转载 ARM指令英文全称及功能

2016-10-27 14:50:34 Strokess 阅读数 2587 更多

指令格式： 指令{条件}{S} {目的Register}，{OP1}，{OP2}			"{ }"中的内容可选。即，可以不带条件只有目的寄存器，或 只有目的寄存器和操作数1，也可以同时包含所有选项。“S” 决定指令的操作是否影响CPSR中条件标志位的值，当没有S时指令不更新CPSR中条件标志位的值
	助记符	英文全称	示例、功能
跳转指令	B	Branch 跳转指令	B Label ；程序无条件跳转到标号Label处执行
	BL	Branch with Link 带返回的跳转指令	BL Label ；当程序无条件跳转到标号Label处执行时，同时将当前的PC值保存到R14中
	BLX	Branch with Link and exchange带返回和状态切换的跳转指令	BLX Label；从ARM指令集跳转到指令中所指定的目标地址，并将处理器的工作状态有ARM状态切换到Thumb状态，该指令同时将PC的当前内容保存到寄存器R14中
	BX	Branch and exchange 带状态切换的跳转指令	BX Label；跳转到指令中所指定的目标地址，目标地址处的指令既可以是ARM指令，也可以是Thumb指令
数据处理	MOV	Move **数据传送**	MOV R1，R0，LSL＃3 ；将寄存器R0的值左移3位后传送到R1
	MVN	Move NOT **数据非传送**	MVN R0，＃0；将立即数0取反传送到寄存器R0中，完成后R0=-1
	CMP	Compare **比较指令**	CMP R1，R0 ；将寄存器R1的值与寄存器R0的值相减，并根据结果设置CPSR的标志位
	CMN	Compare negative **负数比较指令**	CMN R1，R0 ；将寄存器R1的值与寄存器R0的值相加，并根据结果设置CPSR的标志位
	TST	Test **位测试指令**	TST R1，＃0xffe；将寄存器R1的值与立即数0xffe按位与，并根据结果设置CPSR的标志位
	TEQ	Test equivalence **相等测试指令**	TEQ R1，R2；将寄存器R1的值与寄存器R2的值按位异或，并根据结果设置CPSR的标志位
	ADD	Add **加法运算指令**	ADD R0，R2，R3，LSL#1 ； R0 = R2 + (R3 << 1)
	ADC	Add with carry **带进位加法**	ADCS R2，R6，R10； R2 = R6 + R10 + !C，且更新CPSR的进位标志位
	SUB	Subtract **减法运算指令**	SUB R0，R1，#256 ； R0 = R1 – 256
	SBC	Subtract with carry **带进位减法指令**	SUBS R0，R1，R2； R0 = R1 - R2 - ！C，并根据结果设置CPSR的进位标志位
	RSB	Reverse subtract **逆向减法指令**	RSB R0，R1，R2； R0 = R2 – R1
	RSC	Reverse subtract with carry **带进位逆向减法指令**	RSC R0，R1，R2 ； R0 = R2 – R1 - ！C
	AND	And **逻辑与操作指令**	AND R0，R0，＃3； 该指令保持R0的0、1位，其余位清零。
	ORR	OR **逻辑或操作指令**	ORR R0，R0，＃3 ； 该指令设置R0的0、1位，其余位保持不变。
	EOR	Exclusive OR **逻辑异或操作指令**	EOR R0，R0，＃3； 该指令反转R0的0、1位，其余位保持不变。
	BIC	Bit clear **位清除指令**	BIC R0，R0，＃0b1011； 该指令清除 R0 中的位 0、1、和 3，其余的位保持不变。
	CLZ	Count left zero	计算操作数最高端0的个数

