UNIVERSITY OF DUBLIN TRINITY COLLEGE

Faculty of Engineering, Mathematics and Science

School of Computer Science & Statistics

Integrated Engineering Year 3 Examination

Trinity Term 2014

3D2 Microprocessor Systems 2

Friday May 9, 2014

Luce Upper

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:

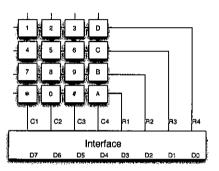
An ASCII code table (one page) and an ARM Instruction Set Summary (six pages) accompany this examination paper.

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 R0, 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

		ın Numb			•			
Row Number	000	001	010	011	100	101	110	111
0000	NUL	DLE	◊	0	@	P	`	p
0001	SOH	DC1	!	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	ЕТВ	I	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	_	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 \(\rightarrow \text{here.} \)

Quick Reference Card ARM® Instruction Set

				control (control and, and, and,	
				SMMI.S(R){cond} Rd Rm Rs	and subtract
				SMMI.A(R)(cond) Rd Rm	
				SMMUL(R)(cond) Rd, Rm,	Signed most significant word multiply 6
0				SMLSLD{X} {cond	and accumulate long 6
0				SMLSD(X) {cond} Rd,	and accumulate 6
0				SMUSD(X)(cond) Rd, Rm, Rs	Dual signed multiply, subtract 6
Q				SMLALD{X}{cond} RdHi, RdLo, Rm, Rs	and accumulate long 6
0				SMLAD(X)(cond) Rd, Rm, Rs, Rn	and accumulate 6
0				SMUAD(x) {cond} Rd,	Dual signed multiply, add 6
					16, 16 bit and accumulate long
<				Shirawa (Cond.) Ku, Ku, Ku, Ku	
) A				contract cond bd bm Ba	
0				SMLAxy(cond) Rd. Rm.	16 * 16 bit and accumulate 5E
				SMULWy{cond} Rd,	32 * 16 bit 5E
				E SMULxy{cond} Rd, Rm, Rs	16 * 16 bit 5E
	*	ů		SMLAL(cond) [S] RdLo, RdHi, Rm, Rs	and accumulate long M
	*	ů	Z	SMULL(cond){S} RdLo, RdHi, Rm, Rs	Signed multiply long M
				UMAAL(cond) RdLo, RdHi, Rm, Rs	unsigned double accumulate long 6
	*	Č		UMLAL(cond){S} RdLo, RdH1, Rm, Rs	
				omulti(cond)(s) kalo, kali, km, ks	
				Minister (1) (2) - 11.	ξ τ
	•		1 :	and the second of the second o	
-	*	* ?		MUT.{cond}{s} Rd Rm Rs	
0					double saturating 5E
0				E QSUB {cond} Rd, Rm, Rn	saturating 5E
	<			RSC{cond}{S} Rd, Rn, <operand2></operand2>	reverse subtract with carry
	<			Rd, Rn, < Operand2>	reverse subtract
			: 2	au, au, coperantes	with carry
				l bd bn (Oregando)	with carry
,	<			SUB(cond)(S) Rd. Rn. <operand?></operand?>	
0				QDADD(con	double saturating 5E
0				QADD(cond) Rd, Rm, Rr	saturating 5E
	<		N	<operand2></operand2>	with carry
	٧.	2		ADD{cond}(S) Rd, Rn, <operand2></operand2>	Arithmetic Add
٥		ates	S updates	Assembler	Operation
	~	R)	L.	d by <p_mode></p_mode>	R13 for the processor mode specified by <p_mode></p_mode>
8	<iflags></iflags>	Δ.			<pre><p_mode></p_mode></pre>
		w		lel instructions	
	ì	+/-		nt. Omerwise, KSX is KS.	
	~			A 32-bit Collisiant, Jointed by Alghrichamy an o-bit value by all even holliboer of bits.	"IECT OT >
4+28	<regitsc+f< td=""><td>- A</td><td></td><td>meaning joined.</td><td></td></regitsc+f<>	- A		meaning joined.	
4-28	4-2STIBETS	A		rout ofeatet that of Equal 1145s. Atways upuated by parallet ands and subdates.	
	<tegitec></tegitec>	ŕ	-	Sticky 1128; Fundys opuates on overlow (no 3 opuon), kead and teset using MIKS and MISK.	
0	Ca_modeux	6	_	s ing is only concerned in Architecture 17 and carrier, discussing on in Architecture 42 and 1867.	•
י י	ca_mode#ox	6		and and antice makened in Architecture us and leave	V +
, 1	to mode 45.	6	_	Indates condition flore if Concepts	-
	va_mode4T	\ /	_	itiis Remister) or SPSR (Saved Processor Status Remister	
,	1	١ (_ !		
32P>	<a mode2p="">	٥.	įs	Refer to Table Flexible Operand 2. Shift and rotate are only available as part of Operand2	ind2>
12 V	<a mode2="">	ŏ,		Init for unconditional execution.	(cond) Refer to Table Condition Field. Omit for unconditional execution
mes	endiannes	<u>-</u>	┙		Key to Tables

