中国科学院计算技术研究所
INSTITUTE OF COMPUTING TECHNOLOGY, CHINESE ACADEMY OF SCIENCES

中国开放指令生态（RISC-V）联盟
China RISC-V Alliance

北京开源芯片研究院
BEIJING INSTITUTE OF OPEN SOURCE CHIP

XiangShan: An Open-Source High-Performance RISC-V Processor and Infrastructure for Architecture Research

The XiangShan Team

Institute of Computing Technology (ICT)

Chinese Academy of Sciences (CAS)

ASPLOS'23@Vancouver, Canada

March 25, 2023

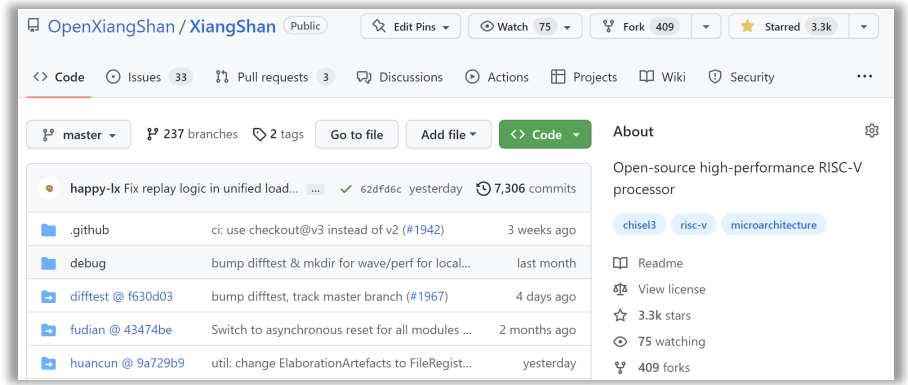
Schedule

Time	Topic
9:00-9:25	Introduction of the XiangShan Project
9:25-9:35	Tutorial Overview and Highlights
9:35-10:20	***Microarchitecture Design and Implementation***
	Coffee Break
10:40-12:00	Hands-on Development

XiangShan: Open-Source High-Performance Processor

- **1st generation: YANQIHU**
  - 2020/6: first commit of design RTL
  - 2021/7: **28nm tape-out, 1.3GHz**
  - Performance: **SPEC CPU2006 7.01@1GHz, DDR4-1600**
- **2nd generation: NANHU**
  - 2021/5: starting design exploration and RTL design
  - 2022/12: RTL Freeze
  - Plan: **14-nm tape-out soon, estimated SPEC CPU2006 20@2GHz**
- **3rd generation: KUNMINGHU**
  - ISA feature: **Vector (V) extension, Hypervisor (H) extension**
  - Comprehensive performance improvement

Fragrant Hills in Beijing



>3.2K stars, >400 forks on GitHub

- Open-sourced at https://github.com/OpenXiangShan/XiangShan

Yanqihu: 1st generation of XiangShan

Yanqi Lake in Beijing

- **Yanqihu**: named after a lake in Beijing, China
  - RV64GC, 11-stage, superscalar, out-of-order
  - 5.3 CoreMark/MHz (gcc-9.3.0 –O2)
  - Real chip: SPEC CPU2006 7@1GHz with DDR4-1600 (DDR not fully optimized)
- **Tape-out**: single XiangShan core (commit hash ccbca07) with 1MB L2 Cache

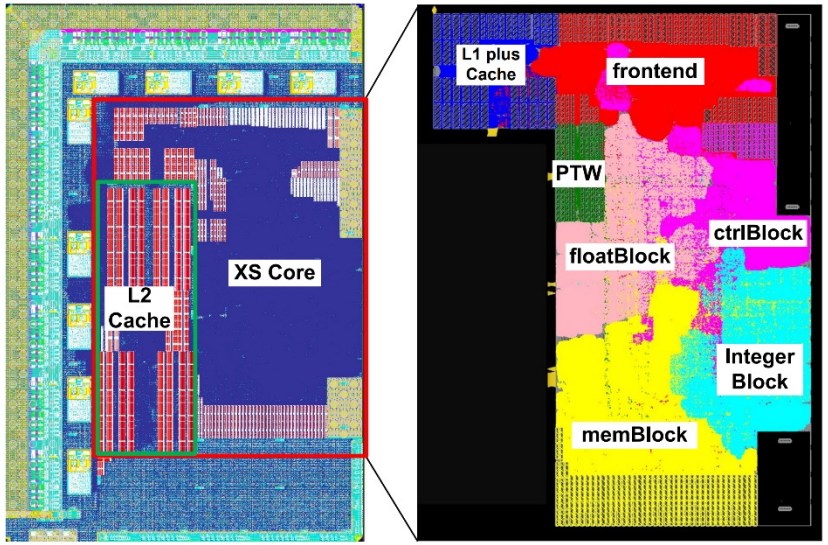
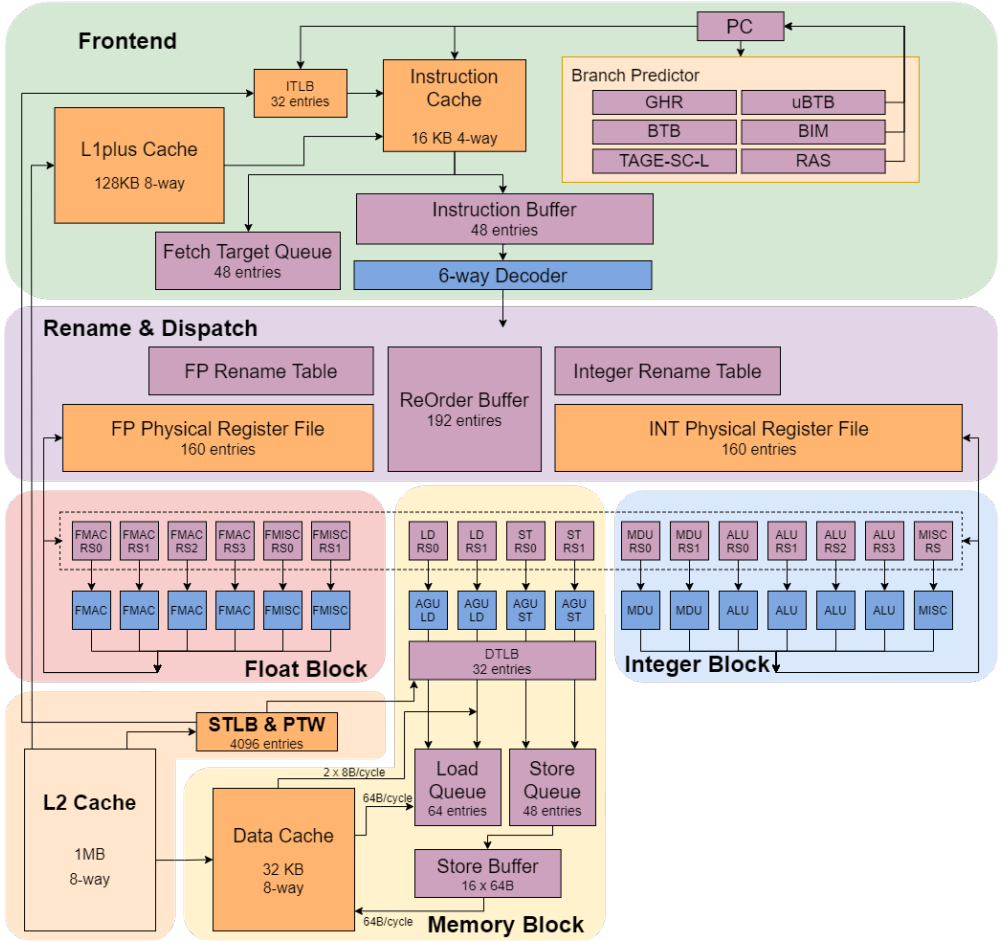


Figure. Layout of (a) the entire chip; (b) the core

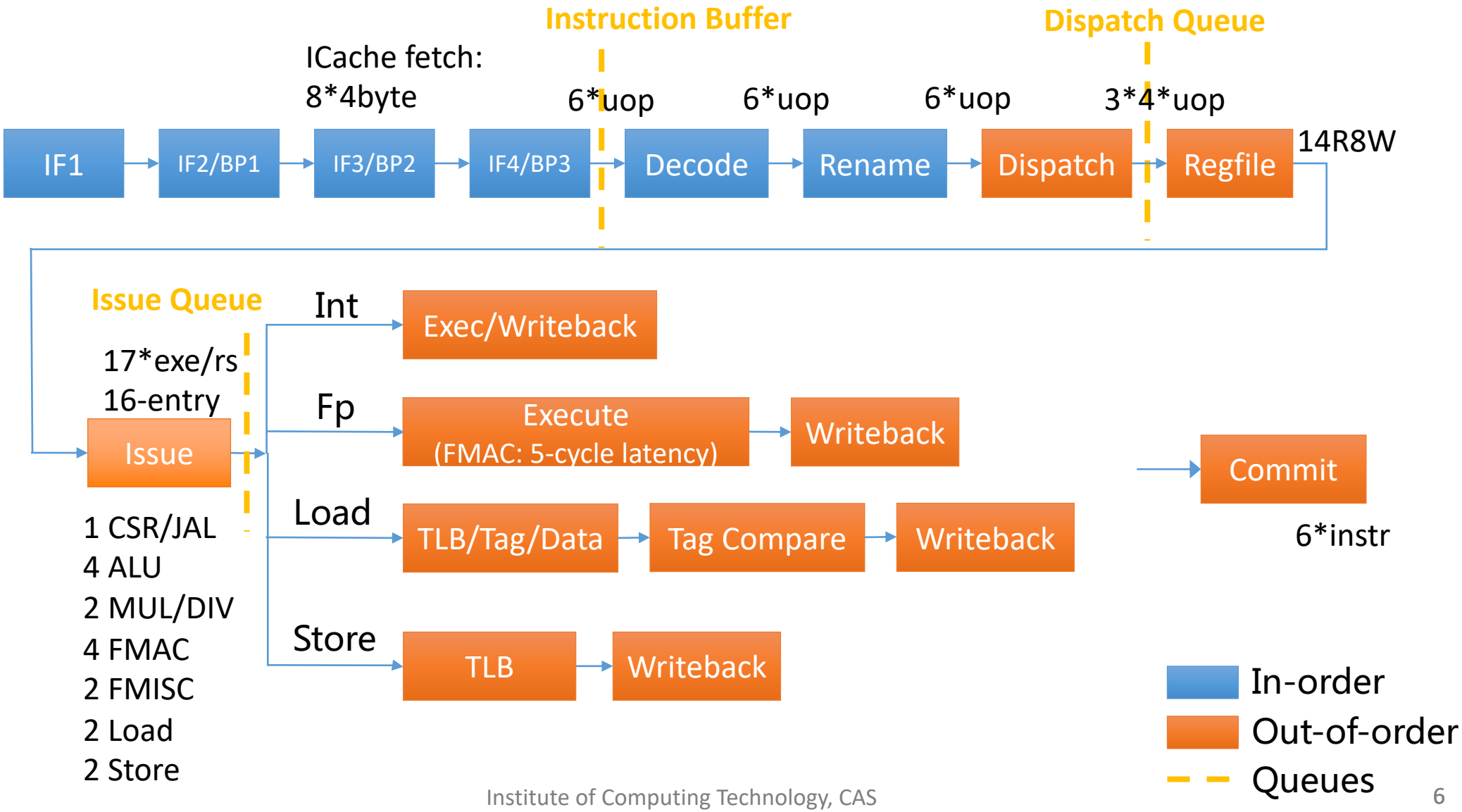
Tape-out information for the processor core	
Process Node	28nm
Die Size	8.6 mm²
Std Cell	5.05M, 4.27 mm²
Mem	261, 1.7mm²
Density	66%
Cell	ULVT 1.04%, LVT 19.32%, SVT 25.19%, HVT 53.67%
Estimated Power	5W
Frequency	1.3GHz, TT85C

