|  |  |
| --- | --- |
| 文档编号 |  |
| 文档版本 |  |
| 文档管控 | 内部公开 |
| 存档日期 |  |

XX项目XX模块AS

|  |  |
| --- | --- |
| 编 写： | 陈国凯 高泽宇 满洋 |
| 校 对： |  |
| 审 核： |  |
| 批 准： |  |

昆明湖V1项目

2024年XX月XX日

文档修订记录

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **序号** | **版本编号** | **变化状态** | **变更说明** | **作者** | **日期** |
| 1 | V0.1 | C | 昆明湖版本初稿 | 陈国凯 高泽宇 满洋 | 2024-01-15 |
|  |  |  |  |  |  |
| 3 |  |  |  |  |  |
|  |  |  |  |  |  |

\*变化状态：C—创建，A—增加，M—修改，D—删除

文档审批信息

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **版本** | **审核** | **会签** | **批准** | **备注** |
| V1.0 |  |  |  |  |
|  |  |  |  |  |

目 录

[1 简介 1](#_Toc107389929)

[1.1 文档介绍 1](#_Toc107389930)

[1.2 参考文档 1](#_Toc107389931)

[1.3 术语说明 1](#_Toc107389932)

[1.4 技术背景 1](#_Toc107389933)

[2 设计规格 2](#_Toc107389934)

[3 功能描述 2](#_Toc107389935)

[4 总体设计 2](#_Toc107389936)

[4.1 整体框图 2](#_Toc107389937)

[4.2 接口列表 3](#_Toc107389938)

[4.3 接口时序 3](#_Toc107389939)

[4.4 时钟复位 4](#_Toc107389940)

[4.5 寄存器配置 4](#_Toc107389941)

[4.6 补充说明 5](#_Toc107389942)

[5 模块设计 5](#_Toc107389943)

[5.1 二级模块A 6](#_Toc107389944)

[5.1.1 功能 6](#_Toc107389945)

[5.1.2 整体框图 6](#_Toc107389946)

[5.1.3 接口列表 6](#_Toc107389947)

[5.1.4 接口时序 6](#_Toc107389948)

[5.1.5 关键电路 6](#_Toc107389949)

[5.1.6 三级模块设计 6](#_Toc107389950)

[5.2 二级模块B 6](#_Toc107389951)

[5.3 关键电路 6](#_Toc107389952)

[5.3.1 时钟切换电路 6](#_Toc107389953)

[6 PPA优化设计 7](#_Toc107389954)

[7 验证关注点 7](#_Toc107389955)

[8 Floorplan 建议 7](#_Toc107389956)

[9 遗留问题 8](#_Toc107389957)

1. 简介
   1. 文档介绍

本文档是分支预测单元（BPU）的文档，描述南湖架构分支预测单元设计。

本文档主要用于指导芯片模块的详细设计及验证。

* 1. 参考文档

1. Reinman G, Austin T, Calder B. A scalable front-end architecture for fast instruction delivery[J]. ACM SIGARCH Computer Architecture News, 1999, 27(2): 234-245. ↩↩↩
2. Perais A, Sheikh R, Yen L, et al. Elastic instruction fetching[C]//2019 IEEE International Symposium on High Performance Computer Architecture (HPCA). IEEE, 2019: 478-490. ↩
3. Software Optimization Guide for AMD Family 19h Processors (PUB), Chap. 2.8.1.5, https://www.amd.com/system/files/TechDocs/56665.zip ↩
4. Seznec A, Michaud P. A case for (partially) TAgged GEometric history length branch prediction[J]. The Journal of Instruction-Level Parallelism, 2006, 8: 23. ↩
5. Seznec A. A 256 kbits l-tage branch predictor[J]. Journal of Instruction-Level Parallelism (JILP) Special Issue: The Second Championship Branch Prediction Competition (CBP-2), 2007, 9: 1-6. ↩
6. Seznec A. A new case for the tage branch predictor[C]//Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture. 2011: 117-127. ↩
7. Seznec A. The O-GEHL branch predictor[J]. The 1st JILP Championship Branch Prediction Competition (CBP-1), 2004. ↩
8. Jiménez D A, Lin C. Dynamic branch prediction with perceptrons[C]//Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture. IEEE, 2001: 197-206. ↩
9. Seznec A. A 64-Kbytes ITTAGE indirect branch predictor[C]//JWAC-2: Championship Branch Prediction. 2011. ↩
   1. 术语说明

表1.1 术语说明

|  |  |  |
| --- | --- | --- |
| **缩写** | **全称** | **描述** |
| BPU | Branch Prediction Unit | 分支预测单元 |
| IFU | Instruction Fetch Unit | 取指单元 |
| FTQ | Fetch Target Queue | 取指目标单元 |
| uFTB | Micro Fetch Target Buffer | 分支目标缓冲 |
| FTB | Fetch Target Buffer | 取指目标缓冲 |
| TAGE | TAgged GEometric length predictor | 一种条件分支预测器 |
| SC | statistical corrector predictor | 一种用于在统计偏向情况下纠正TAGE预测的条件分支预测器 |
| ITTAGE | Indirect Target TAgged GEometric length predictor | 一种用于预测间接跳转指令目标地址的分支预测器 |
| RAS | Return Address Stack | 一种用于预测调用指令对应返回指令目标地址的分支预测器 |

* 1. 技术背景

无。

1. 设计规格
2. 支持一次生成一个分支预测块及其对应预测器附加信息
3. 支持无空泡的简单预测
4. 支持多种精确预测器及覆盖机制
5. 支持训练预测器
6. 支持分支历史信息维护及误预测恢复
7. 支持topdown性能事件的统计
8. 功能描述
   1. 功能概述

BPU模块接收来自模块外部后端执行单元及后续流水级的重定向信号，按照地址采用多种预测器为当前PC值开始的位置生成预测块及生成该预测块时各预测器内部的meta信息，传递给后续取指目标队列（FTQ）存储，预测块供取指单元（IFU）使用，meta信息供未来训练恢复预测器使用。其中，BPU模块使用全相连uFTB作为next line predictor，生成理想条件下延迟仅1周期的无空泡简单预测结果，该结果会被直接作为输出传递到FTQ。与此同时，这一基础预测结果还将在BPU后续流水线内流动，供高级预测部件使用以提供更为精确的预测结果。一旦高级预测器在后续流水级的预测结果与已有结果不一致，就将会使用高级预测器结果作为新的输出更新后续FTQ中存储预测块结果并重定向s0级PC，清空新结果流水级之前的流水级的错误路径结果。针对不同种类的指令预测的信息也有不同，条件分支指令的目标地址由uFTB提供，需要预测其方向；无条件直接跳转指令的目标地址由uFTB提供，不需要做任何特别的结果预测；间接跳转指令的跳转方向不需要预测，但uFTB提供的跳转地址结果并不一定正确，需要预测。

BPU内的高级预测器包括FTB、TAGE-SC、ITTAGE和RAS。其中，FTB负责维护预测块的起始地址，终止地址，所含分支指令PC地址、类型（是否branch、是否jalr、是否jal、是否call、是否return）、基础的方向结果。TAGE-SC是条件分支指令的主预测器，ITTAGE 用于预测间接跳转指令。RAS负责预测return类型的间接跳转指令跳转地址。

预测单元内多种预测器使用了分支预测历史作为预测条件，为提高历史与执行真实轨迹的匹配度，分支预测历史也会随预测结果做推测更新。全局分支历史在BPU顶层使用多个更新源进行更新维护，维护时会按照TAGE、SC和ITTAGE预测需要的长度进行维护，不同分支历史长度的分支历史维护算法一致。具体地，TAGE使用的分支历史长度有8、13、32、119；ITTAGE使用的历史长度有4，8，13，16，32；SC使用的历史长度有0，4，10，16。分支预测历史在BPU模块顶层统一维护，更新源按照优先级从低到高依次为s0阻塞暂存的分支历史、利用s1预测结果更新的分支历史、利用s2预测结果更新的分支历史、利用s3预测结果更新的分支历史和BPU外部重定向的分支历史。每个分支历史的具体维护策略为：

s0阻塞暂存的分支历史：不进行任何主动的更新，始终与最新全局折叠历史保持一致。

s1预测结果更新的分支历史：利用s1阶段的分支预测结果在随流水线传递来的

s0全局折叠历史上更新。具体地，将预测结果根据位于的分支预测slot shift进全局分支历史，在0号slot则直接shift进去，在1号slot则shift进去0（slot 0没有taken）和当前预测结果。

s2预测结果更新的分支历史：利用s2阶段的分支预测结果在随流水线传递来的

s1全局折叠历史上更新。更新算法与s1相同。s2的更新仅在s2预测结果与之前s1不同时生效。

s3更新策略与s1、s2相同，更新仅在s3结果与之前s1或s2不同时生效。

重定向的分支历史更新仅在发生重定向时进行，根据重定向信息中addIntoHist信号情况，分别将传回的分支历史直接或添加重定向对应分支指令的方向结果后用于更新BPU的全局分支历史。

为保证较为准确的预测结果，各分支预测器都需要不断使用最新的执行结果训练预测器。具体地，更新的预测块及做出该预测时各预测器内部状态的meta信息将会在FTQ模块内生成并传递回BPU单元供各预测器更新内部状态。

分支预测并不能保证结果正确性，在预测结果与真实状态不符时，需要将状态恢复到使用错误预测而更新的状态之前，主要为分支历史的恢复及预测块起始地址的重定向。

* 1. 功能详述
     1. 分支预测块及meta信息生成
        1. 分支预测块思想

分支预测的目的是对执行流中存在的分支指令跳转方向与目标进行预测以在真实执行当前指令前推测地生成后续取指的PC范围信息以保证指令的连续供应。

一个分支预测块内包含了本分支预测块有效位（BranchPredictionBundle.valid）、起始地址、完整预测结果、FTB项、折叠的分支历史、RAS预测器栈顶等信息。其中，完整预测结果在第一流水级来自uFTB，后续来自FTB等高级预测器，FTB项来自uFTB与FTB读出结果。

完整预测结果内记录了分支指令跳转方向、块内记录的分支指令信息是否有效(slot\_valids，即是否存在该分支指令，一个valid对应一个slot，一个slot对应一条预测块内的分支指令，共计2个slot，其中最后一个slot可能记录块内第二条分支指令或一条无条件跳转/间接跳转指令)、分支指令目标、jalr指令目标、分支指令块内偏移、无跳转时指令块结束地址、结束地址是否有误（块起始地址大于结束地址，表明存在false hit）、最后一条分支指令类型、最后一条指令是否为RVI的call指令、第二个分支指令slot是否记录分支指令而非无条件/间接跳转指令、是否命中等信息。如前所属，分支预测块内最多出现2条分支指令/1条分支指令+1条无条件跳转指令，当预测块内实际记录指令数超出该限制时，后续FTQ模块会将其拆分。此外，若分支预测块内不存在分支指令或未超出分支指令数量限制但达到了分支预测块的最大宽度（32B）也将被截断。

FTB项内记录了项是否有效（FTB采用直接映射方式，若读地址不曾被写入则该标记无效）、第一个分支指令slot、结尾的分支/跳转共用slot、结束地址、指令类型、最后一条指令为RVI call指令、是否总是跳转等信息。FTB项可被用于生成完整预测结果。

* + - 1. 分支预测块生成

分支预测块中基础的PC信息最初由模块外传入的复位地址指定，随后处理器运行过程中正常情况下不断按照预测块的跳转地址推测更新，在遇到误预测时，PC值将依据redirect通道所给值更新。next line的完整预测结果由uFTB模块读出。完整预测结果中，FTB项由FTQ模块根据以往训练结果生成，条件分支指令的跳转方向由uFTB生成并在后续由TAGE-SC预测器更新，间接跳转指令的跳转地址由uFTB生成并在后续由ITTAGE预测器更新，RAS预测器会针对return类型的间接跳转指令覆盖ITTAGE的预测结果。被覆盖的结果仅体现在预测器输出，不会立刻反向反馈给结果被覆盖的预测器更新内部状态。

* + - 1. meta信息生成

各预测器为便于自身的更新，会将做出预测时预测器内部状态信息（例如作出该预测时命中预测表序号、命中index）作为meta信息和预测结果一同随流水线传递。

* + 1. 无空泡简单预测

uFTB作为BPU的next line predictor，为处理器作出无空泡的基础预测以连续生成下一个推测PC值。

* + - 1. uFTB请求接收

每次1阶段请求有效时，截取传入预测块起始PC的16到1位生成tag发送给本模块内的全相连uFTB用于读取FTB项，FTB项记录内容如前所述。uFTB内有32项使用寄存器搭建的全相连结构。由于使用寄存器实现，各项可在当拍根据其中存储数据是否有效及存储的tag值是否与传入信息匹配来生成本项是否命中的信号及读出的FTB项数据并返回到uFTB层次。

* + - 1. uFTB数据读取与返回

在当拍，uFTB存储体已返回命中信号及读出数据。在本阶段将会从返回的命中信号中选出至多一命中项并利用该命中项生成预测结果，生成完整预测结果的算法在后续FTB模块有详细叙述，这里uFTB有一额外补充的counter机制，为uFTB内每一项内至多2条分支指令增加一个2位宽counter，若counter大于1或FTB项内always\_taken有效（后者机制也存在于FTB模块中）则预测结果为跳转。此外，本级的命中信号及选出的命中路编号还被作为本预测器的meta信息等待其他预测器一起进入s3阶段时送出预测器，随最终预测结果一起存储到FTQ。本预测器在2、3阶段没有其他额外动作。

* + - 1. uFTB数据更新

当该预测块对应指令全部提交时，由FTQ传入BPU一直连到本模块的update通道中将包含FTQ模块根据指令提交信息更新的FTB项。由于全相连uFTB全部采用寄存器搭建存储体，写操作不会影响并行的读操作，传来的更新信息将始终用于更新。在更新通道有效时，在当拍将利用传入的更新pc值生成tag与uFTB内现有各项匹配并生成是否匹配及匹配的路信号。在下一拍，若存在已有匹配，将拉高匹配路的写入信号，否则利用伪LRU替换算法选出一待替换路拉高对应路写入信号，写入数据即为更新的FTB项。

针对每个分支指令的counter维护也在update通道拉高时一并更新，在update通道拉高下一拍，更新FTB项内跳转的分支指令及其之前的分支指令对应的counter。若taken则counter+1，若不taken则counter-1，如达到饱和（0或全1）则维持当前值不变。

伪LRU算法也需要数据更新，其共有两个数据源，其一为作出预测时命中的路编码，其二为uFTB更新时要写入的路编码，若其中任意有效，则利用其信息更新伪LRU状态，当都有效时一拍内使用组合逻辑依次使用两信息更新。

* + 1. 多种高级预测器及覆盖机制

各高级预测器输出的结果将与之前流水级生成并随流水线传递来的结果（uFTB的minimal结果或之后流水级由FTB提供的完成结果）比较，如有不同将以较新结果冲刷流水线。

* + - 1. Composer

Composer是一个用于组合多个预测器的模块。在本项目中，其组合了uFTB、FTB、TAGE-SC、ITTAGE和RAS五个预测器，并对外抽象成了一个三级流水覆盖预测器。Composer中的各个预测器可以通过写自定义寄存器sbpctl来实现开关，可以按需使用预测器。在检测到来自外部的重定向后，Composer会把重定向请求发送给各预测器，以用于恢复推测更新的元素。在预测块所有指令提交后，Composer中的各预测器会进行训练。最终，Composer将三级预测结果输出至Predictor。各预测器的meta信息在本模块拼接在一起传递给FTQ，FTQ返回的训练用meta信息也在本模块内拆分后发送给各模块。

* + - * 1. **起始PC的配置**

Composer的IO接口io\_reset\_vector可以实现起始PC的配置。只需要将期望的起始PC传递给该IO即可。

* + - * 1. **与预测器的连接**

Composer将uFTB、FTB、TAGE-SC、ITTAGE和RAS五个预测器连接起来。共有三个分支预测器的流水级，每个预测器的相同流水级从前向后以组合逻辑连接，且每个预测器是固定延迟的，到那个流水级就一定完成预测，所以Composer的只需要在对应流水级输出对应预测器的预测结果即可。

* + - * 1. **预测器的开关**

通过Zicsr指令，我们可以读写sbpctl这一自定义CSR来控制Composer中的各预测器的使能。sbpctl[6:0]代表了{LOOP, RAS, SC, TAGE, BIM, BTB, uFTB}这七个预测器的使能。其中，高电平代表使能，低电平代表未使能。具体地，spbctl这一CSR的值通过Composer的IO接口io\_ctrl\_\*传入各个预测器，并由各预测器负责使能的实现。当前架构中未加入Loop和BIM两预测器，因此对应位无效。

* + - * 1. **重定向的恢复**

Composer通过io\_s2\_redirect、io\_s3\_redirect和io\_redirect\_\*等IO端口接收重定向请求。这些请求被发送给其各个预测器，用于恢复推测更新的元素，如RAS栈顶项等。

* + - 1. FTB

FTB暂存FTB项，为后续的TAGE、ITTAGE、SC、RAS等高级预测器提供更为精确的分支指令位置、类型、目标地址等分支预测块关键信息，也为总是跳转的分支指令提供基础的方向预测。FTB模块内有一FTBBank模块负责FTB项的实际存储，模块内使用了一块多路SRAM作为存储器。SRAM规格格式详见后续。

1. **请求接收**

0阶段时，FTB模块向内部FTBBank发送读请求，其请求pc值为s0传入的PC。

1. **数据读取与返回**

在发送请求的下一拍也就是预测器的1阶段，将暂存从FTB SRAM中读出的多路信号。

再下一拍也就是预测器的2阶段，从暂存数据中根据各路的tag和实际请求时由PC高位生成tag的匹配情况生成命中信号并在命中时选出命中FTB数据。若存在hit请求，则返回值为选出的FTB项及命中的路信息，若未hit，则输出数据无意义。

