Xgate Co-Processor (xgate) Specification

Author: Robert Hayes
rehayes@opencores.org

Rev. 0.1
August 12, 2010

This page has been intentionally left blank.

Revision History

Rev.	Date	Author	Description
0.1	09/29/09	Robert Hayes	

Contents

1

Introduction

The Xgate Co-processor Module, Xgate, is a 16 bit programmable RISC processor that is managed by a host CPU to reduce the host load in handling interrupts. Because the Xgate is user programmable there is a great deal of user control in how to preprocess data from peripheral modules. This module may be configured as a simple DMA controller to organize data such that the host only operates with whole messages and not individual words or bytes. The Xgate may also handle higher levels of messaging protocols than the peripheral hardware recognizes. Encryption algorithms are also supported by the instruction set.

The ideal application for the Xgate co-processor is in an ASIC environment where a specific host processor is required but the host as a standalone processor lacks sufficient computational resources to service the target application. Such a situation might occur in an existing application that has an extensive software base or the host processor has a significant history in similar applications. In this environment the Xgate provides the additional resources needed for the application while also providing the user programmability that may allow the ASIC to be used in other applications.

The use of an Xgate co-processor in an FPGA may not be as advantageous as in an ASIC environment because of the reprogramability of the FPGA hardware. Because the Xgate co-processor offers a generic solution to a variety of problems and the functionality can be changed by upgrading user software without changing the FPGA hardware implementation it may still be a valuable solution to certain problems. The use of the Xgate module may also reduce development time since the application can begin debug sooner and changes can be prototyped in user software.

FEATURES

- **Instruction set compatible with Freescale XGATE co-processor**
- **Handles up to 127 interrupt inputs**
- **Eight software triggerable interrupt channels.**
- **Eight semaphore registers to coordinate host/Xgate shared memory.**

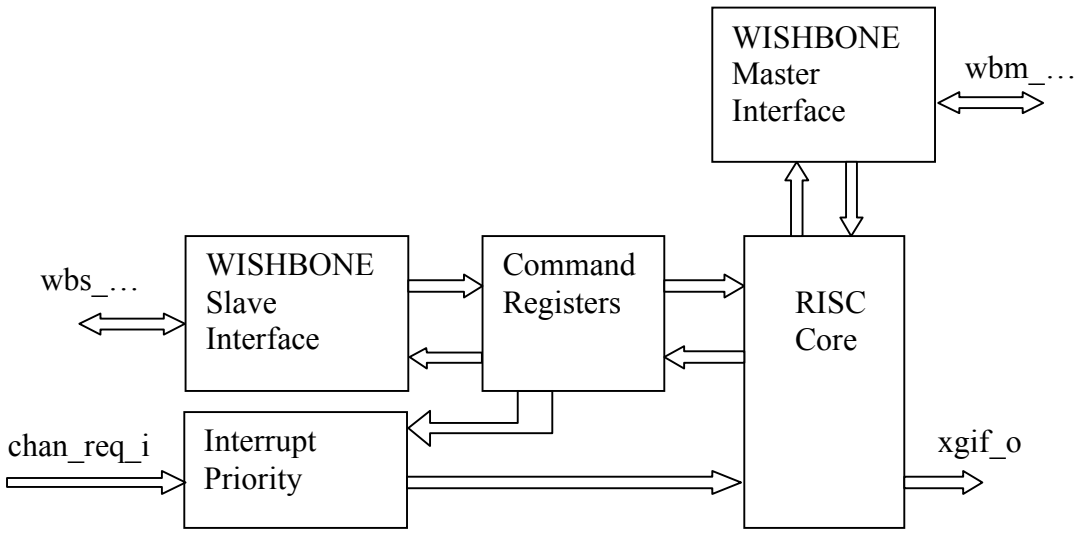
- **Static synchronous design**
- **Fully synthesizable**

2

Architecture

The Xgate core is built around five primary blocks; the WISHBONE Slave Interface, WISHBONE Master Interface, the Control Registers, Interrupt Priority, and RISC Processor Core.

Figure 2-1 Internal Xgate Structural Overview



2.1 WISHBONE Slave Interface

The host processor uses this bus to access the Xgate control and status registers. The WISHBONE Slave Interface isolates the Xgate functionality from the WISHBONE bus. This interface takes the bus specific signals and generates a generic set of control signals to drive the Xgate control registers. Isolating the WISHBONE bus should help promote Xgate module reusability by localizing the scope of changes needed to retarget the Xgate module to another bus environment.

2.2 Control Registers

This module receives the generic write signals from the WISHBONE Slave Interface and applies these signals to the registers that the host processor uses to control Xgate operation.

2.3 Interrupt Interface

The Interrupt Interface takes the input interrupts from other slave peripheral modules and prioritizes them for processing by the Xgate RISC Processor Core. . The active interrupt input with the lowest index number becomes the chosen channel select enable.

2.3.1 Simple Interrupt Interface

Previous versions of the Xgate module used this simple interrupt interface because it had the least hardware and least functionality. This module takes the 127 interrupt inputs and does a simple encoding to generate the six bit bus that becomes the active channel select signal

Because the Xgate module is in the signal path from the interrupt sources to the host CPU there must always be always be some code running in the Xgate module to activate the appropriate interrupt output to the CPU.

2.3.2 Bypass Interrupt Interface

The current version of the Xgate module uses this implementation. This module does the same basic interrupt encoding as the “Simple Interrupt Interface” except it includes additional functionality to allow an interrupt input to be bypassed around the Xgate module and connected directly to the host CPU. This will eliminate the latency and software overhead when it is required for the host CPU to process some interrupts directly while still reserving the option to enable the Xgate module to preprocess those

same interrupt sources at some other time. To implement this functionality the IRQ_BP0 – IRQ_BP7 control registers are added to the WISHBONE slave interface.

2.3.3 Programmable Interrupt Interface

This is a proposed functional improvement and is not currently implemented. This module will include the functionality of the "Bypass Interrupt Interface" with additional functionality to allow interrupt priorities to be set by the host CPU. This will eliminate the latency and software overhead when it is required for the host CPU to process some interrupts directly while still reserving the option to enable the Xgate module to preprocess those same interrupt sources at some other time. To implement this functionality additional control registers will need to be added to the WISHBONE slave interface.

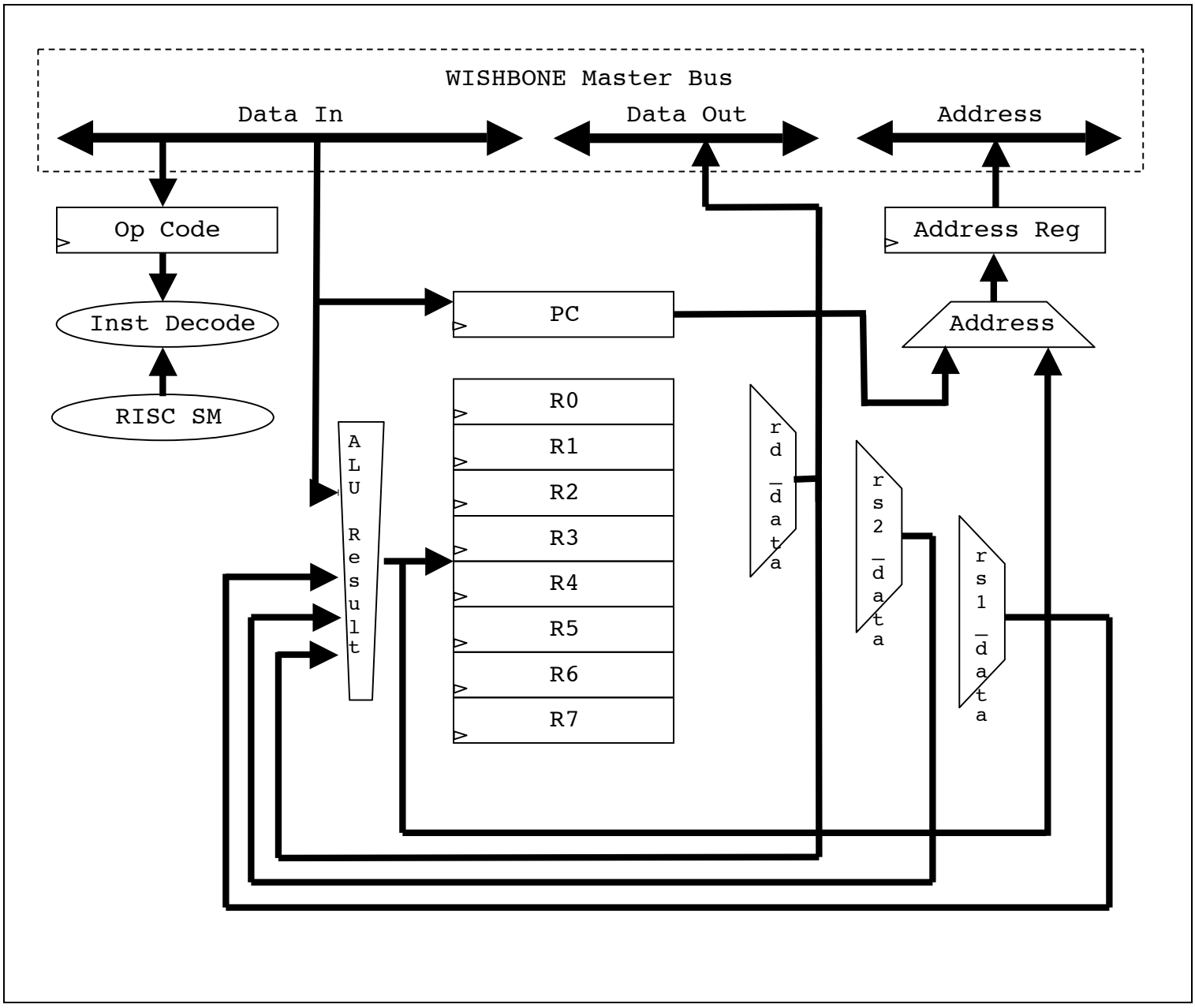
2.4 WISHBONE Master Interface

The Xgate module uses the WISHBONE Master Interface to access the shared memory space where it's software code, data, and other slave peripheral registers may be accessed. The WISHBONE Master Interface isolates the Xgate functionality from the WISHBONE bus. This interface takes the bus specific signals and generates a generic set of control signals to drive the Xgate RISC Processor. Isolating the WISHBONE bus should help promote Xgate module reusability by localizing the scope of changes needed to retarget the Xgate module to another bus environment.

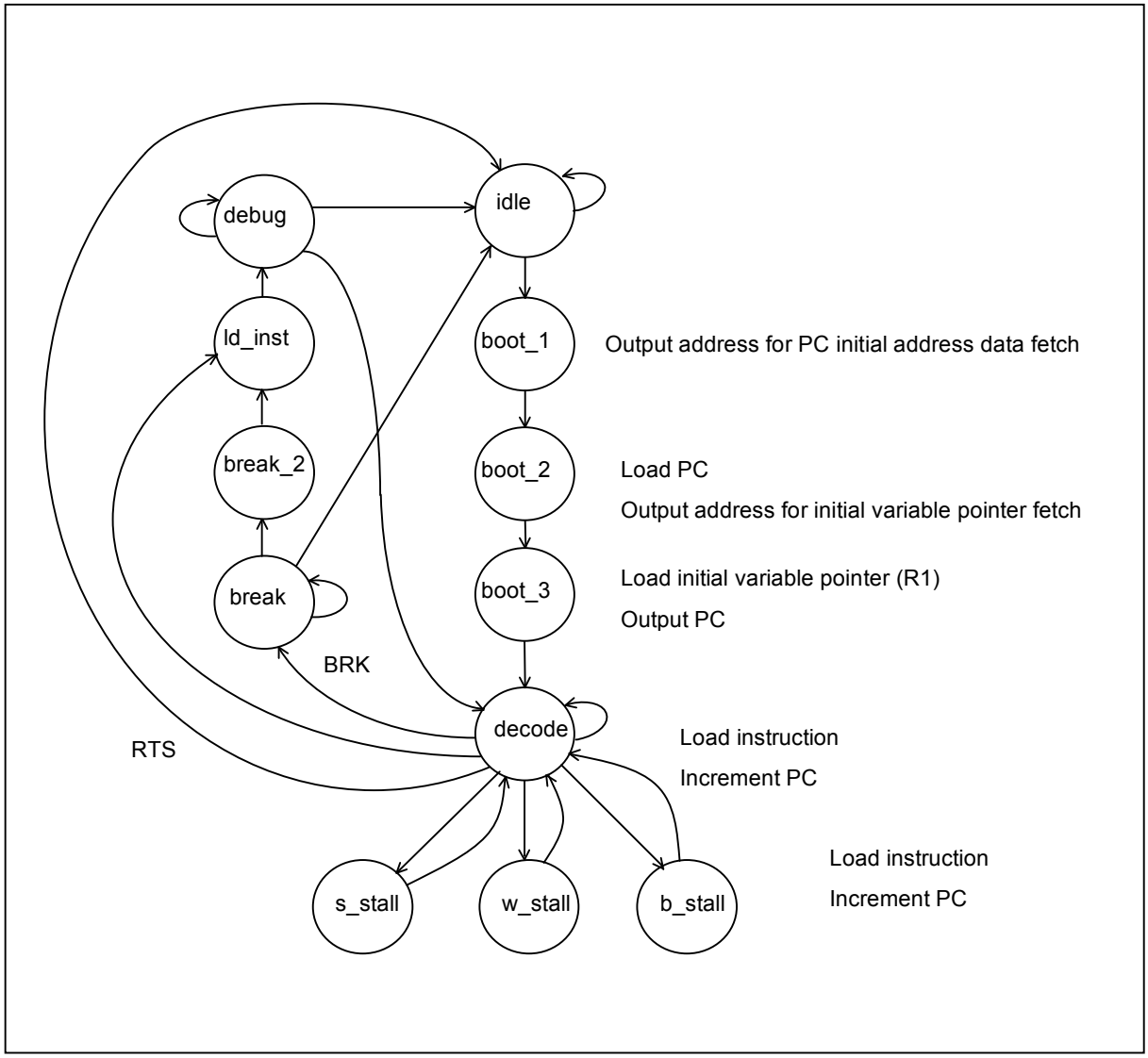
2.5 RISC Processor Core

The RISC Processor Core is the key element of the Xgate module. This module implements the Xgate instruction set.

2.5.1 Hardware Model



2.5.2 State Machine



Idle: The idle state is the condition where the xgate RISC processor waits to be enabled and an interrupt input to be activated.

Boot_1: State entered after an input interrupt is detected. Output the address stored in XGVBR + 2*XGCHID to fetch PC (Program Counter).

