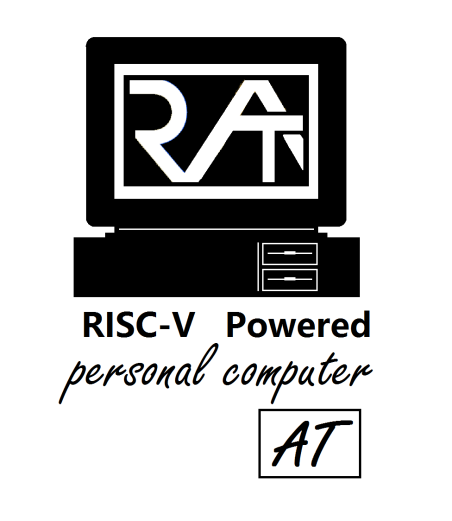
****

**PRV464SXR处理器编程手册**

编号：20200713

**目录**

**前言**

**1、概览**

1.1、PRV464SXR流水线结构

1.2、PRV464SXR支持的通用寄存器、CSR列表

1.3、PRV464SXR支持的指令集列表

**2、RV64I指令介绍**

2.1、RV64I指令列表

2.2、RV64I指令功能

2.2.1、LUI，AUIPC指令

2.2.2、使用立即数的整数运算指令

ADDI、SLTI、SLTIU、ANDI、ORI、XORI、SLLI、SRLI、SRAI

2.2.3、寄存器-寄存器间的整数运算指令

ADD、SUB、SLT、SLTIU、AND、OR、XOR、SLL、SRL、SRA

2.2.4、分支和跳转指令

JAL，JALR，BEQ，BNE，BLT、BLTU，BGE、BGEU

2.2.5、整数存取指令

SB、SH、SW、SD，LB，LBU，LH，LHU，LW，LWU，LD

2.2.6、RV64I 32位操作指令

ADDIW，SLLIW，SRLIW，SRAIW，ADDW，SUBW，SLLW，SRLW，SRAW

2.2.7、存储器屏障指令 FENCE、FENCE.I

2.2.8、CSR指令（详见4.1：系统指令）

**3、RV64A指令介绍**

3.1、RV64A指令列表

3.2、RV64A指令功能

3.2.1、原子交换指令（AMOSWAPW，AMOSWAPD）

3.2.2、原子加指令（AMOADD.W， AMOADD.D）

3.2.3、 原子与指令（AMOAND.W， AMOAND.D）

3.2.4、原子或指令（AMOOR.W，AMOOR.D

3.2.5、原子异或指令（AMOXOR.W， AMOXOR.D）

3.2.6、带符号的原子取大指令（AMOMAX.W， AMOMAX.D）

3.2.7、带符号的原子取小指令（AMOMIN.W， AMOMIN.D

3.2.8、不带符号的原子取大指令（AMOMAX.W， AMOMAX.D）

3.2.9、不带符号的原子取小指令（AMOMIN.W， AMOMIN.D）

3.2.10、互斥写和互斥读指令（LR.W, LR.D, SC.W, SC.D）

**4、CSR寄存器和系统指令**

4.1、系统指令

4.2、CSR寄存器介绍

4.2.1、M模式CSR

4.2.1.1、MISA寄存器

4.2.1.2、MVENDORID寄存器

4.2.1.3、MARCHID寄存器

4.2.1.4、MIMPID寄存器

4.2.1.5、MHARTID寄存器

4.2.1.6、MSTATUS寄存器

4.2.1.6.1、MSTATUS寄存器的虚拟化管理

4.2.1.6.2、MSTATUS寄存器的内存权限管理

4.2.1.6.3、MSTATUS寄存器的全局/特权中断

4.2.1.7、MTVEC寄存器

4.2.1.8、机器模式下中断/异常委托（MEDELEG和MIDELEG）

4.2.1.9、机器模式中断寄存器（MIE&MIP）

4.2.1.10、机器模式定时器MTIME &MTIMECMP

4.2.1.11、硬件性能监视器 MCYCLE&MINSTRET

4.2.1.12、计数器使能寄存器MCOUNTEREN&SCOUNTEREN

4.2.1.13、计数器停止寄存器（MCOUNTERINHIBIT）

4.2.1.14、机器模式临时寄存器（MSCRATCH）

4.2.1.15、机器模式异常程序计数器（MEPC）

4.2.1.16、机器模式异常原因寄存器（MCAUSE）

4.2.1.17、机器模式异常值寄存器（MTVAL）

4.2.2、S模式CSR

4.2.2.1、S模式状态寄存器（SSTATUS）

4.2.2.2、S模式中断向量寄存器（STVEC）

4.2.2.3、S模式中断控制寄存器（SIP&SIE）

4.2.2.4、S模式计数器使能寄存器（SCOUNTEREN）

4.2.2.5、S模式临时寄存器（SSCRATCH）

4.2.2.6、S模式异常程序指针寄存器（SEPC）

4.2.2.7、S模式异常原因寄存器（SCAUSE）

4.2.2.8、S模式异常值寄存器（STVAL）

4.2.2.9、S模式地址转换和保护寄存器（SATP）

4.2.3、U模式CSR

**5、异常和特权**

5.1、RISCV中断/异常的管理

5.2、RISCV中断/异常的处理

5.3、RISCV分页和特权

5.3.1、基于Sv39的虚拟内存

5.3.2、RISCV特权

**6、致谢**

**附录A、RISC-V指令编码（RV64）**

A1、RV64I指令编码。

A2、RV64A指令编码。

**附录B、PRV464SXR程序优化说明。**

B1、寄存器使用间隔

B2、TLB和L1缓存

B3、指令运行时间

**附录C、P46459平台中断控制器（PLIC）**

**附录F、S拓展指令**

**前言：**

在目前计算机教学中，还有很多学校使用的是固定逻辑器件如74xx系列逻辑电路培训学生数字逻辑能力，这种方式下学生所学到的本领早已不适合当前高速发展的数字逻辑行业对人才的需求。随着近几年可编程逻辑器件，尤其是FPGA的高速发展：10K-100K门级的FPGA价格进入到大部分人可以接受的价格，同时FPGA性能得到了长足的发展，在单片FPGA中实现一个功能完备，性能较好的系统成为可能。同时，部分学校开始考虑使用基于FPGA平台的计算机原理/数字逻辑教学平台来适应新时代对学生能力的需求。

RISC-V，或者简称RV，是美国加州伯克利大学研发的第五代精简指令系统（RISC：精简指令系统计算机、 V：第五代），它同时具备开源、简洁、可定制化的特点，目前采用RISC-V指令集处理器的系统其市场占有率自2015年起逐步提高，已经在嵌入式市场占据了一部分市场份额，。

近十几年来，我国计算机专业学生学习的处理器指令体系主要是X86-16（诞生于1978年），8051（诞生于1975年），小部分学习MIPS-32和arm32指令集（诞生于上个世纪80年代），这些指令集要么过于老旧，严重落后于时代（如X86-16），要么已经在市场竞争中逐渐走向衰落（如MIPS，MIPS指令集因为MIPS公司的破产而变得群魔乱舞，各家指令系统之间都有区别，指令集碎片化严重），或者是因为在长期的发展中变得过于复杂而难以学习，如ARM公司的ARM系列指令集，其指令手册达到了2000页，X86-32/64含拓展指令的指令文档更是达到了6000页的恐怖程度，这对于教学来说是非常困难的。 RISC-V指令集充分的吸取了上述指令集的特点，根据数十年来计算机工业发展的实践教训，大幅缩减了指令体系，采用模块化指令拓展，其指令文档仅有219页，特权架构文档仅有79页（2019608版）。精简并不代表功能精简，相反，得益于设计年份很晚，RISC-V在设计之初就考虑了32位，64位乃至128位的指令集，而不像之前的指令集设计的时候压根就没考虑更高的位数。目前，RISC-V处理器已经获得了Open-SBI库的支持，大大方便了通用操作系统的开发，并已经建立了具备引导完整Linux操作系统的能力的UEFI。

本文档是一种采用RISC-V64I，A指令拓展的RISC-V处理器软核（简称RV64IA）的编程手册，这个处理器是作者曾经设计的采用状态机设计的RV32IA处理器（PRV332）的一个进阶版本，为了能与之前设计的RV32处理器形成高低搭配。读者可以在阅读此文档的过程中了解到在RV64架构上编程所需的绝大部分知识，尤其是在特权指令方面，在国内鲜有详细的RISC-V特权架构的文档，希望读者们能在阅读的过程中收获到不一样的知识！

编者

2020年3月15日

**第一章：概览**

PRV464SX-R处理器是PRV4系列处理器的第二个型号，主要面向入门级FPGA（容量在10K-20KLUT4），为了尽可能地减少核心大小和逻辑复杂度，采用纯手写Verilog代码进行构建，并且只提供了对I和A拓展指令的支持。经过测试，该核心在CycloneIV平台上占用14.5KLUT4，在安路EG4S20平台上占用12KLUT5。

特点：

\*高效简洁的四级流水线

\*8KByte+8KByte8-WayL1高速缓存

\*4+4entryTLB

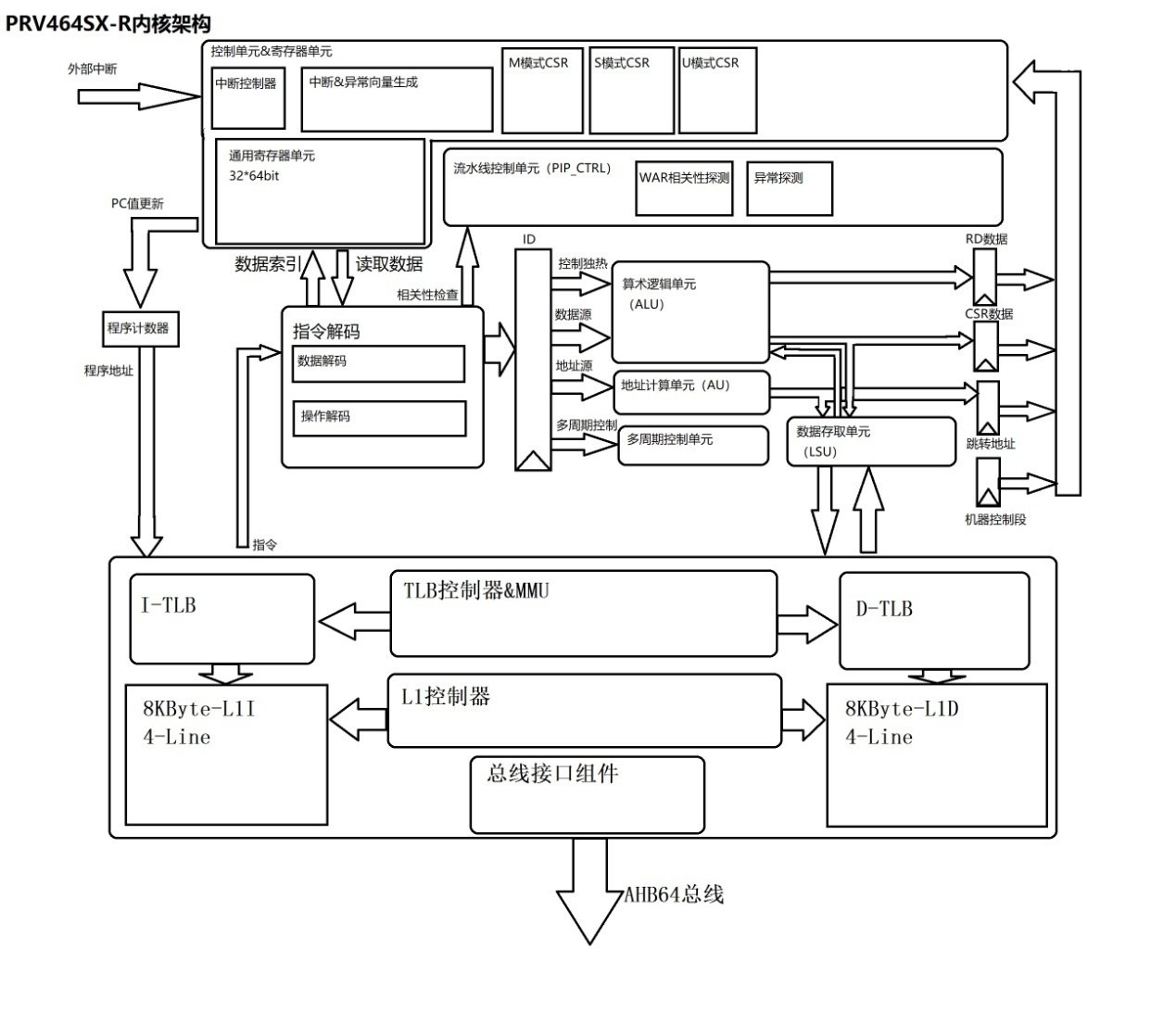
\*禁用执行位技术

\*完全可隔离的代码和数据

\*三层权限，支持Sv39分页方案，最大寻址空间达2^55

**1.1、PRV464SXR流水线结构**

PRV464SXR使用四级流水线，分别是：IF（取指令），ID（指令解码），EX（执行），WB（写回），架构如下图所示：



并且，我们给PRV464SX-R处理器配备了8K+8K的L1缓存，4+4entryTLB，采用写透策略，一次缓存一个对齐的指令页面和一个对齐的数据页面，在少量增加系统复杂度的情况下获得了较好的执行能力。大部分程序的平均CPI为2-3，优化良好的程序可以跑出CPI=1。

