MOVE® Coprocessor

Technical Reference Manual

ARM®

ARM DDI 0235C

MOVE Coprocessor
Technical Reference Manual

Release Information

The following changes have been made to this manual.

Change history

Date	Issue	Change
December 2001	A	First release.
April 2002	B	Input and output added to Figure 2-2. Signal name changed from **DABORT** to **CPABORT** on page A-2.
23 August 2004	C	Updated confidentiality.

Proprietary Notice

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Product Status

The information in this document is final, that is for a developed product.

Web Address

`http://www.arm.com`

Contents
MOVE Coprocessor Technical Reference Manual

List of Tables
MOVE Coprocessor Technical Reference Manual

List of Figures
MOVE Coprocessor
Technical Reference Manual

Preface

This preface introduces the ARM MOVE® (ARM9x6) coprocessor. It contains the following sections:

About this manual

This is the technical reference manual for the MOVE (ARM9x6) coprocessor that enhances performance of MPEG4 encoder applications. The manual covers the functional specification for both hardware and software. It does not include the implementation details of either the hardware or the software.

Please contact `info@arm.com` for more information on the MOVE product family and support package.

Intended audience

This manual has been written for hardware and software engineers implementing System-on-Chip designs. It provides information to enable designers to integrate the peripheral into a target system as quickly as possible.

Using this manual

This manual is organized as follows:

Chapter 1 ***Introduction***

Read this chapter for an introduction to the MOVE coprocessor and its features.

Chapter 2 ***Functional Description***

Read this chapter for a description of the major functional blocks of the MOVE coprocessor.

Chapter 3 ***Programmer's Model***

Read this chapter for a description of the MOVE coprocessor registers and programming details.

Appendix A ***System Connectivity and Signals***

Read this appendix for details of the MOVE coprocessor signals.

Conventions

Conventions that this manual can use are described in:

- *Typographical* on page xi
- *Timing diagrams* on page xi
- *Signals* on page xii
- *Numbering* on page xiii.

Typographical

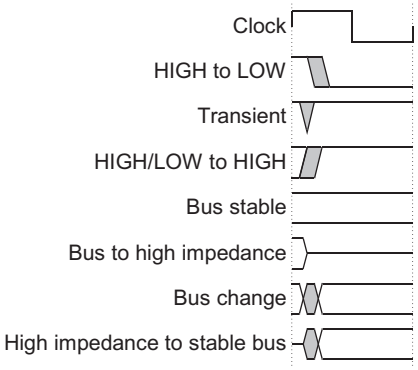
The typographical conventions are:

italic	Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.
bold	Highlights interface elements, such as menu names. Denotes ARM processor signal names. Also used for terms in descriptive lists, where appropriate.
`monospace`	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
`mono`space	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`monospace italic`	Denotes arguments to monospace text where the argument is to be replaced by a specific value.
`monospace bold`	Denotes language keywords when used outside example code.
< and >	Angle brackets enclose replaceable terms for assembler syntax where they appear in code or code fragments. They appear in normal font in running text. For example: • `MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>` • The Opcode_2 value selects which register is accessed.

Timing diagrams

The figure named *Key to timing diagram conventions* on page xii explains the components used in timing diagrams. Variations, when they occur, have clear labels. You must not assume any timing information that is not explicit in the diagrams.

Shaded bus and signal areas are undefined, so the bus or signal can assume any value within the shaded area at that time. The actual level is unimportant and does not affect normal operation.



Key to timing diagram conventions

Signals

The signal conventions are:

Signal level The level of an asserted signal depends on whether the signal is active-HIGH or active-LOW. Asserted means HIGH for active-HIGH signals and LOW for active-LOW signals.

Prefix A Denotes AXI global and address channel signals.

Prefix B Denotes AXI write response channel signals.

Prefix C Denotes AXI low-power interface signals.

Prefix H Denotes *Advanced High-performance Bus* (AHB) signals.

Prefix n Denotes active-LOW signals except in the case of AXI, AHB, or *Advanced Peripheral Bus* (APB) reset signals.

Prefix P Denotes APB signals.

Prefix R Denotes AXI read data channel signals.

Prefix W Denotes AXI write data channel signals.

Suffix n Denotes AXI, AHB, and APB reset signals.

