LCU				
Inst	Operands	Name	Description	
nop	nop	No operation		rd must be a local register of the LCU so R0-R3 or SRF(i) rs1 can be a local register R0-R3, ZERO, SRF(i) or an inmediate rs2 can be a local register R0-R3, ZERO, ONE or SRF(i) Br_mode = 0
sadd	sadd rd, rs1, rs2	Signed add	rd = rs1 + rs2	
ssub	ssub rd, rs1, rs2	Signed substraction	rd = rs1 - rs2	
land	land rd, rs1, rs2	Logical and	rd = rs1 & rs2	
lor	lor rd, rs1, rs2	Logical or	rd = rs1 | rs2	
lxor	lxor rd, rs1, rs2	Logical xor	rd = rs1 ^ rs2	
sll	sll rd, rs1, rs2	Shift left logical	rd = rs1 << rs2[3:0]	
srl	srl rd, rs1, rs2	Shift right logical	rd = rs1 >> rs2[3:0]	
sra	sra rd, rs1, rs2	Shift right arithmetic	rd = rs1 >> rs2[3:0]	
jump	jump [rd,] rs1, rs2	Jump	PC = rs1 + rs2	rd is optional and can be a local register of the LCU so R0-R3 or SRF(i). rs1 can be a local register R0-R3, ZERO, ONE or SRF(i) rs2 can be a local register R0-R3, ZERO, SRF(i) or an inmediate Br_mode = 0
beq	beq [rd,] rs1, rs2, imm	Branch if equal	if (rs1 == rs2) PC = imm	
bne	bne [rd,] rs1, rs2, imm	Branch if not equal	if (rs1 != rs2) PC = imm	
blt	blt [rd,] rs1, rs2, imm	Branch if lower than	if (rs1 < rs2) PC = imm	
bgepd	bgepd [rd,] rs1, rs2, imm	Branch if greater or equal with pre-decrement	if ((--rs1) == rs2) PC = imm	
beqr	beqr imm	Branch if equal (rc mode)	if (beq_flag_RC_i | beq_flag_RC_j: for all RC_i and RC_j of the column) PC = imm	Br_mode = 1
bner	bner imm	Branch if not equal (rc mode)	if (bne_flag_RC_i | bne_flag_RC_j: for all RC_i and RC_j of the column) PC = imm	
bltr	nltr imm	Branch if lower than (rc mode)	if (blt_flag_RC_i | blt_flag_RC_j: for all RC_i and RC_j of the column) PC = imm	
bger	bger imm	Branch if greater or equal (rc mode)	if (bge_flag_RC_i | bge_flag_RC_j: for all RC_i and RC_j of the column) PC = imm	
exit	exit	Exit		

LSU				
Inst	Operands	Name	Description	
sadd	sadd rd, rs1, rs2	Signed add	rd = rs1 + rs2	Artihmetic ops rd must be a local register of the LSU so R0-R7, SRF(i) rs1 can be a local register R0-R7, ZERO, ONE, TWO or SRF(i) rs2 can be a local register R0-R3, ZERO, ONE, TWO or SRF(i) Since there is no NOP is nothing wants to be done, the rd must be set to _ so a write would not be performed to any register
ssub	ssub rd, rs1, rs2	Signed substraction	rd = rs1 - rs2	
land	land rd, rs1, rs2	Logical and	rd = rs1 & rs2	
lor	lor rd, rs1, rs2	Logical or	rd = rs1 | rs2	
lxor	lxor rd, rs1, rs2	Logical xor	rd = rs1 ^ rs2	
sll	sll rd, rs1, rs2	Shift left logical	rd = rs1 << rs2[3:0]	
srl	srl rd, rs1, rs2	Shift right logical	rd = rs1 >> rs2[3:0]	
bitrev	bitrev rd, rs1, rs2	Bit reversal operation	rd = reverse(rs1) >> rs2[2:0] where reverse(rs1)={rs1[0], rs1[1],...,rs1[n-1], rs1[n]}	
