

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

Integrated Computer Science Year 2 Annual Examinations Trinity Term 2018

Microprocessor Systems

Saturday 12 May 2018

RDS Main Hall

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.

- 1. (a) Explain the term "Memory Hierarchy". Give examples of some typical components of the memory hierarchy of a typical computer system and list their principal properties. [4 marks]
 - (b) What does the term Memory Mapped Input/Output mean? [2 marks]
 - (c) Give an account of the purpose, organisation and operation of a typical cache. Explain how memory locations might be mapped to it and explain how it flushes entries. [6 marks]
 - (d) Explain the operation of a typical three-stage pipelined processor such as the ARM processor you have been using in laboratories. [4 marks]
 - (e) Explain how branch or jump instructions can disrupt the operation of a processor's pipeline. What can be done to minimise this problem? [4 marks]
- 2. (a) In the context of a typical Von-Neumann computer architecture, what does an *interface* do? [2 marks]
 - (b) What is the principal difference between a normal interface and a *Direct Memory Access (DMA)* interface? [2 marks]
 - (c) Write a subroutine to read the state of four switches each of which is connected to one bit of an eight-bit parallel interface whose address is equated to the label KEYS. The switches are connected to bits 0, 1, 2 and 3 respectively. The subroutine should return when a key has been pressed and it should return the bit number of the switch being pressed in RO. If more than one switch is pressed, the subroutine should return the value -1 in RO. Normally, when a switch is not pressed, the value of its corresponding bit is 1, and when it is pressed, the value is 0. [8 marks]
 - (d) Give a detailed outline, or write an extra subroutine to display the output of the above subroutine – a value between 0 and 3 in RO – on a seven-segment display connected to an interface whose address is equated to the label DISPLAY. [8 marks]

- 3. (a) Given an interrupt that occurs at 500 microsecond intervals, write an interrupt handler that generates a square-wave output by setting an interface bit high for a specific number of 500 microsecond periods and then sets the interface bit low for the same number of periods before repeating the process. The interface is at a location whose address is equated to the label GEN and the relevant bit is bit 0. [10 marks]
 - (b) What is meant by the "context" of a program? [2 marks]
 - (c) Explain how a very simple two-thread scheduler can be built around a timed interrupt handler. Give a detailed description of the operation of the interrupt handler, including an explanation of all the data areas needed and how they are used. Pay particular attention to describing how the context of the programs that are being scheduled is managed. [8 marks]

ASCII Code

| | Colun | nn Numl | per | | | | | |
|---------------|-------|---------|------------|-----|-----|-----|-----|-----------|
| 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 | Н | 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 | 1 | ? | 0 | | 0 | $D\!E\!L$ |

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

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

and 0 of the ASCII.

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

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

The Space is ASCII 0100000, and is shown as \Diamond here.

