UNIVERSITY OF DUBLIN TRINITY COLLEGE

Faculty of Engineering, Mathematics and Science

School of Computer Science & Statistics

Integrated Computer Science Year 2 Examination Trinity Term 2014

Microprocessor Systems

Friday May 9, 2014

RDS Main Hall

14:00-16:00

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:

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

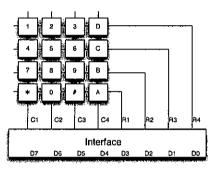
Non-programmable calculators are permitted for this examination — please indicate the make and model of your calculator on each answer book used.

- 1. (a) Explain how a four-stage pipeline architecture works. What factors prevent a pipeline running at full speed? How can they be addressed? [6 marks]
 - (b) List the components of the standard *memory hierarchy*. Why is there a memory hierarchy? [5 marks]
 - (c) Describe how cache memories are organised. [5 marks]
 - (d) How would you go about estimating how long a program would take to execute? How would the components of the memory hierarchy affect your estimates?

 [4 marks]

2. (a) Explain the difference between polling and interrupts. Why is polling sometime better? Why are interrupts sometimes better? [5 marks]

Imagine you have a 16-key keypad (see diagram) where the the keys are arranged as a four-by-four square, interfaced to your computer at location 0xE0F05204, providing a 2-of-8 code for each key. When a key is pressed, the value of its row line and its column line, which are normally 1, are pulled down to 0.



- (b) Explain what a lookup table is and how you would use it to encode the relationship between the binary patterns presented on the interface above when a key is pressed, to the ASCII code of the character printed on the keytop. [5 marks]
- (c) When a key on the keyboard shown above is pressed, it may *bounce*; that is, its state may change rapidly for about 5 ms before finally settling down its true value.

Write a polling subroutine that reliably returns, in RO, the ASCII code of a key when it is pressed. Explain clearly how your subroutine works. *Note*, your subroutine should work when the key is pressed, not when it is released. [10 marks]

- 3. (a) What are the different modes of operation of the ARM? What are they for? Why does the FIQ mode have its own copy of some of the registers?

 [2 marks]
 - (b) Explain the difference between vectored and non-vectored interrupt handling. Which is better? [4 marks]
 - (c) Design, document and write an interrupt handler which is called by a timer every millisecond and which monitors a one-bit input at bit 5 or location 0x40567884. The value of the input is normally zero but sometimes goes to one. You interrupt handler has to record the length of the longest time for which the input is continuously high and store its value in milliseconds at a location in RAM identified by the label LONGEST_HIGH_TIME.

Do not attempt to write the timer initialisation code. Assume it has already been set up—your code doesn't have to do any further setup. Just write a comment in your code where you would finally enable interrupts when everything else in your code is ready.

Likewise, assume the interface at location 0x40567884 has already been set up. [14 marks]

© UNIVERSITY OF DUBLIN 2014

ASCII Code

		n Numbe						
Row Number	000	001	010	011	100	101	110	111
0000	NUL	DLE	\Diamond	0	@	P	•	p
0001	SOH	DCI	!	1	Α	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	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BELL	ETB	1	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	\	1	
1101	CR	GS	-	=	M]	m	}
1110	SO	RS	•	>	N	٨	n	~
1111	SI	US	/	?	0	_	o	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 ASCIÎ 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.

