XuanTie Eseries instruction manual

Aug 24, 2021

T-HEAD Semiconductor Co.,LTD

Web: www.t-head.cn

History

Version	Description	Date
01	First Version。	2021.08.20

目录

1 Appendix A Instruction sets **1**

1.1 Standard instruction sets . . . 1

1.2 XuanTie extended instruction sets . . . 1

1.2.1 Cache instructions . . . 1

1.2.1.1 DCACHE.CALL: an instruction that clears all dirty page table entries in the D-Cache. . . . 1

1.2.1.2 DCACHE.CIALL: an instruction that clears all dirty page table entries in the D-Cache and invalidates the D-Cache. . . . 2

1.2.1.3 DCACHE.CIPA: an instruction that clears dirty page table entries that match the specified physical addresses from the D-Cache and invalidates the the D-Cache. . . . 3

1.2.1.4 DCACHE.CSW: an instruction that clears page table entries in the D-Cache based on the specified way and set. . . . 3

1.2.1.5 DCACHE.CISW: an instruction that clears page table entries in the D-Cache based on the specified way and set and invalidates the D-Cache. . . . 4

1.2.1.6 DCACHE.CPA: an instruction that clears dirty page table entries that match the specified physical addresses from the D-Cache. . . . 5

1.2.1.7 DCACHE.IPA: an instruction that invalidates page table entries that match the specified physical addresses in the D-Cache. . . . 5

1.2.1.8 DCACHE.ISW: an instruction that invalidates page table entries in the D-Cache based on the specified way and set and invalidates the D-Cache. . . . 6

1.2.1.9 DCACHE.IALL: an instruction that invalidates all page table entries in the D-Cache. . . . 7

1.2.1.10 ICACHE.IALL: an instruction that invalidates all page table entries in the I-Cache. . . . 7

1.2.1.11 ICACHE.IPA: an instruction that invalidates page table entries that match the specified physical addresses in the I-Cache. . . . 8

1.2.2 Synchronization instructions . . . 8
1.2.2.1 SYNC: an instruction that performs the synchronization operation. . . . 8
1.2.2.2 SYNC.I: an instruction that synchronizes the clearing operation. . . . 9
1.2.3 Arithmetic operation instructions . . . 9
1.2.3.1 ADDSL: an add register instruction that shifts registers. . . . 9
1.2.3.2 MULA: an instruction that performs a multiply-accumulate operation. . . . 10
1.2.3.3 MULAH: an instruction that performs a multiply-accumulate operation on lower 16 bits. . . . 10
1.2.3.4 MULS: an instruction that performs a multiply-subtraction operation. . . . 11
1.2.3.5 MULSH: an instruction that performs a multiply-subtraction operation on lower 16 bits. . . . 11
1.2.3.6 MVEQZ: an instruction that sends a message when the register is 0. . . . 12
1.2.3.7 MVNEZ: an instruction that sends a message when the register is not 0. . . . 12
1.2.3.8 SRRI: an instruction that implements a cyclic right shift operation on a linked list. . . . 12
1.2.4 Bitwise operation instructions . . . 13
1.2.4.1 EXT: a signed extension instruction that extracts consecutive bits of a register. 13
1.2.4.2 EXTU: an unsigned extension instruction that extracts consecutive bits of a register. . . . 14
1.2.4.3 FF0: an instruction that finds the first bit with the value of 0 in a register. . . 14
1.2.4.4 FF1: an instruction that finds the bit with the value of 1. . . . 15
1.2.4.5 REV: an instruction that reverses the byte order in a word stored in the register. . . . 15
1.2.4.6 TST: an instruction that tests bits with the value of 0. . . . 16
1.2.4.7 TSTNBZ: an instruction that tests bytes with the value of 0. . . . 16
1.2.5 Storage instructions . . . 17
1.2.5.1 LBIA: a base-address auto-increment instruction that extends signed bits and loads bytes. . . . 17
1.2.5.2 LBIB: a signed extension instruction that loads bytes and auto-increments the base address. . . . 17
1.2.5.3 LBUIA: a base-address auto-increment instruction that loads bytes and extends unsigned bits. . . . 18
1.2.5.4 LBUIB: an unsigned extension instruction that loads bytes and auto-increments the base address. . . . 18
1.2.5.5 LHIA: a base-address auto-increment instruction that loads halfwords and extends signed bits. . . . 19
1.2.5.6 LHIB: a signed extension instruction that loads halfwords and auto-increments the base address. . . . 19
1.2.5.7 LHUIA: a base-address auto-increment instruction that loads halfwords and extends unsigned bits. . . . 20
1.2.5.8 LHUIB: an unsigned extension instruction that loads halfwords and auto-increments the base address. . . . 20

