

### Faculty of Engineering, Mathematics and Science School of Computer Science & Statistics

Integrated Engineering Year 3 Examination

Trinity Term 2016

3D2 Microprocessor Systems 2

13 May 2016

**RDS Main Hall** 

14:00 - 16:00

Dr Mike Brady

### Instructions to Candidates:

You may not start this examination until you are instructed to do so by the Invigilator.

Attempt two questions.

All questions carry equal marks. Each question is marked out of a total of 20 marks.

### Materials permitted for this examination:

An ASCII code table (one page) and an ARM Instruction Set Summary (six pages) accompany this examination paper.

Non-programmable calculators are permitted for this examination — please indicate the make and model of your calculator on each answer book used.

- (a) Write a fragment of ARM assembly code to form the sum of 1,000 integers stored from location ARRAY upwards and to store the result at location SUM. How would you deal with arithmetic overflow? [8 marks]
  - (b) Give an account of what a *cache* is and how caches are organised and managed. [6 marks]
  - (c) Assuming a three-stage pipeline (fetch-decode-execute), a 1 nanosecond processor clock, 10 nanosecond main memory and a cache that can be accessed by the processor instantly, estimate the amount of time it would take to execute the main loop of your code fragment above. Give an account of any problems estimating the time.

    [6 marks]

- (a) List the different modes available in the ARM processor. What are they for, and how does the ARM processor move between them? Write a fragment of code to put the processor into the user mode.
   [6 marks]
  - (b) Write an interrupt handler that is called by a quartz crystal-controller clock interrupt every 0.1641417 milliseconds to maintain a seconds counter in location SECONDS. Your code must introduce no extra inaccuracies to the time by making any approximations. [14 marks]

- 3. (a) Explain exactly what the *context* of a program is. How does an interrupt handler (or other exception handler) preserve the context of programs when it interrupts a program?
  [4 marks]
  - (b) Write a fragment of an interrupt handler to save the entire register and CSPR context of a user mode program it has interrupted on the program's own stack. [16 marks]

### **ASCII** Code

		ın Numb	oer					
Row Number	000	001	010	011	100	101	110	111
0000	NUL	DLE	<b>◊</b>	0	@	P		p
0001	SOH	DCI	1	1	Α	Q	a	q
0010	STX	DC2	11	2	В	R	b	r
0011	ETX	DC3	#	3	С	S	c	s
0100	EOT	DC4	\$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	${f f}$	v
0111	BELL	ETB	ľ	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	X
1001	HT	EM	)	9	I	Y	i	у
1010	LF	SUB	*	:	J	Z	j	Z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	I	
1101	CR	GS	-	=	M	]	m	}
1110	so	RS	•	>	N	٨	n	~
1111	SI	US	1	?	О	_	0	DEL

The ASCII code of a character is found by combining its Column Number (given in 3-bit binary)

with its Row Number (given in 4-bit binary).
The Column Number forms bits 6, 5 and 4 of the ASCII, and the Row Number forms bits 3, 2, 1 and 0 of the ASCII.

Example of use: to get ASCII code for letter "n", locate it in Column 110, Row 1110. Hence its ASCII code is 1101110.

The **Control Code** mnemonics are given in italics above; e.g. CR for Carriage Return, LF for Line Feed, BELL for the Bell, DEL for Delete.

The Space is ASCII 0100000, and is shown as ◊ here.