ARM® Instruction Set Quick Reference Card

Refer to Table Pocessor Status Register) or SPSR (Saved Processor Status Register) Refer to Table PSR floids. Refer to Table PSR floids sending and PsR value by parallel adds and subtracts. Refer to Table PSR floids por Traeming [51:16]. A 23-bit constant, formed by ptgh-to-conding an 4-bit value by an even number of bits. Refer to Table PSR forestor (150), or Traeming [51:16]. Refer to Table PSR forestor Flooding an 4-bit value by an even number of bits. Refer to Table PSR forestor flooding and 4-bit value by an even number of bits. Refer to Table PSR forestor flooding and 4-bit value by an even number of bits. Refer to Table PSR forestor flooding and 4-bit value by an even number of bits. Refer to Table PSR forestor flooding and 4-bit value by an even number of bits. Refer to Table PSR forestor flooding and 4-bit value by an even number of bits. Refer to Table PSR forestor flooding and 4-bit value by an even number of bits. Refer to Table PSR forestor flooding and 4-bit value by an even number of bits. Refer to Table PSR forestor flooding and 4-bit value by an even number of bits. Refer to Table PSR forestor flooding and 4-bit value by and leave the processor floodes. Refer to Table PSR forestor flooding and 4-bit value by and even number of bits. Refer to Table PSR forestor flooding and 4-bit value by and even number of bits. Refer to Table PSR forestor flooding and 4-bit value by and even number of bits. Refer to Table PSR forestor flooding and 4-bit value processor floodes. Refer to Table PSR forestor flooding and 4-bit value processor floodes. Refer to Table PSR forestor flooding and floodi	Key	Key to Tables				\overline{a}	end:	anr	{endianness} (Can be BE (Big Endian) or LE (Little Endian).
Refer to Table Foxible Operand 2 Shift and totate are only available as part of Operand2. Refer to Table Foxible Operand 2 Shift and totate are only available as part of Operand2. Refer to Table Foxible Shift gas a control of the Shift gas and totate are only available as part of Operand2. Refer to Table Foxible Shift gas (Saved Processor Status Register) Fox of Caracter than of Earling Shift gas (Saved Processor Status Register) Refer to Table Foxible Shift gas (Saved Processor Status Register) Refer to Table Foxible Shift gas (Saved Processor Status Register) Refer to Table Foxible Shift gas (Averys updated by parelled shift gas shift gas (Saved Processor Modes) Refer to Table Foxible Shift gas (Averys updated by parelled shift gas (Averys updated by gas (Cond.) (S) Rd., Rn., cOperand2> N Z C V Rd. = R. Refer to Table Foxessor Modes Ref	9	nd)	Refer to Table Condition Field.	Omit for unconditional execution.		٨	a ≣	ode2		Refer to Table Addressing Mode 2.
Either CFSR (Current Processor Stams Register) or SFSR (Swed Processor Stams Register) c_m_models> c_m_models c_m_models> c_m_models	Ç	erand2>	Refer to Table Flexible Operan	d 2. Shift and rotate are only available as part of Operan	<u>1</u> 2	۸	ĭ 1	ode2		Refer to Table Addressing Mode 2 (Post-indexed only).
Comparison Com		elds>	Refer to Table PSR fields.			_	a	ode3	`	Refer to Table Addressing Mode 3.
Ujedause condition flags if S present. V* Flag is unpredictable in Architecture via and earlier, unchanged in Architecture via and lance. Sickly flag, Always updates on overflow (no S opion), Read and react using MRS and MSR. Four Greater than or Equal flags, Always updates by parallel adds and subtracts. B menting plaff-register [150], or T meaning [211:6]. Refer to Table Processor mode specified by e.g. modes Refer to Table Processor mode	<ps< td=""><td>R></td><td>Either CPSR (Current Processor S</td><td>tatus Register) or SPSR (Saved Processor Status Registe</td><td><u>=</u></td><td>^</td><td>ا ق</td><td>ode4</td><td></td><td>Refer to Table Addressing Mode 4 (Block load or Stack pop).</td></ps<>	R>	Either CPSR (Current Processor S	tatus Register) or SPSR (Saved Processor Status Registe	<u>=</u>	^	ا ق	ode4		Refer to Table Addressing Mode 4 (Block load or Stack pop).
Flag is unpredicable in Architecture v4 and earlier, unchanged in Architecture v5 and earlier, unchanged v6 parallel adds and submates. Four Greater than or Equal flags, Always upshaded by parallel adds and stabutants. Cregitist-PC-Four Greater v6 parallel instructions	2((s)		Updates condition flags if S prese	# ·		۸	⊒	ode4		Refer to Table Addressing Mode 4 (Block store or Stack push).
Sicky flag, Always updates on overflow (no Soption), Read and reset using that Frequent than or Equal flags. Always updated by purallel adds and submates. Four Greater than or Equal flags Always updated by purallel adds and submates. Cregilist-1	C*,	٧*	Flag is unpredictable in Architectu	tre v4 and earlier, unchanged in Architecture v5 and late	—	^	ו מו	odes		Refer to Table Addressing Mode 5.
Four Greuter than or Equal flags. Always updated by parallel adds and subtracts. B meating his flags per per paraing [31:10] and accumulate long unsigned accumulate long and accumulate and subtract long long and accumulate and subtract long and accumulate long and	ю		Sticky flag. Always updates on ov	erflow (no S option). Read and reset using MRS and MS	Ä	_	reg	list		A comma-separated list of registers, enclosed in braces { and }.
B meaning hilf-register [15th], or I meaning [31:16].	GE		Four Greater than or Equal flags.	Always updated by parallel adds and subtracts.			reg.	list		As <reglist>, must not include the PC.</reglist>
Refer to Table Prefixes for Parallel Instructions	Υ,×))	b meaning nair-register [15:0], or	I meaning [31:16].		- ^	-bar	List		As <reglist>, including the PC.</reglist>
Refer to Table Prefixes for Parallel instructions Refer to Table Processor mode specified by ep_modes R13 for the processor mode specified by ep_modes R13 for the processor mode specified by ep_modes with carry saturating double saturating Subfacer Subfacer Subfacer Subfacer	× (×	Tot >	RsX is Rs rotated 16 bits if X pres	-notating at o-oft value by all even number of oils. ent. Otherwise, RsX is Rs.		+	7 ::		1 -	opulies base register after data transfer if i present + or (+ may be omitted.)
