

Gentech C910 User's Manual (openc910)

October 19, 2021

Copyright 2021 T-Head Semiconductor Co., Ltd.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either See the License for the specific language governing permissions and limitations under the License.

Version History

releases	descriptive	dates
01	The first version of openc910 is released.	2021.10.19

catalogs

Chapt er 1	1
1.1 Introduction	1
1.2 Features	1
1.2.1 Key Features of the C910MP Processor Architecture	1
1.2.2 Key Features of the C910 Core	2
1.3 Configuration	2
1.4 Extension Techniques for the Xenon Architecture	3
1.5 Imprint	3
1.6 Naming rules	3
1.6.1 Symbols	3
1.6.2 Terminology	3
Chapt er II	6
2.1 Block Diagrams	6
2.2 Subsystems in the nucleus	7
2.2.1 Instruction Extraction Unit	7
2.2.2 Instruction Decoding Unit	7
2.2.3 Execution Units	7
2.2.4 Storage Load Cell	7

2.2.5	Instruction Retirement Unit	8
2.2.6	Virtual Memory Management Unit	8
2.2.7	Physical Memory Protection Unit	8
2.3	Multicore subsystems	8
2.3.1	Data Conformance Interface Unit	8
2.3.2	L2 Cache	8
2.3.3	Master Device Interface	9
2.3.4	Platform Level Interrupt Controller	9
2.3.5	Timer	9
2.4	Interface Overview	9
Chapt	instruction set	10
er III		
3.1	RV Basic Instruction Set	10
3.1.1	Integer Instruction Set (RV64I)	10

3.1.2	Multiplication and Division Instruction Set (RV64M)	13
3.1.3	Atomic Instruction Set (RV64A)	14
3.1.4	Single-precision floating-point instruction set (RV64F)	14
3.1.5	Double precision floating point instruction set (RV64D)	16
3.1.6	Compressed Instruction Set (RV64C)	18
3.2	Xenon Extended Instruction Set.....	20
3.2.1	Arithmetic Class Instructions	20
3.2.2	Bit Operation Class Instructions	21
3.2.3	Memory Access Class Instructions.....	21
3.2.4	Cache Command.....	24
3.2.5	Multicore Synchronization Instructions.....	25
3.2.6	Half-Precision Floating-Point Class Instructions	25
Chapter 4 Processor Modes and Registers		27
4.1	Processor Mode.....	27
4.2	Register View	27
4.3	General Purpose Registers	29
4.4	Floating Point Registers.....	29
4.4.1	Floating Point Registers and General Purpose Registers Transferring Data	30
4.4.2	Maintaining Consistent Register Accuracy	30
4.5	System Control Registers.....	30
4.5.1	Standard Control Registers	30
4.5.2	Extended Control Registers	32
4.6	Data format.....	34
4.6.1	Integer Data Format.....	34
4.6.2	Floating Point Data Format	34
4.7	Big and small end	36
Chapter 5 Exceptions and Interrupts		37
5.1	Overview.....	37
5.2	Abnormalities.....	39
5.2.1	Exception Response.....	39
5.2.2	Exception return	39
5.2.3	Imprecise anomalies	40
5.3	Interruptions	40
5.3.1	Interrupt Priority.....	40
5.3.2	Interrupt Response	40
5.3.3	Interrupt Return.....	41
Chapter 6: Memory Modeling		42
6.1	Overview of the Memory Model	42
6.1.1	Memory Properties	42
6.1.2	Memory Consistency Model	43

6.2 Virtual Memory Management	44
6.2.1 MMU Overview	44
6.2.2 TLB Organizational Forms	44
6.2.3 Address Translation Flow	44
6.2.4 System Control Registers	45
6.2.4.1 MMU Address Translation Register (SATP)	46
6.2.4.2 MMU Control Register (SMCIR)	46
6.2.4.3 MMU Index Register (SMIR)	48
6.2.4.4 MMU EntryHi Register (SMEH)	49
6.2.4.5 MMU EntryLo Register (SMEL)	49
6.3 Physical memory protection	51
6.3.1 PMP Overview	51
6.3.2 PMP Control Registers	52
6.3.2.1 Physical Memory Protection Setting Register (PMPCFG)	52
6.3.2.2 Physical Memory Protection Address Register (PMPADDR)	54
6.4 Memory access order	54
Chapt er VII	56
7.1 Memory Subsystem Overview	56
7.2 L1 Command Cache	56
7.2.1 Overview	56
7.2.2 Road Prediction	56
7.2.3 Cyclic Acceleration Cache	57
7.2.4 Branch History Table	57
7.2.5 Branch Jump Goal Predictor	57

7.2.6	Indirect Branch Predictor	57
7.2.7	Return Address Predictor	58
7.2.8	Quick Jump Goal Predictor	58
7.3	L1 Data Cache	59
7.3.1	Overview	59
7.3.2	Cache Consistency	59
7.3.3	Exclusive Access	59
7.4	L2 Cache	60
7.4.1	L2 Cache Overview	60
7.4.2	Cache Consistency	60
7.4.3	Organizational Forms	61
7.4.4	RAM Delay	61
7.5	Memory Accelerated Access	63
7.5.1	L1 I-Cache Instruction Prefetching	63
7.5.2	L1 D-Cache Multi-Channel Data Prefetching	63
7.5.3	L1 Adaptive Write Allocation Mechanism	64
7.5.4	L2 Prefetching Mechanism	64

7.6	Instructions and Registers Related to L1/L2 Cache Operations	64
7.6.1	L1 Cache Expansion Register.....	65
7.6.2	L2 Cache Expansion Register.....	65
7.6.3	L1/L2 Cache Operation Instructions.....	65
Chapter 8 Interrupt Controller		67
8.1	CLINT Interrupt Controller.....	67
8.1.1	CLINT Register Address Mapping	67
8.1.2	Software Interrupt.....	68
8.1.3	Timer Interrupt.....	69
8.2	PLIC Interrupt Controller.....	70
8.2.1	Interrupted arbitration.....	71
8.2.2	Interrupt Requests and Responses	71
8.2.3	Completion of interruptions.....	71
8.2.4	PLIC Register Address Mapping.....	72
8.2.5	Interrupt Priority Configuration Register (PLIC_PRIO)	74
8.2.6	Interrupt Wait Register (PLIC_IP)	74
8.2.7	Interrupt Enable Register (PLIC_IE)	75
8.2.8	PLIC Privilege Control Register (PLIC_PER)	75
8.2.9	Interrupt threshold register (PLIC_TH).....	76
8.2.10	Interrupt response/completion register (PLIC_CLAIM)	76
8.3	Multicore Interrupt.....	76
8.3.1	Multiple Cores Handle External Interrupts Simultaneously	77
8.3.2	Inter-core send software interrupt	77
Chapter 9 Bus Interfaces		78
9.1	AXI Master Device Interface	78
9.1.1	Characteristics of the AXI Master Device Interface	78
9.1.2	Outstanding capability of the master device interface.....	78
9.1.3	Supported Transmission Types.....	79
9.1.4	Supported Response Types	79
9.1.5	Behavior with different bus responses.....	80
9.1.6	AXI Master Device Interface Signal.....	80
Chapter 10 Commissioning		84
10.1	Debug Unit Functions.....	84
10.2	Debug Unit Connection to CPU Core	84
10.3	Debug Interface Signals	86
Chapter 11 Power Consumption Management		88
11.1	Power Domain.....	88
11.2	Low Power Mode Summary	88
11.3	Core WFI Processes	88

11.4	Individual Core Power Down Flow	89
11.5	Cluster Power Down Process (Hardware Clear L2)	90
11.6	Cluster Power Down Procedure (Software Clear L2)	91
11.7	Simplified Scenario: Cluster Overall Power Down Process (Hardware Clear L2)	92
11.8	Simplified Scenario: Cluster Overall Power Down Process (Software Clear L2)	92
11.9	Programming Models and Interface Signals Related to Low Power Consumption	94
11.9.1	Programming Model	94
11.9.2	Interface signals	94
Chapter XII	Performance Monitoring Unit	95
12.1	Introduction to PMU	95
12.2	Programming Model for PMUs	95
12.2.1	Basic PMU Usage	95
12.2.2	PMU Event Overflow Interrupt	96
12.3	PMU-related control registers	96
12.3.1	Machine Mode Counter Access Authorization Register (mcounteren)	96
12.3.2	Super User Mode Counter Access Authorization Register (scounteren)	97
12.3.3	Machine Mode Count Inhibit Register (mcountinhibit)	97
12.3.4	Superuser Write Enable Register (mcounterwen)	98
12.3.5	Performance Monitoring Event Selection Register	98
12.3.6	Event counters	100
Chapter XIII	Example of a program	101
13.1	Processor Optimized Performance Configuration	101
13.2	MMU Setup Example	102

13.3	PMP Setup Example	105
13.4	Cache Example	106
13.4.1	Cache Enable Example	106
13.4.2	Instruction Cache and Data Cache Synchronization Example	107
13.4.3	Example of TLB Synchronization with Data Cache	108
13.5	Examples of Synchronized Proxies	108
13.6	PLIC Setup Example	109
13.7	PMU Setup Example	110
Chapter XIV	Appendix A Standardized Directive Terminology	111
14.1	Appendix A-1 I Command Terminology	111
14.1.1	ADD - Signed Addition Instruction	111
14.1.2	ADDI - Signed Immediate Number Addition Instruction	111
14.1.3	ADDIW - Low 32-Bit Signed Immediate Number Addition Instruction	112
14.1.4	ADDW - Low 32-bit Signed Addition Instruction	112
14.1.5	AND - By Bit with Instructions	113
14.1.6	ANDI - Immediate Number By Bit and Instruction	113
14.1.7	AUIPC - PC Higher Immediate Number Addition Instruction	114

14.1.8 BEQ - Equal Branching Instruction	114
14.1.9 BGE - Signed Greater Than Equal Branching Instruction.....	115
14.1.10 BGEU - unsigned greater than or equal to branch instruction.....	115
14.1.11 BLT - Signed Less Than Branching Instruction	116
14.1.12 BLTU - Unsigned Less Than Branching Instruction.....	117
14.1.13 BNE - Unequal Branching Instruction.....	117
14.1.14 CSRRC - Control Register Clear Transfer Instruction	118
14.1.15 CSRRCI - Control Register Immediate Count Clear Transfer Instruction.....	119
14.1.16 CSRRS - Control Register Placement Transfer Instruction	119
14.1.17 CSRRSI - Control Register Immediate Number Reset Transfer Instruction.....	120
14.1.18 CSRRW - Control Register Read/Write Transfer Instruction	120
14.1.19 CSRRWI - Control Register Immediate Read/Write Transfer Instruction.....	121
14.1.20 EBREAK - Breakpoint Instruction	122
14.1.21 ECALL - Environmental Exception Directive.....	122
14.1.22 FENCE - Store Synchronization Instruction	122
14.1.23 FENCE.I - Command Flow Synchronization Instruction	123
14.1.24 JAL - Direct Jump Subprogram Instruction	123
14.1.25 JALR - Register Jump Rotor Program Instruction	124
14.1.26 LB - Signed Extended Byte Load Instruction	125
14.1.27 LBU - Unsigned Extended Byte Load Instruction.....	125
14.1.28 LD - Double Word Load Instruction.....	125
14.1.29 LH - Signed Extended Half-Word Load Instruction.....	126
14.1.30 LHU - Unsigned Extended Halfword Load Instruction.....	126
14.1.31 LUI - High Immediate Number Load Instruction	127
14.1.32 LW - Signed Extended Word Load Instruction.....	127
14.1.33 LWU - unsigned extended word load instruction.....	128
14.1.34 MRET - Machine Mode Exception Return Instruction.....	128
14.1.35 OR - By Bit or Command	129
14.1.36 ORI - Immediate Number by Bit or Instruction.....	129
14.1.37 SB - Byte Storage Instruction.....	130
14.1.38 SD - Double Word Storage Instruction	130
14.1.39 SFENCE.VMA - Virtual Memory Synchronization Instruction.....	130
14.1.40 SH - Half-Word Storage Instruction	131
14.1.41 SLL - Logical Left Shift Instruction.....	132
14.1.42 SLLI - Immediate Logical Left Shift Instruction.....	132
14.1.43 SLLIW - Low 32-bit Immediate Logical Left Shift Instruction.....	133
14.1.44 SLLW - low 32-bit logic left shift instruction.....	133
14.1.45 SLT - Signed Compare Less Than Placement Instruction	134
14.1.46 SLTI - Signed Immediate Number Compare Less Than Placement Instruction....	134
14.1.47 SLTIU - Unsigned Immediate Number Compare Less Than Placement Instruction	135
14.1.48 SLTU - unsigned compare less than place instruction	135
14.1.49 SRA - arithmetic right shift instruction.....	136

14.1.50 SRAI - Immediate Number Arithmetic Right Shift Instruction	136
14.1.51 SRAIW - Low 32-bit Immediate Count Arithmetic Right Shift Instruction	137
14.1.52 SRAW - Low 32-Bit Arithmetic Right Shift Instruction	137
14.1.53 SRET - Super User Mode Exception Return Instruction	137
14.1.54 SRL - Logical Right Shift Instruction	138
14.1.55 SRLI - Immediate Logical Right Shift Instruction	138
14.1.56 SRLIW - Low 32-bit Immediate Logical Right Shift Instruction	139
14.1.57 SRLW - Low 32-bit Logical Right Shift Instruction	139
14.1.58 SUB - Signed Subtraction Instruction	140
14.1.59 SUBW - Low 32-bit Signed Subtract Instruction	140
14.1.60 SW - Word Storage Instructions	141
14.1.61 WFI - Enter Low Power Mode Command	141
14.1.62 XOR - Per-position Alter-or Instruction	141
14.1.63 XORI - Immediate Number by Bit Isothermal Instruction	142
14.2 Appendix A-2 M Command Terminology	142
14.2.1 DIV - Signed Division Instruction	142
14.2.2 DIVU - Unsigned Division Instruction	143
14.2.3 DIVUW - Low 32-bit Unsigned Division Instruction	143
14.2.4 DIVW - Low 32-bit Signed Division Instruction	144
14.2.5 MUL - Signed Multiplication Instructions	145
14.2.6 MULH - Signed Multiplication Take Higher Instruction	145
14.2.7 MULHSU - Signed Unsigned Multiplication Take Higher Instruction	145
14.2.8 MULHU - Unsigned Multiply Take Higher Instruction	146

14.2.9 MULW - Low 32-bit Signed Multiply Instruction	146
14.2.10 REM - Signed Remainder Instruction	147
14.2.11 REMU - unsigned remainder instruction	147
14.2.12 REMUW - Low 32-bit Unsigned Remainder Instruction	148
14.2.13 REMW - Low 32-bit Signed Remainder Instruction	148
14.3 Appendix A-3 A Command Terminology	149
14.3.1 AMOADD.D - Atomic Addition Instruction	149
14.3.2 AMOADD.W - Low 32-bit Atomic Addition Instruction	150
14.3.3 AMOAND.D - Atomic By Bits and Instructions	151
14.3.4 AMOAND.W - Low 32-bit Atomic By-bit with Instructions	151
14.3.5 AMOMAX.D - Atomic Signed Maximization Instruction	152
14.3.6 AMOMAX.W - Low 32-bit Atomic Signed Maximum Instruction	153
14.3.7 AMOMAXU.D - Atomic Unsigned Maximization Instruction	154
14.3.8 AMOMAXU.W - Low 32-bit Atomic Unsigned Maximization Instruction	155
14.3.9 AMOMIN.D - Atomic Signed Minimization Instruction	155
14.3.10 AMOMIN.W - Low 32-bit Atomic Signed Minimization Instruction	156
14.3.11 AMOMINU.D - Atomic Unsigned Take Minimum Instruction	157
14.3.12 AMOMINU.W - Low 32-bit Atomic Unsigned Minimize Instruction	158
14.3.13 AMOOR.D - Atomic By Bit or Instruction	159

14.3.14	AMOOR.W - Low 32-bit Atomic By Bit Or Instruction	159
14.3.15	AMOSWAP.D - Atomic Exchange Directive	160
14.3.16	AMOSWAP.W - Low 32-bit Atomic Swap Instruction.....	161
14.3.17	AMOXOR.D - Atomic By Bit Alteration Instruction.....	162
14.3.18	AMOXOR.W - Low 32-bit Atomic Per-bit Alter-or Instruction.....	163
14.3.19	LR.D - Double Word Load Reserved Instructions.....	163
14.3.20	LR.W - Word Load Reserved Instructions.....	164
14.3.21	SC.D - Double Word Conditional Storage Instruction.....	165
14.3.22	SC.W - Word Conditional Storage Instruction.....	166
14.4	Appendix A-4 F Instruction Terminology	167
14.4.1	FADD.S - Single-Precision Floating Point Addition Instruction	167
14.4.2	FCLASS.S - Single-Precision Floating Point Classification Instruction.....	168
14.4.3	FCVT.L.S - Single Precision Floating Point to Signed Long Integer Instructions...	169
14.4.4	FCVT.L.U.S - Single Precision Floating Point to Unsigned Long Integer Instructions	170
14.4.5	FCVT.S.L - Signed Long Integer to Single Precision Floating Point Instruction....	171
14.4.6	FCVT.S.LU - Unsigned Long Integer to Single Precision Floating Point Instruction	172
14.4.7	FCVT.S.W - Signed Integer to Single Precision Floating Point Instruction	172
14.4.8	FCVT.S.WU - Unsigned Integer to Single Precision Floating Point Instruction.....	173
14.4.9	FCVT.W.S - Single Precision Floating Point to Signed Integer Instructions.....	174
14.4.10	FCVT.W.U.S - Single Precision Floating Point to Unsigned Integer Instructions....	175
14.4.11	FDIV.S - Single-Precision Floating-Point Division Instruction	176
14.4.12	FEQ.S - Single-Precision Floating-Point Compare Equal Instruction.....	177
14.4.13	FLE.S - Single Precision Floating Point Compare Less Than or Equal Instruction	177
14.4.14	FLT.S - Single Precision Floating Point Compare Less Than Instruction	178
14.4.15	FLW - Single Precision Floating Point Load Instruction	178
14.4.16	FMADD.S - Single-Precision Floating-Point Multiply-Accumulate Instruction.....	179
14.4.17	FMAX.S - Single Precision Floating Point Maximize Instruction	180
14.4.18	FMIN.S - Single Precision Floating Point Minimize Instruction	180
14.4.19	FMSUB.S - Single Precision Floating Point Multiply Accumulate Decrease Instruction	181
14.4.20	FMUL.S - Single-Precision Floating-Point Multiplication Instruction.....	182
14.4.21	FMV.W.X - Single-Precision Floating-Point Write Transfer Instructions.....	183
14.4.22	FMV.X.W - Single-Precision Floating-Point Register Read Transfer Instruction...	183
14.4.23	FNMADD.S - Single-Precision Floating-Point Multiply Accumulate Take Negative Instruction	184
14.4.24	FNMSUB.S - Single-Precision Floating-Point Multiply Accumulate Decrease Take Negative Instruction	185
14.4.25	FSGNJ.S - Single-Precision Floating-Point Symbol Injection Instruction.....	185
14.4.26	FSGNJS.N - Single-Precision Floating-Point Symbol Take and Invert Injection Instruction	186
14.4.27	FSGNJS.S - Single-Precision Floating-Point Symbol Alias Injection Instruction...	187
14.4.28	FSQRT.S - Single Precision Floating Point Open Square Instruction.....	187
14.4.29	FSUB.S - Single-Precision Floating-Point Subtraction Instruction.....	188

14.4.30 FSW - Single-Precision Floating Point Store Instruction.....	189
14.5 Appendix A-5 D Instruction Terminology	189
14.5.1 FADD.D - Double Precision Floating Point Addition Instruction.....	189

14.5.2	FCLASS.D - Double Precision Floating Point Classification Instruction.....	190
14.5.3	FCVT.D.L - Signed Long Integer to Double Precision Floating Point Instruction..	191
14.5.4	FCVT.D.LU - Unsigned Long Integer to Double Precision Floating Point Instruction	192
14.5.5	FCVT.D.S - Single-Precision Floating-Point to Double-Precision Floating-Point Conversion Instruction	193
14.5.6	FCVT.D.W - Signed Integer to Double Precision Floating Point Instruction	194
14.5.7	FCVT.D.WU - Unsigned Integer to Double Precision Floating Point Instruction ...	194
14.5.8	FCVT.L.D - Double Precision Floating Point to Signed Long Integer Instruction..	195
14.5.9	FCVT.LUD - Double Precision Floating Point to Unsigned Long Integer Instructions	195
14.5.10	FCVT.S.D - Double Precision Floating Point to Single Precision Floating Point Instructions	196
14.5.11	FCVT.W.D - Double Precision Floating Point to Signed Integer Instruction	197
14.5.12	FCVT.WUD - Double Precision Floating Point to Unsigned Integer Instruction...	198
14.5.13	FDIV.D - Double Precision Floating Point Division Instruction.....	199
14.5.14	FEQ.D - Double precision floating point compare equal instruction	200
14.5.15	FLD - Double Precision Floating Point Load Instruction.....	200
14.5.16	FLE.D - Double Precision Floating Point Compare Less Than or Equal Instruction	201
14.5.17	FLT.D - Double precision floating point compare less than instruction	201
14.5.18	FMADD.D - Double Precision Floating Point Multiply Accumulate Instruction...	202
14.5.19	FMAX.D - Double Precision Floating Point Maximize Instruction.....	203
14.5.20	FMIN.D - Double Precision Floating Point Minimize Instruction.....	203
14.5.21	FMSUB.D - Double Precision Floating Point Multiply Accumulate Decrease Instruction	204
14.5.22	FMUL.D - Double Precision Floating Point Multiply Instruction.....	205
14.5.23	FMV.D.X - Double Precision Floating Point Write Transfer Instruction.....	206
14.5.24	FMV.X.D - Double Precision Floating Point Read Transfer Instruction	206
14.5.25	FNMADD.D - Double Precision Floating Point Multiply Accumulate Take Negative Instruction	207
14.5.26	FNMSUB.D - Double Precision Floating Point Multiply Accumulate Decrease Take Negative Instruction	208
14.5.27	FSD - Double Precision Floating Point Store Instruction	208
14.5.28	FSGNJ.D - Double Precision Floating Point Symbol Injection Instruction	209
14.5.29	FSGNJD - Double Precision Floating Point Symbol Take and Invert Injection Instruction	209
14.5.30	FSGNJD - Double-Precision Floating-Point Symbol Iso-Or Injection Instruction	210
14.5.31	FSQRT.D - Double Precision Floating Point Open Square Instruction.....	211
14.5.32	FSUB.D - Double Precision Floating Point Subtraction Instruction.....	211
14.6	Appendix A-6 C Command Terminology	212
14.6.1	C.ADD - Signed addition instruction.....	212
14.6.2	C.ADDI - Signed Immediate Number Addition Instruction.....	213
14.6.3	C.ADDIW - Low 32-bit Signed Immediate Number Addition Instruction.....	213
14.6.4	C.ADDI4SPN - 4x Immediate Number and Stack Pointer Summing Instruction ...	214
14.6.5	C.ADDI16SP - Add 16x Immediate Number to Stack Pointer Instruction	215

14.6.6	C.ADDW - Low 32-bit signed addition instruction.....	215
14.6.7	C.AND - By Bit with Instruction.....	216
14.6.8	C.ANDI - Immediate Number by Bit with Instruction.....	217
14.6.9	C.BEQZ - Branch Equal to Zero Instruction.....	218
14.6.10	C.BNEZ - Branch Not Equal to Zero Instruction.....	219

14.6.11 C.EBREAK - Breakpoint Instruction	220
14.6.12 C.FLD - Floating Point Double Word Load Instruction	220
14.6.13 C.FLDSP - Floating Point Double Word Stack Load Instruction	221
14.6.14 C.FSD - Floating Point Double Word Storage Instruction	222
14.6.15 C.FSDSP - Floating Point Double Word Stack Store Instruction	223
14.6.16 C.J - Unconditional Jump Instructions	223
14.6.17 C.JALR - Register Jump Rotor Program Instruction	224
14.6.18 C.JR - Register Jump Instructions	225
14.6.19 C.LD - Double Word Load Instruction	225
14.6.20 C.LDSP - Double Word Stack Load Instruction	226
14.6.21 C.LI - Immediate Number Transfer Instruction	227
14.6.22 C.LUI - High Immediate Number Transfer Instruction	227
14.6.23 C.LW - Word Load Instruction	228
14.6.24 C.LWSP - Word Stack Load Instruction	229
14.6.25 C.MV - Data Transfer Instructions	229
14.6.26 C.NOP - Null Instruction	230
14.6.27 C.OR - by bit or instruction	230
14.6.28 C.SD - Double Word Storage Instruction	231
14.6.29 C.SDSP - Double Word Stack Store Instructions	232
14.6.30 C.SLLI - Immediate Logical Left Shift Instruction	232
14.6.31 C.SRAI - Immediate Number Arithmetic Right Shift Instruction	233
14.6.32 C.SRLI - Immediate Logical Right Shift Instruction	234

14.6.33 C.SW - Word Storage Instructions	235	
14.6.34 C.SWSP - Word Stack Store Instructions	235	
14.6.35 C.SUB - Signed Subtraction Instructions	236	
14.6.36 C.SUBW - Low 32-bit Signed Subtraction Instruction	237	
14.6.37 C.XOR - Bitwise Alter-or Instruction	238	
14.7 Appendix A-8 Pseudo Instruction List	238	
Chapter XV	Appendix B Flathead Extended Directive Terminology	241
15.1 Appendix B-1 Cache Instruction Terminology	241	
15.1.1 DCACHE.CALL - DCACHE Clear All Dirty Table Entries command	241	
15.1.2 DCACHE.CIALL - DCACHE Invalid command after clearing all dirty table entries	242	
15.1.3 DCACHE.CIPA -- DCACHE clears dirty table entries by physical address and invalidates them	242	
15.1.4 DCACHE.CISW - DCACHE clears dirty table entries and invalidates commands by way/set	243	
15.1.5 DCACHE.CIVA - DCACHE clears dirty table entries by virtual address and invalidates them	243	
15.1.6 DCACHE.CPA - DCACHE Clear dirty table entries by physical address	244	
15.1.7 DCACHE.CPAL1 --L1DCACHE Clear dirty table entries by physical address	244	
15.1.8 DCACHE.CVA - DCACHE Clear Dirty Table Entries by Virtual Address	245	
15.1.9 DCACHE.CVAL1 - L1DCACHE Clear dirty table entries by virtual address	245	
15.1.10 DCACHE.IPA -- DCACHE Invalid by Physical Address Directive	246	
15.1.11 DCACHE.ISW -- DCACHE by set/way Invalid Directive	247	
15.1.12 DCACHE.IVA -- DCACHE Invalid by Virtual Address Directive	247	
15.1.13 DCACHE.IALL - DCACHE Invalid All Table Entries command	248	
15.1.14 ICACHE.IALL - ICACHE Invalidate All Table Entries command	248	

15.1.15 ICACHE.IALLS - ICACHE Broadcast Invalid All Table Entries command	249
15.1.16 ICACHE.IPA - ICACHE Invalid Table Entry by Physical Address command	249
15.1.17 ICACHE.IVA - ICACHE Invalid Table Entry by Virtual Address command	250
15.1.18 L2CACHE.CALL - L2CACHE Clear All Dirty Table Entries command	250
15.1.19 L2CACHE.CIALL--L2CACHE Clear all dirty table entries and invalidate commands	251
15.1.20 L2CACHE.IALL - L2CACHE Invalid Instruction	251
15.1.21 DCACHE.CSW -- DCACHE cleans dirty table entries by set/way	252
15.2 Appendix B-2 Multicore Synchronization Instruction Terminology	252
15.2.1 SFENCE.VMAS - Virtual Memory Synchronization Broadcast Instruction	253
15.2.2 SYNC - Synchronization Command	254
15.2.3 SYNC.I - Synchronized Clearance Command	254
15.2.4 SYNC.IS - Synchronized Clear Broadcast Command	254
15.2.5 SYNC.S - Synchronized Broadcast Command	255
15.3 Appendix B-3 Arithmetic Instruction Terminology	255
15.3.1 ADDSL - Register Shift Sum Instruction	255
15.3.2 MULA - Multiply Accumulate Instructions	256
15.3.3 MULAH - Low 16-bit Multiply Accumulate Instruction	256
15.3.4 MULAW - Low 32-Bit Multiply-Accumulate Instruction	257
15.3.5 MULS - Multiply and Accumulate Instructions	257
15.3.6 MULSH - Low 16-bit Multiply Accumulate Decrease Instruction	258
15.3.7 MULSW - Low 32-bit Multiply Accumulate Decrease Instruction	258
15.3.8 MVEQZ - Register 0 Transmit Instruction	259
15.3.9 MVNEZ - Register Non-Zero Transfer Instructions	259

15.3.10 SRRI - Cyclic Right Shift Instruction	260
15.3.11 SRRIW - Low 32-bit Loop Right Shift Instruction	260
15.4 Appendix B-4 Bit Operation Instruction Terminology	260
15.4.1 EXT - Register Contiguous Bit Extract Symbol Bit Extension Instruction	261
15.4.2 EXTU - Register Contiguous Bit Extract Zero Extension Instruction	261
15.4.3 FF0 - Fast Find 0 Instruction	262
15.4.4 FF1 - Quick Find 1 Command	262
15.4.5 REV - Byte Reverse Instruction	262
15.4.6 REVW - Lower 32-Bit Byte Reverse Instruction	263
15.4.7 TST - Bit to 0 Test Instruction	264
15.4.8 TSTNBZ - Byte 0 Test Instruction	264
15.5 Appendix B-5 Memory Instruction Terminology	265
15.5.1 FLRD - Floating Point Register Shift Double Word Load Instruction	265
15.5.2 FLRW - Floating Point Register Shift Word Load Instruction	265
15.5.3 FLURD - Floating Point Register Lower 32 Bit Shifted Double Word Load Instruction	266
15.5.4 FLURW - Floating Point Register Lower 32-Bit Shift Word Load Instruction	267

15.5.5	FSRD - Floating Point Register Shift Double Word Store Instruction.....	267
15.5.6	FSRW - Floating Point Register Shift Word Store Instruction.....	268
15.5.7	FSURD - Floating Point Register Lower 32 Bit Shift Double Word Store Instruction	268
15.5.8	FSURW - Floating Point Register Lower 32 Bit Shift Word Store Instruction.....	269
15.5.9	LBIA - Signed Bit Extended Byte Load Base Address Self-Increment Instruction.	269
15.5.10	LBIB - Base Address Self-Incrementing Symbolic Bit Extended Byte Load Instruction	270
15.5.11	LBUIA - Zero Extended Byte Load Base Address Self-Incrementing Instruction ..	270
15.5.12	LBUIB - Base Address Self-Incrementing Zero Extended Byte Load Instruction ..	271
15.5.13	LDI - Dual Register Load Instruction.....	271
15.5.14	LDIA - Signed Bit Extended Double Word Load Base Address Self-Increment Instruction	
	272
15.5.15	LDIB - Base Address Self-Incrementing Symbolic Bit Extended Double Word Load Instruction.....	272
15.5.16	LHIA - Signed Bit Extended Half-Word Load Base Address Self-Increment Instruction	273
15.5.17	LHIB - Base Address Self-Incrementing Symbolic Bit Extended Half-Word Load Instruction	
	273
15.5.18	LHUIA - Zero Extended Half-Word Load Base Address Self-Increment Instruction	274
15.5.19	LHUIB - Base Address Self-Incrementing Zero Extended Half-Word Load Instruction	
	274
15.5.20	LRB - Register Shift Symbol Bit Extended Byte Load Instruction.....	275
15.5.21	LRBU - Register Shift Zero Extension Extension Byte Load Instruction.....	275
15.5.22	LRD - Register Shift Double Word Load Instruction.....	276
15.5.23	LRH - Register Shift Symbol Bit Extended Half-Word Load Instruction.....	276
15.5.24	LRHU - Register Shift Zero Extension Extension Half Word Load Instruction.....	277
15.5.25	LRW - Register Shift Symbol Bit Extended Word Load Instruction.....	277
15.5.26	LRWU - Register Shift Zero Extension Extended Word Load Instruction.....	277
15.5.27	LURB - Register Lower 32 Bit Shifted Signed Bit Extended Byte Load Instruction	278
15.5.28	LURBU - Register Lower 32 Bit Shift Zero Extended Byte Load Instruction.....	278
15.5.29	LURD - Register Low 32 Bit Shift Double Word Load Instruction.....	279
15.5.30	LURH - Register Lower 32 Bit Shifted Signed Bit Extended Half-Word Load Instruction	
	279
15.5.31	LURHU - Register Lower 32 Bit Shifted Zero Extended Half Word Load Instruction	280
15.5.32	LURW - Register Low 32 Bit Shifted Signed Bit Extended Word Load Instruction	280
15.5.33	LURWU - Register Low 32 Bit Shift Zero Extended Word Load Instruction	281
15.5.34	LWD - Signed Bit Extended Dual Register Word Load Instruction	281
15.5.35	LWIA - Signed Bit Extended Word Load Base Address Self-Increment Instruction	282
15.5.36	LWIB - Base Address Self-Incrementing Symbolic Bit Extended Word Load Instruction	
	282
15.5.37	LWUD - Zero Extended Dual Register Word Load Instruction	283
15.5.38	LWUIA - Zero Extended Word Load Base Address Self-Increment Instruction....	284
15.5.39	LWUIB - Base Address Self-Incrementing Zero Extended Word Load Instruction	284
15.5.40	SBIA - Byte Storage Base Address Self-Incrementing Instruction.....	285
15.5.41	SBIB - Base Address Self-Incrementing Byte Store Instruction	285
15.5.42	SDD - Dual Register Storage Directive.....	286
15.5.43	SDIA - Double Word Storage Base Address Self-Increment Instruction	286

15.5.44 SDIB - Base Address Self-Incrementing Double Word Storage Instruction	287
15.5.45 SHIA - Half-Word Storage Base Address Self-Incrementing Instruction	287
15.5.46 SHIB - Base Address Self-Incrementing Half-Word Storage Instruction	287

15.5.47 SRB - Register Shift Byte Store Instruction	288
15.5.48 SRD - Register Shift Double Word Store Instruction.....	288
15.5.49 SRH - Register Shift Half-Word Storage Instruction	289
15.5.50 SRW - Register Shift Word Store Instruction.....	289
15.5.51 SURB - Register Lower 32 Bit Shift Byte Store Instruction	290
15.5.52 SURD - Register Lower 32 Bit Shift Double Word Store Instruction.....	290
15.5.53 SURH--Register Lower 32 Bit Shift Half-Word Store Instruction.....	291
15.5.54 SURW - Register Lower 32 Bit Shift Word Store Instruction.....	291
15.5.55 SWIA - Word Storage Base Address Self-Increment Instruction	292
15.5.56 SWIB - Base Address Self-Incrementing Word Store Instruction.....	292
15.5.57 SWD - Dual Register Low 32-Bit Store Instruction	293
15.6 Appendix B-6 Floating-Point Half-Precision Instruction Terminology.....	293
15.6.1 FADD.H - Half-Precision Floating Point Addition Instruction.....	293
15.6.2 FCLASS.H - Half-Precision Floating Point Classification Instruction	294
15.6.3 FCVT.D.H - Half-Precision Floating Point to Double-Precision Floating Point Instruction	295
15.6.4 FCVT.H.D - Double Precision Floating Point to Half Precision Floating Point Instructions	296
15.6.5 FCVT.H.L - Signed Long Integer to Half-Precision Floating Point Instruction	297
15.6.6 FCVT.H.LU - Unsigned Long Integer to Half-Precision Floating Point Instruction	297
15.6.7 FCVT.H.S - Single Precision Floating Point to Half Precision Floating Point Instructions	298
15.6.8 FCVT.H.W - Signed Integer to Half-Precision Floating Point Instruction.....	299
15.6.9 FCVT.H.WU - Unsigned Integer to Half-Precision Floating Point Instruction	300
15.6.10 FCVT.L.H - Half-Precision Floating Point to Signed Long Integer Instruction.....	301
15.6.11 FCVT.LU.H - Half-Precision Floating Point to Unsigned Long Integer Instructions	302
15.6.12 FCVT.S.H - Half-Precision Floating Point to Single-Precision Floating Point Instruction	303
15.6.13 FCVT.W.H - Half-Precision Floating Point to Signed Integer Instruction	303
15.6.14 FCVT.WU.H - Half-Precision Floating Point to Unsigned Integer Instructions	304
15.6.15 FDIV.H - Half-Precision Floating-Point Division Instruction.....	305
15.6.16 FEQ.H - Half-precision floating-point compare equal instruction.....	306
15.6.17 FLE.H - half-precision floating-point compare less than or equal to instruction..	306
15.6.18 FLH - Half-Precision Floating Point Load Instruction	307
15.6.19 FLT.H - half-precision floating-point compare less than instruction.....	307
15.6.20 FMADD.H - Half-Precision Floating-Point Multiply-Accumulate Instruction.....	308
15.6.21 FMAX.H - Half-Precision Floating Point Maximize Instruction.....	309
15.6.22 FMIN.H - Half-Precision Floating Point Minimize Instruction	309
15.6.23 FMSUB.H - Half-Precision Floating Point Multiply Accumulate Decrease Instruction	310
15.6.24 FMUL.H - Half-Precision Floating-Point Multiplication Instruction	311
15.6.25 FMV.H.X - Half-Precision Floating Point Write Transfer Instruction	312
15.6.26 FMV.X.H - Half-Precision Floating Point Register Read Transfer Instruction.....	312
15.6.27 FNFMADD.H - Half-Precision Floating Point Multiply Accumulate Take Negative Instruction	313
15.6.28 FNMSUB.H - Half-Precision Floating Point Multiply Accumulate Decrease Take Negative	

Instruction	314
15.6.29 FSGNJ.H - Half-Precision Floating Point Symbol Injection Instruction.....	315
15.6.30 FSGNHN.H - Half-Precision Floating Point Symbol Take and Invert Injection Instruction	315

15.6.31 FSGNJP.H - Half-Precision Floating-Point Symbol Alias Injection Instruction	316
15.6.32 FSH - Half-Precision Floating-Point Storage Instruction	316
15.6.33 FSQRT.H - Half-Precision Floating Point Open Square Instruction	317
15.6.34 FSUB.H - Half-Precision Floating Point Subtraction Instruction	318
Chapter XVI Appendix C Control Registers	319
16.1 Appendix C-1 Machine Mode Control Registers	319
16.1.1 Machine Mode Information Register Set	319
16.1.1.1 Machine Mode Vendor Number Register (MVENDORID)	319
16.1.1.2 Machine Mode Architecture Number Register (MARCHID)	319
16.1.1.3 Machine Mode Hardware Implementation Numbering Register (MIMPID)	319
16.1.1.4 Machine Mode Logic Kernel Number Register (MHARTID)	320
16.1.2 Machine Mode Exception Configuration Register Set	320
16.1.2.1 Machine Mode Processor Status Register (MSTATUS)	320
16.1.2.2 Machine Mode Processor Instruction Set Feature Register (MISA)	322
16.1.2.3 Machine Mode Exception Degradation Control Register (MEDELEG)	323
16.1.2.4 Machine Mode Interrupt Degradation Control Register (MIDELEG)	323
16.1.2.5 Machine Mode Interrupt Enable Control Register (MIE)	323
16.1.2.6 Machine Mode Vector Base Address Register (MTVEC)	325
16.1.2.7 Machine Mode Counter Access Authorization Register (MCOUNTEREN)	325
16.1.3 Machine Mode Exception Handling Register Set	325
16.1.3.1 Machine Mode Exception Temporary Data Backup Register (MSCRATCH)	325
16.1.3.2 Machine Mode Exception Reserved Program Counter Register (MEPC)	326
16.1.3.3 Machine Mode Abnormal Event Vector Register (MCAUSE)	326

16.1.3.4 Machine Mode Interrupt Waiting Status Register (MIP)	326
16.1.4 Machine Mode Memory Protection Register Set	328
16.1.4.1 Machine Mode Physical Memory Protection Configuration Register (PMPCFG)	328
16.1.4.2 Machine Mode Physical Memory Address Register (PMPADDR)	328
16.1.5 Machine Mode Counter Register Set	328
16.1.5.1 Machine Mode Cycle Counter (MCYCLE)	328
16.1.5.2 Machine Mode Retirement Instruction Counter (MINSTRET)	328
16.1.5.3 Machine Mode Event Counter (MHPMCOUNTERn)	329
16.1.6 Machine Mode Counter Configuration Register Set	329
16.1.6.1 Machine Mode Event Selector (MHPMEVENTn)	329
16.1.7 Machine Mode Processor Control and Status Extension Register Set	329
16.1.7.1 Machine Mode Extended Status Register (MXSTATUS)	329
16.1.7.2 Machine Mode Hardware Configuration Register (MHCR)	331
16.1.7.3 Machine Mode Hardware Operation Register (MCOR)	333
16.1.7.4 Machine Mode L2Cache Control Register (MCCR2)	334
16.1.7.5 Machine Mode Implicit Operation Register (MHINT)	335
16.1.7.6 Machine Mode Reset Vector Base Address Register (MRVBR)	338
16.1.7.7 Super Household Counter Write Enable Register (MCOUNTERWEN)	338

16.1.7.8	Machine mode event interrupt enable register (MCOUNTERINTEN).....	339
16.1.7.9	Overflow labeling register on machine mode events (MCOUNTEROF)	339
16.1.8	Machine Mode Cache Access Extended Register Set	340
16.1.8.1	Machine Mode Cache Instruction Registers (MCINS)	340
16.1.8.2	Machine Mode Cache Access Index Register (MCINDEX)	340
16.1.8.3	Machine Mode Cache Data Register (MCDATA0/1)	341
16.1.9	Machine Mode Processor Model Register Group.....	342
16.1.9.1	Machine Mode Processor Model Register (MCPUID)	342
16.1.9.2	On-chip bus base address register (MAPBADDR).....	342
16.1.10	Multicore Extended Register Set	342
16.1.10.1	Snoop Listener Enable Register (MSMPR)	342
16.2	Appendix C-2 Super User Mode Control Registers.....	343
16.2.1	Superuser Mode Exception Configuration Register Set	343
16.2.1.1	Super User Mode Processor Status Register (SSTATUS).....	343
16.2.1.2	Super User Mode Interrupt Enable Control Register (SIE).....	343
16.2.1.3	Super User Mode Vector Base Address Register (STVEC).....	344
16.2.1.4	Super User Mode Counter Access Authorization Register (SCOUNTEREN)	344
16.2.2	Super User Mode Exception Handling Register Set	344
16.2.2.1	Super User Mode Exception Temporary Data Backup Register (SSCRATCH)	344
16.2.2.2	Super User Mode Exceptionally Reserved Program Counter Register (SEPC)	345
16.2.2.3	Super User Mode Abnormal Event Vector Register (SCAUSE)	345
16.2.2.4	Super User Mode Interrupt Waiting Status Register (SIP)	345
16.2.3	Super User Mode Address Translation Register Set	345
16.2.3.1	Super User Mode Address Translation Register (SATP)	345
16.2.4	Super User Mode Processor Control and Status Extension Register Set.....	346
16.2.4.1	Super User Mode Extended Status Register (SXSTATUS)	346
16.2.4.2	Super User Mode Hardware Control Register (SHCR)	346
16.2.4.3	Super User Mode Event Overflow Interrupt Enable Register (SCOUNTERINTEN)	346
16.2.4.4	Super User Mode Event On Overflow Labeling Register (SCOUNTEROF) ..	346
16.2.4.5	Super User Mode Cycle Counter (SCYCLE)	346
16.2.4.6	Super User Mode Retirement Instruction Counter (SINSTRET)	347
16.2.4.7	Super User Mode Event Counter (SHPMCOUNTERn).....	347
16.2.5	Super User Mode MMU Expansion Registers.....	347
16.2.5.1	Super User Mode MMU Control Register (SMCIR).....	347
16.2.5.2	Super User Mode MMU Control Register (SMIR)	347
16.2.5.3	Super User Mode MMU Control Register (SMEH)	347
16.2.5.4	Super User Mode MMU Control Register (SMEL).....	347
16.3	Appendix C-3 User Mode Control Registers	348
16.3.1	User Mode Floating Point Control Register Set	348
16.3.1.1	Floating Point Exception Accumulation Status Register (FFLAGS).....	348
16.3.1.2	Floating Point Dynamic Rounding Mode Register (FRM)	348
16.3.1.3	Floating Point Control Status Register (FCSR)	348

16.3.2	User Mode Counting/Timing Register Set	349
16.3.2.1	User Mode Cycle Counter (CYCLE).....	349
16.3.2.2	User Mode Time Counter (TIME)	349
16.3.2.3	User Mode Retirement Instruction Counter (INSTRET)	350
16.3.2.4	User Mode Event Counter (HPMCOUNTERn)	350
16.3.3	User Mode Extended Floating Point Control Register Set	350
16.3.3.1	User Mode Floating Point Extended Control Register (FXCR).....	350

Chapter I. Overview

This document is the user manual for the open source version of C910 (openc910). The expressions "C910", "C910MP", "Core", "Core", etc. appearing in this document refer to the open source version of C910. C910", "C910MP", "Core", "Core", etc. are used to refer to the open source version of C910.

1.1 summary

C910MP is a 64-bit high-performance multi-core processor based on RISC-V instruction architecture, which is mainly targeted at edge computing fields with strict performance requirements, such as edge servers, edge computing cards, high-end machine vision, high-end video surveillance, automated driving, mobile intelligent terminals, and 5G base stations, etc. C910MP adopts isomorphic multi-core architecture and supports dual cores. C910MP adopts isomorphic multi-core architecture and supports dual-core. Each C910 core adopts independently designed micro-architecture and focuses on performance optimization, introducing high performance technologies such as superscalar architecture with 3 decodes and 8 executes, and multi-channel data prefetching. In addition, the C910 cores support real-time detection and shutdown of internal idle function modules to reduce the dynamic power consumption of the processor.

1.2 specificities

1.2.1 Key Features of the C910MP Processor Architecture

- Isomorphic multi-core architecture with dual-core support;
- Supports independent power down for each core and cluster power down;
- Supports 1 AXI4.0 Master interface with 128 bit bus width;
- A two-tiered high-relief structure with a Harvard structural primary high-relief and a shared secondary high-relief;
- The first-level instruction/data cache is 64KB respectively, and the cache line SIZE is 64B;
- The first level cache supports the coherency protocol of MESI and the second level cache supports the coherency protocol of MOESI;

- Level 2 cache size is 1MB, cache line SIZE is 64B, and 16-way groups are connected;
- Private interrupt controller CLINT and public interrupt controller PLIC are supported;
- Supports timer function;
- Supports customized, RISC-V-compatible multicore debugging frameworks;

Chapter I.

Overview

1.2.2 Key features of the C910 core

- RISC-V 64GC instruction architecture;
- Support for small end mode;
- 9~12 level deep flow water architecture;
- 3-decode 8-execute superscalar architecture that is completely transparent to the software;
- Sequential Finger Taking, Messy Launch, Messy Finish and Sequential Retirement;
- Two-stage TLB memory management unit for real-virtual address translation and memory management;
- The instruction cache and data cache size is 64KB and the cache line is 64B;
- Command prefetch function, hardware auto-detection and dynamic startup;
- Low-power access techniques for instruction-high slow-path prediction;
- Low-power execution techniques for short cyclic caches;
- A 64Kb two-stage multi-way parallel branch predictor;
- 1024 Branch target buffer for table entries;
- Supports 12 levels of hardware return address stacks;
- 256 Indirect jump branch predictor for table entries;
- Non-blocking firing, speculative guessing execution;
- Physical register-based renaming techniques;
- The 0 delay move command is supported;
- Dual-launch, fully-disordered execution of load and store instructions;
- Supports 8 concurrent bus accesses for both read and write;
- Support write merge;
- Supports 8-channel data cache hardware prefetching and stride prefetching;
- Contains floating-point units and supports half-precision, single-precision, and double-precision;

1.3 configure

The C910MP utilizes the following configuration:

- Number of cores: 2
- L1 I-Cache: 64KB
- L1 D-Cache: 64KB

Chapter I. Overview

- L2 Cache: 1MB
- Master Interface: AXI4
- Number of external interrupts: 144

1.4 Extended technology for basalt architecture

The C910 is compatible with the Gentech C-Series 1.0 Extended Architecture, specifically:

- Arithmetic instruction extensions: Increased arithmetic capabilities in integer, floating point, load/store, etc., a powerful addition to the RISC-V base instruction set.
- Cache Operation Extension: Facilitates Cache maintenance operations and improves Cache efficiency.
- Memory Model Extension: Efficient Management of Address Properties for Improved Memory Access Efficiency
- Control Register Extension: Provides richer functionality than standard RISC-V
- Multicore Synchronization Instruction Extension: Improving the Efficiency of Multicore Consistency Maintenance
- PLIC Expansion: Embedded PLIC Interrupt Controller

1.5 Imprint

The C910 is compatible with the RISC-V standard, specifically the version:

- *The RISC-V Instruction Set Manual, Volume I: RISC-V User-Level ISA, Version 2.2.*
- *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10.*
- The Performance Monitoring Unit (PMU) adds the mcountinhibit register from *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 20190125-Public-Review-draft.*

1.6 Naming rules

1.6.1 nota tion

The standard symbols and operators used in this document are shown in Figure 1.1.

1.6.2 nomenclature

- Logic 1 is the level value that corresponds to the Boolean logic true.

Chapter I. Overview

- A **logic 0** is a level value that corresponds to a Boolean logic pseudo.

符号	功能
+	加
-	减
*	乘
/	除
>	大于
<	小于
=	等于
\geq	大于或等于
\leq	小于或等于
\neq	不等于
&	与
	或
\oplus	异或
NOT	取反
:	连接
\Rightarrow	传输
\Leftrightarrow	交换
\pm	误差
0b0011	二进制数
0x0F	十六进制数
rd	整型目的寄存器
rs1	整型源寄存器1
rs2	整型源寄存器2
rs3	整型源寄存器3
fd	浮点目的寄存器
fs1	浮点源寄存器1
fs2	浮点源寄存器2
fs3	浮点源寄存器3
vd	矢量目的寄存器
vs1	矢量源寄存器1
vs2	矢量源寄存器2
vs3	矢量源寄存器3

Figure 1.1: Symbol List

Chapter I. Overview

- **Setting** refers to bringing one or more bits to the level corresponding to logic 1.
- **Clear** means to bring one or more bits to the level corresponding to a logic 0.
- **Reserved bits** are reserved for function expansion and have a value of 0 when not specified.
- A **signal** is an electrical value that conveys information through its state or transitions between states.
- A **pin** is an indication of an external electrical physical connection, and multiple signals can be connected to the same pin.
- **Enable** means to put a discrete signal in an active state:
 - The active low signal switches from high to low;
 - The active high level signal switches from low to high.
- **Disable** means to make a signal in the enable state change its state:
 - The active low level signal switches from low to high;
 - The active signal switches from high level to low level.
- **LSB** represents the least significant bit and **MSB** represents the most significant bit.
- **Signals, bit fields, and control bits** are represented using a common rule.
- **The identifier is followed by a number indicating the range**, from high to low indicating a set of signals.

For example, "addr[4:0]" represents a set of address buses where the highest bit is addr[4] and the lowest bit is addr[0].

- **A single identifier** represents a single signal.

For example, "pad_cpu_rst_b" represents a separate signal.

Sometimes a number is added to an identifier to indicate a certain meaning, for example, addr15 means the 16th bus in a group.

Bit.

Chapter 2 Processor Introduction

2.1 Structural Block Diagram

The block diagram of the C910MP is shown in Figure 2.1.

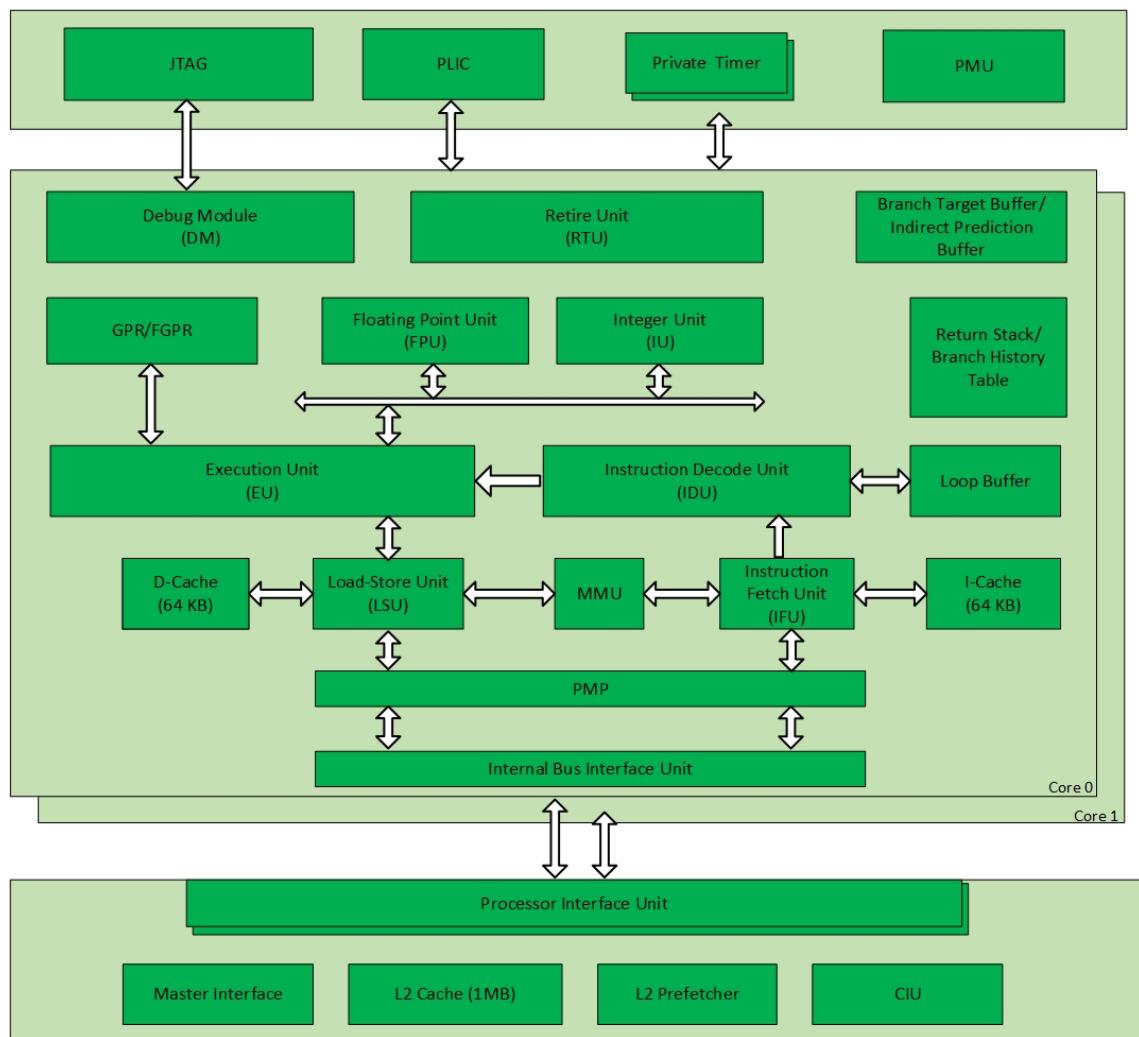


Figure 2.1: C910MP Microarchitecture Diagram

2.2 nuclear subsystem

The C910 in-core subsystems mainly contain: Instruction Fetch Unit (IFU), Instruction Decode Unit (IDU), Integer Execution Unit (IU), Floating Point Unit (FPU) Store Load Unit (LSU) Instruction Retirement Unit (RTU) Virtual Memory Management Unit (MMU) and Physical Memory Protection Unit (PMP)

2.2.1 command extraction unit

Instruction Fetching Unit (IFU) that can fetch up to eight instructions at a time and process them in parallel. Several techniques are implemented to improve access efficiency, such as instruction high-buffer prediction, instruction staging, cycle-accelerated cache, and direct/indirect branch prediction. The entire instruction extraction unit is characterized by low power consumption, high branch prediction accuracy, and high instruction prefetching efficiency.

2.2.2 command decoding unit

The instruction decoding unit (IDU) can decode three instructions simultaneously and detect data correlation. The data correlation between instructions is resolved by utilizing physical register renaming techniques and instructions are sent in disorganized order to the lower pipeline for execution. The instruction decoding unit supports the disordered scheduling and distribution of instructions and mitigates the performance loss due to data correlation through speculative firing.

2.2.3 implementation unit

The execution unit contains an integer unit (IU) and a floating point unit (FPU)

The integer unit consists of an arithmetic logic unit (ALU) a multiplication unit (MULT) a division unit (DIV), and a jump unit (BJU). The ALU performs 64-bit integer operations. The MULT supports 16*16, 32*32, and 64*64 integer multiplication. The divider design utilizes a base-16 SRT algorithm with execution cycles that vary depending on the number of operations. The BJU can complete branch prediction error handling in a single cycle.

The floating-point unit consists of the floating-point arithmetic logic unit (FALU) the floating-point fusion multiply accumulate unit (FMAU) and the floating-point divide and square unit (FDSU) which support half-precision, single-precision, and double-precision operations. The Floating Point Arithmetic Logic Unit (FALU) is responsible for operations such as addition, subtraction, comparison, conversion, register transfers, symbol injection, and classification. The floating-point fused multiply-accumulate unit (FMAU) is responsible for ordinary multiplication, fused multiply-accumulate, and other operations. Floating-point division and square unit (FDSU) is responsible for operations such as floating-point division, floating-point squaring, and so on.

2.2.4 storage load unit (SLO)

Chapter 2

Processor

The store-load unit (LSU) supports double firing of scalar **store/load** instructions, single firing of vector **store/load** instructions, and full promiscuous execution of all store/load instructions, and non-blocking access to the cache. Byte, half-word, word, double-word, and quad-word **store/load** instructions are supported, as well as sign bit and zero extensions for byte and half-word load instructions. **Store/load** instructions can be executed in streaming, enabling data throughput up to one data access per cycle. Supports 8-way data stream hardware prefetching technology, which puts data into the L1 data cache in advance. Supports parallel access to the bus when the data cache is missing.

2.2.5 Command Retirement Unit

The instruction retirement unit (RTU) consists of a reorder buffer and a physical register stack. The reorder buffer is responsible for the disordered recall and sequential retirement of instructions, and the physical register stack is responsible for the disordered recall and delivery of results. The instruction retirement efficiency is improved by supporting parallel recycling and fast retirement. The instruction retirement unit retires three instructions per clock cycle in parallel and supports precise exceptions.