{cond}
<Operand2>
<fields>
<PSR>
{S}
C\*, V\*

Key to Tables

ReXi is Re rotated 16 bits if X present. Otherwise, ReXi is Re.   Refer to Table Prefixes for Parallel Instructions   Refer to Table Profixes for Parallel Instructions   Re	<immed 8r=""></immed>	A 32-bit constant, formed by right	A 32-bit constant, formed by right-rotating an 8-bit value by an even number of bits.		_	<u>;</u> _;			Updates base register after data transfer if ! present.
Refer to Table Processor Modes	{ <u>x</u> }	RsX is Rs rotated 16 bits if X pres	ent. Otherwise, RsX is Rs.	_	+	-			+ or (+ may be omitted.)
Refer to Table Processor modes   Ridge   Rid	<prefix></prefix>	Refer to Table Prefixes for Pari	allel instructions		507				Refer to Table ARM architecture versions.
R13 for the processor mode specified by ep_modes   R24 for the processor mode specified by ep_modes   R25 for the processor   R	<p_mode></p_mode>	Refer to Table Processor Mode	is		_	i£1	3gg:	•	Interrupt flags. One or more of a, i, f (abort, interrupt, fast interrupt).
S   Assembler   S   Lasembler   S   Lasembler   S   Lasembler   S   Lasembler   S   Labol   Corol.] {6} Rd, Rn, <0perand2>   N   Z   C   V   Rd = Rn + Operand2   Corol.] {6} Rd, Rn, <0perand2>   N   Z   C   V   Rd = Rn + Operand2   Corol.] {6} Rd, Rn, Rn   Coperand2>   N   Z   C   V   Rd = Rn + Operand2   Corol.] {6} Rd, Rn, Rn   Rn, <0perand2>   N   Z   C   V   Rd = Rn + Operand2   Corol.] {6} Rd, Rn, Rn   Rn, <0perand2>   N   Z   C   V   Rd = Rn + Operand2   Corol.] {6} Rd, Rn, Rn   Rn, <0perand2>   N   Z   C   V   Rd = Rn - Operand2   Corol.] {6} Rd, Rn, Rn   Rn, <0perand2>   N   Z   C   V   Rd = Rn - Operand2   Rd   Rn, Rn   Rn   Rn   Rn   Rn   Rn   Rn	R13m	R13 for the processor mode specif	ied by <p_mode></p_mode>	L		~			Rounds result to nearest if R present, otherwise truncates result.
with carry with carry with carry sunrating surrating substact with carry substact with carry severes subtract with carry surrating sunrating substact with carry substact with carry surrating sunrating su	Operation			S F	date	ŭ.			ction
Mance   Connal   Sal   Rad, Ran, And   Ran   Ran	Arithmetic Ad		ADD(cond)(S) Rd, Rn,	z	7	`[	4	ㅠ	d := Rn + Operand2
SE   QADD    cond  Rd., Rm., Rn   Rn   Rn   Rn   Rn   Rn   Rn   Rn		with carry	Rd, Rn,	z	Z		<	<b>-</b>	id := Rn + Operand2 + Carry
DADDY   Cond			QADD(cond) Rd, Rm, R						d := SAT(Rm + Rn)
SUB Cond S S Rd, Rn, <operand2></operand2>	_		5E QDADD {cond} Rd, Rm, Rn						id := SAT(Rm + SAT(Rn * 2))
SBC(cond) {S} Rd, Rn, <operand2>   N</operand2>	Su	btract	SUB{cond}{s} Rd, Rn, <operand2></operand2>	z	Z		<	121	d := Rn - Operand2
RSB{cond}{S} Rd, Rn, <operand2>   N Z C V    </operand2>		with carry	SBC(cond)(S) Rd, Rn, <operand2></operand2>	z	2		<	771	td := Rn - Operand2 - NOT(Сату)
RSC(cond) { S} Rd, Rn, < Operand2>   N Z C V   Rd := Operand2-Rn		reverse subtract	RSB{cond}{S} Rd, Rn, <operand2></operand2>	Z	Z		<	Part	td := Operand2 - Rn
SE QSUB{cond} Rd, Rm, Rn   Q Rd := SAT(Rm-Rn)		reverse subtract with carry	Rd, Rn,	Z	Z		<	271	kd := Operand2 Rn NOT(Carry)
SE   QDSUB{cond} Rd, Rm, Rn   Q   Rd := SAT(Rm - SA)			QSUB{cond} Rd, Rm, R						td := SAT(Rm - Rn)
2   MUL(cond){S} Rd, Rm, Rs   N   Z   C*   Rd := (Rm *Rs)[31:1   M   WMLL{cond}{S} Rd, Rm, Rs, Rn   N   Z   C*   Rd := (Rm *Rs)[31:1   Rm, Rs   N   Z   C*   Rd := (Rm *Rs)+R   Rm   Rs   Rn   Rn   Rn   Rn   Rn   Rn   Rn									td := SAT(Rm - SAT(Rn * 2))
2   MLA{cond}{S} Rd, Rm, Rs, Rn   N   Z   C*   Rd := ((Rm *Rs) + R   M   UMULL{cond} {S} RdLo, RdHi, Rm, Rs   N   Z   C*   V*   RdHi,RdLo := unsign accumulate long   6   UMAAL{cond} {S} RdLo, RdHi, Rm, Rs   N   Z   C*   V*   RdHi,RdLo := unsign   M   SMULL{cond} {S} RdLo, RdHi, Rm, Rs   N   Z   C*   V*   RdHi,RdLo := unsign   M   SMULL{cond} {S} RdLo, RdHi, Rm, Rs   N   Z   C*   V*   RdHi,RdLo := unsign   M   SMULL{cond} {S} RdLo, RdHi, Rm, Rs   N   Z   C*   V*   RdHi,RdLo := unsign   M   SMULL{cond} {S} RdLo, RdHi, Rm, Rs   N   Z   C*   V*   RdHi,RdLo := signed   RdHi, Rm, Rs   N   Z   C*   V*   RdHi,RdLo := signed   RdHi, Rm, Rs   N   Z   C*   V*   RdHi,RdLo := signed   RdHi, Rm, Rs   N   Z   C*   V*   RdHi,RdLo := signed   RdHi, Rm, Rs   N   Z   C*   V*   RdHi,RdLo := signed   RdHi, Rm, Rs   N   Z   C*   V*   RdHi,RdLo := signed   RdHi, Rm, Rs   N   Z   C*   V*   RdHi,RdLo := signed   RdHi, Rm, Rs   N   Z   C*   V*   RdHi,RdLo := signed   RdHi, Rm, Rs   N   Z   C*   V*   RdHi,RdLo := signed   RdHi, Rm, Rs   N   Z   C*   V*   RdHi,RdLo := signed   RdHi, Rm, Rs   N   Z   C*   V*   RdHi,RdLo := signed   RdHi, Rm, Rs   N   Z   C*   V*   RdHi,RdLo := signed   RdHi, Rm, Rs   N   Z   C*   V*   RdHi,RdLo := signed   RdHi, Rm, Rs   N   Z   C*   V*   RdHi,RdLo := signed   RdHi, Rm, Rs   Rs   N   Z   C*   V*   RdHi,RdLo := signed   RdHi, Rm, Rs   Rs   N   Z   C*   V*   RdHi,RdLo := signed   RdHi, Rm, Rs   Rs   N   Z   C*   V*   RdHi,RdLo := signed   RdHi,RdLo := RdHi,R	Mr	ltiply	MUL(cond)(s) Rd, Rm,	z		ů		77	\( \text{d} := (\text{Rm * Rs})[31:0]
M   UMULL		and accumulate	MLA(cond){S} Rd, Rm, Rs,	Z		ů,		771	\( \text{d} := ((\text{Rm * Rs}) + \text{Rn})[31:0]
Autolong			UMULL{cond}{S} RdLo, RdHi, Rm,	z			*	771	dHi,RdLo := unsigned(Rm * Rs)
accumulate long  6 UM-ALT (cond) RdLo, RdHi, Rm, Rs  M SMULT (cond) {\$5} RdLo, RdHi, Rm, Rs  M SMULT (cond) {\$5} RdLo, RdHi, Rm, Rs  N Z C* V*  RdHi,RdLo:= signed  SE SMULXy (cond) Rd, Rm, Rs  SE SMULXy (cond) Rd, Rm, Rs  Ccumulate  SE SMLAXY (cond) Rd, Rm, Rs, Rn  SE SMLAXY (cond) Rd, Rm, Rs, Rn  Ccumulate long  SE SMLAXY (cond) Rd, Rm, Rs, Rn  SE SMLAXY (cond) Rd, Rm, Rs, Rn  Add  6 SMLAD{X} (cond) Rd, Rm, Rs, Rn  SE SMLAD{X} (cond) Rd, Rm, Rs  A SMLAD{X} (cond) Rd, Rm, Rs, Rn  SE SMLAD{X} (cond) Rd, Rm, Rs, Rn  SE SMLAD{X} (cond) Rd, Rm, Rs, Rn  G SMLSD{X} (cond) Rd, Rm, Rs, Rn  G SMLSD{X} (cond) Rd, Rm, Rs, Rn  G SMLSD{X} (cond) Rd, Rm, Rs, Rn  G SMLSLD{X} (cond) Rd, Rm, Rs  G SMLSLD{X} (cond) Rd, Rm, Rs  G SMLSLD{X} (cond) Rd, Rm, Rs  G			UMLAL {cond} {S} RdLo, RdHi, Rm,	Z			*		dHi,RdLo := unsigned(RdHi,RdLo + Rm * Rs)
M   SMULE   Cond   E   RdLo, RdH1, Rm, Rs   N   Z   C* V*   RdH; RdLo := signed   SE   SMULxy   Cond   Rd, Rm, Rs   N   Z   C* V*   RdH; RdLo := signed   SE   SMULxy   Cond   Rd, Rm, Rs   Rs   Rd := Rm[X] * Rs[y]   Rd := Rm[X] * Rs[y] * Rs[	!	accumulate long	UMAAL { cond }	;			:	1 7	idHi,RdLo ≔ unsigned(RdHi + RdLo + Rm * Rs)
SHULLAY (cond) Rd, Rm, Rs   Rd := Rm[x] * Rs[y]    SE   SMULLAY (cond) Rd, Rm, Rs   Rd := Rm[x] * Rs[y]    SE   SMULLAY (cond) Rd, Rm, Rs   Rn     SE   SMLAXY (cond) Rd, Rm, Rs     Rd := Rn - (Rm * R	Ji C		SMULL(cond){S} Kalo, Kall, Km,	2 7			7	- 12	(dH, kdLo := signed(km + ks)
SMILWy{cond} Rd, Rm, Rs, Rn   Rd := (Rm *Rs(y)) 4    SE SMILWy{cond} Rd, Rm, Rs, Rn   Q Rd := Rn + Rm[x] *     SE SMILWy{cond} RdLo, RdH1, Rm, Rs     6 SMILWy{cond} RdLo, RdH2, Rm, Rs     6 SMILWY{cond} Rd, Rm, Rs, Rn   Q Rd := Rn + Rm[15:0] * Rs     6 SMILWY{cond} Rd, Rm, Rs, Rn   Q Rd := Rn + Rm[15:0] * Rs     6 SMILWY{cond} Rd, Rm, Rs, Rn   Q Rd := Rn + Rm[15:0] * Rs     6 SMILWY{cond} Rd, Rm, Rs, Rn   Q Rd := Rn + Rm[15:0] * Rs     6 SMILWY{cond} Rd, Rm, Rs   Q Rd := Rn + Rm[15:0] * Rs     6 SMILWY{cond} Rd, Rm, Rs   Q Rd := Rn + Rm[15:0] * Rs     6 SMILWY{cond} Rd, Rm, Rs   Q Rd := Rn + Rm[15:0] * Rs     6 SMILWY{cond} Rd, Rm, Rs   Rn   Q Rd := Rn + Rm[15:0] * Rs     6 SMILWY{cond} Rd, Rm, Rs   Rn   Q Rd := Rn + Rm[15:0] * Rs     6 SMILWY{cond} Rd, Rm, Rs   Rn   R   Rn + Rs     6 SMILWY{cond} Rd, Rm, Rs   Rn   Rd := Rn + (Rm *Rs)[63:0] * Rd     6 SMILWY{cond} Rd, Rm, Rs   Rn   Rd := Rn + (Rm *Rs)     7 SMILWY{cond} Rd, Rm, Rs   Rd := Rn + (Rm *Rs)     8 SMILWY{cond} Rd, Rm, Rs   Rd := Ac + Rm[15:0] * Rd := number of lead     8 SMILWY{cond} Rd, Rm   Rs   Rd := number of lead     8 SMILWY{cond} Rd, Rm   Rs   Rd := number of lead     8 SMILWY{cond} Rd, Rm   Rs   Rd := number of lead     8 SMILWY{cond} Rd, Rm   Rs   Rd := number of lead     8 SMILWY{cond} Rd, Rm   Rs   Rd := number of lead     8 SMILWY{cond} Rd, Rm   Rs   Rd := number of lead     8 SMILWY{cond} Rd, Rm   Rs   Rd := number of lead     8 SMILWY{cond} Rd, Rm   Rs   Rd := number of lead     8 SMILWY{cond} Rd, Rm   Rs   Rd := number of lead     8 SMILWY{cond} Rd, Rm   Rs   Rd := number of lead     8 SMILWY{cond} Rd, Rm   Rs   Rd := number of lead     8 SMILWY{cond} Rd, Rm   Rs   Rd := number of lead     8 SMILWY{cond} Rd, Rm   Rs   Rd := number of lead     8 SMILWY{cond} Rd, Rm   Rs   Rd := number of lead     8 SMILWY{cond} Rd, Rm   Rs   Rd := number of lead     8 SMILWY{cond} Rd, Rm   Rs   Rd := number of lead     8 SMILWY{cond} Rd, Rm   Rs   Rd := number of lead     8 SMILWY{cond} Rd, Rm   Rs   Rd     8 SMILWY{cond} Rd, Rm   Rs   Rd     9 Rd :=	_		SMITTAY (cond) Rd. Rm. Rs	:	ı	(		<del></del>	d' = Rm(x) * Re(v)