FTBBank模块读出的数据在FTB模块内作为2阶段的预测结果传递给BPU后续预测器的s2阶段以获取分支指令类型、PC信息，此外这一读出的结果还会被暂存到FTB模块内，在3阶段作为预测结果以组合逻辑传递给后续预测器。若FTB命中，则读出的命中路编号与命中信息、周期数等也会随流水线向后传递，最终若该预测块未被流水线中途冲刷，则在s3作为meta信息传递给后续FTQ模块，其中周期数仅在仿真环境用于性能统计，在FPGA等环境不存在。

此外，若FTB项内记录的有效分支指令存在always taken标志，表示该分支指令历史上不曾有非跳转情况，则2阶段的预测结果中对应br\_taken\_mask也在本模块内直接拉高处理，直接预测该分支指令跳转，不再使用其他高级预测器的预测结果。

* + - 1. TAGE-SC

TAGE-SC 是南湖架构条件分支的主预测器，属于精确预测器（Accurate Predictor，简称 APD）。

其中 TAGE 利用历史长度不同的多个预测表，可以挖掘极长的分支历史信息；SC 是统计校正器。

TAGE 由一个基预测表和多个历史表组成，基预测表用 PC 索引，而历史表用 PC 和一定长度的分支历史折叠后的结果异或索引，不同历史表使用的分支历史长度不同。在预测时，还会用 PC 和每个历史表对应的分支历史的另一种折叠结果异或计算 tag，与表中读出的 tag 进行匹配，如果匹配成功则该表命中。最终的结果取决于命中的历史长度最长的预测表的结果。

当 SC 认为 TAGE 有较大的概率误预测时，它会反转最终的预测结果。

在南湖架构中，每次预测最多同时预测 2 条条件分支指令。在访问 TAGE 的各个历史表时，用预测块的起始地址作为 PC，同时取出两个预测结果，它们所用的分支历史也是相同的。

1. **TAGE：预测时序**

TAGE 是高精度条件分支方向预测器。使用不同长度的分支历史和当前 PC 值寻址多个 SRAM 表，当在多个表中出现命中时，优先选择最命中的历史长度最长的对应表项的预测结果作为最终结果。

TAGE需要 2 拍延迟：

● 0 拍生成 SRAM 寻址用 index。index的生成过程就是把折叠历史和pc异或，折叠历史的管理不在ITTAGE和TAGE内部，而在BPU里

● 1 拍读出结果

● 2 拍输出预测结果

1. **TAGE：折叠历史**

TAGE类预测器的每一个历史表都有一个特定的历史长度，为了与PC异或后进行历史表的索引，很长的分支历史序列需要被分成很多段，然后全部异或起来。每一段的长度一般等于历史表深度的对数。由于异或的次数一般较多，为了避免预测路径上多级异或的时延，我们会直接存储折叠后的历史。由于不同长度历史折叠方式不同 ，所需折叠历史的份数等于 (历史长度,折叠后长度) 元组去重后的个数。在更新一位历史时只需要把折叠前的最老的那一位和最新的一位异或到相应的位置，再做一个移位操作即可。

1. **TAGE：备选预测逻辑**

实现了USE\_ALT\_ON\_NA寄存器，动态决定是否在最长历史匹配结果信心不足时使用备选预测。在实现中处于时序考虑，始终用基预测表的结果作为备选预测，这带来的准确率损失很小。

1. **SC：时序**

一些应用上，一些分支行为与分支历史或路径相关性较弱，表现出一个统计上的预测偏向性。对于这些分支，相比TAGE，使用计数器捕捉统计偏向的方法更为有效。TAGE在预测非常相关的分支时非常有效，TAGE 未能预测有统计偏向的分支，例如只对一个方向有小偏差，但与历史路径没有强相关性的分支。

统计校正的目的是检测不太可靠的预测并将其恢复，来自TAGE的预测以及分支信息（地址、全局历史、全局路径、局部历史）被呈现给统计校正预测器，其决定是否反转预测。SC 负责预测具有统计偏向的条件分支指令并在该情形下反转 TAGE 预测器的结果。

SC的预测算法依赖TAGE里面的是否有历史表hit的信号provided，以及provider的预测结果taken，从而来决定SC自己的预测。provided是使用SC预测的必要条件之一，provider的taken作为choose bit，选出SC最终的预测，这是因为SC在TAGE 预测结果不同的场景下可能有不同的预测。

SC需要3拍延迟：

● 0 拍生成寻址 index得到s0\_idx，index的生成过程就是把折叠历史和pc异或，折叠历史的管理不在ITTAGE和TAGE内部，而在BPU里

● 1 拍读出SCTable对应s0\_idx的计数器数据 s1\_scResps

● 2 拍根据s1\_scResps选择是否需要反转预测结果

● 3 拍输出完整的预测结果

* + - 1. ITTAGE

ITTAGE接收来自BPU内部的预测请求，其内部由一个基预测表和多个历史表组成，每个表项中都有一个用于存储间接跳转指令目标地址的字段。基预测表用PC索引，而历史表用PC和一定长度的分支历史折叠后的结果异或索引，不同历史表使用的分支历史长度不同。在预测时，还会用PC和每个历史表对应的分支历史的另一种折叠结果异或计算tag，与表中读出的tag进行匹配，如果匹配成功则该表命中。最终的结果取决于命中的历史长度最长的预测表的结果。最终，ITTAGE将预测结果输出至composer。

1. **间接跳转指令的预测**

ITTAGE 用于预测间接跳转指令。普通分支指令和无条件跳转指令的跳转目标直接编码于指令中，便于预测，而间接跳转指令的跳转地址来自运行时可变的寄存器，从而有多种可能选择，需要根据分支历史对其作出预测。为此，ITTAGE 的每个表项在 TAGE 表项的基础上加入了所预测的跳转地址项，最后输出结果为选出的命中预测跳转地址而非选出的跳转方向。由于每个 FTB 项仅存储至多一条间接跳转指令信息，ITTAGE 预测器每周期也最多预测一条间接跳转指令的目标地址。

ITTAGE 需要 3 拍延迟：

● 0 拍生成寻址 index

● 1 拍读出数据

● 2 拍选出命中结果

● 3 拍输出

1. **折叠分支历史**

历史表有特定的历史长度，为了与 PC 异或后进行历史表的索引，很长的分支历史序列需要被分成很多段，然后全部异或起来。每一段的长度一般等于历史表深度的对数。由于异或的次数一般较多，为了避免预测路径上多级异或的时延，我们会直接存储折叠后的历史。由于不同长度历史折叠方式不同，所需折叠历史的份数等于 (历史长度,折叠后长度) 元组去重后的个数。在更新一位历史时只需要把折叠前的最老的那一位和最新的一位异或到相应的位置，再做一个移位操作即可。

* + - 1. RAS

RAS使用栈结构来预测函数调用与返回这类具有成对匹配特性的执行流的返回地址。其中调用（push/call）类指令的特征为目标寄存器地址为1或5的jal/jalr指令。返回（ret）类指令的特征为源寄存器为1或5的jalr指令。这类指令为无条件跳转指令，其类型、所在块内偏移量已在FTB中读出。

在实现中，RAS预测器在s2和s3两阶段提供预测结果。

1. **2阶段结果**

在2阶段，由于s3阶段还可能存在预测结果需要更新，当前FTB项并不一定为最终执行路径，此时作出的预测推测了此时3阶段预测结果（也即前一个预测块）不会刷新当前流水级内的预测起始地址。若2阶段从FTB传来的FTB项有效且其中存在push类（call）指令，则将该指令之后下一指令的PC值压入RAS栈；若s2阶段传来FTB项有效且其中存在pop类（return）指令，则将当前栈顶的地址作为结果返回并对结果出栈。

在RAS栈模块内，上述行为分别体现为，在push操作时，如当前地址和栈顶地址不同，则压栈一个新项目，其对应计数器为0，否则将栈顶项的计数器增加1。两操作都需将这一顶部信息设置为写bypass项以供当前读操作使用。在pop操作时，若当前栈顶项计数器为0，则栈顶指针减1，若计数器大于0，则将计数器减1。为时序优化考虑，写入到RAS栈内的数据将会延迟一拍后写入，考虑到可能存在本拍写入的数据下一拍需要获取读数据的情况，设计了写bypass机制，准备写入的数据将在本拍首先用于更新写bypass相关的项，包括写操作指针及写操作数据。下一拍要求读取的指针若与写bypass记录的指针位置匹配，则使用bypass值，否则使用真正的栈顶值。

1. **3阶段结果**

在3阶段，2阶段曾发生过的推测push/pop操作记录会随流水线传递过来，3阶段会根据3阶段FTB项（此时不再存在后续流水级，也即能进入3阶段的FTB项不会被后续流水级冲刷）结果以与2阶段相同逻辑生成push/pop控制信号，若发现2阶段推测结果与3阶段判定结果不同，即RAS位于2阶段时，当时3阶段预测冲刷过BPU流水线，则2阶段做出的预测结果所基于的情况已经发生变化，对RAS栈的操作不正确，需要仿照误预测进行状态恢复，具体细节见后。

* + 1. 预测器训练

预测器作出的每一个预测，在其中所有指令都成功提交后会由FTQ生成预测块更新信息，与传递到FTQ的各预测器meta信息一起送回预测器进行训练。

* + - 1. 预测器训练数据接收

从FTQ传入的预测器训练数据在BPU模FTB项、更新PC等其他信号一起传递给各预测器。各预测器视其时序压力再暂存update信号或立刻进行

* + - 1. FTB

FTB项的更新具体逻辑详见FTQ模块。

收到update请求后，FTB模块会根据meta信息中记录的这一预测做出时原来的读取结果是否hit决定更新时机。若meta中显示做出预测时hit，则在本拍立刻更新将新的FTB数据写入SRAM，否则需要延迟2周期等待读出FTB内现有结果决定写入路后才可更新。

在FTBBank内部，当存在更新请求时，该模块行为也因立即更新和推迟更新两情况而有所不同。立即更新时，FTBBank内的SRAM写通道拉高，按照给定的信息完成写入。推迟2周期更新时，FTBBank首先收到一个update的读请求且优先级高于普通预测的读请求，而后下一拍读出数据，选出给定地址命中的路编码传递给外部FTB模块(命中场景：两次针对这一FTB项的请求，第一次请求未命中，它更新之前发生了第二次访问，当前第一次的更新已完成，此更新为第二次对应的更新)。而若这一拍未命中，则下一拍需要写入到在读出FTB项后一拍由路选取算法分配的路中。路选取规则为，若所有路均已写满，则使用替换算法（此处为伪LRU，详见ICache文档）选取要替换的路，否则选取一空路。

* + - 1. Composer

Composer通过IO端口io\_update\_\*将训练信号发送给其各个预测器。总的来说，为防止错误执行路径对预测器内容的污染，各部分预测器在预测块的所有指令提交后进行训练。它们的训练内容来自自身的预测信息和预测块中指令的译码结果和执行结果，它们会被从FTQ中读出，并送回BPU。其中，自身的预测信息会在预测后打包传进FTQ中存储；指令的译码结果来自IFU的预译码模块，在取到指令后写回FTQ；而执行结果来自各个执行单元。

* + - 1. TAGE-SC & ITTAGE

表项中包含一个useful域，它的值不为0表示该项是一个有用的项，便不会被训练时的分配算法当作空项分配出去。在训练时，用一个饱和计数器动态监测分配的成功/失败次数，当分配失败的次数足够多，计数器达到饱和时，把所有的useful域清零。

* + - 1. RAS

当RAS在2阶段的推测结果与3阶段不同或之前的预测结果遇到了redirect，需要恢复状态。其中，redirect信息在实际恢复前被暂存到RAS内寄存器，延迟一拍再更新。3阶段检测到不同的下一拍完成更新。若为redirect中的call误预测（出错指令预译码为call类型指令）或3阶段的push操作不匹配，则进行recover\_push操作，将原来错误pop的RAS栈顶重新push进去。若为redirect中的ret误预测（出错指令预译码为pop指令）或3阶段的pop操作不匹配，则进行recover\_pop操作，弹出RAS栈顶本应弹出的地址。栈指针在redirect时恢复为redirect传来的栈指针，否则为当前值。栈顶在redirect恢复时恢复为redirect传来的栈顶项，否则为当前值，恢复的新地址在redirect时为redirect信号下一条指令的值，否则为2阶段推测值。

在RAS栈内部，状态恢复时同样生成push、pop等操作，这类操作的处理方式与前述2阶段相同，此处仅列举存在不同的情况。在push操作且非分配新项的情况下，若处于recover状态，sp、栈顶指针、顶部返回地址要设置为更新值。在pop操作且当前栈顶计数器非0时，so、栈顶指针、顶部返回地址要设置为更新值。在既非push也非pop时，需要恢复sp、栈顶指针、顶部返回地址同时处理写bypass。

* + 1. 分支历史信息维护
       1. 推测更新

在流水级中预测器生成推测结果后，后续请求所用的分支历史也将包含这一推测值以提高预测准确率。

* + - 1. redirect恢复

在遇到分支预测错误时，分支历史也被一并恢复到出错状态前，这样可以保证分支历史的准确度。

* + 1. topdown性能分析事件统计

在BPU流水级中预测器生成推测结果可能因为各种原因阻塞，而阻塞可能最终导致处理器整体的流水线空泡。为对性能瓶颈进行较为准确的分析定位，昆明湖架构增加了topdown性能计数器，收集流水线中各流水级的空泡/阻塞信息并将指令提交时空泡（实际提交指令数与发射数理想值间差值）归因到具体的模块，从而实现对瓶颈部件的定位。上述性能分析建模方法细节详见Intel发表的论文《A Top-Down method for performance analysis and counters architecture》。在BPU可以统计到的阻塞事件包括因各预测器误预测恢复所引入的流水线空泡、因后端访存违例恢复所引入的流水线空泡、因BPU内部override预测刷新较老预测结果所引入的流水线空泡、因分支指令训练预测器而阻塞BPU所引入的流水线空泡和因FTQ满无法接收新的分支预测块阻塞BPU所引入的流水线空泡。在BPU内不处理各空泡原因间的优先级，而只是在符合对应的统计条件时将空泡控制信号拉高并随处理器流水线传递。

目前BPU内有FTB（含uFTB和主FTB）、TAGE、SC、ITTAGE和RAS共计5种预测器。Topdown将分支误预测原因细分到以上5个预测器。具体地，将每个误预测空泡分解到各预测器的条件为：

1. FTB：发生了分支指令相关的重定向且误预测的指令在对应预测块的FTB内并没有记录
2. TAGE：发生了分支指令相关的重定向且误预测的指令在对应预测块的FTB内有记录，但SC预测器并未给出对应的预测
3. SC：发生了分支指令相关的重定向且误预测的指令在对应预测块的FTB内有记录，同时SC预测器给出了对应的预测结果
4. ITTAGE：发生了分支指令相关的重定向，误预测的指令为jalr指令但不是return指令且在FTB项中命中
5. RAS：发生了分支指令相关的重定向。误预测指令为return指令（一类特殊的jalr指令，详见后续RAS模块说明）。

后端访存违例恢复所引入的流水线空泡判断条件为后端发来的重定向信号指示重定向来自访存违例。

BPU内部override引入的流水线空泡有两个可能的来源，分别为BPU第2和第3流水级的重定向信号。

因分支预测器训练而引入的流水线空泡有3个可能的来源，分别为BPU第1，2和3流水线的ready信号。上述ready信号为BPU内各预测器的ready信号取或操作的结果。

因FTQ满而引入的流水线空泡判断条件为BPU发往FTQ模块的握手接口ready信号拉低。

对分支误预测和访存违例所导致的空泡，其将被标记在当前BPU各流水级的topdown信号中。对BPU内部overrride引入的空泡，其将被标记在override当前所在流水级和更早流水级，而不影响比这一override更早的预测块所在流水级。对分支预测器训练所导致的空泡处理类似BPU override。