XiangShan Microarchitecture (Yanqihu)

- **11**-stage, **6**-wide decode/rename
- **TAGE-SC-L** branch prediction
- **160** Int PRF + **160** FP PRF
- **192**-entry ROB, **64**-entry LQ, **48**-entry SQ
- **16**-entry RS for each FU
- **16KB** L1 Cache, **128KB** L1plus Cache for instruction
- **32KB** L1 Data Cache
- **32**-entry ITLB/DTLB, **4K**-entry STLB
- **1MB** inclusive L2 Cache



XiangShan Microarchitecture (Yanqihu)



NANHU: 2nd generation microarchitecture

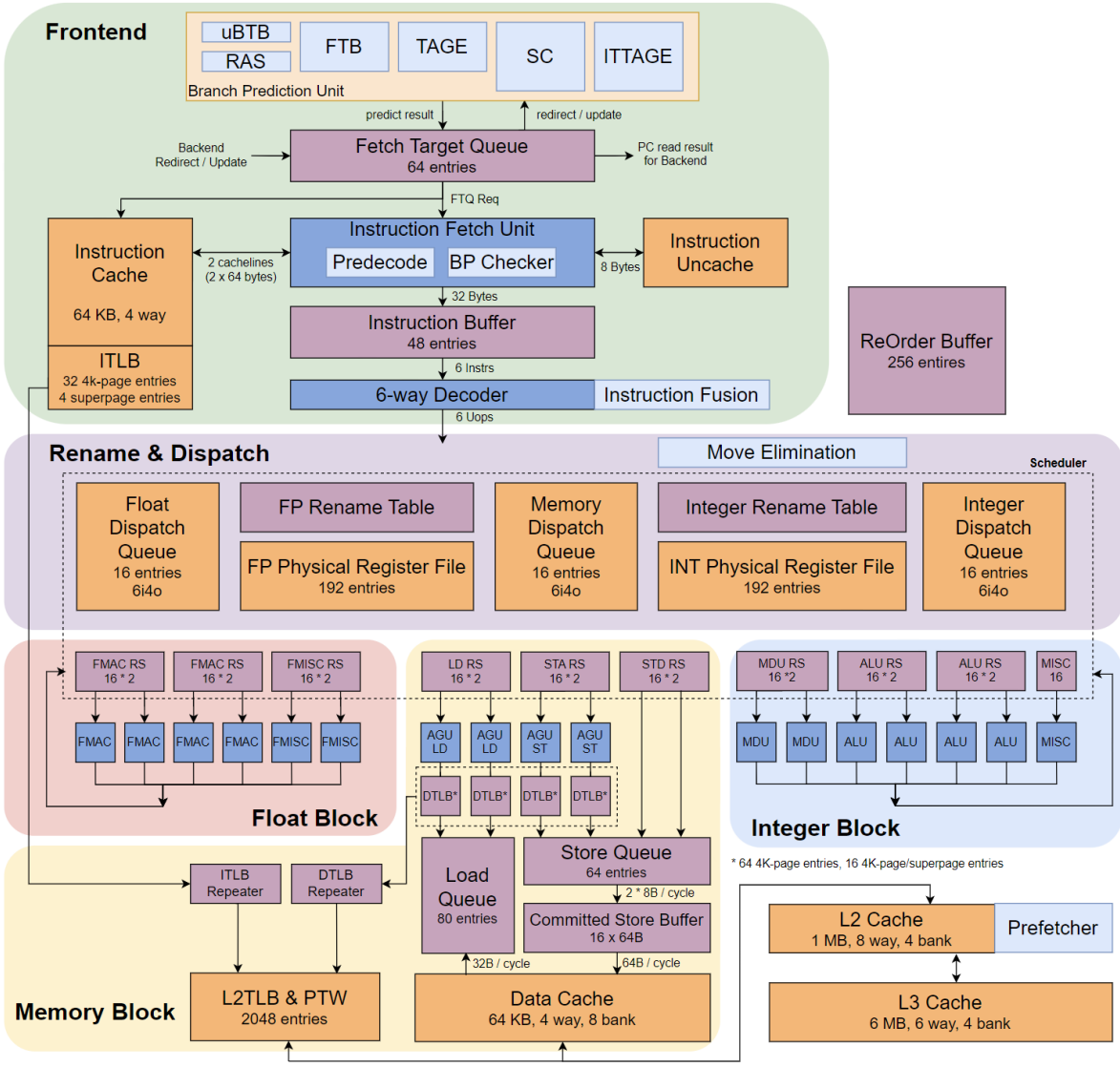
- Named after a lake in Jiaxing, Zhejiang, China
- Target: **2GHz@14nm**
- Estimated **SPECint 2006 19.10, SPECfp 2006 22.18**

- **Major changes**
  - New frontend design: decoupled BP and instruction fetch
  - Improved backend: better scheduler, instruction fusions, and more
  - New L2 cache: designed for high frequency and high performance
  - Tape-out with dual cores (RV64GCBK), more devices support (PCIe, USB, …)

- **Continuously optimize the performance, frequency, ..., in an agile way**

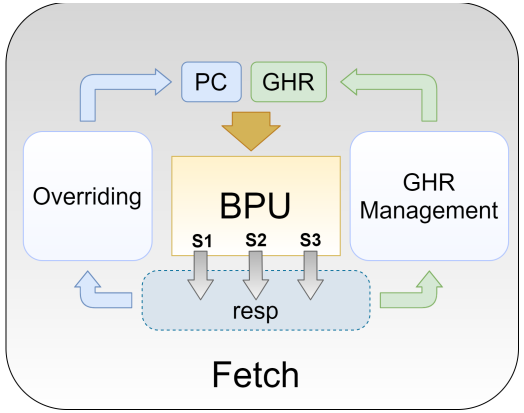
XiangShan Microarchitecture (Nanhu)

- **192** Int PRF + **192** FP PRF
- **256**-entry ROB, **80**-entry LQ, **64**-entry SQ
- **16**-entry RS for each FU (32-entry as an RS)
- **64KB** L1 Cache, **64KB** L1 Data Cache
- **80**-entry DTLB, **36**-entry ITLB, **2K**-entry STLB
- **1MB** non-inclusive L2 Cache
- **6MB** non-inclusive L3 Cache (LLC)



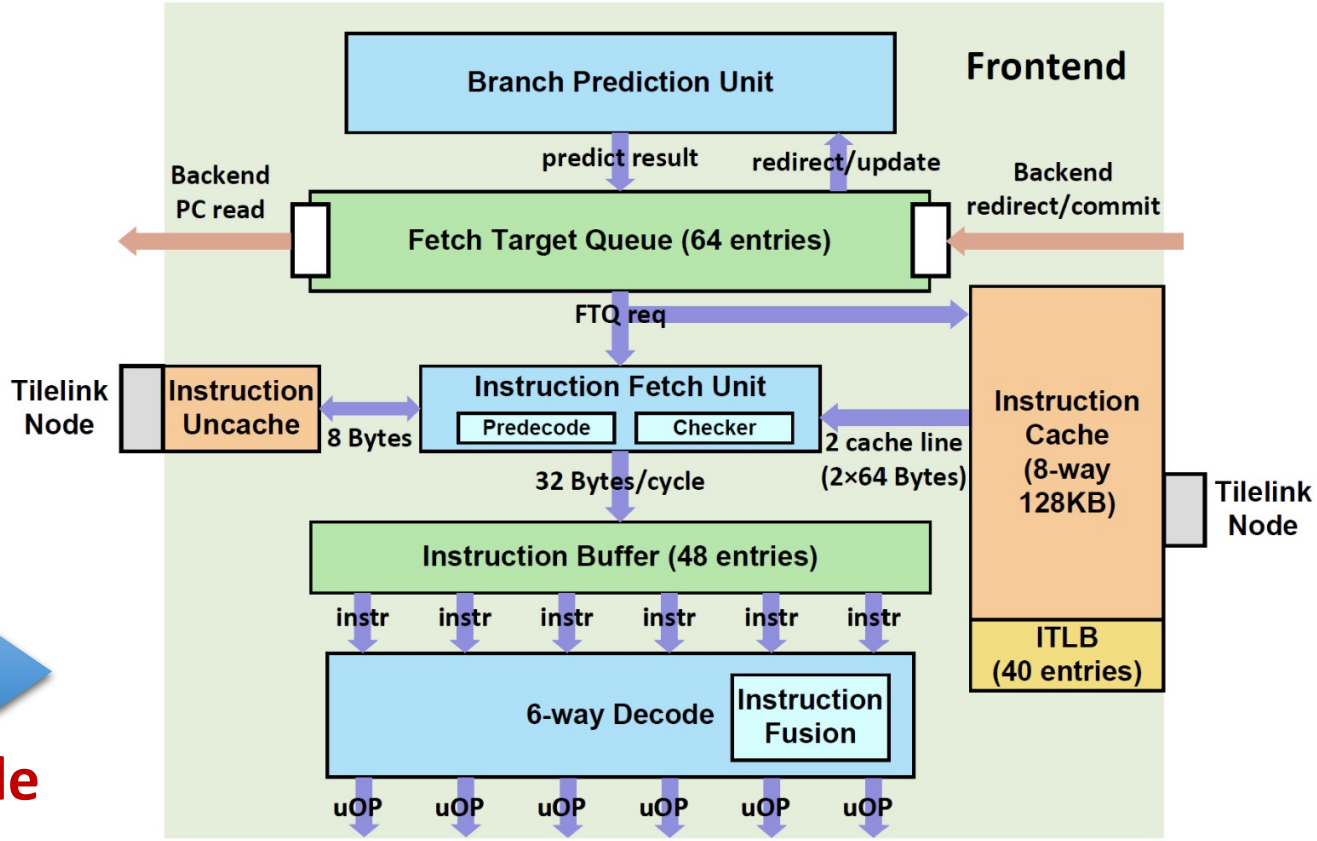
Frontend

- Decoupled frontend
  - **Producer**: branch predictor unit
  - **Consumer**: instruction fetch unit
- Merit
  - Fewer fetch bubbles
  - Fetch directed instr. prefetching



