CAHPv3 Instruction Set Specification

Naoki Matsumoto, Ryotaro Banno, Kotaro Matsuoka

2020/09/21

1 Programmers' Model for CAHPv3 ISA

1.1 Integer Registers

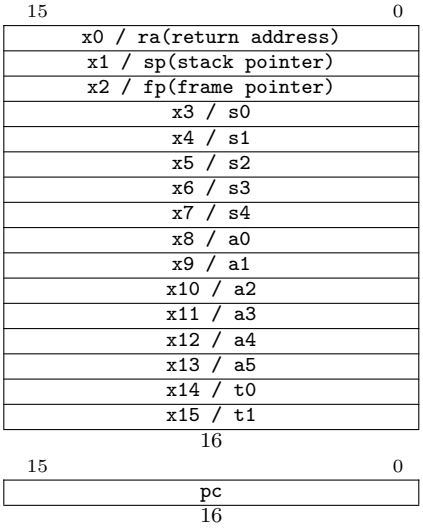
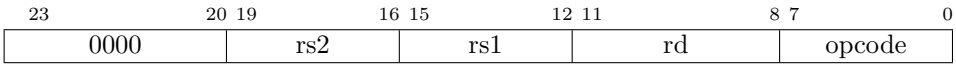


Figure 1: CAHPv3 integer register state.

2 24-bit Length Instructions

2.1 Integer Computational Instructions

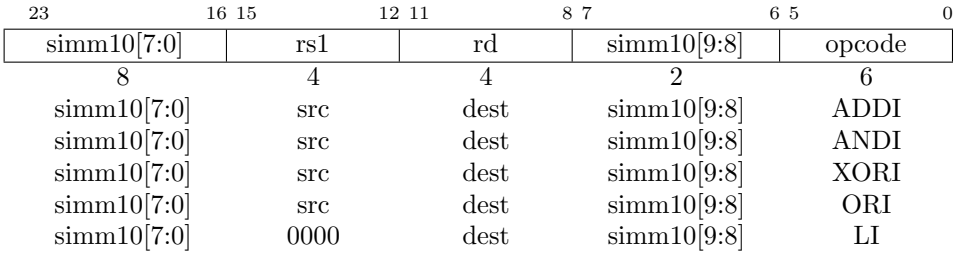
Integer Register-Register Instructions



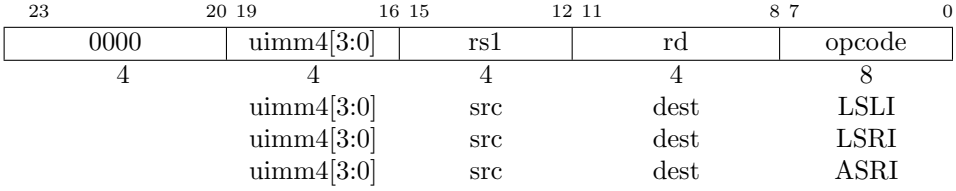
	src2	src1	dest	ADD
	src2	src1	dest	SUB
	src2	src1	dest	AND
	src2	src1	dest	XOR
	src2	src1	dest	OR
	src2	src1	dest	LSL
	src2	src1	dest	LSR
	src2	src1	dest	ASR

Instruction	Opcode	Formula	Description
ADD rd, rs1, rs2	00000001	rd <- rs1 + rs2	add rs1 value to rs2 value, and store to rd
SUB rd, rs1, rs2	00001001	rd <- rs1 - rs2	subtract rs2 value from rs1 value, and store to rd
AND rd, rs1, rs2	00010001	rd <- rs1 & rs2	do bit between rs1 value and rs2 value, and store to rd
XOR rd, rs1, rs2	00011001	rd <- rs1 ^ rs2	do bit xor between rs1 value and rs2 value, and store to rd
OR rd, rs1, rs2	00100001	rd <- rs1 | rs2	do bit or between rs1 value and rs2 value, and store to rd
LSL rd, rs1, rs2	00101001	rd <- rs1 << rs2	do left logical shift rs1 value with width rs2 value, and store to rd
LSR rd, rs1, rs2	00110001	rd <- rs1 >> rs2	do right logical shift rs1 value with width rs2 value, and store to rd
ASR rd, rs1, rs2	00111001	rd <- rs1 >>> rs2	do rigth arithmetic shift rs1 value with width rs2 value, and store to rd