1.2.5.9 LRB: a signed extension instruction that shifts registers and loads bytes. . . 21
1.2.5.10 LRBU: an unsigned extension instruction that shifts registers and loads bytes. 21
1.2.5.11 LRH: a load halfword instruction that shifts registers and extends signed bits. 22
1.2.5.12 LRHU: an unsigned extension instruction that shifts registers, extends zero bits, and loads halfwords. 22
1.2.5.13 LRW: a load word instruction that shifts registers. 23
1.2.5.14 LWIA: a base-address auto-increment instruction that loads words. 23
1.2.5.15 LWIB: a load word instruction that auto-increments the base address. 24
1.2.5.16 SBIA: a base-address auto-increment instruction that stores bytes. 24
1.2.5.17 SBIB: a store byte instruction that auto-increments the base address. 25
1.2.5.18 SHIA: a base-address auto-increment instruction that stores halfwords. . . . 25
1.2.5.19 SHIB: a store halfword instruction that auto-increments the base address. . . 25
1.2.5.20 SRB: a store byte instruction that shifts registers. 26
1.2.5.21 SRH: a store halfword instruction that shifts registers. 26
1.2.5.22 SRW: a store word instruction that shifts registers. 27
1.2.5.23 SWIA: a base-address auto-increment instruction that stores words. 27
1.2.5.24 SWIB: a store word instruction that auto-increments the base address. . . . 28
1.2.6 Double-precision floating-point high-bit data transmission instructions 28
1.2.6.1 FMV.X.HW: a transmission instruction that reads double-precision floating-point high-bit data. 28
1.2.6.2 FMV.HW.X: a transmission instruction that writes double-precision floating-point high-bit data. 29
1.2.7 Acceleration interruption instructions 29
1.2.7.1 IPUSH: a push instruction that stops acceleration. 29
1.2.7.2 IPOP: a pop instruction that stops acceleration. 30

CHAPTER 1

Appendix A Instruction sets

1.1 Standard instruction sets

E907 is compatible with the RISC-V architecture. It provides the RV32IMA[F][D]C[P] instruction sets in the following SPEC version:

- *The RISC-V Instruction Set Manual, Volume I: RISC-V User-Level ISA, Version 2.2*

The P instruction set is implemented in V0.9.4. Apart from standard instruction sets, E907 also supports instruction sets optional for Zp64. The P instruction sets are implemented in the following version:

- *RISC-V "P" Extension Proposal Version 0.9.4*

You can download the standard instruction sets from https://riscv.org/technical/specifications/.

1.2 XuanTie extended instruction sets

1.2.1 Cache instructions

You can use the cache instruction set to manage caches. Each instruction has 32 bits. Cache instructions in this instruction set are described in alphabetical order.

1.2.1.1 DCACHE.CALL: an instruction that clears all dirty page table entries in the D-Cache.

Syntax:

dcache.call

Operation:

Clears all page table entries in the D-Cache and writes all dirty page table entries back into the next-level storage.

Permission:

M mode

Exception:

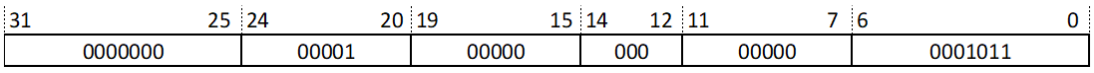
Invalid instruction.

Notes:

If the value of mxstatus.theadisaee is 0, executing this instruction causes an exception of invalid instruction.

If the value of mxstatus.theadisaee is 1, executing this instruction in U mode causes an exception of invalid instruction.

Instruction format:



1.2.1.2 DCACHE.CIALL: an instruction that clears all dirty page table entries in the D-Cache and invalidates the D-Cache.

Syntax:

dcache.ciall

Operation:

Writes all dirty page table entries in the D-Cache back into the next-level storage and invalidates all these page table entries.

Permission:

M mode

Exception:

Invalid instruction.

Notes:

If the value of mxstatus.theadisaee is 0, executing this instruction causes an exception of invalid instruction.

If the value of mxstatus.theadisaee is 1, executing this instruction in U mode causes an exception of invalid instruction.

Instruction format:

31 25	24 20	19 15	14 12	11 7	6 0
0000000	00011	00000	000	00000	0001011

1.2.1.3 DCACHE.CIPA: an instruction that clears dirty page table entries that match the specified physical addresses from the D-Cache and invalidates the the D-Cache.

Syntax:

dcache.cipa rs1

Operation:

Writes the page table entry that matches the specified physical address from the D-Cache of the rs1 register back into the next-level storage and invalidates this page table entry.

Permission:

M mode

Exception:

Invalid instruction.

Notes:

If the value of mxstatus.theadisaee is 0, executing this instruction causes an exception of invalid instruction.

If the value of mxstatus.theadisaee is 1, executing this instruction in U mode causes an exception of invalid instruction.

Instruction format:

31 25	24 20	19 15	14 12	11 7	6 0
0000001	01011	rs1	000	00000	0001011

1.2.1.4 DCACHE.CSW: an instruction that clears page table entries in the D-Cache based on the specified way and set.

Syntax:

dcache.csw rs1

Operation:

Writes the dirty page table entry in the D-Cache of rs1 back into the next-level storage based on the specified way and set.

Permission:

M mode

Exception:

Invalid instruction.

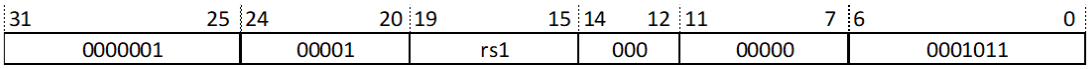
Notes:

E907 D-Cache is a 2-way set-associative cache. rs1[31] specifies the way and rs1[s:6] specifies the set. When the size of the D-Cache is 32 KiB, s denotes 13. When the size of the D-Cache is 16 KiB, s denotes 12, and so forth.

If the value of mxstatus.theadisaee is 0, executing this instruction causes an exception of invalid instruction.

If the value of mxstatus.theadisaee is 1, executing this instruction in U mode causes an exception of invalid instruction.

Instruction format:



1.2.1.5 DCACHE.CISW: an instruction that clears page table entries in the D-Cache based on the specified way and set and invalidates the D-Cache.

Syntax:

dcache.cisw rs1

Operation:

Writes the dirty page table entry that matches the specified way and set from the D-Cache of rs1 back into the next-level storage and invalidates this page table entry.