Numbering

The numbering convention is:

<size in bits>'<base><number>

This is a Verilog method of abbreviating constant numbers. For example:

- 'h7B4 is an unsized hexadecimal value.
- 'o7654 is an unsized octal value.
- 8'd9 is an eight-bit wide decimal value of 9.
- 8'h3F is an eight-bit wide hexadecimal value of `0x3F`. This is equivalent to b00111111.
- 8'b1111 is an eight-bit wide binary value of b00001111.

Further reading

This section lists publications by ARM Limited, and by third parties.

ARM Limited periodically provides updates and corrections to its documentation. See `http://www.arm.com` for current errata sheets, addenda, and the Frequently Asked Questions list.

ARM publications

This manual contains information that is specific to the ARM MOVE coprocessor. Refer to the following documents for other relevant information:

- *ARM9E-S Technical Reference Manual* (ARM DDI 0165).
- *ARM926E-S Engineering Specification* (ARM CP023 ESPC 0001).
- *ARM946E-S Engineering Specification* (ARM CP018 ESPC 0000).
- *ARM966E-S Engineering Specification* (ARM CP017 ESPC 0000).
- *ARM Architecture Reference Manual* (ARM DDI 0100).

Feedback

ARM Limited welcomes feedback on the ARM MOVE coprocessor and its documentation.

Feedback on this ARM MOVE coprocessor

If you have any comments or suggestions about this product, contact your supplier giving:

- the product name
- a concise explanation of your comments.

Feedback on this manual

If you have any comments on this manual, send email to `errata@arm.com` giving:

- the title
- the number
- the relevant page number(s) to which your comments apply
- a concise explanation of your comments.

ARM Limited also welcomes general suggestions for additions and improvements.

Chapter 1
Introduction

This chapter introduces the ARM MOVE coprocessor. It contains the following sections:

1.1 About the MOVE coprocessor

The MOVE coprocessor is a video encoding acceleration coprocessor designed to accelerate *Motion Estimation* (ME) algorithms within block-based video encoding schemes such as MPEG4 and H.263. It provides support for the execution of *Sum of Absolute Differences* (SAD) calculations, which account for most of the processing activity within an ME algorithm. The ME algorithms require many comparisons of 8x8 pixel blocks to be made between a current frame and a reference frame.

This coprocessor is one element in the MOVE product group. It is assigned the letter U. All coprocessor instruction mnemonics are prefixed with the letter U.

Note

This document refers to the coprocessor as the MOVE.

1.2 MOVE structure

The MOVE interprets a set of coprocessor instructions that are part of the ARM instruction set. The ARM coprocessor instructions enable the ARM processor to:

- transfer data from memory to the MOVE, using `LDC` instructions
- transfer data from MOVE to ARM registers, using `MCR` instructions, and from ARM to MOVE registers, using `MRC` instructions.

The ARM processor acts as an address generator and data pump for the MOVE. The MOVE consists of:

- a register bank
- a data path
- control logic.

Figure 1-1 shows the major blocks of the MOVE.

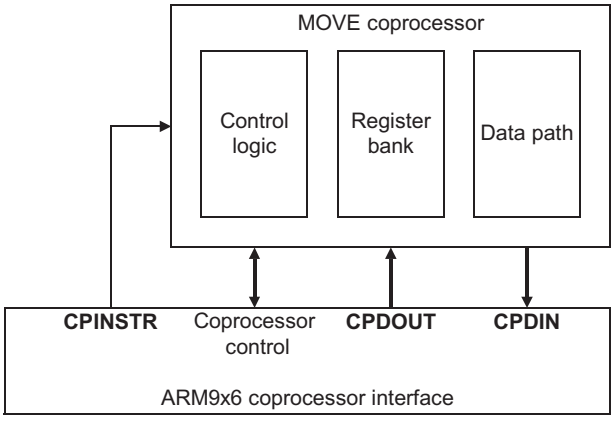


Figure 1-1 MOVE block diagram

Note

For details of the blocks, see *Functional overview* on page 2-4.

1.3 MOVE interface

The only connection to the MOVE is the coprocessor interface from the ARM processor. All other system connectivity, such as AMBA or interrupts, are handled using the ARM processor.

Note

Connection of multiple external coprocessors is not supported.

For more information on the coprocessor interface, see the applicable ARM9x6 processor Technical Reference Manual.

Chapter 2
Functional Description

This chapter gives a functional description of the ARM MOVE coprocessor, and illustrates its functional and block diagrams. It contains the following sections:

2.1 MOVE overview

The MOVE is closely coupled to the ARM9x6 family of processors. Each instruction that is destined for the MOVE is processed immediately as if the ARM core is executing the instruction.

