SuperH

SuperH^{TM} (SH) 64-Bit RISC Series

SH-5 CPU Core, Volume 4: Implementation

Last updated 22 February 2002

SuperH, Inc.
San Jose, U.S.A. - Bristol, United Kingdom - Tokyo, Japan

www.superh.com

Contents

Preface **vii**

SuperH SH-5 document identification and control vii

SuperH SH-5 CPU core documentation suite viii

1 Overview **1**

1.1 Introduction 1

1.2 Undefined behavior and values 2

1.3 SH compatibility model 3

1.4 Floating-point unit (FPU) 3

2 Architectural state **5**

2.1 Implementation-specific properties 5

3 Data representation **7**

3.1 Implementation-specific properties 7

4 SHmedia instruction set **9**

4.1 Implementation-specific properties 9

4.2 Instruction formats 11

4.3 Integer instructions 11
4.4 Floating-point instructions 12
4.4.1 Floating-point status and control register 12
4.5 System instructions 14
4.5.1 Control register instructions 14
4.5.2 Configuration register instructions 15

5 SHcompact instruction set 17
5.1 Implementation-specific properties 17
5.2 Floating-point modes 17

6 Control and configuration registers 19
6.1 Specification 19
6.2 Control registers 21
6.2.1 SR 21
6.2.2 SSR 26
6.2.3 PSSR 31
6.2.4 INTEVT 36
6.2.1 EXPEVT 37
6.2.5 PEXPEVT 38
6.2.6 TRA 39
6.2.7 SPC 40
6.2.8 PSPC 41
6.2.9 RESVEC 42
6.2.10 VBR 44
6.2.11 TEA 45
6.2.12 DCR, KCR0, KCR1 46
6.2.13 CTC 47
6.2.14 USR 48
6.2.15 Reserved control registers 50
6.2.16 Undefined control registers 50

6.3 Configuration registers 51

7 Event handling 55

7.1 Implementation-specific properties 55

7.1.1 Handler address calculation 55

7.1.2 RTE 55

7.1.3 Power-on reset state 56

7.2 System architecture properties 56

7.2.1 Resets and interrupts 56

7.2.2 Debug features 56

7.2.3 Power management 56

8 Memory management 57

8.1 SH-5 MMU organization 57

8.2 SH-5 PTE contents 58

8.2.1 Enable (PTEH.V) 58

8.2.2 Page size (PTEL.SZ) 58

8.2.3 Cache behavior (PTEL.CB) 58

8.2.4 Protection (PTEL.PR) 59

8.2.5 Physical page number (PTEL.PPN) 59

8.2.6 Shared page (PTEH.SH) 59

8.2.7 Address space identifier (PTEH.ASID) 60

8.2.8 Effective page number (PTEH.EPN) 60

8.3 SH-5 translation 60

8.4 SH-5 MMU and caches 60

8.5 SH-5 MMU configuration registers 61

8.5.1 MMU configuration register map 61

8.5.2 MMUIR 61

8.5.3 MMUDR 67

8.6 MMU code sequences 74

8.6.1 Enabling and disabling the MMU 74

8.6.2 Enabling and disabling a PTE 78

8.7 Future MMU implementations 79
8.7.1 MMU architecture parameters 79
8.7.2 MMU implementation parameters 80

9 Caches 81

9.1 SH-5 cache implementation 81
9.1.1 SH-5 cache organization 81
9.1.2 SH-5 cache synonyms and aliases 83
9.1.3 SH-5 cache replacement 83
9.1.4 SH-5 cache locking mechanism 86
9.1.5 SH-5 cache instructions 87
9.1.6 SH-5 cache access 90
9.2 SH-5 cache configuration registers 93
9.2.1 Access to ICCR and OCCR 93
9.2.2 Access to tag and data configuration registers 93
9.2.3 Cache configuration register map 94
9.2.4 ICCR 96
9.2.5 ICACHETAG 98
9.2.6 ICACHEDATA 101
9.2.7 OCCR 102
9.2.8 OCACHETAG 106
9.2.9 OCACHEDATA 111
9.3 Cache code sequences 112
9.3.1 Cache initialization sequence 112
9.3.2 Cache coherency sequences 112
9.3.3 Cache locking sequence 114
9.4 Future cache implementations 115
9.4.1 Cache architecture parameters 115
9.4.2 Cache implementation parameters 116

Index 117

Preface

This document is part of the SuperH SH-5 CPU core documentation suite detailed below. Comments on this or other books in the documentation suite should be made by contacting your local sales office or distributor.

SuperH SH-5 document identification and control

Each book in the documentation suite carries a unique identifier in the form:

05-CC-nnnnn Vx.x

Where, *n* is the document number and *x.x* is the revision.

Whenever making comments on a SuperH SH-5 document the complete identification 05-CC-1000n Vx.x should be quoted.

SuperH SH-5 CPU core documentation suite

The SuperH SH-5 CPU core documentation suite comprises the following volumes:

- SH-5 CPU Core, Volume 1: Architecture (05-CC-10001)
- SH-5 CPU Core, Volume 2: SHmedia (05-CC-10002)
- SH-5 CPU Core, Volume 3: SHcompact (05-CC-10003)
- SH-5 CPU Core, Volume 4: Implementation (05-CC-10004)

SuperH

1 Overview

1.1 Introduction

This document describes the implementation-defined properties of the SH-5 CPU.

These properties can vary between different implementations of the architecture. This information should therefore not be exploited where portability of software to other implementations is desired.

The information here is intended to be read in conjunction with the generic architecture documents. The chapters in this document correspond to chapters in the generic architecture documents as follows:

1 *Chapter 1: Overview on page 1* to *Chapter 3: Data representation on page 31* correspond to similarly named chapters in *Volume 1: Architecture (05-CC-10001 V1.0).*

2 *Chapter 4: SHmedia instruction set on page 9* corresponds to:

- Chapters describing SHmedia in *Volume 1: Architecture (05-CC-10001 V1.0).*
- The instruction set specification in *Volume 2: SHmedia (05-CC-10002 V1.0).*

3 *Chapter 5: SHcompact instruction set on page 17* corresponds to:

- Chapters describing SHcompact in *Volume 1: Architecture (05-CC-10001 V1.0).*
- The instruction set specification in *Volume 3: SHcompact (05-CC-10003 V1.0).*

4 *Chapter 6: Control and configuration registers on page 19* to *Chapter 9: Caches on page 81* correspond to similarly named chapters in *Volume 1: Architecture (05-CC-10001 V1.0).*

1.2 Undefined behavior and values

In certain situations the architecture permits an implementation to exhibit architecturally-undefined behavior and to return architecturally-undefined values.

This implementation distinguishes three different kinds of architecturally-undefined behavior:

- Implementation-defined behavior: the implementation provides a well-defined behavior that is described in this document but is specific to this implementation.
- Implementation-undefined behavior: the implementation does not provide a well-defined behavior causing unreliable instruction execution. An instruction, or sequence of instructions, that exhibits implementation-undefined behavior can execute incorrectly with respect to the expected semantics. This can result in, for example, an incorrect register value, an incorrect memory value or an incorrect PC. However, the implementation will continue to execute instructions and will not provide any behavior that would breach the privilege model.
- Catastrophic behavior: the behavior of the implementation is completely undefined. The execution of further instructions in the program could be inhibited. This state must be avoided by software. It is conceivable that activation of this state could shorten the operational life-time of the device.

This implementation distinguishes two different kinds of architecturally-undefined values:

- Implementation-defined value: the implementation provides a well-defined value, generated in a manner described in this document, though the value is specific to this implementation.
- Implementation-undefined value: the implementation does not define the value. The actual value is unreliable.

Software should exploit only architecturally-defined behavior and architecturally-defined values where portability to future implementations is required. Where software exercises any of the implementation behavior or values described above, portability will be impaired.

1.3 SH compatibility model

The MOVCA.L instruction (move with cache block allocation) and the OCBI instruction (operand cache block invalidate) implicitly reveal the cache line size to a program exploiting them. Both SH-4 and SH-5 have a 32-byte cache line. Hence SHcompact user-visible semantics of MOVCA.L and OCBI are compatible with SH-4.

The SH-5 MMU architecture provides all of the SH-4 permission attributes. It also supports all of the SH-4 page sizes except the 1 kbyte page size. SHcompact is compatible with SH-4 program binaries providing that they do not rely on translations with 1 kbyte granularity.

The physical address map is a property of the SH-5 system architecture.

1.4 Floating-point unit (FPU)

This specification considers two distinct versions of the SH-5 CPU core. The only difference between these two cores is that one provides the FPU and the other does not. The presence of the FPU can be detected by writing 0 to SR.FD and reading its value back to check whether floating-point was successfully enabled.

For an SH-5 implementation with an FPU, the floating-point registers and floating-point instructions are available as described in *Volume 1, Chapter 8: SHmedia floating-point* and *Volume 1, Chapter 13: SHcompact floating-point*. The SR.FD flag can be used to enable or disable the FPU as required.

For an SH-5 implementation without an FPU, the behavior is as if the FPU was permanently disabled. Any attempt to execute a floating-point opcode or to access the floating-point register state will generate an exception. It is possible to emulate the floating-point instructions and state in software.

SH-5 software should take into consideration that the FPU can be removed on some SH-5 implementations. There are significant software implications when removing the FPU:

- If the FPU is removed and the application requires floating-point support, then floating-point arithmetic software will need to be provided. Ideally, this would have the same feature set as the architectural floating-point support. For example, this software could take the form of a library of FPU routines, or of system software that emulates the floating-point instruction set and state.

- The potential for implementations both with and without an FPU can lead to multiple parameter passing conventions for floating-point values. For an implementation with an FPU, it is most efficient to pass floating-point parameters in floating-point registers. However, this is not the case for an implementation without an FPU, since access to the floating-point registers will generate an exception.

  An alternative approach is to accept inefficient operation for the implementation without an FPU, and rely on exceptions and software emulation when passing floating-point parameters. This approach allows the use of one Application Binary Interface (ABI) for all SH-5 implementations.

- SH-5 operating system software should consider that the FPU can be removed on some SH-5 implementations.For example, operating system code for saving and restoring the floating-point registers should take account of whether the FPU is present or not.

Architectural state

2

2.1 Implementation-specific properties

The architectural state has the following implementation-specific properties:

- The number of bits in effective addresses is described in *Section 3.1: Implementation-specific properties on page 7*.
- Implementation-specific properties of control registers are described in *Section 4.5.1: Control register instructions on page 14*.
- Implementation-specific properties of configuration registers are described in *Section 4.5.2: Configuration register instructions on page 15*, *Section 8.5: SH-5 MMU configuration registers on page 61* and *Section 9.2: SH-5 cache configuration registers on page 93*.
- Some registers are used as scratch state during the execution of SHcompact instructions. The scratch registers are summarized in *Table 1*.

Scratch register	Becomes implementation undefined:
R_{20} to R_{23} inclusive	when any SHcompact instruction is executed (even if the instruction causes an exception).
TR_0 to TR_3 inclusive	when any SHcompact instruction is executed (even if the instruction causes an exception).
FR_{33}	when any SHcompact floating-point instruction is executed (even if the instruction causes an exception).

Table 1: Scratch registers

SuperH

Data representation 3

3.1 Implementation-specific properties

The number of implemented bits of effective address, neff, is defined in *Table 2*.

Quantity	Value
neff	32

Table 2: neff

The value of neff is used to size the implemented part of architectural state that contains effective addresses:

- Registers that hold effective addresses of instructions:
  - The program counter (PC).
  - Target registers (TR).
  - The following control registers: SPC, PSPC, RESVEC and VBR.
  - Configuration registers that hold effective addresses of instructions.
- Registers that hold effective addresses of data:
  - The following control register: TEA.
  - Configuration registers that hold effective addresses of data.

PC overflow occurs when any instruction is executed and the PC has one of the following values:

1 0xFFFFFFFFFFFFFFFE: this is $2^{64} - 2$

2 0xFFFFFFFFFFFFFFFC: this is $2^{64} - 4$

3 0x000000007FFFFFFE: this is $2^{31} - 2$

4 0x000000007FFFFFFC: this is $2^{31} - 4$

Note that cases *1* and *3* can only occur for SHcompact instructions, while cases *2* and *4* can occur for both SHmedia and SHcompact instructions. If a PC overflow occurs, the behavior is implementation undefined.

SHmedia instruction set

4

4.1 Implementation-specific properties

Table 3 summarizes the SHmedia instructions with implementation-specific properties.

Instruction	Implementation-specific property	Reference
Branch instructions: BEQ, BEQI, BGE, BGEU, BGT, BGTU, BNE, BNEI	Interpretation of the I-bit. This interpretation can affect performance but does not affect semantics.	See *Section 4.3*
Prepare-target instructions: PTA, PTABS, PTB, PTREL	Interpretation of the I-bit. This interpretation can affect performance but does not affect semantics.	See *Section 4.3*
Memory prefetch instructions: LD.B, LD.L, LD.Q, LD.UB, LD.UW, LD.W, LDX.B, LDX.L, LDX.Q LDX.UB, LDX.UW, LDX.W where destination is R_{63}	Effect of prefetch on performance	See *Section 9.1.5*
Approximate floating-point: FIPR.S, FTRV.S FCOSA.S, FSINA.S, FSRRA.S	Algorithms used to compute approximate floating-point results are implementation dependent.	The algorithms used by the SH-5 implementation are not described.

Table 3: SHmedia instructions with implementation-specific properties

Instruction	Implementation-specific property	Reference
Control register access: GETCON, PUTCON	Control registers have some implementation-dependent properties. Future implementations can provide additional control registers and fields.	See *Section 4.5.1*
Configuration register access: GETCFG, PUTCFG	The set of configuration registers and their fields is highly implementation dependent.	See *Section 4.5.2*
Return from exception: RTE	Effect of RTE with inappropriate SPC or SSR	See *Section 7.1.2*
Enter sleep mode: SLEEP	Behavior in sleep mode. The behavior affects power consumption but does not affect semantics.	See *Section 7.2*
Cache instructions: ALLOCO, ICBI, OCBI, OCBP, OCBWB, PREFI	Cache block size. Presence, organization and size of any caches Effect of ALLOCO on memory values Effect of PREFI on performance	See *Section 9.1.5*

Table 3: SHmedia instructions with implementation-specific properties

4.2 Instruction formats

The architecture requires that reserved fields in the instruction encodings are set to specific values. The behavior of SH-5 for architecturally-undefined cases is described in *Table 4*. The rules used to determine the type of an unused operand are listed in *Volume 1, Chapter 4: SHmedia instructions*.

Bit-field type	Reserved bits	Required value	Behavior for inappropriate value
r	Reserved bits in a used operand field	0	Implementation undefined
x	Unused general-purpose source operand	0b111111 (this corresponds to a read of R63)	Implementation undefined
	Unused general-purpose destination operand	0b111111 (this corresponds to a write to R63)	Implementation undefined
	Unused floating-point source operand	Set to the same value as the used floating-point source operand	Implementation undefined
	Unused floating-point destination operand	0b000000	Implementation undefined

Table 4: Reserved encoding fields

4.3 Integer instructions

Prepare-target and conditional branch instructions include an l-bit ('l' for likely) which can be used to pass a performance hint to the implementation. The l-bit has no architectural effect on the behavior of the instruction.

4.4 Floating-point instructions

The architecture supports implementations with an FPU and without an FPU. The presence of the FPU can be detected by writing 0 to SR.FD and reading its value back to check whether floating-point was successfully enabled.

4.4.1 Floating-point status and control register

Table 5 specifies the floating-point status and control register, FPSCR.

