Copyright Information

©2003 Analog Devices, Inc., ALL RIGHTS RESERVED. This document may not be reproduced in any form without prior, express written consent from Analog Devices, Inc.

Printed in the USA.

Disclaimer

Analog Devices, Inc. reserves the right to change this product without prior notice. Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use; nor for any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under the patent rights of Analog Devices, Inc.

Trademark and Service Mark Notice

The Analog Devices logo, the SHARC logo, EZ-ICE, and SHARC are registered trademarks of Analog Devices, Inc.

VisualDSP++ is a trademark of Analog Devices, Inc.

All other brand and product names are trademarks or service marks of their respective owners.
CONTENTS

PREFACE

For Additional Information About Analog Products ....................... -xiii
For Technical or Customer Support .............................................. -xiv
What's This Book About and Who's It For? ................................. -xiv
How to Use This Manual .......................................................... -xvi
Related Documents .................................................................. -xviii
Conventions of Notation ............................................................ -xix

INSTRUCTION SET REFERENCE

Instruction Summary ................................................................. A-2
  Compute and Move/Modify Summary ...................................... A-4
  Program Flow Control Summary .......................................... A-6
  Immediate Move Summary .................................................. A-8
  Miscellaneous Instructions Summary .................................... A-9
  Reference Notation Summary ............................................... A-11
  Register Types Summary ................................................... A-15
  Memory Addressing Summary ............................................. A-18
  Opcode Notation ............................................................... A-19
  Universal Register Codes ................................................... A-24
CONTENTS

Group I Instructions (Compute & Move) .............................................. A-28
  Compute/dreg ⇔ DM/dreg ⇔ PM (Type 1) ..................................... A-30
  Compute (Type 2) ................................................................. A-32
  Compute/ureg ⇔ DM|PM, register modify (Type 3) ......................... A-33
  Compute/dreg ⇔ DM|PM, immediate modify (Type 4) ............... A-35
  Compute/ureg ⇔ ureg (Type 5) ............................................... A-37
  Immediate Shift/dreg ⇔ DM|PM (Type 6) .................................. A-39
  Compute/modify (Type 7) ...................................................... A-42
Group II Instructions (Program Flow Control) ..................................... A-44
  Direct Jump|Call (Type 8) ....................................................... A-45
  Indirect Jump|Call / Compute (Type 9) .................................... A-48
  Indirect Jump or Compute/dreg ⇔ DM (Type 10) ...................... A-52
  Return From Subroutine|Interrupt/Compute (Type 11) .............. A-55
  Do Until Counter Expired (Type 12) .................................... A-58
  Do Until (Type 13) ............................................................... A-60
Group III Instructions (Immediate Move) ........................................ A-62
  Ureg ⇔ DM|PM (direct addressing) (Type 14) .......................... A-63
  Ureg ⇔ DM|PM (indirect addressing) (Type 15) ...................... A-65
  Immediate data ⇔ DM|PM (Type 16) ...................................... A-67
  Immediate data ⇔ ureg (Type 17) ......................................... A-69
Group IV Instructions (Miscellaneous) ............................................. A-70
  System Register Bit Manipulation (Type 18) ............................ A-71
  Register Modify/bit-reverse (Type 19) .................................... A-73
CONTENTS

Push|Pop Stacks/Flush Cache (Type 20) ........................................ A-75
Nop (Type 21) ............................................................................... A-77
Idle (Type 22) ............................................................................... A-78
Idle16 (Type 23) ........................................................................... A-79
Cjump/Rframe (Type 24) .............................................................. A-81

COMPUTE OPERATION REFERENCE

Single-Function Operations .......................................................... B-2
ALU Operations .......................................................................... B-2
Rn = Rx + Ry ............................................................................... B-6
Rn = Rx – Ry ............................................................................... B-7
Rn = Rx + Ry + CI ....................................................................... B-8
Rn = Rx – Ry + CI – 1 ................................................................. B-9
Rn = (Rx + Ry)/2 ......................................................................... B-10
COMP(Rx, Ry) ............................................................................. B-11
Rn = Rx + CI ............................................................................... B-12
Rn = Rx + CI – 1 .......................................................................... B-13
Rn = Rx + 1 ............................................................................... B-14
Rn = Rx – 1 ............................................................................... B-15
Rn = –Rx ..................................................................................... B-16
Rn = ABS Rx ............................................................................... B-17
Rn = PASS Rx ............................................................................. B-18
Rn = Rx AND Ry ......................................................................... B-19
Rn = Rx OR Ry ........................................................................... B-20
Rn = Rx XOR Ry ......................................................................... B-21
CONTENTS

Rn = NOT Rx ................................................................. B-22
Rn = MIN(Rx, Ry) ........................................................... B-23
Rn = MAX(Rx, Ry) ........................................................... B-24
Rn = CLIP Rx BY Ry ..................................................... B-25
Fn = Fx + Fy ................................................................. B-26
Fn = Fx – Fy ................................................................. B-27
Fn = ABS (Fx + Fy) ....................................................... B-28
Fn = ABS (Fx – Fy) ....................................................... B-29
Fn = (Fx + Fy)/2 ........................................................... B-30
COMP(Fx, Fy) ................................................................. B-31
Fn = –Fx ...................................................................... B-32
Fn = ABS Fx ................................................................. B-33
Fn = PASS Fx ................................................................. B-34
Fn = RND Fx ................................................................. B-35
Fn = SCALB Fx BY Ry .................................................. B-36
Rn = MANT Fx ............................................................... B-37
Rn = LOGB Fx ............................................................... B-38
Rn = FIX Fx
   Rn = TRUNC Fx
Rn = FIX Fx BY Ry
   Rn = TRUNC Fx BY Ry ................................................ B-39
Fn = FLOAT Rx BY Ry
   Fn = FLOAT Rx .......................................................... B-41
Fn = RECIPS Fx ........................................................... B-42
Fn = RSQRTS Fx ........................................................... B-44
CONTENTS

Fn = Fx COPYSIGN Fy .............................................................. B-46
Fn = MIN(Fx, Fy) ................................................................. B-47
Fn = MAX(Fx, Fy) ................................................................. B-48
Fn = CLIP Fx BY Fy .............................................................. B-49
Multiplier Operations ............................................................ B-50

Rn = Rx * Ry mod2
  MRF = Rx * Ry mod2
  MRB = Rx * Ry mod2 ........................................................ B-54

Rn = MRF + Rx * Ry mod2
Rn = MRB + Rx * Ry mod2
MRF = MRF + Rx * Ry mod2
MRB = MRB + Rx * Ry mod2 ................................................. B-55

Rn = MRF – Rx * Ry mod2
Rn = MRB – Rx * Ry mod2
MRF = MRF – Rx * Ry mod2
MRB = MRB – Rx * Ry mod2 ................................................. B-56

Rn = SAT MRF mod1
Rn = SAT MRB mod1
MRF = SAT MRF mod1
MRB = SAT MRB mod1 ........................................................ B-57

Rn = RND MRF mod1
Rn = RND MRB mod1
MRF = RND MRF mod1
MRB = RND MRB mod1 ........................................................ B-58

MRF = 0
MRB = 0 .............................................................................. B-59
MR = Rn/Rn = MR ................................................................. B-60
Fn = Fx * Fy ........................................................................ B-62
Shifter Operations ................................................................. B-63

Rn = LSHIFT Rx BY Ry
Rn = LSHIFT Rx BY <data8> ................................................. B-65

Rn = Rn OR LSHIFT Rx BY Ry
Rn = Rn OR LSHIFT Rx BY <data8> ........................................ B-66

Rn = ASHIFT Rx BY Ry
Rn = ASHIFT Rx BY <data8> .............................................. B-67

Rn = Rn OR ASHIFT Rx BY Ry
Rn = Rn OR ASHIFT Rx BY <data8> .................................... B-68

Rn = ROT Rx BY Ry
Rn = ROT Rx BY <data8> .................................................. B-69

Rn = BCLR Rx BY Ry
Rn = BCLR Rx BY <data8> ............................................... B-70

Rn = BSET Rx BY Ry
Rn = BSET Rx BY <data8> ................................................. B-71

Rn = BTGL Rx BY Ry
Rn = BTGL Rx BY <data8> ................................................ B-72

BTST Rx BY Ry
BTST Rx BY <data8> ...................................................... B-73

Rn = FDEP Rx BY Ry
Rn = FDEP Rx BY <bit6>:<len6> ......................................... B-74

Rn = Rn OR FDEP Rx BY Ry
Rn = Rn OR FDEP Rx BY <bit6>:<len6> ............................. B-76

Rn = FDEP Rx BY Ry (SE)
Rn = FDEP Rx BY <bit6>:<len6> (SE) ................................. B-78

Rn = Rn OR FDEP Rx BY Ry (SE)
Rn = Rn OR FDEP Rx BY <bit6>:<len6> (SE) ....................... B-80
CONTENTS

Rn = FEXT Rx BY Ry
   Rn = FEXT Rx BY <bit6>:<len6> ......................................... B-82

Rn = FEXT Rx BY Ry (SE)
   Rn = FEXT Rx BY <bit6>:<len6> (SE) ............................... B-84

Rn = EXP Rx ........................................................................ B-86

Rn = EXP Rx (EX) .................................................................. B-87

Rn = LEFTZ Rx ................................................................... B-88

Rn = LEFTO Rx .................................................................. B-89

Rn = FPACK Fx ................................................................... B-90

Fn = FUNPACK Rx ............................................................. B-92

Multifunction Computations ..................................................... B-94

   Dual Add/Subtract (Fixed-Pt.) ............................................ B-96

   Dual Add/Subtract (Floating-Pt.) ....................................... B-98

   Parallel Multiplier and ALU (Fixed-Pt.) .............................. B-100

   Parallel Multiplier & ALU (Floating-Point) ......................... B-101

   Parallel Multiplier and Dual Add/Subtract ......................... B-104
CONTENTS

NUMERIC FORMATS

Single-Precision Floating-Point Format ........................................... C-2
Extended-Precision Floating-Point Format .................................... C-4
Short Word Floating-Point Format ............................................. C-5
Fixed-Point Formats .................................................................. C-8

JTAG TEST ACCESS PORT

Test Access Port (TAP) ................................................................. D-2
Instruction Register .................................................................. D-3
Boundary Register ..................................................................... D-6
Device Identification Register .................................................. D-28
Built-In Self-Test Instructions (BIST) .................................... D-28
Private Instructions .................................................................... D-29
References ................................................................................. D-29

CONTROL AND STATUS REGISTERS

System Registers ........................................................................ E-2
Latencies—Effect and Read ...................................................... E-4
System Register Bit Manipulation Instruction .......................... E-5
Bit Test Flag ............................................................................ E-6
ASTAT
  Arithmetic Status Register .................................................... E-8
IMASK and IRPTL
  Interrupt Mask and Latch Registers ................................. E-12
MODE1 Register .................................................................. E-16
MODE2 Register .................................................................. E-21
<table>
<thead>
<tr>
<th>CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sticky Status Register (STKY) .................................................. E-27</td>
</tr>
<tr>
<td>IOP Registers ............................................................................. E-31</td>
</tr>
<tr>
<td>IOP Registers Summary ................................................................ E-31</td>
</tr>
<tr>
<td>IOP Register Access Restrictions ............................................ E-40</td>
</tr>
<tr>
<td>IOP Register Group Access Contention ....................................... E-41</td>
</tr>
<tr>
<td>IOP Register Write Latencies .................................................. E-42</td>
</tr>
<tr>
<td>DMACx</td>
</tr>
<tr>
<td>External Port DMA Control Registers ....................................... E-54</td>
</tr>
<tr>
<td>DMASTAT</td>
</tr>
<tr>
<td>DMA Channel Status Register ................................................ E-64</td>
</tr>
<tr>
<td>IOCTL</td>
</tr>
<tr>
<td>Programmable I/O and SDRAM Control Register ............................ E-68</td>
</tr>
<tr>
<td>IOSTAT</td>
</tr>
<tr>
<td>Programmable I/O Status Register ........................................ E-75</td>
</tr>
<tr>
<td>RDIVx/TDIVx</td>
</tr>
<tr>
<td>SPORT Divisor Registers .................................................... E-78</td>
</tr>
<tr>
<td>SRCTLx</td>
</tr>
<tr>
<td>SPORT Receive Control Register ........................................... E-81</td>
</tr>
<tr>
<td>STCTLx</td>
</tr>
<tr>
<td>SPORT Transmit Control Register ........................................ E-90</td>
</tr>
<tr>
<td>SYSCON</td>
</tr>
<tr>
<td>System Configuration Register ........................................... E-99</td>
</tr>
<tr>
<td>SYSTAT</td>
</tr>
<tr>
<td>System Status Register .................................................... E-106</td>
</tr>
<tr>
<td>WAIT</td>
</tr>
<tr>
<td>External Memory Wait State Control Register ............................ E-111</td>
</tr>
</tbody>
</table>
CONTENTS

SYMBOL DEFINITIONS FILE
(def21065L.h) ................................................................. E-116

INTERRUPT VECTOR ADDRESSES

INDEX
Congratulations on your purchase of Analog Devices ADSP-21065L SHARC® DSP, the high-performance Digital Signal Processor of choice!

The ADSP-21065L is a 32-bit DSP with 544K bits of on-chip memory that is designed to support a wide variety of applications—audio, automotive, communications, industrial, and instrumentation.

For Additional Information About Analog Products

Analog Devices is online on the internet at http://www.analog.com. Our Web pages provide information on the company and products, including access to technical information and documentation, product overviews, and product announcements. You may also obtain additional information about Analog Devices and its products in any of the following ways:

- FAX questions or requests for information to 1(781)461-3010.
- Send questions by mail to:
  Analog Devices, Inc.
  One Technology Way
  P.O. Box 9106
  Norwood, MA 02062-9106
  USA
For Technical or Customer Support

- Access the division’s File Transfer Protocol (FTP) site at ftp
  This site is a mirror of the BBS.

For Technical or Customer Support

You can reach our Customer Support group in the following ways:

- Call the Analog Devices automated Customer Support Hot Line at 1(800)ANALOG-D.
- E-mail questions to dsp.support@analog.com or
dsp.europe@analog.com (European customer support).

What’s This Book About and Who’s It For?

The ADSP-21065L documentation set contains two manuals, the
ADSP-21065L SHARC DSP User’s Manual and the ADSP-21065L
SHARC DSP Technical Reference. These manuals are reference guides for
hardware and software engineers who want to develop applications using
the ADSP-21065L. These manuals assume that the user has a working
knowledge of the ADSP-21065L’s Super Harvard Architecture.

The ADSP-21065L SHARC DSP User’s Manual describes the architecture
and operation of the ADSP-21065L’s individual components, intercom-
ponent connections and access, off-chip connections and access, and the
processor’s hardware/software interface.
The information in this book includes:

- Pin definitions and instructions for connecting the pins to external devices and peripherals in single- and multiprocessor systems.
- Processor features and instructions for configuring the processor for specific operation options.
- Internal and external data paths and instructions for moving data between internal components and between the processor and external devices and peripherals.
- Timing, sequencing, and throughput of control signals and data accesses.

The *ADSP-21065L SHARC DSP Technical Reference* provides detailed technical information on programming the ADSP-21065L. This information includes:

- A description of each instruction in the processor’s instruction set, supported numeric formats, and the default bit definitions for all of the processor’s control and status registers.
- A description of the pins and the control and data registers of the JTAG test access port.
- A list of all vector interrupts and their addresses.

To supplement the information in these manuals, users can attend scheduled workshops sponsored by Analog Devices, Inc. (ADI) and access other ADI documentation related specifically to this product. For details, see “Related Documents” on page xviii.
## How to Use This Manual

<table>
<thead>
<tr>
<th>For information on...</th>
<th>See...</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU operation</td>
<td>Chapter 2, Computation Units; Appendix B, Compute Operation Reference</td>
</tr>
<tr>
<td>Address generation</td>
<td>Chapter 4, Data Addressing; Chapter 5, Memory; Chapter 6, DMA</td>
</tr>
<tr>
<td>Booting</td>
<td>Chapter 5, Memory; Chapter 7, System Design</td>
</tr>
<tr>
<td>Clock generation</td>
<td>Chapter 9, Serial Ports; Chapter 11, Programmable Timers and I/O Ports; Chapter 12, System Design</td>
</tr>
<tr>
<td>Computation units</td>
<td>Chapter 2, Computation Units; Appendix B, Compute Operation Reference; Appendix C, Numeric Formats</td>
</tr>
<tr>
<td>Data delays, latencies, throughput</td>
<td>Chapter 10, SDRAM Interface; Chapter 12, System Design</td>
</tr>
<tr>
<td>Data packing</td>
<td>Chapter 6, DMA; Chapter 8, Host Interface; Chapter 9, Serial Ports</td>
</tr>
<tr>
<td>DMA</td>
<td>Chapter 6, DMA; Chapter 7, Multiprocessing; Chapter 8, Host Interface</td>
</tr>
<tr>
<td>External port</td>
<td>Chapter 6, DMA; Chapter 7, Multiprocessing; Chapter 8, Host Interface</td>
</tr>
<tr>
<td>High-frequency design issues</td>
<td>Chapter 12, System Design</td>
</tr>
<tr>
<td>Host interface</td>
<td>Chapter 8, Host Interface</td>
</tr>
<tr>
<td>Instruction cache</td>
<td>Chapter 3, Program Sequencing; Chapter 5, Memory</td>
</tr>
<tr>
<td>For information on...</td>
<td>See...</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Instruction set</td>
<td>Appendix A, Instruction Set Reference; Appendix B, Compute Operation Reference; Appendix C, Numeric Formats</td>
</tr>
<tr>
<td>Internal buses</td>
<td>Chapter 5, Memory; Chapter 6, DMA; Chapter 8, Host Interface</td>
</tr>
<tr>
<td>Interrupts</td>
<td>Chapter 3, Program Sequencing; Chapter 5, Memory; Appendix F, Interrupt Vector Addresses</td>
</tr>
<tr>
<td>JTAG test port</td>
<td>Chapter 12, System Design; Appendix D, JTAG Test Access Port</td>
</tr>
<tr>
<td>Memory</td>
<td>Chapter 5, Memory</td>
</tr>
<tr>
<td>Multiplier operation</td>
<td>Chapter 2, Computation Units; Appendix B, Compute Operation Reference</td>
</tr>
<tr>
<td>Multiprocessing</td>
<td>Chapter 7, Multiprocessing</td>
</tr>
<tr>
<td>Pin definitions</td>
<td>Chapter 12, System Design</td>
</tr>
<tr>
<td>Processor architecture</td>
<td>Chapter 1, Introduction</td>
</tr>
<tr>
<td>Processor configuration</td>
<td>Appendix E, Control and Status Registers</td>
</tr>
<tr>
<td>Program flow</td>
<td>Chapter 3, Program Sequencing</td>
</tr>
<tr>
<td>Programmable I/O ports</td>
<td>Chapter 11, Programmable Timers and I/O Ports</td>
</tr>
<tr>
<td>Programmable timers</td>
<td>Chapter 11, Programmable Timers and I/O Ports</td>
</tr>
<tr>
<td>Programming considerations</td>
<td>Chapter 13, Programming Considerations</td>
</tr>
</tbody>
</table>
Related Documents

<table>
<thead>
<tr>
<th>For information on...</th>
<th>See...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset</td>
<td>Chapter 7, Multiprocessing; Chapter 9, Serial Ports; Chapter 12, System Design</td>
</tr>
<tr>
<td>SDRAM interface</td>
<td>Chapter 10 SDRAM Interface</td>
</tr>
<tr>
<td>Serial ports</td>
<td>Chapter 9, Serial Ports</td>
</tr>
<tr>
<td>Shifter operation</td>
<td>Chapter 2, Computation Units; Appendix B, Compute Operation Reference</td>
</tr>
<tr>
<td>System Design</td>
<td>Chapter 12, System Design</td>
</tr>
<tr>
<td>Wait states</td>
<td>Chapter 5, Memory; Chapter 12, System Design; Appendix E, Control and Status Registers</td>
</tr>
<tr>
<td>Indexes</td>
<td>Both manuals are cross-indexed. Pages with an alphabetic prefix (as C-12) reference information in ADSP-21065L SHARC DSP Technical Reference. Pages with a numeric prefix (as 5-41) reference information in ADSP-21065L SHARC DSP User’s Manual.</td>
</tr>
</tbody>
</table>

Related Documents

For information on related products, see the following documents available from Analog Devices, Inc.:

- **ADSP-21065L SHARC DSP, 198 MFLOPS, 3.3v Data Sheet (Rev. C, 6/03)**
- **VisualDSP++ Quick Installation Reference Card**
- **VisualDSP++ 3.0 User’s Guide for SHARC DSPs**
- **VisualDSP++ 3.0 Getting Started Guide for SHARC DSPs**
Conventions of Notation

The following conventions apply to all chapters within this manual. Additional conventions that apply to specific chapters only are documented at the beginning of the chapter in which they appear.

<table>
<thead>
<tr>
<th>This notation...</th>
<th>Denotes...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Letter Gothic font</strong></td>
<td>Code, software or command line options or keywords; input you must enter from the keyboard.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Special terminology; titles of books.</td>
</tr>
<tr>
<td><img src="hint.png" alt="Hint" /></td>
<td>A hint or tip.</td>
</tr>
<tr>
<td><img src="warning.png" alt="Warning" /></td>
<td>A warning or caution.</td>
</tr>
</tbody>
</table>
Appendix A and B describe the processor’s instruction set. This appendix explains each instruction type, including the assembly language syntax and opcodes, which result from instruction assembly.

Many instructions’ opcodes contain a `COMPUTE` field that specifies a compute operation using the ALU, Multiplier, or Shifter. Because a large number of options are available for computations, their descriptions appear in Appendix B.

Because data moves between the MR registers and the Register File are considered Multiplier operations, their descriptions appear in Appendix B.
Instruction Summary

Each instruction is specified in this appendix. The reference page for an instruction shows the syntax of the instruction, describes its function, gives one or two assembly-language examples, and identifies fields of its opcode. The instruction types are organized into four groups:

- “Group I Instructions (Compute & Move)” on page A-28
  These instructions specify a compute operation in parallel with one or two data moves or an index register modify.

- “Group II Instructions (Program Flow Control)” on page A-44
  These instructions specify various types of branches, calls, returns, and loops. Some may also specify a compute operation or a data move.

- “Group III Instructions (Immediate Move)” on page A-62
  These instructions use immediate instruction fields as operators for addressing.

- “Group IV Instructions (Miscellaneous)” on page A-70
  These instructions include bit modify, bit test, no operation, and idle.

The instructions are referred to by type, ranging from 1 to 23. These types correspond to the opcodes that the processor recognizes, but are for reference only and have no bearing on programming.

Some instructions have more than one syntactical form; for example, instruction “Compute/dreg ≠ DM|PM, immediate modify (Type 4)” on page A-35 has four distinct forms.
Many instructions can be conditional. These instructions are prefaced by
IF COND; for example:

If COND compute, $|DM(Ia,Mb)| = ureg$;

In a conditional instruction, the execution of the entire instruction is
based on the specified condition.
Compute and Move/Modify Summary

Compute and move/modify instructions are classed as Group I instructions, and they provide math, conditional, memory or register access services. For a complete description of these instructions, see the noted pages.

For all compute and move/modify instructions, IF COND is optional.

“Compute/dreg ↔ DM/dreg ↔ PM (Type 1)” page A-30

```
compute , DM(Ia, Mb) = dreg1 , PM(Ic, Md) = dreg2 ;
   dreg1 = DM(Ia, Mb) , dreg2 = PM(Ic, Md)
```

“Compute (Type 2)” on page A-32

```
IF COND compute ;
```

“Compute/ureg ↔ DM|PM, register modify (Type 3)” on page A-33

```
IF COND compute |
  , DM(Ia, Mb) = ureg ;
  , PM(Ic, Md)
|
  , DM(Mb, Ia) = ureg ;
  , PM(Md, Ic)
|
  , ureg = DM(Ia, Mb) ;
  , PM(Ic, Md)
|
  , ureg = DM(Mb, Ia) ;
  , PM(Md, Ic)
```
“Compute/dreg⇔DM|PM, immediate modify (Type 4)” on page A-35

\[
\begin{align*}
IF \ \text{COND compute} & \quad DM(Ia, <data6>) = dreg ; \\
& \quad PM(Ic, <data6>) \\
& \quad DM(<data6>, Ia) = dreg ; \\
& \quad PM(<data6>, Ic) \\
& \quad dreg = DM(Ia, <data6>) ; \\
& \quad PM(Ic, <data6>) ; \\
& \quad dreg = DM(<data6>, Ia) ; \\
& \quad PM(<data6>, Ic) ; \\
\end{align*}
\]

“Compute/ureg⇔ureg (Type 5)” on page A-37

\[
\begin{align*}
IF \ \text{COND compute,} & \quad ureg1 = ureg2 ; \\
\end{align*}
\]

“Immediate Shift/dreg⇔DM|PM (Type 6)” on page A-39

\[
\begin{align*}
IF \ \text{COND shiftimm} & \quad DM(Ia, Mb) = dreg ; \\
& \quad PM(Ic, Md) \\
& \quad dreg = DM(Ia, Mb) ; \\
& \quad PM(Ic, Md) ; \\
\end{align*}
\]

“Compute/modify (Type 7)” on page A-42

\[
\begin{align*}
IF \ \text{COND compute} & \quad MODIFY (Ia, Mb) ; \\
& \quad (Ic, Md) ; \\
\end{align*}
\]
Program Flow Control Summary

Program flow control instructions are classed as Group II instructions, and they provide control of program execution flow. For a complete description of these instructions, see the noted pages.

For all program flow control instructions, except type 10 instructions, IF COND is optional.

“Direct Jump|Call (Type 8)” on page A-45

| IF COND | <addr24> | (DB) ; |
| JUMP    | (PC, <reladdr24>) | (LA) |
|         |                 | (CI) |
|         |                 | (DB, LA) |
|         |                 | (DB, CI) |

“Indirect Jump|Call / Compute (Type 9)” on page A-48

| IF COND | (Md, Ic) | (DB) ; |
| JUMP    | (PC, <reladdr6>) | (LA) |
|         |              | (CI) |
|         |              | (DB, LA) |
|         |              | (DB, CI) |

| IF COND | (Md, Ic) | (DB) ; |
| CALL    | (PC, <reladdr6>) | . compute |
|         |              | ELSE compute |

“Indirect Jump or Compute/dreg⇌DM (Type 10)” on page A-52

| IF COND | (Md, Ic) | . ELSE compute. DM(Ia, Mb) = dreg ; |
| Jump    | (PC, <reladdr6>) | compute. dreg = DM(Ia, Mb) ; |
“Return From Subroutine|Interrupt/Compute (Type 11)” on page A-55

\[ IF \ \text{COND} \ \text{RTS} \quad (DB) \quad , \quad \text{compute} \quad : \]
\[ \quad (LR) \quad , \quad \text{ELSE compute} \quad : \]
\[ \quad (DB, \ LR) \quad , \quad \text{ELSE compute} \quad : \]

“Do Until Counter Expired (Type 12)” on page A-58

\[ \text{LCNTR} = \quad <\text{data16}> \quad , \quad \text{DO} \quad <\text{addr24}> \quad \text{UNTIL LCE} : \]
\[ \quad \text{ureg} \quad , \quad <\text{addr24}> \quad (\langle \text{PC, reladdr24} \rangle) \quad \text{UNTIL LCE} : \]

“Do Until (Type 13)” on page A-60

\[ \text{DO} \quad <\text{addr24}> \quad \text{UNTIL termination} : \]
\[ \quad (\text{PC, } \langle \text{reladdr24} \rangle) \quad \text{UNTIL termination} : \]
Immediate Move Summary

Immediate move instructions are classed as Group III instructions, and they provide memory and register access services. For a complete description of these instructions, see the noted pages.

“Ureg⇌DM|PM (direct addressing) (Type 14)” on page A-63

\[
\begin{align*}
\text{DM(<addr32>)} & \quad = \quad \text{ureg} ; \\
\text{PM(<addr24>)} & \\
\text{ureg} = & \quad \text{DM(<addr32>)} ; \\
& \quad \text{PM(<addr24>)}
\end{align*}
\]

“Ureg⇌DM|PM (indirect addressing) (Type 15)” on page A-65

\[
\begin{align*}
\text{DM(<data32>, Ia)} & \quad = \quad \text{ureg} ; \\
\text{PM(<data24>, Ic)} & \\
\text{ureg} = & \quad \text{DM(<data32>, Ia)} ; \\
& \quad \text{PM(<data24>, Ic)}
\end{align*}
\]

“Immediate data⇌DM|PM (Type 16)” on page A-67

\[
\begin{align*}
\text{DM(Ia, Mb)} & \quad = \quad <\text{data32}> ; \\
\text{PM(Ic, Md)} &
\end{align*}
\]

“Immediate data⇌ureg (Type 17)” on page A-69

\[
\begin{align*}
\text{ureg} = & \quad <\text{data32}> ;
\end{align*}
\]
Miscellaneous Instructions Summary

Miscellaneous instructions are classed as Group IV instructions, and they provide system register, bit manipulation, and low power services. For a complete description of these instructions, see the noted pages.

“System Register Bit Manipulation (Type 18)” on page A-71

BIT
      SET
CLR
TGL
TST
XOR

sreg <data32>

“Register Modify/bit-reverse (Type 19)” on page A-73

MODIFY
      (Ia. <data32>)
      (Ic. <data24>)

BITREV
      (Ia. <data32>)
      (Ic. <data24>)

“Push|Pop Stacks/Flush Cache (Type 20)” on page A-75

PUSH
      LOOP
PUSH
      STS
PUSH
      PCSTK
PUSH
      FLUSH CACHE
POP

“Nop (Type 21)” on page A-77

NOP

“Idle (Type 22)” on page A-78

IDLE
Instruction Summary

“Idle16 (Type 23)” on page A-79

IDLE16 :

“Cjump/Rframe (Type 24)” on page A-81

CJUMP function (DB) :

(RC, <reladdr24>)

RFRAME :
### Reference Notation Summary

The conventions for instruction syntax descriptions appear in Table A-1. This section also covers other parts of the instruction syntax and opcode information.

Table A-1. Instruction set notation

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>⇔, ⇔</td>
<td>Data transfer (read/write) direction.</td>
</tr>
<tr>
<td>UPPERCASE</td>
<td>Explicit syntax-assembler keyword (notation only: assembler is case-insensitive and lower-case is the preferred programming convention)</td>
</tr>
<tr>
<td>;</td>
<td>Semicolon (instruction terminator)</td>
</tr>
<tr>
<td>.</td>
<td>Comma (separates parallel operations in an instruction)</td>
</tr>
<tr>
<td>italics</td>
<td>Optional part of instruction</td>
</tr>
<tr>
<td>(comment)</td>
<td>Brackets enclose comments or remarks that explain code. Ignored by assembler.</td>
</tr>
<tr>
<td></td>
<td>List of options between vertical bars (choose one)</td>
</tr>
<tr>
<td>compute</td>
<td>ALU, Multiplier, Shifter or multifunction operation (see Appendix B, Compute Operation Reference)</td>
</tr>
<tr>
<td>shiftimm</td>
<td>Shifter immediate operation (see Appendix B, Compute Operation Reference)</td>
</tr>
<tr>
<td>condition</td>
<td>Status condition (see Table A-2 on page A-13)</td>
</tr>
<tr>
<td>termination</td>
<td>Loop termination condition (see Table A-2 on page A-13)</td>
</tr>
</tbody>
</table>
### Instruction Summary

Table A-1. Instruction set notation (Cont’d)

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ureg</td>
<td>Universal register</td>
</tr>
<tr>
<td>sreg</td>
<td>System register</td>
</tr>
<tr>
<td>dreg</td>
<td>Data register (Register File): R15-R0 or F15-F0</td>
</tr>
<tr>
<td>Ia</td>
<td>I7-I0 (DAG1 index register)</td>
</tr>
<tr>
<td>Mb</td>
<td>M7-M0 (DAG1 modify register)</td>
</tr>
<tr>
<td>Ic</td>
<td>I15-I8 (DAG2 index register)</td>
</tr>
<tr>
<td>Md</td>
<td>M15-M8 (DAG2 modify register)</td>
</tr>
<tr>
<td>&lt;datan&gt;</td>
<td>n-bit immediate data value</td>
</tr>
<tr>
<td>&lt;addrn&gt;</td>
<td>n-bit immediate address value</td>
</tr>
<tr>
<td>&lt;reladdrn&gt;</td>
<td>n-bit immediate PC-relative address value</td>
</tr>
<tr>
<td>(DB)</td>
<td>Delayed branch</td>
</tr>
<tr>
<td>(LA)</td>
<td>Loop abort (pop loop and PC stacks on branch)</td>
</tr>
<tr>
<td>(CI)</td>
<td>Clear interrupt</td>
</tr>
</tbody>
</table>
In a conditional instruction, execution of the entire instruction depends on the specified condition (cond or terminate). Table A-2 lists the codes that you can use in conditionals.

Table A-2. Condition and termination codes (IF & DO UNTIL)

<table>
<thead>
<tr>
<th>Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQ</td>
<td>ALU equal zero</td>
</tr>
<tr>
<td>LT</td>
<td>ALU less than zero</td>
</tr>
<tr>
<td>LE</td>
<td>ALU less than or equal zero</td>
</tr>
<tr>
<td>AC</td>
<td>ALU carry</td>
</tr>
<tr>
<td>AV</td>
<td>ALU overflow</td>
</tr>
<tr>
<td>MV</td>
<td>Multiplier overflow</td>
</tr>
<tr>
<td>MS</td>
<td>Multiplier sign</td>
</tr>
<tr>
<td>SV</td>
<td>Shifter overflow</td>
</tr>
<tr>
<td>SZ</td>
<td>Shifter zero</td>
</tr>
<tr>
<td>FLAG0_IN</td>
<td>Flag 0 input</td>
</tr>
<tr>
<td>FLAG1_IN</td>
<td>Flag 1 input</td>
</tr>
<tr>
<td>FLAG2_IN</td>
<td>Flag 2 input</td>
</tr>
<tr>
<td>FLAG3_IN</td>
<td>Flag 3 input</td>
</tr>
<tr>
<td>TF</td>
<td>Bit test flag</td>
</tr>
<tr>
<td>BM</td>
<td>Bus master</td>
</tr>
<tr>
<td>LCE</td>
<td>Loop counter expired (DO UNTIL)</td>
</tr>
</tbody>
</table>
### Instruction Summary

#### Table A-2. Condition and termination codes (IF & DO UNTIL) (Cont’d)

<table>
<thead>
<tr>
<th>Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT LCE</td>
<td>Loop counter not expired (IF)</td>
</tr>
<tr>
<td>NE</td>
<td>ALU not equal to zero</td>
</tr>
<tr>
<td>GE</td>
<td>ALU greater than or equal zero</td>
</tr>
<tr>
<td>GT</td>
<td>ALU greater than zero</td>
</tr>
<tr>
<td>NOT AC</td>
<td>Not ALU carry</td>
</tr>
<tr>
<td>NOT AV</td>
<td>Not ALU overflow</td>
</tr>
<tr>
<td>NOT MV</td>
<td>Not Multiplier overflow</td>
</tr>
<tr>
<td>NOT MS</td>
<td>Not Multiplier sign</td>
</tr>
<tr>
<td>NOT SV</td>
<td>Not Shifter overflow</td>
</tr>
<tr>
<td>NOT S2</td>
<td>Not Shifter zero</td>
</tr>
<tr>
<td>NOT FLAG0_IN</td>
<td>Not Flag 0 input</td>
</tr>
<tr>
<td>NOT FLAG1_IN</td>
<td>Not Flag 1 input</td>
</tr>
<tr>
<td>NOT FLAG2_IN</td>
<td>Not Flag 2 input</td>
</tr>
<tr>
<td>NOT FLAG3_IN</td>
<td>Not Flag 3 input</td>
</tr>
<tr>
<td>NOT TF</td>
<td>Not bit test flag</td>
</tr>
<tr>
<td>NBM</td>
<td>Not bus master</td>
</tr>
<tr>
<td>FOREVER</td>
<td>Always false (DO UNTIL)</td>
</tr>
<tr>
<td>TRUE</td>
<td>Always true (IF)</td>
</tr>
</tbody>
</table>
Register Types Summary

The processor contains three types of registers: Universal registers, Multiplier registers, and IOP registers. Table A-3 and Table A-4 list the Universal and Multiplier registers, which are associated with the processor's core. The IOP registers are associated with the processor's I/O processor and are described in Appendix E, Control and Status Registers.

Table A-3. Universal registers (UREG)

<table>
<thead>
<tr>
<th>Type</th>
<th>Subregisters</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register File</td>
<td>RO–R15</td>
<td>Register file locations, fixed-point</td>
</tr>
<tr>
<td></td>
<td>FO–F15</td>
<td>Register file locations, floating-point</td>
</tr>
<tr>
<td>Program Sequencer</td>
<td>PC</td>
<td>Program counter (read-only)</td>
</tr>
<tr>
<td></td>
<td>PCSTK</td>
<td>Top of PC stack</td>
</tr>
<tr>
<td></td>
<td>PCSTKP</td>
<td>PC stack pointer</td>
</tr>
<tr>
<td></td>
<td>FADDR</td>
<td>Fetch address (read-only)</td>
</tr>
<tr>
<td></td>
<td>DADDR</td>
<td>Decode address (read-only)</td>
</tr>
<tr>
<td></td>
<td>LADDR</td>
<td>Loop termination address, code; top of loop address stack</td>
</tr>
<tr>
<td></td>
<td>CURLCNTR</td>
<td>Current loop counter; top of loop count stack</td>
</tr>
<tr>
<td>Data Address Generators</td>
<td>IO–I7</td>
<td>DAG1 index registers</td>
</tr>
</tbody>
</table>
### Instruction Summary

#### Table A-3. Universal registers (UREG) (Cont’d)

<table>
<thead>
<tr>
<th>Type</th>
<th>Subregisters</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Address Generators (Cont’d)</td>
<td>MO-M7</td>
<td>DAG1 modify registers</td>
</tr>
<tr>
<td></td>
<td>LD-L7</td>
<td>DAG1 length registers</td>
</tr>
<tr>
<td></td>
<td>BO-B7</td>
<td>DAG1 base registers</td>
</tr>
<tr>
<td></td>
<td>I8-I15</td>
<td>DAG2 index registers</td>
</tr>
<tr>
<td></td>
<td>MB-M15</td>
<td>DAG2 modify registers</td>
</tr>
<tr>
<td></td>
<td>LB-L15</td>
<td>DAG2 length registers</td>
</tr>
<tr>
<td></td>
<td>BB-B15</td>
<td>DAG2 base registers</td>
</tr>
<tr>
<td>Bus Exchange</td>
<td>PX1</td>
<td>PMD-DMD bus exchange 1 (16 bits)</td>
</tr>
<tr>
<td></td>
<td>PX2</td>
<td>PMD-DMD bus exchange 2 (32 bits)</td>
</tr>
<tr>
<td></td>
<td>PX</td>
<td>48-bit combination of PX1 and PX2</td>
</tr>
<tr>
<td>System Registers (core)</td>
<td>MODE1</td>
<td>Mode control and status</td>
</tr>
<tr>
<td></td>
<td>MODE2</td>
<td>Mode control and status</td>
</tr>
<tr>
<td></td>
<td>IRPTL</td>
<td>Interrupt latch</td>
</tr>
<tr>
<td></td>
<td>IMASK</td>
<td>Interrupt mask</td>
</tr>
<tr>
<td></td>
<td>IMASKP</td>
<td>Interrupt mask pointer (for nesting)</td>
</tr>
<tr>
<td></td>
<td>ASTAT</td>
<td>Arithmetic status flags, bit test flag, etc.</td>
</tr>
</tbody>
</table>
Table A-3. Universal registers (UREG) (Cont’d)

<table>
<thead>
<tr>
<th>Type</th>
<th>Subregisters</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Registers</td>
<td>STKY</td>
<td>Sticky arithmetic status flags, stack status flags, etc.</td>
</tr>
<tr>
<td>(Cont’d)</td>
<td>USTAT1</td>
<td>User status register 1</td>
</tr>
<tr>
<td></td>
<td>USTAT2</td>
<td>User status register 2</td>
</tr>
</tbody>
</table>

Table A-4. Multiplier registers

<table>
<thead>
<tr>
<th>Registers</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>MR, MRO–MR2</td>
<td>Multiplier results</td>
</tr>
<tr>
<td>MRF, MROF–MR2F</td>
<td>Multiplier results, foreground</td>
</tr>
<tr>
<td>MRB, MR0B–MR2B</td>
<td>Multiplier results, background</td>
</tr>
</tbody>
</table>
Instruction Summary

Memory Addressing Summary

The processor supports the following types of addressing:

Direct Addressing

Absolute address (Instruction Types 8, 12, 13, 14)

\[
\text{dm}(0x000015F0) = \text{astat};
\]

\[
\text{if ne jump label2:}
\]

('label2' is an address label)

PC-relative address (Instruction Types 8, 9, 10, 12, 13)

\[
\text{call(pc,10), r0=r6+r3};
\]

\[
\text{do(pc,length) until sz:}
\]

('length' is a variable)

Indirect Addressing (using DAG registers):

Postmodify with M register, update I register
(Instruction Types 1, 3, 6, 16)

\[
f5=\text{pm}(i9,m12);
\]

\[
\text{dm}(i0,m3)=r3, r1=\text{pm}(i15,m10);
\]

Premodify with M register, no update
(Instruction Types 3, 9, 10)

\[
r1=\text{pm}(m10,i15);
\]

\[
\text{jump}(m13,i11);
\]

Postmodify with immediate value, update I register
(Instruction Type 4)

\[
f15=\text{dm}(i0,6);
\]

\[
\text{if av r1=pm(i15,0x11)};
\]

Premodify with immediate value, no update
(Instruction Types 4, 15)

\[
\text{if av r1=pm(0x11,i15)};
\]

\[
\text{dm}(127,i5)=\text{laddr};
\]
**Opcode Notation**

In the processor’s opcodes, some bits are explicitly defined as zeros (0s) or ones (1s). The values of other bits or fields set various parameters for the instruction. The processor ignores unspecified bits when it decodes the instruction, but reserves the bits for future use. Table A-5 lists and defines the bits, fields, and states of these opcodes.

