RIDECORE: RIsc-v Dynamic Exection CORE

笔记本:	markdown		
创建时间:	2019/10/29 7:56	**更新时间:**	2019/10/29 20:16
作者:	fmrt_2016owen@163.com		
URL:	file:///C:/Users/fmrtc/AppData/Local/youdao/dict/Dict/8.5.2.0/resultui/html/hua...		

RIDECORE: RIsc-v Dynamic Exection CORE

RIDECORE Overview

This part first explains the overall behavior of RIDECORE. After that, various parameters that determine the specifications of RIDECORE are shown.

1、总体行为

图1显示了RIDECORE流水线的概述。 RIDECORE具有六级流水线结构，包括指令提取（IF），指令解码（ID），调度（DP），选择和唤醒（SW），执行（EX）和完成（COM）。除EX阶段的Load / Store单元外，每个流水线阶段均执行一个时钟周期。

在IF阶段，最多可以使用程序计数器（PC）从指令存储器（IMEM）中提取两条指令，并将其发送到后续阶段。通过在下降沿时钟跳变上读取IMEM并在上升沿时钟跳变上写入IF / ID锁存器，可以在一个时钟周期内执行IF级。使用Gshare分支预测变量推测性更新PC。

ID阶段解码在IF阶段提取的指令，并为后续阶段生成数据。推测标签gen将推测标签分配给指令。如果存在分支指令，则将同时更新“推测标签生成”和“未命中预测修复表”。

DP阶段首先从架构寄存器文件（ARF）和重命名寄存器文件（RRF）中获取两条指令的源操作数。接下来，它通过将重排序缓冲区和RRF（在RIDECORE中称为RRFTag）中的条目分配给指令来执行寄存器重命名。最后，分配单元将执行指令所需的数据写入保留站（RS）。使用Source Operand Manager执行数据转发。

在SW阶段，发行单元选择所有操作数已经就绪的指令，并将其发放到 EX 阶段。发出指令后，会立即将其从预留站中删除

在 EX 阶段，将执行指令。指令执行完成后，结果将写入 RRF。同时，将通知重新排序缓冲区完成执行。 分支单元负责确定分支预测是否正确，并将预测结果发送到某些模块以执行适当的操作。当发生分支错误预测时，使用Miss Prediction Fix Table将推测执行的指令失效。在Ex阶段的每条指令，除了Load/Store指令，都在一个时钟周期内执行，加载/存储单元的操作分为两个阶段流水。

在COM阶段，最多完成 2 个重新排序缓冲区中的指令。指令完成后，RRF 中指令的数据移至 ARF。ARF 中的重命名表将相应地更新。分支预测器也使用新信息进行更新。

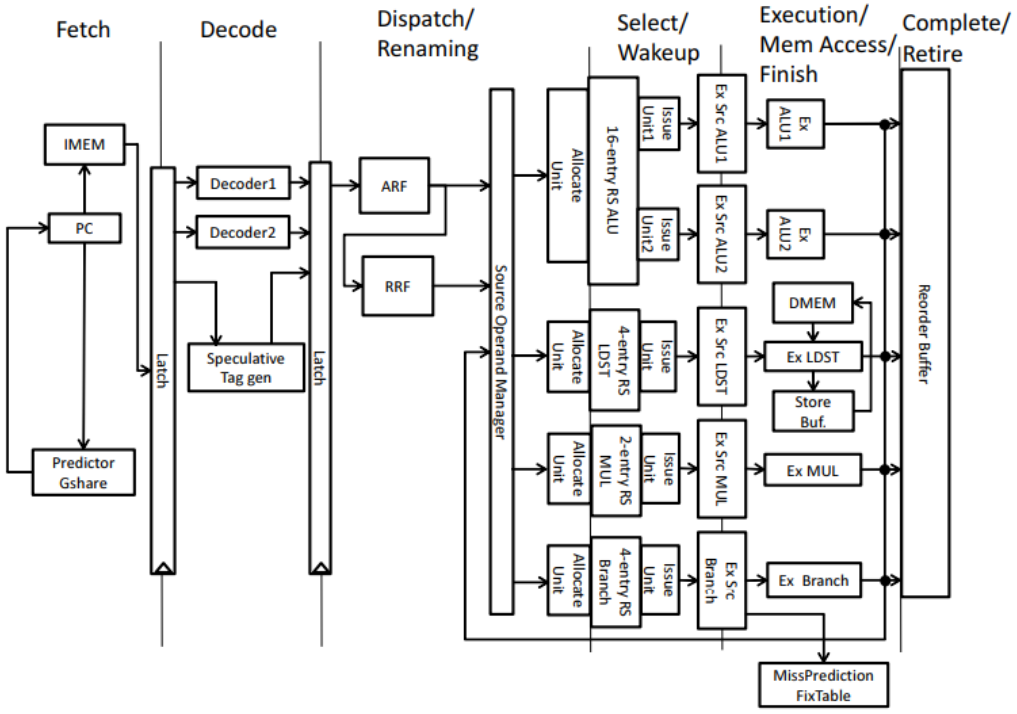


Fig.1 RIDECORE pipeline

2、规范描述

表1列出了RIDECORE的规范

Table 1 The specification of RIDECORE

Instruction Set Architecture	RISC-V (A part of RV32IM, see doc/RISC-V-subset.pdf)
Number of ways	2 (Fetch, Decode, Dispatch, Complete)
Width	
Data	32
Address	32
Number of entries	
Architected Register File (ARF)	32
Rename Register File (RRF)	64
Reorder Buffer	64
Number of Checkpoint(Speculative Tag)	5
Number of entries in each Reservation Station	
ALU	16
Load/Store	4
Branch	4
MUL	2
Number of Execution units	
ALU	2
Load/Store	1
Branch	1
MUL	1

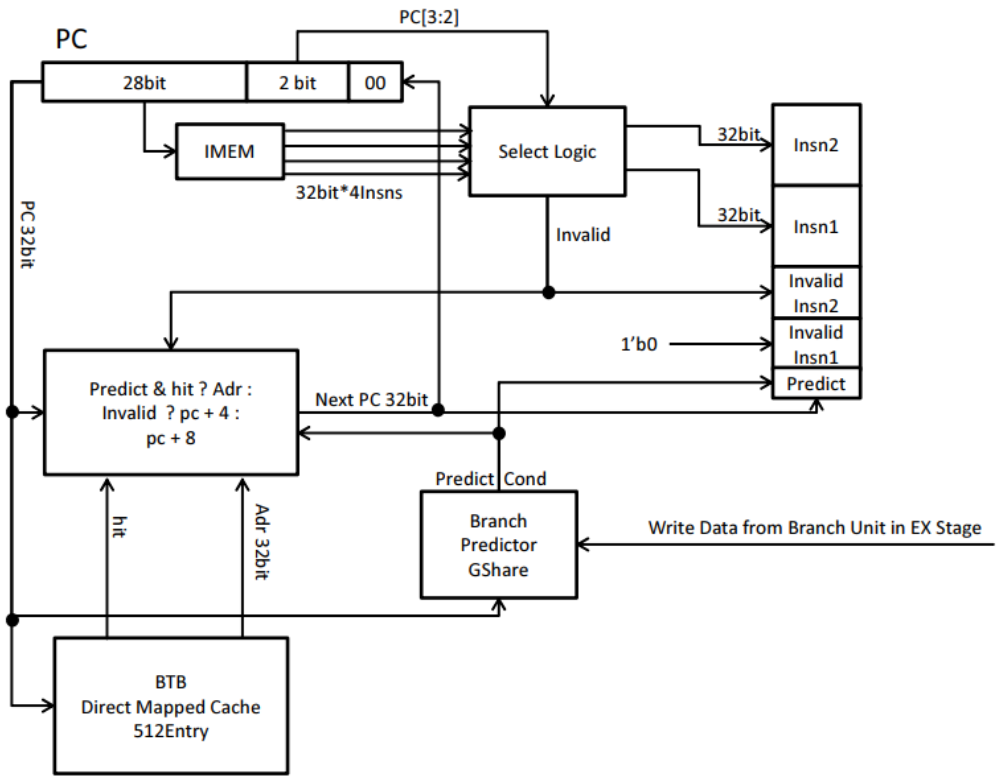


Fig.2 pipe_if

硬件模型

本部分描述了RIDECORE内部的硬件模块。有关某些模块的更深入的解释，请参见John Paul Shen和Mikko H. Lipasti [1]。在这种情况下，我们建议读者先阅读本书中的解释，然后再阅读本文档中的解释。例如，如果应该在本书的第228页至第231页中阅读模块的说明，我们将在模块说明部分的标题后附加（第228-231页）。

3、流水线之IF阶段

在 IF 阶段，从 IMEM 获取两个指令。使用分支预测器更新 PC。图 2 显示 IF 级的电路。

IMEM 中每个条目的大小为 128 位。因此，由于 RV32IM 具有固定长度的 32 位指令，因此 IMEM 的每个条目可以包含四个指令。

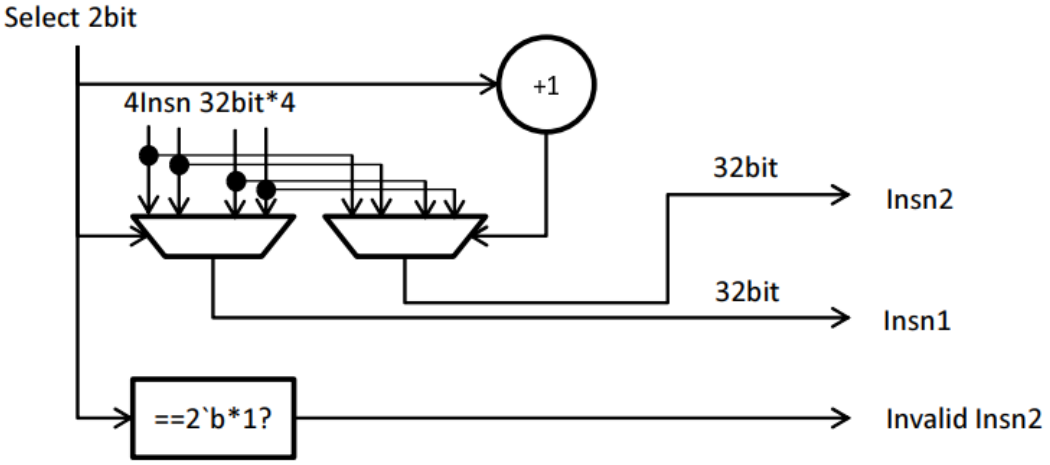


Fig.3 Select logic

在每个负边缘时钟翻转时，使用 PC[31：4] 作为 IMEM 的索引来获取四个指令。然后sellog（图 3）使用 PC[3：2] 从四个指令中选择两个指令，并在正边缘时钟转换时将它们写入 IF/ID 闩锁。当 PC 不能被 8 整除时，sellog 会使第 2 个选定指令（图 3 中的 Insn2）无效，因为此指令以前已选择。例如，当