Multiply with internal 40-bit accumulate

Count leading zeroes

half word packed halfword

Atc XS MIA(cond) Ac, Rm, Rs
XS MIAPH(cond) Ac, Rm, Rs
XS MIAXY(cond) Ac, Rm, Rs
5 CLZ(cond) Rd, Rm

Rd := Rn + (Rm * Rs)[63:32] Rd := Rn - (Rm * Rs)[63:32]

Rd := (Rm * Rs)[63:32]

Ac := Ac + Rm * Rs

Ac := Ac + Rm[15:0] * Rs[15:0] + Rm[31:16] * Rs[31:16]

Ac := Ac + Rm[x] * Rs[y]

Rd := Rn + Rm[15:0] * RsX[15:0] - Rm[31:16] * RsX[31:16]

|RdHi,RdLo := RdHi,RdLo + Rm[15:0] * RsX[15:0] - Rm[31:16] * RsX[31:16]

Rd := Rm[15:0] * RsX[15:0] - Rm[31:16] * RsX[31:16]

RdHi,RdLo := RdHi,RdLo + Rm[15:0] * RsX[15:0] + Rm[31:16] * RsX[31:16]

Rd := Rn + (Rm * Rs[y])[47:16]

RdHi,RdLo := RdHi,RdLo + Rm[x] * Rs[y]

Rd := Rn + Rm[x] * Rs[y]Rd := (Rm * Rs[y])[47:16]

Rd := Rm[x] * Rs[y]

RdHi,RdLo := signed(RdHi,RdLo + Rm * Rs)

RdHi,RdLo := signed(Rm * Rs)

RdHi,RdLo := unsigned(RdHi,RdLo + Rm * Rs)
RdHi,RdLo := unsigned(RdHi + RdLo + Rm * Rs)

Rd := 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]

	{endianness}	Can be BE (Big Endian) or LE (Little Endian).
ion.	<a_mode2></a_mode2>	Refer to Table Addressing Mode 2.
vailable as part 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.
Processor Status 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).
Architecture v5 and later.	<a_mode5></a_mode5>	Refer to Table Addressing Mode 5.
reset using MRS and MSR.	<reglist></reglist>	A comma-separated list of registers, enclosed in braces { and }.
s and subtracts.	<reglist-pc></reglist-pc>	As <reglist>, must not include the PC.</reglist>
	<reglist+pc></reglist+pc>	As <reglist>, including the PC.</reglist>
ven number of bits.	{:}	Updates base register after data transfer if ! present.
	+/-	+ or (+ may be omitted.)
	son	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.

Action

Rd := SAT(Rm + SAT(Rn * 2))

Rd := Rn + Operand2 + Carry

Rd := Rn - Operand2 Rd := SAT(Rm + Rn)Rd := Rn + Operand2

Rd := SAT(Rm - SAT(Rn * 2))

Rd := ((Rm * Rs) + Rn)[31:0]Rd := (Rm * Rs)[31:0]

RdHi,RdLo := unsigned(Rm * Rs)

Rd := SAT(Rm - Rn)

Rd := Operand2 - Rn - NOT(Carry) Rd := Rn - Operand2 - NOT(Carry)