FPSCR					
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
RM	0	1	No	Floating-point rounding mode	RW
	Operation		If 0x0: round to nearest If 0x1: round to zero		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		
FLAG	[2,6]	5	No	Floating-point exceptions sticky flags	RW
	Operation		Bit 2 of FPSCR: sticky flag for inexact exceptions (I) Bit 3 of FPSCR: sticky flag for underflow exceptions (U) Bit 4 of FPSCR: sticky flag for overflow exceptions (O) Bit 5 of FPSCR: sticky flag for divide by zero exceptions (Z) Bit 6 of FPSCR: sticky flag for invalid exceptions (V)		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		

Table 5: FPSCR

FPSCR					
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
ENABLE	[7,11]	5	No	Floating-point exceptions enable flags	RW
	Operation		Bit 7 of FPSCR: enable flag for inexact exceptions (I) Bit 8 of FPSCR: enable flag for underflow exceptions (U) Bit 9 of FPSCR: enable flag for overflow exceptions (O) Bit 10 of FPSCR: enable flag for divide by zero exceptions (Z) Bit 11 of FPSCR: enable flag for invalid exceptions (V)		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		
CAUSE	[12,17]	6	No	Floating-point exceptions cause flags	RW
	Operation		Bit 12 of FPSCR: cause flag for inexact exceptions (I) Bit 13 of FPSCR: cause flag for underflow exceptions (U) Bit 14 of FPSCR: cause flag for overflow exceptions (O) Bit 15 of FPSCR: cause flag for divide by zero exceptions (Z) Bit 16 of FPSCR: cause flag for invalid exceptions (V) Bit 17 of FPSCR: cause flag for FPU error exceptions (E)		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		

Table 5: FPSCR

FPSCR					
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
DN	18	1	No	Floating-point denormalization mode	RW
	Operation		If 0: a denormalized source operand raises an FPU error exception If 1: a denormalized source operand is flushed to zero before the floating-point operation is performed, and a denormalized result is flushed to zero after the floating-point operation is performed		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		
RES	1, [19,31]	14	No	RESERVED	RES
	Operation		Software should always write 0 to these bits. Software should always ignore the value read from these bits.		
	When read		Reads as 0 (behavior of future implementations may vary)		
	When written		Writes ignored (behavior of future implementations may vary)		
	Power-On reset		0 (behavior of future implementations may vary)		

Table 5: FPSCR

4.5 System instructions

4.5.1 Control register instructions

A PUTCON to an UNDEFINED control register causes implementation-undefined behavior. A GETCON from an UNDEFINED control register returns an implementation-undefined value. Where the power-on reset value of a control register or field is architecturally undefined, it is also implementation undefined.

The DEFINED SH-5 control registers are specified in *Section 6.2: Control registers on page 21*.

4.5.2 Configuration register instructions

A PUTCFG to an UNDEFINED configuration register causes implementation-undefined behavior. A GETCFG from an UNDEFINED configuration register returns an implementation-undefined value.

The SH-5 configuration register map is specified in *Section 6.3: Configuration registers on page 51*.

5 SHcompact instruction set

5.1 Implementation-specific properties

Table 6 summarizes the SHcompact instructions with implementation-specific properties.

Instruction	Implementation-specific property	Reference
Approximate floating-point: FIPR, FTRV, FSCA, FSRRA	Algorithms used to compute approximate floating-point results are implementation dependent.	The algorithms used by the SH-5 implementation are not described.
Cache instructions: OCBI, OCBP, OCBWB, MOVCA.L, PREF	Cache block size. Presence, organization and size of any caches Effect of MOVCA.L on memory values Effect of PREF on performance	See *Section 9.1.5*

Table 6: SHmedia instructions with implementation-specific properties

5.2 Floating-point modes

The floating-point mode setting where FPSCR.PR=1 and FPSCR.SZ=1 is reserved. The behavior of SHcompact floating-point instructions is architecturally undefined in this mode. The behavior is also implementation undefined.

SuperH

6 Control and configuration registers

6.1 Specification

The standard format for describing the layout of a control or configuration register is illustrated in *Table 7*. This table is only used for DEFINED registers. Registers that are not DEFINED have no fields and are therefore not described with tables.

REGISTER			**REG**	**NUMBER**	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
FIELD	BITS	SIZE	VOLATILE	SYNOPSIS	TYPE
	Operation		OPERATION		
	When read		READ		
	When written		WRITE		
	Power-On reset		RESET		

Table 7: Standard table format for describing register layout

The capitalized fields in this table are place-holders for the following information:

- REGISTER: the name of the register.
- REG: this is CON for a control register, and CFG for a configuration register.
- NUMBER: the number of the register.
- FIELD: the name of the field.

- BITS: the bit numbers occupied by this field. The least significant bit in a register is bit 0; the most significant bit in a register is bit 63. A single number indicates a single bit. The notation [x,y] represents the inclusive contiguous range of bits starting at bit x and ending at bit y.
- SIZE: the number of bits occupied by this field.
- VOLATILE: a ‘yes’ indicates that the field is volatile, while ‘no’ indicates that the field is not volatile.
- SYNOPSIS: a summary of the purpose of this field.
- TYPE: the abbreviated type of this field (RES, EXP, RO, RW or OTHER).
- OPERATION: defines the operation of this field.
- READ: defines the behavior of this field for explicit read accesses.
- WRITE: defines the behavior of this field for explicit write accesses.
- RESET: defines the value of this field after a power-on reset.

The set of rows used to describe a field are repeated for each field in the register.

The field types are summarized in the following table.

Field type	Abbreviation	Usage
RESERVED	RES	Field is reserved
EXPANSION	EXP	Field is reserved for address space expansion
READ-ONLY	RO	Field is read-only and cannot be modified by software
READ-WRITE	RW	Field is readable and writable by software
OTHER	OTHER	Field has unusual semantics

Table 8: Register field types

Further information on the terminology used to define control and configuration registers can be found in *Volume 1, Chapter 3: Data representation*.

6.2 Control registers

This section describes the behavior and layout of the SH-5 control registers.

6.2.1 SR

SR			CON	0x0	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
S	1	1	No	Saturation control (SHcompact mode)	RW
	Operation		See *Volume 1, Chapter 11: SHcompact integer instructions*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		
IMASK	[4,7]	4	Yes	Interrupt request mask level	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		
Q	8	1	No	State for divide step (SHcompact mode)	RW
	Operation		See *Volume 1, Chapter 11: SHcompact integer instructions*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		

Table 9: SR

SR			CON	0x0	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
M	9	1	No	State for divide step (SHcompact mode)	RW
	Operation		See *Volume 1, Chapter 11: SHcompact integer instructions*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		
CD	11	1	No	Clock tick counter disable flag	RW
	Operation		See *Volume 1, Chapter 9: SHmedia system instructions*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		0		
PR	12	1	No	Floating-point precision (SHcompact mode)	RW
	Operation		See *Volume 1, Chapter 13: SHcompact floating-point* This field is available with RW semantics regardless of whether the FPU is enabled, disabled or not provided.		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		0		
SZ	13	1	No	Floating-point transfer size (SHcompact mode)	RW
	Operation		See *Volume 1, Chapter 13: SHcompact floating-point* This field is available with RW semantics regardless of whether the FPU is enabled, disabled or not provided.		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		0		

Table 9: SR

SR			CON	0x0	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
FR	14	1	No	Floating-point register bank (SHcompact mode)	RW
	Operation		See *Volume 1, Chapter 13: SHcompact floating-point* This field is available with RW semantics regardless of whether the FPU is enabled, disabled or not provided.		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		0		
FD	15	1	No	Floating-point disable flag	RW/RO
	Operation		See *Volume 1, Chapter 8: SHmedia floating-point* and *Volume 1, Chapter 13: SHcompact floating-point* This specification considers two distinct versions of the SH-5 CPU core. The only difference between these two cores is that one provides the floating-point unit and the other does not. The behavior of this field differs between the two versions: If FPU is provided: RW If FPU is not provided: RO		
	When read		Returns current value		
	When written		If FPU is provided: updates current value If FPU is not provided: writes ignored		
	Power-On reset		1		
ASID	[16,23]	8	No	Address Space IDentifier	RO
	Operation		See *Volume 1, Chapter 17: Memory management*		
	When read		Returns current value		
	When written		Writes ignored (use RTE to modify)		
	Power-On reset		Undefined		

Table 9: SR

SR			CON	0x0	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
WATCH	26	1	Yes	Watch-point enable flag	RO
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Writes ignored (use RTE to modify)		
	Power-On reset		0		
STEP	27	1	Yes	Single-step enable flag	RO
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Writes ignored (use RTE to modify)		
	Power-On reset		0		
BL	28	1	Yes	Flag to block exception, trap or interrupt	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		1		
MD	30	1	Yes	User or privileged mode	RO
	Operation		MD=1: Privileged mode MD=0: User mode See *Volume 1, Chapter 2: Architectural state*		
	When read		Returns current value		
	When written		Writes ignored (use RTE to modify)		
	Power-On reset		1		

Table 9: SR

SR			CON	0x0	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
MMU	31	1	Yes	MMU enable flag	RO
	Operation		See *Volume 1, Chapter 17: Memory management*		
	When read		Returns current value		
	When written		Writes ignored (use RTE to modify)		
	Power-On reset		0		
RES	0, [2,3], 10, [24,25], 29, [32,63]	39	No	RESERVED	RES
	Operation		When reading from this register, software should not interpret the value of these bits. When writing to this register, software should write these bits using a value previously read from this register. If no appropriate previous value is available, then software should write these bits as 0.		
	When read		Reads as 0 (behavior of other implementations may vary)		
	When written		Writes ignored (behavior of other implementations may vary)		
	Power-On reset		0 (behavior of other implementations may vary)		

Table 9: SR

6.2.2 SSR

SSR			CON	0x1	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
S	1	1	Yes	Saturation control (SHcompact mode)	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		
IMASK	[4,7]	4	Yes	Interrupt request mask level	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		
Q	8	1	Yes	State for divide step (SHcompact mode)	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		
M	9	1	Yes	State for divide step (SHcompact mode)	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		

Table 10: SSR

SSR			CON	0x1	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
CD	11	1	Yes	Clock tick counter disable flag	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		
PR	12	1	Yes	Floating-point precision (SHcompact mode)	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		
SZ	13	1	Yes	Floating-point transfer size (SHcompact mode)	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		
FR	14	1	Yes	Floating-point register bank (SHcompact mode)	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		

Table 10: SSR

SSR			CON	0x1	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
FD	15	1	Yes	Floating-point disable flag	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value If the FPU is not provided by the implementation (i.e. SR.FD is read-only and reads as 1), the value written to SSR should not set SSR.FD to 0. This condition will lead to architecturally-undefined behavior if RTE is then used.		
	Power-On reset		Undefined		
ASID	[16,23]	8	Yes	Address Space IDentifier	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		
WATCH	26	1	Yes	Watch-point enable flag	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		
STEP	27	1	Yes	Single-step enable flag	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		

Table 10: SSR

SSR			CON	0x1	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
BL	28	1	Yes	Flag to block exception, trap or interrupt	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		
MD	30	1	Yes	User or privileged mode	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		
MMU	31	1	Yes	MMU enable flag	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		

Table 10: SSR

SSR			**CON**	**0x1**	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
RES	0, [2,3], 10, [24,25], 29, [32,63]	39	No	RESERVED	RES
	Operation		When reading from this register, software should not interpret the value of these bits. When writing to this register, software should write these bits using a value previously read from this register. If no appropriate previous value is available, then software should write these bits as 0. Where possible, software should preserve all reserved bits of SSR from the point of launch to the point of return from launch.		
	When read		Reads as 0 (behavior of other implementations may vary)		
	When written		Writes ignored (behavior of other implementations may vary)		
	Power-On reset		0 (behavior of other implementations may vary)		

Table 10: SSR

6.2.3 PSSR

PSSR			CON	0x2	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
S	1	1	Yes	Saturation control (SHcompact mode)	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		
IMASK	[4,7]	4	Yes	Interrupt request mask level	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		
Q	8	1	Yes	State for divide step (SHcompact mode)	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		
M	9	1	Yes	State for divide step (SHcompact mode)	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		

Table 11: PSSR

PSSR			CON	0x2	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
CD	11	1	Yes	Clock tick counter disable flag	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		
PR	12	1	Yes	Floating-point precision (SHcompact mode)	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		
SZ	13	1	Yes	Floating-point transfer size (SHcompact mode)	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		
FR	14	1	Yes	Floating-point register bank (SHcompact mode)	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		

Table 11: PSSR

PSSR			CON	0x2	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
FD	15	1	Yes	Floating-point disable flag	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		
ASID	[16,23]	8	Yes	Address Space IDentifier	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		
WATCH	26	1	Yes	Watch-point enable flag	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		
STEP	27	1	Yes	Single-step enable flag	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		

Table 11: PSSR

PSSR			CON	0x2	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
BL	28	1	Yes	Flag to block exception, trap or interrupt	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		
MD	30	1	Yes	User or privileged mode	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		
MMU	31	1	Yes	MMU enable flag	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		

Table 11: PSSR

PSSR			CON	0x2	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
RES	0, [2,3], 10, [24,25], 29, [32,63]	39	No	RESERVED	RES
	Operation		When reading from this register, software should not interpret the value of these bits. When writing to this register, software should write these bits using a value previously read from this register. If no appropriate previous value is available, then software should write these bits as 0. Where possible, software should preserve all reserved bits of PSSR from the launch of a panic handler launch to the return from that panic handler.		
	When read		Reads as 0 (behavior of other implementations may vary)		
	When written		Writes ignored (behavior of other implementations may vary)		
	Power-On reset		0 (behavior of other implementations may vary)		

Table 11: PSSR

6.2.4 INTEVT

INTEVT			CON	0x4	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
CODE	[0,31]	32	Yes	Holds the event code for most recent interrupt launch	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		
RES	[32,63]	32	No	RESERVED	RES
	Operation		When reading from this register, software should not interpret the value of these bits. When writing to this register, software should write these bits using a value previously read from this register. If no appropriate previous value is available, then software should write these bits as 0.		
	When read		Reads as 0 (behavior of other implementations may vary)		
	When written		Writes ignored (behavior of other implementations may vary)		
	Power-On reset		0 (behavior of other implementations may vary)		

Table 12: INTEVT

6.2.1 EXPEVT

EXPEVT			CON	0x5	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
CODE	[0,31]	32	Yes	Holds the event code for most recent exception launch	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		0		
RES	[32,63]	32	No	RESERVED	RES
	Operation		When reading from this register, software should not interpret the value of these bits. When writing to this register, software should write these bits using a value previously read from this register. If no appropriate previous value is available, then software should write these bits as 0.		
	When read		Reads as 0 (behavior of other implementations may vary)		
	When written		Writes ignored (behavior of other implementations may vary)		
	Power-On reset		0 (behavior of other implementations may vary)		

Table 13: EXPEVT

6.2.5 PEXPEVT

PEXPEVT			CON	0x6	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
CODE	[0,31]	32	Yes	Holds the event code of the event which was being handled before the panic.	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		
RES	[32,63]	32	No	RESERVED	RES
	Operation		When reading from this register, software should not interpret the value of these bits. When writing to this register, software should write these bits using a value previously read from this register. If no appropriate previous value is available, then software should write these bits as 0.		
	When read		Reads as 0 (behavior of other implementations may vary)		
	When written		Writes ignored (behavior of other implementations may vary)		
	Power-On reset		0 (behavior of other implementations may vary)		

Table 14: PEXPEVT

6.2.6 TRA

TRA			CON	0x7	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
VALUE	[0,31]	32	Yes	Holds the lower 32 bits of the operand value from a TRAPA instruction	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		
RES	[32,63]	32	No	RESERVED	RES
	Operation		When reading from this register, software should not interpret the value of these bits. When writing to this register, software should write these bits using a value previously read from this register. If no appropriate previous value is available, then software should write these bits as 0.		
	When read		Reads as 0 (behavior of other implementations may vary)		
	When written		Writes ignored (behavior of other implementations may vary)		
	Power-On reset		0 (behavior of other implementations may vary)		

Table 15: TRA

6.2.7 SPC

SPC			CON	0x8	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
ISA	0	1	Yes	ISA mode to be used after returning from event	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		
ADDR	[1,31]	31	Yes	Address to return to after handling event	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value The value written to SPC should not set both SPC.ISA to 1 and the lowest bit of SPC.ADDR to 1. This condition will lead to architecturally-undefined behavior if RTE is then used.		
	Power-On reset		Undefined		
EXP	[32,63]	32	No	EXPANSION	EXP
	Operation		These bits may be used on other implementations to expand the address space using a sign-extended convention. Software should always write a sign-extension of bit 31 into these bits. This approach is necessary if software on this implementation is to be executed on another implementation with more implemented address space.		
	When read		Reads as a sign-extension of bit 31 (behavior of other implementations may vary)		
	When written		Writes ignored (behavior of other implementations may vary)		
	Power-On reset		Sign-extension of bit 31 (behavior of other implementations may vary)		

Table 16: SPC

6.2.8 PSPC

PSPC			CON	0x9	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
ISA	0	1	Yes	The value of SPC.ISA prior to last panic	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		
ADDR	[1,31]	31	Yes	The value of SPC.ADDR prior to last panic	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		
EXP	[32,63]	32	No	EXPANSION	EXP
	Operation		These bits may be used on other implementations to expand the address space using a sign-extended convention. Software should always write a sign-extension of bit 31 into these bits. This approach is necessary if software on this implementation is to be executed on another implementation with more implemented address space.		
	When read		Reads as a sign-extension of bit 31 (behavior of other implementations may vary)		
	When written		Writes ignored (behavior of other implementations may vary)		
	Power-On reset		Sign-extension of bit 31 (behavior of other implementations may vary)		