Table A-5. Opcode acronyms

<table>
<thead>
<tr>
<th>Bit/Field</th>
<th>Description</th>
<th>States</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Loop abort code</td>
<td>0    Do not pop loop, PC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>stacks on branch</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1    Pop loop, PC stacks</td>
</tr>
<tr>
<td></td>
<td></td>
<td>on branch</td>
</tr>
<tr>
<td>ADDR</td>
<td>Immediate address field</td>
<td></td>
</tr>
<tr>
<td>AI</td>
<td>Computation unit register</td>
<td>0000 MROF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0001 MR1F</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0010 MR2F</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0100 MROB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0101 MR1B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0110 MR2B</td>
</tr>
<tr>
<td>B</td>
<td>Branch type</td>
<td>0    Jump</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1    Call</td>
</tr>
<tr>
<td>BOP</td>
<td>Bit Operation select codes</td>
<td>000 Set</td>
</tr>
<tr>
<td></td>
<td></td>
<td>001 Clear</td>
</tr>
<tr>
<td></td>
<td></td>
<td>010 Toggle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100 Test</td>
</tr>
<tr>
<td></td>
<td></td>
<td>101 XOR</td>
</tr>
</tbody>
</table>
## Opcode Notation

<table>
<thead>
<tr>
<th>Bit/Field</th>
<th>Description</th>
<th>States</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPUTE</td>
<td>Compute operation field (see Appendix B, Compute Operation Reference)</td>
<td></td>
</tr>
<tr>
<td>COND</td>
<td>Status Condition codes</td>
<td>0–31</td>
</tr>
</tbody>
</table>
| CI        | Clear interrupt code | 0  Do not clear current interrupt  
1  Clear current interrupt |
| CU        | Computation unit select codes | 00  ALU  
01  Multiplier  
10  Shifter |
| DATA      | Immediate data field | |
| DEC       | Counter decrement code | 0  No counter decrement  
1  Counter decrement |
| DMD       | Memory access direction | 0  Read  
1  Write |
| DMI       | Index (I) register numbers, DAG1 | 0–7 |
| DMM       | Modify (M) register numbers, DAG1 | 0–7 |
| DREG      | Register file locations | 0–15 |
| E         | ELSE clause code | 0  No ELSE clause  
1  ELSE clause |
## Instruction Set Reference

Table A-5. Opcode acronyms (Cont’d)

<table>
<thead>
<tr>
<th>Bit/Field</th>
<th>Description</th>
<th>States</th>
</tr>
</thead>
<tbody>
<tr>
<td>FC</td>
<td>Flush cache code</td>
<td>0  No cache flush</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1  Cache flush</td>
</tr>
<tr>
<td>G</td>
<td>DAG/Memory select</td>
<td>0  DAG1 or Data Memory</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1  DAG2 or Program Memory</td>
</tr>
<tr>
<td>INC</td>
<td>Counter increment code</td>
<td>0  No counter increment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1  Counter increment</td>
</tr>
<tr>
<td>J</td>
<td>Jump Type</td>
<td>0  nondelayed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1  Delayed</td>
</tr>
<tr>
<td>LPO</td>
<td>Loop stack pop code</td>
<td>0  No stack pop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1  Stack pop</td>
</tr>
<tr>
<td>LPU</td>
<td>Loop stack push code</td>
<td>0  No stack push</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1  Stack push</td>
</tr>
<tr>
<td>LR</td>
<td>Loop reentry code</td>
<td>0  No loop reentry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1  Loop reentry</td>
</tr>
<tr>
<td>NUM</td>
<td>Interrupt vector</td>
<td>0 - 7</td>
</tr>
<tr>
<td>OPCODE</td>
<td>Computation unit opcodes (see Appendix B, Compute Operation Reference)</td>
<td></td>
</tr>
<tr>
<td>PMD</td>
<td>Memory access direction</td>
<td>0  Read</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1  Write</td>
</tr>
<tr>
<td>PMI</td>
<td>Index (I) register numbers, DAG2</td>
<td>8-15</td>
</tr>
</tbody>
</table>
## Opcode Notation

### Table A-5. Opcode acronyms (Cont’d)

<table>
<thead>
<tr>
<th>Bit/Field</th>
<th>Description</th>
<th>States</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMM</td>
<td>Modify (M) register numbers, DAG2</td>
<td>8–15</td>
</tr>
<tr>
<td>PPO</td>
<td>PC stack pop code</td>
<td>0   No stack pop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1   Stack pop</td>
</tr>
<tr>
<td>PPU</td>
<td>PC stack push code</td>
<td>0   No stack push</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1   Stack push</td>
</tr>
<tr>
<td>RELADDR</td>
<td>PC-relative address field</td>
<td></td>
</tr>
<tr>
<td>SPO</td>
<td>Status stack pop code</td>
<td>0   No stack pop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1   Stack pop</td>
</tr>
<tr>
<td>SPU</td>
<td>Status stack push code</td>
<td>0   No stack push</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1   Stack push</td>
</tr>
<tr>
<td>SREG</td>
<td>System Register code</td>
<td>0–15 (see “Universal Register Codes” on page A-24)</td>
</tr>
<tr>
<td>TERM</td>
<td>Termination Condition codes</td>
<td>0–31</td>
</tr>
<tr>
<td>U</td>
<td>Update, index (I) register</td>
<td>0   Premodify, no update</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1   Postmodify with update</td>
</tr>
<tr>
<td>UREG</td>
<td>Universal Register code</td>
<td>0–256 (see “Universal Register Codes” on page A-24)</td>
</tr>
<tr>
<td>RA, RM, RN, RS, RX, RY</td>
<td>Register file locations for compute operands and results</td>
<td>0–15</td>
</tr>
</tbody>
</table>
### Table A-5. Opcode acronyms (Cont’d)

<table>
<thead>
<tr>
<th>Bit/Field</th>
<th>Description</th>
<th>States</th>
</tr>
</thead>
<tbody>
<tr>
<td>RXA</td>
<td>ALU x-operand Register File location for multifunction operations</td>
<td>8–11</td>
</tr>
<tr>
<td>RXM</td>
<td>Multiplier x-operand Register File location for multifunction operations</td>
<td>0–3</td>
</tr>
<tr>
<td>RYA</td>
<td>ALU y-operand Register File location for multifunction operations</td>
<td>12–15</td>
</tr>
<tr>
<td>RYM</td>
<td>Multiplier y-operand Register File location for multifunction operations</td>
<td>4–7</td>
</tr>
</tbody>
</table>
### Universal Register Codes

Table A-6, Table A-7, Table A-8, Table A-9, and Table A-10 in this section list the bit codes for registers that appear within opcode fields.

**Table A-6. Map 1 registers**

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>program counter</td>
</tr>
<tr>
<td>PCSTK</td>
<td>top of PC stack</td>
</tr>
<tr>
<td>PCSTKP</td>
<td>PC stack pointer</td>
</tr>
<tr>
<td>FADDR</td>
<td>fetch address</td>
</tr>
<tr>
<td>DADDR</td>
<td>decode address</td>
</tr>
<tr>
<td>LADDR</td>
<td>loop termination address</td>
</tr>
<tr>
<td>CURLCNTR</td>
<td>current loop counter</td>
</tr>
<tr>
<td>LCNTR</td>
<td>loop counter</td>
</tr>
<tr>
<td>R15–R0</td>
<td>Register File locations</td>
</tr>
<tr>
<td>I15–I0</td>
<td>DAG1 and DAG2 index registers</td>
</tr>
<tr>
<td>M15–M0</td>
<td>DAG1 and DAG2 modify registers</td>
</tr>
<tr>
<td>L15–L0</td>
<td>DAG1 and DAG2 length registers</td>
</tr>
<tr>
<td>B15–B0</td>
<td>DAG1 and DAG2 base registers</td>
</tr>
</tbody>
</table>
### Table A-7. Map 1 system registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODE1</td>
<td>mode control 1</td>
</tr>
<tr>
<td>MODE2</td>
<td>mode control 2</td>
</tr>
<tr>
<td>IRPTL</td>
<td>interrupt latch</td>
</tr>
<tr>
<td>IMASK</td>
<td>interrupt mask</td>
</tr>
<tr>
<td>IMASKP</td>
<td>interrupt mask pointer</td>
</tr>
<tr>
<td>ASTAT</td>
<td>arithmetic status</td>
</tr>
<tr>
<td>STKY</td>
<td>sticky status</td>
</tr>
<tr>
<td>USTAT1</td>
<td>user status reg 1</td>
</tr>
<tr>
<td>USTAT2</td>
<td>user status reg 2</td>
</tr>
</tbody>
</table>

### Table A-8. Map 2 registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PX</td>
<td>48-bit PX1 and PX2 combination</td>
</tr>
<tr>
<td>PX1</td>
<td>bus exchange 1 (16 bits)</td>
</tr>
<tr>
<td>PX2</td>
<td>bus exchange 2 (32 bits)</td>
</tr>
</tbody>
</table>
# Opcode Notation

## Table A-9. Map 1, universal register codes

<table>
<thead>
<tr>
<th>Bits</th>
<th>Bits:7654</th>
</tr>
</thead>
<tbody>
<tr>
<td>3210</td>
<td>0000 0001 0010 0011 0100 0101 0110 0111</td>
</tr>
<tr>
<td>0000</td>
<td>R0 I0 M0 L0 B0 FADDR USTAT1</td>
</tr>
<tr>
<td>0001</td>
<td>R1 I1 M1 L1 B1 DADDR USTAT2</td>
</tr>
<tr>
<td>0010</td>
<td>R2 I2 M2 L2 B2</td>
</tr>
<tr>
<td>0011</td>
<td>R3 I3 M3 L3 B3 PC</td>
</tr>
<tr>
<td>0100</td>
<td>R4 I4 M4 L4 B4 PCSTK</td>
</tr>
<tr>
<td>0101</td>
<td>R5 I5 M5 L5 B5 PCSTKP</td>
</tr>
<tr>
<td>0110</td>
<td>R6 I6 M6 L6 B6 LADDR</td>
</tr>
<tr>
<td>0111</td>
<td>R7 I7 M7 L7 B7 CURL-CNTR</td>
</tr>
<tr>
<td>1000</td>
<td>R8 I8 M8 L8 B8 LCNTR</td>
</tr>
<tr>
<td>1001</td>
<td>R9 I9 M9 L9 B9 IRPTL</td>
</tr>
<tr>
<td>1010</td>
<td>R10 I10 M10 L10 B10 MODE2</td>
</tr>
<tr>
<td>1011</td>
<td>R11 I11 M11 L11 B11 MODE1</td>
</tr>
<tr>
<td>1100</td>
<td>R12 I12 M12 L12 B12 ASTAT</td>
</tr>
<tr>
<td>1101</td>
<td>R13 I13 M13 L13 B13 IMASK</td>
</tr>
<tr>
<td>1110</td>
<td>R14 I14 M14 L14 B14 STKY</td>
</tr>
<tr>
<td>1111</td>
<td>R15 I15 M15 L15 B15 IMASKP</td>
</tr>
</tbody>
</table>
Table A-10. Map 2, universal register codes

<table>
<thead>
<tr>
<th>Bits:</th>
<th>Bits: 7654</th>
</tr>
</thead>
<tbody>
<tr>
<td>3210</td>
<td>1000 1001 1010 1011 1100 1101 1110 1111</td>
</tr>
<tr>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>..</td>
<td></td>
</tr>
<tr>
<td>1011</td>
<td>PX</td>
</tr>
<tr>
<td>1100</td>
<td>PX1</td>
</tr>
<tr>
<td>1101</td>
<td>PX2</td>
</tr>
<tr>
<td>..</td>
<td></td>
</tr>
<tr>
<td>1111</td>
<td></td>
</tr>
</tbody>
</table>
Group I Instructions (Compute & Move)

- “Compute/dreg\(\leftrightarrow\) DM/dreg\(\leftrightarrow\) PM (Type 1)” on page A-30.
  Parallel data memory and program memory transfers with Register File, optional compute operation.
- “Compute (Type 2)” on page A-32.
  Compute operation, optional condition.
- “Compute/ureg\(\leftrightarrow\) DM|PM, register modify (Type 3)” on page A-33.
  Transfer between data or program memory and universal register, optional condition, optional compute operation.
- “Compute/dreg\(\leftrightarrow\) DM|PM, immediate modify (Type 4)” on page A-35.
  PC-relative transfer between data or program memory and Register File, optional condition, optional compute operation.
- “Compute/ureg\(\leftrightarrow\) ureg (Type 5)” on page A-37.
  Transfer between two universal registers, optional condition, optional compute operation.
- “Immediate Shift/dreg\(\leftrightarrow\) DM|PM (Type 6)” on page A-39.
  Immediate shift operation, optional condition, optional transfer between data or program memory and Register File.
“Compute/modify (Type 7)” on page A-42.

Index register modify, optional condition, optional compute operation.

For all compute and move/modify instructions, IF COND is optional.
Group I Instructions (Compute & Move)

Compute/dreg $\leftrightarrow$ DM/dreg $\leftrightarrow$ PM (Type 1)

Parallel data memory and program memory transfers with Register File, option compute operation.

Syntax

\[
\text{compute}, \quad \text{DM}(I_a, M_b) = \text{dreg}_1 \quad \text{PM}(I_c, M_d) = \text{dreg}_2;
\]
\[
\text{dreg}_1 = \text{DM}(I_a, M_b) \quad \text{dreg}_2 = \text{PM}(I_c, M_d)
\]

Function

Parallel accesses to data memory and program memory from the Register File. The specified I registers address data memory and program memory. The I values are postmodified and updated by the specified M registers. Premodify offset addressing is not supported. For more information on register restrictions, see Chapter 4, Data Addressing, in ADSP-21065L SHARC DSP User’s Manual.

Examples

R7=BSET R6 BY R0, DM(I0,M3)=R5, PM(I11,M15)=R4;
R8=DM(I4,M1), PM(I12 M12)=R0;

Type 1 Opcode

<table>
<thead>
<tr>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>D</td>
<td>M</td>
<td>DMI</td>
<td>DMM</td>
<td>P</td>
<td>M</td>
<td>D</td>
<td>DM</td>
<td>DREG</td>
<td>PMI</td>
<td>PMM</td>
<td>PM</td>
<td>DREG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

COMPUTE
### Instruction Set Reference

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMD, PMD</td>
<td>Select the access types (read or write).</td>
</tr>
<tr>
<td>DM DREG,</td>
<td>Specify Register File locations.</td>
</tr>
<tr>
<td>PM DREG</td>
<td></td>
</tr>
<tr>
<td>DMI, PMI</td>
<td>Specify I registers for data and program memory.</td>
</tr>
<tr>
<td>DMM, PMM</td>
<td>Specify M registers used to update the I registers.</td>
</tr>
<tr>
<td>COMPUTE</td>
<td>Defines a compute operation to be performed in parallel with the data accesses; this is a NOP if no compute operation is specified in the instruction.</td>
</tr>
</tbody>
</table>
Compute (Type 2)

Compute operation, optional condition.

Syntax

```
IF COND compute :
```

Function

Conditional compute instruction. The instruction is executed if the specified condition tests true.

Examples

```
IF MS MRF=0;
F6=(F2+F3)/2;
```

Type 2 Opcode

```
47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23
000 00001 COND
```

**Bits Description**

<table>
<thead>
<tr>
<th>COND</th>
<th>Selects whether the operation specified in the COMPUTE field is executed. If the COND is true, the compute is executed. If no condition is specified, COND is TRUE condition, and the compute is executed.</th>
</tr>
</thead>
</table>

A-32        ADSP-21065L SHARC DSP Technical Reference
Compute/ureg ↔ DM|PM, register modify (Type 3)

Transfer operation between data or program memory and universal register, optional condition, optional compute operation.

Syntax

```plaintext
IF COND compute
    DM(Ia, Mb) = ureg ;
    PM(Ic, Md) ;

    DM(Mb, Ia) = ureg ;
    PM(Md, Ic) ;

    ureg = DM(Ia, Mb) ;
    PM(Ic, Md) ;

    ureg = DM(Mb, Ia) ;
    PM(Md, Ic) ;
```

Function

Access between data memory or program memory and a universal register. The specified I register addresses data memory or program memory. The I value is either premodified (M, I order) or postmodified (I, M order) by the specified M register. If it is postmodified, the I register is updated with the modified value. If a compute operation is specified, it is performed in parallel with the data access. If a condition is specified, it affects entire instruction. Note that the UREG may not be from the same DAG (i.e. DAG1 or DAG2) as Ia/Mb or Ic/Md. For more information on register restrictions, see Chapter 4, Data Addressing, in ADSP-21065L SHARC DSP User’s Manual.

Examples

```plaintext
R6=R3-R11, DM(IO,M1)=ASTAT;
IF NOT SV F8=CLIP F2 BY F14, PX=PM(I12,M12);
```
Group I Instructions (Compute & Move)

Type 3 Opcode

<table>
<thead>
<tr>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td>010</td>
<td>U</td>
<td>I</td>
<td>M</td>
<td>COND</td>
<td>G</td>
<td>D</td>
<td>UREG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COND</td>
<td>Specifies the test condition. If no condition is specified, COND is TRUE, and the instruction is executed.</td>
</tr>
<tr>
<td>D</td>
<td>Selects the access type (read or write).</td>
</tr>
<tr>
<td>G</td>
<td>Selects data memory or program memory.</td>
</tr>
<tr>
<td>UREG</td>
<td>Specifies the universal register.</td>
</tr>
<tr>
<td>I</td>
<td>Specifies the I register.</td>
</tr>
<tr>
<td>M</td>
<td>Specifies the M register.</td>
</tr>
<tr>
<td>U</td>
<td>Selects either premodify without update or post-modify with update.</td>
</tr>
<tr>
<td>COMPUTE</td>
<td>Defines a compute operation to be performed in parallel with the data access; this is a no-operation if no compute operation is specified in the instruction.</td>
</tr>
</tbody>
</table>
Compute/dreg ↔ DM | PM, immediate modify (Type 4)

PC-relative transfer between data or program memory and Register File, optional condition, optional compute operation.

Syntax

\[
\begin{align*}
\text{IF COND compute} & : \\
& . \text{DM}(Ia, <data6>) = \text{dreg} ; \\
& . \text{PM}(Ic, <data6>) \\
& . \text{DM}(<data6>, Ia) = \text{dreg} ; \\
& . \text{PM}(<data6>, Ic) \\
& . \text{dreg} = \text{DM}(Ia, <data6>) ; \\
& \text{PM}(Ic, <data6>) ; \\
& . \text{dreg} = \text{DM}(<data6>, Ia) ; \\
& \text{PM}(<data6>, Ic) ;
\end{align*}
\]

Function

Access between data memory or program memory and the Register File. The specified I register addresses data memory or program memory. The I value is either premodified (data order, I) or postmodified (I, data order) by the specified immediate data. If it is postmodified, the I register is updated with the modified value. If a compute operation is specified, it is performed in parallel with the data access. If a condition is specified, it affects entire instruction. For more information on register restrictions, see Chapter 4, Data Addressing, in ADSP-21065L SHARC DSP User's Manual.

Examples

\[
\begin{align*}
\text{IF FLAG0-IN F1=F5*F12, F11=PM(I10,40);} \\
\text{R12=R3 AND R1, DM(6,11)=R6;}
\end{align*}
\]
Group I Instructions (Compute & Move)

Type 4 Opcode

<table>
<thead>
<tr>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td>011</td>
<td>0</td>
<td>I</td>
<td>G</td>
<td>D</td>
<td>U</td>
<td>COND</td>
<td>DATA</td>
<td>DREG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| COMPUTE |

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COND</td>
<td>Specifies the test condition. If no condition is specified, COND is TRUE, and the instruction is executed.</td>
</tr>
<tr>
<td>D</td>
<td>Selects the access type (read or write).</td>
</tr>
<tr>
<td>G</td>
<td>Selects data memory or program memory.</td>
</tr>
<tr>
<td>DREG</td>
<td>Specifies the Register File location.</td>
</tr>
<tr>
<td>I</td>
<td>Specifies the I register.</td>
</tr>
<tr>
<td>DATA</td>
<td>Specifies a 6-bit, two's-complement modify value.</td>
</tr>
<tr>
<td>U</td>
<td>Selects either premodify without update or post-modify with update.</td>
</tr>
<tr>
<td>COMPUTE</td>
<td>Defines a compute operation to be performed in parallel with the data access; this is a no-operation if no compute operation is specified in the instruction.</td>
</tr>
</tbody>
</table>
Compute/ureg ↔ ureg (Type 5)

Transfer between two universal registers, optional condition, optional compute operation.

Syntax

IF COND compute, ureg1 = ureg2 :

Function

Transfer from one universal register to another. If a compute operation is specified, it is performed in parallel with the data access. If a condition is specified, it affects entire instruction.

Examples

IF TF MRF=R2*R6(SSFR), M4=R0;
LCNTR=L7;

Type 5 Opcode

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COND</td>
<td>Specifies the test condition. If no condition is specified, COND is TRUE, and the instruction is executed.</td>
</tr>
</tbody>
</table>
### Group I Instructions (Compute & Move)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRC UREG</td>
<td>Identifies the universal register source.</td>
</tr>
<tr>
<td>DEST UREG</td>
<td>Identifies the universal register destination.</td>
</tr>
<tr>
<td>COMPUTE</td>
<td>Defines a compute operation to be performed in parallel with the data transfer; this is a no-operation if no compute operation is specified in the instruction.</td>
</tr>
</tbody>
</table>
Immediate Shift/dreg $\leftrightarrow$ DM | PM (Type 6)

Immediate shift operation, optional condition, optional transfer between data or program memory and Register File.

Syntax

$$\text{IF COND shiftimm} \rightarrow \begin{array}{c} \text{DM(Ia, Mb)} \\ \text{PM(Ic, Md)} \end{array} = \text{dreg} ;$$

$$\begin{array}{c} \text{dreg} = \\ \text{DM(Ia, Mb)} \\ \text{PM(Ic, Md)} \end{array}$$

Function

An immediate shift operation is a Shifter operation that takes immediate data as its y-operand. The immediate data is one 8-bit value or two 6-bit values, depending on the operation. The x-operand and the result are Register File locations.

If an access to data or program memory from the Register File is specified, it is performed in parallel with the Shifter operation. The I register addresses data or program memory. The I value is postmodified by the specified M register and updated with the modified value. If a condition is specified, it affects entire instruction.

For more information on register restrictions, see Chapter 4, Data Addressing, in ADSP-21065L SHARC DSP User’s Manual.

Examples

IF GT R2=R6 LSHIFT BY 30, DM(I4,M4)=R0;
IF NOT SZ R3=FEXT R1 BY 8:4:
### Group I Instructions (Compute & Move)

#### Type 6 Opcode (with data access)

<table>
<thead>
<tr>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0</td>
<td>I</td>
<td>M</td>
<td>COND</td>
<td>G</td>
<td>D</td>
<td>DATAEX</td>
<td>DREG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0 | SHIFTOP | DATA | RN | RX |

#### Type 6 Opcode (without data access)

<table>
<thead>
<tr>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>00010</td>
<td>COND</td>
<td>DATAEX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0 | SHIFTOP | DATA | RN | RX |

#### Bits Description

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COND</td>
<td>Specifies the test condition. If no condition is specified, COND is TRUE, and the instruction is executed.</td>
</tr>
<tr>
<td>SHIFTOP</td>
<td>Specifies the Shifter operation.</td>
</tr>
<tr>
<td>DATA</td>
<td>Specifies an 8-bit immediate shift value. For Shifter operations requiring two 6-bit values (a shift value and a length value), the DATAEX field adds 4 MSBs to the DATA field, creating a 12-bit immediate value. The six LSBs are the shift value, and the six MSBs are the length value.</td>
</tr>
</tbody>
</table>
## Instruction Set Reference

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>Selects the access type (read or write) if a memory access is specified.</td>
</tr>
<tr>
<td>G</td>
<td>Selects data memory or program memory.</td>
</tr>
<tr>
<td>DREG</td>
<td>Specifies the Register File location.</td>
</tr>
<tr>
<td>I</td>
<td>Specifies the I register, which is postmodified and updated by the M register.</td>
</tr>
<tr>
<td>M</td>
<td>Identifies the M register for postmodify.</td>
</tr>
</tbody>
</table>
Group I Instructions (Compute & Move)

Compute/modify (Type 7)

Index register modify, optional condition, optional compute operation.

Syntax

\[ IF \text{ COND} \text{ compute} \text{ MODIFY} \begin{array}{l} (I_a, M_b) \ ; \\ (I_c, M_d) \ ; \end{array} \]

Function

Update of the specified I register by the specified M register. If a compute operation is specified, it is performed in parallel with the data access. If a condition is specified, it affects entire instruction. For more information on register restrictions, see Chapter 4, Data Addressing, in *ADSP-21065L SHARC DSP User's Manual*.

Examples

IF NOT FLAG2_IN R4=R6*R12(SUF), MODIFY(I10,M8);
IF NOT LCE MODIFY(I3,M1);

Type 7 Opcode

<table>
<thead>
<tr>
<th>000</th>
<th>00100</th>
<th>G</th>
<th>COND</th>
<th>I</th>
<th>M</th>
</tr>
</thead>
</table>

| 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 000 | 00100 | G | COND | I | M |

COMPUTE
## Instruction Set Reference

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COND</td>
<td>Specifies the test condition. If no condition is specified, COND is TRUE, and the instruction is executed.</td>
</tr>
<tr>
<td>G</td>
<td>Selects DAG1 or DAG2.</td>
</tr>
<tr>
<td>I</td>
<td>Specifies the I register.</td>
</tr>
<tr>
<td>M</td>
<td>Specifies the M register.</td>
</tr>
<tr>
<td>COMPUTE</td>
<td>Defines a compute operation to be performed in parallel with the data access; this is a no-operation if no compute operation is specified in the instruction.</td>
</tr>
</tbody>
</table>
Group II Instructions (Program Flow Control)

- “Direct Jump|Call (Type 8)” on page A-45.
  Direct (or PC-relative) jump/call, optional condition.
- “Indirect Jump|Call / Compute (Type 9)” on page A-48.
  Indirect (or PC-relative) jump/call, optional condition, optional compute operation.
- “Indirect Jump or Compute/dreg⇌DM (Type 10)” on page A-52.
  Indirect (or PC-relative) jump or optional compute operation with transfer between data memory and Register File.
- “Return From Subroutine|Interrupt/Compute (Type 11)” on page A-55.
  Return from subroutine or interrupt, optional condition, optional compute operation.
- “Do Until Counter Expired (Type 12)” on page A-58.
  Load loop counter, do loop until loop counter expired.
- “Do Until (Type 13)” on page A-60.
  Do until termination.

⚠️ For all program flow control instructions, except type 10 instructions, IF COND is optional.
Direct Jump | Call (Type 8)

Direct (or PC-relative) jump/call, optional condition.

Syntax

```
IF COND
JUMP         <addr24>    (DB)    ;
            (PC, <reladdr24>) (LA)
            (CI)          (DB, LA)
            (DB, CI)

IF COND
CALL        <addr24>    (DB)    ;
            (PC, <reladdr24>)
```

Function

A jump or call to the specified address or PC-relative address. The PC-relative address is a 24-bit, two's-complement value. If the delayed branch (DB) modifier is specified, the branch is delayed; otherwise, it is nondelayed. If the loop abort (LA) modifier is specified for a jump, the loop stacks and PC stack are popped when the jump is executed. Use the (LA) modifier if the jump transfers program execution outside of a loop. If there is no loop or the jump address is within the loop, do not use the (LA) modifier.

The clear interrupt (CI) modifier enables reuse of an interrupt while it is being serviced. Normally, the processor ignores and does not latch an interrupt that reoccurs while its service routine is already executing. Locate the JUMP (CI) instruction within the interrupt service routine. JUMP (CI) clears the status of the current interrupt without leaving the interrupt service routine and reduces the interrupt routine to a normal subroutine. This allows the interrupt to occur again, as a result of a different event or task in the processor system. For details on interrupts, see
Group II Instructions (Program Flow Control)


The JUMP (CI) instruction reduces an interrupt service routine to a normal subroutine by clearing the appropriate bit in the interrupt latch register (IRPTL) and interrupt mask pointer (IMASKP). The processor then allows the interrupt to occur again.

When returning from a subroutine that a JUMP (CI) instruction has reduced from an interrupt service routine, your application must use the (LR) modifier of the RTS instruction if the interrupt occurred during the last two instructions of a loop. For related information, see “Return From Subroutine|Interrupt/Compute (Type 11)” on page A-55.

Examples

IF AV JUMP(PC,0x00A4)(LA);
CALL init (DB);       [init is a program label]
JUMP (PC,2) (DB,CI);  [clear current int. for reuse]

Type 8 Opcode (with direct branch)

<table>
<thead>
<tr>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>00110</td>
<td>B</td>
<td>A</td>
<td>COND</td>
<td></td>
<td>J</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Instruction Set Reference

Type 8 Opcode (with PC-relative branch)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COND</td>
<td>Specifies the test condition. If no condition is specified, COND is TRUE, and the instruction is executed.</td>
</tr>
<tr>
<td>B</td>
<td>Selects the branch type, jump or call. For calls, A and CI are ignored.</td>
</tr>
<tr>
<td>J</td>
<td>Determines whether the branch is delayed or nondelayed.</td>
</tr>
<tr>
<td>ADDR</td>
<td>Specifies a 24-bit program memory address.</td>
</tr>
<tr>
<td>A</td>
<td>Activates loop abort.</td>
</tr>
<tr>
<td>CI</td>
<td>Activates clear interrupt.</td>
</tr>
<tr>
<td>RELADDR</td>
<td>Holds a 24-bit, twos-complement value that is added to the current PC value to generate the branch address.</td>
</tr>
</tbody>
</table>
Indirect Jump | Call / Compute (Type 9)

Indirect (or PC-relative) jump/call, optional condition, optional compute operation.

Syntax

IF COND
JUMP
(Md, Ic) (DB) . compute ;
(PC, <reladdr6>) (LA) ELSE compute
(CI)
(DB, LA)
(DB, CI)

IF COND
CALL
(Md, Ic) (DB) . compute ;
(PC, <reladdr6>) ELSE compute

Function

A jump or call to the specified PC-relative address or premodified I register value. The PC-relative address is a 6-bit, twos-complement value. If an I register is specified, it is modified by the specified M register to generate the branch address. The I register is not affected by the modify operation.

The jump or call is executed if a condition is specified and is true. If a compute operation is specified without the ELSE, it is performed in parallel with the jump or call. If a compute operation is specified with the ELSE, it is performed only if the condition specified is false. Note that a condition must be specified if an ELSE compute clause is specified.

If the delayed branch (DB) modifier is specified, the jump or call is delayed; otherwise, it is nondelayed. If the loop abort (LA) modifier is specified for a jump, the loop stacks and PC stack are popped when the jump is executed. You should use the (LA) modifier if the jump will transfer program execution outside of a loop. If there is no loop, or if the jump address is within the loop, you should not use the (LA) modifier.
The clear interrupt (CI) modifier allows the reuse of an interrupt while it is being serviced. Normally the processor ignores and does not latch an interrupt that reoccurs while its service routine is already executing. Locate the JUMP (CI) instruction within the interrupt service routine. JUMP (CI) clears the status of the current interrupt without leaving the interrupt service routine and reduces the interrupt routine to a normal subroutine. This allows the interrupt to occur again, as a result of a different event. For more information on interrupts, see Chapter 3, Program Sequencing, in *ADSP-21065L SHARC DSP User’s Manual*.

The JUMP (CI) instruction reduces an interrupt service routine to a normal subroutine by clearing the appropriate bit in the interrupt latch register (IRPTL) and interrupt mask pointer (IMASKP). The processor then permits the interrupt to occur again.

When returning from a subroutine that a JUMP (CI) instruction has reduced from an interrupt service routine, your application must use the (LR) modifier of the RTS instruction if the interrupt occurred during the last two instructions of a loop. (See “Return From Subroutine|Interrupt/Compute (Type 11)” on page A-55).

For more information on indirect branches, see Chapter 4, Data Addressing, in *ADSP-21065L SHARC DSP User’s Manual*.

**Examples**

JUMP(M8,I12), R6=R6-1;
IF EQ CALL(PC,17)(DB) , ELSE R6=R6-1;
Group II Instructions (Program Flow Control)

Type 9 Opcode (with indirect branch)

```
<table>
<thead>
<tr>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>01000</td>
<td>B</td>
<td>A</td>
<td>COND</td>
<td>PMI</td>
<td>PMM</td>
<td>J</td>
<td>E</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

COMPUTE

Type 9 Opcode (with PC-relative branch)

```
<table>
<thead>
<tr>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>01001</td>
<td>B</td>
<td>A</td>
<td>COND</td>
<td>RELADDR</td>
<td>J</td>
<td>E</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

COMPUTE

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COND</td>
<td>Specifies the test condition. If no condition is specified, COND is true, and the instruction is executed.</td>
</tr>
<tr>
<td>E</td>
<td>Specifies whether or not an ELSE clause is used.</td>
</tr>
<tr>
<td>B</td>
<td>Selects the branch type, jump or call. For calls, A and CI are ignored.</td>
</tr>
<tr>
<td>J</td>
<td>Determines whether the branch is delayed or non-delayed.</td>
</tr>
<tr>
<td>Bits</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>A</td>
<td>Activates loop abort.</td>
</tr>
<tr>
<td>CI</td>
<td>Activates clear interrupt.</td>
</tr>
<tr>
<td>COMPUTE</td>
<td>Defines a compute operation to be performed in parallel with the data access; this is a NOP if no compute operation is specified in the instruction.</td>
</tr>
<tr>
<td>RELADDR</td>
<td>Holds a 6-bit, twos-complement value that is added to the current PC value to generate the branch address.</td>
</tr>
<tr>
<td>PMI</td>
<td>Specifies the I register for indirect branches. The I register is premodified but not updated by the M register.</td>
</tr>
<tr>
<td>PMM</td>
<td>Specifies the M register for premodifies.</td>
</tr>
</tbody>
</table>
Group II Instructions (Program Flow Control)

Indirect Jump or Compute/dreg $\leftrightarrow$ DM (Type 10)

Indirect (or PC-relative) jump or optional compute operation with transfer between data memory and Register File.

Type 10 instructions require IF COND.

Syntax

\[
\text{IF COND Jump} (\text{Md, Ic}) , \text{Else compute. DM(Ia, Mb) = dreg ;}
\]

\[
(PC, \langle \text{reladdr6} \rangle \text{ compute. dreg = DM(Ia, Mb) ;})
\]

Function

Conditional jump to the specified PC-relative address or premodified I register value, or optional compute operation in parallel with a transfer between data memory and the Register File. In this instruction, the IF condition and ELSE keyword are not optional and must be used. If the specified condition is true, the jump is executed. If the specified condition is false, the compute operation and data memory transfer are performed in parallel. Only the compute operation is optional in this instruction.

The PC-relative address for the jump is a 6-bit, two's-complement value. If an I register is specified (Ic), it is modified by the specified M register (Md) to generate the branch address. The I register is not affected by the modify operation. Note that the delay branch (DB), loop abort (LA), and clear interrupt (CI) modifiers are not available for this jump instruction.

For the data memory access, the I register (Ia) provides the address. The I register value is postmodified by the specified M register and is updated with the modified value. Premodify addressing is not available for this data memory access.
For more information on indirect branches, see Chapter 4, Data Addressing, in *ADSP-21065L SHARC DSP User’s Manual*.

**Examples**

```
IF TF JUMP(M8, I8),
   ELSE R6=DM(I6, M1):

IF NE JUMP(PC, 0x20),
   ELSE F12=FLOAT R10 BY R3, R6=DM(I5, M0):
```

**Type 10 Opcode (with indirect jump)**

```
<table>
<thead>
<tr>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td>110</td>
<td>D</td>
<td>DMI</td>
<td>DMM</td>
<td>COND</td>
<td>PMI</td>
<td>PMM</td>
<td>DREG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
```

**Type 10 Opcode (with PC-relative jump)**

```
<table>
<thead>
<tr>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>D</td>
<td>DMI</td>
<td>DMM</td>
<td>COND</td>
<td>RELADDR</td>
<td>DREG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
```

**COMPUTE**
### Group II Instructions (Program Flow Control)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COND</td>
<td>Specifies the condition to test.</td>
</tr>
<tr>
<td>PMI</td>
<td>Specifies the I register for indirect branches. The I register is premodified, but not updated by the M register.</td>
</tr>
<tr>
<td>PMM</td>
<td>Specifies the M register for premodifies.</td>
</tr>
<tr>
<td>D</td>
<td>Selects the data memory access type (read or write).</td>
</tr>
<tr>
<td>DREG</td>
<td>Specifies the Register File location.</td>
</tr>
<tr>
<td>DMI</td>
<td>Specifies the I register which is postmodified and updated by the M register.</td>
</tr>
<tr>
<td>DMM</td>
<td>Identifies the M register for postmodifies.</td>
</tr>
<tr>
<td>COMPUTE</td>
<td>Defines a compute operation to be performed in parallel with the data access; this is a NOP if no compute operation is specified in the instruction.</td>
</tr>
<tr>
<td>RELADDR</td>
<td>Holds a 6-bit, two's-complement value that is added to the current PC value to generate the branch address.</td>
</tr>
</tbody>
</table>
Return From Subroutine | Interrupt/Compute (Type 11)

Indirect (or PC-relative) jump or optional compute operation with transfer between data memory and Register File.

Syntax

\[
\text{IF } \text{COND } \text{RTS} \quad (\text{DB}) \quad (\text{LR}) \quad (\text{DB}, \text{LR}) \quad \text{. compute} \quad \text{. ELSE compute} \quad ; \\
\text{IF } \text{COND } \text{RTI} \quad (\text{DB}) \quad \text{. compute} \quad \text{. ELSE compute} \\
\]

Function

A return from a subroutine (RTS) or return from an interrupt service routine (RTI). If the delayed branch (DB) modifier is specified, the return is delayed; otherwise, it is nondelayed.

A return causes the processor to branch to the address stored at the top of the PC stack. The difference between RTS and RTI is that the RTI instruction not only pops the return address off the PC stack, but also 1) pops status stack if the ASTAT and MODE1 status registers have been pushed (if the interrupt was IRQ2-0, the timer interrupt, or the VIRPT vector interrupt), and 2) clears the appropriate bit in the interrupt latch register (IRPTL) and the interrupt mask pointer (IMASKP).

The return is executed if a condition is specified and is true. If a compute operation is specified without the ELSE, it is performed in parallel with the return. If a compute operation is specified with the ELSE, it is performed only if the condition is false. Note that a condition must be specified if an ELSE compute clause is specified.

If a nondelayed call is used as one of the last three instructions of a loop, the loop reentry (LR) modifier must be used with the RTS instruction.
that returns from the subroutine. The (LR) modifier assures proper reentry into the loop. In counter-based loops, for example, the termination condition is checked by decrementing the current loop counter (CURLCNTR) during execution of the instruction two locations before the end of the loop. The RTS (LR) instruction prevents the loop counter from being decremented again (i.e. twice for the same loop iteration).

The (LR) modifier of RTS must also be used when returning from a subroutine which has been reduced from an interrupt service routine with a JUMP (CI) instruction (in case the interrupt occurred during the last two instructions of a loop). For a description of JUMP (CI), refer to “Direct Jump|Call (Type 8)” on page A-45 or “Indirect Jump|Call / Compute (Type 9)” on page A-48.

**Examples**

RTI, R6=R5 XOR R1;
IF NOT GT RTS(DB);
IF SZ RTS, ELSE R0=LSHIFT R1 BY R15;

**Type 11 Opcode** (return from subroutine)

```
| 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 000 | 01010 | COND | | J | E | L | R |
| 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

COMPUTE
```
**Type 11 Opcode** *(return from interrupt)*

<table>
<thead>
<tr>
<th>47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23</th>
<th>000 01011</th>
<th>COND</th>
<th>J</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
<td></td>
<td><strong>COMPUTE</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits** | **Description**
---|---
COND | Specifies the test condition. If no condition is specified, COND is true, and the return is executed.
J | Determines whether the return is delayed or nondelayed.
E | Specifies whether or not an ELSE clause is used.
COMPUTE | Defines the compute operation to be performed; this is a NOP if no compute operation is specified.
LR | Specifies whether or not the loop reentry modifier is specified.
Group II Instructions (Program Flow Control)

Do Until Counter Expired (Type 12)

Load loop counter, do loop until loop counter expired.

Syntax

\[
\text{LCNTR} = \begin{cases} \text{<data16>} & , \text{DO} & \text{<addr24>} & \text{UNTIL LCE} ; \\ \text{ureg} & (\text{<PC, reladdr24>}) & \end{cases}
\]

Function

Sets up a counter-based program loop. The loop counter LCNTR is loaded with 16-bit immediate data or from a universal register. The loop start address is pushed on the PC stack. The loop end address and the LCE termination condition are pushed on the loop address stack. The end address can be either a label for an absolute 24-bit program memory address, or a PC-relative 24-bit twos-complement address. The LCNTR is pushed on the loop counter stack and becomes the CURLCNTR value. The loop executes until the CURLCNTR reaches zero.

Examples

\[
\begin{align*}
\text{LCNTR=} & \text{100, DO } \text{fmax UNTIL LCE;} \quad \text{fmax is a program label} \\
\text{LCNTR=} & \text{R12, DO (PC,16) UNTIL LCE;}
\end{align*}
\]

Type 12 Opcode (with immediate loop counter load)

<table>
<thead>
<tr>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>01100</td>
<td>DATA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RELADDR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Type 12 Opcode (with loop counter load from a UREG)

| 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 000 | 01101 | UREG |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RELADDR |

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RELADDR</td>
<td>Specifies the end-of-loop address relative to the DO LOOP instruction address. The Assembler also accepts an absolute address and converts the absolute address to the equivalent relative address for coding.</td>
</tr>
<tr>
<td>DATA</td>
<td>Specifies a 16-bit value to load into the loop counter (LCNTR) for an immediate load.</td>
</tr>
<tr>
<td>UREG</td>
<td>Specifies a register containing a 16-bit value to load into the loop counter (LCNTR) for a load from an universal register.</td>
</tr>
</tbody>
</table>
Group II Instructions (Program Flow Control)

Do Until (Type 13)

Do until termination.

Syntax

```
DO <addr24> UNTIL termination ;
    (PC, <reladdr24>)
```

Function

Sets up a condition-based program loop. The loop start address is pushed on the PC stack. The loop end address and the termination condition are pushed on the loop stack. The end address can be either a label for an absolute 24-bit program memory address or a PC-relative, 24-bit two's-complement address. The loop executes until the termination condition tests true.

