ARMv7 Quick Reference Guide
West Valley College
Saratoga, California, 2019

General Registers AAPCS

R0-R3		Arguments/Paramters and return values (OK to Use)
R4-R7		General purpose registers (must be Saved/Restored)
R8-R10		General purpose registers (must be Saved/Restored)
R11	FP	Frame Pointer
R12	IP	Intra-procedure-call scratch register
R13	SP	Stack pointer
R14	LR	Return address
R15	PC	Program counter

Assembly Directives for Program Labels

loop:	Label All Labels end with ':'
MyLabel:	Label Labels are case sensitive

Assembly Directives for Program Comments

//	Line Comment	Ignore all characters to end of line
@	Line Comment	Ignore all characters to end of line
/* . . . */	Block Comment	Ignore all characters between /* and */

Assembly Directives for Control of Listing File

.title	"My Program"	Title for each page
.sbttl	"Some Function"	Subtitle for each page
.psize	50,100	Create Landscape orientation listing file
.eject		Skip to Next Page

Assembly Storage Directives

.DATA		Define the start of the data section
.TEXT		Define the start of the code section
.BALIGN	4, 0x20	Add Bytes to get to 4 byte boundary if needed. Optional value, in this case fill with 0x20.
.BYTE	0x12	Allocate a byte with Hex 12
.HWORD	0x3456	Allocate 2 bytes with Hex 5635
.WORD	0x6789ABCD	Allocate 4 bytes with Hex CDAB8967
.QUAD	0x2345	Allocate 8 bytes with Hex 452300...00
.OCTA	0x6789	Allocate 16 bytes with Hex 896700...00
.FLOAT	3.141592654	Allocate 4 bytes with 7 digits of π
.SINGLE	3.141592654	Allocate 4 bytes with 7 digits of π
.DOUBLE	3.141592654...	Allocate 8 bytes with 15 digits of π
.ASCII	"ABC"	ASCII Encode String as 0x414243
.ASCIZ	"ABC"	.ASCII with Zero byte added at end
.STRING	"ABC	Same as .ASCIZ
.ORG	. + 0x100	Fill 0x100 byte with 0x00
.SKIP	64	Skips 64 bytes and fills each with 0x00
.FILL	16, 2 0x20	Fills 16 bytes with 0x0020 repeated

C Functions

putchar	Input	R0$_{7:0}$ Is the ACSII Character to display
puts	Input	R0 is the Address of the ASCIZ string to display
printf	Input	R0 is the Address of the ASCIZ format string R1-3 holds the optional 1st-3rd value in the format string %[parameter] [flags] [width] [.precision] [length] type
scanf	Input	R0 is the Address of the ASCIZ format string R1-3 is the Address for the 1st-3rd value in the format string
Format	String	%c Char, %d %i Int, %f - Double, %x %X Hex, %s - String