Yanqihu

Upgrade



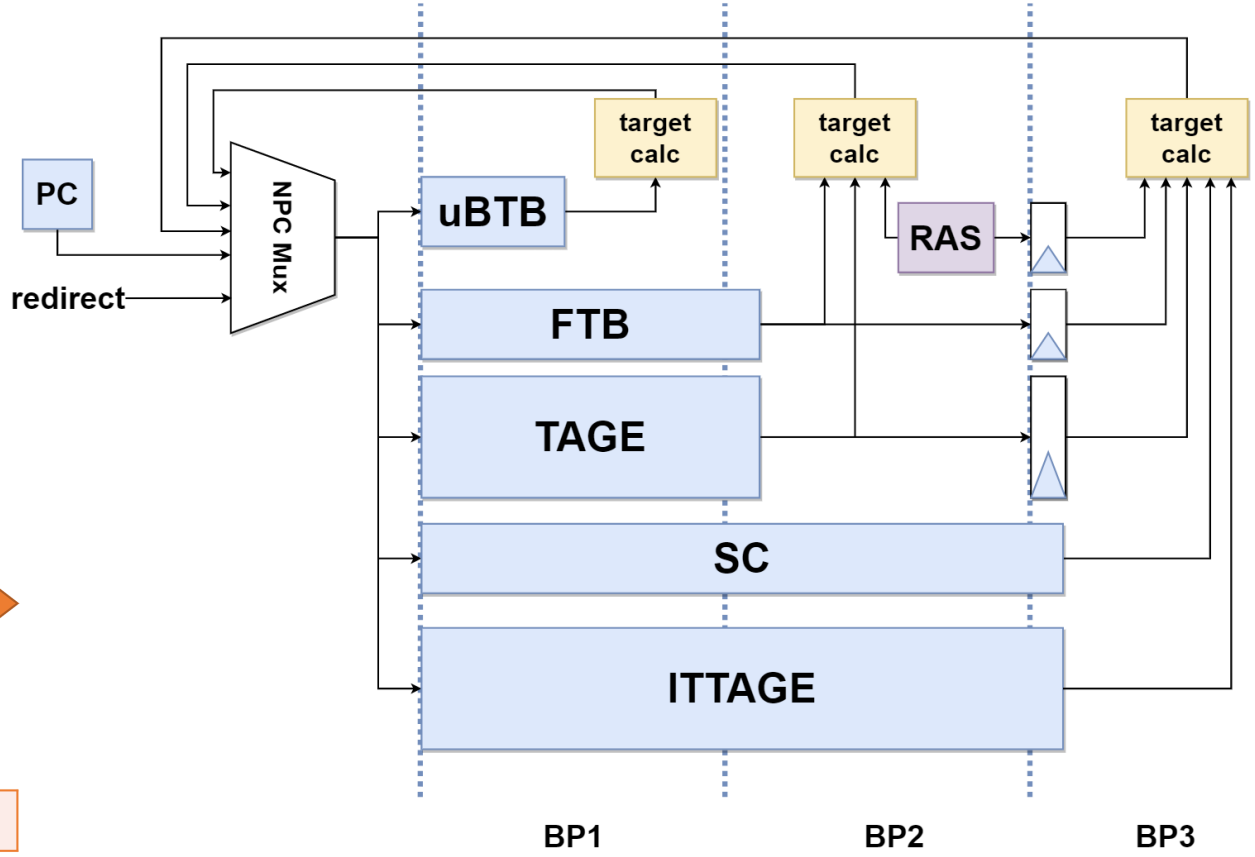
Nanhu

* Frontend design of YQH is partially learnt from BOOM

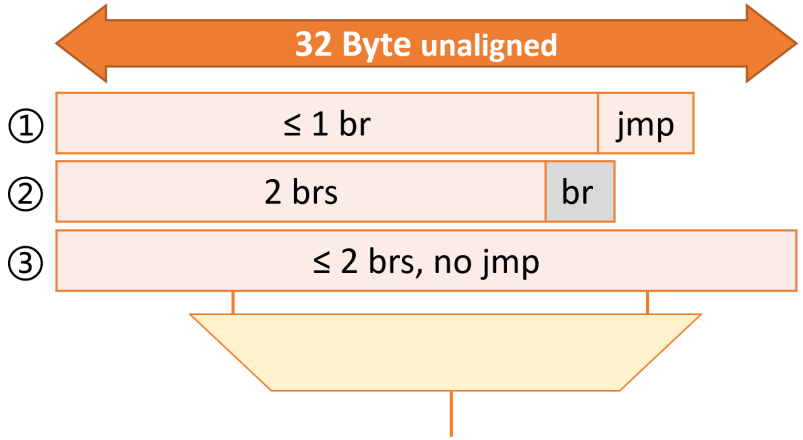
Branch Predictor

Three-stage overriding prediction

- **Stage 1**：uBTB
- **Stage 2**：FTB + TAGE
- **Stage 3**：SC + ITTAGE + RAS

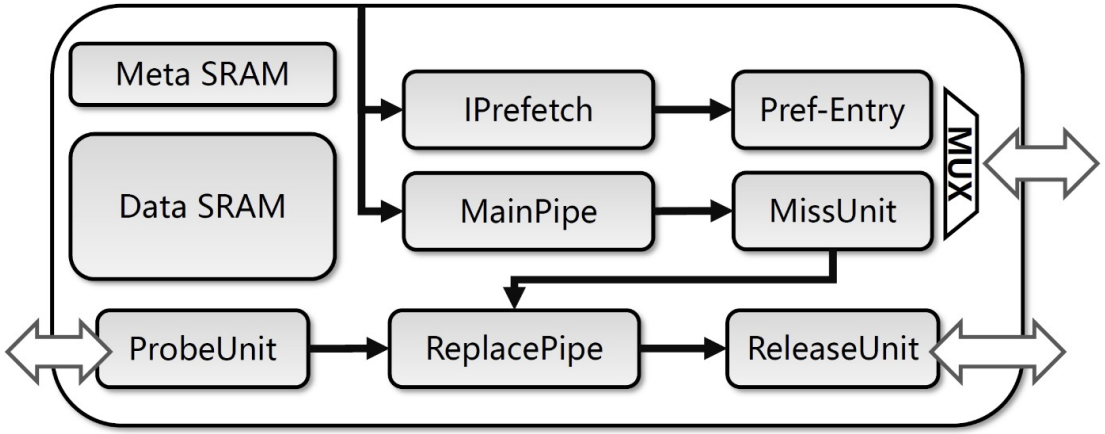


- FTB: Fetch Target Buffer



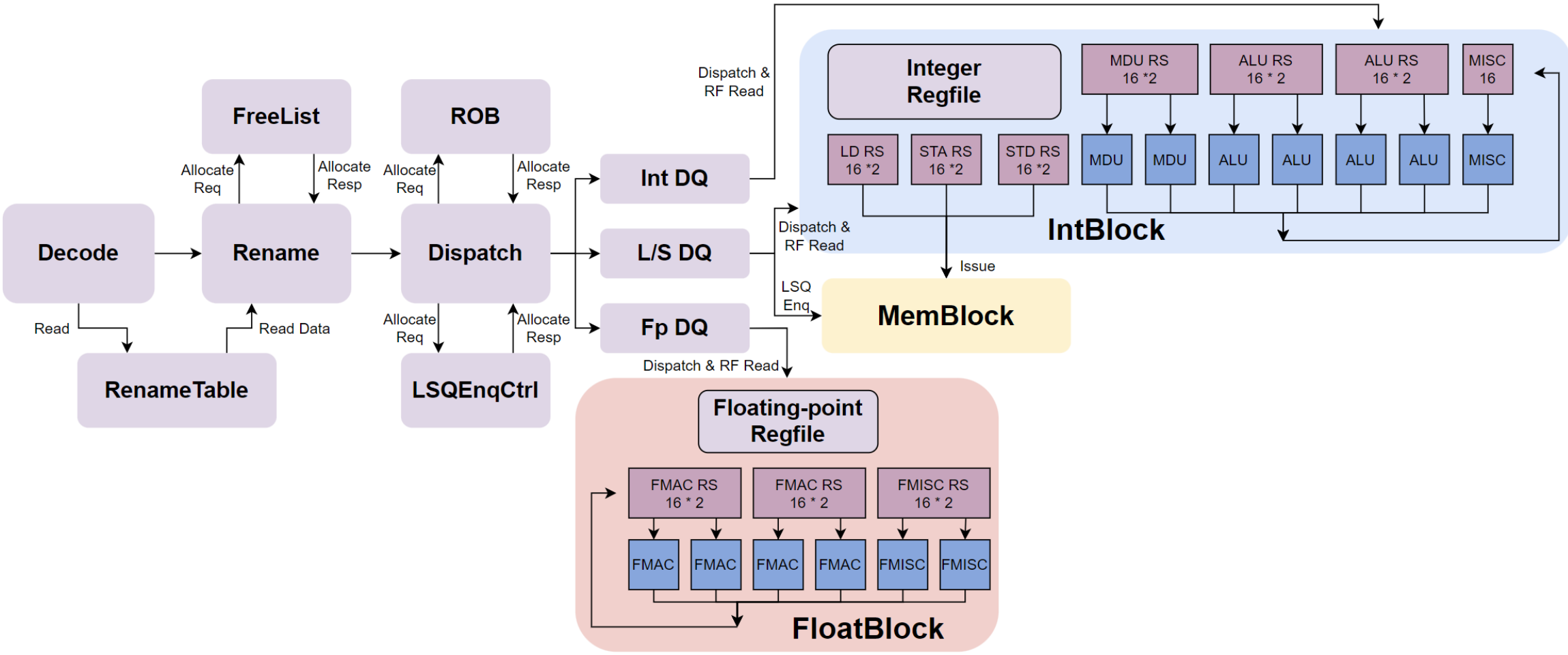
Instruction Cache

- **Main feature**
  - 64KB 4-w VIPT blocking cache
  - Even/odd bank interleaving read
  - **Support TileLink protocol**
  - PLRU replacement



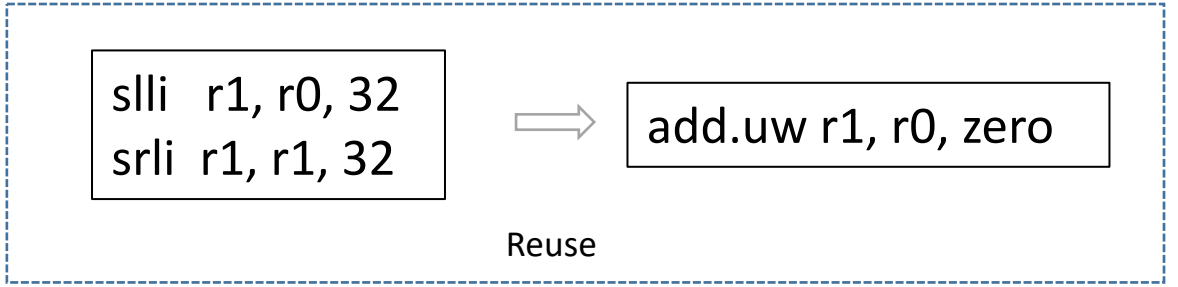
	Yanqihu	Nanhu	
ICache level	Two	One	Reduce
Size	16KB, 4-w	64KB, 4-w	4×
L1plus Cache	128KB, 8-w	-	Reduce
Read bandwidth	64 B	128 B	2×
TileLink Support	No	Yes	New
Instruction prefetch	Stream	L2 FDIP	New

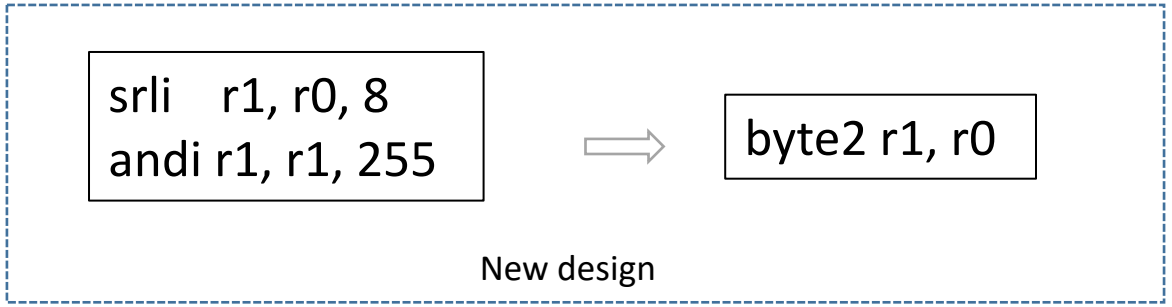
Backend


FreeList
ROB
Allocate Req
Allocate Resp
Allocate Req
Allocate Resp
Decode
Rename
Dispatch
Int DQ
L/S DQ
Fp DQ
Read
Read Data
Allocate Req
Allocate Resp
RenameTable
LSQEnqCtrl
Dispatch & RF Read
Integer Regfile
MDU RS 16 *2
ALU RS 16 * 2
ALU RS 16 * 2
MISC 16
LD RS 16 *2
STA RS 16 *2
STD RS 16 *2
MDU
MDU
ALU
ALU
ALU
ALU
MISC
IntBlock
Dispatch & RF Read
Issue
LSQ Enq
MemBlock
Dispatch & RF Read
Floating-point Regfile
FMAC RS 16 * 2
FMAC RS 16 * 2
FMISC RS 16 * 2
FMAC
FMAC
FMAC
FMAC
FMISC
FMISC
FloatBlock

