TRINITY COLLEGE DUBLIN THE UNIVERSITY OF DUBLIN

Faculty of Engineering, Mathematics and Science

School of Computer Science & Statistics

Integrated Computer Science Programme Year 2 Annual Examinations

Trinity Term 2015

Microprocessor Systems

Thursday 30 April 2015

Sports Centre

09:30 - 11:30

Dr Mike Brady

Instructions to Candidates:

Attempt **two** questions. All questions carry equal marks. Each question is scored out of a total of 20 marks.

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

Materials permitted for this examination:

A two-page document, entitled "Pthread Types and Function Prototypes" accompanies 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) Give an account of pipelining and how it can improve the performance of a
 processor. In your answer, explain how and why the full potential of pipelines is
 not always achievable. [10 marks]
 - (b) Explain how a cache works. Give a worked example of how cache would speed up the execution of a program loop designed to calculate the average of 1,000 integers stored sequentially in memory, given 10 ns memory access time, cache with an idealised access time of 0 ns and a processor that can execute a complete instruction every 1 ns, provided the instruction and data are in cache. [10 marks]

- 2. (a) Compare and contrast polled and interrupt-driven I/O. In your answer, highlight the advantages and drawbacks of each approach. For example, given that polling is so simple, why is it not used all the time, and given that interrupt handling is so processor efficient, why is it not used all the time? [8 marks]
 - (b) A system has four push-button switches S1, S2, S3 and S4 and four light-emitting diodes (LEDs) L1, L2, L3 and L4, much as you would have seen on the ARM boards. Write a fragment of ARM assembly language, complete with any equates and memory reservations needed, so that whenever a switch is pressed, the state of the corresponding LED should toggle, that is, it should change state if it was lit it should go dim, and if it was dim it should be lit.

Assume the switches are connected to location 0xE0040002 in bit positions 0, 1, 2 and 3 respectively such that the a bit is 1 if its switch is pressed and 0 otherwise, with no switch bounce.

Assume that the LEDs are connected to location 0xE0040004 in bit positions 0, 1, 2 and 3 respectively. To light a LED, set its bit to 1; to make it dim, set its bit to 0. Assume the other four bits in that location are "don't cares" — i.e. it doesn't matter what values you set them to. Assume that 0xE0040004 is write-only, i.e. that you can't read back values from it, and explain why this assumption is important. [12 marks]

- 3. (a) What is the purpose of different modes of operation of a processor, such as the different modes that are provided on the ARM processor? What are the extra registers for? [4 marks]
 - (b) What are desirable properties of an interrupt handler and why are they desirable?

 [4 marks]
 - (c) Write an interrupt handler which is called every 1 ms and which provides a debounced version of a push button switch's input. When pressed or release, the switch's output may bounce between its initial value and its final value for up to 7 ms. The switch is connected to bit 0 of location 0xE0040006 and the debounced version of it should be maintained at bit 0 of a location in RAM labelled CLEANSWITCH. [12 marks]

ASCII Code

		n Numbe		011	100	•••	110	
Row Number	000	001	010	011	100	101	110	111
0000	NUL	DLE	\Diamond	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	11	2	В	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	\$	4	D	T	d	ŧ
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BELL	ETB	t	7	G	W	g	w
1000	BS	CAN	(8	H	X	h	X
1001	HT	EM)	9	Ι	Y	i	у
1010	<i>LF</i>	SUB	*	•	J	Z	j	Z
1011	VT	ESC	+	;	K	[k	{
1100	FF	FS	,	<	L	\	1	
1101	CR	GS	-	=	M]	m	}
1110	SO	RS	•	>	N	٨	n	~
1111	SI	US	1	?	O	_	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 \Diamond here.

ARM[®] Instruction Set Quick Reference Card

Refer to Table Condition Field. Omit for unconditional execution. Coperand2> Refer to Table Flexible Operand 2. Shift and rotate are only available as part of Operand2. Falcium Cable PSR fields. Refer to Table Poerand 2. Shift and rotate are only available as part of Operand2. Refer to Table Condition Field. Omit for unconditional execution. Refer to Table Condition Field. Omit for unconditional execution.
id. Omit for unconditional execution. And 2. Shift and rotate are only available as part of Operand r Status Register) or SPSR (Saved Processor Status Register) sent. Status and earlier, unchanged in Architecture v5 and later. overflow (no S option). Read and reset using MRS and MSI overflow (no S option).

	{endianness}	Can be BE (Big Endian) or LE (Little Endian).
	<a_mode2></a_mode2>	Refer to Table Addressing Mode 2.
rand2.	<a_mode2p></a_mode2p>	Refer to Table Addressing Mode 2 (Post-indexed only).
	<a_mode3></a_mode3>	Refer to Table Addressing Mode 3.
ister)	<a_mode4l></a_mode4l>	Refer to Table Addressing Mode 4 (Block load or Stack pop).
	<a_mode4s></a_mode4s>	Refer to Table Addressing Mode 4 (Block store or Stack push).
ater.	<a_mode5></a_mode5>	Refer to Table Addressing Mode 5.
MSR.	<reglist></reglist>	A comma-separated list of registers, enclosed in braces { and }.
	<reglist-pc></reglist-pc>	As <reglist>, must not include the PC.</reglist>
	<reglist+pc></reglist+pc>	As <reglist>, including the PC.</reglist>
	{i}	Updates base register after data transfer if I present.
	+/-	+ or (+ may be omitted.)
	603	Refer to Table ARM architecture versions.
	<iflags></iflags>	Interrupt flags. One or more of a, i, f (abort, interrupt, fast interrupt).
	{R}	Rounds result to nearest if R present, otherwise truncates result.