PC_0x2c（不可被 8 整除时）时，来自 IMEM 的四条指令的指令为 0x20、0x24、0x28 和 0x2c。Insn1 是 0x2c 处的指令，而 Insn2 是 0x20 处的指令，之前已选择该指令（我们预计 Insn2 为 0x30）。在这种情况下，Insn2 因此无效。

为简单起见，如图2所示，无效Insn1始终设置为0。但是，当检测到某些事件（如分支预测错误）时，可以将其设置为1以杀死IF阶段的所有指令。

4、gshare predictor (pp. 223-236, 469-472)

图4示出了Gshare分支预测器电路。它通过使用共享的分支历史寄存器（BHR），模式历史表（PHT），PC和分支目标缓冲区（BTB）来预测是否在IF阶段进行分支。在这里，我们仅介绍实现预测器时的一些设计决策。

在每个时钟脉冲的负沿，我们读取PHT的条目，其读取地址= PC [12：3]⊕BHR。如果读取的数据大于1，则将预测结果设置为“已接收”。否则，将其设置为“不采用”。使用预测结果更新BHR。但是，如果预测结果不正确，则BHR将恢复为原始值。因此，它总是在每次更新之前备份。

4.1 PHT

PHT是Gshare预测器中使用的2位饱和计数器的数组。图5示出了PHT的电路。分支指令完成后更新PHT。PHT的表项的增加/减少对应分支指令的条件是“已接受” /“未接受”。

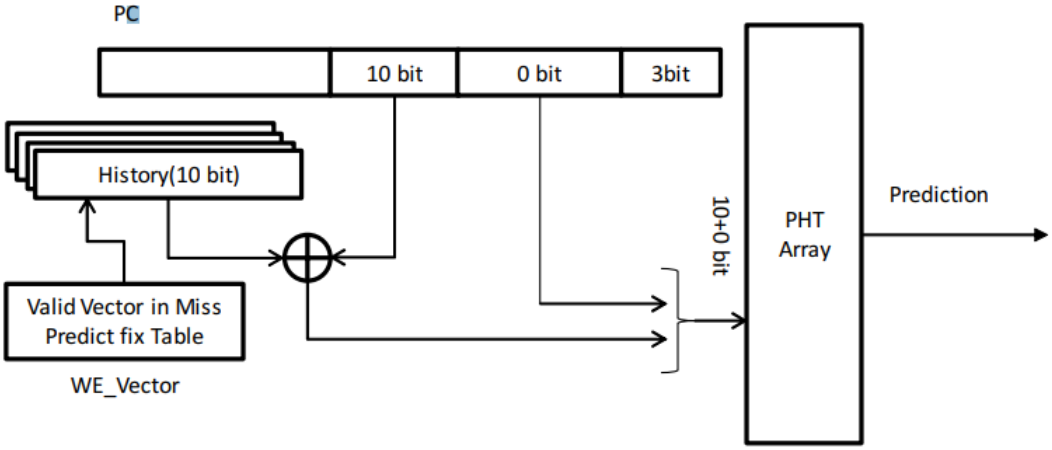


Fig.4 Gshare branch predictor

在COM阶段，通过以下方式更新PHT的条目：先在负沿时钟跳变上读取其原始值，然后在正沿时钟跳变上写入更新的值。PHT有两个读取端口（一个在IF阶段读取，另一个在COM阶段）和一个写端口（在COM阶段进行写入）。在RIDECORE中，它是使用两个真正的双端口BRAM实现的。

4.2 BTB

分支目标缓冲区（BTB）由成对的分支源地址和目标地址组成。它被实现为直接映射的缓存，以减少硬件资源的使用。因此，预测质量适中。

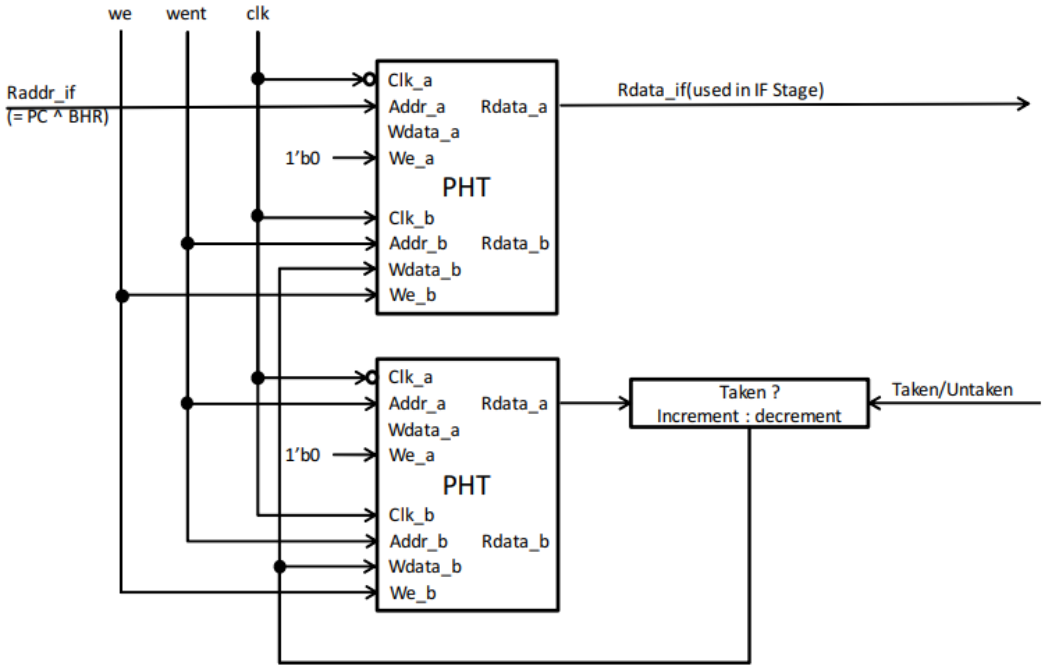


Fig.5 Pattern History Table (PHT)

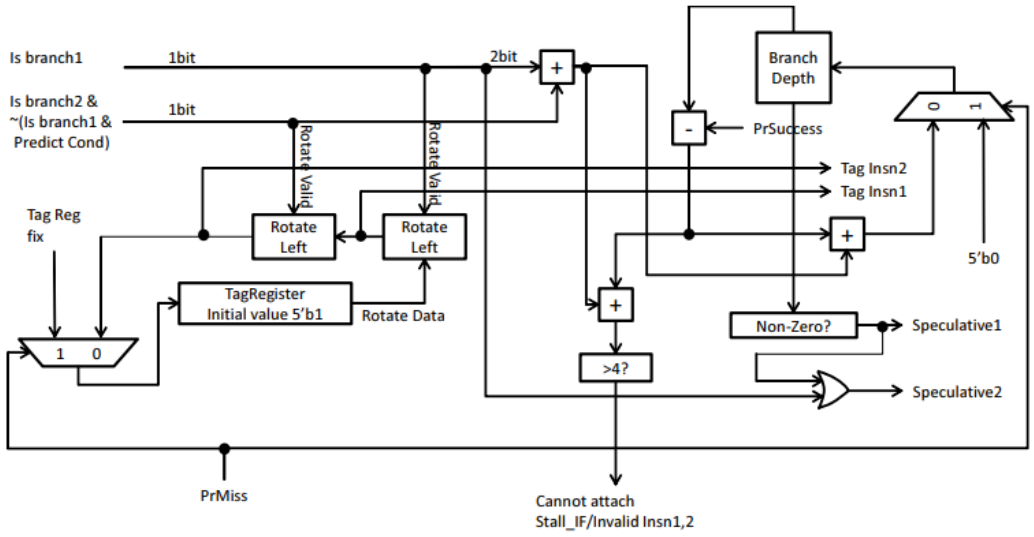


Fig.6 Speculative Tag Generator

5 tag generator (pp. 228-231)

图6示出了标签生成器的电路。该电路有两个寄存器。其中一个包含当前的推测性标记（tagreg）。另一个包含当前分支深度（brdepth）。标记生成器从外部接收三个信号：branchvalid1，branchvalid2和enable。 branchvalid1 / 2

指示Insn1 / 2是否有效，而enable指示是否可以发出指令。使用这些信号，标记生成器生成推测标记（sptag1 / 2），推测标记（speculative1 / 2），并更新tagreg和brdepth。

推测标签按00001、00010，...，10000的顺序分配给指令（从00001开始向左旋转）。如果没有推测性标签可用，标签生成器会将可附加的标志设置为零，并停止标签分配过程。

当分支预测成功（成功== 1）时，brdepth递减以释放推测性标记。发生分支错误预测时（prmiss == 1），将tagreg和brdepth恢复为执行预测之前的值。使用tagregfix（在“分支”单元中生成）还原tagreg。 brdepth设置为零，因为按顺序执行了分支指令（在使推测指令无效之后，就没有剩余的推测指令）。

6、译码

译码器是ID阶段使用的模块。该模块使用传入指令中的信息来生成数据以用于后续阶段（DP，EX等）。图7示出了译码器的I / O定义。

```
module decoder
  (
   input wire [31:0]                    inst,
   output reg [`IMM_TYPE_WIDTH-1:0]     imm_type,
   output wire [`REG_SEL-1:0]           rs1,
   output wire [`REG_SEL-1:0]           rs2,
   output wire [`REG_SEL-1:0]           rd,
   output reg [`SRC_A_SEL_WIDTH-1:0]    src_a_sel,
   output reg [`SRC_B_SEL_WIDTH-1:0]    src_b_sel,
   output reg                           wr_reg,
   output reg                           uses_rs1,
   output reg                           uses_rs2,
   output reg                           illegal_instruction,
   output reg [`ALU_OP_WIDTH-1:0]       alu_op,
   output reg [`RS_ENT_SEL-1:0]         rs_ent,
   output wire [2:0]                    dmem_size,
   output wire [`MEM_TYPE_WIDTH-1:0]    dmem_type,
   output reg [`MD_OP_WIDTH-1:0]        md_req_op,
   output reg                           md_req_in_1_signed,
   output reg                           md_req_in_2_signed,
   output reg [`MD_OUT_SEL_WIDTH-1:0]   md_req_out_sel
   );
```