Examples

```
DO end UNTIL FLAG1_IN; {end is a program label}
DO (PC,7) UNTIL AC;
```

Type 13 Opcode (relative addressing)

<table>
<thead>
<tr>
<th>000</th>
<th>01110</th>
<th>TERM</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RELADDR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A-60 ADSP-21065L SHARC DSP Technical Reference
### Instruction Set Reference

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RELADDR</td>
<td>Specifies the end-of-loop address relative to the DO LOOP instruction address. The Assembler accepts an absolute address as well and converts the absolute address to the equivalent relative address for coding.</td>
</tr>
<tr>
<td>TERM</td>
<td>Specifies the termination condition.</td>
</tr>
</tbody>
</table>
Group III Instructions (Immediate Move)

- “Ureg → DM|PM (direct addressing) (Type 14)” on page A-63.
  Transfer between data or program memory and universal register, direct addressing, immediate address.
- “Ureg → DM|PM (indirect addressing) (Type 15)” on page A-65.
  Transfer between data or program memory and universal register, indirect addressing, immediate modifier.
- “Immediate data → DM|PM (Type 16)” on page A-67.
  Immediate data write to data or program memory.
- “Immediate data → ureg (Type 17)” on page A-69.
  Immediate data write to universal register.
Instruction Set Reference

Ureg <-> DM | PM (direct addressing) (Type 14)

Transfer between data or program memory and universal register, direct addressing, immediate address.

Syntax

\[
\begin{align*}
DM(<addr32>) & \quad = \quad \text{ureg} ; \\
PM(<addr24>) & \quad = \quad \text{ureg} ; \\
\text{ureg} & \quad = \quad DM(<addr32>) ; \\
& \quad = \quad PM(<addr24>) ;
\end{align*}
\]

Function

Access between data memory or program memory and a universal register, with direct addressing. The entire data memory or program memory address is specified in the instruction. Data memory addresses are 32 bits wide (0 to \(2^{32} - 1\)). Program memory addresses are 24 bits wide (0 to \(2^{24} - 1\)).

Examples

\[
\begin{align*}
\text{DM(temp)} &= \text{MODE1;} \quad \{\text{temp is a program label}\} \\
\text{DMWAIT} & = \text{PM(0x489060)};
\end{align*}
\]

Type 14 Opcode

<table>
<thead>
<tr>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>100</td>
<td>G</td>
<td>D</td>
<td>UREG</td>
<td>ADDR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(upper 8-bits)

<table>
<thead>
<tr>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDR</td>
<td>(lower 24-bits)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Group III Instructions (Immediate Move)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>Selects the access type (read or write).</td>
</tr>
<tr>
<td>G</td>
<td>Selects the memory type (data or program).</td>
</tr>
<tr>
<td>UREG</td>
<td>Specifies the number of a universal register.</td>
</tr>
<tr>
<td>ADDR</td>
<td>Contains the immediate address value.</td>
</tr>
</tbody>
</table>
**Ureg ↔ DM | PM (indirect addressing) (Type 15)**

Transfer between data or program memory and universal register, indirect addressing, immediate modifier.

**Syntax**

<table>
<thead>
<tr>
<th>DM(&lt;data32&gt;, Ia)</th>
<th>= ureg ;</th>
</tr>
</thead>
<tbody>
<tr>
<td>PM(&lt;data24&gt;, Ic)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ureg =</th>
</tr>
</thead>
<tbody>
<tr>
<td>DM(&lt;data32&gt;, Ia) ;</td>
</tr>
<tr>
<td>PM(&lt;data24&gt;, Ic) ;</td>
</tr>
</tbody>
</table>

**Function**

Access between data memory or program memory and a universal register, with indirect addressing using I registers. The I register is premodified with an immediate value specified in the instruction. The I register is not updated. Data memory address modifiers are 32 bits wide (0 to $2^{32}-1$). Program memory address modifiers are 24 bits wide (0 to $2^{24}-1$). The ureg may not be from the same DAG (that is, DAG1 or DAG2) as Ia/Mb or Ic/Md. For more information on register restrictions, see Chapter 4, Data Addressing, in *ADSP-21065L SHARC DSP User’s Manual.*

**Examples**

```
DM(24,I5)=TCOUNT;
USTAT1=PM(offs,I13);  /*offs* is a defined constant*/
```
Group III Instructions (Immediate Move)

Type 15 Opcode

| 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 101 | G | I | D | UREG | DATA |

(upper 8-bits)

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| DATA |

(lower 24-bits)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>Selects the access type (read or write).</td>
</tr>
<tr>
<td>G</td>
<td>Selects the memory type (data or program).</td>
</tr>
<tr>
<td>UREG</td>
<td>Specifies the number of a universal register.</td>
</tr>
<tr>
<td>DATA</td>
<td>Specifies the immediate modify value for the I register.</td>
</tr>
</tbody>
</table>
**Immediate data⇒ DM | PM (Type 16)**

Immediate data write to data or program memory.

**Syntax**

\[
\begin{align*}
\text{DM}(Ia, Mb) & = \text{<data32>} : \\
\text{PM}(Ic, Md) &
\end{align*}
\]

**Function**

A write of 32-bit immediate data to data or program memory, with indirect addressing. The data is placed in the most significant 32 bits of the 40-bit memory word. The least significant 8 bits are loaded with 0s. The I register is postmodified and updated by the specified M register. The ureg may not be from the same DAG (that is, DAG1 or DAG2) as Ia/Mb or Ic/Md. For more information on register restrictions, see Chapter 4, Data Addressing, in *ADSP-21065L SHARC DSP User’s Manual*.

**Examples**

\[
\begin{align*}
\text{DM}(I4,M0) &= 19304; \\
\text{PM}(I14,M11) &= \text{count}; \quad \text{(count is user-defined constant)}
\end{align*}
\]

**Type 16 Opcode**

<table>
<thead>
<tr>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1</td>
<td>I</td>
<td>M</td>
<td>G</td>
<td>DATA (upper 8-bits)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA (lower 24-bits)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Group III Instructions (Immediate Move)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>Selects the I register.</td>
</tr>
<tr>
<td>M</td>
<td>Selects the M register.</td>
</tr>
<tr>
<td>G</td>
<td>Selects the memory (data or program).</td>
</tr>
<tr>
<td>DATA</td>
<td>Specifies the 32-bit immediate data.</td>
</tr>
</tbody>
</table>
Immediate data\(\rightarrow\) ureg (Type 17)

Immediate data write to universal register.

Syntax

\[
\text{ureg} = \langle\text{data32}\rangle ;
\]

Function

A write of 32-bit immediate data to a universal register. If the register is 40 bits wide, the data is placed in the most significant 32 bits, and the least significant 8 bits are loaded with 0s.

Examples

IMASK=0xFFFC0060:
M15=mod1: {mod1 is user-defined constant}

Type 17 Opcode

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>47</td>
<td>UREG</td>
</tr>
<tr>
<td>46</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>DATA (upper 8-bits)</td>
</tr>
<tr>
<td>43</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>000</td>
</tr>
<tr>
<td>31</td>
<td>01111</td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Bits Description

- **UREG**: Specifies the number of a universal register.
- **DATA**: Specifies the immediate modify value for the I register.
Group IV Instructions (Miscellaneous)

Group IV Instructions (Miscellaneous)

- “System Register Bit Manipulation (Type 18)” on page A-71.
  System register bit manipulation.
- “Register Modify/bit-reverse (Type 19)” on page A-73.
  Immediate I register modify, with or without bit-reverse.
- “Push|Pop Stacks/Flush Cache (Type 20)” on page A-75.
  Push or Pop of loop and/or status stacks.
- “Nop (Type 21)” on page A-77.
  No Operation (NOP).
- “Idle (Type 22)” on page A-78.
  Idle.
- “Idle16 (Type 23)” on page A-79.
  Idle16.
- “Cjump/Rframe (Type 24)” on page A-81.
  CJUMP/RFRAME (Compiler-generated instruction).
System Register Bit Manipulation (Type 18)

System register bit manipulation.

Syntax

<table>
<thead>
<tr>
<th>BIT</th>
<th>SET</th>
<th>CLR</th>
<th>TGL</th>
<th>TST</th>
<th>XOR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>sreg</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Function

A bit manipulation operation on a system register. This instruction can set, clear, toggle or test specified bits, or compare (XOR) the system register with a specified data value. In the first four operations, the immediate data value is a mask. The set operation sets all the bits in the specified system register that are also set in the specified data value. The clear operation clears all the bits that are set in the data value. The toggle operation toggles all the bits that are set in the data value. The test operation sets the bit test flag (BTF in ASTAT) if all the bits that are set in the data value are also set in the system register. The XOR operation sets the bit test flag (BTF in ASTAT) if the system register value is the same as the data value. For more information on Shifter operations, see Appendix B, Compute Operation Reference. For more information on system registers, see Appendix E, Control and Status Registers.

Examples

BIT SET MODE2 0x00000070;
BIT TST ASTAT 0x00002000;
Group IV Instructions (Miscellaneous)

**Type 18 Opcode**

<table>
<thead>
<tr>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>10100</td>
<td>BOP</td>
<td>SREG</td>
<td>DATA (upper 8-bits)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| DATA (lower 24-bits) |

**Bits** | **Description**
--- | ---
BOP | Selects one of the five bit operations.
SREG | Specifies the system register.
DATA | Specifies the data value.
Register Modify/bit-reverse (Type 19)

Immediate I register modify, with or without bit-reverse.

Syntax

| MODIFY         | (Ia, <data32>) | ;  |
|               | (Ic, <data24>) |   |
| BITREV        | (Ia, <data32>) | ;  |
|               | (Ic, <data24>) |   |

Function

Modifies and updates the specified I register by an immediate 32-bit (DAG1) or 24-bit (DAG2) data value. If the address is to be bit-reversed, you must specify a DAG1 register (I0-I7) or DAG2 register (I8-I15), and the modified value is bit-reversed before being written back to the I register. No address is output in either case. For more information on register restrictions, see Chapter 4, Data Addressing, in ADSP-21065L SHARC DSP User’s Manual.

Examples

MODIFY (I4,304);
BITREV (I7,space);   {space is a defined constant}

Type 19 Opcode (without bit-reverse)

| 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 000 | 10110 | G | I | DATA |

(upper 8-bits)

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |

DATA

(lower 24-bits)
## Group IV Instructions (Miscellaneous)

### Type 19 Opcode (with bit-reverse)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>47</td>
<td>46</td>
</tr>
<tr>
<td>000</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA (lower 24-bits)</td>
<td></td>
</tr>
</tbody>
</table>

### Bits Description

- **G**: Selects the data address generator:
  - G=0 for DAG1
  - G=1 for DAG2

- **I**: Selects the I register:
  - I=0-7 for I0-I7 (for DAG1)
  - I=0-7 for I8-I15 (for DAG2)

- **DATA**: Specifies the immediate modifier.
Push | Pop Stacks/Flush Cache (Type 20)

Push or Pop of loop and/or status stacks.

Syntax

```
PUSH | LOOP, | PUSH | STS, | PUSH | PCSTK, | FLUSH CACHE:
PUSH | POP, | POP | STS, | POP | PCSTK, | POP |
```

Function

Pushes or pops the loop address and loop counter stacks, the status stack, and/or the PC stack, and/or clear the instruction cache. Any of these options may be combined in a single instruction.

Flushing the instruction cache invalidates all entries in the cache, with no latency—the cache is cleared at the end of the cycle.

Examples

```
PUSH LOOP, PUSH STS;
PUSH PCSTK, FLUSH CACHE;
```

Type 20 Opcode

```
| 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 000 | 10111 | L | P | U | L | P | U | S | P | O | S | P | O | P | P | P | F | C |
```

```
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
```
### Group IV Instructions (Miscellaneous)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPU</td>
<td>Pushes the loop stacks.</td>
</tr>
<tr>
<td>LPO</td>
<td>Pops the loop stacks.</td>
</tr>
<tr>
<td>SPU</td>
<td>Pushes the status stack.</td>
</tr>
<tr>
<td>SPO</td>
<td>Pops the status stack.</td>
</tr>
<tr>
<td>PPU</td>
<td>Pushes the PC stack.</td>
</tr>
<tr>
<td>PPO</td>
<td>Pops the PC stack.</td>
</tr>
<tr>
<td>FC</td>
<td>Causes a cache flush.</td>
</tr>
</tbody>
</table>
Nop (Type 21)

No Operation (NOP).

Syntax

NOP;

Function

A null operation; only increments the fetch address.

Type 21 Opcode

```
  47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24
   000 00000 0

  23 22 21 20 19 18 17 16 15 14 13 12 11 10 9  8  7  6  5  4  3  2  1  0
   0   0   0   0   0   0   0
```
Group IV Instructions (Miscellaneous)

Idle (Type 22)

Idle.

Syntax

IDLE ;

Function

Executes a NOP and puts the processor in a low power state. The processor remains in the low power state until an interrupt occurs. On return from the interrupt, execution continues at the instruction following the IDLE instruction.

Type 22 Opcode

| 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 000 | 00000 | 1 |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
**Idle16 (Type 23)**

Idle16.

**Syntax**

```
IDLE16 ;
```

**Function**

The processor does not support this instruction during DMA transfers, host accesses, or multiprocessing.

This instruction executes a NOP and puts the processor in a low power state until an external interrupt (IRQ2-0), a DMA interrupt, or a VIRPT vector interrupt occurs.

IDLE16 is a lower power version of the IDLE instruction. Like the IDLE instruction, IDLE16 halts the processor, but the internal clock continues to run at 1/16th the rate of CLKin. All internal memory transfers require an extra fifteen cycles. The serial clocks and frame syncs (if the processor is source) are divided down by a factor of sixteen during IDLE16.

The processor remains in the low power state until an interrupt occurs.

To exit IDLE16, your application software can:

- Assert the external IRQx pin.
- Generate a timer interrupt.

After returning from the interrupt, execution continues at the instruction following the IDLE16 instruction.
Group IV Instructions (Miscellaneous)

During IDLE16, the processor does not support:

- Host accesses
  Make sure your application software does not assert HBR.
- Multiprocessor bus arbitration (synchronous accesses)
- External port DMA
- SDRAM accesses
- Serial port transfers

Type 23 Opcode

<table>
<thead>
<tr>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>00000</td>
<td>1</td>
<td>01</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Cjump/Rframe (Type 24)

CJUMP/RFRAME (Compiler-generated instruction).

Syntax

\[
\begin{align*}
\text{CJUMP} & \quad \text{function} \quad (DB) : \\
& \quad (PC, <\text{reladdr24}>)
\end{align*}
\]

\[
\text{RFRAME} :
\]

Function

The CJUMP instruction is generated by the C compiler for function calls, and is not intended for use in assembly language programs. CJUMP combines a direct or PC-relative jump with register transfer operations that save the frame and stack pointers. The RFRAME instruction reverses the register transfers to restore the frame and stack pointers.

The symbol “function” is a 24-bit immediate address for direct jumps. The PC-relative address is a 24-bit, twos-complement value. The (DB) modifier causes the jump to be delayed.

The different forms of this instruction perform various operations.

<table>
<thead>
<tr>
<th>Compiler-Generated Instruction</th>
<th>Operations Performed</th>
</tr>
</thead>
<tbody>
<tr>
<td>CJUMP function (DB);</td>
<td>JUMP function (DB), R2=I6, I6=I7;</td>
</tr>
<tr>
<td>CJUMP (PC,&lt;reladdr24&gt;) (DB);</td>
<td>JUMP (PC, function) (DB), R2=I6, I6=I7;</td>
</tr>
<tr>
<td>RFRAME;</td>
<td>I7=I6, I6=DM(0,16);</td>
</tr>
</tbody>
</table>
### Group IV Instructions (Miscellaneous)

#### Type 24 Opcode (with direct branch)

<table>
<thead>
<tr>
<th></th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0001</td>
<td>1000</td>
<td>0000</td>
<td>0100</td>
<td>0000</td>
<td>0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Type 24 Opcode (with PC-relative branch)

<table>
<thead>
<tr>
<th></th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0001</td>
<td>1000</td>
<td>0100</td>
<td>0100</td>
<td>0000</td>
<td>0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Type 24 Opcode (RFRAME)

<table>
<thead>
<tr>
<th></th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0001</td>
<td>1001</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A-82 ADSP-21065L SHARC DSP Technical Reference
### Instruction Set Reference

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDR</td>
<td>Specifies a 24-bit program memory address for “function.”</td>
</tr>
<tr>
<td>RELADDR</td>
<td>Specifies a 24-bit, twos-complement value that is added to the current PC value to generate the branch address.</td>
</tr>
</tbody>
</table>
Group IV Instructions (Miscellaneous)
Compute operations execute in the Multiplier, the ALU and the Shifter. The 23-bit compute field is like a mini-instruction within the instruction and can be specified for a variety of compute operations. This appendix describes each compute operation in detail, including its assembly language syntax and opcode field.

A compute operation is one of the following:

- Single-function operations involve a single computation unit.
- Multifunction operations specify parallel operation of the Multiplier and the ALU or two operations in the ALU.
- The MR register transfer is a special type of compute operation used to access the fixed-point accumulator in the Multiplier. For more information, see “MR = Rn/Rn = MR” on page B-60.

The operations in each category are described in the following sections. For each operation, the assembly language syntax, the function, and the opcode format and contents are specified. For more information, see Table A-1 on page A-11.
**Single-Function Operations**

The compute field of a single-function operation looks like:

<table>
<thead>
<tr>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CU</td>
<td>OPCODE</td>
<td>RN</td>
<td>RX</td>
<td>RY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

An operation determined by `OPCODE` is executed in the computation unit specified by `CU`. The $x$- and the $y$-operands are received from data registers $RX$ and $RY$. The result operand is returned to data register $RN$.

The `CU` (computation unit) field is defined as follows:

- **CU=00** ALU operations
- **CU=01** Multiplier operations
- **CU=10** Shifter operations

In some Shifter operations, data register $RN$ is used both as a destination for a result operand and as source for a third input operand.

The available operations and their 8-bit `OPCODE` values are listed in the following sections, organized by computation unit: ALU, Multiplier and Shifter. In each section, the syntax and opcodes for the operations are first summarized and then the operations are described in detail.

**ALU Operations**

This section describes the ALU operations. Table B-1 and Table B-2 summarize the syntax and opcodes for the fixed-point and floating-point ALU operations, respectively.

In these tables, the individual registers of the Register File are prefixed with an “$F$” when used in floating-point computations. They are prefixed
with an "R" when used in fixed-point computations. The following instructions, for example, use the same registers:

\[
\begin{align*}
F0 &= F1 \times F2; \text{ floating-point multiply} \\
R0 &= R1 \times R2; \text{ fixed-point multiply}
\end{align*}
\]

The \( F \) and \( R \) prefixes do not affect the 32-bit (or 40-bit) data transfer. They determine only how the ALU, Multiplier, or Shifter treat the data. Since the assembler is case-insensitive, the \( F \) and \( R \) prefixes can be upper- or lowercase.

Table B-1. Fixed-point ALU operations

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rn = Rx + Ry</td>
<td>0000 0001</td>
</tr>
<tr>
<td>Rn = Rx - Ry</td>
<td>0000 0010</td>
</tr>
<tr>
<td>Rn = Rx + Ry + CI</td>
<td>0000 0101</td>
</tr>
<tr>
<td>Rn = Rx - Ry + CI - 1</td>
<td>0000 0110</td>
</tr>
<tr>
<td>Rn = (Rx + Ry)/2</td>
<td>0000 1001</td>
</tr>
<tr>
<td>COMP(Rx, Ry)</td>
<td>0000 1010</td>
</tr>
<tr>
<td>Rn = Rx + CI</td>
<td>0010 0101</td>
</tr>
<tr>
<td>Rn = Rx + CI - 1</td>
<td>0010 0110</td>
</tr>
<tr>
<td>Rn = Rx + 1</td>
<td>0010 1001</td>
</tr>
<tr>
<td>Rn = Rx - 1</td>
<td>0010 1010</td>
</tr>
<tr>
<td>Rn = −Rx</td>
<td>0010 0010</td>
</tr>
<tr>
<td>Rn = ABS Rx</td>
<td>0011 0000</td>
</tr>
<tr>
<td>Rn = PASS Rx</td>
<td>0010 0001</td>
</tr>
</tbody>
</table>
Single-Function Operations

Table B-1. Fixed-point ALU operations (Cont'd)

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R_n = R_x \text{ AND } R_y )</td>
<td>0100 0000</td>
</tr>
<tr>
<td>( R_n = R_x \text{ OR } R_y )</td>
<td>0100 0001</td>
</tr>
<tr>
<td>( R_n = R_x \text{ XOR } R_y )</td>
<td>0100 0010</td>
</tr>
<tr>
<td>( R_n = \text{NOT } R_x )</td>
<td>0100 0011</td>
</tr>
<tr>
<td>( R_n = \text{MIN}(R_x, R_y) )</td>
<td>0110 0001</td>
</tr>
<tr>
<td>( R_n = \text{MAX}(R_x, R_y) )</td>
<td>0110 0010</td>
</tr>
<tr>
<td>( R_n = \text{CLIP } R_x \text{ BY } R_y )</td>
<td>0110 0011</td>
</tr>
</tbody>
</table>

Table B-2. Floating-point ALU operations

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>( F_n = F_x + F_y )</td>
<td>1000 0001</td>
</tr>
<tr>
<td>( F_n = F_x - F_y )</td>
<td>1000 0010</td>
</tr>
<tr>
<td>( F_n = \text{ABS } (F_x + F_y) )</td>
<td>1001 0001</td>
</tr>
<tr>
<td>( F_n = \text{ABS } (F_x - F_y) )</td>
<td>1001 0010</td>
</tr>
<tr>
<td>( F_n = (F_x + F_y)/2 )</td>
<td>1000 1001</td>
</tr>
<tr>
<td>( F_n = \text{COMP}(F_x, F_y) )</td>
<td>1000 1010</td>
</tr>
<tr>
<td>( F_n = -F_x )</td>
<td>1010 0010</td>
</tr>
<tr>
<td>( F_n = \text{ABS } F_x )</td>
<td>1011 0000</td>
</tr>
<tr>
<td>( F_n = \text{PASS } F_x )</td>
<td>1010 0001</td>
</tr>
</tbody>
</table>
Table B-2. Floating-point ALU operations (Cont’d)

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_n = \text{RND } F_x$</td>
<td>1010 0101</td>
</tr>
<tr>
<td>$F_n = \text{SCALB } F_x \text{ BY } R_y$</td>
<td>1011 1101</td>
</tr>
<tr>
<td>$R_n = \text{MANT } F_x$</td>
<td>1010 1101</td>
</tr>
<tr>
<td>$R_n = \text{LOGB } F_x$</td>
<td>1100 0001</td>
</tr>
<tr>
<td>$R_n = \text{FIX } F_x \text{ BY } R_y$</td>
<td>1101 1001</td>
</tr>
<tr>
<td>$R_n = \text{FIX } F_x$</td>
<td>1100 1001</td>
</tr>
<tr>
<td>$R_n = \text{TRUNC } F_x \text{ BY } R_y$</td>
<td>1101 1101</td>
</tr>
<tr>
<td>$R_n = \text{TRUNC } F_x$</td>
<td>1100 1101</td>
</tr>
<tr>
<td>$F_n = \text{FLOAT } R_x \text{ BY } R_y$</td>
<td>1101 1010</td>
</tr>
<tr>
<td>$F_n = \text{FLOAT } R_x$</td>
<td>1100 1010</td>
</tr>
<tr>
<td>$F_n = \text{RECIPS } F_x$</td>
<td>1100 0100</td>
</tr>
<tr>
<td>$F_n = \text{RSQRTS } F_x$</td>
<td>1100 0101</td>
</tr>
<tr>
<td>$F_n = F_x \text{ COPYSIGN } F_y$</td>
<td>1110 0000</td>
</tr>
<tr>
<td>$F_n = \text{MIN}(F_x, F_y)$</td>
<td>1110 0001</td>
</tr>
<tr>
<td>$F_n = \text{MAX}(F_x, F_y)$</td>
<td>1110 0010</td>
</tr>
<tr>
<td>$F_n = \text{CLIP } F_x \text{ BY } F_y$</td>
<td>1110 0011</td>
</tr>
</tbody>
</table>
Single-Function Operations

\[ R_n = R_x + R_y \]

Function

Adds the fixed-point fields in registers \( R_x \) and \( R_y \). The result is placed in the fixed-point field in register \( R_n \). The floating-point extension field in \( R_n \) is set to all 0s. In saturation mode (the ALU saturation mode bit in MODE1 set) positive overflows return the maximum positive number (0x7FFF FFFF), and negative overflows return the minimum negative number (0x8000 0000).

Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the fixed-point output is all 0s, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if the most significant output bit is 1, otherwise cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Set if the XOR of the carries of the two most significant adder stages is 1, otherwise cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Set if the carry from the most significant adder stage is 1, otherwise cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
**Compute Operation Reference**

**Rn = Rx – Ry**

**Function**

Subtracts the fixed-point field in register `Ry` from the fixed-point field in register `Rx`. The result is placed in the fixed-point field in register `Rn`. The floating-point extension field in `Rn` is set to all 0s. In saturation mode (the ALU saturation mode bit in MODE1 set) positive overflows return the maximum positive number (0x7FFF FFFF), and negative overflows return the minimum negative number (0x8000 0000).

**Status Flags**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the fixed-point output is all 0s, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if the most significant output bit is 1, otherwise cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Set if the XOR of the carries of the two most significant adder stages is 1, otherwise cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Set if the carry from the most significant adder stage is 1, otherwise cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
Single-Function Operations

\[ R_n = R_x + R_y + C_l \]

**Function**

Adds with carry (\(AC\) from ASTAT) the fixed-point fields in registers \(R_x\) and \(R_y\). The result is placed in the fixed-point field in register \(R_n\). The floating-point extension field in \(R_n\) is set to all 0s. In saturation mode (the ALU saturation mode bit in MODE1 set) positive overflows return the maximum positive number (0x7FFF FFFF), and negative overflows return the minimum negative number (0x8000 0000).

**Status Flags**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the fixed-point output is all 0s, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if the most significant output bit is 1, otherwise cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Set if the XOR of the carries of the two most significant adder stages is 1, otherwise cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Set if the carry from the most significant adder stage is 1, otherwise cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
Compute Operation Reference

\[ R_n = R_x - R_y + C_I - 1 \]

**Function**

Subtracts with borrow \((AC - 1)\) from \(A\) the fixed-point field in register \(R_y\) from the fixed-point field in register \(R_x\). The result is placed in the fixed-point field in register \(R_n\). The floating-point extension field in \(R_n\) is set to all 0s. In saturation mode (the ALU saturation mode bit in MODE1 set) positive overflows return the maximum positive number \((0x7FFF FFFF)\), and negative overflows return the minimum negative number \((0x8000 0000)\).

**Status Flags**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the fixed-point output is all 0s, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if the most significant output bit is 1, otherwise cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Set if the XOR of the carries of the two most significant adder stages is 1, otherwise cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Set if the carry from the most significant adder stage is 1, otherwise cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
Single-Function Operations

\[ R_n = \frac{R_x + R_y}{2} \]

Function

Adds the fixed-point fields in registers \( R_x \) and \( R_y \) and divides the result by 2. The result is placed in the fixed-point field in register \( R_n \). The floating-point extension field in \( R_n \) is set to all 0s. Rounding is to nearest (IEEE) or by truncation, as defined by the rounding mode bit in the MODE1 register.

Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the fixed-point output is all 0s, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if the most significant output bit is 1, otherwise cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Set if the carry from the most significant adder stage is 1, otherwise cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>


Compute Operation Reference

**COMP(Rx, Ry)**

**Function**

Compares the fixed-point field in register Rx with the fixed-point field in register Ry. Sets the AZ flag if the two operands are equal, and the AN flag if the operand in register Rx is smaller than the operand in register Ry.

The ASTAT register stores the results of the previous eight ALU compare operations in bits 24:31. These bits are shifted right (bit 24 is overwritten) whenever a fixed-point or floating-point compare instruction is executed. The MSB of ASTAT is set if the X operand is greater than the Y operand (its value is the AND of AZ and AN); otherwise, it is cleared.

**Status Flags**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the operands in registers Rx and Ry are equal, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if the operand in the Rx register is smaller than the operand in the Ry register, otherwise cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
Single-Function Operations

**Rn = Rx + CI**

Function

Adds the fixed-point field in register $Rx$ with the carry flag from the ASTAT register ($AC$). The result is placed in the fixed-point field in register $Rn$. The floating-point extension field in $Rn$ is set to all $0$s. In saturation mode (the ALU saturation mode bit in MODE1 set) positive overflows return the maximum positive number ($0x7FFF FFFF$).

Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the fixed-point output is all $0$s, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if the most significant output bit is $1$, otherwise cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Set if the XOR of the carries of the two most significant adder stages is $1$, otherwise cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Set if the carry from the most significant adder stage is $1$, otherwise cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
**Rn = Rx + CI − 1**

**Function**

Adds the fixed-point field in register Rx with the borrow from the ASTAT register (AC −1). The result is placed in the fixed-point field in register Rn. The floating-point extension field in Rn is set to all 0s. In saturation mode (the ALU saturation mode bit in MODE1 set) positive overflows return the maximum positive number (0x7FFF FFFF).

**Status Flags**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the fixed-point output is all 0s, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if the most significant output bit is 1, otherwise cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Set if the XOR of the carries of the two most significant adder stages is 1, otherwise cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Set if the carry from the most significant adder stage is 1, otherwise cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
**Single-Function Operations**

**Rn = Rx + 1**

**Function**

Increments the fixed-point operand in register Rx. The result is placed in the fixed-point field in register Rn. The floating-point extension field in Rn is set to all 0s. In saturation mode (the ALU saturation mode bit in MODE1 set), overflow causes the maximum positive number (0x7FFF FFFF) to be returned.

**Status Flags**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the fixed-point output is all 0s, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if the most significant output bit is 1, otherwise cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Set if the XOR of the carries of the two most significant adder stages is 1, otherwise cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Set if the carry from the most significant adder stage is 1, otherwise cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
**Rn = Rx – 1**

**Function**

Decrement the fixed-point operand in register Rx. The result is placed in the fixed-point field in register Rn. The floating-point extension field in Rn is set to all 0s. In saturation mode (the ALU saturation mode bit in MODE1 set), underflow causes the minimum negative number (0x8000 0000) to be returned.

**Status Flags**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the fixed-point output is all 0s, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if the most significant output bit is 1, otherwise cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Set if the XOR of the carries of the two most significant adder stages is 1, otherwise cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Set if the carry from the most significant adder stage is 1, otherwise cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
**Single-Function Operations**

**Rn = -Rx**

**Function**

Negates the fixed-point operand in Rx by twos complement. The result is placed in the fixed-point field in register Rn. The floating-point extension field in Rn is set to all 0s. Negation of the minimum negative number (0x8000 0000) causes an overflow. In saturation mode (the ALU saturation mode bit in MODE1 set), overflow causes the maximum positive number (0x7FFF FFFF) to be returned.

**Status Flags**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the fixed-point output is all 0s.</td>
</tr>
<tr>
<td>AU</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if the most significant output bit is 1.</td>
</tr>
<tr>
<td>AV</td>
<td>Set if the XOR of the carries of the two most significant adder stages is 1.</td>
</tr>
<tr>
<td>AC</td>
<td>Set if the carry from the most significant adder stage is 1, otherwise cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
**Rn = ABS Rx**

**Function**

Determines the absolute value of the fixed-point operand in Rx. The result is placed in the fixed-point field in register Rn. The floating-point extension field in Rn is set to all 0s. ABS of the minimum negative number (0x8000 0000) causes an overflow. In saturation mode (the ALU saturation mode bit in MODE1 set), overflow causes the maximum positive number (0x7FFF FFFF) to be returned.

**Status Flags**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the fixed-point output is all 0s, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if the most significant output bit is 1, otherwise cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Set if the XOR of the carries of the two most significant adder stages is 1, otherwise cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Set if the carry from the most significant adder stage is 1, otherwise cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Set if the fixed-point operand in Rx is negative, otherwise cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
Single-Function Operations

Rn = PASS Rx

Function

Passes the fixed-point operand in Rx through the ALU to the fixed-point field in register Rn. The floating-point extension field in Rn is set to all 0s.

Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the fixed-point output is all 0s. otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if the most significant output bit is 1. otherwise cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
Compute Operation Reference

**Rn = Rx AND Ry**

**Function**

Logically ANDs the fixed-point operands in Rx and Ry. The result is placed in the fixed-point field in Rn. The floating-point extension field in Rn is set to all 0s.

**Status Flags**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the fixed-point output is all 0s, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if the most significant output bit is 1, otherwise cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
Single-Function Operations

\[ R_n = R_x \text{ OR } R_y \]

Function

Logically ORs the fixed-point operands in \( R_x \) and \( R_y \). The result is placed in the fixed-point field in \( R_n \). The floating-point extension field in \( R_n \) is set to all 0s.

Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the fixed-point output is all 0s, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if the most significant output bit is 1, otherwise cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
\[ R_n = R_x \text{ XOR } R_y \]

**Function**

Logically XORs the fixed-point operands in \( R_x \) and \( R_y \). The result is placed in the fixed-point field in \( R_n \). The floating-point extension field in \( R_n \) is set to all 0s.

**Status Flags**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the fixed-point output is all 0s, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if the most significant output bit is 1, otherwise cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
Single-Function Operations

\textbf{Rn = NOT Rx}

Function

Logically complements the fixed-point operand in Rx. The result is placed in the fixed-point field in Rn. The floating-point extension field in Rn is set to all 0s.

Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the fixed-point output is all Os, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if the most significant output bit is 1, otherwise cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
Compute Operation Reference

\( R_n = \text{MIN}(R_x, R_y) \)

**Function**

Returns the smaller of the two fixed-point operands in \( R_x \) and \( R_y \). The result is placed in the fixed-point field in register \( R_n \). The floating-point extension field in \( R_n \) is set to all 0s.

**Status Flags**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the fixed-point output is all 0s, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if the most significant output bit is 1, otherwise cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
Single-Function Operations

\[ R_n = \text{MAX}(R_x, R_y) \]

**Function**

Returns the larger of the two fixed-point operands in \( R_x \) and \( R_y \). The result is placed in the fixed-point field in register \( R_n \). The floating-point extension field in \( R_n \) is set to all 0s.

**Status Flags**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the fixed-point output is all 0s, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if the most significant output bit is 1, otherwise cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
**Rn = CLIP Rx BY Ry**

**Function**

Returns the fixed-point operand in \( Rx \) if the absolute value of the operand in \( Rx \) is less than the absolute value of the fixed-point operand in \( Ry \). Otherwise, returns \(|Ry|\) if \( Rx \) is positive, and \(-|Ry|\) if \( Rx \) is negative. The result is placed in the fixed-point field in register \( Rn \). The floating-point extension field in \( Rn \) is set to all 0s.

**Status Flags**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the fixed-point output is all 0s, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if the most significant output bit is 1, otherwise cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
Single-Function Operations

Fn = Fx + Fy

Function

Adds the floating-point operands in registers Fx and Fy. The normalized result is placed in register Fn. Rounding is to nearest (IEEE) or by truncation, to a 32-bit or to a 40-bit boundary, as defined by the rounding mode and rounding boundary bits in MODE1. Postrounded overflow returns ±Infinity (round-to-nearest) or ±NORM.MAX (round-to-zero). Postrounded denormal returns ±Zero. Denormal inputs are flushed to ±Zero. A NAN input returns an all 1s result.

Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the postrounded result is a denormal (unbiased exponent &lt; -126) or zero, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Set if the postrounded result is a denormal, otherwise cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if the floating-point result is negative, otherwise cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Set if the postrounded result overflows (unbiased exponent &gt; +127), otherwise cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Set if either of the input operands is a NAN, or if they are opposite-signed Infinities, otherwise cleared.</td>
</tr>
</tbody>
</table>
Compute Operation Reference

\[ \text{Fn} = \text{Fx} - \text{Fy} \]

**Function**

Subtracts the floating-point operand in register \( F_y \) from the floating-point operand in register \( F_x \). The normalized result is placed in register \( F_n \). Rounding is to nearest (IEEE) or by truncation, to a 32-bit or to a 40-bit boundary, as defined by the rounding mode and rounding boundary bits in MODE1. Postrounded overflow returns \( \pm \text{Infinity} \) (round-to-nearest) or \( \pm \text{NORM.MAX} \) (round-to-zero). Postrounded denormal returns \( \pm \text{Zero} \). Denormal inputs are flushed to \( \pm \text{Zero} \). A NAN input returns an all 1s result.

**Status Flags**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the postrounded result is a denormal (unbiased exponent &lt; -126) or zero, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Set if the postrounded result is a denormal, otherwise cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if the floating-point result is negative, otherwise cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Set if the postrounded result overflows (unbiased exponent &gt; +127), otherwise cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Set if either of the input operands is a NAN, or if they are like-signed Infinities, otherwise cleared.</td>
</tr>
</tbody>
</table>
Single-Function Operations

Fn = ABS (Fx + Fy)

Function

Adds the floating-point operands in registers Fx and Fy, and places the absolute value of the normalized result in register Fn. Rounding is to nearest (IEEE) or by truncation, to a 32-bit or to a 40-bit boundary, as defined by the rounding mode and rounding boundary bits in MODE1. Postrounded overflow returns +Infinity (round-to-nearest) or +NORM.MAX (round-to-zero). Postrounded denormal returns +Zero. Denormal inputs are flushed to ±Zero. A NAN input returns an all 1s result.

Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the postrounded result is a denormal (unbiased exponent $&lt;-126$) or zero, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Set if the postrounded result is a denormal, otherwise cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Set if the postrounded result overflows (unbiased exponent $&gt;+127$), otherwise cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Set if either of the input operands is a NAN, or if they are opposite-signed Infinities, otherwise cleared.</td>
</tr>
</tbody>
</table>
Compute Operation Reference

$F_n = \text{ABS}\ (F_x - F_y)$

Function

Subtracts the floating-point operand in $F_y$ from the floating-point operand in $F_x$ and places the absolute value of the normalized result in register $F_n$. Rounding is to nearest (IEEE) or by truncation, to a 32-bit or to a 40-bit boundary, as defined by the rounding mode and rounding boundary bits in MODE1. Postrounded overflow returns $+\text{Infinity}$ (round-to-nearest) or $+\text{NORM.MAX}$ (round-to-zero). Postrounded denormal returns $+\text{Zero}$. Denormal inputs are flushed to $\pm\text{Zero}$. A NAN input returns an all 1s result.

Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the postrounded result is a denormal (unbiased exponent $&lt;-126$) or zero, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Set if the postrounded result is a denormal, otherwise cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Set if the postrounded result overflows (unbiased exponent $&gt;+127$), otherwise cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Set if either of the input operands is a NaN, or if they are like-signed Infinities, otherwise cleared.</td>
</tr>
</tbody>
</table>
Single-Function Operations

\[
Fn = \frac{(Fx + Fy)}{2}
\]

Function

Adds the floating-point operands in registers \( Fx \) and \( Fy \) and divides the result by 2, by decrementing the exponent of the sum before rounding. The normalized result is placed in register \( Fn \). Rounding is to nearest (IEEE) or by truncation, to a 32-bit or to a 40-bit boundary, as defined by the rounding mode and rounding boundary bits in MODE1. Postrounded overflow returns ±Infinity (round-to-nearest) or ±NORM.MAX (round-to-zero). Postrounded denormal results return ±Zero. A denormal input is flushed to ±Zero. A NAN input returns an all 1s result.

Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the postrounded result is a denormal (unbiased exponent &lt; -126) or zero, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Set if the postrounded result is a denormal, otherwise cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if the floating-point result is negative, otherwise cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Set if the postrounded result overflows (unbiased exponent &gt; +127), otherwise cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Set if either of the input operands is a NAN, or if they are opposite-signed Infinities, otherwise cleared.</td>
</tr>
</tbody>
</table>
**COMP(Fx, Fy)**

**Function**

Compares the floating-point operand in register $Fx$ with the floating-point operand in register $Fy$. Sets the $AZ$ flag if the two operands are equal, and the $AN$ flag if the operand in register $Fx$ is smaller than the operand in register $Fy$.

The ASTAT register stores the results of the previous eight ALU compare operations in bits 24-31. These bits are shifted right (bit 24 is overwritten) whenever a fixed-point or floating-point compare instruction is executed. The MSB of ASTAT is set if the $X$-operand is greater than the $Y$-operand (its value is the AND of $AZ$ and $AN$); otherwise, it is cleared.

**Status Flags**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$AZ$</td>
<td>Set if the operands in registers $Fx$ and $Fy$ are equal, otherwise cleared.</td>
</tr>
<tr>
<td>$AU$</td>
<td>Cleared.</td>
</tr>
<tr>
<td>$AN$</td>
<td>Set if the operand in the $Fx$ register is smaller than the operand in the $Fy$ register, otherwise cleared.</td>
</tr>
<tr>
<td>$AV$</td>
<td>Cleared.</td>
</tr>
<tr>
<td>$AC$</td>
<td>Cleared.</td>
</tr>
<tr>
<td>$AS$</td>
<td>Cleared.</td>
</tr>
<tr>
<td>$AI$</td>
<td>Set if either of the input operands is a NAN, otherwise cleared.</td>
</tr>
</tbody>
</table>
Single-Function Operations

\[ F_n = -F_x \]

Function

Complements the sign bit of the floating-point operand in \( F_x \). The complemented result is placed in register \( F_n \). A denormal input is flushed to ±Zero. A NAN input returns an all 1s result.

Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the result operand is a ±Zero, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if the floating-point result is negative, otherwise cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Set if the input operand is a NAN, otherwise cleared.</td>
</tr>
</tbody>
</table>
**Compute Operation Reference**

**Fn = ABS Fx**

**Function**

Returns the absolute value of the floating-point operand in register \( F_x \) by setting the sign bit of the operand to 0. Denormal inputs are flushed to +Zero. A NAN input returns an all 1s result.

**Status Flags**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the result operand is +Zero, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Set if the input operand is negative, otherwise cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Set if the input operand is a NAN, otherwise cleared.</td>
</tr>
</tbody>
</table>
Single-Function Operations

Fn = PASS Fx

Function

Passes the floating-point operand in $F_x$ through the ALU to the floating-point field in register $F_n$. Denormal inputs are flushed to ±Zero. A NAN input returns an all 1s result.

Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the result operand is a ±Zero, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if the floating-point result is negative, otherwise cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Set if the input operand is a NAN, otherwise cleared.</td>
</tr>
</tbody>
</table>
**Fn = RND Fx**

**Function**

Rounds the floating-point operand in register \( F_x \) to a 32 bit boundary. Rounding is to nearest (IEEE) or by truncation, as defined by the rounding mode bit in MODE1. Postrounded overflow returns ±Infinity (round-to-nearest) or ±NORM.MAX (round-to-zero). A denormal input is flushed to ±Zero. A NAN input returns an all 1s result.

**Status Flags**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the result operand is a ±Zero, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if the floating-point result is negative, otherwise cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Set if the postrounded result overflows (unbiased exponent &gt; +127), otherwise cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Set if the input operand is a NAN, otherwise cleared.</td>
</tr>
</tbody>
</table>
Single-Function Operations

**Fn = SCALB Fx BY Ry**

**Function**

Scales the exponent of the floating-point operand in Fx by adding to it the fixed-point two's-complement integer in Ry. The scaled floating-point result is placed in register Fn. Overflow returns ±Infinity (round-to-nearest) or ±NORM.MAX (round-to-zero). Denormal returns ±Zero. Denormal inputs are flushed to ±Zero. A NAN input returns an all 1s result.

