ADuC7XXX Software Quick Reference Guide — ARM Instruction Set

Key to table	
{cond}	Refer to Table **Condition Field {cond}**
<Oprnd2>	Refer to Table **Operand 2**
<fields>	Refer to Table **PSR fields**
{S}	Updates condition flags if S present
C*, V*	Flag is unpredictable
x,y	B meaning half-register [15:0], or T meaning [31:16]
<immed_8r>	A 32-bit constant, formed by right-rotating an 8-bit value by an even number of bits
<immed_8*4>	A 10-bit constant, formed by left-shifting an 8-bit value by two bits
<a_mode2>	Refer to Table **Addressing Mode 2**
<a_mode2P>	Refer to Table **Addressing Mode 2 (Post-indexed only)**
<a_mode3>	Refer to Table **Addressing Mode 3**
<a_mode4L>	Refer to Table **Addressing Mode 4 (Block load or Stack pop)**
<a_mode4S>	Refer to Table **Addressing Mode 4 (Block store or Stack push)**
<reglist>	A comma-separated list of registers, enclosed in braces ({ and })
{!}	Updates base register after data transfer if ! present

Condition Field {cond}	
Mnemonic	**Description**
EQ	Equal
NE	Not Equal
CS/HS	Carry Set / Unsigned higher or same
CC/LO	Carry Clear / Unsigned lower
MI	Negative
PL	Positive or zero
VS	Overflow
VC	No overflow
HI	Unsigned higher
LS	Unsigned lower or same
GE	Signed greater than or equal
LT	Signed less than
GT	Signed greater than
LE	Signed less than or equal
AL	Always (normally omitted)

Operation		Assembler	S updates	Action
Move	Move	MOV{cond}{S} Rd, <Oprnd2>	N Z C	Rd := Oprnd2
	NOT	MVN{cond}{S} Rd, <Oprnd2>	N Z C	Rd := 0xFFFFFFFF EOR Oprnd2
	SPSR to register	MRS{cond} Rd, SPSR		Rd := SPSR
	CPSR to register	MRS{cond} Rd, CPSR		Rd := CPSR
	register to SPSR	MSR{cond} SPSR_<fields>, Rm		SPSR := Rm (selected bytes only)
	register to CPSR	MSR{cond} CPSR_<fields>, Rm		CPSR := Rm (selected bytes only)
	immediate to SPSR	MSR{cond} SPSR_<fields>, #<immed_8r>		SPSR := immed_8r (selected bytes only)
	immediate to CPSR	MSR{cond} CPSR_<fields>, #<immed_8r>		CPSR := immed_8r (selected bytes only)
Arithmetic	Add	ADD{cond}{S} Rd, Rn, <Oprnd2>	N Z C V	Rd := Rn + Oprnd2
	with carry	ADC{cond}{S} Rd, Rn, <Oprnd2>	N Z C V	Rd := Rn + Oprnd2 + Carry
	Subtract	SUB{cond}{S} Rd, Rn, <Oprnd2>	N Z C V	Rd := Rn - Oprnd2
	with carry	SBC{cond}{S} Rd, Rn, <Oprnd2>	N Z C V	Rd := Rn - Oprnd2 - NOT(Carry)
	reverse subtract	RSB{cond}{S} Rd, Rn, <Oprnd2>	N Z C V	Rd := Oprnd2 - Rn
	reverse subtract with carry	RSC{cond}{S} Rd, Rn, <Oprnd2>	N Z C V	Rd := Oprnd2 - Rn - NOT(Carry)
	Multiply	MUL{cond}{S} Rd, Rm, Rs	N Z C*	Rd := (Rm * Rs)[31:0]
	accumulate	MLA{cond}{S} Rd, Rm, Rs, Rn	N Z C*	Rd := ((Rm * Rs) + Rn)[31:0]
	unsigned long	UMULL{cond}{S} RdLo, RdHi, Rm, Rs	N Z C* V*	RdHi,RdLo := unsigned(Rm * Rs)
	unsigned accumulate long	UMLAL{cond}{S} RdLo, RdHi, Rm, Rs	N Z C* V*	RdHi,RdLo := unsigned(RdHi,RdLo + Rm *
	signed long	SMULL{cond}{S} RdLo, RdHi, Rm, Rs	N Z C* V*	RdHi,RdLo := signed(Rm * Rs)
	signed accumulate long	SMLAL{cond}{S} RdLo, RdHi, Rm, Rs	N Z C* V*	RdHi,RdLo := signed(RdHi,RdLo + Rm * Rs)
Logical	Test	TST{cond} Rn, <Oprnd2>	N Z C	Update CPSR flags on Rn AND Oprnd2
	Test equivalence	TEQ{cond} Rn, <Oprnd2>	N Z C	Update CPSR flags on Rn EOR Oprnd2
	AND	AND{cond}{S} Rd, Rn, <Oprnd2>	N Z C	Rd := Rn AND Oprnd2
	EOR	EOR{cond}{S} Rd, Rn, <Oprnd2>	N Z C	Rd := Rn EOR Oprnd2
	ORR	ORR{cond}{S} Rd, Rn, <Oprnd2>	N Z C	Rd := Rn OR Oprnd2
	Bit Clear	BIC{cond}{S} Rd, Rn, <Oprnd2>	N Z C	Rd := Rn AND NOT Oprnd2
	No operation	NOP		R0 := R0
	Shift/Rotate			See table operand 2
Compare	Compare	CMP{cond} Rn, <Oprnd2>	N Z C V	Update CPSR flags on Rn - Oprnd2
	negative	CMN{cond} Rn, <Oprnd2>	N Z C V	Update CPSR flags on Rn + Oprnd2

