第一部分：ARM 机器指令集（硬件执行）

1. 数据传送指令（MOV/MVN）

用于在寄存器之间传送数据，或者将立即数加载到寄存器。

指令	示例代码	解释
MOV	MOV R0, #5	将立即数 5 传送到寄存器 R0
	MOV R0, R1	将 R1 的值传送到 R0
	MOV R0, R1, LSL #5	将 R1 的值逻辑左移 5 位后传送到 R0
MVN	MVN R0, #0	将 0 取反（即-1，0xFFFFFFFF）传送到 R0

2. 算术运算指令（ADD/SUB 等）

用于加减乘运算。

指令	示例代码	解释
ADD	ADD R0, R1, #5	R0 = R1 + 5
	ADD R0, R1, R2	R0 = R1 + R2
ADC	ADC R1, R3, R5	带进位加法：R1 = R3 + R5 + Carry（常用于 64 位加法的高位运算）
SUB	SUB R0, R1, #5	R0 = R1 - 5
SBC	SBC R1, R3, R5	带借位减法：R1 = R3 - R5 - !Carry（常用于 64 位减法的高位运算）
RSB	RSB R0, R1, #5	反向减法：R0 = 5 - R1
RSC	RSC R1, R5, R3	带借位反向减法：R1 = R3 - R5 - !Carry
MUL	MULS R0, R1, R2	32 位乘法：R0 = R1 × R2，S 表示结果影响标志位
MLA	MLA R0, R1, R2, R3	乘加：R0 = (R1 × R2) + R3
SMULL	SMULL R0, R1, R2, R3	64 位有符号乘法：R1(高 32 位):R0(低 32 位) = R2 × R3
SMLAL	SMLAL R0, R1, R2, R3	64 位有符号乘加：R1:R0 = R2 × R3 + R1:R0
UMULL	UMULL R0, R1, R2, R3	64 位无符号乘法：R1:R0 = R2 × R3
UMLAL	UMLAL R0, R1, R2, R3	64 位无符号乘加：R1:R0 = R2 × R3 + R1:R0

3. 逻辑运算指令（AND/ORR 等）

用于位操作（置位、清零、取反）。

指令	示例代码	解释
AND	AND R0, R0, #5	逻辑与：保留 R0 的第 0、2 位，其余清零（掩码功能）
ORR	ORR R0, R0, #5	逻辑或：将 R0 的第 0、2 位置 1，其余不变
EOR	EOR R0, R0, #5	逻辑异或：将 R0 的第 0、2 位取反，其余不变
BIC	BIC R0, R0, #5	位清除：将 R0 的第 0、2 位清零（#5 为掩码），其余不变

4. 比较与测试指令（CMP/TST 等）

不保存运算结果，只更新 CPSR 中的标志位（N, Z, C, V）。

指令	示例代码	解释
CMP	CMP R0, #5	比较 R0 和 5（做减法 R0 - 5），更新标志位
CMN	CMN R0, #5	负值比较（做加法 R0 + 5），实际是比较 R0 和-5
TST	TST R0, #5	位测试（做逻辑与 R0 & 5），测试第 0、2 位是否为 1
TEQ	TEQ R0, #5	相等测试（做异或 R0 ^ 5），判断 R0 是否等于 5

5. 跳转指令（Branch）

用于控制程序流程。

指令	示例代码	解释
B	B exit	无条件跳转到标号 exit 处
	BNE loop	条件跳转：如果不相等（Z=0），则跳转到 loop
BL	BL func	跳转到子程序 func，并将返回地址保存到 LR (R14)
BX	BX R0	跳转到 R0 指定的地址，并根据 R0 最低位切换状态（ARM/Thumb）
BLX	BLX T16	带返回的跳转，同时切换到 Thumb 状态

6. 存储器访问指令（Load/Store）

用于在寄存器和内存之间传输数据。

单寄存器传输：

指令	示例代码	解释
LDR	LDR R0, [R1]	将 R1 指向的内存地址中的字数据（4 字节）加载到 R0
	LDR R0, [R1, #5]	变址寻址：将地址（R1+5）处的数据加载到 R0
STR	STR R0, [R1]	将 R0 中的字数据存储到 R1 指向的内存地址
	STR R2, [R1, #16]	将 R2 的数据存储到（R1+16）的地址
LDRB	LDRB R0, [R1]	加载一个字节（8 位）到 R0，高位清零
STRB	STRB R0, [R1, #4]!	将 R0 低 8 位存储到（R1+4），并将新地址写回 R1
LDRH	LDRH R0, [R1]	加载半字（16 位）到 R0
LDRSB	LDRSB R0, [R1]	加载有符号字节（带符号扩展）