**Status Flags**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the result is a denormal (unbiased exponent &lt; -126) or zero, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Set if the postrounded result is a denormal, otherwise cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if the floating-point result is negative, otherwise cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Set if the result overflows (unbiased exponent &gt; +127), otherwise cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Set if the input is a NAN, otherwise cleared.</td>
</tr>
</tbody>
</table>
**Rn = MANT Fx**

**Function**

Extracts the mantissa (fraction bits with explicit hidden bit, excluding the sign bit) from the floating-point operand in $F_x$. The unsigned-magnitude result is left-justified (1.31 format) in the fixed-point field in $R_n$. Rounding modes are ignored and no rounding is performed because all results are inherently exact. Denormal inputs are flushed to ±Zero. A NAN or an Infinity input returns an all 1s result ($–1$ in signed fixed-point format).

**Status Flags**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the result is zero, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Set if the input is negative, otherwise cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Set if the input operands is a NAN or an Infinity, otherwise cleared.</td>
</tr>
</tbody>
</table>
Single-Function Operations

Rn = LOGB Fx

Function

Converts the exponent of the floating-point operand in register Fx to an unbiased twos-complement fixed-point integer. The result is placed in the fixed-point field in register Rn. Unbiasing is done by subtracting 127 from the floating-point exponent in Fx. If saturation mode is not set, a ±Infinity input returns a floating-point +Infinity and a ±Zero input returns a floating-point –Infinity. If saturation mode is set, a ±Infinity input returns the maximum positive value (0x7FFF FFFF), and a ±Zero input returns the maximum negative value (0x8000 0000). Denormal inputs are flushed to ±Zero. A NAN input returns an all 1s result.

Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the fixed-point result is zero, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if the result is negative, otherwise cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Set if the input operand is an Infinity or a Zero, otherwise cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Set if the input is a NAN, otherwise cleared.</td>
</tr>
</tbody>
</table>
Compute Operation Reference

\[ Rn = \text{FIX } Fx \]
\[ Rn = \text{TRUNC } Fx \]
\[ Rn = \text{FIX } Fx \text{ BY } Ry \]
\[ Rn = \text{TRUNC } Fx \text{ BY } Ry \]

Function

Converts the floating-point operand in \( Fx \) to a two's-complement 32-bit fixed-point integer result. If the MODE1 register TRUNC bit=1, the FIX operation truncates the mantissa towards \(-\text{Infinity}\). If the TRUNC bit=0, the FIX operation rounds the mantissa towards the nearest integer. The TRUNC operation always truncates toward 0. The TRUNC bit does not influence operation of the TRUNC instruction.

If a scaling factor (\( Ry \)) is specified, the fixed-point two's-complement integer in \( Ry \) is added to the exponent of the floating-point operand in \( Fx \) before the conversion. The result of the conversion is right-justified (32.0 format) in the fixed-point field in register \( Rn \). The floating-point extension field in \( Rn \) is set to all 0s. In saturation mode (the ALU saturation mode bit in MODE1 set) positive overflows and \( +\text{Infinity} \) return the maximum positive number (0x7FFF FFFF), and negative overflows and \( -\text{Infinity} \) return the minimum negative number (0x8000 0000).

For the FIX operation, rounding is to nearest (IEEE) or by truncation, as defined by the rounding mode bit in MODE1. A NaN input returns a floating-point all 1s result. If saturation mode is not set, an Infinity input or a result that overflows returns a floating-point result of all 1s. All positive underflows return zero (0). Negative underflows that are rounded-to-nearest return zero (0), and negative underflows that are rounded by truncation return \(-1\) (0xFF FFFF FF00).
Single-Function Operations

Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the fixed-point result is Zero, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Set if the pre-rounded result is a denormal, otherwise cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if the fixed-point result is negative, otherwise cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Set if the conversion causes the floating-point mantissa to be shifted left, i.e. if the floating-point exponent + scale bias is &gt;157 (127 + 31 - 1) or if the input is ±Infinity, otherwise cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Set if the input operand is a NAN or, when saturation mode is not set, either input is an Infinity or the result overflows, otherwise cleared.</td>
</tr>
</tbody>
</table>
**Function**

Converts the fixed-point operand in \( Rx \) to a floating-point result. If a scaling factor \( Ry \) is specified, the fixed-point two's-complement integer in \( Ry \) is added to the exponent of the floating-point result. The final result is placed in register \( Fn \).

Rounding is to nearest (IEEE) or by truncation, as defined by the rounding mode, to a 40-bit boundary, regardless of the values of the rounding boundary bits in MODE1. The exponent scale bias may cause a floating-point overflow or a floating-point underflow. Overflow generates a return of \( \pm \text{Infinity} \) (round-to-nearest) or \( \pm \text{NORM.MAX} \) (round-to-zero); underflow generates a return of \( \pm \text{Zero} \).

**Status Flags**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the result is a denormal (unbiased exponent &lt; -126) or zero, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Set if the postrounded result is a denormal, otherwise cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if the floating-point result is negative, otherwise cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Set if the result overflows (unbiased exponent &gt;127).</td>
</tr>
<tr>
<td>AC</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
Single-Function Operations

Fn = RECIPS Fx

Function

Creates an 8-bit accurate seed for 1/Fx, the reciprocal of Fx. The mantissa of the seed is determined from a ROM table using the seven MSBs (excluding the hidden bit) of the Fx mantissa as an index. The unbiased exponent of the seed is calculated as the twos complement of the unbiased Fx exponent, decremented by one; i.e., if e is the unbiased exponent of Fx, then the unbiased exponent of Fn = –e – 1. The sign of the seed is the sign of the input. ±Zero returns ±Infinity and sets the overflow flag. If the unbiased exponent of Fx is greater than +125, the result is ±Zero. A NAN input returns an all 1s result.

The following code performs floating-point division using an iterative convergence algorithm.* The result is accurate to one LSB in whichever format mode, 32-bit or 40-bit, is set. The following inputs are required: F0=numerator, F12=denominator, F11=2.0. The quotient is returned in F0.

(The two highlighted instructions can be removed if only a ±1 LSB accurate, single-precision result is necessary.)

F0=RECIPS F12, F7=F0;         {Get 8 bit seed R0=1/D}
F12=F0*F12;                   {D' = D*R0}
F7=F0*F7, F0=F11·F12;         {F0=R1=2·D', F7=N·R0}
F12=F0*F12;                   {F12=D'·D'*R1}
F7=F0*F7, F0=F11·F12;         {F7=N·R0·R1, F0=R2=2·D'}
F12=F0*F12;                   {F12=D'·D'*R2}
F7=F0*F7, F0=F11·F12;         {F7=N·R0·R1·R2, F0=R3=2·D'}
F0=F0*F7;                 {F7=N·R0·R1·R2·R3}

Note that this code segment can be made into a subroutine by adding an RTS(DB) clause to the third-to-last instruction.

# Compute Operation Reference

## Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the floating-point result is ±Zero (unbiased exponent of Fx is greater than +125), otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if the input operand is negative, otherwise cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Set if the input operand is ±Zero, otherwise cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Set if the input operand is a NAN, otherwise cleared.</td>
</tr>
</tbody>
</table>
Single-Function Operations

**Fn = RSQRTS Fx**

Function

Creates a 4-bit accurate seed for $1/\sqrt{Fx}$, the reciprocal square root of $Fx$. The mantissa of the seed is determined from a ROM table, using the LSB of the biased exponent of $Fx$ concatenated with the six MSBs (excluding the hidden bit of the mantissa) of $Fx$ as an index. The unbiased exponent of the seed is calculated as the twos complement of the unbiased $Fx$ exponent, shifted right by one bit and decremented by one; that is, if $e$ is the unbiased exponent of $Fx$, then the unbiased exponent of $Fn = -\text{INT}[e/2] - 1$. The sign of the seed is the sign of the input. The input ±Zero returns ±Infinity and sets the overflow flag. The input +Infinity returns +Zero. A NAN input or a negative nonzero input returns a result of all 1s.

The following code calculates a floating-point $1/\sqrt{x}$, reciprocal square root, using a Newton-Raphson iteration algorithm.* The result is accurate to one LSB in whichever format mode, 32-bit or 40-bit, is set. To calculate the square root, simply multiply the result by the original input. The following inputs are required: $F0=$input, $F8=3.0$, $F1=0.5$. The result is returned in $F4$.

(The four highlighted instructions can be removed if only a ±1 LSB accurate, single-precision result is necessary.)

```
F4=RSQRTS F0;                  {Fetch 4-bit seed}
F12=F4*F4;                     {F12=X0^2}
F12=F12*F0;                    {F12=C*X0^2}
F4=F1*F4, F12=F8-F12;          {F4=.5*X0, F12=3-C*X0^2}
F4=F4*F12;                     {F4=X1=.5*X0(3-C*X0^2)}
F12=F4*F4;                     {F12=X1^2}
F12=F12*F0;                    {F12=C*X1^2}
F4=F1*F4, F12=F8-F12;          {F4=.5*X1, F12=3-C*X1^2}
F4=F4*F12;                     {F4=X2=.5*X1(3-C*X1^2)}
F12=F4*F4;                     {F12=X2^2}
```

Compute Operation Reference

\[
\begin{align*}
F_{12} &= F_{12} \times F_0; \quad \{F_{12} = C \times X_2^2\} \\
F_4 &= F_1 \times F_4, \quad F_{12} = F_8 - F_{12}; \quad \{F_4 = 0.5 \times X_2, \quad F_{12} = 3 - C \times X_2^2\} \\
F_4 &= F_4 \times F_{12}; \quad \{F_4 = X_3 = 0.5 \times X_2(3 - C \times X_2^2)\}
\end{align*}
\]

Note that this code segment can be made into a subroutine by adding an RTS(DB) clause to the third-to-last instruction.

**Status Flags**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the floating-point result is +Zero (Fx = +Infinity), otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if the input operand is -Zero, otherwise cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Set if the input operand is ±Zero, otherwise cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Set if the input operand is negative and nonzero, or a NaN, otherwise cleared.</td>
</tr>
</tbody>
</table>
Single-Function Operations

**Fn = Fx COPYSIGN Fy**

**Function**

Copies the sign of the floating-point operand in register \( F_y \) to the floating-point operand from register \( F_x \) without changing the exponent or the mantissa. The result is placed in register \( F_n \). A denormal input is flushed to ±Zero. A NAN input returns an all 1s result.

**Status Flags**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the floating-point result is ±Zero, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if the floating-point result is negative, otherwise cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Set if either of the input operands is a NAN, otherwise cleared.</td>
</tr>
</tbody>
</table>
Compute Operation Reference

Fn = MIN(Fx, Fy)

Function

Returns the smaller of the floating-point operands in register Fx and Fy. A NAN input returns an all 1s result. MIN of +Zero and −Zero returns −Zero. Denormal inputs are flushed to ±Zero.

Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the floating-point result is ±Zero, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if the floating-point result is negative, otherwise cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Set if either of the input operands is a NAN, otherwise cleared.</td>
</tr>
</tbody>
</table>
Single-Function Operations

**Fn = MAX(Fx, Fy)**

Function

Returns the larger of the floating-point operands in registers Fx and Fy. A NAN input returns an all 1s result. MAX of +Zero and −Zero returns +Zero. Denormal inputs are flushed to ±Zero.

Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the floating-point result is ±Zero, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if the floating-point result is negative, otherwise cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Set if either of the input operands is a NAN, otherwise cleared.</td>
</tr>
</tbody>
</table>
Compute Operation Reference

Fn = CLIP Fx BY Fy

Function

Returns the floating-point operand in Fx if the absolute value of the operand in Fx is less than the absolute value of the floating-point operand in Fy. Otherwise, returns |Fy| if Fx is positive, and −|Fy| if Fx is negative. A NAN input returns an all 1s result. Denormal inputs are flushed to ±Zero.

Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if the floating-point result is ±Zero, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if the floating-point result is negative, otherwise cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Set if either of the input operands is a NAN, otherwise cleared.</td>
</tr>
</tbody>
</table>
Multiplier Operations

The Multiplier operations are described in this section. Table B-3 summarizes the syntax and opcodes for the fixed-point and floating-point Multiplier operations. The rest of this section contains detailed descriptions of each operation.

Table B-3. Summary of multiplier operations

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rn = Rx*Ry \mod 2</td>
<td>01yx f00r</td>
</tr>
<tr>
<td>MRF = Rx*Ry \mod 2</td>
<td>01yx f10r</td>
</tr>
<tr>
<td>MRB = Rx*Ry \mod 2</td>
<td>01yx f11r</td>
</tr>
<tr>
<td>Rn = MRF +Rx*Ry \mod 2</td>
<td>10yx f00r</td>
</tr>
<tr>
<td>Rn = MRB +Rx*Ry \mod 2</td>
<td>10yx f01r</td>
</tr>
<tr>
<td>MRF = MRF +Rx*Ry \mod 2</td>
<td>10yx f10r</td>
</tr>
<tr>
<td>MRB = MRB +Rx*Ry \mod 2</td>
<td>10yx f11r</td>
</tr>
<tr>
<td>Rn = MRF -Rx*Ry \mod 2</td>
<td>11yx f00r</td>
</tr>
<tr>
<td>Rn = MRB -Rx*Ry \mod 2</td>
<td>11yx f01r</td>
</tr>
<tr>
<td>MRF = MRF -Rx*Ry \mod 2</td>
<td>11yx f10r</td>
</tr>
<tr>
<td>MRB = MRB -Rx*Ry \mod 2</td>
<td>11yx f11r</td>
</tr>
<tr>
<td>Rn = SAT MRF \mod 1^3</td>
<td>0000 f00x</td>
</tr>
<tr>
<td>Rn = SAT MRB \mod 1</td>
<td>0000 f01x</td>
</tr>
<tr>
<td>MRF = SAT MRF \mod 1</td>
<td>0000 f10x</td>
</tr>
</tbody>
</table>
As shown in Table B-3, many Multiplier operations can include an optional modifier, mod1 or mod2.

Table B-4 on page B-52 lists the options and corresponding opcode values for mod2. The options, enclosed in parentheses, consists of three or four letters that indicate whether the x-input is signed (S) or unsigned (U), whether the y-input is signed or unsigned, whether the inputs are in

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRB = SAT MRB mod1</td>
<td>0000 f11x</td>
</tr>
<tr>
<td>Rn = RND MRF mod1</td>
<td>0001 100x</td>
</tr>
<tr>
<td>Rn = RND MRB mod1</td>
<td>0001 101x</td>
</tr>
<tr>
<td>MRF = RND MRF mod1</td>
<td>0001 110x</td>
</tr>
<tr>
<td>MRB = RND MRB mod1</td>
<td>0001 111x</td>
</tr>
<tr>
<td>MRF = 0</td>
<td>0001 0100</td>
</tr>
<tr>
<td>MRB = 0</td>
<td>0001 0110</td>
</tr>
<tr>
<td>MR = Rn</td>
<td></td>
</tr>
<tr>
<td>Rn = MR</td>
<td></td>
</tr>
<tr>
<td>Fn = Fx*Fy</td>
<td>0011 0000</td>
</tr>
</tbody>
</table>

1 y = y-input (1 = signed, 0 = unsigned)
   x = x-input (1 = signed, 0 = unsigned)
   f = format (1 = fractional, 0 = integer)
   r = rounding (1 = yes, 0 = no)
   R = fixed-point
   F = floating-point
2 For mod2 codes, see Table B-4.
3 For mod1 codes, see Table B-5.
Multiplier Operations

integer (I) or fractional (F) format, and whether the result is rounded-to-nearest (R) when written to the Register File.

Table B-4. Multiplier Mod2 Options

<table>
<thead>
<tr>
<th>Mod2</th>
<th>Opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>(SSI)</td>
<td>_ _11 0_0</td>
</tr>
<tr>
<td>(SUI)</td>
<td>_ _01 0_0</td>
</tr>
<tr>
<td>(USI)</td>
<td>_ _10 0_0</td>
</tr>
<tr>
<td>(UUI)</td>
<td>_ _00 0_0</td>
</tr>
<tr>
<td>(SSF)</td>
<td>_ _11 1_0</td>
</tr>
<tr>
<td>(SUU)</td>
<td>_ _01 1_0</td>
</tr>
<tr>
<td>(USU)</td>
<td>_ _10 1_0</td>
</tr>
<tr>
<td>(UUF)</td>
<td>_ _00 1_0</td>
</tr>
<tr>
<td>(SSF)</td>
<td>_ _11 1_1</td>
</tr>
<tr>
<td>(SUFR)</td>
<td>_ _01 1_1</td>
</tr>
<tr>
<td>(USFR)</td>
<td>_ _10 1_1</td>
</tr>
<tr>
<td>(UUF)</td>
<td>_ _00 1_1</td>
</tr>
</tbody>
</table>

Table B-5 on page B-53 lists the options and corresponding opcode values for mod1. The options, enclosed in parentheses, consist of two letters that indicate whether the input is signed (S) or unsigned (U) and whether the input is in integer (I) or fractional (F) format.
### Table B-5. Multiplier Mod1 Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>(SI) (for SAT only)</td>
<td>_ _ _ _ 0 _ _ 1</td>
</tr>
<tr>
<td>(UI) (for SAT only)</td>
<td>_ _ _ _ 0 _ _ 0</td>
</tr>
<tr>
<td>(SF)</td>
<td>_ _ _ _ 1 _ _ 1</td>
</tr>
<tr>
<td>(UF)</td>
<td>_ _ _ _ 1 _ _ 0</td>
</tr>
</tbody>
</table>
Multiplier Operations

\[ R_n = Rx \times Ry \mod 2 \]
\[ MRF = Rx \times Ry \mod 2 \]
\[ MRB = Rx \times Ry \mod 2 \]

Function

Multiplies the fixed-point fields in registers \( Rx \) and \( Ry \). If rounding is specified (fractional data only), the result is rounded. The result is placed either in the fixed-point field in register \( R_n \) or one of the MR accumulation registers. If \( R_n \) is specified, only the portion of the result that has the same format as the inputs is transferred (bits 31:0 for integers, bits 63:32 for fractional). The floating-point extension field in \( R_n \) is set to all 0s. If \( MRF \) or \( MRB \) is specified, the entire 80-bit result is placed in \( MRF \) or \( MRB \).

Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MN</td>
<td>Set if the result is negative, otherwise cleared.</td>
</tr>
<tr>
<td>MV</td>
<td>Set if the upper bits are not all zeros (signed or unsigned result) or ones (signed result). Number of upper bits depends on format. For a signed result, fractional=33, integer=49. For an unsigned result, fractional=32, integer=48.</td>
</tr>
<tr>
<td>MU</td>
<td>Set if the upper 48 bits of a fractional result are all zeros (signed or unsigned result) or ones (signed result) and the lower 32 bits are not all zeros. Integer results do not underflow.</td>
</tr>
<tr>
<td>MI</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
Compute Operation Reference

\[ Rn = MRF + Rx \times Ry \mod 2 \]
\[ Rn = MRB + Rx \times Ry \mod 2 \]
\[ MRF = MRF + Rx \times Ry \mod 2 \]
\[ MRB = MRB + Rx \times Ry \mod 2 \]

Function

Multiplies the fixed-point fields in registers \( Rx \) and \( Ry \), and adds the product to the specified MR register value. If rounding is specified (fractional data only), the result is rounded. The result is placed either in the fixed-point field in register \( Rn \) or one of the MR accumulation registers, which must be the same MR register that provided the input. If \( Rn \) is specified, only the portion of the result that has the same format as the inputs is transferred (bits 31:0 for integers, bits 63:32 for fractional). The floating-point extension field in \( Rn \) is set to all 0s. If \( MRF \) or \( MRB \) is specified, the entire 80-bit result is placed in \( MRF \) or \( MRB \).

Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MN</td>
<td>Set if the result is negative, otherwise cleared.</td>
</tr>
<tr>
<td>MV</td>
<td>Set if the upper bits are not all zeros (signed or unsigned result) or ones (signed result). Number of upper bits depends on format. For a signed result, fractional=33, integer=49. For an unsigned result, fractional=32, integer=48.</td>
</tr>
<tr>
<td>MU</td>
<td>Set if the upper 48 bits of a fractional result are all zeros (signed or unsigned result) or ones (signed result) and the lower 32 bits are not all zeros. Integer results do not underflow.</td>
</tr>
<tr>
<td>MI</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
Multiplier Operations

\[
R_n = MRF - Rx \times Ry \mod 2 \\
R_n = MRB - Rx \times Ry \mod 2 \\
MRF = MRF - Rx \times Ry \mod 2 \\
MRB = MRB - Rx \times Ry \mod 2
\]

Function

Multiplies the fixed-point fields in registers \(Rx\) and \(Ry\), and subtracts the product from the specified \(MR\) register value. If rounding is specified (fractional data only), the result is rounded. The result is placed either in the fixed-point field in register \(Rn\) or in one of the \(MR\) accumulation registers, which must be the same \(MR\) register that provided the input. If \(Rn\) is specified, only the portion of the result that has the same format as the inputs is transferred (bits 31:0 for integers, bits 63:32 for fractional). The floating-point extension field in \(Rn\) is set to all 0s. If \(MRF\) or \(MRB\) is specified, the entire 80-bit result is placed in \(MRF\) or \(MRB\).

Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MN</td>
<td>Set if the result is negative, otherwise cleared.</td>
</tr>
<tr>
<td>MV</td>
<td>Set if the upper bits are not all zeros (signed or unsigned result) or ones (signed result). Number of upper bits depends on format. For a signed result, fractional=33, integer=49. For an unsigned result, fractional=32, integer=48.</td>
</tr>
<tr>
<td>MU</td>
<td>Set if the upper 48 bits of a fractional result are all zeros (signed or unsigned result) or ones (signed result) and the lower 32 bits are not all zeros. Integer results do not underflow.</td>
</tr>
<tr>
<td>MI</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
Compute Operation Reference

**Function**

If the value of the specified MR register is greater than the maximum value for the specified data format, the Multiplier sets the result to the maximum value. Otherwise, the MR value is unaffected. The result is placed either in the fixed-point field in register Rn or one of the MR accumulation registers, which must be the same MR register that provided the input. If Rn is specified, only the portion of the result that has the same format as the inputs is transferred (bits 31:0 for integers, bits 63:32 for fractional). The floating-point extension field in Rn is set to all 0s. If MRF or MRB is specified, the entire 80-bit result is placed in MRF or MRB.

**Status Flags**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MN</td>
<td>Set if the result is negative, otherwise cleared.</td>
</tr>
<tr>
<td>MV</td>
<td>Cleared.</td>
</tr>
<tr>
<td>MU</td>
<td>Set if the upper 48 bits of a fractional result are all zeros (signed or unsigned result) or ones (signed result) and the lower 32 bits are not all zeros. Integer results do not underflow.</td>
</tr>
<tr>
<td>MI</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
Multiplier Operations

\[ \text{Rn} = \text{RND MRF \ mod1} \]
\[ \text{Rn} = \text{RND MRB \ mod1} \]
\[ \text{MRF} = \text{RND MRF \ mod1} \]
\[ \text{MRB} = \text{RND MRB \ mod1} \]

Function

Rounds the specified MR value to nearest at bit 32 (the MR1-MR0 boundary). The result is placed either in the fixed-point field in register Rn or one of the MR accumulation registers, which must be the same MR register that provided the input. If Rn is specified, only the portion of the result that has the same format as the inputs is transferred (bits 31:0 for integers, bits 63:32 for fractional). The floating-point extension field in Rn is set to all 0s. If MRF or MRB is specified, the entire 80-bit result is placed in MRF or MRB.

Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MN</td>
<td>Set if the result is negative, otherwise cleared.</td>
</tr>
<tr>
<td>MV</td>
<td>Set if the upper bits are not all zeros (signed or unsigned result) or ones (signed result). Number of upper bits depends on format. For a signed result, fractional=33, integer=49. For an unsigned result, fractional=32, integer=48.</td>
</tr>
<tr>
<td>MU</td>
<td>Set if the upper 48 bits of a fractional result are all zeros (signed or unsigned result) or ones (signed result) and the lower 32 bits are not all zeros. Integer results do not underflow.</td>
</tr>
<tr>
<td>MI</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
MRF = 0
MRB = 0

Function

Sets the value of the specified MR register to zero (0). All 80 bits (MR2, MR1, MR0) are cleared.

Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MN</td>
<td>Cleared.</td>
</tr>
<tr>
<td>MV</td>
<td>Cleared.</td>
</tr>
<tr>
<td>MU</td>
<td>Cleared.</td>
</tr>
<tr>
<td>MI</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
## Multiplier Operations

### MR = Rn/Rn = MR

#### Function

A transfer to a MR register places the fixed-point field of register Rn in the specified MR register. The floating-point extension field in Rn is ignored. A transfer from an MR register places the specified MR register in the fixed-point field in register Rn. The floating-point extension field in Rn is set to all 0s.

#### Syntax Variations

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MR0F = Rn</td>
<td>Rn = MR0F</td>
</tr>
<tr>
<td>MR1F = Rn</td>
<td>Rn = MR1F</td>
</tr>
<tr>
<td>MR2F = Rn</td>
<td>Rn = MR2F</td>
</tr>
<tr>
<td>MR0B = Rn</td>
<td>Rn = MR0B</td>
</tr>
<tr>
<td>MR1B = Rn</td>
<td>Rn = MR1B</td>
</tr>
<tr>
<td>MR2B = Rn</td>
<td>Rn = MR2B</td>
</tr>
</tbody>
</table>

#### Compute Field

The MR register is specified by Ai and the data register by Rk. The direction of the transfer is determined by T (0=to Register File, 1=to MR register).

<table>
<thead>
<tr>
<th>Ai</th>
<th>MR Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>MR0F</td>
</tr>
<tr>
<td>0001</td>
<td>MR1F</td>
</tr>
<tr>
<td>0010</td>
<td>MR2F</td>
</tr>
<tr>
<td>0100</td>
<td>MR0B</td>
</tr>
<tr>
<td>0101</td>
<td>MR1B</td>
</tr>
<tr>
<td>0110</td>
<td>MR2B</td>
</tr>
<tr>
<td>1000</td>
<td>MR0F</td>
</tr>
<tr>
<td>1001</td>
<td>MR1F</td>
</tr>
<tr>
<td>1010</td>
<td>MR2F</td>
</tr>
<tr>
<td>1100</td>
<td>MR0B</td>
</tr>
<tr>
<td>1101</td>
<td>MR1B</td>
</tr>
<tr>
<td>1110</td>
<td>MR2B</td>
</tr>
</tbody>
</table>

---

B-60 ADSP-21065L SHARC DSP Technical Reference
### Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MN</td>
<td>Cleared.</td>
</tr>
<tr>
<td>MV</td>
<td>Cleared.</td>
</tr>
<tr>
<td>MU</td>
<td>Cleared.</td>
</tr>
<tr>
<td>MI</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>

#### Ai MR Register

<table>
<thead>
<tr>
<th>Ai</th>
<th>MR Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>0101</td>
<td>MR1B</td>
</tr>
<tr>
<td>0110</td>
<td>MR2B</td>
</tr>
</tbody>
</table>
## Multiplier Operations

\[
Fn = Fx \times Fy
\]

**Function**

Multiplies the floating-point operands in registers \( Fx \) and \( Fy \). The result is placed in the register \( Fn \).

**Status Flags**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MN</td>
<td>Set if the result is negative, otherwise cleared.</td>
</tr>
<tr>
<td>MV</td>
<td>Set if the unbiased exponent of the result is greater than 127, otherwise cleared.</td>
</tr>
<tr>
<td>MU</td>
<td>Set if the unbiased exponent of the result is less than –126, otherwise cleared.</td>
</tr>
<tr>
<td>MI</td>
<td>Set if either input is a NAN or if the inputs are ±Infinity and ±Zero, otherwise cleared.</td>
</tr>
</tbody>
</table>

Reminder: The individual registers file are prefixed with an “\( F \)” when used in floating-point computations. The registers are prefixed with an “\( R \)” when used in fixed-point computations. The following instructions, for example, use the same registers:

\[
F0=F1 \times F2; \text{ floating-point multiply}
R0=R1 \times R2; \text{ fixed-point multiply}
\]

The \( F \) and \( R \) prefixes do not affect the 32-bit (or 40-bit) data transfer; they determine only how the ALU, Multiplier, or Shifter treat the data. The \( F \) and \( R \) can be either uppercase or lowercase since the assembler is case-insensitive.
Shifter Operations

Shifter operations are described in this section. Table B-6 summarizes the syntax and opcodes for the Shifter operations. The succeeding pages provide detailed descriptions of each operation.

The Shifter operates on the Register File’s 32-bit fixed-point fields (bits 39:8). Two-input Shifter operations can take their y-input from the Register File or from immediate data provided in the instruction. Either form uses the same opcode. However, the latter case, called an immediate shift or Shifter immediate operation, is allowed only with instruction type 6, which has an immediate data field in its opcode for this purpose. All other instruction types must obtain the y-input from the Register File when the compute operation is a two-input Shifter operation.

Table B-6. Summary of Shifter operations

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rn = LSHIFT Rx BY Ry</td>
<td>&lt;data8&gt;</td>
</tr>
<tr>
<td>Rn = Rn OR LSHIFT Rx BY Ry</td>
<td>&lt;data8&gt;</td>
</tr>
<tr>
<td>Rn = ASHIFT Rx BY Ry</td>
<td>&lt;data8&gt;</td>
</tr>
<tr>
<td>Rn = Rn OR ASHIFT Rx BY Ry</td>
<td>&lt;data8&gt;</td>
</tr>
<tr>
<td>Rn = ROT Rx BY Ry</td>
<td>&lt;data8&gt;</td>
</tr>
<tr>
<td>Rn = BCLR Rx BY Ry</td>
<td>&lt;data8&gt;</td>
</tr>
<tr>
<td>Rn = BSET Rx BY Ry</td>
<td>&lt;data8&gt;</td>
</tr>
<tr>
<td>Rn = BTGL Rx BY Ry</td>
<td>&lt;data8&gt;</td>
</tr>
</tbody>
</table>

(SE) = Sign extension of deposited or extracted field.
(EX) = Extended exponent extract.
## Shifter Operations

Table B-6. Summary of Shifter operations (Cont’d)

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>BTST Rx BY Ry</td>
<td>&lt;data8&gt;</td>
</tr>
<tr>
<td>Rn = FDEP Rx BY Ry</td>
<td>&lt;bit6&gt;:&lt;len6&gt;</td>
</tr>
<tr>
<td>Rn = Rn OR FDEP Rx BY Ry</td>
<td>&lt;bit6&gt;:&lt;len6&gt;</td>
</tr>
<tr>
<td>Rn = FDEP Rx BY Ry</td>
<td>&lt;bit6&gt;:&lt;len6&gt; (SE)</td>
</tr>
<tr>
<td>Rn = Rn OR FDEP Rx BY Ry</td>
<td>&lt;bit6&gt;:&lt;len6&gt;(SE)</td>
</tr>
<tr>
<td>Rn = FEXT RX BY Ry</td>
<td>&lt;bit6&gt;:&lt;len6&gt;</td>
</tr>
<tr>
<td>Rn = FEXT Rx BY Ry</td>
<td>&lt;bit6&gt;:&lt;len6&gt;(SE)</td>
</tr>
<tr>
<td>Rn = EXP Rx</td>
<td>1000 0000</td>
</tr>
<tr>
<td>Rn = EXP Rx (EX)</td>
<td>1000 0100</td>
</tr>
<tr>
<td>Rn = LEFTZ Rx</td>
<td>1000 1000</td>
</tr>
<tr>
<td>Rn = LEFTO Rx</td>
<td>1000 1100</td>
</tr>
<tr>
<td>Rn = FPACK Fx</td>
<td>1001 0000</td>
</tr>
<tr>
<td>Fn = FUNPACK Rx</td>
<td>1001 0100</td>
</tr>
</tbody>
</table>

(SE) = Sign extension of deposited or extracted field.  
(EX) = Extended exponent extract.
**Rn = LSHIFT Rx BY Ry**
**Rn = LSHIFT Rx BY <data8>**

**Function**

Logically shifts the fixed-point operand in register $Rx$ by the 32-bit value in register $Ry$ or by the 8-bit immediate value in the instruction. The shifted result is placed in the fixed-point field of register $Rn$. The floating-point extension field of $Rn$ is set to all 0s. The shift values are two's-complement numbers. Positive values select a left shift, negative values select a right shift. The 8-bit immediate data can take values between $-128$ and $127$ inclusive, which accommodates a shift of a 32-bit field from off-scale right to off-scale left.

**Status Flags**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$SZ$</td>
<td>Set if the shifted result is zero, otherwise cleared.</td>
</tr>
<tr>
<td>$SV$</td>
<td>Set if the input is shifted to the left by more than 0, otherwise cleared.</td>
</tr>
<tr>
<td>$SS$</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>


**Shifter Operations**

\[
Rn = Rn \text{ OR LSHIFT } Rx \text{ BY } Ry
\]
\[
Rn = Rn \text{ OR LSHIFT } Rx \text{ BY } \text{<data8>}
\]

**Function**

Logically shifts the fixed-point operand in register \( Rx \) by the 32-bit value in register \( Ry \) or by the 8-bit immediate value in the instruction. The shifted result is logically ORed with the fixed-point field of register \( Rn \) and then written back to register \( Rn \). The floating-point extension field of \( Rn \) is set to all 0s. The shift values are twos-complement numbers. Positive values select a left shift, negative values select a right shift. The 8-bit immediate data can take values between \(-128\) and \(127\) inclusive, which accommodates a 32-bit field from off-scale right to off-scale left.

**Status Flags**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SZ</td>
<td>Set if the shifted result is zero, otherwise cleared.</td>
</tr>
<tr>
<td>SV</td>
<td>Set if the input is shifted left by more than 0, otherwise cleared.</td>
</tr>
<tr>
<td>SS</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
Rn = ASHIFT Rx BY Ry
Rn = ASHIFT Rx BY <data8>

Function

Arithmetically shifts the fixed-point operand in register Rx by the 32-bit value in register Ry or by the 8-bit immediate value in the instruction. The shifted result is placed in the fixed-point field of register Rn. The floating-point extension field of Rn is set to all 0s. The shift values are twos-complement numbers. Positive values select a left shift, negative values select a right shift. The 8-bit immediate data can take values between $-128$ and $127$ inclusive, which accommodates a 32-bit field from off-scale right to off-scale left.

Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SZ</td>
<td>Set if the shifted result is zero, otherwise cleared.</td>
</tr>
<tr>
<td>SV</td>
<td>Set if the input is shifted left by more than 0, otherwise cleared.</td>
</tr>
<tr>
<td>SS</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
Shifter Operations

\[ Rn = Rn \text{ OR } \text{ASHIFT } Rx \text{ BY } Ry \]
\[ Rn = Rn \text{ OR } \text{ASHIFT } Rx \text{ BY } \text{<data8>} \]

**Function**

Arithmetically shifts the fixed-point operand in register \( Rx \) by the 32-bit value in register \( Ry \) or by the 8-bit immediate value in the instruction. The shifted result is logically ORed with the fixed-point field of register \( Rn \) and then written back to register \( Rn \). The floating-point extension field of \( Rn \) is set to all 0s. The shift values are twos-complement numbers. Positive values select a left shift, negative values select a right shift. The 8-bit immediate data can take values between \(-128\) and \(127\) inclusive, which accommodates a 32-bit field from off-scale right to off-scale left.

**Status Flags**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SZ</td>
<td>Set if the shifted result is zero, otherwise cleared.</td>
</tr>
<tr>
<td>SV</td>
<td>Set if the input is shifted left by more than 0, otherwise cleared.</td>
</tr>
<tr>
<td>SS</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>

B-68       ADSP-21065L SHARC DSP Technical Reference
Compute Operation Reference

\[ R_n = \text{ROT} \, R_x \, \text{BY} \, R_y \]
\[ R_n = \text{ROT} \, R_x \, \text{BY} \, \text{<data8>} \]

Function

Rotates the fixed-point operand in register \( R_x \) by the 32-bit value in register \( R_y \) or by the 8-bit immediate value in the instruction. The rotated result is placed in the fixed-point field of register \( R_n \). The floating-point extension field of \( R_n \) is set to all 0s. The shift values are two's-complement numbers. Positive values select a rotate left; negative values select a rotate right. The 8-bit immediate data can take values between \(-128\) and \(127\) inclusive, which accommodates a 32-bit field from full right wrap around to full left wrap around.

Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SZ</td>
<td>Set if the rotated result is zero, otherwise cleared.</td>
</tr>
<tr>
<td>SV</td>
<td>Cleared.</td>
</tr>
<tr>
<td>SS</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
Shifter Operations

\[ R_n = \text{BCLR} \ Rx \ BY \ Ry \]
\[ R_n = \text{BCLR} \ Rx \ BY \ <\text{data8}> \]

Function

Clears a bit in the fixed-point operand in register \( Rx \). The result is placed in the fixed-point field of register \( R_n \). The floating-point extension field of \( R_n \) is set to all 0s. The position of the bit is the 32-bit value in register \( Ry \) or the 8-bit immediate value in the instruction. The 8-bit immediate data can take values between 31 and 0 inclusive, allowing for any bit within a 32-bit field to be cleared. If the bit position value is greater than 31 or less than 0, no bits are cleared.

Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( SZ )</td>
<td>Set if the output operand is 0, otherwise cleared.</td>
</tr>
<tr>
<td>( SV )</td>
<td>Set if the bit position is greater than 31, otherwise cleared.</td>
</tr>
<tr>
<td>( SS )</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>

Note: This compute operation affects a bit in a Register File location. There is also a bit manipulation instruction that affects one or more bits in a system register. This BIT CLR instruction should not be confused with the BCLR Shifter operation. See Appendix E, Control and Status Registers for more information on BIT CLR.
Rn = BSET Rx BY Ry
Rn = BSET Rx BY <data8>

Function
Sets a bit in the fixed-point operand in register Rx. The result is placed in the fixed-point field of register Rn. The floating-point extension field of Rn is set to all 0s. The position of the bit is the 32-bit value in register Ry or the 8-bit immediate value in the instruction. The 8-bit immediate data can take values between 31 and 0 inclusive, allowing for any bit within a 32-bit field to be set. If the bit position value is greater than 31 or less than 0, no bits are set.

Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SZ</td>
<td>Set if the output operand is 0, otherwise cleared.</td>
</tr>
<tr>
<td>SV</td>
<td>Set if the bit position is greater than 31, otherwise cleared.</td>
</tr>
<tr>
<td>SS</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>

Note: This compute operation affects a bit in a Register File location. There is also a bit manipulation instruction that affects one or more bits in a system register. This BIT SET instruction should not be confused with the BSET Shifter operation. See Appendix E, Control and Status Registers for more information on BIT SET.
Shifter Operations

\[
R_n = \text{BTGL} \; R_x \; BY \; R_y \\
R_n = \text{BTGL} \; R_x \; BY \; <\text{data8}>
\]

Function

Toggles a bit in the fixed-point operand in register \( R_x \). The result is placed in the fixed-point field of register \( R_n \). The floating-point extension field of \( R_n \) is set to all 0s. The position of the bit is the 32-bit value in register \( R_y \) or the 8-bit immediate value in the instruction. The 8-bit immediate data can take values between 31 and 0 inclusive, allowing for any bit within a 32-bit field to be toggled. If the bit position value is greater than 31 or less than 0, no bits are toggled.

Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SZ</td>
<td>Set if the output operand is 0, otherwise cleared.</td>
</tr>
<tr>
<td>SV</td>
<td>Set if the bit position is greater than 31, otherwise cleared.</td>
</tr>
<tr>
<td>SS</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>

Note: This compute operation affects a bit in a Register File location. There is also a bit manipulation instruction that affects one or more bits in a system register. This BIT TGL instruction should not be confused with the BTGL Shifter operation. See Appendix E, Control and Status Registers for more information on BIT TGL.
BTST Rx BY Ry
BTST Rx BY <data8>

Function
Tests a bit in the fixed-point operand in register Rx. The $SZ$ flag is set if the bit is a 0 and cleared if the bit is a 1. The position of the bit is the 32-bit value in register Ry or the 8-bit immediate value in the instruction. The 8-bit immediate data can take values between 31 and 0 inclusive, allowing for any bit within a 32-bit field to be tested. If the bit position value is greater than 31 or less than 0, no bits are tested.

Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$SZ$</td>
<td>Cleared if the tested bit is a 1, is set if the tested bit is a 0 or if the bit position is greater than 31.</td>
</tr>
<tr>
<td>$SV$</td>
<td>Set if the bit position is greater than 31, otherwise cleared.</td>
</tr>
<tr>
<td>$SS$</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>

This compute operation tests a bit in a Register File location. There is also a bit manipulation instruction that tests one or more bits in a system register. This BIT TST instruction should not be confused with the BTST Shifter operation. See Appendix E, Control and Status Registers for more information on BIT TST.
Shifter Operations

**Rn = FDEP Rx BY Ry**

**Rn = FDEP Rx BY <bit6>:<len6>**

**Function**

Deposits a field from register `Rx` to register `Rn`.

The input field is right-aligned within the fixed-point field of `Rx` (see **Figure B-1**). Its length is determined by the `len6` field in register `Ry` or by the immediate `len6` field in the instruction.

The field is deposited in the fixed-point field of `Rn`, starting from a bit position determined by the `bit6` field in register `Ry` or by the immediate `bit6` field in the instruction.

Bits to the left and to the right of the deposited field are set to 0. The floating-pt. extension field of `Rn` (bits 7:0 of the 40-bit word) is set to all 0s.

`bit6` and `len6` can take values between 0 and 63 inclusive, allowing for deposit of fields ranging in length from 0 to 32 bits, and to bit positions ranging from 0 to off-scale left.

**Figure B-1. Field alignment**
Example

If $\text{len}_6=14$ and $\text{bit}_6=13$, then the 14 bits of $\text{Rx}$ are deposited in $\text{Rn}$ bits 34-21 (of the 40-bit word).

```
  39  31  23  15  7  0
|--------|--------|--abcdef|ghijklmn|--------|
\--------------/
  14 bits

  39  31  23  15  7  0
|00000abc|defghijk|lmn00000|00000000|00000000|00000000|
\--------------/
    bit position 13 (from reference point)
```

Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SZ</td>
<td>Set if the output operand is 0, otherwise cleared.</td>
</tr>
<tr>
<td>SV</td>
<td>Set if any bits are deposited to the left of the 32-bit fixed-point output field (i.e., if $\text{len}_6 + \text{bit}_6 &gt; 32$), otherwise cleared.</td>
</tr>
<tr>
<td>SS</td>
<td>Cleared</td>
</tr>
</tbody>
</table>
Shifter Operations

Rn = Rn OR FDEP Rx BY Ry
Rn = Rn OR FDEP Rx BY <bit6>:<len6>

Function

Deposits a field from register Rx to register Rn.