Fig.7 Decoder module Interface

下面是这些信号的解释

•imm_type：此信号在模块imm_gen中使用。它指示指令中立即数据的格式。

•rs1，rs2，rd：第一个源操作数，第二个源操作数和目标寄存器号。

•src_a_sel，src_b_sel：用于选择ALU操作数。

•wr reg：此信号指示指令是否将数据写入目的寄存器。

•uses_rs1，uses_rs2：rs1和rs2的有效信号。如果rs1 / 2有效，则需要获取数据。这些信号用于防止获取不必要的数据（在获取所有必需的数据之前，不能发出指令）。

•illegal_instruction：此信号表明该处理器中未定义指令。将来它将用于异常处理。

•alu_op：ALU运算的类型。

•rs_ent：保留站ID。每个执行单元都包含一个保留站，其ID定义如下：

– ALU：1

–BRANCH UNIT（分支单元）：2

–MULTIPLIER乘法器：3

– LOAD / STORE：4

•dmem_size，dmem_type：确定装载/存储数据的大小（4字节/ 2字节/ 1字节）。不使用这些信号，因为RIDECORE仅支持4字节加载/存储指令。 •md_req_op：操作类型（乘法或除法或模数）.这个信号没有使用因为TIDECORE 不支持除法操作和取模运算

•md_req_in_1_signed，md_req_in_2_signed：这些信号指示乘法器的第一个和第二个源操作数是否带符号。

•md_req_out_sel：选择高/低。在RIDECORE中，乘法器的输出为64位，而ARF和RRF中的每个寄存器均为32位。该信号确定选择乘法器输出的哪一部分作为最终乘法结果：高32位还是低32位。

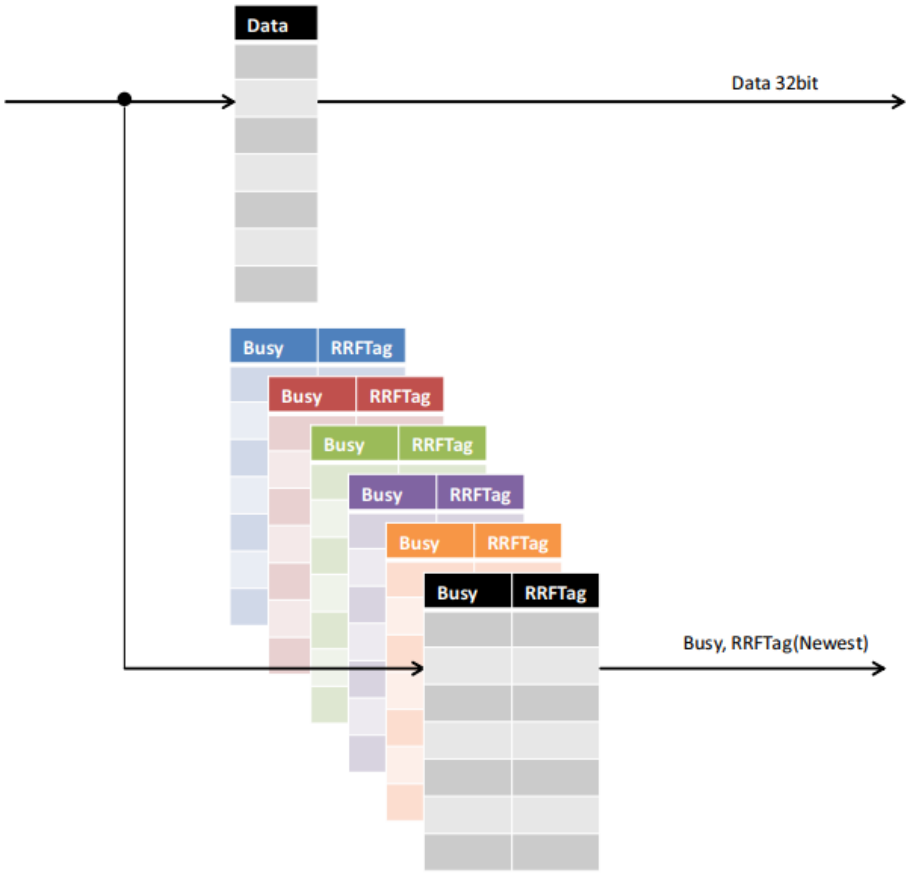


Fig.8 ARF Read

7、ARF（体系结构寄存器）

体系结构寄存器组 （ARF） 的每个条目都包含一个在 COM 阶段更新的寄存器（数据）以及一些重命名信息（RRFTag、Busy）。图 8、9、10 显示了 ARF 的电路和行为。重命名表包含重命名信息。发生分支错误预测时，必须将其还原到预测前的状态。因此，重命名表的数量等于推测标记的数量。下面我们将解释 ARF 的读取和写入操作，以及在发生分支错误预测时如何还原重命名表。

图 8 显示了 ARF 的读取行为。最新重命名表（黑色表）中的数据用作当前重命名信息。图 9 显示了 ARF 的写入行为。写入请求在 DP 阶段和 COM 阶段生成。我们重命名表的写入使能信

号是预测修复表 （mpft） 中有效矢量的补充。有关 mpft 的更多说明，请阅读"预测错误修复表"部分。

图 10 显示了重命名表的还原行为。当分支指令位于分支单元中并计算分支条件时，将执行还原。重命名表中的 RRFTag 和"Busy"实现为 32 位矢量，因此可以在一个周期内重写所有条目。在还原进行期间，流水线将停止运行，因为无法正确读取重命名表。

图 11 显示了还原行为的示例。如果对分支 1 的预测（推测标记 = 5'b00010）不正确，我们不得不恢复所有重命名表的信息到状态（推测标记 = 5'b00001）。因此，修复矢量设置为 6'b111111，红色重命名表被选为修复数据。

另一方面，如果分支 1 的预测正确，则备份表的内容（红色表，推测标记 = 5'b0001）将变得没有必要。我们使用黑色表中的最新信息覆盖这些内容以备份当前状态。因此，修复矢量设置为 6'b010000，黑色重命名表被选为修复数据。