Table 17: PSPC

6.2.9 RESVEC

RESVEC			CON	0xA	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
MMUOFF	0	1	No	MMU (and hence cache) disable	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		0		
RES	1	1	No	RESERVED	RES
	Operation		When reading from this register, software should not interpret the value of these bits. When writing to this register, software should write these bits using a value previously read from this register. If no appropriate previous value is available, then software should write these bits as 0.		
	When read		Reads as 0 (behavior of other implementations may vary)		
	When written		Writes ignored (behavior of other implementations may vary)		
	Power-On reset		0 (behavior of other implementations may vary)		
ADDR	[2,31]	30	Yes	Reset vector	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		0		

Table 18: RESVEC

RESVEC			CON	0xA	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
EXP	[32,63]	32	No	EXPANSION	EXP
	Operation		These bits may be used on other implementations to expand the address space using a sign-extended convention. Software should always write a sign-extension of bit 31 into these bits. This approach is necessary if software on this implementation is to be executed on another implementation with more implemented address space.		
	When read		Reads as a sign-extension of bit 31 (behavior of other implementations may vary)		
	When written		Writes ignored (behavior of other implementations may vary)		
	Power-On reset		Sign-extension of bit 31 (behavior of other implementations may vary)		

Table 18: RESVEC

6.2.10 VBR

VBR			CON	0xB	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
RES	[0,1]	2	No	RESERVED	RES
	Operation		When reading from this register, software should not interpret the value of these bits. When writing to this register, software should write these bits using a value previously read from this register. If no appropriate previous value is available, then software should write these bits as 0.		
	When read		Reads as 0 (behavior of other implementations may vary)		
	When written		Writes ignored (behavior of other implementations may vary)		
	Power-On reset		0 (behavior of other implementations may vary)		
ADDR	[2,31]	30	No	Vector Base Register	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		0		
EXP	[32,63]	32	No	EXPANSION	EXP
	Operation		These bits may be used on other implementations to expand the address space using a sign-extended convention. Software should always write a sign-extension of bit 31 into these bits. This approach is necessary if software on this implementation is to be executed on another implementation with more implemented address space.		
	When read		Reads as a sign-extension of bit 31 (behavior of other implementations may vary)		
	When written		Writes ignored (behavior of other implementations may vary)		
	Power-On reset		Sign-extension of bit 31 (behavior of other implementations may vary)		

Table 19: VBR

6.2.11 TEA

TEA			CON	0xD	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
ADDR	[0,31]	32	Yes	This field contains the lowest 32 bits of the address which triggered the most recent instruction fetch or memory access exception. The upper 32 bits of the address are discarded.	RW
	Operation		See *Volume 1, Chapter 16: Event handling*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		
EXP	[32,63]	32	No	EXPANSION	EXP
	Operation		These bits may be used on other implementations to expand the address space using a sign-extended convention. Software should always write a sign-extension of bit 31 into these bits. This approach is necessary if software on this implementation is to be executed on another implementation with more implemented address space.		
	When read		Reads as a sign-extension of bit 31 (behavior of other implementations may vary)		
	When written		Writes ignored (behavior of other implementations may vary)		
	Power-On reset		Sign-extension of bit 31 (behavior of other implementations may vary)		

Table 20: TEA

6.2.12 DCR, KCR0, KCR1

DCR			CON	0x10	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
VALUE	[0,63]	64	No	Debug control register	RW
	Operation		Provides privileged state for use by debug software.		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		

Table 21: DCR

KCR0			CON	0x11	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
VALUE	[0,63]	64	No	Kernel control register 0	RW
	Operation		Provides privileged state for use by kernel software.		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		

Table 22: KCR0

KCR1			CON	0x12	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
VALUE	[0,63]	64	No	Kernel control register 1	RW
	Operation		Provides privileged state for use by kernel software.		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		

Table 23: KCR1

6.2.13 CTC

CTC			CON	0x3E	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
TICKS	[0,31]	32	Yes	Clock tick counter	RW
	Operation		The clock tick counter is decremented by 1 on each CPU clock cycle. The counter wraps around to its maximum value when it decrements past zero. The frequency of the CPU clock is system dependent.		
	When read		If SR.MD is 0 and SR.CD is 1, returns zero If SR.MD is 1 or SR.CD is 0, returns current value		
	When written		If SR.MD is 0, writes ignored If SR.MD is 1, updates current value		
	Power-On reset		Undefined		
RES	[32,63]	32	No	RESERVED	RES
	Operation		When reading from this register, software should not interpret the value of these bits. When writing to this register, software should write these bits using a value previously read from this register. If no appropriate previous value is available, then software should write these bits as 0.		
	When read		Reads as 0 (behavior of other implementations may vary)		
	When written		Writes ignored (behavior of other implementations may vary)		
	Power-On reset		0 (behavior of other implementations may vary)		

Table 24: CTC

SH-5 provides a 32-bit clock tick counter.

6.2.14 USR

USR			CON	0x3F	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
GPRS	[0,7]	8	Yes	Dirty bits for general-purpose registers	RW
	Operation		See *Volume 1, Chapter 15: Control registers*		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		
FPRS	[8,15]	8	Yes	Dirty bits for floating-point registers	RW
	Operation		See *Volume 1, Chapter 15: Control registers* This field is available with RW semantics regardless of whether the FPU is enabled, disabled or not provided.		
	When read		Returns current value		
	When written		Updates current value		
	Power-On reset		Undefined		
RES	[16,63]	48	No	RESERVED	RES
	Operation		When reading from this register, software should not interpret the value of these bits. When writing to this register, software should write these bits using a value previously read from this register. If no appropriate previous value is available, then software should write these bits as 0.		
	When read		Reads as 0 (behavior of other implementations may vary)		
	When written		Writes ignored (behavior of other implementations may vary)		
	Power-On reset		0 (behavior of other implementations may vary)		

Table 25: USR

The architectural properties of USR are described in *Volume 1, Chapter 15: Control registers*. The implementation-specific properties for SH-5 are described below:

- SH-5 implements all 8 bits of USR.GPRS. The dirty status of general-purpose registers can be monitored on subsets containing 8 registers. The hardware sets the appropriate bit in USR.GPRS when a register in the subset is written to.
- SH-5 implements all 8 bits of USR.FPRS. The dirty status of floating-point registers can be monitored on subsets containing 8 registers. The hardware sets the appropriate bit in USR.FPRS when a register in the subset is written to.
- After power-on reset, the values of USR.GPRS and USR.FPRS are undefined. They must be initialized before use by software.
- A write to R_{63} does not set the appropriate dirty bit (bit 7 of USR.GPRS).
- A GETCON from USR will read the value of USR before marking the destination register of the GETCON as dirty.
- The SH-5 implementation guarantees that a GETCON from USR will observe all registers that have become dirty due to previous instructions (since the last PUTCON to USR). Note that R_{63} always reads as zero and is never considered dirty by the SH-5 implementation.
- The SH-5 implementation updates the dirty bits imprecisely:
  - For event launches: the dirty bits can be updated imprecisely for instructions that are partially executed, but do not complete (i.e. they are cancelled), due to the processor accepting an event and launching an event handler.
  - For branches: the dirty bits can be updated imprecisely for instructions that are partially executed, but do not complete (i.e. they are cancelled), from the predicted path of execution following a conditional branch instruction if the predicted branch outcome is found to be incorrect.

If the FPU is disabled or is not present (i.e. SR.FD is set to 1), the USR.FPRS field is still implemented. GETCON can be used to read USR.FPRS and PUTCON can be used to update USR.FPRS to a new value. However, when SR.FD is 1 the implementation will not implicitly set any bits in USR.FPRS to indicate dirty registers. This is because all instructions that can modify floating-point registers will raise an exception when SR.FD is 1. Additionally, when SR.FD is 1 the implementation will not update USR.FPRS imprecisely.

6.2.15 Reserved control registers

RESERVED[n] where n is in the range [0,29]			**CON**	**0x20 + (1 * n)**	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
RES	[0,63]	64	No	RESERVED	RES
	Operation		Software should not read nor write this register		
	When read		Reads as 0 (behavior of other implementations may vary)		
	When written		Writes ignored (behavior of other implementations may vary)		
	Power-On reset		0 (behavior of other implementations may vary)		

Table 26: RESERVED[n]

6.2.16 Undefined control registers

All other control registers exhibit undefined behavior. A PUTCON to an UNDEFINED control register causes implementation-undefined behavior. A GETCON from an UNDEFINED control register returns an implementation-undefined value.

UNDEFINED[n] where n is in the range [0,16]			**CON**	**0x3, 0xC, 0xE, 0xF, [0x13, 0x1F]**	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
UNDEF	[0,63]	64	UNDEF	UNDEFINED	UNDEF
	Operation		Software must not read nor write this register		
	When read		Do not read: returns implementation-undefined value		
	When written		Do not write: causes implementation-undefined behavior		
	Power-On reset		Undefined		

Table 27: UNDEFINED[n]

6.3 Configuration registers

This section describes the configuration register map for SH-5. SH-5 uses configuration registers for MMU and cache state. These are described in *Section 8.5: SH-5 MMU configuration registers on page 61* and *Section 9.2: SH-5 cache configuration registers on page 93*.

The configuration register space is partitioned using the address conventions shown in *Table 28*.

Address bits	Interpretation	Limit	SH-5 usage
[32,63]	Must be set to zero	Not used	Not used
[24,31]	Region selection	Supports up to 256 regions	Region 0x0: MMU Region 0x1: CACHE
[21,23]	Bank selection	Supports up to 8 banks	MMU: Bank 0x0: MMUIR Bank 0x4: MMUDR CACHE: Bank 0x0: ICACHETAG Bank 0x1: ICACHEDATA Bank 0x3: ICCR Bank 0x4: OCACHETAG Bank 0x5: OCACHEDATA Bank 0x7: OCCR
[16,20]	Way selection	Supports up to 32 ways	MMU: unused, must be zero CACHE: uses ways [0,3]
[4,15]	Index selection	Supports up to 4096 indices	MMU: uses indices [0,63] CACHE: uses indices [0,255]

Table 28: SH-5 configuration register addressing

Address bits	Interpretation	Limit	SH-5 usage
[0,3]	Register selection	Supports up to 16 registers	MMU: MMUIR: uses registers [0,1] MMUDR: uses registers [0,1] CACHE: ICACHETAG: uses register 0 ICACHEDATA: uses registers [0,3] ICCR: uses registers [0,1] OCACHETAG: uses registers [0,1] OCACHEDATA: uses registers [0,3] OCCR: uses registers [0,1]

Table 28: SH-5 configuration register addressing

Table 29 summarizes the SH-5 configuration registers.

Name	Configuration register number	Number of defined registers	Behavior
MMUIR	0x00000000 + (16*index) + reg, where: index is in [0,63], reg is in [0,1]	128	See *Section 8.5.2: MMUIR on page 61*
MMUDR	0x00800000 + (16*index) + reg, where: index is in [0,63], reg is in [0,1]	128	See *Section 8.5.3: MMUDR on page 67*
ICACHETAG	0x01000000 + (65536*way) + (16*index) + reg, where: way is in [0,3] index is in [0,255], reg is 0	1024	See *Section 9.2.5: ICACHETAG on page 98*

Table 29: SH-5 configuration registers

Name	Configuration register number	Number of defined registers	Behavior
ICACHEDATA	0x01200000 + (65536*way) + (16*index) + reg where: way is in [0,3] index is in [0,255], reg is in [0,3]	4096	See *Section 9.2.6: ICACHEDATA on page 101*
ICCR	0x01600000 + reg, where reg is in [0,1]	2	See *Section 9.2.4: ICCR on page 96*
OCACHETAG	0x01800000 + (65536*way) + (16*index) + reg, where: way is in [0,3] index is in [0,255], reg is in [0,1]	2048	See *Section 9.2.8: OCACHETAG on page 106*
OCACHEDATA	0x01A00000 + (65536*way) + (16*index) + reg where: way is in [0,3] index is in [0,255], reg is in [0,3]	4096	See *Section 9.2.9: OCACHEDATA on page 111*
OCCR	0x01E00000 + reg, where reg is in [0,1]	2	See *Section 9.2.7: OCCR on page 102*
UNDEFINED	All other configuration registers	Many	UNDEFINED

Table 29: SH-5 configuration registers

7 Event handling

7.1 Implementation-specific properties

7.1.1 Handler address calculation

If the calculation of the handler address (the addition of a base register with an offset) results in an address outside of the implemented effective address space, the behavior is architecturally undefined. The behavior is also implementation undefined in this case.

7.1.2 RTE

RTE results in architecturally-undefined behavior if the values of SPC and SSR are inappropriate:

- Execution of RTE when SPC.ISA is 1 and lowest bit of SPC.ADDR is 1: this setting corresponds to a misaligned SHmedia instruction and is not supported.
- Execution of RTE when SSR.FD is 0 on an implementation without a floating-point unit: this setting corresponds to an attempt to enable the FPU when it is not supported.

In both of these cases, the behavior is also implementation undefined.

Note that there are related cases for the values of PSPC and PSSR. The RTE instruction copies PSPC to SPC and PSSR to SSR without regard to whether the values of PSPC or PSSR are inappropriate. Thus, RTE is architecturally defined when PSPR or PSSR take inappropriate values. However, if a subsequent RTE instruction is executed with inappropriate values in SPC or SSR, then the behavior is both architecturally undefined and implementation undefined as described above.

7.1.3 Power-on reset state

Any state which has an architecturally-undefined value after power-on reset is also implementation undefined.

7.2 System architecture properties

The CPU events architecture refers to a number of implementation-specific properties relating to system architecture. There properties are described in the documentation for the SH-5 system architecture. The following sections summarize these properties.

7.2.1 Resets and interrupts

The mechanisms used to deliver reset events to the CPU are properties of the system architecture. The effects of MANUAL and DEBUG resets, as far as the CPU core architecture is concerned, are the same as a power-on reset. The additional effects of these resets are specified by the system architecture.

The mechanisms used to deliver interrupt events to the CPU are properties of the system architecture. The event codes used to qualify debug interrupts and external interrupts are also determined by the system architecture.

7.2.2 Debug features

The debug vector (DBRVEC) and the vectoring mode (DBRMODE) are properties of the system architecture. DBRVEC and DBRMODE are memory-mapped registers. The CPU architecture states that these two registers exist and specifies how the values of these registers affect debug event launch. Other details of these registers, including their addresses and layout, are defined separately by the system architecture.

The mechanisms used to generate watch-point exceptions are properties of the system architecture.

7.2.3 Power management

The CPU architecture provides mechanisms to allow the CPU to be switched between sleep and active modes. SH-5 supports power-down of the CPU, and the power consumed by the CPU is significantly reduced while in sleep mode. SH-5 also supports independent power-down of the FPU, and power consumption is also reduced when the FPU is disabled.

Memory management 8

8.1 SH-5 MMU organization

SH-5 provides a fully-featured MMU that supports translation. The MMU is organized using a split PTE array, giving separate translations for instruction fetch and data access. The parameters of the MMU are given in *Table 30*.

MMU parameter	Value
Number of implemented bits of effective address space (neff)	32
Number of implemented bits of physical address space (nphys)	32
Number of supported page sizes	4
Number of supported address space identifiers	256
Organization of PTE arrays	Split
Number of entries in data PTE array	64
Number of entries in instruction PTE array	64

Table 30: SH-5 MMU parameters

8.2 SH-5 PTE contents

SH-5 implements all of the above registers and fields. For SH-5, neff and nphys are 32. Note that the upper 32 bits of all of these registers are reserved on SH-5.

8.2.1 Enable (PTEH.V)

This bit controls whether this PTE is enabled or disabled. It has no implementation-specific properties.

8.2.2 Page size (PTEL.SZ)

The SH-5 page sizes are shown in *Table 31*.

PTEL.SZ	Page size
0x0	4 kbytes
0x1	64 kbytes
0x2	1 Mbyte
0x3	512 Mbytes

Table 31: SH-5 page sizes

8.2.3 Cache behavior (PTEL.CB)

This field is specified as 2 separate bits:

- PTEL.CB0: this corresponds to bit 0 of PTEL.CB
- PTEL.CB1: this corresponds to bit 1 of PTEL.CB

The PTEL.CB field is defined in *Volume 1, Chapter 17: Memory management*. The instruction PTE arrays implement PTEL.CB0 as a reserved bit and PTEL.CB1 as a read-write bit. The data PTE arrays implement both PTEL.CB0 and PTEL.CB1 as read-write bits.

