ARM® 和 Thumb®-2 指令集
快速参考卡

表关键字			
Rm {, <opsh>}	请参阅表**寄存器，可选择移动常数个位**		
<Operand2>	请参阅表**灵活的操作数 2**。移位和循环移位只可用于 Operand2。	<reglist>	以逗号隔开的寄存器列表，括在大括号 { 和 } 内。
<fields>	请参阅表 **PSR 域**。	<reglist-PC>	作为 <reglist>，不能包含 PC。
<PSR>	CPSR（当前处理器状态寄存器）或 SPSR（保存的处理器状态寄存器）	<reglist+PC>	作为 <reglist>，包含 PC。
C* ,V*	在体系结构 v4 及其更早版本中，标记不可预知；在体系结构 v5 及其以后版本中，标记保持不变。	+/-	+ 或 -£（+ 可省略。）
<Rs|sh>	可为 Rs 或一个直接移位值。每种移位类型的允许值与		请参阅表 **ARM 体系结构版本**。
	寄存器，可选择移动常数个位 中的相同。	<iflags>	中断标记。一个或多个 a、i、f（中止、中断、快速中断）。
x、y	B 或 T，B 表示半寄存器 [15:0]，T 表示半寄存器 [31:16]。	<p_mode>	请参阅表**处理器模式**
<imm8m>	ARM：32 位常数，由 8 位值向右循环移偶数位生成。	SPm	<p_mode> 所指定的处理模式的 SP
	Thumb：32 位常数，由 8 位值左移任意位生成，	<lsb>	位域的最低有效位。
	格式模式为 0xXYXYXYXY、0x00XY00XY 或 0xXY00XY00。	<width>	位域宽度，<width> + <lsb> 必须小于或等于 32。
<prefix>	请参阅**并行指令的前缀**	{X}	如果有 X，则 RsX 为 Rs 循环 16 位生成。否则，RsX 为 Rs。
{IA|IB|DA|DB}	之后增加、之前增加、之后减小、之前减小。	{!}	如果有 !，则在数据传送完毕后更新基址寄存器（前变址）。
	IB 和 DA 不可用于 Thumb 状态下。如果省略，则缺省时为 IA。	{S}	如果有 S，则更新条件标记。
<size>	B、SB、H 或 SH，含义分别为字节、有符号字节、半字和有符号半字。	{T}	如果有 T，则带有用户模式特权。
	SB 和 SH 不可用于 STR 指令。	{R}	如果存在 R，则对结果进行舍入，否则将其截断。

