CAHPv3 Instruction Set Specification

Naoki Matsumoto, Ryotaro Banno, Kotaro Matsuoka2020/09/21

1 Programmers' Model for CAHPv3 ISA

1.1 Integer Registers

15	0
x0 / ra(return address)	
x1 / sp(stack pointer)	
x2 / fp(frame pointer)	
x3 / s0	
x4 / s1	
x5 / s2	
x6 / s3	
x7 / s4	
x8 / a0	
x9 / a1	
x10 / a2	
x11 / a3	
x12 / a4	
x13 / a5	
x14 / t0	
x15 / t1	
16	
15	0
рс	
16	

Figure 1: CAHPv3 integer register state.

2 24-bit Length Instructions

2.1 Integer Computational Instructions

Integer Register-Register Instructions

23	20 19 16	6 15 12	11 8	3 7 0
0000	rs2	rs1	rd	opcode
4	4	4	4	8
	src2	$\operatorname{src}1$	dest	ADD
	src2	$\operatorname{src}1$	dest	SUB
	src2	$\operatorname{src}1$	dest	AND
	src2	$\operatorname{src}1$	dest	XOR
	src2	$\operatorname{src}1$	dest	OR
	src2	$\operatorname{src}1$	dest	LSL
	src2	$\operatorname{src}1$	dest	LSR
	src2	$\operatorname{src}1$	dest	ASR

Instruction	Opcode	Formula	Description
ADD rd, rs1, rs2	00000001	$rd \leftarrow rs1 + rs2$	Add rs1 value to rs2 value, and store to rd.
SUB rd, rs1, rs2	00001001	$rd \leftarrow rs1 - rs2$	Subtract rs2 value from rs1 value, and store to rd.
AND rd, rs1, rs2	00010001	$rd \leftarrow rs1 \wedge rs2$	Do bit and between rs1 value and rs2 value, and store to rd.
XOR rd, rs1, rs2	00011001	$rd \leftarrow rs1 \oplus rs2$	Do bit xor between rs1 value and rs2 value, and store to rd.
OR rd, rs1, rs2	00100001	$rd \leftarrow rs1 \vee rs2$	Do bit or between rs1 value and rs2 value, and store to rd.
LSL rd, rs1, rs2	00101001	$rd \leftarrow rs1 \ll rs2$	Do left logical shift rs1 value with width rs2 value, and store to rd.
LSR rd, rs1, rs2	00110001	$rd \leftarrow rs1 \gg rs2$	Do right logical shift rs1 value with width rs2 value, and store to rd.
ASR rd, rs1, rs2	00111001	$rd \leftarrow rs1 \ggg rs2$	Do right arithmetic shift rs1 value with width rs2 value, and store to rd.

${\bf Integer\ Register\text{-}Immediate\ Instructions}$

23	16	15 12	11	8 7	6 5	0
simm10[7:0	0]	rs1	rd	simm10[9:8]	opcode	
8		4	4	2	6	
simm10[7]	[0:	src	dest	simm10[9:8]	ADDI	
simm10[7]	[0:	src	dest	simm10[9:8]	ANDI	
simm10[7]	[0:	src	dest	simm10[9:8]	XORI	
simm10[7]	[0:	src	dest	simm10[9:8]	ORI	
simm10[7]	:0]	0000	dest	simm10[9:8]	LI	

Instruction	Opcode	Formula	Description
ADDI rd, rs1, simm10	000011	$rd \leftarrow rs1 + simm10$	Add rs1 value to simm10, and store to rd.
ANDI rd, rs1, simm10	010011	$rd \leftarrow rs1 \wedge simm10$	Do bit and between rs1 value and simm10, and store to rd.
XORI rd, rs1, simm10	011011	$rd \leftarrow rs1 \oplus simm10$	Do bit xor between rs1 value and simm10, and store to rd.
ORI rd, rs1, simm10	100011	$rd \leftarrow rs1 \vee simm10$	Do bit or between rs1 value and simm10, and store to rd.
LI rd, simm10	110101	$rd \leftarrow simm10$	Load simm10, and store to rd.

