Marvell® WMMX™ 2 Coprocessor

Programmers Reference Manual

Product number to be assigned at a future date

Doc. No. MV-TBD-00, Rev. A
December 5, 2006
Document Classification: Proprietary Information
Not approved by Document Control. For review only.

MOVING FORWARD FASTER®

MARVELL®

Document Conventions

Note

Provides related information or information of special importance.

Caution

Indicates potential damage to hardware or software, or loss of data.

Warning

Indicates a risk of personal injury.

Document Status

Draft	For internal use. This document has not passed a complete technical review cycle and ECN signoff process.
Preliminary Tapeout (Advance)	This document contains design specifications for a product in its initial stage of design and development. A revision of this document or supplementary information may be published at a later date. Marvell may make changes to these specifications at any time without notice. Contact Marvell Field Application Engineers for more information.
Preliminary Information	This document contains preliminary specifications. A revision of this document or supplementary information may be published at a later date. Marvell may make changes to these specifications at any time without notice. . Contact Marvell Field Application Engineers for more information.
Complete Information	This document contains specifications for a product in its final qualification stages. Marvell may make changes to these specifications at any time without notice. Contact Marvell Field Application Engineers for more information.

Milestone Indicator:
Draft = 0.xx
Advance = 1.xx
Preliminary = 2.xx
Complete = 3.xx

X . Y Z

Work in Progress Indicator
Zero means document is released.

Various Revisions Indicator

Doc Status: Preliminary	Technical Publication: 0.xx

For more information, visit our website at: www.marvell.com

Contents

1 Introduction 11
1.1 About This Document 11
1.1.1 Other Relevant Documents 11
1.1.2 Who to Contact About This Document 11
1.2 Marvell® Wireless MMX™ 2 Coprocessor Instruction Set Overview 11
1.2.1 Marvell® WMMX™ 2 Technology Compatibility 11
1.2.2 SSE Compatibility 13
1.2.3 New Media Technology Extensions 13
1.2.4 Transfer Instruction Mapping 14
1.2.5 Other 16
1.2.6 Load/Store 16
1.2.7 Synthetic Instructions 16
1.3 Terminology and Conventions 17
1.3.1 Number Representation 17
1.3.2 Terminology and Acronyms 17
1.3.3 Symbols Glossary 17
1.3.4 Data Types 18
1.3.5 Register Names 18
1.3.6 Instruction Qualifiers 19
1.3.7 Conditional Execution Specifiers 19
2 Programmers Model 21
2.1 ARM* Overview 21
2.2 Coprocessor Model 21
2.3 Marvell® Wireless MMX™ 2 Coprocessor 21
2.3.1 Coprocessor Instructions 21
2.3.2 Enabling Marvell® Wireless MMX™ 2 Coprocessor 22
2.4 Marvell® Wireless MMX™ 2 Coprocessor Register File Organization 22
2.5 Marvell® Wireless MMX™ 2 Coprocessor Status and Control Registers 23
2.5.1 Register Map 23
2.5.2 Arithmetic Flags 24
2.5.3 Saturation Flags 26
2.5.4 General-Purpose Registers 28
2.5.5 Coprocessor ID 29
2.5.6 Coprocessor Control Register 30
2.6 Big-Endian and Little-Endian Support 30
2.7 Qm.n Fixed Point Format 31
2.8 Saturating Integer Arithmetic 31
2.9 Saturating Q15 and Q31 Fractional Arithmetic 31
2.10 Multiply and Multiply-Accumulate Fractional Modes 32
2.11 Biased Rounding of Multiplier Results 32

2.12 Cross (X) Selection of Operands 33
2.13 Instruction Execution 33
2.13.1 Conditional Execution 33
2.13.2 Reserved Instruction Fields 34
3 Marvell® Wireless MMX™ 2 Coprocessor Instruction Set 35
3.1 Instruction Summary 35
3.2 Marvell® Wireless MMX™ 2 Coprocessor Instruction Set Details 37
3.2.1 TANDC 37
3.2.2 TBCST 38
3.2.3 TEXTRC 40
3.2.4 TEXTRM 42
3.2.5 TINSR 43
3.2.6 TMCR 44
3.2.7 TMCRR 45
3.2.8 TMIA 46
3.2.9 TMIAPH 47
3.2.10 TMIAxy 48
3.2.11 TMOVMSK 49
3.2.12 TMRC 50
3.2.13 TMRRC 51
3.2.14 TORC 51
3.2.15 TORVSC 52
3.2.16 WABS 53
3.2.17 WABSDIFF 54
3.2.18 WACC 56
3.2.19 WADD 56
3.2.20 WADDSUBHX 59
3.2.21 WALIGNI 60
3.2.22 WALIGNR 61
3.2.23 WAND 62
3.2.24 WANDN 62
3.2.25 WAVG2 63
3.2.26 WAVG4 64
3.2.27 WCMPEQ 65
3.2.28 WCMPGT 66
3.2.29 WLDR 68
3.2.30 WMAC 71
3.2.31 WMADD 72
3.2.32 WMAX 74
3.2.33 WMERGE 75
3.2.34 WMIAxy 76
3.2.35 WMIAWxy 77
3.2.36 WMIN 79
3.2.37 WMOV 80
3.2.38 WMUL 80

3.2.39 WMULW 82
3.2.40 WOR 83
3.2.41 WPACK 84
3.2.42 WQMIAxy 86
3.2.43 WQMULM 88
3.2.44 WQMULWM 90
3.2.45 WROR 91
3.2.46 WSAD 93
3.2.47 WSHUFH 94
3.2.48 WSLL 96
3.2.49 WSRA 98
3.2.50 WSRL 100
3.2.51 WSTR 102
3.2.52 WSUB 105
3.2.53 WSUBADDHX 106
3.2.54 WUNPCKEH 107
3.2.55 WUNPCKIH 110
3.2.56 WUNPCKEL 111
3.2.57 WUNPCKIL 113
3.2.58 WXOR 115
3.2.59 WZERO 116
A Integer Instruction Encoding Summary 117
A.1 Coprocessor Data (CDP) Instructions 117
A.2 Coprocessor Data Transfer Instructions 119
A.2.1 Transfers to Coprocessor Register (MCR) 119
A.2.2 Transfers to Coprocessor Register (MCRR) 120
A.2.3 Transfers from Coprocessor Register (MRC) 121
A.2.4 Transfers from Coprocessor Register (MRRC) 121
A.3 Load Store Instructions 122
A.3.1 Load/Store to Marvell® Wireless MMX™ 2 Coprocessor Data Registers 122
A.3.2 Load/Store to Marvell® Wireless MMX™ 2 Coprocessor Control Registers 122
A.3.3 Load/Store with Register Offset to Marvell® Wireless MMX™ 2 Coprocessor Data Registers 122
B Mapping to Marvell® Wireless MMX™ 2 Technology and SSE 123
B.1 Marvell® Wireless MMX™ 2 Coprocessor Mapping 123
C Marvell® Wireless MMX™ 2 Coprocessor Instruction Additions 125
C.1 Introduction 125
C.2 Enhancements to Existing Instructions 125
C.3 New Instructions 126
D Performance Optimization 129
D.1 Introduction 129
D.2 Issue Cycle and Result Latency 129

D.3 Performance Hazards 131
D.3.1 Data Hazards 131
D.3.2 Resource Hazard 132

Figures

2-1 Register File Organization 22
D-1 High-Level Pipeline Organization 132

THIS PAGE INTENTIONALLY LEFT BLANK

Tables

1-1 Symbol and Function Glossary 17
1-2 ARM* to IA-32 Data-Type Name Mapping 18
2-1 Status and Control Register Mappings 23
2-2 Register Bitmap for wCASF 24
2-3 Register Format of wCASF After a Byte Operation – SIMD8 PSR 24
2-4 Register Format for wCASF After a Half-Word Operation – SIMD16 PSR 25
2-5 Register Format for wCASF After a Word Operation – SIMD32 PSR 25
2-6 Register Bitmap for wCASF After a Double-Word Operation – SIMD64 PSR 25
2-7 Register Bitmap for wCSSF – SIMD Saturation Flags 27
2-8 Register Bitmap for wGRn Registers 28
2-9 Register Bitmap for wCID – Coprocessor ID 29
2-10 Register Bitmap for wCon – Coprocessor Control 30
2-11 Little-Endian Byte Order 30
2-12 Big-Endian Byte Order 30
2-13 Effect of Reserved fields on Execution 34
3-1 Immediate Offset Address Mode Encoding 69
3-2 Address Mode Assembler Offset Specification and Instruction Encoding 69
3-3 Register offset address mode encoding 70
3-4 Immediate offset address mode encoding 103
3-5 Address Mode Assembler Offset Specification and Instruction Encoding 103
3-6 Register offset address mode encoding 104
A-1 CDP Instruction Encoding 117
A-2 Coprocessor Data Transfer Instruction Encoding 119
A-3 Coprocessor Data Transfer Multiply-Accumulate Instruction Encoding 120
A-4 Transfers to Coprocessor Register Instruction Encoding 120
A-5 Transfers From Coprocessor Register Subfield Instruction Encoding (MRC) 121
A-6 Transfers from Coprocessor Register Subfield Instruction Encoding (MRRC) 122
B-1 Mapping Marvell® Wireless MMX™ 2 Coprocessor Instructions to Marvell® Wireless MMX™ 2 Technology and SSE Integer Instructions 123
C-1 Enhancements to Existing Instructions 125
C-2 New Media Processing Instructions 126
D-1 Issue Cycle and Result Latency of the Marvell® Wireless MMX™ 2 Coprocessor Instructions 129
D-2 Resource Availability Delay for the Execution Pipeline 133
D-3 Multiply Pipe Instruction Classes 134
D-4 Resource Availability Delay for the Multiplier Pipeline 134
D-5 Resource Availability Delay for the Memory Pipeline 135
D-6 Resource Availability Delay for the Coprocessor Interface Pipeline 136

THIS PAGE INTENTIONALLY LEFT BLANK

Introduction 1

1.1 About This Document

This document is the definitive reference for the Marvell® Wireless MMX™ 2 coprocessor microarchitecture.

1.1.1 Other Relevant Documents

- *ARM Architecture Version 5T Specification* Document Number: ARM DDI 0100D-10
  This document describes Version 5T of the ARM* architecture, which includes Thumb ISA. This document is not publicly available; contact the individuals listed in Section 1.1.2 for more information on how to obtain a copy.
- *Intel XScale® Microarchitecture Megacell Users Guide*.

1.1.2 Who to Contact About This Document

Comments, corrections, or suggestions about this document should be sent to:

Marvell Semiconductor
1601 S. Mopac, Suite 400
Austin, TX 78746

1.2 Marvell® Wireless MMX™ 2 Coprocessor Instruction Set Overview

This section provides an overview of the instruction set and identifies which instructions offer the same functionality as the Marvell® Wireless MMX™ 2 technology and Streaming SIMD Extensions (SSE) instructions. The new media processing extension instructions are also introduced. Section 1.3 describes the terminology and usage of the data type and register specifiers used in this section.

1.2.1 Marvell® WMMX™ 2 Technology Compatibility

WADD Performs vector addition of wRn and wRm for vectors of 8-, 16-, or 32-bit signed or unsigned data, and places the result in wRd. saturation can be specified as signed, unsigned, or no saturation. The use of the carry flags from the wCASF register may be optionally supplied as the Carry-in to the addition operation using the C-qualifier. The add with carry-in option is valid for 16 and 32-bit operands only.

WAND Performs a bitwise logical “AND” between wRn and wRm, and places the result in the destination register, wRd.

WANDN Performs a bitwise logical “AND” between wRn and “NOT” wRm, and places the result in the destination register, wRd.

WCMPEQ Performs vector-equality comparison of wRn and wRm for vectors of 8-, 16-, or 32-bit data, setting the corresponding data elements of wRd to all ones when source operands are equal; else all zeros.

WCMPGT Performs vector-magnitude comparison of wRn and wRm for vectors of 8-, 16-, or 32-bit data, setting the corresponding data elements of wRd to all ones when corresponding fields of wRn and greater than wRm; else all zeros; operation can be performed on either signed or unsigned data.

WMUL Performs a vector multiplication of wRn and wRm on vectors of 16-bit data only; M-qualifier indicates that the higher order 16 bits of the result are to be stored in wRd, the L-qualifier indicates that the lower 16 bits of the result are to be stored in wRd; can be performed on signed or unsigned data. Biased rounding is available with the M-qualifier through the use of the R-qualifier.

WMADD Performs a 16-bit vector multiplication and then sums the lower two products into the bottom word of wRd, and the upper two products into the upper word of wRd; intermediate products are 32 bit; can operate on either signed or unsigned data. The operands may be cross multiplied with the use of the X-qualifier and may be optionally subtracted with the use of the N-qualifier. The X-qualifier may only be used with the default multiply-add operation.

WOR Performs a bitwise logical “OR” between wRn and wRm and places the result in the destination register, wRd

WPACK Packs data from wRn and wRm into wRd, with wRm being packed into the upper half, and wRn being packed into the lower half for vectors of 16-, 32-, or 64-bit data, and saturate the results and place in the destination register wRd; packing can be performed with signed saturation or unsigned saturation

WSLL Performs vector logical shift-left of wRn by wRm , or a control register (wCx), or a 5-bit immediate for vectors of 16-, 32-, or 64-bit data and places the result in wRd.

WSRA Performs vector arithmetic shift-right of wRn by wRm, or a control register (wCx), or a 5-bit immediate for vectors of 16-, 32-, or 64-bit data and places the result in wRd.

WSRL Performs vector logical shift-right of wRn by wRm, or a control register (wCx), or a 5-bit immediate for vectors of 16-, 32-, or 64-bit data and places the result in wRd.

WSUB Performs vector subtraction of wRm from wRn for vectors of 8-, 16-, or 32-bit signed or unsigned data, and places the result in wRd. Saturation can be specified as signed, unsigned, or no saturation.

WUNPCKEH Unpacks 8-bit, 16-bit, or 32-bit data from the top half of wRn (the source register), and either zero- or signed- extends each field, and places the result into the destination register, wRd.

WUNPCKIH Unpacks either 8-bit, 16-bit, or 32 bit data from the top half of wRn, interleaves with the top half of wRm, and places the result into the destination register, wRd.

WUNPCKEL Unpacks either 8-bit, 16-bit, or 32 bit data from the lower half of wRn (the source register), and either zero- or sign- extends each field, and places the result into the destination register, wRd.

WUNPCKIL Unpacks either 8-bit, 16-bit, or 32 bit data from the lower half of wRn and the lower half of wRm, and places the result into the destination register, wRd.

WXOR Performs a bitwise logical “XOR” between wRn and wRm, and places the result in wRd.

1.2.2 SSE Compatibility

WAVG2 Performs a 2-pixel average of wRn and wRm on unsigned vectors of 8- or 16-bit data with optional biased rounding and places the result in the destination register, wRd.

WMAX Performs vector maximum selection of elements from wRn and wRm for vectors of 8-, 16-, or 32-bit data and places the maximum fields in the destination register, wRd; can be performed on signed or unsigned data.

WMIN Performs vector minimum selection of elements from wRn and wRm for vectors of 8-, 16-, or 32-bit data, and places the minimum fields in the destination register, wRd; can be performed on signed or unsigned data.

WSAD Performs the sum of absolute differences of wRn and wRm, and accumulates the result with wRd; can be applied to 8-bit or 16-bit unsigned data vectors.

WSHUFH Select (shuffle) 16-bit data values in destination register, wRd, from 16-bit fields in source register specified by the 8-bit immediate value.

1.2.3 New Media Technology Extensions

WABS Performs a vector absolute value of wRn on 8-bit, 16-bit, or 32-bit signed data and places the result into the destination register, wRd.

WABSDIFF Performs a vector absolute value of the differences of wRm and wRn for vectors 8-bit, 16-bit, or 32-bit unsigned data, and places the result in wRd.

WADDSUBX Performs complex vector addition/subtraction of wRn and wRm for vectors of 16-bit data, and places the result in wRd. The four operands from each of the source registers are alternately added and subtracted using a cross selection in each of the parallel operations. The result of the operation is saturated to the signed limits, for example, 0x8000 to 0x7fff or -32768 to 32767.

WAVG4 Performs seven 4-pixel averages of unsigned 8-bit operands obtained from the bytes of the wRn and wRm source registers. Biased rounding is supported through the use of the R-qualifier. The seven 4-pixel averages are packed into the lower seven bytes of the destination register with the most significant byte written with 0x00.

WMERGE Extracts a 64-bit value that contains elements from the two 64-bit source registers (wRn,wRm), and places a merged 64-bit result in the destination register, wRd. The number of adjacent elements from each register is represented with a 3-bit immediate.

WMIAxy Performs a signed parallel 16-bit multiply, or multiply-negate, followed by 64-bit accumulation. The upper or lower 16-bits of each source register half is selected by specifying either the B (bottom) or T (top) in each of the xy positions of the mnemonic. The xy selection is the same for both the upper and lower halves of the 64-bit operand source registers, wRn and wRm. The resulting product for the upper and lower halves are then added to the 64-bit accumulator which occupies the destination register wRd. The N-qualifier optionally provides for a multiply-negate-accumulate operation instead of the default multiply-accumulate.

WMIAWxy Performs multiply-accumulate using signed 32-bit operands from the two source wRm and wRn, and accumulates the result with the destination register, wRd. The upper or lower 32-bits of each source register half is selected by specifying either the B (bottom) or T (top) in each of the xy positions of the mnemonic.The N-qualifier optionally provides for a multiply-negate-accumulate operation instead of the default multiply-accumulate.

WMULW Performs a vector multiplication of wRn and wRm on vectors of 32-bit data only; UM-qualifier indicates that the higher order 32 bits of the result of the unsigned multiplication are to be stored in wRd, the SM-qualifier indicates that the higher order 32 bits of the signed multiplication are to be stored in wRd, and the L-qualifier indicates that the lower 32 bits of the result are to be stored in wRd. Biased rounding is available with the UM and SM options through the use of the R-qualifier.

WQMIAxy Performs a signed fractional parallel 16-bit multiply, or multiply-negate, followed by parallel 32-bit accumulation. The upper or lower 16-bits of each Marvell® Wireless MMX™ 2 coprocessor source register half is selected by specifying either the B (bottom) or T (top) in each of the xy positions of the mnemonic. The xy selection is the same for both the upper and lower halves of the 64-bit operand source registers, wRn and wRm. The resulting product for the upper and lower halves are then added to the two 32-bit accumulators which occupy the upper and lower halves of the 64-bit destination register wRd. A left shift correction is applied following the multiplication and saturation is provided with the addition for 32-bit signed operands. The N-qualifier optionally provides for a multiply-negate-accumulate operation instead of the default multiply-accumulate.

WQMULM Performs a vector multiplication of wRn and wRm on vectors of signed fractional 16-bit Qm.n values. The upper half of the results are stored in wRd with a left shift correction to renormalize the fractional data. Biased rounding, “round to nearest” is supported with the R-qualifier with truncation as the default behavior.

WQMULWM Performs a vector multiplication of wRn and wRm on vectors of signed fractional 32-bit Qm.n values. The upper half of the results are stored in wRd with a left shift correction to renormalize the fractional data. Biased rounding, “round to nearest” is supported with the R-qualifier with truncation as the default behavior.

WSUBADDX Performs complex vector subtraction/addition of wRn and wRm for vectors of 16-bit data, and places the result in wRd. The four operands from each of the source registers are alternately added and subtracted using a cross selection in each of the parallel operations.The result of the operation is saturated to the signed limits, for example, 0x8000 to 0x7fff or -32768 to 32767.

TORVSC Performs “OR” across the fields of the SIMD saturation flags (wCSSF) and sends the result to the ARM* CPSR Overflow, (V), flag; operation can be performed after a byte, half-word or word operation that sets the flags.

1.2.4 Transfer Instruction Mapping

These instructions are mapped onto the coprocessor transfer instructions.

TANDC Performs “AND” across the fields of the SIMD processor status register (PSR) (wCASF) and sends the result to the ARM* CPSR; can be performed after a byte, half-word or word operation that sets the flags.

TBCST Broadcasts a value from the ARM* source register, Rn, or to every SIMD position in the Marvell® Wireless MMX™ 2 coprocessor destination register, wRd; can operate on 8-, 16-, and 32-bit data values.

TEXTRC Extracts 4-/8-/16-bit field specified by the 3-bit immediate field data from the SIMD PSR (wCASF), and transfers to the ARM* CPSR.

TEXTRM Extracts 8-/16-/32-bit field specified by the 3-bit immediate field data from Marvell® Wireless MMX™ 2 coprocessor source register, wRn, and transfers to the specified ARM* core register, Rd.

TINSR Transfers and inserts 8-/16-/32-bit data from ARM* source register, Rn, to the position in Marvell® Wireless MMX™ 2 coprocessor destination register, wRd, specifed by the 3-bit immediate.

TMCR TMCR is a pseudo-instruction that maps onto ARM* MCR instruction, and is provided for convenience; TMCR transfers the contents of the Rn ARM* core registers to a 32-bit wCx Marvell® Wireless MMX™ 2 coprocessor control register.

TMCRR TMCRR is a pseudo-instruction that maps onto an ARM* MCRR instruction and is provided for convenience; TMCRR transfers the contents of the two ARM* registers (RdHi and RdLo) to wRd (the 64-bit Marvell® Wireless MMX™ 2 coprocessor destination register).

TMIA Provides the same functionality as the Intel XScale® microarchitecture MIA instruction; performs multiply-accumulate using signed 32-bit operands from the two source ARM* core registers (Rm,Rs), and accumulates the result with the Marvell® Wireless MMX™ 2 coprocessor destination register, wRd.

TMIAPH Provides the same functionality as the Intel XScale® microarchitecture MIAPH instruction; performs multiply-accumulate using signed 16-bit operands from the two source ARM* core registers (Rm.Rs), and accumulates the result with the Marvell® Wireless MMX™ 2 coprocessor destination register, wRd.

TMIAxy Provides same functionality as Intel XScale® microarchitecture MIAxy instruction; performs a 16-bit multiply-accumulate using two signed operands from the ARM* core registers (Rm,Rs) and accumulates the result with the Marvell® Wireless MMX™ 2 coprocessor destination register; upper or lower 16-bits of each source register is selected by specifying either a B (bottom) or T (qualifier) in each of the xy positions of the mnemonic.

TMOVMSK Transfers the most significant bit of each SIMD field of the Marvell® Wireless MMX™ 2 coprocessor source register (wRn) to the least significant 8, 4, or 2 bits of the specifed ARM* destination register (Rd); this instruction operates on 8-,16-, and 32-bit data values.

TMRC TMRC is a pseudo- instruction that maps onto an ARM* core MRC instruction and is provided for convenience; TMRC transfers the contents of the 32-bit Marvell® Wireless MMX™ 2 coprocessor control register (wCx) to the ARM* core destination register (Rd).

TMRRC TMRRC is a pseudo-instruction that maps onto a standard ARM* MRRC instruction and is provided for convenience; TMRRC transfers the contents of the wRn 64-bit Marvell® Wireless MMX™ 2 coprocessor data register to two ARM* core destination registers (RdHi, RdLo).

TORC Performs “OR” across the fields of the SIMD PSR (wCASF) and sends the result to the ARM* CPSR; operation can be performed after a byte, half-word or word operation that sets the flags.

1.2.5 Other

WACC Performs an unsigned accumulate across the source register (wRn) fields, and writes result to destination register, wRd; operation can be performed on fields of byte, half-word, and word size.

WALIGNI Extracts an 64-bit value from the two 64-bit source registers (wRn,wRm), and places the result in the destination register, wRd; instruction uses a 3-bit immediate value to specify the byte offset of the value to extract.

WALIGNR Extracts a 64-bit value from the two 64-bit source registers (wRn,wRm), and places the result in the destination register, wRd; instruction uses a 3-bit value stored in the specified general-purpose register to specify the byte offset of the value to extract.

WMAC Performs a vector multiplication of wRn and wRm and can accumulate the result with wRd on vectors of 16-bit data only.

WROR Performs vector logical rotate-right of wRn by wRm, or a control register (wCx), or a 5-bit immediate for vectors of 16-, 32-, or 64-bit data and places the result in the destination register, wRd.

1.2.6 Load/Store

WLDR Performs a load from memory into either a data register (wRd) or a control register (wCx); 8-,16-, 32-, and 64-bit data values can be loaded in the data registers and 32-bit values only can be loaded into the control registers; loaded data is zero-extended. The immediate offset addressing mode is provided for data and control register load operations. The register offset addressing mode is provided for 64-bit loads to the data registers. A left shift may be optionally applied to the offset when using the register offset addressing mode.

WSTR Performs a store from the source register (either wRm or wCx) into the memory location whose address is specified by the Rn register and corresponding address modifiers; 8-, 16-, 32-, and 64-bit data values can be stored from the data registers (wRm), and 32-bits only can be stored from the control registers (wCx). The immediate offset addressing mode is provided for both data and control register store operations. The register offset addressing mode is provided for 64-bit store operations from the data registers. A left shift may be optionally applied to the offset when using the register offset addressing mode.

1.2.7 Synthetic Instructions

These instructions are aliases to specific forms of other instructions:

WMOV This pseudo-instruction moves register wRn to register wRd; this instruction is a form of WOR.

WZERO This pseudo-instruction zeros the Marvell® Wireless MMX™ 2 coprocessor destination register, wRd; this instruction is a form of the WANDN instruction.

1.3 Terminology and Conventions

1.3.1 Number Representation

All numbers in this document are base 10, unless designated otherwise. In text and pseudo-code descriptions, hexadecimal numbers have a prefix of 0x and binary numbers have a prefix of 0b. For example, 107 would be represented as 0x6B in hexadecimal and 0b1101011 in binary.

1.3.2 Terminology and Acronyms

ASSP	Application Specific Standard Product
SIMD	Single Instruction Multiple Data
PSR	Processor Status Register.
Reserved	A **reserved** field is a field that may be used by an implementation. If software provides the initial value of a reserved field, this value must be zero. Software should not modify reserved fields or depend on any values in reserved fields.

1.3.3 Symbols Glossary

This section describes the functions and symbols used to describe the operation of each instruction in Section 3, “Marvell® Wireless MMX™ 2 Coprocessor Instruction Set” on page 3-35 (refer to Table 1-1). This document assumes basic symbols need not be described.