1. 总体设计
   1. 整体框图

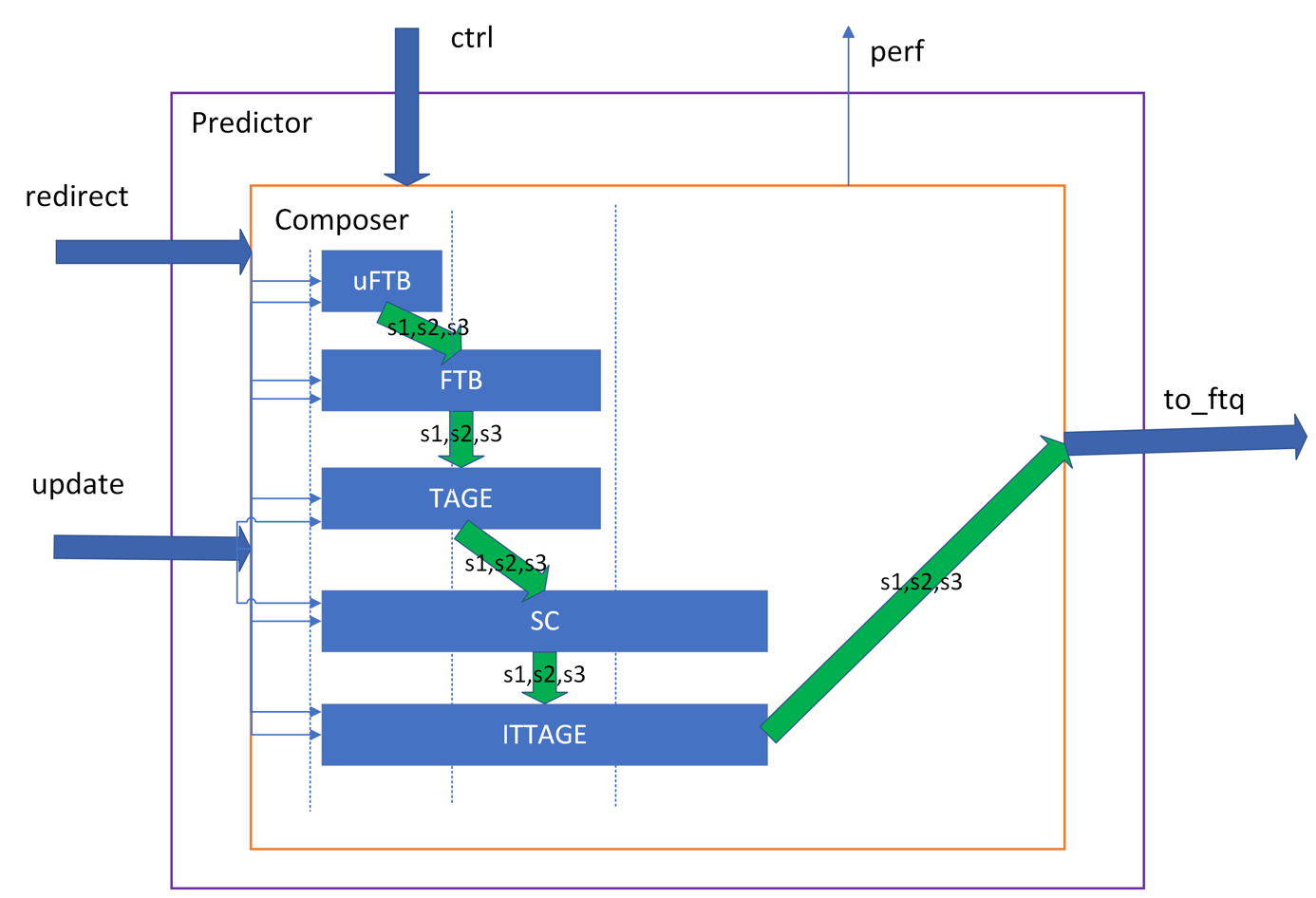
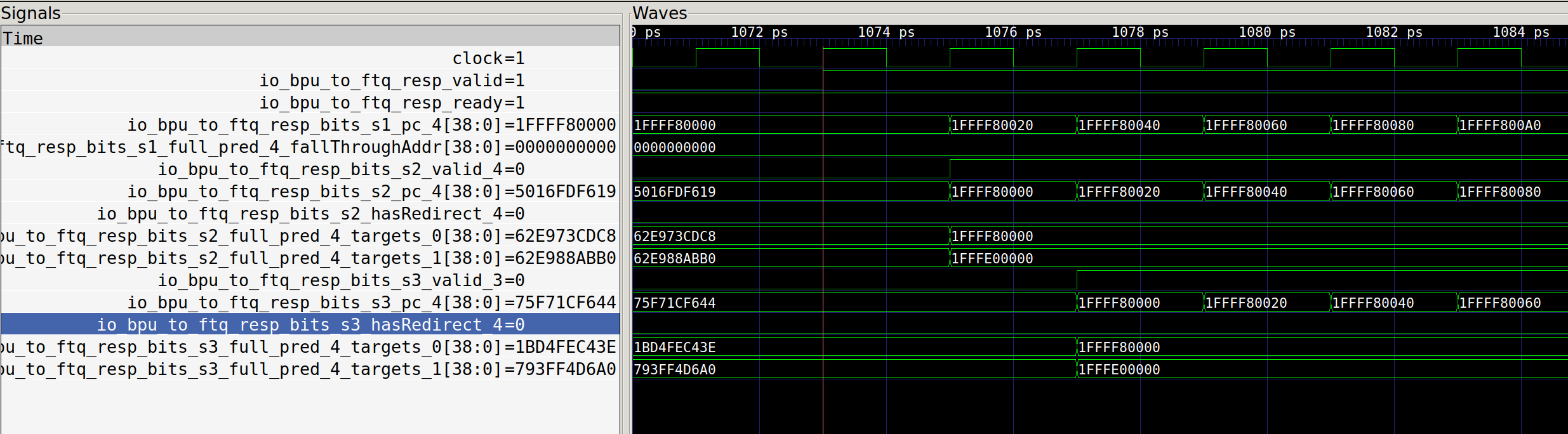


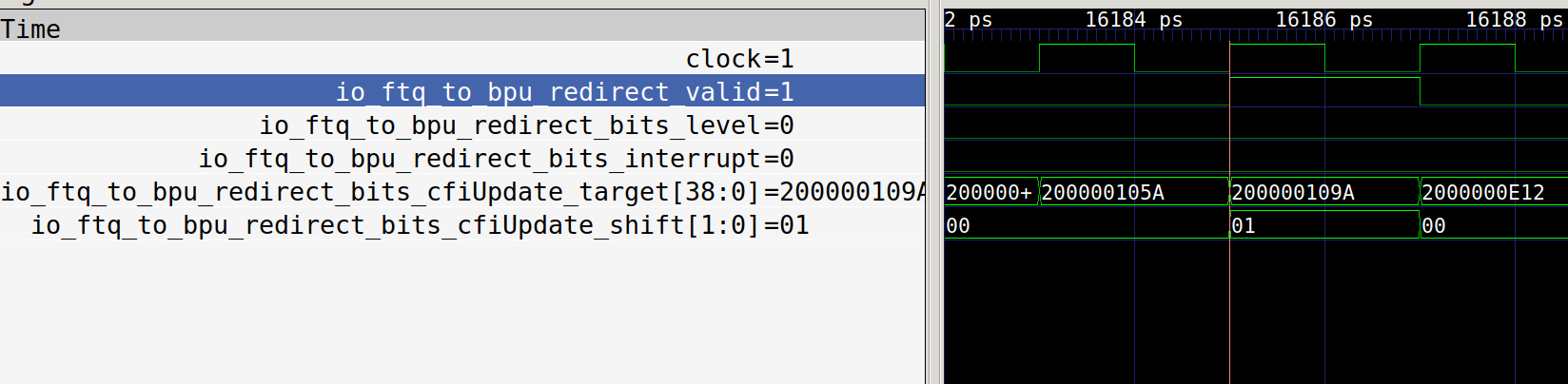
图4.1 BPU模块整体框图

* 1. 接口时序
     1. BPU到FTQ接口时序



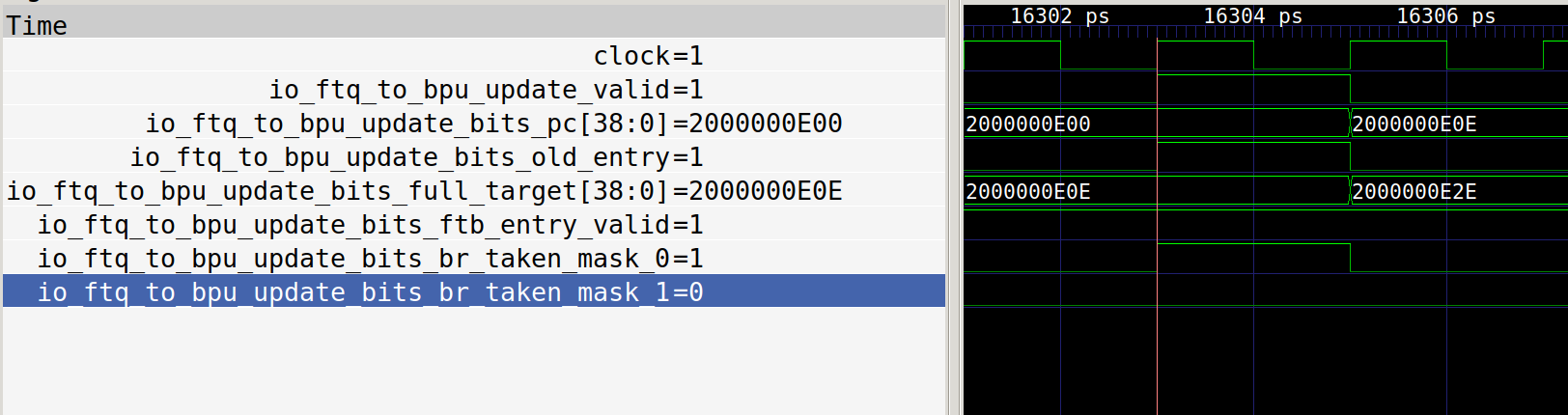
上图展示了BPU到FTQ的预测结果接口时序。图中针对0x1FFFF80020起始地址的预测结果在流水线内1、2、3阶段分别输出，若结果与之前流水级不一致则redirect信号拉高表明需要刷新预测流水线。

* + 1. FTQ到BPU redirect接口时序



上图展示了FTQ到BPU的redirect接口时序，redirect核心信号为cfiUpdate\_target，其指定了redirect的目标地址为0x200000109a。

* + 1. FTQ到BPU update接口时序



上图展示了FTQ到BPU的update接口时序，这一更新是为0x2000000e00开始预测块准备的，其目标跳转地址为0x200000e0e。

* 1. 时钟复位

|  |  |  |
| --- | --- | --- |
| Module | Clock | Reset |
| BPU | clock | reset |

* 1. 寄存器配置

表4.2 BPU寄存器说明

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **寄存器** | **地址** | **复位值** | **属性** | **描述** |
| sbpctl | 0x5C0 | 64’d0 | RW | bit63-7:  bit0: uFTB使能信号  bit1: FTB使能信号  bit2: BIM使能信号（保留）  bit3: TAGE使能信号  bit4: SC使能信号  bit5: RAS使能信号  bit6： loop预测器使能信号（保留） |

注：RO——只读寄存器；RW——可读可写寄存器。

* 1. 补充说明

无。

1. 模块设计

FTB：

单bank，512 set，4 way，使用单口SRAM，无读保持，有上电复位。

20 bit tag，60 bit FTB项。

其中FTB项

1 bit valid

20 bit br slot（4 bit offset，12 bit lower 2 bit tarStat, 1bit sharing, 1 bit valid）

28 bit tail slot (4 bit offset , 20 bit lower, 2 bit tarStat, 1 bit sharing, 1 bit valid)

4 bit pftAddr

1 bit carry

1 bit isCall

1 bit isRet

1 bit isJalr

1 bit 末尾可能为rvi call

2 bit always taken

* 1. Composer
     1. 功能概述

Composer是一个用于组合多个预测器的模块。在南湖中，其组合了uFTB、FTB、TAGE-SC、ITTAGE和RAS五个预测器，并对外抽象成了一个三级流水覆盖预测器。Composer中的各个预测器可以通过写自定义寄存器sbpctl来实现开关，可以按需使用预测器。在检测到来自外部的重定向后，Composer会把重定向请求发送给各预测器，以用于恢复推测更新的元素。在预测块所有指令提交后，Composer中的各预测器会进行训练。最终，Composer将三级预测结果输出至Predictor。

三级BPU流水级内部重定向的时候如果有预测错误只会恢复那些推测更新的状态，比如说分支历史和RAS，其它的预测器更新都是在提交后做。

此时如果不刷新预测器，只是刷新流水线，那下次这个地方不是还会预测错误？刷新流水线的同时是从纠正过的正确路径开始预测，如果接下来的路径又经过了同一个地方，是可能再次预测同一个结果的，但是也有可能因为分支历史不同，从而在TAGE等预测器里索引不同的表项。

如果在执行时发现目标地址错误，不会发起重定向，而是统一等到指令提交时再重定向。这样设计的一个原因是本身误预测的重定向就是在错误路径上的，它的执行结果可能也是错误的，这种情况下去训练，可能对预测器造成污染。

* + - 1. 起始PC的配置

Composer的IO接口io\_reset\_vector可以实现起始PC的配置。只需要将期望的起始PC传递给该IO即可。

* + - 1. 与预测器的连接

Composer将uFTB、FTB、TAGE-SC、ITTAGE和RAS五个预测器连接起来。因为共有三个分支预测器的流水级，且每个预测器是固定延迟的，到那个流水级就一定完成预测，所以Composer只需要在对应流水级输出对应预测器的预测结果即可。

meta是预测器预测的时候的数据，update的时候拿回来更新用。都叫meta是因为composer将所有预测器整合起来，用共同的接口meta和外界交互。

* + - 1. 预测器的开关

通过Zicsr指令，我们可以读写sbpctl这一自定义CSR来控制Composer中的各预测器的使能。sbpctl[6:0]代表了{LOOP, RAS, SC, TAGE, BIM, BTB, uFTB}这七个预测器的使能。其中，高电平代表使能，低电平代表未使能。具体地，spbctl这一CSR的值通过Composer的IO接口io\_ctrl\_\*传入各个预测器，并由各预测器负责使能的实现。

* + - 1. 重定向的恢复

Composer通过io\_s2\_redirect、io\_s3\_redirect和io\_redirect\_\*等IO端口接收重定向请求。这些请求被发送给其各个预测器，用于恢复推测更新的元素，如RAS栈顶项等。

* + - 1. 预测器训练

Composer通过IO端口io\_update\_\*将训练信号发送给其各个预测器。总的来说，为防止错误执行路径对预测器内容的污染，各部分预测器在预测块的所有指令提交后进行训练。它们的训练内容来自自身的预测信息和预测块中指令的译码结果和执行结果，它们会被从FTQ中读出，并送回BPU。其中，自身的预测信息会在预测后打包传进FTQ中存储；指令的译码结果来自IFU的预译码模块，在取到指令后写回FTQ；而执行结果来自各个执行单元。

* + 1. 整体框图

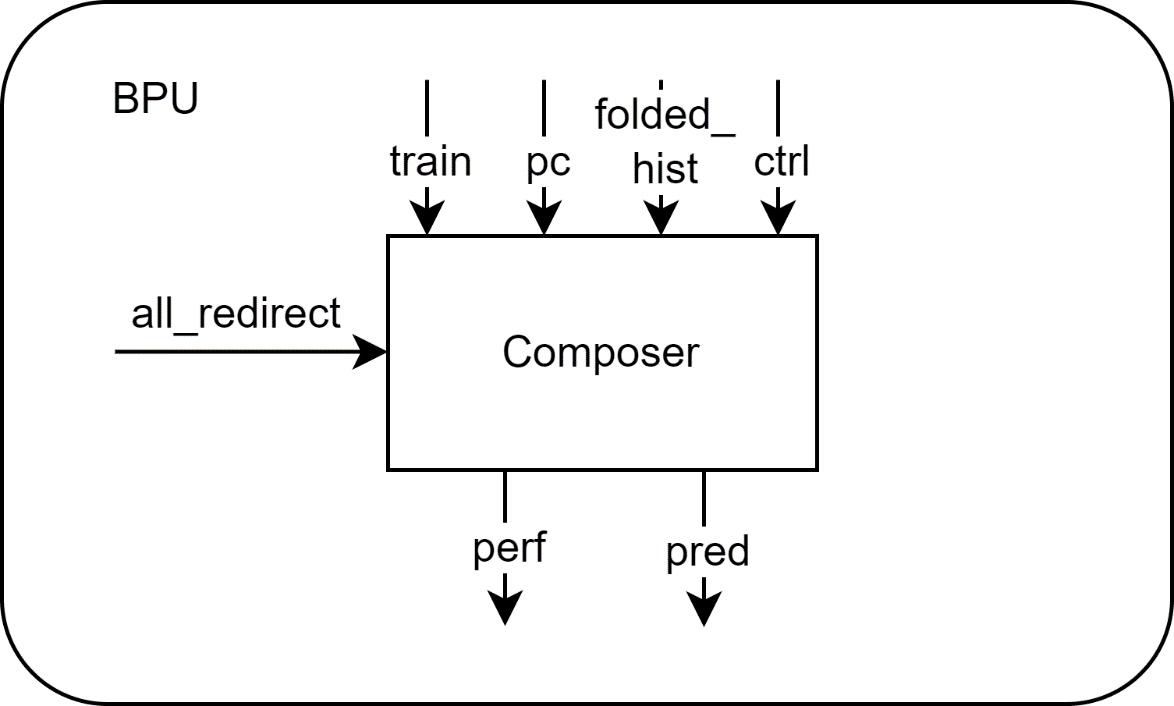
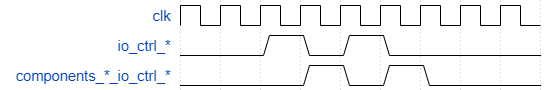


图5.1 Composer模块整体框图

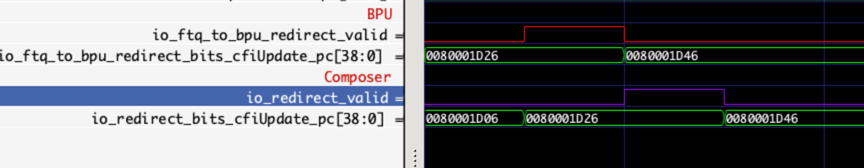
* + 1. 接口时序

1. 控制信号Ctrl接口时序



上图示意了Composer模块控制信号Ctrl接口的时序示例，io\_ctrl信号在传入Composer模块后，被delay一拍传给内部components子模块。

1. 重定向接口时序



上图展示了Composer模块重定向请求的接口，在BPU接收到来自后端的重定向请求后，会延迟一拍发往Composer，因此Composer内预测器会晚一拍收到相应请求。