| Key to Tables | | ; | _ | ᅴ | endi | ani | {endiamess} | Can be BE (Big Endian) or LE (Little Endian). |
|------------------------------|--|---|--------------|----------------|-----------------------|----------|---------------------------|---|
| (cond) | Refer to Table Condition Field | Refer to Table Condition Field. Omit for unconditional execution. | | Δ. | <a_mode2></a_mode2> | ide2 | Ÿ | |
| <operand2></operand2> | Refer to Table Flexible Opera | Refer to Table Flexible Operand 2. Shift and rotate are only available as part of Operand2. | ij | ۸ | <a_mode2p></a_mode2p> |)de2 | ۸Ą | Refer to Table Addressing Mode 2 (Post-indexed only). |
| <psr></psr> | Either CPSR (Current Processor | Either CPSR (Current Processor Status Register) or SPSR (Saved Processor Status Register) | <u>ء</u> | <u> </u> | <a mode4l=""> | de de de | FV | Refer to Table Addressing Mode 4 (Block load or Stack pop). |
| {s} | Updates condition flags if S present. | 知. | | Δ : | <a_mode45></a_mode45> | de4 | į, | Refer to Table Addressing Mode 4 (Block store or Stack push). |
| C*, V* | Flag is unpredictable in Architec | Flag is unpredictable in Architecture v4 and earlier, unchanged in Architecture v5 and later. | _ | Δ | <a_mode5></a_mode5> |)de5 | Ϋ́ | Refer to Table Addressing Mode 5. |
| ٥ | Sticky flag. Always updates on o | Sticky flag. Always updates on overflow (no S option). Read and reset using MRS and MSR | = | Λ | <reglist></reglist> | ist | ٧ | A comma-separated list of registers, enclosed in braces { and }. |
| GE C | Four Greater than or Equal flags. | Four Greater than or Equal flags. Always updated by parallel adds and subtracts. | | _ | regl | 191 | <reglist-pc></reglist-pc> | As <reglist>, must not include the PC.</reglist> |
| <immed 8r=""></immed> | A 32-bit constant, formed by right-rotating an 8-bit va | is meaning nair-register [15:0], or a meaning [51:16]. A 32-bit constant, formed by right-rotating an 8-bit value by an even number of bits. | | - ^ | {!} | 180 | <reglist+pc></reglist+pc> | As <reglist>, including the PC. Undates base register after data transfer if ! present.</reglist> |
| { <u>x</u> } | RsX is Rs rotated 16 bits if X present. Otherwise, RsX is Rs | sent. Otherwise, RsX is Rs. | | + | + | | | + or (+ may be omitted.) |
| <pre><prefix></prefix></pre> | Refer to Table Prefixes for Parallel instructions | allel instructions | | 605 | | | | Refer to Table ARM architecture versions. |
| <p_mode></p_mode> | Refer to Table Processor Modes | es | | A | <iflags></iflags> | ce51 | ٠ | Interrupt flags. One or more of a, i, f (abort, interrupt, fast interrupt). |
| R13m | R13 for the processor mode specified by <p_mode></p_mode> | lied by <p_mode></p_mode> | L | | R | | | Rounds result to nearest if R present, otherwise truncates result. |
| Operation | | § Assembler | Supdates | date | " | ٥ | Action | מת |
| Arithmetic Add | • | ADD(cond)(S) Rd, Rn, <operand2></operand2> | z | 7. | C | _ | Rd := | Rd := Rn + Operand2 |
| S 2 | with carry | ADC(cond) (S) Rd, Rn, <operand2></operand2> | | | | | | Rd ≔ Rn + Operand2 + Carry |
| do do | double saturating | | | | | _ | Q Rd : | Rd := SAT(Rm + SAT(Rn * 2)) |
| Subtract | . g | SUB{cond}{S} Rd, Rn, <operand2></operand2> | | | | | | Rd := Rn - Operand2 |
| | With carry | SBC(cond)(s) Rd, Rn, <operand2></operand2> | | | | _ | | Rd := Rn - Operand2 - NOT(Carry) |
| 6.5 | reverse subtract with carry | RSC(cond)(S) Rd, Rn, <operand2></operand2> | Z 2 | | ი ი | < < | Z 2 | Rd := Operand2 = Rn = NOT(Carry) |
| Sal | saturating | 5E QSUB(cond) Rd, Rm, Rn | | | | _ | | Rd := SAT(Rm - Rn) |
| doub | double saturating | SE QDSUB(cond) Rd, Rm, Rn | | | í | _ | Q Rd:: | Rd := SAT(Rm - SAT(Rn * 2)) |
| | and accumulate | 2 MLA(cond)(S) Rd, Rm, Rs, Rn | | 0 0 | က္ (| | Rd :: | Rd := ((Rm * Rs) + Rn)[3]:0] |
| | unsigned long | | z | | | * | RdHi | RdHi,RdLo := unsigned(Rm * Rs) |
| | unsigned accumulate long | M UMLAL (cond) (S) RdLo, RdHi, Rm, Rs | | | ۸ ű | <u> </u> | RdHi | RdHi,RdLo := unsigned(RdHi,RdLo + Rm * Rs) |
| Signo | Signed multiply long | M SMULL {cond} {S} RdLo, RdHi, Rm, Rs | | | Ç. | ¥ | RdHi | RdHi,RdLo := signed(Rm * Rs) |
| | and accumulate long | | z | 2 | | * | RdHi | RdHi,RdLo := signed(RdHi,RdLo + Rm * Rs) |
| 3 10 | 33 * 16 bit | 5E SMULxy{cond} Rd, Rm, Rs | | | | | Rd ;= | Rd := Rm[x] * Rs[y] |
| 16 | 16 * 16 bit and accumulate | 5E SMLAxy{cond} Rd, Rm, Rs, Rn | | | | | | Kd := Rn + Rm[x] * Rs[v] Rd := Rn + Rm[x] * Rs[v] |
| 32 | 32 * 16 bit and accumulate | SMLAWy [cond] Rd. | | | | _ | Q Rd:= | Rd := Rn + (Rm * Rs[y])[47:16] |
| | 16 * 16 bit and accumulate long | | | | | | RdHi | $RdHi_{R}dLo := RdHi_{R}dLo + Rm[x] * Rs[y]$ |
| an an | and accumulate | 6 SMLAD(X)(cond) Rd, Rm, Rs, Rn | | | | | 2 2 2 2 1 1 | Rd := Rm + Rmf(5:0) * RsXf(15:0) + Rmf(3):16) * RsXf(3):16) Rd := Rn + Rmf(5:0) * RsXf(5:0) + Rmf(3):16) * RsXf(3):16) |
| an | and accumulate long | 6 SMLALD{X}{cond} RdHi, RdLo, Rm, Rs | | | | _ | | RdHi,RdLo:=RdHi,RdLo+Rm[15:0]*RsX[15:0]+Rm[31:16]*RsX[31:16] |
| Dual s | Dual signed multiply, subtract | | | | | _ | | Rd := Rm[15:0] * RsX[15:0] - Rm[31:16] * RsX[31:16] |
| 2 1 | and accumulate long | 6 SMLSD(X){cond} RdH: RdLo Rm Re | | | | | | Rd := Rn + Rm[15:0] * RsX[15:0] - Rm[31:16] * RsX[31:16] RdH; RdF o RdH; RdF o + RmF (5:0) * RsXF15:01 - RmF31:161 * RsXF3:161 |
| Signer | Signed most significant word multiply | SMMUL(R)(cond) Rd, Rm, Rs | | | | | | Rd := (Rm * Rs)[63:32] |
| an | and accumulate | | | | | | Rd := | Rd := Rn + (Rm * Rs)[63:32] |
| an | and subtract | 6 SMMLS(R)(cond) Rd, Rm, Rs, Rn | | | | | Rd := | Rd := Rn (Rm * Rs)[63:32] |
| luma | multiply with internal 40-bit accumulate | XS MIA(cond) Ac, Rm, Rs | | | | | À. | |
| ha | halfword | XS MIAXY (cond) AC, Rm. Rs | | | | | A A | := Ac + Rm[r3] * Rs[r3] + km[31:16] * ks[31:16] := Ac + Rm[r3] * Rs[r3] |
| Count | Count leading zeroes | CLZ(cond) Rd, Rm | | | | | R ? | Rd := number of leading zeroes in Rm |
| | | | | | | ŀ | } | 9 |