**1.2、PRV464SXR支持的通用寄存器、CSR列表**

PRV464支持RV64IA指令集，根据RISCVABI（应用程序二进制接口）标准，一共支持32个通用寄存器：

|  |  |  |  |
| --- | --- | --- | --- |
| 寄存器名称 | ABI名称 | 解释 |  |
| X0 | Zero | 常值0 |  |
| X1 | RA | 返回地址 |  |
| X2 | SP | 堆栈指针 |  |
| X3 | GP | 全局指针 |  |
| X4 | TP | 线指针 |  |
| X5 | T0 | 临时值 |  |
| X6-T7 | T1-T2 | 临时值 |  |
| X8 | S0/FP | 保存寄存器/帧指针 |  |
| X9 | S1 | 保存寄存器 |  |
| X10-X11 | A0-A1 |  |  |
| X12-X17 | A2-A7 |  |  |
| X18-X27 | S2-S11 |  |  |
| X28-X31 | T3-T6 |  |  |

CSR寄存器列表：（关于CSR寄存器的详解可以参考第四章）

|  |  |  |  |
| --- | --- | --- | --- |
| 地址 | 权限 | 名称 |  |
| 0xC00 | URO | CYCLE |  |
| 0xC01 | URO | TIME |  |
| 0xC02 | URO | INSTRET |  |
| 0xC03 | URO | HPMCOUNTER3 |  |
| 0xC04 | URO | HPMCOUNTER4 |  |
| 0x100 | SRW | SSTATUS |  |
| 0x102 | SRW | SEDELEG |  |
| 0x103 | SRW | SIDELEG |  |
| 0x104 | SRW | SIE |  |
| 0x105 | SRW | STVEC |  |
| 0x106 | SRW | SCOUNTEREN |  |
| 0x140 | SRW | SSCRATCH |  |
| 0x141 | SRW | SEPC |  |
| 0x142 | SRW | SCAUSE |  |
| 0x143 | SRW | STVAL |  |
| 0x144 | SRW | SIP |  |
| 0x180 | SRW | SATP |  |
| 0xF11 | MRO | MVENDORID |  |
| 0xF12 | MRO | MARCHID |  |
| 0xF13 | MRO | MIMPID |  |
| 0xF14 | MRO | MHARTID |  |
| 0x300 | MRW | MSTATUS |  |
| 0x301 | MRW | MISA |  |
| 0x302 | MRW | MEDELEG |  |
| 0x303 | MRW | MIDELEG |  |
| 0x304 | MRW | MIE |  |
| 0x305 | MRW | MTVEC |  |
| 0x306 | MRW | SCOUNTEREN |  |
| 0x340 | MRW | MSCRATCH |  |
| 0x341 | MRW | MEPC |  |
| 0x342 | MRW | MCAUSE |  |
| 0x343 | MRW | MTVAL |  |
| 0x344 | MRW | MIP |  |
| 0xB00 | MRW | MCYCLE |  |
| 0xB02 | MRW | MINSTRET |  |
| 0x320 | MRW | MCOUNTINHIBIT |  |

\*URO： UserReadOnly， 用户模式只读

\*SRO： SupervisorReadOnly，监督者模式只读

\*SRW： SupervisorReadWrite，监督者模式可读可写

\*MRO： MachineReadOnly，机器模式只读

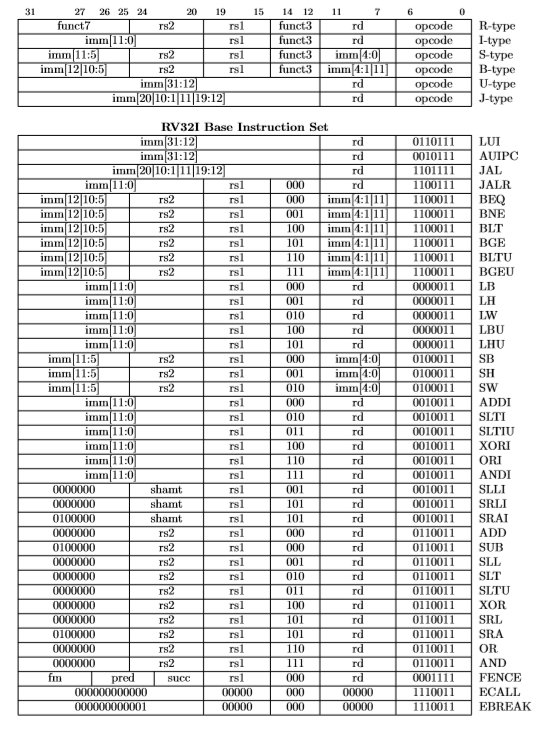
\*MRW： MachineReadWrite，机器模式可读可写

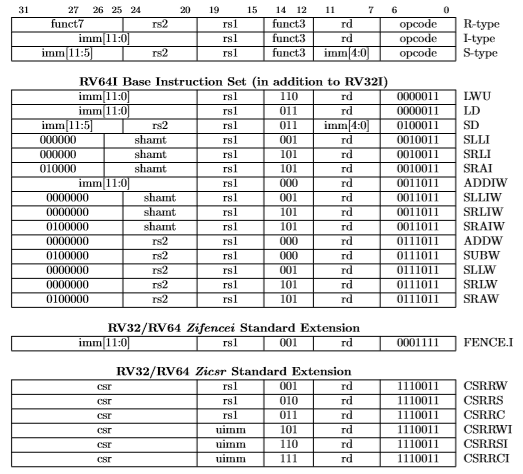
**1.3、PRV464SXR支持的指令列表**

PRV464SXR处理器支持RV64I和A拓展，为了进一步的优化处理器的占用，暂时不使用其他的拓展指令。

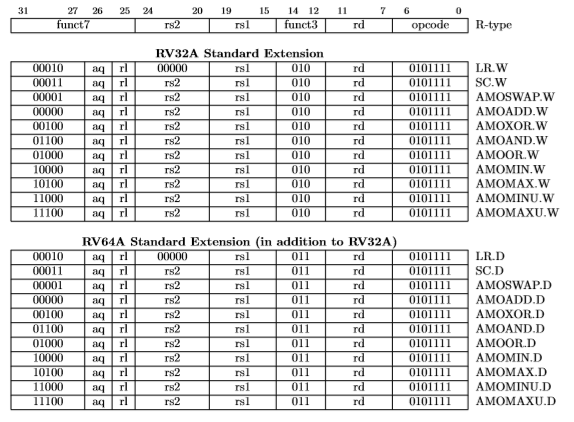
**1.3.1、RV64I指令集**

RV64I指令集是RV32I指令集的一个拓展，在RV64I中，所有的寄存器被拓展到64位，因此，增加了一些32位操作指令来操作32位值；也增加了64位内存读写指令。



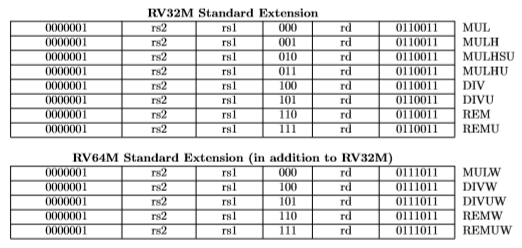


**1.3.2、RV64A指令集**



**1.3.3、RV64M指令集**

RV64M拓展指令在之后可能被用到。



**第二章：RV64I指令介绍**

**2.1、RV64I指令列表**

RV64I指令集是RV32I指令集的一个拓展，得益于设计年份较晚，RISCV指令集在设计时便考虑了64位指令集，RV64相当于是RV32将寄存器宽度拉长成64位，并且添加了几个32位的操作指令，这些32位指令用W结尾，如ADDIWRD， RS1， imm[11:0]。

由于RV32平台上几乎没有什么应用被开发出来，目前采用RV32指令架构的大多都是单片机，使用虚拟内存系统的处理器大多都选用的RV64架构，因此在RV64上做RV32的兼容是毫无意义的，故RISCV官方并没有要求RV64需要兼容RV32的程序，这给处理器设计和程序设计带来了极大的方便。

\*编者注：在RV的指令编码中，将结果寄存器RD和源寄存器RS1，RS2严格分开，不像其他架构一样目的寄存器和源寄存器复用，如德州仪器MSP430内核，执行ADDR5，R6指令时候，是将R5和R6相加而后覆盖写回R5，其他处理器如ARM公司的处理器也是这样设计的，对于寄存器更少的处理器，如8051/8086内核来说，它们压根就没有通用整数寄存器的概念，对于8051核心，只有一个ACC（累加器），意思是所有的结果都会被覆盖的写入这个寄存器中，而对于8086，勉强算得上通用寄存器的只有Ax，Bx，Cx，Dx四个，而除开Ax外其余三个还有自己的特殊功能。

**2.2、RV64I指令介绍**

**2.2.1、LUI，AUIPC指令**

（1）汇编指令格式：

LUI RD， IMM[19:0]

AUIPCRD， IMM[19:0]

（2）指令行为：

LUI指令将一个20位的立即数左移12位，低12位补0,，高32位进行符号位拓展合成一个64位立即数，写入寄存器RD中。

AUIPC指令同样将一个20位立即数左移12位，低12位补0,，高32位进行符号位拓展合成一个64位立即数，与当前指令PC值相加后写入寄存器RD中。

**2.2.2、使用立即数的整数运算指令**

**ADDI、SLTI、SLTIU、ANDI、ORI、XORI、SLLI、SRLI、SRAI**

1. 汇编指令格式

ADDIRD， RS1， IMM[11:0]

SLTI RD, RS 1, IMM[11:0]

SLTIU RD, RS1, IMM[11:0]

……

1. 指令详解

ADDI指令将12位立即数进行符号位拓展到64位，与RS1寄存器中的值相加，所得结果写回RD寄存器。

SLTI指令将寄存器RS1的值和和12位立即数（进行符号位拓展），当作有符号数进行比较，如果RS1小于12位立即数，写回1到RD寄存器，其他情况则写0。

SLTIU指令将寄存器RS1的值和12位立即数（仍然进行符号位拓展），当作无符号数比较，如果RS1小于12位立即数，则写1到RD寄存器。

ANDI，ORI，XORI指令将RS1的值和12位立即数（进行符号位拓展）进行与/或/异或操作，然后将结果写回RD寄存器。

SLL指令将RS1中的数进行左移，移位位数使用6位立即数编码，低位补充0，结果写回RD寄存器。

SRL指令将RS1中的数进行右移，移位位数使用6位立即数编码，高位补充符号位，结果写回RD寄存器。

SRA指令将RS1中的数进行右移，移位位数使用6位立即数编码，高位补充0，结果写回RD寄存器。

\*编者注：在RISCV的立即数定义里，所有立即数都进行符号位拓展到64位，这样做的目的是简化译码。

**2.2.3、寄存器-寄存器间的整数运算指令**

**ADD、SUB、SLT、SLTIU、AND、OR、XOR、SLL、SRL、SRA**

除了支持使用立即数的整数运算指令，RISCV还支持寄存器和寄存器间的运算指令，这些指令的特点就是将12位立即数换成了RS2中的值。显而易见的，这个时候不需要进行符号位拓展了。

**2.2.4、分支和跳转指令**

**JAL，JALR，BEQ，BNE，BLT、BLTU，BGE、BGEU**

1. 汇编指令格式

JAL RD， LABLE

JALR RD， RS1， IMM

BEQ RS1， RS2， LABLE

BNE RS1， RS2， LABLE

BLT RS1， RS2， LABLE

BLTU RS1， RS2， LABLE

BGE RS1， RS2， LABLE

BGEU RS1， RS2， LABLE

1. 指令详解

JAL指令使用20位立即数作为偏移量，将其左移一位，高位补入符号位，和当前的PC值相加计算出跳转地址。同时将当前指令的PC值+4写入寄存器RD中。

JALR指令使用12位立即数作为偏移量（高位进行符号位拓展），与RS1寄存器中的值相加生成最终的跳转地址，JALR指令将下一条指令的PC（当前指令PC+4）写入寄存器RD中。

