广义地讲，将处理器运行时影响程序正常运行过程的所有事件都称为“异常”，而不管这些事件是来源于处理器的内部还是外部。

广义“异常(Exception)”常被分为狭义“异常”和“中断(Interrupt)”。

狭义“异常”通常指处理器内核的错误事件，或者由处理器的专有指令生成的事件。后者又称为“软中断”或“内陷(Trap)”。

“中断”通常指来自处理器内核外部的事件。中断由与处理器相连的特定物理信号的电平变化产生。

本书后面所提及的“异常”指狭义的 “异常”。

RISC-V 处理器架构定义了一套相对简单的中断和异常处理机制。通过 CSR 寄存器，处理器内核能够更加方便、灵活地管理异常和中断处理过程。RISC-V 处理器架构还允许处理器设计者根据应用需求定制和扩展中断和异常处理功能。本书将在 4.3 节中详细讨论 RISC-V 处理器的异常和中断处理机制。

4.2　RV32I 指令集

模块化是 RISC-V 指令集的最大特点之一。基本指令集 RV32I 充分体现了 RISC-V 指令集简单、精炼的风格。RV32I 指令集共有 47 条指令，包括算术和逻辑运算、存储器访问，以及分支与跳转等操作。与 RV32I 相比，RV64I、RV128I 仅扩展了 64 位、128 位数据访问指令，其他基本操作指令没有变化。

4.2.1　RV32I 指令

RV32I 指令是能够被处理器内核解码并执行的二进制数。在汇编语言中，用助记符表示指令，以便于编程和理解。下面首先介绍 RV32I 指令格式和助记符，然后详细讨论 RV32I 指令集中的指令。

1. 指令长度

RISC-V 架构支持固定长度指令。

一条指令通常包括两个部分，操作码和操作数。操作码标识该指令的操作类型，操作数则是该指令的操作对象。

RV32I 是 32 位 RISC-V 处理器架构的基础指令集。RV32I 指令集指令长度是 32 位，其中低 7 位是指令的操作码(Opcode)，高 25 位是指令的操作数(Operands)。RV32I 指令结构如图 4.2 所示。

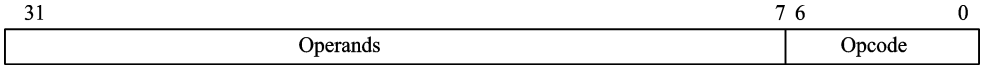

图 4.2　RV32I 指令结构

RV64I 和 RV128I 指令集中的指令长度也是 32 位，其结构与图 4.2 相同。扩展指令集“M”“A”“F”“Q”“D”，以及 CSR 寄存器访问指令的长度也是 32 位，扩展指令集“C”的指令长度是 16 位。

在内存中，32 位指令必须位于 4 字节对齐的边界，16 位指令必须是 2 字节边界对齐。否则，处理器内核运行程序时将无法正确取指令，出现异常错误。

2. 助记符

为了便于记忆和编程，编写汇编程序时用文本格式符号表示指令，即助记符。

RISC-V 汇编语言采用字母联想方式定义指令的助记符。通常，如果描述操作的短语只有一个单词，则取单词的前 3 个字母构成助记符。如果描述操作的短语多于一个单词，则取短语中所有单词的首字母，顺序构成助记符。

RV32I 指令集中包括算术、逻辑和移位 3 种整数运算指令。通常在运算助记符后增加“i”，构成新的助记符，表示源操作数是立即数。如果助记符后附加“u”，则表示操作数是无符号整数。表 4.5 列出了 RV32I 整数计算指令的助记符。其中，助记符后的“(i)”和“(u)”是可选项，分别说明操作数是立即数和无符号整数类型。