Integer Register-Immediate Instructions



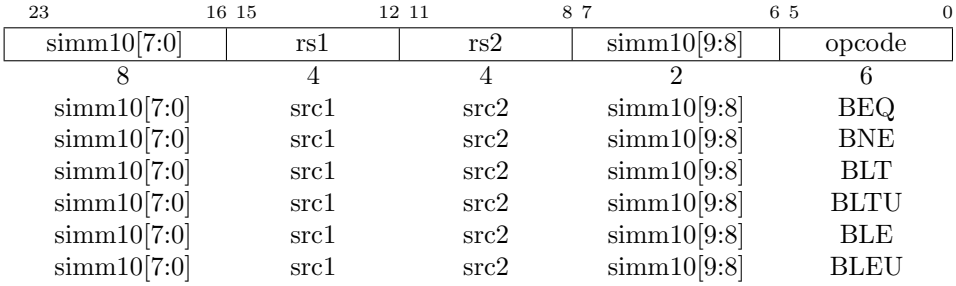
23 – 16	15 – 12	11 – 8	7 – 6	5 – 0
simm10[7:0]	rs1	rd	simm10[9:8]	opcode
8	4	4	2	6
simm10[7:0]	src	dest	simm10[9:8]	ADDI
simm10[7:0]	src	dest	simm10[9:8]	ANDI
simm10[7:0]	src	dest	simm10[9:8]	XORI
simm10[7:0]	src	dest	simm10[9:8]	ORI
simm10[7:0]	0000	dest	simm10[9:8]	LI



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
ADDI rd, rs1, simm10	000011	rd <- rs1 + simm10	add rs1 value to simm10, and store to rd
ANDI rd, rs1, simm10	010011	rd <- rs1 & simm10	do bit and between rs1 value and simm10, and store to rd
XORI rd, rs1, simm10	011011	rd <- rs1 ^ simm10	do bit xor between rs1 value and simm10, and store to rd
ORI rd, rs1, simm10	100011	rd <- rs1 | simm10	do bit or between rs1 value and simm10, and store to rd
LI rd, simm10	110101	rd <- rs1 << simm10	load simm10, and store to rd
LSLI rd, rs1, uimm4	00101011	rd <- rs1 << uimm4	do left logical shift rs1 value with width uimm4, and store to rd
LSRI rd, rs1, uimm4	00110011	rd <- rs1 >> uimm4	do rigth logical shift rs1 value with width uimm4, and store to rd
ASRI rd, rs1, uimm4	00111011	rd <- rs1 >>> 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

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
LW rd, simm10(rs)	010101	rd <- [rs + simm10]	load word value with address rs value + simm10, and store to rd
LB rd, simm10(rs)	100101	rd <- [rs + simm10]	load byte value with address rs value + simm10, and store sign extended one to rd
LBU rd, simm10(rs)	000101	rd <- [rs + simm10]	load byte value with address rs value + simm10, and store to rd without sign extension
SW rs, simm10(rd)	001101	[rd + simm10] <- rs	load value from rs, and store word value to address rs value + simm10
SB rs, simm10(rd)	110101	[rd + simm10] <- rs	load value from rs, and store value[7:0] to address rs value + simm10

3 16-bit Length Instructions

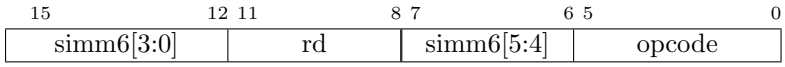
3.1 Integer Computational Instructions

Integer Register-Register Instructions

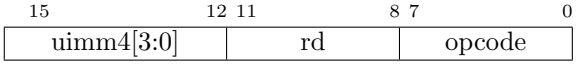
15 12	11 8	7 0
rs	rd	opcode
4	4	8
src	dest	MOV
src	dest	ADD2
src	dest	SUB2
src	dest	AND2
src	dest	XOR2
src	dest	OR2
src	dest	LSL2
src	dest	LSR2
src	dest	ASR2