Instruction Fusion

- Fusion of adjacent UOPs
  - Improve backend efficiency
  - Reduce data bypass delay
- How can we do that?
  1. Reuse of the original calculation
  2. Designing new types of calculations
- **Reduce dynamic instr. greatly**





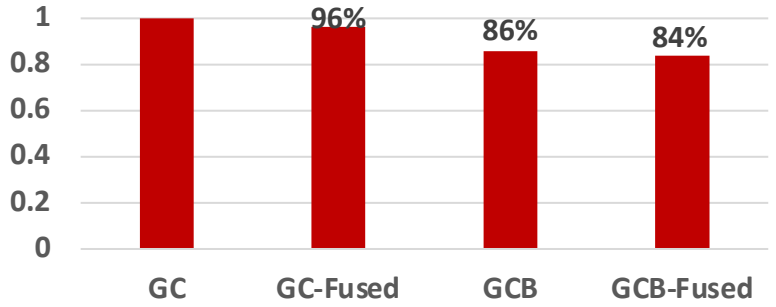
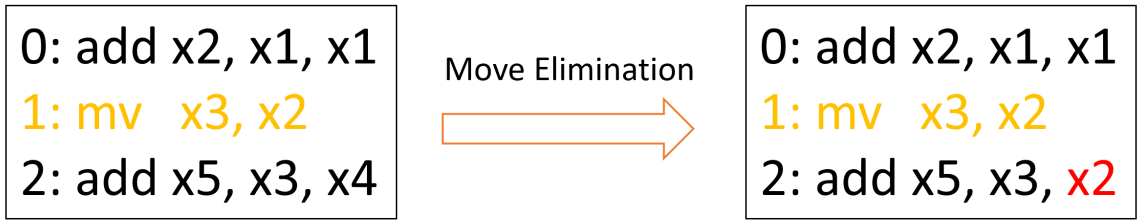
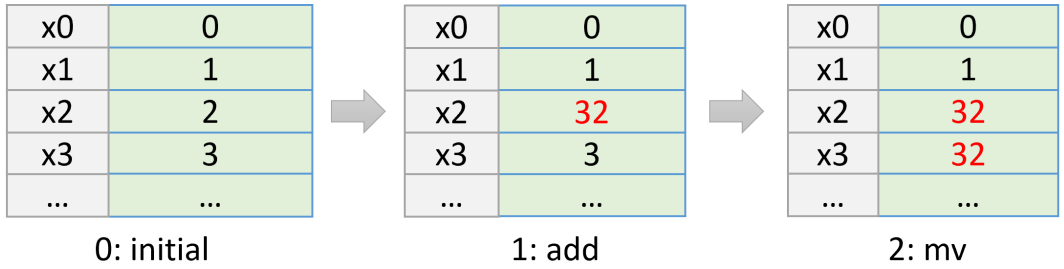


Figure. Normalized Dynamic Instruction Count for CoreMark (-O2)

Rename & Move Elimination

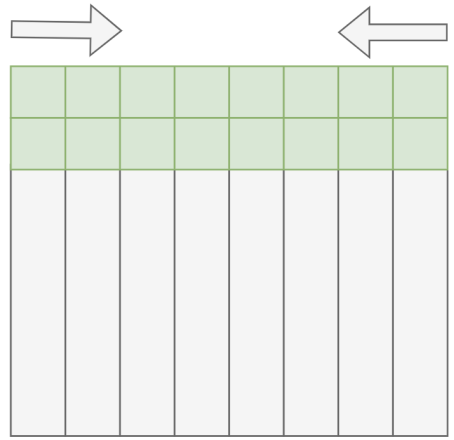
- Maintain the mapping between **arch registers** and **physical registers**
- Move Elimination
  - Mark as **Complete** when Move instruction enters ROB
  - Use **Reference-Counter** to record mapping counts of physical registers



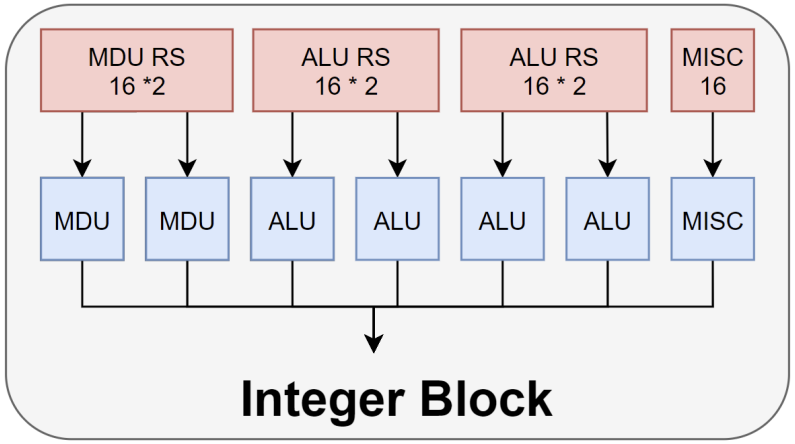


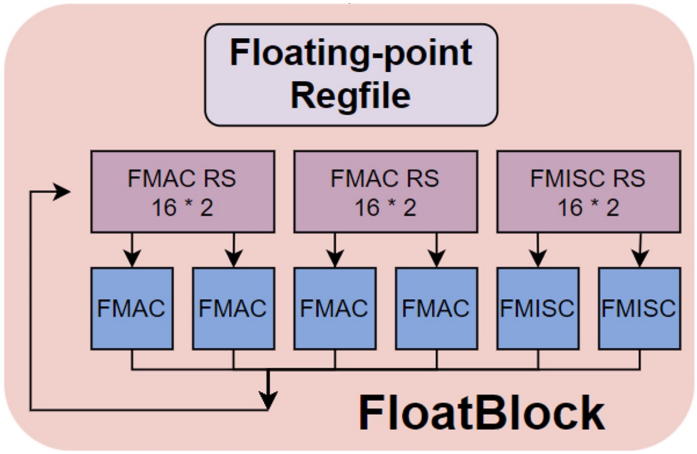
Reservation Station

- Hybrid reservation station design
  - 2i1o & 1i1o
  - normal selection & age matrix base selection



Normal Select





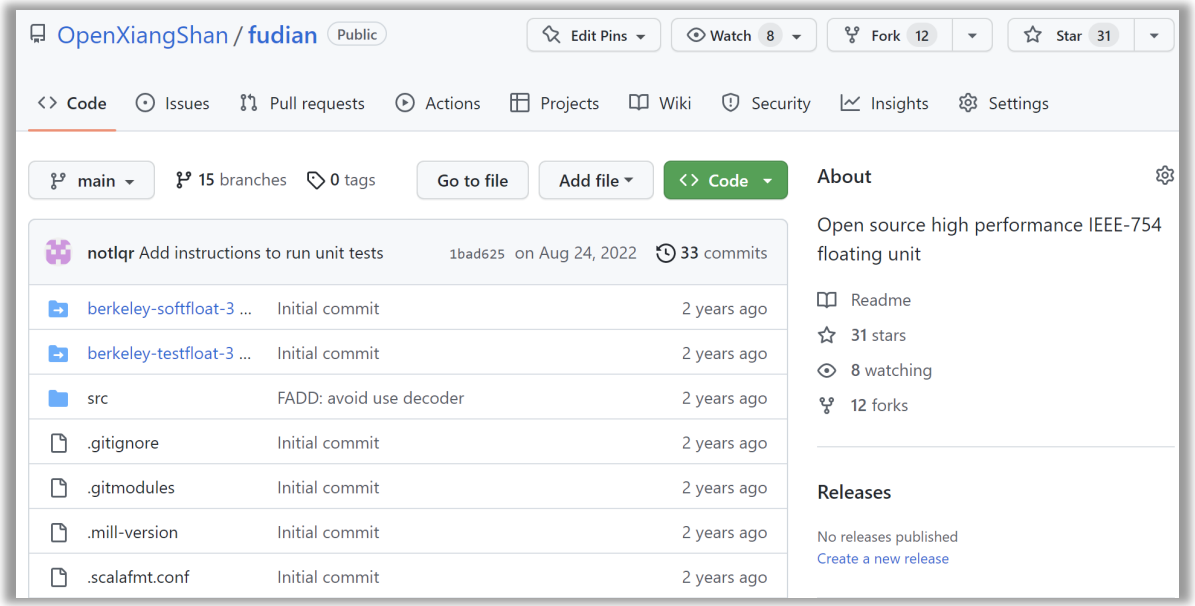
0	true			
1	true	true		
2	true	false	true	
3	true	false	false	true
	0	1	2	3

Age Matrix

Floating-Point Units (FPUs)

- Industry competitive Float Point Unit, **IEEE 754** compatible

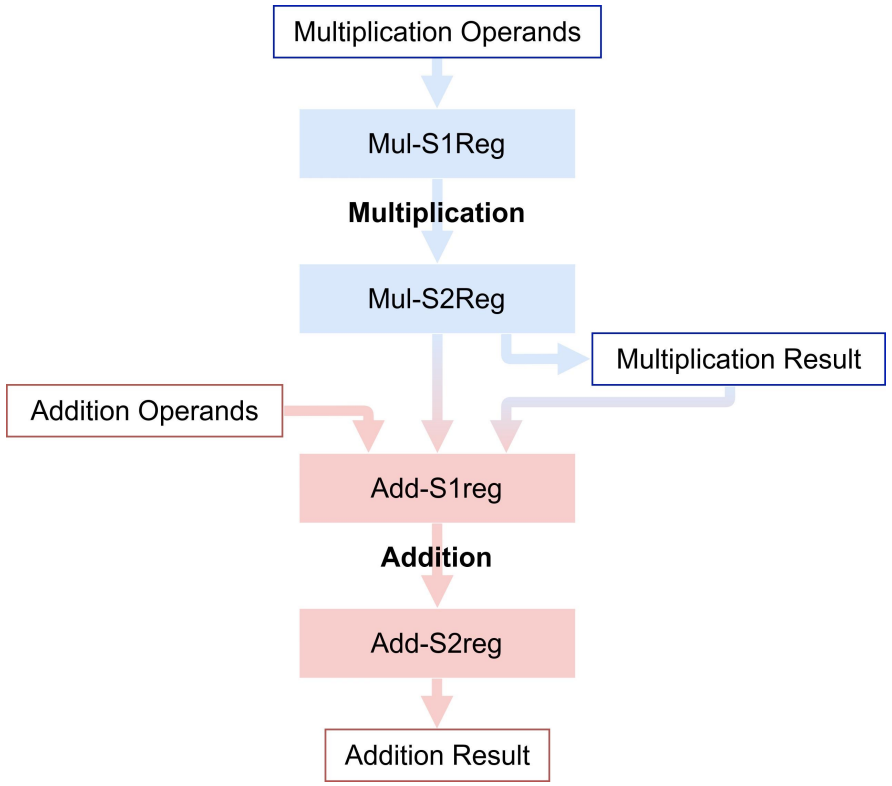
Operations	Latency
FADD	3
FMUL	3
FMA	5
FDIV	≤ 11 (float) ≤ 18 (double)
FSQRT	≤ 17 (float) ≤ 31 (double)
FCVT, FCMP	3



