Carte de référence rapide

Set d'instructions ARM

Key to Tables	
{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 after these instructions in Architecture v4 and earlier
Q	Sticky flag. Always updates on overflow (no S option). Read and reset using MRS and MSR
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)**
<a_mode5>	Refer to Table **Addressing Mode 5**
<reglist>	A comma-separated list of registers, enclosed in braces ({ and })
{!}	Updates base register after data transfer if ! present
¤	Refer to Table **ARM architecture versions**

Operation		¤	Assembler	S updates	Q	Action	Notes
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	3	MRS{cond} Rd, SPSR			Rd := SPSR	
	CPSR to register	3	MRS{cond} Rd, CPSR			Rd := CPSR	
	register to SPSR	3	MSR{cond} SPSR_<fields>, Rm			SPSR := Rm (selected bytes only)	
	register to CPSR	3	MSR{cond} CPSR_<fields>, Rm			CPSR := Rm (selected bytes only)	
	immediate to SPSR	3	MSR{cond} SPSR_<fields>, #<immed_8r>			SPSR := immed_8r (selected bytes only)	
	immediate to CPSR	3	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	
	saturating	5E	QADD{cond} Rd, Rm, Rn		Q	Rd := SAT(Rm + Rn)	No shift/rotate.
	double saturating	5E	QDADD{cond} Rd, Rm, Rn		Q	Rd := SAT(Rm + SAT(Rn * 2))	No shift/rotate.
	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)	
	saturating	5E	QSUB{cond} Rd, Rm, Rn		Q	Rd := SAT(Rm - Rn)	No shift/rotate.
	double saturating	5E	QDSUB{cond} Rd, Rm, Rn		Q	Rd := SAT(Rm - SAT(Rn * 2))	No shift/rotate.
	Multiply	2	MUL{cond}{S} Rd, Rm, Rs	N Z C*		Rd := (Rm * Rs)[31:0]	
	accumulate	2	MLA{cond}{S} Rd, Rm, Rs, Rn	N Z C*		Rd := ((Rm * Rs) + Rn)[31:0]	
	unsigned long	M	UMULL{cond}{S} RdLo, RdHi, Rm, Rs	N Z C* V*		RdHi,RdLo := unsigned(Rm * Rs)	
	unsigned accumulate long	M	UMLAL{cond}{S} RdLo, RdHi, Rm, Rs	N Z C* V*		RdHi,RdLo := unsigned(RdHi,RdLo + Rm * Rs)	
	signed long	M	SMULL{cond}{S} RdLo, RdHi, Rm, Rs	N Z C* V*		RdHi,RdLo := signed(Rm * Rs)	
	signed accumulate long	M	SMLAL{cond}{S} RdLo, RdHi, Rm, Rs	N Z C* V*		RdHi,RdLo := signed(RdHi,RdLo + Rm * Rs)	
	signed 16 * 16 bit	5E	SMULxy{cond} Rd, Rm, Rs			Rd := Rm[x] * Rs[y]	No shift/rotate.
	signed 32 * 16 bit	5E	SMULWy{cond} Rd, Rm, Rs			Rd := (Rm * Rs[y])[47:16]	No shift/rotate.
	signed accumulate 16 * 16	5E	SMLAxy{cond} Rd, Rm, Rs, Rn		Q	Rd := Rn + Rm[x] * Rs[y]	No shift/rotate.
	signed accumulate 32 * 16	5E	SMLAWy{cond} Rd, Rm, Rs, Rn		Q	Rd := Rn + (Rm * Rs[y])[47:16]	No shift/rotate.
	signed accumulate long 16 * 16	5E	SMLALxy{cond} RdLo, RdHi, Rm, Rs			RdHi,RdLo := RdHi,RdLo + Rm[x] * Rs[y]	No shift/rotate.
	Count leading zeroes	5	CLZ{cond} Rd, Rm			Rd := number of leading zeroes in Rm	
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	Flags not affected.
	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	4T	`BX{cond} Rm`	R15 := Rm, Change to Thumb if Rm[0] is 1	
	with link and exchange (1)	5T	`BLX label`	R14 := R15 - 4, R15 := label, Change to Thumb	Cannot be conditional. label must be within ±32Mb of current instruction.
	with link and exchange (2)	5T	`BLX{cond} Rm`	R14 := R15 - 4, R15 := Rm[31:1] 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] (¤ 5T: Change to Thumb if [address][0] is 1)	
	Byte		`LDR{cond}B Rd, <a_mode2>`	Rd := ZeroExtend[byte from address]	
	User mode privilege		`LDR{cond}BT Rd, <a_mode2P>`		
	signed	4	`LDR{cond}SB Rd, <a_mode3>`	Rd := SignExtend[byte from address]	
	Halfword	4	`LDR{cond}H Rd, <a_mode3>`	Rd := ZeroExtent[halfword from address]	
	signed	4	`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] (¤ 5T: Change to Thumb if [address][0] is 1)	
	and restore CPSR		`LDM{cond}<a_mode4L> Rd{!}, <reglist+pc>^`	Load registers, branch (¤ 5T: and exchange), 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	4	`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	3	`SWP{cond} Rd, Rm, [Rn]`	temp := [Rn], [Rn] := Rm, Rd := temp	
	Byte	3	`SWP{cond}B Rd, Rm, [Rn]`	temp := ZeroExtend([Rn][7:0]), [Rn][7:0] := Rm[7:0], Rd := temp	
Coprocessors	Data operations	2	`CDP{cond} p<cpnum>, <op1>, CRd, CRn, CRm, <op2>`	Coprocessor defined	
		5	`CDP2 p<cpnum>, <op1>, CRd, CRn, CRm, <op2>`		Cannot be conditional.
	Move to ARM reg from coproc	2	`MRC{cond} p<cpnum>, <op1>, Rd, CRn, CRm, <op2>`		
		5	`MRC2 p<cpnum>, <op1>, Rd, CRn, CRm, <op2>`		Cannot be conditional.
	Move to coproc from ARM reg	2	`MCR{cond} p<cpnum>, <op1>, Rd, CRn, CRm, <op2>`		
		5	`MCR2 p<cpnum>, <op1>, Rd, CRn, CRm, <op2>`		Cannot be conditional.
	Load	2	`LDC{cond} p<cpnum>, CRd, <a_mode5>`		
		5	`LDC2 p<cpnum>, CRd, <a_mode5>`		Cannot be conditional.
	Store	2	`STC{cond} p<cpnum>, CRd, <a_mode5>`		
		5	`STC2 p<cpnum>, CRd, <a_mode5>`		Cannot be conditional.
Software interrupt			`SWI{cond} <immed_24>`	Software interrupt processor exception	24-bit value encoded in instruction.
Breakpoint		5	`BKPT <immed_16>`	Prefetch abort *or* enter debug state	Cannot be conditional.