2.2.6 VMMU

The Virtual Memory Management Unit (MMU) complies with the RISC-V SV39 standard for translating 39-bit virtual addresses into 40-bit physical addresses. The C910 MMU extends the hardware backfill standard defined in SV39 with software backfill methods and address attributes.

Refer to the [memory model](#) for specific information.

2.2.7 Physical Memory Protection Unit

C910 Physical Memory Protection Unit (PMP) is RISC-V compliant, supports 8 table entries with a minimum granularity of 4KB, does not support NA4 Mode.

Refer to the [memory model](#) for specific information.

2.3 multinuclear subsystem

The C910 multicore subsystem consists of a Data Coherency Interface Unit (CIU), a L2 cache, a Master Device Interface Unit (MDIU), a Platform Level Interrupt Controller (PLIC) timers, and a customized multicore single-port debugging framework.

2.3.1 Data Conformance Interface Unit

Data consistency interface unit (CIU), using the MESI protocol to maintain the consistency of each L1 data cache. Two listening buffers are set up to process multiple listening requests in parallel to maximize the use of listening bandwidth. The CIU adopts an efficient data bypass mechanism to bypass data directly to the request initiating core when the listening request hits the listened L1 data cache. In addition, the CIU supports the broadcasting of TLB and ICACHE invalid operation requests, which simplifies the software maintenance cost of TLB/ICache and DCache data consistency.

2.3.2 L2 cache

Chapter 2

Processor Introduction The L2 cache, tightly coupled with the CIU, realizes synchronous access with the L1 data cache; it adopts a chunked pipeline architecture, which can process two access requests in a single cycle in parallel, and the maximum access bandwidth can reach 1024 bits. The L2 cache operates at the same frequency as the C910, and the access latency of the TAG RAM and DATA RAM can be configured by software.

2.3.3 Master Device Interface

The Master Device Interface Unit supports the AXI4.0 protocol, keyword-first address access, and can operate at different system clock to CPU clock ratios (1:1, 1:2, 1:3, 1:4, 1:5, 1:6, 1:7, 1:8).

2.3.4 Platform-level interrupt controller

The Platform Level Interrupt Controller (PLIC) supports sampling and distribution of 144 external interrupt sources, supports level and pulse interrupts, and can be set up to 32 level of interrupt priority.

Refer to [the interrupt controller](#) for specific information

2.3.5 timekeeping device (sundial, water clock)

A 64-bit system timer is shared in the multi-core system, and each core has a private timer comparison value register, which generates a timer signal by collecting the system timer value and comparing it with the private timer comparison value register set by the software.

Refer to [the interrupt controller](#) for specific information

2.4 Interface Overview

The interfaces of C910 are divided into the following categories according to their functions: clock reset signals, bus system, interrupt system, debugging system, low power system, DFT system, and CPU operation observation signals, etc. The main interfaces of C910 are shown in Figure 2.2.

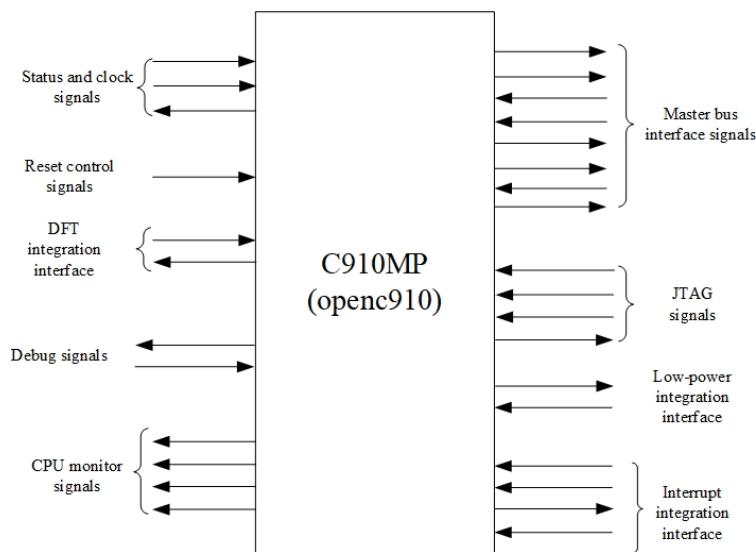


Figure 2.2: C910MP Interface Overview

Chapter 3 Instruction Set

This chapter focuses on the instruction set implemented in the C910, which is divided into two main parts: the RV base instruction set and the Xantei extended instruction set.

3.1 RV Basic Instruction Set

3.1.1 Integer instruction

set (RV64I)

The integer instruction set can be categorized by function as follows.

- addition and subtraction commands
- logical operation instruction
- shift command (computing)
- compare instruction
- data transmission instruction
- branch jump instruction
- memory access instruction
- Control register operation instructions:
- Low-power commands
- Exception return instruction
- Special Function Instructions

Table 3.1: List of Integer Instructions (RV64I) Instructions

command name	command description	execution delay
addition and subtraction commands		
ADD	signed addition instruction	1
ADDW	Low 32-bit signed addition instructions	1
ADDI	Signed Immediate Number Addition Instructions	1

Continued on next page

Chapter 3
Instruction
Set

Table 3.1 - continued from previous page

ADDIW	Low 32-bit Signed Immediate Addition Instructions	1
SUB	signed subtraction instruction	1
SUBW	Low 32-bit signed subtraction instructions	1
logical operation instruction		
AND	pushbutton and command	1
ANDI	Immediate Number by Bit and Instruction	1
OR	press-bit or command	1
ORI	Immediate number by bit or instruction	1
XOR	bitwisewise allosteric instruction	1
XORI	Immediate Bitwise Iso instruction	1
shift command (computing)		
SLL	logical left instruction	1
SLLW	Low 32-bit Logical Word Left Shift Instruction	1
SLLI	Immediate Logical Left Shift Instruction	1
SLLIW	Low 32-bit Immediate Logical Left Shift Instruction	1
SRL	Logical right shift instruction	1
SRLW	Low 32-bit Logical Right Shift Instruction	1
SRLI	Immediate Logical Right Shift Instruction	1
SRLIW	Low 32-bit Immediate Logical Right Shift Instruction	1
SRA	Arithmetic right shift instruction	1
SRAW	Low 32-bit arithmetic right shift instruction	1
SRAI	Immediate number arithmetic right shift instruction	1
SRAIW	Low 32-bit Immediate Arithmetic Right Shift Instructions	1 Continued on next page
compare instruction		
SLT	Signed Compare Less Than Set instruction	1
SLTU www.t-head.cn	Unsigned compare less than set instruction	1
SLTI	Signed Immediate Compare Less Than Set instruction	1

Chapter 3
Instruction
Set

Table 3.1 - continued from previous page

JAL	Direct jump program instructions	1
JALR	Register jump subroutine instructions	1

memory access instruction

LB	Signed Extended Byte Load Instruction	WEAK ORDER LOAD: >=3 STORE: 1 STRONG ORDER indeterminate period
LBU	Unsigned Extended Byte Load Instruction	ibid
LH	Signed Extended Half-Word Load Instruction	ibid
LHU	Unsigned Extended Half-Word Load Instruction	ibid
LW	Signed Extended Word Load Instruction	ibid
LWU	Unsigned Extended Word Load Instruction	ibid
LD	double-word load command (computing)	ibid
SB	byte memory instruction	ibid
SH	halfword storage instruction	ibid
SW	word storage instruction	ibid
SD	double-word store instruction	ibid

Control Register Operation Instructions

CSRRW	Control register read/write transfer instructions	阻塞执行(阻塞执行) indeterminate period
CSRRS	Control Register Setting Transfer Instruction	ibid
CSRRC	Control register clear transfer instruction	ibid
CSRRWI	Control Register Immediate Number Read/Write Transfer Instructions	ibid
CSRRSI	Control Register Immediate Number Placement Transfer Instruction	ibid
CSRRCI	Control Register Immediate Count Clear Transfer Instruction	ibid

Low-power commands

WFI	Enter low-power wait mode command	indeterminate period
-----	-----------------------------------	----------------------

Exception return instruction

Chapter 3

Instruction Set

Refer to [Appendix A-1 I Command Terminology](#) for specific command descriptions and definitions.

3.1.2 Multiplication and Division Instruction Set (RV64M)

Table 3.2: Integer Multiply/Divide (RV64M) Instruction List

command name	command description	execution delay
MUL	Signed multiply instruction	4
MULW	Low 32-bit signed multiply instructions	4
MULH	Signed multiplication fetch-high instruction	4
MULHS	Signed Unsigned Multiplication Take Higher Instruction	4
MULHU	Unsigned multiply fetch-high instruction	4
DIV	signed division instruction	3-20
DIVW	Low 32-bit signed division instruction	3-12
DIVU	unsigned division instruction	3-20
DIVUW	Low 32-bit unsigned divide instruction	3-12
REM	Signed remainder instruction	3-20
REMW	Low 32-bit signed remainder instruction	3-12
REMU	unsigned remainder instruction	3-20
REMUW	Low 32-bit unsigned remainder instruction	3-12

Refer to [Appendix A-2 M Command Terminology](#) for specific command descriptions and definitions.

Chapter 3 Instruction

3.1.3 Atomic instruction set (RV64A)

Table 3.3: List of Atomic Instructions (RV64A) Instructions

command name	command description	execution delay
LR.	word load reserved instruction	
LR.	Double word load reserved instruction	
SC.W	word-conditional memory instruction	
SC.	double-word conditional storage instruction	
AMOSWAP.	Low 32-bit atomic swap instructions	
AMOSWAP.	atomic exchange instruction	
AMOADD.W	Low 32-bit atomic addition instructions	
AMOADD.	atomic addition instruction	
AMOXOR.W	Low 32-bit Atomic Bitwise Alias Instruction	
AMOXOR.	atomic perpositional allosteric instruction	
AMOAND.W	Low 32-bit Atomic By Bit with Instructions	
AMOAND.	atomic press and command	
AMOOR.W	Low 32-bit atomic per-bit or instruction	
AMOOR.	atomic press or command (computing)	
AMOMIN.W	Low 32-bit Atomic Signed Minimum Instruction	
AMOMIN.D	Atomic Signed Minimization Instruction	
AMOMAX.W	Low 32-bit Atomic Signed Maximum Instruction	
AMOMAX.	Atomic Signed Maximum Instruction	
AMOMINU.W	Low 32-bit Atomic Unsigned Minimize Instruction	
AMOMINU.D	Atomic Unsigned Minimize Instruction	
AMOMAXU.W	Low 32-bit Atomic Unsigned Maximum Instruction	
AMOMAXU.D	Atomic Unsigned Maximize Instruction	

Chapter 3

Instruction

Set Refer to [Appendix A-3 A Command Terminology](#) for specific command descriptions and definitions.

3.1.4 Single-precision floating-point instruction set (RV64F)

The single-precision floating-point instruction set can be categorized into the following types by function:

- operating instruction
- Symbol Injection Instruction
- data transmission instruction
- compare instruction
- data type conversion instruction
- memory storage instruction

Chapter 3

Instruction

Set - Floating Point Number Classification Instruction

Table 3.4: List of Single-Precision Floating Point (RV64F) Instructions

command name	command description	execution delay
operating instruction		
FADD.	Single-precision floating-point addition instructions	3
FSUB.	Single-precision floating-point subtraction instruction	3
FMUL.	Single-precision floating-point multiply instruction	4
FMADD.	Single-precision floating-point multiply-accumulate instruction	5
FMSUB.	Single-precision floating-point multiply-accumulate instruction	5
FNMADD.	Single-precision floating-point multiply-accumulate-take-negative instruction	5
FNMSUB.	Single-precision floating-point multiply-accumulate-minus instruction	5
FDIV.	Single-precision floating-point divide instruction	4-10
FSQRT.	Single-precision floating-point open-square instructions	4-10
Symbol Injection Instruction		
FSGNJ.	Single-precision floating-point symbol injection instructions	3
FSGNJP.	Single-Precision Floating-Point Symbol Fetch and Invert Injection Instruction	3
FSGNJP.	Single-precision floating-point symbolic heterodyne injection instruction	3
data transmission instruction		
FMV.X.W	Single-precision floating-point read transfer instruction	Split implementation 1+1
FMV.W.X	Single-precision floating-point write transfer instructions	Split implementation 1+1
compare instruction		
FMIN.	Single-precision floating-point minima instruction	3

Chapter 3

Instruction Set		
FMAX.	Single-Precision Floating-Point Maximize Instruction	3
FEQ.	Single-precision floating-point compare-equal instruction	Split implementation 3+1
FLT.	Single precision floating point compare less than instruction	Split implementation 3+1
FLE.	Single-precision floating-point comparison is less than or equal to the instruction	Split implementation 3+1
data type conversion instruction		
FCVT.	Single-precision floating-point to signed integer fingers honorific title	Split implementation 3+1
FCVT.	Single-precision floating-point to unsigned integer pointer honorific title	Split implementation 3+1
FCVT.	Signed integer to single-precision floating-point finger. honorific title	Split implementation 3+1
FCVT.S.WU	Unsigned integer to single-precision floating-point finger. honorific title	Split implementation 3+1
FCVT.	Single precision floating point to signed long integer conversion directives	Split implementation 3+1

Continued on next page

Table 3.4 - continued from previous page

FCVT.	Single precision floating point to unsigned long integer conversion directives	Split implementation 3+1
FCVT.	Signed long integer to single precision floating point conversion directives	Split implementation 1+3
FCVT.	Unsigned long integer to single precision floating point conversion directives	Split implementation 1+3
memory storage instruction		
FLW	Single-precision floating-point load instruction	WEAK ORDER LOAD: >=3 STORE: 1 STRONG ORDER indeterminate period
FSW	Single-precision floating-point memory instruction	ibid
Floating Point Number Classification Instruction		
FCLASS.	Single-precision floating-point sorting instructions	1+1

For specific instruction descriptions and definitions, refer to [Appendix A-4 F Instruction Terminology](#).

3.1.5 Double precision floating point instruction set (RV64D)

The double precision floating point instruction set can be categorized into the following types by function:

- operating instruction
- Symbol Injection Instruction
- data transmission instruction
- compare instruction
- data type conversion instruction
- memory storage instruction

Table 3.5: List of Double Precision Floating Point (RV64D) Instructions

command name	command description	execution delay
operating instruction		

Chapter 3

Instruction Set	Description	Category
FADD.	Double-precision floating-point addition instruction	3
FSUB.	Double precision floating point subtract instruction	3
FMUL.	Double precision floating point multiply instruction	4
FMADD.	Double-precision floating-point multiply-accumulate instruction	5
FMSUB.	Double-precision floating-point multiply-accumulate instruction	5

Continued on next page

Chapter 3
Instruction
Set

Table 3.5 - continued from previous page

FNMSUB.	Double-precision floating-point multiply-accumulate-invert instruction	5
FNMADD.	Double-precision floating-point multiply-accumulate-minus-invert instruction	5
FDIV.	Double precision floating point divide instruction	4-17
FSQRT.	Double precision floating-point open-square instruction	4-17
Symbol Injection Instruction		
FSGNJ.D	Double precision floating point symbol injection instruction	3
FSGNJD.D	Double Precision Floating Point Symbol Fetch and Invert Injection Instruction	3
FSGNJD.X	Double Precision Floating Point Symbol Alias Injection Instruction	3
data transmission instruction		
FMV.	Double precision floating point read transfer instruction	1+1
FMV.D.X	Double precision floating point write transfer instruction	1+1
compare instruction		
FMIN.	Double Precision Floating Point Minimize Instruction	3
FMAX.	Double Precision Floating Point Maximize Instruction	3
FEQ.	Double precision floating point compare equal instruction	Split implementation 3+1
FLT.	Double precision floating point compare less than instruction	Split implementation 3+1
FLE.	Double precision floating point compare less than or equal to instruction	Split implementation 3+1
data type conversion instruction		
FCVT.	Double Precision Floating Point to Single Precision Floating Point Conversion Guide honorific title	3 Continued on next page
FCVT.	Single-precision floating-point to double-precision floating-point fingers honorific title	3
www.t-head.cn	20 honorific title	
FCVT.	Double Precision Floating Point to Signed Integer Finger	Split implementation 3+1

Table 3.5 - continued from previous page

FLD	Double precision floating point load instruction	WEAK ORDER LOAD: >=3 STORE: 1 STRONG ORDER indeterminate period
FSD	Double precision floating point memory instruction	ibid
Floating Point Number Classification Instruction		
FCLASS.	Double Precision Floating Point Categorical Instructions	1+1

Refer to [Appendix A-5 D Command Terminology](#) for specific command descriptions and definitions.

3.1.6 Compressed instruction set (RV64C)

The compression instruction set can be categorized into the following types by function:

- addition and subtraction commands
- logical operation instruction
- shift command (computing)
- data transmission instruction
- branch jump instruction
- Immediate Number Offset Access Instruction

Table 3.6: List of Compression Instructions (RV64C) Instructions

command name	command description	execution delay
addition and subtraction commands		
C. ADD	signed addition instruction	1
C. ADDW	Low 32-bit signed addition instructions	1
C. ADDI	Signed Immediate Number Addition Instructions	1
C. ADDIW	Low 32-bit Signed Immediate Addition Instructions	1
C.SUB	Signed Subtract Compression Instruction	1
C. SUBW	Low 32-bit signed subtraction instructions	1
C.ADDI16SP	Add 16 times the immediate number to the stack pointer.	1
C. ADDI4SPN	4x immediate and stack	1

Chapter 3

Instruction	pointer addition	
Set logical operation instruction		
C.AND	pushbutton and command	1
C. ANDI	Immediate Number by Bit and Instruction	1
C.OR	press-bit or command	1

Continued on next page

Chapter 3
Instruction
Set

Table 3.6 - continued from previous page

C. XOR	bitwise allosteric instruction	1
shift command (computing)		
C. SLLI	Immediate Logical Left Shift Instruction	1
C. SRLI	Immediate Logical Right Shift Instruction	1
C. SRAI	Immediate number arithmetic right shift instruction	1
data transmission instruction		
C. MV	data transmission instruction	1
C.LI	Low Immediate Transfer Instruction	1
C. LUI	High Immediate Number Transfer Instruction	1
branch jump instruction		
C. BEQZ	Equal to zero branching instruction	1
C. BNEZ	Not equal to zero branching instruction	1
C.J	unconditional jump instruction	1
C.JR	register jump instruction	1
C. JALR	Register jump subroutine instructions	1
Immediate Number Offset Access Instruction		
C.LW	word load command (computing)	WEAK ORDER LOAD: >=3 STORE: 1 STRONG ORDER indeterminate period
C.SW	word storage instruction	ibid
C. LWSP	word stack loading instruction	ibid
C. SWSP	word stack storage instruction	ibid
C. LD	double-word load command (computing)	ibid
C.SD	double-word store instruction	ibid
C. LDSP	Double-word stack load instruction	ibid
C. SDSP	double-word stack store instruction	ibid
C. FLD	Double precision loading instructions	ibid

Chapter 3

Instruction

C.FSD Set	Double precision store instruction	ibid
C.FLDSP	Double precision stack store instructions	ibid
C.FSDSP	Double precision stack load instruction	ibid
special instructions		
C.NOP	empty instruction	1
C.EBREAK	breakpoint instruction	1

For specific instruction descriptions and definitions, refer to [Appendix A-6 C Instruction Terminology](#).

3.2 Xenon Extended Instruction Set

In addition to supporting the RV64GC instruction set, the C910 also extends some of the customized instructions on this basis. half-precision floating-point instructions can be used directly in the C910 extended instruction set, but all other C910 extended instruction sets need to turn on the Extended Instruction Set Enable Bit (THEADISAE) in the Machine Mode Extended Status Register (MXS-TATUS) before they can be used normally. Otherwise, an illegal instruction exception will be thrown.

3.2.1 Arithmetic operations class of instructions

Table 3.7: Arithmetic Instruction Set

command name	command description	execution delay
addition and subtraction commands		
ADDSSL	register shift sum instruction	1
MULA	multiply-accumulate instruction	Non-accumulator correlation:4
MULS	multiply and decrease instruction	Non-accumulator correlation:4
MULAW	Low 32-bit multiply-accumulate instructions	Cumulative number correlation:1
MULSW	Low 32-bit multiply-accumulate instructions	Cumulative number correlation:1
MULAH	Low 16-bit multiply-accumulate instruction	Cumulative number correlation:1
MULSH	Lower 16 bits into an accumulating instruction	Cumulative number correlation:1
displacement instruction		
SRRI	Cyclic right shift instruction	1
SRRIW	Low 32-bit circular right shift instruction	1
Send command		
MVEQZ	Register is 0 Transmit Command	1

Chapter 3 Instruction Set

MVNEZ	Register non-zero Transmit instruction	1
-------	---	---

Refer to [Appendix B-3, Arithmetic Instruction Terminology](#),
for instruction descriptions and definitions.

3.2.2 Bit Operation Class Instruction

Table 3.8: Bit Manipulation Instruction Set

command name	command description	execution delay
bit operation instruction		
TST	Bit 0 test command	1
TSTNBZ	Byte 0 test command	1
REV	byte reversal instruction	1
REVVW	Low 32-Bit Byte Reverse Instruction	1
FF0	Quick Find 0 command	1
FF1	Quick Find 1 command.	1
EXT	Register Contiguous Bit Extract Symbol Bit Extension Instruction	1
EXTU	Register Contiguous Bit Extract Zero Expansion Instruction	1

Refer to [Appendix B-4 Bit Operation Instruction Terminology](#) for specific instruction descriptions and definitions.

3.2.3 Memory Access Class Instruction

Table 3.9: Memory Access Instruction Set

memory instruction	execution delay	
FLRD	Floating Point Register Shift Double Word Load Instruction	WEAK ORDER ≥ 3 STRONG ORDER indeterminate period
FLRW	Floating Point Register Shift Word Load Instruction	
FLURD	Floating-point register low 32-bit shift double-word-add load instruction	
FLURW	Floating Point Register Low 32-Bit Shift Word Load directives	

Chapter 3

Instruction

LRB Set	Register Shift Symbol Bit Extended Byte Load directives	
LRH	Register Shift Symbol Bit Extended Half-Word Load directives	
LRW	Register Shift Symbol Bit Extended Word Load Finger honorific title	
LRD	Register Shift Double Word Load Instruction	
LRBU	Register Shift Zero Extended Byte Load Instruction	

Continued on next page

Chapter 3
Instruction
Set

Table 3.9 - continued from previous page

LRHU	Register Shift Zero Extended Half-Word Load Instruction	
LRWU	Register Shift Zero Extended Word Load Instruction	
LURB	Register Low 32-bit Shifted Symbolic Bit Extension byte-loading instruction	
LURH	Register Low 32-bit Shifted Symbolic Bit Extension halfword load command (computing)	
LURW	Register Low 32-bit Shifted Symbolic Bit Extension word load command (computing)	
LURD	Register Low 32-bit Shift Double Word Load Finger honorific title	
LURBU	Register Low 32-bit Shift Zero Expansion Byte load command	
LURHU	Register Low 32-bit Shift Zero Extended Halfword load command	
LURWU	Register Low 32-bit Shift Zero Extended Word Plus load instruction	
LBIA	Signed bit extension byte load base address self-incrementing directives	Split the execution into load and alu instructions
LBIB	Base Address Self-Incrementing Symbolic Bit Extended Byte Load directives	
LHIA	Symbol Bit Extended Half-Word Load Base Address Self-Incrementing directives	
LHIB	Base Address Self-Incrementing Symbolic Bit Extended Half-Word Load directives	Continued on next page
LWIA	Symbol Bit Extension Word Load Base Address Self-	
www.t-head.cn	Incrementing Finger honorific title	
LWIB	Base Address Self-	

**Chapter 3
Instruction
Set**

Table 3.9 - continued from previous page

LWD	Symbolic Bit Extended Dual Register Word Load Instruction	
LWUD	Zero-Expansion Dual-Register Word Load Instruction	
FSRD	Floating Point Register Shift Double Word Store Instruction	WEAK ORDER 1 STRONG ORDER indeterminate period
FSRW	Floating Point Register Shift Word Storage Instruction	
FSURD	Floating Point Register Low 32-bit Shifted Double Word Store storage directive	
FSURW	Floating Point Register Low 32-bit Shift Word Storage directives	
SRB	Register Shift Byte Store Instruction	
SRW	Register Shift Word Storage Instructions	
SRD	Register Shift Double Word Store Instruction	
SURB	Register low 32-bit shift byte storage finger honorific title	
SURH	Register low 32-bit shifted half-word memory pointer honorific title	
SURW	Register Low 32-bit Shift Word Storage Instruction	
SURD	Register Low 32-bit Shifted Double Word Memory Finger honorific title	
SBIA	Byte memory base address self-increment instruction	Split into store and alu commands classifier for objects in rows such as words
SBIB	Base Address Self-Incrementing Byte Store Instruction	
SHIA	Half-Word Memory Base Address Self-Increment	

Chapter 3

Instruction Set	Instruction	
SHIB	Base Address Self-Incrementing Half-Word Store Instruction	
SWIA	Word memory base address self-increment instruction	
SWIB	Base Address Self-Incrementing Word Store Instruction	
SDIA	Double-word memory base address self-increment instruction	
SDIB	Base Address Self-Incrementing Double-Word Storage Instruction	
SDD	Dual register plus store instruction	Split into two store executions
SWD	Dual register low 32-bit memory instructions	

Refer to Appendix B-5 Memory Instruction Terminology for specific instruction descriptions and definitions.

Chapter 3

Instruction

3.2.4 Cache Command

Table 3.10: List of Cache Instructions

command name	command description	execute Execution Extension of time Time (LMUL=1)
DCACHE.CALL	DCACHE Clear All Dirty Table Entries command	Blocking
DCACHE.CIALL	DCACHE Invalid command after clearing all dirty table entries	Execution
DCACHE.CIPA	DCACHE clears dirty table entries by physical address and invalidates instructions (scopes contain L2CACHE)	Indefinite Cycle
DCACHE.CISW	DCACHE Press set/way to clear dirty table entries and invalidate commands.	
DCACHE.CIVA	DCACHE Clear dirty table entries by virtual address and invalidate commands (scope contains L2CACHE)	
DCACHE.CPA	DCACHE Clear Dirty Table Entries by Physical Address command (scope contains L2CACHE)	
DCACHE.CPAL1	L1DCACHE Clear dirty table entries by physical address command	
DCACHE.CSW	DCACHE Press the set/way clear dirty table entry command.	
DCACHE.	DCACHE clear dirty table entries by virtual address command (scope contains L2CACHE)	
DCACHE.CVAL1	L1DCACHE Clear dirty table entries by virtual address command	
DCACHE.IPA	DCACHE Invalidate command by physical address (scope contains L2CACHE)	
DCACHE.ISW	DCACHE Press set/way Invalid command	
DCACHE.IVA	DCACHE Invalidate command by virtual address (scope contains L2CACHE)	
DCACHE.IALL	DCACHE Invalidate All Table Entries command	
ICACHE.IALL	ICACHE Invalidate All Table Entries command	indeterminate period
ICACHE.IALLS	ICACHE Broadcast Invalidate All Table Entries command	
ICACHE.IPA	ICACHE Invalid Table Entry by Physical Address command	
ICACHE.IVA	ICACHE Invalid Table Entry by Virtual Address command	
L2CACHE.CALL	L2CACHE The clear all dirty table entries command.	

Chapter 3

Instruction

Set	L2CACHE.CIALL L2CACHE Clear all dirty table entries and invalidate commands.
L2CACHE.IALL	L2CACHE Invalid command

Refer to [Appendix B-1 Cache Instruction Terminology](#) for specific instruction descriptions and definitions.

3.2.5 multicore synchronization instruction

Table 3.11: Multicore Synchronization Instruction Set

multicore synchronization instruction	descriptive
SFENCE.VMAS	virtual memory synchronous broadcast instruction
SYNC	synchronization instruction
SYNC.	simulcast command
SYNC.	Synchronized Empty Command
SYNC.IS	Synchronized clear broadcast command

For specific instruction descriptions and definitions, refer to

[Appendix B-2 Multicore Synchronization Instruction Terminology](#).

3.2.6 Half-precision floating-point type instructions

Table 3.12: Half-Precision Floating-Point Instruction Set

command name	command description	execution delay
operating instruction		
FADD.H	Half-precision floating-point addition instruction	3
FSUB.	Half-precision floating-point subtraction instruction	3
FMUL.	Half-precision floating-point multiply instruction	3
FMADD.H	Half-precision floating-point multiply-accumulate instruction	4
FMSUB.	Half-precision floating-point multiply-accumulate instruction	4
FNMADD.H	Half-precision floating-point multiply-accumulate-take-negative instruction	4
FNMSUB.	Half-precision floating-point	4

Chapter 3

Instruction Set		
	multiply-accumulate-minus instruction	
FDIV.H	Half-precision floating-point divide instruction	4-7
FSQRT.	Half-precision floating-point open-square instruction	4-7
Symbol Injection Instruction		
FSGNJ.H	Half-precision floating-point symbol injection instructions	3
FSGNHN.H	Half-Precision Floating-Point Symbol Fetch and Invert Injection Instruction	3
FSGNJPX.	Half-Precision Floating-Point Symbol Alias Injection Instruction	3
data transmission instruction		
FMV.X.H	Half-precision floating-point read transfer instruction	1+1
FMV.H.X	Half-precision floating-point write transfer instruction	1+1
compare instruction		
FMIN.	Half-Precision Floating Point Minimize Instruction	3
FMAX.	Half-Precision Floating Point Maximize Instruction	3
FEQ.H	Half-precision floating-point compare equal instruction	Split implementation 3+1
FLT.	Half-precision floating-point compare less than instruction	Split implementation 3+1

Continued on next page

Table 3.12 - Continued from previous page

FLE.	Half-precision floating-point comparison is less than or equal to the instruction	Split implementation 3+1
data type conversion instruction		
FCVT.	Half-precision floating-point to single-precision floating-point fingers honorific title	3
FCVT.	Single-precision floating-point to half-precision floating-point fingers honorific title	3
FCVT.	Half-precision floating-point to double-precision floating-point fingers honorific title	3
FCVT.	Double-precision floating-point to half-precision floating-point fingers honorific title	3
FCVT.W.H	Half-precision floating-point to signed integer fingers honorific title	Split implementation 3+1
FCVT.	Half-precision floating-point to unsigned integer fingers honorific title	Split implementation 3+1
FCVT.H.W	Signed Integer to Half-Precision Floating Point Finger honorific title	Split implementation 3+1
FCVT.H.WU	Unsigned Integer to Half-Precision Floating Point Function honorific title	Split implementation 3+1
FCVT.	Half-precision floating-point to signed long integer conversion directives	Split implementation 3+1
FCVT.	Half-precision floating point to unsigned long integer conversion directives	Split implementation 3+1
FCVT.	Signed Long Integer to Half-Precision Floating Point Conversion directives	Split Executive 3+1

Chapter 3

Instruction

FCVT. Set	Unsigned long integer to half-precision floating point conversion directives	
memory storage instruction		
FLH	Half-precision floating-point load instruction	WEAK ORDER LOAD: >=3 STORE: 1 STRONG ORDER indeterminate period
FSH	Half-precision floating-point memory instruction	ibid
Floating Point Number Classification Instruction		
FCLASS.	Single-precision floating-point sorting instructions	1+1

Refer to [Appendix B-6 Floating-Point Half-Precision Instruction Terminology](#) for specific instruction descriptions and definitions.

Chapter 4 Processor Modes and Registers

4.1 processor mode

The C910 supports three **privileged modes** for RISC-V: machine mode, superuser mode, and user mode. The processor executes the program in the machine mode after reset. The three operation modes correspond to different operation privileges, and the differences are mainly reflected in the following aspects:

1. Access to registers;
 2. Use of Privileged Directives;
 3. Access to memory space.
- **User mode privileges are minimal.**

Ordinary user programs are allowed to access only the registers assigned to the ordinary user mode. Access to privileged information by normal user programs is avoided, and the operating system provides management and services to normal user programs by coordinating their behavior.

- **Superuser mode has higher privileges than user mode, but lower than machine mode.**

Programs running in superuser mode do not have access to machine mode control registers and are restricted by PMP. Using page-based virtual memory, this feature forms the core of superuser mode.

- **Machine mode has the highest authority.**

Programs running in machine mode have full access to memory, I/O, and some of the underlying functionality necessary to boot and configure the system. By default (exceptions and interrupts are not downgraded) exceptions and interrupts that occur in any mode are switched to machine mode for response.

Most instructions can be executed in all three modes, but some privileged instructions that have a significant impact on the system can only be executed in superuser mode or machine mode. Refer to [Appendix A, Standard Instruction Terminology](#), and [Appendix B, Flathead Extended Instruction Terminology](#), for specific information on the execution privileges of instructions.

The processor's operating mode changes in response to an exception (the privileged mode in which the exception is responded to is different from the privileged mode in which the exception occurred) enters a higher privileged mode to respond to the exception, and returns to a lower privileged mode after the exception is responded to.

4.2 Register View

The register view of the C910 is shown in Figure 4.1:



Figure 4.1: Register View

4.3 general-purpose register

The C910 has 32 64-bit general-purpose registers with the same functional definitions as RISC-V, as shown in Table 4.1.

Table 4.1: General Purpose Registers

processor register	ABI Name	descriptive
x0	zero	Hardware Tie 0
x1	ra	Return address
x2	sp	stack pointer
x3	gp	global pointer
x4	tp	thread pointer
x5	t0	Temporary/alternate link registers
x6-7	t1-2	Temporary registers
x8	s0/fp	Reserved Register/Frame Pointer
x9	s1	reserved register
x10-11	a0-1	Function Parameters/Return Values
x12-17	a2-7	function parameter
x18-27	s2-11	reserved register
x28-31	t3-6	Temporary registers

The general-purpose registers are used to hold instruction operands, instruction execution results, and address information.

4.4 floating point register

In addition to supporting the standard RV64FD instruction set, the C910's floating-point unit extends support for floating-point half-precision calculations with 32 independent 64-bit floating-point registers that can be accessed in Normal, Super User, and Machine modes.

Table 4.2: Floating Point Registers

processor register	ABI Name	descriptive

Chapter 4 Processor Modes and Registers

r		
f0-7	ft0-7	Floating Point Temporary Register
f8-9	fs0-1	Floating Point Reserved Register
f10-11	fa0-1	Floating point parameters/return values
f12-17	fa2-7	floating-point parameter
f18-27	fs2-11	Floating Point Reserved Register
f28-31	ft8-11	Floating Point Temporary Register

The floating-point register f0, unlike the general-purpose register x0, is not hardware-tied to 0, but is variable like the other floating-point registers. Single-precision floating-point numbers use only the lower 32 bits of the 64-bit floating-point register, and the upper 32 bits must all be 1's or they will be treated as non-numbers; half-precision floating-point

Numbers use only the lower 16 bits of the 64-bit floating-point register; the upper 48 bits must be all ones or they will be treated as nonnumbers.

Chapter 4 Processor Modes and Registers

Adding separate floating-point registers increases register capacity and bandwidth, which in turn increases processor performance. Floating-point load and store instructions must also be added, as well as data transfer instructions between the floating-point and general-purpose registers.

4.4.1 Floating Point Registers and General Purpose Registers Transferring Data

Data transfers between general purpose registers and floating point registers can be accomplished with floating point register transfer instructions. Floating point register transfer instructions include:

- FMV.X.H/FMV.H.X Floating-point register half-precision transfer instructions.
- FMV.X.W/FMV.W.X Floating-point register single-precision transfer instructions.
- FMV.X.D/FMV.D.X Floating-point register double-precision transfer instructions.

Transferring a half/single/double precision floating-point data from a general-purpose register to a floating-point register, the data format does not change as a result of the transfer, so the program can use these registers directly without having to go through a type conversion.

Specific instruction descriptions and definitions can be found in [Appendix A-4 F Instruction Terminology](#).

4.4.2 Maintain consistent register accuracy

Floating-point registers can store half-precision floating-point numbers, single-precision floating-point numbers, double-precision floating-point numbers, and shaped data. As an example, the type of data stored in the floating-point register f1, depending on the last write operation, could be any of the four data types.

The floating-point unit does not do any data format checking on the data type in hardware, and the hardware's parsing of the data format in the floating-point register depends only on the executed floating-point instruction itself, and does not care about the data format used for the last write operation in this register. It is entirely up to the compiler or the program itself to ensure the consistency of the precision of the data in the registers.

4.5 System Control Register

4.5.1 Standard Control Register

This section describes the C910 implementation of the RISC-V standard control registers, which are described according to machine mode, superuser mode, and user mode.

Chapter 4 Processor Modes and Registers

The machine mode control registers defined by the RISC-V standard implemented in the C910 are shown in Table 4.3.

Table 4.3: RISC-V Standard Machine Mode Control Registers

name (of a thing)	read-write access	Register number	descriptive
Machine Mode Information Register Set			
mvendorid	Machine mode read-only	0xF11	Vendor Number Register
marchid	Machine mode read-only	0xF12	Architecture Number Register
mimpid	Machine mode read-only	0xF13	Machine mode hardware implementation numbered registers
mhartid	Machine mode read-only	0xF14	Machine Mode Logic Kernel Number Register

Continued on next page

Table 4.3 - continued from previous page

name (of a thing)	read-write access	Register number	descriptive
Machine Mode Exception Configuration Register Set			
mstatus	Machine mode read/write	0x300	Machine Mode Processor Status Register
misa	Machine mode read/write	0x301	Machine Mode Processor Instruction Set Characterization Registers
medeleg	Machine mode read/write	0x302	Machine Mode Abnormal Degradation Control Register
mdeleg	Machine mode read/write	0x303	Machine Mode Interrupt Degradation Control Register
mie	Machine mode read/write	0x304	Machine Mode Interrupt Enable Control Register
mtvec	Machine mode read/write	0x305	Machine Mode Vector Base Address Register
mcounteren	Machine mode read/write	0x306	Machine Mode Counter Authorization Control Register
mcountinhibit	Machine mode read/write	0x320	Machine Mode Count Disable Register
Machine Mode Exception Handling Register Set			
mscratch	Machine mode read/write	0x340	Machine Mode Exception Temporary Data Backup Register
mepc	Machine mode read/write	0x341	Machine Mode Exception Retention Program Counter
mcause	Machine mode read/write	0x342	Machine Mode Abnormal Event Cause Register
mtval	Machine mode read/write	0x343	Machine Mode Abnormal Event Vector Register
mip	Machine mode read/write	0x344	Machine Mode Interrupt Waiting Status Register
Machine Mode Memory Protection Register Set			
pmpcfg0	Machine mode read/write	0x3A0	Physical Memory Protection Configuration Register 0
pmpaddr0	Machine mode read/write	0x3B0	Physical Memory Protection Base Address Register 0
...			
pmpaddr7	Machine mode read/write	0x3B7	Physical Memory Protection Base Address Register 7
Machine Mode Counter/Timer			
mcycle	Machine mode read/write	0xB00	Machine Mode Cycle Counter
minstret	Machine mode	0xB02	Machine Mode Retirement

Chapter 4 Processor

Modes and Registers

	read/write		Instruction Counter
mhpcounter3	Machine mode read/write	0xB03	Machine Mode Counter 3
...			
mhpcounter31	Machine mode read/write	0xB1F	Machine Mode Counter 31
Machine Mode Counter Configuration Register Set			
mhpmevent3	Machine mode read/write	0x323	Machine Mode Event Selection Register 3
...			
mhpmevent31	Machine mode read/write	0x33F	Machine Mode Event Selection Register 31

The Super User Mode Control Registers defined by the RISC-V standard implemented in the C910 are shown in [Table 4.4](#).

Chapter 4 Processor Modes and Registers

Table 4.4: RISC-V Standard Superuser Mode Control Registers

name (of a thing)	read-write access	Register number	descriptive
Superuser Mode Exception Configuration Register Group			
sstatus	Superuser mode reading and writing	0x100	Super User Mode Processor Status Register
sie	Superuser mode reading and writing	0x104	Super User Mode Interrupt Enable Control Register
stvec	Superuser mode reading and writing	0x105	Super User Mode Vector Base Address Register
scounteren	Superuser mode reading and writing	0x106	Super User Mode Counter Enable Control Register
Super User Mode Exception Handling Register Set			
sscratch	Superuser mode reading and writing	0x140	Super User Mode Exception Temporary Data Backup Registers
sepc	Superuser mode reading and writing	0x141	Super User Mode Exception Retention Program Counter
scause	Superuser mode reading and writing	0x142	Super User Mode Exception Event Cause Register
stval	Superuser mode reading and writing	0x143	Super User Mode Exception Event Vector Register
sip	Superuser mode reading and writing	0x144	Super User Mode Interrupt Waiting Status Register
Super User Mode Address Translation Register Set			
satp	Superuser mode reading and writing	0x180	Superuser Virtual Address Translation and Protection Registers

The user mode control registers defined by the RISC-V standard implemented in the C910

are shown in the *RISC-V Standard User Mode Control Registers*.

name (of a thing)	read-write access	Register number	descriptive
Table 4.5: RISC-V Standard User Mode Control Registers			
User Mode Floating Point Control Register Set			
fflags	User mode read/write	0x001 35	Floating Point Exception Accumulation Status Register
frm	User mode	0x002	Floating-point dynamic

4.5.2 Extended Control Register

This section describes the extended control registers implemented in the C910, described separately by machine mode, super user mode, and user mode.

The extended machine mode control registers in the C910 are shown in Table 4.6.

Table 4.6: C910 Extended Machine Mode Control Registers

name (of a thing)	read-write access	Register number	descriptive
Machine Mode Processor Control and Status Extension Register Set			
mxstatus	Machine mode read/write	0x7C0	Machine Mode Extended Status Register
mhcr	Machine mode read/write	0x7C1	Machine Mode Hardware Configuration Registers
mcor	Machine mode read/write	0x7C2	Machine Mode Hardware Operation Registers
mccr2	Machine mode read/write	0x7C3	Machine Mode L2 Cache Control Registers
mhint	Machine mode read/write	0x7C5	Machine Mode Implicit Operation Register
mrvbr	Machine mode read-only	0x7C7	Machine Mode Reset Vector Base Address Register
mcounterwen	Machine mode read/write	0x7C9	Super User Mode Counter Write Enable Register
mcounterinten	Machine mode read/write	0x7CA	Machine Mode Event Interrupt Enable Register
mcounterof	Machine mode read/write	0x7CB	Overflow Labeling Register on Machine Mode Events
Machine Mode Cache Access to Extended Register Set			
mcins	Machine mode read/write	0x7D2	Machine Mode Cache Instruction Register
mcindex	Machine mode read/write	0x7D3	Machine Mode Cache Access Index Registers
mcdata0	Machine mode read/write	0x7D4	Machine Mode Cache Data Register 0
mcdata1	Machine mode read/write	0x7D5	Machine Mode Cache Data Register 1

Chapter 4 Processor

Modes and Registers

Machine Mode Processor Model Extended Register Set			
mcpuid	Machine mode read-only	0xFC0	Machine Mode Processor Model Register
mapbaddr	Machine mode read-only	0xFC1	On-chip bus base address
Multicore Extended Register Set			
msmpr	Machine mode read/write	0x7F3	Snoop Listener Enable Register

Refer to [Appendix C-1 Machine Mode Control Registers](#) for specific register definitions and functions.

The extended super user mode control registers in the C910 are shown in [Table 4.7](#).

Chapter 4 Processor Modes and Registers

Table 4.7: C910 Extended Superuser Mode Control Registers

name (of a thing)	read-write access	Register number	descriptive
Super User Mode Processor Control and Status Extension Register Set			
sxtatus	Superuser mode reading and writing	0x5C0	Super User Mode Extended Status Register
shcr	Superuser mode reading and writing	0x5C1	Super User Mode Hardware Control Registers
scounterinten	Superuser mode reading and writing	0x5C4	Super User Mode Event Interrupt Enable Register
scounterof	Superuser mode reading and writing	0x5C5	Overflow Labeling Register on Super User Mode Events
scycle	Superuser mode reading and writing	0x5E0	Super User Mode Cycle Counter
...			
shpmcounter31	Superuser mode reading and writing	0x5FF	Super User Mode Counter 31
Super User Mode MMU Extended Register Set			
smir	Superuser mode reading and writing	0x9C0	Super User Mode MMU Index Register
smel	Superuser mode reading and writing	0x9C1	Super User Mode MMU EntryLo Register
smeh	Superuser mode reading and writing	0x9C2	Super User Mode MMU EntryHi Register
smcir	Superuser mode reading and writing	0x9C3	Super User Mode MMU Control Registers

Refer to [Appendix C-2 Super User Mode Control](#) Registers for specific register definitions and functions.

The extended user mode control registers in the C910 are shown in [Table 4.8](#).

Table 4.8: C910 Extended User Mode Control Registers

name	read-write	Register	descriptive
-------------	-------------------	-----------------	--------------------

(of a thing)	access	number	
User Mode Extended Floating Point Control Register Set			
fxcr	User mode read/write	0x800	User Mode Extended Floating Point Control Register

Refer to Appendix C-3 User Mode Control Registers for specific register definitions and functions.

4.6 data format

4.6.1 Integer data

format

The values inside the registers are not differentiated by size, but only by signed and unsigned. The format of the registers is right-to-left logical low-to-high, as shown in Figure 4.2.

4.6.2 Floating Point Data Format

The C910 floating-point unit complies with the RISC-V standard and is compatible with the ANSI/IEEE 754-2008 floating-point standard. It supports half-precision, single-precision, and double-precision floating-point operations, and its data format is shown in Figure 4.3. The data format is shown in Figure 4.3. The single-precision data uses only the lower 32 bits of the 64-bit floating-point registers, and the upper 32 bits require

All 1's or they will be treated as non-numbers; half-precision data uses only the lower 16 bits of the 64-bit floating-point register, and the upper 48 bits need to be all 1's or they will be treated as non-numbers.

Chapter 4 Processor Modes and Registers

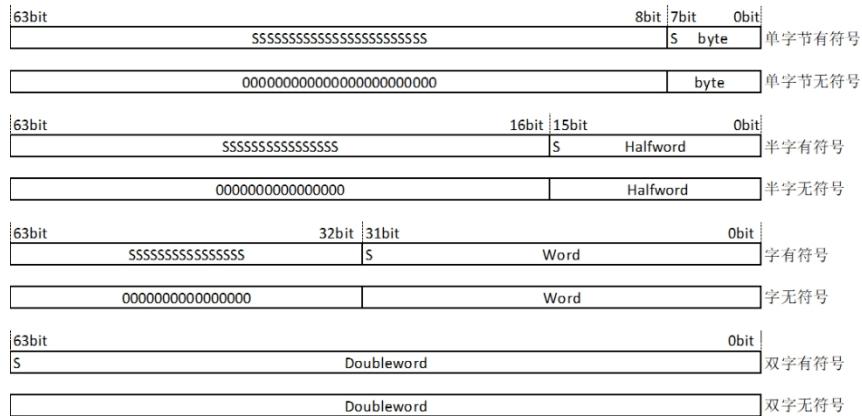


Figure 4.2: Organization of Integer Data in Registers

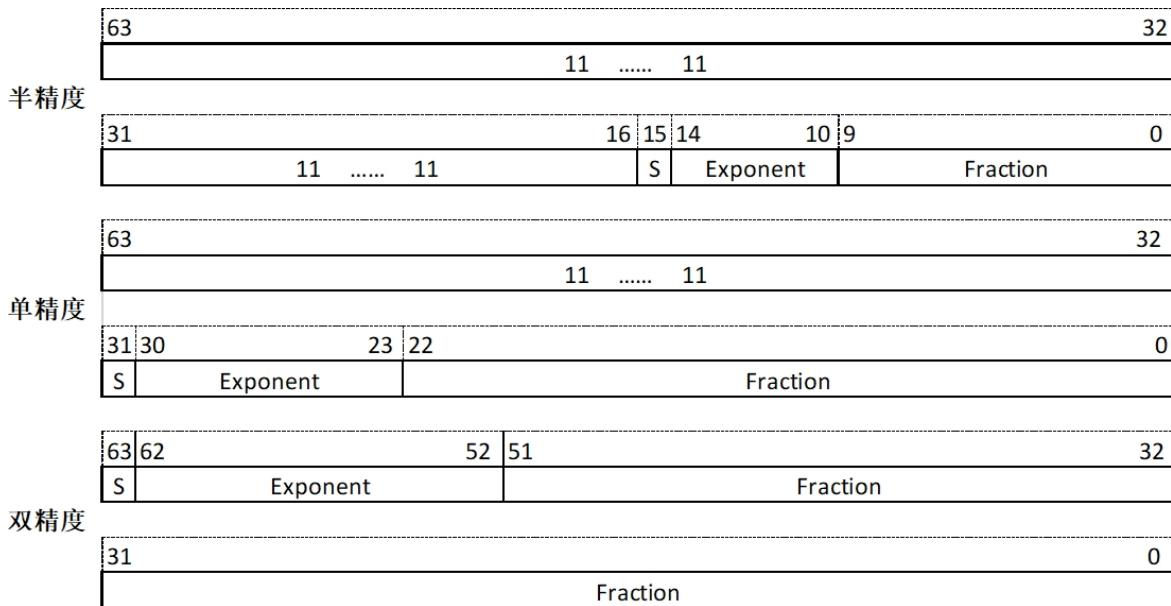


Figure 4.3: Organization of Floating Point Data in Registers

4.7 Big and small end

The concept of big-end and little-end is relative to the format of memory data storage. The low end of the high address byte stored into physical memory is defined as the big end; the high end of the high address byte stored into physical memory is defined as the little end, as shown in Figure 4.4.

A+7	A+6	A+5	A+4	A+3	A+2	A+1	A	
Byte 7	Byte 6	Byte 5	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0	Double word at A
Byte 7	Byte 6	Byte 5	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0	Word at A
Byte 7	Byte 6	Byte 5	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0	Half word at A
Byte 7	Byte 6	Byte 5	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0	Byte at A

a) 小端模式

A+4	A+5	A+6	A+7	A	A+1	A+2	A+3	
Byte 3	Byte 2	Byte 1	Byte 0	Byte 7	Byte 6	Byte 5	Byte 4	Double word at A
Byte 4	Byte 5	Byte 6	Byte 7	Byte 3	Byte 2	Byte 1	Byte 0	Word at A
Byte 6	Byte 7	Byte 4	Byte 5	Byte 1	Byte 0	Byte 3	Byte 2	Half word at A
Byte 7	Byte 6	Byte 5	Byte 4	Byte 0	Byte 1	Byte 2	Byte 3	Byte at A

b) 大端V1模式

A+7	A+6	A+5	A+4	A+3	A+2	A+1	A	
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Double word at A
Byte 4	Byte 5	Byte 6	Byte 7	Byte 0	Byte 1	Byte 2	Byte 3	Word at A
Byte 6	Byte 7	Byte 4	Byte 5	Byte 2	Byte 3	Byte 0	Byte 1	Half word at A
Byte 7	Byte 6	Byte 5	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0	Byte at A

b) 大端V2模式

Figure 4.4: Organization of data in memory

The C910 supports only small-end mode, which supports standard complementary binary integers. The length of each instruction operand can either be explicitly encoded in the program (load/store instructions), or implicitly in instruction operations (index operation, byte extraction). Typically, instructions receive 64-bit operands and produce 64-bit results.

Chapter 5 Exceptions and Interruptions

5.1 summarize

Exception handling (including instruction exceptions and external interrupts) is an important function of the processor, and is used to divert the processor to the processing of certain abnormal events when they occur. These events include hardware errors, instruction execution errors, user program requests for service, and so on.

The key to exception handling is to preserve the current running state of the CPU when an exception occurs, and to restore the preexception state when exiting exception handling. Exceptions can be recognized at various stages of the instruction pipeline, and the CPU hardware ensures that subsequent instructions do not change the state of the CPU. Exceptions are handled on instruction boundaries, i.e., the CPU responds to the exception when the instruction retires and saves the address of the instruction that will be executed when it exits exception handling. Even if an exception instruction is recognized before it retires, the exception is not handled until the corresponding instruction retires. For the correct functioning of the program, the CPU avoids repeating the execution of instructions that have completed execution after the exception handling is finished.

To respond to an exception in machine mode, for example, the steps are as follows("exception" here refers to instruction exceptions and external interrupts in general)

Step 1: The processor saves the PC to the mepc.

Step 2: Update mcause and mtval based on the type of exception that occurred.

Step 3: Save the interrupt enable bit MIE of mstatus to MPIE, clear MIE to zero, and disable responding to interrupts.

Step 4: Save the privilege mode before the exception occurred to the MPP of mstatus and switch to machine mode.

Step 5: Based on the base address and pattern in mtvec, the exception service program entry address is obtained. The processor starts execution at the first instruction of the exception service program.

The C910 complies with the RISC-V standard exception vector table as shown in [Table 5.1](#).

Table 5.1: Exception and Interrupt Vector Assignments

interrupt marker	Exception Vector Number	descriptive
1	0	unrealized

1	1	Superuser mode software interruption
1	2	reservations
1	3	Machine mode software interrupt
1	4	unrealized
1	5	Super User Mode Timer Interrupt
1	6	reservations

Continued on next page

Chapter 5 Exceptions and Interruptions

Table 5.1 - continued from previous page

interrupt marker	Exception Vector Number	descriptive
1	7	Machine mode timer interrupt
1	8	unrealized
1	9	Super User Mode External Interrupt
1	10	reservations
1	11	Machine mode external interrupt
1	17	Performance Detection Overflow Interrupt
1	(sth. or sb) else	reservations
0	0	unrealized
0	1	Fetch command access error exception
0	2	Illegal instruction exception
0	3	Debug breakpoint exception
0	4	Load instruction unaligned access exception
0	5	Load instruction access error exception
0	6	Stored/atomic instruction unaligned access exception
0	7	Memory/Atomic Instruction Access Error Exception
0	8	User mode environment call exception
0	9	Superuser mode environment call exception
0	10	reservations
0	11	Machine mode environment call exception
0	12	Fetch Finger Page Error Exception
0	13	Load command page error exception
0	14	reservations
0	15	Store/Atomic Instruction Page Error Exception
0	>= 16	reservations

The C910 supports delegation of exceptions and interrupts, which require the processor to switch to machine mode when an exception or interrupt occurs in superuser mode, resulting in a loss of performance. When an exception or interrupt occurs in superuser mode, the processor needs to switch to machine mode to respond, and the mode switching causes a loss of processor performance. The delegation mechanism supports configuring interrupts and exceptions to respond in superuser mode. The Delegation mechanism supports configuring interrupts and exceptions to be responded to in superuser mode, where exceptions occurring in machine mode are not controlled by delegation and are responded to only in machine mode. Machine mode external interrupts, machine mode software interrupts, and machine mode timer interrupts do not support degradation to superuser mode responses; all other interrupts can be degraded to the superuser mode state. Degraded interrupts are not responded to in machine mode.

Responds to all eligible interrupts and exceptions in both Super User Mode and User Mode. For interrupts and exceptions that are not degraded, they are processed in machine mode, updating the machine mode exception handling register. Degraded interrupts and exceptions are responded to in super user mode, updating the super user mode exception handling register.

5.2 exceptions

5.2.1 Exception Response

To respond to an exception in machine mode, for example, the steps are as follows ("exception" here refers specifically to events such as illegal commands, access errors, etc.)

Step 1: The processor saves the PC on which the exception occurred to the mepc.

Step 2: Set the interrupt flag of mcause to 0, write the exception number to mcause, and update mtval according to the rules in [Table 5.2](#).

Step 3: Save the interrupt enable bit MIE of mstatus to MPIE, clear MIE to zero, and disable responding to interrupts.

Step 4: Save the privilege mode before the exception occurred to the MPP of mstatus and switch to machine mode.

Step 5: The PC fetches the instruction from mtvec.Base and executes it. Typically, the fetched instruction is a jump instruction that jumps to the top-level handler function. This function analyzes the mcause to get the exception number and calls the handler function corresponding to that number.

Table 5.2: Updates to the mtval on an exception

Exception Vector Number	Exception type	mtval Update value
1	Fetch command access error exception	Fetch refers to the virtual address of the access command code
2	Illegal instruction exception	0
3	Debug breakpoint exception	Virtual address for load access
4	Load instruction unaligned access exception	0
5	Load instruction access error exception	Virtual address for storage/atomic access
6	Stored/atomic instruction unaligned access exception	0
7	Memory/Atomic Instruction Access Error Exception	0
8	User mode environment call exception	0
9	Superuser mode	0

Chapter 5 Exceptions and Interruptions

	environment call exception	
11	Machine mode environment call exception	0
12	Fetch Finger Page Error Exception	Fetch refers to the virtual address of the access
13	Load command page error exception	Virtual address for load access
15	Store/Atomic Instruction Page Error Exception	Virtual address for storage/atomic access

5.2.2 Exception return

An exception return is achieved by executing the mret instruction. At this point, the processor does the following:

- Restore mepc to PC(mepc saves the PC on which the exception occurred. by adjusting mepc, the instruction on which the exception occurred can be skipped; if it is not adjusted, the instruction on which the exception occurred is re-executed)
- Restore mstatus.MPIE to mstatus.MIE.
- Restore the privilege mode from mstatus.MPP before the exception occurred.

Chapter 5

Exceptions and Interruptions

5.2.3 Imprecise anomaly

In rare cases, a processor may exhibit the behavior of an "imprecise exception". An imprecise exception is one in which the mepc does not point to the instruction that triggered the exception. For example, the CPU executes a load instruction, and the bus returns Error. because the pipeline has the characteristic of rapid instruction retirement, when the bus returns Error, the load instruction has already retired, so the mepc points to a subsequent instruction, not to the load instruction itself.

It is worth noting that inexact exceptions are extremely unlikely to occur in a real system, and when they do, they imply a fatal error.

5.3 disruptions

5.3.1 interrupt priority

When more than one interrupt request occurs at the same time, the priority is determined in the following order (from highest to lowest)

- M-state external interrupt
- M-state software interrupt
- M-state timer interrupt
- S-state external interrupt
- S-State Software Interrupts
- S-State Timer Interrupt
- PMU Overflow Interrupt
- S-state external interrupt (demoted)
- S-Statement Software Interrupt (Degraded)
- S-state timer interrupt (demoted)
- PMU overflow interrupt (demoted)

5.3.2 Interrupt Response

As an example, the steps to respond to an interrupt in machine mode are:

Step 1: The processor finishes executing the current instruction and saves the PC of the next instruction to mepc.

Step 2: Set the interrupt flag of mcause to 1, write the interrupt number to mcause, and update mtval to 0. **Step 3:** Save the interrupt enable bit MIE of

Chapter 5 Exceptions and Interruptions

mstatus to the MPIE, clear the MIE to zero, and disable responding to interrupts.

Step 4: Save the privilege mode before the interrupt occurs into the MPP of mstatus and switch to machine mode.

Step 5 (mtvec.Mode=0, passthrough interrupt) the PC fetches the instruction from mtvec.Base and executes it. Typically, the fetched instruction is a jump instruction that jumps to the top-level handler function. This function gets the interrupt number by analyzing mcause and calls the handler function corresponding to that number.