Table 1-1. Symbol and Function Glossary

Symbol	Description
>>>	Arithmetic Shift Right
>>	Shift Right
<<	Shift Left
Nibble x	Refers to Nibble x, counting from 0 at the LSB
Byte x	Refers to Byte x, counting from 0 at the LSB
Half x	Refers to Half Word x, counting from 0 at the LSB
Word X	Refers to Word x, counting from 0 at the LSB
rotate_by(X)	Logical right rotation by X bits.
signReplicate(X,Y)	Replicate Sign bit of X across the width (Y) of the specifed field
signExtend(X,Y)	Sign Extend X to the width of the result field
saturate(X,Y,Z)	Saturate x to the maximum or minimum value, with the output result width specified by Z and using Y to determine whether to use the signed or unsigned maximum and minimum values

Table 1-1. Symbol and Function Glossary (Continued)

Symbol	Description
Low_DB_word	Use the lower double word of the result
(Condition)? X: Y	If condition is true then X else Y
abs(X)	The result is the absolute value of X

1.3.4 Data Types

In the context of the ARM* architecture, a word consists of 32 bits. Therefore, these naming conventions apply to the different data types in the Marvell® Wireless MMX™ 2 coprocessor device:

- 8 bits = byte (abbreviation **B**)
- 16 bits = half word (abbreviation **H**)
- 32 bits = word (abbreviation **W**)
- 64 bits = double word (abbreviation **D**)

Note: This naming convention is not the same as the Intel IA-32 naming convention, where a "word" comprises 16-bits. Therefore, be careful when translating between Marvell® Wireless MMX™ 2 coprocessor and both Marvell® Wireless MMX™ 2 technology and SSE. Table 1-2 shows the equivalent data-type naming.

Table 1-2. ARM* to IA-32 Data-Type Name Mapping

Data Size	Marvell® Wireless MMX™ 2 Coprocessor	Marvell® WMMX™ 2 Technology/ SSE
8	Byte	Byte
16	Half-Word	Word
32	Word	Double Word
64	Double Word	Quad Word

1.3.5 Register Names

These register name conventions are used:

- Rn – ARM* primary source register
- Rm – ARM* secondary source register
- Rd – ARM* destination register

Marvell® Wireless MMX™ 2 coprocessor data registers are prefixed by "wR":

- wRn – Marvell® Wireless MMX™ 2 coprocessor primary source register

- wRm – Marvell® Wireless MMX™ 2 coprocessor secondary source register
- wRd – Marvell® Wireless MMX™ 2 coprocessor destination register

Marvell® Wireless MMX™ 2 coprocessor control registers are prefixed by “wC”:

- wCx – Refers to any control register
- wCGRn – Refers to any of the general-purpose control registers

1.3.6 Instruction Qualifiers

- Optional qualifiers to instructions contained in “{“ “}” need not be specified, if not required. For example, WADD <B,H,W>{US,SS}{Cond} wRd, wRn, wRm. In this case US, SS are optional qualifiers; B, H, and W are compulsory.
- Required qualifiers are contained in “<“ “>”. One of the qualifiers must be chosen. For example, for WACC<B,H,W> Rd, wRn the B, H, or W flag must be specified; that is, valid mnemonics are WACCB, WACCH, or WACCW. WACC alone is not valid.

1.3.7 Conditional Execution Specifiers

The condition-code specifiers in the Marvell® Wireless MMX™ 2 coprocessor mnemonics are the same as in the ARM* architecture.

THIS PAGE INTENTIONALLY LEFT BLANK

Programmers Model 2

2.1 ARM* Overview

For an overview of the general programming model on the ARM* architecture, refer to the *Programmers Reference Guide for Architecture ARM V5* and also the Intel XScale® *Microarchitecture Programmers Reference Manual*.

2.2 Coprocessor Model

The ARM* architecture lets you add coprocessors to the main ARM* core. The ARM*architecture also specifies as many as 16 coprocessors in the standard coprocessor space, with each coprocessor having a unique number. The coprocessors typically have their own state in the form of a register file. Coprocessor instructions can be conditionally executed on flags set in the main ARM* status register. Conditional execution (a feature of the ARM* architecture not present in the Marvell® Wireless MMX™ 2 technology/SSE architectures) allows the elimination of some branches.

2.3 Marvell® Wireless MMX™ 2 Coprocessor

The Marvell® Wireless MMX™ 2 coprocessor maps onto two ARM* coprocessors, currently identified as coprocessor 0 and coprocessor 1. The two coprocessors support the number of instructions, and provide elegant support of Marvell® Wireless MMX™ 2 coprocessor data and control registers using standard ARM* coprocessor transfer instructions (MCR/MRC). The coprocessor interface supports both a 32- and 64-bit interface to the core (to support MRC and MRCC class of coprocessor data transfer instructions).

2.3.1 Coprocessor Instructions

The ARM* architecture specifies an interface to coprocessors that is supported by these instruction types:

- Coprocessor data transfers
- Coprocessor data operations
- Coprocessor register load and store

The previous instructions can be executed conditionally, depending on the state of the ARM* flags. The exception to this is are:

- Load and stores to control (wCx) registers
- Shifts or rotate with immediate values
- Load and stores with register offset

A data operation can be performed while transferring data between the ARM* and a coprocessor register file (this is how the Intel XScale® microarchitecture multiply accumulate (MIA) is currently mapped into the ARM* instruction set).

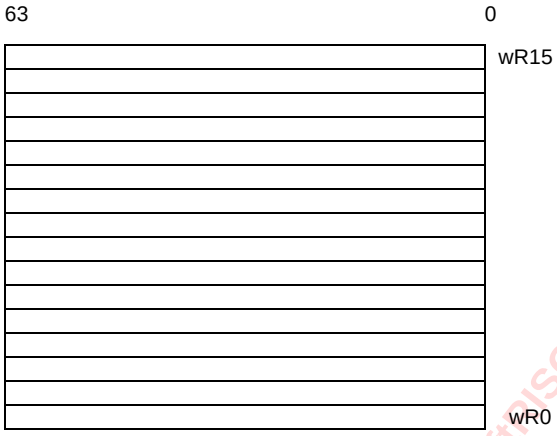
2.3.2 Enabling Marvell® Wireless MMX™ 2 Coprocessor

To use the Marvell® Wireless MMX™ 2 coprocessor the coprocessors need to be enabled in the coprocessor access register which is located in register 15 of coprocessor 15. Both bits 0 and 1 of this register (corresponding to coprocessor 0 and 1) should be set to a 1 to enable access to the Marvell® Wireless MMX™ 2 coprocessor. If the Marvell® Wireless MMX™ 2 coprocessor is not enabled in this way, all Marvell® Wireless MMX™ 2 coprocessor instructions take the undefined instruction trap. See the *Intel XScale® Microarchitecture Megacell Users Guide* for more information about enabling coprocessors.

2.4 Marvell® Wireless MMX™ 2 Coprocessor Register File Organization

The main register file is organized as sixteen 64-bit registers as shown in Figure 2-1.

Figure 2-1. Register File Organization



The registers are located in coprocessor 0. To transfer data between these registers and the ARM* core register file use the TMRCC and TMCRR instructions. If some of the Marvell® Wireless MMX™ 2 coprocessor register is to be transferred, use the TEXTRM and TINSTR instructions to extract and insert sub-fields of the register.

Note: The contents of the data registers wR0 thru wR15 are unknown at reset.

2.5 Marvell® Wireless MMX™ 2 Coprocessor Status and Control Registers

For ARM* coprocessor transfers, the status and control registers of the Marvell® Wireless MMX™ 2 coprocessor unit map into the registers of coprocessor 1, which allows the standard MRC and MCR ARM* instructions to read and write to these registers.

The status registers include SIMD arithmetic flags and saturation flags.

The control registers include:

- Saturation status (wCSSF)
- wCGR0-wCGR3 (32-bit general-purpose registers used for alignment, and shift.)
- Coprocessor ID (wCID)
- Coprocessor Control register (wCon)

Note: When any wCGRn register is used as an argument, the encoding for the register is the coprocessor register number (shown below in Table 2-1), for example, encoding for wCGR0 = 0b1000. Using a non-valid encoding for a wCGRn register results in unpredictable behavior.

2.5.1 Register Map

Table 2-1 shows how the control and status registers map into coprocessor 1 register addresses.

Table 2-1. Status and Control Register Mappings

Mnemonic	Register Name	Coprocessor Register Number
wCID	Coprocessor ID, Revision, Status	0
wCon	Control	1
wCSSF	Saturation SIMD Flags	2
wCASF	Arithmetic SIMD Flags	3
—	Reserved	4
—	Reserved	5
—	Reserved	6
—	Reserved	7
wCGR0	General Purpose register 0	8
wCGR1	General Purpose register 1	9
wCGR2	General Purpose register 2	10
wCGR3	General Purpose register 3	11
—	Reserved	12

Table 2-1. Status and Control Register Mappings

Mnemonic	Register Name	Coprocessor Register Number
—	Reserved	13
—	Reserved	14
—	Reserved	15

2.5.2 Arithmetic Flags

2.5.2.1 SIMD Flags

The wCASF register contains arithmetic flags at each of the SIMD field positions. The information contained in the processor status register (PSR) therefore reflects the last arithmetic operation that set the SIMD arithmetic flags. Table 2-2 shows the overall register format and accessibility for wCASF, the detailed format after each type of data operations is shown later in this section.

Table 2-2. Register Bitmap for wCASF

	Register 3																					wCASF										Coproc 1									
Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0									
	SIMD Flags																																								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0									

Bits	Access	Name	Description
31:0	Read/Write	SIMD Flags	Contains updated SIMD PSR flags following SIMD operation, format is either SIMD8, SIMD16, SIMD32 or SIMD64 as shown below.

After a byte-wide SIMD operation, the PSR would look like that shown in Table 2-3.

Table 2-3. Register Format of wCASF After a Byte Operation – SIMD8 PSR

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	N	Z	C	V	N	Z	C	V	N	Z	C	V	N	Z	C	V	N	Z	C	V	N	Z	C	V	N	Z	C	V	N	Z	C	V

	Name	Description
	N	Result in this byte field was **N**egative
	Z	Result in this byte field was **Z**ero
	C	Result in this byte field has a **C**arry out
	V	Result in this byte field o**V**erflowed

If a wider SIMD field operation is performed, the lower order unused bits are set to zero. Therefore, for a 16-bit SIMD operation, the PSR would contain:

Table 2-4. Register Format for wCASF After a Half-Word Operation – SIMD16 PSR

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	N	Z	C	V	0	0	0	0	N	Z	C	V	0	0	0	0	N	Z	C	V	0	0	0	0	N	Z	C	V	0	0	0	0

	Name	Description
	N	Result in this half word field was **N**egative
	Z	Result in this half word field was **Z**ero
	C	Result in this half word field has a **C**arry out
	V	Result in this half word field o**V**erflowed

For 32-bit SIMD format, this would then become what is shown in Table 2-5:

Table 2-5. Register Format for wCASF After a Word Operation – SIMD32 PSR

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	N	Z	C	V	0	0	0	0	0	0	0	0	0	0	0	0	N	Z	C	V	0	0	0	0	0	0	0	0	0	0	0	0

	Name	Description
	N	Result in this word field was **N**egative
	Z	Result in this word field was **Z**ero
	C	Result in this word field has a **C**arry out
	V	Result in this word field o**V**erflowed

And finally, for a 64-bit operation (Table 2-6):

Table 2-6. Register Bitmap for wCASF After a Double-Word Operation – SIMD64 PSR

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	N	Z	C	V	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	Name	Description
	N	Result in this word field was **N**egative
	Z	Result in this word field was **Z**ero
	C	Result in this word field has a **C**arry out
	V	Result in this word field o**V**erflowed

2.5.2.2 Transferring Flags to the ARM* CPSR

The flags can be transferred to the ARM* status register by performing an AND/OR operation across the SIMD word (See TANDC/TORC), and sending the flags to the ARM* CPSR. Here, subsequent conditional instructions can use them (both ARM* core and coprocessor conditional instructions). Individual fields can also be extracted into ARM* registers using the TEXTRC instruction. Use of TANDC and TORC supports conditional instructions that operate on conditions such as “if *any* SIMD field is zero” (using TORC) and “if *all* SIMD fields are zero” (using TANDC). These new forms of conditions are known as group condition tests.

2.5.2.3 Group Conditions

Here is a list of the new group conditions that can be defined:

- If any field has overflowed
- If any field has not overflowed
- If any field is positive (or zero)
- If any field is negative
- If any field is zero
- If any field is not zero
- If any field has a carry out
- If any field does not have a carry out
- If all fields have overflowed
- If all fields have not overflowed
- If all fields are positive (or zero)
- If all fields are negative
- If all fields are zero
- If all fields are non-zero
- If all fields have a carry out
- If all fields do not have a carry out.

2.5.3 Saturation Flags

Certain instructions allow the results to saturate to the maximum or minimum value rather than overflow (see Section 3.2.19 as an example of an instruction which can saturate). The Saturation register indicates to programmers that saturation has occurred. The saturation flags indicate that saturation occurred at that particular byte, half word, or word position in the 64-bit double word. The saturation bit remains set until explicitly cleared by writing to the register. Additionally, a 16- or 32-bit SIMD operation does not clear the lower unused bits to zero. Instead, the lower unused bits remain at their previous value. The format is as shown in Table 2-7 on page 2-27.

Table 2-7. Register Bitmap for wCSSF – SIMD Saturation Flags

Register 2 wCSSF Coproc 1

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved																								B7	B6	B5	B4	B3	B2	B1	B0
Reset	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	0	0	0	0	0	0	0	0

Bits	Access	Name	Description
31:7	N/A	Reserved	Write as zero, read as unknown.
7	Read/Write	B7	Saturation flag for byte 7 or half word 3 or word 1 or double word 0
6	Read/Write	B6	Saturation flag for byte 6
5	Read/Write	B5	Saturation flag for byte 5 or half word 2
4	Read/Write	B4	Saturation flag for byte 4
3	Read/Write	B3	Saturation flag for byte 3 or half word 1 or word 0
2	Read/Write	B2	Saturation flag for byte 2
1	Read/Write	B1	Saturation flag for byte 1 or half word 0
0	Read/Write	B0	Saturation flag for byte 0

- For byte SIMD operation, bits B0-B7 can be set if saturation occurs.
- For half-word SIMD operation, bits B1, B3, B5, and B7 can be set if saturation occurs.
- For word SIMD operation, bits B3 or B7 may be set.
- For 64-bit operation, only B7 is set on saturation.

Note: The saturation flags remain set until cleared by explicitly writing zeros into the register. The flags are cleared at reset.

2.5.4 General-Purpose Registers

There are four 32-bit, general-purpose registers defined in the Marvell® Wireless MMX™ 2 coprocessor that can store constants for use in memory alignment and shifting. The registers are addressed by the mnemonic wCGR0-wCGR3. Table 2-8 shows the format and accessibility of the general-purpose registers.

Table 2-8. Register Bitmap for wGRn Registers

Registers 8 – 11	wCGRn	Coproc 1

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	GPData																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	Access	Name	Description
31:0	Read/Write	GPData	General-purpose data register

2.5.4.1 Data Transfer

Data can be transferred from the ARM* to and from the general-purpose registers using TMRC and TMCR instructions. Use the general-purpose registers for constants and address offsets. As most of the data to be loaded into these registers comes from the ARM* registers (address offsets, constants), there is no direct path between the wCx registers and the wRx SIMD registers.

2.5.4.2 Memory Alignment

The general-purpose registers can contain the offset from a word boundary for unaligned data being loaded in using a standard ARM*-aligned load. This offset allows the WALIGN instruction to extract the correct 64-bit double word from two aligned double words that have already been loaded. The memory-alignment value is set by extracting the lower order *address* bits from the ARM* base register and transferring these to the Marvell® Wireless MMX™ 2 coprocessor destination general-purpose register.

2.5.4.3 Shift Value

A shift instruction with the G-qualifier specified uses the shift value stored in the general-purpose register (specified in the wRm field).

2.5.5 Coprocessor ID

The coprocessor ID allows a program to identify the coprocessor connected and the revision number of the coprocessor itself. Table 2-9 shows a read-only register.

Table 2-9. Register Bitmap for wCID – Coprocessor ID

	Register 0								wCID								Coproc 1							
	Intel Vendor ID								Architecture Version								Coprocessor Type							
Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	1	0	1	0	0	1	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	Major				Minor			
Reset	0	1	1	0	1	0	0	1	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	Implementation Dependant				implementation Dependant			

Bits	Access	Name	Description
31:24	Read Only	Vendor	Intel® vendor ID 0x69 – Intel ID
23:16	Read Only	Arch	Architecture Version 0x05 – ARM* Architecture V5 Compliant
15:8	Read Only	CPTyper	Coprocessor type 0x20 – Marvell® Wireless MMX™ 2 coprocessor
7:4	Read Only	Major	Major revision number – implementation dependant
3:0	Read Only	Minor	Minor revision number – implementation dependant

Note: Writes to this register are ignored and do not effect the CUP bit in the coprocessor control register.

2.5.6 Coprocessor Control Register

The coprocessor control register allows all user-configuration variables to be set. It also provides a way of detecting that the coprocessor is currently in use (See Table 2-11).

Table 2-10. Register Bitmap for wCon – Coprocessor Control

Register 1	wCon	Coproc 1
Alias		

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved																														MUP	CUP
Reset	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	0	0

Bits	Access	Name	Description
31:2	N/A	Reserved	Reserved for future use – Write as zero, reads as unknown.
1	Read/Write 1 to clear	MUP	One of the main registers (wRn) has been updated 0 = Not updated 1 = Updated
0	Read/Write 1 to clear	CUP	One of the Control Registers (wCx) has been updated. 0 = Not Updated 1 = Updated

The *CUP/MUP* bits are set if the control (wCx)/data registers (wRn) contents have been updated since the last time this bit was cleared. This is typically used by the operating system (OS) to determine whether the registers need saving out to memory on a context switch (if the registers are unchanged then the values already stored in memory can be used instead). Note writing or loading this register does not set the CUP bit.

2.6 Big-Endian and Little-Endian Support

The memory system Endian mode is set by the ARM* microprocessor and is usually implemented as bit 7 in register 1 of coprocessor 15. The data internal to the microprocessor is stored in Little- Endian mode. When data is loaded, an Endian swap occurs on the way into the core subsystem. Therefore, there is no explicit instruction in the Marvell® Wireless MMX™ 2 coprocessor unit for byte swapping. The memory byte order for Little Endian is shown in Table 2-11 and Table 2-12. The Endian ordering for 32-bit words is the same as in the ARM* architecture definition.

Table 2-11. Little-Endian Byte Order

7	6	5	4	3	2	1	0

Table 2-12. Big-Endian Byte Order

0	1	2	3	4	5	6	7

2.7 Qm.n Fixed Point Format

The Marvell® Wireless MMX™ 2 coprocessor provides instruction support for multiply and multiply-accumulate operations for both fractional as well as integer data types. The "Qm.n" format (also known as fixed-point format) provides a standard mechanism for representing fractional values using an integer data type.The integer binary word has been partitioned using an imaginary fixed point. The n-bits to the right of the imaginary point comprise the fractional portion of the value being represented, and these n-bits act as weights for negative powers of 2. The m-bits to the left of the imaginary point comprise the integer portion of the value being represented, and these m-bits act as weights for positive powers of 2. The overall Qm.n representation requires a total of m+n (in case of unsigned numbers) and m+n+1 (in case of signed numbers) bits, with the additional bit required for the sign

This document uses the abbreviated notation "Qn" to denote "Qm.n," where m = (L-n-1) in case of signed numbers and (L-n) in case of unsigned numbers, and L is the word length of the underlying data type in bits. In other words, a particular value for m is implied by the combination of a data type, a particular choice of n specified by "Qn" and the sign. For example, the Q15 fractional representation for 16-bit operands is 1 sign bit and 15 fractional bits, and the Q31 fractional representation for 32-bit operands it is 1 sign bit and 31 fractional bits. The conversions from a signed integer to a signed fraction is accomplished by dividing the signed integer by $2^{(n-1)}$, where n represents the number of bits available. The range represented by the fractional value is from -1 to $+1\text{-}2^{-(n-1)}$.

2.8 Saturating Integer Arithmetic

Conventional arithmetic, also called wraparound arithmetic, can produce partial results. For example, adding or subtracting two large 16-bit numbers may produce a 17-bit result, and adding or subtracting two large 32-bit numbers may produce a 33-bit result. If an addition or subtraction produces a mathematical result which lies outside of the range from -2^{15} (0x8000) to $+2^{15}$-1 (0x7fff) for 16-bit signed operations, or -2^{31} (0x80000000) to $+2^{31}$-1 (0x7fffffff) for 32-bit signed operations, the extra bits are normally discarded. This produces the correct value reduced modulo 2^n as the result, where *n* represents the number of bits available.

In saturating arithmetic, if a result cannot be represented, it is clamped to the largest or smallest possible value for the given data type. The Marvell® Wireless MMX™ 2 coprocessor supports two types of saturating integer arithmetic, signed and unsigned saturation. In signed saturation, the operands and results of the operation are considered to be signed numbers, and in unsigned saturation the operands and the results of the operation are considered unsigned numbers.

Signed Saturation Limits – The largest possible value is $(2^{(n-1)} - 1)$, and the smallest possible value is $(-2^{(n-1)})$, where n is the number of bits available.

Unsigned Saturation Limits –The largest possible value is 2^n, and the smallest possible value is 0.

2.9 Saturating Q15 and Q31 Fractional Arithmetic

The smallest negative and largest positive values which can be represented in the Q15 signed 2's complement fractional format are 0x8000 and 0x7fff respectively. Similarly, for the Q31 format, the maximum and minimum values are 0x80000000 and 0x7fffffff. This corresponds to allowable range of -1 through $(1\text{-}2^{-15})$ for the Q15 format, and -1 through $(1\text{-}2^{-31})$ for the Q31 format. Saturating additions and subtractions can be performed on fractional numbers using the same instructions for saturating integer arithmetic. Multiplication operations should

be performed using the fractional multiplication instructions where possible since the result of an integer multiplication is not quite in the Q15 or Q31 form and the programmer would need to manage the position of the decimal point. For example, when the multipliers multiply two operands in Q15 (1.15) format the result is in Q30 (2.30) format with 2 sign bits and 30 bits of fractional data. The Q30 result needs to be left shifted by 1-bit to remove the redundant sign bit putting it back into the Q31 format.

The left shift correction for the fractional multiplication also needs to handle the special case of a full-scale negative times itself, for example, -1 x -1 = +1. If a Q15 number representing -1 (0x8000) is multiplied by itself using an integer multiply instruction, the value produced is 0x4000000. This value would then need to be doubled to put it into the Q31 form, and saturated to 0x7fffffff which is the closest representation of +1 in Q31 format. The QMULM, QMULWM, and WQMIAxy instruction group supports the left shift correction with saturation for the fractional multiply operation.

2.10 Multiply and Multiply-Accumulate Fractional Modes

The Marvell® Wireless MMX™ 2 coprocessor provides instruction support for 16-bit and 32-bit signed fractional multiplication, and 16-bit signed fractional multiply accumulate operations. When using the fractional multiply and multiply accumulate instructions, an automatic left shift by 1 bit removes the redundant sign bit, and the special case of a full-scale negative times a full-scale negative saturating to a full-scale positive is provided.

The fractional multiply instructions, WQMULM and WQMULWM also provide the option to either truncate the lower 15-bits or round these using biased rounding into the upper bits. The fractional multiply accumulate provides a fractional multiplication with left shift correction and saturation. The WQMIAxy also provides for saturation of the accumulator on bit position 32 in order to conform to the bit exact behavior of the GSM voice codecs.

2.11 Biased Rounding of Multiplier Results

The Marvell® Wireless MMX™ 2 coprocessor provides for biased rounding (or round-to-nearest) as an option when performing either integer or fractional multiplication operations where the most significant 16-bits or 32-bits of a 32-bit or 64-bit result are needed. The rounding operation reduces the precision of a number by removing the lower order bits and possibly modifying the remaining bits to more closely represent the higher precision number. In contrast, when truncation is used the value is always smaller or equal to the original number.

The simplest implementation of the round to nearest technique is to add a constant to the value which is equal to half of the least significant bit of the output, followed by truncation of the lower bits. The WMUL instruction provides for a round to nearest option with the selection of the upper 16-bits of the 32-bit product by adding the constant 0x8000. Similarly the WMULW adds the constant 0x80000000 for rounding the 64-bit result when the upper 32-bits are selected. The fractional multiply instructions involve a left-shift correction, which places the least-significant bit in the output word at bit positions 15 and 31 for the WQMULM and WQMULWM instructions respectively. The constants, 0x4000 and 0x40000000, equal to one half of the value of the least-significant bit position in these cases are also added prior to truncation when the rounding option is selected.

The round-to-nearest method rounds the value to the nearest value that can be represented by the reduced precision format. However, when numbers lie exactly midway between the two nearest reduced precision output values, the method always rounds up to the larger of the two. Because this method always rounds up, it is also known as biased rounding.

2.12 Cross (X) Selection of Operands

The Marvell® Wireless MMX™ 2 coprocessor provides for cross selection of operands as an option when performing signed/unsigned integer or unsigned fractional multiplication operations with the WMADDX instruction and also signed addition/subtraction with the WADDSUBHX and WSUBADDHX instructions.

2.13 Instruction Execution

This section describes overall rules for instruction execution. This includes conditional execution and execution of an invalid instruction.

2.13.1 Conditional Execution

The ARM* architecture specifies that instructions can be conditionally executed. The Marvell® Wireless MMX™ 2 coprocessor instructions that can be conditionally executed (all but two) have a "{Cond}" qualifier shown in the usage field of each instruction. This indicates that the instruction is executed if the main ARM* arithmetic flags in the CPSR match the condition specified. The conditional execution mechanism is defined in detail in the ARM* V5 architecture specification.

The conditional part of the operation is not shown explicitly in the operation of each instruction in this document but it is the same as defined in the ARM* V5 specification.

Note: A condition code of "1111b" is used to encode coprocessor 2 space as per ARM* V5 specification, not the NV condition.

2.13.2 Reserved Instruction Fields

An invalid instruction is defined as an instruction that contains any field encoding that is indicated as reserved in this document. Invalid instructions either take an undefined instruction exception or execute without taking the exception and the result is defined to be unpredictable. If an instruction encoding is marked as “not valid for this instruction”, then using this encoding may decode to another instruction. Table 2-13 describes the exact definition of each type.

Table 2-13. Effect of Reserved fields on Execution

Type	Condition
Unpredictable Result	The result of the operation is defined to be unpredictable if the instruction decodes to one of instruction types defined in this document but one or more of the qualifier fields decode to a reserved value. The instruction still executes, but the results are not defined. No exception is caused.
Undefined Exception	An undefined exception results if the instruction does not decode to one of the instruction types defined in this document or the coprocessor number does not match those of the Marvell® Wireless MMX™ 2 coprocessor.

Marvell® Wireless MMX™ 2 Coprocessor Instruction Set 3

This chapter describes the Marvell® Wireless MMX™ 2 coprocessor instruction set in alphabetical order.