Carte de référence rapide

Modes d'adressage ARM

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

Addressing Mode 5 - Coprocessor Data Transfer			
Pre-indexed	Immediate offset	[Rn, #+/-<immed_8*4>]{!}	
	Zero offset	[Rn]	Equivalent to [Rn,#0]
Post-indexed	Immediate offset	[Rn], #+/-<immed_8*4>	
Unindexed	No offset	[Rn], {8-bit copro. option}	

ARM architecture versions	
n	ARM architecture version *n* and above.
*n*T	T variants of ARM architecture version *n* and above.
M	ARM architecture version 3M, and 4 and above excluding xM variants
*n*E	E variants of ARM architecture version *n* and above.

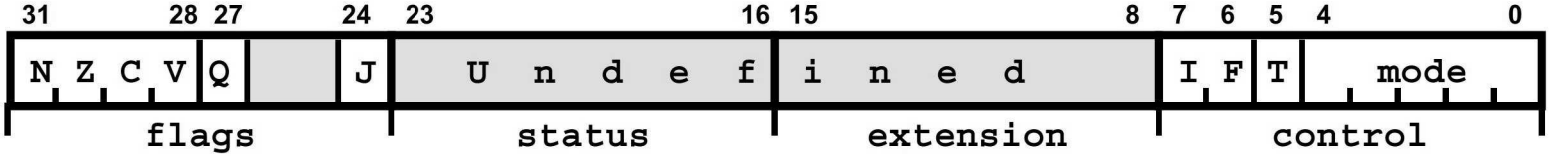
Operand 2		
Immediate value	#<immed_8r>	
Logical shift left immediate	Rm, LSL #<immed_5>	Allowed shifts 0-31
Logical shift right immediate	Rm, LSR #<immed_5>	Allowed shifts 1-32
Arithmetic shift right immediate	Rm, ASR #<immed_5>	Allowed shifts 1-32
Rotate right immediate	Rm, ROR #<immed_5>	Allowed shifts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

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]

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

Key to tables	
{!}	Updates base register after data transfer if ! present. (Post-indexed always updates.)
<immed_8r>	A 32-bit constant, formed by right-rotating an 8-bit value by an even number of bits.
+/-	+ or -. (+ may be omitted.)

Registre d'état **CPSR** (Current Program Status Register)



- Condition code flags
  - ✓ N = Negative result from ALU
  - ✓ Z = Zero result from ALU
  - ✓ C = ALU operation Carried out
  - ✓ V = ALU operation oVerflowed
- Sticky Overflow flag - Q flag
  - ✓ Architecture 5TE/J only
  - ✓ Indicates if saturation has occurred
- J bit
  - ✓ Architecture 5TEJ only
  - ✓ J = 1: Processor in Jazelle state
- Interrupt Disable bits.
  - ✓ I = 1: Disables the IRQ.
  - ✓ F = 1: Disables the FIQ.
- T Bit
  - ✓ Architecture xT only
  - ✓ T = 0: Processor in ARM state
  - ✓ T = 1: Processor in Thumb state
- Mode bits
  - 10000 USER
  - 10001 FIQ
  - 10010 IRQ
  - 10011 Supervisor
  - 10111 Abort
  - 11011 Undefined
  - 11111 System