| Halfword-wise subtraction Byte-wise subtraction Byte-wise subtraction Halfword-wise exchang Halfword-wise exchang Unsigned sum of absolt and accumulate Move | arithmetic Halfword-wise subtraction Byte-wise addition Byte-wise subtraction Halfword-wise exchange, : Halfword-wise subtraction and accumulate Move NOT PSR to register to PSR immediate to PSR im | arithmetic Hallword-wise additi Byte-wise additi Byte-wise subtra Hallword-wise e Hallword-wis | arithmetic Halfword-wise saddiiti Byte-wise addiiti Byte-wise subtrav Halfword-wise es Halfword-wise subtrav Hove Move NOT PSR to register to PS immediate to 40-bit acoum register to 40 Copy Logical Test Test Tost equivalence AND LOGICAL TEST | arithmetic Halfword-wise additi Byte-wise subtra Halfword-wise subtra Halfword-wise subtra Halfword-wise subtra Halfword-wise sum o and accumul Move Move NOT PSR to register to Psi immediate to 40-bit accum register to 40-bit accum register to 40-bit accum Test Copy Logical Test ORR ORR Bit Clear | arithmetic Halfword-wise additi Byte-wise subtrate Halfword-wise of Halfword-wise of Halfword-wise of Halfword-wise of Halfword-wise of Unsigned sum of and accumulate Move NOT STR to register to Primmediate of 40-bit accumung register to 40-bit accumung | | | 6 |
|--|--|---|---|---|---|---|--|---|
| Byte-wise subtraction Byte-wise subtraction Halfword-wise exchange, add, subtract Halfword-wise exchange, subtract, add Unsigned sum of absolute differences and accumulate Nove | wise subtraction ord-wise exchange, add, subtract ord-wise exchange, subtract, add ned sum of absolute differences of accumulate OT SR to register gister to PSR mucdiate to PSR bit accumulator to register gister to 40-bit accumulator | wise subtraction wise subtraction wise exchange, add, subtract rord-wise exchange, subtract, add ned sum of absolute differences and accumulate OT OT SR to register gister to PSR mediate to PSR bbit accumulator to register gister to 40-bit accumulator | wise subtraction wise subtraction rord-wise exchange, add, subtract rord-wise exchange, subtract, add ned sum of absolute differences ad accumulate OT OR Rk to register gister to PSR nuncdiate to PSR horication to register gister to 40-bit accumulator gister to 40-bit accumulator gister to 40-bit accumulator | wise scounter wise subtraction wise exchange, add, subtract ord-wise exchange, subtract, add ned sum of absolute differences and accumulate OT OT SR to register gister to PSR mediate to PSR mediate to PSR Did accumulator to register gister to 40-bit accumulator gister to 40-bit accumulator gister to 40-bit accumulator | e subtraction e subtraction l-wise exchange, add, subtract l-wise exchange, subtract, add I sum of absolute differences ccumulate to register to register to to PSR tiaccumulator to register ter to 40-bit accumulator valence | e subtraction e subtraction e subtraction l-wise exchange, add, subtract l-wise exchange, subtract, add sum of absolute differences eccumulate to register ter to PSR diate to PSR diate to PSR taccumulator to register ter to 40-bit accumulator valence valence We We | e subtraction e subtraction e subtraction l-wise exchange, add, subtract l-wise exchange, subtract, add sum of absolute differences ccumulate to register ter to PSR diate to PSR diate to PSR caccumulator to register ter to 40-bit accumulator valence valence valence ive | e subtraction e subtraction -twise exchange, add, subtract -twise exchange, subtract, add sum of absolute differences ccumulate cumulate cr to PSR t accumulator to register ter to 4D-hit accumulator valence valence valence iturate word, right shift hift sturate two halfwords saturate word, right shift |
| 2 2 2 2 2 | 0 % % w w w w w w w | 6 % % w w w a a a a a | 0 X X X X X X X X X X X X X X X X X X X | 0 X X X 3 3 6 6 6 6 6 | 0 X X X X X A A A A A | 6 6 8 8 8 8 8 8 8 8 8 | ccs 6 6 5 XX X | CCS XX XX 3 6 6 6 6 XX X |
| <pre>cprefix>SUBADDX(cond) Rd, Rn, USAD0(cond) Rd, Rm, Rs USADA0(cond) Rd, Rm, Rs, Rn MOV(cond)(8) Rd, <pre>coperand2></pre></pre> | 6 <pre>GPTefix>SUBADDX(cond) Rd, Rm, Rs 6 USADB8(cond) Rd, Rm, Rs, Rn 6 USADB8(cond) Rd, Em, Rs, Rn MOV(cond){S} Rd, <operand2> MVN(cond){S} Rd, <operand2> MVN(cond) Rd, <psr> <fields>, R 3 MSR(cond) <psr> <fields>, R 3 MSR(cond) RdLo, RdHi, Ac XS MAR(cond) RdLo, RdHi, Ac XS MAR(cond) Rd, <operand2> 6 CFY(cond) Rd, <operand2></operand2></operand2></fields></psr></fields></psr></operand2></operand2></pre> | <pre>cprefix>SUBADDX(cond) Rd USADB(cond) Rd, Rm, Rs, USADB(cond) Rd, CODETAN MOV(cond){S} Rd, CODETAN MOV(cond){S} Rd, CODETAN MOV(cond){S} Rd, CODETAN MOV(cond) Rd, CODETAN MSR(cond) RD, CEICLES MSR(cond) RD, RdHi, AC MAR(cond) RdLo, RdHi, AC MAR(cond) Rd, CODETANDO TST(cond) Rd, CODETANDO T</pre> | <pre>cprefix>SUBADDX(cond) Rd, Rm, Rs USADB8(cond) Rd, Rm, Rs, Rn MOV(cond){S} Rd, <operand2> MVM(cond){S} Rd, <operand2> MVM(cond) Rd, ePSR> MSR(cond) Rd, <epsr> MSR(cond) Rd, <epsr> MSR(cond) Rd, <epsr> MSR(cond) Rd, <epsr> SIGNATOR RdHi, Ac MRA(cond) Rd, RdHi, Ac MAR(cond) Rd, RdHi, Ac MAR(cond) Rd, <operand2> TST(cond) Rd, <operand2> TST(cond) Rd, <operand2> TSQ(cond) Rd, Rd, <operand2> TSQ(cond) Rd, Rd, Rn, <operand2> TSQ(cond) Rd, Rn, <operand2> TSQ(cond) Rd, Rd, Rd, Rd, Rd, Rd, Rd, Rd, Rd, Rd,</operand2></operand2></operand2></operand2></operand2></operand2></operand2></operand2></operand2></operand2></operand2></operand2></operand2></operand2></operand2></operand2></operand2></operand2></operand2></operand2></epsr></epsr></epsr></epsr></operand2></operand2></pre> | <pre>cprefix>SUBADDX(cond) Rd, Rn, R USADB(cond) Rd, Rm, Rs, Rn WOV(cond){S} Rd, <operand2> MOV(cond){S} Rd, <operand2> MOV(cond){S} Rd, <operand2> MOV(cond) Rd, <operand2> MSR(cond) <psr>_<fields>, Rm MSR(cond) <psr>_<fields>, #<imm MSR(cond) RdIO, RdHi, Ac MRA(cond) RdIO, RdHi, Ac MRA(cond) Rd, <operand2> TST(cond) Rd, <operand2> TST(cond) Rd, <operand2> TSQ(cond){S} Rd, Rn, <operand2> DOR(cond){S} Rd, Rn, <operand2></operand2></operand2></operand2></operand2></operand2></operand2></operand2></operand2></operand2></operand2></operand2></operand2></imm </fields></psr></fields></psr></operand2></operand2></operand2></operand2></pre> | <pre>cprefixsSUBADDX(cond) Rd, Rn, F USADB{cond} Rd, Rm, Rs, Rn WOV{cond}{S} Rd, COperand2> MVM(cond){S} Rd, COperand2> MVM(cond){S} Rd, COperand2> MRS{cond} Rd, CPSR> MSR(cond) Rd, CPSR> MSR(cond) RdLO, RdHi, Ac MAR(cond) RdLO, RdHi, Ac MAR(cond) Rd, COperand2> TST[cond] Rd, COperand2> TST[cond] Rn, COperand2> DOR(cond){S} Rd, Rn, COperand2> CMP(cond){S} Rd, Rn, COperand2> CMP(cond){S} Rd, Rn, COperand2> CMP(cond){Rn, COperand2> CMP(cond){Rn, COperand2> CMP(cond){Rn, COperand2> CMP(cond){Rn, COperand2> CMP(cond){Rn, COperand2> CMP(cond){Rn, COperand2> CMP(cond){Rn, COperand2> CMP(cond){Rn, COperand2></pre> | <pre>cprefixxSUBADDX(cond) Rd, Rn, Ru USADB8(cond) Rd, Rm, Rs, Rn WOV(cond){S} Rd, Coperand2> WMV(cond){S} Rd, Coperand2> MMV(cond) Rd, Rd, Coperand2> MMX(cond) Rd, Coperand2> MMX(cond) Rd, Coperand2> MMX(cond) RdLO, RdHi, Ac MAR(cond) RdLO, RdHi, Ac MAR(cond) Rd, Coperand2> TST(cond) Rd, Coperand2> TST(cond) Rd, Coperand2> DRA(cond){S} Rd, Rn, Coperand2> CMM(cond){Rd, Coperand2> CMM(cond){Rd, Coperand2> CMM(cond){Rd, Rn, Coperand2></pre> | <pre>sprefix>SUBADDX(cond) Rd, Rm, Rm USADB8(cond) Rd, Rm, Rs, Rn WOV(cond){S} Rd, <pre>coperand2> MOV(cond){S} Rd, <pre>coperand2> MNM(cond){S} Rd, <pre>coperand2> MNM(cond){S} Rd, <pre>coperand2> MNM(cond){S} Rd, <pre>coperand2> MNM(cond){CPSR>_fields>, #<immed_bry <pre="" ac="" mar(cond)="" msr(cond)="" rd,="" rdhi,="" rdlo,="">coperand2> TST(cond) Rn, <pre>coperand2> TST(cond){S} Rd, Rn, <pre>coperand2> DCM(cond){S} Rd, Rn, <pre>coperand2> CMM(cond){Rl, <pre>coperand2> CMM(cond) Rd, <pre>coperand2> SSAT(cond) Rd, #<sat>, Rm(, ASR <sh>) SSAT(cond) Rd, #<sat>, Rm(, ASR <sh>) SSAT(cond) Rd, #<sat>, Rm</sat></sh></sat></sh></sat></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></immed_bry></pre></pre></pre></pre></pre></pre> | <pre>cprefixxSUBADDX(cond) Rd, Rm, Rm USADB8(cond) Rd, Rm, Rs, Rn WOV(cond){S} Rd, <operand2> MVM(cond){S} Rd, <operand2> MRS(cond) Rd, PSR> MRS(cond) Rd, CFIELds>, Rm MSR(cond) Rd, CFIELds>, Rcimmed_8r; MRS(cond) Rd, CRdLo, RdHi, Ac MRA(cond) Rd, <operand2> TST(cond) Rd, <operand2> TST(cond) Rd, <operand2> TST(cond) Rd, <operand2> TST(cond) Rd, <operand2> DOR(cond){S} Rd, Rn, <operand2> DOR(cond){S} Rd, Rn, <operand2> COM(cond){S} Rd, Rn, <operand2> DOR(cond){S} Rd, Rn, <operand2> DOR(cond){Rn, Operand2> CMP(cond) Rn, Operand2> SSATI(cond) Rd, #ssat>, Rm{, ASR <sh> SSATI(cond) Rd, #ssat>, Rm{, ASR <sh} #ssat="" rd,="" usat(cond)="">, Rm{, ISI <sh} #s<="" rd,="" td="" usat(cond)=""></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh}></sh></operand2></operand2></operand2></operand2></operand2></operand2></operand2></operand2></operand2></operand2></operand2></operand2></operand2></operand2></operand2></operand2></pre> |
| | d2> , Rm , # <immed_8r></immed_8r> | d2> , Rm , # <immed_8r></immed_8r> | dz> , Rm , # <immed_8r> cerand2></immed_8r> | dz> , Rm , # <immed_8r> erand2> erand2> erand2> erand2></immed_8r> | dz> , Rm , # <immed_8r> erand2> erand2> erand2> erand2></immed_8r> | d2> , Rm , # <immed_8r> erand2> erand2> erand2> erand2> erand2> erand2></immed_8r> | dz, Rm, Rm, R <pre>, Rm , #<immed_8r> erand2> erand2> erand2> erand2> erand2> erand2> Rm, ASR <sh>} RM</sh></immed_8r></pre> | dz> , Rm , Rm erand2> |
| | 8 | | 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 | | Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z | | | |
| | | | | | | ۵۵ | 000 | |
| Ku := oxerefere EOK Operand2 | Rd := PSR PSR := Rm (selected bytes only) PSR := inimed_8r (selected bytes only) RdLo := Ac[31:0], RdHi := Ac[39:32] Ac[31:0] := RdI.o, Ac[39:32] := RdHi Rd := Operand2 | Rd := PSR PSR := Rm (selected bytes PSR := immed_8r (selected RdLo := Ac[31:0], RdH; := Ac[31:0] := RdLo, Ac[39:0] Rd := Operand2 Urdane CPSR floor on Rn | Rd := PSR PSR := Rm (selected bytes only) PSR := inimed_8: (selected bytes only) PSR := inimed_8: (selected bytes only) RdLo := Ac[31:0], RdHi := Ac[39:32] Ac[31:0], := RdI .o, Ac[39:32] := RdHi Rd := Operand2 Update CPSR flags on Rn AND Operand2 Update CPSR flags on Rn FOR Operand2 Rd := RAND Operand2 Rd := RAND Operand2 | Rd := PSR PSR := Rm (selected PSR := Immed, 8r (selected Rd := Operand2 Rd := Operand2 Update CPSR flags r Update CPSR flags r Rd := Rn AND Opera Rd := Rn OR Operar Rd := Rn AND NOJ | Rd := PSR PSR := Rm (selected PSR := immed. &r (se RdLo := Ac[31:0]; RdI.o. A Ac[31:0] := RdI.o. A Rd := Operand2 Updane CPSR flags o Update CPSR flags o Update CPSR flags o Rd := Rn AND Oper Rd := Rn AND NOT Rd := Rn AND NOT Update CPSR flags o Update CPSR flags o | Rd := PSR PSR := Rm (selector PSR := inimical_8r (s Rd Lo := Ac[31:0]. Ac[31:0] := Rd1.o. Ac[31:0] := Rd1.o. Rd := Operand2 Update CPSR flags Update CPSR flags Rd := Rn AND Oper Rd := Rn AND Oper Rd := Rn AND NO: Update CPSR flags Rd := SignedSat(R) | Rd := PSR PSR := Rm (selecte PSR := Innned, Br. Rd Lo := Ac[31:0]. Ac[31:0] := RdLa, Ac[31:0] := RdLa, Rd := Operand2 Update CPSR flags Update CPSR flags Update CPSR flags Rd := Rn AND Oper Rd := Rn AND Oper Rd := Rn AND NO Update CPSR flags Update CPSR flags Update CPSR flags Rd := Rn AND NO Update CPSR flags Update CPSR flags Update CPSR flags Rd := SignedSat([R Rd := S | Rd := PSR PSR = Rm (selected bytes only) PSR:= immed_8r (selected bytes only) PSR:= immed_8r (selected bytes only) PSR:= immed_8r (selected bytes only) RdLo := Ac[31:0], RdHi := Ac[39:32] Ac[31:0] := RdLo, Ac[39:32] := RdHi Rd := Operand2 Update CPSR lags on Rn AND Operand2 Update CPSR lags on Rn FOR Operand2 Rd := Rn AND Operand2 Rd := Rn AND NOT Operand2 Rd := Rn AND NOT Operand2 Rd := Rn AND NOT Operand2 Update CPSR lags on Rn - Operand2 Update CPSR lags on Rn - Operand2 Update CPSR lags on Rn + Operand2 Rd := SignedSat((Rm ASR sh), sat). <sat> range 0-31, <sh> range 0-31. Rd := SignedSat((Rm LSL sh), sat). <sat> range 0-31, <sh> range 0-31. Rd := UnsignedSat((Rm LSL sh), sat). <sat> range 0-31, <sh> range 0-31. Rd := UnsignedSat((Rm LSL sh), sat). <sat> range 0-31, <sh> range 1-32. Rd := UnsignedSat((Rm LSL sh), sat). <sat> range 0-31, <sh> range 1-32. Rd := UnsignedSat((Rm LSL sh), sat). <sat> range 0-31, <sh> range 1-32. Rd := UnsignedSat((Rm LSL sh), sat). <sat> range 0-31, <sh> range 1-32. Rd := UnsignedSat((Rm LSL sh), sat). <sat> range 0-31, <sh> range 1-32.</sh></sat></sh></sat></sh></sat></sh></sat></sh></sat></sh></sat></sh></sat></sh></sat> |

