ARMv4T Partial Instruction Set Summary

Operation		Syntax	Page Num.	S updates				Action and Comments
Move	Move	**mov**{*cond*}{**s**} R*d*, *shift_op*	A4-56 (156)	N	Z	C		R*d* := *shift_op*
	with NOT	**mvn**{*cond*}{**s**} R*d*, *shift_op*	A4-68 (168)	N	Z	C		R*d* := NOT(*shift_op*)
	CPSR to register	**mrs**{*cond*} R*d*, **cpsr**	A4-60 (160)					R*d* := CPSR
	SPSR to register	**mrs**{*cond*} R*d*, **spsr**	A4-60 (160)					R*d* := SPSR; Not valid in System or User modes
	register to CPSR	**msr**{*cond*} **cpsr_***fields*, R*m*	A4-62 (162)					CPSR := R*d* (selected bytes only)
	register to SPSR	**msr**{*cond*} **spsr_***fields*, R*m*	A4-62 (162)					SPSR := R*d* (selected bytes only); Not valid in System or User modes
	immediate to CPSR	**msr**{*cond*} **cpsr_***fields*, #*imm8r*	A4-62 (162)					CPSR := *imm8r* (selected bytes only)
	immediate to SPSR	**msr**{*cond*} **spsr_***fields*, #*imm8r*	A4-62 (162)					SPSR := *imm8r* (selected bytes only); Not valid in System or User modes
Arithmetic	Add	**add**{*cond*}{**s**} R*d*, R*n*, *shift_op*	A4-6 (106)	N	Z	C	V	R*d* := R*n* + *shift_op*
	with carry	**adc**{*cond*}{**s**} R*d*, R*n*, *shift_op*	A4-4 (104)	N	Z	C	V	R*d* := R*n* + *shift_op* + Carry
	Subtract	**sub**{*cond*}{**s**} R*d*, R*n*, *shift_op*	A4-98 (198)	N	Z	C	V	R*d* := R*n* − *shift_op*
	with carry	**sbc**{*cond*}{**s**} R*d*, R*n*, *shift_op*	A4-76 (176)	N	Z	C	V	R*d* := R*n* − *shift_op* − NOT(Carry)
	reverse subtract	**rsb**{*cond*}{**s**} R*d*, R*n*, *shift_op*	A4-72 (172)	N	Z	C	V	R*d* := *shift_op* − R*n*
	reverse subtract with carry	**rsc**{*cond*}{**s**} R*d*, R*n*, *shift_op*	A4-74 (174)	N	Z	C	V	R*d* := *shift_op* − R*n* − NOT(Carry)
	Multiply	**mul**{*cond*}{**s**} R*d*, R*m*, R*s*	A4-66 (166)	N	Z	C?		R*d* := R*m* × R*s* (lower 32 bits only)
	with accumulate	**mla**{*cond*}{**s**} R*d*, R*m*, R*s*, R*n*	A4-54 (154)	N	Z	C?		R*d* := (R*m* × R*s*) + R*n* (lower 32 bits only)
	unsigned long	**umull**{*cond*}{**s**} R*dLo*, R*dHi*, R*m*, R*s*	A4-111 (211)	N	Z	C?	V?	R*dHi*,R*dLo* := unsigned(R*m* × R*s*)
	unsigned long with accumulate	**umlal**{*cond*}{**s**} R*dLo*, R*dHi*, R*m*, R*s*	A4-109 (209)	N	Z	C?	V?	R*dHi*,R*dLo* := unsigned(R*dHi*,R*dLo* + R*m* × R*s*)
	signed long	**smull**{*cond*}{**s**} R*dLo*, R*dHi*, R*m*, R*s*	A4-80 (180)	N	Z	C?	V?	R*dHi*,R*dLo* := signed(R*m* × R*s*)