nop	nop	No operation		Memory ops dest can be VWR_A, VWR_B, VWR_C or SRF
ld.vwr	ld.vwr dest	Load a line of the SPM to a VWR	VWR_X <- SPM(R7)	
st.vwr	str.vwr dest	Store VWR to SPM	SPM(R7) <- VWR_X	
sh.il.up	shilup	Shuffle VWRA and VWRB interleaving upper part	VWRC <- VWRA[0], VWRB[0], ..., VWRA[63], VWRB[63]	Shuffle ops
sh.il.lo	shillo	Shuffle VWRA and VWRB interleaving lower part	VWRC <- VWRA[64], VWRB[64], ..., VWRA[127], VWRB[127]	
sh.even	sheven	Shuffle VWRA and VWRB even indexes	VWRC <- VWRA[0], VWRA[2], ..., VWRA[126], VWRB[0], ...,VWRB[124], VWRB[126]	
sh.odd	shodd	Shuffle VWRA and VWRB odd indexes	VWRC <- VWRA[1], VWRA[3], ..., VWRA[127], VWRB[1], ...,VWRB[125], VWRB[127]	
sh.bre.up	shbreup	Shuffle VWRA and VWRB concatenated bit reversal upper part	VWRC <- VWRA[0], VWRB[0], VWRA[64], VWRB[64], ..., VWRA[126], VWRB[126]	
sh.bre.lo	shbrelo	Shuffle VWRA and VWRB concatenated bit reversal lower part	VWRC <- VWRA[1], VWRB[1], VWRA[65], VWRB[65], ..., VWRA[127], VWRB[127]	
sh.cshift.up	shcshiftup	Shuffle VWRA and VWRB concatenated slice circular shift up upper part	VWRC <- VWRA[1], VWRA[2], ..., VWRA[127], VWRB[0]	
sh.cshift.lo	shcshiftlo	Shuffle VWRA and VWRB concatenated slice circular shift up lower part	VWRC <- VWRB[1], VWRB[2], ..., VWRB[127], VWRA[0]	
nop	nop	nop for programming simplicity for arith ops, in real hw it would be a land without writting the result		
			Bit reversal order: 0, 64, 32, 96, 16, 80, 48, 112, 8, 72, 40, 104, 24, 88, 56, 120, 4, 68, 36, 100, 20, 84, 52, 116, 12, 76, 44, 108, 28, 92, 60, 124, 2, 66, 34, 98, 18, 82, 50, 114, 10, 74, 42, 106, 26, 90, 58, 122, 6, 70, 38, 102, 22, 86, 54, 118, 14, 78, 46, 110, 30, 94, 62, 126, 1, 65, 33, 97, 17, 81, 49, 113, 9, 73, 41, 105, 25, 89, 57, 121, 5, 69, 37, 101, 21, 85, 53, 117, 13, 77, 45, 109, 29, 93, 61, 125, 3, 67, 35, 99, 19, 83, 51, 115, 11, 75, 43, 107, 27, 91, 59, 123, 7, 71, 39, 103, 23, 87, 55, 119, 15, 79, 47, 111, 31, 95, 63, 127	

RC

Inst	Operands	Name	Description	Notes	
nop	nop	No operation			
sadd	sadd rd, rs1, rs2	Signed add	rd = rs1 + rs2		rd can be: a local register of the RC, so R0-R1; a special register ROUT, so the neighbours can access the value but it is not stored further; an element on it's slice of a VWR, so VWR_A, VWR_B or VWR_C; or, (only for RC0) a register of the SRF, so SRF(i). rs1 and rs2 can be the same as rd plus ONE, ZERO, MAX_INT or MIN_INT. Any op is written to ROUT without specifying it
ssub	ssub rd, rs1, rs2	Signed substraction	rd = rs1 - rs2		
smul	smul rd, rs1, rs2	Signed multiplication	rd = rs1 * rs2		
sdiv	sdiv rd, rs1, rs2	Signed division	rd = rs1 / rs2		
sll	sll rd, rs1, rs2	Shift left logical	rd = rs1 << rs2[3:0]		
srl	srl rd, rs1, rs2	Shift right logical	rd = rs1 >> rs2[3:0]		