| No Op | Software interrupt | | | | | | change | state | JOSS | | | | | | | Digital | Select | | | bytes | | extend with add | neimad | with add | Signed | | extend | Unsigned | | extend | Signed | 900 | operation |
|--------------|---|--------------------------------------|---------------------------------|---|---|--------------------------------------|------------------------|---|--|--------------------------|---|-------------------------|---|--|--|--|--|---|---|---|--|-----------------------------|--|---|---|---|---|--|---|--|--|---|----------------------------|
| No operation | Software interrupt | Breakpoint | Return from exception | Store return state | | Sct endianness | Change processor mode | | Change processor state | and change to Java state | with link and exchange (2) | | with link and exchange (1) | and exchange | with link | branch | Select bytes | in low halfword, sign extend | in both halfwords | In word | Byte to word, add | Two bytes to halfwords, add | Byte to word, add | Two bytes to halfwords, add | Hallword to word, add | Byte to word | Two bytes to halfwords | Halfword to word | Byte to word | Two bytes to halfwords | Hallword to word | Pack halfword top + bottom | Dook halfamad harram v tan |
| 5 | | ᆫ | 2 | | | 0, | | 6 | ٠٥, | 51, 6 | U | | ¥ | 4T,5 | | | 0 | ٥ | | | ┺ | | | | 9 | 6 | ٥ | | 6 | | 6 | 6 0 | |
| NOP | SWI{cond} <immed_24></immed_24> | BKPT <immed_16></immed_16> | RFE <a_mode41> Rn{!}</a_mode41> | SRS <a_mode4s> #<p_mode>{!}</p_mode></a_mode4s> | | SETEND <endianness></endianness> | | CPSIE <iflags> {, #<p_mode>}</p_mode></iflags> | CPSID <iflags> {, #<p_mode>}</p_mode></iflags> | SI, 6 BXJ (cond) Rm | BLX{cond} Rm | | BLX label | 4T,5 BX{cond} Rm | BL{cond} label | B{cond} 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> | n, Rm | SXTAB(cond) Rd, Rn, Rm(, ROR # <sh>)</sh> | SXTAB16{cond} Rd, Rn, Rm{, ROR # <sh>}</sh> | SXTAH(cond) Rd, Rn, Rm(, ROR # <sh>)</sh> | UXTB{cond} Rd, Rm(, ROR # <sh>)</sh> | UXTB16{cond} Rd, Rm{, ROR # <sh>}</sh> | UXIH{cond} Rd, Rm{, ROR # <sh>}</sh> | SXTB{cond} Rd, Rm{, ROR # <sh>}</sh> | SXTB16(cond) Rd, Rm(, ROR # <sh>)</sh> | SXTH{cond} Rd, Rm{, ROR # <sh>}</sh> | PKHTB(cond) Rd, Rn, Rm(, LSL # <sh>)</sh> | 2 |
| None | Software interrupt processor exception. | Prefetch abort or enter debug state. | PC := [Rn], CPSR := [Rn + 4] | [R13m] := R14, [R13m + 4] := CPSR | <endianness> can be BE (Big Endian) or LE (Little Endian).</endianness> | Sets endianness for loads and saves. | | 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 | | R14 := address of next instruction, R15 := label, Change to Thumb | R15 := Rm, Change to Thumb if Rm[0] is 1 | R14 := address of next instruction, R15 := label | K15 := labe: | 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 | $Rd[15:8] := Rm[7:0], Rd[7:0] := Rm[15:8], \\ Rd[31:24] := Rm[23:16], Rd[23:16] := Rm[31:24]$ | $Rd[51:24] := Rm[7:0], Rd[23:16] := Rm[15:8], \\ Rd[15:8] := Rm[23:16], Rd[7:0] := Rm[31:24]$ | 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] := ZcroExtend((Rm ROR (8 * sh))[15:0]). sh 0-3. | Rd[31:0] := SignExtend((Rm ROR (8 * sh))[7:0]). sh 0-3. | Rd[31:16] := SignExtend((Rm ROR (8 * sh))[23:16]), Rd[15:0] := SignExtend((Rm ROR (8 * sh))[7:0]), sh 0-3. | Rd[31:0] := SignExtend((Rm ROR (8 * sh))[15:0]). sh 0-3. | [Kd[3]:16] := Kn[3]:16], Kd[3]:10] := (Km LSL sn)[3]:16], sn 0-31. $[Kd[3]:16] := Kn[3]:16], Kd[15:0] := (Km ASR sh)[15:0], sh 1-32.$ | Action |
| | 24-bit value encoded in instruction. | Cannot be conditional. | Cannot he conditional. | Cannot be conditional. | | Cannot be conditional. | Cannot be conditional. | Cannot be conditional. | Cannot be conditional. | | | of current instruction. | Cannot be conditional. | | label must be within ±32Mb of current instruction. | label must be within ±32Mb of current instruction. | | | | | | | | | | | | | | | | | Notes |