The MOVE is not a *fire and forget* coprocessor. MOVE instructions are executed immediately and the result is available for the following instruction, subject to certain conditions. See *Data hazards* on page 3-17. This means that the ARM processor does not have to poll the status of the coprocessor, and the coprocessor does not have to interrupt the ARM processor.

2.2 Connectivity

Figure 2-1 shows the connections of the MOVE to the ARM coprocessor interface.

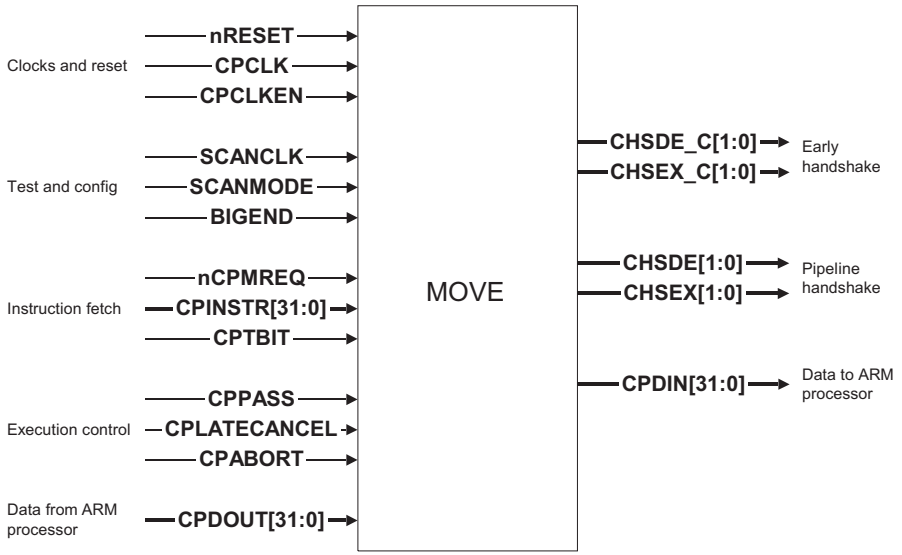


Figure 2-1 MOVE connections

2.3 Functional overview

Figure 2-2 shows a MOVE functional diagram.

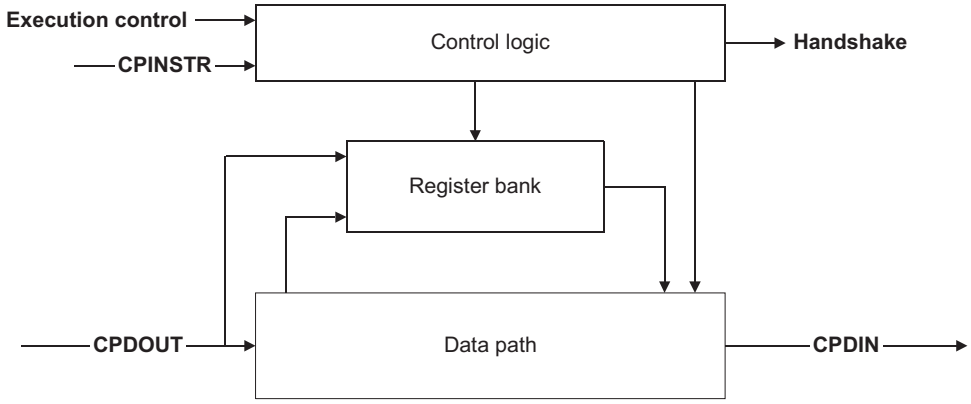


Figure 2-2 MOVE functional diagram

2.3.1 Control logic

The control logic reads each instruction as it arrives from memory. It contains a pipeline follower so that the MOVE and the ARM processor are in step.

The coprocessor pipeline consists of the following stages:

- Fetch
- Decode
- Execute
- Memory
- Write
- SAD accumulate.

2.3.2 Register bank

The register bank holds:

- the block buffer and the `MCR`/`MRC`-capable registers
- the logic to increment CR_IDX.

2.3.3 Data path

The data path contains the logic for the `USALD` instruction, including:

- byte manipulation

- the SAD
- the accumulate.

The data path also handles register read selection and result-forwarding for internal operations, and `UMRC` and `UMRBB` instructions.

Chapter 3
Programmer's Model

This chapter describes the programmer's model for the ARM MOVE coprocessor. It contains the following sections:

3.1 Registers

The MOVE coprocessor contains two types of data storage:

Registers These are used to transfer data directly between ARM registers and the coprocessor. These can be accessed by `MCR` and `MRC` operations.