Bitwise and Move Instructions

AND{S}	Rd, Rn, op2	Rd = Rn & op2	
ASR{S}	Rd, Rn, #shift$_5$	Rd = Rn $\bar{\gg}$ shift	
ASR{S}	Rd, Rn, Rs	Rd = Rn $\bar{\gg}$ Rs	
BFC	Rd, #p, #n	Rd$_{p+n-1:p}$ = 0$_n$	6t
BFI	Rd, Rn, #p, #n	Rd$_{p+n-1:p}$ = Rn$_{n-1:0}$	6t
BIC{S}	Rd, Rn, op2	Rd = Rn & ∼op2	
CLZ	Rd, Rn	Rd = Count Leading Zeros in Rn	
EOR{S}	Rd, Rn, op2	Rd = Rn $\oplus$ op2	
LSL{S}	Rd, Rn, #shift$_5$	Rd = Rn $\ll$ shift	
LSL{S}	Rd, Rn, Rs	Rd = Rn $\ll$ Rs	
LSR{S}	Rd, Rn, #shift$_5$	Rd = Rn $\gg$ shift	
LSR{S}	Rd, Rn, Rs	Rd = Rn $\gg$ Rs	
MOV{S}	Rd, op2	Rd = op2	
MOVT	Rd, #i$_{16}$	Rd$_{31:16}$ = i	6t
MOVW	Rd, #i$_{16}$	Rd = i$^{\emptyset}$	6t
MVN{S}	Rd, op2	Rd = ∼op2	
ORR{S}	Rd, Rn, op2	Rd = Rn | op2	
RBIT	Rd, Rn	Rd = ReverseBits(Rn)	6t
REV	Rd, Rn	Rd = Rn$_{B0}$:Rn$_{B1}$:Rn$_{B2}$:Rn$_{B3}$	6
REV16	Rd, Rn	Rd = Rn$_{B2}$:Rn$_{B3}$:Rn$_{B0}$:Rn$_{B1}$	6
REVSH	Rd, Rn	Rd = Rn$^{\pm}_{B0}$:Rn$_{B1}$	6
ROR{S}	Rd, Rn, #shift$_5$	Rd = Rn $\ggg$ shift	
ROR{S}	Rd, Rn, Rs	Rd = Rn $\ggg$ Rs	
RRX{S}	Rd, Rn	Rd = C:Rn$_{31:1}$; C = Rn$_0$	
SBFX	Rd, Rn, #p, #n	Rd = Rn$^{\pm}_{p+n-1:p}$	6t
TEQ	Rd, op2	Rd $\oplus$ op2	
TST	Rd, op2	Rd & op2	
UBFX	Rd, Rn, #p, #n	Rd = Rn$^{\emptyset}_{p+n-1:p}$	6t

Operand 2

#i$_{32}$	i$_8$ $\ggg$ i$_4$:0 (aka ROR 4 bits *2)
Rm	Rm (same as Rm, LSL #0)
Rm, LSL #n	Rm $\ll$ {1..31}
Rm, LSR #n	Rm $\gg$ {1..32}
Rm, ASR #n	Rm $\bar{\gg}$ {1..32}
Rm, ROR #n	Rm $\ggg$ {1..31}
Rm, RRX	C:Rm$_{31:1}$; C = Rm$_0$ (C is Carry Flag)
Rm, LSL Rs	Rm $\ll$ Rs
Rm, LSR Rs	Rm $\gg$ Rs
Rm, ASR Rs	Rm $\bar{\gg}$ Rs
Rm, ROR Rs	Rm $\ggg$ Rs

Load and Store Addressing Modes

MOV{S}	Rd, #0b000111	Rd = 000111$_2$	
LDR	Rt, =someLabel	Rt = address of someLabel Load 32 bit Address from Literal Pool	P
LDR	Rt, [Rn]	Rt = Mem[addr in Rn]	
LDR	Rt, [Rn, #4]	Rt = Mem[Rn+ 4]	
LDR	Rt, [Rn], #+4	Post:Rt= Mem[Rn^{Orig}];Rn = Rn+4	
LDR	Rt, [Rn, Rm]	Rt = Mem[Rn + Rm]	
LDR	Rt, [Rn, Rm, LSL #3]	Rt = Mem[Rn + Rm$\ll$3]	
STR	Rt, [Rn]	Mem[addr in Rn] = Rt	
STR	Rt, [Rn, #4]	Mem[Rn+ 4] = Rt	
STR	Rt, [Rn, #-4]!	Pre:Rn= Rn+4;Mem[Rn^{New}] = Rt	
STR	Rt, [Rn, Rm]	Mem[Rn + Rm] = Rt	
STR	Rt, [Rn, Rm, LSR #2]	Mem[Rn + Rm$\gg$2] = Rt	