Refer to Table Processor Modes Ril for the processor mode specified by cp_mode> Castlags> Ril for the processor mode specified by cp_mode> Ril for the processor mode Ril for the processor mode specified by cp_mode> Ril for the processor mode Ril for the processo	Ą.	efix>	Refer to Table Prefixes for Par	allel instructions		000				Refer to Table ARM architecture versions.
R13 for the processor mode specified by ep_mode> R3 for the processor mode specified by ep_mode> R4 for the processor mode specified by ep_m	,ô	mode>	Refer to Table Processor Mode	is .		۸	ifl;	sge		Interrupt flags. One or more of a, i, f (abort, interrupt, fast interrupt).
Add with carry Supdates ADD[cond] {\$ Rd, Rn, <0perand2> N Z C V	R13	3	R13 for the processor mode speci	ied by <p_mode></p_mode>			\ Z		-	Rounds resul
Add ADD (cond) {\$} Rd, Rn, <operand2> N Z C V </operand2>	용	ration		Assembler		date	ún		_	
ADC(cond){S} Rd, Rm, COPETAND2> N Z C V	Arit	hmetic Add		Rd, Rn,	- 1	7		_	Rd ≔ Rn	슣
SE QADD(cond) Rd, Rm, Rm SE QADD(cond) Rd, Rm, Rn SUB(cond) SRd, Rm, COperand2> N Z C V SBC(cond) SRd, Rm, COperand2> N Z C V SBC(cond) SRd, Rm, COperand2> N Z C V RSB(cond) SRd, Rm, Rn SE QSUB(cond) Rd, Rm, Rn SE QDSUB(cond) Rd, Rm, Rs, Rn M UMLA(cond) SRd, Rm, Rs, Rn M UMLA(cond) SRd, Rm, Rs, Rn M UMLAL(cond) SRdLo, RdHi, Rm, Rs N Z C* V* M UMLAL(cond) SRdLo, RdHi, Rm, Rs N Z C* V* M SMULL(cond) SRdLo, RdHi, Rm, Rs N Z C* V* M SMULL(cond) SRdLo, RdHi, Rm, Rs N Z C* V* M SMULL(cond) SRdLo, RdHi, Rm, Rs N Z C* V* SE SMULWy(cond) Rd, Rm, Rs, Rn SE SMULD(X)(cond) Rd, Rm			чіћ сапу	Rd, Rn,		Z		_	Rd := Rn	÷ Ор
DEADD (cond) Rd, Rm, Rn			•					_		(T(Rm
SBC(cond) {S} Rd, Rn, <operand2> N Z C V RSB(cond) {S} Rd, Rn, <operand2> N Z C V RSC(cond) {S} Rd, Rn, <operand2> N Z C V SE QSUB(cond) Rd, Rm, Rn QDSUB(cond) Rd, Rm, Rn QDSUB(cond) {S} Rd, Rm, Rs M UMUL(cond) {S} RdLo, RdHi, Rm, Rs N Z C* M UMAL(cond) {S} RdLo, RdHi, Rm, Rs N Z C* M UMAL(cond) {S} RdLo, RdHi, Rm, Rs N Z C* M SMULL(cond) {S} RdLo, RdHi, Rm, Rs N Z C* M SMULL(cond) {S} RdLo, RdHi, Rm, Rs N Z C* M SMULL(cond) {S} RdLo, RdHi, Rm, Rs N Z C* M SMULL(cond) {S} RdLo, RdHi, Rm, Rs N Z C* M SMULL(cond) {S} RdLo, RdHi, Rm, Rs N Z C* M SMULL(cond) {S} RdLo, RdHi, Rm, Rs N Z C* M SMULL(cond) {S} RdLo, RdHi, Rm, Rs SE SMULMY(cond) Rd, Rm, Rs, Rn SE SMULMY(cond) Rd, Rm, Rs, Rn SE SMLAD(X) {cond} RdLo, RdHi, Rm, Rs SE SMLAD(X) {cond} RdHi, RdLo, Rm, Rs SMUSD(X) {cond} Rd, Rm, Rs, Rn S SMLSD(X) {cond} RdHi, RdLo, Rm, Rs SMLSD(X) {cond} RdHi, RdLo, Rm, Rs SMLSD(X) {cond} RdH, Rm, Rs, Rn SMLSD(X) {cond} RdH, Rm, Rs, Rn SMLAN(Cond) Rd, Rm, Rs, Rn SMLAN(SR) {cond} Rd, Rm, Rs, Rn SMLAN(SCOND) Ac, Rm, Rs SMLAN(SCOND) Rd, Rm, Rs SMLAN(</operand2></operand2></operand2>		Subr	ie saturatniš							
RSB{cond}{S} Rd, Rn, <operand2></operand2>		1	vith carry	SBC(cond)(S) Rd, Rn, <operand2></operand2>				< <	Rd ii Rn	မှ မွ
SEC (cond) {S} Rd, Rn, < Operand2>			everse subtract						Rd := Op	erand
SE USUB CONIA RG Rm Rn			btract with carry	RSC{cond}{S} Rd, Rn,						erand
MUMICAMA (S) Rd, Rm, Rs MUMILL(cond){S} Rd, Rm, Rs, Rn MUMILL(cond){S} RdLo, RdHi, Rm, Rs MUMILL(cond){Rd, Rm, Rs, Rn SE SMULWy{cond} Rd, Rm, Rs, Rn SMUSD{X}{cond} Rd, Rm, Rs, Rn SMULSD{X}{cond} Rd, Rm, Rs, Rn SMULWY{cond} Rd, Rm, Rs, Rn SMULW{R}{cond} Rd, Rm, Rs, Rn SMULW{Rm, Rs, Rn SMULW{Rm, Rs, Rn SMULW{R}{cond} Rd, Rm, Rs, Rn SMULW{Rm, Rs, Rn Rm, Rs, Rn Rm, Rs, Rn SMULW{Rm, Rs, Rn Rm, Rs, Rn R										
2 MLA{cond}{S} Rd, Rm, Rs, Rn M UMULL{cond}{S} RdLo, RdHi, Rm, Rs M UMULL{cond}{S} RdLo, RdHi, Rm, Rs M UMLAL{cond}{S} RdLo, RdHi, Rm, Rs M SMLAL{cond}{S} RdLo, RdHi, Rm, Rs N Z C* V* M SMLAL{cond}{R} Rd, Rm, Rs SE SMULWy{cond} Rd, Rm, Rs, Rn SE SMLAWy{cond} Rd, Rm, Rs, Rn SE SMLAWy{cond} Rd, Rm, Rs, Rn SE SMLAWy{cond} Rd, Rm, Rs, Rn SE SMLAD{X}{cond} Rd, Rm, Rs, Rn SE SMLAD{X}{cond} Rd, Rm, Rs, Rn SE SMLAD{X}{cond} Rd, Rm, Rs, Rn S SMLSD{X}{cond} Rd, Rm, Rs, Rn S SMLACAT{R}{cond} Rd, Rm, Rs, Rn S SMLACATATATATATATATATATATATATATATATATATATA		Muli	ic sautaurig				Ç.			# * R
M UMULL{cond}{S} RdLo, RdHi, Rm, Rs N Z C* V* M UMLAL{cond}{S} RdLo, RdHi, Rm, Rs 6 UMAAL{cond} RdLo, RdHi, Rm, Rs M SMULAL{cond}{S} RdLo, RdHi, Rm, Rs M SMULAL{cond}{S} RdLo, RdHi, Rm, Rs N Z C* V* M SMULAL{cond}{S} RdLo, RdHi, Rm, Rs N Z C* V* SE SMULWy{cond} Rd, Rm, Rs SE SMULWy{cond} Rd, Rm, Rs, Rn SE SMLAWy{cond} Rd, Rm, Rs, Rn SE SMLAWy{cond} RdLo, RdHi, Rm, Rs SE SMLAWy{cond} Rd, Rm, Rs, Rn SE SMLAD{X}{cond} Rd, Rm, Rs, Rn SE SMLAD{X}{cond} Rd, Rm, Rs, Rn SE SMLAD{X}{cond} Rd, Rm, Rs S SMLAD{X}{cond} Rd, Rm, Rs S SMLAD{X}{cond} Rd, Rm, Rs S SMLSD{X}{cond} Rd, Rm, Rs S SMLSD{X}{cond} Rd, Rm, Rs, Rn S SMMLSD{X}{cond} Rd, Rm, Rs, Rn S SMMLSD{X}{cond} Rd, Rm, Rs, Rn S SMMLX{R}{cond} Rd, Rm, Rs, Rn S SMMLX{R}{R}{cond} Rd, Rm, Rs, Rn			nd accumulate	MLA{cond}{S} Rd, Rm, Rs,			()		Rd := ((R	m * R
M UMLAL (cond) {S} RdLo, RdHi, Rm, Rs N Z C* V* 6 UMAAL (cond) RdLo, RdHi, Rm, Rs M SMULL (cond) {S} RdLo, RdHi, Rm, Rs N SMLAL (cond) {S} RdLo, RdHi, Rm, Rs N Z C* V* SE SMULWy {cond} Rd, Rm, Rs SE SMULWy {cond} Rd, Rm, Rs, Rn SE SMLAWy {cond} Rd, Rm, Rs, Rn SE SMLAWy {cond} RdLo, RdHi, Rm, Rs SE SMLAWy {cond} Rd, Rm, Rs, Rn SE SMLAD{X} {cond} Rd, Rm, Rs, Rn SE SMLAD{X} {cond} Rd, Rm, Rs SMLSD{X} {cond} RdHi, RdLo, Rm, Rs SMLSD{X} {cond} Rd, Rm, Rs, Rn SMMLX{R} {cond} Rd, Rm, Rs, Rn		_	nsigned long	UMULL{cond}{S} RdLo, RdHi, Rm,				*	RdHi,Rd	اء ∹ ما
M SMULL(cond) KOLLO, ROHLI, RM, RS M SMULL(cond) {S} RAILO, RHHI, RM, RS M SMULL(cond) {S} RAILO, RHHI, RM, RS N Z C* V* SE SMULWY(cond) Rd, RM, RS SE SMULWY(cond) Rd, RM, RS, RN SE SMLAWY(cond) Rd, RM, RS, RN SE SMLAWY(cond) Rd, RM, RS, RN SE SMLALXY(cond) Rd, RM, RS, RN SE SMLAD(X) {cond} Rd, RM, RS, RN SE SMLAD(X) {cond} Rd, RM, RS, RN SMLAD(X) {cond} Rd, RM, RS, RN SMLAD(X) {cond} Rd, RM, RS, RN SMLSD(X) {cond} Rd, RM, RS, RN SMLSD(X) {cond} Rd, RM, RS, RN SMMLS(R) {cond} Rd, RM, RS, RN SMMLIR {cond} Rd, RM, RS, RN SMMLA(R) {cond} Rd, RM, RS, RN SMMLA(R) {cond} Rd, RM, RS, RN SMMLA(R) {cond} Rd, RM, RS, RN SMMLA(Cond) AC, RM, RS XS MIAPH(cond) AC, RM, RS SCLZ(cond) Rd, RM, RS SCLZ(cond) Rd, RM, RS			nsigned accumulate long	UMLAL(cond)(S) RdLo, RdHi, Rm,				*	RdHi,Rd	n≕oj