1. 分支预测块训练接口时序



类似重定向，为优化时序，分支预测块训练的update接口同样在BPU内部被延迟一拍发往Composer及其内部各预测器。

* + 1. 关键电路

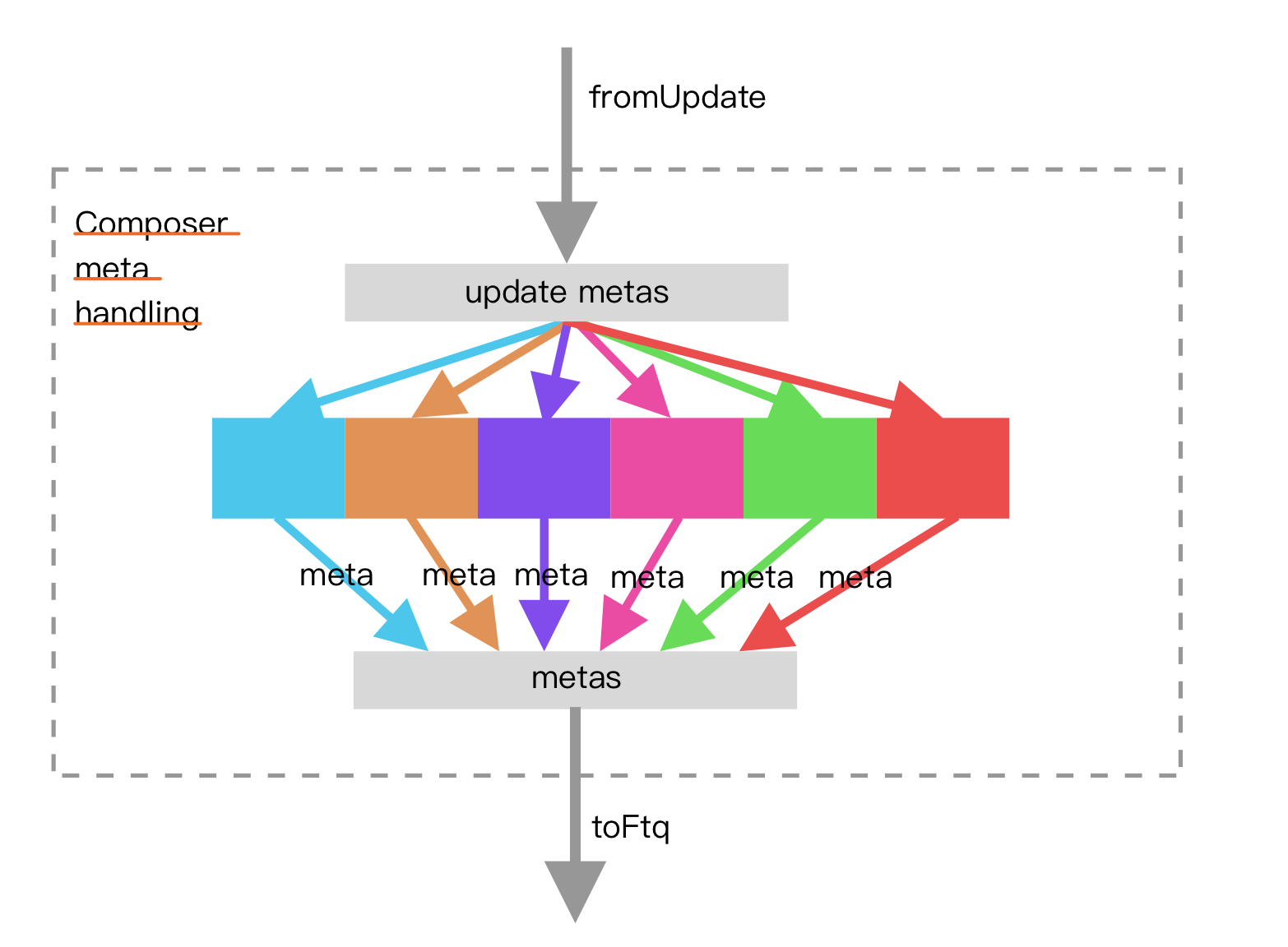


图5.2 Composer meta拼接

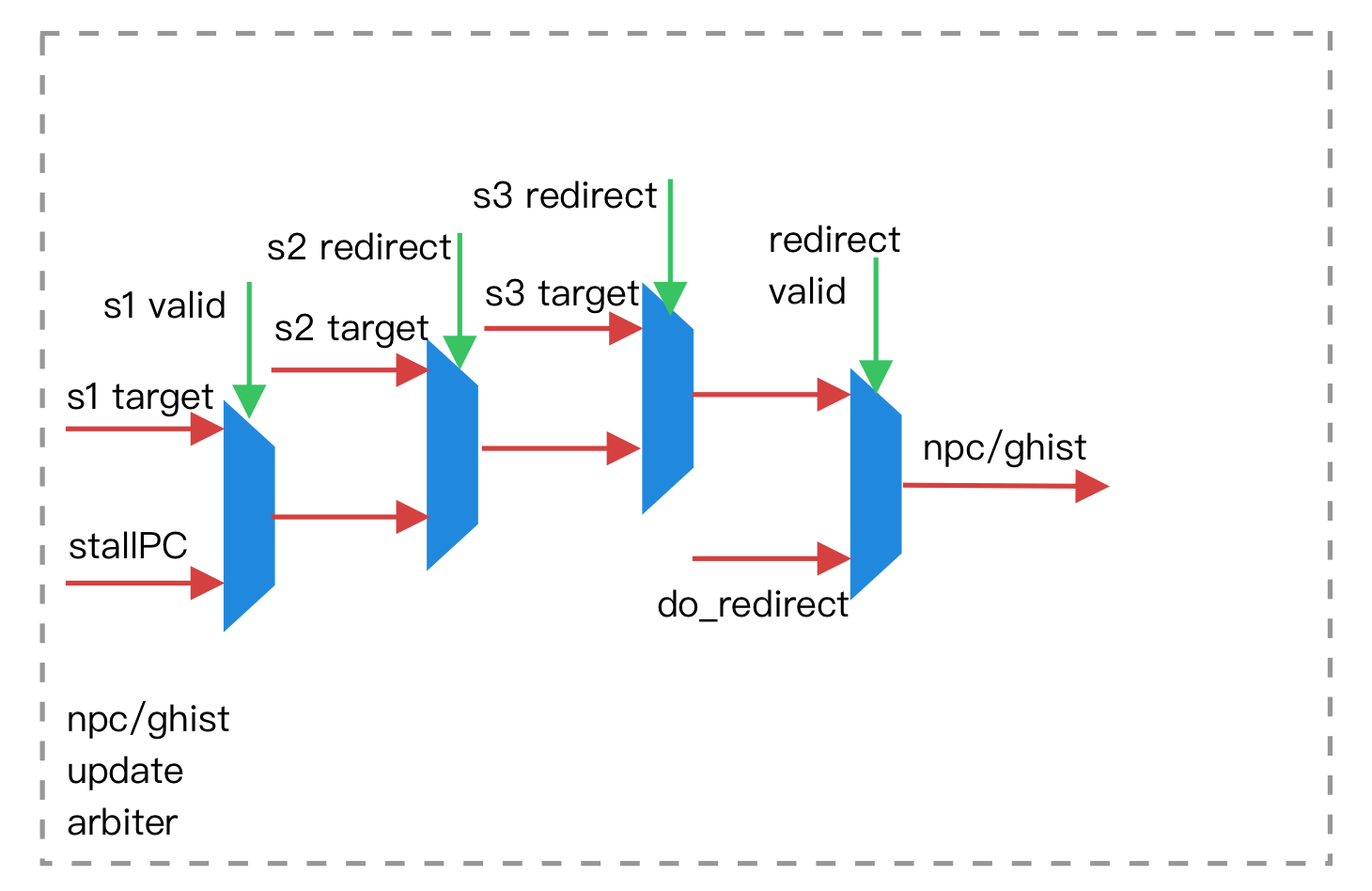


图5.3 重定向/分支历史更新来源仲裁逻辑

* 1. RAS
     1. 功能

RAS预测器使用栈结构来预测函数调用与返回这类具有成对匹配特性的执行流。其中调用（push/call）类指令的特征为类指令的特征为目标寄存器地址为1或5的jal/jalr指令。返回（ret）类指令的特征为源寄存器为1或5的jalr指令。

在实现中，RAS预测器在s2和s3两阶段提供预测结果。

昆明湖架构的RAS预测器与南湖架构差异较大。通过引入持久化队列，新的RAS预测器结构解决了局部预测器因推测执行所导致的预测器数据污染问题。同时，新结构也保留了和南湖架构类似的栈结构作为提交栈，存储提交后的压栈信息，以弥补持久化队列存储密度不高的缺点。

* + - 1. 基于持久化队列设计的RAS预测器架构整体概述

RAS预测器利用分支预测FTB块中的call/return指令局部的配对信息对return指令作出预测。由于正常调用的函数执行完毕会返回到调用指令的下一条指令，RAS预测器可在遇到call指令时根据当前指令PC及当前指令是否为RVC指令（确定指令宽度为2还是4）生成其函数调用返回地址的预测结果并压栈。

由于现代超标量乱序处理器通常采用深流水线推测执行技术，分支预测器会在之前所预测指令执行结果确定前为后续指令生成预测结果，也即，若面临ABC三个连续的分支指令预测请求，RAS预测器并不能在预测B时获取到A块内指令的最终执行结果，而只能获取到A块的Z块内指令的最终执行结果和基于分支预测结果的Z到A块间推测结果。若Z到A块间存在错误的分支预测结果，则需要对其涉及的分支预测器状态进行误预测恢复。如前文所述，RAS预测器基于栈结构对call-return指令对作出预测，配对信息准确性对其准确率至关重要。要精确地恢复RAS预测器在发生错误预测点的状态，最直观的做法是从当前最新预测点回溯到发生误预测点，撤销其间对RAS栈作出的所有改动。而为提升误预测恢复及时性，现代处理器通常难以允许误预测恢复时进行如此高复杂度的操作。在南湖架构设计中，RAS预测器在发生误预测时仅恢复发生误预测预测块对应栈顶项和RAS栈顶指针。这一恢复策略可以应对因推测执行导致的误预测点RAS栈顶及其更上方数据的污染，但无法应对推测执行中因先pop再push对RAS栈顶下方数据的污染。如下图所示，这类污染无法通过仅恢复栈顶项和栈顶指针修复。

这类污染会导致后续的return指令跳转目标出错，从而产生误预测。为解决这一问题，昆明湖架构的新RAS预测器通过引入持久化队列，保存RAS推测执行过程中的所有局部状态，实现了抗上述污染的预测。具体地，持久化队列模拟的栈结构有读指针TOSR、写指针TOSW和栈底指针BOS三个指针，每一项除记录自身数据外，还记录其前一项在持久化队列中的位置指针NOS；在每次栈push操作时均前移一次TOSW为当前压栈数据新分配一项存储空间，并将当前TOSR指向TOSW原位置（即新分配空间），新压栈的项记录的NOS存储push操作前的读指针位置；在每次栈pop操作时，将TOSR移动到当前TOSR指向的项内NOS指针位置。通过上述方式，RAS能够通过前向链表遍历当前版本（对应一个推测执行路径）栈的所有项，这一设计也不会覆盖任何非本版本（对应其他推测执行路径）的RAS栈数据。

为提升RAS有效存储容量，并非所有RAS项均采用持久化队列形式存储。根据实践数据，为满足推测执行路径需要，持久化队列最大约需要28项，因而RTL实现中使用了32项持久化队列。在RAS项对应预测块（即含call指令的预测块）指令提交后，我们可以将该块从持久化队列释放移入提交栈中，释放操作通过改变BOS指针到提交预测块对应的TOSW指针。提交栈采用与南湖架构相同的设计，压栈时增加提交栈指针nsp并将压入数据写入新栈顶；出栈时减少栈指针nsp。由于其仅存放提交后的确定性信息，不存在推测执行污染问题。原BOS到新TOSW这一区间在提交栈中可能存在其他错误路径上的压栈结果，而这些结果可以在这一BOS移动过程中被自然释放。

由于引入了持久化队列和提交栈两个结构，栈顶项可能在二者之一，在提供预测结果时会需要动态判断。持久化队列是一个环形队列，其指针除用于寻址的value外还有一flag位，这一位可协助判定BOS和TOSW、TOSR的位置关系。当TOSR位于BOS之上TOSW之下时，栈顶项位于持久化队列内部；当TOSR位于BOS之下时，栈顶项位于持久化队列外部，即位于提交栈内。因而，我们能够在运行时动态选出栈顶项。注意到，我们从提交栈获取的栈顶项并非始终与已提交指令的栈顶一致，因而，我们需要为RAS预测器维护另一个提交栈指针ssp，其含义为，持久化队列中该项在被压入提交栈后所处的位置。在从提交栈访问栈顶项时利用ssp而非nsp完成数据读取。

上述讨论全部基于持久化队列容量充足而不发生溢出的情况。若当前程序段内push操作过于密集而后端执行速度不够快，可能会出现持久化队列容量不足发生溢出的情况。这一情况有两种可能的处理方案：阻塞分支预测和强行覆盖。当前昆明湖架构选择了后一种实现策略。在当前BOS和TOSW即将重叠时，将BOS强制前进一项来避免持久化队列被意外清空。由于BOS记录项并不一定需要在此期间使用，此策略能够略微减少前端的阻塞。

为时序优化考虑，栈顶项的读取/更新并非在收到读取请求的当拍完成，而是在其上一拍或上N拍根据当拍的压/出栈操作更新。为减少对推测队列的写操作，在BPU 2流水级的push/pop结果也不会直接写入推测队列，而是延迟一拍后写入。考虑到存在本拍写入的数据在下一拍需要获取读数据的情况，设计了写bypass机制，准备写入的数据将在本拍首先用于更新写bypass的writeEntry项。下一拍要求读取的指针若与写bypass记录的指针位置匹配，则使用bypass值，否则使用从栈顶读取的值（实际因时序优化考虑这一逻辑被提前到读取栈顶的上一拍）。

* + - 1. 2阶段结果

在BPU 2阶段，由于其他分支预测器的预测结果还未完全确定，可能存在需要更新的预测结果，当前的预测结果并非最终确定的推测执行路径。当前的预测结果对同一时刻位于3阶段的分支预测块不会改变当前2阶段预测块的起始地址和现在位于call/ret指令前的其他分支指令不会再被预测跳转作出了成立假设。若2阶段从FTB传来的FTB项有效且其中存在push类指令，则将该指令之后下一指令的PC值压入RAS栈；若2阶段从FTB预测器传来的FTB项有效且其中存在pop类指令，则将当前栈顶的地址作为结果返回并对结果出栈。

在RAS内，上述行为具体分解如下：

在2阶段的FTB项内看到预测跳转的call指令，则拉高s2\_spec\_push信号并根据当前call指令pc生成压栈的地址信息，指示内部RASStack模块动作。RASStack模块在检测到push动作时，会利用传入的栈顶地址作为新写入持久化队列entry的预测地址，若新写入地址与原地址相同且原栈顶counter未饱和，则基于原栈顶项counter+1作为新项counter，否则新项counter取0。生成的新entry被用于以下三个用途：1，下一拍更新writeByassEntry寄存器，供连续预测读取栈顶项使用；2，当拍内更新栈顶项，供连续预测读取使用；3，打拍后用于在下下拍更新持久化队列。与此同时，如上所述，TOSR被更新为当前的TOSW，TOSW更新为当前TOSW+1，全局的ssp、sctr类似上述counter更新算法，若新旧栈顶地址相同且原sctr（与原栈顶entry的ctr相同）未饱和，则ssp不变，sctr=sctr+1；否则ssp=ssp+1，sctr=0。为处理可能的持久化队列溢出情形，若此时TOSW新值与BOS相等，则强制BOS=BOS+1。

在2阶段的FTB项内看到预测跳转的call指令，则拉高s2\_spec\_pop信号，指示内部RASStack模块动作。RASStack模块在检测到pop动作时，若当前sctr非0，则sctr=sctr-1，ssp不变；否则ssp=ssp-1，sctr=新栈顶项的sctr。TOSR=原top的NOS，TOSW保持不变。单独维护的栈顶项也会利用更新后的ssp、sctr、TOSR和TOSW来更新。

* + - 1. 3阶段结果

在3阶段的FTB项内看到预测跳转的return指令，则拉高s3\_push信号；在3阶段的FTB项内看到预测跳转的call指令，则拉高s3\_pop信号。当前预测块在2阶段作出的预测结果也分别被打拍到3阶段（s3\_pushed\_in\_s2和s3\_poped\_in\_s2）。若2阶段与3阶段判断要采取的动作不同，则需要在3阶段进行恢复，无论是否恢复，3阶段均使用2阶段读出的RAS栈顶项作为预测结果。

因为2阶段的push/pop和3阶段的push和pop仅存在如下几种情况，因而3阶段可通过push/pop操作撤销2阶段的操作。

|  |  |  |
| --- | --- | --- |
| S2 push | S3 push | No fix needed |
| S2 push | S3 keep | Fix by pop |
| S2 keep | S3 push | Fix by push |
| S2 keep | S3 keep | No fix needed |
| S2 keep | S3 pop | Fix by pop |
| S2 pop | S3 keep | Fix by push |
| S2 pop | S3 pop | No fix needed |

3阶段push/pop操作在RASStack内的具体动作与2阶段相同，此处不再重复。

* + - 1. 误预测状态恢复

在预测块离开BPU后，在IFU或后端执行时可能会触发重定向，在遇到重定向时，RAS预测器需要进行状态恢复。具体地，RAS的TOSR、TOSW、ssp和sctr都需要根据误预测发生前的对应meta信息恢复，随后，根据误预测的指令自身是否为call/return指令，还需要分别通过push和pop操作来调整栈结构。Push和pop操作的具体动作同2、3预测流水级，此处略。

* + - 1. 提交entry迁移

如上所述，在含call指令的预测块提交时，BOS会被更新为预测时的TOSW，同时将预测块对应的entry写入commit stack栈顶（以nsp寻址）并更新nsp。Nsp更新算法类似ssp，若存在递归且栈顶counter未满则counter=counter+1，nsp不变，否则nsp=nsp+1，counter=0。