Arithmetic Instructions

ADC{S}	Rd, Rn, op2	Rd = Rn + op2 + C (Carry)	
ADD{S}	Rd, Rn, op2	Rd = Rn + op2	
ADR	Rd, ±rel$_{12}$	Rd = PC + rel$^{\pm}$	
CMN	Rd, op2	Rd + op2	
CMP	Rd, op2	Rd − op2	
MUL{S}	Rd, Rn, Rm	Rd = Rn × Rm	B
QADD	Rd, Rm, Rn	Rd = SATS(Rm + Rn, 32)	D
QDADD	Rd, Rm, Rn	Rd = SATS(Rm + SATS(2×Rn, 32), 32)	D
QDSUB	Rd, Rm, Rn	Rd = SATS(Rm − SATS(2×Rn, 32), 32)	D
QSUB	Rd, Rm, Rn	Rd = SATS(Rm − Rn, 32)	D
RSB{S}	Rd, Rn, op2	Rd = op2 − Rn	
RSC{S}	Rd, Rn, op2	Rd = op2 − (Rn + C (Carry))	
SBC{S}	Rd, Rn, op2	Rd = Rn − (op2 + C (Carry))	
SDIV	Rd, Rn, Rm	Rd = Rn $\bar{\div}$ Rm	7
SSAT	Rd, #st$_5$, Rn{slr}	Rd = SATS(Rn $\ll$ $\bar{\gg}$ sh, st)$^{\pm}$	6
SSAT16	Rd, #sat$_4$, Rn	Rd = SATS(Rn$^{\pm}_{H1}$, sat)$^{\pm}$:SATS(Rn$^{\pm}_{H0}$, sat)$^{\pm}$	6D
SUB{S}	Rd, Rn, op2	Rd = Rn − op2	
UDIV	Rd, Rn, Rm	Rd = Rn ÷ Rm	7
USAT	Rd, #st$_5$, Rn{slr}	Rd = SATU(Rn $\ll$ $\bar{\gg}$ sh, st)$^{\pm}$	6
USAT16	Rd, #sat$_4$, Rn	Rd = SATU(Rn$^{\pm}_{H1}$, sat)$^{\pm}$:SATU(Rn$^{\pm}_{H0}$, i)$^{\pm}$	6D

Branch and Jump Instructions

B	rel$_{26}$	PC = PC + rel$^{\pm}_{25:2}$:0$_{1:0}$	
Bcc	rel$_{21}$	if(cc) PC = PC + rel$^{\pm}_{20:1}$:0	I
BKPT	#i$_{16}$	BreakPoint(i)	I
BL	rel$_{26}$	LR=PC$_{31:1}$:0; PC+=rel$^{\pm}_{25:2}$:0$_{1:0}$	
BLX	rel$_{26}$	LR=PC$_{31:1}$:0; Set=1; PC+=rel$^{\pm}_{25:1}$:0	
BLX	Rm	LR=PC$_{31:1}$:0; Set=Rm$_0$; PC=Rm$_{31:1}$:0	
BX	Rm	Set = Rm$_0$; PC = Rm$_{31:1}$:0	

Condition Codes (cc)

EQ	Equal	Z
NE	Not equal	!Z
CS/HS	Carry set, Unsigned higher or same	C
CC/LO	Carry clear, Unsigned lower	!C
MI	Minus, Negative	N
PL	Plus, Positive or zero	!N
VS	Overflow	V
VC	No overflow	!V
HI	Unsigned higher	C & !Z
LS	Unsigned lower or same	!C | Z
GE	Signed greater than or equal	N = V
LT	Signed less than	N ≠ V
GT	Signed greater than	!Z & N = V
LE	Signed less than or equal	Z | N ≠ V
AL	Always (default)	1