Operation		Assembler	Action	Notes
Branch	Branch	B{cond} label	R15 := label	label must be within ±32Mb of current instruction.
	with link	BL{cond} label	R14 := R15-4, R15 := label	label must be within ±32Mb of current instruction.
	and exchange	BX{cond} Rm	R15 := Rm, Change to Thumb if Rm[0] is 1	
Load	Word	LDR{cond} Rd, <a_mode2>	Rd := [address]	
	User mode privilege	LDR{cond}T Rd, <a_mode2P>		
	branch (and exchange)	LDR{cond} R15, <a_mode2>	R15 := [address][31:1]	
	Byte	LDR{cond}B Rd, <a_mode2>	Rd := ZeroExtend[byte from address]	
	User mode privilege	LDR{cond}BT Rd, <a_mode2P>		
	Signed	LDR{cond}SB Rd, <a_mode3>	Rd := SignExtend[byte from address]	
	Halfword	LDR{cond}H Rd, <a_mode3>	Rd := ZeroExtent[halfword from address]	
	Signed	LDR{cond}SH Rd, <a_mode3>	Rd := SignExtend[halfword from address]	
Load multiple	Pop, or Block data load	LDM{cond}<a_mode4L> Rd{!}, <reglist-pc>	Load list of registers from [Rd]	
	return (and exchange)	LDM{cond}<a_mode4L> Rd{!}, <reglist+pc>	Load registers, R15 := [address][31:1]	
	and restore CPSR	LDM{cond}<a_mode4L> Rd{!}, <reglist+pc>^	Load registers, branch, CPSR := SPSR	Use from exception modes only.
	User mode registers	LDM{cond}<a_mode4L> Rd, <reglist-pc>^	Load list of User mode registers from [Rd]	Use from privileged modes only.
Store	Word	STR{cond} Rd, <a_mode2>	[address] := Rd	
	User mode privilege	STR{cond}T Rd, <a_mode2P>	[address] := Rd	
	Byte	STR{cond}B Rd, <a_mode2>	[address][7:0] := Rd[7:0]	
	User mode privilege	STR{cond}BT Rd, <a_mode2P>	[address][7:0] := Rd[7:0]	
	Halfword	STR{cond}H Rd, <a_mode3>	[address][15:0] := Rd[15:0]	
Store multiple	Push, or Block data store	STM{cond}<a_mode4S> Rd{!}, <reglist>	Store list of registers to [Rd]	
	User mode registers	STM{cond}<a_mode4S> Rd{!}, <reglist>^	Store list of User mode registers to [Rd]	Use from privileged modes only.
Swap	Word	SWP{cond} Rd, Rm, [Rn]	temp := [Rn], [Rn] := Rm, Rd := temp	
	Byte	SWP{cond}B Rd, Rm, [Rn]	temp := ZeroExtend([Rn][7:0]), [Rn][7:0] := Rm[7:0], Rd := temp	
Software interrupt		SWI{cond} <immed_24>	Software interrupt processor exception	24-bit value encoded in instruction.