	signed long with accumulate	**smlal**{*cond*}{**s**} R*dLo*, R*dHi*, R*m*, R*s*	A4-78 (178)	N	Z	C?	V?	R*dHi*,R*dLo* := signed(R*dHi*,R*dLo* + R*m* × R*s*)
Logical	AND	**and**{*cond*}{**s**} R*d*, R*n*, *shift_op*	A4-8 (108)	N	Z	C		R*d* := R*n* AND *shift_op*
	OR	**orr**{*cond*}{**s**} R*d*, R*n*, *shift_op*	A4-70 (170)	N	Z	C		R*d* := R*n* OR *shift_op*
	XOR	**eor**{*cond*}{**s**} R*d*, R*n*, *shift_op*	A4-26 (126)	N	Z	C		R*d* := R*n* XOR *shift_op*
	Bit clear	**bic**{*cond*}{**s**} R*d*, R*n*, *shift_op*	A4-12 (112)	N	Z	C		R*d* := R*n* AND NOT(*shift_op*)
	Rotates and shifts	Usually **mov**{*cond*}{**s**} R*d*, *shift_op*	A4-56 (156)	N	Z	C		Usually R*d* := *shift_op*; May use other instructions (eg, **add**, **sub**)
Compare	Compare	**cmp**{*cond*} R*n*, *shift_op*	A4-25 (125)	N	Z	C	V	Update CPSR flags for R*n* − *shift_op*
	negated	**cmn**{*cond*} R*n*, *shift_op*	A4-23 (123)	N	Z	C	V	Update CPSR flags for R*n* + *shift_op*
	Bitwise test	**tst**{*cond*} R*n*, *shift_op*	A4-107 (207)	N	Z	C		Update CPSR flags for R*n* AND *shift_op*
	Equivalence test	**teq**{*cond*} R*n*, *shift_op*	A4-106 (206)	N	Z	C		Update CPSR flags for R*n* XOR *shift_op*
Branch	Branch	**b**{*cond*} *label*	A4-10 (110)					R15 := address of *label*; Jump to *label*
	with link	**bl**{*cond*} *label*	A4-10 (110)					R14 := R15 − 4; R15 := address of label; Call subroutine/function at *label*
	and exchange	**bx**{*cond*} R*m*	A4-19 (119)					R15 := R*m*; Changes to Thumb mode if bit 0 of R*m* is 1
	Return from subroutine	Usually **mov pc, lr** or **bx lr**	A4-11 (111)					R15 := R14
Load	Load word	**ldr**{*cond*} R*d*, *am2*	A4-37 (137)					R*d* := word stored at address *am2*
	using User mode privileges	**ldr**{*cond*}**t** R*d*, *am2P*	A4-50 (150)					R*d* := word stored at address *am2P*; Usually used in non-User modes
	Load byte	**ldr**{*cond*}**b** R*d*, *am2*	A4-40 (140)					R*d* := ZeroExtend(byte at address *am2*)
	signed	**ldr**{*cond*}**sb** R*d*, *am3*	A4-46 (146)					R*d* := SignExtend(byte at address *am3*)
	using User mode privileges	**ldr**{*cond*}**bt** R*d*, *am2P*	A4-42 (142)					R*d* := ZeroExtend(byte at address *am2P*); Usually used in non-User modes
	Load half-word	**ldr**{*cond*}**h** R*d*, *am3*	A4-44 (144)					R*d* := ZeroExtend(half-word at address *am3*)
	signed	**ldr**{*cond*}**sh** R*d*, *am3*	A4-48 (148)					R*d* := SignExtend(half-word at address *am3*)
Store	Store word	**str**{*cond*} R*d*, *am2*	A4-88 (188)					Store word in R*d* at address *am2*