Keys

{S}	Optional suffix, if present update flags
op2	Immediate or shifted register

ACSII Character Set

NUL 00	DLE 10	Space 20	0 30	@ 40	P 50	‘ 60	p 70
SOH 01	DC1 11	! 21	1 31	A 41	Q 51	a 61	q 71
STX 02	DC2 12	" 22	2 32	B 42	R 52	b 62	r 72
STX 03	DC3 13	# 23	3 33	C 43	S 53	c 63	s 73
EOT 04	DC4 14	$ 24	4 34	D 44	T 54	d 64	t 74
ENQ 05	NAK 15	% 25	5 35	E 45	U 55	e 65	u 75
ACK 06	SYN 16	& 26	6 36	F 46	V 56	f 66	v 76
BEL 07	ETB 17	' 27	7 37	G 47	W 57	g 67	w 77
BS 08	CAN 18	(28	8 38	H 48	X 58	h 68	x 78
TAB 09	EM 19	) 29	9 39	I 49	Y 59	i 69	y 79
LF 0A	SUB 1A	* 2A	: 3A	J 4A	Z 5A	j 6A	z 7A
VT 0B	ESC 1B	+ 2B	; 3B	K 4B	[5B	k 6B	{ 7B
FF 0C	FS 1C	, 2C	< 3C	L 4C	\ 5C	l 6C	| 7C
CR 0D	GS 1D	- 2D	= 3D	M 4D	] 5D	m 6D	} 7D
SO 0E	RS 1E	. 2E	> 3E	N 4E	^ 5E	n 6E	~ 7E
SI 0F	US 1F	/ 2F	? 3F	O 4F	_ 5F	o 6F	DEL 7F

Powers|Hexadecimal

N=	Hex	Binary	M$_{16}$=	iB	2^N and 16^M	Notes
0	0	0000	0		1	
1	1	0001			2	
2	2	0010			4	
3	3	0011			8	
4	4	0100	1		16	
5	5	0101			32	
6	6	0110			64	
7	7	0111			128	
8	8	1000	2		256	
9	9	1001			512	
10	A	1010		KiB	1 024	
11	B	1011			2 048	
12	C	1100	3		4 096	
13	D	1101			8 192	
13	E	1110			16 383	
15	F	1111			32 768	
16	10	1 0000	4		65 536	
20	14	1 0100	5	MiB	1 048 576	
24	18	1 1000	6		16 777 216	
28	1C	1 1100	7		268 435 456	
30	1E	1 1110		GiB	1 073 741 824	
32	20	10 0000	8		4 294 967 296	
36	24	10 0100	9		68 719 476 736	
40	28	10 1000	10	TiB	1 099 511 627 776	

GNU Debugger (GDB) Commands

list	*line number*	Show Program source code listing, optional line #
break	line number	Set a break point at a specific line number
delete	break number	Delete a break point at a specific line number
run	*arguments*	Run the program with optional cmd line arguments
continue	finish	Continue (or finish function/loop) after breakpoint
step	next	Step runs one instruction / next will enter function
info	b, f, r, s	Display breakpoint, register, fp registers or stack
print/*f*	*(type)* myVar	Display w/type of (int),(float) /f=> x-hex, f-flt, t-bin
eXamine	*/nfu* Address	Display value at address/n=> Number of elements to display f=> String, Instruction, heX / u=>size in byte, halfword, Word
quit	help	Quit GDB / Get Help on a GDB command

ARMv7 Floating-Point Instruction Quick Reference Guid

Load and Store Instructions

LDMDA	Rn{!}, rlist	rlist = [Rn−4×cnt+4]; if(!) Rn−=4×cnt	
LDMDB	Rn{!}, rlist	rlist = [Rn − 4×cnt]; if(!) Rn−=4×cnt	
LDM*IA*	Rn{!}, rlist	rlist = [Rn]; if(!) Rn += 4×cnt	
LDMIB	Rn{!}, rlist	rlist = [Rn + 4]; if(!) Rn += 4×cnt	
LDR{T}	Rt, [addr]	Rt = [addr]	
LDRB{T}	Rt, [addr]	Rt = $[addr]^{\emptyset}_{8}$	
LDRD	Rt, ry, [addr]	ry:Rt = [addr]	
LDRH{T}	Rt, [addr]	Rt = $[addr]^{\emptyset}_{16}$	
LDRSB{T}	Rt, [addr]	Rt = $[addr]^{\pm}_{8}$	
LDRSH{T}	Rt, [addr]	Rt = $[addr]^{\pm}_{16}$	
POP	rlist	rlist = [SP]; SP += 4×cnt	
PUSH	rlist	SP −= 4×cnt; [SP] = rlist	
STMDA	Rn{!}, rlist	[Rn−4×cnt+4] = rlist; if(!) Rn−=4×cnt	
STMDB	Rn{!}, rlist	[Rn − 4×cnt] = rlist; if(!) Rn−=4×cnt	
STM*IA*	Rn{!}, rlist	[Rn] = rlist; if(!) Rn += 4×cnt	
STMIB	Rn{!}, rlist	[Rn+4] = rlist; if(!) Rn += 4×cnt	
STR{T}	Rt, [addr]	[addr] = Rt	
STRB{T}	Rt, [addr]	$[addr]_8$ = $Rt_{7:0}$	
STRD	Rt, ry, [addr]	[addr] = ry:Rt	
STRH{T}	Rt, [addr]	$[addr]_{16}$ = $Rt_{15:0}$	