Rd := Operand2 - Rn

ARM Addressing Modes Quick Reference Card

			S updates Q GE Action	읎	
If word - wice addition					
THE POLICE HERO GROUNDS	^	<pre><pre><pre><pre><pre><pre><pre><pre></pre></pre></pre></pre></pre></pre></pre></pre>		<u>유</u>	GE $[Rd[31:16] := Rn[31:16] + Rm[31:16], Rd[15:0] := Rn[15:0] + Rm[15:0]$
arithmetic Halfword-wise subtraction	9	prefix>SUB16{cond} Rd, Rn, Rm		8	GE[Rd[31:16] := Rn[31:16] - Rm[31:16], Rd[15:0] := Rn[15:0] - Rm[15:0]
Byte-wise addition	۷	prefix>ADD8{cond} Rd, Rn, Rm		£	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	0	prefix>SUB8{cond} Rd, Rn, Rm		윤	
Halfword-wise exchange, add, subtract	6	:prefix>ADDSUBX{cond} Rd, Rn, Rm		Œ	E Rd[31:16] := Rn[31:16] + Rm[15:0], Rd[15:0] := Rn[15:0] Rm[31:16]
Halfword-wise exchange, subtract, add	۷	prefix>SUBADDX{cond} Rd, Rn, Rm		GE	E Rd[31:16] := Rn[31:16] - Rm[15:0], Rd[15:0] := Rn[15:0] + Rm[31:16]
Unsigned sum of absolute differences	6 0	ISAD8 {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 U	ISADA8(cond) Rd, Rm, Rs, Rn			Rd := Rn + Abs(Rm[31:24] - Rs[31:24]) + Abs(Rm[23:16] \sim Rs[23:16]) + Abs(Rm[15:8] - Rs[15:8]) + Abs(Rm[7:0] - Rs[7:0])
Move	7		Z	_	Rd := Operand2
NOT	37		2		Rd := 0xFFFFFFFF EOR Operand2
PSR to register	ω Z	IRS{cond} Rd, <psr></psr>			Rd := PSR
register to PSR	3	<u> </u>			PSR := Rm (selected bytes only)
immediate to PSR	ω Z	-		-	PSR := immed_8r (selected bytes only)
40-bit accumulator to register	SX	<u> </u>			RdLo := Ac[31:0], RdHi := Ac[39:32]
gister to 40-bit accumulator	XX.	MAR (cond) Ac, RdLo, RdHi			Ac[31:0] := RdLo, Ac[39:32] := RdHi
Сору	6	:PY{cond} Rd, <operand2></operand2>			Rd := Operand2
Test	1	Rn, <operand2></operand2>	NZC		Update CPSR flags on Rn AND Operand2
Test equivalence	ы		7		Update CPSR flags on Rn EOR Operand2
AND	₩		7		Rd := Rn AND Operand2
EOR	jr:		Z		Rd := Rn EOR Operand2
ORR	0	•	7	_	Rd := Rn OR Operand2
Bit Clear	tri		2		Rd := Rn AND NOT Operand2
Compare	_		NZCV		Update CPSR flags on Rn - Operand2
negative	_		NZCV		Update CPSR flags on Rn + Operand2
Signed saturate word, right shift	6	SAT{cond} Rd, # <sat>, Rm{, ASR <sh>}</sh></sat>		0	Rd := SignedSat((Rm ASR sh), sat). <sat> range 0-31, <sh> range 1-32.</sh></sat>
left shift	ΙΛ	SAT{cond} Rd, # <sat>, Rm{, LSL <sh>}</sh></sat>		0	Rd := SignedSat((Rm LSL sh), sat). < sat > range 0-31, < sh > range 0-31.
Signed saturate two halfwords	6	SAT16(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.
Unsigned saturate word, right shift	0	SAT{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	-	SAT(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	0	SAT16{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>
	add, subtract subtract, add differences differences egister entator shift shift shift	add, subtract 6 subtract, add 6 differences 6 differences 6 xx segister xx subtract add 6 ght shift 6 ght shift 6	6 <pre>prefix>SUB16{cond} Rd, Rn, Rm 6 <pre>cprefix>ADD8{cond} Rd, Rn, Rm 6 <pre>cprefix>ADD8UBX{cond} Rd, Rn, Rm ddd, subtract, add 6 <pre>cprefix>SUBADDSUBX{cond} Rd, Rn, Rm subtract, add 6 <pre>cprefix>SUBADDX{cond} Rd, Rn, Rm differences 6 USADA8{cond} Rd, Rm, Rs, Rn MVV{cond}{S} Rd, <pre>coperand2> MVX{cond}{S} Rd, <pre>coperand2> MXS{cond} Rd, <pre>cpread2> 3 MSR{cond} Rd, <pre>cpread2> 3 MSR{cond} Rd, <pre>cpread2> 3 MSR{cond} Rd, <pre>cpread2> 3 MSR{cond} Rd, <pre>cpread2> 5 MSR{cond} Rd, <pre>cpread2> 6 CYY{cond} Rd, <pre>cpread2> 7 MSR{cond} Rd, <pre>cpread2> 7 MSR{cond} Rd, <pre>cpread2> 7 MSC{cond} Rd, Rn, <pre>coperand2> 7 MSC{cond} Rd, Rn, <pre< th=""><th>6 <pre>cprefix>SUB16{cond} Rd, Rn, Rm 6 <pre>cprefix>ADDB{cond} Rd, Rn, Rm 6 <pre>cprefix>ADDBB{cond} Rd, Rn, Rm 6 <pre>cprefix>BDBB{cond} Rd, Rn, Rm ddd, subtract, add 6 <pre>cprefix>SUBADSUBX{cond} Rd, Rn, Rm subtract, add 6 <pre>cprefix>SUBADDX{cond} Rd, Rn, Rm differences fo USADAB{cond} Rd, Rm, Rs, Rn MOV{cond}{S} Rd, Coperand2> MVM{cond}{S} Rd, Coperand2> MVM{cond}{S} Rd, Coperand2> MSR{cond} Rn, Coperand2> MSR{cond} Rn, Coperand2> MSR{cond} Rn, Coperand2> MSR{cond} Rn, Coperand2> MSR{cond}{S} Rd, Rn, Coperand2> MSR{cond}{S} Rd, Rn, Coperand2> MS C MM{cond}{S} Rd, Rn, Coperand2> MS C MM{cond}{S} Rd, Rn, Coperand2> MS C MM{cond}{Rn, Coperand2> MS C MM{cond}{Rn, Coperand2> MS C MM{cond}{Rn, Coperand2> MS C MS C MS C MM{cond}{Rn, Coperand2> MS C MS C MS C MS C MM{cond}{Rn, Coperand2> MS C MS C MS C MS C MM{cond}{Rn, Coperand2> MS C MS C</pre></pre></pre></pre></pre></pre></th><th>6 <pre>cprefix>SUB16{cond} Rd, Rn, Rm 6 <pre>cprefix>ADD8{cond} Rd, Rn, Rm 6 <pre>cprefix>ADD8(cond) Rd, Rn, Rm differences 6 <pre>cprefix>SUBADDX(cond) Rd, Rn, Rm ubbract, add 6 <pre>cprefix>SUBADDX(cond) Rd, Rn, Rm differences 6 <pre>USADA8{cond} Rd, Rm, Rs, Rn MOV{cond}{S} Rd, Coperand2> MVN{cond}{S} Rd, Coperand2> MVN{cond}{S} Rd, Coperand2> MRS{cond} Cond} Rd, CPSR> MRS{cond} Cond} RdLo, RdHi, Ac mulator XS MRA{cond} Rc, RdLo, RdHi, Ac mulator XS MRA{cond} Rd, Coperand2> TST{cond} Rd, Coperand2> TST{cond} Rd, Coperand2> TST{cond} Rd, Coperand2> N Z C AND{cond}{S} Rd, Rn, Coperand2> N Z C EDR{cond}{S} Rd, Rn, Coperand2> N Z C CMR{cond}{S} Rd, Rn, Coperand2> N Z C CMR{cond}{Rd, *csat>, Rm{, ASR csh>} CMR{cond} Rd, *csat>, Rm{, ASR csh>} USAT{cond} Rd, *csat>, Rm{, ASR csh>} USAT{cond} Rd, *csat>, Rm{, LSL csh>} USAT{cond} Rd, *csat>, Rm{, LSL csh>} USATI{cond} Rd,</pre></pre></pre></pre></pre></pre></th></pre<></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre>	6 <pre>cprefix>SUB16{cond} Rd, Rn, Rm 6 <pre>cprefix>ADDB{cond} Rd, Rn, Rm 6 <pre>cprefix>ADDBB{cond} Rd, Rn, Rm 6 <pre>cprefix>BDBB{cond} Rd, Rn, Rm ddd, subtract, add 6 <pre>cprefix>SUBADSUBX{cond} Rd, Rn, Rm subtract, add 6 <pre>cprefix>SUBADDX{cond} Rd, Rn, Rm differences fo USADAB{cond} Rd, Rm, Rs, Rn MOV{cond}{S} Rd, Coperand2> MVM{cond}{S} Rd, Coperand2> MVM{cond}{S} Rd, Coperand2> MSR{cond} Rn, Coperand2> MSR{cond} Rn, Coperand2> MSR{cond} Rn, Coperand2> MSR{cond} Rn, Coperand2> MSR{cond}{S} Rd, Rn, Coperand2> MSR{cond}{S} Rd, Rn, Coperand2> MS C MM{cond}{S} Rd, Rn, Coperand2> MS C MM{cond}{S} Rd, Rn, Coperand2> MS C MM{cond}{Rn, Coperand2> MS C MM{cond}{Rn, Coperand2> MS C MM{cond}{Rn, Coperand2> MS C MS C MS C MM{cond}{Rn, Coperand2> MS C MS C MS C MS C MM{cond}{Rn, Coperand2> MS C MS C MS C MS C MM{cond}{Rn, Coperand2> MS C MS C</pre></pre></pre></pre></pre></pre>	6 <pre>cprefix>SUB16{cond} Rd, Rn, Rm 6 <pre>cprefix>ADD8{cond} Rd, Rn, Rm 6 <pre>cprefix>ADD8(cond) Rd, Rn, Rm differences 6 <pre>cprefix>SUBADDX(cond) Rd, Rn, Rm ubbract, add 6 <pre>cprefix>SUBADDX(cond) Rd, Rn, Rm differences 6 <pre>USADA8{cond} Rd, Rm, Rs, Rn MOV{cond}{S} Rd, Coperand2> MVN{cond}{S} Rd, Coperand2> MVN{cond}{S} Rd, Coperand2> MRS{cond} Cond} Rd, CPSR> MRS{cond} Cond} RdLo, RdHi, Ac mulator XS MRA{cond} Rc, RdLo, RdHi, Ac mulator XS MRA{cond} Rd, Coperand2> TST{cond} Rd, Coperand2> TST{cond} Rd, Coperand2> TST{cond} Rd, Coperand2> N Z C AND{cond}{S} Rd, Rn, Coperand2> N Z C EDR{cond}{S} Rd, Rn, Coperand2> N Z C CMR{cond}{S} Rd, Rn, Coperand2> N Z C CMR{cond}{Rd, *csat>, Rm{, ASR csh>} CMR{cond} Rd, *csat>, Rm{, ASR csh>} USAT{cond} Rd, *csat>, Rm{, ASR csh>} USAT{cond} Rd, *csat>, Rm{, LSL csh>} USAT{cond} Rd, *csat>, Rm{, LSL csh>} USATI{cond} Rd,</pre></pre></pre></pre></pre></pre>

