（工程样品版，少部分内容可能在之后的版本被更改）

PRV464SX 处理器编程手册

编号：20200505

目录

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-V64 I，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

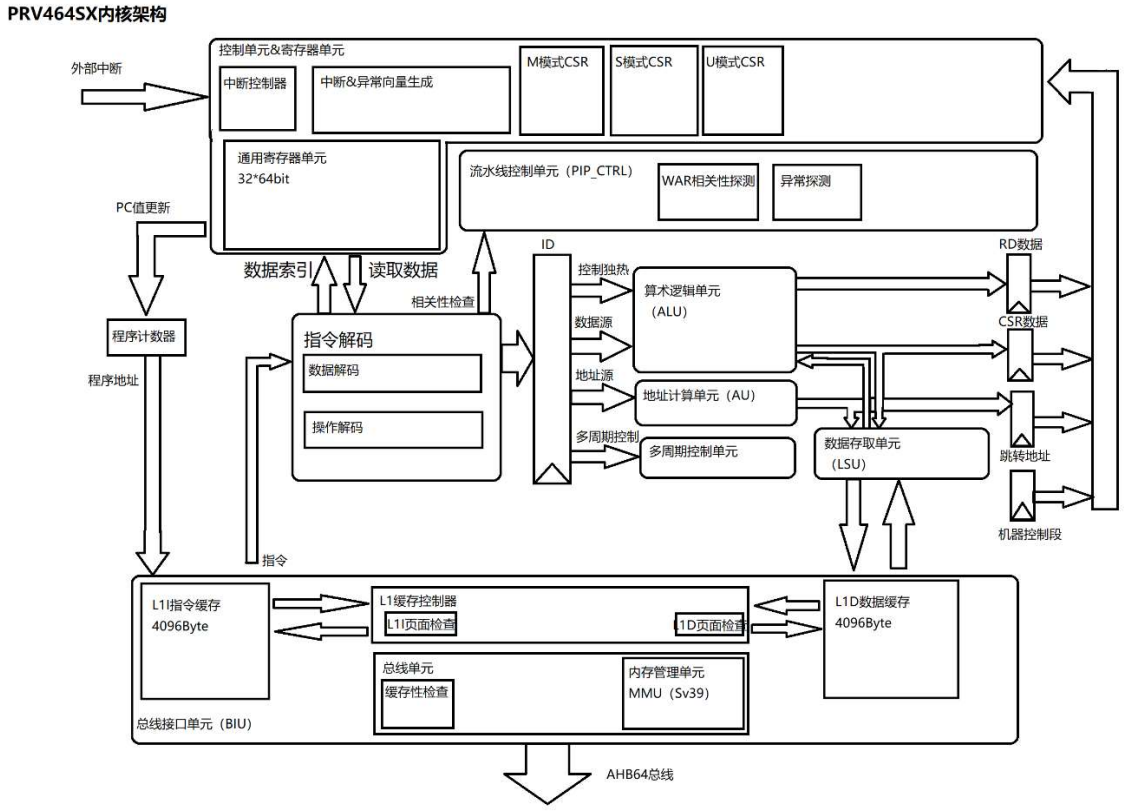
2020 年 3 月 15 日

第一章：概览

PRV464SX 处理器是 PRV4 系列处理器的第一个型号，主要面向入门级 FPGA（容量在 10K-20K LUT4），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 Cyclone IV 平台上占用 11K LUT4，在安路 EG4S20 平台上占用 8K LUT5。

1.1、PRV464SX 流水线结构

PRV464SX 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 处理器配备了 4K+4K 的 L1 缓存，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、PRV464SX 支持的通用寄存器、CSR 列表

PRV464 支持 RV64IA 指令集，根据 RISCV ABI（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： User Read Only， 用户模式只读

*SRO： Supervisor Read Only，监督者模式只读

*SRW： Supervisor Read Write，监督者模式可读可写

*MRO： Machine Read Only，机器模式只读

*MRW： Machine Read Write，机器模式可读可写

1.3、PRV464SX 支持的指令列表

以下内容截取于 RISC-V spec 文档第 132 页。

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。

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]					rd	opcode	U-type
imm[20|10:1|11|19:12]					rd	opcode	J-type

