

CAHPv3 Instruction Set Specification

Naoki Matsumoto, Ryotaro Banno, Kotaro Matsuoka

2020/09/21

1 Programmers’ Model for CAHPv3 ISA

1.1 Integer Registers

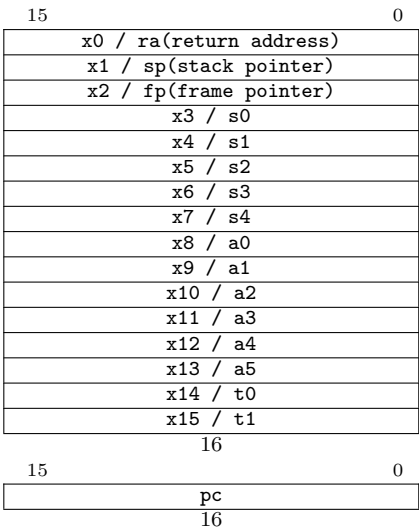
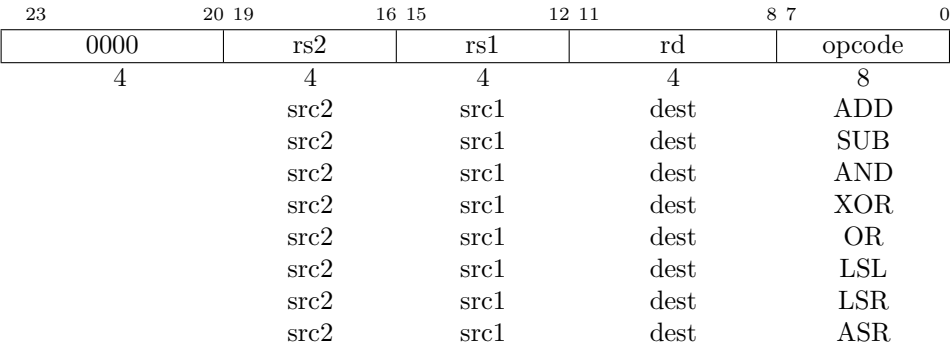


Figure 1: CAHPv3 integer register state.

2 24-bit Length Instructions

2.1 Integer Computational Instructions

Integer Register-Register Instructions



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.

Integer Register-Immediate Instructions

23	16 15	12 11	8 7	6 5	0
simm10[7:0]	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	$rd \leftarrow rs1 + \text{simm10}$	Add rs1 value to simm10, and store to rd.
ANDI rd, rs1, simm10	010011	$rd \leftarrow rs1 \wedge \text{simm10}$	Do bit and between rs1 value and simm10, and store to rd.
XORI rd, rs1, simm10	011011	$rd \leftarrow rs1 \oplus \text{simm10}$	Do bit xor between rs1 value and simm10, and store to rd.
ORI rd, rs1, simm10	100011	$rd \leftarrow rs1 \vee \text{simm10}$	Do bit or between rs1 value and simm10, and store to rd.
LI rd, simm10	110101	$rd \leftarrow \text{simm10}$	Load simm10, and store to rd.

23	20 19	16 15	12 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 15	12 11	8 7	6 5	0
simm10[7:0]	rs1	rs2	simm10[9:8]	opcode	
8	4	4	2	6	
simm10[7:0]	src1	src2	simm10[9:8]	BEQ	
simm10[7:0]	src1	src2	simm10[9:8]	BNE	
simm10[7:0]	src1	src2	simm10[9:8]	BLT	
simm10[7:0]	src1	src2	simm10[9:8]	BLTU	
simm10[7:0]	src1	src2	simm10[9:8]	BLE	
simm10[7:0]	src1	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 + simm10.

2.3 Load and Store Instructions

23	16 15	12 11	8 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]	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	src	simm10[9:8]	SW	
simm10[7:0]	dest	src	simm10[9:8]	SB	

Instruction	Opcode	Formula	Description
SW rs, simm10(rd)	001101	$[rd + \text{simm10}] \leftarrow rs$	Load value from rs, and store word value to address rs value + simm10.
SB rs, simm10(rd)	110101	$[rd + \text{simm10}] \leftarrow rs$	Load value from rs, and store value[7:0] to address 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 to rd.
LSR2 rd, rs	10110000	$rd \leftarrow rd \gg rs$	Do right logical shift rd value with width rs value, and store to rd.
ASR2 rd, rs	10111000	$rd \leftarrow rd \ggg rs$	Do right arithmetic shift rd value with width rs value, and store to rd.