The field value is logically ORed bitwise with the specified field of register Rn and the new value is written back to register Rn.

The input field is right-aligned within the fixed-point field of Rx. Its length is determined by the len field in register Ry or by the immediate len field in the instruction.

The field is deposited in the fixed-point field of Rn, starting from a bit position determined by the bit6 field in register Ry or by the immediate bit6 field in the instruction.

Bit6 and len6 can take values between 0 and 63 inclusive, allowing for deposit of fields ranging in length from 0 to 32 bits, and to bit positions ranging from 0 to off-scale left.

Example

\[
\begin{align*}
\begin{array}{cccccccc}
39 & 31 & 23 & 15 & 7 & 0 \\
|---------------|--------|\cdots|abcdefgh|ijkl\cdots|-----|
\end{array}
\quad Rx \\
\begin{array}{cccccccc}
|\cdots|----------|--------|-----|
\end{array}
\quad \text{len6 bits}
\end{align*}
\]

\[
\begin{align*}
\begin{array}{cccccccc}
39 & 31 & 23 & 15 & 7 & 0 \\
|abcdefghijklmnopqrstuvwxyz|abcdefghijklmn|-----|
\end{array}
\quad \text{Rn old}
\end{align*}
\]

\[
\begin{align*}
\begin{array}{cccccccc}
39 & 31 & 23 & 15 & 7 & 0 \\
|--------|--------|--------|
\end{array}
\quad \text{bit position bit6 (from reference point)}
\end{align*}
\]

\[
\begin{align*}
\begin{array}{cccccccc}
39 & 31 & 23 & 15 & 7 & 0 \\
|abcdefghijklmnopqrstuvwxyz|abcdefghijklmn|-----|
\end{array}
\quad \text{Rn new}
\end{align*}
\]

\[
\begin{align*}
\begin{array}{cccccccc}
39 & 31 & 23 & 15 & 7 & 0 \\
|abcd\cdots|rstuwxyz|zabc\cdots|
\end{array}
\quad \text{OR result}
\end{align*}
\]
## Compute Operation Reference

### Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SZ</td>
<td>Set if the output operand is 0, otherwise cleared.</td>
</tr>
<tr>
<td>SV</td>
<td>Set if any bits are deposited to the left of the 32-bit fixed-point output field (i.e., if len6 + bit6 &gt; 32), otherwise cleared.</td>
</tr>
<tr>
<td>SS</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
Shifter Operations

**Rn = FDEP Rx BY Ry (SE)**

**Rn = FDEP Rx BY <bit6>:<len6> (SE)**

**Function**

Deposits and sign-extends a field from register `Rx` to register `Rn`.

The input field is right-aligned within the fixed-point field of `Rx` (see **Figure B-2**). Its length is determined by the `len6` field in register `Ry` or by the immediate `len6` field in the instruction.

The field is deposited in the fixed-point field of `Rn`, starting from a bit position determined by the `bit6` field in register `Ry` or by the immediate `bit6` field in the instruction. The MSBs of `Rn` are sign-extended by the MSB of the deposited field, unless the MSB of the deposited field is off-scale left. Bits to the right of the deposited field are set to 0.

**Figure B-2. Field alignment**

The floating-point extension field of `Rn` (bits 7:0 of the 40-bit word) is set to all 0s. `Bit6` and `len6` can take values between 0 and 63 inclusive, allowing for deposit of fields ranging in length from 0 to 32 bits into bit positions ranging from 0 to off-scale left.
Compute Operation Reference

Example

\[
\begin{array}{cccccc}
39 & 31 & 23 & 15 & 7 & 0 \\
\hline
\text{aaaaaaaa} & \text{abcdef} & \text{ghijklmn} & \text{-----------} & \text{Rx} \\
& \text{len6 bits} & & & \\
39 & 31 & 23 & 15 & 7 & 0 \\
\hline
\text{aaaaaaaa} & \text{defghijk} & \text{lmn00000} & \text{00000000} & \text{00000000} & \text{Rn} \\
& \text{sign} & \text{extension} & \text{bit position bit6} & & \\
\end{array}
\]

(from reference point)

Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SZ</td>
<td>Set if the output operand is 0, otherwise cleared.</td>
</tr>
<tr>
<td>SV</td>
<td>Set if any bits are deposited to the left of the 32-bit fixed-point output field (i.e., if len6 + bit6 &gt; 32), otherwise cleared.</td>
</tr>
<tr>
<td>SS</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
Shifter Operations

**Rn = Rn OR FDEP Rx BY Ry (SE)**

**Rn = Rn OR FDEP Rx BY <bit6>:<len6> (SE)**

**Function**

Deposits and sign-extends a field from register Rx to register Rn.

The sign-extended field value is logically ORed bitwise with the value of register Rn and the new value is written back to register Rn. The input field is right-aligned within the fixed-point field of Rx. Its length is determined by the len6 field in register Ry or by the immediate len6 field in the instruction.

The field is deposited in the fixed-point field of Rn, starting from a bit position determined by the bit6 field in register Ry or by the immediate bit6 field in the instruction. Bit6 and len6 can take values between 0 and 63 inclusive, allowing for deposit of fields ranging in length from 0 to 32 bits into bit positions ranging from 0 to off-scale left.

**Example**

```
  39  31  23  15   7   0
|--------|--------|--abcdef|ghijklmn|--------| Rx
                  \--------/---------------------/
                 len6 bits

  39  31  23  15   7   0
|aaaaaabc|defghijk|lmn00000|00000000|00000000|00000000|
 \--------/---------------------/
  sign
  extension bit position bit6
                   (from reference point)

  39  31  23  15   7   0
|abcdefghijklmnopqrstuvwx|yzabcdef|ghijklmn| Rn old
  39  31  23  15   7   0
|wxyzabc|defghijk|lmntuvwxy|yzabcdef|ghijklmn| Rn new
                 OR result
```
**Compute Operation Reference**

**Status Flags**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SZ</td>
<td>Set if the output operand is 0, otherwise cleared</td>
</tr>
<tr>
<td>SV</td>
<td>Set if any bits are deposited to the left of the 32-bit fixed-point output field (i.e., if ( \text{len6 + bit6 &gt; 32} )). otherwise cleared.</td>
</tr>
<tr>
<td>SS</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
Shifter Operations

\[ Rn = \text{FEXT} \; Rx \; \text{BY} \; Ry \]
\[ Rn = \text{FEXT} \; Rx \; \text{BY} \; \text{<bit6>:<len6>} \]

**Function**

Extracts a field from register \( Rx \) to register \( Rn \).

The output field is placed right-aligned in the fixed-point field of \( Rn \) (see Figure B-3). Its length is determined by the \( \text{len6} \) field in register \( Ry \) or by the immediate \( \text{len6} \) field in the instruction.

The field is extracted from the fixed-point field of \( Rx \) starting from a bit position determined by the \( \text{bit6} \) field in register \( Ry \) or by the immediate \( \text{bit6} \) field in the instruction.

Bits to the left of the extracted field are set to 0 in register \( Rn \). The floating-point extension field of \( Rn \) (bits 7:0 of the 40-bit word) is set to all 0s. \( \text{bit6} \) and \( \text{len6} \) can take values between 0 and 63 inclusive, allowing for extraction of fields ranging in length from 0 to 32 bits, and from bit positions ranging from 0 to off-scale left.

![Figure B-3. Field alignment](image)

\[ Rn \]

extracted bits placed in \( Rn \), starting at LSB of 32-bit field
Example

39  31  23  15  7  0  
|-----abc|defghijk|lmn-----|--------|--------|    Rx
\----------------/  
len6 bits  |  
bit position bit6  
(from reference point)

39  31  23  15  7  0  
|00000000|00000000|00abcdef|ghijklmn|00000000|    Rn

Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SZ</td>
<td>Set if the output operand is 0, otherwise cleared.</td>
</tr>
<tr>
<td>SV</td>
<td>Set if any bits are extracted from the left of the 32-bit fixed-point, input field (i.e., if len6 + bit6 &gt; 32), otherwise cleared.</td>
</tr>
<tr>
<td>SS</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
Shifter Operations

\[ \text{Rn} = \text{FEXT Rx BY Ry (SE)} \]
\[ \text{Rn} = \text{FEXT Rx BY <bit6>:<len6> (SE)} \]

**Function**

Extracts and sign-extends a field from register \( Rx \) to register \( Rn \).

The output field is placed right-aligned in the fixed-point field of \( Rn \). Its length is determined by the \( \text{len6} \) field in register \( Ry \) or by the immediate \( \text{len6} \) field in the instruction.

The field is extracted from the fixed-point field of \( Rx \) starting from a bit position determined by the \( \text{bit6} \) field in register \( Ry \) or by the immediate \( \text{bit6} \) field in the instruction.

The MSBs of \( Rn \) are sign-extended by the MSB of the extracted field, unless the MSB is extracted from off-scale left.

The floating-point extension field of \( Rn \) (bits 7:0 of the 40-bit word) is set to all 0s.

\( \text{bit6} \) and \( \text{len6} \) can take values between 0 and 63 inclusive, allowing for extraction of fields ranging in length from 0 to 32 bits and from bit positions ranging from 0 to off-scale left.

**Example**

\[
\begin{array}{ccccccc}
39 & 31 & 23 & 15 & 7 & 0 \\
\hline
\text{------abc|defghijk|lmn------|--------|--------|} \quad \text{Rx} \\
\hline
\text{----------/} \\
\text{len6 bits} \\
\text{bit position bit6} \\
\text{(from reference point)} \\
\hline
39 & 31 & 23 & 15 & 7 & 0 \\
\hline
\text{|aaaaaaaa|aaaaaaaa|aabc|ghijklmn|00000000|} \quad \text{Rn} \\
\hline
\text{----------/} \\
\text{sign extension}
\end{array}
\]
## Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SZ</td>
<td>Set if the output operand is 0, otherwise cleared.</td>
</tr>
<tr>
<td>SV</td>
<td>Set if any bits are extracted from the left of the 32-bit fixed-point input field (i.e., if len6 + bit6 &gt; 32), otherwise cleared.</td>
</tr>
<tr>
<td>SS</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
Shifter Operations

**Rn = EXP Rx**

Function

Extracts the exponent of the fixed-point operand in Rx. The exponent is placed in the **shf8** field in register Rn. The exponent is calculated as the two's complement of:

\[
\# \text{ leading sign bits in } Rx - 1
\]

Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SZ</td>
<td>Set if the extracted exponent is 0, otherwise cleared.</td>
</tr>
<tr>
<td>SV</td>
<td>Cleared.</td>
</tr>
<tr>
<td>SS</td>
<td>Set if the fixed-point operand in Rx is negative (bit 31 is a 1), otherwise cleared.</td>
</tr>
</tbody>
</table>
Rn = EXP Rx (EX)

Function

Extracts the exponent of the fixed-point operand in Rx, assuming that the operand is the result of an ALU operation. The exponent is placed in the shf8 field in register Rn. If the AV status bit is set, a value of +1 is placed in the shf8 field to indicate an extra bit (the ALU overflow bit). If the AV status bit is not set, the exponent is calculated as the two's complement of:

# leading sign bits in Rx − 1

Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SZ</td>
<td>Set if the extracted exponent is 0, otherwise cleared.</td>
</tr>
<tr>
<td>SV</td>
<td>Cleared.</td>
</tr>
<tr>
<td>SS</td>
<td>Set if the exclusive OR of the AV status bit and the sign bit (bit 31) of the fixed-point operand in Rx is equal to 1, otherwise cleared.</td>
</tr>
</tbody>
</table>
Shifter Operations

Rn = LEFTZ Rx

Function

Extracts the number of leading 0s from the fixed-point operand in Rx. The extracted number is placed in the bit6 field in Rn.

Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SZ</td>
<td>Set if the MSB of Rx is 1, otherwise cleared.</td>
</tr>
<tr>
<td>SV</td>
<td>Set if the result is 32, otherwise cleared.</td>
</tr>
<tr>
<td>SS</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
Rn = LEFTO Rx

Function

Extracts the number of leading 1s from the fixed-point operand in Rx. The extracted number is placed in the bit6 field in Rn.

Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SZ</td>
<td>Set if the MSB of Rx is 0, otherwise cleared.</td>
</tr>
<tr>
<td>SV</td>
<td>Set if the result is 32, otherwise cleared.</td>
</tr>
<tr>
<td>SS</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
Shifter Operations

Rn = FPACK Fx

Function

Converts the IEEE 32-bit floating-point value in Fx to a 16-bit floating-point value stored in Rn. The short float data format has an 11-bit mantissa with a four-bit exponent and a sign bit. The 16-bit floating-point numbers reside in the lower 16 bits of the 32-bit floating-point field.

Table B-7 shows the result of the FPACK operation.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>135 &lt; exp</td>
<td>Largest magnitude representation.</td>
</tr>
<tr>
<td>120 &lt; exp ≤ 135</td>
<td>Exponent is MSB of source exponent concatenated with the three LSBs of source exponent. The packed fraction is the rounded upper 11 bits of the source fraction.</td>
</tr>
<tr>
<td>109 &lt; exp ≤ 120</td>
<td>Exponent=0. Packed fraction is the upper bits (source exponent - 110) of the source fraction prefixed by zeros (0s) and the “hidden” 1. The packed fraction is rounded.</td>
</tr>
<tr>
<td>exp &lt; 110</td>
<td>Packed word is all zeros (0s).</td>
</tr>
</tbody>
</table>

1 exp = source exponent sign bit remains the same in all cases

The short float type supports gradual underflow. This method sacrifices precision for dynamic range. When packing a number that would have underflowed, the exponent is set to zero (0) and the mantissa (including “hidden” 1) is right-shifted the appropriate amount. The packed result is a denormal, which can be unpacked into a normal IEEE floating-point number.
# Compute Operation Reference

## Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SZ</td>
<td>Cleared.</td>
</tr>
<tr>
<td>SV</td>
<td>Set if overflow occurs, cleared otherwise.</td>
</tr>
<tr>
<td>SS</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
Shifter Operations

\textbf{Fn = FUNPACK Rx}

\textbf{Function}

Converts the 16-bit floating-point value in $Rx$ to an IEEE 32-bit floating-point value stored in $Fx$.

Table B-8 shows the result of the FUNPACK operation.

Table B-8. FUNPACK Result

<table>
<thead>
<tr>
<th>Condition</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0 &lt; \text{exp}^1 \leq 15$</td>
<td>Exponent is the three LSBs of the source exponent prefixed by the MSB of the source exponent and four copies of the complement of the MSB. The unpacked fraction is the source fraction with 12 zeros appended.</td>
</tr>
<tr>
<td>$\text{exp} = 0$</td>
<td>Exponent is $(120 - N)$ where $N$ is the number of leading zeros in the source fraction. The unpacked fraction is the remainder of the source fraction with zeros appended to pad it and the “hidden” 1 is stripped away.</td>
</tr>
</tbody>
</table>

\(^1\) $\text{exp} = \text{source exponent sign bit remains the same in all cases}$

The short float type supports gradual underflow. This method sacrifices precision for dynamic range. When packing a number that would have underflowed, the exponent is set to 0 and the mantissa (including “hidden” 1) is right-shifted the appropriate amount. The packed result is a denormal, which can be unpacked into a normal IEEE floating-point number.
## Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SZ</td>
<td>Cleared.</td>
</tr>
<tr>
<td>SV</td>
<td>Cleared.</td>
</tr>
<tr>
<td>SS</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
Multifunction Computations

Each of the three types of multifunction computations,

- Dual add/subtract
- Parallel Multiplier/ALU
- Parallel Multiplier and add/subtract

has a different format for the 23-bit compute field.


Each of the four input operands for multifunction computations are constrained to a different set of four Register File locations, as shown in Figure B-4 on page B-95. For example, the X-input to the ALU must be R8, R9, R10 or R11. In all other compute operations, the input operands can be any Register File location.
Figure B-4. Valid input registers for multifunction computations
Multifunction Computations

Dual Add/Subtract (Fixed-Pt.)

The dual add/subtract operation computes the sum and the difference of two inputs and returns the two results to different registers. This operation has fixed-point and floating-point versions.

Syntax (fixed point version)

Ra = Rx + Ry, Rs = Rx - Ry

Compute Field

| 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0  | 00 | 0111 | RS | RA | RX | RY |

Function

Does a dual add/subtract of the fixed-point fields in registers Rx and Ry. The sum is placed in the fixed-point field of register Ra and the difference in the fixed-point field of Rs. The floating-point extension fields of Ra and Rs are set to all 0s. In saturation mode (the ALU saturation mode bit in MODE1 set) positive overflows return the maximum positive number (0x7FFF FFFF), and negative overflows return the minimum negative number (0x8000 0000).

Status Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if either of the fixed-point outputs is all 0s, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>
## Compute Operation Reference

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AN</td>
<td>Set if the most significant output bit is 1 of either of the outputs, otherwise cleared.</td>
</tr>
<tr>
<td>AV</td>
<td>Set if the XOR of the carries of the two most significant adder stages of either of the outputs is 1, otherwise cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Set if the carry from the most significant adder stage of either of the outputs is 1, otherwise cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Cleared.</td>
</tr>
</tbody>
</table>


**Multifunction Computations**

**Dual Add/Subtract (Floating-Pt.)**

**Syntax (floating point version)**

\[
Fa = Fx + Fy, \quad Fs = Fx - Fy
\]

**Compute Field**

| 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0  | 00 | 1111 | FS | FA | FX | FY |

**Function**

Does a dual add/subtract of the floating-point operands in registers \( Fx \) and \( Fy \). The normalized results are placed in registers \( Fa \) and \( Fs \): the sum in \( Fa \) and the difference in \( Fs \). Rounding is to nearest (IEEE) or by truncation, to a 32-bit or to a 40-bit boundary, as defined by the rounding mode and rounding boundary bits in MODE1. Postrounded overflow returns \( \pm \)Infinity (round-to-nearest) or \( \pm \)NORM.MAX (round-to-zero). Postrounded denormal returns \( \pm \)Zero. Denormal inputs are flushed to \( \pm \)Zero. A NAN input returns an all 1s result.

**Status Flags**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Set if either postrounded result is a denormal (unbiased exponent (&lt; -126)) or 0, otherwise cleared.</td>
</tr>
<tr>
<td>AU</td>
<td>Set if either postrounded result is a denormal, otherwise cleared.</td>
</tr>
<tr>
<td>AN</td>
<td>Set if either of the floating-point results is negative, otherwise cleared.</td>
</tr>
</tbody>
</table>
## Compute Operation Reference

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AV</td>
<td>Set if either of the postrounded results overflows (unbiased exponent &gt; +127), otherwise cleared.</td>
</tr>
<tr>
<td>AC</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AS</td>
<td>Cleared.</td>
</tr>
<tr>
<td>AI</td>
<td>Set if either of the input operands is a NaN, or if both of the input operands are Infinities, otherwise cleared.</td>
</tr>
</tbody>
</table>
Multifunction Computations

Parallel Multiplier and ALU (Fixed-Pt.)

The parallel Multiplier/ALU operation performs a multiply or multiply/accumulate and one of the following ALU operations—add, subtract, average, fixed-point to floating-point conversion, or floating-point to fixed-point conversion—and floating-point ABS, MIN, or MAX.

For detailed information on a particular operation, see “Single-Function Operations” on page B-2.

Syntax

See Table B-10

Compute Field

| 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 1  | OPCODE | RM | RA | RXM | RYM | RXA | RYA |


Parallel Multiplier & ALU (Floating-Point)

Syntax

See Table B-10

Compute Field

| 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 1  | OPCODE | FM | FA | FXM | FYM | FXA | FYA |

The Multiplier and ALU operations are determined by OPCODE. The selections for the 6-bit OPCODE field are listed in Table B-10. The Multiplier x- and y-operands are received from data registers RXM (FXM) and RYM (FYM). The Multiplier result operand is returned to data register RM (FM). The ALU x- and y-operands are received from data registers RXA (FXA) and RYA (FYA). The ALU result operand is returned to data register RA (FA).

The result operands can be returned to any registers within the Register File. Each of the four input operands is restricted to a particular set of four data registers.

Table B-9. Valid sources of the input operands

<table>
<thead>
<tr>
<th>Input</th>
<th>Allowed Sources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplier X:</td>
<td>R3-R0 (F3-F0)</td>
</tr>
<tr>
<td>Multiplier Y:</td>
<td>R7-R4 (F7-F4)</td>
</tr>
<tr>
<td>ALU X:</td>
<td>R11-R8 (F11-F8)</td>
</tr>
<tr>
<td>ALU Y:</td>
<td>R15-R12 (F15-F12)</td>
</tr>
</tbody>
</table>
## Multifunction Computations

Table B-10 provides the syntax and opcode for each of the parallel Multiplier and ALU instructions for both fixed point and floating point versions.

### Table B-10. Parallel Multiplier/ALU Computations

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_m = R_{3-0} \times R_{7-4} (SSFR)$, $R_a = R_{11-8} + R_{15-12}$</td>
<td>000100</td>
</tr>
<tr>
<td>$R_m = R_{3-0} \times R_{7-4} (SSFR)$, $R_a = R_{11-8} - R_{15-12}$</td>
<td>000101</td>
</tr>
<tr>
<td>$R_m = R_{3-0} \times R_{7-4} (SSFR)$, $R_a = (R_{11-8} + R_{15-12})/2$</td>
<td>000110</td>
</tr>
<tr>
<td>$MRF = MRF + R_{3-0} \times R_{7-4} (SSF)$, $R_a = R_{11-8} + R_{15-12}$</td>
<td>001000</td>
</tr>
<tr>
<td>$MRF = MRF + R_{3-0} \times R_{7-4} (SSF)$, $R_a = R_{11-8} - R_{15-12}$</td>
<td>001001</td>
</tr>
<tr>
<td>$MRF = MRF + R_{3-0} \times R_{7-4} (SSF)$, $R_a = (R_{11-8} + R_{15-12})/2$</td>
<td>001010</td>
</tr>
<tr>
<td>$R_m = MRF + R_{3-0} \times R_{7-4} (SSFR)$, $R_a = R_{11-8} + R_{15-12}$</td>
<td>001100</td>
</tr>
<tr>
<td>$R_m = MRF + R_{3-0} \times R_{7-4} (SSFR)$, $R_a = R_{11-8} - R_{15-12}$</td>
<td>001101</td>
</tr>
<tr>
<td>$R_m = MRF + R_{3-0} \times R_{7-4} (SSFR)$, $R_a = (R_{11-8} + R_{15-12})/2$</td>
<td>001110</td>
</tr>
<tr>
<td>$MRF = MRF - R_{3-0} \times R_{7-4} (SSF)$, $R_a = R_{11-8} + R_{15-12}$</td>
<td>010000</td>
</tr>
<tr>
<td>$MRF = MRF - R_{3-0} \times R_{7-4} (SSF)$, $R_a = R_{11-8} - R_{15-12}$</td>
<td>010001</td>
</tr>
<tr>
<td>$MRF = MRF - R_{3-0} \times R_{7-4} (SSF)$, $R_a = (R_{11-8} + R_{15-12})/2$</td>
<td>010010</td>
</tr>
<tr>
<td>$R_m = MRF - R_{3-0} \times R_{7-4} (SSFR)$, $R_a = R_{11-8} + R_{15-12}$</td>
<td>010100</td>
</tr>
</tbody>
</table>
## Compute Operation Reference

Table B-10. Parallel Multiplier/ALU Computations (Cont’d)

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R_m = MRF - R_{30} \times R_{74} ) (SSFR), ( R_a = R_{118} - R_{1512} )</td>
<td>010101</td>
</tr>
<tr>
<td>( R_m = MRF - R_{30} \times R_{74} ) (SSFR), ( R_a = (R_{118} + R_{1512})/2 )</td>
<td>010110</td>
</tr>
<tr>
<td>( F_m = F_{30} \times F_{74} ), ( F_a = F_{118} + F_{1512} )</td>
<td>011000</td>
</tr>
<tr>
<td>( F_m = F_{30} \times F_{74} ), ( F_a = F_{118} - F_{1512} )</td>
<td>011001</td>
</tr>
<tr>
<td>( F_m = F_{30} \times F_{74} ), ( F_a = \text{FLOAT } R_{118} \text{ by } R_{1512} )</td>
<td>011010</td>
</tr>
<tr>
<td>( F_m = F_{30} \times F_{74} ), ( F_a = \text{FIX } F_{118} \text{ by } R_{1512} )</td>
<td>011011</td>
</tr>
<tr>
<td>( F_m = F_{30} \times F_{74} ), ( F_a = \text{ABS } F_{118} )</td>
<td>011101</td>
</tr>
<tr>
<td>( F_m = F_{30} \times F_{74} ), ( F_a = \text{MAX } (F_{118}, F_{1512}) )</td>
<td>011110</td>
</tr>
<tr>
<td>( F_m = F_{30} \times F_{74} ), ( F_a = \text{MIN } (F_{118}, F_{1512}) )</td>
<td>011111</td>
</tr>
</tbody>
</table>
Multifunction Computations

Parallel Multiplier and Dual Add/Subtract

The parallel Multiplier and dual add/subtract operation performs a multiply or multiply/accumulate and computes the sum and the difference of the ALU inputs. For detailed information on the Multiplier operations, see the individual descriptions under “Multiplier Operations” on page B-50. For information on the dual add/subtract operation, see the individual Dual Add/Subtract operations. This operation has fixed-point and floating-point versions.

Syntax (Fixed-point versions)

Rm=R3-0 * R7-4 (SSFR), Ra=R11-8 + R15-12, Rs=R11-8 - R15-12

Compute Field

<table>
<thead>
<tr>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RS</td>
<td>RM</td>
<td>RA</td>
<td>RXM</td>
<td>RYM</td>
<td>RXA</td>
<td>RYA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Syntax (Floating-point versions)

Fm=F3-0 * F7-4, Fa=F11-8 + F15-12, Fs=F11-8 - F15-12

Compute Field

<table>
<thead>
<tr>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FS</td>
<td>FM</td>
<td>FA</td>
<td>FXM</td>
<td>FYM</td>
<td>FXA</td>
<td>FYA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The Multiplier x- and y-operands are received from data registers RXM (FXM) and RYM (FYM). The Multiplier result operand is returned to data register RM (FM). The ALU x- and y-operands are received from data registers RXA (FXA) and RYA (FYA). The ALU result operands are returned to data register RA (FA) and RS (FS).
The result operands can be returned to any registers within the Register File. Each of the four input operands is restricted to a different set of four data registers, as shown in Table B-11.

Table B-11. Valid sources of the input operands

<table>
<thead>
<tr>
<th>Input</th>
<th>Valid Sources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplier X:</td>
<td>R3-R0 (F3-F0)</td>
</tr>
<tr>
<td>Multiplier Y:</td>
<td>R7-R4 (F7-F4)</td>
</tr>
<tr>
<td>ALU X:</td>
<td>R11-R8 (F11-F8)</td>
</tr>
<tr>
<td>ALU Y:</td>
<td>R15-R12 (F15-F12)</td>
</tr>
</tbody>
</table>
Multifunction Computations
C NUMERIC FORMATS

The processor supports several numeric formats:


- An extended-precision version of the 32-bit, single-precision floating-point format that has eight additional bits in the mantissa (40 bits total).

- 32-bit, fixed-point formats that include both fractions and integers in signed (twos-complement) or unsigned formats.
IEEE Standard 754/854 specifies a 32-bit, single-precision floating-point format, as shown in Figure C-1. A number in this format consists of a sign bit $s$, a 24-bit significant, and an 8-bit unsigned-magnitude exponent $e$.

![IEEE 32-bit single-precision floating-point format](image)

Figure C-1. IEEE 32-bit single-precision floating-point format

For normalized numbers, the significant consists of a 23-bit fraction $f$ and a hidden bit 1 understood to precede $f_{22}$ in the significant. The binary point is understood to lie between the hidden bit and $f_{22}$. The least significant bit (LSB) of the fraction is $f_0$. The LSB of the exponent is $e_0$.

The hidden bit effectively increases the precision of the floating-point significant to twenty-four bits from the twenty-three bits actually stored in the data format. It also ensures that the significant of any IEEE normalized number is always $\geq 1$ and $<2$.

In the single-precision format, the unsigned exponent $e$ ranges between $1 \leq e \leq 254$ for normal numbers. This exponent is biased by +127 ($254 + 2$). To calculate the true unbiased exponent, you subtract 127 from $e$. 
The IEEE standard also provides for several special data types in the single-precision floating-point format, as shown in Table C-1.

Table C-1. Supported single-precision, floating-point special data types

<table>
<thead>
<tr>
<th>Type</th>
<th>Exponent</th>
<th>Fraction</th>
<th>Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infinity</td>
<td>255</td>
<td>0</td>
<td>((-1)^s \text{Infinity})</td>
<td>Because the fraction is signed, can represent (\pm \text{Infinity})</td>
</tr>
<tr>
<td>Normal</td>
<td>(1 \leq e \leq 254)</td>
<td>any</td>
<td>((-1)^s (1.f_{22-0})2^{e-127})</td>
<td></td>
</tr>
<tr>
<td>Zero</td>
<td>0</td>
<td>0</td>
<td>((-1)^s \text{Zero})</td>
<td>Represents (\pm \text{Zero})</td>
</tr>
<tr>
<td>NAN (Not-A-Number)</td>
<td>255 (all 1s)</td>
<td>nonzero</td>
<td>undefined</td>
<td>Typical uses are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Flags for data flow control</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Values of uninitialized variables</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Results of invalid operations (as 0 * (\infty))</td>
</tr>
</tbody>
</table>
Extended-Precision Floating-Point Format

The extended-precision floating-point format, as shown in Figure C-2, is the same as the single-precision format, except this format:

- Is forty bits wide
- Has a 32-bit significant

![40-bit extended floating-point format](image)

Figure C-2. 40-bit extended floating-point format
Short Word Floating-Point Format

The processor supports a 16-bit, floating-point data type and provides conversion instructions for it.

This format has an 11-bit mantissa, a 4-bit exponent, and a sign bit, as shown in Figure C-3. The 16-bit floating-point numbers reside in the lower sixteen bits of the 32-bit floating-point field.

![Figure C-3. 16-bit floating-point format](image)

Two shifter instructions, FPACK and FUNPACK, perform the packing and unpacking conversions between 32- and 16-bit floating-point words.

The FPACK instruction converts a 32-bit IEEE floating-point number to a 16-bit floating-point number.

The FUNPACK instruction converts a 16-bit floating-point number to a 32-bit IEEE floating-point number.
Short Word Floating-Point Format

Each instruction executes in a single cycle. Table C-2 lists and describes the results of the FPACK and FUNPACK operations.

Table C-2. Results of the FPACK and FUNPACK operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Condition</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPACK</td>
<td>135 &lt; exp</td>
<td>Largest magnitude representation</td>
</tr>
<tr>
<td></td>
<td>120 &lt; exp ≤ 135</td>
<td>Exponent is MSB of source exponent concatenated with the three LSBs of source exponent. The packed fraction is the rounded upper 11 bits of the source fraction.</td>
</tr>
<tr>
<td></td>
<td>109 &lt; exp ≤ 120</td>
<td>Exponent=0. Packed fraction is the upper bits (source exponent -110) of the source fraction prefixed by zeros and the “hidden” 1. The packed fraction is rounded.</td>
</tr>
<tr>
<td></td>
<td>exp &lt; 110</td>
<td>Packed word is all zeros (0s).</td>
</tr>
</tbody>
</table>

exp = source exponent; sign bit remains the same in all cases
The short float type supports gradual underflow, which sacrifices precision for dynamic range. When packing a number that would have underflowed, the processor sets the exponent to zero (0) and right shifts the mantissa (including the hidden 1) the appropriate amount. The packed result is a denormal, which you can unpack into a normal IEEE floating-point number.

During the FPACK operation, an overflow condition sets the SV flag, and a nonoverflow condition clears it. During the FUNPACK operation, the Shifter clears the SV flag. For both instructions, the Shifter clears the SZ and SS flags. For details, see Chapter 2, Computation Units, in ADSP-21065L SHARC DSP User’s Manual.
Fixed-Point Formats

The processor supports two 32-bit fixed-point formats—fractional and integer—both of which include signed (twos-complement) and unsigned numbers. Figure C-4 shows the four possible combinations.

Figure C-4. 32-bit fixed-point formats

The fractional format includes an implied binary point to the left of the most significant magnitude bit. The integer format includes an implied binary point to the right of the LSB. In signed (twos-complement) format, the sign bit is negatively weighted.
ALU outputs always have the same width and data format as the inputs.

The Multiplier, however, produces a 64-bit product from two 32-bit inputs. Multiplier results follow these rules:

- If both operands are unsigned integers, the result is a 64-bit unsigned integer.
- If both operands are unsigned fractions, the result is a 64-bit unsigned fraction.

Figure C-5 shows both of these results.

![Unsigned Integer](image)

![Unsigned Fractional](image)

- If one operand is signed and the other is unsigned, the result is signed.
- If both inputs are signed, the result is signed and automatically shifted left one bit.

The LSB becomes zero (0) and bit 62 moves into the sign bit position.

Normally bit 63 and bit 62 are identical when both operands are signed. (The only exception occurs when a full-scale negative is multiplied by itself.) So, the left shift normally removes a redundant
Fixed-Point Formats

sign bit, increasing the precision of the most significant product (MSP).

If the data format is fractional, a single-bit left shift renormalizes the MSP to a fractional format. Figure C-6 shows the signed formats with and without left shifting.

Figure C-6. 64-bit signed, fixed-point product

The Multiplier has an 80-bit accumulator for accumulating 64-bit products. For details, see Chapter 2, Computation Units, in ADSP-21065L SHARC DSP User's Manual.
A boundary scan enables a system designer, with minimal test-specific hardware, to test interconnections on a printed circuit board.

The ability to control and monitor each input and output pin on each chip through a set of serially scannable latches makes the scan possible. Each input and output is connected to a latch, and each latch is connected as a long shift register, so a test program can read and write data from or to the latches through a serial test access port (TAP).

The processor contains a test access port that is compatible with the industry-standard IEEE 1149.1 (JTAG) specification.

This appendix describes the IEEE 1149.1 features specific to the ADSP-21065L. For more information, see the IEEE 1149.1 specification and other references listed at the end of this appendix.

The boundary scan supports a variety of functions for testing each input and output signal of the ADSP-21065L. Each input has a latch that can either monitor the value of an incoming signal or drive data into the chip. Similarly, each output has a latch that can either monitor the value of an outgoing signal or drive the output. For bidirectional pins, you can combine input and output functions.

Each latch associated with a pin is part of a single, serial-shift register path. Each latch is a master/slave type latch, with the controlling clock provided externally. This clock (TCK) is asynchronous to the ADSP-21065L’s system clock (CLKIN).
Test Access Port (TAP)

The test access port (TAP) controls the operation of the boundary scan. The TAP consists of five pins that control a state machine, including the boundary scan. The state machine and pins conform to the IEEE 1149.1 specification.

**TCK** (input)
Test Clock.

Used to clock serial data into scan latches and control sequencing of the test state machine. TCK can be asynchronous with CLkin.

**TMS** (input)
Test Mode Select.

Primary control signal for the state machine. Synchronous with TCK. A sequence of values on TMS adjusts the current state of the TAP.

**TDI** (input)
Test Data Input.

Serial input data to the scan latches. Synchronous with TCK.

**TDO** (output)
Test Data Output.

Serial output data from the scan latches. Synchronous with TCK.

**TRST** (input)
Test Reset.

Resets the test state machine. Can be asynchronous with TCK.
A BSDL (Boundary Scan Description Language) file for the ADSP-21065L is available on Analog Devices’ web site.

### Instruction Register

The instruction register enables the processor to shift in an instruction. This instruction selects the test to perform and/or the test data register to access. The instruction register is five-bits long with no parity bit. The processor loads a binary value of 10000 (LSB nearest TDI) into the instruction register whenever the TAP reset state is entered.

Table D-1 lists the binary code for each instruction. Bit 0 is nearest TDI and bit 4 is nearest TDO. An “x” specifies a “don’t-care” state. None of the public instructions place data registers into test modes. The instructions affect the ADSP-21065L as defined in the 1149.1 specification. The ADSP-21065L does not support the optional instructions RUNBIST, IDCODE, or USERCODEL.

Table D-1. Test instructions

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Register (Serial Path)</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 x x x x</td>
<td>BYPASS</td>
<td>Bypass</td>
<td>Public</td>
</tr>
<tr>
<td>0 0 0 0 0</td>
<td>EXTEST</td>
<td>Boundary</td>
<td>Public</td>
</tr>
<tr>
<td>0 0 0 0 1</td>
<td>SAMPLE/PRELOAD</td>
<td>Boundary</td>
<td>Public</td>
</tr>
<tr>
<td>0 0 0 1 0</td>
<td>Reserved for emulation</td>
<td>NA</td>
<td>Private</td>
</tr>
<tr>
<td>0 0 0 1 1</td>
<td>INTEST</td>
<td>Boundary</td>
<td>Public</td>
</tr>
<tr>
<td>0 0 1 0 0</td>
<td>Reserved for emulation</td>
<td>NA</td>
<td>Private</td>
</tr>
<tr>
<td>0 0 1 0 1</td>
<td>Reserved for emulation</td>
<td>NA</td>
<td>Private</td>
</tr>
</tbody>
</table>
The entry under “Register” is the serial scan path, either Boundary or Bypass in this case, that the instruction enabled. Figure D-1 shows these register paths. The single-bit Bypass register is fully defined in the 1149.1 specification. The Boundary register is described in the next section.

You do not need to write special values into any register prior to selecting any instruction. As Table D-1 shows, certain instructions are reserved for the emulator. For details, see “Private Instructions” on page D-29.

### Table D-1. Test instructions

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Register (Serial Path)</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 1 1 0</td>
<td>Reserved for emulation</td>
<td>NA</td>
<td>Private</td>
</tr>
<tr>
<td>0 0 1 1 1</td>
<td>Reserved for emulation</td>
<td>NA</td>
<td>Private</td>
</tr>
<tr>
<td>0 1 x x x</td>
<td>Reserved for emulation</td>
<td>NA</td>
<td>Private</td>
</tr>
</tbody>
</table>
Figure D-1. Serial scan paths
Boundary Register

The Boundary register is 285 bits long.

Table D-2 lists and defines the latch type and function of each position in the scan path. The positions are numbered from 0 to 284. Bit 0 is the first bit output (closest to TDO) and bit 284 is the last bit output (closest to TDI).