运算			汇编程序	S 更新	操作	注释
加法	加法		ADD{S} Rd, Rn, <Operand2>	N Z C V	Rd := Rn + Operand2	N
	带进位		ADC{S} Rd, Rn, <Operand2>	N Z C V	Rd := Rn + Operand2 + 进位	N
	宽	T2	ADD Rd, Rn, #<imm12>		Rd := Rn + imm12，imm12 的范围为 0-4095	T、P
	饱和 {加倍}	5E	Q{D}ADD Rd, Rm, Rn		Rd := SAT(Rm + Rn) 加倍: Rd := SAT(Rm + SAT(Rn * 2))	Q
寻址	PC 相对的寻址		ADR Rd, <label>		Rd := <label>，有关 <label> 相对于当前指令的范围，请参阅注释 L	N、L
减法	减法		SUB{S} Rd, Rn, <Operand2>	N Z C V	Rd := Rn - Operand2	N
	带进位		SBC{S} Rd, Rn, <Operand2>	N Z C V	Rd := Rn - Operand2 - NOT（进位）	N
	宽	T2	SUB Rd, Rn, #<imm12>	N Z C V	Rd := Rn - imm12，imm12 的范围为 0-4095	T、P
	反向减法		RSB{S} Rd, Rn, <Operand2>	N Z C V	Rd := Operand2 - Rn	N
	带进位反向减法		RSC{S} Rd, Rn, <Operand2>	N Z C V	Rd := Operand2 - Rn - NOT（进位）	A
	饱和 {加倍}	5E	Q{D}SUB Rd, Rm, Rn		Rd := SAT(Rm - Rn) 加倍: Rd := SAT(Rm - SAT(Rn * 2))	Q
	从异常中返回，无出栈。		SUBS PC, LR, #<imm8>		PC = LR - imm8，CPSR = SPSR（当前模式），imm8 的范围为 0-255。	T
并行算法	半字方式加法	6	<prefix>ADD16 Rd, Rn, Rm		Rd[31:16] := Rn[31:16] + Rm[31:16]，Rd[15:0] := Rn[15:0] + Rm[15:0]	G
	半字方式减法	6	<prefix>SUB16 Rd, Rn, Rm		Rd[31:16] := Rn[31:16] - Rm[31:16]，Rd[15:0] := Rn[15:0] - Rm[15:0]	G
	字节方式加法	6	<prefix>ADD8 Rd, Rn, Rm		Rd[31:24] := Rn[31:24] + Rm[31:24]，Rd[23:16] := Rn[23:16] + Rm[23:16]，Rd[15:8] := Rn[15:8] + Rm[15:8]，Rd[7:0] := Rn[7:0] + Rm[7:0]	G
	字节方式减法	6	<prefix>SUB8 Rd, Rn, Rm		Rd[31:24] := Rn[31:24] - Rm[31:24]，Rd[23:16] := Rn[23:16] - Rm[23:16]，Rd[15:8] := Rn[15:8] - Rm[15:8]，Rd[7:0] := Rn[7:0] - Rm[7:0]	G
	交换半字，半字方式加法，半字方式减法	6	<prefix>ASX Rd, Rn, Rm		Rd[31:16] := Rn[31:16] + Rm[15:0]，Rd[15:0] := Rn[15:0] - Rm[31:16]	G
	交换半字，半字方减法，半字方式加法	6	<prefix>SAX Rd, Rn, Rm		Rd[31:16] := Rn[31:16] - Rm[15:0]，Rd[15:0] := Rn[15:0] + Rm[31:16]	G
	差值的绝对值无符号求和	6	USAD8 Rd, Rm, Rs		Rd := Abs(Rm[31:24] - Rs[31:24]) + Abs(Rm[23:16] - Rs[23:16]) + Abs(Rm[15:8] - Rs[15:8]) + Abs(Rm[7:0] - Rs[7:0])	
	差值的绝对值无符号求和，再累加	6	USADA8 Rd, Rm, Rs, Rn		Rd := Rn + Abs(Rm[31:24] - Rs[31:24]) + Abs(Rm[23:16] - Rs[23:16]) + Abs(Rm[15:8] - Rs[15:8]) + Abs(Rm[7:0] - Rs[7:0])	
饱和	有符号饱和字，右移	6	SSAT Rd, #<sat>, Rm{, ASR <sh>}		Rd := SignedSat((Rm ASR sh), sat)。<sat> 的范围为 1-32，<sh> 的范围为 1-31。	Q、R
	有符号饱和字，左移	6	SSAT Rd, #<sat>, Rm{, LSL <sh>}		Rd := SignedSat((Rm LSL sh), sat)。<sat> 的范围为 1-32，<sh> 的范围为 0-31。	Q
	有符号饱和两个半字	6	SSAT16 Rd, #<sat>, Rm		Rd[31:16] := SignedSat(Rm[31:16], sat)，Rd[15:0] := SignedSat(Rm[15:0], sat)。<sat> 的范围为 1-16。	Q
	无符号饱和字，右移	6	USAT Rd, #<sat>, Rm{, ASR <sh>}		Rd := UnsignedSat((Rm ASR sh), sat)。<sat> 的范围为 0-31，<sh> 的范围为 1-31。	Q、R
	无符号饱和字，左移	6	USAT Rd, #<sat>, Rm{, LSL <sh>}		Rd := UnsignedSat((Rm LSL sh), sat)。<sat> 的范围为 0-31，<sh> 的范围为 0-31。	Q
	无符号饱和两个半字	6	USAT16 Rd, #<sat>, Rm		Rd[31:16] := UnsignedSat(Rm[31:16], sat)，Rd[15:0] := UnsignedSat(Rm[15:0], sat)。<sat> 范围为 0-15。	Q

