

XuanTie Cseries instruction manual

Aug 24, 2021

Copyright © 2021 T-HEAD Semiconductor Co.,Ltd. All rights reserved.

This document is the property of T-HEAD Semiconductor Co.,Ltd. This document may only be distributed to: (i) a T-HEAD party having a legitimate business need for the information contained herein, or (ii) a non-T-HEAD party having a legitimate business need for the information contained herein. No license, expressed or implied, under any patent, copyright or trade secret right is granted or implied by the conveyance of this document. No part of this document may be reproduced, transmitted, transcribed, stored in a retrieval system, translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise without the prior written permission of T-HEAD Semiconductor Co.,Ltd.

Trademarks and Permissions

The T-HEAD Logo and all other trademarks indicated as such herein are trademarks of Hangzhou T-HEAD Semiconductor Co.,Ltd. All other products or service names are the property of their respective owners.

Notice

The purchased products, services and features are stipulated by the contract made between T-HEAD and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied. The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

T-HEAD Semiconductor Co.,LTD

Web: www.t-head.cn

History

Version	Description	Date
01	First Version.	2021.08.20

1	Appendix B-1 Cache instructions	2
1.1	DCACHE.CALL: an instruction that clears all page table entries in the D-Cache.	2
1.2	DCACHE.CIALL: an instruction that clears all dirty page table entries in the D-Cache and invalidates the D-Cache.	3
1.3	DCACHE.CIPA: an instruction that clears page table entries that match the specified physical addresses from the D-Cache and invalidates the the D-Cache.	3
1.4	DCACHE.CISW: an instruction that clears dirty page table entries in the D-Cache based on the specified way and set and invalidates the D-Cache.	4
1.5	DCACHE.CIVA: an instruction that clears dirty page table entries that match the specified virtual addresses in the D-Cache and invalidates the D-Cache.	5
1.6	DCACHE.CPA: an instruction that clears dirty page table entries that match the specified physical addresses from the D-Cache.	6
1.7	DCACHE.CPAL1: an instruction that clears dirty page table entries that match the specified physical addresses from in the L1 D-Cache.	6
1.8	DCACHE.CVA: an instruction that clears dirty page table entries that match the specified virtual addresses in the D-Cache.	7
1.9	DCACHE.CVAL1: an instruction that clears dirty page table entries that match the specified virtual addresses in the L1 D-Cache.	8
1.10	DCACHE.IPA: an instruction that invalidates page table entries that match the specified physical addresses in the D-Cache.	8
1.11	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.	9
1.12	DCACHE.IVA: an instruction that invalidates the D-Cache based on the specified virtual address.	10
1.13	DCACHE.IALL: an instruction that invalidates all page table entries in the D-Cache.	10
1.14	ICACHE.IALL: an instruction that invalidates all page table entries in the I-Cache.	11

1.15	ICACHE.IALLS: an instruction that invalidates all page table entries in the I-Cache through broadcasting.	12
1.16	ICACHE.IPA: an instruction that invalidates page table entries that match the specified physical addresses in the I-Cache.	12
1.17	ICACHE.IVA: an instruction that invalidates page table entries that match the specified virtual addresses in the I-Cache.	13
1.18	L2CACHE.CALL: an instruction that clears all dirty page table entries in the L2 Cache. . .	14
1.19	L2CACHE.CIALL: an instruction that clears all dirty page table entries in the L2 Cache and invalidates the L2 Cache.	14
1.20	L2CACHE.IALL: an instruction that invalidates the L2 Cache.	15
1.21	DCACHE.CSW: an instruction that clears dirty page table entries in the D-Cache based on the specified set and way.	16
2	Appendix B-2 Multi-core synchronization instructions	17
2.1	SFENCE.VMAS: a broadcast instruction that synchronizes the virtual memory address. . . .	17
2.2	SYNC: an instruction that performs the synchronization operation.	18
2.3	SYNC.I: an instruction that synchronizes the clearing operation.	19
2.4	SYNC.IS: a broadcast instruction that synchronizes the clearing operation.	19
2.5	SYNC.S: a broadcast instruction that performs a synchronization operation.	20
3	Appendix B-3 Arithmetic operation instructions	21
3.1	ADDSL: an add register instruction that shifts registers.	21
3.2	MULA: an instruction that performs a multiply-accumulate operation.	22
3.3	MULAH: an instruction that performs a multiply-accumulate operation on lower 16 bits. . .	22
3.4	MULAW: an instruction that performs a multiply-accumulate operation on lower 32 bits. . .	23
3.5	MULS: an instruction that performs a multiply-subtraction operation.	23
3.6	MULSH: an instruction that performs a multiply-subtraction operation on lower 16 bits. . .	24
3.7	MULSW: an instruction that performs a multiply-subtraction operation on lower 32 bits. . .	24
3.8	MVEQZ: an instruction that sends a message when the register is 0.	25
3.9	MVNEZ: an instruction that sends a message when the register is not 0.	25
3.10	SRRI: an instruction that implements a cyclic right shift operation on a linked list.	26
3.11	SRRIW: an instruction that implements a cyclic right shift operation on a linked list of low 32 bits of registers.	26
4	Appendix B-4 Bitwise operation instructions	28
4.1	EXT: a signed extension instruction that extracts consecutive bits of a register.	28
4.2	EXTU: a zero extension instruction that extracts consecutive bits of a register.	29
4.3	FF0: an instruction that finds the first bit with the value of 0 in a register.	29
4.4	FF1: an instruction that finds the bit with the value of 1.	30
4.5	REV: an instruction that reverses the byte order in a word stored in the register.	30
4.6	RE VW: an instruction that reverses the byte order in a low 32-bit word.	31
4.7	TST: an instruction that tests bits with the value of 0.	32
4.8	TSTNBZ: an instruction that tests bytes with the value of 0.	32

5	Appendix B-5 Storage instructions	34
5.1	FLRD: a load doubleword instruction that shifts floating-point registers.	34
5.2	FLRW: a load word instruction that shifts floating-point registers.	35
5.3	FLURD: a load doubleword instruction that shifts low 32 bits of floating-point registers. . .	35
5.4	FLURW: a load word instruction that shifts low 32 bits of floating-point registers.	36
5.5	FSRD: a store doubleword instruction that shifts floating-point registers.	36
5.6	FSRW: a store word instruction that shifts floating-point registers.	37
5.7	FSURD: a store doubleword instruction that shifts low 32 bits of floating-point registers. . .	38
5.8	FSURW: a store word instruction that shifts low 32 bits of floating-point registers.	38
5.9	LBIA: a base-address auto-increment instruction that loads bytes and extends signed bits. .	39
5.10	LBIB: a load byte instruction that auto-increments the base address and extends signed bits.	39
5.11	LBUIA: a base-address auto-increment instruction that extends zero bits and loads bytes. . .	40
5.12	LBUIB: a load byte instruction that auto-increments the base address and extends zero bits.	41
5.13	LDD: an instruction that loads double registers.	41
5.14	LDIA: a base-address auto-increment instruction that loads doublewords and extends signed bits.	42
5.15	LDIB: a load doubleword instruction that auto-increments the base address and extends the signed bits.	42
5.16	LHIA: a base-address auto-increment instruction that loads halfwords and extends signed bits.	43
5.17	LHIB: a load halfword instruction that auto-increments the base address and extends signed bits.	43
5.18	LHUIA: a base-address auto-increment instruction that extends zero bits and loads halfwords.	44
5.19	LHUIB: a load halfword instruction that auto-increments the base address and extends zero bits.	45
5.20	LRB: a load byte instruction that shifts registers and extends signed bits.	45
5.21	LRBU: a load byte instruction that shifts registers and extends zero bits.	46
5.22	LRD: a load doubleword instruction that shifts registers.	46
5.23	LRH: a load halfword instruction that shifts registers and extends signed bits.	47
5.24	LRHU: a load halfword instruction that shifts registers and extends zero bits.	47
5.25	LRW: a load word instruction that shifts registers and extends signed bits.	48
5.26	LRWU: a load word instruction that shifts registers and extends zero bits.	48
5.27	LURB: a load byte instruction that shifts low 32 bits of registers and extends signed bits. . .	49
5.28	LURBU: a load byte instruction that shifts low 32 bits of registers and extends zero bits. . .	49
5.29	LURD: a load doubleword instruction that shifts low 32 bits of registers.	50
5.30	LURH: a load halfword instruction that shifts low 32 bits of registers and extends signed bits.	50
5.31	LURHU: a load halfword instruction that shifts low 32 bits of registers and extends zero bits.	51
5.32	LURW: a load word instruction that shifts low 32 bits of registers and extends signed bits. .	51
5.33	LURWU: a load word instruction that shifts 32 bits of registers and extends zero bits.	52
5.34	LWD: a load word instruction that loads double registers and extends signed bits.	53
5.35	LWIA: a base-address auto-increment instruction that extends signed bits and loads words. .	53
5.36	LWIB: a load word instruction that auto-increments the base address and extends signed bits.	54
5.37	LWUD: a load word instruction that loads double registers and extends zero bits.	54

5.38	LWUIA: a base-address auto-increment instruction that extends zero bits and loads words. .	55
5.39	LWUIB: a load word instruction that auto-increments the base address and extends zero bits.	55
5.40	SBIA: a base-address auto-increment instruction that stores bytes.	56
5.41	SBIB: a store byte instruction that auto-increments the base address.	56
5.42	SDD: an instruction that stores double registers.	57
5.43	SDIA: a base-address auto-increment instruction that stores doublewords.	57
5.44	SDIB: a store doubleword instruction that auto-increments the base address.	58
5.45	SHIA: a base-address auto-increment instruction that stores halfwords.	58
5.46	SHIB: a store halfword instruction that auto-increments the base address.	59
5.47	SRB: a store byte instruction that shifts registers.	59
5.48	SRD: a store doubleword instruction that shifts registers.	60
5.49	SRH: a store halfword instruction that shifts registers.	60
5.50	SRW: a store word instruction that shifts registers.	61
5.51	SURB: a store byte instruction that shifts low 32 bits of registers.	61
5.52	SURD: a store doubleword instruction that shifts low 32 bits of registers.	62
5.53	SURH: a store halfword instruction that shifts low 32 bits of registers.	62
5.54	SURW: a store word instruction that shifts low 32 bits of registers.	63
5.55	SWIA: a base-address auto-increment instruction that stores words.	63
5.56	SWIB: a store word instruction that auto-increments the base address.	64
5.57	SWD: an instruction that stores the low 32 bits of double registers.	64
6	Appendix B-6 Half-precision floating-point instructions	66
6.1	FADD.H: a half-precision floating-point add instruction.	66
6.2	FCLASS.H: a half-precision floating-point classification instruction.	67
6.3	FCVT.D.H: an instruction that converts half-precision floating-point data to double-precision floating-point data.	68
6.4	FCVT.H.D: an instruction that converts double-precision floating-point data to half-precision floating-point data.	69
6.5	FCVT.H.L: an instruction that converts a signed long integer into a half-precision floating-point number.	70
6.6	FCVT.H.LU: an instruction that converts an unsigned long integer into a half-precision floating-point number.	71
6.7	FCVT.H.S: an instruction that converts single precision floating-point data to half-precision floating-point data.	72
6.8	FCVT.H.W: an instruction that converts a signed integer into a half-precision floating-point number.	73
6.9	FCVT.H.WU: an instruction that converts an unsigned integer into a half-precision floating-point number.	74
6.10	FCVT.L.H: an instruction that converts a half-precision floating-point number to a signed long integer.	75
6.11	FCVT.LU.H: an instruction that converts a half-precision floating-point number to an unsigned long integer.	76