ARM LDR/STR Addressing Modes

non-T	[Rn{, #±i_8}]{!}	addr = Rn + $i^{\pm}$; if(!) Rn = addr
xxR{,B}	[Rn{, #±i_{12}}]{!}	addr = Rn + $i^{\pm}$; if(!) Rn = addr
any	[Rn]{, #±i_8}	addr = Rn; Rn += $i^{\pm}$
xxR{,B}{T}	[Rn], #±i_{12}	addr = Rn; Rn += $i^{\pm}$
non-T	[Rn, ±Rm]{!}	addr = Rn ± Rm; if(!) Rn = addr
xxR{,B}	[Rn, ±Rm{AS}]{!}	addr = Rn ± AS(Rm); if(!) Rn = addr
any	[Rn], ±Rm	addr = Rn; Rn ±= Rm
xxR{,B}{T}	[Rn], ±Rm{AS}	addr = Rn; Rn ±= AS(Rm)
LD non-T	±rel_8	addr = PC + $rel^{\pm}$
LDR{,B}	±rel_{12}	addr = PC + $rel^{\pm}$

Special Instructions

DBG	#i_4	DebugHint(i)	7
DMB	option	DataMemoryBarrier(option)	I,7
DSB	option	DataSynchronizationBarrier(option)	I,7
ISB	SY	InstructionSynchronizationBarrier(SY	I,7
NOP			6k
PLD{W}	[addr]	PreloadData(addr)	
PLI	[addr]	PreloadInstr(addr)	7
SETEND	{BE/LE}	EndianState = {BE/LE}	I,6
SEV		SendEvent()	6k
SVC	#i_{24}	CallSupervisor(i)	
UDF	#i_{16}	UndefinedException(i)	
WFE		WaitForEvent()	6k
WFI		WaitForInterrupt()	6k
YIELD		HintYield()	6k