with carry with carry saturating Subtract with carry with carry with carry with carry with carry with carry saturating double saturating freverse subtract with carry saturating double saturating saturating double saturating Moltiply and accumulate unsigned accumulate long unsigned accumulate long signed multiply long and accumulate long 16 * 16 bit and accumulate 32 * 16 bit and accumulate 16 * 16 bit and accumulate 32 * 16 bit and accumulate 32 * 16 bit and accumulate 30 * 16 bit and accumulate 31 * 16 bit and accumulate 32 * 16 bit and accumulate 35 * 5MULA 36 * 5MULA 37 * 16 bit and accumulate 36 * 5MULA 37 * 16 bit and accumulate 38 * 5MULA 39 * 5MULA 30 * 5MU	§ Assembler	Ш	늉	ates	۵		
出出 は は は は は は は は は は な の の の の の の の の の の の の の	ADD(cond)(s	ADD{cond}{S} Rd, Rn, <operand2> ADC{cond}{S} Rd, Rn, <operand2></operand2></operand2>	Z Z	ი ი	< <	ᄝᄰ	Rd := Rn + Operand2 Rd := Rn + Operand2 + Carry
おのりののののはおのとのののののおのとのののののおびとのののののおびとのののののおびとのののののおびとのののののおびとののののおびとののののののおびとののののののののおびとののののののののおびとのののののののののおびとのののののののののののののののののののののののののののののののののののの	5E QADD{cond} Rd, Rm, Rn	•			-		Rd := SAT(Rm + Rn)
お B B B B B B B B B B B B B B B B B B B		Rd, Rm, Rn			_	Q R	Rd := SAT(Rm + SAT(Rn * 2))
おいと X X A X X X X X X X X X X X X X X X X		Operand2>			٧		Rd := Rn - Operand2
おりりりりりりかめば K K K K K A K K N N N N N N N N N N N N	SBC(cond){s	_			۷	R.	Rd := Rn - Operand2 - NOT(Carry)
のりりりりりりの 3 25 25 25 25 25 25 25 25 25 25 25 25 25 		_	z z	a	<	굔	Rd := Operand2 - Rn
3 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2		_			٧	Rd	Rd := Operand2 - Rn - NOT(Carry)
3.000000000000000000000000000000000000		Rd, Rm, Rn				<u> </u>	Rd := SAT(Rm - Rn)
33NNNNMR 35 35 35 35 40 40 40 40 40 40 40 40 40 40 40 40 40 	SE.	Rd, Rm, Rn			_		1.11
9 9 9 9 9 9 9 9 8 8 8 8 8 8 8 8 8 8 8 8	2 MUL(cond)(S) Rd, Rm, Rs	ts		Ĝ			:= SAI(Rm - SAI(Rn * 2))
りりりりりりりりが 35 25 25 25 25 25 25 25 25 25 25 25 25 25	2	, Rn	Z Z			집	Rd := SAT(Rm - SAT(Rn * 2)) Rd := (Rm * Rs)[31:0]
りりりりりりりのは 35 25 25 25 25 25 25 25 25 25 25 25 25 25		Rm, Rs		ů	*	Rd	Rd := SAI(Rm - SAI(Rn * 2)) Rd := (Rm * Rs)[31:0] Rd := ((Rm * Rs) + Rn)[31:0]
りりりりりゅうががががますり		Rs	N N	Ç		R _d	Rd := SAI(Km ~ SAI(Rn * 2)) Rd := (Rm * Rs)[31:0] Rd := (Rm * Rs) + Rn)[31:0] RdHi,RdLo := unsigned(Rm * Rs)
とりりりょう ななば 出出 出来 X					٧*		<u>≅</u> .€
		UMAAL (cond) RdLo, RdHi, Rm, Rs		Ť.	4	몺	Rd := SAI(Rm * ZA) Rd := (Rm * Rs)[31:0] Rd := (Rm * Rs)+Rn)[31:0] RdHi,RdL o := unsigned(Rm * Rs) RdHi,RdL o := unsigned(RdHi,RdL o + Rm RdHi,RdL o := unsigned(RdHi + RdL o + Rr
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6	ong	R S	N		Ç Ç	문문	:= SAI(km - SAI(kn * 2)) := (Rm * Rs)[31:0] := (Rm * Rs) + Rn)[31:0] ii,RdLo := unsigned(Rm * Rs) ii,RdLo := unsigned(RdHi,RdLo + Rm ii,RdLo := unsigned(RdHi, RdLo + Rt ii,RdLo := unsigned(RdHi + RdLo + Rt ii,RdLo := signed(Rm * Rs)
現 現 現 み る る る る る み み み み み み み み み み み 	5E SMULxy{cond} Rd, Rm,		z z n n		<u> </u>	2 2 2 Z	Rd := SAI(Km - SAI(Kn * 2)) Rd := (Rm * Rs)[31:0] Rd := (Rm * Rs) + Rn)[31:0] RdHi,RdLo := unsigned(Rm * Rs) RdHi,RdLo := unsigned(RdHi,RdLo + Rm RdHi,RdLo := unsigned(RdHi + RdLo + Rt RdHi,RdLo := signed(Rm * Rs) RdHi,RdLo := signed(RdHi,RdLo + Rm * F)
8 3 3 3 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6	SE SMULWy{cond} Rd, Rm,	Hi, Rm, Rs RdHi, Rm, Rs RdHi, Rm, Rs Rs			<u> </u>	R R R	Rd := RAI(Km - SAI(Rn * 2)) Rd := (Rm * Rs)[31:0] Rd := (Rm * Rs) + Rn)[31:0] RdHi,RdLo := unsigned(Rm * Rs) RdHi,RdLo := unsigned(RdHi,RdLo + Rm * RdHi,RdLo := unsigned(RdH + RdLo + RdHi,RdLo := signed(RdH + RdLo + RdHi,RdLo := signed(RdHi,RdLo + Rm * RdHi,RdHi,RdLo := signed(RdHi,RdLo + Rm * RdHi,RdHi,RdLo := signed(RdHi,RdLo + Rm * RdHi,RdHi,RdHi,RdHi,RdHi))
8 H 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9	nd accumulate 5E SMLAXY{cond} Rd, Rm, Rs,	Hi, Rm, Rs RdHi, Rm, Rs RdHi, Rm, Rs Rs Rs			<u> </u>	R R R R R	Rd := RAI(Rm * ZA); Rd := (Rm * Rs)[31:0] Rd := (Rm * Rs) + Rn)[31:0] Rd := (Rm * Rs) + Rn)[31:0] RdHi,RdLo := unsigned(Rm * Rs) RdHi,RdLo := unsigned(RdHi,RdLo + Rm RdHi,RdLo := unsigned(RdH + RdLo + Rt RdHi,RdLo := signed(Rm * Rs) RdHi,RdLo := signed(RdH,RdLo + Rm * I RdHi,RdLo := signed(RdH,RdLo + Rm * I Rd := Rm[x] * Rs[y]]
6 6 6 6 6 5 E	nd accumulate SE SMLAWy{cond} Rd, Rm, Rs,	Hi, Rm, Rs RdHi, Rm, Rs RdHi, Rm, Rs Rs Rs					Rd := SAT(Rm * Z)) Rd := (Rm * Rs)[31:0] Rd := (Rm * Rs)[31:0] Rd := (Rm * Rs) + Rn)[31:0] RdHi,RdLo := unsigned(Rm * Rs) RdHi,RdLo := unsigned(RdHi,RdLo + Rm RdHi,RdLo := unsigned(RdHi + RdLo + RdHi,RdLo := signed(Rm * Rs) RdHi,RdLo := signed(RdHi,RdLo + Rm * 1) RdHi,RdLo := signed(RdHi,RdLo + Rm * 1) Rd := Rm[x] * Rs[y] Rd := (Rm * Rs[y])[47:16] Rd := (Rm * Rs[y])[47:16]
00000000		Rm, Rs i, Rm, Rs i, Rm, Rs Rn Rn			***************************************	O O R R R R R R R	:= SAI(km - SAI(Rn * 2)) := (Rm * Rs)[31:0] := ((Rm * Rs) + Rn)[31:0] ii,RdLo := unsigned(Rdm * Rs) ii,RdLo := unsigned(RdHi,RdLo + Rm ii,RdLo := signed(RdHi + RdLo + Rm ii,RdLo := signed(RdHi + RdLo + Rm ii,RdLo := signed(RdHi,RdLo + Rm * I;RdLo := signed(RdHi,RdLo + Rm * I;RdLo := signed(RdHi,RdLo + Rm * I) i= Rm[x] * Rs[y] i= Rm + Rm[x] * Rs[y] i= Rn + Rm[x] * Rs[y] i= Rn + (Rm * Rs[y])[47:16]
000000		Rm, Rs i, Rm, Rs i, Rm, Rs Rn Rn Rn Rn Rn					Rd := \(\)AT(\(\)Rm - \(\) \(\)AT(\(\)Rm \) \(\)
00000	•	Rm, Rs i, Rm, Rs i, Rm, Rs Rn Rn Rn Rn Rn Rn Rn Rn Rn					:= SAI(km - SAI(Rn * 2)) := (Rm * Rs)[31:0] := (Rm * Rs)[31:0] := (Rm * Rs) + Rn)[31:0] i,RdLo := unsigned(Rm * Rs) i,RdLo := unsigned(RdHi,RdLo + Rm i,RdLo := signed(RdHi + RdLo + R i,RdLo := signed(RdHi,RdLo + Rm * I i,RdLo := Rf[x] * Rs[y] := Rm[x] * Rs[y] := Rm + Rs[y])[47:16] := Rn + (Rm * Rs[y])[47:16]
00000	•	Rm, Rs i, Rm, Rs i, Rm, Rs Rn					:= SAI(Rm * SA](Rn * 2)) := (Rm * Rs)[31:0] := (Rm * Rs); Rn)[31:0] ii,RdLo := unsigned(RdH;RdLo + Rm ii,RdLo := unsigned(RdH; + RdLo + R ii,RdLo := signed(RdH; + RdLo + R ii,RdLo := signed(RdH;RdLo + Rm * i,RdLo := Rm[x] * Rs[y]] := Rm * Rs[y])[47:16] := Rn + Rm[15:0] * RsX[15:0] + Rm[3] * Rs[y] ii,RdLo := RdH;RdLo + Rm[x] * Rs[y] ii,RdLo := RdH;RdLo + RdH;RdLo + Rm[x] * Rs[y] ii,RdLo := RdH;RdLo + RdH;R
0000	999	Rm, Rs i, Rm, Rs i, Rm, Rs Rn Rn Rn Rn Rn Rn, Rs s, Rm, Rs					Rd := RM1(Rm - SAT(Rn * 2)) Rd := (Rm * Rs)[31:0] Rd := ((Rm * Rs) + Rn)[31:0] Rd := ((Rm * Rs) + Rn)[31:0] RdHI,RdL o := unsigned(RHI,RdL o + Rm RdHI,RdL o := unsigned(RdHI, RdL o + Rm RdHI,RdL o := signed(RdHI, RdL o + R RdHI,RdL o := signed(RdHI,RdL o + Rm * RdHI,RdL o := signed(RdHI,RdL o + Rm * RdHI,RdL o := signed(RdHI,RdL o + Rm * RdHI,RdL o := Rs[y]] Rd := (Rm * Rs[y])[47:16] RdHI,RdL o := (RdHI,RdL o + Rm[3]:16] RdHI,RdL o := (RdHI,RdL o + Rm[15:0] + Rm[3]:16] RdHI,RdL o := (RdHI,RdL o + Rm[15:0] + Rm[3]:16]
000	6666	Rm, Rs i, Rm, Rs i, Rm, Rs Rn					Rd := RAT(Rm - SAT(Rn * 2)) Rd := (Rm * Rs) S1:0] Rd := ((Rm * Rs) + Rn)[31:0] Rd := ((Rm * Rs) + Rn)[31:0] RdHI,RdL o := unsigned(RHI,RdL o + Rm RdHI,RdL o := unsigned(RdHI,RdL o + Rm RdHI,RdL o := signed(RdHI,RdL o + Rm * RdHI,RdL o := Rs[y]][47:16] Rd := Rn + Rm[x] * Rs[y]][47:16] Rd := Rn + Rm[x] * Rs[y][47:16] Rd := Rn + Rm[x] * Rs[y][47:16] Rd := Rn + Rm[x] * Rs[y][47:16] Rd := Rn + Rm[15:0] * RsX[15:0] + Rm[31:16] Rd := Rn + Rm[15:0] * RsX[15:0] + Rm[31:16] Rd := Rn + Rm[15:0] * RsX[15:0] + Rm[31:16]