| | | 1 | | | |
|-----------------|---------------------------------------|------|--|--|---------------------------------|
| Operation | | is | 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 (§ ST: and exchange) | | LDR(cond) R15, <a_mode2></a_mode2> | R15 := [address][31:1] | |
| | | | | (§ 5T: Change to Thumb if [address][0] is 1) | |
| | Вусе | | 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 := ZcroExtent[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 | ij | 5E* LDR(cond)D Rd, <a_mode3></a_mode3> | | 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> | Load list of registers from [Rn] | |
| | return (and exchange) | | <pre>LDM(cond)<a_mode4l> Rn(!), <reglist+pc></reglist+pc></a_mode4l></pre> | Load registers, R15 := [address][31:1] | |
| | 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 | | LDM{cond} <a_mode4l> Rn, <reglist-pc>^</reglist-pc></a_mode4l> | | Use from privileged modes only. |
| Soft preload | Memory system hint | E SE | 5E* PLD <a_mode2></a_mode2> | Memory may prepare to load from address | 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] | |
| | Daubleword | SE# | 5F* 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 | | STM(cond) <a_mode4s> Rn(!), <reglist></reglist></a_mode4s> | Store list of registers to [Rn] | |
| | User mode registers | | STM(cond) <a_mode4s> Rn(1), <reglist>^</reglist></a_mode4s> | 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. |
| Swap | Word | ψı | SWP(cond) Rd, Rm, [Rn] | temp := [Rn], [Rn] := Rm, Rd := temp | |
| | Вую | ψ | SWP(cond)B Rd, Rm, [Rn] | temp := $ZeroExtend([Rn][7:0])$, | |
| | | ľ | | | |