ARM 和 Thumb-2 指令集
快速参考卡

运算			汇编程序	S 更新	操作	注释
乘法	乘法		`MUL{S} Rd, Rm, Rs`	N Z C*	Rd := (Rm * Rs)[31:0] （如果 Rm 为 Rd，则 S 可用于 Thumb-2 中。）	N、S
	乘加		`MLA{S} Rd, Rm, Rs, Rn`	N Z C*	Rd := (Rn + (Rm * Rs))[31:0]	S
	乘减	T2	`MLS Rd, Rm, Rs, Rn`		Rd := (Rn - (Rm * Rs))[31:0]	
	无符号长乘法		`UMULL{S} RdLo, RdHi, Rm, Rs`	N Z C* V*	RdHi,RdLo := unsigned(Rm * Rs)	S
	长整数无符号乘加		`UMLAL{S} RdLo, RdHi, Rm, Rs`	N Z C* V*	RdHi,RdLo := unsigned(RdHi,RdLo + Rm * Rs)	S
	无符号长乘法，两次加法	6	`UMAAL RdLo, RdHi, Rm, Rs`		RdHi,RdLo := unsigned(RdHi + RdLo + Rm * Rs)	
	长整数有符号乘法		`SMULL{S} RdLo, RdHi, Rm, Rs`	N Z C* V*	RdHi,RdLo := signed(Rm * Rs)	S
	长整数乘加		`SMLAL{S} RdLo, RdHi, Rm, Rs`	N Z C* V*	RdHi,RdLo := signed(RdHi,RdLo + Rm * Rs)	S
	16 * 16 位	5E	`SMULxy Rd, Rm, Rs`		Rd := Rm[x] * Rs[y]	
	32 * 16 位	5E	`SMULWy Rd, Rm, Rs`		Rd := (Rm * Rs[y])[47:16]	
	16 * 16 位并累加	5E	`SMLAxy Rd, Rm, Rs, Rn`		Rd := Rn + Rm[x] * Rs[y]	Q
	32 * 16 位并累加	5E	`SMLAWy Rd, Rm, Rs, Rn`		Rd := Rn + (Rm * Rs[y])[47:16]	Q
	长整数 16 * 16 位并累加	5E	`SMLALxy RdLo, RdHi, Rm, Rs`		RdHi,RdLo := RdHi,RdLo + Rm[x] * Rs[y]	
	两次有符号乘法，乘积相加	6	`SMUAD{X} Rd, Rm, Rs`		Rd := Rm[15:0] * RsX[15:0] + Rm[31:16] * RsX[31:16]	Q
	并累加	6	`SMLAD{X} Rd, Rm, Rs, Rn`		Rd := Rn + Rm[15:0] * RsX[15:0] + Rm[31:16] * RsX[31:16]	Q
	并累加（长整数）	6	`SMLALD{X} RdLo, RdHi, Rm, Rs`		RdHi,RdLo := RdHi,RdLo + Rm[15:0] * RsX[15:0] + Rm[31:16] * RsX[31:16]	
	两次有符号乘法，乘积相减	6	`SMUSD{X} Rd, Rm, Rs`		Rd := Rm[15:0] * RsX[15:0] - Rm[31:16] * RsX[31:16]	Q
	并累加	6	`SMLSD{X} Rd, Rm, Rs, Rn`		Rd := Rn + Rm[15:0] * RsX[15:0] - Rm[31:16] * RsX[31:16]	Q
	并累加（长整数）	6	`SMLSLD{X} RdLo, RdHi, Rm, Rs`		RdHi,RdLo := RdHi,RdLo + Rm[15:0] * RsX[15:0] - Rm[31:16] * RsX[31:16]	
	有符号高位字乘法	6	`SMMUL{R} Rd, Rm, Rs`		Rd := (Rm * Rs)[63:32]	
	并累加	6	`SMMLA{R} Rd, Rm, Rs, Rn`		Rd := Rn + (Rm * Rs)[63:32]	
	乘减	6	`SMMLS{R} Rd, Rm, Rs, Rn`		Rd := Rn - (Rm * Rs)[63:32]	
	带内部 40 位累加	XS	`MIA Ac, Rm, Rs`		Ac := Ac + Rm * Rs	
	组合半字	XS	`MIAPH Ac, Rm, Rs`		Ac := Ac + Rm[15:0] * Rs[15:0] + Rm[31:16] * Rs[31:16]	
	半字	XS	`MIAxy Ac, Rm, Rs`		Ac := Ac + Rm[x] * Rs[y]	
除法	有符号或无符号	RM	`<op> Rd, Rn, Rm`		Rd := Rn / Rm <op> 为 `SDIV`（有符号）或 `UDIV`（无符号）	
移动数据	移动		`MOV{S} Rd, <Operand2>`	N Z C	Rd := Operand2 请参阅移位指令	N
	求反移动		`MVN{S} Rd, <Operand2>`	N Z C	Rd := 0xFFFFFFFF EOR Operand2	N
	移到顶部	T2	`MOVT Rd, #<imm16>`		Rd[31:16] := imm16，Rd[15:0] 不受影响，imm16 的范围为 0-65535	
	宽	T2	`MOV Rd, #<imm16>`		Rd[15:0] := imm16，Rd[31:16] = 0，imm16 范围为 0-65535	
	40 位累加器到寄存器	XS	`MRA RdLo, RdHi, Ac`		RdLo := Ac[31:0]，RdHi := Ac[39:32]	
	寄存器到 40 位累加器	XS	`MAR Ac, RdLo, RdHi`		Ac[31:0] := RdLo，Ac[39:32] := RdHi	
移位	算术右移		`ASR{S} Rd, Rm, <Rs|sh>`	N Z C	Rd := ASR(Rm, Rs|sh) 与 `MOV{S} Rd, Rm, ASR <Rs|sh>` 相同	N
	逻辑左移		`LSL{S} Rd, Rm, <Rs|sh>`	N Z C	Rd := LSL(Rm, Rs|sh) 与 `MOV{S} Rd, Rm, LSL <Rs|sh>` 相同	N
	逻辑右移		`LSR{S} Rd, Rm, <Rs|sh>`	N Z C	Rd := LSR(Rm, Rs|sh) 与 `MOV{S} Rd, Rm, LSR <Rs|sh>` 相同	N
	向右循环移		`ROR{S} Rd, Rm, <Rs|sh>`	N Z C	Rd := ROR(Rm, Rs|sh) 与 `MOV{S} Rd, Rm, ROR <Rs|sh>` 相同	N
	带扩展的向右循环移		`RRX{S} Rd, Rm`	N Z C	Rd := RRX(Rm) 与 `MOV{S} Rd, Rm, RRX` 相同	
计算前导零数目		5	`CLZ Rd, Rm`		Rd := Rm 中的前导零的数目	
比较	比较		`CMP Rn, <Operand2>`	N Z C V	更新 Rn - Operand2 的 CPSR 标记	N
	与负数比较		`CMN Rn, <Operand2>`	N Z C V	更新 Rn + Operand2 的 CPSR 标记	N
逻辑	测试		`TST Rn, <Operand2>`	N Z C	更新 Rn AND Operand2 的 CPSR 标记	N
	相等测试		`TEQ Rn, <Operand2>`	N Z C	更新 Rn EOR Operand2 的 CPSR 标记	
	与		`AND{S} Rd, Rn, <Operand2>`	N Z C	Rd := Rn AND Operand2	N
	异或		`EOR{S} Rd, Rn, <Operand2>`	N Z C	Rd := Rn EOR Operand2	N
	或		`ORR{S} Rd, Rn, <Operand2>`	N Z C	Rd := Rn OR Operand2	N
	或非	T2	`ORN{S} Rd, Rn, <Operand2>`	N Z C	Rd := Rn OR NOT Operand2	T
	位清除		`BIC{S} Rd, Rn, <Operand2>`	N Z C	Rd := Rn AND NOT Operand2	N