Boot_2: State entered after Boot_1. Load the PC and output the address of the variable pointer, XGVBR + 2*XGCHID + 2.

Boot_3: State entered after Boot_2. Load the variable pointer. Output the PC and prepare to load the first instruction.

Decode: In normal user operation this is the state entered after Boot_3 and the instruction is decoded and executed. For single cycle instructions, if Debug Mode is not active, then the Program Counter is incremented and the next instruction is loaded. For multi-cycle instructions one of the three instruction continue states is selected. If an RTS instruction is decoded then the state machine is returned to the Idle state to wait for the next change in XGCHID.

S_stall: Simple stall is the state that branch instructions use to change the PC when a change of flow is required. This state is also used for Store instructions.

W_stall: Word stall is the state Load instructions use to retrieve a word of RAM memory data.

B_stall: Byte stall is the state Load instructions use to retrieve a byte of RAM memory data.

Break: If the Decode state detects a BRK command then the Break state is entered. The RISC processor remains in the Break state till:

Single Step Command

XGEN is set to zero

XGCHID is set to zero

Break_2: After a Single Step Command is issued in the Break state the PC is incremented in this state.

Ld_inst: The Load Instruction state is used in debug mode to load the next instruction to be executed into the RISC instruction register.

Debug: debug is the state where the RISC processor waits for an input to execute the next command Exit conditions for this state are:

Single Step Command

XGEN is set to zero

XGCHID is set to zero

2.5.3 Semaphore Bits

Each semaphore bit is a simple state machine that is controlled by write commands from either the host or RISC processor. The first write command to set a semaphore bit takes control of the semaphore state until the bit is cleared by the setting processor. The host has priority if the host and RISC processor attempt to write the semaphore bit in the same clock cycle. Each processor has a different read view of the semaphore bit so it can determine if it has successfully set the bit.

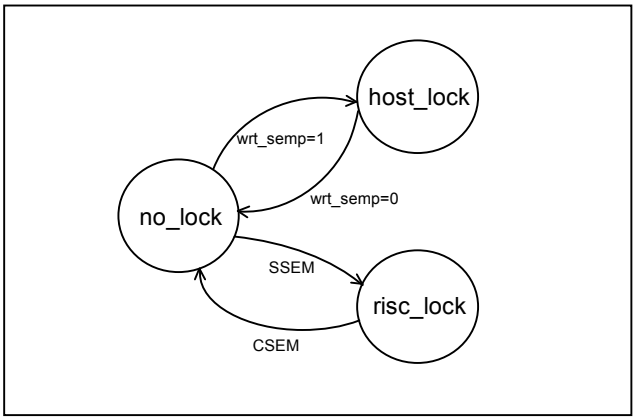
// Semaphore states

NO_LOCK = 2'b00

RISC_LOCK = 2'b10

HOST_LOCK = 2'b11

These are the states for the semaphore bits. Only three of the four possible states are required to cover all the real things that can happen. For decoding the state of a semaphore bit the MSB is equivalent to "locked" and the LSB is equivalent to which processor is the one that owns the lock. Therefore state 2'b01 doesn't make sense because it implies there is no "lock" but the bit is owned by the host. In terms of the actual logic it probably makes little if any difference how the states are encoded but from the mental perspective it maybe helpful in understanding and debug to have the individual bits of the state be connected to a unique meaning.

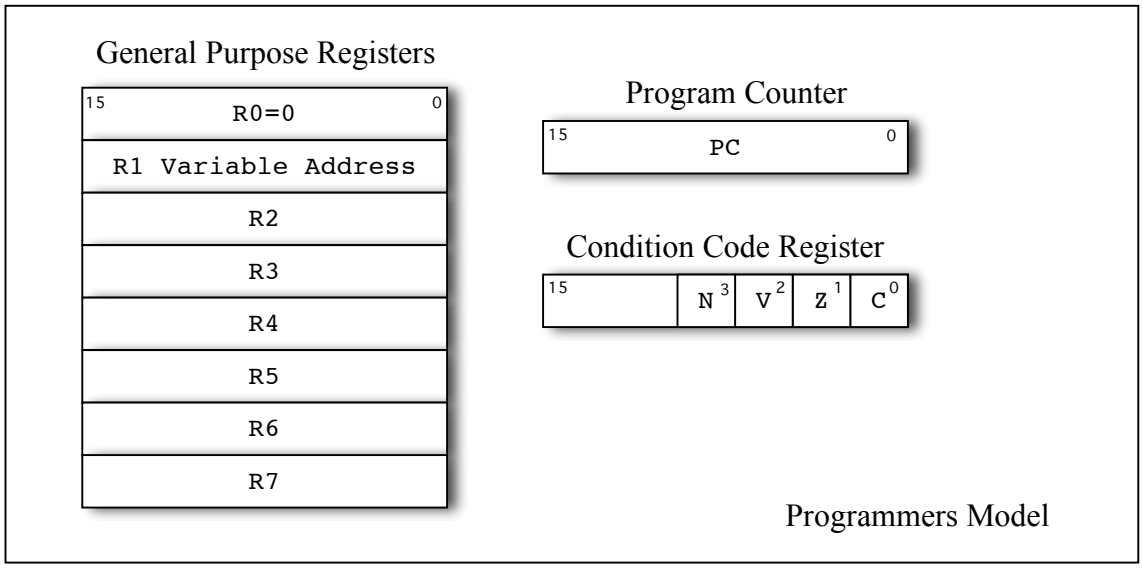


3

Operation

The Xgate Module is a simple RISC CPU with an instruction set that is compatible with the Freescale XGATE module.

The COP module also has the capability to generate an interrupt at a programmed number of cycles before the cop_rst_o initiates a system reset. This functionality is primarily intended as a debug feature.



3.1 Software Example

The recommended software procedure for using the Xgate Module:

1. Initialize Xgate
   a) Load RAM with Xgate software.
   b) Set XGVBR register to base address of input interrupt vectors.
   c) Enable Xgate interrupts by clearing bits in the IRQ_BP0 – IRQ_BP7 control registers.

d) Enable Xgate by setting the XGE bit.

e) Enable host to accept interrupts.

f) Enable peripheral modules to output interrupts as required.

2. Normal Operation

a) Service Xgate interrupt requests as they are received by the host.

3.2 Software Triggers

3.3 Semaphore Bits

3.4 Program Memory Connection Options

3.5 Debug Mode

3.6 Instruction Set Summary

The RISC Processor Core is the key element of the Xgate module. This module implements the Xgate instruction set.

Function	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Return to Scheduler and other																
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

3.7 Inconsistent Instruction Set Documentation

The following instructions have problems in the Freescale documentation.

3.7.1 SUBL

Freescale Documentation:

Z:
V: Set if a two's complement overflow resulted from the 8-bit operation; cleared otherwise.

RD[15]old & !RD[15]new
C: Set if there is a carry from the bit 7 to bit 8 of the result; cleared otherwise.
!RD[15]old & RD[15]new

Xgate implementation:

For signed subtraction the rules for overflow are:

Negative number minus Positive number creates a Positive number OR

Positive number minus Negative number creates a Negative number

So the overflow calculation should be:

V = RD[7] & !IMM8[7] & !result[7] | !RD[7] & IMM8[7] & result[7]

The Carry calculation should be:

C = !RD[7] & IMM8[7] | !RD[7] & result[7] | IMM8[7] & result[7]

These calculations for setting the Carry and Overflow bits are also consistent with the calculations used to set the Carry and Overflow bits using the CMPL instruction.

3.7.2 ADDL

Freescale Documentation:

V: Set if a two's complement overflow resulted from the 8-bit operation; cleared otherwise.
RD[15]old & RD[15]new
C: Set if there is a carry from the bit 7 to bit 8 of the result; cleared otherwise.
RD[15]old & RD[15]new

Xgate implementation:

For signed addition the rules for overflow are:

Positive number plus Positive number creates a Negative number OR

Negative number plus Negative number creates a Positive number

So the overflow calculation should be:

V = !RD[7] & !IMM8[7] & result[7] | RD[7] & IMM8[7] & !result[7]

The Carry calculation should be:

C = !RD[7] & IMM8[7] | RD[7] & !result[7] | IMM8[7] & !result[7]

3.7.2 SUBH

Freescale Documentation:

Z = ????

4

Registers

The Xgate Module has a WISHBONE slave bus interface configured for a 16-bit bus width with 8-bit granularity.

List of Registers

Name	Address	Width	Access	Description
XGMCTL	0x00	16	RW	Xgate Module Control Register
XGCHID	0x02	16	RW	Xgate Channel ID Register
	0x04	16	RW	Reserved
	0x06	16	RW	Reserved
XGVBR	0x08	16	RW	Xgate Vector Base Address
XGIF_7	0x0A	16	RW	Xgate Interrupt Flag Register #7
XGIF_6	0x0C	16	RW	Xgate Interrupt Flag Register #6
XGIF_5	0x0E	16	RW	Xgate Interrupt Flag Register #5
XGIF_4	0x10	16	RW	Xgate Interrupt Flag Register #4
XGIF_3	0x12	16	RW	Xgate Interrupt Flag Register #3
XGIF_2	0x14	16	RW	Xgate Interrupt Flag Register #2
XGIF_1	0x16	16	RW	Xgate Interrupt Flag Register #1
XGIF_0	0x18	16	RW	Xgate Interrupt Flag Register #0
XGSWT	0x1A	16	RW	Xgate Software Trigger Register
XGSEM	0x1C	16	RW	Xgate Semaphore Register
	0x1E	16	R	Reserved
XGCCR	0x20	16	RW	Xgate Condition Code Register
XGPC	0x22	16	RW	Xgate Program Counter
	0x24	16	R	Reserved

Name	Address	Width	Access	Description
XGR1	0x26	16	RW	Xgate Register 1
XGR2	0x28	16	RW	Xgate Register 2
XGR3	0x2A	16	RW	Xgate Register 3
XGR4	0x2C	16	RW	Xgate Register 4
XGR5	0x2E	16	RW	Xgate Register 5
XGR6	0x30	16	RW	Xgate Register 6
XGR7	0x32	16	RW	Xgate Register 7
		16	R	Reserved
IRQ_BP0	0x40	16	RW	IRQ Bypass Register 0
IRQ_BP1	0x42	16	RW	IRQ Bypass Register 1
IRQ_BP2	0x44	16	RW	IRQ Bypass Register 2
IRQ_BP3	0x46	16	RW	IRQ Bypass Register 3
IRQ_BP4	0x48	16	RW	IRQ Bypass Register 4
IRQ_BP5	0x4A	16	RW	IRQ Bypass Register 5
IRQ_BP6	0x4C	16	RW	IRQ Bypass Register 6
IRQ_BP7	0x4E	16	RW	IRQ Bypass Register 7

Table 1: List of registers

4.1 Xgate Module Control Register (XGMCTL)

Note: The XGMCTL register can only be written using a word(16 bit) access.

Bit #	Access	Description
15	W	XGEM, XGE Mask – The XGE bit can only be changed when the XGEM bit is set in the same register access. ‘0’ No change allowed to XGE control bit. ‘1’ Change allowed to XGE control bit. Always reads as ‘0’.
14	W	XGFRZM, XGFRZ Mask – The XGFRZ bit can only be changed when the XGEM bit is set in the same register access. ‘0’ No change allowed to XGFRZ control bit. ‘1’ Change allowed XGFRZ control bit.

Bit #	Access	Description
		Always reads as '0'.
13	W	XGDGBM, XGDGB Mask – The XGDGB bit can only be changed when the XGDGBM bit is set in the same register access. '0' No change allowed to XGDBG control bit. '1' Change allowed to XGDBG Control bit. Always reads as '0'.
12	W	XGSSM, XGSS Mask – The XGSS bit can only be changed when the XGSSM bit is set in the same register access. '0' No change allowed to XGSS control bit. '1' Change allowed to XGSS control bit. Always reads as '0'.
11	W	XGFACTM, XGFACT Mask – The XGFACT bit can only be changed when the XGFACTM bit is set in the same register access. '0' No change allowed to XGFACT control bit. '1' Change allowed to XGFACT control bit. Always reads as '0'.
10		BRK_IRQ_ENM, BRK_IRQ_EN Mask – The BRK_IRQ_EN bit can only be changed when the BRK_IRQ_ENM bit is set in the same register access. '0' No change allowed to BRK_IRQ_EN control bit. '1' Change allowed to BRK_IRQ_EN control bit. Always reads as '0'.
9	RW	XGSWEIFM, XGSWEIF Mask – The XGSWEIF bit can only be changed when the XGSWEIFM bit is set in the same register access. '0' No change allowed to XGSWEIF control bit. '1' Change allowed to XGSWEIF control bit. Always reads as '0'.
8	RW	XGIEM, XGIE Mask – The XGIE bit can only be changed when the XGIEM bit is set in the same register access. '0' No change allowed to XGIE control bit. '1' Change allowed to XGIE control bit. Always reads as '0'.

Bit #	Access	Description
7	RW	XGE, XG Enable – The XGE bit can only be changed when the XGEM bit is set in the same register access. '0' The Xgate module is disabled. The currently active interrupt request will be completed and then no new interrupt processing requests will be accepted. '1' The Xgate module is enabled to start processing channel interrupts.
6	RW	XGFRZ, XG Freeze Mode – The XGFRZ bit functionality is currently unimplemented. This bit can only be changed when the XGFRZM bit is set in the same register access. '0' The XGFRZ register bit is clear. '1' The XGFRZ register bit is set.
5	RW	XGDBG, XG Debug Mode – The XGDBG bit can only be changed when the XGDBGM bit is set in the same register access. '0' The Xgate module is not in DEBUG Mode. '1' The Xgate module is in DEBUG Mode. Along with writing to the XGGBG bit from the host interface DEBUG Mode can also be entered by executing the 'BRK' instruction by the RISC Processor Core.
4	RW	XGSS, XG Single Step – The XGSS bit can only be changed when the XGSSM bit is set in the same register access. '0' The Xgate module is disabled. The currently active channel interrupt will be completed and then no new interrupt channel processing requests will be accepted. '1' Write - Xgate module should execute a single RISC instruction. Read – A RISC instruction is being processed.
3	RW	XGFACT, XG Fake Activity – The XGFACT bit functionality is currently unimplemented. This bit can only be changed when the XGFACTM bit is set in the same register access. '0' The XGFACT bit is clear. '1' The XGFACT is set.
2	RW	BRK_IRQ_EN, Break Interrupt Enable – The BRK_IRQ_EN bit can only be changed when the BRK_IRQ_EN M bit is set in the same register access. '0' The Xgate module is operating in a mode compatible with the Freescale XGATE module. BRK instructions do not generate an Xgate error interrupt output. '1' The Xgate module is enabled to output an error interrupt

Bit #	Access	Description
		whenever a BRK command is encountered. Set this bit whenever no BRK commands are expected to be used in the software.
1	RW	XGSWEIF, XG Software Error Interrupt Flag – The XGSWEIF bit can only be changed when the XGSWEIFM bit is set in the same register access. '0' The Xgate module is operating correctly. Writing '0' has no effect. '1' The Xgate RISC processor has detected an error condition. Writing '1' will clear a set XGSWEIF bit and terminate the current interrupt process. The Xgate module will go to an IDLE state if no other interrupt inputs are active.
0	RW	XGIE, XG Interrupt Enable – The XGIE bit can only be changed when the XGEM bit is set in the same register access. '0' All Xgate module interrupt outputs are disabled. '1' All Xgate module interrupt outputs are enabled.