If a RESERVED cache behavior setting is written to PTEL.CB on SH-5, the values written to the reserved protection bits will be discarded and ignored. Note that the PTEL.CB field will then read with a different value to that written. Other implementations of the architecture can have different behavior, and software must not rely on the SH-5 behavior otherwise portability will be impaired. Software should ensure that RESERVED cache behavior settings are not used.

8.2.4 Protection (PTEL.PR)

This field is specified as 4 separate bits:

- PTEL.PRR: this corresponds to bit 0 of PTEL.PR
- PTEL.PRX: this corresponds to bit 1 of PTEL.PR
- PTEL.PRW: this corresponds to bit 2 of PTEL.PR
- PTEL.PRU: this corresponds to bit 3 of PTEL.PR

The instruction PTE arrays implement PTEL.PRR and PTEL.PRW as reserved bits, and PTEL.PRX and PTEL.PRU as read-write bits. The data PTE arrays implement PTEL.PRX as a reserved bit, and PTEL.PRR, PTEL.PRW and PTEL.PRU as read-write bits.

If a RESERVED protection setting is written to PTEL.PR on SH-5, the values written to the reserved protection bits will be discarded and ignored. Note that the PTEL.PR field will then read with a different value to that written. Other implementations of the architecture can have different behavior, and software must not rely on the SH-5 behavior otherwise portability will be impaired. Software should ensure that RESERVED protection settings are not used.

8.2.5 Physical page number (PTEL.PPN)

SH-5 has 20 bits in the PTEL.PPN field. The number of PPN bits required for each of the SH-5 page sizes is shown in *Table 32*.

Page size	Number of bits in PPN
4 kbytes	20
64 kbytes	16
1 Mbyte	12
512 Mbytes	3

Table 32: SH-5 PPN bits

8.2.6 Shared page (PTEH.SH)

This bit is used to control sharing of pages between different ASID values.

8.2.7 Address space identifier (PTEH.ASID)

This field is used to distinguish different effective address spaces. All bits in the 8-bit ASID field are implemented supporting 256 different ASID values.The value of PTEH.ASID is irrelevant for a shared page.

8.2.8 Effective page number (PTEH.EPN)

SH-5 has 20 bits in the PTEH.EPN field. The number of EPN bits required for each of the SH-5 page sizes is shown in *Table 33*.

Page size	Number of bits in EPN
4 kbytes	20
64 kbytes	16
1 Mbyte	12
512 Mbytes	3

Table 33: SH-5 EPN bits

8.3 SH-5 translation

SH-5 provides translation, and all PTE fields are read-write.

If the MMU is enabled and there are multiple mappings present for any effective address and ASID combination, then the behavior is architecturally undefined. In this eventuality the implementation can exhibit catastrophic behavior.

8.4 SH-5 MMU and caches

Volume 1, Chapter 17: Memory management describes constraints that are necessary to avoid cache synonyms. The value of nsynbits for SH-5 is 1. This means that cachable mappings using 4 kbyte page size are constrained by 1 synonym bit. Larger page sizes are not constrained at all.

It is highly recommended that software honors the stricter architecturally-defined nsynmax constraint, rather than the weaker implementation-specific nsynbits constraint. This guarantee allows software to arrange its memory mappings in a way that will be compatible with future implementations.

8.5 SH-5 MMU configuration registers

The MMU configuration register layout and the precise behavior of each field is implementation dependent. This section describes the layout for SH-5.

8.5.1 MMU configuration register map

SH-5 uses a split PTE array organization. PTE configuration registers are held in MMUIR for instruction access and in MMUDR for data access. The structure of the configuration registers within MMUIR and MMUDR are the same.

There are 64 instruction PTEs with 2 implemented registers per PTE in MMUIR. Similarly, there are 64 data PTEs with 2 implemented registers per PTE in MMUDR.

Name	Configuration register number	Registers in this range	Behavior
MMUIR	0x00000000 + (16*index) + reg, where: index is in [0,63], reg is in [0,1]	128	See *Section 8.5.2: MMUIR on page 61*
MMUDR	0x00800000 + (16*index) + reg, where: index is in [0,63], reg is in [0,1]	128	See *Section 8.5.3: MMUDR on page 67*

Table 34: SH-5 MMU configuration register map

8.5.2 MMUIR

Each element of the MMUIR array contains two implemented configuration registers as shown in *Table 35*.

Register	Offset	Behavior
MMUIR[n].PTEH	0	See *MMUIR[n].PTEH on page 62*
MMUIR[n].PTEL	1	See *MMUIR[n].PTEL on page 64*

Table 35: Contents of MMUIR[n]

MMUIR[n].PTEH

A PTE is enabled when PTEH.V is 1, and disabled when it is 0. Changes to the PTE must only be made when the PTE is disabled. When PTEH.V is 1:

- PUTCFG must not be used with PTEL.
- A PUTCFG to PTEH is only allowed if it clears PTEH.V (that is, disables the PTE), though it can change other PTEH fields at the same time.

When PTEH.V is 0:

- PUTCFG is allowed to both PTEL and PTEH.
- A PUTCFG to PTEH can set PTEH.V (that is, enable the PTE), and at the same time modify other PTEH fields.

MMUIR[n].PTEH where n is in the range [0,63]			**CFG**	**0x00000000 + (16 * n)**	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
V	0	1	No	Enable flag	RW
	Operation		See *Section 8.2.1: Enable (PTEH.V) on page 58*		
	When read		Returns current value		
	When written		Updates current value (see restrictions in *MMUIR[n].PTEH on page 62*)		
	Power-On reset		Undefined		
SH	1	1	No	Shared page	RW
	Operation		See *Section 8.2.6: Shared page (PTEH.SH) on page 59*		
	When read		Returns current value		
	When written		Updates current value (see restrictions in *MMUIR[n].PTEH on page 62*)		
	Power-On reset		Undefined		

Table 36: MMUIR[n].PTEH

MMUIR[n].PTEH where n is in the range [0,63]			**CFG**	**0x00000000 + (16 * n)**	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
ASID	[2,9]	8	No	Address space identifier	RW
	Operation		See *Section 8.2.7: Address space identifier (PTEH.ASID) on page 60* The value of PTEH.ASID is irrelevant for a shared page.		
	When read		Returns current value		
	When written		Updates current value (see restrictions in *MMUIR[n].PTEH on page 62*) The values [0, 255] distinguish 256 address spaces		
	Power-On reset		Undefined		
RES	[10,11]	2	No	RESERVED	RES
	Operation		Software should always write 0 to these bits. Software should not interpret the value read from these bits.		
	When read		Reads as 0 (behavior of other implementations may vary)		
	When written		Writes ignored (behavior of other implementations may vary)		
	Power-On reset		0 (behavior of other implementations may vary)		
EPN	[12,31]	20	No	Effective page number	RW
	Operation		See *Section 8.2.8: Effective page number (PTEH.EPN) on page 60*		
	When read		Returns current value		
	When written		Updates current value (see restrictions in *MMUIR[n].PTEH on page 62*)		
	Power-On reset		Undefined		

Table 36: MMUIR[n].PTEH

MMUIR[n].PTEH where n is in the range [0,63]			**CFG**	**0x00000000 + (16 * n)**	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
RESEPN	[32,63]	32	No	RESERVED for EPN expansion	RES
	Operation		These bits may be used on other implementations to expand the address space using a sign-extended convention. Software should always write a sign-extension of bit 31 into these bits. Software should not interpret the value read from these bits. This approach is necessary if software on this implementation is to be executed on another implementation with more implemented address space. Note that these bits read as zero on this implementation.		
	When read		Reads as 0 (behavior of other implementations may vary)		
	When written		Writes ignored (behavior of other implementations may vary)		
	Power-On reset		0 (behavior of other implementations may vary)		

Table 36: MMUIR[n].PTEH

MMUIR[n].PTEL

MMUIR[n].PTEL where n is in the range [0,63]			**CFG**	**0x00000001 + (16 * n)**	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
CB0	0	1	No	Cache behavior bit 0 (reserved)	RES
	Operation		See *Section 8.2.3: Cache behavior (PTEL.CB) on page 58*		
	When read		Reads as 0 (behavior of other implementations may vary)		
	When written		Writes ignored (behavior of other implementations may vary)		
	Power-On reset		0 (behavior of other implementations may vary)		

Table 37: MMUIR[n].PTEL

MMUIR[n].PTEL where n is in the range [0,63]			**CFG**	**0x00000001 + (16 * n)**	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
CB1	1	1	No	Cache behavior bit 1 (implemented)	RW
	Operation		See *Section 8.2.3: Cache behavior (PTEL.CB) on page 58*		
	When read		Returns current value		
	When written		Updates current value (see restrictions in *MMUIR[n].PTEH on page 62*)		
	Power-On reset		Undefined		
SZ	[3,4]	2	No	Page size	RW
	Operation		See *Section 8.2.2: Page size (PTEL.SZ) on page 58*		
	When read		Returns current value		
	When written		Updates current value (see restrictions in *MMUIR[n].PTEH on page 62*)		
	Power-On reset		Undefined		
PRR	6	1	No	Protection bit R (reserved)	RES
	Operation		See *Section 8.2.4: Protection (PTEL.PR) on page 59*		
	When read		Reads as 0 (behavior of other implementations may vary)		
	When written		Writes ignored (behavior of other implementations may vary)		
	Power-On reset		0 (behavior of other implementations may vary)		
PRX	7	1	No	Protection bit X (implemented)	RW
	Operation		See *Section 8.2.4: Protection (PTEL.PR) on page 59*		
	When read		Returns current value		
	When written		Updates current value (see restrictions in *MMUIR[n].PTEH on page 62*)		
	Power-On reset		Undefined		

Table 37: MMUIR[n].PTEL

MMUIR[n].PTEL where n is in the range [0,63]			**CFG**	**0x00000001 + (16 * n)**	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
PRW	8	1	No	Protection bit W (reserved)	RES
	Operation		See *Section 8.2.4: Protection (PTEL.PR) on page 59*		
	When read		Reads as 0 (behavior of other implementations may vary)		
	When written		Writes ignored (behavior of other implementations may vary)		
	Power-On reset		0 (behavior of other implementations may vary)		
PRU	9	1	No	Protection bit U (implemented)	RW
	Operation		See *Section 8.2.4: Protection (PTEL.PR) on page 59*		
	When read		Returns current value		
	When written		Updates current value (see restrictions in *MMUIR[n].PTEH on page 62*)		
	Power-On reset		Undefined		
PPN	[12,31]	20	No	Physical page number	RW
	Operation		See *Section 8.2.5: Physical page number (PTEL.PPN) on page 59*		
	When read		Returns current value		
	When written		Updates current value (see restrictions in *MMUIR[n].PTEH on page 62*)		
	Power-On reset		Undefined		
RES	2, 5, [10,11]	4	No	RESERVED	RES
	Operation		Software should always write 0 to these bits. Software should not interpret the value read from these bits.		
	When read		Reads as 0 (behavior of other implementations may vary)		
	When written		Writes ignored (behavior of other implementations may vary)		
	Power-On reset		0 (behavior of other implementations may vary)		

Table 37: MMUIR[n].PTEL

MMUIR[n].PTEL where n is in the range [0,63]			CFG	0x00000001 + (16 * n)	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
RESPPN	[32,63]	32	No	RESERVED for PPN expansion	RES
	Operation		These bits may be used on other implementations to expand the address space using a sign-extended convention. Software should always write a sign-extension of bit 31 into these bits. Software should not interpret the value read from these bits. This approach is necessary if software on this implementation is to be executed on another implementation with more implemented address space. Note that these bits read as zero on this implementation.		
	When read		Reads as 0 (behavior of other implementations may vary)		
	When written		Writes ignored (behavior of other implementations may vary)		
	Power-On reset		0 (behavior of other implementations may vary)		

Table 37: MMUIR[n].PTEL

8.5.3 MMUDR

Each element of the MMUDR array contains two implemented configuration registers as shown in *Table 38*.

Register	**Offset**	**Behavior**
MMUDR[n].PTEH	0	See *MMUDR[n].PTEH on page 68*
MMUDR[n].PTEL	1	See *MMUDR[n].PTEL on page 71*

Table 38: Contents of MMUDR[n]

MMUDR[n].PTEH

A PTE is enabled when PTEH.V is 1, and disabled when it is 0. Changes to the PTE must only be made when the PTE is disabled. When PTEH.V is 1:

- PUTCFG must not be used with PTEL.
- A PUTCFG to PTEH is only allowed if it clears PTEH.V (that is, disables the PTE), though it can change other PTEH fields at the same time.

When PTEH.V is 0:

- PUTCFG is allowed to both PTEL and PTEH.
- A PUTCFG to PTEH can set PTEH.V (that is, enable the PTE), and at the same time modify other PTEH fields.

MMUDR[n].PTEH where n is in the range [0,63]			**CFG**	**0x00800000 + (16 * n)**	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
V	0	1	No	Enable flag	RW
	Operation		See *Section 8.2.1: Enable (PTEH.V) on page 58*		
	When read		Returns current value		
	When written		Updates current value (see restrictions in *MMUDR[n].PTEH on page 68*)		
	Power-On reset		Undefined		
SH	1	1	No	Shared page	RW
	Operation		See *Section 8.2.6: Shared page (PTEH.SH) on page 59*		
	When read		Returns current value		
	When written		Updates current value (see restrictions in *MMUDR[n].PTEH on page 68*)		
	Power-On reset		Undefined		

Table 39: MMUDR[n].PTEH

MMUDR[n].PTEH where n is in the range [0,63]			CFG	0x00800000 + (16 * n)	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
ASID	[2,9]	8	No	Address space identifier	RW
	Operation		See *Section 8.2.7: Address space identifier (PTEH.ASID) on page 60* The value of PTEH.ASID is irrelevant for a shared page.		
	When read		Returns current value		
	When written		Updates current value (see restrictions in *MMUDR[n].PTEH on page 68*) The values [0, 255] distinguish 256 address spaces		
	Power-On reset		Undefined		
RES	[10,11]	2	No	RESERVED	RES
	Operation		Software should always write 0 to these bits. Software should not interpret the value read from these bits.		
	When read		Reads as 0 (behavior of other implementations may vary)		
	When written		Writes ignored (behavior of other implementations may vary)		
	Power-On reset		0 (behavior of other implementations may vary)		
EPN	[12,31]	20	No	Effective page number	RW
	Operation		See *Section 8.2.8: Effective page number (PTEH.EPN) on page 60*		
	When read		Returns current value		
	When written		Updates current value (see restrictions in *MMUDR[n].PTEH on page 68*)		
	Power-On reset		Undefined		

Table 39: MMUDR[n].PTEH

MMUDR[n].PTEH where n is in the range [0,63]			CFG	0x00800000 + (16 * n)	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
RESEPN	[32,63]	32	No	RESERVED for EPN expansion	RES
	Operation		These bits may be used on other implementations to expand the address space using a sign-extended convention. Software should always write a sign-extension of bit 31 into these bits. Software should not interpret the value read from these bits. This approach is necessary if software on this implementation is to be executed on another implementation with more implemented address space. Note that these bits read as zero on this implementation.		
	When read		Reads as 0 (behavior of other implementations may vary)		
	When written		Writes ignored (behavior of other implementations may vary)		
	Power-On reset		0 (behavior of other implementations may vary)		

Table 39: MMUDR[n].PTEH

MMUDR[n].PTEL

MMUDR[n].PTEL where n is in the range [0,63]			CFG	0x00800001 + (16 * n)	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
CB0	0	1	No	Cache behavior bit 0 (implemented)	RW
	Operation		See *Section 8.2.3: Cache behavior (PTEL.CB) on page 58*		
	When read		Returns current value		
	When written		Updates current value (see restrictions in *MMUDR[n].PTEH on page 68*)		
	Power-On reset		Undefined		
CB1	1	1	No	Cache behavior bit 1 (implemented)	RW
	Operation		See *Section 8.2.3: Cache behavior (PTEL.CB) on page 58*		
	When read		Returns current value		
	When written		Updates current value (see restrictions in *MMUDR[n].PTEH on page 68*)		
	Power-On reset		Undefined		
SZ	[3,4]	2	No	Page size	RW
	Operation		See *Section 8.2.2: Page size (PTEL.SZ) on page 58*		
	When read		Returns current value		
	When written		Updates current value (see restrictions in *MMUDR[n].PTEH on page 68*)		
	Power-On reset		Undefined		