BEQ指令将RS1和RS2寄存器的值作比较，如果RS1=RS2的值，则跳转到12位立即数左移一位，高位补入符号位，与当前PC相加的地址。

BNE指令将RS1和RS2寄存器的值作比较，如果RS1≠RS2的值，则跳转到12位立即数左移一位，高位补入符号位，与当前PC相加的地址。

BLT指令将RS1和RS2寄存器的值当作有符号数作比较，如果RS1<RS2的值，则跳转到12位立即数左移一位，高位补入符号位，与当前PC相加的地址。

BLTU指令将RS1和RS2寄存器的值当作无符号数作比较，如果RS1<RS2的值，则跳转到12位立即数左移一位，高位补入符号位，与当前PC相加的地址。

BGE指令将RS1和RS2寄存器的值当作有符号数作比较，如果RS1>RS2的值，则跳转到12位立即数左移一位，高位补入符号位，与当前PC相加的地址。

BGEU指令将RS1和RS2寄存器的值当作无符号数作比较，如果RS1>RS2的值，则跳转到12位立即数左移一位，高位补入符号位，与当前PC相加的地址。

\*编者注：汇编器会自动的根据Lable计算偏移地址然后填入Bxx指令的imm编码中。

**2.2.5、整数存取指令**

**SB、SH、SW、SD，LB，LBU，LH，LHU，LW，LWU，LD**

RISCV的Store指令固定的使用RS1的数据与指令编码中12位偏移量OFF（高位补入符号位）相加作为地址，把RS2的数据存到内存中。load指令固定的使用RS1的数据与指令编码中12位偏移量（高位补入符号位）相加作为地址，把地址所指示的数据存到内存中。

（1）、指令格式

SB RS2， OFF[11:0](RS1)

SH RS2， OFF[11:0](RS1)

SW RS2， OFF[11:0](RS1)

SD RS2， OFF[11:0](RS1)

LB RD， OFF[11:0](RS1)

LBU RD, OFF[11:0](RS1)

LH RD， OFF[11:0](RS1)

LHU RD, OFF[11:0](RS1)

LW RD， OFF[11:0](RS1)

LWU RD, OFF[11:0](RS1)

LD RD， OFF[11:0](RS1)

1. 指令详解

LB指令从内存中读取一个8位数据，高位进行符号位拓展到64位后写回寄存器RD中。

LBU指令从内存中读取一个8位数据，高位补0后写回寄存器RD中。

LH指令从内存中读取一个16位数据，高位进行符号位拓展到64位后写回寄存器RD中。

LHU指令从内存中读取一个16位数据，高位补0后写回寄存器RD中。

LW指令从内存中读取一个32位数据，高位进行符号位拓展到64位后写回寄存器RD中。

LWU指令从内存中读取一个32位数据，高位补0后写回寄存器RD中。

LD指令从内存中读取一个64位数据，写回寄存器RD中。

SB指令将RS2中低8位数据存回内存中。

SH指令将RS2中低16位数据存回内存中。

SW指令将RS2中低32位数据存回内存中。

SD指令将RS2中的64位数据存回内存中。

\*编者注：在很多早期开发的处理器中，定义了非常多的寻址方式，以X86系统为例，有的可以用指令编码的立即地址寻址（有的处理器系统中也叫直接寻址），有的可以用当前PC值加上指令编码的PC值寻址（有的叫做相对PC偏移），有的可以使用寄存器中的值作为地址进行寻址（被叫做间接寻址），如此繁文缛节的寻址方式不但给程序员编程带来了压力，也给处理器设计带来了一些困难。早期的计算机因为寄存器位宽太低，单个寄存器中不能存放整个地址，因此早期的计算机通常会选择将两个或多个寄存器进行拼接相加来得到地址，如X86系统中的段式管理，就是使用两个16位的寄存器进行拼接得到20位地址。

**2.2.6、RV64I 32位操作指令**

**ADDIW，SLLIW，SRLIW，SRAIW，ADDW，SUBW，SLLW，SRLW，SRAW**

这些指令的操作同它的64位同名指令一样，只是将指令的操作宽度限制在了32位，如ADDIW，只是将RS1和立即数的求和的值取低32位，然后高32位进行符号位拓展并写回结果寄存器RD中。

同样的，ADDW和SUBW也对RS1和RS2的值进行操作后，忽略高32位，保留低32位并进行符号位拓展到64位之后写回结果寄存器RD中。

移位指令只操作低32位，高32位不操作并将结果写回结果寄存器RD中。

\*编者注：我不知道为什么RV64要添加32位操作指令，建议程序员在程序中不要使用这些指令。

**2.2.7、存储器屏障指令 FENCE、FENCE.I**

（1）指令格式

FENCE

FENCE.I

（2）指令详解

存储器屏障指令的作用是让屏障指令之前执行的内存访问操作对存储器屏障指令之后执行的内存操作可见，比如前面程序中向内存地址A0中存入了一个数值D，后面的程序要去A0中读这个值，如果程序不执行fence指令，后面的程序代码有可能无法读到D，如果执行了fence指令，那么后来的程序代码是一定可以读到D的。

本处理器的fence和fence.i用于设置不同的存储器屏障，如果程序修改了指令，那么务必需要执行一条fence.i指令以确保指令被正确的写入了。

\*编者注：关于处理器自编程的概念，即在程序运行的过程中修改后面还没有被执行的代码，早期的处理器如6502，Z80上运行的程序广泛的运用自编程的能力来在程序运行过程中动态修改没有被执行的代码，从而达到降低程序大小的目的。在流水线结构的处理器出现之后，动态修改没有被执行的代码却变得“有风险”，因为前面指令可能访问内存还没有被执行，处理器流水线已经将后面N条指令取进来了，这样就无法达到修改内存中没执行代码来改变运行结果的目的。在现在的处理器编程中，如果修改了代码（不管这些代码有没有被执行），在再次运行这些代码之前，程序员都需要执行一条FENCE.I指令来保证之前访问内存被完全的执行了。

**第三章：RV64A指令介绍**

**3.1、RV64A指令编码**

**3.2、RV64A指令功能**

在现代多核处理器系统中，如何解决多核之间进行数据同步的问题？假设有两个核心C0和C1需要同步数据，他们约定了一片内存地址来进行数据交换，C0将数据写好之后通知C1来取。为了达到这个目的，需要在内存中设置一个标志位来表示当前内存区域的状态。传统的Load和Store指令因为其执行顺序可能被硬件所更改，而不能达到这个目的，因此引入了AMO，即原子指令。

**3.2.1、原子交换指令（AMOSWAP.W，AMOSWAP.D）**

（1）指令格式

AMOSWAP.W RD, RS2, (RS1)

AMOSWAP.D RD, RS2, (RS1)

1. 指令描述

AMOSWAP指令从RS1作为地址所指的内存中读取一个值，写回RD寄存器中，并将RS2的值写回RS1作为地址所指的内存中。

结尾.W和.D指示的这条指令的操作位宽是32位还是64位，如果是32位，那么就会把RS2中的低32位数据进行操作，高32位忽略，以下同理。

**3.2.2、原子加指令（AMOADD.W， AMOADD.D）**

（1）指令格式

AMOADD.W RD, RS2, (RS1)

AMOADD.D RD, RS2, (RS1)

（2）指令描述

AMOADD指令从RS1作为地址所指的内存中读取一个值，写回RD寄存器中，并将RS2和刚才读取的值求和并写回RS1作为地址所指的内存中。

**3.2.3、 原子与指令（AMOAND.W， AMOAND.D）**

（1）指令格式

AMOAND.W RD, RS2, (RS1)

AMOAND.D RD, RS2, (RS1)

（2）指令描述

AMOAND指令从RS1作为地址所指的内存中读取一个值，写回RD寄存器中，并将RS2和刚才读取的值进行位与操作并写回RS1作为地址所指的内存中。

**3.2.4、原子或指令（AMOOR.W，AMOOR.D**

（1）指令格式

AMOOR.W RD, RS2, (RS1)

AMOOR.D RD, RS2, (RS1)

（2）指令描述

AMOOR指令从RS1作为地址所指的内存中读取一个值，写回RD寄存器中，并将RS2和刚才读取的值进行或操作并写回RS1作为地址所指的内存中。

**3.2.5、原子异或指令（AMOXOR.W， AMOXOR.D）**

（1）指令格式

AMOXOR.W RD, RS2, (RS1)

AMOXOR.D RD, RS2, (RS1)

（2）指令描述

AMOXOR指令从RS1作为地址所指的内存中读取一个值，写回RD寄存器中，并将RS2和刚才读取的值进行异或操作并写回RS1作为地址所指的内存中。

**3.2.6、带符号的原子取大指令（AMOMAX.W， AMOMAX.D）**

（1）指令格式

AMOMAX.W RD, RS2, (RS1)

AMOMAX.D RD, RS2, (RS1)

（2）指令描述

AMOMAX指令从RS1作为地址所指的内存中读取一个值，写回RD寄存器中（如果是32位的，就先进行符号位拓展到64位之后写回RD），并将RS2和刚才读取的值（当作有符号数）比较大小，选出其中较大的并写回RS1作为地址所指的内存中。

如果.W结尾的AMOMAX指令，则将读入的32位数据进行符号位拓展之后和RS2的值作比较，RS2和内存中读入的值都作为有符号数参与比较。

**3.2.7、带符号的原子取小指令（AMOMIN.W， AMOMAX.D）**

（1）指令格式

AMOMIN.W RD, RS2, (RS1)

AMOMIN.D RD, RS2, (RS1)

（2）指令描述

AMOMIN指令从RS1作为地址所指的内存中读取一个值，写回RD寄存器中（如果是32位的，就先进行符号位拓展到64位之后写回RD），并将RS2和刚才读取的值（当作有符号数）比较大小，选出其中较小的并写回RS1作为地址所指的内存中。

如果.W结尾的AMOMIN指令，则将读入的32位数据进行符号位拓展之后和RS2的值作比较，RS2和内存中读入的值都作为有符号数参与比较。

**3.2.8、不带符号的原子取大指令（AMOMAX.W， AMOMAX.D）**

（1）指令格式

AMOMAXU.W RD, RS2, (RS1)

AMOMAXU.D RD, RS2, (RS1)

（2）指令描述

AMOMAXU指令从RS1作为地址所指的内存中读取一个值，写回RD寄存器中（如果是32位的，就先进行符号位拓展到64位之后写回RD），并将RS2和刚才读取的值（当作有符号数）比较大小，选出其中较大的并写回RS1作为地址所指的内存中。

如果.W结尾的AMOMAXU指令，则将读入的32位数据进行符号位拓展之后和RS2的值作比较，RS2和内存中读入的值都作为无符号数参与比较。

**3.2.9、不带符号的原子取小指令（AMOMIN.W， AMOMIN.D）**

（1）指令格式

AMOMINU.W RD, RS2, (RS1)

AMOMINU.D RD, RS2, (RS1)

（2）指令描述

AMOMINU指令从RS1作为地址所指的内存中读取一个值，写回RD寄存器中（如果是32位的，就先进行符号位拓展到64位之后写回RD），并将RS2和刚才读取的值（当作有符号数）比较大小，选出其中较大的并写回RS1作为地址所指的内存中。

如果.W结尾的AMOMINU指令，则将读入的32位数据进行符号位拓展之后和RS2的值作比较，RS2和内存中读入的值都作为有符号数参与比较。

**3.2.10、互斥写和互斥读指令（LR.W, LR.D, SC.W, SC.D）**

（1）指令格式

LR.W RD, (RS1)

LR.D RD, (RS1)

SC.W RS2, (RS1)

SC.D RS2, (RS1)

（2）指令描述

LR指令从RS1寄存器指示的内存地址读入一个数到RD寄存器中，如果是32位，则进行符号位拓展之后写入RD寄存器。

SC指令把RS2的值存到RS1寄存器指示的内存中，如果访问失败，那么RD寄存器将会被写非0值，如果执行成功，那么RD寄存器将会写0值。

编者注： LR/SC指令主要在多核系统中使用，在本机的实现中因为完全没有考虑到多核的操作，故AMO指令所具备的一些特性：如释放一致性的模型，本机不支持，LR/SC指令表现的行为和常规Load/Store指令表现相同。

**第四章: CSR寄存器和系统指令**

**4.1、系统指令**

前文中已经提到，RISCV架构中将指令编码的低7位作为opcode，其中，opcode为7'b1110011的为“SYSTEM”，即系统指令。

本机支持的SYSTEM指令有：ECALL（环境调用），EBREAK（断点），CSRRW（CSR写），CSRRS（CSR置位），CSRRC（CSR清零），CSRRWI（CSR写，立即数），CSRRSI（CSR置位，立即数），CSRRCI（CSR清零，立即数），MRET（机器模式返回），SRET（监督者模式返回），SFENCE.VMA（虚拟内存fence指令），WFI（等待中断）