Chapter 5

Exceptions and

Interruptions

Step 5 (mtvec.Mode=1, vector interrupt) the PC fetches the instruction from mtvec.Base + 4

* interrupt number and executes it. Usually, the fetched instruction is a jump instruction that jumps to the handler function of the corresponding interrupt.

5.3.3 Interrupt Return

An interrupt return is accomplished by executing the mret instruction. At this point, the processor does the following:

- Restore mepc to PC(mepc saves the PC for the next command, so no adjustment is needed)
- Restore mstatus.MPIE to mstatus.MIE.
- Restore the privilege mode from mstatus.MPP that existed before the interruption occurred.

Chapter 6: Memory Modeling

6.1 Memory Model Overview

6.1.1 Memory Properties

The C910 supports two memory types, Memory and Device, distinguished by SO bits. Among them, Memory type is characterized by supporting speculative execution and disordered execution, and is further classified into Cacheable memory and Non-cacheable memory according to Cacheable (C) or not, and Device type is characterized by Non-speculative execution and must be executed in order, so Device must have the attribute of Non-cacheable. Devices are categorized into Bufferable devices and Non-bufferable devices according to whether they are Bufferable (B) or not. Bufferable means that a write access allows a write response to be returned quickly at an intermediate node; conversely, Non-bufferable means that a write access allows a write response to be returned quickly at an intermediate node; conversely, Non-bufferable means that a write access allows a write response to be returned quickly at an intermediate node. Bufferable means that a write access is allowed to return a write response quickly at some intermediate node; conversely, non-bufferable means that a write access returns a write response only after the final device has actually written.

In order to support data sharing among multiple cores, C910 adds the attribute of shareable pages (Shareable, SH). For a shareable page, it means that the page is shared among multiple cores and the data consistency is maintained by the hardware; for a non-shareable page, it means that the page is exclusively occupied by a single core and the hardware is not required to maintain the data consistency. The multicore data consistency of a non-shareable page needs to be maintained by software.

The attributes of SH can be configured for cacheable memory, while the SH attributes of non-cacheable memory types and peripheral types are not configurable and are fixed as shareable.

In addition, the C910 supports configuration of secure page attributes (Security, SEC).

Table 6.1 gives the page attributes corresponding to each memory type.

Table 6.1: Classification of Memory Types

Memory Type	SO	C	B	SH	SEC
cacheable	0	1	1	mis matc	mism atch

				h	
No high speed memory	0	0	1	1	mismatch
Cacheable peripherals	1	0	1	1	mismatch
Non-cacheable peripherals	1	0	0	1	mismatch

The CPU can obtain page attributes for an address in two ways: sysmap.h or page table entries (pte) The details are described below:

1. In all cases where virtual-to-physical address translation is not performed (i.e., machine mode or MMU off) the page attributes of the address are defined by the sysmap.h decides.
2. In all cases where virtual-to-physical address translation is performed (i.e., non-machine mode and MMU is on) the source of the page attributes of the address depends on mxstatus.maee. If maee is on, the page attributes of the address are determined by the page attributes of the extension in the corresponding pte. If

Chapter 6:

Memory

Models

macro off, then the page attributes for the address are determined by sysmap.h.

sysmap.h is the configuration file of C910 extension, which is open to the user, and the user can define the page attributes of different address segments according to their needs.

sysmap.h supports the setting of attributes for 8 address spaces. The upper address limit (not included) of the i-th ($i=0\sim 7$) address space is defined by the macro SYSMAP_BASE_ADDR*i* ($i=0\sim 7$), and the lower address limit (included) is defined by SYSMAP_BASE_ADDR($i-1$), that is:

`SYSMAP_BASE_ADDR(i-1) <= i - t h address space address < SYSMAP_BASE_ADDRi`

The lower limit of the 0th address space is 0x0. The address attributes for memory addresses that do not fall within the 8 address bands set in the sysmap.h file default to cacheable/bufferable/shareable/security. The upper and lower boundaries of each address space are 4KB aligned, so the macro SYSMAP_BASE_ADDR*i* defines the upper 28 bits of the address. The upper and lower boundaries of each address space are 4KB aligned, so the macro SYSMAP_BASE_ADDR*i* defines the upper 28 bits of the address.

The attributes of the address falling in the i-th ($i=0\sim 7$) address space are defined by the macro SYSMAP_FLAG*i* ($i=0\sim 7$), and the attributes are arranged as shown in Figure 6.1 shown:

4	3	2	1	0
Strong order	Cacheable	Bufferable	Shareable	Security

Figure 6.1: sysmap.h Address Attribute Format

6.1.2 memory consistency model

C910MP uses a loose memory consistency model (Weak Memory Ordering) which is defined as follows:

- Each CORE guarantees sequential access to the same address, including read-after-read, write-after-write, read-after-write, and write-after-read; the
- Each CORE relaxes the sequential nature of different address accesses, including read-after-read, write-after-write, read-after-write, and write-after-read; the
- Guaranteeing the atomicity of other-multi-copy requires that when a nucleus can access the write data of another nucleus, it guarantees that the other nucleus can also access that write data at that time; and when a nucleus can access the write data of its own nucleus, it does not require that the other nucleus can also access that write data at that time.

Due to the Weak Memory Ordering model, the actual order of memory reads and writes across multiple cores can be inconsistent with the order of accesses given by the program. Therefore, the C910 extends the SYNC instruction to allow software to enforce the ordering of memory accesses.

The SYNC instruction limits the execution order of all instructions, ensuring that all

Chapter 6:

Memory Models

Instructions preceding the SYNC instruction must be completed before the SYNC instruction is executed. In addition, the SYNC instruction can synchronize the instruction memory additionally, that is, to clear the pipeline when the instruction preceding the SYNC instruction finishes and fetch the instruction again. The specific instructions are shown in Table 6.2.

Table 6.2: SYNC Command Descriptions

mnemon ic sign	command description	scope (computing)
SYNC.IS	Synchronize data and instruction memory	Shareable
SYNC.	Synchronize data and instruction memory	Non-shareable
SYNC.	Synchronize data memory	Shareable
SYNC	Synchronize data memory	Non-shareable

6.2 VMMS

6.2.1 MMU

Overview

The C910 MMU (Memory Management Unit) is compatible with the RISC-V SV39 standard. The main functions are.

- **Address Translation:** Converts a virtual address (39 bits) to a physical address (40 bits)
- **Page protection:** by checking read/write/execute permissions for visitors to a page.
- **Page attribute management:** extend the address attribute bit to get the corresponding attributes of the page according to the access address for further use by the system.

6.2.2 TLB Forms of organization

The MMU mainly utilizes the TLB (Translation Look-aside Buffer) to realize the above functions. the TLB takes the virtual address used by the CPU to access the memory as input, checks the page attributes of the TLB before translation, and then outputs the physical address corresponding to the virtual address.

The C910 MMU employs two levels of TLBs, the first level is the uTLB for instructions I-uTLB and data D-uTLB, and the second level is the jTLB. after processor reset, the hardware will invalidate all the table entries of the uTLBs and jTLBs, and the software does not need to do the initialization.

The I-uTLB has 32 fully-associated table entries and can store a mix of 4K, 2M, and 1G page sizes, taking the finger request to hit the I-uTLB.

When the beat can get the physical address and the corresponding permission attributes.

D-uTLB has 17 fully-associated table entries that can store a mix of 4K, 2M, and 1G page sizes, and load and store request hits.

When D-uTLB is used, the physical address and the corresponding privilege attributes can be obtained when the shot is taken.

The jTLB is a command and data shared, 4-way group concatenated structure with 1024 table entries, and can store a mixture of 4K, 2M and 1G page sizes. uTLB is missing, and when the jTLB is hit, the fastest return time is 3 cycles to the physical address and the corresponding privilege attribute.

6.2.3 Address translation process

The main function of MMU is to translate virtual addresses into physical addresses and perform corresponding privilege checking, and the specific address mapping relationship and corresponding privileges are configured by the operating system and stored in the page table. The specific address mapping relationship and corresponding privileges are configured by the operating system and stored in the page table.C910 adopts up to three levels of page table index to

Chapter 6:

Memory

Models

realize address translation. Access to the first level page table gets the base address and corresponding permission attributes of the second level page table; access to the second level page table gets the base address and corresponding permission attributes of the third level page table; access to the third level page table gets the final physical address and corresponding permission attributes. Each level of access has the possibility of getting the final physical address, i.e., the leaf table entry. The virtual page number VPN has 27-bits, divided equally into three 9-bit VPNs [i], with each access using a portion of the VPN for indexing.

The contents of the leaf table entries (i.e., the physical addresses and corresponding privilege attributes derived from virtual address translation) are cached in the TLB to accelerate address translation. If the uTLB is mismatched, the jTLB is accessed, and if the jTLB is further mismatched, the MMU initiates a Hardware Page Table Walk and accesses memory to obtain the final address translation result.

The page table is used to store the entry address of the subordinate page table or the physical information of the final page table, and its structure is as follows:

Flags - 9:0 bit page attributes

Functions as described in the *MMU EntryLo Register (SMEL)*.

Flags - 63:59 bit page attributes

Chapter 6:

Memory

Models

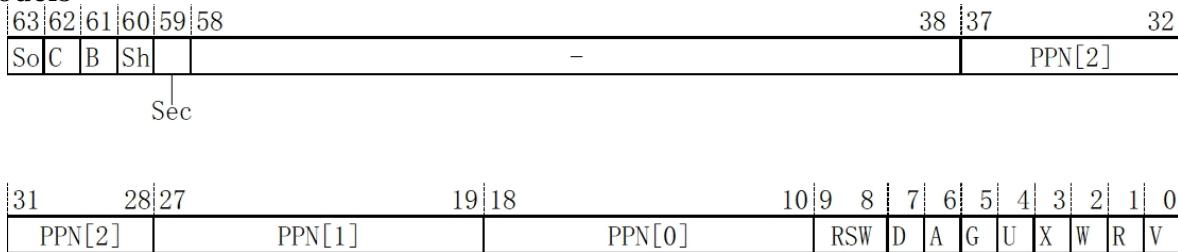


Figure 6.2: Page Table Structure

C910 Custom page attribute, present when MAEE e n a b l e is turned on in the mxstatus register, functions as [MMU EntryLo registers \(SMEL\)](#) as described in the

PPN - Page Physical Address

PPN[i] represents the PPN value corresponding to the three levels of page table conversion, respectively. The detailed flow of address conversion is described as follows:

To access a virtual address, the CPU obtains the physical address and related attributes directly from the TLB if the TLB is hit. If the TLB is missing, the specific steps for address translation are:

1. According to SATP.PPN and VPN[2] to get the first level page table access address {SATP.PPN, VPN[2], 3^b0}, use this address to access the Dcache/memory to get the 64-bit first level page table PTE; the
2. Check whether the PTE meets the PMP authority, if not, then generate the corresponding access error exception; if it meets, then judge whether X/W/R-bit meets the conditions of leaf page table according to the rules shown in [Table 6.4](#), if it meets the conditions of leaf page table, then it means that the final physical address has been found, go to step 3; if it does not meet the conditions, then go to step 1, use PTE.PPN to splice the next level VPN [PPN, and then splice 3^b0 to get the next level of page table access address to continue accessing Dcache/memory;
3. The leaf page table is found, combined with the X/W/R/L bits in the PMP and the X/W/R bits in the PTE to get the least privilege for both for permission checking, and the contents of the PTE are backfilled into the JTLB;
4. During any step of the PMP check, if there is a privilege violation, a corresponding access error exception is generated based on the type of access;
5. If the leaf page table is obtained but: the access type violates the A/D/X/W/R/U-bit setting, a corresponding page fault exception is generated; if the leaf page table is not obtained even after three accesses are completed, a corresponding page fault exception is generated; if an access error response is obtained during the access to Dcache/memory, a page fault exception is generated. page fault exception.

Chapter 6:

Memory Models 6. If you get a leaf page table but the number of accesses is less than 3, you get a large page table. Check if the PPN of the large page table is aligned according to the page table size, if not, a page fault exception is thrown.

6.2.4 System Control Register

In addition to supporting the standard SATP registers, the C910 MMU also customizes and extends the SMIR, SMCIR, SMEL and SMEH control registers. Users can read, write, query and invalidate TLBs directly through these extended registers.

Chapter 6:

Memory

Models

6.2.4.1 MMU Address Translation Register (SATP)

SATP is the SV39 specification MMU control register.

63	60 59		44 43	32
Mode		ASID	-	
31	28 27		0	
-		PPN		

Figure 6.3: SATP Register Description

Mode - MMU Address Translation

Mode

Table 6.3: MMU Address Translation Modes

RV64		
Value	Name	Description
0	Bare	No translation or protection
1-7	-	Reserved
8	Sv39	Page-based 39-bit virtual addressing
9	Sv48	Page-based 48-bit virtual addressing
10	Sv57	Reserved for page-based 57-bit virtual addressing
11	Sv64	Reserved for page-based 64-bit virtual addressing
12-15	-	Reserved

ASID - Current ASID

Indicates the ASID number of the current program.

PPN - Hardware Backfill Root PPN

PPN used for first stage hardware backfill.

6.2.4.2 MMU Control Register (SMCIR)

The SMCIR register implements a variety of operations on the MMU, including TLB lookup, TLB read/write, and TLB invalid operations.

TLBP: TLB Query

Query the TLB based on the EntryHi register.

When the query hits, the Index register is updated with the serial number of the TLB.

TLBR: TLB Read

Based on the Index register index, the values of the corresponding TLB table entries are read and used to update the SMEH and SMEL registers.

Chapter 6:

Memory

Models

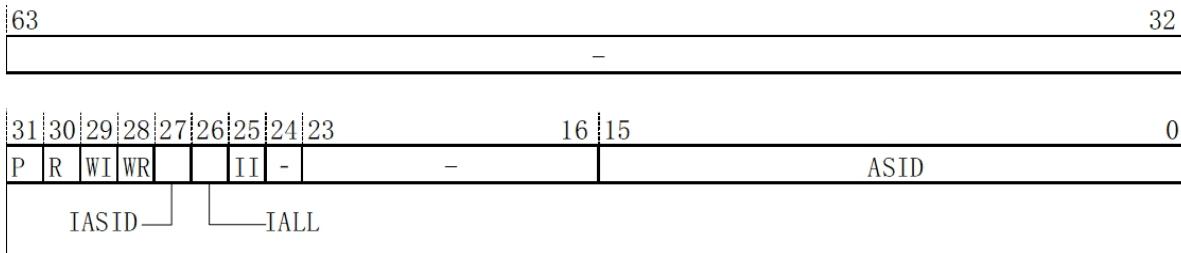


Figure 6.4: SMCIR Register Description

TLBWI: TLB Index Writing

Write registers SMEH and SMEL to TLB corresponding table entries according to Index register index value.

TLBWR: TLB Random Write

Write registers SMEH, SMEL to TLB corresponding table entries according to Random register index value.

TLBIASID: TLB Invalid according to ASID

All TLB table entries matching the ASID are invalid.

TLBIALL: TLB Initialization

Invalidate all TLB table entries, initialize.

TLBII: TLB Invalid according to Index

Invalidates the corresponding TLB table entry based on the Index register index value.

TLBIAW: TLB Invalid under the World

Invalidates all TLB table entries corresponding to trusted and untrusted worlds.

This bit is only relevant when configured with TEE extensions, which are not currently supported by the C910.

ASID: ASID number

TLBIASID operations use this ASID for matching. The SMCIR register implements a variety of operations on the MMU, including TLB

Find, TLB Read/Write, TLB Invalid, and other operations.

TLBP - TLB queries

Query the TLB based on the EntryHi register, and when the query hits, update the Index register with the serial number of the TLB.

TLBR - TLB Read

Based on the Index register index, read the values of the corresponding TLB table entries and update the SMEH and SMEL registers with these values.

Chapter 6:

Memory

TLBWI - TLB Index Write

Models

Write registers SMEH, SMEL to the corresponding table entries of TLB according to the index value of Index register.

TLBWR - TLB Write Randomly

Chapter 6:

Memory

Models

Write registers SMEH, SMEL to TLB corresponding table entries according to Random register index value.

TLBIASID - TLB Invalid according to ASID

All TLB table entries matching the ASID are invalid.

TLBIALL - TLB Initialization

Invalidate all TLB table entries, initialize.

TLBII - TLB Invalid by Indexing

Invalidates the corresponding TLB table entry based on the Index register index value.

ASID - ASID number

The TLBIASID operation uses this ASID for matching.

6.2.4.3 MMU Index Register (SMIR)

MMU index register for indexing TLBs. index of hit table entries is updated when a TLB query is performed. when writing indexes to TLBs, the mapping relationship can be written to the jTLB at the corresponding index by writing the index field of SMIR.

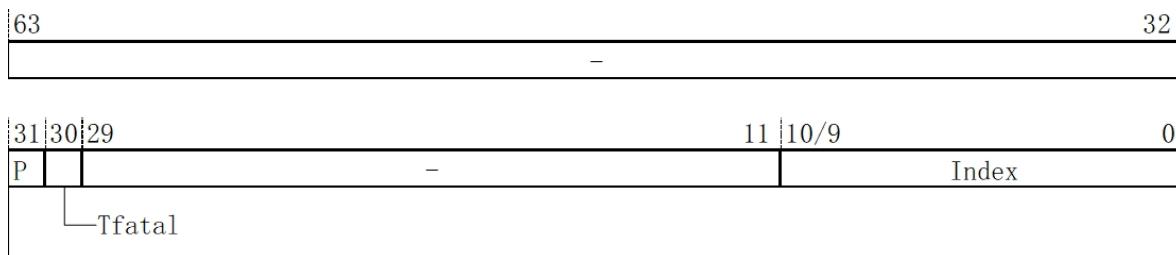


Figure 6.5: SMIR Register Description

P - Probe Failure

0: TLBP query match hit.

1: TLBP query did not hit.

Tfatal - Probe multiple

The execution of the TLBP instruction is whether multiple matches occur.

0: No multiple matches.

1: Multiple matches occurred.

Index - TLB Index

1024-entry configuration: Index[9:8] is the way index, Index[7:0] is the set/entry index; (4 way, 256 entries)

2048-entry configuration: Index[10:9] is the way index, Index[8:0] is the set/entry index; (4 way, 512 entries) 512 entries)

Chapter 6:

Memory

Models

6.2.4.4 MMU EntryHi Register (SMEH)

The SMEH register has two functions: it contains information about the virtual address of the TLB access and the value of the current VPN in case of a TLB exception, and the ASID indicates the process number corresponding to the current page.

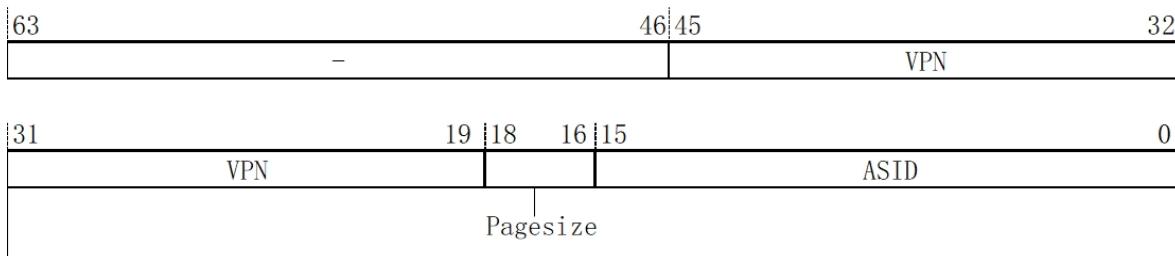


Figure 6.6: SMEH Register Description

VPN - Virtual Page Frame Number

This field is updated in hardware on TLB reads and page error exceptions, and is pre-written by software before software writes the TLB table entry.

Pagesize - page size

Onehot indicates 4K, 2M, 1G page sizes from low to high.

This field is updated in hardware on TLB reads and is pre-written by software before software writes the TLB table entry.

ASID - Address Space Identification

This field is typically used to hold an identification of the current address space as seen by the operating system to distinguish between different processes. It is updated by hardware when the TLB is read, and pre-written by software before software writes the TLB table entry.

6.2.4.5 MMU EntryLo Register (SMEL)

The SMEL contains physical address and page attribute information for TLB access.

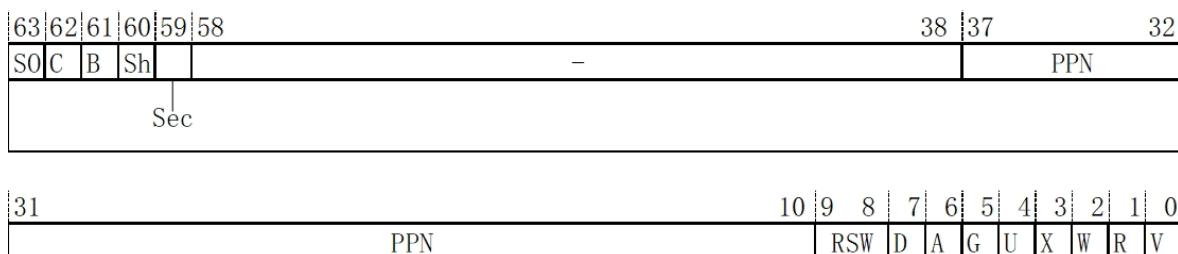


Figure 6.7: SMEL Register Description

Chapter 6:

Memory
PPN - Physical Page
Models

Number, 28-bit SO -

Strong Order

Chapter 6:

Memory

Models

Used to indicate memory requirements for order of access

1[▼]b0: No strong order (Normal memory) 1[▼]b1:

Strong order (Device)

C - Cacheable

1[▼]b0: Uncacheable

1[▼]b1: Cacheable

B - Buffer

1[▼]b0: Unbufferable

1[▼]b1: Bufferable **SH**

- Shareable

Used to characterize

the shared properties

of a page 1[▼]b0:

Unshareable 1[▼]b1:

Shareable

Sec (T - Trustable)

Used to characterize a page as belonging to a trusted or untrusted world, this bit is only meaningful when equipped with the TEE pro extension

1[▼]b0: non-trustalbe

1[▼]b1: trustable

RSW - Reserved for Software

Used to reserve a bit for the software to do custom page table functions. default is 1[▼]1

D-Dirty.

A D bit of 1 indicates whether the page has been rewritten.

1[▼]b0: current page is not written/unwritable

1[▼]b1: current page is written/writable

If the D bit is 0, a write operation to this page triggers a Page Fault (Store) exception, which maintains that the D bit meets the definition of rewritable/writable by software manipulation of the D bit in the exception service program.

A - Accessed

A bit of 1 indicates that the page is accessible. A 0 indicates that the page is not accessible, otherwise a Page Fault (corresponding to the type of access) exception is triggered.

1[▼]b0: Current page not accessible

Chapter 6:

Memory
Models^[lə'mɔdəlz]: Current page accessible

G - Global

Chapter 6:

Memory

Models

Global page identifier, current page can be shared by multiple processes

1^b0 : Non-shared page, process number ASID private

1^b1 : Share page

U - User

User mode access

1^b0 : User mode is not accessible, when user mode is accessed, a page fault exception occurs. **default is 1^b0**

1^b1 : User mode accessible

X W R - Executable, Writable, Readable

Table 6.4: XWR Privilege Descriptions

X	W	R	Meaning
0	0	0	Pointer to next level of page table
0	0	1	Read-only page
0	1	0	Reserved for future use
0	1	1	Read-write page
1	0	0	Execute-only page
1	0	1	Read-execute page
1	1	0	Reserved for future page
1	1	1	Read-write-execute page

V - Valid

Indicates whether the physical page is allocated in memory or not. accessing a page with V=0 will trigger a Page Fault exception.

1^b0 : Current page not assigned

1^b1 : The current page has been assigned.

6.3 Physical memory protection

6.3.1 PMP

Overview

The C910 PMP (Physical Memory Protection) complies with the RISC-V standard. The PMP unit is responsible for checking the access rights to a physical address and determining whether the CPU has read/write/execute rights to the address in the current operating mode.

The main features of the C910 PMP unit are:

- Supports 8 PMP table entries, each identified and indexed by a number from 0-7
- Minimum address segmentation granularity of 4KB

Chapter 6:

Memory

Models

- Supports OFF, TOR, and NAPOT address matching modes, does not support NA4 matching mode

Chapter 6:

Memory

Models

- Supports configuration of readable, writable and executable permissions
- PMP Table Entry Support Software Lock

6.3.2 PMP Control Register

The PMP table entry consists mainly of an 8-bit setup register and a 64-bit address register. All PMP control registers can only be accessed in machine mode, and other modes of access will generate an illegal instruction exception.

6.3.2.1 Physical Memory Protection Setting Register (PMPCFG)

The Physical Memory Protection Settings register provides permission settings for eight table entries.

63	56	55	48	47	40	39	32	31	24	23	16	15	8	7	0
entry7_cfg	entry6_cfg	entry5_cfg	entry4_cfg	entry3_cfg	entry2_cfg	entry1_cfg	entry0_cfg								
8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8

pmcfg0

Figure 6.8: Overall Distribution of Physical Memory Protection Settings Registers

7	6	5	4	3	2	1	0
L(WARL)	0(WARL)	A(WARL)	X(WARL)	W(WARL)	R(WARL)		
1	2	2	1	1	1		

Figure 6.9: Physical Memory Protection Setup Registers

The PMP control registers are described in Table 6.5.

Table 6.5: PMP Control Register Descriptions

classifier for honorifi c people	name (of a thing)	descriptive
0	R	Readable attributes of the table entry: 0: Table entry match address unreadable 1: Table entry matching address readable
1	W	Writable attribute of the table entry: 0: Table entry match address is not writable 1: Table entry matching address writable
2	X	Executable attribute of the table entry: 0: Table entry matching address not executable 1: Table entry matching address executable
4:3	A	The address matching pattern of the table entry: 00: OFF, invalid table entry 01: TOR (Top of range) a pattern that uses the addresses of neighboring table entries as the matching interval 10: NA4 (Naturally aligned four-byte region), a matching pattern with an interval size of 4 bytes, this pattern is not supported 11: NAPOT (Naturally aligned power-of-2 regions) intervals of size power of 2 The matching pattern is at least 4KB.
7	L	Lock enable bit of the table entry: 0: All machine mode accesses will succeed System Mode/User Mode access is judged successful or unsuccessful based on R/W/X. 1: Table entries are locked and modifications cannot be made to the relevant table entries When TOR mode is configured, the address register of its previous table entry cannot be modified either. All modes require a successful access decision based on R/W/X.

For TOR mode, the condition for hitting table entry i is: pmpaddr(i-1) A < pmpaddr(i), assuming access address A. For table entry 0, it uses 0 as the lower boundary.

For NAPOT mode, the relationship between address and interval size is shown in Table 6.6.

Table 6.6: Protected Area Codes

pmpaddr	pmpcfg.	Size of protected areas	note
aaaa_aaa_aaa_aaa_aaa_aaa_aaa_aaa_aaa_aaa_aaa	NA4	4B	unsupp

Chapter 6:

Memory Models

_aaa_aaa				orted
aaaa_aaa_aaa_aaa_aaa_aaa_aaa_aaa0	NAPOT	8B		unsupp orted
aaaa_aaa_aaa_aaa_aaa_aaa_aaa_aaa01	NAPOT	16B		unsupp orted
aaaa_aaaa_aaaa_aaaa_aaaa_aa011	NAPOT	32B		unsupp orted
aaaa_aaa_aaa_aaa_aaa_aaa_aaa_aaa_0111	NAPOT	64B		unsupp orted
aaaa_aaa_aaa_aaa_aaa_aaa_aaa_aaa0_1111	NAPOT	128B		unsupp orted
aaaa_aaa_aaa_aaa_aaa_aaa_aaa_aaa01_1111	NAPOT	256B		unsupp orted
aaaa_aaaa_aaaa_aaaa_aaaa_aa011_1111	NAPOT	512B		unsupp orted

Continued on next page

Chapter 6:
Memory
Models

Table 6.6 - continued from previous page

pmpaddr	pmpcfg.	Size of protected areas	note
aaaa_aaa_aaa_aaa_aaa_aaa_0111_1111	NAPOT	1KB	unsupported
aaaa_aaa_aaa_aaa_aaa_aaa0_1111_1111	NAPOT	2KB	unsupported
aaaa_aaa_aaa_aaa_aaa_aaa01_1111_1111	NAPOT	4KB	be in favor of
aaaa_aaa_aaa_aaa_aaa_aaa_a011_1111_1111	NAPOT	8KB	be in favor of
aaaa_aaa_aaa_aaa_aaa_0111_1111_1111	NAPOT	16KB	be in favor of
aaaa_aaaa_aaaa_aaa0_1111_1111_1111_1111	NAPOT	32KB	be in favor of
aaaa_aaaa_aaaa_aaa01_1111_1111_1111_1111	NAPOT	64KB	be in favor of
aaaa_aaaa_aaaa_aaa011_1111_1111_1111_1111	NAPOT	128KB	be in favor of
aaaa_aaaa_aaaa_aaa0111_1111_1111_1111_1111	NAPOT	256KB	be in favor of
aaaa_aaaa_aaaa_aaa0_1111_1111_1111_1111_1111	NAPOT	512KB	be in favor of
aaaa_aaaa_aaaa_aa01_1111_1111_1111_1111_1111	NAPOT	1M	be in favor of
aaaa_aaaa_aaaa_aaa011_1111_1111_1111_1111_1111	NAPOT	2M	be in favor of
aaaa_aaaa_aaaa_0111_1111_1111_1111_1111_1111	NAPOT	4M	be in favor of
aaaa_aaaa_aaaa_aaa0_1111_1111_1111_1111_1111_1111	NAPOT	8M	be in favor of
aaaa_aaaa_aa01_1111_1111_1111_1111_1111_1111_1111	NAPOT	16M	be in favor of
aaaa_aaaa_aaa011_1111_1111_1111_1111_1111_1111_1111	NAPOT	32M	be in favor of
aaaa_aaaa_0111_1111_1111_1111_1111_1111_1111_1111	NAPOT	64M	be in favor of
aaaa_aaa0_1111_1111_1111_1111_1111_1111_1111_1111	NAPOT	128M	be in favor of
aaaa_aa01_1111_1111_1111_1111_1111_1111_1111_1111	NAPOT	256M	be in favor of
aaaa_a011_1111_1111_1111_1111_1111_1111_1111_1111	NAPOT	512M	be in favor of
aaaa_0111_1111_1111_1111_1111_1111_1111_1111_1111	NAPOT	1G	be in

Chapter 6:

Memory Models

1111			favor of
aaa0_1111_1111_1111_1111_1111_1111_1111_1111	NAPOT	2G	be in favor of
aa01_1111_1111_1111_1111_1111_1111_1111_1111	NAPOT	4G	be in favor of
a011_1111_1111_1111_1111_1111_1111_1111_1111	NAPOT	8G	be in favor of
0111_1111_1111_1111_1111_1111_1111_1111_1111	NAPOT	16G	be in favor of
1111_1111_1111_1111_1111_1111_1111_1111_1111	Reserved	-	be in favor of

It should be noted that the minimum granularity supported by the C910 PMP is 4KB.

6.3.2.2 Physical Memory Protection Address Register (PMPADDR)

PMP implements a total of 8 address registers pmpaddr0-pmpaddr7, which hold the physical addresses of table entries.

RISC-V specifies that the PMP address register holds [39:2] bits of the physical address, because the C910 PMP table entry granularity supports at least 4KB, so bit[8:0] will not be used for address authentication logic.

6.4 Memory access order

The process of accessing the address space by the C910 in different scenarios is briefly summarized below:

Chapter 6:

Memory

Models

Reset	63	38	37	9	8	0
	0		address [37:9] (WARL)	0 (WARL)		
	0		0	0		

Figure 6.10: PMP Address Registers

Scenario 1: No VA-PA Conversion

- CPU to access the PA;
- Get the attributes of the address via sysmap.h;
- PMP check to confirm that the read/write/execute permissions comply with the PMP settings;
- Performs an access to

the address. Scenario 2:

Performing a VA-PA

Conversion

- CPU to access the VA;
- Address translation is performed through the MMU to get the page table entries (pte)
- The following information can be obtained from the pte: PA, address attributes (Note 1), and read/write/execute permissions;
- PMP check to make sure the read/write/execute permissions are in accordance with the PMP settings; (the final read/write/execute permissions will be the "minimum" of the PMP and the pte)
- Performs access to the address.

(Note 1) When maee=1, the address attribute comes from pte; when maee=0, the address attribute comes from sysmap.h.

Chapter 7 Memory Subsystem

7.1 Memory Subsystem Overview

Each core of the C910 has a separate Instruction Cache and Data Cache, and the dual cores share a single L2 Cache. data consistency between cores is maintained by the hardware.

7.2 L1 Command Cache

7.2.1 sum

mar

ize

The main features of the L1 instruction cache are as follows:

- The size of the cache line is 64KB for a 2-way group concatenation;
- Virtual Address Index, Physical Address Tag (VIPT)
- The access data bit width is 128 bits;
- A first-in-first-out replacement strategy is used;
- Supports invalidation of the entire instruction cache and invalidation of a single cache line;
- Supports the command prefetch function;
- Support Road Forecast;
- A request for an instruction cache miss snoops the data cache (switch bit control exists)

7.2.2 road forecast

The C910 instruction cache adopts the structure of two-way group association. In order to reduce the power consumption of accessing the two caches in parallel, the C910MP implements the road prediction function of the instruction cache. When the road prediction information is valid, the access to the invalid data road is turned off and only the data of the predicted road is accessed. The user can enable the road prediction function of the instruction cache by configuring the implicit operation register MHINT.

Depending on the fetching behavior, road forecasts can be classified into the following two categories:

- **Sequential access:** when performing sequential in-row finger picking, the road information for this

access is predicted based on the road hit information from the previous access.

Chapter 7:

Memory

subsystem

- **Jump access:** the branch instruction obtains the road prediction information of the target cache line while obtaining the jump target address, and accesses one of the road caches based on this information.

7.2.3 Cyclic Acceleration Cache

For the large number of short loops in the program, the C910 sets up a 32B loop acceleration buffer. When a short loop instruction sequence is detected, the loop body is loaded into the loop acceleration cache; when a subsequent fetch hits the cache, the instruction and the jump target address are directly fetched from the cache, and the access to the instruction cache and branch history table and branch jump target predictor is turned off, thus reducing the dynamic power consumption of the fetch. The user can enable the short cycle acceleration function by configuring the implicit operation register MHINT.LPE.

7.2.4 Branch History Table

C910 uses a branch history table to predict the jump direction of conditional branches. The branch history table has a capacity of 64Kb and uses the BI-MODE predictor as the prediction mechanism to support one branch result prediction per cycle. The branch history table consists of two parts: predictor and selector. Among them, the predictors are divided into skip predictors and non-skip predictors, and each predictor is maintained in real time according to the branch history information. The branch history table indexes each path by branch history information and the current branch instruction address to obtain the prediction result of the branch instruction jump direction.

Conditional branching instructions for branch history tables to make predictions include:

beq, bne, blt, bltu, bge, bgeu, c.beqz, c.bnez

7.2.5 Branch Jump Goal Predictor

The C910 uses the branch jump target predictor to predict the jump target address of a branch instruction. The branch jump target predictor logs the branch instruction history target address. If the current branch instruction hits the branch jump target predictor, the logged target address is used as the current branch instruction predicted target address.

The main features of the Branch Jump Goal Predictor include:

- 1024 table entries are supported;
- Two way group in-phase structure with replacement based on branch command low PC selection;
- Maintains instruction cache road prediction information;
- Use the current branch instruction section PC for indexing;

Branching instructions for branch jumping to the target predictor for prediction include:

- beq, bne, blt, bltu, bge, bgeu, c.beqz, c.bnez
- JAL, CJ

Chapter 7:

Memory

subsystem

7.2.6 Indirect Branch Predictor

The C910 uses an indirect branch predictor that is responsible for predicting the target address of an indirect branch. Indirect branch instructions obtain their target addresses through registers, and a single indirect branch instruction can contain multiple branch target addresses that cannot be predicted by a traditional branch jump target predictor. Therefore, the C910

Chapter 7:

Memory

subsystem

The branch history-based indirect branch prediction mechanism is used to correlate the historical target address of an indirect branch instruction with the previous branch history information of that branch, and discretize different target addresses of the same indirect branch with different branch history information, so as to realize the prediction of multiple different target addresses.

Indirect branching instructions include:

- JALR: Source registers except X1, X5
- C.JALR: Source registers are X5 excluded
- C.JR: Source registers except X1, X5

7.2.7 Return address predictor

The return address predictor is used for fast and accurate prediction of the return address at the end of a function call. When the finger-taking unit decodes a valid function call instruction, the return address of the function will be stacked into the return address predictor; when the finger-taking unit decodes a valid function return instruction, the stack will be popped from the return address predictor to get the target address of the function return. The return address predictor supports up to 12 levels of function call nesting, exceeding the number of nesting will lead to target address prediction errors.

- Function call instructions include: JAL, JALR, C.JALR
- Function return instructions include:
JALR, C.JR, C.JALR instruction function

can be divided as [Table 7.1](#).

Table 7.1: Specific division of command functions

rd	rs1	rs1=rd	RAS action
!link	!link	-	none
!link	link	-	pop
link	!link	-	push
link	link	0	push and pop
link	link	1	push

7.2.8 Quick Jump Target Predictor

In order to speed up the finger picking efficiency of the finger picking unit during consecutive jumps, the C910 adds a fast jump target predictor to the first level of the finger picking unit. When the finger picker unit makes a consecutive jump, the fast jump target predictor will record the address of the second jump instruction of the consecutive jump and the target address of the jump. If the fast jump target predictor is hit during finger picking, a jump is initiated at the first stage, reducing the performance loss by at least one cycle.

Branching instructions to quickly jump to the target predictor for prediction include:

Chapter 7:

Memory subsystem

- beq, bne, blt, bltu, bge, bgeu, c.beqz, c.bnez
- JAL, C.J
- Function return instruction

7.3 L1 Data Cache

7.3.1 sum

mar
ize

The main features of the L1 data cache are as follows:

- The size of the cache line is 64KB for a 2-way group concatenation;
- Physical Address Index, Physical Address Tag (PIPT);
- The maximum width of each read access is 128 bits and supports byte/half-word/word/double-word/quadword accesses;
- The maximum width of each write access is 256 bits, and any combination of bytes is supported;
- The write policy supports write-back-write allocation mode and write-back-write unallocated mode;
- A first-in-first-out replacement strategy is used;
- Supports invalidate and clear operations for the entire data cache, and invalidate and clear operations for a single cache line;
- Command data prefetching function for multiple channels.

7.3.2 Cache Consistency

For requests with page attributes configured as shareable and cacheable, the hardware maintains data consistency across the L1 data caches of the different cores.

For requests where the page attribute is configured as non-shareable and cacheable, the processor does not maintain data consistency across multiple L1 data caches. If pages requiring this attribute are shared across multiple cores, software is required to maintain data consistency.

The C910MP Level 1 cache uses the MESI protocol to maintain the coherency of the data cache across multiple processor cores. MESI represents the four states of each cache line on the data cache, which are:

- M: indicates that the cache line is located only in this data cache and is written dirty(UniqueDirty)
- E: indicates that the cache line is located only in this data cache and is clean(UniqueClean)
- S: indicates that the cache line may be located in more than one data cache and is clean(ShareClean)
- I: Indicates that the cache line is not in this data cache. (Invalid)

7.3.3 exclusive access

The C910 supports exclusive memory access instructions LR and SC, which can be used to synchronize different processes on the same core or between different cores by forming synchronization primitives such as atomic locks. The LR instruction marks an address for exclusive

Chapter 7: Memory subsystem

access, and the SC instruction determines whether the marked address is preempted by another process. the C910 has a local monitor at the L1 data cache and a global monitor at the L2 cache for each core. Each monitor consists of a state machine and an address buffer, where the state machine contains two states: IDLE and EXCLUSIVE.

Chapter 7:

Memory

subsystem

For the page whose attribute is set as cacheable, exclusive access can be realized through local monitor; LR instruction sets the state machine of local monitor to EXCLUSIVE state during execution and saves the accessed address and size into the cache; SC instruction reads the state, address and size of local monitor during execution; if the state is EXCLUSIVE and the address and size of match exactly, then the write operation is executed and returns to IDLE state; otherwise, if one of the state or address/size does not meet the condition or the data cache is not enabled, then the write operation is executed and clears the state machine to return to IDLE state. If the status is EXCLUSIVE and the address and size of match exactly, then the write operation is executed, returning a successful write and clearing the state machine to return to the IDLE state. Otherwise, if one of the conditions of status or address/size is not satisfied, or if the data cache is not enabled, then the write operation is not executed, returning a failed write and clearing the state machine to return to the IDLE state. Writes from other cores that match the local monitor at the same cacheline address also clear the state machine back to IDLE; writes from this core or exclusive accesses to different addresses do not affect the local monitor. In addition, local monitors need to be cleared during a process switch.

For the page whose attribute is set as not cacheable, both local monitor and global monitor are needed to realize exclusive access; LR not only needs to set local monitor but also needs to set global monitor during execution; SC needs to further check global monitor after local monitor passes the check; only if global monitor passes the check as well, the write operation will be executed, return write success, and clear the state machine; otherwise, the write operation is not executed, return write failure, and clear the state machine. The write operation of other cores will clear the state of a global monitor back to the IDLE state when the address matches that global monitor.

In C910-based systems, it is recommended to use the LR and SC instructions to implement atomic locking operations. If the address attribute of an atomic lock is cacheable (both shared and non-shared), no special design of the SoC system is required. This is the typical case. If the address attribute of the atomic lock is Non-cacheable/Device/Strongly Ordered, the user needs to integrate the exclusive monitor function in the system (e.g. Slave side). Otherwise, the result is UNPREDICTABLE.

7.4 L2 Cache

7.4.1 L2 Cache Summary

The main features of the L2 cache are as follows:

- The size of the cache line is 64B for a 1MB, 16-way group concatenation;
- L2 Cache is strictly Inclusive with L1 D-Cache. L2 cache is non-strictly Inclusive with L1 I-Cache;
- Physical Address Index, Physical Address Tag (PIPT)
- The maximum width per access is 64B;
- Write-back-write-allocate and write-back-write-unallocate write policies are supported;
- A first-in-first-out replacement strategy is used;

Chapter 7:

Memory

subsystem Programmable RAM access delay;

- Supports instruction prefetching and TLB prefetching mechanisms;
- Using chunked assembly line technology.

7.4.2 Cache Consistency

The C910MP L2 cache uses the MOESI protocol to maintain the coherency of the data cache across multiple processor cores. MOESI represents the five states of each cache line on the data cache, which are:

Chapter 7:

Memory

subsystem.

- M: Indicates that the cache line is located only in this data cache and is written dirty; (UniqueDirty)
- O: Indicates that the cache line may be located in more than one data cache and is written dirty; (ShareDirty)
- E: Indicates that the cache line is located only in this data cache and is clean; (UniqueClean)
- S: indicates that the cache line may be located in more than one data cache and that is clean; (ShareClean)
- I: Indicates that the cache line is not in this data cache. (Invalid)

7.4.3 Forms of organization

The C910MP L2 cache is organized in a chunked, pipelined architecture that discretizes access addresses in two different blocks, allowing parallel processing of multiple accesses to improve access efficiency.

The chunking mechanism is shown in Figure 7.1.

- The TAG RAM is divided into two tag sub-blocks, Tag bank0 and Tag bank1, according to address PA[6] to support parallel processing of 2 access requests in the same clock cycle.
- Similarly, the DATA RAM is also divided into two data sub-blocks according to the address PA[6], Data bank0 and Data bank1; corresponding to each data sub-block, it is further divided into four micro-blocks, each of which has a data width of 128 bits, so as to realize the purpose of fetching a cache line in parallel.

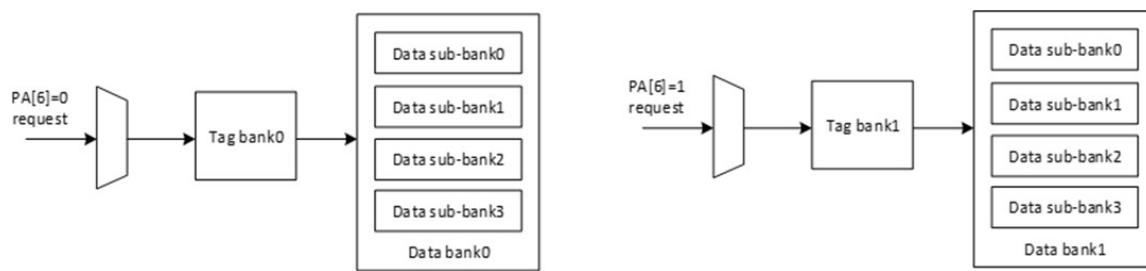


Figure 7.1: L2 Cache Organization

7.4.4 RAM Delay

Due to the large size of the L2 Cache, the access latency is long and usually requires multiple clock cycles to complete the access. the C910MP provides configurable access latency, which can be manually set according to the setup time and latency of the RAM used in different processes. The configuration details are shown in Table 7.2.

Chapter 7: Memory subsystem

Table 7.2: RAM Access Latency Configuration

Configuration options	functionality	clarification
L2 TAG setup	L2 Cache Tag RAM setup: 1b0 0 cycle. default 1b1 1 cycle.	The configuration of the L2 Cache Tag RAM only affects TAG RAM accesses
L2 TAG latency	L2 Cache Tag RAM latency: 3b000 1 cycle. default 3b001 2 cycle. 3b010 3 cycle. 3b011 4 cycle. 3b1xx 5 cycle.	
L2 DATA setup	L2 Cache Data RAM setup. 1b0 0 cycle. default 1b1 1 cycle.	The L2 Cache Data RAM configuration only affects the DATA RAM. visits
L2 DATA latency	L2 Data RAM latency: 3b000 1 cycle. default 3b001 2 cycles. 3b010 3 cycle. 3b011 4 cycle. 3b100 5 cycle. 3b101 6 cycle. 3b110 7 cycle. 3b111 8 cycle.	

Users configure the latency according to the access time of the used RAM; setup defaults to 0, and you can choose to configure setup to 1 when the setup time of the used RAM is longer, or when the winding is longer.

After configuring the above options, the resulting number of access cycles is shown in Table 7.3.

Table 7.3: TAG RAM Effective Access Latency

TAG latency	TAG RAM Effective Access Latency	
	TAG setup = 0	TAG setup = 1
/		
000	1	2
001	2	3
010	3	4
011	4	5
1xx	5	5

Table 7.4: DATA RAM Effective Access Latency

TAG latency	DATA RAM effective access delay	
/	TAG setup = 0	TAG setup = 1
000	1	2
001	2	3
010	3	4
011	4	5
100	5	6
101	6	7
110	7	8
111	8	8

- L2 Tag latency The maximum effective latency is 5 cycles;
- TAG setup set to 1 adds a 1-cycle access, flopping the signal at the SRAM input before accessing the SRAM;
- L2 Data latency The maximum effective latency is 8 cycles;
- When DATA setup is set to 1, a 1-cycle access is added, and the SRAM input signals are flopped before accessing the SRAM.

7.5 Memory Accelerated Access

This subsection concentrates on characterizing the C910 L1/L2 cache in terms of memory-accelerated access.

7.5.1 L1 I-Cache Instruction Prefetch

The L1 instruction cache supports instruction prefetching, which is realized by configuring the implicit operation register MHINT.IPLD. When the current cache line access is missing, prefetching of the next consecutive cache line is enabled, and the result of prefetching is cached in the prefetch buffer. When an instruction access hits the prefetch buffer, the instruction is fetched directly from the buffer and backfilled into the instruction cache, thus reducing the fetch latency.

Instruction prefetching requires that the prefetched cache line be located on the same page as the currently accessed cache line, thus securing the fetch finger address. In addition, read-sensitive peripheral address space is prohibited from being allocated to instruction area space.

7.5.2 L1 D-Cache Multi-Channel Data Prefetching

In order to minimize memory access latency for large memory such as DDR, the C910 supports data prefetching. By detecting data cache misses, it matches a fixed access pattern, and then the

Chapter 7:

Memory subsystem hardware automatically prefetches the cache line and backfills the L1 data cache.

The C910 supports up to 8 channels of data prefetching and implements 2 different prefetching methods: sequential prefetching and interval prefetching (stride \leq 32 cache lines).

In addition, forward prefetching and reverse prefetching (i.e., stride is negative) are implemented to support various possible access modes. The data prefetching function is stopped when the processor performs data cache invalidation and clear operations.

Chapter 7:

Memory

subsystem

Users can enable the data prefetching function by setting the implicit register MHINT.DPLD; and determine the number of cache lines to be prefetched at one time by setting MHINT.DPLD_DIS.

The commands that support data prefetching are as follows:

- lb, lbu, lh, lhu, lw, lwu, ld
- FLW, FLD
- lrblrhrlrwrlrbu[rhulrwu]lrlhjurwjlurd[lrbu[rhu, lurwu, lbi, lhi, lwi, ldi, lbi, lhui-lhui-lwui-lwui-ldd, lwd, lwud

7.5.3 L1 Adaptive write distribution mechanism

The L1 of the C910 implements an adaptive write-allocation mechanism. When the processor detects successive memory write operations, the write-allocation attribute of the page is automatically turned off.

The user can enable L1 adaptive write allocation by setting the implicit register MHINT.AMR.

The processor's adaptive write-allocation mechanism is turned off when a data-high-slow invalidation or clear operation is performed; the memory sequential write behavior is re-detected after the cache operation is complete.

The commands that support adaptive write allocation are listed below:

- SB, SH, SW, SD
- FSW, FSD
- srb, srh, srw, srd, surb, surh, surw, surd, sbi, shi, swi, sdi, sdd, swd

7.5.4 L2 Prefetching mechanism

The L2 cache has a prefetch feature that supports prefetching of fetch instructions and TLB accesses.

The supported features are as follows:

- The number of instruction prefetches (0, 1, 2, 3) that can be assigned by the software, all prefetches backfill the L2 cache;
- The TLB prefix is fixed to 1;
- The prefetching mechanism uses the 4KB page table as the boundary, and will stop prefetching if the address crosses the 4KB boundary during prefetching;
- The prefetch mechanism can be configured via MCCR2.

7.6 Instructions and Registers Related to L1/L2 Cache Operations

After a processor reset, the instruction and data caches are automatically invalidated, and

Chapter 7:

Memory

subsystem turns off the instruction and data caches by default. Similarly, after a processor reset, the L2 cache will be invalidated automatically, and L2 will be turned on automatically after the operation,且 cannot be turned off.

Closed. It is important to note that when the L1 cache is closed, L2 does not initiate a backfill operation on a miss.

7.6.1 L1 Cache Expansion Register

The C910 L1 cache related expansion registers are mainly categorized by function:

- Cache Enable and Mode Configuration: The Machine Mode Hardware Configuration Register (mhcr) enables the switching of the instruction and data caches, as well as the configuration of write allocate and write return modes. The Superuser Mode Hardware Configuration Register (shcr) is a map of mhcr and is a read-only register.
- Dirty Table Entry Clearing and Invalidation Operations: The Machine Mode Cache Operations Register (mcor) allows dirty table entry and invalidation operations to be performed on the instruction and data caches.
- Cache read operation: machine mode cache access instruction register (mcins) cache access index register (mcindex) and cache access data register 0/1 (mcdat0/1)

Specific control register descriptions can be found in the [Machine Mode Processor Control and Status Extended Register Set](#) and the [Machine Mode Cache Access Extended Register Set](#).

7.6.2 L2 Cache Extension Register

The C910 L2 Cache Related Extension Registers are mainly categorized by function:

- L2 Cache Enable and Latency Configuration: The Machine Mode L2 Cache Enable Register (mccr2) enables the setting of the L2 cache access latency.
- L2 cache read operation: machine mode cache access instruction register (mcins) cache access index register (mcindex) and cache access data register 0/1 (mcdat0/1)

For definitions and descriptions of the related control registers, refer to the [Machine Mode Processor Control and Status Expansion Register Set](#) and [Machine Mode Cache](#) accesses the extended register set.

7.6.3 L1/L2 Cache Operation Instructions

The C910 extends the L1/L2 cache operation instructions, including Invalidate by Address, Invalidate All, Clear Dirty Table Entries by Address, Clear All Dirty Table Entries, Clear Dirty Table Entries by Address and Invalidate, and Clear All Dirty Table Entries and Invalidate, as shown in [Table 7.5](#).

Table 7.5: L1/L2 Cache Operation Instructions

ICACHE.IALL	ICACHE Invalid all table entries
ICACHE.IALLS	ICACHE Broadcast Invalid All Table Entries
ICACHE.IPA	ICACHE Invalid Table Entry by Physical Address
ICACHE.IVA	ICACHE Invalid Table Entry by Virtual Address
DCACHE.CALL	DCACHE Clear all dirty table entries
DCACHE.CIALL	DCACHE Clear all dirty table entries and invalidate them.
DCACHE.CIPA	DCACHE clears dirty table entries by physical address and invalidates the
DCACHE.CISW	DCACHE clears dirty table entries by set/way and invalidates them.
DCACHE.CIVA	DCACHE clears and invalidates dirty table entries by virtual address
DCACHE.CPA	DCACHE Clear dirty table entries by physical address
DCACHE.CPAL1	L1 DCACHE Clear dirty table entries by physical address
DCACHE.	DCACHE Clear dirty table entries by virtual address
DCACHE.CSW	DCACHE clears dirty table entries by set/way
DCACHE.CVAL1	L1 DCACHE Clear dirty table entries by virtual address
DCACHE.IPA	DCACHE Invalid by physical address
DCACHE.ISW	DCACHE press set/way is invalid
DCACHE.IVA	DCACHE Invalid by virtual address
DCACHE.IALL	DCACHE Invalid all table entries
L2CACHE.CALL	L2CACHE Clear all dirty table entries
L2CACHE.CIALL	L2CACHE Clear and invalidate all dirty table entries.
L2CACHE.IALL	L2CACHE All table entries are invalid

Refer to Appendix *B-1 Cache Instruction Terminology* for instruction specific descriptions.

Chapter 8 Interrupt Controller

8.1 CLINT Interrupt Controller

The C910 implements the Processor Core Local Interrupt Controller (hereafter referred to as CLINT), a memory-address-mapped module that handles software interrupts and timer interrupts.

8.1.1 CLINT Register Address Mapping

The CLINT interrupt controller occupies 64KB of memory space. Its high 13-bit address is determined by SoC hardware integration, and its low 27-bit address mapping is shown [in Table 8.1](#). All registers support only word-aligned access.

Table 8.1: CLINT Register Memory Map Addresses

address	name (of a thing)	typology	starting value	descriptive
0x4000000	MSIP0	Read/Write	0x00000000	Nucleus 0 Machine mode software interrupt High bound 0, bit[0] is valid
0x4000004	MSIP1	Read/Write	0x00000000	Nucleus 1 Machine mode software interrupt Configuration register high bit tied 0, bit[0] valid
Reserved	-	-	-	-
0x4004000	MTIMECMPL0	Read/Write	0xFFFFFFFF	Core 0 Machine Mode Clock Timer Compare Value Register (Lower 32 bits)
0x4004004	MTIMECMPH0	Read/Write	0xFFFFFFFF	Core 0 Machine Mode Clock Timer Comparison Value Register (High 32 bits)
0x4004008	MTIMECMPL1	Read/Write	0xFFFFFFFF	Nucleus 1 Machine mode clock timer Compare Value Register (Lower 32 bits)
0x400400C	MTIMECMPH1	Read/Write	0xFFFFFFFF	Nucleus 1 Machine mode clock timer Comparison Value Register (High 32 bits)
Reserved	-	-	-	-
0x400C000	SSIP0	Read/Write	0x00000000	Nucleus 0 Superuser mode software interrupt Configuration register high bit tied 0, bit[0] valid
0x400C004	SSIP1	Read/Write	0x00000000	Kernel 1 Superuser Mode Software Interrupt Configuration Registers High bound 0, bit[0] is valid
Reserved	-	-	-	-
0x400D000	STIMECMPL0	Read/Write	0xFFFFFFFF	Kernel 0 Super User Mode Clock Timer Comparison Value Register (Low) 32-bit
0x400D004	STIMECMPH0	Read/Write	0xFFFFFFFF	Kernel 0 Super User Mode Clock Timer Comparison Value Register (High) 32-bit
0x400D008	STIMECMPL1	Read/Write	0xFFFFFFFF	Kernel 1 Super User Mode Clock Timer Comparison Value Register (low) 32-bit
0x400D00C	STIMECMPH1	Read/Write	0xFFFFFFFF	Kernel 1 Super User Mode Clock Timer Comparison Value Register (High) 32-bit
Reserved	-	-	-	-

8.1.2 software interruption

Chapter 8

Interrupt

CLINT can be used to generate software interrupts.

Controller

Software interrupts are controlled by configuring address-mapped software interrupt configuration registers. The machine mode software interrupts are controlled by the Machine Mode Software Interrupt Configuration Register (MSIP) and the Super User Mode software interrupts are controlled by the Super User Mode Software Interrupt Configuration Register (SSIP).

The user can generate a software interrupt by setting the xSIP bit to 1, and clear the software interrupt by clearing the xSIP bit to 0. The user can also generate a software interrupt by setting the xSIP bit to 0. The CLINT super user mode software interrupt request is only valid when the CLINTEE bit is enabled on the corresponding core.

Machine mode has the privilege of modifying and accessing all registers related to software interruptions; superuser mode only has the privilege of accessing and modifying the superuser mode

Chapter 8

Interrupt

Controller

Style Software Interrupt Configuration Register (SSIP) privileges; normal user mode has no privileges.

The structure of the two groups of registers is the same, and their register bit distribution and bit definitions are shown in Figure 8.1.

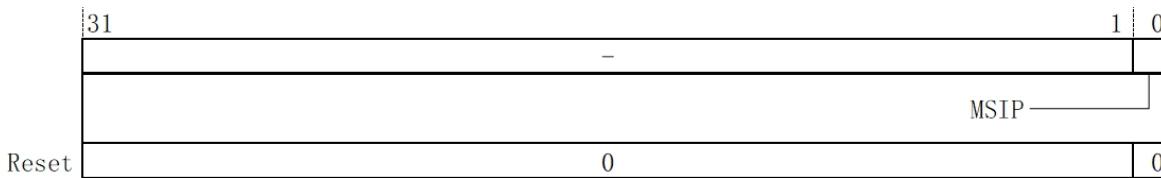


Figure 8.1: Machine Mode Software Interrupt Configuration Register (MSIP)

- **MSIP: Machine Mode Software Interrupt Wait Bit**

This bit indicates the interrupt status of the machine mode software interrupt.

- When MSIP position 1, there is currently a valid machine mode software interrupt request.
- When MSIP position 0, there is currently no valid machine mode software interrupt request.