Table 40: MMUDR[n].PTEL

MMUDR[n].PTEL where n is in the range [0,63]			**CFG**	**0x00800001 + (16 * n)**	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
PRR	6	1	No	Protection bit R (implemented)	RW
	Operation		See *Section 8.2.4: Protection (PTEL.PR) on page 59*		
	When read		Returns current value		
	When written		Updates current value (see restrictions in *MMUDR[n].PTEH on page 68*)		
	Power-On reset		Undefined		
PRX	7	1	No	Protection bit X (reserved)	RES
	Operation		See *Section 8.2.4: Protection (PTEL.PR) on page 59*		
	When read		Reads as 0 (behavior of other implementations may vary)		
	When written		Writes ignored (behavior of other implementations may vary)		
	Power-On reset		0 (behavior of other implementations may vary)		
PRW	8	1	No	Protection bit W (implemented)	RW
	Operation		See *Section 8.2.4: Protection (PTEL.PR) on page 59*		
	When read		Returns current value		
	When written		Updates current value (see restrictions in *MMUDR[n].PTEH on page 68*)		
	Power-On reset		Undefined		

Table 40: MMUDR[n].PTEL

MMUDR[n].PTEL where n is in the range [0,63]			CFG	0x00800001 + (16 * n)	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
PRU	9	1	No	Protection bit U (implemented)	RW
	Operation		See *Section 8.2.4: Protection (PTEL.PR) on page 59*		
	When read		Returns current value		
	When written		Updates current value (see restrictions in *MMUDR[n].PTEH on page 68*)		
	Power-On reset		Undefined		
PPN	[12,31]	20	No	Physical page number	RW
	Operation		See *Section 8.2.5: Physical page number (PTEL.PPN) on page 59*		
	When read		Returns current value		
	When written		Updates current value (see restrictions in *MMUDR[n].PTEH on page 68*)		
	Power-On reset		Undefined		
RES	2, 5, [10,11]	4	No	RESERVED	RES
	Operation		Software should always write 0 to these bits. Software should not interpret the value read from these bits.		
	When read		Reads as 0 (behavior of other implementations may vary)		
	When written		Writes ignored (behavior of other implementations may vary)		
	Power-On reset		0 (behavior of other implementations may vary)		

Table 40: MMUDR[n].PTEL

MMUDR[n].PTEL where n is in the range [0,63]			CFG	0x00800001 + (16 * n)	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
RESPPN	[32,63]	32	No	RESERVED for PPN expansion	RES
	Operation		These bits may be used on other implementations to expand the address space using a sign-extended convention. Software should always write a sign-extension of bit 31 into these bits. Software should not interpret the value read from these bits. This approach is necessary if software on this implementation is to be executed on another implementation with more implemented address space. Note that these bits read as zero on this implementation.		
	When read		Reads as 0 (behavior of other implementations may vary)		
	When written		Writes ignored (behavior of other implementations may vary)		
	Power-On reset		0 (behavior of other implementations may vary)		

Table 40: MMUDR[n].PTEL

8.6 MMU code sequences

This section describes code sequences that manipulate the MMU. These sequences must be executed in SHmedia and in privileged mode. Interrupts should typically be prevented across these critical code sequences.

8.6.1 Enabling and disabling the MMU

It is not possible to enable or disable the MMU using the PUTCON instruction. This is because the MMU bit of SR is read-only for PUTCON accesses. It is possible to disable the MMU during event launch, specifically the launches for reset and debug events. However, it is not possible to enable the MMU during event launch.

Otherwise, only the RTE instruction can enable or disable the MMU since it transfers the contents of SSR into the SR when it is executed. By providing suitable SPC and SSR values, the RTE instruction can atomically switch the PC and the SR to new values. This allows the MMU to be enabled or disabled at the same time as changing the PC.

Instruction fetching is automatically synchronized across an RTE instruction. The RTE instruction is fetched according to the original value of SR. Execution of the RTE instruction atomically switches PC to SPC and SR to SSR. The next executed instruction is at the new PC and is fetched according to the new SR. It is not necessary to use a SYNCI instruction. There is no requirement to use an identity translation when changing the MMU enable/disable status.

Data accesses are not automatically synchronized across an RTE instruction. When the MMU is enabled or disabled, it is necessary to use a SYNCO before the RTE in order to synchronize data accesses. This ensures that all previous data accesses are completed, including flushing of any access buffering, before the MMU status is changed. Data synchronization is important because changing the MMU status can dramatically change the cache behavior, and it is necessary to ensure that this occurs at a well-defined point in time relative to memory accesses.

The code sequences use the following conventions:

```
; - SR denotes the SR control register
; - SSR denotes the SSR control register
; - SPC denotes the SPC control register
; - MMU_BIT is the bit number of the MMU field within SR
; - R0, R1 and TR0 can be used as a temporaries
```

Using an arbitrary translation

An example recommended code sequence for enabling the MMU using an arbitrary translation is given below. The code sequence uses RTE to jump to a target instruction at the point when the MMU is enabled.

R4 specifies the address of the target instruction with the least significant bit of R4 indicating the ISA mode of that target instruction. The PTE configuration should include an executable mapping for the target address.

```
; Pre-conditions:
; - the MMU is currently disabled
; - the PTE configuration is valid
; - a PTE gives the target instruction an executable mapping
; - the cache has been appropriately configured

GETCON SR, R0 ; get current SR, must have suitable ASID value
MOVI 1, R1
SHLLI R1, MMU_BIT, R1
OR R0, R1, R0
PUTCON R0, SSR; set the target SR (with the MMU enabled)
PUTCON R4, SPC ; set the target PC
```

```
SYNCO ; synchronize data accesses
RTE

; Post-conditions:
; - execution continues at the address indicated by R4
; - execution proceeds with the MMU enabled
```

The MMU can be disabled using a similar sequence. In this case an ANDC instruction is used, instead of the OR, so that the MMU bit of SR is cleared rather than set. Also, the target instruction is specified in R4, and it refers to instructions that are executed with the MMU disabled (and no translation is required).

Using an identity translation

It is sometimes convenient to enable or disable the MMU within the confines of an identity translation. This gives a straightforward code sequence. This can be achieved by ensuring that an identity executable mapping (that is, EPN matches PPN) is provided for the entire set of instructions in the code sequence. This requires an appropriate setup of the PTE configuration registers.

An example recommended code sequence for enabling the MMU using an identity translation is:

```
; Pre-conditions:
; - the MMU is currently disabled
; - the PTE configuration is valid
; - a PTE gives these instructions an identity executable mapping
; - the cache has been appropriately configured

GETCON SR, R0 ; get current SR
MOVI 1, R1
SHLLI R1, MMU_BIT, R1
OR R0, R1, R0
PUTCON R0, SSR; set the target SR (with the MMU enabled)
PTA label, TR0 ; calculate target PC
GETTR TR0, R0
PUTCON R0, SPC ; set the target PC
SYNCO ; synchronize data accesses
RTE
label:

; Post-conditions:
; - execution continues at the address indicated by the label
; - execution proceeds with the MMU enabled
```

The MMU can be disabled using a similar sequence. In this case an ANDC instruction is used, instead of the OR, so that the MMU bit of SR is cleared rather than set.

8.6.2 Enabling and disabling a PTE

A PTE can be enabled and disabled using a simple sequence of PUTCFG instructions. When a PTE is enabled or disabled, software should execute a SYNCI or RTE instruction before any access to that PTE. This ensures that translation look-up, exception detection and memory access are performed correctly with respect to the modified PTE state.

An example recommended code sequence for enabling a PTE is:

```
; Pre-conditions:
; - R0 contains configuration space index of the PTE
; - R1 contains new PTEH value (PTEH.V is set)
; - R2 contains new PTEL value
; - OFFSET_PTEH is offset of PTEH within the PTE
; - OFFSET_PTEL is offset of PTEL within the PTE
PUTCFG R0, OFFSET_PTEH, R63; disable PTE before modification
PUTCFG R0, OFFSET_PTEL, R2 ; set new PTEL value
PUTCFG R0, OFFSET_PTEH, R1 ; set new PTEH value, enables the PTE
; Post-conditions:
; - Ensure SYNCI or RTE is executed before any access through
; - the enabled PTE. This ensures that the access is translated
; - correctly using the new PTE.
```

The value of a PTE field must not be modified while the PTE is enabled. The PTE should be disabled before modifying its contents. However, the contents of a PTE can be safely read at any time. A PTE can be disabled by:

```
; Pre-conditions:
; - R0 contains configuration space index of the PTE
; - OFFSET_PTEH is offset of PTEH within the PTE

PUTCFG R0, OFFSET_PTEH, R63
```

8.7 Future MMU implementations

Many properties of the MMU are implementation-specific and can be varied in future implementations of the architecture. The MMU implementation options are described in *Volume 1, Chapter 17: Memory management*, and the SH-5 specific properties are described in this chapter. It is intended that future MMU implementations will be based on the MMU configuration register map and configuration register definitions used by the SH-5 implementation.

Note that the information in this section does not require future implementations to use these options, nor does it constrain future implementations to just these options.

8.7.1 MMU architecture parameters

The SH-5 MMU configuration register map defined in *Section 6.3: Configuration registers on page 51* and in *Section 8.5: SH-5 MMU configuration registers on page 61* supports the different PTE array organizations described by the architecture in *Volume 1, Chapter 17: Memory management*:

- A unified organization consists of a single array of page table entries. Each entry controls the behavior of both data and instruction accesses to the described page. The number of entries in the array is implementation defined and is represented here by u. The configuration registers in the unified array are called:
  - MMUDR[n].PTEH and MMUDR[n].PTEL

  where n varies in the range [0, u).
- A split organization consists of two arrays of page table entries. An entry in the data register array controls the behavior of data accesses to the described page, whereas an entry in the instruction register array controls the behavior of instruction accesses to the described page. The number of entries in these arrays is implementation defined and is represented here by d for the data register array and i for the instruction register array. The configuration registers in the data array are called:
  - MMUDR[n].PTEH and MMUDR[n].PTEL

  where n varies in the range [0, d). The configuration registers in the instruction array are called:
  - MMUIR[n].PTEH and MMUIR[n].PTEL

  where n varies in the range [0, i).

All entries in a PTE array are equivalent. The PTE arrays are fully associative.

8.7.2 MMU implementation parameters

The layout of the SH-5 MMU configuration register map is:

- MMUIR: 0x00000000 + (16*index) + reg
- MMUDR: 0x00800000 + (16*index) + reg

The supported ‘index’ range in the configuration register map is [0, 4095] using the partitioning described in *Section 6.3: Configuration registers on page 51*. This allows each PTE array to be scaled from 0 entries to 4096 entries in future implementations. There is no requirement for the number of PTEs to be an integral power-of-two. The supported ‘reg’ range in the configuration register map allows up to 16 registers per PTE.

The MMU configuration register map limits the number of indices to a maximum value which is less than the architectural parameterization. However, there is significant room for future expansion, and this limit is unlikely to be a problem for future implementations.

Additionally, it is possible that future implementations can make extensions to the MMU configuration registers:

- More configuration registers could be provided for each PTE (i.e. in addition to PTEH and PTEL).
- Reserved fields within PTEH and PTEL could be given defined semantics on future implementations. In particular:
  - Extension of EPN (using RESEPN) to support more effective address space.
  - Extension of PPN (using RESPPN) to support more physical address space.
  - Extension of ASID to increase the number of supported address spaces.
  - Extension of CB to increase the number of supported cache behaviors.
  - Extension of SZ to increase the number of supported page sizes.
  - Extension of PR to increase the number of supported protection attributes.

Software should be carefully written with consideration given to potential future changes in the MMU implementation. Ideally, SH-5 software should be parameterized so that it can be readily updated to support future implementations.

Caches 9

9.1 SH-5 cache implementation

This section describes the implementation-specific properties of the SH-5 cache. This information should not be exploited where portability of software to other implementations is desired.

9.1.1 SH-5 cache organization

SH-5 has a split cache organization. There are separate caches for operand data and for instructions.

The SH-5 caches are indexed using an effective address and tagged by an effective address[1]. Additionally, the operand cache contains physical address tags. These allow the implementation to resolve operand cache aliases arising due to the use of effective address tagging.

The cache implementation allows most cache hits to be completed in the cache without needing to consult the PTE arrays, giving performance and power advantages relative to an implementation based on physical addresses. This approach requires that the implementation keeps some of the PTE information in cache blocks.

The MMU and cache architecture described in *Volume 1, Chapter 17: Memory management* and *Volume 1, Chapter 18: Caches* fully supports this arrangement. Note that software must ensure cache coherency when the contents of page table entries are changed. This implication is already accommodated in the architecture.

1. This arrangement is called a virtual cache in some other architectures.

The internal state of the SH-5 caches is visible through configuration registers, and is described in *Section 9.2: SH-5 cache configuration registers on page 93*.

The properties of the SH-5 caches are summarized in *Table 41*.

Property	Operand cache	Instruction cache
Cache block size, nbytes	32 bytes	32 bytes
Set size, nways (the associativity)	4 ways	4 ways
Number of sets, nsets	256 sets	256 sets
Cache size	32 kbytes	32 kbytes
Cache is indexed by:	Effective address	Effective address
Cache is tagged by:	Effective address (for effective look-up) Physical address (for physical look-up) (see *Section 9.1.6*)	Effective address
Offset bits, $\log_2(\text{nbytes})$	5	5
Index bits, $\log_2(\text{nsets})$	8	8
Implemented tag bits	19 bits of effective address tag 20 bits of physical address tag	19 bits of effective address tag No physical address tag
Cache look-up (see *Section 9.1.6*)	Usually by effective address, sometimes by physical address	Always by effective address

Table 41: SH-5 cache parameters

In a cache which uses the effective address for both indexing and tagging, the number of effective address tag bits is given by the number of bits of effective address less the number of offset and index bits. For SH-5, these numbers are 32, 5 and 8 respectively giving 19 bits for the effective address tag.

The operand cache also contains a physical address tag. The size of this tag is given by the number of bits of physical address less $\log_2$ of the smallest page size. For SH-5, these numbers are 32 and 12 giving 20 bits for the physical address tag present in the operand cache.

9.1.2 SH-5 cache synonyms and aliases

The constraints placed on software to avoid cache synonyms are described in *Section 8.4: SH-5 MMU and caches on page 60* and *Volume 1, Chapter 17: Memory management*.

SH-5 resolves cache aliases in its operand cache as required by the architecture.

SH-5 does not resolve cache aliases in its instruction cache. Since the instruction cache is 4-way associative, there can be up to 4 cache aliases of a particular physical address in the instruction cache. Software must take special care when invalidating instructions. This implication is already accommodated in the architecture as described in *Volume 1, Chapter 18: Caches*.

9.1.3 SH-5 cache replacement

Both SH-5 caches are 4-way associative. The replacement algorithm uses 6 bits of state per set to implement a least-recently-used policy (LRU). The LRU state orders the valid blocks in that set in an order determined by their last usage. This state is equivalent to an ordered list, with the head element representing the least-recently-used valid block and the tail element representing the most-recently used valid block. Invalid blocks do not appear on this list.

Additionally, SH-5 provides a cache locking mechanism. Cache locking allows software to arrange for specified memory blocks to be locked into the cache. The granularity of locking is the way. Each way in the cache may be independently locked or unlocked. Once a way is locked, that way is not a candidate for replacement, and thus normal cache operation will not evict a cache block in a locked way.

For each cachable access, the replacement policy behaves as follows:

- If the access hits the cache, then this cache block is marked as the most-recently-used by moving it to the tail of the order list.
- Otherwise, if the access misses the cache and the set contains blocks that are both invalid and unlocked, then one of those blocks is selected. If there are multiple such blocks, then this implementation selects the block with the lowest way number. The selected block is marked as the most-recently-used by moving it to the tail of the order list.
- Otherwise, if the access misses the cache and the set contains blocks that are both valid and unlocked, then one of those blocks is selected. This implementation selects the block that is least-recently-used; this is the one

nearest the head of the order list. The selected block is marked as the most-recently-used by moving it to the end of the order list.

- Otherwise, the access has missed the cache and all blocks are locked (they may be valid or invalid). In this case, there are no candidates for replacement and the access is implemented on memory with no caching.

For replacement purposes, all cache instructions count as accesses and cause the least-recently-used information to be updated as required by the above algorithm.

The SH-5 implementation uses a 6-bit field (called LRU) to record the status of the replacement policy. There is an LRU field associated with each cache set.The interpretation of the 6 LRU bits is described in *Table 42*.

