ARM7TDMI Instruction Set Quick Reference

Opcode [31:28]	Mnemonic Extension	Meaning	Condition flag state
0000	EQ	Equal	Z==1
0001	NE	Not equal	Z==0
0010	CS/HS	Carry set / unsigned higher or same	C==1
0011	CC/LO	Carry clear / unsigned lower	C==0
0100	MI	Minus / negative	N==1
0101	PL	Plus / positive or zero	N==0
0110	VS	Overflow	V==1
0111	VC	No overflow	V==0
1000	HI	Unsigned higher	(C==1) AND (Z==0)
1001	LS	Unsigned lower or same	(C==0) OR (Z==1)
1010	GE	Signed greater than or equal	N == V
1011	LT	Signed less than	N != V
1100	GT	Signed greater than	(Z==0) AND (N==V)
1101	LE	Signed less than or equal	(Z==1) OR (N!=V)
1110	AL	Always (unconditional)	Not applicable
1111	(NV)	Never	Obsolete, unpredictable in ARM7TDMI

Shifter Operands

	Syntax	Example
Immediate	#<immediate>	CMP R0, #7
Register	<Rm>	CMP R0, R1
Shifted Register	<Rm>, LSL/LSR/ASR/ROR #<immediate>	CMP R0, R1, LSL #7
	<Rm>, LSL/LSR/ASR/ROR <Rs>	CMP R0, R1, ROR R2
	<Rm>, RRX	CMP R0, R1, RRX

Load/Store Register Addressing Modes (LDR/LDRB/STR/STRB)

Mode	Syntax	Effects
Base register with immediate offset	[Rn, #+/-<offset12>]	memory_address = Rn +/- offset12 Rn is unchanged after instruction
Base register with register offset	[Rn, +/-<Rm>]	memory_address = Rn +/- Rm Rn is unchanged after instruction
Base register with shifted register offset	[Rn, +/-<Rm>, <shift> #<shift_immediate>]	memory_address = Rn +/- shifted_Rm Rn is unchanged after instruction
Base register with immediate offset, pre-indexed	[Rn, #+/-<offset12>]!	memory_address = Rn +/- offset12 Rn = memory_address after instruction
Base register with register offset, pre-indexed	[Rn, +/-<Rm>]!	memory_address = Rn +/- Rm Rn = memory_address after instruction
Base register with shifted register offset, pre-indexed	[Rn, +/-<Rm>, <shift> #<shift_immediate>]!	memory_address = Rn +/- shifted_Rm Rn = memory_address after instruction
Base register with immediate offset, post-indexed	[Rn], #+/-<offset12>	memory_address = Rn Rn = Rn +/- offset12 after instruction
Base register with register offset, post-indexed	[Rn], +/-<Rm>	memory_address = Rn Rn = Rn +/- Rm after instruction
Base register with shifted register offset, post-indexed	[Rn], +/-<Rm>, <shift> #<shift_immediate>	memory_address = Rn Rn = Rn +/- shifted_Rm after instruction

The *<shift> #<shift_immediate>* fields can be one of LSL #0-31, LSR #1-32, ASR #1-32, ROR #1-32, RRX

Load/Store Register Addressing Modes (LDRSB, LDRH, LDRSH, STRH)

Mode	Syntax	Effects
Base register with immediate offset	[Rn, #+/-<offset8>]	memory_address = Rn +/- offset8 Rn is unchanged after instruction
Base register with register offset	[Rn, +/-<Rm>]	memory_address = Rn +/- Rm Rn is unchanged after instruction
Base register with immediate offset, pre-indexed	[Rn, #+/-<offset8>]!	memory_address = Rn +/- offset8 Rn = memory_address after instruction
Base register with register offset, pre-indexed	[Rn, +/-<Rm>]!	memory_address = Rn +/- Rm Rn = memory_address after instruction
Base register with immediate offset, post-indexed	[Rn], #+/-<offset8>	memory_address = Rn Rn = Rn +/- offset8 after instruction
Base register with register offset, post-indexed	[Rn], +/-<Rm>	memory_address = Rn Rn = Rn +/- Rm after instruction

