Ariel CPU Cheat Sheet

CPU Information

The Ariel CPU is a 32-bit CPU that uses registers and instructions similar to the ARM CPU, but it's both simplified and more versatile.

Each instruction is stored on minimum 2, maximum 8 bytes, and they must be aligned to even memory addresses.

The registers store 32-bit unsigned values, that can be also handled as signed values by certain instructions. The special registers like PC and SP are not sharing space with normal registers, R0 – R15 are completely independent from the special registers.

Coming soon: Floating Point instructions that use the normal registers.

Registers

Register	Type	Description
`R0 - R15`	Normal	Register to store 32-bit values and memory addresses
`PC`	Special	Program Counter that points to an executable instruction
`SR`	Special	Status Register with CPU flags and status codes
`SP`	Special	Stack Pointer that points to the current position in Stack
`FP`	Special	Stack Frame Pointer (not in use)
`LR`	Special	Link Register that stores the return address for Linked Subroutines

Addressing Modes

Implied

Implied instructions do not have parameter values.

Register

A single register, that typically can be a Normal (general-purpose) or a Special register. Most instructions accept only Normal registers.

```
POP R0
POP SP
PUSH R0
PUSH SP
```

Register Range

A range of Normal (general-purpose) registers, highest register value included.

```
POP R0-R15
PUSH R8-R11
CLR R4-R9
```

Registers

Two Normal (general-purpose) registers in a Destination, Source order.

```
ABS R0, R2
NOT R4, R5
```

Register Value

A Normal (general-purpose) register used as Destination. The Source value may be a Normal register or an Immediate value up to 32 bits. The Immediate value sometimes needs to be a small value up to 15 (0x0F), or up to 31 (0x1F) for bit shifting instructions. The MOV instruction is unique, it can use any Normal or Special register combinations.

```
INC R0, #$0F
MOV R2, #$1234
MOV R0, R2
MOV SP, R2
MOV R2, SP
MOV LR, SP
```

Register Combination

A Normal (general-purpose) register used as Destination and as Source. The third Option value may be a Normal register or an Immediate value up to 32 bits.

```
EOR R0, R2, R4
ORR R0, R2, #$12345678
AND R0, R2, #-$1234
```

Compare

Similar to the Register Combination addressing mode, but the Source and Option values are just compared with CPU flags set or cleared, and the end result is not stored in a Destination register (therefore Rd is not specified).

```
CMP R0, R2
CMP R2, #$1234
```

Memory Address

This is the most complex instruction addressing method with a lot of options. It is used to either…

- **Load** a value from a memory address into a register via **LDR**
- **Store** a value from a register into a memory address via **STR**
- **Create** a calculated memory address pointer via **ADR**
- **Swap** a register value and a memory value (atomic) via **SWP**
- **Jump** to a memory address via **JMP**

JMP can only use the *Absolute*, *Indirect*, and *Indirect using Rd* types.

Relative, absolute, and indirect memory addresses

Type	Description	Example

Relative (15+1 bit)	Relative memory address from the instruction's memory address	LDR R0, [PC, #$1234]
		LDR R0, [PC, #-$1234]
		LDR R0, [PC, Label]

Relative (31+1 bit)	Relative memory address from the instruction's memory address	LDR R0, [PC, #$123456]
		LDR R0, [PC, #-$123456]
		LDR R0, [PC, Far-Label]

Absolute	Absolute memory address (32-bit value)	LDR R0, $12345678
		SWP R0, $12345678
		JMP $12345678

Indirect	Memory address (32-bit value) loaded from an absolute memory address	JMP [$12345678]
Indirect using Register Destination	Memory address (32-bit value) loaded from a register value	JMP [R0]

Indirect memory addresses with indexing and modification

Type	Description	Example
Indirect using Register	Indirect memory address using a register value	LDR R0, [R1]
Indirect using Register with Register	Indirect memory address using a register value, offset by a register value	LDR R0, [R1, R2]
Indirect using Register with Register, Pre-Indexed	Indirect memory address using a register value. First the Source register gets modified by the Option register's value (R1 = R1+R2)	LDR R0, [R1, R2]!
Indirect using Register with Register, Post-Indexed	Indirect memory address using a register value. Afterwards the Source register gets modified by the Option register's value (R1 = R1+R2)	LDR R0, [R1], R2
Indirect using Register with Signed Byte value	Indirect memory address using a register value, offset by a signed Byte value	LDR R0, [R1, #$12] LDR R0, [R1, #-$12]
Indirect using Register with Signed Byte value, Pre-Indexed	Indirect memory address using a register value. First the Source register gets modified by the signed Byte value (R1 = R1+$12)	LDR R0, [R1, #$12]! LDR R0, [R1, #-$12]!
Indirect using Register with Signed Byte value, Post-Indexed	Indirect memory address using a register value. Afterwards the Source register gets modified by the signed Byte value (R1 = R1+$12)	LDR R0, [R1], #$12 LDR R0, [R1], #-$12
Indirect using Register with Signed Word value	Indirect memory address using a register value, offset by a signed Word value	LDR R0, [R1, #$1234] LDR R0, [R1, #-$1234]
Indirect using Register with Signed Word value, Pre-Indexed	Indirect memory address using a register value. First the Source register gets modified by the signed Word value (R1 = R1+$1234)	LDR R0, [R1, #$1234]! LDR R0, [R1, #-$1234]!
Indirect using Register with Signed Word value, Post-Indexed	Indirect memory address using a register value. Afterwards the Source register gets modified by the signed Word value (R1 = R1+$1234)	LDR R0, [R1], #$1234 LDR R0, [R1], #-$1234