Table D-2. Scan path position definitions

<table>
<thead>
<tr>
<th>Position</th>
<th>Latch Type</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>I</td>
<td>BSEL</td>
</tr>
<tr>
<td>1</td>
<td>O</td>
<td>BMS</td>
</tr>
<tr>
<td>2</td>
<td>I</td>
<td>BMS</td>
</tr>
<tr>
<td>3</td>
<td>I</td>
<td>Reserved1</td>
</tr>
<tr>
<td>4</td>
<td>OE</td>
<td>BMS output enable</td>
</tr>
<tr>
<td>5</td>
<td>I</td>
<td>RESET</td>
</tr>
<tr>
<td>6</td>
<td>O</td>
<td>ADDR23</td>
</tr>
<tr>
<td>7</td>
<td>I</td>
<td>ADDR23</td>
</tr>
</tbody>
</table>

I = Input  
O = Output  
OE = OutputEnable
1 = Drive the associated signals during EXTEST and INTEST instructions.  
0 = Disable the associated signals during EXTEST and INTEST instructions.  
NC = Do not connect
Table D-2. Scan path position definitions (Cont’d)

<table>
<thead>
<tr>
<th>Position</th>
<th>Latch Type</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0</td>
<td>ADDR_{22}</td>
</tr>
<tr>
<td>9</td>
<td>I</td>
<td>ADDR_{22}</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>ADDR_{21}</td>
</tr>
<tr>
<td>11</td>
<td>I</td>
<td>ADDR_{21}</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>ADDR_{20}</td>
</tr>
<tr>
<td>13</td>
<td>I</td>
<td>ADDR_{20}</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>ADDR_{19}</td>
</tr>
<tr>
<td>15</td>
<td>I</td>
<td>ADDR_{19}</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>ADDR_{18}</td>
</tr>
<tr>
<td>17</td>
<td>I</td>
<td>ADDR_{18}</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
<td>ADDR_{17}</td>
</tr>
<tr>
<td>19</td>
<td>I</td>
<td>ADDR_{17}</td>
</tr>
</tbody>
</table>

I = Input  
0 = Output  
OE = OutputEnable  
1 = Drive the associated signals during EXTEST and INTEST instructions,  
0 = Disable the associated signals during EXTEST and INTEST instructions)  
NC = Do not connect
### Boundary Register

Table D-2. Scan path position definitions (Cont’d)

<table>
<thead>
<tr>
<th>Position</th>
<th>Latch Type</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>0</td>
<td>ADDR(_{16})</td>
</tr>
<tr>
<td>21</td>
<td>I</td>
<td>ADDR(_{16})</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
<td>ADDR(_{15})</td>
</tr>
<tr>
<td>23</td>
<td>I</td>
<td>ADDR(_{15})</td>
</tr>
<tr>
<td>24</td>
<td>0</td>
<td>ADDR(_{14})</td>
</tr>
<tr>
<td>25</td>
<td>I</td>
<td>ADDR(_{14})</td>
</tr>
<tr>
<td>26</td>
<td>0</td>
<td>ADDR(_{13})</td>
</tr>
<tr>
<td>27</td>
<td>I</td>
<td>ADDR(_{13})</td>
</tr>
<tr>
<td>28</td>
<td>0</td>
<td>ADDR(_{12})</td>
</tr>
<tr>
<td>29</td>
<td>I</td>
<td>ADDR(_{12})</td>
</tr>
<tr>
<td>30</td>
<td>0</td>
<td>ADDR(_{11})</td>
</tr>
<tr>
<td>31</td>
<td>I</td>
<td>ADDR(_{11})</td>
</tr>
</tbody>
</table>

I= Input  
0= Output  
OE= OutputEnable  
1 = Drive the associated signals during EXTEST and INTEST instructions,  
0 = Disable the associated signals during EXTEST and INTEST instructions)  
NC= Do not connect
### JTAG Test Access Port

Table D-2. Scan path position definitions  (Cont’d)

<table>
<thead>
<tr>
<th>Position</th>
<th>Latch Type</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>0</td>
<td>ADDR_{10}</td>
</tr>
<tr>
<td>33</td>
<td>I</td>
<td>ADDR_{10}</td>
</tr>
<tr>
<td>34</td>
<td>0</td>
<td>ADDR_{9}</td>
</tr>
<tr>
<td>35</td>
<td>I</td>
<td>ADDR_{9}</td>
</tr>
<tr>
<td>36</td>
<td>OE</td>
<td>ADDR output enable</td>
</tr>
<tr>
<td>37</td>
<td>0</td>
<td>ADDR_{8}</td>
</tr>
<tr>
<td>38</td>
<td>I</td>
<td>ADDR_{8}</td>
</tr>
<tr>
<td>39</td>
<td>0</td>
<td>ADDR_{7}</td>
</tr>
<tr>
<td>40</td>
<td>I</td>
<td>ADDR_{7}</td>
</tr>
<tr>
<td>41</td>
<td>0</td>
<td>ADDR_{6}</td>
</tr>
<tr>
<td>42</td>
<td>I</td>
<td>ADDR_{6}</td>
</tr>
<tr>
<td>43</td>
<td>0</td>
<td>ADDR_{5}</td>
</tr>
<tr>
<td>44</td>
<td>I</td>
<td>ADDR_{5}</td>
</tr>
</tbody>
</table>

I= Input  
0= Output  
OE= OutputEnable  
1 = Drive the associated signals during EXTEST and INTEST instructions,  
0 = Disable the associated signals during EXTEST and INTEST instructions)  
NC= Do not connect
### Boundary Register

Table D-2. Scan path position definitions (Cont’d)

<table>
<thead>
<tr>
<th>Position</th>
<th>Latch Type</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>0</td>
<td>ADDR₄</td>
</tr>
<tr>
<td>46</td>
<td>I</td>
<td>ADDR₄</td>
</tr>
<tr>
<td>47</td>
<td>0</td>
<td>ADDR₃</td>
</tr>
<tr>
<td>48</td>
<td>I</td>
<td>ADDR₃</td>
</tr>
<tr>
<td>49</td>
<td>0</td>
<td>ADDR₂</td>
</tr>
<tr>
<td>50</td>
<td>I</td>
<td>ADDR₂</td>
</tr>
<tr>
<td>51</td>
<td>0</td>
<td>ADDR₁</td>
</tr>
<tr>
<td>52</td>
<td>I</td>
<td>ADDR₁</td>
</tr>
<tr>
<td>53</td>
<td>0</td>
<td>ADDR₀</td>
</tr>
<tr>
<td>54</td>
<td>I</td>
<td>ADDR₀</td>
</tr>
<tr>
<td>55</td>
<td>OE</td>
<td>FLAG₀ output enable</td>
</tr>
<tr>
<td>56</td>
<td>OE</td>
<td>FLAG₁ output enable</td>
</tr>
<tr>
<td>57</td>
<td>0</td>
<td>FLAG₀</td>
</tr>
</tbody>
</table>

I= Input  
O= Output  
OE= OutputEnable  
1 = Drive the associated signals during EXTEST and INTEST instructions,  
0 = Disable the associated signals during EXTEST and INTEST instructions)  
NC= Do not connect
### Table D-2. Scan path position definitions (Cont’d)

<table>
<thead>
<tr>
<th>Position</th>
<th>Latch Type</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>58</td>
<td>I</td>
<td>FLAG0</td>
</tr>
<tr>
<td>59</td>
<td>O</td>
<td>FLAG1</td>
</tr>
<tr>
<td>60</td>
<td>I</td>
<td>FLAG1</td>
</tr>
<tr>
<td>61</td>
<td>O</td>
<td>FLAG2</td>
</tr>
<tr>
<td>62</td>
<td>I</td>
<td>FLAG2</td>
</tr>
<tr>
<td>63</td>
<td>OE</td>
<td>FLAG2 output enable</td>
</tr>
<tr>
<td>64</td>
<td>OE</td>
<td>FLAG3 output enable</td>
</tr>
<tr>
<td>65</td>
<td>O</td>
<td>FLAG3</td>
</tr>
<tr>
<td>66</td>
<td>I</td>
<td>FLAG3</td>
</tr>
<tr>
<td>67</td>
<td>OE</td>
<td>SPARE1 output enable</td>
</tr>
<tr>
<td>68</td>
<td>O</td>
<td>SPARE1</td>
</tr>
<tr>
<td>69</td>
<td>I</td>
<td>SPARE1</td>
</tr>
<tr>
<td>70</td>
<td>OE</td>
<td>SPARE0 output enable</td>
</tr>
</tbody>
</table>

**Legend:**

- **I** = Input
- **O** = Output
- **OE** = Output Enable

1 = Drive the associated signals during EXTEST and INTEST instructions,
0 = Disable the associated signals during EXTEST and INTEST instructions)

NC = Do not connect
Boundary Register

Table D-2. Scan path position definitions (Cont’d)

<table>
<thead>
<tr>
<th>Position</th>
<th>Latch Type</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>71</td>
<td>0</td>
<td>SPARE0</td>
</tr>
<tr>
<td>72</td>
<td>I</td>
<td>SPARE0</td>
</tr>
<tr>
<td>73</td>
<td>I</td>
<td>TRQ0</td>
</tr>
<tr>
<td>74</td>
<td>I</td>
<td>TRQ1</td>
</tr>
<tr>
<td>75</td>
<td>I</td>
<td>TRQ2</td>
</tr>
<tr>
<td>76</td>
<td>OE</td>
<td>SPARE6 output enable</td>
</tr>
<tr>
<td>77</td>
<td>0</td>
<td>SPARE6</td>
</tr>
<tr>
<td>78</td>
<td>I</td>
<td>SPARE6</td>
</tr>
<tr>
<td>79</td>
<td>0</td>
<td>RFS0</td>
</tr>
<tr>
<td>80</td>
<td>I</td>
<td>RFS0</td>
</tr>
<tr>
<td>81</td>
<td>OE</td>
<td>RFS0 output enable</td>
</tr>
<tr>
<td>82</td>
<td>OE</td>
<td>RCLK0 output enable</td>
</tr>
<tr>
<td>83</td>
<td>OE</td>
<td>TFS0 output enable</td>
</tr>
</tbody>
</table>

I= Input  
0= Output  
OE= OutputEnable  
1 = Drive the associated signals during EXTEST and INTEST instructions,  
0 = Disable the associated signals during EXTEST and INTEST instructions)  
NC= Do not connect
### JTAG Test Access Port

Table D-2. Scan path position definitions (Cont'd)

<table>
<thead>
<tr>
<th>Position</th>
<th>Latch Type</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>84</td>
<td>0</td>
<td>RCLK0</td>
</tr>
<tr>
<td>85</td>
<td>I</td>
<td>RCLK0</td>
</tr>
<tr>
<td>86</td>
<td>I</td>
<td>DR0_A</td>
</tr>
<tr>
<td>87</td>
<td>I</td>
<td>DR0_B</td>
</tr>
<tr>
<td>88</td>
<td>0</td>
<td>TFS0</td>
</tr>
<tr>
<td>89</td>
<td>I</td>
<td>TFS0</td>
</tr>
<tr>
<td>90</td>
<td>0</td>
<td>TCLKO</td>
</tr>
<tr>
<td>91</td>
<td>I</td>
<td>TCLKO</td>
</tr>
<tr>
<td>92</td>
<td>OE</td>
<td>TCLKO output enable</td>
</tr>
<tr>
<td>93</td>
<td>OE</td>
<td>DTO_A output enable</td>
</tr>
<tr>
<td>94</td>
<td>OE</td>
<td>DTO_B output enable</td>
</tr>
<tr>
<td>95</td>
<td>0</td>
<td>DTO_A</td>
</tr>
<tr>
<td>96</td>
<td>0</td>
<td>DTO_B</td>
</tr>
</tbody>
</table>

- **I** = Input
- **O** = Output
- **OE** = OutputEnable
- 1 = Drive the associated signals during EXTEST and INTEST instructions.
- 0 = Disable the associated signals during EXTEST and INTEST instructions)
- **NC** = Do not connect
### Boundary Register

#### Table D-2. Scan path position definitions (Cont'd)

<table>
<thead>
<tr>
<th>Position</th>
<th>Latch Type</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>97</td>
<td>0</td>
<td>RFS1</td>
</tr>
<tr>
<td>98</td>
<td>I</td>
<td>RFS1</td>
</tr>
<tr>
<td>99</td>
<td>OE</td>
<td>RFS1 output enable</td>
</tr>
<tr>
<td>100</td>
<td>OE</td>
<td>RCLK1 output enable</td>
</tr>
<tr>
<td>101</td>
<td>OE</td>
<td>TFS1 output enable</td>
</tr>
<tr>
<td>102</td>
<td>0</td>
<td>RCLK1</td>
</tr>
<tr>
<td>103</td>
<td>I</td>
<td>RCLK1</td>
</tr>
<tr>
<td>104</td>
<td>I</td>
<td>DR1_A</td>
</tr>
<tr>
<td>105</td>
<td>I</td>
<td>DR1_B</td>
</tr>
<tr>
<td>106</td>
<td>0</td>
<td>TFS1</td>
</tr>
<tr>
<td>107</td>
<td>I</td>
<td>TFS1</td>
</tr>
<tr>
<td>108</td>
<td>0</td>
<td>TCLK1</td>
</tr>
<tr>
<td>109</td>
<td>I</td>
<td>TCLK1</td>
</tr>
</tbody>
</table>

**Legend:**

- **I** = Input
- **O** = Output
- **OE** = Output Enable
  - 1 = Drive the associated signals during EXTEST and INTEST instructions.
  - 0 = Disable the associated signals during EXTEST and INTEST instructions.
- **NC** = Do not connect
Table D-2. Scan path position definitions (Cont'd)

<table>
<thead>
<tr>
<th>Position</th>
<th>Latch Type</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>110</td>
<td>OE</td>
<td>TCLK1 output enable</td>
</tr>
<tr>
<td>111</td>
<td>OE</td>
<td>DT1_A output enable</td>
</tr>
<tr>
<td>112</td>
<td>OE</td>
<td>DT1_B output enable</td>
</tr>
<tr>
<td>113</td>
<td>O</td>
<td>DT1_A</td>
</tr>
<tr>
<td>114</td>
<td>O</td>
<td>DT1_B</td>
</tr>
<tr>
<td>115</td>
<td>O</td>
<td>PWM_EVENT0</td>
</tr>
<tr>
<td>116</td>
<td>I</td>
<td>PWM_EVENT1</td>
</tr>
<tr>
<td>117</td>
<td>OE</td>
<td>PWM_EVENT1 output enable</td>
</tr>
<tr>
<td>118</td>
<td>OE</td>
<td>PWM_EVENT0 output enable</td>
</tr>
<tr>
<td>119</td>
<td>OE</td>
<td>BR1 output enable</td>
</tr>
<tr>
<td>120</td>
<td>OE</td>
<td>BR2 output enable</td>
</tr>
<tr>
<td>121</td>
<td>O</td>
<td>PWM_EVENT0</td>
</tr>
<tr>
<td>122</td>
<td>I</td>
<td>PWM_EVENT0</td>
</tr>
</tbody>
</table>

I= Input
0= Output
OE= OutputEnable

1 = Drive the associated signals during EXTEST and INTEST instructions.
0 = Disable the associated signals during EXTEST and INTEST instructions)
NC= Do not connect
### Boundary Register

Table D-2. Scan path position definitions (Cont'd)

<table>
<thead>
<tr>
<th>Position</th>
<th>Latch Type</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>0</td>
<td>BR1</td>
</tr>
<tr>
<td>124</td>
<td>I</td>
<td>BR1</td>
</tr>
<tr>
<td>125</td>
<td>0</td>
<td>BR2</td>
</tr>
<tr>
<td>126</td>
<td>I</td>
<td>BR2</td>
</tr>
<tr>
<td>127</td>
<td>I</td>
<td>CLKIN</td>
</tr>
<tr>
<td>128</td>
<td>OE</td>
<td>SDCLK1 output enable</td>
</tr>
<tr>
<td>129</td>
<td>0</td>
<td>SDCLK1</td>
</tr>
<tr>
<td>130</td>
<td>I</td>
<td>SDCLK1</td>
</tr>
<tr>
<td>131</td>
<td>OE</td>
<td>SDCLK0, RAS, CAS, DQM, SDCKE, SDA10 output enable</td>
</tr>
<tr>
<td>132</td>
<td>0</td>
<td>SDCLK0</td>
</tr>
<tr>
<td>133</td>
<td>I</td>
<td>SDCLK0</td>
</tr>
<tr>
<td>134</td>
<td>I</td>
<td>DMAR1</td>
</tr>
<tr>
<td>135</td>
<td>I</td>
<td>DMAR2</td>
</tr>
</tbody>
</table>

I= Input  
0= Output  
OE= OutputEnable  
1 = Drive the associated signals during EXTEST and INTEST instructions.  
0 = Disable the associated signals during EXTEST and INTEST instructions)  
NC = Do not connect
### Table D-2. Scan path position definitions (Cont'd)

<table>
<thead>
<tr>
<th>Position</th>
<th>Latch Type</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>136</td>
<td>I</td>
<td>HBR</td>
</tr>
<tr>
<td>137</td>
<td>0</td>
<td>RAS</td>
</tr>
<tr>
<td>138</td>
<td>I</td>
<td>RAS</td>
</tr>
<tr>
<td>139</td>
<td>0</td>
<td>CAS</td>
</tr>
<tr>
<td>140</td>
<td>I</td>
<td>CAS</td>
</tr>
<tr>
<td>141</td>
<td>0</td>
<td>SDWE</td>
</tr>
<tr>
<td>142</td>
<td>I</td>
<td>SDWE</td>
</tr>
<tr>
<td>143</td>
<td>0</td>
<td>DMAG1</td>
</tr>
<tr>
<td>144</td>
<td>0</td>
<td>SDCKE</td>
</tr>
<tr>
<td>145</td>
<td>I</td>
<td>SDCKE</td>
</tr>
<tr>
<td>146</td>
<td>0</td>
<td>SDA10</td>
</tr>
<tr>
<td>147</td>
<td>OE</td>
<td>HBG output enable</td>
</tr>
<tr>
<td>148</td>
<td>0</td>
<td>DMAG1</td>
</tr>
</tbody>
</table>

I = Input  
0 = Output  
OE = OutputEnable  
1 = Drive the associated signals during EXTEST and INTEST instructions,  
0 = Disable the associated signals during EXTEST and INTEST instructions)  
NC = Do not connect
## Boundary Register

Table D-2. Scan path position definitions (Cont’d)

<table>
<thead>
<tr>
<th>Position</th>
<th>Latch Type</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>149</td>
<td>0</td>
<td>DMAG2</td>
</tr>
<tr>
<td>150</td>
<td>0</td>
<td>HBG</td>
</tr>
<tr>
<td>151</td>
<td>I</td>
<td>HBG</td>
</tr>
<tr>
<td>152</td>
<td>0</td>
<td>BMSTR</td>
</tr>
<tr>
<td>153</td>
<td>OE</td>
<td>RD, WR, DMAG1, DMAG2, MS, SW, output enable</td>
</tr>
<tr>
<td>154</td>
<td>I</td>
<td>CS</td>
</tr>
<tr>
<td>155</td>
<td>I</td>
<td>STBS</td>
</tr>
<tr>
<td>156</td>
<td>I</td>
<td>Reserved2</td>
</tr>
<tr>
<td>157</td>
<td>0</td>
<td>WR</td>
</tr>
<tr>
<td>158</td>
<td>I</td>
<td>WR</td>
</tr>
<tr>
<td>159</td>
<td>0</td>
<td>RO</td>
</tr>
<tr>
<td>160</td>
<td>I</td>
<td>RO</td>
</tr>
<tr>
<td>161</td>
<td>OE</td>
<td>REDY output enable</td>
</tr>
</tbody>
</table>

I= Input  
0= Output  
OE= OutputEnable  
1 = Drive the associated signals during EXTEST and INTEST instructions.  
0 = Disable the associated signals during EXTEST and INTEST instructions)  
NC= Do not connect
## Table D-2. Scan path position definitions (Cont’d)

<table>
<thead>
<tr>
<th>Position</th>
<th>Latch Type</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>162</td>
<td>0</td>
<td>REDY</td>
</tr>
<tr>
<td>163</td>
<td>0</td>
<td>SW</td>
</tr>
<tr>
<td>164</td>
<td>I</td>
<td>SW</td>
</tr>
<tr>
<td>165</td>
<td>0</td>
<td>CPA</td>
</tr>
<tr>
<td>166</td>
<td>I</td>
<td>CPA</td>
</tr>
<tr>
<td>167</td>
<td>OE</td>
<td>ACK output enable</td>
</tr>
<tr>
<td>168</td>
<td>I</td>
<td>Reserved3</td>
</tr>
<tr>
<td>169</td>
<td>0</td>
<td>ACK</td>
</tr>
<tr>
<td>170</td>
<td>I</td>
<td>ACK</td>
</tr>
<tr>
<td>171</td>
<td>0</td>
<td>MS0</td>
</tr>
<tr>
<td>172</td>
<td>I</td>
<td>MS0</td>
</tr>
<tr>
<td>173</td>
<td>0</td>
<td>MS1</td>
</tr>
<tr>
<td>174</td>
<td>I</td>
<td>MS1</td>
</tr>
</tbody>
</table>

**Notes:**
- **I** = Input
- **O** = Output
- **OE** = OutputEnable
  - 1 = Drive the associated signals during EXTEST and INTEST instructions.
  - 0 = Disable the associated signals during EXTEST and INTEST instructions)
- **NC** = Do not connect
## Boundary Register

Table D-2. Scan path position definitions (Cont’d)

<table>
<thead>
<tr>
<th>Position</th>
<th>Latch Type</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>175</td>
<td>0</td>
<td>M52</td>
</tr>
<tr>
<td>176</td>
<td>I</td>
<td>M52</td>
</tr>
<tr>
<td>177</td>
<td>0</td>
<td>M53</td>
</tr>
<tr>
<td>178</td>
<td>I</td>
<td>M53</td>
</tr>
<tr>
<td>179</td>
<td>0</td>
<td>FLAG11</td>
</tr>
<tr>
<td>180</td>
<td>I</td>
<td>FLAG11</td>
</tr>
<tr>
<td>181</td>
<td>OE</td>
<td>FLAG11 output enable</td>
</tr>
<tr>
<td>182</td>
<td>OE</td>
<td>FLAG10 output enable</td>
</tr>
<tr>
<td>183</td>
<td>OE</td>
<td>FLAG9 output enable</td>
</tr>
<tr>
<td>184</td>
<td>0</td>
<td>FLAG10</td>
</tr>
<tr>
<td>185</td>
<td>I</td>
<td>FLAG10</td>
</tr>
<tr>
<td>186</td>
<td>0</td>
<td>FLAG9</td>
</tr>
<tr>
<td>187</td>
<td>I</td>
<td>FLAG9</td>
</tr>
</tbody>
</table>

I= Input  
0= Output  
OE= OutputEnable  
1 = Drive the associated signals during EXTEST and INTEST instructions.  
0 = Disable the associated signals during EXTEST and INTEST instructions)  
NC= Do not connect
Table D-2. Scan path position definitions (Cont'd)

<table>
<thead>
<tr>
<th>Position</th>
<th>Latch Type</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>188</td>
<td>0</td>
<td>FLAG8</td>
</tr>
<tr>
<td>189</td>
<td>I</td>
<td>FLAG8</td>
</tr>
<tr>
<td>190</td>
<td>OE</td>
<td>FLAG8 output enable</td>
</tr>
<tr>
<td>191</td>
<td>0</td>
<td>DATA0</td>
</tr>
<tr>
<td>192</td>
<td>I</td>
<td>DATA0</td>
</tr>
<tr>
<td>193</td>
<td>0</td>
<td>DATA1</td>
</tr>
<tr>
<td>194</td>
<td>I</td>
<td>DATA1</td>
</tr>
<tr>
<td>195</td>
<td>0</td>
<td>DATA2</td>
</tr>
<tr>
<td>196</td>
<td>I</td>
<td>DATA2</td>
</tr>
<tr>
<td>197</td>
<td>0</td>
<td>DATA3</td>
</tr>
<tr>
<td>198</td>
<td>I</td>
<td>DATA3</td>
</tr>
<tr>
<td>199</td>
<td>0</td>
<td>DATA4</td>
</tr>
<tr>
<td>200</td>
<td>I</td>
<td>DATA4</td>
</tr>
</tbody>
</table>

I= Input
0= Output
OE= OutputEnable

1 = Drive the associated signals during EXTEST and INTEST instructions.
0 = Disable the associated signals during EXTEST and INTEST instructions
NC= Do not connect
### Boundary Register

Table D-2. Scan path position definitions (Cont'd)

<table>
<thead>
<tr>
<th>Position</th>
<th>Latch Type</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>201</td>
<td>0</td>
<td>DATA5</td>
</tr>
<tr>
<td>202</td>
<td>I</td>
<td>DATA5</td>
</tr>
<tr>
<td>203</td>
<td>0</td>
<td>DATA6</td>
</tr>
<tr>
<td>204</td>
<td>I</td>
<td>DATA6</td>
</tr>
<tr>
<td>205</td>
<td>0</td>
<td>DATA7</td>
</tr>
<tr>
<td>206</td>
<td>I</td>
<td>DATA7</td>
</tr>
<tr>
<td>207</td>
<td>0</td>
<td>DATA8</td>
</tr>
<tr>
<td>208</td>
<td>I</td>
<td>DATA8</td>
</tr>
<tr>
<td>209</td>
<td>OE</td>
<td>DATA13:0 output enable</td>
</tr>
<tr>
<td>210</td>
<td>0</td>
<td>DATA9</td>
</tr>
<tr>
<td>211</td>
<td>I</td>
<td>DATA9</td>
</tr>
<tr>
<td>212</td>
<td>0</td>
<td>DATA10</td>
</tr>
<tr>
<td>213</td>
<td>I</td>
<td>DATA10</td>
</tr>
</tbody>
</table>

I= Input  
O= Output  
OE= OutputEnable  
1 = Drive the associated signals during EXTEST and INTEST instructions.  
0 = Disable the associated signals during EXTEST and INTEST instructions)  
NC= Do not connect
### JTAG Test Access Port

Table D-2. Scan path position definitions (Cont'd)

<table>
<thead>
<tr>
<th>Position</th>
<th>Latch Type</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>214</td>
<td>O</td>
<td>DATA11</td>
</tr>
<tr>
<td>215</td>
<td>I</td>
<td>DATA11</td>
</tr>
<tr>
<td>216</td>
<td>O</td>
<td>DATA12</td>
</tr>
<tr>
<td>217</td>
<td>I</td>
<td>DATA12</td>
</tr>
<tr>
<td>218</td>
<td>O</td>
<td>DATA13</td>
</tr>
<tr>
<td>219</td>
<td>I</td>
<td>DATA13</td>
</tr>
<tr>
<td>220</td>
<td>OE</td>
<td>SPARE5 output enable</td>
</tr>
<tr>
<td>221</td>
<td>O</td>
<td>SPARE5</td>
</tr>
<tr>
<td>222</td>
<td>I</td>
<td>SPARE5</td>
</tr>
<tr>
<td>223</td>
<td>OE</td>
<td>SPARE4 output enable</td>
</tr>
<tr>
<td>224</td>
<td>O</td>
<td>SPARE4</td>
</tr>
<tr>
<td>225</td>
<td>I</td>
<td>SPARE4</td>
</tr>
<tr>
<td>226</td>
<td>O</td>
<td>DATA14</td>
</tr>
</tbody>
</table>

I= Input  
O= Output  
OE= OutputEnable  
1 = Drive the associated signals during EXTEST and INTEST instructions.  
0 = Disable the associated signals during EXTEST and INTEST instructions)  
NC= Do not connect
## Boundary Register

Table D-2. Scan path position definitions (Cont’d)

<table>
<thead>
<tr>
<th>Position</th>
<th>Latch Type</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>227</td>
<td>I</td>
<td>DATA14</td>
</tr>
<tr>
<td>228</td>
<td>0</td>
<td>DATA15</td>
</tr>
<tr>
<td>229</td>
<td>I</td>
<td>DATA15</td>
</tr>
<tr>
<td>230</td>
<td>0</td>
<td>DATA16</td>
</tr>
<tr>
<td>231</td>
<td>I</td>
<td>DATA16</td>
</tr>
<tr>
<td>232</td>
<td>0</td>
<td>DATA17</td>
</tr>
<tr>
<td>233</td>
<td>I</td>
<td>DATA17</td>
</tr>
<tr>
<td>234</td>
<td>0</td>
<td>DATA18</td>
</tr>
<tr>
<td>235</td>
<td>I</td>
<td>DATA18</td>
</tr>
<tr>
<td>236</td>
<td>0</td>
<td>DATA19</td>
</tr>
<tr>
<td>237</td>
<td>I</td>
<td>DATA19</td>
</tr>
<tr>
<td>238</td>
<td>0</td>
<td>DATA20</td>
</tr>
<tr>
<td>239</td>
<td>I</td>
<td>DATA20</td>
</tr>
</tbody>
</table>

I= Input  
O= Output  
OE= OutputEnable  
1 = Drive the associated signals during EXTEST and INTEST instructions.  
0 = Disable the associated signals during EXTEST and INTEST instructions)  
NC= Do not connect
### JTAG Test Access Port

Table D-2. Scan path position definitions (Cont'd)

<table>
<thead>
<tr>
<th>Position</th>
<th>Latch Type</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>240</td>
<td>OE</td>
<td>SPARE3 output enable</td>
</tr>
<tr>
<td>241</td>
<td>0</td>
<td>SPARE3</td>
</tr>
<tr>
<td>242</td>
<td>I</td>
<td>SPARE3</td>
</tr>
<tr>
<td>243</td>
<td>OE</td>
<td>DATA31:14 output enable</td>
</tr>
<tr>
<td>244</td>
<td>0</td>
<td>DATA21</td>
</tr>
<tr>
<td>245</td>
<td>I</td>
<td>DATA21</td>
</tr>
<tr>
<td>246</td>
<td>0</td>
<td>DATA22</td>
</tr>
<tr>
<td>247</td>
<td>I</td>
<td>DATA22</td>
</tr>
<tr>
<td>248</td>
<td>0</td>
<td>DATA23</td>
</tr>
<tr>
<td>249</td>
<td>I</td>
<td>DATA23</td>
</tr>
<tr>
<td>250</td>
<td>0</td>
<td>DATA23</td>
</tr>
<tr>
<td>251</td>
<td>I</td>
<td>DATA24</td>
</tr>
<tr>
<td>252</td>
<td>0</td>
<td>DATA25</td>
</tr>
</tbody>
</table>

I = Input  
0 = Output  
OE = OutputEnable  

1 = Drive the associated signals during EXTEST and INTEST instructions,  
0 = Disable the associated signals during EXTEST and INTEST instructions)  
NC = Do not connect
Boundary Register

Table D-2. Scan path position definitions (Cont’d)

<table>
<thead>
<tr>
<th>Position</th>
<th>Latch Type</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>253</td>
<td>I</td>
<td>DATA25</td>
</tr>
<tr>
<td>254</td>
<td>0</td>
<td>DATA26</td>
</tr>
<tr>
<td>255</td>
<td>I</td>
<td>DATA26</td>
</tr>
<tr>
<td>256</td>
<td>0</td>
<td>DATA27</td>
</tr>
<tr>
<td>257</td>
<td>I</td>
<td>DATA27</td>
</tr>
<tr>
<td>258</td>
<td>0</td>
<td>DATA28</td>
</tr>
<tr>
<td>259</td>
<td>I</td>
<td>DATA28</td>
</tr>
<tr>
<td>260</td>
<td>0</td>
<td>DATA29</td>
</tr>
<tr>
<td>261</td>
<td>I</td>
<td>DATA29</td>
</tr>
<tr>
<td>262</td>
<td>0</td>
<td>DATA30</td>
</tr>
<tr>
<td>263</td>
<td>I</td>
<td>DATA30</td>
</tr>
<tr>
<td>264</td>
<td>0</td>
<td>DATA31</td>
</tr>
<tr>
<td>265</td>
<td>I</td>
<td>DATA31</td>
</tr>
</tbody>
</table>

I= Input  
0= Output  
OE= OutputEnable

1 = Drive the associated signals during EXTEST and INTEST instructions.  
0 = Disable the associated signals during EXTEST and INTEST instructions)

NC= Do not connect
Table D-2. Scan path position definitions (Cont’d)

<table>
<thead>
<tr>
<th>Position</th>
<th>Latch Type</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>266</td>
<td>0</td>
<td>FLAG7</td>
</tr>
<tr>
<td>267</td>
<td>I</td>
<td>FLAG7</td>
</tr>
<tr>
<td>268</td>
<td>OE</td>
<td>FLAG7 output enable</td>
</tr>
<tr>
<td>269</td>
<td>OE</td>
<td>FLAG6 output enable</td>
</tr>
<tr>
<td>270</td>
<td>OE</td>
<td>FLAG5 output enable</td>
</tr>
<tr>
<td>271</td>
<td>0</td>
<td>FLAG6</td>
</tr>
<tr>
<td>272</td>
<td>I</td>
<td>FLAG6</td>
</tr>
<tr>
<td>273</td>
<td>0</td>
<td>FLAG5</td>
</tr>
<tr>
<td>274</td>
<td>I</td>
<td>FLAG5</td>
</tr>
<tr>
<td>275</td>
<td>0</td>
<td>FLAG4</td>
</tr>
<tr>
<td>276</td>
<td>I</td>
<td>FLAG4</td>
</tr>
<tr>
<td>277</td>
<td>OE</td>
<td>FLAG4 output enable</td>
</tr>
<tr>
<td>278</td>
<td>OE</td>
<td>EMU output enable</td>
</tr>
</tbody>
</table>

I= Input
0= Output
OE= OutputEnable
1 = Drive the associated signals during EXTEST and INTEST instructions,
0 = Disable the associated signals during EXTEST and INTEST instructions)
NC= Do not connect
Device Identification Register

The ADSP-21065L does not include a device identification register.

Built-In Self-Test Instructions (BIST)

The ADSP-12065L does not support self-test functions.

### Table D-2. Scan path position definitions (Cont’d)

<table>
<thead>
<tr>
<th>Position</th>
<th>Latch Type</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>279</td>
<td>OE</td>
<td>SPARE2 output enable</td>
</tr>
<tr>
<td>280</td>
<td>0</td>
<td>SPARE2</td>
</tr>
<tr>
<td>281</td>
<td>I</td>
<td>SPARE2</td>
</tr>
<tr>
<td>282</td>
<td>I</td>
<td>ID1</td>
</tr>
<tr>
<td>283</td>
<td>I</td>
<td>ID0</td>
</tr>
<tr>
<td>284</td>
<td>0</td>
<td>EMU (This end closest to TDI scan in last)</td>
</tr>
</tbody>
</table>

I= Input  
O= Output  
OE= OutputEnable  

1 = Drive the associated signals during EXTEST and INTEST instructions.  
0 = Disable the associated signals during EXTEST and INTEST instructions)  
NC= Do not connect
Private Instructions

Loading a value of $001xx$ into the instruction register enables the private instructions reserved for emulation. The ADSP-21065L EZ-ICE® emulator uses the TAP and boundary scan to access the processor in the target system. The EZ-ICE emulator requires a target board connector for access to the TAP. For details, see “EZ-ICE Emulator” on page 12-36, in *ADSP-21065L SHARC DSP User’s Manual*.

References


This appendix lists and describes the bit definitions for the processor’s control and status registers.

Some of the control and status registers are located in the processor’s core. These registers are called system registers.

The remaining control and status registers are located in the processor’s I/O processor. These registers are called IOP registers.

All control and status bits are active high unless otherwise noted. If a bit definition gives no default value, the bit is defined at reset or its value depends on processor inputs. Make sure your application software always writes zero (0) to all reserved bits.
System Registers

System Registers

System registers are a subset of the processor’s universal register set.

Application software can write to them from an immediate field within an instruction, load them from or store them in data memory, and transfer them, in one cycle, to or from any other universal register.

The system registers are:

- **ASTAT**
  Contains arithmetic status flags.

- **IMASK**
  Contains the interrupt mask.

- **IMASKP**
  Contains the interrupt mask pointer (for nested interrupts).

- **IRPTL**
  Contains the interrupt latch.

- **MODE1**
  Contains mode control bits for the DAGs, Register File registers, data formats, interrupts, and so on.

- **MODE2**
  Contains mode control bits for the FLAG3-0, IRQ2-0, programmable timers and I/O ports, interrupts, cache, and so on.
Control and Status Registers

- **STKY**
  
  Contains status bits for ALU operations, multiplier operations, DAG operations, and status stacks. Once set, these bits remain set until they are explicitly cleared.

- **USTAT1**
  
  Contains thirty-two undefined status bits provided for use as low-overhead, general-purpose software flags or for temporarily storing data. Application software can use system register instructions to set and test the bits in this register.

- **USTAT2**
  
  Contains thirty-two undefined status bits provided for use as low-overhead, general-purpose software flags or for temporarily storing data. Application software can use system register instructions to set and test the bits in this register.

Table E-1 lists the initialization values of the system registers after reset. All control and status bits are active high unless otherwise noted. Bit values shown are the default values after reset. If no value is shown, the bit is undefined at reset or its value depends on processor inputs. Make sure your application software always writes zeros (0) to reserved bits.

Table E-1. Initialization values of the system registers after reset

<table>
<thead>
<tr>
<th>Register</th>
<th>Initialization after reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASTAT1</td>
<td>0x00nn 0000</td>
</tr>
<tr>
<td>IMASK</td>
<td>0x0003</td>
</tr>
<tr>
<td>IMASKP</td>
<td>0x0000 (cleared)</td>
</tr>
<tr>
<td>IRPTL</td>
<td>0x0000 (cleared)</td>
</tr>
</tbody>
</table>
System Registers

Table E-1. Initialization values of the system registers after reset (Cont’d)

<table>
<thead>
<tr>
<th>Register</th>
<th>Initialization after reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODE1</td>
<td>0x0000 (cleared)</td>
</tr>
<tr>
<td>MODE2</td>
<td>0xnn00 0000</td>
</tr>
<tr>
<td>STKY</td>
<td>0x540 000</td>
</tr>
<tr>
<td>USTAT1</td>
<td>0x0000 (cleared)</td>
</tr>
<tr>
<td>USTAT2</td>
<td>0x0000 (cleared)</td>
</tr>
</tbody>
</table>

1. Bits 22:19 equal the values of the FLAG3:0 inputs after reset. The flag pins become input pins after reset.
2. Bits 31:25 are the processor’s ID and revision number.

Latencies—Effect and Read

A write to any system register other than USTAT1 or USTAT2 incurs one cycle of latency before any changes take effect. This delay is called effect latency.

A read immediately following a write to a system register, except IMASKP, always reads the new value. For IMASKP, updating the contents with the new value requires an extra cycle. This delay is called read latency.
Table E-2 lists the effect latency and read latency for the ADSP-21065L system registers.

Table E-2. Read and effect latencies of the system registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Read latency</th>
<th>Effect Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASTAT</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>IRPTL</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>IMASK</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>IMASKP</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MODE1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>MODE2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>STKY</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>USTAT1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>USTAT2</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

0= Write takes effect on the cycle immediately after the write instruction executes.
1= One cycle of latency.

System Register Bit Manipulation Instruction

Application software can use the system register bit manipulation instruction to set, clear, toggle, or test specific bits in the system registers.

An immediate field in the bit manipulation instruction specifies the affected bits. For a detailed description of this instruction, see “Group IV–Miscellaneous” in Appendix A, Instruction Set Reference.
System Registers

For example:

\[
\begin{align*}
\text{BIT SET MODE2 0x00000070;} \\
\text{BIT TST ASTAT 0x00002000; (result in BTF flag)}
\end{align*}
\]

Although both the Shifter and ALU have bit manipulation capabilities, these computations operate on Register File locations only.

System register bit manipulation instructions eliminate the overhead associated with transferring system registers to and from the Register File. Table E-3 lists these operations.

Table E-3. System register bit manipulation operations

<table>
<thead>
<tr>
<th>Bit Instruction (System Registers)</th>
<th>Shifter Operation (Data Register File)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT SET register data</td>
<td>(Rn = \text{BSET Rx BY Ry</td>
</tr>
<tr>
<td>BIT CLR register data</td>
<td>(Rn = \text{BCLR Rx BY Ry</td>
</tr>
<tr>
<td>BIT TGL register data</td>
<td>(Rn = \text{BTGL Rx BY Ry</td>
</tr>
<tr>
<td>BIT TST register data(^1)</td>
<td>(\text{BTST Rx BY Ry</td>
</tr>
<tr>
<td>BIT XOR register data(^1)</td>
<td></td>
</tr>
</tbody>
</table>

\(^1\) Result stored in BTF flag (ASTAT).
\(^2\) Result stored in SZ status flag (ASTAT).

Bit Test Flag

The Bit Test Flag (BTF), bit 18 in the ASTAT register, stores the result from the system register bit manipulation instruction’s test and XOR operations:

- The test operation sets BTF if all specified bits in the system register are set.
Control and Status Registers

- The XOR operation sets BTF if all bits in the system register match the specified bit pattern.

Application software can use the state of the BTF bit in conditional instructions accordingly.
System Registers

ASTAT
Arithmetic Status Register

The ASTAT register provides status information on the most recent ALU and Multiplier operations and stores the input values of the programmable I/O ports FLAG\(_{3:0}\) only.

The processor bases comparisons for conditional instructions on this status information.

For details on using the ASTAT register, in *ADSP-21065L SHARC DSP User’s Manual* see:

- Chapter 2, Computation Units
- Chapter 3, Program Sequencing
- Chapter 12, System Design

In this manual, see:

- Appendix A, Instruction Set Reference
- Appendix B, Compute Operation Reference

After reset, all bits in the ASTAT register, except 22:19 (FLG\(_{3:0}\)), are initialized to 0. The value of bits 22:19 correspond to the value of the FLAG\(_{3:0}\) inputs.

Figure E-1 shows the default values of the ASTAT register bits.
Control and Status Registers

Figure E-1. ASTAT register bits
**System Registers**

Table E-4 lists and describes the individual bits of the ASTAT register.

Table E-4. ASTAT register

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>AZ</td>
<td>ALU result zero or floating-point underflow</td>
</tr>
<tr>
<td>1</td>
<td>AV</td>
<td>ALU overflow</td>
</tr>
<tr>
<td>2</td>
<td>AN</td>
<td>ALU result negative</td>
</tr>
<tr>
<td>3</td>
<td>AC</td>
<td>ALU fixed-point carry</td>
</tr>
<tr>
<td>4</td>
<td>AS</td>
<td>ALU X-input sign (ABS and MANT operations)</td>
</tr>
<tr>
<td>5</td>
<td>AI</td>
<td>ALU floating-point invalid operation</td>
</tr>
<tr>
<td>6</td>
<td>MN</td>
<td>Multiplier result negative</td>
</tr>
<tr>
<td>7</td>
<td>MV</td>
<td>Multiplier overflow</td>
</tr>
<tr>
<td>8</td>
<td>MU</td>
<td>Multiplier floating-point underflow</td>
</tr>
<tr>
<td>9</td>
<td>MI</td>
<td>Multiplier floating-point invalid operation</td>
</tr>
<tr>
<td>10</td>
<td>AF</td>
<td>ALU floating-point operation</td>
</tr>
<tr>
<td>11</td>
<td>SV</td>
<td>Shifter overflow</td>
</tr>
<tr>
<td>12</td>
<td>SZ</td>
<td>Shifter result zero</td>
</tr>
<tr>
<td>13</td>
<td>SS</td>
<td>Shifter input sign</td>
</tr>
<tr>
<td>14-17</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>BTF</td>
<td>Bit test flag for system registers</td>
</tr>
<tr>
<td>19</td>
<td>FLG0</td>
<td>FLAG0 value</td>
</tr>
<tr>
<td>20</td>
<td>FLG1</td>
<td>FLAG1 value</td>
</tr>
</tbody>
</table>
### Control and Status Registers

Table E-4. ASTAT register (Cont’d)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>FLG2</td>
<td>FLAG2 value</td>
</tr>
<tr>
<td>22</td>
<td>FLG3</td>
<td>FLAG3 value</td>
</tr>
<tr>
<td>23</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>24-31</td>
<td>CACC</td>
<td>Compare accumulation shift register</td>
</tr>
</tbody>
</table>
System Registers

IMASK and IRPTL
Interrupt Mask and Latch Registers

The IMASK and IRPTL registers have identical bit positions 0 through 31 that correspond to the ADSP-21065L interrupts in order of priority from highest to lowest.

For details on using the IMASK and IRPTL registers, in ADSP-21065L SHARC DSP User’s Manual see:

- Chapter 2, Computation Units
- Chapter 3, Program Sequencing
- Chapter 4, Data Addressing
- Chapter 5, Memory
- Chapter 6, DMA
- Chapter 7, Multiprocessing
- Chapter 8, Host Interface

In this manual, see Appendix A, Instruction Set Reference.

After reset, the IRPTL register is initialized to $0\times0000\ 0000$, and the IMASK register is initialized to $0\times0000\ 0003$. Figure E-2 shows the default values of the IMASK register bits only, with bit values: $0 = \text{bit masked (disabled)}$, and $1 = \text{bit unmasked (enabled)}$. 
Control and Status Registers

Figure E-2. IMASK and IRPTL register bits

Vector addresses of individual bits in Table E-5 are the offsets from 0x0000 8000, the base address of the interrupt vector table in internal
System Registers

memory. The base address of the interrupt vector table in external memory is 0x0002 0000.

Table E-5 lists and describes the individual bits of the IMASK and IRPTL registers.