Block buffer This an 8-byte (64-bit) by 8-line store. The block buffer is accessed by a set of special MOVE instructions. ARM sees these as `LDC`, `MCR`, and `MRC` instructions.

Table 3-1 shows the MOVE register summary.

Table 3-1 MOVE register summary

Name	Type	Reset value	Description
CR_ACC	Read/ write	0	See *Accumulated Results Register*
-	-	-	Reserved
-	-	-	Reserved
CR_IDX	Read/ write	0	See *Line Index Register* on page 3-3
-	-	-	Reserved
CR_BYO	Read/ write	0	See *Byte Offset Register* on page 3-3
CR_CFG	Read/ write	0	See *Configuration Register* on page 3-4
CR_ID	Read	41001020	See *Identification Register* on page 3-5

3.1.1 Accumulated Results Register

The CR_ACC Register is a 14-bit read/write register. You can update it directly using `MCR` instructions and indirectly using SAD operations. Figure 3-1 shows the CR_ACC Register bit assignments.

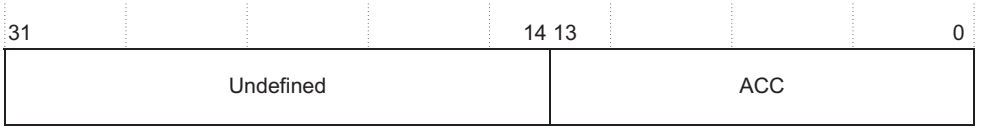


Figure 3-1 CR_ACC Register bit assignments

Table 3-2 lists the CR_ACC Register bit assignments.

Table 3-2 CR_ACC Register bit assignments

Bits	Name	Function
[31:14]	-	Reserved, return to zero on reads, writes ignored.
[13:0]	ACC	Stores the accumulated result.

3.1.2 Line Index Register

The CR_IDX Register is a 3-bit read/write register. You can update it directly using `MCR` instructions. You can update it indirectly using block buffer loads or stores, or SAD operations that increment the line index. Figure 3-2 shows the CR_IDX Register bit assignments.

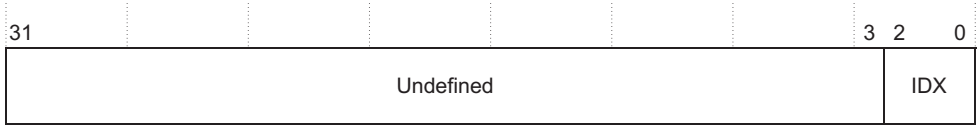


Figure 3-2 CR_IDX Register bit assignments

Table 3-3 lists the CR_IDX Register bit assignments.

Table 3-3 CR_IDX Register bit assignments

Bits	Name	Function
[31:3]	-	Reserved, masked on reads, set to zero for writes.
[2:0]	IDX	Indicates the line index for the block buffer. This sets which line is referenced by the next operation that accesses the block buffer. These operations only ever use a single line from the block buffer. When this register is incremented past the value 7, it wraps to zero.

3.1.3 Byte Offset Register

The CR_BYO Register is a 2-bit read/write register. You can only update it using `MCR` instructions. Figure 3-3 on page 3-4 shows the CR_BYO Register bit assignments.

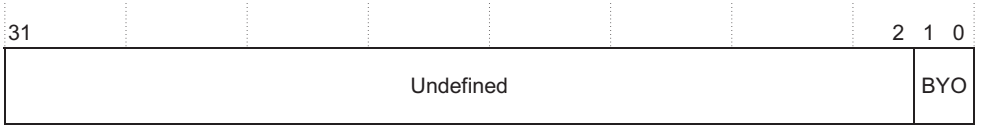


Figure 3-3 CR_BYO Register bit assignments

Table 3-4 lists the CR_BYO Register bit assignments.

Table 3-4 CR_BYO Register bit assignments

Bits	Name	Function
[31:2]	-	Reserved, return to zero on reads, writes ignored.
[1:0]	BYO	The MOVE supports accesses to the reference frame from byte-aligned addresses. The lower two bits of the address bus for these loads are stored in this register. **Note** The coprocessor has no direct visibility of the address value used by the ARM processor for memory accesses, so software must program the byte offset into CR_BYO separately. Also, the ARM processor does not directly support nonword-aligned loads for coprocessors, so the MOVE performs three word loads and then extracts the required eight bytes.