ARM Addressing Modes Quick Reference Card

| Pre-indexed Immediate officet [Pn #+/immed 10-15] | Pre-indexed Immediate offert | 102 | 11/1/21 Pammit-/-/+ | 7 | 2-11-1 | |
|---|---|--------|----------------------------------|-------|------------------------------------|--|
| | Zero offset | R | | 1 | | Fanivalent to (Rn #0) |
| | Register offset | Rn | [Rm, +/-Rm]{!} | Ξ | | |
| | Scaled register offset [Rn, +/-Rm, LSL # <shift>] {!} Allowed shifts 0-31</shift> | [Rm, | +/-Rm, | TST # | | Allowed shifts 0-31 |
| | | Rn, | +/-Rm, | LSR # | <shift>] [1]</shift> | $[Rn, +/-Rm, LSR \#] \{ l \} Allowed shifts 1-32$ |
| | | [Rn | +/-Rm, | ASR # | <shift>] [:}</shift> | [Rn, +/-Rm, ASR # <shift>][!] Allowed shifts 1-32</shift> |
| | | [Rn. | +/-Rm, | ROR # | <shift>] {!}</shift> | [Rn, +/-Rm, ROR # <shift>] {!} Allowed shifts 1-31</shift> |
| | | Em, | <pre>[Rn, +/-Rm, RRX]{!}</pre> | RRX] | Ξ. | |
| Post-indexed | Immediate offset | [Rn], | [Rn], #+/- <immed_12></immed_12> | med | 12> | |
| | Register offset | [Rn], | +/-Rm | | | |
| | Scaled register offset [Rn], +/-Rm, LSL # <shift></shift> | [Rn], | +/-Rm, | TST | | Allowed shifts 0-31 |
| | | [Rn], | +/Rm, | LSR | 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 | æ | | |

| | | | Post | Àd | | Γ | | | | | _ | Post | : | | | | | | | |
|------------------------------------|-----------------|-----------------------|----------------------------------|---------------------------------------|-----------------------------|-----------------------|------------------------------------|------------------------------|----------------------------------|-------------------------------|------------------------------|----------------------------------|---------------------|--|---|---|--|-----------------|-----------------------|----------------|
| | | | -indexed | ressing | | | | | | | | Post-indexed | | | | | | | | |
| Scaled register offset | Register offset | Zero offset | Post-indexed Immediate offset | Addressing Mode 2 (Post-indexed only) | | | | | | Scaled register offset | Register offset | Immediate offset | | | | | Scaled register offset | Register offset | Zero offset | |
| [Rn], | [Rm], | [Rn] | [Rm], | ad only | | [Rn], | [Rn], | [Rn] , | [Rn] , | [Rn], | [Rn] , | [Rn], | [Rn, | Rn. | [Rn, | Rn, | [Rn | R | [Ra] | |
| [Rn], +/-Rm, LSL # <shift></shift> | +/-Rm | | [Rn], #+/- <immed_12></immed_12> | | | [Rn], +/-Rm, RRX | [Rn], +/-Rm, ROR # <shift></shift> | +/-Rm, ASR # <shift></shift> | +/-Rm, LSR # <shift></shift> | +/-Rm, LSL # <shift></shift> | +/-Rm | [Rn], #+/- <immed_12></immed_12> | [Rn, +/-Rm, RRX]{!} | [Rn, +/-Rm, ROR # <shift>] {!} Allowed shifts 1-31</shift> | $[Rn, +/-Rm, ASR \#] \{i\} Allowed shifts 1-32$ | +/-Rm, LSR # <shift>] {!} Allowed shifts 1-32</shift> | $[Rn, +/-Rm, LSL \#]{!} Allowed shifts 0-31$ | +/-Rm] { ! } | | |
| Allowed shifts 0-31 | | Equivalent to [Rn],#0 | | | | | Allowed shifts 1-31 | Allowed shifts 1-32 | Allowed shifts 1-32 | Allowed shifts 0-31 | | | | Allowed shifts 1-31 | Allowed shifts 1-32 | Allowed shifts 1-32 | Allowed shifts 0-31 | | Equivalent to [Rn,#0] | |
| PSR fields | | Rotate | Arithn | Logica | Logica | Rotate | Register | Rotate | Aritho | Logica | Logica | Imme | Flexi | | XS | #E# | πE | 3 | nT, nI | 7 |
| fields | | Rotate right register | netic shift 1 | Logical shift right register | Logical shift left register | Rotate right extended | Ċ, | Rotate right immediate | netic shift ı | al shift righ | Logical shift left immediate | Immediate value | Flexible Operand 2 | | | * | | | nJ | |
| (use at least or | | ter | Arithmetic shift right register | ıt register | register | nded | | ediate | Arithmetic shili right immediate | Logical shift right immediate | immediate | | nd 2 | | XScale coprox | E variants of a | All E variants | ARM architec | T or J variants | CIVIAL GICTIAN |