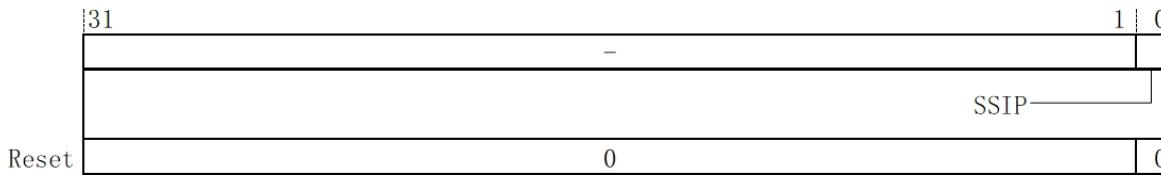


Figure 8.2: Super User Mode Software Interrupt Configuration Register (SSIP)

- **SSIP: Super User Mode Software Interrupt Wait Bit**

This bit indicates the interrupt status of the super user mode software interrupt.

- When SSIP position 1, there is currently a valid superuser software interrupt request.
- When SSIP position 0, there are currently no valid superuser software interrupt requests.

8.1.3 timer interrupt

CLINT can be used to generate timer interrupts.

Only one 64-bit system timer, MTIME, exists in multicore systems and is required to operate in the always-on voltage domain. The system timer is not writable and can only be cleared by reset 0. The current value of the system timer can be obtained by reading the TIME register of the PMU. The main purpose of the system timer is to provide a uniform time reference for multiple cores.

Each core has a set of 64-bit Machine Mode Clock Timer Comparison Value Registers (*MTIMECMPL*, *MTIMECMPH*) and a set of 64-bit Super User Mode Clock Timer Comparison Value Registers (*STIMECMPL*, *STIMECMPH*). Each of these registers can be modified by address word-

Chapter 8

Interrupt Controller
aligned access to either its high 32 bits or low 32 bits, respectively.

Chapter 8

Interrupt

Controller

CLINT determines whether to generate a timer interrupt by comparing the values of {CMPH[31:0],CMPL[31:0]} with the current value of the system timer. When {CMPH[31:0],CMPL[31:0]} is greater than the system timer value, no interrupt is generated; when {CMPH[31:0],CMPL[31:0]} is less than or equal to the system timer value, CLINT generates the corresponding timer interrupt. The software can clear the corresponding timer interrupt by rewriting the value of MTIMECMP/STIMECMP. The super user mode timer interrupt request is valid only when the corresponding core enables the CLINTEE bit.

Machine mode has modification access to all timer interrupt related registers; Super User mode has access to modify only the Super User mode clock timer compare value registers (STIMECMPL, STIMECMPH); Normal User mode has no access.

Each group of registers has the same structure, and its register bit distribution and bit definition are shown in Figure 8.3.

	63	32	31	0
Reset	MTIMECMPH		MTIMECMPL	
	0xffffffff		0xffffffff	

Figure 8.3: Machine Mode Timer Interrupt Compare Value Register (High/Low)

- **MTIMECMPH/MTIMECMPL: Machine Mode Timer Interrupt Comparison Value Register High/Low**

This register stores the timer comparison value

- MTIMECMPH: Timer comparison value is 32 bits high;
- MTIMECMPL: Timer comparison value is 32 bits lower;

	63	32	31	0
Reset	STIMECMPH		STIMECMPL	
	0xffffffff		0xffffffff	

Figure 8.4: Super User Mode Timer Interrupt Compare Value Register (High/Low)

- **STIMECMPH/STIMECMPL: Super User Mode Timer Interrupt Comparison Value Register High/Low**

This register stores the timer comparison value

- STIMECMPH: Timer comparison value is 32 bits high;
- STIMECMPL: Timer comparison value is 32 bits lower;

8.2 PLIC Interrupt Controller

Platform Level Interrupt Controller (hereafter PLIC) for sampling, priority arbitration and distribution of external interrupt sources. Both machine mode and

Chapter 8

Interrupt

super user mode of each core can be used as valid interrupt targets in the PLIC Controller model.

The basic functions of the PLIC unit implemented in the C910 are as follows:

- Supports interrupt distribution for 2 cores/4 interrupt targets;
- 144 interrupt sources sampling, support level interrupt, pulse interrupt;

Chapter 8

Interrupt

Controller

- 32 levels of interrupt priority;
- Interrupt enable is maintained independently for each interrupt target;
- Interrupt thresholds are maintained independently for each interrupt target;
- PLIC register access is configurable.

8.2.1 Interrupted arbitration

In PLIC, only interrupt sources that meet the conditions will participate in arbitration for a given interrupt target. The conditions that are met are listed below:

- Interrupt source in wait state ($IP = 1$)
- Interrupt priority is greater than 0.
- Enable bit is turned on for this interrupt target.

When there are multiple interrupts in the PLIC in the Pending state for a certain interrupt target, the PLIC arbitrates the interrupt with the highest priority. In the PLIC implementation of the C910, the machine mode interrupt always has a higher priority than the super user mode interrupt. In the C910 PLIC implementation, the machine mode interrupt priority is always higher than the super user mode interrupt. When the modes are the same, the higher the value of the priority configuration register, the higher the priority, and the interrupt with priority 0 is invalidated; if multiple interrupts have the same priority, the one with the smaller ID will be processed first.

The PLIC updates the arbitration result in the form of an interrupt ID into the interrupt response/completion register of the corresponding interrupt target.

8.2.2 Interrupt request and response

The PLIC initiates an interrupt request to a specific interrupt target when a valid interrupt request exists for that interrupt target,且 with a priority greater than the interrupt threshold for that interrupt target. When the interrupt target receives an interrupt request,且 can respond to the interrupt request by sending an interrupt response message to the PLIC.

The interrupt response mechanism is as follows:

- The interrupt target initiates a read operation to its corresponding interrupt response/completion register. The read operation returns an ID indicating the current PLIC arbitrated interrupt ID, and the interrupt target proceeds to the next step based on the ID obtained. If the obtained interrupt ID is 0, there is no valid interrupt request and the interrupt target terminates interrupt processing.
- When the PLIC receives a read operation initiated by the interrupt target and returns the corresponding ID, it will clear the IP bit of the interrupt source corresponding to the ID to 0, and will block the subsequent samples of the interrupt source before the interrupt is completed.

Chapter 8

Interrupt

8.2.3 Completion of interruptions

When the interrupt target finishes interrupt processing, it needs to send an interrupt completion message to the PLIC. The interrupt completion mechanism is as follows:

- The interrupt target initiates a write operation to the interrupt response/completion register, and the value of the write operation is the interrupt ID of the current completion. if the interrupt type is a level interrupt, the external interrupt source needs to be cleared before initiating the above write operation.
- After the PLIC receives this interrupt completion request, it does not update the interrupt response/completion register, unmasks the interrupt source sampling corresponding to the ID, and ends the entire interrupt processing.

8.2.4 PLIC register address mapping

The PLIC interrupt controller occupies 64MB of memory space. Its high 13-bit address is determined by the SoC hardware integration, and its low 27-bit address is mapped as shown in the table below 8.2 shown. All registers support only address word-aligned access. (PLIC registers are accessed via a word access instruction (Load *word* instruction), and the result of the access is placed in the lower 32 bits of the 64-bit *GPR*)

Note: Registers that are not supported by the open source version of the C910 have the attribute "reserved".

Table 8.2: PLIC Address Mapping

address	name (of a thing)	typology	starting value	descriptive
0x00000000	-	-	-	-
0x00000004	PLIC_PRIO1	R/W	0x0	Interrupt
0x00000008	PLIC_PRIO2	R/W	0X0	Source 1 ~ 1023
0x0000000C	PLIC_PRIO3	R/W	0x0	Priority
...	Configuration
0x0000FFC	PLIC_PRIO1023	R/W	0x0	Registers
0x0001000	PLIC_IP0	R/W	0x0	Interrupts 1 to 31
				Interrupt Waiting Register
0x0001004	PLIC_IP1	R/W	0x0	Interrupt 32 to 63
				Interrupt Waiting Register
...
0x000107C	PLIC_IP31	R/W	0x0	Interruptions 992 to 1023
				Interrupt Waiting Register
Reserved	-	-	-	-
0x0002000	PLIC_H0_MIE0	R/W	0x0	Nuclei 0 1 to 31 Machine Mode Interrupt Enable Register
0x0002004	PLIC_H0_MIE1	R/W	0x0	Nucleus 0 32 to 63 Machine Mode Interrupt Enable Register
...
0x000207C	PLIC_H0_MIE31	R/W	0x0	Nuclei 0 992 to 1023 Machine Mode Interrupt Enable Register
0x0002080	PLIC_H0_SIE0	R/W	0x0	Nucleus 0 1 to 31 Super User Mode Interrupt Enable Register

Chapter 8

Interrupt 0x0002084 Controller	PLIC_H0_SIE1	R/W	0x0	Nucleus 0 32 to 63 Super User Mode Interrupt Enable Register
...
0x00020FC	PLIC_H0_SIE31	R/W	0x0	Nuclei 0 992 to 1023 Super User Mode Interrupt Enable Register

Continued on next page

Chapter 8 Interrupt Controller

Table 8.2 - continued from previous page

address	name (of a thing)	typology	starting value	descriptive
0x0002100	PLIC_H1_MIE0	R/W	0x0	Nucleus 1 1 to 31 Machine Mode Interrupt Enable Register
0x0002104	PLIC_H1_MIE1	R/W	0x0	Nucleus 1 32 to 63 Machine Mode Interrupt Enable Register
...
0x000217C	PLIC_H1_MIE31	R/W	0x0	Nuclei 1,992 to 1023 Machine Mode Interrupt Enable Register
0x0002180	PLIC_H1_SIE0	R/W	0x0	Nucleus 1 1 to 31 Super User Mode Interrupt Enable Register
0x0002184	PLIC_H1_SIE1	R/W	0x0	Nucleus 1 32 to 63 Super User Mode Interrupt Enable Register
...
0x00021FC	PLIC_H1_SIE31	R/W	0x0	Nuclei 1,992 to 1023 Super User Mode Interrupt Enable Register
Reserved	-	-	-	-
0x01FFFFC	PLIC_PER	R/W	0x0	PLIC Privilege Control Register
0x0200000	PLIC_H0_MTH	R/W	0x0	Nucleus 0 Machine mode interrupt Threshold Register
0x0200004	PLIC_H0_MCLAIM	R/W	0x0	Nucleus 0 Machine mode interrupt Response/Completion Register
Reserved	-	-	-	-
0x0201000	PLIC_H0_STH	R/W	0x0	Kernel 0 Superuser mode interrupt Threshold Register
0x0201004	PLIC_H0_SCLAIM	R/W	0x0	Kernel 0 Superuser mode interrupt Response/Completion Register
Reserved	-	-	-	-
0x0202000	PLIC_H1_MTH	R/W	0x0	Nucleus 1 Machine mode interrupt Threshold Register
0x0202004	PLIC_H1_MCLAIM	R/W	0x0	Nucleus 1 Machine mode interrupt Response/Completion Register
Reserved	-	-	-	-
0x0203000	PLIC_H1_STH	R/W	0x0	Nucleus 1 Superuser mode interrupt Threshold Register
www.t-head.cn			78	
0x0203004	PLIC_H1_SCLAIM	R/W	0x0	Nucleus 1 Superuser mode interrupt Response/Completion Register

Chapter 8

Interrupt

Controller

8.2.5 Interrupt Priority Configuration Register (PLIC_PRIO)

This register sets the priority of the interrupt source. The register read and write privileges are described with reference to the Privilege Control Register (PLIC_PER). The register bit distribution and bit definitions are shown in Figure 8.5.

	31		5 4	0
	—		Prio	
Reset	0		0	

Figure 8.5: Interrupt Priority Configuration Register (PLIC_PRIO)

- **PRI0: Priority of Interrupt**

The lower 5 bits of the Priority Configuration Register are writable and support 32 different levels of priority. A priority of 0 means the interrupt is invalid.

Machine mode interrupts have unconditionally higher priority than super user mode interrupts. When the modes are the same, priority 1 is the lowest priority and priority

31 is the highest. When the priority is the same, the interrupt source IDs are further compared and the one with the smaller ID has the higher priority.

8.2.6 Interrupt wait register (PLIC_IP)

The wait status of each interrupt source can be obtained by reading the information in the interrupt wait register. For interrupts with interrupt ID N, its interrupt information is stored on IP y ($y = N \bmod 32$) in the PLIC_IP x ($x = N/32$) register. where bit 0 of the PLIC_IP0 register is fixed tied to 0. The register read and write permissions are described in the Permission Control Register (PLIC_PER). The register bit distribution and bit definitions are shown in Figure 8.6.

	31	30	29		2	1	0
Reset					IP ID (32x+30) IP ID (32x+31)	IP ID (32x+1) IP ID (32x)		

Figure 8.6: PLIC_IP x Interrupt Wait Register (PLIC_IP)

- **IP: Interrupt Waiting State**

This bit indicates the interrupt wait status of the corresponding interrupt source.

- When the IP bit is 1, it indicates that there is currently an interrupt waiting to be responded to by this external interrupt source. This bit can be set to 1 by a memory store instruction, and will also be set to 1 after the corresponding interrupt source

Chapter 8

Interrupt Controller Sampling logic samples a valid level or pulse interrupt.

- An IP bit of 0 indicates that there are currently no interrupts waiting for a response from this external interrupt source. This bit can be cleared by a memory store instruction, and the PLIC will clear the corresponding IP bit when the interrupt is responded to.

8.2.7 Interrupt Enable Register (PLIC_IE)

Each interrupt target has an interrupt enable bit for each interrupt source that can be used to enable the corresponding interrupt. The machine mode interrupt enable register is used to enable the machine mode external interrupt and the super user mode interrupt enable register is used to enable the super user mode external interrupt.

For an interrupt ID of N, its interrupt enable information is stored on IE_y ($y = N \bmod 32$) in the PLIC_IE_x ($x = N/32$) register. The IE bit corresponding to ID0 is fixedly tied to 0. The register read/write privileges are described in the privilege control register (PLIC_PER).

The register bit distribution and bit definitions are shown in Figure 8.7.

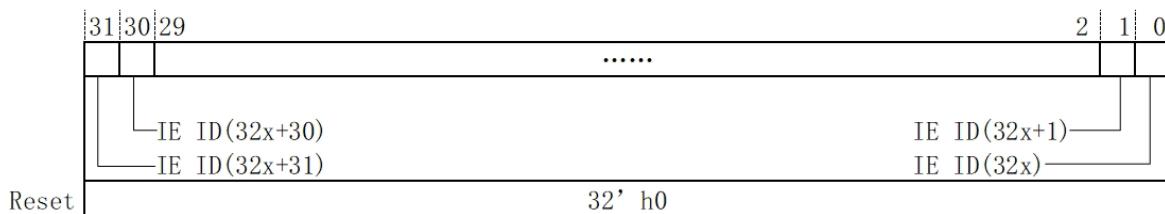


Figure 8.7: PLIC_IE_x Interrupt Enable Register (PLIC_IE)

- **IE Interrupt Enable:**

This bit indicates the interrupt enable state of the corresponding interrupt source.

- When the IE bit is 1, it indicates that the interrupt is enabled for this target.
- When the IE bit is 0, it indicates that the interrupt is masked to this target.

8.2.8 PLIC Privilege Control Register (PLIC_PER)

The PLIC Privilege Control Register is used to control Super User mode access to some of the PLIC registers.

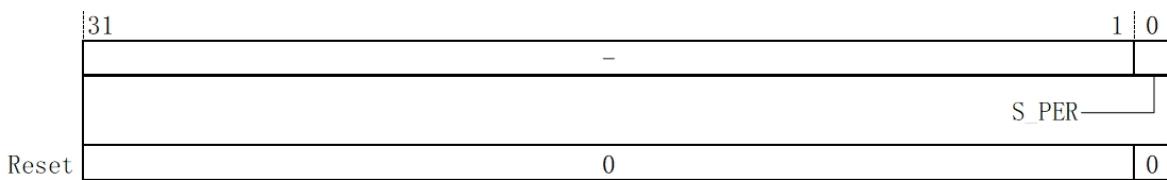


Figure 8.8: PLIC Privilege Control Register (PLIC_PER)

- **S_PER The access rights control bit:**

When S_PER is 0, only machine mode has access to all PLIC registers. Super User Mode does not have access to the PLIC Privilege Control Register, Interrupt Priority Configuration Register, Interrupt Wait Register, and Interrupt Enable Register, and only has access to the Super User Mode Interrupt Threshold Register and Super User Mode Interrupt

Chapter 8

Interrupt

Response/Completion Register. Normal user mode does not have access to any PLIC registers.

With S_PER at 1, machine mode has all privileges. Superuser mode has all privileges except for the PLIC privilege control register.

PLIC register permissions. Normal user mode does not have any PLIC register access.

8.2.9 Interrupt Threshold Register (PLIC_TH)

Each interrupt target has a corresponding interrupt threshold register. Only valid interrupts with a priority greater than the interrupt threshold will initiate an interrupt request to the interrupt target. The register read and write privileges are described with reference to the Privilege Control Register (PLIC_PER).

The register bit distribution and bit definitions are shown in Figure 8.9.

	31	–	5 4	0
Reset		0	PRIOTHRESHOLD	0

Figure 8.9: Interrupt Threshold Register (PLIC_TH)

- **PRIOTHRESHOLD Priority threshold:**

Indicates the interrupt threshold for the current interrupt target. The threshold is configured to 0, indicating that all interrupts are allowed.

8.2.10 Interrupt response/completion register (PLIC_CLAIM)

Each interrupt target has a corresponding interrupt response/completion register. This register is updated when the PLIC completes arbitration, and the updated value is PLIC The interrupt ID of the result of this arbitration. register read and write permissions are described with reference to the Permission Control Register (PLIC_PER).

The register bit distribution and bit definitions are shown in Figure 8.10.

	31	–	10 9	0
Reset		0	CLAIM_ID	0

Figure 8.10: Interrupt Response/Completion Register (PLIC_CLAIM)

- **CLAIM_ID Interrupt request ID:**

Read operation on this register: returns the ID value currently stored in the register. This read operation indicates that the interrupt corresponding to the ID has begun processing. the PLIC begins interrupt response processing.

Write operation to this register: Indicates that the interrupt with the ID corresponding to the write value has completed processing. this write operation does not update the interrupt response/completion register. the PLIC starts interrupt completion processing.

8.3 multicore interruptions

Chapter 8

Interrupt

Two common multi-core interrupt application scenarios are briefly described below.

Controller

8.3.1 Multiple cores handle external interrupts simultaneously

Under the PLIC model, it is permissible to map a single interrupt source to multiple cores at the same time. When the interrupt source generates an interrupt request, it is in the Pending state with respect to multiple cores at the same time. Due to the different states of the cores, each core responds to the interrupt and reads the CLAIM register successively to obtain the interrupt ID. PLIC's design ensures that only the first core that reads the CLAIM register obtains the real ID, while the other cores get an invalid ID (i.e., ID=0) and are not processed. Therefore, the interrupt will only be processed once.

Mapping an interrupt to multiple cores at the same time reduces the overall interrupt response time (any of the multiple cores may handle the interrupt) while taking up a fraction more of the processor's resources (the core that gets the invalid ID consumes bandwidth for nothing)

8.3.2 Inter-core send software interrupt

In the programming model of CLINT, software interrupts have specialized registers, which are:

- M-state software interrupt: MSIP0 MSIP1
- S-Statement software interrupt: SSIP0 SSIP1

The addresses of the above registers are uniform for all cores and are visible, so each core can send software interrupts to any core (including itself) by simply performing write operations to the above registers.

Chapter 9 Bus Interfaces

9.1 AXI Master Device Interface

The C910MP's master device interface supports the AMBA 4.0 AXI protocol. Refer to the *AMBA Specification-AMBA® AXI™ and ACE™ Protocol Specification*.

9.1.1 Features of the AXI Master Device Interface

The AXI Master Device Interface is responsible for address control and data transfer between the C910 and the AXI system bus, and its basic features include:

- Supports AMBA 4.0 AXI bus protocol;
- Supports 128-bit bus width;
- Supports different frequency ratios of the system clock to the CPU master clock;
- All output signals are flopped out and input signals are flopped for better timing.

9.1.2 Outstanding capability of the master device interface

This subsection lists the Outstanding capabilities of the C910 AXI Master Device Interface as follows:

Table 9.1: AXI Master Device Interface Outstanding Capabilities

parameters	numerical value	clarification
Read Issuing Capability	$8n+28$ n= number of cores	Issue up to 8 Non-cacheable and Device reads per core. Requests. There is a global maximum of 28 Cacheable read requests.
Write Issuing Capability	$8n+32$ n= number of cores	Each core issues up to 8 Non-cacheable and Device writes. Requests. Up to 32 Cacheable write requests globally.

Table 9.2: AXI Master Device Interface ARID Codes

ARID[7:0]	Applicable Scenarios	Outstanding for each ID
{2'b10, 6'b??????}	Cacheable Read Request	No outstanding per ID, all cacheable read requests outstanding. The total number of standing is 28.
{1'b0, 2'b(coreid), 1'h18}	Non-cacheable weak-ordered Read request	All Non-cacheable read requests outstanding 31 in total.
{1'b0, 2'b(coreid), 1'h1d}	Non-cacheable strong-ordered Read request	

Table 9.3: AXI Master Device Interface AWID Codes

AWID[7:0]	Applicable Scenarios	Outstanding for each ID
{3'b111, 5'b?????}	Cacheable Write Request	No outstanding per ID, all cacheable write requests outstanding. The total number of standing is 32.
{4'b0000, 4'b????}	Non-cacheable weak-ordered Write request	Each ID has no outstanding, all Non-cacheable weak- The total number of ordered write requests outstanding is 16.
{1'b0, 2'b(coreid), 1'h1d}	Non-cacheable strong-ordered Write request	Non-cacheable strong-ordered write request outstanding The number is 31.

Note: The coding of the above ARID/AWID may change as the processor version evolves; therefore, SoC integration should not rely on specific ID values, but should follow the generalized rules of the AXI protocol.

9.1.3 Supported transmission types

The transmission characteristics supported by the master device interface are as follows:

- BURST supports INCR and WRAP transmissions; no other burst types are supported;
- LEN supports transmission lengths of 1 or 4; other transmission lengths are not supported;
- Exclusive access is supported;
- SIZE Quadword, doubleword, word, halfword, and byte transfers are supported; other transfer sizes are not;
- Read and write operations are supported.

Note: The C910's master device interface implements only a subset of all AXI transports. However, SoC integration should not be dependent on a specific transport type, but should follow the generalized rules of the AXI protocol.

9.1.4 Supported response types

The master device interface receives a response from the slave device of type:

- OKAY

- EXOKAY
- SLVERR
- DECERR

9.1.5 Behavior with different bus responses

The CPU behavior when different bus responses occur on the bus is shown in [Table 9.4](#).

Table 9.4: Bus Exception Handling

RRESP/BRESP	in the end
OKAY	Normal transmission access succeeds, or exclusive transmission access fails; read transmission exclusive access failure means that the bus does not support exclusive transmission and generates an access error exception, write transmission exclusive An access failure is only a lock grab failure, no exception is returned.
EXOKAY	exclusive Access was successful;
SLVERR/DECERR	Access error, read transfers generate an access error exception, write transfers ignore this error;

9.1.6 AXI Master Device Interface Signal

The interface signals of the AXI4.0 master device are all the signals in [Table 9.5](#).

Table 9.5: AXI Protocol Channel Interface Signals

signal name	I/O	Reset	define
Read Address Channel Related Interface			
biu_pad_arid[7:0]	O	0	Read Request Address ID
biu_pad_araddr[39:0]	O	0	read request address
biu_pad_arlen[1:0]	O	0	Read Request burst length 00: 1 transfer 11: 4 transfers
biu_pad_arsize[2:0]	O	0	Read request per beat data bit width 000: 1 byte 001: 2 bytes 010: 4 bytes 011: 8 bytes 100: 16 bytes
biu_pad_arburst[1:0]	O	0	The transport type corresponding to the read request:

Chapter 9 Bus

Interfaces

			01: INCR 10: WRAP
--	--	--	----------------------

Continued on next page

Table 9.5 - continued from previous page

signal name	I/O	Reset	define
biu_pad_arlock	O	0	The access method corresponding to the read request: 0: normal access 1: exclusive access
biu_pad_arcache[3:0]	O	0	The type of memory access corresponding to the read request: 0000: Device Non-bufferable (strong order) 0001: Device Bufferable (strong order) 0011: Normal Non-cacheable Bufferable (weak order) 1111: Cacheable
biu_pad_arprot[2:0]	O	0	Read the type of protection requested: 0 1 [2]: Data Instruction [1]: Secure N on-Secure, fixed to 1; [0]: User Privileged
biu_pad_aruser[2:0]	O	0	Read request user-defined signals: [2]: L2 Cache prefetch request [1]: machine mode request [0]: mmu backfill request
biu_pad_arvalid	O	0	Read address channel valid signal
pad_biu_arready	I	-	Read address channel slave ready signal
Read Data Channel Related Interface			
pad_biu_rid[7:0]	I	-	Read Request Data ID
pad_biu_rdata[127:0]	I	-	Read request data
pad_biu_rrresp[3:0]	I	-	Read the response information of the request [1:0]: 00: OKAY 01: EXOKAY Continued on next page 10: SLVERR 11: DECERR [2]: PASSDIRTY [3]: ISSHARED
pad_biu_rlast	I	-	Read the last beat of the requested data
pad_biu_rvalid	I	83 -	Valid signal for read request data
biu_pad_rready	O	1	Read the ready information of the data channel



Interfaces

Table 9.5 continued from previous page

signal name	I/O	Reset	define
biu_pad_awaddr[39:0]	O	0	Write request address
biu_pad_awlen[1:0]	O	0	Write request burst length 00:1 Shoot 11:4 Shoot
biu_pad_awsize[2:0]	O	0	Write request per beat data bit width 000: 1 byte 001: 2 bytes 010: 4 bytes 011: 8 bytes 100: 16 bytes
biu_pad_awburst[1:0]	O	0	The transport type corresponding to the write request: 01: INCR 10: WRAP
biu_pad_awlock	O	0	Write the access method corresponding to the request: 0: normal access 1: exclusive access
biu_pad_awcache[3:0]	O	0	The type of memory access corresponding to the write request: 0000: Device Non-bufferable (strong order) 0001: Device Bufferable (strong order) 0011: Normal Non-cacheable Bufferable (weak order) 0111: Write-back No-allocate 1111: Write-back Cacheable
biu_pad_awprot[2:0]	O	0	The type of protection for the write request: 0 1 [2]: Data Instruction [1]: Secure N on-Secure, fixed to 1; [0]: User Privileged
biu_pad_awvalid	O	0	Write address channel valid signal
pad_biu_arready	I	-	Write address channel slave ready signal

Write Data Channel Related Interface

biu_pad_wdata[127:0]	O	0	Write request data
biu_pad_wstrb[15:0]	O	820	Valid data fields in the data
biu_pad_wlast	O	0	Final data
biu_pad_wvalid	O	0	Write data channel valid signal

Table 9.5 - continued from previous page

signal name	I/O	Reset	define
biu_pad_wready	I	-	Write data channel slave ready signal
Write-Answer Channel Related Interfaces			
pad_biu_bid[7:0]	I	-	ID of the write response
pad_biu_bresp[1:0]	I	-	Write an answer message Write request response message [1:0]: 00:OKAY 01:EXOKAY 10: SLVERR 11: DECERR
pad_biu_bvalid	I	-	Write answer channel's valid signal
biu_pad_bready	O	1	Write the ready signal of the answer channel

Chapter X. Commissioning

10.1 Functions of the Debug Unit

The debug interface is the channel through which the software interacts with the processor. Through the debug interface, the user can obtain information about the CPU's registers and memory contents, including other on-chip device information. In addition, operations such as program download can also be done through the debug interface.

The C910MP supports the IEEE-1149.1-compliant JTAG communication protocol (commonly known as JTAG5) and can be integrated with existing JTAG components or standalone JTAG controllers.

The main features of the debugging interface are as follows:

- Use the standard JTAG interface for debugging;
- Supports both synchronous and asynchronous debugging, ensuring that the processor is put into debug mode under extreme conditions;
- Soft breakpoints are supported;
- Multiple memory breakpoints can be set;
- Checks and sets CPU register values;
- Check and change memory values;
- Single-step or multi-step execution of instructions is possible;
- Quickly download the program;
- Debug mode can be entered after CPU reset.

The debugging work of C910 is accomplished by the debugging software, debugging agent service program, debugger and debugging interface. The location of the debugging interface in the whole CPU debugging environment is shown in [Figure 10.1](#). The debugging software and the debugging agent service program are interconnected through the network, the debugging agent service program and the debugger are connected through USB, and the debugger communicates with the debugging interface of the CPU in JTAG mode.

10.2 Debug Unit Connection to CPU Core

The C910MP utilizes a multi-core, single-port debugging framework with access to each CORE's auxiliary debug unit through a shared JTAG interface.

(HAD) which triggers the CORE to enter and exit debug mode and access processor resources. The target CORE is specified by setting the CORESEL field of HACR in the JTAG interface and then configuring the HAD register of the target CORE.

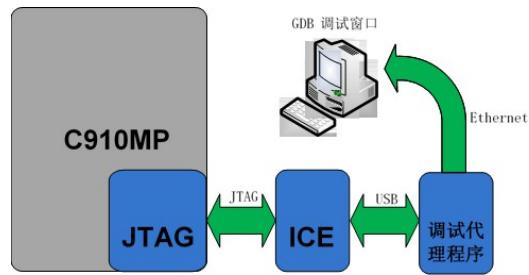


Figure 10.1: Location of the Debug Interface in the Overall CPU Debug Environment

In a multicore debugging scenario, when a CORE enters debug mode, it needs to pull one or more other cores into debug mode; and when a CORE exits debug mode, it needs to pull one or more other cores out of debug mode. Therefore, the C910MP is designed with a centralized Event Transfer Module (ETM) for transferring debug events (entering debug and exiting debug) between cores. When the C910 core receives a debug command from ICE and generates a debug in or debug out event, it sends the event to the ETM at the same time, which forwards the event to other COREs, realizing the purpose of synchronous debug in and debug out of multiple COREs. The C910 multicore debugging framework is shown in Figure 10.2 (dual-core scenario)

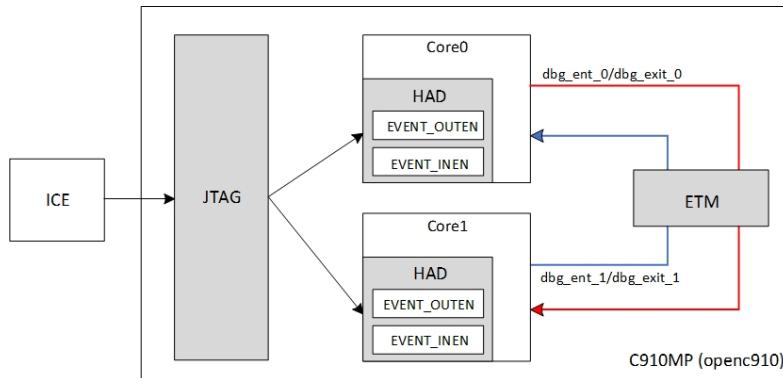


Figure 10.2: Overall Framework for Multicore Debugging

- Scenario where a multi-core system enters debugging:

When a CORE enters the debug mode, it will generate an outgoing DBG_ENT signal, and the EVENT_OUTEN register of the CORE controls whether the DBG signal can be transmitted to the ETM module. When the EVENT_OUTEN register is enabled, the DBG signal is transmitted to other COREs through the ETM module, and whether the other COREs can be pulled into the debug state depends on their respective EVENT_INEN registers.

- Scenarios for exiting debugging of multicore systems:

When a CORE exits the debug mode, it will generate an outgoing DBG_EXIT signal, and the EVENT_OUTEN register of the CORE controls whether the DBG_EXIT signal can be transmitted to the ETM module. When the EVENT_OUTEN register is

Chapter 10

Commissi

on~~ing~~abled, the DBG_EXIT signal is transmitted through the ETM module to the other COREs, and whether the other COREs can be pulled out of the debug state depends on their respective EVENT_INEN registers.

10.3 Debug Interface Signals

The interfaces between the debug module and the outside world are mainly JTAG-related interface signals and debug-related interface signals. Table 10.1 lists the debug-related interface signals.

Table 10.1: External Interface Signals for the Commissioning Module

signal name	orientations
corex_pad_halted	export s
pad_corex_dbgrq_b	importation
corex_pad_jdb_pm[1:0]	export s
pad_corex_dbg_mask	importation
pad_had_jtg_tclk	importation
pad_had_jtg_trst_b	importation
pad_had_jtg_tdi	importation
had_pad_jtg_tdo	export s
had_pad_jtg_tdo_en	export s
pad_had_jtg_tms	importation

corex_pad_halted

A high level indicates that the corresponding CORE is in debug mode.

pad_corex_dbgrq_b

Synchronize to debug mode request signal, active low. This signal is an external input signal to the CORE, which is synchronized by the system clock in the CORE and input to the HAD, and is used by the HAD to synchronize the CORE into debug mode. Pulling down this signal has the same function as setting the DR bit of HAD register HCR.

corex_pad_jdb_pm[1:0]

The corex_pad_jdb_pm[1:0] signals indicate the current working mode of the corresponding CORE, and can be used to determine whether the CPU has entered the debug mode. The details are shown in Table 10.2.

Table 10.2: Current CPU Status Indicated by PM

had_pad_jdb_pm[1:0]	clarificatio n
00	normal mode
01	Low Power Mode
10	debug mode
11	reservations

pad_corex_dbg_mask

The debug request block signal, driven by the SoC, is used to block debug requests for COREEx. This signal needs to be set high in the power-down flow.

Chapter 10

Commissi

onin

pad_had_jtg_tclk

JTAG clock signal. This signal is an external input signal, usually the clock signal generated by the debugger, to ensure that the frequency of the clock signal is lower than the frequency of the CPU clock signal 1/2 in order to ensure that the debugging module and the normal operation between the CORE.

pad_had_jtg_trst_b

The pad_had_jtg_trst_b signal is a JTAG reset signal that resets the TAP state machine and other related control signals.

JTAG5 Related Signals

- The pad_had_jtg_tdi signal is the HAD-side JTAG serial input signal, which is sampled by the HAD-side on the rising edge of the JTAG clock signal, tclk, and set by the external debugger on the falling edge of the JTAG clock;
- The pad_had_jtg_tms signal is a JTAG mode selection signal that is issued by the debugger to control the operation of the TAP state machine in the HAD.
- The had_pad_jtg_tdo signal is the HAD-side JTAG serial output signal, which is set by the HAD-side on the falling edge of the JTAG clock signal tclk, and sampled by the external debugger on the rising edge of the JTAG clock;
- The had_pad_jtg_tdo_en signal indicates that the had_pad_jtg_tdo signal is valid, and is normally monitored by an external debugger to determine if the had_pad_jtg_tdo signal is valid. The debugger can also decide to sample the had_pad_jtg_tdo signal without observing the signal by using the state of the TAP state machine within the debugger. This signal is mainly used when implementing multiple TAPs. This signal is mainly used to determine which JTAG outputs are available when multiple TAP state machines are implemented.

Chapter XI Power Consumption Management

The C910 supports various power management features, including multiple power domains, single core power down, cluster power down, and L2 cache clearing through external hardware interface. This chapter describes the power management features of C910 in detail.

11.1 Power Domain

The C910 can be divided into up to 3 Power Domains, which are:

- Each core is a Power Domain that includes the core's compute units, control logic, and Cache RAM;
- The L2 subsystem (also known as the "top level") is a Power Domain that includes the CIU, L2C, Debug, PLIC, CLINT, and SYSIO.
and other submodules.

11.2 Low Power Mode Summary

The C910 supports the following low-power modes:

- Normal Mode: Each core and L2 are in normal operation.
- Core WFI Mode: Individual cores are in WFI mode.
- Individual Core Power Down: Individual cores are in a power down state.
- Cluster powered down: The entire cluster, including all cores and L2s, is powered down.

11.3 Core WFI Process

The core executes the WFI low power instruction to enter WFI mode, and at the same time outputs the signal `core(x)_pad_lpmd_b[1:0]=2'b00` to indicate that this core has entered WFI mode. At this point, the L2 subsystem will turn off the global ICG for this core within the cluster.

The core wakes up and exits WFI mode when the following events occur:

- Reset;
 - Interrupt request: External interrupt, software interrupt or timer interrupt request sent by PLIC or CLINT.
 - Debugging Request.
-

Chapter 11

Power

Consumption

When the following event occurs, the core is temporarily woken up to process the event and re-enters low-power mode after processing, but does not exit WFI mode throughout:

- Snoop Requests: Snoop requests sent from other cores.

11.4 Individual core power-down process

The system can completely shut down the static power consumption of the core by shutting down the core power supply. The core power shutdown process is as follows:

C910 Operation performed by the shutdown core:

- Notifies the SoC that it is about to perform a single core power-down process in a manner that depends on the design of the SoC.
- Mask all interrupt requests from the core, including external interrupts, soft interrupts, and timer interrupts, and turn off the interrupt enable bits of MSTATUS/SSTATUS registers (mie,sie) and MIE/SIE registers. If the power-down process is executed in M state, turn off the interrupt enable of MSTATUS and MIE; if the power-down process is executed in S state, turn off the interrupt enable of SSTATUS and SIE.
- Turn off data prefetching.
- The core executes D-Cache INV&CLR ALL to write the dirty line back to L2 Cache.
- The kernel shuts down D-Cache (note: there can't be a store instruction between clearing the cache and shutting it down)
- Turns off the SMPEN bit for a core, blocking snoop requests to that core.
- The core executes the fence iorw, iowr instruction.
- The core executes the low-power instruction

WFI and the core enters the low-power mode.

The system performs the operation:

- The system detects that the core's low-power output signal core(x)_pad_lpmd_b is valid.
- The system sets pad_core(x)_dbg_mask high to block debug requests for cores to be powered down.
- Activates the output signal clamp of the core to be powered down.
- The system pulls down the reset signal pad_core(x)_rst_b for the core to be powered down.
- Power down the core.

The core can only be rebooted by a reset in a powered down state. The re-powering process of the core is as follows:

- The system detects a specific event and decides to re-power up the core (also known as a "wake-up")

Chapter 11

Power

Consumption sets the reset address of the core being awakened.

Management

- Pull down the core's reset signal.

- Turn on the power and keep the reset signal unreleased.
- Release the output signal clamp of the core.
- Releases the core reset signal.

Chapter 11

Power

Consumption

- The awakened core executes the initialization procedure, turns on the SMPEN bit, and performs initialization operations such as MMU, DCACHE enable, and so on.

11.5 Cluster power-down process (hardware clear L2)

First, make sure that all the cores in the Cluster are powered off except for the primary core. The "primary" core is the last core to be powered down, which can be any of the 2 cores.

The operation performed by the master core:

- Notifies the SoC that it is about to perform a Cluster power-down process. How this is done depends on the design of the SoC.
- Masks all interrupt requests from the core, including external interrupts, soft interrupts, and timer interrupts, and turns off the MSTATUS/SSTATUS register interrupt enable bits (mie, sie) and the interrupt enable bits in the MIE/SIE register.
- Turn off data prefetching.
- Perform the D-Cache INV&CLR ALL operation.
- Close the D-Cache (note: there can be no store instruction between clear cache and close cache)
- Turns off the SMPEN bit for the core.
- Executes the fence iorw, iorw command.
- Execute the low-power command WFI to enter the low-power mode. The system performs the operation:
 - The system detects that the low-power output signal core(x)_pad_lpmd_b is valid for the main core.
 - The system sets pad_core(x)_dbg_mask high to block debugging requests for the primary core.
 - Activates the output signal clamp of the master core.
 - The system pulls down the reset signal pad_core(x)_rst_b for the main core.
 - Power down the main core.
 - The system pulls up pad_cpu_l2cache_flush_req to start the process of emptying the L2 cache.
 - Waiting for C910 to return cpu_pad_l2cache_flush_done=1.
 - The system pulls down pad_cpu_l2cache_flush_req(Subsequently the C910 will pull down cpu_pad_l2cache_flush_done)
 - Waiting for C910 to return cpu_pad_no_op=1.
 - Activates the output signal clamp at the top level.

Chapter 11

Power

Consumption the L2 reset signal pad_cpu_rst_b.

Management

Turn off the top power.

The Cluster is re-powered by reset with the following process:

- Pull down the reset signals for all cores and top layers within the Cluster.

Chapter 11

Power

Consumption

- Turn on the power, keep the reset signal unreleased and pll stable.

Management

- Release the output signal clamp on each core and top layer.
- Release the reset signal for each core and top level.
- Execute the reset exception service program to restore the CPU state.

11.6 Cluster power-down process (software clears L2)

First, make sure that all cores in the Cluster are powered off except for the primary core. In this scenario, it is recommended that the SoC differentiate between the roles of the "primary" and "secondary" cores, with Core 0 acting as the "primary" core.

The operation performed by the master core:

- Notifies the SoC that it is about to perform a Cluster power-down process. How this is done depends on the design of the SoC.
- Masks all interrupt requests from the core, including external interrupts, soft interrupts, and timer interrupts, and turns off the MSTATUS/SSTATUS register interrupt enable bits (mie, sie) and the interrupt enable bits in the MIE/SIE register.
- Turn off data prefetching.
- Perform the D-Cache INV&CLR ALL operation.
- Close the D-Cache (note: there can be no store instruction between clear cache and close cache).
- Turns off the SMPEN bit for the core.
- Executes the fence iorw, iorw command.
- Performs CLR & INV L2 Cache operations.
- Executes the fence iorw, iorw command.
- Execute the low-power command WFI

to enter the low-power mode. The

system performs the operation:

- The system detects that the low-power output signal core(x)_pad_lpmd_b is valid for the main core.
- The system sets pad_core(x)_dbg_mask high to block debugging requests for the primary core.
- Activates the output signal clamp of the master core.
- The system pulls down the reset signal pad_core(x)_rst_b for the main core.
- Power down the main core.
- Waiting for C910 to return cpu_pad_no_op=1.

Chapter 11

Power

Consumption Set the output signal clamp at the top level.

Management

- Pull down the L2 reset signal pad_cpu_rst_b.

- Turn off the top power.

11.7 Simplified Scenario: Cluster Overall Power Down Process (Hardware Clear L2)

In some systems, SoC designers may take a simpler way to divide the power domain, i.e., the C910 Cluster (2 cores + L2) is treated as a power domain and powered down as a whole, without distinguishing between the individual cores. In such a scenario, the Cluster power-down process (hardware clearing L2) can take the following steps:

The operation performed by the system:

- Informs the SoC that it is about to enter the overall Cluster power-down process, depending on the design of the SoC. Operations performed by the core(At this point, there is no need to distinguish between the "primary" and "secondary"cores, they follow the same process.)
- Mask all interrupt requests from the core, including external interrupts, soft interrupts, and timer interrupts, and turn off the MSTATUS/SSTATUS register interrupt enable bits (mie, sie) and the MIE/SIE register interrupt enable bits.
- Turn off data prefetching.
- The core executes INV&CLR D-Cache ALL to write the dirty line back to L2 Cache.
- The kernel shuts down D-Cache (there can be no store instructions between clearing the cache and shutting it down)
- The core turns off the SMPEN bit, blocking snoop requests to that core.
- The core executes the fence iorw, iorw instruction.
- The core performs WFI.
- the system performs the operation:
 - Wait for all core(x)_pad_lpmd_b[1:0]==2'b00, i.e.: all CPUs have entered low-power state.
 - Set all pad_core(x)_dbg_mask high to block debugging requests.
 - Pull up pad_cpu_l2cache_flush_req to start the process of hardware flushing the L2 cache.
 - Waiting for the C910 to return cpu_pad_l2cache_flush_done=1, which means the L2 has been flushed.
 - Pull down pad_cpu_l2cache_flush_req(The C910 will then pull down cpu_pad_l2cache_flush_done)
 - Wait for cpu_pad_no_op==1'b1, i.e. L2 goes into idle state. (All CPUs are still in low-power state at

Chapter 11

Power

Consumption (more too)

Management

• Activates the clamp of the cluster output signal.

- Assert all reset signals.
- Power down the entire cluster.

11.8 Simplified Scenario: Cluster Overall Power Down Process (Software Clear L2)

Similarly, the content of this subsection is also applicable to the simplified power domain delineation, i.e., the C910 Cluster (2 cores + L2) is treated as a power domain and powered down as a whole without distinguishing the individual cores. In such a scenario, the Cluster power down process (software clearing L2) can take the following steps:

The operation performed by the system:

Chapter 11

Power

Consumption

- The SoC is notified that it is about to enter the overall Cluster power-Management

down process in a manner that depends on the design of the SoC. The operation performed by the "sub-core"(e.g. CPU1):

- Mask all interrupt requests from the core, including external interrupts, soft interrupts, and timer interrupts, and turn off the MSTATUS/SSTATUS register interrupt enable bits (mie, sie) and the MIE/SIE register interrupt enable bits.
- Turn off data prefetching.
- The core executes INV&CLR D-Cache ALL to write the dirty line back to L2 Cache.
- The kernel shuts down D-Cache (there can be no store instructions between clearing the cache and shutting it down)
- The core turns off the SMPEN bit, blocking snoop requests to that core.
- The core executes the fence iorw, iorw instruction.
- The core performs WFI.

Operations performed by the "main core"(e.g. CPU0):

- Mask all interrupt requests from the core, including external interrupts, soft interrupts, and timer interrupts, and turn off the MSTATUS/SSTATUS register interrupt enable bits (mie, sie) and the MIE/SIE register interrupt enable bits.
 - Turn off data prefetching.
 - The core executes INV&CLR D-Cache ALL to write the dirty line back to L2 Cache.
 - The kernel shuts down D-Cache (there can be no store instructions between clearing the cache and shutting it down)
 - The core turns off the SMPEN bit, blocking snoop requests to that core.
 - The core executes the fence iorw, iorw instruction.
 - The primary core waits for all the secondary cores to enter WFI(The exact way to do this depends on the design of the SoC.)
 - The core executes INV&CLR L2 Cache ALL to clear the L2 cache.
 - The core executes the fence iorw, iorw instruction.
 - The core performs WFI.
- the system performs the operation:
- Wait for all core(x)_pad_lpmd_b[1:0]==2'b00,且 cpu_pad_no_op==1'b1, i.e.: all CPUs have entered the low-power state, L2 has entered the idle state.

Chapter 11

Power

Consumption pad_core(x)_dbg_mask high to block debugging requests.

Management

- Activates the clamp of the cluster output signal.

- Assert all reset signals.
- Power down the entire cluster.

11.9.1 program
ming
model11.9 Programming models and interface signals
related to low power consumption**Machine Mode Listen Enable Register (MSMPR)**

This register is 64 bits wide, only bit[0] is defined (=SMPEN) and the default value is 0. The function of this register is: to control whether the core can accept a listen (snoop) request.

- When MSMPR.SMPEN = 0, the core is not able to process listen requests and the top-level mask sends listen requests to the core.
- When MSMPR.SMPEN = 1, the core is able to handle listen requests and the top layer sends a listen request to the core.

Before the core is powered down, it is required to set the core's corresponding SMPEN=0. After the core is powered up, the software needs to set SMPEN=1 before turning on the D-Cache and MMU. the core must keep SMPEN=1 in normal operation mode.

Machine Mode Reset Vector Base Address Register (MRVBR)

Each core has its own MRVBR, which determines the reset boot address of the core, and which is accessed with "MRO" privilege. The initial value of MRVBR for each core is determined by the hardware signal pad_core(x)_rvba[39:0]. Note: pad_core(x)_rvba[0] should be tied to 0.

11.9.2 interface signal

Control of reset signal

The C910 top level has three reset signals: pad_core0_rst_b, pad_core1_rst_b, and pad_cpu_rst_b. The SoC can control the reset of Core 0, Core 1, and L2 through the above signals.

The C910 communicates with the SoC Power Management Unit mainly through the following signals:

- core(x)_pad_lpmd_b: can be used to determine if a core is in WFI mode. 1 for normal mode; 0 for WFI mode.
- cpu_pad_no_op: L2 Cache idle indication signal. This signal is valid (active high) when all cores are in low power mode and the L2 Cache has completed all transfers.
- pad_cpu_l2cache_flush_req and pad_cpu_l2cache_flush_done: this set of signals is used to flush the L2 cache under the control of the SoC, and the application scenario is Cluster power down. The "req" signal is driven by the SoC, and the "done" signal is driven by the C910. The operation sequence is: SoC first pulls up and holds "req" to start the process of clearing L2; C910 completes the action of clearing L2 and then returns to "done".

Chapter 11

Power

Consumption, please C pulls down the "req" signal; then the C910 pulls down the "done" signal.

Management

Chapter 12: Performance Monitoring Module

12.1 Introduction to PMU

The C910 Performance Monitoring Unit (PMU) complies with the RISC-V standard and is used to count software and some hardware information during program operation for software developers to optimize their programs.

The hardware and software information counted by the performance monitoring unit is divided into the following categories:

- Number and duration of running clocks
- Command Information Statistics
- Processor Key Component Information Statistics

12.2 Programming Model of PMU

12.2.1 PMU's base本 usage

The basic usage of PMU is as follows:

- Disable counting of all events via the mcountinhibit register.
- Zero out the current value of each PMU counter, including: mcycle, minstret, mhpmccounter3 ~ mhpmccounter31.
- Configure the corresponding event for each PMU counter. the C910's "event-counter" correspondence is fixed, so it must be configured according to a fixed pattern. For example, mhpmevent3 must write 0x1 to indicate that mhpmccounter3 is fixed for event 0x1.
(L1 ICache accesses) count; mhpmevent4 must write 0x2 to indicate that mhpmccounter4 is fixed for event 0x2
(L1 ICache Miss count) count; and so on.
- Access authorization: mcounteren determines whether the S state can access the PMU counter; scounteren determines whether the U state can access the PMU counter.
- Uninhibit via mcountinhibit register to start counting. For a specific example, refer to the [PMU Setup Example](#)

12.2.2 PMU Event Overflow Interrupt

The C910 customizes the Machine Mode Event Overflow Register (MCOUNTEROF) and the Machine Mode Event Interrupt Enable Register (MCOUNTERINTEN), whose functions and read/write privileges are described in Appendix c-1, Machine Mode Control Registers. Each bit in the Machine Mode Event Overflow Register (MCOUNTEROF) corresponds to the event counter and is used to indicate whether an overflow has occurred in each event counter. Each bit in the machine mode event interrupt enable register (MCOUNTERINTEN) corresponds to an event counter and is used to control whether an interrupt request will be initiated if an overflow occurs in the corresponding event counter.

The overflow interrupt initiated by the Performance Detection Unit has a uniform interrupt vector number of 17. The enabling and handling of the interrupt is the same as that of a normal private interrupt, see [Exceptions and Interrupts](#) for details.

12.3 PMU-related control registers

12.3.1 Machine mode counter access authorization register (mcouteren)

The machine mode counter access authorization register (mcouteren) which is used to authorize the superuser mode to have access to the user mode counters.

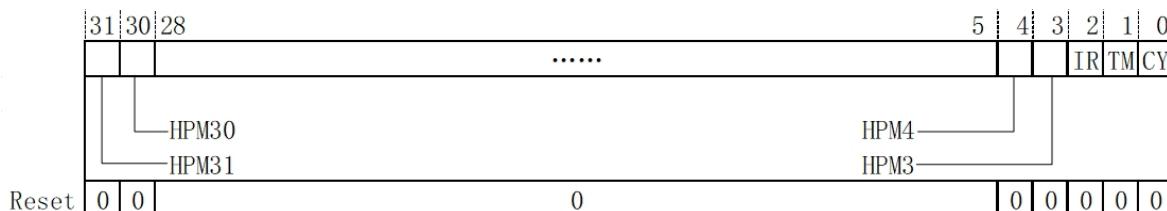


Figure 12.1: Machine Mode Counter Access Authorization Register (MCOUNTEREN)

Table 12.1: Machine Mode Counter Access Authorization Register Descriptions

classifier for honorific people	fill out or in (informat ion on a form)	name (of a thing)	present (sb for a job etc)
31:3	fill out or in (informat ion on a form)	HPM_n	$hpmcountern$ register S-mode access bit: 0: S-mode access to $hpmcountern$ An illegal command exception will occur. 1: S-mode can access $hpmcountern$ normally.

Chapter XII.

Performance

	monitoring module in (information on a form)	IR	minstret register S-mode access bit: 0: S-mode access to the minstret register will generate an illegal instruction exception 1: S-mode can access the minstret register normally.
1	fill out or in (information on a form)	TM	time register S-mode access bit: 0: S-mode access to the time register will occur with an illegal instruction exception 1: When the mco unteren corresponding bit is 1, S-mode can access the time register normally. Otherwise, an illegal instruction exception will occur
0	fill out or in (information on a form)	CY	mcycle register S-mode access bit: 0: S-mode access to cycle registers will occur with an illegal instruction exception 1: S-mode can access cycle registers normally

Chapter XII.

Performance

monitoring module

12.3.2 Superuser mode counter access authorization register (SCOUNTEREN)

The Super User Mode Counter Access Authorization Register (SCOUNTEREN) is used to authorize whether the user mode can access the user mode counters.

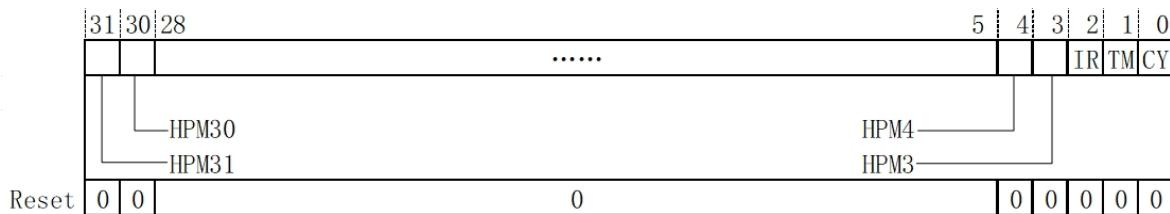


Figure 12.2: Super User Mode Counter Access Authorization Register (SCOUNTEREN)

Table 12.2: Super User Mode Counter Access Authorization Register Description

classifier for honorific people	fill out or in (information on a form)	name (of a thing)	present (sb for a job etc)
31:3	fill out or in (information on a form)	<i>HPMn</i>	<i>hpmcounteren</i> register U-mode access bit: 0: U-mode access to <i>hpmcounteren</i> An illegal instruction exception will occur. 1: When the bit corresponding to <i>mcounteren</i> is 1, the U-mode can access <i>hpmcoun</i> normally. <i>tern</i> , otherwise an illegal instruction exception will occur
2	fill out or in (information on a form)	IR	instret register U-mode access bit: 0: U-mode access to <i>instret</i> registers will result in an illegal instruction exception 1: When the bit corresponding to <i>mcount eren</i> is 1, the U-mode can access the <i>ins tret</i> hosting normally. device, otherwise an illegal instruction exception will occur
1	fill out or in (information on a form)	TM	time register U-mode access bit: 0: U-mo de access to the time register will occur with an illegal instruction exception 1: When the <i>mco unteren</i> corresponding bit is 1, U-mode can access the time register normally. Otherwise, an illegal instruction exception will occur
0	fill out or in (information on a form)	CY	cycle register U-mode access bit: 0: U-mod e access to cycle registers will occur with an illegal instruction exception 1: When <i>mcou nteren</i> corresponds to bit 1, U-mode can access c ycle hosting normally

Chapter XII.

Performance

monitoring module		device, otherwise an illegal instruction exception will occur
-------------------	--	---

12.3.3 Machine mode count inhibit register (**mcountinhibit**)

The machine mode disable count register (**mcountinhibit**), which disables the machine mode counter count. Turning off the counters in scenarios where performance analysis is not required reduces the power consumption of the processor.

Chapter XII. Performance monitoring module

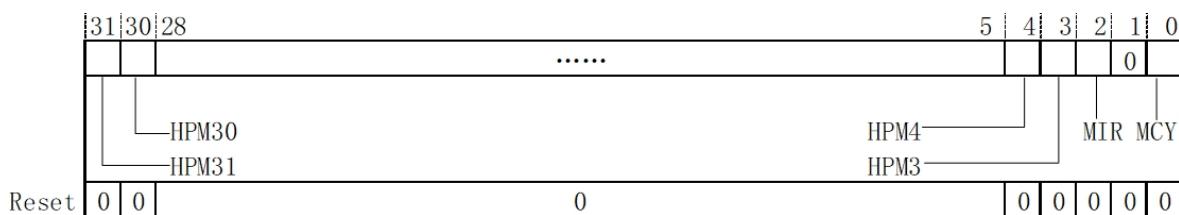


Figure 12.3: Machine Mode Count Inhibit Authorization Register (MCOUNTINHIBIT)

Table 12.3: Machine Mode Count Disable Authorization Register Register Descriptions

classifier for honorific people	fill out or in (information on a form)	name (of a thing)	present (sb for a job etc)
31:3	fill out or in (information on a form)	<i>MHPMn</i>	<i>mhpmcountrn</i> Register disable count bit: 0: Normal count 1: Prohibition of counting
2	fill out or in (information on a form)	MIR	minstret Register disable count bit: 0: Normal count 1: Prohibition of counting
1	-	-	-
0	fill out or in	MCY	mcycle register disable count bit: 0: Normal count 1: Prohibition of counting

Chapter XII.

Performance

monitoring module information on a form)		
---	--	--

12.3.4 Superuser write enable register (mcOUNTERwen)

The Super User Mode Counter Write Enable Register (MCOUNTERwen) is used to authorize the Super User Mode whether or not the Super User Mode can write the Super User Mode Event Counter. This register is a Machine Mode Expansion Register, see [Appendix C-1 Machine Mode Control Registers](#) for a specific description of the register.

12.3.5 Performance Monitoring Event Selection Register

The performance monitoring event selector (mhpmevent3-31)s used to select the counting event corresponding to each counter. In C910, each counter corresponds to one event and cannot be modified, so each event selector can only write the corresponding event number. Only after the event selector writes the index value of the corresponding event and the corresponding event counter is initialized by the csrw instruction, it can be counted normally.

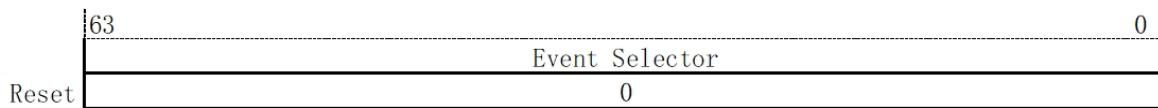


Figure 12.4: Machine Mode Performance Monitoring Event Selection Register (MHPMEVENT)

[Table 12.4](#) provides a description of the Machine Mode Performance Monitoring Event Selection Register.