Instruction Set

Syntax	**RTL (if condition is met)**	**Flags**
ADC{<cond>}{S} <Rd>, <Rn>, <shifter_operand>	Rd ← Rn + shifter_operand + C	N, Z, V, C
ADD{<cond>}{S} <Rd>, <Rn>, <shifter_operand>	Rd ← Rn + shifter_operand	N, Z, V, C
AND{<cond>}{S} <Rd>, <Rn>, <shifter_operand>	Rd ← Rn AND shifter_operand	N, Z, C
B{<cond>} <target_address>	PC ← PC + (signed_immediate_24 << 2)	None
BL{<cond>} <target_address>	R14 ← address of next instruction (return address) PC ← PC + (signed_immediate_24 << 2)	None
BIC{<cond>}{S} <Rd>, <Rn>, <shifter_operand>	Rd ← Rn AND NOT shifter_operand	N, Z
BX{<cond>} <Rm>	T flag ← Rm[0] PC ← Rm & 0xFFFFFFFE	None
CMN{<cond>} <Rn>, <shifter_operand>	Rn + shifter_operand	N, Z, V, C
CMP{<cond>} <Rn>, <shifter_operand>	Rn - shifter_operand	N, Z, V, C
EOR{<cond>}{S} <Rd>, <Rn>, <shifter_operand>	Rd ← Rn XOR shifter_operand	N, Z, C

LDM{<cond>}<addressing_mode>, <Rn>{!}, <registers>	start_address ← Rn
	for i = 0 to 14
	  if(register_list[i] == 1)
	  Ri ← memory[next_address]
	if(register_list[15] == 1)
	  PC ← memory[next_address] & 0xFFFFFFFC
	if(writeback) Rn ← end_address
		None

LDR{<cond>} <Rd>, <addressing_mode>	Rd ← memory[memory_address] if(writeback) Rn ← end_address	None
LDRB{<cond>} <Rd>, <addressing_mode>	Rd[7:0] ← memory[memory_address], Rd[31:8] ← 0 if(writeback) Rn ← end_address	
LDRH{<cond>} <Rd>, <addressing_mode>	Rd[15:0] ← memory[memory_address], Rd[31:16] ← 0 if(writeback) Rn ← end_address	None
LDRSB{<cond>} <Rd>, <addressing_mode>	Rd[7:0] ← memory[memory_address] Rd[31:8] ← Rd[7] (sign-extension) if(writeback) Rn ← end_address	None

Syntax	RTL (if condition is met)	Flags
LDRSH{<cond>} <Rd>, <addressing_mode>	Rd[15:0] ← memory[memory_address], Rd[31:16] ← Rd[15] (sign-extension) if(writeback) Rn ← end_address	None
MLA{<cond>}{S} <Rd >, <Rm>, <Rs>, <Rn>	Rd ← Rn + (Rs • Rm)	N, Z (C unpredictable)
MOV{<cond>}{S} <Rd>, <shifter_operand>	Rd ← shifter_operand if(S==1 and Rd==R15) CPSR ← SPSR	N, Z, C
MRS{<cond>} <Rd >, CPSR/SPSR	Rd ← CPSR/SPSR	None
MSR{<cond>} SPSR_/CPSR_<fields>, #<immediate> MSR{<cond>} SPSR_/CPSR_<fields>, <Rm>	CPSR/SPSR ← immediate/register value	N/A
MUL{<cond>}{S} <Rd >, <Rm>, <Rs>	Rd ← Rs • Rm	N, Z (C unpredictable)
MVN{<cond>}{S} <Rd>, <shifter_operand>	Rd ← NOT shifter_operand if(S==1 and Rd==R15) CPSR ← SPSR	N, Z, C
ORR{<cond>}{S} <Rd>, <Rn>, <shifter_operand>	Rd ← Rn OR shifter_operand	N, Z, C
RSB{<cond>}{S} <Rd>, <Rn>, <shifter_operand>	Rd ← shifter_operand - Rn	N, Z, V, C
RSC{<cond>}{S} <Rd>, <Rn>, <shifter_operand>	Rd ← shifter_operand – Rn – NOT C	N, Z, V, C
SBC{<cond>}{S} <Rd>, <Rn>, <shifter_operand>	Rd ← Rn - shifter_operand – NOT C	N, Z, V, C
SMLAL{<cond>}{S} <Rd_LSW>, <Rd_MSW>, <Rm>, <Rs>	Rd_MSW:Rd_LSW ← Rd_MSW:Rd_LSW + (Rs • Rm)	N, Z (V, C unpredictable)
SMULL{<cond>}{S} <Rd_LSW>, <Rd_MSW>, <Rm>, <Rs>	Rd_MSW:Rd_LSW ← Rs • Rm	N, Z (V, C unpredictable)