LRU bit number	Meaning when clear	Meaning when set
0	Way 0 was accessed less recently than way 1	Way 0 was accessed more recently than way 1
1	Way 0 was accessed less recently than way 2	Way 0 was accessed more recently than way 2
2	Way 0 was accessed less recently than way 3	Way 0 was accessed more recently than way 3
3	Way 1 was accessed less recently than way 2	Way 1 was accessed more recently than way 2
4	Way 1 was accessed less recently than way 3	Way 1 was accessed more recently than way 3
5	Way 2 was accessed less recently than way 3	Way 2 was accessed more recently than way 3

Table 42: SH-5 LRU field

There are 64 possible combinations of the 6 LRU bits. Of these combinations, 24 correspond to valid cache states while the remaining 40 lead to contradictions and are invalid. The valid combinations are given in *Table 43* along with the corresponding way order, shown with the most recently used way on the left through to the least recently used way on the right. For example, the first row shows that the combination with all LRU bits set to zero indicates that way 3 was the most recently accessed, followed by way 2, way 1 and finally way 0.

After POWERON reset the internal state of the cache, including LRU, is undefined. It is necessary to invalidate the caches (see *Section 9.3.1: Cache initialization sequence on page 112*) so that all cache blocks are invalid before the caches are enabled. Note that the replacement algorithm selects invalid blocks in preference to valid blocks, and that the selection order for invalid blocks is independent of the LRU state. These properties ensure the cache replacement order is deterministic after cache invalidation.

LRU bit 5	LRU bit 4	LRU bit 3	LRU bit 2	LRU bit 1	LRU bit 0	Most recently accessed		Least recently accessed	
0	0	0	0	0	0	3	2	1	0
0	0	0	0	0	1	3	2	0	1
0	0	0	0	1	1	3	0	2	1
0	0	0	1	1	1	0	3	2	1
0	0	1	0	0	0	3	1	2	0
0	0	1	0	1	0	3	1	0	2
0	0	1	0	1	1	3	0	1	2
0	0	1	1	1	1	0	3	1	2
0	1	1	0	0	0	1	3	2	0
0	1	1	0	1	0	1	3	0	2
0	1	1	1	1	0	1	0	3	2
0	1	1	1	1	1	0	1	3	2
1	0	0	0	0	0	2	3	1	0
1	0	0	0	0	1	2	3	0	1
1	0	0	1	0	1	2	0	3	1
1	0	0	1	1	1	0	2	3	1
1	1	0	0	0	0	2	1	3	0
1	1	0	1	0	0	2	1	0	3
1	1	0	1	0	1	2	0	1	3
1	1	0	1	1	1	0	2	1	3
1	1	1	0	0	0	1	2	3	0

Table 43: Valid LRU combinations

LRU bit 5	LRU bit 4	LRU bit 3	LRU bit 2	LRU bit 1	LRU bit 0	Most recently accessed		Least recently accessed	
1	1	1	1	0	0	1	2	0	3
1	1	1	1	1	0	1	0	2	3
1	1	1	1	1	1	0	1	2	3

Table 43: Valid LRU combinations

It is possible to over-ride the above replacement policy to allow a prefetch into a specified way. This feature is provided to allow locked ways to be initialized, and is described in *Section 9.1.4: SH-5 cache locking mechanism on page 86*.

9.1.4 SH-5 cache locking mechanism

Cache locking is configured through cache configuration registers, and can therefore only be managed in privileged mode. The current cache locking configuration affects all threads, both user and privileged, regardless of address space identifier.

A typical usage of cache locking is to partition the cache state between cache operation and RAM operation. One or more cache ways would be locked and loaded with a set of memory locations. Those memory locations will behave as low-latency RAM, while any unlocked cache ways will continue to behave as cache.

The only effect of the cache locking mechanism is to influence the cache replacement algorithm. Other cache properties and behaviors are unaffected by the use of cache locking. When a cache block is locked into the cache, that cache block can still be modified by reads, writes, cache instructions and the normal operation of the cache. The only property that cache locking provides is to prevent a locked cache block from being chosen for replacement.

The SH-5 cache replacement algorithm, including the effects of cache locking, are described in *Section 9.1.3: SH-5 cache replacement on page 83*. Once a way is locked, that way is not a candidate for replacement, and thus normal cache operation will not evict a cache block in a locked way. This rule is applied regardless of whether the cache block is valid or invalid. Thus, an invalid cache block in a locked way is not a candidate for replacement.

It is possible to lock any or all ways in the cache. If some ways are unlocked, normal cache operation continues in all those unlocked ways. If all ways are locked, then cache misses cannot cause cache blocks to be allocated in the cache and are achieved directly on memory without any caching.

Cache coherency instructions operate directly on cache blocks regardless of whether those cache blocks are locked. The protection mechanisms provided by the MMU can be used, where required, to protect locked cache blocks against inappropriate access. Note that if a thread has executability for an instruction cache block, then the thread can invalidate that block (regardless of locking). Similarly, if a thread has writability for an operand cache block, the thread can invalidate that block (regardless of locking).

The cache provides a mechanism to over-ride the normal replacement algorithm so that memory blocks can be loaded into a specified way using prefetches. This uses the cache configuration registers defined in *I CCR0 on page 96* and *ICCR1 on page 97*. The mechanism operates as follows:

- When OCCR1.OW_LE is set to 1 and a data prefetch misses the cache and causes a cache block to be allocated, then the way specified by OW_LOAD is chosen. The choice made by the normal replacement algorithm and the lock flags for each way are ignored.
- When ICCR1.IW_LE is set to 1 and an instruction prefetch misses the cache and causes a cache block to be allocated, then the way specified by IW_LOAD is chosen. The choice made by the normal replacement algorithm and the lock flags for each way are ignored.

A suitable sequence for locking a way with certain pre-loaded data is described in *Section 9.3.3: Cache locking sequence on page 114*.

9.1.5 SH-5 cache instructions

This section describes the implementation-specific properties of the cache instructions. All cache instructions operate at cache block granularity. The cache block size is 32 bytes for SH-5.

Allocate

The SH-5 implementation of the ALLOCO instruction has the following behavior.

If the ALLOCO instruction raises an exception, there is no effect on the operand cache.

If the ALLOCO instruction does not raise an exception, then the behavior depends on whether the MMU is enabled and, if so, on the page type:

1. If the MMU is disabled, there is no effect on the operand cache.
2. If the MMU is enabled, and the resultant cache behavior of the access is device, uncached or write-through, then there is no effect on the operand cache.

3 If the MMU is enabled, and the resultant cache behavior of the access is write-back, then the behavior depends on whether there is cache hit or cache miss:

- If the access hits the operand cache, there is no effect on the operand cache.
- If the access misses the operand cache and no block can be allocated (using the LRU replacement algorithm described in *Section 9.1.3: SH-5 cache replacement on page 83*), there is no effect on the operand cache.
- If the access misses the operand cache and a block can be allocated (using the LRU replacement algorithm described in *Section 9.1.3: SH-5 cache replacement on page 83*), then that block is allocated using the following operations:
    - If the cache block is already dirty, it is written back to memory.
    - The cache block is allocated to the memory block specified by the ALLOCO access without fetch of that block from memory.
    - The allocated cache block is filled with zeroes. This ensures that ALLOCO does not reveal any data which could break the privilege and protection models. Software must not rely on this zero-fill behavior since it is highly implementation-dependent and since the conditions under which a zero-fill takes place are very specific.
    - The allocated block is marked as dirty.

The determination of the resultant cache behavior is described in *Volume 1, Chapter 18: Caches*, and depends on the global cache behavior as well as the page-level cache behavior.

Instruction cache coherency

SH-5 uses an effective-indexed, effective-tagged instruction cache with no physical tags. This has several implications for ICBI:

- The SH-5 implementation of ICBI performs instruction invalidation on effective memory only. There can be up to 4 cache aliases of a particular physical address in the instruction cache. In addition, due to cache synonyms the physical address can be present in either or both of 2 cache sets. This gives a total of 8 different instruction cache blocks that could contain the physical address. Note that software can (optionally) avoid instruction cache synonyms by placing constraints on instruction translations.
- The SH-5 implementation of ICBI never raises an ITLBMISS exception. This is because if there is an entry in the instruction cache that can be invalidated by this ICBI then the protection check can be performed using the protection

information held in the cache. There is no need to perform a translation look-up, and hence there is no need to raise ITLBMISS.

- The SH-5 implementation of ICBI has no effect on the instruction cache when the MMU is disabled. This is because the architecture requires that when the MMU is disabled, the effective address calculated by ICBI is identity translated into a physical address. However, there is no mechanism for this physical address to be used to look-up into the instruction cache since there are no physical tags.

These properties of ICBI are allowed by the architecture.

ICBI disregards cache locking information. An ICBI instruction can invalidate a locked cache block.

Operand cache coherency

These instructions disregard cache locking information. OCBI, OCBWB and OCBP instructions can invalidate, write-back and purge locked cache blocks, respectively.

Prefetch

Prefetch refers specifically to the instructions that perform software-directed prefetching. These are:

- SHmedia PREFI instruction.
- SHmedia aligned load instructions where the destination register is R_{63}.
- SHcompact PREF instruction.

The term ‘prefetch’ excludes other speculative instruction or data access provided by an implementation that is not initiated by the instructions above.

SH-5 provides cache locking and uses prefetches to preload information into locked parts of the cache. For this implementation prefetches are more than just a hint to the implementation. Providing that the requested prefetch is architecturally possible, the implementation guarantees to prefetch. This property ensures that the cache locking mechanism can be used deterministically.

Note that the generic architecture does not guarantee this prefetch property. Another implementation can choose to treat prefetches as just a hint, and disregard prefetches in some circumstances.

9.1.6 SH-5 cache access

The architectural properties of cache access are described in *Volume 1, Chapter 18: Caches*. In summary, cache access consists of the following stages:

1 The address of the access is mapped to a set in the cache through an indexing procedure. SH-5 has separate caches and the instruction cache is used for instruction fetches, and the operand cache for data accesses. For SH-5, indexing uses bits 5 to 12 (inclusive) of the address to select one of the 256 sets in the appropriate cache. Additionally for SH-5, indexing into the operand or instruction cache always uses index bits taken from the effective address and never from the physical address.

2 Each cache block in the set is checked to see if its tag matches the tag of the access. This process is called cache look-up. The cache look-up and replacement algorithm is designed so that there can be at most one match in the set. For SH-5, there are 4 cache blocks in each set since the caches are 4-way associative.

3 There are two possible outcomes of the tag comparison:

3.1 If there is no match then this is a cache miss.

3.2 If there is a match, then this is a cache hit.

This section describes the look-up policies for SH-5 in more detail. These implementation choices are summarized in *Table 44* and *Table 45*.

The operand cache supports both effective and physical address tags. Look-up is normally achieved by effective address. This requires the hardware to compare the relevant bits of the effective address with effective address tags from the relevant set in the operand cache. Look-up by physical address requires the hardware to translate the effective address into a physical address using the TLB, before performing a comparison between the relevant bits of the physical address and the physical address tags from the relevant set in the operand cache.

The physical address is used whenever a look-up by effective address misses in order to resolve potential cache aliases. For SWAP.Q, OCBI, OCBP and OCBWB, however, SH-5 does not attempt an effective address look-up and immediately performs a look-up by physical address. If the look-up by physical address misses the cache, then the physical address is not present in the cache and needs to be obtained from memory.

The rationale for the choices taken by this implementation for the operand cache is as follows:

- For this implementation, look-up by effective address is more efficient than look-up by physical address. This results in a performance advantage and power reduction when look-up by effective address is successful.
- Load, store, data prefetch and cache allocate instructions are important instructions for performance and relatively common. Additionally, in the case of a cache hit for the effective address look-up and an appropriate choice of cache behavior, it is possible for the memory access of these instructions to be completed in the cache without an access to external memory. In this case the access can be completed without any use of the physical address.
- The SWAP.Q instruction always causes an external memory access and the physical address is always required. There is no advantage in performing an effective address look-up first. In the case where the data accessed by the SWAP.Q instruction is in the operand cache, the SWAP.Q will automatically invalidate or purge that cache line as required to maintain cache coherency. The actions are described in *Volume 1, Chapter 6: SHmedia memory instructions*. However, note that correct operation of the cache always requires that cache paradoxes are avoided, see *Volume 1, Chapter 18: Caches*.
- The implementation chooses to handle OCBI, OCBP and OCBWB instructions by performing just a physical address look-up. In many cases it is necessary to refer to the physical address anyway, either to resolve potential cache aliases or to cause a write-back to external memory using the physical address (for OCBP and OCBWB only, depending on cache behavior).

  In principal, for the case of a cache hit for an effective look-up where no write-back to external memory is required, it is possible for the access to be completed without reference to the physical address. However, the implementation chooses not to take this approach and implements all 3 of these instructions in a regular way using only a look-up by physical address.

The instruction cache only supports an effective address tag, and all instruction cache look-ups are achieved using the effective address. The rationale for this choice is to eliminate physical tags in the instruction cache and to eliminate the associated look-up mechanism. This choice is possible because the architecture allows instruction cache aliases since these do not cause incorrect program behavior.

Apart from potential performance effects, the distinction between an effective and a physical address look-up is otherwise almost irrelevant to software. For the operand cache the hardware transparently uses look-up by physical address to resolve any problems resulting from cache aliases. However, software can distinguish between effective and physical address look-up due to TLB misses. A cache hit using an effective address look-up never causes a TLB miss, since the access can be completed in the cache without consulting the MMU. However, a cache hit using a physical address look-up requires a TLB entry, and will cause a TLB miss if a translation is not present.

For both caches, the LRU bits will be updated by all accesses that hit in the cache or cause a block to be allocated or refilled into the cache.

Access type	Cache index by effective address or physical address	Cache look-up by effective address or physical address?	LRU updated? (on condition)
Any load instruction (including SHmedia data prefetches)	Effective address	Effective address (then physical address if effective address misses)	Yes (hit/refill)
Any store instruction	Effective address	Effective address (then physical address if effective address misses)	Yes (hit/refill)
ALLOCO (SHmedia)	Effective address	Effective address (then physical address if effective address misses)	Yes (hit/allocate)
MOVCA.L (SHcompact)	Effective address	Effective address (then physical address if effective address misses)	Yes (hit/allocate)
PREF (SHcompact)	Effective address	Effective address (then physical address if effective address misses)	Yes (hit/refill)
SWAP.Q (SHmedia)	Effective address	Physical address	Yes (hit)
OCBI (SHmedia or SHcompact)	Effective address	Physical address	Yes (hit)
OCBP (SHmedia or SHcompact)	Effective address	Physical address	Yes (hit)
OCBWB (SHmedia or SHcompact)	Effective address	Physical address	Yes (hit)

Table 44: Operand cache look-up

Access type	Cache index by effective address or physical address	Cache look-up by effective address or physical address?	LRU updated? (condition)
PREFI (SHmedia)	Effective address	Effective address	Yes (hit/refill)
ICBI (SHmedia)	Effective address	Effective address	Yes (hit)

Table 45: Instruction cache look-up

9.2 SH-5 cache configuration registers

The cache configuration register layout and the precise behavior of each field is implementation dependent. This section describes the layout for SH-5.

SH-5 supports separate instruction and operand caches. The cache configuration registers are also split in the same way. Each cache is associated with the following registers:

- Cache configuration registers to control global cache behavior and cache locking (ICCR and OCCR).
- An array of configuration registers containing cache tag information (ICACHETAG and OCACHETAG).
- An array of configuration registers containing cache data information (ICACHEDATA and OCACHEDATA).

9.2.1 Access to ICCR and OCCR

ICCR and OCCR can be read using GETCFG and written using PUTCFG. They are used to enable caching, global cache invalidation, write-through/write-back selection (operand cache only) and management of cache locking.

A PUTCFG to ICCR must be followed by SYNCI, while a PUTCFG to OCCR must be followed by SYNCO. These instructions ensure synchronization of instruction fetch and data access while cache properties are being modified.

9.2.2 Access to tag and data configuration registers

These configuration registers can be read using GETCFG. This allows a privileged mode thread to view the internal state of the cache. This can be used in combination with cache coherency instructions to cohere specific cache blocks (see *Section 9.3.2: Cache coherency sequences on page 112* for examples). It can also be used a debugger to give visibility of cache state while debugging.

Note that the cache state is highly volatile and some care is required to achieve predictable results. The cache state can be observed in a stable state in the following situations:

- When the MMU is disabled, both the instruction cache and operand cache are frozen. The state of these caches will be non-volatile when observed through GETCFG.