* + 1. 整体框图

*要求如4.1*

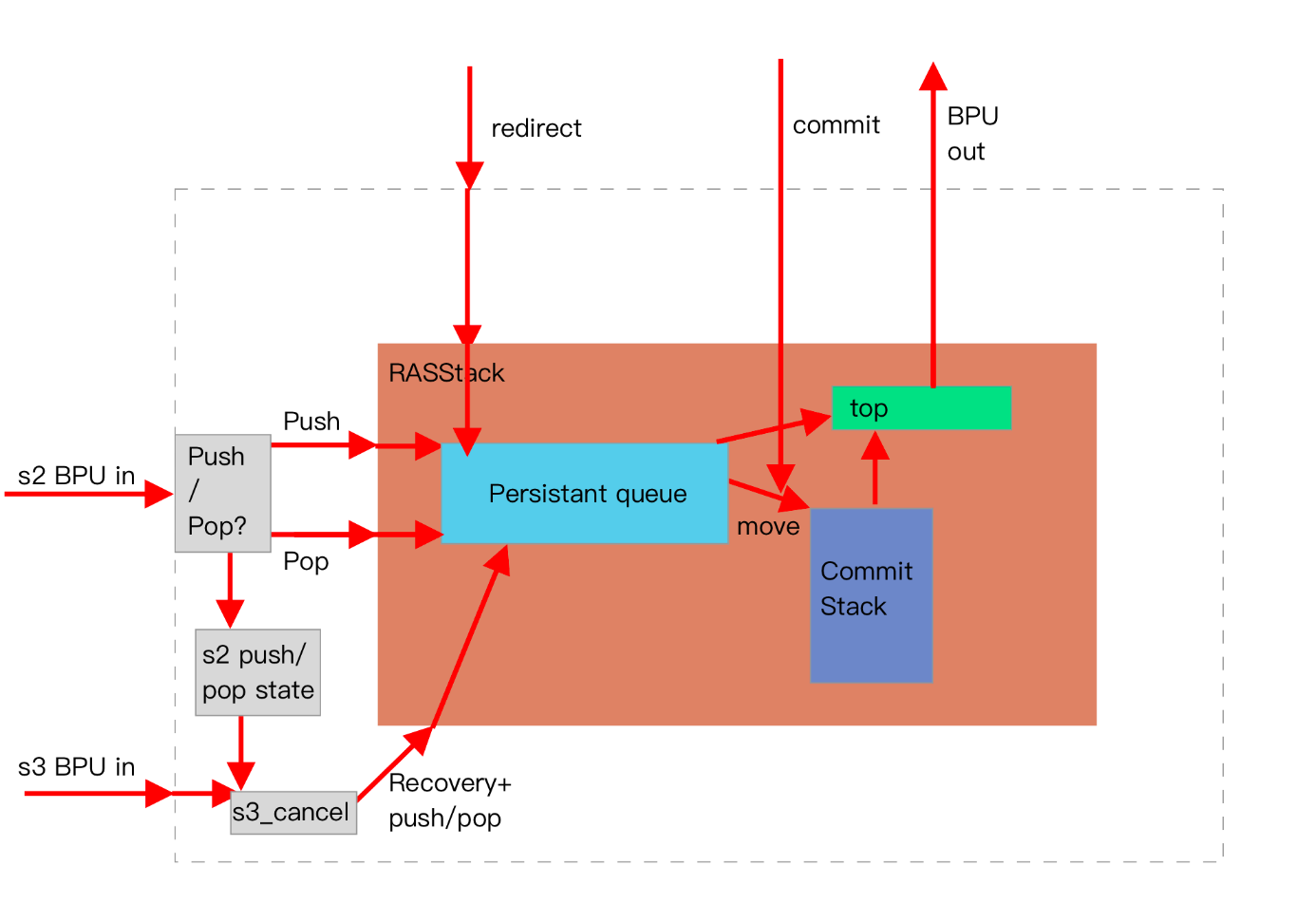
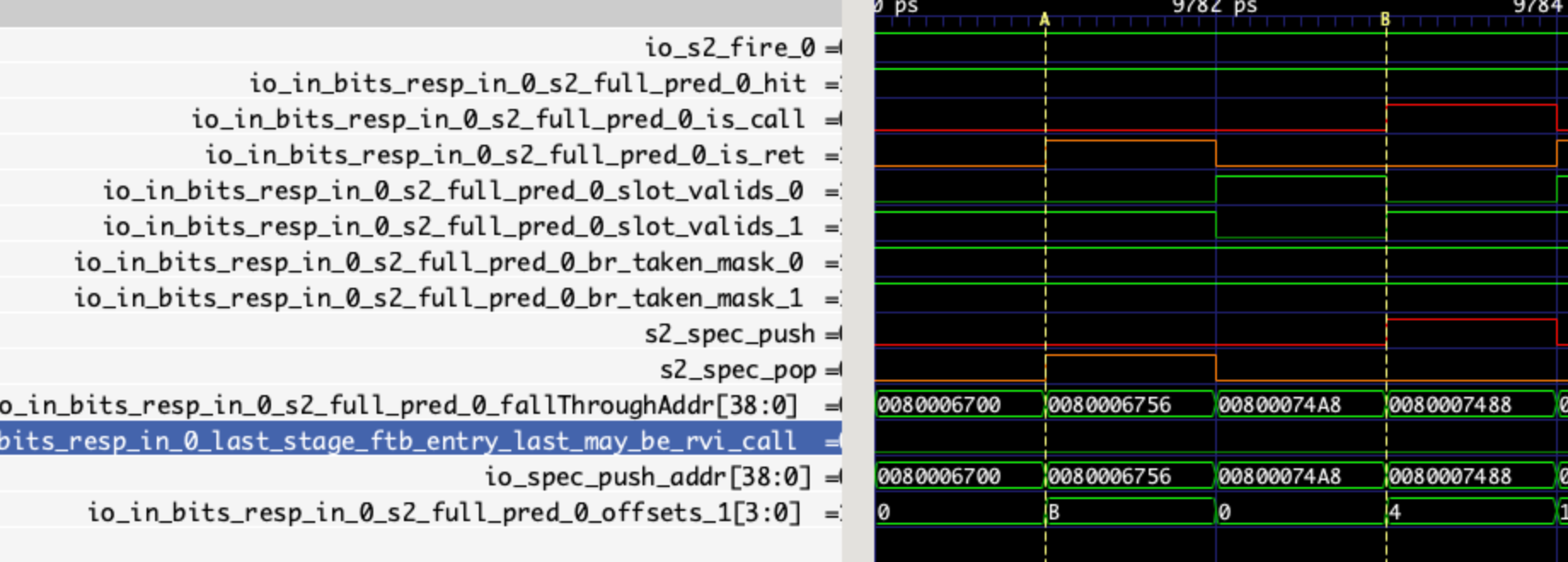


图5.4 RAS预测器整体框图

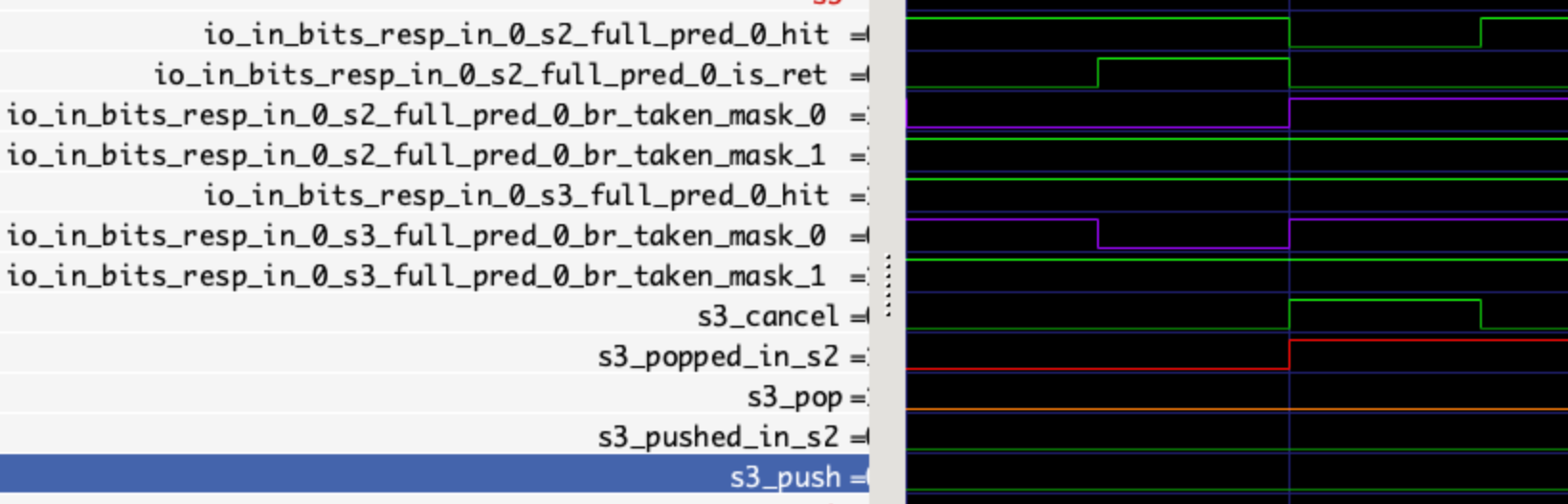
* + 1. 接口时序

RAS模块2阶段更新输入输出接口



上图展示了RAS模块2阶段更新的一次pop和一次push，由于push和pop的块前一个分支预测slot均无效，在本流水级看到return和call指令跳转，因而分别指示RASStack模块出/压栈。在压栈时，使用FTB的fallThrough地址作为跳转返回地址。若最后一条指令为被截断的RVI call指令，则该地址+2才是正确的返回地址。

RAS模块3阶段更新输入输出接口

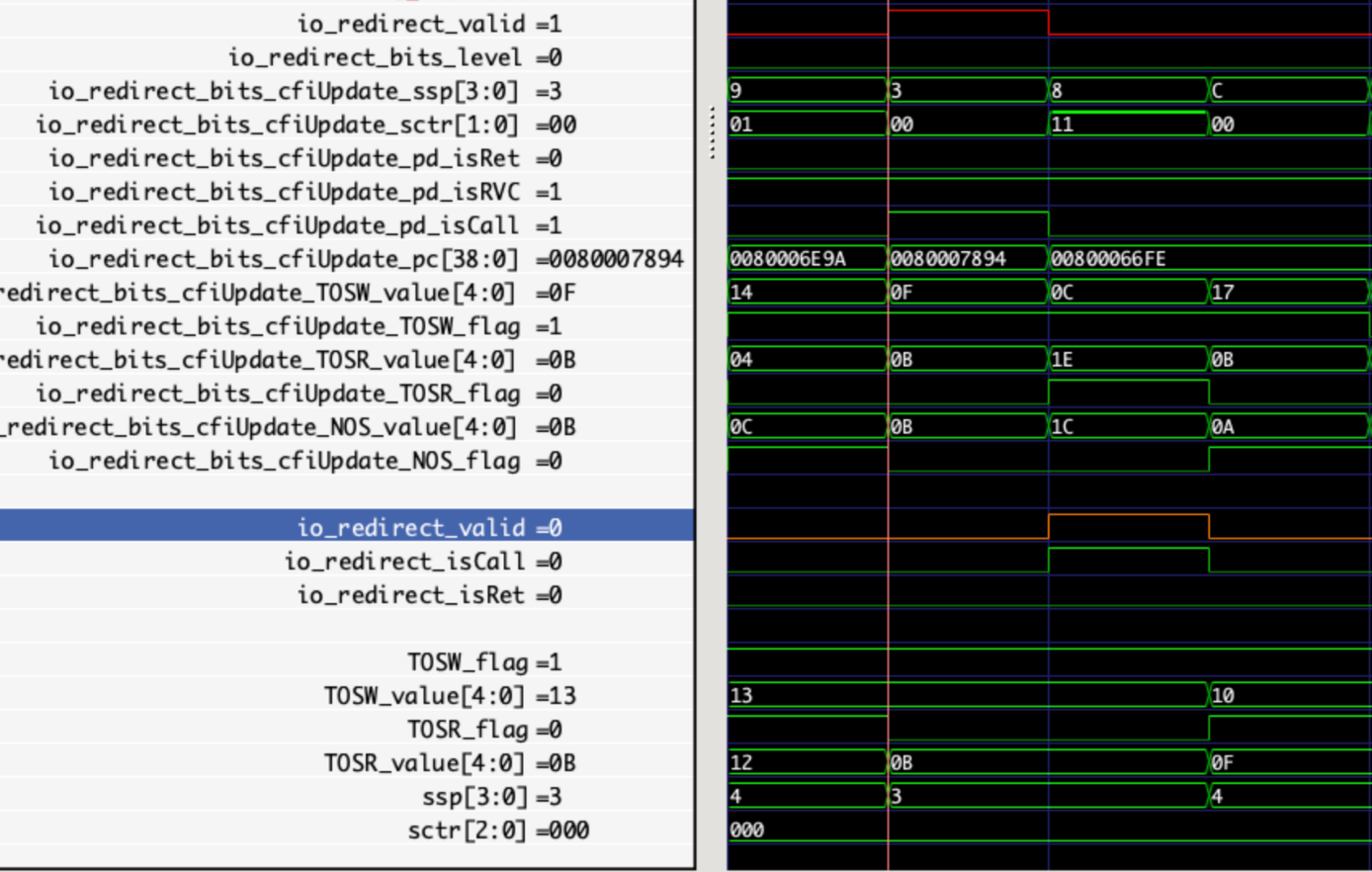


RASStack模块输入接口

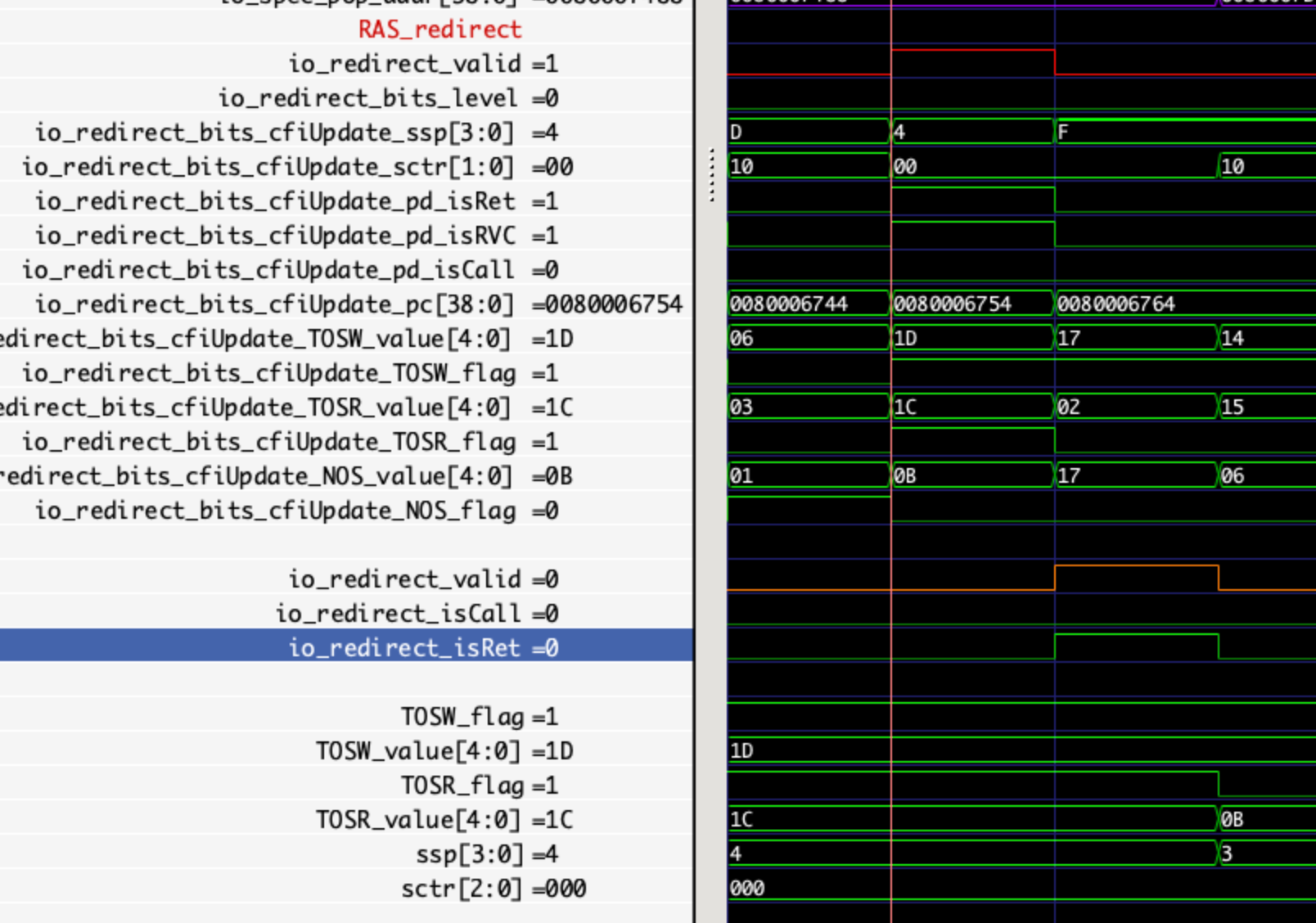


上图展示了RASStack模块2阶段更新的一次push和一次pop操作，可以注意到，在push操作后，从RASStack模块中读出的栈顶被更新为新push的值，而pop操作后，栈顶恢复为push操作前的值。

