UNIVERSITY OF DUBLIN TRINITY COLLEGE

Faculty of Engineering, Mathematics and Science

School of Computer Science & Statistics

Integrated Computer Science Year 2 Examination

Trinity Term 2013

Microprocessor Systems

Monday April 29, 2013

Sports Centre (455)

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:

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

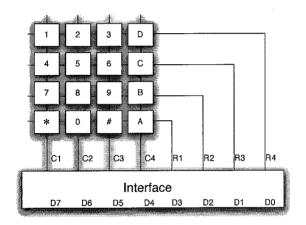
Non-programmable calculators are permitted for this examination — please indicate the make and model of your calculator on each answer book used.

 (a) Explain how a pipelined architecture can speed up the execution of instructions on a processor. Explain the issues that can slow a pipeline down and discuss ways of getting around some of these problems.

[10 marks]

(b) List the components and properties of the *memory hierarchy*, and hence explain why a memory hierarchy seems to be necessary in the first place.
 Describe the function and operation of cache memories. In your answer, discuss the different types of cache organization and replacement polices, and list their advantages and disadvantages. [10 marks]

- (a) Two ways to detect activity on peripheral interfaces are by *polling* and by interrupt handling. Briefly explain both approaches, and explain the difference between them.
 [5 marks]
 - (b) 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.



When a key is pressed, it *bounces*; as it changes state (from open to closed or from closed to open), the springiness and inertia of the moving parts combine to make the electronic contacts open and close rapidly and irregularly for a short period, less than 5 milliseconds, making the output code change rapidly before finally settling down its true value.

Write a polling subroutine that reliably returns, in R0, the ASCII code of a key when it is pressed. Explain clearly how your subroutine works.

[15 marks]

- 3. (a) What are the differences between an IRQ and a FIQ? [2 marks]
 - (b) What is meant by latency in the context of an interrupt handler. [2 marks]
 - (c) Explain the difference between vectored and non-vectored interrupt handling. Which is better? [4 marks]
 - (d) Design and write the code for a main program and an interrupt handler, to make an LED flash a particular number of times. The interrupt handler is called by a timer interrupt every one millisecond (1 ms). Each flash of the LED should consist of 60 ms of the LED being lit followed by 40 ms of the LED being off; thus, for example, it would take 5 seconds to flash the LED 50 times. When the LED has flashed the required number of times, it should remain dim for a further 500 ms.

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 to the LED has already been set up. To turn on the LED, write a 1 to bit 7 of location 0x0E004003A; to turn it off, write a 0. [12 marks]

ASCII Code

C 1 NT 1

	Colun	nn Numl	oer					
_	000	001	010	011	100	101	110	111
Row Number								
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	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	V
0111	BELL	ETB	ŧ	7	G	W	g	W
1000	BS	CAN	(8	Н	X	h	X
1001	HT	EM)	9	I	Y	i	y
1010	LF	SUB	*	:	J.	Z	j	\mathbf{Z}
1011	VT	ESC	+	;	K	[k	{
1100	FF	FS	,	<	L	\	1	ı
1101	CR	GS	-	=	M]	m	}
1110	SO	RS	•	>	N	٨	n	~
1111	SI	US	/	?	Ο		О	DEL

The ASCII code of a character is found by combining its Column Number (given in 3-bit binary) with its Row Number (given in 4-bit binary).

The Column Number forms bits 6, 5 and 4 of the ASCII, and the Row Number forms bits 3, 2, 1 and 0 of the ASCII.

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

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

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

ARM® Instruction Set Quick Reference Card

veh to lanes	
{cond}	Refer to Table Condition Field. Omit for unconditional execution.
<operand2></operand2>	Refer to Table Flexible Operand 2. Shift and rotate are only available as part of Operand2.
<fields></fields>	Refer to Table PSR fields.
<psr></psr>	Either CPSR (Current Processor Status Register) or SPSR (Saved Processor Status Register)
{s}	Updates condition flags if S present.
C*, V*	Flag is unpredictable in Architecture v4 and earlier, unchanged in Architecture v5 and later.
Ø	Sticky flag. Always updates on overflow (no S option). Read and reset using MRS and MSR.
GE	Four Greater than or Equal flags. Always updated by parallel adds and subtracts.
x,y	B meaning half-register [15:0], or T meaning [31:16].
<immed_8r></immed_8r>	A 32-bit constant, formed by right-rotating an 8-bit value by an even number of bits.
{x}	RsX is Rs rotated 16 bits if X present. Otherwise, RsX is Rs.
<prefix></prefix>	Refer to Table Prefixes for Parallel instructions
<p_mode></p_mode>	Refer to Table Processor Modes
R13m	R13 for the processor mode specified by <p_mode></p_mode>

	{endianness}	Can be BE (Big Endian) or LE (Little Endian).
	<a_mode2></a_mode2>	Refer to Table Addressing Mode 2.
of Operand2.	<a_mode2p></a_mode2p>	Refer to Table Addressing Mode 2 (Post-indexed only).
	<a_mode3></a_mode3>	Refer to Table Addressing Mode 3.
us Register)	<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).
5 and later.	<a_mode5></a_mode5>	Refer to Table Addressing Mode 5.
RS and 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>
bits.	{:}	Updates base register after data transfer if ! present.
	+/-	+ or (+ may be omitted.)
	∞s	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.

