


# **XGATE**

## **Block Guide**

### **01.04**

**Original Release Date: 18 Jun 2003**  
**Revised: 31 Mar 2004**

**8/16 Bit Division, TSPG**  
**Motorola, Inc.**

Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters which may be provided in Motorola data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part. Motorola and  are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

©Motorola, Inc., 2001

## Revision History

Version Number	Revision Date	Effective Date	Author	Description of Changes
00.00	18 Jun 2003	18 Jun 2003		Initial release
01.00	28 Jan 2004	28 Jan 2004		First official release
01.02	19 Feb 2004	19 Feb 2004		Minor corrections
01.03	27 Feb 2004	27 Feb 2004		Removed reserved instructions from <b>Table 4-2</b>
01.04	31 Mar 2004	31 Mar 2004		Minor corrections

# Table of Contents

## Section 1 Introduction

1.1	Overview . . . . .	11
1.2	Features . . . . .	12
1.3	Modes of Operation . . . . .	12

## Section 2 External Signal Description

## Section 3 Memory Map/Register Definition

3.1	Register Descriptions . . . . .	13
3.1.1	XGATE Module Control Register (XGMCTL) . . . . .	14
3.1.2	XGATE Channel ID Register (XGCHID) . . . . .	16
3.1.3	XGATE Vector Base Address Register (XGVBR) . . . . .	16
3.1.4	XGATE Channel Interrupt Flag Vector (XGIF) . . . . .	17
3.1.5	XGATE Software Trigger Register (XGSWT) . . . . .	19
3.1.6	XGATE Semaphore Register (XGSEM) . . . . .	20
3.1.7	XGATE Condition Code Register (XGCCR) . . . . .	21
3.1.8	XGATE Program Counter Register (XGPC) . . . . .	22
3.1.9	XGATE Register 1 (XGR1) . . . . .	22
3.1.10	XGATE Register 2 (XGR2) . . . . .	22
3.1.11	XGATE Register 3 (XGR3) . . . . .	23
3.1.12	XGATE Register 4 (XGR4) . . . . .	23
3.1.13	XGATE Register 5 (XGR5) . . . . .	24
3.1.14	XGATE Register 6 (XGR6) . . . . .	24
3.1.15	XGATE Register 7 (XGR7) . . . . .	25

## Section 4 Functional Description

4.1	XGATE RISC Core . . . . .	26
4.1.1	Programmer's Model . . . . .	26
4.1.2	Memory Map . . . . .	27
4.2	Semaphores . . . . .	28
4.3	Software Error Detection . . . . .	30
4.4	Interrupts . . . . .	31
4.4.1	Incoming Interrupt Requests . . . . .	31

4.4.2	Outgoing Interrupt Requests . . . . .	31
4.5	Debug Mode. . . . .	31
4.5.1	Debug Features . . . . .	31
4.5.2	Entering Debug Mode . . . . .	32
4.5.3	Leaving Debug Mode. . . . .	33
4.6	Security. . . . .	33
4.7	Instruction Set. . . . .	33
4.7.1	Addressing Modes . . . . .	33
4.7.2	Instruction Summary and Usage . . . . .	37
4.7.3	Cycle Notation . . . . .	39
4.7.4	Thread Execution . . . . .	39
4.7.5	Instruction Glossary . . . . .	40
4.7.6	Instruction Coding . . . . .	113

## Section 5 Initialization/Application Information

5.1	Initialization. . . . .	115
5.2	Code Example (transmit "Hello World!" on SCI). . . . .	115

# List of Figures

Figure 1-1	XGATE Block Diagram . . . . .	11
Figure 3-1	XGATE Module Control Register (XGMCTL) . . . . .	14
Figure 3-2	XGATE Channel ID Register (XGCHID) . . . . .	16
Figure 3-3	XGATE Vector Base Address Register (XGVBR) . . . . .	16
Figure 3-4	XGATE Channel Interrupt Flag Vector (XGIF) . . . . .	18
Figure 3-5	XGATE Software Trigger Register (XGSWT) . . . . .	19
Figure 3-6	XGATE Semaphore Register (XGSEM) . . . . .	20
Figure 3-7	XGATE Condition Code Register (XGCCR) . . . . .	21
Figure 3-8	XGATE Program Counter Register (XGPC) . . . . .	22
Figure 3-9	XGATE Register 1 (XGR1) . . . . .	22
Figure 3-10	XGATE Register 2 (XGR2) . . . . .	23
Figure 3-11	XGATE Register 3 (XGR3) . . . . .	23
Figure 3-12	XGATE Register 4 (XGR4) . . . . .	24
Figure 3-13	XGATE Register 5 (XGR5) . . . . .	24
Figure 3-14	XGATE Register 6 (XGR6) . . . . .	25
Figure 3-15	XGATE Register 7 (XGR7) . . . . .	25
Figure 4-1	Programmer's Model . . . . .	26
Figure 4-2	XGATE Memory Map. . . . .	27
Figure 4-3	XGATE Vector Block . . . . .	28
Figure 4-4	Semaphore State Transitions . . . . .	29
Figure 4-5	Algorithm for Locking and Releasing Semaphores . . . . .	30
Figure 4-6	Bit Field Addressing. . . . .	38



# List of Tables

Table 3-1    Module Memory Map . . . . . 13

Table 4-1    Access Detail Notation. . . . . 39

Table 4-2    Instruction Set Summary . . . . . 113





# Preface

## Terminology

### XGATE Request:

A service request from a peripheral module which is directed to the XGATE by the S12X\_INT module (see **Figure 1-1**).

### XGATE Channel

The resources in the XGATE module (i.e. Channel ID number, Service Request Vector, Interrupt Flag) that are associated with a particular XGATE Request.

### XGATE Channel ID

Each XGATE request has a 7-bit identifier. In S12X designs valid Channel IDs range from \$78 to \$09.

### XGATE Channel Interrupt

An S12X\_CPU interrupt that is triggered by a code sequence running on the XGATE module.

### XGATE Software Channel

Special XGATE channel that is not associated with any peripheral service request. A Software Channel is triggered by its Software Trigger Bit which is implemented in the XGATE module.

### XGATE Semaphore

A set of hardware flip-flops that can be exclusively set by either the S12X\_CPU or the XGATE. (see **4.2**)

### XGATE Thread

A code sequence that is executed by the XGATE's RISC core after receiving an XGATE request.

### XGATE Debug Mode

A special mode in which the XGATE's RISC core is halted for debug purposes. This mode enables the XGATE's debug features (see **4.5**).

### XGATE Software Error

The XGATE is able to detect a number of error conditions caused by erratic software (see **4.3**). These error conditions will cause the XGATE to seize program execution and flag an Interrupt to the S12X\_CPU.

### Word

A 16-bit entity.

### Byte

An 8-bit entity.





- Debug features (**Section 4.5**)
- Security (**Section 4.6**)
- Instruction set (**Section 4.7**)

## 1.2 Features

The XGATE module includes these features:

- Data movement between RAM and peripheral modules.
- Data manipulation through built in RISC core
- Provides up to 112 XGATE channels
  - - 104 hardware triggered channels
  - - 8 software triggered channels
- Hardware semaphores which are shared between the S12X\_CPU and the XGATE module
- Able to trigger S12X\_CPU interrupts upon completion of an XGATE transfer

## 1.3 Modes of Operation

There are four run modes on S12X devices.

- Run Mode, Wait Mode, Stop Mode

The XGATE is able to operate in all of these three system modes. Clock activity will be automatically stopped when the XGATE module is idle.

- Freeze Mode (BDM active)

In freeze mode all clocks of the XGATE module may be stopped, depending on the module configuration (see **3.1.1**).

## Section 2 External Signal Description

The XGATE module has no external pins.

## Section 3 Memory Map/Register Definition

This section provides a detailed description of address space and registers used by the XGATE module.

The memory map for the XGATE module is given below in **Table 3-1**. The address listed for each register is the sum of a base address and an address offset. The base address is defined at the SoC level and the address offset is defined at the module level. Reserved registers read zero. Write accesses to the reserved registers have no effect.

**Table 3-1 Module Memory Map**

Address	Use	Access
+\$00	XGATE Module Control Register (XGMCTL)	Read/Write <sup>1</sup>
+\$01	Reserved	None
+\$02	XGATE Channel ID Register (XGCHID)	Read
+\$03	Reserved	None
+\$04, +\$05 +\$06, +\$07	XGATE Vector Base Address (XGVBR)	Read/Write
+\$08, +\$09, +\$0A, +\$0B, +\$0C, +\$0D, +\$0E, +\$0F, +\$10, +\$11, +\$12, +\$13, +\$14, +\$15, +\$16, +\$17	XGATE Interrupt Flag Vector (XGIF)	Read/Write <sup>1</sup>
+\$18, +\$19	XGATE Software Trigger Register (XGSWT)	Read/Write
+\$1A, +\$1B	XGATE Semaphore Register (XGSEM)	Read/Write <sup>2</sup>
+\$1C	Reserved	None
+\$1D	XGATE Condition Code Register (XGCCR)	Read/Write <sup>1,3</sup>
+\$1E, +\$1F	XGATE Program Counter (XGPC)	Read/Write <sup>3</sup>
+\$20, +\$21	Reserved	None
+\$22, +\$23	XGATE Register 1 (XGR1)	Read/Write <sup>3</sup>
+\$24, +\$25	XGATE Register 2 (XGR2)	Read/Write <sup>3</sup>
+\$26, +\$27	XGATE Register 3 (XGR3)	Read/Write <sup>3</sup>
+\$28, +\$29	XGATE Register 4 (XGR4)	Read/Write <sup>3</sup>
+\$2A, +\$2B	XGATE Register 5 (XGR5)	Read/Write <sup>3</sup>
+\$2C, +\$2D	XGATE Register 6 (XGR6)	Read/Write <sup>3</sup>
+\$2E, +\$2F	XGATE Register 7 (XGR7)	Read/Write <sup>3</sup>
+\$30, +\$31, +\$32, +\$33, +\$34, +\$35, +\$36, +\$37, +\$38, +\$39	Reserved	None

**NOTES:**

1. Certain bits are not writable.
2. see **4.2**
3. Write only if in Debug Mode.

## 3.1 Register Descriptions

This section consists of register descriptions in address order. Each description includes a standard register diagram with an associated figure number. Details of register bit and field function follow the register diagrams, in bit order.

### 3.1.1 XGATE Module Control Register (XGMCTL)

All module level switches and flags are located in the Module Control Register **Figure 3-1**.



**Figure 3-1 XGATE Module Control Register (XGMCTL)**

Read: anytime

Write: anytime

#### XGE - XGATE Module Enable

This bit enables the XGATE module. If the XGATE module is disabled, pending XGATE requests will be ignored. The thread that is executed by the RISC core while the XGE bit is cleared will continue to run.

Read:

1 = XGATE module is enabled

0 = XGATE module is disabled

Write:

1 = Enable XGATE module

0 = Disable XGATE module

#### XGFRZ - Stop XGATE in Freeze Mode

The XGFRZ bit controls the XGATE operation in Freeze Mode (BDM active).

Read:

1 = RISC core stops in Freeze Mode (BDM active)

0 = RISC core operates normally in Freeze (BDM active)

Write:

1 = Stop RISC core in Freeze Mode (BDM active)

0 = Don't stop RISC core in Freeze Mode (BDM active)

#### XGDBGM - XGATE Debug Mode Mask

This bit controls the write access to the XGDBG bit. The XGDBG bit can only be set or cleared if a "1" is written to the XGDBGM bit in the same register access.

Read:

This bit will always read "0".

Write:

1 = Enable write access to the XGDBG in the same bus cycle

0 = Disable write access to the XGDBG in the same bus cycle

### XGDBG - XGATE Debug Mode

This bit indicates that the XGATE is in Debug Mode (see **4.5**). Debug Mode can be entered by Software Breakpoints (BRK instruction), Tagged or Forced Breakpoints (see **S12X\_DBG Block User Guide**), or by writing a "1" to this bit.

Read:

- 1 = RISC core is in Debug Mode
- 0 = RISC core is not in Debug Mode

Write:

- 1 = Enter Debug Mode
- 0 = Leave Debug Mode

**NOTE:** *Freeze Mode and Software Error Interrupts have no effect on the XGDBG bit.*

### XGSS - XGATE Single Step

This bit forces the execution of a single instruction if the XGATE is in DEBUG Mode and no software error has occurred (XGSWEIF cleared).

Read:

This bit will always read "0".

Write

- 1 = Execute a single RISC instruction
- 0 = No effect

### XGSWEIF - XGATE Software Error Interrupt Flag

This bit signals a pending Software Error Interrupt. It is set if the RISC core detects an error condition (see **4.3**). The RISC core is stopped while this bit is set. Clearing this bit will terminate the current thread and cause the XGATE to become idle.

Read:

- 1 = Software Error Interrupt is pending if XGIE is set
- 0 = Software Error Interrupt is not pending

Write:

- 1 = Clears the XGSWEIF bit
- 0 = No effect

### XGIE - XGATE Interrupt Enable

This bit acts as a global interrupt enable for the XGATE module

Read:

- 1 = All XGATE interrupts enabled
- 0 = All XGATE interrupts disabled

Write:

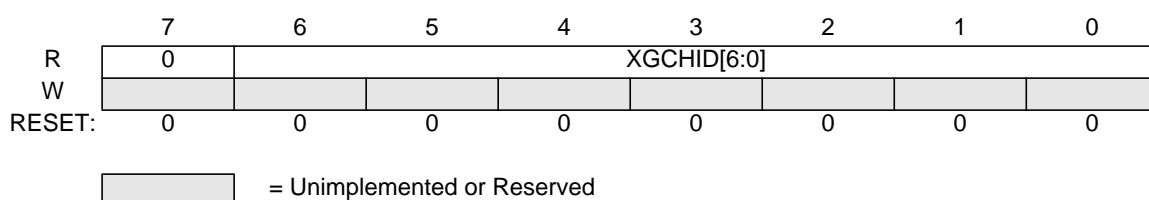
- 1 = Enable all XGATE interrupts

0 = Disable all XGATE interrupts

### 3.1.2 XGATE Channel ID Register (XGCHID)

The XGATE Channel ID register (**Figure 3-2**) shows the identifier of the XGATE channel that is currently active. This register will read “\$00” if the XGATE module is idle. In Debug mode this register can be used to start and terminate threads (see **4.5.1**).

**XGATE+\$02**



**Figure 3-2 XGATE Channel ID Register (XGCHID)**

Read: anytime

Write: in Debug Mode

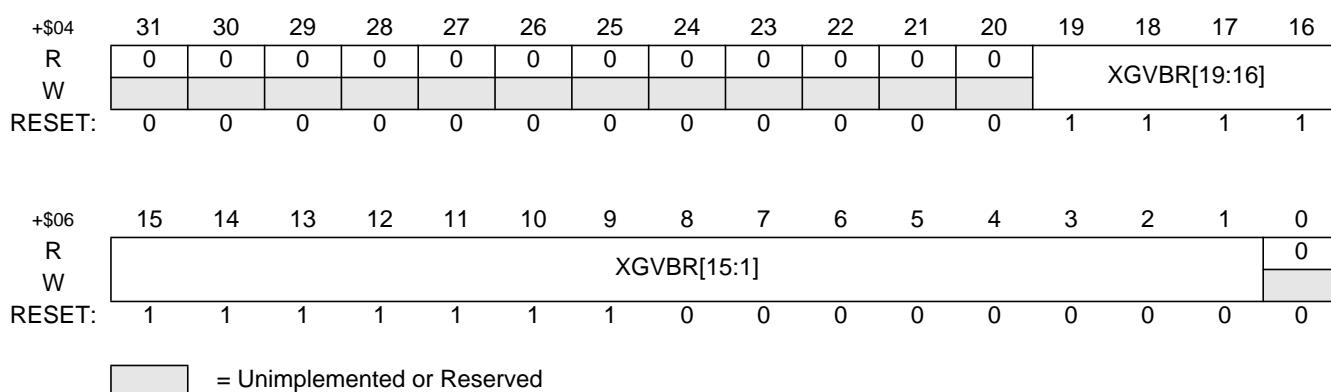
XGCHID[6:0] - Request Identifier

ID of the currently active channel

### 3.1.3 XGATE Vector Base Address Register (XGVBR)

The Vector Base Address Register (**Figure 3-3**) determines the location of the XGATE vector block.

**XGATE+\$04**



**Figure 3-3 XGATE Vector Base Address Register (XGVBR)**

Read: anytime

Write: only if the module is disabled (XGE=0) and idle (XGCHID=\$00))



## XGVBR - Vector Base Address

The XGVBR register holds the start address of the vector block in the global memory map. The upper address bits are read-only depending on the size of the internal RAM (see **SoC Guide**).

512K ≥ RAM size > 256K:

XGVBR[19] is read only.

256K ≥ RAM size > 128K:

XGVBR[19:18] is read only.

128K ≥ RAM size > 64K:

XGVBR[19:17] is read only.

64K ≥ RAM size > 32K:

XGVBR[19:16] is read only.

32K ≥ RAM size > 16K:

XGVBR[19:15] is read only.

16K ≥ RAM size > 8K:

XGVBR[19:14] is read only.

8K ≥ RAM size > 4K:

XGVBR[19:13] is read only.

4K ≥ RAM size > 2K:

XGVBR[19:12] is read only.

**NOTE:** *Suggested Mnemonics for accessing the Vector Base Register on a word basis are:*

**XGVBR\_HI** (XGVBR[31:16]),

**XGVBR\_LO** (XGVBR[15:0])

### 3.1.4 XGATE Channel Interrupt Flag Vector (XGIF)

The Interrupt Flag Vector (**Figure 3-4**) provides access to the Interrupt Flags bits of each channel. Each flag may be cleared by writing a "1" to its bit location.

**XGATE+\$08**

	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112	
+\$08																	
R	0	0	0	0	0	0	0		XGIF_78	XGF_77	XGIF_76	XGIF_75	XGIF_74	XGIF_73	XGIF_72	XGIF_71	XGIF_70
W																	
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


+\$0C	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
R																
W	XGIF_5F	XGIF_5E	XGIF_5D	XGIF_5C	XGIF_5B	XGIF_5A	XGIF_59	XGIF_58	XGF_57	XGIF_56	XGIF_55	XGIF_54	XGIF_53	XGIF_52	XGIF_51	XGIF_50
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

+\$0E	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
R																
W	XGIF_4F	XGIF_4E	XGIF_4D	XGIF_4C	XGIF_4B	XGIF_4A	XGIF_49	XGIF_48	XGF_47	XGIF_46	XGIF_45	XGIF_44	XGIF_43	XGIF_42	XGIF_41	XGIF_40
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0




+\$16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R		XGIF_0F	XGIF_0E	XGIF_0D	XGIF_0C	XGIF_0B	XGIF_0A	XGIF_09								
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

**Figure 3-4 XGATE Channel Interrupt Flag Vector (XGIF)**

Read: anytime

Write: anytime

XGIF\_F0...XG\_12 - Channel Interrupt Flags

These bits signal pending Channel Interrupts. They can only be set by the RISC core. Each flag can be cleared by writing a "1" to its bit location. Unimplemented interrupt flags will always read "0". Refer to Section 5 of the **SoC Guide** for a list of implemented Interrupts.