ARM Instruction Set Quick Reference Card

No Op	Software S interrupt	تر التا				change	SSOF						Dialicit		I		Reverse li bytes	L		Unsigned 1-	with add		E	extend 1	Unsigned I	- E	extend	Signed I-	ים די	1
No operation	Software interrupt	Breakpoint	Store return state	ort cindiallicas	Citange processor mode		Change processor state	and change to Java state	with link and exchange (2)		with link and exchange (1)	with link	branch	Select bytes	In low halfword, sign extend	In both halfwords	In word	Byte to word, add	Two bytes to halfwords, add	Halfword to word, add	Two bytes to halfwords, add	Halfword to word, add	Byte to word	Two bytes to halfwords	Halfword to word	Byte to word	Two bytes to halfwords	Halfword to word	Pack halfword top + bottom	Pack halfword hollom 1 fon
5 18	(0	5 0			y C			5J, 6 E	2		C,14		ı tr	5	9	6	я 0	6 1	6			3 9	6 T	6		6 5	6	_	6 6	4
don	SWI{cond} <immed_24></immed_24>	<pre>RFE<a_mode4l> Rn{!} BKPT <immed_16></immed_16></a_mode4l></pre>	SRS <a_mode4s> #<p_mode>{!}</p_mode></a_mode4s>	ODIDAN SEHUTUHANNA	CES #cp_modes	CPSIE <1IIags> {, # <p_mode>}</p_mode>	·	BXJ(cond) Rm	BLX{cond} Rm		BX(cond) km	BL(cond) Label	B{CONG} Label	SEL{cond} Rd, Rn, Rm	REVSH{cond} Rd, Rm	REV16{cond} Rd, Rm	REV{cond} Rd, Rm	UXTAB{cond} Rd, Rn, Rm{, ROR # <sh>}</sh>	UXTAB16{cond} Rd, Rm, Rm{, ROR # <sh>}</sh>	UXTAH (cond) Rd. Rn. Rm(, KOR # <sh>)</sh>	, _C ,	SXTAH(cond) Rd, Rn, Rm(, ROR # <sh>)</sh>	UXTB(cond) Rd, Rm{, ROR # <sh>}</sh>	ı>}	UXTH(cond) Rd, Rm{, ROR # <sh>}</sh>	SXTB(cond) Rd, Rm(, ROR # <sh>)</sh>	SXTB16{cond} Rd, Rm{, ROR # <sh>}</sh>	Rm{, ROR #	PKHTB{cond} Rd, Rn, Rm{, ASR # <sh>}</sh>	not on one ter
None	Software interrupt processor exception.	PC := [kn], CPSR := [kn + 4] Prefetch abort or enter debug state.	CPSR	Sets entrancess for loads and saves. <endianness (big="" (little="" be="" can="" endian)="" endian).<="" le="" or="" p=""></endianness>		Enable specified interrups, optional change mode.	Disable specified interrups, optional change mode.	Change to Java state	R14 := address of next instruction, R15 := Rm[31:1] Change to Thumb if Rm[0] is 1	Q	R14 := address of next instruction R15 := label Change to Thumb	R14 := address of next instruction, R15 := label	KID = label	Rd[7:0] := Rn[7:0] if GE[0] = 1, else Rd[7:0] := Rm[7:0] Bis[15:8], [23:16], [31:24] selected similarly by GE[1], GE[2], GE[3]	Rd[15:8] := Rm[7:0], Rd[7:0] := Rm[15:8], Rd[31:16] := Rm[7] * &:FFFF	Rd[15:8] := Rm[7:0], Rd[7:0] := Rm[15:8], Rd[31:24] := Rm[23:16], Rd[23:16] := Rm[31:24]	Rd[31:24] := Rm[7:0], Rd[23:16] := Rm[15:8], Rd[15:8] := Rm[23:16], Rd[7:0] := Rm[31:24]	Rd[31:0] := Rn[31:0] + ZeroExtend((Rm ROR (8 * sh))[7:0]). sh 0-3.		Rd[31:0] := Rn[31:0] + SignExtend((Rm ROR (8 * sh))[7:0]), sh 0-3.		Rd[31:0] := Rn[31:0] + SignExtend((Rm ROR (8 * sh))[15:0]), sh 0-3.	Rd[31:0] := ZeroExtend((Rm ROR (8 * sh))[7:0]). sh 0-3.	Rd[31:16] := ZeroExtend((Rm ROR (8 * sh))[23:16]), Rd[15:0] := ZeroExtend((Rm ROR (8 * sh))[7:0]), sh 0-3.	Rd[31:0] := ZeroExtend((Rm ROR (8 * sh))[15:0]). sh 0-3.	Rd[31:0] := SignExtend((Rm ROR (8 * sh))[7:0]). sh 0-3.		Rd[31:0] := SignExtend((Rm ROR (8 * sh))[15:0]). sh 0-3.	Rd[31:16] := Rn[31:16], Rd[15:0] := (Rm ASR sh)[15:0], sh 1-32.	
	24-bit value encoded in instruction.	Cannot be conditional.	Cannot be conditional.	Cannot be conditional.	Cannot be conditional.	Cannot be conditional.	Cannot be conditional.			label must be within ±32Mb of current instruction.	Cannot be conditional	label must be within ±32Mb of current instruction.	of current instruction.				i													10000