**ECALL**： 环境调用，当执行该条指令时，将会引起环境调用异常，详见5.1节对异常和中断的说明。

**EBREAK**：环境断点，当执行这条指令时，会引起环境断点异常，详见5.1节对异常和中断的说明。

**CSRRW**：(指令格式 csrrwrd, csr, rs1），即将CSR索引的CSR寄存器读出，写回RD中，并将RS1读出的值写回CSR中。

**CSRRS**：（指令格式 csrrsrd, csr, rs1）,将CSR索引的CSR寄存器读出，写回RD，并将RS1与读出的CSR进行或操作，结果写回CSR中。此操作对外表现为置位。

**CSRRC**：（指令格式 csrrcrd, csr, rs1），将CSR索引的CSR寄存器读出，写回RD，并将RS1取反，与读出的CSR进行与操作，结果写回CSR中/对外表现为RS1中为1的位让CSR中对应位被清零了。

**CSRRWI**：(指令格式 csrrwird, csr, imm[4:0]）,将CSR索引的CSR寄存器读出，写回RD寄存器。五位立即数（imm[4:0],高位用0拓展），写入CSR。

**CSRRSI**：同CSRRSI，只是把RS1的数换成了imm[4:0]，高位补0拓展。

**CSRRCI**：同CSRRCI，只是把RS1的数换成了imm[4:0]，高位补0拓展。

\*注: CSRRSI CSRRWI CSRRCI指令的立即数都是高位补0拓展，而不是像其他指令一样高位进行符号位拓展。

**MRET**：机器模式返回，用于运行在机器模式（M）下异常和中断的返回（详见第五章关于中断和异常的描述），这条指令只能在机器工作在M模式的时候被执行，如果在其他权限下被执行，将会引起非法指令异常。

**SRET**：监督者模式返回，用于运行在监督者模式（S）下异常和中断的返回，这条指令只能在机器工作在S 模式的时候被执行，如果在其他权限被执行，将会引起非法指令异常。

**SFENCE.VMA：**虚拟内存屏障，用来通知TLB刷新。当程序更改了页表之后，如果需要让页表在下一次访问的时候被成功的运用了，需要执行这条指令。例如，如果操作系统更改了页表中的一个PTE（页表项），当前TLB中存放的是之前的映射，需要执行一个sfence.vma指令来迫使TLB清空。

\*注: M模式可以通过设置MSTATUS寄存器中的TSR（Trap Superviser Return）位来让S模式执行SRET指令的时候产生一个非法指令异常，这样可以让M模式接管S模式。

本机并没有实现N拓展，故没有URET（用户模式异常返回），在大部分的处理器中，都没有允许在用户模式直接响应中断，除了X86系列的处理器可以直接在Ring3，即用户层响应中断。在用户层响应中断和异常的目的是为了让中断响应有更好的实时性，用户态响应中断对实时操作系统来说非常重要，但是对于linux这种分时操作系统来说，用户态响应中断显得不是那么必要。

**4.2、CSR寄存器介绍**

本机支持的CSR寄存器一览表如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 地址 | 权限 | 名称 |  |
| 0xC00 | URO | CYCLE |  |
| 0xC01 | URO | TIME |  |
| 0xC02 | URO | INSTRET |  |
| 0xC03 | URO | HPMCOUNTER3 |  |
| 0xC04 | URO | HPMCOUNTER4 |  |
| 0x100 | SRW | SSTATUS |  |
| 0x102 | SRW | SEDELEG |  |
| 0x103 | SRW | SIDELEG |  |
| 0x104 | SRW | SIE |  |
| 0x105 | SRW | STVEC |  |
| 0x106 | SRW | SCOUNTEREN |  |
| 0x140 | SRW | SSCRATCH |  |
| 0x141 | SRW | SEPC |  |
| 0x142 | SRW | SCAUSE |  |
| 0x143 | SRW | STVAL |  |
| 0x144 | SRW | SIP |  |
| 0x180 | SRW | SATP |  |
| 0xF11 | MRO | MVENDORID |  |
| 0xF12 | MRO | MARCHID |  |
| 0xF13 | MRO | MIMPID |  |
| 0xF14 | MRO | MHARTID |  |
| 0x300 | MRW | MSTATUS |  |
| 0x301 | MRW | MISA |  |
| 0x302 | MRW | MEDELEG |  |
| 0x303 | MRW | MIDELEG |  |
| 0x304 | MRW | MIE |  |
| 0x305 | MRW | MTVEC |  |
| 0x306 | MRW | SCOUNTEREN |  |
| 0x340 | MRW | MSCRATCH |  |
| 0x341 | MRW | MEPC |  |
| 0x342 | MRW | MCAUSE |  |
| 0x343 | MRW | MTVAL |  |
| 0x344 | MRW | MIP |  |
| 0xB00 | MRW | MCYCLE |  |
| 0xB02 | MRW | MINSTRET |  |
| 0xB03 | MRW | MPHCOUNTER3 |  |
| 0xB04 | MRW | MPHCOUNTER4 |  |
| 0x320 | MRW | MCOUNTINHIBIT |  |
| 0x323 | MRW | MHPMEVENT3 |  |

\*注: URO（User Read Only），即User可以读，不能写，S和M模式随意访问。

SRW（Supervisier Read Write），即S可以读可以写，M模式随意。

MRW（Machine Read Write），只有M模式可以读可以写。

**4.2.1 机器模式CSR**

**4.2.1.1、MISA寄存器**

MISA是一个64位的只读寄存器（在RV64上），它指示了当前处理器支持的指令集，其编码格式如下：

|  |  |  |
| --- | --- | --- |
| 63：62 | 60：26 | 25：0 |
| MXL[1:0] | WLRL | Extensions |

\* WLRL：Write leagal， Read Leagal，即读写必须按照常规值进行操作。

MXL[1:0]：指示当前处理器支持什么位数，本机只支持RV64，故此硬连线为2'b10，即64位，不可以被改写。

Extensions[25:0]: 指示当前机器支持什么拓展指令，本机只支持A和I拓展指令，故只有第0位，第8位为1，其余均为0；该寄存器也是不可以被改写的。

**4.2.1.2、MVENDORID寄存器**

MVENDORID寄存器是一个存储供应商ID的32位寄存器，在这里我们没有实现，故该寄存器硬连线为0。

**4.2.1.3、MARCHID寄存器**

MAARCHID寄存器是一个64位寄存器，它描述的是当前处理器的微架构编号，此处理器中该寄存器被硬件编码为：PRX0\_0004。

**4.2.1.4、MIMPID寄存器**

MIMPID寄存器是一个64位寄存器，它描述的是当前处理器的实现编号，在本系列处理器中，处理器实现编号可能根据硬件设计的修改而有所不同。

\*编者注：以上寄存器可以参考对照CPU-Z软甲下读取的：家族、步进号、支持指令集来理解。

**4.2.1.5、MHARTID寄存器**

MHARTID寄存器是一个64位只读寄存器，反映当前Hart的编号。本机是单核处理器，故该寄存器读取的值是1。

**4.2.1.6、MSTATUS寄存器**

MSTATUS寄存器是一个64位，在机器模式下可读可写的寄存器，此寄存器反映了当前机器模式的状态寄存器。寄存器格式如下：

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 63：36 | 35：32 | 31：23 | 22 | 21 | 20 | 19 | 18 | 17 | 16：13 | 12：11 |
| 0 | 1111 | 0 | TSR | TW | TVM | MXR | SUM | MPRV | 0 | MPP |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10：9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| 00 | SPP | MPIE | 0 | SPIE | 0 | MIE | 0 | SIE |

|  |
| --- |
| 0 |
| 0 |

\*注：其中为0的位置是RISCV对未来可能使用的位的保留，软件应当忽略这些为0的位，如果软件尝试对这些为0的位写1，将会被忽略。

**4.2.1.6.1、MSTATUS寄存器的虚拟化管理**

**TSR**：TrapSupervisorReturn，即让S模式下执行SRET指令时产生一个非法指令异常，当TSR=1时候，就会让S模式下执行SRET指令发生异常，当TSR为0时，S模式允许执行SRET指令。这样做的目的是在没有硬件虚拟机支持的时候，能让M模式模拟这条指令执行。

**TW：**TimeoutWait，当此位为1时候，在更低权限模式（M模式以下）执行WFI指令时产生异常。当WFI为0时，这个指令被允许执行。

**TVM:** TrapVirtualMemory， 当TVM=1时候，可以让S模式在进行虚拟内存管理的时候（执行SFENCE.VMA,访问SATP寄存器）产生一个非法指令异常。当TVM=0时候允许S模式进行虚拟内存管理。

\*编者注：目前RISCV对于是否加入虚拟化，各家的意见还比较杂乱，为了能在M模式下具备一定的虚拟机运行能力，使用以上几个位：TSR，TW，TVM来增加M模式对S模式的控制能力。

**4.2.1.6.2、MSTATUS寄存器的内存权限管理**

**MPRV：**ModifyPRiVilege， 改变权限，当MPRV为1时候，访问内存的方式被变更到MPP位所指的权限上。

例：假设当前MPP=2’b11，即M模式，如果MPRV=0，内存访问将按照当前权限进行，如果MPRV=1，内存访问将绕过虚拟内存（无论虚拟内存是否被打开），直接使用物理地址进行访问，即使用MPP中的M模式来进行访问。

注意，此位并不影响取指令时的地址转换和保护。

**MXR：** MakeeXcutableReadable，使可执行的为可读，当该位为1时，从页面标记为Readable或Executable（PTE中的R=1或者X=1）的页面中读数据是允许的。 当MXR=0时，只能从R=1的页面中读数据。当虚拟内存没有被打开时，这个位是无效的。

**SUM：** 使S模式可以访问U模式的页面，当SUM=0时，S模式访问到标记为U的页面将会出错，即产生一个访问页面错误，当SUM=1时是允许的。此位的目的是保护系统内核，避免S模式因为堆栈溢出等错误程序跑到用户页面里，从而使用户有攻击系统内核的可能性。 此位当虚拟内存没有打开时无效。

\*编者注：使操作系统堆栈溢出是众多攻击操作系统的手段中的一种，通过让操作系统内核堆栈溢出使系统主程序跑飞到其他内存区域，进而就可以执行用户内存中的恶意代码。虽然这种攻击手段需要一些运气，因为操作系统主程序跑飞之后并不一定刚好到恶意代码的入口处，但是通过多次尝试仍然是有可能让恶意代码被执行。通过禁用S模式访问U模式的代码可以完全的解决这个问题，一般来说，为了保证安全，操作系统开始运行后都是把SUM关闭的。

**4.2.1.6.3、MSTATUS寄存器的全局/特权中断**

**\*建议阅读此小节之前先阅读第五章：中断和特权**

**MIE：**M模式的全局中断开关，当MIE打开时，所有对M模式的中断都是被打开的。

**SIE：** S模式的全局中断开关，当SIE打开时，所有对S模式的中断都是被打开的。

**MPIE：**M模式中断等待位，当发生机器模式受理的异常/中断时，MPIE将会被更新为MIE的值。

**SPIE：**S模式终端等待位，当发生S模式受理的异常/中断时，SPIE将会被更新为SIE的值。

**MPP：**发生异常前的机器权限，当发生M模式受理的异常/中断时，MPP会被更新为发生异常前的机器权限。

**SPP：**发生异常前的机器权限，当发生S模式受理的异常/中断时，SPP会被更新为发生异常前的机器权限。

编者注：在处理器开始设计时，最新的指令集版本是2019608版，对于2019608文档之后添加的新特性，本处理器皆不具备。

**4.2.1.7、MTVEC寄存器**

MTVEC寄存器是一个存放中断/异常向量的寄存器，与X86等大多数CISC机不同的，RISCV将中断基址存放在内部寄存器中，而不是像X86处理器使用内存中的中断向量表存放中断，此举的目的是提高系统响应中断的速度。

|  |  |
| --- | --- |
| 63：2 | 1：0 |
| BASE | MODE |

**BASE**：存放跳转的基地址

**MODE**：在发生中断/异常时的地址计算方式。 当MODE=0时，所有中断都会跳转到BASE所指定的地址；当MODE=1时候，所有中断和异常都会跳转到BASE+4\*CAUSE的地址处。

在本机的实现中，对MODE域填写除01之外的值是不被允许的，也是不起作用的。

\*编者注：在使用向量模式处理中断时候，每个向量所指的地址之间的间隔最少只有4个字节，因此在向量所指的地址上应当使用JMP指令跳转到真正的中断处理函数处。

**4.2.1.8、机器模式下中断/异常委托（MEDELEG和MIDELEG寄存器）**

**\*建议阅读此小节之前先阅读第五章：中断和特权**

MIDELEG（Machine Interrupt Delegation）和MEDELEG寄存器是两个64位寄存器，管理的是中断和异常的委托。

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | DSEI | 0 | 0 | 0 | DSTI | 0 | 0 | 0 | DSSI | 0 |

MIDELEG寄存器位列表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 64：16 | 15 | 14 | 13 | 12 | 11 | 10 |
| 0 | DSPF | 0 | DLPF | DIPF | 0 | 0 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| DECS | DECU | DSAF | DSAM | DLAF | DLAM | DBK | DII | DIAF | DIAM |