Permission:

M mode

Exception:

Invalid instruction.

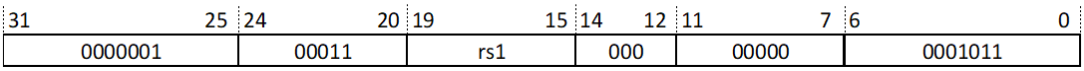
Notes:

E907 D-Cache is a 2-way set-associative cache. rs1[31] specifies the way and rs1[s:6] specifies the set. When the size of the D-Cache is 32 KiB, s denotes 13. When the size of the D-Cache is 16 KiB, s denotes 12, and so forth.

If the value of mxstatus.theadisaee is 0, executing this instruction causes an exception of invalid instruction.

If the value of mxstatus.theadisaee is 1, executing this instruction in U mode causes an exception of invalid instruction.

Instruction format:



31 25	24 20	19 15	14 12	11 7	6 0
0000001	00011	rs1	000	00000	0001011

1.2.1.6 DCACHE.CPA: an instruction that clears dirty page table entries that match the specified physical addresses from the D-Cache.

Syntax:

dcache.cpa rs1

Operation:

Writes the page table entry that matches the specified physical address in the D-Cache of rs1 back into the next-level storage.

Permission:

M mode

Exception:

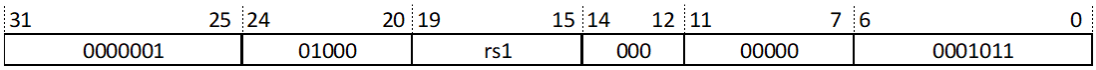
Invalid instruction.

Notes:

If the value of mxstatus.theadisaee is 0, executing this instruction causes an exception of invalid instruction.

If the value of mxstatus.theadisaee is 1, executing this instruction in U mode causes an exception of invalid instruction.

Instruction format:



31 25	24 20	19 15	14 12	11 7	6 0
0000001	01000	rs1	000	00000	0001011

1.2.1.7 DCACHE.IPA: an instruction that invalidates page table entries that match the specified physical addresses in the D-Cache.

Syntax:

dcache.ipa rs1

Operation:

Invalidates the page table entry that matches the specified physical address in the D-Cache of rs1.

Permission:

M mode

Exception:

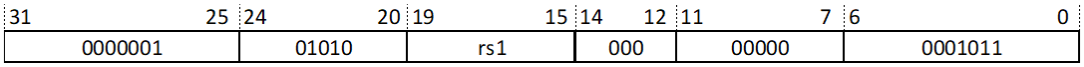
Invalid instruction.

Notes:

If the value of mxstatus.theadisaee is 0, executing this instruction causes an exception of invalid instruction.

If the value of mxstatus.theadisaee is 1, executing this instruction in U mode causes an exception of invalid instruction.

Instruction format:



31 25	24 20	19 15	14 12	11 7	6 0
0000001	01010	rs1	000	00000	0001011

1.2.1.8 DCACHE.ISW: an instruction that invalidates page table entries in the D-Cache based on the specified way and set and invalidates the D-Cache.

Syntax:

dcache.isw rs1

Operation:

Invalidates the page table entry in the D-Cache based on the specified way and set.

Permission:

M mode

Exception:

Invalid instruction.

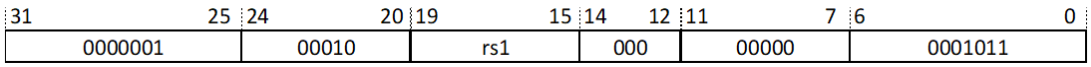
Notes:

E907 D-Cache is a 2-way set-associative cache. rs1[31] specifies the way and rs1[s:6] specifies the set. When the size of the D-Cache is 32 KiB, s denotes 13. When the size of the D-Cache is 16 KB, s denotes 12, and so forth.

If the value of mxstatus.theadisaee is 0, executing this instruction causes an exception of invalid instruction.

If the value of mxstatus.theadisaee is 1, executing this instruction in U mode causes an exception of invalid instruction.

Instruction format:



31 25	24 20	19 15	14 12	11 7	6 0
0000001	00010	rs1	000	00000	0001011

1.2.1.9 DCACHE.IALL: an instruction that invalidates all page table entries in the D-Cache.

Syntax:

dcache.iall

Operation:

Invalidates all page table entries in the D-Cache.

Permission:

M mode

Exception:

Invalid instruction.

Notes:

If the value of mxstatus.theadisaee is 0, executing this instruction causes an exception of invalid instruction.

If the value of mxstatus.theadisaee is 1, executing this instruction in U mode causes an exception of invalid instruction.

Instruction format:

31 25	24 20	19 15	14 12	11 7	6 0
0000000	00010	00000	000	00000	0001011

1.2.1.10 ICACHE.IALL: an instruction that invalidates all page table entries in the I-Cache.

Syntax:

icache.iall

Operation:

Invalidates all page table entries in the I-Cache.

Permission:

M mode

Exception:

Invalid instruction.

Notes:

If the value of mxstatus.theadisaee is 0, executing this instruction causes an exception of invalid instruction.

If the value of mxstatus.theadisaee is 1, executing this instruction in U mode causes an exception of invalid instruction.

Instruction format:

31 25	24 20	19 15	14 12	11 7	6 0
0000000	10000	00000	000	00000	0001011

1.2.1.11 ICACHE.IPA: an instruction that invalidates page table entries that match the specified physical addresses in the I-Cache.