Chapter XII. Performance monitoring module

Table 12.4: Machine Mode Performance Monitoring Event Selection Register Descriptions

classifier for honorific people	fill out or in (informat ion on a form)	name (of a thing)	present (sb for a job etc)
63:0	fill out or in (informat ion on a form)	Event Index	Performance monitoring event index: 0: No event; 0x1~0x1A: Hardware-implemented performance monitoring events, see Table 12.5 ; >0x1A: Hardware undefined performance monitoring event, used for software customized events.

Table 12.5 shows the correspondence between event selectors and events and counters.

Table 12.5: Counter Event Correspondence List

event selector	indexin g	event	register
mhpmevent3	0x1	L1 ICache Access Counter	mhpmcnter3
mhpmevent4	0x2	L1 ICache Miss Counter	mhpmcnter4
mhpmevent5	0x3	I-UTLB Miss Counter	mhpmcnter5
mhpmevent6	0x4	D-UTLB Miss Counter	mhpmcnter6
mhpmevent7	0x5	JTLB Miss Counter	mhpmcnter7
mhpmevent8	0x6	Conditional Branch Mispredict Counter	mhpmcnter8
mhpmevent9	0x7	reserved	mhpmcnter9
mhpmevent10	0x8	Indirect Branch Mispredict Counter	mhpmcnter10
mhpmevent11	0x9	Indirect Branch Instruction Counter	mhpmcnter11
mhpmevent12	0xA	LSU Spec Fail Counter	mhpmcnter12
mhpmevent13	0xB	Store Instruction Counter	mhpmcnter13
mhpmevent14	0xC	L1 DCache read access Counter	mhpmcnter14
mhpmevent15	0xD	L1 DCache read miss Counter	mhpmcnter15
mhpmevent16	0xE	L1 DCache write access Counter	mhpmcnter16
mhpmevent17	0xF	L1 DCache write miss Counter	mhpmcnter17
mhpmevent18	0x10	L2 Cache read access Counter	mhpmcnter18
mhpmevent19	0x11	L2 Cache read miss Counter	mhpmcnter19
mhpmevent20	0x12	L2 Cache write access Counter	mhpmcnter20
mhpmevent21	0x13	L2 Cache write miss Counter	mhpmcnter21
mhpmevent22	0x14	RF Launch Fail Counter	mhpmcnter22
mhpmevent23	0x15	RF Reg Launch Fail Counter	mhpmcnter23
mhpmevent24	0x16	RF Instruction Counter	mhpmcnter24
mhpmevent25	0x17	LSU Cross 4K Stall Counter	mhpmcnter25

Chapter XII.

Performance

monitoring module			
mhpmevent27	0x18	LSU Other Stall Counter	mhpmccounter26
mhpmevent28	0x19	LSU SQ Discard Counter	mhpmccounter27
mhpmevent29~31	0x1A ≥= 0x1B	LSU SQ Data Discard Counter currently undefined	mhpmccounter28 mhpmccounter29~31

Chapter XII.

Performance

monitoring module

12.3.6 event counter

There are three sets of event counters: Machine Mode Event Counter, User Mode Event Counter and C910 Extended Super User Mode Event Counter. The details are shown in Table 12.6.

Table 12.6: List of Machine Mode Event Counters

name (of a thing)	indexing	fill out or in (information on a form)	startin g value	present (sb for a job etc)
MCYCLE	0xB00	MRW	0x0	cycle counter
MINSTRET	0xB02	MRW	0x0	instructions-retired counter
MHPMCOUNTER3	0xB03	MRW	0x0	performance-monitoring counter
MHPMCOUNTER4	0xB04	MRW	0x0	performance-monitoring counter
...
MHPMCOUNTER31	0xB1F	MRW	0x0	performance-monitoring counter

The list of user mode event counters is shown in Table 12.7.

Table 12.7: List of User Mode Event Counters

name (of a thing)	indexing	fill out or in (information on a form)	startin g value	present (sb for a job etc)
CYCLE	0xC00	URO	0x0	cycle counter
TIME	0xC01	URO	0x0	timer
INSTRET	0xC02	URO	0x0	instructions-retired counter
HPMCOUNTER3	0xC03	URO	0x0	performance-monitoring counter
HPMCOUNTER4	0xC04	URO	0x0	performance-monitoring counter
...
HPMCOUNTER31	0xC1F	URO	0x0	performance-monitoring counter

Table 12.8: List of Superuser Mode Event Counters

name (of a thing)	indexing	fill out or in	startin g value	present (sb for a job etc)

Chapter XII.

Performance

monitoring module

		(information on a form)		
SCYCLE	0x5E0	SRO	0x0	cycle counter
SINSTRET	0x5E2	SRO	0x0	instructions-retired counter
SHPMCOUNTER3	0x5E3	SRO	0x0	performance-monitoring counter
SHPMCOUNTER4	0x5E4	SRO	0x0	performance-monitoring counter
...
SHPMCOUNTER31	0x5FF	SRO	0x0	performance-monitoring counter

Where CYCLE, INSTRET and HPMCOUNTERn for user mode are read-only mappings corresponding to machine mode event counters and TIMER counters are read-only mappings of MTIME registers; and SCYCLE, SINSTRET and SHPMCOUNTERn for super user mode are mappings corresponding to machine mode event counters.

Chapter 13: Sample Programs

This chapter introduces a variety of program examples, including: MMU setup example, PMP setup example, cache setup example, multi-core boot example, synchronization primitive example, PLIC setup example, and PMU setup example.

13.1 Processor Optimized Performance Configuration

The following configurations are used to optimize the performance of the C910:

- MHCR = 0x11ff
- MHINT = 0x6e30c
- MCCR2 = 0xe0000009 (Note: mccc2 contains RAM delay settings, in this example all delays = 0. Customers need to set the appropriate RAM delay according to the actual situation)
- MXSTATUS = 0x638000
- MSMPR = 0x1

```
# mhcr
li x3, 0x11ff
csrs mhcr,x3

#mhint
li x3, 0x6e30c
csrs mhint,x3

# mxstatus
x3, 0x638000 csrs
mxstatus,x3
x3, 0x638000
csrs mxstatus,x3

# msmpr
csrsi msmpr,0x1

# mccc2
```

(continued on next page)


```
li x3,  
0xe0000009 csrs  
mccr2,x3
```

*****13.2 MMU Setting Example*****

```
* Function: An example of setting C910MP MMU.  
* Memory space: Virtual address <-> physical address.  
  
* Pagesize 4K: vpn: {vpn2, vpn1, vpn0} <-> ppn: {ppn2, ppn1, ppn0}  
* Pagesize 2M: vpn: {vpn2, vpn1} <-> ppn: {ppn2, ppn1}  
* Pagesize 1G: vpn: {vpn2} <-> ppn: {ppn2}  
  
*****  
  
/*C910 will invalidate all MMU TLB entries automatically when reset*/  
/*You can use sfence.vma to invalid all MMU TLB entries if necessary*/  
sfence.vma x0, x0  
  
/* Pagesize 4K: vpn: {vpn2, vpn1, vpn0} <-> ppn: {ppn2, ppn1, ppn0}*/  
/* First-level page addr base: PPN (defined in satp)*/  
/* Second-level page addr base: BASE2 (self define)*/  
/* Third-level page addr base: BASE3 (self define)*/  
/* 1. Get first-level page addr base: PPN and vpn*/  
/* Get PPN*/  
csrr x3, satp  
li x4, 0xffffffffffff  
and x3, x3, x4  
  
/*2. Config first-level page*/  
/*First-level page addr: {PPN, vpn2, 3'b0}, first-level page pte:{ 4'bBASE2, 10'b1}  
/*  
/*Get first-level page  
addr*/ slli x3, x3, 12  
/*Get  
vpn2*/ li  
x4, VPN  
li x5, 0x7fc0000
```

(continued on next page)

```
and x4, x4, x5
srli x4, x4, 15
and x5, x3, x4
/*Store pte at first-level page addr*/
li x6, {44'b BASE2, 10'b1}
sd x6, 0(x5)

/*3. Config second-level page*/
/*Second-level page addr: {BASE2, vpn1, 3'b0}, second-level page pte:{ 4'bBASE3, 1'b1} */
/*Get second-level page addr*/
/* VPN1*/
li x4, VPN
li      x5,
0x3fe00 and
x4, x4, x5
srli x4, x4,
9
/*BASE2*/ li
x5, BASE2
srli x5, x5, 12
and x5, x5, x4
/*Store pte at second-level page addr*/
li x6, {44'b BASE3, 10'b1}
sd x6, 0(x5)
/*4. Config third-level page*/
/*Third-level page addr: {BASE3, vpn0, 3'b0}, third-level page pte:{ theadflag, ppn2, ppn1, ppn0, 9'bflags,1'b1} */
/*Get second-level page addr*/
/* VPN0*/
li x4, VPN
li x5,
0x1ff
and x4, x4, x5
srli x4, x4, 3
/*BASE3*/ li
x5, BASE3
srli x5, x5, 12
and x5, x5, x4
/*Store pte at second-level page addr*/
li x6, { theadflag, ppn2, ppn1, ppn0, !b!flags,
1'b1} sd x6, 0(x5)
```

Chapter 13:
Sample
Programs



(continued on next page)

```
/* Pagesize 2M: vpn: {vpn2, vpn1} <-> ppn: {ppn2, ppn1}*/
/*First-level page addr base: PPN (defined in satp)*/
/*Second-level page addr base: BASE2 (self define)*/

/*1. Get first-level page addr base: PPN and vpn*/
/* Get PPN*/
csrr x3, satp
li x4, 0xffffffffffff
and x3, x3, x4

/*2. Config first-level page*/
/*First-level page addr: {PPN, vpn2, 3'b0}, first-level page pte:{4'bBASE2, 10'b1}*/
/*Get first-level page addr*/
slli x3, x3, 12
/*Get
vpn2*/ li
x4, VPN
li x5, 0x7fc0000
and x4, x4, x5
srli x4, x4, 15
and x5, x3, x4
/*Store pte at first-level page addr*/
li x6, {4'b BASE2, 10'b1}
sd x6, 0(x5)

/*3. Config second-level page*/
/*Second-level page addr: {BASE2, vpn1, 3'b0}, second-level page pte:{theadflag, ppn2, ppn1, 9'b0, 9'b flags,1'b1} */
/*Get second-level page addr*/
/*VPN1*/
li x4,
VPN
li      x5,
0x3fe00 and
x4, x4, x5
srli x4, x4,
9
/*BASE2*/ li
x5, BASE2
srli x5, x5, 12
```

Chapter 13:
Sample
Programs^{x5, x4}

```
/*Store pte at second-level page addr*/
```

(continued on next page)

```

li x6, { theadflag, ppn2, ppn1, 9'b0, 9'b flags,1'b1}
sd x6, 0(x5)

/* Pagesize 1G: vpn: {vpn2} <-> ppn: {ppn2}*/
/*First-level page addr base: PPN (defined in satp)*/
/*1. Get first-level page addr base: PPN and vpn*/
/* Get PPN*/
csrr x3, satp
li x4, 0xffffffffffff
and x3, x3, x4

/*2. Config first-level page*/
/*First-level page addr: {PPN, vpn2, 3'b0}, first-level page pte:{theadflag, ppn2, 9'b0, 9'b0, 9'b flags,1'b1}*/
/*Get first-level page
addr*/ slli x3, x3, 12
/*Get
vpn2*/ li
x4, VPN
li x5, 0x7fc0000
and x4, x4, x5
srli x4, x4, 15
and x5, x3, x4
/*Store pte at first-level page addr*/
li x6, { theadflag, ppn2, 9'b0, 9'b0, 9'b flags,1'b1} sd
x6, 0(x5)

```

13.3 PMP Setup Example

```

*****
* Function: An example of setting C910MP PMP.
* 0x0 ~ 0xf0000000, TOR mode, RWX
* 0xf0000000 ~ 0xf8000000, NAPOT mode, RW
*0xffff73000 ~ 0xffff74000, NAPOT mode, RW
*0xffffc0000 ~ 0xffffc2000, NAPOT mode, RW
* The above 4 slices are configured with different execution privileges. In addition, in order
to prevent the CPU from speculatively executing to unsupported address areas in different
modes, especially in the machine mode which has full execution privileges by default, it is
necessary to configure the PMP. Specifically, the
is to configure the remaining address areas without any privileges after configuring the address
areas that require execution privileges, as shown in the following example.
*****/

```

```
# pmpaddr0, 0x0 ~ 0xf0000000, TOR mode, read/write executable rights
li x3, (0xf0000000 >> 2)
csrw pmpaddr0, x3

# pmpaddr1, 0xf0000000 ~ 0xf8000000, NAPOT mode, read/write access
li x3, ( 0xf0000000 >> 2 | (0x8000000-1) >> 3))
csrw pmpaddr1, x3

# pmpaddr2, 0xffff73000 ~ 0xffff74000, NAPOT mode, read/write access
li x3, ( 0xffff73000 >> 2 | (0x1000-1) >> 3))
csrw pmpaddr2, x3

# pmpaddr3, 0xfffffc0000 ~ 0xfffffc2000, NAPOT mode, read/write access
li x3, ( 0xfffffc0000 >> 2 | (0x2000-1) >> 3))
csrw pmpaddr3, x3

# pmpaddr4, 0xf0000000 ~ 0x100000000, NAPOT mode, no permissions
li x3, ( 0xf0000000 >> 2 | (0x10000000-1) >> 3))
csrw pmpaddr4, x3

# pmpaddr5, 0x100000000 ~ 0xffffffffffff, TOR mode, no privileges
li x3, (0xffffffffffff >> 2)
csrw pmpaddr5, x3

# PMPCFG0, configure the execution
privilege/mode/lock bit of each table entry,
if lock is 1, the table entry is valid only in
machine mode
li x3,0x88989b9b9b8f
csrw pmppcfg0, x3

# pmpaddr5, 0x100000000 ~ 0xffffffffffff, TOR mode, pmpaddr5 is hit at
0x100000000 <= addr < 0xffffffffffffffff, but 0xffffffffffff000 ~
The 0xffffffffffff address interval cannot hit pmpaddr5 (the minimum granularity of PMP in
the C910 is 4K) and if you need to mask the last 4K of 1T space, you need to configure
another table entry in NAPOT mode.
```

13.4 Cache Example

13.4.1 Example of Cache Enabling

```
/*C910 will invalidate all I-cache automatically when reset*/
/*You can invalidate I-cache by yourself if necessary*/
/*Invalidate I-cache*/
Chapter0x33:
Sample, x3
Programx1
```



(continued on next page)

```
csrs mcor, x3
// You can also use icache instrucitons to replace the invalidate sequence
// if theadisae is enabled.
//icache.iall
//sync.is

/*Enable I-cache*/
li x3, 0x1
csrs mhcr, x3

/*C910 will invalidate all D-cache automatically when reset*/
/*You can invalidate D-cache by yourself if necessary*/
/*Invalidate D-cache*/
li x3, 0x33
csrc mcor, x3
li x3, 0x12
csrs mcor, x3

// You can also use dcache instrucitons to replace the invalidate sequence
// if theadisae is enabled.
// dcache.iall
// sync.is

/*Enable D-cache*/
li x3, 0x2
csrs mhcr, x3

/*C910 will invalidate all L2 cache automatically when reset*/
/*You can invalidate L2 by yourself if necessary*/
/*Invalidate L2-cache if theadisae is enabled*/
l2cache.iall
sync.is

/*Enable L2-cache*/
li x3, 8
csrs mccr2, x3
```

13.4.2 Example of Synchronization of Instruction Cache and Data Cache

CPU0

Chapter 13: Sample Programs

```
sd x3,0(x4) // a new instruction defined in x3
               // is stored to program memory address defined in x4.
dcache.cval1 r0 // clean the new instruction to the shared L2
cache.sync.    s// Ensure completion of the sync.s// ensure
completion of clean operation.
               // The dcache clean is not necessarily if INSDE is not enabled.
icache.iva r0 // Invalid icache according to shareable configuration.
sync.s/fence.i // ensure completion in all CPUs.
sd x5,0(x6)   //set flag to signal operation completion.
sync.is
jr x4 // jmp to new code
```

CPU1~CPU3

```
WAIT_FINISH.
ld x7,0(x6)
bne x7,x5, WAIT_FINISH // wait CPU0 modification finish.
sync.is
jr           x4// jmp to new code
```

13.4.3 Example of TLB Synchronization with Data Cache

CPU0

```
sd x4,0(x3) // update a new translation table entry
sync.is/fence.i // ensure completion of update operation.
sfence.vma x5,x0 // invalid the TLB by va
sync.is/fence.i // ensure completion of TLB invalidation and
               // synchronises context
```

13.5 Example of synchronized proto-language

CPU0

```
0x1 li  
x6, 0x0  
                                // (x3) is the lock address. 0: Free; 1:  
ACQUIRE_LOCK.  
$ample0(x3)          // Read lock  
Diagram ACQUIRE_LOCK // Try again if the lock is in use
```



(continued on next page)

```
sc x5, x1, 0(x3)          //Attempt to store newvalue
bne x6, x5, ACQUIRE_LOCK // Try again if fail
sync.s

...                      // Critical section code
```

CPU1

```
sync.s/fence.i      // Ensure all operations are observed before clearing the
sd x0, 0(x3)        lock.
                     // Clear the lock.
```

13.6 PLIC Setting Example

```
//Init id 1 machine mode int for hart 0
/*1.set hart threshold if needed*/
li x3, (PLIC_base_addr + 0x200000) // h0 mthreshold addr
li x4, 0xa //threshold value
sw x4,0x0(x3) // set hart0 threshold as 0xa

/*2.set priority for int id 1*/
li x3, (PLIC_base_addr + 0x0) // int id 1 prio addr
li x4, 0x1f // prio value
sw x4,0x4(x3) // init id1 priority as 0x1f

/*3. enable m-mode int id1 to hart*/
li x3, (PLIC_base_addr + 0x2000) // h0 mie0
addr li x4, 0x2
sw x4,0x0(x3) // enable int id1 to hart0

/*4. set ip or wait external int*/
/*following code set ip*/
li x3, (PLIC_base_addr + 0x1000) // h0 mthreshold addr
li x4, 0x2 // id 1 pending
sw x4, 0x0(x3) // set int id1 pending

/*5.core enters interrupt handler, read PLIC CLAIM and get ID*/

/*6.core takes interrupt*/
```

(continued on next page)

```
/*7.core needs to clear external interrupt source if LEVEL(not PULSE)
configured, then core writes ID to PLIC CLAIM and exits interrupt*/.
```

13.7 PMU Setting Example

```
/*1.inhibit counters
counting*/ li x3, 0xffffffffffff
csrw mcountinhibit, x3

/*2.C910 will initialize all pmu counters when reset*/
/*you can initial pmu counters manually if necessarily*/
csrw mcycle, x0
csrw minstret, x0
csrw mhpmcounter3,
x0
...
csrw mhpmcounter31, x0

/*3.configure mhpmevent*/
li x3, 0x1
csrw mhpmevent3, x3 // mhpmcounter3 count event: L1 ICache Access Counter
li x3, 0x2
csrw mhpmevent4, x3 // mhpmcounter4 count event: L1 ICache Miss Counter
...
li x3, 0x13
csrw mhpmevent21, x3 // mhpmcounter21 count event: L2 Cache write miss Counter

/*4. configure mcounteren and scounteren*/
li x3, 0xffffffffffff
csrw mcounteren, x3 // enable super mode to read
hpmcounter li x3, 0xffffffffffff
csrw scounteren, x3 // enable user mode to read hpmcounter

/*5. enable counters to count when you want*/
csrw mcountinhibit, x0
```

Chapter 14 Appendix A Standardized Directive Terminology

The C910MP implements the RV64IMAFDC instruction set package, and the following sections describe each instruction according to the different instruction sets.

14.1 Appendix A-1 I Command Terminology

The following is a specific description of the RISC-V I instruction set as implemented in the C910, with instructions listed in alphabetical order.

The bit width of the instructions in this section is 32 bits by default, but the system will compile some instructions into 16-bit compressed instructions under certain circumstances, please refer to [Appendix A-6 C Instruction Terminology](#) for the detailed description of compressed instructions.

14.1.1 ADD - signed addition instruction

Grammar:

add rd, rs1, rs2

Operation:

$rd \leftarrow rs1 + rs2$

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
0000000	rs2	rs1	000	rd	0110011	

14.1.2 ADDI - Signed Immediate Number Addition Instruction

Grammar:

addi rd, rs1, imm12

Chapter 14 Appendix A

Standardized Directive

Terminology:

$rd \leftarrow rs1 + \text{sign_extend}(imm12)$

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Command Format:

31	20 19	15 14	12 11	7 6	0
imm12[11:0]	rs1	000	rd	0010011	

14.1.3 ADDIW - Low 32-bit Signed Immediate Number Addition Instruction

Grammar:

addiw rd, rs1, imm12

Operation:

$tmp[31:0] \leftarrow rs1[31:0] + \text{sign_extend}(imm12)[31:0]$

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

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Command Format:

31	20 19	15 14	12 11	7 6	0
imm12[11:0]	rs1	000	rd	0011011	

14.1.4 ADDW - low 32-bit signed addition instruction

Grammar:

addw rd, rs1, rs2

Operation:

$tmp[31:0] \leftarrow rs1[31:0] + rs2[31:0]$

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

Execute permissions:

Chapter 14 Appendix A

Standardized Directive

Terminology

M mode/S mode/U mode

Exception:

not have

Command Format:

:31	25 24	20 19	15 14	12 11	7 6	0
0000000	rs2	rs1	000	rd	0111011	

14.1.5 AND - By Bit and Command

Grammar:

and rd, rs1, rs2

Operation:

$rd \leftarrow rs1 \& rs2$

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Command Format:

:31	25 24	20 19	15 14	12 11	7 6	0
0000000	rs2	rs1	111	rd	0110011	

14.1.6 ANDI - Immediate Number By Bit with Instruction

Grammar:

andi rd, rs1, imm12

Operation:

$rd \leftarrow rs1 \& \text{sign_extend}(imm12)$

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Command Format:

Chapter 14 Appendix A

Standardized Directive

Terminology:

31 imm12[11:0]	20 19 rs1	15 14 111	12 11 rd	7 6 0	0010011
------------------	---------------	---------------	--------------	-----------	---------

14.1.7 AUIPC - PC Higher Immediate Number Addition Instruction

Grammar:

auipc rd, imm20

Operation:

$rd \leftarrow \text{current pc} + \text{sign_extend}(imm20 << 12)$

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Command Format:

31 imm20[19:0]	12 11 rd	7 6 0	0010111
------------------	--------------	-----------	---------

14.1.8 BEQ - Branch Equalization Instruction

Grammar:

beq rs1, rs2, label

Operation:

if ($rs1 == rs2$)

next pc = current pc + sign_extend(imm12 << 1)

else

next pc = current pc + 4

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Description:

The assembler calculates imm12 from the label

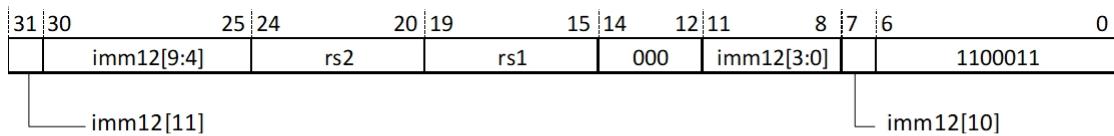
Instruction jump range ±4KB address space

Chapter 14 Appendix A

Standardized Directive

Terminology

Command Format:



14.1.9 BGE - Signed Greater Than Equal Branching Instructions

Grammar:

bge rs1, rs2, label

Operation:

if ($rs1 \geq rs2$)

next pc = current pc + sign_extend(imm12 <<1)

else

next pc = current pc + 4

Execute permissions:

M mode/S mode/U mode

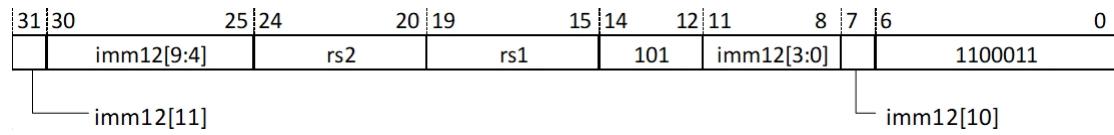
Exception:

not have

Description:

The assembler calculates the jump range of the imm12 instruction from the label to be $\pm 4KB$ address space

instruction format:



14.1.10 BGEU - unsigned greater than or equal to branching instruction

Grammar:

Chapter 14 Appendix A

Standardized Directive

bgeu rs1, rs2, label

Operation:

```
if (rs1 >= rs2)
    next pc = current pc + sign_extend(imm12<<1)
else
    next pc = current pc + 4
```

Execute permissions:

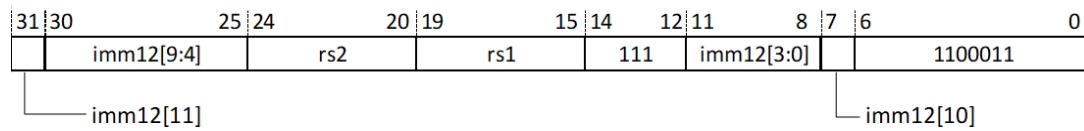
M mode/S mode/U mode

Exception:

not have

Description:

The assembler calculates the jump range of the imm12 instruction from the label to



be $\pm 4\text{KB}$ address space

instruction format:

14.1.11 BLT - Signed Less Than Branching Instructions

Grammar:

blt rs1, rs2, label

Operation:

```
if (rs1 < rs2)
    next pc = current pc + sign_extend(imm12<<1)
else
    next pc = current pc + 4
```

Execute permissions:

M mode/S mode/U mode

Exception:

not have

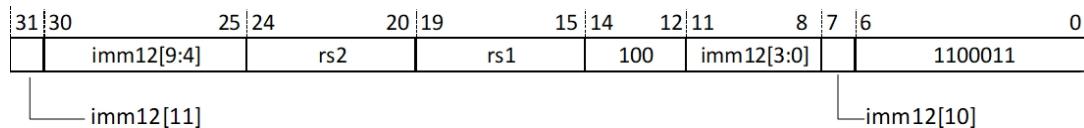
Chapter 14 Appendix A

Standardized Directive

Description:

The assembler calculates the jump range of the imm12 instruction from the label to be $\pm 4\text{KB}$ address space

instruction format:



14.1.12 BLTU - Unsigned Less Than Branching Instruction

Grammar:

bltu rs1, rs2, label

Operation:

if ($\text{rs1} < \text{rs2}$)

next pc = current pc + sign_extend(imm12<<1)

else

next pc = current pc + 4

Execute permissions:

M mode/S mode/U mode

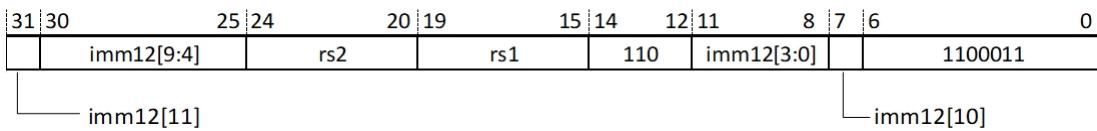
Exception:

not have

Description:

The assembler calculates the jump range of the imm12 instruction from the label to be $\pm 4\text{KB}$ address space

instruction format:



14.1.13 BNE - Branch Not Equal Instruction

Chapter 14 Appendix A

Standardized Directive

Terminology:

Chapter 14 Appendix A

Standardized Directive

Terminology
one rs1, rs2, label

Operation:

```
if (rs1 != rs2)
    next pc = current pc + sign_extend(imm12<<1)
else
    next pc = current pc + 4
```

Execute permissions:

M mode/S mode/U mode

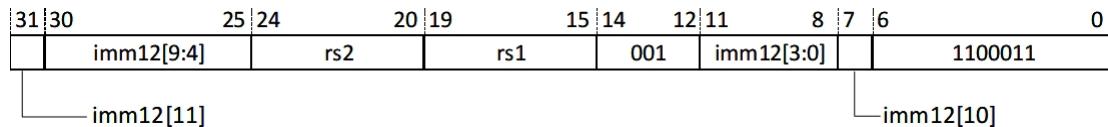
Exception:

not have

Description:

The assembler calculates the jump range of the imm12 instruction from the label to be $\pm 4\text{KB}$ address space

instruction format:



14.1.14 CSRRC - Control Register Clear Transfer Instruction

Grammar:

csrrc rd, csr, rs1

Operation:

rd \leftrightarrow csr

csr \leftarrow csr & (\sim rs1)

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Description:

Chapter 14 Appendix A

Standardized Directive

Terminology

The control registers that are allowed to be accessed under each authority are different, please refer to the Control Registers section for details. When rs1=x0, this instruction does not generate a write operation and does not generate an exception raised by the write behavior. **Instruction Format:**

31	20 19	15 14	12 11	7 6	0
csr	rs1	011	rd	1110011	

14.1.15 CSRRCI - Control Register Immediate Count Clear Transfer Instruction

Grammar:

csrrci rd, csr, imm5

Operation:

rd ↔ csr

csr ← csr & ~zero_extend(imm5)

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Description:

The control registers that are allowed to be accessed under each authority are different, please refer to the Control Registers section for details. When rs1=x0, this instruction does not generate a write operation and does not generate an exception raised by the write behavior. **Instruction Format:**

31	20 19	15 14	12 11	7 6	0
csr	imm5	111	rd	1110011	

14.1.16 CSRRS - Control Register Reset Transfer Instruction

Grammar:

csrrs rd, csr, rs1

Operation:

rd ↔ csr

csr ← csr | rs1

Chapter 14 Appendix A

Standardized Directive

Terminology permissions:

M mode/S mode/U mode

Chapter 14 Appendix A

Standardized Directive

Terminology:

Illegal instruction exception

Description:

The control registers that are allowed to be accessed under each authority are different, please refer to the Control Registers section for details. When rs1=x0, this instruction does not generate a write operation and does not generate an exception raised by the write behavior. **Instruction Format:**

31	20 19	15 14	12	11	7 6	0
csr	rs1	010		rd	1110011	

14.1.17 CSRRSI - Control Register Immediate Number Reset Transfer Instruction

Grammar:

csrrsi rd, csr, imm5

Operation:

rd ↔ csr

csr ← csr | zero_extend(imm5)

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Description:

The control registers that are allowed to be accessed under each authority are different, please refer to the Control Registers section for details. When rs1=x0, this instruction does not generate a write operation and does not generate an exception raised by the write behavior. **Instruction Format:**

31	20 19	15 14	12	11	7 6	0
csr	imm5	110		rd	1110011	

14.1.18 CSRRW - Control Register Read/Write Transfer Instruction

Grammar:

Chapter 14 Appendix A

Standardized Directive

Terminology csrrw, rd, csr, rs1

Operation:

Chapter 14 Appendix A

Standardized Directive

Terminology

$rd \leftrightarrow csr \leftrightarrow sr$

$\leftarrow rs1$

execute

permisso

n:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Description:

The control registers that are allowed to be accessed under each authority are different, please refer to the Control Registers section for details. When $rs1=x0$, this instruction does not generate a write operation and does not generate an exception raised by the write behavior. **Instruction Format:**

31	20 19	15 14	12 11	7 6	0
csr	rs1	001	rd	1110011	

14.1.19 CSRRWI - Control Register Immediate Read/Write Transfer Instruction

Grammar:

csrrwi rd, csr, imm5

Operation:

$rd \leftrightarrow csr$

$csr[4:0] \leftarrow imm5$

$csr[63:5] \leftarrow csr[63:5]$

execute

permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Description:

The control registers that are allowed to be accessed under

Chapter 14 Appendix A

Standardized Directive

Terminology authority are different, please refer to the Control

Registers section for details. When rs1=x0, this instruction does not generate a write operation and does not generate an exception raised by the write behavior. **Instruction Format:**

31 csr	20 19 imm5	15 14 101	12 11 rd	7 6 1110011	0
-----------	---------------	--------------	-------------	----------------	---

Chapter 14 Appendix A

Standardized Directive

14.1.20 EBREAK - breakpoint instruction

Syntax:

x:

ebreak

Operat

ion:

Generate a breakpoint exception or enter debug mode

Execute permissions:

M mode/S mode/U mode

Exception:

breakpoint exception

Command Format:

31	20 19	15 14	12	11	7 6	0
000000000001	00000	000		00000	1110011	

14.1.21 ECALL - Environmental Exception Directive

Syntax:

x: ecall

Oper

ation:

create environmental anomalies

Execute permissions:

M mode/S mode/U mode

Exception:

User mode environment call exception, super user mode environment call exception, machine mode environment call exception

Command Format:

31	20 19	15 14	12	11	7 6	0
000000000000	00000	000		00000	1110011	

14.1.22 FENCE - store synchronization instructions

Chapter 14 Appendix A

Standardized Directive

Terminology:

fence iorw, iorw

Operation:

Chapter 14 Appendix A

Standardized Directive

Terminology

It is guaranteed that all read/write memory or peripheral instructions in the presequence of the instruction are observed earlier than all read/write memory or peripheral instructions in the postsequence of the instruction.

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Description:

$pi=1, so=1$, the command syntax is fence i,o, and so on

Command Format:

31	28	27	26	25	24	23	22	21	20	19	15	14	12	11	7	6	0
0000	pi	po	pr	pw	si	so	sr	sw	00000	00000	000	00000	00000	0001111			

14.1.23 FENCE.I - Command Flow Synchronization Command

Synta

x:

fence.i

Opera

tion:

Clear the icache to ensure that all data accesses in the preamble of the instruction can be accessed by the fetch operation after the instruction.

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Command Format:

31	28	27	24	23	20	19	15	14	12	11	7	6	0
0000	0000	0000	00000	00000	001	00000	0001111						

14.1.24 JAL - Direct Jump Subprogram Instruction

Grammar:

jal rd, label

Operation:

Chapter 14 Appendix A

Standardized Directive

Terminology $\text{next_pc} \leftarrow \text{current_pc} + \text{sign_extend}(\text{imm20} << 1)$

$\text{rd} \leftarrow \text{current_pc} + 4$

Chapter 14 Appendix A

Standardized Directive

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Description:

The assembler calculates the imm20 instruction jump range from the label to $\pm 1\text{MB}$ address space **instruction**

format:

31 30	20 19	11	7 6	0
imm20[9:0]	imm20[18:11]	rd	1101111	
imm20[19]		imm20[10]		

14.1.25 JALR - Register Jump Rotor Program Instruction

Grammar:

jalr rd, rs1, imm12

Operation:

next pc \leftarrow (rs1 + sign_extend(imm12)) &

64 \blacktriangleright hfffffffffffffff fe rd \leftarrow current pc + 4

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Description:

When machine mode or MMU is off, the instruction jump range is all HTB of address space When non-machine mode \boxtimes MMU is on, the instruction jump range is all 512GB of address

space Instruction Format:

31	20 19	15 14	12 11	7 6	0
imm12[11:0]	rs1	000	rd	1100111	

Chapter 14 Appendix A

Standardized Directive

14.1.26 LB - Signed Extended Byte Load Instruction

Grammar:

lb rd, imm12(rs1)

Operation: address

$\leftarrow rs1 + \text{sign_extend}(imm12)$ rd \leftarrow

sign_extend(mem[address]) Execute

permission:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception

Command Format:

31	20 19	15 14	12 11	7 6	0
imm12[11:0]	rs1	000	rd	0000011	

14.1.27 LBU - unsigned extended byte load instruction

Grammar:

lbu rd, imm12(rs1)

Operation: address

$\leftarrow rs1 + \text{sign_extend}(imm12)$ rd \leftarrow

zero_extend(mem[address]) Execute

permission:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception

Command Format:

31	20 19	15 14	12 11	7 6	0
imm12[11:0]	rs1	100	rd	0000011	

14.1.28 LD - double word load instruction

Grammar:

Chapter 14 Appendix A

Standardized Directive

Terminology: lh rd, imm12(rs1)

Operation:

address \leftarrow rs1+sign_extend(imm12)

rd \leftarrow mem[(address+7):address]

Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception

Command Format:

31	20 19	15 14	12 11	7 6	0
imm12[11:0]	rs1	011	rd	0000011	

14.1.29 LH - Signed Extended Half-Word Load Instruction

Grammar:

lh rd, imm12(rs1)

Operation:

address \leftarrow rs1+sign_extend(imm12)

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

Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception

Command Format:

31	20 19	15 14	12 11	7 6	0
imm12[11:0]	rs1	001	rd	0000011	

14.1.30 LHU - unsigned extended half-word load instruction

Grammar:

lhu rd, imm12(rs1)

Operation:

Chapter 14 Appendix A

Standardized Directive

Terminology

$\text{address} \leftarrow \text{rs1} + \text{sign_extend}(\text{imm12})$

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

Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception

Command Format:

31	20 19	15 14	12 11	7 6	0
imm12[11:0]	rs1	101	rd	0000011	

14.1.31 LUI - High Immediate Number Load Instruction

Grammar:

lui rd, imm20

Action:

$\text{rd} \leftarrow \text{sign_extend}(\text{imm20} \ll 12)$

execute permissions:

M mode/S mode/U mode

Exception:

not have

Command Format:

31	12 11	7 6	0
imm20[19:0]	rd	0110111	

14.1.32 LW - Signed Extended Word Load Instruction

Grammar:

lw rd, imm12(rs1)

Operation:

$\text{address} \leftarrow \text{rs1} + \text{sign_extend}(\text{imm12})$

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

Execute permissions:

M mode/S mode/U mode

Chapter 14 Appendix A

Standardized Directive

Terminology:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception

Command Format:

31	20 19	15 14	12 11	7 6	0
imm12[11:0]	rs1	010	rd	0000011	

14.1.33 LWU - unsigned extended word load instruction

Grammar:

lwu rd, imm12(rs1)

Operation:

address \leftarrow rs1+sign_extend(imm12)

rd \leftarrow zero_extend(mem[(address+3):address])

Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception

Command Format:

31	20 19	15 14	12 11	7 6	0
imm12[11:0]	rs1	110	rd	0000011	

14.1.34 MRET - Machine Mode Exception Return Instruction

Syntax

x: mret

Operation:

next pc \leftarrow mepc

mstatus.mie \leftarrow mstatus.mpie

mstatus.mpie \leftarrow 1

Execute permissions:

M mode

Chapter 14 Appendix A

Standardized Directive

Termination:

Illegal instruction exception

Chapter 14 Appendix A

Standardized Directive

Terminology

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
0011000		00010		00000	000		00000		1110011		

14.1.35 OR - Orders by Bit or

Grammar:

or rd, rs1, rs2

Operation:

$rd \leftarrow rs1 | rs2$

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
0000000		rs2		rs1	110		rd		0110011		

14.1.36 ORI - immediate number by bit or instruction

Grammar:

ori rd, rs1, imm12

Operation:

$rd \leftarrow rs1 | sign_extend(imm12)$

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Command Format:

31	20	19	15	14	12	11	7	6	0
imm12[11:0]		rs1	110		rd		0010011		

Chapter 14 Appendix A

Standardized Directive

14.1.37 SB - byte storage instructions

Grammar:

sb rs2, imm12(rs1)

Operation:

address \leftarrow rs1+sign_extend(imm12)

mem[:address] \leftarrow rs2[7:0]

Execute permissions:

M mode/S mode/U mode

Exception:

Memory Instruction Non-Aligned Access Exception, Memory Instruction Access Error Exception, Memory Instruction Page Error Exception

Command Format:

:31	25	24	20	19	15	14	12	11	7	6	0
imm12[11:5]		rs2		rs1		000		imm12[4:0]		0100011	

14.1.38 SD - Double Word Storage Instruction

Grammar:

sd rs2, imm12(rs1)

Operation:

address \leftarrow rs1+sign_extend(imm12)

mem[(address+7):address] \leftarrow rs2

Execute permissions:

M mode/S mode/U mode

Exception:

Memory Instruction Non-Aligned Access Exception, Memory Instruction Access Error Exception, Memory Instruction Page Error Exception

Command Format:

:31	25	24	20	19	15	14	12	11	7	6	0
imm12[11:5]		rs2		rs1		011		imm12[4:0]		0100011	

14.1.39 SFENCE.VMA - Virtual Memory Synchronization Instruction

Grammar:

Chapter 14 Appendix A

Standardized Directive

Terminology

silence.vma rs1,rs2

Operation:

Invalidation and synchronization operations for virtual memory

Execute permissions:

M mode/S mode

Exception:

Illegal instruction exception

Description: mstatus.tvm=1, execution of this command in

superuser mode causes an illegal command exception. rs1:

virtual address, rs2: asid

- rs1=x0, rs2=x0 Invalidates all table entries in the TLB.
- When rs1!=x0 and rs2=x0, invalidate all table entries in the TLB that hit the rs1 virtual address.
- When rs1=x0 and rs2!=x0, invalidate all table entries in the TB that hit the rs2 process number.
- When rs1!=x0 and rs2!=x0, invalidate all table entries in the TLB that hit the rs1 virtual address and rs2 process number.

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
0001001		rs2		rs1	000		00000		1110011		

14.1.40 SH - Half Word Storage Instruction

Grammar:

sh rs2, imm12(rs1)

Operation:

address←rs1+sign_extend(imm12)

mem[(address+1):address] ← rs2[15:0]

Execute permissions:

M mode/S mode/U mode

Exception:

Memory Instruction Non-Aligned Access Exception, Memory Instruction Access Error Exception, Memory Instruction Page Error Exception

Command Format:

Chapter 14 Appendix A

Standardized Directive

Terminology

31	25 24	20 19	15 14	12 11	7 6	0
imm12[11:5]	rs2	rs1	001	imm12[4:0]	0100011	

14.1.41 SLL - Logical Left Shift Instruction

Grammar:

sll rd, rs1, rs2

Operation:

$rd \leftarrow rs1 \ll rs2[5:0]$

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
0000000	rs2	rs1	001	rd	0110011	

14.1.42 SLLI - Immediate Logical Left Shift Instruction

Grammar:

slli rd, rs1, sham6

Operation:

$rd \leftarrow rs1 \ll sham6$

Execute permissions:

M mode/S mode/U mode

Exception:

Chapter 14 Appendix A

Standardized Directive

Terminology

not have

Command Format:

31	26 25	20 19	15 14	12 11	7 6	0
000000	shamt6	rs1	001	rd	0010011	

14.1.43 SLLIW - Low 32-bit Immediate Logical Left Shift Instruction

Grammar:

slliw rd, rs1, sham5

Operation:

$\text{tmp}[31:0] \leftarrow (\text{rs1}[31:0] \ll \text{shamt5}[31:0])$

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

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
0000000	shamt5	rs1	001	rd	0011011	

14.1.44 SLLW - Low 32-bit Logic Left Shift Instruction

Grammar:

sllw rd, rs1, rs2

Operation:

$\text{tmp}[31:0] \leftarrow (\text{rs1}[31:0] \ll \text{rs2}[4:0])[31:0]$

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

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Command Format:

Chapter 14 Appendix A

Standardized Directive

Terminology

31	25 24	20 19	15 14	12 11	7 6	0
0000000	rs2	rs1	001	rd	0111011	

14.1.45 SLT - Signed Compare Less Than Placement Instruction

Grammar:

slt rd, rs1, rs2

Operation:

if (rs1 < rs2)

 rd←1

else

 rd←0

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
0000000	rs2	rs1	010	rd	0110011	

14.1.46 SLTI - Signed Immediate Number Compare Less Than Placement Instruction

Grammar:

slti rd, rs1, imm12

Operation:

if (rs1 < sign_extend(imm12))

 rd←1

else

 rd←0

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Chapter 14 Appendix A

Standardized Directive

Terminology Command Format:

31	20 19	15 14	12	11	7	6	0
imm12[11:0]	rs1	010		rd		0010011	

14.1.47 SLTIU - Unsigned Immediate Number Compare Less Than Placement Instruction

Grammar:

sltiu rd, rs1, imm12

Operation:

```
if (rs1 < zero_extend(imm12))
```

```
    rd←1
```

```
else
```

```
    rd←0
```

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Command Format:

31	20 19	15 14	12	11	7	6	0
imm12[11:0]	rs1	011		rd		0010011	

14.1.48 SLTU - unsigned compare less than set instruction

Grammar:

sltu rd, rs1, rs2

Operation:

```
if (rs1 < rs2)
```

```
    rd←1
```

```
else
```

```
    rd←0
```

Execute permissions:

M mode/S mode/U mode

Exception:

Chapter 14 Appendix A

Standardized Directive

Terminology

not have

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
0000000	rs2	rs1	011	rd	0110011	

14.1.49 SRA - arithmetic right shift instruction

Grammar:

sra rd, rs1, rs2

Operation:

$rd \leftarrow rs1 >>> rs2[5:0]$

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
0100000	rs2	rs1	101	rd	0110011	

14.1.50 SRAI - Immediate Number Arithmetic Right Shift Instruction

Grammar:

srai rd, rs1, sham6

Operation:

$rd \leftarrow rs1 >>> sham6$

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Command Format:

31	26 25	20 19	15 14	12 11	7 6	0
010000	sham6	rs1	101	rd	0010011	

Chapter 14 Appendix A

Standardized Directive

14.1.51 SRAIW - Low 32-bit Immediate Count Arithmetic Right Shift

Instruction

Grammar:

sraiw rd, rs1, shamt5

Operation:

$\text{tmp}[31:0] \leftarrow (\text{rs1}[31:0] \ggg \text{shamt5})[31:0]$

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

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
0100000	shamt5	rs1	101	rd	0011011	

14.1.52 SRAW - low 32-bit arithmetic right shift instruction

Grammar:

sraw rd, rs1, rs2

Operation:

$\text{tmp} \leftarrow (\text{rs1}[31:0] \ggg \text{rs2}[4:0])[31:0]$

$\text{rd} \leftarrow \text{sign_extend}(\text{tmp})$

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
0100000	rs2	rs1	101	rd	0111011	

14.1.53 SRET - Superuser mode exception return instruction

Grammar:

Chapter 14 Appendix A

Standardized Directive

Terminology

Operation:

`next pc ← sepc`

`sstatus.sie ← sstatus.spie`

`sstatus.spie ← 1`

Execute permissions:

S mode

Exception:

Illegal instruction exception

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
0001000	00010	00000	000	00000	1110011	

14.1.54 SRL - Logical Right Shift Instruction

Grammar:

`srl rd, rs1, rs2`

Operation:

`rd ← rs1 >> rs2[5:0]`

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
0000000	rs2	rs1	101	rd		0110011

14.1.55 SRLI - Immediate Logical Right Shift Instruction

Grammar:

`srlti rd, rs1, sham6`

Operation:

`rd ← rs1 >> sham6`

Chapter 14 Appendix A

Standardized Directive

Terminology

permissions:

M mode/S mode/U mode

Exception:

not have

Command Format:

31	26	25	20	19	15	14	12	11	7	6	0
000000		shamt6		rs1		101		rd		0010011	

14.1.56 SRLIW - Low 32-bit Immediate Logical Right Shift Instruction

Grammar:

srliw rd, rs1, sham5

Operation:

$\text{tmp}[31:0] \leftarrow (\text{rs1}[31:0] >> \text{shamt5})[31:0]$

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

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
0000000		shamt5		rs1		101		rd		0011011	

14.1.57 SRLW - low 32-bit logical right shift instruction

Grammar:

srlw rd, rs1, rs2

Operation:

$\text{tmp} \leftarrow (\text{rs1}[31:0] >> \text{rs2}[4:0])[31:0]$

$\text{rd} \leftarrow \text{sign_extend}(\text{tmp})$

Execute permissions:

M mode/S mode/U mode

Exception:

Chapter 14 Appendix A

Standardized Directive

Terminology

not have

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
0000000	rs2	rs1	101	rd	0111011	

14.1.58 SUB - Signed Subtraction Instruction

Grammar:

sub rd, rs1, rs2

Operation:

$rd \leftarrow rs1 - rs2$

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
0100000	rs2	rs1	000	rd	0110011	

14.1.59 SUBW--Lower 32-bit Signed Subtract Instruction

Grammar:

subw rd, rs1, rs2

Operation:

$tmp[31:0] \leftarrow rs1[31:0] - rs2[31:0]$

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

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
0100000	rs2	rs1	000	rd	0111011	

Chapter 14 Appendix A

Standardized Directive

14.1.60 SW - word storage instructions

Grammar:

sw rs2, imm12(rs1)

Operation:

address \leftarrow rs1+sign_extend(imm12)

mem[(address+3):address] \leftarrow rs2[31:0]

Execute permissions:

M mode/S mode/U mode

Exception:

Memory Instruction Non-Aligned Access Exception, Memory Instruction Access Error Exception, Memory Instruction Page Error Exception

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
imm12[11:5]	rs2	rs1	010	imm12[4:0]	0100011	

14.1.61 WFI - Enter Low Power Mode Instruction

Grammar:

wfi

Operation:

The processor enters a low-power mode where the CPU clock is turned off and most peripheral clocks are also turned off

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
0001000	00101	00000	000	00000	1110011	

14.1.62 XOR - per-bitwise-orthogonal-or instruction

Grammar:

xor rd, rs1, rs2

Chapter 14 Appendix A

Standardized Directive

Terminology:

$rd \leftarrow rs1 \wedge rs2$

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
0000000		rs2		rs1	100		rd		0110011		

14.1.63 XORI - Immediate Number by Bit Orders

Grammar:

xori rd, rs1, imm12

Operation:

$rd \leftarrow rs1 \& \text{sign_extend}(imm12)$

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Command Format:

31	20	19	15	14	12	11	7	6	0
imm12[11:0]		rs1	100		rd		0010011		

14.2 Appendix A-2 M Command Terminology

The following is a detailed description of the RISC-V M instruction set as implemented in the C910. The instructions in this section are 32 bits wide and are listed in alphabetical order.

14.2.1 DIV - Signed Division Instruction

Grammar:

div rd, rs1, rs2

Chapter 14 Appendix A

Standardized Directive

Terminology:

$rd \leftarrow rs1 / rs2$

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Description:

If the divisor is 0, the result of division is

0xffffffffffffffffffff...ffff

When overflow is generated, the division result is 0x8000000000000000

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
0000001		rs2		rs1		100		rd		0110011	

14.2.2 DIVU - Unsigned Division Instruction

Grammar:

divu rd, rs1, rs2

Operation:

$rd \leftarrow rs1 / rs2$

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Description:

If the divisor is 0, the result of division is

0xffffffffffffffffffff...ffff

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
0000001		rs2		rs1		101		rd		0110011	

14.2.3 DIVUW - low 32-bit unsigned divide instruction

Grammar:

divuw rd, rs1, rs2

Chapter 14 Appendix A

Standardized Directive

Terminology:

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

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

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Description:

If the divisor is 0, the result of division is

0xffffffffffffffffffffffffffffffffffff...ffff

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
0000001		rs2		rs1		101		rd		0111011	

14.2.4 DIVW - Low 32-bit Signed Division Instruction

Grammar:

divw rd, rs1, rs2

Operation:

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

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

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Description:

If the divisor is 0, the result of division is

0xffffffffffffffffffff...ffff

When overflow is generated, the division result is 0xffffffff80000000

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
0000001		rs2		rs1		100		rd		0111011	

Chapter 14 Appendix A

Standardized Directive

14.2.5 MUL - Signed Multiplication Instruction

Grammar:

mul rd, rs1, rs2

Operation:

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

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
0000001	rs2	rs1	000	rd	0110011	

14.2.6 MULH - Signed Multiplication Take Higher Instruction

Grammar:

mulh rd, rs1, rs2

Operation:

$rd \leftarrow (rs1 * rs2)[127:64]$

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
0000001	rs2	rs1	001	rd	0110011	

14.2.7 MULHSU - Signed Unsigned Multiplication Take Higher Instruction

Grammar:

mulusu rd, rs1, rs2

Operation:

Chapter 14 Appendix A

Standardized Directive

Terminology
 $rd \leftarrow (rs1 * rs2)[127:64]$

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Description:

rs1 signed numbers, rs2 unsigned numbers

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
0000001	rs2	rs1	010	rd	0110011	

14.2.8 MULHU - Unsigned Multiply Take Higher Instruction

Grammar:

mulhu rd, rs1, rs2

Operation:

$rd \leftarrow (rs1 * rs2)[127:64]$

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
0000001	rs2	rs1	011	rd	0110011	

14.2.9 MULW - Low 32-bit Signed Multiply Instruction

Grammar:

mulw rd, rs1, rs2

Operation:

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

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

Execute permissions:

Chapter 14 Appendix A

Standardized Directive

Terminology

M mode/S mode/U mode

Exception:

not have

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
0000001	rs2	rs1	000	rd	0111011	

14.2.10 REM - signed remainder instruction

Grammar:

rem rd, rs1, rs2

Operation:

$rd \leftarrow rs1 \% rs2$

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Description:

When the divisor is 0, the result of the remainder is the divisor, and when overflow is generated, the result of the remainder is in the 0x0

instruction format:

31	25 24	20 19	15 14	12 11	7 6	0
0000001	rs2	rs1	110	rd	0110011	

14.2.11 REMU - unsigned remainder instruction

Grammar:

remu rd, rs1, rs2

Operation:

$rd \leftarrow rs1 \% rs2$

Execute permissions:

M mode/S mode/U mode

Chapter 14 Appendix A

Standardized Directive

Termination:

Chapter 14 Appendix A

Standardized Directive

Terminology

not have

Description:

When the divisor is 0, the result of the remainder is the divisor.

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
0000001	rs2	rs1	111	rd	0110011	

14.2.12 REMUW - low 32-bit unsigned remainder instruction

Grammar:

remw rd, rs1, rs2

Operation:

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

$\text{rd} \leftarrow \text{sign_extend}(\text{tmp})$

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Description:

When the divisor is 0, the result is the result of sign bit expansion of [31] bits of the divisor.

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
0000001	rs2	rs1	111	rd	0111011	

14.2.13 REMW - low 32-bit signed remainder instruction

Grammar:

remw rd, rs1, rs2

Operation:

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

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

Execute permissions:

M mode/S mode/U mode

Chapter 14 Appendix A

Standardized Directive

Terminology:

not have

Description:

When the divisor is 0, the remainder result is the result of expanding the sign bit of the divisor by [31] bits, and when overflow occurs, the remainder result is 0x0.

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
0000001		rs2		rs1		110		rd		0111011	

14.3 Appendix A-3 A Command Terminology

The following is a specific description of the RISC-V A instructions implemented in the C910. The instructions in this section are 32 bits wide and are listed in alphabetical order.

14.3.1 AMOADD.D - Atomic Addition Instruction

Grammar:

amoadd.d.aqrl rd, rs2, (rs1)

Operation:

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

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

Execute permissions:

M mode/S mode/U mode

Exception:

Atomic Instruction Non-Aligned Access Exception, Atomic Instruction Access Error Exception, Atomic Instruction Page Error Exception

Impact Flag Bits:

not have

Description:

The aq bit and the rl bit determine the order in which the instruction's preorder and postorder memory access instructions are executed:

- aq=0,rl=0: corresponding assembly instruction amoadd.d rd, rs2, (rs1).

Chapter 14 Appendix A

Standardized Directive

Terminology
 $aq=0,rl=1$: Corresponding assembly instruction amoadd.d.rl rd, rs2, (rs1), the result of all instructions accessing the store in the preamble of this instruction must be observed before this instruction is executed.

- $aq=1,rl=0$: Corresponding assembly instruction amoadd.d.aq rd, rs2, (rs1), which post sequences all instructions accessing the store must wait for the execution of this instruction to complete before they begin execution.

Chapter 14 Appendix A

Standardized Directive

Terminology

aq=1,rl=1: Corresponding assembly instruction amoadd.d.aqrl rd, rs2, (rs1), the result of all instructions accessing storage in the preamble of this instruction must be observed before this instruction, and the execution of all instructions accessing storage in the postamble of this instruction must wait until the execution of this instruction is complete before execution begins.

Command Format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
00000	aq	rl		rs2		rs1		011		rd		0101111	

14.3.2 AMOADD.W - low 32-bit atomic addition instruction

Grammar:

amoadd.w.aqrl rd, rs2, (rs1)

Operation:

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

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

Execute permissions:

M mode/S mode/U mode

Exception:

Atomic Instruction Non-Aligned Access Exception, Atomic Instruction Access Error Exception, Atomic Instruction Page Error Exception

Impact Flag Bits:

not have

Description:

The aq bit and the rl bit determine the order in which the instruction's preorder and postorder memory access instructions are executed:

- aq=0,rl=0: corresponding assembly instruction amoadd.w rd, rs2, (rs1).
- aq=0,rl=1: Corresponding assembly instruction amoadd.w.rl rd, rs2, (rs1), the result of all instructions accessing the store in the preamble of this instruction must be observed before this instruction is executed.
- aq=1,rl=0: Corresponding assembly instruction amoadd.w.aq rd, rs2, (rs1), which post sequences all instructions accessing the store must wait for the execution of this instruction to complete before they begin execution.
- aq=1,rl=1: Corresponding assembly instruction amoadd.w.aqrl rd, rs2, (rs1), the result of all instructions accessing storage in the preamble of this instruction must be observed before this instruction, and the execution of all instructions accessing storage in the postamble of this instruction must wait for the completion of the

Chapter 14 Appendix A

Standardized Directive

Terminology execution of this instruction before execution begins.

Command Format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
00000	aq	rl		rs2		rs1	010		rd		0101111		

Chapter 14 Appendix A

Standardized Directive

14.3.3 AMOAND.D - Atomic By Bits and Directives

Grammar:

amoand.d.aqrl rd, rs2, (rs1)

Operation:

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

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

Execution authority: M mode/S

mode/U mode

Exception:

Atomic Instruction Non-Aligned Access Exception, Atomic Instruction Access Error Exception,
Atomic Instruction Page Error Exception

Impact Flag Bits:

not have

Description:

The aq bit and the rl bit determine the order in which the instruction's preorder and postorder memory access instructions are executed:

- aq=0,rl=0: corresponding assembly instructions amoand.d rd, rs2, (rs1).
- aq=0,rl=1: Corresponding assembly instruction amoand.d.rl rd, rs2, (rs1), which precedes all accesses to stored instructions whose results must be observed before the instruction is executed.
- aq=1,rl=0: Corresponding assembly instruction amoand.d.aq rd, rs2, (rs1), which post sequences all instructions accessing the store must wait for the execution of this instruction to complete before they begin execution.
- aq=1,rl=1: Corresponding assembly instruction amoand.d.aqrl rd, rs2, (rs1), the result of all instructions accessing the store in the preamble of this instruction must be observed before this instruction, and the execution of all instructions accessing the store in the postamble of this instruction must wait until the execution of this instruction is complete before execution begins.