23	20 19	16 15 12	2 11	8 7 0
0000	uimm4[3:0]	rs1	rd	opcode
4	4	4	4	8
	uimm4[3:0]	src	dest	LSLI
	uimm4[3:0]	src	dest	LSRI
	uimm4[3:0]	src	dest	ASRI

Instruction	Opcode	Formula	Description
LSLI rd, rs1, uimm4	00101011	$rd \leftarrow rs1 \ll uimm4$	Do left logical shift rs1 value with width uimm4, and store to rd.
LSRI rd, rs1, uimm4	00110011	$rd \leftarrow rs1 \gg uimm4$	Do right logical shift rs1 value with width uimm4, and store to rd.
ASRI rd, rs1, uimm4	00111011	$rd \leftarrow rs1 \ggg uimm4$	Do right arithmetic shift rs1 value with width uimm4, and store to rd.

2.2 Conditional Branches

23	16 1	5 1	2 11	8 7	6 5	0
simm10	[7:0]	rs1	rs2	simm10[9:8]	opcode	
8		4	4	2	6	
$\operatorname{simm}1$	0[7:0]	$\operatorname{src}1$	src2	simm10[9:8]	BEQ	
$\operatorname{simm}1$	0[7:0]	$\operatorname{src}1$	src2	simm10[9:8]	BNE	
$\operatorname{simm}1$	0[7:0]	$\operatorname{src}1$	src2	simm10[9:8]	BLT	
$\operatorname{simm}1$	0[7:0]	$\operatorname{src}1$	src2	simm10[9:8]	BLTU	
$\operatorname{simm}1$	0[7:0]	$\operatorname{src}1$	src2	simm10[9:8]	BLE	
$\operatorname{simm}1$	0[7:0]	$\operatorname{src}1$	src2	simm10[9:8]	BLEU	

Instruction	Opcode	Description
BEQ rs1, rs2, simm10	001111	If rs1 value is equal to rs2 value, jump to pc + simm10.
BNE rs1, rs2, simm10	101111	If rs1 value is not equal to rs2 value, jump to pc + simm10.
BLT rs1, rs2, simm10	110111	If rs1 signed value is less than rs2 one, jump to pc + simm10.
BLTU rs1, rs2, simm10	010111	If rs1 unsigned value is less than rs2 one, jump to pc + simm10.
BLE rs1, rs2, simm10	111111	If rs1 signed value is equal or less than rs2 one, jump to pc + simm10.
BLEU rs1, rs2, simm10	011111	If rs1 unsigned value is equal or less than rs2 one, jump to pc +
DDDC 131, 182, SIIIIII10	011111	simm10.

2.3 Load and Store Instructions

23	6 15 1:	2 11 8	3 7	6 5	0
simm10[7:0]	rs	rd	simm10[9:8]	opcode	
8	4	4	2	6	
simm10[7:0]	src	dest	simm10[9:8]	LW	
simm10[7:0]	src	dest	simm10[9:8]	LB	
simm10[7:0]	src	dest	simm10[9:8]	$_{ m LBU}$	

Instruction	Opcode	Formula	Description	
LW rd, simm10(rs)	010101	$rd \leftarrow [rs + simm10]$	Load word value with address rs value + simm10, and store to rd.	
LB rd, simm10(rs)	100101	$rd \leftarrow [rs + simm10]$	Load byte value with address rs value + simm10, and store sign extended one to rd.	
LBU rd, simm10(rs)	000101	$rd \leftarrow [rs + simm10]$	Load byte value with address rs value + simm10 and store to rd without sign extension.	

