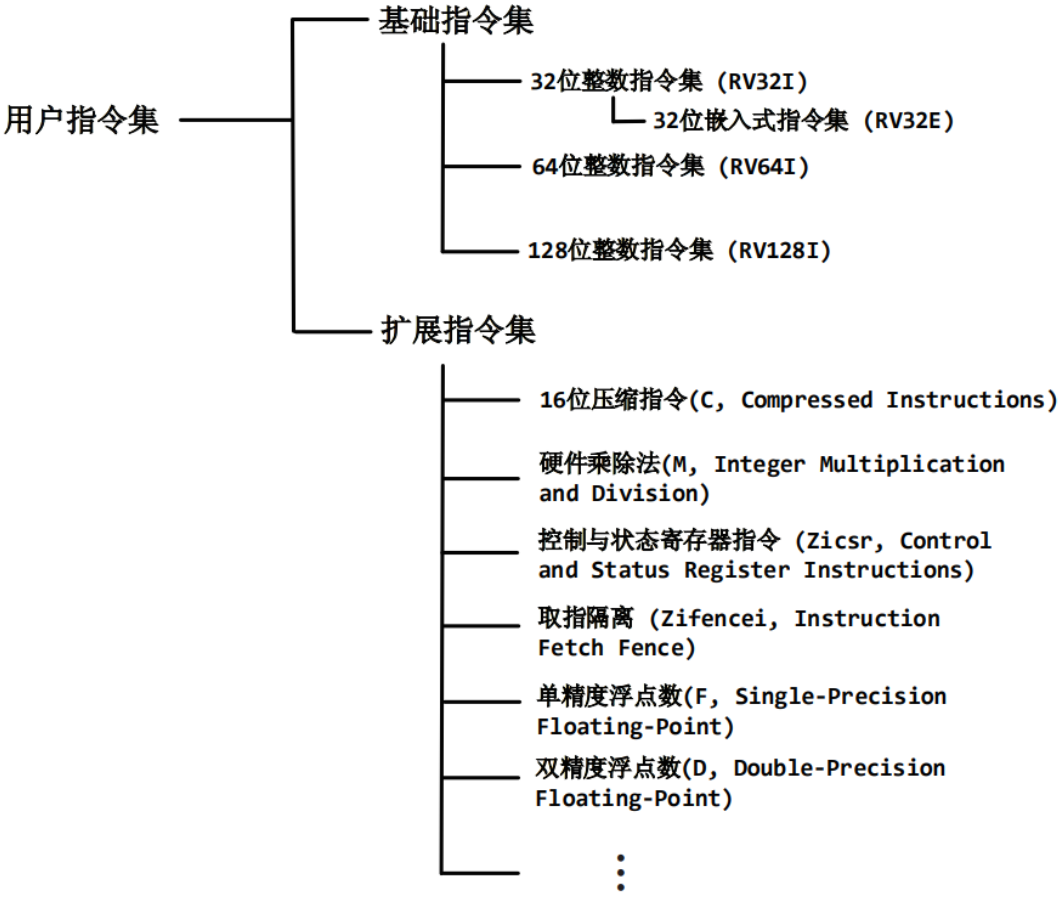
RISC-V ISA

RISC-V基础指令集

RISV-V的官方标准主要分成两个部分：用户指令集（User-Level Instruction Set Architecture）与特权架构（Privileged Architecture）
用户指令集分类如图所示



RISC-V地址空间

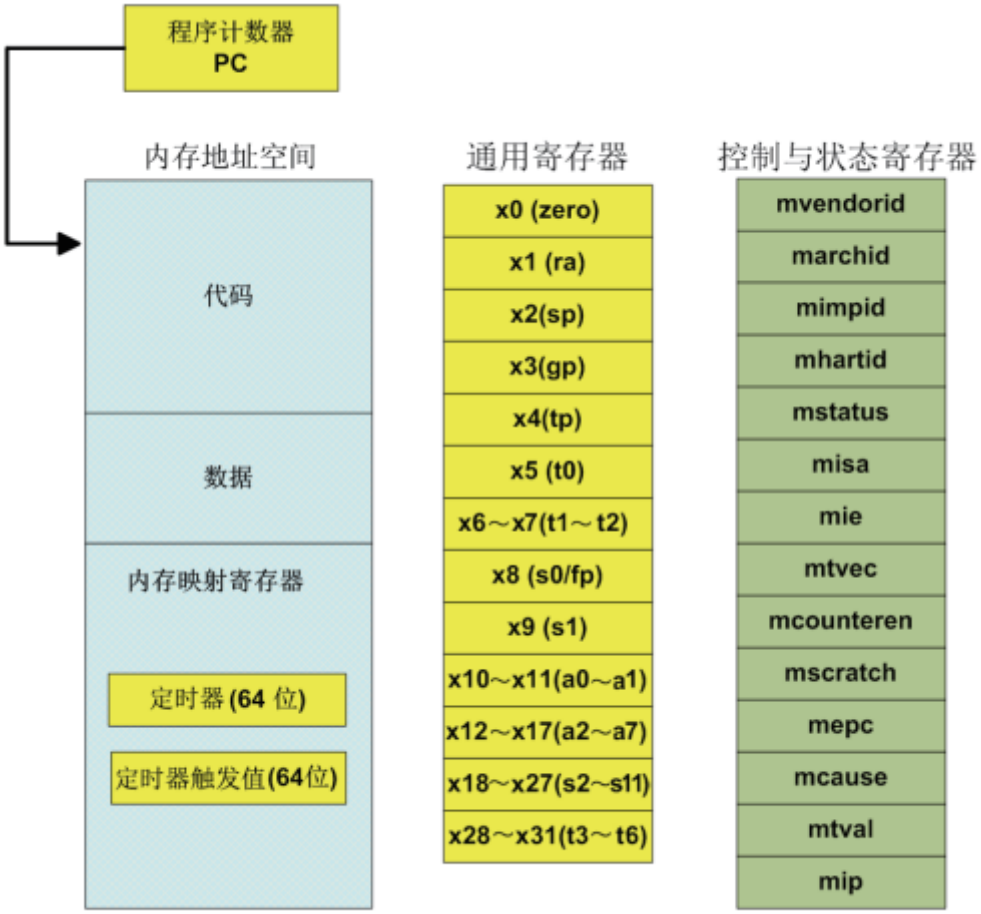


图3-4　RISC-V地址空间

注意RV32I指令集包括32个通用寄存器，而RV32E只有16个这样的寄存器

寄存器约定

表 3-1　函数调用约定的寄存器分配

寄存器名称	汇编名称	功能描述	调用返回后其值是否会保证不变
x0	zero	零寄存器	未定义
x1	ra	返回地址	否
x2	sp	栈指针	是
x3	gp	全局指针	未定义
x4	tp	线程指针	未定义
x5	t0	临时寄存器，或者用作替代链接寄存器（见后续章节详述）	否
x6	t1	临时寄存器	否
x7	t2	临时寄存器	否
x8	s0/fp	该寄存器需要被调函数予以保存，或也可用作调用栈的帧指针	是
x9	s1	该寄存器需要被调函数予以保存	是
x10~x11	a0~ a1	函数参数或返回值	否
x12~x17	a2~a7	函数参数	否
x18~x27	s2~s11	该寄存器需要被调函数予以保存	是
x28~x31	t3~t6	临时寄存器	否