M SMULL/COND) (S) RGLIO, RGHI, RM, RS N Z C* V* M SMULLY(COND) (R), RM, RM, RS SE SMULWY(COND) Rd, RM, RS, RN SE SMULWY(COND) Rd, RM, RS, RN SE SMLAWY(COND) Rd, RM, RS, RN SE SMLAWY(COND) Rd, RM, RS, RN SE SMLAWY(COND) Rd, RM, RS, RN SE SMLAD(X)(COND) Rd, RM, RS, RN SE SMLAD(X)(COND) Rd, RM, RS, RN SMLAD(X)(COND) Rd, RM, RS, RN SMLAD(X)(COND) Rd, RM, RS, RN SMLSD(X)(COND) Rd, RM, RS, RN SMLAD(X)(COND) Rd, RM, RS, RN SMLAPH(COND) AC, RM, RS XS MIARY(COND) Rd, RM, RS SCLZ(COND) Rd, RM, RS SCLZ(COND) Rd, RM, RS		;	nsigned double accumulate long	UMAAL (cond) RdLo, RdHi, Rm, Rs				<u> </u>	RdHi,Rd	m ≕ o∩
SE SMULAY(cond) Rd, Rm, Rs SE SMULAY(cond) Rd, Rm, Rs SE SMULAY(cond) Rd, Rm, Rs, Rn SE SMLAXY(cond) Rd, Rm, Rs, Rn SE SMLAXY(cond) Rd, Rm, Rs, Rn SE SMLALAY(cond) Rd, Rm, Rs, Rn SE SMLALAY(cond) Rd, Rm, Rs, Rn SMLALD(X)(cond) Rd, Rm, Rs, Rn SMLAD(X)(cond) Rd, Rm, Rs, Rn SMLSD(X)(cond) Rd, Rm, Rs, Rn SMLSD(X)(cond) Rd, Rm, Rs, Rn SMLSD(X)(cond) Rd, Rm, Rs, Rn SMMLIR(Cond) Rd, Rm, Rs, Rn SMMLIR(Cond) Rd, Rm, Rs, Rn SMMLAR(Cond) Rd, Rm, Rs, Rn SMMLA(Cond) Rd, Rm, Rs, Rn		Sign		SMULL(cond)(S) RdLo, RdHi, Rm,				*	RdHi,Rd	Lo ≔ sig
SE SMULWY(cond) Rd, Rm, Rs SE SMULWy(cond) Rd, Rm, Rs, Rn SE SMLAWy(cond) Rd, Rm, Rs, Rn SE SMLAWy(cond) Rd, Rm, Rs, Rn SE SMLAWy(cond) Rd, Rm, Rs, Rn 6 SMLAD(X)(cond) Rd, Rm, Rs, Rn 6 SMLAD(X)(cond) Rd, Rm, Rs, Rn 6 SMLAD(X)(cond) Rd, Rm, Rs 6 SMUSD(X)(cond) Rd, Rm, Rs, Rn 6 SMLSD(X)(cond) Rd, Rm, Rs, Rn 6 SMLSD(X)(cond) Rd, Rm, Rs, Rn 6 SMMUL(R)(cond) Rd, Rm, Rs, Rn 7 SMMLA(R)(cond) Rd, Rm, Rs, Rn 8 SMMLA(R)(cond) Rd, Rm, Rs, Rn 9 CLZ(cond) Ac, Rm, Rs 7 SMIAY(cond) Ac, Rm, Rs 7 SMIAY(cond) Rd, Rm, Rs		. 50		SMLAL(cond){S} RdLo, RdHi, Rm,				*	RdHi,Rd	Lo := si
SE SMLAWY(cond) Rd, Rm, Rs, Rn SE SMLAWY(cond) Rd, Rm, Rs, Rn SE SMLAWY(cond) RdLo, RdHi, Rm, Rs 6 SMLAD(X) (cond) RdLo, RdHi, Rm, Rs 6 SMLAD(X) (cond) Rd, Rm, Rs, Rn 6 SMLAD(X) (cond) Rd, Rm, Rs, Rn 6 SMLAD(X) (cond) Rd, Rm, Rs 6 SMLSD(X) (cond) Rd, Rm, Rs, Rn 6 SMLSD(X) (cond) Rd, Rm, Rs, Rn 6 SMMLS(R) (cond) Rd, Rm, Rs, Rn 7 SMLA(cond) Ac, Rm, Rs, Rn 8 SMMLS(R) (cond) Rd, Rm, Rs, Rn 9 CLZ(cond) Ac, Rm, Rs 7 SMIAY(cond) Ac, Rm, Rs				SMULxy{cond} Rd, Rm,					Rd := Rn] * [x] n
SMLAMY(cond) Rd. Rm. Rs. Rn. Rs. Rn. Q			and accumulate	SMIAXY{cond} pd Pm Ps				_		L P K
5E SMLALXY (cond) RdLo, RdHi, Rm, Rs 6 SMUAD (X) (cond) Rd, Rm, Rs, Rn 6 SMLALD (X) (cond) Rd, Rm, Rs, Rn 6 SMLALD (X) (cond) RdHi, RdLo, Rm, Rs 6 SMUAD (X) (cond) Rd, Rm, Rs, Rn 6 SMLSD (X) (cond) Rd, Rm, Rs, Rn 6 SMMLSLD (X) (cond) Rd, Rm, Rs, Rn 6 SMMLS(R) (cond) Rd, Rm, Rs 6 SMMLARY (cond) Rd, Rm, Rs, Rn 6 SMMLARY (cond) Rd, Rm, Rs, Rn 7 SMLACOND Rd, Rm, Rs, Rn 8 SMMLARY (cond) Rd, Rm, Rs, Rn 9 CLZ (cond) Rd, Rm, Rs				SMLAWy{cond} Rd, Rm, Rs,						+ 7
6 SMUAD{X}{cond} Rd, Rm, Rs 6 SMLAD{X}{cond} Rd, Rm, Rs, Rn 6 SMLAD{X}{cond} RdHi, RdLo, Rm, Rs 6 SMLSD{X}{cond} Rd, Rm, Rs 6 SMLSD{X}{cond} Rd, Rm, Rs 6 SMLSD{X}{cond} Rd, Rm, Rs, Rn 6 SMLSLD{X}{cond} Rd, Rm, Rs, Rn 7 Cond} Rd, Rm, Rs, Rn 8 SMMUL{R}{cond} Rd, Rm, Rs, Rn 8 SMMLX{R}{cond} Rd, Rm, Rs, Rn 9 CONDREASE RATE CONDREASE RATE RATE RATE RATE RATE RATE RATE RAT			long	SMLALXY {cond} RdLo, RdHi, Rm,						ا ا
6 SMLAD{X}{cond} Rd, Rm, Rs, Rn Q 6 SMLALD{X}{cond} RdHi, RdLo, Rm, Rs 6 SMLSD{X}{cond} Rd, Rm, Rs 6 SMLSD{X}{cond} Rd, Rm, Rs, Rn Q 6 SMLSD{X}{cond} Rd, Rm, Rs, Rn Q 6 SMMUSD{X}{cond} Rd, Rm, Rs, Rn Q 6 SMMUS{R}{cond} Rd, Rm, Rs 6 SMMUS{R}{cond} Rd, Rm, Rs 6 SMMLS{R}{cond} Rd, Rm, Rs, Rn RS NIA{cond} Rd, Rm, Rs, Rn RS XS MIAFURCOND Ac, Rm, Rs XS MIAFURCOND Ac, Rm, Rs SS MIAFURCOND RD, RM, Rs		Dual	signed multiply, add					_		:51]r
6 SMLALD{X}{cond} RdHi, RdLo, Rm, Rs 6 SMLSD{X}{cond} Rd, Rm, Rs Rs, Rn 7 SMLSD{X}{cond} Rd, Rm, Rs, Rn 8 SMMLX{R}{cond} Rd, Rm, Rs, Rn 9 SMMLX{R}{cond} Rd, Rm, Rs 9 SMMLX{R}{cond} Rd, Rm, Rs 9 SMMLX{R}{cond} Rd, Rm, Rs, Rn 10 SMMLX{R}{cond} Rd, Rm, Rs, Rn 11 SMMLX{R}{cond} Rd, Rm, Rs, Rn 12 SMMLX{R}{cond} Rd, Rm, Rs, Rn 13 SMMLX{Cond} Ac, Rm, Rs 14 SMMLX{Cond} Rd, Rm, Rs 15 CLZ{cond} Rd, Rm, Rs			nd accumulate					_		+ 2
6 SMMSD{X}{cond} Rd, Rm, Rs 6 SMLSD{X}{cond} Rd, Rm, Rs, Rn Q 6 SMLSLD{X}{cond} Rd, Rm, Rs, Rn Q 6 SMMLSLD{X}{cond} Rd, Rm, Rs 6 SMMLX{R}{cond} Rd, Rm, Rs 6 SMMLX{R}{cond} Rd, Rm, Rs, Rn 6 SMMLX{R}{cond} Rd, Rm, Rs, Rn 8 SMMLX{R}{cond} Rd, Rm, Rs, Rn 9 SMMLX{R}{cond} Rd, Rm, Rs, Rn 10 XS MIA{cond} Ac, Rm, Rs XS MIAPH{cond} Ac, Rm, Rs S SMIAPH{cond} Rd, Rm, Rs S SMIAPH{cond} Rd, Rm, Rs			nd accumulate long	SMLALD{x}(cond) RdHi, RdLo, Rm,						
6 SMLSD{X}{cond} Rd, Rm, Rs, Rn 6 SMLSLD{X}{cond} RdHi, RdLo, Rm, Rs 6 SMMLT{R}{cond} Rd, Rm, Rs 6 SMMLT{R}{cond} Rd, Rm, Rs, Rn 6 SMMLS{R}{cond} Rd, Rm, Rs, Rn 8 SMMLS{R}{cond} Rd, Rm, Rs, Rn 10 XS MIA{cond} Ac, Rm, Rs XS MIAPH{cond} Ac, Rm, Rs XS MIAPH{cond} Ac, Rm, Rs 5 CLZ{cond} Rd, Rm		Dual	signed multiply, subtract	SMUSD(x) (cond) Rd, Rm,				_		1[15:
6 SMM_SLD[X]{cond} RdHi, RdIo, Rm, Rs 6 SMMUI_{R}{cond} Rd, Rm, Rs 6 SMMUI_{R}{cond} Rd, Rm, Rs, Rn 6 SMMIS{R}{cond} Rd, Rm, Rs, Rn 8 XS MIA{cond} Ac, Rm, Rs XS MIAPH{cond} Ac, Rm, Rs XS MIAPH{cond} Ac, Rm, Rs S MIAXY{cond} Rd, Rm, Rs			nd accumulate	SMLSD{X}{cond} Rd, Rm, Rs,						+ R
6 SMMIL{R}{cond} Rd, Rm, Rs 6 SMMILA{R}{cond} Rd, Rm, Rs, Rn 6 SMMLS{R}{cond} Rd, Rm, Rs, Rn 8 XS MIA{cond} Ac, Rm, Rs XS MIAPH{cond} Ac, Rm, Rs XS MIAPH{cond} Ac, Rm, Rs 5 CLZ{cond} Rd, Rm		!	nd accumulate long	SMLSLD{X}(cond) RdHi, RdLo, Rm,						
6 SMMLS{R}{cond} Rd, Rm, Rs, Rn 6 SMMLS{R}{cond} Ac, Rm, Rs, Rn XS MIAPH(cond) Ac, Rm, Rs XS MIAPH(cond) Ac, Rm, Rs XS MIAXY(cond) Ac, Rm, Rs CLZ{cond} Rd, Rm		Sign	ed most significant word multiply	SMMUL(R)(cond) Rd, Rm, Rs					Rd := (Rı	. = *
XS MIA (cond) Ac, Rm, Rs XS MIAPH(cond) Ac, Rm, Rs XS MIAPH(cond) Ac, Rm, Rs XS MIAXY(cond) Ac, Rm, Rs 5 CLZ (cond) Rd, Rm		3 B	זה שככתוותומוני	SMMLA(K) (CONG) KG, KM, KS,					Ka ii Ka	1 2
XS MIAPH(cond) Ac. Rm., Rs XS MIAXY(cond) Ac., Rm., Rs 5 CLZ(cond) Rd., Rm		Mul.	iply with internal 40-bit accumulate	MIA(cond) Ac. Rm. Rs					Ac ii Ac	+ Rm
XS MIAxy{cond} Ac, Rm, Rs 5 CLZ{cond} Rd, Rm	·		acked halfword	MIAPH (cond) Ac, Rm,					Ac := Ac	÷ Em
5 CLZ(cond) Rd, Rm				MIAxy(cond) Ac, Rm,					Ac := Ac	+ Rm
	Γ	Cou	it leading zeroes	_					Rd := nuı	mber