Instructions

Implied Instructions

Mnemonic	Addressing	Description
NOP	Implied	No operation
AFH	Implied	No operation - Placeholder for Ariel File Header $41
HALT	Implied	Halt the CPU (No recovery from this state)
WAIT	Implied	Wait for an Interrupt (IRQ or NMI), then continue
EI	Implied	Enable All Maskable Interrupts (Chosen by IRQ Select)
DI	Implied	Disable All Maskable Interrupts
SEC	Implied	Set the Carry flag
CLC	Implied	Clear the Carry flag
NEC	Implied	Negate the Carry flag
SEV	Implied	Set the Overflow flag
CLV	Implied	Clear the Overflow flag
NEV	Implied	Negate the Overflow flag
RTS	Implied	Return from Subroutine, using the Stack POP PC
RTL	Implied	Return from Subroutine, using the Link Register LR -> PC
RTI	Implied	Return from Interrupt, using the Stack POP PC, POP SR

Transfer Instructions

Mnemonic	Addressing	Description	Example
POP	Register	Retrieve a value from Stack	POP R0 POP SP
POP	Register range	Retrieve multiple values from Stack	POP R0-R15
PUSH	Register	Store a value on Stack	PUSH R0 PUSH SP
PUSH	Register range	Store multiple values on Stack	PUSH R0-R15
CLR	Register range	Store Zero into multiple Registers	CLR R0-R15

MOV	Register value	Stores a value in a Register	MOV R0, R2
			MOV SP, R0
			MOV R0, SP
			MOV FP, SP
			MOV R0, #$12345678

MOVB	Register combination	Move a Register's selected Byte into a Register (#0-3)	MOVB R0, R2, #3 MOVB R0, R2, R4
MOVW	Register combination	Move a Register's selected Word into a Register (#0-1)	MOVW R0, R2, #1 MOVW R0, R2, R4
MOVT	Register value	Move a Register or Value's Low Word into a Register's High Word	MOVT R0, #$1234 MOVT R0, R2
SWP	Registers	Swap the values of two Registers (atomic)	SWP R0, R2
SWP	Address	Swap the values of a Memory Address and a Register (atomic)	SWP R0, $12345678

Storage Instructions

Mnemonic	Addressing	Description	Example
SWP	Address	Swap the values of a Memory Address and a Register (atomic)	SWP R0, $12345678
ADR	Address	Store a calculated Memory Address in a Register	ADR R0, $12345678
LDR	Address	Load a value from a Memory Address into a Register (Double Word)	LDR R0, $12345678
LDRB	Address	Load a value from a Memory Address into a Register (Byte)	LDRB R0, $12345678
LDRW	Address	Load a value from a Memory Address into a Register (Word)	LDRW R0, $12345678
LDRBS	Address	Load a value from a Memory Address into a Register (Signed Byte)	LDRBS R0, $12345678
LDRWS	Address	Load a value from a Memory Address into a Register (Signed Word)	LDRWS R0, $12345678
STR	Address	Store the value of a Register at a Memory Address (Double Word)	STR R0, $12345678
STRB	Address	Store the value of a Register at a Memory Address (Byte)	STRB R0, $12345678
STRW	Address	Store the value of a Register at a Memory Address (Word)	STRW R0, $12345678
STRBS	Address	Store the value of a Register at a Memory Address (S. Byte)	STRBS R0, $12345678
STRWS	Address	Store the value of a Register at a Memory Address (S. Word)	STRWS R0, $12345678

For more complex examples, see the Addressing Method descriptions.

