**Biriscv技术文档**

2023年8月3日

温榆河项目组

**目 录**

[1 处理器 1](#_Toc142052794)

[1.1 简介 1](#_Toc142052795)

[1.2 Biriscv微架构 1](#_Toc142052796)

[1.3 前端 2](#_Toc142052797)

[1.3.1 分支预测单元 2](#_Toc142052798)

[1.3.2 取指单元 2](#_Toc142052799)

[1.3.3 译码单元 2](#_Toc142052800)

[1.3.4 指令缓存 2](#_Toc142052801)

[1.4 后端 3](#_Toc142052802)

[1.4.1 发射单元 3](#_Toc142052803)

[1.4.2 执行单元 3](#_Toc142052804)

[1.5 访存系统 3](#_Toc142052805)

[1.5.1 LSU 3](#_Toc142052806)

[1.5.2 MMU 4](#_Toc142052807)

[1.5.3 数据缓存 4](#_Toc142052808)

[2 分支预测单元 5](#_Toc142052809)

[2.1 简介 5](#_Toc142052810)

[2.1.1 Next PC判定逻辑 6](#_Toc142052811)

[3 取指单元 7](#_Toc142052812)

[3.1 简介 7](#_Toc142052813)

[4 译码单元介绍 8](#_Toc142052814)

[4.1 简介 8](#_Toc142052815)

[5 执行单元介绍 9](#_Toc142052816)

[5.1 简介 9](#_Toc142052817)

[6 发射单元介绍 10](#_Toc142052818)

[6.1 简介 10](#_Toc142052819)

[6.1.1 PC单元 10](#_Toc142052820)

[6.1.2 发射选择单元 10](#_Toc142052821)

[6.1.3 指令译码单元 11](#_Toc142052822)

[6.1.4 发射逻辑单元 12](#_Toc142052823)

[6.1.5 流水线控制单元 15](#_Toc142052824)

[6.1.6 分支预测更新单元 21](#_Toc142052825)

[7 MMU单元介绍 23](#_Toc142052826)

[8 指令缓存单元介绍 24](#_Toc142052827)

[8.1 简介 24](#_Toc142052828)

[8.1.1 TAG RAM 24](#_Toc142052829)

[8.1.2 Data RAM 25](#_Toc142052830)

[9 数据缓存单元介绍 26](#_Toc142052831)

[9.1 简介 26](#_Toc142052832)

[10 处理器状态转换 28](#_Toc142052833)

[10.1 M模式寄存器 28](#_Toc142052834)

[10.1.1 Machine ISA Register (misa) 28](#_Toc142052835)

[10.1.2 Machine Vendor ID Register (mvendorid) 30](#_Toc142052836)

[10.1.3 Machine Architecture ID Register (marchid) 30](#_Toc142052837)

[10.1.4 Machine Implementation ID Register (mimpid) 30](#_Toc142052838)

[10.1.5 Hart ID Register (mhartid) 30](#_Toc142052839)

[10.1.6 Machine Status Registers (mstatus) 31](#_Toc142052840)

[10.1.7 Machine Trap-Vector Base-Address Register (mtvec) 34](#_Toc142052841)

[10.1.8 Machine Trap Delegation（medeleg/mideleg） 35](#_Toc142052842)

[10.1.9 Machine Interrupt Register（mip/mie） 36](#_Toc142052843)

[10.1.10 Hardware Performance Monitor 37](#_Toc142052844)

[10.1.11 Machine Counter-Enable Register（mcounteren） 38](#_Toc142052845)

[10.1.12 Machine Counter-Inhibit Register（mcountinhibit） 38](#_Toc142052846)

[10.1.13 Machine Scratch Register（mscratch） 39](#_Toc142052847)

[10.1.14 Machine Exception Program Counter (mepc) 39](#_Toc142052848)

[10.1.15 Machine Cause Register (mcause) 40](#_Toc142052849)

[10.1.16 Machine Trap Value Register (mtval) 43](#_Toc142052850)

[10.1.17 Machine Configuration Pointer Register (mconfigptr) 44](#_Toc142052851)

[10.1.18 Machine Environment Configuration Registers (menvcfg and menvcfgh) 44](#_Toc142052852)

[10.1.19 Machine Security Configuration Register (mseccfg) 45](#_Toc142052853)

[10.1.20 Machine Timer Register(mtime and mtimecmp) 46](#_Toc142052854)

[10.2 S模式寄存器 48](#_Toc142052855)

[10.2.1 Supervisor Status Register (sstatus) 48](#_Toc142052856)

[10.2.2 Supervisor Trap Vector Base Address Register(stvec) 49](#_Toc142052857)

[10.2.3 Supervisor Interrupt Registers (sip and sie) 49](#_Toc142052858)

[10.2.4 Counter-Enable Register (scounteren) 50](#_Toc142052859)

[10.2.5 Supervisor Scratch Register (sscratch) 50](#_Toc142052860)

[10.2.6 Supervisor Exception Program Counter (sepc) 50](#_Toc142052861)

[10.2.7 Supervisor Cause Register (scause) 51](#_Toc142052862)

[10.2.8 Supervisor Trap Value (stval) Register 51](#_Toc142052863)

[10.2.9 Supervisor Environment Configuration Register (senvcfg) 52](#_Toc142052864)

[10.2.10 Supervisor Address Translation and Protection (satp) Register 52](#_Toc142052865)

[11 Biriscv实现 54](#_Toc142052866)

[11.1 M-mode 54](#_Toc142052867)

[11.1.1 csr\_mstatus 54](#_Toc142052868)

[11.1.2 csr\_mcause（mcause） 55](#_Toc142052869)

[11.1.3 csr\_mtval(mtval) 55](#_Toc142052870)

[11.1.4 csr\_mtvec(mtvec) 55](#_Toc142052871)

[11.1.5 csr\_mip（mip）,csr\_mie(mie) 55](#_Toc142052872)

[11.1.6 csr\_mscratch(mscratch) 56](#_Toc142052873)

[11.1.7 csr\_mcycle，csr\_mcycle\_h，csr\_mtimecmp(mcycle,mtimecmp) 56](#_Toc142052874)

[11.1.8 csr\_medeleg，csr\_mideleg(medeleg,mideleg) 56](#_Toc142052875)

[11.1.9 csr\_mepc(mepc) 56](#_Toc142052876)

[11.1.10 csr\_misa(misa) 56](#_Toc142052877)

[11.1.11 csr\_mhartid(mhartid) 57](#_Toc142052878)

[11.1.12 csr\_sepc（sepc） 57](#_Toc142052879)

[11.1.13 csr\_stvec(stvec) 57](#_Toc142052880)

[11.1.14 csr\_scause(scause) 57](#_Toc142052881)

[11.1.15 csr\_stval(stval) 57](#_Toc142052882)

[11.1.16 csr\_satp(satp) 58](#_Toc142052883)

[11.1.17 csr\_sscratch 58](#_Toc142052884)

[11.1.18 csr\_sie,csr\_sip 58](#_Toc142052885)

[12 局部中断控制器（CLINT） 59](#_Toc142052886)

[12.1 IPI中断 59](#_Toc142052887)

[12.1.1 寄存器 59](#_Toc142052888)

[12.1.2 中断处理过程 60](#_Toc142052889)

[13 中断 62](#_Toc142052890)

[13.1 时钟中断 62](#_Toc142052891)

[13.1.1 寄存器 62](#_Toc142052892)

[13.1.2 中断处理过程 63](#_Toc142052893)

[13.2 UART中断 64](#_Toc142052894)

[13.2.1 UART核发起中断请求 64](#_Toc142052895)

[13.2.2 CSR更改相应寄存器 64](#_Toc142052896)

[13.3 端口列表 66](#_Toc142052897)

[14 Wishbone总线 67](#_Toc142052898)

[14.1 时序 67](#_Toc142052899)

[附录1 信号列表 68](#_Toc142052900)

[附录2参考文献 74](#_Toc142052901)

[附录3 寄存器列表/属性 75](#_Toc142052902)

# 

# 处理器

## 简介

Biriscv处理器由Biriscv微架构 + DDR + UART + Debug Module + Bootrom组成。其中Biriscv微架构是In-order双发射结构设计，目前支持M扩展，Z扩展（RV32IMZicsr）。Biriscv处理器前端流水线包括分支预测单元，取指单元，译码单元，指令缓存单元。后端包括发射单元，执行单元。访存系统包括LSU，MMU，数据缓存。

主要功能：

* 64-bit取指，32-bit数据访问
* 2 x ALU单元
* 1 x LSU单元
* 1 x 非流水线除法单元
* 最高支持每周期发射并完成两条指令
* 支持User，Supervisor，Machine mode特权等级
* 1 x MMU单元
* 可boot Linux

## Biriscv微架构

A picture containing text, screenshot

Description automatically generated

图 1.1 微架构示意图

## 前端

### 分支预测单元

**输入端口：**branch\_info\_request\_i, branch\_info\_pc [31:0]

**输出端口：**next\_pc\_f\_o [31:0]

**单元功能介绍：**分支预测算法采用Gshare两级预测架构，其主要组成部分包括2比特饱和计数器，全局历史寄存器（Global history register）。其主要算法为：使用全局历史寄存器结果异或PC值，得到2比特饱和计数器的索引值，根据2比特饱和计数器读出的结果判定分支预测的结果。同时，利用PC值作为分支目标缓冲（BTB）的索引值，读出对应的分支跳转地址，输出到取值模块。

### 取指单元

**输入端口：**next\_pc\_f\_i [31:0], branch\_request\_i, branch\_pc\_i [31:0], icache\_inst\_i [63:0]

**输出端口：**fetch\_pc\_o [31:0], fetch\_instr\_o [63:0], icache\_pc\_o [31:0]

**单元功能介绍：**取指单元主要负责向处理器输送需要执行的指令 fetch\_instr\_o，取指单元首先收到来自分支预测单元的PC值 next\_pc\_f\_i，输出地址到MMU。如果branch\_request\_i 信号被拉高，此时说明分支预测错误，取值单元将接受来自执行单元的PC值branch\_pc\_i ，输出到MMU。收到来自MMU的指令后，输出到译码单元。

### 译码单元

**输入端口：**fetch\_pc\_instr\_i [31:0], fetch\_in\_instr\_i [63:0], branch\_request\_i

**输出端口：**fetch\_out0\_instr\_o [31:0], fetch\_out0\_pc\_o [31:0], fetch\_out1\_instr\_o [31:0], fetch\_out1\_pc\_o [31:0]

**单元功能介绍：**译码单元主要负责将来自取指单元的64比特指令拆拆封成两路32比特指令fetch\_out0\_instr\_o，fetch\_out1\_instr\_o，根据指令的opcode译码出该指令所对应的执行单元操作。将执行单元操作编译为1bit信号，输出到发射单元。

### 指令缓存

**输入端口：**axi\_rdata\_i[31:0],req\_pc\_i[31:0]

**输出端口：**fetch\_out\_pc\_o[31:0]

**单元功能介绍：**指令缓存的作用是缓存指令，将要执行的指令和该指令邻近的指令都读取到指令缓存中缓存起来，要执行指令时先去指令缓存中查找，没有找到再去内存中读取。其中axi\_rdata\_i 信号提供指令，req\_pc\_i 信号使用Tag进行比对，命中则将指令取出；未命中则请求写入，将cache中数据更新。fetch\_in\_inst\_o 信号将查找到的指令发送给MMU单元。本设计采用两路组相联，大小为16KB。

## 后端

### 发射单元

**输入端口：**fetch0\_instr\_i [31:0], fetch0\_pc\_i [31:0], fetch0\_instr\_i [31:0], fetch0\_pc\_i [31:0]

**输出端口：**opcode0\_ra\_operand\_o[31:0], opcode0\_rb\_operand\_o[31:0],

opcode1\_ra\_operand\_o[31:0], opcode1\_rb\_operand\_o[31:0], csr\_opcode\_ra\_operand\_o[31:0], csr\_opcode\_rb\_operand\_o[31:0], mul\_opcode\_ra\_operand\_o[31:0], mul\_opcode\_rb\_operand\_o[31:0], lsu\_opcode\_ra\_operand\_o[31:0], lsu\_opcode\_rb\_operand\_o[31:0]

**单元功能介绍：**发射单元主要由发射队列，相关性检查，旁路网络组成。指令在发射阶段涉及到入队，选择，读寄存器堆，出队等操作。指令在发射单元将在发射队列中等候唤醒，同时对指令源操作数进行相关性检查，并读取寄存器堆。如果此时源操作数未就绪，指令将在发射队列中等候旁路网络前递源操作数，源操作数就绪后，两条指令将并行进入流水线。

### 执行单元

**输入端口：**alu\_a\_i[31:0], alu\_b\_i[31:0], opcode\_ra\_operand\_i[31:0], opcode\_rb\_operand\_i[31:0]

**输出端口：**alu\_p\_o[31:0], alu\_p\_o[31:0], csr\_result\_e1\_value\_o[31:0], writeback\_value\_o[31:0],

**单元功能介绍：**执行单元由两个ALU单元，一个CSR单元，一个MUL单元组成。ALU单元主要执行整数运算操作，并将运算后的结果通过alu\_p\_o信号传输给旁路网络和寄存器堆。CSR单元主要负责执行CSR相关操作。在Briscv中，CSR操作将直接触发流水线停顿，操作将进行2-3个时钟周期。CSR操作的结果将通过csr\_result\_e1\_value\_o 写回到CSR寄存器堆。MUL单元主要负责执行乘法操作，操作将进行2-34个时钟周期。MUL结果将通过writeback\_value\_o写回。

## 访存系统

### LSU

**输入端口：**opcode\_ra\_operand\_i[31:0], opcode\_rb\_operand\_i[31:0], mem\_data\_rd\_i[31:0]

**输出端口：**writeback\_value\_o[31:0], mem\_addr\_o[31:0], mem\_data\_wr\_o[31:0]

**单元功能介绍：**LSU单元主要执行LOAD和STORE指令。LOAD指令时，LSU单元收到来自发射单元的源操作数opcode\_ra\_operand\_i 和 opcode\_rb\_operand\_i，计算出访存地址mem\_addr\_o并输出给MMU。访存结果将通过mem\_data\_rd\_i 端口输入到LSU单元，通过mem\_addr\_o 端口写回。STORE指令时，访存数据将通过mem\_data\_wr\_o 输出到MMU。

### MMU

**输入端口：**fetch\_out\_inst\_i[63:0],fetch\_in\_pc\_i[31:0], lsu\_in\_addr\_i[31:0], lsu\_out\_data\_wr\_w[31:0],lsu\_out\_data\_rd\_i[31:0]，

**输出端口：**req\_inst\_o[63:0],fetch\_out\_pc\_o[31:0], lsu\_in\_data\_rd\_o[31:0],lsu\_out\_addr\_o[31:0],lsu\_out\_data\_wr\_o[31:0]

**单元功能介绍：**虚实地址翻译。将输入虚拟地址与TLB页表进行比对，如果命中，则取出物理地址，若是未命中，则进入Page fault状态，对TLB进行update。其中MMU可分为IMMU和DMMU，IMMU功能为指令内存管理模块，DMMU为数据内存管理模块。

### 数据缓存

**输入端口：**mem\_addr\_i[31:0],mem\_data\_wr\_i[31:0]

**输出端口：**mem\_data\_rd\_o[31:0]

**单元功能介绍：**dcache功能与icache结构和功能相类似，但是dcache的作用是缓存数据，同样也会经过Tag比较，判命中最终Data阵列出数；未命中则请求写入，将cache中数据更新。

# 分支预测单元

## 简介

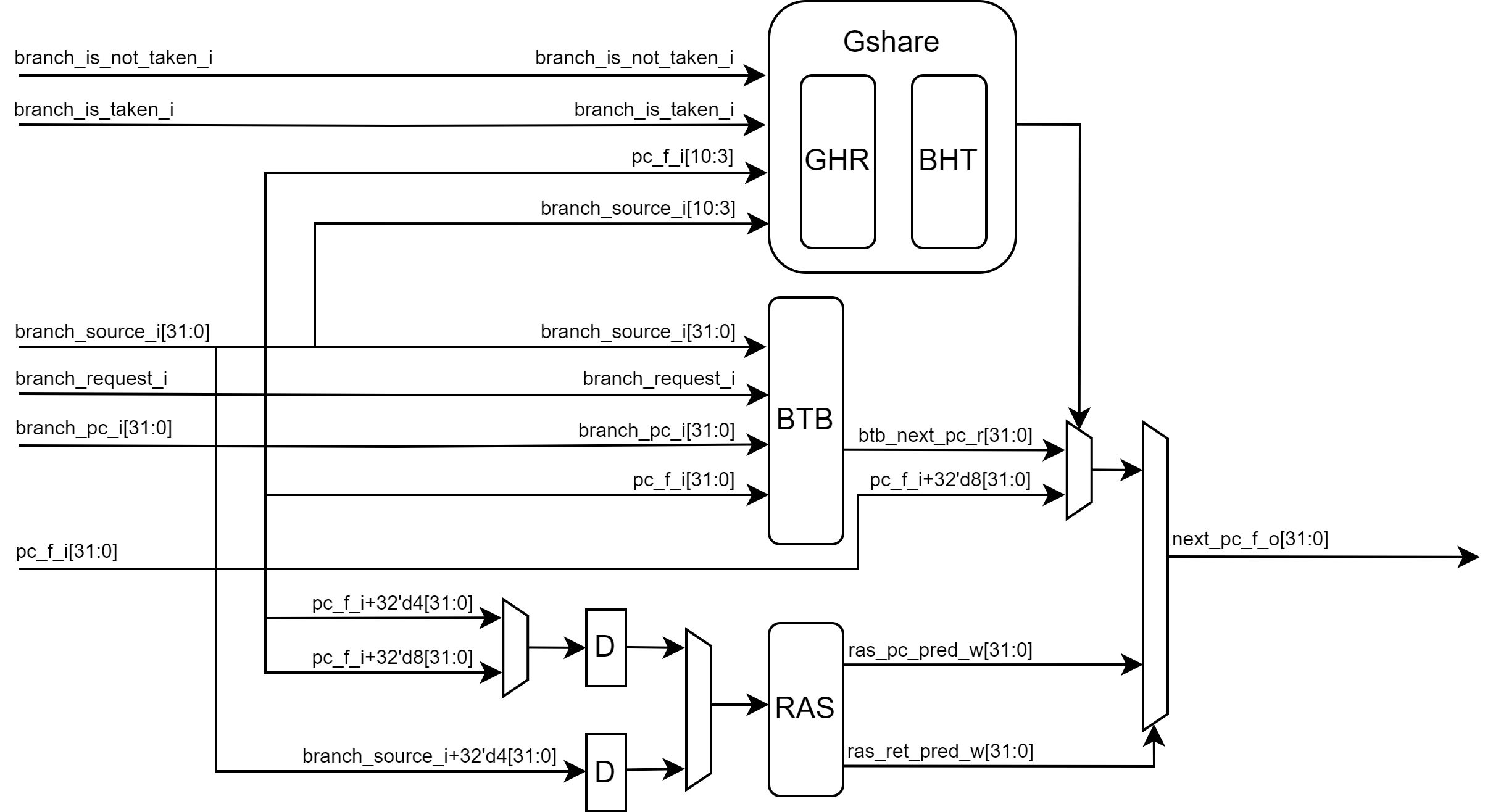


图 2.1 分支预测单元图示

在Biriscv的分支预测单元当中。此分支预测模块基于BTB, BHT, GHR和RAS给出下一个PC的地址。其中RAS模块是 用来存储call指令的PC+4作为call结束的返回地址，RAS有两个栈指针，分别为ras\_index\_real\_q代表真实的栈指针，ras\_index\_q代表推测的栈指针。当BTB预测该指令是call时，ras\_index\_q加1；当BTB预测该指令是return时，ras\_index\_q减1。当issue返回预测错误并且该指令是call时，ras\_index\_real\_q加1；当issue返回预测错误并且该指令是return时，ras\_index\_real\_q减1。发生预测错误时，推测的栈指针ras\_index\_q会被置为真实的栈指针ras\_index\_real\_q。当推测该指令为call或者Issue返回预测错误并且该指令是call时，把下一条指令（PC+4）的值放到栈里面；当推测该指令为return或者Issue返回预测错误并且该指令是return时，弹出ras\_index\_q指针对应的值。

GHR分为两个历史寄存器，global\_history\_real\_q用来存放issue返回的真实分支结果，global\_history\_q用来存放预测的分支结果。当分支预测失败时，global\_history\_q会被置为global\_history\_real\_q的值并更新此次分支结果。

BHT是一个深度为512的饱和计数器，采用两种不同算法寻址：

使用gshare算法寻址：用GHR和PC值的组合作为地址来查找或修改BHT。

使用PC的一部分来寻址：使用PC的[11:2]位来寻址。

BTB是一个深度为32的存储器，存储了当前PC值，目标PC值，call，return，jump的标志位。当分支预测错误时，如果BTB中找到相应的entry，那么就更新该entry；如果没有找到相应的entry，就用lfsr算法随机选择BTB中的一个entry进行替换。

### Next PC判定逻辑

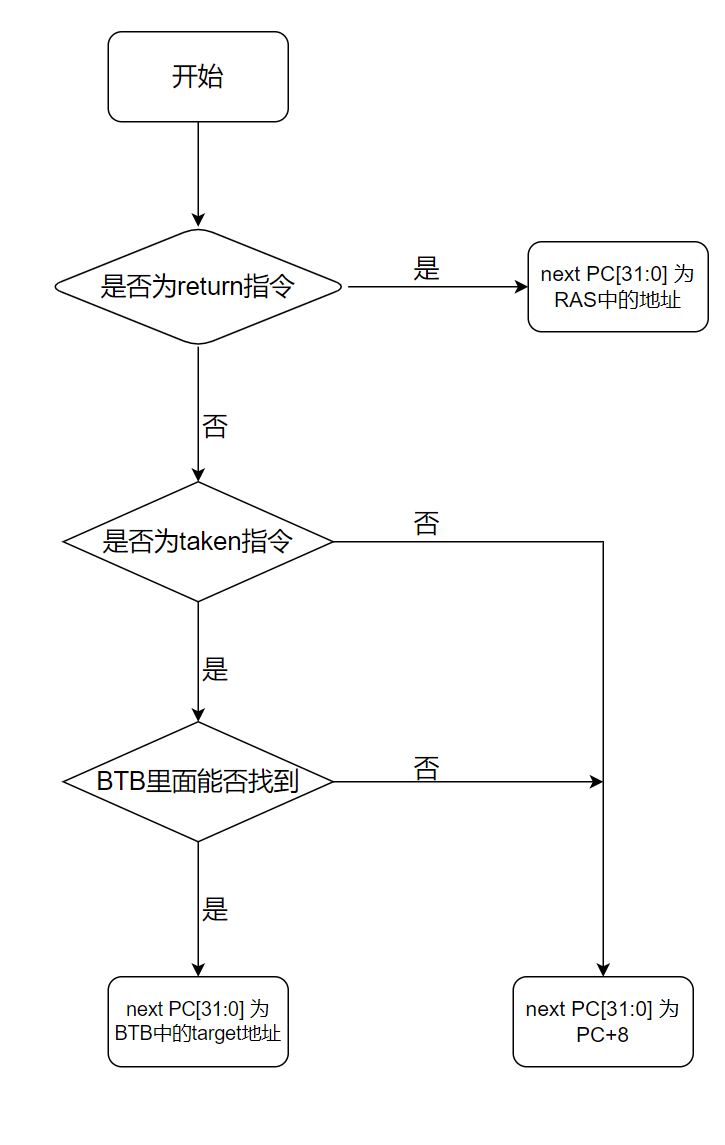


图 2.2 逻辑判定示意图

# 取指单元

## 简介

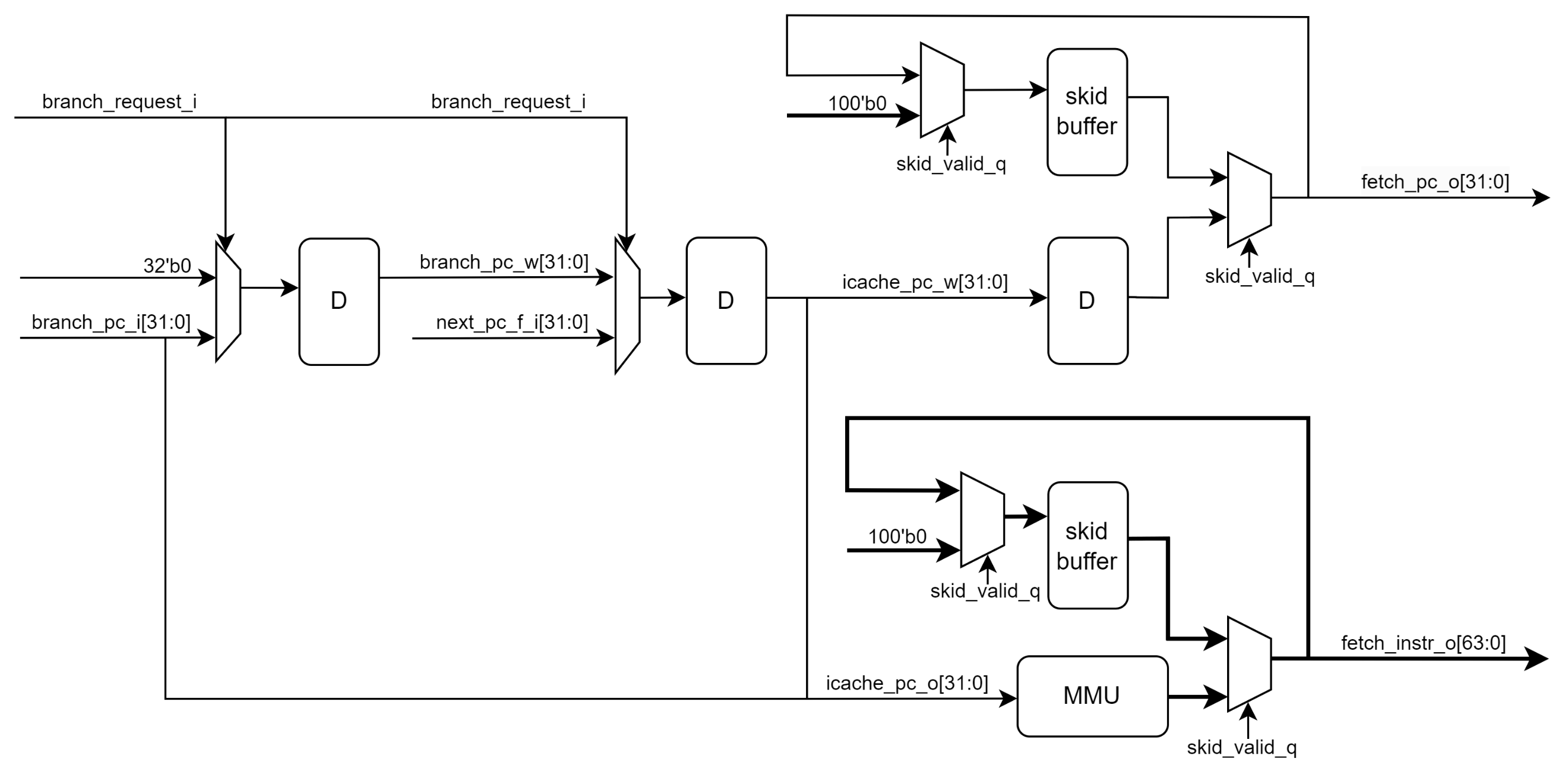


图 3.1 取指单元数据通路

取指单元主要负责向处理器输送需要执行的指令fetch\_instr\_o，取指单元首先收到来自分支预测单元的 PC 值 next\_pc\_f\_i，输出地址MMU。如果 branch\_request\_i 信号被拉高，此时说明分支预测错误，取值单元将接受来自执行单元的 PC 值 branch\_pc\_i ，输出到 MMU。收到来自 MMU 的指令后，输出到译码单元。