Reset Value:

XGMCTL: 0004h

Table 2: CNTRL Register Bits

4.2 Xgate Channel ID Register (XGCHID)

Bit #	Access	Description
15:7	R	Reserved, write zeros for future compatibility.
6:0	RW	XGCHID, XG Channel ID – The XGCHID register reflects the value of the current interrupt channel being processed. '0' No interrupt currently being processed. '1' The encoded value of the input interrupt currently being processed. The XGCHID can only be written in DEBUG Mode. When a value is written to the XGCHID register the RISC Processor Core will start executing the code associated with that interrupt input.

Reset Value:

XGCHID: 0000h

Table 4: Channel ID Register Bits

4.3 Xgate Vector Base Address Register (XGVBR)

Bit #	Access	Description
15:1	RW	XGVBR, Sets the vector base address that the Xgate RISC Core uses to determine where in the memory map the code and data space is for a specific interrupt is reserved. XGVBR can only be changed when XGE is zero.

Reset Value:

XGVBR: FE00h

Table 5: XGVBR Register Bits

4.4 Xgate Interrupt Flag Register 7 (XGIFR_7)

The bits in this register reflect the state of interrupts that are being output from the Xgate module to the host CPU. A bit is set by the Xgate module with the "SIF" instruction usually at the end of a process and the host CPU clears the XGIF bit as part of its interrupt service routine.

Clear the interrupt output by writing a '1' to the corresponding XGIF register bit.

The highest numbered interrupt is in the MSB position of the register bank.

Some bits in this register may be unimplemented in a specific hardware design based on the setting of the "MAX_CHANNEL" parameter and will always read as "0".

Bit #	Access	Description
15:0	RW	XGIF_127 – XGIF_112 '0' No interrupt. '1' Interrupt output active.

Reset Value:

XGIFR_7: 0000h

Table 5: XGIF_7 Register Bits

4.12 Xgate Software Trigger Register (XGSWT)

Note: The XGSWT register can only be written using a word(16 bit) access.

Bit #	Access	Description
15:8	W	XGSWTM[7:0], Read always returns zero. XGSWTM, XGSWT Mask – The XGSWT bits can only be changed when the XGSWTM bit is set in the same register access. ‘0’ No change allowed to XGSWT control bit. ‘1’ Change allowed to XGSWT control bit. Always reads as ‘0’.
7:0	RW	XGSWT[7:0], XG Software Trigger – XGSWT bits can only be changed when the corresponding XGSWTM bit is set in the same register access. ‘0’ The task associated with the XGSWT is disabled. ‘1’ The task associated with the XGSWT bit is queued to start processing when it becomes the highest priority Xgate interrupt input.

Reset Value:
XGSWT: 0000h

Table 5: XGSWT Register Bits

4.13 Xgate Semaphore Register (XGSEM)

Note: The XGSEM register can only be written using a word(16 bit) access.

Bit #	Access	Description
15:8	W	XGSEMM, XGSEM Mask – The XGSEM bit can only be changed when the XGSEMM bit is set in the same register access. ‘0’ No change allowed to XGSEM control bit. ‘1’ Change allowed to XGSEM control bit. Always reads as ‘0’.

Bit #	Access	Description
7:0	RW	XGSWT[7:0], XG Software Trigger – The XGSWT bit can only be changed when the XGSWTM bit is set in the same register access. '0' The semaphore bit is either unlocked or locked by the Xgate module. If the host attempts to set the bit by writing a '1' and a subsequent read of the bit returns a '0' the bit is locked by the Xgate module. '1' The semaphore bit is locked by the host processor. The host writes a '0' to clear the bit and unlock the semaphore.

Reset Value:
XGSEM: 0000h

Table 5: XGSEM Register Bits

4.14 Xgate Condition Code Register (XGCCR)

Note: The XGCCR register can only be written when DEBUG mode is active or the XGE bit is clear.

Bit #	Access	Description
15:4	R	Reserved, write zeros for future compatibility.
3	RW	XGN, Xgate Negative – Reflects the current state of the "Negative" bit of the Xgate condition code register.
2	RW	XGZ, Xgate Zero – Reflects the current state of the "Zero" bit of the Xgate condition code register.
1	RW	XGV, Xgate Overflow – Reflects the current state of the "Overflow" bit of the Xgate condition code register.
0	RW	XGC, Xgate Carry – Reflects the current state of the "Carry" bit of the Xgate condition code register.

Reset Value:
XGCCR: 0000h

Table 5: XGCCR Register Bits

4.15 Xgate Program Counter Register (XGPC)

Note: The XGPC register can only be written when DEBUG mode is active or the XGE bit is clear.

Bit #	Access	Description
15:0	RW	XGPC, Read returns the current state of the Xgate Program Counter. Writes to the XGPC register can only be done in Debug Mode.

Reset Value:
XGPC: 0000h

Table 5: XGPC Register Bits

4.16 Xgate Register 1 (XGR1)

Note: The XGR1 register can only be written when DEBUG mode is active or the XGE bit is clear.

Bit #	Access	Description
15:0	RW	XGR1, Read returns the current state of the Xgate General Purpose Register 1. Writes to the XGR1 register can only be done in Debug Mode.

Reset Value:
XGR1: 0000h

Table 5: XGR1 Register Bits

4.23 Interrupt Bypass Register 0 (IRQ_BP0)

The bits in this register are used to control the visibility of interrupts coming from the chan_req_i inputs to the Xgate RISC processor or if they are bypassed to the host processor.

Bit #	Access	Description
15:0	RW	IRQ_BYPASS[15:0] '0' The input interrupts, chan_reg_i[15:0], are accepted by the

Bit #	Access	Description
		Xgate module and blocked from passing through to the xgif_o[15:0] outputs. An internal Xgate processing thread is triggered. The Xgate module triggers the corresponding output interrupt, xgif_o[15:0], as required by the Xgate software. '1' The input interrupts, chan_req_i[15:0] are bypassed directly to the Xgate interrupt outputs, xgif_o[15:0]. The input interrupt is disabled from triggering an Xgate processing thread.

Reset Value:

IRQ_BP0: FFFFh

Table 5: IRQ_BP0 Register Bits

5

Clocks

Name	Source	Rates (MHz)			Remarks	Description
		Max	Min	Resolution		
wbs_clk_i	System	200	-	-	Master clock for all Xgate bus registers. Positive edge active.	System clock.
risc_clk_i	System					RISC Processor clock

Table 3: List of clocks

The wbs_clk_i has no timing constraints based on the RTL implementation although there may be constrains applied for synthesis results to be compatible with the target physical implementation. If the Xgate is targeted for an ASIC implementation then [wbs_clk_i] should be used as the scan clock, any clock multiplexing required to make [wbs_clk_i] the scan clock should be done at the system level external to the Xgate Module.

The risc_clk_i is the clock used to run the Xgate RISC processor submodule. The frequency of this clock should be the same as or 2X the frequency of the wbs_clk_i input. The risc_clk_i input is assumed to be in phase with the wbs_clk_i input.

6

IO Ports

Port	Width	Direction	Description
wbs_clk_i	1	Input	WISHBONE Bus Clock Input, Master Clock
wbs_rst_i	1	Input	WISHBONE Slave Synchronous Reset
wbs_adr_i	5	Input	WISHBONE Slave Lower address bits
wbs_dat_i	16	Input	WISHBONE Slave Bus Data
wbs_dat_o	16	Output	WISHBONE Slave Bus Data
wbs_we_i	1	Input	WISHBONE Slave Write enable
wbs_stb_i	1	Input	WISHBONE Slave Strobe signal/Core select
wbs_cyc_i	1	Input	WISHBONE Slave Valid bus cycle
wbs_sel_i	2	Input	WISHBONE Slave Data Bus Byte Select
wbs_ack_o	1	Output	WISHBONE Slave Bus cycle acknowledge
wbm_adr_o	16	Output	WISHBONE Master address bits
wbm_dat_i	16	Input	WISHBONE Master Bus Data
wbm_dat_o	16	Output	WISHBONE Master Bus Data
wbm_we_o	1	Output	WISHBONE Master Write enable
wbm_stb_o	1	Output	WISHBONE Master Strobe signal/Core select
wbm_cyc_o	1	Output	WISHBONE Master Valid bus cycle
wbm_sel_o	2	Output	WISHBONE Master Data Bus Byte Select
wbm_ack_i	1	Input	WISHBONE Master Bus cycle acknowledge

Port	Width	Direction	Description
arst_i	1	Input	Asynchronous Reset
risc_clk_i	1	Input	Clock signal for the RISC Core Processor
chan_req_i	127[1]	Input	Xgate Interrupt Request
xgif_o	127[1]	Output	Xgate Interrupt Request
xg_sw_irq	1	Output	Xgate Software Error Interrupt
xgswt	8	Output	Xgate Software Trigger
debug_mode_i	1	Input	Force Xgate into debug mode
secure_mode_i	1	Input	Limit host access to RISC core registers
scantestmode_i	1	Input	Scan Test Mode Enable

1) The actual width is determined by the parameter MAX_CHANNEL, the default is the maximum value: 127.

Table 4: List of IO ports

6.1 WISHBONE Slave Interface

The core features a WISHBONE RevB.3 compliant WISHBONE Classic interface that operates in SLAVE mode. All output signals are registered. Each access takes 2 clock cycles. To limit a WISHBONE access to just two clock cycles the following synthesis rules should be used:

- Single cycle timing for wbs_cyc_i and wbs_stb_i
- Two cycle timing for wbs_adr_i, wbs_data_i, and wbs_data_o. (Single cycle timing could be used but it would be a waste of resources to meet an over constrained timing path.)

Note: Use the "SINGLE_CYCLE" parameter to do a WISHBONE bus access in one clock cycle.

WISHBONE DATASHEET	
Description	Specification
General description:	16-bit SLAVE

Supported Cycles:	SLAVE, READ/WRITE	
Data port, size: Data port, granularity: Data port, maximum operand size: Data transfer ordering: Data transfer sequencing:	16 bit word 8 bit byte	
Supported signal list and cross reference to equivalent WISHBONE signals:	Signal Name	WISHBONE Equiv.
	wbs_clk_i	CLK_I
	wbs_rst_i	RST_I
	wbs_adr_i	ADR_I()
	wbs_dat_i	DAT_I()
	wbs_dat_o	DAT_O()
	wbs_we_i	WE_I
	wbs_stb_i	STB_I
	wbs_cyc_i	CYC_I
	wbs_sel_i	SEL_I()
	wbs_ack_o	ACK_O

6.1.1 wbs_rst_i

The synchronous reset signal has a minimum pulse width requirement of one [wbs_clk_i] clock period. It will take two [wbs_clk_i] clock cycles for all registers in the Xgate Module to initialize. Also see information on pin [arst_i].

6.1.2 wbs_adr_i

The slave WISHBONE address pins are defined to be compatible with a WISHBONE bus that is word addressed and byte accessible. This means that there is no [wbs_adr_i[0]] input. If only word accesses are to be allowed then the LSB of the host address bus can be connected to [wbs_adr_i[1]].

6.1.3 wbs_sel_i

The [wbs_sel_i] are the WISHBONE byte lane select signals. These signals allow selected Xgate registers to be written in byte mode. Note that some registers can only be written in 16 bit word mode. If only word mode accesses are to be done then these signals should be tied hi.

6.2 Xgate signals

6.2.1 xgif_o[127:1]

The Xgate output signal. This output is the result of executing the SIF command by the Xgate RISC processor. The normal connection is to the interrupt inputs of the host processor.

6.2.2 chan_req_i[127:1]

These signals are the interrupt inputs from the peripheral modules that the Xgate services. The eight outputs of the xgswt_o bus may also connect to some subset these inputs.

6.2.3 xgswt_o[7:0]

The [xgswt_o] signal is active low. These signals are activated by the host processor writing to the XGSWT register. These signals are connected to a user selected subset of the [chan_req_i] inputs to trigger the Xgate coprocessor to execute a software routine associated with the selected XGSWT register bit. The host should respond the the setting of one of the [xgif_o] outputs by clearing the set bit in the XGSWT register. Priority of the software routine executed is determined by which [chan_req_i] input the [xgswt_o] signal is connected to.

6.2.4 arst_i

The signal [arst_i] is an asynchronous reset signal that goes to all flops in the COP. It is provided for FPGA implementations and test methodologies that require this function for initialization. Using [arst_i] instead of [wb_rst_i] can result in lower cell-usage and higher performance for a FPGAs implementation because the standard FPGA cell already provides a dedicated asynchronous reset path. Using [wb_rst_i] for an ASIC implementation might synthesize to a smaller module because smaller non_reset flops can be used. Use either [arst_i] or [wb_rst_i], tie the other to a negated state. The active level of [arst_i] is determined by the parameter ARST_LVL that defaults to active low.

6.2.6 xg_sw_irq_o

This output signal is activated whenever the Xgate is activated whenever the encounters a software error. These errors are:

1. Instruction error
2. Instruction Address error
3. Load/Store Word Address error

6.2.7 risc_clk_i

The [risc_clk_i] is the clock input signal used by the RISC core processor.

6.2.8 debug_mode_i

The [debug_mode_i] input can be used by an external source such as a debug module to force the Xgate RISC core processor into debug mode. The exact number of clock cycles between the [debug_mode_i] signal going active and the RISC core stopping instruction execution may vary between one and several clock cycles. If no external debug activation is required then the [debug_mode_i] signal should be tied low.

6.2.9 secure_mode_i

The [secure_mode_i] input is used to limit host read access to the RISC core registers for software security reasons. In an ASIC implementation this pin would be tied to a non-volatile memory bit that is only cleared when all of the Xgate program memory is also cleared. In an FPGA implementation the signal may be tied to a level consistent with the desired application security level. If no software security is required then the [secure_mode_i] signal should be tied low.