STM{<cond>}<addressing_mode>, <Rn>{!}, <registers>	start_address ← Rn
	for i = 0 to 15
	  if(register_list[i] == 1)
	  memory[next_address] ← Ri
	if(writeback) Rn ← end_address
		None

STR{<cond>} <Rd>, <addressing_mode>	memory[memory_address] ← Rd if(writeback) Rn ← end_address	None
STRB{<cond>} <Rd>, <addressing_mode>	memory[memory_address] ← Rd[7:0] if(writeback) Rn ← end_address	None

Syntax	**RTL (if condition is met)**	**Flags**
STRH{<cond>} <Rd>, <addressing_mode>	memory[memory_address] ← Rd[15:0] if(writeback) Rn ← end_address	None
SUB{<cond>}{S} <Rd>, <Rn>, <shifter_operand>	Rd ← Rn - shifter_operand	N, Z, V, C
SWI{<cond>} <immediate_24>	R14_svc ← address of next instruction after SWI instruction SPSR_svc ← CPSR ; save current CPSR CPSR[4:0] ← 10011b ; supervisor mode CPSR[5] ← 0 ; ARM execution CPSR[7] ← 1 ; disable interrupts PC ← 0x00000008 ; jump to exception vector	N/A
SWP{<cond>} <Rd>, <Rm>, [<Rn>]	temp ← [Rn], [Rn] ← Rm, Rd ← temp	None
SWPB{<cond>} <Rd>, <Rm>, [<Rn>]	temp ← [Rn], [Rn] ← Rm, Rd ← temp	None
TEQ{<cond>} <Rn>, <shifter_operand>	Rn XOR shifter_operand	N, Z, C
TST{<cond>} <Rn>, <shifter_operand>	Rn AND shifter_operand	N, Z, C
UMLAL{<cond>}{S} <Rd_LSW>, <Rd_MSW>, <Rm>, <Rs>	Rd_MSW:Rd_LSW ← Rd_MSW:Rd_LSW + (Rs • Rm)	N, Z (V, C unpredictable)
UMULL{<cond>}{S} <Rd_LSW>, <Rd_MSW>, <Rm>, <Rs>	Rd_MSW:Rd_LSW ← Rs • Rm	N, Z (V, C unpredictable)

Pseudo-Instructions	
ADR{cond} <Rd>, <label> ADRL{cond} <Rd>, <label>	Rd ← label_address
ASR{cond}{S} <Rd>, <Rm>, <Rs> ASR{cond}{S} <Rd>, <Rm>, <#shift_count>	Alternate syntax for MOV{S} <Rd>, <Rm>, ASR <Rs> or <#shift_count> If Rm is not specified, it is assumed to be the same as Rd
LDR{cond} <Rd>, =<expression> LDR{cond} <Rd>, =<label-expression>	Assembler will try to encode as a MOV immediate. If it cannot, it will allocate a word initialized with the value and load from there using PC-relative addressing. If it is a label, the address of the label is stored in the literal pool and loaded from there.
LSL{cond}{S} <Rd>, <Rm>, <Rs> LSL{cond}{S} <Rd>, <Rm>, <#shift_count>	Alternate syntax for MOV{S} <Rd>, <Rm>, LSL <Rs> or <#shift_count> If Rm is not specified, it is assumed to be the same as Rd
LSR{cond}{S} <Rd>, <Rm>, <Rs> LSR{cond}{S} <Rd>, <Rm>, <#shift_count>	Alternate syntax for MOV{S} <Rd>, <Rm>, LSR <Rs> or <#shift_count> If Rm is not specified, it is assumed to be the same as Rd
NOP	No operation – encoded as MOV R0, R0, LSL#0 (assembler dependent)
POP{cond} reg_list	Implements FD stack (equivalent to LDMIA R13!, reg_list)
PUSH{cond} reg_list	Implements FD stack (equivalent to STMDB R13!, reg_list)
ROR{cond}{S} <Rd>, <Rm>, <Rs> ROR{cond}{S} <Rd>, <Rm>, <#shift_count>	Alternate syntax for MOV{S} <Rd>, <Rm>, ROR <Rs> or <#shift_count> If Rm is not specified, it is assumed to be the same as Rd
RRX{cond}{S} <Rd>, <Rm>	Alternate syntax for MOV{S} <Rd>, <Rm>, RRX If Rm is not specified, it is assumed to be the same as Rd