Read:

- 1 = Channel Interrupt is pending if XGIE is set
- 0 = Channel Interrupt is not pending

Write:

- 1 = Clears the Interrupt Flag
- 0 = No effect

**NOTE:** *Suggested Mnemonics for accessing the Interrupt Flag Vector on a word basis are:*

**XGIF\_7F\_70** (XGIF[127:112]),

**XGIF\_6F\_60** (XGIF[111:96]),

**XGIF\_5F\_50** (XGIF[95:80]),

**XGIF\_4F\_40** (XGIF[79:64]),

**XGIF\_3F\_30** (XGIF[63:48]),

**XGIF\_2F\_20** (XGIF[47:32]),

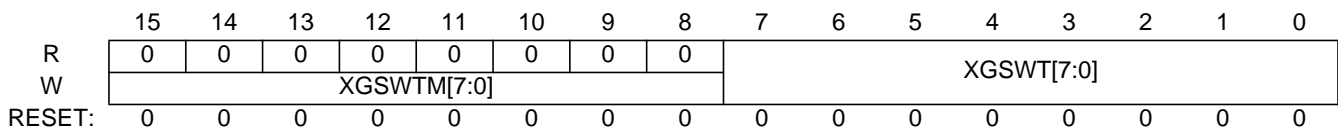
**XGIF\_1F\_10** (XGIF[31:16]),

**XGIF\_0F\_00** (XGIF[15:0])

### 3.1.5 XGATE Software Trigger Register (XGSWT)

The eight Software Triggers of the XGATE module can be set and cleared through the XGATE Software Trigger Register (**Figure 3-5**). The upper byte of this register, the Software Trigger Mask, controls the write access to the lower byte, the Software Trigger bits. These bits can be set or cleared if a "1" is written to the associated mask in the same bus cycle.

**XGATE+\$18**



 = Unimplemented or Reserved

**Figure 3-5 XGATE Software Trigger Register (XGSWT)**

Read: anytime

Write: anytime

XGSWTM[7:0] - Software Trigger Mask

These bits control the write access to the XGSWT bits. Each XGSWT bit can only be written if a "1" is written to the corresponding XGSWTM bit in the same access.

Read:

These bits will always read "0".

Write:

- 1 = Enable write access to the corresponding XGSWT bit in the same bus cycle
- 0 = Disable write access to the XGSWT in the same bus cycle

### XGSWT[7:0] - Software Trigger Bits

These bits act as interrupt flags that are able to trigger XGATE software channels. They can only be set and cleared by software.

Read:

- 1 = Software trigger pending if the XGIE bit is set
- 0 = No software trigger pending

Write:

- 1 = Set Software Trigger
- 0 = Clear Software Trigger

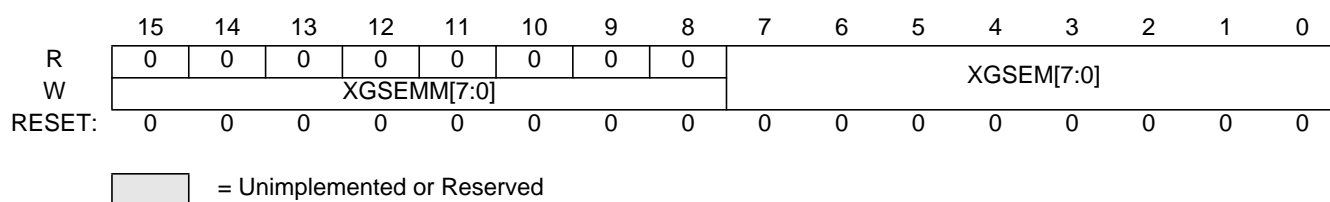
**NOTE:** The XGATE Channel IDs that are associated with the eight Software Triggers are determined on chip integration level. (see Section 5 of the **Soc Guide**)

**NOTE:** XGATE Software Triggers work like any peripheral interrupt. They can be used as XGATE Requests as well as S12X\_CPU Interrupts. The target of the software trigger must be selected in the S12X\_INT module.

## 3.1.6 XGATE Semaphore Register (XGSEM)

The XGATE provides a set of eight hardware semaphores that can be shared between the S12X\_CPU and the XGATE RISC Core. Each semaphore can either be unlocked, locked by the S12X\_CPU or locked by the RISC core. The RISC core is able to lock and unlock a semaphore through its SSEM and CSEM instructions. The S12X\_CPU has access to the semaphores through the XGATE Semaphore Register (**Figure 3-6**). Refer to section 4.2 for details.

### XGATE+\$1A



**Figure 3-6 XGATE Semaphore Register (XGSEM)**

Read: anytime

Write: anytime (see 4.2)

### XGSEMM[7:0] - Semaphore Mask

These bits control the write access to the XGSEM bits.

Read:

These bits will always read "0".

Write:

- 1 = Enable write access to the XGSEM in the same bus cycle
- 0 = Disable write access to the XGSEM in the same bus cycle

### XGSEM[7:0] - Semaphore Bits

These bits indicate whether a semaphore is locked by the S12X\_CPU. A semaphore can be attempted to be set by writing a "1" to the XGSEM bit and to the corresponding XGSEMM bit in the same write access. Only unlocked semaphores can be set. A semaphore can be cleared by writing a "0" to the XGSEM bit and a "1" to the corresponding SGSEMM bit in the same write access.

Read:

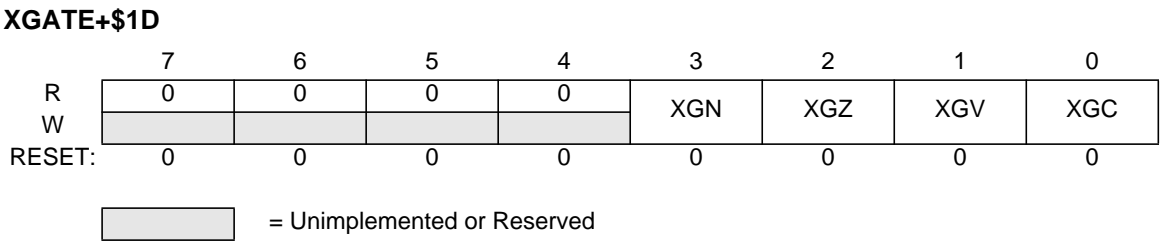
- 1 = Semaphore is locked by the S12X\_CPU
- 0 = Semaphore is unlocked or locked by the RISC core

Write:

- 1 = Attempt to lock semaphore by the S12X\_CPU
- 0 = No effect

## 3.1.7 XGATE Condition Code Register (XGCCR)

The XGCCR register (**Figure 3-7**) provides access to the RISC core's Condition Code Register.



**Figure 3-7 XGATE Condition Code Register (XGCCR)**

Read: in Debug Mode if unsecured  
 Write: in Debug Mode if unsecured

XGN - Sign Flag

The RISC core's Sign flag

XGZ - Zero Flag

The RISC core's Zero flag

XGV - Overflow Flag

The RISC core's Overflow flag

XGC - Carry Flag

The RISC core's Carry flag

### 3.1.8 XGATE Program Counter Register (XGPC)

The XGPC register (**Figure 3-8**) provides access to the RISC core's Program Counter.

**XGATE+\$1E**



**Figure 3-8 XGATE Program Counter Register (XGPC)**

Read: in Debug Mode if unsecured

Write: in Debug Mode if unsecured

XGPC - Program Counter

The RISC core's Program Counter

### 3.1.9 XGATE Register 1 (XGR1)

The XGR1 register (**Figure 3-9**) provides access to the RISC core's Register 1.

**XGATE+\$22**



**Figure 3-9 XGATE Register 1 (XGR1)**

Read: in Debug Mode if unsecured

Write: in Debug Mode if unsecured

XGR1 - R1

The RISC core's Register 1

### 3.1.10 XGATE Register 2 (XGR2)

The XGR2 register (**Figure 3-10**) provides access to the RISC core's Register 2.

**XGATE+\$24****Figure 3-10 XGATE Register 2 (XGR2)**

Read: in Debug Mode if unsecured

Write: in Debug Mode if unsecured

XGR2 - R2

The RISC core's Register 2

**3.1.11 XGATE Register 3 (XGR3)**

The XGR3 register (**Figure 3-11**) provides access to the RISC core's Register 3.

**XGATE+\$26****Figure 3-11 XGATE Register 3 (XGR3)**

Read: in Debug Mode if unsecured

Write: in Debug Mode if unsecured

XGR3 - R3

The RISC core's Register 3

**3.1.12 XGATE Register 4 (XGR4)**

The XGR4 register (**Figure 3-12**) provides access to the RISC core's Register 4.

**XGATE+\$28****Figure 3-12 XGATE Register 4 (XGR4)**

Read: in Debug Mode if unsecured

Write: in Debug Mode if unsecured

XGR4 - R4

The RISC core's Register 4

**3.1.13 XGATE Register 5 (XGR5)**The XGR5 register (**Figure 3-13**) provides access to the RISC core's Register 5.**XGATE+\$2A****Figure 3-13 XGATE Register 5 (XGR5)**

Read: in Debug Mode if unsecured

Write: in Debug Mode if unsecured

XGR5 - R5

The RISC core's Register 5

**3.1.14 XGATE Register 6 (XGR6)**The XGR6 register (**Figure 3-14**) provides access to the RISC core's Register 6.



**XGATE+\$2C****Figure 3-14 XGATE Register 6 (XGR6)**

Read: in Debug Mode if unsecured

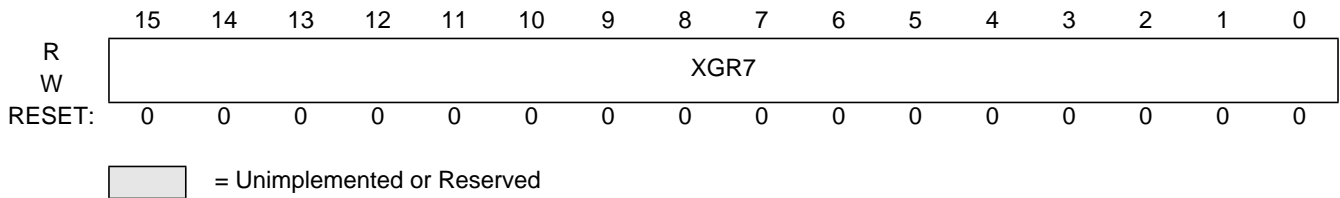
Write: in Debug Mode if unsecured

XGR6 - R6

The RISC core's Register 6

**3.1.15 XGATE Register 7 (XGR7)**

The XGR7 register (**Figure 3-15**) provides access to the RISC core's Register 7.

**XGATE+\$2E****Figure 3-15 XGATE Register 7 (XGR7)**

Read: in Debug Mode if unsecured

Write: in Debug Mode if unsecured

XGR7 - R7

The RISC core's Register 7

**Section 4 Functional Description**

The core of the XGATE module is a RISC processor which is able to access the MCU's internal RAM and peripherals (see **Figure 1-1**). The RISC processor always remains in an idle state until it is triggered by an XGATE request. Then it executes a code sequence that is associated with the request and optionally triggers an interrupt to the S12X\_CPU upon completion. Code sequences are not interruptible. A new XGATE request can only be serviced when the previous sequence is finished and the RISC core becomes idle.

The XGATE module also provides a set of hardware semaphores which are necessary to ensure data consistency whenever RAM locations or peripherals are shared with the S12X\_CPU.

The following sections describe the components of the XGATE module in further detail.

## 4.1 XGATE RISC Core

The RISC core is a 16-bit processor with an instruction set that is well suited for data transfers, bit manipulations, and simple arithmetic operations (see **4.7**).

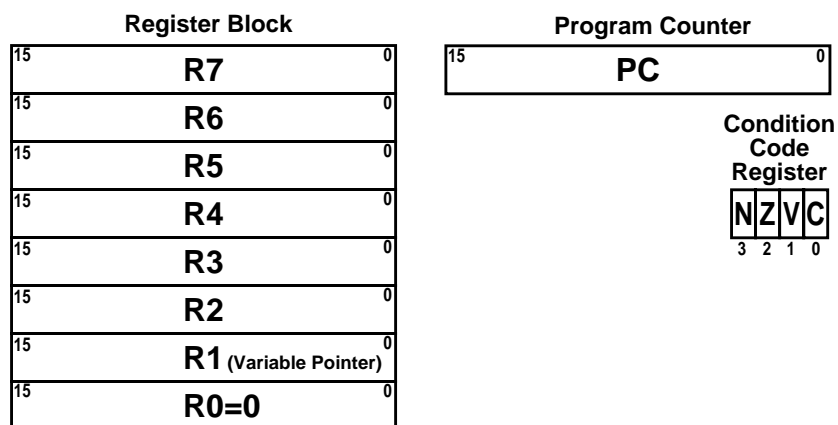
It is able to access the MCU's RAM and peripherals without blocking these resources from the S12X\_CPU. Whenever the S12X\_CPU and the RISC core access the same resource, the RISC core will be stalled until the resource becomes available again.

The XGATE offers a high access rate to the MCU's internal RAM. Depending on the bus load, the RISC core can perform up to two RAM accesses per S12X\_CPU bus cycle. A minimum throughput of one RAM access per S12X\_CPU bus cycle is guaranteed.

Bus accesses to peripheral registers are slower. A transfer rate of one register access per S12X\_CPU bus cycle can not be exceeded.

The XGATE module is intended to execute short interrupt service routines that are triggered by peripheral modules or by software.

### 4.1.1 Programmer's Model



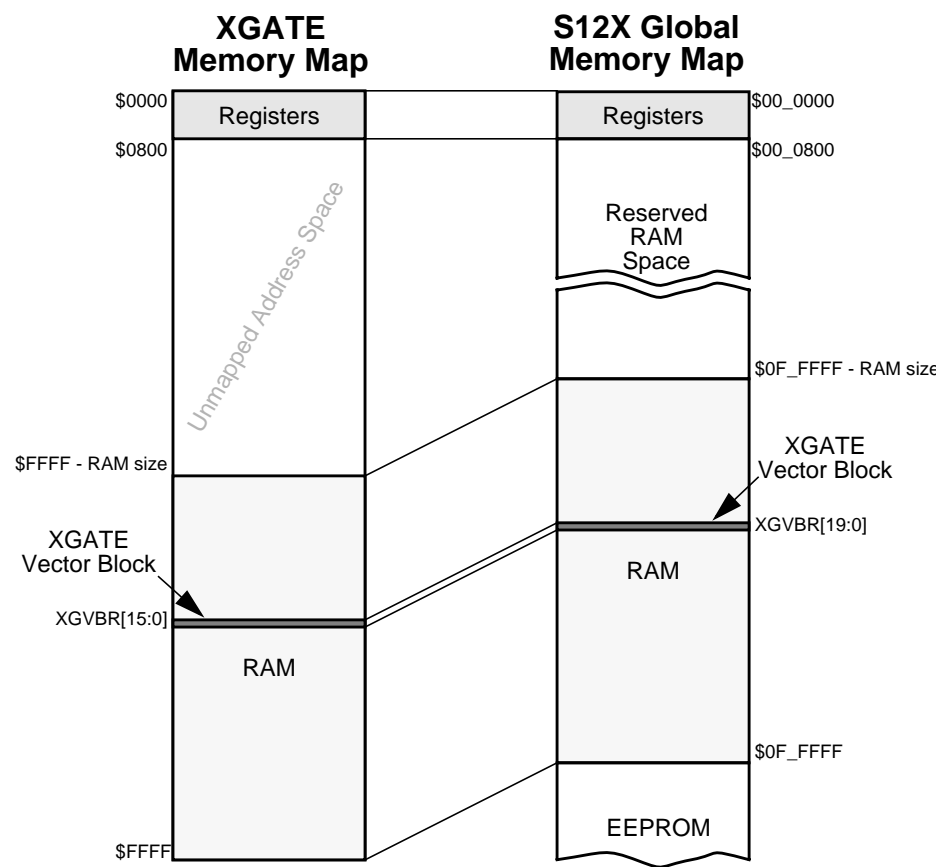
**Figure 4-1 Programmer's Model**

The programmer's model of the XGATE RISC core is shown in **Figure 4-1**. The processor offers a set of seven general purpose registers (R1 - R7), which serve as accumulators and index registers. An additional eighth register (R0) is tied to the value "\$0000". Register R1 has an additional functionality. It is preloaded with the initial variable pointer of the channel's service request vector (see **Figure 4-3**). The initial content of the remaining general purpose registers is undefined.

The 16 bit program counter allows the addressing of a 64 kbyte address space (see **Figure 4-2**).

The Condition Code Register contains four bits: The Sign bit (S), the Zero flag (Z), the Overflow flag (V) and the Carry bit (C). The initial content of the Condition Code Register is undefined.

### 4.1.2 Memory Map



**Figure 4-2 XGATE Memory Map**

The XGATE module is able to access an address space of 64 kb, which is divided into three segments: the Register Space, the RAM Space, and the Unmapped Address Space. **Figure 4-2** shows the XGATE memory map for RAM sizes smaller than 64 kb.

The 2 kb Register Space of the S12X MCU ranges from address \$0000 - \$0800.

The RAM Address Space is aligned to address \$FFFF. Its start address depends on the RAM size.