3.1 Instruction Summary

- Section 3.2.1, “TANDC”
- Section 3.2.2, “TBCST”
- Section 3.2.3, “TEXTRC”
- Section 3.2.4, “TEXTRM”
- Section 3.2.5, “TINSR”
- Section 3.2.6, “TMCR”
- Section 3.2.7, “TMCRR”
- Section 3.2.8, “TMIA”
- Section 3.2.9, “TMIAPH”
- Section 3.2.10, “TMIAxy”
- Section 3.2.11, “TMOVMSK”
- Section 3.2.12, “TMRC”
- Section 3.2.13, “TMRRC”
- Section 3.2.14, “TORC”
- Section 3.2.16, “WABS”
- Section 3.2.17, “WABSDIFF”
- Section 3.2.18, “WACC”
- Section 3.2.19, “WADD”
- Section 3.2.20, “WADDSUBHX”
- Section 3.2.21, “WALIGNI”
- Section 3.2.22, “WALIGNR”
- Section 3.2.23, “WAND”
- Section 3.2.24, “WANDN”
- Section 3.2.25, “WAVG2”
- Section 3.2.26, “WAVG4”

- Section 3.2.27, “WCMPEQ”
- Section 3.2.28, “WCMPGT”
- Section 3.2.29, “WLDR”
- Section 3.2.30, “WMAC”
- Section 3.2.31, “WMADD”
- Section 3.2.32, “WMAX”
- Section 3.2.33, “WMERGE”
- Section 3.2.34, “WMIAxy”
- Section 3.2.35, “WMIAWxy”
- Section 3.2.36, “WMIN”
- Section 3.2.37, “WMOV”
- Section 3.2.38, “WMUL”
- Section 3.2.39, “WMULW”
- Section 3.2.40, “WOR”
- Section 3.2.41, “WPACK”
- Section 3.2.42, “WQMIAxy”
- Section 3.2.43, “WQMULM”
- Section 3.2.44, “WQMULWM”
- Section 3.2.45, “WROR”
- Section 3.2.46, “WSAD”
- Section 3.2.47, “WSHUFH”
- Section 3.2.48, “WSLL”
- Section 3.2.49, “WSRA”
- Section 3.2.50, “WSRL”
- Section 3.2.51, “WSTR”
- Section 3.2.52, “WSUB”
- Section 3.2.53, “WSUBADDHX”
- Section 3.2.54, “WUNPCKEH”
- Section 3.2.55, “WUNPCKIH”
- Section 3.2.56, “WUNPCKEL”
- Section 3.2.57, “WUNPCKIL”
- Section 3.2.58, “WXOR”
- Section 3.2.59, “WZERO”

3.2 Marvell® Wireless MMX™ 2 Coprocessor Instruction Set Details

3.2.1 TANDC

Overview Performs "AND" across the fields of the SIMD processor status register (PSR) (wCASF) and sends the result to the ARM* CPSR; can be performed after a byte, half-word or word operation that sets the flags.

Usage TANDC<B,H,W>{Cond} R15

Qualifiers
B – Transfer flags after a byte operation
H – Transfer flags after a half-word operation
W – Transfer flags after a word operation

Operation

```
if (B Specified) then
   CPSR[31:28] = wCASF[31:28] & wCASF[27:24] & wCASF[23:20] & wCASF[19:16]
    & wCASF[15:12] & wCASF[11:8] & wCASF[7:4] & wCASF[3:0];

if(H Specified) then
   CPSR[31:28] = wCASF[31:28] & wCASF[23:20] & wCASF[15:12] & wCASF[7:4];

else if (W Specified)
   CPSR[31:28] = wCASF[31:28] & wCASF[15:12];
```

Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				ww0			1	0011				1111				0001				001			1	0000			

Sub-Field Encoding

Qualifier	Field	Value
B	ww	00
H	ww	01
W	ww	10
Reserved	ww	11

SIMD Flags Does not effect the SIMD PSR flags (wCASF) or saturation flags (wCSSF)

Note: Specifying any destination register other than R15 causes a undefined instruction exception.

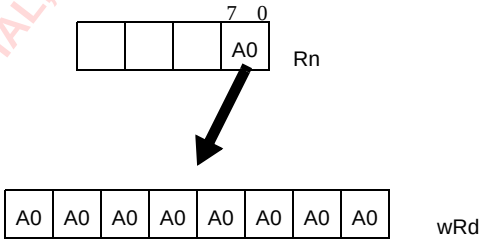
3.2.2 TBCST

Overview Broadcasts a value from the ARM* source register, Rn, or to every SIMD position in the Marvell® Wireless MMX™ 2 coprocessor destination register, wRd; can operate on 8-, 16-, and 32-bit data values.

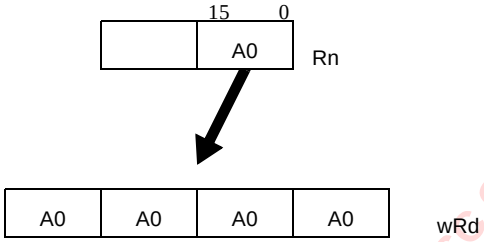
Usage TBCST<B,H,W>{Cond} wRd, Rn

Qualifiers
B – 8-bit (byte) SIMD
H – 16-bit (half word) SIMD
W – 32-bit (word) SIMD

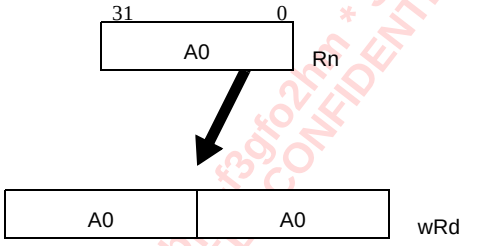
Operation TBCSTB wRd,Rn;



TBCSTH wRd,Rn ;



TBCSTW wRd,Rn



Exceptions None

Encoding

TBCST

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				010			0	wRd				Rn				0000				ww0			1	0000			

Qualifier	Field	Value
B	ww	00
H	ww	01
W	ww	10
Reserved	ww	11

SIMD Flags Does not effect the SIMD PSR flags (wCASF) or saturation flags (wCSSF)

3.2.3 TEXTRC

Overview Extracts 4-/8-/16-bit field specified by the 3-bit immediate field data from the SIMD PSR (wCASF), and transfers to the ARM* CPSR.

Usage TEXTRC<B,H,W>{Cond} R15, #Imm3

Qualifiers
B – Transfer flags after a byte operation
H – Transfer flags after a half-word operation
W – Transfer flags after a word operation
Imm – Specifies the field to transfer flags from

Operation

```
if (B Specified) then
    CPSR[31:28] = wCASF[Nibble[#Imm3[2:0]]];
else if (H Specified) then
    CPSR[31:28] = wCASF[Nibble[#Imm3[1:0],1]];
else if (W Specified)
    CPSR[31:28] = wCASF[Nibble[#Imm3[0],1,1]];
```

IMM3 Values to get the corresponding flag fields

	31							0	
B qualifier	111	110	101	100	011	010	001	000	wCASF - SIMD8 PSR

H qualifier	-11	-10	-01	-00	wCASF - SIMD16 PSR

W qualifier	--1	--0	wCASF - SIMD32 PSR

Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				ww0			1	0011				1111				0001				011			1	0bbb			

Sub-Field Encoding

Qualifier	Field	Value
B	ww	00
H	ww	01
W	ww	10
Reserved	ww	11
Imm3	bbb	3-Bit Offset

Note: The upper 2-bits of Imm3 are not used when W-qualifier is specified, and the upper 1-bit is not used when the H-qualifier is specified

SIMD Flags Does not effect the SIMD PSR flags (wCASF) or saturation flags (wCSSF)

Note: Imm value specifies which nibble/byte/half to extract; Specifying any destination register other than R15 causes a undefined instruction exception.

3.2.4 TEXTRM

Overview Extracts 8-/16-/32-bit field specified by the 3-bit immediate field data from Marvell® Wireless MMX™ 2 coprocessor source register, wRn, and transfers to the specified ARM* core register, Rd.

Usage TEXTRM<U,S><B,H,W>{Cond} Rd, wRn, #Imm3

Qualifiers
B – Extract 8 -bits (byte)
H – Extract 16-bits (half word)
W – Extract 32-bits (word)
S – Sign extend
U – Unsigned
Imm – Specifies which byte/half/word to extract

Operation

```
if (B Specified) then
    Rd[7:0] = wRn[Byte #Imm3[2:0]];
    Rd[31:8]= (S Specified) ? signReplicate(wRn[Byte #Imm3[2:0]],24): 0;
else if (H Specified)
   Rd[15:0] = wRn[Half #Imm3[1:0]];
    Rd[31:16]=(S Specified) ?signReplicate(wRn[Half #Imm3[1:0]],16): 0;
else if (W Specified) then
   Rd[31:0] = wRn[Word #Imm3[0]];
```

Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				ww0			1	wRn				Rd				0000				011			1	sbbb			

Sub-Field Encoding

Qualifier	Field	Value
B	ww	00
H	ww	01
W	ww	10
Reserved	ww	11
U	s	0
S	s	1
Imm3	bbb	3-Bit Offset

Note: The upper 2-bits of Imm3 are not used when W-qualifier is specified, and the upper 1-bit is not used when the H-qualifier is specified

SIMD Flags Does not effect the SIMD PSR flags (wCASF) or saturation flags (wCSSF)

Note: Imm value specifies which byte/half/word to extract; specifying R15 as the destination register produces an unpredictable result.

3.2.5 TINSR

Overview Transfers and inserts 8-/16-/32-bit data from ARM* source register, Rn, to the position in Marvell® Wireless MMX™ 2 coprocessor destination register, wRd, specifed by the 3-bit immediate.

Usage TINSR<B,H,W>{Cond} wRd, Rn, #Imm3

Qualifiers
B – Insert 8 -bits (byte)
H – Insert 16-bits (half word)
W – Insert 32-bits (word)
Imm – Specifes which byte/half/word to insert

Operation

```
if (B Specified) then
   wRd[Byte #Imm3[2:0]] = Rn[7:0];
else if (H Specified)
   wRd[Half #Imm3[1:0]] = Rn[15:0];
else if (W Specified) then
   wRd[Word #Imm3[0]] = Rn[31:0];
```

Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				011			0	wRd				Rn				0000				ww0			1	0bbb			

Sub-Field Encoding

Qualifier	Field	Value
B	ww	00
H	ww	01
W	ww	10
Reserved	ww	11
Imm3	bbb	3-Bit Offset

Note: The upper 2-bits of Imm3 are not used when W-qualifier is specified, and the upper 1-bit is not used when the H-qualifier is specified

SIMD Flags Does not effect the SIMD PSR flags (wCASF) or saturation flags (wCSSF).

Note: Imm value specifies which byte/half/word to insert; remaining fields of the destination register are unchanged. Specifying R15 as the source register produces an unpredictable result

3.2.6 TMCR

Overview TMCR is a pseudo-instruction that maps onto ARM* MCR instruction, and is provided for convenience; TMCR transfers the contents of the Rn ARM* core registers to a 32-bit wCx Marvell® Wireless MMX™ 2 coprocessor control register.

Usage TMCR{Cond} wCx, Rn

Qualifiers None

Operation wCx[31:0]=Rn;

Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				000			0	wCx				Rn				0001				000			1	0000			

Sub-Field Encoding
None

SIMD Flags Does not effect the SIMD PSR flags (wCASF) or saturation flags (wCSSF), however wCASF or wCSSF is updated if it is specified as the destination register (wCx).

Note: Used with Marvell® Wireless MMX™ 2 coprocessor control register; for data registers, use TEXTRM, TINSR, TMRRC and TMCRR instead. Specifying an illegal value for wCx causes an undefined instruction exception. Specifying R15 as the source register produces an unpredictable result.

3.2.7 TMCRR

Overview TMCRR is a pseudo-instruction that maps onto an ARM* MCRR instruction and is provided for convenience; TMCRR transfers the contents of the two ARM* registers (RdHi and RdLo) to wRd (the 64-bit Marvell® Wireless MMX™ 2 coprocessor destination register).

Usage TMCRR{Cond} wRd, RdLo, RdHi

Qualifiers None

Operation

```
wRd[63:32]=RdHi;
wRd[31:0] =RdLo;
```

Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				11000100								RdHi				RdLo				0000				0000				wRd			

Sub-Field Encoding
None

SIMD Flags Does not effect the SIMD PSR flags (wCASF) or saturation flags (wCSSF)

Note: Can only be used with Marvell® Wireless MMX™ 2 coprocessor data registers; similar functionality as provided by the Intel XScale® MAR instruction (note that the core MAR transfers a 40-bit accumulator whereas Marvell® Wireless MMX™ 2 coprocessor provides a full 64-bit register transfer); specifying R15 as the source register produces an unpredictable result

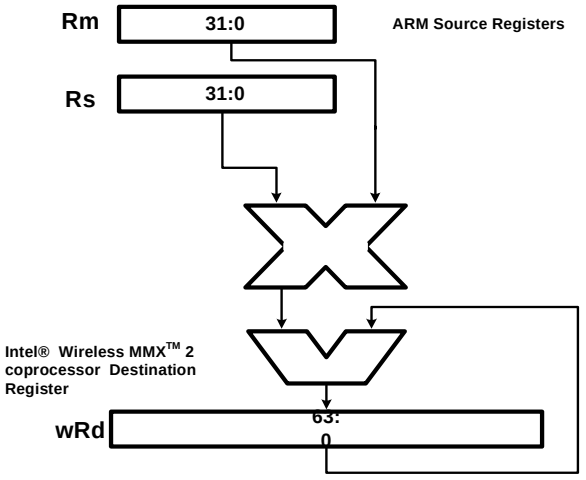
3.2.8 TMIA

Overview Provides the same functionality as the Intel XScale® microarchitecture MIA instruction; performs multiply-accumulate using signed 32-bit operands from the two source ARM* core registers (Rm,Rs), and accumulates the result with the Marvell® Wireless MMX™ 2 coprocessor destination register, wRd.

Usage TMIA {Cond} wRd, Rm, Rs

Qualifiers None

Operation wRd = (Rm[31:0] * Rs[31:0]) + wRd[64:0];



Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				001			0	0000				Rs				000			wRd				1	Rm			

Sub-Field Encoding None

SIMD Flags Does not effect the SIMD PSR flags (wCASF) or saturation flags (wCSSF)

Note: Can also use mnemonic without leading T (that is, MIA); specifying R15 as Rs or Rm produces an unpredictable result; provides same functionality as provided by the Intel XScale® core MIA instruction. (note that the core MIA calculates a 40-bit result whereas Marvell® Wireless MMX™ 2 coprocessor provides a full 64-bit register result; core and Marvell® Wireless MMX™ 2 coprocessor therefore produces different results if the result overflows 40-bits—the core MIA overflows whereas Marvell® Wireless MMX™ 2 coprocessor TMIA produces the correct results until a 64-bit overflow)

3.2.9 TMIAPH

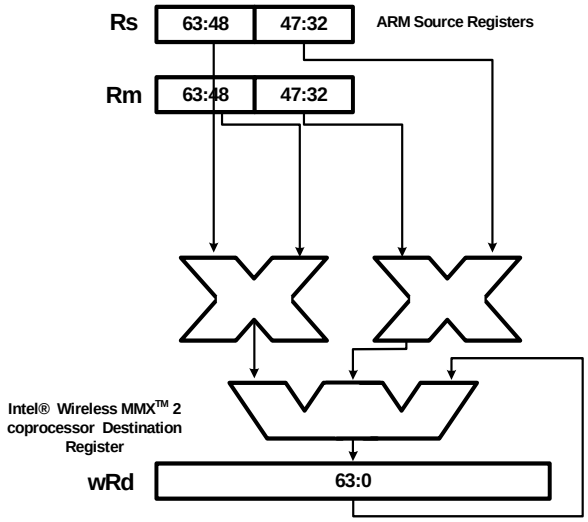
Overview Provides the same functionality as the Intel XScale® microarchitecture MIAPH instruction; performs multiply-accumulate using signed 16-bit operands from the two source ARM* core registers (Rm.Rs), and accumulates the result with the Marvell® Wireless MMX™ 2 coprocessor destination register, wRd.

Usage TMIAPH{Cond} wRd, Rm, Rs

Qualifiers None

Operation

wRd = signExtend((Rm[31:16]*Rs[31:16])+(Rm[15:0]*Rs[15:0])) +wRd;



Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				001			0	1000				Rs				000			wRd				1	Rm			

Sub-Field Encoding

SIMD Flags Does not effect the SIMD PSR flags (wCASF) or saturation flags (wCSSF)

Note: Can also use mnemonic without leading T (that is, MIAPH); specifying R15 as Rs or Rm produces an unpredictable result; provides same functionality as that from the Intel XScale® core MIAPH instruction (note that the core MIAPH calculates a 40-bit result whereas Marvell® Wireless MMX™ 2 coprocessor provides a full 64-bit register result; core and Marvell®

Wireless MMX™ 2 coprocessor therefore produces different results if the result overflows 40-bits—core MIAPH overflows whereas Marvell® Wireless MMX™ 2 coprocessor TMIAPH produces the correct results until a 64-bit overflow)

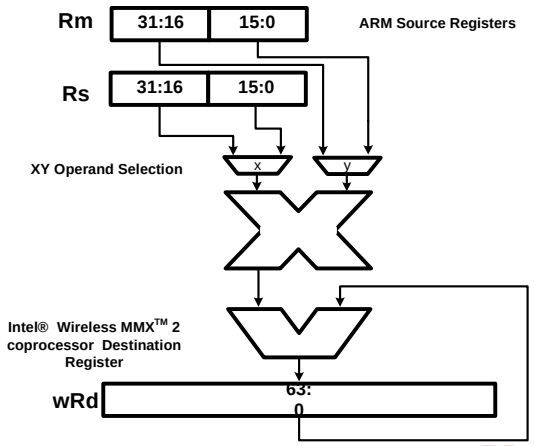
3.2.10 TMIAxy

Overview Provides same functionality as Intel XScale® microarchitecture MIAxy instruction; performs a 16-bit multiply-accumulate using two signed operands from the ARM* core registers (Rm,Rs) and accumulates the result with the Marvell® Wireless MMX™ 2 coprocessor destination register; upper or lower 16-bits of each source register is selected by specifying either a B (bottom) or T (qualifier) in each of the xy positions of the mnemonic.

Usage TMIA<T,B><T,B>{Cond} wRd, Rm, Rs

Qualifiers B – Use bottom field
T – Use top field

Operation
```
<operand1> = (T Specified in x position)>? Rm[31:16] : Rm[15:0];
<operand2> = (T Specified in y position)? Rs[31:16] : Rs[15:0];
wRd[63:0] = signExtend((<operand1>*<operand2>),64) + wRd[63:0];
```



Exceptions None

Encoding TBD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				001			0	11xy				Rs				000			wRd				1	Rm			

Sub-Field Encoding

Qualifier	Field	Value
T	x	1
B	x	0
T	y	1
B	y	0

SIMD Flags Does not effect the SIMD PSR flags (wCASF) or saturation flags (wCSSF)

Note: Can also use mnemonic without leading T (that is, MIAxy); specifying R15 as Rs or Rm produces an unpredictable result; provides same functionality as provided by the Intel XScale® core MIAxy instruction (note that the core MIAxy calculates a 40-bit result whereas Marvell® Wireless MMX™ 2 coprocessor provides a full 64-bit register result; core and Marvell® Wireless MMX™ 2 coprocessor therefore produces different results if the result overflows 40-bits—core MIAxy overflows whereas Marvell® Wireless MMX™ 2 coprocessor TMIAxy produces the correct results until a 64-bit overflow)

3.2.11 TMOVMSK

Overview Transfers the most significant bit of each SIMD field of the Marvell® Wireless MMX™ 2 coprocessor source register (wRn) to the least significant 8, 4, or 2 bits of the specifed ARM* destination register (Rd); this instruction operates on 8-,16-, and 32-bit data values.

Usage TMOVMSK<B,H,W>{Cond} Rd, wRn

Qualifiers
B – 8-bit (byte) SIMD
H – 16-bit (half word) SIMD
W – 32-bit (word) SIMD

Operation

```
if (B Specified) then
   Rd[7]=wRn[63]; Rd[6]=wRn[55]; Rd[5]=wRn[47]; Rd[4]=wRn[39];
   Rd[3]=wRn[31]; Rd[2]=wRn[23]; Rd[1]=wRn[15]; Rd[0]=wRn[7];
   Rd[31:8] =0;
 else if (H Specified) then
   Rd[3]=wRn[63]; Rd[2]=wRn[47]; Rd[1]=wRn[31]; Rd[0]=wRn[15];
   Rd[31:4] =0;
 else if (W Specified) then
   Rd[1]=wRn[63]; Rd[0]=wRn[31]; Rd[31:2]=0;
```

Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				ww0			1	wRn				Rd				0000				001			1	0000			

Sub-Field Encoding

Qualifier	Field	Value
B	ww	00
H	ww	01
W	ww	10
Reserved	ww	11

SIMD Flags Does not effect the SIMD PSR flags (wCASF) or saturation flags (wCSSF)

Note: Specifying R15 as the destination register produces an unpredictable result

3.2.12 TMRC

Overview TMRC is a pseudo- instruction that maps onto an ARM* core MRC instruction and is provided for convenience; TMRC transfers the contents of the 32-bit Marvell® Wireless MMX™ 2 coprocessor control register (wCx) to the ARM* core destination register (Rd).

Usage TMRC{Cond}Rd, wCx

Qualifiers None

Operation
```
Rd = wCx[31:0];
```

Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				000			1	wCx				Rd				0001				000			1	0000			

Sub-Field Encoding
None

SIMD Flags Does not effect the SIMD PSR flags (wCASF) or saturation flags (wCSSF)

Note: Can only be used with Marvell® Wireless MMX™ 2 coprocessor control registers; for data registers, use TEXTRM, TINSR, TMRRC and TMCRR instead; specifying R15 as the destination register produces an unpredictable result. Specifying an illegal value for wCx causes an undefined instruction exception.

3.2.13 TMRRC

Overview TMRRC is a pseudo-instruction that maps onto a standard ARM* MRRC instruction and is provided for convenience; TMRRC transfers the contents of the wRn 64-bit Marvell® Wireless MMX™ 2 coprocessor data register to two ARM* core destination registers (RdHi, RdLo).

Usage TMRRC{Cond} RdLo, RdHi, wRn

Qualifiers None

Operation

```
RdHi=wRn[63:32];
RdLo=wRn[31:0];
```

Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				11000101								RdHi				RdLo				0000				0000				wRn			

Sub-Field Encoding
None

SIMD Flags Does not effect the SIMD PSR flags (wCASF) or saturation flags (wCSSF).

Note: Can only be used with Marvell® Wireless MMX™ 2 coprocessor data registers; results of this instruction are unpredictable if RdHi=RdLo. Specifying R15 as one of the destination registers (RdHi,RdLo) also produces an unpredictable result. Similar functionality as provided by the Intel XScale® core MRA instruction (note that the core MRA transfers a 40-bit accumulator whereas Marvell® Wireless MMX™ 2 coprocessor provides a full 64-bit register transfer)

3.2.14 TORC

Overview Performs “OR” across the fields of the SIMD PSR (wCASF) and sends the result to the ARM* CPSR; operation can be performed after a byte, half-word or word operation that sets the flags.

Usage TORC<B,H,W>{Cond} R15

Qualifiers
B – Transfer flags after a byte operation
H – Transfer flags after a half-word operation
W – Transfer flags after a word operation

Operation

```
if (B Specified) then
   CPSR[31:28] = wCASF[31:28] | wCASF[27:24] | wCASF[23:20] | wCASF[19:16]
    | wCASF[15:12] | wCASF[11:8] | wCASF[7:4] | wCASF[3:0];

 if(H Specified) then
   CPSR[31:28] = wCASF[31:28] | wCASF[23:20] | wCASF[15:12] | wCASF[7:4];

 else if (W Specified)
```

CPSR[31:28] = wCASF[31:28] | wCASF[15:12];

Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				ww0			1	0011				1111				0001				010			1	0000			

Sub-Field Encoding

Qualifier	Field	Value
B	ww	00
H	ww	01
W	ww	10
Reserved	ww	11

SIMD Flags Does not effect the SIMD PSR flags (wCASF) or saturation flags (wCSSF)

Note: Specifying any destination register other than R15 causes a undefined instruction exception.

Uses:

3.2.15 TORVSC

Overview Performs “OR” across the fields of the SIMD saturation flags (wCSSF) and sends the result to the ARM* CPSR Overflow, (V), flag; operation can be performed after a byte, half-word or word operation that sets the flags.

Usage TORVSC<B,H,W>{Cond} R15

Qualifiers
B – Transfer flag after a byte operation
H – Transfer flag after a half-word operation
W – Transfer flag after a word operation

Operation

```
if (B Specified) then{
   CPSR[31:29] = 0;

   CPSR[28] = wCSSF[7] | wCSSF[6] | wCSSF[5] | wCSSF[4] | wCSSF[3] | wCSSF[2] | wCSSF[1]
    | wCSSF[0];
}
else if (H Specified) then {
   CPSR[31:29] =0;
   CPSR[28] = wCSSF[7] | wCSSF[5] | wCSSF[3] | wCSSF[1] ;
}
else if (W Specified){
   CPSR[31:29] = 0;
```

```
    CPSR[28] = wCSSF[7] | wCSSF[3];
}
```

Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				ww0			1	0010				1111				0001				100			1	0000			

Sub-Field Encoding

Qualifier	Field	Value
B	ww	00
H	ww	01
W	ww	10
Reserved	ww	11

SIMD Flags Does not effect the SIMD PSR flags (wCASF) or saturation flags (wCSSF)

Note: Specifying any destination register other than R15 causes a undefined instruction exception.

3.2.16 WABS

Overview Performs a vector absolute value of wRn on 8-bit, 16-bit, or 32-bit signed data and places the result into the destination register, wRd.

Usage WABS<B,H, W>{Cond} wRd, wRn

Qualifiers
B – 8-bit (byte) SIMD
H – 16-bit (half word) SIMD
W – 32-bit (word) SIMD

Operation

```
if (B Specified) then {
   wRd[byte 7]=abs(wRn[byte 7]);
   wRd[byte 6]=abs(wRn[byte 6]);
   wRd[byte 5]=abs(wRn[byte 5]);
   wRd[byte 4]=abs(wRn[byte 4]);
   wRd[byte 3]=abs(wRn[byte 3]);
   wRd[byte 2]=abs(wRn[byte 2]);
   wRd[byte 1]=abs(wRn[byte 1]);
   wRd[byte 0]=abs(wRn[byte 0]);
}
else if (H Specified) then {
   wRd[half 0]=abs(wRn[half 0]);
   wRd[half 1]=abs(wRn[half 1]);
   wRd[half 2]=abs(wRn[half 2]);
```

```
  wRd[half 3]=abs(wRn[half 3]);
}
else if (W Specified) then {
  wRd[word 0]=abs(wRn[word 0]);
  wRd[word 1]=abs(wRn[word 1]);
}
```

Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				ww10				wRn				wRd				0001				110			0	0000			

Sub-Field Encoding

Qualifier	Field	Value
B	ww	00
H	ww	01
W	ww	10
Reserved	ww	11

SIMD Flags Clears N, C, V, and Z and sets Z *if* field zero. Does not effect the saturation flags (wCSSF)

Flag	Action
N	Cleared
Z	Set on Final Result
C	Cleared
V	Cleared

Note: The full-scale negative 0x80, 0x8000, and 0x80000000 result in a full-scale positive.