0 0	6666	Rm, Rs i, Rm, Rs i, Rm, Rs Rn					:= SAI(Rm - SAI(Rn * 2)) := (Rm * Rs)[31:0] := (Rm * Rs)[31:0] ii,RdLo := unsigned(Rm * Rs) ii,RdLo := unsigned(RdHi,RdLo + Rm ii,RdLo := signed(RdHi,RdLo + Rm ii,RdLo := signed(RdHi,RdLo + Rm * rs) ii,RdLo := signed(RdHi,RdLo + Rm * rs[y][47:16] := Rm [x] * Rs[y] := Rm [x] * Rs[y][47:16] := Rm + Rm[x] * Rs[y][47:16] := Rn + Rm[x] * Rs[x][5:0] + Rm[x] * Rs[y] := Rn + Rm[15:0] * RsX[15:0] + Rm[15:0] *
٥	<u> </u>	Rm, Rs i, Rm, Rs i, Rm, Rs Rn					Rd := (Rm * Rs)[31:0] Rd := (Rm * Rs)[31:0] Rd := ((Rm * Rs) + Rn)[31:0] Rd := ((Rm * Rs) + Rn)[31:0] RdHi,RdLo := unsigned(Rm * Rs) RdHi,RdLo := unsigned(RdHi,RdLo + Rm RdHi,RdLo := signed(RdHi,RdLo + Rm RdHi,RdLo := signed(RdHi,RdLo + Rm * RdHi,RdLo := signed(RdHi,RdLo + Rm * RdHi,RdLo + Rm * Rs[y] Rd := Rm[x] * Rs[y] Rd := Rn + Rm[x] * Rs[y] Rd := Rn + Rm[x] * Rs[y] Rd := Rn + Rm[15:0] * RsX[15:0] + Rm[31:16] Rd := Rn + Rm[15:0] * RsX[15:0] - Rm[31:16] Rd := Rn + Rm[15:0] * RsX[15:0] - Rm[31:16] Rd := Rn + Rm[15:0] * RsX[15:0] - Rm[31:16] Rd := Rn + Rm[15:0] * RsX[15:0] - Rm[31:16] Rd := Rn + Rm[15:0] * RsX[15:0] - Rm[31:16] Rd := Rn + Rm[15:0] * RsX[15:0] - Rm[31:16]
	0000000	Rm, Rs i, Rm, Rs i, Rm, Rs Rn Rn Rn Rn Ru c, Rm, Rs s, Rn Lo, Rm, Rs s, Rn s, Rn Rs s, Rn Rs					Rd := (Rm * Rs.)[31:0] Rd := (Rm * Rs.) + Rn)[31:0] Rd := (Rm * Rs.) + Rn)[31:0] Rd := (Rm * Rs.) + Rn)[31:0] RdHi,RdLo := unsigned(RdHi,RdLo + Rm RdHi,RdLo := unsigned(RdHi,RdLo + Rm RdHi,RdLo := signed(RdHi,RdLo + Rn RdHi,RdLo := signed(RdHi,RdLo + Rm RdHi,RdLo := signed(RdHi,RdLo + Rm * 1) Rd := Rm[x] * Rs[y] Rd := Rm[x] * Rs[y] Rd := Rm + Rs[y][47:16] Rd := Rn + Rm[15:0] * Rs[y] Rd := Rn + Rm[15:0] * RsX[15:0] + Rm[31:16] Rd := Rn + Rm[15:0] * RsX[15:0] + Rm[31:16] Rd := Rn + Rm[15:0] * RsX[15:0] - Rm[31:16]
and subtract 6 SMML	00000000	Rm, Rs i, Rm, Rs i, Rm, Rs Rn Rn Rn kn Lo, Rm, Rs s, Rn Lo, Rm, Rs s, Rn Lo, Rm, Rs					Rd := RAT(Rm * Z)) Rd := (Rm * Rs)[31:0] Rd := (Rm * Rs)[31:0] Rd := (Rm * Rs) + Rn)[31:0] Rd := (Rm * Rs) + Rn)[31:0] Rd := (Rm * Rs) + Rn)[31:0] RdHi,RdLo := unsigned(RdHi,RdLo + Rm RdHi,RdLo := signed(RdHi,RdLo + Rm RdHi,RdLo := signed(RdHi,RdLo + Rm RdHi,RdLo := signed(RdHi,RdLo + Rm *) RdHi,RdLo := signed(RdHi,RdLo + Rm *) RdHi,RdLo := signed(RdHi,RdLo + Rm *) Rd := Rm[x] * Rs[y] Rd := (Rm * Rs[y])[47:16] Rd := (Rm * Rs[y])[47:16] Rd := (Rm + (Rm * Rs[y])[47:16] Rd := Rn + (Rm * Rs[y])[47:16] Rd := Rn + Rm[15:0] * RsX[15:0] + Rm[31:16] Rd := Rm + Rm[15:0] * RsX[15:0] - Rm[31:6] * R Rd := Rn + Rm[15:0] * RsX[15:0] - Rm[31:6] Rd := Rn + Rm[15:0] * RsX[15:0] - Rm[31:6] Rd := (Rm * Rs)[63:32] RdHi,RdLo := RdHi,RdLo + Rm[15:0] * R Rd := (Rm * Rs)[63:32] Rd := (Rm * Rs)[63:32] Rd := (Rm * Rs)[63:32]
Multiply with internal 40-bit accumulate XS MIA (cond) Ac, Rm, Rs	000000000	Rm, Rs i, Rm, Rs i, Rm, Rs Rn Rn Rn kn , Rm, Rs s, Rn LO, Rm, Rs s, Rn LO, Rm, Rs s, Rn LO, Rm, Rs					Rd := RAT(Rm * Z)) Rd := (Rm * Rs)[31:0] Rd := (Rm * Rs)[31:0] Rd := (Rm * Rs) + Rn)[31:0] Rd := (Rm * Rs) + Rn)[31:0] Rd := (Rm * Rs) + Rn)[31:0] RdHi,RdLo := unsigned(RdHi, RdLo + Rm RdHi,RdLo := unsigned(RdHi, RdLo + Rm RdHi,RdLo := signed(RdHi,RdLo + Rm * FRHI,RdLo := signed(RdHi,RdLo + Rm * FRHI,RdLo + Rm * Rs[y])[47:16] Rd := Rm + Rm[x] * Rs[y] Rd := Rn + (Rm * Rs[y])[47:16] Rd := Rn + (Rm * Rs[y])[47:16] Rd := Rn + Rm[15:0] * RsX[15:0] + Rm[31:16] * Rs[y] Rd := Rn + Rm[15:0] * RsX[15:0] + Rm[31:16] * Rs[x] Rd := Rn + Rm[15:0] * RsX[15:0] + Rm[31:16] * Rs[x] Rd := Rn + Rm[15:0] * RsX[15:0] + Rm[31:16] * Rs[x] Rd := Rn + Rm[15:0] * RsX[15:0] + Rm[31:16] * Rs[x] Rd := Rn + Rm[15:0] * RsX[15:0] + Rm[31:16] * Rs[x] RdHi,RdLo := RdHi,RdLo + Rm[15:0] * Rs[x] RdHi,RdLo := RdHi,RdLo + Rm[x] RdHi,RdLo
packed halfword XS MIAP	<u>x</u>	Hi, Rm, Rs RdHi, Rm, Rs RdHi, Rm, Rs Rs Rs Rs Rs, Rn Rs, Rn Rs, Rn Rs, Rn R, Rs R, Rn R, Rs R, Rn RGHLO, Rm, Rs M, Rs R, Rn RGHLO, Rm, Rs M, Rs RGHLO, Rm, Rs RS RGHLO, Rm, Rs RS RGHLO, Rm, RS RGHLO,					Rd := RAT(Km - SAT(Kn * 2)) Rd := (Rm * Rs)[31:0] Rd := (Rm * Rs)[31:0] Rd := (Rm * Rs) + Rn)[31:0] Rd := (Rm * Rs) + Rn)[31:0] RdHi,RdLo := unsigned(Rm * Rs) RdHi,RdLo := unsigned(RdHi + RdLo + Rm * Rs) RdHi,RdLo := unsigned(RdHi + RdLo + Rm * Rs) RdHi,RdLo := signed(RdHi,RdLo + Rm * Rs) RdHi,RdLo := signed(RdHi,RdLo + Rm * Rs) Rd := Rm [x] * Rs[y] Rd := Rn + Rm[x] * Rs[x] Rd := Rn + Rm[x] Rd := Rn + Rs[x] Rd := Rn
halfword XS MIAx	XX 8	Hi, Rm, Rs RdHi, Rm, Rs Rs Rs Rs Rs Rs, Rn Rs, Rn Rs, Rn RdHi, Rm, Rs m, Rs m, Rs m, Rs RdLo, Rm, Rs m, Rs n, Rs redLo, Rm, Rs m, Rs redLo, Rm, Rs m, Rs RdLo, Rm, Rs RdLo,					Rd := KAT(Rm * Z)) Rd := (Rm * Rs)[31:0] Rd := (Rm * Rs)[31:0] Rd := (Rm * Rs) + Rn)[31:0] Rd := (Rm * Rs) + Rn)[31:0] RdHi,RdL o := unsigned(RdH; RdL o + Rm RdHi,RdL o := unsigned(RdH; RdL o + Rm RdHi,RdL o := signed(RdH; RdL o + Rm + RdHi,RdL o + Rm[x] * Rs]y] Rd := Rm + Rm[x] * Rs[y] Rd := Rn + Rm[x] * Rs[x] + Rm[x] * Rs[y] Rd := Rn + Rm[x] * Rs[x] + Rm[x] * Rs[x] Rd := Rn + Rm[x] * Rs[x] + Rm[x] * Rs[x] Rd := Rn + Rm[x] * Rs[x] + Rm[x] * Rs[x] Rd := Rn + Rm[x] * Rs[x] * Rs[x] + Rm[x] * Rs[x] Rd := Rn + (Rm * Rs)[63:32] Rd := Rn - (Rm * Rs)[63:32]
Count leading zeroes 5 CLZ (cond) Rd, Rm	<u> </u>	Hi, Rm, Rs RdHi, Rm, Rs RdHi, Rm, Rs Rs Rs Rs Rs Rs, Rn Rs, Rn RdHi, Rm, Rs m, Rs, Rn n, Rs, Rn n, Rs, Rn n, Rs, Rn n, Rs, Rn r, RdLo, Rm, Rs m, Rs n, Rs, Rn r, RdLo, Rm, Rs m, Rs r, Rd, Rn r, Rd, Rn r, Rs, Rn r, Rd, Rs r, Rs, Rn r, Rs,					Rd := (Rm * Rs)[31:0] Rd := (Rm * Rs) + Rn)[31:0] RdHi,RdLo := unsigned(RdHi,RdLo + Rm * Rs) RdHi,RdLo := unsigned(RdHi+ RdLo + Rm * Rs) RdHi,RdLo := signed(RdHi+ RdLo + Rm * Rs) Rd := Rm[x] * Rs[y] Rd := Rn + Rm[x] * Rs[y] Rd := Rn + Rm[x] * Rs[y] Rd := Rn + Rm[15:0] * RsX[15:0] + Rm[31:16] * RsX[31:16] Rd := Rn + Rm[15:0] * RsX[15:0] + Rm[31:16] * RsX[31:16] Rd := Rn + Rm[15:0] * RsX[15:0] - Rm[31:16] * RsX[31:16] Rd := Rn + Rm[15:0] * RsX[15:0] - Rm[31:16] * RsX[31:16] Rd := Rn + Rm[15:0] * RsX[15:0] - Rm[31:16] * RsX[31:16] Rd := Rn + Rm[15:0] * RsX[15:0] - Rm[31:16] * RsX[31:16] Rd := Rn + Rm[15:0] * RsX[15:0] - Rm[31:16] * RsX[31:16] Rd := Rn + Rm[15:0] * RsX[15:0] - Rm[31:16] * RsX[31:16] Rd := Rn + Rm[15:0] * RsX[15:0] - Rm[31:16] * RsX[31:16] Rd := Rn + Rm[15:0] * RsX[15:0] - Rm[31:16] * RsX[31:16] Rd := Rn + Rm[15:0] * RsX[15:0] - Rm[31:16] * RsX[31:16] Rd := Rn + Rm[15:0] * RsX[15:0] - Rm[31:16] * RsX[31:16] Rd := Rn + Rm[15:0] * RsX[31:0] - Rm[31:16] * RsX[31:16] Rd := Rn + Rm[15:0] * RsX[31:0] - Rm[31:16] * RsX[31:16] Rd := Rn + Rm[15:0] * RsX[31:0] - Rm[31:16] * RsX[31:16] Rd := Rn + Rm[15:0] * RsX[31:0] - Rm[31:16] * RsX[31:16] Rd := Rn + Rm[15:0] * RsX[31:0] - Rm[31:16] * RsX[31:16] Rd := Rn + Rm[15:0] * RsX[31:0] - Rm[31:16] * RsX[31:16] Rd := Rn + Rm[15:0] * RsX[31:0] - Rm[31:16] * RsX[31:16] Rd := Rn + Rm[15:0] * RsX[31:16] + Rm[31:16] * RsX[31:16] Rd := Rn + Rm[15:0] * RsX[31:16] + Rm[31:16] * RsX[31:16] Rd := Rn + Rm[15:0] * RsX[31:16] + Rm[31:16] * RsX[31:16] Rd := Rn + Rm[15:0] * RsX[31:16] + Rm[31:16] * R