Multiplication Instructions

MLA	Rd, Rn, Rm, Ra	Rd = Ra + Rn × Rm	
MLA{S}	Rd, Rn, Rm, Ra	Rd = Ra + Rn × Rm	
MLS	Rd, Rn, Rm, Ra	Rd = Ra − Rn × Rm	6t
MUL	Rd, Rn, Rm	Rd = Rn × Rm	
MUL{S}	Rd, Rn, Rm	Rd = Rn × Rm	
SMLAxy	Rd, Rn, Rm, Ra	Rd = Ra + $Rn^{\pm}_{Hx} \bar{\times} Rm^{\pm}_{Hy}$	D
SMLaD	Rd, Rn, Rm, Ra	Rd = Ra + $Rn^{\pm}_{H0} \bar{\times} Rm^{\pm}_{H0} \pm Rn^{\pm}_{H1} \bar{\times} Rm^{\pm}_{H1}$	6,D
SMLaDX	Rd, Rn, Rm, Ra	Rd = Ra + $Rn^{\pm}_{H0} \bar{\times} Rm^{\pm}_{H1}$ ±xyz $Rn^{\pm}_{H1} \bar{\times} Rm^{\pm}_{H0}$	D
SMLaLD	Rd, Rn, Rm, Ra	Rn:Rd += $Rm^{\pm}_{H0} \bar{\times} Ra^{\pm}_{H0} \pm Rm^{\pm}_{H1} \bar{\times} Ra^{\pm}_{H1}$	6,D
SMLaLDX	Rd, Rn, Rm, Ra	Rn:Rd += $Rm^{\pm}_{H0} \bar{\times} Ra^{\pm}_{H1} \pm Rm^{\pm}_{H1} \bar{\times} Ra^{\pm}_{H0}$	D
SMLAL	Rd, Rn, Rm, Ra	Rn:Rd += Rm $\bar{\times}$ Ra	
SMLAL{S}	Rd, Rn, Rm, Ra	Rn:Rd += Rm $\bar{\times}$ Ra	
SMLALxy	Rd, Rn, Rm, Ra	Rn:Rd += $Rm^{\pm}_{Hx} \bar{\times} Ra^{\pm}_{Hy}$	D
SMLAWy	Rd, Rn, Rm, Ra	Rd = Ra + Rn $\bar{\times}$ $Rm^{\pm}_{Hy}$	D
SMMLa	Rd, Rn, Rm, Ra	Rd = Ra ± $(Rn \bar{\times} Rm)_{63:32}$	6,D
SMMLaR	Rd, Rn, Rm, Ra	Rd = Ra ± $(Rn \bar{\times} Rm + 0x80000000)_{63:32}$	D
SMMUL	Rd, Rn, Rm	Rd = $(Rn \bar{\times} Rm)_{63:32}$	6,D
SMMULR	Rd, Rn, Rm	Rd = $(Rn \bar{\times} Rm + 0x80000000)_{63:32}$	D
SMUaD	Rd, Rn, Rm	Rd = $Rn^{\pm}_{H0} \bar{\times} Rm^{\pm}_{H0} \pm Rn^{\pm}_{H1} \bar{\times} Rm^{\pm}_{H1}$	6,D
SMUaDX	Rd, Rn, Rm	Rd = $Rn^{\pm}_{H0} \bar{\times} Rm^{\pm}_{H1} \pm Rn^{\pm}_{H1} \bar{\times} Rm^{\pm}_{H0}$	D
SMULxy	Rd, Rn, Rm	Rd = $Rn^{\pm}_{Hx} \bar{\times} Rm^{\pm}_{Hy}$	D
SMULL	Rd, Rn, Rm, Ra	Rn:Rd = Rm $\bar{\times}$ Ra	
SMULL{S}	Rd, Rn, Rm, Ra	Rn:Rd = Rm $\bar{\times}$ Ra	
SMULWy	Rd, Rn, Rm	Rd = $(Rn \bar{\times} Rm^{\pm}_{Hy})_{47:16}$	D
UMAAL	Rd, Rn, Rm, Ra	Rn:Rd = Rn + Rd + Rm × Ra	D
UMLAL	Rd, Rn, Rm, Ra	Rn:Rd += Rm × Ra	
UMULL	Rd, Rn, Rm, Ra	Rn:Rd = Rm × Ra	

Exclusive Load and Store Instructions

CLREX		ClearExclusiveLocal()	I,6k
LDREX	Rt, [Rn]	Rt = [Rn]; SetExcluMonitor	6k
LDREX	Rt, [Rn, #i_{10}]	Rt = [Rn+$i^{\emptyset}_{9:2}$:$0_{1:0}$]; SetExcluMonitor	6k
LDREXB	Rt, [Rn]	Rt = $[Rn]_{7:0}$; SetExclusiveMonitor	6k
LDREXD	Rt, Rt2, [Rn]	Rt2:Rt = [Rn]; SetExclusiveMonitor	6k
LDREXH	Rt, [Rn]	Rt = $[Rn]_{15:0}$; SetExclusiveMonitor	6k
STREX	Rd,Rt,[Rn]	if(Pass) [Rn]=Rt; Rd = Pass ? 1 : 0	6k
STREX	Rd,Rt,[Rn,#i_{10}]	if(Pass)[Rn+$i^{\emptyset}_{9:2}$:$0_{1:0}$]=Rt; Rd=Pass?1:0	6k
STREXB	Rd,Rt,[Rn]	if(Pass) $[Rn]_8$= $Rt_{7:0}$; Rd = Pass?1:0	6k
STREXD	Rd,Rt,Rt2,[Rn]	if(Pass) [Rn]= Rt2:Rt; Rd=Pass?1:0	6k
STREXH	Rd,Rt,[Rn]	if(Pass) $[Rn]_{16}$=$Rt_{15:0}$; Rd=Pass?1:0	6k