Integer Register-Immediate Instructions

15	12 11	8 7	6 5	0
simm6[3:0]	rd	simm6[5:4]	opcode	
4	4	2	6	
simm6[3:0]	dest	simm6[5:4]	ADDI2	
simm6[3:0]	dest	simm6[5:4]	ANDI2	
simm6[3:0]	dest	simm6[5:4]	LSI	
simm6[3:0]	dest	simm6[5:4]	LUI	

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

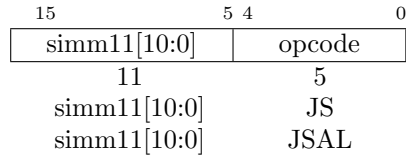
15	12 11	8 7	0
uimm4[3:0]	rd	opcode	
4	4	8	
uimm4[3:0]	dest	LSLI2	
uimm4[3:0]	dest	LSRI2	
uimm4[3:0]	dest	ASRI2	

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

3.2 Unconditional Jumps

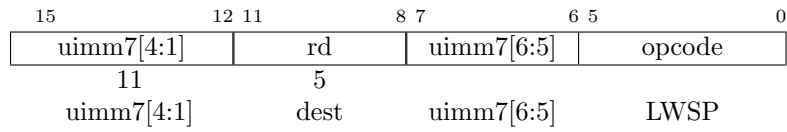
15	12 11	8 7	0
0000	rs	opcode	
4	4	8	
	src	JALR	
	src	JR	

Instruction	Opcode	Formula	Description
JALR rs	00010110	$ra \leftarrow pc + 2$ $pc \leftarrow rs$	Jump to rs value, and store pc + 2 to return address register ra.
JR rs	00000110	$pc \leftarrow rs$	Jump to rs value.

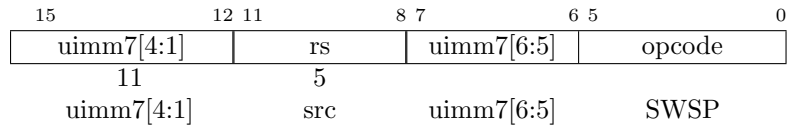


Instruction	Opcode	Formula	Description
JS simm11	01110	$pc \leftarrow pc + \text{simm11}$	Jump to $pc + \text{simm11}$.
JSAL simm11	11110	$ra \leftarrow pc + 2$ $pc \leftarrow pc + \text{simm11}$	Jump to $pc + \text{simm11}$, and store $pc + 2$ to return address register ra.

3.3 Load and Store Instructions

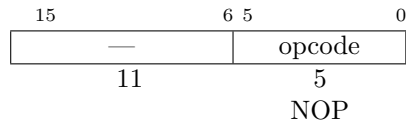


Instruction	Opcode	Formula	Description
LWSP rd, uimm7(sp)	010100	$rd \leftarrow [sp + \text{uimm7}]$	Load word value with address stack pointer register value + uimm7, and store to rd.



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

3.4 Others



Instruction	Opcode	Formula	Description
NOP	000000	—	No operation.