# 译码单元介绍

## 简介

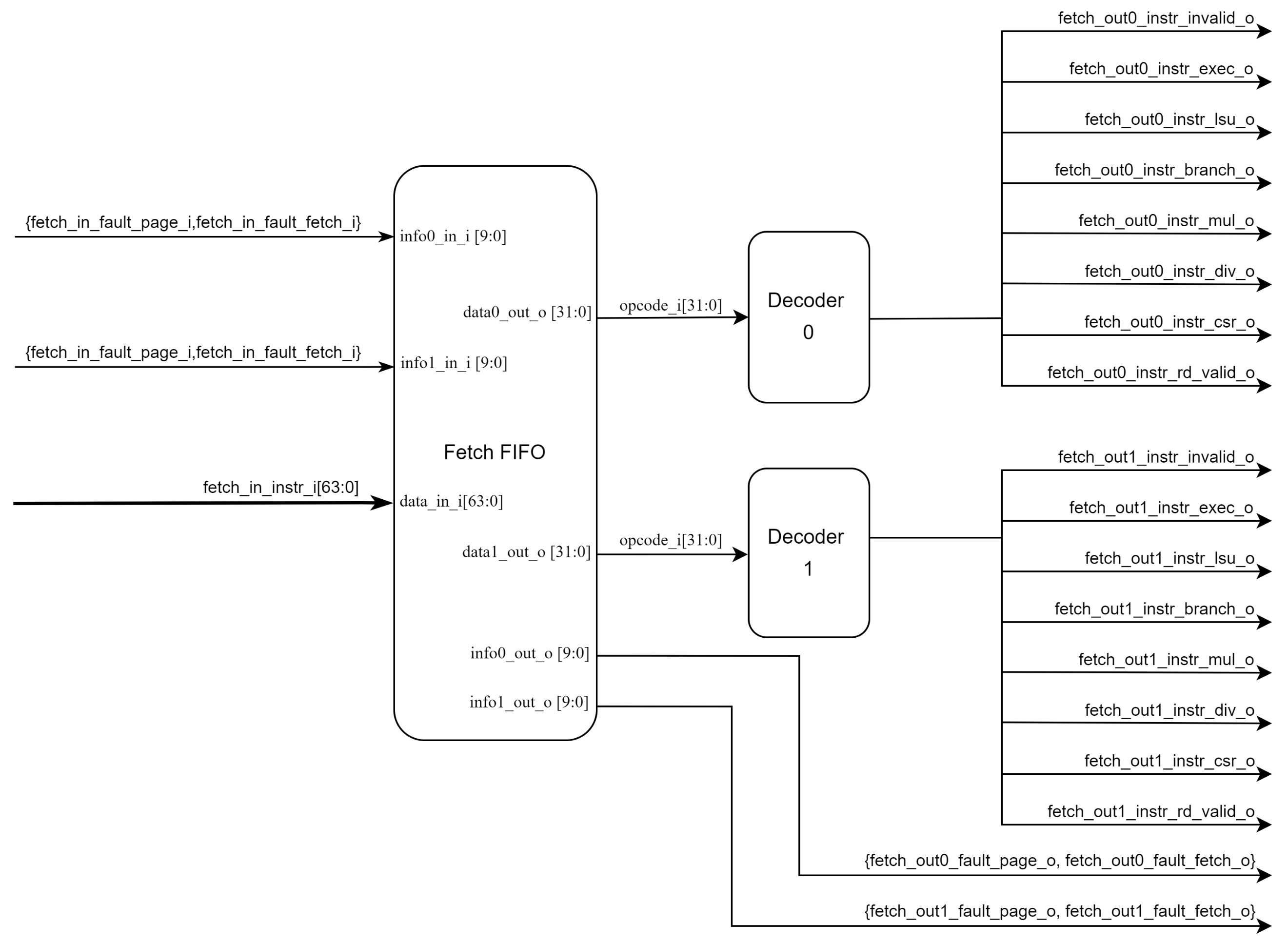


图 4.1 译码单元数据通路

译码单元主要负责将来自取指单元的64 比特指令拆拆封成两路 32 比特指令fetch\_out0\_instr\_o，fetch\_out1\_instr\_o，根据指令的 opcode 译码出该指令所对应的执行单元操作。将执行单元操作编译为1bit 信号，输出到发射单元。

# 执行单元介绍

## 简介

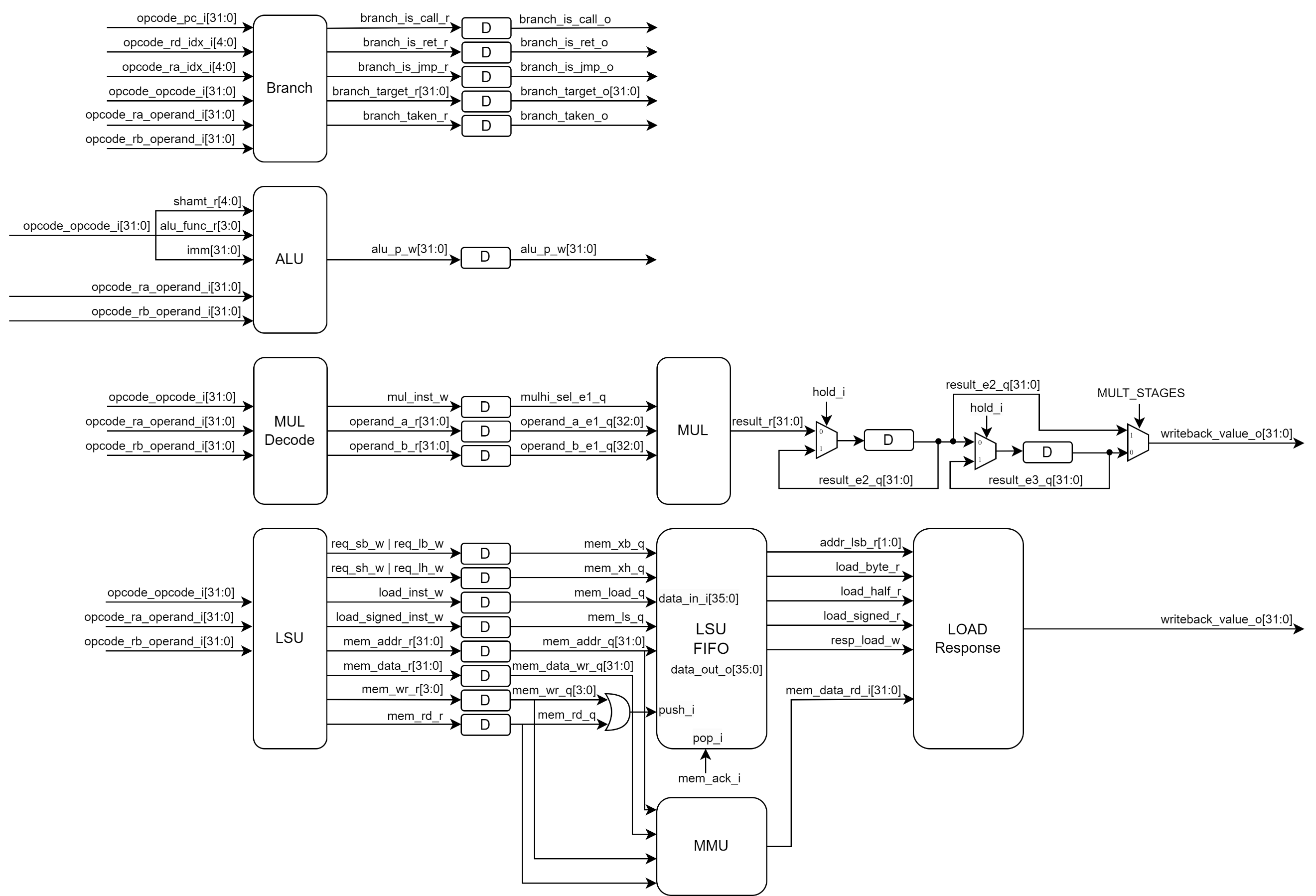


图 5.1 执行单元数据通路

在Biriscv当中，执行单元主要包括ALU，Branch，MUL，LSU模块。在ALU模块中主要负责进行整数运算操作，ALU模块的源操作数来源于执行单元根据得到的指令判断对应的指令操作，获取操作数或者从指令中取到的立即数。

Branch模块主要负责判断指令是否跳转，是什么类型的跳转指令以及跳转地址。MUL模块中主要负责乘法运算操作，根据issue得到的指令判断对应的指令操作，获取对应的操作数进行乘法运算。

在LSU当中，主要由DECODE，FIFO，和RESPONSE模块组成，在DECODE模块中，LSU decode模块通过指令解析出对应的Load和Store指令，包括这些指令对应的读写信息。然后根据指令信息和源操作数解析出对应的load/store的地址以及store指令对应的数据信息。LSU FIFO深度为2，将store指令和load指令对应的地址和指令状态信息存储在起来，接收到mmu的ack信号弹出数据。从fifo中弹出数据后，如果指令是load指令，通过load指令解析出来的信息，将mmu传过来的数据结果写给issue。在LOAD RESPONSE模块中，从fifo中弹出数据后，如果指令是load指令，通过load指令解析出来的信息，将mmu传过来的数据结果写给issue。

# 发射单元介绍

## 简介

Issue单元主要由PC单元，发射选择(Issue select)单元，指令译码(Instruction decoder)单元，发射逻辑(Issue logic)单元，旁路网络(Bypass Network)单元组成，流水线控制(pipeline control)单元，分支预测更新(branch predictor info)单元。下文将逐一介绍这些模块。

### PC单元

在Brisicv的Issue单元中，如下图所示，PC模块的用途是存储正确的PC值。此PC值被存放在pc\_x\_q寄存器当中。RST时pc\_x\_q置零。当遇CSR跳转请求时，pc\_x\_q更新为CSR跳转地址。当遇到Branch跳转请求时，pc\_x\_q更新为Branch跳转地址，这个Branch跳转地址是由ALU单元计算出来的。如果以上情况都不符合，那么判断此时CPU是双发射(Dual Issue)模式或者单发(Single Issue)模式，分别对应PC+8和PC+4。经过更新后，输出最新的pc\_x\_q [31:0], 此信号为PC的期待值。

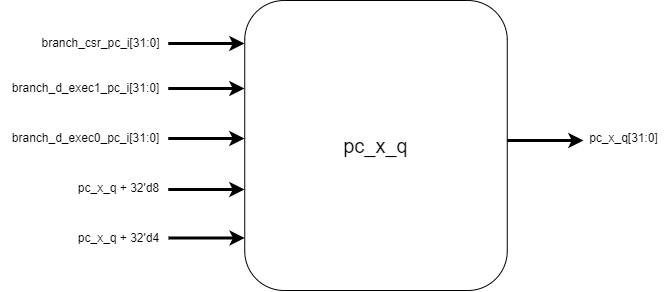


图 6.1 pc\_x\_寄存器

### 发射选择单元

在Briscv当中的发射选择单元主要是负责判断分支预测错误的发生。上文我们提到过pc\_x\_q寄存器，这个寄存器存储着正确的PC值，在发射选择单元中，会首先将fetch0指令的PC值和pc\_x\_q寄存器的值对比，若一致，则将对应的slot0\_valid信号置1。如果发现fetch0指令的PC与pc\_x\_q的pc值不一致，则比较fetch1指令。如果fetch1指令有效的PC值正确，将slot1\_valid置1。如果两条指令的pc地址与pc\_x\_q均不一致，同时指令有效，那么将触发Mispredict信号，通知前端冲刷流水线。除了Mispredict，CSR跳转也会触发冲刷流水线的操作。Mispredict和CSR跳转由 branch\_request\_o信号向前端传递。跳转的PC地址由branch\_pc\_o向前端传递。

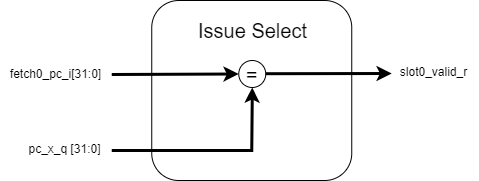


图 6.2 Issue Select单元

### 指令译码单元

指令译码单元主要负责对指令进行译码，分发。其中译码部分主要是从指令中提取源寄存器1，源寄存器2，和目标寄存器。分发部分则是根据slot0，和slot1的valid值来决定是否将指令分配给slot0/1。在Biriscv处理器当中，CSR指令只能经过slot0进行发射，所以slot1的输出会有一条通路是连接到slot0上边的。下文将通过代码示例对此单元功能进行详细描述。

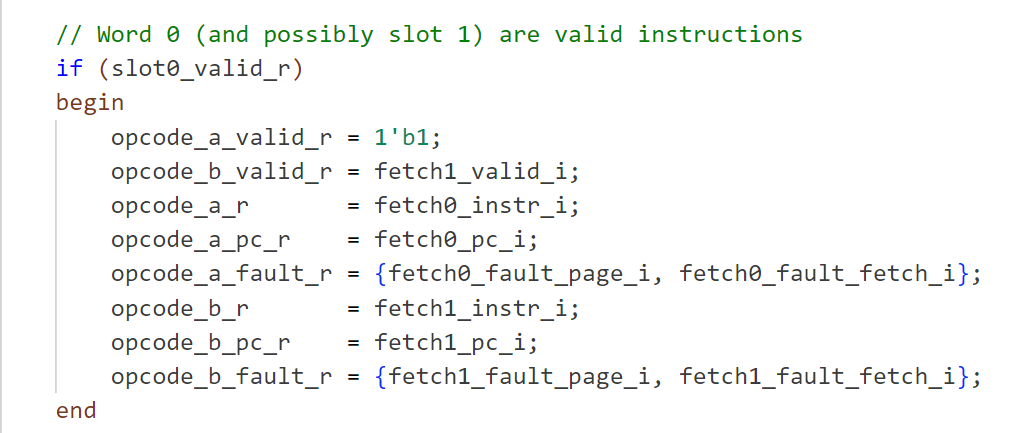


图 6.3 Slot0 部分代码

在上图中，如果slot0\_valid\_r信号被拉高，此时说明pc\_x\_q和fetch0指令的pc地址一致，那么第一条指令一定是正确的，因为在比较pc值的时候我们也对fetch0的valid信号进行了检查。但是我们不能保证第二条指令也是正确的，所以此时opcode\_b\_valid\_r不能被直接写入1’b1，而是要写入fetch1\_valid\_i。这部分代码同时也对每条指令对应的各个（pc值，指令，错误代码）寄存器进行赋值。

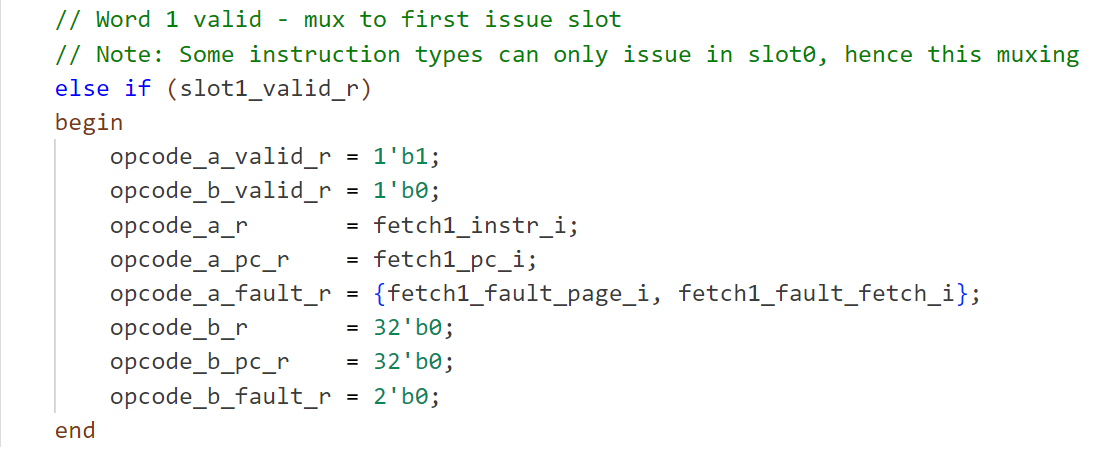


图 6.4 Slot1部分代码

在上图中，如果slot1\_valid\_r信号被拉高，此说明fetch0指令的pc值是错误的，fetch1指令的pc值是正确的。那么此时我们舍弃掉fetch0指令，只执行fetch1指令。将fetch1指令赋值给opcode\_a相关的寄存器。将opcode\_b相关的寄存器置0。

### 发射逻辑单元

如下图所示当一条指令经过了指令译码单元之后，会得到这条指令的ra, rb, rd也就是两个源寄存器和一个目标寄存器的index。此时这条指令会进入发射逻辑单元当中，发射逻辑单元的主要任务分为三项：

* 对指令的操作数进行相关性检查；
* 判断是否可以进行双发射；
* 如果流水线当中正在处理执行周期较长的指令，那么后续同样执行周期较长的指令将被延迟发射。

首先，在进行指令源操作数的相关性检查的时候，发射逻辑单元主要利用Scoreboard来判定当前指令的操作数是否被占用。Scoreboard的数据结构是一个32位数组。当一条指令进入发射逻辑单元后，发射逻辑单元将利用目标寄存器的index作为索引，将Scoreboard中对应的空位置1。后续指令可以通过查找Scoreboard来判定当前指令的寄存器是否被占用。当指令通过相关性检查后，对应的valid信号将被置1。

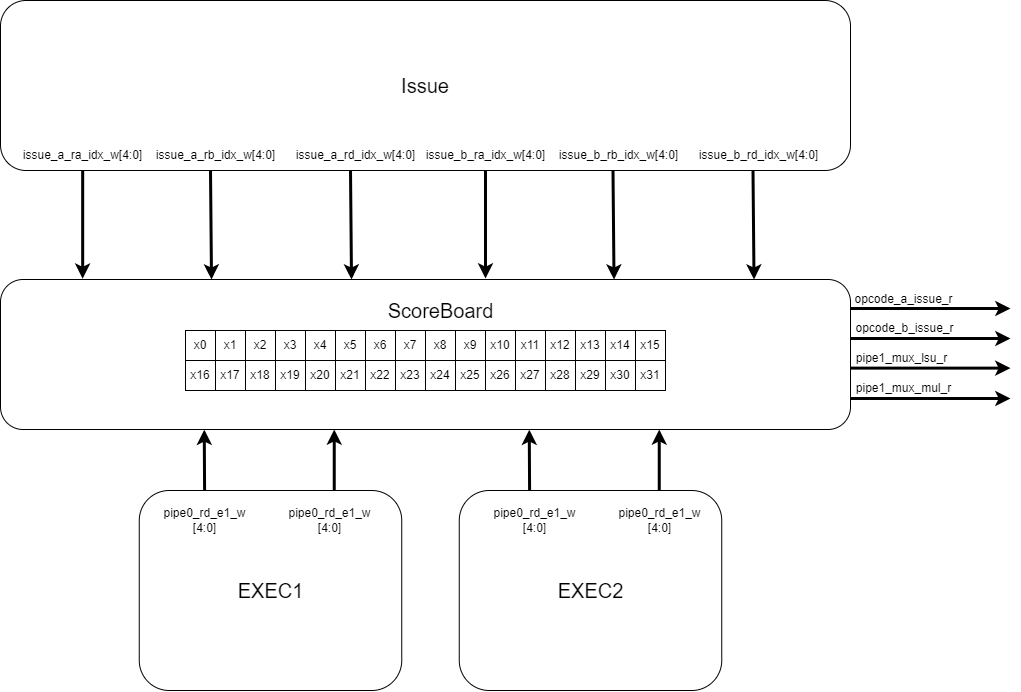


图 6.5 Scoreboard示例图

发射逻辑单元要做的第二项工作是判断是否可以双发射，这里将贴出部分代码作为示例:

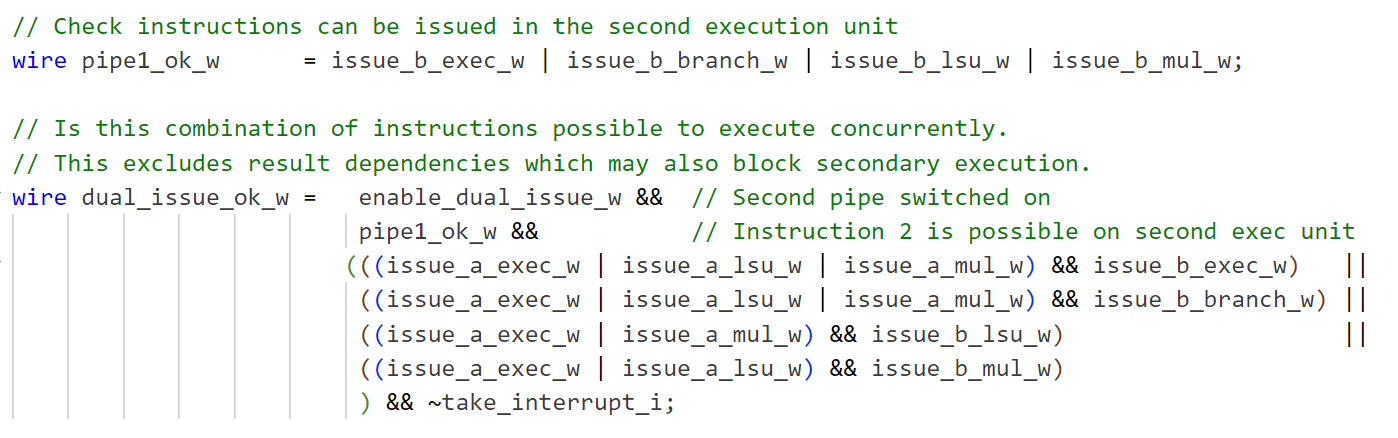


图 6.6 双发射仲裁逻辑代码

通过上图我们可以看到，双发射仲裁的第一步是判断指令是否可以在pipe1（第二条流水线）当中执行。可以在pipe1中执行的指令有：ALU相关指令，跳转指令，Load/Store指令，乘法指令。通过第一步判断后，我们还需要判断pipe1和pipe0是否可以同时开始执行。在Birsicv当中，允许双发射的指令组合有如下几种：

表 6.1 Biriscv双发射指令组合

|  |  |
| --- | --- |
| Issue\_a | Issue\_b |
| ALU指令 | ALU指令 |
| Load/Store指令 | ALU指令 |
| 乘法指令 | ALU指令 |
| ALU指令 | 跳转指令 |
| Load/Store指令 | 跳转指令 |
| 乘法指令 | 跳转指令 |
| ALU指令 | Load/Store指令 |
| 乘法指令 | Load/Store指令 |
| ALU指令 | 乘法指令 |
| Load/Store指令 | 乘法指令 |

下一步发射逻辑单元所要做的事情就是根据发射指令类型，对后续的指令进行延迟发射。具体实现如下：

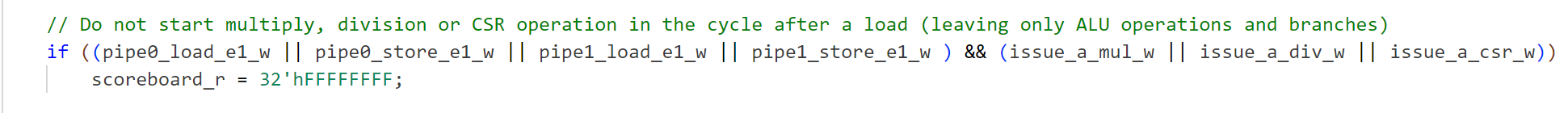


图 6.7 Biriscv双发射指令延迟逻辑

如上图所示，如果发射逻辑单元检测到上一个周期发射的指令为Load指令或Store指令，当前周期要发射指令Mul，Div，CSR中的任何一个，当前周期停止发射。

旁路网络单元：

旁路网络单元主要功能是对源操作数的index进行监控，当源操作数就绪的时候进行数据前递。在Biriscv当中，双发射的两条指令是由不同的流水线控制单元进行管理的。所以slot0和slot1的两个旁路网络单元要对两个流水线控制单元(pipe0 & pipe1)进行监控。当源操作数就绪以后，此时指令已经经过了PC检查，指令译码，相关性检查，源操作数前递四个步骤。当全部就绪后，指令就可以发送给流水线控制单元了。