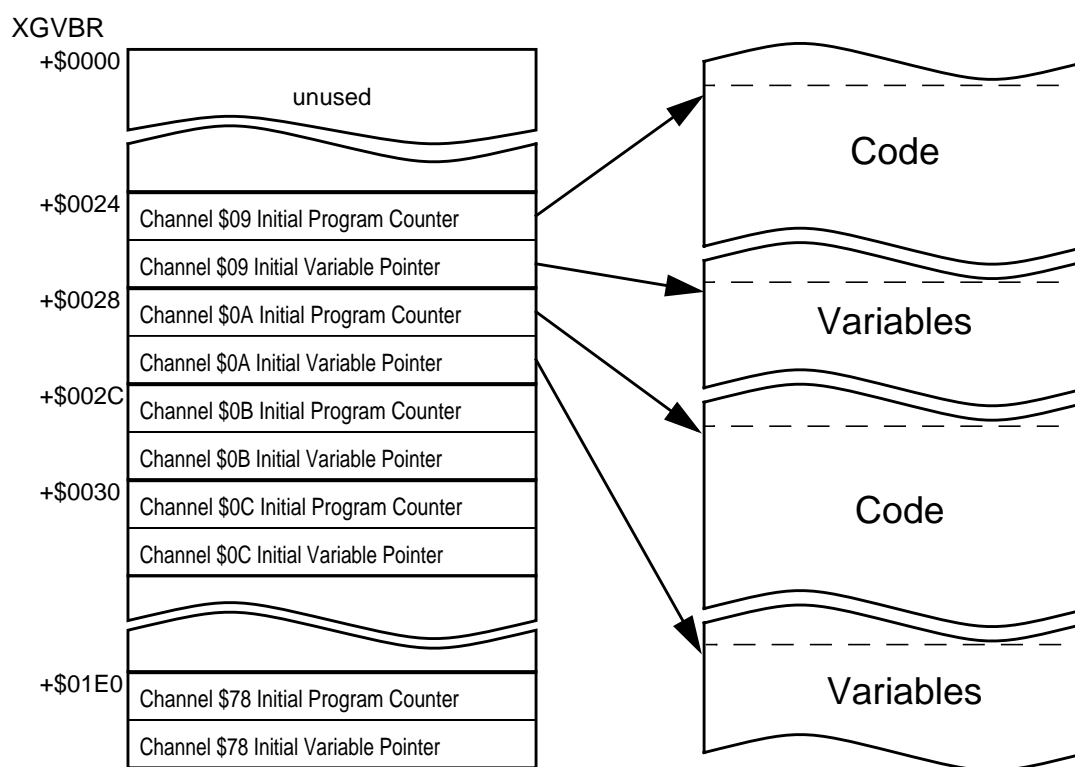
For RAM sizes smaller than 62 kb, there will be a range of unmapped addresses (Unmapped Address Space). Read accesses to this address range will return the value zero. Write accesses have no effect.

For RAM sizes larger than 62 kb, the Register Space and The RAM Space will overlap. In this case accesses to the address range \$0000 to \$0800 will still address the Register Space.

For RAM sizes larger than 64 kb only a 64 kb window of the RAM can be mapped into the XGATE address space. This window is determined by bits 19 to 16 of the XGATE Vector Base Register (XGVBR[19:16], see **3.1.3**)

Within the RAM space, certain regions can be protected from XGATE accesses. Refer to section 4 of the **XSRAM Block User Guide** for a detailed description.

The XGATE vector block assigns a start address and a variable pointer to each XGATE channel. It must be located in the RAM space. Its exact position can be adjusted through the XGVBR register (see **3.1.3**). **Figure 4-3** shows the layout of the vector block. Each vector consists of two 16-bit words. The first contains the start address of the service routine. This value will be loaded into the Program Counter before a service routine is executed. The second word is a pointer to the service routine's variable space. This value will be loaded into register R1 before a service routine is executed.



**Figure 4-3 XGATE Vector Block**

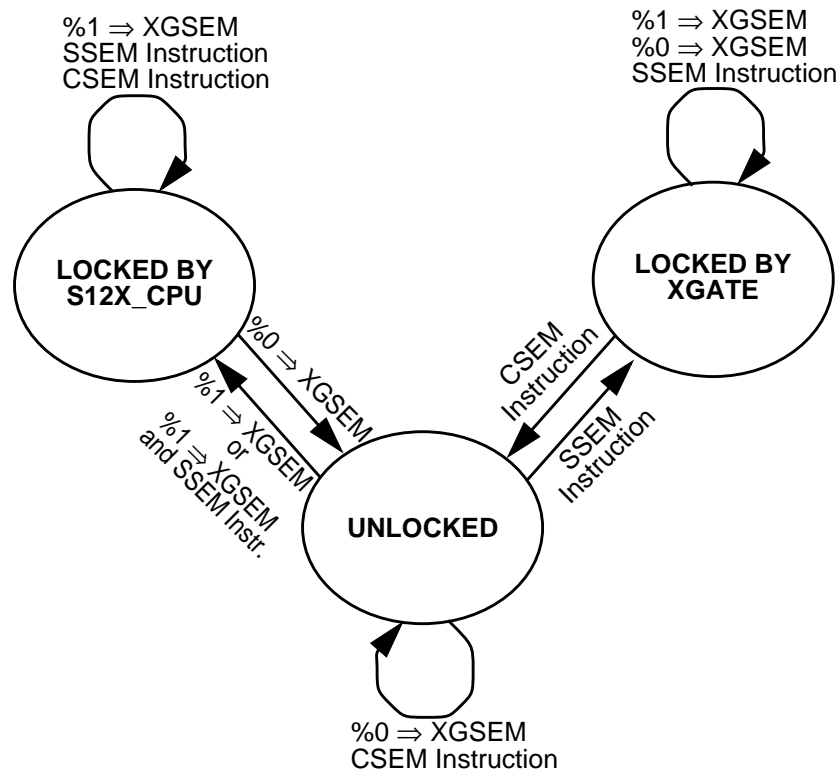
## 4.2 Semaphores

The XGATE module offers a set of eight hardware semaphores. These semaphores provide a mechanism to protect system resources that are shared between two concurrent threads of program execution. Typically one thread will run on the S12X\_CPU and one will run on the XGATE RISC core.

Each semaphore can only be in one of the three states: “Unlocked”, “Locked by S12X\_CPU”, and “Locked by XGATE”. The S12X\_CPU can check and change a semaphore’s state through the XGATE semaphore register (XGSEM, see **3.1.6**). The RISC core does this through its SSEM and CSEM instructions.

If the S12X\_CPU and the RISC core attempt to lock an unlocked semaphore at the same time, it will be locked by the S12X\_CPU.

**Figure 4-4** illustrates the valid state transitions.



**Figure 4-4 Semaphore State Transitions**

**Figure 4-5** gives an example of the typical usage of the XGATE hardware semaphores:

Two concurrent threads are running on the system. One is running on the S12X\_CPU and the other is running on the RISC core. They both have a critical section of code that accesses the same system resource. To guarantee that the system resource is only accessed by one thread at a time, the critical code sequence must be embedded in a Semaphore lock/release sequence as shown.

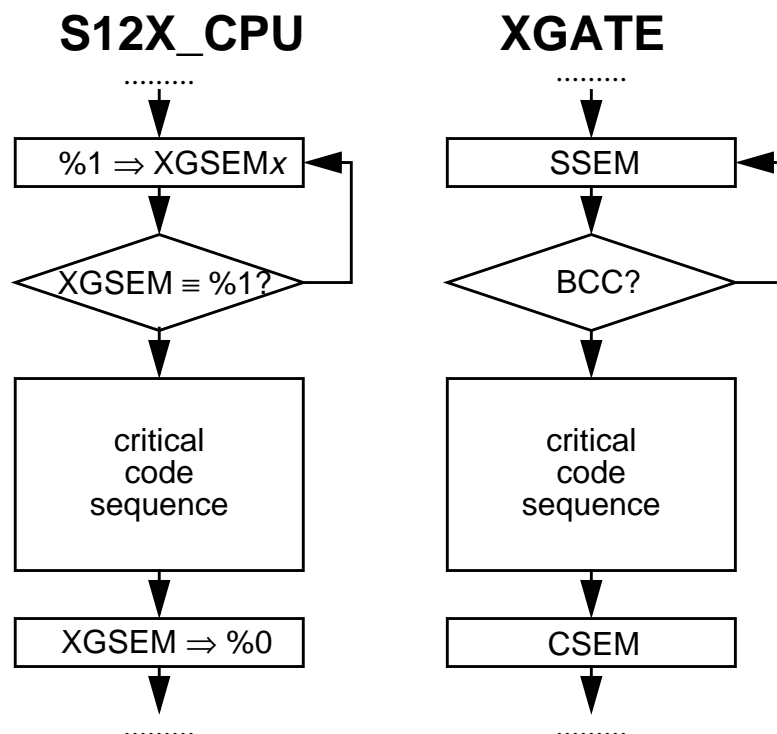


Figure 4-5 Algorithm for Locking and Releasing Semaphores

### 4.3 Software Error Detection

The XGATE module will immediately terminate program execution after detecting an error condition caused by erratic application code. These Error conditions are:

- Vector fetches from peripheral address space
- Vector fetches from unmapped address space
- Misaligned opcode fetches
- Opcode fetches from peripheral address space
- Opcode fetches from unmapped address space
- Misaligned load/store word accesses
- Load/store accesses to unmapped address space
- Illegal Opcodes

As soon as an error condition is detected, the XGSWEIF bit will be set. Clearing the XGSWEIF through the S12X\_CPU will put the RISC core into idle mode.

Illegal bus accesses will be executed before the XGATE stops on the error condition.

All opcodes which are not listed in section 4.7.5 are considered illegal opcodes.

**NOTE:** *Executing an instructions which is located at address \$FFFE will result in a Software Error because an opcode prefetch to address \$0000 will occur.*

## 4.4 Interrupts

### 4.4.1 Incoming Interrupt Requests

XGATE threads are triggered by interrupt requests which are routed to the XGATE module (see **S12X\_INT Block User Guide**). Only a subset of the MCU's interrupt requests can be routed to the XGATE. Which specific interrupt requests these are and which channel ID they are assigned to is documented in section 5.2 of the **SoC Guide**.

### 4.4.2 Outgoing Interrupt Requests

There are three types of interrupt requests which can be triggered by the XGATE module:

- **Channel Interrupts:**  
For each XGATE channel there is an associated interrupt flag in the XGATE Interrupt Flag Vector (XGIF, see 3.1.4). These flags can be set through the "SIF" instruction by the RISC core. They are typically used to flag an interrupt to the S12X\_CPU when the XGATE has completed one of its tasks.
- **Software Triggers:**  
Software Triggers are interrupt flags, which can be set and cleared by software (see 3.1.5). They are typically used to trigger XGATE tasks by the S12X\_CPU software. However these interrupts can also be routed to the S12X\_CPU (see **S12X\_INT Block User Guide**) and triggered by the XGATE software.
- **Software Error Interrupt:**  
The Software Error Interrupt signals to the S12X\_CPU the detection of an error condition in the XGATE application code (see 4.3).

All XGATE Interrupts can be disabled by the XGIE bit in the XGATE Module Control Register (XGMCTL, see 3.1.1).

## 4.5 Debug Mode

The XGATE Debug Mode is a feature to allow debugging of application code.

### 4.5.1 Debug Features

In Debug mode the RISC core will be halted and the following debug features will be enabled:

- Read and Write accesses to RISC core registers (XGCCCR, XGPC, XGR1-XGR7)<sup>1</sup>

All RISC core registers can be modified. Leaving Debug Mode will cause the RISC core to continue program execution with the modified register values.

- Single Stepping<sup>1</sup>

Writing a "1" to the XGSS bit will call the RISC core to execute a single instruction. All RISC core registers will be updated accordingly.

- Write accesses to the XGCHID register

Three operations can be performed by writing to the XGCHID register:

- Change of channel ID:

If a non-zero value is written to the XGCHID while a thread is active ( $XGCHID \neq \$00$ ), then the current channel ID will be changed without any influence on the program counter or the other RISC core registers.

- Start of a thread:

If a non-zero value is written to the XGCHID while the XGATE is idle ( $XGCHID = \$00$ ), then the thread that is associated with the new channel ID will be executed upon leaving Debug Mode.

- Termination of a thread:

If zero is written to the XGCHID while a thread is active ( $XGCHID \neq \$00$ ), then the current thread will be terminated and the XGATE will become idle.

## 4.5.2 Entering Debug Mode

Debug Mode can be entered in four ways:

- Setting XGDBG to "1"

Writing a "1" to XGDBG and XGDBGM in the same write access causes the XGATE to enter Debug Mode upon completion of the current instruction.

**NOTE:** *After writing to the XGDBG bit the XGATE will not immediately enter Debug Mode. Depending on the instruction that is executed at this time there may be a delay of several clock cycles. The XGDBG will read "0" until Debug Mode is entered.*

- Software Breakpoints

XGATE programs are stored in RAM which allows the use of Software Breakpoints. A Software Breakpoint is set by replacing an instruction of the program code with the "BRK" instruction.

As soon as the program execution reaches the "BRK" instruction, the XGATE enters Debug Mode. Additionally a Software Breakpoint Request is sent to the S12X\_DBG module (see section 4.9 of the **S12X\_DBG Block User Guide**).

### NOTES:

1. Only possible if MCU is unsecured



Upon entering Debug Mode, the program counter will point to the "BRK" instruction. The other RISC core register will hold the result of the previous instruction.

To resume program execution, the "BRK" instruction must be replaced by the original instruction before leaving Debug Mode.

- Tagged Breakpoints

The S12X\_DBG module is able to place tags on fetched opcodes. The XGATE is able to enter Debug Mode right before a tagged opcode is executed (see section 4.9 of the **S12X\_DBG Block User Guide**). Upon entering Debug Mode, the program counter will point to the tagged instruction. The other RISC core register will hold the result of the previous instruction.

- Forced Breakpoints

Forced breakpoints are triggered by the S12X\_DBG module (see section 4.9 of the **S12X\_DBG Block User Guide**). When a forced breakpoint occurs, the XGATE will enter Debug Mode upon completion of the current instruction.

### 4.5.3 Leaving Debug Mode

Debug Mode can only be left by setting the XGDBG bit to "0". If a thread is active (XGCHID has not been cleared in Debug Mode), program execution will resume at the value of XGPC.

## 4.6 Security

In order to protect XGATE application code on secured S12X devices, a few restrictions in the debug features have been made. These are:

- Registers XGCCR, XGPC, and XGR1 - XGR7 will read zero on a secured device
- Registers XGCCR, XGPC, and XGR1 - XGR7 can not be written on a secured device
- Single Stepping is not possible on a secured device

## 4.7 Instruction Set

### 4.7.1 Addressing Modes

For the ease of implementation the architecture is a strict Load/Store RISC machine, which means all operations must have one of the eight general purpose registers R0 ... R7 as their source as well their destination.

All word accesses must work with a word aligned address e.g. A0 = 0!

#### 4.7.1.1 Naming Conventions

RD - destination register, allowed range is R0 - R7

RD.L - low byte of the destination register, bits [7:0]

RD.H - high byte of the destination register, bits [15:8]

RS, RS1, RS2 - source register, allowed range is R0 - R7  
 RS.L, RS1.L, RS2.L - low byte of the source register, bits [7:0]  
 RS.H, RS1.H, RS2.H - high byte of the source register, bits[15:8]

RB - base register for indexed addressing modes, allowed range is R0 - R7  
 RI - offset register for indexed addressing modes with register offset, allowed range is R0 - R7  
 RI+ - offset register for indexed addressing modes with register offset and post-increment, allowed range is R0 - R7 (R0+ is equivalent to R0)  
 -RI - offset register for indexed addressing modes with register offset and pre-decrement, allowed range is R0 - R7 (-R0 is equivalent to R0)

**NOTE:** Even though register R1 is intended to be used as a pointer to the variable segment, it may be used as a general purpose data register as well.

**NOTE:** Selecting R0 as destination register will discard the result of the instruction. Only the Condition Code Register will be updated

#### 4.7.1.2 Inherent Addressing Mode (INH)

Instructions that use this addressing mode either have no operands or all operands are in internal XGATE registers.

Examples

```
BRK
RTS
```

#### 4.7.1.3 Immediate 3 Bit Wide (IMM3)

Operands for immediate mode instructions are included in the instruction stream and are fetched into the instruction queue along with the rest of the 16 Bit instruction. The '#' symbol is used to indicate an immediate addressing mode operand. This address mode is used for shift instructions.

Examples:

```
CSEM    #1      ; Unlock semaphore 1
SSEM    #3      ; Lock Semaphore 3
```

#### 4.7.1.4 Immediate 4 Bit Wide (IMM4)

Operands for immediate mode instructions are included in the instruction stream and are fetched into the instruction queue along with the rest of the 16 Bit instruction. The '#' symbol is used to indicate an immediate addressing mode operand. This address mode is used for shift instructions.

$RD = RD * imm4$

Examples:

```
LSL     R4,#1    ; R4 = R4 << 1; shift register R4 by 1 bit to the left
LSR     R4,#3    ; R4 = R4 >> 3; shift register R4 by 3 bits to the right
```

#### 4.7.1.5 Immediate 8 Bit Wide (IMM8)

Operands for immediate mode instructions are included in the instruction stream and are fetched into the instruction queue along with the upper byte of the 16 bit instruction. The '#' symbol is used to indicate an immediate addressing mode operand. Four major commands (ADD, SUB, LD, CMP) support this addressing mode.

$RD = RD * imm8$

Examples:

```
ADDL    R1,#IMM8    ; adds an 8 Bit value to register R1
SUBL    R2,#IMM8    ; subtracts an 8 Bit value from register R2
LDH     R3,#IMM8    ; loads an 8 bit immediate into the high byte of Register R3
CMPL    R4,#IMM8    ; compares the low byte of register R4 with an immediate value
```

#### 4.7.1.6 Monadic Addressing (MON)

In this addressing mode only one operand is explicitly given. This operand can either be the source ( $f(RD)$ ), the target ( $RD = f()$ ), or both source and target of the operation ( $RD = f(RD)$ ).

Examples:

```
JAL     R1           ; PC = R1, R1 = PC+2
SIF     R2           ; Trigger IRQ associated with the channel number in R2.L
```

#### 4.7.1.7 Dyadic Addressing (DYA)

In this mode the result of an operation between two registers is stored in one of the registers used as operands.

$RD = RD * RS$  is the general register to register format, with register RD being the first operand and RS the second. RD and RS can be any of the 8 general purpose registers R0 ... R7. If R0 is used as the destination register, only the condition code flags are updated. This addressing mode is used only for shift operations with a variable shift value

Examples:

```
LSL     R4,R5        ; R4 = R4 << R5
LSR     R4,R5        ; R4 = R4 >> R5
```

#### 4.7.1.8 Triadic Addressing (TRI)

In this mode the result of an operation between two or three registers is stored into a third one.

$RD = RS1 * RS2$  is the general format used in the order RD, RS1, RS1. RD, RS1, RS2 can be any of the 8 general purpose registers R0 ... R7. If R0 is used as the destination register RD, only the condition code flags are updated. This addressing mode is used for all arithmetic and logical operations.

Examples:

```
ADC     R5,R6,R7      ; R5 = R6 + R7 + Carry
SUB     R5,R6,R7      ; R5 = R6 - R7
```

#### 4.7.1.9 Relative Addressing 9 Bit Wide (REL9)

A 9-bit signed address offset is included in the instruction word. This addressing mode is used for conditional branch instructions.