乘加指令	MUL	Multiply 32位乘法指令	MUL R0，R1，R2 ； R0 = R1 × R2
	MLA	Multiply and accumulate 32位乘加指令	MLAS R0，R1，R2，R3 ； R0 = R1 × R2 + R3，同时设置CPSR中的相关条件标志位
	SMULL	Signed multiply long 64位有符号数乘法指令	SMULL R0，R1，R2，R3 ； R0 = （R2 × R3）的低32位 R1 = （R2 × R3）的高32位
	SMLAL	Signed mul l and accumulate l 64位有符号数乘加指令	SMLAL R0，R1，R2，R3 ； R0 = （R2 × R3）的低32位 + R0; R1 = （R2 × R3）的高32位 + R1
	UMULL	Unsigned multiply long 64位无符号数乘法指令	UMULL R0，R1，R2，R3 ； R0 = （R2 × R3）的低32位; R1 = （R2 × R3）的高32位
	UMLAL	Unsigned mul&accumulate lon 64位无符号数乘法指令	UMLAL R0，R1，R2，R3 ； R0 = （R2 × R3）的低位 + R0; R1 = （R2 × R3）的高32位 + R1
PSR访问	MRS	Move PSR to register 程序状态寄存器到通用寄存器的数据传送指令	MRS R0，CPSR ； 传送CPSR的内容到R0
	MSR	Move register to PSR通用寄存器到程序状态寄存器的数据传送指令	MSR CPSR_c ，R0 ； 传送R0的内容到SPSR，但仅仅修改CPSR中的控制位域
加载/存储指令	LDR	Load word 字数据加载指令	LDR R0，[R1，R2]！ ； 将存储器地址为R1+R2的字数据读入R0，并将新地址R1 + R2写入R1。
	LDRB	Load byte 字节数据加载指令	LDRB R0，[R1，＃8] ； 将存储器地址为R1 + 8的字节数据读入R0，并将R0的高24位清零
	LDRH	Load half word 半字数据加载指令	LDRH R0，[R1] ； 将存储器地址为R1的半字数据读入寄存器R0，并将R0的高16位清零
	LDM	Load multiple 批量数据加载指令	LDMFD R13!，{R0，R4-R12，PC} ； 将堆栈内容恢复到寄存器（R0，R4到R12，LR）
	STR	Store 字数据存储指令	STR R0，[R1]，＃8 ； 将R0中的字数据写入R1为地址的存储器中，并将新地址R1 + 8写入R1
	STRB	Store byte 字节数据加载存储指令	STRB R0，[R1，＃8] ； 将寄存器R0中的字节数据写入以R1 + 8为地址的存储器中
	STRH	Store half word 半字数据存储指令	STRH R0，[R1，＃8] ； 将寄存器R0中的半字数据写入以R1 + 8为地址的存储器中
	STM	Store multiple 批量数据存储指令	STMFD R13!，{R0，R4-R12，LR} ； 将寄存器列表中的寄存器（R0，R4到R12，LR）存入堆栈
数据交换	SWP	Swap word 字数据交换指令	SWP R0，R1，[R2] ； R2所指的字数据传送到R0，同时R1的数据传送到R2所指的单元
	SWPB	Swap byte 字节数据交换指令	SWPB R0，R1，[R2] ； R2所指的字节数据传送到R0，R0高24位清零，同时R1低8位送R2所指单元。
移位指令	LSL	Logic shift left 逻辑左移操作	MOV R0, R1, LSL#2（ASL#2） ； 将R1中的内容左移两位后传送到R0中,低位用0填充
	ASL	Arithmetic shift left 算术左移操作	
	LSR	Logic shift right 逻辑右移操作	MOV R0, R1, LSR#2 ； 将R1中的内容右移两位后传送到R0中，左端用零来填充
	ASR	Arithmetic shift right 算术右移操作	MOV R0, R1, ASR#2 ； 将R1中的内容右移两位后传送到R0中，左端用第31位的值来填充
	ROR	Rotate right 循环右移操作	MOV R0, R1, ROR#2 ； 将R1中的内容循环右移两位后传送到R0中
	RRX	Rotate right extended 带拓展的循环右移操作	左端用进位标志位C来填充
协处	CDP	Data operations	协处理器数操作指令

理器	LDC	Load	协处理器数据加载指令
	STC	Store	协处理器数据存储指令
	MCR	Move to coproc fr ARM reg	处理器寄存器到协处理器寄存器的数据传送指令
	MRC	M to ARM reg fr coprocessor	协处理器寄存器到处理器寄存器的数据传送指令

PSR field	F (Flags field mask byte)								S (Stats field mask byte)								X (Extension field mask byte)								C (control field mask byte)							
CPSR	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
意义	N	Z	C	V	Q	DNZ (RAZ) 系统扩展用																			I	F	T	M4	M3	M2	M1	M