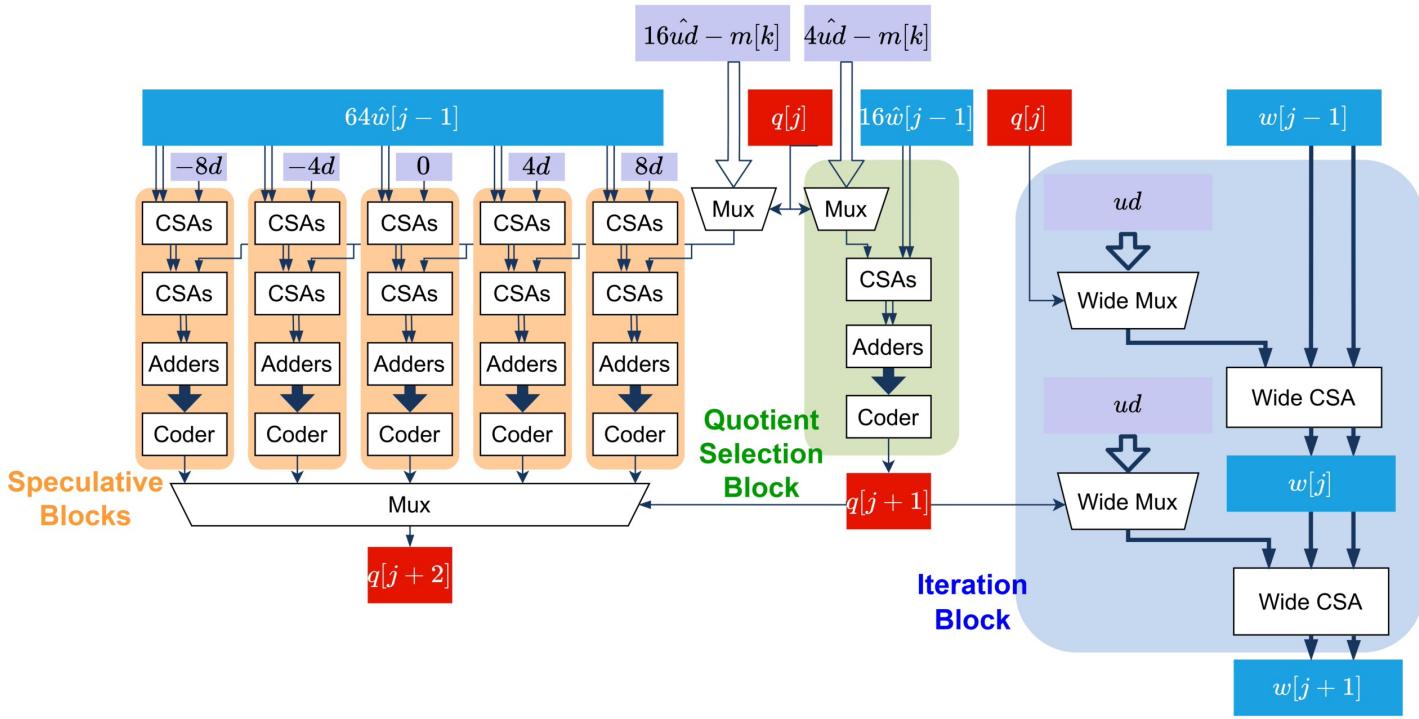
https://github.com/OpenXiangShan/fudian

FPU

- **Cascade FMA**
  - Reduce FADD delay 5->3



- **SRT16 Division units**
  - ½ delay compared with Yanqihu (SRT4)



B/K Extension Implemented

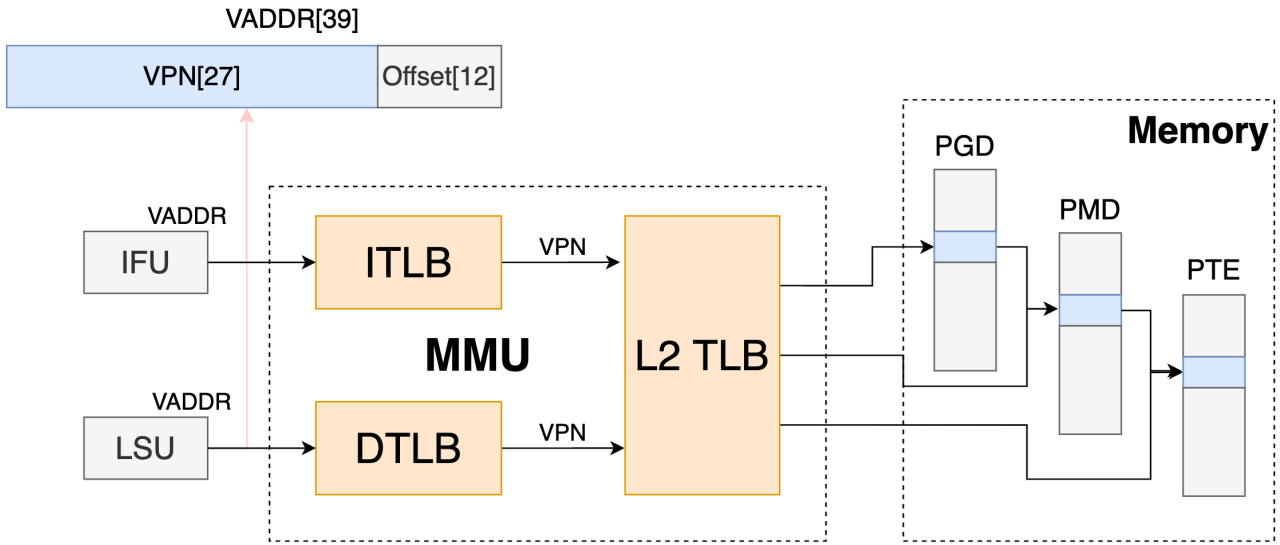
- **B**: Bit-Manipulation extension
- **K**: Cryptographic extension
- * **V**(Vector)/**H**(Hypervisor) extensions are in development!

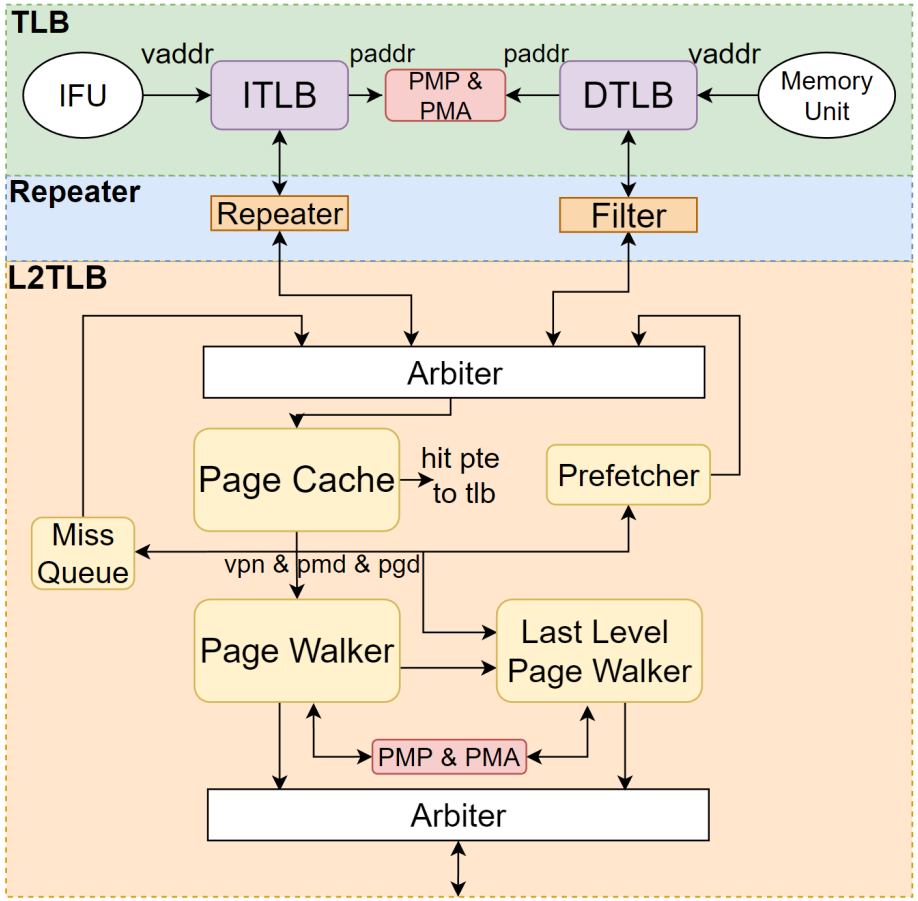
B Extension

K Extension

Memory Management Unit

- Translate from virtual address to physical address
- Privilege checking
- Support Sv39