Addressing Mode 2 – Word and Unsigned Byte Data Transfer			
Pre-indexed	Immediate offset	[Rn, #+/-<immed_12>]{!}	
	Zero offset	[Rn]	Equivalent to [Rn,#0]
	Register offset	[Rn, +/-Rm]{!}	
	Scaled register offset	[Rn, +/-Rm, LSL #<immed_5>]{!}	Allowed shifts 0-31
		[Rn, +/-Rm, LSR #<immed_5>]{!}	Allowed shifts 1-32
		[Rn, +/-Rm, ASR #<immed_5>]{!}	Allowed shifts 1-32
		[Rn, +/-Rm, ROR #<immed_5>]{!}	Allowed shifts 1-31
		[Rn, +/-Rm, RRX]{!}	
Post-indexed	Immediate offset	[Rn], #+/-<immed_12>	
	Register offset	[Rn], +/-Rm	
	Scaled register offset	[Rn], +/-Rm, LSL #<immed_5>	Allowed shifts 0-31
		[Rn], +/-Rm, LSR #<immed_5>	Allowed shifts 1-32
		[Rn], +/-Rm, ASR #<immed_5>	Allowed shifts 1-32
		[Rn], +/-Rm, ROR #<immed_5>	Allowed shifts 1-31
		[Rn], +/-Rm, RRX	

Addressing Mode 2 (Post-indexed only)			
Post-indexed	Immediate offset	[Rn], #+/-<immed_12>	
	Zero offset	[Rn]	Equivalent to [Rn,#0]
	Register offset	[Rn], +/-Rm	
	Scaled register offset	[Rn], +/-Rm, LSL #<immed_5>	Allowed shifts 0-31
		[Rn], +/-Rm, LSR #<immed_5>	Allowed shifts 1-32
		[Rn], +/-Rm, ASR #<immed_5>	Allowed shifts 1-32
		[Rn], +/-Rm, ROR #<immed_5>	Allowed shifts 1-31
		[Rn], +/-Rm, RRX	

Addressing Mode 3 – Halfword, Signed Byte, and Doubleword Data Transfer			
Pre-indexed	Immediate offset	[Rn, #+/-<immed_8>]{!}	
	Zero offset	[Rn]	Equivalent to [Rn,#0]
	Register	[Rn, +/-Rm]{!}	
Pre-indexed	Immediate offset	[Rn], #+/-<immed_8>	
	Register	[Rn], +/-Rm	

Addressing Mode 4 - Multiple Data Transfer	
Block load	**Stack pop**
IA Increment After	FD Full Descending
IB Increment Before	ED Empty Descending
DA Decrement After	FA Full Ascending
DB Decrement Before	EA Empty Ascending
Block store	**Stack push**
IA Increment After	EA Empty Ascending
IB Increment Before	FA Full Ascending
DA Decrement After	ED Empty Descending
DB Decrement Before	FD Full Descending

PSR fields (use at least one suffix)		
Suffix	**Meaning**	
c	Control field mask byte	PSR[7:0]
f	Flags field mask byte	PSR[31:24]
s	Status field mask byte	PSR[23:16]
x	Extension field mask byte	PSR[15:8]

Operand 2		
Immediate value	#<immed_8r>	
Logical shift left immediate	Rm, LSL #<immed_5>	Allowed shifts 0-31
Logical shift right immediate	Rm, LSR #<immed_5>	Allowed shifts 1-32
Arithmetic shift right immediate	Rm, ASR #<immed_5>	Allowed shifts 1-32
Rotate right immediate	Rm, ROR #<immed_5>	Allowed shifts 1-31
Register	Rm	
Rotate right extended	Rm, RRX	
Logical shift left register	Rm, LSL Rs	
Logical shift right register	Rm, LSR Rs	
Arithmetic shift right register	Rm, ASR Rs	
Rotate right register	Rm, ROR Rs	

All Thumb registers are Lo (R0-R7) except where specified. Hi registers are R8-R15.