MEDELEG寄存器位列表

在默认情况下，所有中断/异常都会被提高到M模式进行处理，但是在一个操作系统里面，分配好几个权限层、中断必须穿过这些权限层才能进行处理是非常低效而且不方便管理的（如在S模式下运行的linux操作系统，它本身是无法访问到M模式的这些寄存器的，要访问只能通过调用SBI或者让linux操作系统的一部分工作在M模式里，这样做是非常不方便管理的）所以M模式里增加了MEDELEG和MIDELEG寄存器，以让中断/异常可以被委托到下一个权限层被处理，这样做的目的是让系统运行起来之后，S模式下操作系统完全的接管M模式的工作。

根据RISCV指令特权文档描述，设置MIDELEG，MEDELEG寄存器中对应的位，会使对应的中断/异常被委托给S模式，例如在MCAUSE寄存器中，InstructionaddressMisaligned的编码是0，故在MEDELEG寄存器中设置BIT0=1，就会将InstructionAddressMisaligned委托给S模式，由S模式进行处理，以此类推。（不同中断/异常的编码可以在MCAUSE寄存器一节找到）

当一个中断/异常被委托到下一个权限时候（如在这里被委托到S模式），所有异常的处理将会被交给S模式。例如，假设在S模式下发生了非法指令异常，且MEDELEG寄存器中DIL位被设置为1，则这个异常将由S模式直接处理，而不是M。异常的处理会由SCAUSE，SSTATUS，SEPC，STVEC，STVAL等寄存器进行处理，M模式下的MCAUSE，MSTATUS，MEPC，MTVEC，MTVAL等将不会被影响。

注意！一个中断/异常的委托永远不会从一个高的权限交给更低的权限，例如，如果MEDELEG寄存器的DIL被设定为1，当前机器工作在M模式，且遭遇了一个非法指令异常，这个异常不会被交给S模式进行处理器，而是直接交给M模式。

机器模式下产生的中断是不可以被委托给权限更低的模式的。

**4.2.1.9、机器模式中断寄存器（MIE&MIP）**

MIP和MIE寄存器是两个64位的部分可读可写寄存器，它们指示了当前中断打开（enable）和等待（pending）的状态。

|  |  |  |  |
| --- | --- | --- | --- |
| 63：12 | 11 | 10 | 9 |
| 0 | MEIP | 0 | SEIP |
| N/A | 只读 |  | 读写 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | MTIP | 0 | STIP | 0 | MSIP | 0 | SSIP | 0 |
| N/A | 只读 | N/A | 读写 | N/A | 只读 | N/A | 读写 | N/A |

MIP寄存器位列表

|  |  |  |  |
| --- | --- | --- | --- |
| 63：12 | 11 | 10 | 9 |
| 0 | MEIE | 0 | SEIE |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | MTIE | 0 | STIE | 0 | MSIE | 0 | SSIE | 0 |

MIE寄存器位列表

MIE，MIP寄存器中的一部分位被暴露给了SIE，SIP寄存器，这一部分内容可以在SIE，SIP寄存器章节中被找到。

在MIP寄存器中，只有低于机器权限的 SSIP、STIP、SEIP是可以进行读写的，而M模式的MTIP，MEIP，MSIP寄存器是只读的，要更改这两个寄存器，需要读写内存中的机器模式定时器、软件中断控制器、外部中断控制器来更改。这样做的目的是为了让机器模式能够虚拟一个中断给S模式，从而使机器模拟更多的虚拟机行为。

中断被分成了三种类型：Timer（定时器），Software（软件），External（外部），不同全权限下的三种中断又被分为：STI（S模式定时器中断）、MTI（M模式定时器中断）、SSI（S模式软件中断），MSI（M模式软件中断）、SEI（S模式外部中断）、MEI（M模式外部中断）。

**MTIP（机器模式定时器中断等待）**，这个位当内存中的机器模式定时器计数达到设定值后发生被置1，并且只能由软件在重新设定定时器值，使定时器不满足中断条件之后才会被清零。

**STIP（S模式定时器中断等待）**，用于机器模式向S模式传递定时器中断，这是一个可以被M模式读写的寄存器，M模式可以通过写STI向S模式产生一个定时器中断，S模式只有通过进行AEE和SEE调用来让M模式清除这个位。

**MSIP（M模式软件中断等待）**，用于表示机器模式下软件中断等待，这个位是只读的，软件只能通过修改内存中的软件中断控制器来清除这个中断。

**SSIP（S模式软件中断等待）**，用于表示在S模式下软件中断等待，这个位对M模式是可读可写的，M模式可以通过写这个位向S模式产生一个软件中断，但是，这个位的清除并不需要调用AEE或者SEE，因为S模式也可以读写Sipping寄存器中的SSIP位。

**MEIP（M模式外部中断等待）**，用于表示在M模式下外部中断等待，这个位是只读的，M模式软件需要访问外部中断控制器来清除这个中断。

**SEIP（S模式外部中断等待）**，用于表示在S模式下外部中断等待状态，这个位对M来说是可读可写的，同时的，外部中断控制器也可以选择是否向S模式发起一个外部中断，这个可读可写位的变更状态和写入的值之间的关系如下表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SEIP状态 | 外部中断状态 | 要写的位 | 读取的值 | 写入后SEIP值 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |

这样设计的目的是让S模式可以被M模式虚拟一个中断，同时又不错过任何一个外部中断。

\*编者注：在RISCV目前定义的所有中断中，MTIME定时器产生的中断是固定的向M模式进行中断，即影响MTIP寄存器、机器模式软件中断也只是固定的影响MSIP位。但是外部中断可以选择中断目标到S模式或者M模式，默认的，开机复位之后外部中断的中断目标是M模式，当引导完成操作系统之后，M模式下的固件程序会将中断控制器中可以分配给S模式的中断分配给S。

在RV/AT标准中定义了一个I46459中断控制器，可以参考附录D。

**4.2.1.10、机器模式定时器MTIME &MTIMECMP**

|  |
| --- |
| MTIME |
| 63：0 |

|  |
| --- |
| MTIMECMP |
| 63：0 |

机器模式定时器MTIME和MTIMECMP寄存器是映射在内存中的两个64位寄存器，这两个寄存器的地址可以查阅对应SoC的实现手册，在RV/AT兼容机中，这两个寄存器的地址为 。

这两个寄存器的时钟使用低频时钟而非机器cycle时钟，这样做是因为现在的处理器都工作在非常高的频率（应用处理器（AP）常常工作在1GHz以上的频率），如果使用机器的主时钟，会导致功耗巨大。

当MTIME中的值大于等于MTIMECMP之后，会产生一个定时器中断，直到软件更改这两个寄存器的值之后，才会清除中断。只能通过更改内存映射中的MTIME和MTIMECMP寄存器来清除这个中断。

**4.2.1.11、硬件性能监视器**

RV提供了32个硬件性能监视器，其中，MCYCLE和MINSTRET是必须实现的，这两个计数器反映了当前机器运行指令的能力，将会指示操作系统分配合理的运行时间。

|  |  |
| --- | --- |
| MCYCLE |  |
| MINSTRET |  |
| MHPMCOUNTER3 | MHPEVENT3 |
| MHPMCOUNTER4 | MHPEVENT4 |
| 63：0 | 63：0 |

MCYCLE、MINSTRET，MHPMCOUNTER3。MHPMCOUNTER4 都是64位计数器，其中，MCYCLE，MINSTRET计数器分别是周期数和指令运行条数，操作系统可以根据这两个来判断机器执行指令的效率。

MHPEVENT3和MHPEVENT4计数器是RX/AT兼容处理器中可选实现的两个寄存器，其描述的是MHPMCOUNTER3和MHPMCOUNTER4计数器中事件计数的类型，我们规定为：

|  |  |
| --- | --- |
| ASCII=HOLD | 停止计数 |
| ASCII=L1DM | L1D访问未命中次数 |
| ASCII=TMAC | 总的内存访问次数 |

为了方便兼容RV32处理器，我们采用四位ASCII码，RV64处理器只需要进行高位补0即可。

\*注：在PRV464SX处理器中，并没有实现HPM3和HPM4，程序员需要查询CPU的IMPID来获得处理器型号信息。

**4.2.1.12、计数器使能寄存器MCOUNTEREN&SCOUNTEREN**

为了能让较低权限能访问内存，设定MCOUNTEREN和SCOUNTEREN寄存器让计数器CSR暴露给更低权限。其位如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 63：5 | 4 | 3 | 2 | 1 | 0 |
| 0 | HPM4 | HPM3 | IR | TM | CY |

SCOUNTEREN和MCOUNTEREN寄存器的布局和上图是一样的。里面的不同位表示这个寄存器是否可以暴露给下一个权限。

在PRV464的实现中，为了方便译码器的设计和不必要的麻烦，本机的TIME，CYCLE，HPM3，HPM4都是可以在任何权限被读取的（但是不可以被改写）。

参考第二页的CSR列表，URO的TIME。CYCLE，INSTRET寄存器都是可以被读取的，但是无法被改写，因为这几个寄存器是MTIME，MCYCLE，MINSTRET寄存器的一个只读副本。

**4.2.1.13、计数器停止寄存器（MCOUNTERINHIBIT）**

在很低功耗的系统里面，不必要的计数器行为将会增加功耗，除了执行WFI指令来降低动态功耗以外，停止计数器也是一个手段。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 63：0 | 4 | 3 | 2 | 1 | 0 |
| 0 | HPM4 | HPM3 | IR | 0 | CY |

MCOUNTERINHIBIT寄存器位列表

当软件置位其中的任意一个位（CYIRHPM3HPM4）时，该计数器就会停止动作。

注意！MTIME计数器是不可以被停下来的，一旦系统定时器出现故障，系统将无法正常轮转，一个经典的例子是iphone5s出现的后台冻住，其实就是因为系统定时器故障。

**4.2.1.14、机器模式临时寄存器（MSCRATCH）**

机器模式临时寄存器是一个64位可读可写寄存器，主要是在机器模式进行现场恢复/保存/交换值时使用。它不具备任何功能性特点。

**4.2.1.15、机器模式异常程序计数器（MEPC）**

MEPC是一个64位可读可写寄存器，保存的是机器在遭遇中断/异常之后程序指针的位置。

|  |
| --- |
| 63：0 |
| MEPC |

在很多其他的CISC机上，机器遭遇了异常/中断之后处理器会自动压栈来保存当前的PC值以方便返回，但是向内存中压栈会引起不确定的时间开销（如果缓存没命中）对于实时系统来说，这是非常难以接受的事情。RV采用了将异常PC值保存在内部寄存器中以提高机器的性能。

\*要理解这个寄存器，烦请查阅第五章

**4.2.1.16、机器模式异常原因寄存器（MCAUSE）**

机器模式异常原因寄存器是一个64位寄存器，它保存的是机器模式下异常/中断的原因，方便软件查询。

|  |  |
| --- | --- |
| INT | ExceptionCode |
| 63 | 62：0 |

MCAUSE寄存器被分为两个域：INT和ExceptionCode，其中INT域指示当前异常是否是中断，其编码如下：

|  |  |  |
| --- | --- | --- |
| INT | ExceptionCode | 描述 |
| 1 | 0 | USI：用户模式软件中断 |
| 1 | 1 | SSI：S模式软件中断 |
| 1 | 2 | 保留 |
| 1 | 3 | MSI：机器模式软件中断 |
| 1 | 4 | UTI：用户模式定时器中断 |
| 1 | 5 | STI：S模式定时器中断 |
| 1 | 6 | 保留 |
| 1 | 7 | MTI：机器模式定时器中断 |
| 1 | 8 | UEI：用户模式外部中断 |
| 1 | 9 | SEI：S模式外部中断 |
| 1 | 10 | 保留 |
| 1 | 11 | MEI：机器模式外部中断 |
| 1 | 12-15 | 保留 |
| 1 | >=16 | 保留 |
| 0 | 0 | IAM：指令地址不对齐 |
| 0 | 1 | IAF：指令访问失败 |
| 0 | 2 | II：非法指令 |
| 0 | 3 | BK：断点 |
| 0 | 4 | LAM：读数据地址不对齐 |
| 0 | 5 | LAF：读访问失败 |
| 0 | 6 | SAM：存数据地址不对齐 |
| 0 | 7 | SAF：存数据访问失败 |
| 0 | 8 | ECU：U模式的环境调用 |
| 0 | 9 | ECS：S模式环境调用 |
| 0 | 10 | 保留 |
| 0 | 11 | ECM：M模式环境调用 |
| 0 | 12 | IPF：指令页面错误 |
| 0 | 13 | LPF：读数据页面错误 |
| 0 | 14 | 保留 |
| 0 | 16 | SPF：存数据页面错误 |
| 0 | >=16 | 保留 |