Notes for Instruction Set

6,6k,6t,7	Introduced in ARMv6, ARMv6k, ARMv6T2, or ARMv7
B	Techinclly a Multiple Groups Instruction
D	Not available on ARM-M without DSP extension
I	Can't be conditional
P	Pseudo Instruction

Floating-Point Register Transfer Instructions

VMOV.F32	Sd, #±immFlt	Sd = ±immediate FP Value	
VMOV.F64	Dd, #±immFlt	Sd = ±immediate FP Value	
VMOV.F32	Sd, Sm	Sd = Sm	
VMOV.F32	Dd, Dm	Dd = Dm	
VMOV	Rd, Sn	Rd = Sn Copy Only, Not converted	
VMOV	Sn, Rd	Sn = Rnd Copy Only, Not converted	
VMOV	Dm, Rd, Rn	Dm = Rn:Rd Copy Only, Not converted	
VMOV	Rd, Rn, Dm	Rn:Rd =Dm Copy Only, Not converted	
VMOV	Sm, Sm1, Rd, Rn	Sm=Rd, Sm1=Rn Copy Only, Not	
VMOV	Rd, Rn, Sm, Sm1	Rd=Sm, Rn=Sm1 Copy Only, Not	
VMOV{.32}	Dn[x], Rd	if(x==0)$Dn_{31:0}$= Rd else $Dn_{63:32}$= Rd	
VMOV{.32}	Rd, Dn[x]	if(x==0)Rd= $Dn_{31:0}$ else Rd= $Dn_{63:32}$	
VMRS	Rd, fpreg	Rd = fpreg	
VMRS	APSR_nzcv,FPSCR	APSR_nzcv = FPSCR_nzcv	
VMSR	fpreg, Rd	fpreg = Rd	

Floating-Point Instructions

VABS.f	fx, fy	fx = $\lvert fy \rvert$ Absolute Value	
VADD.f	fx, fy, fz	fx = fy + fz	
VCMP{E}.f	fd, #0.0	FPSCR_nzcv = Compare(fx, 0)	
VCMP{E}.f	fx, fy	FPSCR_nzcv = Compare(fx, fy)	
VCVT.F32.F64	Sx, Dy	Sx = Float2Float(Dy) Converts	
VCVT.F64.F32	Dx, Sy	Dx = Float2Float(Sy) Converts	
VCVT.sz.f	fx, fy, #i_5	fx = $Float2Fixed(fy, i)^s$ Converts	V3
VCVT.f.s32	fx, sy	fx = Int2Float(sy^s) Converts	
VCVT.f.sz	fx, fy, #i_5	fx = Fixed2Float(fy^s, i) Converts	V3
VCVT{R}.s32.f	Sx, fy	Sx = $Float2Int(fy)^s$ Converts	
VCVTx.F16.F32	Sx, Sy	Sx_{Hx} = Float2Float(Sy) Converts	V3
VCVTx.F32.F16	Sx, Sy	Sx = Float2Float(Sy_{Hx}) Converts	V3
VDIV.f	fx, fy, fz	fx = fy ÷ dz	
VFMa.f	fx, fy, fz	fx ±= fy × fz	V4
VFMNa.f	fx, fy, fz	fx = −fx ± fy × fz	V4
VMLa.f	fx, fy, fz	fx ±= ⌊fy × fz⌋	
VMUL.f	fx, fy, fz	fx = fy × fz	
VNEG.f	fx, fy	fx = −fy	
VNMLa.f	fx, fy, fz	fx = −fx ± −⌊fy × fz⌋	
VNMUL.f	fx, fy, fz	fx = −⌊fy × fz⌋	
VSQRT.f	fx, fy	fx = SQRT(fy)	
VSUB.f	fx, fy, fz	fx = fy − fz	