RV32I指令格式

- RV32I的基本指令格式有4中，分别为寄存器类型(R-TYPE),短立即数类型(I-TYPE),内存存储类型(S-TYPE),高位立即数类型(U-TYPE)
- 为了方便跳转，RV32I通过S-TYPE衍生出来B-TYPE(Branch，条件跳转)和通过U-TYPE衍生出来J-TYPE（Jump,无条件跳转）
- 上面这些格式，除了R-TYPE外，其他格式都需要将最高位（第31位）作为符号扩展，来产生一个32位的立即数，作为指令的操作数

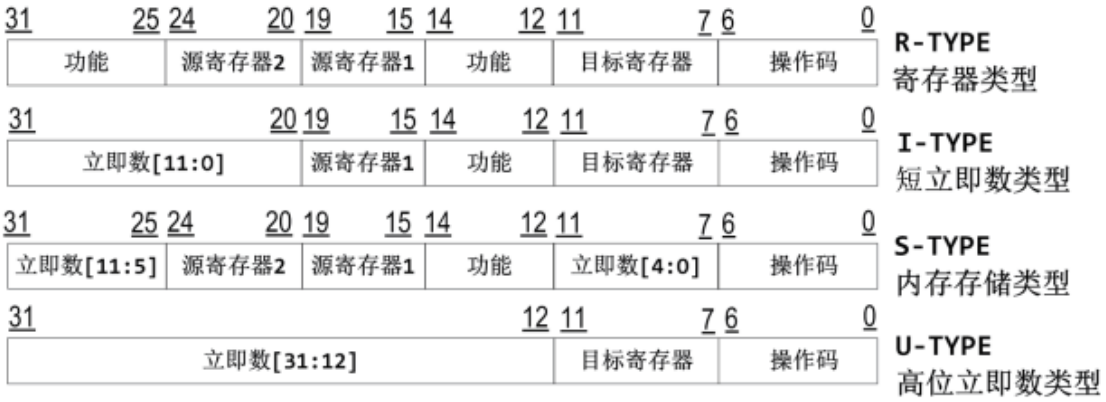


图3-5　RV32I基本指令格式

RV32I指令解析

31 25	24 20	19 15	14 12	11 7	6 0	
imm[31:12]				rd	0110111	U lui
imm[31:12]				rd	0010111	U auipc
imm[20|10:1|11|19:12]				rd	1101111	J jal
imm[11:0]		rs1	000	rd	1100111	I jalr
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	B beq
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	B bne
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	B blt
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	B bge
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	B bltu
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	B bgeu
imm[11:0]		rs1	000	rd	0000011	I lb
imm[11:0]		rs1	001	rd	0000011	I lh
imm[11:0]		rs1	010	rd	0000011	I lw
imm[11:0]		rs1	100	rd	0000011	I lbu
imm[11:0]		rs1	101	rd	0000011	I lhu
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	S sb
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	S sh
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	S sw
imm[11:0]		rs1	000	rd	0010011	I addi
imm[11:0]		rs1	010	rd	0010011	I slti
imm[11:0]		rs1	011	rd	0010011	I sltiu
imm[11:0]		rs1	100	rd	0010011	I xori
imm[11:0]		rs1	110	rd	0010011	I ori
imm[11:0]		rs1	111	rd	0010011	I andi
0000000	shamt	rs1	001	rd	0010011	I slli
0000000	shamt	rs1	101	rd	0010011	I srli
0100000	shamt	rs1	101	rd	0010011	I srai
0000000	rs2	rs1	000	rd	0110011	R add
0100000	rs2	rs1	000	rd	0110011	R sub
0000000	rs2	rs1	001	rd	0110011	R sll
0000000	rs2	rs1	010	rd	0110011	R slt
0000000	rs2	rs1	011	rd	0110011	Rsltu
0000000	rs2	rs1	100	rd	0110011	R xor
0000000	rs2	rs1	101	rd	0110011	R srl
0100000	rs2	rs1	101	rd	0110011	R sra
0000000	rs2	rs1	110	rd	0110011	R or
0000000	rs2	rs1	111	rd	0110011	R and
0000 pred	succ	00000	000	00000	0001111	I fence
0000 0000	0000	00000	001	00000	0001111	I fence.i
000000000000		00000	00	00000	1110011	I ecall
000000000000		00000	000	00000	1110011	I ebreak
csr		rs1	001	rd	1110011	I csrrw
csr		rs1	010	rd	1110011	I csrrs
csr		rs1	011	rd	1110011	I csrrc
csr		zimm	101	rd	1110011	I csrrwi
csr		zimm	110	rd	1110011	I cssrrsi
csr		zimm	111	rd	1110011	I csrrci

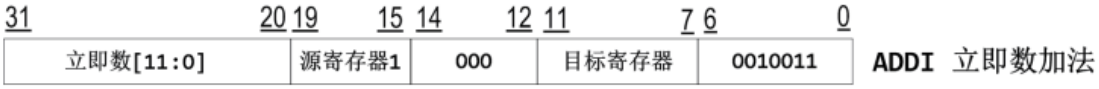
RV32I算术与逻辑指令

add(Add) R-type

- 使用方法: add rd,rs1,rs2
- 实际操作：x[rd]=x[rs1]+x[rs2]
- 实际含义:把寄存器x[rs2]加到寄存器x[rs1]上，结果写入x[rd],忽略算术溢出
- 二进制串:

addi(Add Immediate) I-type

- 使用方法: addi rd,rs1,immediate
- 实际操作: x[rd] = x[rs1] + sext(immediate)
- 实际含义: 把符号位扩展的立即数加到寄存器x[rs1]上，结果写入x[rd],忽略算术溢出
- 二进制串:

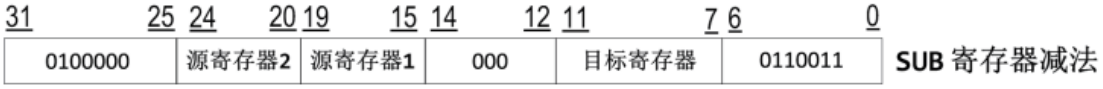


- 注意事项

  1.算术结果是否溢出无法通过这个命令来判断，需要比较和跳转指令实现
  2.immediate的范围为-2^11到2^11-1,即-2024 ~ 2023

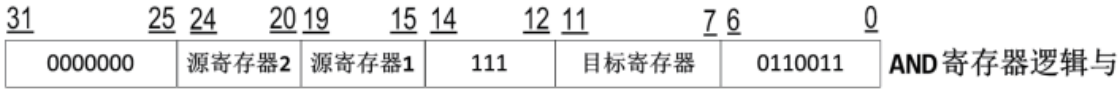
sub(substraction) R-type

- 使用方法: sub rd,rs1,rs2
- 实际操作：x[rd] = x[rs1]-x[rs2]
- 实际含义:x[rs1]减去x[rs2]，结果写入x[rd].忽略算术溢出
- 二进制串:



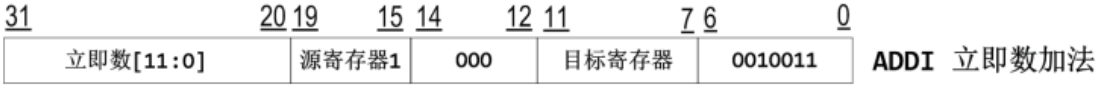
and(And) R-TYPE

- 使用方法：and rd,rs1,rs2
- 实际操作：x[rd] = x[rs1]&x[rs2]
- 实际含义：x[rs1]与x[rs2]与运算
- 二进制串:



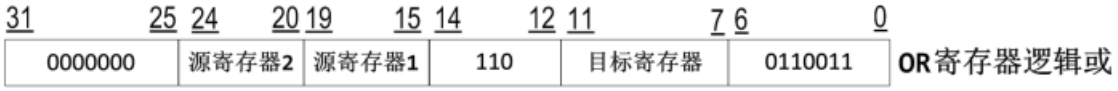
andi(And immediate) I-TYPE

- 使用方法：andi rd,rs1,immediate
- 实际操作: x[rd] = x[rs1]&sext(immediate)
- 二进制串:



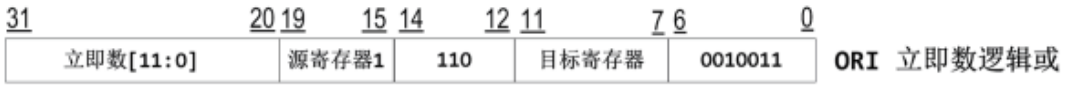
or(OR) R-TYPE

- 使用方法：or rd,rs1,rs2
- 实际操作：x[rd] = x[rs1]|x[rs2]
- 实际含义：x[rs1]与x[rs2]或运算
- 二进制串：



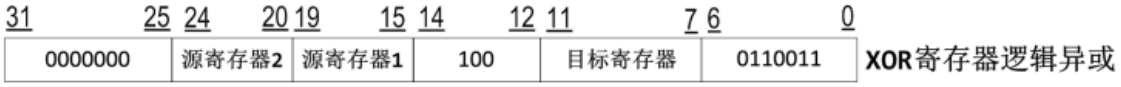
ori(OR immediate) I-TYPE

- 使用方法：ori rd,rs1,immediate
- 实际操作: x[rd] = x[rs1]|sext(immediate)
- 二进制串:



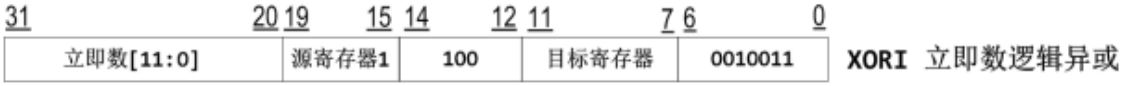
xor(Exclusive-OR) R-TYPE

- 使用方法：or rd,rs1,rs2
- 实际操作：x[rd] = x[rs1]^x[rs2]
- 实际含义：x[rs1]与x[rs2]异或运算
- 二进制串：



xori(OR immediate) I-TYPE

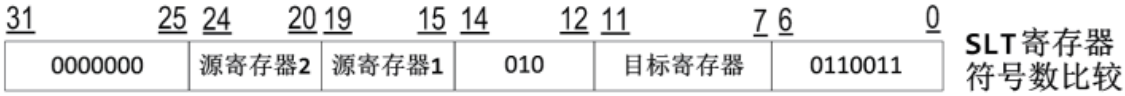
- 使用方法：xori rd,rs1,immediate
- 实际操作: x[rd] = x[rs1]^sext(immediate)
- 二进制串:



- 注意事项:RV32I无NOT指令，可以通过XORI来实现(只需要将XORI指令中的立即数为全1即可)

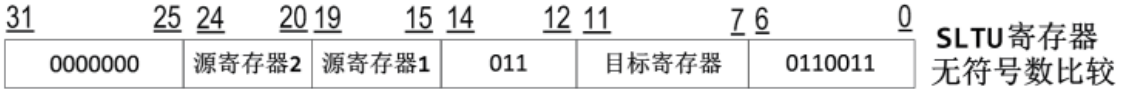
slt(Set if Less Than) R-TYPE

- 使用方法: slt rd,rs1,rs2
- 实际操作: x[rd] = (x[rs1] < x[rs2]))
- 实际含义: 比较x[rs1]和x[rs2],若x[rs1]更小，写入1，否则写入0
- 二进制串:



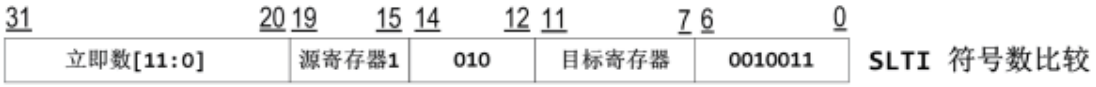
sltu(Set if Less Than,Unsigned) R-TYPE

- 使用方法: slt rd,rs1,rs2
- 实际操作: x[rd] = (x[rs1] < x[rs2]))(Unsigned)
- 实际含义: 比较x[rs1]和x[rs2],比较时视为无符号数,若x[rs1]更小，写入1，否则写入0
- 二进制串:



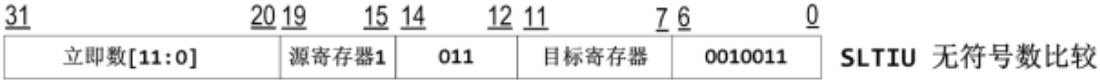
slti(Set if Less Than Immediate) I-TYPE

- 使用方法: slt rd,rs1,immediate
- 实际操作: x[rd] = (x[rs1] < sext(immediate))
- 实际含义: 比较x[rs1]和有符号扩展的immdediate,若x[rs1]更小，写入1，否则写入0
- 二进制串:



sltiu(Set if Less Than Immediate,Unsigned) I-TYPE

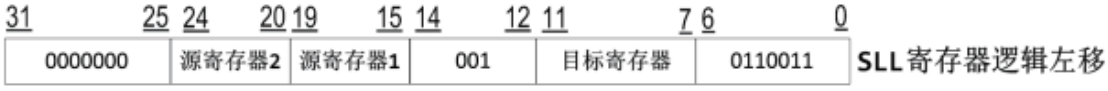
- 使用方法: slt rd,rs1,immediate
- 实际操作: x[rd] = (x[rs1] < sext(immediate))
- 实际含义: 比较x[rs1]和有符号扩展的immdediate,比较时视为无符号数,若x[rs1]更小，写入1，否则写入0
- 二进制串:



- 注意事项:sltiu和slti的区别是第12-14位

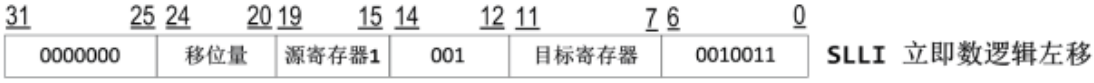
sll(Shift Left Logical) R-TYPE

- 使用方法: sll rd,rs1,rs2
- 实际操作: x[rd] = x[rs1] << x[rs2]
- 实际含义: 把寄存器x[rs1]左移x[rs2]位，空出的位置填入0.x[rs2]的低5位代表移动位数，高位被忽略
- 二进制串:



slli(Shift Left Logical Immediate) I-TYPE

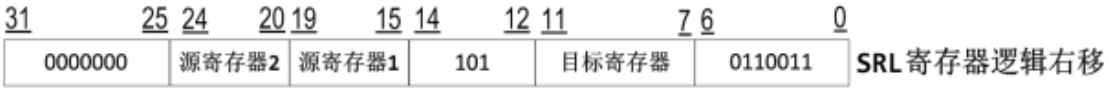
- 使用方法: sll rd,rs1,shamt
- 实际操作: x[rd] = x[rs1] << shamt
- 实际含义: 把寄存器x[rs1]左移shamt位，空出的位置填入0.
- 二进制串:



- 注意事项:对于RV32I,仅当shamt[5]=0时，指令才是有效的

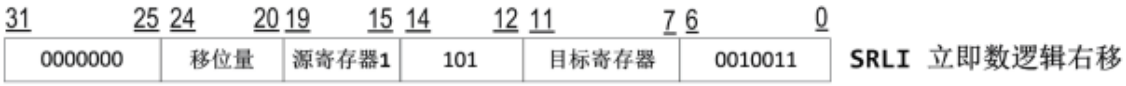
srl(Shift Right Logical) R-TYPE

- 使用方法: srl rd,rs1,rs2
- 实际操作: x[rd] = x[rs1] >> x[rs2]
- 实际含义: 把寄存器x[rs1]右移x[rs2]位，空出的位置填入0.x[rs2]的低5位代表移动位数，高位被忽略
- 二进制串:



srli(Shift Left Logical Immediate) I-TYPE

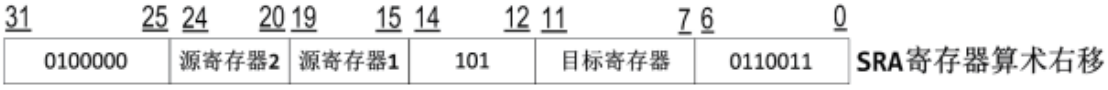
- 使用方法: srl rd,rs1,shamt
- 实际操作: x[rd] = x[rs1] >> shamt
- 实际含义: 把寄存器x[rs1]右移shamt位，空出的位置填入0.
- 二进制串:



- 注意事项:对于RV32I,仅当shamt[5]=0时，指令才是有效的

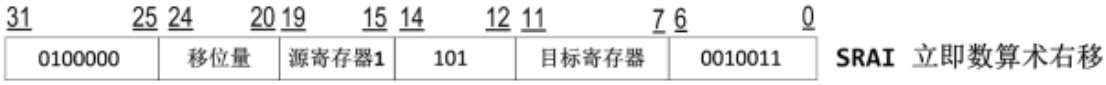
sra(Shift Right Arithmetic) R-TYPE

- 使用方法: sra rd,rs1,rs2
- 实际操作: x[rd] = x[rs1] >>(s) x[rs2]
- 实际含义: 把寄存器x[rs1]右移x[rs2]位，空出的位置填入x[rs1]的最高位.x[rs2]的低5位代表移动位数，高位被忽略
- 二进制串:



srai(Shift Left Arithmetic Immediate) I-TYPE

- 使用方法: srai rd,rs1,shamt
- 实际操作: x[rd] = x[rs1] >>(s) shamt
- 实际含义: 把寄存器x[rs1]右移shamt位，空出的位置填入x[rs1]的最高位.
- 二进制串:



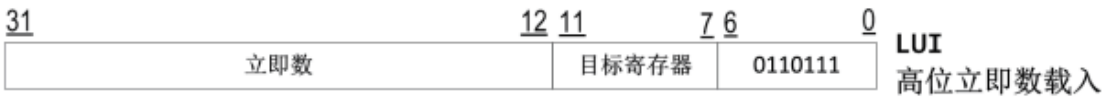
- 注意事项:

  1.对于RV32I,仅当shamt[5]=0时，指令才是有效的

  2.对比srai与srli的区别可知，只有第30位不同，硬件通过判断此位加以区分

lui(Load Upper Immediate) U-TYPE

- 使用方法: lui rd,immediate
- 实际操作: x[rd] = sext(immediate[31:12] << 12)
- 实际含义: 将符号位扩展的20位立即数immediate左移12位，并且将低12位清零，写入x[rd]中
- 二进制串:



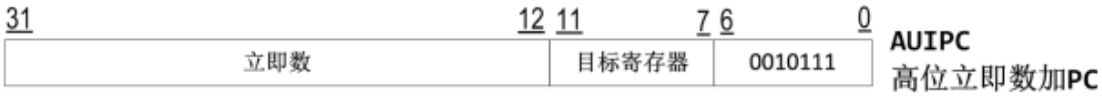
- 注意事项:

  1.通过对之前指令的格式观察，U-TYPE指令中的立即数有20位，而I-TYPE指令中的立即数有12位，32位立即数可以通过一条U-TYPE指令和一条I-TYPE指令来联合构建(LUI+ADDI)

  2.在使用addi添加后12位时需要考虑bit11，如果为1会进行有符号扩展

auipc(Add Upper Immediate to PC) U-TYPE

- 使用方法: auipc rd
- 实际操作: x[rd] = pc + sext(immediate[31:12] << 12)
- 实际含义: 把符号位扩展的20位立即数(左移12位)加到pc上，结果写入x[rd]
- 二进制串:



- 注意事项:

  1.该指令的好处是可以快速向寄存器中载入PC相对寻址的字节，而不需要另外的寄存器作为中介

控制转移指令

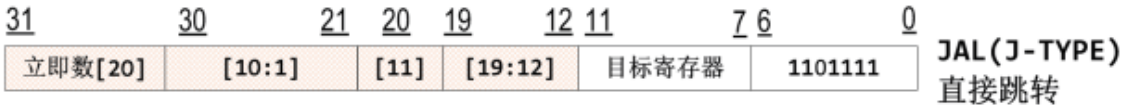
RISC-V中的控制转移指令(Control Transfer Instruction)主要包括两类

> (1) 无条件跳转（Unconditional Jump）
>
> (2) 有条件跳转 （conditional Branches）

下面先叙述无条件跳转，在叙述有条件跳转

jal(Jump and Link) J-TYPE

- 使用方法: jal rd,offset
- 实际操作: x[rd] = pc+4;pc += sext(offset)
- 实际含义: 把下一条指令的地址(pc+4)存入寄存器x[rd],然后把pc设置为当前值加上符号位扩展的offset。rd默认为x1
- 二进制串:



- 注意事项:

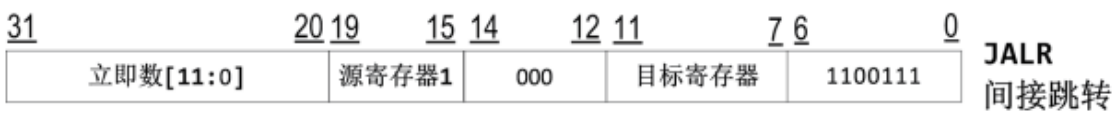
  1.将J-TYPE和U-TYPE比较可知,可以发现除了立即数的某些位做了位置调整以外，其他都没有变化。