		the same (and the same to the
	<a_mode2></a_mode2>	Refer to Table Addressing Mode 2.
erand2.	<a_mode2p></a_mode2p>	Refer to Table Addressing Mode 2 (Post-indexed only).
	<a_mode3></a_mode3>	Refer to Table Addressing Mode 3.
gister)	<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).
later.	<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 ! present.
	+/-	+ or -, (+ may be omitted.)
	\$	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.

		1					
Parallel	Balfund wise addition	, 4	Assembler	Supulates & GE Action	5	16	Z1. n_r31.121
arithmetic	arithmetic Halfword-wise subtraction	6	<pre><pre><pre><pre><pre><pre><pre><pre></pre></pre></pre></pre></pre></pre></pre></pre>			G (GE Rd[3]:16] := Rn[3]:16] = Rm[3]:16], Rd[15:0] := Rn[15:0] = Rm[15:0]
	Byte-wise addition	6	<pre><prefix>ADD8(cond) Rd, Rn, Rm</prefix></pre>			ရှု	~ +
	Byte-wise subtraction	6	<pre><prefix>SUB8{cond} Rd, Rn, Rm</prefix></pre>			읪	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	6	<pre><prefix>ADDSUBX{cond} Rd, Rn, Rm</prefix></pre>		_	GH H	Rd[31:16] := Rn[31:16] + Rm[15:0], Rd[15:0] := Rn[15:0] - Rm[31:16]
	Halfword-wise exchange, subtract, add	6	<pre><prefix>SUBADDX{cond} Rd, Rn, Rm</prefix></pre>		_	Œ	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				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				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	Move		MOV(cond)(s) Rd, <operand2></operand2>	Z			Rd := Operand2
	NOT	-	MVN{cond}{S} Rd, <operand2></operand2>	NZC			Rd := 0xFFFFFFF EOR Operand2
	PSR to register	w	MRS {cond} Rd, <psr></psr>			_	Rd := PSR
	register to PSR	Ç	MSR{cond} <psr>_<fields>, Rm</fields></psr>			_	PSR := Rm (selected bytes only)
	immediate to PSR	ü	٠.				PSR := immed_8r (selected bytes only)
	40-bit accumulator to register	5	-				KdLo := Ac[31:0], KdHi := Ac[39:32]
	Conv	ر د	6 CBV(cond) Bd (Cherand)				Ac[51:0] := KaLo, Ac[59:52] := RaHi Rd := Operation
Logical	Test	T	R	NZC	#		Update CPSR flags on Rn AND Operand2
	Test equivalence		_	NZC		_	
	AND		AND {cond}{S} Rd, Rn, <operand2></operand2>	N		_	Rd := Rn AND Operand2
	EOR		EGR{cond}{S} Rd, Rn, <operand2></operand2>			_	Rd := Rn EOR Operand2
	ORR		ORR(cond){S} Rd, Rn, <operand2></operand2>	N		_	Rd := Rn OR Operand2
	Bit Clear		BIC(cond){S} Rd, Rn, <operand2></operand2>				Rd := Rn AND NOT Operand2
Compare	Compare		CMP(cond) Rn, <operand2></operand2>	NZCV		_	Update CPSR flags on Rn – Operand2
	negative	Ī	CMN(cond) Rn, <operand2></operand2>	NZCV		_	Update CPSR flags on Rn + Operand2
Saturate	Signed saturate word, right shift	9	SSAT{cond} Rd, # <sat>, Rm{, ASR <sh>}</sh></sat>		Q	_	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	_	Rd := SignedSat((Rm LSL sh), sat). <sat> range 0-31, <sh> range 0-31.</sh></sat>
	Signed saturate two halfwords	6	SSAT16(cond) Rd, # <sat>, Rm</sat>		٥		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	_	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>		Q	_	Rd := UnsignedSat((Rm LSL sh), sat). <sat> range 0-31, <sh> range 0-31.</sh></sat>
	Unsigned saturate two halfwords	6	USAT16{cond} Rd, # <sat>, Rm</sat>		۵		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