6.2.10 scantestmode_i

The [scantestmode_i] input is an optional signal used to put the module into scan test mode. When [scantestmode_i] is active the [startup_osc_clk_i] is replaced by the [wb_clk_i] clock so all register are clocked by a common clock source.

6.3 Xgate Core Parameters

Parameter	Type	Default	Description
ARST_LVL	Bit	1'b0	Asynchronous reset level

MAX_CHANNEL	Integer	127	Number of input an output interrupts
SINGLE_CYCLE	Bit	1'b0	WISHBONE wait state
WB_RD_DEFAULT	Bit	1'b0	Default state for the WISHBONE read bus

6.3.1 ARST_LVL

The asynchronous reset level can be set to either active high (1'b1) or active low (1'b0).

Allowed values: 1'b0, 1'b1

6.3.2 MAX_CHANNEL

The maximun index value for chan_req_i and xgif.

Allowed values: any integer value between 1 and 127

6.3.3 SINGLE_CYCLE

The default operation of the Xgate WISHBONE bus interface is to insert one wait state by delaying the assertion of the wb_ack_o by one wb_clk_i period. Setting the SINGLE_CYCLE parameter generates the wb_ack_o combinationaly so a WISHBONE bus cycle can be completed in one wb_clk_i period.

Allowed values: 1'b0, 1'b1

6.3.4 WB_RD_DEFAULT

The WB_RD_DEFAULT parameter is used to facilitate the connection of the WISHBONE slave data output bus in a "multiplexer interconnection" scheme on an SoC with a minimum of external glue logic. The value of the parameter determines the state of the wbs_dat_o data bus when the Xgate module is not selected for a read operation.

Allowed values: 1'b0, 1'b1

Appendix A

Instruction Set Details

ADC – Add with Carry

Function:

RS1 + RS2 + C => RD

Use binary addition to add RS1, RS2, and Carry and store the sum in the RD register.

Example:

```
                        ; Do 32 bit addition R7:R6 = R5:R4 + R3:R2
                        ; --- Add least significant 16 bit words
ADD   R6, R4, R2        ; R6 = R4 + R2
                        ; --- Add most significant 16 bit words
ADC   R7, R5, R3        ; R7 = R5 + R3 + C
```

CCR Effect:

N	Z	V	C
Δ	Δ	Δ	Δ

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

Z: Set if the result register, RD, is all zero's and Z was set before the operation, otherwise cleared.

V: Set if there is a two's complement overflow, otherwise cleared.

$$(RS1[15] \,\&\, RS2[15] \,\&\, \sim RD[15]_{new}) \mid (\sim RS1[15] \,\&\, \sim RS2[15] \,\&\, RD[15]_{new})$$

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

$$(RS1[15] \,\&\, RS2[15]) \mid (RS1[15] \,\&\, \sim RD[15]_{new}) \mid (RS2[15] \,\&\, \sim RD[15]_{new})$$

Code and Cycles

ASM Code	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

Function:

RS1 + RS2 => RD

Use binary addition to add RS1, RS2, and store the sum in the RD register.

Example:

```
                          ; Do 16 bit addition R6 = R4 + R2
ADD    R6, R4, R2   ; R6 = R4 + R2
```

CCR Effect:

N	Z	V	C
Δ	Δ	Δ	Δ

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

Z: Set if the result register, RD, is all zero's, otherwise cleared.

V: Set if there is a two's complement overflow, otherwise cleared.

$(RS1[15] \& RS2[15] \& \sim RD[15]_{new}) \mid (\sim RS1[15] \& \sim RS2[15] \& RD[15]_{new})$

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

$(RS1[15] \& RS2[15]) \mid (RS1[15] \& \sim RD[15]_{new}) \mid (RS2[15] \& \sim RD[15]_{new})$

Code and Cycles

ASM Code	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

Function:

RD + IMM8:$00 => RD

Use binary addition to add RD, and a signed 8-bit constant and store the sum in the high byte of the RD register.

Example:

```
                                ; --- Add 16 bit immediate
ADDL   R6, #LOWBYTE             ; R6 = R6 + 8 bit immediate
ADDH   R6, #HIGHBYTE            ; R6 = R6 + 16 bit immediate
```

CCR Effect:

N	Z	V	C
Δ	Δ	Δ	Δ

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

Z: Set if the result register, RD, is all zero's, otherwise cleared.

V: Set if there is a two's complement overflow, otherwise cleared.

$(RD[15]_{old} \& IMM8[7] \& \sim RD[15]_{new}) | (\sim RD[15] \& \sim IMM8[7] \& RD[15]_{new})$

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

$(RD[15]old \& IMM8[7]) | (RD[15]old \& \sim RD[15]_{new}) | (IMM8[7] \& \sim RD[15]_{new})$

Code and Cycles

ASM Code	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

Function:

RD +$00:IMM8 => RD

Use binary addition to add RD, and an unsigned 8-bit constant and store the sum in the RD register.

Example:

```
                                ; --- Add 16 bit immediate
ADDL   R6, #LOWBYTE             ; R6 = R6 + 8 bit immediate
ADDH   R6, #HIGHBYTE            ; R6 = R6 + 16 bit immediate
```

CCR Effect:

N	Z	V	C
Δ	Δ	Δ	Δ

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

Z: Set if the result register, RD, is all zero's, otherwise cleared.

V: Set if there is a two's complement overflow, otherwise cleared.

(~RD[7] & ~IMM8[7] & $RD[7]_{new}$) || (RD[7] & IMM8[7] & ~$RD[7]_{new}$)

C: Set if there is a carry from bit 7 to bit 8 of the result, otherwise cleared.

(~$RD[7]_{old}$ & IMM8[7]) | (RD[7] & ~$RD[7]_{new}$) | (~$RD[7]_{new}$ & IMM8[7])

Code and Cycles

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

AND – Logical AND

Function:

RS1 & RS2 => RD

Do a bit wise logical AND between the RS1 and RS2 registers and store the sum in the RD register.

Example:

```
                    ; Bitwise 16 bit AND operation
AND   R6, R4, R2    ; R6 = R4 & R2
AND   R6, R0, R2    ; R6 = $0000
AND   R0, R4, R2    ; Set CCR without storing result
```

CCR Effect:

N	Z	V	C
Δ	Δ	0	-

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

Z: Set if the result register, RD, is all zero's, otherwise cleared.

V: Always cleared.

C: Not affected.

Code and Cycles

ASM Code	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

Function:

RD.H & IMM8:$FF => RD

Do a bit wise logical AND between the 8-bit constant and the high byte of the RD register and store the sum in the RD register. The low byte of the RD register is unaffected.

Example:

```
ANDH  R6, #HIGHBYTE     ; R6.H = R6.H & 8 bit immediate
```

CCR Effect:

N	Z	V	C
Δ	Δ	0	-

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

Z: Set if the 8 bit result, is all zero's, otherwise cleared.

V: Always cleared.

C: Not affected.

Code and Cycles

ASM Code	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

Function:

RD.L + $FF:IMM8 => RD

Da a logical AND function between the 8-bit constant and the low byte of the RD register and store the result in the RD register. The high byte of the RD register is unchanged.

Example:

```
ANDL  R6, #LOWBYTE      ; R6.L = R6.L & 8 bit immediate
```

CCR Effect:

N	Z	V	C
Δ	Δ	0	-

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

Z: Set if the 8 bit result is all zero's, otherwise cleared.

V: Always cleared.

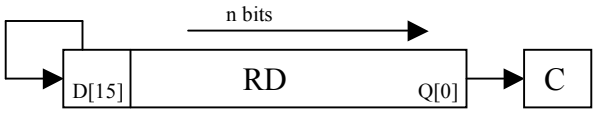
C: Not affected.

Code and Cycles

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

ASR – Arithmetic Shift Right

Function:



n = RS or IMM4

Use binary addition to add RS1, RS2, and Carry and store the sum in the RD register.

Example:

```
ASR   R6, #3      ;
ASR   R6, R1      ;
```

CCR Effect:

N	Z	V	C
Δ	Δ	0	Δ

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

Z: Set if the result register, RD, is all zero's, otherwise cleared.

V: Always cleared.

C: Set if n > 0 and RD[n-1] = 1, if n = 0 then uneffected.

Code and Cycles

ASM Code	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

Function:

If C = 0 then PC + $0002 + (REL9 <<1) => PC

Tests if the Carry Flag and branches if C = 0.

Example:

```
            BCC   CARRY_C     ;
            ;... Carry Set Code
      CARRY_C
            ;... Carry Clear Code
```

CCR Effect:

N	Z	V	C
-	-	-	-

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

Code and Cycles

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

BCS – Branch if Carry Set

Function:

If C = 1 then PC + $0002 + (REL9 <<1) => PC

Tests if the Carry Flag and branches if C = 1.

Example:

```
            BCS   CARRY_S     ;
            ;... Carry Clear Code
      CARRY_S
            ;... Carry Set Code
```

CCR Effect:

N	Z	V	C
-	-	-	-

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

Code and Cycles

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

BEQ – Branch if Equal

Function:

If Z = 1 then PC + $0002 + (REL9 <<1) => PC

Tests if the Zero Flag and branches if Z = 1.

Example:

```
            TST   R1, R2
            BEQ   A_EQ_B      ;
            ;... Not equal Code
      A_EQ_B
            ;... Equal Code
```

CCR Effect:

N	Z	V	C
-	-	-	-

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

Code and Cycles

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

BFEXT – Bit Field Extract

Function:

RS1[(offset+width):offset] => RD[width:0]; 0 => RD[15:width+1)]

width=(RS2[7:4])

offset=(RS2[3:0])

Extracts (width+1) bits from register RS1 starting at index offset and writes them right aligned to register RD. The remaining bits of RD are cleared. If (offset+width)>15 only bits with index [15:offset] get extracted.

Example:

```
LDL   R5, $5a     ;
LDL   R2, $5a     ; width = $5, offset = $a
BFEXT R7, R5, R2  ; R5[15:10] => R7[5:0], 0 => R7[15:6]
```

CCR Effect:

N	Z	V	C
Δ	Δ	0	-

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

Z: Set if the result register, RD, is all zero's, otherwise cleared.

V: Always cleared.

C: Not effected.

Code and Cycles

ASM Code	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

Function:

FirstOne (RS) => RD;

Searches for the index of the most siginificant “1” in the RS register and puts that index into the RD register. The MSB’s of RD are cleared. If RS is $0000 then RD will be set to $0000 and the carry bit will be set. If the only “1” in RS is at index 0 then RD will be set to $0000 and the carry flag will be cleared.

Example:

```
                    ; Find MSB of R2
BFFO   R3, R2       ; R3 = Index of R2 MSB
                    ; --- Branch if R2 was $0000
BCS    NO_ONES      ; R2 had no set bits
```

CCR Effect:

N	Z	V	C
0	Δ	0	Δ

N: Always cleared.

Z: Set if the result register, RD, is all zero’s, otherwise cleared.

V: Always cleared.

C: Set if RS is $0000, otherwise cleared.

Code and Cycles

ASM Code	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

Function:

RS1[width:0] => RD[width+offset:offset]

width=(RS2[7:4])

offset=(RS2[3:0])

Extracts (width+1) bits from register RS1 starting at index 0 and writes them right into register RD at position offset. The remaining bits of RD are not effected. If (offset+width)>15 the upper bits are ignored. If R0 is used as RS1 then a bit field can be cleared.

Example:

```
LDL   R5, $5a     ;
LDL   R2, $5a     ; width = $5, offset = $a
BFINS R7, R5, R2  ; R7[15:10] = R5[5:0]
```

CCR Effect:

N	Z	V	C
Δ	Δ	0	-

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

Z: Set if the result register, RD, is all zero's, otherwise cleared.

V: Always cleared.

C: Not effected.

Code and Cycles

ASM Code	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

Function:

~RS1[width:0] => RD[width+offset:offset]

width=(RS2[7:4])

offset=(RS2[3:0])

Extracts (width+1) bits from register RS1 starting at index 0, invert them, and then writes the result into register RD at position offset. The remaining bits of RD are not effected. If (offset+width)>15 the upper bit are ignored. If R0 is used as RS1 then a bit field can be set.

Example:

```
LDL         R2, $5a     ; width = $5, offset = $a
BFINSI      R7, R5, R2  ; R7[15:10] = ~R5[5:0]
```

CCR Effect:

N	Z	V	C
Δ	Δ	0	-

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

Z: Set if the result register, RD, is all zero's, otherwise cleared.

V: Always cleared.

C: Not effected.

Code and Cycles

ASM Code	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

Function:

(~RS1[width:0] ^ RD[width+offset:offset])=> RD[width+offset:offset]

width=(RS2[7:4])

offset=(RS2[3:0])

Extracts (width+1) bits from register RS1 starting at index 0, performs an XNOR with them and the bits of RD[width+offset:offset], and then writes the result into register RD at position offset. If R0 is used as RS1 then a bit field can be toggled.

Example:

```
LDL         R2, $5a     ; width = $5, offset = $a
BFINSX      R7, R5, R2  ; R7[15:10] = R5[5:0] ^ R7[15:10]
```

CCR Effect:

N	Z	V	C
Δ	Δ	0	-

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

Z: Set if the result register, RD, is all zero's, otherwise cleared.

V: Always cleared.

C: Not effected.

Code and Cycles

ASM Code	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

Function:

If N ^ V = 0 then PC + $0002 + (REL9 <<1) => PC

Branch instruction to compare signed numbers.

Example:

```
                        ; Branch if RS1 GTE RS2
SUB    R0, R1, R2       ;
BGE    REL9
```

CCR Effect:

N	Z	V	C
-	-	-	-

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

Code and Cycles

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

BGT – Branch if Greater than Zero

Function:

If Z | (N ^ V) = 0 then PC + $0002 + (REL9 <<1) => PC

Branch instruction to compare signed numbers.

Example:

```
                        ; Branch if RS1 GT RS2
SUB    R0, R1, R2       ;
BGT    REL9
```

CCR Effect:

N	Z	V	C
-	-	-	-

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

Code and Cycles

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

BHI – Branch if Higher

Function:

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

Branch instruction to compare unsigned numbers.

Example:

```
                        ; Branch if RS1 GT RS2
SUB    R0, R1, R2       ;
BHI    REL9
```

CCR Effect:

N	Z	V	C
-	-	-	-

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

Code and Cycles

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

BHS – Branch if Higher or Same

Function:

If C = 0 then PC + $0002 + (REL9 <<1) => PC

--- Compiler alias to the command BCC REL9

Branch instruction to compare unsigned numbers.