5E SMLAXY(cond) Rd, Rm, Rs, Rn       Q Rd := Rn + Rm[X]* *         5E SMLAWy(cond) Rd, Rm, Rs, Rn       Q Rd := Rn + (Rm * Rs Rn XL) (Rd := Rn + (Rm * Rs Rn XL) (Rd := Rn + (Rm * Rs Rn XL) (Rd := Rn + (Rm * Rs Rn XL) (Rd := Rn + (Rm * Rs Rn XL) (Rd := Rn + (Rm + Rm[15:0] * Rs Rn			SMULWy (cond) Rd, Rm,					<del>'</del> =1	d := (Rm * Rs[y])[47:16]
5E       SMLAWy{cond} Rd, Rm, Rs, Rn       Q Rd:=Rn+(Rm*Rs         5E       SMLALXy{cond} RdLo, RdHi, Rm, Rs       RdHi,RdLo:=RdHi, RdLo; RdLo         6       SMLAD{X}{cond} Rd, Rm, Rs       Q Rd:=Rn [5:0]*Rs         6       SMLAD{X}{cond} Rd, Rm, Rs       Q Rd:=Rn [5:0]*Rs         6       SMLAD{X}{cond} Rd, Rm, Rs       Q Rd:=Rn+Rm [5:0]*Rs         6       SMLAD{X}{cond} Rd, Rm, Rs       Q Rd:=Rn+Rm [5:0]*Rs         6       SMLSD{X}{cond} Rd, Rm, Rs       Q Rd:=Rn+Rm [5:0]*Rs         6       SMLSD{X}{cond} Rd, Rm, Rs       Q Rd:=Rn+Rm [5:0]*Rs         6       SMLSD{X}{cond} Rd, Rm, Rs       Q RdHi,RdLo:=RdHi,RdHi,RdLo:=RdHi,RdLo:=RdHi,RdLo:=RdHi,RdLo:=RdHi,RdLo:=RdHi,RdLo:=RdHi,RdHi,RdHi,RdHi,RdHi,RdHi,RdHi,RdHi,			SMLAxy(cond)						td := Rn + Rm[x] * Rs[y]
5E SMLALxy{cond} RdLo, RdHi, Rm, Rs       RdHi,RdLo = RdHi,         6 SMLAD{X}{cond} Rd, Rm, Rs       Q Rd := Rm[15:0] * Rs         6 SMLAD{X}{cond} Rd, Rm, Rs, Rn       Q Rd := Rm[15:0] * Rs         6 SMLAD{X}{cond} Rd, Rm, Rs, Rn       Q Rd := Rm + Rm[15:0] * Rs         6 SMLAD{X}{cond} Rd, Rm, Rs, Rn       Q Rd := Rn + Rm[15:0] * Rs         6 SMLSD{X}{cond} RdHi, RdLo, Rm, Rs       Q Rd := Rn + Rm[15:0] * Rs         6 SMLSD{X}{cond} Rd, Rm, Rs, Rn       Q Rd := Rn + Rm[15:0] * Rs         6 SMMLSLD{X}{cond} Rd, Rm, Rs       Q RdHi,RdLo := RdHi, Rm[15:0] * Rs         6 SMMLSLD{X}{cond} Rd, Rm, Rs       Q RdHi,RdLo := RdHi, Rm[15:0] * Rs         6 SMMLSLD{X}{cond} Rd, Rm, Rs       Q RdHi,RdLo := RdHi, Rm[15:0] * Rs         6 SMMLSLD{X}{cond} Rd, Rm, Rs       Q RdHi,RdLo := RdHi, Rm[15:0] * Rs         6 SMMLSLD{X}{cond} Rd, Rm, Rs       Rd := Rn + (Rm * Rs)[63:0] * Rd := Rn + (Rm * Rs			SMLAWy{cond} Rd, Rm, Rs,						d := Rn + (Rm * Rs[y])[47:16]
6 SMMAD{X}{cond} Rd, Rm, Rs 6 SMLAD{X}{cond} Rd, Rm, Rs 7 Rn Q Rd:=Rn+Rm[15:0]*Rs 6 SMLAD{X}{cond} Rd, Rm, Rs, Rn 7 Q Rd:=Rn+Rm[15:0]*Rs 6 SMLSD{X}{cond} RdHi, RdLo, Rm, Rs 7 Q Rd:=Rn+Rm[15:0]*Rs 8 Q Rd:=Rn	-		SMLALxy{cond} RdLo, RdHi, Rm,					늄	(dHi,RdLo := RdHi,RdLo + Rm[x] * Rs[y]
6 SMLAD{X}{cond} Rd, Rm, Rs, Rn 6 SMLAD{X}{cond} RdHi, RdLo, Rm, Rs 6 SMLSD{X}{cond} RdHi, RdLo, Rm, Rs 6 SMUSD{X}{cond} Rd, Rm, Rs 6 SMUSD{X}{cond} Rd, Rm, Rs 6 SMLSD{X}{cond} Rd, Rm, Rs, Rn 6 SMLSD{X}{cond} Rd, Rm, Rs, Rn 7 RdLo, Rm, Rs 8 SMMLSD{X}{cond} Rd, Rm, Rs, Rn 8 CRHIRGLO:=RdHi, RdLo:=RdHi, RdLo:=RdHi, RdLo:=RdHi, RdHi, Rs, Rn 9 Rd:=Rn+Rm[15:0]*Rd 9 RdHi,RdLo:=RdHi, RdHi, Rm, Rs 10 Rd:=Rn+Rm[15:0]*Rd:=Rm+Rm[15:0]*Rd 11 Rd:=Rn-(Rm*Rs)[63:0]*Rd:=Rn-(Rm*Rs)[63:0]*Rd 12 Rd:=Rn-(Rm*Rs)[63:0]*Rd 13 Rd:=Rn-(Rm*Rs)[63:0]*Rd 14 Rd:=Rn-(Rm*Rs)[63:0]*Rd 15 SMMLS{R}{cond} Ac, Rm, Rs 16 SMMLS{R}{cond} Ac, Rm, Rs 17 Rd:=Rn-(Rm*Rs) 18 Rd:=Rn-(Rm*Rs) 18 Rd:=Ac+Rm[15:0]*Rd 18 Rd:=number of lead 18 Rd:=number of lead 18 Rd:=number of lead	Į.	al signed multiply, add							td := Rm[15:0] * RsX[15:0] + Rm[31:16] * RsX[31:16]
6 SMLALD{X}{cond} RdHi, RdLo, Rm, Rs 6 SMLSD{X}{cond} Rd, Rm, Rs, Rn 6 SMLSLD{X}{cond} Rd, Rm, Rs, Rn C Rdi=Rn+Rm[15:0]*Rs 6 SMMUL{R}{cond} Rd, Rm, Rs, Rn C Rdi=Rn+Rm[15:0]*Rs 6 SMMUL{R}{cond} Rd, Rm, Rs C SMMUL{R}{cond} Rd, Rm, Rs C SMMLS{R}{cond} Rd, Rm, Rs, Rn C SMMLS{R}{cond} Rd, Rm, Rs, Rn C SMMLS{R}{cond} Rd, Rm, Rs C SMMLS{R}{cond} Rd, Rm C SMMLS{R}{cond} Rd SMMLS{R}{cond} C SMMLS{R}{cond} Rd, Rm C SMMLS{R}{cond} Rd SMMLS{R}{cond} C Rd = number of lead Rd = number of lead		and accumulate	SMLAD{X}{cond} Rd, Rm, Rs,						\d := Rn + Rm[15:0] * RsX[15:0] + Rm[31:16] * RsX[31:16]
6 SMUSD{X}{cond} Rd, Rm, Rs 6 SMLSD{X}{cond} Rd, Rm, Rs 7 Rd = Rn[15:0]*Rs 7 Rd = Rn + Rn[15:0]*Rs 8 SMLSD{X}{cond} Rd, Rm, Rs, Rn 9 Rd = Rn + Rn[15:0]*Rs 1 SMUL{R}{cond} RdH, RdLo, Rm, Rs 1 SMUL{R}{cond} Rd, Rm, Rs 1 SMUL{R}{cond} Rd, Rm, Rs, Rn 1 SMUL{R}{cond} Rd, Rm, Rs, Rn 2 Rd = Rn + Rn[15:0]*RdH, RdLo := RdH, RdH, Rs](63:0) 3 SMUL{R}{cond} Rd, Rm, Rs, Rn 4 Rd = Rn + (Rm * Rs)(63:0) 4 Rd = Rn + (Rm * Rs) 4 Rd = Rn - (Rm * Rs) 4 Rd = Rn - (Rm * Rs) 4 Rd = Ac + Rm[15:0]*Rd 4 Rd := number of lead 6 SMUSD{X} MIASY{cond} Rd, Rm 6 SMUL{Rd} Rd := number of lead 6 SMUSD{X} Rd := number of lead 6 SMUSD{X} Rd := number of lead	~=-	and accumulate long	SMLALD{X}{cond} RdHi, RdLo, Rm,					_	dHi,RdLo := RdHi,RdLo + Rm[15:0] * RsX[15:0] + Rm[31:16] * RsX[31:16]
6 SMLSD{X}{cond} Rd, Rm, Rs, Rn 6 SMLSD{X}{cond} Rd, Rm, Rs, Rn 6 SMLSD{X}{cond} RdHi, RdLo, Rm, Rs 6 SMMUL{R}{cond} RdHi, RdLo, Rm, Rs 6 SMMUL{R}{cond} Rd, Rm, Rs 6 SMMLS{R}{cond} Rd, Rm, Rs, Rn 6 SMMLS{R}{cond} Rd, Rm, Rs, Rn 6 SMMLS{R}{cond} Rd, Rm, Rs, Rn 7 SMLS{R}{cond} Rd, Rm, Rs, Rn 8 Rd := Rn + Rm[15:0] 8 Rd := Ac + Rm[15:0] 8 Rd := number of lead 8 Rd := number of lead	ַם	al signed multiply, subtract	SMUSD(X)(cond) Rd, Rm,						kd := Rm[15:0] * RsX[15:0] - Rm[31:16] * RsX[31:16]
6 SMMSLD{X}{cond} RdHi, RdLo, Rm, Rs 6 SMMTL{R}{cond} Rd, Rm, Rs 6 SMMTL{R}{cond} Rd, Rm, Rs 6 SMMTLA{R}{cond} Rd, Rm, Rs 6 SMMLS{R}{cond} Rd, Rm, Rs, Rn 6 SMMLS{R}{cond} Rd, Rm, Rs, Rn 7 Rd := Rn + (Rm * Rs)[63:6] 8 Rd := Rn - (Rm * Rs) 8 Ac := Ac + Rm[15:0] 8 XS MIARH{cond} Ac, Rm, Rs 8 XS MIARY{cond} Ac, Rm, Rs 8 Ac := Ac + Rm[15:0] 8 CLZ{cond} Rd, Rm 8 Rd := number of lead 8 Rd := number of lead		and accumulate	SMLSD(X)(cond) Rd, Rm, Rs,						\( \d := \text{Rn} + \text{Rm}[15:0] * \text{RsX}[15:0] - \text{Rm}[31:16] * \text{RsX}[31:16]
6 SMMIL(R) (cond) Rd, Rm, Rs 6 SMMILA(R) (cond) Rd, Rm, Rs, Rn 6 SMMILS(R) (cond) Rd, Rm, Rs, Rn (te XS MIA(cond) Ac, Rm, Rs XS MIAPH (cond) Ac, Rm, Rs XS MIAXY (cond) Ac, Rm, Rs 5 CLZ(cond) Rd, Rm	<u> </u>	and accumulate long	SMLSLD{X}{cond} RdHi, RdLo, Rm,						kdHi, RdLo := RdHi, RdLo + Rm[15:0] * RsX[15:0] - Rm[31:16] * RsX[31:16]
SMMLA[R] (cond) Rd, Rm, Rs, Rn SMMLS[R] (cond) Rd, Rm, Rs, Rn MIA(cond) Ac, Rm, Rs MIAPH(cond) Ac, Rm, Rs MIARY(cond) Ac, Rm, Rs CLZ(cond) Rd, Rm	Si	med most significant word multiply	SMMUL(R)(cond) Rd, Rm,					-	kd := (Rm * Rs)[63:32]
SMMLE(R) (CONG) KG, KM, KS, KN MIA(CONG) AC, RM, RS MIAPH(CONG) AC, RM, RS MIARY(CONG) AC, RM, RS CLZ(CONG) Rd, RM		and accumulate	SMMLA{R}{cond} Rd, Rm, Rs,					1 727	ld := Rn + (Rm * Rs)[63:32]
MIA(cond) Ac, Rm, Rs MIAPH(cond) Ac, Rm, Rs MIAXY(cond) Ac, Rm, Rs CLZ(cond) Rd, Rm		and subtract	SMMLS(R)(cond) Rd, Rm, Rs,					17	(d := Kn - (Km * Ks)[63:32]
XS MIAPH{cond} Ac, Rm, Rs XS MIAxy{cond} Ac, Rm, Rs 5 CLZ{cond} Rd, Rm	M	ltiply with internal 40-bit accumulate							Ac := Ac + Rm * Rs
XS MIAxy{cond} Ac, Rm, Rs 5 CLZ{cond} Rd, Rm			XS MIAPH {cond} Ac, Rm, Rs	_					\c := Ac + Rm[15:0] * Rs[15:0] + Rm[31:16] * Rs[31:16]
5 CLZ cond Rd, Rm			MIAxy{cond} Ac, Rm,					_	\c := Ac + Rm[x] * Rs[y]
	රි	unt leading zeroes	CLZ{cond} Rd,					H	Rd := number of leading zeroes in Rm