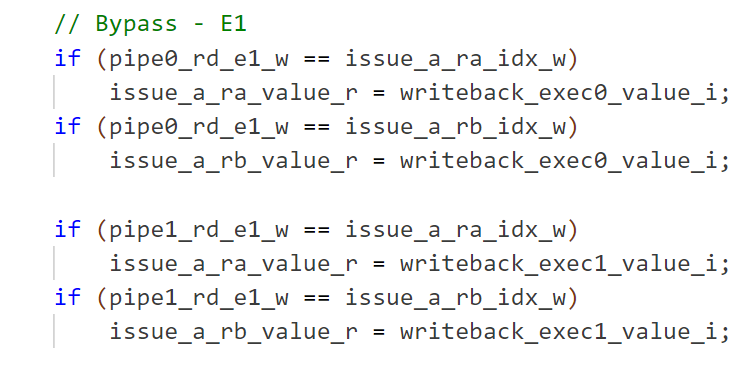


图 6.8 Exec 1数据前递逻辑

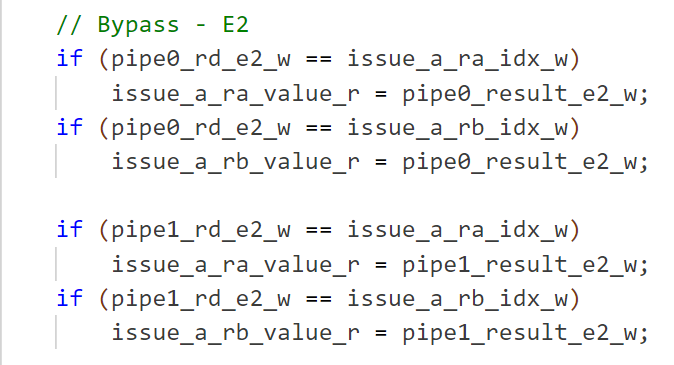


图 6.9 Exec 2数据前递逻辑

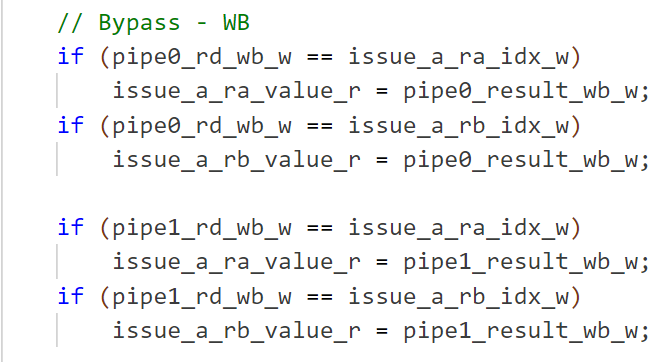


图 6.10 写回阶段数据前递逻辑

上图所示的分别是，Exec1，Exec2，WB阶段的数据前递逻辑。其原理是通过对比各个阶段的目标寄存器Index，和发射阶段的源寄存器Index。如果发现一致，则将数据进行前递。

### 流水线控制单元

在Biriscv处理器当中，流水线总共分为6级或7级(可自定义decode stage)。其中后三级流水线：exec1，exec2，wb都是由流水线控制单元进行管理的。流水线控制单元包含三个模块E1、E2、E3，分别负责地址分配、存储结果、将结果寄存器中的值写回到寄存器堆。下面将逐一介绍这些模块。

表 6.2 E1模块

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 位宽 | 信号类型 | 说明 |
| issue\_stall\_i | 1 | input | 流水线停顿信号 |
| issue\_lsu\_i | 1 | input | lsu信号 |
| issue\_csr\_i | 1 | input | csr信号 |
| issue\_div\_i | 1 | input | div信号 |
| issue\_mul\_i | 1 | input | mul信号 |
| issue\_accept\_i | 1 | input | accept信号 |
| issue\_valid\_i | 1 | input | issue有效信号 |
| squash\_e1\_e2\_i | 1 | input | 流水线冲刷信号 |
| issue\_rd\_valid\_i | 1 | input | 目标寄存器信号 |
| issue\_branch\_i | 1 | input | 跳转信号 |
| take\_interrupt\_i | 1 | input | 中断信号 |
| branch\_misaligned\_w | 1 | input | 分支预测未对准 |
| squash\_e1\_e2\_o | 1 | input | 流水线冲刷信号 |
| issue\_branch\_taken\_i | 1 | input | 分支预测被采用 |
| issue\_exception\_i | 6 | input | 异常信号 |
| issue\_pc\_i | 32 | input | pc地址信号 |
| issue\_branch\_target\_i | 32 | input | 分支预测next pc地址信号 |
| issue\_opcode\_i | 32 | input | 操作码信号 |
| issue\_operand\_ra\_i | 32 | input | 操作数 |
| issue\_operand\_rb\_i | 32 | input | 操作数 |
| valid\_e1\_q | 1 | output | E1有效信号 |
| opcode\_e1\_q | 32 | output | E1中操作码信号 |
| pc\_e1\_q | 32 | output | E1中pc地址 |
| npc\_e1\_q | 32 | output | E1中next pc地址 |
| operand\_ra\_e1\_q | 32 | output | E1中操作数 |
| operand\_rb\_e1\_q | 32 | output | E1中操作数 |
| ctrl\_e1\_q | `PCINFO\_W | output | E1中控制信号 |
| exception\_e1\_q | `EXCEPTION\_W | output | E1中异常信号 |

#### 流水线正常工作时

pc\_e1\_q[31:0]、opcode\_e1\_q[31：0]、operand\_ra\_e1\_q[31:0]、operand\_rb\_e1\_q[31:0]的值分别由输入信号issue\_opcode\_i[31:0]、issue\_operand\_ra\_i[31:0]、issue\_pc\_i[31:0]、issue\_operand\_rb\_i[31:0]赋予。npc\_e1\_q[31:0]为下一周期的pc地址，在输入信号issue\_branch\_taken\_i为1表示分支预测正确时，取issue\_branch\_target\_i[31:0]的值，否则取issue\_pc\_i[31:0]+31’b4。

exception\_e1\_q[5:0]存储异常信息，只要输入信号issue\_exception\_i[5:0]中有1信号，就将其值赋予exception\_e1\_q[5:0]。否则检查branch\_misaligned\_w信号，为1 时对exception\_e1\_q[5:0]赋`EXCEPTION\_MISALIGNED\_FETCH，为0时赋6’b0。

ctrl\_e1\_q[9：0]为控制信号，每一位信号分别对应一个功能单元，从小到达依次为ALU、LOAD、STORE、CSR、DIV、MUL、BRANCH、RD\_VALID、INTR、COMPLETE。在不发生中断（take\_interrupt\_i信号不为1）时，前八位信号继承来自issue的控制信号，INTR位取0，COMPLETE位取1。

#### 流水线中发生异常/中断时

发生中断时，ctrl\_e1\_q[7：1]为0，INTR位和COMPLETE位为1。

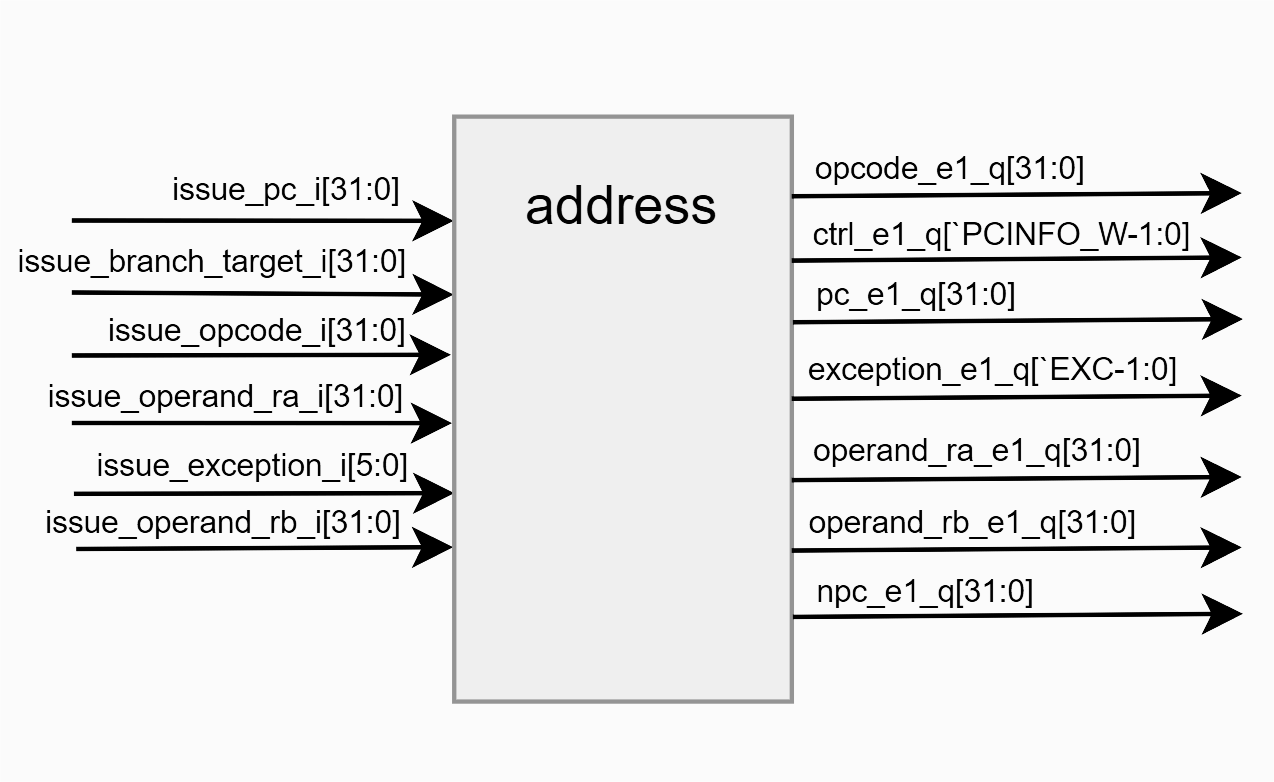


图 6.11 端口示意图

表 6.3 E2模块

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 位宽 | 信号类型 | 说明 |
| squash\_e1\_e2\_i | 1 | input | 流水线冲刷信号 |
| issue\_stall\_i | 1 | input | 流水线停顿信号 |
| squash\_e1\_e2\_o | 1 | input | 流水线冲刷信号 |
| csr\_result\_write\_e1\_i | 1 | input | 写回有效信号 |
| valid\_e1\_q | 1 | input | E1有效信号 |
| csr\_result\_exception\_e1\_i | 6 | input | csr异常信号 |
| pc\_e1\_q | 32 | input | 来自E1的pc地址 |
| operand\_ra\_e1\_q | 32 | input | 来自E1的操作数 |
| operand\_rb\_e1\_q | 32 | input | 来自E1的操作数 |
| npc\_e1\_q | 32 | input | 来自E1的next pc地址 |
| csr\_result\_wdata\_e1\_i | 32 | input | 要写入csr寄存器的值 |
| div\_result\_i | 32 | input | 写回的div值 |
| csr\_result\_value\_e1\_i | 32 | input | csr读取结果 |
| alu\_result\_e1\_i | 32 | input | alu结果 |
| ctrl\_e1\_q | `PCINFO\_W | input | 来自E1的ctrl信号 |
| exception\_e1\_q | `EXCEPTION\_W | input | 来自E1的异常信号 |
| valid\_e2\_q | 1 | output | E2有效信号 |
| csr\_wr\_e2\_q | 1 | output | csr写回信号 |
| result\_e2\_q | 32 | output | 写回数据 |
| csr\_wdata\_e2\_q | 32 | output | 要写入csr寄存器的值 |
| pc\_e2\_q | 32 | output | E2中pc地址 |
| npc\_e2\_q | 32 | output | E2中next pc地址 |
| opcode\_e2\_q | 32 | output | E2中操作码 |
| operand\_ra\_e2\_q | 32 | output | E2中操作数 |
| operand\_rb\_e2\_q32 | 32 | output | E2中操作数 |
| exception\_e2\_q | `EXCEPTION\_W | output | E2中异常 |
| ctrl\_e2\_q | `PCINFO\_W | output | E2中控制信号 |

#### 流水线正常工作时

csr\_result\_exception\_e1\_i[5:0]的值载入exception\_e2\_q[5:0]。如果ctrl\_e1\_q[9:0]中DIV位为1，result\_e2\_q[31:0]赋div\_result\_i[31:0]，否则检查CSR位信号，为1时将csr\_result\_value\_e1\_i[31:0]赋予result\_e2\_q[31:0]，为0时将alu\_result\_e1\_i[31:0]赋予result\_e2\_q[31:0]。

#### 流水线中发生异常/中断时

当系统中存在异常时，冲洗掉模块中的信号。如果ctrl\_e1\_q[9:0]中INTR位为1,exception\_e2\_q[5:0]赋6’h20，否则检查前端exception\_e1\_q[5:0]是否存在异常，如果有则将exception\_e1\_q[5:0]赋予exception\_e2\_q[5:0]、valid\_e2\_q赋0。

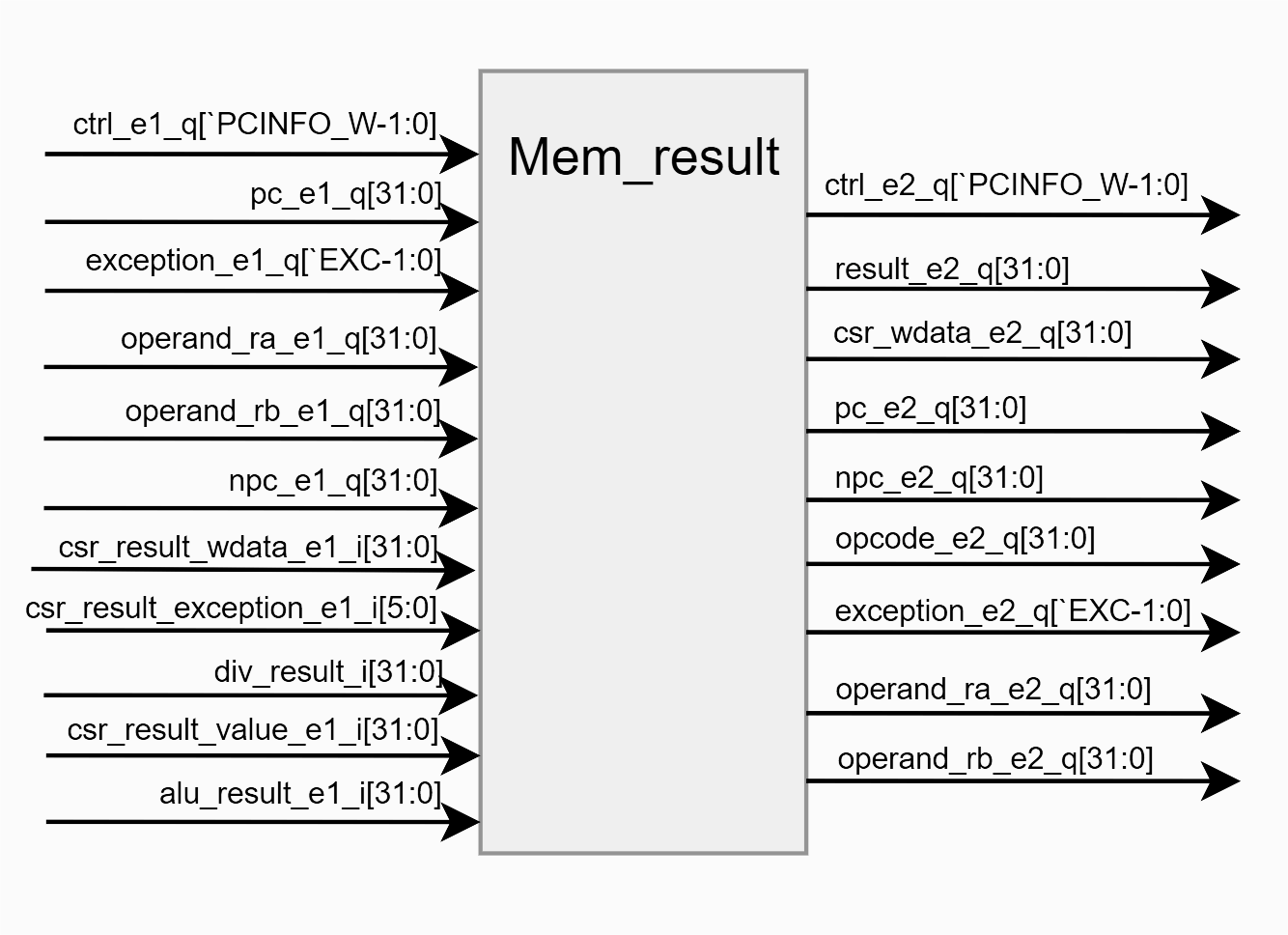


图 6.12 端口示意图

流水线控制单元输出的result\_e2\_o[31:0]在mul\_result\_e2\_i[31:0]、result\_e2\_q[31:0]、mem\_result\_e2\_i[31:0]三个信号当中做出仲裁选择。SUPPORT\_LOAD\_BYPASS、valid\_e2\_w为1，ctrl\_e2\_q信号中的LOAD位或STORE位为1时赋mul\_result\_e2\_i[31:0]的值。其次在SUPPORT\_MUL\_BYPASS、valid\_e2\_w、ctrl\_e2\_q信号中的MUL位为1时赋mul\_result\_e2\_i[31:0]的值，否则赋result\_e2\_q[31:0]的值。

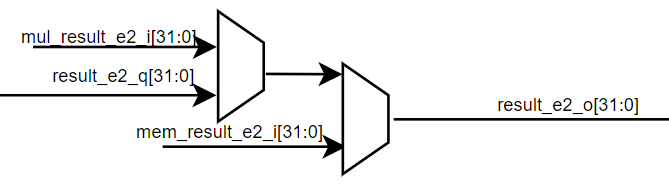


图 6.13 数据通路

表 6.4 E3模块

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 位宽 | 信号类型 | 说明 |
| squash\_wb\_i | 1 | input | 流水线冲刷信号 |
| valid\_e2\_w | 1 | input | E2有效信号 |
| issue\_stall\_i | 1 | input | 流水线停顿信号，为1时，流水线停止运行 |
| valid\_e2\_q | 1 | input | E2有效信号 |
| csr\_wr\_e2\_q | 1 | input | csr写回信号 |
| mem\_result\_e2\_i | 32 | input | 写回缓存结果 |
| mul\_result\_e2\_i | 32 | input | 写回的mul结果 |
| result\_e2\_q | 32 | input | 来自E2的写回结果 |
| csr\_wdata\_e2\_q | 32 | input | 要写入csr寄存器的值 |
| pc\_e2\_q | 32 | input | 来自E2的pc地址 |
| npc\_e2\_q | 32 | input | 来自E2的next pc地址 |
| opcode\_e2\_q | 32 | input | 来自E2操作码 |
| operand\_ra\_e2\_q | 32 | input | 来自E2操作数 |
| operand\_rb\_e2\_q | 32 | input | 来自E2操作数 |
| exception\_e2\_q | `EXCEPTION\_W | input | 来自E2的异常信号 |
| ctrl\_e2\_q | `PCINFO\_W | input | 来自E2的控制信号 |
| csr\_write\_wb\_q | 1 | output | 写回信号 |
| valid\_wb\_q | 1 | output | 写回有效信号 |
| opcode\_wb\_q | 32 | output | 写回操作码 |
| pc\_wb\_q | 32 | output | 写回pc地址 |
| result\_wb\_q | 32 | output | 写回结果 |
| operand\_ra\_wb\_q | 32 | output | 写回操作数 |
| operand\_rb\_wb\_q | 32 | output | 写回操作数 |
| csr\_wdata\_wb\_q | 32 | output | 写回csr寄存器数据 |
| npc\_wb\_q | 32 | output | 写回next pc地址 |
| exception\_wb\_q | `EXCEPTION\_W | output | 写回异常信号 |
| ctrl\_wb\_q | `PCINFO\_W | output | 写回控制信号 |

#### 流水线正常工作时

pc\_wb\_o[31:0]、opcode\_wb\_o[31:0]、operand\_ra\_wb\_o[31:0]、operand\_rb\_wb\_o[31:0]、exception\_wb\_o[31:0]、csr\_wdata\_wb\_o[31:0]、ctrl\_wb\_q[31:0]等信号分别由pc\_e2\_q[31:0]、opcode\_wb\_q[31:0]、operand\_ra\_e2\_q[31:0]、operand\_rb\_e2\_q[31:0]、exception\_e2\_r[31:0]、csr\_wdata\_e2\_q[31:0]\ctrl\_e2\_q[31:0]的值赋予。

#### 流水线中发生异常/中断时

当系统中存在异常时，冲洗掉模块中的信号,ctrl\_wb\_o[9:0]接受ctrl\_e2\_q[9:0]中除RD\_VALID位以外的信号。valid\_wb\_q信号在E2模块中发生load为对准、load失败、store未对准、store失败、page fault load和page fault store等异常时置零。

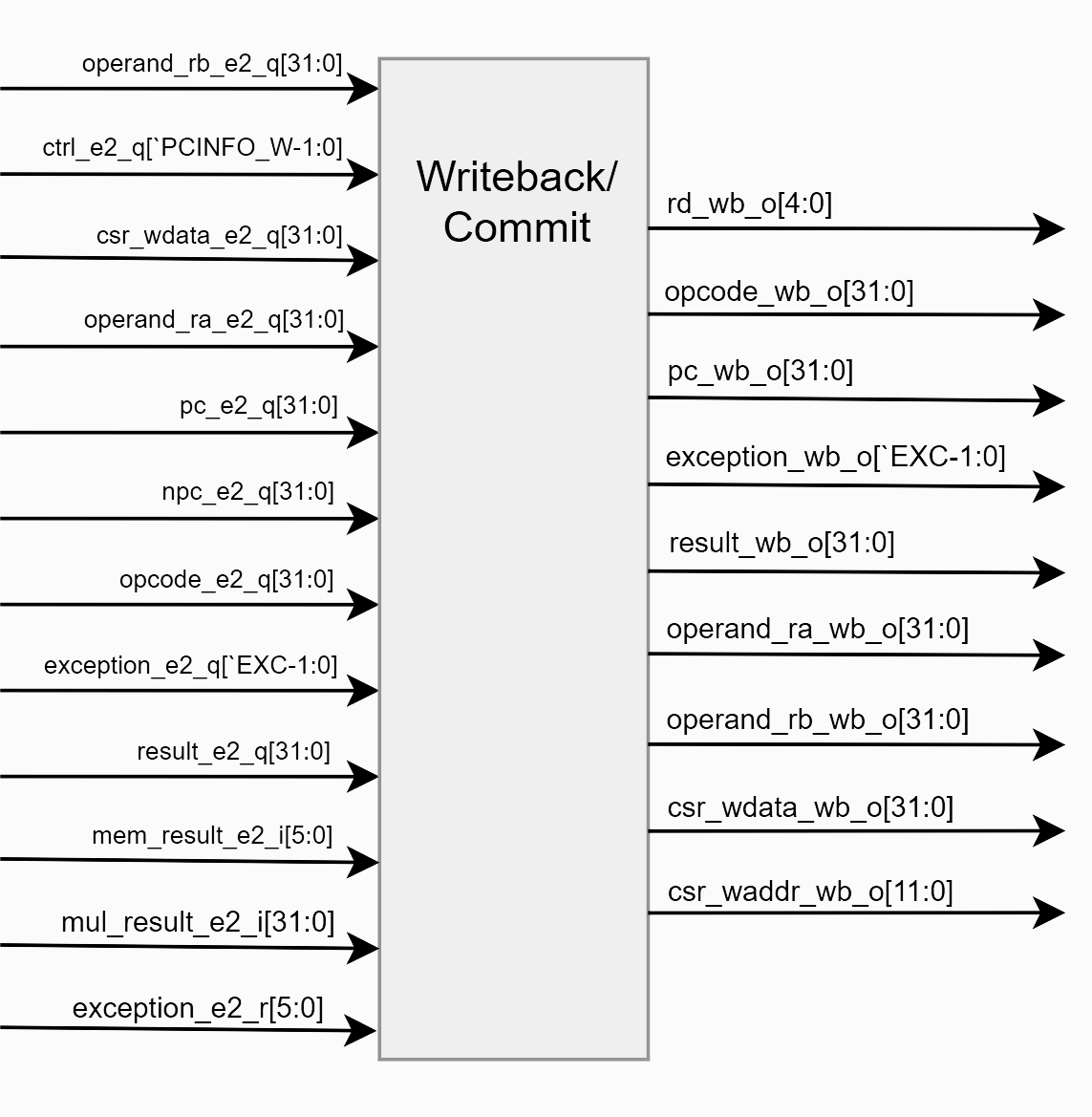


图 6.14 Writeback/Commit示意图

输出信号rd\_wb\_o[4:0]的值由vslid\_wb\_o、ctrl\_wb\_q[9:0]信号的RD\_VALID位、stall\_o的非信号的逻辑与运算的5位拓展值与opcode\_wb\_q[31:0]信号的[11:7]位信号的与运算结果得出。

### 分支预测更新单元

分支预测更新单元的主要工作是把最新的跳转信息更新给前端的BPU单元。BPU接收到来自Issue的最新跳转信息就可以对饱和计数器和RAS做相应的更改。下图是分支预测更新单元的代码示例。如图所示，分支预测更新单元会根据pipe0和pipe1的执行情况，选择对应的数据，向BPU发送。