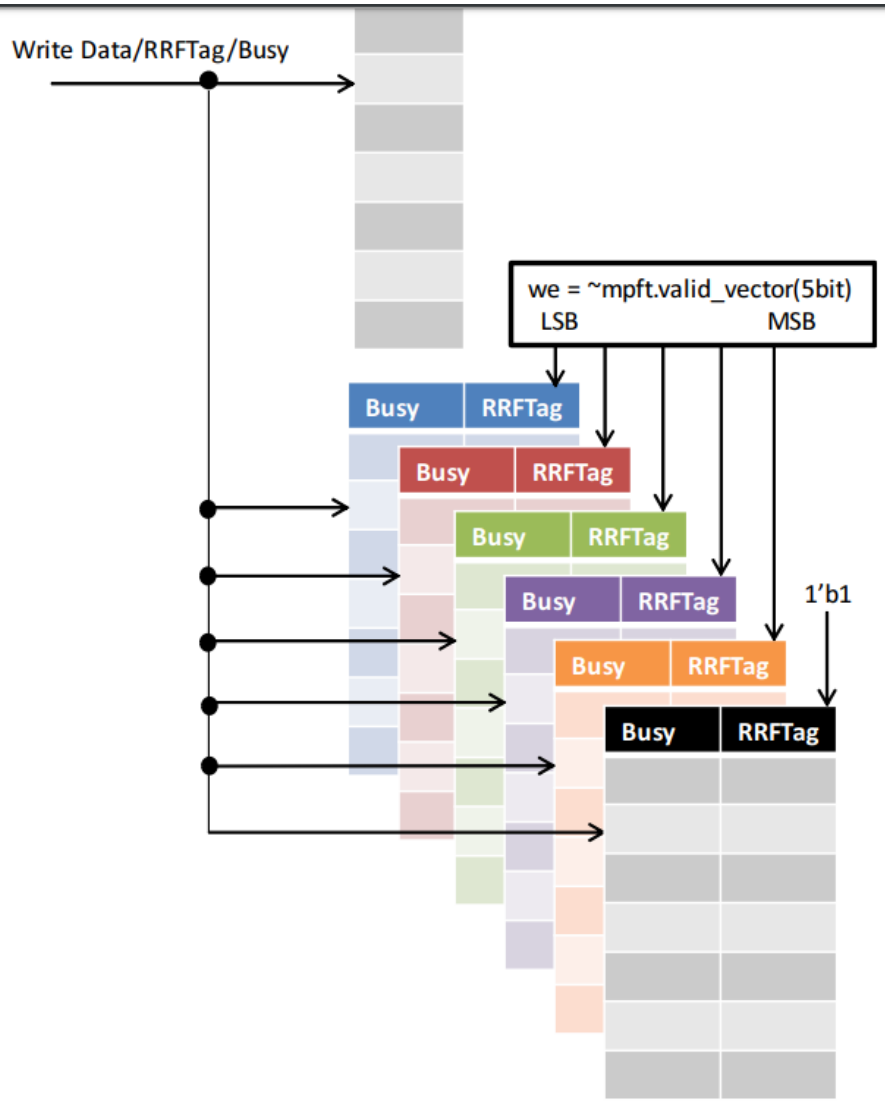


Fig.9 ARF Write

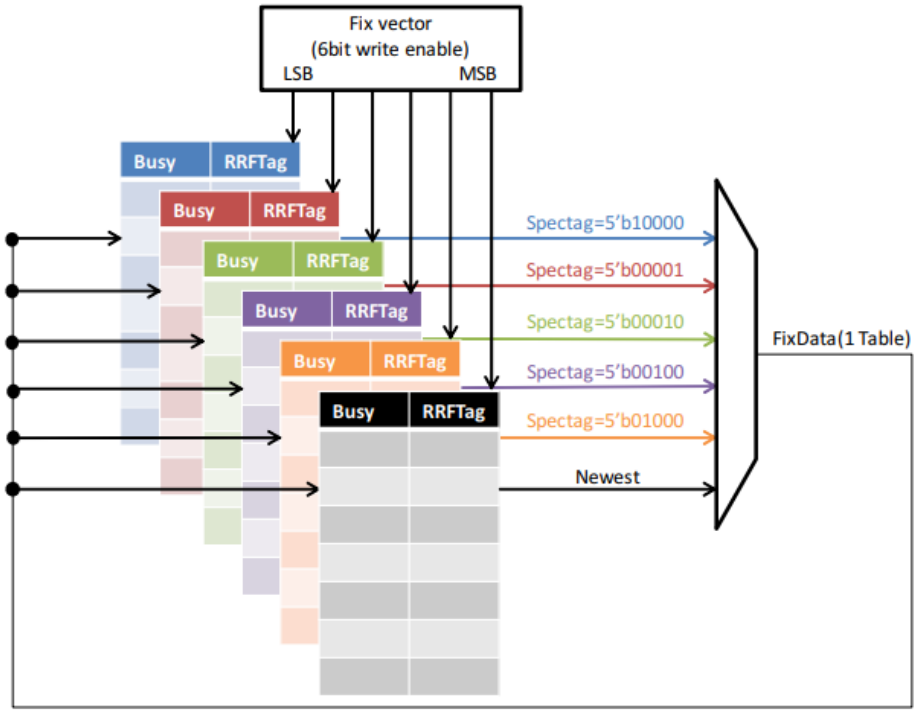


Fig.10 Fix Renaming Table

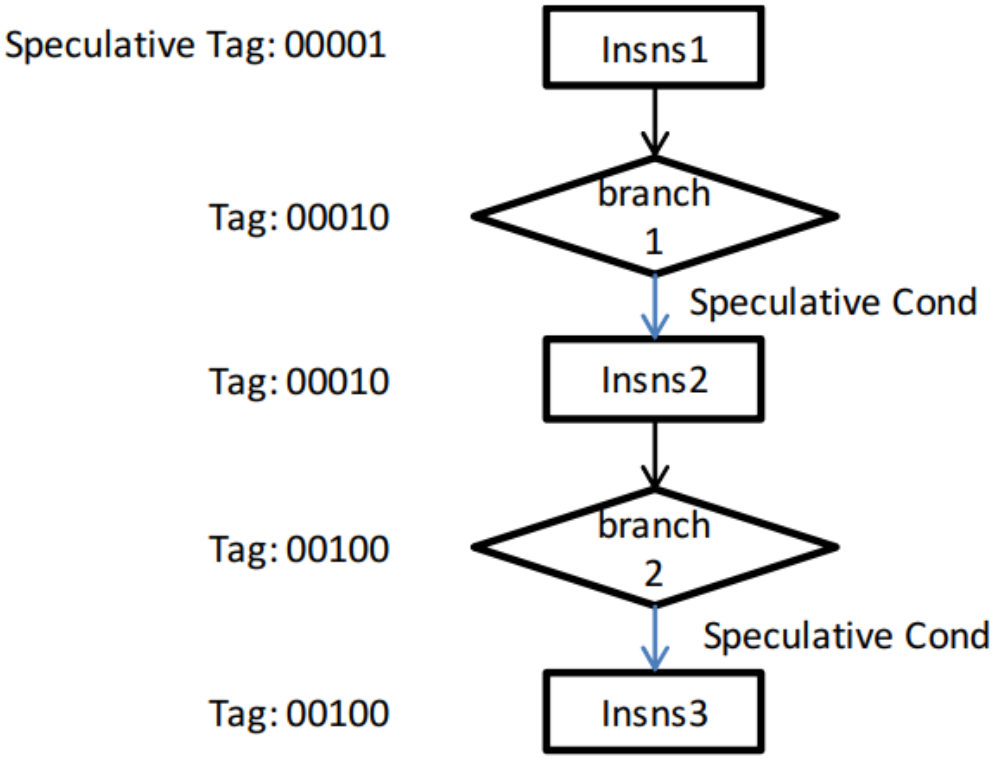


Fig.11 Example of Speculative Exection

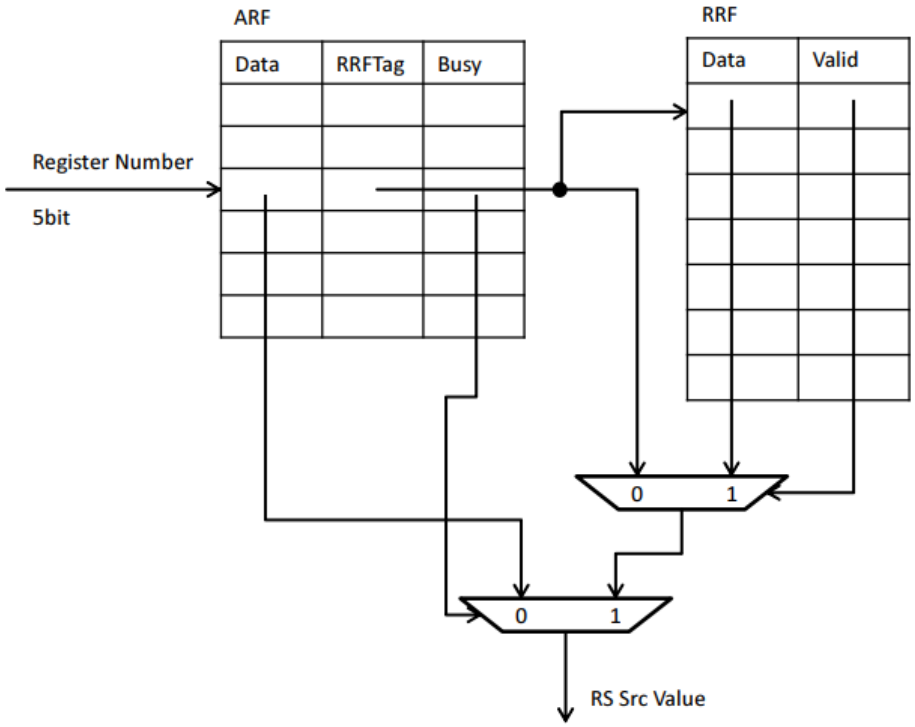


Fig.12 Register Renaming using ARF and RRF

8、RRF（pp.239-244）

重命名寄存器组 （RRF） 的表项有两个属性：RRFData 和 RRFValid。RRFData 包含重命名寄存器的值（超标量设计中还有一项Busy）。RRFValid 确定该条目是否有效。在 RIDECORE 中，RRF 的大小为 64 条目，与重新排序缓冲区的大小相同。事实上，RRF 和重新排序缓冲区之间存在一对一的对应关系。因此，用于访问 RRF 的标记也可用于访问重新排序缓冲区。 (具体参考《超标量》P-99)

9、源操作数管理（pp,239_244）

sourceoperand-manger将产生的数据写入到保留站，包含请求的寄存器号，ARF和RRF的信息，inst1的RRFTag。图片12展示了使用ARF和RRF实现使用寄存器重命名的电路。

Table 2 Three possible states of a requested register and the data dispatched to Reservation Station in each case.

ARF.Busy	RRF.Valid	State	Write data to RS
0	*	Available in ARF	ARF.Data
1	1	Available in RRF	RRF.Data
1	0	Not available	RRFTag

```
module sourceoperand_manager
  (
   input wire [`DATA_LEN-1:0]  arfdata,
   input wire                  arf_busy,
   input wire                  rrf_valid,
   input wire [`RRF_SEL-1:0]   rrftag,
   input wire [`DATA_LEN-1:0]  rrfdata,
   input wire [`RRF_SEL-1:0]   dst1_renamed,
   input wire                  src_eq_dst1,
   input wire                  src_eq_0,
   output wire [`DATA_LEN-1:0] src,
   output wire                 rdy
   );

   assign src = src_eq_0 ? `DATA_LEN'b0 :
                src_eq_dst1 ? dst1_renamed :
                ~arf_busy ? arfdata :
                rrf_valid ? rrfdata :
                rrftag;
   assign rdy = src_eq_0 | (~src_eq_dst1 & (~arf_busy | rrf_valid));

endmodule // sourceoperand_manager
```

Fig.13 Source Operand Manager

表2列出了每种情况下请求的寄存器和分派到保留站的数据的三种可能状态。假设请求的寄存器是regsrc。它在ARF和RRF中的状态分别为ARF.Busy和RRF.Busy。当且仅当regsrc不是任何未完成指令的目标寄存器时，ARF.Busy才等于0。在这种情况下，ARF中提供了regsrc的最新值。另一方面，如果ARF.Busy等于1，则存在带有目标寄存器regsrc的未完成指令。 RRF.Valid确定此指令的EX阶段是否已完成。 EX阶段完成后，将结果写入RRF和RRF.Valid设置为1。当RRF.Valid等于零时，即EX阶段的执行尚未完成，RRFTag会发送到RS，以便稍后将最新的regsrc值从RRF传输到RS（使用regsrc作为目的寄存器的指令的所有阶段都已经完成）。

图 13 显示了源操作管理器的源代码。src 是发送到保留站的数据。当 rdy 等于零时，src 设置为 RRFTag。当请求的寄存器号为 0 时，src 设置为零（如 MIPS ISA，RISC-V ISA 定义一个特殊的零寄存器 ）。当inst2中一个源寄存器是inst1中的目的寄存器的时候（src eq dst1 is equal to one），src被设置为inst1的RRFTag，除了这些情况外，该模块的操作如表2所示。