	{endianness}	Can be BE (Big Endian) or LE (Little Endian).
Refer to Table Condition Field. Omit for unconditional execution.	<a_mode2></a_mode2>	Refer to Table Addressing Mode 2.
Refer to Table Flexible Operand 2. Shift and rotate are only available as part of Operand2.	<a_mode2p></a_mode2p>	Refer to Table Addressing Mode 2 (Post-indexed only).
Refer to Table PSH fields.	<a_mode3></a_mode3>	Refer to Table Addressing Mode 3.
Either CPSR (Current Processor Status Register) or SPSR (Saved Processor Status Register)	<a_mode4l></a_mode4l>	Refer to Table Addressing Mode 4 (Block load or Stack pop).
Updates condition flags if S present.	<a_mode45></a_mode45>	Refer to Table Addressing Mode 4 (Block store or Stack push).
Flag is unpredictable in Architecture v4 and earlier, unchanged in Architecture v5 and later.	<a_mode5></a_mode5>	Refer to Table Addressing Mode 5.
Sticky flag, Always updates on overflow (no S option). Read and reset using MRS and MSR.	<reglist></reglist>	A comma-separated list of registers, enclosed in braces ( and ).
Four Greater than or Equal flags. Always updated by parallel adds and subtracts.	<reglist-pc></reglist-pc>	As <reglist>, must not include the PC.</reglist>
B meaning half-register [15:0], or T meaning [31:16].	<reglist+pc></reglist+pc>	As <reglist>, including the PC.</reglist>
A 32-bit constant, formed by right-rotating an 8-bit value by an even number of bits.	{ : }	Updates base register after data transfer if ! present.
RsX is Rs rotated 16 bits if X present. Otherwise, RsX is Rs.	+/-	+ or (+ may be omitted.)
Refer to Table Prefixes for Parallel instructions	507	Refer to Table ARM architecture versions.
Refer to Table Processor Modes	<iflags></iflags>	Interrupt flags. One or more of a, i, f (abort, interrupt, fast interrupt).
R13 for the processor mode specified by	{R}	Rounds result to nearest if R present, otherwise truncates result.