RAS模块重定向恢复接口

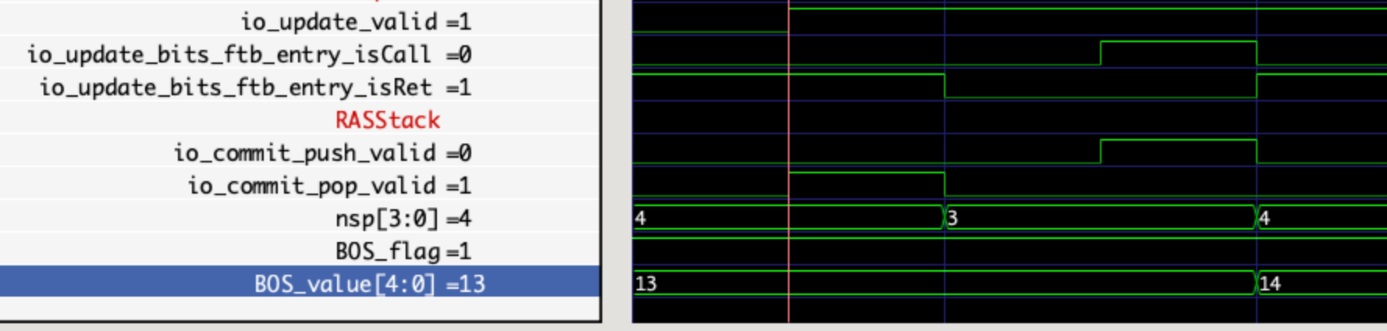


上图展示了RAS和RASStack模块重定向恢复且恢复点指令为call指令的情形，BPU传入的重定向信号在RAS预测器内部被打一拍后送入RASStack，用于恢复持久化队列中各项指针。由于该误预测点指令为call指令，还需压入新项。



类似地，若误预测点指令为return指令，则需基于当时的栈顶pop出一项。

RAS模块指令提交训练接口



如图展示了一次含return和一次含call指令的指令提交，可以看到，在提交时提交栈顶发生变化，且在call指令提交后BOS指针也相应调整。

* + 1. 关键电路

*如果该层级只有例化，没有其他电路可不写。*

*主要包括Memory、流水线设计、仲裁、状态机等。多画图，配以文字说明*

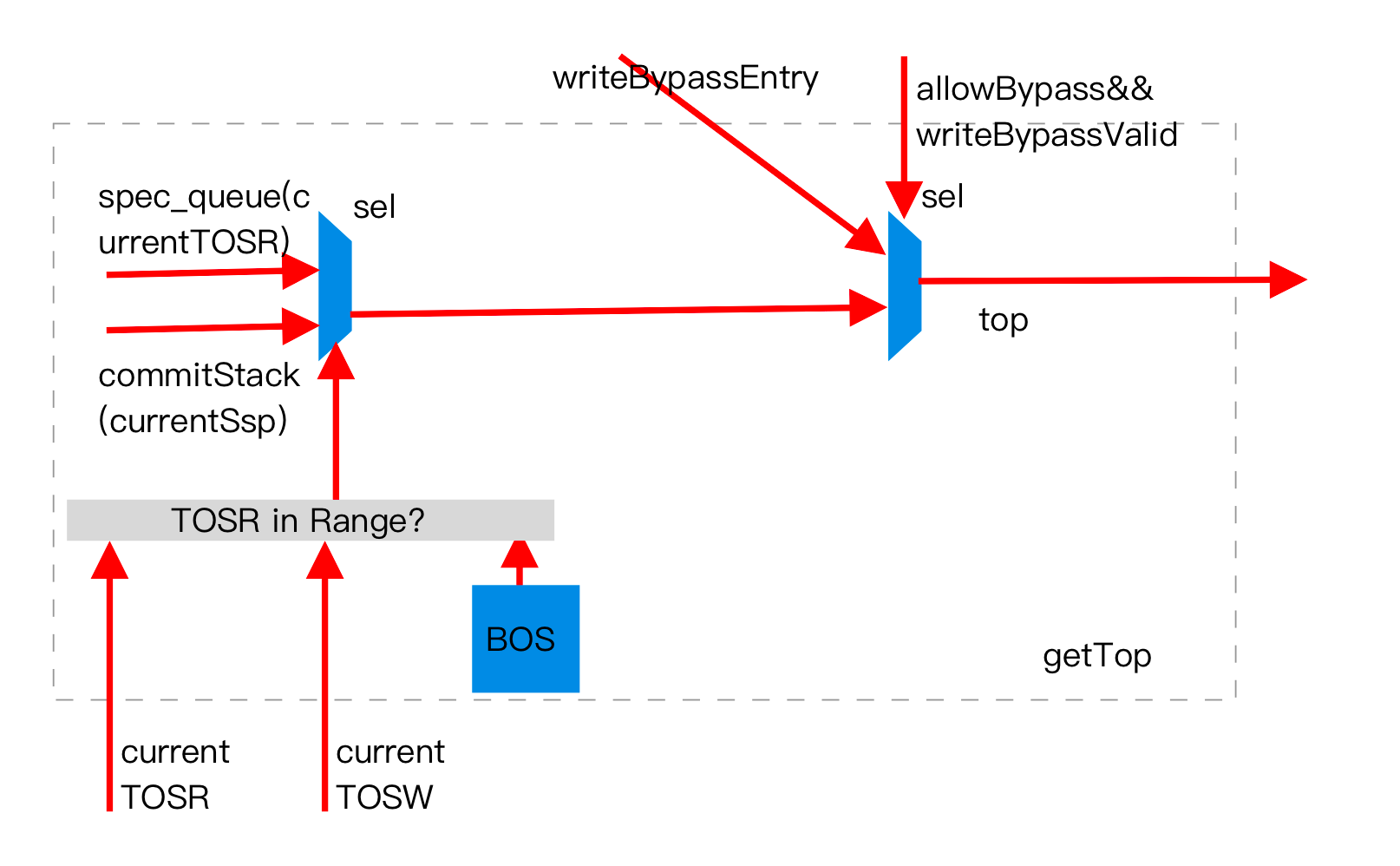


图5.5 getTop逻辑细节

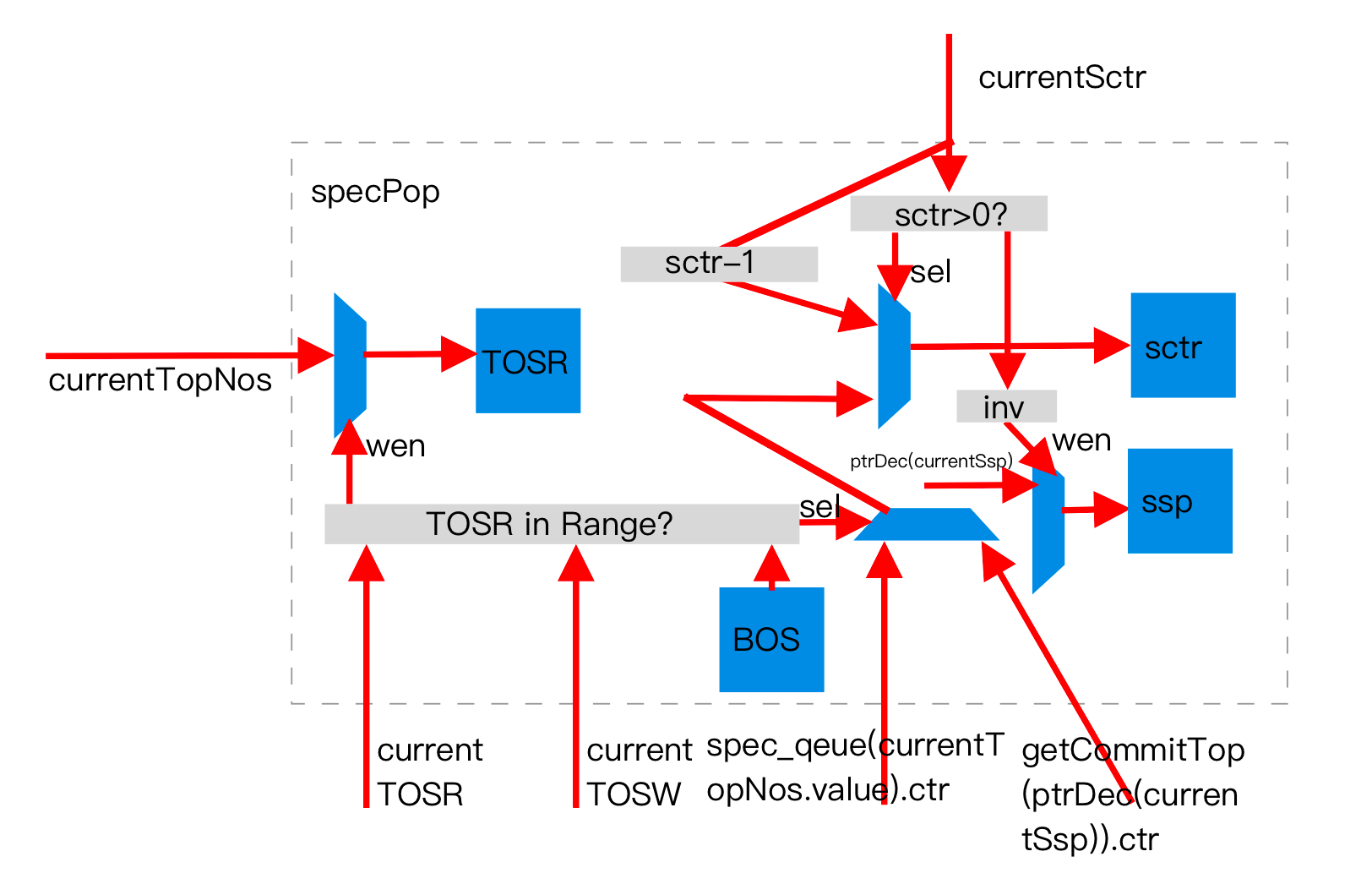


图5.6 specPop逻辑细节

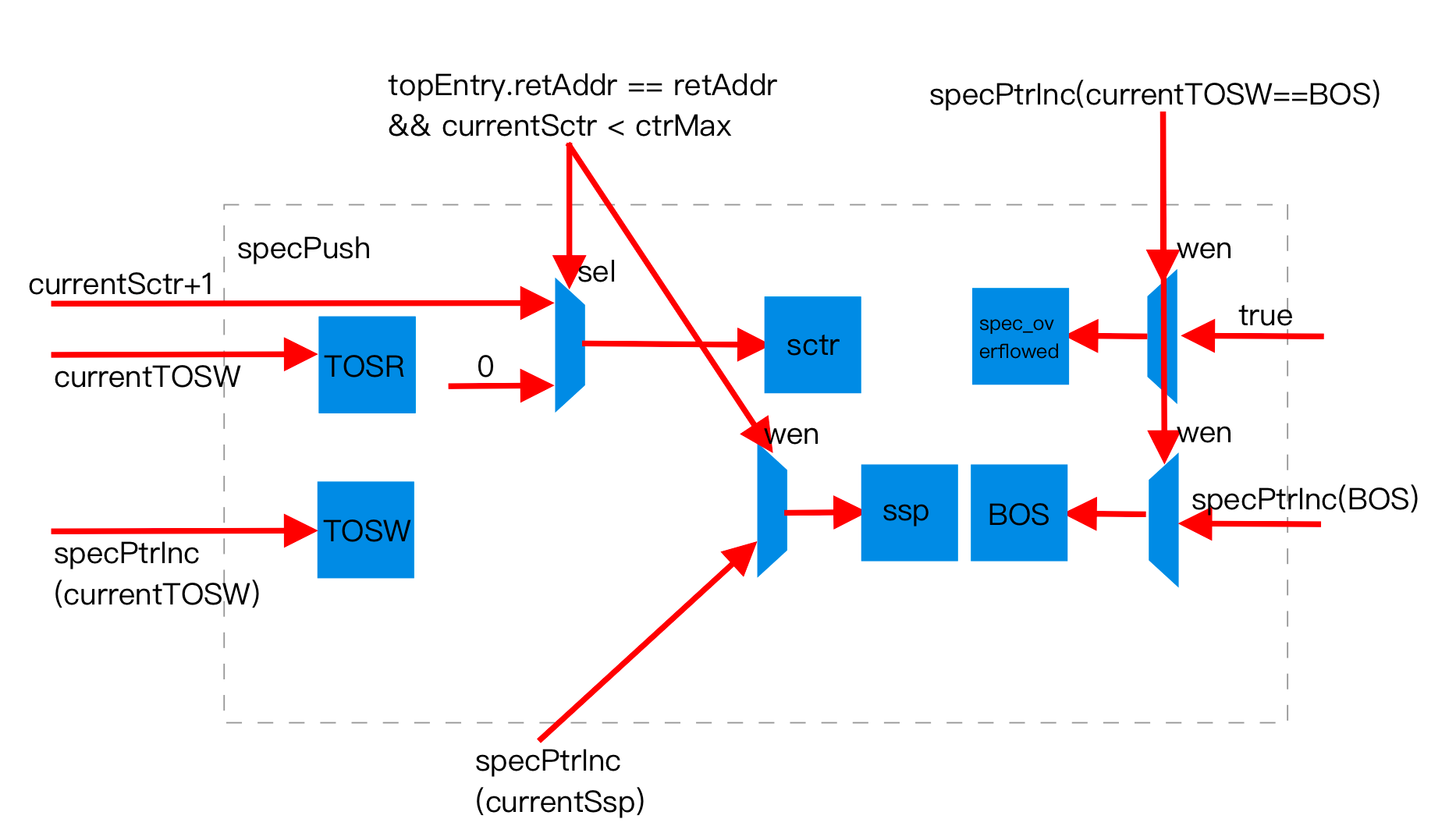
**

图5.7 specPop逻辑细节

* + 1. 三级模块设计

*内容和要求同二级模块设计，如果还有4级模块需要一级级往下写。*

* 1. TAGE-SC
     1. 功能

TAGE-SC 是南湖架构条件分支的主预测器，属于精确预测器（Accurate Predictor，简称 APD）。其中 TAGE 利用历史长度不同的多个预测表，可以挖掘极长的分支历史信息；SC 是统计校正器。

TAGE 由一个基预测表和多个历史表组成，基预测表用 PC 索引，而历史表用 PC 和一定长度的分支历史折叠后的结果异或索引，不同历史表使用的分支历史长度不同。在预测时，还会用 PC 和每个历史表对应的分支历史的另一种折叠结果异或计算 tag，与表中读出的 tag 进行匹配，如果匹配成功则该表命中。最终的结果取决于命中的历史长度最长的预测表的结果。

当 SC 认为 TAGE 有较大的概率误预测时，它会反转最终的预测结果。

在南湖架构中，每次预测最多同时预测 2 条条件分支指令。在访问 TAGE 的各个历史表时，用预测块的起始地址作为 PC，同时取出两个预测结果，它们所用的分支历史也是相同的。

* + - 1. TAGE:设计思路

TAGE作为一种混合式预测器，其优势在于可以同时根据不同长度的分支历史序列，对某一个分支指令分别进行分支预测，并且对在该分支指令在各个历史序列下的准确率进行评估，并且选择历史准确率最高者作为最终分支预测的判断标准。

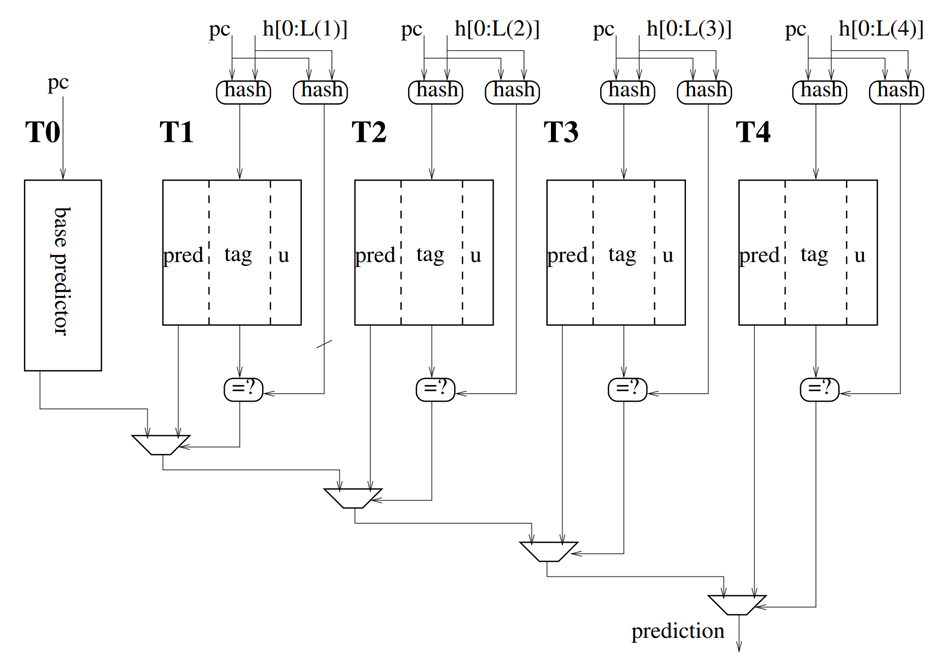
相较于传统的混合式预测器，TAGE兼具下两个新的设计特性，使得其预测准确率得到可观的提升：

* 对于每个预测表中的表项添加了tag数据。在传统的优先分支预测器中，往往仅采用分支历史以及当前分支指令的PC值作为依据来对预测表进行取指。这种情况会导致多条不同的分支指令指向同一个预测表表项的情况(aliasing)产生，而这种情况对混合式预测器中采取的分支历史长度较短的部分预测表所给出的分支预测准确率影响尤为显著。因此，在TAGE的设计中，通过partial tagging的方法，可以更好地将当前指令与预测表中的表项进行实际的匹配，从而较大程度地避免上述情况的产生。
* 采用几何级数变化的分支历史长度来索引不同的预测表，从而使得在分支预测过程中对各个预测表中的表项选择过程的粒度得到了很大的提升。除此之外，在每个表项中还添加了一个usefulness计数器，用于记录该表项对于分支预测的有用程度。通过这种设计，各类分支指令都有更高的几率藉由预测准确率最高的分支历史长度进行索引来做出最终的分支预测。