**4.2.1.17、机器模式异常值寄存器（MTVAL）**

机器模式异常值寄存器是一个64位可读可写寄存器，它在发生BK，I/L/A，AF、AM、PF时候记录当前出现错误的虚地址（关于虚地址的概念可以参考S模式的分页模式），当执行到非法指令异常时，MTVAL寄存器被更新为异常的指令编码（高位补0），其余情况MTVAL寄存器被更新为0。

|  |
| --- |
| 63：0 |
| MTVAL |

**4.2.1.18、PMP寄存器**

由于PMU功能和MMU功能重复，本机并没有实现PMP寄存器的功能，并且删除了PMP寄存器。感兴趣的读者可以查阅RISCV特权指令文档关于PMP寄存器的说明。

**4.2.2、S模式CSR**

**4.2.2.1、S模式状态寄存器（SSTATUS）**

S模式状态寄存器是一个64位可读可写寄存器，SSTATUS寄存器是MSTATUS寄存器的一个子集，里面去除了MSTATUS寄存器中的一些只有M模式才允许控制的位，在这里我们将MSTATUS寄存器和SSTATUS寄存器放在一起进行比较。

MSTATUS：

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 63：23 | 35：32 | 31：23 | 22 | 21 | 20 | 19 | 18 | 17 | 16：13 | 12：11 |
| 0 | 1111 | 0 | TSR | TW | TVM | MXR | SUM | MPRV | 0 | MPP |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10：9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| 00 | SPP | MPIE | 0 | SPIE | 0 | MIE | 0 | SIE |

|  |
| --- |
| 0 |
| 0 |

SSTATUS：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 63：23 | 33：32 | 31：20 | 19 | 18 | 17：9 |
| 0 | 11 | 0 | MXR | SUM | 0 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SPP | 0 | 0 | SPIE | 0 | 0 | 0 | SIE | 0 |

可以发现这些位和MSTATUS寄存器同名位的位置是一样的，这是因为SSTATUS寄存器中的位和MSTATUS中同名位是联动的。换句话说，假设软件更改了SSTATUS寄存器中的MXR位，那么MSTATUS寄存器中的MXR位也会改变。

**4.2.2.1.1、SSTATUS寄存器的内存权限管理**

**MXR：** MakeeXcutableReadable，使可执行的为可读，当该位为1时，从页面标记为Readable或Executable（PTE中的R=1或者X=1）的页面中读数据是允许的。 当MXR=0时，只能从R=1的页面中读数据。当虚拟内存没有被打开时，这个位是无效的。

**SUM：** 使S模式可以访问U模式的页面，当SUM=0时，S模式访问到标记为U的页面将会出错，即产生一个访问页面错误，当SUM=1时是允许的。此位的目的是保护系统内核，避免S模式因为堆栈溢出等错误程序跑到用户页面里，从而使用户有攻击系统内核的可能性。 此位当虚拟内存没有打开时无效。

\*编者注：使操作系统堆栈溢出是众多攻击操作系统的手段中的一种，通过让操作系统内核堆栈溢出使系统主程序跑飞到其他内存区域，进而就可以执行用户内存中的恶意代码。虽然这种攻击手段需要一些运气，因为操作系统主程序跑飞之后并不一定刚好到恶意代码的入口处，但是通过多次尝试仍然是有可能让恶意代码被执行。通过禁用S模式访问U模式的代码可以完全的解决这个问题，一般来说，为了保证安全，操作系统开始运行后都是把SUM关闭的。

**4.2.1.6.3、SSTATUS寄存器的全局/特权中断**

**\*建议阅读此小节之前先阅读第五章：中断和特权**

**SIE：** S模式的全局中断开关，当SIE打开时，所有对S模式的中断都是被打开的。

**SPIE：**S模式终端等待位，当发生S模式受理的异常/中断时，SPIE将会被更新为SIE的值。

**SPP：**中断/异常前的机器权限，如果是S模式这个位是1，如果是U模式则这个位是0。只有当S模式受理了异常/中断时，这个位才会被更新。

**4.2.2.2、S模式中断向量寄存器（STVEC）**

同M模式设置了MTVEC寄存器，S模式也有自己的STVEC寄存器来处理中断，STVEC寄存器是一个存放中断/异常向量的寄存器，与X86等大多数CISC机不同的，RISCV将中断基址存放在内部寄存器中，而不是像X86处理器使用内存中的中断向量表存放中断，此举的目的是提高系统响应中断的速度。

|  |  |
| --- | --- |
| 63：2 | 1：0 |
| BASE | MODE |

**BASE**：存放跳转的基地址

**MODE**：在发生中断/异常时的地址计算方式。 当MODE=0时，所有中断都会跳转到BASE所指定的地址；当MODE=1时候，所有中断和异常都会跳转到BASE+4\*CAUSE的地址处。

在本机的实现中，对MODE域填写除01之外的值是不被允许的，也是不起作用的。

\*编者注：在使用向量模式处理中断时候，每个向量所指的地址之间的间隔最少只有4个字节，因此在向量所指的地址上应当使用JMP指令跳转到真正的中断处理函数处。

**4.2.2.3、S模式中断控制寄存器（SIP&SIE）**

S模式的SIP和SIE寄存器也是M模式的MIP和MIE寄存器的一个子集，其位如下：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 63：10 | 9 | 8 | 7：6 | 5 | 4 | 3：2 | 1 | 0 |
| 0 | SEIP | 0 | 0 | STIP | 0 | 00 | SSIP | 0 |

SIP寄存器

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 63：10 | 9 | 8 | 7：6 | 5 | 4 | 3：2 | 1 | 0 |
| 0 | SEIE | 0 | 00 | STIE | 0 | 00 | SSIE | 0 |
| N/A | 只读 | N/A | N/A | 只读 | N/A | N/A | 读写 | N/A |

SIE寄存器

需要注意的是，与MIP寄存器中可以读写SSIP、SEIP、STIP位不同，在S模式下只能读写SSIP位，其他位都是只读的，程序在写入的时候寄存器会忽略写入除开SSIP的位的值。

允许S模式下运行的程序读写SSIP位的目的是让S模式通过读写SSIP产生软件中断，当然，前提条件是M模式将该中断委托给了S模式且SIE中SSIE位要被打开才能触发，或者没有委托，直接会产生一个中断让M模式受理。

**4.2.2.4、S模式计数器使能寄存器（SCOUNTEREN）**

同M模式下的MCOUNTEREN寄存器，SCOUNTEREN寄存器管理TIME，CYCLE，INSTRET寄存器对U模式下程序的可见性，在本机实现中，U模式可以读这些寄存器而不被遮蔽，其位如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 63：5 | 4 | 3 | 2 | 1 | 0 |
| 0 | HPM4 | HPM3 | IR | TM | CY |

SCOUNTEREN和MCOUNTEREN寄存器的布局和上图是一样的。里面的不同位表示这个寄存器是否可以暴露给下一个权限。

在PRV464的实现中，为了方便译码器的设计和不必要的麻烦，本机的TIME，CYCLE，HPM3，HPM4都是可以在任何权限被读取的（但是不可以被改写）。

参考第二页的CSR列表，URO的TIME。CYCLE，INSTRET寄存器都是可以被读取的，但是无法被改写，因为这几个寄存器是MTIME，MCYCLE，MINSTRET寄存器的一个副本。

**4.2.2.5、S模式临时寄存器（SSCRATCH）**

SSCRATCH寄存器的功能和M模式下使用的MSCRATCH功能一样。

**4.2.2.6、S模式异常程序指针寄存器（SEPC）**

SEPC是一个64位可读可写寄存器，保存的是机器在遭遇中断/异常之后程序指针的位置。

|  |
| --- |
| 63：0 |
| SEPC |

在很多其他的CISC机上，机器遭遇了异常/中断之后处理器会自动压栈来保存当前的PC值以方便返回，但是向内存中压栈会引起不确定的时间开销（如果缓存没命中）对于实时系统来说，这是非常难以接受的事情。RV采用了将异常PC值保存在内部寄存器中以提高机器的性能。

\*要理解这个寄存器，烦请查阅第五章

**4.2.2.7、S模式异常原因寄存器（SCAUSE）**

SCAUSE寄存器的功能和M模式下使用的MCAUSE寄存器功能一样，监督者模式异常原因寄存器是一个64位寄存器，它保存的是监督者模式下异常/中断的原因，方便软件查询。

|  |  |
| --- | --- |
| INT | ExceptionCode |
| 63 | 62：0 |

SCAUSE寄存器被分为两个域：INT和ExceptionCode，其中INT域指示当前异常是否是中断，其编码如下：

|  |  |  |
| --- | --- | --- |
| INT | ExceptionCode | 描述 |
| 1 | 0 | USI：用户模式软件中断 |
| 1 | 1 | SSI：S模式软件中断 |
| 1 | 2 | 保留 |
| 1 | 3 | 保留 |
| 1 | 4 | UTI：用户模式定时器中断 |
| 1 | 5 | STI：S模式定时器中断 |
| 1 | 6 | 保留 |
| 1 | 7 | 保留 |
| 1 | 8 | UEI：用户模式外部中断 |
| 1 | 9 | SEI：S模式外部中断 |
| 1 | 10 | 保留 |
| 1 | 11 | 保留 |
| 1 | 12-15 | 保留 |
| 1 | >=16 | 保留 |
| 0 | 0 | IAM：指令地址不对齐 |
| 0 | 1 | IAF：指令访问失败 |
| 0 | 2 | II：非法指令 |
| 0 | 3 | BK：断点 |
| 0 | 4 | LAM：读数据地址不对齐 |
| 0 | 5 | LAF：读访问失败 |
| 0 | 6 | SAM：存数据地址不对齐 |
| 0 | 7 | SAF：存数据访问失败 |
| 0 | 8 | ECU：U模式的环境调用 |
| 0 | 9 | ECS：S模式环境调用 |
| 0 | 10 | 保留 |
| 0 | 11 | ECM：M模式环境调用 |
| 0 | 12 | IPF：指令页面错误 |
| 0 | 13 | LPF：读数据页面错误 |
| 0 | 14 | 保留 |
| 0 | 16 | SPF：存数据页面错误 |
| 0 | >=16 | 保留 |

**4.2.2.8、S模式异常值寄存器（STVAL）**

监督者模式异常值寄存器是一个64位可读可写寄存器，它在发生BK，I/L/A，AF、AM、PF时候记录当前出现错误的虚地址（关于虚地址的概念可以参考S模式的分页模式），当执行到非法指令异常时，MTVAL寄存器被更新为异常的指令编码（高位补0），其余情况MTVAL寄存器被更新为0。

|  |
| --- |
| 63：0 |
| STVAL |

**4.2.2.9、S模式地址转换和保护寄存器（SATP）**

SATP寄存器是S模式独有的寄存器，这个寄存器决定了是否打开虚拟内存。

|  |  |  |
| --- | --- | --- |
| 63：60 | 59：44 | 43：0 |
| MODE | ASID（0） | PPN |

SATP寄存器中的MODE域指示当前运用的分页方式，目前RV64规定了：Sv39和Sv48两种分页方式，因为实在没有必要支持如此庞大的内存大小，故本机只选择支持Sv39（或改进型Sv39CT）分页方案。当MODE域为0时，没有分页，系统运行在物理地址上，当MODE=8时，工作在Sv39分页模式上。

ASID存储当前的线程号，由于本机并没有TLB，故ASID被硬连线0，在RV架构要求里，程序需要对ASID全写1，然后读取以确定有多少位是有效的。

**4.2.3、U模式的寄存器**

**4.2.3.1、CYCLE**

此寄存器是MCYCLE寄存器的一个只读副本，在任何模式下，U和S模式的软件只能读取这个寄存器。

**4.2.3.2、TIME**

此寄存器是映射在内存中的MTIME寄存器的一个只读副本，在任何模式下，U和S只能读取这个寄存器。

**4.2.3.3、INSTRET**

此寄存器是MINSTRET寄存器的一个只读副本。

**4.2.3.4、HPMCOUNTER3**

目前的处理器版本（PRV464SX）暂未实现该寄存器。

**4.2.3.5、HPMCOUNTER4**

目前的处理器版本（PRV464SX）暂未实现该寄存器。

编者注：在本文档撰写的过程中，参考的文档是RISCV2019608文档，如果在之后的版本中RISCV对其中某些特性做了更改，本文档不做修改。

**第五章：异常和特权**

**5.1、RISCV对中断/异常的管理**

中断，即处理器运行过程中突然发生了某件事，让处理器不得不停下当前指令流而执行其他指令。在RISCV架构的定义中，将中断和异常精确的定义给了两种不同的情形：

1. 中断，在xIE寄存器中是可以被遮蔽的，RV定义了三种中断类型，TIMER（定时器）、SOFT（软件）、EXT（外部）。
2. 异常，是不可以被遮蔽的，处理器遭遇了异常之后无论如何都需要进入异常服务程序。RV架构定义了多种异常，可见上一章节对MCAUSE寄存器的描述。