# ARM Addressing Modes Quick Reference Card

Defaulton   Defa	7		7	A	Odata	2	OE i Autini
6	Parallel	Halfword-wise addition	9	<pre><prefix>ADD16{cond} Rd, Rn, Rm</prefix></pre>	-	1	6] := Rn[31:16]
6 <pre>cprefix&gt;ADD8(cond) Rd, Rn, Rm 6 <pre>cprefix&gt;SUB8(cond) Rd, Rn, Rm 6 <pre>cprefix&gt;SUB8(cond) Rd, Rn, Rm differences</pre>6 <pre>USADA8(cond) Rd, Rn, Rm differences</pre>6 USADA8(cond) Rd, Rm, Rs, Rn  MOV(cond) {S} Rd, COPErand2&gt;</pre></pre>	arithmetic	Halfword-wise subtraction	6	<pre><prefix>SUB16{cond} Rd, Rn, Rm</prefix></pre>			GE[Rd[31:16] := Rn[31:16] - Rm[31:16], Rd[15:0] := Rn[15:0] - Rm[15:0]
Byte-wise subtraction		Byte-wise addition	9	<pre><prefix>ADD8{cond} Rd, Rn, Rm</prefix></pre>			GE $Rd[31:24] := Rn[31:24] + Rm[31:24], Rd[23:16] := Rn[23:16] + Rm[23:16], Rd[15:8] := Rn[15:8] + Rm[15:8], Rd[7:0] := Rn[7:0] + Rm[7:0]$
Halfword-wise exchange, add, subtract Halfword-wise exchange, subtract, add Unsigned sum of absolute differences  and accumulate  Move NOT PSR to register register to PSR immediate to PSR immed		Byte-wise subtraction	6	<pre><prefix>SUB8{cond} Rd, Rn, Rm</prefix></pre>			GE $ Rd[31:24] := Rn[31:24] - Rnn[31:24], Rd[23:16] := Rn[23:16] - Rnn[23:16], Rd[15:8] := Rn[15:8] - Rnn[15:8], Rd[7:0] := Rn[7:0] - Rnn[7:0]$
Halfword-wise exchange, subtract, add   6   Cprefix>SUBADDX{cond} Rd, Rn, Rm   CBE   Unsigned sum of absolute differences   6   USADB&{cond} Rd, Rm, Rs, Rn   Rm   Move   Move   Mov{cond} Rd, Rm, Rs, Rn   Move   Mov{cond} Rd, Coperand2>   N Z C   MvN{cond} Rd, Coperand2>   N Z C   MvN{cond} Rd, Coperand2>   N Z C   MvN{cond} Rd, Coperand2>   N Z C   Rd-bit accumulator to register to PSR   MsR{cond} Rd, Coperand2>   Rd-bit accumulator to register to 40-bit accumulator   XS Mrx{cond} Rd, Coperand2>   Rd-coperand2>   Rd-co		Halfword-wise exchange, add, subtract	6				Rd[31:16] := Rn[31:16]
Unsigned sum of absolute differences  and accumulate  Move  NOT  NOT  PSR to register  register to PSR  40-bit accumulator to register  Copy  Compare  DOR  ORR  Compare  Bit Clear  Compare  negative  negative  negative  negative  negative  negative  negative  negister two halfwords  Consigned saturate two halfwords  ORR  Cond)  Signed saturate two halfwords  ORR  Cond)		Halfword-wise exchange, subtract, add	6	<pre><prefix>SUBADDX{cond} Rd, Rn, Rm</prefix></pre>			GE $Rd[31:16] := Rn[31:16] - Rm[15:0], Rd[15:0] := Rn[15:0] + Rm[31:16]$
Move	=	Unsigned sum of absolute differences	6	USADB{cond} Rd, Rm, Rs			Rd := Abs(Rm[31:24] - Rs[31:24]) + Abs(Rm[23:16] - Rs[23:16]) + Abs(Rm[15:8] - Rs[15:8]) + Abs(Rm[7:0] - Rs[7:0])
Move         MOV{cond}{S} Rd, <operand2>         N Z C           NOT         MVN{cond}{S} Rd, <operand2>         N Z C           PSR to register         3 MSR{cond} Rd, <psr>         Rd, <psr>         Rd           register to PSR         3 MSR{cond} Rd, <psr>         ACPSR&gt;           40-bit accumulator         XS MRA{cond} RdLo, RdHi, Ac         N Z C           1 Copy         XS MRA{cond} RdLo, RdHi, Ac         N Z C           1 Test         TST{cond} Rd, <operand2>         N Z C           1 Test equivalence         TST{cond} Rd, <operand2>         N Z C           1 Test equivalence         TST{cond} Rd, <operand2>         N Z C           1 Test equivalence         TST{cond} Rd, Rd, <operand2>         N Z C           1 Test equivalence         TST{cond} Rd, Rd, <operand2>         N Z C           2 CRR         TSD{cond} Rd, Rd, <operand2>         N Z C           2 DRR         DRR{cond} S Rd, Rn, <operand2>         N Z C           3 BIC(cond) Rd, Rd, Rn, <operand2>         N Z C           4 CMP{cond} Rd, Rd, Rd, Rd, <operand2>         N Z C           5 SSAT{cond} Rd, Rd, Rd, <operand2>         N Z C V           6 SSAT{cond} Rd, Resat&gt;, Rm{, ASR &lt; sh&gt;}         Q           9 Unsigned saturate two halfwords         6 USAT[cond] Rd, Rd, Rd, Asat&gt;, Rm         Q</operand2></operand2></operand2></operand2></operand2></operand2></operand2></operand2></operand2></operand2></psr></psr></psr></operand2></operand2>		and accumulate	6	USADA8{cond} Rd, Rm, Rs, Rn			Rd := Rn + Abs(Rm[31:24] - Rs[31:24]) + Abs(Rm[23:16] - Rs[23:16]) + Abs(Rm[15:8] - Rs[15:8]) + Abs(Rm[7:0] - Rs[7:0])
NOT	Move	Move		MOV{cond}{S} Rd, <operand2></operand2>	7		Rd := Operand2
PSR to register         3 MRS{cond} Rd, <psr> register to PSR         3 MSR{cond} <psr> - cfields&gt;, Rm immediate to PSR         3 MSR{cond} <psr> - cfields&gt;, Rm immediate to PSR         3 MSR{cond} <psr> - cfields&gt;, RdHi, Ac           40-bit accumulator to register register to 40-bit accumulator         XS MRA{cond} RdLo, RdHi, Ac         N Z C           Test         TST{cond} Rd, <operand2>         N Z C           Test equivalence         TST{cond} Rn, <operand2>         N Z C           AND         AND{cond} SR, Rd, Rn, <operand2>         N Z C           EOR         DRR{cond} SR, Rd, Rn, <operand2>         N Z C           EOR         DRR{cond} SR, Rd, Rn, <operand2>         N Z C           Bit Clear         DRR{cond} SR, Rd, Rn, <operand2>         N Z C           Bit Clear         DRR{cond} SR, Rd, Rn, <operand2>         N Z C           Bit Clear         DRR{cond} Rn, <operand2>         N Z C           Compare         CMP{cond} Rn, <operand2>         N Z C           Bit Clear         DRR{cond} Rn, <operand2>         N Z C           Bit Clear         DRR{cond} Rn, <operand2>         N Z C V           Signed saturate two halfwords         6 SSAT{cond} Rd, #csat&gt;, Rm{, LSI <sh>}         Q           Unsigned saturate two halfwords         6 USAT[cond] Rd, #csat&gt;, Rm         Q           Unsigned saturate two halfwords</sh></operand2></operand2></operand2></operand2></operand2></operand2></operand2></operand2></operand2></operand2></operand2></psr></psr></psr></psr>		NOT		$MVN\{cond\}\{S\}\ Rd$ , <operand2></operand2>	Z		Rd := 0xFFFFFFFF EOR Operand2
register to PSR  immediate to PSR  40-bit accumulator to register  Copy  Copy  Copy  Test  Copy  Test  Copy  Test  Copy  Test equivalence  AND  Bit Clear  Compare  negative  Compare  negative  Compare  Signed saturate two halfwords  Cond  C		PSR to register	ယ	MRS {cond} Rd, <psr></psr>			Rd := PSR
immediate to PSR  40-bit accumulator to register  Copy  Copy  Test  Test  Test  Copy  Test  Copy  Cond  AND  AND  EOR  CORR  Bit Clear  Compare  megative  Compare  Compare  megative  Compare  Signed saturate two halfwords  Cond		register to PSR	w	<psr>_<fields>,</fields></psr>			PSR := Rm (selected bytes only)
## 40-bit accumulator to register   XS   MRA   cond   RdLo, RdH1, Ac   register to 40-bit accumulator   XS   MRA   cond   Rd, cond   Rd, coperand2   Copy   Copy   Cond   Rd, coperand2   Rd   Coperand2   Rd   Coperand2   Rd   Coperand2   Rd   Coperand2   Rd   Rd   Coperand2   Rd   Rd   Rd   Coperand2   Rd   Rd   Rd   Coperand2   Rd   Rd   Rd   Rd   Rd   Rd   Rd		immediate to PSR	ψ	<psr>_<fields>,</fields></psr>			PSR := immed_8r (selected bytes only)
Copy		40-bit accumulator to register	X	MRA (cond)			RdLo := Ac[31:0], RdHi := Ac[39:32]
Test equivalence		register to 40-bit accumulator	X				Ac[31:0] := RdLo, Ac[39:32] := RdHi
Test		Сору	6	Rd,			Rd := Operand2
Test equivalence	Logical	Test		TST(cond) Rn, <operand2></operand2>	Z		Update CPSR flags on Rn AND Operand2
AND  AND  AND  AND  AND  AND  AND  AND		Test equivalence		TEQ(cond) Rn, <operand2></operand2>	Z		Update CPSR flags on Rn EOR Operand2
BOR   Cond   S   Rd, Rn, < Operand   N Z C		AND		AND(cond)(S) Rd, Rn, <operand2></operand2>	8		Rd := Rn AND Operand2
ORR         ORR{cond}{S} Rd, Rn, <operand2>         N Z C           Bit Clear         BIC{cond}{S} Rd, Rn, <operand2>         N Z C           BCOmpare         CMP{cond} Rd, Rn, <operand2>         N Z C V           CMP{cond} Rn, <operand2>         N Z C V           Signed saturate word, right shift         6 SSAT{cond} Rd, #csat&gt;, Rm{, ASR <sh>}         Q           Left shift         SSAT{cond} Rd, #csat&gt;, Rm{, LSL <sh>}         Q           Signed saturate two halfwords         6 SSATI6{cond} Rd, #csat&gt;, Rm{, ASR <sh>}         Q           Unsigned saturate word, right shift         6 USAT{cond} Rd, #csat&gt;, Rm{, ASR <sh>}         Q           Unsigned saturate two halfwords         6 USAT{cond} Rd, #csat&gt;, Rm{, LSL <sh>}         Q           Unsigned saturate two halfwords         6 USAT16{cond} Rd, #csat&gt;, Rm{, LSL <sh>}         Q</sh></sh></sh></sh></sh></sh></operand2></operand2></operand2></operand2>		EOR		EOR{cond}{S} Rd, Rn, <operand2></operand2>	Z		Rd := Rn EOR Operand2
Bit Clear  Compare		ORR		ORR{cond}{S} Rd, Rn, <operand2></operand2>	2		Rd := Rn OR Operand2
Compare   CMP{cond} Rn, <operand2>   N Z C V     negative   CMP{cond} Rn, <operand2>   N Z C V     Signed saturate word, right shift   6   SSAT{cond} Rd,   #<sat>, Rm{, ASR <sh>}   Q     Signed saturate two halfwords   CMP{cond} Rd,   #<sat>, Rm{, LSL <sh>}   Q     Unsigned saturate two halfwords   CMP{cond} Rd,   #<sat>, Rm     Unsigned saturate two halfwords   CMPT{cond} Rd,   #<sat>, Rm{, LSL <sh>}   Q     USAT{cond} Rd,   #<sat>, Rm{, LSL <sh>}   Q     Usat   CMP{cond} Rd,   #<sat>, Rm{, LSL <sh} #<sat="" cmp{cond}="" rd,="" usat=""  ="">, Rm{, LSL <sh} #<sat="" cmp{cond}="" rd,="" usat=""  ="">, Rm{, LSL <sh} #<sat="" cmp{cond}="" rd,="" usat=""  ="">, Rm{, LSL <sh} #<sat="" cmp{cond}="" halfwords="" rd,="" saturate="" two="" unsigned=""  ="">, Rm{, LSL <sh} #<sat="" cmp{cond}="" halfwords="" rd,="" saturate="" two="" unsigned=""  ="">, Rm{, LSL <sh} #<sat="" cmp{cond}="" halfwords="" rd,="" saturate="" two="" unsigned=""  ="">, Rm{, LSL <sh} #<sat="" cmp{cond}="" halfwords="" rd,="" saturate="" two="" unsigned=""  ="">, Rm{, LSL <sh} #<sat="" cmp{cond}="" halfwords="" rd,="" saturate="" two="" unsigned=""  ="">, Rm{, LSL <sh} #<sat="" cmp{cond}="" halfwords="" rd,="" saturate="" two="" unsigned=""  ="">, Rm{, LSL <sh} #<sat="" cmp{cond}="" halfwords="" rd,="" saturate="" two="" unsigned=""  ="">, Rm{, LSL <sh} #<sat="" cmp{cond}="" halfwords="" rd,="" saturate="" two="" unsigned=""  ="">, Rm{, LSL <sh} #<sat="" cmp{cond}="" halfwords="" rd,="" saturate="" two="" unsigned=""  ="">, Rm{, LSL <sh} #<sat="" cmp{cond}="" halfwords="" rd,="" saturate="" two="" unsigned=""  ="">, Rm{, LSL <sh} #<sat="" cmp{cond}="" halfwords="" rd,="" saturate="" two="" unsigned=""  ="">, Rm{, LSL <sh} #<sat="" cmp{cond}="" halfwords="" rd,="" saturate="" two="" unsigned=""  ="">, Rm{, LSL <sh} #<sat="" cmp{cond}="" halfwords="" rd,="" saturate="" two="" unsigned=""  ="">, Rm{, LSL <sh} #<sat="" cmp{cond}="" halfwords="" rd,="" saturate="" two="" unsigned=""  ="">, Rm{, LSL <sh} #<sat="" cmp{cond}="" halfwords="" rd,="" saturate="" two="" unsigned=""  ="">, Rm{, LSL <sh} td="" unsigne<=""  =""><th></th><td>Bit Clear</td><td></td><td>BIC(cond)(S) Rd, Rn, <operand2></operand2></td><td>Z</td><td></td><td>Rd := Rn AND NOT Operand2</td></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sat></sh></sat></sh></sat></sh></sat></sh></sat></sh></sat></sh></sat></sh></sat></sat></sh></sat></sh></sat></operand2></operand2>		Bit Clear		BIC(cond)(S) Rd, Rn, <operand2></operand2>	Z		Rd := Rn AND NOT Operand2
CMM   Cond   Rn , < Operand2 >   N Z C V	Compare	Compare		CMP{cond} Rn, <operand2></operand2>	ZC		Update CPSR flags on Rn - Operand2
Signed saturate word, right shift  left shift  SSAT {cond} Rd, # <sat>, Rm{, ASR <sh>}  Q  SSAT {cond} Rd, #<sat>, Rm{, LSL <sh>}  Q  Signed saturate two halfwords  SSAT {cond} Rd, #<sat>, Rm  Q  Unsigned saturate two halfwords  6 USAT {cond} Rd, #<sat>, Rm{, ASR <sh>}  Q  USAT {cond} Rd, #<sat>, Rm{, ASR <sh>}  Q  USAT {cond} Rd, #<sat>, Rm{, LSL <sh>}  Q  USAT {cond} Rd, #<sat>, Rm{, LSL <sh>}  Q  USAT {cond} Rd, #<sat>, Rm{, LSL <sh>}  Q  Q</sh></sat></sh></sat></sh></sat></sh></sat></sh></sat></sat></sh></sat></sh></sat>		negative		CMN{cond} Rn, <operand2></operand2>	Z C		Update CPSR flags on Rn + Operand2
SSAT{cond} Rd, # <sat>, Rm{, LSL <sh>} 6 SSAT16{cond} Rd, #<sat>, Rm 6 USAT{cond} Rd, #<sat>, Rm{, ASR <sh>} USAT{cond} Rd, #<sat>, Rm{, LSL <sh>} 0 USAT{cond} Rd, #<sat>, Rm{, LSL <sh>} 0 USAT16{cond} Rd, #<sat>, Rm (, LSL <sh>) 0 USAT16{cond} Rd, #<sat>, Rm</sat></sh></sat></sh></sat></sh></sat></sh></sat></sat></sh></sat>	Saturate	Signed saturate word, right shift	6			0	Rd := SignedSat((Rm ASR sh), sat). <sat> range 0-31, <sh> range 1-32.</sh></sat>
6 SSAT16{cond} Rd, # <sat>, Rm Q 6 USAT{cond} Rd, #<sat>, Rm(, ASR <sh>} USAT{cond} Rd, #<sat>, Rm(, LSL <sh>} Q 6 USAT16{cond} Rd, #<sat>, Rm Q Q</sat></sh></sat></sh></sat></sat>		left shift		SSAT{cond} Rd, # <sat>, Rm{, LSL <sh>}</sh></sat>		0	Rd := SignedSat((Rm LSL sh), sat). <sat> range 0-31, <sh> range 0-31.</sh></sat>
6 USAT{cond} Rd, # <sat>, Rm{, ASR <sh>} Q USAT{cond} Rd, #<sat>, Rm{, LSL <sh>} Q 6 USAT16{cond} Rd, #<sat>, Rm Q</sat></sh></sat></sh></sat>		Signed saturate two halfwords	6	SSAT16{cond} Rd, # <sat>, Rm</sat>		Q	Rd[31:16] := SignedSat(Rm[31:16], sat), < sat > range 0-15.
USAT(cond) Rd, # <sat>, Rm{, LSL <sh>} Q Rd := UnsignedSat((Rm Q Rd):1:16] := UnsignedSat((Rm</sh></sat>		Unsigned saturate word, right shift	6	USAT(cond) Rd, # <sat>, Rm(, ASR <sh>)</sh></sat>		0	Rd := UnsignedSat((Rm ASR sh), sat). <sat>range 0-31, <sh>range 1-32.</sh></sat>
6 USAT16{cond} Rd, # <sat>, Rm Q</sat>		left shift		USAT(cond) Rd, # <sat>, Rm(, LSL <sh>)</sh></sat>		0	
		Unsigned saturate two halfwords	6	USAT16{cond} Rd, # <sat>, Rm</sat>		0	Rd[31:16] := UnsignedSat(Rm[31:16], sat), Rd[15:0] := UnsignedSat(Rm[15:0], sat). <sat> range 0-15.</sat>