23	16 15	12 11	8	7	6 5	0
simm10[7:0]	rd		rs	simm10[9:8]	opcode	
8	4		4	2	6	
simm10[7:0]	dest	5	src	simm10[9:8]	SW	
simm10[7:0]	dest	5	src	simm10[9:8]	$_{ m SB}$	

Instruction Opcode Formula		Formula	Description	
SW rs, simm10(rd)	001101	$[rd + simm10] \leftarrow rs$	Load value from rs, and store word value to address	
2 11 15, SIIIIII (14)	001101		rs value + simm10.	
SB rs, simm10(rd)	110101	$\boxed{ [rd + simm10] \leftarrow rs }$	Load value from rs, and store value[7:0] to address	
SD 15, SIIIIIIIII(IU)	110101		rs value $+ simm10$.	

3 16-bit Length Instructions

3.1 Integer Computational Instructions

Integer Register-Register Instructions

15	12 11	8 7	0
rs	rd	opcode	
4	4	8	
src	dest	MOV	
src	dest	ADD2	
src	dest	SUB2	
src	dest	AND2	
src	dest	XOR2	
src	dest	OR2	
src	dest	LSL2	
src	dest	LSR2	
src	dest	ASR2	

Instruction	Opcode	Formula	Description
MOV rd, rs	11000000	$rd \leftarrow rs$	Load rs value, and store to rd.
ADD2 rd, rs	10000000	$rd \leftarrow rd + rs$	Add rs1 value to rd value, and store to rd.
SUB2 rd, rs	10001000	$rd \leftarrow rd - rs$	Subtract rd value from rs value, and store to rd.
AND2 rd, rs	10010000	$rd \leftarrow rd \wedge rs$	Do bit and between rd value and rs value, and store to rd.
XOR2 rd, rs	10011000	$rd \leftarrow rd \oplus rs$	Do bit xor between rd value and rs value, and store to rd.
OR2 rd, rs	10100000	$rd \leftarrow rd \vee rs$	Do bit or between rd value and rs value, and store to rd.
LSL2 rd, rs	10101000	$rd \leftarrow rd \ll rs$	Do left logical shift rd value with width rs value, and store
LSLZ IU, IS	10101000	1 1 d ← 1 d ≪ 1 s	to rd.
LSR2 rd, rs	10110000	$rd \leftarrow rd \gg rs$	Do right logical shift rd value with width rs value, and store
L5102 10, 15	10110000	1d — 1d // 1s	to rd.
ASR2 rd, rs	10111000	$rd \leftarrow rd \gg rs$	Do right arithmetic shift rd value with width rs value, and
A5112 10, 18	101111000	ru ← ru ≫ rs	store to rd.

${\bf Integer\ Register\text{-}Immediate\ Instructions}$

15	12 11	8 7	6 5		0
simm6[3:0]	rd	simme	3[5:4]	opcode	
4	4	2		6	
simm6[3:0]	dest	$\operatorname{simm} 6$	6[5:4]	ADDI2	
simm6[3:0]	dest	$\operatorname{simm} 6$	3[5:4]	ANDI2	
simm6[3:0]	dest	$\operatorname{simm} 6$	3[5:4]	LSI	
simm6[3:0]	dest	$\operatorname{simm} 6$	6[5:4]	LUI	

Instruction	Opcode	Formula	Description		
ADDI2 rd, simm6	000010	$rd \leftarrow rd + simm6$	Add rd value to simm6, and store to rd.		
ANDI2 rd, simm6	010010	$rd \leftarrow rd \wedge simm6$	Do bit and between rd value and simm6, and store		
			to rd.		
LSI rd, simm6	110100	$rd \leftarrow simm6$	Load simm6, and store to rd.		
LUI rd, simm6	000100	rd ((gimm6 // 10)	Load simm6 with left logical shift with width 10,		
LOT Id, Sillillo	000100	$rd \leftarrow (simm6 \ll 10)$	and store to rd.		

15	12 11	8	3 7	0
uimm4[3:0]	rd	opcode	
4	·	4	8	
uimm4	[3:0]	dest	LSLI2	
uimm4	[3:0]	dest	LSRI2	
uimm4	[3:0]	dest	ASRI2	

Instruction	Opcode	Formula	Description		
LSLI2 rd, uimm4	00101010	$rd \leftarrow rd \ll uimm4$	Do left logical shift rd value with width uimm4, and store to rd.		
LSRI2 rd, uimm4	00110010	$rd \leftarrow rd \gg uimm4$	Do right logical shift rd value with width uimm4, and store to rd.		
ASRI2 rd, uimm4	00111010	$rd \leftarrow rd \ggg uimm4$	Do right arithmetic shift rd value with width uimm4, and store to rd.		