6.12	FCVT.S.H: an instruction that converts half-precision floating-point data to single precision floating-point data.	77
6.13	FCVT.W.H: an instruction that converts a half-precision floating-point number to a signed integer.	77
6.14	FCVT.WU.H: an instruction that converts a half-precision floating-point number to an unsigned integer.	78
6.15	FDIV.H: a half-precision floating-point division instruction.	79
6.16	FEQ.H: an equal instruction that compares two half-precision numbers.	80
6.17	FLE.H: a less than or equal to instruction that compares two half-precision floating-point numbers.	81
6.18	FLH: an instruction that loads half-precision floating-point data.	81
6.19	FLT.H: a less than instruction that compares two half-precision floating-point numbers. . . .	82
6.20	FMADD.H: a half-precision floating-point multiply-add instruction.	82
6.21	FMAX.H: a half-precision floating-point maximum instruction.	83
6.22	FMIN.H: a half-precision floating-point minimum instruction.	84
6.23	FMSUB.H: a half-precision floating-point multiply-subtract instruction.	85
6.24	FMUL.H: a half-precision floating-point multiply instruction.	86
6.25	FMV.H.X: a half-precision floating-point write transmit instruction.	87
6.26	FMV.X.H: a transmission instruction that reads half-precision floating-point registers. . . .	87
6.27	FNMADD.H: a half-precision floating-point negative multiply-add instruction.	88
6.28	FNMSUB.H: a half-precision floating-point negative multiply-subtract instruction.	89
6.29	FSGNJ.H: a half-precision floating-point sign-injection instruction.	90
6.30	FSGNHN.H: a half-precision floating-point sign-injection negate instruction.	90
6.31	FSGNJX.H: a half-precision floating-point sign-injection XOR instruction.	91
6.32	FSH: an instruction that stores half-precision floating point numbers.	92
6.33	FSQRT.H: an instruction that calculates the square root of the half-precision floating-point number.	92
6.34	FSUB.H: a half-precision floating-point subtract instruction.	93

Apart from the GCV instruction sets defined in the standard, C920 provides custom instruction sets, including the cache instruction set, synchronization instruction set, arithmetic operation instruction set, bitwise operation instruction set, storage instruction set, and half-precision floating-point instruction set.

Among these instruction sets, the cache instructions, synchronization instructions, arithmetic operation instructions, bitwise operation instructions, and storage instructions can be executed only when the value of `mxstatus.theadisaee` is 1. Otherwise, an instruction exception will occur. Half-precision floating-point instructions can be executed only when the value of `mstatus.fs` is 2'b00. Otherwise, an instruction exception will occur. The following section describes each instruction in these instruction sets.

Appendix B-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.

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

Syntax:

`dcache.call`

Operation:

Clears all page table entries in the L1 D-Cache and writes all dirty page table entries back into the next-level storage. You can perform this operation only on the current core.

Permission:

M mode/S 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		00001		00000		000		00000		0001011	

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 L1 D-Cache back into the next-level storage and invalidates all these page table entries.

Permission:

M mode/S 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	

1.3 DCACHE.CIPA: an instruction that clears 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 page table entries that match the specified physical addresses of the D-Cache or L2 Cache of rs1 back into the next-level storage and invalidates these page table entries. You can perform this operation on all cores and the L2 Cache.

Permission:

M mode/S 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						

1.4 DCACHE.CISW: an instruction that clears dirty 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 L1 Cache of rs1 back into the next-level storage and invalidates this page table entry. You can perform this operation only on the current core.

Permission:

M mode/S mode

Exception:

Invalid instruction.

Notes:

C920 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, `w` denotes 13. When the size of the D-Cache is 64 KB, `w` denotes 14, 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		

1.5 DCACHE.CIVA: an instruction that clears dirty page table entries that match the specified virtual addresses in the D-Cache and invalidates the D-Cache.

Syntax:

`dcache.civa rs1`

Operation:

Writes the page table entry that matches the specified virtual address from the D-Cache or L2 Cache of `rs1` back into the next-level storage and invalidates this page table entry. You can perform this operation on the current core and the L2 Cache. The sharing attribute of the virtual address determines whether you can perform this operation on other cores.

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction or error page during instruction loading.

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 and the value of `mxstatus.ucme` is 1, this instruction can be executed in U mode.
- If the value of `mxstatus.theadisaee` is 1 and the value of `mxstatus.ucme` is 0, 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	00111	rs1	000	00000	0001011						

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 from the D-Cache or L2 Cache of rs1 back into the next-level storage. You can perform this operation on all cores and the L2 Cache.

Permission:

M mode/S 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	01001	rs1	000	00000	0001011						

1.7 DCACHE.CPAL1: an instruction that clears dirty page table entries that match the specified physical addresses from in the L1 D-Cache.

Syntax:

dcache.cpal1 rs1

Operation: Writes the page table entry that matches the specified physical address from the D-Cache of rs1 back into the next-level storage. You can perform this operation on all cores and the L1 Cache.

Permission:

M mode/S 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						

1.8 DCACHE.CVA: an instruction that clears dirty page table entries that match the specified virtual addresses in the D-Cache.

Syntax:

`dcache.cva rs1`

Operation:

Writes the page table entry that matches the specified virtual address from the D-Cache or L2 Cache of `rs1` back into the next-level storage. You can perform this operation on the current core and the L2 Cache. The sharing attribute of the virtual address determines whether you can perform this operation on other cores.

Permission:

M mode/S mode

Exception:

Invalid instruction or error page during instruction loading.

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		00101		rs1		000		00000		0001011	

1.9 DCACHE.CVAL1: an instruction that clears dirty page table entries that match the specified virtual addresses in the L1 D-Cache.

Syntax:

dcache.cval1 rs1

Operation:

Writes the page table entry that matches the specified virtual address from the D-Cache of s1 back into the next-level storage. You can perform this operation on all cores and the L1 Cache.

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction or error page during instruction loading.

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 and the value of mxstatus.ucme is 0, 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		00100		rs1		000		00000		0001011	

1.10 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 or L2 Cache of rs1. You can perform this operation on all cores and the L2 Cache.

Permission:

M mode/S 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	

1.11 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 set and way. You can perform this operation only on the current core.

Permission:

M mode/S mode

Exception:

Invalid instruction.

Notes:

C920 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, `w` denotes 13. When the size of the D-Cache is 64 KB, `w` denotes 14, 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						

1.12 DCACHE.IVA: an instruction that invalidates the D-Cache based on the specified virtual address.

Syntax:

`dcache.iva rs1`

Operation:

Invalidates the page table entry that matches the specified virtual address from the D-Cache or L2 Cache of rs1. You can perform this operation on the current core and the L2 Cache. The sharing attribute of the virtual address determines whether you can perform this operation on other cores.

Permission:

M mode/S mode

Exception:

Invalid instruction or error page during instruction loading.

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	00110	rs1	000	00000	0001011						

1.13 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 L1 Cache. You can perform this operation only on the current core.

Permission:

M mode/S mode

Exception:

Invalid instruction.

Notes:

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

1.14 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. You can perform this operation only on the current core.

Permission:

M mode/S mode

Exception:

Invalid instruction.

Notes:

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

If the value of `mxstatus.theadisae` 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						

1.15 ICACHE.IALLS: an instruction that invalidates all page table entries in the I-Cache through broadcasting.

Syntax:

icache.ialls

Operation:

Invalidates all page table entries in the I-Cache and invalidates all page table entries in the I-Cache of other cores through broadcasting. You can perform this operation on all cores.

Permission:

M mode/S 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	10001	00000	000	00000	0001011						

1.16 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. You can perform this operation on all cores.

Permission:

M mode/S 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						

1.17 ICACHE.IVA: an instruction that invalidates page table entries that match the specified virtual addresses in the I-Cache.

Syntax:

`icache.iva rs1`

Operation:

Invalidates the page table entry that matches the specified virtual address in the I-Cache of `rs1`. You can perform this operation only on the current core. The sharing attribute of the virtual address determines whether you can perform this operation on other cores.

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction or error page during instruction loading.

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 and the value of `mxstatus.ucme` is 1, this instruction can be executed in U mode.

If the value of `mxstatus.theadisaee` is 1 and the value of `mxstatus.ucme` is 0, 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		10000		rs1		000		00000		0001011	

1.18 L2CACHE.CALL: an instruction that clears all dirty page table entries in the L2 Cache.

Syntax:

`l2cache.call`

Operation:

Writes all dirty page table entries from the L2 Cache back into the next-level storage.

Permission:

M mode/S 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		10101		00000		000		00000		0001011	

1.19 L2CACHE.CIALL: an instruction that clears all dirty page table entries in the L2 Cache and invalidates the L2 Cache.

Syntax:

`l2cache.ciall`

Operation:

Writes all dirty page table entries from the L2 Cache back into the next-level storage and invalidates all page table entries in the L2 Cache.

Permission:

M mode/S 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	10111	00000	000	00000	0001011						

1.20 L2CACHE.IALL: an instruction that invalidates the L2 Cache.

Syntax:

`l2cache.iall`

Operation:

Invalidates all page table entries in the L2 Cache.

Permission:

M mode/S mode

Exception:

Invalid instruction.

Notes:

- If the value of `mxstatus.cskisayee` is 0, executing this instruction causes an exception of invalid instruction.
- If the value of `mxstatus.cskisayee` 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	10110	00000	000	00000	0001011						

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

Syntax:

dcache.csw rs1

Operation:

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

Permission:

M mode/S mode

Exception:

Invalid instruction.

Notes:

C920 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, w denotes 13. When the size of the D-Cache is 64 KB, w denotes 14, 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:

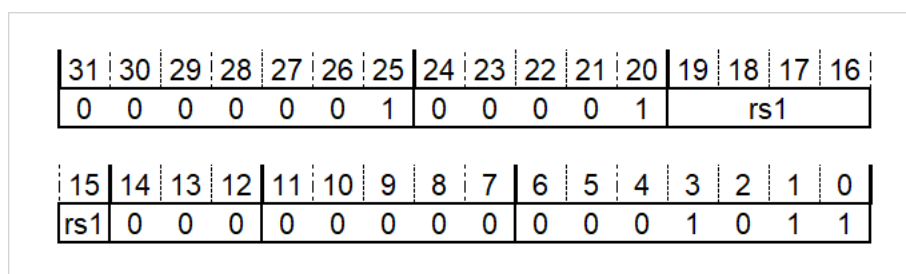


Fig. 1.1: DCACHE.CSW

Appendix B-2 Multi-core synchronization instructions

This synchronization instruction set extends multi-core synchronization instructions. Each instruction has 32 bits. Synchronization instructions in this instruction set are described in alphabetical order.

2.1 SFENCE.VMAS: a broadcast instruction that synchronizes the virtual memory address.

Syntax:

`sfence.vmas rs1,rs2`

Operation:

Invalidates and synchronizes page table entries in the virtual memory and broadcasts them to other cores in the cluster.

Permission:

M mode/S mode

Exception:

Invalid instruction.

Notes:

rs1 is the virtual address and rs2 is the address space identifier (ASID).

- If the value of rs1 is x0 and the value of rs2 is x0, invalidate all page table entries in the TLB and broadcast them to other cores in the cluster.
- If the value of rs1 is x0 and the value of rs2 is x0, invalidate all page table entries that match the virtual addresses of rs1 in the TLB and broadcast them to other cores in the cluster.
- If the value of rs1 is x0 and the value of rs2! is x0, invalidate all page table entries that match the ASIDs of rs2 in the TLB and broadcast them to other cores in the cluster.
- If the value of rs1! is x0 and the value of rs2! is x0, invalidate all page table entries that match the virtual addresses of rs1 and ASIDs of rs2 in the TLB and broadcast them to other cores in the cluster.