Syntax:

icache.ipa rs1

Operation:

Invalidates the page table entry that matches the specified physical address in the I-Cache of rs1.

Permission:

M mode

Exception:

Invalid instruction.

Notes:

If the value of mxstatus.theadisaee is 0, executing this instruction causes an exception of invalid instruction.

If the value of mxstatus.theadisaee is 1, executing this instruction in U mode causes an exception of invalid instruction.

Instruction format:

31 25	24 20	19 15	14 12	11 7	6 0
0000001	11000	rs1	000	00000	0001011

1.2.2 Synchronization instructions

The synchronization instruction set extends synchronization instructions. Each instruction has 32 bits. Synchronization instructions in this instruction set are described in alphabetical order.

1.2.2.1 SYNC: an instruction that performs the synchronization operation.

Syntax:

sync

Operation:

Ensures that all preceding instructions retire earlier than this instruction and all subsequent instructions retire later than this instruction.

Permission:

M mode/U mode

Exception:

Invalid instruction.

Instruction format:

31 25	24 20	19 15	14 12	11 7	6 0
0000000	11000	00000	000	00000	0001011

1.2.2.2 SYNC.I: an instruction that synchronizes the clearing operation.

Syntax:

sync.i

Operation:

Ensures that all preceding instructions retire earlier than this instruction and all subsequent instructions retire later than this instruction, and clears the pipeline when this instruction retires.

Permission:

M mode/U mode

Exception:

Invalid instruction.

Instruction format:

31 25	24 20	19 15	14 12	11 7	6 0
0000000	11010	00000	000	00000	0001011

1.2.3 Arithmetic operation instructions

The arithmetic operation instruction set extends integer operations. Each instruction has 32 bits. Arithmetic operation instructions in this instruction set are described in alphabetical order.

1.2.3.1 ADDSL: an add register instruction that shifts registers.

Syntax:

addsl rd rs1, rs2, imm2

Operation:

rd ← rs1+ rs2<<imm2

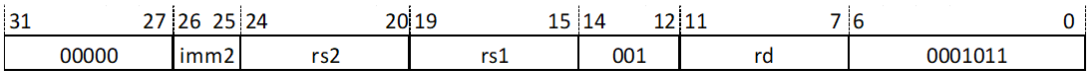
Permission:

M mode/U mode

Exception:

Invalid instruction.

Instruction format:



1.2.3.2 MULA: an instruction that performs a multiply-accumulate operation.

Syntax:

mula rd, rs1, rs2

Operation:

rd ← rd+ (rs1 * rs2)[31:0]

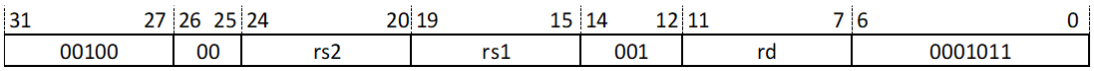
Permission:

M mode/U mode

Exception:

Invalid instruction.

Instruction format:



1.2.3.3 MULAH: an instruction that performs a multiply-accumulate operation on lower 16 bits.

Syntax:

mulah rd, rs1, rs2

Operation:

rd ← rd+ (rs1[15:0] * rs[15:0])

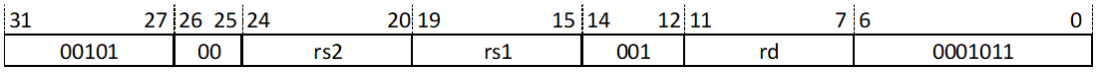
Permission:

M mode/U mode

Exception:

Invalid instruction.

Instruction format:



1.2.3.4 MULS: an instruction that performs a multiply-subtraction operation.

Syntax:

muls rd, rs1, rs2

Operation:

rd ← rd- (rs1 * rs2)[31:0]

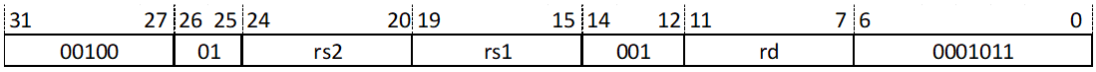
Permission:

M mode/U mode

Exception:

Invalid instruction.

Instruction format:



1.2.3.5 MULSH: an instruction that performs a multiply-subtraction operation on lower 16 bits.

Syntax:

mulsh rd, rs1, rs2

Operation:

tmp[31:0] ← rd[31:0]- (rs1[15:0] * rs[15:0])

rd ←sign_extend(tmp[31:0])

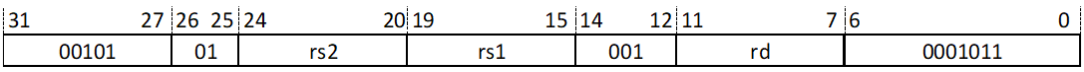
Permission:

M mode/U mode

Exception:

Invalid instruction.

Instruction format:



1.2.3.6 MVEQZ: an instruction that sends a message when the register is 0.

Syntax:

mveqz rd, rs1, rs2

Operation:

if (rs2 == 0)

rd ← rs1

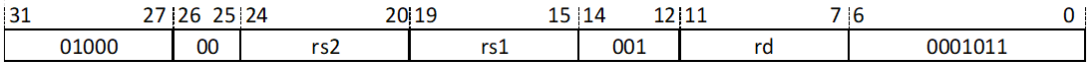
Permission:

M mode/U mode

Exception:

Invalid instruction.