ARM Addressing Modes Quick Reference Card

	S	Assembler	Supdates	0	Η	ction
		<pre><prefix>ADD16(cond) Rd, Rn, Rm</prefix></pre>		$\frac{1}{2}$	GE R	GE Rd[31:16] := Rn[31:16] + Rm[31:16], Rd[15:0] := Rn[15:0] + Rm[15:0]
Halfword-wise subtraction	6	<pre><prefix>SUB16(cond) Rd, Rn, Rm</prefix></pre>		_	H R	GE Rd[31:16] := Rn[31:16] - Rm[31:16], Rd[15:0] := Rn[15:0] - Rm[15:0]
Byte-wise addition	6	<pre><prefix>ADD8{cond} Rd, Rn, Rm</prefix></pre>			E R	GE $Rd[31:24] := Rn[31:24] + Rm[31:24], Rd[23:16] := Rn[23:16] + Rm[23:16], Rd[15:8] := Rn[15:8] + Rm[15:8], Rd[7:0] := Rn[7:0] + Rm[7:0]$
Byte-wise subtraction	6	<pre><prefix>SUB8{cond} Rd, Rn, Rm</prefix></pre>		_		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	6	<pre><prefix>ADDSUBX{cond} Rd, Rn, Rm</prefix></pre>		_	E R	GE $ Rd[31:16] := Rn[31:16] + Rm[15:0], Rd[15:0] := Rn[15:0] - Rm[31:16]$
Halfword-wise exchange, subtract, add	9	<pre><prefix>SUBADDX{cond} Rd, Rn, Rm</prefix></pre>		_	Œ R	GE $Rd[31:16] := Rn[31:16] - Rm[15:0], Rd[15:0] := Rn[15:0] + Rm[31:16]$
Unsigned sum of absolute differences	6	USAD8{cond} Rd, Rm, Rs			+ 72	Rd := Abs(Rm[31:24] - Rs[31:24]) + Abs(Rm[23:16] - Rs[23:16]) + $Abs(Rm[15:8] - Rs[15:8]) + Abs(Rm[7:0] - Rs[7:0])$
and accumulate	6	USADA8{cond} Rd, Rm, Rs, Rn			+ R	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])
Move		MOV(cond)(S) Rd, <operand2></operand2>	N		R	Rd := Operand2
NOT		MVN(cond)(S) Rd, <operand2></operand2>			×	Rd := 0xFFFFFFFF EOR Operand2
PSR to register	ω	MRS(cond) Rd, <psr></psr>			×	Rd := PSR
register to PSR		MSR(cond) <psr>_<fields>, Rm</fields></psr>			P	PSR := Rm (selected bytes only)
immediate to PSK	3	•			٠.٠	PSR := immed_8r (selected bytes only)
register to 40-bit accumulator	ž ž	Mag(cond) ac pdro pay:		_	> 7	Ac[21-0] - Pall o Ac[20:20] - Pall:
Сору	6	CPY(cond) Rd, <operand2></operand2>			R	Rd := Operand2
Test		TST{cond} Rn, <operand2></operand2>	NZC		u	Update CPSR flags on Rn AND Operand2
Test equivalence		TEQ{cond} Rn, <operand2></operand2>		_	u	Update CPSR flags on Rn EOR Operand2
AND		AND[cond]{S} Rd, Rn, <operand2></operand2>			ফ	Rd := Rn AND Operand2
EOR		EOR(cond)(S) Rd, Rn, <operand2></operand2>			Z,	Rd := Rn EOR Operand2
ORR		ORR{cond}{S} Rd, Rn, <operand2></operand2>			Ŗ	Rd := Rn OR Operand2
Bit Clear		BIC(cond){S} Rd, Rn, <operand2></operand2>	NZC	F	Ŗ	Rd := Rn AND NOT Operand2
Compare		CMP{cond} Rn, <operand2></operand2>	Z C		ď	Update CPSR flags on Rn - Operand2
negative		CMN(cond) Rn, <operand2></operand2>	Z C		G	Update CPSR flags on Rn + Operand2
Signed saturate word, right shift	6	SSAT{cond} Rd, # <sat>, Rm{, ASR <sh>}</sh></sat>		0	Ŗ	Rd := SignedSat((Rm ASR sh), sat). <sat> range 0-31, <sh> range 1-32.</sh></sat>
left shift		SSAT{cond} Rd, # <sat>, Rm{, LSL <sh>}</sh></sat>		0	묫	Rd := SignedSat((Rm LSL sh), sat). < sat> range 0-31, < sh> range 0-31.
Signed saturate two halfwords	6	SSAT16{cond} Rd, # <sat>, Rm</sat>		0	אא	Rd[31:16] := SignedSat(Rm[31:16], sat), Rd[15:0] := SignedSat(Rm[15:0], sat), <sat> range 0-15.</sat>
Unsigned saturate word, right shift	6	USAT{cond} Rd, # <sat>, Rm{, ASR <sh>}</sh></sat>		0	77	Rd := UnsignedSat((Rm ASR sh), sat). <sat> range 0-31, <sh> range 1-32.</sh></sat>
left shift		USAT{cond} Rd, # <sat>, Rm{, LSL <sh>}</sh></sat>		0	R	Rd := UnsignedSat((Rm LSL sh), sat). <sat> range 0-31, <sh> range 0-31.</sh></sat>
Unsigned saturate two halfwords		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>
Operation Parallel Parallel arithmetic Compare Compare		Halfword-wise addition Halfword-wise subtraction Byte-wise addition 6 Byte-wise addition 6 Byte-wise subtraction 6 Halfword-wise exchange, add, subtract 6 Halfword-wise exchange, subtract, add Unsigned sum of absolute differences 6 Move NOT PSR to register PSR to register to PSR immediate to PSR immediate to PSR 7 register to 40-bit accumulator register to 40-bit accumulator Ropy Test register to 40-bit accumulator register to 40-bit accumulator register to 40-bit accumulator register to 40-bit accumulator foopy Test register to 40-bit accumulator register to 40-bit accumulator register to 40-bit accumulator footnote register to 40-bit accumulator register to 40-bit accumulator register to 40-bit accumulator register to 40-bit accumulator footnote register to 40-bit accumulator register to 40-bit accumulator register to 40-bit accumulator footnote register to 40-bit accumulator register to 40-bit accumulator footnote fo	Balfword-wise subtraction	Balfword-wise subtraction	S Assembler S Department S D	S Assembler S Application S Application S Application C C C C C C C C C