Operation Pack	Pack halfword bottom + top	0 %	Rd, Rn,	Rd[15:0] := Rn[15:0], Rd[31:16] := (Rm LSL sh)
	Pack halfword top + bottom	6	Rd, Rn, Rm{,	Rd[31:16] := Rn[31:16], Rd[15:0] := (Rm ASR sh)[15:0] sh 1-32.
Signed	Halfword to word	у D	SXTH(cond) Rd, Rm(, ROR # <sh>)</sh>	Rd[31:0] := SignExtend((Rm ROR (8 * sh))[15:0
	Rute to word	7	STRICONAL DA DE LOCALINA	Rd[15:0] := SignExtend((Rm ROR (8 * sh))[7:0]) $Rd[15:0] := SignExtend((Rm ROR (8 * sh))[7:0])$
Unsigned	Halfword to word	4		Rd[31:0] := ZeroExtend((Rm ROR (8 * sh))[15:0]). sh 0-3.
extend	Two bytes to halfwords	6	UXTB16(cond) Rd, Rm{, ROR # <sh>}</sh>	Rd[3]:16] := ZeroExtend((Rm ROR (8 * sh))[23:Rd[15:0] := ZeroExtend((Rm ROR (8 * sh))[7:0])
	Byte to word	6	UXTB{cond} Rd, Rm{, ROR # <sh>}</sh>	Rd[31:0] := ZeroExtend((Rm ROR (8 * sh))[7:0])
Signed	Halfword to word, add	9		Rd[31:0] := Rn[31:0] + SignExtend((Rm ROR (8)))
extend with add	Two bytes to halfwords, add	6	SXTAB16{cond} Rd, Rn, Rm{, ROR # <sh>}</sh>	Rd[31:16] := Rn[31:16] + SignExtend((Rm ROR Rd[15:0] := Rn[15:0] + SignExtend((Rm ROR (8
	Byte to word, add	6	SXTAB{cond} Rd, Rn, Rm{, ROR # <sh>}</sh>	Rd[31:0] := Rn[31:0] + SignExtend((Rm ROR (8
Unsigned	Halfword to word, add	6	걸	Rd[31:0] := Rn[31:0] + ZeroExtend((Rm ROR (8)))
extend with add	Two bytes to halfwords, add	6	UXTAB16 {cond} Rd, Rn, Rm{, ROR # <sh>}</sh>	Rd[31:16] := Rn[31:16] + ZeroExtend((Rm ROR Rd[15:0]) := Rn[15:0] + ZeroExtend((Rm ROR (8 Rd[15:0])))
	Byte to word, add	6	UXTAB{cond} Rd, Rn, Rm{, ROR # <sh>}</sh>	Rd[31:0] := Rn[31:0] + ZeroExtend((Rm ROR (8)))
Reverse bytes	In word	0,	REV(cond) Rd, Rm	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 both halfwords	6	REV16{cond} Rd, Rm	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 low halfword, sign extend	6	REVSH{cond} Rd, Rm	Rd[15:8] := Rm[7:0], Rd[7:0] := Rm[15:8], Rd[31:16] := Rm[7] * &FFFF
Select	Select bytes	6	SEL(cond) Rd, Rn, Rm	Rd[7:0] := Rn[7:0] if GE[0] = 1, else Rd[7:0] := 1 Bits[15:8], [23:16], [31:24] selected similarly by
Branch	Branch		B{cond} label	R15 := label
	with link		BL{cond} label	R14 := address of next instruction, R15 := label
	and exchange with link and exchange (1)	4T,5 ST	4T,5 BX{cond} Rm 5T BLX label	R15 := Rm, Change to Thumb if Rm[0] is 1 R14 := address of next instruction, R15 := label,
	with link and exchange (2)	Ų	BLX{cond} Rm	R14 := address of next instruction, R15 := Rm[3] Change to Thumb if Rm[0] is 1
	and change to Java state	51,6		Change to Java state
Processor	Change processor state	. 6	- -	Disable specified interrups, optional change mod
change	Change processor mode	0 0	CPS # {, # <p_mode>}</p_mode>	Enable specified interrups, optional change mode
	Set endianness	6	SETEND <endianness></endianness>	Sets endianness for loads and saves, <endianness> can be BE (Big Endian) or LE</endianness>
	Store return state	. 6	SRS <a_mode48> #<p_mode>{!}</p_mode></a_mode48>	[R13m] := R14, [R13m + 4] := CPSR
	Breakpoint	r, o	EKPT <inmed 16=""></inmed>	PC := [Kn], CPSK := [Kn + 4] Prefetch abort <i>or</i> enter debug state.
Software interrupt	Software interrupt		SWI{cond} <immed_24></immed_24>	Software interrupt processor exception.
No Op	No operation	2	NOP	None