Command Format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
01100	aq	rl		rs2		rs1	011		rd		0101111		

14.3.4 AMOAND.W - Low 32-bit Atomic By Bit with Instructions

Grammar:

amoand.w.aqrl rd, rs2, (rs1)

Chapter 14 Appendix A

Standardized Directive

Termination:

```
rd ← sign_extend(mem[rs1+3: rs1])  
mem[rs1+3:rs1] ← mem[rs1+3:rs1] & rs2[31:0]
```

Chapter 14 Appendix A

Standardized Directive

Terminology Execute permissions:

M mode/S mode/U mode

Exception:

Atomic Instruction Non-Aligned Access Exception, Atomic Instruction Access Error Exception, Atomic Instruction Page Error Exception

Impact Flag Bits:

not have

Description:

The aq bit and the rl bit determine the order in which the instruction's preorder and postorder memory access instructions are executed:

- aq=0,rl=0: corresponding assembly instruction amoand.w rd, rs2, (rs1).
- aq=0,rl=1: Corresponding assembly instruction amoand.w.rl rd, rs2, (rs1), the result of all instructions accessing the store in the preamble of this instruction must be observed before this instruction is executed.
- aq=1,rl=0: Corresponding assembly instruction amoand.w.aq rd, rs2, (rs1), which post sequences all instructions accessing the storage must wait for the execution of this instruction to be completed before starting execution.
- aq=1,rl=1: Corresponding assembly instruction amoand.w.aql rd, rs2, (rs1), the result of all instructions accessing the store in the preamble of this instruction must be observed before this instruction, and the execution of all instructions accessing the store in the postamble of this instruction must wait until the execution of this instruction is complete before execution begins.

Command Format:

31	27	26	25	24	20 19	15 14	12 11	7 6	0
01100	aq	rl		rs2		rs1	010	rd	0101111

14.3.5 AMOMAX.D - Atomic Signed Maximum Instruction

Grammar:

amomax.d.aql rd, rs2, (rs1)

Operation:

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

$\text{mem}[rs1+7: rs1] \leftarrow \max(\text{mem}[rs1+7: rs1], rs2)$

Execute permissions:

M mode/S mode/U mode

Exception:

Chapter 14 Appendix A

Standardized Directive

Terminology: Instruction Non-Aligned Access Exception, Atomic Instruction Access Error Exception,

Atomic Instruction Page Error Exception

Impact Flag Bits:

not have

Chapter 14 Appendix A

Standardized Directive

Terminology Description:

The aq bit and the rl bit determine the order in which the instruction's preorder and postorder memory access instructions are executed:

- aq=0,rl=0: corresponding assembly instruction amomax.d rd, rs2, (rs1).
- aq=0,rl=1: Corresponding assembly instruction amomax.d.rl rd, rs2, (rs1), the result of all instructions accessing the store in the preamble of this instruction must be observed before this instruction is executed.
- aq=1,rl=0: Corresponding assembly instruction amomax.d.aq rd, rs2, (rs1), which post sequences all instructions accessing the store must wait for the execution of this instruction to complete before they begin execution.
- aq=1,rl=1: Corresponding assembly instruction amomax.d.aql rd, rs2, (rs1), the result of all instructions accessing the store in the preamble of this instruction must be observed before this instruction, and all instructions accessing the store in the postamble of this instruction must wait until the execution of this instruction is complete before execution begins.

Command Format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
10100	aq	rl		rs2		rs1		011		rd		0101111	

14.3.6 AMOMAX.W - Low 32-bit Atomic Signed Maximum Instruction

Grammar:

amomax.w.aql rd, rs2, (rs1)

Operation:

```
rd ← sign_extend( mem[rs1+3: rs1] )
mem[rs1+3:rs1] ← max(mem[rs1+3:rs1],
rs2[31:0])
```

Execute permissions:

M mode/S mode/U mode

Exception:

Atomic Instruction Non-Aligned Access Exception, Atomic Instruction Access Error Exception, Atomic Instruction Page Error Exception

Impact Flag Bits:

not have

Description:

The aq bit and the rl bit determine the order in which the instruction's preorder and postorder

Chapter 14 Appendix A

Standardized Directive

Terminology access instructions are executed:

- aq=0,rl=0: corresponding assembly instruction amomax.w rd, rs2, (rs1).
- aq=0,rl=1: Corresponding assembly instruction amomax.w.rl rd, rs2, (rs1), the result of all instructions accessing the store in the preamble of this instruction must be observed before this instruction is executed.

Chapter 14 Appendix A

Standardized Directive

Terminology

aq=1,rl=0: Corresponding assembly instruction `amomax.w.aq rd, rs2, (rs1)`, which post sequences all instructions accessing the store must wait for the execution of this instruction to complete before they begin execution.

- **aq=1,rl=1:** Corresponding assembly instruction `amomax.w.aql rd, rs2, (rs1)`, the result of all instructions accessing the store in the preamble of this instruction must be observed before this instruction, and all instructions accessing the store in the postamble of this instruction must wait until the execution of this instruction is complete before execution begins.

Command Format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
10100	aq	rl		rs2		rs1	010		rd		0101111		

14.3.7 AMOMAXU.D - Atomic Unsigned Maximization Instruction

Grammar:

`amomaxu.d.aqrl rd, rs2, (rs1)`

Operation:

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

$\text{mem}[rs1+7:rs1] \leftarrow \max(\text{mem}[rs1+7:rs1], rs2)$

Execute permissions:

M mode/S mode/U mode

Exception:

Atomic Instruction Non-Aligned Access Exception, Atomic Instruction Access Error Exception, Atomic Instruction Page Error Exception

Impact flag bits:

not have

Description:

The aq bit and the rl bit determine the order in which the instruction's preorder and postorder memory access instructions are executed:

- **aq=0,rl=0:** corresponding assembly instructions `amomaxu.d rd, rs2, (rs1)`.
- **aq=0,rl=1:** the corresponding assembly instruction `amomaxu.d.rl rd, rs2, (rs1)`, which presequences all accesses to stored instructions whose results must be observed before the instruction is executed.
- **aq=1,rl=0:** the corresponding assembly instruction `amomaxu.d.aq rd, rs2, (rs1)`, which post sequences all instructions accessing the store must wait for the completion of the execution of this instruction before they begin execution.
- **aq=1,rl=1:** the corresponding assembly instruction `amomaxu.d.aql rd, rs2, (rs1)`, the

Chapter 14 Appendix A

Standardized Directive

Terminology result of all instructions accessing storage in the preorder of this instruction must be observed before this instruction, and the execution of all instructions accessing storage in the postorder of this instruction must wait until the execution of this instruction is complete before execution begins.

Command Format:

Chapter 14 Appendix A

Standardized Directive

Terminology

31	27	26	25	24	20	19	15	14	12	11	7	6	0
11100	aq	rl			rs2		rs1		011		rd		0101111

14.3.8 AMOMAXU.W - Low 32-bit Atomic Unsigned Maximizer Instruction

Grammar:

amomaxu.w.aqrl rd, rs2, (rs1)

Operation:

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

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

Execute permissions:

M mode/S mode/U mode

Exception:

Atomic Instruction Non-Aligned Access Exception, Atomic Instruction Access Error Exception, Atomic Instruction Page Error Exception

Impact Flag Bits:

not have

Description:

The aq bit and the rl bit determine the order in which the instruction's preorder and postorder memory access instructions are executed:

- aq=0,rl=0: corresponding assembly instruction amomaxu.w rd, rs2, (rs1).
- aq=0,rl=1: the corresponding assembly instruction amomaxu.w.rl rd, rs2, (rs1), which presequences all accesses to stored instructions whose results must be observed before the instruction is executed.
- aq=1,rl=0: the corresponding assembly instruction amomaxu.w.aq rd, rs2, (rs1), which post sequences all instructions accessing the store must wait for the completion of the execution of this instruction before they begin execution.
- aq=1,rl=1: the corresponding assembly instruction amomaxu.w.aqrl rd, rs2, (rs1), the result of all instructions accessing the store in the preorder of the instruction must be observed before the execution of the instruction, and all instructions accessing the store in the postorder of the instruction must wait for the completion of the execution of the instruction before execution begins.

Command Format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
11100	aq	rl			rs2		rs1		010		rd		0101111

Chapter 14 Appendix A

Standardized Directive

TRN.D AMOMIN.D - Atomic Signed Minimum Instruction

Grammar:

amomin.d.aqrl rd, rs2, (rs1)

Chapter 14 Appendix A

Standardized Directive

Terminology:

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

$\text{mem}[rs1+7:rs1] \leftarrow \min(\text{mem}[rs1+7:rs1], rs2)$

Execute permissions:

M mode/S mode/U mode

Exception:

Atomic Instruction Non-Aligned Access Exception, Atomic Instruction Access Error Exception,
Atomic Instruction Page Error Exception

Impact Flag Bits:

not have

Description:

The aq bit and the rl bit determine the order in which the instruction's preorder and postorder memory access instructions are executed:

- aq=0,rl=0: corresponding assembly instructions `amomin.d rd, rs2, (rs1)`.
- aq=0,rl=1: the corresponding assembly instruction `amomin.d.rl rd, rs2, (rs1)`, which presequences all accesses to stored instructions whose results must be observed before the instruction is executed.
- aq=1,rl=0: the corresponding assembly instruction `amomin.d.aq rd, rs2, (rs1)`, which post sequences all instructions accessing the store must wait for the completion of the execution of this instruction before they begin execution.
- aq=1,rl=1: the corresponding assembly instruction `amomin.d.aql rd, rs2, (rs1)`, the result of all instructions accessing the store in the preorder of the instruction must be observed before the execution of that instruction, and all instructions accessing the store in the postorder of the instruction must wait for the completion of the execution of that instruction before execution begins.

Command Format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
10000	aq	rl		rs2		rs1	011		rd		0101111		

14.3.10 AMOMIN.W - Low 32-bit Atomic Signed Minimum Instruction

Grammar:

`amomin.w.aql rd, rs2, (rs1)`

Operation:

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

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

Chapter 14 Appendix A

Standardized Directive

Terminology

M mode/S mode/U mode

Exception:

Chapter 14 Appendix A

Standardized Directive

Terminology

Atomic Instruction Non-Aligned Access Exception, Atomic Instruction Access Error Exception, Atomic Instruction Page Error Exception

Impact Flag Bits:

not have

Description:

The aq bit and the rl bit determine the order in which the instruction's preorder and postorder memory access instructions are executed:

- aq=0,rl=0: corresponding assembly instructions amomin.w rd, rs2, (rs1).
- aq=0,rl=1: the corresponding assembly instruction amomin.w.rl rd, rs2, (rs1), which precedes all accesses to stored instructions whose results must be observed before the instruction is executed.
- aq=1,rl=0: the corresponding assembly instruction amomin.w.aq rd, rs2, (rs1), which post sequences all instructions accessing the store must wait for the completion of the execution of this instruction before they begin execution.
- aq=1,rl=1: the corresponding assembly instruction amomin.w.aql rd, rs2, (rs1), the result of all instructions accessing the store in the instruction presequence must be observed before the execution of that instruction, and all instructions accessing the store in the instruction postsequence must wait for the completion of the execution of that instruction before execution begins.

Command Format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
10000	aq	rl		rs2		rs1		010		rd		0101111	

14.3.11 AMOMINU.D - Atomic Unsigned Minimization Instruction

Grammar:

amominu.d.aql rd, rs2, (rs1)

Operation:

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

$\text{mem}[rs1+7: rs1] \leftarrow \min(\text{mem}[rs1+7: rs1], rs2)$

Execute permissions:

M mode/S mode/U mode

Exception:

Atomic Instruction Non-Aligned Access Exception, Atomic Instruction Access Error Exception, Atomic Instruction Page Error Exception

Impact Flag Bits:

Chapter 14 Appendix A

Standardized Directive

Terminology

Description:

The aq bit and the rl bit determine the order in which the instruction's preorder and postorder memory access instructions are executed:

- aq=0,rl=0: corresponding assembly instructions amominu.d rd, rs2, (rs1).

Chapter 14 Appendix A

Standardized Directive

Terminology

$aq=0, rl=1$: the corresponding assembly instruction `amominu.d.rl rd, rs2, (rs1)`, which precedes all accesses to stored instructions whose results must be observed before the instruction is executed.

- $aq=1, rl=0$: the corresponding assembly instruction `amominu.d.aq rd, rs2, (rs1)`, which post sequences all instructions accessing the store must wait for the completion of the execution of this instruction before they begin execution.
- $aq=1, rl=1$: the corresponding assembly instruction `amominu.d.aql rd, rs2, (rs1)`, the result of all instructions accessing the store in the preorder of the instruction must be observed before the execution of that instruction, and all instructions accessing the store in the postorder of the instruction must wait for the completion of the execution of that instruction before execution begins.

Command Format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
11000	aq	rl		rs2		rs1	011		rd		0101111		

14.3.12 AMOMINU.W - Low 32-bit Atomic Unsigned Minimum Value

Instruction

Grammar:

`amominu.w.aql rd, rs2, (rs1)`

Operation:

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

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

Execute permissions:

M mode/S mode/U mode

Exception:

Atomic Instruction Non-Aligned Access Exception, Atomic Instruction Access Error Exception,

Atomic Instruction Page Error Exception

Impact Flag Bits:

not have

Description:

The `aq` bit and the `rl` bit determine the order in which the instruction's preorder and postorder memory access instructions are executed:

- $aq=0, rl=0$: corresponding assembly instruction `amominu.w rd, rs2, (rs1)`.
- $aq=0, rl=1$: the corresponding assembly instruction `amominu.w.rl rd, rs2, (rs1)`, which presequences all accesses to stored instructions whose results must be observed before the instruction is executed.

Chapter 14 Appendix A

Standardized Directive

Terminology $aq=1, rl=0$: the corresponding assembly instruction `amominu.w.aq rd, rs2, (rs1)`, which post sequences all instructions accessing the store must wait for the completion of the execution of this instruction before they begin execution.

- $aq=1, rl=1$: the corresponding assembly instruction `amominu.w.aql rd, rs2, (rs1)`, the result of all instructions accessing the store in the instruction presequence must be observed before the execution of that instruction, and all instructions accessing the store in the instruction postsequence must wait for the completion of the execution of that instruction before execution begins.

Chapter 14 Appendix A

Standardized Directive

Terminology

Command Format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
11000	aq	rl		rs2		rs1		010		rd		0101111	

14.3.13 AMOOR.D - Atomic By Bit or Command

Grammar:

amoor.d.aqrl rd, rs2, (rs1)

Operation:

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

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

Execute permissions:

M mode/S mode/U mode

Exception:

Atomic Instruction Non-Aligned Access Exception, Atomic Instruction Access Error Exception, Atomic Instruction Page Error Exception

Impact Flag Bits:

not have

Description:

The aq bit and the rl bit determine the order in which the instruction's preorder and postorder memory access instructions are executed:

- aq=0,rl=0: corresponding assembly instructions amoor.d rd, rs2, (rs1).
- aq=0,rl=1: the corresponding assembly instruction amoor.d.rl rd, rs2, (rs1), which presequences all accesses to stored instructions whose results must be observed before the instruction is executed.
- aq=1,rl=0: the corresponding assembly instruction amoor.d.aq rd, rs2, (rs1), which post sequences all instructions accessing the store must wait for the execution of this instruction to be completed before starting execution.
- aq=1,rl=1: the corresponding assembly instruction amoor.d.aqrl rd, rs2, (rs1), the result of all instructions accessing the store in the instruction presequence must be observed before the execution of that instruction, and all instructions accessing the store in the instruction postsequence must wait for the completion of the execution of that instruction before execution begins.

Command Format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
01000	aq	rl		rs2		rs1		011		rd		0101111	

Chapter 14 Appendix A

Standardized Directive

~~T43n10logy~~ MOOR.W - low 32-bit atomic per-bit or instruction

Grammar:

Chapter 14 Appendix A

Standardized Directive

Terminology

amoor.w.aqrl rd, rs2, (rs1)

Operation:

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

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

Execute permissions:

M mode/S mode/U mode

Exception:

Atomic Instruction Non-Aligned Access Exception, Atomic Instruction Access Error Exception, Atomic Instruction Page Error Exception

Impact Flag Bits:

not have

Description:

The aq bit and the rl bit determine the order in which the instruction's preorder and postorder memory access instructions are executed:

- aq=0,rl=0: corresponding assembly instruction amoor.w rd, rs2, (rs1).
- aq=0,rl=1: the corresponding assembly instruction amoor.w.rl rd, rs2, (rs1), which presequences all accesses to stored instructions whose results must be observed before the instruction is executed.
- aq=1,rl=0: the corresponding assembly instruction amoor.w.aq rd, rs2, (rs1), which post sequences all instructions accessing the store must wait for the execution of this instruction to be completed before starting execution.
- aq=1,rl=1: the corresponding assembly instruction amoor.w.aqrl rd, rs2, (rs1), the result of all instructions accessing the store in the instruction presequence must be observed before the execution of that instruction, and all instructions accessing the store in the instruction postsequence must wait for the completion of the execution of that instruction before execution begins.

Command Format:

31	27	26	25	24	20 19	15 14	12 11	7 6	0
01000	aq	rl		rs2		rs1	010	rd	0101111

14.3.15 AMOSWAP.D - Atomic Exchange Directive

Grammar:

amoswap.d.aqrl rd, rs2, (rs1)

Operation:

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

Chapter 14 Appendix A

Standardized Directive
Terminology $\text{mem}[rs1+7:rs1] \leftarrow rs2$

Execute permissions:

M mode/S mode/U mode

Chapter 14 Appendix A

Standardized Directive

Terminology:

Atomic Instruction Non-Aligned Access Exception, Atomic Instruction Access Error Exception,
Atomic Instruction Page Error Exception

Influence flag bit:

None

Description:

The aq bit and the rl bit determine the order in which the instruction's preorder and postorder memory access instructions are executed:

- aq=0,rl=0: corresponding assembly instructions amoswap.d rd, rs2, (rs1).
- aq=0,rl=1: the corresponding assembly instruction amoswap.d.rl rd, rs2, (rs1), which presequences all accesses to stored instructions whose results must be observed before the instruction is executed.
- aq=1,rl=0: the corresponding assembly instruction amoswap.d.aq rd, rs2, (rs1), which post sequences all instructions accessing the store must wait for the execution of this instruction to be completed before starting execution.
- aq=1,rl=1: the corresponding assembly instruction amoswap.d.aql rd, rs2, (rs1), the result of all instructions accessing the store in the preorder of the instruction must be observed before the execution of that instruction, and all instructions accessing the store in the postorder of the instruction must wait for the completion of the execution of that instruction before execution begins.

Command Format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
00001	aq	rl		rs2		rs1		011		rd		0101111	

14.3.16 AMOSWAP.W - Low 32-bit Atomic Swap Instruction

Grammar:

amoswap.w.aql rd, rs2, (rs1)

Operation:

```
rd ← sign_extend( mem[rs1+3: rs1] )
mem[rs1+3:rs1] ← rs2[31:0]
```

Execute permissions:

M mode/S mode/U mode

Exception:

Atomic Instruction Non-Aligned Access Exception, Atomic Instruction Access Error Exception,
Atomic Instruction Page Error Exception

Chapter 14 Appendix A

Standardized Directive

Influence flag bit:

None

Description:

The aq bit and the rl bit determine the order in which the instruction's preorder and postorder memory access instructions are executed:

- aq=0,rl=0: corresponding assembly instruction amoswap.w rd, rs2, (rs1).

Chapter 14 Appendix A

Standardized Directive

Terminology

$\text{aq}=0, \text{rl}=1$: the corresponding assembly instruction `amoswap.w.rl rd, rs2, (rs1)`, which presequences all accesses to stored instructions whose results must be observed before the instruction is executed.

- $\text{aq}=1, \text{rl}=0$: the corresponding assembly instruction `amoswap.w.aq rd, rs2, (rs1)`, which post sequences all instructions accessing the storage must wait for the completion of the execution of this instruction before they begin execution.
- $\text{aq}=1, \text{rl}=1$: the corresponding assembly instruction `amoswap.w.aql rd, rs2, (rs1)`, the result of all instructions accessing the store in the preorder of the instruction must be observed before the execution of that instruction, and all instructions accessing the store in the postorder of the instruction must wait for the completion of the execution of that instruction before execution begins.

Command Format:

31	27	26	25	24	20 19	15	14	12 11	7 6	0
00001	aq	rl		rs2		rs1	010		rd	0101111

14.3.17 AMOXOR.D - Atomic By Bit Alteration Instruction

Grammar:

`amoxor.d.aql rd, rs2, (rs1)`

Operation:

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

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

Execute permissions:

M mode/S mode/U mode

Exception:

Atomic Instruction Non-Aligned Access Exception, Atomic Instruction Access Error Exception, Atomic Instruction Page Error Exception

Impact Flag Bits:

not have

Description:

The `aq` bit and the `rl` bit determine the order in which the instruction's preorder and postorder memory access instructions are executed:

- $\text{aq}=0, \text{rl}=0$: corresponding assembly instructions `amoxor.d rd, rs2, (rs1)`.
- $\text{aq}=0, \text{rl}=1$: the corresponding assembly instruction `amoxor.d.rl rd, rs2, (rs1)`, which presequences all accesses to stored instructions whose results must be observed before the instruction is executed.
- $\text{aq}=1, \text{rl}=0$: the corresponding assembly instruction `amoxor.d.aq rd, rs2, (rs1)`, which

Chapter 14 Appendix A

Standardized Directive

Terminology Post sequences all instructions accessing the storage must wait for the execution of this instruction to be completed before starting execution.

- aq=1,rl=1: the corresponding assembly instruction amoxor.d.aqrl rd, rs2, (rs1), the result of all instructions accessing the store in the preorder of the instruction must be observed before the execution of the instruction, and all instructions accessing the store in the postorder of the instruction must wait for the completion of the execution of the instruction before they begin to execute.

Chapter 14 Appendix A

Standardized Directive

Terminology Command Format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
00100	aq	rl		rs2		rs1		011		rd		0101111	

14.3.18 AMOXOR.W - Low 32-bit Atomic Per Bit Isothermal Instruction

Grammar:

amoxor.w.aqrl rd, rs2, (rs1)

Operation:

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

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

Execute permissions:

M mode/S mode/U mode

Exception:

Atomic Instruction Non-Aligned Access Exception, Atomic Instruction Access Error Exception, Atomic Instruction Page Error Exception

Impact Flag Bits:

not have

Description:

The aq bit and the rl bit determine the order in which the instruction's preorder and postorder memory access instructions are executed:

- aq=0,rl=0: corresponding assembly instruction amoxor.w rd, rs2, (rs1).
- aq=0,rl=1: the corresponding assembly instruction amoxor.w.rl rd, rs2, (rs1), which presequences all accesses to stored instructions whose results must be observed before the instruction is executed.
- aq=1,rl=0: the corresponding assembly instruction amoxor.w.aq rd, rs2, (rs1), all instructions accessing storage in the post sequence of this instruction must wait for the execution of this instruction to complete before starting execution.
- aq=1,rl=1: the corresponding assembly instruction amoxor.w.aqrl rd, rs2, (rs1), the result of all instructions accessing the store in the instruction presequence must be observed before the execution of that instruction, and all instructions accessing the store in the instruction postsequence must wait until the execution of that instruction is complete before execution begins.

Command Format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
00100	aq	rl		rs2		rs1		010		rd		0101111	

Chapter 14 Appendix A

Standardized Directive

~~T4R19logy~~R.D - Double Word Load Reserved Instruction

Grammar:

Chapter 14 Appendix A

Standardized Directive

Terminology: lr.d.aqr rd, (rs1)

Operation:

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

$\text{mem}[rs1+7:rs1]$ is reserved

Execute permissions:

M mode/S mode/U mode

Exception:

Atomic Instruction Non-Aligned Access Exception, Atomic Instruction Access Error Exception,
Atomic Instruction Page Error Exception

Impact Flag Bits:

not have

Description:

The aq bit and the rl bit determine the order in which the instruction's preorder and postorder memory access instructions are executed:

- aq=0,rl=0: corresponding assembly instruction lr.d rd, (rs1).
- aq=0,rl=1: the corresponding assembly instruction lr.d.rl rd, (rs1), which presequences all accesses to stored instructions whose results must be observed before the instruction is executed.
- aq=1,rl=0: the corresponding assembly instruction lr.d.aq rd, (rs1), which post sequences all instructions accessing the store must wait for the completion of the execution of this instruction before they begin execution.
- aq=1,rl=1: the corresponding assembly instruction lr.d.aql rd, (rs1), the results of all instructions accessing storage in the instruction's presequence must be observed before the instruction is executed, and all instructions accessing storage in the instruction's postsequence must wait for the execution of the instruction to complete before execution begins.

Command Format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
00010	aq	rl		00000		rs1		011		rd		0101111	

14.3.20 LR.W - Word Load Reserved Instructions

Grammar:

lr.w.aql rd, (rs1)

Operation:

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

Chapter 14 Appendix A

Standardized Directive

mem[rs1+3:rs1] is reserved

Terminology

Execute permissions:

M mode/S mode/U mode

Chapter 14 Appendix A

Standardized Directive

Exception: Atomic Instruction Non-Aligned Access Exception, Atomic Instruction Access Error Exception, Atomic Instruction Page Error Exception

Influence flag bit: None

Description: The aq bit and the rl bit determine the order in which the instruction's preamble and postamble memory access instructions are executed:

- aq=0,rl=0: corresponding assembly instruction lr.w rd, (rs1).
- aq=0,rl=1: the corresponding assembly instruction lr.w.rl rd, (rs1), which presequences all accesses to stored instructions whose results must be observed before the instruction is executed.
- aq=1,rl=0: the corresponding assembly instruction lr.w.aq rd, (rs1), which post sequences all instructions accessing the store must wait for the completion of the execution of this instruction before they begin execution.
- aq=1,rl=1: the corresponding assembly instruction lr.w.aql rd, (rs1), the results of all instructions accessing the store in the instruction's preamble must be observed prior to the execution of that instruction, and all instructions accessing the store in the instruction's postamble must wait until the execution of that instruction is complete before execution begins.

Command Format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
00010	aq	rl		00000		rs1		010		rd		0101111	

14.3.21 SC.D - Double Word Conditional Storage Instruction

Grammar:

sc.d.aql rd, rs2, (rs1)

Operation:

If(mem[rs1+7:rs1] is reserved)

mem[rs1+7: rs1] ← rs2

rd ← 0

else

rd ← 1

Execute permissions:

M mode/S mode/U mode

Exception:

Atomic Instruction Non-Aligned Access Exception, Atomic Instruction Access Error Exception, Atomic Instruction Page Error Exception

Chapter 14 Appendix A

Standardized Directive

Terminology Flag Bits:

not have

Description:

The aq bit and the rl bit determine the order in which the instruction's preorder and postorder memory access instructions are executed:

Chapter 14 Appendix A

Standardized Directive

Terminology: $aq=0, rl=0$: corresponding assembly instruction sc.d rd, rs2, (rs1).

- $aq=0, rl=1$: the corresponding assembly instruction sc.d.rl rd, rs2, (rs1), which presequences all accesses to stored instructions whose results must be observed before the instruction is executed.
- $aq=1, rl=0$: the corresponding assembly instruction sc.d.aq rd, rs2, (rs1), which post sequences all instructions accessing the store must wait for the completion of the execution of this instruction before they begin execution.
- $aq=1, rl=1$: the corresponding assembly instruction sc.d.aql rd, rs2, (rs1), the result of all instructions accessing the store in the instruction presequence must be observed before the execution of that instruction, and all instructions accessing the store in the instruction postsequence must wait for the completion of the execution of that instruction before execution begins.

Command Format:

31	27	26	25	24	20 19	15 14	12 11	7 6	0
00011	aq	rl		rs2		rs1	011	rd	0101111

14.3.22 SC.W - Word Conditional Storage Instruction

Grammar:

sc.w.aql rd, rs2, (rs1)

Operation:

if(mem[rs1+3:rs1] is reserved)

mem[rs1+3:rs1] \leftarrow rs2[31:0]

rd \leftarrow 0

else

rd \leftarrow 1

Execute permissions:

M mode/S mode/U mode

Exception:

Atomic Instruction Non-Aligned Access Exception, Atomic Instruction Access Error Exception,
Atomic Instruction Page Error Exception

Impact Flag Bits:

not have

Description:

The aq bit and the rl bit determine the order in which the instruction's preorder and postorder memory access instructions are executed:

Chapter 14 Appendix A

Standardized Directive

Terminology aq=0,rl=0: corresponding assembly instruction sc.w rd, rs2, (rs1).

- aq=0,rl=1: the corresponding assembly instruction sc.w.rl rd, rs2, (rs1), which precedes all accesses to stored instructions whose results must be observed before the instruction is executed.

Chapter 14 Appendix A

Standardized Directive

Terminology $aq=1, rl=0$: the corresponding assembly instruction sc.w.aq rd, rs2, (rs1), which post sequences all instructions accessing the store must wait for the execution of this instruction to be completed before starting execution.

- $aq=1, rl=1$: the corresponding assembly instruction sc.w.aqrl rd, rs2, (rs1), the result of all instructions accessing storage in the instruction's presequence must be observed before the instruction is executed, and all instructions accessing storage in the instruction's postsequence must wait for the execution of the instruction to complete before execution begins.

Command Format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
00011	aq	rl		rs2		rs1	010		rd		0101111		

14.4 Appendix A-4 F Instruction Terminology

The following is a detailed description of the RISC-V F instruction set as implemented in the C910. The instructions in this section are 32 bits wide and are listed in alphabetical order.

For single-precision floating-point instructions, if the upper 32 bits of the source register are not all ones, the single-precision data is treated as qNaN.

When mstatus.fs==2^b00, execution of all instructions in this section generates an illegal instruction exception, and when mstatus.fs != 2^b00, mstatus.fs is set to 2^b11 after executing any instruction in this section.

14.4.1 FADD.S - Single precision floating point addition instruction

Grammar:

fadd.s fd, fs1, fs2, rm

Operation:

frd \leftarrow fs1 + fs2

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Impact Flag Bits:

Floating-point status bits NV/OF/NX

Description:

Chapter 14 Appendix A

Standardized Directive

Terminology determines the rounding mode.

- 3[▼]b000: rounds to an even number, corresponding to the assembly instruction fadd.s fd, fs1,fs2,rne.
- 3[▼]b001: Rounding to zero, corresponding to the assembly instruction fadd.s fd, fs1,fs2,rtz.

Chapter 14 Appendix A

Standardized Directive

Terminology

b_{00} : Rounding to negative infinity, corresponding to the assembly instruction fadd.s

fd, fs1,fs2,rdn.

- $b_{01}^{\blacktriangleright}$: Rounding to positive infinity, corresponding to the assembly instruction fadd.s fd, fs1,fs2,rup.
- $b_{10}^{\blacktriangleright}$: Round to the nearest larger value, corresponding to the assembly instruction fadd.s fd, fs1,fs2,rmm.
- $b_{11}^{\blacktriangleright}$: Not used at this time, this code will not appear.
- $b_{100}^{\blacktriangleright}$: Not used at this time, this code will not appear.
- $b_{111}^{\blacktriangleright}$: Dynamic rounding, determines the rounding mode based on the rm bit in the floating-point control register fcsr, corresponding to the assembly instructions fadd.s fd, fs1,fs2.

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
0000000		fs2		fs1		rm		fd		1010011	

14.4.2 FCLASS.S - Single-precision Floating Point Classification Instruction

Syntax:

fclass.s 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 ← 64▼h20
if ( fs1 = +norm)
    rd ← 64▼h40

```

Chapter 14 Appendix A

Standardized Directive

Terminology if(fs1 = +Inf)

rd ← 64'b80

Chapter 14 Appendix A

Standardized Directive

Terminology

if (fs1 = qNaN)

rd ← 64[▼]h100

if (fs1 = qNaN)

rd ← 64[▼]h200

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal

Instruction

Exceptionally

Affected

Flag Bit:

None

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
1110000		00000		fs1		001		rd		1010011	

14.4.3 FCVT.L.S - Single-precision Floating-point to Signed Long Integer Instructions

Grammar:

fcvt.l.s rd, fs1, rm

Operation:

rd ← single_convert_to_signed_long(fs1)

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Impact Flag Bits:

Floating Point Status Bit NV/NX

Description:

rm determines the rounding mode.

- 3[▼]b000: Rounding to an even number, corresponding to the assembly instruction fcvt.l.s rd,fs1,rne.

Chapter 14 Appendix A

Standardized Directive

Terminology: 1001: Rounding to zero, corresponding to the assembly instruction fcvt.l.s rd,fs1,rtz.

- 3 b010: Rounding to negative infinity, corresponding to the assembly instruction fcvt.l.s rd,fs1,rdn.

Chapter 14 Appendix A

Standardized Directive

Terminology

$3^{\text{v}}b011$: Rounding to positive infinity, corresponding to the assembly instruction `fcvt.ls rd,fs1,rup.`

- $3^{\text{v}}b100$: Round to the nearest large value, corresponding to the assembly instruction `fcvt.ls rd,fs1,rmm.`
- $3^{\text{v}}b101$: Not used at this time, this code will not appear.
- $3^{\text{v}}b110$: Not used at this time, this code will not appear.
- $3^{\text{v}}b111$: Dynamic rounding, which determines the rounding mode based on the rm bit in the floating-point control register fcsr, corresponding to the assembly instruction `fcvt.ls rd, fs1.`

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
1100000		00010		fs1		rm		rd		1010011	

14.4.4 FCVT.LU.S - Single-precision Floating-point to Unsigned Long Integer Instructions

Grammar:

`fcvt.lu.s rd, fs1, rm`

Operation:

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

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Impact Flag Bits:

Floating Point Status Bit NV/NX

Description:

rm determines the rounding mode.

- $3^{\text{v}}b000$: rounds to an even number, corresponding to the assembly instruction `fcvt.lu.s rd,fs1,rne.`
- $3^{\text{v}}b001$: Rounding to zero, corresponding to the assembly instruction `fcvt.lu.s rd,fs1,rtz.`
- $3^{\text{v}}b010$: Rounding to negative infinity, corresponding to the assembly instruction `fcvt.lu.s rd,fs1,rdn.`
- $3^{\text{v}}b011$: Rounding to positive infinity, corresponding to the assembly instruction `fcvt.lu.s rd,fs1,rup.`

Chapter 14 Appendix A

Standardized Directive

Terminology: Round to the nearest larger value, corresponding to the assembly instruction

fcvt.lu.s rd,fs1,rmm.

- 3⁷b101: Not used at this time, this code will not appear.
- 3⁷b110: Not used at this time, this code will not appear.

Chapter 14 Appendix A

Standardized Directive

Terminology

Dynamic rounding: Dynamic rounding, which determines the rounding mode based on the rm bit in the floating-point control register fcsr, corresponding to the assembly instruction fcvt.lu.s rd, fs1.

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
1100000		00011		fs1		rm		rd		1010011	

14.4.5 FCVT.S.L - Signed Long Integer to Single Precision Floating Point Instruction

Grammar:

fcvt.s.l fd, rs1, rm

Operation:

fd \leftarrow signed_long_convert_to_single(fs1)

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal

Instruction

Exception

Impact Flag

Bit: Floating

Point Status

Bit NX

Description:

rm determines the rounding mode.

- 3[▼]b000: Rounding to an even number as close as possible, corresponding to the assembly instruction fcvt.s.l fd,rs1,rne.
- 3[▼]b001: Rounding to zero, corresponding to the assembly instruction fcvt.s.l fd,rs1,rtz.
- 3[▼]b010: Rounding to negative infinity, corresponding to the assembly instruction fcvt.s.l fd,fs1,rdn.
- 3[▼]b011: Rounding to positive infinity, corresponding to the assembly instruction fcvt.s.l fd,fs1,rup.
- 3[▼]b100: Round to the nearest large value, corresponding to the assembly instruction fcvt.s.l fd,fs1,rmm.

Chapter 14 Appendix A

Standardized Directive

Terminology: Not used at this time, this code will not appear.

- 3'b110: Not used at this time, this code will not appear.
- 3'b111: Dynamic rounding, which determines the rounding mode based on the rm bit in the floating-point control register fcsr, corresponding to the assembly instruction fcvt.s.l fd, fs1.

Command Format:

31	25:24	20:19	15:14	12:11	7:6	0
1101000	00010	rs1	rm	fd	1010011	

Chapter 14 Appendix A

Standardized Directive

Terminology: FCVT.S.LU - Unsigned Long Integer to Single Precision Floating Point Instruction

Grammar:

fcvt.s.l fd, fs1, rm

Operation:

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

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal

Instruction

Exception

Impact Flag

Bit: Floating

Point Status

Bit NX

Description:

rm determines the rounding mode.

- 3[▼]b00: Rounding to an even number, corresponding to the assembly instruction fcvt.s.lu fd,fs1,rne.
- 3[▼]b001: Rounding to zero, corresponding to the assembly instruction fcvt.s.lu fd, fs1,rtz.
- 3[▼]b010: Rounding to negative infinity, corresponding to the assembly instruction fcvt.s.lu fd, fs1,rdn.
- 3[▼]b011: Rounding to positive infinity, corresponding to the assembly instruction fcvt.s.lu fd, fs1,rup.
- 3[▼]b100: Rounding to large values as close as possible, corresponding to the assembly instruction fcvt.s.lu fd, fs1,rmm.
- 3[▼]b101: Not used at this time, this code will not appear.
- 3[▼]b110: Not used at this time, this code will not appear.
- 3[▼]b111: Dynamic rounding, which determines the rounding mode based on the rm bit in the floating-point control register fcsr, corresponding to the assembly instruction fcvt.s.lu fd, fs1.

Command Format:

31	25 24	20 19	15 14	12 11	7	6	0
www.t-l	1101000	00011	rs1	rm	fd	1010011	

Chapter 14 Appendix A

Standardized Directive

Terminology

14.4.7 FCVT.S.W - Signed Integer to Single Precision Floating Point Instruction

Grammar:

fcvt.s.w fd, rs1, rm

Operation:

Chapter 14 Appendix A

Standardized Directive

Terminology

`fd ← signed_int_convert_to_single(fs1)`

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal

Instruction

Exception

Impact Flag

Bit: Floating

Point Status

Bit NX

Description:

rm determines the rounding mode.

- 3[▼]b000: Rounding to an even number as close as possible, corresponding to the assembly instruction `fcvt.s.w fd,rs1,rne`.
- 3[▼]b001: Rounding to zero, corresponding to the assembly instruction `fcvt.s.w fd,rs1,rtz`.
- 3[▼]b010: Rounding to negative infinity, corresponding to the assembly instruction `fcvt.s.w fd,rs1,rdn`.
- 3[▼]b011: Rounding to positive infinity, corresponding to the assembly instruction `fcvt.s.w fd,rs1,rup`.
- 3[▼]b100: Rounding to large values as close as possible, corresponding to the assembly instruction `fcvt.s.w fd,rs1,rmm`.
- 3[▼]b101: Not used at this time, this code will not appear.
- 3[▼]b110: Not used at this time, this code will not appear.
- 3[▼]b111: Dynamic rounding, which determines the rounding mode based on the rm bit in the floating-point control register fcsr, corresponding to the assembly instruction `fcvt.s.w fd, rs1`.

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
1101000	00000	rs1	rm	fd	1010011	

14.4.8 FCVT.S.WU - Unsigned Integer to Single Precision Floating Point Instruction

Grammar:

Chapter 14 Appendix A

Standardized Directive

fcvt.s.wu fd, rs1, rm

Terminology

Operation:

fd ← unsigned_int_convert_to_single_fp(fs1)

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Chapter 14 Appendix A

Standardized Directive

Terminology

~~Impact Flag~~

Bit: Floating

Point Status

Bit NX

Description:

rm determines the rounding mode.

- 3[▼]b000: Rounding to an even number as close as possible, corresponding to the assembly instruction fcvt.s.wu fd,rs1,rne.
- 3[▼]b001: Rounding to zero, corresponding to the assembly instruction fcvt.s.wu fd,rs1,rtz.
- 3[▼]b010: Rounding to negative infinity, corresponding to the assembly instruction fcvt.s.wu fd,rs1,rdn.
- 3[▼]b011: Rounding to positive infinity, corresponding to the assembly instruction fcvt.s.wu fd,rs1,rup.
- 3[▼]b100: Rounding to large values as close as possible, corresponding to the assembly instruction fcvt.s.wu fd,rs1,rmm.
- 3[▼]b101: Not used at this time, this code will not appear.
- 3[▼]b110: Not used at this time, this code will not appear.
- 3[▼]b111: Dynamic rounding, which determines the rounding mode based on the rm bit in the floating-point control register fcsr, corresponding to the assembly instruction fcvt.s.wu fd, rs1.

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
1101000	00001	rs1	rm	fd	1010011	

14.4.9 FCVT.W.S - Single Precision Floating Point to Signed Integer Instructions

Grammar:

fcvt.w.s rd, fs1, rm

Operation:

```
tmp ← single_convert_to_signed_int(fs1)
rd←sign_extend(tmp)
```

Execute permissions:

M mode/S mode/U mode

Exception:

Chapter 14 Appendix A

Standardized Directive

Terminology

Impact Flag Bits:

Floating Point Status Bit NV/NX

Description:

rm determines the rounding mode.

Chapter 14 Appendix A

Standardized Directive

Terminology: 000: Rounding to an even number, corresponding to the assembly instruction

fcvt.w.s rd,fs1,rne.

- 3[▼]b001: Rounding to zero, corresponding to the assembly instruction fcvt.w.s rd,fs1,rtz.
- 3[▼]b010: Rounding to negative infinity, corresponding to the assembly instruction fcvt.w.s rd,fs1,rdn.
- 3[▼]b011: Rounding to positive infinity, corresponding to the assembly instruction fcvt.w.s rd,fs1,rup.
- 3[▼]b100: Round to the nearest large value, corresponding to the assembly instruction fcvt.w.s rd,fs1,rmm.
- 3[▼]b101: Not used at this time, this code will not appear.
- 3[▼]b110: Not used at this time, this code will not appear.
- 3[▼]b111: Dynamic rounding, which determines the rounding mode based on the rm bit in the floating-point control register fcsr, corresponding to the assembly instruction fcvt.w.s rd, fs1.

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
1100000		00000		fs1		rm		rd		1010011	

14.4.10 FCVT.WU.S - Single-precision Floating Point to Unsigned Integer Instructions

Grammar:

fcvt.wu.s rd, fs1, rm

Operation:

```
tmp ← single_convert_to_unsigned_int(fs1)
rd←sign_extend(tmp)
```

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Impact Flag Bits:

Floating Point Status Bit NV/NX

Description:

rm determines the rounding mode.

Chapter 14 Appendix A

Standardized Directive

Terminology: 0000: rounds to an even number, corresponding to the assembly instruction fcvt.wu.s

rd,fs1,rne.

- 3[▼]b001: Rounding to zero, corresponding to the assembly instruction fcvt.wu.s rd,fs1,rtz.
- 3[▼]b010: Rounding to negative infinity, corresponding to the assembly instruction fcvt.wu.s rd,fs1,rdn.
- 3[▼]b011: Rounding to positive infinity, corresponding to the assembly instruction fcvt.wu.s rd,fs1,rup.

Chapter 14 Appendix A

Standardized Directive

Terminology: 00: Round to the nearest larger value, corresponding to the assembly instruction

fcvt.wu.s rd,fs1,rmm.

- 3^vb101: Not used at this time, this code will not appear.
- 3^vb110: Not used at this time, this code will not appear.
- 3^vb111: Dynamic rounding, which determines the rounding mode based on the rm bit in the floating-point control register fcsr, corresponding to the assembly instruction fcvt.wu.s rd, fs1.

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
1100000		00001		fs1		rm		rd		1010011	

14.4.11 FDIV.S - Single Precision Floating Point Division Instruction

Grammar:

fdiv.s fd, fs1, fs2, rm

Operation:

fd \leftarrow fs1 / fs2

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Impact Flag Bits:

Floating-point status bits NV/DZ/OF/UF/NX

Description:

rm determines the rounding mode.

- 3^vb000: r o u n d s to an even number, corresponding to the assembly instruction fdiv.s fs1,fs2,rne.
- 3^vb001: Rounding to zero, corresponding to the assembly instruction fdiv.s fd fs1,fs2,rtz.
- 3^vb010: Rounding to negative infinity, corresponding to the assembly instruction fdiv.s fd, fs1,fs2,rdn.
- 3^vb011: Rounding to positive infinity, corresponding to the assembly instruction fdiv.s fd, fs1,fs2,rup.
- 3^vb100: Round to the nearest larger value, corresponding to the assembly instruction fdiv.s fd, fs1,fs2,rmm.
- 3^vb101: Not used at this time, this code will not appear.

Chapter 14 Appendix A

Standardized Directive

Terminology: Not used at this time, this code will not appear.

- 3'b111: Dynamic rounding, determines the rounding mode based on the rm bit in the floating-point control register fcsr, corresponding to the assembly instructions fdiv.s fd, fs1,fs2.

Chapter 14 Appendix A

Standardized Directive

Terminology Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
0001100	fs1	fs2	rm	fd	1010011	

14.4.12 FEQ.S - Single precision floating point compare equal instruction

Grammar:

feq.s rd, fs1, fs2

Operation:

if(fs1 == fs2)

rd \leftarrow 1

else

rd \leftarrow 0

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal

Instruction

Exception

Affect Flag

Bit: Floating

Point Status

Bit NV

Instruction

Format:

31	25 24	20 19	15 14	12 11	7 6	0
1010000	fs2	fs1	010	rd	1010011	

14.4.13 FLE.S - Single precision floating point compare less than or equal to instruction

Grammar:

fle.s rd, fs1, fs2

Operation:

if(fs1 \leq fs2)

Chapter 14 Appendix A

Standardized Directive

Terminology^{rd ← 1}

else

rd ← 0

Execute permissions:

Chapter 14 Appendix A

Standardized Directive

Terminology

M mode/S mode/U mode

Exception:

Illegal

Instruction

Exception

Affect Flag

Bit: Floating

Point Status

Bit NV

Instruction

Format:

31	25 24	20 19	15 14	12 11	7 6	0
1010000	fs2	fs1	000	rd	1010011	

14.4.14 FLT.S - Single precision floating point compare less than instruction

Grammar:

flt.s rd, fs1, fs2

Operation:

if(fs1 < fs2)

rd ← 1

else

rd ← 0

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal

Instruction

Exception

Affect Flag

31	25 24	20 19	15 14	12 11	7 6	0
1010000	fs2	fs1	001	rd	1010011	

Bit: Floating

Point Status

Chapter 14 Appendix A

Standardized Directive

Terminology

Instruction

Format:

14.4.15 FLW - Single precision floating point load instruction

Grammar:

flw fd, imm12(rs1)

Operation:

Chapter 14 Appendix A

Standardized Directive

Terminology

$\text{address} \leftarrow \text{rs1} + \text{sign_extend}(\text{imm12})$

$\text{fd}[31:0] \leftarrow \text{mem}[(\text{address}+3):\text{address}]$

$\text{fd}[63:32] \leftarrow 32^{\blacktriangleright} \text{h}ffffffffff$

Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception, Illegal Instruction Exception

Impact Flag Bits:

not have

Command Format:

31	20 19	15 14	12 11	7 6	0
imm12[11:0]	rs1	010	fd	0000111	

14.4.16 FMADD.S - Single Precision Floating Point Multiply Accumulate Instruction

Grammar:

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

Operation:

$\text{rd} \leftarrow \text{fs1} * \text{fs2} + \text{fs3}$

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Impact Flag Bits:

Floating-point status bits NV/OF/UF/IX

Description:

rm determines the rounding mode.

- 3 \blacktriangleright b00: Rounding to an even number, corresponding to the assembly instruction
fmadd.s fd,fs1, fs2, fs3, rne.
- 3 \blacktriangleright b001: Rounding to zero, corresponding to the assembly instruction fmadd.s fd,fs1, fs2, fs3,
rtz.

Chapter 14 Appendix A

Standardized Directive

Terminology b010: Rounding to negative infinity, corresponding to the assembly instruction

fmadd.s fd,fs1, fs2, fs3, rdn.

- 3 b011: Rounding to positive infinity, corresponding to the assembly instruction fmadd.s
fd,fs1, fs2, fs3, rup.

Chapter 14 Appendix A

Standardized Directive

Terminology: Round to the nearest larger value, corresponding to the assembly instruction

fmadd.s fd,fs1, fs2, fs3, rmm.

- 3[▼]b101: Not used at this time, this code will not appear.
- 3[▼]b110: Not used at this time, this code will not appear.
- 3[▼]b111: Dynamic rounding, depending on the rm bit in the floating-point control register fcsr, determines the rounding mode, corresponding to the assembly instructions fmadd.s fd,fs1, fs2, fs3.

Command Format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
fs3	00		fs2		fs1		rm		fd		1000011		

14.4.17 FMAX.S - Single precision floating point take maximum instruction

Grammar:

fmax.s fd, fs1, fs2

Operation:

if(fs1 >= fs2)

 fd ← fs1

else

 fd ← fs2

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal

Instruction

Exception

Affect Flag

Bit: Floating

Point Status

Bit NV

Instruction

Format:

31	25	24	20	19	15	14	12	11	7	6	0
0010100		fs2		fs1	001		fd		1010011		

Chapter 14 Appendix A

Standardized Directive

T4.118 logMIN.S - Single precision floating point take minimum instruction**Grammar:**

fmin.s fd, fs1, fs2

Operation:

Chapter 14 Appendix A

Standardized Directive

Terminology

fd \leftarrow fs2

else

fd \leftarrow fs1

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal

Instruction

Exception

Affect Flag

Bit: Floating

Point Status

Bit NV

Instruction

Format:

31	25	24	20	19	15	14	12	11	7	6	0
0010100		fs2		fs1		000		fd		1010011	

14.4.19 FMSUB.S - Single precision floating point multiply-accumulate instruction

Grammar:

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

Operation:

fd \leftarrow fs1 * fs2 - fs3

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Impact Flag Bits:

Floating-point status bits NV/OF/UF/IX

Description:

Chapter 14 Appendix A

Standardized Directive

Terminology determines the rounding mode.

- 3[▼]b000: Rounding to an even number, corresponding to the assembly instruction
fmsub.s fd,fs1, fs2, fs3, rne.
- 3[▼]b001: Rounding to zero, corresponding to assembly instructions fmsub.s fd,fs1, fs2, fs3, rtz.
- 3[▼]b010: Rounding to negative infinity, corresponding to the assembly instruction
fmsub.s fd,fs1, fs2, fs3, rdn.



Standardized Directive

Terminology

- 3^vb011: Rounding to positive infinity, corresponding to assembly instructions fmsub.s fd,fs1, fs2, fs3, rup.
- 3^vb100: Rounding to the nearest larger value, corresponding to the assembly instruction fmsub.s fd, fs1, fs2, fs3,rmm.
- 3^vb101: Not used at this time, this code will not appear.
- 3^vb110: Not used at this time, this code will not appear.
- 3^vb111: Dynamic rounding, which determines the rounding mode based on the rm bit in the floating-point control register fcsr. The corresponding assembly refers to to make fmsub.s fd,fs1, fs2, fs3.

Command Format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
fs3	00			fs2		fs1		rm		fd		1000111	

14.4.20 FMUL.S - Single precision floating point multiply instruction

Grammar:

fmul.s fd, fs1, fs2, rm

Operation:

fd \leftarrow fs1 * fs2

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Impact Flag Bits:

Floating-point status bits NV/OF/UF/NX

Description:

rm determines the rounding mode.

- 3^vb000: Rounding to an even number, corresponding to the assembly instruction fmuls. fd, fs1, fs2, rne.
- 3^vb001: Rounding to zero, corresponding to the assembly instruction fmuls. fd, fs1, fs2, rtz.
- 3^vb010: Rounding to negative infinity, corresponding to the assembly instruction fmuls. fd, fs1, fs2, rdn.
- 3^vb011: Rounding to positive infinity, corresponding to the assembly instruction fmuls. fd, fs1, fs2, rup.

Chapter 14 Appendix A

Standardized Directive

Terminology 100: Round to the nearest larger value, corresponding to the assembly instruction

fmul.s fd, fs1,fs2, rmm.

- 3[▼]b101: Not used at this time, this code will not appear.
- 3[▼]b110: Not used at this time, this code will not appear.

Chapter 14 Appendix A

Standardized Directive

Terminology

Dynamic rounding: Dynamic rounding, determines the rounding mode based on the rm bit in the floating-point control register fcsr, corresponding to the assembly instruction `fmul.s fs1,fs2`.

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
0001000		fs2		fs1		rm		fd		1010011	

14.4.21 FMV.W.X - Single-Precision Floating-Point Write Transfer Instruction

Syntax:

`fmv.w.x fd, rs1`

Operation:

$fd[31:0] \leftarrow rs[31:0]$

$fd[63:32] \leftarrow 32^{\text{h}}\text{fffff}ff$

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal

Instruction

Exceptionally

Affected

Flag Bit:

None

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
1111000		00000		rs1		000		fd		1010011	

14.4.22 FMV.X.W - Single-Precision Floating-Point Register Read Transfer Instruction

Syntax:

`fmv.x.w rd, fs1`

Operation:

$tmp[31:0] \leftarrow fs1[31:0]$

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

Chapter 14 Appendix A

Standardized Directive

Terminology permissions:

M mode/S mode/U mode

Exception:

Chapter 14 Appendix A

Standardized Directive

Terminology

Instruction

Exceptionally

Affected

Flag Bit:

None

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
1110000		00000		fs1		000		rd		1010011	

14.4.23 FNMADD.S - single precision floating point multiply accumulate take negative instruction

Grammar:

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

Operation:

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

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Impact flag bits:

Floating-point status bits NV/OF/UF/IX

Description:

rm determines the rounding mode.

- 3[▼]b000: Rounding to an even number, corresponding to the assembly instruction
fnmadd.s fd,fs1, fs2, fs3, rne.
- 3[▼]b001: Rounding to zero, corresponding to the assembly instruction fnmadd.s fd,fs1, fs2, fs3, rtz.
- 3[▼]b010: Rounding to negative infinity, corresponding to the assembly instruction
fnmadd.s fd,fs1, fs2, fs3, rdn.
- 3[▼]b011: Rounding to positive infinity, corresponding to the assembly instruction fnmadd.s
fd,fs1, fs2, fs3, rup.
- 3[▼]b100: Round to the nearest larger value, corresponding to the assembly instruction

Chapter 14 Appendix A

Standardized Directive

Terminology: fadd.s fd,fs1, fs2, fs3, rmm.

- 3[▼]b101: Not used at this time, this code will not appear.
- 3[▼]b110: Not used at this time, this code will not appear.
- 3[▼]b111: Dynamic rounding, which determines the rounding mode based on the rm bit in the floating-point control register fcsr, corresponding to the assembly instructions fnmadd.s fd,fs1, fs2, fs3.

Command Format:

Chapter 14 Appendix A

Standardized Directive

Term	31	27	26	25	24	20	19	15	14	12	11	7	6	0
	fs3	00		fs2		fs1	100	rm		fd		1001111		

14.4.24 FNMSUB.S - Single-precision floating-point multiply-accumulate-take-negative instruction

Grammar:

fnmsub.s fd, fs1, fs2, fs3, rm

Operation:

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

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Impact Flag Bits:

Floating-point status bits NV/OF/UF/IX

Description:

rm determines the rounding mode.

- 3^vb000: Rounding to an even number, corresponding to the assembly instruction fnmsub.s fd,fs1, fs2, fs3, rne.
- 3^vb001: Rounding to zero, corresponding to the assembly instruction fnmsub.s fd,fs1, fs2, fs3, rtz.
- 3^vb010: Rounding to negative infinity, corresponding to the assembly instruction fnmsub.s fd,fs1, fs2, fs3, rdn.
- 3^vb011: Rounding to positive infinity, corresponding to the assembly instruction fnmsub.s fd,fs1, fs2, fs3, rup.
- 3^vb100: Rounding to the nearest larger value, corresponding to the assembly instruction fnmsub.s fd,fs1, fs2, fs3, rmm.
- 3^vb101: Not used at this time, this code will not appear.
- 3^vb110: Not used at this time, this code will not appear.
- 3^vb111: Dynamic rounding, which determines the rounding mode based on the rm bit in the floating-point control register fcsr, corresponding to the assembly instructions fnmsub.s fd,fs1, fs2, fs3.

Command Format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
fs3	00		fs2		fs1	100	rm		fd		1001011		

Chapter 14 Appendix A

Standardized Directive

Terminology

14.4.25 FSGNJ.S - Single-Precision Floating-Point Symbol Injection Instruction

Grammar:

Chapter 14 Appendix A

Standardized Directive

Terminology
fsgnjn.s fd, fs1, fs2

Operation:

$fd[30:0] \leftarrow fs1[30:0]$
 $fd[31] \leftarrow fs2[31]$
 $fd[63:32] \leftarrow 32^{\top} hffffffffff$

execute permission:

M mode/S mode/U mode

Exception:

Illegal

Instruction

Exceptionally

Affected

Flag Bit:

None

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
0010000		fs2		fs1	000		fd		1010011		

14.4.26 FSGNIN.S - Single-precision floating-point symbol fetch inverse injection instruction

Grammar:

fsgnjn.s fd, fs1, fs2

Operation:

$fd[30:0] \leftarrow fs1[30:0]$
 $fd[31] \leftarrow ! fs2[31]$
 $fd[63:32] \leftarrow 32^{\top} hffffffffff$

execute permission:

M mode/S mode/U mode

Exception:

Illegal

Instruction

Exceptionally

Chapter 14 Appendix A

Standardized Directive

TermAffected

Flag Bit:

None

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
0010000		fs2		fs1		001		fd		1010011	

Chapter 14 Appendix A

Standardized Directive

14.4.27 FSGNJX.S - Single-Precision Floating-Point Symbol Alias Injection Instruction

Grammar:

fsgnjx.s fd, fs1, fs2

Operation:

fd[30:0] ← fs1[30:0]

fd[31] ← fs1[31] ^ fs2[31]

fd[63:32] ← 32'bXXXXXXXXXX

Execute permission:

M mode/S mode/U mode

Exception:

Illegal

Instruction

Exceptionally

Affected

Flag Bit:

None

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
0010000		fs2		fs1		010		fd		1010011	

14.4.28 FSQRT.S - Single precision floating point open square instruction

Grammar:

fsqrt.s fd, fs1, rm

Operation:

fd ← sqrt(fs1)

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Impact Flag Bits:

Floating Point Status Bit NV/NX

Chapter 14 Appendix A

Standardized Directive

Terminology:

rm determines the rounding mode.

Chapter 14 Appendix A

Standardized Directive

Terminology

Round to even numbers, corresponding to the assembly instruction fsqrt.s fd, fs1,rne

- 3^vb000: Rounding to zero, corresponding to the assembly instruction fsqrt.s fd, fs1,rtz
- 3^vb010: Rounding to negative infinity, corresponding to the assembly instruction fsqrt.s fd, fs1,rdn
- 3^vb011: Rounding to positive infinity, corresponding to the assembly instruction fsqrt.s fd, fs1,rup
- 3^vb100: round to the nearest large value, corresponding to the assembly instruction fsqrt.s fd, fs1,rmm
- 3^vb101: Not used at this time, this code will not appear.
- 3^vb110: Not used at this time, this code will not appear.
- 3^vb111: Dynamic rounding, which determines the rounding mode based on the rm bit in the floating-point control register fcsr, corresponding to the assembly instruction fsqrt.s fd, fs1.

Command Format:

31	25	24	20 19	15 14	12 11	7 6	0
0101100		00000	fs1	rm	fd		1010011

14.4.29 FSUB.S - Single precision floating point subtraction instruction

Grammar:

fsub.s fd, fs1, fs2, rm

Operation:

fd \leftarrow fs1 - fs2

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Impact flag bits:

Floating-point status bits NV/OF/NX

Description:

rm determines the rounding mode.

- 3^vb000: rounds to an even number, corresponding to assembly instructions fsub.fd, fs1,fs2,rne
- 3^vb001: Rounding to zero, corresponding to assembly instruction fsub.s fd, fs1,fs2,rtz
- 3^vb010: Rounding to negative infinity, corresponding to the assembly instruction fsub.s fd, fs1,fs2,rdn

Chapter 14 Appendix A

Standardized Directive

Terminology 011: Rounding to positive infinity, corresponding to the assembly instruction fsub.s fd, fs1,fs2,rup

- 3 b100: round to the nearest large value, corresponding to the assembly instruction fsub.s fd, fs1,fs2,rmm

Chapter 14 Appendix A

Standardized Directive

Terminology: 3¹⁰: Not used at this time, this code will not appear.

- 3¹¹: Not used at this time, this code will not appear.
- 3¹²: Dynamic rounding, determines the rounding mode based on the rm bit in the floating-point control register fcsr, corresponding to the assembly instructions fsub.s fd, fs1,fs2.

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
0000100	fs2	fs1	rm	fd	1010011	

14.4.30 FSW - Single precision floating point store instructions

Grammar:

fsw fs2, imm12(rs1)

Operation:

address←rs1+sign_extend(imm12)

mem[(address+31):address] ← fs2[31:0]

Execute permissions:

M mode/S mode/U mode

Exception:

Memory Instruction Non-Aligned Access Exception, Memory Instruction Access Error Exception, Memory Instruction Page Error Exception, Illegal Instruction Exception

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
imm12[11:5]	fs2	rs1	010	imm12[4:0]	0100111	

14.5 Appendix A-5 D Instruction Terminology

The following is a specific description of the RISC-V D instruction set as implemented in the C910. The instructions in this section are 32 bits wide and are listed in alphabetical order.

When mstatus.fs==2¹⁰, execution of all instructions in this section generates an illegal instruction exception, and when mstatus.fs != 2¹⁰, mstatus.fs is set to 2¹⁰ after executing any instruction in this section.

14.5.1 FADD.D - Double precision floating point addition instruction

Chapter 14 Appendix A

Standardized Directive

Terminology:

fadd.d fd, fs1, fs2, rm

Chapter 14 Appendix A

Standardized Directive

Terminology:

$fd \leftarrow fs1 + fs2$

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Impact Flag Bits:

Floating-point status bits NV/OF/NX

Description:

rm determines the rounding mode.

- 3[▼]b000: rounds to an even number, corresponding to the assembly instruction fadd.d fd, fs1,fs2,rne
- 3[▼]b001: Rounding to zero, corresponding to the assembly instruction fadd.d fd, fs1,fs2,rtz
- 3[▼]b010: Rounding to negative infinity, corresponding to the assembly instruction fadd.d fd, fs1,fs2,rdn
- 3[▼]b011: Rounding to positive infinity, corresponding to the assembly instruction fadd.d fd, fs1,fs2,rup
- 3[▼]b100: round to the nearest large value, corresponding to the assembly instruction fadd.d fd, fs1,fs2,rmm
- 3[▼]b101: Not used at this time, this code will not appear.
- 3[▼]b110: Not used at this time, this code will not appear.
- 3[▼]b111: Dynamic rounding, determines the rounding mode based on the rm bit in the floating-point control register fcsr, corresponding to the assembly instructions fadd.d fd, fs1,fs2.

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
0000001	fs2	fs1	rm	fd	1010011	

14.5.2 FCLASS.D - Double Precision Floating Point Classification Instruction

Grammar:

fclass.d rd, fs1

Operation: if (fs1 = -Inf)

$rd \leftarrow 64^{\blacktriangledown} h1$

if (fs1 = -norm)

$rd \leftarrow 64^{\blacktriangledown} h2$

Chapter 14 Appendix A

Standardized Directive

Terminology if ($fsl =$ -subnorm)

Chapter 14 Appendix A

Standardized Directive

Terminology