Operation		**Assembler**	**Flag updates**	**Action**	**Notes**
Move	Immediate	MOV Rd, #<immed_8>	yes	Rd := immed_8	8-bit immediate value.
	Lo to Lo	MOV Rd, Rm	Yes	Rd := Rm	
	Hi to Lo, Lo to Hi, Hi to Hi	MOV Rd, Rm	No	Rd := Rm	Not Lo to Lo
Arithmetic	Add	ADD Rd, Rn, #<immed_3>	Yes	Rd := Rn + immed_3	3-bit immediate value.
	Lo and Lo	ADD Rd, Rn, Rm	Yes	Rd := Rn + Rm	
	Hi to Lo, Lo to Hi, Hi to Hi	ADD Rd, Rm	Yes	Rd := Rd + Rm	Not Lo to Lo
	immediate	ADD Rd, #<immed_8>	Yes	Rd := Rd + immed_8	8-bit immediate value.
	with carry	ADC Rd, Rm	Yes	Rd := Rd + Rm + C-bit	
	value to SP	ADD SP, #<immed_7*4>	Yes	SP := SP + immed_7 * 4	9-bit immediate value (word-aligned).
	form address from SP	ADD Rd, SP, #<immed_8*4>	No	Rd := SP + immed_8 * 4	10-bit immediate value (word-aligned).
	form address from PC	ADD Rd, PC, #<immed_8*4>	No	Rd := (PC AND 0xFFFFFFFC) + immed_8 * 4	10-bit immediate value (word-aligned).
	Subtract	SUB Rd, Rn, Rm	Yes	Rd := Rn - Rm	
	immediate 3	SUB Rd, Rn, #<immed_3>	Yes	Rd := Rn - immed_3	3-bit immediate value.
	immediate 8	SUB Rd, #<immed_8>	Yes	Rd := Rd - immed_8	8-bit immediate value.
	with carry	SBC Rd, Rm	Yes	Rd := Rd - Rm - NOT C-bit	
	value from SP	SUB SP, #<immed_7*4>	No	SP := SP - immed_7 * 4	9-bit immediate value (word-aligned).
	Negate	NEG Rd, Rm	Yes	Rd := - Rm	
	Multiply	MUL Rd, Rm	Yes	Rd := Rm * Rd	
	Compare	CMP Rn, Rm	Yes	update CPSR flags on Rn - Rm	Can be Lo to Lo, Lo to Hi, Hi to Lo, or Hi to Hi.
	negative	CMN Rn, Rm	Yes	update CPSR flags on Rn + Rm	
	immediate	CMP Rn, #<immed_8>	Yes	update CPSR flags on Rn - immed_8	8-bit immediate value.
	No operation	NOP	No	R8 := R8	Flags not affected.