Operation		§ Assembler	Su	Supdates	Š	ຄ	_	Action
Arithmetic Add		ADD{cond}{S} Rd, Rn, <operand2></operand2>	z	Z		<	_	Rd := Rn + Operand2
	with carry	ADC(cond)(S) Rd, Rn, <operand2></operand2>	Z	Z		_	<u>P</u>	Rd := Rn + Operand2 + Carry
		5E QADD{cond} Rd, Rm, Rn				_		Rd := SAT(Rm + Rn)
_	double saturating	5E QDADD {cond} Rd, Rm, Rn					$\frac{Q}{R}$	Rd := SAT(Rm + SAT(Rn * 2))
	Subtract	SUB{cond}{S} Rd, Rn, <operand2></operand2>	z	Z	C.	<	Ŗ	Rd := Rn - Operand2
	with carry	Rd,	z	Z		<u> </u>	ᄝ	Rd := Rn - Operand2 - NOT(Carry)
	reverse subtract	RSB{cond}{S} Rd, Rn, <operand2></operand2>	z	Z	C	<	Ŗ	Rd := Operand2 Rn
	reverse subtract with carry	RSC{cond}{S} Rd, Rn, <operand2></operand2>	z	2	Ω	<	<u>بر</u>	Rd := Operand2 - Rn - NOT(Carry)
	saturating	5E QSUB{cond} Rd, Rm, Rn				_		Rd := SAT(Rm - Rn)
	double saturating	5E QDSUB{cond} Rd, Rm, Rn				_	Q R	Rd := SAT(Rm - SAT(Rn * 2))
	Multiply	2 MUL{cond}{S} Rd, Rm, Rs	z	Z	Ç*		Ŗ	Rd := (Rm * Rs)[31:0]
	and accumulate	2 MLA{cond}{S} Rd, Rm, Rs, Rn	z		ů,		Ŗ	Rd := ((Rm * Rs) + Rn)[31:0]
	unsigned long	M UMULL {cond} {S} RdLo, RdHi, Rm, Rs	z	Z	C* 1	٧ <u>*</u>	R	RdHi,RdLo := unsigned(Rm * Rs)
	unsigned accumulate long	M UMLAL {cond} {S} RdLo, RdHi, Rm, Rs		Z	C* \	٧ <u>*</u>	R	RdHi,RdLo := unsigned(RdHi,RdLo + Rm * Rs)
	unsigned double accumulate long	6 UMAAL (cond) RdLo, RdHi, Rm, Rs					×	RdHi,RdLo := unsigned(RdHi + RdLo + Rm * Rs)
	Signed multiply long	M SMULL {cond} {S} RdLo, RdHi, Rm, Rs	Z	Z	C* 1	<u>*</u>	Ŗ	RdHi,RdLo := signed(Rm * Rs)
	and accumulate long	M SMLAL(cond) (S) RdLo, RdHi, Rm, Rs	z	Z	C* 1	*	Ŗ	RdHi,RdLo := signed(RdHi,RdLo + Rm * Rs)
	16 * 16 bit	5E SMULxy{cond} Rd, Rm, Rs					70	Rd := Rm[x] * Rs[y]
	32 * 16 bit	5E SMULWy (cond) Rd, Rm, Rs					Z	Rd := (Rm * Rs[y])[47:16]
	16 * 16 bit and accumulate	5E SMLAxy{cond} Rd, Rm, Rs, Rn				_	Q	Rd := Rn + Rm[x] * Rs[y]
	32 * 16 bit and accumulate	5E SMLAWY{cond} Rd, Rm, Rs, Rn				_	QR	Rd := Rn + (Rm * Rs[y])[47:16]
	16 * 16 bit and accumulate long	5E SMLALxy{cond} RdLo, RdHi, Rm, Rs					R	RdHi,RdLo := RdHi,RdLo + Rm[x] * Rs[y]
	Dual signed multiply, add	6 SMUAD{x}{cond} Rd, Rm, Rs				_	QR	Rd := Rm[15:0] * RsX[15:0] + Rm[31:16] * RsX[31:16]
	and accumulate	6 SMLAD{X}{cond} Rd, Rm, Rs, Rn				_	QR	Rd := Rn + Rm[15:0] * RsX[15:0] + Rm[31:16] * RsX[31:16]
	and accumulate long	6 SMLALD{X}{cond} RdHi, RdLo, Rm, Rs	tri			_		RdHi,RdLo := RdHi,RdLo + Rm[15:0] * RsX[15:0] + Rm[31:16] * RsX[31:16]
	Dual signed multiply, subtract	6 SMUSD{x}{cond} Rd, Rm, Rs				_	QR	Rd := Rm[15:0] * RsX[15:0] - Rm[31:16] * RsX[31:16]
	and accumulate	6 SMLSD{x}{cond} Rd, Rm, Rs, Rn				_	QR	Rd := Rn + Rm[15:0] * RsX[15:0] - Rm[31:16] * RsX[31:16]
	and accumulate long	6 SMLSLD{X}{cond} RdHi, RdLo, Rm, Rs	ta			_		RdHi,RdLo := RdHi,RdLo + Rm[15:0] * RsX[15:0] - Rm[31:16] * RsX[31:16]
	Signed most significant word multiply	6 SMMUL{R}{cond} Rd, Rm, Rs				_	R	Rd := (Rm * Rs)[63:32]
	and accumulate	6 SMMLA{R}{cond} Rd, Rm, Rs, Rn					R	Rd := Rn + (Rm * Rs)[63:32]
	and subtract	6 SMMLS{R}{cond} Rd, Rm, Rs, Rn					R	Rd := Rn - (Rm * Rs)[63:32]
	Multiply with internal 40-bit accumulate XS MIA (cond) Ac, Rm, Rs	XS MIA{cond} Ac, Rm, Rs	-				A	Ac := Ac + Rm * Rs
	packed halfword	XS MIAPH {cond} Ac, Rm, Rs	-				Α	Ac := Ac + Rm[15:0] * Rs[15:0] + Rm[31:16] * Rs[31:16]
	halfword	XS MIAxy{cond} Ac, Rm, Rs					>	Ac := Ac + Rm[x] * Rs[y]
	Count leading zeroes	5 CLZ(cond) Rd, Rm				_	_	Rd := number of leading zeroes in Rm