表 4.5　RV32I 整数计算指令助记符

功　能	操　作	助记符	解　释
整数计算 (Integer Computation)	addition (immediate)	add(i)	“加法”运算(i,立即数)
	subtract	sub	“减法”运算(i,立即数)
	and (immediate)	and(i)	“与”运算(i,立即数)
	or (immediate)	or(i)	“或”运算(i,立即数)
	xor (immediate)	xor(i)	“异或”运算(i,立即数)
	shift left logical (immediate)	sll(i)	逻辑左移(i,立即数)
	shift right arithmetic(immediate)	sra(i)	算术右移(i,立即数)
	shift right logical(immediate)	srl(i)	逻辑右移(i,立即数)
	load upper immediate	lui	装载立即数到寄存器的高 20 位
	add upper immediate to pc	auipc	把立即数加到程序指针的高 20 位
	set less than immediate unsigned	slt(i)(u)	根据比较结果设置寄存器值

在 RV32I 指令集中，内存访问指令装载(load)和存储(store)支持 32 位(word)、16 位(halfword)和 8 位(byte)三种数据格式。对于 halfword 和 byte 类型数据，load 指令还支持无符号(unsigned)形式，不需要扩展所装载数据的符号位。表 4.6 列出了 RV32I 的不同类型 load 和 store 指令助记符。load 和 store 指令要求指令中内存数据地址按所访问的数据类型对齐。例如，“lw”指令要求数据地址按 4 字节对齐。

表 4.6　RV32I load 和 store 指令助记符

功　能	操　作	助记符	解　释
装载与存储(Load and Store)	load byte (unsigned)	lb(u)	读取 1 字节(8 位,u,无符号)
	load halfword (unsigned)	lh(u)	读取 2 字节(16 位,u,无符号)
	load word	lw	读取 4 字节(32 位)
	store byte	sb	保存 1 字节(8 位)
	store halfword	sh	保存 2 字节(16 位)
	store word	sw	保存 4 字节(32 位)

表 4.7 列出了 RV32I 控制转移指令的助记符。

表 4.7　RV32I 控制转移指令助记符

功　能	操　作	助记符	解　释
控制转移(Control transfer)	branch equal	beq	如果相等则跳转
	branch not equal	bne	如果不等则跳转
	branch less than (unsigned)	blt(u)	如果小于则跳转(u,无符号数)
	branchgreater and equal(unsigned)	bge(u)	如果不小于则跳转(u,无符号数)
	jump and link	jal	可返回跳转,PC=PC+立即数
	jump and link register	jalr	可返回跳转,PC=寄存器中的值

RV32I 指令集包括分支转移(branch)和无条件跳转(jump)两种程序控制转移指令。branch 指令支持相等(equal)、不等(not equal)、大于或等于(greater than or equal)和小于(less than)四种条件判断。branch 指令跳转时,将 12 位立即数乘以 2 的结果与 PC 相加,作为分支跳转的目标地址。

执行 jump 指令时,将 jump 的下一条指令的地址(PC + 4,PC 是当前指令 jump 的指针) 保存到返回地址寄存器 ra,然后跳转到目标地址。处理器从子程序返回到主程序时,从 ra 寄存器读取程序返回的指针。因为 x0 不能更改,如果使用 x0 作为目标寄存器,则实现无条件和无返回跳转。

在控制转移指令中,获取目标指令地址的方式有立即数寻址和寄存器寻址两种。

RV32I 指令集中还包括一些其他(Miscellaneous)指令。如表 4.8 所列,其他指令包括控制状态寄存器访问、运行时环境调用以及同步访问指令。

3. 指令格式

RV32I 指令集包括算术运算、逻辑运算、移位操作、存储访问、跳转和其他指令,不同类型指令的格式有所差别。

表 4.8　RV32I 其他指令助记符

功　能	操　作	助记符	解　释
其他 (Miscellaneous)	fence loads & stores	fence	同步内存访问
	fence. instruction & data	fence. i	同步指令流
	envirmonent break	ebreak	环境断点
	envirmonent call	ecall	环境调用
	control status register read & clear bit (immediate)	csrrc(i)	CSR 寄存器读并清除位(i,立即数)
	control status register read & set bit (immediate)	csrrs(i)	CSR 寄存器读并置位(i,立即数)
	control status register read & write (immediate)	csrrw(i)	CSR 寄存器读写(i,立即数)