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
0000010		rs2		rs1		000		00000		0001011	

2.2 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/S 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	

2.3 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/S 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						

2.4 SYNC.IS: a broadcast instruction that synchronizes the clearing operation.

Syntax:

sync.is

Operation:

Ensures that all preceding instructions retire earlier than this instruction and all subsequent instructions retire later than this instruction. Clears the pipeline when this instruction retires and broadcasts the request to other cores.

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Instruction format:

31	25	24	20	19	15	14	12	11	7	6	0
0000000	11011	00000	000	00000	0001011						

2.5 SYNC.S: a broadcast instruction that performs a synchronization operation.

Syntax:

sync.s

Operation:

Ensures that all preceding instructions retire earlier than this instruction and all subsequent instructions retire later than this instruction, and broadcasts the request to other cores.

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Instruction format:

31	25	24	20	19	15	14	12	11	7	6	0
0000000	11001	00000	000	00000	0001011						

Appendix B-3 Arithmetic operation instructions

The arithmetic operation instruction set extends arithmetic operation instructions. Each instruction has 32 bits.

Arithmetic operation instructions in this instruction set are described in alphabetical order.

3.1 ADDSL: an add register instruction that shifts registers.

Syntax:

addsl rd rs1, rs2, imm2

Operation:

$rd \leftarrow rs1 + rs2 \ll imm2$

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
00000	imm2	rs2	rs1	001	rd	0001011							

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

Syntax:

mula rd, rs1, rs2

Operation:

$rd \leftarrow rd + (rs1 * rs2)[63:0]$

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
00100	00			rs2		rs1		001		rd		0001011	

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

Syntax:

mulah rd, rs1, rs2

Operation:

$tmp[31:0] \leftarrow rd[31:0] + (rs1[15:0] * rs2[15:0])$

$rd \leftarrow \text{sign_extend}(tmp[31:0])$

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
00101	00			rs2		rs1		001		rd		0001011	

3.4 MULAW: an instruction that performs a multiply-accumulate operation on lower 32 bits.

Syntax:

```
mulaw rd, rs1, rs2
```

Operation:

$$\text{tmp}[31:0] \leftarrow \text{rd}[31:0] + (\text{rs1}[31:0] * \text{rs2}[31:0])[31:0]$$

$$\text{rd} \leftarrow \text{sign_extend}(\text{tmp}[31:0])$$

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
00100	10			rs2		rs1		001		rd		0001011	

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

Syntax:

```
mults rd, rs1, rs2
```

Operation:

$$\text{rd} \leftarrow \text{rd} - (\text{rs1} * \text{rs2})[63:0]$$

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
00100	01			rs2		rs1		001		rd		0001011	

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

Syntax:

```
mulsh rd, rs1, rs2
```

Operation:

$$\text{tmp}[31:0] \leftarrow \text{rd}[31:0] - (\text{rs1}[15:0] * \text{rs2}[15:0])$$

$$\text{rd} \leftarrow \text{sign_extend}(\text{tmp}[31:0])$$

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
00101	01	rs2				rs1				001	rd		0001011

3.7 MULSW: an instruction that performs a multiply-subtraction operation on lower 32 bits.

Syntax:

```
mulaw rd, rs1, rs2
```

Operation:

$$\text{tmp}[31:0] \leftarrow \text{rd}[31:0] - (\text{rs1}[31:0] * \text{rs2}[31:0])$$

$$\text{rd} \leftarrow \text{sign_extend}(\text{tmp}[31:0])$$

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
00100	11	rs2				rs1				001	rd		0001011

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

Syntax:

```
mveqz rd, rs1, rs2
```

Operation: if ($rs2 == 0$)

```
rd  $\leftarrow$  rs1
```

```
else
```

```
rd  $\leftarrow$  rd
```

Permission:

M mode/S 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							

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

Syntax:

```
mvnez rd, rs1, rs2
```

Operation:

```
if ( $rs2 \neq 0$ )
```

```
rd  $\leftarrow$  rs1
```

```
else
```

```
rd  $\leftarrow$  rd
```

Permission:

M mode/S 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

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

Syntax:

srri rd, rs1, imm6

Operation:

$rd \leftarrow rs1 \gg \gg \gg \text{imm6}$

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/S mode/U mode

Exception:

Invalid instruction.

Instruction format:

31	26	25	20	19	15	14	12	11	7	6	0
000100		imm6			rs1		001		rd		0001011

3.11 SRRIW: an instruction that implements a cyclic right shift operation on a linked list of low 32 bits of registers.

Syntax:

srriw rd, rs1, imm5

Operation:

$rd \leftarrow \text{sign_extend}(rs1[31:0] \gg \gg \gg \text{imm5})$

Shifts the original value of rs1[31:0] 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/S mode/U mode

Exception:

Invalid instruction.

Instruction format:

31	25	24	20	19	15	14	12	11	7	6	0
0001010				imm5		rs1		001	rd		0001011

Appendix B-4 Bitwise operation instructions

The bitwise operation instruction set extends bitwise operation instructions. Each instruction has 32 bits.

Bitwise operation instructions in this instruction set are described in alphabetical order.

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

Syntax:

ext rd, rs1, imm1,imm2

Operation:

$rd \leftarrow \text{sign_extend}(rs1[imm1:imm2])$

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Note:

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

Instruction format:

31	26	25	20	19	15	14	12	11	7	6	0
imm1			imm2			rs1		010	rd		0001011

4.2 EXTU: a zero extension instruction that extracts consecutive bits of a register.

Syntax:

```
extu rd, rs1, imm1,imm2
```

Operation:

$$rd \leftarrow \text{zero_extend}(rs1[imm1:imm2])$$

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Note:

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

Instruction format:

31	26	25	20	19	15	14	12	11	7	6	0
imm1			imm2			rs1		011	rd		0001011

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 64 is returned.

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
10000	10	00000	rs1	001	rd	0001011							

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 index of this bit back into rd. If the highest bit of rs1 is 1, the result 0 is returned. If all the bits in rs1 are 1, the result 64 is returned.

Permission:

M mode/S 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							

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

Syntax:

rev rd, rs1

Operation:

rd[63:56] \leftarrow rs1[7:0]

rd[55:48] \leftarrow rs1[15:8]

rd[47:40] \leftarrow rs1[23:16]

rd[39:32] \leftarrow rs1[31:24]

rd[31:24] \leftarrow rs1[39:32]

$$\text{rd}[23:16] \leftarrow \text{rs1}[47:40]$$

$$\text{rd}[15:8] \leftarrow \text{rs1}[55:48]$$

$$\text{rd}[7:0] \leftarrow \text{rs1}[63:56]$$
Permission:

M mode/S 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	

4.6 REVW: an instruction that reverses the byte order in a low 32-bit word.

Syntax:

revw rd, rs1

Operation:

$$\text{tmp}[31:24] \leftarrow \text{rs1}[7:0]$$

$$\text{tmp}[23:16] \leftarrow \text{rs1}[15:8]$$

$$\text{tmp}[15:8] \leftarrow \text{rs1}[23:16]$$

$$\text{tmp}[7:0] \leftarrow \text{rs1}[31:24]$$

$$\text{rd} \leftarrow \text{sign_extend}(\text{tmp}[31:0])$$
Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
10010	00			00000		rs1		001		rd		0001011	

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

Syntax:

```
tst rd, rs1, imm6
```

Operation:

```
if(rs1[imm6] == 1)
```

```
    rd ← 1
```

```
else
```

```
    rd ← 0
```

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Instruction format:

31	26	25	20	19	15	14	12	11	7	6	0
100010	imm6				rs1		001	rd		0001011	

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

Syntax:

```
tstnbz rd, rs1
```

Operation:

```
rd[63:56] ← (rs1[63:56] == 0) ? 8' hff : 8' h0
```

```
rd[55:48] ← (rs1[55:48] == 0) ? 8' hff : 8' h0
```

```
rd[47:40] ← (rs1[47:40] == 0) ? 8' hff : 8' h0
```

```
rd[39:32] ← (rs1[39:32] == 0) ? 8' hff : 8' h0
```

```
rd[31:24] ← (rs1[31:24] == 0) ? 8' hff : 8' h0
```

```
rd[23:16] ← (rs1[23:16] == 0) ? 8' hff : 8' h0
```

```
rd[15:8] ← (rs1[15:8] == 0) ? 8' hff : 8' h0
```

```
rd[7:0] ← (rs1[7:0] == 0) ? 8' hff : 8' h0
```

Permission:

M mode/S 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							

Appendix B-5 Storage instructions

The storage instruction set extends storage instructions. Each instruction has 32 bits.

Storage instructions in this instruction set are described in alphabetical order.

5.1 FLRD: a load doubleword instruction that shifts floating-point registers.

Syntax:

`fldr rd, rs1, rs2, imm2`

Operation:

$rd \leftarrow \text{mem}[(rs1+rs2 \ll \text{imm2})+7: (rs1+rs2 \ll \text{imm2})]$

Permission:

M mode/S mode/U mode

Exception:

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

Note:

If the value of `mxstatus.theadisaee` is 1' b0 or the value of `mstatus.fs` is 2' b00, executing this instruction causes an exception of invalid instruction.

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
01100	imm2	rs2	rs1	110	rd	0001011							

5.2 FLRW: a load word instruction that shifts floating-point registers.**Syntax:**

flrw rd, rs1, rs2, imm2

Operation:

$rd \leftarrow \text{one_extend}(\text{mem}[(rs1+rs2 \ll \text{imm2})+3: (rs1+rs2 \ll \text{imm2})])$

Permission:

M mode/S mode/U mode

Exception:

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

Note:

If the value of mxstatus.theadisaee is 1' b0 or the value of mstatus.fs is 2' b00, executing this instruction causes an exception of invalid instruction.

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
01000	imm2	rs2	rs1	110	rd	0001011							

5.3 FLURD: a load doubleword instruction that shifts low 32 bits of floating-point registers.**Syntax:**

flurd rd, rs1, rs2, imm2

Operation:

$rd \leftarrow \text{mem}[(rs1+rs2[31:0] \ll \text{imm2})+7: (rs1+rs2[31:0] \ll \text{imm2})]$

Permission:

M mode/S mode/U mode

Exception:

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

Notes:

rs2[31:0] specifies an unsigned value. 0s are added to the high bits [63:32] for address calculation.

If the value of mxstatus.theadisaee is 1' b0 or the value of mstatus.fs is 2' b00, executing this instruction causes an exception of invalid instruction.

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
01110	imm2	rs2	rs1	110	rd	0001011							

5.4 FLURW: a load word instruction that shifts low 32 bits of floating-point registers.

Syntax:

```
flurw rd, rs1, rs2, imm2
```

Operation:

$$rd \leftarrow \text{one_extend}(\text{mem}[(rs1+rs2[31:0] \ll \text{imm2})+3: (rs1+rs2[31:0] \ll \text{imm2})])$$
Permission:

M mode/S mode/U mode

Exception:

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

Notes:

rs2[31:0] specifies an unsigned value. 0s are added to the high bits [63:32] for address calculation.

If the value of mxstatus.theadisaee is 1' b0 or the value of mstatus.fs is 2' b00, executing this instruction causes an exception of invalid instruction.

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
01010	imm2	rs2	rs1	110	rd	0001011							

5.5 FSRD: a store doubleword instruction that shifts floating-point registers.

Syntax:

```
fsrd rd, rs1, rs2, imm2
```

Operation:

$$\text{mem}[(\text{rs1} + \text{rs2} \ll \text{imm2}) + 7: (\text{rs1} + \text{rs2} \ll \text{imm2})] \leftarrow \text{rd}[63:0]$$
Permission:

M mode/S mode/U mode

Exception:

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

Note:

If the value of `mxstatus.theadisaee` is 1' b0 or the value of `mstatus.fs` is 2' b00, executing this instruction causes an exception of invalid instruction.