4 Instructions List

Ops	Formula	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	08	07	06	05	04	03	02	01	00
24-bit Integer Register-Register Instructions																									
ADD rd, rs1, rs2	rd ← rs1 + rs2	0	0	0	0	0	0	rs2	rs2	rs1	rs1	rd	rd	0	0	0	0	0	0	0	0	0	0	0	1
SUB rd, rs1, rs2	rd ← rs1 − rs2	0	0	0	0	0	0	rs2	rs2	rs1	rs1	rd	rd	0	0	0	0	0	0	0	0	0	0	0	1
AND rd, rs1, rs2	rd ← rs1 ∧ rs2	0	0	0	0	0	0	rs2	rs2	rs1	rs1	rd	rd	0	0	0	0	0	0	0	0	0	0	0	1
XOR rd, rs1, rs2	rd ← rs1 ⊕ rs2	0	0	0	0	0	0	rs2	rs2	rs1	rs1	rd	rd	0	0	0	0	0	0	0	0	0	0	0	1
OR rd, rs1, rs2	rd ← rs1 ∨ rs2	0	0	0	0	0	0	rs2	rs2	rs1	rs1	rd	rd	0	0	0	0	0	0	0	0	0	0	0	1
LSL rd, rs1, rs2	rd ← rs1 ≪ rs2	0	0	0	0	0	0	rs2	rs2	rs1	rs1	rd	rd	0	0	0	0	0	0	0	0	0	0	0	1
LSR rd, rs1, rs2	rd ← rs1 ≫ rs2	0	0	0	0	0	0	rs2	rs2	rs1	rs1	rd	rd	0	0	0	0	0	0	0	0	0	0	0	1
ASR rd, rs1, rs2	rd ← rs1 ≫≧ rs2	0	0	0	0	0	0	rs2	rs2	rs1	rs1	rd	rd	0	0	0	0	0	0	0	0	0	0	0	1
24-bit Integer Register-Immediate Instructions																									
ADDI rd, rs1, simm10	rd ← rs1 + simm10	simm10[7:0]							rs1	rs1	rd	rd	simm10[9:8]		0	0	0	0	1	1	0	0	0	0	1
ANDI rd, rs1, simm10	rd ← rs1 ∧ simm10	simm10[7:0]							rs1	rs1	rd	rd	simm10[9:8]		0	1	0	0	1	1	0	0	0	0	1
XORI rd, rs1, simm10	rd ← rs1 ⊕ simm10	simm10[7:0]							rs1	rs1	rd	rd	simm10[9:8]		0	1	0	0	1	1	0	0	0	0	1
ORI rd, rs1, simm10	rd ← rs1 ∨ simm10	simm10[7:0]							rs1	rs1	rd	rd	simm10[9:8]		1	0	0	0	1	1	0	0	0	0	1
LSLI rd, rs1, uimm4	rd ← rs1 ≪ uimm4	0	0	0	0	uimm4[3:0]			rs1	rs1	rd	rd	0	0	0	0	0	0	0	0	0	0	0	1	
LSRI rd, rs1, uimm4	rd ← rs1 ≫ uimm4	0	0	0	0	uimm4[3:0]			rs1	rs1	rd	rd	0	0	0	0	0	0	0	0	0	0	0	1	
ASRI rd, rs1, uimm4	rd ← rs1 ≫≧ uimm4	0	0	0	0	uimm4[3:0]			rs1	rs1	rd	rd	0	0	0	0	0	0	0	0	0	0	0	1	
24-bit Conditional Branches																									
BEQ rs1, rs2, simm10	if rs1 = rs2 then pc ← pc + simm10	simm10[7:0]							rs1	rs1	rs2	rs2	simm10[9:8]		0	0	1	1	1	1	0	0	0	0	1
BNE rs1, rs2, simm10	if rs1 ≠ rs2 then pc ← pc + simm10	simm10[7:0]							rs1	rs1	rs2	rs2	simm10[9:8]		1	0	1	1	1	1	0	0	0	0	1
BLT rs1, rs2, simm10	if rs1 < rs2 then pc ← pc + simm10	simm10[7:0]							rs1	rs1	rs2	rs2	simm10[9:8]		1	1	0	1	1	1	0	0	0	0	1
BLTU rs1, rs2, simm10	if rs1 < rs2 then pc ← pc + simm10	simm10[7:0]							rs1	rs1	rs2	rs2	simm10[9:8]		0	1	0	1	1	1	0	0	0	0	1
BLE rs1, rs2, simm10	if rs1 ≤ rs2 then pc ← pc + simm10	simm10[7:0]							rs1	rs1	rs2	rs2	simm10[9:8]		1	1	1	1	1	1	0	0	0	0	1
BLEU rs1, rs2, simm10	if rs1 ≤ rs2 then pc ← pc + simm10	simm10[7:0]							rs1	rs1	rs2	rs2	simm10[9:8]		0	1	1	1	1	1	0	0	0	0	1
J simm16	pc ← pc + simm16	simm16[15:0]							rs1	rs1	rs2	rs2	simm10[9:8]		0	0	0	0	0	0	0	0	0	0	1
JAL simm16	ra ← pc + 4, pc ← pc + simm16	simm16[15:0]							rs1	rs1	rs2	rs2	simm10[9:8]		0	0	0	0	0	0	0	0	0	0	1
24-bit Load and Store Instructions																									
LW rd, simm10(rs)	rd ← [rs + simm10]	simm10[7:0]							rs	rs	rd	rd	simm10[9:8]		0	1	0	1	0	1	0	0	0	0	1
LB rd, simm10(rs)	rd ← [rs + simm10]	simm10[7:0]							rs	rs	rd	rd	simm10[9:8]		1	0	0	1	0	1	0	0	0	0	1
LBU rd, simm10(rs)	rd ← [rs + simm10]	simm10[7:0]							rs	rs	rd	rd	simm10[9:8]		0	0	0	1	0	1	0	0	0	0	1
SW rs, simm10(rd)	[rd + simm10] ← rs	simm10[7:0]							rd	rd	rs	rs	simm10[9:8]		0	1	1	0	1	0	1	0	0	0	1
SB rs, simm10(rd)	[rd + simm10] ← rs	simm10[7:0]							rd	rd	rs	rs	simm10[9:8]		0	0	1	0	1	0	1	0	0	0	1
LI rd, simm10	rd ← simm10	simm10[7:0]							0	0	0	0	simm10[9:8]		1	1	0	1	0	1	0	0	0	0	1
16-bit Integer Register-Register Instructions																									
MOV rd, rs	rd ← rs	rs(rs2)							rs(rs2)	rs(rs2)	rd(rs1)	rd(rs1)	1		1	0	0	0	0	0	0	0	0	0	0
ADD2 rd, rs	rd ← rd + rs	rs(rs2)							rs(rs2)	rs(rs2)	rd(rs1)	rd(rs1)	1		0	0	0	0	0	0	0	0	0	0	0
SUB2 rd, rs	rd ← rd − rs	rs(rs2)							rs(rs2)	rs(rs2)	rd(rs1)	rd(rs1)	1		0	0	0	0	0	0	0	0	0	0	0