ARM Addressing Modes Quick Reference Card

	m	Assembler	Sundates	0	E Action
Halfword-wise addition	6	<pre><prefix>ADD16{cond} Rd, Rn, Rm</prefix></pre>	,	0	E Rd[31:16] := Rn[31:16] + Rm[31:16], Rd[15:0] := Rn[15:0] + Rm[15:0]
arithmetic Halfword-wise subtraction	6	<pre><prefix>SUB16{cond} Rd, Rn, Rm</prefix></pre>		0	
Byte-wise addition	6	<pre><prefix>ADD8{cond} Rd, Rn, Rm</prefix></pre>			E 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>		0	E 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>		0	Rd[31:16] := Rn[31:16]
Halfword-wise exchange, subtract, add	6	<pre><prefix>SUBADDX{cond} Rd, Rn, Rm</prefix></pre>		0	Rd[31:16] := Rn[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		MOV{cond}{S} Rd, <operand2></operand2>	NZC		Rd := Operand2
NOT		MVN(cond)(S) Rd, <operand2></operand2>			Rd := 0xFFFFFFF EOR Operand2
PSR to register	3	MRS{cond} Rd, <psr></psr>			Rd := PSR
register to PSR	w	MSR{cond} <psr>_<fields>, Rm</fields></psr>			PSR := Rm (selected bytes only)
immediate to PSR	s	MSR(cond) <psr>_<fields>, #<immed_8r></immed_8r></fields></psr>			PSR := immed_8r (selected bytes only)
40-bit accumulator to register	XX	MRA{cond} RdLo, RdHi, Ac			RdLo := Ac[31:0], RdHi := Ac[39:32]
register to 40-bit accumulator	X	Ac,			Ac[31:0] := RdLo, Ac[39:32] := RdHi
Test	,	R	- 1	+	IIndate CPOR flags on Rn AND Operand?
Test equivalence		TEQ{cond} Rn, <operand2></operand2>	N		Update CPSR flags on Rn EOR Operand2
AND		AND{cond}{S} Rd, Rn, <operand2></operand2>			Rd := Rn AND Operand2
EOR		<pre>EOR{cond}{S} Rd, Rn, <operand2></operand2></pre>	2		Rd := Rn EOR Operand2
ORR		ORR{cond}{S} Rd, Rn, <operand2></operand2>			Rd := Rn OR Operand2
Bit Clear		BIC(cond){S} Rd, Rn, <operand2></operand2>			Rd := Rn AND NOT Operand2
Compare		CMP{cond} Rn, <operand2></operand2>	NZCV		Update CPSR flags on Rn - Operand2
negative		CMN {cond} Rn, <operand2></operand2>	NZCV		Update CPSR flags on Rn + Operand2
Signed saturate word, right shift	6	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>		0	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	<pre>USAT{cond} Rd, #<sat>, Rm{, ASR <sh>}</sh></sat></pre>		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>		0	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>		0	Rd[31:16] := UnsignedSat(Rm[31:16], sat), Rd[15:0] := UnsignedSat(Rm[15:0], sat). <sat> range 0-15.</sat>
	Halfword-wise addition Halfword-wise subtraction Byte-wise addition Byte-wise addition Halfword-wise exchange, add, subtract Halfword-wise exchange, subtract, add Unsigned sum of absolute differences and accumulate Move NOT PSR to register register to PSR immediate to PSR immed	add, subtract subtract, add differences differences thirt register mulator shift tshift tshift	S Assembler	S Assembler	S Assembler S updates

