

Instruction <Condition Field> {S} Rd, Rn, < shifter_operand >

Mnemonic	Instruction	Action
ADC	Add with carry	Rd := Rn + Op2 + Carry
ADD	Add	Rd := Rn + Op2
AND	AND	Rd := Rn AND Op2
B	Branch	R15 := address
BIC	Bit Clear	Rd := Rn AND NOT Op2
BL	Branch with Link	R14 = R15, R15 := address
BX	Branch and Exchange	R15 := Rn, T bit := Rn[0]
CDP	Coprocessor Data Processing	(Coprocessor-specific)
CMN	Compare Negative	CPSR flags := Rn + Op2
CMP	Compare	CPSR flags := Rn - Op2
EOR	Exclusive OR	Rd := (Rn AND NOT Op2)
LDC	Load coprocessor from memory	Coprocessor load
LDM	Load multiple registers	Stack manipulation (Pop)
LDR	Load register from memory	Rd := (address)
MCR	Move CPU register to coprocessor register	cRn := rRn {<op>cRm}
MLA	Multiply Accumulate	Rd := (Rm x Rs) + Rn
MOV	Move register or constant	Rd := Op2
MRC	Move from coprocessor register to CPU register	Rn := cRn {<op>cRm}
MRS	Move PSR status/flags to register	Rn := PSR
MSR	Move register to PSR status/flags	PSR := Rm
MUL	Multiply	Rd := Rm x Rs
MVN	Move negative register	Rd := 0 x FFFFFFFF EOR Op2
ORR	OR	Rd := Rn OR Op2
RSB	Reverse Subtract	Rd := Op2 - Rn
RSC	Reverse Subtract with Carry	Rd := Op2 - Rn - 1 + Carry
SBC	Subtract with Carry	Rd := Rn - Op2 - 1 + Carry
STC	Store coprocessor register to memory	address := CRn
STM	Store Multiple	Stack manipulation (Push)
STR	Store register to memory	<address> := Rd
SUB	Subtract	Rd := Rn - Op2
SWI	Software Interrupt	OS call
SWP	Swap register with memory	Rd := [Rn], [Rn] := Rm
TEQ	Test bitwise equality	CPSR flags := Rn EOR Op2
TST	Test bits	CPSR flags := Rn AND Op2

ADR/ADRL	Loading Address / Long Address into register	same as LDR Rd, =(address)
LDR	[H SH SB]	[Halfword SignedHalfword Signed Byte]
STR	[H]	[Halfword] --only storing UNSIGNED data
LDR STR	R0, [R1, #offset_12/8 R2] +12 with R0, [R1, #offset_12/8 R2] ! [H..]+8 R0, [R1], #offset_12/8 R2	Run(R1+offset) R1 += offset --> Run(R1) Run(R1) --> R1 += offset
LDM STM	[IA IB DA DB] [FD ED FA EA]	I = Increment ; D = Decrement ; A = After E = Empty ; F = Full ; B = Before
STM FA	r13!, {r0-r5}; Push onto a Full Ascending Stack	ascending stack: from low -> higher address descending stack: from high -> lower address SP can point to the last used location (full stack) or the first unused location (empty stack)
LDM FA	r13!, {r0-r5}; Pop from a Full Ascending Stack	
STM FD	r13!, {r0-r5}; Push onto a Full Descending Stack	
LDM FD	r13!, {r0-r5}; Pop from a Full Descending Stack	
STM EA	r13!, {r0-r5}; Push onto an Empty Ascending Stack	
LDM EA	r13!, {r0-r5}; Pop from an Empty Ascending Stack	
STM ED	r13!, {r0-r5}; Push onto Empty Descending Stack	<-- example: r13 is a (stack pointer)
LDM ED	r13!, {r0-r5}; Pop from an Empty Descending Stack	

Normal multiply (produce 32-bit result):		
MUL	MUL<<<{S} Rd, Rm, Rs	Rd := Rm x Rs
MLA	MLA<<<{S} Rd, Rm, Rs, Rn	Rd := (Rm x Rs) + Rn
Long Unsigned Multiply (produce 64-bit result):		
UMULL	UMULL<<<{S} RdLo, RmHi, Rm, Rs UMULL r6, r8, r0, r1	[RmHi, RdLo] = Rm x Rs r8, r6 = r0 x r1
Unsigned Multiply Accumulate long(produce 64-bit result):		
UMLAL	UMLAL<<<{S} RdLo, RmHi, Rm, Rs UMLAL r5, r8, r0, r1	[RmHi, RdLo] = [RmHi, RdLo] + Rm x Rs Action: [r8, r5] = [r8, r5] + r0 x r1
SMULL Signed Multiply Long(produce 64-bit result):		
SMULL	SMULL<<<{S} RdLo, RmHi, Rm, Rs	[RmHi, RdLo] = Rm x Rs
SMLAL Signed Multiply Accumulate Long(produce 64-bit result):		
SMLAL	SMLAL<<<{S} RdLo, RmHi, Rm, Rs	[RmHi, RdLo] = [RmHi, RdLo] + Rm x Rs

