Appendix D

ArdEx Instruction Set Reference

D.1 Addressing Modes

The ArdEx instruction set supports four different categories of argument. Each of these categories supports one or more addressing modes. This section describes these addressing modes and provides a key to the next section where the instructions themselves are described.

Mode	Syntax	Description
Immediate	#*num*	value = *num*
Absolute	*num*	value = MEM[*num*]
Register	R_N	value = R_N
Register Indirect	@R_N	value = MEM[R_N]
Register Indirect Postincrement	@R_N+	value = MEM[R_N]; R_N = R_N + 1
Register Indirect Predecrement	@–R_N	R_N = R_N - 1; value = MEM[R_N]
16-bit Literal	*num*	value = *num*
16-bit Register	Reg_M,Reg_N	value = $256 \times Reg_M^+ + Reg_N^+$
Branch Constant	*num*	value = *num*
Branch Register	Reg_N	value = Reg_N^+

Immediate value stored as part of the program code to be used as a *src* operand.

Absolute address from which a *src* is loaded or to which a *dst* is stored.

Register contents of a CPU register.

Register Indirect memory address pointed to by the contents of a CPU register.

Register Indirect With Postincrement Same as Register Indirect except register is incremented after memory access.

Register Indirect With Predecrement Same as Register Indirect except CPU register is decremented before memory access.

16-bit Literal numeric value in the range [0:65535].

16-bit Register pair of registers permitting *any* of the register addressing modes. Value is first register $\times$ 256 + second register.

Branch Constant branch destination step number in the range [0:255].

Branch Register A register permitting *any* of the register addressing modes. The resulting 8-bit value is the branch destination step number.

Key to Arguments

The next section describes all the instructions and their arguments. Each argument identifies the legal addressing modes as listed here.

src Immediate, Absolute, Register, Register Indirect, Register Indirect Postincrement, Register Indirect Predecrement.

dst Absolute, Register, Register Indirect, Register Indirect Postincrement, Register Indirect Predecrement.

val16 16-bit Literal, 16-bit Register

step Branch Constant, Branch Register

D.2 Instruction Set

ADD

ADD	*src,dst*	$dst = src + dst$

Affected flags: ZC

Calculates the 8-bit sum of *src* and *dst* and stores the result in *dst*. Zero flag set if result is 0, cleared otherwise. Carry flag set if result $>$ `0xFF`, cleared otherwise.

ADDC

ADDC	*src,dst*	$dst = src + dst + Carry$

Affected flags: ZC

Calculates the 8-bit sum of the existing Carry flag, *src* and *dst* and stores the result in *dst*. Zero flag set if result is 0, cleared otherwise. Carry flag set if result $>$ `0xFF`, cleared otherwise.

AND

AND	*src,dst*	$dst = src \cap dst$

Affected flags: Z

Calculates the bitwise AND of *src* and *dst* and stores the result in *dst*. Zero flag is set if all bits of the result are 0, cleared otherwise. In short, the result has 1 bits only where both arguments have 1 bits.

Bitwise AND truth table:

LHS	RHS	$\rightarrow$	AND
0	0		0
0	1		0
1	0		0
1	1		1

BIC

BIC	*src,dst*	$dst = \overline{src} \cap dst$

Affected flags: None

Clears any bits in *dst* that are set in *src*. Flags are not affected.

BIS

BIS	*src,dst*	$dst = src \cup dst$

Affected flags: None

Sets any bits in *dst* that are set in *src*. Flags are not affected.

BIT

BIT	*src,dst*	DISCARD$(src \cap dst)$

Affected flags: ZC

Calculates the bitwise AND of *src* and *dst*. Zero flag is set if all bits of the result are 0, cleared otherwise. Carry flag is set if any bit of the result is 1, cleared otherwise. The result itself is discarded.

CALL

CALL	*step*	SAVE($current\_step$); $jump\ step$

Affected flags: None

Execution jumps to *step* after the step number of the next instruction has been saved on the top of the return stack. A later `RET` instruction will retrieve this return address and resume execution there.

If calls are nested deeper than 5 levels, execuation is halted with an error message.

Flags are not affected.

CMP

CMP	*src,dst*	DISCARD($dst - src$)

Affected flags: ZC

Performs a temporary subtraction of *src* from *dst*. The Zero flag is set *src* and *dst* are equal, cleared otherwise. The Carry flag is set if *src* $>$ *dst*, cleared otherwise. The result itself is discarded.