ARM Instruction Set Quick Reference Card

Operation		Ş	Assembler	Action	Notes
Pack Pa	Pack halfword bottom + top Pack halfword top + bottom	6 6	PKHTB{cond} Rd, Rn, Rm{, LSL # <sh>} PKHTB{cond} Rd, Rn, Rm{, ASR #<sh>}</sh></sh>	Rd[15:0] := Rn[15:0], $Rd[31:16] := (Rm LSL sh)[31:16]$, sh 0-31. Rd[31:16] := Rn[31:16], $Rd[15:0] := (Rm ASR sh)[15:0]$, sh 1-32.	
Signed H	Halfword to word	6	Rm{, ROR	Rd[31:0] := SignExtend((Rm ROR (8 * sh))[15:0]). sh 0-3.	
	Two bytes to halfwords	6	SXTB16{cond} Rd, Rm{, ROR # <sh>}</sh>	Rd[31:16] := SignExtend((Rm ROR (8 * sh))[23:16]), Rd[15:0] := SignExtend((Rm ROR (8 * sh))[7:0]). sh 0-3.	
В	Byte to word	6	SXTB(cond) Rd, Rm{, ROR # <sh>}</sh>	Rd[31:0] := SignExtend((Rm ROR (8 * sh))[7:0]). sh 0-3.	
æ	Halfword to word	9	Rm{,	Rd[31:0] := ZeroExtend((Rm ROR (8 * sh))[15:0]). sh 0-3.	
extend T	Two bytes to halfwords	6	UXTB16{cond} Rd, Rm{, ROR # <sh>}</sh>	Rd[31:16] := ZeroExtend((Rm ROR (8 * sh))[23:16]), Rd[15:0] := ZeroExtend((Rm ROR (8 * sh))[7:0]). sh 0-3.	
В	Byte to word	6	UXTB{cond} Rd, Rm{, ROR # <sh>}</sh>	Rd[31:0] := ZeroExtend((Rm ROR (8 * sh))[7:0]). sh 0-3.	
Signed H	Halfword to word, add	6	, Rn,	Rd[31:0] := Rn[31:0] + SignExtend((Rm ROR (8 * sh))[15:0]). sh 0-3.	
<u> </u>	Two bytes to halfwords, add	6	SXTAB16{cond} Rd, Rn, Rm{, ROR # <sh>}</sh>	Rd[31:16] := Rn[31:16] + SignExtend((Rm ROR (8 * sh))[23:16]), Rd[15:0] := Rn[15:0] + SignExtend((Rm ROR (8 * sh))[7:0]), sh 0-3.	
В	Byte to word, add	6	SXTAB{cond} Rd, Rn, Rm{, ROR # <sh>}</sh>		
Unsigned H	Halfword to word, add	6	Rn, Rm{,	[Rd[31:0] := Rn[31:0] + ZeroExtend((Rm ROR (8 * sh))[15:0]). sh 0-3.	
	Two bytes to halfwords, add	6	UXTAB16{cond} Rd, Rn, Rm{, ROR # <sh>}</sh>		
В	Byte to word, add	6	UXTAB{cond} Rd, Rn, Rm{, ROR # <sh>}</sh>		
Reverse In	In word	9		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[31:94] := Rm[73:16] Rd[73:16] := Rm[31:94]	
- In	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 S	Select bytes	6	SEI{cond} Rd, Rn, Rm	Rd[7:0] := Rn[7:0] if $GE[0] = 1$, else $Rd[7:0] := Rm[7:0]Bits[15:8], [23:16], [31:24] selected similarly by GE[1], GE[2], GE[3]$	
Branch B	Branch		B{cond} label	R15 := label	label must be within ±32Mb of current instruction.
	with link		BL{cond} label	R14 := address of next instruction, R15 := label	label must be within ±32Mb of current instruction.
	and exchange with link and exchange (1)	4T,5 5T		R15 := Rm, Change to Thumb if Rm[0] is 1 R14 := address of next instruction. R15 := label. Change to Thumb	Cannot be co
	WITH HIER AND EXCHANGE (1)	٢	דיט דממפד	N.1+ .= addess of sext instruction, N.13 .= ladel, Change to Thumb	label must be within ±32Mb of current instruction.
	with link and exchange (2)	5	BLX{cond} Rm	R14 := address of next instruction, R15 := Rm[31:1] Change to Thumb if Rm[0] is 1	
	and change to Java state	51, 6	BXJ{cond} Rm	Change to Java state	
essor	Change processor state	6	CPSID <iflags> {, #<p_mode>}</p_mode></iflags>	Disable specified interrups, optional change mode.	Cannot be conditional
	•	6	CPSIE <iflags> {, #<p_mode>}</p_mode></iflags>	Enable specified interrups, optional change mode.	Cannot be conditional.
Change	Change processor mode	6	CPS # <p_mode></p_mode>		Cannot be conditional.
- S	Set endianness	6	SETEND <endianness></endianness>	Sets endianness for loads and saves. <endianness> can be BE (Big Endian) or LE (Little Endian).</endianness>	Cannot be conditional
S	Store return state	6	SRS <a_mode4s> #<p_mode>{!}</p_mode></a_mode4s>	[R13m] := R14, [R13m + 4] := CPSR	Cannot be conditional
R	Return from exception	6	RFE <a_mode4l> Rn{!}</a_mode4l>	PC := [Rn], CPSR := [Rn + 4]	Cannot be conditional
	Breakpoint	5	BKPT <immed_16></immed_16>	Prefetch abort or enter debug state.	Cannot be conditional
Software Sinterrupt	Software interrupt		SWI{cond} <immed_24></immed_24>	Software interrupt processor exception.	24-bit value encoded in instruction.
No Op	No operation	5	NOP	None	