Example:

```
                        ; Branch if RS1 GTE RS2
        SUB   R0, R1, R2  ;
        BHS   REL9
```

CCR Effect:

N	Z	V	C
-	-	-	-

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

Code and Cycles

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

BITH – Bit Test Immediate 8 Bit Constant high byte

Function:

RD.H & IMM8 => None

Preforms a bit wise logical AND operation between the IMM8 and the high byte of the RD register. Used to detect Set bits in the RD register. RD is not changed and only the condition bits are updated after the instruction is completed.

Example:

```
                                ; --- Test the high byte of R6
LDH    R6, $80                  ; R6 = $80??
BITH   R6, #HIGHBYTE            ; R6 = R6
```

CCR Effect:

N	Z	V	C
Δ	Δ	0	-

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

Z: Set if the 8 bit result is $00, otherwise cleared.

V: Always set to zero.

C: Not affected.

Code and Cycles

ASM Code	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

Function:

RD.L & IMM8 => None

Preforms a bit wise logical AND operation between the IMM8 and the low byte of the RD register. Used to detect set bits in the RD register. RD is not changed and only the condition bits are updated after the instruction is completed.

Example:

```
                                ; --- Test the low byte of R6
        BITL   R6, #LOWBYTE     ; R6 = R6
```

CCR Effect:

N	Z	V	C
Δ	Δ	0	-

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

Z: Set if the 8 bit result is $00, otherwise cleared.

V: Always set to zero.

C: Not affected.

Code and Cycles

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

BLE – Branch if Less or Equal Zero

Function:

If Z | (N ^ V) = 1 then PC + $0002 + (REL9 <<1) => PC

Branch instruction to compare signed numbers.

Example:

```
                        ; Branch if RS1 LTE RS2
SUB    R0, R1, R2       ;
BLE    REL9
```

CCR Effect:

N	Z	V	C
-	-	-	-

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

Code and Cycles

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

BLO – Branch if Less Than

Function:

If C = 1 then PC + $0002 + (REL9 <<1) => PC

--- Compiler alias to the command BCS REL9

Branch instruction to compare unsigned numbers.

Example:

```
                        ; Branch if RS1 LT RS2
SUB    R0, R1, R2   ;
BLO    REL9
```

CCR Effect:

N	Z	V	C
-	-	-	-

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

Code and Cycles

ASM Code	Address Mode	Machine Code								Cycles
BLO REL9	REL9	0	0	1	0	0	0	1	REL9	PP/P

BLS – Branch Lower or Same

Function:

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

Branch instruction to compare unsigned numbers.

Example:

```
                        ; Branch if RS1 LTE RS2
SUB    R0, R1, R2       ;
BLS    REL9
```

CCR Effect:

N	Z	V	C
-	-	-	-

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

Code and Cycles

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

BLT – Branch Lower Than Zero

Function:

If N ^ V = 1 then PC + $0002 + (REL9 <<1) => PC

Branch instruction to compare signed numbers.

Example:

```
                        ; Branch if RS1 LT RS2
SUB    R0, R1, R2       ;
BLS    REL9
```

CCR Effect:

N	Z	V	C
-	-	-	-

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

Code and Cycles

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

BMI – Branch if Minus

Function:

If N = 1 then PC + $0002 + (REL9 <<1) => PC

Tests the Sign Flag and branch if N = 1.

Example:

```
                        ; Branch if RS1 is Negative
SUB    R0, R1, R0   ;
BMI    REL9
```

CCR Effect:

N	Z	V	C
-	-	-	-

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

Code and Cycles

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

BNE – Branch if Not Equal

Function:

If Z = 0 then PC + $0002 + (REL9 <<1) => PC

Tests the Zero Flag and branch if Z = 0.

Example:

```
                        ; Branch if RS1 NE RS2
SUB    R0, R1, R2       ;
BNE    REL9
```

CCR Effect:

N	Z	V	C
-	-	-	-

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

Code and Cycles

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

BPL – Branch if Plus

Function:

If N = 0 then PC + $0002 + (REL9 <<1) => PC

Tests the Sign Flag and branch if N = 0.

Example:

```
                        ; Branch if RS1 Postive
ADD   R0, R1, R0  ;
BLS   REL9
```

CCR Effect:

N	Z	V	C
-	-	-	-

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

Code and Cycles

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

BRA – Branch Always

Function:

PC + $0002 + (REL10 << 1) => PC

Branch unconditionaly.

Example:

```
BRA    REL10
```

CCR Effect:

N	Z	V	C
-	-	-	-

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

Code and Cycles

ASM Code	Address Mode	Machine Code							Cycles
BRA REL10	REL10	0	0	1	1	1	1	REL10	PP

BRK – Break

Function:

Put Xgate into Debug Mode.

Example:

```
BRK   ; Goto debug mode and stop instruction execution
```

CCR Effect:

N	Z	V	C
-	-	-	-

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

Code and Cycles

ASM Code	Address Mode	Machine Code															Cycles
BRK	INH	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PAff

BVC – Branch if Overflow Cleared

Function:

If V = 0 then PC + $0002 + (REL9 << 1) => PC

Tests the Overflag Flag and branch if V = 0.

Example:

```
                        ; Branch if RS1 LTE RS2
SUB    R0, R1, R2       ;
BVC    REL9
```

CCR Effect:

N	Z	V	C
-	-	-	-

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

Code and Cycles

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

BVS – Branch if Overflow Set

Function:

If V = 1 then PC + $0002 + (REL9 <<1) => PC

Tests the Overflow Flag and branch if V = 1.

Example:

```
                        ; Branch if RS1 LTE RS2
SUB    R0, R1, R2       ;
BVS    REL9
```

CCR Effect:

N	Z	V	C
-	-	-	-

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

Code and Cycles

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

CMP – Compare

Function:

RS1 - RS2 => None

--- Compiler alias to the command SUB R0, RS1 RS2

Subtract the contents of RS2 from RS1 using binary subtraction and update the condition code registers.

Example:

```
                        ; --- Compare two 16 bit words
CMP   R5, R3            ; R5 - R3
```

CCR Effect:

N	Z	V	C
Δ	Δ	Δ	Δ

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

Z: Set if the result register, RD, is all zero's, otherwise cleared.

V: Set if there is a two's complement overflow, otherwise cleared.

$(RS1[15] \& \sim RS2[15] \& \sim RD[15]_{new}) \mid (\sim RS1[15] \& RS2[15] \& RD[15]_{new})$

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

$(\sim RS1[15] \& RS2[15]) \mid (\sim RS1[15] \& RD[15]_{new}) \mid (RS2[15] \& RD[15]_{new})$

Code and Cycles

ASM Code	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

Function:

RS.L – IMM8 => None

Subtract the IMM8 value from the low byte of RS using binary subtraction and update the condition code registers.

Example:

```
                        ; Compare low byte of R7 to $55
        CMPL   R7, #$55 ; R5.L - $55
```

CCR Effect:

N	Z	V	C
Δ	Δ	Δ	Δ

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

Z: Set if the 8 bit result is all zero's, otherwise cleared.

V: Set if there is a two's complement overflow, otherwise cleared.

(RS[7] & ~IMM8[7] & ~result[7]) | (~RS[7] & IMM8[7] & result[15])

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

(~RS[7] & RS2[7]) | (~RS[7] & result[7]$_{new}$) | (RS[7] & result[7])

Code and Cycles

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

COM – One's Complement

Function:

~RS => RD

~RD => RD

--- Compiler alias to the command XNOR RD, R0 RS

--- Compiler alias to the command XNOR RD, R0 RD

Do the one's complement(bit wise invert) on the RS and store the result in the RD register..

Example:

```
COM    R6, R4,      ; R6 = ~R4
COM    R7           ; R7 = ~R7
```

CCR Effect:

N	Z	V	C
Δ	Δ	0	-

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

Z: Set if the result is all zero's, otherwise cleared.

V: Always cleared.

C: Not affected.

Code and Cycles

ASM Code	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

Function:

RS1 - RS2 – C => None

--- Compiler alias to the command SBC R0, RS1 RS2

Subtract the Carry bit and the contents of RS2 from RS1 using binary subtraction, and update the condition code registers.

Example:

```
                ; Do 32 bit subtraction R7:R6 = R5:R4 - R3:R2
                ; --- Subtract least significant 16 bit words
SUB   R6, R4, R2  ; R6 = R4 - R2
                ; --- Subtract most significant 16 bit words
SPC   R5, R3      ; R5 - R3 - C
```

CCR Effect:

N	Z	V	C
Δ	Δ	Δ	Δ

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

Z: Set if the result register, RD, is all zero's, otherwise cleared.

V: Set if there is a two's complement overflow, otherwise cleared.

(RS1[15] & ~RS2[15] & ~result[15]) | (~RS1[15] & RS2[15] & result[15])

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

(~RS1[15] & RS2[15]) | (~RS1[15] & result[15]) | (RS2[15] & result[15])

Code and Cycles

ASM Code	Address Mode	Machine Code												Cycles
SPC RS1,RS2	TRI	0	0	0	1	1	0	0	0	RS1	RS2	0	1	P

CPCH – Compare Immediate with bit constant with Carry high byte

Function:

RS.H – IMM8 - C => None

Subtract the contents of RS2 from RS1 using binary subtraction and update the condition code registers.

Example:

```
                        ; Do 16 bit compare of constant to R2
CMPL   R6, #LOWBYTE      ; Update Condition Code Register
CMPH   R6, #HIGHBYTE     ; Update Condition Code Register
BCS    IS_EQUAL
```

CCR Effect:

N	Z	V	C
Δ	Δ	Δ	Δ

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

Z: Set if the result is all $00 and Z was set before this instruction, otherwise cleared.

V: Set if there is a two's complement overflow, otherwise cleared.

(RS[15] & ~IMM8[7] & ~result[15]) | (~RS[15] & IMM8[7] & result[15])

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

(~RS[15] & IMM8[7]) | (~RS1[15] & result[15]) | (IMM8[7] & result[15])

Code and Cycles

ASM Code	Address Mode	Machine Code							Cycles
CPCH RS,#IMM8	TRI	1	1	0	1	1	RS	IMM8	P

CSEM – Clear Semaphore Bit

Function:

0 => XGSEM[IMM3]

0 => XGSEM[RD[2:0]]

Unlocks a semaphore bit that was set by the RISC core.

Example:

```
                        ; Do 32 bit addition R7:R6 = R5:R4 + R3:R2
                        ; --- Add least significant 16 bit words
ADD    R6, R4, R2       ; R6 = R4 + R2
                        ; --- Add most significant 16 bit words
ADC    R7, R5, R3       ; R7 = R5 + R3 + C
```

CCR Effect:

N	Z	V	C
-	-	-	-

N: Not affected.

Z: Not affected.

V: Not affected.

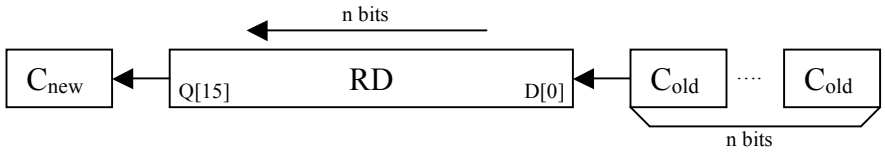
C: Not affected.

Code and Cycles

ASM Code	Address Mode	Machine Code														Cycles
CSEM #IMM3	IMM3	0	0	0	0	0	IMM3	1	1	1	1	0	0	0	0	P
CSEM RS	MON	0	0	0	0	0	RS	1	1	1	1	0	0	0	1	P

CSL – Logical Shift Left with Carry

Function:



n = RS or IMM4; $0 \leq n \leq 16$

Left shift the contents of the RD register n positions. The lower n bits of RD are filled with the contents of the carry register. At the end of the operation the carry register will be updated with the contents of RD[16-n] when n is greater than zero.

Example:

```
                        ; Do 16 bit compare of constant to R2
CMPL   R6, #LOWBYTE      ; Update Condition Code Register
CMPH   R6, #HIGHBYTE     ; Update Condition Code Register
BCS    IS_EQUAL
```

CCR Effect:

N	Z	V	C
Δ	Δ	Δ	Δ

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

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

V: Set if there is a two's complement overflow, otherwise cleared.

$RD[15]_{old}$ ^ $RD[15]_{new}$

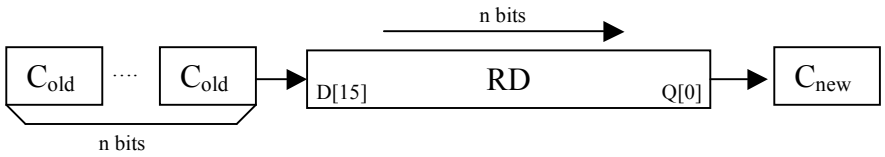
C: Set if (n > 0) and RD[16-n] = 1. If (n = 0) then there is no change in C.

Code and Cycles

ASM Code	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

Function:



n = RS or IMM4; $0 \leq n \leq 16$.

Example:

```
                        ; Do 16 bit compare of constant to R2
CMPL   R6, #LOWBYTE      ; Update Condition Code Register
CMPH   R6, #HIGHBYTE     ; Update Condition Code Register
BCS    IS_EQUAL
```

CCR Effect:

N	Z	V	C
Δ	Δ	Δ	Δ

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

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

V: Set if there is a two's complement overflow, otherwise cleared.

$RD[15]_{old}$ ^ $RD[15]_{new}$

C: Set if (n > 0) and RD[n-1] = 1. If (n = 0) then there is no change in C.

Code and Cycles

ASM Code	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

Function:

PC + $0002 => RD; RD => PC

Branches to the address in RD and stores the return address to RD.

Example:

```
            TFR   R5, PC      ; Save the return address in R5
            BRA   SUB_1       ; Goto the subroutine code
            ; More Code
      SUB_1
            ; Subroutine Code
            JAL   RD          ; Return to the address after the BRA
```

CCR Effect:

N	Z	V	C
-	-	-	-

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

Code and Cycles

ASM Code	Address Mode	Machine Code														Cycles
JAL RD	IMM4	0	0	0	0	0	RD	1	1	1	1	0	1	1	0	PP

LDB – Load Byte From Memory Low Byte

Function:

M[RB, $OFFS5] => RD.L; $00 => RD.H ($OFFS5 = unsigned offset)

M[RB, RI] => RD.L; $00 => RD.H

M[RB, RI+] => RD.L; $00 => RD.H; RI + 1 => RI (Post increment)

RI – 1 => RI ;M[RS, -RI] => RD.L; $00 => RD.H (Pre decrement)

Loads the low byte of RD from memory and clear the high byte of RD.

Example:

```
LDB   R5, (R1,#9) ; Load R5 from the address at R1+9
```

CCR Effect:

N	Z	V	C
-	-	-	-

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

Code and Cycles

ASM Code	Address Mode	Machine Code										Cycles
LDB RD, (RB, #IMM8)	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+)[1]	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

Note:

If the RB == RD the final value of RD will be M[RB,RI]; RD will not be incremented.