ARM Addressing Modes Quick Reference Card

Operation		S	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)			R15 := [address][31:1] (8 5T: Change to Thumb if [address][0] is 1)	
	Byte		LDR{cond}B Rd, <a_mode2></a_mode2>		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*	5E* 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>	Load list of registers from [Rn]	
	return (and exchange)		<reglist+pc></reglist+pc>	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>	Load registers, branch (§ 5T: and exchange), CPSR := SPSR	Use from exception modes only.
	User mode registers				
Soft preload	Memory system hint	£*			Cannot be conditional.
Load exclusive	Load exclusive Semaphore operation	6	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	
	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*	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		STM(cond) <a_mode4s> Rn(!), <reglist>^</reglist></a_mode4s>	Store list of User mode registers to [Rn]	Use from privileged modes only.
Store exclusive	Store exclusive Semaphore operation	6			Rd, Rm, Rn must not be R15.
Swap	Word	3	SWP{cond} Rd, Rm, [Rn]	temp := [Rn], [Rn] := Rm, Rd := temp	
	Byte	w	SWP{cond}B Rd, Rm, [Rn]	temp := ZeroExtend([Ra][7:0]), [Rn][7:0] := Rm[7:0] Rd := temp	
				E LE CATALON CONTRACTOR CONTRACTO	

Page 9 of 11

ARM Addressing Modes Quick Reference Card

Addressing	Mode 2 - Word and I	Jnsige	Addressing Mode 2 - Word and Unsigned Byte Data Transfer		ARM architec
Pre-indexed	Immediate offset	[Rn,	[Rn, #+/- <immed_12>]{1}</immed_12>		n
	Zero offset	配		Equivalent to [Rn,#0]	nT, nJ
	Register offset	[Rn,	[Rn, +/-Rm]{!}		×
	Scaled register offset	[Rn,	Scaled register offset [Rn, +/-Rm, LSL # <shift>] {1} Allowed shifts 0-31</shift>	Allowed shifts 0-31	пE
		[Rn,	$[Rn, +/-Rm, LSR \#] {!} Allowed shifts 1-32$	Allowed shifts 1-32	ηE*
		[Rn,	$[Rn, +/-Rm, ASR \#]{1} Allowed shifts 1-32$	Allowed shifts 1-32	XS
		Rn,	$[Rn, +/-Rm, ROR \#]{!}$ Allowed shifts 1-31	Allowed shifts 1-31	
		Rn,	[Rn, +/-Rm, RRX] {!}		Flexible Oper
Post-indexed	Immediate offset	[Rn]	[Rn], #+/- <immed_12></immed_12>		Immediate valu
	Register offset	[RE	[Rn], +/-Rm		Logical shift lcf
	Scaled register offset	[Rn],	+/-Rm, LSL # <shift></shift>	Allowed shifts 0-31	Logical shift rig
		[Rn]	[Rn], +/-Rm, LSR # <shift></shift>	Allowed shifts 1-32	Arithmetic shift
		配	[Rn], +/-Rm, ASR # <shift></shift>	Allowed shifts 1-32	Rotate right imr
		[RE	[Rn], +/-Rm, ROR # <shift></shift>	Allowed shifts 1-31	Register
		[Rn]	[Rn], +/-Rm, RRX		Rotate right exte
					Logical shift lef

Billecanna	Audiceanis inoue z (Poetinekeu om)	(Killy)				
Post-indexed	Post-indexed Immediate offset	[Rn],	[Rn], #+/- <immed_12></immed_12>	mmed_	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>LSI:</td><td>#<shift></shift></td><td>Allowed shifts 0-31</td></shift:<>	[Rn],	+/-Rm,	LSI:	# <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 :	+/-Rm, ROR # <shift></shift>	Allowed shifts 1-31
		[Rn],	[Rn], +/-Rm, RRX	RRX		