ARM Addressing Modes Quick Reference Card

Load Word Lord	Operation		ω	Assembler	Action	Notes
User mode privilege LDR (cond) R 15, <a_mode2p> LDR (cond) R 15, <a_mode2p> R15 = [address][3]:1] </a_mode2p></a_mode2p>		Word		LDR{cond} Rd, <a_mode2></a_mode2>		Rd must not be R15.
Byre		User mode privilege		LDR{cond}T Rd, <a_mode2p></a_mode2p>		Rd must not be R15.
Byte LDR cond B Rd, <a_mode2> Kd := ZeroExtend(byte from address) is 1) </a_mode2>		branch (§ 5T: and exchange)		LDR{cond} R15, <a_mode2></a_mode2>	R15 := [address][31:1]	
Byve LDR {cond} Br Rd, <a_mode2> Rd := ZeroExtend(byte from address) </a_mode2>					(§ 5T: Change to Thumb if [address][0] is 1)	
LDR (cond) BT Rd, <a_mode2p> Rd := SignExtend[byte from address] </a_mode2p>		Byte		LDR{cond}B Rd, <a_mode2></a_mode2>		Rd must not be R15.
Signed A LDR{cond}SB Rd, <a_mode3> Rd := SignExtend[byte from address] </a_mode3>		User mode privilege		LDR{cond}BT Rd, <a_mode2p></a_mode2p>		Rd must not be R15.
Halfword Halfword Halfword Halfword Halfword Halfword Halfword Halfword Daubleword Doubleword Doubleword Doubleword Pop. or Block data load return (and exchange) LDM {cond} > H.DM {cond} > R.d. = mode3 > Rd := SignExtend[halfword from address] Rd := Rd := SignExtend[halfword from address] Rd := Rd := SignExtend[halfword from address] Load its of registers, branch (§ ST: address][31:1] Load registers, branch (§ ST: address][31:1] Rd := [Rn], tag address as exclusive access PSR Load registers, branch (§ ST: address][31:1] Rd := [Rn], tag address] as registers to load from address Rd := SignExtend[halfword from address ST: Cange to Thumb if laddress][31:1] Rd := [Rn], tag address as exclusive access Rd := Rd := [Rn], tag address as exclusive access Rd := SignExtend[s] ST: address][31:1] Rd := [Rn], tag address] Stag address as exclusive access Rd := Rd := [Rn], tag address] Stag address as exclusive access		signed	4	LDR{cond}SB Rd, <a_mode3></a_mode3>		Rd must not be R15.
Signed Doubleword Doubleword Doubleword Push, or Block data load return (and exchange) LDM {cond} > Rd. = mode3 > Rd := SignExtend[halfword from address] LDM {cond} > Rd. = mode4.		Halfword	4	LDR{cond}H Rd, <a_mode3></a_mode3>		Rd must not be R15.
multiple Doubleword Pop, or Block data load return (and exchange) 5F* LDR {cond} D Rd, <a_mode3> Rd := [address], R(d+1) := [address + 4] Pop, or Block data load return (and exchange) LDM {cond} <a_mode4l> Rn {!}, <reglist+pc> Load list of registers, R15 := [address][31:1] User mode registers LDM {cond} <a_mode4l> Rn {!}, <reglist+pc> Load registers, R15 := [address][31:1] Word LDM {cond} <a_mode4l> Rn {!}, <reglist+pc> Load list of registers, R15 := [address][31:1] LDM {cond} <a_mode4l> Rn {!}, <reglist+pc> Load list of User mode registers branch (§ 5T: and exchange), CPSR := SPSR Word LDM {cond} <a_mode4l> Rn {!}, <reglist-pc>^ Load list of User mode registers from [Rn] Memory system hint 5F* PLD <a_mode4l> Rn {!}, <reglist-pc>^ Load list of User mode registers from [Rn] Memory system hint 5F* PLD <a_mode4l> Rn {!}, <reglist-pc>^ Load list of User mode registers from [Rn] Memory system hint 5F* PLD <a_mode4l> Rn {!}, <reglist-pc>^ Load list of User mode registers from [Rn] Memory may prepare to load from address Rd := [Rn], use address as exclusive access Memory may prepare to load from address Rd := [Rn], use address as exclusive access Memory may prepare to load from address Rd := [Rn], use address = Rd User mode privilege STR {cond} Rd, <a_mode2> [address][7:0] := Rd[7:0] STR {cond} Rd, Rd, <a_mode3< td=""> [address][7:0] := Rd][7:0]<</a_mode3<></a_mode2></reglist-pc></a_mode4l></reglist-pc></a_mode4l></reglist-pc></a_mode4l></reglist-pc></a_mode4l></reglist+pc></a_mode4l></reglist+pc></a_mode4l></reglist+pc></a_mode4l></reglist+pc></a_mode4l></a_mode3>		signed	4	LDR{cond}SH Rd, <a_mode3></a_mode3>		Rd must not be R15.
multiple Pop. or Block data load return (and exchange) LDM(cond)		Doubleword	5E*			Rd must be even, and not
return (and exchange) and restore CPSR User mode registers Word Byte return (and exchange) LDM(cond) <a_mode41> Rn{!}, <reg1ist+pc> (§ 5T: Change to Thumb if [address][31:1] LDM(cond) <a_mode41> Rn {!}, <reg1ist+pc> (S 5T: Change to Thumb if [address][0] is 1) LDM(cond) <a_mode41> Rn {!}, <reg1ist+pc> (S 5T: Change to Thumb if [address][0] is 1) LDM(cond) <a_mode41> Rn {!}, <reg1ist+pc> (S 5T: Change to Thumb if [address][0] is 1) LDM(cond) <a_mode41> Rn {!}, <reg1ist+pc> (S 5T: Change to Thumb if [address][0] is 1) LDM(cond) <a_mode41> Rn {!}, <reg1ist+pc> (S 5T: Change to Thumb if [address][0] is 1) LDM(cond) <a_mode41> Rn {!}, <reg1ist+pc> (S 5T: Change to Thumb if [address][0] is 1) Memory system hint SE* PLD <a_mode2> Chad is to User mode registers from [Rn] Memory may prepare to load from address Rd := [Rn], tag address as exclusive access Outstanding tag set if not shared address [address] := Rd [address] := Rd [address] := Rd [address] := Rd [address] := Rd [address] := Rd[7:0] [address] := Rd[7:0] := Rd[7:0] [address] := Rd, [address + 4] := R</a_mode2></reg1ist+pc></a_mode41></reg1ist+pc></a_mode41></reg1ist+pc></a_mode41></reg1ist+pc></a_mode41></reg1ist+pc></a_mode41></reg1ist+pc></a_mode41></reg1ist+pc></a_mode41>		Pop, or Block data load		LDM(cond) <a_mode4l> Rn(!), <reglist-pc></reglist-pc></a_mode4l>	Load list of registers from [Rn]	
And restore CPSR LDM cond camode4L		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)	
User mode registers LDM (cond) < a_mode4L> Rn, <reglist-pc></reglist-pc>		and restore CPSR		LDM(cond) <a_mode4l> Rn(!), <reglist+pc>^</reglist+pc></a_mode4l>	Load registers, branch (§ 5T: and exchange), CPSR := SPSR	Use from exception modes