Instruction format:



31 27	26 25	24 20	19 15	14 12	11 7	6 0
01000	00	rs2	rs1	001	rd	0001011

1.2.3.7 MVNEZ: an instruction that sends a message when the register is not 0.

Syntax:

mvnez rd, rs1, rs2

Operation:

if (rs2 != 0)

rd ← rs1

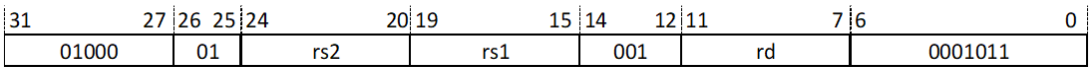
Permission:

M mode/U mode

Exception:

Invalid instruction.

Instruction format:



31 27	26 25	24 20	19 15	14 12	11 7	6 0
01000	01	rs2	rs1	001	rd	0001011

1.2.3.8 SRRI: an instruction that implements a cyclic right shift operation on a linked list.

Syntax:

srri rd, rs1, imm5

Operation:

rd ← rs1 >>>> imm5

Shifts the original value of rs1 to the right, disconnects the last value on the list, and re-attaches the value to the start of the linked list.

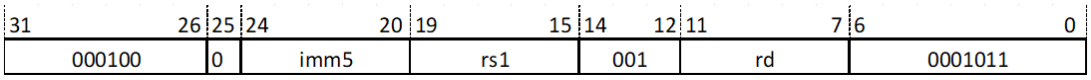
Permission:

M mode/U mode

Exception:

Invalid instruction.

Instruction format:



1.2.4 Bitwise operation instructions

The bitwise operation instruction set extends bitwise operations. Each instruction has 32 bits. Bitwise operation instructions in this instruction set are described in alphabetical order.

1.2.4.1 EXT: a signed extension instruction that extracts consecutive bits of a register.

Syntax:

ext rd, rs1, imm1,imm2

Operation:

rd←sign_extend(rs1[imm1:imm2])

Permission:

M mode/U mode

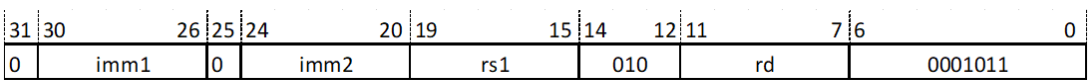
Exception:

Invalid instruction.

Note:

If imm1 is smaller than imm2, the action of this instruction is not predictable.

Instruction format:



1.2.4.2 EXTU: an unsigned extension instruction that extracts consecutive bits of a register.

Syntax:

extu rd, rs1, imm1,imm2

Operation:

rd←zero_extend(rs1[imm1:imm2])

Permission:

M mode/U mode

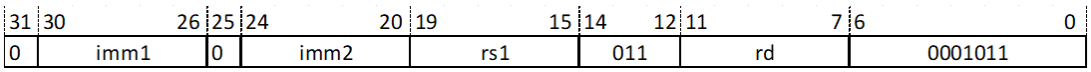
Exception:

Invalid instruction.

Note:

If imm1 is smaller than imm2, the action of this instruction is not predictable.

Instruction format:



1.2.4.3 FF0: an instruction that finds the first bit with the value of 0 in a register.

Syntax:

ff0 rd, rs1

Operation:

Finds the first bit with the value of 0 from the highest bit of rs1 and writes the result back into the rd register. If the highest bit of rs1 is 0, the result 0 is returned. If all the bits in rs1 are 1, the result 32 is returned.

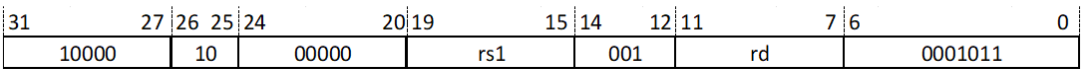
Permission:

M mode/U mode

Exception:

Invalid instruction.

Instruction format:



1.2.4.4 FF1: an instruction that finds the bit with the value of 1.

Syntax:

ff1 rd, rs1

Operation:

Finds the first bit with the value of 1 from the highest bit of rs1 and writes the result back into rd. If the highest bit of rs1 is 1, the result 0 is returned. If all the bits in rs1 are 0, the result 32 is returned and written into rd.

Permission:

M mode/U mode

Exception:

Invalid instruction.

Instruction format:

31 27	26 25	24 20	19 15	14 12	11 7	6 0
10000	11	00000	rs1	001	rd	0001011

1.2.4.5 REV: an instruction that reverses the byte order in a word stored in the register.

Syntax:

rev rd, rs1

Operation:

rd[31:24] ←rs1[7:0]

rd[23:16] ←rs1[15:8]

rd[15:8] ←rs1[23:16]

rd[7:0] ←rs1[31:24]

Permission:

M mode/U mode

Exception:

Invalid instruction.

Instruction format:

31 27	26 25	24 20	19 15	14 12	11 7	6 0
10000	01	00000	rs1	001	rd	0001011

1.2.4.6 TST: an instruction that tests bits with the value of 0.

Syntax:

tst rd, rs1, imm5

Operation:

if(rs1[imm5] == 1)

rd←1

else

rd←0

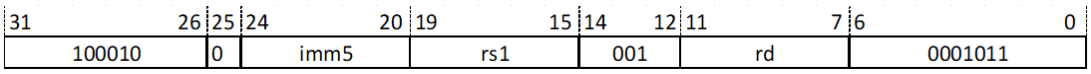
Permission:

M mode/U mode

Exception:

Invalid instruction.