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
01100	imm2	rs2	rs1	111	rd	0001011							

5.6 FSRW: a store word instruction that shifts floating-point registers.

Syntax:

fsrw rd, rs1, rs2, imm2

Operation:

$$\text{mem}[(\text{rs1} + \text{rs2} \ll \text{imm2}) + 3: (\text{rs1} + \text{rs2} \ll \text{imm2})] \leftarrow \text{rd}[31:0]$$
Permission:

M mode/S mode/U mode

Exception:

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

Note:

If the value of `mxstatus.theadisaee` is 1' b0 or the value of `mstatus.fs` is 2' b00, executing this instruction causes an exception of invalid instruction.

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
01000	imm2	rs2	rs1	111	rd	0001011							

5.7 FSURD: a store doubleword instruction that shifts low 32 bits of floating-point registers.

Syntax:

fsurd rd, rs1, rs2, imm2

Operation:

$\text{mem}[(\text{rs1} + \text{rs2}[31:0] \ll \text{imm2}) + 7: (\text{rs1} + \text{rs2}[31:0] \ll \text{imm2})] \leftarrow \text{rd}[63:0]$

Permission:

M mode/S mode/U mode

Exception:

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

Notes:

rs2[31:0] specifies an unsigned value. 0s are added to the high bits [63:32] for address calculation.

If the value of mxstatus.theadisae is 1' b0 or the value of mstatus.fs is 2' b00, executing this instruction causes an exception of invalid instruction.

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0	
01110		imm2			rs2		rs1		111		rd		0001011	

5.8 FSURW: a store word instruction that shifts low 32 bits of floating-point registers.

Syntax:

fsurw rd, rs1, rs2, imm2

Operation:

$\text{mem}[(\text{rs1} + \text{rs2}[31:0] \ll \text{imm2}) + 3: (\text{rs1} + \text{rs2}[31:0] \ll \text{imm2})] \leftarrow \text{rd}[31:0]$

Permission:

M mode/S mode/U mode

Exception:

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

Notes:

rs2[31:0] specifies an unsigned value. 0s are added to the high bits [63:32] for address calculation.

If the value of mxstatus.theadisaee is 1' b0 or the value of mstatus.fs is 2' b00, executing this instruction causes an exception of invalid instruction.

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
01010	imm2	rs2	rs1	111	rd	0001011							

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

Syntax:

lbia rd, (rs1), imm5,imm2

Operation:

$rd \leftarrow \text{sign_extend}(\text{mem}[\text{rs1}])$

$\text{rs1} \leftarrow \text{rs1} + \text{sign_extend}(\text{imm5} \ll \text{imm2})$

Permission:

M mode/S 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
00011	imm2	imm5	rs1	100	rd	0001011							

5.10 LBIB: a load byte instruction that auto-increments the base address and extends signed bits.

Syntax:

lbib rd, (rs1), imm5,imm2

Operation:

$$rs1 \leftarrow rs1 + \text{sign_extend}(\text{imm5} \ll \text{imm2})$$

$$rd \leftarrow \text{sign_extend}(\text{mem}[rs1])$$
Permission:

M mode/S 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
00001	imm2	imm5	rs1	100	rd	0001011							

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

Syntax:

lbuia rd, (rs1), imm5,imm2

Operation:

$$rd \leftarrow \text{zero_extend}(\text{mem}[rs1])$$

$$rs1 \leftarrow rs1 + \text{sign_extend}(\text{imm5} \ll \text{imm2})$$
Permission:

M mode/S 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
10011	imm2	imm5	rs1	100	rd	0001011							

5.12 LBUIB: a load byte instruction that auto-increments the base address and extends zero bits.

Syntax:

```
lbuib rd, (rs1), imm5,imm2
```

Operation:

$$rs1 \leftarrow rs1 + \text{sign_extend}(imm5 \ll imm2)$$

$$rd \leftarrow \text{zero_extend}(\text{mem}[rs1])$$

Permission:

M mode/S 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
10001		imm2		imm5		rs1		100		rd		0001011	

5.13 LDD: an instruction that loads double registers.

Syntax:

```
ldd rd1,rd2, (rs1),imm2
```

Operation:

$$\text{address} \leftarrow rs1 + \text{zero_extend}(imm2 \ll 4)$$

$$rd1 \leftarrow \text{mem}[\text{address}+7:\text{address}]$$

$$rd2 \leftarrow \text{mem}[\text{address}+15:\text{address}+8]$$

Permission:

M mode/S mode/U mode

Exception:

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

Note:

The values of rd1, rd2, and rs1 must not be the same.

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
11111	imm2	rd2	rs1	100	rd1	0001011							

5.14 LDIA: a base-address auto-increment instruction that loads doublewords and extends signed bits.

Syntax:

ldia rd, (rs1), imm5,imm2

Operation:

$rd \leftarrow \text{sign_extend}(\text{mem}[\text{rs1}+7:\text{rs1}])$

$\text{rs1} \leftarrow \text{rs1} + \text{sign_extend}(\text{imm5} \ll \text{imm2})$

Permission:

M mode/S 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
01111	imm2	imm5	rs1	100	rd	0001011							

5.15 LDIB: a load doubleword instruction that auto-increments the base address and extends the signed bits.

Syntax:

ldib rd, (rs1), imm5,imm2

Operation:

$\text{rs1} \leftarrow \text{rs1} + \text{sign_extend}(\text{imm5} \ll \text{imm2})$

$rd \leftarrow \text{sign_extend}(\text{mem}[\text{rs1}+7:\text{rs1}])$

Permission:

M mode/S 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
01101	imm2			imm5			rs1		100		rd		0001011

5.16 LHIA: a base-address auto-increment instruction that loads half-words and extends signed bits.

Syntax:

lhia rd, (rs1), imm5,imm2

Operation: $rd \leftarrow \text{sign_extend}(\text{mem}[\text{rs1}+1:\text{rs1}])$ $\text{rs1} \leftarrow \text{rs1} + \text{sign_extend}(\text{imm5} \ll \text{imm2})$ **Permission:**

M mode/S 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
00111	imm2			imm5			rs1		100		rd		0001011

5.17 LHIB: a load halfword instruction that auto-increments the base address and extends signed bits.

Syntax:

lhib rd, (rs1), imm5,imm2

Operation:

$rs1 \leftarrow rs1 + \text{sign_extend}(\text{imm5} \ll \text{imm2})$

$rd \leftarrow \text{sign_extend}(\text{mem}[rs1+1:rs1])$

Permission:

M mode/S 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
00101	imm2			imm5			rs1		100	rd		0001011	

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

Syntax:

lhuiA rd, (rs1), imm5,imm2

Operation:

$rd \leftarrow \text{zero_extend}(\text{mem}[rs1+1:rs1])$

$rs1 \leftarrow rs1 + \text{sign_extend}(\text{imm5} \ll \text{imm2})$

Permission:

M mode/S 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
10111	imm2			imm5			rs1		100	rd		0001011	

5.19 LHUIB: a load halfword instruction that auto-increments the base address and extends zero bits.

Syntax:

lhuib rd, (rs1), imm5,imm2

Operation:

$rs1 \leftarrow rs1 + \text{sign_extend}(\text{imm5} \ll \text{imm2})$

$rd \leftarrow \text{zero_extend}(\text{mem}[rs1+1:rs1])$

Permission:

M mode/S 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
10101	imm2	imm5	rs1	100	rd	0001011							

5.20 LRB: a load byte instruction that shifts registers and extends signed bits.

Syntax:

lrb rd, rs1, rs2, imm2

Operation:

$rd \leftarrow \text{sign_extend}(\text{mem}[(rs1+rs2 \ll \text{imm2})])$

Permission:

M mode/S mode/U mode

Exception:

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

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
00000	imm2	rs2	rs1	100	rd	0001011							

5.21 LRBU: a load byte instruction that shifts registers and extends zero bits.

Syntax:

lrbu rd, rs1, rs2, imm2

Operation:

$rd \leftarrow \text{zero_extend}(\text{mem}[(rs1+rs2 \ll imm2)])$

Permission:

M mode/S mode/U mode

Exception:

Unaligned access, access error, page 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							

5.22 LRD: a load doubleword instruction that shifts registers.

Syntax:

lrd rd, rs1, rs2, imm2

Operation:

$rd \leftarrow \text{mem}[(rs1+rs2 \ll imm2)+7: (rs1+rs2 \ll imm2)]$

Permission:

M mode/S mode/U mode

Exception:

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

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
01100	imm2	rs2	rs1	100	rd	0001011							

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

Syntax:

lrh rd, rs1, rs2, imm2

Operation:

$rd \leftarrow \text{sign_extend}(\text{mem}[(rs1+rs2 \ll \text{imm2})+1: (rs1+rs2 \ll \text{imm2})])$

Permission:

M mode/S mode/U mode

Exception:

Unaligned access, access error, page 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							

5.24 LRHU: a load halfword instruction that shifts registers and extends zero bits.

Syntax:

lrhu rd, rs1, rs2, imm2

Operation:

$rd \leftarrow \text{zero_extend}(\text{mem}[(rs1+rs2 \ll \text{imm2})+1: (rs1+rs2 \ll \text{imm2})])$

Permission:

M mode/S mode/U mode

Exception:

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

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
10100	imm2	rs2	rs1	100	rd	0001011							

5.25 LRW: a load word instruction that shifts registers and extends signed bits.

Syntax:

lrw rd, rs1, rs2, imm2

Operation:

$rd \leftarrow \text{sign_extend}(\text{mem}[(rs1+rs2 \ll \text{imm2})+3: (rs1+rs2 \ll \text{imm2})])$

Permission:

M mode/S mode/U mode

Exception:

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

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
01000	imm2	rs2	rs1	100	rd	0001011							

5.26 LRWU: a load word instruction that shifts registers and extends zero bits.

Syntax:

lrwu rd, rs1, rs2, imm2

Operation:

$rd \leftarrow \text{zero_extend}(\text{mem}[(rs1+rs2 \ll \text{imm2})+3: (rs1+rs2 \ll \text{imm2})])$

Permission:

M mode/S mode/U mode

Exception:

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

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
11000	imm2	rs2	rs1	100	rd	0001011							

5.27 LURB: a load byte instruction that shifts low 32 bits of registers and extends signed bits.

Syntax:

`lurb rd, rs1, rs2, imm2`

Operation:

$rd \leftarrow \text{sign_extend}(\text{mem}[(rs1 + rs2[31:0] \ll \text{imm2})])$

Permission:

M mode/S mode/U mode

Exception:

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

Note:

$rs2[31:0]$ specifies an unsigned value. 0s are added to the high bits [63:32] for address calculation.

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0	
00010		imm2			rs2		rs1		100		rd		0001011	

5.28 LURBU: a load byte instruction that shifts low 32 bits of registers and extends zero bits.

Syntax:

`lurbu rd, rs1, rs2, imm2`

Operation:

$rd \leftarrow \text{zero_extend}(\text{mem}[(rs1 + rs2[31:0] \ll \text{imm2})])$

Permission:

M mode/S mode/U mode

Exception:

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

Note:

$rs2[31:0]$ specifies an unsigned value. 0s are added to the high bits [63:32] for address calculation.

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
10010	imm2	rs2	rs1	100	rd	0001011							

5.29 LURD: a load doubleword instruction that shifts low 32 bits of registers.

Syntax:

`lurd rd, rs1, rs2, imm2`

Operation:

$rd \leftarrow \text{mem}[(rs1 + rs2[31:0] \ll \text{imm2}) + 7: (rs1 + rs2[31:0] \ll \text{imm2})]$

Permission:

M mode/S mode/U mode

Exception:

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

Note:

$rs2[31:0]$ specifies an unsigned value. 0s are added to the high bits [63:32] for address calculation.