reload Memory system hint sxclusive Semaphore operation Word User mode privilege Byte User mode privilege Halfword Doubleword Push, or Block data store User mode registers STM {cond} Rd, <a_mode2></a_mode2>		User mode registers		LDM(cond) <a_mode4l> Rn, <reglist-pc>^</reglist-pc></a_mode4l>	Load list of User mode registers from [Rn]	Use from privileged mode
exclusive Semaphore operation 6 LDREX{cond} Rd, [Rn] Rd := [Rn], tag address as exclusive access Word Word STR{cond} Rd, <a_mode2> [address] := Rd User mode privilege STR{cond} Rd, <a_mode2> [address] := Rd User mode privilege STR{cond} Rd, <a_mode2> [address] [7:0] := Rd[7:0] Halfword STR{cond} BT Rd, <a_mode2> [address] [7:0] := Rd[7:0] Doubleword STR{cond} Rd, <a_mode3> [address] [7:0] := Rd[7:0] := Rd[7:0] Push, or Block data store STM{cond} A _a_mode4S > Rn [! }, <reg1ist> Store list of registers to [Rn] User mode registers STM{cond} A _a_mode4S > Rn [! }, <reg1ist> Store list of registers to [Rn] Semaphore operation STREX{cond} Rd, Rm, [Rn] [Rn] := Rm if allowed, Rd := 0 if successful, else 1 Word 3 SWP{cond} Rd, Rm, [Rn] [Rn] := Rm, Rd := temp Byte 3 SWP{cond} Rd, Rm, [Rn] [Rn] := Rm, Rd := temp</reg1ist></reg1ist></a_mode3></a_mode2></a_mode2></a_mode2></a_mode2>		Memory system hint	5E*			Cannot be conditional.
Word STR {cond} Rd, <a_mode2> [address] := Rd User mode privilege STR {cond}T Rd, <a_mode2p> [address] := Rd Byte User mode privilege STR {cond}B Rd, <a_mode2p> [address][7:0] := Rd[7:0] User mode privilege STR {cond}BT Rd, <a_mode2p> [address][7:0] := Rd[7:0] := Rd[7:0] Halfword 4 STR {cond}BT Rd, <a_mode2p> [address][7:0] := Rd[7:0] :=</a_mode2p></a_mode2p></a_mode2p></a_mode2p></a_mode2>	Load exclusive	Semaphore operation	0	LDREX(cond) Rd, [Rn]	SS	Rd, Rn must not be R15.
User mode privilege		Word		STR{cond} Rd, <a_mode2></a_mode2>	[address] := Rd	
Byte		User mode privilege	,	STR{cond}T Rd, <a_mode2p></a_mode2p>	[address] := Rd	
User mode privilege Halfword Doubleword Doubleword Push, or Block data store User mode registers User mode registers Semaphore operation Word Byte User mode privilege 4 STR {cond} F Rd, <a_mode2p> [address][7:0] := Rd[7:0] FRD { cond} F Rd, <a_mode3> [address][15:0] := Rd[15:0] STM {cond} F Rd, <a_mode3> [address] := Rd, [address] + 4] := R(d+1) STM {cond} F Rd, = mode4S F Rn { ! }, <reg1ist> STM {cond} F Rd, Rm, [Rn] Rd := 0 if successful, else 1 Rd := 0 if successful, else 1 Byte STM {cond} F Rd, Rm, [Rn] STM {cond} F Rd, Rm, [Rn] FTM {cond} F Rd, Rm, Rd := temp FTM {cond} F Rd, Rm, [Rn] FTM {cond} F Rd, Rd := temp FTM {cond} F Rd, Rm, Rd := temp FTM {con</reg1ist></a_mode3></a_mode3></a_mode2p>		Byte		STR{cond}B Rd, <a_mode2></a_mode2>	[address][7:0] := Rd[7:0]	
Halfword Halfword Halfword Halfword Halfword Halfword A STR{cond}H Rd, <a_mode3> Doubleword SE* STR{cond}D Rd, <a_mode3> Push, or Block data store User mode registers User mode registers Semaphore operation Word Halfword A STR{cond}H Rd, <a_mode3> STM{cond} <a_mode4s> Rn{!}, <reglist> Store list of registers to [Rn] STM{cond} <a_mode4s> Rn{!}, <reglist> Store list of User mode registers to [Rn] Rd := Rm if allowed. Rd := 0 if successful, else 1 Rd := 0 if successful, else 1</reglist></a_mode4s></reglist></a_mode4s></a_mode3></a_mode3></a_mode3>		User mode privilege		STR{cond}BT Rd, <a_mode2p></a_mode2p>	[address][7:0] := Rd[7:0]	
multiple multiple Doubleword 5E* STR{cond}D Rd, <a_mode3> [address] := Rd, [address + 4] := R(d+1) Push, or Block data store STM{cond}<a_mode45> Rn{!}, <reglist> Store list of registers to [Rn] Exclusive Semaphore operation STREX{cond} Rd, Rm, [Rn] [Rn] := Rm if allowed, Rd := 0 if successful, else 1 Word 3 SWP{cond} Rd, Rm, [Rn] [Rn] := Rm, Rd := temp Byte 3 SWP{cond}B Rd, Rm, [Rn] [Rn] [Rn] := Rm, Rd := temp</reglist></a_mode45></a_mode3>		Halfword	4	STR{cond}H Rd, <a_mode3></a_mode3>	[address][15:0] := Rd[15:0]	
multiple Push, or Block data store STM{cond} <a_mode4\$> Rn{!}, <reglist> Store list of registers to [Rn] User mode registers STM{cond}<a_mode4\$> Rn{!}, <reglist>^ Store list of User mode registers to [Rn] Exclusive Semaphore operation 6 STREX{cond} Rd, Rm, [Rn] [Rn]:= Rm if allowed, Rd:= temp Word 3 SWP{cond} Rd, Rm, [Rn] temp := [Rn], [Rn] := Rm, Rd := temp Byte 3 SWP{cond} Rd, Rm, [Rn] temp := ZeroExtend([Rn][7:0]).</reglist></a_mode4\$></reglist></a_mode4\$>		Doubleword	5E*			Rd must be even, and not l
User mode registers SIM{cond} <a_mode4s> Rn{!}, <reglist>^ Store list of User mode registers to [Rn] Byte Semaphore operation SIME(cond) Rd, Rm, [Rn] SIME(cond) Rd, Rm</reglist></a_mode4s>		Push, or Block data store		STM{cond} <a_mode4s> Rn{!}, <reglist></reglist></a_mode4s>	Store list of registers to [Rn]	
exclusive Semaphore operation 6 STREX{cond} Rd, Rm, [Rn] [Rn] := Rm if allowed, Rd := 0 if successful, else 1 Word 3 SWP{cond} Rd, Rm, [Rn] temp := [Rn], [Rn] := Rm, Rd := temp Byte 3 SWP{cond} Rd, Rm, [Rn] temp := [Rn], [Rn] := Rm if allowed, Rd := temp Byte 3 SWP{cond} Rd, Rm, [Rn] temp := [Rn], [Rn] := Rm if allowed, Rd := temp		User mode registers		STM{cond} <a_mode4s> Rn{!}, <reglist>^</reglist></a_mode4s>	Store list of User mode registers to [Rn]	Use from privileged modes
Word 3 SWP{cond} Rd, Rm, [Rn] Byte 3 SWP{cond}B Rd, Rm, [Rn]	Store exclusive	Semaphore operation	6			Rd, Rm, Rn must not be R
Byte 3 SWP{cond}B Rd, Rm, [Rn]		Word	3	SWP{cond} Rd, Rm, [Rn]	temp := [Rn], [Rn] := Rm, Rd := temp	
		Byte	ü	SWP{cond}B Rd, Rm, [Rn]	temp := ZeroExtend([Rn][7:0]),	