**5.1.1、中断的遮蔽机制**

在RISCV架构中，中断可以通过寄存器进行遮蔽，其中主要有两个寄存器负责这个工作：xSTATUS寄存器中的xIE、xIE寄存器，在本机中并不支持U模式直接处理中断，故x只可能等于M或者S。请注意，xSTATUS寄存器中的xIE位和xIE寄存器不是一个东西，下文中为了作出区分，我们将xSTATUS寄存器中的xIE位叫做“xIE位”，xIE寄存器叫做“xIE寄存器”。

xIE位是x模式下中断的全局开关，当xIE位被软件置1时候，这个模式的所有中断都被关闭，不管xIE寄存器里面的设定如何。当xIE位被置1时，x模式的中断被全局打开，此时xIE寄存器中的设置才会起作用。

xIE位、xIE寄存器中的位管理的是当前权限x下异常的遮蔽，只有当xIP寄存器中的一个中断在等待，且xIE寄存器中对应的位被打开且xIE位打开的时候，中断才会被受理。例如，MIE寄存器中的MTIE被打开，且MIE被打开，之后一个定时器中断使MIP寄存器的MTIP位置1，那么此时中断就会被受理，处理器自动的跳转到中断服务程序入口地址处。

**5.1.2、中断/异常的委托机制**

在默认情况下，所有中断/异常都会被提高到M模式进行处理，但是在一个操作系统里面，分配好几个权限层、中断必须穿过这些权限层才能进行处理是非常低效而且不方便管理的（如在S模式下运行的linux操作系统，它本身是无法访问到M模式的这些寄存器的，要访问只能通过调用SBI或者让linux操作系统的一部分工作在M模式里，这样做是非常不方便管理的）所以M模式里增加了MEDELEG和MIDELEG寄存器，以让中断/异常可以被委托到下一个权限层被处理，这样做的目的是让系统允许起来之后，S模式下操作系统完全的接管M模式的工作。

根据RISCV指令特权文档描述，设置MIDELEG，MEDELEG寄存器中对应的位，会使对应的中断/异常被委托给S模式，例如在MCAUSE寄存器中，InstructionaddressMisaligned的编码是0，故在MEDELEG寄存器中设置Bit0=1，就会将InstructionAddressMisaligned委托给S模式，由S模式进行处理，以此类推。（不同中断/异常的编码可以在MCAUSE寄存器一节找到）

当一个异常被委托到下一个权限时候（如在这里被委托到S模式），该异常的处理将会被交给S模式。例如，假设在S模式下发生了非法指令异常，且MEDELEG寄存器中DIL位被设置为1，则这个异常将由S模式直接处理，而不是M。异常的处理会由SCAUSE，SSTATUS，SEPC，STVEC，STVAL等寄存器进行处理，M模式下的MCAUSE，MSTATUS，MEPC，MTVEC，MTVAL等将不会被影响。

当一个中断被委托给下一个权限的时候（如委托给S模式），与异常委托不一样的是，中断的委托是可以被委托的模式给遮蔽的。例如，如果一个外部中断在S模式下发生了，且该中断被委托给了S模式，如果SIE位为1且SEIE为1，这个中断将会被S模式接受并处理；如果SIE位为0或者SEIE为0，那么S模式将不会受理这个中断。

当一个中断被委托之后，运行在高于被委托权限的权限下，处理器将不会受理这个中断，例如如果M模式将SEI委托给了S模式，而后工作在M模式时发生了一个S模式的外部中断，M模式将不会受理这个中断。

默认的，当一个中断没有被交给下一个权限级，假设MIDELEG寄存器中DSTIP位没有置1，当前运行在S模式，且STIP位为1表示有一个定时器中断发生了，那么处理器无论如何会转到M模式来处理这个中断而不管SIE寄存器的状态。

\*注：RISC-V让在低权限下运行的时候，发生的没有被委托的中断可以无视IE寄存器直接转到M模式的目的是让M模式更好的模拟虚拟机。

下表表示了中断委托和受理机制：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 当前权限 | 是否进行委托 | 被委托的模式是否遮蔽 | M模式是否遮蔽 | 是否处理这个中断 |
| M | 否 | x | 是 | 否，M模式遮蔽了中断 |
| M | 否 | x | 否 | 是，以M模式受理 |
| M | 是 | x | x | 否，已经委托给S模式 |
| S | 否 | x | x | 是，以M模式受理 |
| S | 是 | 是 | x | 否，S模式屏蔽了中断。 |
| S | 是 | 否 | x | 是，以S模式受理 |
| U | 否 | x | x | 是，以M模式 |
| U | 是 | 是 | x | 否，S模式遮蔽了中断 |
| U | 是 | 否 | x | 是，以S模式受理 |
|  |  |  |  |  |

* X代表任意值

中断委托机制说明

注意！一个中断/异常的委托永远不会从一个高的权限交给更低的权限，例如，如果MEDELEG寄存器的DIL被设定为1，当前机器工作在M模式，且遭遇了一个非法指令异常，这个异常不会被交给S模式进行处理器，而是直接交给M模式，同样的M模式的中断也不会移交给S模式进行处理。

**5.2、RISCV中断/异常的处理**

**5.2.1、进入中断/异常**

大部分处理器在进入中断/异常服务程序之后都会全自动的对现场进行保存，这个被成为“保存上下文”，当程序从中断/异常服务程序返回时，会进行现场恢复，也就是恢复之前程序运行的寄存器值和处理器状态，这个动作被称为“恢复上下文”，例如万恶之源 X86系列处理器（以实模式举例），首先处理器会查询IVT（InterruptVectorTable），即中断向量表，获得当前异常的服务程序入口地址，然后处理器自动的跳转到这个地址，同时自动的对PC值进行压栈。如前文中对MTVEC寄存器的描述，处理器每次遭遇中断之后都去查询IVT是相当缓慢的，尤其在缓存没有命中的情况下，虽然这种“全自动”行为确实节约了程序在保存/恢复现场时的代码。

RISC-V架构采用了截然不同的中断处理流程，当遭遇了中断/异常之后，RV处理器会自动的执行以下动作：

1. 更新异常原因寄存器，如果这个中断/异常被委托给了S模式，那么更新的是S模式异常原因寄存器（SCAUSE）。
2. 更新异常程序计数器寄存器（xEPC），同上文描述的一样，当这个中断/异常被委托给S的时候，更新的是SEPC，否则是MEPC被更新。
3. 更新异常值寄存器（xTVAL）。
4. 更新状态寄存器（xSTATUS）。
5. 从xTVEC寄存器指定的地址开始运行。

**\*更新异常原因寄存器：**

xCAUSE存储当前遭遇异常的种类，xCAUSE寄存器中编码的定义如第四章中对MCAUSE寄存器介绍的一样，例如，当处理器遭遇了一个IAF（指令访问失败）xCAUSE寄存器会被更新为1。

**\*更新异常程序计数器寄存器（xEPC）**

xEPC寄存器保持了当前遇到异常的指令流的异常点，相当于某些其他架构的处理器会将当前遇到异常的指令的地址存入堆栈中。

对于遭遇的中断而言，处理器会自动的将当前指令流的下一条地址写回xEPC，这样处理器在返回的时候就可以接着之前的指令流。如果是遭遇了异常，那么xEPC将会被更新为遭到异常指令的地址。

特别需要注意的是，对于ECALL和EBREAK异常而言，xEPC会自动的更新为下一条指令的地址而不是当前指令的地址，这样做的目的是让返回之后跳过ECALL和EBREAK指令，不然处理器就会陷入ECALL/EBREAK-受理异常-返回-再次遇到ECALL/EBREAK的死循环。

xEPC也是一个可读可写的寄存器，程序如果需要的话，可以去更改xEPC来更改返回地址。

**\*更新异常值寄存器**

当遭遇/中断异常时，xCAUSE会被更新为当前遭遇异常时的值。

1. 在遭遇非法指令异常时，更新为非法指令的编码。
2. 在遭遇ECALL和EBREAK时，更新为ECALL和EBREAK的地址。
3. 访问存储器失败的时候，更新为访问存储器的虚拟地址。

**\*更新状态寄存器xSTATUS**

1、更新xIE位到xPIE，同时清零xIE。例如：如果是M模式受理了这个异常，MIE位被复制到MPIE位，然后MIE位被清零。此举的目的是让M模式彻底将中断关闭，避免中断/异常被嵌套导致MEPC，MTVAL，MCAUSE中保存的值丢失。

2、更新异常/中断前的权限到xPP，例如：如果在S模式下发生了机器模式定时器中断，M模式受理了这个中断，那么MPP位将会被更新为2’b01，即S模式的编码，同时机器权限会被切换到M模式。

**\*从xTVEC指定的地址开始运行**

xTVEC的MODE域管理在发生中断/异常时的地址计算方式。 当MODE=0时，所有中断都会跳转到BASE所指定的地址；当MODE=1时候，所有中断和异常都会跳转到BASE+4\*CAUSE的地址处。

**5.2.2、退出中断、异常**

同大部分处理器使用RET指令来从中断/异常服务程序返回一样，RV也使用xRET指令来从中断/异常服务程序中返回。并且，RV一共定义了三种指令从不同模式下返回：URET：从U模式返回、SRET：从S模式返回、MRET：从M模式返回。例如，如果现在执行的是M模式下的中断/异常服务程序，那么程序需要通过执行MRET指令来返回，S模式下的服务程序就需要执行SRET指令返回。由于本机并没有实现N拓展指令，故没有URET指令。

当执行了xRET指令时，以下内容发生：机器权限变更到xPP中定义的权限、从MEPC中保存的地址开始执行。

编者注：当处理器在执行S模式下的中断/异常服务程序时，如果一个更高阶的中断发生了，如机器模式定时器产生了一个定时器中断，那么这个时候中断是无条件的被移交给M模式进行处理的（在大部分情况下，M模式都不会关掉机器模式定时器）。可以这样进行嵌套的原因是S模式和M模式使用完全不同的中断处理寄存器：SEPCvsMEPC、 SSTATUSvsMSTATUS等。 在RV架构的定义中，硬件是不支持在同一个模式下运行的程序进行自动嵌套的，即需要软件先保存了上下文，才能打开中断开关允许下一次中断。

**5.3、RISCV分页（Sv39）和特权**

**5.3.1、基于Sv39的虚拟内存**

当程序在SATP寄存器中把MODE域设定为8，即打开Sv39分页方案时，虚拟内存保护就被打开了，程序在修改了SATP寄存器后需要执行一个SFENCE.VMA指令以确保分页内存被完全的打开。

一个标准的分页流程如下：

1. A=SATP.PPN\*PAGE SIZE, I = LEVEL–1

( RV64的LEVEL是3，PAGESIZE=2^12)

1. 从地址A+VA.VPN[i]\*PTESIZE中取得第页表（Sv39的PTESIZE=8）
2. 如果V=0，或者R=0且W=1，造成页面错误。
3. 其余情况下，这个页表是有效的，如果R=1或者X=1，转到第五步，否则这个页是一个指针，它指向下一级页表。将I =I-1，如果I=0则造成页面错误，其他情况下 A=PTE.PPN\*PAGESIZE,然后到第二步。
4. 当前页表是一个末端页表，MMU会根据该页表中R，W，X位的状态和当前访问的类型来决定是否可以访问，如果不允许访问，则会引起页面错误。
5. 如果I > 0 但是PTE.PPN[i-1:0]不等于0，这是一个不对齐的超页面，这将会引起页面错误，如果没有发生错误前往下一步。
6. 这个页面被成功的访问了，如果A=0，那么MMU自动的将A位置1表示这个页面被访问了，如果这个页面是写数据且D位不为1，那么还会把D位置1
7. 地址已经转换完成

\*PA.PGOFF = VA.PGOFF（VA和PA的地址偏移相同）

\*如果I> 0，那么这是一个超页面，PA.PPN[i-1:0]=VA.PPN[i-1:0]

\*最后，PA.PPN[LEVEL-1:0]=PTE.PPN[LEVELS-1:0]

在Sv39分页方案中的PTE（页表）、PA（物理地址）、VA（虚拟地址）划分如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 39:30 | 29:21 | 20:12 | 11：0 |
| VPN[2] | VPN[1] | VPN[0] | Page offset |

Sv39 虚拟地址构成

\*注：Sv39分页方案的虚拟地址并不允许使用39位以上的地址，齐39位以上的地址需要保持与第38位相同（相当于符号位拓展），这样做的目的是为了限制程序访问的空间的大小，在计算机历史上，因为允许程序访问太大范围的地址而造成后期升级困难的问题已经层出不穷。

|  |  |  |  |
| --- | --- | --- | --- |
| 55：30 | 29：21 | 20：12 | 11：0 |
| PPN[2] | PPN[1] | PPN[0] | Page offset |