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
01110	imm2	rs2	rs1	100	rd	0001011							

5.30 LURH: a load halfword instruction that shifts low 32 bits of registers and extends signed bits.

Syntax:

`lurh rd, rs1, rs2, imm2`

Operation:

$rd \leftarrow \text{sign_extend}(\text{mem}[(rs1 + rs2[31:0] \ll \text{imm2}) + 1: (rs1 + rs2[31:0] \ll \text{imm2})])$

Permission:

M mode/S mode/U mode

Exception:

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

Note:

rs2[31:0] specifies an unsigned value. 0s are added to the high bits [63:32] for address calculation.

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
00110	imm2	rs2	rs1	100	rd	0001011							

5.31 LURHU: a load halfword instruction that shifts low 32 bits of registers and extends zero bits.

Syntax:

lurhu rd, rs1, rs2, imm2

Operation:

$rd \leftarrow \text{zero_extend}(\text{mem}[(rs1 + rs2[31:0] \ll imm2) + 1:$

$(rs1 + rs2[31:0] \ll imm2)])$

Permission:

M mode/S mode/U mode

Exception:

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

Note:

rs2[31:0] specifies an unsigned value. 0s are added to the high bits [63:32] for address calculation.

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
10110	imm2	rs2	rs1	100	rd	0001011							

5.32 LURW: a load word instruction that shifts low 32 bits of registers and extends signed bits.

Syntax:

lurw rd, rs1, rs2, imm2

Operation:

$rd \leftarrow \text{sign_extend}(\text{mem}[(rs1 + rs2[31:0] \ll imm2) + 3:$

$(rs1 + rs2[31:0] \ll imm2)])$

Permission:

M mode/S mode/U mode

Exception:

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

Note:

rs2[31:0] specifies an unsigned value. 0s are added to the high bits [63:32] for address calculation.

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
01010	imm2	rs2	rs1	100	rd	0001011							

5.33 LURWU: a load word instruction that shifts 32 bits of registers and extends zero bits.

Syntax:

lwd rd1, rd2, (rs1),imm2

Operation: $address \leftarrow rs1 + zero_extend(imm2 \ll 3)$ $rd1 \leftarrow sign_extend(mem[address+3: address])$ $rd2 \leftarrow sign_extend(mem[address+7: address+4])$ **Permission:**

M mode/S mode/U mode

Exception:

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

Note:

The values of rd1, rd2, and rs1 must not be the same.

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
11010	imm2	rs2	rs1	100	rd	0001011							

5.34 LWD: a load word instruction that loads double registers and extends signed bits.

Syntax:

lwd rd, imm7(rs1)

Operation:

address \leftarrow rs1 + sign_extend(imm7)

rd \leftarrow sign_extend(mem[address+31: address])

rd+1 \leftarrow sign_extend(mem[address+63: address+32])

Permission:

M mode/S mode/U mode

Exception:

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

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
11100	imm2	rd2	rs1	100	rd1	0001011							

5.35 LWIA: a base-address auto-increment instruction that extends signed bits and loads words.

Syntax:

lwia rd, (rs1), imm5,imm2

Operation:

rd \leftarrow sign_extend(mem[rs1+3:rs1])

rs1 \leftarrow rs1 + sign_extend(imm5 << imm2)

Permission:

M mode/S 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
01011		imm2		imm5		rs1		100		rd		0001011	

5.36 LWIB: a load word instruction that auto-increments the base address and extends signed bits.

Syntax:

lwib rd, (rs1), imm5,imm2

Operation:

$rs1 \leftarrow rs1 + \text{sign_extend}(imm5 \ll imm2)$

$rd \leftarrow \text{sign_extend}(\text{mem}[rs1+3:rs1])$

Permission:

M mode/S 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	

5.37 LWUD: a load word instruction that loads double registers and extends zero bits.

Syntax:

lwud rd1,rd2, (rs1),imm2

Operation:

$\text{address} \leftarrow rs1 + \text{zero_extend}(imm2 \ll 3)$

$rd1 \leftarrow \text{zero_extend}(\text{mem}[\text{address}+3: \text{address}])$

$rd2 \leftarrow \text{zero_extend}(\text{mem}[\text{address}+7: \text{address}+4])$

Permission:

M mode/S mode/U mode

Exception:

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

Note:

The values of rd1, rd2, and rs1 must not be the same.

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
11110	imm2			rd2			rs1		100		rd1		0001011

5.38 LWUIA: a base-address auto-increment instruction that extends zero bits and loads words.

Syntax:

lwuia rd, (rs1), imm5,imm2

Operation:

$rd \leftarrow \text{zero_extend}(\text{mem}[\text{rs1}+3:\text{rs1}])$

$\text{rs1} \leftarrow \text{rs1} + \text{sign_extend}(\text{imm5} \ll \text{imm2})$

Permission:

M mode/S 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
11011	imm2			imm5			rs1		100		rd		0001011

5.39 LWUIB: a load word instruction that auto-increments the base address and extends zero bits.

Syntax:

lwuib rd, (rs1), imm5,imm2

Operation:

$$rs1 \leftarrow rs1 + \text{sign_extend}(\text{imm5} \ll \text{imm2})$$

$$rd \leftarrow \text{zero_extend}(\text{mem}[rs1+3:rs1])$$
Permission:

M mode/S 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
11001	imm2	imm5			rs1		100		rd			0001011	

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

Syntax:
`sbia rs2, (rs1), imm5,imm2`
Operation:

$$\text{mem}[rs1] \leftarrow rs2[7:0]$$

$$rs1 \leftarrow rs1 + \text{sign_extend}(\text{imm5} \ll \text{imm2})$$
Permission:

M mode/S mode/U mode

Exception:

Unaligned access, access error, page 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	

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

Syntax:

sbib rs2, (rs1), imm5,imm2

Operation:

$rs1 \leftarrow rs1 + \text{sign_extend}(\text{imm5} \ll \text{imm2})$

$\text{mem}[rs1] \leftarrow rs2[7:0]$

Permission:

M mode/S mode/U mode

Exception:

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

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0			
00001				imm2		imm5			rs1		101		rs2		0001011	

5.42 SDD: an instruction that stores double registers.

Syntax:

sdd rd1,rd2, (rs1),imm2

Operation:

$\text{address} \leftarrow rs1 + \text{zero_extend}(\text{imm2} \ll 4)$

$\text{mem}[\text{address}+7:\text{address}] \leftarrow rd1$

$\text{mem}[\text{address}+15:\text{address}+8] \leftarrow rd2$

Permission:

M mode/S mode/U mode

Exception:

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

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
11111		imm2			rd2		rs1		101		rd1		0001011

5.43 SDIA: a base-address auto-increment instruction that stores doublewords.

Syntax:

sdia rs2, (rs1), imm5,imm2

Operation:

$\text{mem}[\text{rs1}+7:\text{rs1}] \leftarrow \text{rs2}[63:0]$

$\text{rs1} \leftarrow \text{rs1} + \text{sign_extend}(\text{imm5} \ll \text{imm2})$

Permission:

M mode/S mode/U mode

Exception:

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

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
01111	imm2	imm5	rs1	101	rs2	0001011							

5.44 SDIB: a store doubleword instruction that auto-increments the base address.

Syntax:

sdib rs2, (rs1), imm5,imm2

Operation:

$\text{rs1} \leftarrow \text{rs1} + \text{sign_extend}(\text{imm5} \ll \text{imm2})$

$\text{mem}[\text{rs1}+7:\text{rs1}] \leftarrow \text{rs2}[63:0]$

Permission:

M mode/S mode/U mode

Exception:

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

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
01101	imm2	imm5	rs1	101	rs2	0001011							

5.45 SHIA: a base-address auto-increment instruction that stores half-words.

Syntax:

shia rs2, (rs1), imm5,imm2

Operation:

$\text{mem}[\text{rs1}+1:\text{rs1}] \leftarrow \text{rs2}[15:0]$

$\text{rs1} \leftarrow \text{rs1} + \text{sign_extend}(\text{imm5} \ll \text{imm2})$

Permission:

M mode/S mode/U mode

Exception:

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

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
00111	imm2	imm5	rs1	101	rs2	0001011							

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

Syntax:

shib rs2, (rs1), imm5,imm2

Operation:

$\text{rs1} \leftarrow \text{rs1} + \text{sign_extend}(\text{imm5} \ll \text{imm2})$

$\text{mem}[\text{rs1}+1:\text{rs1}] \leftarrow \text{rs2}[15:0]$

Permission:

M mode/S mode/U mode

Exception:

Unaligned access, access error, page 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							

5.47 SRB: a store byte instruction that shifts registers.

Syntax:

srb rd, rs1, rs2, imm2

Operation:

$$\text{mem}[(\text{rs1} + \text{rs2} \ll \text{imm2})] \leftarrow \text{rd}[7:0]$$
Permission:

M mode/S mode/U mode

Exception:

Unaligned access, access error, page 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							

5.48 SRD: a store doubleword instruction that shifts registers.

Syntax:

srd rd, rs1, rs2, imm2

Operation:

$$\text{mem}[(\text{rs1} + \text{rs2} \ll \text{imm2}) + 7: (\text{rs1} + \text{rs2} \ll \text{imm2})] \leftarrow \text{rd}[63:0]$$
Permission:

M mode/S mode/U mode

Exception:

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

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
01100	imm2	rs2	rs1	101	rd	0001011							

5.49 SRH: a store halfword instruction that shifts registers.

Syntax:

srh rd, rs1, rs2, imm2

Operation:

$$\text{mem}[(\text{rs1} + \text{rs2} \ll \text{imm2}) + 1: (\text{rs1} + \text{rs2} \ll \text{imm2})] \leftarrow \text{rd}[15:0]$$
Permission:

M mode/S mode/U mode

Exception:

Unaligned access, access error, page 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							

5.50 SRW: a store word instruction that shifts registers.

Syntax:

srw rd, rs1, rs2, imm2

Operation:

$\text{mem}[(\text{rs1} + \text{rs2} \ll \text{imm2}) + 3: (\text{rs1} + \text{rs2} \ll \text{imm2})] \leftarrow \text{rd}[31:0]$

Permission:

M mode/S mode/U mode

Exception:

Unaligned access, access error, page 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							

5.51 SURB: a store byte instruction that shifts low 32 bits of registers.

Syntax:

surb rd, rs1, rs2, imm2

Operation:

$\text{mem}[(\text{rs1} + \text{rs2}[31:0] \ll \text{imm2})] \leftarrow \text{rd}[7:0]$

Permission:

M mode/S mode/U mode

Exception:

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

Note:

rs2[31:0] specifies an unsigned value. 0s are added to the high bits [63:32] for address calculation.

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
00010		imm2		rs2		rs1		101		rd		0001011	

5.52 SURD: a store doubleword instruction that shifts low 32 bits of registers.

Syntax:

surd rd, rs1, rs2, imm2

Operation:

$\text{mem}[(\text{rs1} + \text{rs2}[31:0] \ll \text{imm2}) + 7: (\text{rs1} + \text{rs2}[31:0] \ll \text{imm2})] \leftarrow \text{rd}[63:0]$

Permission:

M mode/S mode/U mode

Exception:

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

Note:

rs2[31:0] specifies an unsigned value. 0s are added to the high bits [63:32] for address calculation.