SIMD Register Transfer Instructions

VDUP.z	mx, ry	mx_{z*} = ry_z	
VDUP.z	mx, dy[i]	mx_{z*} = dy_{zi}	
VMOV.z	dx[i], ry	dx_{zi} = ry_z	
VMOV.sz	rx, dy[i]	rx = dy^{s}_{zi}	
VMOV	mx, my	mx = my	

Floating-point System Registers

FPEXC	Floating-point Exception Control
FPSCR	Floating-point Status and Control
FPSID	Floating-point System ID
MVFR{0..1}	Media and VFP Feature {0..1}

Floating-point Status and Control Register (FPSCR)

IOC	0x00000001	Invalid Operation cumulative exception
DZC	0x00000002	Division by Zero cumulative exception
OFC	0x00000004	Overflow cumulative exception
UFC	0x00000008	Underflow cumulative exception
IXC	0x00000010	Inexact cumulative exception
IDC	0x00000080	Input Denormal cumulative exception
IOE	0x00000100	Invalid Operation exception trap enable
DZE	0x00000200	Division by Zero exception trap enable
OFE	0x00000400	Overflow exception trap enable
UFE	0x00000800	Underflow exception trap enable
IXE	0x00001000	Inexact exception trap enable
IDE	0x00008000	Input Denormal exception trap enable
RMode	0x00c00000	Rounding mode (RN,RP,RM,RZ)
FZ	0x01000000	Flush-to-zero mode
DN	0x02000000	Default NaN mode
AHP	0x04000000	Alternative half-precision
QC	0x08000000	Cumulative saturation
V	0x10000000	Overflow condition flag
C	0x20000000	Carry condition flag
Z	0x40000000	Zero condition flag
N	0x80000000	Negative condition flag

Floating-point and SIMD Keys

s	Operation signess (S or U)
z	Data size (8, 16, or 32)
f	Floating-point size (F32 or F64)
sx, dx, qx	Singleword/doubleword/quadword register
fx, fy, fz	Floating-point register (Sx or Dx depending on data size)
mx, my, mz	SIMD register (Dx or Qx)
cm	SIMD comparison operator (GE, GT, LE, or LT)
⌊x⌋	Value is rounded
x ≪≫ y	(y < 0) ? (x ≫ −y) : (x ≪ y)
∧ ∨	Maximum/minimum value

Notes for Floating-point and SIMD Instructions

V3,V4	Introduced in VFPv3, VFPv4
SH,S2	Introduced in SIMD with half-precision floats, SIMDv2
F	Instruction with data type F32 also exists

Keys

{T}	LDR/STR instruction uses user privileges.
a	A or S to add or subtract operand.
di	DA, DB, IA or IB for decrease/increase before/after.
i, j	Immediate operand, range 0..max / 1..max+1
rx, ry, rz, rw	General register
rlist	Comma separated list of registers within { }.
SAT{S,U}(x,b)	Saturated signed/unsigned b bit value
{rb}	Optional rotate (ROR 8, ROR 16 or ROR 24)
{slr}	Optional shift (LSL #{1..31} or ASR #{1..32})
{sl}	Optional left shift (LSL #{1..31})
{sr}	Optional right shift (ASR #{1..32})
{AS}	ARM shift or rotate (LSL/ROR #{1..31}, LSR/ASR #{1..32} or RRX)
$value^{\pm}$, $value^{\emptyset}$	Value is sign/zero extended
$\bar{\times}$ $\bar{\div}$ $\bar{\gg}$	Operation is signed