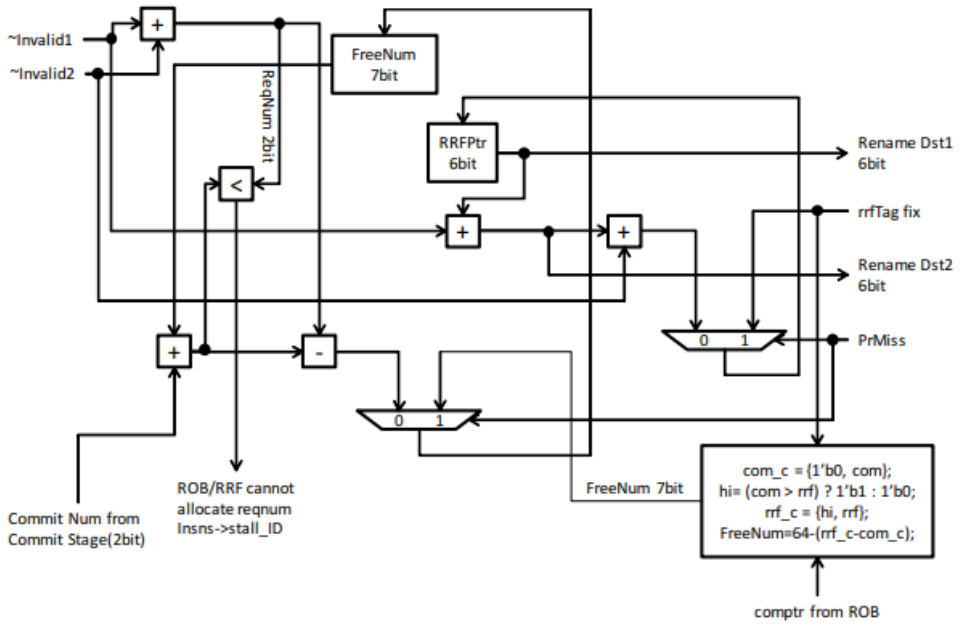


Fig.14 RRF Free List Manager

10、rrf freelistmanager (pp. 239-244)

图14显示了管理free entries of RRF和重定序缓冲的rrf_freelistmanger的电路。 有两个寄存器：FreeNum 和 RRFPtr. FreeNum 存储free entries的数量。RRFPtr 是 RRF 和保留站的当前入口编号。

此电路将 RRFTag 分配给 DP 阶段中的指令。当 FreeNum 小于请求写入保留站的指令数时，IF 阶段和 ID 阶段都将停止，因为无法调度任何指令。在 COM 阶段完成指令时，将释放此指令使用的 RRF 标记。此外，FreeNum也会递增。发生分支错误预测时，也会释放所有推测指令的 RRFTags。为此，RRFPtr被设置为rrfTag来修正导致分支错误预测的下一条分支指令的rrfTag。FreeNum也可以通过使用rrfTag fix和来自Reorder Buffer的comptr来恢复。

```
module src_manager
  (
   input wire [`DATA_LEN-1:0]  opr,
   input wire                  opr_rdy,
   input wire [`DATA_LEN-1:0]  exrslt1,
   input wire [`RRF_SEL-1:0]   exdst1,
   input wire                  kill_spec1,
   input wire [`DATA_LEN-1:0]  exrslt2,
   input wire [`RRF_SEL-1:0]   exdst2,
   input wire                  kill_spec2,
   input wire [`DATA_LEN-1:0]  exrslt3,
   input wire [`RRF_SEL-1:0]   exdst3,
   input wire                  kill_spec3,
   input wire [`DATA_LEN-1:0]  exrslt4,
   input wire [`RRF_SEL-1:0]   exdst4,
   input wire                  kill_spec4,
   input wire [`DATA_LEN-1:0]  exrslt5,
   input wire [`RRF_SEL-1:0]   exdst5,
   input wire                  kill_spec5,
   output wire [`DATA_LEN-1:0] src,
   output wire                 resolved
   );

   assign src = opr_rdy ? opr :
                ~kill_spec1 & (exdst1 == opr) ? exrslt1 :
                ~kill_spec2 & (exdst2 == opr) ? exrslt2 :
                ~kill_spec3 & (exdst3 == opr) ? exrslt3 :
                ~kill_spec4 & (exdst4 == opr) ? exrslt4 :
                ~kill_spec5 & (exdst5 == opr) ? exrslt5 : opr;

   assign resolved = opr_rdy |
                     (~kill_spec1 & (exdst1 == opr)) |
                     (~kill_spec2 & (exdst2 == opr)) |
                     (~kill_spec3 & (exdst3 == opr)) |
                     (~kill_spec4 & (exdst4 == opr)) |
                     (~kill_spec5 & (exdst5 == opr));

endmodule // src_manager
```

Fig.15 Source Manager

11、src manager (pp. 256-259)

图 15 显示了 Verilog HDL 中的源代码。此模块用于在 DP 阶段转发结果，并在预留站中等待所需的操作数。在这里，我们解释它的输入/输出。

•opr: DP阶段的操作数(从sourceoperand -manager发送到保留站)。如果opr-rdy等于0，则opr包含RRFTag，它生成所需的值。否则，opr包含所需的值。（操作数就位就直接取操作数，否则是操作数的来源标签）

•opr_rdy:决定在opr中存储什么。

• exrslt *，exdst *，kill _spec *：五个执行单元（ALU1 / 2，加载/存储单元,分支单元，乘法）的输出，当exdsti等于opr并且opr _rdy和∽kill spec *都等于零时，src设置为exrslti。

•src，resolved：opr，只要有可能，就将opr rdy转发给这些信号。

12、imm gen, brimm gen

在RISC-V ISA中，某些指令在其操作数内包含立即值。 imm-gen和brimm-gen从解码数据（imm类型）和指令数据生成32位带符号立即数。 brimm gen用于分支指令，而imm gen用于其他指令。

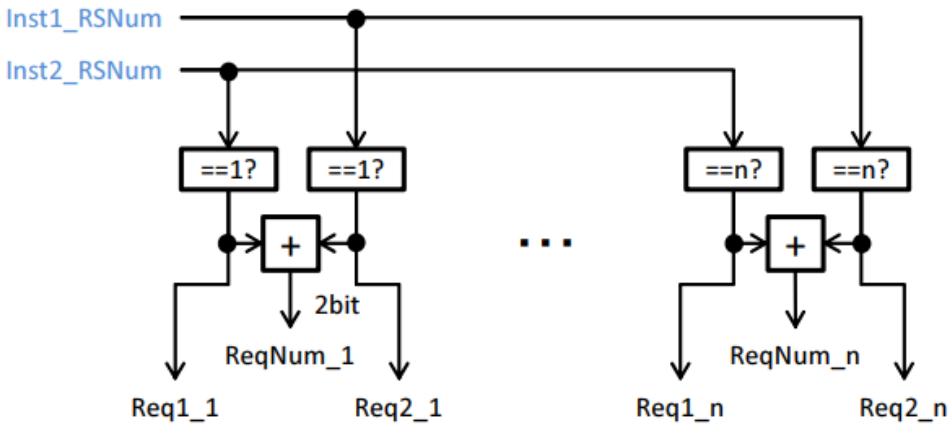


Fig.16 RS Request Generator

13、rs requestgenerator (pp. 254-259)

保留站接收两个指令（inst1/2）的数据以及两个写入使能信号。写使能信号由模块 rs-requestgenerator 产生。图16显

示了该模块的电路。该电路从译码器生成的RSNum1/2生成Req1/2(inst1的写使能信号)。

14、reservation station(rs *) (pp. 199-203, 254-261)

保留站（RS） 负责获取和存储寄存器指令的操作数。 每个执行单元都有一个保留站。分配单元向保留站分配指令。发射单元向执行单元发射来自保留站站的指令。

图17显示了保留站的I/O及其与分配单元和发射单元之间的接口。当分配单元接收写使能信号 （we1/2 ） 时，它会在保留站中查找空闲表项并生成写入地址 （waddr1/2 ）。当保留站中有指令准备执行，并且相应的执行单元在下一个周期中不忙时，发射单元向执行单元发出指令。发出单元生成指令 （raddr） 的输入编号（对应的执行功能单元），保留站将数据 （rdata） 输出到执行单元。有关分配/发射操作的更深入说明，请参阅下一节（分配单元和发射单元）。

图 18 显示了保留站入口的电路。橙色方块表示寄存器。写入数据是需要执行的寄存器指令（操作数、操作码、RRFTag 等）所需的数据。执行结果通过 Src 管理器转发到两个寄存器 Opr1 和 Opr2。以下是保留站记录的说明。

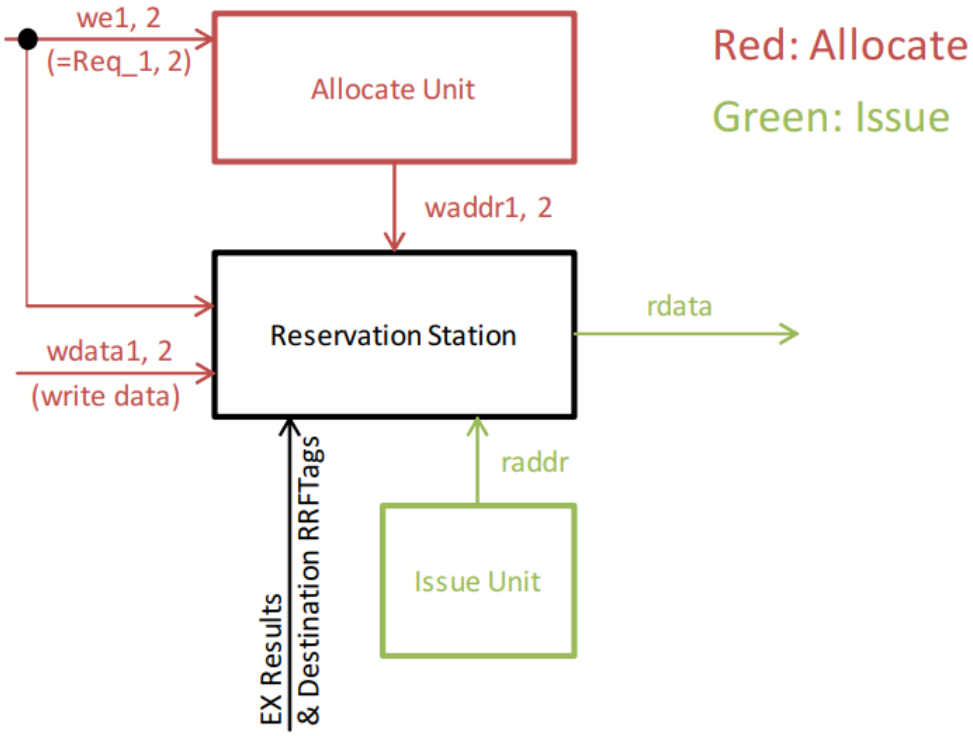


Fig.17 Reservation Station (with Allocate Unit and Issue Unit)