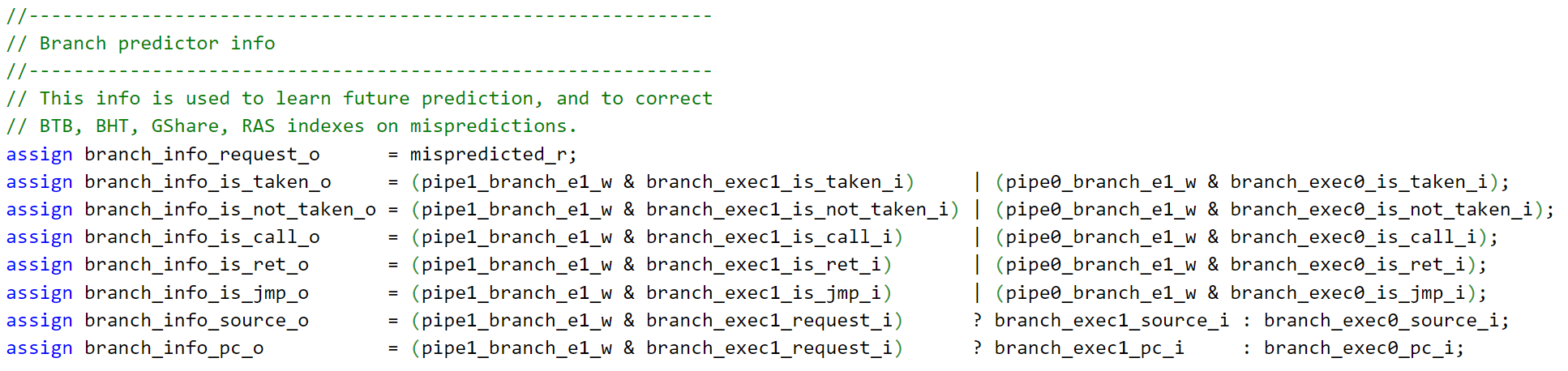


图 6.15 分支预测更新单元代码示例

# MMU单元介绍

MMU单元待修改

# 指令缓存单元介绍

## 简介

I-Cache采取两路组相连结构，分为Tag部分和Data部分。

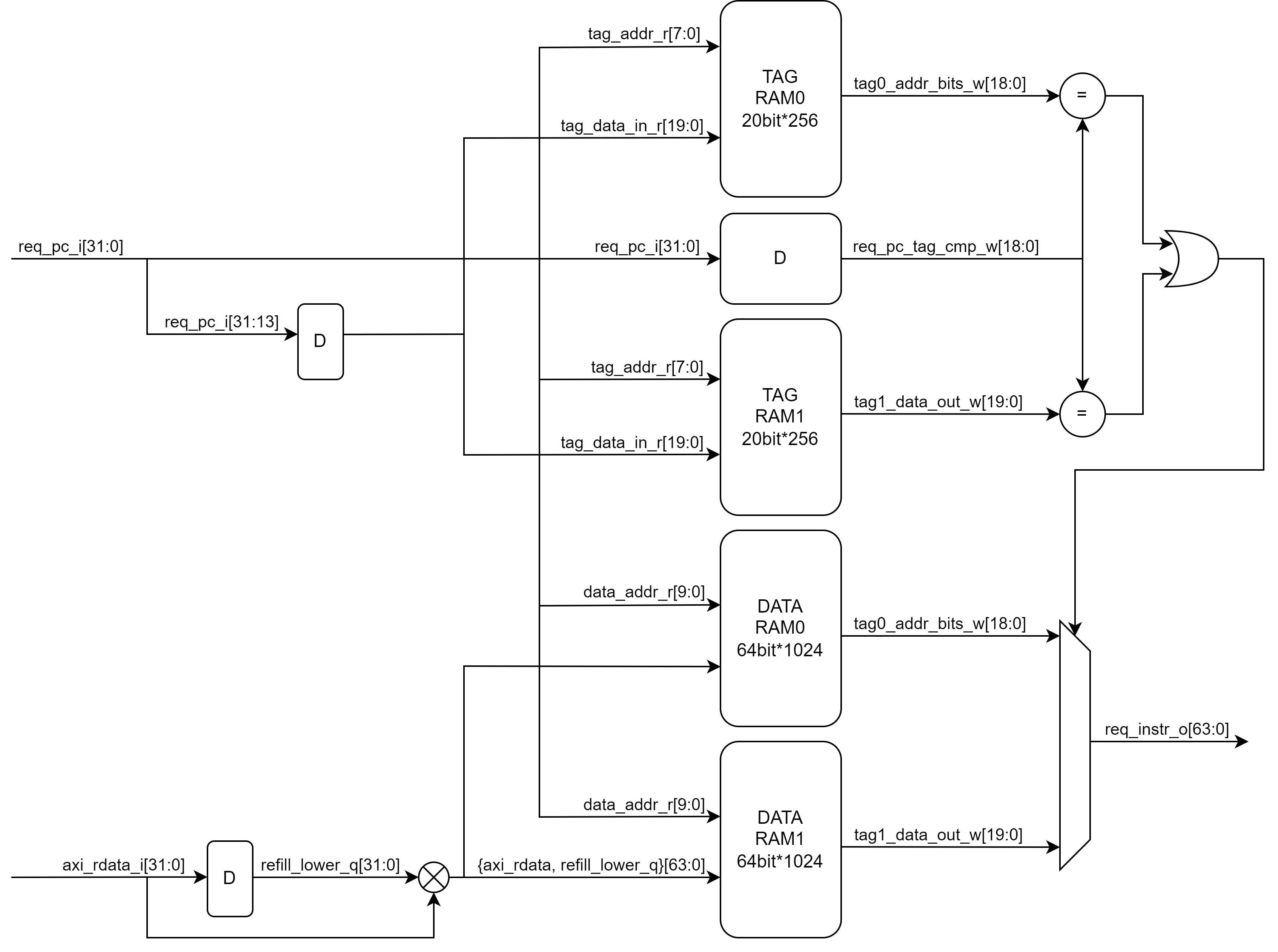


图 8.1 I-Cache数据通路

### TAG RAM

Tag RAM的深度为256，数据宽度为20bit，包含1位valid和19位的Tag，这里的Tag是指req\_pc\_i的[31:13]位。req\_pc\_i的[12:5]位作为index，用来在Tag RAM中寻址。用index寻址后分别在两路Tag RAM中比较Tag的值，如果有一个相同则代表hit成功。

### Data RAM

Data RAM的深度为1024，数据宽度为64bit，使用PC的[12:3]寻址。

# 数据缓存单元介绍

## 简介

整0个D-cache分为5个模块：Core、Uncache、Mux、Pmem Mux、AXI。

Mux：对输入的数据进行选择与控制。

Core模块：使用cache时候的数据模块，包含两个DATA RAM和TAG RAM。

Uncache模块：不使用数据时候的数据模块。

Pmem Mux：对输出信号进行选择与控制。

AXI：AXI总线控制模块。

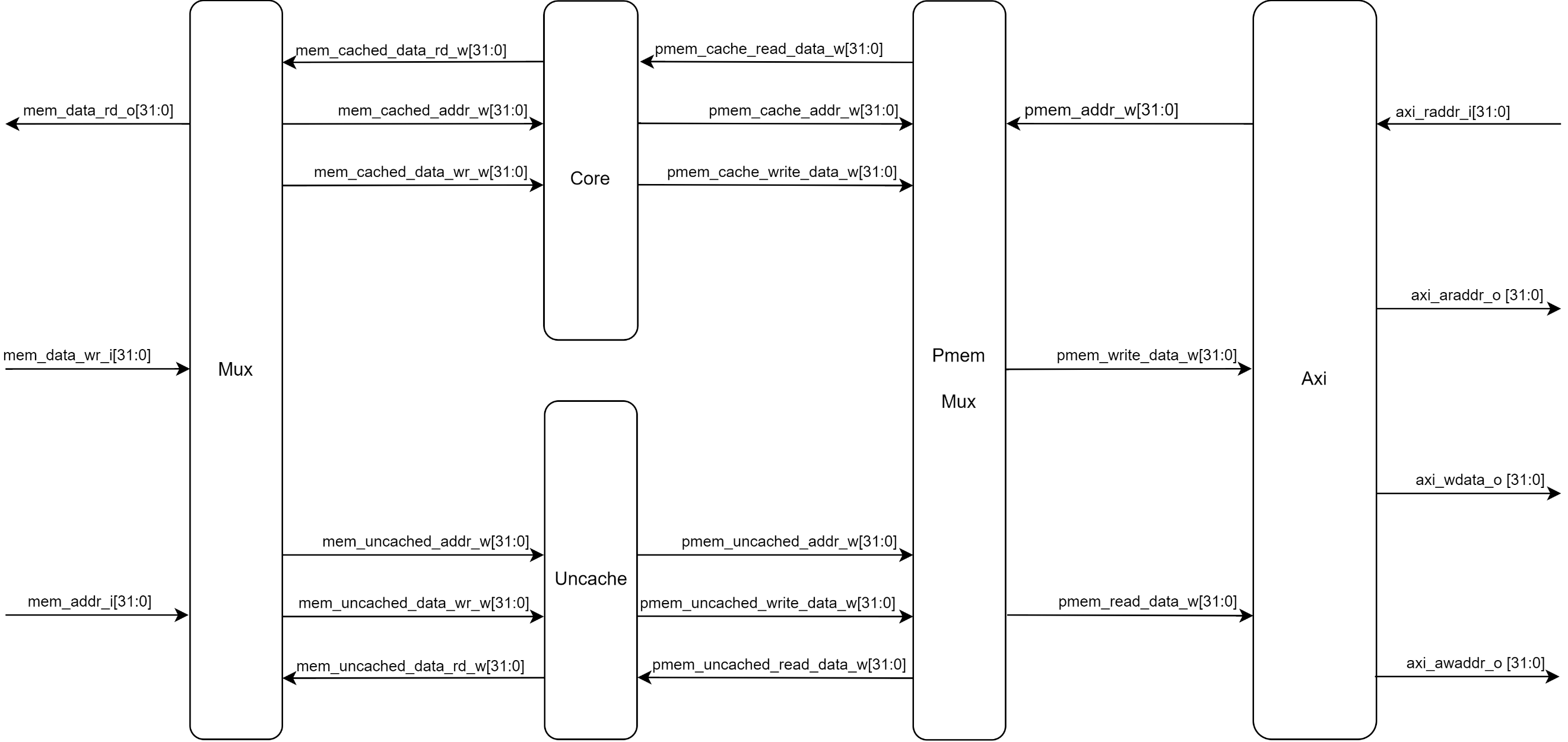


图 9.1 D-Cache数据通路

表 9.1 端口列表

|  |  |
| --- | --- |
| Name | Description |
| axi\_rdata\_i [31:0] | 从外部(L2cache或者存储器)输入的读数据 |
| mem\_addr\_i [31:0] | 由处理器内部MMU模块输入的地址 |
| mem\_data\_wr\_i  [31:0] | 由处理器内部MMU模块输入的写数据 |
|  |  |
| axi\_wdata\_o [31:0] | 写数据输出 |
| axi\_araddr\_o [31:0] | 读地址输出 |
| axi\_awaddr\_o [31:0] | 写地址输出 |
| mem\_data\_rd\_o [31:0] | 命中后直接读取输出数据输出到MMU中 |

# 处理器状态转换

S模式转换M模式：

* Trap发生
* 查询medeleg/mideleg寄存器，判断是否需要委托当前异常/中断
* 设置mepc为当前pc值，mepc为mret跳转地址
* 设置mcause寄存器，写入当前异常/中断错误代码
* 设置mstatus.MIE为0，禁用中断
* 设置mstatus.MPP为01，代表原始特权等级为Supervisor
* 设置mstatus.MPIE为1，代表原始中断使能为1
* 设置mtval寄存器为0或由软件设置为相关信息
* 读取mtvec寄存器，赋值给pc寄存器
* 进入Machine模式

## M模式寄存器

### Machine ISA Register (misa)

misa CSR是一个WARL读写寄存器，报告hart支持的ISA。该寄存器在任何实现中都必须是可读的，但是可以返回0值来表示未实现misa寄存器，如果misa为零，寄存器的宽度则有一个固定值。



图 10.1 misa寄存器

MXL[1:0]字段编码了M模式下寄存器宽度（32，64，128），MXL部分的编码如下表。

表 10.1 MXL部分的编码

|  |  |
| --- | --- |
| MXL | XLEN |
| 1 | 32 |
| 2 | 64 |
| 3 | 128 |

如果对misa的写入导致MXLEN发生变化，则MXL的位置移动到misa在新宽度处的最重要的两位。

如果misa为零，也可以通过将立即数4放入寄存器中，然后每次将寄存器向左移动31位来找到基宽。如果一次移位后为零，则宽度为RV32。如果两次移位后为零，则宽度为RV64，否则为RV128。

Extension[25:0] 字段对标准扩展名的存在进行编码，每个字母使用一个bit，bit0代表“A”扩展，bit1代表“B”扩展，以此类推，bit25代表“Z”扩展。如果“I”扩展和“E”扩展同时被使能，那么“I”扩展将被采用，“E”扩展将被忽略。如果支持User和Supervisor模式，“U”和“S”位将分别被置1。如果有任何非标准扩展，则 “X”位置1。

Extension字段的每一位的详细对应关系如下表。

表 10.2 Extension字段

|  |  |  |
| --- | --- | --- |
| Bit | Character | Description |
| 0 | A | Atomic extension |
| 1 | B | Tentatively reserved for Bit-Manipulation extension |
| 2 | C | Compressed extension |
| 3 | D | Double-precision floating-point extension |
| 4 | E | RV32E base ISA |
| 5 | F | Single-precision floating-point extension |
| 6 | G | Reserved |
| 7 | H | Hypervisor extension |
| 8 | I | RV32I/64I/128I base ISA |
| 9 | J | Tentatively reserved for Dynamically Translated Languages extension |
| 10 | K | Reserved |
| 11 | L | Reserved |
| 12 | M | Integer Multiply/Divide extension |
| 13 | N | Tentatively reserved for User-Level Interrupts extension |
| 14 | O | Reserved |
| 15 | P | Tentatively reserved for Packed-SIMD extension |
| 16 | Q | Quad-precision floating-point extension |
| 17 | R | Reserved |
| 18 | S | Supervisor mode implemented |
| 19 | T | Reserved |
| 20 | U | User mode implemented |
| 21 | V | Tentatively reserved for Vector extension |
| 22 | W | Reserved |
| 23 | X | Non-standard extensions present |
| 24 | Y | Reserved |
| 25 | Z | Reserved |

“E”位为只读。除非misa全为零，否则“E”位总是与“I”位相反。同时支持RV32E和RV32I的实现可以通过 “I”位置0来选择RV32E。

如果一个ISA扩展x依赖于一个ISA扩展y，那么尝试启用扩展x而禁用扩展y会导致两个扩展都被禁用。例如，设置“F”=0，“D”=1，结果将禁用“F”和“D”。

### Machine Vendor ID Register (mvendorid)

mvendorid CSR是一个32位只读寄存器，提供核心提供商的JEDEC制造商ID。这个寄存器在任何实现中都必须是可读的，但是可以返回0值来表示该字段没有实现，或者这是一个非商业实现。

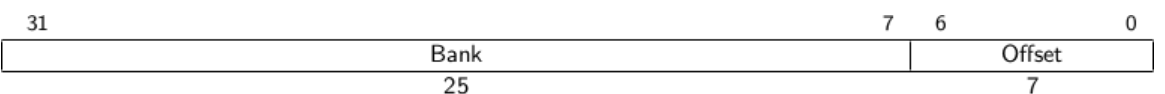


图 10.2 mvendorid寄存器

### Machine Architecture ID Register (marchid)

marchid CSR是一个mxlen位的只读寄存器，编码hart的基本微体系结构。该寄存器在任何实现中都必须是可读的，但是可以返回0值来表示未实现该字段。mvendorid和marchid的组合应该唯一地标识所实现的hart微架构的类型。



图 10.3 marchid寄存器

### Machine Implementation ID Register (mimpid)

mimid CSR提供了处理器实现版本的唯一编码。这个寄存器在任何实现中都必须是可读的，但是可以返回0值来表示该字段没有实现。实现值应该反映RISC-V处理器本身的设计，而不是任何周围的系统。

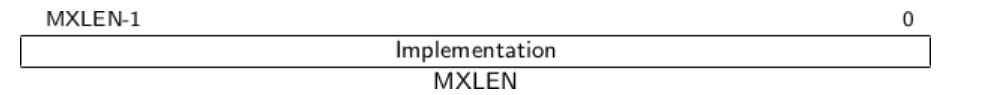


图 10.4 mimpid寄存器

### Hart ID Register (mhartid)

hartid CSR是一个mxlen位的只读寄存器，包含运行代码的硬件线程的整数ID。这个寄存器在任何实现中都必须是可读的。在多处理器系统中，Hart ID不一定连续编号，但至少有一个Hart ID必须为零。Hart id在执行环境中必须是唯一的。

### Machine Status Registers (mstatus)

mstatus寄存器是一个mxlen位可读写寄存器，RV32和RV64的格式分别如图所示。状态寄存器跟踪并控制hart的当前运行状态。

日程表

描述已自动生成

图 10.5 mstatus寄存器

mstatush是一个32位读写寄存器，仅存在于RV32中。mstatush的30:4位通常包含与RV64的mstatus的62:36位相同的字段。mstatush中不存在SD、SXL和UXL字段。

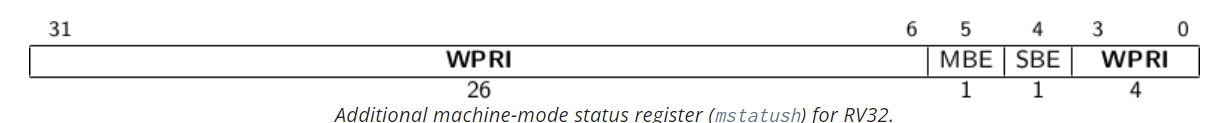


图 10.6 mstatus寄存器

#### 特权和全局中断使能堆栈MIE和SIE位

分别为M模式和S模式提供全局中断使能。这些位主要用于保证当前特权模式下中断处理程序的原子性。

当hart以特权模式x执行时，当xIE=1时全局启用中断，当xIE=0时全局禁用中断。无论为低特权模式设置了任何全局wIE位,低特权模式的中断w<x总是全局禁用的。无论高特权模式的全局yIE位设置如何，高特权模式的中断y>x始终是全局启用的。

在将控制权交给低特权模式之前，高特权级别的代码可以使用单独的每个中断启用位来禁用所选的高特权模式中断。xPIE保留在trap发生之前激活的中断启用位的值，xPP保留之前的特权模式。MPP的值是两位宽（代表M=11，S=01，U=00），SPP是一位宽（S=1，U=0）。xPP字段是WARL字段，只能包含特权模式x和任何实现的特权模式低于x。如果没有实现特权模式x，则xPP必须为只读0。

MRET或SRET指令分别用于从m模式或s模式返回。在执行xRET指令时，假设xPP值是y，则将xIE置为xPIE;将特权模式变为y;xPIE设置为1; xPP设置为支持的最小特权模式(如果支持U-mode，则为U，否则为M)。xPP≠M时，xRET也设置MPRV=0。对于RV64系统，SXL和UXL字段是分别控制S模式和U模式下XLEN值的WARL字段。这些字段的编码与misa的MXL字段相同，如下表所示。S模式和U模式下的有效XLEN分别称为SXLEN和UXLEN。

表 10.3 MXL字段

|  |  |
| --- | --- |
| SXL/UXL | XLEN |
| 1 | 32 |
| 2 | 64 |
| 3 | 128 |

对于RV32系统，SXL和UXL字段不存在，并且SXLEN=32和UXLEN=32。对于RV64系统，如果不支持S模式，则SXL为只读零。否则，它是一个编码SXLEN当前值的WARL字段。具体来说，可以使SXL成为只读字段，其值始终确保SXLEN=MXLEN。

对于RV64系统，如果不支持U模式，则UXL为只读零。否则，它是一个编码UXLEN当前值的WARL字段。具体来说，实现可以使UXL成为只读字段，其值始终确保UXLEN=MXLEN或UXLEN=SXLEN。

无论在何种模式下，只要将XLEN设置为小于所支持的最宽XLEN的值，所有操作都必须忽略配置的XLEN以上的源操作数寄存器位，并且必须对结果进行符号扩展以填充目标寄存器中所支持的整个最宽XLEN。类似地，大于XLEN的pc位被忽略，当写入pc时，它被符号扩展以填充支持的最宽XLEN。

#### 存储特权MPRV(Modify PRiVilege)位

修改有效特权模式，即执行加载和存储时的特权级别。当MPRV=0时，加载和存储正常运行，使用当前特权模式的转换和保护机制。当MPRV=1时，加载和存储指令的内存地址将按照当前特权模式（MPP）进行地址转换和保护，并且应用字节序。这意味着当MPRV为1时，加载和存储指令将使用MPP特权模式的地址转换和保护机制，并且按照MPP的字节序进行数据操作。如果不支持U模式，则MPRV位是只读的，并且始终为0。如果MRET或SRET指令将特权模式更改为特权低于M的模式，也会设置MPRV=0。

当前特权模式和MPP指定的特权模式可能具有不同的XLEN设置。当MPRV=1时，加载和存储内存地址被视为当前XLEN被设置为MPP的XLEN。MXR(Make eXecutable Readable)位修改加载访问虚拟内存的权限。当MXR=0时，只可以读取标记为可读的页。当MXR=1时，可以读取标记为可读或可执行(R=1或X=1)的页。当基于页的虚拟内存无效时，MXR不起作用。如果不支持S模式，则MXR为只读0。SUM(允许Supervisor User Memory access)位修改S模式加载和存储访问虚拟内存的权限。当SUM=0时，S模式访问U模式可访问的页将出错。当SUM=1时，允许这些访问。

当基于页的虚拟内存无效时，SUM不起作用。

虽然SUM在不以S模式执行时通常会被忽略，但当MPRV=1且MPP=S时，它会生效。如果不支持s模式，SUM为只读0。MODE为只读0。

#### 端序控制MBE、SBE和UBE位

是WARL字段，它们控制内存访问(指令获取除外)的端序。指令读取总是小端顺序的。MBE控制M模式(假设mstatus.MPRV=0)进行的非指令获取内存访问是小端(MBE=0)还是大端(MBE=1)。

SBE控制S模式进行的显式读取和存储是小端(SBE=0)还是大端(SBE=1)。如果不支持S模式，则SBE为只读0。

UBE控制U模式进行的显式读取和存储是小端(UBE=0)还是大端(UBE=1)。如果不支持U模式，则UBE为只读0。

对于隐式访问超级级别的内存管理数据结构（例如页表），字节序始终由SBE控制。当改变SBE时，会影响对这些数据结构的解释方式。如果在SBE发生变化时仍然使用这些数据结构，M模式的软件必须执行一条SFENCE.VMA指令，并将rs1和rs2都设置为x0，以确保内存管理数据结构的一致性。

如果支持S模式，则实现可以使SBE成为MBE的只读副本。如果支持U模式，则实现可以使UBE成为MBE或SBE的只读副本。

#### 虚拟存储控制TVM（Trap Virtual Memory）位

是一个WARL字段，它支持拦截监控虚拟内存管理操作。当TVM=1时，在S模式下尝试读写satp CSR或执行SFENCE.VMA或SINVAL.VMA指令执行时将引发非法指令异常。当TVM=0时，在S模式下允许这些操作。不支持S模式时，TVM为只读0。

#### TW(Timeout Wait)位

是一个WARL字段，它支持拦截WFI指令。当TW=0时，如果不被其他原因阻止，WFI指令可以在较低权限模式下执行。当TW=1时，如果WFI在任何低特权模式下执行，并且它没有在特定的有限时间内完成，则WFI指令会导致非法指令异常。时间限制可能一直为0，在这种情况下，当TW=1时，WFI总是在较低特权模式下导致非法指令异常。没有比M模式更低的模式时，TW为只读0。如果存在S模式，在U模式下执行WFI会导致非法指令异常，除非它在特定于实现的有限时间限制内完成。

#### TSR (Trap SRET)位

是一个WARL字段，它支持拦截异常返回指令SRET。当TSR=1时，在S模式下执行SRET将引发非法指令异常。当TSR=0时，表示允许。扩展上下文状态

FS[1:0]和VS[1:0] WARL字段和XS[1:0]只读字段分别用于通过设置和跟踪浮点单元和任何其他用户模式扩展的当前状态来降低上下文保存和恢复的成本。FS字段编码浮点单元状态的状态，包括浮点寄存器f0-f31和fcsr、frm和fflags。VS字段用于编码向量扩展状态的状态，包括向量寄存器v0 ~ v31和csr寄存器vcsr、vxrm、vxsat、vstart、vl、vtype和vlenb。XS字段对其他用户模式扩展的状态和相关状态进行编码。

上下文切换例程可以检查这些字段，以快速确定是否需要状态保存或恢复。如果需要保存或恢复，则通常需要额外的指令和csr来影响和优化该流程。

FS、VS和XS字段使用下表的状态编码，四个可能的状态值分别是Off、Initial、Clean和Dirty。根据浮点状态位，可以判断上下文切换的时候，是否需要保存浮点相关寄存器。 当扩展的状态设置为Off时，任何试图读取或写入相应状态的指令都将导致非法指令异常。当状态为Initial时，对应的状态应有一个初始常数值。当状态为Clean时，相应的状态可能与初始值不同，但与存储在上下文交换中的最后一个值匹配。当状态为Dirty时，对应的状态自上次上下文保存以来可能已经被修改。