3.2 Unconditional Jumps

15	12 11	3 7 0
0000	rs	opcode
4	4	8
	src	JALR
	src	$_{ m JR}$

Instruction	Opcode	Formula	Description	
JALR rs	00010110	$ \begin{array}{c} \text{ra} \leftarrow \text{pc} + 2 \\ \text{pc} \leftarrow \text{rs} \end{array} $	Jump to rs value, and store $pc + 2$ to return address register ra.	
JR rs	00000110	$pc \leftarrow rs$	Jump to rs value.	

15	5 4	0
simm11[10:0]	opcode	
11	5	
simm11[10:0]	JS	
simm11[10:0]	JSAL	

Instruction	Opcode Formula		Description
JS simm11	01110	$pc \leftarrow pc + simm11$	Jump to pc + simm11.
JSAL simm11	11110	$ra \leftarrow pc + 2$ $pc \leftarrow pc + simm11$	Jump to $pc + simm11$, and store $pc + 2$ to return address register ra.

3.3 Load and Store Instructions

15	$12 \ 11$	8	3 7	6 5	0
uimm7[4:1]		rd	uimm7[6:5]	(pcode
11		5			
uimm7[4:1]		dest	uimm7[6:5]]	LWSP

Instruction	Opcode	Formula	Description
LWSP rd, uimm7(sp)	010100	$rd \leftarrow [sp + uimm7]$	Load word value with address stack pointer register value + uimm7, and store to rd.

15	12 11	8	7	6 5	0
uimm7[4:1]		rs	uimm7[6:5]	opcode	
11		5			
uimm7[4:1]		src	$\operatorname{uimm7}[6:5]$	SWSP	

Instruction	Opcode	Formula	Description
SWSP rs, uimm7(sp)	011100	$[\mathrm{sp} + \mathrm{uimm7}] \leftarrow \mathrm{rs}$	Store rs value to uimm8 with address stack pointer register value + uimm7.

3.4 Others

15	12 11	8	3 7	6 5		0
uimm7[4:1]		rd	uimm7[6:5]		opcode	
11		5				
uimm7[4:1]		dest	uimm7[6:5]		NOP	

Instruction	Opcode	Formula	Description
NOP	000000	_	No operation.