	Zero offset	[Rn]				Equivalent to [Rn],#0
	Register offset	[Rm], +	+/-Rm			
	Scaled register offset [Rn] , +/-Rm, LSL # <shift></shift>	[Rn], +	/-Rm,	TSI	# <shift></shift>	Allowed shifts 0-31
		[Rn] , +	/-Rm,	LSR	[Rn] , +/-Rm, LSR # <shift></shift>	Allowed shifts 1-32
		[Rn], +	/-Rm,	ASR 1	+/-Rm, ASR # <shift></shift>	Allowed shifts 1-32
		[Rn], +	/-Rm,	ROR i	[Rn], +/-Rm, ROR # <shift></shift>	Allowed shifts 1-31
		[Rn], +/-Rm, RRX	/-Rm,	RRX		
Addressing	Addressing Mode 3 - Halfword, Signed Byte, and Doubleword Data Transfer	Signed By	te, and	Doub	leword Data Ti	ransfer
Pre-indexed	Immediate offset	[Rn, #+/- <immed_8>]{!}</immed_8>	/- <imn< td=""><td>ed_8:</td><td><1 } [<</td><td></td></imn<>	ed_8:	<1 } [<	
	Zero offset	[Rn]				Equivalent to [Rn,#0]
	Register	[Rn, +/-Rm]{!}	-Rm] { !	سيدا		
Post-indexed	Immediate offset	[Rn], #+/- <immed_8></immed_8>	+/- <in< td=""><td>med :</td><td>8></td><td></td></in<>	med :	8>	
	Register	[Rn], +/-Rm	/-Rm			

Block load Increment After FD In Increment Before ED	IB ·		DA]		ock stare		ock stare	ock store
Increment After Increment Before	Increment Before	,	Decrement Arter	Decrement Before	Decrement Before	Decrement Before Increment After	Decrement Before Increment After Increment Before	Decrement After Increment After Increment After Decrement After
Stack pop	ED		FA	FA EA	FA EA Stack push	FA EA Stack push	FA EA Stack push EA	FA EA Stack push FA ED
Full Descending Empty Descendi	Empty Descending		Full Ascending	Full Ascending Empty Ascending	Full Ascending Empty Ascending	Full Ascending Empty Ascending Empty Ascending	Empty Ascending Empty Ascending Empty Ascending Full Ascending	Empty Ascending Empty Ascending Empty Ascending Empty Ascending Full Ascending Empty Descending

Pre-indexed

Post-indexed

Immediate offset Immediate offset

Zero offset

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

Equivalent to [Rn.#0]

(8-bit capro. option)

Addressing Mode 5 - Coprocessor Data Transfer

							indexed	ressing								
				Scaled register offset	Register offset	Zero offset	indexed Immediate offset	ressing Mode 2 (Post-indexed only)						Scaled register offset	Register offset	
[Rn],	[Rn],	[Rn],	[Rn],	[Rn],	[Rn],	[Rn]	[Rn],	d only		[Rn],	[Rn],	[Rn],	[Rn],	[Rn],	[Rm],	
[Rn], +/-Rm, RRX	+/-Rm,	+/-Rm,	+/-Rm,	+/-Rm,	+/-Rm		[Rn], #+/- <immed_12></immed_12>			[Rn], +/-Rm, RRX	+/-Rm,	+/-Rm,	+/-Rm,	+/-Rm,	[Rn], +/-Rm	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
RRX	ROR	ASR	LSR	$_{\rm LSL}$			med			RRX	ROR	ASR	LSR	LSL		ľ
	[Rn], +/-Rm, ROR # <shift></shift>	[Rn], +/-Rm, ASR # <shift></shift>	+/-Rm, LSR # <shift></shift>	[Rn], +/-Rm, LSL # <shift></shift>			12>				[Rn], +/-Rm, ROR # <shift></shift>	[Rn], +/-Rm, ASR # <shift></shift>	[Rn], +/-Rm, LSR # <shift></shift>	[Rn], +/-Rm, LSL # <shift></shift>		
	Allowed shifts 1-31	Allowed shifts 1-32	Allowed shifts 1-32	Allowed shifts 0-31		Equivalent to [Rn],#0					Allowed shifts 1-31	Allowed shifts 1-32	Allowed shifts 1-32	Allowed shifts 0-31		
ທ	Ħ	C	Suffix	PSR fields		Rotate right register	Arithmetic shift right register	Logical shift right register	Logical shift left register	Rotate right extended	Register	Rotate right immediate	Arithmetic shift right immediate	Logical shift right immediate	Logical shift left immediate	The second second
Status field mas	Flags field masl	Control field m	Meaning	(use at least one		ster	right register	n register	register	nded		lediate	right immediate	ıt immediate	immediate	