Bitwise Instructions

Mnemonic	Addressing	Description	Example
LSL	Register combination	Shift bits of a Register to the Left (#0-32)	LSL R0, R2, R4 LSL R0, R2, #$12
LSR	Register combination	Shift bits of a Register to the Right (#0-32)	LSR R0, R2, R4 LSR R0, R2, #$12
ROL	Register combination	Rotate bits of a Register to the Left (#0-32)	ROL R0, R2, R4 ROL R0, R2, #$12
ROR	Register combination	Rotate bits of a Register to the Right (#0-32)	ROR R0, R2, R4 ROR R0, R2, #$12
RCL	Register combination	Rotate bits of a Register to the Left, through Carry (#0-32)	RCL R0, R2, R4 RCL R0, R2, #$12
RCR	Register combination	Rotate bits of a Register to the Right, through Carry (#0-32)	RCR R0, R2, R4 RCR R0, R2, #$12
ASR	Register combination	Arithmetic Shift the bits of a Register to the Right (#0-32)	ASR R0, R2, R4 ASR R0, R2, #$12
AND	Register combination	Logical AND of a Register's value	AND R0, R2, R4 AND R0, R2, #$12345678
ORR	Register combination	Logical OR of a Register's value	ORR R0, R2, R4 ORR R0, R2, #$12345678
EOR	Register combination	Logical EOR of a Register's value	EOR R0, R2, R4 EOR R0, R2, #$12345678
NOT	Registers	Logical NOT of a Register's value	NOT R0, R2
CLZ	Registers	Count Leading Zeros of a Register's value	CLZ R0, R2
CTZ	Registers	Count Trailing Zeros of a Register's value	CTZ R0, R2
RBIT	Registers	Reverse All Bits of a Register's value	RBIT R0, R2
REVB	Registers	Reverse Endianness of a Register's value	REVB R0, R2
REVW	Registers	Reverse the Low and High Words of a Register's value	REVW R0, R2

Arithmetic Instructions

Mnemonic	Addressing	Description	Example
ADD	Register combination	Add values into a Register	ADD R0, R2, R4 ADD R0, R2, #$12345678
ADC	Register combination	Add values into a Register, using the Carry flag	ADC R0, R2, R4 ADC R0, R2, #$12345678
SUB	Register combination	Subtract values into a Register	SUB R0, R2, R4 SUB R0, R2, #$12345678
SBC	Register combination	Subtract values into a Register, using the Carry flag	SBC R0, R2, R4 SBC R0, R2, #$12345678
RSB	Register combination	Reverse Subtract values into a Register	RSB R0, R2, R4 RSB R0, R2, #$12345678
RSC	Register combination	Reverse Subtract values into a Register, using the Carry flag	RSC R0, R2, R4 RSC R0, R2, #$12345678
DIV	Register combination	Divide values into a Register	DIV R0, R2, R4 DIV R0, R2, #$12345678
DIVS	Register combination	Divide Signed values into a Register	DIVS R0, R2, R4 DIVS R0, R2, #$12345678
REM	Register combination	Division Remainder of values into a Register	DIV R0, R2, R4 DIV R0, R2, #$12345678
REMS	Register combination	Division Remainder of Signed values into a Register	DIVS R0, R2, R4 DIVS R0, R2, #$12345678
MUL	Register combination	Multiply values into a Register	MUL R0, R2, R4 MUL R0, R2, #$12345678
MULS	Register combination	Multiply Signed values into a Register	MULS R0, R2, R4 MULS R0, R2, #$12345678
ABS	Registers	Store the Absolute value of a Register into a Register	ABS R0, R2
NEG	Registers	Store the Negated value of a Register into a Register	NEG R0, R2
INC	Register value	Increase a Register's value by a small value (1 by default)	INC R0 INC R0, #4
DEC	Register value	Decrease a Register's value by a small value (1 by default)	DEC R0 DEC R0, #4

Conversion Instructions

Mnemonic	Addressing	Description	Example
CFB	Registers	Convert From a Byte Value (Use only the lowest 8 bits)	CFB R0, R2
CFBS	Registers	Convert From a Signed Byte Value (Use only the lowest 8 bits)	CFBS R0, R2
CFW	Registers	Convert From a Word Value (Use only the lowest 16 bits)	CFW R0, R2
CFWS	Registers	Convert From a Signed Word Value (Use only the lowest 16 bits)	CFWS R0, R2
CFBCD	Registers	Convert From a Binary Coded Decimal Value, like $99999999	CFBCD R0, R2
CTB	Registers	Convert To a Byte Value (Use only the lowest 8 bits)	CTB R0, R2
CTBS	Registers	Convert To a Signed Byte Value (Truncate to 7+1 bits)	CTBS R0, R2
CTW	Registers	Convert To a Word Value (Truncate to 16 bits)	CTW R0, R2
CTWS	Registers	Convert To a Signed Word Value (Truncate to 15+1 bits)	CTWS R0, R2
CTBCD	Registers	Convert To a Binary Coded Decimal Value, like $99999999	CTBCD R0, R2

Conditional Branch and Subroutine Call Instructions

Branch and Subroutine Call instructions can operate Always or based on certain CPU Flag Conditions. They use relative memory address pointers and Subroutine calls typically return to the next instruction after the call.