ARM 和 Thumb-2 指令集
快速参考卡

运算			汇编程序	操作	注释
位域	位域清零	T2	`BFC Rd, #<lsb>, #<width>`	Rd[(width+lsb-1):lsb] := 0，Rd 的其他位不受影响	
	位域插入	T2	`BFI Rd, Rn, #<lsb>, #<width>`	Rd[(width+lsb-1):lsb] := Rn[(width-1):0]，Rd 的其他位不受影响	
	有符号位域提取	T2	`SBFX Rd, Rn, #<lsb>, #<width>`	Rd[(width-1):0] = Rn[(width+lsb-1):lsb]，Rd[31:width] = 复制 (Rn[width+lsb-1])	
	无符号位域提取	T2	`UBFX Rd, Rn, #<lsb>, #<width>`	Rd[(width-1):0] = Rn[(width+lsb-1):lsb]，Rd[31:width] = 复制 (0)	
组合	组合 低半字 + 高半字	6	`PKHBT Rd, Rn, Rm{, LSL #<sh>}`	Rd[15:0] := Rn[15:0]，Rd[31:16] := (Rm LSL sh)[31:16]。sh 的范围为 0-31。	
	组合 高半字 + 低半字	6	`PKHTB Rd, Rn, Rm{, ASR #<sh>}`	Rd[31:16] := Rn[31:16]，Rd[15:0] := (Rm ASR sh)[15:0]。sh 的范围为 1-32。	
有符号扩展	半字到字	6	`SXTH Rd, Rm{, ROR #<sh>}`	Rd[31:0] := SignExtend((Rm ROR (8 * sh))[15:0])。sh 的范围为 0-3。	N
	两个字节到半字	6	`SXTB16 Rd, Rm{, ROR #<sh>}`	Rd[31:16] := SignExtend((Rm ROR (8 * sh))[23:16]), Rd[15:0] := SignExtend((Rm ROR (8 * sh))[7:0])。sh 的范围为 0-3。	
	字节到字	6	`SXTB Rd, Rm{, ROR #<sh>}`	Rd[31:0] := SignExtend((Rm ROR (8 * sh))[7:0])。sh 的范围为 0-3。	N
无符号扩展	半字到字	6	`UXTH Rd, Rm{, ROR #<sh>}`	Rd[31:0] := ZeroExtend((Rm ROR (8 * sh))[15:0])。sh 的范围为 0-3。	N
	两个字节到半字	6	`UXTB16 Rd, Rm{, ROR #<sh>}`	Rd[31:16] := ZeroExtend((Rm ROR (8 * sh))[23:16]), Rd[15:0] := ZeroExtend((Rm ROR (8 * sh))[7:0])。sh 的范围为 0-3。	
	字节到字	6	`UXTB Rd, Rm{, ROR #<sh>}`	Rd[31:0] := ZeroExtend((Rm ROR (8 * sh))[7:0])。sh 的范围为 0-3。	N
带加法的有符号扩展	半字到字，加法	6	`SXTAH Rd, Rn, Rm{, ROR #<sh>}`	Rd[31:0] := Rn[31:0] + SignExtend((Rm ROR (8 * sh))[15:0])。sh 的范围为 0-3。	
	两个字节到半字，加法	6	`SXTAB16 Rd, Rn, Rm{, ROR #<sh>}`	Rd[31:16] := Rn[31:16] + SignExtend((Rm ROR (8 * sh))[23:16]), Rd[15:0] := Rn[15:0] + SignExtend((Rm ROR (8 * sh))[7:0])。sh 的范围为 0-3。	
	字节到字，加法	6	`SXTAB Rd, Rn, Rm{, ROR #<sh>}`	Rd[31:0] := Rn[31:0] + SignExtend((Rm ROR (8 * sh))[7:0])。sh 的范围为 0-3。	
带加法的无符号扩展	半字到字，加法	6	`UXTAH Rd, Rn, Rm{, ROR #<sh>}`	Rd[31:0] := Rn[31:0] + ZeroExtend((Rm ROR (8 * sh))[15:0])。sh 的范围为 0-3。	
	两个字节到半字，加法	6	`UXTAB16 Rd, Rn, Rm{, ROR #<sh>}`	Rd[31:16] := Rn[31:16] + ZeroExtend((Rm ROR (8 * sh))[23:16]), Rd[15:0] := Rn[15:0] + ZeroExtend((Rm ROR (8 * sh))[7:0])。sh 的范围为 0-3。	