4 Instructions List

08 07 06 05 04 03 02 01 00	0 0 0 0 0		0	0 0 1 1 0	0 1 0 0 0	0 1 0 1 0	0 1 1 0 0	0 0 1 1 1 0 0 1		simm10[9:8] 0 0 0 1 1	simm10[9:8] 0 1 0 0 1 1	simm10[9:8] 0 1 1 0 1 1	simm10[9:8] 1 0 0 0 1 1			0 0 $ 1 1 1 0 1 1$		simm10[9:8] 0 0 1 1 1 1					simm10[9:8] simm10[9:8] simm10[9:8] simm10[9:8] simm10[9:8]	imm10[9:8] imm10[9:8] imm10[9:8] imm10[9:8] imm10[9:8] imm10[9:8]	imm10[9:8] imm10[9:8] imm10[9:8] imm10[9:8] imm10[9:8] imm10[9:8]	imm10[9:8] imm10[9:8] imm10[9:8] imm10[9:8] imm10[9:8] imm10[9:8]	imm10[9:8] 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	imm10[9:8] 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	imm10[9:8] 0 0 1 1 1 imm10[9:8] 1 0 0 1 1 1 imm10[9:8] 1 1 1 0 1 imm10[9:8] 0 1 0 1 1 1 1 imm10[9:8] 0 1 0 0 1 imm10[9:8] 0 1 0 0 1 imm10[9:8] 0 0 0 0 1 imm10[9:8] 0 0 0 0 0 0 0 1 imm10[9:8] 0 0 0 0 0 0 0 0 0 1 imm10[9:8] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	imm10[9:8] 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	imm10[9:8] 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	simm10[9:8] 0 0 1 1 1 1 1 1 1 1 1 1
11 10 09 08 07	rd	rd	rd	rd	rd	rd	rd	rd		rd	rd	rd	rd	rd	rd	rd		rs2	rs2 rs2	rs2 rs2 rs2	1.82 1.82 1.82 1.82	1S2 1S2 1S2 1S2 1S2 1S2	182 182 182 182 182	182 182 182 182 182	182 182 182 182 182	182 182 182 182 182 182 17d	182 182 182 182 182 182 17	182 182 182 182 182 182 174 174	182 182 182 182 182 182 174 174 175 176 177 178 178 178 178 178 178 178 178 178	182 182 182 182 182 182 183 184 185	182 182 182 182 182 184 184	182 182
151413 12 uctions	rs1	rs1	rs1	rs1	rs1	rs1	rs1	rs1	$\operatorname{Instructions}$	rs1	rs1	rs1	rs1		rs1	rs1		rs1	rs1 rs1	rs1 rs1	rs1 rs1 rs1	rs1 rs1 rs1 rs1	rates rs1 rs1 rs1 rs1 rs1 rs1 rs1 rs1 rs1 rs1 rs1 rs1 rs1 rs1	5[15:(3[15:(5[15:0	5[15:0	5[15:0	5[15:0	rsl rsl rsl rsl rsl [15:0] [15:0] [15:0] rs rs rs	rs1	rs1
	0 0 0 0 rs2	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0	0 0 0 0 rs2	Register-Immediate Inst	[0.7]0[7:0]	[0.7] simm $10[7:0]$	[simm10[7:0]]	simr	0 0 0	0 0 0	0 0 0 0 uimm4[3:0]	24-bit Conditional Branches	[7:0]	$\begin{array}{c} \mathrm{simm10[7:0]} \\ \mathrm{simm10[7:0]} \end{array}$	simm10[7:0] simm10[7:0] simm10[7:0]	simm10[7:0] simm10[7:0] simm10[7:0] simm10[7:0]	simm10[7:0] simm10[7:0] simm10[7:0] simm10[7:0] simm10[7:0]	simm10[7:0] simm10[7:0] simm10[7:0] simm10[7:0] simm10[7:0] simm10[7:0]	simm10[7:0 simm10[7:0 simm10[7:0 simm10[7:0 simm10[7:0	simm10[7:0 simm10[7:0 simm10[7:0 simm10[7:0 simm10[7:0 and Store Instr	simm10[7:0 simm10[7:0 simm10[7:0 simm10[7:0 simm10[7:0 simm10[7:0	simm10[7:0 simm10[7:0 simm10[7:0 simm10[7:0 simm10[7:0 simm10[7:0 simm10[7:0 simm10[7:0 simm10[7:0	simm10[7:0 simm10[7:0 simm10[7:0 simm10[7:0 simm10[7:0 simm10[7:0 simm10[7:0 simm10[7:0 simm10[7:0	simm10[7:0	simm10[7:0	simm10[7:0] simm10[7:0] simm10[7:0] simm10[7:0] simm10[7:0] simm simm10[7:0]	simm10[7:0