	using User mode privileges	**str**{*cond*}**t** R*d*, *am2P*	A4-96 (196)					Store word in R*d* at address *am2P*; Usually used in non-User modes
	Store byte	**str**{*cond*}**b** R*d*, *am2*	A4-90 (190)					Store byte in R*d*[7..0] at address *am2*
	using User mode privileges	**str**{*cond*}**bt** R*d*, *am2P*	A4-92 (192)					Store byte in R*d*[7..0] at address *am2P*; Usually used in non-User modes
	Store half-word	**str**{*cond*}**h** R*d*, *am3*	A4-94 (194)					Store half-word in R*d*[15..0] at address *am3*
Load Multiple	Pop, or Block data load	**ldm**{*cond*}{*am4L*} R*d*{**!**}, **{***reglist***}**	A4-30 (130)					Load all listed registers from address in R*d*
	using User mode privileges	**ldm**{*cond*}{*am4L*} R*d*{**!**}, **{***reglist-pc***}^**	A4-32 (132)					Load all listed registers (PC register not listed) from address in R*d*; Not valid in User mode
	return and restore CPSR	**ldm**{*cond*}{*am4L*} R*d*{**!**}, **{***reglist+pc***}^**	A4-34 (134)					Load registers, CPSR := SPSR, branch to PC; Not valid in User or System modes
Store Multiple	Push, or Block data store	**stm**{*cond*}{*am4S*} R*d*{**!**}, **{***reglist***}**	A4-84 (184)					Store all listed registers to address in R*d*
	using User mode privileges	**stm**{*cond*}{*am4S*} R*d*{**!**}, **{***reglist***}^**	A4-86 (186)					Store all listed registers to address in R*d*; Usually used in non-User modes
Miscellaneous	Swap word	**swp**{*cond*} R*d*, R*m*, **[**R*n***]**	A4-102 (202)					temp := word at address in R*n*, store R*m* to address in R*n*, R*d* := temp
	Swap byte	**swp**{*cond*}**b** R*d*, R*m*, **[**R*n***]**	A4-104 (204)					temp := ZeroExtend(byte at address in R*n*), store R*m*[7..0] to address in R*n*, R*d* := temp
	Software interrupt	**swi**{*cond*} *imm24*	A4-100 (200)					Branch-and-link to address 0x00000008 in Supervisor mode
Pseudo-instructions	No operation	**nop**{*cond*}						Does nothing; translates to **mov**{*cond*} **r0, r0**
	Load variable	**ldr**{*cond*} R*d*, *label*						R*d* := word stored at address *label*; translates to **ldr**{*cond*} R*d*, **[r15, #***offset***]**
	Move constant	**ldr**{*cond*} R*d*, =*imm32*						**mov** R*d*, #*imm32* or **ldr** R*d*, **[r15, #***offset***]**
	Load address	**adr**{*cond*} R*d*, *label*						**add** R*d*, **r15**, #*offset*
	long version	**adr**{*cond*}**l** R*d*, *label*						Two-instruction form of **adr**