Examples:

```
BCC      REL9      ; PC = PC + 2 + (REL9 << 1)
BEQ      REL9      ; PC = PC + 2 + (REL9 << 1)
```

#### 4.7.1.10 Relative Addressing 10 Bit Wide (REL10)

An 11-bit signed address offset is included in the instruction word. This addressing mode is used for the unconditional branch instruction.

Examples:

```
BRA      REL10     ; PC = PC + 2 + (REL10 << 1)
```

#### 4.7.1.11 Index Register plus Immediate Offset (IDO5)

(RS, #offset5) provides an unsigned offset from the base register.

Examples:

```
LDB      R4,(R1,#offset) ; loads a byte from R1+offset into R4
STW      R4,(R1,#offset) ; stores R4 as a word to R1+offset
```

#### 4.7.1.12 Index Register plus Register Offset (IDR)

For load and store instructions (RS, RI) provides a variable offset in a register.

Examples:

```
LDB      R4,(R1,R2)      ; loads a byte from R1+R2 into R4
STW      R4,(R1,R2)      ; stores R4 as a word to R1+R2
```

#### 4.7.1.13 Index Register plus Register Offset with Post-increment (IDR+)

[RS, RI+] provides a variable offset in a register, which is incremented after accessing the memory. In case of a byte access the index register will be incremented by one. In case of a word access it will be incremented by two.

Examples:

```
LDB      R4,(R1,R2+)     ; loads a byte from R1+R2 into R4, R2+=1
STW      R4,(R1,R2+)     ; stores R4 as a word to R1+R2, R2+=2
```

#### 4.7.1.14 Index Register plus Register Offset with Pre-decrement (-IDR)

[RS, -RI] provides a variable offset in a register, which is decremented before accessing the memory. In case of a byte access the index register will be decremented by one. In case of a word access it will be decremented by two.

Examples:

```
LDB      R4,(R1,-R2)      ; R2 -=1, loads a byte from R1+R2 into R4
STW      R4,(R1,-R2)      ; R2 -=2, stores R4 as a word to R1+R2
```

## 4.7.2 Instruction Summary and Usage

### 4.7.2.1 Load & Store Instructions

Any register can be loaded either with an immediate or from the address space using indexed addressing modes.

```
LDL      RD,#IMM8          ; loads an immediate 8 bit value to the lower byte of RD
LDW      RD,(RB,RI)        ; loads data using RS+RI as effective address

LDB      RD,(RB, RI+)      ; loads data using RS+RI as effective address
                        ; followed by an increment of RI depending on
                        ; the size of the operation
```

The same set of modes is available for the store instructions

```
STB      RS,(RB, RI)       ; stores data using RS+RI as effective address

STW      RS,(RB, RI+)      ; stores data using RS+RI as effective address
                        ; followed by an increment of RI depending on
                        ; the size of the operation.
```

### 4.7.2.2 Logic and Arithmetic Instructions

All logic and arithmetic instructions support the 8-bit immediate addressing mode (IMM8:  $RD = RD * \#IMM8$ ) and the triadic addressing mode (TRI:  $RD = RS1 * RS2$ ).

All arithmetic is considered as signed, sign, overflow, zero and carry flag will be updated. The carry will not be affected for logical operations.

```
ADDL     R2,#1             ; increment R2
ANDH     R4,#$FE           ; R4.H = R4.H & $FE, clear lower bit of higher byte

ADD      R3,R4,R5          ; R3 = R4 + R5
SUB      R3,R4,R5          ; R3 = R4 - R5

AND      R3,R4,R5          ; R3 = R4 & R5 logical AND on the whole word
OR       R3,R4,R5          ; R3 = R4 | R5
```

### 4.7.2.3 Register - Register transfers

This group comprises transfers from and to some special registers

```
TFR      R3,CCR            ; transfers the condition code register to the low byte of
                        ; register R3
```

#### 4.7.2.4 Branch Instructions

The branch offset is +255 words or -256 words counted from the beginning of the next instruction. Since instructions have a fixed 16 bit width, the branch offsets are word aligned by shifting the offset value by 2.

```
BEQ      label          ; if Z flag = 1 branch to label
```

An unconditional branch allows a +511 words or -512 words branch distance.

```
BRA      label
```

#### 4.7.2.5 Shift Instructions

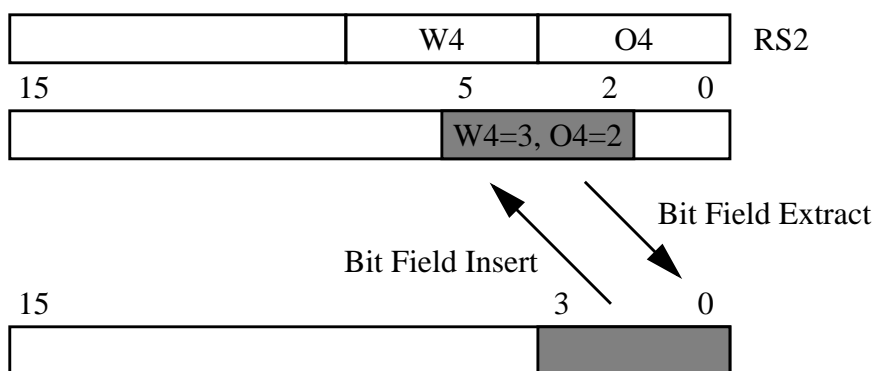
Shift operations allow the use of a 4 bit wide immediate value to identify a shift width within a 16 bit word. For shift operations a value of 0 does not shift at all, while a value of 15 shifts the register RD by 15 bits. In a second form the shift value is contained in the bits 3:0 of the register RS.

Examples:

```
LSL      R4,#1          ; R4 = R4 << 1; shift register R4 by 1 bit to the left
LSR      R4,#3          ; R4 = R4 >> 3; shift register R4 by 3 bits to the right
ASR      R4,R2          ; R4 = R4 >> R2; arithmetic shift register R4 right by the amount
                        ; of bits contained in R2[3:0].
```

#### 4.7.2.6 Bit Field Operations

This addressing mode is used to identify the position and size of a bit field for insertion or extraction. The width and offset are coded in the lower byte of the source register 2, RS2. The content of the upper byte is ignored. An offset of 0 denotes the right most position and a width of 0 denotes 1 bit. These instructions are very useful to extract, insert, clear, set or toggle portions of a 16 bit word.



**Figure 4-6 Bit Field Addressing**

```
BFEXT    R3,R4,R5      ; R5: W4 bits offset O4, will be extracted from R4 into R3
```

### 4.7.2.7 Special Instructions for DMA usage

The XGATE offers a number of additional instructions for flag manipulation, program flow control and debugging:

1. SIF: Set a channel interrupt flag
2. SSEM: Test and set a hardware semaphore
3. CSEM: Clear a hardware semaphore
4. BRK: Software breakpoint
5. NOP: No Operation
6. RTS: Terminate the current thread

### 4.7.3 Cycle Notation

**Table 4-1** show the XGATE access detail notation. Each code letter equals one XGATE cycle. Each letter implies additional wait cycles if RAM or peripherals are not accessible. RAM or peripherals are not accessible if they are blocked by the S12X\_CPU. In addition to this Peripherals are only accessible every other XGATE cycle. Uppercase letters denote 16-bit operations. Lowercase letters denote 8-bit operations. The XGATE is able to perform two bus or wait cycles per S12X\_CPU cycle.

**Table 4-1 Access Detail Notation**

v	— Vector fetch: always an aligned word read, lasts for at least one RISC core cycle
P	— Program word fetch: always an aligned word read, lasts for at least one RISC core cycle
r	— 8-bit data read: lasts for at least one RISC core cycle
R	— 16-bit data read: lasts for at least one RISC core cycle
w	— 8-bit data write: lasts for at least one RISC core cycle
W	— 16-bit data write: lasts for at least one RISC core cycle
A	— Alignment cycle: no read or write, lasts for zero or one RISC core cycles
f	— Free cycle: no read or write, lasts for one RISC core cycles

**Special Cases**

PP/P — Branch: PP if branch taken, P if not

### 4.7.4 Thread Execution

When the RISC core is triggered by an interrupt request (see **Figure 1-1**) it first executes a vector fetch sequence which performs three bus accesses:

1. A V-cycle to fetch the initial content of the program counter.
2. A V-cycle to fetch the initial content of the data segment pointer (R1).
3. A P-cycle to load the initial opcode.

Afterwards a sequence of instructions (thread) is executed which is terminated by an "RTS" instruction. If further interrupt requests are pending after a thread has been terminated, a new vector fetch will be

performed. Otherwise the RISC core will idle until a new interrupt request is received. A thread can not be interrupted by an interrupt request.

### **4.7.5 Instruction Glossary**

The following section describes the XGATE instruction set in alphabetical order.



ADC

Add with Carry

ADC

Operation

$RS1 + RS2 + C \Rightarrow RD$

Adds the content of register RS1, the content of register RS2 and the value of the Carry bit using binary addition and stores the result in the destination register RD. The Zero Flag is also carried forward from the previous operation allowing 32 and more bit additions.

Example :

```
ADC      R6,R2,R2
ADC      R7,R3,R3 ; R7:R6 = R5:R4 + R3:R2
BCC      ; conditional branch on 32 bit addition
```

CCR Effects

N	Z	V	C
Δ	Δ	Δ	Δ

- N: Set if bit 15 of the result is set; cleared otherwise.
- Z: Set if the result is \$0000 and Z was set before this operation; cleared otherwise.
- V: Set if a two’s complement overflow resulted from the operation; cleared otherwise.
- $RS1[15] \& RS2[15] \& \overline{RD[15]}_{new} \mid \overline{RS1[15]} \& \overline{RS2[15]} \& RD[15]_{new}$
- C: Set if there is a carry from bit 15 of the result; cleared otherwise.
- $RS1[15] \& RS2[15] \mid RS1[15] \& \overline{RD[15]}_{new} \mid \overline{RS2[15]} \& \overline{RD[15]}_{new}$

Code and CPU Cycles

Source Form	Address Mode	Machine Code								Cycles
ADC RD, RS1, RS2	TRI	0	0	0	1	1	RD	RS1	RS2	1 1 P

ADD

Add without Carry

ADD

Operation

$RS1 + RS2 \Rightarrow RD$

Adds the content of register RS1 and the content of register RS2 using binary addition and stores the result in the destination register RD.

CCR Effects

N	Z	V	C
$\Delta$	$\Delta$	$\Delta$	$\Delta$

- N: Set if bit 15 of the result is set; cleared otherwise.
- Z: Set if the result is \$0000; cleared otherwise.
- V: Set if a two’s complement overflow resulted from the operation; cleared otherwise.  
 $RS1[15] \& RS2[15] \& \overline{RD[15]}_{new} \mid \overline{RS1[15]} \& \overline{RS2[15]} \& RD[15]_{new}$
- C: Set if there is a carry from bit 15 of the result; cleared otherwise.  
 $RS1[15] \& RS2[15] \mid RS1[15] \& \overline{RD[15]}_{new} \mid RS2[15] \& \overline{RD[15]}_{new}$

Code and CPU Cycles

Source Form	Address Mode	Machine Code								Cycles	
ADD RD, RS1, RS2	TRI	0	0	0	1	1	RD	RS1	RS2	1	0 P

ADDH

Add Immediate 8-Bit Constant  
(High Byte)

ADDH

Operation

$RD + IMM8: \$00 \Rightarrow RD$

Adds the content of high byte of register RD and a signed immediate 8-Bit constant using binary addition and stores the result in the high byte of the destination register RD. This instruction can be used after an ADDL for a 16-bit immediate addition.

EXAMPLE :

```
ADDL    R2, #LOWBYTE
ADDH    R2, #HIGHBYTE    ; R2 = R2 + 16 Bit immediate
```

CCR Effects

N	Z	V	C
Δ	Δ	Δ	Δ

- N: Set if bit 15 of the result is set; cleared otherwise.
- Z: Set if the result is \$0000; cleared otherwise.
- V: Set if a two’s complement overflow resulted from the operation; cleared otherwise.
- $RD[15]_{old} \& IMM8[7] \& \overline{RD[15]}_{new} \mid \overline{RD[15]}_{old} \& IMM8[7] \& RD[15]_{new}$
- C: Set if there is a carry from the bit 15 of the result; cleared otherwise.
- $RD[15]_{old} \& IMM8[7] \mid RD[15]_{old} \& \overline{RD[15]}_{new} \mid IMM8[7] \& \overline{RD[15]}_{new}$

Code and CPU Cycles

Source Form	Address Mode	Machine Code							Cycles
ADDH RD, #IMM8	IMM8	1	1	1	0	1	RD	IMM8	P

ADDL

Add Immediate 8-Bit Constant  
(Low Byte)

ADDL

Operation

$RD + \$00:IMM8 \Rightarrow RD$

Adds the content of register RD and an unsigned immediate 8-Bit constant using binary addition and stores the result in the destination register RD. This instruction must be used first for a 16-bit immediate addition in conjunction with the ADDH instruction.

CCR Effects

N	Z	V	C
Δ	Δ	Δ	Δ

- N: Set if bit 15 of the result is set; cleared otherwise.
- Z: Set if the result is \$0000; cleared otherwise.
- V: Set if a two’s complement overflow resulted from the 8-bit operation; cleared otherwise.
- $\overline{RD[15]}_{old} \& RD[15]_{new}$
- C: Set if there is a carry from bit 7 to bit 8 of the result; cleared otherwise.
- $RD[15]_{old} \& \overline{RD[15]}_{new}$

Code and CPU Cycles

Source Form	Address Mode	Machine Code							Cycles
ADDL RD, #IMM8	IMM8	1	1	1	0	0	RD	IMM8	P

# AND

## Logical AND

# AND

### Operation

RS1 & RS2  $\Rightarrow$  RD

Performs a bit wise logical AND between the content of register RS1 and the content of register RS2 and stores the result in the destination register RD.

Remark: There is no complement to the BITH and BITL functions. This can be imitated by using R0 as a destination register. AND R0, RS1, RS2 performs a bit wise test without storing a result.

### CCR Effects

N	Z	V	C
$\Delta$	$\Delta$	0	—

N: Set if bit 15 of the result is set; cleared otherwise.

Z: Set if the result is \$0000; cleared otherwise.

V: 0; cleared.

C: Not affected.

### Code and CPU Cycles

Source Form	Address Mode	Machine Code										Cycles
AND RD, RS1, RS2	TRI	0	0	0	1	0	RD	RS1	RS2	0	0	P

# ANDH

Logical AND Immediate 8-Bit Constant  
(High Byte)

# ANDH

## Operation

$RD.H \& IMM8 \Rightarrow RD.H$

Performs a bit wise logical AND between the high byte of register RD and an immediate 8-Bit constant and stores the result in the destination register RD.H. The low byte of RD is not affected.

## CCR Effects

N	Z	V	C
$\Delta$	$\Delta$	0	—

N: Set if bit 15 of the result is set; cleared otherwise.

Z: Set if the 8-bit result is \$00; cleared otherwise.

V: 0; cleared.

C: Not affected.

## Code and CPU Cycles

Source Form	Address Mode	Machine Code							Cycles
ANDH RD, #IMM8	IMM8	1	0	0	0	1	RD	IMM8	P

# ANDL

Logical AND Immediate 8-Bit Constant  
(Low Byte)

# ANDL

## Operation

RD.L & IMM8  $\Rightarrow$  RD.L

Performs a bit wise logical AND between the low byte of register RD and an immediate 8-Bit constant and stores the result in the destination register RD.L. The high byte of RD is not affected.

## CCR Effects

**N    Z    V    C**

$\Delta$	$\Delta$	0	—
----------	----------	---	---

N: Set if bit 7 of the result is set; cleared otherwise.

Z: Set if the 8-Bit result is \$00; cleared otherwise.

V: 0; cleared.

C: Not affected.

## Code and CPU Cycles

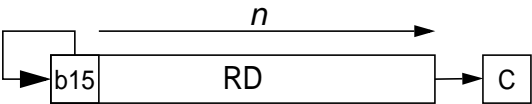
Source Form	Address Mode	Machine Code						Cycles
ANDL RD, #IMM8	IMM8	1	0	0	0	0	RD    IMM8	P

# ASR

## Arithmetic Shift Right

# ASR

### Operation



$n = \text{RS or IMM4}$

Shifts the bits in register RD  $n$  positions to the right. The higher  $n$  bits of the register RD become filled with the sign bit (RD[15]). The carry flag will be updated to the bit contained in RD[n-1] before the shift for  $n > 0$ .

$n$  can range from 0 to 16.

In immediate address mode,  $n$  is determined by the operand IMM4.  $n$  is considered to be 16 in IMM4 is equal to 0.

In dyadic address mode,  $n$  is determined by the content of RS.  $n$  is considered to be 16 if the content of RS is greater than 15.

### CCR Effects

N	Z	V	C
$\Delta$	$\Delta$	0	$\Delta$

N: Set if bit 15 of the result is set; cleared otherwise.

Z: Set if the result is \$0000; cleared otherwise.

V: Set if a two's complement overflow resulted from the operation; cleared otherwise.

$$\text{RD}[15]_{\text{old}} \wedge \text{RD}[15]_{\text{new}}$$

C: Set if  $n > 0$  and RD[n-1] = 1; if  $n = 0$  unaffected.

### Code and CPU Cycles

Source Form	Address Mode	Machine Code											Cycles	
ASR RD, #IMM4	IMM4	0	0	0	0	1	RD	IMM4		1	0	0	1	P
ASR RD, RS	DYA	0	0	0	0	1	RD	RS	1	0	0	0	1	P



BCC

Branch if Carry Cleared  
(Same as BHS)

BCC

Operation

If C = 0, then PC + \$0002 + (REL9 << 1) ⇒ PC

Tests the Carry flag and branches if C = 0.

CCR Effects

N	Z	V	C
—	—	—	—

- N: Not affected.
- Z: Not affected.
- V: Not affected.
- C: Not affected.

Code and CPU Cycles

Source Form	Address Mode	Machine Code								Cycles
BCC REL9	REL9	0	0	1	0	0	0	0	REL9	PP/P

# BCS

Branch if Carry Set  
(Same as BLO)

# BCS

### Operation

If C = 1, then PC + \$0002 + (REL9 << 1) ⇒ PC

Tests the Carry flag and branches if C = 1.

### CCR Effects

N	Z	V	C
—	—	—	—

N: Not affected.  
Z: Not affected.  
V: Not affected.  
C: Not affected.

### Code and CPU Cycles

Source Form	Address Mode	Machine Code								Cycles
BCS REL9	REL9	0	0	1	0	0	0	1	REL9	PP/P

# BEQ

Branch if Equal

# BEQ

## Operation

If  $Z = 1$ , then  $PC + \$0002 + (REL9 \ll 1) \Rightarrow PC$

Tests the Zero flag and branches if  $Z = 1$ .