- When the MMU is enabled, considerable care is needed to observe a stable cache state. One technique is to use the cache locking mechanism to prevent the cache replacement strategy from changing cache state:
  - The ICACHETAG and ICACHEDATA configuration registers, corresponding to locked instruction cache ways, will be non-volatile when observed through GETCFG.
  - For the operand cache, it is also necessary to avoid making any load or store accesses that hit the operand cache since these can result in changes to OCACHETAG and OCACHEDATA. In order to observe a stable operand cache state, software should be written to avoid using load and stores in these GETCFG sequences; this may require appropriate SYNCO barriers. In this case, the OCACHETAG and OCACHEDATA configuration registers, corresponding to locked operand cache ways, will be non-volatile when observed through GETCFG.

These configuration registers should not be written to. A PUTCFG to any of these registers leads to implementation-undefined behavior. In particular, the memory model could be compromised and the behavior of memory accesses can be unpredictable.

9.2.3 Cache configuration register map

Table 46 summarizes the SH-5 cache configuration registers.

Name	Configuration register number	Number of defined registers in this range	Behavior
ICACHETAG	0x01000000 + (65536*way) + (16*index) + reg, where: way is in [0,3] index is in [0,255] reg is 0	1024	See *Section 9.2.5: ICACHETAG on page 98*

Table 46: SH-5 cache configuration register map

Name	Configuration register number	Number of defined registers in this range	Behavior
ICACHEDATA	0x01200000 + (65536*way) + (16*index) + reg where: way is in [0,3] index is in [0,255] reg is in [0,3]	4096	See *Section 9.2.6: ICACHEDATA on page 101*
ICCR	0x01600000 + reg, where reg is in [0,1]	2	See *Section 9.2.4: ICCR on page 96*
OCACHETAG	0x01800000 + (65536*way) + (16*index) + reg, where: way is in [0,3] index is in [0,255] reg is in [0,1]	2048	See *Section 9.2.8: OCACHETAG on page 106*
OCACHEDATA	0x01A00000 + (65536*way) + (16*index) + reg where: way is in [0,3] index is in [0,255] reg is in [0,3]	4096	See *Section 9.2.9: OCACHEDATA on page 111*
OCCR	0x01E00000 + reg, where reg is in [0,1]	2	See *Section 9.2.7: OCCR on page 102*

Table 46: SH-5 cache configuration register map

9.2.4 ICCR

There are 2 instruction cache control registers: ICCR0 and ICCR1.

ICCR0

Software should exercise care when writing to this register. If instruction caching is changed from enabled to disabled, the instruction cache should simultaneously be invalidated to prevent cache paradoxes.

ICCR0			CFG	0x01600000	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
ICE	0	1	No	Instruction cache enable	RW
	Operation		If 0: instruction cache is disabled If 1: instruction cache is enabled		
	When read		Returns current value		
	When written		Updates current value		
	Power-on reset		Undefined		
ICI	1	1	No	Instruction cache invalidate	OTHER
	Operation		Write with 1 to invalidate the entire instruction cache		
	When read		Returns 0		
	When written		Write of 0: no effect Write of 1: invalidate the entire instruction cache Writes have no effect on the value of this field		
	Power-on reset		0		
RES	[2,63]	62	No	RESERVED	RES
	Operation		Software should always write 0 to these bits. Software should not interpret the value read from these bits.		
	When read		Reads as 0 (behavior of other implementations may vary)		
	When written		Writes ignored (behavior of other implementations may vary)		
	HARD reset		0 (behavior of other implementations may vary)		

Table 47: ICCR0

ICCR1

ICCR1			CFG	0x01600001	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
IW_LOAD	[0,1]	2	No	Instruction way load	RW
	Operation		Selects instruction cache way for instruction prefetch when IW_LE is set. The value of IW_LOAD selects one of the 4 ways in the instruction cache. It does not contain a bit per way. See *Section 9.3.3: Cache locking sequence on page 114*.		
	When read		Returns current value		
	When written		Updates current value		
	Power-on reset		Undefined		
IW_LE	7	1	No	Instruction way load enable	RW
	Operation		if 0: prefetched instructions go to way selected by replacement algorithm. if 1: prefetched instructions go to way selected by IW_LOAD		
	When read		Returns current value		
	When written		Updates current value		
	Power-on reset		Undefined		
IW_LOCK	[8,11]	4	No	Instruction way lock	RW
	Operation		Bit w of IW_LOCK is the lock bit for way w in the instruction cache. If bit w is 0, way w of the instruction cache is unlocked. If bit w is 1, way w of the instruction cache is locked.		
	When read		Returns current value		
	When written		Updates current value		
	Power-on reset		Undefined		

Table 48: ICCR1

ICCR1			CFG	0x01600001	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
RES	[2,6], [12,63]	57	No	RESERVED	RES
	Operation		Software should always write 0 to these bits. Software should not interpret the value read from these bits.		
	When read		Reads as 0 (behavior of other implementations may vary)		
	When written		Writes ignored (behavior of other implementations may vary)		
	HARD reset		0 (behavior of other implementations may vary)		

Table 48: ICCR1

9.2.5 ICACHETAG

The ICACHETAG configuration registers are organized as a 2-dimensional array. This array is subscripted by way number and index number to give the tag information for a particular block in the instruction cache.

The tag information for each instruction cache block is held in 1 configuration register: ICACHETAG0 holds the effective address tag. This register contains a valid bit. This will be clear to indicate an invalid block, or set to indicate a valid block. When a block is invalid, all other fields have undefined values.

ICACHETAG0[w,i] where w is in the range [0,3] and i is in the range [0,255]			CFG	0x01000000 + (65536 * w) + (16 * i)	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
V	0	1	Yes	Valid bit	OTHER
	Operation		If 0: the cache block is invalid If 1: the cache block is valid		
	When read		Returns current value		
	When written		A write gives implementation-undefined behavior		
	Power-on reset		Undefined		

Table 49: Instruction cache tag

ICACHETAG0[w,i] where w is in the range [0,3] and i is in the range [0,255]			**CFG**	**0x01000000 + (65536 * w) + (16 * i)**	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
SH	1	1	Yes	Shared bit	OTHER
	Operation		If 0: the cache block is not shared If 1: the cache block is shared This field is a copy of the associated PTEH.SH		
	When read		Returns current value		
	When written		A write gives implementation-undefined behavior		
	Power-on reset		Undefined		
ASID	[2,9]	8	Yes	Address space identifier	OTHER
	Operation		Identifies the address space identifier for this cache block This field is a copy of the SR.ASID for the thread that caused this cache block to be allocated in the cache. For unshared pages this is the same as the associated PTEH.ASID. For shared pages the values of SR.ASID and PTEH.ASID can differ.		
	When read		Returns current value		
	When written		A write gives implementation-undefined behavior		
	Power-on reset		Undefined		
EADDR	[13,31]	19	Yes	Effective address tag	OTHER
	Operation		Identifies the effective address tag for this cache block Bits 0 to 4 of the effective address are the offset into this cache block. Bits 5 to 12 of the effective address are the index into the cache (i).		
	When read		Returns current value		
	When written		A write gives implementation-undefined behavior		
	Power-on reset		Undefined		

Table 49: Instruction cache tag

ICACHETAG0[w,i] where w is in the range [0,3] and i is in the range [0,255]			**CFG**	**0x01000000 + (65536 * w) + (16 * i)**	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
PRU	55	1	Yes	Access protection	OTHER
	Operation		If 0: the cache block is accessible to just privileged mode If 1: the cache block is accessible to user and privileged mode This field is a copy of the associated PTEL.PR.U		
	When read		Returns current value		
	When written		A write gives implementation-undefined behavior		
	Power-on reset		Undefined		
LRU	[58,63]	6	Yes	Replacement information	OTHER
	Operation		This field contains the replacement information for a set. All cache blocks in the same set are associated with the same LRU state. See *Table 42: SH-5 LRU field on page 84*.		
	When read		Returns current value		
	When written		A write gives implementation-undefined behavior		
	Power-on reset		Undefined		
RES	[10,12], [32,54], [56,57]	28	No	RESERVED	RES
	Operation		Software should not interpret the value read from these bits.		
	When read		Reads as 0 (behavior of other implementations may vary)		
	When written		A write gives implementation-undefined behavior		
	Power-on reset		0 (behavior of other implementations may vary)		

Table 49: Instruction cache tag

9.2.6 ICACHEDATA

The ICACHEDATA configuration registers are organized as a 3-dimensional array. This array is subscripted by way number, index number and register number to give the data information for a particular block in the instruction cache. The information in ICACHEDATA is only defined when the corresponding ICACHETAG is valid.

On SH-5, each instruction cache block contains 32 bytes of data. These 32 bytes are distributed over four 64-bit configuration registers. These registers are numbered r where r is in [0,3]. Let the physical address cached by this block be represented by P where P is a multiple of the cache block size. Let physical memory be represented by a byte-array called PMEM, and support slicing using the memory slicing notation from *Volume 2, Chapter 1: SHmedia specification*.

Register r contains cached instructions corresponding to PMEM[P+8r FOR 8].

The endianness of the instructions in each ICACHEDATA register is consistent with a 64-bit memory access. GETCFG performs a 64-bit read and will therefore return a value which is consistent with memory endianness.

ICACHEDATA[w,i,r] where w is in the range [0,3] and i is in the range [0,255] and r is in the range [0,3]			**CFG**	**0x01200000 + (65536 * w) + (16 * i) + r**	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
DATA	[0,63]	64	Yes	Instruction cache data	OTHER
	Operation		This register contains 64 bits of data corresponding to the r$^{th.}$ element of the 32 byte cache block		
	When read		Returns current value		
	When written		A write gives implementation-undefined behavior		
	Power-on reset		Undefined		

Table 50: Instruction cache data

9.2.7 OCCR

There are 2 operand cache control registers: OCCR0 and OCCR1.

OCCR0

Software should exercise care when writing to this register. If the operand cache is invalidated, then the state of any dirty cache blocks will be lost. Changing the value of either OCCR0.OCE or OCCR0.OWT can result in a change in cache behavior. It may be necessary to flush, purge or invalidate the operand cache to avoid paradoxes.

OCCR0			CFG	0x01E00000	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
OCE	0	1	No	Operand cache enable.	RW
	Operation		If 0: operand cache is disabled If 1: operand cache is enabled		
	When read		Returns current value		
	When written		Updates current value		
	Power-on reset		Undefined		
OCI	1	1	No	Operand cache invalidate	OTHER
	Operation		Write with 1 to invalidate the entire operand cache		
	When read		Returns 0		
	When written		Write of 0: no effect Write of 1: invalidate the entire operand cache Writes do not change the value of this field		
	Power-on reset		Undefined		

Table 51: OCCR0

OCCR0			CFG	0x01E00000	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
OWT	2	1	No	Operand cache write-through mode	RW
	Operation		If 0: write-through and write-back are distinguished by MMU If 1: write-back is implemented as write-through		
	When read		Returns current value		
	When written		Updates current value		
	Power-on reset		Undefined		
RES	[3,63]	61	No	RESERVED	RES
	Operation		Software should always write 0 to these bits. Software should not interpret the value read from these bits.		
	When read		Reads as 0 (behavior of other implementations may vary)		
	When written		Writes ignored (behavior of other implementations may vary)		
	Power-on reset		0 (behavior of other implementations may vary)		

Table 51: OCCR0

OCCR1

OCCR1			CFG	0x01E00001	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
OW_LOAD	[0,1]	2	No	Operand way load	RW
	Operation		Selects operand cache way for data prefetch when OW_LE is set. The value of OW_LOAD selects one of the 4 ways in the operand cache. It does not contain a bit per way. See *Section 9.3.3: Cache locking sequence on page 114*.		
	When read		Returns current value		
	When written		Updates current value		
	Power-on reset		Undefined		
OW_LE	7	1	No	Operand way load enable	RW
	Operation		If 0: prefetched data goes to way selected by replacement algorithm. If 1: prefetched data goes to way selected by OW_LOAD		
	When read		Returns current value		
	When written		Updates current value		
	Power-on reset		Undefined		
OW_LOCK	[8,11]	4	No	Operand Way Lock	RW
	Operation		Bit w of OW_LOCK is the lock bit for way w in the operand cache. If bit w is 0, way w of the operand cache is unlocked. If bit w is 1, way w of the operand cache is locked.		
	When read		Returns current value		
	When written		Updates current value		
	Power-on reset		Undefined		

Table 52: OCCR1

OCCR1			CFG	0x01E00001	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
RES	[2,6], [12,63]	57	No	RESERVED	RES
	Operation		Software should always write 0 to these bits. Software should not interpret the value read from these bits.		
	When read		Reads as 0 (behavior of other implementations may vary)		
	When written		Writes ignored (behavior of other implementations may vary)		
	HARD reset		0 (behavior of other implementations may vary)		

Table 52: OCCR1

9.2.8 OCACHETAG

The OCACHETAG configuration registers are organized as a 2-dimensional array. This array is subscripted by way number and index number to give the tag information for a particular block in the operand cache.

The tag information for each operand cache block is held in 2 configuration registers: OCACHETAG0 holds the effective address tag, and OCACHETAG1 holds the physical address tag. Each register contains a valid bit. These will either both be clear to indicate an invalid block, or both be set to indicate a valid block. When a block is invalid, all other fields have undefined values.

OCACHETAG0 is described in *Table 53*.

OCACHETAG0[w,i] where w is in the range [0,3] and i is in the range [0,255]			**CFG**	**0x01800000 + (65536 * w) + (16 * i)**	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
V	0	1	Yes	Valid bit	OTHER
	Operation		If 0: the cache block is invalid If 1: the cache block is valid		
	When read		Returns current value		
	When written		A write gives implementation-undefined behavior		
	Power-on reset		Undefined		
SH	1	1	Yes	Shared bit	OTHER
	Operation		If 0: the cache block is not shared If 1: the cache block is shared This field is a copy of the associated PTEH.SH		
	When read		Returns current value		
	When written		A write gives implementation-undefined behavior		
	Power-on reset		Undefined		

Table 53: Operand cache tag register 0

OCACHETAG0[w,i] where w is in the range [0,3] and i is in the range [0,255]			**CFG**	**0x01800000 + (65536 * w) + (16 * i)**	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
ASID	[2,9]	8	Yes	Address space identifier	OTHER
	Operation		Identifies the address space identifier for this cache block This field is a copy of the SR.ASID for the thread that caused this cache block to be allocated in the cache. For unshared pages this is the same as the associated PTEH.ASID. For shared pages the values of SR.ASID and PTEH.ASID can differ.		
	When read		Returns current value		
	When written		A write gives implementation-undefined behavior		
	Power-on reset		Undefined		
EADDR	[13,31]	19	Yes	Effective address tag	OTHER
	Operation		Identifies the effective address tag for this cache block Bits 0 to 4 of the effective address are the offset into this cache block. Bits 5 to 12 of the effective address are the index into the cache (i).		
	When read		Returns current value		
	When written		A write gives implementation-undefined behavior		
	Power-on reset		Undefined		
PRR	52	1	Yes	Read protection	OTHER
	Operation		If 0: the cache block is non-readable If 1: the cache block is readable This field is a copy of the associated PTEL.PR.R		
	When read		Returns current value		
	When written		A write gives implementation-undefined behavior		
	Power-on reset		Undefined		

Table 53: Operand cache tag register 0

OCACHETAG0[w,i] where w is in the range [0,3] and i is in the range [0,255]			CFG	0x01800000 + (65536 * w) + (16 * i)	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
PRW	54	1	Yes	Write protection	OTHER
	Operation		If 0: the cache block is non-writable If 1: the cache block is writable This field is a copy of the associated PTEL.PR.W		
	When read		Returns current value		
	When written		A write gives implementation-undefined behavior		
	Power-on reset		Undefined		
PRU	55	1	Yes	Access protection	OTHER
	Operation		If 0: the cache block is accessible to just privileged mode If 1: the cache block is accessible to user and privileged mode This field is a copy of the associated PTEL.PR.U		
	When read		Returns current value		
	When written		A write gives implementation-undefined behavior		
	Power-on reset		Undefined		
WT	56	1	Yes	Write-through/write-back bit	OTHER
	Operation		If 0: the cache block is in write-back mode If 1: the cache block is in write-through mode This field is a copy of the associated PTEL.CB0		
	When read		Returns current value		
	When written		A write gives implementation-undefined behavior		
	Power-on reset		Undefined		

Table 53: Operand cache tag register 0

OCACHETAG0[w,i] where w is in the range [0,3] and i is in the range [0,255]			**CFG**	**0x01800000 + (65536 * w) + (16 * i)**	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
U	57	1	Yes	Line dirty bit	OTHER
	Operation		If 0: the cache block is clean If 1: the cache block is dirty		
	When read		Returns current value		
	When written		A write gives implementation-undefined behavior		
	Power-on reset		Undefined		
LRU	[58,63]	6	Yes	Replacement information	OTHER
	Operation		This field contains the replacement information for a set. All cache blocks in the same set are associated with the same LRU state. See *Table 42: SH-5 LRU field on page 84*.		
	When read		Returns current value		
	When written		A write gives implementation-undefined behavior		
	Power-on reset		Undefined		
RES	[10,12], [32,51], 53	24	No	RESERVED	RES
	Operation		Software should not interpret the value read from these bits.		
	When read		Reads as 0 (behavior of other implementations may vary)		
	When written		A write gives implementation-undefined behavior		
	Power-on reset		0 (behavior of other implementations may vary)		

Table 53: Operand cache tag register 0

OCACHETAG1 is described in *Table 54*.