表 10.4 状态编码

|  |  |  |
| --- | --- | --- |
| Status | FS and VS Meaning | XS Meaning |
| 0 | Off | All off |
| 1 | Initial | None dirty or clean, some on |
| 2 | Clean | None dirty, some clean |
| 3 | Dirty | Some dirty |

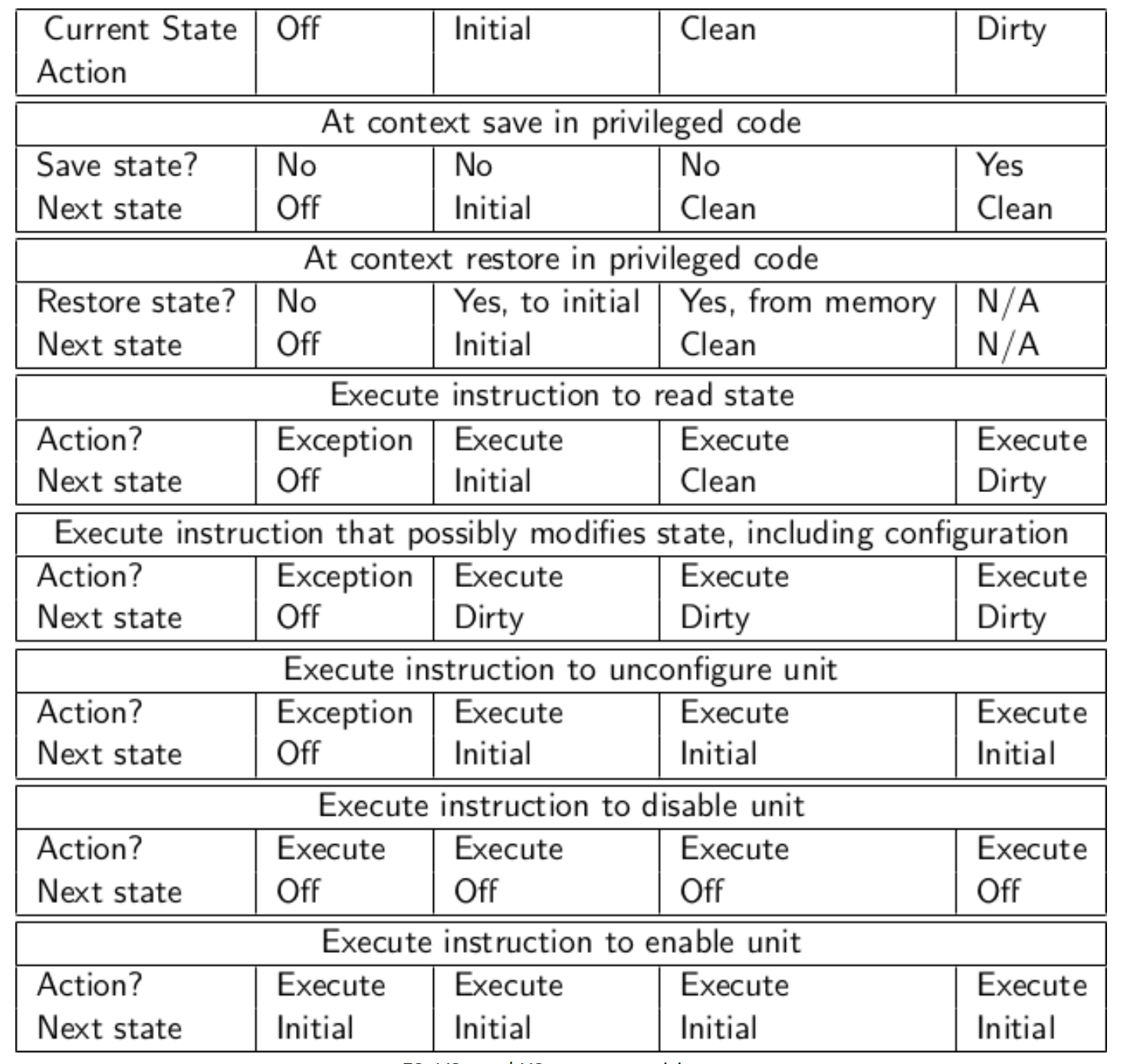


图 10.7 状态转换图

SD位是一个只读位，它总结FS、VS或XS字段是否表示存在一些需要将扩展用户上下文保存到内存中的脏状态。当FS、VS或XS位编码Dirty状态(即SD=(FS==11)或(XS==11)或(VS==11))时设置。这允许特权代码快速确定除了整数寄存器集和PC之外何时不需要额外的上下文保存。

如果FS、XS和VS都是只读零，那么SD也总是零。

### Machine Trap-Vector Base-Address Register (mtvec)

mtvec寄存器是一个mxlen位的WARL读/写寄存器，它保存trap向量配置，包括一个向量基址(base)和一个向量模式(mode)。



图 10.8 mtvec寄存器

当MODE=Direct时，所有进入机器模式的陷阱都会导致pc被设置为BASE字段中的地址。当MODE= vector时，所有进入机器模式的同步异常导致pc被设置为BASE字段中的地址，而中断导致pc被设置为BASE字段中的地址加上中断原因编号的四倍。例如，机器模式定时器中断导致pc被设置为BASE+0x1c。

表 10.5 MODE字段表

|  |  |  |
| --- | --- | --- |
| Value | Name | Description |
| 0 | Direct | All exceptions set pc to BASE. |
| 1 | Vectored | Asynchronous interrupts set pc to BASE+4×cause. |
| ≥2 | — | Reserved |

### Machine Trap Delegation（medeleg/mideleg）

默认状态下，所有trap指令都由机器模式处理。尽管机器模式的trap handler可以利用MRET指令将trap重新委托给更低特群等级。利用medeleg和mideleg当中的读写位可以指定特定的异常和中断由特定的特权等级执行。当一个Trap被委托给S模式处理时我们需要进行如下操作：

需要将异常或中断错误代码写入scause寄存器。将发生异常pc地址写入sepc寄存器。stval寄存器写入异常处理相关信息。mstatus.SPP写入Trap发生时，CPU的特权等级将mstatus.SIE的值写入到mastatus.SPIE清空，mstatus.SIE。mcause寄存器，mepc寄存器，mtval寄存器，mstatus.MPP，mstatus.MPIE不需要写入。

medeleg和mideleg寄存器不应该由任何只读比特位，任何支持被委托的Trap也应该支持不被委托。需要注意的是，Trap不允许由高特权等级向低特权等级传送。例如，如果M模式将Trap委托给了S模式，之后在M模式下运行的软件触发了非法指令异常，那么此时Trap应在M模式处理。相同的情况下，如果S模式下运行的软件触发了非法指令异常，那么Trap应该在S模式执行。被委托的中断会被发出委托的等级忽略，例如，如果Supervisor timer interrupt（STI）被M模式委托给了S模式。那么当机器在M模式运行的时候，STI请求将被忽略。相反，如果STI没有被委托，那么无论机器在任何模式运行的时候，STI请求都会被接受，并且将机器模式转为M模式。

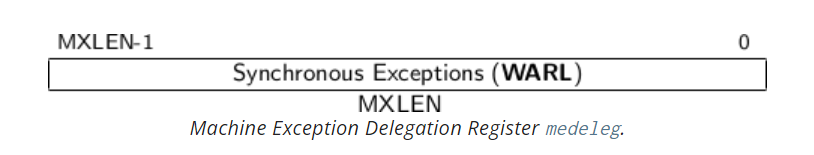


图 10.9 Medeleg寄存器



图 10.10 Mideleg寄存器

medeleg寄存器利用每一个比特的索引值来代表异常的错误代码。通过比较medeleg的索引值和mcause的返回值，来判断当前异常是否被委托给了其他特权等级。mideleg寄存器的比特位排布和mip寄存器的比特位排布相同。例如STIP在mip当中由比特位5控制，在mideleg当中同样有比特位5控制。

### Machine Interrupt Register（mip/mie）

mip寄存器存储了等候中的中断请求的信息。mie寄存器存储了中断使能比特位。mip与 mie寄存器当中的比特位对应mcause当中相同的比特位。[15:0]位为普通终端使用，[31:16]位为用户自定义。

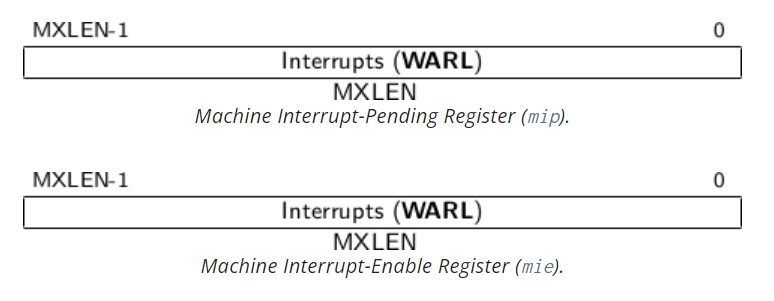


图 10.11 mie/mip寄存器

当满足以下情况是，中断请求信号会陷入到M模式当中：

当前机器运行模式为M，mstatus.MIE为1或者当前特权等级小于M模式mie和mip当中，对应当前中断请求的信号为1。当前中断请求信号在mideleg当中没有被委托需要注意的是，当终端在mip当中等候时，需在有限时间内对当前中断是否可以trap做出判断。在xRET指令执行后立刻判定。陷入到M模式的中断请求比陷入到其他模式当中的中断请求拥有更高优先级。mip寄存器当中的任何一个bit都可以设置为可写或只读。假设mip寄存器当中的bit i是可写的，那么等候中的中断请求i可以通过将mip当中的bit i置零来清除。如果中断请求i是等候状态但是mip中的bit i是只读的，那么具体架构实现必须提供其他方法来清除等候中的中断请求。如果一个中断请求可以被pending，那么mie当中对应的bit位就可以置为可写状态。mie当中的只读位必须设置为只读0。

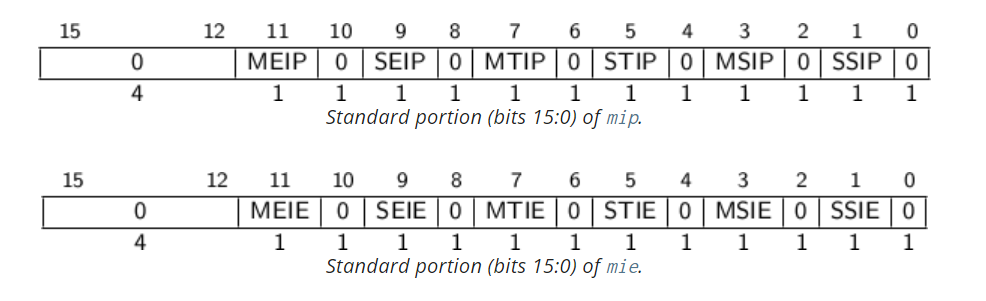


图 10.12 mip/mie寄存器bit位

mip.MEIP和mip.MEIE分别对应机器模式中外部中断的pending和enable。其中mip.MEIP为只读，只能被特定的中断控制器赋值和清除。mip.MEIP和mie.MEIE分别对应时钟中断的中断等候和中断使能信号mip.MTIP为只读位，可以通过写入memory-mapped机器模式时钟对比寄存器来清除。mip.MSIP和mie.MSIE是分别对应机器模式的软件中断等候和中断使能信号。mip.MSIP为只读信号。可通过memory-mapped控制寄存器写入。通常在多hart场景下用来提供machine-level的处理器内部中断信号。每一个hart可以通过相同的memory-mapped控制寄存器来写入mip.MSIP。对于单hart系统来说，或者对于一个支持通过外部中断来发送machine-level中断的架构来说，mip.MSIP和mie.MSIE可以同时为只读恒零。如果架构不支持supervisor模式，mip.SEIP，mip.STIP，mip.SSIP，mie.SIE，mie.STIE，mie.SSIE可以为只读恒零。如果架构支持supervisor模式mip.SEIP和mie.SEIE分别为supervisor-level的外部中断等候和使能信号。其中mip.SEIP为可写位，也可以通过机器模式软件写入。除此之外，架构层面的中断控制器也可以产生supervisor-level的外部中断信号。Supervisor-level的外部中断通过将软件可写的mip.SEIP位和外部中断控制器信号进行logical-OR操作来实现终端等候。当mip寄存器通过CSR指令读取的时候，mip.SEIP通过rd寄存器发挥的值是软件可写位和中断控制器信号的logical-OR结果。但是中断控制器信号并不参与计算SEIP的写入值。只有软件可写mip.SEIP位参与CSRRS指令和CSRRC指令的read-modify-write操作。当supervisor模式启用的时候，mip.STIP和mie.STIE分别对应supervisor-level时钟中断的终端等候和中断使能信号。mip.STIP为只读位，同时也可以通过M模式软件写入来向S模式发送时钟中断信号。当supervisor模式启用的时候，mip.SSIP和mip.SSIE分别对应supervisor-level的软件中断等候和中断使能信号。其中mip.SSIP为可写位，同时也可以被platform中断控制器置1。多个M模式同时中断通过以下优先级处理：MEI，MSI，MTI，SEI，SSI，STI。如果中断被委托给S模式，那么此中断在sip寄存器可被发现，也可以通过sie寄存器忽略。除此之外，sip和sie中对应的bit位为只读恒零。

### Hardware Performance Monitor

M模式包括基本的硬件性能监测单元。mcycle寄存器存储了当前处理器的周期数。minstret存储了退休指令的数量。mcycle和minstret寄存器在rv32和rv64系统当中均有64-bit精度。计数器寄存器在hart重置时候会有一个随意数值，也可以被写入指定数值。任何 CSR 写入都会在写入指令完成后生效。mcycle寄存器可以在同一个core的不同hart之间共享，这种情况下，写入mcycle的操作也会被不同hart看到。Platform应支持查询hart和mcycle的共享关系。硬件性能监测单元包括29个额外的事件计数器：mhpmcounter3-mhpmcounter31，事件选择CSR：mhpmevent3-mhpmevent31。这些寄存器负责控制事件和对应计数器的增长关系。这些事件的定义是由platform决定的。但是事件0代表没有事件。所有计数器都应该被实现，一个符合规定的实现方法就是将计数器及其对应的时间选择器置为只读恒零。

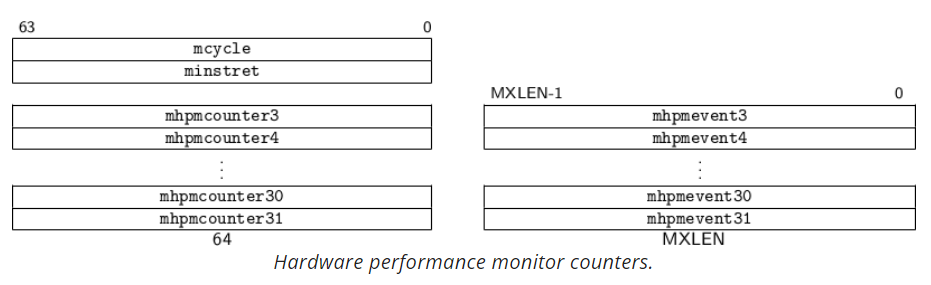


图 10.13 硬件性能监控计数器

Mhpcounter为WARL寄存器，在rv32和rv64系统中最高支持64bit精度。当寄存器位宽为32位时，读取mcycle，minstret，mhpmcountern将返回31-0位的值。同时写操作也将写入31-0位。读取mcycleh，minstreth，mhpmcounterh将返回63-32位，写操作将写入63-32位。

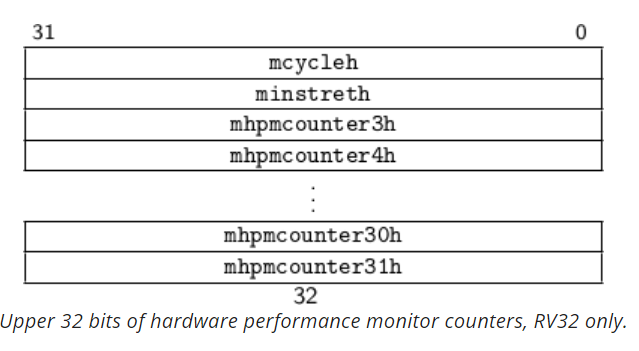


图 10.14 上部32位硬件性能监控计数器

### Machine Counter-Enable Register（mcounteren）

Counter-enable寄存器mcountern是一个32位寄存器。主要负责控制更低特权等级的硬件监测计数器的使能信号。

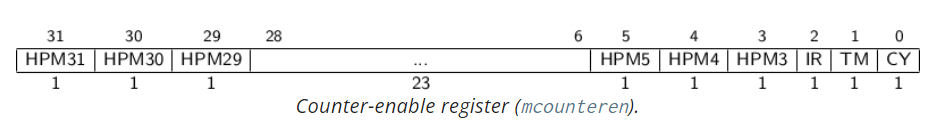


图 10.15 mcounteren寄存器

Mcounteren寄存器的设置之负责控制accessibility。读操作和写操作并不会影响潜在的计数器。计数器数值会持续增加无论是否accessible。当CY，TM，IR或HPMn比特位被清空，任何在U模式或S模式尝试读取cycle，time，instret，或hpmcountern寄存器的操作将会被视为违法指令异常。当其中任何一个比特位被置1后，对应的寄存器访问操作会在更低的特权等级被允许。cycle，instret，hpmcountern CSR是mcycle，minstret，mhpmcounter的只读映射。同理，time CSR是memory-mapped mtime寄存器的只读映射。以上映射关系在RV32I当中也是同理。timeh CSR是mtime寄存器63-32位的只读映射，而time则是31-0位的只读映射。在实现了U模式的系统当中，mcounteren寄存器必须存在。但是所有位都遵循WARL，而且可能是只读恒零。这代表了当机器在更低特权等级运行时，读取相应计数器将导致非法指令异常。在没有实现U模式的系统当中，mcounteren寄存器不应该存在。

### Machine Counter-Inhibit Register（mcountinhibit）

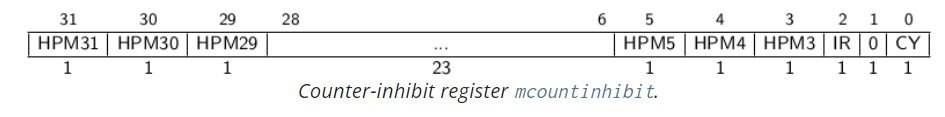


图 10.16 mcountinhibit寄存器

mcountinhibit是一个32位WARL寄存器。主要负责控制硬件监测寄存器数值的增长。寄存器的设置只负责控制寄存器数值增长与否。寄存器的accessibility并不会被mcountinhibit寄存器的设置所影响。当CY，IR，或HPMn比特位被清除后，cycle，instre，hpmcountern寄存器的数值增长不会被影响。当CY，IR，HPMn比特被置1后，相应寄存器的数值增长停止。mcycle寄存器可以在同一个core上被不同的hart共享。在这种情况下，mcountinhibit.CY也会被共享，写入mcountinhibit.CY也会被其他hart发现。如果mcountinhibit寄存器没有被实现，那么相应的操作会被置0。

### Machine Scratch Register（mscratch）

Mscratch寄存器是一个MXLEN长度的读写寄存器。主要用于机器模式下，程序临时保存一些数据。

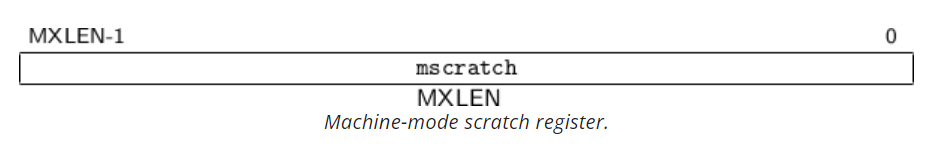


图 10.17 mscratch寄存器

### Machine Exception Program Counter (mepc)

mepc是一个MXLEN-bit读/写寄存器，格式如图所示。mepc的低位（mepc[0]）始终为零。在仅支持IALIGN=32的实现中，两个低位（mepc[1:0]）始终为零。



图 10.18 mepc寄存器

如果一个实现允许IALIGN为16或32（例如，通过更改CSR misa），那么，每当IALIGN=32时，位mepc[1]在读取时被屏蔽，从而看起来为0。这种屏蔽也发生在MRET指令的隐式读取中。尽管被屏蔽，但当IALIGN=32时，mepc[1]仍然是可写的。

mepc是一个WARL寄存器，必须能够保存所有有效的虚拟地址。它不需要能够保存所有可能的无效地址。在编写mepc之前，实现可能会将一个无效地址转换为mepc能够保持的其他无效地址。

当地址转换无效时，虚拟地址和物理地址是相等的。因此，mepc必须能够表示的地址集包括可以用作有效pc或有效地址的物理地址集。

当trap进入M模式时，mepc将使用被中断或遇到异常的指令的虚拟地址写入。否则，mepc永远不会实现编写，尽管它可能是由软件明确编写的。

### Machine Cause Register (mcause)

mcause寄存器是一个MXLEN-bit读写寄存器，其格式如图所示。当trap进入M模式时，mcause会编写一个代码，指示导致trap的事件。否则，mcause永远不会进行编写，尽管它可能是由软件显式编写的。

如果trap是由中断引起的，则会拉高mcause寄存器中的中断位。异常代码字段包含一个标识最后一个异常或中断的代码。表[mcauses]列出了可能的机器级异常代码。异常代码是一个WLRL字段，因此只能保证包含支持的异常代码。

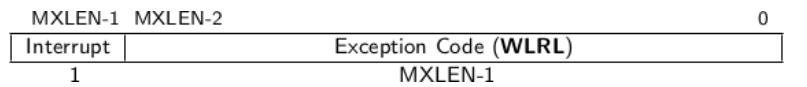


图 10.19 mcause寄存器

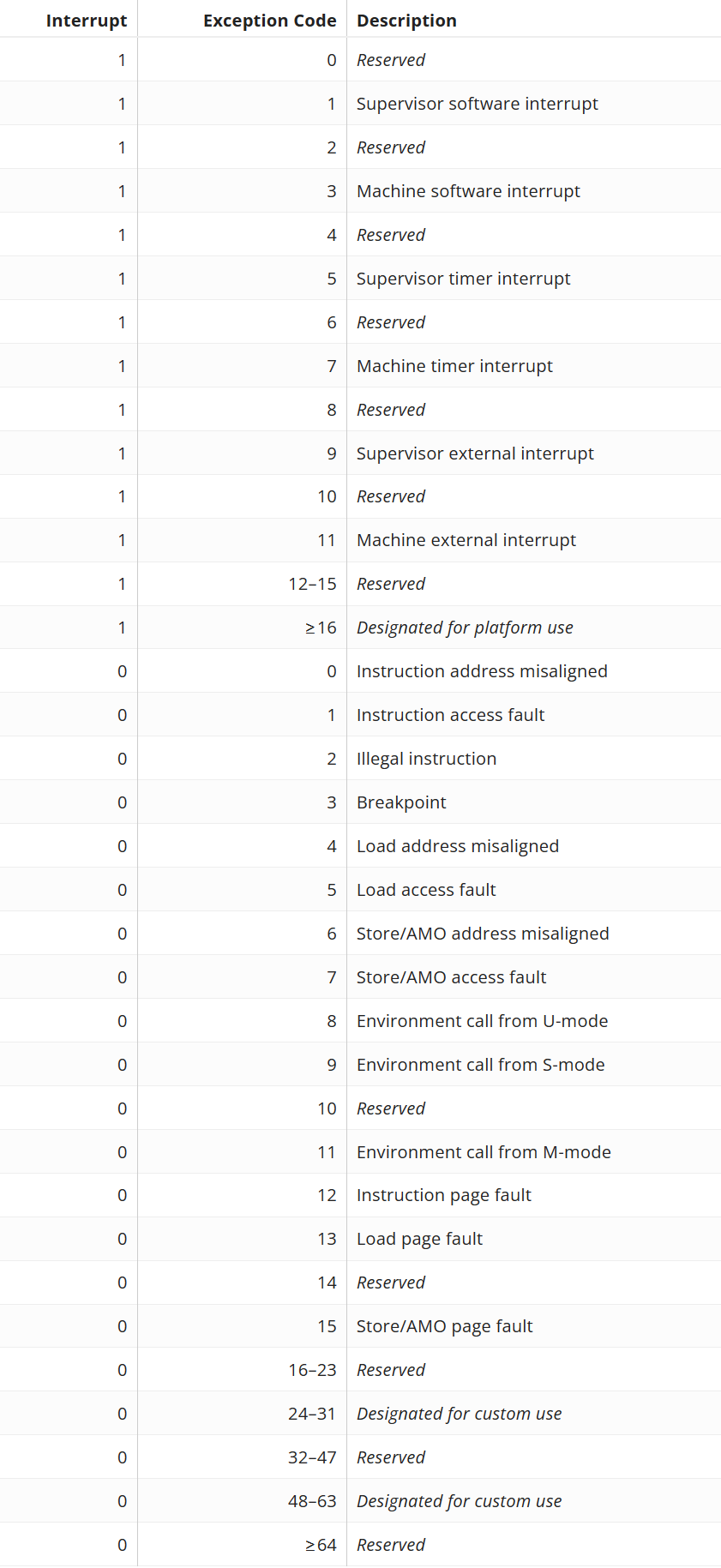


图 10.20 中断异常编码

Load和load-reserved指令会生成加载异常，而store，store-conditional和AMO指令会生成store/AMO异常。

中断可以通过mcause寄存器值符号上的单个分支与其他trap分离。左移可以移除中断位，并缩放异常代码以索引到陷阱向量表中。我们不区分特权指令异常和非法操作码异常。这简化了体系结构，还隐藏了实现支持哪些更高权限指令的细节。为trap提供服务的特权级别可以实现一个策略，决定是否需要区分这些操作码，如果需要，则将给定的操作码视为非法操作码还是特权操作码。