AND2 rd, rs	rd ← rd ∧ rs	—	rs(rs2)	rd(rs1)	1	0	0	1	0	0	0
XOR2 rd, rs	rd ← rd ⊕ rs	—	rs(rs2)	rd(rs1)	1	0	0	0	1	0	0
OR2 rd, rs	rd ← rd ∨ rs	—	rs(rs2)	rd(rs1)	1	0	0	0	0	0	0
LSL2 rd, rs	rd ← rd ≪ rs	—	rs(rs2)	rd(rs1)	1	0	0	0	0	0	0
LSR2 rd, rs	rd ← rd ≫ rs	—	rs(rs2)	rd(rs1)	1	0	0	0	0	0	0
ASR2 rd, rs	rd ← rd ≫≧ rs	—	rs(rs2)	rd(rs1)	1	0	0	0	0	0	0
16-bit Integer Register-Immediate Instructions											
ADDI2 rd, simm6	rd ← rd + simm6	—	simm6[3:0]	rd(rs1)	simm6[5:4]	0	0	0	0	1	0
ANDI2 rd, simm6	rd ← rd ∧ simm6	—	simm6[3:0]	rd(rs1)	simm6[5:4]	0	1	0	0	1	0
LSI rd, simm6	rd ← simm6	—	simm6[3:0]	rd	simm6[5:4]	1	1	0	1	0	0
LUI rd, simm6	rd ← (simm6 ≪ 10)	—	simm6[3:0]	rd	simm6[5:4]	0	0	0	1	0	0
LSLI2 rd, uimm4	rd ← rd ≪ uimm4	—	uimm4[3:0]	rd(rs1)	0	1	0	1	0	1	0
LSRI2 rd, uimm4	rd ← rd ≫ uimm4	—	uimm4[3:0]	rd(rs1)	0	1	1	0	0	1	0
ASRI2 rd, uimm4	rd ← rd ≫≧ uimm4	—	uimm4[3:0]	rd(rs1)	0	1	1	0	1	0	0
16-bit Unconditional Jumps											
JALR rs	ra ← pc + 2, pc ← rs	—	0 0 0	rs(rs1)	0	0	1	0	1	1	0
JR rs	pc ← rs	—	0 0 0	rs(rs1)	0	0	0	1	1	0	0
JS simm11	pc ← pc + simm11	—	simm11[10:0]			0	1	1	1	0	0
JSAL simm11	ra ← pc + 2, pc ← pc + simm11	—	simm11[10:0]			1	1	1	1	0	0
16-bit Load and Store Instructions											
LWSP rd, uimm7(sp)	rd ← [sp + uimm7]	—	uimm7[4:1]	rd	uimm7[6:5]	0	1	0	1	0	0
SWSP rs, uimm7(sp)	[sp + uimm7] ← rs	—	uimm7[4:1]	rs	uimm7[6:5]	0	1	1	1	0	0
Others											
NOP	—	—	0 0 0	0	0 0 0 0 0	0	0	0	0	0	0