| | | | | Scalco | Regist | Post-indexed Immer | | | | | Scale | Regist | Zero offset | Pre-indexed Immediate offset | |
|-----------------------|------------------------------|------------------------------|----------------------------------|-------------------------------|------------------------------|----------------------------------|---------------------|--|--|--|---|------------------|-----------------------|-------------------------------------|--|
| | • | | | Scaled register offset | Register offset | Immediate offset | | | | | register offset | Register offset | offset | | |
| E . | [Rn], | [Rn], | [Rn], | [Rn], | [RE] , | [Rn], | En, | [Rn, | Rn, | Rn, | [Rn | Rn | [Rn] | R | |
| [Rn], +/-Rn, RRX | +/-Rm, Ro | +/-Rm, At | +/Rm, L: | +/-Rm, L: | +/-Rm | [Rn], #+/- <immed_12></immed_12> | [Rn, +/-Rm, RRX]{!} | +/-Rm, ROI | +/-Rm, ASI | +/-Rm, LSI | +/-Rm, LS1 | [Rn, +/-Rm]{!} | | [Rn, #+/- <immed_12>]{!}</immed_12> | The state of the s |
| Z. | +/-Rm, ROR # <shift></shift> | +/-Rm, ASR # <shift></shift> | +/-Rm, LSR # <shift></shift> | +/-Rm, LSL # <shift></shift> | | ed_12> | X) (:) | [Rn, +/-Rm, ROR # <shift>] {!} Allowed shifts 1-31</shift> | $[Rn, +/-Rm, ASR \#]{!} Allowed shifts 1-32$ | $[Rn, +/-Rm, LSR \#]{!} Allowed shifts 1-32$ | Scaled register offset [Rn, +/-Rm, LSL # <shift>] { } Allowed shifts 0-31</shift> | | | d_12>] {!} | |
| | Allowed shifts 1-31 | Allowed shifts 1-32 | Allowed shifts 1-32 | Allowed shifts 0-31 | | | | Allowed shifts 1-31 | Allowed shifts 1-32 | Allowed shifts 1-32 | Allowed shifts 0-31 | | Equivalent to [Rn,#0] | | |
| Rotate right extended | Register | Rotate right immediate | Arithmetic shili right immediate | Logical shift right immediate | Logical shift left immediate | Immediate value | Flexible Operand 2 | | XS | *32 | ηE | Z | nT, nI | 77 | THE PROPERTY OF THE PROPERTY O |
| nded | | ediate | tight immediate | nt immediate | immediate | | ınd 2 | | XScale coprocess | E variants of ARI | All E variants of | ARM architecture | T or J variants of | ARM architectur | ale Aeraiolia |

Rm, LSL #<shift>
Rm, LSR #<shift>
Rm, ASR #<shift>
Rm, ROR #<shift>

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

#<immed_8r>

, ms , ms , ms

RRX LSL Rs LSR Rs ASR Rs ROR Rs

| | ARM architecture versions | ture versions |
|----------------|---------------------------|---|
| | 77 | ARM architecture version n and above. |
| ent to [Rn,#0] | nT, nI | T or J variants of ARM architecture version n and above. |
| | × | ARM architecture version 3M, and 4 and above, except xM variants. |
| shifts 0-31 | пE | All E variants of ARM architecture version n and above. |
| shifts 1-32 | *35* | E variants of ARM architecture version n and above, except xP variants. |
| shifts 1-32 | XS | XScale coprocessor instruction |
| | | |

| | | , | |
|--------------|--|-------------|------------------|
| Addressing N | Addressing Mode 4 - Multiple Data Transfer | ta Transfer | |
| Block load | | Stack pop | |
| IA | Increment After | FD | Full Descending |
| я | Increment Before | ED | Empty Descending |
| DA | Decrement After | FA | Full Ascending |
| DB | Decrement Before | EA | Empty Ascending |
| Block store | Þ | doing dacts | |

| Addressing | Mode 3 - Halfword, | Mode 3 - Halfword, Signed Byte, and Doubleword Data Transfe | ransfer |
|--------------|--------------------|---|-----------------------|
| Pre-indexed | Immediate offset | [Rn, #+/- <immed 8="">] [!}</immed> | |
| | Zero offset | [Rn] | Equivalent to [Rn,#0] |
| | Register | [Rn, +/-Rm]{!} | |
| Post-indexed | Immediate offset | [Rn], #+/- <immed_8></immed_8> | |
| | Register | [Rn]. +/~Rm | |

[Rn], +/-Rm, LSR #<shift>
[Rn], +/-Rm, ASR #<shift>
[Rn], +/-Rm, ROR #<shift>
[Rn], +/-Rm, REX