Sv39物理地址

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 9：8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RSW | D | A | G | U | X | W | R | V |

|  |  |  |  |
| --- | --- | --- | --- |
| 63：54 | 53：28 | 27：19 | 18：10 |
| 保留 | PPN[2] | PPN[1] | PPN[0] |

Sv39页表

页表中的不同位代表了这个页面不同的属性：

V：Value，表示这个页面有效，如果在地址转换中V=0，表示这个页面无效。

R：Readable，表示这个页面可以读。

W：Writeable，表示这个页面可以写。

X：eXecute，表示这个页面可以执行。

U：User，表示这个页面是用户页面。

G：Global，表示这个页面是否全局有效，本机没有TLB，故该位无效。

A：Accessed，表示这个页面已经被访问过。当A被清零时，如果程序访问了这个页面，那么A位将会被自动的置1。

D：Dirty，脏位，表示这个页面被写过。当D被清除时，如果程序对这个页面进行了写入，那么这个位会被自动置1。

在Sv39分页方案中，X，W，R同时充当着指示当前页面是否是末端页面的作用，其组合和含义如下：

|  |  |  |  |
| --- | --- | --- | --- |
| X | W | R | 含义 |
| 0 | 0 | 0 | 指向下一级页表的指针 |
| 0 | 0 | 1 | 只读页面 |
| 0 | 1 | 0 | 保留 |
| 0 | 1 | 1 | 可读可写页面 |
| 1 | 0 | 0 | 只可执行页面 |
| 1 | 0 | 1 | 可读可执行页面 |
| 1 | 1 | 0 | 保留 |
| 1 | 1 | 1 | 可读可写可执行页面 |

\*编者注： 为了增加处理器运行的效率，在不和外存交换页面时，程序可以把A和D位置1来提高运行效率。

**5.3.2、RISCV特权**

RISCV一共定义了M（机器模式），H（虚拟），S（监督者），U（用户）四种权限，但是H模式因为到现在各家争论不一还没有被正式的发布（参考2019608文档）。故本机只实现了M，S，U三种权限。

|  |  |  |  |
| --- | --- | --- | --- |
| 层级 | 编码 | 名称 | 缩写 |
| 0 | 00 | User（用户） | U |
| 1 | 01 | Supervisor（监督者） | S |
| 2 | 10 | 保留（虚拟） | H |
| 3 | 11 | Machine（机器） | M |

RISCV关于特权的定义

\*编者注：在目前设计的大量ISA中，减少权限层的数量已经成为共识，大部分的RISC处理器只需要两个权限（S和U）就可以正常运行。在RISCV中也是如此，linux操作系统工作在S层，OpenSBI和固件工作在M层，用户软件工作在U层。

如果需要切换权限，有两种情况：A、从低权限切换到高权限。B、从高权限切换到低权限。

1. 程序通过执行ECALL或者EBREAK指令，切换到更高的权限。
2. 程序通过写xSTATUS寄存器的xPP位，然后执行RET指令降低到更低的权限，注意，以这种方式切换权限时候，需要注意xEPC寄存器的值。

**第六章：致谢**

从头开始理解一个指令体系架构是非常困难的，在处理器设计和本文档编写的过程中，我从各方专业人士处得到了帮助，在此我向他们表达由衷的谢意：

感谢我的家人在我开发过程中提供的默默支持；感谢“金刚光”同志，“四倍速”同志，和广大RISC-V开发者们提供的无私帮助；最后，感谢芯来科技在本处理器开发中提供的支持和鼓励！

**附录A、RISC-V指令编码（RV64）**

**A1、RV64I指令编码。**

**A2、RV64A指令编码。**

**附录B、PRV464程序优化说明。**

**B1、寄存器使用间隔**

PRV464SX使用采用四级流水线，同时没有采用任何的数据旁路机制，产生数据相关性后，处理器会自动的对产生相关性的指令进行延迟，并等待前面的指令完全的被执行之后才会发射新的指令。在这里，产生的数据相关性只会是WAR（WriteAfterRead），即读后写相关性，意思是指令在前一个指令修改了一个寄存器（即目的寄存器），而后续指令需要用到这个寄存器（即源寄存器）。

一个产生相关性的汇编代码如下：

*ADDX2, X0, X1 ;写回寄存器是X2*

*ADD X3, X2, X1;源寄存器使用了X2*

或者：

*ADD X2, X0, X1;写回寄存器使用X2*

*ADD X15, X16, X0;没有使用X2寄存器*

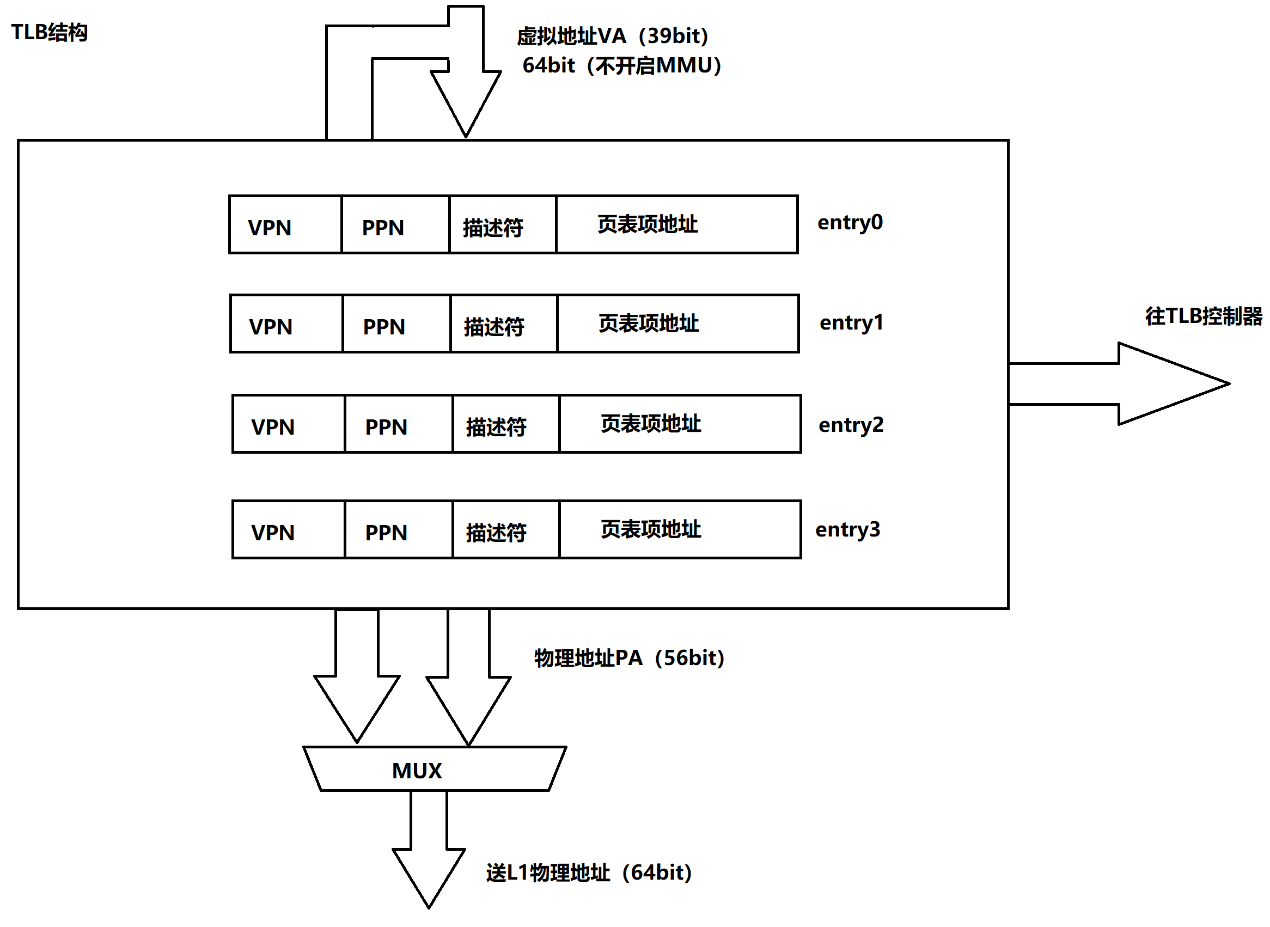
*ADD X3, X2, X1;源寄存器使用了X2*

在PRV464处理器中，两条发生数据相关性的指令应当间隔至少2条指令，这样处理器就可以以最佳速度运行，而不会自动插入等待状态。

\*注：这不是一个强制要求，只是为了提高代码运行速度进行的优化。

**B2、TLB和L1缓存**

TLB为4+4entry，指令/数据各4个入口，TLB简单原理框图如下：



每个TLBentry包含了一个独立的VPN—PPN映射，同时保留了此页表项的地址，这样硬件可以在当前页面发生写入时，自动的进行PTED位更新，在这里推荐程序员如果不需要和外存交换页面时，把所有页表项的D位置1。

当硬件打开了虚拟内存（Sv39分页方案），所有的地址立刻变换为虚拟地址，无需使用fence指令进行隔离。程序给出的虚拟地址会首先在TLB中进行查询，当TLB没有命中后，硬件会自动的启动TLB控制器从内存中获得页表，装填到之中。

替换TLB表项遵守如下规则：

1. 优先替换valid=0的页表项。
2. 若1不满足，优先替换出G=0的项
3. 若2不满足，则替换出访问次数最少的页表项，如果多个页表访问次数相等，则随机替换。

L1缓存为8+8，指令和数据L1缓存完全独立，且均为独立的4-Way，每个line有2Kbyte。当L1不命中时，硬件自动的选取一行替换，替换策略为：

1. 优先替换出valid=0的行
2. 若1不满足，则替换出访问次数最少的行。
3. 若2不满足，则随机选取一行。

**B3、指令运行时间**

PRV464SX处理器执行时间从1T-10T不等，下表标明了每个指令所需的运行时间（最好情况下，即没有相关性和等待）

|  |  |  |  |
| --- | --- | --- | --- |
| 指令 | 周期 | 指令 | 周期 |
| LUI | 1 |  |  |
| AUIPC | 1 |  |  |
| JAL | 4 |  |  |
| JALR | 4 |  |  |
| BEQ | 4 |  |  |
| BNE | 4 |  |  |
| BLT | 4 |  |  |
| BGE | 4 |  |  |
| BLTU | 4 |  |  |
| BGEU | 4 |  |  |
| LB | 3 |  |  |
| LH | 3 |  |  |
| LW | 3 |  |  |
| LBU | 3 |  |  |

**附录C、P46459平台中断控制器**

P46459中断控制器是RV/AT平台中一个标准化的中断控制器，为了减少LUT的占用，本机给该中断控制器分配了64Kbyte的空间。

由于本机计划使用一些X86的外设，所以部分中断信号按照X86的中断顺序排列。例如在RV/AT中使用的ISA插槽，和在未来的RV/A2T中使用的PCI乃至PCIE插槽

**附录F、PVS-BT拓展指令集**

类似Transmeta公司的CMS软硬件协同二进制翻译解决方案与中科龙芯在LoongISA中提供的LoongBT指令集，为了加速用户态QEMU对x86/x86-64的指令二进制翻译过程，我们决定追加PVS-BT指令集。

设计该指令集的目的有以下几点：

1. 增加一系列位于RISC-V S态的指令用于对跨平台虚拟机的内存管理和直接硬件访问机制进行高效仿真和管理
2. 用于对老旧架构的FLAG寄存器、指令条件执行机制进行高效兼容
3. 用于将老旧架构的权限管理归并到RISC-V系统的S/M/U三层结构中
4. （与S&V指令扩展协同）实现对x86历史遗留的80bit浮点结构高效兼容

**附录S、S拓展指令**

由于RV在最新的标准里（20200229），为了保证对历史遗留问题的兼容，添加了大小端序支持，为了更好的支持RISC-V的发展，兼容更多历史遗留问题，使RISCV走到X86的高度，我们决定制作S拓展指令。

1、BCD十进制调整指令

BDA RD， RS1 ；将RS1的数值进行十进制调整到BCD，带符号

BDAU RD， RS1；将RS1的数值进行十进制调整到BCD，不带符号

2、二进制调整BCD指令

BBA RD，RS1 ；将RS1中的BCD码做二进制调整，带符号

BBAU RD， RS1 ；将RS1的BCD码做二进制调整，不带符号

3、XS-3十进制调整指令

XDA RD， RS1 ；将RS1的数值进行十进制调整到XS-3，带符号

XDAU RD， RS1；将RS1的数值进行十进制调整到XS-3，不带符号

4、二进制调整XS-3指令

XBA RD，RS1 ；将RS1中的BCD码做二进制调整，带符号

XBAU RD， RS1 ；将RS1的BCD码做二进制调整，不带符号

5、字节交换指令

BSP RD， RS1

HSP RD， RS1

WSP RD， RS1

6、ASCII字符大小写切换指令

CAP RD， RS1 ；将RS1中的ASCII字符切换到大写

ICAP RD， RS1 ；将RS1的字符切换到小写

7、ASCII-数码管译码指令

BTS RD， RS1 ；对RS1中的BCD进行7段共阳数码管显示译码

XTS RD， RS1 ；对RS1中的XS-3码进行数码管显示译码

\*注：S拓展指令可能在之后的464处理器版本中被加入，此拓展指令的必要性还在讨论中。