Table E-5. IMASK and IRPTL registers

<table>
<thead>
<tr>
<th>Bit</th>
<th>Vector address</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x00</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0x04</td>
<td>RSTI</td>
<td>Reset (read only, nonmaskable)</td>
</tr>
<tr>
<td>2</td>
<td>0x08</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0x0C</td>
<td>SOVFI</td>
<td>Status stack or loop stack overflow or PC stack full</td>
</tr>
<tr>
<td>4</td>
<td>0x10</td>
<td>TMZHI</td>
<td>Timer-0 (high priority option)</td>
</tr>
<tr>
<td>5</td>
<td>0x14</td>
<td>VIRPTI</td>
<td>Vector interrupt</td>
</tr>
<tr>
<td>6</td>
<td>0x18</td>
<td>IRQ2I</td>
<td>IRQ2 asserted</td>
</tr>
<tr>
<td>7</td>
<td>0x1C</td>
<td>IRQ1I</td>
<td>IRQ1 asserted</td>
</tr>
<tr>
<td>8</td>
<td>0x20</td>
<td>IRQ0I</td>
<td>IRQ0 asserted</td>
</tr>
<tr>
<td>9</td>
<td>0x24</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0x28</td>
<td>SPR0I</td>
<td>DMA channel 0/1; SPORT0 receive A&amp;B</td>
</tr>
<tr>
<td>11</td>
<td>0x2C</td>
<td>SPR1I</td>
<td>DMA channel 2/3; SPORT1 receive A&amp;B</td>
</tr>
<tr>
<td>12</td>
<td>0x30</td>
<td>SPT0I</td>
<td>DMA channel 4/5; SPORT0 transmit A&amp;B</td>
</tr>
</tbody>
</table>
Table E-5. IMASK and IRPTL registers (Cont’d)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Vector address</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>0x34</td>
<td>SPT1I</td>
<td>DMA channel 6/7; SPORT1 transmit A&amp;B</td>
</tr>
<tr>
<td>14-15</td>
<td>0x38-0x3C</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>0x40</td>
<td>EP0I</td>
<td>DMA chn 8; external port buffer 0</td>
</tr>
<tr>
<td>17</td>
<td>0x44</td>
<td>EP1I</td>
<td>DMA chn 9; external port buffer 1</td>
</tr>
<tr>
<td>18-20</td>
<td>0x48-0x50</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>0x54</td>
<td>CB7I</td>
<td>Circular buffer 7 overflow</td>
</tr>
<tr>
<td>22</td>
<td>0x58</td>
<td>CB15I</td>
<td>Circular buffer 15 overflow</td>
</tr>
<tr>
<td>23</td>
<td>0x5C</td>
<td>TMZLI</td>
<td>Timer—0 (low priority option)</td>
</tr>
<tr>
<td>24</td>
<td>0x60</td>
<td>FIXI</td>
<td>Fixed-point overflow</td>
</tr>
<tr>
<td>25</td>
<td>0x64</td>
<td>FLTOI</td>
<td>Floating-point overflow exception</td>
</tr>
<tr>
<td>26</td>
<td>0x68</td>
<td>FLTUI</td>
<td>Floating-point underflow exception</td>
</tr>
<tr>
<td>27</td>
<td>0x6C</td>
<td>FLTII</td>
<td>Floating-point invalid exception</td>
</tr>
<tr>
<td>28</td>
<td>0x70</td>
<td>SFT0I</td>
<td>User software interrupt 0</td>
</tr>
<tr>
<td>29</td>
<td>0x74</td>
<td>SFT1I</td>
<td>User software interrupt 1</td>
</tr>
<tr>
<td>30</td>
<td>0x78</td>
<td>SFT2I</td>
<td>User software interrupt 2</td>
</tr>
<tr>
<td>31</td>
<td>0x7C</td>
<td>SFT3I</td>
<td>User software interrupt 3</td>
</tr>
</tbody>
</table>
System Registers

**MODE1 Register**

The MODE1 register provides control of ALU and Multiplier fixed- and floating-point operations, interrupt nesting, and DAGx operation.

For details on using the MODE1 register, in *ADSP-21065L SHARC DSP User's Manual* see:

- Chapter 2, Computation Units
- Chapter 3, Program Sequencing
- Chapter 4, Data Addressing
- Chapter 5, Memory

In this manual, see *Appendix A, Instruction Set Reference*.

After reset, the MODE1 register is initialized to 0x0000 0000 as shown in Figure E-3.
Control and Status Registers

Figure E-3. MODE1 register bits
System Registers

Application software can use the Shifter and ALU instructions on Register File locations or the System Register Bit Manipulation instruction on system registers to set individual bits. See Table E-3 on page E-6.

Table E-6 lists and describes the individual bits of the MODE1 register.

Table E-6. MODE1 register

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>BR8</td>
<td>Bit reversing for I8 (DAG2).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = disable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = enable</td>
</tr>
<tr>
<td>1</td>
<td>BR0</td>
<td>Bit reversing for I0 (DAG1).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = disable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = enable</td>
</tr>
<tr>
<td>2</td>
<td>SRCU</td>
<td>Alternate register select for computation units.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = enable as primary</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = enable as alternate</td>
</tr>
<tr>
<td>3</td>
<td>SRD1H</td>
<td>DAG1 alternate register select (7-4).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = enable as primary</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = enable as alternate</td>
</tr>
<tr>
<td>4</td>
<td>SRD1L</td>
<td>DAG1 alternate register select (3-0).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = enable as primary</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = enable as alternate</td>
</tr>
<tr>
<td>5</td>
<td>SRD2H</td>
<td>DAG2 alternate register select (15-12).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = enable as primary</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = enable as alternate</td>
</tr>
</tbody>
</table>
## Control and Status Registers

### Table E-6. MODE1 register (Cont’d)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>SRD2L</td>
<td>DAG2 alternate register select (11-8).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = enable as primary</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = enable as alternate</td>
</tr>
<tr>
<td>7</td>
<td>SRRFH</td>
<td>Register file alternate select for R15-R8.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = enable as primary</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = enable as alternate</td>
</tr>
<tr>
<td>8-9</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>SRRFL</td>
<td>Register file alternate select for R7-R0.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = enable as primary</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = enable as alternate</td>
</tr>
<tr>
<td>11</td>
<td>NESTM</td>
<td>Interrupt nesting enable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = disable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = enable</td>
</tr>
<tr>
<td>12</td>
<td>IRPTEN</td>
<td>Global interrupt enable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = disable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = enable</td>
</tr>
<tr>
<td>13</td>
<td>ALUSAT</td>
<td>ALU saturation enable (full scale in fixed-point).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = disable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = enable</td>
</tr>
<tr>
<td>14</td>
<td>SSE¹</td>
<td>Short word, sign extension enable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = disable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = enable</td>
</tr>
</tbody>
</table>
**System Registers**

Table E-6. MODE1 register (Cont'd)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>TRUNC</td>
<td>Floating-point data rounding enable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = round to nearest</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = truncate</td>
</tr>
<tr>
<td>16</td>
<td>RND32</td>
<td>Floating-point data rounding length.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = round to 40 bits</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = round to 32 bits</td>
</tr>
<tr>
<td>17-18</td>
<td>CSEL</td>
<td>Condition code select.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 = bus master condition²</td>
</tr>
<tr>
<td>19-31</td>
<td>Reserved</td>
<td></td>
</tr>
</tbody>
</table>

1. Does not apply to PX register writes.
2. The bus master condition (BM) indicates whether the ADSP-21065L is the current bus master in a multiprocessor system. To enable this condition, both bits 17 and 18 must be zero (0); otherwise the condition always evaluates false.
**MODE2 Register**

The MODE2 register provides control of the programmable I/O ports \( \text{FLAG}_{3:0} \) only, the programmable timers and their interrupts, interrupt request sensitivity, and the instruction cache.

For details on using the MODE2 register, in *ADSP-21065L SHARC DSP User’s Manual* see:

- Chapter 3, Program Sequencing
- Chapter 7, Multiprocessing
- Chapter 11, Programmable Timers and I/O Ports
- Chapter 12, System Design

In this manual, see Appendix A, Instruction Set Reference.

After reset, all bits of the MODE2 register, except bits 31:25, are initialized to 0 as shown in Figure E-4. Bits 31:25 are the processor’s ID and revision number.
Application software can use the Shifter and ALU instructions on Register File locations or the System Register Bit Manipulation instruction on system registers to set individual bits. See Table E-3 on page E-6.
Table E-7 lists and describes the individual bits of the MODE2 register.

Table E-7. MODE2 register

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>IRQ0E</td>
<td>IRQ0 sensitivity.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0= level-sensitive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1= edge-sensitive</td>
</tr>
<tr>
<td>1</td>
<td>IRQ1E</td>
<td>IRQ1 sensitivity.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0= level-sensitive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1= edge-sensitive</td>
</tr>
<tr>
<td>2</td>
<td>IRQ2E</td>
<td>IRQ2 sensitivity.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0= level-sensitive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1= edge-sensitive</td>
</tr>
<tr>
<td>3</td>
<td>PERIOD_CNT0</td>
<td>Timer 0 period count enable (pulse counter mode only).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0= enable width count</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1= enable period count</td>
</tr>
<tr>
<td>4</td>
<td>CADIS</td>
<td>Cache disable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0= enable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1= disable</td>
</tr>
<tr>
<td>5</td>
<td>TIMENO</td>
<td>Timer 0 enable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0= disable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1= enable</td>
</tr>
</tbody>
</table>
## System Registers

Table E-7. MODE2 register (Cont'd)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>BUSLK</td>
<td>External bus lock (multiprocessor systems).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0= disable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1= enable</td>
</tr>
<tr>
<td>7</td>
<td>PWMOUT0</td>
<td>Timer 0 mode control.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0= enable pulse counter mode (PWM_EVENT pin is input)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1= enable pulsewidth generation mode (PWM_EVNT pin is output)</td>
</tr>
<tr>
<td>8</td>
<td>INT_HI0</td>
<td>Timer 0 interrupt vector location.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For interrupt status values, see Table E-8 on page E-26</td>
</tr>
<tr>
<td>9</td>
<td>PULSE_HI0</td>
<td>Timer 0 leading edge select (pulse width counter mode only).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0= low to high transition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1= high to low transition</td>
</tr>
<tr>
<td>10</td>
<td>PERIOD_CNT1</td>
<td>Timer 1 period count enable (pulse counter mode only).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0= enable width count capture</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1= enable period count capture</td>
</tr>
<tr>
<td>11</td>
<td>TIMEN1</td>
<td>Timer 1 enable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0= disable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1= enable</td>
</tr>
</tbody>
</table>
### Control and Status Registers

Table E-7. MODE2 register (Cont’d)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 12  | PWMOUT1  | Timer 1 mode control.  
0= enable pulse counter mode (PWM_EVENT pin is input)  
1= enable pulsewidth generation mode (PWM_EVNT pin is output) |
| 13  | INT_HI1  | Timer 1 interrupt vector location.  
For interrupt status values, see Table E-8 on page E-26 |
| 14  | PULSE_HI1| Timer 1 leading edge select (pulse width counter mode only).  
0= low to high transition  
1= high to low transition |
| 15  | FLG00    | FLAG0 status.  
0= input  
1= output |
| 16  | FLG10    | FLAG1 status.  
0= input  
1= output |
| 17  | FLG20    | FLAG2 status.  
0= input  
1= output |
| 18  | FLG30    | FLAG3 status.  
0= input  
1= output |
System Registers

Table E-7. MODE2 register (Cont’d)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 19  | CAFRZ | Cache freeze.  
|     |      | 0= update cache  
|     |      | 1= freeze cache |
| 20-24 | Reserved |
| 25-31 | Processor ID and revision number.  
|       | (read-only) |
|       | Processor ID in bits 31:30 and 27:25.  
|       | ADSP-21065L ID=11001.  
|       | Revision number in bits 29:28. |

Table E-8. Timer interrupt status

<table>
<thead>
<tr>
<th>INT_HI0</th>
<th>INT_HI1</th>
<th>IRPTL Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Both timers latch to TMZLI</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Timer 1 latches to TMZLI; timer 0 latches to TMZHI</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Timer 1 latches to TMZHI; timer 0 latches to TMZLI</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Both timers latch to TMZHI</td>
</tr>
</tbody>
</table>

TMZLI = IRPTL register bit 23  
TMZHI = IRPTL register bit 4
Control and Status Registers

Sticky Status Register (STKY)

The STKY register provides status information on ALU, Multiplier, DAGx, and status stack exceptions.

For details on using the STKY register, in ADSP-21065L SHARC DSP User’s Manual see:

- Chapter 2, Computation Units
- Chapter 3, Program Sequencing
- Chapter 4, Data Addressing
- Chapter 11, Programmable Timers and I/O Ports

In this manual, see Appendix A, Instruction Set Reference.

After reset, the STKY register is initialized to 0x0540 0000 as shown in Figure E-5.
System Registers

Figure E-5. STKY register bits
Control and Status Registers

All STKY register bits, are sticky, except 21, 22, 24, and 26, which are read-only (see Chapter 3, Program Sequencing, in ADSP-21065L SHARC DSP User’s Manual). A sticky bit remains set until explicitly cleared.

Application software can use the Shifter and ALU instructions on Register File locations or the System Register Bit Manipulation instruction on system registers to set individual bits. See Figure E-3 on page E-6. However, since bits 21:26 are read-only, writes to the STKY register have no effect on them.

Table E-9 lists and describes the individual bits of the STKY register.

Table E-9. STKY register

<table>
<thead>
<tr>
<th>Bit</th>
<th>Bit Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>AUS</td>
<td>ALU floating-point underflow</td>
</tr>
<tr>
<td>1</td>
<td>AVS</td>
<td>ALU floating-point overflow</td>
</tr>
<tr>
<td>2</td>
<td>AOS</td>
<td>ALU fixed-point overflow</td>
</tr>
<tr>
<td>3-4</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>AIS</td>
<td>ALU floating-point invalid operation</td>
</tr>
<tr>
<td>6</td>
<td>MOS</td>
<td>Multiplier fixed-point overflow</td>
</tr>
<tr>
<td>7</td>
<td>MVS</td>
<td>Multiplier fixed-point overflow</td>
</tr>
<tr>
<td>8</td>
<td>MUS</td>
<td>Multiplier floating-point underflow</td>
</tr>
<tr>
<td>9</td>
<td>MIS</td>
<td>Multiplier floating-point invalid operation</td>
</tr>
<tr>
<td>10-11</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>PULSE_CAPO</td>
<td>Timer 0 pulse captured bit.</td>
</tr>
</tbody>
</table>
## System Registers

### Table E-9. STKY register  (Cont’d)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Bit Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>CNT_EXP0 /</td>
<td>Timer 0 counter expired or counter overflowed</td>
</tr>
<tr>
<td></td>
<td>CNT_OVF0</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>PULSE_CAP1</td>
<td>Timer 1 pulse captured bit</td>
</tr>
<tr>
<td>15</td>
<td>CNT_EXP /</td>
<td>Timer 1 counter expired or counter overflowed</td>
</tr>
<tr>
<td></td>
<td>CNT_OVF1</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>CB7S</td>
<td>DAG1 circular buffer 7 overflow</td>
</tr>
<tr>
<td>18</td>
<td>CB15S</td>
<td>DAG2 circular buffer 15 overflow</td>
</tr>
<tr>
<td>19-20</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>PCFL</td>
<td>PC stack full (nonsticky)</td>
</tr>
<tr>
<td>22</td>
<td>PCEM</td>
<td>PC stack empty (nonsticky)</td>
</tr>
<tr>
<td>23</td>
<td>SSOV</td>
<td>Status stack overflow (MODE1 and ASTAT)</td>
</tr>
<tr>
<td>24</td>
<td>SSEM</td>
<td>Status stack empty (nonsticky)</td>
</tr>
<tr>
<td>25</td>
<td>LSOV</td>
<td>Loop stack overflow (loop address and loop counter)</td>
</tr>
<tr>
<td>26</td>
<td>LSEM</td>
<td>Loop stack empty (nonsticky)</td>
</tr>
<tr>
<td>27-31</td>
<td>Reserved</td>
<td></td>
</tr>
</tbody>
</table>
IOP Registers

The IOP registers are a separate set of control and data registers that are memory-mapped into the processor’s internal memory.

Application software use the IOP registers to configure system-level functions, including serial port I/O, DMA transfers, programmable timers, general-purpose I/O ports, vector interrupts, and the SDRAM interface. The processor’s on-chip I/O processor handles I/O operations independently of and transparently to the processor’s core.

To program the IOP registers, application software must write to the appropriate address in memory. Code executing in the processor’s core or on an external device, such as a host processor or another ADSP-21065L, can program the IOP registers.

Application software can use the symbolic names of the registers or individual bits. The file `def21065L.h`, provided in the INCLUDE directory of the ADSP-21000 Family Development Software, contains the `#define` definitions for these symbols. Listing E.6 on page E-116 lists the contents of the `def21065L.h` file.

IOP Registers Summary

Tables E-10, E-11, E-12, and E-13 on page E-32 through page E-35 list the IOP registers (by functional group) that configure processor and system control, DMA operations, and serial port operations. Table E-15 on page E-43 shows the memory-mapped address, functional group, and reset initialization value of each IOP register.

Any external device, either another ADSP-21065L or a host processor, that is bus master can access the memory-mapped IOP registers. This enables, for example, an external device to set up a DMA transfer to the processor’s internal memory without the processor’s intervention.
A conflict occurs when both the processor and an external bus master try to access the same IOP register group at the same time. In this case, the external device always has priority, forcing the processor to wait until the external device has completed its access. Table E-15 on page E-43 shows the different IOP register groups.

For easy access to the most important registers, the IOP registers are arranged so that a host processor (or other bus master) can read or write to the smallest amount of memory. The host needs to control only a small number of address lines to access a set of 16, 32, or 64 IOP registers, including SYSCON, SYSTAT, VIRPT, WAIT, MSGR7-0, and one or two full DMA channels.

Table E-10. System control (SC) IOP registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSCON</td>
<td>32</td>
<td>System configuration register</td>
</tr>
<tr>
<td>SYSTAT</td>
<td>32</td>
<td>System status register</td>
</tr>
<tr>
<td>DMSTAT</td>
<td>32</td>
<td>DMA status register</td>
</tr>
<tr>
<td>WAIT</td>
<td>32</td>
<td>Memory wait state configuration register</td>
</tr>
<tr>
<td>VIRPT</td>
<td>32</td>
<td>Multiprocessor vector interrupt register</td>
</tr>
<tr>
<td>MSGR0</td>
<td>32</td>
<td>Message register 0</td>
</tr>
<tr>
<td>MSGR1</td>
<td>32</td>
<td>Message register 1</td>
</tr>
<tr>
<td>MSGR2</td>
<td>32</td>
<td>Message register 2</td>
</tr>
<tr>
<td>MSGR3</td>
<td>32</td>
<td>Message register 3</td>
</tr>
<tr>
<td>MSGR4</td>
<td>32</td>
<td>Message register 4</td>
</tr>
<tr>
<td>MSGR5</td>
<td>32</td>
<td>Message register 5</td>
</tr>
</tbody>
</table>
Control and Status Registers

Table E-10. System control (SC) IOP registers (Cont’d)

<table>
<thead>
<tr>
<th>Register</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSGR6</td>
<td>32</td>
<td>Message register 6</td>
</tr>
<tr>
<td>MSGR7</td>
<td>32</td>
<td>Message register 7</td>
</tr>
<tr>
<td>BMAX</td>
<td>32</td>
<td>Bus timeout maximum</td>
</tr>
<tr>
<td>BCNT</td>
<td>16</td>
<td>Bus timeout counter</td>
</tr>
<tr>
<td>SDRDIV</td>
<td>32</td>
<td>SDRAM refresh counter</td>
</tr>
<tr>
<td>IOCTL</td>
<td>32</td>
<td>SDRAM and general-purpose I/O port control</td>
</tr>
<tr>
<td>IOSTAT</td>
<td>32</td>
<td>General-purpose I/O port status</td>
</tr>
<tr>
<td>TPERIOD0</td>
<td>32</td>
<td>Timer 0 count period</td>
</tr>
<tr>
<td>TPWIDTH0</td>
<td>32</td>
<td>Timer 0 pulse width</td>
</tr>
<tr>
<td>TCOUNT0</td>
<td>32</td>
<td>Timer 0 counter</td>
</tr>
<tr>
<td>TPERIOD1</td>
<td>32</td>
<td>Timer 1 count period</td>
</tr>
<tr>
<td>TPWIDTH1</td>
<td>32</td>
<td>Timer 1 pulse width</td>
</tr>
<tr>
<td>TCOUNT1</td>
<td>32</td>
<td>Timer 1 counter</td>
</tr>
</tbody>
</table>

Table E-11. DMA address (DA) IOP registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IIOA, IMROA, CROA, CPR0A, GPROA</td>
<td>16-18</td>
<td>DMA channel 0 parameter registers (SPORT0 receive; A data)</td>
</tr>
</tbody>
</table>
### IOP Registers

Table E-11. DMA address (DA) IOP registers  (Cont’d)

<table>
<thead>
<tr>
<th>Register</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IIR0B, IMR0B, CROB, CPR0B, GPROB</td>
<td>16-18</td>
<td>DMA channel 1 parameter registers (SPORT0 receive: B data)</td>
</tr>
<tr>
<td>IIR1A, IMR1A, CR1A, CPR1A, GPR1A</td>
<td>16-18</td>
<td>DMA channel 2 parameter registers (SPORT1 receive: A data)</td>
</tr>
<tr>
<td>IIR1B, IMR1B, CR1B, CPR1B, GPR1B</td>
<td>16-18</td>
<td>DMA channel 3 parameter registers (SPORT1 receive: B data)</td>
</tr>
<tr>
<td>IIT0A, IMT0A, CT0A, CPT0A, GPT0A</td>
<td>16-18</td>
<td>DMA channel 4 parameter registers (SPORT0 transmit: A data)</td>
</tr>
<tr>
<td>IIT0B, IMT0B, CT0B, CPT0B, GPT0B</td>
<td>16-18</td>
<td>DMA channel 5 parameter registers (SPORT0 transmit: B data)</td>
</tr>
<tr>
<td>IIT1A, IMT1A, CT1A, CPT1A, GPT1A</td>
<td>16-32</td>
<td>DMA channel 6 parameter registers (SPORT1 transmit: A data)</td>
</tr>
<tr>
<td>IIT1B, IMT1B, CT1B, CPT1B, GPT1B</td>
<td>16-32</td>
<td>DMA channel 7 parameter registers (SPORT1 transmit: B data)</td>
</tr>
<tr>
<td>IIEP0, IMEP0, CEPO, CPEPO, GPEP0, EIEP0, EMEP0, ECEP0</td>
<td>16-32</td>
<td>DMA channel 8 parameter registers (external port buffer 0)</td>
</tr>
<tr>
<td>IIEP1, IMEP1, CEPI, CPEPI, GPEP1, EIEP1, EMEP1, ECEP1</td>
<td>16-32</td>
<td>DMA channel 9 parameter registers (external port buffer 1)</td>
</tr>
</tbody>
</table>
## Control and Status Registers

### Table E-12. DMA buffer (DB) IOP registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPB0</td>
<td>48</td>
<td>External port FIFO buffer 0</td>
</tr>
<tr>
<td>EPB1</td>
<td>48</td>
<td>External port FIFO buffer 1</td>
</tr>
<tr>
<td>DMAC0</td>
<td>16</td>
<td>DMA channel 8 control register or external port buffer 0</td>
</tr>
<tr>
<td>DMAC1</td>
<td>16</td>
<td>DMA channel 9 control register or external port buffer 1</td>
</tr>
</tbody>
</table>

### Table E-13. Serial port (SP) IOP registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STCTL0</td>
<td>32</td>
<td>SPORT0 transmit control register</td>
</tr>
<tr>
<td>SRCTL0</td>
<td>32</td>
<td>SPORT0 receive control register</td>
</tr>
<tr>
<td>TXO_A</td>
<td>32</td>
<td>SPORT0 transmit data buffer A</td>
</tr>
<tr>
<td>RXO_A</td>
<td>32</td>
<td>SPORT0 receive data buffer A</td>
</tr>
<tr>
<td>TDIVO</td>
<td>32</td>
<td>SPORT0 transmit divisors</td>
</tr>
<tr>
<td>RDIVO</td>
<td>32</td>
<td>SPORT0 receive divisors</td>
</tr>
<tr>
<td>MTCS0</td>
<td>32</td>
<td>SPORT0 multichannel transmit selector</td>
</tr>
<tr>
<td>MRCS0</td>
<td>32</td>
<td>SPORT0 multichannel receive selector</td>
</tr>
<tr>
<td>MTCCSO</td>
<td>32</td>
<td>SPORT0 multichannel transmit compand selector</td>
</tr>
<tr>
<td>MRCCSO</td>
<td>32</td>
<td>SPORT0 multichannel receive compand selector</td>
</tr>
</tbody>
</table>
Table E-13. Serial port (SP) IOP registers (Cont’d)

<table>
<thead>
<tr>
<th>Register</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEYWD0</td>
<td>32</td>
<td>SPORT0 receive comparison</td>
</tr>
<tr>
<td>KEYMASKD</td>
<td>32</td>
<td>SPORT0 receive comparison mask</td>
</tr>
<tr>
<td>TX0_B</td>
<td>32</td>
<td>SPORT0 transmit data buffer B</td>
</tr>
<tr>
<td>RX0_B</td>
<td>32</td>
<td>SPORT0 receive data buffer B</td>
</tr>
<tr>
<td>STCTL1</td>
<td>32</td>
<td>SPORT1 transmit control register</td>
</tr>
<tr>
<td>SRCTL1</td>
<td>32</td>
<td>SPORT1 receive control register</td>
</tr>
<tr>
<td>TX1_A</td>
<td>32</td>
<td>SPORT1 transmit data buffer A</td>
</tr>
<tr>
<td>RX1_A</td>
<td>32</td>
<td>SPORT1 receive data buffer A</td>
</tr>
<tr>
<td>TDIV1</td>
<td>32</td>
<td>SPORT1 transmit divisors</td>
</tr>
<tr>
<td>RDIV1</td>
<td>32</td>
<td>SPORT1 receive divisors</td>
</tr>
<tr>
<td>MTCS1</td>
<td>32</td>
<td>SPORT1 multichannel transmit selector</td>
</tr>
<tr>
<td>MRCS1</td>
<td>32</td>
<td>SPORT1 multichannel receive selector</td>
</tr>
<tr>
<td>MTCCS1</td>
<td>32</td>
<td>SPORT1 multichannel transmit compand selector</td>
</tr>
<tr>
<td>MRCCS1</td>
<td>32</td>
<td>SPORT1 multichannel receive compand selector</td>
</tr>
<tr>
<td>KEYWD1</td>
<td>32</td>
<td>SPORT1 receive comparison</td>
</tr>
<tr>
<td>KEYMASK1</td>
<td>32</td>
<td>SPORT1 receive comparison mask</td>
</tr>
</tbody>
</table>
Control and Status Registers

Table E-13. Serial port (SP) IOP registers (Cont’d)

<table>
<thead>
<tr>
<th>Register</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TX1_B</td>
<td>32</td>
<td>SPORT1 transmit data buffer B</td>
</tr>
<tr>
<td>RX1_B</td>
<td>32</td>
<td>SPORT1 receive data buffer B</td>
</tr>
</tbody>
</table>

This section lists and defines the individual bits in the following IOP registers:

- **BCNT**
  
  Bus timeout counter register.

- **BMAX**
  
  Bus timeout maximum register.

- **DMAC_{1-0}**
  
  External port DMA control register for DMA channels 8 and 9.

- **DMASTAT**
  
  DMA channel status register. Contains the status bits for each DMA channel.

- **IOCTL**
  
  SDRAM and programmable I/O port (for FLAG_{11-4}) control register.

- **IOSTAT**
  
  Programmable I/O port status register for FLAG_{11-4}. 

---

ADSP-21065L SHARC DSP Technical Reference   E-37
IOP Registers

- KEYMASK\textsubscript{1-0}
  Key word mask registers for serial ports 0 and 1.
- KEYWD\textsubscript{1-0}
  Key word registers for serial ports 0 and 1.
- MRCCS\textsubscript{1-0}
  Multichannel receive companding control registers for serial ports 0 and 1.
- MRCS\textsubscript{1-0}
  Multichannel receive control registers for serial ports 0 and 1.
- MSG\textsubscript{7-0}
  Message registers.
- MTCCS\textsubscript{1-0}
  Multichannel transmit companding control registers for serial ports 0 and 1.
- MTCS\textsubscript{1-0}
  Multichannel transmit control registers for serial ports 0 and 1.
- RDIV\textsubscript{1-0}
  Receive clock divisor registers for serial ports 0 and 1.
- SDRDIV
  SDRAM refresh counter register.
Control and Status Registers

- **SRCTL\(_{1-0}\)**
  Receive control registers for serial ports 0 and 1.

- **STCTL\(_{1-0}\)**
  Transmit control registers for serial ports 0 and 1.

- **SYSCON**
  System control register.

- **SYSTAT**
  System status register.

- **TCOUNT\(_{1-0}\)**
  Counter register for timers 0 and 1.

- **TDIV\(_{1-0}\)**
  Transmit clock divisor registers for serial ports 0 and 1.

- **TPERIOD\(_{1-0}\)**
  Timer count period registers for timers 0 and 1.

- **TPWIDTH\(_{1-0}\)**
  Timer counter output pulse width registers for timers 0 and 1.

- **VIRPT**
  Vector interrupt register.

- **WAIT**
  External memory wait state register.
IOP Registers

Table E-14 lists the initialization values of the major IOP registers after reset. All control and status bits are active high unless otherwise noted. Bit values shown are the default values after reset. If no value is shown, the bit is undefined at reset, or its value depends on processor inputs. Make sure your application software always writes zeros (0) to reserved bits.

Table E-14. Initialization values of the IOP registers after reset

<table>
<thead>
<tr>
<th>Register</th>
<th>Initialization after reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMACx</td>
<td>0x0000 0000</td>
</tr>
<tr>
<td>DMASTAT</td>
<td>0xnnnn nnnn (not initialized)</td>
</tr>
<tr>
<td>IOCTL</td>
<td>0x0000 0000</td>
</tr>
<tr>
<td>IOSTAT</td>
<td>0x0000 0000</td>
</tr>
<tr>
<td>RDIVx/TDIVx</td>
<td>0xnnnn nnnn (not initialized)</td>
</tr>
<tr>
<td>SRCTLx</td>
<td>0x0000 0000</td>
</tr>
<tr>
<td>STCTLx</td>
<td>0x0000 0000</td>
</tr>
<tr>
<td>SYSCON</td>
<td>0x0000 0020</td>
</tr>
<tr>
<td>SYSTAT</td>
<td>0x0000 nnn0(^1)</td>
</tr>
<tr>
<td>WAIT</td>
<td>0x200D 6B5A</td>
</tr>
</tbody>
</table>

\(^1\) Bits 11:4 depend on the value of the ID\(_{1-0}\) inputs.

IOP Register Access Restrictions

Because the IOP registers are memory-mapped, you cannot write to them directly with data from memory. Instead, you must write data from or read data to the processor’s core registers, usually one of the Register File’s...
Control and Status Registers

general-purpose registers (R15–R0). External devices, usually another ADSP-21065L or a host, can also write or read the IOP registers.

You cannot perform an internal DMA transfer to any of the processor’s IOP registers. DMA transfers occur through the IOP register’s DMA buffers only. These transfers are directly controlled by the processor’s DMA controller, however, not with addresses generated over the I/O address bus. During DMA transfers, the DMA controller writes or reads the DMA buffer registers to internal memory over the I/O data bus. The DMA buffer registers include EPB0, EPB1 (external port data buffers 0 and 1) and TX0_x, RX0_x, TX1_x, and RX1_x (serial port data buffers).

IOP Register Group Access Contention

The processor has four separate on-chip buses that can access the memory-mapped IOP registers independently:

- **PM bus**
  The PMD bus connects the processor’s core registers to its IOP registers, memory, and the external port data buffers.

- **DM bus**
  The DMD bus connects the processor’s core registers to its IOP registers, memory, and the external port data buffers.

- **I/O bus**
  The I/O bus connects the external port’s data buffers to memory and to the on-chip I/O processor. The I/O bus carries data transferring to or from the IOP register’s DMA buffers.

- **External port bus**
  The external port bus connects the off-chip DATA\textsubscript{32-0} bus to all on-chip buses.
IOP Registers

Each of these buses can attempt to read or write an IOP register at any time. Contention occurs when more than one bus attempts to access the same group of IOP registers at the same time (see Table E-15 on page E-43). However, both the I/O bus and the external port bus can access the IOP register’s DMA buffers simultaneously, enabling DMA transfers to internal memory to occur at the processor’s full speed.

The processor resolves IOP register group access conflicts on a fixed priority basis:

- External port ↔ IOP register accesses 1st priority
- PM/DM bus ↔ IOP register accesses 2nd priority
- I/O bus ↔ IOP register accesses 3rd priority

The bus with the highest priority gains access to the IOP registers first, and the processor’s core or its I/O processor generates extra cycles to hold off any lower priority accesses. If the DMA controller has granted a DMA I/O access, it completes that access before the processor grants an access from another bus.

The external port DMA data buffers (EPB0 and EPB1) are six-word deep FIFOs. An input to the buffers can occur in the same cycle as an output. The external port bus has separate and independent access to these buffers. Contention occurs when the PM bus, the DM bus, and/or the I/O bus try to access the data buffers at the same time. In this case the I/O bus access has first priority, but the processor holds off subsequent I/O bus accesses until the PM and/or DM bus accesses finish.

IOP Register Write Latencies

The processor completes internal writes to the IOP register at the end of the cycle in which they occur. Therefore, the IOP register reads back the newly written value on the very next cycle.
Control and Status Registers

Not all writes, however, take effect in the next cycle. Most control and mode bits take effect in the second cycle after completion of the write. The external port packing control bits and buffer flush bits, however, take effect in the third cycle after completion of the write.

Accesses by the external port and the processor’s core may conflict if they attempt to access the same IOP register group. In this case, the processor delays the core’s access until all external port accesses have finished.

Table E-15. IOP register addresses, reset values, and groups

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>Reset Value</th>
<th>Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSCON</td>
<td>0x0000</td>
<td>0x0000 0020</td>
<td>SC</td>
<td>System configuration</td>
</tr>
<tr>
<td>VIRPT</td>
<td>0x0001</td>
<td>0x0000 8014</td>
<td>SC</td>
<td>Vector interrupt table</td>
</tr>
<tr>
<td>WAIT</td>
<td>0x0002</td>
<td>0x21AD 6B5A</td>
<td>SC</td>
<td>External memory wait state</td>
</tr>
<tr>
<td>SYSTAT</td>
<td>0x0003</td>
<td>0x0000 0nn0</td>
<td>SC</td>
<td>System status</td>
</tr>
<tr>
<td>EPB0</td>
<td>0x0004</td>
<td>NI</td>
<td>DB</td>
<td>External port DMA FIFO buffer 0</td>
</tr>
<tr>
<td>EPB1</td>
<td>0x0005</td>
<td>NI</td>
<td>DB</td>
<td>External port DMA FIFO buffer 1</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x0006 - 0x0007</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSGR0</td>
<td>0x0008</td>
<td>NI</td>
<td>SC</td>
<td>Message register 0</td>
</tr>
<tr>
<td>MSGR1</td>
<td>0x0009</td>
<td>NI</td>
<td>SC</td>
<td>Message register 1</td>
</tr>
</tbody>
</table>

Groups: DA = DMA Address register; DB = DMA Buffer; SC = System Control; SP = Serial Port
NI = Not Initialized
### Table E-15. IOP register addresses, reset values, and groups (Cont’d)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>Reset Value</th>
<th>Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSGR2</td>
<td>0x000A</td>
<td>NI</td>
<td>SC</td>
<td>Message register 2</td>
</tr>
<tr>
<td>MSGR3</td>
<td>0x000B</td>
<td>NI</td>
<td>SC</td>
<td>Message register 3</td>
</tr>
<tr>
<td>MSGR4</td>
<td>0x000C</td>
<td>NI</td>
<td>SC</td>
<td>Message register 4</td>
</tr>
<tr>
<td>MSGR5</td>
<td>0x000D</td>
<td>NI</td>
<td>SC</td>
<td>Message register 5</td>
</tr>
<tr>
<td>MSGR6</td>
<td>0x000E</td>
<td>NI</td>
<td>SC</td>
<td>Message register 6</td>
</tr>
<tr>
<td>MSGR7</td>
<td>0x000F</td>
<td>NI</td>
<td>SC</td>
<td>Message register 7</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x0010-0x0017</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BMAX</td>
<td>0x0018</td>
<td>0x0000 0000</td>
<td>SC</td>
<td>Bus timeout maximum</td>
</tr>
<tr>
<td>BCNT</td>
<td>0x0019</td>
<td>0x0000 0000</td>
<td>SC</td>
<td>BUs timeout counter</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x001A-0x001B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DMAC0</td>
<td>0x001C</td>
<td>0x0000 0000</td>
<td>DB</td>
<td>DMA chn 8 control register (Ext. port buffer 0)</td>
</tr>
<tr>
<td>DMAC1</td>
<td>0x001D</td>
<td>0x0000 0000</td>
<td>DB</td>
<td>DMA chn 9 control register (Ext. port buffer 1)</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x001E-0x001F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SDRDIV</td>
<td>0x0020</td>
<td>NI</td>
<td>SC</td>
<td>SDRAM refresh counter</td>
</tr>
</tbody>
</table>

Groups: DA = DMA Address register; DB = DMA Buffer; SC = System Control; SP = Serial Port
NI = Not Initialized
### Control and Status Registers

---

**Table E-15. IOP register addresses, reset values, and groups (Cont’d)**

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>Reset Value</th>
<th>Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved 0x0021-0x0027</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPERIOD0 0x0028</td>
<td>NI</td>
<td>SC</td>
<td>Timer 0 count period</td>
<td></td>
</tr>
<tr>
<td>TPWIDTH0 0x0029</td>
<td>NI</td>
<td>SC</td>
<td>Timer 0 output pulse width</td>
<td></td>
</tr>
<tr>
<td>TCOUNT0 0x002A</td>
<td>NI</td>
<td>SC</td>
<td>Timer 0 counter</td>
<td></td>
</tr>
<tr>
<td>TPERIOD1 0x002B</td>
<td>NI</td>
<td>SC</td>
<td>Timer 1 count period</td>
<td></td>
</tr>
<tr>
<td>TPWIDTH1 0x002C</td>
<td>NI</td>
<td>SC</td>
<td>Timer 1 output pulse width</td>
<td></td>
</tr>
<tr>
<td>TCOUNT1 0x002D</td>
<td>NI</td>
<td>SC</td>
<td>Timer 1 counter</td>
<td></td>
</tr>
<tr>
<td>IOCTL 0x002E</td>
<td>0x0000 0000</td>
<td>SC</td>
<td>General-purpose FLG11-4 I/O and SDRAM control</td>
<td></td>
</tr>
<tr>
<td>IOSTAT 0x002F</td>
<td>0x0000 0000</td>
<td>SC</td>
<td>General-purpose FLG11-4 I/O status</td>
<td></td>
</tr>
<tr>
<td>IIR0B 0x0030</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 1 index (SPORT0 rcv B)</td>
<td></td>
</tr>
<tr>
<td>IMROB 0x0031</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 1 modify</td>
<td></td>
</tr>
<tr>
<td>CR0B 0x0032</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 1 count</td>
<td></td>
</tr>
<tr>
<td>CPR0B 0x0033</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 1 chain pointer</td>
<td></td>
</tr>
</tbody>
</table>

Groups: DA = DMA Address register; DB = DMA Buffer; SC = System Control; SP = Serial Port; NI = Not Initialized

---
Table E-15. IOP register addresses, reset values, and groups  (Cont’d)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>Reset Value</th>
<th>Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPROB</td>
<td>0x0034</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 1 general purpose</td>
</tr>
</tbody>
</table>

Reserved 0x0035-0x0036

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>Reset Value</th>
<th>Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMASTAT</td>
<td>0x0037</td>
<td>NI</td>
<td>SC</td>
<td>DMA channel status</td>
</tr>
<tr>
<td>IIR1B</td>
<td>0x0038</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 3 index (SPORT1 rcv B)</td>
</tr>
<tr>
<td>IMR1B</td>
<td>0x0039</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 3 modify</td>
</tr>
<tr>
<td>CR1B</td>
<td>0x003A</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 3 count</td>
</tr>
<tr>
<td>CPR1B</td>
<td>0x003B</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 3 chain pointer</td>
</tr>
<tr>
<td>GPR1B</td>
<td>0x003C</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 3 general purpose</td>
</tr>
</tbody>
</table>

Reserved 0x003D-0x003F

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>Reset Value</th>
<th>Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IIEP0</td>
<td>0x0040</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 8 index (EPB0)</td>
</tr>
<tr>
<td>IMEPO</td>
<td>0x0041</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 8 modify</td>
</tr>
<tr>
<td>CEPO</td>
<td>0x0042</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 8 count</td>
</tr>
<tr>
<td>CPEPO</td>
<td>0x0043</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 8 chain pointer</td>
</tr>
</tbody>
</table>

Groups: DA = DMA Address register; DB = DMA Buffer; SC = System Control; SP = Serial Port
NI = Not Initialized
## Control and Status Registers

Table E-15. IOP register addresses, reset values, and groups (Cont’d)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>Reset Value</th>
<th>Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPEPO</td>
<td>0x0044</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 8 general purpose</td>
</tr>
<tr>
<td>EIEPO</td>
<td>0x0045</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 8 external index</td>
</tr>
<tr>
<td>EMEPO</td>
<td>0x0046</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 8 external modify</td>
</tr>
<tr>
<td>ECEPO</td>
<td>0x0047</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 8 external count</td>
</tr>
<tr>
<td>IIEP1</td>
<td>0x0048</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 9 index (EPB1)</td>
</tr>
<tr>
<td>IMEP1</td>
<td>0x0049</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 9 modify</td>
</tr>
<tr>
<td>CEP1</td>
<td>0x004A</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 9 count</td>
</tr>
<tr>
<td>CPEP1</td>
<td>0x004B</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 9 chain pointer</td>
</tr>
<tr>
<td>GPEP1</td>
<td>0x004C</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 9 general purpose</td>
</tr>
<tr>
<td>EIEP1</td>
<td>0x004D</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 9 external index</td>
</tr>
<tr>
<td>EMEP1</td>
<td>0x004E</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 9 external modify</td>
</tr>
<tr>
<td>ECEP1</td>
<td>0x004F</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 9 external count</td>
</tr>
</tbody>
</table>

Groups: DA = DMA Address register; DB = DMA Buffer; SC = System Control; SP = Serial Port
NI = Not Initialized
### IOP Registers

Table E-15. IOP register addresses, reset values, and groups  (Cont’d)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>Reset Value</th>
<th>Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IIT0B</td>
<td>0x0050</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 5 index (SPORT0 xmit B)</td>
</tr>
<tr>
<td>IMT0B</td>
<td>0x0051</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 5 modify</td>
</tr>
<tr>
<td>CT0B</td>
<td>0x0052</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 5 count</td>
</tr>
<tr>
<td>CPT0B</td>
<td>0x0053</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 5 chain pointer</td>
</tr>
<tr>
<td>GPT0B</td>
<td>0x0054</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 5 general purpose</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x0055-0x0057</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IIT1B</td>
<td>0x0058</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 7 index (SPORT1 xmit B)</td>
</tr>
<tr>
<td>IMT1B</td>
<td>0x0059</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 7 modify</td>
</tr>
<tr>
<td>CT1B</td>
<td>0x005A</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 7 count</td>
</tr>
<tr>
<td>CPT1B</td>
<td>0x005B</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 7 chain pointer</td>
</tr>
<tr>
<td>GPT1B</td>
<td>0x005C</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 7 general purpose</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x005D-0x005F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IIR0A</td>
<td>0x0060</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 0 index (SPORT0 rcv A)</td>
</tr>
</tbody>
</table>