Operation		Ś	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>		
•				(§ 5T: Change to Thumb if [address][0] is 1)	
	Byte		LDR(cond)B Rd, <a_mode2></a_mode2>	Rd := ZeroExtend[byte 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	LDR(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 := SignExtend[halfword from address]	Rd must not be R15.
	Doubleword	5E*	LDR{cond}D Rd, <a mode3="">	Rd := [address], R(d+1) := [address + 4]	Rd must be even, and not R14.
Load multiple	Pop, or Block data load		LDM(cond) <a_mode4l> Rn(!), <reglist-pc></reglist-pc></a_mode4l>		-
	return (and exchange)		LDM{cond} <a_mode4l> Rn{!}, <reglist+pc></reglist+pc></a_mode4l>	Load registers, R15 := [address][31:1] (§ 5T: Change to Thumb if [address][0] is 1)	
	and restore CPSR		LDM(cond) <a_mode4l> Rn(!), <reglist+pc>^</reglist+pc></a_mode4l>	CPSR := SPSR	Use from exception modes only.
	User mode registers		LDM(cond) <a_mode4l> Rn, <reglist-pc>^</reglist-pc></a_mode4l>		Use from privileged modes only.
Soft preload	Memory system hint	ΣĘ	5E* PLD <a_mode2></a_mode2>		Cannot be conditional.
Load exclusive	Load exclusive Semaphore operation	2	LDREX{cond} Rd, [Rn]	Rd := [Rn], tag address as exclusive access Outstanding tag set if not shared address	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_mode2p></a_mode2p>	[address] := Rd	
	Вуте		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	Ę	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		<reglist>^</reglist>	ters 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.
Swan	Ward		משול ביים ואם ייים והיו	KG := U if successful, else l	
	Ryfe	۰ ،	confords and am (and	tone :- Zone Enter Mille 117.01)	
	x-3 ***	٠,	one (come) is and, and, family	[Rn][7:0] := Rm[7:0], Rd := temp	

Page 9 of 11

[Rn, #+/- <immed_12>]{!}</immed_12>		
[Rn]	Equivalent to [Rn,#0]	Ta
[Rn, +/-Rm]{!}		3
	Allowed shifts 0-31	nE.
[Rn, +/-Rm, LSR # <shift>]{!}</shift>	Allowed shifts 1-32	nE.
[Rn, +/-Rm, ASR # <shift>]{!}</shift>	Allowed shifts 1-32	×
[Rn, +/-Rm, ROR # <shift>]{!}</shift>	Allowed shifts 1-31	1
[Rn, +/-Rm, RRX] {!}		Flexi
[Rn], #+/- <immed_12></immed_12>		Imme
[Rn], +/-Rm		Logic
	Allowed shifts 0-31	Logic
[Rn], +/-Rm, LSR # <shift></shift>	Allowed shifts 1-32	Arith
[Rn], +/-Rm, ASR # <shift></shift>	Allowed shifts 1-32	Rotat
[Rn], +/-Rm, ROR # <shift></shift>	Allowed shifts 1-31	Regis
[Rn], +/-Rm, RRX		Rotat
	Immediate offset	

Addressing	Addressing Mode 2 (Post-Indexed only	ed only)				
Post-indexed	Post-indexed Immediate offset	(ma]	[Rn], #+/- <immed_12;< th=""><th>med_1</th><th>12></th><th></th></immed_12;<>	med_1	12>	
	Zero offset	[Rn]				Equivalent to [Rn],#0
	Register offset	[Rn],	[Rn], +/-Rm			
	Scaled register offset [Rn], +/-Rm, LSL # <shift></shift>	[Rn],	+/-Rm,	tsr ‡	# <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
		[Rm],	[Rn], +/-Rm, RRX	RRX		

Addressing	Mode 3 - Halfword,	Addressing Mode 3 - Halfword, Signed Byte, and Doubleword Data Transfer	ransfer
Pre-indexed	Immediate offset	[Rn, #+/- <immed_8>]{!}</immed_8>	
	Zero offset		Equivalent to [Rn,#0]
	Register	[Rn, +/-Rm] {!}	
Post-indexed	Immediate offset	[Rn], #+/- <immed_8></immed_8>	
	Register	[Rn], +/-Rm	

Addressing I	Addressing Mode 4 - Multiple Data Transfer	ta Transfer	
Block load		Stack pop	
IA	Increment After	FD	Full Descending
IB	Increment Before	Ħ	Empty Descending
DA	Decrement After	FA	Full Ascending
DB	Decrement Before	EA	Empty Ascending
Block store	8	Stack push	
TA	Increment After	EA	Empty Ascending
ĭΒ	Increment Before	FA	Full Ascending
DA	Decrement After	Ħ	Empty Descending
DB	Decrement Before	FD	Full Descending

Addressing	Addressing Mode 5 - Coprocessor Data Transfer	or Data Transfer	
Pre-indexed	Pre-indexed Immediate offset	[Rn, #+/- <immed_8*4>]{!}</immed_8*4>	
	Zero offset		Equivalent to [Rn.#0]
Post-indexed	Immediate offset	[Rn], #+/- <immed_8*4></immed_8*4>	
Unindexed	No offset	[Rn], {8-bit copro. option}	