RV32I Base Instruction Set

imm[31:12]					rd	0110111	LUI
imm[31:12]					rd	0010111	AUIPC
imm[20|10:1|11|19:12]					rd	1101111	JAL
imm[11:0]			rs1	000	rd	1100111	JALR
imm[12|10:5]		rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]		rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]		rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]		rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]		rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]		rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]			rs1	000	rd	0000011	LB
imm[11:0]			rs1	001	rd	0000011	LH
imm[11:0]			rs1	010	rd	0000011	LW
imm[11:0]			rs1	100	rd	0000011	LBU
imm[11:0]			rs1	101	rd	0000011	LHU
imm[11:5]		rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]		rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]		rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]			rs1	000	rd	0010011	ADDI
imm[11:0]			rs1	010	rd	0010011	SLTI
imm[11:0]			rs1	011	rd	0010011	SLTIU
imm[11:0]			rs1	100	rd	0010011	XORI
imm[11:0]			rs1	110	rd	0010011	ORI
imm[11:0]			rs1	111	rd	0010011	ANDI
0000000		shamt	rs1	001	rd	0010011	SLLI
0000000		shamt	rs1	101	rd	0010011	SRLI
0100000		shamt	rs1	101	rd	0010011	SRAI
0000000		rs2	rs1	000	rd	0110011	ADD
0100000		rs2	rs1	000	rd	0110011	SUB
0000000		rs2	rs1	001	rd	0110011	SLL
0000000		rs2	rs1	010	rd	0110011	SLT
0000000		rs2	rs1	011	rd	0110011	SLTU
0000000		rs2	rs1	100	rd	0110011	XOR
0000000		rs2	rs1	101	rd	0110011	SRL
0100000		rs2	rs1	101	rd	0110011	SRA
0000000		rs2	rs1	110	rd	0110011	OR
0000000		rs2	rs1	111	rd	0110011	AND
fm	pred	succ	rs1	000	rd	0001111	FENCE
000000000000			00000	000	00000	1110011	ECALL
000000000001			00000	000	00000	1110011	EBREAK

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type

RV64I Base Instruction Set (in addition to RV32I)

imm[11:0]		rs1	110	rd	0000011	LWU
imm[11:0]		rs1	011	rd	0000011	LD
imm[11:5]	rs2	rs1	011	imm[4:0]	0100011	SD
000000	shamt	rs1	001	rd	0010011	SLLI
000000	shamt	rs1	101	rd	0010011	SRLI
010000	shamt	rs1	101	rd	0010011	SRAI
imm[11:0]		rs1	000	rd	0011011	ADDIW
0000000	shamt	rs1	001	rd	0011011	SLLIW
0000000	shamt	rs1	101	rd	0011011	SRLIW
0100000	shamt	rs1	101	rd	0011011	SRAIW
0000000	rs2	rs1	000	rd	0111011	ADDW
0100000	rs2	rs1	000	rd	0111011	SUBW
0000000	rs2	rs1	001	rd	0111011	SLLW
0000000	rs2	rs1	101	rd	0111011	SRLW
0100000	rs2	rs1	101	rd	0111011	SRAW

RV32/RV64 *Zifencei* Standard Extension

imm[11:0]	rs1	001	rd	0001111	FENCE.I

RV32/RV64 *Zicsr* Standard Extension

csr	rs1	001	rd	1110011	CSRRW
csr	rs1	010	rd	1110011	CSRRS
csr	rs1	011	rd	1110011	CSRRC
csr	uimm	101	rd	1110011	CSRRWI
csr	uimm	110	rd	1110011	CSRRSI
csr	uimm	111	rd	1110011	CSRRCI

1.3.2、RV64A 指令集

31 27	26	25	24 20	19 15	14 12	11 7	6 0	
funct7			rs2	rs1	funct3	rd	opcode	R-type

RV32A Standard Extension

00010	aq	rl	00000	rs1	010	rd	0101111	LR.W
00011	aq	rl	rs2	rs1	010	rd	0101111	SC.W
00001	aq	rl	rs2	rs1	010	rd	0101111	AMOSWAP.W
00000	aq	rl	rs2	rs1	010	rd	0101111	AMOADD.W
00100	aq	rl	rs2	rs1	010	rd	0101111	AMOXOR.W
01100	aq	rl	rs2	rs1	010	rd	0101111	AMOAND.W
01000	aq	rl	rs2	rs1	010	rd	0101111	AMOOR.W
10000	aq	rl	rs2	rs1	010	rd	0101111	AMOMIN.W
10100	aq	rl	rs2	rs1	010	rd	0101111	AMOMAX.W
11000	aq	rl	rs2	rs1	010	rd	0101111	AMOMINU.W
11100	aq	rl	rs2	rs1	010	rd	0101111	AMOMAXU.W