Formula 24-bit Integer I	$\leftarrow \text{rs1} + \text{rs2}$	$\leftarrow \mathrm{rs}1$	\leftarrow rs1	$\mathrm{rd} \leftarrow \mathrm{rs1} \oplus \mathrm{rs2}$	\leftarrow rs1	$\leftarrow \text{rs1}$	$\leftarrow \mathrm{rs1}$	$\mathrm{rd}\leftarrow\mathrm{rs}1\gg\mathrm{rs}2$	24-bit Integer R	$\leftarrow \text{rs1} +$	$\leftarrow \mathrm{rs1}$	$rd \leftarrow rs1 \oplus simm10$	\downarrow	$\leftarrow \text{rs1}$	$\leftarrow rs1$	$rd \leftarrow rs1 \gg uimm4$	24-bit	= rs2	$= rs2 then pc \leftarrow \\ \neq rs2 then pc \leftarrow$	= rs2 then pc \leftarrow pc + \neq rs2 then pc \leftarrow pc + < rs2 then pc \leftarrow pc +	= rs2 then pc \leftarrow pc + \neq rs2 then pc \leftarrow pc + < rs2 then pc \leftarrow pc + < rs2 then pc \leftarrow pc +	= rs2 $\neq rs2$ < rs2 < rs2 < rs2 < rs2 < rs2 < rs2	= rs2 then pc \leftarrow pc + \neq rs2 then pc \leftarrow pc + < rs2 then pc \leftarrow pc +	= rs2 then pc \leftarrow pc \rightarrow rs2 then pc \leftarrow pc $<$ pc $<$ pc \leftarrow pc	= rs2 then pc \leftarrow pc + \neq rs2 then pc \leftarrow pc + < rs2 then pc \leftarrow pc + pc \leftarrow pc + simm16 pc \leftarrow pc + simm16	= rs2 then pc \leftarrow pc- \neq rs2 then pc \leftarrow pc- < rs2 then pc \leftarrow pc- pc \leftarrow pc + simm1 - pc + 4, pc \leftarrow pc + simm1	= rs2 then pc \leftarrow pc \rightarrow pc \leftarrow pc \rightarrow simm10 rd \leftarrow [rs + simm10]	= rs2 then pc \leftarrow pc- < rs2 then pc \leftarrow pc- pc \leftarrow pc + simml - pc + 4, pc \leftarrow pc + simml rd \leftarrow [rs + simml(rd \leftarrow]]]])	= rs2 then pc \leftarrow pc- \neq rs2 then pc \leftarrow pc- $<$ rs2 then pc \leftarrow pc- $<$ rs2 then pc \leftarrow pc- \leq rs2 then pc \leftarrow pc- \leq rs2 then pc \leftarrow pc- \leq rs2 then pc \leftarrow pc- \Rightarrow pc \leftarrow pc + simml \Rightarrow pc \leftarrow pc + simml \Rightarrow rd \leftarrow [rs + simml rd \leftarrow [rs + simml rd \leftarrow [rs + simml] rd \leftarrow [rs + simml] rd \leftarrow [rs + simml] rd \leftarrow [rs + simml]	= rs2 then pc \leftarrow pc- \neq rs2 then pc \leftarrow pc- < rs2 then pc \leftarrow pc- S rs2 then pc \leftarrow pc- pc \leftarrow pc + simml - pc \leftarrow pc + simml rd \leftarrow [rs + simml0] rd \leftarrow [rs + simml0] rd \leftarrow [rs + simml0] rd \leftarrow [rd + simml0] \leftarrow r	= rs2 then pc \leftarrow < pc \leftarrow pc + sin pc \leftarrow pc + sin rd \leftarrow [rs + sin rd \leftarrow [rd + sinmulo] rd \leftarrow sinmulo]	= rs2 then pc \leftarrow \neq rs2 then pc \leftarrow < rs2 then pc \leftarrow pc \leftarrow pc + sin pc \leftarrow pc \leftarrow pc + sin rd \leftarrow [rs + sin rd \leftarrow [rs + sin rd \leftarrow [rs + sin rd \leftarrow [rd + sinm10] [rd \leftarrow sinm rd \leftarrow rs + sin rd \leftarrow rs + sin rd \leftarrow rd \leftarrow rs + sin rd \leftarrow rs rd \leftarrow rd \leftarrow rs