d and	Unsign	d and Unsigned Byte Data Transfer		ARM architecture versions	ure versions			
Set	[Rn,	[Rn, #+/- <immed_12>]{!}</immed_12>		и	ARM architecture version n and above.	on 11 an	d above.	
	[Rn]		Equivalent to [Rn,#0]	nT, nJ	T or J variants of ARM architecture version n and above.	archite	cture version n and abov	ře.
-	[Rn,	[Rn, +/-Rm] {!}		×	ARM architecture versi	on 3M,	ARM architecture version 3M, and 4 and above, except xM variants.	t xM variants.
r offset	[Rn,	r offset $ [Rn, +/-Rm, LSL \#] {:} Allowed shifts 0-31$] { : } Allowed shifts 0-31	πE	All E variants of ARM	archited	All E variants of ARM architecture version n and above	'n
	[Rn,	<pre>[Rn, +/-Rm, LSR #<shift>]{!} Allowed shifts 1-32</shift></pre>] { 1 } Allowed shifts 1-32	»E*	E variants of ARM architecture version n and above, except xP variants.	itecture	e version n and above, en	ccept xP variants.
	[Rn	$[Rn, +/-Rm, ASR #]{!}$ Allowed shifts 1-32] { ! } Allowed shifts 1-32	XX	XScale coprocessor instruction	ruction		
	[Rn,	$[Rn, +/-Rm, ROR \#]{!}Allowed shifts 1-3!$] { 1 } Allowed shifts 1-31					
	[Rn,	[Rn, +/-Rm, RRX] {!}		Flexible Operand 2	nd 2			
set	[Rn]	[Rn], #+/- <immed_12></immed_12>		Immediate value		# <imm< td=""><td>#<immed_8r></immed_8r></td><td></td></imm<>	# <immed_8r></immed_8r>	
	[RF]	[Rn], +/-Rm		Logical shift left immediate		Rm, L	Rm, LSL # <shift></shift>	Allowed shifts 0-2
r offset	[Rn]	roffset [Rn], +/-Rm, LSL # <shift:< td=""><td>> Allowed shifts 0-31</td><td>Logical shift right immediate</td><td></td><td>Rm, L</td><td>Rm, LSR #<shift></shift></td><td>Allowed shifts 1-3</td></shift:<>	> Allowed shifts 0-31	Logical shift right immediate		Rm, L	Rm, LSR # <shift></shift>	Allowed shifts 1-3
	[Rn]	[Rn], +/-Rm, LSR # <shift< td=""><td>> Allowed shifts 1-32</td><td>Arithmetic shift right immediate</td><td></td><td>Rm, A</td><td>ASR #<shift></shift></td><td>Allowed shifts 1-3</td></shift<>	> Allowed shifts 1-32	Arithmetic shift right immediate		Rm, A	ASR # <shift></shift>	Allowed shifts 1-3
	[Rn]	[Rn], +/-Rm, ASR # <shift:< td=""><td>> Allowed shifts 1-32</td><td>Rotate right immediate</td><td></td><td>Rm, R</td><td>Rm, ROR #<shift></shift></td><td>Allowed shifts 1-3</td></shift:<>	> Allowed shifts 1-32	Rotate right immediate		Rm, R	Rm, ROR # <shift></shift>	Allowed shifts 1-3
	[Rn]	[Rn], +/-Rm, ROR # <shift:< td=""><td>> Allowed shifts 1-31</td><td>Register</td><td></td><td>Rm</td><td></td><td></td></shift:<>	> Allowed shifts 1-31	Register		Rm		
	[Rn]	[Rn], +/-Rm, RRX		Rotate right extended		Rm, R	RRX	
				T period chief last maries	•	1	101 101	

	PSR[15:8]	yte	Extension field mask byte	×
	PSR[23:16]		Status field mask byte	ts
	PSR[31:24]		Flags field mask byte	Fħ
	[PSR[7:0]	()	Control field mask byte	a
			Meaning	Suffix
			(use at least one suffix)	PSR fields
	Rm, ROR Rs	Rm,	ster	Rotate right register
	ASR Rs	Rm,	right register	Anthmetic shift right register
	LSR Rs	Rm,	ht register	Logical shift right register
	LSL Rs	Rm,	tregister	Logical shift left register
	RRX	Rm,	ended	Rotate right extended
		Rm		Register
Allowed shifts 1-31	ROR # <shift></shift>	Rm,	nediate	Rotate right immediate
Allowed shifts 1-32	ASR # <shift></shift>	Rm,	right immediate	Arithmetic shift right immediate
Allowed shifts 1-32	LSR # <shift></shift>	Rm,	ht immediate	Logical shift right immediate
Allowed shifts 0-31	LSL # <shift></shift>	Rm,	t immediate	Logical shift left immediate
	# <immed_8r></immed_8r>	# <im< td=""><td></td><td>Immediate value</td></im<>		Immediate value

Condition Field	<u>a</u>	
Mnemonic	Description	Description (VFP)
ΒQ	Equal	Equal
NE	Not equal	Not equal, or unordered
CS / HS	Carry Set / Unsigned higher or same	Greater than or equal, or unordered
CC / TO	Carry Clear / Unsigned lower	Less than
MI	Negative	Less than
Jd.	Positive or zero	Greater than or equal, or unordered
νs	Overflow	Unordered (at least one NaN operand)
٧c	No overflow	Not unordered
HI	Unsigned higher	Greater than, or unordered
ST	Unsigned lower or same	Less than or equal
GE GE	Signed greater than or equal	Greater than or equal
LT	Signed less than	Less than, or unordered
GT.	Signed greater than	Greater than
TE	Signed less than or equal	Less than or equal, or unordered
ΑL	Always (normally omitted)	Always (normally omitted)

Processor Modes	des	Prefi	Prefixes for Parallel Instruction
16	User	ß	Signed arithmetic modulo 28
17	FIQ Fast Interrupt	Ø	Signed saturating arithmetic
18	IRQ Interrupt	HS	Signed arithmetic, halving re
19	Supervisor	□	Unsigned arithmetic modulo
23	Abort	ő	Unsigned saturating arithme
27	Undefined	H	Unsigned arithmetic, halving
31	System		

	Prefi	Prefixes for Parallel Instructions
	S	Signed arithmetic modulo 28 or 216, sets CPSR GE bits
ast Interrupt	Ø	Signed saturating arithmetic
Interrupt	HS	Signed arithmetic, halving results
rvisor	ਖ	Unsigned arithmetic modulo 28 or 216, sets CPSR GE bits
	ő	Unsigned saturating arithmetic
fined	HD	Unsigned arithmetic, halving results

Coprocessor operations	§ Assembler	Action	Notes
Data operations	2 CDP{cond} <copr>, <opl>, CRd, CRn, CRm{, <op2>}</op2></opl></copr>	Coprocessor dependent	
Alternative data operations	5 CDP2 <copr>, <opl>, CRd, CRn, CRm{, <op2>}</op2></opl></copr>	_	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>	_	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>, <opl>, Rd, Rn, CRm</opl></copr>	_	Cannot be conditional.
Move to coproc from ARM reg	2 MCR(cond) <copr>, <op1>, Rd, CRn, CRm{, <op2>}</op2></op1></copr>	Coprocessor dependent	
Alternative move	5 MCR2 <copr>, <op1>, Rd, CRn, CRm{, <op2>}</op2></op1></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>, <opl>, Rd, Rn, CRm</opl></copr>	_	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.

Document Number

ARM QRC 0001H

Change L	Log		
Issue	Date	Ву	Change
Α	June 1995	BJH	First Release
В	Sept 1996	BJH	Second Release
C	Nov 1998	BJH	Third Release
D	Oct 1999	CKS	Fourth Release
m	Oct 2000	CKS	Fifth Release
ъ	Sept 2001	CKS	Sixth Release
G	Jan 2003	CKS	Seventh Release
H	Oct 2003	CKS	Eighth Release