• Opr1，Opr2：第一和第二寄存器操作数。这些数据有不同的含义，具体取决于有效位。

•valid1/2 = 1：Opr1/2 准备就绪。当指令不使用 Opr1/2 时，valid1/2 也变为 1。

• valid1/2 = 0：Opr1/2 仍未准备好执行指令。在这种情况下，Opr1/2 包含生成所需操作数的 RRFTag。

•Busy：确定指令是否已登记。

•其他数据：执行指令所需的其他数据。

– imm：立即数值。

– rrftag：RRFTag。

– dstval：确定指令是否需要将数据写入ARF。

– specbit，spectag：specbit确定指令是否为推测的，而spectag包含指令分配的推测性标记。

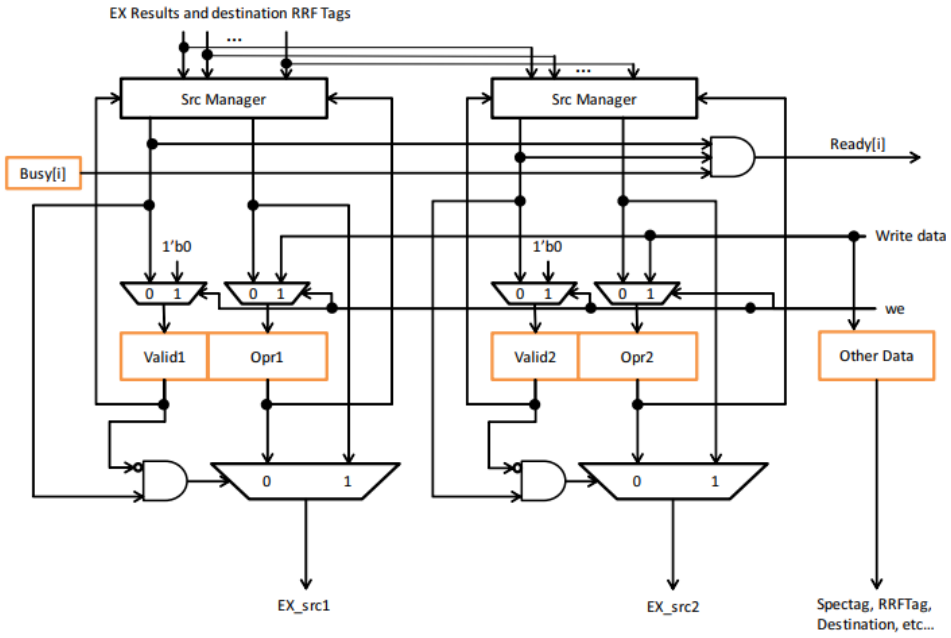


Fig.18 An entry in Reservation Station

Table 3 Allocate Pattern

Req1	Req2	waddr1	waddr2
0	0	*	*
1	0	Free_ent1	*
0	1	*	Free_ent1
1	1	Free_ent1	Free_ent2

15、Allocate Unit and Issue Unit (pp. 254-261)

15.1 Out-of-Order Issue

图 19 显示了分配单元的电路。图 20 显示了分配单元中RS 空闲表项查找的电路。分配单元，其功能类似于保留站的地址解析器。首先，分配单位接收写请求 Req1/2 到保留站。然后，

该单元使用 RS 空闲表项查找器在在保留站查找空闲表项（Rree_ent1/2），并将指令数据写入找到的空闲表项。

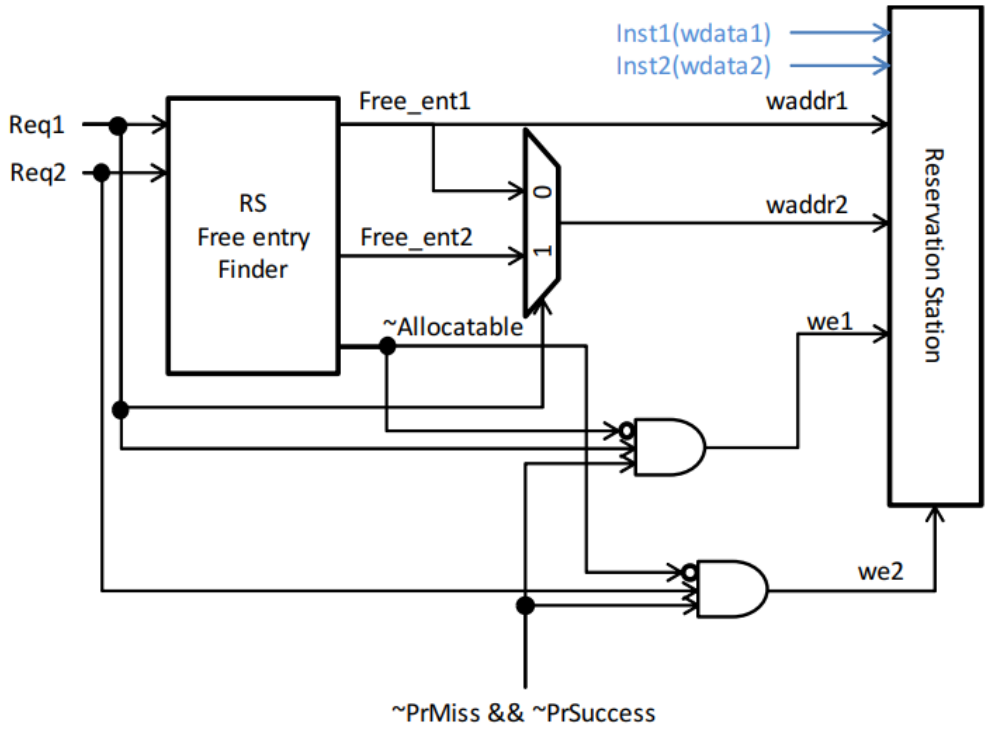


Fig.19 Allocate Unit

RIDECORE中保留站的接口与四端口内存类似（两个读取端口和两个写入端口）。 wdata1 / 2，waddr1 / 2和we1 / 2分别是写数据，写地址和写使能信号。表3显示了如何确定waddr1和waddr1。

RS Free Entry Finder通过使用两个优先级编码器查找最多两个空闲表项。第一优先级编码器的输入是对来自保留站的Busy vector的取反（∽Busy0-∽BusyN-1）。为避免选择第一优先级编码器已经选择的空闲表项，第二优先级编码器的输入将被屏蔽单元屏蔽。第二优先编码器的输入被掩码单元屏蔽。Entry_en1/2决定Free_ent1/2是否有效。当Entry_en1 +Entry_en2不小于Req1 + Req2时，可分配项变为1。

发射单元选择一条所有操作数都已经就绪的指令发射到EX阶段。指令后发射后，立即将其从保留站中删除。我们使用最早的优先算法选择要发布的指令。（涉及到发射算法）图21显示了用于实现优先算法的最小值选择电路。

图22显示了oldest finder 表项的格式。 The MSB of the entry is ˜rdy，因为发射单元必须对具有所有必要操作数的指令进行优先级排序。由于RRFTag从0开始分配，并且随着时间的增加而增加，因此我们可以通过比较RRFTag来选择最早的指令。然而，这是不对的当RRFTag 溢出的时候。因此，如图22所示，我们在MSB和RRFTag之间插入一个sorting bit。将指令分派到保留站时，其sorting bit将被置位。当RRFTag溢出时，保留站中所有指令的sorting bit 将被清除。通过这种方式，发射单元基本上可以选择具有所有必要操作数的最早的指令。但是，有一个限制。在RIDECORE中，一个时钟周期内最多可以将两条指令发送到保留站。因此，当两条指令的RRFTag为63和0（由于RRFTag为6位，在变为63后溢出）时，RRFTag = 0的指令的sorting bit 也被清除。这意味着发行单元未严格实现最早优先算法。

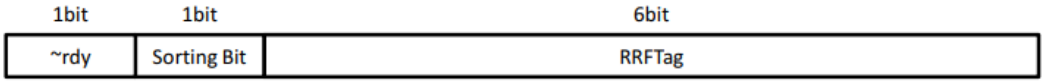


Fig.22 An entry in *oldest finder*

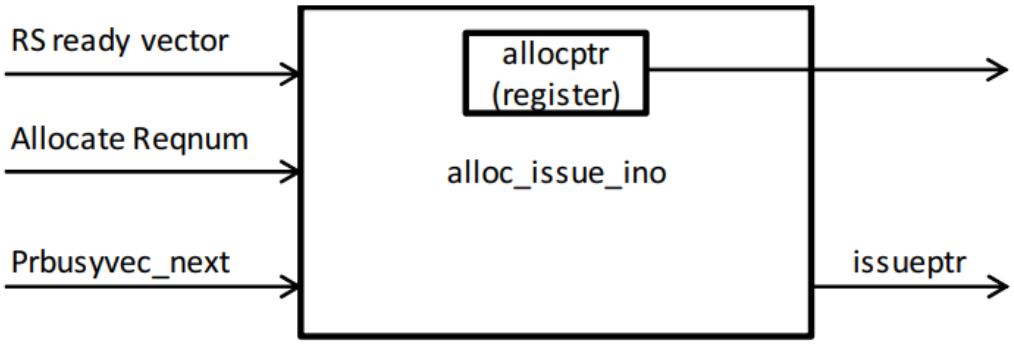


Fig.23 Allocate and Issue In Order

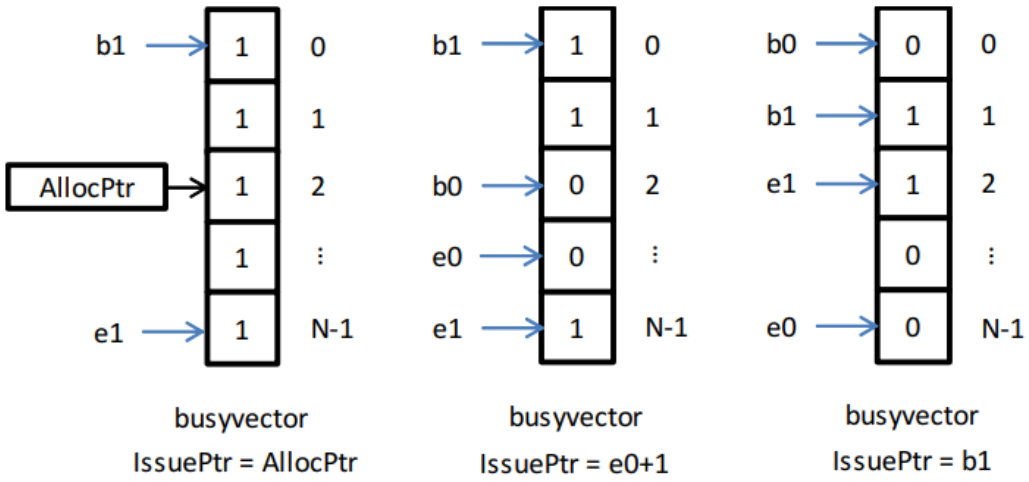


Fig.24 IssuePtr Calculation

15.2 In-Order Issue

图23示出了使用保留站作为FIFO缓冲器并实现有序执行的分配问题的电路。该电路中只有一个寄存器AllocPtr。 IssuePtr指向下一条要发射的指令，并根据保留站和寄存器AllocPtr的busy vector计算得出。