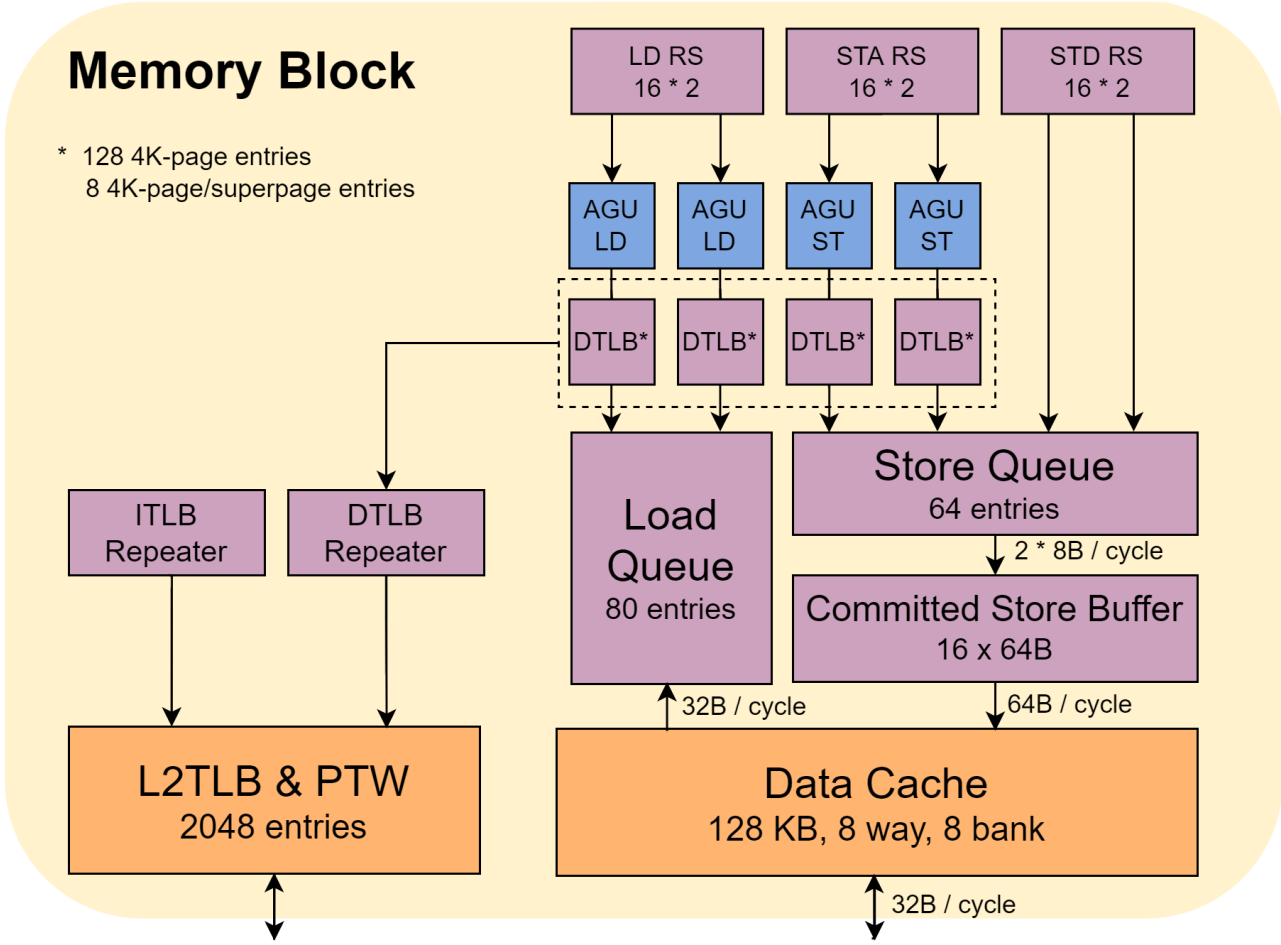




Memory Block

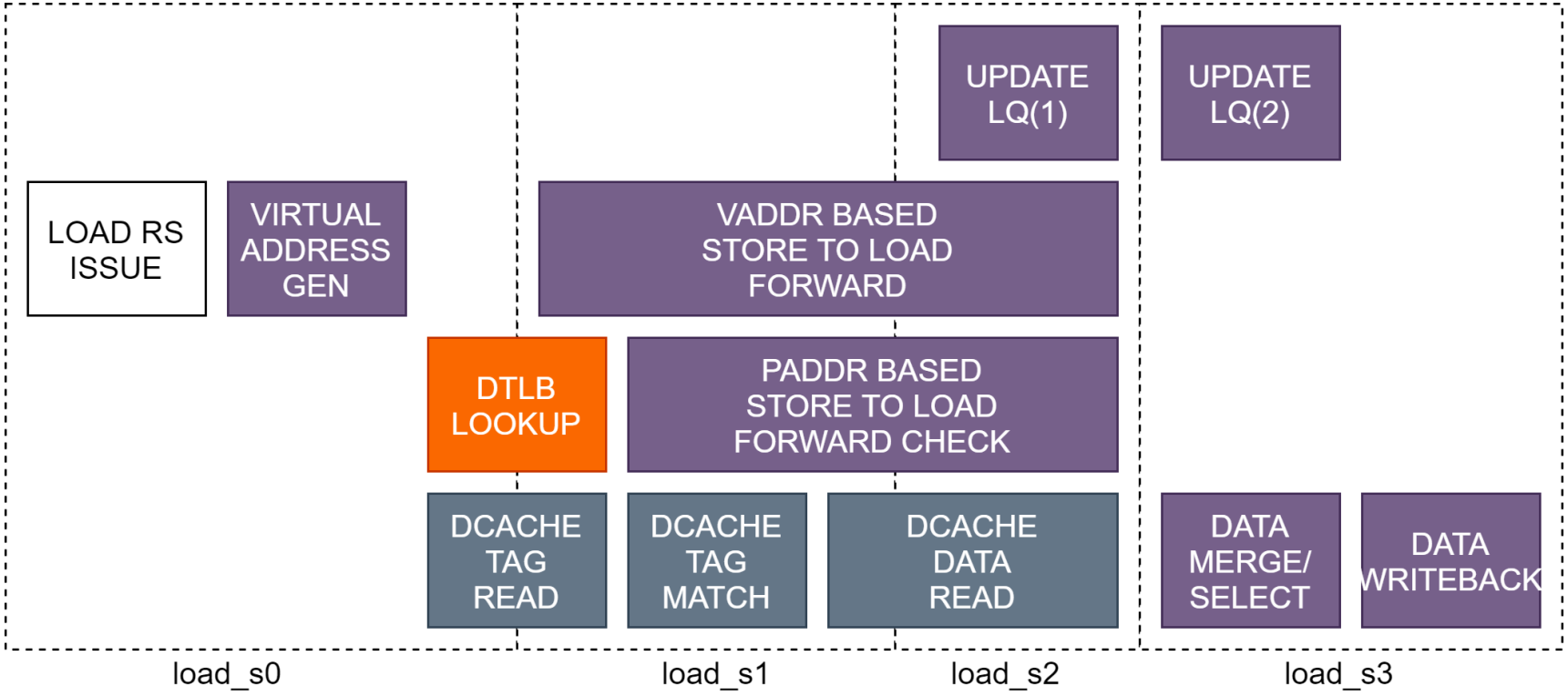
- Datasheets

Load pipeline	2
Store address pipeline	2
Store data pipeline	2
L1 LD hit latency	4
L1 LD hit bandwidth	2x8B/cycle
Store data bandwidth	2x8B/cycle
Load Queue Entry	80
Store Queue Entry	64
Merged Store Buffer	16 cachelines



Load Pipeline

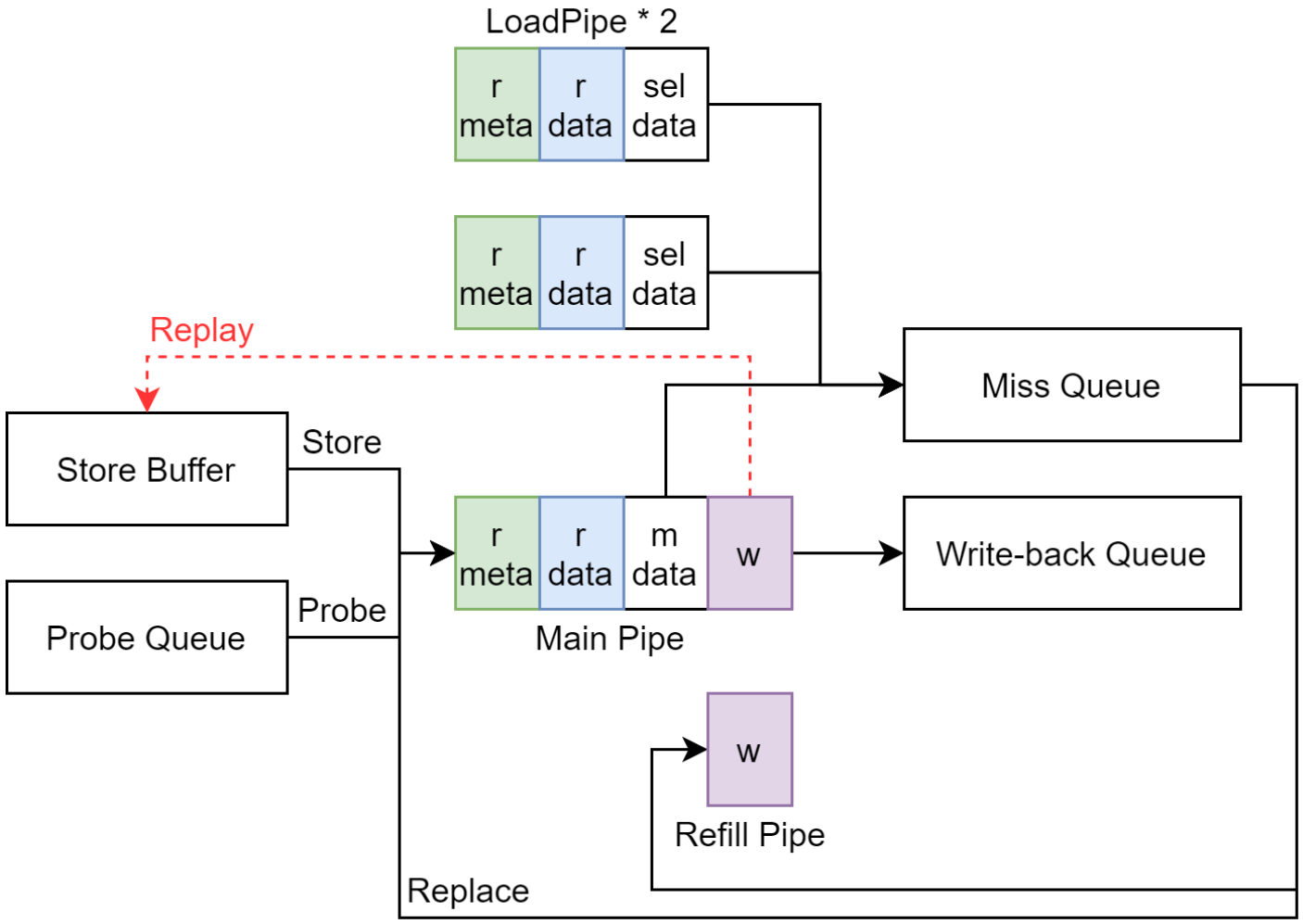
- Four stage
- Optimized for load-to-load & load-store bypass



L1 DCache

- Coupled with load pipeline
- TileLink protocol

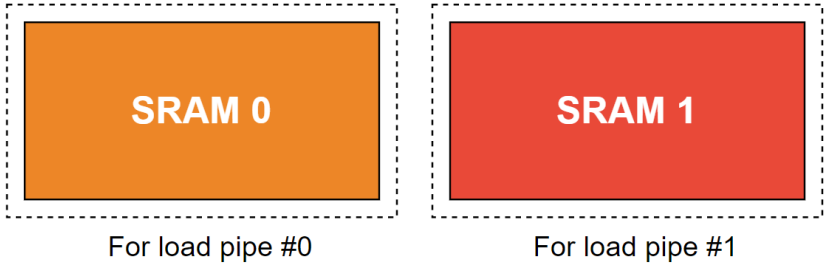
L1 Data cache size	**64KB**
L1/L2 Bus Width	256bit
Store Buffer	16
Probe Queue	8
Miss Queue	16
Write-back Queue	18
Prefetch Instruction	**Support**
ECC	**Support**



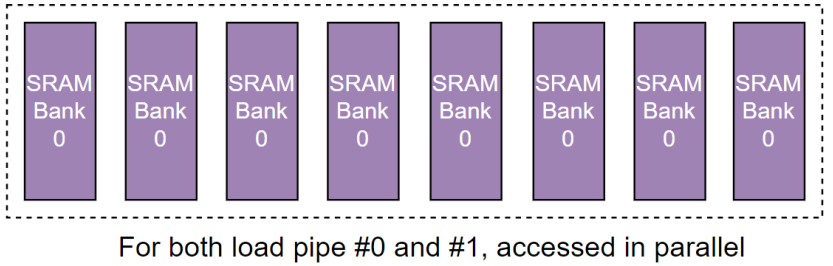
L1 DCache

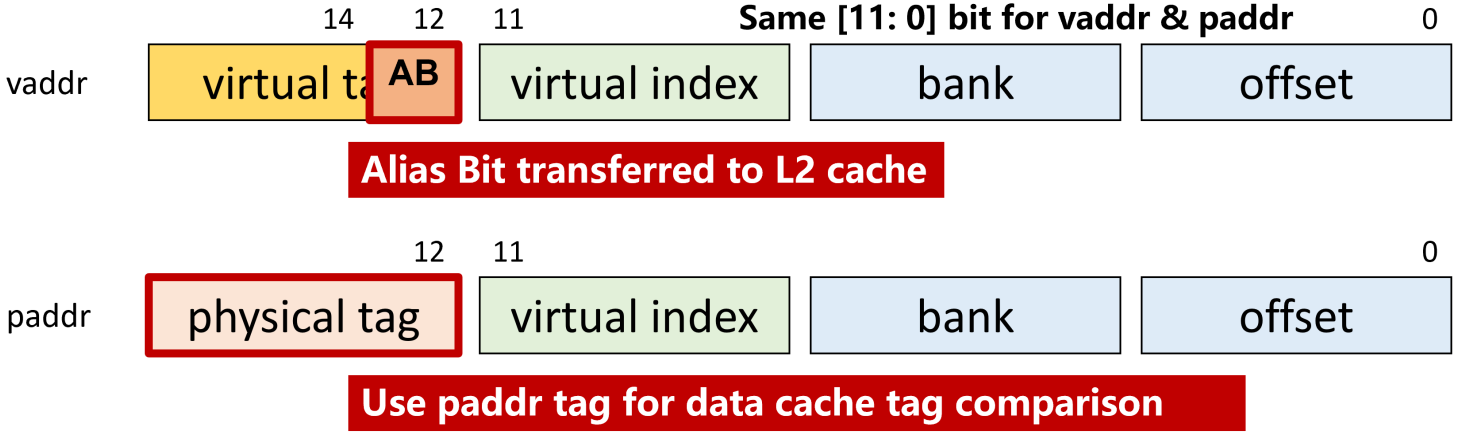
- Noteworthy features
  - Multi-bank
  - Anti-alias
  - MSHR request merge
  - Optional L1D prefetcher