ARM Instruction Set Quick Reference Card

Flexic Pack half-word between + top Graph Concel Rd. Rb. Inc. Lat. Lat. Rd. 15(1)		None	5 NOP	No operation	No Op
Pack halfword to brottom + top	24-bit value encoded in instruction.	Software interrupt processor exception.	SWI{cond} <immed_24></immed_24>	Software interrupt	interrupt
Dack halfword buttons ++ top 6 PREETS (cond.) Rd., Pat., Pat., L. L.S., # https://docs.org/line.com/ Rd., Pat., Pat., L. L.S., # https://docs.org/line.com/ Rd., Pat.,	Cannot be conditional.	Prefetch abort or enter debug state.	L	Dicaspoili	Cott
Pack Lalfword toword	Cannot be conditional.	PC ≒ [Kn], CPSR ≒ [Kn + 4]	RFE <a_mode4l></a_mode4l>	Realization exception	
Packind/word bottom + top 6 PSEEST (cond.) Rd., Rm., Rm. (2625 4-62h.) Rd[3:0] = Rd[3:0] = Rdm 152, ah[3:1:0] = Rdm 152, ah[3:1:0] = Rdm 152, ah[3:1:0] = Rdm 152, ah[3:0] = Rdm	Cannot be conditional.	[R13m] := R14, [R13m + 4] := CPSR		Store return state	
Packindiwood bottom + top					
Pack halfword bottom + top 6 PRIST (cond.) Rd., Rn. (, LSL # <ahra) 0-31.="" ah)[31:16]="" at="" lsl="" rd[150]="Rd[150]," rd[1516]="(Rm" td="" ="" <=""><td>Cannot be conditional.</td><td>Sets endianness for loads and saves.</td><td></td><td>Set endianness</td><td></td></ahra)>	Cannot be conditional.	Sets endianness for loads and saves.		Set endianness	
Pack halfword buton + top 6 PRIST(cond) Rd, Rn, Rn(, LSL # <ahr> Pack halfword buton + top 6 PRIST(cond) Rd, Rn, Rn, Rd, ASR #<ahr> Pack halfword to word 6 EXTENS(cond) Rd, Rn, Rn, Rn #<ahr> Two bytes to halfwords 6 EXTENS(cond) Rd, Rn, Rn(, ROR #<ahr> Two bytes to halfwords 6 EXTENS(cond) Rd, Rn, Rn(, ROR #<ahr> Two bytes to halfwords 6 EXTENS(cond) Rd, Rn, Rn(, ROR #<ahr> Byte to word 6 EXTENS(cond) Rd, Rn, Rn(, ROR #<ahr> Byte to word 6 EXTENS(cond) Rd, Rn, Rn(, ROR #<ahr> Byte to word 6 EXTENS(cond) Rd, Rn, Rn(, ROR #<ahr> Byte to word Add Cond) Rd, Rn, Rn(, ROR #<ahr> Byte to word, add Two bytes to halfwords 6 EXTENS(cond) Rd, Rn, Rn(, ROR #<ahr> Byte to word, add Two bytes to halfwords, add 6 EXTENS(cond) Rd, Rn, Rn(, ROR #<ahr> Byte to word, add Two bytes to halfwords, add 6 EXTENS(cond) Rd, Rn, Rn(, ROR #<ahr> Byte to word, add Two bytes to halfwords, add 6 EXTENS(cond) Rd, Rn, Rn(, ROR #<ahr> Byte to word, add CXTENS(cond) Rd, Rn, Rn(, ROR #<ahr> Byte to word, add Two bytes to halfwords, add 6 EXTENS(cond) Rd, Rn, Rn(, ROR #<ahr> Byte to word, add CXTENS(cond) Rd, Rn, Rn(, ROR #<ahr> Byte to word, add CXTENS(cond) Rd, Rn, Rn(, ROR #<ahr> Byte to word, add 6 EXTENS(cond) Rd, Rn, Rn(, ROR #<ahr> Byte to word, add 6 EXTENS(cond) Rd, Rn, Rn(, ROR #<ahr> Byte to word, add 6 EXTENS(cond) Rd, Rn, Rn(, ROR #<ahr> Byte to word, add 6 EXTENS(cond) Rd, Rn, Rn(, ROR #<ahr> Byte to word, add 6 EXTENS(cond) Rd, Rn, Rn(, ROR #<ahr> Byte to word, add 6 EXTENS(cond) Rd, Rn, Rn(, ROR #<ahr> Byte to word, add 6 EXTENS(cond) Rd, Rn, Rn(, ROR #<ahr> Byte to word, Rn(, Rn(, Rn(, Rn)) Rn(, Rn(, Rn)) Rn(, Rn(, Rn(, Rn(, Rn(, Rn(, Rn(, Rn(,</ahr></ahr></ahr></ahr></ahr></ahr></ahr></ahr></ahr></ahr></ahr></ahr></ahr></ahr></ahr></ahr></ahr></ahr></ahr></ahr></ahr></ahr></ahr></ahr></ahr>	Cannot be conditional.			Change processor mode	e i i i i
Pack halfword bottom + top 6 PKHEFT Cond Rd , Rn , Rm LEL #4.ab> Rd[150] = Rd[150] = Rd[150] = Rd[150] = Rdm ASR sh[150] sh 0-31. Pack halfword bottom 6 PKHEFT Cond Rd , Rm Rm Rd Rd[3110] = Rdm Rd[3110] = Rdm Rd[310] =	Cannot be conditional.	Enable specified interrups, optional change mode.	CPSIE <iflags> {,</iflags>		chance
Pack halfword bottom + top 6 PRIEST Cond.] Rd., Rn., Rm[, LSL #cab>] Rd[15:0] = Rad[15:0] = Rd[11:0] = Rd[11:0] = (Rm LSL sh)[31:0] = do.3.]	Cannot be conditional.	Disable specified interrups, optional change mode.	CPSID <iflags> {,</iflags>	Change processor state	Processor
Pack halfword bottom + top 6 PREPT (cond.) Rd., Em. Rd[15.0] = Rd[15.0], Rd[15.0] = Rd		Change to Java state	SI, 6 BXJ {cond} Rm		
Pack halfword bottom + top		on, R15 := Rm[3	BLX(cond)	with link and exchange (2)	_
Pack halfword bottom + top	label must be within ±3; of current instruction.				
Pack halfword bottom + top 6 PRIEPT (cond.) Rd., Rn., Rm(, LSL. #-csh.) Rd[31:16] = Rm[15:0], Rd[31:16] = (Rm LSL. sh)[31:16], sh 0-31. Pack halfword to word 6 SETTE (cond.) Rd., Rm., Rm(, ASR. #-csh.) Rd[31:16] = SignExtend((Rm ROR (8 * sh))[7:30), sh 0-3. Rd[31:16] = SignExtend((Rm	Cannot be conditional.		ST BIX label		
Pack halfword bottom + top 6 EXCHET (cond.) Rd., Rm., Rm., Rm., Rm., Rm., Rm., Rd.]3:16] = Rm.[15:0], Rd.[3:0] = (Rm.LSL.sh)[3:16], sh.0-31. Rd., Rm., Rm., Rm., Rm., Rd.]3:16] = Rm.[3:16], Rd.[15:0] = (Rm.ASR.sh)[15:0], sh.0-3. Rd.]3:16] = SignExtend((Rm.ROR. (8 * sh))[15:0]), sh.0-3. Rd.]3:16] = SignExtend((Rm.ROR. (8 * sh))[15:0]), sh.0-3. Rd.]3:16] = SignExtend((Rm.ROR. (8 * sh))[25:10]), sh.0-3. Rd., Rm., Ror. # <sh.> Rd., Rd.]3:16] = SignExtend((Rm.ROR. (8 * sh))[25:10]), sh.0-3. Rd., Rm., Ror. #<sh.> Rd., Rd.]3:16] = SignExtend((Rm.ROR. (8 * sh))[25:10]), sh.0-3. Rd., Rm., Ror. #<sh.> Rd., Rd.]3:16] = SignExtend((Rm.ROR. (8 * sh))[25:10]), sh.0-3. Rd., Rm., Ror. #<sh.> Rd., Rd.]3:16] = SignExtend((Rm.ROR. (8 * sh))[25:10]), sh.0-3. Rd., Rm., Ror. #<sh.> Rd., Rd.]3:16] = SignExtend((Rm.ROR. (8 * sh))[25:10]), sh.0-3. Rd., Rm., Ror. #<sh.> Rd., Rd.]3:16] = SignExtend((Rm.ROR. (8 * sh))[25:10]), sh.0-3. Rd., Rm., Ror. #<sh.> Rd., Rd.]3:16] = SignExtend((Rm.ROR. (8 * sh))[25:10]), sh.0-3. Rd., Rm., Ror., Ror., Ror. #<sh.