Page 9 of 11

Quick Reference Card ARM Addressing Modes

		, , ,	
Pre-indexed	Immediate offset	[Rn, #+/- <immed_12>]{!}</immed_12>	
	Zero offset	[Rn]	Equivalent to [Rn,#0]
	Register offset	[Rn, +/-Rm]{!}	
	Scaled register offset	Scaled register offset [Rn, +/-Rm, LSL # <shift>] {!} Allowed shifts 0-31</shift>	Allowed shifts 0-31
		[Rn, +/-Rm, LSR # <shift>] {!} Allowed shifts 1-32</shift>	Allowed shifts 1-32
		[Rn, +/-Rm, ASR # <shift>] { Allowed shifts 1-32</shift>	Allowed shifts 1-32
		[Rn, +/-Rm, ROR # <shift>] {!} Allowed shifts 1-31</shift>	Allowed shifts 1-31
		[Rn, +/-Rm, RRX] {!}	
Post-indexed	Immediate offset	[Rn], #+/- <immed_12></immed_12>	
	Register offset	[Rn], +/-Rm	
	Scaled register offset	Scaled register offset [Rn], +/-Rm, LSL # <shift></shift>	Allowed shifts 0-31
		[Rn], +/-Rm, LSR # <shift></shift>	Allowed shifts 1-32
		[Rn], +/-Rm, ASR # <shift></shift>	Allowed shifts 1-32
		[Rn], +/-Rm, ROR # <shift></shift>	Allowed shifts 1-31
		[Rn], +/-Rm, RRX	

Addressing	Addressing Mode 2 (Post-indexed only)	id only)				
Post-indexed	Immediate offset	[Rn],	[Rn], #+/- <immed_12></immed_12>	nmed_	_12>	
	Zero offset	[Rn]				Equivalent to [Rn],#0
	Register offset	[Rn],	+/-Rm			
	Scaled register offset [Rn] , +/-Rm, LSL # <shift></shift>	[Rn],	+/-Rm,	TST	# <shift></shift>	Allowed shifts 0-31
		[Rn],	+/-Rm,	LSR	+/-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		

Addressing	Addressing Mode 2 (Post-indexed only)	d only)				
Post-indexed	Immediate offset	[Rn],	[Rn], #+/- <immed_12></immed_12>	nmed_	12>	
	Zero offset	[Rn]				Equivalent to [Rn],#0
	Register offset	[Rn],	+/-Rm			
	Scaled register offset [Rn] , +/-Rm, LSL # <shift></shift>	[Rn],	+/-Rm,	TST	# <shift></shift>	Allowed shifts 0-31
		[Rn],	+/-Rm,	LSR	+/-Rm, LSR # <shift></shift>	Allowed shifts 1-32
		[Rn],		ASR	+/-Rm, ASR # <shift></shift>	Allowed shifts 1-32
		[Rn],	+/-Rm,	ROR	+/-Rm, ROR # <shift></shift>	Allowed shifts 1-31
		[Rn],	[Rn], +/-Rm, RRX	RRX		

Addressing	Addressing Mode 2 (Post-indexed only)	≱d only)				
Post-indexed	Immediate offset	[Rn],	[Rn], #+/- <immed_12></immed_12>	mmed_	12>	
	Zero offset	[Rn]				Equivalent to [Rn],#0
	Register offset	[Rn],	+/-Rm			
	Scaled register offset	[Rn],	+/-Rm,	TST	[Rn], +/-Rm, LSL # <shift></shift>	Allowed shifts 0-31
		[Rn],	+/-Rm,	LSR	+/-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		

7		The state of the s	7	77 EF	
Id	Condition Field		[Rn, #+/- <immed_8>]{!}</immed_8>	Pre-indexed Immediate offset [Rn, #+/- <immed_8>] {!}</immed_8>	
		ransfer	Addressing Mode 3 - Halfword, Signed Byte, and Doubleword Data Transfer	Addressing Mode 3 - Halfword, S	
Extension field mask byte	×				_
Status field mask byte	m		[Rn], +/-Rm, RRX		
Flags field mask byte	Hs.	Allowed shifts 1-31	[Rn], +/-Rm, ROR # <shift></shift>		
Control field mask byte	Ω	Allowed shifts 1-32	[Rn], +/-Rm, ASR # <shift></shift>		
Meaning	Suffix	Allowed shifts 1-32	[Rn], +/-Rm, LSR # <shift></shift>		
(use at least one suffix)	PSR fields	Allowed shifts 0-31	Scaled register offset [Rn], +/-Rm, LSL # <shift></shift>	Scaled register offset	
			[Rn], +/-Rm	Register offset	_
Ster Rm,	Rotate right register	Equivalent to [Rn],#0	[Rn]	Zero offset	
0	0				_