(1) 运算指令

图 4.3 是运算指令汇编语句格式。算术运算、逻辑运算和移位操作指令都包含 3 个操作数,目标操作数(寄存器)rd,源操作数 1 和源操作数 2。源操作数 1 是寄存器 rs1,源操作数 2 是寄存器 rs2 或立即数 imm,目标操作数(寄存器)rd 保存运算结果。

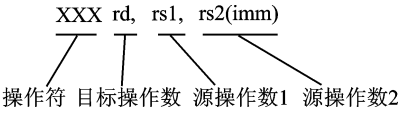

图 4.3　运算指令汇编格式

表 4.9～表 4.11 分别列出了 RV32I 指令集中算术运算、逻辑运算和移位操作的汇编语句示例,并解释了语句的执行结果。

表 4.9　算术运算汇编语句示例

指　令	示　例	操　作
加法	add t0,t1,t2	t0=t1+t2;
减法	sub t0,t1,t2	t0=t1-t2;
立即数加法	addi t0,t1,200	t0=t1+200;

在 RV32I 算术运算指令中,立即数的数值范围是 imm[11:0],12 位。

表 4.10　逻辑运算汇编语句示例

指　令	示　例	操　作
与	and t0,t1,t2	t0=t1&t2 ;按位与
或	or t0,t1,t2	t0=t1|t2;按位或
异或	xor t0,t1,t2	t0=t1^t2;按位异或
立即数与	andi t0,t1,200	t0=t1&200;按位与
立即数或	ori t0,t1,200	t0=t1|200;按位或
立即数异或	xori t0,t1,200	t0=t1^200;按位异或

在 RV32I 逻辑运算指令中,立即数的数值范围是 imm[11:0],12 位。

表 4.11　移位操作汇编语句示例

指　令	示　例	操　作
逻辑左移	sll t0,t1,t2	t0=t1<<t2;低位补 0
逻辑右移	srl t0,t1,t2	t0=t1>>t2;高位补 0
算术右移	sar t0,t1,t2	t0=t1>>t2;负数高位补 1,正数高位补 0
立即数逻辑左移	slli t0,t1,10	t0=t1<<10;低位补 0
立即数逻辑右移	srli t0,t1,10	t0=t1>>10;高位补 0
立即数算术右移	srai t0,t1,10	t0=t1>>10;负数高位补 1,正数高位补 0

在 RV32I 移位操作指令中,立即数的数值范围是 imm[4:0],5 位。

(2) 数据装载和存储指令

装载(load)指令将内存中数据读到寄存器,保存(store)指令将寄存器中数据保存到内存中。如图 4.4 所示,一般情况下,装载和保存指令有 2 个操作数,数据寄存器(reg)和内存地址寄存器(reg2)。处理器内核从地址寄存器(reg2)获取内存基地址,加上偏移值(offset)后得到内存中的数据地址,然后从数据地址处读取数据,并写入数据寄存器中。