## ARM Instruction Set Quick Reference Card

Softw
[R13m] := R14, [R13m + 4] := CPSR PC := [Rn], CPSR := [Rn + 4] Prefetch abort or enter debug state.
Disable specified interrups, optional change mode.  Enable specified interrups, optional change mode.  Sets endianness for loads and saves. <endianness> can be BE (Big Endian) or LE (</endianness>
R14 := address of next instruction, R15 := Rm[3] Change to Thumb if Rm[0] is 1 Change to Java state
R15 := Rm, Change to Thumb if Rm[0] is 1 R14 := address of next instruction, R15 := label,
R15 := label R14 := address of next instruction, R15 := label
Rd[7:0] := Ru[7:0] if $GE[0] = 1$ , else $Rd[7:0] := 1Bits[15:8], [23:16], [31:24] selected similarly by$
Rd[31:24] := Rm[23:16], Rd[23:16] := Rm[ Rd[15:8] := Rm[7:0], Rd[7:0] := Rm[15:8], Rd[31:16] := Rm[7] * &FFFF
Rd[31:24] := Rm[7:0], Rd[23:16] := Rm[15:8], Rd[15:8] := Rm[23:16], Rd[7:0] := Rm[31:24] Rd[15:8] := Rm[7:0], Rd[7:0] := Rm[15:8],
Rd[31:16] := Rn[31:16] + ZeroExtend((Rm ROR (8 Rd[15:0] := Rn[15:0] + ZeroExtend((Rm ROR (8 Rd[31:0] := Rn[31:0] + ZeroExtend((Rm ROR (8
Rd[31:0] := Rn[31:0] + SignExtend((Rm ROR (8) Rd[31:0] := Rn[31:0] + ZeroExtend((Rm ROR (8) Rd[31:0] + ZeroExtend((Rm ROR (Rm ROR (Rm ROR
R(d)31:0) := Rn(31:0) + StgnExtend((Rm ROR) R) (31:16) := Rn(31:16) + StgnExtend((Rm ROR) (8 R) (15:0) := Rn(15:0) + StgnExtend((Rm ROR) (8 R) (15:0) + StgnExtend((Rm ROR) (8 R)
Rd[31:0] := ZeroExtend((Rm ROR (8 * sh))[7:0]). sh 0-3.
Rd[31:16] := ZeroExtend((Rm ROR (8 * sh))[73:16]), Rd[15:0] := ZeroExtend((Rm ROR (8 * sh))[7:0]). sh 0-3.
Rd[31:0] := SignExtend((Rm ROR (8 * sh))[7:0]), sh 0-3,
Rd[31:16] := SignExtend((Rm ROR (8 * sh))[23: Rd[15:0] := SignExtend((Rm ROR (8 * sh))[7:0])
Rd[31:16] := Rn[31:16], Rd[15:0] := (Rm ASR sh Rd[31:0] := SionExtendi/Rm ROR (8 * sh))/[15:0]
Rd[15:0] := Rn[15:0], Rd[31:16] := (Rm LSL sh)
Action