以上两个设计特性使得TAGE预测器既不会因为所选取的历史分支长度过短而使得某个预测表中的表项同时映射到多条不同的指令从而降低表项的信息有效程度，也不会因为选取的历史长度分支过大而使得整个TAGE需要经过很长时间的更新之后才能够发挥有效的预测功能。因此，TAGE所可采用的分支历史长度范围非常大，可以用于判断各种代码语境下的分支指令。

* + - 1. TAGE:硬件实现

TAGE 是高精度条件分支方向预测器。使用不同长度的分支历史和当前 PC 值寻址多个 SRAM 表，当在多个表中出现命中时，优先选择最命中的历史长度最长的对应表项的预测结果作为最终结果。



TAGE的结构原理图如上图所示，提供了一个基准预测器T0和四个带tag 的预测表T1-T4，基准预测器和带tag的预测表的基本信息见下表。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **预测器** | **是否带tag** | **作用** | **表项构成** | **项数** |
| 基准预测器T0 | 否 | 用于在四个局部tag预测表的tag均不匹配时提供预测结果 | ctr 2bits（最高位给出预测结果，1跳转，0不跳转） | set 2048  way 2 |
| 预测表T1-T4 | 是 | 存在tag匹配时，选历史长度最长者提供预测结果 | valid 1bit  tag 8bits  ctr 3bits（最高位给出预测结果，1跳转，0不跳）  us: 1bit（usefulness计数器） | T1-T4各4096项 |

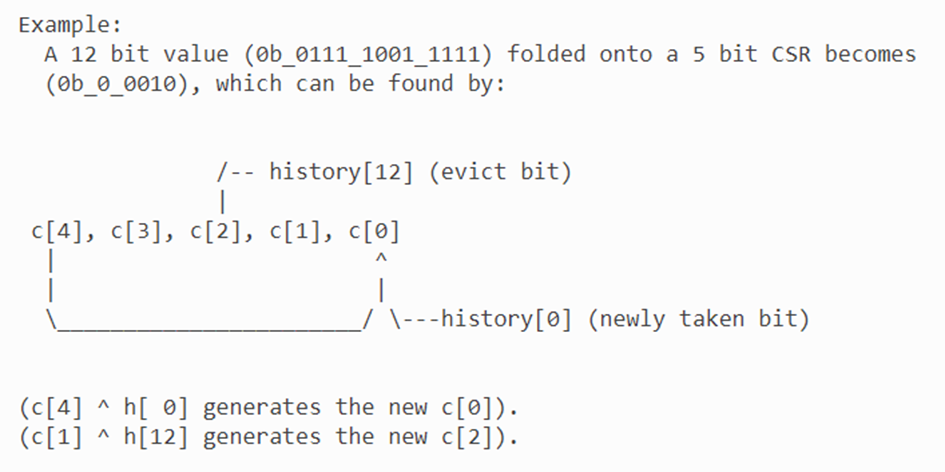
注：上表中的way是chisel里SRAMTemplate类的参数名字，这个参数的值表达在SRAM里面存储几份同样类型的数据，不一定代表通常意义上的需要tag匹配的way。T0有两个way是因为预测块内最多两条分支指令，不同的way是为了分别对预测块内两条不同的分支指令进行预测。

每个预测块的两条跳转指令的预测内容是分开存储、分开update的。带tag的预测表T1-T4的4096项要平均分成两个bank分给预测块内最多两条分支，每个bank有2048项，所以预测表T1-T4的index只有11bit，是log2（预测表的项数/2）。同一个预测块中的两条分支指令的index是相同的，但是索引的是两个不同的bank，一次读出来的来自两个bank代表两条预测信息的结果可能会出现一个是hit，一个没有hit。

TAGE类预测器的每一个预测表都有特定的历史长度。为了让原本很长的全局分支历史表能够与PC异或后进行预测表的索引或tag匹配，原本很长的分支历史序列需要被分成很多段，然后全部异或起来。每一段的长度一般等于历史表深度的对数。由于异或的次数一般较多，为了避免预测路径上多级异或的时延，我们会直接存储折叠后的历史。

每个预测表对应的折叠分支历史都有三份，一份用于索引预测表，两份用于tag匹配。在BPU模块中维护了一个256比特的全局分支历史ghv，并为TAGE的4个带tag的预测表分别维护了各自的折叠分支历史。具体配置见下表（见TageTableInfos），其中ghv是一个循环队列，“低”n 位指的是从ptr指示的位置算起的低位：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **历史** | **index折叠分支历史长度** | **tag折叠分支历史1长度** | **tag折叠分支历史2长度** | **设计原理** |
| 全局分支历史ghv | 256比特（非折叠） | 无 | 无 | 每比特代表对应分支跳转与否 |
| T1对应折叠分支历史 | 8比特 | 8比特 | 7比特 | ghv取ptr起低8比特折叠异或 |
| T2对应折叠分支历史 | 11比特 | 8比特 | 7比特 | ghv取ptr起低13比特折叠异或 |
| T3对应折叠分支历史 | 11比特 | 8比特 | 7比特 | ghv取ptr起低32比特折叠异或 |
| T4对应折叠分支历史 | 11比特 | 8比特 | 7比特 | ghv取ptr起低119比特折叠异或 |



如上图所示，出于时序考虑，折叠分支历史的具体实现方式并不是设计原理中的折叠，而是把折叠前的分支历史最老的那一位（图中对应h[12]）和最新的一位（图中对应h[0]）异或到相应的位置（图中对应c[1]和c[4]），再做一个移位操作（图中对应变为c[0]和c[2]）即可，伪代码如下所示：

c[0] <= c[4] ^ h[0];

c[1] <= c[0];

c[2] <= c[1] ^ h[12];

c[3] <= c[2];

c[4] <= c[3];

分支历史表不只会在后端提交之后更新，s1~s3中间每一级都有可能更新，更新时更新的是pointer和值。如果在s1~s3产生了新的结果就会恢复pointer，更新新的值.但是如果预测没有错误就不会修改（因为之前已经更新过了）。

注：由TAGE的折叠分支历史具体配置表可见，tag折叠分支历史有2个。二者都是使用相同的折叠算法，区别只是在于折叠长度不同，T1-T4的 tag折叠分支历史1都为8比特，而 tag折叠分支历史2都为7比特。

* + - 1. TAGE：预测时序

在每次进行预测时，首先在每个带tag的预测表中使用pc值以及各自的分支历史进行两种不同的哈希函数计算（见下表），计算结果分别用于计算该运算的最终tag以及对预测表的索引。

|  |  |
| --- | --- |
|  | 计算方式 |
| Index | （index折叠分支历史）^（（pc>>1）的低位） |
| Tag | （tag折叠分支历史1）^（tag折叠分支历史2<<1）^（（pc>>1）的低位） |

若T1-T4中索引所得到的valid为1，且tag与计算tag哈希函数得到的结果匹配，则该预测表中给出的pred最高位被加入最终的分支预判序列。最终，通过多级MUX，可以从所有tag匹配的分支预判中选择历史长度最长者作为最终预测结果。

若T1-T4无匹配，则采用T0为最终的预测结果。T0的索引方式为直接用pc的低11位进行索引。

TAGE需要 2 拍延迟：

● 0 拍生成 SRAM 寻址用 index。index的生成过程就是把折叠历史和pc异或，折叠历史的管理不在ITTAGE和TAGE内部，而在BPU里

● 1 拍读出结果

● 2 拍输出预测结果

* + - 1. TAGE：预测器训练

首先，定义所有产生tag匹配的预测表中所需历史长度最长者为provider，而其余产生tag匹配的预测表（若存在的话）被称为altpred。

TAGE表项中包含一个usefulness域，当provider预测正确而altpred预测错误时provider的usefulness置1，表示该项是一个有用的项，便不会被训练时的分配算法当作空项分配出去。当provider产生的预测被证实为一个正确的预测，且此时的provider与altpred的预测结果不同，则provider的usefulness域被置1。

若分支指令实际是跳转，则将对应provider表项的ctr计数器自增1；若分支指令实际是不跳转，则将对应provider表项的ctr计数器自减1。

若由于误预测导致TAGE表项需要更新，且误预测不是由使用altpred而抛弃了正确的provider导致的，则说明需要增添表项。但此时并不一定真的能够增添表项。还需要满足provider所源自的预测表并非所需历史长度最长的预测表，则此时执行表项增添操作。

这里从逻辑上举一个判断是否满足provider所源自的预测表并非所需历史长度最长的预测表的例子：

s1\_providers(i)表示预测块中第i条分支的provider对应的预测表序号，假设provider在预测表T2中，则LowerMask(UIntToOH(s1\_providers(i)), TageNTables)为0b0011。 s1\_provideds(i)表示预测块中第i条分支的provider是否在T1~T4，根据刚刚的假设，Fill(TageNTables, s1\_provideds(i).asUInt)为0b1111，二者相与，得到结果为0b0011，再取反得到0b1100，于是可以看出T3、T4都是比provider历史长度更长的预测表。

再举一个例子。在最开始，预测表都是空的，不存在provider，此时对应的 s1\_providers(i)为0， s1\_provideds(i)为false，则此时Fill(TageNTables, s1\_provideds(i).asUInt)为0b0000，二者相与取反一定得到0b1111，说明T1~T4都属于所谓的比provider历史长度更长的预测表。

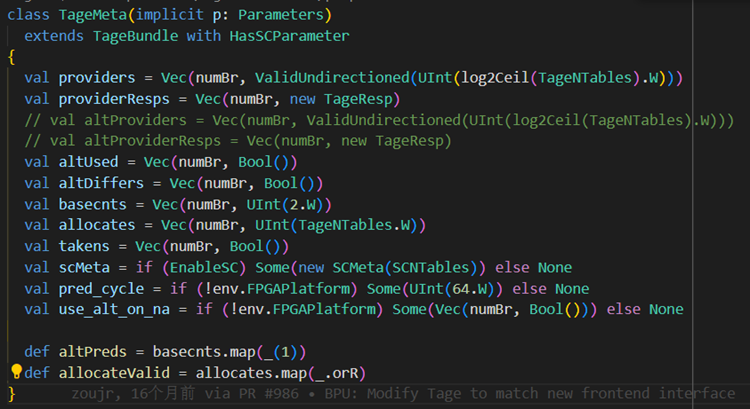
具体的表项增添操作如下：

表项增添操作会首先读取所有历史长度长于provider的预测表的usefulness。若此时有某表的usefulness域值为0，则在该表中分配一对应的表项；若没有找到满足 usefulness域值为0的表，则分配失败。当有多个预测表（如Tj,Tk两项）的usefulness域均为0时，表项的分配概率是随机的，分配的时候随机把某些table给mask掉，让它不会每次都分配同一个。这个mask的具体实现是：待选的历史表中（长度大于provider且u为0的所有历史表），使用产生的随机数随机将一些表屏蔽掉，如果 maskedEntry的第一个不可用，那么选没mask的第一个。这里的表项分配的随机性是通过chisel的util包里的64位线性反馈移位寄存器原语LFSR64生成伪随机数来实现的，在verilog代码中对应allocLFSR\_lfsr寄存器。在训练时，用7bit饱和计数器bankTickCtrs统计分配失败次数-成功次数，当分配失败的次数足够多，bankTickCtrs计数器计数到满达到饱和时，触发全局useful bit reset，把所有的usefulness域清零。

最后，在初始化时/TAGE表分配新项时，所有的表项中的ctr计数器均被设置为0，所有的usefulness域均被设置为0。

注：同一个预测块中的两条分支指令对应的 usefulness不一定是相等的，如果altpred的第一条分支预测跳转，第二条分支预测不跳转，provider的都预测跳转，就只有第一个u要置一。

注：meta是预测器预测的时候的数据，update的时候拿回来更新用。都叫meta是因为composer将所有预测器整合起来，用共同的接口meta和外界交互。TAGE的meta具体构成见下图：



* + - 1. TAGE：备选预测逻辑（USE ALT ON NA）

当tag匹配的项的置信度计数器ctr为0时，altpred有时比正常预测更准确。因此在实现中使用一个4-bit计数器useAltCtr，动态决定是否在最长历史匹配结果信心不足时使用备选预测。在实现中处于时序考虑，始终用基预测表的结果作为备选预测，这带来的准确率损失很小。

备选预测逻辑的具体实现如下：

ProviderUnconf表示最长历史匹配结果的信心不足。当provider对应的ctr值为0b100、0b011时，说明最长历史匹配结果的信心很足，此时providerUnconf为false；当provider对应的ctr值为0b01、0b10时，说明最长历史匹配结果的信心不足，此时providerUnconf为true。

useAltOnNaCtrs是128个4-bit饱和计数器构成的计数器组，每个计数器都被初始化为0b1000。在TAGE收到训练更新请求时，如果拿来训练的预测中，发现provider的预测结果与altpred不同，且provider的预测结果信心不足，则讨论备选预测结果是否正确。如果备选预测结果正确而provider错误，则对应useAltOnNaCtrs计数器的值+1；若备选预测结果错误而provider正确，则对应useAltOnNaCtrs计数器的值-1。因为useAltOnNaCtrs是饱和计数器，所以当useAltOnNaCtrs值已经为0b1111且正确或已经为0b0000且错误时，useAltOnNaCtrs的值不变。

useAltOnNa是利用pc(7, 1)，即pc的对应低位索引useAltOnNaCtrs计数器组后得到的计数结果的最高位。

当providerUnconf && useAltOnNa为真时，使用备选预测结果（即基预测表的结果）为TAGE最终的预测结果，而不是provider的预测结果。

* + - 1. TAGE：wrbypass

Wrbypass里面有Mem，也有Cam，用于给更新做定序，每次TAGE更新时都会写进这个wrbypass，同时写进对应预测表的sram。每次更新的时候会查这个wrbypass，如果hit了，那就把读出的TAGE的ctr值作为旧值，之前随branch指令带到后端再送回前端的ctr旧值就不要了。这样如果一个分支重复更新，那wrbypass可以保证某一次更新一定能拿到相邻的上一次更新的最终值。

T0有一个对应的wrbypass，T1~T4各有16个。每个预测表对应的wrbypass中，Mem都有8个entry，T0每个entry存储2个（对应两个bank）预测表的表项，T1~T4每个entry存储1个（因为2个bank存在不同的wrbypass里）预测表的表项 ；Cam有8个entry，输入更新的idx和tag（T0没有tag）就可以读到对应数据在Cam中的位置。Cam和Mem是同时写的，所以数据在Cam中的位置就是在Mem中的位置。于是利用这个Cam，我们就可以在更新的时候查看对应idx的数据是否在wrbypass中。

* + - 1. SC：设计思路

一些应用上，一些分支行为与分支历史或路径相关性较弱，表现出一个统计上的预测偏向性。对于这些分支，相比TAGE，使用计数器捕捉统计偏向的方法更为有效。TAGE在预测非常相关的分支时非常有效，TAGE 未能预测有统计偏向的分支，例如只对一个方向有小偏差，但与历史路径没有强相关性的分支。

SC统计校正的目的是检测不太可靠的预测并将其恢复。SC负责预测具有统计偏向的条件分支指令并在该情形下反转 TAGE 预测器的结果。

* + - 1. SC：硬件实现

SC共维护了4个预测表，预测表的具体参数见下表。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 项数 | ctr长度 | 折叠后历史长度 | 设计原理 |
| T1 | 512 | 6 | 0 | ghv取ptr起低0比特折叠异或 |
| T2 | 512 | 6 | 4 | ghv取ptr起低4比特折叠异或 |
| T3 | 512 | 6 | 8 | ghv取ptr起低10比特折叠异或 |
| T4 | 512 | 6 | 8 | ghv取ptr起低16比特折叠异或 |

* + - 1. SC：预测时序

SC收到来自TAGE的预测结果pred和ctr、来自BPU的折叠分支历史信息和pc，根据（（index折叠分支历史）^（（pc>>1）的低位））索引SC预测表，根据SC预测表的结果ctr和TAGE的预测结果pred和ctr动态判定（见HasSC）是否反转TAGE的预测。

SC的动态判定具体逻辑如下：

SC中实现了2个bank的scThresholds寄存器组，2个bank是为了和TAGE的2个bank对应，都是因为一个预测块内最多会有两条分支指令。SC中动态判定的寄存器组一共就是这两个。每个bank的scThresholds中有一个5比特的饱和计数器ctr（初始值为0b10000）和一个8比特的thres（初始值为6）。为了以示区别，我们后面用tage\_ctr、sc\_ctr、thres\_ctr来分别表示TAGE传给SC的ctr、SC自己的ctr、scThresholds中的ctr。

scThresholds的更新：当SC收到训练数据时，如果当时SC翻转了TAGE的预测结果，且((sc\_ctr\_0\*2+1)+(sc\_ctr\_1\*2+1)+(sc\_ctr\_2\*2+1)+(sc\_ctr\_3\*2+1)+(2\*(tage\_ctr-4)+1)\*8（有符号进位加）后取绝对值后在[thres-4, thres-2]的范围内，则scThresholds会被更新。

* scThresholds中ctr的更新：若预测正确，则thres\_ctr+1；若预测错误，则thres\_ctr-1；若thres\_ctr已为0b11111且预测正确，或thres\_ctr已为0b00000且预测错误，则thres\_ctr保持不变。
* ScThresholds中thres的更新：若thres\_ctr更新后的值已达0b11111且thres<= 31，则thres+2；若 thres\_ctr更新后的值为0且thres>=6，则thres-2。其余情况thres不变。
* Thres更新判断结束后，会对thres\_ctr再做一次判断，若更新后的thres\_ctr若为0b11111或0，则thres\_ctr会被置回初始值0b10000。