CPSR各位详细意义	N		当前指令运算结果为负时，N = 1; 结果为非负时，N = 0
	Z		运算结果为0，Z = 1；否则Z = 0
	C		上溢出、进位C = 1；下溢出、借位C = 0
	V		加减法V = 1表示符号位溢出
	I		I = 1时，禁止IRQ中断
	F		F = 1时，禁止FIQ中断
	T		T = 0，ARM指令；T = 1，Thumb指令
	M[4:0]	0b10000	User
		0b10001	FIQ
		0b10010	IRQ
		0b10011	Supervisor
		0b10111	Abort
		0b11011	Undefined
		0b11111	System

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令格式	Cond							Opcode				S	Rn				Rd				Shift_operand											

opcode		指令操作符编码
S		决定指令的操作是否影响CPSR的值
Rd		目标寄存器编码
Rn		包含第一个操作数的寄存器编码
Shift_oprand		表示第二个操作数
Cond		指令执行的条件编码，详细如下所示

条件	EQ	Z = 1	Equal
	NE	Z = 0	Not equal, or unordered

域					
	CS/HS	C = 1	Carry set / Unsigned higher or same	Great than or equal, or unordered	
	CC/LO	C = 0	Carry clear / Unsigned lower	Less than	
	MI	N = 1	Negative	Less than	
	PL	N = 0	Positive or zero	Greater than or equal , or unordered	
	VS	V = 1	Overflow	Unordered	
	VC	V = 0	No overflow	Not unordered	
	HI	C = 1且Z = 0	Unsigned higher	Greater than, or unordered	
	LS	C = 0或Z = 1	Unsigned lower or same	Less than or equal	
	GE	N = 1且V = 1 或N = 0且V = 0	Signed greater than or equal	Greater than or equal	
	LT	N = 1且V = 0 或N = 0且V = 1	Signed less than	Less than , or unordered	
	GT	Z = 0或N = V	Signed greater than	Great than	
	LE	Z = 1或N！ = V	Signed less than or equal	Less than or equal , or unordered	
	AL		Always (normally omitted)		
并行指令前缀	S		Signed arithmetic modulo 2^8 or 2^{16} ,sets CPSR GE bit		
	Q		Signed saturating arithmetic		
	SH		Signed arithmetic, halving results		
	U		Unsigned arithmetic modulo 2^8 or 2^{16} ,sets CPSR GE bit		
	UQ		Unsigned saturating arithmetic		
	UH		Unsigned arithmetic ,halving results		
批量传输地址模式	Block load / store		Stack pop / push		
	IA	Increment after	FD	Full descending	
	IB	Increment before	ED	Empty descending	
	DA	Decrement after	FA	Full ascending	
	DB	Decrement before	EA	Empty ascending	
ARM指令寻址方式	立即寻址	ADD R0，R0，＃0x3f	R0←R0 + 0x3f		
	寄存器寻址	ADD R0，R1，R2	R0←R1 + R2		
	间接寻址	ADD R0，R1，[R2]	R0←R1 + [R2]		
	变址寻址	LDR R0，[R1，＃4]	R0←[R1 + 4]		
		LDR R0，[R1，＃4]!	R0←[R1 + 4]、R1←R1 + 4		
		LDR R0，[R1]，＃4	R0←[R1]、R1←R1 + 4		
		LDR R0，[R1，R2]	R0←[R1 + R2]		
	多寄存器寻址	LDMIA R0，{R1，R2，R3，R4}	R1←[R0]；R2←[R0 + 4]；R3←[R0 + 8]；R4←[R0 + 12]		
伪指令及伪操作					
符号定义		GBLA / LCLA	定义一个全局 / 局部的数字变量，并初始化为0		
		GBLL / LCLL	定义一个全局 / 局部的逻辑变量，并初始化为F（假）		