LDH – Load Immediate 8 Bit Constant High Byte

Function:

IMM8 => RD.H

Put an 8 bit constant into the high byte of the RD register. The low byte of RD is unaffected.

Example:

```
LDL   R5, $55     ; R5 = $0055
LDH   R5, $AA     ; R5 = $AA55
```

CCR Effect:

N	Z	V	C
-	-	-	-

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

Code and Cycles

ASM Code	Address Mode	Machine Code							Cycles
LDH #IMM8	IMM8	1	1	1	1	1	RD	IMM8	P

LDL – Load Immediate 8 Bit Constant Low Byte

Function:

IMM8 => RD.L; $00 => RD.H

Put an 8 bit constant into the low byte of the RD register. The high byte of RD is set to $00.

Example:

```
LDL   R5, $55     ; R5 = $0055
LDH   R5, $AA     ; R5 = $AA55
```

CCR Effect:

N	Z	V	C
-	-	-	-

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

Code and Cycles

ASM Code	Address Mode	Machine Code							Cycles
LDL #IMM8	IMM8	1	1	1	1	0	RD	IMM8	P

LDW – Load Word From Memory

Function:

M[RB, $OFFS5] => RD

M[RB, RI] => RD

M[RB, RI+] => RD; RI + 2 => RI (Post increment)

M[RS, -RI] => RD; RI – 2 => RI (Pre decrement)

Loads the RD register from memory.

Example:

```
LDW   R5, (R1,#2) ; Load R5 from the address at R1+2
                  ;  Immediate Constant should not exceed 30
```

CCR Effect:

N	Z	V	C
-	-	-	-

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

Code and Cycles

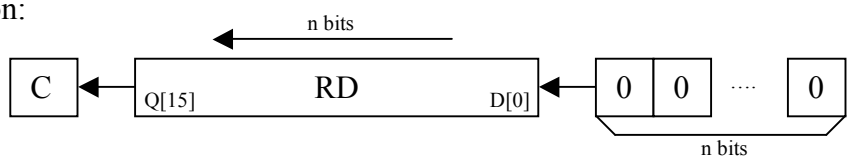
ASM Code	Address Mode	Machine Code										Cycles
LDW RD, (RB, #IMM8)	IDO5	0	1	0	0	1	RD	RB	OFFS5			PR
LDW RD, (RS, RI)	IDR	0	1	1	0	1	RD	RB	RI	0	0	PR
LDW RD, (RS, RI+)[1]	IDR+	0	1	1	0	1	RD	RB	RI	0	1	PR
LDW RD, (RS, -RI)	-IDR	0	1	1	0	1	RD	RB	RI	1	0	PR

Note:

1) If the RB == RD the final value of RD will be M[RB,RI]; RD will not be incremented.

LSL – Logical Shift Left

Function:



Use binary addition to add RS1, RS2, and Carry and store the sum in the RD register.

Example:

```
LSL   R6, #5      ; R6 = R6 left shifted by 5 bits
                  ;  R6[4:0] = 0
```

CCR Effect:

N	Z	V	C
Δ	Δ	Δ	Δ

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

Z: Set if the result register, RD, is all zero's, otherwise cleared.

V: .

$RD[15]_{old}$ ^ $RD[15]_{new}$.

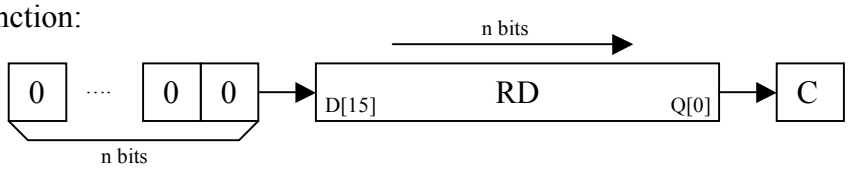
C: Set if n > 0 and RD[16-n] = 1, if n = 0 then uneffected.

Code and Cycles

ASM Code	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

Function:



Use binary addition to add RS1, RS2, and Carry and store the sum in the RD register.

Example:

```
LSR   R6, #5      ; R6 = R6 right shifted by 5 bits
                  ;  R6[15:11] = 0
```

CCR Effect:

N	Z	V	C
Δ	Δ	Δ	Δ

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

Z: Set if the result register, RD, is all zero's, otherwise cleared.

V: Set if RD[15] changed.

$RD[15]_{old}$ ^ $RD[15]_{new}$.

C: Set if n > 0 and RD[16-n] = 1, if n = 0 then uneffected.

Code and Cycles

ASM Code	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

Function:

RS => RD

--- Compiler alias to the command OR RD, R0 RS

Copies the contents of the RS register to the RD register.

Example:

```
MOV   R6, R4      ; R6 = R4
```

CCR Effect:

N	Z	V	C
Δ	Δ	0	-

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

Z: Set if the result register, RD, is all zero's, otherwise cleared.

V: Always cleared.

C: Not affected

Code and Cycles

ASM Code	Address Mode	Machine Code												Cycles
MOV RD,RS	TRI	0	0	0	1	0	RD	0	0	0	RS2	1	0	P

NEG – Two's Complement

Function:

-RS => RD

-RD => RD

--- Compiler alias to the command SUB RD, R0 RS

--- Compiler alias to the command SUB RD, R0 RD

Calculate the two's complement of a general purpose register.

Example:

```
NEG   R6, R4      ; R6 = -R4
NEG   R7          ; R7 = -R7
```

CCR Effect:

N	Z	V	C
Δ	Δ	Δ	Δ

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

Z: Set if the result register, RD, is all zero's, otherwise cleared.

V: Set if there is a two's complement overflow, otherwise cleared.

$(RS1[15] \& \sim RS2[15] \& \sim RD[15]_{new}) \mid (\sim RS1[15] \& RS2[15] \& RD[15]_{new})$

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

$(\sim RS1[15] \& RS2[15]) \mid (\sim RS1[15] \& RD[15]_{new}) \mid (RS2[15] \& RD[15]_{new})$

Code and Cycles

ASM Code	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

Function:

PC + $0002 => PC

Consume one cycle of RISC CPU time without doing anything.

Example:

```
NOP          ; Stall for one CPU cycle
```

CCR Effect:

N	Z	V	C
-	-	-	-

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

Code and Cycles

ASM Code	Address Mode	Machine Code															Cycles
NOP	INH	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	P

OR – Logical OR

Function:

RS1 | RS2 => RD

Do a bit wise logical OR operation between the RS1 register and the RS2 register and put the result in the RD register.

Example:

```
                        ; Do 16 bit logical OR
OR      R6, R4, R2      ; R6 = R4 | R2
OR      R6, R0, R4      ; R6 = R4
```

CCR Effect:

N	Z	V	C
Δ	Δ	0	-

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

Z: Set if the result register, RD, is all zero's, otherwise cleared.

V: Always cleared.

C: Not affected

Code and Cycles

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

ORH – Logical OR 8 Bit Immediate Constant High Byte

Function:

RD.H | IMM8 => RD.H

Do a bit wise logical OR operation between the 8 bit immediate value and the RD high byte and put the result in the RD high byte register.

Example:

```
                        ; Do 8 bit logical OR
ORH   R6, #$AA          ; R6 = R6 | $AA00
```

CCR Effect:

N	Z	V	C
Δ	Δ	0	-

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

Z: Set if the 8 bit result is $00, otherwise cleared.

V: Always cleared.

C: Not affected

Code and Cycles

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

ORL – Logical OR 8 Bit Immediate Constant Low Byte

Function:

RD.H | IMM8 => RD.H

Do a bit wise logical OR operation between the 8 bit immediate value and the RD low byte and put the result in the RD low byte register.

Example:

```
                        ; Do 8 bit logical OR
ORL   R6, #$AA          ; R6 = R6 | $0055
```

CCR Effect:

N	Z	V	C
Δ	Δ	0	-

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

Z: Set if the 8 bit result is $00, otherwise cleared.

V: Always cleared.

C: Not affected

Code and Cycles

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

PAR – Calculate Parity

Function:

RD.H | IMM8 => RD.H

Set the Carry flag if the number of one's in the RD register is odd..

Example:

```
                    ; Do 16 bit parity
PAR    R6           ; R6 = R6; Carry = Odd or Even
```

CCR Effect:

N	Z	V	C
Δ	Δ	0	Δ

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

Z: Set if the RD is $0000, otherwise cleared.

V: Always cleared.

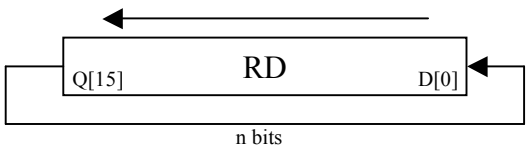
C: Set if RD has an odd number of 1's, otherwise cleared.

Code and Cycles

ASM Code	Address Mode	Machine Code														Cycles
PAR RD	IMM8	0	0	0	0	0	RD	1	1	1	1	0	1	0	1	P

ROL – Rotate Left

Function:



n = RS or IMM4.

Set the Carry flag if the number of one's in the RD register is odd..

Example:

```
ROL   R6, #5      ; R6[15:0] = R6[10:0] R6[15:11]
```

CCR Effect:

N	Z	V	C
Δ	Δ	0	-

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

Z: Set if the RD is $0000, otherwise cleared.

V: Always cleared.

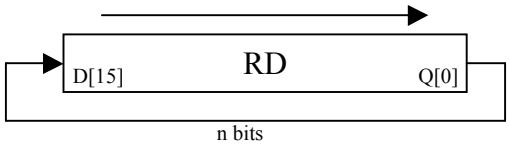
C: No affected.

Code and Cycles

ASM Code	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

Function:



n = RS or IMM4.

Example:

```
ROR   R6,#3 ; R6 = R6 ROR 3 places
```

CCR Effect:

N	Z	V	C
Δ	Δ	0	-

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

Z: Set if the RD is $0000, otherwise cleared.

V: Always cleared.

C: No affected.

Code and Cycles

ASM Code	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

Function:

Completes the current thread of program instructions and remain idle till the hardware scheduler starts a new thread.

Example:

```
                    ;
        RTS         ; Channel thread complete
```

CCR Effect:

N	Z	V	C
-	-	-	-

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

Code and Cycles

ASM Code	Address Mode	Machine Code															Cycles
RTS	INH	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	P

SBC – Subtract with Carry

Function:

RS1 - RS2 – C => RD

Subtract the Carry bit and the contents of RS2 from RS1 using binary subtraction, store the result in RD and update the condition code registers.

Example:

```
                ; Do 32 bit subtraction R7:R6 = R5:R4 - R3:R2
                ; --- Subtract least significant 16 bit words
SUB    R6, R4, R2  ; R6 = R4 - R2
                ; --- Subtract most significant 16 bit words
SBC    R7, R5, R3  ; R7 = R5 - R3 - C
```

CCR Effect:

N	Z	V	C
Δ	Δ	Δ	Δ

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

Z: Set if the result register, RD, is all zero's and Z was set before the operation, otherwise cleared.

V: Set if there is a two's complement overflow, otherwise cleared.

$$(RS1[15] \,\&\, \sim RS2[15] \,\&\, \sim RD[15]_{new}) \mid (\sim RS1[15] \,\&\, RS2[15] \,\&\, RD[15]_{new})$$

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

$$(\sim RS1[15] \,\&\, RS2[15]) \mid (\sim RS1[15] \,\&\, RD[15]_{new}) \mid (RS2[15] \,\&\, RD[15]_{new})$$

Code and Cycles

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

SSEM – Set Semaphore Bit

Function:

1 => XGSEM[IMM3]

1 => XGSEM[RD[2:0]]

Locks a semaphore bit that was set by the RISC core.

Example:

```
NOT_MINE
      SSEM  R6          ; Attempt to set semaphore bit
      BCC   NOT_MINE    ; --- Wait till RISC has semaphore bit
      SSEM  #3          ; Attempt to set semaphore bit 3
```

CCR Effect:

N	Z	V	C
-	-	-	Δ

N: Not affected.

Z: Not affected.

V: Not affected.

C: Set if semaphore is locked by RISC core.

Code and Cycles

ASM Code	Address Mode	Machine Code														Cycles
SSEM #IMM3	IMM3	0	0	0	0	0	IMM3	1	1	1	1	0	0	1	0	P
SSEM RS	MON	0	0	0	0	0	RS	1	1	1	1	0	0	1	1	P

SEX – Sign Extend

Function:

RD.L => RD

The 8 bit two's complement number stored in RD.L is sign extended to fill the RD register.

Example:

```
SEX   R6           ; --- Add 8 bit number to 16 bit number
ADD   R7, R6, R3   ; R7 = R6 + R3
```

CCR Effect:

N	Z	V	C
Δ	Δ	0	-

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

Z: Set if the result register, RD, is all zero's, otherwise cleared.

V: Always cleared.

C: Not affected.

Code and Cycles

ASM Code	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

Function:

1 => XGIF[XGCHID]

1 => XGIF[RS[6:0]]

Set the output interrupt flag.

Example:

```
SIF   R6    ; Signal Host to release semaphore register?
SIF         ; Signal Host that thread is complete
RTS         ; Allow Xgate to accept new input interrupt
```

CCR Effect:

N	Z	V	C
-	-	-	-

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

Code and Cycles

ASM Code	Address Mode	Machine Code																Cycles
SIF	INH	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	
SIF RS	MON	0	0	0	0	0	RS			1	1	1	1	0	1	1	1	P

STB – Store Byte To Memory Low Byte

Function:

RS.L M[RB, $OFFS5]

RS.L M[RB, RI]

RS.L M[RB, +RI]; RI + 1 => RI

RS.L M[RS, -RI]; RI – 1 => RI

Stores the low byte of register RS to memory.

Example:

```
STB   R5, (R1,#9) ; Store R5[7:0] to the address at R1+9
```

CCR Effect:

N	Z	V	C
-	-	-	-

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

Code and Cycles

ASM Code	Address Mode	Machine Code										Cycles
STB RD, (RB, #IMM8)	IDO5	0	1	0	1	0	RD	RB	OFFS5			PR
STB RD, (RS, RI)	IDR	0	1	1	1	0	RD	RB	RI	0	0	PR
STB RD, (RS, RI+)	IDR+	0	1	1	1	0	RD	RB	RI	0	1	PR
STB RD, (RS, -RI)	-IDR	0	1	1	1	0	RD	RB	RI	1	0	PR

STW – Store Word To Memory

Function:

RS => M[RB, $OFFS5]

RS => M[RB, RI]

RS => M[RB, +RI]; RI + 2 => RI

RS => M[RS, -RI]; RI – 2 => RI

Stores the contents of the RD register to memory.