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
01110		imm2		rs2		rs1		101		rd		0001011	

5.53 SURH: a store halfword instruction that shifts low 32 bits of registers.

Syntax:

surh rd, rs1, rs2, imm2

Operation:

$\text{mem}[(\text{rs1} + \text{rs2}[31:0] \ll \text{imm2}) + 1: (\text{rs1} + \text{rs2}[31:0] \ll \text{imm2})] \leftarrow \text{rd}[15:0]$

Permission:

M mode/S mode/U mode

Exception:

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

Note:

rs2[31:0] specifies an unsigned value. 0s are added to the high bits [63:32] for address calculation.

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
00110	imm2	rs2	rs1	101	rd	0001011							

5.54 SURW: a store word instruction that shifts low 32 bits of registers.

Syntax:

surw rd, rs1, rs2, imm2

Operation:

$\text{mem}[(\text{rs1} + \text{rs2}[31:0] \ll \text{imm2}) + 3: (\text{rs1} + \text{rs2}[31:0] \ll \text{imm2})] \leftarrow \text{rd}[31:0]$

Permission:

M mode/S mode/U mode

Exception:

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

Note:

rs2[31:0] specifies an unsigned value. 0s are added to the high bits [63:32] for address calculation.

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
01010	imm2	rs2	rs1	101	rd	0001011							

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

Syntax:

swia rs2, (rs1), imm5, imm2

Operation:

$\text{mem}[\text{rs1} + 3: \text{rs1}] \leftarrow \text{rs2}[31:0]$

$\text{rs1} \leftarrow \text{rs1} + \text{sign_extend}(\text{imm5} \ll \text{imm2})$

Permission:

M mode/S mode/U mode

Exception:

Unaligned access, access error, page 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

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

Syntax:

swib rs2, (rs1), imm5,imm2

Operation:

$rs1 \leftarrow rs1 + \text{sign_extend}(imm5 \ll imm2)$

$\text{mem}[rs1+3:rs1] \leftarrow rs2[31:0]$

Permission:

M mode/S mode/U mode

Exception:

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

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
01001	imm2			imm5			rs1		101		rs2		0001011

5.57 SWD: an instruction that stores the low 32 bits of double registers.

Syntax:

swd rd1,rd2,(rs1),imm2

Operation:

$\text{address} \leftarrow rs1 + \text{zero_extend}(imm2 \ll 3)$

$\text{mem}[\text{address}+3:\text{address}] \leftarrow rd1[31:0]$

$\text{mem}[\text{address}+7:\text{address}+4] \leftarrow rd2[31:0]$

Permission:

M mode/S mode/U mode

Exception:

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

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
11100	imm2	rd2	rs1	101	rd1	0001011							

Appendix B-6 Half-precision floating-point instructions

You can use instructions in this instruction set to process floating-point half-precision data. Each instruction has 32 bits. Instructions in this instruction set are described in alphabetical order.

6.1 FADD.H: a half-precision floating-point add instruction.

Syntax:

fadd.h fd, fs1, fs2, rm

Operation:

$fd \leftarrow fs1 + fs2$

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Affected flag bits:

Floating-point status bits NV, OF, and NX

Notes:

RM determines the round-off mode:

- 3' b000:Rounds off to the nearest even number. The corresponding instruction is fadd.h fd, fs1,fs2,rne.
- 3' b001:Rounds off to zero. The corresponding instruction is fadd.h fd, fs1,fs2,rtz.
- 3' b010:Rounds off to negative infinity. The corresponding instruction is fadd.h fd, fs1,fs2,rdn.
- 3' b011:Rounds off to positive infinity. The corresponding instruction is fadd.h fd, fs1,fs2,rup.
- 3' b100:Rounds off to the nearest large value. The corresponding instruction is fadd.h fd, fs1,fs2,rmm.
- 3' b101:This code is reserved and not used.
- 3' b110:This code is reserved and not used.
- 3' b111:Dynamically rounds off based on the rm bit of the floating-point register FCSR. The corresponding instruction is fadd.h fd, fs1,fs2.

Instruction format:

31	25	24	20	19	15	14	12	11	7	6	0
0000010				fs2		fs1		rm		fd	
										1010011	

6.2 FCLASS.H: a half-precision floating-point classification instruction.**Syntax:**

```
fclass.h rd, fs1
```

Operation:

```
if ( fs1 = -inf)
```

```
    rd ← 64' h1
```

```
if ( fs1 = -norm)
```

```
    rd ← 64' h2
```

```
if ( fs1 = -subnorm)
```

```
    rd ← 64' h4
```

```
if ( fs1 = -zero)
```

```
    rd ← 64' h8
```

```
if ( fs1 = +zero)
```

```
    rd ← 64' h10
```

if (fs1 = +subnorm)

rd \leftarrow 64' h20

if (fs1 = +norm)

rd \leftarrow 64' h40

if (fs1 = +inf)

rd \leftarrow 64' h80

if (fs1 = sNaN)

rd \leftarrow 64' h100

if (fs1 = qNaN)

rd \leftarrow 64' h200

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Affected flag bits:

None

Instruction format:

31	25	24	20	19	15	14	12	11	7	6	0
1110010	00000	fs1	001	rd	1010011						

6.3 FCVT.D.H: an instruction that converts half-precision floating-point data to double-precision floating-point data.

Syntax:

fcvt.d.h fd, fs1

Operation:

fd \leftarrow half_convert_to_double(fs1)

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Affected flag bits:

None

Instruction format:

31	25	24	20	19	15	14	12	11	7	6	0
0100001	00010	fs1	000	fd	1010011						

6.4 FCVT.H.D: an instruction that converts double-precision floating-point data to half-precision floating-point data.

Syntax:

fcvt.h.d fd, fs1, rm

Operation:

fd \leftarrow double_convert_to_half(fs1)

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Affected flag bits:

Floating-point status bits NV, OF, UF, and NX

Notes:

RM determines the round-off mode:

- 3' b000:Rounds off to the nearest even number. The corresponding instruction is fcvt.h.d fd,fs1,rne.
- 3' b001:Rounds off to zero. The corresponding instruction is fcvt.h.d fd,fs1,rtz.
- 3' b010:Rounds off to negative infinity. The corresponding instruction is fcvt.h.d fd,fs1,rdn.
- 3' b011:Rounds off to positive infinity. The corresponding instruction is fcvt.h.d fd,fs1,rup.
- 3' b100:Rounds off to the nearest large value. The corresponding instruction is fcvt.h.d fd,fs1,rmm.
- 3' b101:This code is reserved and not used.
- 3' b110:This code is reserved and not used.
- 3' b111:Dynamically rounds off based on the rm bit of the floating-point register FCSR. The corresponding instruction is fcvt.h.d fd, fs1.

Instruction format:

31	25	24	20	19	15	14	12	11	7	6	0
0100010	00001	fs1	rm	fd	1010011						

6.5 FCVT.H.L: an instruction that converts a signed long integer into a half-precision floating-point number.

Syntax:

fcvt.h.l fd, rs1, rm

Operation:

$fd \leftarrow \text{signed_long_convert_to_half}(rs1)$

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Affected flag bits:

Floating-point status bits NX and OF

Notes:

RM determines the round-off mode:

- 3' b000:Rounds off to the nearest even number. The corresponding instruction is fcvt.h.l fd,rs1,rne.
- 3' b001:Rounds off to zero. The corresponding instruction is fcvt.h.l fd,rs1,rtz.
- 3' b010:Rounds off to negative infinity. The corresponding instruction is fcvt.h.l fd,rs1,fdn.
- 3' b011:Rounds off to positive infinity. The corresponding instruction is fcvt.h.l fd,rs1,rup.
- 3' b100:Rounds off to the nearest large value. The corresponding instruction is fcvt.h.l fd,rs1,rmm.
- 3' b101:This code is reserved and not used.
- 3' b110:This code is reserved and not used.
- 3' b111:Dynamically rounds off based on the rm bit of the floating-point register FCSR. The corresponding instruction is fcvt.h.l fd, rs1.

Instruction format:

31	25	24	20	19	15	14	12	11	7	6	0
1101010	00010	rs1	rm	fd	1010011						

6.6 FCVT.H.LU: an instruction that converts an unsigned long integer into a half-precision floating-point number.

Syntax:

fcvt.h.lu fd, rs1, rm

Operation:

$fd \leftarrow \text{unsigned_long_convert_to_half_fp}(rs1)$

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Affected flag bits:

Floating-point status bits NX and OF

Notes:

RM determines the round-off mode:

- 3' b000:Rounds off to the nearest even number. The corresponding instruction is fcvt.h.lu fd,rs1,rne.
- 3' b001:Rounds off to zero. The corresponding instruction is fcvt.h.lu fd, rs1,rtz.
- 3' b010:Rounds off to negative infinity. The corresponding instruction is fcvt.h.lu fd, rs1,fdn.
- 3' b011:Rounds off to positive infinity. The corresponding instruction is fcvt.h.lu fd, rs1,rup.
- 3' b100:Rounds off to the nearest large value. The corresponding instruction is fcvt.h.lu fd, rs1,rmm.
- 3' b101:This code is reserved and not used.
- 3' b110:This code is reserved and not used.
- 3' b111:Dynamically rounds off based on the rm bit of the floating-point register FCSR. The corresponding instruction is fcvt.h.lu fd, rs1.

Instruction format:

31	25	24	20	19	15	14	12	11	7	6	0
1101010	00011	rs1	rm	fd	1010011						

6.7 FCVT.H.S: an instruction that converts single precision floating-point data to half-precision floating-point data.

Syntax:

fcvt.h.s fd, fs1, rm

Operation:

$fd \leftarrow \text{single_convert_to_half}(fs1)$

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Affected flag bits:

Floating-point status bits NV, OF, UF, and NX

Notes:

RM determines the round-off mode:

- 3' b000:Rounds off to the nearest even number. The corresponding instruction is fcvt.h.s fd,fs1,rne.
- 3' b001:Rounds off to zero. The corresponding instruction is fcvt.h.s fd,fs1,rtz.
- 3' b010:Rounds off to negative infinity. The corresponding instruction is fcvt.h.s fd,fs1,fdn.
- 3' b011:Rounds off to positive infinity. The corresponding instruction is fcvt.h.s fd,fs1,rup.
- 3' b100:Rounds off to the nearest large value. The corresponding instruction is fcvt.h.s fd,fs1,rmm.
- 3' b101:This code is reserved and not used.
- 3' b110:This code is reserved and not used.
- 3' b111:Dynamically rounds off based on the rm bit of the floating-point register FCSR. The corresponding instruction is fcvt.h.s fd, fs1.

Instruction format:

31	25	24	20	19	15	14	12	11	7	6	0
0100010	00000	fs1	rm	fd	1010011						

6.8 FCVT.H.W: an instruction that converts a signed integer into a half-precision floating-point number.

Syntax:

fcvt.h.w fd, rs1, rm

Operation:

$fd \leftarrow \text{signed_int_convert_to_half}(rs1)$

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Affected flag bits:

Floating-point status bits NX and OF

Notes:

RM determines the round-off mode:

- 3' b000:Rounds off to the nearest even number. The corresponding instruction is fcvt.h.w fd,rs1,rne.
- 3' b001:Rounds off to zero. The corresponding instruction is fcvt.h.w fd,rs1,rtz.
- 3' b010:Rounds off to negative infinity. The corresponding instruction is fcvt.h.w fd,rs1,fdn.
- 3' b011:Rounds off to positive infinity. The corresponding instruction is fcvt.h.w fd,rs1,rup.
- 3' b100:Rounds off to the nearest large value. The corresponding instruction is fcvt.h.w fd,rs1,rmm.
- 3' b101:This code is reserved and not used.
- 3' b110:This code is reserved and not used.
- 3' b111:Dynamically rounds off based on the rm bit of the floating-point register FCSR. The corresponding instruction is fcvt.h.w fd, rs1.