3.2.17 WABSDIFF

Overview Performs a vector absolute value of the differences of wRm and wRn for vectors 8-bit, 16-bit, or 32-bit unsigned data, and places the result in wRd.

Usage WABSDIFF <B,H, W>{Cond} wRd, wRn, wRm

Qualifiers
B – 8-bit (byte) SIMD
H – 16-bit (half word) SIMD

W – 32-bit (word) SIMD

Operation

```
if (B Specified) then{
  wRd[byte 7]=abs(wRn[byte 7] – wRm[byte 7]);
  wRd[byte 6]=abs(wRn[byte 6] – wRm[byte 6]);
  wRd[byte 5]=abs(wRn[byte 5] – wRm[byte 5]);
  wRd[byte 4]=abs(wRn[byte 4] – wRm[byte 4]);
  wRd[byte 3]=abs(wRn[byte 3] – wRm[byte 3]);
  wRd[byte 2]=abs(wRn[byte 2] – wRm[byte 2]);
  wRd[byte 1]=abs(wRn[byte 1] – wRm[byte 1]);
  wRd[byte 0]=abs(wRn[byte 0] – wRm[byte 0]);
}
else if (H Specified) then{
  wRd[half 0]=abs(wRn[half 0] – wRm[half 0]);
  wRd[half 1]=abs(wRn[half 1] – wRm[half 1]);
  wRd[half 2]=abs(wRn[half 2] – wRm[half 2]);
  wRd[half 3]=abs(wRn[half 3] – wRm[half 3]);
}
else if (W Specified) then{
  wRd[word0]=abs(wRn[word0] – wRm[word 0]);
  wRd[word1] =abs(wRn[word 1] – wRm[word 1]);
}
```

Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				ww01				wRn				wRd				0001				110			0	wRm			

Sub-Field Encoding

Qualifier	Field	Value
B	ww	00
H	ww	01
W	ww	10

Note: The combination ww= 11 is not valid for this instruction.

SIMD Flags Clears N, C, V,and Z and sets Z *if* field zero. Does not effect the SIMD saturation flags (wCSSF)

Flag	Action
N	Cleared
Z	Set on Final Result
C	Cleared
V	Cleared

Note: The intermediate value for the absolute difference calculation are 9-bit, 17-bit, and 33-bit for B, H, and W qualifiers respectively.

3.2.18 WACC

Overview Performs an unsigned accumulate across the source register (wRn) fields, and writes result to destination register, wRd; operation can be performed on fields of byte, half-word, and word size.

Usage WACC<B,H,W>{Cond} wRd, wRn

Qualifiers
B – 8-bit (byte) SIMD
H – 16-bit (half word) SIMD
W – 32-bit (word) SIMD

Operation

```
if (B Specified) then
   wRd = wRn[63:56] + wRn[55:48] + wRn[47:40]wRn[39:32]
    + wRn[31:24] + wRn[23:16] + wRn[15:8] +wRn[7:0];
else if (H Specified)
   wRd = wRn[63:48] + wRn[47:32] + wRn[31:16] + wRn[15:0];
else if (W Specified) then
   wRd = wRn[63:32] + wRn[31:0];
```

Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				ww00				wRn				wRd				0001				110			0	0000			

Sub-Field Encoding

Qualifier	Field	Value
B	ww	00
H	ww	01
W	ww	10
Reserved	ww	11

SIMD Flags Does not effect the SIMD PSR flags (wCASF) or saturation flags (wCSSF)

3.2.19 WADD

Overview Performs vector addition of wRn and wRm for vectors of 8-, 16-, or 32-bit signed or unsigned data, and places the result in wRd. saturation can be specified as signed, unsigned, or no saturation. The use of the carry flags from the wCASF register may be optionally supplied as the

Carry-in to the addition operation using the C-qualifier. The add with carry-in option is valid for 16 and 32-bit operands only.

Usage WADD<B,H,W>{US,SS,C}{Cond} wRd, wRn, wRm

Qualifiers
B – 8-bit (byte) SIMD
H – 16-bit (half word) SIMD
W – 32-bit (word) SIMD
C – Carry in from SIMD PSR used as input
SS – Signed saturation
US – Unsigned saturation

Operation

```
if(C Specified) then{
  if (H Specified)then{
    wRd[half 3] = wRn[half 3] + wRm[half 3] + wCASF[29];
    wRd[half 2] = wRn[half 2] + wRm[half 2] + wCASF[21];
    wRd[half 1] = wRn[half 1] + wRm[half 1] + wCASF[13];
    wRd[half 0] = wRn[half 0] + wRm[half 0] + wCASF[5];
  }
  else if (W Specified) then{
    wRd[word 1] = wRn[word 1] + wRm[word 1] + wCASF[29];
    wRd[word 0] = wRn[word 0] + wRm[word 0] + wCASF[13];
  }
}
else if (B Specified) then {
  wRd[byte 7] = saturate(wRn[byte 7] + wRm[byte 7], {US,SS}, 8);
  wRd[byte 6] = saturate(wRn[byte 6] + wRm[byte 6], {US,SS}, 8);
  wRd[byte 5] = saturate(wRn[byte 5] + wRm[byte 5], {US,SS}, 8);
  wRd[byte 4] = saturate(wRn[byte 4] + wRm[byte 4], {US,SS}, 8);
  wRd[byte 3] = saturate(wRn[byte 3] + wRm[byte 3], {US,SS}, 8);
  wRd[byte 2] = saturate(wRn[byte 2] + wRm[byte 2], {US,SS}, 8);
  wRd[byte 1] = saturate(wRn[byte 1] + wRm[byte 1], {US,SS}, 8);
  wRd[byte 0] = saturate(wRn[byte 0] + wRm[byte 0], {US,SS}, 8);
}
else if (H Specified) then {
  wRd[half 3] = saturate(wRn[half 3] + wRm[half 3], {US,SS}, 16);
  wRd[half 2] = saturate(wRn[half 2] + wRm[half 2], {US,SS}, 16);
  wRd[half 1] = saturate(wRn[half 1] + wRm[half 1], {US,SS}, 16);
  wRd[half 0] = saturate(wRn[half 0] + wRm[half 0], {US,SS}, 16);
}
else if (W Specified) then {
  wRd[word 1] = saturate(wRn[word 1] + wRm[word 1], {US,SS}, 32);
  wRd[word 0] = saturate(wRn[word 0] + wRm[word 0], {US,SS}, 32);
}
```

Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				wwss				wRn				wRd				0001				100			0	wRm			

Sub-Field Encoding

Qualifier	Field	Value
B	ww	00
H	ww	01
W	ww	10
Reserved	ww	11
US	ss	01
SS	ss	11
No Saturation	ss	00
C	ss	10

Note: The encoding space for wwss= 0010 is reserved.

SIMD Flags Sets SIMD (wCASF) flags on the result of the addition wRn+wRm. If US/SS is specified and saturation occurs then the SIMD (wCASF) flags are set on the post-saturation (final) value. The saturation flags (wCSSF) are set if the US/SS qualifier is specified and the corresponding fields saturates.

Flag	No Saturation Specified	US/SS specified and no Result Saturation	US/SS specified and Result Saturates	C option Specified
N	Set on Final Result	Set on Final Result	Set on Final Result	Set on Final Result
Z	Set on Final Result	Set on Final Result	Set on Final Result	Set on Final Result
C	Set on Final Result	Set on Final Result	Cleared	Set on Final Result
V	Set on Final Result	Set on Final Result	Cleared	Set on Final Result

Uses: The carry-in options are useful for performing long addition/subtraction operations common to security applications, RC6 encryption and RSA.

3.2.20 WADDSUBHX

Overview Performs complex vector addition/subtraction of wRn and wRm for vectors of 16-bit data, and places the result in wRd. The four operands from each of the source registers are alternately added and subtracted using a cross selection in each of the parallel operations. The result of the operation is saturated to the signed limits, for example, 0x8000 to 0x7fff or -32768 to 32767.

Usage WADDSUBHX {Cond} wRd, wRn, wRm

Qualifiers None

Operation

```
wRd[half 3] = saturate(wRn[half 3] - wRm[half 2], SS, 16);
wRd[half 2] = saturate(wRn[half 2] + wRm[half 3], SS, 16);
wRd[half 1] = saturate(wRn[half 1] - wRm[half 0], SS, 16);
wRd[half 0] = saturate(wRn[half 0] + wRm[half 1], SS, 16);
```

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				1010				wRn				wRd				0001				101			0	wRm			

Sub-Field Encoding

SIMD Flags Sets the SIMD16 (wCASF) flags as shown below. The saturation flags (wCSSF) are set if the corresponding fields saturates.

.

Flag	No Saturation	Result Saturates
N	Set on Final Result	Set on Final Result
Z	Set on Final Result	Set on Final Result
C	Set on Final Result	Cleared
V	Set on Final Result	Cleared

Note: This instruction is useful for performing radix-2 and radix-4 butterflies on real or complex data structures. The application support include DCT and FFT calculations common to voice, audio, and video applications.

3.2.21 WALIGNI

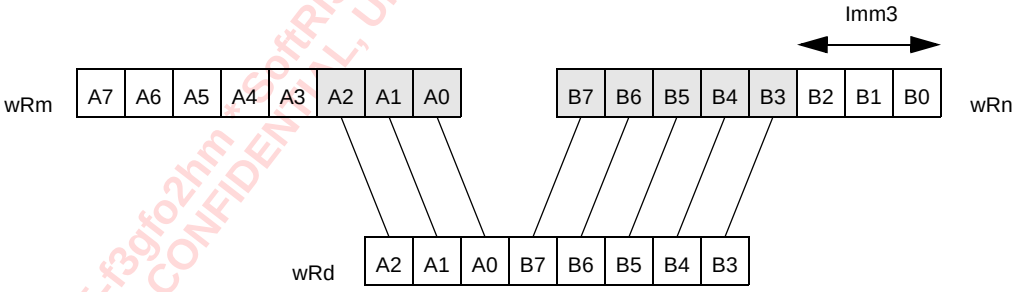
Overview Extracts an 64-bit value from the two 64-bit source registers (wRn,wRm), and places the result in the destination register, wRd; instruction uses a 3-bit immediate value to specify the byte offset of the value to extract.

Usage WALIGNI{Cond} wRd, wRn, wRm, #Imm3

Qualifiers None

Operation wRd = Low_DB_word((wRm,wRn) >>(Imm3 * 8));

Example Operation, with Imm3=3



Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				0vvv				wRn				wRd				0000				001			0	wRm			

Sub-Field Encoding

Qualifier	Field	Value
Imm3	vvv	3-bit Value

SIMD Flags Does not effect the SIMD PSR flags (wCASF) or saturation flags (wCSSF)

Note: Offset is in bytes.

3.2.22 WALIGNR

Overview Extracts a 64-bit value from the two 64-bit source registers (wRn,wRm), and places the result in the destination register, wRd; instruction uses a 3-bit value stored in the specified general-purpose register to specify the byte offset of the value to extract.

Usage WALIGNR<0,1,2,3>{Cond} wRd, wRn, wRm

Qualifiers
0 – Use general purpose register 0 for alignment
1 – Use general purpose register 1 for alignment
2 – Use general purpose register 2 for alignment
3 – Use general purpose register 3 for alignment

Operation

```
if (0 Specified) then
   wRd = Low_DB_word((wRm,wRn) >> (wCGR0[2:0] * 8));
else if (1 Specified) then
   wRd = Low_DB_word((wRm,wRn) >> (wCGR1[2:0] * 8));
else if (2 Specified) then
   wRd = Low_DB_word((wRm,wRn) >> (wCGR2[2:0] * 8));
else if (3 Specified) then
   wRd = Low_DB_word((wRm,wRn) >> (wCGR3[2:0] * 8));
```

Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				10vv				wRn				wRd				0000				001			0	wRm			

Sub-Field Encoding

Qualifier	Field	Value
0	vv	00
1	vv	01
2	vv	10
3	vv	11

SIMD Flags Does not effect the SIMD PSR flags (wCASF) or saturation flags (wCSSF)

Note: Offset is in bytes

3.2.23 WAND

Overview Performs a bitwise logical “AND” between wRn and wRm, and places the result in the destination register, wRd.

Usage WAND{Cond} wRd, wRn, wRm

Qualifiers

Operation

```
wRd = wRn & wRm
```

Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				0010				wRn				wRd				0000				000			0	wRm			

SIMD Flags Sets SIMD64 flags only; C and V flags are cleared and the N and Z flags are set according to the result; does not effect the saturation flags (wCSSF).

Flag	Action
N	Set on Final Result
Z	Set on Final Result
C	Cleared
V	Cleared

Note:

3.2.24 WANDN

Overview Performs a bitwise logical “AND” between wRn and “NOT” wRm, and places the result in the destination register, wRd.

Usage WANDN{Cond} wRd, wRn, wRm

Qualifiers

Operation

```
wRd = wRn & ~wRm;
```

Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				0011				wRn				wRd				0000				000			0	wRm			

SIMD Flags Sets SIMD64 flags only; C and V flags are cleared and the N and Z flags are set according to the result; does not effect the saturation flags (wCSSF).

Flag	Action
N	Set on Final Result
Z	Set on Final Result
C	Cleared
V	Cleared

Note:

3.2.25 WAVG2

Overview Performs a 2-pixel average of wRn and wRm on unsigned vectors of 8- or 16-bit data with optional biased rounding and places the result in the destination register, wRd.

Usage WAVG2<B,H>{R}{Cond} wRd, wRn, wRm

Qualifiers
B – 8-bit (byte) SIMD
H – 16-bit (half word) SIMD
R – Round using + 1, else no rounding

Operation

```
Round = (R Specified) ? : 1 :0;
if (B Specified) then{
   wRd[byte 7] = (wRn[byte 7] + wRm[byte 7] + Round) >> 1;
   wRd[byte 6] = (wRn[byte 6] + wRm[byte 6] + Round) >> 1;
   wRd[byte 5] = (wRn[byte 5] + wRm[byte 5] + Round) >> 1;
   wRd[byte 4] = (wRn[byte 4] + wRm[byte 4] + Round) >> 1;
   wRd[byte 3] = (wRn[byte 3] + wRm[byte 3] + Round) >> 1;
   wRd[byte 2] = (wRn[byte 2] + wRm[byte 2] + Round) >> 1;
   wRd[byte 1] = (wRn[byte 1] + wRm[byte 1] + Round) >> 1;
   wRd[byte 0] = (wRn[byte 0] + wRm[byte 0] + Round) >> 1;
}
else if (H Specified) then{
   wRd[half 3] = (wRn[half 3] + wRm[half 3] + Round) >> 1;
   wRd[half 2] = (wRn[half 2] + wRm[half 2] + Round) >> 1;
   wRd[half 1] = (wRn[half 1] + wRm[half 1] + Round) >> 1;
   wRd[half 0] = (wRn[half 0] + wRm[half 0] + Round) >> 1;
}
```

Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				1h0r				wRn				wRd				0000				000			0	wRm			

Sub-Field Encoding

Qualifier	Field	Value
B	h	0
H	h	1
R	r	1
No R	r	0

SIMD Flags Clears N,C,V, Sets Z if zero; does not affect the saturation flags (wCSSF).

Flag	Action
N	Cleared
Z	Set on Final Result
C	Cleared
V	Cleared

Note: The rounding specifies whether a 1 is added to the intermediate result before the right shift.

3.2.26 WAVG4

Overview Performs seven 4-pixel averages of unsigned 8-bit operands obtained from the bytes of the wRn and wRm source registers. Biased rounding is supported through the use of the R-qualifier. The seven 4-pixel averages are packed into the lower seven bytes of the destination register with the most significant byte written with 0x00.

Usage WAVG4 {R} {Cond} wRd, wRn, wRm

Qualifiers R – Round using + 2, else use +1

Operation

```
Round = (R Specified) ? : 1 :0;
   wRd[byte 7] = 0
   wRd[byte 6] = (wRn[byte 7] + wRm[byte 7] + wRn[byte 6] + wRm[byte 6] + 1 + Round) >>2;
   wRd[byte 5] = (wRn[byte 6] + wRm[byte 6] + wRn[byte 5] + wRm[byte 5] + 1 + Round) >>2;
   wRd[byte 4] = (wRn[byte 5] + wRm[byte 5] + wRn[byte 4] + wRm[byte 4] + 1 + Round) >>2;
   wRd[byte 3] = (wRn[byte 4] + wRm[byte 4] + wRn[byte 3] + wRm[byte 3] + 1 + Round) >>2;
   wRd[byte 2] = (wRn[byte 3] + wRm[byte 3] + wRn[byte 2] + wRm[byte 2] + 1 + Round) >>2;
   wRd[byte 1] = (wRn[byte 2] + wRm[byte 2] + wRn[byte 1] + wRm[byte 1] + 1 + Round) >>2;
   wRd[byte 0] = (wRn[byte 1] + wRm[byte 1] + wRn[byte 0] + wRm[byte 0] + 1 + Round) >>2;
```

Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				010r				wRn				wRd				0000				000			0	wRm			

Sub-Field Encoding

Qualifier	Field	Value
R	r	1
No R	r	0

SIMD Flags Sets SIMD8 flags only; clears N,C,V, Sets Z if zero; does not affect the saturation flags (wCSSF).

Flag	Action
N	Cleared
Z	Set on Final Result
C	Cleared
V	Cleared

Uses: This instruction is useful for both video encode and decode algorithms. The fractional motion estimation makes use of this operation in the encode chain, and the 1/2XY motion compensation makes use of this operation in the decode chain.

3.2.27 WCMPEQ

Overview Performs vector-equality comparison of wRn and wRm for vectors of 8-, 16-, or 32-bit data, setting the corresponding data elements of wRd to all ones when source operands are equal; else all zeros.

Usage WCMPEQ<B,H,W>{Cond} wRd, wRn, wRm

Qualifiers
B – 8-bit (byte) SIMD
H – 16-bit (half word) SIMD
W – 32-bit (word) SIMD

Operation

```
if (B Specified)then
   wRd[byte 7] = (wRn[byte 7] == wRm[byte 7]) ? 0xFF : 0x00;
   wRd[byte 6] = (wRn[byte 6] == wRm[byte 6]) ? 0xFF : 0x00;
   wRd[byte 5] = (wRn[byte 5] == wRm[byte 5]) ? 0xFF : 0x00;
   wRd[byte 4] = (wRn[byte 4] == wRm[byte 4]) ? 0xFF : 0x00;
   wRd[byte 3] = (wRn[byte 3] == wRm[byte 3]) ? 0xFF : 0x00;
   wRd[byte 2] = (wRn[byte 2] == wRm[byte 2]) ? 0xFF : 0x00;
   wRd[byte 1] = (wRn[byte 1] == wRm[byte 1]) ? 0xFF : 0x00;
   wRd[byte 0] = (wRn[byte 0] == wRm[byte 0]) ? 0xFF : 0x00;
else if (H Specified) then
   wRd[half 3] = (wRn[half 3] == wRm[half 3]) ? 0xFFFF : 0x0000;
```

```
    wRd[half 2] = (wRn[half 2] == wRm[half 2]) ? 0xFFFF : 0x0000;
    wRd[half 1] = (wRn[half 1] == wRm[half 1]) ? 0xFFFF : 0x0000;
    wRd[half 0] = (wRn[half 0] == wRm[half 0]) ? 0xFFFF : 0x0000;
else if (W Specified) then
    wRd[word 1] = (wRn[word 1] == wRm[word 1]) ? 0xFFFFFFFF : 0x00000000;
    wRd[word 0] = (wRn[word 0] == wRm[word 0]) ? 0xFFFFFFFF : 0x00000000;
```

Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				ww00				wRn				wRd				0000				011			0	wRm			

Sub-Field Encoding

Qualifier	Field	Value
B	ww	00
H	ww	01
W	ww	10
Reserved	ww	11

SIMD Flags Set SIMD flags as if operation on the final has occurred; does not effect the saturation flags (wCSSF). Z flag is cleared if operands are equal (final result = 0xFF) and is set if there is a mismatch (final result = 0x00).

Flag	Action
N	Set on Final Result
Z	Set on Final Result
C	Cleared
V	Cleared

3.2.28 WCMPGT

Overview Performs vector-magnitude comparison of wRn and wRm for vectors of 8-, 16-, or 32-bit data, setting the corresponding data elements of wRd to all ones when corresponding fields of wRn and greater than wRm; else all zeros; operation can be performed on either signed or unsigned data.

Usage WCMPGT<U,S><B,H,W>{Cond} wRd, wRn, wRm

Qualifiers
B – 8-bit (byte) SIMD
H – 16-bit (half word) SIMD
W – 32-bit (word) SIMD

S – Signed comparison
U – Unsigned comparison

Operation

```
if(S Specified) then
   wRn and wRm contain signed values
else
   wRn and wRm contain unsigned values.

if (B Specified) then
   wRd[byte 7] = (wRn[byte 7] > wRm[byte 7]) ? 0xFF : 0x00;
   wRd[byte 6] = (wRn[byte 6] > wRm[byte 6]) ? 0xFF : 0x00;
   wRd[byte 5] = (wRn[byte 5] > wRm[byte 5]) ? 0xFF : 0x00;
   wRd[byte 4] = (wRn[byte 4] > wRm[byte 4]) ? 0xFF : 0x00;
   wRd[byte 3] = (wRn[byte 3] > wRm[byte 3]) ? 0xFF : 0x00;
   wRd[byte 2] = (wRn[byte 2] > wRm[byte 2]) ? 0xFF : 0x00;
   wRd[byte 1] = (wRn[byte 1] > wRm[byte 1]) ? 0xFF : 0x00;
   wRd[byte 0] = (wRn[byte 0] > wRm[byte 0]) ? 0xFF : 0x00;
else if (H Specified) then
   wRd[half 3] = (wRn[half 3] > wRm[half 3]) ? 0xFFFF : 0x0000;
   wRd[half 2] = (wRn[half 2] > wRm[half 2]) ? 0xFFFF : 0x0000;
   wRd[half 1] = (wRn[half 1] > wRm[half 1]) ? 0xFFFF : 0x0000;
   wRd[half 0] = (wRn[half 0] > wRm[half 0]) ? 0xFFFF : 0x0000;
else if (W Specified) then
   wRd[word 1] = (wRn[word 1] > wRm[word 1]) ? 0xFFFFFFFF : 0x00000000;
   wRd[word 0] = (wRn[word 0] > wRm[word 0]) ? 0xFFFFFFFF : 0x00000000;
```

Exceptions

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				wws1				wRn				wRd				0000				011			0	wRm			

Sub-Field Encoding

Qualifier	Field	Value
B	ww	00
H	ww	01
W	ww	10
Reserved	ww	11
S	s	1
U	s	0

SIMD Flags Sets SIMD PSR flags; does not effect the saturation flags (wCSSF).

Flag	Action
N	Set on Final Result
Z	Set on Final Result
C	Cleared
V	Cleared

3.2.29 WLDR

Overview Performs a load from memory into either a data register (wRd) or a control register (wCx); 8-,16-, 32-, and 64-bit data values can be loaded in the data registers and 32-bit values only can be loaded into the control registers; loaded data is zero-extended. The immediate offset addressing mode is provided for data and control register load operations. The register offset addressing mode is provided for 64-bit loads to the data registers. A left shift may be optionally applied to the offset when using the register offset addressing mode.

Usage
```
WLDR<B,H,W,D>{Cond} wRd, <Imm_offset_address_mode>
WLDRD wRd, <Reg_offset_address_mode>
WLDRW wCx, <Imm_offset_address_mode>
```

Qualifiers
- B – Load byte
- H – Load half
- W – Load word
- D – Load double

Operation
```
if (B specified) then
   wRd[7:0] = Mem[<Imm_offset_address_mode>] ; wRd[63:8]=0;
else if (H specified) then
   wRd[15:0] = Mem[<Imm_offset_address_mode>] ; wRd[63:16]=0;
else if (W specified) then
   wRd[31:0] = Mem[<Imm_offset_address_mode>] ; wRd[63:32]=0;
else if (D Specifed) then
   wRd[63:0] = Mem[<Imm_offset_address_mode> or <Reg_offset_address_mode>] ;
```

Exceptions None

Encoding WLDR<B,H,W,D>{Cond} wRd

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				110			P	U	N	W	1	Rn				wRd				000			M	offset_8								

Sub-Field Encoding

Qualifier	Field	Value
B	N,M	0,0
H	N,M	1,0
W	N,M	0,1
D	N,M	1,1

Sub-Field Encoding<Imm_offset_address_mode>

The following standard ARM* V5 coprocessor immediate offset address modes are supported:

Table 3-1. Immediate Offset Address Mode Encoding

P	U	W	Mnemonic	Description
0	0	1	[Rn], #- AS_offset	Post-index, negative offset, always writeback
0	1	1	[Rn], #+ AS_offset	Post-index, positive offset, always writeback
1	0	0	[Rn, #- AS_offset]	Pre-index, negative offset
1	0	1	[Rn, #- AS_offset]!	Pre-index, negative offset, with writeback
1	1	0	[Rn, #+ AS_offset]	Pre-index, positive offset
1	1	1	[Rn, #+ AS_offset]!	Pre-index, positive offset, with writeback

Note: For the immediate offset used in the address calculation is always specified to the assembler in bytes. However for the W and D qualifiers the immediate offset must be a word multiple (that is divisible by 4). This allows the word- and double-load variants to have a greater offset range (0-0x3FC) than the byte-and half-word load variants (0 -0xFF). See Table 3-2.

Table 3-2. Address Mode Assembler Offset Specification and Instruction Encoding

Flag	Assembler Offset Units (AS_offset)	Assembler Offset Restriction	Instruction encoding (offset_8)
B	Byte	AS_offset < 0xFF	AS_offset[7:0]
H	Byte	AS_offset < 0xFF	AS_offset[7:0]
W	Byte	AS_offset < 0x3FC & AS_offset mod 4 = 0	AS_offset[9:2]
D	Byte	AS_offset < 0x3FC & AS_offset mod 4 = 0	AS_offset[9:2]

If an offset is required that does not meet the above restrictions, the address must be modified using a separate instruction (for example, add/subtract).

Encoding WLDRD wRd, <Reg_offset_address_mode>

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1111				110			P	U	1	W	1	Rn				wRd				000			1	Imm4				Rm			

Sub-Field Encoding

These register offset address modes are supported:

Table 3-3. Register offset address mode encoding

P	U	W	Mnemonic	Description
0	0	1	[Rn], - Rm {,LSL #Imm4}	Post-index, negative Rm, always writeback, with optional shift of Rm
0	1	1	[Rn], + Rm {,LSL #Imm4}	Post-index, positive Rm, always writeback, with optional shift of Rm
1	0	0	[Rn, - Rm {,LSL #Imm4}]	Pre-index, negative Rm, with optional shift of Rm
1	0	1	[Rn, - Rm{,LSL #Imm4}]!	Pre-index, negative Rm, with writeback, with optional shift of Rm
1	1	0	[Rn, + Rm {,LSL #Imm4}]	Pre-index, positive Rm, with optional shift of Rm
1	1	1	[Rn, + Rm {,LSL #Imm4}]!	Pre-index, positive Rm, with writeback, with optional shift of Rm

Note: "{}" indicates optional mnemonic

Encoding WLDRW wCx

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1111				110			P	U	0	W	1	Rn				wCx				0001				offset_8							

Note: Specifying an illegal value for wCx causes an undefined instruction exception

Sub-Field Encoding

Uses same address mode encodings as shown in Table 3-1.