如果一条指令可能引发多个同步异常，则表[exception-priority]的优先级递减顺序指示在mcause中执行和报告哪个异常。任何自定义同步异常的优先级都是由具体实现定义的。

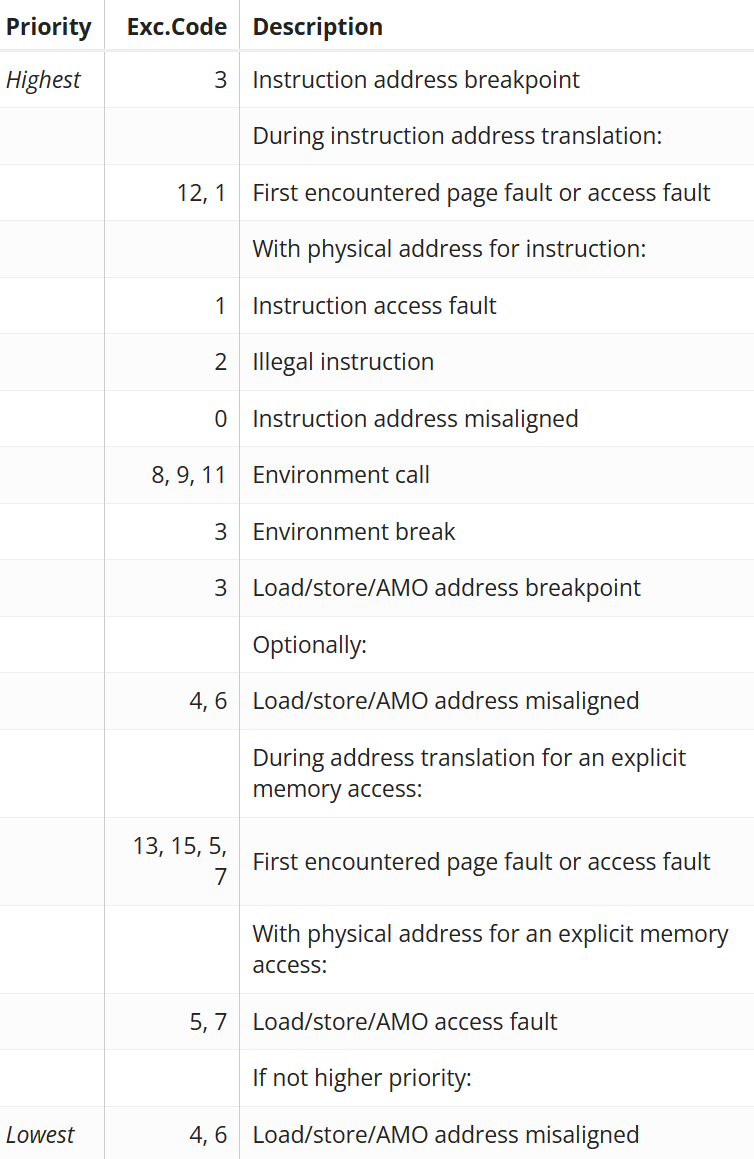


图 10.21 异常优先级

当虚拟地址被转换为物理地址时，地址转换算法确定可能引发的特定异常。

Load/store/AMO地址未对齐异常的优先级可能高于或者低于load/store/AMO页面故障和访问故障异常。

Load/store/AMO地址未对齐和页面错误异常的相对优先级是为了灵活地满足两个设计点而定义的。从不支持未对齐访问的实现可以无条件地引发未对齐地址异常，而无需执行地址转换或保护检查。仅支持对某些物理地址进行未对齐访问的实现必须转换并检查地址，然后才能确定是否可以继续进行未对齐的访问，在这种情况下，引发页面错误异常或访问更合适。

指令地址断点与数据地址断点（也称为观察点）和环境中断异常（由EBREAK指令引发）具有相同的原因值，但优先级不同。

指令地址未对齐异常是由目标未对齐的控制流指令引发的，而不是由获取指令的行为引发的。因此，这些异常的优先级低于其他指令地址异常。

### Machine Trap Value Register (mtval)

mtval寄存器是一个MXLEN位读写寄存器，格式如图所示。当trap进入M模式时，mtval要么设置为零，要么写入异常特定信息，以帮助软件处理trap。否则，mtval永远不会由实现编写，尽管它可能是由软件显式编写的。硬件平台将指定哪些异常必须以信息方式设置mtval，哪些可以无条件地将其设置为零。如果硬件平台指定没有异常将mtval设置为非零值，则mtval为只读零。

如果在指令获取、加载或存储中发生断点、地址未对齐、访问错误或页面错误异常时，mtval是用非零值写入的，则mtval将包含出错的虚拟地址。

当启用基于页面的虚拟内存时，即使对于物理内存访问故障异常，也会使用错误的虚拟地址写入mtval。这种设计降低了大多数实现的数据路径成本，尤其是那些具有硬件页表遍历器的实现。

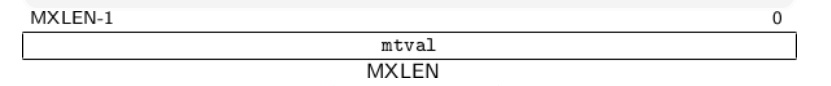


图 10.22 mtval寄存器

如果在未对齐的加载或存储导致访问故障或页面故障异常时，mtval是用非零值写入的，则mtval将包含导致故障的访问部分的虚拟地址。

如果在具有可变长度指令的系统上发生指令访问故障或页面故障异常时，mtval是用非零值写入的，则mtval将包含导致故障的指令部分的虚拟地址，而mepc将指向指令的开头。

mtval寄存器还可以选择性地用于在非法指令异常（mepc指向内存中的错误指令）时返回错误指令位。如果在发生非法指令异常时，mtval是用非零值写入的，则mtval将包含以下值中的最短值：

实际错误指令。出错指令的前ILEN位。错误指令的前MXLEN位。

在非法指令异常时加载到mtval中的值是右对齐的，并且所有未使用的高位都被清除为零。在mtval中捕获出错的指令可以减少指令仿真的开销，从而在指令未对齐的情况下可以避免若干部分指令载入，以及在使用加载将指令取入数据寄存器时可能出现的data cache未命中或slow uncached accesses 。在动态翻译系统中如果另一个代理正在操作指令内存，也存在原子性问题。一个要求是在捕获trap之前将整个指令（或至少第一个MXLEN位）提取到mtval中。这不应该限制实现，实现通常会在尝试解码指令之前获取整个指令，并避免使软件处理程序复杂化。

mtval中的值为零表示不支持该功能，或者提取了非法的零指令。来自mepc指向的指令内存的加载可以用于区分这两种情况（或者，可以询问系统配置信息，以便在运行时之前安装适当的陷阱处理）。对于其他trap，mtval设置为零，但未来的标准可能会重新定义mtval对其他陷阱的设置。如果mtval不是只读零，那么它是一个WARL寄存器，必须能够保存所有有效的虚拟地址和零值。它不需要能够保存所有可能的无效地址。在写入mtval之前，实现可以将无效地址转换为mtval能够保存的其他无效地址。如果实现了返回错误指令位的功能，mtval还必须能够保持小于2N的所有值，其中N是MXLEN和ILEN中的较小值。

### Machine Configuration Pointer Register (mconfigptr)

mconfigptr是一个MXLEN位只读CSR，格式如图所示，它保存配置数据结构的物理地址。软件可以遍历此数据结构，以发现有关harts、平台及其配置的信息。

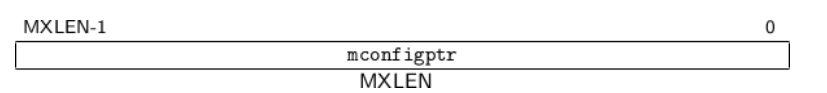


图 10.23 mconfigptr寄存器

以位为单位的指针对齐必须不小于支持的最大MXLEN：即，如果支持的最大MX LEN为8 × n、 则mconfigptr[log2n-1:0]必须为零。

必须实现mconfigptr，但它可能为零，表示配置数据结构不存在，或者必须使用替代机制来定位它。配置数据结构的格式和模式尚未标准化。虽然mconfigptr在某些实现中只是硬接线的，但其他实现可能提供一种方法来配置CSR读取时返回的值。例如，mconfigptr可能会在引导过程开始时显示由平台或M模式软件编程的内存映射寄存器的值。

### Machine Environment Configuration Registers (menvcfg and menvcfgh)

menvcfg CSR是一个MXLEN位读/写寄存器，格式化为MXLEN=64，如下图所示，用于控制特权低于M的模式的执行环境的某些特性。

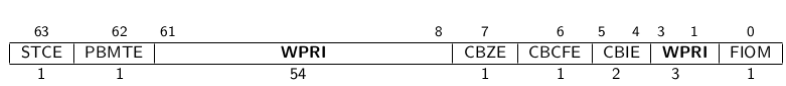


图 10.24 menvcfg寄存器（MXLEN=64）

如果在menvcfg中将位FIOM（fence of I/O意味着内存）置为1，则在权限低于M的模式下执行的Fence指令将被修改，因此对设备I/O的顺序访问要求也意味着对主内存的顺序访问的要求。表https://www.five-embeddev.com/riscv-isa-manual/latest/machine.html#tab:menvcfg-FIOM详细说明了当FIOM=1时，对于特权低于M的模式，FENCE指令位PI、PO、SI和SO的修改解释。

类似地，当FIOM=1时，对于特权低于M的模式，如果访问按设备I/O排序的区域的原子指令设置了aq and/or rl位，则该指令的排序就好像它同时访问设备I/O和内存一样。如果不支持S模式，或者satp.MODE为只读零（始终为Bare），则实现可能会使FIOM为只读零。

表 10.6 指令位及其含义

|  |  |
| --- | --- |
| 指令位 | 含义 |
| PI | 前置设备输入和存储器读取（隐含PR） |
| PO | 前置设备输出和内存写入（隐含PW） |
| SI | 后续设备输入和内存读取（隐含SR） |
| SO | 后续设备输出和内存写入（软件隐含） |

在menvcfg中需要位FIOM，因此M模式可以模拟第[hypervisor]章的hypervisor扩展，该扩展在hypervisor CSR henvcfg中具有等效的FIOM位。PBMTE位控制Svpbmt扩展是否可用于S模式和G级地址转换（即，用于指向satp或hgatp的页表）。当PBMTE=1时，Svpbmt可用于S模式和G阶段地址转换。当PBMTE=0时，表现为未执行Svpbmt。如果未实现Svpbmt，则PBMTE为只读零。此外，对于具有hypervisor扩展的实现，如果menvcfg.PBMTE为零，则henvcfg.PBMTE为只读零。

STCE字段的定义将由即将到来的Sstc扩展提供。在批准扩展之前，其在menvcfg内部的分配可能会发生变化。

CBZE字段的定义将由即将到来的Zicboz扩展提供。在批准扩展之前，其在menvcfg内部的分配可能会发生变化。CBCFE和CBIE字段的定义将由即将到来的Zicbom扩展提供。在批准这一扩展之前，它们在menvcfg内部的分配可能会发生变化。当MXLEN=32时，menvcfg包含与MXLEN=64时menvcfg的位31:0相同的字段。此外，当MXLEN=32时，menvcfgh是一个32位读/写寄存器，它包含与MXLEN=64时menvcfg的位63:32相同的字段。当MXLEN=64时，寄存器menvcfgh不存在。如果不支持U模式，则寄存器menvcfg和menvcfgh不存在。

### Machine Security Configuration Register (mseccfg)

mseccfg是一个可选的MXLEN-bit读/写寄存器，格式如下图所示，用于控制安全功能。

仅当MXLEN=32时，mseccfgh是一个32位读/写寄存器，其包含与MXLEN=64时的mseccfg位63:32相同的字段。

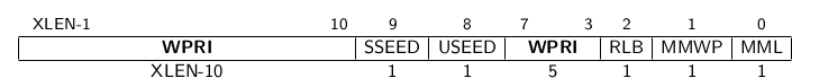


图 10.25 mseccfg寄存器

SSEED和USEED字段的定义将由即将到来的熵源（entropy-source ）扩展Zkr提供。在批准扩展之前，它们在msecfg内的分配可能会发生变化。

RLB、MMWP和MML字段的定义将由即将进行的PMP-enhancement扩展Smepmp提供。在批准扩展之前，它们在msecfg内的分配可能会发生变化。

### Machine Timer Register(mtime and mtimecmp)

mtime寄存器是平台提供的一个实时计数器，用作内存映射机器模式读写寄存器。mtime必须以恒定的频率递增，并且平台必须提供用于确定mtime tick的周期的机制。如果计数溢出，mtime寄存器将wrap around。

在RV32和RV64系统中mtime寄存器均有64bit的精度。平台提供了一个64位内存映射的机器模式定时器比较寄存器（mtimecmp）。每当mtime包含大于或等于mtimecmp的值时，机器计时器中断就会等候，并将这些值视为无符号整数。中断一直保持发布状态，直到mtimecmp大于mtime（通常是写入mtimecmp的结果）。只有当中断被使能并且MTIE位被设置在mie寄存器中时，才会进行中断。

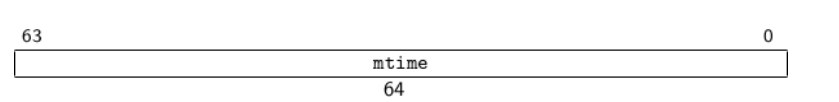


图 10.26 mtime寄存器

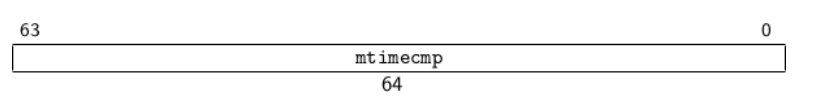


图 10.27 mtimecmp寄存器

定时器设施被定义为使用wall-clock时间而不是循环计数器来支持以高度可变的时钟频率运行的现代处理器，以通过动态电压和频率缩放来节省能量。提供精确的实时时钟（RTC）相对昂贵（需要晶体或MEMS振荡器），并且即使在系统的其余部分无使能时也必须运行，因此系统中通常只有一个位于与处理器不同的频率/电压域中。因此，RTC必须由系统中的所有harts共享，并且对RTC的访问可能会导致voltage-level-shifter 和clock-domain crossing的惩罚。因此，将mtime公开为内存映射寄存器比公开为CSR更合理。

较低的特权级别没有自己的timecmp寄存器。相反，机器模式下软件可以通过将下一个定时器中断多路复用到mtimecmp寄存器中，在hart上实现任意数量的虚拟定时器。

简单的固定频率系统可以使用单个时钟进行周期计数和wall-clock计时。对mtime和mtimecmp的写入最终一定会反映在MTIP中，但不一定会立即反映出来。

如果中断处理程序增加mtimecmp然后立即返回，则可能会发生伪定时器中断，因为在此期间MTIP可能尚未拉低。所有软件都应该被编写为假设该事件是可能的，但大多数软件应该假设该事件极不可能发生。与轮询MTIP直到其失败相比，偶尔引起虚假计时器中断似乎总是更具性能。在RV32中，对mtimecmp的内存映射写入只修改寄存器的一个32位部分。以下代码序列设置一个64位的mtimecmp值，而不会由于比较器的中间值而错误地生成计时器中断：

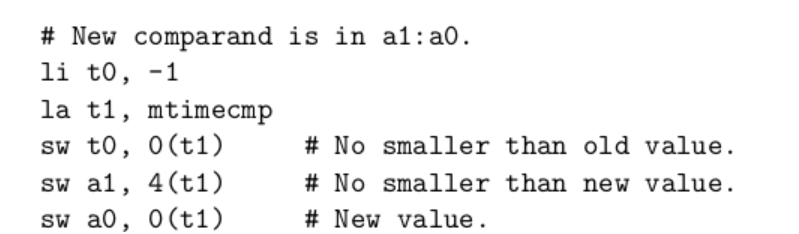


图 10.28 示意图

用于在RV32中设置64位时间比较器的示例代码，假设采用小端序存储系统，并且寄 存器位于Strong Ordered I/O区域中。将-1存储到mtimecmp的低位可以防止mtimecmp暂时小于旧值和新值中的较小值。对于RV64，额外支持对mtime和mtimecmp寄存器的自然对齐64位内存访问，并且是原子访问。

## S模式寄存器

### Supervisor Status Register (sstatus)

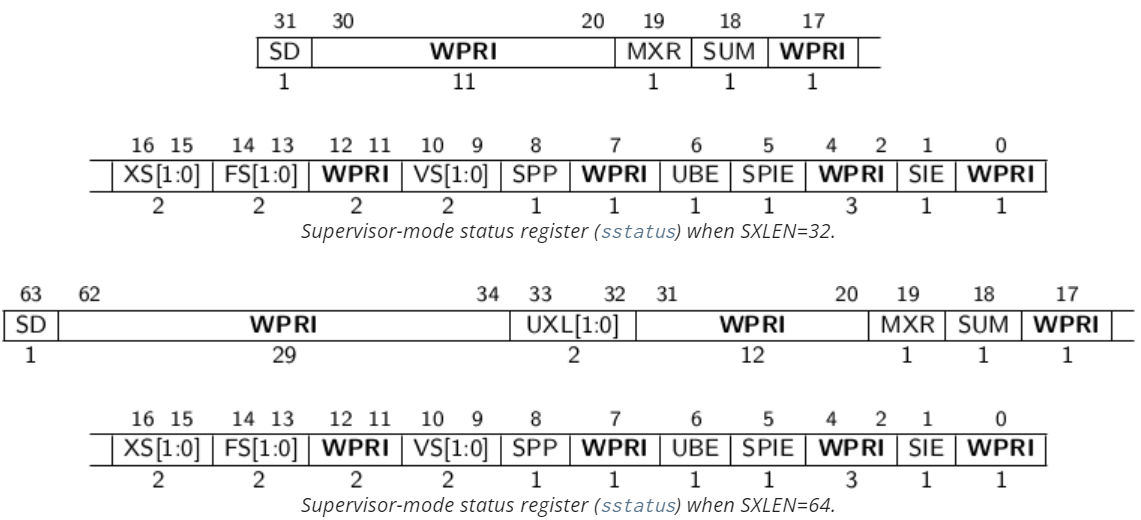


图 10.29 sstatus寄存器

**SPP:**SPP位表示hart在进入supervisor模式之前正在执行的特权级别。当接收到trap时，如果该trap来自user模式，则SPP设置为0，否则设置为1。当执行一条SRET指令(从trap处理程序返回时，如果SPP位为0，则特权级别被设置为用户模式，如果SPP位为1，则特权级别被设置为supervisor模式;SPP设置为0。

**SIE:**SIE位在supervisor模式下启用或禁用所有中断。当SIE为0时，在supervisor模式下禁用中断。当hart在user模式下运行时，SIE中的值将被忽略，并且启用管理员级中断。supervisor可以使用sie CSR禁用单个中断源。

**SPIE:**SPIE位表示在进入supervisor模式之前是否使能了supervisor中断。当trap进入supervisor模式时，SPIE设置为SIE, SIE设置为0。当执行一条SRET指令时，SIE被设置为SPIE，然后SPIE被设置为1。

**UXL:** UXL字段控制user模式的XLEN值。当SXLEN=32时，UXL字段不存在，UXLEN=32。当SXLEN=64时，它是一个编码UXLEN当前值的WARL字段。

**MXR:** MXR位控制加载访问虚拟内存的权限。当MXR=0时，只能访问可读的内存区域。当MXR=1时，可以访问可读和可操作的内存区域。如果虚拟内存无效，MXR没有用处。

**SUM:**SUM修改supervisor模式加载和存储访问虚拟内存的权限, SUM机制防止管理软件无意中访问用户内存。当SUM=0时，supervisor模式内存访问user模式可访问的页面将出错。当SUM=1时，允许这些访问。当基于页面的虚拟内存无效时，SUM不起作用，当以user模式执行时也不起作用。无论SUM的状态如何，s模式永远不能执行来自用户页面的指令

**UBE:**UBE位用于控制user模式下是否启动错误指令异常，当设置为1时，用户模式下的错误指令会引发异常，设置为0时，用户模式下的错误指令会被忽略，不会引发异常。

### Supervisor Trap Vector Base Address Register(stvec)

stvec寄存器是一个sxlen位的读/写寄存器，它保存trap向量配置，由一个向量基址(base)和一个向量模式(mode)组成。

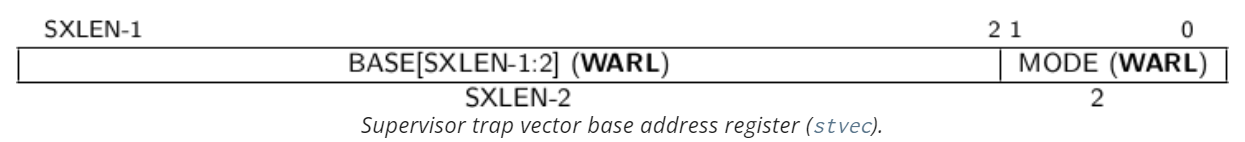


图 10.30 stvec寄存器

stvec中的BASE字段是一个WARL字段，它可以保存任何有效的虚拟地址或物理地址。

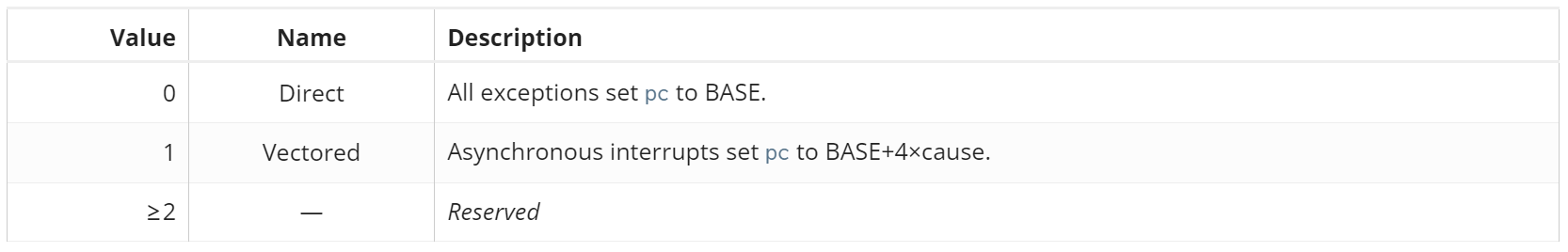


图 10.31 MODE字段编码

MODE字段的编码如表所示。当MODE=Direct时，所有进入supervisor模式的trap都会将pc设置为BASE字段中的地址。当MODE= vector时，所有进入supervisor模式的同步异常导致pc被设置为BASE字段中的地址，而中断导致pc被设置为BASE字段中的地址加上中断原因编号的四倍。

### Supervisor Interrupt Registers (sip and sie)

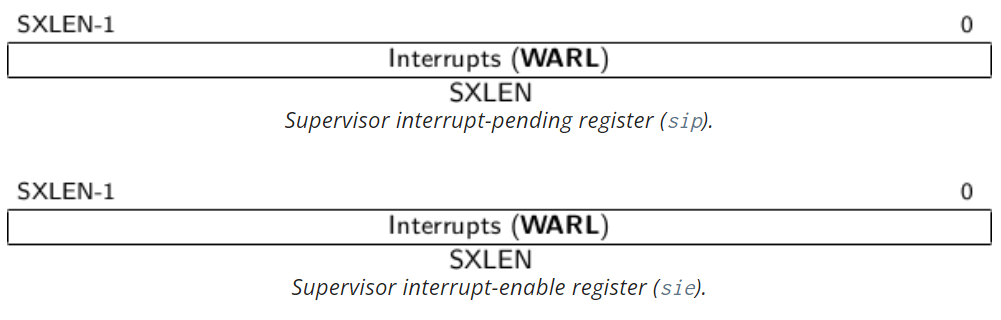


图 10.32 sip/sie寄存器

sip寄存器是一个包含挂起中断信息的sxlen位读/写寄存器，而sie是对应的包含中断使能位的sxlen位读/写寄存器。15:0 bit位分配给中断原因。16bit以及以上的位用于自定义使用。

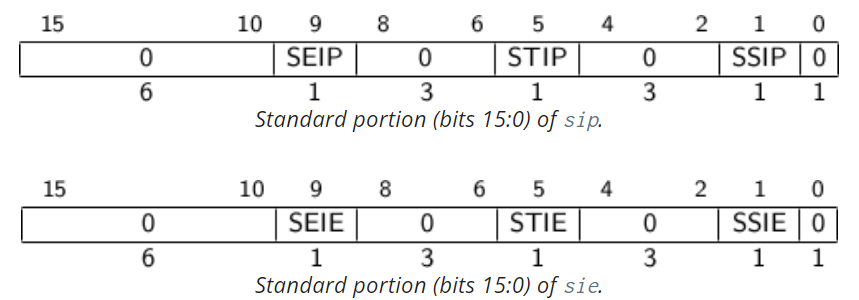


图 10.33 sip/sie寄存器[15:0] bit

sip.SEIP和sie.SEIE用于supervisor级外部中断的中断挂起和中断使能位。

sip.STIP和sie.STIE是supervisor级定时器中断的中断挂起和中断启用位。

sip.SSIP和sie.SSIE是supervisor级软件中断的中断挂起和中断启用位。

### Counter-Enable Register (scounteren)

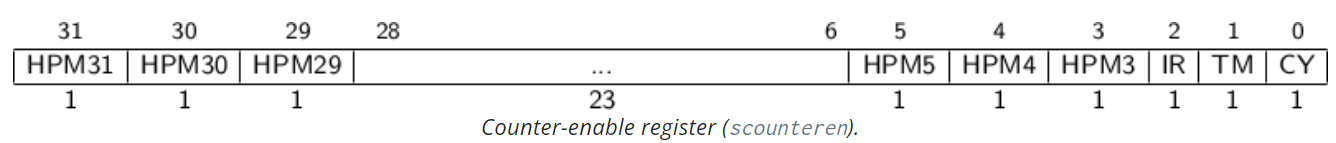


图 10.34 scounteren寄存器

counter-enable寄存器scounteren是一个32位寄存器，它控制硬件性能监控计数器到user模式的可用性。当计数器寄存器中的CY、TM、IR或HPMn位清除时，在user模式下执行时试图读取周期、时间等寄存器将导致非法指令异常。当这些位之一被设置时，就允许访问相应的寄存器。