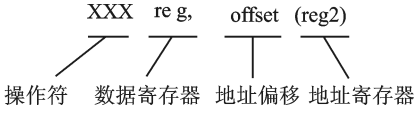

图 4.4　内存访问指令汇编格式

表 4.12 列出了 load 和 store 指令常用的汇编语句形式,并解释了语句的执行结果。

表 4.12　load 和 store 汇编语句示例

指　令	示　例	操　作
装载字(32 位)	lw t0, 50(t1)	t0=memory[t1+50]; 将起始地址 t1+50 内存中 4 字节数读入 t0
装载半字(16 位)	lh t0, 50(t1)	t0=memory[t1+50]; 将起始地址 t1+50 内存中 2 字节数读入 t0 低 16 位;正数,高 16 位补 0;负数,高 16 位补 1
装载半字(无符号)	lhu t0, 50(t1)	t0=memory[t1+50]; 将起始地址 t1+50 内存中 2 字节数读入 t0 低 16 位;高 16 位补 0
装载字节(8 位)	lb t0, 50(t1)	t0=memory[t1+50]; 将地址 t1+50 内存中 1 字节数读入 t0;正数,高 24 位补 0;负数,高 24 位补 1
装载字节(无符号)	lbu t0, 50(t1)	t0=memory[t1+50]; 将地址 t1+50 内存中 1 字节数读入 t0;高 24 位补 0
写字	sw t0, 50(t1)	memory[t1+50]=t0; 将 t0 中数据写入起始地址 t1+50 的内存中
写半字	sh t0, 50(t1)	memory[t1+50]=t0; 将 t0 中数据的低 16 位写入起始地址 t1+50 的内存中
写字节	sb t0, 50(t1)	memory[t1+50]=t0; 将 t0 中数据的低 8 位写入地址 t1+50 的内存中

在 RV32I load 和 store 指令中,偏移量立即数的数值范围是 offset[11:0],12 位。

在 RV32I 指令集中,“lui”指令将立即数装载到目标寄存器的高 20 位,目标寄存器的低 12 位置 0。“auipc”指令将立即数加到 pc 的高 20 位。以下是指令的汇编语句示例。

```
lui t0, 0x12345          //执行后 t0 = 0x1235000
auipc t0, 0x12345        //执行后 t0 = 0x12345000 + pc
```

“lui”和“auipc”指令中立即数的范围是 imm[19:0],20 位。

(3) 跳转指令

跳转指令包括分支跳转和无条件跳转两类。

分支跳转指令包含 3 个操作数,其结构如图 4.5 所示。处理器比较数据寄存器 rs1 和 rs2 中的数值,如果满足操作符中的条件,则程序指针指向当前 PC 加上立即数 imm 后所得数值的位置。

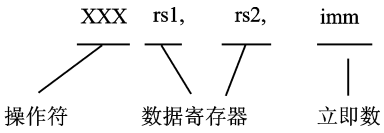

图 4.5　分支跳转指令结构

分支跳转指令汇编语句示例如表 4.13 所列。

表 4.13　分支跳转指令汇编语句示例

指　令	示　例	操　作
相等条件分支	beq t0,t1,200	if(t0＝＝t1) pc＝pc＋200;跳转
不等条件分支	bne t0,t1,200	if(t0 !＝t1) pc＝pc＋200;跳转
小于条件分支	blt t0,t1,200	if(t0＜t1) pc＝pc＋200;跳转
大于或等于条件分支	bge t0,t1,200	if(t0＞＝t1) pc＝pc＋200;跳转
小于条件分支(无符号)	bltu t0,t1,200	if(t0＜t1) pc＝pc＋200;无符号数比较
大于或等于分支(无符号)	bgeu t0,t1,200	if(t0＞＝t1) pc＝pc＋200;无符号数比较

如表 4.14 所列,无条件跳转指令包含两个操作数,返回指针寄存器(ra)和跳转目标地址。

对于指令 jal,跳转目标地址是语句中的立即数与当前 pc 值之和。立即数的范围是 imm[20:1],20 位。

对于指令 jalr,跳转的目标地址为地址寄存器(t0)中的值与偏移量之和。偏移量的数值范围是 offset[11:0],12 位。

表 4.14　无条件跳转指令汇编语句示例

指　令	示　例	操　作
带返回跳转	jal ra, 200	ra＝ pc＋4;保存下一条指令指针。 pc＝pc＋200; pc 相对跳转
带返回跳转(寄存器)	jalr ra, 200(t0)	ra＝ pc＋4;保存下一条指令指针。 pc＝t0＋200; 寄存器相对跳转