	字节到字，加法	6	`UXTAB Rd, Rn, Rm{, ROR #<sh>}`	Rd[31:0] := Rn[31:0] + ZeroExtend((Rm ROR (8 * sh))[7:0])。sh 的范围为 0-3。	
反转	字中的位	T2	`RBIT Rd, Rm`	For (i = 0; i < 32; i++) : Rd[i] = Rm[31- i]	
	字中的字节	6	`REV Rd, Rm`	Rd[31:24] := Rm[7:0]，Rd[23:16] := Rm[15:8]，Rd[15:8] := Rm[23:16]，Rd[7:0] := Rm[31:24]	N
	两个半字中的字节	6	`REV16 Rd, Rm`	Rd[15:8] := Rm[7:0]，Rd[7:0] := Rm[15:8]，Rd[31:24] := Rm[23:16]，Rd[23:16] := Rm[31:24]	N
	低半字中的字节，符号扩展	6	`REVSH Rd, Rm`	Rd[15:8] := Rm[7:0]，Rd[7:0] := Rm[15:8]，Rd[31:16] := Rm[7] * &FFFF	N
选择	选择字节	6	`SEL Rd, Rn, Rm`	如果 GE[0] = 1，则 Rd[7:0] := Rn[7:0]，否则 Rd[7:0] := Rm[7:0] GE[1]、GE[2]、GE[3] 时位 [15:8]、[23:16]、[31:24] 的选择方法与 GE[0] 相似	
条件判断	条件判断	T2	`IT{pattern} {cond}`	依据不同的模式，最多由连续四个条件指令句组成。模式为一个字符串，最多三个字母。所有字母都可为 T（然后）或 E（否则）。 IT 之后的第一条指令可有条件 cond。如果相应的字母为 T，则后续指令可有条件 cond；如果相应的字母为 E，则为该 cond 的反面情况。 有关可用条件代码的信息，请参阅表**条件字段**。	T、U
跳转	跳转		`B <label>`	PC := label。label 为此指令 ±32MB（T2: ±16MB，T: -252 - +256B）	N、B
	带链接的跳转		`BL <label>`	LR := 下一指令的地址，PC := label。label 为此指令 ±32MB (T2: ±16MB)。	
	跳转并交换	4T	`BX Rm`	PC := Rm。如果 Rm[0] 为 1，目标为 Thumb；如果 Rm[0] 为 0，目标则为 ARM。	N
	带链接和交换 (1)	5T	`BLX <label>`	LR := 下一指令的地址，PC := label，更改指令集。 label 为此指令 ±32MB (T2: ±16MB)。	C
	带链接和交换 (2)	5	`BLX Rm`	LR := 下一指令的地址，PC := Rm[31:1]。如果 Rm[0] 为 1，更改为 Thumb；如果 Rm[0] 为 0，则更改为 ARM。	N
	跳转并更改为 Jazelle 状态	5J	`BXJ Rm`	如果可用，更改为 Jazelle	
	比较，如果为（非）零，则跳转	T2	`CB{N}Z Rn,<label>`	如果 Rn {== 或 !=} 0，则 PC := label。label 为（此指令 + 4-130）。	N、T、U
	表跳转字节	T2	`TBB [Rn, Rm]`	PC = PC + ZeroExtend(Memory(Rn + Rm, 1) << 1)。跳转范围为 4-512。Rn 可为 PC。	T、U
	表跳转半字	T2	`TBH [Rn, Rm, LSL #1]`	PC = PC + ZeroExtend(Memory(Rn + Rm << 1, 2) << 1)。跳转范围为 4-131072。Rn 可为 PC。	T、U
移到 PSR 或从 PSR 移出	PSR 到寄存器		`MRS Rd, <PSR>`	Rd := PSR	
	寄存器到 PSR		`MSR <PSR>_<fields>, Rm`	PSR := Rm（仅选择字节）	
	立即数到 PSR		`MSR <PSR>_<fields>, #<imm8m>`	PSR := immed_8r（仅选择字节）	
更改处理器状态	更改处理器状态	6	`CPSID <iflags> {, #<p_mode>}`	禁用指定的中断，可选择更改模式。	U、N
		6	`CPSIE <iflags> {, #<p_mode>}`	启用指定的中断，可选择更改模式。	U、N
	改变处理器模式	6	`CPS #<p_mode>`		U
	设置端序	6	`SETEND <endianness>`	为加载和保存设置端序。`<endianness>` 可为 `BE`（大端）或 `LE`（小端）。	U、N