The difference between the instructions (with or without Condition)

- BRA and JMP branch (jump) and don't return afterwards.
- JSR Pushes the next instruction's address to the Stack, and RTS returns by Popping this value back into the Program Counter.
- LNK stores the next instruction's address in the Link Register (LR), and RTL returns by moving LR's value back into the Program Counter.

Mnemonic	Return via	Description	Example
BRA	-	Branch to an instruction at the selected Memory Address, Always.	BRA Label
B(Cond.)	-	Branch to an instruction at the selected Memory Address, If the selected CPU Flag Condition is True.	BEQ Label BNE Label BCS Label
JSR	**RTS**	Jump to an instruction at the selected Memory Address, Always. Then return to the next instruction.	JSR Label
J(Cond.)	**RTS**	Jump to an instruction at the selected Memory Address, If the selected CPU Flag Condition is True. Then return to the next instruction.	JEQ Label JNE Label JCS Label
LNK	**RTL**	Jump to an instruction at the selected Memory Address, Always. Then return to the next instruction.	LNK Label
L(Cond.)	**RTL**	Jump to an instruction at the selected Memory Address, If the selected CPU Flag Condition is True. Then return to the next instruction.	LEQ Label LNE Label LCS Label
JMP	-	Jump to an Absolute or Indirect Memory Address, Always.	JMP $12345678 JMP [$12345678]

Conditions for Branch and Subroutine Call Instructions

See the **Comparer Instructions (CMP, CMN, TST, TEQ)** that set up the CPU Flags before these Conditional instructions get executed.

Condition	Description	Example
EQ	Equal Zero Flag set	BEQ Label JEQ Label / LEQ Label
NE	Not equal Zero Flag clear	BNE Label JNE Label / LNE Label
CS	Unsigned greater than or equal Carry Flag set	BCS Label JCS Label / LCS Label
CC	Unsigned less than Carry Flag clear	BCC Label JCC Label / LCC Label
MI	Minus / Negative Negative Flag set	BMI Label JMI Label / LMI Label
PL	Plus / Positive or zero Negative Flag clear	BPL Label JPL Label / LPL Label
VS	Overflow Overflow Flag set	BVS Label JVS Label / LVS Label
VC	No overflow Overflow Flag clear	BVC Label JVC Label / LVC Label
HI	Unsigned greater than C set and Z clear (C=1, Z=0)	BHI Label JHI Label / LHI Label
LS	Unsigned less than or equal C clear or Z set (C=0 or Z=1)	BLS Label JLS Label / LLS Label
GE	Signed greater than or equal N set and V set, or N clear and V clear (N==V)	BGE Label JGE Label / LGE Label
LT	Signed less than N set and V clear, or N clear and V set (N<>V)	BLT Label JLT Label / LLT Label
GT	Signed greater than Z clear, and either N set and V set, or N clear and V clear (Z=0, N=V)	BGT Label JGT Label / LGT Label
LE	Signed less than or equal Z set, or N set and V clear, or N clear and V set (Z=1 or N<>V)	BLE Label JLE Label / LLE Label

Comparer Instructions

Mnemonic	Addressing	Description	Example
CMP	Compare	Compare a Register value to an Immediate or Register value	CMP R0, R2 CMP R0, #$12345678
CMN	Compare	Negative Compare a Register value to an Immediate or Register value	CMN R0, R2 CMN R0, #$12345678
TST	Compare	Test Bits in a Register value	TST R0, R2 TST R0, #$12345678
TEQ	Compare	Test Equivalence in a Register value	TEQ R0, R2 TEQ R0, #$12345678

Interrupt Instructions

Mnemonic	Return via	Description	Example
BRK	**RTI**	Call the non-maskable Abort Interrupt. PUSH SR, PUSH PC+4	BRK #$1234
DBG	**RTI**	Call the non-maskable Debug Interrupt. PUSH SR, PUSH PC+4	DBG #$1234
SVC	**RTI**	Call the non-maskable Supervisor Call Interrupt. Used for Kernel Function Calls, when implemented. PUSH SR, PUSH PC+4	SVC #$1234
SWI	**RTS**	Call one of the 32 available Software Interrupt Vectors as a Subroutine. Used for soft Kernel Function Calls (#0-31). The input parameters should be in select Registers like R0. Typically, the return value is placed in R0 before returning. PUSH PC+2	SWI #0 SWI #31

Miscellaneous Instructions

Mnemonic	Addressing	Description	Example
RAND	Registers	Generate a Random value in a Register	RAND R0, R2
SEED	Register	Set the Random Seed using a Register value (R0-R15)	SEED R0