## CCR Effects

N	Z	V	C
–	–	–	–

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

## Code and CPU Cycles

Source Form	Address Mode	Machine Code								Cycles
BEQ REL9	REL9	0	0	1	0	0	1	1	REL9	PP/P

# BFEXT

## Bit Field Extract

# BFEXT

### Operation

$$RS1[(o+w):o] \Rightarrow RD[w:0]; 0 \Rightarrow RD[15:(w+1)]$$

$$w=(RS2[7:4])$$

$$o=(RS2[3:0])$$

Extracts  $w+1$  bits from register RS1 starting at position  $o$  and writes them right aligned into register RD. The remaining bits in RD will be cleared. If  $(o+w) > 15$  only bits  $[15:o]$  get extracted.

### CCR Effects

N	Z	V	C
$\Delta$	$\Delta$	0	—

N: Set if bit 15 of the result is set; cleared otherwise.

Z: Set if the result is \$0000; cleared otherwise.

V: 0; cleared.

C: Not affected.

### Code and CPU Cycles

Source Form	Address Mode	Machine Code										Cycles
BFEXT RD, RS1, RS2	TRI	0	1	1	0	0	RD	RS1	RS2	1	1	P

# BFFO

## Bit Field Find First One

# BFFO

### Operation

FirstOne (RS)  $\Rightarrow$  RD;

Searches the first “1” beginning from the MSB=15 down to LSB=0 in register RS and places the result into the destination register RD. The upper bits of RD are cleared. In case the content of RS is equal to \$0000, RD will be cleared and the carry flag will be set. This is used to distinguish a “1” in position 0 versus no “1” in the whole RS register at all.

### CCR Effects

N	Z	V	C
0	$\Delta$	0	$\Delta$

N: 0; cleared.

Z: Set if the result is \$0000; cleared otherwise.

V: 0; cleared.

C: Set if RS = \$0000<sup>1</sup>; cleared otherwise.

NOTES:

1. before executing the instruction

### Code and CPU Cycles

Source Form	Address Mode	Machine Code										Cycles		
BFFO RD, RS	DYA	0	0	0	0	1	RD	RS	1	0	0	0	0	P

# BFINS

## Bit Field Insert

# BFINS

### Operation

$$RS1[w:0] \Rightarrow RD[(w+o):o];$$

$$w=(RS2[7:4])$$

$$o=(RS2[3:0])$$

Extracts  $w+1$  bits from register RS1 starting at position 0 and writes them into register RD at position  $o$ . The remaining bits in RD are not affected. If  $(o+w) > 15$  the upper bits are ignored. Using R0 as a RS1, this command can be used to clear bits.

### CCR Effects

N	Z	V	C
Δ	Δ	0	—

- N: Set if bit 15 of the result is set; cleared otherwise.  
Z: Set if the result is \$0000; cleared otherwise.  
V: 0; cleared.  
C: Not affected.

### Code and CPU Cycles

Source Form	Address Mode	Machine Code								Cycles
BFINS RD, RS1, RS2	TRI	0	1	1	0	1	RD	RS1	RS2	1 1 P

BFINSI

Bit Field Insert and Invert

BFINSI

Operation

$!RS1[w:o] \Rightarrow RD[w+o:o];$

$w=(RS2[7:4])$   
 $o=(RS2[3:0])$

Extracts  $w+1$  bits from register RS1 starting at position 0, inverts them and writes into register RD at position  $o$ . The remaining bits in RD are not affected. If  $(o+w) > 15$  the upper bits are ignored. Using R0 as a RS1, this command can be used to set bits.

CCR Effects

N	Z	V	C
Δ	Δ	0	–

- N: Set if bit 15 of the result is set; cleared otherwise.
- Z: Set if the result is \$0000; cleared otherwise.
- V: 0; cleared.
- C: Not affected.

Code and CPU Cycles

Source Form	Address Mode	Machine Code								Cycles
BFINSI RD, RS1, RS2	TRI	0	1	1	1	0	RD	RS1	RS2	1 1 P

BFINSX

Bit Field Insert and XNOR

BFINSX

Operation

$$!(RS1[w:0] \wedge RD[w+o:0]) \Rightarrow RD[w+o:0];$$

$$w=(RS2[7:4])$$
$$o=(RS2[3:0])$$

Extracts  $w+1$  bits from register RS1 starting at position 0, performs an XNOR with RD[w+o:0] and writes the bits back. The remaining bits in RD are not affected. If  $(o+w) > 15$  the upper bits are ignored. Using R0 as a RS1, this command can be used to toggle bits.

CCR Effects

N	Z	V	C
Δ	Δ	0	–

- N: Set if bit 15 of the result is set; cleared otherwise.  
Z: Set if the result is \$0000; cleared otherwise.  
V: 0; cleared.  
C: Not affected.

Code and CPU Cycles

Source Form	Address Mode	Machine Code								Cycles
BFINSX RD, RS1, RS2	TRI	0	1	1	1	1	RD	RS1	RS2	1 1 P



BGE

Branch if Greater than or Equal to Zero

BGE

Operation

If  $N \wedge V = 0$ , then  $PC + \$0002 + (REL9 \ll 1) \Rightarrow PC$

Branch instruction to compare signed numbers.  
Branch if  $RS1 \geq RS2$ :

SUBR0,RS1,RS2

BGEREL9

CCR Effects

N	Z	V	C
–	–	–	–

- N: Not affected.
- Z: Not affected.
- V: Not affected.
- C: Not affected.

Code and CPU Cycles

Source Form	Address Mode	Machine Code								Cycles
BGE REL9	REL9	0	0	1	1	0	1	0	REL9	PP/P

# BGT

Branch if Greater than Zero

# BGT

## Operation

If  $Z \mid (N \wedge V) = 0$ , then  $PC + \$0002 + (REL9 \ll 1) \Rightarrow PC$

Branch instruction to compare signed numbers.

Branch if  $RS1 > RS2$ :

SUB	R0,RS1,RS2
BGE	REL9

## CCR Effects

N	Z	V	C
–	–	–	–

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

## Code and CPU Cycles

Source Form	Address Mode	Machine Code								Cycles
BGT REL9	REL9	0	0	1	1	1	0	0	REL9	PP/P

BHI

Branch if Higher

BHI

Operation

If C | Z = 0, then PC + \$0002 + (REL9 << 1) ⇒ PC

Branch instruction to compare unsigned numbers.  
Branch if RS1 > RS2:

SUBR0,RS1,RS2

BHIR

CCR Effects

N	Z	V	C
–	–	–	–

- N: Not affected.
- Z: Not affected.
- V: Not affected.
- C: Not affected.

Code and CPU Cycles

Source Form	Address Mode	Machine Code								Cycles
BHI REL9	REL9	0	0	1	1	0	0	0	REL9	PP/P

# BHS

Branch if Higher or Same  
(Same as BCC)

# BHS

## Operation

If C = 0, then  $PC + \$0002 + (REL9 \ll 1) \Rightarrow PC$

Branch instruction to compare unsigned numbers.  
Branch if  $RS1 \geq RS2$ :

SUB	R0,RS1,RS2
BHS	REL9

## CCR Effects

N	Z	V	C
—	—	—	—

N: Not affected.  
Z: Not affected.  
V: Not affected.  
C: Not affected.

## Code and CPU Cycles

Source Form	Address Mode	Machine Code	Cycles
BHS REL9	REL9	0 0 1 0 0 0 0 0      REL9	PP/P

BITH

Bit Test Immediate 8-Bit Constant  
(High Byte)

BITH

Operation

RD.H & IMM8 ⇒ NONE

Performs a bit wise logical AND between the high byte of register RD and an immediate 8-Bit constant. Only the condition code flags get updated, but no result is written back

CCR Effects

N	Z	V	C
Δ	Δ	0	–

- N: Set if bit 15 of the result is set; cleared otherwise.
- Z: Set if the 8-bit result is \$00; cleared otherwise.
- V: 0; cleared.
- C: Not affected.

Code and CPU Cycles

Source Form	Address Mode	Machine Code							Cycles
BITH RD, #IMM8	IMM8	1	0	0	1	1	RD	IMM8	P

# BITL

Bit Test Immediate 8-Bit Constant  
(Low Byte)

# BITL

### Operation

RD.L & IMM8  $\Rightarrow$  NONE

Performs a bit wise logical AND between the low byte of register RD and an immediate 8-Bit constant. Only the condition code flags get updated, but no result is written back.

### CCR Effects

N	Z	V	C
$\Delta$	$\Delta$	0	—

N: Set if bit 7 of the result is set; cleared otherwise.

Z: Set if the 8-bit result is \$00; cleared otherwise.

V: 0; cleared.

C: Not affected.

### Code and CPU Cycles

Source Form	Address Mode	Machine Code							Cycles
BITL RD, #IMM8	IMM8	1	0	0	1	0	RD	IMM8	P

BLE

Branch if Less or Equal to Zero

BLE

Operation

If  $Z \mid (N \wedge V) = 1$ , then  $PC + \$0002 + (REL9 \ll 1) \Rightarrow PC$

Branch instruction to compare signed numbers.  
Branch if  $RS1 \leq RS2$ :

SUBR0,RS1,RS2

BLEREL9

CCR Effects

N	Z	V	C
–	–	–	–

- N: Not affected.
- Z: Not affected.
- V: Not affected.
- C: Not affected.

Code and CPU Cycles

Source Form	Address Mode	Machine Code								Cycles
BLE REL9	REL9	0	0	1	1	1	0	1	REL9	PP/P

# BLO

Branch if Carry Set  
(Same as BCS)

# BLO

## Operation

If C = 1, then  $PC + \$0002 + (REL9 \ll 1) \Rightarrow PC$

Branch instruction to compare unsigned numbers.  
Branch if RS1 < RS2:

SUB	R0,RS1,RS2
BLO	REL9

## CCR Effects

N	Z	V	C
—	—	—	—

N: Not affected.  
Z: Not affected.  
V: Not affected.  
C: Not affected.

## Code and CPU Cycles

Source Form	Address Mode	Machine Code								Cycles
BCS REL9	REL9	0	0	1	0	0	0	1	REL9	PP/P



BLS

Branch if Lower or Same

BLS

Operation

If  $C \mid Z = 1$ , then  $PC + \$0002 + (REL9 << 1) \Rightarrow PC$

Branch instruction to compare unsigned numbers.  
Branch if  $RS1 \leq RS2$ :

SUB           R0,RS1,RS2

BLS           REL9

CCR Effects

N	Z	V	C
–	–	–	–

- N: Not affected.  
Z: Not affected.  
V: Not affected.  
C: Not affected.

Code and CPU Cycles

Source Form	Address Mode	Machine Code								Cycles
BLS REL9	REL9	0	0	1	1	0	0	1	REL9	PP/P

BLT

Branch if Lower than Zero

BLT

Operation

If  $N \wedge V = 1$ , then  $PC + \$0002 + (REL9 \ll 1) \Rightarrow PC$

Branch instruction to compare signed numbers.

Branch if  $RS1 < RS2$ :

SUBR0,RS1,RS2

BLTREL9

CCR Effects

N	Z	V	C
–	–	–	–

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

Code and CPU Cycles

Source Form	Address Mode	Machine Code								Cycles
BLT REL9	REL9	0	0	1	1	0	1	1	REL9	PP/P

BMI

Branch if Minus

BMI

Operation

If N = 1, then PC + \$0002 + (REL9 << 1) ⇒ PC

Tests the Sign flag and branches if N = 1.

CCR Effects

N	Z	V	C
—	—	—	—

- N: Not affected.
- Z: Not affected.
- V: Not affected.
- C: Not affected.

Code and CPU Cycles

Source Form	Address Mode	Machine Code								Cycles
BMI REL9	REL9	0	0	1	0	1	0	1	REL9	PP/P

# BNE

Branch if Not Equal

# BNE

## Operation

If  $Z = 0$ , then  $PC + \$0002 + (REL9 \ll 1) \Rightarrow PC$

Tests the Zero flag and branches if  $Z = 0$ .

## CCR Effects

N	Z	V	C
—	—	—	—

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

## Code and CPU Cycles

Source Form	Address Mode	Machine Code								Cycles
BNE REL9	REL9	0	0	1	0	0	1	0	REL9	PP/P

BPL

Branch if Plus

BPL

Operation

If N = 0, then PC + \$0002 + (REL9 << 1) ⇒ PC

Tests the Sign flag and branches if N = 0.

CCR Effects

N	Z	V	C
—	—	—	—

- N: Not affected.
- Z: Not affected.
- V: Not affected.
- C: Not affected.

Code and CPU Cycles

Source Form	Address Mode	Machine Code								Cycles
BPL REL9	REL9	0	0	1	0	1	0	0	REL9	PP/P

# BRA

Branch Always

# BRA

### Operation

$PC + \$0002 + (REL10 \ll 1) \Rightarrow PC$

Branches always

### CCR Effects

N	Z	V	C
—	—	—	—

N: Not affected.  
Z: Not affected.  
V: Not affected.  
C: Not affected.

### Code and CPU Cycles

Source Form	Address Mode	Machine Code							Cycles
BRA REL10	REL10	0	0	1	1	1	1	REL10	PP

BRK

Break

BRK

Operation

Put XGATE into Debug Mode (see 4.5.2) and signals a Software breakpoint to the S12X\_DBG module (see section 4.9 of the **S12X\_DBG Block User Guide**).

CCR Effects

N	Z	V	C
—	—	—	—

- N: Not affected.
- Z: Not affected.
- V: Not affected.
- C: Not affected.

Code and CPU Cycles

Source Form	Address Mode	Machine Code																Cycles
BRK	INH	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PAff

# BVC

Branch if Overflow Cleared

# BVC

## Operation

If  $V = 0$ , then  $PC + \$0002 + (REL9 \ll 1) \Rightarrow PC$

Tests the Overflow flag and branches if  $V = 0$ .

## CCR Effects

N	Z	V	C
—	—	—	—

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

## Code and CPU Cycles

Source Form	Address Mode	Machine Code								Cycles
BVC REL9	REL9	0	0	1	0	1	1	0	REL9	PP/P



# BVS

## Branch if Overflow Set

# BVS

### Operation

If  $V = 1$ , then  $PC + \$0002 + (REL9 \ll 1) \Rightarrow PC$

Tests the Overflow flag and branches if  $V = 1$ .

### CCR Effects

N	Z	V	C
—	—	—	—

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

### Code and CPU Cycles

Source Form	Address Mode	Machine Code								Cycles
BVS REL9	REL9	0	0	1	0	1	1	1	REL9	PP/P

# CMP

Compare

# CMP

### Operation

RS2 - RS1  $\Rightarrow$  NONE (translates to SUB R0, RS1, RS2)

Subtracts the content of register RS2 from the content of register RS1 using binary subtraction and discards the result.

### CCR Effects

N	Z	V	C
$\Delta$	$\Delta$	$\Delta$	$\Delta$

- N: Set if bit 15 of the result is set; cleared otherwise.  
Z: Set if the result is \$0000; cleared otherwise.  
V: Set if a two's complement overflow resulted from the operation; cleared otherwise.  
 $RS1[15] \& RS2[15] \& \overline{result[15]} \mid \overline{RS1[15]} \& RS2[15] \& result[15]$   
C: Set if there is a carry from the bit 15 of the result; cleared otherwise.  
 $\overline{RS1[15]} \& RS2[15] \mid \overline{RS1[15]} \& result[15] \mid RS2[15] \& result[15]$

### Code and CPU Cycles

Source Form	Address Mode	Machine Code										Cycles		
CMP RS1, RS2	TRI	0	0	0	1	1	0	0	0	RS1	RS2	0	0	P

CMPL

Compare Immediate 8-Bit Constant  
(Low Byte)

CMPL

Operation

RS.L - IMM8 ⇒ NONE, only condition code flags get updated

Subtracts the 8-Bit constant IMM8 contained in the instruction code from the low byte of the source register RS.L using binary subtraction and updates the condition code register accordingly.

Remark: There is no equivalent operation using triadic addressing. Comparing the values of two registers can be performed by using the subtract instruction with R0 as destination register.

CCR Effects

N	Z	V	C
Δ	Δ	Δ	Δ

- N: Set if bit 7 of the result is set; cleared otherwise.
- Z: Set if the 8-bit result is \$00; cleared otherwise.
- V: Set if a two’s complement overflow resulted from the 8-bit operation; cleared otherwise.  
 $RS[7] \& \overline{IMM8[7]} \& \overline{result[7]} \mid \overline{RS[7]} \& IMM8[7] \& result[7]$
- C: Set if there is a carry from the Bit 7 to Bit 8 of the result; cleared otherwise.  
 $\overline{RS[7]} \& IMM8[7] \mid \overline{RS[7]} \& result[7] \mid IMM8[7] \& result[7]$

Code and CPU Cycles

Source Form	Address Mode	Machine Code							Cycles
CMPL RS, #IMM8	IMM8	1	1	0	1	0	RS	IMM8	P

# COM

## One's Complement

# COM

### Operation

~RS ⇒ RD (translates to XNOR RD, R0, RS)  
~RD ⇒ RD (translates to XNOR RD, R0, RD)  
Performs a one's complement on a general purpose register.

### CCR Effects

N	Z	V	C
Δ	Δ	0	–

N: Set if bit 15 of the result is set; cleared otherwise.  
Z: Set if the result is \$0000; cleared otherwise.  
V: 0; cleared.  
C: Not affected.

### Code and CPU Cycles

Source Form	Address Mode	Machine Code										Cycles		
COM RD, RS	TRI	0	0	0	1	0	RD	0	0	0	RS	1	1	P
COM RD	TRI	0	0	0	1	0	RD	0	0	0	RD	1	1	P

CPC

Compare with Carry

CPC

Operation

RS2 - RS1 – X ⇒ NONE (translates to SBC R0, RS1, RS2)

Subtracts the carry bit and the content of register RS2 from the content of register RS1 using binary subtraction and discards the result.

CCR Effects

N	Z	V	C
Δ	Δ	Δ	Δ

- N: Set if bit 15 of the result is set; cleared otherwise.
- Z: Set if the result is \$0000; cleared otherwise.
- V: Set if a two’s complement overflow resulted from the operation; cleared otherwise.  
RS1[15] & RS2[15] & result[15] | RS1[15] & RS2[15] & result[15]
- C: Set if there is a carry from the bit 15 of the result; cleared otherwise.  
RS1[15] & RS2[15] | RS1[15] & result[15] | RS2[15] & result[15]

Code and CPU Cycles

Source Form	Address Mode	Machine Code										Cycles		
CMP RS1, RS2	TRI	0	0	0	1	1	0	0	0	RS1	RS2	0	1	P

CPCH

Compare Immediate 8-Bit Constant with  
Carry (High Byte)

CPCH

Operation

RS.H - IMM8 - C  $\Rightarrow$  NONE, only condition code flags get updated

Subtracts the carry bit and the 8-Bit constant IMM8 contained in the instruction code from the high byte of the source register RD using binary subtraction and updates the condition code register accordingly. The carry bit and Zero bits are taken into account to allow a 16-Bit compare in the form of

```
CMPL      R2, #LOWBYTE
CPCH      R2, #HIGHBYTE
BCC                               ; branch condition
```

Remark: There is no equivalent operation using triadic addressing. Comparing the values of two registers can be performed by using the subtract instruction with R0 as destination register.