ARM 指令集
快速参考卡

加载和存储单个数据项		§	汇编程序	当 <op> 为 LDR 时执行的操作	当 <op> 为 STR 时执行的操作	注释
加载或存储字、字节或半字	直接偏移量		<op>{size}{T} Rd, [Rn {, #<offset>}]{!}	Rd := [address, size]	[address, size] := Rd	1、N
	后变址，立即数		<op>{size}{T} Rd, [Rn], #<offset>	Rd := [address, size]	[address, size] := Rd	2
	寄存器偏移量		<op>{size} Rd, [Rn, +/-Rm {, <opsh>}]{!}	Rd := [address, size]	[address, size] := Rd	3、N
	后变址，寄存器		<op>{size}{T} Rd, [Rn], +/-Rm {, <opsh>}	Rd := [address, size]	[address, size] := Rd	4
	PC 相对的		<op>{size} Rd, <label>	Rd := [label, size]	不可用	5、N
加载或存储双字	直接偏移量	5E	<op>D Rd1, Rd2, [Rn {, #<offset>}]{!}	Rd1 := [address]，Rd2 := [address + 4]	[address] := Rd1，[address + 4] := Rd2	6、9
	后变址，立即数	5E	<op>D Rd1, Rd2, [Rn], #<offset>	Rd1 := [address]，Rd2 := [address + 4]	[address] := Rd1，[address + 4] := Rd2	6、9
	寄存器偏移量	5E	<op>D Rd1, Rd2, [Rn, +/-Rm {, <opsh>}]{!}	Rd1 := [address]，Rd2 := [address + 4]	[address] := Rd1，[address + 4] := Rd2	7、9
	后变址，寄存器	5E	<op>D Rd1, Rd2, [Rn], +/-Rm {, <opsh>}	Rd1 := [address]，Rd2 := [address + 4]	[address] := Rd1，[address + 4] := Rd2	7、9
	PC 相对的	5E	<op>D Rd1, Rd2, <label>	Rd1 := [label]，Rd2 := [label + 4]	不可用	8、9

预载数据或指令		§ (PLD)	(PLI)	汇编程序	当 <op> 为 PLD 时执行的操作	当 <op> 为 PLI 时执行的操作	注释
	直接偏移量	5E	7	<op> [Rn {, #<offset>}]	预载 [address, 32]（数据）	预载 [address, 32]（指令）	1、C
	寄存器偏移量	5E	7	<op> [Rn, +/-Rm {, <opsh>}]	预载 [address, 32]（数据）	预载 [address, 32]（指令）	3、C
	PC 相对的	5E	7	<op> <label>	预载 [label, 32]（数据）	预载 [label, 32]（指令）	5、C