> Rd., Rd.]3:16] = SignExtend((Rm.ROR. (8 * sh))[25:10]), sh.0-3. Rd., Rm., Rm., Ror., Rm., Ror. #<sh.> Rd., Rd.]3:16] = SignExtend((Rm.ROR. (8 * sh))[25:10]), sh.0-3. Rd., Rm., Rm., Ror., Rm., Ror., Ror., Rd., Rd., Rd.]3:16] = SignExtend((Rm.ROR. (8 * sh))[25:10]), sh.0-3. Rd., Rm., Rm., Ror., Rm., Ror., Rm., Rd.]3:16] = SignExtend((Rm.ROR. (8 * sh))[25:10]), sh.0-3. Rd., Rm., Rm., Ror., Rm., Ror., Rm., Rd.]3:16] = SignExtend((Rm.ROR. (8 * sh))[25:10]), sh.0-3. Rd., Rm., Rm., Ror., Rm., Ror., Rm., Rd., Ror., Rm., Rd.]3:16] = SignExtend((Rm.ROR. (8 * sh))[25:10]), sh.0-3. Rd., Rm., Rm., Ror., Rm., Rm., Ror., Rm., Rd.]3:16] = SignExtend((Rm.ROR. (8 * sh))[25:10]), sh.0-3. Rd., Rm., Rm., Ror., Rm., Rm., Ror., Ror., Rm., Rd.]3:16] = SignExtend((Rm.ROR. (8 * sh))[25:10]), sh.0-3. Rd., Rm., Rm., Ror., Rm., Rm., Ror., Ror., Rm., Rd.]3:16] = Rd.]3:16] + SignExtend((Rm.ROR. (8 * sh))[25:10]), sh.0-3.</sh.></sh.></sh.></sh.></sh.></sh.></sh.></sh.></sh.>	of current instruction.	R14 := address of next instruction, R15 := label	Bi{cond} Label		
Pack halfword bottom + top 6 PRHBT(cond) Rd, Rn, Rm(, LSL #csh>) Rd[31:6] = Rm[31:16] = (Rm LSL sh)[31:16].sh 0-31.	of current instruction.	NY := MORT	a count toner	. A Thirty of the same of the	
Pack halfword bottom + top 6 PKHFT (cond.) Rd., Rn., Rm(, LSL # <sh>) Rd[15:0] := Rd[15:0], Rd[3:16] := (Rm LSL sh)[3:16], sh 0-31. </sh>		$ \mathbf{x}\mathbf{u} _{1}^{2}$	Bigordi labal	Branch	Branch
Pack halfword bottom + top 6 PRHSP{ cond Rd Rn Rm LSL		1	REVSH(cond) Ra, Kn	Select before	Coloct
Pack halfword bottom + top 6 PEHBT cond Rd Rn Rm LSL		Rd[15:8] := Rm[7:0], Rd[7:0] := Rm[15:8], Rd[31:24] := Rm[23:16], Rd[23:16] := Rm[31:24]		In both halfwords	
Pack halfword bottom + top 6 PRHBT cond Rd , Rn , Rm LS1 + <sh> Rd[15:0] := Rn[11:16], Rd[15:0] := (Rm LSL sh)[15:0] sh 0-31. </sh>		$Rd[31:24] := Rm[7:0], Rd[23:16] := Rm[15:8], \\ Rd[15:8] := Rm[23:16], Rd[7:0] := Rm[31:24]$		in word	bytes
Pack halfword bottom + top 6 PKHET { cond } Rd , Rn , Rm { , LSL # <sh> } Rd[15:0] = Rn[15:0], Rd[31:16] = (Rm LSL sh)[31:16], sh 0-31. </sh>			UXTAB{cond} Rd, Rn, Rm{,	Byte to word, add	
Pack halfword bottom + top 6 PKHET { cond } Rd , Rn , Rn { , LSL # <sh> } Rd[15:0] := Rn[15:0], Rd[31:16] := (Rm LSL sh)[31:16], sh 0-31. </sh>		Rd[31:16] := Rn[31:16] + ZeroExtend((Rm ROR (8 * sh))[23:16]), Rd[31:16] := Rn[15:0] + ZeroExtend((Rm ROR (8 * sh))[23:16]), Rd[15:0] := Rn[15:0] + ZeroExtend((Rm ROR (8 * sh))[7:0]), sh 0-3.		Two bytes to halfwords, add	extend with add
Pack halfword bottom + top 6 PKHBT { cond } Rd , Rn , Rm { , LSL # <sh> } Rd[15:0] = Rn[15:0], Rd[31:16] := (Rm LSL sh)[31:16], sh 0-31. Pack halfword top + bottom 6 PKHBT { cond } Rd , Rn , Rm { , ASR #<sh> } Rd[31:16] := Rn[31:16], Rd[15:0] := (Rm ASR sh)[15:0], sh 0-3. d</sh></sh>		Rd[31:0] := Rn[31:0] + SignExtend((Rm ROR (8 * sh))[7:0]), sh 0-3.	SXTAB(cond) Rd, Rn,	Halfword to word add	Insigned
Pack halfword bottom + top 6 PKHET { cond } Rd , Rn , Rn { , LSL # <sh> } Rd[15:0] := Rn[15:0], Rd[31:16] := (Rm LSL sh)[31:16], sh 0-31. </sh>		Rd[31:16] := Rn[31:16] + SignExtend((Rm ROR (8 * sh))[73:16]), Rd[15:0] := Rn[15:0] + SignExtend((Rm ROR (8 * sh))[7:0]). sh 0-3.		Two bytes to halfwords, add	with add
Pack halfword bottom + top 6 PKHET { cond } Rd , Rn , Rn { , LSL # <sh> } Rd[15:0] := Rn[15:0], Rd[31:16] := (Rm LSL sh)[31:16], sh 0-31. </sh>		Rd[31:0] := Rn[31:0] + SignExtend((Rm ROR (8 * sh))[15:0]). sh 0-3.		Halfword to word, add	Signed
Pack halfword bottom + top 6 PKHET { cond } Rd , Rn , Rm { , LSL # <sh> } Rd[15:0] = Rn[15:0], Rd[31:16] = (Rm LSL sh)[31:16], sh 0-31.</sh>		Rd[31:0] := ZeroExtend((Rm ROR (8 * sh))[7:0]). sh 0-3.	_	Byte to word	
Pack halfword bottom + top 6 PKHBT {cond} Rd, Rn, Rm{, LSL # <sh>} Rd[15:0] := Rn[15:0], Rd[31:16] := (Rm LSL sh)[31:16], sh 0-31. </sh>		Rd[31:16] := ZeroExtend((Rm ROR (8 * sh))[23:16]), Rd[15:0] := ZeroExtend((Rm ROR (8 * sh))[7:0]). sh 0-3.		Two bytes to halfwords	extend
Pack halfword bottom + top 6 PKHBT {cond} Rd, Rn, Rm{, LSL # <sh>} Rd[15:0] := Rn[15:0], Rd[31:16] := (Rm LSL sh)[31:16], sh 0-31. </sh>		[Rd[31:0] := ZeroExtend((Rm ROR (8 * sh))[15:0]). sh 0-3.		Halfword to word	Unsigned
Pack halfword bottom + top 6 PKHBT {cond} Rd, Rn, Rm{, LSL # <sh>} Rd[15:0] := Rn[15:0], Rd[31:16] := (Rm LSL sh)[31:16], sh 0-31.</sh>		Rd[31:0] := SignExtend((Rm ROR (8 * sh))[7:0]). sh 0-3.	SXTB(cond) Rd, Rm(,	Byte to word	
Pack halfword bottom + top 6 PKHET { cond } Rd , Rn , Rm { , LSL # <sh>} Rd[15:0] = Rn[15:0], Rd[31:16] = (Rm LSL sh)[31:16], sh 0-31.</sh>		Rd[31:16] := SignExtend((Rm ROR (8 * sh))[23:16]), Rd[15:0] := SignExtend((Rm ROR (8 * sh))[7:0]). sh 0-3.		Two bytes to halfwords	extend
Pack halfword bottom + top 6 PKHBF cond Rd, Rn, Rm (, LS1 # <sh> Rd[15:0] = Rn[15:0], Rd[31:16] = (Rm LSL sh)[31:16], sh 0-31. Pack halfword top + bottom 6 PKHBF cond Rd, Rn, Rm (, LS1 #<sh> Rd[31:16] = Rn[31:16], Rd[15:0] = (Rm ASR sh)[15:0], sh 1-32.</sh></sh>		Rd[31:0] := SignExtend((Rm ROR (8 * sh))[15:0]). sh 0-3.	SXTH(cond) Rd, Rm{, ROR +	Halfword to word	Signed
Pack halfword bottom + top 6 FKHET (cond.) Rd., Rn., Rn.(, LSL # <sh> Rd15:01 = Rn[15:0], Rd[31:16] = (Rm I Sl., sh/31:16], sh 0.31</sh>		Rd[31:16] := Rn[31:16], Rd[15:0] := (Rm ASR sh)[15:0], sh 1-32.	PKHTB{cond} Rd, Rn, Rm{,	Pack halfword top + bottom	
	10000	Rd[15:0] := Rn[15:0], Rd[31:16] := (Rm LSL sh)[31:16], sh (1-31)	6 PKHBT {cond} Rd, Rn, Rmf,		Pack

