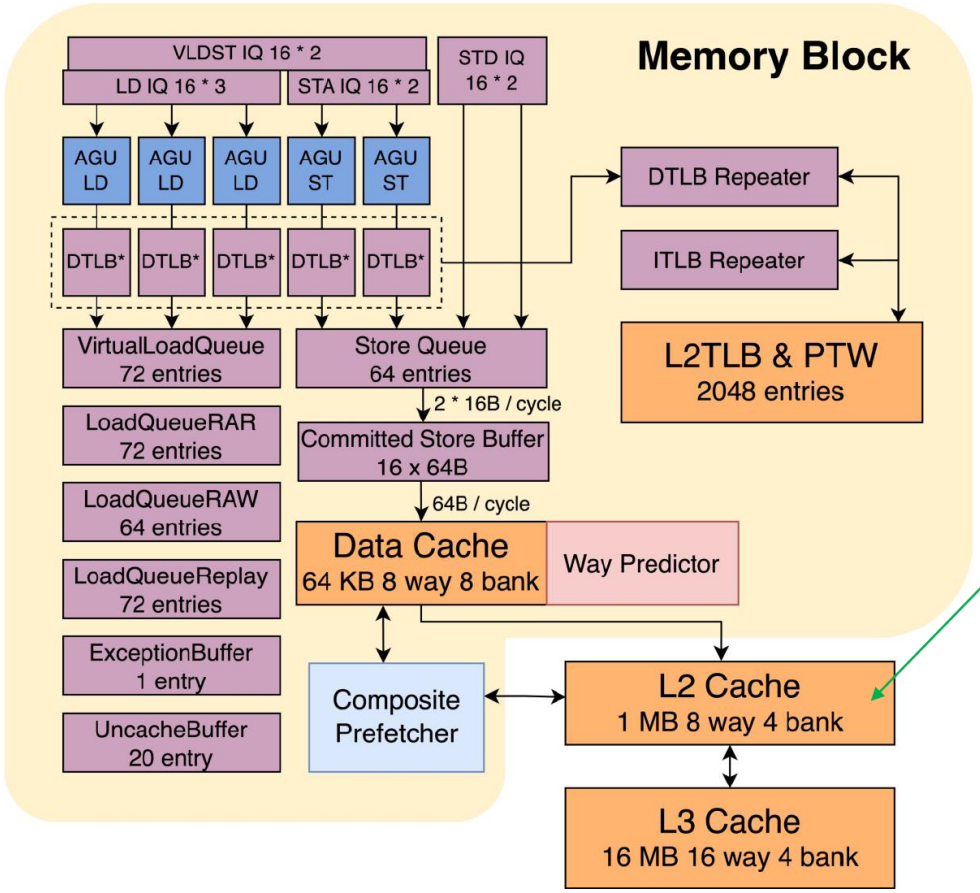
- **48b** virtual/**52b** physical addressing
- **48**-entry L1 DTLB, **2K**-entry L2 TLB
- Up to **7** outstanding page table walks
- Nested walk for virtualization

Data cache

- **64KB 8-way** VIPT (HW anti-aliasing)
- Way predictor for power efficiency
- Composite prefetcher
  - Stream & Stride prefetching
  - SMS & BOP prefetching
  - Temporal prefetching



Caches

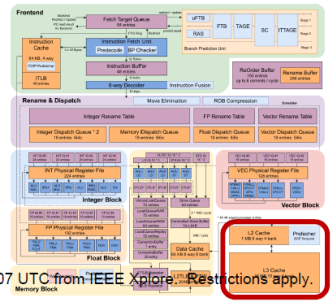


Private L2 cache

- **8-way**, up to **1MB** per core
- Inclusive to D$, non-inclusive to I$
- **64**+ OTs (configurable)
- **10-cycle** load-use latency
- PLRU/RRIP replacement

Shared L3 cache

- **16-way**, up to **16MB**
- Inclusive/Non-inclusive to L2$
- **150**+ OTs (configurable)
- **~30-cycle** load-use latency
- PLRU/RRIP replacement



ITMO

Other open-source OoO CPU projects

- riscy-OOO (RISC-V, MIT, in Bluespec HDL)
  paper: https://people.csail.mit.edu/szzhang/paper/micro2018.pdf
  repo: https://github.com/csail-csg/riscy-OOO
- SweRV EH1 (RISC-V, limited OoO, Western Digital, in SystemVerilog HDL)
  repo: https://github.com/chipsalliance/Cores-SweRV
- RSD (RISC-V, Japan, in SystemVerilog HDL)
  repo: https://github.com/rsd-devel/rsd

Thank you for the lesson!

Alexander Antonov, Assoc. Prof., antonov@itmo.ru

Hangzhou, 2025