CCR Effects

N	Z	V	C
$\Delta$	$\Delta$	$\Delta$	$\Delta$

- N: Set if bit 15 of the result is set; cleared otherwise.
- Z: Set if the result is \$00 and Z was set before this operation; cleared otherwise.
- V: Set if a two's complement overflow resulted from the operation; cleared otherwise.  
 $RS[15] \& IMM8[7] \& result[15] \mid \overline{RS[15]} \& IMM8[7] \& result[15]$
- C: Set if there is a carry from the bit 15 of the result; cleared otherwise.  
 $\overline{RS[15]} \& IMM8[7] \mid RS[15] \& result[15] \mid IMM8[7] \& result[15]$

Code and CPU Cycles

Source Form	Address Mode	Machine Code							Cycles
CPCH RD, #IMM8	IMM8	1	1	0	1	1	RS	IMM8	P

CSEM

Clear Semaphore

CSEM

Operation

Unlocks a semaphore that was locked by the RISC core.

In monadic address mode, bits RS[2:0] select the semaphore to be cleared.

CCR Effects

N	Z	V	C
—	—	—	—

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

Code and CPU Cycles

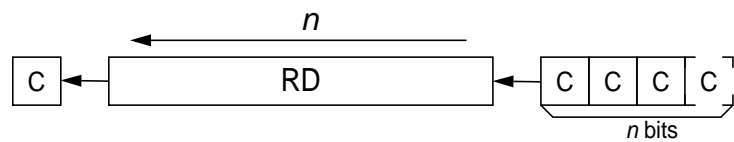
Source Form	Address Mode	Machine Code														Cycles
CSEM #IMM3	IMM3	0	0	0	0	0	IMM3	1	1	1	1	0	0	0	0	PA
CSEM RS	MON	0	0	0	0	0	RS	1	1	1	1	0	0	0	1	PA

CSL

Logical Shift Left with Carry

CSL

Operation



$n = \text{RS or IMM4}$

Shifts the bits in register RD  $n$  positions to the left. The lower  $n$  bits of the register RD become filled with the carry flag. The carry flag will be updated to the bit contained in RD[16- $n$ ] before the shift for  $n > 0$ .

$n$  can range from 0 to 16.

In immediate address mode,  $n$  is determined by the operand IMM4.  $n$  is considered to be 16 in IMM4 is equal to 0.

In dyadic address mode,  $n$  is determined by the content of RS.  $n$  is considered to be 16 if the content of RS is greater than 15.

CCR Effects

N	Z	V	C
Δ	Δ	Δ	Δ

N: Set if bit 15 of the result is set; cleared otherwise.

Z: Set if the result is \$0000; cleared otherwise.

V: Set if a two's complement overflow resulted from the operation; cleared otherwise.

$$\text{RD}[15]_{\text{old}} \wedge \text{RD}[15]_{\text{new}}$$

C: Set if  $n > 0$  and RD[16- $n$ ] = 1; if  $n = 0$  unaffected.

Code and CPU Cycles

Source Form	Address Mode	Machine Code												Cycles
CSL RD, #IMM4	IMM4	0	0	0	0	1	RD	IMM4		1	0	1	0	P
CSL RD, RS	DYA	0	0	0	0	1	RD	RS	1	0	0	1	0	P

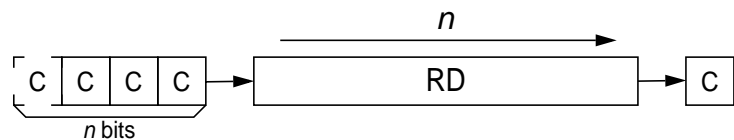


CSR

Logical Shift Right with Carry

CSR

Operation



$n = \text{RS or IMM4}$

Shifts the bits in register RD  $n$  positions to the right. The higher  $n$  bits of the register RD become filled with the carry flag. The carry flag will be updated to the bit contained in RD[n-1] before the shift for  $n > 0$ .

$n$  can range from 0 to 16.

In immediate address mode,  $n$  is determined by the operand IMM4.  $n$  is considered to be 16 in IMM4 is equal to 0.

In dyadic address mode,  $n$  is determined by the content of RS.  $n$  is considered to be 16 if the content of RS is greater than 15.

CCR Effects

N	Z	V	C
Δ	Δ	Δ	Δ

- N: Set if bit 15 of the result is set; cleared otherwise.
- Z: Set if the result is \$0000; cleared otherwise.
- V: Set if a two’s complement overflow resulted from the operation; cleared otherwise.
- $RD[15]_{old} \wedge RD[15]_{new}$
- C: Set if  $n > 0$  and  $RD[n-1] = 1$ ; if  $n = 0$  unaffected.

Code and CPU Cycles

Source Form	Address Mode	Machine Code											Cycles	
CSR RD, #IMM4	IMM4	0	0	0	0	1	RD	IMM4		1	0	1	1	P
CSR RD, RS	DYA	0	0	0	0	1	RD	RS	1	0	0	1	1	P

# JAL

## Jump and Link

# JAL

### Operation

$PC + \$0002 \Rightarrow RD; RD \Rightarrow PC$

Jumps to the address stored in RD and saves the return address in RD.

### CCR Effects

N	Z	V	C
—	—	—	—

- N: Not affected.  
Z: Not affected.  
V: Not affected.  
C: Not affected.

### Code and CPU Cycles

Source Form	Address Mode	Machine Code														Cycles
JAL RD	MON	0	0	0	0	0	RD	1	1	1	1	0	1	1	0	PP

# LDB

## Load Byte from Memory (Low Byte)

# LDB

### Operation

$M[RB, \#OFFS5] \Rightarrow RD.L; \$00 \Rightarrow RD.H$   
 $M[RB, RI] \Rightarrow RD.L; \$00 \Rightarrow RD.H$   
 $M[RB, RI] \Rightarrow RD.L; \$00 \Rightarrow RD.H; RI+1 \Rightarrow RI;^1$   
 $RI-1 \Rightarrow RI; M[RS, RI] \Rightarrow RD.L; \$00 \Rightarrow RD.H$

Loads a byte from memory into the low byte of register RD. The high byte is cleared.

1

### CCR Effects

N	Z	V	C
–	–	–	–

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

### Code and CPU Cycles

Source Form	Address Mode	Machine Code								Cycles
LDB RD, (RB, #OFFS5)	IDO5	0	1	0	0	0	RD	RB	OFFS5	Pr
LDB RD, (RS, RI)	IDR	0	1	1	0	0	RD	RB	RI 0 0	Pr
LDB RD, (RS, RI+)	IDR+	0	1	1	0	0	RD	RB	RI 0 1	Pr
LDB RD, (RS, -RI)	-IDR	0	1	1	0	0	RD	RB	RI 1 0	Pr

#### NOTES:

1. If the same general purpose register is used as index (RI) and destination register (RD), the content of the register will not be incremented after the data move:  $M[RB, RI] \Rightarrow RD.L; \$00 \Rightarrow RD.H$

# LDH

## Load Immediate 8-Bit Constant (High Byte)

# LDH

**Operation**

IMM8 ⇒ RD.H;

Loads an eight bit immediate constant into the high byte of register RD. The low byte is not affected.

**CCR Effects**

N	Z	V	C
—	—	—	—

N: Not affected.  
Z: Not affected.  
V: Not affected.  
C: Not affected.

**Code and CPU Cycles**

Source Form	Address Mode	Machine Code							Cycles
LDH RD, #IMM8	IMM8	1	1	1	1	1	RD	IMM8	P

# LDL

## Load Immediate 8-Bit Constant (Low Byte)

# LDL

### Operation

IMM8  $\Rightarrow$  RD.L; \$00  $\Rightarrow$  RD.H

Loads an eight bit immediate constant into the low byte of register RD. The high byte is cleared.

### CCR Effects

N	Z	V	C
–	–	–	–

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

### Code and CPU Cycles

Source Form	Address Mode	Machine Code						Cycles	
LDL RD, #IMM8	IMM8	1	1	1	1	0	RD	IMM8	P

# LDW

## Load Word from Memory

# LDW

### Operation

$M[RB, \#OFFS5] \Rightarrow RD;$   
 $M[RB, RI] \Rightarrow RD;$   
 $M[RB, RI] \Rightarrow RD; RI+2 \Rightarrow RI;^1$   
 $RI-2 \Rightarrow RI; M[RS, RI] \Rightarrow RD;$

Loads a word from memory into the register RD.

### CCR Effects

N	Z	V	C
—	—	—	—

N: Not affected.  
Z: Not affected.  
V: Not affected.  
C: Not affected.

### Code and CPU Cycles

Source Form	Address Mode	Machine Code								Cycles
LDW RD, (RB, #OFFS5)	IDO5	0	1	0	0	1	RD	RB	OFFS5	PR
LDW RD, (RB, RI)	IDR	0	1	1	0	1	RD	RB	RI 0 0	PR
LDW RD, (RB, RI+)	IDR+	0	1	1	0	1	RD	RB	RI 0 1	PR
LDW RD, (RB, -RI)	-IDR	0	1	1	0	1	RD	RB	RI 1 0	PR

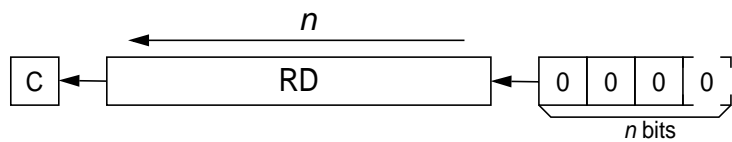
NOTES:  
1. If the same general purpose register is used as index (RI) and destination register (RD), the content of the register will not be incremented after the data move:  $M[RB, RI] \Rightarrow RD$

LSL

Logical Shift Left

LSL

Operation



$n = \text{RS or IMM4}$

Shifts the bits in register RD  $n$  positions to the left. The lower  $n$  bits of the register RD become filled with zeros. The carry flag will be updated to the bit contained in RD[16- $n$ ] before the shift for  $n > 0$ .  
 $n$  can range from 0 to 16.  
In immediate address mode,  $n$  is determined by the operand IMM4.  $n$  is considered to be 16 in IMM4 is equal to 0.  
In dyadic address mode,  $n$  is determined by the content of RS.  $n$  is considered to be 16 if the content of RS is greater than 15.

CCR Effects

N	Z	V	C
Δ	Δ	Δ	Δ

- N: Set if bit 15 of the result is set; cleared otherwise.  
Z: Set if the result is \$0000; cleared otherwise.  
V: Set if a two’s complement overflow resulted from the operation; cleared otherwise.  
 $\text{RD}[15]_{\text{old}} \wedge \text{RD}[15]_{\text{new}}$   
C: Set if  $n > 0$  and  $\text{RD}[16-n] = 1$ ; if  $n = 0$  unaffected.

Code and CPU Cycles

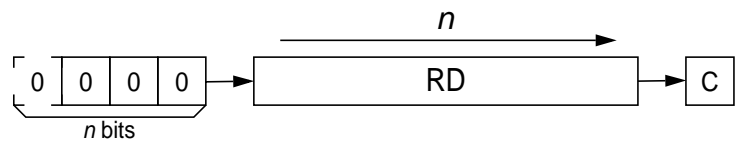
Source Form	Address Mode	Machine Code										Cycles		
LSL RD, #IMM4	IMM4	0	0	0	0	1	RD	IMM4		1	1	0	0	P
LSL RD, RS	DYA	0	0	0	0	1	RD	RS	1	0	1	0	0	P

LSR

Logical Shift Right

LSR

Operation



$n = \text{RS or IMM4}$

Shifts the bits in register RD  $n$  positions to the right. The higher  $n$  bits of the register RD become filled with zeros. The carry flag will be updated to the bit contained in RD[n-1] before the shift for  $n > 0$ .

$n$  can range from 0 to 16.

In immediate address mode,  $n$  is determined by the operand IMM4.  $n$  is considered to be 16 in IMM4 is equal to 0.

In dyadic address mode,  $n$  is determined by the content of RS.  $n$  is considered to be 16 if the content of RS is greater than 15.

CCR Effects

N	Z	V	C
Δ	Δ	Δ	Δ

N: Set if bit 15 of the result is set; cleared otherwise.

Z: Set if the result is \$0000; cleared otherwise.

V: Set if a two's complement overflow resulted from the operation; cleared otherwise.

$$\text{RD}[15]_{\text{old}} \wedge \text{RD}[15]_{\text{new}}$$

C: Set if  $n > 0$  and RD[n-1] = 1; if  $n = 0$  unaffected.

Code and CPU Cycles

Source Form	Address Mode	Machine Code											Cycles	
LSR RD, #IMM4	IMM4	0	0	0	0	1	RD	IMM4		1	1	0	1	P
LSR RD, RS	DYA	0	0	0	0	1	RD	RS	1	0	1	0	1	P



# MOV

## Move Register Content

# MOV

### Operation

RS  $\Rightarrow$  RD (translates to OR RD, R0, RS)

Copies the content of RS to RD.

### CCR Effects

N	Z	V	C
$\Delta$	$\Delta$	0	—

N: Set if bit 15 of the result is set; cleared otherwise.

Z: Set if the result is \$0000; cleared otherwise.

V: 0; cleared.

C: Not affected.

### Code and CPU Cycles

Source Form	Address Mode	Machine Code										Cycles		
MOV RD, RS	TRI	0	0	0	1	0	RD	0	0	0	RS	1	0	P

# NEG

## Two's Complement

# NEG

### Operation

-RS  $\Rightarrow$  RD (translates to SUB RD, R0, RS)  
-RD  $\Rightarrow$  RD (translates to SUB RD, R0, RD)

Performs a two's complement on a general purpose register.

### CCR Effects

N	Z	V	C
$\Delta$	$\Delta$	$\Delta$	$\Delta$

- N: Set if bit 15 of the result is set; cleared otherwise.  
Z: Set if the result is \$0000; cleared otherwise.  
V: Set if a two's complement overflow resulted from the operation; cleared otherwise.  
     $RS[15] \& RD[15]_{new}$   
C: Set if there is a carry from the bit 15 of the result; cleared otherwise  
     $RS[15] \mid RD[15]_{new}$

### Code and CPU Cycles

Source Form	Address Mode	Machine Code										Cycles		
NEG RD, RS	TRI	0	0	0	1	1	RD	0	0	0	RS	0	0	P
NEG RD	TRI	0	0	0	1	1	RD	0	0	0	RD	0	0	P

# NOP

No Operation

# NOP

Operation

No Operation for one cycle.

CCR Effects

N	Z	V	C
–	–	–	–

- N: Not affected.
- Z: Not affected.
- V: Not affected.
- C: Not affected.

Code and CPU Cycles

Source Form	Address Mode	Machine Code	Cycles
NOP	INH	0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0	P

OR

Logical OR

OR

Operation

RS1 | RS2 ⇒ RD

Performs a bit wise logical OR between the content of register RS1 and the content of register RS2 and stores the result in the destination register RD.

CCR Effects

N	Z	V	C
Δ	Δ	0	–

- N: Set if bit 15 of the result is set; cleared otherwise.
- Z: Set if the result is \$0000; cleared otherwise.
- V: 0; cleared.
- C: Not affected.

Code and CPU Cycles

Source Form	Address Mode	Machine Code								Cycles
OR RD, RS1, RS2	TRI	0	0	0	1	0	RD	RS1	RS2	1 0 P

# ORH

## Logical OR Immediate 8-Bit Constant (High Byte)

# ORH

### Operation

RD.H | IMM8  $\Rightarrow$  RD.H

Performs a bit wise logical OR between the high byte of register RD and an immediate 8-Bit constant and stores the result in the destination register RD.H. The low byte of RD is not affected.

### CCR Effects

N	Z	V	C
$\Delta$	$\Delta$	0	—

N: Set if bit 15 of the result is set; cleared otherwise.

Z: Set if the 8-Bit result is \$00; cleared otherwise.

V: 0; cleared.

C: Not affected.

### Code and CPU Cycles

Source Form	Address Mode	Machine Code							Cycles
ORH RD, #IMM8	IMM8	1	0	1	0	1	RD	IMM8	P

# ORL

Logical OR Immediate 8-Bit Constant  
(Low Byte)

# ORL

## Operation

RD.L | IMM8  $\Rightarrow$  RD.L

Performs a bit wise logical OR between the low byte of register RD and an immediate 8-Bit constant and stores the result in the destination register RD.L. The high byte of RD is not affected.

## CCR Effects

N	Z	V	C
$\Delta$	$\Delta$	0	—

N: Set if bit 7 of the result is set; cleared otherwise.

Z: Set if the 8-Bit result is \$00; cleared otherwise.

V: 0; cleared.

C: Not affected.

## Code and CPU Cycles

Source Form	Address Mode	Machine Code							Cycles
ORL RD, #IMM8	IMM8	1	0	1	0	0	RD	IMM8	P

PAR

Calculate Parity

PAR

Operation

Calculates the number of ones in the register RD. The Carry flag will be set if the number is odd, otherwise it will be cleared.

CCR Effects

N	Z	V	C
0	$\Delta$	0	$\Delta$

- N: 0; cleared.
- Z: Set if RD is \$0000; cleared otherwise.
- V: 0; cleared.
- C: Set if there the number of ones in the register RD is odd; cleared otherwise.

Code and CPU Cycles

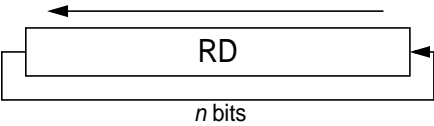
Source Form	Address Mode	Machine Code														Cycles
PAR, RD	MON	0	0	0	0	0	RD	1	1	1	1	0	1	0	1	P

ROL

Rotate Left

ROL

Operation



$n = \text{RS or IMM4}$

Rotates the bits in register RD  $n$  positions to the left. The lower  $n$  bits of the register RD are filled with the upper  $n$  bits. Two source forms are available. In the first form, the parameter  $n$  is contained in the instruction code as an immediate operand. In the second form, the parameter is contained in the lower bits of the source register RS[3:0]. All other bits in RS are ignored. If  $n$  is zero, no shift will take place and the register RD will be unaffected; however, the condition code flags will be updated.

CCR Effects

N	Z	V	C
Δ	Δ	0	–

- N: Set if bit 15 of the result is set; cleared otherwise.
- Z: Set if the result is \$0000; cleared otherwise.
- V: 0; cleared.
- C: Not affected.