Operation		Assembler	Flag updates	Action	Notes
Logical	AND	AND Rd, Rm	Yes	Rd := Rd AND Rm	
	Exclusive OR	EOR Rd, Rm	Yes	Rd := Rd EOR Rm	
	OR	ORR Rd, Rm	Yes	Rd := Rd OR Rm	
	Bit clear	BIC Rd, Rm	Yes	Rd := Rd AND NOT Rm	
	Move NOT	MVN Rd, Rm	Yes	Rd := NOT Rm	
	Test bits	TST Rn, Rm	Yes	update CPSR flags on Rn AND Rm	
Shift/rotate	Logical shift left	LSL Rd, Rm, #<immed_5>	Yes	Rd := Rm << immed_5	5-bit immediate shift. Allowed shifts 0-31.
		LSL Rd, Rs	Yes	Rd := Rd << Rs	
	Logical shift right	LSR Rd, Rm, #<immed_5>	Yes	Rd := Rm >> immed_5	5-bit immediate shift. Allowed shifts 1-32.
		LSR Rd, Rs	Yes	Rd := Rd >> Rs	
	Arithmetic shift right	ASR Rd, Rm, #<immed_5>	Yes	Rd := Rm ASR immed_5	5-bit immediate shift. Allowed shifts 1-32.
		ASR Rd, Rs	Yes	Rd := Rd ASR Rs	
	Rotate right	ROR Rd, Rs	Yes	Rd := Rd ROR Rs	
Branch	Conditional branch	B{cond} label		R15 := label	label must be within -252 to +258 bytes of current instruction. See Table Condition Field (ARM side). AL not allowed.
	Unconditional branch	B label		R15 := label	label must be within ±2Kb of current instruction.
	Long branch with link	BL label		R14 := R15 - 2, R15 := label	Encoded as two Thumb instructions. label must be within ±4Mb of current instruction.
	Branch and exchange	BX Rm		R15 := Rm AND 0xFFFFFFFE	Change to ARM state if Rm[0] = 0.
Software Interrupt		SWI <immed_8>		Software interrupt processor exception	8-bit immediate value encoded in instruction.
Load	with immediate offset, word	LDR Rd, [Rn, #<immed_5*4>]		Rd := [Rn + immed_5 * 4]	
	halfword	LDRH Rd, [Rn, #<immed_5*2>]		Rd := ZeroExtend([Rn + immed_5 * 2][15:0])	Clears bits 31:16
	byte	LDRB Rd, [Rn, #<immed_5>]		Rd := ZeroExtend([Rn + immed_5][7:0])	Clears bits 31:8
	with register offset, word	LDR Rd, [Rn, Rm]		Rd := [Rn + Rm]	
	halfword	LDRH Rd, [Rn, Rm]		Rd := ZeroExtend([Rn + Rm][15:0])	Clears bits 31:16
	signed halfword	LDRSH Rd, [Rn, Rm]		Rd := SignExtend([Rn + Rm][15:0])	Sets bits 31:16 to bit 15
	byte	LDRB Rd, [Rn, Rm]		Rd := ZeroExtend([Rn + Rm][7:0])	Clears bits 31:8
	signed byte	LDRSB Rd, [Rn, Rm]		Rd := SignExtend([Rn + Rm][7:0])	Sets bits 31:8 to bit 7
	PC-relative	LDR Rd, [PC, #<immed_8*4>]		Rd := [(PC AND 0xFFFFFFFC) + immed_8 * 4]	
	SP-relative	LDR Rd, [SP, #<immed_8*4>]		Rd := [SP + immed_8 * 4]	
	Multiple	LDMIA Rn!, <reglist>		Loads list of registers	Always updates base register.
Store	with immediate offset, word	STR Rd, [Rn, #<immed_5*4>]		[Rn + immed_5 * 4] := Rd	
	halfword	STRH Rd, [Rn, #<immed_5*2>]		[Rn + immed_5 * 2][15:0] := Rd[15:0]	Ignores Rd[31:16]
	byte	STRB Rd, [Rn, #<immed_5>]		[Rn + immed_5][7:0] := Rd[7:0]	Ignores Rd[31:8]
	with register offset, word	STR Rd, [Rn, Rm]		[Rn + Rm] := Rd	
	halfword	STRH Rd, [Rn, Rm]		[Rn + Rm][15:0] := Rd[15:0]	Ignores Rd[31:16]
	byte	STRB Rd, [Rn, Rm]		[Rn + Rm][7:0] := Rd[7:0]	Ignores Rd[31:8]
	SP-relative, word	STR Rd, [SP, #<immed_8*4>]		[SP + immed_8 * 4] := Rd	
	Multiple	STMIA Rn!, <reglist>		Stores list of registers	Always updates base register.
Push/Pop	Push	PUSH <reglist>		Push registers onto stack	Full descending stack.
	Push with link	PUSH <reglist, LR>		Push LR and registers onto stack	
	Pop	POP <reglist>		Pop registers from stack	
	Pop and return	POP <reglist, PC>		Pop registers, branch to address loaded to PC	