ARMv4T Partial Instruction Set Tables

Register Names and Aliases

Reg	Aliases	Purpose in ARM Thumb Procedure Call Standard
`r0`	`a1`	Argument/result/scratch register 1
`r1`	`a2`	Argument/result/scratch register 2
`r2`	`a3`	Argument/result/scratch register 3
`r3`	`a4`	Argument/result/scratch register 4
`r4`	`v1`	Variable register 1
`r5`	`v2`	Variable register 2
`r6`	`v3`	Variable register 3
`r7`	`v4`	Variable register 4
`r8`	`v5`	Variable register 5
`r9`	`v6` or `sb`	Variable register 6; sometimes Stack Base register
`r10`	`v7` or `sl`	Variable register 7; sometimes Stack Limit register
`r11`	`v8` or `fp`	Variable register 8; usually Frame Pointer register
`r12`	`ip`	Intra-procedure-call scratch register
`r13`	`sp`	Stack Pointer
`r14`	`lr`	Link Register
`r15`	`pc`	Program Counter

Program Status Register Format

31			28	24	16	8	7	6	5	4 0
N	Z	C	V	*Undef.*	*Undef.*	*Undef.*	I	F	T	Mode

Program Status Register Modes

Value	Mode	Accessible registers
`0b10000`	User	PC, R14–R0, CPSR
`0b10001`	Fast Interrupt	PC, R14_fiq–R8_fiq, R7–R0, CPSR, SPSR_fiq
`0b10010`	Interrupt	PC, R14_irq–R13_irq, R12–R0, CPSR, SPSR_irq
`0b10011`	Supervisor	PC, R14_svc–R13_svc, R12–R0, CPSR, SPSR_svc
`0b10111`	Abort	PC, R14_abt–R13_abt, R12–R0, CPSR, SPSR_abt
`0b11011`	Undefined	PC, R14_und–R13_und, R12–R0, CPSR, SPSR_und
`0b11111`	System	PC, R14–R0, CPSR

Program Status Register Fields: *fields*

Suffix	Bits in PSRs	Description
`c`	0–7	Control field mask
`x`	8–15	Extension field mask (no bits currently defined)
`s`	16–23	Status field mask (no bits currently defined)
`f`	24–31	Flags field mask (bits 24–27 undefined)

Optional Condition Field: *cond*

Mnemonic	Description	Condition flags state
`EQ`	Equal	Z set
`NE`	Not equal	Z clear
`CS` or `HS`	Carry set/unsigned higher or same	C set
`CC` or `LO`	Carry clear/unsigned lower	C clear
`MI`	Minus/negative	N set
`PL`	Plus/positive or zero	N clear
`VS`	Overflow	V set
`VC`	No overflow	V clear
`HI`	Unsigned higher	C set and Z clear
`LS`	Unsigned lower or same	C clear or Z set
`GE`	Signed greater than or equal	N equal to V
`LT`	Signed less than	N not equal to V
`GT`	Signed greater than	Z clear and N equal to V
`LE`	Signed less than or equal	Z set and N not equal to V
`AL`	Always (unconditional, default)	Irrelevant

Data Processing Mode: *shifter_op*

Operation	Syntax	Comments
Immediate value	#*imm8r*	
Register	R*m*	
Logical shift left immediate	R*m*, `lsl` #*imm5*	Allowed 0–31 only
Logical shift left by register	R*m*, `lsl` R*s*	
Logical shift right immediate	R*m*, `lsr` #*imm5*	Allowed 1–32 only
Logical shift right by register	R*m*, `lsr` R*s*	
Arithmetic shift right immediate	R*m*, `asr` #*imm5*	Allowed 1–32 only
Arithmetic shift right by register	R*m*, `asr` R*s*	
Rotate right immediate	R*m*, `ror` #*imm5*	Allowed 1–31 only
Rotate right by register	R*m*, `ror` R*s*	
Rotate right with extend	R*m*, `rrx`	

Load or Store Word/Unsigned Byte Mode: *am2*

Operation		Syntax	Cmts.
Pre-indexed	Immediate offset	[R*n*, #±*imm12*]{!}	
	Zero offset	[R*n*]	[R*n*, #0]
	Register offset	[R*n*, ±R*m*]{!}	
	Scaled register offset	[R*n*, ±R*m*, `lsl` #*imm5*]{!}	0–31 only
		[R*n*, ±R*m*, `lsr` #*imm5*]{!}	1–32 only
		[R*n*, ±R*m*, `asr` #*imm5*]{!}	1–32 only
		[R*n*, ±R*m*, `ror` #*imm5*]{!}	1–31 only
		[R*n*, ±R*m*, `rrx`]{!}	
Post-indexed	Immediate offset	[R*n*], #±*imm12*	
	Register offset	[R*n*], ±R*m*	
	Scaled register offset	[R*n*], ±R*m*, `lsl` #*imm5*	0–31 only
		[R*n*], ±R*m*, `lsr` #*imm5*	1–32 only
		[R*n*], ±R*m*, `asr` #*imm5*	1–32 only
		[R*n*], ±R*m*, `ror` #*imm5*	1–31 only
		[R*n*], ±R*m*, `rrx`	