	ARM architecture versions	ture versions					
	п	ARM architecture version n and above	on n	and ab	ove.		
	nT, nJ	T or J variants of ARM architecture version n and above	archi	tectur	e version n and abov	/e.	
	Z	ARM architecture version 3M, and 4 and above, except xM variants.	ion 3N	A, and	4 and above, except	t xM variants.	
	πE	All E variants of ARM architecture version n and above	archit	ecture	version n and above	ŗ.	
	nE*	E variants of ARM architecture version n and above, except xP variants.	nitecti	ire vei	rsion n and above, ex	xcept xP variants.	
	XS	XScale coprocessor instruction	tructi	on			
	Flexible Operand 2	and 2					
	Immediate value	•	# <ir< th=""><th>#<immed_8r></immed_8r></th><th>8r></th><th></th><th></th></ir<>	# <immed_8r></immed_8r>	8r>		
	Logical shift left immediate	t immediate	Rm,	$_{\rm ISI}$	Rm, LSL # <shift></shift>	Allowed shifts 0-31	
	Logical shift right immediate	ht immediate	Rm,	LSR	LSR # <shift></shift>	Allowed shifts 1-32	
	Arithmetic shift right immediate	right immediate	Rm,	ASR	# <shift></shift>	Allowed shifts 1-32	_
	Rotate right immediate	nediate	Rm,	ROR	ROR # <shift></shift>	Allowed shifts 1-31	
	Register		Rm				
_	Rotate right extended	ended	Rm,	RRX			
	Logical shift left register	t register	Rm,	ISI	Rs		
	Logical shift right register	ht register	Rm,	LSR	Rs		
	Arithmetic shift right register	right register	Rm,	ASR Rs	Rs		
	Rotate right register	•	R T T	Rm. ROR Rs	Rs		

Condition Field	8			
Mnemonic	Description			Description (VFP)
ΕQ	Equal			Equal
NE	Not equal			Not equal, or unordered
CS / HS	Carry Set / Unsigned higher or same	d higher o	or same	Greater than or equal, or unordered
CC / LO	Carry Clear / Unsigned lower	ned lower	-,	Less than
MI	Negative			Less than
PL	Positive or zero			Greater than or equal, or unordered
VS	Overflow			Unordered (at least one NaN operand)
VC	No overflow			Not unordered
IH	Unsigned higher			Greater than, or unordered
ST	Unsigned lower or same	same		Less than or equal
GE	Signed greater than or equal	or equal		Greater than or equal
LT	Signed less than			Less than, or unordered
GT	Signed greater than			Greater than
ĽΕ	Signed less than or equal	equal		Less than or equal, or unordered
AL	Always (normally omitted)	mitted)		Always (normally omitted)
Processor Modes	des	Prefi	xes for Par	Prefixes for Parallel Instructions
16	User	ß	Signed arith	Signed arithmetic modulo 28 or 216, sets CPSR GE b
17	FIQ Fast Interrupt	Ю	Signed satu	Signed saturating arithmetic
18	IRQ Interrupt	HS	Signed arith	Signed arithmetic, halving results
19	Supervisor	U	Unsigned a	Unsigned arithmetic modulo 28 or 216, sets CPSR GI
23	Abort	Δ'n	Unsigned s	Unsigned saturating arithmetic
27	Undefined	HD	Unsigned a	Unsigned arithmetic, halving results

					Equivalent to [Rn,#0]	
31	27	23	19	18	17	16
System	Undefined	Abort	Supervisor	IRQ Interrupt	FIQ Fast Interrupt	User

Post-indexed Immediate offset Unindexed No offset

[Rn], #+/-<immed_8*4>
[Rn], {8-bit copro. option}

Addressing Mode 5 - Coprocessor Data Transfer

Immediate offset

Zero offset

[Rn]

[Rn, #+/-<immed_8*4>]{!}

Addressing Mode 4 - Multiple Data Transfer

Stack pop

IB DA DB

Decrement After

FD FA EA

Empty Descending
Full Ascending
Empty Ascending

Full Descending

Block store

Stack push

ΙA Block load

IB DA DB

Decrement After

EA FA ED FD

Full Ascending
Empty Descending
Full Descending

Empty Ascending

Decrement Before Increment Before Increment After Decrement Before Increment Before Increment After Post-indexed

Immediate offset Zero offset

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

Equivalent to [Rn,#0]

Register

	Prefi	Prefixes for Parallel Instructions
	ß	Signed arithmetic modulo 28 or 216, sets CPSR GE bits
terrupt	Ю	Signed saturating arithmetic
φt	HS	Signed arithmetic, halving results
	ū	Unsigned arithmetic modulo 28 or 216, sets CPSR GE bits
	Δn	Unsigned saturating arithmetic
	HD	Unsigned arithmetic, halving results

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

Quick Reference Card ARM Addressing Modes

Coprocessor operations	§ Assembler	Action	Notes
Data operations	2 CDP(cond) <copr>, <op1>, CRd, CRn, CRm(, <op2>)</op2></op1></copr>	Coprocessor dependent	
Alternative data operations	5 CDP2 <copr>, <op1>, CRd, CRn, CRm{, <op2>}</op2></op1></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>, <op1>, Rd, CRn, CRm{, <op2>}</op2></op1></copr>	Coprocessor dependent	Cannot be conditional.
Two ARM register move	5E* MRRC(cond) <copr>, <op1>, Rd, Rn, CRm</op1></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	<pre>2 MCR{cond} <copr>, <op1>, Rd, CRn, CRm{, <op2>}</op2></op1></copr></pre>	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>, <op1>, Rd, Rn, CRm</op1></copr>	Coprocessor dependent	
Alternative two ARM register move	6 MCRR2 <copr>, <opl>, Rd, Rn, CRm</opl></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></a_mode5></copr>	Coprocessor dependent	Cannot be conditional.

Proprietary Notice

Words and logos marked with $^{\odot}$ or $^{\rm max}$ 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. Issue A C C C D D D E E G G

ARM QRC 0001H **Document Number**

Change Log

 Date	Ву	Change
June 1995	ВЈН	First Release
Sept 1996	ВЈН	Second Relea
Nov 1998	ВЈН	Third Release
Oct 1999	CKS	Fourth Releas
Oct 2000	CKS	Fifth Release
Sept 2001	CKS	Sixth Release
Jan 2003	CKS	Seventh Relea
Oct 2003	CKS	Eighth Releas