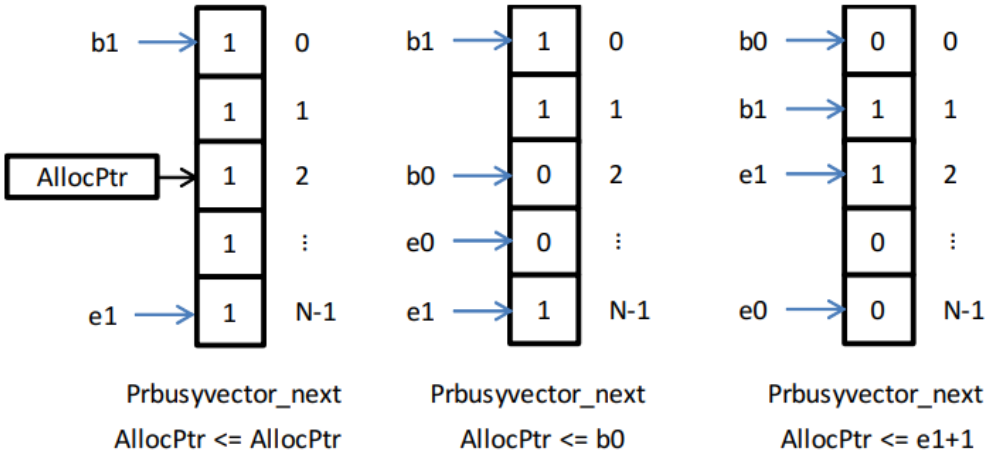


Fig.25 AllocPtr Re-calculation

图 24 显示了IssuePtr是如何计算的。图中的 b0/1、e0/1 分别通过搜索开始和搜索结束（稍后描述）计算。b0/1 是第一个0/1表项，而 e0/1 是最后一个0/1表项的表项号，in the busy vector,有三种模式的busy 导向。

•模式1:busy vector 的所有（记录）都等于1。在本例中，IssuePtr被设置为AllocPtr。

•模式2:b0和e0位于b1和e1之间。在本例中，IssuePtr被设置为e0 + 1。

•模式3:b1和e1位于b0和e0之间。在本例中，IssuePtr被设置为b1。

发生分支错误预测时，由于保留站中的某些推测性指令无效，因此必须重新计算AllocPtr。根据Prbusyvector重新计算AllocPtr，该向量是失效后的繁忙向量。此重新计算与IssuePtr的计算几乎相同

15.3 search begin, search end

搜索开始和搜索结束均被实现为优先级编码器。但是，搜索开始优先处理低位，而搜索结束优先处理高位。这些模块在alloc_issue_ino和storebuf中使用按序执行。

16、exection unit(exunit *) (pp. 203-206)

RIDECORE具有五个执行单元：两个ALU，一个乘法器，一个加载/存储单元和一个分支单元。图26，图27和图28示出了这些执行单元的电路。图29示出了在图26，图27和图28中使用的“ Kill Gen”电路。当发生分支错误预测时，Kill Gen确定是否使执行单元中当前正在执行的指令无效。有你可以理解这个电路通过读Missue Prediction Fix Table 章节。在RIDECORE中，每个乘法运算需要一个时钟周期。因此，乘法器电路与ALU几乎相同。

当执行单元完成指令的执行时，它会将完成通知（ROB WE）通知给Reorder Buffer，并将数据（RRF Write Addr / Data）写入RRF。除了这些基本操作之外，加载/存储单元和分支单元还执行以下一些其他操作。

加载/存储单元的操作分为两个阶段。当本机从保留站接收到加载指令时，它会在第一阶段访问D-MEM和存储缓冲区，并从存储缓冲区接收数据。在第二阶段，它从D-MEM接收数据并将数据（如果还有尚未存储到D-MEM的数据，则从Store Buffer写入数据；否则从DMEM写入数据）到RRF。

当加载/存储单元接收到存储指令时，它在第一阶段将存储数据和存储地址写入存储缓冲区，并在第二阶段将完成情况通知重新排序缓冲区。一旦存储指令完成其COM阶段，就可以将存储

缓冲区中的存储数据写入D-MEM。每当没有加载指令访问D-MEM时，会将存储缓冲区中的数据写入D-MEM。

分支机构计算分支目标并将其与IF阶段预测的分支目标进行比较。如果分支预测正确，我们将更新未命中预测修复表，并清除与推测标记匹配的推测比特位。另一方面，如果发生分支预测错误，我们将处理器的状态恢复为执行分支预测之前的状态。在两种情况下，流水线将会阻塞（用于备份或者还原）。在COM阶段，为了将分支指令的完成通知分支预测器，我们需要该指令的信息。因此，分支单元将分支目标和分支条件写入重排序缓冲器。