3.1.4 Configuration Register

The CR_CFG Register is a single-bit read/write register. You can only update it using `MCR` instructions. shows the CR_CFG Register bit assignments.

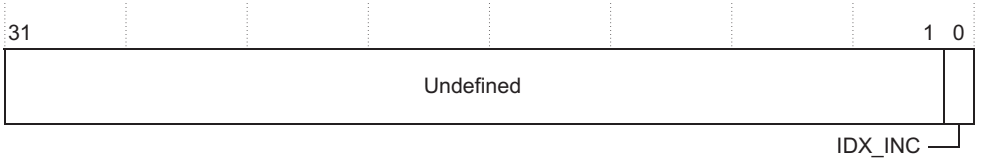


Figure 3-4 CR_CFG Register bit assignments

Table 3-5 lists the CR_CFG Register bit assignments.

Table 3-5 CR_CFG Register bit assignments

Bits	Name	Function
[31:1]	-	Reserved, masked on reads, set to zero for writes.
[0]	IDX_INC	This bit controls whether the CR_IDX register is incremented after a block buffer load/store or SAD operation: 0 = no increment 1 = CR_IDX is incremented after the block buffer or SAD operation completes

3.1.5 Identification Register

The CR_ID Register is a 32-bit read-only register that contains the MOVE architecture and revision code. shows the CR_ID Register bit assignments.

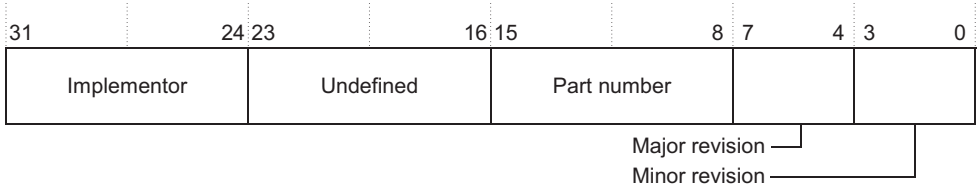


Figure 3-5 CR_ID Register bit assignments

Table 3-6 lists the CR_ID Register bit assignments.

Table 3-6 CR_ID Register bit assignments

Bits	Function
[31:24]	Implementor. The code number for the implementor. `0x41` = ARM.
[23:16]	Undefined.
[15:8]	Part number of the MOVE. `0x10` = MOVE coprocessor.
[7:4]	Contains the major version number for the implementation.
[3:0]	Contains the minor version number for the implementation.

3.2 Instruction set overview

The assembler syntax of the MOVE coprocessor uses the same format as that of the ARM processor, where:

{}	Indicates an optional field.
cond	Is the ARM instruction condition code field.
dest	Specifies the MOVE destination register.
Rn	Is an ARM register.
CRn	Is a MOVE register.
CBBn	Is a word in the MOVE block buffer.
!	Indicates that the calculated address must be written back to the base register.
UCP	Is MOVE coprocessor number.
8_bit_offset	Is an expression evaluating to an 8-bit word offset. This offset is added to the base register to form the load address. The offset must be a multiple of four.

Note

Post indexing with or without write-back is allowed.

3.2.1 MOVE coprocessor instruction classes

The MOVE instructions are in the coprocessor instruction classes shown in Table 3-7.

Table 3-7 MOVE instruction classes

MOVE instruction	ARM coprocessor encoding
UMCR/UMBBR	MCR
UMRC/UMRBB	MRC
UBBLD	LDC (multiple load)
USALD	LDC (multiple load)

The MOVE instructions are encoded as shown in Table 3-8.

Table 3-8 MOVE instruction encoding

	31 28	27	26	25	24	23	22	21	20	19 16	15 14 13 12	11 8	7 6 5 4 3 2 1 0
UMCR	cond	1	1	1	0	0	0	0	0	CRd	Rn	UCP	0 0 0 1 0 0 0 0
UMRC	cond	1	1	1	0	0	0	0	1	CRn	Rd	UCP	0 0 0 1 0 0 0 0
UMBBR	cond	1	1	1	0	0	0	0	0	CBBd	Rn	UCP	0 0 1 1 0 0 0 0
UMRBB	cond	1	1	1	0	0	0	0	1	CBBn	Rd	UCP	0 0 1 1 0 0 0 0
UBBLD	cond	1	1	0	P	U	1	W	1	Rn	1 0 0 0	UCP	8_bit_offset
USALD	cond	1	1	0	P	U	1	W	1	Rn	1 0 0 1	UCP	8_bit_offset

Note

Only the following opcodes are valid for the MOVE. Any variations on these opcodes (which retain UCP in bits 11 down to 8) results in unpredictable behavior.

cond Is the condition code.