Instruction format:

31	25	24	20	19	15	14	12	11	7	6	0
1101010	00000	rs1	rm	fd	1010011						

6.9 FCVT.H.WU: an instruction that converts an unsigned integer into a half-precision floating-point number.

Syntax:

fcvt.h.wu fd, rs1, rm

Operation:

$fd \leftarrow \text{unsigned_int_convert_to_half_fp}(rs1)$

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Affected flag bits:

Floating-point status bits NX and OF

Notes:

RM determines the round-off mode:

- 3' b000:Rounds off to the nearest even number. The corresponding instruction is fcvt.h.wu fd,rs1,rne.
- 3' b001:Rounds off to zero. The corresponding instruction is fcvt.h.wu fd,rs1,rtz.
- 3' b010:Rounds off to negative infinity. The corresponding instruction is fcvt.h.wu fd,rs1,fdn.
- 3' b011:Rounds off to positive infinity. The corresponding instruction is fcvt.h.wu fd,rs1,rup.
- 3' b100:Rounds off to the nearest large value. The corresponding instruction is fcvt.h.wu fd,rs1,rmm.
- 3' b101:This code is reserved and not used.
- 3' b110:This code is reserved and not used.
- 3' b111:Dynamically rounds off based on the rm bit of the floating-point register FCSR. The corresponding instruction is fcvt.h.wu fd, rs1.

Instruction format:

31	25	24	20	19	15	14	12	11	7	6	0		
1101010				00001		rs1		rm		fd		1010011	

6.10 FCVT.L.H: an instruction that converts a half-precision floating-point number to a signed long integer.

Syntax:

fcvt.l.h rd, fs1, rm

Operation:

$rd \leftarrow \text{half_convert_to_signed_long}(fs1)$

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Affected flag bits:

Floating-point status bits NV and NX

Notes:

RM determines the round-off mode:

- 3' b000:Rounds off to the nearest even number. The corresponding instruction is fcvt.l.h rd,fs1,rne.
- 3' b001:Rounds off to zero. The corresponding instruction is fcvt.l.h rd,fs1,rtz.
- 3' b010:Rounds off to negative infinity. The corresponding instruction is fcvt.l.h rd,fs1,rdn.
- 3' b011:Rounds off to positive infinity. The corresponding instruction is fcvt.l.h rd,fs1,rup.
- 3' b100:Rounds off to the nearest large value. The corresponding instruction is fcvt.l.h rd,fs1,rmm.
- 3' b101:This code is reserved and not used.
- 3' b110:This code is reserved and not used.
- 3' b111:Dynamically rounds off based on the rm bit of the floating-point register FCSR. The corresponding instruction is fcvt.l.h rd, fs1.

Instruction format:

31	25	24	20	19	15	14	12	11	7	6	0		
1100010				00010		fs1		rm		rd		1010011	

6.11 FCVT.LU.H: an instruction that converts a half-precision floating-point number to an unsigned long integer.

Syntax:

fcvt.lu.h rd, fs1, rm

Operation:

$rd \leftarrow \text{half_convert_to_unsigned_long}(fs1)$

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Affected flag bits:

Floating-point status bits NV and NX

Notes:

RM determines the round-off mode:

- 3' b000:Rounds off to the nearest even number. The corresponding instruction is fcvt.lu.h rd,fs1,rne.
- 3' b001:Rounds off to zero. The corresponding instruction is fcvt.lu.h rd,fs1,rtz.
- 3' b010:Rounds off to negative infinity. The corresponding instruction is fcvt.lu.h rd,fs1,rdn.
- 3' b011:Rounds off to positive infinity. The corresponding instruction is fcvt.lu.h rd,fs1,rup.
- 3' b100:Rounds off to the nearest large value. The corresponding instruction is fcvt.lu.h rd,fs1,rmm.
- 3' b101:This code is reserved and not used.
- 3' b110:This code is reserved and not used.
- 3' b111:Dynamically rounds off based on the rm bit of the floating-point register FCSR. The corresponding instruction is fcvt.lu.h rd, fs1.

Instruction format:

31	25	24	20	19	15	14	12	11	7	6	0
1100010	00011	fs1	rm	rd	1010011						

6.12 FCVT.S.H: an instruction that converts half-precision floating-point data to single precision floating-point data.

Syntax:

```
fcvt.s.h fd, fs1
```

Operation:

```
fd ← half_convert_to_single(fs1)
```

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Affected flag bits:

None

Instruction format:

31	25	24	20	19	15	14	12	11	7	6	0
0100000	00010	fs1	000	fd	1010011						

6.13 FCVT.W.H: an instruction that converts a half-precision floating-point number to a signed integer.

Syntax:

```
fcvt.w.h rd, fs1, rm
```

Operation:

```
tmp ← half_convert_to_signed_int(fs1)
```

```
rd ← sign_extend(tmp)
```

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Affected flag bits:

Floating-point status bits NV and NX

Notes:

RM determines the round-off mode:

- 3' b000:Rounds off to the nearest even number. The corresponding instruction is fcvt.w.h rd,fs1,rne.
- 3' b001:Rounds off to zero. The corresponding instruction is fcvt.w.h rd,fs1,rtz.
- 3' b010:Rounds off to negative infinity. The corresponding instruction is fcvt.w.h rd,fs1,rdn.
- 3' b011:Rounds off to positive infinity. The corresponding instruction is fcvt.w.h rd,fs1,rup.
- 3' b100:Rounds off to the nearest large value. The corresponding instruction is fcvt.w.h rd,fs1,rmm.
- 3' b101:This code is reserved and not used.
- 3' b110:This code is reserved and not used.
- 3' b111:Dynamically rounds off based on the rm bit of the floating-point register FCSR. The corresponding instruction is fcvt.w.h rd, fs1.

Instruction format:

31	25	24	20	19	15	14	12	11	7	6	0
1100010	00000	fs1	rm	rd	1010011						

6.14 FCVT.WU.H: an instruction that converts a half-precision floating-point number to an unsigned integer.

Syntax:

fcvt.wu.h rd, fs1, rm

Operation:

tmp ← half_convert_to_unsigned_int(fs1)

rd ← sign_extend(tmp)

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Affected flag bits:

Floating-point status bits NV and NX

Notes:

RM determines the round-off mode:

- 3' b000:Rounds off to the nearest even number. The corresponding instruction is fcvt.wu.h rd,fs1,rne.
- 3' b001:Rounds off to zero. The corresponding instruction is fcvt.wu.h rd,fs1,rtz.
- 3' b010:Rounds off to negative infinity. The corresponding instruction is fcvt.wu.h rd,fs1,rdn.
- 3' b011:Rounds off to positive infinity. The corresponding instruction is fcvt.wu.h rd,fs1,rup.
- 3' b100:Rounds off to the nearest large value. The corresponding instruction is fcvt.wu.h rd,fs1,rmm.
- 3' b101:This code is reserved and not used.
- 3' b110:This code is reserved and not used.
- 3' b111:Dynamically rounds off based on the rm bit of the floating-point register FCSR. The corresponding instruction is fcvt.wu.h rd, fs1.

Instruction format:

31	25	24	20	19	15	14	12	11	7	6	0
1100010	00001	fs1	rm	rd	1010011						

6.15 FDIV.H: a half-precision floating-point division instruction.

Syntax:

fdiv.h fd, fs1, fs2, rm

Operation:

$fd \leftarrow fs1 / fs2$

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Affected flag bits:

Floating-point status bits NV, DZ, OF, UF, and NX

Notes:

RM determines the round-off mode:

- 3' b000:Rounds off to the nearest even number. The corresponding instruction is fddiv.h fs1,fs2,rne.

- 3' b001:Rounds off to zero. The corresponding instruction is `fdiv.h fd fs1,fs2,rtz`.
- 3' b010:Rounds off to negative infinity. The corresponding instruction is `fdiv.h fd, fs1,fs2,rdn`.
- 3' b011:Rounds off to positive infinity. The corresponding instruction is `fdiv.h fd, fs1,fs2,rup`.
- 3' b100:Rounds off to the nearest large value. The corresponding instruction is `fdiv.h fd, fs1,fs2,rm`.
- 3' b101:This code is reserved and not used.
- 3' b110:This code is reserved and not used.
- 3' b111:Dynamically rounds off based on the `rm` bit of the floating-point register `FCSR`. The corresponding instruction is `fdiv.h fd, fs1,fs2`.

Instruction format:

31	25	24	20	19	15	14	12	11	7	6	0
0001110				fs2		fs1		rm	fd		1010011

6.16 FEQ.H: an equal instruction that compares two half-precision numbers.

Syntax:

`feq.h rd, fs1, fs2`

Operation:

`if(fs1 == fs2)`

`rd ← 1`

`else`

`rd ← 0`

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Affected flag bit:

Floating-point status bit `NV`

Instruction format:

31	25	24	20	19	15	14	12	11	7	6	0
1010010				fs2		fs1		010	rd		1010011

6.17 FLE.H: a less than or equal to instruction that compares two half-precision floating-point numbers.

Syntax:

```
fle.h rd, fs1, fs2
```

Operation:

```
if(fs1 <= fs2)
```

```
    rd ← 1
```

```
else
```

```
    rd ← 0
```

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Affected flag bit:

Floating-point status bit NV

Instruction format:

31	25	24	20	19	15	14	12	11	7	6	0
1010010		fs2		fs1		000		rd		1010011	

6.18 FLH: an instruction that loads half-precision floating-point data.

Syntax:

```
flh fd, imm12(rs1)
```

Operation:

```
address ← rs1 + sign_extend(imm12)
```

```
fd[15:0] ← mem[(address+1):address]
```

```
fd[63:16] ← 48' hfffffffffff
```

Permission:

M mode/S mode/U mode

Exception:

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

Affected flag bits:

None

Instruction format:

31	20	19	15	14	12	11	7	6	0
imm12[11:0]				rs1		001	rd		0000111

6.19 FLT.H: a less than instruction that compares two half-precision floating-point numbers.

Syntax:

flt.h rd, fs1, fs2

Operation:

if(fs1 < fs2)

rd ← 1

else

rd ← 0

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Affected flag bit:

Floating-point status bit NV

Instruction format:

31	25	24	20	19	15	14	12	11	7	6	0
1010010			fs2		fs1		001	rd		1010011	

6.20 FMADD.H: a half-precision floating-point multiply-add instruction.

Syntax:

fmadd.h fd, fs1, fs2, fs3, rm

Operation:

$$fd \leftarrow fs1 * fs2 + fs3$$
Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Affected flag bits:

Floating-point status bits NV, OF, UF, and IX

Notes:

RM determines the round-off mode:

- 3' b000:Rounds off to the nearest even number. The corresponding instruction is `fmadd.h fd,fs1, fs2, fs3, rne`.
- 3' b001:Rounds off to zero. The corresponding instruction is `fmadd.h fd,fs1, fs2, fs3, rtz`.
- 3' b010:Rounds off to negative infinity. The corresponding instruction is `fmadd.h fd,fs1, fs2, fs3, rdn`.
- 3' b011:Rounds off to positive infinity. The corresponding instruction is `fmadd.h fd,fs1, fs2, fs3, rup`.
- 3' b100:Rounds off to the nearest large value. The corresponding instruction is `fmadd.h fd,fs1, fs2, fs3, rmm`.
- 3' b101:This code is reserved and not used.
- 3' b110:This code is reserved and not used.
- 3' b111:Dynamically rounds off based on the `rm` bit of the floating-point register `FCSR`. The corresponding instruction is `fmadd.h fd,fs1, fs2, fs3`.

