Referencia rápida de instrucciones ARMv7

Aritméticas	Suma	**add**{s}<c> {<Rd>,} <Rn>, <Rm> {,<shift>}	*Rd(n)* := *Rn* + *Rm*{shifted}	NZCV
		adc{s}<c> {<Rd>,} <Rn>, <Rm> {,<shift>}	*Rd(n)* := *Rn* + Rm{shifted} + C	NZCV
		add{s}<c> {<Rd>,} <Rn>, #<const>	Rd(n) := Rn + const	NZCV
		adc{s}<c> {<Rd>,} <Rn>, #<const>	*Rd(n)* := *Rn* + *const* + C	NZCV
	Resta	**sub**{s}<c> {<Rd>,} <Rn>, <Rm> {,<shift>}	*Rd(n)* := *Rn* - *Rm*{shifted}	NZCV
		sbc{s}<c> {<Rd>,} <Rn>, <Rm>{,<shift>}	*Rd(n)* := *Rn* - *Rm*{shifted} - not (C)	NZCV
		rsb{s}<c> {<Rd>,} <Rn>, <Rm> {,<shift>}	*Rd(n)* := Rm{shifted} - *Rn*	NZCV
		sub{s}<c> {<Rd>,} <Rn>, #<const>	*Rd(n)* := *Rn* - *const*	NZCV
		sbc{s}<c> {<Rd>,} <Rn>, #<const>	*Rd(n)* := *Rn* - *const* - not (C)	NZCV
		rsb{s}<c> {<Rd>,} <Rn>,#<const>	*Rd(n)* := *const* - *Rn*	NZCV
	Multiplicación	**mul**<c> {<Rd>,} <Rn>, <Rm>	*Rd(n)* := (*Rn***Rm*)	
		mla<c> <Rd>, <Rn>, <Rm>, <Ra>	*Rd* := *Ra* + (*Rn***Rm*)	
		mls<c> <Rd>, <Rn>, <Rm>, <Ra>	*Rd* := *Ra* -(*Rn***Rm*)	
	División	**udiv**<c> <Rd>, <Rn>, <Rm>	*Rd* := unsigned_32_bit (*Rn*/*Rm*); rounded towards 0	
		sdiv<c> <Rd>, <Rn>, <Rm>	*Rd* := signed_32_bit (*Rn*/*Rm*); rounded towards 0	
Bit operation	Lógicas	**and**{s}<c> {<Rd>,} <Rn>, <Rm> {,<shift>}	*Rd(n)* := *Rn* & *Rm*{shifted}	NZCV
		bic{s}<c> {<Rd>,} <Rn>, <Rm> {,<shift>}	*Rd(n)* := *Rn* &~ *Rm*{shifted}	NZCV
		orr{s}<c> {<Rd>,} <Rn>, <Rm> {,<shift>}	*Rd(n)* := *Rn* | *Rm*{shifted}	NZCV
		orn{s}<c> {<Rd>,} <Rn>, <Rm> {,<shift>}	*Rd(n)* := *Rn* |~ *Rm*{shifted}	NZCV
		eor{s}<c> {<Rd>,} <Rn>, <Rm> {,<shift>}	*Rd(n)* := *Rn* ⊕ *Rm*{shifted}	NZCV
		and{s}<c> {<Rd>,} <Rn>, #<const>	*Rd(n)* := *Rn* & *const*	NZCV
		bic{s}<c> {<Rd>,} <Rn>, #<const>	*Rd(n)* := *Rn* &~ *const*	NZCV
		orr{s}<c> {<Rd>,} <Rn>, #<const>	*Rd(n)* := *Rn* | *const*	NZCV
		orn{s}<c> {<Rd>,} <Rn>, #<const>	*Rd(n)* := *Rn* |~ *const*	NZCV
		eor{s}<c> {<Rd>,} <Rn>, #<const>	*Rd(n)* := *Rn* ⊕ *const*	NZCV
	Comparar	**cmp**<c> <Rn>, <Rm> {,<shift>}	*Rn* - *Rm*{shifted}	NZCV
		cmn<c> <Rn>, <Rm> {,<shift>}	*Rn* + *Rm*{shifted}	NZCV
		tst<c> <Rn>, <Rm> {,<shift>}	*Rn* & *Rm*{shifted}	NZCV
		teq<c> <Rn>, <Rm> {,<shift>}	*Rn* ⊕ *Rm*{shifted}	NZCV
Movimientos registros	Move	**mov**{s}<c> <Rd>, <Rm>	*Rd* := *Rm*	NZ
		mov{s}<c> <Rd>, #<const>	*Rd* := *const*	NZC
	Desplazar	**lsr**{s}<c> <Rd>, <Rm>, #<n>	*Rd* := *Rm*{shifted-right by <n>}; filled with 0's, C := last shifted-out	NZC
		lsr{s}<c> <Rd>, <Rm>,<Rs>	*Rd* := *Rm*{shifted-right by Rs}; filled with 0's, C := last shifted-out	NZC
		asr{s}<c> <Rd>, <Rm>,#<n>	*Rd* := *Rm*{shifted-right by <n>}; filled with MSB[2], C := last shifted-out	NZC
		asr{s}<c> <Rd>, <Rm>,<Rs>	*Rd* := *Rm*{shifted-right by Rs}; filled with MSB[2], C := last shifted-out	NZC
		lsl{s}<c> <Rd>, <Rm>,#<n>	*Rd* := *Rm*{shifted-left by <n>}; filled with 0's, C := last shifted-out	NZC
		lsl{s}<c> <Rd>, <Rm>,<Rs>	*Rd* := *Rm*{shifted-left by Rs}; filled with 0's, C := last shifted-out	NZC
Load & Store	Offset	**ldr**<c> <Rd>, [<Rb> {, #+/-<offset>}]	*Rd* := [*Rb* ± *offset*]	
		str<c> <Rs>, [<Rb> {, #+/-<offset>}]	[*Rb* ± *offset*] := *Rs*	
	Pre-offset	**ldr**<c> <Rd>, [<Rb>, #+/-<offset>]!	*Rb* := *Rb* ± *offset; Rd* := [*Rb*];	
		str<c> <Rs>, [<Rb>, #+/-<offset>]!	*Rb* := *Rb* ± *offset;* [*Rb*] := *Rs;*	
	Post-offset	**ldr**<c> <Rd>, [<Rb>], #+/-<offset>	*Rd* := [*Rb*]; *Rb* := *Rb* ± *offset*	
		str<c> <Rs>, [<Rb>], #+/-<offset>	[*Rb*] := *Rs; Rb* := *Rb* ! ± *offset*	
	Indexada	**ldr**<c> <Rd>, [<Rb>, <Ri> {, **lsl** #<shift>}]	*Rd* := [*Rb* + *Ri*{shifted- left}]	
		str<c> <Rs>, [<Rb>, <Ri> {, **lsl** #<shift>}]	[*Rb* + *Ri*{shifted- left}] := *Rs*	
	Literal	**ldr**<c> <Rd>, =<label>	*Rd* := [label]	
		ldr<c> <Rd>, [PC, #+/-<offset>]	*Rd* := [*PC* ! *offset*]	
	Pila	**pop** {<lista de registros>}	Forma resumida de **ldm sp!**, <lista de registros>	
		push {<lista de registros>}	Forma resumida de **stmdb sp!**, <lista de registros>	
Saltos		**b**<c> <label>	if c then *PC* := *label*	
		bl<c> <label>	if c then *LR* := *PC_next*; *PC* := *label*	
		bx<c> <Rm>	if c then *PC* := *Rm*	