Instruction format:



1.2.4.7 TSTNBZ: an instruction that tests bytes with the value of 0.

Syntax:

tstnbz rd, rs1

Operation:

for (i =0; i<4;i++)

if(rs1[8i+7:8i] == 0)

rd[8i+7:8i] = 8' hff

else

rd[8i+7:8i] = 8' h0

Permission:

M mode/U mode

Exception:

Invalid instruction.

Instruction format:

31 27	26 25	24 20	19 15	14 12	11 7	6 0
10000	00	00000	rs1	001	rd	0001011

1.2.5 Storage instructions

The storage instruction set extends storage operations. Each instruction has 32 bits. Storage instructions in this instruction set are described in alphabetical order.

1.2.5.1 LBIA: a base-address auto-increment instruction that extends signed bits and loads bytes.

Syntax:

lbia rd, (rs1), imm5,imm2

Operation:

rd ←sign_extend(mem[rs1])

rs1←rs1 + sign_extend(imm5 << imm2)

Permission:

M mode/U mode

Exception:

Unaligned access, access error, or invalid instruction.

Note:

The values of rd and rs1 must not be the same.

Instruction format:

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00011	imm2	imm5	rs1	100	rd	0001011

1.2.5.2 LBIB: a signed extension instruction that loads bytes and auto-increments the base address.

Syntax:

lbib rd, (rs1), imm5,imm2

Operation:

rs1←rs1 + sign_extend(imm5 << imm2)

rd ←sign_extend(mem[rs1+7:rs1])

Permission:

M mode/U mode

Exception:

Unaligned access, access error, or invalid instruction.

Note:

The values of rd and rs1 must not be the same.

Instruction format:

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00001	imm2	imm5	rs1	100	rd	0001011

1.2.5.3 LBUIA: a base-address auto-increment instruction that loads bytes and extends unsigned bits.

Syntax:

lbuia rd, (rs1), imm5,imm2

Operation:

rd ←zero_extend(mem[rs1])

rs1←rs1 + sign_extend(imm5 << imm2)

Permission:

M mode/U mode

Exception:

Unaligned access, access error, or invalid instruction.

Note:

The values of rd and rs1 must not be the same.

Instruction format:

31 27	26 25	24 20	19 15	14 12	11 7	6 0
10011	imm2	imm5	rs1	100	rd	0001011

1.2.5.4 LBUIB: an unsigned extension instruction that loads bytes and auto-increments the base address.

Syntax:

lbuib rd, (rs1), imm5,imm2

Operation:

rs1←rs1 + sign_extend(imm5 << imm2)

rd ←zero_extend(mem[rs1])

Permission:

M mode/U mode

Exception:

Unaligned access, access error, or invalid instruction.

Note:

The values of rd and rs1 must not be the same.

Instruction format:

31 27	26 25	24 20	19 15	14 12	11 7	6 0
10001	imm2	imm5	rs1	100	rd	0001011

1.2.5.5 LHIA: a base-address auto-increment instruction that loads halfwords and extends signed bits.

Syntax:

lhia rd, (rs1), imm5,imm2

Operation:

rd ←sign_extend(mem[rs1+1:rs1])

rs1←rs1 + sign_extend(imm5 << imm2)

Permission:

M mode/U mode

Exception:

Unaligned access, access error, or invalid instruction.

Note:

The values of rd and rs1 must not be the same.

Instruction format:

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00111	imm2	imm5	rs1	100	rd	0001011

1.2.5.6 LHIB: a signed extension instruction that loads halfwords and auto-increments the base address.

Syntax:

lhib rd, (rs1), imm5,imm2

Operation:

rs1←rs1 + sign_extend(imm5 << imm2)

rd ←sign_extend(mem[rs1+1:rs1])

Permission:

M mode/U mode

Exception:

Unaligned access, access error, or invalid instruction.

Note:

The values of rd and rs1 must not be the same.

Instruction format:

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00101	imm2	imm5	rs1	100	rd	0001011

1.2.5.7 LHUIA: a base-address auto-increment instruction that loads halfwords and extends unsigned bits.

Syntax:

lhuia rd, (rs1), imm5,imm2

Operation:

rd ←zero_extend(mem[rs1+1:rs1])

rs1←rs1 + sign_extend(imm5 << imm2)

Permission:

M mode/U mode

Exception:

Unaligned access, access error, or invalid instruction.

Note:

The values of rd and rs1 must not be the same.

Instruction format:

31 27	26 25	24 20	19 15	14 12	11 7	6 0
10111	imm2	imm5	rs1	100	rd	0001011

1.2.5.8 LHUIB: an unsigned extension instruction that loads halfwords and auto-increments the base address.

Syntax:

lhuib rd, (rs1), imm5,imm2

Operation:

rs1←rs1 + sign_extend(imm5 << imm2)

rd ←zero_extend(mem[rs1+1:rs1])

Permission:

M mode/U mode

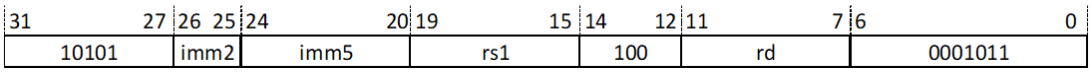
Exception:

Unaligned access, access error, or invalid instruction.

Note:

The values of rd and rs1 must not be the same.

Instruction format:



1.2.5.9 LRB: a signed extension instruction that shifts registers and loads bytes.

Syntax:

lrb rd, rs1, rs2, imm2

Operation:

rd ←sign_extend(mem[(rs1+rs2<<imm2)])

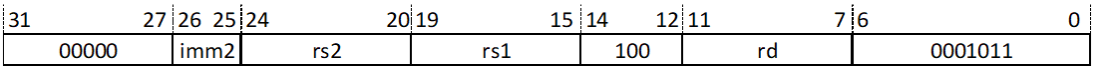
Permission:

M mode/U mode

Exception:

Unaligned access, access error, or invalid instruction.