Example:

```
STB   R5, (R1,#8) ; Store to the address at R1+8
```

CCR Effect:

N	Z	V	C
-	-	-	-

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

Code and Cycles

ASM Code	Address Mode	Machine Code										Cycles
STW RD, (RB, #IMM8)	IDO5	0	1	0	1	1	RD	RB	OFFS5			PR
STW RD, (RS, RI)	IDR	0	1	1	1	1	RD	RB	RI	0	0	PR
STW RD, (RS, RI+)	IDR+	0	1	1	1	1	RD	RB	RI	0	1	PR
STW RD, (RS, -RI)	-IDR	0	1	1	1	1	RD	RB	RI	1	0	PR

SUB – Subtract without Carry

Function:

RS1 - RS2 => RD

Subtract the contents of RS2 from RS1 using binary subtraction, store the result in RD and update the condition code registers.

Example:

```
                ; Do 32 bit subtraction R7:R6 = R5:R4 - R3:R2
                ; --- Subtract least significant 16 bit words
SUB    R6, R4, R2  ; R6 = R4 - R2
                ; --- Subtract most significant 16 bit words
SBC    R7, R5, R3  ; R7 = R5 - R3 - C
```

CCR Effect:

N	Z	V	C
Δ	Δ	Δ	Δ

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

Z: Set if the result register, RD, is all zero's, otherwise cleared.

V: Set if there is a two's complement overflow, otherwise cleared.

$$(RS1[15] \,\&\, \sim RS2[15] \,\&\, \sim RD[15]_{new}) \mid (\sim RS1[15] \,\&\, RS2[15] \,\&\, RD[15]_{new})$$

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

$$(\sim RS1[15] \,\&\, RS2[15]) \mid (\sim RS1[15] \,\&\, RD[15]_{new}) \mid (RS2[15] \,\&\, RD[15]_{new})$$

Code and Cycles

ASM Code	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

Function:

RD - IMM8:$00 => RD

Subtract a signed 8 bit immediate constant from the high byte of RD and store the result in the high byte of the RD register.

Example:

```
                                ; --- Subtract 16 bit immediate
SUBL   R6, #LOWBYTE             ; R6 = R6 - 8 bit immediate
SUBH   R6, #HIGHBYTE            ; R6 = R6 - 16 bit immediate
```

CCR Effect:

N	Z	V	C
Δ	Δ	Δ	Δ

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

Z: Set if the result register, RD, is all zero's, otherwise cleared.

V: Set if there is a two's complement overflow, otherwise cleared.

$(RD[15]_{old} \& \sim IMM8[7] \& \sim RD[15]_{new}) \mid (\sim RD[15]_{old} \& IMM8[7] \& RD[15]_{new})$

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

$(\sim RD[15]_{old} \& IMM8[7] \mid (\sim RD[15]_{old} \& RD[15]_{new}) \mid (IMM8[7] \& RD[15]_{new})$

Code and Cycles

ASM Code	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

Function:

RD - $00:IMM8 => RD

Subtract an 8 bit immediate constant from the RD register and store the result in the RD register. (All bits in the RD register may be affected by this operation.)

Example:

```
                                ; --- Subtract 16 bit immediate
SUBL   R6, #LOWBYTE             ; R6 = R6 - 8 bit immediate
SUBH   R6, #HIGHBYTE            ; R6 = R6 - 16 bit immediate
```

CCR Effect:

N	Z	V	C
Δ	Δ	Δ	Δ

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

Z: Set if the result register, RD, is all zero's, otherwise cleared.

V: Set if there is a two's complement overflow, otherwise cleared.

$(\sim RD[15]_{old} \& RD[15]_{new})$

C: Set if there is a carry from bit 7 to bit 8 of the result, otherwise cleared.

$(\sim RD[7]_{old} \& IMM8[7]) \mid (RD[7] \& \sim RD[7]_{new}) \mid (\sim RD[7]_{new} \& IMM8[7])$

Code and Cycles

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

TFR – Transfer from and To Special Registers

Function:

TFR RD, CCR: CCR + RD[3:0] => RD; 0 => RD[15:4]

TFR CCR, RD: RD[3:0] => CCR

TFR RD,PC: PC + 4 => RD

Copies the contents of one RISC core register to another RISC core register.

Example:

```
            TFR   R5, PC      ; Save the return address in R5
            BRA   SUB_1       ; Goto the subroutine code
            ; More Code
      SUB_1
            ; Subroutine Code
            JAL   RD          ; Return to the address after the BRA
```

CCR Effect:

N	Z	V	C
-	-	-	-

N: Not affected.

Z: Not affected.

V: Not affected.

C: Not affected.

Code and Cycles

ASM Code	Address Mode	Machine Code														Cycles
TFR RD,CCR	MON	0	0	0	0	0	RD	1	1	1	1	1	0	0	0	PP
TFR CCR,RS	MON	0	0	0	0	0	RS	1	1	1	1	1	0	0	1	
TFR RD,PC	MON	0	0	0	0	0	RD	1	1	1	1	1	0	1	0	

TST – Test Register

Function:

RS - 0 => None

--- Compiler alias to the command SUB R0, RS, R0

Subtract zero from the contents of the RS register and update the condition code registers.

Example:

```
TST   R6
BEQ   IS_ZERO     ; R6 == 0
BMI   IS_NEG      ; R6 is a negative number
```

CCR Effect:

N	Z	V	C
Δ	Δ	Δ	Δ

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

Z: Set if the result register, RD, is all zero's, otherwise cleared.

V: Set if there is a two's complement overflow, otherwise cleared.

$(RS1[15] \& \sim RS2[15] \& \sim RD[15]_{new}) \mid (\sim RS1[15] \& RS2[15] \& RD[15]_{new})$

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

$(\sim RS1[15] \& RS2[15]) \mid (\sim RS1[15] \& RD[15]_{new}) \mid (RS2[15] \& RD[15]_{new})$

Code and Cycles

ASM Code	Address Mode	Machine Code														Cycles
TST RS	TRI	0	0	0	1	1	0	0	0	RS	0	0	0	0	0	P

XNOR – Logical Exclusive NOR

Function:

~(RS1 ^ RS2) => RD

Do a bit wise logical XNOR between the RS1 register and the RS2 register and store the result in the RD register.

Example:

```
XNOR  R6, R0, R6  ; R6 = One's complement (invert) R6
XNOR  R6, R0, R0  ; R6 = $FFFF
```

CCR Effect:

N	Z	V	C
Δ	Δ	0	-

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

Z: Set if the result register, RD, is all zero's, otherwise cleared.

V: Always cleared.

C: Not affected.

Code and Cycles

ASM Code	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 Constant High Byte

Function:

~(RD.H ^ IMM8) => RD.H

Do a bit wise logical XNOR between the 8-bit constant and the high byte of the RD register and store the result in the RD register. The low byte of RD is not affected.

Example:

```
XNORH R6, #0      ; R6[15:8] = One's complement (invert) R6
```

CCR Effect:

N	Z	V	C
Δ	Δ	0	-

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

Z: Set if the result register, RD, is all zero's, otherwise cleared.

V: Always cleared.

C: Not affected.

Code and Cycles

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

XNORL – Logical Exclusive NOR Constant Low Byte

Function:

~(RD.L ^ IMM8) => RD.L

Do a bit wise logical XNOR between the 8-bit constant and the low byte of the RD register and store the result in the RD register. The high byte of RD is not affected.

Example:

```
XNORL R6, #0      ; R6[7:0] = One's complement (invert) R6
```

CCR Effect:

N	Z	V	C
Δ	Δ	0	-

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

Z: Set if the result register, RD, is all zero's, otherwise cleared.

V: Always cleared.

C: Not affected.

Code and Cycles

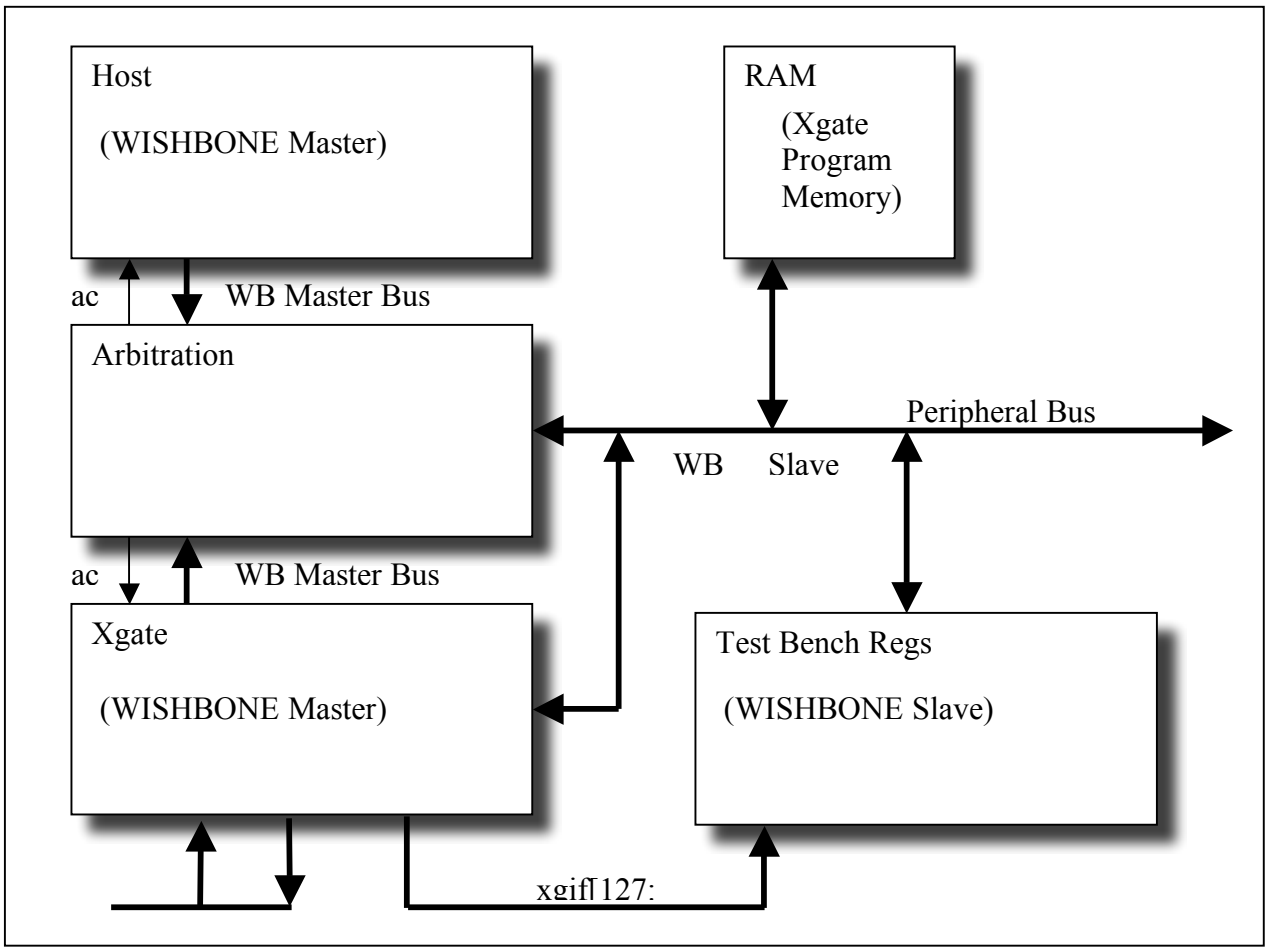
ASM Code	Address Mode	Machine Code							Cycles
XNORL RD,#IMM8	IMM8	1	0	1	1	0	RD	IMM8	P

Appendix B

Test Bench

Test Bench Overview

This is a simplified drawing of the test bench and how the various components are connected. The intent is to place the Xgate into a system that would be somewhat similar to an actual application system that might use the Xgate.



Top Level Test Bench

Tasks:

Host

The Host module provides a wrapper for the test bench tasks which access the WISHBONE slave bus from the top level test bench code. The basic Host module is the same as used on the PIT (Programmable Interrupt Module), COP (Computer Operating Properly) and was originally copied from the "I2C Controller Core" written by Richard Herveille. Modifications include the addition of a global error counter, improved error messages and the control of the WISHBONE byte select signals for 16/8 bit bus access.

These tasks are:

wb_write – Activates the WISHBONE Slave bus to complete a write operation. The write can be ether a 8 or 16 bit operation. Inputs are d, Delay – a, Address – d, Data – s, Size.

wb_read – Cycles the WISHBONE Slave bus to complete a read operation. . The read can be ether a 8 or 16 bit operation. Inputs are d, Delay – a, Address – d, Size. The task returns a data item d, Data which is the value read from the address location.

wb_cmp – This task contains a nested call to the wb_read task and does the additional operation of comparing the value read from the address to a value input to the wb_cmp task. If the data does not match the data read from the addressed location an error signal is generated and the global error counter is incremented. A display message is also added to the simulation run log. The read can be ether a 8 or 16 bit operation. Inputs are d, Delay – a, Address – d, Data for compare reference – s, Size.

Test Bench Regs

This is a fully functional WISHBONE slave module.

Name	Address	Width	Access	Description
CHECK_ POINT	0x00	16	RW	Check Point register used by the Xgate module software to communicate test progress.
CHANNE L_ACK	0x02	16	RW	Channel ack register used by the Xgate module software to indicate the start of an Xgate software thread. Used by the test bench to clear the channel input interrupt chan_req_i

Name	Address	Width	Access	Description
CHANNEL_ERROR	0x04	16	RW	Channel error register used by the Xgate module software to indicate an error condition has been detected while executing the test code.
BRKPT_CNTL	0x06	16	RW	Break point control resister used by the test bench to enable external hardware breakpoints.
BRKPT_ADDR	0x08	16	RW	Breakpoint Address register used by the test bench to set the hardware breakpoint address.
TB_SEMAPHR	0x0A	16	RW	Test Bench Semaphore register used by both the Xgate module software or test bench program to cause a change of flow.
				Reserved
XGIF_0	0x10	16	R	Xgate Interrupt Flag Register #0 xgif[15:1]
XGIF_1	0x12	16	R	Xgate Interrupt Flag Register #1 xgif[31:16]
XGIF_2	0x14	16	R	Xgate Interrupt Flag Register #2 xgif[47:32]
XGIF_3	0x16	16	R	Xgate Interrupt Flag Register #3 xgif[63:48]
XGIF_4	0x18	16	R	Xgate Interrupt Flag Register #4 xgif[79:64]
XGIF_5	0x1A	16	R	Xgate Interrupt Flag Register #5 xgif[95:80]
XGIF_6	0x1C	16	R	Xgate Interrupt Flag Register #6 xgif[111:96]
XGIF_7	0x1E	16	R	Xgate Interrupt Flag Register #7 xgif[127:112]

x.1 Check Point Register (CHECK_POINT)

The CHECK_POINT is normally used by software running on the Xgate module to communicate test progress to the test bench. Whenever the CHECK_POINT register is written a message is printed by the test bench:

("\nSoftware Checkpoint #%h -- at vector=%d\n", check_point_reg, vector)

where “check_point_reg” is the value written to the CHECK_POINT register and “vector” is current value of the test bench clock counter.