Códigos condición		
<c>	Significa	Flags
eq	Equal	Z = 1
ne	Not equal	Z = 0
cs, hs	Carry set, Unsigned higher or same	C = 1
cc, lo	Carry clear, Unsigned lower	C = 0
mi	Minus, Negative	N = 1
pl	Plus, Positive or zero	N = 0
vs	Overflow	V = 1
vc	No overflow	V = 0
hi	Unsigned higher	C = 1 & Z = 0
ls	Unsigned lower or same	C = 0 | Z = 1
ge	Signed greater or equal	N = V
lt	Signed less	N ≠ V
gt	Signed greater	Z = 0 & N = V
le	Signed less or equal	Z = 1 & N ≠ V
al	Always	cualquiera

Shift options	
<shift>	Significa
<omit>	no shifts or rotations, equivalent to LSL #0
LSL #<n>	logical shift left by <*n*> bits, 0 # *n* # 31
LSR #<n>	logical shift right by <*n*> bits, 1 # *n* # 32
ASR #<n>	arithmetic shift right by <*n*> bits, 1 # *n* #32
ROR #<n>	rotate right by <*n*> bits, 1 # *n* # 31
RRX	rotate right by 1 bit through carry flag

Flags		
Flag	significa	Calculado como
N	Negative	MSB[2] (Result) 1
Z	Zero	Result 0
C	Carry	Depends on instruction
V	Overflow	Signed overflow
Q	Saturated	Signed overflow (result saturated)