ARM Addressing Modes Quick Reference Card

		1			
Operation		ø	Assembler	Action	Notes
Load	Word		LDR{cond} Rd, <a_mode2></a_mode2>	Rd := [address]	Rd must not be R15.
	User mode privilege		LDR{cond}T Rd, <a_mode2p></a_mode2p>		Rd must not be R15.
	branch (§ 5T: and exchange)		LDR{cond} R15, <a_mode2></a_mode2>	;	
	Byte		LDR(cond)B Rd. <a mode2="">	Rd := ZeroExtendThyte from address	Rd must not be R15
	User mode privilege		LDR {cond}BT Rd, <a_mode2p></a_mode2p>		Rd must not be R15.
	signed	4	IDR{cond}SB Rd, <a_mode3></a_mode3>	Rd := SignExtend[byte from address]	Rd must not be R15.
	Halfword	4	LDR(cond)H Rd, <a_mode3></a_mode3>	Rd := ZeroExtent[halfword from address]	Rd must not be R15.
	signed	4	LDR(cond)SH Rd, <a_mode3></a_mode3>		Rd must not be R15.
	Doubleword	SE*	SE* LDR(cond)D Rd, <a_mode3></a_mode3>	Rd := [address], R(d+1) := [address + 4]	Rd must be even, and not R14.
Load multiple	Pop, or Block data load		<pre>LDM(cond)<a_mode4l> Rn(!), <reglist-pc></reglist-pc></a_mode4l></pre>		
	return (and exchange)		LDM(cond) <a_mode4l> Rn{!}, <reglist+pc></reglist+pc></a_mode4l>	Load registers, R15 := [address][31:1] (8 ST: Change to Thumb if [address][0] is 1)	
	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 only.
	User mode registers			Load list of User mode registers from [Rn]	Use from privileged modes only
Soft preload	Memory system hint	SE*	SE* PID <a_mode2></a_mode2>		Cannot be conditional.
Load exclusive	Semaphore operation	6	LDREX{cond} Rd, [Rn]	_	Rd, Rn must not be R15.
Store	Word		STR{cond} Rd, <a_mode2></a_mode2>	[address] := Rd	
	User mode privilege		STR{cond}T Rd, <a_mode2f></a_mode2f>	[address] := Rd	
	Byte		STR{cond}B Rd, <a_mode2></a_mode2>	[address][7:0] := Rd[7:0]	
	User mode privilege		STR {cond}BT Rd, <a_mode2p></a_mode2p>	[address][7:0] := Rd[7:0]	
	Halfword	4	STR{cond}H Rd, <a_mode3></a_mode3>	[address][15:0] := Rd[15:0]	
	Doubleword	5E*	STR{cond}D Rd, <a_mode3></a_mode3>	[address] := Rd, [address + 4] := $R(d+1)$	Rd must be even, and not R14.
Store multiple	Push, or Block data store		}, <reglist></reglist>	Store list of registers to [Rn]	
	User mode registers			Store list of User mode registers to [Rn]	Use from privileged modes only.
Store exclusive	Store exclusive Semaphore operation	6	STREX{cond} Rd, Rm, [Rn]		Rd, Rm, Rn must not be R15.
Swap	Word	3	SWP(cond) Rd, Rm, [Rn]	temp := [Rn], [Rn] := Rm, Rd := temp	
	Byte	3	SWP(cond)B Rd, Rm, [Rn]	temp := $ZeroExtend([Rn][7:0])$,	
				[kn][/:U] := km[/:U], Kd := temp	

Quick Reference Card ARM Addressing Modes

Addressing	Mode 2 - Word and (Addressing Mode 2 - Word and Unsigned Byte Data Transfer		ARM architecture versions	and versions
Pre-indexed	Immediate offset	[Rn, #+/- <immed_12>]{!}</immed_12>		n	ARM architectu
	Zero offset	[Rn]	Equivalent to [Rn,#0]	nT, nJ	T or J variants o
	Register offset	[Rn, +/-Rm] {!}		×	ARM architectu
	Scaled register offset	[Rn, +/-Rm, LSL # <shift>] {1} Allowed shifts 0-31</shift>	Allowed shifts 0-31	πE	All E variants of
		[Rn, +/-Rm, LSR # <shift>] {!} Allowed shifts I-32</shift>	Allowed shifts 1-32	иЕ*	E variants of AF
		[Rn, +/-Rm, ASR # <shift>] [1] Allowed shifts 1-32</shift>	Allowed shifts 1-32	XS	XScale coproces
		[Rn, +/-Rm, ROR # <shift>] {!} Allowed shifts 1-31</shift>	Allowed shifts 1-31		
		[Rn, +/-Rm, RRX] {!}		Flexible Operand 2	nd 2
Post-indexed	Immediate offset	[Rn], #+/~ <immed_12></immed_12>		Immediate value	
	Register offset	[Rn], +/-Rm		Logical shift left immediate	immediate
	Scaled register offset	[Rn], +/~Rm, LSL # <shift></shift>	Allowed shifts 0-31	Logical shift right immediate	t immediate
		[Rn], +/-Rm, LSR # <shift></shift>	Allowed shifts 1-32	Arithmetic shift right immediate	ight immediate
		[Rn], +/-Rm, ASR # <shift></shift>	Allowed shifts 1-32	Rotate right immediate	ediate
		[Rn], +/-Rm, ROR # <shift></shift>	Allowed shifts 1-31	Register	
		[Rn], +/-Rm, RRX		Rotate right extended	ıded
				+	