  2.JAL为短跳转指令，跳转至PC+— 1MB的地址范围内

  3.作为过程调用: jal x1,ProcedureLabel

jalr(Jump and Link Register) I-TYPE

- 使用方法: jalr rd,offset(rs1)
- 实际操作: t = pc+4;pc = (x[rs1]+sext(offset))&~1;x[rd]=t
- 实际含义: 把pc设置为x[rs1]+sign-extend(offset),把计算出的地址的最低有效位设置为0，并将原pc+4的值写入f[rd].rd默认为x1
- 二进制串:



- 注意事项:

  1.可以通过如下指令序列将JALR指令和LUI/AUIPC指令相结合，可以实现全地址空间的跳转:

> lui ra,立即数(20位)
>
> jalr ra,ra,立即数(12位)
>
> 或者
>
> auipc ra,立即数(20位)
>
> jalr ra,ra,立即数(12位)

2.用于过程调用结束后的返回: jalr x0,0(x1)

3.RISC-V指令集本身没有对JAL或JALR中目标寄存器的取值做出限制,JAL/JALR常用的目标寄存器有x1(ra,返回地址)和x5(t0，代替链接寄存器).具体解释:

表 3-2　JALR 指令的 RAS 操作

目标寄存器(rd)	源寄存器(rs1)	rd ?= rs1	RAS操作	备　注
(rd ≠ x1) && (rd ≠ x5)	(rs1 ≠ x1) && (rs1 ≠ x5)	无关	无	非函数调用指令，亦非调用返回指令

目标寄存器(rd)	源寄存器(rs1)	rd ?= rs1	RAS操作	备　注
(rd ≠ x1) && (rd ≠ x5)	(rs1 = x1) || (rs1 = x5)	无关	返回地址出栈	调用返回指令
(rd = x1) || (rd = x5)	(rs1 ≠ x1) && (rs1 ≠ x5)	无关	返回地址入栈	函数调用指令
(rd = x1) || (rd = x5)	(rs1 = x1) || (rs1 = x5)	(rd ≠ rs1)	在同一指令周期中，既出栈又入栈	协程
(rd = x1) || (rd = x5)	(rs1 = x1) || (rs1 = x5)	(rd = rs1)	返回地址入栈	宏操作合并

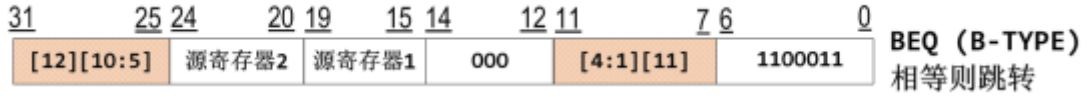
有条件跳转(以下的指令的偏移地址范围为 PC+-4KB)
总览:

表 3-3　有条件比较跳转指令

有条件跳转指令	跳转条件	备　注
BEQ	(源寄存器 1) = (源寄存器 2)	—
BNE	(源寄存器 1) ≠ (源寄存器 2)	—
BLT	(源寄存器 1) < (源寄存器 2)	有符号数比较
BLTU	(源寄存器 1) < (源寄存器 2)	无符号数比较
BGE	(源寄存器 1) ≥ (源寄存器 2)	有符号数比较
BGEU	(源寄存器 1) ≥ (源寄存器 2)	无符号数比较

beq(Branch if Equal) B-TYPE

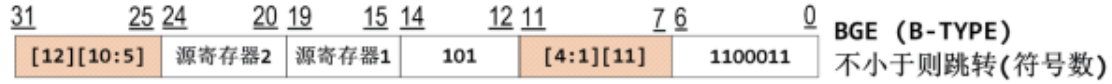
- 使用方法: beq rs1,rs2,offset
- 实际操作: if(rs1 == rs2) pc+= sext(offset)
- 实际含义: 若寄存器x[rs1]和寄存器x[rs2]的值相等，把pc的值设为当前值加上符号位扩展的偏移offset
- 二进制串:



- 注意事项:衍生出来的伪指令（Pesudoinstruction）beqz=> beq rs1,x0,offset

bge(Branch if Greater Than or Equal) B-TYPE

- 使用方法: bge rs1,rs2,offset
- 实际操作: if(rs1 >= rs2) pc += sext(offset)
- 实际含义: 若寄存器x[rs1]大于等于寄存器x[rs2]的值，把pc的值设为当前值加上符号位扩展的偏移offset
- 二进制串:

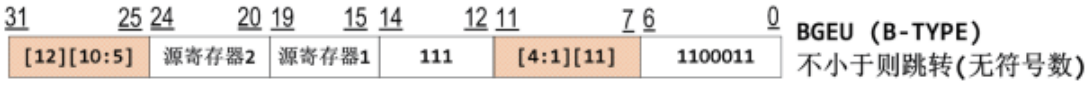


- 注意事项:

> 衍生出来的伪指令（Pesudoinstruction）
>
> bgez=> bge rs1,x0,offset
>
> ble => bge rs2,rs1,offset
>
> blez=> bge x0,rs2,*offset

bgeu(Branch if Greater Than or Equal,Unsigned) B-TYPE

- 使用方法: bgeu rs1,rs2,offset
- 实际操作: if(rs1 >= rs2)(Unsigned) pc+=sext(offset)
- 实际含义: 若寄存器x[rs1]大于等于寄存器x[rs2]的值(均视为无符号数)，把pc的值设为当前值加上符号位扩展的偏移offset
- 二进制串:

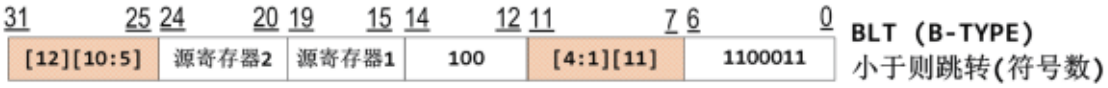


- 注意事项:

> 衍生出来的伪指令（Pesudoinstruction）
>
> bleu=> bgeu rs2,rs1,offset

blt(Branch if Less Than) B-TYPE

- 使用方法: blt rs1,rs2,offset
- 实际操作: if(rs1 < rs2) pc += sext(offset)
- 实际含义: 若寄存器x[rs1]小于寄存器x[rs2]的值，把pc的值设为当前值加上符号位扩展的偏移offset
- 二进制串:

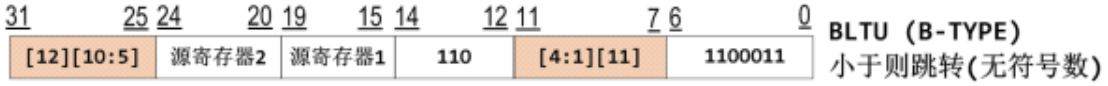


- 注意事项:

> 衍生出来的伪指令（Pesudoinstruction）
>
> bltz => blt rs1,x0,offset (表示小于0时分支)
>
> bgtz => blt x0,rs2,offset (表示大于0时分支)
>
> bgt => blt rs2,rs1,offset (大于时分支)

bltu(Branch if Less Than,Unsigned) B-TYPE

- 使用方法: bltu rs1,rs2,offset
- 实际操作: if(rs1 < rs2)(Unsigned) pc+=sext(offset)
- 实际含义: 若寄存器x[rs1]小于寄存器x[rs2]的值(均视为无符号数)，把pc的值设为当前值加上符号位扩展的偏移offset
- 二进制串:

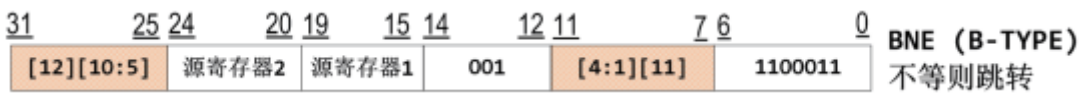


- 注意事项:

> 衍生出来的伪指令（Pesudoinstruction）
>
> bgtu=> bltu rs2,rs1,offset

bne(Branch if Not Equal) B-TYPE

- 使用方法: bne rs1,rs2,offset
- 实际操作: if(rs1 != rs2) pc+=sext(offset)
- 实际含义: 若寄存器x[rs1]和寄存器x[rs]的值不相等，把pc设置为当前值加上符号位扩展的偏移量offset
- 二进制串:



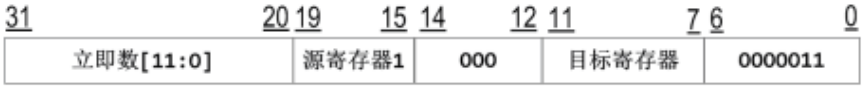
- 注意事项:

> 衍生出来的伪指令（Pesudoinstruction）
>
> bnez => bne rs1,x0,offset

内存载入和存储指令

lb(Load Byte) I-TYPE

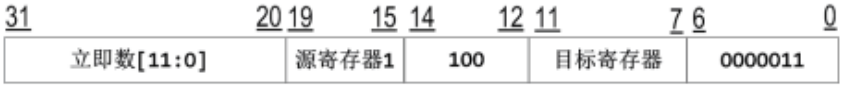
- 使用方法: lb rd,offset(rs1)
- 实际操作: x[rd]=sext(Memory[x[rs1]+sext(offset)[7:0]])
- 实际含义: 从地址x[rs1]+sign-extend(offset)读取一个字节,经符号位扩展后写入x[rd]
- 二进制串:



LB
单字节符号数载入

lbu(Load Byte,Unsigned) I-TYPE

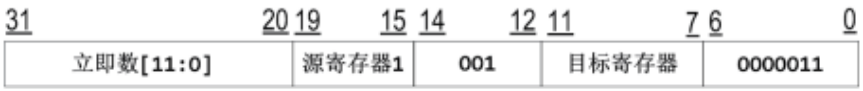
- 使用方法: lbu rd,offset(rs1)
- 实际操作: x[rd]=M[x[rs1]+sext(offset)][7:0]
- 实际含义: 从地址x[rs1]+sign-extend(offset)读取一个字节,经0扩展后写入x[rd]
- 二进制串:



LBU
单字节无符号数载入

lh(Load HalfWord) I-TYPE

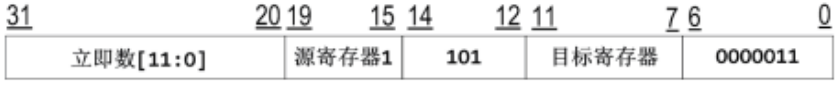
- 使用方法: lh rd,offset(rs1)
- 实际操作: x[rd]=sext(Memory[x[rs1]+sext(offset)[15:0]])
- 实际含义: 从地址x[rs1]+sign-extend(offset)读取两个字节,经符号位扩展后写入x[rd]
- 二进制串:



LH
双字节符号数载入

lhu(Load HalfWord,Unsigned) I-TYPE

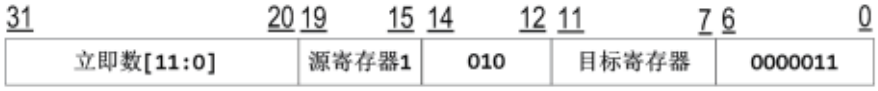
- 使用方法: lhu rd,offset(rs1)
- 实际操作: x[rd]=M[x[rs1]+sext(offset)][15:0]
- 实际含义: 从地址x[rs1]+sign-extend(offset)读取两个字节,经0扩展后写入x[rd]
- 二进制串:



LHU
双字节无符号数载入

lw(Load Word) I-TYPE

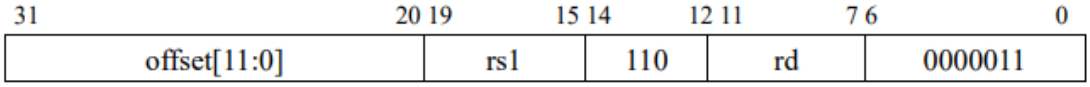
- 使用方法: lw rd,offset(rs1)
- 实际操作: x[rd]=sext(Memory[x[rs1]+sext(offset)[31:0]])
- 实际含义: 从地址x[rs1]+sign-extend(offset)读取四个字节,写入x[rd]（如果RV64I,结果需要符号位扩展）
- 二进制串:



LW
32位符号数载入

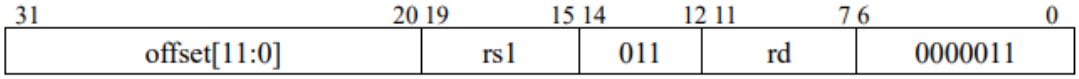
lwu(Load Word,Unsigned) I-TYPE

- 使用方法: lwu rd,offset(rs1)
- 实际操作: x[rd]=M[x[rs1]+sext(offset)][31:0]
- 实际含义: 从地址x[rs1]+sign-extend(offset)读取四个字节,经0扩展后写入x[rd]
- 二进制串:



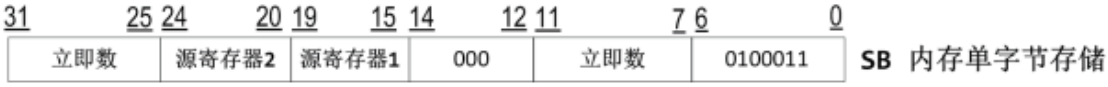
ld(Load DoubleWord) I-TYPE

- 使用方法: ld rd,offset(rs1)
- 实际操作: x[rd]=M[x[rs1]+sext(offset)][63:0]
- 实际含义: 从地址x[rs1]+sign-extend(offset)读取八个字节,写入x[rd]
- 二进制串:



sb(Store Byte) S-TYPE

- 使用方法: sb rs2,offset(rs1)
- 实际操作: M[x[rs1]+sext(offset)] = x[rs2][7:0]
- 实际含义: 将x[rs2]的低位字节存入内存地址x[rs1]+sign-extend(offset)
- 二进制串:

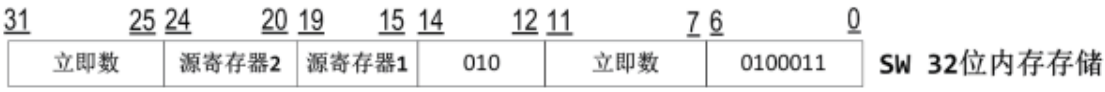


sh(Store HalfWord) S-TYPE

- 使用方法: sh rs2,offset(rs1)
- 实际操作: M[x[rs1]+sext(offset)] = x[rs2][15:0]
- 实际含义: 将x[rs2]的低位2字节存入内存地址x[rs1]+sign-extend(offset)
- 二进制串:

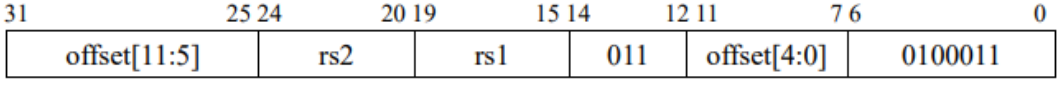
sw(Store Word) S-TYPE

- 使用方法: sw rs2,offset(rs1)
- 实际操作: M[x[rs1]+sext(offset)] = x[rs2][31:0]
- 实际含义: 将x[rs2]的低位2字节存入内存地址x[rs1]+sign-extend(offset)
- 二进制串:



sd(Store DoubleWord) S-TYPE

- 使用方法: sd rs2,offset(rs1)
- 实际操作: M[x[rs1]+sext(offset)] = x[rs2][63:0]
- 实际含义: 将x[rs2]的八字节存入内存地址x[rs1]+sign-extend(offset)
- 二进制串:



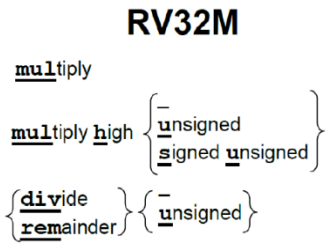
addiw(Add Word Immediate)

- 使用方法: addiw rd,rs1,immediate
- 实际操作: x[rd] = sext((x[rs1]+sext(immediate))[31:0])
- 实际含义: 把符号位扩展的立即数加到寄存器x[rs1],将结果截断为32位，把符号位扩展的结果写入x[rd],忽略算术溢出
- 二进制串:

31 20	19 15	14 12	11 7	6 0
immediate[11:0]	rs1	000	rd	0011011

RV32M指令（扩展）

总览:



31 25	24 20	19 15	14 12	11 7	6 0	
0000001	rs2	rs1	000	rd	0110011	R mul
0000001	rs2	rs1	001	rd	0110011	R mulh
0000001	rs2	rs1	010	rd	0110011	R mulhsu
0000001	rs2	rs1	011	rd	0110011	R mulhu
0000001	rs2	rs1	100	rd	0110011	R div
0000001	rs2	rs1	101	rd	0110011	R divu
0000001	rs2	rs1	110	rd	0110011	R rem
0000001	rs2	rs1	111	rd	0110011	R remu

mul(Multiply) R-TYPE

- 使用方法: mul rd,rs1,rs2
- 实际操作: x[rd] = x[rs1]×x[rs2]
- 实际含义: 把寄存器x[rs2]乘到寄存器x[rs1]上,乘积写入x[rd].忽略算术溢出
- 二进制串:

31 25	24 20	19 15	14 12	11 7	6 0
0000001	源寄存器2	源寄存器1	000	目标寄存器	0110011

MUL 符号数乘法，保留低32位

mulh(Multiply High) R-TYPE

- 使用方法: mul rd,rs1,rs2
- 实际操作: x[rd] = (x[rs1]×x[rs2]) >> XLEN
- 实际含义: 把寄存器x[rs2]乘到寄存器x[rs1]上，都视为2的补码，将乘积的高位写入x[rd]
- 二进制串:

31 25	24 20	19 15	14 12	11 7	6 0
0000001	源寄存器2	源寄存器1	001	目标寄存器	0110011

MULH 符号数乘法，保留高32位

mulhu(Multiply High Unsigned) R-TYPE

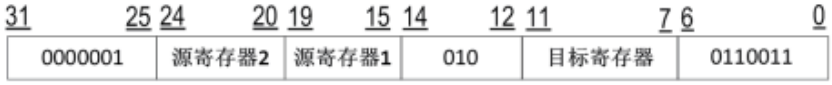
- 使用方法: mulhu rd,rs1,rs2
- 实际操作: x[rd] = (x[rs1]×x[rs2]) >> XLEN (Unsigned)
- 实际含义: 把寄存器x[rs2]乘到寄存器x[rs1]上，x[rs1]、x[rs2]均位无符号数，将乘积的高位写入x[rd]
- 二进制串:

31 25	24 20	19 15	14 12	11 7	6 0
0000001	源寄存器2	源寄存器1	011	目标寄存器	0110011

MULHU 无符号数乘法，保留高32位

mulhsu(Multiply High Signed-Unsigned) R-TYPE

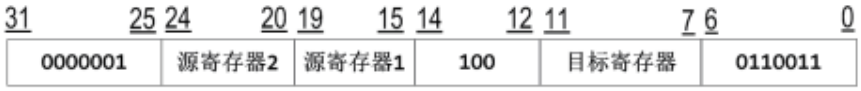
- 使用方法: mulhsu rd,rs1,rs2
- 实际操作: x[rd] = (xrs1×xrs2) >> XLEN
- 实际含义: 把寄存器x[rs2]乘到寄存器x[rs1]上，x[rs1]为2的补码,x[rs2]为无符号数，将乘积的高位写入x[rd]
- 二进制串:



MULHSU 符号数乘无符号数，保留高32位

div(Divid) R-TYPE

- 使用方法: div rd,rs1,rs2
- 实际操作: x[rd] = x[rs1]/x[rs2]
- 实际含义: 用寄存器x[rs1]的值除以寄存器x[rs2]的值,向零舍入，将这些数视为二进制补码，把商写入x[rd]
- 二进制串:



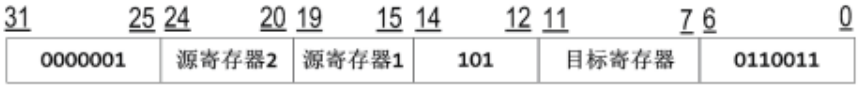
DIV
符号数除法的商

- 注意事项:

  1.当除数为0时，商应该为32位全为1
  2.符号数除法溢出，这种情况只发生在被除数为 -2^31而除数为-1的情况，由于补码的不对称性，2^31无法表示，这种情况下商为32'h80000000

divu(Divid Unsigned) R-TYPE

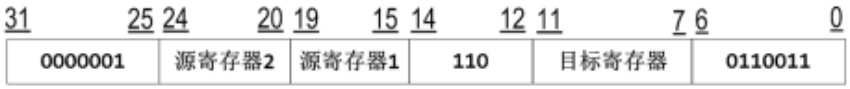
- 使用方法: divu rd,rs1,rs2
- 实际操作: x[rd] = x[rs1]/x[rs2] (unsigned)
- 实际含义: 用寄存器x[rs1]的值除以寄存器x[rs2]的值,向零舍入，将这些数视为无符号数，把商写入x[rd]
- 二进制串:



DIVU
无符号数除法的商

rem(Remainder) R-TYPE

- 使用方法: rem rd,rs1,rs2
- 实际操作: x[rd] = x[rs1]%x[rs2]
- 实际含义: 用寄存器x[rs1]的值除以寄存器x[rs2]的值,向零舍入，将这些数视为二进制补码，把余数写入x[rd]
- 二进制串:



REM
符号数除法的余数

- 注意事项:

  1.当除数为0时，余数应该为被除数
  2.符号数除法溢出，这种情况只发生在被除数为 -2^31而除数为-1的情况，由于补码的不对称性，2^31无法表示，这种情况下余数为0

remu(Remainder Unsigned) R-TYPE

- 使用方法: remu rd,rs1,rs2
- 实际操作: x[rd] = x[rs1]%x[rs2] (unsigned)
- 实际含义: 用寄存器x[rs1]的值除以寄存器x[rs2]的值,向零舍入，将这些数视为无符号数，把余数写入x[rd]
- 二进制串:

31 25	24 20	19 15	14 12	11 7	6 0
0000001	源寄存器2	源寄存器1	111	目标寄存器	0110011

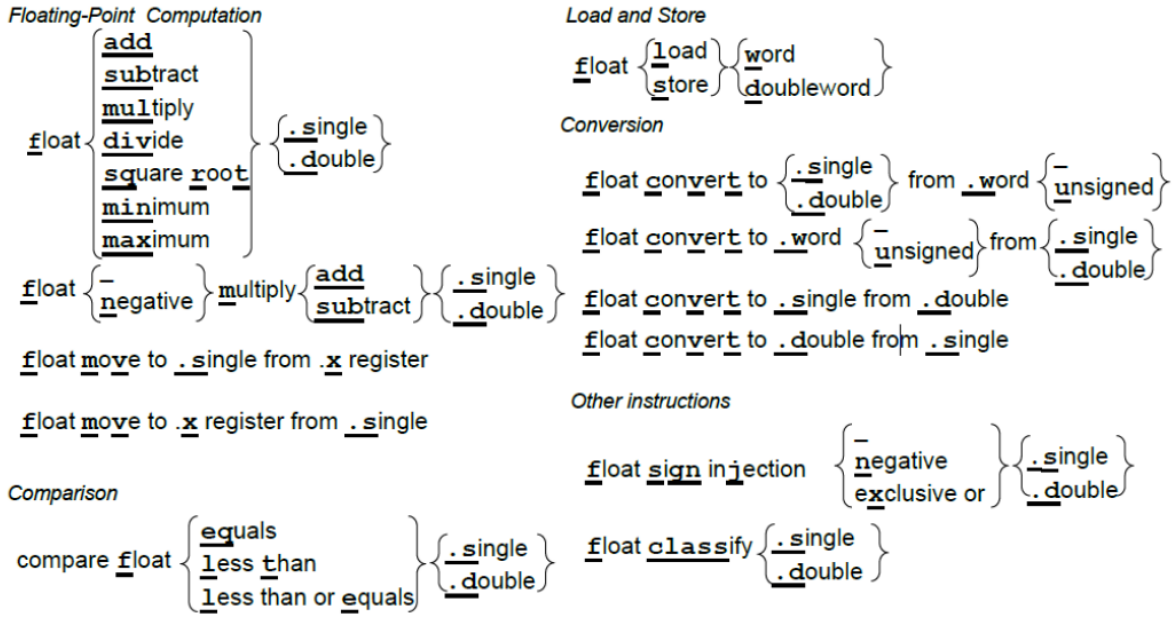
REMU
无符号数除法的余数

RV32F and RV32D

总览:

指令使用方法

RV32F and RV32D



指令操作码表

S For Single;D For Double

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
imm[11:0]			rs1	010	rd	000011	I flw
imm[11:5]		rs2	rs1	010	imm[4:0]	010011	S fsw
rs3	00	rs2	rs1	rm	rd	100001	R4
rs3	00	rs2	rs1	rm	rd	100011	R4
rs3	00	rs2	rs1	rm	rd	100101	R4
rs3	00	rs2	rs1	rm	rd	100111	R4
0000000		rs2	rs1	rm	rd	101001	R fadd.s
0000100		rs2	rs1	rm	rd	101001	R fsub.s
0001000		rs2	rs1	rm	rd	101001	R fmul.s
0001100		rs2	rs1	rm	rd	101001	R fdiv.s
0001100		00000	rs1	rm	rd	101001	R fsqrt.s
0010000		rs2	rs1	000	rd	101001	R fsgnj.s
0010000		rs2	rs1	001	rd	101001	R fsgnjn.s
0010000		rs2	rs1	010	rd	101001	R fsgnjx.s
0010100		rs2	rs1	000	rd	101001	R fmin.s
0010100		rs2	rs1	001	rd	101001	R fmax.s
1100000		00000	rs1	rm	rd	101001	R fcvt.w.s
1100000		00001	rs1	rm	rd	101001	R
1110000		00000	rs1	000	rd	101001	R fmv.x.w
1010000		rs2	rs1	010	rd	101001	R feq.s
1010000		rs2	rs1	001	rd	101001	R flt.s
1010000		rs2	rs1	000	rd	101001	R fle.s
1110000		00000	rs1	001	rd	101001	R fclass.s
1101000		00000	rs1	rm	rd	101001	R fcvt.s.w
1101000		00001	rs1	rm	rd	101001	R
1111000		00000	rs1	000	rd	101001	R fmv.w.x

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
imm[11:0]			rs1	011	rd	000011	I fld
imm[11:5]		rs2	rs1	011	imm[4:0]	010011	S fsd
rs3	01	rs2	rs1	rm	rd	100001	R4
rs3	01	rs2	rs1	rm	rd	100011	R4
rs3	01	rs2	rs1	rm	rd	100101	R4
rs3	01	rs2	rs1	rm	rd	100111	R4
0000001		rs2	rs1	rm	rd	101001	R fadd.d
0000101		rs2	rs1	rm	rd	101001	R fsub.d
0001001		rs2	rs1	rm	rd	101001	R fmul.d
0001101		rs2	rs1	rm	rd	101001	R fdiv.d
0001101		00000	rs1	rm	rd	101001	R fsqrt.d
0010001		rs2	rs1	000	rd	101001	R fsgnj.d
0010001		rs2	rs1	001	rd	101001	R fsgnjn.d
0010001		rs2	rs1	010	rd	101001	R fsgnjx.d
0010101		rs2	rs1	000	rd	101001	R fmin.d
0010101		rs2	rs1	001	rd	101001	R fmax.d
0100000		00001	rs1	rm	rd	101001	R fcvt.s.d
0100001		00000	rs1	rm	rd	101001	R fcvt.d.s
1010001		Rs2	rs1	010	rd	101001	R feq.d
1010001		rs2	rs1	001	rd	101001	R flt.d
1010001		rs2	rs1	000	rd	101001	R fle.d
1110001		00000	rs1	001	rd	101001	R fclass.d
1100001		00000	rs1	rm	rd	101001	R fcvt.w.d
1100001		00001	rs1	rm	rd	101001	R
1101001		00000	rs1	rm	rd	101001	R fmv.d.w
1101001		00001	rs1	rm	rd	101001	R