其他内存操作		§	汇编程序	操作	注释
加载多个	数据块加载		LDM{IA|~~IB|DA|~~DB} Rn{!}, <reglist-PC>	从 [Rn] 加载寄存器列表	N、I
	返回（并交换）		LDM{IA|~~IB|DA|~~DB} Rn{!}, <reglist+PC>	加载寄存器，PC := [address][31:1]（ 5T: 当 [address][0] 为 1 时，更改为 Thumb）	I
	并恢复 CPSR		LDM{IA|IB|DA|DB} Rn{!}, <reglist+PC>^	加载寄存器，跳转（§ 5T: 并交换），CPSR := SPSR。仅限异常模式。	I
	用户模式寄存器		LDM{IA|IB|DA|DB} Rn, <reglist-PC>^	从 [Rn] 加载用户模式寄存器列表。仅限特权模式。	I
弹出			POP <reglist>	LDM SP!, <reglist> 的规范格式	N
加载独占	信号运算	6	LDREX Rd, [Rn]	Rd := [Rn]，将地址标记为独占访问。如果不是共享地址，则为突出显示的标记设置。 Rd、Rn 不可为 PC。	
	半字或字节	6K	LDREX{H|B} Rd, [Rn]	Rd[15:0] := [Rn] 或 Rd[7:0] := [Rn]，将地址标记为独占访问。 如果不是共享地址，则为突出显示的标记设置。Rd、Rn 不可为 PC。	
	双字	6K	LDREXD Rd1, Rd2, [Rn]	Rd1 := [Rn]，Rd2 := [Rn+4]，将地址标记为独占访问 如果不是共享地址，则为突出显示的标记设置。Rd1、Rd2、Rn 不可为 PC。	9
存储多个	推入或阻止数据存储		STM{IA|~~IB|DA|~~DB} Rn{!}, <reglist>	将寄存器列表存储到 [Rn] 中	N、I
	用户模式寄存器		STM{IA|IB|DA|DB} Rn{!}, <reglist>^	将用户模式寄存器列表存储到 [Rn] 中。仅限特权模式。	I
推入			PUSH <reglist>	STMDB SP!, <reglist> 的规范格式	N
存储独占	信号运算	6	STREX Rd, Rm, [Rn]	如果允许，则 [Rn] := Rm，清除独占标记，Rd := 0。否则 Rd := 1。Rd、Rm、Rn 不可为 PC。	
	半字或字节	6K	STREX{H|B} Rd, Rm, [Rn]	如果允许，则 [Rn] := Rm[15:0] 或 [Rn] := Rm[7:0]，清除独占标记，Rd := 0。否则 Rd := 1 Rd、Rm、Rn 不可为 PC。	
	双字	6K	STREXD Rd, Rm1, Rm2, [Rn]	如果允许，则 [Rn] := Rm1，[Rn+4] := Rm2，清除独占标记，Rd := 0。否则 Rd := 1 Rd、Rm1、Rm2、Rn 不可为 PC。	9
清除独占		6K	CLREX	清除局部处理器独占标记	C

注释 加载、存储和预载操作的可用性和选项范围					
注释	ARM 字、B、D	ARM SB、H、SH	ARM T、BT	Thumb-2 字、B、SB、H、SH、D	Thumb-2 T、BT、SBT、HT、SHT
1	偏移量：- 4095 到 +4095	偏移量：-255 到 +255	不可用	偏移量：如果回写，则为 -255 到 +255，否则，为 -255 到 +4095	偏移量：0 到 +255，不允许回写
2	偏移量：- 4095 到 +4095	偏移量：-255 到 +255	偏移量：- 4095 到 +4095	偏移量：-255 到 +255	不可用
3	整个 {, <opsh>} 范围	{, <opsh>} 不允许	不可用	<opsh> 限制为 LSL #<sh>，<sh> 的范围为 0 到 3	不可用
4	整个 {, <opsh>} 范围	{, <opsh>} 不允许	整个 {, <opsh>} 范围	不可用	不可用
5	当前指令的 +/- 4092 范围内的标签	不可用	不可用	当前指令的 +/- 4092 范围内的标签	不可用
6	偏移量：-255 到 +255	-	-	偏移量：-1020 到 +1020，必须是 4 的倍数。	-
7	{, <opsh>} 不允许	-	-	不可用	-
8	当前指令的 +/- 252 范围内的标签	-	-	不可用	-
9	Rd1 编号为偶数，但不可为 r14，Rd2 == Rd1 + 1。	-	-	Rd1 != PC，Rd2 != PC	-