sra	sra rd, rs1, rs2	Shift right arithmetic	rd = rs1 >> rs2[3:0]		
land	land rd, rs1, rs2	Logical and	rd = rs1 & rs2		
lor	lor rd, rs1, rs2	Logical or	rd = rs1 | rs2		
lxor	lxor rd, rs1, rs2	Logical xor	rd = rs1 ^ rs2		
sfga	sfga rd, flag	Input of MUXA out if sign flag = 1 else input of MUXB out	If (sign_flag == 1) rd = rs1 else rd = rs2		flag can be OWN, RCT, RCB, RCR or RCL
zfga	zfga rd, flag	Input of MUXA out if zero flag = 1 else input of MUXB out	If (zero_flag == 1) rs1 else rs2		
mul.fp	mul.fp rd, rs1, rs2	Fixed point multiplication	rd = rs1 * rs2	Uses 1 bit for the sign, half the datapath width for the integer part of the number and the rest of the datapath width for the decimal part of the number	rd same rs1 and rs2 can be the same as rd plus ONE, ZERO, MAX_INT or MIN_INT.
div.fp	div.fp rd, rs1, rs2	Fixed point division (reserved but not implemented)			
sadd.h	sadd rd, rs1, rs2	Signed add with 16-bit precision (reserved but not implemented)	rd = { rs1[31:16] + rs2[31:16], rs1[15:0] + rs2[15:0] }	16-bit precision implies that each register has two values of 16-bit, so the operation would be done to both	
ssub.h	ssub rd, rs1, rs2	Signed substraction with 16-bit precision (reserved but not implemented)	rd = { rs1[31:16] - rs2[31:16], rs1[15:0] - rs2[15:0] }		
smul.h	smul rd, rs1, rs2	Signed multiplication with 16-bit precision (reserved but not implemented)			
sdiv.h	sdiv rd, rs1, rs2	Signed division with 16-bit precision (reserved but not implemented)			
sll.h	sll rd, rs1, rs2	Shift left logical with 16-bit precision (reserved but not implemented)			
srl.h	srl rd, rs1, rs2	Shift right logical with 16-bit precision (reserved but not implemented)			
sra.h	sra rd, rs1, rs2	Shift right arithmetic with 16-bit precision (reserved but not implemented)			
land.h	land rd, rs1, rs2	Logical and with 16-bit precision (reserved but not implemented)			
lor.h	lor rd, rs1, rs2	Logical or with 16-bit precision (reserved but not implemented)			
lxor.h	lxor rd, rs1, rs2	Logical xor with 16-bit precision (reserved but not implemented)			

MXCU				
Inst	Operands	Name	Description	
nop				Artihmetic ops rd must be a local register of the MXCU so R0-R7 or SRF(i) rs1 can be a local register R0-R7, ZERO, ONE, TWO, HALF(=15), LAST(=31) or SRF(i) rs2 can be a local register R0-R3, ZERO, ONE, TWO, HALF(=15), LAST(=31) or SRF(i)
sadd	sadd rd, rs1, rs2	Signed add	rd = rs1 + rs2	
ssub	ssub rd, rs1, rs2	Signed substraction	rd = rs1 - rs2	
sll	sll rd, rs1, rs2	Shift left logical	rd = rs1 << rs2[3:0]	
srl	srl rd, rs1, rs2	Shift right logical	rd = rs1 >> rs2[3:0]	
land	land rd, rs1, rs2	Logical and	rd = rs1 & rs2	
lor	lor rd, rs1, rs2	Logical or	rd = rs1 | rs2	
lxor	lxor rd, rs1, rs2	Logical xor	rd = rs1 ^ rs2	