### Supervisor Scratch Register (sscratch)



图 10.35 sscratch寄存器

Supervisor Scratch Register（SSR）是一个用于Supervisor级别软件的临时存储寄存器。它提供了一个供Supervisor模式的软件使用的、不受保存和恢复规则限制的工作区域。Supervisor Scratch Register可以用于保存Supervisor级别软件在执行期间需要暂时保存的数据、状态或临时变量。由于它不受保存和恢复规则的限制，Supervisor级别的软件可以自由地使用这个寄存器，而不必担心影响到其他寄存器的值。

### Supervisor Exception Program Counter (sepc)

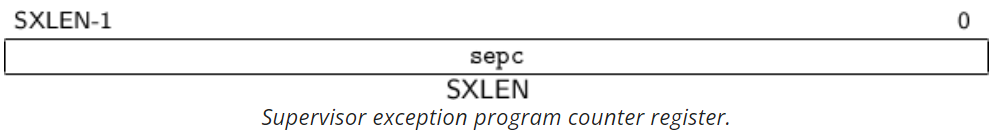
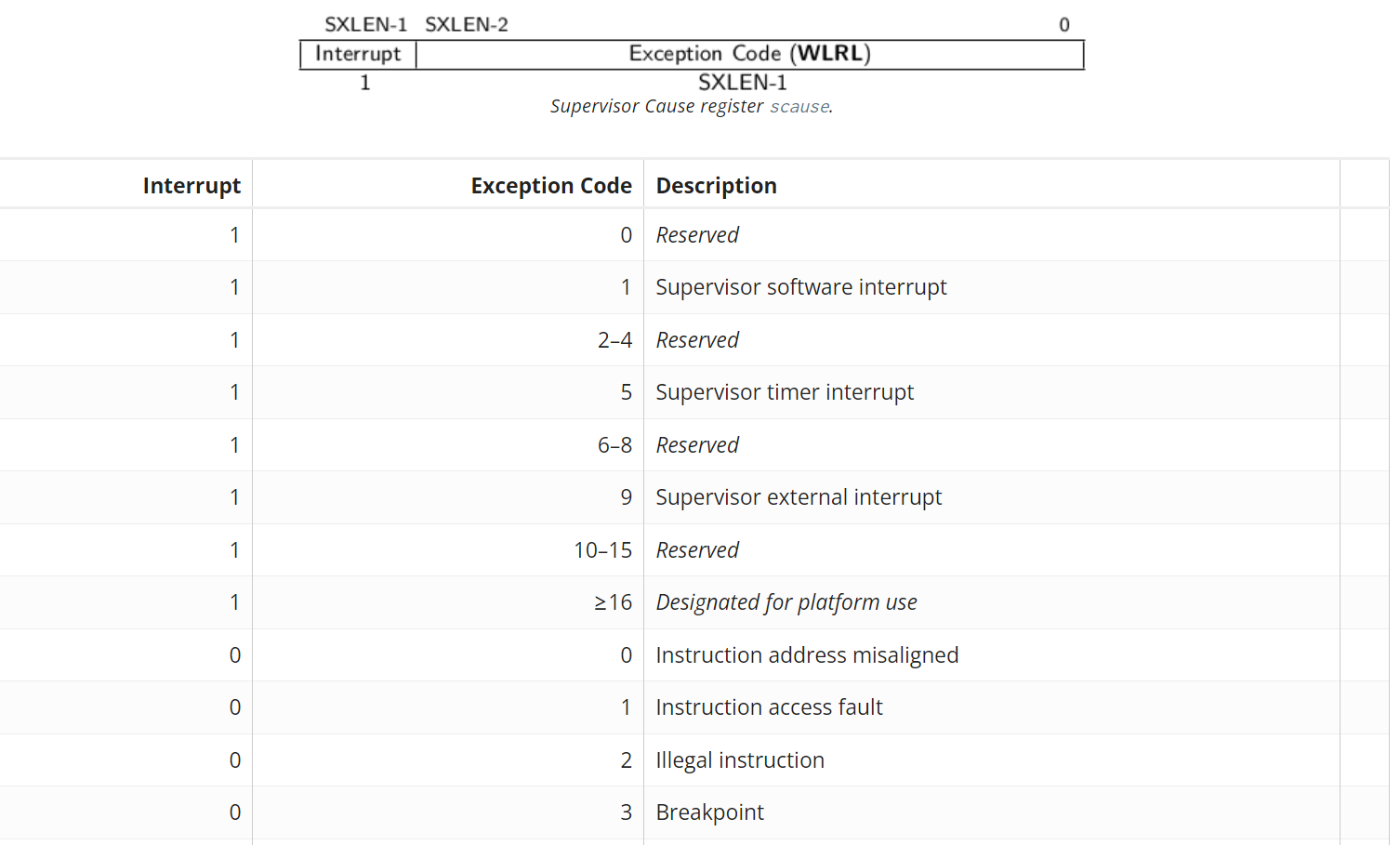


图 10.36 spec寄存器

Supervisor模式下的Supervisor Exception Program Counter（SEPC）用于存储发生异常时的下一条指令的地址。当在Supervisor模式下发生异常（如中断或故障）时，处理器会将SEPC设置为发生异常的指令的地址，以便在异常处理程序执行完毕后能够正确返回到异常指令的下一条指令。

### Supervisor Cause Register (scause)

Supervisor模式下的Supervisor Cause Register（SCAUSE）用于存储异常或中断的原因。



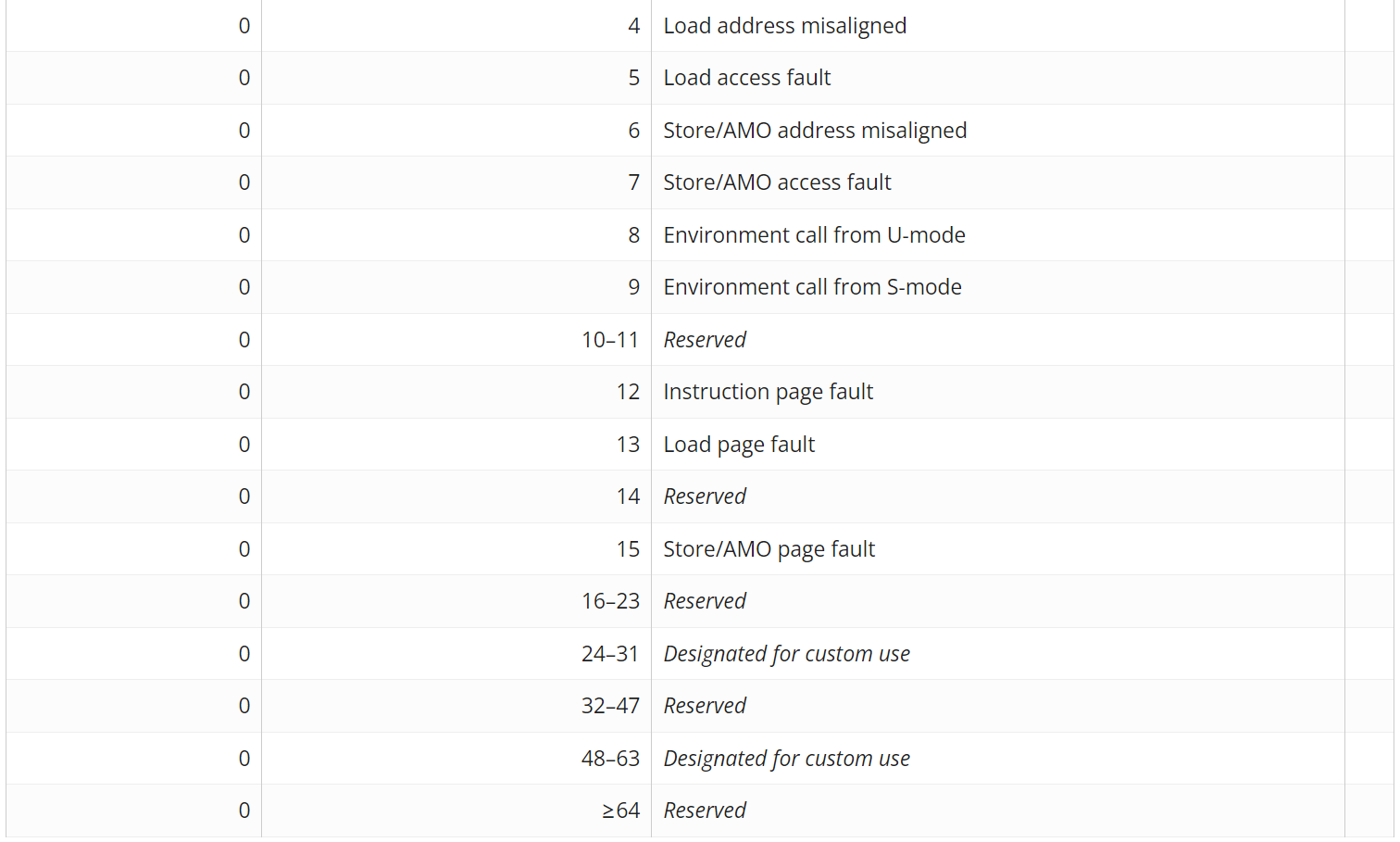


图 10.37 scause寄存器

### Supervisor Trap Value (stval) Register

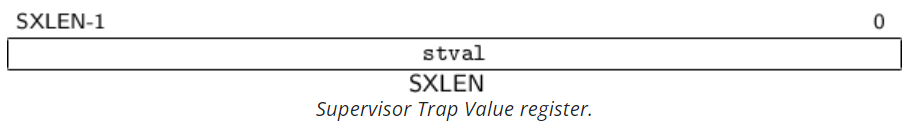


图 10.38 stval寄存器

Supervisor Trap Value Register（STVAL）用于存储导致异常或中断的指令或访问的引发的地址或值。例如在在指令获取、加载或存储中发生断点、地址不对齐、访问错误或页面错误异常等。

### Supervisor Environment Configuration Register (senvcfg)

senvcfg CSR是一个读/写寄存器，它控制user模式执行环境的配置。

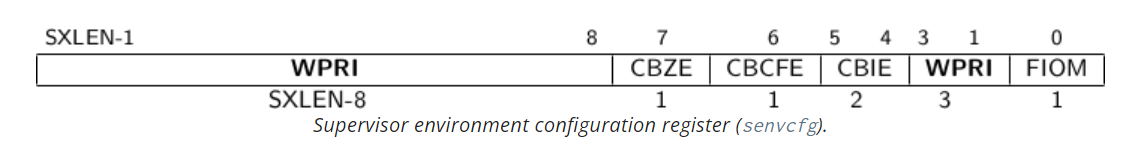


图 10.39 senvcfg寄存器

FIOM:设置为1，在user模式下执行的Fence(用于控制内存访问的顺序和可见性)指令会被修改,可以控制对设备 I/O 的访问顺序要求也包括对主存访问的顺序要求。user模式下Fence指令为PI,PO,SI和SO和修改解释如下。

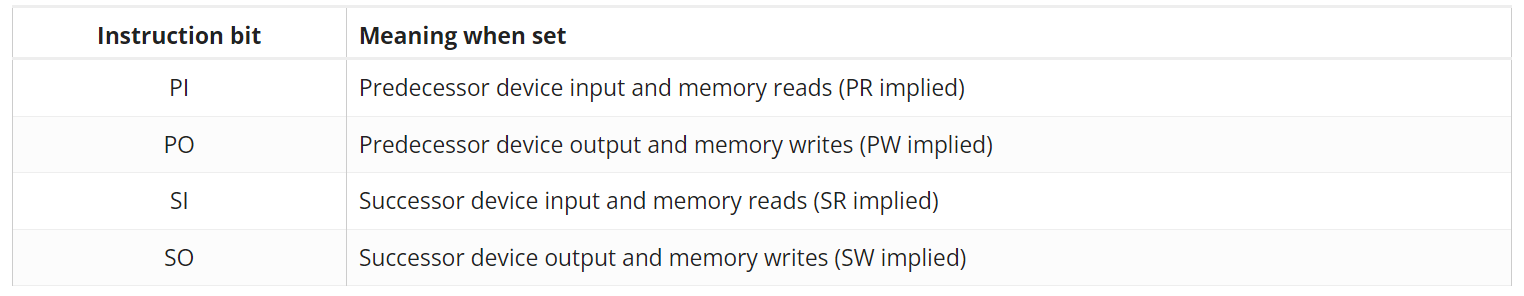


图 10.40 Fence指令

### Supervisor Address Translation and Protection (satp) Register

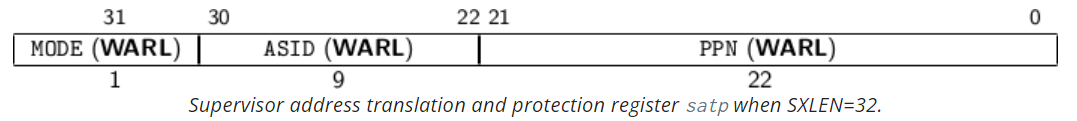


图 10.41 satp寄存器

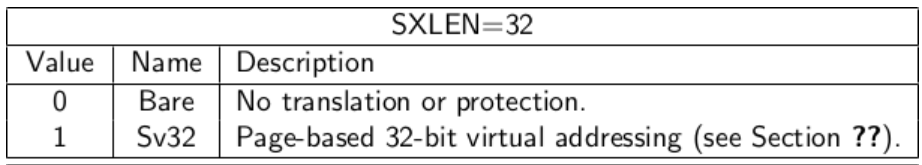


图 10.42 MODE位描述

Satp寄存器用于配置和管理 Supervisor 模式下的地址转换和保护机制。satp 寄存器的作用是指定用于虚拟地址到物理地址转换的页表根据址（Page Table Root）以及当前的地址转换模式。

satp 寄存器的位字段解释如下：

MODE（位 31）：用于指示地址转换模式，0 表示禁用地址转换（Bare模式），1 表示启用 Sv32 模式。

ASID（位 30:22）：用于指定地址空间标识符，用于与 TLB（转换查找缓冲）相关联，以区分不同的地址空间。

PPN（位 21:0）：用于指定页表根据址（Page Table Root）的物理页号，用于虚拟地址到物理地址的转换。

satp 寄存器在 Supervisor 模式下的使用非常重要，它确定了地址转换和内存保护的机制。通过配置合适的页表根据址和地址空间标识符，satp 寄存器允许 Supervisor 模式下的代码和数据访问正确的物理内存位置，并提供内存保护功能，以确保不同的地址空间之间的隔离和安全性。

# Biriscv实现

## M-mode

### csr\_mstatus

mstatus是32位的读写寄存器，用于跟踪和控制当前hart的操作状态。

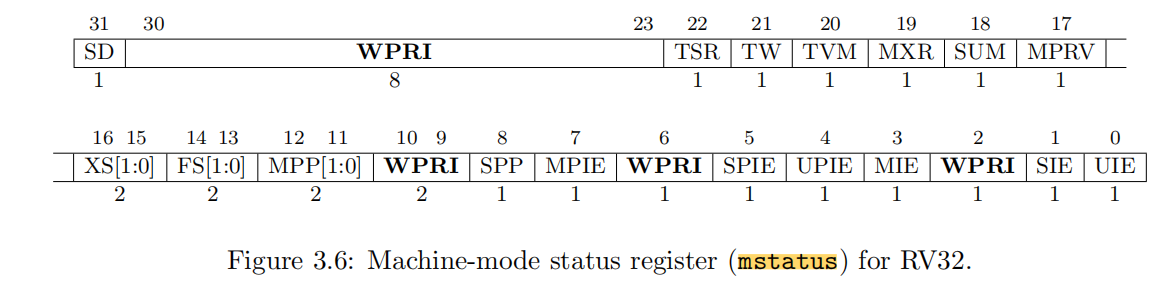


图 11.1 mstatus寄存器

#### SR\_xIE(xIE)

用于控制全局中断的启用和关闭。

#### SR\_xPIE,SR\_xPP(xPIE,xPP)

当触发x模式的中断异常时，xIE的值被设置为0，xPIE用于保存之前xIE的使能状态，xPP用于保存之前所处的特权模式。当执行xRET时，xIE被设置为xPIE的值，特权模式改变为xPP中保存的特权模式，并且xPP被设置为0。

#### SR\_SUM（SUM）

用于修改S模式加载和存储访问虚拟内存的权限，当SUM=0时，S模式存储器对U模式可访问的页面进行访问时将发生故障。当SUM=1时允许这些访问，当基于页面的虚拟内存无效时，SUM无效。

#### SR\_MPRV（MPRV）

MPRV位修改在所有特权模式下加载和存储执行的特权级别。当MPRV=0时，加载和存储行为正常，使用当前特权模式的转换和保护机制。当MPRV＝1时，使用MPP中的特权模式的转换和保护机制。

#### SR\_MXR（MXR）

用于修改加载访问虚拟内存的权限，当MXR=0时，我们可以load标记为可读（R=1）的页面。当MXR＝1时，我们可以load标记为可读或者可执行的（R=1或X=1）的页面。

### csr\_mcause（mcause）

当触发一个trap并进入m模式时，mcause中会保存触发的中断异常的编码。

### csr\_mtval(mtval)

当触发一个trap并进入m模式时，mtval要么设置为零，要么写入异常特定信息，以帮助软件处理陷阱。由硬件平台将指定哪些异常必须设置mtval，哪些异常可以无条件地将其设置为零。下图是mtval的设置规则。在biriscv中，对于非法指令，依然将内存访问的地址存入mtval中。

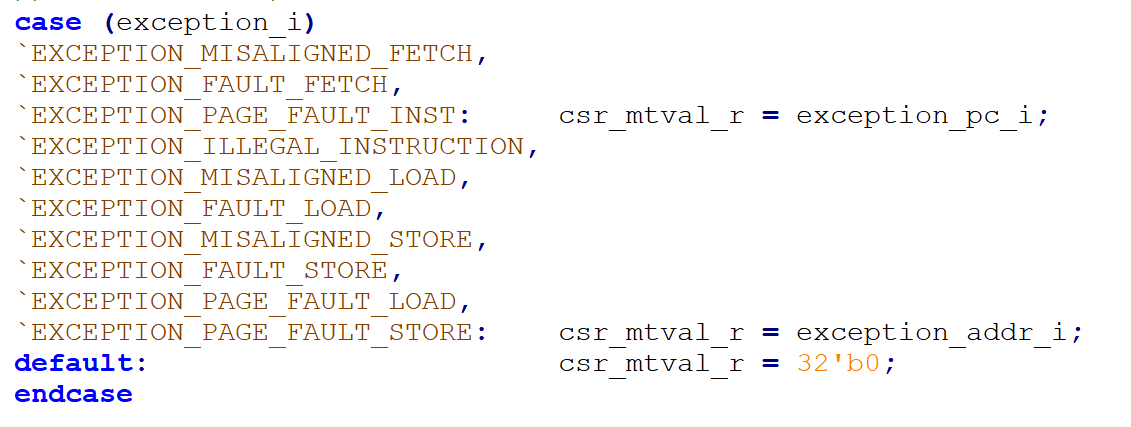


图 11.2 代码示例

### csr\_mtvec(mtvec)

用于保存陷阱矢量的配置。

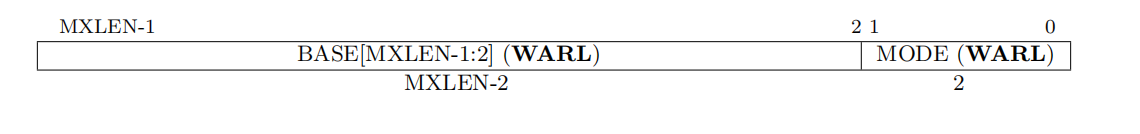


图 11.3 mtvec寄存器

如果MODE域的值为0，则所有陷入机器模式的trap都会将pc设置为BASE字段中的地址。如果MODE域的值为1，那么所有陷入机器模式的trap都会将pc的值设置为BASE+4×cause

csr\_mtvec中保存着trap\_vector的地址

### csr\_mip（mip）,csr\_mie(mie)

mip中包含着挂起中断的信息，mie包含着中断使能的相应信息。在biriscv中，mie仅有MSIE,MTIE,MEIE三位可读可写，mip仅有MSIP,MTIP.MEIP三位可读可写。

### csr\_mscratch(mscratch)

mscratch寄存器是专用于机器模式的MXLEN位读/写寄存器。通常，它用于保存指向机器模式hart本地上下文空间的指针，这里mscratch中保存着机器模式的sp指针，并在进入M模式陷阱处理程序时与用户寄存器交换。

### csr\_mcycle，csr\_mcycle\_h，csr\_mtimecmp(mcycle,mtimecmp)

mcycle CSR用于统计处理器内核执行的时钟周期数,mcycle有64位,在biriscv中用mcycle的低32位与csr\_mtimecmp进行比较，相等时发起定时中断。

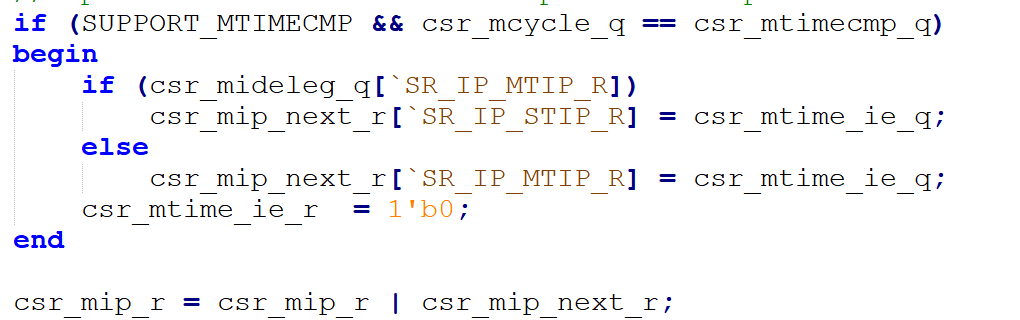


图 11.4 代码示意图

### csr\_medeleg，csr\_mideleg(medeleg,mideleg)

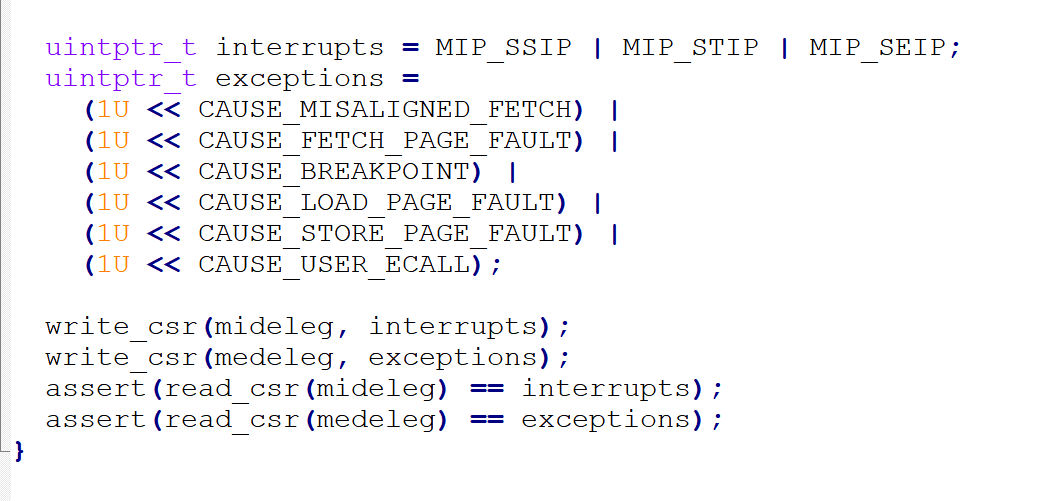


图 11.5 代码示意图

如上图所示，这里将S模式中断和大多数异常直接委托给S模式

### csr\_mepc(mepc)

在进入机器模式中断异常后保存进入中断异常之前pc指针的值

### csr\_misa(misa)

由硬件指定，表明该核支持rv32I基本指令集，并支持M扩展（这里SUPPORT\_MULDIV默认值为1）



图 11.6 代码示意图

### csr\_mhartid(mhartid)

由于biriscv中只有一个硬件线程，mhartid的值恒为零。Mode在biriscv中没有定义sstatus,在进入s模式的中断异常前，在硬件上，将mstatus中SPP域的值更新为进入中断异常之前的特权模式，mstatus的SPIE更新为SIE的值，SIE清零。

### csr\_sepc（sepc）

在进入S特权模式中断异常前保存进入中断异常之前pc指针的值

### csr\_stvec(stvec)

在csr\_stvec中保存handle\_exception的地址,在进入S模式中断异常时，从handle\_exception开始执行。

### csr\_scause(scause)

在进入S模式中断异常前，scause中会保存触发的中断异常的编码

### csr\_stval(stval)

当触发一个trap并进入s模式时，stval要么设置为零，要么写入异常特定信息，以帮助软件处理陷阱。由硬件平台将指定哪些异常必须设置stval，哪些异常可以无条件地将其设置为零。下图是biriscv中stval的设置规则。