UCP Is the MOVE coprocessor number.

Rn Is the ARM register source.

CRn Is the MOVE register source.

CBBn Is the block buffer source word.

Rd Is the ARM register destination.

CRd Is the MOVE register destination.

CBBd Block buffer destination word.

8_bit_offset Is an 8-bit number (0-255) that is used to indicate an address offset.

P Is the pre- or post-indexing bit:

0 = Post-indexing. Add offset after transfer.

1 = Pre-indexing. Add offset before transfer.

U Is the up or down bit.

0 = Down. Subtract from base.

1 = Up. Add offset to base.

W Is the writeback bit.

0 = No writeback.

1 = Write address into base.

3.3 Instruction encodings

This section details the instruction encodings. Table 3-9 shows the MOVE instruction set summary and gives the page reference for the detailed description of each instruction.

Table 3-9 Instruction set summary

MOVE instruction	Description	Detailed description
UMCR	Moves to a MOVE register from an ARM register	*UMCR* on page 3-10
UMRC	Moves to an ARM register from a MOVE register	*UMRC* on page 3-10
UMBBR	Moves to a word in the MOVE block buffer from an ARM register (for debug only)	*UMBBR* on page 3-11
UMRBB	Moves to an ARM register from a word in the MOVE block buffer (for debug only)	*UMRBB* on page 3-12
UBBLD	Loads two words from memory into the block buffer	*UBBLD* on page 3-13
USALD	Performs a byte-aligned load of 8-bytes from memory, performs SAD and accumulates in the CR_ACC Register	*USALD* on page 3-14

3.3.1 UMCR

The UMCR instruction writes to the MOVE registers. UMCR moves data from ARM register Rn to MOVE register CRd. Table 3-10 shows the UMCR encoding.

Table 3-10 UMCR encoding

31	28	27	26	25	24	23	22	21	20	19	16	15	14	13	12	11	8	7	6	5	4	3	2	1	0
cond	1	1	1	0	0	0	0	0	CRd		Rn					UCP		0	0	0	1	0	0	0	0

Syntax

```
UMCR CRd, Rn, {cond}
```

Example

```
UMCR    CR_ACC, R0                  ;load contents of R0 into CR_ACC
```

3.3.2 UMRC

The UMRC instruction reads from the MOVE registers. UMCR moves data from a MOVE register CRn to an ARM register Rd. Table 3-11 shows the UMRC encoding.

Table 3-11 UMRC encoding

31	28	27	26	25	24	23	22	21	20	19	16	15	14	13	12	11	8	7	6	5	4	3	2	1	0
cond	1	1	1	0	0	0	0	1	CRn		Rd					UCP		0	0	0	1	0	0	0	0

Syntax

```
UMRC    Rd, CRn, {cond}
```

Examples

```
UMRC    R6, CR_ID                   ;load ID register into R6.
```

3.3.3 UMBBR

The UMBBR instruction writes to a word in the MOVE block buffer. UMBBR moves data from an ARM register Rn to CBBd in the MOVE block buffer. Table 3-12 shows the UMBBR encoding.

Table 3-12 UMBBR encoding

31	28	27	26	25	24	23	22	21	20	19	16	15	14	13	12	11	8	7	6	5	4	3	2	1	0
cond		1	1	1	0	0	0	0	0	CBBd		Rn				UCP		0	0	1	1	0	0	0	0

Syntax

```
UMBBR    CBBd, Rn, {cond}
```

Example

```
UMBBR    3, R0                    ; load contents of R0 into the second word
                                  ; of the second line of the block buffer.
```

Notes

- This instruction is intended only for use by the MultiICE debugger.
- For the UMBBR instruction, the CBBd opcode field represents a look up into the block buffer:

  Bits[19:17] Refers to the block buffer line number.

  Bit[16] Is 0 for the first word from memory (the first word loaded by a UBBLD operation), and 1 for the second word.

3.3.4 UMRBB

The UMRBB instruction reads a word from the MOVE block buffer. UMRBB moves a word of data from the MOVE block buffer to an ARM register Rd. Table 3-13 shows the UMRBB encoding.