Instruction format:



1.2.5.10 LRBU: an unsigned extension instruction that shifts registers and loads bytes.

Syntax:

lrbu rd, rs1, rs2, imm2

Operation:

rd ←zero_extend(mem[(rs1+rs2<<imm2)])

Permission:

M mode/U mode

Exception:

Unaligned access, access error, or invalid instruction.

Instruction format:

31 27	26 25	24 20	19 15	14 12	11 7	6 0
10000	imm2	rs2	rs1	100	rd	0001011

1.2.5.11 LRH: a load halfword instruction that shifts registers and extends signed bits.

Syntax:

lrh rd, rs1, rs2, imm2

Operation:

rd ←sign_extend(mem[(rs1+rs2<<imm2)+1:(rs1+rs2<<imm2)])

Permission:

M mode/U mode

Exception:

Unaligned access, access error, or invalid instruction.

Instruction format:

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00100	imm2	rs2	rs1	100	rd	0001011

1.2.5.12 LRHU: an unsigned extension instruction that shifts registers, extends zero bits, and loads halfwords.

Syntax:

lrhu rd, rs1, rs2, imm2

Operation:

rd ←zero_extend(mem[(rs1+rs2<<imm2)+1:(rs1+rs2<<imm2)])

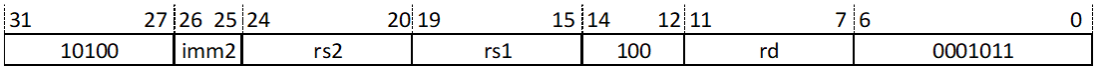
Permission:

M mode/U mode

Exception:

Unaligned access, access error, or invalid instruction.

Instruction format:



1.2.5.13 LRW: a load word instruction that shifts registers.

Syntax:

lrw rd, rs1, rs2, imm2

Operation:

rd ←(mem[(rs1+rs2<<imm2)+3:(rs1+rs2<<imm2)]

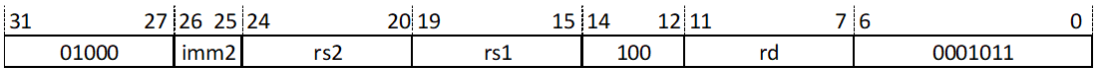
Permission:

M mode/U mode

Exception:

Unaligned access, access error, or invalid instruction.

Instruction format:



1.2.5.14 LWIA: a base-address auto-increment instruction that loads words.

Syntax:

lwia rd, (rs1), imm5,imm2

Operation:

rd ←(mem[rs1+3:rs1])

rs1←rs1 + sign_extend(imm5 << imm2)

Permission:

M mode/U mode

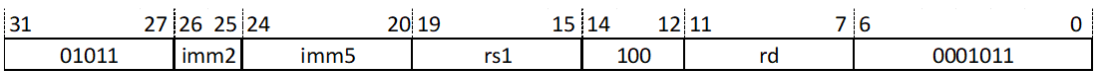
Exception:

Unaligned access, access error, or invalid instruction.

Note:

The values of rd and rs1 must not be the same.

Instruction format:



1.2.5.15 LWIB: a load word instruction that auto-increments the base address.

Syntax:

lwib rd, (rs1), imm5,imm2

Operation:

rs1←rs1 + sign_extend(imm5 << imm2)

rd ←(mem[rs1+3:rs1])

Permission:

M mode/U mode

Exception:

Unaligned access, access error, page error, or invalid instruction.

Note:

The values of rd and rs1 must not be the same.

Instruction format:

31 27	26 25	24 20	19 15	14 12	11 7	6 0
01001	imm2	imm5	rs1	100	rd	0001011

1.2.5.16 SBIA: a base-address auto-increment instruction that stores bytes.

Syntax:

sbia rs2, (rs1), imm5,imm2

Operation:

mem[rs1]←rs2[7:0]

rs1←rs1 + sign_extend(imm5 << imm2)

Permission:

M mode/U mode

Exception:

Unaligned access, access error, or invalid instruction.

Instruction format:

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00011	imm2	imm5	rs1	101	rs2	0001011

1.2.5.17 SBIB: a store byte instruction that auto-increments the base address.

Syntax:

sbib rs2, (rs1), imm5,imm2

Operation:

rs1←rs1 + sign_extend(imm5 << imm2)

mem[rs1] ←rs2[7:0]

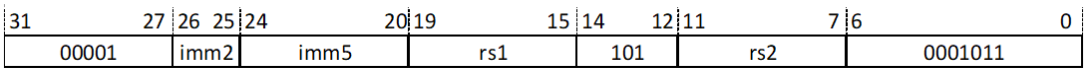
Permission:

M mode/U mode

Exception:

Unaligned access, access error, or invalid instruction.

Instruction format:



1.2.5.18 SHIA: a base-address auto-increment instruction that stores halfwords.

Syntax:

shia rs2, (rs1), imm5,imm2

Operation:

mem[rs1+1:rs1]←rs2[15:0]

rs1←rs1 + sign_extend(imm5 << imm2)

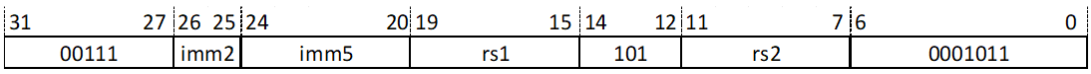
Permission:

M mode/U mode

Exception:

Unaligned access, access error, or invalid instruction.