RV64A Standard Extension (in addition to RV32A)

00010	aq	rl	00000	rs1	011	rd	0101111	LR.D
00011	aq	rl	rs2	rs1	011	rd	0101111	SC.D
00001	aq	rl	rs2	rs1	011	rd	0101111	AMOSWAP.D
00000	aq	rl	rs2	rs1	011	rd	0101111	AMOADD.D
00100	aq	rl	rs2	rs1	011	rd	0101111	AMOXOR.D
01100	aq	rl	rs2	rs1	011	rd	0101111	AMOAND.D
01000	aq	rl	rs2	rs1	011	rd	0101111	AMOOR.D
10000	aq	rl	rs2	rs1	011	rd	0101111	AMOMIN.D
10100	aq	rl	rs2	rs1	011	rd	0101111	AMOMAX.D
11000	aq	rl	rs2	rs1	011	rd	0101111	AMOMINU.D
11100	aq	rl	rs2	rs1	011	rd	0101111	AMOMAXU.D

1.3.3、RV64M 指令集

RV32M Standard Extension

0000001	rs2	rs1	000	rd	0110011	MUL
0000001	rs2	rs1	001	rd	0110011	MULH
0000001	rs2	rs1	010	rd	0110011	MULHSU
0000001	rs2	rs1	011	rd	0110011	MULHU
0000001	rs2	rs1	100	rd	0110011	DIV
0000001	rs2	rs1	101	rd	0110011	DIVU
0000001	rs2	rs1	110	rd	0110011	REM
0000001	rs2	rs1	111	rd	0110011	REMU

RV64M Standard Extension (in addition to RV32M)

0000001	rs2	rs1	000	rd	0111011	MULW
0000001	rs2	rs1	100	rd	0111011	DIVW
0000001	rs2	rs1	101	rd	0111011	DIVUW
0000001	rs2	rs1	110	rd	0111011	REMW
0000001	rs2	rs1	111	rd	0111011	REMUW

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 ADDIW RD， 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 ADD R5，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]

AUIPC　RD，　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

ADDI RD，　RS1，　IMM[11:0]

SLTI　RD,　RS 1，　IMM[11:0]

SLTIU RD,　RS1，　IMM[11:0]

……

（2） 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

（2）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)

（3） 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。

本机将 FENCE 和 FENCE.I 指令一视同仁，都作为存储器屏障指令进行操作，当处理器执行 FENCE、FENCE.I 指令时，流水线将直接被阻塞，不会取新的指令，直到 FENCE 指令前发射的指令全部完成，流水线全部排空为止，这样即可保证所有 FENCE 指令被严格执行。

*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)

（3） 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 csrrw rd, 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 csrrs rd, 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 csrrc rd, 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 csrrwi rd, 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。

*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：Trap Supervisor Return，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：Timeout Wait，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: Trap Virtual Memory， 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：Modify PRiVilege， 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： Make eXcutable Readable， 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 0 1 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，Instruction address Misaligned 的编码是 0，故在 MEDELEG 寄存器中设置

BIT0=1，就会将 Instruction Address Misaligned 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（CY IR HPM3 HPM4）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	Exception Code
63	62：0

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

INT	Exception Code	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： Make eXcutable Readable， 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 0 1 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	Exception Code
63	62：0

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

INT	Exception Code	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 RISCV 2019608 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，Instruction address Misaligned 的编码是 0，故在 MEDELEG 寄存器中设置 Bit0=1，就会将 Instruction Address Misaligned 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（Interrupt Vector Table），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：SEPC vs MEPC、 SSTATUS vs MSTATUS 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，PAGE SIZE = 2^12)

2、 从地址 A + VA.VPN[i]*PTESIZE 中取得第页表（Sv39 的 PTESIZE=8）

3、 如果 V=0，或者 R=0 且 W=1，造成页面错误。

4、 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。

5、 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。

6、 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。

7、 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

8、 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。

A、 程序通过执行 ECALL 或者 EBREAK 指令，切换到更高的权限。

B、 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（Write After Read），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：

```
ADD X2, 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、内存访问

PRV464 处理器使用 4K Byte L1 指令缓存和 4K Byte L1 数据缓存， 缓存是写穿式（Write Through）。PRV464 的缓存控制器每次缓存一个对齐的 4K Byte 内存，即从 0x************000 – 0x************FFF 的数据，程序员在访问内存时，应当尽量在对齐的 4K 片内操作，以提高系统运行效率。

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、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。