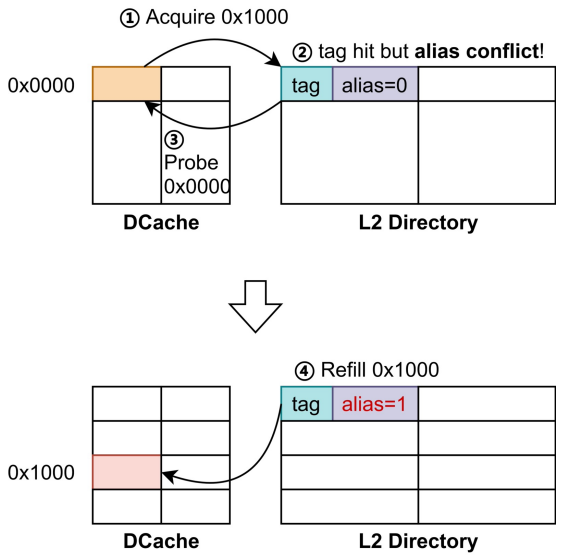
Yanqihu



Nanhu

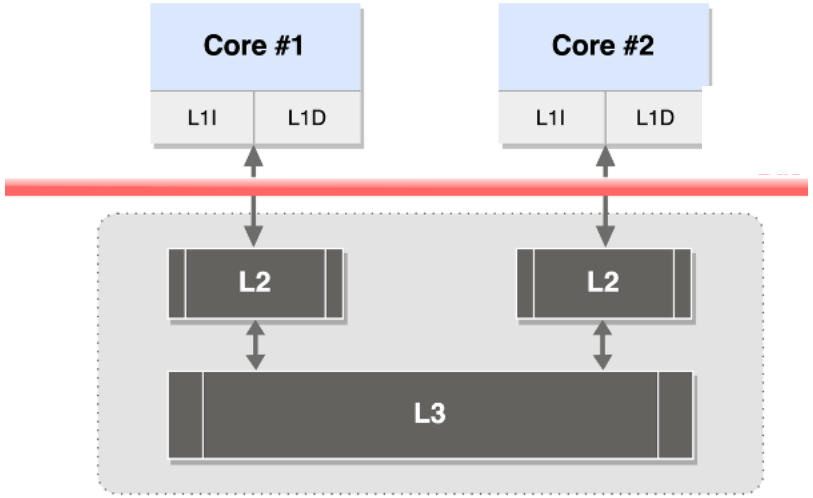




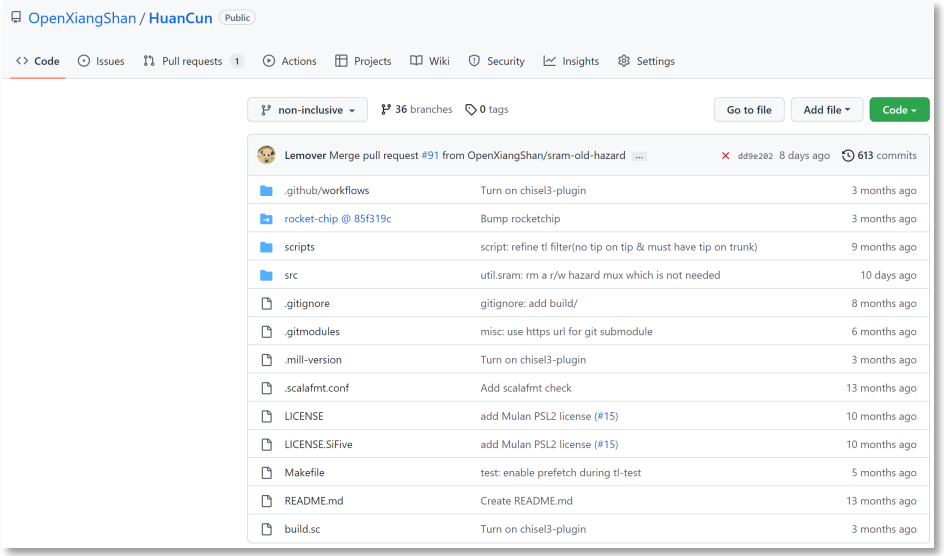


L2/L3 Cache

- High performance non-blocking **L2/L3** cache
- Design features
  - **Directory based coherence**
  - **Inclusive/Non-inclusive**
  - **TileLink protocol**
- Highly configurable parameters
  - Slice, Size, #Ways, #MSHRs
  - Replacement: **Random/PLRU/RRIP**
  - Prefetch: **BOP/SMS**
- Maintain coherence with **both** L1I & L1D



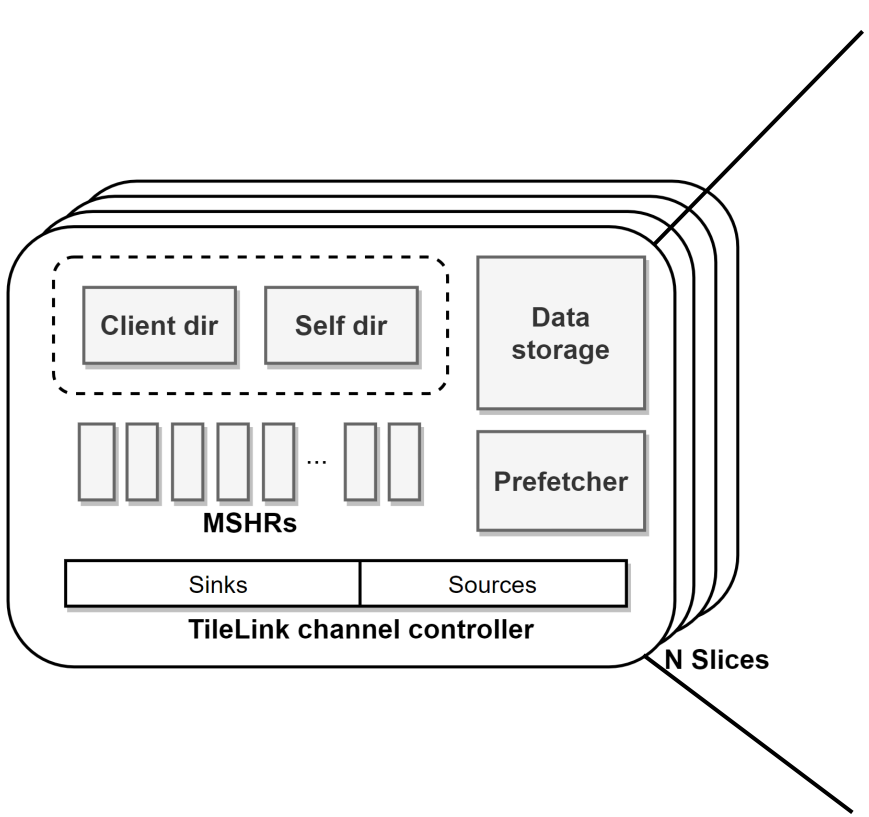
Nanhu Dual-core

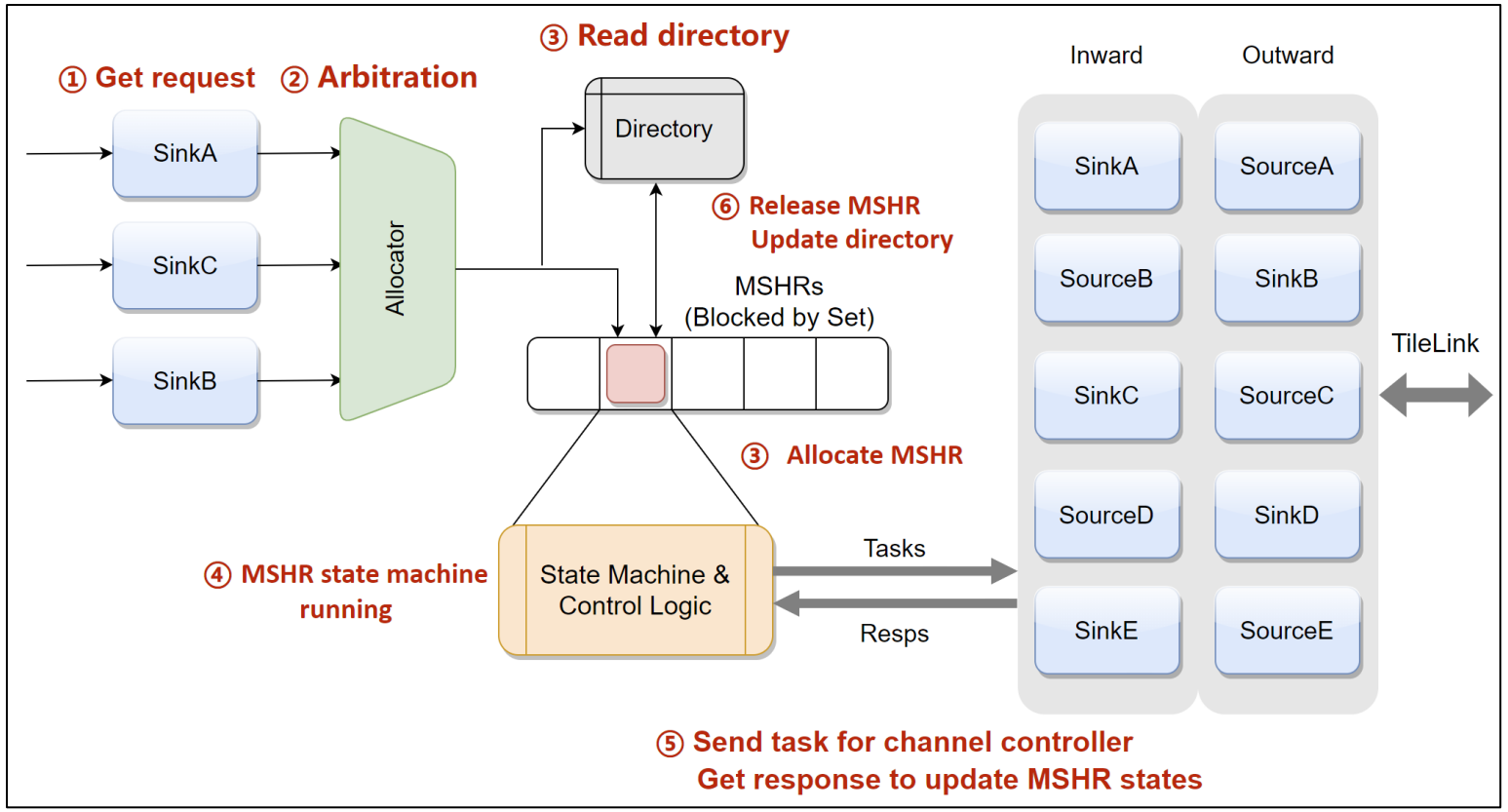


https://github.com/OpenXiangShan/HuanCun

L2/L3 Cache

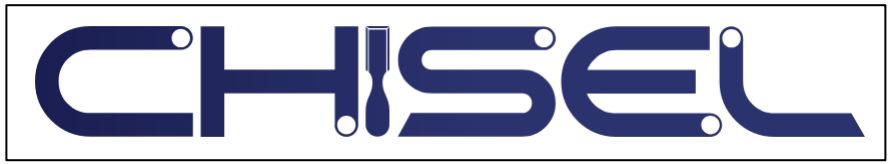
- Concurrent request processing flow





Microarchitecture Implementation

- **Implement by Chisel HDL**
  - High readability
  - High modifiability



- Beyond built-ins, we designed some open-source high-performance utilities/components