1-32	Suffix	Meaning	
1-32	n	Control field mask byte	PSR[7:0]
1-31	H	Flags field mask byte	PSR[31:24]
	ហ	Status field mask byte	PSR[23:16]
	×	Extension field mask byte	PSR[15:8]

	(mac at reast one serves)	
Suffix	Meaning	
c	Control field mask byte	PSR[7:0]
Ħ	Flags field mask byte	PSR[31:24]
ທ	Status field mask byte	PSR[23:16]
×	Extension field mask byte	PSR[15:8]

Mnemonic	Description	Description (VFP)
ŎŒ	Equal	Equal
NE	Not equal	Not equal, or unordered
CS / HS	Carry Set / Unsigned higher or same	Greater than or equal, or unordered
cc / 10	Carry Clear / Unsigned lower	Less than
IW	Negative	Less than
Τđ	Positive or zero	Greater than or equal, or unordered
SΛ	Overflow	Unordered (at least one NaN operand)
٧c	No overflow	Not unordered
Ħ	Unsigned higher	Greater than, or unordered
SŢ	Unsigned lower or same	Less than or equal
æ	Signed greater than or equal	Greater than or equal
LT	Signed less than	Less than, or unordered
GI	Signed greater than	Greater than
31	Signed less than or equal	Less than or equal, or unordered
ΑL	Always (normally omitted)	Always (normally omitted)

System	31	
Undefined	27	
Abort	23	
Supervisor	19	
IRQ Interrupt	18	
FIQ Fast Interrupt	17	
User	16	
des	Processor Modes	

	İ	
	Prefi	Prefixes for Parallel Instructions
	ß	Signed arithmetic modulo 28 or 216, sets CPSR GE bits
Interrupt	Ø	Signed saturating arithmetic
T. T.	HS	Signed arithmetic, halving results
- -	ᆸ	Unsigned arithmetic modulo 28 or 216, sets CPSR GE bits
	ď	Unsigned saturating arithmetic
	H	Unsigned arithmetic, halving results

ARM architecture versions	ture versions				
п	ARM architecture version n and above	ır ttot	and ab	love.	
nT, nJ	T or J variants of ARM architecture version n and above	larchi	tectun	e version n and abov	Æ.
Z	ARM architecture version 3M, and 4 and above, except xM variants	ion 3N	A, and	4 and above, except	xM variants.
πE	All E variants of ARM architecture version n and above	archi	ecture	version n and above	ņ
Æ*	E variants of ARM architecture version n and above, except xP variants.	hitecu	Jre vei	rsion n and above, ex	xcept xP variants.
XS	XScale coprocessor instruction	tructi	on		
Elevible Operand a	3				
Immediate value		#<1:	# <immed 8r=""></immed>	8r>	
Logical shift left immediate	immediate	Rш,	ISI	LSL # <shift></shift>	Allowed shifts 0-31
Logical shift right immediate	nt 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	lediate	Rm,	ROR	ROR # <shift></shift>	Allowed shifts 1-31
Register		Ř'n			
Rotate right extended	nded	Rm,	RRX		
Logical shift left register	register	Rп,	ISI	Rs	
Logical shift right register	nt register	Rш,	LSR	Rs	
Arithmetic shift right register	right register	Rm,	ASR	Rs	
Rotate right register	ster	Rm,	ROR	Rs	

ARM Addressing Modes Quick Reference Card

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	<pre>2 MRC(cond) <copr>, <opl>, Rd, CRn, CRm(, <opl>)</opl></opl></copr></pre>	Coprocessor dependent	
Alternative move	5 MRC2 <copr>, <op1>, Rd, CRn, CRm{, <op2>}</op2></op1></copr>		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>, <opl>, Rd, Rn, CRm</opl></copr>		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>, <opl>, Rd, CRn, CRm{, <op2>}</op2></opl></copr>	Coprocessor dependent	Cannot be conditional.
Two ARM register move	5E* MCRR(cond) <copr>, <opl>, Rd, Rn, CRm</opl></copr>	Coprocessor dependent	
Alternative two ARM register move	6 MCRR2 <copr>, <op1>, Rd, Kn, 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></a_mode5></copr>	Coprocessor dependent	Cannot be conditional.

Proprietary Notice

Words and logos marked with $^{\Theta}$ or $^{\rm m}$ 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

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 finess for purpose, are excluded.

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

Document Number

ARM QRC 0001H

Change	Log		
issue	Date	Ву	Change
>	June 1995	BJH	First Release
В	Sept 1996	BJH	Second Release
C	Nov 1998	BJH	Third Release
D	Oct 1999	CKS	Fourth Release
tπ	Oct 2000	CKS	Fifth Release
, LI	Sept 2001	CKS	Sixth Release
G	Jan 2003	CKS	Seventh Releas
H	Oct 2003	CKS	Eighth Release