Bit #	Access	Description
15:0	R/W	check_point_reg[15:0]

Reset Value:
CHECK_POINT: 0000h

Table 1: CHECK_POINT Register Bits

x.2 Channel Acknowledge Register (CHANNEL_ACK)

The CHANNEL_ACK is normally used by software running on the Xgate module to communicate to the test bench that a specific software thread has started executing. Whenever the CHANNEL_ACK register is written a test bench task, "clear_channel", is activated to remove the input interrupt signal.

Bit #	Access	Description
15:0	R/W	channel_ack_reg [15:0]

Reset Value:
CHANNEL_ACK: 0000h

Table 1: CHANNEL_ACK Register Bits

x.3 Check Point Register (CHANNEL_ERR)

The CHANNEL_ERR is normally used by software running on the Xgate module to communicate test progress to the test bench. Whenever the CHANNEL_ERR register is written a message is printed by the test bench:

("\n ------ !!!!! Software Checkpoint Error #%d -- at vector=%d\n -------",
channel_err_reg, vector)

where "channel_err_reg" is the value written to the CHANNEL_ERR register and "vector" is current value of the test bench clock counter. A signal is also sent to increment the global error counter.

Bit #	Access	Description
15:0	R/W	channel_err_reg [15:0]

Reset Value:
CHANNEL_ERR: 0000h

Table 1: CHANNEL_ERR Register Bits

x.4 Breakpoint Control Register (BRKPT_CNTL)

The BRKPT_CNTL register will normally used by the test bench to enable hardware breakpoints. This functionality is not currently supported by the test bench or the Xgate module. There is a watch point function supported by the test bench debug module that can be used for software analysis.

Bit #	Access	Description
15:8	R/W	TRG_ENA[7:0]. Trigger Enable Mask – The TRG_ENA bits are used to individually enable or disable watch point address triggers. '0' Triggers are blocked for these watch points. '1' Triggers are enabled for these watch points.
7:1	R/W	Reserved for future use.
1	R/W	DEBUG_ENA. Debug Enable Mask – This bit is the master enable for all debug functions. '0' All debug functions are disabled. '1' All debug functions are enabled.

Reset Value:
BRKPT_CNTL: 0000h

Table 1: BRKPT_CNTL Register Bits

x.5 Breakpoint Address Register (BRKPT_ADDR)

The BRKPT_ADDR register will normally used by the test bench to enable hardware breakpoints. This functionality is not currently supported by the test bench or the Xgate module.

Bit #	Access	Description

Bit #	Access	Description
15:0	R/W	brkpt_addr_reg [15:0] Reserved for future use.

Reset Value:
BRKPT_ADDR: 0000h

Table 1: BRKPT_ADDR Register Bits

x.6 Test Bench Semaphore Register (TB_SEMPHORE)

The TB_SEMPHORE is used by software running on the Xgate module to communicate to the test bench that a certain point in the code execution has been reached. The Xgate code may be in "infinite loop" waiting for the test bench to respond by writing a new value to the TB_SEMPHORE register to cause an exit from the loop.

Bit #	Access	Description
15:0	R/W	tb_semaphr_reg [15:0]

Reset Value:
TB_SEMPHORE: 0000h

Table 1: TB_SEMPHORE Register Bits

x.7 Channel IRQ Register (CHANNEL_XGIRQ_0)

The CHANNEL_XGIRQ_0 is used by the test bench to monitor the state of the xgif_o[15:1] output signals. By using the CHANNEL_XGIRQ_0 register to observe the xgif_o signals the resources of the Host task "wb_cmp" can be used to make the test comparison for correct state of these pins.

Bit #	Access	Description
15:1	R	xgif[15:1]
0	R	Always reads "0"

Reset Value:
CHANNEL_XGIRQ_0: 0000h

Table 1: CHANNEL_XGIRQ_0 Register Bits

x.8 Channel IRQ Register (CHANNEL_XGIRQ_1)

The CHANNEL_XGIRQ_1 is used by the test bench to monitor the state of the xgif_o[31:16] output signals. By using the CHANNEL_XGIRQ_0 register to observe the xgif_o signals the resources of the Host task "wb_cmp" can be used to make the test comparison for correct state of these pins.

Bit #	Access	Description
15:0	R	xgif[31:16]

Reset Value:
CHANNEL_XGIRQ_1: 0000h

Table 1: CHANNEL_XGIRQ_1 Register Bits

RAM

The RAM module is a generic single port RAM. The bus interface logic is actually contained in the Arbitration module. A key test bench parameter that effects RAM functionality is "RAM_WAIT_STATES" which determines how many bus wait states are required for a RAM access.

The RAM array is 64K x 8bits to simplify loading the program data that was created in a byte wise S-Record format. The RAM output is 16 bits wide and is created by reading the input address and address+1.

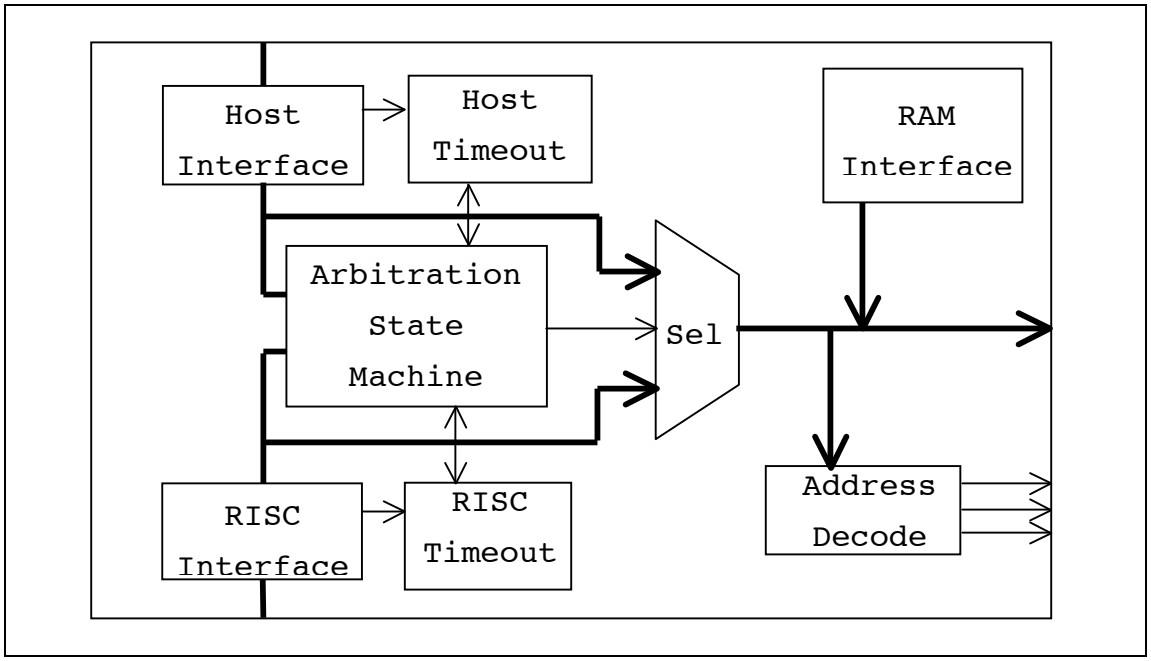
The RAM module also contains a task "dump_ram" which can be used to display RAM contents.

Xgate

The Xgate is the DUT (Device Under Test)

Arbitration

The Arbitration module brings together the hardware to interface the WISHBONE master bus signals from the Host and Xgate module to a single WISHBONE slave bus.



The main function of the arbitration module is to provide a multiplexer to route the WISHBONE master signals of the selected source to the WISHBONE slave bus. The arbitration state machine takes requests from the possible bus masters and selects which bus master is granted control for the slave bus. Because the Xgate module could possibly take total control of the slave bus while it is running a program to get continuous access to the RAM module to fetch instructions, a timeout counter is included. The timeout counter forces the current bus master to release the slave bus after a number of counts if another master is requesting slave bus access.

The module also functions as a wrapper module for the RAM interface to the WISHBONE slave bus. Finally some additional glue logic is included to decode the slave bus address to form module selects for the various slave bus peripherals. The RAM is the default access decode for any address space that is not specifically decoded for a peripheral.

DEBUGGER

To aid in software development a simple debug module is built into the test bench. The debugger loads watch point addresses stored in RAM after the first RAM initialization. The debugger generates trigger signals that can be watched in the waveform viewer and captures a copy of the CPU registers at each trigger event. The TRG_ENA[7:0] bits in the BRKPT_CNTL register controls an enable for each watch point address. The watch point addresses are generated by the software assembler program when the code is compiled

for simulation and stored in the S-Record file. Up to eight addresses can be saved in defined RAM locations for the debugger to use.

Sample Test Program

The following is a segment of ASM test code used to test the Xgate instruction set operation. (The code fragment is from the file /trunk/sw/xgate_test_code/inst_test/inst_test.s) The listed code does a simple test of the BCC (Branch Carry Clear) instruction and a test of the BRA (Branch Always) instruction.

The BCC test has two parts, the first test is structured so that the code should do the required change of flow and the second test is coded so that proper execution will not do a change of flow. If either test fails the code should branch to the error exit and the testbench will log a failed test. This is only a simple test of the BCC bit decoding of the condition code registers, NZOC, a more exhaustive test would try all 16 values of the register to fully verify the instruction functionality.

The BRA instruction is tested in a similar manner. The first test checks the instruction will always cause a change of flow when there is a forward branch (PC + OFFSET) and the second test checks for a backward change of flow (PC – OFFSET).

The sample code shows the basic setup for a test program:

- Setup jump table for each interrupt input
- Define the interrupt code
  - Send Checkpoint back to Verilog testbench for logging to follow test progress
  - Send message back to Verilog testbench to acknowledge the interrupt and cause the testbench to clear the interrupt input
  - Do the targeted instruction test.
  - Complete the interrupt code by sending a message to the Verilog testbench to log code segment completion
  - Set the Xgate output interrupt flag.
  - Define an alternate code ending for an error detect condition that will notify the Verilog testbench to log a failed test.

The ASM code accesses the Verilog testbench resources in the WISHBONE Slave module. These are:

- CHECK_POINT (Address $8000)
- CHANNEL_ACK (Address $8002)
- CHANNEL_ERROR (Address $8004)

Note: The code is pasted from another file and there is line wrap on some of the longer lines. Also most of the test code from the file has been deleted for simplicity and to highlight what has been described in the above narrative.

```
;
3456789012345678901234567890123456789012345678901234567890123456789012345678901234
; Instruction set test for xgate RISC processor core
; Bob Hayes - Sept 1 2009
;  Version 0.1 Basic test of all instruction done. Need to improve
;    Condition Code function testing.


      CPU   XGATE

      ORG   $fe00
      DS.W  2     ; reserve two words at channel 0
      ; channel 1
      DC.W  _START  ; point to start address
      DC.W  V_PTR   ; point to initial variables
      ; channel 2
      DC.W  _START2     ; point to start address
      DC.W  V_PTR   ; point to initial variables
      ; channel 3
      DC.W  _START3     ; point to start address
      DC.W  V_PTR   ; point to initial variables
      ; channel 4
      DC.W  _START4     ; point to start address
      DC.W  V_PTR   ; point to initial variables
      ; channel 5
      DC.W  _START5     ; point to start address
      DC.W  V_PTR   ; point to initial variables
      ; channel 6
      DC.W  _START6     ; point to start address
      DC.W  V_PTR   ; point to initial variables
      ; channel 7
      DC.W  _START7     ; point to start address
      DC.W  V_PTR   ; point to initial variables
      ; channel 8
      DC.W  _START8     ; point to start address
      DC.W  V_PTR   ; point to initial variables
      ; channel 9
      DC.W  _START9     ; point to start address
      DC.W  V_PTR   ; point to initial variables
      ; channel 10
      DC.W  _START10    ; point to start address
      DC.W  V_PTR       ; point to initial variables
```

```
      ; channel 11
      DC.W  _ERROR            ; point to start address
      DC.W  V_PTR       ; point to initial variables
      ; channel 12


      ORG   $2000 ; with comment

V_PTR EQU   123

        DC.W      BACK_
      DS.W  8
      DC.B  $56
      DS.B  11

      ALIGN 1

;-------------------------------------------------------------------------
;   Place where undefined interrupts go
;-------------------------------------------------------------------------
_ERROR:
      LDL   R2,#$04    ; Sent Message to Testbench Error Register
      LDH   R2,#$80
      LDL   R3,#$ff
      STB   R3,(R2,#0)

      SIF
      RTS



;-------------------------------------------------------------------------
;   Test Branch instructions
;-------------------------------------------------------------------------
_START5:
      LDL    R2,#$00    ; Sent Message to Testbench Check Point Register
      LDH    R2,#$80
      LDL    R3,#$09    ; Checkpoint Value
      STB    R3,(R2,#0)
      LDL    R3,#$05    ; Thread Value
      STB     R3,(R2,#2)  ;  Send  Message  to  clear  Testbench  interrupt
register

      ;Test BCC instruction  C = 0
      LDL   R2,#$00
      TFR   CCR,R2   ; Negative=0, Zero=0, Overflow=0, Carry=0
      BCC   _BCC_OK1 ; Take Branch
      BRA   _BR_ERR
```

```
_BCC_OK1:
      LDL   R2,#$01
      TFR   CCR,R2   ; Negative=0, Zero=0, Overflow=0, Carry=1
      BCC   _BR_ERR  ; Don't take branch



      BRA   BRA_FWARD


_BR_ERR:
      LDL   R2,#$04    ; Sent Message to Testbench Error Register
      LDH   R2,#$80
      LDL   R3,#$0a
      STB   R3,(R2,#0)

      SIF
      RTS

_BRA_OK:
      LDL   R2,#$00    ; Sent Message to Testbench Check Point Register
      LDH   R2,#$80
      LDL   R3,#$0a
      STB   R3,(R2,#0)

      SIF
      RTS

BRA_FWARD:
      BRA   _BRA_OK    ; Test backward branch calculation
```

Index

[This section contains an alphabetical list of helpful document entries with their corresponding page numbers.]

Error! No index entries found.