(4) CSR 操作指令

RV32I 用专属指令访问 CSR 寄存器。如图 4.6 所示,CSR 操作指令包含 3 个操作数,目标操作数(寄存器)rd、CSR 寄存器 csr 和源操作数(rs 或 imm)。CSR 立即数操作指令的源操作数是立即数 imm,CSR 寄存器操作指令的源操作数是寄存器 rs。表 4.15 列出了 CSR 操作指令汇编语句示例。

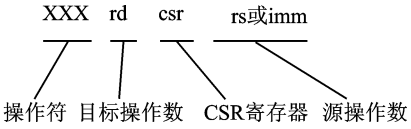

图 4.6 CSR 操作指令汇编格式

表 4.15 CSR 操作指令汇编语句示例

指 令	示 例	操 作
先读后清除 CSR	csrrc t0, 0x123,t1	t0=[0x123]; [0x123]=t0 & (~t1); 把 CSR 寄存器 0x123 中的值读入 t0,然后用计算结果更新 0x123 中的值
先读后置位 CSR	csrrs t0,0x123,t1	t0=[0x123];[0x123]=t0 | t1; 把 CSR 寄存器 0x123 中的值读入 t0,然后用计算结果更新 0x123 中的值
先读后写 CSR	csrrw t0,0x123,t1	t0=[0x123];[0x123]=t1; 把 CSR 寄存器 0x123 中的值读入 t0,然后将 t1 中的值写入 0x123 中
立即数先读后清除 CSR	csrrcit0, 0x123,20	t0=[0x123]; [0x123]=t0 & (~20); 把 CSR 寄存器 0x123 中的值读入 t0,然后用计算结果更新 0x123 中的值
立即数先读后置位 CSR	csrrsi t0,0x123,20	t0=[0x123];[0x123]=t0 | 20; 把 CSR 寄存器 0x123 中的值读入 t0,然后用计算结果更新 0x123 中的值
立即数先读后写 CSR	csrrwi t0,0x123,20	t0=[0x123];[0x123]=20; 把 CSR 寄存器 0x123 中的值读入 t0,然后将立即数 20 写入 0x123 中

在 CSR 立即数操作指令中,立即数的取值范围是 imm[4:0],5 位。

4.2.2 寻址方式

寻址方式是处理器执行指令时获取数据地址,或者下一条指令地址的方式。RISC-V 处理器支持立即数寻址、寄存器寻址、寄存器间接寻址和程序计数(PC)相对寻址 4 种寻址方式。

1. 立即数寻址

立即数寻址是最简单直接的寻址方式,指令中直接以常数作为操作数。在 RISC-V 汇编语句中,通常将字母“i”置于操作符末,表示立即数操作指令。例如,加法运算“add”操作的两个源操作数都是寄存器,而“addi”操作的一个源操作数是寄存器,另一个操作数是立即数。

RV32I 不同类型指令立即数的取值范围有所差别。例如，操作“lui”的立即数范围是 20 位，“addi”和“andi”运算的立即数范围是 12 位。

使用 RV32I 指令组合，可以把任意 32 位整数装载到寄存器中。

例如，通过下列两条指令，能够将 32 位数 0x12345678 装载到寄存器 t0 中。

```
1  lui t0, 0x12345          // t0 = 0x12345000
2  addi t0, t0, 0x678       // t0 = 0x12345678
```

第 1 行，“lui” 将一个 20 位常量加载到寄存器 t0 的第 12 位到第 31 位，即 t0[31:12]，最右边的 12 位 t0[11:0]填充 0。