Table 3-13 UMRBB encoding

31	28	27	26	25	24	23	22	21	20	19	16	15	14	13	12	11	8	7	6	5	4	3	2	1	0
cond	1	1	1	0	0	0	0	1	CBBn		Rd					UCP	0	0	1	1	0	0	0	0

Syntax

```
UMRBB    Rd, CBBn, {cond}
```

Example

```
UMRBB    R6, 4                ; load first word from the third line
                              ; of the block buffer into R6.
```

Notes

- This instruction is intended only for use by the MultiICE debugger.
- For the UMRBB instruction, the CBBn opcode field represents a look up into the block buffer:

  Bits[19:17] Refers to the block buffer line number.

  Bit[16] Is 0 for the first word from memory (the first word loaded by a UBBLD operation), and 1 for the second word.

3.3.5 UBBLD

The UBBLD instruction loads data into the block buffer. Table 3-14 shows the UBBLD encoding.

Table 3-14 UBBLD encoding

31	28	27	26	25	24	23	22	21	20	19	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
cond		1	1	0	P	U	1	W	1	Rn		1	0	0	0	UCP											
																		8_bit_offset									

Syntax

```
UBBLD    [Rn], #0, {cond}
UBBLD    [Rn, #+/-10_Bit_Offset]{!}, {cond}
UBBLD    [Rn], #+/-10_Bit_Offset{!}, {cond}
```

Operation

```
Loadword Block_Buffer(CR_IDX,0)
Loadword Block_Buffer(CR_IDX,1)
If (IDX_INC==1)
++CR_IDX
```

Example

```
UBBLD   [R0], #320!           ; Load two words into block buffer from mem(R0)
                              ; and post-increment R0 to next line
```

3.3.6 USALD

The USALD instruction loads data from the reference block and performs a SAD accumulate operation. Table 3-15 shows the USALD encoding.

Table 3-15 USALD encoding

31	28	27	26	25	24	23	22	21	20	19	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
cond		1	1	0	P	U	1	W	1	Rn		1	0	0	1	UCP				8_bit_offset							

Syntax

```
USALD    [Rn], #0, {cond}
USALD    [Rn, #+/-10_Bit_Offset]{!}, {cond}
USALD    [Rn], #+/-10_Bit_Offset{!}, {cond}
```

Operation

```
If (CR_BYO==0)
    Load word to firstword_tmp
else
    Load word to mux1_tmp;
    Load word to mux2_tmp;
    Based on CR_BYO combine mux1_tmp and mux2_tmp to form non-aligned word load in firstword_tmp.

For each byte in firstword_tmp and each corresponding byte in Block_Buffer(CR_IDX, 0) perform SAD.
Accumulate results in CR_ACC.

If (CR_BYO==0)
    Load word to secondword_tmp
else
    Load word to mux3_tmp;
    Based on CR_BYO combine mux2_tmp and mux3_tmp to form non aligned word load in secondword_tmp.

For each byte in secondword_tmp and each corresponding byte in Block_Buffer(CR_IDX, 1) perform SAD.
Accumulate results in CR_ACC.

If (IDX_INC==1)
    ++CR_IDX
```

Example

```
USALD    [R0], #320!    ; Process two words from mem(R0), perform SAD
                        ; and post-increment R0 to next line.
```

Notes

Arithmetic is unsigned. No facilities are included to detect or handle the overflow of CR_ACC. The size of CR_ACC has been chosen to ensure that overflow cannot occur from an 8x8 block compare.

3.4 Instruction cycle timing

Table 3-16 shows the number of cycles each instruction takes to complete.

Table 3-16 Instruction cycle timing

Instruction	Cycle time
UMCR	1
UMRC	1
UMBBR	1
UMRBB	1
UBBLD	2
USALD	N cycles

Note

N = 2 cycles if CR_BYO is zero, otherwise N = 3 cycles.

3.5 Data hazards

Data hazards are circumstances where data dependencies between instructions can result in unpredictable behavior.

There are no hardware interlocks in the MOVE to cope with data hazards. Instead, the software must include a noncoprocessor instruction where necessary. This can be a NOP.

The dependencies are shown in Table 3-17.