多寄存器传输（常用于堆栈）：

指令	示例代码	解释
LDM	LDMIA R0, {R1-R5}	以 R0 为基址，依次加载数据到 R1 至 R5，IA 表示地址后增

指令	示例代码	解释
	LDMFD R13!, {R0-R4}	出栈操作（满递减堆栈），恢复 R0-R4，并更新 SP
STM	STMFD R13!, {R0-R4}	入栈操作（满递减堆栈），保存 R0-R4，并更新 SP

数据交换（用于信号量）:

指令	示例代码	解释
SWP	SWP R0, R1, [R2]	交换字：[R2]→R0，同时 R1→[R2]
SWPB	SWPB R0, R1, [R2]	交换字节：[R2]字节→R0 低 8 位，同时 R1 低 8 位→[R2]

7. 程序状态寄存器与协处理器指令

指令	示例代码	解释
MRS	MRS R0, CPSR	将状态寄存器 CPSR 的值读取到 R0
MSR	MSR CPSR_f, R0	将 R0 的值写入 CPSR 的标志域
CDP	CDP P1, 2, C1, C2, C3	协处理器数据操作：通知 P1 执行操作
LDC	LDC P3, C4, [R5]	协处理器加载：从 R5 指向的内存加载数据到协处理器寄存器 C4
STC	STCEQ P2, C4, [R5]	协处理器存储：将 C4 数据存入 R5 指向的内存（带条件 EQ）
MCR	MCR P2, 3, R2, C4, C5, 6	ARM 寄存器(R2)传送到协处理器(C4, C5)
MRC	MRC P0, 3, R2, C4, C5, 6	协处理器(C4, C5)传送到 ARM 寄存器(R2)

8. 异常中断指令

指令	示例代码	解释
SWI	SWI 0x05	产生软件中断，调用 5 号系统服务
BKPT	BKPT 0xF02C	断点中断，用于调试，立即数为断点信息

第二部分：汇编伪指令（汇编器处理）

1. 符号定义与赋值

伪指令	示例代码	解释
GBLA	GBLA num1	定义一个全局数字变量 num1
LCLA	LCLA num1	定义一个局部数字变量 num1
SETA	num1 SETA 0x1234	给数字变量 num1 赋值
SETS	str3 SETS "Hello!"	给字符串变量 str3 赋值
RLIST	pblock RLIST {R0-R3}	给寄存器列表定义名称 pblock
EQU	num1 EQU 1234	定义常量 num1 为 1234 (类似 C 语言 #define)

2. 数据定义（内存分配）

伪指令	示例代码	解释
DCB	str1 DCB "Welcome!"	分配字节空间并初始化字符串
DCW	Arrayw1 DCW 0xa, 0xc	分配半字（2 字节）空间
DCD	Arrayd1 DCD 1334	分配字（4 字节）空间
SPACE	freespace SPACE 1000	分配 1000 字节的连续存储区，初始化为 0
MAP	MAP 0xF10000	定义结构化内存表的首地址
FIELD	count FIELD 4	定义结构化内存表中的数据域长度

3. 汇编控制（宏与条件）

伪指令	示例代码	解释
MACRO	MACRO … MEND	定义宏的开始和结束
IF/ELSE	IF R0=0x10 … ENDIF	条件编译，根据逻辑表达式决定是否编译某段代码
WHILE	WHILE Coul < 10 …	条件循环编译

4. 其他通用伪指令

伪指令	示例代码	解释
AREA	AREA codesec, CODE…	定义一个段（代码段或数据段）
ALIGN	ALIGN 4	使当前位置地址字对齐（4 字节对齐）
ENTRY	ENTRY	指定程序的入口点
END	END	告诉编译器源程序结束
EXPORT	EXPORT main	声明全局标号，供外部文件调用
IMPORT	IMPORT _printf	引用外部文件定义的标号
GET	GET prog1.s	包含另一个源文件（类似 #include）
INCBIN	INCBIN data1.dat	包含二进制数据文件

5. ARM 相关特定伪指令（会被替换为机器指令）

伪指令	示例代码	解释
ADR	ADR R2, LOOP	小范围地址读取（编译器替换为 SUB R2, PC, offset）
ADRL	ADRL R4, start+60000	中等范围地址读取（编译器替换为两条指令）
LDR	LDR R1, =0xFFF	大范围读取或加载立即数。如果立即数太大，编译器会将其放入文字池，然后用 PC 相对寻址读取
LTORG	LTORG	声明文字池（Literal Pool）的开始位置，通常放在无条件跳转后，用于存放 LDR 伪指令的大立即数