Ops	add rd, rs1, rs2	sub rd, rs1, rs2	and rd, rs1, rs2	xor rd, rs1, rs2	or rd, rs1, rs2	lsl rd, rs1, rs2	lsr rd, rs1, rs2	asr rd, rs1, rs2		addi rd, rs1, simm10	andi rd, rs1, simm10	xori rd, rs1, simm10	ori rd, rs1, simm10	lsli rd, rs1, uimm4	lsri rd, rs1, uimm4	asri rd, rs1, uimm4		 eq rs1, rs2, simm10	eq rsl, rs2, simm10 nee rs1, rs2, simm10	eq rsl, rs2, simm10 ne rsl, rs2, simm10 olt rsl, rs2, simm10	eq rsl, rs2, simml0 one rsl, rs2, simml0 olt rsl, rs2, simml0 ltu rsl, rs2, simml0	eq rs1, rs2, simm10 ne rs1, rs2, simm10 lt rs1, rs2, simm10 ltu rs1, rs2, simm10 le rs1, rs2, simm10 le rs1, rs2, simm10	eq rs1, rs2, simm10 ne rs1, rs2, simm10 lt rs1, rs2, simm10 ltu rs1, rs2, simm10 le rs1, rs2, simm10 le rs1, rs2, simm10 j simm16	eq rs1, rs2, simm10 ne rs1, rs2, simm10 lt rs1, rs2, simm10 ltu rs1, rs2, simm10 le rs1, rs2, simm10 leu rs1, rs2, simm10 j simm16 j simm16 jal simm16	eq rsl, rs2, simml0 ne rsl, rs2, simml0 lt rsl, rs2, simml0 ltu rsl, rs2, simml0 le rs1, rs2, simml0 le rs1, rs2, simml0 jsimml6 jsimml6 jal simml6	oeq rsl, rs2, simml0 one rsl, rs2, simml0 olt rsl, rs2, simml0 oltu rsl, rs2, simml0 ole rsl, rs2, rs2, rs2, rs2, rs2, rs2, rs2, rs2	beq rsl, rs2, simml0 bne rsl, rs2, simml0 blt rsl, rs2, simml0 bltu rsl, rs2, simml0 ble rsl, rs2, simml0 bleu rsl, rs2, simml0 jsimml6 jal simml6 jal simml6 lw rd, simml0(rs) lb rd, simml0(rs)	oeq rsl, rs2, simml0 one rsl, rs2, simml0 olt rsl, rs2, simml0 oltu rsl, rs2, simml0 ole rsl, rs2, simml0 ole rsl, rs2, simml0 j simml6 j simml6 jal simml6 jal simml0 lw rd, simml0(rs) lb rd, simml0(rs)	one rs1, rs2, simm10 lt rs1, rs2, simm10 ltu rs1, rs2, simm10 ltu rs1, rs2, simm10 le rs1, rs2, simm10 jsimm16 jsimm16 jal simm16 lw rd, simm10(rs) lb rd, simm10(rs) lb rd, simm10(rs) sw rs, simm10(rd)	oeq rsl, rs2, simml0 one rsl, rs2, simml0 olt rsl, rs2, simml0 olt rsl, rs2, simml0 ole rsl, rsmnl6 ole rsl, simml0(rs) ole rsl, simml0(rs) ole rsl, simml0(rs) ole rsl, simml0(rs) ole rsl, simml0(rd)	led rs1, rs2, simm10 lt rs1, rs2, simm10 ltu rs1, rs2, simm10 ltu rs1, rs2, simm10 le rs1, rs2, simm10 leu rs1, rs2, simm10 j simm16 jal simm16 jal simm16 jal simm10(rs) lb rd, simm10(rs) lb rd, simm10(rs) sw rs, simm10(rd) sb rs, simm10(rd) li rd, simm10(rd)	led rsl, rs2, simmlo le rsl, rs2, simmlo lt rsl, rs2, simmlo ltu rsl, rs2, simmlo le rsl, rs2, simmlo leu rsl, rs2, simmlo j simml6 jal simml6 jal simml6 lw rd, simml0(rs) lb rd, simml0(rs) lb rd, simml0(rd) sw rs, simml0(rd) sb rs, simml0(rd) li rd, simml0 li rd, simml0 li rd, simml0