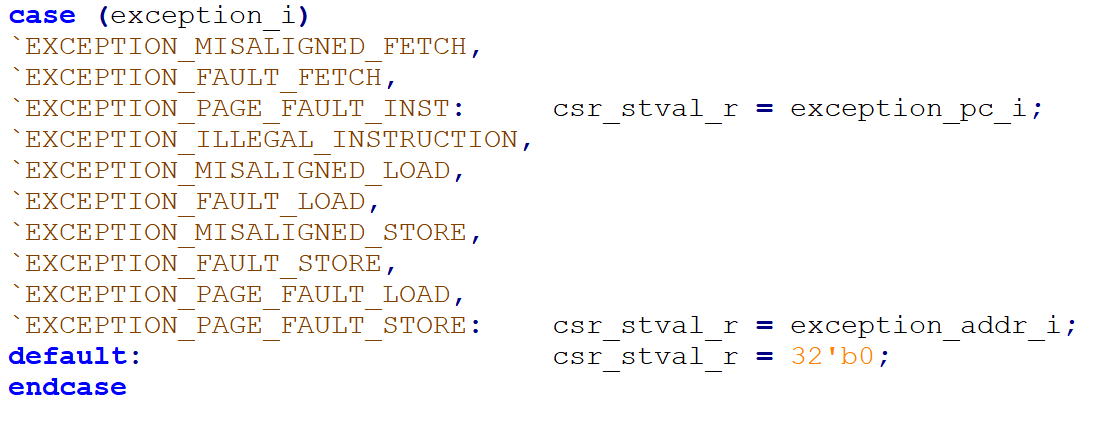


图 11.7 代码示意图

### csr\_satp(satp)

当MODE位为0时，虚拟地址不会进行转化，此刻的虚拟地址就是物理地址。MODE为不同值时，会根据MODE位选择不同结构的页表。ASID（Address Space Identifier， 地址空间标识符）域是可选的，它可以用来降低上下文切换的开销。PPN存放的是最高级页表的起始页号（页目录），根据相应的page大小进行可以得到最高级页表（页目录）的物理地址。这样cpu就可以告诉虚拟内存地址从哪里翻译成物理内存地址。

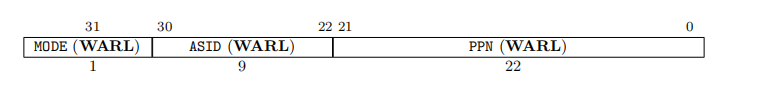


图 11.8 satp寄存器

### csr\_sscratch

sscratch用于在hart执行用户代码时保持指向hart本地S-mode上下文的指针，在进入中断异常后，sscratch与用户态的sp指针的值进行交换，此时sp指向S-mode堆栈。

### csr\_sie,csr\_sip

在biriscv中，sie仅有SSIE,STIE,SEIE三位可读可写，sip仅有SSIP,STIP,SEIP三位可读可写

# 局部中断控制器（CLINT）

Birsicv的中断控制器模块CLINT主要负责处理软件中断。 CLINT中断控制器占据64KB地址空间。模块由两个模块组成：CLINT单元和CLINT寄存器单元。其中CLINT单元为顶层模块，CLINT寄存器单元负责实现MSIP寄存器，和SSIP寄存器。

## IPI中断

CLINT模块可以用于生成IPI中断。CLINT中实现了两个32位msip和两个32位ssip寄存器。软件中断通过配置地址映射的软件中断配置寄存器进行中断信号的控制。具体映射地址可查询表 12.1 CLINT寄存器的存储器映射地址。

### 寄存器

M（Machine）模式和S（Supervisor）模式的软件中断分别由msip和ssip寄存器进行控制。

该寄存器只有最低位为有效位，有效位将作为软件中断信号发送给CPU核。当软件将msip寄存器的有效位置1后，CSR寄存器mip中的msip域会置1表示当前处于中断等待状态。软件也可以通过写0置寄存器来清除软件中断。对于软件中断，CLINT模块将读取msip寄存器的最低位，当最低位为1的时候，CLINT模块将触发IPI中断信号。

#### msip



图 12.1 msip寄存器

表 12.2 msip寄存器描述

|  |  |  |  |
| --- | --- | --- | --- |
| 比特位 | 字段名称 | 属性 | 解释 |
| [0:0] | msip | RW | M模式软件中断是通过写入存储器映射的控制寄存器msip产生的。msip寄存器是一个32位宽的WARL寄存器，其中高31位绑定到0。最低有效位可用于驱动RISC-V 核中CSR 寄存器mip的MSIP位。msip寄存器中的其他位硬连线到0。重置时，msip寄存器被清除为零。 |
| [31:1] | Reserved | RW | 读取时返回0，写入时无效。 |

#### ssip

表格

中度可信度描述已自动生成

图 12.2 ssip寄存器

表 12.3 ssip寄存器描述

|  |  |  |  |
| --- | --- | --- | --- |
| 比特位 | 字段名称 | 属性 | 解释 |
| [0:0] | ssip | RW | S模式软件中断是通过写入存储器映射的控制寄存器ssip产生的。ssip寄存器是一个32位宽的WARL寄存器，其中高31位绑定到0。最低有效位可用于驱动RISC-V 核中CSR 寄存器sip的SSIP位。ssip寄存器中的其他位硬连线到0。重置时，ssip寄存器被清除为零。 |
| [31:1] | Reserved | RW | 读取时返回0，写入时无效。 |

### 中断处理过程

#### RV Linux Kernel发起中断请求

内核程序调用smp相关函数，这里以Linux进程调度为例，首先Linux内核调度程序调用smp\_send\_reschedule()函数，该函数将调用send\_ipi\_single()函数，并将CLINT中msip寄存器最低位置1，触发软件中断。

#### CLINT接收中断

软件将中断控制器中msip寄存器相应MSIP位置1。CLINT通过读取msip寄存器接受中断请求，并通过softirq引脚发送中断请求给CPU Core

#### CSR更改相应寄存器

mip.MSIP位被置1后，同时mie.MSIE为1，则开始处理中断。此时，处理器执行下列操作：

•处理器执行完当前指令，保存下一条指令的 PC 到 mepc 中。

•设置 mcause 的中断标记为 1，将中断编号写入 mcause，并更新 mtval 为 0。

•将 mstatus 的中断使能位 MIE 保存到 MPIE 中，将 MIE 清零，禁止响应中断。

•将发生中断之前的权限模式保存到 mstatus 的 MPP 中，切换到机器模式。

•（mtvec.Mode=0，直通中断）：PC 从 mtvec 处执行。通常，取回的指令是一条跳转指令，跳转至顶层处理函数。该函数通过分析 mcause 获取中断编号，并调用该编号对应的处理函数。

•（mtvec.Mode=1，矢量中断）：PC 从 mtvec.Base + 4 \* 中断编号处执行。通常，取回的指令是一条跳转指令，跳转至相应中断的处理函数。

#### 中断处理程序执行

CPU跳转至软件中断处理函数入口，软件判断中断为IPI中断，跳转至IPI中断处理函数入口，根据IPI中断ID跳转至相应的IPI处理函数并执行。

#### 中断处理程序执行完毕

中断返回执行 mret 指令可以实现中断返回。此时，处理器执行下列操作：

• 将 mepc 恢复到 PC。（mepc 保存的是下一条指令的 PC，所以无需调整）

• 将 mstatus.MPIE 恢复到 mstatus.MIE。

• 从 mstatus.MPP 恢复发生中断之前的权限模式。

# 中断

## 时钟中断

在Biriscv当中，时钟中断由CPU Core内部的CSR单元触发。Biriscv的每个Hart都定义了一个mtime寄存器和mtimecmp寄存器。由操作系统负责寄存器值的加减，并触发相应的时钟中断，在触发时钟中断后，操作系统将根据jiffie值对mtimecmp寄存器的值更新，准备下一次时钟中断。

### 寄存器

#### mtime



图 13.1 mtime寄存器

表 13.1 mtime寄存器描述

|  |  |  |  |
| --- | --- | --- | --- |
| 比特位 | 字段名称 | 属性 | 解释 |
| [63:0] | mtime | RW | 多核系统中仅存在一个64位系统计时器mtime。mtime位于CSR中，用于跟踪从任意时间点开始计数的周期数。它是一个自由运行的计数器，每次循环数递增一次。 |

#### mtimecmp



图 13.2 mtimecmp寄存器

表 13.2 mtimecmp寄存器描述

|  |  |  |  |
| --- | --- | --- | --- |
| 比特位 | 字段名称 | 属性 | 解释 |
| [63:0] | mtimecmp | RW | 位于CSR中，保存一个64位的值。每当mtime大于或等于mtimecmp寄存器中的值时，计时器中断就会挂起。定时器中断用于驱动CSR 寄存器mip的MTIP位。 |

### 中断处理过程

#### CLIINT发起时钟中断请求

CLINT比较mtimecmp[63:0]≤mtime[63:0]时，产生中断。

#### CSR更改相应寄存器

mip.MSIP位被置1后，同时mie.MSIE为1，则开始处理中断。此时，处理器执行下列操作：

* 处理器执行完当前指令，保存下一条指令的 PC 到 mepc 中。
* 设置 mcause 的中断标记为 1，将中断编号写入 mcause，并更新 mtval 为 0。
* 将 mstatus 的中断使能位 MIE 保存到 MPIE 中，将 MIE 清零，禁止响应中断。
* 将发生中断之前的权限模式保存到 mstatus 的 MPP 中，切换到机器模式。
* mtvec.Mode=0，直通中断）：PC 从 mtvec 处执行。通常，取回的指令是一条跳转指令，跳转至顶层处理函数。该函数通过分析 mcause 获取中断编号，并调用该编号对应的处理函数。
* （mtvec.Mode=1，矢量中断）：PC 从 mtvec.Base + 4 \* 中断编号处执行。通常，取回的指令是一条跳转指令，跳转至相应中断的处理函数。

#### 中断处理程序执行

RV Linux Kernel接收到来自CPU Core的时钟中断请求，进入IRQ处理函数并执行时钟中断相关处理函数，读取mtimecmp中的值，并根据系统jiffie值进行更新，并写回mtimecmp寄存器。（待完善）

#### 中断处理程序执行完毕

中断返回执行 mret 指令可以实现中断返回。此时，处理器执行下列操作：

* 将 mepc 恢复到 PC。（mepc 保存的是下一条指令的 PC，所以无需调整）
* 将 mstatus.MPIE 恢复到 mstatus.MIE。
* 从 mstatus.MPP 恢复发生中断之前的权限模式。

机器模式下拥有修改访问所有计时器中断相关寄存器的权限；超级用户模式下仅具有访问修改超级用户模式时钟计时器比较值寄存器（stimecmp）的权限；普通用户模式没有权限。

## UART中断

在Biriscv当中，只有一个外部中断源UART，UART中断通过外部中断引脚直接接入到CPU Core内部。具体UART实现方法不在这里赘述，可查询16550A资料，以下是Biriscv当中的UART中断处理流程。

### UART核发起中断请求

UART通过相应IRQ引脚向CPU核发送中断请求。

### CSR更改相应寄存器

mip.MSIP位被置1后，同时mie.MSIE为1，则开始处理中断。此时，处理器执行下列操作：

* 处理器执行完当前指令，保存下一条指令的 PC 到 mepc 中。
* 设置 mcause 的中断标记为 1，将中断编号写入 mcause，并更新 mtval 为 0。
* 将 mstatus 的中断使能位 MIE 保存到 MPIE 中，将 MIE 清零，禁止响应中断。
* 将发生中断之前的权限模式保存到 mstatus 的 MPP 中，切换到机器模式。
* （mtvec.Mode=0，直通中断）：PC 从 mtvec 处执行。通常，取回的指令是一条跳转指令，跳转至顶层处理函数。该函数通过分析 mcause 获取中断编号，并调用该编号对应的处理函数。
* mtvec.Mode=1，矢量中断）：PC 从 mtvec.Base + 4 \* 中断编号处执行。通常，取回的指令是一条跳转指令，跳转至相应中断的处理函数。

#### 中断处理程序执行

RV Linux Kernel接收到来自UART中断请求，进入IRQ处理函数并执行UART中断相关处理函数。（待完善）

#### 中断处理程序执行完毕

中断返回执行 mret 指令可以实现中断返回。此时，处理器执行下列操作：

* 将 mepc 恢复到 PC。（mepc 保存的是下一条指令的 PC，所以无需调整）
* 将 mstatus.MPIE 恢复到 mstatus.MIE。
* 从 mstatus.MPP 恢复发生中断之前的权限模式。

表 13.3 CLINT寄存器的存储器映射地址

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 地址 | 宽度 | 属性 | 名称 | 初始值 |
| 0x0200\_0000 | 4B | RW | msip0 | 0x00000000 |
| 0x0200\_0004 | 4B | RW | msip1 | 0x00000000 |
| Reserved | - | - | Reserved |  |
| 0x0200\_4000 | 4B | RW | mtimecmp0\_low | 0xffffffff |
| 0x0200\_4004 | 4B | RW | mtimecmp0\_high | 0xffffffff |
| 0x0200\_4008 | 4B | RW | mtimecmp1\_low | 0xffffffff |
| 0x0200\_400C | 4B | RW | mtimecmp1\_high | 0xffffffff |
| Reserved | - | - | Reserved |  |
| 0x0200\_BFF8 | 4B | RW | mtime\_low | 0x00000000 |
| 0x0200\_BFFC | 4B | RW | Mtime\_high | 0x00000000 |
| 0x0200\_C000 | 4B | RW | ssip0 | 0x00000000 |
| 0x0200\_C004 | 4B | RW | ssip1 | 0x00000000 |
| Reserved | - | - | Reserved |  |
| 0x0200\_D000 | 4B | RW | stimecmp0\_low | 0xffffffff |
| 0x0200\_D004 | 4B | RW | stimecmp0\_high | 0xffffffff |
| 0x0200\_D008 | 4B | RW | stimecmp1\_low | 0xffffffff |
| 0x0200\_D00C | 4B | RW | stimecmp1\_high | 0xffffffff |
| Reserved | - | - | Reserved |  |

表 13.4 UART寄存器的存储器映射地址

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 地址 | 宽度 | 属性 | 名称 | 初始值 |
| 0x9200\_0000 | 1B | RW | Divisor Latch Byte 1 | 0x00 |
| 0x9200\_0001 | 1B | RW | Divisor Latch Byte 2 | 0x00 |
| 0x9200\_0000 | 1B | R | Receiver Buffer | 0x00 |
| 0x9200\_0000 | 1B | W | Transmitter Holding Register | 0x00 |
| 0x9200\_0001 | 1B | RW | Interrupt Enable | 0x00 |
| 0x9200\_0002 | 1B | R | Interrupt Identification | 0xC1 |
| 0x9200\_0002 | 1B | W | FIFO Control | 0xC0 |
| 0x9200\_0003 | 1B | RW | Line Control Register | 0x03 |
| 0x9200\_0004 | 1B | W | Modem Control Register | 0x00 |
| 0x9200\_0005 | 1B | R | Line Status Register | 0x00 |
| 0x9200\_0006 | 1B | R | Modem Status Register | 0x00 |

## 端口列表

表 13.5 端口列表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 信号名 | I/O | 初始值 | 时钟域 | 功能描述 |
| timer\_irq\_o[1:0] | O |  |  | 时钟中断信号 |
| ipi\_o[1:0] | O |  |  | 核间中断信号 |
| Exc\_o | O |  |  | 外部中断信号 |

# Wishbone总线

## 时序

##### 附录1 信号列表

|  |  |  |
| --- | --- | --- |
| signal | Dri | Description |
| 分支预测单元信号 | | |
| branch\_info\_request\_i | in | 分支跳转请求信号 |
| branch\_info\_pc [31:0] | in | 分支跳转PC地址 |
| next\_pc\_f\_o [31:0] | out | 取指地址 |
| 取指单元 | | |
| next\_pc\_f\_i [31:0] | in | 取指地址 |
| branch\_request\_i | in | 分支跳转请求信号 |
| branch\_pc\_i [31:0] | in | 分支跳转PC地址 |
| icache\_inst\_i [63:0] | in | I-cache输入指令 |
| 译码单元 | | |
| fetch\_pc\_instr\_i [31:0] | in | PC地址 |
| fetch\_in\_instr\_i [63:0] | in | 指令 |
| branch\_request\_i | in | 分支跳转请求 |
| fetch\_out0\_instr\_o [31:0] | out | 发射1指令 |
| fetch\_out0\_pc\_o [31:0] | out | 发射1PC地址 |
| fetch\_out1\_instr\_o [31:0] | out | 发射2指令 |
| fetch\_out1\_pc\_o [31:0] | out | 发射2PC地址 |
| 指令缓存 | | |
| axi\_rdata\_i[31:0] | in | AXI总线数据 |
| req\_pc\_i[31:0] | in | 指令PC地址 |
| fetch\_out\_pc\_o[31:0] | out | 指令PC地址 |
| 发射单元 | | |
| clk\_i | in | 时钟 |
| rst\_i | in | 复位信号 |
| fetch0\_valid\_i | in | 译码单元FIFO valid信号，FIFO工作正常 |
| fetch0\_instr\_i [31:0] | in | 发射0指令 |
| fetch0\_pc\_i [31:0] | in | 发射0PC地址 |
| fetch0\_fault\_fetch\_i | in | AXI Error |
| fetch0\_fault\_page\_i | in | 特权等级错误 |
| fetch0\_instr\_exec\_i | in | Exec单元指令 |
| fetch0\_instr\_lsu\_i | in | LSU单元指令 |
| fetch0\_instr\_branch\_i | in | Branch指令 |
| fetch0\_instr\_mul\_i | in | Mul指令 |
| fetch0\_instr\_div\_i | in | Div指令 |
| fetch0\_instr\_csr\_i | in | CSR指令 |
| fetch0\_instr\_rd\_valid\_i | in | 指令有目标寄存器，且目标寄存器有效 |
| fetch0\_instr\_invalid\_i | in | 非法指令 |
| fetch1\_valid\_i | in | 译码单元FIFO valid信号，FIFO工作正常 |
| fetch1\_instr\_i [31:0] | in | 发射1指令 |
| fetch1\_pc\_i [31:0] | in | 发射1PC地址 |
| fetch1\_fault\_fetch\_i | in | AXI Error |
| fetch1\_fault\_page\_i | in | 特权等级错误 |
| fetch1\_instr\_exec\_i | in | Exec单元指令 |
| fetch1\_instr\_lsu\_i | in | LSU单元指令 |
| fetch1\_instr\_branch\_i | in | Branch指令 |
| fetch1\_instr\_mul\_i | in | Mul指令 |
| fetch1\_instr\_div\_i | in | Div指令 |
| fetch1\_instr\_csr\_i | in | CSR指令 |
| fetch1\_instr\_rd\_valid\_i | in | 指令有目标寄存器，且目标寄存器有效 |
| fetch1\_instr\_invalid\_i | in | 非法指令 |
| branch\_exec0\_request\_i | in | Branch指令（跳转或不跳转） |
| branch\_exec0\_is\_taken\_i | in | Branch指令跳转 |
| branch\_exec0\_is\_not\_taken\_i | in | Branch指令不跳转 |
| branch\_exec0\_is\_call\_i | in | 跳转指令 |
| alu\_p\_o[31:0] | in | ALU单元写回结果 |
| csr\_result\_e1\_value\_o[31:0] | in | CSR单元写回结果 |
| writeback\_value\_o[31:0] | in | MUL单元写回结果 |
| writeback\_value\_o[31:0] | in | LSU单元写回结果 |
| branch\_pc\_o[31:0] | out | 分支跳转地址 |
| branch\_request\_o | out | 分支跳转请求 |
| branch\_info\_pc\_o[31:0] | out | BPU更新PC跳转地址 |
| branch\_info\_request\_o | out | BPU更新分支跳转请求 |
| opcode0\_ra\_operand\_o[31:0] | out | ALU0源操作数1 |
| opcode0\_rb\_operand\_o[31:0] | out | ALU0源操作数2 |
| opcode1\_ra\_operand\_o[31:0] | out | ALU1源操作数1 |
| opcode1\_rb\_operand\_o[31:0] | out | ALU1源操作数2 |
| csr\_opcode\_ra\_operand\_o[31:0] | out | CSR源操作数1 |
| csr\_opcode\_rb\_operand\_o[31:0] | out | CSR源操作数2 |
| mul\_opcode\_ra\_operand\_o[31:0] | out | MUL源操作数1 |
| mul\_opcode\_rb\_operand\_o[31:0] | out | MUL源操作数2 |
| lsu\_opcode\_ra\_operand\_o[31:0] | out | LSU源操作数1 |
| lsu\_opcode\_rb\_operand\_o[31:0] | out | LSU源操作数2 |
| 执行单元 | | |
| alu\_a\_i[31:0] | in | ALU0源操作数1 |
| alu\_b\_i[31:0] | in | ALU0源操作数2 |
| opcode\_ra\_operand\_i[31:0] | in | CSR/MUL/LSU 源操作数1 |
| opcode\_rb\_operand\_i[31:0] | in | CSR/MUL/LSU 源操作数2 |
| mem\_data\_rd\_i[31:0] | in | 访存结果读取 |
| alu\_p\_o[31:0] | out | ALU写回结果 |
| csr\_result\_e1\_value\_o[31:0] | out | CSR写回结果 |
| writeback\_value\_o[31:0] | out | MUL写回结果 |
| writeback\_value\_o[31:0] | out | LSU写回结果 |
| mem\_addr\_o[31:0] | out | 访存地址 |
| mem\_data\_wr\_o[31:0] | out | 访存写入数据 |
| MMU信号 | | |
| fetch\_out\_inst\_i[63:0] | in | 输入I-MMU指令 |
| fetch\_in\_pc\_i[31:0] | in | 输入I-MMU地址 |
| req\_inst\_o[63:0] | out | I-MMU输出指令 |
| fetch\_out\_pc\_o[31:0] | out | I-MMU输出地址 |
| lsu\_in\_addr\_i[31:0] | in | 输入D-MMU地址 |
| lsu\_out\_data\_wr\_w[31:0] | in | 输入D-MMU的读数据 |
| lsu\_out\_data\_rd\_i[31:0] | in | 输入D-MMU的写数据 |
| lsu\_in\_data\_rd\_o[31:0] | out | D-MMU输出数据 |
| lsu\_out\_addr\_o[31:0] | out | D-MMU输出地址 |
| lsu\_out\_data\_wr\_o[31:0] | out | D-MMU输出写数据 |
| 指令缓存信号 | | |
| axi\_rdata\_i[31:0] | in | 输入总线指令 |
| req\_pc\_i[31:0] | in | 输入地址 |
| fetch\_out\_pc\_o[31:0] | out | I-cache输出指令 |
| 数据缓存信号 | | |
| mem\_addr\_i[31:0] | in | 输入地址 |
| mem\_data\_wr\_i[31:0] | in | 输入指令 |
| mem\_data\_rd\_o[31:0] | out | 输出指令 |

##### 附录2参考文献

Github下载地址: <https://github.com/ultraembedded/biriscv>

BootLoader下载地址: <https://github.com/ultraembedded/riscv-linux-boot>

Kernel image下载地址: <https://github.com/ultraembedded/riscv-linux-prebuilt>

##### 附录3 寄存器列表/属性

|  |  |
| --- | --- |
| Register Name | Attribute |
| User State | |
| x0/zero | Zero |
| x1/ra | Return Address |
| x2/sp | Stack Pointer |
| x3/gp | Global Pointer |
| x4/tp | Thread Pointer |
| x5/t0 | Temporary Register |
| x6/t1 | Temporary Register |
| x7/t2 | Temporary Register |
| x8/s0 | Saved Register/frame pointer |
| x9/s1 | Saved Register |
| x10/a0 | Function Arguments/return values |
| x11/a1 | Function Arguments/return values |
| x12/a2 | Function Arguments |
| x13/a3 | Function Arguments |
| x14/a4 | Function Arguments |
| x15/a5 | Function Arguments |
| x16/a6 | Function Arguments |
| x17/a7 | Function Arguments |
| x18/s2 | Saved Register |
| x19/s3 | Saved Register |
| x20/s4 | Saved Register |
| x21/s5 | Saved Register |
| x22/ss6 | Saved Register |
| x23/s7 | Saved Register |
| x24/s8 | Saved Register |
| x25/s8 | Saved Register |
| x26/s10 | Saved Register |
| x27/s11 | Saved Register |
| x28/t3 | Temporary Register |
| x29/t4 | Temporary Register |
| x30/t5 | Temporary Register |
| x31/t6 | Temporary Register |
| pc | Program Counter |
| Supervisor State | |
| sstatus | Supervisor status register |
| stvec | Supervisor Trap Vector Base Address Register |
| sie | Supervisor Interrupt-Enable Register |
| sip | Supervisor Interrupt-Pending Register |
| scounteren | Counter-Enable Register |
| sscratch | Supervisor Scratch Register |
| sepc | Supervisor Exception Program Counter |
| scause | Supervisor Cause Register |
| stval | Supervisor Trap Value Register |
| senvcfg | Supervisor Environment Configuration Register |
| satp | Supervisor Address Translation and Protection Register |
| Machine State | |
| misa | Machine ISA register |
| mvendorid | Machine Vendor ID register |
| marchid | Machine Architecture ID register |
| mimpid | Machine Implementation ID register |
| mhartid | Hart ID register |
| mstatus | Machine Status Register |
| mstatush | Machine Status Register |
| mtvec | Machine Trap-Vector Base-Address Register |
| medeleg | Machine Exception Delegation Register |
| mideleg | Machine Interrupt Delegation Register |
| mip | Machine Interrupt-Pending Register |
| mie | Machine Interrupt-Enable Register |
| mcycle | Machine clock cycle count register |
| minstret | Machine instruction retired count register |
| mcounteren | Machine Counter-Enable Register |
| mcountinhibit | Machine Counter-Inhibit register |
| mscratch | Machine Scratch Register |
| mepc | Machine Exception Program Counter register |
| mcause | Machine Cause Register |
| mtval | Machine Trap Value Register |
| mconfigptr | Machine Configuration Pointer Register |
| mseccfg | Machine Configuration Pointer Register |
| mtime | Machine time register  (memory -mapped control register) |
| mtimecmp | Machine time compare register  (memory -mapped control register) |