Code and CPU Cycles

Source Form	Address Mode	Machine Code										Cycles		
ROL RD, #IMM4	IMM4	0	0	0	0	1	RD	IMM4		1	1	1	0	P
ROL RD, RS	DYA	0	0	0	0	1	RD	RS	1	0	1	1	0	P

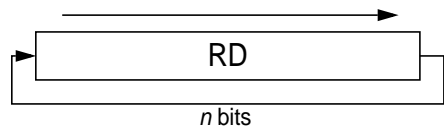


ROR

Rotate Right

ROR

Operation



$n = \text{RS or IMM4}$

Rotates the bits in register RD  $n$  positions to the right. The upper  $n$  bits of the register RD are filled with the lower  $n$  bits. Two source forms are available. In the first form, the parameter  $n$  is contained in the instruction code as an immediate operand. In the second form, the parameter is contained in the lower bits of the source register RS[3:0]. All other bits in RS are ignored. If  $n$  is zero no shift will take place and the register RD will be unaffected; however, the condition code flags will be updated.

CCR Effects

N	Z	V	C
$\Delta$	$\Delta$	0	—

- N: Set if bit 15 of the result is set; cleared otherwise.
- Z: Set if the result is \$0000; cleared otherwise.
- V: 0; cleared.
- C: Not affected.

Code and CPU Cycles

Source Form	Address Mode	Machine Code										Cycles		
ROR RD, #IMM4	IMM4	0	0	0	0	1	RD	IMM4		1	1	1	1	P
ROR RD, RS	DYA	0	0	0	0	1	RD	RS	1	0	1	1	1	P

# RTS

Return to Scheduler

# RTS

### Operation

Terminates the current thread of program execution and remains idle until a new thread is started by the hardware scheduler.

### CCR Effects

N	Z	V	C
—	—	—	—

- N: Not affected.
- Z: Not affected.
- V: Not affected.
- C: Not affected.

### Code and CPU Cycles

Source Form	Address Mode	Machine Code																Cycles	
RTS	INH	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	PA

# SBC

## Subtract with Carry

# SBC

### Operation

$$RS1 - RS2 - C \Rightarrow RD$$

Subtracts the content of register RS2 and the value of the Carry bit from the content of register RS1 using binary subtraction and stores the result in the destination register RD. Also the zero flag is carried forward from the previous operation allowing 32 and more bit subtractions.

Example:

```

SBC      R6,R4,R2
SBC      R7,R5,R3 ; R7:R6 = R5:R4 - R3:R2
BCC                      ; conditional branch on 32 bit addition
  
```

### CCR Effects

N	Z	V	C
Δ	Δ	Δ	Δ

N: Set if bit 15 of the result is set; cleared otherwise.

Z: Set if the result is \$0000 and Z was set before this operation; cleared otherwise.

V: Set if a two's complement overflow resulted from the operation; cleared otherwise.

$$RS1[15] \& \overline{RS2[15]} \& \overline{RD[15]}_{new} \mid \overline{RS1[15]} \& RS2[15] \& RD[15]_{new}$$

C: Set if there is a carry from bit 15 of the result; cleared otherwise.

$$\overline{RS1[15]} \& RS2[15] \mid \overline{RS1[15]} \& RD[15]_{new} \mid RS2[15] \& RD[15]_{new}$$

### Code and CPU Cycles

Source Form	Address Mode	Machine Code										Cycles
SBC RD, RS1, RS2	TRI	0	0	0	1	1	RD	RS1	RS2	0	1	P

SSEM

Set Semaphore

SSEM

Operation

Attempts to set a semaphore. The state of the semaphore will be stored in the Carry-Flag:

- 1 = Semaphore is locked by the RISC core
- 0 = Semaphore is locked by the S12X\_CPU

In monadic address mode, bits RS[2:0] select the semaphore to be set.

CCR Effects

N	Z	V	C
–	–	–	Δ

- N: Not affected.
- Z: Not affected.
- V: Not affected.
- C: Set if semaphore is locked by the RISC core; cleared otherwise.

Code and CPU Cycles

Source Form	Address Mode	Machine Code														Cycles	
SSEM #IMM3	IMM3	0	0	0	0	0	0	IMM3	1	1	1	1	0	0	1	0	PA
SSEM RS	MON	0	0	0	0	0	0	RS	1	1	1	1	0	0	1	1	PA

SEX

Sign Extend Byte to Word

SEX

Operation

The result in RD is the 16-bit sign extended representation of the original two's complement number in the low byte of RD.L.

CCR Effects

N	Z	V	C
Δ	Δ	0	–

- N: Set if bit 15 of the result is set; cleared otherwise.
- Z: Set if the result is \$0000; cleared otherwise.
- V: 0; cleared.
- C: Not affected.

Code and CPU Cycles

Source Form	Address Mode	Machine Code														Cycles
SEX RD	MON	0	0	0	0	0	RD	1	1	1	1	0	1	0	0	P

SIF

Set Interrupt Flag

SIF

Operation

Sets the Interrupt Flag of an XGATE Channel. This instruction supports two source forms. If inherent address mode is used, then the interrupt flag of the current channel (XGCHID) will be set. If the monadic address form is used, the interrupt flag associated with the channel id number contained in RS[6:0] is set. The content of RS[15:7] is ignored.

CCR Effects

N	Z	V	C
—	—	—	—

- N: Not affected.
- Z: Not affected.
- V: Not affected.
- C: Not affected.

Code and CPU Cycles

Source Form	Address Mode	Machine Code																Cycles
SIF	INH	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	PA
SIF RS	MON	0	0	0	0	0	RS			1	1	1	1	0	1	1	1	PA

STB

Store Byte to Memory  
(Low Byte)

STB

Operation

RS.L           ⇒ M[RB, #OFFS5]  
RS.L           ⇒ M[RB, RI]  
RS.L           ⇒ M[RB, RI]; RI+1 ⇒ RI;  
RI-1 ⇒ RI; RS.L ⇒ M[RB, RI]<sup>1</sup>

Stores the low byte of register RD to memory.

CCR Effects

N	Z	V	C
—	—	—	—

N: Not affected.  
Z: Not affected.  
V: Not affected.  
C: Not affected.

Code and CPU Cycles

Source Form	Address Mode	Machine Code								Cycles
STB RS, (RB, #OFFS5),	IDO5	0	1	0	1	0	RS	RB	OFFS5	Pw
STB RS, (RB, RI)	IDR	0	1	1	1	0	RS	RB	RI 0 0	Pw
STB RS, (RB, RI+)	IDR+	0	1	1	1	0	RS	RB	RI 0 1	Pw
STB RS, (RB, -RI)	-IDR	0	1	1	1	0	RS	RB	RI 1 0	Pw

NOTES:  
1. If the same general purpose register is used as index (RI) and source register (RS), the unmodified content of the source register is written to the memory: RS.L ⇒ M[RB, RS-1]; RS-1 ⇒ RS

# STW

## Store Word to Memory

# STW

### Operation

RS  $\Rightarrow$  M[RB, #OFFS5]  
RS  $\Rightarrow$  M[RB, RI]  
RS  $\Rightarrow$  M[RB, RI]; RI+2  $\Rightarrow$  RI;  
RI-2  $\Rightarrow$  RI; RS  $\Rightarrow$  M[RB, RI]<sup>1</sup>

Stores the content of register RD to memory.

### CCR Effects

N	Z	V	C
—	—	—	—

N: Not affected.  
Z: Not affected.  
V: Not affected.  
C: Not affected.

### Code and CPU Cycles

Source Form	Address Mode	Machine Code						Cycles		
STW RS, (RB, #OFFS5)	IDO5	0	1	0	1	1	RS	RB	OFFS5	PW
STW RS, (RB, RI)	IDR	0	1	1	1	1	RS	RB	RI	0 0 PW
STW RS, (RB, RI+)	IDR+	0	1	1	1	1	RS	RB	RI	0 1 PW
STW RS, (RB, -RI)	-IDR	0	1	1	1	1	RS	RB	RI	1 0 PW

NOTES:  
1. If the same general purpose register is used as index (RI) and source register (RS), the unmodified content of the source register is written to the memory: RS  $\Rightarrow$  M[RB, RS-2]; RS-2  $\Rightarrow$  RS



# SUB

Subtract without Carry

# SUB

## Operation

$RS1 - RS2 \Rightarrow RD$

Subtracts the content of register RS2 from the content of register RS1 using binary subtraction and stores the result in the destination register RD.

## CCR Effects

**N    Z    V    C**

$\Delta$	$\Delta$	$\Delta$	$\Delta$
----------	----------	----------	----------

N: Set if bit 15 of the result is set; cleared otherwise.

Z: Set if the result is \$0000; cleared otherwise.

V: Set if a two's complement overflow resulted from the operation; cleared otherwise.

$RS1[15] \& RS2[15] \& RD[15]_{new} \mid \overline{RS1[15]} \& RS2[15] \& RD[15]_{new}$

C: Set if there is a carry from the bit 15 of the result; cleared otherwise.

$\overline{RS1[15]} \& RS2[15] \mid \overline{RS1[15]} \& RD[15]_{new} \mid RS2[15] \& RD[15]_{new}$

## Code and CPU Cycles

Source Form	Address Mode	Machine Code										Cycles
SUB RD, RS1, RS2	TRI	0	0	0	1	1	RD	RS1	RS2	0	0	P

SUBH

Subtract Immediate 8-Bit Constant  
(High Byte)

SUBH

Operation

RD - IMM8:\$00 ⇒ RD

Subtracts a signed immediate 8-Bit constant from the content of high byte of register RD and using binary subtraction and stores the result in the high byte of destination register RD. This instruction can be used after an SUBL for a 16-bit immediate subtraction.

EXAMPLE :

```
SUBL    R2,#LOWBYTE
SUBH    R2,#HIGHBYTE; R2 = R2 - 16 Bit immediate
```

CCR Effects

N	Z	V	C
Δ	Δ	Δ	Δ

- N: Set if bit 15 of the result is set; cleared otherwise.  
Z: Set if the result is \$0000; cleared otherwise.  
V: Set if a two’s complement overflow resulted from the operation; cleared otherwise.  
 $RD[15]_{old} \& \overline{IMM8[7]} \& RD[15]_{new} \mid \overline{RD[15]_{old}} \& IMM8[7] \& RD[15]_{new}$   
C: Set if there is a carry from the bit 15 of the result; cleared otherwise.  
 $\overline{RD[15]_{old}} \& IMM8[7] \mid \overline{RD[15]_{old}} \& RD[15]_{new} \mid IMM8[7] \& RD[15]_{new}$

Code and CPU Cycles

Source Form	Address Mode	Machine Code						Cycles
SUBH RD, #IMM8	IMM8	1	1	0	0	1	RD      IMM8	P

SUBL

Subtract Immediate 8-Bit Constant  
(Low Byte)

SUBL

Operation

$RD - \$00:IMM8 \Rightarrow RD$

Subtracts an immediate 8 Bit constant from the content of register RD using binary subtraction and stores the result in the destination register RD.

CCR Effects

N	Z	V	C
Δ	Δ	Δ	Δ

- N: Set if bit 15 of the result is set; cleared otherwise.
- Z: Set if the result is \$0000; cleared otherwise.
- V: Set if a two’s complement overflow resulted from the 8-bit operation; cleared otherwise.
- $RD[15]_{old} \& \overline{RD[15]_{new}}$
- C: Set if there is a carry from the bit 7 to bit 8 of the result; cleared otherwise.
- $\overline{RD[15]_{old}} \& RD[15]_{new}$

Code and CPU Cycles

Source Form	Address Mode	Machine Code							Cycles
SUBL RD, #IMM8	IMM8	1	1	0	0	0	RD	IMM8	P

TFR

Transfer from and to Special Registers

TFR<sup>1</sup>

Operation

Transfers the content of one RISC core register to another.  
If the content of the Condition Code Register is transferred to a General Purpose Register (RD), bits RD[15:4] become cleared.  
If the content of the General Purpose Register (RS) is transferred to the Condition Code Register, only bits RS[3:0] are copied.

CCR Effects

TFR RD,CCR:

N	Z	V	C
–	–	–	–

N: Not affected.  
Z: Not affected.  
V: Not affected.  
C: Not affected.

TFR CCR,RS:

N	Z	V	C
Δ	Δ	Δ	Δ

N: RS[3].  
Z: RS[2].  
V: RS[1].  
C: RS[0].

Code and CPU Cycles

Source Form		Address Mode	Machine Code														Cycles
TFR RD,CCR	CCR ⇒ RD	MON	0	0	0	0	0	RD	1	1	1	1	1	0	0	0	P
TFR CCR,RS	RS ⇒ CCR	MON	0	0	0	0	0	RS	1	1	1	1	1	0	0	1	P

TST

Test Register

TST

Operation

RS - 0  $\Rightarrow$  NONE (translates to SUB R0, RS, R0)

Subtracts zero from the content of register RS using binary subtraction and discards the result.

CCR Effects

N	Z	V	C
$\Delta$	$\Delta$	$\Delta$	$\Delta$

- N: Set if bit 15 of the result is set; cleared otherwise.
- Z: Set if the result is \$0000; cleared otherwise.
- V: Set if a two’s complement overflow resulted from the operation; cleared otherwise.  
RS[15] & result[15]
- C: Set if there is a carry from the bit 15 of the result; cleared otherwise.  
RS1[15] & result[15]

Code and CPU Cycles

Source Form	Address Mode	Machine Code										Cycles				
TST RS	TRI	0	0	0	1	1	0	0	0	RS1	0	0	0	0	0	P

XNOR

Logical Exclusive NOR

XNOR

Operation

$\sim(RS1 \wedge RS2) \Rightarrow RD$

Performs a bit wise logical exclusive NOR between the content of register RS1 and the content of register RS2 and stores the result in the destination register RD.

Remark: Using R0 as a source registers will calculate the one's complement of the other source register. Using R0 as both source operands will fill RD with \$FFFF.

CCR Effects

N	Z	V	C
Δ	Δ	0	–

- N: Set if bit 15 of the result is set; cleared otherwise.
- Z: Set if the result is \$0000; cleared otherwise.
- V: 0; cleared.
- C: Not affected.

Code and CPU Cycles

Source Form	Address Mode	Machine Code						Cycles		
XNOR RD, RS1, RS2	TRI	0	0	0	1	0	RD	RS1	RS2	1 1 P

XNORH

Logical Exclusive NOR Immediate 8-Bit Constant (High Byte)

XNORH

Operation

$\sim(RD.H \wedge IMM8) \Rightarrow RD.H$

Performs a bit wise logical exclusive NOR between the high byte of register RD and an immediate 8-Bit constant and stores the result in the destination register RD.H. The low byte of RD is not affected.

CCR Effects

N	Z	V	C
Δ	Δ	0	–

- N: Set if bit 15 of the result is set; cleared otherwise.
- Z: Set if the 8-bit result is \$00; cleared otherwise.
- V: 0; cleared.
- C: Not affected.

Code and CPU Cycles

Source Form	Address Mode	Machine Code							Cycles
XNORH RD, #IMM8	IMM8	1	0	1	1	1	RD	IMM8	P

XNORL

Logical Exclusive NOR Immediate 8-Bit  
Constant (Low Byte)

XNORL

Operation

$\sim(RD.L \wedge IMM8) \Rightarrow RD.L$

Performs a bit wise logical exclusive NOR between the low byte of register RD and an immediate 8-Bit constant and stores the result in the destination register RD.L. The high byte of RD is not affected.

CCR Effects

N	Z	V	C
Δ	Δ	0	–

- N: Set if bit 7 of the result is set; cleared otherwise.
- Z: Set if the 8-bit result is \$00; cleared otherwise.
- V: 0; cleared.
- C: Not affected.

Code and CPU Cycles

Source Form	Address Mode	Machine Code							Cycles
XNORL RD, #IMM8	IMM8	1	0	1	1	0	RD	IMM8	P



## 4.7.6 Instruction Coding

The following table summarizes all XGATE instructions in the order of their machine coding.