Instruction	Opcode	Formula	Description
MOV rd, rs	11000000	rd <- rs	load rs value, and store to rd
ADD2 rd, rs	10000000	rd <- rd + rs	add rs1 value to rd value, and store to rd
SUB2 rd, rs	10001000	rd <- rd - rs	subtract rd value from rs value, and store to rd
AND2 rd, rs	10010000	rd <- rd & rs	do bit and between rd value and rs value, and store to rd
XOR2 rd, rs	10011000	rd <- rd ^ rs	do bit xor between rd value and rs value, and store to rd
OR2 rd, rs	10100000	rd <- rd | rs	do bit or between rd value and rs value, and store to rd
LSL2 rd, rs	10101000	rd <- rd << rs	do left logical shift rd value with width rs value, and store to rd
LSR2 rd, rs	10110000	rd <- rd >> rs	do right logical shift rd value with width rs value, and store to rd
ASR2 rd, rs	10111000	rd <- rd >>> rs	do right arithmetic shift rd value with width rs value, and store to rd

Integer Register-Immediate Instructions



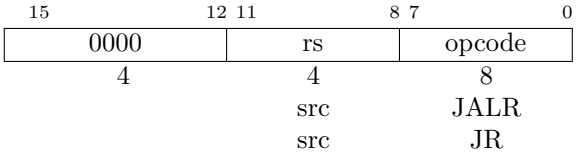
4	4	2	6
simm6[3:0]	dest	simm6[5:4]	ADDI2
simm6[3:0]	dest	simm6[5:4]	ANDI2
simm6[3:0]	dest	simm6[5:4]	LSI
simm6[3:0]	dest	simm6[5:4]	LUI

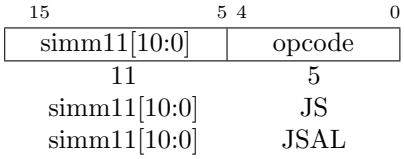


4	4	8
uimm4[3:0]	dest	LSLI2
uimm4[3:0]	dest	LSRI2
uimm4[3:0]	dest	ASRI2

Instruction	Opcode	Formula	Description
ADDI2 rd, simm6	000010	rd <- rd + simm6	add rd value to simm6, and store to rd
ANDI2 rd, simm6	010010	rd <- rd & simm6	do bit and between rd value and simm6, and store to rd
LSI rd, simm6	110100	rd <- simm6	load simm6, and store to rd
LUI rd, simm6	000100	rd <- (simm6 <<10)	load simm6 with left logical shift with width 10, and store to rd
LSLI2 rd, uimm4	00101010	rd <- rd << uimm4	do left logical shift rd value with width uimm4, and store to rd
LSRI2 rd, uimm4	00110010	rd <- rd >> uimm4	do right logical shift rd value with width uimm4, and store to rd
ASRI2 rd, uimm4	00111010	rd <- rd >>> uimm4	do rigth arithmetic shift rd value with width uimm4, and store to rd

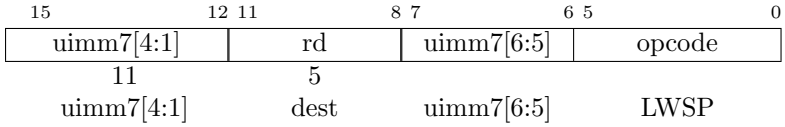
3.2 Unconditional Jumps

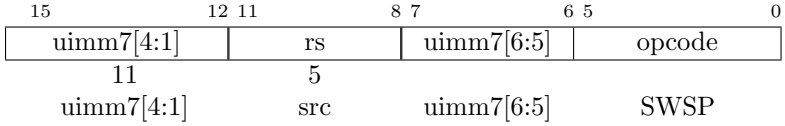




Instruction	Opcode	Formula	Description
JALR rs	00010110	ra <- PC + 2, PC <- rs	jump to rs value, and store PC + 2 to return address register ra
JR rs	00000110	PC <- rs	jump to rs value
JS simm11	01110	PC <- PC + simm11	jump to PC + simm11
JSAL simm11	11110	ra <- PC + 2, PC <- PC + simm11	jump to PC + 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 <- [sp + uimm7]	load word value with address stack pointer register value + uimm7, and store to rd
SWSP rs, uimm7(sp)	011100	[sp + uimm7] <- rs	store rs value to uimm8 with address stack pointer register value + uimm7

4 Instructions List