Table 3-17 Data dependencies between instructions

Instruction		Next instruction	
UMCR		1.	No UMRC which uses CRd for 2 cycles
		2.	If CRd is CR_Byo then no USALD for 2 cycles
UMRC		1.	Any.
UBBLD	when INC_IDX is clear	1.	No UMRBB for 1 cycle
	when INC_IDX is set	1.	No UMRC CR_IDX for 2 cycles
		2.	No UMRBB for 1 cycle
USALD	when INC_IDX is clear	1.	No UMCR CR_ACC for 1 cycle
		2.	No UMRC CR_ACC for 2 cycles
		3.	No UMRBB for 1 cycle
	when INC_IDX is set	1.	No UMCR CR_ACC for 1 cycle
		2.	No UMRC CR_ACC for 2 cycles
		3.	No UMRC CR_IDX for 2 cycles
		4.	No UMRBB for 1 cycle

Appendix A
System Connectivity and Signals

This appendix describes the connectivity and signals for the ARM MOVE coprocessor. It contains the following sections:

A.1 Connecting the MOVE coprocessor

The MOVE is designed to be connected directly to ARM9x6 processors. The **CHSDE_C** and **CHSEX_C** outputs are unused in this configuration. For the ARM966 and the ARM946 processors the **CPABORT** input must be connected to the **ETMDABORT** pin and the ETM must be enabled. For the ARM926 processor, the **CPABORT** pin must be connected to the **CPABORT** pin of the processor, and the **CPTBIT** pin to the **nCPINSTRVALID** pin of the processor.

A.2 Signal description

Table A-1 to Table A-4 on page A-4 show the signals for the MOVE to interface to the ARM9x6.

Table A-1 MOVE instruction fetch interface signals

Name	Direction	Description
CPINSTR[31:0]	Input	Coprocessor instruction data. This is the coprocessor instruction data bus over which instructions are transferred to the pipeline follower in the coprocessor.
CPTBIT	Input	Coprocessor Thumb bit. If HIGH, the coprocessor interface is in the Thumb state.
nCPMREQ	Input	Not coprocessor memory request. When LOW on a rising **CPCLK** edge and **CPCLKEN** HIGH, the instruction on **CPINSTR** must enter the decode stage of the coprocessor pipeline follower. The second instruction previously in the pipeline followers decode stage enters its execute stage.

Table A-2 MOVE data buses

Name	Direction	Description
CPDIN[31:0]	Output	Data into the ARM. The coprocessor data bus for transferring `MRC` data from the coprocessor to the ARM.
CPDOUT[31:0]	Input	Data out from ARM. The coprocessor data bus for transferring `MCR` and `LDC` data to the coprocessor.

Table A-3 MOVE interface signals

Name	Direction	Description
CHSDE[1:0]	Output	Coprocessor handshake decode. The handshake signals from the decode stage of the coprocessor pipeline follower.
CHSDE_C[1:0]	Output	A combinatorial version of **CHSDE**, output a cycle in advance of **CHSDE** (but only valid towards the end of the cycle). Only for use with a 920 retiming wrapper.
CHSEX[1:0]	Output	Coprocessor handshake execute. The handshake signals from the execute stage of the coprocessor pipeline follower.
CHSEX_C[1:0]	Output	A combinatorial version of **CHSEX**, output a cycle in advance of **CHSEX** (but only valid towards the end of the cycle). Only for use with a 920 retiming wrapper.

Table A-3 MOVE interface signals (continued)

Name	Direction	Description
CPABORT	Input	Coprocessor operation data abort. This signal is used by the coprocessor to recover from an abort during the data loads.
CPLATECANCEL	Input	Coprocessor late cancel. When a coprocessor instruction is being executed, if this signal is HIGH during the first memory cycle, the coprocessor instruction must be canceled without having updated the coprocessor state.
CPPASS	Input	Coprocessor pass. This signal indicates that there is a coprocessor instruction in the execute stage of the pipeline, and it must be executed.

Table A-4 MOVE miscellaneous signals

Name	Direction	Description
BIGEND	Input	Static configuration signal. HIGH to support big-endian systems and LOW for little-endian. Usually supplied from the processor (for example, **BIGENDOUT** from 920).
CPCLK	Input	Clock. This clock times all coprocessor memory accesses (both data and instruction) and internal operations.
CPCLKEN	Input	Coprocessor clock enable. The coprocessor clock **CPCLK** is qualified by this signal.
nRESET	Input	Not reset. This is a level-sensitive input signal, which is used to reset the coprocessor. The coprocessor asynchronously reset when **nRESET** goes LOW.
SCANCLK	Input	Clock input for production scan testing.
SCANIN	Input	Test signal.
SCANMODE	Input	Test mode control input for production scan testing.
SCANOUT	Output	Test signal.

Index

N

P

R

S

T

U

W