SIMD Flags Does not effect the SIMD PSR flags (wCASF) or saturation flags (wCSSF)

Note: Load to Marvell® Wireless MMX™ 2 coprocessor control registers are word only and cannot be conditionally executed.
Register offset is always specified in bytes and has no restrictions apart from alignment of the final address. (See below)

Addresses should be aligned at the byte/half/word/double boundary of the data item being load. Lower address bits should be cleared if the unaligned address traps is enabled on the main core, otherwise unaligned address trap is taken.

If unaligned traps are disabled, this table illustrates what happens.

Qualifier	Unaligned Behavior
B	never unaligned
H,W,D	unpredictable

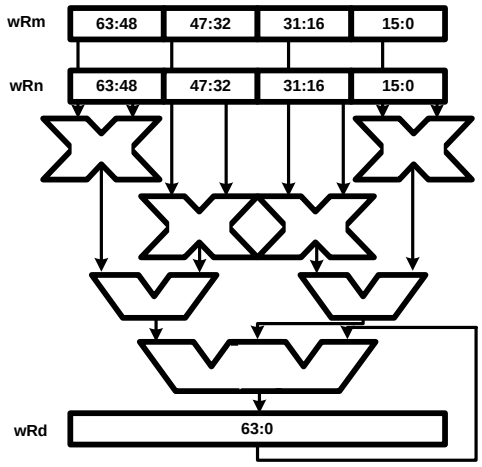
3.2.30 WMAC

Overview Performs a vector multiplication of wRn and wRm and can accumulate the result with wRd on vectors of 16-bit data only.

Usage WMAC<U,S>{Z}{Cond} wRd, wRn, wRm

Qualifiers
Z – Zero destination before accumulation
S – Signed
U – Unsigned

Operation



Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				01sz				wRn				wRd				0001				000			0	wRm			

Sub-Field Encoding

Qualifier	Field	Value
S	s	1
U	s	0
Z	z	1
No Z	z	0

SIMD Flags Does not effect the SIMD PSR flags (wCASF) or saturation flags (wCSSF).

Note: The input arguments are (Ax,Bx) 16-bits, the intermediate values (Px) are 32-bits and the result is 64-bits. Signed and unsigned qualifier applies to both multiplication and the final accumulation. Sign extension is performed, where appropriate, to the intermediate results.

3.2.31 WMADD

Overview Performs a 16-bit vector multiplication and then sums the lower two products into the bottom word of wRd, and the upper two products into the upper word of wRd; intermediate products are 32 bit; can operate on either signed or unsigned data. The operands may be cross multiplied with the use of the X-qualifier and may be optionally subtracted with the use of the N-qualifier. The X-qualifier may only be used with the default multiply-add operation.

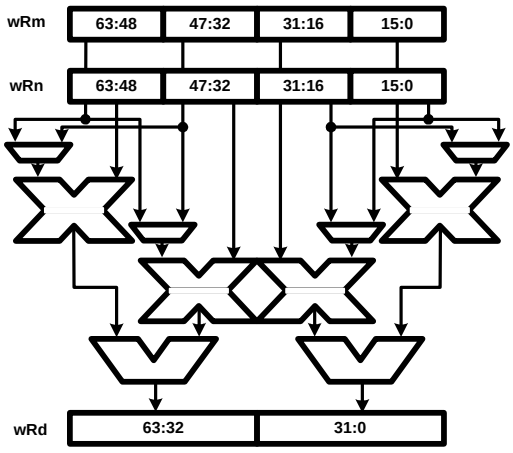
Usage WMADD<U,S>{X,N} {Cond} wRd, wRn, wRm

Qualifiers
S – Signed
U – Unsigned
X – Cross the operands when performing the multiply add
N – Perform a multiply subtract instead of the multiply add.

Operation

```
if (N specified) then {
   wRd[31:0] = (wRn[15:0]*wRm[15:0]) - (wRn[31:16]*wRm[31:16]);
   wRd[63:32] = (wRn[47:32]*wRm[47:32]) - (wRn[63:48]*wRm[63:48]);
 }
 else if (X specified) then {
   wRd[31:0] = (wRn[15:0]*wRm[31:16]) + (wRn[31:16]*wRm[15:0]);
   wRd[63:32] = (wRn[63:48]*wRm[47:32]) + (wRn[47:32]*wRm[63:48]);
 }
 else {
   wRd[31:0] = (wRn[15:0]*wRm[15:0]) + (wRn[31:16]*wRm[31:16]);
   wRd[63:32] = (wRn[47:32]*wRm[47:32]) + (wRn[63:48]*wRm[63:48]);
```

}



Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				nnsx				wRn				wRd				0001				000			0	wRm			

Sub-Field Encoding

nn = 10 (No N)			nn = 11 (N)		
Qualifier	Field	Value	Qualifier	Field	Value
S	s	1	S	s	1
U	s	0	U	s	0
X	x	1	x=0		
No X	x	0			

Note: The negation is valid only without the “X-qualifier. The encoding space for nn=11 and sx= 01 or sx = 11 is not valid for this instruction.

SIMD Flags Does not effect the SIMD PSR flags (wCASF) or saturation flags (wCSSF).

Note: If the result is larger than the result field, the result just truncates; it does not saturate (this is the same behavior as the Marvell® Wireless MMX™ 2 PMADDWM instruction.); overflow can be prevented by prescaling the input arguments

Uses: This instruction is useful for performing a parallel complex multiplication without requiring the user to duplicate coefficients or provide a separate negation (a+jb)*(c+jd) = ac - bd + j(bc+ad) (r+js)*(t+ju) = (rt - su) + j(st + ru).

3.2.32 WMAX

Overview Performs vector maximum selection of elements from wRn and wRm for vectors of 8-, 16-, or 32-bit data and places the maximum fields in the destination register, wRd; can be performed on signed or unsigned data.

Usage WMAX<U,S><B,H,W> {Cond} wRd, wRn, wRm

Qualifiers
B – 8-bit (byte) SIMD
H – 16-bit (half word) SIMD
W – 32-bit (word) SIMD
S – Signed comparison
U – Unsigned comparison

Operation

```
if (B Specified)then
   wRd[byte 7] = (wRn[byte 7] > wRm[byte 7]) ? wRn[byte 7] : wRm[byte 7];
   wRd[byte 6] = (wRn[byte 6] > wRm[byte 6]) ? wRn[byte 6] : wRm[byte 6];
   wRd[byte 5] = (wRn[byte 5] > wRm[byte 5]) ? wRn[byte 5] : wRm[byte 5];
   wRd[byte 4] = (wRn[byte 4] > wRm[byte 4]) ? wRn[byte 4] : wRm[byte 4];
   wRd[byte 3] = (wRn[byte 3] > wRm[byte 3]) ? wRn[byte 3] : wRm[byte 3];
   wRd[byte 2] = (wRn[byte 2] > wRm[byte 2]) ? wRn[byte 2] : wRm[byte 2];
   wRd[byte 1] = (wRn[byte 1] > wRm[byte 1]) ? wRn[byte 1] : wRm[byte 1];
   wRd[byte 0] = (wRn[byte 0] > wRm[byte 0]) ? wRn[byte 0] : wRm[byte 0];
else if (H Specified)then
   wRd[half 3] = (wRn[half 3] > wRm[half 3]) ? wRn[half 3] : wRm[half 3];
   wRd[half 2] = (wRn[half 2] > wRm[half 2]) ? wRn[half 2] : wRm[half 2];
   wRd[half 1] = (wRn[half 1] > wRm[half 1]) ? wRn[half 1] : wRm[half 1];
   wRd[half 0] = (wRn[half 0] > wRm[half 0]) ? wRn[half 0] : wRm[half 0];
else if (W Specified) then
   wRd[word 1] = (wRn[word 1] > wRm[word 1]) ? wRn[word 1] : wRm[word 1];
   wRd[word 0] = (wRn[word 0] > wRm[word 0]) ? wRn[word 0] : wRm[word 0];
```

Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				wws0				wRn				wRd				0001				011			0	wRm			

Sub-Field Encoding

Qualifier	Field	Value
B	ww	00
H	ww	01
W	ww	10
Reserved	ww	11
S	s	1
U	s	0

SIMD Flags Does not effect the SIMD PSR flags (wCASF) or saturation flags (wCSSF)

3.2.33 WMERGE

Overview Extracts a 64-bit value that contains elements from the two 64-bit source registers (wRn,wRm), and places a merged 64-bit result in the destination register, wRd. The number of adjacent elements from each register is represented with a 3-bit immediate.

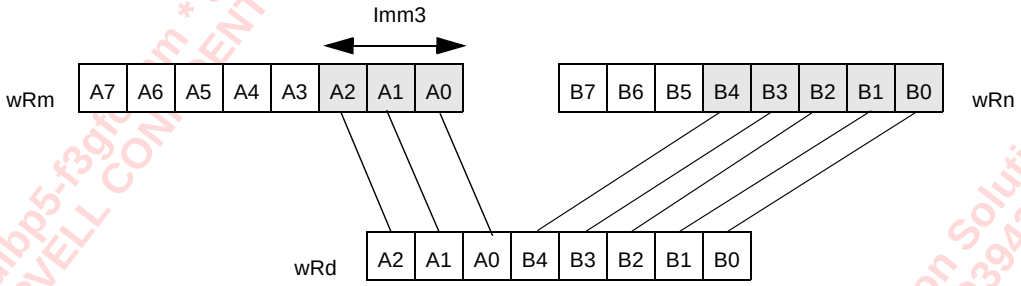
Usage WMERGE{Cond} wRd, wRn, wRm, #Imm3

Qualifiers None

Operation

```
wRd = ((wRm << (64 - Imm3*8) | ((wRn << Imm3*8) >> Imm3*8)));
```

Example Operation, with Imm3=3



Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				cba0				wRn				wRd				0000				100			0	wRm			

Sub-Field Encoding

Field	Value
cba	Imm3[2:0]

SIMD Flags Does not effect the SIMD PSR flags (wCASF) or saturation flags (wCSSF).

Note: Offset is in bytes.

3.2.34 WMIAxy

Overview Performs a signed parallel 16-bit multiply, or multiply-negate, followed by 64-bit accumulation. The upper or lower 16-bits of each source register half is selected by specifying either the B (bottom) or T (top) in each of the xy positions of the mnemonic. The xy selection is the same for both the upper and lower halves of the 64-bit operand source registers, wRn and wRm. The resulting product for the upper and lower halves are then added to the 64-bit accumulator which occupies the destination register wRd. The N-qualifier optionally provides for a multiply-negate-accumulate operation instead of the default multiply-accumulate.

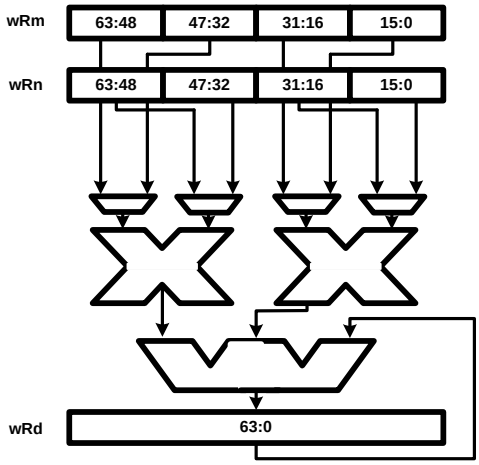
Usage WMIA<T,B><T,B>{N}{Cond} wRd, wRn, wRm

Qualifiers
T – Operands are from the top fields.
B – Obtain operands fromthe bottom fields.
N – Provide multiply negate instead of multiply accumulate

Operation

```
<operand1> = (T Specified in x position) ? wRn[63:48]: wRn[47:32];
<operand2> = (T Specified in y position) ? wRm[63:48]: wRm[47:32];
<operand3> = (T Specified in x position) ?wRn[31:16] : wRn[15:0];
<operand4> = (T Specified in y position) ? wRm[31:16] : wRm[15:0];

if (N specified) then {
   wRd[63:0] = wRd[63:0] - signExtend((<operand1>*<operand2>),64) -
               signExtend((<operand3>*<operand4>),64);
}
else {
   wRd[63:0] = wRd[63:0] + signExtend((<operand1>*<operand2>),64) +
               signExtend((<operand3>*<operand4>),64);
   }
```



Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				0nxy				wRn				wRd				0000				101			0	wRm			

Sub-Field Encoding

Qualifier	Field	Value
T	x	1
B	x	0
T	y	1
B	y	0
N	n	1
No N	n	0

SIMD Flags Does not effect the SIMD PSR flags (wCASF) or saturation flags (wCSSF)

Note: This instruction is useful for implementing complex operations as well as implementing real filtering operations. The negation capability allows coefficient memory use and load bandwidth to be reduced.

3.2.35 WMIAWxy

Overview Performs multiply-accumulate using signed 32-bit operands from the two source wRm and wRn, and accumulates the result with the destination register, wRd. The upper or lower 32-bits of each source register half is selected by specifying either the B (bottom) or T (top) in each of the xy positions of the mnemonic.The N-qualifier optionally provides for a multiply-negate-accumulate operation instead of the default multiply-accumulate.

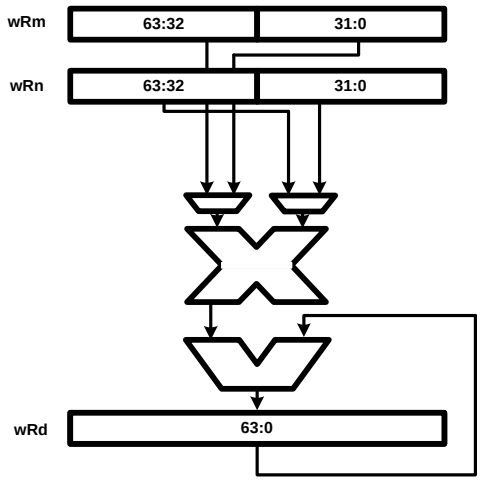
Usage WMIAW<T,B><T,B>{N}{Cond} wRd, wRn, wRm

Qualifiers
T – Obtain operands from the most significant words of wRn,wRm
B – Obtain operands from the least significant words of wRn, wRm
N – Provide multiply negate instead of multiply accumulate

Operation

```
<operand1> = (T Specified in x position) ? wRn[63:32] : wRn[31:0];
<operand2> = (T Specified in y position) ? wRm[63:32] : wRm[31:0];

If (N specified) then {
  wRd[63:0] = wRd[63:0] - (<operand1>*<operand2>);
}
else {
  wRd[63:0] = wRd[63:0] + (<operand1>*<operand2>);
}
```



Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				1nxy				wRn				wRd				0001				001			0	wRm			

Sub-Field Encoding

Qualifier	Field	Value
T	x	1
B	x	0
T	y	1
B	y	0
N	n	1
No N	n	0

SIMD Flags Does not effect the SIMD PSR flags (wCASF) or saturation flags (wCSSF)

Uses: This instruction is useful for higher precision real and complex filtering operations and can be used in audio codecs synthesis filter banks found in MP3 and AAC codecs. This instruction is also useful for 2D graphics operations.

3.2.36 WMIN

Overview Performs vector minimum selection of elements from wRn and wRm for vectors of 8-, 16-, or 32-bit data, and places the minimum fields in the destination register, wRd; can be performed on signed or unsigned data.

Usage WMIN<U,S><B,H,W>{Cond} wRd, wRn, wRm

Qualifiers
B – 8-bit (byte) SIMD
H – 16-bit (half word) SIMD
W – 32-bit (word) SIMD
S – Signed comparison
U – Unsigned comparison

Operation

```
if (B Specified) then
   wRd[byte 7] = (wRn[byte 7] < wRm[byte 7]) ? wRn[byte 7] : wRm[byte 7];
   wRd[byte 6] = (wRn[byte 6] < wRm[byte 6]) ? wRn[byte 6] : wRm[byte 6];
   wRd[byte 5] = (wRn[byte 5] < wRm[byte 5]) ? wRn[byte 5] : wRm[byte 5];
   wRd[byte 4] = (wRn[byte 4] < wRm[byte 4]) ? wRn[byte 4] : wRm[byte 4];
   wRd[byte 3] = (wRn[byte 3] < wRm[byte 3]) ? wRn[byte 3] : wRm[byte 3];
   wRd[byte 2] = (wRn[byte 2] < wRm[byte 2]) ? wRn[byte 2] : wRm[byte 2];
   wRd[byte 1] = (wRn[byte 1] < wRm[byte 1]) ? wRn[byte 1] : wRm[byte 1];
   wRd[byte 0] = (wRn[byte 0] < wRm[byte 0]) ? wRn[byte 0] : wRm[byte 0];
else if (H Specified) then
   wRd[half 3] = (wRn[half 3] < wRm[half 3]) ? wRn[half 3] : wRm[half 3];
   wRd[half 2] = (wRn[half 2] < wRm[half 2]) ? wRn[half 2] : wRm[half 2];
   wRd[half 1] = (wRn[half 1] < wRm[half 1]) ? wRn[half 1] : wRm[half 1];
   wRd[half 0] = (wRn[half 0] < wRm[half 0]) ? wRn[half 0] : wRm[half 0];
else if (W Specified) then
   wRd[word 1] = (wRn[word 1] < wRm[word 1]) ? wRn[word 1] : wRm[word 1];
   wRd[word 0] = (wRn[word 0] < wRm[word 0]) ? wRn[word 0] : wRm[word 0];
```

Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				wws1				wRn				wRd				0001				011			0	wRm			

Sub-Field Encoding

Qualifier	Field	Value
B	ww	00
H	ww	01
W	ww	10
Reserved	ww	11
S	s	1
U	s	0

SIMD Flags Does not effect the SIMD PSR flags (wCASF) or saturation flags (wCSSF)

3.2.37 WMOV

Overview This pseudo-instruction moves register wRn to register wRd; this instruction is a form of WOR.

Usage WMOV{Cond} wRd, wRn

Qualifiers None

Operation WOR wRd,wRn,wRn

Exceptions None

Encoding see WOR

Sub-Field Encoding see WOR

SIMD Flags see WOR

Note: This is a pseudo-instruction that maps onto another Marvell® Wireless MMX™ 2 coprocessor instruction and is provided for convenience

3.2.38 WMUL

Overview Performs a vector multiplication of wRn and wRm on vectors of 16-bit data only; M-qualifier indicates that the higher order 16 bits of the result are to be stored in wRd, the L-qualifier indicates that the lower 16 bits of the result are to be stored in wRd; can be performed on signed or unsigned data. Biased rounding is available with the M-qualifier through the use of the R-qualifier.

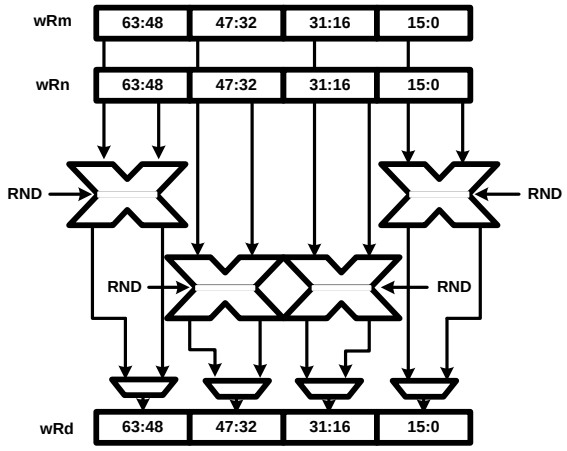
Usage WMUL<U,S><M,L> {R}{Cond} wRd, wRn, wRm

Qualifiers
M – Most significant bits of the result to be saved
L – Least significant bits of the result to be saved
S – Signed
U – Unsigned
R – Round least significant bits into most significant bits.

Operation

```
if (L Specified) {
   wRd[63:48] = (wRm[63:48]*wRn[63:48]);
   wRd[47:32] = (wRm[47:32]*wRn[47:32]) ;
   wRd[31:16] = (wRm[31:16]*wRn[31:16]) ;
   wRd[15:0] = (wRm[15:0]*wRn[15:0]) ;
 }
 else{
 if (R Specified) then {
   wRd[63:48] = (wRm[63:48]*wRn[63:48] + 0x8000) >> 16;
   wRd[47:32] = (wRm[47:32]*wRn[47:32] + 0x8000) >> 16;
   wRd[31:16] = (wRm[31:16]*wRn[31:16] + 0x8000) >> 16;
   wRd[15:0] = (wRm[15:0]*wRn[15:0] + 0x8000)>> 16;
 }
 else {
   wRd[63:48] = (wRm[63:48]*wRn[63:48]) >>16;
   wRd[47:32] = (wRm[47:32]*wRn[47:32])>> 16;
   wRd[31:16] = (wRm[31:16]*wRn[31:16]) >> 16;
   wRd[15:0] = (wRm[15:0]*wRn[15:0]) >> 16;
```

```
}
}
```



Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				rrsf				wRn				wRd				0001				000			0	wRm			

Sub-Field Encoding

rr = 00 (No Rounding)			rr = 11 (Rounding)		
Qualifier	Field	Value	Qualifier	Field	Value
S	s	1	S	s	1
U	s	0	U	s	0
M	f	1	M	f	1
L	f	0			

Note: The rounding is valid only for the M-qualifier (upper half selection), the encoding space for rr=11 and sf = 00 or sf = 10 are not valid for this instruction.

SIMD Flags Does not effect the SIMD PSR flags (wCASF) or saturation flags (wCSSF).

Note: When L is specified the U and S qualifiers produce the same result. The R-qualifier is only valid when the M-qualifier is set.

Uses: This instruction is useful for a number of applications requiring 16-bit precision. The rounding operation finds use in both voice and video codecs.

3.2.39 WMULW

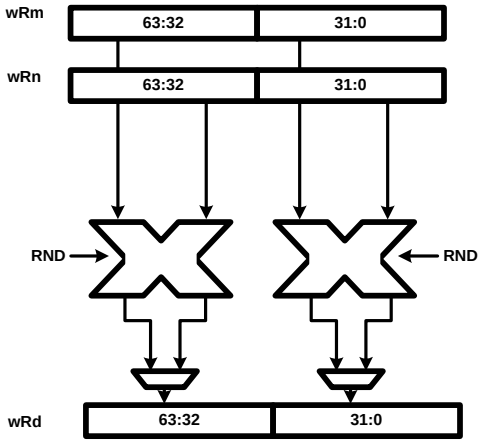
Overview Performs a vector multiplication of wRn and wRm on vectors of 32-bit data only; UM-qualifier indicates that the higher order 32 bits of the result of the unsigned multiplication are to be stored in wRd, the SM-qualifier indicates that the higher order 32 bits of the signed multiplication are to be stored in wRd, and the L-qualifier indicates that the lower 32 bits of the result are to be stored in wRd. Biased rounding is available with the UM and SM options through the use of the R-qualifier.

Usage WMULW <UM,SM,L> {R}{Cond} wRd, wRn, wRm

Qualifiers
UM – Most significant bits of the result of unsigned operation to be saved
SM – Most significant bits of the result of signed operation to be save
L – Least significant bits of the result to be saved
R – Round lower 32-bits into upper 32-bits

Operation

```
if (L Specified) {
   wRd[63:32] = (wRm[63:32]*wRn[63:32]);
   wRd[31:0] = (wRm[31:0]*wRn[31:0]) ;
}
else {
   if (R Specified) then {
   wRd[63:32] = (wRm[63:32]*wRn[63:32] + 0x80000000) >> 32;
   wRd[31:0] = (wRm[31:0]*wRn[31:0] + 0x80000000) >> 32;
}
else{
   wRd[63:32] = (wRm[63:32]*wRn[63:32]) >> 32;
   wRd[31:0] = (wRm[31:0]*wRn[31:0]) >> 32;
}}
```



Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				1fsr				wRn				wRd				0000				110			0	wRm			

Sub-Field Encoding

Qualifier	Field	Value
SM	fs	11
UM	fs	10
L	fs	01
R	r	0
No R	r	1

Note: When the L-qualifier is selected, rounding is not valid and signed/unsigned operands produces the same result. The combinations for fsr = 000, 001, and 010 are not valid for this instruction.

Sub-Field Encoding None

SIMD Flags Does not effect the SIMD PSR flags (wCASF) or saturation flags (wCSSF).

Note: When L is specified the U and S qualifiers produce the same result. The R-qualifier is only valid when the M-qualifier is set.

Uses: This instruction is useful for higher precision operations as found in audio codecs such as MP3 and AAC. This instruction is also useful in implementing 2D graphics functions.

3.2.40 WOR

Overview Performs a bitwise logical “OR” between wRn and wRm and places the result in the destination register, wRd

Usage WOR{Cond} wRd, wRn, wRm

Qualifiers None

Operation wRd = wRn | wRm;

Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				0000				wRn				wRd				0000				000			0	wRm			

Sub-Field Encoding None

SIMD Flags Sets SIMD64 flags only; C and V flags are cleared and the N and Z flags are set according to the result. Does not effect the saturation flags (wCSSF).

Flag	Action
N	Set on Final Result
Z	Set on Final Result
C	Cleared
V	Cleared

3.2.41 WPACK

Overview Packs data from wRn and wRm into wRd, with wRm being packed into the upper half, and wRn being packed into the lower half for vectors of 16-, 32-, or 64-bit data, and saturate the results and place in the destination register wRd; packing can be performed with signed saturation or unsigned saturation

Usage WPACK<H,W,D><US,SS>{Cond} wRd, wRn, wRm

Qualifiers
H – Pack 16-bit (Half Word) into 8-bits
W – Pack 32-bit (Word) into 16-bits
D – Pack 64-bit (Double) into 32-bits
SS – Signed saturation
US – Unsigned saturation

Operation

```
if (H Specified)then
   wRd[byte 7] = saturate(wRm[half 3], {US,SS}, 8);
   wRd[byte 6] = saturate(wRm[half 2], {US,SS}, 8);
   wRd[byte 5] = saturate(wRm[half 1], {US,SS}, 8);
   wRd[byte 4] = saturate(wRm[half 0], {US,SS}, 8);
   wRd[byte 3] = saturate(wRn[half 3], {US,SS}, 8);
   wRd[byte 2] = saturate(wRn[half 2], {US,SS}, 8);
   wRd[byte 1] = saturate(wRn[half 1], {US,SS}, 8);
   wRd[byte 0] = saturate(wRn[half 0], {US,SS}, 8);
else if (W Specified)then
   wRd[half 3] = saturate(wRm[word 1], {US,SS}, 16);
   wRd[half 2] = saturate(wRm[word 0], {US,SS}, 16);
   wRd[half 1] = saturate(wRn[word 1], {US,SS}, 16);
   wRd[half 0] = saturate(wRn[word 0], {US,SS}, 16);
else if (D Specified) then
   wRd[word 1] = saturate(wRm, {US,SS}, 32);
```

wRd[word 0] = saturate(wRn, {US,SS}, 32);

Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				wwss				wRn				wRd				0000				100			0	wRm			

Sub-Field Encoding

Qualifier	Field	Value
H	ww	01
W	ww	10
D	ww	11
US	ss	01
SS	ss	11

Note: The combination ww= 00 is used for other instruction encodings.