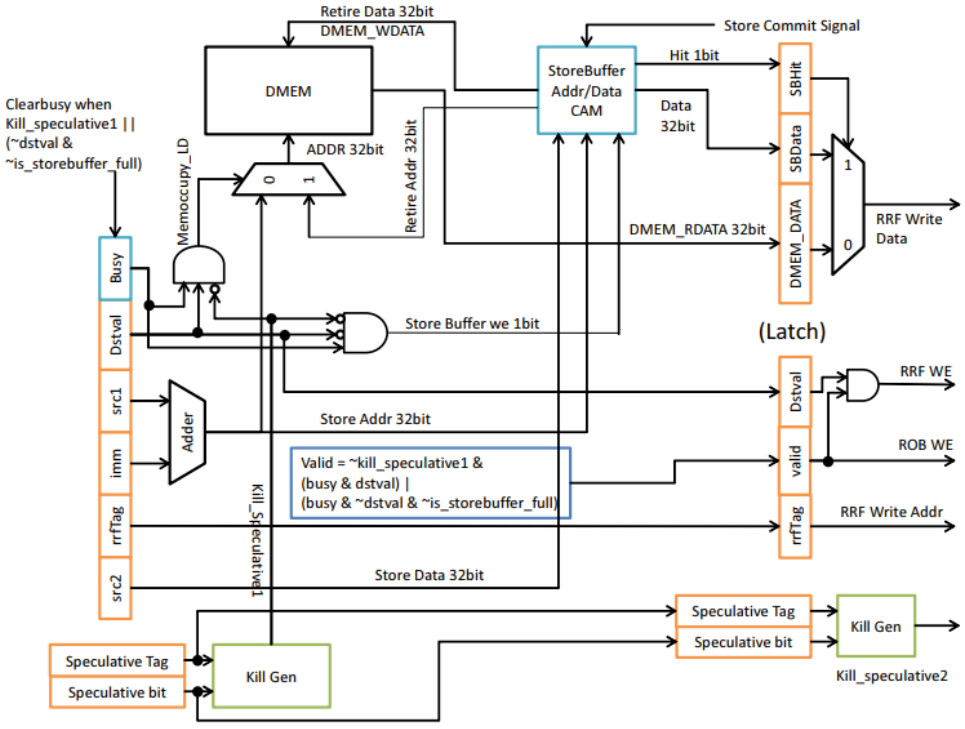


Fig.27 Load/Store Unit

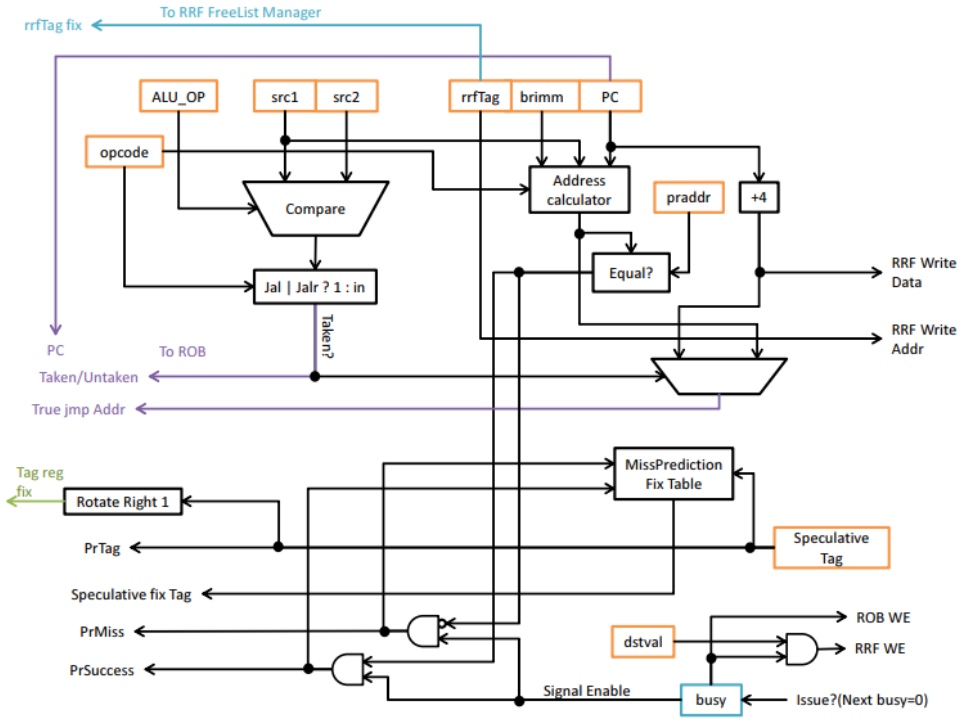


Fig.28 Branch Unit

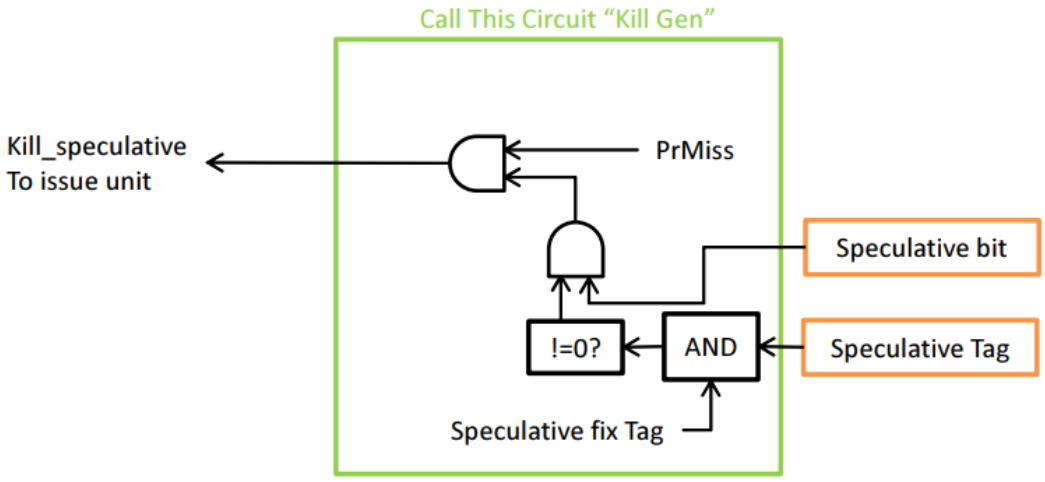
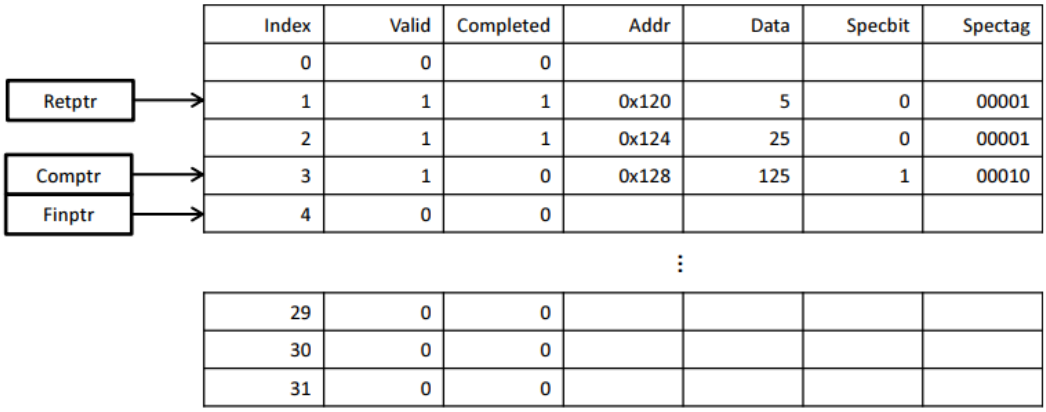


Fig.29 Kill Gen



Index	Valid	Completed	Addr	Data	Specbit	Spectag
0	0	0				
1	1	1	0x120	5	0	00001
2	1	1	0x124	25	0	00001
3	1	0	0x128	125	1	00010
4	0	0				
⋮						
29	0	0				
30	0	0				
31	0	0				

Fig.30 Store Buffer

17、storebuf (pp. 206-209, 262-273)

存储缓冲区包含已完成EX阶段的存储数据和存储指令的存储地址。一旦指令完成其COM阶段，就可以将存储指令的存储数据从存储缓冲区传输到DMEM。存储缓冲区是一个关联存储器。当加载指令将加载地址提供给存储缓冲区时，它将最新的未存储数据返回到加载指令。可以通过旋转加载地址命中矢量轻松选择最新的未存储数据。图30显示了存储缓冲区的表项和寄存器。一共有三个寄存器：Finptr，Comptr和Retptr。 Finptr是完成指令的指针。 Comptr是完成指令的指针。最后，Retptr指向其存储数据已写入DMEM的指令。当发生分支错误

预测时，只需要增加comptr和retptr即可，但是需要以重新计算alloc_issue_ino的相同方式重新计算finptr。

下面是对存储缓冲区中每个条目的属性的解释。

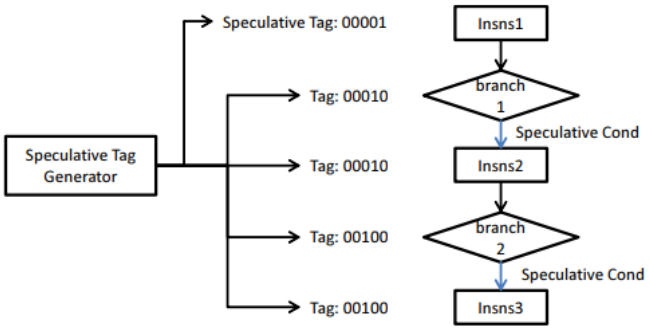
• valid：确定该条目是否有效。

• completed：确定指令是否完成。一旦有效位和完成位等于1，该指令已准备好退役。 #### • addr、data：要写入DMEM 的数据和地址。

• specbit，spectag：specbit确定指令是否是推测的。spectag包含推测标记。当分支预测不正确时，需要这些信息使指令无效。

18、miss prediction fix table (pp. 228-231)

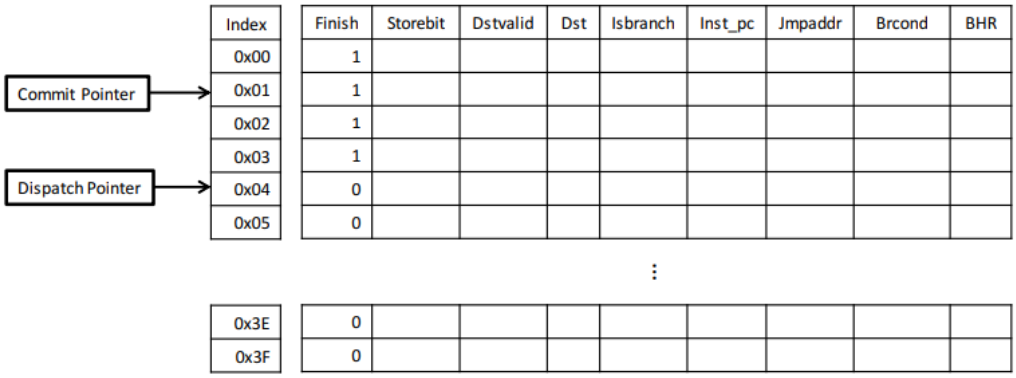
未命中预测修订表存储推测标记的状态。推测标记的状态被两个值定义，Valid和Value。Valid确定具有此推测标签的指令是否为推测指令。Value显示此推测性标签与其他推测性标签之间的依赖性



Miss Prediction Fix Table

Index	Value	Valid
00001	00000	0
00010	00110	1
00100	00100	1
01000	00000	0
10000	00000	0

Fig.31 Speculative Exection



Index	Finish	Storebit	Dstvalid	Dst	Isbranch	Inst_pc	Jmpaddr	Brcond	BHR
0x00	1								
0x01	1								
0x02	1								
0x03	1								
0x04	0								
0x05	0								
⋮									
0x3E	0								
0x3F	0								

Fig.32 Reorder Buffer

接下来，我们使用一个特定的示例（图31）来解释Miss Prediction Fix Table。图31示出了对两个分支指令进行解码之后的指令流和未命中预测修复表的状态。推测性标签生成器将推测性标签按照00001、00010，...，10000的顺序分配给指令。当分支1的预测结果不正确时，我们读取Miss中索引00010（分支1的推测性标签）的值预测修复表。值是00110。那么，要无效的指令是带有推测标记00010和00100的指令。

∀i ∈ Instructions (((i.SpeculativeT ag & V alue) ! = 0) → i is invalidated)

19、reorderbuf (pp. 206-209, 254-259)

重排序缓冲区是允许按顺序提交指令的缓冲区。图32示出了重排序缓冲器的条目和寄存器。 comptr是一个指针，指向下一

步要完成的指令。 dispatchptr，与rrf_freelistmanager中的RRFPtr相等，是指向其下一条指令将被分派的条目的指针。每个时钟周期，重排序缓冲区最多可以完成两条指令。但是，分支指令和存储指令是一一完成的，以减少分支预测器和存储缓冲区中存储器的写端口数量。

指令完成后，将重命名表更新，并将以RRF编写的指令的执行结果复制到ARF。仅当重命名表的RRFTag等于已完成指令的RRFTag时，才清除“重命名表中的Busy位”。

下面是重新排序缓冲区中每个条目的属性的说明。

•finish：指示指令是否已完成。在分派指令时清除，并在指令完成时设置。

•storebit：指示该指令是否为存储指令。如果storebit等于1（即store指令），则必须将指令的完成通知Store Buffer。

•dstvalid：指示指令的目标寄存器是否有效，即指令是否需要将数据写入ARF。

•dst：目标寄存器号。

•isbranch：确定指令是否为分支指令。如果isbranch等于1（即转移指令），则必须在指令完成后将数据写入转移预测器（PHT，BTB等）。

•inst pc，jmpaddr，brcond，bhr：数据到Branch Predictor。 inst pc，jmpaddr，brcond，bhr分别是指令地址，分支目标，分支条件和bhr（用于计算PHT的写地址）。

References

[1] John Paul Shen, Mikko H. Lipasti, Modern Processor Design: Fundamentals of Superscalar Processors, 2013.

[2] McFarling, Scott: Combining branch predictors, Technical Report TN-36, Digital Western Research Laboratory, (1993)

[3] RISC-V, http://riscv.org/ (2016/01/20)