0 1 0 0 0 0	0 1 1 0 0 0	1 0 0 0 0 0	1 0 1 0 0 0	1 1 0 0 0 0	1 1 1 0 0 0		0 0 0 0 1 0	0 1 0 0 1 0	1 1 0 1 0 0	0 0 0 1 0 0	1 0 1 0 1 0	1 1 0 0 1 0	1 1 1 0 1 0		0 1 0 1 1 0	0 0 0 1 1 0	0 1 1 1 0	1 1 1 1 0		0 1 0 1 0 0	0 1 1 1 0 0		0 0 0 0 0 0	
1 0	1 0	1 0	1 0	1 0	1 0		simm6[5:4]	simm6[5:4]	simm6[5:4]	simm6[5:4]	0 0	0 0	0 0		0 0	0 0	.0:0]	[0:0]		[0.000]	uimm7[6:5]		0 0	
$\mid \operatorname{rd}(\operatorname{rs1})$	$ \operatorname{rd}(\operatorname{rs}1) $	$ \operatorname{rd}(\operatorname{rs1}) $	$ \operatorname{rd}(\operatorname{rs1}) $	$ \operatorname{rd}(\operatorname{rs1}) $	$ \operatorname{rd}(\operatorname{rs1}) $		rd(rs1)	$ \operatorname{rd}(\operatorname{rs}1) $	rd	rd	$ \operatorname{rd}(\operatorname{rs1}) $	rd(rs1)	$ \operatorname{rd}(\operatorname{rs1}) $		rs(rs1)	rs(rs1)	simm11[10:0	simm11[10:0]		rd	rs		0 0 0 0 0	
rs(rs2)	rs(rs2)	m rs(rs2)	m rs(rs2)	m rs(rs2)	m rs(rs2)	uctions	simm6[3:0]	simm6[3:0]	simm6[3:0]	[simm6[3:0]]	uimm4[3:0]	$ \operatorname{uimm4}[3:0]$	$\begin{array}{ c c c c c c c c c c c c c c c c c c c$		0 0 0 0	0 0 0 0			SI	$\lim_{n \to \infty} 7[4:1]$	$\mid \text{uimm7}[4:1]$		0 0 0 0	
	1					16-bit Integer Register-Immediate Instructions								16-bit Unconditional Jumps				1	16-bit Load and Store Instructions			Others		
$rd \leftarrow rd \wedge rs$	$rd \leftarrow rd \oplus rs$	$rd \leftarrow rd \vee rs$	$rd \leftarrow rd \ll rs$	$rd \leftarrow rd \gg rs$	$rd \leftarrow rd \ggg rs$	16-bit Integer Re	$rd \leftarrow rd + simm6$	$\mathrm{rd} \leftarrow \mathrm{rd} \wedge \mathrm{simm6}$	$rd \leftarrow simm6$	$rd \leftarrow (simm6 \ll 10)$	$rd \leftarrow rd \ll uimm4$	$rd \leftarrow rd \gg uimm4$	$rd \leftarrow rd \gg uimm4$	16-bit 1	$ra \leftarrow pc + 2, pc \leftarrow rs$	$\mathrm{pc} \leftarrow \mathrm{rs}$	$pc \leftarrow pc + simm11$	$ra \leftarrow pc + 2, pc \leftarrow pc + simm11$	16-bit Loa	$rd \leftarrow [sp + uimm7]$	$[sp + uimm7] \leftarrow rs$			
and2 rd, rs	xor2 rd, rs	or2 rd, rs	lsl2 rd, rs	lsr2 rd, rs	asr2 rd, rs		addi2 rd, simm6	andi2 rd, simm6	lsi rd, simm6	lui rd, simm6	lsli2 rd, uimm4	lsri2 rd, uimm4	asri2 rd, uimm4		jalr rs	jr rs	js simm11	jsal simm11		lwsp rd, uimm7(sp)	swsp rs, $\operatorname{uimm7(sp)}$		dou	