```

if ( fs1 = -zero)
    fd ← 64 ▶ h8
if ( fs1 = +Zero)
    rd ← 64 ▶ h10
if ( fs1 = +subnorm)
    rd ← 64 ▶ h20
if ( fs1 = +norm)
    rd ← 64 ▶ h40
if ( fs1 = +Inf)
    rd ← 64 ▶ h80
if ( fs1 = sNaN)
    rd ← 64 ▶ h100
if ( fs1 = qNaN)
    rd ← 64 ▶ h200

```

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal

Instruction

Exceptionally

Affected

Flag Bit:

None

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
1110001	00000	fs1	001	rd	1010011	

14.5.3 FCVT.D.L - Signed Long Integer to Double Precision Floating Point Instruction

Grammar:

fcvt.d.l fd, rs1, rm

Operation:

Chapter 14 Appendix A

Standardized Directive

Terminology fd ← signed_long_convert_to_double(fs1)

Chapter 14 Appendix A

Standardized Directive

Terminology permissions:

M mode/S mode/U mode

Exception:

Illegal

Instruction

Exception

Impact Flag

Bit: Floating

Point Status

Bit NX

Description:

rm determines the rounding mode.

- 3[▼]b000: Rounding to an even number as close as possible, corresponding to the assembly instruction fcvt.d.l fd,rs1,rne.
- 3[▼]b001: Rounding to zero, corresponding to the assembly instruction fcvt.d.l fd,rs1,rtz.
- 3[▼]b010: Rounding to negative infinity, corresponding to the assembly instruction fcvt.d.l fd,rs1,rdn.
- 3[▼]b011: Rounding to positive infinity, corresponding to the assembly instruction fcvt.d.l fd,rs1,rup.
- 3[▼]b100: Rounding to large values as close as possible, corresponding to the assembly instruction fcvt.d.l fd,rs1,rmm.
- 3[▼]b101: Not used at this time, this code will not appear.
- 3[▼]b110: Not used at this time, this code will not appear.
- 3[▼]b111: Dynamic rounding, which determines the rounding mode based on the rm bit in the floating-point control register fcsr, corresponding to the assembly instruction fcvt.d.l fd, rs1.

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
1101001	00010		rs1		rm		fd		1010011		

14.5.4 FCVT.D.LU - unsigned long integer to double precision floating point instruction

Grammar:

fcvt.d.lu fd, rs1, rm

Chapter 14 Appendix A

Standardized Directive

TerOpnigyn:

```
fd ← unsigned_long_convert_to_double(fs1)
```

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Impact Flag Bits:

Chapter 14 Appendix A

Standardized Directive

Floating Point Status Bit NX

Description:

rm determines the rounding mode.

- 3[▼]b000: Rounding to an even number as close as possible, corresponding to the assembly instruction fcvt.d.lu fd,rs1,rne.
- 3[▼]b001: Rounding to zero, corresponding to the assembly instruction fcvt.d.lu fd,rs1,rtz.
- 3[▼]b010: Rounding to negative infinity, corresponding to the assembly instruction fcvt.d.lu fd,rs1,rdn.
- 3[▼]b011: Rounding to positive infinity, corresponding to the assembly instruction fcvt.d.lu fd,rs1,rup.
- 3[▼]b100: Rounding to large values as close as possible, corresponding to the assembly instruction fcvt.d.lu fd,rs1,rmm.
- 3[▼]b101: Not used at this time, this code will not appear.
- 3[▼]b110: Not used at this time, this code will not appear.
- 3[▼]b111: Dynamic rounding, which determines the rounding mode based on the rm bit in the floating-point control register fcsr, corresponding to the assembly instruction fcvt.d.lu fd, rs1.

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
1101001		00011		rs1		rm		fd		1010011	

14.5.5 FCVT.D.S - Single Precision Floating Point to Double Precision Floating Point Instruction

Syntax:

fcvt.d.s fd, fs1

Operation:

fd \leftarrow single_convert_to_double(fs1)

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal

Instruction

Exceptionally

Chapter 14 Appendix A

Standardized Directive

TermAffected

Flag Bit:

None

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
0100001		00000		fs1		000		fd		1010011	

Chapter 14 Appendix A

Standardized Directive

~~14.5.6~~ FCVT.D.W - Signed Integer to Double Precision Floating Point

Instruction

Syntax:

fcvt.d.w fd, rs1

Operation:

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

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal

Instruction

Exceptionally

Affected

Flag Bit:

None

Command Format:

31	25	24	20 19	15 14	12 11	7 6	0
1101001	00000		rs1	000	fd	1010011	

14.5.7 FCVT.D.WU - unsigned integer to double precision floating point instruction

Syntax:

fcvt.d.wu fd, rs1

Operation:

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

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal

Instruction

Exceptionally

Affected

Chapter 14 Appendix A

Standardized Directive

TerFlagology Bit:

None

Command Format:

31	25 : 24	20 : 19	15 : 14	12 : 11	7 : 6	0
1101001	00001	rs1	000	fd	1010011	

Chapter 14 Appendix A

Standardized Directive

14.5.8 FCVT.L.D - Double Precision Floating Point to Signed Long Integer

Instruction

Grammar:

fcvt.l.d rd, fs1, rm

Operation:

`rd ← double_convert_to_signed_long(fs1)`

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Impact Flag Bits:

Floating Point Status Bit NV/NX

Description:

rm determines the rounding mode.

- 3[▼]b000: Rounding to an even number, corresponding to the assembly instruction `fcvt.l.d rd,fs1,rne.`
- 3[▼]b001: Rounding to zero, corresponding to the assembly instruction `fcvt.l.d rd,fs1,rtz.`
- 3[▼]b010: Rounding to negative infinity, corresponding to the assembly instruction `fcvt.l.d rd,fs1,rdn.`
- 3[▼]b011: Rounding to positive infinity, corresponding to the assembly instruction `fcvt.l.d rd,fs1,rup.`
- 3[▼]b100: Rounding to the nearest larger value, corresponding to the assembly instruction `fcvt.l.d rd,fs1,rmm.`
- 3[▼]b101: Not used at this time, this code will not appear.
- 3[▼]b110: Not used at this time, this code will not appear.
- 3[▼]b111: Dynamic rounding, which determines the rounding mode based on the rm bit in the floating-point control register fcsr, corresponding to the assembly instruction `fcvt.l.d rd, fs1.`

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
1100001		00010		fs1		rm		rd		1010011	

14.5.9 FCVT.LU.D - Double Precision Floating Point to Unsigned Long Integer

Chapter 14 Appendix A

Standardized Directive

Terminology

Grammar:

fcvt.lu.d rd, fs1, rm

Operation:

Chapter 14 Appendix A

Standardized Directive

`Terminology`
 $fd \leftarrow \text{double_convert_to_unsigned_long}(fs1)$

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Impact Flag Bits:

Floating Point Status Bit NV/NX

Description:

rm determines the rounding mode.

- 3[▼]b000: Rounding to an even number, corresponding to the assembly instruction `fcvt.lu.d rd,fs1,rne.`
- 3[▼]b001: Rounding to zero, corresponding to the assembly instruction `fcvt.lu.d rd,fs1,rtz.`
- 3[▼]b010: Rounding to negative infinity, corresponding to the assembly instruction `fcvt.lu.d rd,fs1,rdn.`
- 3[▼]b011: Rounding to positive infinity, corresponding to the assembly instruction `fcvt.lu.d rd,fs1,rup.`
- 3[▼]b100: Round to the nearest larger value, corresponding to the assembly instruction `fcvt.lu.d rd,fs1,rmm.`
- 3[▼]b101: Not used at this time, this code will not appear.
- 3[▼]b110: Not used at this time, this code will not appear.
- 3[▼]b111: Dynamic rounding, which determines the rounding mode based on the rm bit in the floating-point control register fcsr, corresponding to the assembly instruction `fcvt.lu.d rd, fs1.`

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
1100001	00011	fs1	rm	rd	1010011	

14.5.10 FCVT.S.D - Double Precision Floating Point to Single Precision Floating Point Instruction

Grammar:

`fcvt.s.d fd, fs1, rm`

Operation:

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

Execute permissions:

Chapter 14 Appendix A

Standardized Directive Terminology

Exception:

Illegal instruction exception

Chapter 14 Appendix A

Standardized Directive

Terminology Flag Bits:

Floating-point status bits NV/OF/UF/NX

Description:

rm determines the rounding mode.

- 3[▼]b000: Rounding to an even number, corresponding to the assembly instruction fcvt.s.d fd,fs1,rne.
- 3[▼]b001: Rounding to zero, corresponding to the assembly instruction fcvt.s.d fd,fs1,rtz.
- 3[▼]b010: Rounding to negative infinity, corresponding to the assembly instruction fcvt.s.d fd,fs1,rdn.
- 3[▼]b011: Rounding to positive infinity, corresponding to the assembly instruction fcvt.s.d fd,fs1,rup.
- 3[▼]b100: Rounding to the nearest larger value, corresponding to the assembly instruction fcvt.s.d fd,fs1,rmm.
- 3[▼]b101: Not used at this time, this code will not appear.
- 3[▼]b110: Not used at this time, this code will not appear.
- 3[▼]b111: Dynamic rounding, which determines the rounding mode based on the rm bit in the floating-point control register fcsr, corresponding to the assembly instruction fcvt.s.d fd, fs1.

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
0100000		00001		fs1		rm		fd		1010011	

14.5.11 FCVT.W.D - Double Precision Floating Point to Signed Integer Instruction

Grammar:

fcvt.w.d rd, fs1, rm

Operation:

```
tmp ← double_convert_to_signed_int(fs1)
rd←sign_extend(tmp)
```

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Chapter 14 Appendix A

Standardized Directive

Termination Flag Bits:

Floating Point Status Bit NV/NX

Description:

rm determines the rounding mode.

Chapter 14 Appendix A

Standardized Directive

Terminology

000: Rounding to an even number, corresponding to the assembly instruction

fcvt.w.d rd,fs1,rne.

- 3[▼]b001: Rounding to zero, corresponding to the assembly instruction fcvt.w.d rd,fs1,rtz.
- 3[▼]b010: Rounding to negative infinity, corresponding to the assembly instruction fcvt.w.d rd,fs1,rdn.
- 3[▼]b011: Rounding to positive infinity, corresponding to the assembly instruction fcvt.w.d rd,fs1,rup.
- 3[▼]b100: Round to the nearest larger value, corresponding to the assembly instruction fcvt.w.d rd,fs1,rmm.
- 3[▼]b101: Not used at this time, this code will not appear.
- 3[▼]b110: Not used at this time, this code will not appear.
- 3[▼]b111: Dynamic rounding, which determines the rounding mode based on the rm bit in the floating-point control register fcsr, corresponding to the assembly instruction fcvt.w.d rd, fs1.

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
1100001		00000		fs1		rm		rd		1010011	

14.5.12 FCVT.WU.D - Double Precision Floating Point to Unsigned Integer Instruction

Grammar:

fcvt.wu.d rd, fs1, rm

Operation:

```
tmp ← double_convert_to_unsigned_int(fs1)
rd←sign_extend(tmp)
```

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Impact Flag Bits:

Floating Point Status Bit NV/NX

Description:

rm determines the rounding mode.

Chapter 14 Appendix A

Standardized Directive

Terminology: 000: Rounding to an even number, corresponding to the assembly instruction

fcvt.wu.d rd,fs1,rne.

- 3[▼] b001: Rounding to zero, corresponding to the assembly instruction fcvt.wu.d rd,fs1,rtz.
- 3[▼] b010: Rounding to negative infinity, corresponding to the assembly instruction fcvt.wu.d rd,fs1,rdn.
- 3[▼] b011: Rounding to positive infinity, corresponding to the assembly instruction fcvt.wu.d rd,fs1,rup.

Chapter 14 Appendix A

Standardized Directive

Terminology

Y100: Round to the nearest larger value, corresponding to the assembly instruction fcvt.wu.d rd,fs1,rmm.

- 3[▼]b101: Not used at this time, this code will not appear.
- 3[▼]b110: Not used at this time, this code will not appear.
- 3[▼]b111: Dynamic rounding, which determines the rounding mode based on the rm bit in the floating-point control register fcsr, corresponding to the assembly instruction fcvt.wu.d rd, fs1.

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
1100001		00001		fs1		rm		rd		1010011	

14.5.13 FDIV.D - Double Precision Floating Point Division Instruction

Grammar:

fdiv.d fd, fs1, fs2, rm

Operation:

fd \leftarrow fs1 / fs2

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Impact Flag Bits:

Floating-point status bits NV/DZ/OF/UF/NX

Description:

rm determines the rounding mode.

- 3[▼]b000: Rounding to an even number, corresponding to the assembly instruction fdiv.d fd, fs1,fs2,rne.
- 3[▼]b001: Rounding to zero, corresponding to the assembly instruction fdiv.d fd, fs1,fs2,rtz.
- 3[▼]b010: Rounding to negative infinity, corresponding to the assembly instruction fdiv.d fd, fs1,fs2,rdn.
- 3[▼]b011: Rounding to positive infinity, corresponding to the assembly instruction fdiv.d fd, fs1,fs2,rup.
- 3[▼]b100: Rounding to the nearest larger value, corresponding to the assembly instruction fdiv.d fd, fs1,fs2,rmm.
- 3[▼]b101: Not used at this time, this code will not appear.

Chapter 14 Appendix A

Standardized Directive

Terminology: Not used at this time, this code will not appear.

- 3'b111: Dynamic rounding, determines the rounding mode based on the rm bit in the floating-point control register fcsr, corresponding to the assembly instructions fdiv.d fd, fs1,fs2.

Chapter 14 Appendix A

Standardized Directive

Terminology

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
0001101		fs2		fs1		rm		fd		1010011	

14.5.14 FEQ.D - Double precision floating point compare equal instruction

Grammar:

feq.d rd, fs1, fs2

Operation:

if(fs1 == fs2)

rd \leftarrow 1

else

rd \leftarrow 0

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal

Instruction

Exception

Affect Flag

Bit: Floating

Point Status

Bit NV

Instruction

Format:

31	25	24	20	19	15	14	12	11	7	6	0
1010001		fs2		fs1	010		rd		1010011		

14.5.15 FLD - Double Precision Floating Point Load Instruction

Grammar:

fld fd, imm12(rs1)

Operation:

address \leftarrow rs1 + sign_extend(imm12)

fd[63:0] \leftarrow mem[(address+7):address]

Chapter 14 Appendix A

Standardized Directive

Terminology permissions:

M mode/S mode/U mode

Exception:

Chapter 14 Appendix A

Standardized Directive

Terminology

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception, Illegal Instruction Exception

Impact Flag Bits:

not have

Command Format:

31	20 19	15 14	12 11	7 6	0
imm12[11:0]	rs1	011	fd	0000111	

14.5.16 FLE.D - Double precision floating point compare less than or equal to instruction

Grammar:

fle.d rd, fs1, fs2

Operation:

if(fs1 <= fs2)

rd \leftarrow 1

else

rd \leftarrow 0

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal

Instruction

Exception

Affect Flag

Bit: Floating

Point Status

Bit NV

Instruction

Format:

31	25 24	20 19	15 14	12 11	7 6	0
1010001	fs2	fs1	000	rd	1010011	

14.5.17 FLT.D - double precision floating point compare less than instruction

Chapter 14 Appendix A

Standardized Directive

Terminology:

flt.d rd, fs1, fs2

Operation:

if(fs1 < fs2)

rd \leftarrow 1

Chapter 14 Appendix A

Standardized Directive

Terminology

`rd ← 0`

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal

Instruction

Exception

Affect Flag

Bit: Floating

Point Status

Bit NV

Instruction

Format:

31	25	24	20	19	15	14	12	11	7	6	0
1010001		fs2		fs1		001		rd		1010011	

14.5.18 FMADD.D - Double Precision Floating Point Multiply Accumulate Instruction

Grammar:

`fmadd.d fd, fs1, fs2, fs3, rm`

Operation:

`fd ← fs1 * fs2 + fs3`

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Impact flag bits:

Floating-point status bits NV/OF/UF/IX

Description:

`rm` determines the rounding mode.

- 3[▼]b000: Rounding to an even number, corresponding to the assembly instruction
`fmadd.d fd,fs1,fs2,fs3,rne.`

Chapter 14 Appendix A

Standardized Directive

Terminology 001: Rounding to zero, corresponding to the assembly instruction fmadd.d fd,fs1, fs2, fs3, rtz.

- 3[▼] b010: Rounding to negative infinity, corresponding to the assembly instruction fmadd.d fd,fs1, fs2, fs3, rdn.
- 3[▼] b011: Rounding to positive infinity, corresponding to the assembly instruction fmadd.d fd,fs1, fs2, fs3, rup.
- 3[▼] b100: Round to the nearest larger value, corresponding to the assembly instruction fmadd.d fd,fs1, fs2, fs3, rmm.

Chapter 14 Appendix A

Standardized Directive

Terminology: 3⁷b10: Not used at this time, this code will not appear.

- 3⁷b110: Not used at this time, this code will not appear.
- 3⁷b111: Dynamic rounding, depending on the rm bit in the floating-point control register fcsr, determines the rounding mode, corresponding to the assembly instructions fmadd.d fd, fs1, fs2, fs3.

Command Format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
fs3	01			fs2		fs1		rm		fd		1000011	

14.5.19 FMAX.D - Double Precision Floating Point Maximize Instruction

Grammar:

fmax.d fd, fs1, fs2

Operation:

```
if(fs1 >= fs2)
    fd ← fs1
else
    fd ← fs2
```

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal

Instruction

Exception

Affect Flag

Bit: Floating

Point Status

Bit NV

Instruction

Format:

31	25	24	20	19	15	14	12	11	7	6	0
0010101		fs2		fs1		001		fd		1010011	

14.5.20 FMIN.D - Double Precision Floating Point Minimize Instruction

Grammar:

Chapter 14 Appendix A

Standardized Directive

Terminology fmin, fd, fs1, fs2

Operation:

if(fs1 >= fs2)

Chapter 14 Appendix A

Standardized Directive

Terminology

else

$fd \leftarrow fs1$

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal

Instruction

Exception

Affect Flag

Bit: Floating

Point Status

Bit NV

Instruction

Format:

31	25	24	20	19	15	14	12	11	7	6	0
0010101		fs2		fs1		000		fd		1010011	

14.5.21 FMSUB.D - Double Precision Floating Point Multiply Accumulate Decrease Instruction

Grammar:

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

Operation:

$fd \leftarrow fs1 * fs2 - fs3$

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Impact flag bits:

Floating-point status bits NV/OF/UF/IX

Description:

rm determines the rounding mode.

- 3^vb000: Rounding to an even number as close as possible, corresponding to the

Chapter 14 Appendix A

Standardized Directive

Terminology Assembly instructions fmsub.d fd, fs1, fs2, fs3, rne.

- 3[▼]b001: Rounding to zero, corresponding to assembly instructions fmsub.d fd, fs1, fs2, fs3, rtz.
- 3[▼]b010: Rounding to negative infinity, corresponding to the assembly instruction fmsub.d fd, fs1, fs2, fs3, rdn.
- 3[▼]b011: Rounding to positive infinity, corresponding to assembly instructions fmsub.d fd, fs1, fs2, fs3, rup.

Chapter 14 Appendix A

Standardized Directive

Terminology

Y100: Rounding to the nearest larger value, corresponding to the assembly

instructions fmsub.d fd, fs1, fs2, fs3, rmm.

- 3[▼]b101: Not used at this time, this code will not appear.
- 3[▼]b110: Not used at this time, this code will not appear.
- 3[▼]b111: Dynamic rounding, which determines the rounding mode based on the rm bit in the floating-point control register fcsr, corresponding to the assembly instructions fmsub.d fd, fs1, fs2, fs3.

Command Format:

31	27	26 25	24	20 19	15	14	12 11	7 6	0
fs3	01		fs2		fs1		rm	fd	1000111

14.5.22 FMUL.D - Double precision floating point multiply instruction

Grammar:

fmul.d fd, fs1, fs2, rm

Operation:

fd \leftarrow fs1 * fs2

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Impact Flag Bits:

Floating-point status bits NV/OF/UF/NX

Description:

rm determines the rounding mode.

- 3[▼]b000: Rounding to an even number, corresponding to the assembly instruction fmul.d fd, fs1, fs2, rne.
- 3[▼]b001: Rounding to zero, corresponding to the assembly instruction fmul.d fd, fs1, fs2, rtz.
- 3[▼]b010: Rounding to negative infinity, corresponding to the assembly instruction fmul.d fd, fs1, fs2, rdn.
- 3[▼]b011: Rounding to positive infinity, corresponding to the assembly instruction fmul.d fd, fs1, fs2, rup.
- 3[▼]b100: Rounding to the nearest larger value, corresponding to the assembly instruction fmul.d fd, fs1, fs2, rmm.
- 3[▼]b101: Not used at this time, this code will not appear.

Chapter 14 Appendix A

Standardized Directive

Terminology: Not used at this time, this code will not appear.

- 3'b111: Dynamic rounding, the rounding mode is determined by the rm bit in the floating-point control register fcsr, corresponding to the assembly instructions fmul. fd, fs1,fs2.

Chapter 14 Appendix A

Standardized Directive

Terminology

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
0001001	fs2	fs1	rm	fd	1010011	

14.5.23 FMV.D.X - Double Precision Floating Point Write Transfer Instruction

Syntax:

fmv.d.x fd, rs1

Operation:

fd \leftarrow rs1

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal

Instruction

Exceptionally

Affected

Flag Bit:

None

Description:

Integer Register Migration to Floating Point Registers

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
1111001	00000	rs1	000	fd	1010011	

14.5.24 FMV.X.D - Double Precision Floating Point Read Transfer Instruction

Syntax:

fmv.x.d rd, fs1

Operation:

rd \leftarrow fs1

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Chapter 14 Appendix A

Standardized Directive

Terminology Flag Bits:

Chapter 14 Appendix A

Standardized Directive

Terminology

Description:

Floating-point register handling to integer registers

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
1110001	00000	fs1	000	rd	1010011	

14.5.25 FNMADD.D - double precision floating point multiply accumulate take negative instruction

Grammar:

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

Operation:

fd \leftarrow -(fs1 * fs2 + fs3)

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Impact Flag Bits:

Floating-point status bits NV/OF/UF/IX

Description:

rm determines the rounding mode.

- 3[▼]b000: Rounding to an even number, corresponding to the assembly instruction
fnmadd.d fd, fs1, fs2, fs3, rne.
- 3[▼]b001: Rounding to zero, corresponding to the assembly instruction fnmadd.d fd, fs1, fs2,
fs3, rtz.
- 3[▼]b010: Rounding to negative infinity, corresponding to the assembly instruction
fnmadd.d fd, fs1, fs2, fs3, rdn.
- 3[▼]b011: Rounding to positive infinity, corresponding to the assembly instruction fnmadd.d
fd, fs1, fs2, fs3, rup.
- 3[▼]b100: Rounding to the nearest larger value, corresponding to the assembly
instruction fnmadd.d fd, fs1, fs2, fs3, rmm.
- 3[▼]b101: Not used at this time, this code will not appear.
- 3[▼]b110: Not used at this time, this code will not appear.

Chapter 14 Appendix A

Standardized Directive

Terminology: b111: Dynamic rounding, depending on the rm bit in the floating-point control register fcsr, determines the rounding mode, corresponding to the assembly instructions fnmadd.d fd,fs1, fs2, fs3.

Command Format:

Chapter 14 Appendix A

Standardized Directive

Terminal	31	27	26	25	24	20	19	15	14	12	11	7	6	0
	fs3	01		fs2		fs1		rm		fd		1001111		

14.5.26 FNMSUB.D--Double precision floating point multiply-accumulate-take-negative instruction

Grammar:

fnmsub.d fd, fs1, fs2, fs3, rm

Operation:

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

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Impact Flag Bits:

Floating-point status bits NV/OF/UF/IX

Description:

rm determines the rounding mode.

- 3^vb000: Rounding to an even number as close as possible, corresponding to the assembly instructions fnmsub.d fd, fs1, fs2, fs3, rne.
- 3^vb001: Rounding to zero, corresponding to the assembly instruction fnmsub.d fd, fs1, fs2, fs3, rtz.
- 3^vb010: Rounding to negative infinity, corresponding to the assembly instruction fnmsub.d fd, fs1, fs2, fs3, rdn.
- 3^vb011: Rounding to positive infinity, corresponding to assembly instructions fnmsub.d fd, fs1, fs2, fs3, rup.
- 3^vb100: Rounding to the nearest larger value, corresponding to the assembly instructions fnmsub.d fd, fs1, fs2, fs3, rmm.
- 3^vb101: Not used at this time, this code will not appear.
- 3^vb110: Not used at this time, this code will not appear.
- 3^vb111: Dynamic rounding, depending on the rm bit in the floating-point control register fcsr, determines the rounding mode, corresponding to the assembly instructions fnmsub.d fd, fs1, fs2, fs3.

Command Format:

31	27	26	25	24	20	19	15	14	12	11	7	6	0
www.t-head.cn	fs3	01		fs2	fs1		rm		fd		1001011		

Chapter 14 Appendix A

Standardized Directive

Terminology

14.5.27 FSD - Double Precision Floating Point Store Instruction

Grammar:

Chapter 14 Appendix A

Standardized Directive

Terminology
lsd ls2, imm12(rs1)

Operation:

address \leftarrow rs1+sign_extend(imm12)

mem[(address+63):address] \leftarrow fs2[63:0]

Execute permissions:

M mode/S mode/U mode

Exception:

Memory Instruction Non-Aligned Access Exception, Memory Instruction Access Error

Exception, Memory Instruction Page Error Exception, Illegal Instruction Exception

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
imm12[11:5]	fs2	rs1	011	imm12[4:0]	0100111	

14.5.28 FSGNJ.D - Double Precision Floating Point Symbol Injection Instruction

Grammar:

fsgnj.d fd, fs1, fs2

Operation:

fd[62:0] \leftarrow fs1[62:0]

fd[63] \leftarrow fs2[63]

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal

Instruction

Exceptionally

Affected

Flag Bit:

None

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
0010001	fs2	fs1	000	fd	1010011	

Chapter 14 Appendix A

Standardized Directive

~~Transmogrify~~ FSGNJD - Double Precision Floating Point Symbol Take and Invert

Injection Instruction

Grammar:

fsgnjn.d fd, fs1, fs2

Chapter 14 Appendix A

Standardized Directive

Terminology:

fd[62:0] \leftarrow fs1[62:0]

fd[63] \leftarrow !fs2[63]

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal

Instruction

Exceptionally

Affected

Flag Bit:

None

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
0010001		fs2		fs1		001		fd		1010011	

14.5.30 FSGNJD.D - Double Precision Floating Point Symbol Alteration

Injection Instruction

Grammar:

fsgnjx.d fd, fs1, fs2

Operation:

fd[62:0] \leftarrow fs1[62:0]

fd[63] \leftarrow fs1[63] ^ fs2[63]

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal

Instruction

Exceptionally

Affected

Flag Bit:

None

Command Format:

Chapter 14 Appendix A

Standardized Directive

Term	31	25 24	20 19	15 14	12 11	7 6	0
	0010001	fs2	fs1	010	fd	1010011	

Chapter 14 Appendix A

Standardized Directive

14.5.31 FSQRT.D - double precision floating point open square instruction

Grammar:

fsqrt.d fd, fs1, rm

Operation:

$fd \leftarrow \sqrt{fs1}$

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Impact Flag Bits:

Floating Point Status Bit NV/NX

Description:

rm determines the rounding mode.

- 3⁷b000: Rounding to an even number, corresponding to the assembly instruction
fsqrt.d fd, fs1, rne.
- 3⁷b001: Rounding to zero, corresponding to the assembly instruction fsqrt.d fd, fs1, rtz.
- 3⁷b010: Rounding to negative infinity, corresponding to the assembly instruction
fsqrt.d fd, fs1, rdn.
- 3⁷b011: Rounding to positive infinity, corresponding to the assembly instruction
fsqrt.d fd, fs1, rup.
- 3⁷b100: Rounding to the nearest larger value, corresponding to the assembly
instruction fsqrt.d fd, fs1, rmm.
- 3⁷b101: Not used at this time, this code will not appear.
- 3⁷b110: Not used at this time, this code will not appear.
- 3⁷b111: Dynamic rounding, which determines the rounding mode based on the rm
bit in the floating-point control register fcsr, corresponding to the assembly
instruction fsqrt.d fd, fs1.

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
0101101	00000	fs1	rm	fd	1010011	

14.5.32 FSUB.D - Double Precision Floating Point Subtract Instruction

Grammar:

Chapter 14 Appendix A

Standardized Directive
Terminology

Operation:

Chapter 14 Appendix A

Standardized Directive

~~Terminology~~ - fs2

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Impact Flag Bits:

Floating-point status bits NV/OF/NX

Description:

rm determines the rounding mode.

- 3^vb000: Rounding to an even number, corresponding to the assembly instructions fsub.fd, fs1, fs2, rne.
- 3^vb001: Rounding to zero, corresponding to the assembly instruction fsub.d fd, fs1, fs2, rtz.
- 3^vb010: Rounding to negative infinity, corresponding to the assembly instruction fsub.d fd, fs1, fs2, rdn.
- 3^vb011: Rounding to positive infinity, corresponding to the assembly instructions fsub.d fd, fs1, fs2, rup.
- 3^vb100: Rounding to the nearest larger value, corresponding to the assembly instruction fsub.d fd, fs1, fs2, rmm.
- 3^vb101: Not used at this time, this code will not appear.
- 3^vb110: Not used at this time, this code will not appear.
- 3^vb111: Dynamic rounding, depending on the rm bit in the floating-point control register fcsr, determines the rounding mode, corresponding to the assembly instructions fsub.fdf, fs1, fs2.

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
0000101		fs2		fs1		rm		fd		1010011	

14.6 Appendix A-6 C Command Terminology

The following is a specific description of the RISC-V C instructions implemented in the C910. The instructions in this section are 16 bits wide and are listed in alphabetical order.

Chapter 14 Appendix A

Standardized Directive

~~Terminology~~ ADD - signed addition instruction**Syntax:****c .add rd, rs2****Operation:** $rd \leftarrow rs1 + rs2$

Chapter 14 Appendix A

Standardized Directive

~~Execute~~ permissions:

M mode/S mode/U mode

Exception:

not have

Description:

$rs1 = rd \neq 0$

$rs2 \neq 0$

Command Format:

15	13	12	11	7	6	2	1	0
100	1		rs1/rd		rs2		10	

14.6.2 C.ADDI - Signed Immediate Number Addition Instruction

Grammar:

c.addi rd, nzimm6

Operation:

$rd \leftarrow rs1 + \text{sign_extend}(nzimm6)$

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Description:

$rs1 = rd \neq 0$

$nzimm6 \neq 0$

Command Format:

15	13	12	11	7	6	2	1	0
000			rs1/rd		nzimm6[4:0]		01	
					nzimm6[5]			

14.6.3 C.ADDIW - Low 32-bit Signed Immediate Number Addition Instruction

Grammar:

Chapter 14 Appendix A

Standardized Directive

Terminology: c.addiWfd, imm6

Operation:

$\text{tmp}[31:0] \leftarrow \text{rs1}[31:0] + \text{sign_extend}(\text{imm6})$

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

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Description:

$\text{rs1} = \text{rd} \neq 0$

Command Format:

15	13	12	11	7	6	2	1	0
001			rs1/rd		imm6[4:0]	01		
imm6[5]								

14.6.4 C.ADDI4SPN - 4x Immediate and Stack Pointer Summing Instruction

Grammar:

c.addi4spn rd, sp, nzuimm8<<2

Operation:

$\text{rd} \leftarrow \text{sp} + \text{zero_extend}(\text{nzuimm8}<<2)$

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Description:

$\text{nzuimm8} \neq 0$

The rd code represents the registers as follows.

- 000 x8
- 001 x9
- 010 x10

Chapter 14 Appendix A

Standardized Directive

Terminology_{0x11}

- 100 x12
- 101 x13
- 110 x14
- 111 x15

Command Format:

15	13	12	5	4	2	1	0
000			nzimm8[3:2 7:4 0 1]		rd		00

14.6.5 C.ADDI16SP--Add 16 times the number of immediately to stack pointer instruction

Grammar:

c.addi16sp sp, nzimm6<<4

Operation:

$sp \leftarrow sp + sign_extend(nzimm6<<4)$

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Command Format:

15	13	12	11	7	6	2	1	0
011			00010				01	
nzimm6[5]							nzimm6[0 2 4:3 1]	

14.6.6 C.ADDW - low 32-bit signed addition instruction

Syntax:

c .addw rd, rs2

Operation:

$tmp[31:0] \leftarrow rs1[31:0] + rs2[31:0]$

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

Execute permissions:

Chapter 14 Appendix A

Standardized Directive

M mode/S mode/U mode

Exception:

not have

Description:

$rs1 = rd$

The rd/rs1, rs2 codes represent the registers as follows.

- 000: x8
- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

Command Format:

15	13	12	11	10	9	7	6	5	4	2	1	0
100	1	11		rs1/rd	01		rs2	01				

14.6.7 C.AND - By Bit with Instruction

Syntax: c .

and rd, rs2

operations:

$rd \leftarrow rs1 \& rs2$

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Description:

$rs1 = rd$

The rd/rs1, rs2 codes represent the registers as follows.

Chapter 14 Appendix A

Standardized Directive

Terminology

- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

Command Format:

15	13	12	11	10	9	7	6	5	4	2	1	0
100	0	11			rs1/rd	11			rs2	01		

14.6.8 C.ANDI - Immediate Number by Bit and Instruction

Grammar:

c.andi rd, imm6

Operation:

$rd \leftarrow rs1 \& \text{sign_extend}(imm6)$

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Description:

$rs1 = rd$

The rd/rs1 code represents the registers as follows.

- 000: x8
- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13

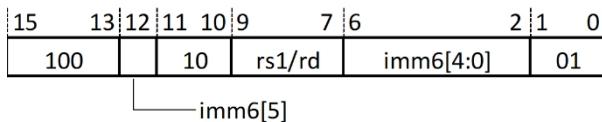
Chapter 14 Appendix A

Standardized Directive

Terminology

- 111: x15

Command Format:



14.6.9 C.BEQZ - Branch Equal to Zero Instruction

Grammar:

c.b eqz rs1, label

Operation:

if (rs1 == 0)

 next pc = current pc + imm8<<1;

else

 next pc = current pc + 2;

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Description:

The rs1 code represents the registers as follows.

- 000: x8
- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

Chapter 14 Appendix A

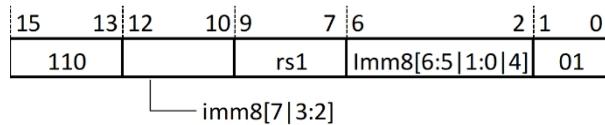
Standardized Directive

Terminology

The assembler calculates imm8 from the label

Command jump range $\pm 256B$ address space

Command Format:



14.6.10 C.BNEZ - Branch Not Equal to Zero Instruction

Grammar:

c.b nez rs1, label

Operation:

if ($rs1 \neq 0$)

next pc = current pc + imm8<<1;

else

next pc = current pc + 2;

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Description:

The rs1 code represents the registers as follows.

- 000: x8
- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

Chapter 14 Appendix A

Standardized Directive

Terminology

The assembler calculates imm12 from the label

Command jump range ±256B address space

Command Format:

15	13	12	10	9	7	6	2	1	0
111				rs1		imm[6:5 1:0 4]		01	
imm8[7 3:2]									

14.6.11 C.EBREAK - Breakpoint Instruction

Syntax:

c.ebreak

operation:

on:

Generate a breakpoint exception or enter debug mode

Execute permissions:

M mode/S mode/U mode

Exception:

breakpoint exception

Command Format:

15	13	12	11	7	6	2	1	0
100	1		00000		00000		10	

14.6.12 C.FLD - Floating Point Double Word Load Instruction

Grammar:

c.fld fd, uimm5<<3(rs1)

Operation:

address ← rs1 + zero_extend(uimm5<<3)

fd ← mem[address+7:address]

Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception

Chapter 14 Appendix A

Standardized Directive

Terminology Description:

The rs1 code represents the registers as follows.

- 000: x8
- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

The fd code represents the registers as follows.

- 000: f8
- 001: f9
- 010: f10
- 011: f11
- 100: f12
- 101: f13
- 110: f14
- 111: f15

Command Format:

15	13	12	10	9	7	6	5	4	2	1	0
011			rs1				fd		00		

uimm5[2:0] uimm5[4:3]

14.6.13 C.FLDSP - Floating Point Double Word Stack Load Instruction

Grammar:

c fldsp fd, uimm6<<3(sp)

Operation:

address \leftarrow sp + zero_extend(uimm6<<3)

fd \leftarrow mem[address+7:address]

Chapter 14 Appendix A

Standardized Directive

~~Terminology~~ Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception

Command Format:

15	13	12	11	7	6	2	1	0
001			fd	uimm6[1:0 5:3]	10			

uimm6[2]

14.6.14 C.FSD - Floating Point Double Word Storage Instruction

Grammar:

c.fsd fs2, uimm5<<3(rs1)

Operation:

address \leftarrow rs1 + zero_extend(uimm5<<3)

mem[address+7:address] \leftarrow fs2

Execute permissions:

M mode/S mode/U mode

Exception:

Memory Instruction Non-Aligned Access Exception, Memory Instruction Access Error Exception, Memory Instruction Page Error Exception

Description:

The fs1 code represents the following registers.

- 000: x8
- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

The rs2 code represents the following registers.

Chapter 14 Appendix A

Standardized Directive

Terminology

- 001: f9
- 010: f10
- 011: f11
- 100: f12
- 101: f13
- 110: f14
- 111: f15

Command Format:

15	13	12	10	9	7	6	5	4	2	1	0
101				rs1				fs2		00	

uimm5[2:0] uimm5[4:3]

14.6.15 C.FSDSP - Floating Point Double Word Stack Store Instruction

Grammar:

c.fsdsp fs2, uimm6<<3(sp)

Operation:

address \leftarrow sp + zero_extend(uimm6<<3)

mem[address+7:address] \leftarrow fs2

Execute permissions:

M mode/S mode/U mode

Exception:

Memory Instruction Non-Aligned Access Exception, Memory Instruction Access Error Exception, Memory Instruction Page Error Exception

Command Format:

15	13	12	7	6	2	1	0
101		uimm6[2:0 5:3]		fs2		10	

14.6.16 C.J - Unconditional Jump Instruction

Grammar:

c.j label

Chapter 14 Appendix A

Standardized Directive

Terminology:

next pc \leftarrow current pc + sign_extend(imm<<1);

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Description:

The assembler calculates the jump range of the imm11 instruction from the label to be $\pm 2\text{KB}$ address space

instruction format:

15	13	12		2	1	0
101		imm11[10 3 8:7 9 5 6 2:0 4]		01		

14.6.17 C.JALR - Register Jump Rotor Program Instruction

Syntax:

c.jalr rs1

Operatio

n:

next pc \leftarrow rs1; x1

\leftarrow current pc + 2;

execute

permissions:

M mode/S mode/U mode

Exception:

not have

Description:

rs1 ! = 0.

When the MMU is on, the jump range is the full 512GB address space. When the MMU is closed, the jump range is the entire 1TB

Chapter 14 Appendix A

Standardized Directive

Terminology space. Command Format:

15	13	12	11	7	6	2	1	0
100	1		rs1		00000		10	

Chapter 14 Appendix A

Standardized Directive

14.6.18 JR - Register Jump Instruction

Syntax

:c.jr rs1

Operat

ion:

next pc = rs1;

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Description:

rs1 != 0.

When the MMU is on, the jump range is the full 512GB address space. When the MMU is closed, the jump range is the entire 1TB address space. **Command Format:**

15	13	12	11	7	6	2	1	0
100	0		rs1		00000		10	

14.6.19 C.LD - double word load instruction

Grammar:

c.ld rd, uimm5<<3(rs1)

Operation:

address ← rs1 + zero_extend(uimm5<<3)

rd ← mem[address+7:address]

Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception

Description:

The rs1/rd code represents the registers as follows.

Chapter 14 Appendix A

Standardized Directive

Terminology^{000; x8}

Chapter 14 Appendix A

Standardized Directive

Terminology x9

- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

Command Format:

15	13	12	10	9	7	6	5	4	2	1	0
011				rs1				rd		00	

uimm5[2:0] uimm5[4:3]

14.6.20 C.LDSP - double word stack load instruction

Grammar:

c.ldsp rd, uimm6<<3(sp)

Operation:

address \leftarrow sp + zero_extend(uimm6<<3)

rd \leftarrow mem[address+7:address]

Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception

Description:

rd != rd !

Command Format:

15	13	12	11		7	6		2	1	0
011				rd		uimm6[1:0 5:3]		10		

uimm6[2]

Chapter 14 Appendix A

Standardized Directive

14.6.21 LI - Immediate Number Transfer Instruction

Grammar:

c.li rd, imm6

Operation:

$rd \leftarrow \text{sign_extend}(\text{imm6})$

Execute permissions:

M mode/S mode/U mode

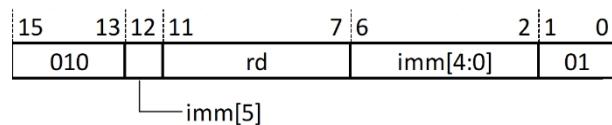
Exception:

not have

Description:

$rd \neq 0.$

Command Format:



14.6.22 C.LUI - High Immediate Number Transfer Instructions

Grammar:

c.lui rd, nzimm6

Operation:

$rd \leftarrow \text{sign_extend}(\text{nzimm6} \ll 12)$

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Description:

$rd \neq 0.$

$\text{Nzimm6} \neq 0.$

Instruction format:

Chapter 14 Appendix A

Standardized Directive

Terminology

15	13	12	11	7	6	2	1	0
011			rd		nzimm6[4:0]	01		

nzimm6[5]

14.6.23 C.LW - word loading instructions

Grammar:

c.lw rd, uimm5<<2(rs1)

Operation:

address \leftarrow rs1 + zero_extend(uimm5<<2)

tmp[31:0] \leftarrow mem[address+3:address]

rd \leftarrow sign_extend(tmp[31:0])

Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception

Description:

The rs1/rd code represents the registers as follows.

- 000: x8
- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

Command Format:

15	13	12	10	9	7	6	5	4	2	1	0
010				rs1			rd		00		

uimm5[3:0] uimm5[0|4]

Chapter 14 Appendix A

Standardized Directive

14.6.24 LWSP - word stack load instruction

Grammar:

c.lwsp rd, uimm6<<2(sp)

Operation:

address \leftarrow sp + zero_extend(uimm6<<2)

tmp[31:0] \leftarrow mem[address+3:address]

rd \leftarrow sign_extend(tmp[31:0])

Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception

Description:

rd != rd !

Command Format:

15	13	12	11	7	6	2	1	0
010			rd		umm6[2:0 5:4]	10		

uimm6[3]

14.6.25 C.MV - Data Transfer Instructions

Syntax:

c.mv rd, rs2

Operation:

rd \leftarrow rs2.

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Description:

rs2 != 0, rd != 0.

Command Format:

Chapter 14 Appendix A

Standardized Directive

Terminology

15	13	12	11	7	6	2	1	0
100	0		rd		rs2		10	

14.6.26 C.NOP - Null Instruction

Syntax

: c.nop

Operation

ion: no

operati

on

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Command Format:

15	13	12	11	7	6	2	1	0
000	0		00000		00000		01	

14.6.27 C.OR - by bit or instruction

Syntax:

c .or rd, rs2

Operation

s:

$rd \leftarrow rs1 | rs2$

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Description:

$rs1 = rd$

The rd/rs1 code represents the registers as follows.

- 000: x8

Chapter 14 Appendix A

Standardized Directive

Terminology^{001; x9}

Chapter 14 Appendix A

Standardized Directive

Terminology

- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

Command Format:

15	13	12	11	10	9	7	6	5	4	2	1	0
100	0	11		rs1/rd		10		rs2		01		

14.6.28 C.SD - double word storage instruction

Grammar:

c.sd rs2, uimm5<<3(rs1)

Operation:

address \leftarrow rs1 + zero_extend(uimm5<<3)

mem[address+7:address] \leftarrow rs2

Execute permissions:

M mode/S mode/U mode

Exception:

Memory Instruction Non-Aligned Access Exception, Memory Instruction Access Error Exception, Memory Instruction Page Error Exception

Description:

The rs1/rd code represents the registers as follows.

- 000: x8
- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

Chapter 14 Appendix A

Standardized Directive

Terminology

Command Format:

15	13	12	10	9	7	6	5	4	2	1	0
111				rs1				rd		00	

uimm5[2:0] uimm5[4:3]

14.6.29 C.SDSP - double word stack store instructions

Grammar:

c.fsdsp rs2, uimm6<<3(sp)

Operation:

address \leftarrow sp+ zero_extend(uimm6<<3)

mem[address+7:address] \leftarrow rs2

Execute permissions:

M mode/S mode/U mode

Exception:

Memory Instruction Non-Aligned Access Exception, Memory Instruction Access Error Exception, Memory Instruction Page Error Exception

Command Format:

15	13	12	7	6	2	1	0
111		uimm6[2:0 5:3]		rs2		10	

14.6.30 C.SLLI - Immediate Logical Left Shift Instruction

Grammar:

c.slli rd, nzuimm6

Operation:

rd \leftarrow rs1 << nzuimm6

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Description:

Chapter 14 Appendix A

Standardized Directive

Terminology

rd/rs1 != 0, nzuimm6 != 0

Command Format:

15	13	12	11	7	6	2	1	0
000			rs1/rd		nzuimm6[4:0]	10		

nzuimm6[5]

14.6.31 C.SRAI - immediate number arithmetic right shift instruction

Grammar:

c.srl rd, nzuimm6

Operation:

rd ← rs1 >>> nzuimm6

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Description:

nzuimm6 !=

0 rs1 == rd

The rs1/rd code represents the registers as follows.

- 000: x8
- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

Command Format:

Chapter 14 Appendix A

Standardized Directive

Terminology:

15	13	12	11 10	9	7	6	2	1	0
100		01	rs1/rd	nzuimm6[4:0]	01				

nzuimm6[5]

14.6.32 C.SRLI - Immediate Logical Right Shift Instruction

Grammar:

c.srli rd, nzuimm6

Operation:

$rd \leftarrow rs1 >> nzuimm6$

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Description:

$nzuimm6 !=$

$0 rs1 == rd$

The rs1/rd code represents the registers as follows.

- 000: x8
- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

Command Format:

15	13	12	11 10	9	7	6	2	1	0
100		00	rs1/rd	nzuimm6[4:0]	01				

nzuimm6[5]

Chapter 14 Appendix A

Standardized Directive

14.6.33 SW - word storage instructions

Grammar:

c.sw rs2, uimm5<<2(rs1)

Operation:

address \leftarrow rs1 + zero_extend(uimm5<<2)

mem[address+3:address] \leftarrow rs2

Execute permissions:

M mode/S mode/U mode

Exception:

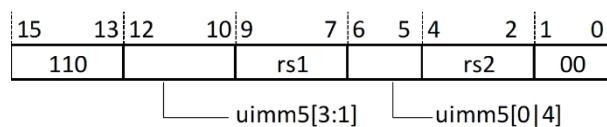
Memory Instruction Non-Aligned Access Exception, Memory Instruction Access Error Exception, Memory Instruction Page Error Exception

Description:

The rs1/rs2 code represents the registers as follows.

- 000: x8
- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

Command Format:



14.6.34 C.SWSP - word stack storage instructions

Grammar:

c.swsp rs2, uimm6<<2(sp)

Operation:

Chapter 14 Appendix A

Standardized Directive

Terminology address\$2—sp+ zero_extend(uimm6<<2)

mem[address+3:address] \leftarrow rs2

Execute permissions:

M mode/S mode/U mode

Exception:

Memory Instruction Non-Aligned Access Exception, Memory Instruction Access Error Exception, Memory Instruction Page Error Exception

Command Format:

15	13	12	7	6	2	1	0
110	uimm6[3:0 5:4]		rs2		10		

14.6.35 C.SUB - Signed Subtraction Instruction

Syntax:

c .sub rd, rs2

Operation:

rd \leftarrow rs1 - rs2

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Description:

rs1 == rd

The rs1/rd code represents the registers as follows.

- 000: x8
- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

Chapter 14 Appendix A

Standardized Directive

Terminology

Command Format:

15	13	12	11	10	9	7	6	5	4	2	1	0
100	0	11		rs1/rd	00		rs2		01			

14.6.36 C.SUBW--Lower 32-bit Signed Subtract Instruction

Syntax:

c.subw rd, rs2

Operation:

$\text{tmp}[31:0] \leftarrow \text{rs1}[31:0] - \text{rs2}[31:0]$

$\text{rd} \leftarrow \text{sign_extend}(\text{tmp})$

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Description:

$\text{rs1} == \text{rd}$

The rs1/rd code represents the registers as follows.

- 000: x8
- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

command format

15	13	12	11	10	9	7	6	5	4	2	1	0
100	1	11		rs1/rd	00		rs2		01			

Chapter 14 Appendix A

Standardized Directive

~~Terminology~~XOR - per-positional different-or instruction

Syntax:

c .xor rd, rs2

Operations

:

$rd \leftarrow rs1 \wedge rs2$

Execute permissions:

M mode/S mode/U mode

Exception:

not have

Description:

$rs1 == rd$

The rs1/rd code represents the registers as follows.

- 000: x8
- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

Command Format:

15	13	12	11	10	9	7	6	5	4	2	1	0
100	0	11		rs1/rd	01		rs2		01			

14.7 Appendix A-8 Pseudo Instruction List

RISC-V implements a series of pseudo-instructions, listed here for reference only, in alphabetical order.

bogus instruction	basic instruction	hidden meaning
beqz rs, offset	beq rs, x0, offset	Register zero branch jump
bnez rs, offset	bne rs, x0, offset	Register non-zero branch jump



Chapter 14 Appendix A
Standardized Directive
Terminology

Continued on next page

Chapter 14 Appendix A

Standardized Directive

Terminology

Table 14.1 - continued from previous page

bogus instruction	basic instruction	hidden meaning
blez rs, offset	bge x0,rs,offset	Register less than or equal to zero jump
bgez rs, offset	bge rs, x0, offset	Register greater than or equal to zero jump
bltz rs, offset	blt rs, x0, offset	Register less than zero jump
bgtz rs, offset	blt x0, xs, offset	Register greater than zero jump
bgt rs, rt, offset	blt rt, rs, offset	Compare greater than branch jump
ble rs, rt, offset	bge rt, rs, offset	Compare less than or equal to branch jump
bgtu rs, rt, offset	bltu rt, rs, offset	Unsigned comparisons are greater than branch jumps
bleu rs, rt, offset	bgeu rt, rs, offset	Unsigned comparison less than or equal to branch jump
call offset	auipc x6, offset[31:12] jalr x1, x6, offset[11:0]	Functions that jump 4KB-4GB of space
csrc csr, rs	csrrc x0, csr, rs	Clear the corresponding bit in the control register
csrci csr, imm	csrrci x0, csr, imm	Clear the ratio in the lower 6 bits of the control register. distinguished
csrs csr, rs	csrrs x0, csr, rs	Setting the corresponding bit in the control register.
csrssi csr, imm	csrrsi x0, csr, imm	The corresponding ratio in the lower 6 bits of the Set Control Register distinguished
csrwr csr, rs	csrrw x0, csr, rs	Write the corresponding bit in the control register
csrwi csr, imm	csrrwi x0, csr, imm	Write the corresponding bit in the lower 6 bits of the control register.
fabs.d rd , rs	fsgnjx.d rd, rs,rs	Double-precision floating-point numbers take absolute values
fabs.s rd , rs	fsgnjx.s rd, rs,rs	Single-precision floating-point numbers take absolute values
fence	fence iorw, iorw	Memory and Peripheral Synchronization Instructions
fl{w d} rd, symbol, rt	auipc rt, symbol[31:12] fl{w d} rd, symbol[11:0](rt)	4GB address space of next page point load instruction
fmv.d rd, rs	fsgnj.d rd, rs,rs	Double precision floating point copy instruction
fmv.s rd, rs	fsgnj.s rd, rs,rs	Single-precision floating-point copy instruction
www.t-head.cn	274	
fneg.d rd, rs	fsgnjn.d rd, rs,rs	Double precision floating point take negative instruction

Chapter 14 Appendix A

Standardized Directive

Terminology

Table 14.1 - continued from previous page

bogus instruction	basic instruction	hidden meaning
fsrmi imm	csrrwi x0, frm,imm	Immediate Number Write Floating Point Rounding Bit Instruction
fsrmi rd, imm	csrrwi rd, frm, imm	Floating Point Rounding Bit Immediate Number Read and Write Instructions
fs{w d} rd, symbol, rt	auipc rt, symbol[31:12] fs{w d} rd, symbol[11:0](rt)	4GB address space floating-point memory instructions
j offset	jal x0, offset	direct jump instruction
jal offset	jal x1, offset	Subroutine Jump and Link Instructions
jalr rs	jalr x1, rs, 0	Subroutine Jump Registers and Link Registers directives
jr rs	jalr x0, rs, 0	Jump Register Instruction
la rd, symbol	auipc rd, symbol[31:12] addi rd, rd, symbol[11:0]	Instruction Address Load Instruction
li rd, immediate	Split into multiple instructions based on immediate size	Immediate Number Load Instruction
l{b h w d} rd, symbol, rt	auipc rt, symbol[31:12] l{b h w d} rd, symbol[11:0](rt)	4GB of address space to load instructions
mv rd, rs	addi rd, rs, 0	data transmission instruction
neg rd, rs	sub rd, x0, rs	Register Fetch Negative Instruction
negw rd, rs	subw rd, x0, rs	Register low 32 bits take negative instruction
nop	addi x0,x0,0	empty instruction
not rd, rs	xori rd, rs, -1	register inverse instruction
rdcycle[h] rd	csrrs rd, cycle[h], x0	Cycle number read command
rdinstret[h] rd	csrrs rd, instret[h], x0	Instruction Number Read Instruction
rdtime[h] rd	csrrs rd, time[h], x0	Real Clock Read command
ret	jalr x0, x1,0	subroutine return instruction
s{b h w d} rd, symbol, rt	auipc rt, symbol[31:12] s{b h w d} rd, symbol[11:0](rt)	4GB of address space to store instructions
seqz rd, rs	sltiu rd, rs, 1	Register is 0 Set 1 Instruction
sextw rd, rs	addiw rd, rs, 0	symbolic bit expansion instruction
sgtz rd, rs	slt rd, rs, x0, rs	Register greater than 0 Set 1 Instruction
sltz rd, rs	slt rd, rs, rs, x0	Register less than 0 Set 1 instruction
snez rd, rs	sltu rd, rs, x0, rs	Register not 0 Set 1 Instruction
tail offset	auipc x6, offset[31:12]	Register unlinked jump rotor

Chapter 15 Appendix B Flathead Extended Directive Terminology

In addition to the GC instruction set defined in the standard, the C910 implements custom instruction sets, including a subset of Cache instructions, a subset of Synchronization instructions, a subset of Arithmetic instructions, a subset of Bit manipulation instructions, a subset of Memory instructions, and a subset of Floating-point Half-precision instructions.

The subset of Cache instructions, the subset of synchronization instructions, the subset of arithmetic operations instructions, the subset of bit manipulation instructions, and the subset of storage instructions need to be defined in the

`mxstatus.theadisae == 1` for normal execution, otherwise an illegal instruction exception is thrown; a subset of floating-point, half-precision instructions are required to be executed in `mstatus.fs`.

The following describes each instruction according to different instruction subset extensions. `=2'b00` can be executed normally, otherwise an illegal instruction exception is generated. The following describes each instruction according to different instruction subset extensions.

15.1 Appendix B-1 Cache Instruction Terminology

The Cache instruction subset implements cache operations, and each instruction is 32 bits wide. The following instructions are listed in alphabetical order.

15.1.1 DCACHE.CALL--DCACHE Clear All Dirty Table Entries Command

Syntax:

`dcache.call`

Operation

:

clear all L1 dcache table entries, writes all dirty table entries back to the next level of storage, and operates only on the current core.

Execute permissions:

M mode/S mode

Exception:

Illegal instruction exception

Description:

mxstatus.theadisaee=0, execution of this instruction generates an illegal instruction exception. mxstatus.theadisaee=1, execution of this instruction in U mode generates an illegal instruction exception. **Instruction format:**

Chapter 15 Appendix B Flathead Extended Command Terminology

31	25 24	20 19	15 14	12 11	7 6	0
0000000	00001	00000	000	00000	0001011	

15.1.2 DCACHE.CIALL--DCACHE Invalid command after clearing all dirty table entries.

Syntax:

dcache.ciall

Operation:

Invalidate all L1 dcache dirty table entries after writing them back to the next level of storage.

Execute permissions:

M mode/S mode

Exception:

Illegal instruction exception

Description:

mxstatus.theadisae=0, execution of this instruction generates an illegal instruction exception. mxstatus.theadisae=1, execution of this instruction in U mode generates an illegal instruction exception. **Instruction format:**

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

15.1.3 DCACHE.CIPA -- DCACHE clears dirty table entries by physical address and invalidates them

Syntax:

dcache.cipa rs1

Operation:

Writes the dcache/L2cache table entry to which the physical address in rs1 belongs back to the lower level storage and invalidates the table entry, manipulating all core and L2CACHE.

Execute permissions:

M mode/S mode

Exception:

Chapter 15 Appendix B Flathead Extended Command Terminology

Illegal instruction exception

Description:

Chapter 15 Appendix B Flathead Extended Command Terminology

`mxstatus.theadisae=0`, execution of this instruction generates an illegal instruction exception. `mxstatus.theadisae=1`, execution of this instruction in U mode generates an illegal instruction exception. **Instruction format:**

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

15.1.4 DCACHE.CISW--DCACHE clears dirty table entries and invalidates instructions by way/set

Syntax:

`dcache.cisw rs1`

Operation:

Writes the L1 dache dirty table entry back to the next level of storage and invalidates the table entry according to the way/set specified in rs1, operating only on the current core.

Execute permissions:

M mode/S mode

Exception:

Illegal instruction exception

Description:

C910 dcache is a two-way group concatenated with rs1[31] encoded as way and rs1[w:6] as set. When dcache is 32K, w is 13, and when dcache is 64K, w is 14.

- `mxstatus.theadisae=0`, execution of this instruction produces an illegal instruction exception.
- `mxstatus.theadisae=1`, the execution of this instruction in U mode generates an illegal instruction exception.

Command Format:

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

15.1.5 DCACHE.CIVA - DCACHE clears dirty table entries by virtual address and invalidates the

Syntax:

`dcache.civa rs1`

Chapter 15 Appendix B Flathead Extended Command Terminology

Operation:

Writes the dcache/L2 cache table entry to which the virtual address specified by rs1 belongs back to lower-level storage and invalidates the table entry, operates the current core and L2CACHE, and decides whether to broadcast to other cores based on the virtual address sharing attributes.

Execute permissions:

Chapter 15 Appendix B Flathead Extended Command Terminology

M mode/S mode/U mode

Exception:

Illegal Instruction Exception/Load Instruction Page Error Exception

Description:

- mxstatus.theadisae=0, execution of this instruction produces an illegal instruction exception.
- mxstatus.theadisae = 1, mxstatus.ucme = 1, the instruction can be executed in U mode.
- mxstatus.theadisae = 1, mxstatus.ucme = 0. Execution of this instruction in U mode generates an illegal instruction exception.

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
0000001		00111		rs1		000		00000		0001011	

15.1.6 DCACHE.CPA - DCACHE Clears dirty table entries by physical address

Syntax:

dcache.cpa rs1

Operation:

Writes the dcache/l2cache table entry corresponding to the physical address in rs1 back to the next level of storage, operating all cores and L2CACHE.

Execute permissions:

M mode/S mode

Exception:

Illegal instruction exception

Description:

mxstatus.theadisae=0, execution of this instruction generates an illegal instruction exception. mxstatus.theadisae=1, execution of this instruction in U mode generates an illegal instruction exception. **Instruction format:**

31	25	24	20	19	15	14	12	11	7	6	0
0000001		01001		rs1		000		00000		0001011	

15.1.7 DCACHE.CPAL1 --L1DCACHE Clear dirty table entries by physical address

Chapter 15 Appendix B Flathead

Extended Command Terminology

Grammar:

```
dcache.cpal1 rs1
```

Operation: Writes the dcache table entry corresponding to the physical address in rs1 back to the next level of storage and operates all core L1CACHES.

Chapter 15 Appendix B Flathead Extended Command Terminology

Execute permissions:

M mode/S mode

Exception:

Illegal instruction exception

Description:

`mxstatus.theadisae=0`, execution of this instruction generates an illegal instruction exception. `mxstatus.theadisae=1`, execution of this instruction in U mode generates an illegal instruction exception. **Instruction format:**

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

15.1.8 DCACHE.CVA--DCACHE Clears dirty table entries by virtual address

Syntax:

`dcache.cva rs1`

Operation:

Writes the dcache/l2cache table entries corresponding to virtual addresses in rs1 back to the next level of storage, operates on the current core and L2CACHE, and decides whether or not to broadcast to other cores based on the virtual address sharing attributes

Execute permissions:

M mode/S mode

Exception:

Illegal Instruction Exception/Load Instruction Page Error Exception

Description:

`mxstatus.theadisae=0`, execution of this instruction generates an illegal instruction exception. `mxstatus.theadisae=1`, execution of this instruction in U mode generates an illegal instruction exception. **Instruction format:**

31	25	24	20	19	15	14	12	11	7	6	0
0000001		00101		rs1		000		00000		0001011	

15.1.9 DCACHE.CVVAL1--L1DCACHE Clear dirty table entries by virtual address

Chapter 15 Appendix B Flathead

Extended Command Terminology

Grammar:

dcache.cval1 rs1

Chapter 15 Appendix B Flathead Extended Command Terminology

Operation:

Writes the dcache table entry corresponding to the virtual address in rs1 back to the next level of storage, operating all core L1CACHES

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal Instruction Exception/Load Instruction Page Error Exception

Description:

`mxstatus.theadisae = 0`, execution of this instruction generates an illegal instruction exception. `mxstatus.theadisae = 1, mxstatus.ucme = 0`, execution of this instruction in U mode generates an illegal instruction exception.

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
0000001	00100	rs1	000	00000	0001011	

15.1.10 DCACHE.IPA -- DCACHE Invalid by Physical Address Instruction

Syntax:

`dcache.ipa rs1`

Operation:

Invalidates the dcache/l2 cache table entry corresponding to the physical address in rs1, operating all cores and L2CACHE.

Execute permissions:

M mode/S mode

Exception:

Illegal instruction exception

Description:

- `mxstatus.theadisae=0`, execution of this instruction produces an illegal instruction exception.
- `mxstatus.theadisae=1`, the execution of this instruction in U mode generates an illegal instruction exception.

Command Format:

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

Chapter 15 Appendix B Flathead

Extended Command Terminology

15.1.11 DCACHE.ISW --DCACHE by set/way Invalid Command

Syntax:

dcache.isw rs1

Operation:

Invalidate dcache table entries that specify SET and WAY and operate only on the current core.

Execute permissions:

M mode/S mode

Exception:

Illegal instruction exception

Description:

C910 dcache is a two-way group concatenated with rs1[31] encoded as way and rs1[w:6] as set. When dcache is 32K, w is 13, and when dcache is 64K, w is 14.

- mxstatus.theadisae=0, execution of this instruction produces an illegal instruction exception.
- mxstatus.theadisae=1, the execution of this instruction in U mode generates an illegal instruction exception.

Command Format:

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

15.1.12 DCACHE.IVA --DCACHE Invalid by Virtual Address Directive

Syntax:

dcache.iva rs1

Operation:

Invalidate the dcache/l2 cache table entry corresponding to the virtual address in rs1, operates the current core and L2CACHE, and decides whether to broadcast to other cores based on the virtual address sharing attribute.

Execute permissions:

M mode/S mode

Exception:

Illegal Instruction Exception/Load Instruction Page Error Exception

Description:

- mxstatus.theadisae=0, execution of this instruction produces an illegal instruction exception.

Chapter 15 Appendix B Flathead

Extended Command Terminology

- mxstatus.theadisae=1, the execution of this instruction in U mode generates an illegal instruction exception.

Chapter 15 Appendix B Flathead Extended Command Terminology

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
0000001		00110		rs1	000		00000		0001011		

15.1.13 DCACHE.IALL - DCACHE Invalid All Table Entries command

Syntax:

dcache.iall

operation

:

Invalidates all L1 dcache table entries and operates only on the current core.

Execute permissions:

M mode/S mode

Exception:

Illegal instruction exception

Description:

- mxstatus.theadisae=0, execution of this instruction produces an illegal instruction exception.
- mxstatus.theadisae=1, the execution of this instruction in U mode generates an illegal instruction exception.

Command Format:

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

15.1.14 ICACHE.IALL - ICACHE Invalid All Table Entries Command

Syntax:

icache.iall

operatio

n:

Invalidates all icache table entries and operates only on the current core.

Execute permissions:

M mode/S mode

Exception:

Illegal instruction exception

Description:

Chapter 15 Appendix B Flathead Extended Command Terminology

`mxstatus.theadisae=0`, execution of this instruction generates an illegal instruction exception. `mxstatus.theadisae=1`, execution of this instruction in U mode generates an illegal instruction exception. **Instruction format:**

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

15.1.15 ICACHE.IALLS - ICACHE Broadcast Invalid All Table Entries Directive

Syntax:

`icache.ialls`

Operation

:

Invalidate all icache table entries and broadcasts to other cores to de-invalidate all of their respective icache table entries, operating on all cores.

Execute permissions:

M mode/S mode

Exception:

Illegal instruction exception

Description:

`mxstatus.theadisae=0`, execution of this instruction generates an illegal instruction exception. `mxstatus.theadisae=1`, execution of this instruction in U mode generates an illegal instruction exception. **Instruction format:**

31	25 24	20 19	15 14	12 11	7 6	0
0000000	10001	00000	000	00000	0001011	

15.1.16 ICACHE.IPA - ICACHE Invalid Table Entry by Physical Address command

Syntax:

`icache.ipa rs1`

Operation:

Invalidate the icache table entry corresponding to the physical address in `rs1`. operate all cores.

Chapter 15 Appendix B Flathead

Extended Command Terminology

Execute permissions:

M mode/S mode

Exception:

Illegal instruction exception

Chapter 15 Appendix B Flathead Extended Command Terminology

Description:

`mxstatus.theadisae=0`, execution of this instruction generates an illegal instruction exception. `mxstatus.theadisae=1`, execution of this instruction in U mode generates an illegal instruction exception. **Instruction format:**

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

15.1.17 ICACHE.IVA - ICACHE Invalid Table Entry by Virtual Address command

Syntax:

`icache.iva rs1`

Operation:

Invalidate the icache table entry corresponding to the virtual address in `rs1`, operates the current core, and decides whether to broadcast to other cores based on the virtual address sharing attributes.

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal Instruction Exception/Load Instruction Page Error Exception

Description:

`mxstatus.theadisae=0`, execution of this instruction generates an illegal instruction exception. `mxstatus.theadisae=1`, `mxstatus.ucme=1`, the instruction can be executed in U mode. `mxstatus.theadisae=1`, `mxstatus.ucme=0`, U mode execution of this instruction generates an illegal instruction exception. **Instruction format:**

31	25	24	20	19	15	14	12	11	7	6	0
0000001		10000		rs1	000		00000		0001011		

15.1.18 L2CACHE.CALL--L2CACHE Clear All Dirty Table Entries Command

Syntax:

`l2cache.call`

Operation:

Write all dirty table entries in l2cache back to the next level of storage

Chapter 15 Appendix B Flathead

Extended Command Terminology

Execute permissions:

Chapter 15 Appendix B Flathead Extended Command Terminology

M mode/S mode

Exception:

Illegal instruction exception

Description:

- mxstatus.theadisae=0, execution of this instruction produces an illegal instruction exception.
- mxstatus.theadisae=1, the execution of this instruction in U mode generates an illegal instruction exception.

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
0000000		10101		00000		000		00000		0001011	

15.1.19 L2CACHE.CIALL--L2CACHE Clears all dirty table entries and invalidates commands.

Syntax:

l2cache.ciall

Operation:

Invalidates all l2 table entries after writing all dirty table entries in l2cache back to the next level of storage

Execute permissions:

M mode/S mode

Exception:

Illegal instruction exception

Description:

- mxstatus.theadisae=0, execution of this instruction produces an illegal instruction exception.
- mxstatus.theadisae=1, the execution of this instruction in U mode generates an illegal instruction exception.

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
0000000		10111		00000		000		00000		0001011	

15.1.20 L2CACHE.IALL--L2CACHE Invalid Instruction

Syntax:

l2cache.iall

Chapter 15 Appendix B Flathead Extended Command Terminology

Operation

:

Invalidate all table entries in l2cache

Chapter 15 Appendix B Flathead Extended Command Terminology

Execute permissions:

M mode/S mode

Exception:

Illegal instruction exception

Description:

- mxstatus.cskisayee=0, execution of this instruction produces an illegal instruction exception.
- mxstatus.cskisayee=1, execution of this instruction in U mode generates an illegal instruction exception.

Command Format:

31	25	24	20	19	15	14	12	11	7	6	0
0000000		10110		00000	000		00000		0001011		

15.1.21 DCACHE.CSW --DCACHE cleans dirty table entries by set/way

Syntax.

dcache.csw rs1

Operation.

Press SET and WAY to write back the dirty table entries in the dcache to the next level of memory.

Execution privileges.

M mode/S mode

Exception.

Illegal instruction exception

Description:

C910 dcache is a two-way group concatenated with rs1[31] encoded as way and rs1[w:6] encoded as set. When dcache is 32K, w is 13, and when dcache is 64K, w is 14.

mxstatus.theadisae=0, execution of this instruction generates an illegal instruction exception. mxstatus.theadisae=1, execution of this instruction in U mode generates an illegal instruction exception. **Instruction Format.**

15.2 Appendix B-2 Multicore Synchronization Instruction Terminology

Chapter 15 Appendix B Flathead

Extended Command Terminology

The synchronous instruction subset implements an extension of the multicore synchronous instructions, each of which is 32 bits wide, and the following instructions are listed in alphabetical order.

Chapter 15 Appendix B Flathead Extended Command Terminology

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	0	0	0	0	0	1	0	0	0	0	1		rs1		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
rs1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1

Figure 15.1: DCACHE.CSW

15.2.1 SFENCE.VMAS - Virtual Memory Simulcast Instruction

Syntax: sfence.vmas

rs1,rs2 Operation:

Virtual memory invalidation and synchronization operations need to be broadcast to other cores in the cluster.

Execute permissions:

M mode/S mode

Exception:

Illegal instruction exception

Description:

rs1: virtual address, rs2: asid

- When rs1=x0, rs2=x0, invalidates all table entries in the TLB and broadcasts to other cores in the cluster
- When rs1!=x0 and rs2=x0, invalidate all table entries in the TLB that hit the rs1 virtual address and broadcast to other cores in the cluster.
- When rs1=x0 and rs2!=x0, invalidate all table entries in the TLB that hit the rs2 process number and broadcast to other cores in the cluster.
- When rs1!=x0 and rs2!=x0, invalidate all table entries in the TLB that hit the rs1 virtual address and the rs2 process number, and broadcast to the Other cores in the cluster.

mxstatus.theadisae=0, execution of this instruction generates an illegal instruction exception. mxstatus.theadisae=1, execution of this instruction in U mode generates an illegal instruction exception. **Instruction format:**

31	25 24	20 19	15 14	12 11	7 6	0
0000010	rs2	rs1	000	00000	0001011	

Chapter 15 Appendix B Flathead

Extended Command Terminology

15.2.2 SYNC - Synchronization Command

Synta

x: sync

opera

tion:

This instruction guarantees that all preceding instructions will retire earlier than this instruction and all following instructions will retire later than this instruction

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Command Format:

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

15.2.3 SYNC.I - Synchronized Clear Command

Synta

x:

sync.i

opera

tion:

This instruction guarantees that all preceding instructions will retire earlier than this instruction, all following instructions will retire later than this instruction, and the pipeline will be cleared when this instruction retires.

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Command Format:

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

15.2.4 SYNC.IS - Synchronized clear broadcast command

Chapter 15 Appendix B Flathead

Extended Command Terminology

Syntax

: sync.is

operat

ion:

Chapter 15 Appendix B Flathead Extended Command Terminology

This instruction guarantees that all preceding instructions will retire earlier than this instruction and all following instructions will retire later than this instruction, and when this instruction retires, it clears the pipeline and broadcasts the request to the other cores.

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Command Format:

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

15.2.5 SYNC.S - Synchronous Broadcast Command

Synta

x:

sync.s

opera

tion:

This instruction ensures that all preceding instructions retire earlier than this instruction and all following instructions retire later than this instruction, and broadcasts the request to the other cores.

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Command Format:

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

15.3 Appendix B-3 Arithmetic Instruction Terminology

The arithmetic instructions subset implements extensions to the arithmetic instructions, each of which is 32 bits wide. The

Chapter 15 Appendix B Flathead

Extended Command Terminology

following instructions are listed in alphabetical order.

15.3.1 ADDSL - Register Shift Summation Instruction

Grammar:

addsl rd rs1, rs2, imm2

Chapter 15 Appendix B Flathead Extended Command Terminology

Operation:

$rd \leftarrow rs1 + rs2 << imm2$

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Command Format:

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

15.3.2 MULA - Multiply and Accumulate Instruction

Grammar:

mula rd, rs1, rs2

Operation:

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

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Command Format:

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

15.3.3 MULAH - Low 16-bit Multiply-Accumulate Instruction

Grammar:

mulah rd, rs1, rs2

Operation:

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

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

Execute permissions:

M mode/S mode/U mode

Chapter 15 Appendix B Flathead Extended Command Terminology

Exception:

Illegal instruction exception

Command Format:

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

15.3.4 MULAW - Low 32-bit Multiply-Accumulate Instruction

Grammar:

mulaw rd, rs1, rs2

Operation:

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

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

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Command Format:

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

15.3.5 MULS - Multiply and Decrease Instructions

Grammar:

muls rd, rs1, rs2

Operation:

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

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Command Format:

Chapter 15 Appendix B Flathead Extended Command Terminology

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

15.3.6 MULSH - low 16-bit multiply-accumulate instruction

Grammar:

mulsh rd, rs1, rs2

Operation:

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

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

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Command Format:

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

15.3.7 MULSW - low 32-bit multiply-accumulate instruction

Grammar:

mulaw rd, rs1, rs2

Operation:

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

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

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Command Format:

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

Chapter 15 Appendix B Flathead Extended Command Terminology

15.3.8 MVEQZ - register is 0 Transmit Instruction

Grammar:

mveqz rd, rs1, rs2

Operation: if ($rs2 == 0$)

$rd \leftarrow rs1$

else

$rd \leftrightarrow rd$

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Command Format:

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

15.3.9 MVNEZ - Register Non-Zero Transmit Instruction

Grammar:

mvnez rd, rs1, rs2

Operation:

if ($rs2 != 0$)

$rd \leftarrow rs1$

else

$rd \leftrightarrow rd$

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Command Format:

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

Chapter 15 Appendix B Flathead

Extended Command Terminology

15.3.10 SRRI - Cyclic Right Shift Instruction

Grammar:

srri rd, rs1, imm6

Operation:

$rd \leftarrow rs1 >>> imm6$

rs1 Original value shifted right, left shift in right shift out bit

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Command Format:

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

15.3.11 SRRIW - low 32-bit circular right shift instruction

Grammar:

srriw rd, rs1, imm5

Operation:

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

rs1[31:0] original value shifted right,

left shifted in right shifted out bit

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Command Format:

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

15.4 Appendix B-4 Bit Operation Instruction Terminology

Chapter 15 Appendix B Flathead

Extended Command Terminology

The bit operation instruction subset implements extensions to the bit operation instructions, each of which is 32 bits wide.

Chapter 15 Appendix B Flathead Extended Command Terminology

The following commands are listed in alphabetical order.

15.4.1 EXT - Register Contiguous Bit Extract Symbol Bit Extension Instruction

Grammar:

ext rd, rs1, imm1,imm2

Action:

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

to enforce permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Description:

If $\text{imm1} < \text{imm2}$, the behavior of the instruction is unpredictable

Command Format:

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

15.4.2 EXTU - Register Continuous Bit Extract Zero Extension Instruction

Grammar:

extu rd, rs1, imm1,imm2

Action:

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

execute permission:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Description:

If $\text{imm1} < \text{imm2}$, the behavior of the instruction is unpredictable

Command Format:

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

Chapter 15 Appendix B Flathead

Extended Command Terminology

15.4.3 FF0 - Fast Find 0 command

Syntax:

ff0 rd, rs1

Operatio

n:

Starting from the highest bit of rs1, find the first bit that is 0, and write the result back to rd. if the highest bit of rs1 is 0, the result is 0. if there is no 0 in rs1, the result is 64.

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Command Format:

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

15.4.4 FF1 - Quick Find 1 command

Syntax:

ff1 rd, rs1

Operatio

n:

Starting from the highest bit of rs1, find the first bit that is a 1 and write the index of that bit back to rd. if the highest bit of rs1 is a 1, the result is 0, if there is no 1 in rs1, the result is 64

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Command Format:

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

15.4.5 REV - Reverse Byte Order Instruction

Grammar:

Chapter 15 Appendix B Flathead
Extended Command Terminology
rev rd, rs1

Chapter 15 Appendix B Flathead Extended Command Terminology

Operation:

```

rd[63:56] ← rs1[7:0]
rd[55:48] ← rs1[15:8]
rd[47:40] ← rs1[23:16]
rd[39:32] ← rs1[31:24]
rd[31:24] ← rs1[39:32]
rd[23:16] ← rs1[47:40]
rd[15:8] ← rs1[55:48]
rd[7:0] ← rs1[63:56]

```

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Command Format:

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

15.4.6 REVW--Lower 32-Bit Byte Reverse Order Instruction

Syntax:

revw rd, rs1

Operation:

```

tmp [31:24] ← rs1[7:0]
tmp [23:16] ← rs1[15:8]
tmp [15:8] ← rs1[23:16]
tmp [7:0] ← rs1[31:24]
rd ← sign_extend(tmp[31:0])

```

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Chapter 15 Appendix B Flathead Extended Command Terminology

Command Format:

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

15.4.7 TST - bit 0 test command

Grammar:

tst rd, rs1, imm6

Operation:

if(rs1[imm6] ==

1) rd←1

else

rd←0

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Command Format:

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

15.4.8 TSTNBZ - byte 0 test instruction

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

```

Chapter 15 Appendix B Flathead Extended Command Terminology

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

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Command Format:

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

15.5 Appendix B-5 Memory Instruction Terminology

The memory instruction subset implements extensions to the memory instructions, each of which is 32 bits wide. The following instructions are listed in alphabetical order.

15.5.1 FLRD - Floating Point Register Shift Double Word Load Instruction

Grammar:

`fldr rd, rs1, rs2, imm2`

Operation:

`rd ← mem[(rs1+rs2<<imm2)+7: (rs1+rs2<<imm2)]`

Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Unaligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception, Illegal Instruction Exception

Description:

This instruction generates an illegal instruction exception when `mxstatus.theadisae=1b00` or `mstatus.fs =2b00`

Command Format:

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

15.5.2 FLRW - Floating Point Register Shift Word Load Instruction

Grammar:

Chapter 15 Appendix B Flathead Extended Command Terminology

flrw rd, rs1, rs2, imm2

Operation:

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

Execute permissions:

M mode/S mode/U mode

Exception:

Load instruction unaligned access exception, load instruction access error exception, load instruction page error exception, illegal instruction exception

Description:

This instruction generates an illegal instruction exception when mxstatus.theadisae=1^{b0} or mstatus.fs = 2^{b00}

Command Format:

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

15.5.3 FLURD - Floating Point Register Lower 32 Bit Shifted Double Word Load Instruction

Grammar:

flurd rd, rs1, rs2, imm2

Operation:

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

Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception, Illegal Instruction Exception

Description:

rs2[31:0] is an unsigned number, high [63:32] complement 0 for address arithmetic mxstatus.theadisae = 1^{b0} or mstatus.fs = 2^{b00}. This instruction generates an illegal instruction exception.

Format:

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

Chapter 15 Appendix B Flathead

Extended Command Terminology

15.5.4 FLURW - Floating Point Register Low 32-Bit Shift Word Load Instruction

Grammar:

flurw rd, rs1, rs2, imm2

Operation:

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

Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception, Illegal Instruction Exception

Description:

rs2[31:0] is an unsigned number, high [63:32] complement 0 for address arithmetic mxstatus.theadisae = 1^{b0} or mstatus.fs = 2^{b00}. This instruction generates an illegal instruction exception **Instruction**

Format:

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

15.5.5 FSRD - Floating Point Register Shift Double Word Store Instruction

Grammar:

fsrd rd, rs1, rs2, imm2

Operation:

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

Execute permissions:

M mode/S mode/U mode

Exception:

Memory Instruction Non-Aligned Access Exception, Memory Instruction Access Error Exception, Memory Instruction Page Error Exception, Illegal Instruction Exception

Description:

This instruction generates an illegal instruction exception when mxstatus.theadisae=1^{b0} or mstatus.fs =2^{b00}

Command Format:

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

15.5.6 FSRW - Floating Point Register Shift Word Store Instruction

Grammar:

fsrw rd, rs1, rs2, imm2

Operation:

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

Execute permissions:

M mode/S mode/U mode

Exception:

Memory Instruction Non-Aligned Access Exception, Memory Instruction Access Error Exception, Memory Instruction Page Error Exception, Illegal Instruction Exception

Description:

This instruction generates an illegal instruction exception when mxstatus.theadisae=1 \wedge b0 or mstatus.fs = 2 \wedge b0

Command Format:

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

15.5.7 FSURD - Floating Point Register Lower 32 Bit Shifted Double Word Store Instruction

Grammar:

fsurd rd, rs1, rs2, imm2

Operation:

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

Execute permissions:

M mode/S mode/U mode

Exception:

Memory Instruction Non-Aligned Access Exception, Memory Instruction Access Error Exception, Memory Instruction Page Error Exception, Illegal Instruction Exception

Description:

rs2[31:0] is an unsigned number, high [63:32] complement 0 for address arithmetic mxstatus.theadisae = 1 \wedge b0 or mstatus.fs = 2 \wedge b0 This instruction generates an illegal instruction exception

Chapter 15 Appendix B Flathead

Extended Command Terminology

Format:

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

Chapter 15 Appendix B Flathead

Extended Command Terminology

15.5.8 FSURW - Floating Point Register Lower 32 Bit Shift Word Store Instruction

Grammar:

fsurw rd, rs1, rs2, imm2

Operation:

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

Execute permissions:

M mode/S mode/U mode

Exception:

Memory Instruction Non-Aligned Access Exception, Memory Instruction Access Error Exception, Memory Instruction Page Error Exception, Illegal Instruction Exception

Description:

$\text{rs2}[31:0]$ is an unsigned number, high [63:32] complement 0 for address arithmetic $\text{mxstatus.theadisae} = 1^{\text{b}}0$ or $\text{mstatus.fs} = 2^{\text{b}}00$. This instruction generates an illegal instruction exception **Instruction**

Format:

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

15.5.9 LBIA - Signed Bit Extended Byte Load Base Address Increment Instruction

Grammar:

lbia rd, (rs1), imm5,imm2

Operation:

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

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

Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception, Illegal Instruction Exception.

Description:

Chapter 15 Appendix B Flathead Extended Command Terminology

rd and rs1 are not equal

Command Format:

Chapter 15 Appendix B Flathead Extended Command Terminology

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

15.5.10 LBIB - Base Address Self-Incrementing Symbolic Bit Extended Byte Load Instruction

Grammar:

lbib rd, (rs1), imm5,imm2

Operation:

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

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

Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception, Illegal Instruction Exception.

Description:

rd and rs1 are not equal

Command Format:

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

15.5.11 LBUIA - zero extended byte load base address self-increment instruction

Grammar:

lbuia rd, (rs1), imm5,imm2

Operation:

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

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

Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load

Chapter 15 Appendix B Flathead

Extended Command Terminology

Instruction Page Error Exception, Illegal Instruction Exception.

Description:

rd and rs1 are not equal

Chapter 15 Appendix B Flathead Extended Command Terminology

Command Format:

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

15.5.12 LBUIB - Base Address Self-Incrementing Zero Extended Byte Load Instruction

Grammar:

lbuib rd, (rs1), imm5,imm2

Operation:

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

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

Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception, Illegal Instruction Exception.

Description:

rd and rs1 are not equal

Command Format:

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

15.5.13 LDD - Load Dual Register Instruction

Grammar:

lld rd1,rd2, (rs1),imm2

Operation:

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

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

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

Execute permissions:

M mode/S mode/U mode

Exception:



Chapter 15 Appendix B Flathead

Extended Command Terminology

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception, Illegal Instruction Exception

Chapter 15 Appendix B Flathead Extended Command Terminology

Description:

rd_1, rd_2, rs_1 are not equal to each other.

Command Format:

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

15.5.14 LDIA - Signed Bit Extended Double Word Load Base Address Increment Instruction

Grammar:

ldia rd, (rs1), imm5,imm2

Operation:

$rd \leftarrow \text{sign_extend}(\text{mem}[rs_1+7:rs_1]) rs_1$
 $\leftarrow rs_1 + \text{sign_extend}(imm5 << imm2)$

Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception, Illegal Instruction Exception.

Description:

rd and rs_1 are not equal

Command Format:

31	27	26 25	24	20	19	15	14	12	11	7	6	0
01111	imm2		imm5		rs1		100		rd		0001011	

15.5.15 LDIB - Base Address Self-Incrementing Symbolic Bit Extended Double Word Load Instruction

Grammar:

ldib rd, (rs1), imm5,imm2

Operation:

$rs_1 \leftarrow rs_1 + \text{sign_extend}(imm5 << imm2)$
 $rd \leftarrow \text{sign_extend}(\text{mem}[rs_1+7:rs_1])$

Execute permissions:

Chapter 15 Appendix B Flathead

Extended Command Terminology

M mode/S mode/U mode

Exception:

Chapter 15 Appendix B Flathead Extended Command Terminology

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception, Illegal Instruction Exception.

Description:

rd and rs1 are not equal

Command Format:

31	27	26 25	24	20 19	15	14	12 11	7	6	0
01101	imm2	imm5		rs1	100		rd		0001011	

15.5.16 LHIA - Signed Bit Extended Halfword Load Base Address Increment Instruction

Grammar:

lhia rd, (rs1), imm5,imm2

Operation:

$rd \leftarrow \text{sign_extend}(\text{mem}[rs1+1:rs1]) \text{ rs1}$
 $\leftarrow rs1 + \text{sign_extend}(imm5 << imm2)$

Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception, Illegal Instruction Exception.

Description:

rd and rs1 are not equal

Command Format:

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

15.5.17 LHIB - Base Address Self-Incrementing Symbolic Bit Extended Half- Word Load Instruction

Grammar:

lhib rd, (rs1), imm5,imm2

Operation:

$rs1 \leftarrow rs1 + \text{sign_extend}(imm5 << imm2)$
 $rd \leftarrow \text{sign_extend}(\text{mem}[rs1+1:rs1])$

Chapter 15 Appendix B Flathead

Extended Command Terminology

Execute permissions:

M mode/S mode/U mode

Chapter 15 Appendix B Flathead Extended Command Terminology

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception, Illegal Instruction Exception.

Description:

rd and rs1 are not equal

Command Format:

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

15.5.18 LHUIA - Zero Extended Half-Word Load Base Address Increment Instruction

Grammar:

lhuia rd, (rs1), imm5,imm2

Operation:

$rd \leftarrow \text{zero_extend}(\text{mem}[rs1+1:rs1])\ rs1$
 $\leftarrow rs1 + \text{sign_extend}(imm5 << imm2)$

Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception, Illegal Instruction Exception.

Description:

rd and rs1 are not equal

Command Format:

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

15.5.19 LHUIB - Base Address Self-Incrementing Zero Extended Half-Word Load Instruction

Grammar:

lhuib rd, (rs1), imm5,imm2

Operation:

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

Chapter 15 Appendix B Flathead

Extended Command Terminology

`rd ← zero_extend(mem[rs1+1:rs1])`

Execute permissions:

Chapter 15 Appendix B Flathead Extended Command Terminology

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception, Illegal Instruction Exception.

Description:

rd and rs1 are not equal

Command Format:

31	27	26 25	24	20 19	15	14	12 11	7 6	0
10101	imm2		imm5		rs1		100	rd	0001011

15.5.20 LRB - Register Shift Symbol Bit Extended Byte Load Instruction

Grammar:

lrb rd, rs1, rs2, imm2

Operation:

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

Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception, Illegal Instruction Exception

Command Format:

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

15.5.21 LRBU - Register Shift Zero Extension Extended Byte Load Instruction

Grammar:

lrbu rd, rs1, rs2, imm2

Operation:

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

Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load



Chapter 15 Appendix B Flathead

Extended Command Terminology

Instruction Page Error Exception, Illegal Instruction Exception

Chapter 15 Appendix B Flathead Extended Command Terminology

Command Format:

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

15.5.22 LRD - Register Shift Double Word Load Instruction

Grammar:

lrd rd, rs1, rs2, imm2

Operation:

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

Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception, Illegal Instruction Exception

Command Format:

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

15.5.23 LRH - Register Shift Symbol Bit Extended Half-Word Load Instruction

Grammar:

lrh rd, rs1, rs2, imm2

Operation:

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

Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception, Illegal Instruction Exception

Command Format:

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

Chapter 15 Appendix B Flathead

Extended Command Terminology

15.5.24 LRHU - Register Shift Zero Extension Extension Half Word Load Instruction

Grammar:

lrhu rd, rs1, rs2, imm2

Operation:

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

Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception, Illegal Instruction Exception

Command Format:

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

15.5.25 LRW - Register Shift Symbol Bit Extended Word Load Instruction

Grammar:

lrw rd, rs1, rs2, imm2

Operation:

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

Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception, Illegal Instruction Exception

Command Format:

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

15.5.26 LRWU - Register Shift Zero Extension Extended Word Load Instruction

Grammar:

lrwu rd, rs1, rs2, imm2

Chapter 15 Appendix B Flathead
Extended Command Terminology
Operation:

Chapter 15 Appendix B Flathead Extended Command Terminology

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

Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception, Illegal Instruction Exception

Command Format:

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

15.5.27 LURB--Lower 32 bits of the register Shifted Signed Bit Extended Byte Load Instruction

Grammar:

lurb rd, rs1, rs2, imm2

Operation:

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

Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception, Illegal Instruction Exception

Description:

rs2[31:0] is an unsigned number, high [63:32] by 0 for address operations

Command Format:

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

15.5.28 LURBU--Lower 32 bits of the register Shift Zero Extended Byte Load Instruction

Grammar:

lurbu rd, rs1, rs2, imm2

Operation:

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

Chapter 15 Appendix B Flathead

Extended Command Terminology

Execute permissions:

M mode/S mode/U mode

Chapter 15 Appendix B Flathead Extended Command Terminology

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception, Illegal Instruction Exception

Description:

$rs2[31:0]$ is an unsigned number, high [63:32] by 0 for address operations

Command Format:

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

15.5.29 LURD--Low 32-bit Register Shift Double Word Load Instruction

Grammar:

lurd rd, rs1, rs2, imm2

Operation:

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

Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception, Illegal Instruction Exception

Description:

$rs2[31:0]$ is an unsigned number, high [63:32] by 0 for address operations

Command Format:

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

15.5.30 LURH--Lower 32 bits of register Shifted Sign Bit Extended Half-Word Load Instruction

Grammar:

lurh rd, rs1, rs2, imm2

Operation:

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

Execute permissions:



Chapter 15 Appendix B Flathead
Extended Command Terminology
M mode/S mode/U mode

Chapter 15 Appendix B Flathead Extended Command Terminology

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception, Illegal Instruction Exception

Description:

$rs2[31:0]$ is an unsigned number, high [63:32] by 0 for address operations

Command Format:

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

15.5.31 LURHU--Lower 32 bits of register Shift Zero Extended Half-Word Load Instruction

Grammar:

lurhu rd, rs1, rs2, imm2

Operation:

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

Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception, Illegal Instruction Exception

Description:

$rs2[31:0]$ is an unsigned number, high [63:32] by 0 for address operations

Command Format:

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

15.5.32 LURW--Lower 32-bit shifted symbolic bits of register Extended Word Load instruction

Grammar:

lurw rd, rs1, rs2, imm2

Operation:

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

**Chapter 15 Appendix B Flathead
Extended Command Terminology**
(rs1+rs2[31:0]<<imm2)])

Execute permissions:

Chapter 15 Appendix B Flathead Extended Command Terminology

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception, Illegal Instruction Exception

Description:

$rs2[31:0]$ is an unsigned number, high [63:32] by 0 for address operations

Command Format:

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

15.5.33 LURWU--Low 32-bit Register Shift Zero Extended Word Load Instruction

Grammar:

lwd rd1, rd2, (rs1),imm2

Operation:

$address \leftarrow rs1 + zero_extend(imm2 << 3)$

$rd1 \leftarrow sign_extend(mem[address+3: address])$ rd2

$\leftarrow sign_extend(mem[address+7: address+4])$

Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception, Illegal Instruction Exception

Description:

rd1,rd2 ,rs1 are not equal to each other.

Command Format:

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

15.5.34 LWD - Signed Bit Extended Dual Register Word Load Instruction

Grammar:

lwd rd, imm7(rs1)

Operation:

Chapter 15 Appendix B Flathead Extended Command Terminology

$\text{address} \leftarrow \text{rs1} + \text{sign_extend}(\text{imm}_7)$
 $\text{rd} \leftarrow \text{sign_extend}(\text{mem}[\text{address}+31: \text{address}])$
 $\text{rd}+1 \leftarrow \text{sign_extend}(\text{mem}[\text{address}+63: \text{address}+32])$

Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception, Illegal Instruction Exception

Command Format:

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

15.5.35 LWIA - Signed Bit Extended Word Load Base Address Increment Instruction

Grammar:

lwia rd, (rs1), imm5,imm2

Operation:

$\text{rd} \leftarrow \text{sign_extend}(\text{mem}[\text{rs1}+3:\text{rs1}]) \text{ rs1}$
 $\leftarrow \text{rs1} + \text{sign_extend}(\text{imm}_5 \ll \text{imm}_2)$

Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception, Illegal Instruction Exception.

Description:

rd and rs1 are not equal

Command Format:

31	27	26 25	24	20 19	15	14	12	11	7	6	0
01011		imm2		imm5		rs1		100		rd	

15.5.36 LWIB - Base Address Self-Incrementing Symbolic Bit Extended Word Load Instruction

Chapter 15 Appendix B Flathead Extended Command Terminology

Grammar:

lwib rd, (rs1), imm5,imm2

Operation:

Chapter 15 Appendix B Flathead Extended Command Terminology

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

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

Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception, Illegal Instruction Exception.

Description:

rd and rs1 are not equal

Command Format:

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

15.5.37 LWUD - Zero Extended Dual Register Word Load Instruction

Grammar:

lwud rd1,rd2, (rs1),imm2

Operation:

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

$rd1 \leftarrow \text{zero_extend}(\text{mem}[\text{address}+3: \text{address}])$ rd2

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

Execute permission:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception, Illegal Instruction Exception

Description:

rd1,rd2 ,rs1 are not equal to each other.

Command Format:

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

Chapter 15 Appendix B Flathead

Extended Command Terminology

15.5.38 LWUIA - zero extended word load base address self-increment instruction

Grammar:

lwuia rd, (rs1), imm5,imm2

Operation:

$$\begin{aligned} \text{rd} &\leftarrow \text{zero_extend}(\text{mem}[\text{rs1}+3:\text{rs1}]) \text{ rs1} \\ &\leftarrow \text{rs1} + \text{sign_extend}(\text{imm5} \ll \text{imm2}) \end{aligned}$$

Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception, Illegal Instruction Exception.

Description:

rd and rs1 are not equal

Command Format:

31	27	26 25	24	20 19	15	14	12	11	7	6	0
11011	imm2		imm5		rs1		100		rd		0001011

15.5.39 LWUIB - Base Address Self-Incrementing Zero Extended Word Load Instruction

Grammar:

lwuib rd, (rs1), imm5,imm2

Operation:

$$\begin{aligned} \text{rs1} &\leftarrow \text{rs1} + \text{sign_extend}(\text{imm5} \ll \text{imm2}) \\ \text{rd} &\leftarrow \text{zero_extend}(\text{mem}[\text{rs1}+3:\text{rs1}]) \end{aligned}$$

Execute permissions:

M mode/S mode/U mode

Exception:

Load Instruction Non-Aligned Access Exception, Load Instruction Access Error Exception, Load Instruction Page Error Exception, Illegal Instruction Exception.

Description:

rd and rs1 are not equal

Chapter 15 Appendix B Flathead

Extended Command Terminology

Command Format:

Chapter 15 Appendix B Flathead Extended Command Terminology

31	27	26 25	24	20 19	15 14	12 11	7 6	0
11001	imm2	imm5		rs1	100	rd	0001011	

15.5.40 SBIA - Byte Store Base Address Increment Instruction

Grammar:

sbia rs2, (rs1), imm5,imm2

Operation:

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

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

Execute permissions:

M mode/S mode/U mode

Exception:

Memory Instruction Non-Aligned Access Exception, Memory Instruction Access Error Exception, Memory Instruction Page Error Exception, Illegal Instruction Exception

Command Format:

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

15.5.41 SBIB - Base Address Self-Incrementing Byte Store Instruction

Grammar:

sbib rs2, (rs1), imm5,imm2

Operation:

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

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

Execute permissions:

M mode/S mode/U mode

Exception:

Memory Instruction Non-Aligned Access Exception, Memory Instruction Access Error Exception, Memory Instruction Page Error Exception, Illegal Instruction Exception

Command Format:

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

Chapter 15 Appendix B Flathead

Extended Command Terminology

15.5.42 SDD - Double Register Storage Directive

Grammar:

sdd rd1,rd2, (rs1),imm2

Operation:

address \leftarrow rs1 + zero_extend(imm2<<4)

mem[address+7:address] \leftarrow rd1

mem[address+15:address+8] \leftarrow rd2

Execute permissions:

M mode/S mode/U mode

Exception:

Memory Instruction Non-Aligned Access Exception, Memory Instruction Access Error Exception, Memory Instruction Page Error Exception, Illegal Instruction Exception

Command Format:

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

15.5.43 SDIA - Double Word Storage Base Address Self-Incrementing Instruction

Grammar:

sdia rs2, (rs1), imm5,imm2

Operation:

mem[rs1+7:rs1] \leftarrow rs2[63:0]

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

Execute permissions:

M mode/S mode/U mode

Exception:

Memory Instruction Non-Aligned Access Exception, Memory Instruction Access Error Exception, Memory Instruction Page Error Exception, Illegal Instruction Exception

Command Format:

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

Chapter 15 Appendix B Flathead

Extended Command Terminology

15.5.44 SDIB - Self-Incrementing Double Word Storage Instruction for Base Addresses

Grammar:

sdib rs2, (rs1), imm5,imm2

Operation:

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

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

Execute permissions:

M mode/S mode/U mode

Exception:

Memory Instruction Non-Aligned Access Exception, Memory Instruction Access Error Exception, Memory Instruction Page Error Exception, Illegal Instruction Exception

Command Format:

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

15.5.45 SHIA - Half-Word Storage Base Address Increment Instruction

Grammar:

shia rs2, (rs1), imm5,imm2

Operation:

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

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

Execute permissions:

M mode/S mode/U mode

Exception:

Memory Instruction Non-Aligned Access Exception, Memory Instruction Access Error Exception, Memory Instruction Page Error Exception, Illegal Instruction Exception

Command Format:

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

15.5.46 SHIB - Base Address Self-Incrementing Half-Word Storage Instruction

Chapter 15 Appendix B Flathead

Extended Command Terminology

Grammar:

Chapter 15 Appendix B Flathead Extended Command Terminology

shib rs2, (rs1), imm5,imm2

Operation:

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

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

Execute permissions:

M mode/S mode/U mode

Exception:

Memory Instruction Non-Aligned Access Exception, Memory Instruction Access Error Exception, Memory Instruction Page Error Exception, Illegal Instruction Exception

Command Format:

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

15.5.47 SRB - Register Shift Byte Store Instruction

Grammar:

srb rd, rs1, rs2, imm2

Operation:

$mem[(rs1+rs2<<imm2)] \leftarrow rd[7:0]$

Execute permissions:

M mode/S mode/U mode

Exception:

Memory Instruction Non-Aligned Access Exception, Memory Instruction Access Error Exception, Memory Instruction Page Error Exception, Illegal Instruction Exception

Command Format:

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

15.5.48 SRD - Register Shift Double Word Store Instruction

Grammar:

srd rd, rs1, rs2, imm2

Operation:

$mem[(rs1+rs2<<imm2)+7: (rs1+rs2<<imm2)] \leftarrow rd[63:0]$

Execute permissions:

Chapter 15 Appendix B Flathead Extended Command Terminology

M mode/S mode/U mode

Exception:

Memory Instruction Non-Aligned Access Exception, Memory Instruction Access Error Exception, Memory Instruction Page Error Exception, Illegal Instruction Exception

Command Format:

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

15.5.49 SRH - Register Shift Half-Word Store Instruction

Grammar:

srh rd, rs1, rs2, imm2

Operation:

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

Execute permissions:

M mode/S mode/U mode

Exception:

Memory Instruction Non-Aligned Access Exception, Memory Instruction Access Error Exception, Memory Instruction Page Error Exception, Illegal Instruction Exception

Command Format:

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

15.5.50 SRW - Register Shift Word Store Instruction

Grammar:

srw rd, rs1, rs2, imm2

Operation:

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

Execute permissions:

M mode/S mode/U mode

Exception:

Memory Instruction Non-Aligned Access Exception, Memory Instruction Access Error Exception, Memory Instruction Page Error Exception, Illegal Instruction Exception

Command Format:

Chapter 15 Appendix B Flathead Extended Command Terminology

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

15.5.51 SURB - Register Lower 32 Bit Shift Byte Store Instruction

Grammar:

surb rd, rs1, rs2, imm2

Operation:

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

Execute permissions:

M mode/S mode/U mode

Exception:

Memory Instruction Non-Aligned Access Exception, Memory Instruction Access Error Exception, Memory Instruction Page Error Exception, Illegal Instruction Exception

Description:

$\text{rs2}[31:0]$ is an unsigned number, upper [63:32] by 0 for address operations.

Command Format:

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

15.5.52 SURD - Register Lower 32 Bit Shift Double Word Store Instruction

Grammar:

surd rd, rs1, rs2, imm2

Operation:

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

Execute permissions:

M mode/S mode/U mode

Exception:

Memory Instruction Non-Aligned Access Exception, Memory Instruction Access Error Exception, Memory Instruction Page Error Exception, Illegal Instruction Exception

Description:

$\text{rs2}[31:0]$ is an unsigned number, upper [63:32] by 0 for address operations.

Command Format:

Chapter 15 Appendix B Flathead Extended Command Terminology

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

15.5.53 SURH--Register Low 32-bit Shift Half-Word Store Instruction

Grammar:

surh rd, rs1, rs2, imm2

Operation:

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

Execute permissions:

M mode/S mode/U mode

Exception:

Memory Instruction Non-Aligned Access Exception, Memory Instruction Access Error Exception, Memory Instruction Page Error Exception, Illegal Instruction Exception

Description:

$\text{rs2}[31:0]$ is an unsigned number, upper [63:32] by 0 for address operations.

Command Format:

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

15.5.54 SURW - Register Lower 32 Bit Shift Word Store Instruction

Grammar:

surw rd, rs1, rs2, imm2

Operation:

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

Execute permissions:

M mode/S mode/U mode

Exception:

Memory Instruction Non-Aligned Access Exception, Memory Instruction Access Error Exception, Memory Instruction Page Error Exception, Illegal Instruction Exception

Description:

$\text{rs2}[31:0]$ is an unsigned number, upper [63:32] by 0 for address operations.

Command Format:

Chapter 15 Appendix B Flathead Extended Command Terminology

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

15.5.55 SWIA - Word Storage Base Address Incrementation Instruction

Grammar:

swia rs2, (rs1), imm5,imm2

Operation:

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

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

Execute permissions:

M mode/S mode/U mode

Exception:

Memory Instruction Non-Aligned Access Exception, Memory Instruction Access Error Exception, Memory Instruction Page Error Exception, Illegal Instruction Exception

Command Format:

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

15.5.56 SWIB - Base Address Self-Incrementing Word Store Instruction

Grammar:

swib rs2, (rs1), imm5,imm2

Operation:

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

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

Execute permissions:

M mode/S mode/U mode

Exception:

Memory Instruction Non-Aligned Access Exception, Memory Instruction Access Error Exception, Memory Instruction Page Error Exception, Illegal Instruction Exception

Command Format:

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

Chapter 15 Appendix B Flathead

Extended Command Terminology

15.5.57 SWD - dual register low 32-bit store instruction

Grammar:

swd rd1,rd2,(rs1),imm2

Operation:

address \leftarrow rs1 + zero_extend(imm2<<3)

mem[address+3:address] \leftarrow rd1[31:0]

mem[address+7:address+4] \leftarrow rd2[31:0]

Execute permissions:

M mode/S mode/U mode

Exception:

Memory Instruction Non-Aligned Access Exception, Memory Instruction Access Error Exception, Memory Instruction Page Error Exception, Illegal Instruction Exception

Command Format:

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

15.6 Appendix B-6 Floating-Point Half-Precision Instruction Terminology

The floating-point half-precision instruction subset implements floating-point half-precision support, each instruction is 32 bits wide, and the following instructions are listed in alphabetical order.

15.6.1 FADD.H - half-precision floating-point addition instruction

Grammar:

fadd.h fd, fs1, fs2, rm

Operation:

fd \leftarrow fs1 + fs2

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Impact Flag Bits:

Chapter 15 Appendix B Flathead Extended Command Terminology

Floating-point status bits NV/OF/NX

Chapter 15 Appendix B Flathead Extended Command Terminology

Description:

rm determines the rounding mode.

- $3^{\blacktriangleright} b000$: rounds to an even number, corresponding to the assembly instruction fadd.h fd, fs1,fs2,rne.
- $3^{\blacktriangleright} b001$: Rounding to zero, corresponding to the assembly instruction fadd.h fd, fs1,fs2,rtz.
- $3^{\blacktriangleright} b010$: Rounding to negative infinity, corresponding to the assembly instruction fadd.h fd, fs1,fs2,rdn.
- $3^{\blacktriangleright} b011$: Rounding to positive infinity, corresponding to the assembly instruction fadd.h fd, fs1,fs2,rup.
- $3^{\blacktriangleright} b100$: Round to the nearest larger value, corresponding to the assembly instruction fadd.h fd, fs1,fs2,rmm.
- $3^{\blacktriangleright} b101$: Not used at this time, this code will not appear.
- $3^{\blacktriangleright} b110$: Not used at this time, this code will not appear.
- $3^{\blacktriangleright} b111$: Dynamic rounding, depending on the rm bit in the floating-point control register fcsr, determines the rounding mode, corresponding to the assembly instructions fadd.h fd, fs1,fs2.

Command Format:

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

15.6.2 FCLASS.H - 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)

```

Chapter 15 Appendix B Flathead Extended Command Terminology

```
rd ← 64•h10  
if ( fs1 = +subnorm)
```

Chapter 15 Appendix B Flathead Extended Command Terminology

```

rd ← 64▼h20
if ( fs1 = +norm)
    rd ← 64▼h40
if ( fs1 = +inf)
    rd ← 64▼h80
if ( fs1 = sNaN)
    rd ← 64▼h100
if ( fs1 = qNaN)
    rd ← 64▼h200

```

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal

Instruction

Exceptionally

Affected

Flag Bit:

None

Command Format:

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

15.6.3 FCVT.D.H - half-precision floating-point to double-precision floating-point conversion instruction

Syntax:

fcvt.d.h fd, fs1

Operation:

```
fd ← half_convert_to_double(fs1)
```

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal

Chapter 15 Appendix B Flathead Extended Command Terminology

Instruction

Exceptionally

Affected

Flag **Bit:**

None

Chapter 15 Appendix B Flathead Extended Command Terminology

Command Format:

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

15.6.4 FCVT.H.D - Double Precision Floating Point to Half Precision Floating Point Instruction

Grammar:

fcvt.h.d fd, fs1, rm

Operation:

fd \leftarrow double_convert_to_half(fs1)

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Impact Flag Bits:

Floating-point status bits NV/OF/UF/NX

Description:

rm determines the rounding mode.

- 3[▼]b00: Rounding to an even number, corresponding to the assembly instruction fcvt.h.d fd,fs1,rne.
- 3[▼]b001: Rounding to zero, corresponding to the assembly instruction fcvt.h.d fd,fs1,rtz.
- 3[▼]b010: Rounding to negative infinity, corresponding to the assembly instruction fcvt.h.d fd,fs1,rdn.
- 3[▼]b011: Rounding to positive infinity, corresponding to the assembly instruction fcvt.h.d fd,fs1,rup.
- 3[▼]b100: Round to the nearest large value, corresponding to the assembly instruction fcvt.h.d fd,fs1,rmm.
- 3[▼]b101: Not used at this time, this code will not appear.
- 3[▼]b110: Not used at this time, this code will not appear.
- 3[▼]b111: Dynamic rounding, which determines the rounding mode based on the rm bit in the floating-point control register fcsr, corresponding to the assembly instruction fcvt.h.d fd, fs1.

Command Format:

31	25 24	20 19	15 14	12 11	7 6	0
www.t-h	0100010	00001	fs1	rm	fd	1010011

Chapter 15 Appendix B Flathead Extended Command Terminology

15.6.5 FCVT.H.L - Signed Long Integer to Half-Precision Floating Point Instruction

Grammar:

fcvt.h.l fd, rs1, rm

Operation:

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

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Impact Flag Bits:

Floating Point Status Bit NX/OF

Description:

rm determines the rounding mode.

- 3⁷b000: Rounding to an even number as close as possible, corresponding to the assembly instruction fcvt.h.l fd,rs1,rne.
- 3⁷b001: Rounding to zero, corresponding to the assembly instruction fcvt.h.l fd,rs1,rtz.
- 3⁷b010: Rounding to negative infinity, corresponding to the assembly instruction fcvt.h.l fd,rs1,fdn.
- 3⁷b011: Rounding to positive infinity, corresponding to the assembly instruction fcvt.h.l fd,rs1,rup.
- 3⁷b100: Rounding to large values as close as possible, corresponding to the assembly instruction fcvt.h.l fd,rs1,rmm.
- 3⁷b101: Not used at this time, this code will not appear.
- 3⁷b110: Not used at this time, this code will not appear.
- 3⁷b111: Dynamic rounding, which determines the rounding mode based on the rm bit in the floating-point control register fcsr, corresponding to the assembly instruction fcvt.h.l fd, rs1.

Command Format:

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

15.6.6 FCVT.H.LU - Unsigned Long Integer to Half-Precision Floating Point

Chapter 15 Appendix B Flathead
Extended Command Terminology
Instruction

Grammar:

fcvt.h.lu fd, rs1, rm

Operation:

Chapter 15 Appendix B Flathead Extended Command Terminology

`fd ← unsigned_long_convert_to_half_fp(rs1)`

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception

Impact Flag Bits:

Floating Point Status Bit NX/OF

Description:

rm determines the rounding mode.

- 3[▼]b000: Rounding to an even number as close as possible, corresponding to the assembly instruction `fcvt.h.lu fd,rs1,rne`.
- 3[▼]b001: Rounding to zero, corresponding to the assembly instruction `fcvt.h.lu fd, rs1,rtz`.
- 3[▼]b010: Rounding to negative infinity, corresponding to the assembly instruction `fcvt.h.lu fd, rs1,fdn`.
- 3[▼]b011: Rounding to positive infinity, corresponding to the assembly instruction `fcvt.h.lu fd, rs1,rup`.
- 3[▼]b100: Rounding to large values as close as possible, corresponding to assembly instructions `fcvt.h.lu fd, rs1,rmm`.
- 3[▼]b101: Not used at this time, this code will not appear.
- 3[▼]b110: Not used at this time, this code will not appear.
- 3[▼]b111: Dynamic rounding, which determines the rounding mode based on the rm bit in the floating-point control register fcsr, corresponding to the assembly instruction `fcvt.h.lu fd, rs1`.

Command Format:

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

15.6.7 FCVT.H.S - Single Precision Floating Point to Half Precision Floating Point Instruction

Grammar:

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

Operation:

`fd ← single_convert_to_half(fs1)`

Chapter 15 Appendix B Flathead

Extended Command Terminology

Execute permissions:

M mode/S mode/U mode

Exception:

Illegal instruction exception