Mouressing N	Addressing Mode z (Post-indexed on	ea only)				
Post-indexed	Post-indexed Immediate offset	[Rn],	#+/- <immed_12></immed_12>		12>	
	Zero offset	[Rn]				Equivalent to [Rn],#0
	Register offset	[Rn],	[Rn], +/-Rm			
	Scaled register offset [Rn], +/-Rm, LSL # <shift:< td=""><td>[Rn],</td><td>+/-Rm,</td><td>TST</td><td>#<shift></shift></td><td>Allowed shifts 0-31</td></shift:<>	[Rn],	+/-Rm,	TST	# <shift></shift>	Allowed shifts 0-31
		[Rn] ,	+/-Rm,	LSR	[Rn], +/-Rm, LSR # <shift></shift>	Allowed shifts 1-32
		[Rn],	+/-Rm,	ASR	[Rn], +/-Rm, ASR # <shift></shift>	Allowed shifts 1-32
		[Rn],	+/-Rm,	ROR	[Rn], +/-Rm, ROR # <shift></shift>	Allowed shifts 1-31
i		[Rn],	[Rn], +/-Rm, RRX	RRX		

Post-indexed	Post-indexed Immediate offset	[Rn],	[Rn], #+/- <immed_12></immed_12>	_med_	12>	
	Zero offset	[Rn]				Equivalent to [Rn],#0
	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	[Rn], +/-Rm, LSR # <shift></shift>	Allowed shifts 1-32
		[Rn],	+/-Rm,	ASR	+/-Rm, ASR # <shift></shift>	Allowed shifts 1-32
		[Rn],	+/-Rm,	ROR	[Rn], +/-Rm, ROR # <shift></shift>	Allowed shifts 1-31
		[Rn],	[Rn], +/-Rm, RRX	RRX		

Pre-indexed

Immediate offset

Zero offset

臣

[Rn, #+/-<immed_8>]{!}

Equivalent to [Rn,#0]

Condition Field
Mnemonic

Description

Suffix c f s

Control field mask byte Flags field mask byte

PSR[7:0] PSR[31:24] PSR[23:16] PSR[15:8]

(use at least one suffix)
Meaning

Extension field mask byte Status field mask byte

CC / TO SH / SD

Carry Clear / Unsigned lower

Carry Set / Unsigned higher or same

Greater than or equal, or unordered Less than Not equal, or unordered

Equa

Description (VFP)

an Oa

Not equal

Register Immediate offset

[Rn], +/-Rm]{!} [Rn], #+/-<immed_8> [Rn], +/-Rm

Register

Addressing Mode 3 - Haltword, Signed Byte, and Doubleword Data Transfer

								Logical shift l
ddressing	ddressing Mode 2 (Post-indexed only	d only)						Logical shift r
ost-indexed	ost-indexed Immediate offset	[Rn],	[Rn], #+/- <immed_12></immed_12>	med	12>			Arithmetic shi
	Zero offset	[Rn]				Equivalent to [Rn],#0		Rotate right re
	Register offset	[Rn],	+/-Rm					
	Scaled register offset [Rn], +/-Rm, LSL # <shift></shift>	[Rn],	+/-Rm,	TST	# <shift></shift>	Allowed shifts 0-31		PSR fields
		[Rn] ,	+/-Rm,	LSR	+/-Rm, LSR # <shift></shift>	Allowed shifts 1-32		Suffix
		[Rn]	+/-Rm,	ASR	+/-Rm, ASR # <shift></shift>	Allowed shifts 1-32	_	c
		[Rn],	+/-Rm,	ROR	[Rn], +/-Rm, ROR # <shift></shift>	Allowed shifts 1-31		н
		[Rn],	[Rn], +/-Rm, RRX	RRX				ហ

	Immediate value	# < 1	# <immed 8r=""></immed>	8r>	
	Logical shift left immediate	Rm,	ISI	Rm, LSL # <shift></shift>	Allowed shifts
shifts 0-31	Logical shift right immediate	Rm,	LSR	LSR # <shift></shift>	Allowed shifts
shifts 1-32	Arithmetic shift right immediate	RII,	ASR	ASR # <shift></shift>	Allowed shifts
shifts 1-32	Rotate right immediate	Ħ,	ROR	ROR # <shift></shift>	Allowed shifts
shifts 1-31	Register	Rm			
	Rotate right extended	Rm,	RRX		
	Logical shift left register	Rm,	TST	Rs	
	Logical shift right register	Rm,	LSR	Rs	
	Arithmetic shift right register	Ra,	ASR	Rs	
11 to [Rn],#0	Rotate right register	Rm,	Rm, ROR Rs	Rs	

п	ARM architecture version n and above	ion n e	ınd ab	ove.		
nT, nJ	T or J variants of ARM architecture version n and above.	[archi	ecture	e version n and above	ē.	
×	ARM architecture version 3M, and 4 and above, except xM variants.	ion 31v	f, and	4 and above, except	txM variants.	
лE	All E variants of ARM architecture version n and above.	archit	ecture	version n and abov	p	
πE*	E variants of ARM architecture version n and above, except xP variants.	hitectu	ге уег	sion n and above, ex	cept xP variants.	
XX	XScale coprocessor instruction	tructio	š			
Flexible Operand 2	and 2					
Immediate value	,	<18_pemmr>#	med	8r>		
Logical shift left immediate	immediate	Rm,	TST	Rm, LSL # <shift></shift>	Allowed shifts 0-31	
Logical shift right immediate	ıt immediate	Rm,	LSR	LSR # <shift></shift>	Allowed shifts 1-32	
Arithmetic shift right immediate	right immediate	Rm,	ASR	ASR # <shift></shift>	Allowed shifts 1-32	
Rotate right immediate	lediate	Rm,	ROR	ROR # <shift></shift>	Allowed shifts 1-31	
Register		Rm				
Rotate right extended	nded	Rm,	RRX			
Logical shift left register	register	Rm,	LSL Rs	Rs		
Logical shift right register	it register	Rm,	LSR Rs	Rs		
Arithmetic shift right register	right register	R'n,	ASR Rs	Rs		

(ne) (o wit come matical)	I Inindexed No offeet	
[Rn], #+/- <immed_8*4></immed_8*4>	Post-indexed Immediate offset	Post
[Rn] Equivalent to [Rn,#0]	Zero offset	
[Rn, #+/- <immed_8*4>]{i}</immed_8*4>	Pre-indexed Immediate offset	Pre-
r Data Transfer	Addressing Mode 5 - Coprocessor Data Transfer	Ado
]
FD Full Descending	B Decrement Before	BG
ED Empty Descending	A Decrement After	DA
FA Full Ascending	B Increment Before	IB
EA Empty Ascending	A Increment After	IA
Stack push	Block store	B
EA Empty Ascending	B Decrement Before	BG
FA Full Ascending	A Decrement After	DA
ED Empty Descending	B Increment Before	ㅂ
FD Full Descending	A Increment After	IA
Stack pop	Block load	
a Transfer	dressing Mode 4 - Multiple Da	Add
	🖼	Increment After

A ways (normanly ountied)		TE GE GE VE
s than Less than, or unordered ater than Greater than Less than or equal Less than or equal, or unordered Always (normally omitted)	H O H reflix	Positive or zero Overflow No overflow Unsigned higher Unsigned lower or same Signed greater than or equal
8	less than l greater than l greater than l less than or equal less than	Sign
a)	I greater than I less than or equal I less than or	Sig
	ways (normally omitted) Prefix Prefix	Si
Always (normally omitted) Always (normally omitted)	er S Prefix S Prefix S S Past Interrupt S S Partisor U U U U U U U U U U U U U U U U U U U	S
	er S S Q Q Drast Interrupt Q Q Interrupt U U Out out UQ UH	$\overline{}$
	ast Interrupt Q sterrupt SH visor U u u u u u u u u u u u u u u u u u u u	
	itempt SH visor U U U UH	
S Q	visor U UQ ined UH	
SH Q	ined UQ	
terrupt Q terrupt SH	댎	
ast Interrupt Q nterrupt SH visor U		-

Unordered (at least one NaN operand) Greater than or equal, or unordered

Less than or equal, or unordered

				трt		
댎	ğ	d	HS	Ø	(O	Prefi
Unsigned arithmetic, halving results	Unsigned saturating arithmetic	Unsigned arithmetic modulo 28 or 216, sets CPSR GE bits	Signed arithmetic, halving results	Signed saturating arithmetic	Signed arithmetic modulo 28 or 216, sets CPSR GE bits	Prefixes for Parallel Instructions

Quick Reference Card ARM Addressing Modes

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

Proprietary Notice

Words and logos marked with [®] or [™] 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.

ARM QRC 0001H **Document Number**

Change Log

д Берерия Бе June 1995 Sept 1996 Nov 1998 Oct 1999 Oct 2000 Sept 2001 Jan 2003 Oct 2003 CKS CKS BHH BHY Ghange
First Release
Second Release
Third Release
Thurd Release
Fourth Release
Fifth Release
Sixth Release
Seventh Release
Eighth Release