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0					
fs3				10	fs2				fs1				rm	fd	1000011			

6.21 FMAX.H: a half-precision floating-point maximum instruction.

Syntax:
`fmax.h fd, fs1, fs2`
Operation:

```
if(fs1 >= fs2)
```

```
    fd ← fs1
```

```
else
```

```
    fd ← fs2
```

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Affected flag bit:

Floating-point status bit NV

Instruction format:

31	25	24	20	19	15	14	12	11	7	6	0
0010110		fs2		fs1		001		fd		1010011	

6.22 FMIN.H: a half-precision floating-point minimum instruction.

Syntax:

```
fmin.h fd, fs1, fs2
```

Operation:

```
if(fs1 >= fs2)
```

```
    fd ← fs2
```

```
else
```

```
    fd ← fs1
```

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Affected flag bit:

Floating-point status bit NV

Instruction format:

31	25	24	20	19	15	14	12	11	7	6	0
0010110	fs2			fs1		000	fd		1010011		

6.23 FMSUB.H: a half-precision floating-point multiply-subtract instruction.

Syntax:

fmsub.h fd, fs1, fs2, fs3, rm

Operation:

$fd \leftarrow fs1 * fs2 - fs3$

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Affected flag bits:

Floating-point status bits NV, OF, UF, and IX

Notes:

RM determines the round-off mode:

- 3' b000:Rounds off to the nearest even number. The corresponding instruction is fmsub.h fd,fs1, fs2, fs3, rne.
- 3' b001:Rounds off to zero. The corresponding instruction is fmsub.h fd,fs1, fs2, fs3, rtz.
- 3' b010:Rounds off to negative infinity. The corresponding instruction is fmsub.h fd,fs1, fs2, fs3, rdn.
- 3' b011:Rounds off to positive infinity. The corresponding instruction is fmsub.h fd,fs1, fs2, fs3, rup.
- 3' b100:Rounds off to the nearest large value. The corresponding instruction is fmsub.h fd, fs1, fs2, fs3, rmm.
- 3' b101:This code is reserved and not used.
- 3' b110:This code is reserved and not used.
- 3' b111:Dynamically rounds off based on the rm bit of the floating-point register FCSR. The corresponding instruction is fmsub.h fd,fs1, fs2, fs3.

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
fs3				10	fs2				fs1		rm	fd	1000111

6.24 FMUL.H: a half-precision floating-point multiply instruction.

Syntax:

fmul.h fd, fs1, fs2, rm

Operation:

$fd \leftarrow fs1 * fs2$

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Affected flag bits:

Floating-point status bits NV, OF, UF, and NX

Notes:

RM determines the round-off mode:

- 3' b000:Rounds off to the nearest even number. The corresponding instruction is `fmul.h fd, fs1, fs2, rne`.
- 3' b001:Rounds off to zero. The corresponding instruction is `fmul.h fd, fs1, fs2, rtz`.
- 3' b010:Rounds off to negative infinity. The corresponding instruction is `fmul.h fd, fs1, fs2, rdn`.
- 3' b011:Rounds off to positive infinity. The corresponding instruction is `fmul.h fd, fs1, fs2, rup`.
- 3' b100:Rounds off to the nearest large value. The corresponding instruction is `fmul.h fd, fs1, fs2, rmm`.
- 3' b101:This code is reserved and not used.
- 3' b110:This code is reserved and not used.
- 3' b111:Dynamically rounds off based on the `rm` bit of the floating-point register `FCSR`. The corresponding instruction is `fmul.h fs1, fs2`.

Instruction format:

31	25	24	20	19	15	14	12	11	7	6	0
0001010		fs2		fs1		rm		fd		1010011	

6.25 FMV.H.X: a half-precision floating-point write transmit instruction.

Syntax:

`fmv.h.x fd, rs1`

Operation:

$fd[15:0] \leftarrow rs1[15:0]$

$fd[63:16] \leftarrow 48' \text{ hfffffffffff}$

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Affected flag bits:

None

Instruction format:

31	25	24	20	19	15	14	12	11	7	6	0
1111010		00000		rs1		000		fd		1010011	

6.26 FMV.X.H: a transmission instruction that reads half-precision floating-point registers.

Syntax:

`fmv.x.h rd, fs1`

Operation:

$tmp[15:0] \leftarrow fs1[15:0]$

$rd \leftarrow \text{sign_extend}(tmp[15:0])$

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Affected flag bits:

None

Instruction format:

31	25	24	20	19	15	14	12	11	7	6	0
1110010	00000	fs1	000	rd	1010011						

6.27 FNMADD.H: a half-precision floating-point negative multiply-add instruction.

Syntax:

fnmadd.h fd, fs1, fs2, fs3, rm

Operation:

$fd \leftarrow -(fs1 * fs2 + fs3)$

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Affected flag bits:

Floating-point status bits NV, OF, UF, and IX

Notes:

RM determines the round-off mode:

- 3' b000:Rounds off to the nearest even number. The corresponding instruction is fnmadd.h fd,fs1, fs2, fs3, rne.
- 3' b001:Rounds off to zero. The corresponding instruction is fnmadd.h fd,fs1, fs2, fs3, rtz.
- 3' b010:Rounds off to negative infinity. The corresponding instruction is fnmadd.h fd,fs1, fs2, fs3, rdn.
- 3' b011:Rounds off to positive infinity. The corresponding instruction is fnmadd.h fd,fs1, fs2, fs3, rup.
- 3' b100:Rounds off to the nearest large value. The corresponding instruction is fnmadd.h fd,fs1, fs2, fs3, rmm.
- 3' b101:This code is reserved and not used.

- 3' b110: This code is reserved and not used.
- 3' b111: Dynamically rounds off based on the rm bit of the floating-point register FCSR. The corresponding instruction is `fnmadd.h fd,fs1, fs2, fs3`.

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0		
fs3				10	fs2				fs1				rm	fd	1001111

6.28 FNMSUB.H: a half-precision floating-point negative multiply-subtract instruction.

Syntax:

`fnmsub.h fd, fs1, fs2, fs3, rm`

Operation:

$fd \leftarrow -(fs1 * fs2 - fs3)$

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Affected flag bits:

Floating-point status bits NV, OF, UF, and IX

Notes:

RM determines the round-off mode:

- 3' b000: Rounds off to the nearest even number. The corresponding instruction is `fnmsub.h fd,fs1, fs2, fs3, rne`.
- 3' b001: Rounds off to zero. The corresponding instruction is `fnmsub.h fd,fs1, fs2, fs3, rtz`.
- 3' b010: Rounds off to negative infinity. The corresponding instruction is `fnmsub.h fd,fs1, fs2, fs3, rdn`.
- 3' b011: Rounds off to positive infinity. The corresponding instruction is `fnmsub.h fd,fs1, fs2, fs3, rup`.
- 3' b100: Rounds off to the nearest large value. The corresponding instruction is `fnmsub.h fd,fs1, fs2, fs3, rmm`.
- 3' b101: This code is reserved and not used.

- 3' b110: This code is reserved and not used.
- 3' b111: Dynamically rounds off based on the rm bit of the floating-point register FCSR.
The corresponding instruction is `fmsub.h fd,fs1, fs2, fs3`.

Instruction format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0					
fs3				10	fs2				fs1				rm	fd	1001011			

6.29 FSGNJ.H: a half-precision floating-point sign-injection instruction.

Syntax:

`fsgnj.h fd, fs1, fs2`

Operation:

$fd[14:0] \leftarrow fs1[14:0]$

$fd[15] \leftarrow fs2[15]$

$fd[63:16] \leftarrow 48' \text{ hfffffffffff}$

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Affected flag bits:

None

Instruction format:

31	25	24	20	19	15	14	12	11	7	6	0	
0010010			fs2		fs1		000		fd		1010011	

6.30 FSGNJN.H: a half-precision floating-point sign-injection negate instruction.

Syntax:

`fsgnjn.h fd, fs1, fs2`

Operation:

$$fd[14:0] \leftarrow fs1[14:0]$$

$$fd[15] \leftarrow ! fs2[15]$$

$$fd[63:16] \leftarrow 48' \text{ hfffffffff}$$
Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Affected flag bits:

None

Instruction format:

31	25	24	20	19	15	14	12	11	7	6	0
0010010		fs2		fs1		001		fd		1010011	

6.31 FSGNJX.H: a half-precision floating-point sign-injection XOR instruction.

Syntax:

fsgnjx.h fd, fs1, fs2

Operation:

$$fd[14:0] \leftarrow fs1[14:0]$$

$$fd[15] \leftarrow fs1[15] \wedge fs2[15]$$

$$fd[63:16] \leftarrow 48' \text{ hfffffffff}$$
Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Affected flag bits:

None

Instruction format:

31	25	24	20	19	15	14	12	11	7	6	0
0010010		fs2		fs1		010		fd		1010011	

6.32 FSH: an instruction that stores half-precision floating point numbers.

Syntax:

fs_h fs2, imm12(fs1)

Operation:

address ← fs1 + sign_extend(imm12)

mem[(address+1):address] ← fs2[15:0]

Permission:

M mode/S mode/U mode

Exception:

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

Instruction format:

31	25	24	20	19	15	14	12	11	7	6	0
imm12[11:5]				fs2		fs1		001	imm12[4:0]		0100111

6.33 FSQRT.H: an instruction that calculates the square root of the half-precision floating-point number.

Syntax:

fsqrt.h fd, fs1, rm

Operation:

fd ← sqrt(fs1)

Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Affected flag bits:

Floating-point status bits NV and NX

Notes:

RM determines the round-off mode:

- 3' b000:Rounds off to the nearest even number. The corresponding instruction is fsqrt.h fd, fs1,rne.
- 3' b001:Rounds off to zero. The corresponding instruction is fsqrt.h fd, fs1,rtz.
- 3' b010:Rounds off to negative infinity. The corresponding instruction is fsqrt.h fd, fs1,rdn.
- 3' b011:Rounds off to positive infinity. The corresponding instruction is fsqrt.h fd, fs1,rup.
- 3' b100:Rounds off to the nearest large value. The corresponding instruction is fsqrt.h fd, fs1,rmm.
- 3' b101:This code is reserved and not used.
- 3' b110:This code is reserved and not used.
- 3' b111:Dynamically rounds off based on the rm bit of the floating-point register FCSR. The corresponding instruction is fsqrt.h fd, fs1.

Instruction format:

31	25	24	20	19	15	14	12	11	7	6	0
0101110	00000	fs1	rm	fd	1010011						

6.34 FSUB.H: a half-precision floating-point subtract instruction.**Syntax:**

```
fsub.h fd, fs1, fs2, rm
```

Operation:

$$fd \leftarrow fs1 - fs2$$
Permission:

M mode/S mode/U mode

Exception:

Invalid instruction.

Affected flag bits:

Floating-point status bits NV, OF, and NX

Notes:

RM determines the round-off mode:

- 3' b000:Rounds off to the nearest even number. The corresponding instruction is fsub.h fd, fs1,fs2,rne.
- 3' b001:Rounds off to zero. The corresponding instruction is fsub.h fd, fs1,fs2,rtz.

- 3' b010:Rounds off to negative infinity. The corresponding instruction is fsub.h fd, fs1,fs2,rdn.
- 3' b011:Rounds off to positive infinity. The corresponding instruction is fsub.h fd, fs1,fs2,rup.
- 3' b100:Rounds off to the nearest large value. The corresponding instruction is fsub.h fd, fs1,fs2,rmm.
- 3' b101:This code is reserved and not used.
- 3' b110:This code is reserved and not used.
- 3' b111:Dynamically rounds off based on the rm bit of the floating-point register FCSR. The corresponding instruction is fsub.h fd, fs1,fs2.

Instruction format:

31	25 24	20 19	15 14	12 11	7 6	0
0000110	fs2	fs1	rm	fd	1010011	