SIMD Flags The saturation flags (wCSSF) are set if the corresponding field saturates (US, SS)
The C and V are cleared and the Z, N set according to the final result of the pack operation.

Flag	Action
N	Set on Final Result
Z	Set on Final Result
C	Cleared
V	Cleared

SIMD flags are set on the SIMD width of the result (see below)

Qualifier	SIMD flags
H	SIMD8
W	SIMD16
D	SIMD32

Note: The US and SS refer to the saturation limits applied not to the type of the incoming arguments. The incoming arguments are always treated as signed values (same as Marvell® Wireless MMX™ 2).

3.2.42 WQMIAxy

Overview Performs a signed fractional parallel 16-bit multiply, or multiply-negate, followed by parallel 32-bit accumulation. The upper or lower 16-bits of each Marvell® Wireless MMX™ 2 coprocessor source register half is selected by specifying either the B (bottom) or T (top) in each of the xy positions of the mnemonic. The xy selection is the same for both the upper and lower halves of the 64-bit operand source registers, wRn and wRm. The resulting product for the upper and lower halves are then added to the two 32-bit accumulators which occupy the upper and lower halves of the 64-bit destination register wRd. A left shift correction is applied following the multiplication and saturation is provided with the addition for 32-bit signed operands. The N-qualifier optionally provides for a multiply-negate-accumulate operation instead of the default multiply-accumulate.

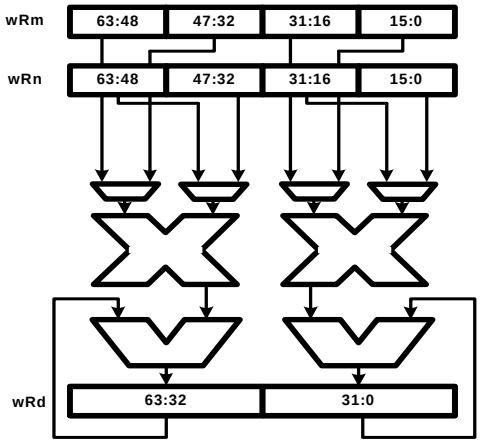
Usage WQMIA<T,B><T,B> {N}{Cond} wRd, wRn, wRm

Qualifiers
T – Obtain 16-bit signed Qm.n operands from the most significant words of wRn,wRm
B – Obtain 16-bit signed Qm.n operands from the least significant words of wRn, wRn
N – Perform multiply negate accumulate instead of multiply accumulate.

Operation

```
<operand1> = (T Specified in x position) ?wRn[31:16] : wRn[15:0];
<operand2> = (T Specified in y position) ? wRm[31:16] : wRm[15:0];
<operand3> = (T Specified in x position) ? wRn[63:48]: wRn[47:32];
<operand4> = (T Specified in y position) ? wRm[63:48]: wRm[47:32];

if (operand1 == 0x8000 && operand 2 == 0x8000)
   tmp1 = 0x7fffffff;
else
   tmp1 = (operand1*operand2) << 1;
if (operand3 == 0x8000 && operand 4== 0x8000)
   tmp2 = 0x7fffffff;
else
   tmp2 = (operand3*operand4) << 1;
if (N specified) then {
   wRd[31:0] = saturate(wRd[31:0] - tmp1, SS, 32);
   wRd[63:32] = saturate(wRd[63:32] - tmp2, SS, 32);
}
else {
   wRd[31:0] = saturate(wRd[31:0] + tmp1, SS, 32);
   wRd[63:32] = saturate(wRd[63:32] + tmp2, SS, 32);
}
```



Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				1nxy				wRn				wRd				0000				101			0	wRm			

Sub-Field Encoding

Qualifier	Field	Value
T	x	1
B	x	0
T	y	1
B	y	0
N	n	1
No N	n	0

SIMD Flags Does not effect the SIMD PSR flags (wCASF). The saturation flags (wCSSF) are set as a result of
the 32-bit Add or Subtract operation.

Note: Saturation must be done after final addition. The special case of 0x8000 times 0x8000 saturates to 0x7fffffff

Uses: This instruction is useful for implementing bit-exact voice codecs GSM EFR, GSM AMR etc. This instruction is also useful for lower precision fractional calculations.

3.2.43 WQMULM

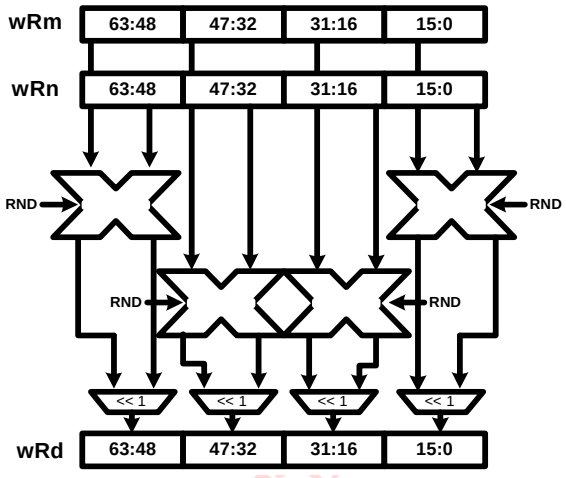
Overview Performs a vector multiplication of wRn and wRm on vectors of signed fractional 16-bit Qm.n values. The upper half of the results are stored in wRd with a left shift correction to renormalize the fractional data. Biased rounding, “round to nearest” is supported with the R-qualifier with truncation as the default behavior.

Usage WQMULM {R}{Cond} wRd, wRn, wRm

Qualifiers R – Round least significant bits into most significant bits.

Operation

```
m  = 15;
n  = 0;
if (R Specified) then{
  for ( i = 0; i < 4; i ++) {
      if (wRm[m:n] == 0x8000 && wRn[m:n] == 0x8000)
        wRd[m:n] = 0x7fff;
      else
        wRd[m:n] = (wRm[m:n]*wRn[m:n]+0x4000)>>15;
      m += 16;
      n += 16;
      }
  }
else{
   for ( i = 0; i < 4; i ++) {
      if (wRm[m:n] == 0x8000 && wRn[m:n] == 0x8000)
        wRd[m:n] = 0x7fff;
      else
        wRd[m:n] = (wRm[m:n]*wRn[m:n])>>15;
      m += 16;
      n += 16;
      }
```



Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				00r1				wRn				wRd				0000				100			0	wRm			

Sub-Field Encoding

Qualifier	Field	Value
R	r	1
No R	r	0

SIMD Flags Does not effect the SIMD PSR flags (wCASF). The saturation flags (wCSSF) are set as a result of the operation.

Note: The special case of 0x8000 times 0x8000 saturates to 0x7fff.

3.2.44 WQMULWM

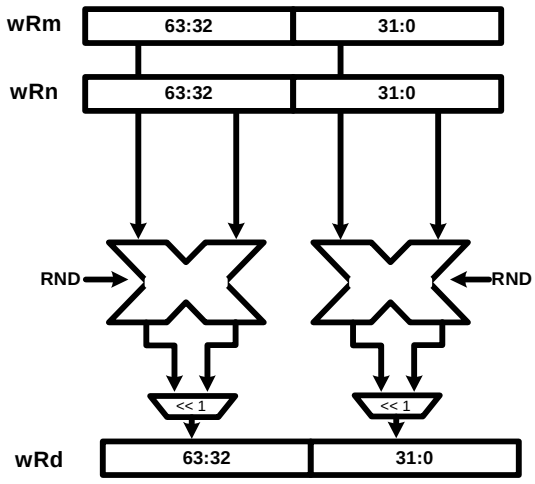
Overview Performs a vector multiplication of wRn and wRm on vectors of signed fractional 32-bit Qm.n values. The upper half of the results are stored in wRd with a left shift correction to renormalize the fractional data. Biased rounding, “round to nearest” is supported with the R-qualifier with truncation as the default behavior.

Usage WQMULWM <R>{Cond} wRd, wRn, wRm

Qualifiers R – Round least significant bits into most significant bits.

Operation

```
m  = 31;
n  = 0;
if (R Specified) then{
  for ( i = 0; i < 2; i ++) {
      if (wRm[m:n] == 0x80000000 && wRn[m:n] == 0x80000000)
         wRd[m:n] = 0x7fffffff;
      else
          wRd[m:n] = (wRm[m:n]*wRn[m:n]+0x4000)>>31;
      m += 32;
      n += 32;
      }
  }
else{
   for ( i = 0; i < 2; i ++) {
      if (wRm[m:n] == 0x80000000 && wRn[m:n] == 0x80000000)
         wRd[m:n] = 0x7fffffff;
      else
          wRd[m:n] = (wRm[m:n]*wRn[m:n])>>31;
      m += 32;
      n += 32;
      }
```



Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				11r0				wRn				wRd				0000				111			0	wRm			

Sub-Field Encoding

Qualifier	Field	Value
R	r	1
No R	r	0

SIMD Flags Does not effect the SIMD PSR flags (wCASF). The saturation flags (wCSSF) are set as a result of the operation.

Note: The special case of 0x80000000 times 0x80000000 saturates to 0x7fffffff.

3.2.45 WROR

Overview Performs vector logical rotate-right of wRn by wRm, or a control register (wCx), or a 5-bit immediate for vectors of 16-, 32-, or 64-bit data and places the result in the destination register, wRd.

Usage
```
WROR<H,W,D>{Cond} wRd, wRn, wRm
WROR<H,W,D>G{Cond} wRd, wRn, wCGRm
WROR <H,W,D> wRd, wRn, #I5
```

Qualifiers
H – 16-bit (Half Word) SIMD
W – 32-bit (Word) SIMD
D – 64-bit (Double)
G – wRm field specifies general purpose register to use for shift value

Operation
```
if (immediate specified)then
   shift_value = I5[4:0];
 else if (G Specified) then
   shift_value = wCGRm[7:0];
 else
   shift_value =wRm[7:0];

 if (H Specified) then
   wRd[half 3] = wRn[half 3] rotate_by (shift_value);
   wRd[half 2] = wRn[half 2] rotate_by (shift_value);
   wRd[half 1] = wRn[half 1] rotate_by (shift_value);
   wRd[half 0] = wRn[half 0] rotate_by (shift_value);
 else if (W Specified) then
   wRd[word 1] = wRn[word 1] rotate_by (shift_value);
   wRd[word 0] = wRn[word 0] rotate_by (shift_value);
 else if (D Specified) then
   wRd = wRn rotate_by (shift_value);
```

Exceptions None

Encoding WROR<H,W,D>{Cond} wRd, wRn, wRm
WROR<H,W,D>G{Cond} wRd, wRn, wCGRm

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				ww11				wRn				wRd				000g				010			0	wRm/wCGRn			

Sub-Field Encoding

Qualifier	Field	Value
H	ww	01
W	ww	10
D	ww	11
Reserved	ww	00
G	g	1
No G	g	0

Encoding WROR <H,W,D> wRd, wRn, #I5

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1111				1110				ww11				wRn				wRd				000			I5[4]	010			0	I5[3:0]			

Sub-Field Encoding

Qualifier	Field	Value
H	ww	01
W	ww	10
D	ww	11
Reserved	ww	00

The encoding for the immediate value #32, is I5[4:0] = 0.

SIMD Flags Clears C and V flags, and sets Z and N according to the result. Does not effect the saturation flags (wCSSF).

Flag	Action
N	Set on Final Result
Z	Set on Final Result
C	Cleared
V	Cleared

Note: Specifying an illegal value for wCGRm when the G-qualifier is used produces an unpredictable result.

The valid range for shift by immediate is 0 thru 32. If the immediate exceeds 31 the immediate is truncated to 5-bits. If an immediate value of #0 is specified, the assembler performs the following mapping:

WRORH wRd, wRn, #0 maps to WRORH wRd, wRn, #16
WRORW wRd, wRn, #0 maps to WRORW wRd, wRn, #32
WRORD wRd, wRn, #0 maps to WOR wRd, wRn, wRn

3.2.46 WSAD

Overview Performs the sum of absolute differences of wRn and wRm, and accumulates the result with wRd; can be applied to 8-bit or 16-bit unsigned data vectors.

Usage WSAD<B,H>{Z}{Cond} wRd, wRn, wRm

Qualifiers
B – 8-bit (byte) SIMD
H – 16-bit (half word) SIMD
Z – Zero accumulator first

Operation

```
wRd[word 1] = 0;
 if (B specified)then
wRd[word 0] = (Z Specified) ? 0: wRd[word 0]
      + abs(wRn[byte 7] - wRm[byte 7])
      + abs(wRn[byte 6] - wRm[byte 6])
      + abs(wRn[byte 5] - wRm[byte 5])
      + abs(wRn[byte 4] - wRm[byte 4])
      + abs(wRn[byte 3] - wRm[byte 3])
      + abs(wRn[byte 2] - wRm[byte 2])
      + abs(wRn[byte 1] - wRm[byte 1])
      + abs(wRn[byte 0] - wRm[byte 0]);
else if (H specified)then
   wRd[word 0] = Z Specified) ? 0: wRd[word 0]
      + abs(wRn[half 3] - wRm[half 3])
      + abs(wRn[half 2] - wRm[half 2])
      + abs(wRn[half 1] - wRm[half 1])
      + abs(wRn[half 0] - wRm[half 0]);
```

Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				0h0z				wRn				wRd				0001				001			0	wRm			

Sub-Field Encoding

Qualifier	Field	Value
B	h	0
H	h	1
Z	z	1
No Z	z	0

SIMD Flags Does not effect the SIMD PSR flags (wCASF) or saturation flags (wCSSF)

Note: Accumulates the result to 32-bits only

3.2.47 WSHUFH

Overview Select (shuffle) 16-bit data values in destination register, wRd, from 16-bit fields in source register specified by the 8-bit immediate value.

Usage WSHUFH{Cond} wRd, wRn, #Imm8

Qualifiers None

Operation

```
wRd[Half 0]=wRn[Half Imm8[1:0]];
wRd[Half 1]=wRn[Half Imm8[3:2]];
wRd[Half 2]=wRn[Half Imm8[5:4]];
wRd[Half 3]=wRn[Half Imm8[7:6]];
```

Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				ddcc				wRn				wRd				0001				111			0	bbaa			

Sub-Field Encoding

Field	Value
bbaa	Imm8[3:0]
ddcc	Imm8[7:4]

SIMD Flags Sets Z and N flag, clears C, and V; sets SIMD16 flags only; does not effect the saturation flags (wCSSF).

Flag	Action
N	Set on Final Result
Z	Set on Final Result
C	Cleared
V	Cleared

3.2.48 WSLL

Overview Performs vector logical shift-left of wRn by wRm , or a control register (wCx), or a 5-bit immediate for vectors of 16-, 32-, or 64-bit data and places the result in wRd.

Usage
```
WSLL<H,W,D>{Cond} wRd, wRn, wRm
WSLL<H,W,D>G {Cond} wRd, wRn, wCGRn
WSLL<H,W,D> wRd, wRn, #I5
```

Qualifiers
- H – 16-bit (Half Word) SIMD
- W – 32-bit (Word) SIMD
- D – 64-bit (Double)
- G – wRm field specifies general purpose register to use for shift value

Operation
```
if (immediate specified)then
   shift_value = I5[4:0];
 else if (G Specified) then
   shift_value = wCGRm[7:0];
 else
   shift_value =wRm[7:0];

 if (H Specified)then
   wRd[half 3] = wRn[half 3] << shift_value;
   wRd[half 2] = wRn[half 2] << shift_value;
   wRd[half 1] = wRn[half 1] << shift_value;
   wRd[half 0] = wRn[half 0] << shift_value;
 else if (W Specified) then
   wRd[word 1] = wRn[word 1] << shift_value;
   wRd[word 0] = wRn[word 0] << shift_value;
 else if (D Specified) then
   wRd = wRn << shift_value;
```

Exceptions None

Encoding
```
WSLL<H,W,D>{Cond} wRd, wRn, wRm
WSLL<H,W,D>G {Cond} wRd, wRn, wCGRn
```

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				ww01				wRn				wRd				000g				010			0	wRm/wCGRn			

Sub-Field Encoding

Qualifier	Field	Value
H	ww	01
W	ww	10
D	ww	11
Reserved	ww	00
G	g	1
No G	g	0

Encoding WSLL<H,W,D> wRd, wRn, #I5

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1111				1110				ww01				wRn				wRd				000			I5[4]	010			0	I5[3:0]			

Sub-Field Encoding

Qualifier	Field	Value
H	ww	01
W	ww	10
D	ww	11
Reserved	ww	00

The encoding for the immediate value #32, is I5[4:0] = 0.

SIMD Flags The V, C flags are cleared; the Z and N flag are set as a result of the operation,; does not effect the saturation flags (wCSSF).

Flag	Action
N	Set on Final Result
Z	Set on Final Result
C	Cleared
V	Cleared

Note: Zero is shifted in from the right; if for the register versions, the shift value > 63 (for D qualifier), 31 (for W qualifier) or 15 (for H qualifier), the destination register will contain all zeros. Specifying an illegal value for wCGRm when the G qualifier is used produces an unpredictable result.

The valid range for shift by immediate is 0 thru 32. If the immediate exceeds 31 the immediate is truncated to 5-bits. If an immediate value of #0 is specified, the assembler performs the following mapping:

WSLLH wRd, wRn, #0 maps to WRORH wRd, wRn, #16
WSLLW wRd, wRn, #0 maps to WRORW wRd, wRn, #32
WSLLD wRd, wRn, #0 maps to WOR wRd, wRn, wRn

.

3.2.49 WSRA

Overview Performs vector arithmetic shift-right of wRn by wRm, or a control register (wCx), or a 5-bit immediate for vectors of 16-, 32-, or 64-bit data and places the result in wRd.

Usage
```
WSRA<H,W,D>{Cond} wRd, wRn, wRm
WSRA<H,W,D>G{Cond} wRd, wRn, wCGRn
WSRA <H,W,D> wRd, wRn, #I5
```

Qualifiers
- H – 16-bit (Half Word) SIMD
- W – 32-bit (Word) SIMD
- D – 64-bit (Double)
- G – wRm field specifies general purpose register to use forshift value

Operation
```
if (immediate specified)then
   shift_value = I5[4:0];
 else if (G Specified) then
   shift_value = wCGRm[7:0];
 else
   shift_value =wRm[7:0];

 if (H Specified)then
   wRd[half 3] = wRn[half 3] >>> shift_value;
   wRd[half 2] = wRn[half 2] >>> shift_value;
   wRd[half 1] = wRn[half 1] >>> shift_value;
   wRd[half 0] = wRn[half 0] >>> shift_value;
 else if (W Specified) then
   wRd[word 1] = wRn[word 1] >>> shift_value;
   wRd[word 0] = wRn[word 0] >>> shift_value;
 else if (D Specified) then
   wRd = wRn >>> shift_value;
```

Exceptions None

Encoding
```
WSRA<H,W,D>{Cond} wRd, wRn, wRm
WSRA<H,W,D>G{Cond} wRd, wRn, wCGRn
```

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				ww00				wRn				wRd				000g				010			0	wRm/wCGRn			

Sub-Field Encoding

Qualifier	Field	Value
H	ww	01
W	ww	10
D	ww	11

Qualifier	Field	Value
Reserved	ww	00
G	g	1
No G	g	0

Encoding WSRA <H,W,D> wRd, wRn, #I5

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1111				1110				ww00				wRn				wRd				000			I5[4]	010			0	I5[3:0]			

Sub-Field Encoding

Qualifier	Field	Value
H	ww	01
W	ww	10
D	ww	11
Reserved	ww	00

The encoding for the immediate value #32, is I5[4:0] = 0.

SIMD Flags The C and V flags are cleared; the and Z and N flags are set as a result of the operation; does not effect the saturation flags (wCSSF).

Flag	Action
N	Set on Final Result
Z	Set on Final Result
C	Cleared
V	Cleared

Note: The original *sign* bit is shifted in from the left. If the shift value in the register versions is > 63 (for D qualifier), 31 (for W qualifier) or 15 (for H qualifier), the destination register will contain all *sign* bits. Specifying an illegal value for wCGRm when the G-qualifier is used produces an unpredictable result.

The valid range for shift by immediate is 0 thru 32. If the immediate exceeds 31 the immediate is truncated to 5-bits. If an immediate value of #0 is specified, the assembler performs the following mapping:

WSRAH wRd, wRn, #0 maps to WRORH wRd, wRn, #16
WSRAW wRd, wRn, #0 maps to WRORW wRd, wRn, #32
WSRAD wRd, wRn, #0 maps to WOR wRd, wRn, wRn

3.2.50 WSRL

Overview Performs vector logical shift-right of wRn by wRm, or a control register (wCx), or a 5-bit immediate for vectors of 16-, 32-, or 64-bit data and places the result in wRd.

Usage
```
WSRL<H,W,D>{Cond} wRd, wRn, wRm
WSRL<H,W,D>G{Cond} wRd, wRn, wCGRn
WSRL<H,W,D> wRd, wRn, #I5
```

Qualifiers
H – 16-bit (Half Word) SIMD
W – 32-bit (Word) SIMD
D – 64-bit (Double)
G – wRm field specifies general purpose register to use for shift value

Operation
```
If (immediate specified)then
  shift_value = I5[4:0];
else if (G Specified) then
  shift_value = wCGRm[7:0];
else
  shift_value =wRm[7:0];

if (H Specified)then
  wRd[half 3] = wRn[half 3] >> shift_value;
  wRd[half 2] = wRn[half 2] >> shift_value;
  wRd[half 1] = wRn[half 1] >> shift_value;
  wRd[half 0] = wRn[half 0] >> shift_value;
else if (W Specified) then
  wRd[word 1] = wRn[word 1] >> shift_value;
  wRd[word 0] = wRn[word 0] >> shift_value;
else if (D Specified) then
  wRd = wRn >> shift_value;
```

Exceptions None

Encoding
```
WSRL<H,W,D>{Cond} wRd, wRn, wRm
WSRL<H,W,D>G{Cond} wRd, wRn, wCGRn
```

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				ww10				wRn				wRd				000g				010			0	wRm/wCGRn			

Sub-Field Encoding

Qualifier	Field	Value
H	ww	01
W	ww	10
D	ww	11

Qualifier	Field	Value
Reserved	ww	00
G	g	1
No G	g	0

Encoding WSRL <H,W,D> wRd, wRn, #I5

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1111				1110				ww10				wRn				wRd				000			I5[4]	010			0	I5[3:0]			

Sub-Field Encoding

Qualifier	Field	Value
H	ww	01
W	ww	10
D	ww	11
Reserved	ww	00

The encoding for the immediate value #32, is I5[4:0] = 0.

SIMD Flags The Cand V flags are cleared; the Z and N flags are set as a result of the operation; does not effect the saturation flags (wCSSF).

Flag	Action
N	Set on Final Result
Z	Set on Final Result
C	Cleared
V	Cleared

Note: Zero is shifted in from the left. If the shift value in the register versions is > 63 (for D qualifier), 31 (for W qualifier) or 15 (for H qualifier), the destination register will contain all zeros. Specifying an illegal value for wCGRm when the G qualifier is used produces an unpredictable result.

The valid range for shift by immediate is 0 thru 32. If the immediate exceeds 31 the immediate is truncated to 5-bits. If an immediate value of #0 is specified, the assembler performs the following mapping:

WSRLH wRd, wRn, #0 maps to WRORH wRd, wRn, #16
WSRLW wRd, wRn, #0 maps to WRORW wRd, wRn, #32
WSRLD wRd, wRn, #0 maps to WOR wRd, wRn, wRn

3.2.51 WSTR

Overview Performs a store from the source register (either wRm or wCx) into the memory location whose address is specified by the Rn register and corresponding address modifiers; 8-, 16-, 32-, and 64-bit data values can be stored from the data registers (wRm), and 32-bits only can be stored from the control registers (wCx). The immediate offset addressing mode is provided for both data and control register store operations. The register offset addressing mode is provided for 64-bit store operations from the data registers. A left shift may be optionally applied to the offset when using the register offset addressing mode.

Usage
```
WSTR<B,H,W,D>{Cond} wRm, <Imm_offset_address_mode>
WSTRD wRm, <Reg_offset_address_mode>
WSTRW wCx, <Imm_offset_address_mode>
```

Qualifiers
B – Store byte
H – Store half
W – Store word
D – Store double

Operation
```
if (B specified) then
   Mem[<Imm_offset_address_mode>] = wRm[7:0];
else if (h specified) then
   Mem[<Imm_offset_address_mode>] = wRm[15:0];
else if (W specified) then
   Mem[<Imm_offset_address_mode>] = wRm[31:0];
else if (D specified) then
   Mem[<Imm_offset_address_mode>or <Reg_offset_address_mode>] = wRm[63:0];
```

Exceptions None

Encoding WSTR<B,H,W,D> {Cond} wRm

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				110			P	U	N	W	0	Rn				wRm				000			M	offset_8							

Sub-Field Encoding WSTR<B,H,W,D>{Cond} wRm

Qualifier	Field	Value
B	N,M	0,0
H	N,M	1,0
W	N,M	0,1
D	N,M	1,1

P,U,W and offset_8 encoding is as specified in the ARM* V5 architecture

Sub-Field Encoding<Imm_offset_address_mode>

The following standard ARM* V5 coprocessor immediate offset address modes are supported:

Table 3-4. Immediate offset address mode encoding

P	U	W	Mnemonic	Description
0	0	1	[Rn], #- AS_offset	Post-index, negative offset, always writeback
0	1	1	[Rn], #+ AS_offset	Post-index, positive offset, always writeback
1	0	0	[Rn, #- AS_offset]	Pre-index, negative offset
1	0	1	[Rn, #- AS_offset]!	Pre-index, negative offset, with writeback
1	1	0	[Rn, #+ AS_offset]	Pre-index, positive offset
1	1	1	[Rn, #+ AS_offset]!	Pre-index, positive offset, with writeback

Note: The data address is generated by the Intel XScale® core where <Imm_offset_address_mode> is allowable modes specified in ARM* store coprocessor. The immediate offset used in the address calculation is always specified to the assembler in bytes. However ,for the W and D qualifiers the offset must be a word multiple (that is, divisible by 4). This allows the word- and double-store variants to have a greater offset range (0 – 0x3FC) than the byte- and half-word store variants (0 – 0xFF). Refer to Table 3-5 for more information.