伪指令	示例代码	解释
NOP	NOP	空操作（不执行任何操作，仅占用时间）

第三部分：ARM 指令的后缀与条件码

1. 条件码后缀 {cond}

几乎所有的 ARM 指令都可以根据 CPSR 寄存器中的标志位（N, Z, C, V）有条件地执行。
如果不加后缀，默认是 AL (Always，无条件执行)。

后缀	含义	检查的标志位逻辑	典型应用场景
EQ	相等 (Equal)	Z = 1	CMP R0, R1 后，若相等则执行
NE	不相等 (Not Equal)	Z = 0	CMP R0, R1 后，若不相等则执行
CS/HS	进位/无符号大于等于	C = 1	无符号数比较 $\boldsymbol{R0 \geq R1}$
CC/LO	无进位/无符号小于	C = 0	无符号数比较 $\boldsymbol{R0 < R1}$
MI	负数 (Minus)	N = 1	判断结果是否为负
PL	正数或零 (Plus)	N = 0	判断结果是否为正或零
VS	溢出 (Overflow Set)	V = 1	有符号运算产生溢出
VC	无溢出 (Overflow Clear)	V = 0	有符号运算未溢出
HI	无符号大于 (Higher)	C = 1 且 Z = 0	无符号数比较 $\boldsymbol{R0 > R1}$
LS	无符号小于等于 (Lower or Same)	C = 0 或 Z = 1	无符号数比较 $\boldsymbol{R0 \leq R1}$
GE	有符号大于等于 (Greater or Equal)	N == V	有符号数比较 $\boldsymbol{R0 \geq R1}$
LT	有符号小于 (Less Than)	N != V	有符号数比较 $\boldsymbol{R0 < R1}$
GT	有符号大于 (Greater Than)	Z = 0 且 N == V	有符号数比较 $\boldsymbol{R0 > R1}$
LE	有符号小于等于 (Less or Equal)	Z = 1 或 N != V	有符号数比较 $\boldsymbol{R0 \leq R1}$
AL	无条件执行 (Always)	(忽略)	默认行为，通常省略不写

2. 状态更新后缀 {S}

该后缀决定指令执行后是否刷新 CPSR 寄存器中的条件标志位（N, Z, C, V）。

变体	说明	示例
无后缀	仅做运算，不影响标志位	ADD R0, R1, R2 (R0=R1+R2, Flags 不变)
S	运算并更新 N, Z, C, V	ADDS R0, R1, R2 (R0=R1+R2, Flags 更新)

特殊说明:

比较指令（CMP, CMN, TST, TEQ）不需要加 S，**默认且强制**更新标志位，且不保存运算结果。

运算指令（ADD, SUB, MOV 等）需要显式加 S 才会更新标志位。

3. 数据宽度/类型后缀 (Load/Store 专用)

默认的 LDR/STR 操作的是 **32 位字 (Word)**。通过后缀可以操作不同宽度的数据。

后缀	数据类型	宽度	说明	示例
无	Word	32 位	字传输	LDR R0, [R1]
B	Byte	8 位	无符号字节（高 24 位清零）	LDRB R0, [R1]
H	Halfword	16 位	无符号半字（高 16 位清零）	LDRH R0, [R1]
SB	Signed Byte	8 位	有符号字节（高 24 位做符号扩展）	LDRSB R0, [R1]
SH	Signed Halfword	16 位	有符号半字（高 16 位做符号扩展）	LDRSH R0, [R1]
T	User Mode	32 位	强制使用用户模式权限访问内存	LDRT R0, [R1]

4. 批量传输模式后缀 (LDM/STM 专用)

用于决定基址寄存器（地址指针）在传输前还是传输后增加/减少。

分为**数据块传输**和**堆栈操作**两套命名，但硬件对应关系是一样的。

A. 数据块传输模式 (地址变化)

后缀	全称	含义	算法描述
IA	Increment After	传送**后**地址加	地址向上增长(常用)
IB	Increment Before	传送**前**地址加	地址向上增长
DA	Decrement After	传送**后**地址减	地址向下增长
DB	Decrement Before	传送**前**地址减	地址向下增长

B. 堆栈操作模式 (栈指针行为)

ARM 标准堆栈通常使用 **FD (Full Descending)**。

后缀	全称	堆栈类型	对应数据块模式
FD	Full Descending	**满递减** (指针指数据，向下生长)	LDMFD = LDMIA / STMFD = STMDB
ED	Empty Descending	**空递减** (指针指空位，向下生长)	LDMED = LDMIB / STMED = STMDA

后缀	全称	堆栈类型	对应数据块模式
FA	Full Ascending	**满递增**（指针指数据，向上生长）	LDMFA = LDMDA / STMFA = STMIB
EA	Empty Ascending	**空递增**（指针指空位，向上生长）	LDMEA = LDMDB / STMEA = STMIA

5. 移位操作后缀（作为操作数）

作为**第 2 个操作数**的修饰符出现。

助记符	含义	效果	示例
LSL	Logic Shift Left	逻辑左移（低位补 0）	MOV R0, R1, LSL #2
LSR	Logic Shift Right	逻辑右移（高位补 0）	MOV R0, R1, LSR #4
ASR	Arithmetic Shift Right	算术右移（符号位保持）	MOV R0, R1, ASR #4
ROR	Rotate Right	循环右移	MOV R0, R1, ROR #4
RRX	Rotate Right with Extend	带扩展循环右移（带 C 位）	MOV R0, R1, RRX