		GBLS / LCLS	定义一个全局 / 局部的字符串变量，并初始化为空	
		SETA / SETL / SETS	给一个数学 / 逻辑 / 字符串变量赋值	
		RLIST	对一个通用寄存器列表定义名称，访问次序为根据寄存器的编号由低到高，与排列次序无关	
数据定义		DCB（=）/ DCW（DCWU）	分配一片连续的字节 / 半字存储单元并用指定的数据初始化	后缀U表示不要求对齐
		DCFS（DCFSU）/DCFD（DCFDU）	分配一片连续的（单 / 双精度的浮点数）字存储单元并用指定的数据初始化	
		DCQ（DCQU）/ DCD（DCDU）	用于分配一片以双字 / 字为单位的连续的存储单元并用指定的数据初始化	
		DCDO	分配字内存但愿，初始化为标号基于静态基址寄存器R9的偏移量	
		DCI	和DCD类似，不同处在于DCI内存中的数据被标识为指令	
		SPACE（%）	DataSpace SPACE 100； 分配连续100字节的存储单元并初始化为0	
		MAP（^）	MAP 0x100，R0 ； 定义结构化内存表首地址的值为0x100 + R0	
		FIELD（#）	A FIELD 16 ； 定义A的长度为16字节	

控制伪指令	IF、ELSE、ENDIF	IF 逻辑表达式
		指令序列1
		ELSE
		指令序列2
		ENDIF
			IF、ELSE、ENDIF伪指令能根据条件的成立与否决定是否执行某个指令序列。
			当IF后面的逻辑表达式为真，则执行指令序列1，否则执行指令序列2。其中，
			ELSE及指令序列2可以没有，此时，当IF后面的逻辑表达式为真，则执行
			指令序列1，否则继续执行后面的指令。	

	WHILE、WEND	WHILE 逻辑表达式 指令序列 WEND	WHILE、WEND伪指令能根据条件的成立与否决定是否循环执行某个指令序列。当WHILE后面的逻辑表达式为真，则执行指令序列，该指令序列执行完毕后，再判断逻辑表达式的值，若为真则继续执行，一直到逻辑表达式的值为假。	
	MACRO、MEND MEXIT	MACRO $标号 宏名 $参数1，$参数2，……指令序列 MEND	$标号在宏指令被展开时，标号会被替换为用户定义的符号， 宏指令可以使用一个或多个参数，当宏指令被展开时，这些参数被相应的值替换。 MEXIT用于从宏定义中跳转出去	
	AREA	AREA 段名 属性1，属性2，……	用于定义一个代码段或数据段。其中，段名若以数字开头，则该段名需用“|”括起来，如|1_test|。	
	ALIGN	AREA Init，CODE，ALIEN = 3	指定后面的指令为8字节对齐	
	CODE	CODE16、CODE32	指定指令序列为16位的Thumb指令或32位的ARM指令	
	ENTRY		在一个完整的汇编程序中至少要有一个ENTRY（也可以有多个，当有多个ENTRY时，程序的真正入口点由链接器指定），但在一个源文件里最多只能有一个ENTRY（可以没有）。	
	EQU（*）	名称 EQU 表达式 {，类型}	为程序中的常量、标号等定义一个等效的字符名称	
	EXPORT	EXPORT 标号	用于在声明一个全局的标号，该标号可在其他的文件中引用。EXPORT可用GLOBAL代替。	
	IMPORT	IMPORT 标号	用于通知编译器要使用的标号在其他的源文件中定义，无论当前源文件是否引用该标号，该标号均会被加入到当前源文件的符号表中	
	EXTERN	EXTERN 标号	用于通知编译器要使用的标号在其他的源文件中定义，但要在当前源文件中引用，如果当前源文件实际并未引用该标号，该标号就不会被加入到当前源文件的符号表中	
	GET	GET 文件名	将一个源文件包含到当前的源文件中，并将被包含的源文件在当前位置进行汇编处理	
	INCBIN	INCBIN 文件名	INCBIN伪指令用于将一个目标文件或数据文件包含到当前的源文件中，被包含的文件不作任何变动的存放在当前文件中，编译器从其后开始继续处理	
	RN	名称 RN 表达式	RN伪指令用于给一个寄存器定义一个别名	
	ROUT	{名称} ROUT	ROUT伪指令用于给一个局部变量定义作用范围。在程序中未使用该伪指令时，局部变量的作用范围为所在的AREA，而使用ROUT后，局部变量的作为范围为当前ROUT和下一个ROUT之间。	