BinaryArbiterNode.scala
BitUtils.scala
ChiselDB.scala
CircularQueuePtr.scala
Constantin.scala
DataModuleTemplate.scala
ECC.scala
ExcitingUtils.scala
ExtractVerilogModules.scala
FastArbiter.scala
FileRegisters.scala
GTimer.scala
Hold.scala
IntBuffer.scala
LFSR64.scala
LatencyPipe.scala
LookupTree.scala
MIMOQueue.scala
Misc.scala
ParallelMux.scala
Pipeline.scala
PipelineConnect.scala
PriorityMuxDefault.scala
PriorityMuxGen.scala
RegMap.scala
Replacement.scala
ResetGen.scala
SRAMTemplate.scala
StopWatch.scala
TLClientsMerger.scala
TLEdgeBuffer.scala

Check it out!

Microarchitecture Implementation

- **Highly configurable**
  - You can setup parameters as you wish

```
IBufSize: Int = 48,
DecodeWidth: Int = 6,
RenameWidth: Int = 6,
CommitWidth: Int = 6,
FtqSize: Int = 64,
EnableLoadFastWakeUp: Boolean = true,
IssQueSize: Int = 16,
NRPhyRegs: Int = 192,
LoadQueueSize: Int = 80,
LoadQueueNWriteBanks: Int = 8,
StoreQueueSize: Int = 64,
StoreQueueNWriteBanks: Int = 8,
RobSize: Int = 256,
dpParams: DispatchParameters =
  DispatchParameters(
    IntDqSize = 16,
    FpDqSize = 16,
    LsDqSize = 16,
    IntDqDeqWidth = 4,
    FpDqDeqWidth = 4,
    LsDqDeqWidth = 4
  ),
exuParameters: ExuParameters =
  ExuParameters(
    JmpCnt = 1,
    AluCnt = 4,
    MulCnt = 0,
    MduCnt = 2,
    FmacCnt = 4,
    FmiscCnt = 2,
    FmiscDivSqrtCnt = 0,
    LduCnt = 2,
    StuCnt = 2
  ),
prefetcher: Option[PrefetcherParams] =
  Some(SMSParams()),
LoadPipelineWidth: Int = 2,
StorePipelineWidth: Int = 2,
StoreBufferSize: Int = 16,
StoreBufferThreshold: Int = 7,
EnableLoadToLoadForward: Boolean = true,
EnableFastForward: Boolean = false,
EnableLdVioCheckAfterReset: Boolean = true,
EnableSoftPrefetchAfterReset: Boolean = true,
EnableCacheErrorAfterReset: Boolean = true,
EnablePTWPreferCache: Boolean = true,
EnableAccurateLoadError: Boolean = true,
```

Some key parameters

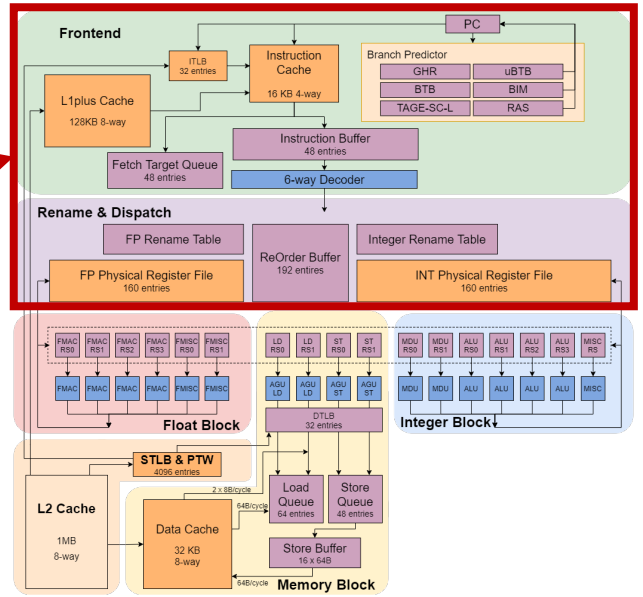
e.g.

How to change rename/decode/issue width?

All you need is to change a number

```
23  // Synthesizable minimal XiangShan
24  // * It is still an out-of-order, super-scalaer arch
25  // * L1 cache included
26  // * L2 cache NOT included
27  // * L3 cache included
28  class MinimalConfig(n: Int = 1) extends Config(
29    new DefaultConfig(n).alter((site, here, up) => {
30      case SoCParamsKey => up(SoCParamsKey).copy(
31        cores = up(SoCParamsKey).cores.map(_.copy(
32          DecodeWidth = 2,
33          RenameWidth = 2,
34          FetchWidth = 4,
35          IssQueSize = 8,
36          NRPhyRegs = 80,
37          LoadQueueSize = 16,
38          StoreQueueSize = 16,
39          RoqSize = 32,
40          BrqSize = 8,
41          FtqSize = 16,
42          IBufSize = 16,
43          StoreBufferSize = 4,
44          StoreBufferThreshold = 3,
45          dpParams = DispatchParameters(
46            IntDqSize = 8,
47            FpDqSize = 8,
48            LsDqSize = 8,
```

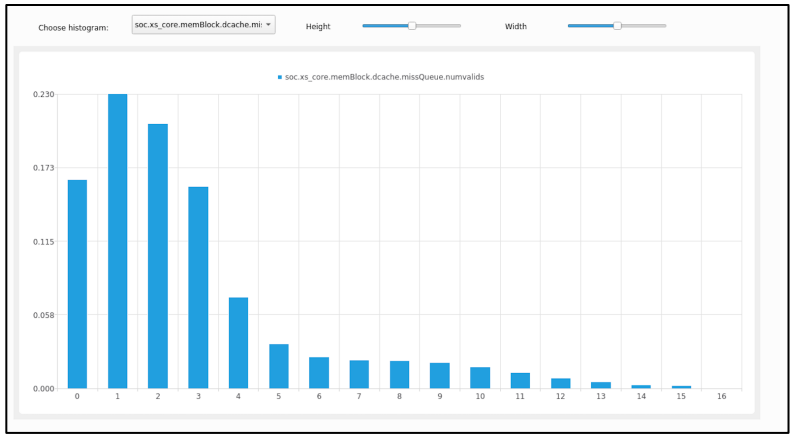
All affected
But it works

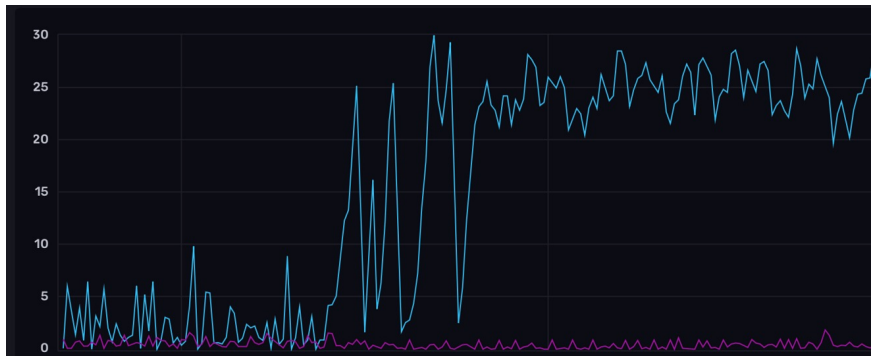


Microarchitecture Implementation

- **Easy-to-use performance counter system**
- Multiple collection types
  - Cumulative counter
  - Delay counter
  - Transaction counter
- Multiple analysis style
  - Key-value list
  - Histogram
  - Database
  - Time series visualization

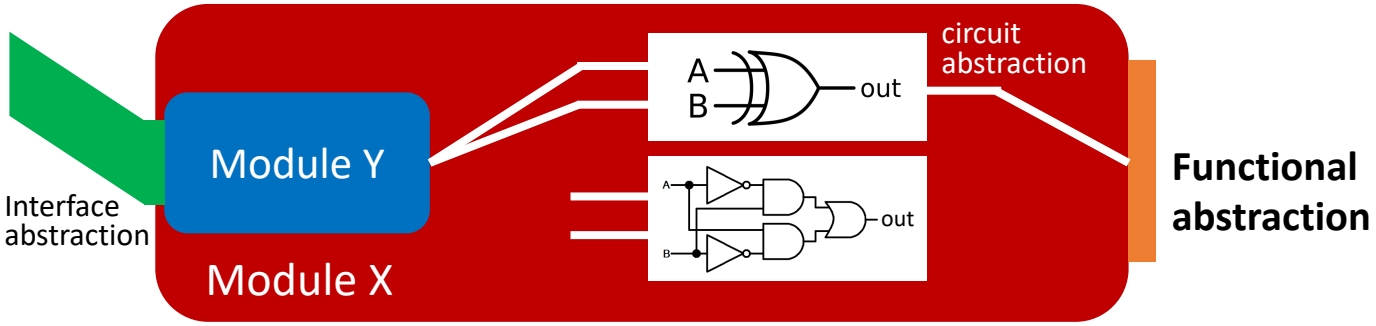
```
TOP.XSSimSoC.soc.xs_core.ctrlBlock.ftq: BpBInstr,              57899
TOP.XSSimSoC.soc.xs_core.ctrlBlock.ftq: BpBRight,              54695
TOP.XSSimSoC.soc.xs_core.ctrlBlock.ftq: BpBWrong,               3204
TOP.XSSimSoC.soc.xs_core.ctrlBlock.ftq: BpCRight,               3615
TOP.XSSimSoC.soc.xs_core.ctrlBlock.ftq: BpCWrong,                 34
TOP.XSSimSoC.soc.xs_core.ctrlBlock.ftq: BpInstr,               78176
TOP.XSSimSoC.soc.xs_core.ctrlBlock.ftq: BpIRight,               5471
TOP.XSSimSoC.soc.xs_core.ctrlBlock.ftq: BpIWrong,               6099
TOP.XSSimSoC.soc.xs_core.ctrlBlock.ftq: BpJRight,               8707
TOP.XSSimSoC.soc.xs_core.ctrlBlock.ftq: BpJWrong,                  0
TOP.XSSimSoC.soc.xs_core.ctrlBlock.ftq: BpRight,               68873
```





Microarchitecture Implementation

- Modules are designed according to object-oriented programming
- Support multi-level abstraction



- e.g., to implement a new prefetcher
  - We provide a clean set of interfaces that allow the developer to focus on the implementation of the algorithm itself

Thanks!

This tutorial is organized by Kaifan Wang, Yinan Xu, Guokai Chen, Xi Chen, Haoyuan Feng, Ziyue Zhang, Kan Shi, and Yungang Bao.