Table 3-5. Address Mode Assembler Offset Specification and Instruction Encoding

Qualifier	Assembler Offset Units (AS_offset)	Assembler Offset Restriction	Instruction encoding (offset_8)
B	Byte	AS_offset =< 0xFF	AS_offset[7:0]
H	Byte	AS_offset =< 0xFF	AS_offset[7:0]
W	Byte	AS_offset =< 0x3FC & AS_offset mod 4 = 0	AS_offset[9:2]
D	Byte	AS_offset =< 0x3FC & AS_offset mod 4 = 0	AS_offset[9:2]

Encoding WSTRD wRm, <Reg_offset_address_mode>

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1111				110			P	U	1	W	0	Rn				wRd				000			1	Imm4				Rm			

Sub-Field Encoding<Reg_offset_address_mode>

The following register offset address modes are supported:

Table 3-6. Register offset address mode encoding

P	U	W	Mnemonic	Description
0	0	1	[Rn], - Rm {,LSL #Imm4}	Post-index, negative Rm, always writeback, with optional shift of Rm
0	1	1	[Rn], + Rm {,LSL #Imm4}	Post-index ,positive Rm, always writeback, with optional shift of Rm
1	0	0	[Rn, - Rm {,LSL #Imm4}]	Pre-index, negative Rm, with optional shift of Rm
1	0	1	[Rn, - Rm{,LSL #Imm4}]!	Pre-index ,negative Rm, with writeback, with optional shift of Rm
1	1	0	[Rn, + Rm {,LSL #Imm4}]	Pre-index, positive Rm, with optional shift of Rm
1	1	1	[Rn, + Rm {,LSL #Imm4}]!	Pre-index ,positive Rm, with writeback, with optional shift of Rm

Note: "{}" indicates optional mnemonic

Encoding WSTRW wCx

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1111				110			P	U	0	W	0	Rn				wCx				0001				offset_8							

Note: Specifying an illegal value for wCx causes an undefined instruction exception

Sub-Field EncodingP,U,W, and offset_8 encoding is as specified in the ARM* V5 architecture

SIMD Flags Does not effect the SIMD PSR flags (wCASF) or saturation flags (wCSSF)

Note: Stores from Marvell® Wireless MMX™ 2 coprocessor control registers are word only and cannot be conditionally executed. Register offset is always specified in bytes and has no restrictions apart from alignment of the final address. (See below)

If an offset is required that does not meet the above restrictions, the address must be modified using a separate instruction (for example, add/subtract).

Addresses should be aligned at the byte/half/word boundary of the data item being stored. Lower *address* bits should be cleared if the unaligned address traps is enabled on the main core, otherwise unaligned address trap is taken.

If unaligned traps are disabled the following occurs.

Qualifier	Unaligned Behavior
B	never unaligned
H,W,D	unpredictable

3.2.52 WSUB

Overview Performs vector subtraction of wRm from wRn for vectors of 8-, 16-, or 32-bit signed or unsigned data, and places the result in wRd. Saturation can be specified as signed, unsigned, or no saturation.

Usage WSUB<B,H,W>{US,SS}{Cond} wRd, wRn, wRm

Qualifiers
B – 8-bit (byte) SIMD
H – 16-bit (half word) SIMD
W – 32-bit (word) SIMD
SS – Signed saturation
US – Unsigned saturation

Operation

```
if (B Specified) then {
  wRd[byte 7] = saturate(wRn[byte 7] - wRm[byte 7], {US,SS}, 8);
  wRd[byte 6] = saturate(wRn[byte 6] - wRm[byte 6], {US,SS}, 8);
  wRd[byte 5] = saturate(wRn[byte 5] - wRm[byte 5], {US,SS}, 8);
  wRd[byte 4] = saturate(wRn[byte 4] - wRm[byte 4], {US,SS}, 8);
  wRd[byte 3] = saturate(wRn[byte 3] - wRm[byte 3], {US,SS}, 8);
  wRd[byte 2] = saturate(wRn[byte 2] - wRm[byte 2], {US,SS}, 8);
  wRd[byte 1] = saturate(wRn[byte 1] - wRm[byte 1], {US,SS}, 8);
  wRd[byte 0] = saturate(wRn[byte 0] - wRm[byte 0], {US,SS}, 8);
 }
 else if (H Specified) then {
  wRd[half 3] = saturate(wRn[half 3] - wRm[half 3], {US,SS}, 16);
  wRd[half 2] = saturate(wRn[half 2] - wRm[half 2], {US,SS}, 16);
  wRd[half 1] = saturate(wRn[half 1] - wRm[half 1], {US,SS}, 16);
  wRd[half 0] = saturate(wRn[half 0] - wRm[half 0], {US,SS}, 16);
 }
 else if (W Specified) then {
  wRd[word 1] = saturate(wRn[word 1] - wRm[word 1], {US,SS}, 32);
  wRd[word 0] = saturate(wRn[word 0] - wRm[word 0], {US,SS}, 32);
 }}
```

Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				wwss				wRn				wRd				0001				101			0	wRm			

Sub-Field Encoding

Qualifier	Field	Value
B	ww	00
H	ww	01
W	ww	10

Qualifier	Field	Value
US	ss	01
SS	ss	11
No Saturation	ss	00

SIMD Flags Sets SIMD flags (wCASF) on the result of the subtraction wRn-wRm. If US/SS is specified and saturation occurs, the SIMD flags (wCASF) are set on the post-saturation (final result) value. The saturation flags (wCSSF) are set if the US/SS qualifier is specified and the corresponding fields saturates.

Flag	No Saturation Specified	US/SS specified and no Result Saturation	US/SS specified and Result Saturates
N	Set on Final Result	Set on Final Result	Set on Final Result
Z	Set on Final Result	Set on Final Result	Set on Final Result
C	Set on Final Result	Set on Final Result	Cleared
V	Set on Final Result	Set on Final Result	Cleared

3.2.53 WSUBADDHX

Overview Performs complex vector subtraction/addition of wRn and wRm for vectors of 16-bit data, and places the result in wRd. The four operands from each of the source registers are alternately added and subtracted using a cross selection in each of the parallel operations.The result of the operation is saturated to the signed limits, for example, 0x8000 to 0x7fff or -32768 to 32767.

Usage WSUBADDHX {Cond} wRd, wRn, wRm

Qualifiers None

Operation

```
wRd[half 3] = saturate(wRn[half 3] + wRm[half 2], SS, 16);
wRd[half 2] = saturate(wRn[half 2] - wRm[half 3], SS, 16);
wRd[half 1] = saturate(wRn[half 1] + wRm[half 0], SS, 16);
wRd[half 0] = saturate(wRn[half 0] - wRm[half 1], SS, 16);
```

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				1101				wRn				wRd				0001				110			0	wRm			

Sub-Field Encoding

SIMD Flags Sets the SIMD16 (wCASF) flags as shown below. The saturation flags (wCSSF) are set if the corresponding fields saturates.

Flag	No Saturation	Result Saturates
N	Set on Final Result	Set on Final Result
Z	Set on Final Result	Set on Final Result
C	Set on Final Result	Cleared
V	Set on Final Result	Cleared

Note: This instruction is useful for performing radix-2 and radix-4 butterflies on real or complex data structures. The application support include DCT and FFT calculations common to voice, audio, and video applications.

3.2.54 WUNPCKEH

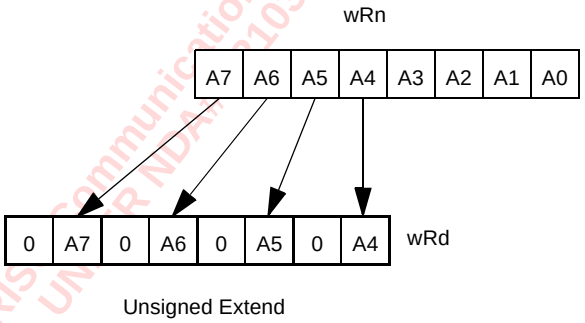
Overview Unpacks 8-bit, 16-bit, or 32-bit data from the top half of wRn (the source register), and either zero- or signed- extends each field, and places the result into the destination register, wRd.

Usage WUNPCKEH<U,S><B,H,W>{Cond} wRd, wRn

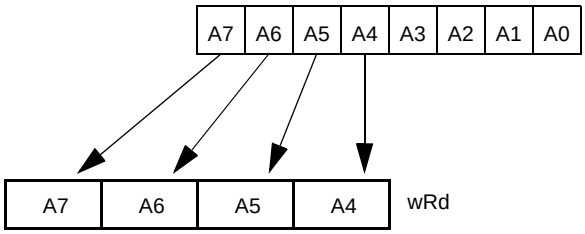
Qualifiers
B – 8-bit (byte) SIMD
H – 16-bit (half word) SIMD
W – 32-bit (word) SIMD

U – Unsigned
S – Sign extend

Operation WUNPCKEHUB wRd,wRn



Unsigned Extend



Signed Extend

Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				wws0				wRn				wRd				0000				110			0	0000			

Sub-Field Encoding

Qualifier	Field	Value
B	ww	00
H	ww	01
W	ww	10
U	s	0
S	s	1

Note: The combination ww= 11 is used for other instruction encodings

SIMD Flags Sets N and Z flags, depending on the result fields, and clears C and V flags.;

Flag	Action
N	Set on Final Result
Z	Set on Final Result
C	Cleared
V	Cleared

SIMD flags are set on the SIMD width of the result (see below)

Qualifier	SIMD flags
B	SIMD16
H	SIMD32
W	SIMD64

3.2.55 WUNPCKIH

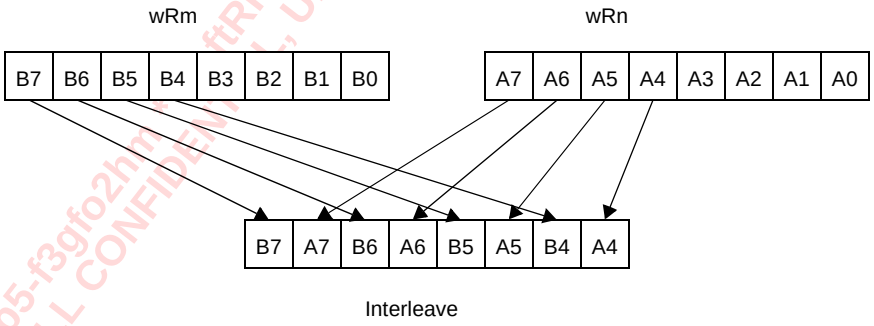
Overview Unpacks either 8-bit, 16-bit, or 32 bit data from the top half of wRn, interleaves with the top half of wRm, and places the result into the destination register, wRd.

Usage WUNPCKIH<B,H,W>{Cond} wRd, wRn, wRm

Qualifiers
B – 8-bit (byte) SIMD
H – 16-bit (half word) SIMD
W – 32-bit (word) SIMD

Operation

WUNPCKIHB wRd,wRn,wRm



Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				ww01				wRn				wRd				0000				110			0	wRm			

Sub-Field Encoding

Qualifier	Field	Value
B	ww	00
H	ww	01
W	ww	10

Note: The combination ww= 11 is not valid for this instruction.

SIMD Flags Sets N and Z flags, depending on the result fields, and clears C and V flags

Flag	Action
N	Set on Final Result
Z	Set on Final Result
C	Cleared
V	Cleared

The SIMD flags are set on the SIMD width of the result (see table below).

Qualifier	SIMD flags
B	SIMD8
H	SIMD16
W	SIMD32

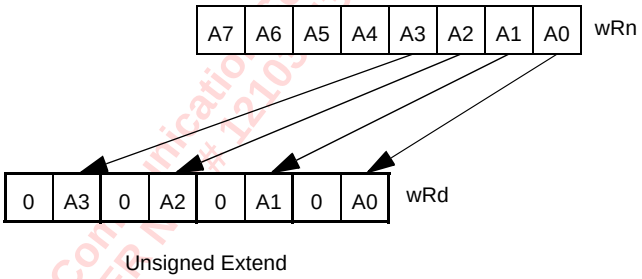
3.2.56 WUNPCKEL

Overview Unpacks either 8-bit, 16-bit, or 32 bit data from the lower half of wRn (the source register), and either zero- or sign- extends each field, and places the result into the destination register, wRd.

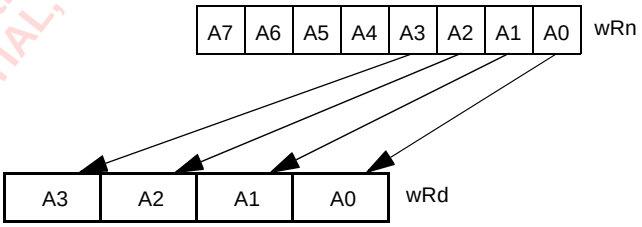
Usage WUNPCKEL<U,S><B,H,W>{Cond} wRd, wRn

Qualifiers
B – 8-bit (byte) SIMD
H – 16-bit (half word) SIMD
W – 32-bit (word) SIMD
U – Unsigned
S – Sign extend

Operation WUNPCKELB wRn



Unsigned Extend



Signed Extend

Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				wws0				wRn				wRd				0000				111			0	0000			

Sub-Field Encoding

Qualifier	Field	Value
B	ww	00
H	ww	01
W	ww	10
U	s	0
S	s	1

Note: The combination ww= 11 is not valid for this instruction.

SIMD Flags Sets N and Z flags, depending on the result fields, and clears C and V flags.

Flag	Action
N	Set on Final Result
Z	Set on Final Result
C	Cleared
V	Cleared

SIMD flags are set on the SIMD width of the result (see below).

Qualifier	SIMD flags
B	SIMD16
H	SIMD32
W	SIMD64

3.2.57 WUNPCKIL

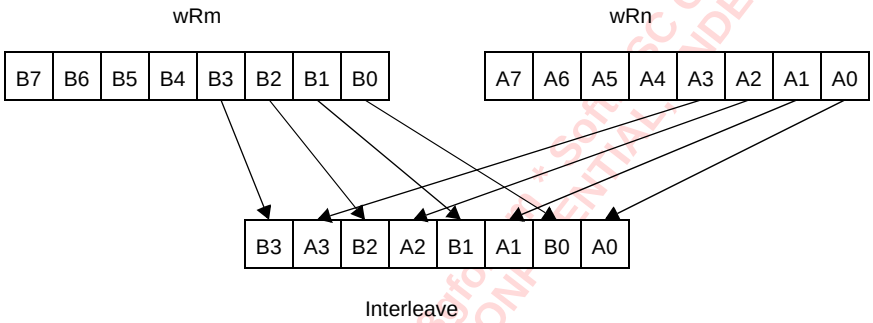
Overview Unpacks either 8-bit, 16-bit, or 32 bit data from the lower half of wRn and the lower half of wRm, and places the result into the destination register, wRd.

Usage WUNPCKIL<B,H,W>{Cond} wRd, wRn,wRm

Qualifiers
B – 8-bit (byte) SIMD
H – 16-bit (half word) SIMD
W – 32-bit (word) SIMD

Operation

WUNPCKILB wRd, wRn,wRm



Exceptions None

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				ww01				wRn				wRd				0000				111			0	wRm			

Sub-Field Encoding

Qualifier	Field	Value
B	ww	00
H	ww	01
W	ww	10
Reserved	ww	11

SIMD Flags Sets N and Z flags, depending on the result fields, and clears C and V flags.

Flag	Action
N	Set on Final Result
Z	Set on Final Result
C	Cleared
V	Cleared

SIMD flags are set on the SIMD width of the result (see table below).

Qualifier	SIMD flags
B	SIMD8
H	SIMD16
W	SIMD32

3.2.58 WXOR

Overview Performs a bitwise logical “XOR” between wRn and wRm, and places the result in wRd.

Usage WXOR{Cond} wRd, wRn, wRm

Qualifiers None

Operation wRd = wRn ^ wRm

Exceptions

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				0001				wRn				wRd				0000				000			0	wRm			

Sub-Field Encoding None

SIMD Flags Sets SIMD64 flags only; C and V flags are cleared and the N and Z flags are set according to the result; does not affect the saturation flags (wCSSF).

Flag	Action
N	Set on Final Result
Z	Set on Final Result
C	Cleared
V	Cleared

3.2.59 WZERO

Overview This pseudo-instruction zeros the Marvell® Wireless MMX™ 2 coprocessor destination register, wRd; this instruction is a form of the WANDN instruction.

Usage WZERO{Cond} wRd

Qualifiers None

Operation WANDN wRd, wRd, wRd

Exceptions None

Encoding see WANDN

Sub-Field Encoding see WANDN

SIMD Flags see WANDN

Note: This is a pseudo-instruction that maps onto another Marvell® Wireless MMX™ 2 coprocessor instruction and is provided for convenience

Integer Instruction Encoding Summary Appendix A

This appendix gives an overview of the instruction encoding for the entire Marvell® Wireless MMX™ 2 coprocessor instruction set.

A.1 Coprocessor Data (CDP) Instructions

Encoded instructions that operate on the coprocessor register file use the ARM* CDP format as show below.

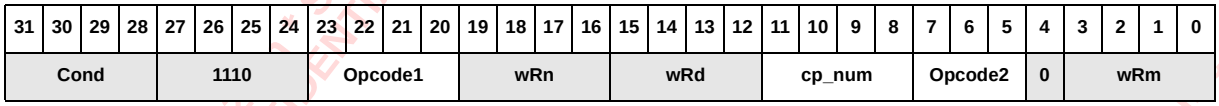


Table A-1 summarizes the encoding for all CDP instructions

Table A-1. CDP Instruction Encoding (Sheet 1 of 3)

Type	Instruction	Opcode2	Opcode1	cp_num	wRm
Misc	WOR	000	0000	0000	—
	WXOR		0001	0000	—
	WAND		0010	0000	—
	WANDN		0011	0000	
	WAVG2		1h0r	0000	—
	WAVG4		010r	0000	—
	reserved		1-1-,011-	0000	—
Align	WALIGNI	001	0vvv	0000	—
	WALIGNR		10vv	0000	—
	reserved		11--	0000	—
Shift	WSRA	010	ww00	000g	—
	WSLL		ww01	000g	—
	WSRL		ww10	000g	—
	WROR		ww11	000g	—
	reserved		00--	000g	—

Table A-1. CDP Instruction Encoding (Sheet 2 of 3)

Type	Instruction	Opcode2	Opcode1	cp_num	wRm
Compare	WCMPEQ	011	ww00	0000	—
	WCMPGT		wws1	0000	—
	reserved		--10, 0-10, 1010	0000	—
PACK	WPACK	100	wwss	0000	—
	WMERGE		cba0	0000	—
Multiply/MAC	WQMULM		00r1	0000	—
	WMIAxy	101	0nxy	0000	—
	WQMIAxy		1nxy	0000	—
	WMULW	110	1fsr	0000	—
UNPACK	reserved		0-11	0000	—
	WUNPCKEH		wws0	0000	—
	WUNPCKIH		ww01	0000	—
	reserved	111	--11, 1101	0000	—
	WUNPCKEL		wws0	0000	—
	WUNPCKIL		ww01	0000	—
Multiply/MAC	WQMULW		11r0	0000	
	WMUL	000	rrsf	0001	—
	WMAC		01sz	0001	—
	WMADD		nnsx	0001	—
	WMIAWxy	001	1nxy	0001	—
Difference	WSAD		0h0z	0001	—
	reserved		0-1-	0001	
Max/Min	WMAX	011	wws0	0001	—
	WMIN		wws1	0001	—
	reserved		11--	0001	—

Table A-1. CDP Instruction Encoding (Sheet 3 of 3)

Type	Instruction	Opcode2	Opcode1	cp_num	wRm
Arithmetic	WADD	100	wwss	0001	—
	reserved		11--, 0010	0001	—
	WSUB	101	wwss	0001	—
	reserved		11--,0-10	0001	—
	WADDSUBHX		1010	0001	—
	WSUBADDHX	110	1101	0001	
	WACC		ww00	0001	—
	WABS		ww10	0001	—
	WABSDIFF		ww01	0001	—
	reserved		--11, 11-0	0001	—
Shuffle	WSHUF	111	ddcc	0001	bbaa

A.2 Coprocessor Data Transfer Instructions

Instructions that transfer data between the ARM* and coprocessor register file use the MRC, MCR, MRRC or MCRR instruction format (depending on direction and size).

A.2.1 Transfers to Coprocessor Register (MCR)

Transfers to the coprocessor register use the MCR format instruction as show below.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				Opcode1			0	wRd				Rn				cp_num				Opcode2			1	wRm			

Table A-2 shows coprocessor data transfer instruction encoding.

Table A-2. Coprocessor Data Transfer Instruction Encoding (Sheet 1 of 2)

Instruction	Opcode1	Opcode2	cp_num	wRm
TMCR	000	000	0001	0000
TMIA	001	See Special case below		
TBCST	010	ww0	0000	0000
TINSR	011	ww0		0bbb

Table A-2. Coprocessor Data Transfer Instruction Encoding (Sheet 2 of 2)

Instruction	Opcode1	Opcode2	cp_num	wRm
reserved	1--,-1-,--1	---	0001	—
	—	1--,-1-,--1		—
	—	—		1---,-1--,--1-,---1
	010	--1	0000	—
	011	--1		—
	1--	—		—

For MIA, this instruction format applies:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				Opcode1			0	OpCode2				Rs				cp_num[3:1]			wRd				1	Rm			

Table A-3 shows coprocessor data transfer multiply-accumulate instruction encoding.

Table A-3. Coprocessor Data Transfer Multiply-Accumulate Instruction Encoding

Instruction	Opcode1	Opcode2	cp_num[3:1]
TMIA	001	0000	000
TMIAPH	001	1000	000
TMIAXY	001	11xy	000
reserved	001	01--	000

A.2.2 Transfers to Coprocessor Register (MCRR)

The TMCRR instruction uses this format:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				11000100								RdHi				RdLo				cp_num				Opcode2				wRn			

Table A-4 shows transfers to coprocessor register instruction encoding.

Table A-4. Transfers to Coprocessor Register Instruction Encoding

Instruction	Opcode2	cp_num
TMCRR	0000	0000
reserved	0001 – 1111	0000
	—	0001

A.2.3 Transfers from Coprocessor Register (MRC)

Transfers from the coprocessor register using the MRC format are:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1110				Opcode1			1	wRn				Rd				cp_num				Opcode2			1	wRm			

Table A-5 shows transfers from coprocessor register subfield instruction encoding (MRC).

Table A-5. Transfers From Coprocessor Register Subfield Instruction Encoding (MRC)

Instruction	Opcode2	Opcode1	cp_num	wRm
TMRC	000	000	0001	0000
TMOVMSK	001	ww0	0000	0000
TANDC	001	ww0	0001	0000
TORC	010	ww0	0001	0000
TORVSC	100	ww0	0001	0000
TEXTRC	011	ww0	0001	0bbb
TEXTRM		ww0	0000	sbbb
reserved	000	1--,-1-,--1	0001	—
	000	000		1---,-1--,--1-,---1
	001	--1, 110		—
	010	--1, 110		—
	011	--1, 110		—
	100	--1, 110		—
	101,11-	-		—
	011	--1, 110	0000	—
	001	--1, 110		—
	1--,0-0	-		

A.2.4 Transfers from Coprocessor Register (MRRC)

The TMRRC instruction uses this format:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				11000101								RdHi				RdLo				cp_num				Opcode2				wRn			

Table A-5 shows transfers from coprocessor register subfield instruction encoding (MRRC).

Table A-6. Transfers from Coprocessor Register Subfield Instruction Encoding (MRRC)

Instruction	Opcode2	cp_num
TMRRC	0000	0000
reserved	0001 - 1111	0000
	—	0001

A.3 Load Store Instructions

A.3.1 Load/Store to Marvell® Wireless MMX™ 2 Coprocessor Data Registers

This instruction uses the ARM* load/store coprocessor instruction so flags are compatible with the N-flag and 1 bit of the coprocessor number encode the 4 size options (byte, half, word, double word).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				110			P	U	N	W	L	Rn				wRd				000m				offset_8								

A.3.2 Load/Store to Marvell® Wireless MMX™ 2 Coprocessor Control Registers

This instruction uses the ARM* LDC2/STC2 instruction class, which cannot be conditionally executed.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1111				110			P	U	N	W	L	Rn				wRd				0001				offset_8								

A.3.3 Load/Store with Register Offset to Marvell® Wireless MMX™ 2 Coprocessor Data Registers

This instruction uses the ARM* LDC2/STC2 instruction class, which cannot be conditionally executed.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1111				110			P	U	1	W	L	Rn				wRd				000			1	Imm4				Rm				

Mapping to Marvell® Wireless MMX™ 2 Technology and SSE Appendix B

B.1 Marvell® Wireless MMX™ 2 Coprocessor Mapping

Table -B-1 shows the mapping of Marvell® Wireless MMX™ 2 coprocessor instructions to Marvell® Wireless MMX™ 2 technology and SSE integer instructions:

Table B-1. Mapping Marvell® Wireless MMX™ 2 Coprocessor Instructions to Marvell® Wireless MMX™ 2 Technology and SSE Integer Instructions (Sheet 1 of 2)

Marvell® Wireless MMX™ 2 Coprocessor	Marvell® Wireless MMX™ 2 Technology	SSE	Comments
WADD{b/h/w}	PADD{b/w/d}	—	
WSUB{b/h/w}	PSUB{b/w/d}	—	
WCMPEQ{b/h/w}	PCMPEQ{b/w/d}	—	
WCMPGT{b/h/w}	PCMPGT{b/w/d}	—	
WMUL{L}	PMULLW	—	
WMUL{H}	PMULHW	—	
WMADD	PMADDWD	—	
WSRA{h/w}	PSRA{w/d}	—	
WSLL{h/w/d}	PSLL{w/d/q}	—	
WSRL{h/w/d}	PSRL{w/d/q}	—	
WUNPCKIL{b/h/w}	PUNPCKL{bw/wd/dq}		Marvell® Wireless MMX™ 2 coprocessor is a superset
WUNPCKIH{b/h/w}	PUNPCKH{bw/wd/dq}		Marvell® Wireless MMX™ 2 coprocessor is a superset
WPACK{h/w}{SS}	PACKSS{wb/dw}	—	
WPACK{h/w}{US}	PACKUS{wb/dw}	—	
WAND	PAND	—	
WANDN	PANDN	—	
WOR	POR	—	
WXOR	PXOR	—	
WMOV/WLDR	MOV{d/q}		
WMAX{B}{U}	—	PMAXUB	Marvell® Wireless MMX™ 2 coprocessor is a superset

Table B-1. Mapping Marvell® Wireless MMX™ 2 Coprocessor Instructions to Marvell® Wireless MMX™ 2 Technology and SSE Integer Instructions (Sheet 2 of 2)

Marvell® Wireless MMX™ 2 Coprocessor	Marvell® Wireless MMX™ 2 Technology	SSE	Comments
WMAX{H}{S}	—	PMAXSW	Marvell® Wireless MMX™ 2 coprocessor is a superset
WMIN{B}{U}	—	PMINUB	Marvell® Wireless MMX™ 2 coprocessor is a superset
WMAX{H}{S}	—	PMIXSW	Marvell® Wireless MMX™ 2 coprocessor is a superset
TMOVMSK{B}	—	PMOVMSKB	Transfer instruction
WAVG2{B,W}	—	PAVG{BW}	Marvell® Wireless MMX™ 2 coprocessor is a superset
TINSR{W}	—	PINSRW	Marvell® Wireless MMX™ 2 coprocessor is a superset
TEXTRM{W}	—	PEXTRW	Marvell® Wireless MMX™ 2 coprocessor is a superset
WSAD{B,W}	—	PSAD{BW}	
WSHUFH	—	PSHUFW	

Marvell® Wireless MMX™ 2 Coprocessor Instruction Additions

Appendix C

C.1 Introduction

This appendix is an overview of the changes for the instruction set architecture from the Concan SIMD engine to the Marvell® Wireless MMX™ 2 coprocessor. The Marvell® Wireless MMX™ 2 coprocessor is targeting additional performance enhancement for audio, voice, and video applications. Instruction selection for implementation is based on design complexity, performance yield, and application targets.