IN

IN	*dst*	$dst = \text{INPUT}()$

Affected flags: Z

Accepts up to one byte of input from the user console. If no input is available, the Zero flag is set. Otherwise the input byte is stored at *dst* and the Zero flag is cleared.

JC

JC	*step*	$if\ Carry\ jump\ step$

Affected flags: None

If the Carry flag is set start executing at *step*.

JEQ

JEQ	*step*	$if\ Zero\ jump\ step$

Affected flags: None

If the Zero flag is set start executing at *step*.

JMP

JMP	*step*	$jump\ step$

Affected flags: None

Start executing at *step*.

JNC

JNC	*step*	$if\ \overline{Carry}\ jump\ step$

Affected flags: None

If the Carry flag is clear start executing at *step*.

JNE

JNE	*step*	$if\ \overline{Zero}\ jump\ step$

Affected flags: None

If the Zero flag is clear start executing at *step*.

MOV

MOV	*src,dst*	$dst = src$

Affected flags: None

Copy the contents of *src* to *dst*. Flags are not affected.

OR

OR	*src,dst*	$dst = src \cup dst$

Affected flags: Z

Calculates the bitwise OR of *src* and *dst* and stores the result in *dst*. Zero flag is set if all bits of the result are 0, cleared otherwise. In short, the result has 1 bits where either argument has 1 bits.

Bitwise OR truth table:

LHS	RHS	→	OR
0	0		0
0	1		1
1	0		1
1	1		1

OUT

OUT	*val16*	$\mathrm{OUTPUT}(val16 \cap 0xFF)$

Affected flags: None

Outputs the low-order byte of *val16* to the user console. Flags are not affected.

RET

RET	$jump\ last\ saved\ step$

Affected flags: None

Takes the last saved value from the return stack and execution resumes at that step. Flags are not affected.

If the return stack is empty, execution is halted with an error message.

ROL

ROL	*dst*	$dst = 2 \times dst + Carry$

Affected flags: ZC

Shifts all bits in *dst* left one place. Carry flag is shifted in to BIT_0. BIT_7 is copied to the Carry flag. If all bits in *dst* are zero, Zero flag is set, otherwise it is cleared.

ROR

ROR	*dst*	$dst = dst/2 + 0x80 \times Carry$

Affected flags: ZC

Shifts all bits in *dst* right one place. Carry flag is shifted in to BIT_7. BIT_0 is copied to the Carry flag. If all bits in *dst* are zero, Zero flag is set, otherwise it is cleared.

SL

SL	*dst*	$dst = 2 \times dst$

Affected flags: ZC

Shifts all bits in *dst* left one place. A 0 is copied in to BIT_0. BIT_7 is copied to the Carry flag. If all bits in *dst* are zero, Zero flag is set, otherwise it is cleared.

SR

SR	*dst*	$dst = dst/2$

Affected flags: ZC

Shifts all bits in *dst* right one place. A 0 is copied in to BIT_7. BIT_0 is copied to the Carry flag. If all bits in *dst* are zero, Zero flag is set, otherwise it is cleared.

STOP

STOP		HALT()

Affected flags: None

Execution halts.

SUB

SUB	*src,dst*	$dst = dst - src$

Affected flags: ZC

Subtracts *src* from *dst*. The Zero flag is set if the result is 0. The Carry flag is set if *src* $>$ *dst*.

SUBC

SUBC	*src,dst*	$dst = dst - src - Carry$

Affected flags: ZC

Subtracts *src* + Carry from *dst*. The Zero flag is set if the result is 0. The Carry flag is set if *src* + Carry $>$ *dst*.

XOR

XOR	*src,dst*	$dst = (src \cap \overline{dst}) \cup (dst \cap \overline{src})$

Affected flags: Z

Calculates the bitwise EXCLUSIVE OR of *src* and *dst* and stores the result in *dst*. Zero flag is set if all bits of the result are 0, cleared otherwise. In short, the result has 1 bits where only one of the arguments has 1 bits.

Bitwise XOR truth table:

LHS	RHS	→	XOR
0	0		0
0	1		1
1	0		1
1	1		0

WAIT

WAIT	*val16*	PAUSE($val16 \times 100\mu s$)

Affected flags: None

Execution stops for *val16*/10 milliseconds. No flags are set and execution resumes at the next instruction.