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

| | POH BEIGS | (use at least one suffix) | |
|---|-----------|---------------------------|-------------|
| | Suffix | Meaning | |
| | a | Control field mask byte | PSR[7:0] |
| | H | Flags field mask byte | PSR[31:24] |
| _ | ω | Status field mask byte | PSR[23: 16] |
| J | × | Extension field mask byte | PSR[15:8] |
| | | | |

| SR fields | (use at least one suffix) | |
|-----------|---------------------------|------------|
| Suffix | Meaning | |
| ດ | Control field mask byte | PSR[7:0] |
| H | Flags field mask byte | PSR[31:24] |
| to | Status field mask byte | PSR[23:16] |
| × | Extension field mask byte | PSR[15:8] |

| <u>+1</u> | 27 | 23 | 19 | 18 | 17 | 16 | Processor Modes | |
|-----------|---------------|---------------|----------------|---------------|--------------------|---------------|-------------------|--|
| System | Undelined | Abort | Supervisor | IRQ Interrupt | FIQ Fast Interrupt | User | odes | |
| | ИH | ğ | □ | HS | Ø | Ø | Prefi | |
| | Unsigned aria | Unsigned satu | Unsigned arith | Signed arithm | Signed satural | Signed arithm | refixes for Paral | |

Post-indexed Unindexed

Immediate offset No offset

[Rn]

Equivalent to [Rn,#0]

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

[8-bit copro. option]

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

Prc-indexed

Immediate offset Zero offset

Increment After
Increment Before
Decrement After
Decrement Before

Empty Ascending
Full Ascending
Empty Descending
Full Descending

Addressing Mode 5 - Coprocessor Data Transfer

| Prefi | Prefixes for Parallel Instructions |
|-------|---|
| Ø | Signed arithmetic modulo 28 or 216, sets CPSR GE bits |
| Ø | Signed saturating arithmetic |
| HS | Signed arithmetic, halving results |
| □ | Unsigned arithmetic modulo 28 or 216, sets CPSR GE bits |
| g | Unsigned saturating arithmetic |
| UH | Unsigned arithmetic, halving results |
| | |
| | SH UQ |

| S / HS C / LO MI PL VS VC HI GE GE LT GE TH | Condition Field Minemonic EQ | Description Equal |
|--|------------------------------------|--|
| Negative Positive or zero Overflow No overflow No overflow Unsigned higher Unsigned lower or sa Signed greater than o Signed less than Signed less than or ce Always (normally or Odes User FIQ Fast Interrupt IRQ Interrupt Supervisor | | Not equal Carry Set / Unsigned higher or same Carry Cher / Hindonski hower |
| r or sam han or s | <u> </u> | Carry Set / Unsigned higher o Carry Clear / Unsigned lower |
| rflow speed higher gened lower or same led greater than or equal led less than led greater than led greater than led sess than or equal ays (normally omitted) Fref Fast interrupt Greater than SH U Fref U | Posi | ative tive or zero |
| signed higher signed lower or same signed lower or equal ince less than ince greater than or equal ince less than inc | ž õ | erflow |
| signed lower or sam ined greater than or e ined less than ined greater than ined less than or equi ined less than or i | | Unsigned higher |
| gned greater than or equipmed less than gned greater than or equipmed greater than gned greater than gned greater than or equipmed greater than gned gned gned gned gned gned gned gne | | Unsigned lower or sa |
| Less than, or unordered GT Signed greater than LEss than, or unordered GT Signed greater than Less than, or unordered AL Always (normally omitted) Always (normally omitted) Cessor Modes GE User GEST Modes GE | æ | Signed greater than o |
| gred greater than gred less than or equal ways (normally omitted) Prefixer or Q Fast Interrupt Q Interrupt Q Interrupt Q SH pervisor U | H | Signed less than |
| ways (normally omit ways (normally omit or or Q Fast Interrupt Q Interrupt pervisor | GI | Signed greater than |
| ways (normally omit | LE | Signed less than or e |
| er Q Fast Interrupt Q Interrupt pervisor | AL | Always (normally or |
| er 2 Fast Interrupt 2 Interrupt pervisor | Processor M | ndes |
| FIQ Fast Interrupt IRQ Interrupt Supervisor | 16 | User |
| IRQ Interrupt Supervisor | 17 | FIQ Fast Interrupt |
| | 18 | IRQ Interrupt |
| 7 | 3 19 | Supervisor |

ARM Addressing Modes Quick Reference Card

| Coprocessor operations | Ş | Assembler | Action | Notes |
|---------------------------------------|----------|--|-----------------------|------------------------|
| Data operations | 2 | CDP(cond) <copr>, <opr>, CRd, CRn, CRm{, <opr>, <opr></opr></opr></opr></copr> | Coprocessor dependent | |
| Alternative data operations | Ls. | CDP2 <copr>, <opl>, CRd, CRn, CRm{, <op2>}</op2></opl></copr> | Coprocessor dependent | Cannot be conditional. |
| Move to ARM register from coprocessor | 2 | MRC(cond) <copr>, <op1>, Rd, CRn, CRm{, <op2>}</op2></op1></copr> | Coprocessor dependent | |
| Alternative move | ٠, | MRC2 <copr>, <opl>, Rd, CRn, CRm{, <op2>}</op2></opl></copr> | Coprocessor dependent | Cannot be conditional. |
| Two ARM register move | Ħ | 5E* MRRC(cond) <copr>, <opl>, Rd, Rn, CRm</opl></copr> | Coprocessor dependent | |
| Alternative two ARM register move | | MRRC2 <copr>, <opl>, Rd, Rn, CRm</opl></copr> | Coprocessor dependent | Cannol be conditional. |
| Move to coproc from ARM reg | 13 | MCR(cond) <copr>, <opl>, Rd, CRn, CRm(, <op2>)</op2></opl></copr> | Coprocessor dependent | |
| Alternative move | <u>.</u> | MCR2 <copr>, <opl>, Rd, CRn, CRm{, <op2>}</op2></opl></copr> | Coprocessor dependent | Cannot be conditional. |
| Two ARM register move | 뜐 | 5E* MCRR(cond) <copr>, <opl>, Rd, Rn, CRm</opl></copr> | Coprocessor dependent | • |
| Alternative two ARM register move | <u></u> | 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 | <u>ب</u> | 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 | <u>ر</u> | STC2 <copr>, CRd, <a_mode5></a_mode5></copr> | Coprocessor dependent | Cannot be conditional. |

orietary Notic

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

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

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

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

Document Number

ARM QRC 0001H

Change Log

| Н | G * | 11 (12 | 10 | חו | B | A | issue |
|----------------|-----------------|---------------|----------------|---------------|----------------|---------------|--------|
| Oct 2003 | Jan 2003 | Oct 2000 | Oct 1999 | Nov 1998 | Sept 1996 | June 1995 | Date |
| CKS | CKS CKS | CKS | CKS | ВЈН | ВЈН | BJH | Ву |
| Highth Release | Seventh Release | Fitth Release | Fourth Release | Third Release | Second Release | First Release | Change |