ARM 指令集
快速参考卡

协处理器运算	§	汇编程序		操作	注释
数据操作		CDP{2} <copr>, <op1>, CRd, CRn, CRm{, <op2>}		由协处理器定义	C2
从协处理器移到 ARM 寄存器		MRC{2} <copr>, <op1>, Rd, CRn, CRm{, <op2>}		由协处理器定义	C2
两个 ARM 寄存器移动	5E	MRRC <copr>, <op1>, Rd, Rn, CRm		由协处理器定义	
另两个 ARM 寄存器移动	6	MRRC2 <copr>, <op1>, Rd, Rn, CRm		由协处理器定义	C
从 ARM 寄存器移到协处理器		MCR{2} <copr>, <op1>, Rd, CRn, CRm{, <op2>}		由协处理器定义	C2
两个 ARM 寄存器移动	5E	MCRR <copr>, <op1>, Rd, Rn, CRm		由协处理器定义	
另两个 ARM 寄存器移动	6	MCRR2 <copr>, <op1>, Rd, Rn, CRm		由协处理器定义	C
加载和存储，前变址		<op>{2} <copr>, CRd, [Rn, #+/-<offset8*4>]{!}	op：LDC 或 STC。偏移量：0 到 1020 范围内 4 的倍数。	由协处理器定义	C2
加载和存储，零偏移量		<op>{2} <copr>, CRd, [Rn] {, 8-bit copro. option}	op：LDC 或 STC。	由协处理器定义	C2
加载和存储，后变址		<op>{2} <copr>, CRd, [Rn], #+/-<offset8*4>	op：LDC 或 STC。偏移量：0 到 1020 范围内 4 的倍数。	由协处理器定义	C2

其他运算		§	汇编程序	操作	注释
交换字			SWP Rd, Rm, [Rn]	temp := [Rn]，[Rn] := Rm，Rd := temp。	D
交换字节			SWPB Rd, Rm, [Rn]	temp := ZeroExtend([Rn][7:0])，[Rn][7:0] := Rm[7:0]，Rd := temp	D
存储返回状态		6	SRS{IA|IB|DA|DB} SP{!}, #<p_mode>	[SPm] := LR，[SPm + 4] := CPSR	C、I
从异常中返回		6	RFE{IA|IB|DA|DB} Rn{!}	PC := [Rn]，CPSR := [Rn + 4]	C、I
断点		5	BKPT <imm16>	预取中止*或*进入调试状态。指令中编码为 16 位的位域。	C、N
安全监控调用		Z	SMC <imm16>	安全监控调用异常。指令中编码为 16 位的位域。以前为 SMI。	
超级用户调用			SVC <imm24>	超级用户调用异常。指令中编码为 24 位的位域。以前为 SWI。	N
无操作		6	NOP	无操作，可能不花费任何时间。	N
提示	调试提示	7	DBG	向调试系统及其相关系统发送提示。	
	数据内存屏障	7	DMB	确保内存访问的观察顺序。	C
	数据同步屏障	7	DSB	确保内存访问完成。	C
	指令同步屏障	7	ISB	刷新处理器管道并跳转预测逻辑。	C
	设置事件	T2	SEV	向多处理器系统发送事件信号。如果不执行，则为 NOP。	N
	等待事件	T2	WFE	等待事件、IRQ、FIQ、不精确的中止或调试进入请求。如果不执行，则为 NOP。	N
	等待中断	T2	WFI	等待 IRQ、FIQ、不精确的中止或调试进入请求。如果不执行，则为 NOP。	N
	Yield	T2	YIELD	生成对其他线程的控制。如果不执行，则为 NOP。	N

注释

A	Thumb 状态下不可用。	**N**	在 Thumb-2 代码中，此指令的某些格式或所有格式为 16 位（窄）指令。有关详细信息，请参阅 *Thumb 16 位指令集 (UAL) 快速参考卡*。
B	在 Thumb 状态下可带有条件，且无须在 IT 块内。		
C	ARM 状态中不允许使用条件代码。	**P**	在 Thumb 状态下，此指令中的 Rn 可为 PC。
C2	备选格式 2 可用于 ARMv5 中。它可提供另一种备选运算。在 ARM 状态下，备选格式不允许使用条件代码。	**Q**	如果发生饱和（加法或减法）或溢出（乘法），则设置 Q 标记。使用 MRS 和 MSR 读取和重置 Q 标记。
D	已弃用。使用 LDREX 和 STREX 来代替。	**R**	在 ARM 指令中，<sh> 范围为 1-32。
G	根据各个运算的结果更新 CPSR 中的 4 个 GE 标记。	**S**	S 修饰符在 Thumb-2 指令中不可用。
I	IA 是缺省值，通常省略。	**T**	ARM 状态中不可用。
L	ARM：<imm8m>。16 位 Thumb：0-1020 范围内 4 的倍数。32 位 Thumb：0-4095。	**U**	不允许在 IT 块中使用。不允许在 ARM 或 Thumb 状态下使用条件代码。