bit-31-> [N,Z,C,V] [...] [...] [...] [...] [...] [I,F,T,M4] [M3,M2,M1,M0] -> bit-0		
Move PSR (CPSR or SPSR) to general register:		
MRS<<< Rd, CPSR		Rd = CPSR
MRS<<< Rd, SPSR		Rd = SPSR
Move to CPSR or SPSR from register or an immediate operand:		
MSR<<< CPSR, (#<immediate> Rm)		CPSR = #<immediate> Rm
MSR<<< SPSR, (#<immediate> Rm)		SPSR = #<immediate> Rm
Example: For pre-setting condition flags in user mode.		
MRS R0, CPSR		; Read the CPSR
BIC R0, R0, #0xF0000000		; Clear the N, Z, C and V bits
MSR CPSR, R0		; Update the flag bits in the CPSR [N, Z, C, V] ; flags now all clear
Example: For pre-setting modes, interrupt bits and state bit in privileged mode.		
MRS R0, CPSR		; Read the CPSR
ORR R0, R0, #0x80		; Set the interrupt disable bit
MSR CPSR, R0		; Update the control bits in the CPSR ; interrupts (IRQ) now disabled

Code	Suffix	Flags	Meaning
0	EQ	Z set	equal
1	NE	Z clear	not equal
10	CS	C set	unsigned higher or same
11	CC	C clear	unsigned lower
100	MI	N set	negative
101	PL	N clear	positive or zero
110	VS	V set	overflow
111	VC	V clear	no overflow
1000	HI	C set and Z clear	unsigned higher
1001	LS	C clear or Z set	unsigned lower or same
1010	GE	N equals V	greater or equal
1011	LT	N not equal to V	less than
1100	GT	Z clear AND (N equals V)	greater than
1101	LE	Z set OR (N not equal to V)	less than or equal
1110	AL	(ignored)	always

LSL	shift left by n bits (multiplication by 2 ⁿ)
LSR	logical shift right by n bits (unsigned division by 2 ⁿ)
ASR	arithmetic shift right by n bits. The bits fed into the top end of the operand are copies of the original top—or sign—bit. sign bit shifted in (signed division by 2 ⁿ)
ROR	rotate right by n bits
RRX	rotate right extended by 1 bit. This is a 33-bit rotate, where the 33 rd bit is the PSR Carry flag

Conventionally, a SP (stack pointer) always points to the top of the stack. The stack grows downward (in terms of memory addresses).

Adding a data to the stack represents a **push** operation.
To **push** a data onto the stack, first **subtract 4** from the stack pointer, then store the data at the address in the stack pointer.

Removing an item from a stack is called a **pop** operation.
To **pop** a data from a stack, copy the data pointed at by the SP, then **add 4** to the SP.

Note that the data size of the stack is 4-byte for ARM processor.

BX	Branch and Exchange	BX{cond} Rn
If bit0 of Rn = 1, switch to THUMB state. If bit0 of Rn = 0, switch to ARM state. If Rn = R15 (i.e., PC), the behavior is undefined.		

CLZ	Count Leading Zeros	CLZ<CC> Rd, Rm
It counts the number of binary zero bits before the first binary 1 bit in Rm and store the result in Rd. (only for certain ARM architectures)		

Swap a byte or word quantity between a register and external memory		
<SWP>{cond}{B} Rd,Rm,[Rn]		
SWP R0,R1,[R2] :Load R0 with the word stored at an addressed indicated by R2, and store the content in R1 at address indicated by R2.		
SWPB R2,R3,[R4] :Load R2 with the byte addressed by R4, and store bits 0 to 7 of R3 at address indicated by R4.		
If Rd and Rm are the same register, this instruction swaps the value of the register and the value at the memory address.		

SWI	Software Interrupt	SWI<CC> expression
item	now	save location
PC	0x08	R14_svc
CPSR	new	SPSR_svc
		MOVS PC,R14_svc