**Table 4-2 Instruction Set Summary**

Functionality	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Return to Scheduler and others																
BRK	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
NOP	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
RTS	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
SIF	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
Semaphore Instructions																
CSEM IMM3	0	0	0	0	0	IMM3		1	1	1	1	0	0	0	0	0
CSEM RS	0	0	0	0	0	RS		1	1	1	1	0	0	0	0	1
SSEM IMM3	0	0	0	0	0	IMM3		1	1	1	1	0	0	1	0	0
SSEM RS	0	0	0	0	0	RS		1	1	1	1	0	0	1	1	1
Single Register Instructions																
SEX RD	0	0	0	0	0	RD		1	1	1	1	0	1	0	0	0
PAR RD	0	0	0	0	0	RD		1	1	1	1	0	1	0	0	1
JAL RD	0	0	0	0	0	RD		1	1	1	1	0	1	1	0	0
SIF RS	0	0	0	0	0	RS		1	1	1	1	0	1	1	1	1
Special Move instructions																
TFR RD,CCR	0	0	0	0	0	RD		1	1	1	1	1	0	0	0	0
TFR CCR,RS	0	0	0	0	0	RS		1	1	1	1	1	0	0	0	1
Shift instructions dyadic																
BFFO RD, RS	0	0	0	0	1	RD		RS		1	0	0	0	0	0	0
ASR RD, RS	0	0	0	0	1	RD		RS		1	0	0	0	0	1	0
CSL RD, RS	0	0	0	0	1	RD		RS		1	0	0	1	0	0	0
CSR RD, RS	0	0	0	0	1	RD		RS		1	0	0	1	1	0	0
LSL RD, RS	0	0	0	0	1	RD		RS		1	0	1	0	0	0	0
LSR RD, RS	0	0	0	0	1	RD		RS		1	0	1	0	1	0	0
ROL RD, RS	0	0	0	0	1	RD		RS		1	0	1	1	0	0	0
ROR RD, RS	0	0	0	0	1	RD		RS		1	0	1	1	1	0	0
Shift instructions immediate																
ASR RD, #IMM4	0	0	0	0	1	RD		IMM4		1	0	0	0	1	0	0
CSL RD, #IMM4	0	0	0	0	1	RD		IMM4		1	0	1	0	0	0	0
CSR RD, #IMM4	0	0	0	0	1	RD		IMM4		1	0	1	1	0	0	0
LSL RD, #IMM4	0	0	0	0	1	RD		IMM4		1	1	0	0	0	0	0
LSR RD, #IMM4	0	0	0	0	1	RD		IMM4		1	1	0	1	0	0	0
ROL RD, #IMM4	0	0	0	0	1	RD		IMM4		1	1	1	0	0	0	0
ROR RD, #IMM4	0	0	0	0	1	RD		IMM4		1	1	1	1	0	0	0
Logical triadic																
AND RD, RS1, RS2	0	0	0	1	0	RD		RS1		RS2		0		0	0	0
OR RD, RS1, RS2	0	0	0	1	0	RD		RS1		RS2		1		0	0	0
XNOR RD, RS1, RS2	0	0	0	1	0	RD		RS1		RS2		1		1	0	0
Arithmetic triadic	For compare use SUB R0,Rs1,Rs2															
SUB RD, RS1, RS2	0	0	0	1	1	RD		RS1		RS2		0		0	0	0
SBC RD, RS1, RS2	0	0	0	1	1	RD		RS1		RS2		0		1	0	0
ADD RD, RS1, RS2	0	0	0	1	1	RD		RS1		RS2		1		0	0	0
ADC RD, RS1, RS2	0	0	0	1	1	RD		RS1		RS2		1		1	0	0
Branches																
BCC REL9	0	0	1	0	0	0	0	REL9								
BCS REL9	0	0	1	0	0	0	1	REL9								

Table 4-2 Instruction Set Summary

Functionality	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BNE REL9	0	0	1	0	0	1	0	REL9								
BEQ REL9	0	0	1	0	0	1	1	REL9								
BPL REL9	0	0	1	0	1	0	0	REL9								
BMI REL9	0	0	1	0	1	0	1	REL9								
BVC REL9	0	0	1	0	1	1	0	REL9								
BVS REL9	0	0	1	0	1	1	1	REL9								
BHI REL9	0	0	1	1	0	0	0	REL9								
BLS REL9	0	0	1	1	0	0	1	REL9								
BGE REL9	0	0	1	1	0	1	0	REL9								
BLT REL9	0	0	1	1	0	1	1	REL9								
BGT REL9	0	0	1	1	1	0	0	REL9								
BLE REL9	0	0	1	1	1	0	1	REL9								
BRA REL10	0	0	1	1	1	1	REL10									
Load & Store Instructions																
LDB RD, (RB, #OFFS5)	0	1	0	0	0	RD		RB		OFFS5						
LDW RD, (RB, #OFFS5)	0	1	0	0	1	RD		RB		OFFS5						
STB RS, (RB, #OFFS5)	0	1	0	1	0	RS		RB		OFFS5						
STW RS, (RB, #OFFS5)	0	1	0	1	1	RS		RB		OFFS5						
LDB RD, (RB, RI)	0	1	1	0	0	RD		RB		RI		0	0			
LDW RD, (RB, RI)	0	1	1	0	1	RD		RB		RI		0	0			
STB RS, (RB, RI)	0	1	1	1	0	RS		RB		RI		0	0			
STW RS, (RB, RI)	0	1	1	1	1	RS		RB		RI		0	0			
LDB RD, (RB, RI+)	0	1	1	0	0	RD		RB		RI		0	1			
LDW RD, (RB, RI+)	0	1	1	0	1	RD		RB		RI		0	1			
STB RS, (RB, RI+)	0	1	1	1	0	RS		RB		RI		0	1			
STW RS, (RB, RI+)	0	1	1	1	1	RS		RB		RI		0	1			
LDB RD, (RB, -RI)	0	1	1	0	0	RD		RB		RI		1	0			
LDW RD, (RB, -RI)	0	1	1	0	1	RD		RB		RI		1	0			
STB RS, (RB, -RI)	0	1	1	1	0	RS		RB		RI		1	0			
STW RS, (RB, -RI)	0	1	1	1	1	RS		RB		RI		1	0			
Bit Field Instructions																
BFEXT RD, RS1, RS2	0	1	1	0	0	RD		RS1		RS2		1	1			
BFINS RD, RS1, RS2	0	1	1	0	1	RD		RS1		RS2		1	1			
BFINSI RD, RS1, RS2	0	1	1	1	0	RD		RS1		RS2		1	1			
BFINSX RD, RS1, RS2	0	1	1	1	1	RD		RS1		RS2		1	1			
Logic Immediate Instructions																
ANDL RD, #IMM8	1	0	0	0	0	RD		IMM8								
ANDH RD, #IMM8	1	0	0	0	1	RD		IMM8								
BITL RD, #IMM8	1	0	0	1	0	RD		IMM8								
BITH RD, #IMM8	1	0	0	1	1	RD		IMM8								
ORL RD, #IMM8	1	0	1	0	0	RD		IMM8								
ORH RD, #IMM8	1	0	1	0	1	RD		IMM8								
XNORL RD, #IMM8	1	0	1	1	0	RD		IMM8								
XNORH RD, #IMM8	1	0	1	1	1	RD		IMM8								
Arithmetic Immediate Instructions																
SUBL RD, #IMM8	1	1	0	0	0	RD		IMM8								
SUBH RD, #IMM8	1	1	0	0	1	RD		IMM8								
CMPL RS, #IMM8	1	1	0	1	0	RS		IMM8								

**Table 4-2 Instruction Set Summary**

Functionality	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CPCH RS, #IMM8	1	1	0	1	1	RS			IMM8							
ADDL RD, #IMM8	1	1	1	0	0	RD			IMM8							
ADDH RD, #IMM8	1	1	1	0	1	RD			IMM8							
LDL RD, #IMM8	1	1	1	1	0	RD			IMM8							
LDH RD, #IMM8	1	1	1	1	1	RD			IMM8							

## Section 5 Initialization/Application Information

### 5.1 Initialization

The recommended initialization of the XGATE is as follows:

1. Clear the XGE bit to suppress any incoming service requests.
2. Make sure that no thread is running on the XGATE.  
This can be done in several ways:
  - a. Poll the XGCHID register until it reads \$00. Also poll XGDBG and XGSWEIF to make sure that the XGATE has not been stopped.
  - b. Enter Debug Mode by setting the XGDBG bit. Clear the XGCHID register. Clear the XGDBG bit.

The recommended method is a.

3. Set the XGVBR register to the lowest address of the XGATE vector space.
4. Clear all Channel ID flags.
5. Copy XGATE vectors and code into the RAM.
6. Initialize the S12X\_INT module.
7. Enable the XGATE by setting the XGE bit.

The following code example implements the XGATE initialization sequence.

### 5.2 Code Example (transmit "Hello World!" on SCI)

```

CPU      S12
; #####
; #          SYMBOLS          #
; #####
SCI_REGS EQU    $00C8      ;SCI register space
SCIBDH   EQU    SCI_REGS+$00 ;SCI Baud Rate Register
SCICR2   EQU    SCI_REGS+$03 ;SCI Control Register 2
SCISR1   EQU    SCI_REGS+$04 ;SCI Status Register 1
SCIDRL   EQU    SCI_REGS+$07 ;SCI Control Register 2
TIE      EQU    $80        ;TIE bit mask
TE       EQU    $08        ;TE bit mask
RE       EQU    $04        ;RE bit mask

```

## Block Guide — S12XGATEV1/D 01.04

```

SCI_VEC          EQU      $D6              ;SCI vector number

INT_REGS         EQU      $0120            ;S12X_INT register space
INT_CFADDR       EQU      INT_REGS+$07     ;Interrupt Configurattion Adress Register
INT_CFDATA       EQU      INT_REGS+$08     ;Interrupt Configurattion Data Registers
RQST             EQU      $80              ;RQST bit mask

XGATE_REGS       EQU      $0380            ;XGATE register space
XGMCTL           EQU      XGATE_REGS+$00   ;XGATE Module Control Register
XGE              EQU      $80
XGDBGM           EQU      $20
XGDBG           EQU      $10
XGSWEIF          EQU      $04
XGCHID           EQU      XGATE_REGS+$02   ;XGATE Channel ID Register
XGVBR            EQU      XGATE_REGS+$04   ;XGATE Vector Base Register
XGIF             EQU      XGATE_REGS+$08   ;XGATE Interrupt Flag Vector
XGSWT            EQU      XGATE_REGS+$18   ;XGATE Software Trigger Register
XGSEM            EQU      XGATE_REGS+$1A   ;XGATE Semaphore Register

RPAGE            EQU      $0016

RAM_SIZE         EQU      20*$400          ;20k RAM
RAM_START_GLOBAL EQU      $10_0000-RAM_SIZE
RAM_START_XGATE  EQU      $1_0000-RAM_SIZE
RAM_START_S12    EQU      $1000
RPAGE_VALUE      EQU      RAM_START_GLOBAL>>12

XGATE_VECTORS     EQU      RAM_START_S12
XGATE_DATA        EQU      RAM_START_S12+(4*128)

XGATE_OFFSET     EQU      (RAM_START_XGATE+(4*128))-XGATE_DATA_BEGIN

BUS_FREQ_HZ      EQU      40_000000

; #####
; #          RESET VECTOR          #
; #####
;          ORG      $FFFE
;          DW       START_OF_CODE

;          ORG      $3000,$9000

START_OF_CODE
; #####
; #          INITIALIZE SCI          #
; #####
INIT_SCI          MOVW     #(BUS_FREQ_HZ/(16*9600)), SCIBDH      ;set baud rate
                  MOVB     #(TIE|TE|RE), SCICR2 ;enable tx buffer empty interrupt

; #####
; #          INITIALIZE S12X_INT      #
; #####
INIT_INT          SEI                      ;disable interrupts
                  MOVB     #(SCI_VEC&$F0), INT_CFADDR           ;switch SCI interrupts to XGATE
                  MOVB     #RQST|$01, INT_CFDATA+((SCI_VEC&$0F)>>1)

; #####
; #          INITIALIZE XGATE          #
; #####
INIT_XGATE        MOVB     #(XGDBGM|XGSWEIF), XGMCTL           ;clear all XGMCTL bits

```

```

BRSET    XGCHID, $FF, INIT_XGATE          ;wait until current thread is done
MOVW     #$10000-RAM_SIZE, XGVBR+2        ;set vector base register

LDX      #XGIF                             ;clear all channel interrupt flags
LDD      #$FFFF
STD      2,X+
STD      2,X+
STD      2,X+
STD      2,X+
STD      2,X+
STD      2,X+
STD      2,X+
STD      2,X+
STD      2,X+

MOVW     #$FF00, XGSWT                     ;clear all software triggers

;#####
;#      INITIALIZE XGATE VECTOR SPACE      #
;#####
INIT_XGATE_VECTOR_SPACE MOVW     #(RAM_START_GLOBAL>>12), RPAGE ;set all vectors to dummy
service routine
LDX      #128
LDY      #RAM_START_S12
LDD      #XGATE_DUMMY+XGATE_OFFSET
INIT_XGATE_VECTOR_SPACE_LOOP
STD      4,Y+
DBNE     X,INIT_XGATE_VECTOR_SPACE_LOOP
MOVW     #XGATE_CODE_BEGIN+XGATE_OFFSET, RAM_START_S12+(2*SCI_VEC) ;set
SCI INTERRUPT VECTOR
MOVW     #XGATE_DATA_BEGIN+XGATE_OFFSET, RAM_START_S12+(2*SCI_VEC)+2

;#####
;#      COPY XGATE CODE                     #
;#####
COPY_XGATE_CODE LDX      #XGATE_DATA_BEGIN
COPY_XGATE_CODE_LOOP MOVW     2,X+, 2,Y+
MOVW     2,X+, 2,Y+
MOVW     2,X+, 2,Y+
MOVW     2,X+, 2,Y+
CPX      #XGATE_CODE_END
BLS      COPY_XGATE_CODE_LOOP

;#####
;#      START XGATE                         #
;#####
START_XGATE MOVW     #(XGE|XGDBGM|XGSWEIF), XGMCTL ;enable XGATE
BRA      *

CPU      XGATE
;#####
;#      XGATE DATA                         #
;#####
ALIGN    1
XGATE_DATA_BEGIN
XGATE_DATA_SCI_PTR DW      SCI_REGS ;pointer to SCI register space
XGATE_DATA_MSG_IDX DB      XGATE_DATA_MSG-XGATE_DATA_BEGIN ;string pointer
XGATE_DATA_MSG FCC      "Hello World!" ;ASCII string
XGATE_DATA_END DB      $0D ;CR

```

```

#####
;#                XGATE CODE                #
#####
                ALIGN    1
XGATE_CODE_BEGIN    LDW      R2,(R1,#(XGATE_DATA_SCI_PTR-XGATE_DATA_BEGIN));SCI -> R2
                   LDB      R3,(R1,#(XGATE_DATA_MSG_IDX-XGATE_DATA_BEGIN));msg -> R3
                   LDB      R4,(R1,R3+)                                ;curr. char -> R4
                   STB      R3,(R1,#(XGATE_DATA_MSG_IDX-XGATE_DATA_BEGIN));R3 -> idx
                   LDB      R0,(R2,#(SCISR1-SCI_REGS))                ;initiate SCI transmit
                   STB      R4,(R2,#(SCIDRL-SCI_REGS))                ;initiate SCI transmit
                   CMPL     R4,#$0D
                   BEQ      XGATE_CODE_DONE
                   RTS
XGATE_CODE_DONE     LDL      R4,$$00                                ;disable SCI interrupts
                   STB      R4,(R2,#(SCICR2-SCI_REGS))
                   LDL      R3,#(XGATE_DATA_MSG-XGATE_DATA_BEGIN);reset R3
                   STB      R3,(R1,#(XGATE_DATA_MSG_IDX-XGATE_DATA_BEGIN))
XGATE_CODE_END      RTS
XGATE_DUMMY         EQU      XGATE_CODE_END

```

# Index

## –A–

ADC instruction 41  
 ADD instruction 42  
 ADDH instruction 43  
 Addition instructions 41, 42, 43, 44  
 ADDL instruction 44  
 Addressing mode  
     Dyadic (DYA) 35  
     Immediate 3 bit wide (IMM3) 34  
     Immediate 4 bit wide (IMM4) 34  
     Immediate 8 bit wide (IMM8) 35  
     Index register plus immediate offset (IDO5) 36  
     Index register plus register offset (IDR) 36  
     Index register plus register offset with post-increment (IDR+) 36  
     Post-increment 36  
     Index register plus register offset with pre-decrement (-IDR) 36  
     Pre-decrement 36  
     Inherent (INH) 34  
     Monadic (MON) 35  
     Relative 11 Bit Wide (REL11) 36  
     Relative 9 bit wide (REL9) 36  
     Triadic (TRI) 35  
 Addressing modes 33  
 AND instruction 45  
 ANDH instruction 46  
 ANDL instruction 47  
 Arithmetic instructions 37  
     Addition 41, 42, 43, 44  
     Parity 95  
     Sign extension 101  
     Subtraction 99, 105, 106, 107  
 ASR instruction 48

## –B–

BCC instruction 49, 50, 64  
 BEQ instruction 51  
 BFEXT instruction 52  
 BFFO instruction 53  
 BFINS instruction 54  
 BFINSI instruction 55  
 BFINSX instruction 56  
 BGE instruction 57  
 BGT instruction 58  
 BHI instruction 59  
 BHS instruction 60

Bit field instructions 52, 53, 54, 55, 56  
 Bit Field Operations 38  
 Bit test instructions 61, 62  
 BITH instruction 61  
 BITL instruction 62  
 BLE instruction 63  
 Block diagram 11  
 BLS instruction 65, 66  
 BMI instruction 67  
 BNE instruction 68  
 Boolean logic instructions 37  
     AND 45, 46, 47  
     OR 92, 93, 94  
     XNOR 110, 111, 112  
 BPL instruction 69  
 BRA instruction 70  
 Branch instructions 38, 49, 50, 51, 57, 58, 59, 60, 63, 64, 65, 66, 67, 68, 69, 70, 72, 73  
 BRK instruction 71  
 BVC instruction 72  
 BVS instruction 73

## –C–

CMP instruction 74, 77  
 CMPL instruction 75  
 COM instruction 76, 89  
 Compare instructions 74, 75, 77, 78  
 Complement instructions 76, 89, 90  
 CPCH instruction 78  
 CSEM instruction 79  
 CSL instruction 80  
 CSR instruction 81  
 Cycle notation 39

## –D–

Debug Features 31  
 Dyadic addressing mode 35

## –F–

Features 12

## –I–

Immediate addressing mode 34, 35  
 Indexed addressing mode 36  
 Inherent addressing mode 34  
 Instruction coding 113

## –J–

JAL instruction 82

–L–

LDB instruction 83  
LDH instruction 84  
LDL instruction 85  
LDW instruction 86  
Load instructions 37, 83, 84, 85, 86  
LSL instruction 87  
LSR instruction 88

–M–

Memory map 27  
Monadic addressing mode 35

–N–

Naming conventions 33  
NEG instruction 90  
NOP instruction 91

–O–

OR instruction 92  
ORH instruction 93  
ORL instruction 94

–P–

PAR instruction 95  
Programming model 26

–R–

Register map 12  
Registers 13  
    XGATE Channel ID Register (XGCHID) 16  
    XGATE Condition Code Register (XGCCR) 21  
    XGATE Module Control Register (XGMCTL) 14  
    XGATE Program Counter Register (XGPC) 22  
    XGATE Register 1 (XGR1) 22  
    XGATE Register 2 (XGR2) 22  
    XGATE Register 3 (XGR3) 23  
    XGATE Register 4 (XGR4) 23  
    XGATE Register 5 (XGR5) 24  
    XGATE Register 6 (XGR6) 24  
    XGATE Register 7 (XGR7) 25  
    XGATE Semaphore Register (XGSEM) 20  
    XGATE Software Trigger Register (XGSWT) 19  
Relative Addressing Mode 36  
Relative addressing mode 36  
ROL instruction 96  
ROR instruction 97  
RTS instruction 98

–S–

SBC instruction 99  
Semaphore instructions 79, 100  
Semaphores 28  
SEX instruction 101  
Shift instructions 38, 48, 80, 81, 87, 88, 96, 97  
SIF instruction 102  
Sign extension instructions 101  
SSEM instruction 100  
STB instruction 103  
Store instructions 37, 103, 104  
STW instruction 104  
SUB instruction 105  
SUBH instruction 106  
SUBL instruction 107  
Subtraction instructions 99, 105, 106, 107

–T–

TFR instruction 108  
Transfer instructions 37, 108  
Triadic addressing mode 35  
TST instruction 109

–X–

XGATE Module Control Register (XGMCTL) 14  
XGCCR register 21  
XGCHID Register 16  
XGMCTL Register 14  
XGMCTL register 14  
XGPC register 22  
XGR1 register 22  
XGR2 register 22  
XGR3 register 23  
XGR4 register 23  
XGR5 register 24  
XGR6 register 24  
XGR7 register 25  
XGSEM register 20  
XGSWT register 19  
XNOR instruction 110  
XNORH instruction 111  
XNORL instruction 112



# Block Guide End Sheet

**FINAL PAGE OF  
122  
PAGES**