# ARM Addressing Modes Quick Reference Card

Load   Word   Load cachange)   Load ca	Operation		S.	Assembler	Action	Notes
Byte		Word User mode privilege		LDR(cond) Rd, <a_mode2> LDR(cond)T Rd, <a_mode2p></a_mode2p></a_mode2>		Rd must not be R15.  Rd must not be R15.
Byte		hranch (8 5T; and exchange)		TDR(cond) R15 <a mode2=""></a>		
Byte   LDR   cond   PR d. <a_mode2>   Rd := ZeroExtend(byte from address]    </a_mode2>		,		ı	(§ 5T: Change to Thumb if [address][0] is 1)	
LDR   Cond   BT Rd   ca_mode2P		Byte		LDR(cond)B Rd, <a_mode2></a_mode2>		Rd must not be R15.
Signed   A   LDR (cond) SB Rd, <a_mode3>   Rd := SignExten(lbyte from address)    </a_mode3>		User mode privilege		LDR(cond)BT Rd, <a_mode2p></a_mode2p>		Rd must not be R15.
Halfword   Signed   A   LDR (cond) H Rd, <a_mode3>   Rd := ZeroExtent[halfword from address]    </a_mode3>		signed	4	LDR(cond)SB Rd, <a_mode3></a_mode3>		Rd must not be R15.
Signed   Doubleword   Doublew		Halfword	4	LDR{cond}H Rd, <a_mode3></a_mode3>		Rd must not be R15.
Doubleword   STR {cond}   Rd , Rm , [Rn]   Cond   Rd , Rm , [Rn]   Rm]   Doubleword   Doubleword   STR {cond}   Rd , Rm , [Rn]   Rm]   Rd   Stre mode registers to [Rn]   Rd   STR {cond}   Rd , Rm , [Rn]   Rd   Stre mode registers to [Rn]   Rd   STR {cond}   Rd , Rm , [Rn]   Rd   Stre mode registers to [Rn]   Rd   STR {cond}   Rd , Rm , [Rn]   Rd   Stre mode registers to [Rn]   Rd   STR {cond}   Rd , Rm , [Rn]   Rd   Stre mode registers to [Rn]   Rd   STR {cond}   Rd , Rm , [Rn]   Rd   Stre mode registers to [Rn]   Rd   Rd   STR {cond}   Rd , Rm , [Rn]   Rd   Stre mode registers to [Rn]   Rd   STR {cond}   Rd , Rm , [Rn]   Rd   Stre mode registers to [Rn]   Rd   STR {cond}   Rd , Rm , [Rn]   Rd   Stre mode registers to [Rn]   Rd   Rd   Rd   Rd   Rd   Rd   Rd   R		signed	4	LDR(cond)SH Rd, <a_mode3></a_mode3>		Rd must not be R15.
multiple return (and exchange)    Application   CPSR   LDM   Cond   <a _mode4l=""> Rn   1   <a _reglist-pc="">   Load itst of registers from [Rn]    </a></a>			5E*	LDR(cond)D Rd, <a_mode3></a_mode3>		Rd must be even, and not R
return (and exchange)    IDM{cond}<=_mode41> Rn{!}, <reglist+pc></reglist+pc>				-		
### Pipe   And resistors   And registers		return (and exchange)			Load registers, R15 := [address][31:1]   (§ 5T: Change to Thumb if [address][0] is 1)	
User mode registers   LDM{cond} <a mode4l=""> Rn, <reglist-pc></reglist-pc></a>		and restore CPSR		LDM{cond} <a_mode4l> Rn{!}, <reglist+pc>^</reglist+pc></a_mode4l>	Load registers, branch (§ 5T: and exchange), CPSR := SPSR	Use from exception modes
reload Memory system hint  exclusive Semaphore operation  Word  User mode privilege  Byte  User mode privilege  Byte  User mode privilege  Byte  User mode privilege  User mode privilege  Byte  User mode privilege  User mode privilege  Byte  User mode privilege  Byte  User mode privilege  User mode privilege  Byte  User mode privilege  Byte  STR {cond} Rd, <a_mode2>  STR {cond} BT Rd, <a_mode3>  STR {cond} BT Rd, <a_mode4>  STR {cond} BT Rd, <a_mode4>  STR {cond} BT Rd, <a_mode3>  STR {cond} BT Rd, <a_mode3>  STR {cond} BT Rd, <a_mode4>  STR {cond} BT Rd, <a_mode3>  STR {cond} BT Rd, <a_mode4>  STR {cond} BT Rd, <a_mod< th=""><th></th><th>User mode registers</th><th></th><th>LDM{cond}<a_mode4l> Rn, <reglist-pc>^</reglist-pc></a_mode4l></th><th></th><th>Use from privileged modes</th></a_mod<></a_mode4></a_mode4></a_mode4></a_mode4></a_mode4></a_mode4></a_mode4></a_mode4></a_mode4></a_mode4></a_mode4></a_mode4></a_mode4></a_mode4></a_mode4></a_mode4></a_mode4></a_mode4></a_mode4></a_mode4></a_mode4></a_mode4></a_mode4></a_mode4></a_mode4></a_mode4></a_mode4></a_mode4></a_mode3></a_mode4></a_mode3></a_mode3></a_mode4></a_mode4></a_mode3></a_mode3></a_mode3></a_mode3></a_mode3></a_mode3></a_mode3></a_mode3></a_mode3></a_mode3></a_mode3></a_mode3></a_mode3></a_mode3></a_mode3></a_mode3></a_mode3></a_mode3></a_mode3></a_mode2></a_mode2></a_mode2></a_mode2></a_mode2>		User mode registers		LDM{cond} <a_mode4l> Rn, <reglist-pc>^</reglist-pc></a_mode4l>		Use from privileged modes
Word   Semaphore operation   6   LDREX{cond} Rd, [Rn]   Rd := [Rn], rag address as exclusive access			SE*	PLD <a_mode2></a_mode2>		Cannot be conditional.
Word   STR {cond} Rd, <a_mode2>   [address] := Rd                                    </a_mode2>	Load exclusive	Semaphore operation	6	LDREX{cond} Rd, [Rn]		Rd, Rn must not be R15.
Byte   User mode privilege   STR {cond} T Rd, <a_mode2p>   [address] := Rd    </a_mode2p>		Word		STR{cond} Rd, <a_mode2></a_mode2>	[address] := Rd	
Byte   STR {cond} B Rd, <a_mode2>   [address][7:0] := Rd[7:0]     Halfword</a_mode2>		User mode privilege		STR{cond}T Rd, <a_mode2p></a_mode2p>	[address] := Rd	
User mode privilege		Byte		STR{cond}B Rd, <a_mode2></a_mode2>	[address][7:0] := Rd[7:0]	
Halfword		User mode privilege		STR{cond}BT Rd, <a_mode2p></a_mode2p>	[address][7:0] := Rd[7:0]	
Doubleword   Dou			4	STR{cond}H Rd, <a_mode3></a_mode3>	[address][15:0] := Rd[15:0]	
multiple       Push, or Block data store       STM{cond} <a_mode48> Rn{!}, <reglist>       Store list of registers to [Rn]         exclusive       Semaphore operation       6 STREX{cond} Rd, Rm, [Rn]       [Rn] = Rm if allowed.         Word       3 SWP{cond} Rd, Rm, [Rn]       Rm, [Rn]         Byte       3 SWP{cond} Rd, Rm, [Rn]       Rm [Rn]         Byte       3 SWP{cond} Rd, Rm, [Rn]       Rm]         Rn] := Rm, Rd := temp       := Rm[7:0]. Rd := temp         Rd] := Rm[7:0]. Rd := temp</reglist></a_mode48>			5E*	STR{cond}D Rd, <a_mode3></a_mode3>		Rd must be even, and not R
User mode registers				~	Store list of registers to [Rn]	
exclusive         Semaphore operation         6         STREX{cond} Rd, Rm, [Rn]         [Rn] = Rm if allowed, Rd := 0 if successful, else I           Word         3         SWP{cond} Rd, Rm, [Rn]         temp := Rni, [Rn] := Rm, Rd := temp := ZcroExtend([Rn]]7:0]; Rni[7:0] := Rni[7:0]. Rd := temp := ZcroExtend([Rn]]7:0]; Rd := temp := ZcroExtend([		User mode registers				Use from privileged modes
Word  3 SWP{cond} Rd, Rm, [Rn]  Byte  3 SWP{cond}B Rd, Rm, [Rn]	Store exclusive	Semaphore operation	6		Rn] := Rm if allowed,  Rd := 0 if successful. else 1	Rd, Rm, Rn must not be R1
3 SWP{cond}B Rd, Rm, [Rn]		Word	w	SWP{cond} Rd, Rm, [Rn]	temp := [Rn], [Rn] := Rm, Rd := temp	
		Byte	ယ	SWP{cond}B Rd, Rm, [Rn]	lemp := ZeroExtend([Rn][7:0]),	

### **Quick Reference Card ARM Addressing Modes**

Addressing	Addressing Mode 2 - Word and Unsigned Byte Data Transfer	Jnsign	ed Byte D	ata T	ansfer	
Pre-indexed	Immediate offset	Rn,	[Rn, #+/- <immed_12>]{!}</immed_12>	med_1	2>] {1}	
	Zero offset	[Rn]				Equivalent to [Rn,#0]
	Register offset	Rn,	[Rn, +/-Rm]{!}	Ξ		
	Scaled register offset	Rn.	+/-Rm,	‡ TST	<shift>] {!}</shift>	[Rn, +/-Rm, LSL # <shift>] {!} Allowed shifts 0-31</shift>
		Rn,	+/-Rm,	LSR #	<shift>] {!}</shift>	[Rn, +/-Rm, LSR $\#<$ shift>] {!} Allowed shifts 1-32
		Rn,	+/-Rm,	ASR #	<shift>] {!}</shift>	$[Rn, +/-Rm, ASR \#] \{!\} Allowed shifts 1-32$
		Rn,	+/-Rm,	ROR #	<shift>] [1]</shift>	[Rn, +/-Rm, ROR $\#$ <shift>] {1} Allowed shifts 1-31</shift>
		Rn,	+/-Rm, RRX] [1]	RRX]	Ξ.	
Post-indexed	Immediate offset	[Rn] ,	[Rn], #+/- <inmed_12></inmed_12>	nmed	12>	
	Register offset	[Rn],	[Rn], +/-Rm			
	Scaled register offset [Rn], +/-Rm, LSL # <shift></shift>	[Rn],	+/-Rm,	TST	# <shift></shift>	Allowed shifts 0-31
		[Rn],	+/-Rm, LSR	LSR	# <shift></shift>	Allowed shifts 1-32
		[Rn],	, +/-Rm, ASR	ASR	# <shift></shift>	Allowed shifts 1-32
		[Rn],	+/-Rm,	ROR	[Rn], +/-Rm, ROR # <shift></shift>	Allowed shifts 1-31
		[Rn],	[Rn], +/-Rm, RRX	RRX		

7			
Rota		[Rn], +/-Rm, RRX	
Regi	Allowed shifts 1-31	[Rn], +/-Rm, ROR # <shift></shift>	
Rota	Allowed shifts 1-32	[Rn], +/-Rm, ASR # <shift></shift>	
Arit	Allowed shifts 1-32	[Rn], +/-Rm, LSR # <shift></shift>	
Logi	Allowed shifts 0-31	[Rn], +/-Rm, LSL # <shift></shift>	Scaled register offset
Logi		[Rn], +/-Rm	Register offset
Imm		[Rn], #+/- <immed_12></immed_12>	Post-indexed Immediate offset
Flex		[Rn, +/-Rm, RRX] {!}	
	Allowed shifts 1-31	[Rn, +/-Rm, ROR # <shift>] {1} Allowed shifts 1-31</shift>	
×	Allowed shifts 1-32	[Rn, +/-Rm, ASR # <shift>] {!} Allowed shifts I-32</shift>	
n.	Allowed shifts 1-32	[Rn, +/-Rm, LSR # <shift>] {!} Allowed shifts 1-32</shift>	
- A	Allowed shifts 0-31	$[Rn, +/-Rm, LSL #]{!}$ Allowed shifts 0-31	Scaled register offset
		[Rn, +/-Rm]{!}	Register offset
<u> </u>	Equivalent to [Rn,#0]	[Rn]	Zero offset
=		[Rn, #+/- <immed_12>]{!}</immed_12>	Pre-indexed Immediate offset
I			