Instruction format:



1.2.5.19 SHIB: a store halfword instruction that auto-increments the base address.

Syntax:

shib rs2, (rs1), imm5,imm2

Operation:

rs1←rs1 + sign_extend(imm5 << imm2)

mem[rs1+1:rs1] ←rs2[15:0]

Permission:

M mode/U mode

Exception:

Unaligned access, access error, or invalid instruction.

Instruction format:

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00101	imm2	imm5	rs1	101	rs2	0001011

1.2.5.20 SRB: a store byte instruction that shifts registers.

Syntax:

srb rd, rs1, rs2, imm2

Operation:

mem[(rs1+rs2<<imm2)] ←rd[7:0]

Permission:

M mode/U mode

Exception:

Unaligned access, access error, or invalid instruction.

Instruction format:

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00000	imm2	imm5	rs1	101	rd	0001011

1.2.5.21 SRH: a store halfword instruction that shifts registers.

Syntax:

srh rd, rs1, rs2, imm2

Operation:

mem[(rs1+rs2<<imm2)+1:(rs1+rs2<<imm2)] ←rd[15:0]

Permission:

M mode/U mode

Exception:

Unaligned access, access error, or invalid instruction.

Instruction format:

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00100	imm2	rs2	rs1	101	rd	0001011

1.2.5.22 SRW: a store word instruction that shifts registers.

Syntax:

srw rd, rs1, rs2, imm2

Operation:

mem[(rs1+rs2<<imm2)+3:(rs1+rs2<<imm2)] ←rd[31:0]

Permission:

M mode/U mode

Exception:

Unaligned access, access error, or invalid instruction.

Instruction format:

31 27	26 25	24 20	19 15	14 12	11 7	6 0
01000	imm2	rs2	rs1	101	rd	0001011

1.2.5.23 SWIA: a base-address auto-increment instruction that stores words.

Syntax:

swia rs2, (rs1), imm5,imm2

Operation:

mem[rs1+3:rs1]←rs2

rs1←rs1 + sign_extend(imm5 << imm2)

Permission:

M mode/U mode

Exception:

Unaligned access, access error, or invalid instruction.

Instruction format:

31 27	26 25	24 20	19 15	14 12	11 7	6 0
01011	imm2	imm5	rs1	101	rs2	0001011

1.2.5.24 SWIB: a store word instruction that auto-increments the base address.

Syntax:

swib rs2, (rs1), imm5,imm2

Operation:

rs1←rs1 + sign_extend(imm5 << imm2)

mem[rs1+3:rs1] ←rs2

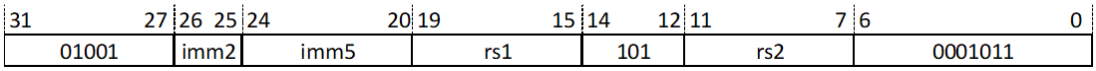
Permission:

M mode/U mode

Exception:

Unaligned access, access error, or invalid instruction.

Instruction format:



1.2.6 Double-precision floating-point high-bit data transmission instructions

You can use instructions in this instruction set to extend data transmission operations for double-precision floating-point data and integer data. Each instruction has 32 bits. Instructions in this instruction set are described in alphabetical order.

1.2.6.1 FMV.X.HW: a transmission instruction that reads double-precision floating-point high-bit data.

Syntax:

fmv.x.hw rd, fs1

Operation:

rd←fs1[63:32]

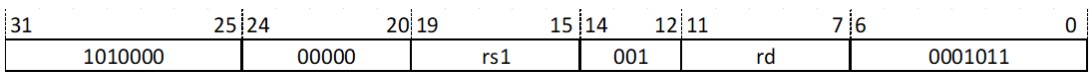
Permission:

M mode/U mode

Exception:

Invalid instruction.

Instruction format:



1.2.6.2 FMV.HW.X: a transmission instruction that writes double-precision floating-point high-bit data.

Syntax:

fmv.x.hw rd, fs1

Operation:

fs1[63:32]←rd

Permission:

M mode/U mode

Exception:

Invalid instruction.

Instruction format:

31 25	24 20	19 15	14 12	11 7	6 0
1100000	00000	rs1	001	rd	0001011

1.2.7 Acceleration interruption instructions

You can use instructions in this instruction set to extend acceleration interruption response scenarios. Each instruction has 32 bits. Instructions in this instruction set are described in alphabetical order.

1.2.7.1 IPUSH: a push instruction that stops acceleration.

Syntax:

ipush

Operation:

mem[int_sp-4]~mem[int_sp-72] ←{mcause,mepc,Xn};

int_sp=int_sp-72

The values of Xn are X1, X5-X7, X10-X17, and X28-X31, respectively.

Permission:

M mode

Exception:

Unaligned access, access error, or invalid instruction.

Instruction format:

31 25	24 20	19 15	14 12	11 7	6 0
0000000	00100	00000	000	00000	0001011

1.2.7.2 IPOP: a pop instruction that stops acceleration.

Syntax:

ipop

Operation:

{mcause,mepc,Xn}←mem[int_sp+68]~mem[int_sp];int_sp=int_sp+72;mret

The values of Xn are X1, X5-X7, X10-X17, and X28-X31, respectively.

Permission:

M mode

Exception:

Unaligned access, access error, or invalid instruction.

Instruction format:

31 25	24 20	19 15	14 12	11 7	6 0
0000000	00101	00000	000	00000	0001011