第 2 行，“addi” 将 12 位立即数加到 t0 的第 0 位到第 11 位，即 t0[11:0]。

在装载和存储指令中，地址偏移量“offset”也是立即数，其取值范围是 12 位，即 offset[11:0]。

2. 寄存器寻址

寄存器寻址指令的源操作数是寄存器，从寄存器读取数据，并把结果保存到寄存器中。在 RV32I 指令集中，“add”、“sub”、“and”、“or”和“xor”等运算指令的所有操作数都是寄存器，是典型的寄存器寻址指令。表 4.9～表 4.11 中，末字母非“i”的指令是寄存器寻址指令。

3. 寄存器间接寻址

寄存器间接寻址指令以寄存器的数值作为内存地址（存储地址的寄存器又称为地址寄存器），从该内存地址所指定的存储单元读取数据，或者将数据写入到内存地址所指定的存储单元。如果指令中有偏移量“offset”，则存储单元的地址是地址寄存器的数值与“offset”之和。

下面通过示例说明间接寻址指令的操作。

表 4.16 列出了 0x800000 至 0x80001f 内存段中每个字节的数据。其中，第 1 列是 4 字节对齐地址，第 1 行是各字节的偏移地址，其他部分是相应内存单元中的数据。

表 4.16　内存数据

内存地址	0	1	2	3
0x800000～0x800003	0x00	0x10	0x20	0x30
0x800004～0x800007	0x04	0x14	0x24	0x34
0x800008～0x80000b	0x08	0x18	0x28	0x38
0x80000c～0x80000f	0x0c	0x1c	0x2c	0x3c
0x800010～0x800013	0x10	0x20	0x30	0x40
0x800014～0x800017	0x14	0x24	0x34	0x44
0x800018～0x80001b	0x18	0x28	0x38	0x48
0x80001c～0x80001f	0x1c	0x2c	0x3c	0x4c

RISC-V 仅支持小端(little-endian)存储格式。在字或半字数据中,数据中低位字节存放在内存的低地址。

如果 t1 寄存器中初始数值为 0x800000,则下列第 1 条、第 2 条和第 3 条语句执行后 t0 中的数值分别为 0x30201000、0x38281808 和 0x28。

```
1  lw t0, (t1)          // t0 = 0x30201000
2  lw  t0, 8(t1)        // t0 = 0x38281808
3  lb  t0, 10(t1)       // t0 = 0x28
```

4. PC 相对寻址

PC 相对寻址以当前 PC 值为基地址,以指令中操作数为偏移量,两者相加后得到新的内存地址。处理器从新的内存地址读取数据,或跳转到新的程序地址。

RISC-V 用 PC 相对寻址实现条件分支和无条件跳转。在下列汇编程序中,第 4 行语句中的“end”汇编后转成立即数 12,第 6 行中的“start”汇编后转成立即数 −16,都是内存中当前指令到目标位置的距离,地址增加的方向为正,地址减小的方向为负。条件分支指令立即数范围是±4 KB。

```
1  start:
2    add t0, t0, t1
3    ld t2, 0(t0)
4    bne t2, t3, end          // if(t2!= t3) PC = PC + 12
5    addi t4, t4, 1
6    beq t0, t0, start        // PC = PC - 16
7  end:
```

下列两条是无条件跳转指令,语句 1 中“jal”跳转的范围是±1 MB,语句 2 中“jalr”跳转的范围是±2 GB。

```
1  jal ra, dst                // PC = PC + dst, ra = PC + 4
2  jalr ra, 0(t0)             // PC = t0, ra = PC + 4
```

4.3 RISC-V 异常和中断处理

异常和中断处理是处理器中不可缺少而又复杂的功能,使处理器在正常程序运行过程中能够响应和处理异常事件或中断请求。图 4.7 是处理器响应异常事件过程的示意图。当异常事件发生时,处理器暂停执行当前主程序,从暂停处跳转到异常事件处理程序入口,执行异常处理程序。异常处理程序结束后,返回主程序暂停处的下一条指令,然后继续执行主程序。

RISC-V 特权架构定义了 RISC-V 处理器异常处理机制。在 RISC-V 特权架构