Addressing Mode 2 (Post-indexed only)  Logical shift rig	LOBIcat Shift tell	Toxical this loss	[Rn], +/-Rm, RRX Rotate right exte	[Rn], +/-Rm, ROR # <shift> Allowed shifts 1-31 Register</shift>	[Rn], +/-Rm, ASR # <shift> Allowed shifts 1-32 Rotate right imn</shift>	[Rn], +/-Rm, LSR # <shift> Allowed shifts 1-32 Arithmetic shift</shift>	Scaled register offset [Rn], +/-Rm, LSL # <shift> Allowed shifts 0-31   Logical shift rig</shift>	Register offset [Rn], +/-Rm   Logical shift left	Post-indexed Immediate offset [Rn], #+/- <i.mmed_12> Immediate value</i.mmed_12>	[Rn, +/-Rm, RRX] {1}   Hexible Opera	[Rn, +/-Rm, ROR # <shift>] {1} Allowed shifts 1-31</shift>	[Rn, +/-Rm, ASR # <shift>] {!} Allowed shifts 1-32 XS</shift>	[Rn, +/-Rm, LSR $*$ <shift>] {!} Allowed shifts 1-32   <math>nE*</math></shift>	Scaled register offset   [Rn, +/-Rm, LSL # <shift>] {!} Allowed shifts 0-31   nE</shift>	Register offset [Rn, +/-Rm] {!}	Zero offset [Rn] Equivalent to [Rn,#0] nT, nJ	Pre-indexed Immediate offset [Rn, #+/- <immed_12>] {!}</immed_12>	
Arithmetic shift right register	Logical shift right register	Logical shift left register	Rotate right extended	Register	Rotate right immediate	Arithmetic shift right immediate	Logical shift right immediate	Logical shift left immediate	Immediate value	Flexible Operand 2		XS XScale coproce	nE* E variants of A	nE All E variants o	M ARM architectu	nT, nJ T or J variants	n ARM architectu	

_	ARM architecture versions	ure versions
	п	ARM architecture version n and above.
n.#0]	nT, nJ	T or J variants of ARM architecture version n and above.
	×	ARM architecture version 3M, and 4 and above, except xM variants.
-31	лE	All E variants of ARM architecture version n and above.
-32	nE‡	E variants of ARM architecture version n and above, except xP variants.
-32	XS	XScale coprocessor instruction
-31		
	Elavible Cherand 3	and 9

#<immed\_8r>
Rm, LSL #<shift>
Rm, LSR #<shift>

Rm, Rm,

ASR #<shift> ROR #<shift>

Allowed shifts 0-31 Allowed shifts 1-32 Allowed shifts 1-32 Allowed shifts 1-31

Rm, Rm, Rm, Rm,

RRX LSL Rs LSR Rs ASR Rs ROR Rs

2			
cs /		[Rn], +/-Rm	Register
NE		[Rn], #+/- <immed_8></immed_8>	Post-indexed Immediate offset
EQ	•	[Rn, +/-Rm]{!}	Register
Mnem	Equivalent to [Rn,#0]	[Rn]	Zero offset
Conditio		[Rn, #+/- <immed_8>]{!}</immed_8>	Pre-indexed Immediate offset
	ransfer	Addressing Mode 3 - Halfword, Signed Byte, and Doubleword Data Transfer	Addressing Mode 3 - Halfword,
×			
to.		[Rn], +/-Rm, RRX	
f	Allowed shifts 1-31	[Rn], +/-Rm, ROR # <shift></shift>	
a	Allowed shifts 1-32	[Rn], +/-Rm, ASR # <shift></shift>	
Suffix	Allowed shifts 1-32	[Rn], +/-Rm, LSR # <shift></shift>	
PSR field	Allowed shifts 0-31	[Rn], +/-Rm, LSL # <shift></shift>	Scaled register offset
		[Rn], +/-Rm	Register offset
Rotate rig	Equivalent to [Rn],#0	[Rn]	Zero offset
Arithmetic		[Rn], #+/- <immed_12></immed_12>	Post-indexed Immediate offset

Addressing Mode 4 - Multiple Data Transfer
Block load Stack pop

Stack pop

IA
IB
DA
DB
Elock store
IA
IB
DA
DB
DD
DD

Decrement Before Increment After

Stack push

Decrement After

Increment Before Increment After

EA E E

Full Ascending

PSR fields	(use at least one suffix)	
Suffix	Meaning	
C	Control field mask byte	PSR[7:0]
Ħ	Flags field mask byte	PSR[31:24]
ຜ	Status field mask byte	PSR[23:16]
×	Extension field mask byte	PSR[15:8]

* ******		
a	Control field mask byte	PSR[7:0]
f	Flags field mask byte	PSR[31:24]
to	Status field mask byte	PSR[23:16]
×	Extension field mask byte	PSR[15:8]

Condition Field	۵		
Mnemonic	Description		Description (VFP)
EQ	Equal		Equal
NE	Not equal		Not equal, or unordered
CS / HS	Carry Set / Unsigned higher or same	higher or same	Greater than or equal, or unordered
CC / LO	Carry Clear / Unsigned lower	led lower	Less than
IW	Negative		Less than
Jđ	Positive or zero		Greater than or equal, or unordered
VS	Overflow		Unordered (at least one NaN operand)
VC	No overflow		Not unordered
H	Unsigned higher		Greater than, or unordered
SI	Unsigned lower or same	ame	Less than or equal
æ	Signed greater than or equal	or equal	Greater than or equal
TT	Signed less than		Less than, or unordered
GT	Signed greater than		Greater than
87	Signed less than or equal	qual	Less than or equal, or unordered
ΑL	Always (normally omitted)	nitted)	Always (normally omitted)
Processor Modes	des	Prefixes for Pa	Prefixes for Parallel Instructions
16	User	S Signed ari	Signed arithmetic modulo 28 or 216, sets CPSR GE bi
17	FIQ Fast Interrupt	Q Signed sat	Signed saturating arithmetic
8	IRQ Interrupt	SH Signed ani	Signed arithmetic, halving results
19	Supervisor	U Unsigned	Unsigned arithmetic modulo 28 or 216, sets CPSR GE
23	Abort	UQ Unsigned	Unsigned saturating arithmetic
27	Undefined	UH Unsigned	Unsigned arithmetic, halving results
31	System		

	Prett	Prefixes for Parallel Instructions
er	ຜ	Signed arithmetic modulo 28 or 216, sets CPSR GE bits
Fast Interrupt	Ю	Signed saturating arithmetic
Q Interrupt	HS	Signed arithmetic, halving results
pervisor	q	Unsigned arithmetic modulo 28 or 216, sets CPSR GE bits
ort	g	Unsigned saturating arithmetic
defined	HU	Unsigned arithmetic, halving results
stem		

Pre-indexed

Post-indexed

Zero offset Immediate offset

[Rn, #+/-<immed\_8\*4>]{|
[Rn]
[Rn], #+/-<immed\_8\*4>
[Rn], #8-bit copro. opt

{8-bit copro. option}

#+/-<immed\_8\*4>] [1]

Equivalent to [Rn,#0]

Addressing Mode 5 - Coprocessor Data Transfer

Decrement After Decrement Before

Increment Before

EA FA ED

Empty Descending

Full Ascending Empty Ascending Empty Ascending Empty Descending Full Descending

### **Quick Reference Card** ARM Addressing Modes

Coprocessor operations	§ Assembler	bler	Action	Notes
Data operations	2 CDF(c	CDP(cond) <copr>, <op1>, CRd, CRn, CRm(, <op2>)</op2></op1></copr>	Coprocessor dependent	
Alternative data operations	5 CDP2	CDP2 <copr>, <op1>, CRd, CRn, CRm{, <op2>}</op2></op1></copr>	Coprocessor dependent	Cannot be conditional.
Move to ARM register from coprocessor	2 MRC{c	MRC(cond) <copr>, <op1>, Rd, CRn, CRm(, <op2>)</op2></op1></copr>	Coprocessor dependent	
Alternative move	5 MRC2	MRC2 <copr>, <opl>, Rd, CRn, CRm{, <op2>}</op2></opl></copr>	Coprocessor dependent	Cannot be conditional.
Two ARM register move	5E* MRRC(	5E* MRRC(cond) <copr>, <opl>, Rd, Rn, CRm</opl></copr>	Coprocessor dependent	
Alternative two ARM register move	6 MRRC2	MRRC2 <copr>, <op1>, Rd, Rn, CRm</op1></copr>	Coprocessor dependent	Cannot be conditional.
Move to coproc from ARM reg	2 MCR(c	MCR(cond) <copr>, <op1>, Rd, CRn, CRm{, <op2>}</op2></op1></copr>	Coprocessor dependent	
Alternative move	5 MCR2	MCR2 <copr>, <opl>, Rd, CRn, CRm{, <op2>}</op2></opl></copr>	Coprocessor dependent	Cannot be conditional.
Two ARM register move	5E* MCRR	5E* MCRR {cond} <copr>, <op1>, Rd, Rn, CRm</op1></copr>	Coprocessor dependent	
Alternative two ARM register move	6 MCRR2	MCRR2 <copr>, <opl>, Rd, Rn, CRm</opl></copr>	Coprocessor dependent	Cannot be conditional.
Load	2   LDC(c	LDC{cond} <copr>, CRd, <a_mode5></a_mode5></copr>	Coprocessor dependent	
Alternative loads	5 LDC2	LDC2 <copr>, CRd, <a_mode5></a_mode5></copr>	Coprocessor dependent	Cannot be conditional.
Store	2 STC{c	STC{cond} <copr>, CRd, <a_mode5></a_mode5></copr>	Coprocessor dependent	
Alternative stores	5 STC2	5 STC2 <copr>. CRd. <a mode5=""></a></copr>	Coprocessor dependent	Cannot be conditional.

### **Proprietary Notice**

Words and logos marked with  $^{\otimes}$  or  $^{\bowtie}$  are registered trademarks or trademarks owned by ARM Limited. Other brands and names mentioned herein may be the trademarks of their respective owners.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by ARM in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This reference card is intended only to assist the reader in the use of the product. ARM Ltd shall not be liable for any loss or damage arising from the use of any information in this reference card, or any error or omission in such information, or any incorrect use of the product.

## **Document Number**

ARM QRC 0001H

Change Log

πQΉ	про	issue A B
Sept 2001 Jan 2003 Oct 2003	Nov 1998 Oct 1999 Oct 2000	Date June 1995 Sept 1996
688 688 688 688 688 688 688 688 688 688	CKS CKS BH	<b>Ву</b> ВЈН ВЈН
Sixth Release Seventh Release Eighth Release	Third Release Fourth Release Fifth Release	Change First Release Second Release