The changes to the instruction set architecture fall into two distinct categories, enhancements to existing instructions and new instructions.

C.2 Enhancements to Existing Instructions

Table C-1 summarizes the enhancements to existing instructions.

Table C-1. Enhancements to Existing Instructions (Sheet 1 of 2)

Instruction	Section	New Instruction Feature
WADD	Chapter 3, Section 3.2.19 on page 56	The use of the carry flags from the wCASF register may be optionally supplied as the carry-in to the addition operation using the C-qualifier. The add with carry-in option is valid for 16 and 32-bit unsigned operands only.
WLDR	Chapter 3, Section 3.2.29 on page 68	The register offset addressing mode is provided for 64-bit loads to the data registers. A left shift may be optionally applied to the offset when using the register offset addressing mode.
WSTR	Chapter 3, Section 3.2.51 on page 102	The register offset addressing mode is provided for 64-bit store operations from the data registers. A left shift may be optionally applied to the offset when using the register offset addressing mode
WMUL	Chapter 3, Section 3.2.38 on page 80	Biased rounding, “round to nearest” is supported with the R-qualifier for signed and unsigned integer multiplication. Rounding is valid only with the M option.
WMADD	Chapter 3, Section 3.2.31 on page 72	The operands may be cross multiplied with the use of the X-qualifier and may be optionally subtracted with the use of the N-qualifier. The X-qualifier may only be used with the default multiply-add operation.
WSLL	Chapter 3, Section 3.2.48 on page 96	The logical shift left by immediate is supported with a 5-bit immediate.

Table C-1. Enhancements to Existing Instructions (Sheet 2 of 2)

Instruction	Section	New Instruction Feature
WSRA	Chapter 3, Section 3.2.49 on page 98	The arithmetic shift right by immediate is supported with a 5-bit immediate.
WSRL	Chapter 3, Section 3.2.50 on page 100	The logical shift right by immediate is supported with a 5-bit immediate.
WROR	Chapter 3, Section 3.2.45 on page 91	The logical rotate right by immediate is supported with a 5-bit operand.

C.3 New Instructions

Table C-2 summarizes the new instruction support.

Table C-2. New Media Processing Instructions (Sheet 1 of 2)

Instruction	Section	Description
TORVSC	Chapter 3, Section 3.2.15 on page 52	Performs “OR” across the fields of the SIMD saturation flags (wCSSF) and sends the result to the ARM* CPSR Overflow, (V), flag; operation can be performed after a byte, half-word or word operation that sets the flags.
WABS	Chapter 3, Section 3.2.16 on page 53	Performs a vector absolute value of wRn on 8-bit, 16-bit, or 32-bit signed data and places the result into the destination register, wRd.
WABSDIFF	Chapter 3, Section 3.2.17 on page 54	Performs a vector absolute value of the differences of wRm and wRn for vectors 8-bit, 16-bit, or 32-bit unsigned data, and places the result in wRd.
WADDSUBHX	Chapter 3, Section 3.2.20 on page 59	Performs complex vector addition/subtraction of wRn and wRm for vectors of 16-bit data, and places the result in wRd. The four operands from each of the source registers are alternately added and subtracted using a cross selection in each of the parallel operations. The result of the operation is saturated to the signed limits, for example, 0x8000 to 0x7fff or -32768 to 32767.
WAVG4	Chapter 3, Section 3.2.26 on page 64	Performs seven 4-pixel averages of unsigned 8-bit operands obtained from the bytes of the wRn and wRm source registers. Biased rounding is supported through the use of the R-qualifier. The seven 4-pixel averages are packed into the lower seven bytes of the destination register with the most significant byte written with 0x00.
WMERGE	Chapter 3, Section 3.2.33 on page 75	Extracts a 64-bit value that contains elements from the two 64-bit source registers (wRn,wRm), and places a merged 64-bit result in the destination register, wRd. The number of adjacent elements from each register is represented with a 3-bit immediate.
WMIAxy	Chapter 3, Section 3.2.34 on page 76	Performs a signed parallel 16-bit multiply, or multiply-negate, followed by 64-bit accumulation. The upper or lower 16-bits of each source register half is selected by specifying either the B (bottom) or T (top) in each of the xy positions of the mnemonic. The xy selection is the same for both the upper and lower halves of the 64-bit operand source registers, wRn and wRm. The resulting product for the upper and lower halves are then added to the 64-bit accumulator which occupies the destination register wRd. The N-qualifier optionally provides for a multiply-negate-accumulate operation instead of the default multiply-accumulate.

Table C-2. New Media Processing Instructions (Sheet 2 of 2)

Instruction	Section	Description
WMIAWxy	Chapter 3, Section 3.2.35 on page 77	Performs multiply-accumulate using signed 32-bit operands from the two source wRm and wRn, and accumulates the result with the destination register, wRd. The upper or lower 32-bits of each source register half is selected by specifying either the B (bottom) or T (top) in each of the xy positions of the mnemonic.The N-qualifier optionally provides for a multiply-negate-accumulate operation instead of the default multiply-accumulate.
WMULW	Chapter 3, Section 3.2.39 on page 82	Performs a vector multiplication of wRn and wRm on vectors of 32-bit data only; UM-qualifier indicates that the higher order 32 bits of the result of the unsigned multiplication are to be stored in wRd, the SM-qualifier indicates that the higher order 32 bits of the signed multiplication are to be stored in wRd, and the L-qualifier indicates that the lower 32 bits of the result are to be stored in wRd. Biased rounding is available with the UM and SM options through the use of the R-qualifier.
WQMIAxy	Chapter 3, Section 3.2.42 on page 86	Performs a signed fractional parallel 16-bit multiply, or multiply-negate, followed by parallel 32-bit accumulation. The upper or lower 16-bits of each Marvell® Wireless MMX™ 2 coprocessor source register half is selected by specifying either the B (bottom) or T (top) in each of the xy positions of the mnemonic. The xy selection is the same for both the upper and lower halves of the 64-bit operand source registers, wRn and wRm. The resulting product for the upper and lower halves are then added to the two 32-bit accumulators which occupy the upper and lower halves of the 64-bit destination register wRd. A left shift correction is applied following the multiplication and saturation is provided with the addition for 32-bit signed operands. The N- qualifier optionally provides for a multiply-negate-accumulate operation instead of the default multiply-accumulate.
WQMULM	Chapter 3, Section 3.2.43 on page 88	Performs a vector multiplication of wRn and wRm on vectors of signed fractional 16-bit Qm.n values. The upper half of the results are stored in wRd with a left shift correction to renormalize the fractional data. Biased rounding, “round to nearest” is supported with the R-qualifier with truncation as the default behavior.
WQMULWM	Chapter 3, Section 3.2.44 on page 90	Performs a vector multiplication of wRn and wRm on vectors of signed fractional 32-bit Qm.n values. The upper half of the results are stored in wRd with a left shift correction to renormalize the fractional data. Biased rounding, “round to nearest” is supported with the R-qualifier with truncation as the default behavior.
WSUBADDHX	Chapter 3, Section 3.2.53 on page 106	Performs complex vector subtraction/addition of wRn and wRm for vectors of 16-bit data, and places the result in wRd. The four operands from each of the source registers are alternately added and subtracted using a cross selection in each of the parallel operations.The result of the operation is saturated to the signed limits, for example, 0x8000 to 0x7fff or -32768 to 32767.

THIS PAGE INTENTIONALLY LEFT BLANK

Performance Optimization Appendix D

D.1 Introduction

This chapter describes the factors that effect performance for the Marvell® Wireless MMX™ 2 Coprocessor 1.0 implementation. Use this section as a guide to achieving maximum performance with the Marvell® Wireless MMX™ 2 coprocessor 1.0 implementation. The factors described in this section are implementation-dependant, so should not be relied upon for future Marvell® Wireless MMX™ 2 Coprocessor implementations.

The two main factors that effect performance are the following:

- How many clock cycles an instruction consumes in the issue stage, often referred to as issue cycles
- How many clock cycles before the instruction result is available to a following instruction, often referred to as result latency

Section D.2 describes the default issue cycles and result latency for all Marvell® Wireless MMX™ 2 coprocessor instructions. The performance of each instruction is also influenced by performance hazards. Section D.3 describes the two main performance hazards: resource conflicts and data dependencies.

Note: This preliminary chapter will evolve as the more detailed issues are resolved in the implementation.

D.2 Issue Cycle and Result Latency

The issue cycle and result latency of all the Marvell® Wireless MMX™ 2 Coprocessor instructions is shown in Table D-1. In this table, the issue cycle is the number of cycles that an instruction takes to leave the register file. The result latency is the number of cycles required to calculate the result and make it available to the bypassing logic. A result latency of *1* indicates that the value is available immediately to the following instruction. This table shows the best case result latency that can be degraded by data or result hazards.

Table D-1. Issue Cycle and Result Latency of the Marvell® Wireless MMX™ 2 Coprocessor Instructions (Sheet 1 of 3)

Instructions	Issue Cycle	Result Latency
WABS	1	2
WABSIDFF	1	2
WADD	1	1
WADDSUBX	1	2
WSUBADDX	1	2
WSUB	1	1
WCMPEQ	1	2
WCMPGT	1	2

Table D-1. Issue Cycle and Result Latency of the Marvell® Wireless MMX™ 2 Coprocessor Instructions (Sheet 2 of 3)

Instructions	Issue Cycle	Result Latency
WAND	1	1
WANDN	1	1
WOR	1	1
WXOR	1	1
WAVG2	1	1
WAVG4	1	2
WMAX	1	2
WMIN	1	2
WSAD	1	1
WACC	1	1
WQMULM	1	2
WQMULWM	1	4
WMUL	1	2
WMULW	1	4
WMADD	1	2
WMAC	1	2
WQMIAxy	1	1
WMIAxy	1	1
WMIAWxy	1	2
TMIA	1	2
TMIAPH	1	1
TMIAxy	1	1
WSLL	1	1
WSRA	1	1
WSRL	1	1
WROR	1	1
WPACK	1	2
WUNPCKEH	1	1
WUNPCKEL	1	1
WUNPCKIH	1	1
WUNPCKIL	1	1
WALIGNI	1	1
WALIGNR	1	1
WMERGE	1	1
WSHUF	1	1

Table D-1. Issue Cycle and Result Latency of the Marvell® Wireless MMX™ 2 Coprocessor Instructions (Sheet 3 of 3)

Instructions	Issue Cycle	Result Latency
TANDC	1	1
TORC	1	1
TEXTRC	1	1
TEXTRM	1	2
TMCR	1	1
TMCRR	1	1
TMRC	1	2
TMRRC (rdlo)	1	2
TMRRC (rdhi)	1	3
TMOVMSK	1	2
TINSTR	1	1
TBCST	1	1
WLDR to main regfile	1	3
WLDRW to control regfile	1	4
WSTR	1	na

D.3 Performance Hazards

The basic performance of the system can be effected by stalls caused by data or resource hazards. This section describes the factors effecting each type of hazard and the implications for performance.

D.3.1 Data Hazards

A data hazard occurs when an instruction requires data that cannot be provided by the register file or the data-forwarding mechanism or if two instructions update the same destination register in an out of order fashion. The first hazard is termed as read-after-write (RAW) and the second hazard is termed as write-after-write (WAW). The processing of the new instruction is stalled until the data becomes available for RAW hazards, and until it can be guaranteed that the new instruction updates the register file after previous instruction has updated the same destination register, for WAW hazards. The Marvell® Wireless MMX™ 2 coprocessor device contains a bypassing mechanism for ensuring that data and different stages of the pipeline can forwarded to the correct instructions. There are, however, certain combinations of instructions where it is not possible to forward directly between instructions in the Marvell® Wireless MMX™ 2 coprocessor 1.0 implementation.

The result latency shown in Table D-1 and best-case result latency are generally achievable. However there are certain instruction combinations where these result latencies do not hold because not all combinations of bypassing logic exist in the hardware, and some instructions require more time to calculate the result when certain qualifiers are specified. The following list describes the data hazards for the Marvell® Wireless MMX™ 2 coprocessor 1.0 implementation:

- When saturation is specified for WADD or WSUB, the result latency is increased to two cycles.

- When the result of a previous execution pipeline instruction is needed as an operand to the WADDSUBX or WSUBADDX instructions, the result latency is increased to three cycles.
- The destination register (accumulator) for certain multiplier instructions (WMAC, WSAD, TMIA, TMIAPH, TMIAxy) can be forwarded for accumulation to the same destination register only. If the destination register results are needed by another instruction as source operands, there is an additional result latency as the result is available from the regular forwarding paths, external to the multiplier.
  - For WMAC, WACC, WMUL, WMADD, WMIAxy, WMIAWxy, WQMULM, WQMIAxy, WMULW and WQMULWM there is an additional result latency of 3 cycles.
  - For WSAD, TMIA, TMIAPH, and TMIAxy there is an additional result latency of 4 cycles.
- If an instruction is updating a destination register from the multiply pipeline, the next instruction in the execute, memory or core interface pipelines updating the same destination register is stalled. The instruction is stalled until it can be guaranteed that the following instruction updates the register file after the earlier instruction in the multiply pipe has updated the register file

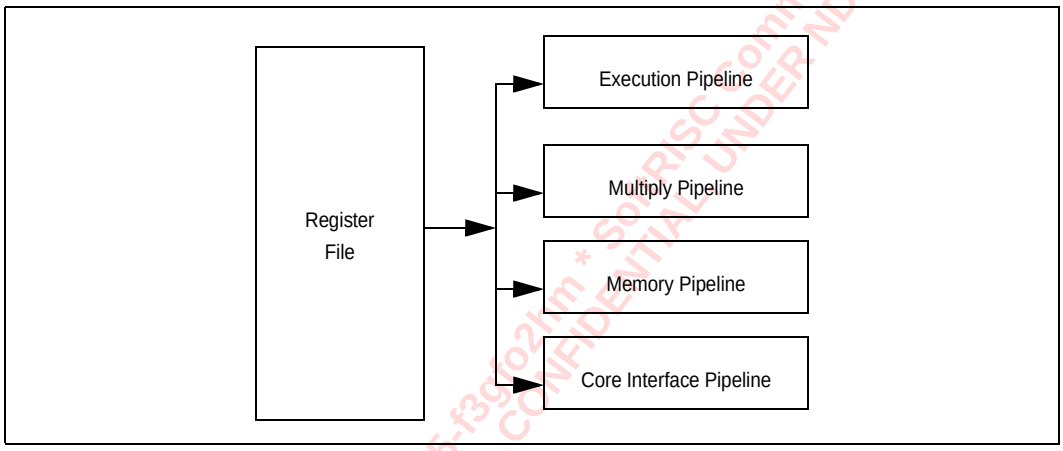
If the Intel XScale® microarchitecture MAC unit is in use, the resulting latency of a TMRC, TMRRC, and TEXTRM increases accordingly

D.3.2 Resource Hazard

A resource hazard is caused when an instruction requires a resource that is already in use. When this condition is detected, the processing of the new instruction is stalled at the register file stage.

Figure D-1 shows a high-level representation of the operation of the Marvell® Wireless MMX™ 2 coprocessor. After the register file, there are four concurrent pipelines to which an instruction can be dispatched. An instruction can be issued to a pipeline if the resource is available and there are no unresolved data dependencies. For example, a load instruction that uses the memory pipeline can be issued while a multiply instruction is completing in the multiply pipeline (assuming there are no data hazards.)

Figure D-1. High-Level Pipeline Organization



The performance effect of resource contention can be quantified by examining the delay taken for a particular instruction to release the resource after starting execution. The definition of “release the resource” in this context is that the resource can accept another instruction (note: the resource may still be processing the previous instruction further down its internal pipeline). A delay of one clock cycle indicates that the resource is available

immediately to the next instruction. A delay greater than one clock cycle stalls the next instruction if the same resource is required. The following sections examine the resource-usage delays for the four pipelines, and how these map onto the instruction set.

D.3.2.1 Execution Pipeline

An instruction can be accepted into the execution pipeline when the first stage of the pipeline is empty. Table D-2 shows the instructions that execute in the main execution pipeline. All these instructions have a resource usage delay of one clock cycle. Therefore, the execution pipeline is always available to the next instruction.

Table D-2. Resource Availability Delay for the Execution Pipeline

Instructions	Delay (Clocks)
WABS	1
WABSDIFF	1
WADD	1
WSUB	1
WADDSUBX	1
WSUBADDX	1
WCMPEQ	1
WCMPGT	1
WAND	1
WANDN	1
WOR	1
WXOR	1
WAVG2	1
WAVG4	1
WMAX	1
WMIN	1
WSAD	1†
WSLL	1
WSRA	1
WSRL	1
WROR	1
WPACK	1
WUNPCKEH	1
WUNPCKEL	1
WUNPCKIH	1
WUNPCKIL	1
WALIGNI	1
WALIGNR	1

Table D-2. Resource Availability Delay for the Execution Pipeline (Continued)

Instructions	Delay (Clocks)
WMERGE	1
WSHUF	1
† The WSAD executes in both the main execution pipeline and the multiplier pipeline. It executes for one cycle in the execution pipeline and the rest in the multiplier pipeline. See Chapter 3, Section 3.2.46 on page 93 for more details	

D.3.2.2 Multiply Pipeline

Instructions issued to the multiply pipeline may take up to two cycles before another instruction can be issued to the pipeline. The instructions in the multiply pipe can be categorized into 5 classes shown in Table D-3. The resource-availability delay for the instructions that are mapped onto the multiplier pipeline depend upon the class of the multiply instruction that subsequently wants to use the multiply resource. These delays are shown below in Table D-4. For example if a TMIA instruction is followed by a TMIAph (class3) instruction, then the TMIAph sees a resource availability of 2 cycles.

Table D-3. Multiply Pipe Instruction Classes

Instructions	Class
WACC, WQMIAxy, WMIAxy	1
WMAC, WMUL, WQMULM, WMIAWxy, WMADD	2
WSAD, TMIAph, TMIAxy	3
TMIA	4
WMULW, WQMULWM	5

Table D-4. Resource Availability Delay for the Multiplier Pipeline (Sheet 1 of 2)

Instructions	Delay(Clocks) for a subsequent class 1 multiply pipe instruction	Delay(Clocks) for a subsequent class 2 multiply pipe instruction	Delay(Clocks) for a subsequent class 3 multiply pipe instruction	Delay(Clocks) for a subsequent class 4 multiply pipe instruction	Delay(Clocks) for a subsequent class 5multiply pipe instruction
WSAD	2	2	1	1	2
WACC	1	1	1	1	1
WMUL	2	2	1	1	2
WMULW	4	4	3	3	4
WQMULM	2	2	1	1	2
WQMULWM	4	4	3	3	4
WMADD	2	2	1	1	2
WMAC	2	2	1	1	2

Table D-4. Resource Availability Delay for the Multiplier Pipeline (Sheet 2 of 2)

Instructions	Delay(Clocks) for a subsequent class 1 multiply pipe instruction	Delay(Clocks) for a subsequent class 2 multiply pipe instruction	Delay(Clocks) for a subsequent class 3 multiply pipe instruction	Delay(Clocks) for a subsequent class 4 multiply pipe instruction	Delay(Clocks) for a subsequent class 5multiply pipe instruction
WMIAxy	1	1	1	1	1
WMIAWxy	2	2	1	1	2
WQMIAxy	1	1	1	1	1
TMIA	3	3	2	2	3
TMIAPH	2	2	1	1	2
TMIAxy	2	2	1	1	2

WSAD (See Chapter 3, “WSAD”), TMIA (See Chapter 3, “TMIA”), TMIAxy (See Chapter 3, “TMIAxy”), and TMIAph (See Chapter 3, “TMIAPH”) execute in both the main execution pipeline and the multiplier pipeline.

D.3.2.3 Memory Control Pipeline

The memory control pipeline is responsible for coordinating the load/store activity with the main core. The external interface to memory is 64-bits for both loads from memory and stores to memory. The Marvell® Wireless MMX™ 2 coprocessor supports up to 8 outstanding loads. Any additional loads will see latencies as highlighted in Table D-5.

Table D-5. Resource Availability Delay for the Memory Pipeline

Instructions	Delay(Clocks)	Condition
WLDRD	1 +M	Eight loads already outstanding (M is delay for main memory if cache miss)

There is also an additional stall introduced by the core when 2 double word (64 bits) are issued back to back such as:

WLDRD or WSTRD

WLDR[B,H,W,D] or WSTR[B,H,W,D] <- 1 cycle stall.

Critical inner loop sequences can use non memory related instructions following a WLDRD or WSTRD.

D.3.2.4 Coprocessor Interface Pipeline

The coprocessor interface pipeline also contains buffering to allow multiple outstanding MRC/MRRC operations. The coprocessor interface pipeline can continue to accept MRC and MRRC instructions every cycle until its buffers are full. Currently there is sufficient storage in the buffer for either four MRC data values (32-bit) or two MRRC data values (64-bit).Table D-6 shows a summary of the resource availability delay for the coprocessor interface.

Table D-6. Resource Availability Delay for the Coprocessor Interface Pipeline

Instructions	Delay(Clocks)	Condition
TMRC	1	Buffer Empty
TMRC	2	Buffer Full
TMRRC	1	Buffer empty
TMRRC	2	Buffer Full

There is also an interaction between TMRC/TMRRC and any instructions in the core that utilize the MAC unit of the core. For optimum performance, the MAC unit in the core should not be used adjacent to TMRC instructions as they both share the route back to the core register file.

D.3.2.5 Multiple Pipelines

The WSAD instruction is a special case that executes in both the main execution pipeline and the multiplier pipeline. The instruction executes one cycle in the execution pipeline and the rest in the multiplier pipeline. The WSAD always issues without stalls to the execution pipeline (see Section D.3.2.1). The multiplier pipeline is always available when the WSAD needs to use the resource. However, if the WSAD instruction is followed by an instruction that requires the multiplier, there is an effective resource availability delay of two.

THIS PAGE INTENTIONALLY LEFT BLANK

MOVING FORWARD FASTER®

Marvell Semiconductor, Inc.

700 First Avenue
Sunnyvale, CA 94089, USA

Tel: 1.408.222.2500
Fax: 1.408.752.9028

www.marvell.com

Worldwide Corporate Offices

Marvell Semiconductor, Inc.
700 First Avenue
Sunnyvale, CA 94089, USA
Tel: 1.408.222.2500
Fax: 1.408.752.9028

Marvell Semiconductor, Inc.
5400 Bayfront Plaza
Santa Clara, CA 95054, USA
Tel: 1.408.222.2500

Marvell Asia Pte, Ltd.
151 Lorong Chuan, #02-05
New Tech Park, Singapore 556741
Tel: 65.6756.1600
Fax: 65.6756.7600

Marvell Japan K.K.
Shinjuku Center Bldg. 44F
1-25-1, Nishi-Shinjuku, Shinjuku-ku
Tokyo 163-0644, Japan
Tel: 81.(0).3.5324.0355
Fax: 81.(0).3.5324.0354

Marvell Semiconductor Israel, Ltd.
6 Hamada Street
Mordot HaCarmel Industrial Park
Yokneam 20692, Israel
Tel: 972.(0).4.909.1500
Fax: 972.(0).4.909.1501

Marvell Semiconductor Korea, Ltd.
Rm. 603, Trade Center
159-2 Samsung-Dong, Kangnam-Ku
Seoul 135-731, Korea
Tel: 82.(0).2.551-6070/6079
Fax: 82.(0).2.551.6080

Radlan Computer Communications, Ltd.
Atidim Technological Park, Bldg. #4
Tel Aviv 61131, Israel
Tel: 972.(0).3.645.8555
Fax: 972.(0).3.645.8544

Worldwide Sales Offices

Western US
Marvell
700 First Avenue
Sunnyvale, CA 94089, USA
Tel: 1.408.222.2500
Fax: 1.408.752.9028
Sales Fax: 1.408.752.9029

Marvell
5400 Bayfront Plaza
Santa Clara, CA 95054, USA
Tel: 1.408.222.2500

Central US
Marvell
9600 North MoPac Drive, Suite #215
Austin, TX 78759, USA
Tel: 1.512.343.0593
Fax: 1.512.340.9970

Eastern US/Canada
Marvell
Parlee Office Park
1 Meeting House Road, Suite 1
Chelmsford, MA 01824 , USA
Tel: 1.978.250.0588
Fax: 1.978.250.0589

Europe
Marvell
5 Marchmont Gate
Boundary Way
Hemel Hempstead
Hertfordshire, HP2 7BF
United Kingdom
Tel: 44.(0).1442.211668
Fax: 44.(0).1442.211543

Israel
Marvell
6 Hamada Street
Mordot HaCarmel Industrial Park
Yokneam 20692, Israel
Tel: 972.(0).4.909.1500
Fax: 972.(0).4.909.1501

China
Marvell
5J1, 1800 Zhongshan West Road
Shanghai, PRC 200233
Tel: 86.21.6440.1350
Fax: 86.21.6440.0799

Marvell
Rm. 1102/1103, Jintian Fudi Mansion
#9 An Ning Zhuang West Rd.
Qing He, Haidian District
Beijing, PRC 100085
Tel: 86.10.8274.3831
Fax: 86.10.8274.3830

Japan
Marvell
Shinjuku Center Bldg. 44F
1-25-1, Nishi-Shinjuku, Shinjuku-ku
Tokyo 163-0644, Japan
Tel: 81.(0).3.5324.0355
Fax: 81.(0).3.5324.0354

Taiwan
Marvell
2Fl., No.1, Alley 20, Lane 407, Sec. 2
Ti-Ding Blvd., Nei Hu District
Taipei, Taiwan, 114, R.O.C
Tel: 886.(0).2.8177.7071
Fax: 886.(0).2.8752.5707

Korea
Marvell
Rm. 603, Trade Center
159-2 Samsung-Dong, Kangnam-Ku
Seoul 135-731, Korea
Tel: 82.(0).2.551-6070/6079
Fax: 82.(0).2.551.6080

For more information, visit our website at:
www.marvell.com