设定scTableSums为 ((sc\_ctr\_0\*2+1)+(sc\_ctr\_1\*2+1)+(sc\_ctr\_2\*2+1)+(sc\_ctr\_3\*2+1) （有符号进位加），tagePrvdCtrCentered为 (2\*(tage\_ctr-4)+1)\*8（有符号进位加），totalSum为scSum+tagePvdr（有符号进位加）。若scTableSums>(有符号的thres - tagePrvdCtrCentered)并且totalSum为正，或者scTableSums<(-有符号的thres - tagePrvdCtrCentered)并且totalSum为负，都说明已经超过了阈值，翻转TAGE的预测结果，否则仍使用TAGE的预测结果不变。

SC的预测算法依赖TAGE里面的是否有历史表hit的信号provided，以及provider的预测结果taken，从而来决定SC自己的预测。provided是使用SC预测的必要条件之一，provider的taken作为choose bit，选出SC最终的预测，这是因为SC在TAGE 预测结果不同的场景下可能有不同的预测。

sc反转TAGE的预测结果会造成TAGE表增添新项。

SC需要3拍延迟：

● 0 拍生成寻址 index得到索引s0\_idx

● 1 拍读出SCTable对应s0\_idx的计数器数据s1\_scResps

● 2 拍根据s1\_scResps选择是否需要反转预测结果

● 3 拍输出完整的预测结果

* + - 1. SC：Wrbypass

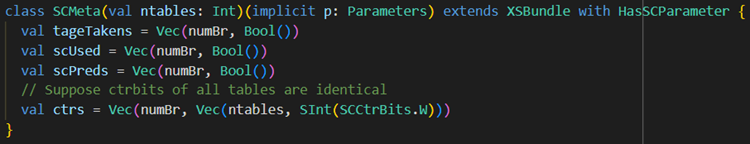
Wrbypass里面有Mem，也有Cam，用于给更新做定序，每次SC更新时都会写进这个wrbypass，同时写进对应预测表的sram。每次更新的时候会查这个wrbypass，如果hit了，那就把读出的SC的ctr值作为旧值，之前随branch指令带到后端再送回前端的ctr旧值就不要了。这样如果一个分支重复更新，那wrbypass可以保证某一次更新一定能拿到相邻的上一次更新的最终值。

SC的T1~T4各有2个wrbypass，每个预测表的wrbypass中，Mem都有16个entry，每个entry存储2个预测表的表项；Cam有16个entry，输入更新的idx就可以读到对应数据在Cam中的位置。Cam和Mem是同时写的，所以数据在Cam中的位置就是在Mem中的位置。于是利用这个Cam，我们就可以在更新的时候查看对应idx的数据是否在wrbypass中。

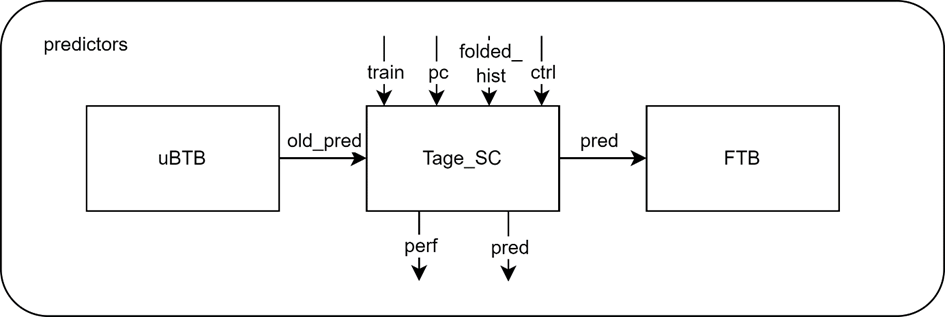
* + - 1. SC：预测器训练

sc\_ctr（见signedSatUpdate）是一个6比特的有符号饱和计数器，在指令实际taken后+1，nottaken-1，计数范围是[-32，31]。

注：meta是预测器预测的时候的数据，update的时候拿回来更新用。都叫meta是因为composer将所有预测器整合起来，用共同的接口meta和外界交互。SC的meta具体构成见下图：



* + 1. 整体框图

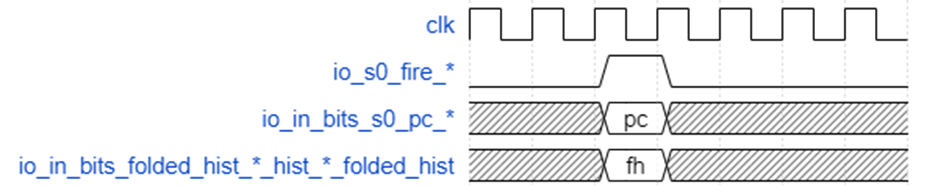


* + 1. 接口列表

见excel。

* + 1. 接口时序

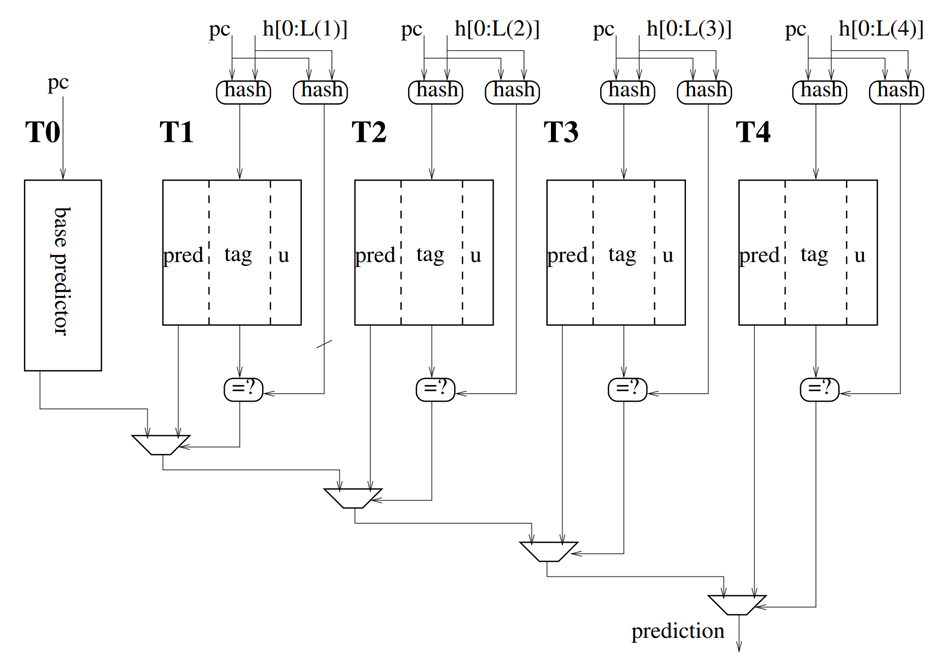
1. s0输入pc和折叠历史时序示例



上图示意了s0输入pc和折叠历史时序的示例，当io\_s0\_fire为高时，输入的io\_in\_bits数据有效。

* + 1. 关键电路

1. *如果该层级只有例化，没有其他电路可不写。*
2. *主要包括Memory、流水线设计、仲裁、状态机等。多画图，配以文字说明*



1. 图表 1 TAGE实现原理图
   1. ITTAGE
      1. 功能

ITTAGE接收来自BPU内部的预测请求，其内部由一个基预测表和多个历史表组成，每个表项中都有一个用于存储间接跳转指令目标地址的字段。基预测表用PC索引，而历史表用PC和一定长度的分支历史折叠后的结果异或索引，不同历史表使用的分支历史长度不同。在预测时，还会用PC和每个历史表对应的分支历史的另一种折叠结果异或计算tag，与表中读出的tag进行匹配，如果匹配成功则该表命中。最终的结果取决于命中的历史长度最长的预测表的结果。最终，ITTAGE将预测结果输出至composer。

* + - 1. 间接跳转指令的预测

ITTAGE用于预测间接跳转指令。普通分支指令和无条件跳转指令的跳转目标直接编码于指令中，便于预测，而间接跳转指令的跳转地址来自运行时可变的寄存器，从而有多种可能选择，需要根据分支历史对其作出预测。

为此，ITTAGE 的每个表项在TAGE表项的基础上加入了所预测的跳转地址项，最后输出结果为选出的命中预测跳转地址而非选出的跳转方向。

由于每个FTB项仅存储至多一条间接跳转指令信息，ITTAGE预测器每周期也最多预测一条间接跳转指令的目标地址。

香山南湖架构中的ITTAGE提供了5个带tag的预测表T1-T5，基准预测器和带tag的预测表的基本信息见下表。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **预测器** | **是否带tag** | **作用** | **表项构成** | **项数** |
| 基准预测器T0 | 否 | 用于在带tag预测表的tag均不匹配时提供预测结果 | ITTAGE中并没有实现T0，而是直接使用ftb的预测结果作为基准预测结果 |  |
| 预测表T1-T5 | 是 | 存在tag匹配时，选历史长度最长者提供预测结果 | valid 1bit  tag 9bits  ctr 2bits（最高位表示要不要输出这个预测结果 ）  us: 1bit（usefulness计数器）  target：39 bits | T1-T2各256项  T3-T5各512项 |

在BPU模块中维护了一个256比特的全局分支历史ghv，并为ITTAGE的5个带tag的预测表分别维护了各自的折叠分支历史，折叠算法同TAGE。折叠历史具体配置见下表，其中ghv是一个循环队列，“低”n 位指的是从ptr指示的位置算起的低位：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **历史** | **index折叠分支历史长度** | **tag折叠分支历史1长度** | **tag折叠分支历史2长度** | **设计原理** |
| 全局分支历史ghv | 256比特 | 无 | 无 | 每比特代表对应分支跳转与否 |
| T1对应折叠分支历史 | 4比特 | 4比特 | 4比特 | ghv取ptr起低4比特折叠异或 |
| T2对应折叠分支历史 | 8比特 | 8比特 | 8比特 | ghv取ptr起低8比特折叠异或 |
| T3对应折叠分支历史 | 9比特 | 9比特 | 8比特 | ghv取ptr起低13比特折叠异或 |
| T4对应折叠分支历史 | 9比特 | 9比特 | 8比特 | ghv取ptr起低16比特折叠异或 |
| T5对应折叠分支历史 | 9比特 | 9比特 | 8比特 | ghv取ptr起低32比特折叠异或 |

ITTAGE 需要 3 拍延迟：

● 0 拍生成寻址 index

● 1 拍读出数据

● 2 拍选出命中结果

● 3 拍输出

* + - 1. Wrbypass

Wrbypass里面有Mem，也有Cam，用于给更新做定序，每次ITTAGE更新时都会写进这个wrbypass，同时写进对应预测表的sram。每次更新的时候会查这个wrbypass，如果hit了，那就把读出的ITTAGE的ctr值作为旧值，之前随branch指令带到后端再送回前端的ctr旧值就不要了。这样如果一个分支重复更新，那wrbypass可以保证某一次更新一定能拿到相邻的上一次更新的最终值。

ITTAGE的每一个预测表T1~T5都有着一个对应的wrbypass，每个预测表的wrbypass中，Mem都有4个entry，每个entry存储1个ctr；Cam有4个entry，输入更新的idx和tag就可以读到对应数据在Cam中的位置。Cam和Mem是同时写的，所以数据在Cam中的位置就是在Mem中的位置。于是利用这个Cam，我们就可以在更新的时候查看对应idx的数据是否在wrbypass中。

* + - 1. 预测器训练

首先，定义所有产生tag匹配的预测表中所需历史长度最长者为provider，而其余匹配的预测表（若存在的话）被称为altpred。 当provider的ctr为0时，会选择altpred的结果作为预测结果。

ITTAGE表项中包含一个usefulness域，当provider预测正确而altpred预测错误时provider的usefulness置1，表示该项是一个有用的项，便不会被训练时的分配算法当作空项分配出去。当provider产生的预测被证实为一个正确的预测，且此时的provider与altpred的预测结果不同，则provider的usefulness域被置1。

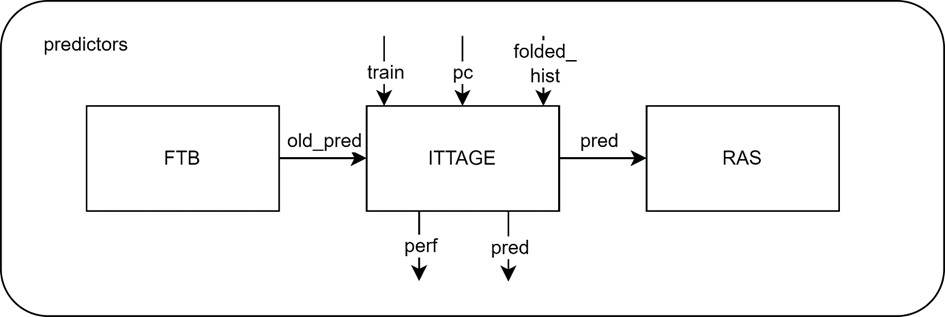
若预测地址与实际一致，则将对应provider表项的ctr计数器自增1；若预测地址与实际不一致，则将对应provider表项的ctr计数器自减1。ITTAGE中，会根据ctr判断是否采取这个预测的跳转目标结果。当ctr为0时，会选择altpred的结果。

接下来，若该provider所源自的预测表并非所需历史长度最高的预测表，则此时执行如下的表项增添操作。表项增添操作会首先读取所有历史长度长于provider的预测表的usefulness域。若此时有某表的usefulness域值为0，则在该表中分配一对应的表项；若没有找到满足 usefulness域值为0的表，则分配失败。当有多个预测表（如Tj,Tk两项）的usefulness域均为0时，表项的分配概率是随机的，分配的时候随机把某些table给mask掉，让它不会每次都分配同一个。这里的表项分配的随机性是通过chisel的util包里的64位线性反馈移位寄存器原语LFSR64生成伪随机数来实现的，在verilog代码中对应allocLFSR\_lfsr寄存器。在训练时，用**8位**饱和计数器**tickCtr**统计分配失败次数-成功次数，当分配失败的次数足够多，tickCtr计数器计数到满达到饱和时，触发全局useful bit reset，把所有的usefulness域清零。

注：ITTAGE的清零usefulness域的饱和计数器名字是tickCtr，长度为8比特，名字和长度均与TAGE不同。

最后，在初始化时/TAGE表分配新项时，所有的表项中的ctr计数器均被设置为0，所有的usefulness域均被设置为0。

* + 1. 整体框图



* + 1. 接口列表

见excel。

* + 1. 接口时序

S0输入pc和折叠历史时序示例



上图示意了s0输入pc和折叠历史时序的示例，当io\_s0\_fire为高时，输入的io\_in\_bits数据有效。

* + 1. 关键电路

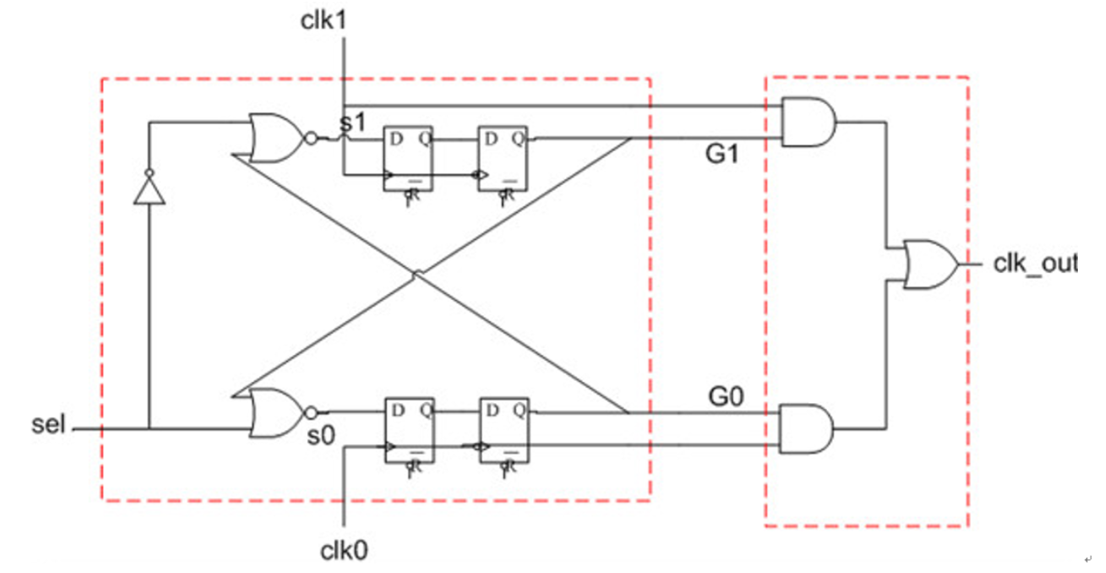
*例*

* + 1. 时钟切换电路

上述CRG框图中，紫色粗框内的clock\_mux是动态时钟切换模块，在用户程序配置PLL之后，通过配置寄存器cfg\_clk\_sel=1将系统时钟从晶振动态切换到PLL时钟。

为了保证时钟动态切换不会导致系统出错，需要使用无毛刺时钟切换电路，电路图如下：

图 2 无毛刺时钟切换电路图



上图中，下面两个寄存器的复位值为1，上面两个寄存器的复位值为0。复位时，clk\_out默认选择clk0时钟。

对于两级同步寄存器，同步器的第一级采用时钟上升沿触发，第二级采用时钟下降沿触发。

1. PPA

*内容包含:*

1. *Power*

*描述功耗设计目标*

*详细描述设计的功耗预估情况*

1. *Performance*

*详细描述设计的性能目标数据*

*详细描述性能的预估情况*

1. *Area*

*详细描述设计的面积目标数据*

*详细描述面积的预估情况*

*4.为优化PPA做的一些关键设计点（例如为了时序收敛做的一些面积/功耗/性能上的折中）*

1. 验证关注点

*从设计角度列举需要验证人员特别关注的测试点。*

*不涉及填“NA”*

1. Floorplan 建议

*芯片的floorplan考虑，依据数据流向，IO排布，模块大小进行芯片布局摆放设计*

*不涉及填“NA”*

1. 遗留问题

*需要跟踪的遗留问题*

*不涉及填“NA”*