Load or Store with Translation Mode: *am2P*

Operation		Syntax	Cmts.
Post-indexed	Immediate offset	[R*n*], #±*imm12*	
	Register offset	[R*n*], ±R*m*	
	Scaled register offset	[R*n*], ±R*m*, `lsl` #*imm5*	0–31 only
		[R*n*], ±R*m*, `lsr` #*imm5*	1–32 only
		[R*n*], ±R*m*, `asr` #*imm5*	1–32 only
		[R*n*], ±R*m*, `ror` #*imm5*	1–31 only
		[R*n*], ±R*m*, `rrx`	

Load or Store Half-Word/Signed Byte Mode: *am3*

Operation		Syntax	Comments
Pre-indexed	Immediate offset	[R*n*, #±*imm8*]{!}	Note: not *imm8r*
	Zero offset	[R*n*]	Same as [R*n*, #0]
	Register offset	[R*n*, ±R*m*]{!}	
Post-indexed	Immediate offset	[R*n*], #±*imm8*	Note: not *imm8r*
	Register offset	[R*n*], ±R*m*	

Load Multiple Data Mode: *am4L*

Suffix	Non-stack Addressing Mode	Suffix	Stack Addressing Mode
`ia`	Increment after	`fd`	Full descending
`ib`	Increment before	`ed`	Empty descending
`da`	Decrement after	`fa`	Full ascending
`db`	Decrement before	`ea`	Empty ascending

Store Multiple Data Mode: *am4S*

Suffix	Non-stack Addressing Mode	Suffix	Stack Addressing Mode
`ia`	Increment after	`ea`	Empty ascending
`ib`	Increment before	`fa`	Full ascending
`da`	Decrement after	`ed`	Empty descending
`db`	Decrement before	`fd`	Full descending

Exception Vector Table

Address	Mode	Exception Type
`0x00000000`	Supervisor	Reset
`0x00000004`	Undefined	Undefined instruction
`0x00000008`	Supervisor	Software interrupt
`0x0000000C`	Abort	Prefetch abort (instruction fetch abort)
`0x00000010`	Abort	Data abort (data access memory abort)
`0x00000014`	(None)	(Not used)
`0x00000018`	Interrupt	Normal-priority interrupt
`0x0000001C`	Fast Interrupt	High-priority (fast) interrupt

Miscellaneous

Symbol	Meaning
imm5	Immediate 5-bit number, either 0–31, 1–32 or 1–31
imm8	Immediate 8-bit number, between 0–255
imm8r	A 32-bit number that can be formed by rotating an 8-bit number (0–255) by an even number between 0 and 30
imm12	Immediate 12-bit number, between 0–4095
imm24	Immediate 24-bit number, between 0–16,777,215
imm32	Immediate 32-bit number, between 0–4,294,967,295
{s}	If present, the instruction will update the condition flags
N	Negative flag: 1 if result is negative
Z	Zero flag: 1 if result is zero
C	Carry flag
V	Signed Overflow flag
C?	Carry flag ends in an unpredictable state, if flags are set
V?	Overflow flag ends in an unpredictable state, if flags are set
I	Interrupt Disable bit in the PSRs: 1 to disable interrupts
F	Fast Interrupt Disable bit: 1 to disable fast interrupts
T	ARM or Thumb state: 0 for ARM execution, 1 for Thumb
{*reglist*}	List of registers separated by commas or dashes, surrounded by braces, eg, `{r0,r1,r2}` or `{r0-r3,r5}`
{*reglist-pc*}	List of registers that does *not* include PC (R15)
{*reglist+pc*}	List of registers that *does* include PC (R15)
{!}	If present, the instruction updates the base register after the memory transfer. Post-indexed accesses *always* update the base register
±	Either + or − may be supplied; + is assumed if not present

This document contains a summary of the ARMv4T instruction set architecture in tabular format. It does not list every instruction available in the ARM architecture: the coprocessor instructions, in particular, have not been listed. Page numbers refer to both the printed and on-line versions of the *ARM Architecture Reference Manual*, Second Edition, published by Addison-Wesley in December 2000 (ISBN 0-201-73719-1).

This document was created by John Zaitseff for the Digital Systems Laboratory at the University of New South Wales.