OCACHETAG1[w,i] where w is in the range [0,3] and i is in the range [0,255]			**CFG**	**0x01800000 + (65536 * w) + (16 * i)**	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
V	0	1	Yes	Valid bit	OTHER
	Operation		If 0: the cache block is invalid If 1: the cache block is valid		
	When read		Returns current value		
	When written		A write gives implementation-undefined behavior		
	Power-on reset		Undefined		
PADDR	[12,31]	20	Yes	Physical address tag	OTHER
	Operation		Identifies the physical address tag for this cache block All 20 bits of this tag are implemented by SH-5 Bits 0 to 4 of the physical address are the offset into this cache block. Bits 5 to 11 of the physical address are the lower 7 bits of the index into the cache (i). Note that bit 12 of the physical address is indicated by PADDR and not by i.		
	When read		Returns current value		
	When written		A write gives implementation-undefined behavior		
	Power-on reset		Undefined		
RES	[1,11], [32,63]	43	No	RESERVED	RES
	Operation		Software should not interpret the value read from these bits.		
	When read		Reads as 0 (behavior of other implementations may vary)		
	When written		A write gives implementation-undefined behavior		
	Power-on reset		0 (behavior of other implementations may vary)		

Table 54: Operand cache tag register 1

9.2.9 OCACHEDATA

The OCACHEDATA configuration registers are organized as a 3-dimensional array. This array is subscripted by way number, index number and register number to give the data information for a particular block in the operand cache. The information in OCACHEDATA is only defined when the corresponding OCACHETAG is valid.

On SH-5, each operand cache block contains 32 bytes of data. These 32 bytes are distributed over four 64-bit configuration registers. These registers are numbered r where r is in [0,3]. Let the physical address cached by this block be represented by P where P is a multiple of the cache block size. Let physical memory be represented by a byte-array called PMEM, and support slicing using the memory slicing notation from *Volume 2, Chapter 1: SHmedia specification*.

Register r contains cached data corresponding to PMEM[P+8r FOR 8].

The endianness of the data in each OCACHEDATA register is consistent with a 64-bit memory access. GETCFG performs a 64-bit read and will therefore return a value which is consistent with memory endianness.

OCACHEDATA[w,i,r] where w is in the range [0,3] and i is in the range [0,255] and r is in the range [0,3]			**CFG**	**0x01A00000 + (65536 * w) + (16 * i) + r**	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
DATA	[0,63]	64	Yes	Operand cache data	OTHER
	Operation		This register contains 64 bits of data corresponding to the $r^{th.}$ 8-byte element of the 32 byte cache block		
	When read		Returns current value		
	When written		A write gives implementation-undefined behavior		
	Power-on reset		Undefined		

Table 55: Operand cache data

9.3 Cache code sequences

This section describes recommended code sequences and algorithms for managing the cache.

9.3.1 Cache initialization sequence

The cache must be initialized before the MMU is enabled. The necessary steps are:

- Write to OCCR0 to specify the global behavior of the operand cache, and to invalidate the state of the operand cache before it is used.
- Write to OCCR1 to configure the operand cache locking information.
- Write to ICCR0 to specify the global behavior of the instruction cache, and to invalidate the state of the instruction cache before it is used.
- Write to ICCR1 to configure the instruction cache locking information.

If cache locking is to be used, note that the caches cannot be pre-loaded until the MMU is enabled since this is necessary for pre-fetches to modify cache state. Cache locking sequences are described in *Section 9.3.3: Cache locking sequence on page 114*.

9.3.2 Cache coherency sequences

There are 4 basic coherency operations:

- Invalidation of operand cache blocks. This is achieved using OCBI or OCCR0.OCI. Note that invalidation of operand cache blocks will result in dirty operand cache blocks being discarded. This should be done with care since it can result in loss of memory state.
- Write-back of operand cache blocks. This is achieved using OCBWB.
- Purge of operand cache blocks. This is achieved using OCBP.
- Invalidation of instruction cache blocks. This is achieved using ICBI or ICCR0.ICI.

These can be performed at 3 different granularities.

- Memory location: the appropriate instruction should be applied to the memory location. This will cohere a cache block sized memory block surrounding the supplied effective address. This can be achieved in user or privileged mode.
- Page of memory: for a small page of memory (such as the 4 kbyte page on SH-5), the appropriate cache coherency instruction should be iterated through the page with the effective address incrementing through the page in cache block size intervals. This can be achieved in user or privileged mode. For larger memory pages, it is more efficient to use privileged mode and to scan through the cache state as viewed though the cache configuration registers. Each cache block that contains address information corresponding to the target page should be cohered using the appropriate cache coherency instruction. The target effective address can be calculated from the address information in the cache block.
- All cache: this can be achieved in privileged mode only. Invalidation of the whole operand cache can be achieved using OCCR0.OCI, and of the whole instruction cache using ICCR0.ICI. For write-back or purge operations, a scan is necessary through the cache state as viewed through the cache configuration registers. Each valid cache block should be cohered using the appropriate cache coherency instruction. The target effective address can be calculated from the address information in the cache block.

When instruction cache invalidation is achieved through ICBI, invalidation is only guaranteed for cache blocks corresponding to the effective address used for the invalidation. This is because the instruction cache does not contain physical tags and does not support look-up by physical address.

In some cases, however, instruction invalidation may be required at the physical level, to ensure that the instruction is invalidated in all effective address spaces that map the physical address of the instruction. The recommended approach is to use privileged mode and to inspect the instruction cache state through the cache configuration registers. The instruction cache state should be indexed using the cache index field of the effective address being invalidated. This identifies a set in the cache; all cache blocks that can hold a copy of the instruction will be in this set providing that the constraints in *Volume 1, Chapter 17: Memory management* are being followed. Each of these cache blocks should be investigated in the cache configuration registers, and invalidated using an appropriately targeted ICBI if required. It may be quicker to invalidate each case unconditionally, rather than performing a software check to see if that invalidation is really necessary.

If it is necessary to invalidate many physical instructions, it may be easier or quicker to simply invalidate the entire instruction cache using ICCR0.ICI.

9.3.3 Cache locking sequence

The following sequence can be used to lock a single cache block in a particular way:

1 The following pre-conditions must hold:

- Privileged mode must be used since configuration register access is needed.
- The MMU must be enabled; SR.MMU should be set.
- Caching must be enabled. OCCR0.OCE should be set if locking into the operand cache; ICCR0.ICE should be set if locking into the instruction cache.
- The target effective address should be mapped by a translation that is cachable and contains appropriate permission. Read permission is required for prefetching into the data cache, and execute permission for the instruction cache.

2 The target way should be locked. The appropriate bit of OCCR1.OW_LOCK or ICCR1.IW_LOCK should be set. The way should be locked before following steps to ensure that other accesses do not interfere with this sequence.

3 The target effective address should not already be in the cache. If this is not the case, it can be removed from the cache using OCBP, OCBI or ICBI as appropriate. Since instruction fetching is performed independently of program execution, instruction invalidation should always be achieved explicitly using ICBI. This must be done after locking the way in step *2*.

4 The cache should be configured so that prefetches are performed into the target way. For operand cache locking, OCCR1.OW_LE should be set and OCCR1.OW_LOAD should be set to indicate the target way. For instruction cache locking, ICCR1.IW_LE should be set and ICCR1.IW_LOAD should be set to indicate the target way.

5 The cache block should be prefetched. Execute a data prefetch or an instruction prefetch, as appropriate, on the target effective address. The previous steps have arranged that this prefetch will miss the cache and cause the cache block in the specified way to be refilled from memory. Note that if there is no translation or if the prefetch has no permission, then the prefetch will be ignored. Software must arrange for appropriate translation as described in step *1*.

6 The load enable bit, OCCR1.OW_LE or ICCR1.IW_LE, can now be cleared to restart normal cache replacement.

A locked cache block can be removed from the cache through an appropriate purge or invalidation instruction. If the way is subsequently unlocked, then that way becomes a candidate for cache replacement.

9.4 Future cache implementations

Many properties of the cache are implementation-specific and can be varied in future implementations of the architecture. The cache implementation options are described in *Volume 1, Chapter 18: Caches*, and the SH-5 specific properties are described in this chapter. It is intended that future cache implementations will be based on the cache configuration register map and configuration register definitions used by the SH-5 implementation.

Note that the information in this section does not require future implementations to use these options, nor does it constrain future implementations to just these options.

9.4.1 Cache architecture parameters

The SH-5 cache configuration register map defined in *Section 6.3: Configuration registers on page 51* and in *Section 9.2: SH-5 cache configuration registers on page 93* supports the different cache organizations described by the architecture in *Volume 1, Chapter 18: Caches*:

- No cache: all accesses are performed on memory without caching.
- Unified cache: data and instruction accesses pass through a single cache.
- Split cache: data and instruction accesses are treated separately. The following implementation-specific options are available:
  - Only an operand cache is provided. Data accesses pass through the operand cache, while instruction accesses are performed on memory without caching. The terms ‘operand cache’ and ‘data cache’ are interchangeable.
  - Only an instruction cache is provided. Instruction accesses pass through the instruction cache. Data accesses are performed on memory without caching.
  - Both an operand cache and an instruction cache are provided. Data access pass through the operand cache, while instruction accesses pass independently and separately through the instruction cache.

The architecture defines that a cache is parameterized by:

- nbytes: the number of bytes in a cache block.
  nbytes can be any 2^i such that i is an integer in [3, 12].
- nways: the number of cache blocks in a set.
  nways is a power-of-2 and greater than 0.
- nsets: the number of sets of cache blocks in the cache.
  nsets is a power-of-2 and greater than 0.

9.4.2 Cache implementation parameters

The layout of the SH-5 cache configuration register map is:

- ICACHETAG: 0x01000000 + (65536*way) + (16*index) + reg
- ICACHEDATA: 0x01200000 + (65536*way) + (16*index) + reg
- ICCR: 0x01600000 + reg
- OCACHETAG: 0x01800000 + (65536*way) + (16*index) + reg
- OCACHEDATA: 0x01A00000 + (65536*way) + (16*index) + reg
- OCCR: 0x01E00000 + reg

The supported ‘index’ range (corresponding to the architectural parameter nsets) in the configuration register map is [0, 4095] using the partitioning described in *Section 6.3: Configuration registers on page 51*. Similarly, the supported ‘way’ range (corresponding to the architectural parameter nways) is [0, 31]. The supported ‘reg’ range allows up to 16 registers. Since each register can provide 8 bytes of data in the ICACHEDATA and OCACHEDATA registers, the maximum supported cache block size is 128 bytes (corresponding to the architectural parameter nbytes).

The cache configuration register map limits nbytes, nways and nsets to maximum values which are less then the architectural parameterization. However, there is significant room for future expansion, and these limits are unlikely to be a problem for future implementations.

Additionally, it is possible that future implementations can make extensions to the cache configuration registers:

- More configuration registers could be provided.
- Reserved fields within existing configuration registers be given defined semantics on future implementations. For ICACHETAG and OCACHETAG:
  - Extension of EADDR to support more effective address space.
  - Extension of PADDR to support more physical address space.
  - Extension of ASID to increase the number of supported address spaces.
  - Extensions to increase the number of supported cache behaviors.
  - Extensions to increase the number of supported protection attributes.

Software should be carefully written with consideration given to potential future changes in the cache implementation. Ideally, SH-5 software should be parameterized so that it can be readily updated to support future implementations.

Index

A

ALLOCO 10, 87-88
ANDC 76-77

B

BEQ 9
BEQI 9
BGE 9
BGEU 9
BGT 9
BGTU 9
BNE 9
BNEI 9

D

DBRMODE 56
DBRVEC 56
DEFINED 14

E

EPN 60, 77

F

Field
  ASID 59-60, 75, 99, 107
  CB 58, 108
  DATA 101, 111
  EADDR 99, 107
  ICE 96, 112-114
  ICI 96, 113
  IW_LE 87, 97, 114
  IW_LOAD 87, 97, 114
  IW_LOCK 97, 114
  LRU 83-84, 100, 109
  OCE 102, 112-114
  OCI 102
  OW_LE 87, 104, 114
  OW_LOAD 87, 104, 114
  OW_LOCK 104, 114
  OWT 102-103
  PR 59, 100, 107-108
  PRR 107
  PRU 100, 108
  PRW 108
  RES 14, 96, 98, 100, 103, 105, 109-110
  SZ 58
  U 12-13, 100, 108-109
  V . 12-13, 58, 62, 68, 78, 98, 106, 110
  WT 108
FIPR 9
FIPR.S 9
FOR 101, 111
FPU 3, 12-14, 56
FTRV 9, 17

FTRV.S 9

G

GETCFG 10, 15, 93-94, 101, 111
GETCON 10, 14, 75, 77
GETTR 77

H

HARD 96, 98, 105

I

ICBI 10, 88-89, 112-114
ICCR0.ICE 112-114
ITLBMISS 88-89

L

LD.B 9
LD.L 9
LD.Q 9
LD.UB 9
LD.UW 9
LD.W 9
LDX.B 9
LDX.L 9
LDX.Q 9
LDX.UB 9
LDX.UW 9
LDX.W 9

M

MMU 3, 57, 60-61, 74-77, 81, 87, 93-94, 112, ... 114
MMUDR 61, 67-68, 71
MMUIR 61-62, 64
MOVACA.L 17
MOVCA.L 3, 17
MOVI 75, 77

O

OCACHETAG0 106
OCACHETAG1 . . . . 106, 110
OCBI 3, 10, 17, 89, 112, 114
OCBP 10, 17, 89, 112, 114
OCBWB 10, 17, 89, 112
OCCR0 . . . . 102, 112-114
OCCR0.OCE 102, 112-114
OCCR0.OWT 102
OCCR1 . . . . 87, 102, 104, 112, 114
OFFSET_PTEH 78
OFFSET_PTEL 78
OR 75-77

P

PADDR 110
PC 2, 7-8, 74-75, 77
PPN 59, 77
PREF 17
PREFI 10, 89
PSPC 7, 55
PSPC.ISA 55
PTA 9
PTABS 9
PTB 9, 77
PTE . . . 57-58, 60-62, 68, 75, 77-79, 81
PTEH 58-62, 67-68, 78-79, 99, 106-107
PTEH.EPN 60
PTEH.V 58, 62, 68, 78
PTEL 58-59, 61-62, 64, 67-68, 71, 78-79, ... 100, 107-108
PTEL.PPN 59
PTREL 9
PUTCFG 10, 15, 62, 68, 78, 93-94
PUTCON 10, 14, 74-75, 77

R

Register
  Field Type
    EXPANSION 20
    OTHER 20
    READ-ONLY 20
    READ-WRITE 20
    RESERVED 20
  ICACHEDATA 93-95, 101, 116
  ICACHETAG 93-94, 98, 101, 116
  ICACHETAG0 98
  ICCR 93, 95-96, 116
  ICCR0 . . . . 96, 112-114
  ICCR1 . . . . 87, 96-97, 112, 114
  OCACHEDATA 93-95, 111, 116
  OCACHETAG . . 93-95, 106, 111, 116
  OCCR 93, 95, 102, 116
  R 5, 75-78, 89
  SR 74-77, 114
  SR.MMU .114
  SSR 74-75, 77
  TEA 7
  TR .5, 7, 75, 77
  VBR 7
RESERVED . 14, 96, 98, 100, 103, 105,
    109- 110
RESVEC 7
RTE 10, 55, 74-78

S

SHLLI 75, 77
SLEEP 10
SPC 7, 10, 55, 74-75, 77
SPC.ADDR 55
SPC.ISA 55
SYNCI 75, 78, 93
SYNCO 75-77, 93-94

U

UNDEFINED 14-15