Groups: DA = DMA Address register; DB = DMA Buffer; SC = System Control; SP = Serial Port
NI = Not Initialized
## Control and Status Registers

### Table E-15. IOP register addresses, reset values, and groups (Cont'd)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>Reset Value</th>
<th>Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMROA</td>
<td>0x0061</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 0 modify</td>
</tr>
<tr>
<td>CR0A</td>
<td>0x0062</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 0 count</td>
</tr>
<tr>
<td>CPR0A</td>
<td>0x0063</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 0 chain pointer</td>
</tr>
<tr>
<td>GPR0A</td>
<td>0x0064</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 0 general purpose</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x0065-0x0067</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IIR1A</td>
<td>0x0068</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 2 index (SPORT1 rcv A)</td>
</tr>
<tr>
<td>IMR1A</td>
<td>0x0069</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 2 modify</td>
</tr>
<tr>
<td>CR1A</td>
<td>0x006A</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 2 count</td>
</tr>
<tr>
<td>CPR1A</td>
<td>0x006B</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 2 chain pointer</td>
</tr>
<tr>
<td>GPR1A</td>
<td>0x006C</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 2 general purpose</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x006D-0x006F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IIT0A</td>
<td>0x0070</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 4 index (SPORT0 xmit A)</td>
</tr>
<tr>
<td>IMT0A</td>
<td>0x0071</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 4 modify</td>
</tr>
</tbody>
</table>

Groups: DA = DMA Address register; DB = DMA Buffer; SC = System Control; SP = Serial Port
NI = Not Initialized
## IOP Registers

Table E-15. IOP register addresses, reset values, and groups (Cont’d)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>Reset Value</th>
<th>Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTOA</td>
<td>0x0072</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 4 count</td>
</tr>
<tr>
<td>CPTOA</td>
<td>0x0073</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 4 chain pointer</td>
</tr>
<tr>
<td>GPTOA</td>
<td>0x0074</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 4 general purpose</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x0075-0x0077</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IIT1A</td>
<td>0x0078</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 6 index (SPORT1 xmit A)</td>
</tr>
<tr>
<td>IMT1A</td>
<td>0x0079</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 6 modify</td>
</tr>
<tr>
<td>CT1A</td>
<td>0x007A</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 6 count</td>
</tr>
<tr>
<td>CPT1A</td>
<td>0x007B</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 6 chain pointer</td>
</tr>
<tr>
<td>GPT1A</td>
<td>0x007C</td>
<td>NI</td>
<td>DA</td>
<td>DMA chn 6 general purpose</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x007D-0x00DF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STCTL0</td>
<td>0x00E0</td>
<td>0x0000 0000</td>
<td>SP</td>
<td>SPORT0 transmit control</td>
</tr>
<tr>
<td>SRCTL0</td>
<td>0x00E1</td>
<td>0x0000 0000</td>
<td>SP</td>
<td>SPORT0 receive control</td>
</tr>
<tr>
<td>TXO_A</td>
<td>0x00E2</td>
<td>NI</td>
<td>SP</td>
<td>SPORT0 transmit data buffer A</td>
</tr>
</tbody>
</table>

Groups: DA = DMA Address register; DB = DMA Buffer; SC = System Control; SP = Serial Port
NI = Not Initialized
## Control and Status Registers

Table E-15. IOP register addresses, reset values, and groups (Cont'd)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>Reset Value</th>
<th>Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RX0_A</td>
<td>0x00E3</td>
<td>NI</td>
<td>SP</td>
<td>SPORT0 receive data buffer A</td>
</tr>
<tr>
<td>TDIV0</td>
<td>0x00E4</td>
<td>NI</td>
<td>SP</td>
<td>SPORT0 transmit divisor</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x00E5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RDIV0</td>
<td>0x00E6</td>
<td>NI</td>
<td>SP</td>
<td>SPORT0 receive divisor</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x00E7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MTCS0</td>
<td>0x00E8</td>
<td>NI</td>
<td>SP</td>
<td>SPORT0 multichn xmit select</td>
</tr>
<tr>
<td>MRCS0</td>
<td>0x00E9</td>
<td>NI</td>
<td>SP</td>
<td>SPORT0 multichn rcv select</td>
</tr>
<tr>
<td>MTCCS0</td>
<td>0x00EA</td>
<td>NI</td>
<td>SP</td>
<td>SPORT0 multichn xmit compand select</td>
</tr>
<tr>
<td>MRCCS0</td>
<td>0x00EB</td>
<td>NI</td>
<td>SP</td>
<td>SPORT0 multichn rcv compand select</td>
</tr>
<tr>
<td>KEYWD0</td>
<td>0x00EC</td>
<td>NI</td>
<td>SP</td>
<td>SPORT0 keyword</td>
</tr>
<tr>
<td>IMASK0</td>
<td>0x00ED</td>
<td>NI</td>
<td>SP</td>
<td>SPORT0 keyword mask</td>
</tr>
<tr>
<td>TX0_B</td>
<td>0x00EE</td>
<td>NI</td>
<td>SP</td>
<td>SPORT0 transmit data buffer B</td>
</tr>
</tbody>
</table>

Groups: DA = DMA Address register; DB = DMA Buffer; SC = System Control; SP = Serial Port
NI = Not Initialized
### Table E-15. IOP register addresses, reset values, and groups  (Cont’d)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>Reset Value</th>
<th>Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RX0_B</td>
<td>0x00EF</td>
<td>NI</td>
<td>SP</td>
<td>SPORT0 receive data buffer B</td>
</tr>
<tr>
<td>STCTL1</td>
<td>0x00F0</td>
<td>0x0000 0000</td>
<td>SP</td>
<td>SPORT1 transmit control</td>
</tr>
<tr>
<td>SRCTL1</td>
<td>0x00F1</td>
<td>0x0000 0000</td>
<td>SP</td>
<td>SPORT1 receive control</td>
</tr>
<tr>
<td>TX1_A</td>
<td>0x00F2</td>
<td>NI</td>
<td>SP</td>
<td>SPORT1 transmit data buffer A</td>
</tr>
<tr>
<td>RX1_A</td>
<td>0x00F3</td>
<td>NI</td>
<td>SP</td>
<td>SPORT1 receive data buffer A</td>
</tr>
<tr>
<td>TDIV1</td>
<td>0x00F4</td>
<td>NI</td>
<td>SP</td>
<td>SPORT1 transmit divisor</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x00F5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RDIV1</td>
<td>0x00F6</td>
<td>NI</td>
<td>SP</td>
<td>SPORT1 receive divisor</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x00F7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MTCS1</td>
<td>0x00F8</td>
<td>NI</td>
<td>SP</td>
<td>SPORT1 multichn xmit select</td>
</tr>
<tr>
<td>MRCS1</td>
<td>0x00F9</td>
<td>NI</td>
<td>SP</td>
<td>SPORT1 multichn rcv select</td>
</tr>
<tr>
<td>MTCCS1</td>
<td>0x00FA</td>
<td>NI</td>
<td>SP</td>
<td>SPORT1 multichn xmit compand select</td>
</tr>
</tbody>
</table>

Groups: DA = DMA Address register; DB = DMA Buffer; SC = System Control; SP = Serial Port
NI = Not Initialized
## Control and Status Registers

Table E-15. IOP register addresses, reset values, and groups (Cont’d)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>Reset Value</th>
<th>Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRCCS1</td>
<td>0x00FB</td>
<td>NI</td>
<td>SP</td>
<td>SPORT1 multichn rcv compand select</td>
</tr>
<tr>
<td>KEYWD1</td>
<td>0x00FC</td>
<td>NI</td>
<td>SP</td>
<td>SPORT1 keyword</td>
</tr>
<tr>
<td>IMASK1</td>
<td>0x00FD</td>
<td>NI</td>
<td>SP</td>
<td>SPORT1 keyword mask</td>
</tr>
<tr>
<td>TX1_B</td>
<td>0x00FE</td>
<td>NI</td>
<td>SP</td>
<td>SPORT1 transmit data buffer B</td>
</tr>
<tr>
<td>RX1_B</td>
<td>0x00FF</td>
<td>NI</td>
<td>SP</td>
<td>SPORT1 receive data buffer B</td>
</tr>
</tbody>
</table>

Groups: DA = DMA Address register; DB = DMA Buffer; SC = System Control; SP = Serial Port
NI = Not Initialized
DMACx
External Port DMA Control Registers

Applications use the DMACx registers to control external port DMA operations on DMA channels 8 and 9 (EPB0 and EPB1 data buffers).

For details on using the DMACx register, in *ADSP-21065L SHARC DSP User’s Manual* see:

- Chapter 6, DMA
- Chapter 7, Multiprocessing
- Chapter 8, Host Interface
- Chapter 9, Serial Ports

In this manual, see *Appendix A, Instruction Set Reference*.

The DMAC1-0 registers are memory-mapped in internal memory at addresses 0x001C and 0x001D, respectively.

After reset, the DMACx registers are initialized to 0x0000 0000 as shown in Figure E-6. DMAC0 is initialized during booting according to the booting mode in use.
Control and Status Registers

Figure E-6. DMACx register bits
Table E-16 lists and describes the individual bits of the DMACx register.

Table E-16. DMACx register

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>DEN</td>
<td>DMA enable for external ports. Enables/disables DMA operations on the external port buffers.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0= disable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1= enable</td>
</tr>
<tr>
<td>1</td>
<td>CHEN</td>
<td>DMA chaining enable for external ports. Enables/disables DMA chaining operations on the external port buffers.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0= disable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>With DEN=0, specifies both DMA and DMA chaining disabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>With DEN=1, specifies DMA enabled, chaining disabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1= enable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>With DEN=0, specifies chain insertion mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>With DEN=1, specifies DMA, chaining, and autochaining enabled</td>
</tr>
</tbody>
</table>
Control and Status Registers

Table E-16. DMACx register (Cont’d)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 2   | TRAN | DMA transfer direction.  
|     |      | Changes the direction of data transfers on external port channels 8/9.  
|     |      | 0= receive (external to internal)  
|     |      | With EXTERN=1, specifies a read from external memory.  
|     |      | 1= transmit (internal to external)  
|     |      | With EXTERN=1, specifies a write to external memory.  
| 3-4 | PS   | Pack status (read-only).  
|     |      | Indicates which packing stage (1st, 2nd, or 3rd) the packing buffer is currently on.  
|     |      | 00= packing done (3rd stage)  
|     |      | 01= in first stage of packing/unpacking (all modes)  
|     |      | 10= in second stage of packing/unpacking 16- to 48-bit words or 32- to 48-bit words  
|     |      | 11= reserved  

### Table E-16. DMACx register (Cont'd)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>DTYPE</td>
<td>Data type. &lt;br&gt;Identifies the type of data transferring through the external port buffers. &lt;br&gt;0= data &lt;br&gt;Data word either 32- or 40-bits, depending on IMDW (SYSCON) bits. &lt;br&gt;1= instructions &lt;br&gt;Overrides the IMDW bits and forces a 48-bit, 3-column memory transfer. &lt;br&gt;DMA controller uses this information to determine the word width for internal memory.</td>
</tr>
<tr>
<td>6-7</td>
<td>PMODE</td>
<td>Packing mode. &lt;br&gt;Specifies the internal word width for the packing mode. &lt;br&gt;00= no packing/unpacking &lt;br&gt;01= 16-bit ↔ 32-bit &lt;br&gt;10= 16-bit ↔ 48-bit &lt;br&gt;11= 32-bit ↔ 48-bit &lt;br&gt;Used with the HBW bits (SYSCON), which specify the external word width.</td>
</tr>
<tr>
<td>8</td>
<td>MSWF</td>
<td>Most significant word first. &lt;br&gt;Specifies the word order for packing 16-bit data to 32- or 48-bit data. &lt;br&gt;0= LSW 16-bit word first &lt;br&gt;1= 16-bit word first</td>
</tr>
</tbody>
</table>
## Control and Status Registers

### Table E-16. DMACx register (Cont'd)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 9   | MASTER | DMA master mode enable.  
In combination with HSHAKE and EXTERN to set the DMA transfer mode.  
0= disable  
1= enable  
See Table E-17 on page E-61. |
| 10  | HSHAKE | DMA handshake enable.  
In combination with HSHAKE and EXTERN to set the DMA transfer mode.  
0= disable  
1= enable  
See Table E-17 on page E-61. |
| 11  | INTIO | Single word I/O interrupt enable.  
Enables/disables interrupts for individual words the external port buffers transmit or receive.  
0=disable  
1=enable  
With TRAN=0, a full or partially full EPBx RX buffer generates an interrupt.  
With TRAN=1, an empty or partially full EPBx TX buffer generates an interrupt.  
Single word I/O interrupts are useful for implementing interrupt-driven, single-word transfers under the control of the processor's core. |
## IOP Registers

### Table E-16. DMACx register (Cont'd)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 12  | EXTERN | External DMA handshake mode enable. In combination with HSHAKE and EXTERN to set the DMA transfer mode. 
     |      | 0=’disable  
     |      | 1= enable  
     |      | See Table E-17 on page E-61. |
| 13  | FLSH | Flush external port buffer. 
     |      | Reinitializes the state of the DMA channel by flushing the EPBx buffer and resetting any internal DMA states and clearing the FS and PS status bits. This operation has a two-cycle latency.  
     |      | 1= flush  
     |      | This self-clearing control bit is not latched and always reads as 0.  
     |      | To avoid unexpected results, use FLSH to clear a DMA channel only when the channel is inactive and at least one cycle before setting any other DMACx control bit. Read the DMASTAT register to determine a channel’s active status. |
Control and Status Registers

Table E-16. DMACx register (Cont’d)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>14-15 FS</td>
<td>External port buffer status. A read-only status bit that indicates whether or not data is present in the EPBx buffer. During an off-chip transfer, these bits indicate whether the TX buffer has room for more data. During an on-chip transfer, these bits indicate whether the RX buffer contains new data. 00= empty 01= reserved 10= partially full 11= full</td>
<td></td>
</tr>
<tr>
<td>16-31</td>
<td>Reserved</td>
<td></td>
</tr>
</tbody>
</table>

Table E-17. DMA transfer modes

<table>
<thead>
<tr>
<th>MASTER</th>
<th>HSHAKE</th>
<th>EXTERN</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Slave mode. The DMA controller generates a DMA request whenever an RX buffer is not empty or a TX buffer is not full.</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>
Table E-17. DMA transfer modes  (Cont’d)

<table>
<thead>
<tr>
<th>MASTER</th>
<th>HSHAKE</th>
<th>EXTERN</th>
<th>Description</th>
</tr>
</thead>
</table>
| 0      | 1      | 0      | Handshake mode.  
Applies to the EPBx buffers (channels 8 and 9) only.  
The DMA controller generates a DMA request when the DMARx line is asserted and transfers the data when the DMAGx line is asserted. |
| 0      | 1      | 1      | External handshake mode.²  
Applies to the EPBx buffers (channels 8 and 9) only.  
Identical to handshake mode, except the DMA controller transfers the data between external memory and an external device. |
| 1      | 0      | 0      | Master mode.  
The DMA controller attempts to transfer data whenever the DMA counter >0³ and either the RX buffer is not empty or the TX buffer is not full.  
Keep DMAR1 high if DMA channel 8 is in master mode.  
Keep DMAR2 high if DMA channel 9 is in master mode. |
| 1      | 0      | 1      | Reserved. |
Control and Status Registers

Table E-17. DMA transfer modes (Cont’d)

<table>
<thead>
<tr>
<th>MASTER</th>
<th>HSHAKE</th>
<th>EXTERN</th>
<th>Description</th>
</tr>
</thead>
</table>
| 1      | 1      | 0      | Paced master mode.\(^2\)  
Applies to the EPBx buffers (channels 8 and 9) only.  
The DMARx signal paces DMA transfers. The DMA controller generates a DMA request when DMARx is asserted.  
DMARx requests function the same way as in handshake mode, and the DMA controller transfers the data when RD or WR is asserted.  
The address is driven as in normal master mode.  
ORing the RD-DMAGx and WR-DMAGx pairs requires no external gates, enabling buffer access with zero-wait state and no idle states.  
Wait states and Acknowledge (ACK) apply to paced master mode transfers. For details, see Chapter 5, Memory, in ADSP-21065L SHARC DSP User’s Manual. |
| 1      | 1      | 1      | Reserved. |

---

1 If TRAN=1 for an external read of the EPBx buffer, the DMA controller fills the buffer as soon as the DEN bit is set to 1.  
2 You cannot use DMA paced master mode or external handshake mode with SDRAM transfers.  
3 When an external DMA channel is configured for output (TRAN=1), the EPBx buffer starts to fill as soon as the channel becomes enabled, whether or not DMARx assertions or DMA slave mode DMA buffer reads have been made.
**DMASTAT**

**DMA Channel Status Register**

The DMASTAT register maintains status bits for each DMA channel.

For details on using the DMASTAT register, in *ADSP-21065L SHARC DSP User’s Manual* see:

- Chapter 6, DMA
- Chapter 7, Multiprocessing
- Chapter 8, Host Interface
- Chapter 9, Serial Ports

In this manual, see *Appendix A, Instruction Set Reference*.

The DMASTAT register is memory-mapped in internal memory at address 0x0037.

For a particular channel, the DMA controller sets the channel active status bit when DMA is enabled and the current DMA sequence has not finished. It sets the chaining status bit if the channel is currently performing chaining operations or if a chaining operation is pending.

A single cycle of latency occurs between the time changes in internal status occur and the time the DMA controller updates the DMASTAT register.

Status does not change on the master ADSP-21065L during an external port DMA operation until the external portion has finished (until the EPBx buffers are empty).

In chain insertion mode (DEN=0, CHEN=1), a channel’s chaining status will never be 1. Make sure to test channel status for readiness, so your program can rewrite the channels’s chain pointer (CPx register).
The processor does not initialize the DMASTAT register at reset as shown in Figure E-7.

Status bit value 0 = inactive (disabled), and status bit value 1 = active. Depending on the type of status, channel or chaining, active means transferring or waiting to transfer a current block of data or TCB. For channel status, active also means not transferring TCB, and inactive means DMA disabled or transfer finished or chaining in progress.
Table E-18 lists and describes the individual bits of the DMASTAT register.

Table E-18. DMASTAT register

<table>
<thead>
<tr>
<th>Bit</th>
<th>DMA Chn.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Status of receive buffer RX0_A</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>Status of receive buffer RX1_A</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>Status of transmit buffer TX0_A</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>Status of transmit buffer TX1_A</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>Status of receive buffer RX0_B</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>Status of receive buffer RX1_B</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>Status of external port buffer EPB0</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td>Status of external port buffer EPB1</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>Status of transmit buffer TX0_B</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td>Status of transmit buffer TX1_B</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>Chaining status of receive buffer RX0_A</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>Chaining status of receive buffer RX1_A</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>Chaining status of transmit buffer TX0_A</td>
</tr>
<tr>
<td>13</td>
<td>6</td>
<td>Chaining status of transmit buffer TX1_A</td>
</tr>
</tbody>
</table>

Channel status:
1= active, current block (not xfering TCB).
0= inactive, DMA disabled, xfer complete, or chaining.

Channel chaining status:
1= xfering TCB or waiting to xfer TCB.
0 = chaining disabled.
## Control and Status Registers

Table E-18. DMASTAT register

<table>
<thead>
<tr>
<th>Bit</th>
<th>DMA Chn.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>1</td>
<td>Chaining status of receive buffer RX0_B</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>Chaining status of receive buffer RX1_B</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>Chaining status of external port buffer EPB0</td>
</tr>
<tr>
<td>17</td>
<td>9</td>
<td>Chaining status of external port buffer EPB1</td>
</tr>
<tr>
<td>18</td>
<td>5</td>
<td>Chaining status of transmit buffer TX0_B</td>
</tr>
<tr>
<td>19</td>
<td>7</td>
<td>Chaining status of transmit buffer TX1_B</td>
</tr>
<tr>
<td>20-31</td>
<td>Reserved</td>
<td></td>
</tr>
</tbody>
</table>

Channel status:
- 1 = active, current block (not xfering TCB).
- 0 = inactive, DMA disabled, xfer complete, or chaining.

Channel chaining status:
- 1 = xfering TCB or waiting to xfer TCB.
- 0 = chaining disabled.
Applications use the IOCTL register to set the direction of the programmable general-purpose I/O ports (FLG_{11-4} only) and to set up SDRAM configuration selections.

For details on using the IOCTL register, in *ADSP-21065L SHARC DSP User’s Manual* see the following chapters.

- Chapter 3, Program Sequencing
- Chapter 10, SDRAM Interface
- Chapter 11, Programmable Timers and I/O Ports

In this manual, see Appendix A, Instruction Set Reference.

IOCTL is memory-mapped in internal memory at address $0x002E$.

After reset, the IOCTL register is initialized to $0x0000\ 0000$ as shown in Figure E-8.
Figure E-8. IOCTL register bits
Table E-19 lists and describes the bits of the IOCTL register.

Table E-19. IOCTL register

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>FLG40</td>
<td>FLAG4 direction set.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0= input</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1= output</td>
</tr>
<tr>
<td>1</td>
<td>FLG50</td>
<td>FLAG5 direction set.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0= input</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1= output</td>
</tr>
<tr>
<td>2</td>
<td>FLG60</td>
<td>FLAG6 direction set.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0= input</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1= output</td>
</tr>
<tr>
<td>3</td>
<td>FLG70</td>
<td>FLAG7 direction set.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0= input</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1= output</td>
</tr>
<tr>
<td>4</td>
<td>FLG80</td>
<td>FLAG8 direction set.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0= input</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1= output</td>
</tr>
<tr>
<td>5</td>
<td>FLG90</td>
<td>FLAG9 direction set.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0= input</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1= output</td>
</tr>
<tr>
<td>6</td>
<td>FLG100</td>
<td>FLAG10 direction set.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0= input</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1= output</td>
</tr>
</tbody>
</table>
Control and Status Registers

Table E-19. IOCTL register  (Cont’d)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 7   | FLG110 | FLAG11 direction set.  
|     |       | 0= input 
|     |       | 1= output |
| 8-9 | Reserved | |
| 10  | DSDCTL | Disable SDCLK0, RAS, CAS, SDWE, DQM, SDCKE, 
|     |       | and MUX. 
|     |       | Hi-Zs all SDRAM control signals.  
|     |       | 0= enable 
|     |       | 1= disable |
| 11  | DSDCK1 | Disable SDCLK1.  
|     |       | Hi-Zs SDCLK1 signal only.  
|     |       | 0= enable 
|     |       | 1= disable |
| 12-14 | SDPGS | SDRAM page size.  
|       |       | Specifies the size of the SDRAM page, in number of words.  
|       |       | 000=1024 words  
|       |       | 001=512 words  
|       |       | 010=256 words  
|       |       | others = reserved |
| 15  | SDSRF | SDRAM self-refresh mode.  
|     |       | This bit always reads as 0.  
|     |       | 0= disable 
|     |       | 1= enable |
### IOP Registers

Table E-19. IOCTL register  (Cont'd)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 16-17 | SDCL | SDRAM CAS latency.  
Sets the delay, in number of clock cycles, between the time the SDRAM detects the read command and the time the data is available at its outputs.  
00= no SDRAM  
01= 1 cycle  
10= 2 cycles  
11= 3 cycles |
| 18-20 | SDTRAS | SDRAM $t_{\text{ras}}$ spec in number of clock cycles. |
| 21-23 | SDTRP  | SDRAM $t_{\text{rp}}$ spec in number of clock cycles. |
| 24    | SDPM  | SDRAM power-up option.  
Specifies the sequence of commands in the SDRAM power-up cycle.  
0= precharge. 8 CBR. mode register set  
1= precharge. mode register set. 8 CBR |
Control and Status Registers

Table E-19. IOCTL register (Cont’d)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 25-27 | SDBS | SDRAM bank select. Specifies which of the ADSP-21065L’s external memory bank connects to SDRAM.  
000=no SDRAM  
100=bank 0  
101=bank 1  
110=bank 2  
111=bank 3  
other = reserved  
For proper operation of the SDRAM controller, in the WAIT register, set the EBxWS bits to 0 and the EBxWM bits appropriately for the external memory bank to which the SDRAM connects. See Table E. on page E-113 |
| 28   | SDBUF | SDRAM buffer. Enables/disables pipelining of address and control signals when using external buffering between the ADSP-21065L and SDRAM. Supports multiple SDRAMs connected in parallel.  
0= disable  
1= enable |
Table E-19. IOCTL register (Cont'd)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 29-30 | SDBN  | SDRAM number of banks.  
|       |       | Specifies the number of banks the SDRAM contains.  
|       |       | 00= 2 banks  
|       |       | 01= 4 banks  
|       |       | 1x= reserved  |
| 31    | SDPSS | Start SDRAM power-up sequence.  
|       |       | Write 1 to initiate power-up sequence. This bit always reads as 0.  

1 MSx is the external memory bank to which the SDRAM connects. If SBDS=000, indicating no SDRAM in use, the processor does not Hi-Z any of the MSx signals.
IOSTAT
Programmable I/O Status Register

The IOSTAT register provides status information on the general-purpose, programmable I/O ports, FLAG_{11-4} only.

For details on using the IOSTAT register, in *ADSP-21065L SHARC DSP User’s Manual* see:

- Chapter 3, Program Sequencing
- Chapter 11, Programmable Timers and I/O Ports

In this manual, see Appendix A, Instruction Set Reference.

The IOSTAT register is memory-mapped in internal memory at address 0x002F.

After reset, the IOSTAT register is initialized to 0x0000 0000 as shown in Figure E-9 on page E-76.
Figure E-9. IOSTAT register bits

In Table E-20, bits 0–7 are the flag pin values on the IOSTAT register.

Table E-20. Flag Pin Values on the IOSTAT register

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>FLG4</td>
<td>Status of the FLAG4 I/O port.</td>
</tr>
<tr>
<td>1</td>
<td>FLG5</td>
<td>Status of the FLAG5 I/O port.</td>
</tr>
<tr>
<td>2</td>
<td>FLG6</td>
<td>Status of the FLAG6 I/O port.</td>
</tr>
<tr>
<td>3</td>
<td>FLG7</td>
<td>Status of the FLAG7 I/O port.</td>
</tr>
<tr>
<td>4</td>
<td>FLG8</td>
<td>Status of the FLAG8 I/O port.</td>
</tr>
</tbody>
</table>
### Control and Status Registers

Table E-20. Flag Pin Values on the IOSTAT register (Cont’d)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>FLG9</td>
<td>Status of the FLAG9 I/O port.</td>
</tr>
<tr>
<td>6</td>
<td>FLG10</td>
<td>Status of the FLAG10 I/O port.</td>
</tr>
<tr>
<td>7</td>
<td>FLG11</td>
<td>Status of the FLAG11 I/O port.</td>
</tr>
<tr>
<td>8-31</td>
<td>Reserved</td>
<td></td>
</tr>
</tbody>
</table>
IOP Registers

RDIVx/TDIVx
SPORT Divisor Registers

The TDIV0, TDIV1, RDIV0, and RDIV1 registers contain divisor values that determine the frequencies for internally generated serial port clocks and frame syncs. Figure E-10 on page E-79 shows the RDIVx register bits and Figure E-11 on page E-79 shows the TDIVx register bits.

For details on using the RDIVx and TDIVx registers, in ADSP-21065L SHARC DSP User's Manual see:

- Chapter 3, Program Sequencing
- Chapter 9, Serial Ports

In this manual, see Appendix A, Instruction Set Reference.

These four registers are memory-mapped in internal memory at addresses 0x00E4, 0x00F4, 0x00E6, and 0x00F6, respectively.

These registers are not initialized after reset.
Control and Status Registers

Figure E-10. RDIVx register bits

Figure E-11. TDIVx register bits
IOP Registers

Tables E-22 and E-21 list and describe the individual bits of the RDIVx and TDIVx registers.

Table E-21. RDIVx bits

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-0</td>
<td>RCLKDIV</td>
<td>Recv clock divisor</td>
</tr>
<tr>
<td>31-16</td>
<td>RFSDIV</td>
<td>Recv frame sync divisor</td>
</tr>
</tbody>
</table>

Table E-22. TDIVx bits

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-0</td>
<td>TCLKDIV</td>
<td>Xmit clock divisor</td>
</tr>
<tr>
<td>31-16</td>
<td>TFSDIV</td>
<td>Xmit frame sync divisor</td>
</tr>
</tbody>
</table>

$$x_{CLKDIV} = \frac{2 \times f_{CLKIN}}{\text{serial clock frequency}} - 1$$

$$x_{FSDIV} = \frac{\text{serial clock frequency}}{\text{frame sync frequency}} - 1$$
SRCTLx
SPORT Receive Control Register

SRCTL0 and SRCTL1 are the receive control registers for SPORT0 and SPORT1 respectively.

For details on using the SRCTLx register, in ADSP-21065L SHARC DSP User’s Manual see Chapter 9, Serial Ports.

In this manual, see Appendix A, Instruction Set Reference.

SRCTL0 is memory-mapped at address 0x00E1, and SRCTL1 is memory-mapped at address 0x00F1.

After reset, these registers are initialized to 0x0000 0000 as shown in figures E-12, E-13, and E-14. When changing operating modes, make sure you write all zeros (0) to the serial port’s control register to clear it before writing the new mode.

Some bit definitions of the SRCTLx register depend on the mode (standard, I^2S, or multichannel) for which the serial port is configured.
IOP Registers

Figure E-12. SRCTLx register bits—standard mode

---

E-82 ADSP-21065L SHARC DSP Technical Reference
Control and Status Registers

Figure E-13. SRCTLx register bits—\(I^2S\) mode
Figure E-14. SRCTLx register bits—multichannel mode
Control and Status Registers

Table E-23 lists and describes the individual bits in the SRCTLx register.

Table E-23. SRCTLx bits

<table>
<thead>
<tr>
<th>Bit</th>
<th>Standard</th>
<th>I²S</th>
<th>Multichn.</th>
<th>Description</th>
</tr>
</thead>
</table>
| 0   | SPEN_A   | SPEN_A | Reserved  | SPORT enable A.  
|     |          |       |           | 0= disable  
|     |          |       |           | 1= enable    |
| 1-2 | DTYPE 1:0| Reserved | DTYPE 1:0 | Data type.  
|     |          |         |           | 00=right-justify; fill  
|     |          |         |           | MSBs w/0s  
|     |          |         |           | 01=right-justify;  
|     |          |         |           | sign-extend MSBs  
|     |          |         |           | 10=compand with µ-law  
|     |          |         |           | 11=compand with A-law |
| 3   | SENDN    | Reserved | SENDN  | Endian word format.  
|     |          |         |           | 0= MSB first  
|     |          |         |           | 1= LSB first |
| 4-8 | SLEN 4:0 | SLEN 4:0 | SLEN 4:0 | Serial word length -1 |
| 9   | PACK     | PACK   | PACK     | 16- to 32-bit word  
|     |          |         |           | packing.  
|     |          |         |           | 0= disable packing  
|     |          |         |           | 1= enable packing |
## IOP Registers

Table E-23. SRCTLx bits (Cont’d)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Standard</th>
<th>I²S</th>
<th>Multichn.</th>
<th>Description</th>
</tr>
</thead>
</table>
| 10  | ICLK     | MSTR| ICLK      | Receive clock source (ICLK).  
0= externally generated  
1= internally generated  
Master/slave mode (MSTR).  
0= RX is slave  
1= RX is master |
| 11  | OPMODE   | OPMODE| OPMODE  | SPORT operation mode.  
0= non-I²S mode  
1= I²S mode |
| 12  | CKRE     | Reserved| CKRE  | Active clock edge for data and frame sync sampling.  
0= falling edge  
1= rising edge |
| 13  | RFSR     | Reserved| Reserved| Receive frame sync requirement.  
0= no RFS required  
1= RFS required |
| 14  | IRFS     | Reserved| IRFS  | RFS source.  
0= externally generated  
1= internally generated |
## Control and Status Registers

Table E-23. SRCTLx bits (Cont'd)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Standard</th>
<th>I²S</th>
<th>Multichn.</th>
<th>Description</th>
</tr>
</thead>
</table>
| 15  | Reserved | Reserved | IMODE | Receive compare.  
0= disable  
1= enable |
| 16  | LRFS | L_FIRST | LRFS | Active state TFS (LRFS).  
0= active high  
1= active low  
Receive first channel (L_FIRST).  
0= right channel first  
1= left channel first |
| 17  | LAFS | Reserved | Reserved | RFS timing.  
0= early RFS  
1= late RFS |
| 18  | SDEN_A | SDEN_A | SDEN_A | SPORT receive DMA enable A.  
0= disable  
1= enable |
| 19  | SCHEN_A | SCHEN_A | SCHEN_A | SPORT receive chaining enable A.  
0= disable  
1= enable |
## IOP Registers

### Table E-23. SRCTLx bits (Cont’d)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Standard</th>
<th>I²S</th>
<th>Multichn.</th>
<th>Description</th>
</tr>
</thead>
</table>
| 20  | SDEN_B   | SDEN_B | IMAT      | SPORT receive DMA enable B (SDEN_B).  
|     |          |       |           | 0= disable  
|     |          |       |           | 1= enable  
|     |          |       |           | Receive compare data (IMAT).  
|     |          |       |           | 0= accept if false  
|     |          |       |           | 1= accept if true |
| 21  | SCHEN_B  | SCHEN_B | Reserved  | SPORT receive DMA chaining enable B. |
| 22  | SPL      | SPL   | Reserved  | SPORT loopback mode.  
|     |          |       |           | 0= disable  
|     |          |       |           | 1= enable |
| 23  | MCE      | Reserved | MCE      | SPORT mode.  
|     |          |       |           | 0= standard mode  
|     |          |       |           | 1= multichannel mode |
| 24  | SPEN_B   | SPEN_B | NCH₀      | SPORT enable B (SPEN_B).  
|     |          |       |           | 0= disable  
|     |          |       |           | 1= enable  
|     |          |       |           | Number of channel slots -1 (NCH). |
| 25  | Reserved | Reserved | NCH₁      | Number of channel slots -1. |
### Control and Status Registers

#### Table E-23. SRCTLx bits (Cont’d)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Standard</th>
<th>$I^2S$</th>
<th>Multichn.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>ROVF_B</td>
<td>ROVF_B</td>
<td>NCH$_2$</td>
<td>RX_B overflow status (ROVF_B). Read-only, sticky status bit. Number of channel slots -1 (NCH).</td>
</tr>
<tr>
<td>27-28</td>
<td>RXS_B$_1:0$</td>
<td>RXS_B$_1:0$</td>
<td>NCH$_3:4$</td>
<td>RX_B data buffer status (RXS_B). Read-only. 00=empty 10=partially full 11=full Number of channel slots -1 (NCH).</td>
</tr>
<tr>
<td>29</td>
<td>ROVF_A</td>
<td>ROVF_A</td>
<td>ROVF_A</td>
<td>RX_A overflow status. Read-only, sticky status bit.</td>
</tr>
<tr>
<td>30-31</td>
<td>RXS_A$_1:0$</td>
<td>RXS_A$_1:0$</td>
<td>RXS_A$_1:0$</td>
<td>RX_A data buffer status. Read-only. 00=empty 10=partially full 11=full</td>
</tr>
</tbody>
</table>
STCTLx
SPORT Transmit Control Register

STCTL0 and STCTL1 are the transmit control registers for SPORT0 and SPORT1 respectively.

For details on using the STCTLx register, in ADSP-21065L SHARC DSP User’s Manual see Chapter 9, Serial Ports.

In this manual, see Appendix A, Instruction Set Reference.

STCTL0 is memory-mapped at address 0x00E0, and STCTL1 is memory-mapped at address 0x00F0.

After reset, these registers are initialized to 0x0000 0000 as shown in figures E-15, E-16, and E-17.

When changing operating modes, make sure to write all zeros (0) to the serial port’s control register to clear it before writing the new mode.

Some bit definitions of the STCTLx register depend on the mode (standard, I²S, or multichannel) for which the serial port is configured.
Control and Status Registers

Figure E-15. STCTLx register bits—standard mode

* Status is read-only
Figure E-16. STCTLx register bits—I^2S mode

* Status is read-only
Control and Status Registers

Figure E-17. STCTLx register bits—multichannel mode
Table E-24 lists and describes the individual bits of the STCTLx register.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Standard</th>
<th>I²S</th>
<th>Multichn.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SPEN_A</td>
<td>SPEN_A</td>
<td>Reserved</td>
<td>SPORT enable A.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0= disable</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1= enable</td>
</tr>
<tr>
<td>1-2</td>
<td>DTYPE_{1:0}</td>
<td>Reserved</td>
<td>DTYPE_{1:0}</td>
<td>Data type.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>00= right-justify; fill MSB s w/0s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>01= right-justify; sign-extend MSBs</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10= compand w/µ-law</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>11= compand w/A-law</td>
</tr>
<tr>
<td>3</td>
<td>SENDN</td>
<td>Reserved</td>
<td>SENDN</td>
<td>Endian word format.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0= MSB first</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1= LSB first</td>
</tr>
<tr>
<td>4-8</td>
<td>SLEN_{4:0}</td>
<td>SLEN_{4:0}</td>
<td>SLEN_{4:0}</td>
<td>Serial word length -1</td>
</tr>
<tr>
<td>9</td>
<td>PACK</td>
<td>PACK</td>
<td>PACK</td>
<td>16- to 32-bit word packing.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0= disable</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1= enable</td>
</tr>
</tbody>
</table>
## Control and Status Registers

Table E-24. STCTLx bits (Cont’d)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Standard</th>
<th>i2S</th>
<th>Multichn.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>ICLK</td>
<td>MSTR</td>
<td>Reserved</td>
<td>Transmit clock source (ICLK).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0 = external clock</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1 = internal clock</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Master/slave mode (MSTR).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0 = Tx slave</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1 = Tx master</td>
</tr>
<tr>
<td>11</td>
<td>OPMODE</td>
<td>OPMODE</td>
<td>OPMODE</td>
<td>Operation mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0 = non-I2S mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1 = I2S mode</td>
</tr>
<tr>
<td>12</td>
<td>CKRE</td>
<td>Reserved</td>
<td>CKRE</td>
<td>Active clock edge for data and frame sync</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>sampling.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0 = falling edge</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1 = rising edge</td>
</tr>
<tr>
<td>13</td>
<td>TFSR</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Transmit TFS requirement.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0 = not required</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1 = required</td>
</tr>
<tr>
<td>14</td>
<td>ITFS</td>
<td>Reserved</td>
<td>Reserved</td>
<td>TFS source.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0 = external</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1 = internal</td>
</tr>
</tbody>
</table>
### IOP Registers

#### Table E-24. STCTLx bits  (Cont’d)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Standard</th>
<th>I²S</th>
<th>Multichn.</th>
<th>Description</th>
</tr>
</thead>
</table>
| 15  | DITFS    | DITFS | DITFS     | TFS data dependency.  
0= data-dependent  
1= data-independent |
| 16  | LTFS     | L-FIRST | LTFS     | Active low TFS (LTFS).  
0= active high  
1= active low  
First transmit channel select (L_FIRST).  
0= right channel first  
1= left channel first |
| 17  | LAFS     | Reserved | Reserved | TFS timing.  
0= early TFS  
1= late TFS |
| 18  | SDEN_A   | SDEN_A | SDEN_A    | SPORT transmit DMA enable A.  
0= disable  
1= enable |
| 19  | SCHEN_A  | SCHEN_A | SCHEN_A  | SPORT transmit DMA chaining enable A.  
0= disable  
1= enable |
### Control and Status Registers

Table E-24. STCTLx bits (Cont'd)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Standard</th>
<th>i²S</th>
<th>Multichn.</th>
<th>Description</th>
</tr>
</thead>
</table>
| 20  | SDEN_B   | SDEN_B | MFD<sub>0</sub> | SPORT transmit DMA enable B (SDEN_B).  
0= disable  
1= enable  
Multichannel frame delay (MFD). |
| 21  | SCHEN_B  | SCHEN_B | MFD<sub>1</sub> | SPORT transmit DMA chaining enable B (SCHEN_B).  
0= disable  
1= enable  
Multichannel frame delay (MFD). |
| 22  | FS_BOTH  | FS_BOTH | MFD<sub>2</sub> | Word select.  
0= issue if data in either TX buffer  
1= issue only if data in both TX buffers  
Multichannel frame delay (MFD). |
| 23  | Reserved | Reserved | MFD<sub>3</sub> | Multichannel frame delay. |
| 24  | SPEN_B   | SPEN_B | CHNL<sub>0</sub> | SPORT enable B (SPEN_B).  
Currently selected channel (CHNL).  
Read-only, sticky status bits (values 0-31). |
## IOP Registers

Table E-24. STCTLx bits  (Cont’d)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Standard</th>
<th>I²S</th>
<th>Multichn.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>Reserved</td>
<td>Reserved</td>
<td>CHNL₁</td>
<td>Currently selected channel (CHNL). Read-only.</td>
</tr>
<tr>
<td>26</td>
<td>TUVF_B</td>
<td>TUVF_B</td>
<td>CHNL₂</td>
<td>TX_B underflow (TUVF_B). Read-only, sticky status bit. Currently selected channel (CHNL). Read-only.</td>
</tr>
<tr>
<td>27-28</td>
<td>TXS_B</td>
<td>TXS_B</td>
<td>CHNL₃:₄</td>
<td>TXS_B data buffer status (TXS_B). Read-only, sticky bit. 00= empty 10= partially full 11= full Currently selected channel (CHNL). Read-only.</td>
</tr>
<tr>
<td>29</td>
<td>TUVF_A</td>
<td>TUVF_A</td>
<td>TUVF_A</td>
<td>TX_A underflow (TUV_A). Read-only, sticky status bit.</td>
</tr>
<tr>
<td>30-31</td>
<td>TXS_A</td>
<td>TXS_A</td>
<td>TXS_A</td>
<td>TX_A data buffer status (TXS_A). Read-only, sticky bit.</td>
</tr>
</tbody>
</table>
SYSCON
System Configuration Register

Applications use the SYSCON register to program system configuration settings.

For details on using the SYSCON register, in *ADSP-21065L SHARC DSP User’s Manual* see:

- Chapter 7, Multiprocessing
- Chapter 8, Host Interface

In this manual, see Appendix A, Instruction Set Reference.

The SYSCON register is memory-mapped in internal memory at address 0x0000.

After reset the SYSCON register is initialized to 0x0000 0020 as shown in Figure E-18 on page E-100.

Initialization causes the ADSP-21065L to assume an 8-bit bus for any host processor. To change the value of the HBW bits, applications must write four 8-bit words to SYSCON (in the HBW bits), even if the host bus is 16- or 32-bits wide.
Figure E-18. SYSCON register bits
Table E-25 lists and describes the individual bits of the SYSCON register.

Table E-25. SYSCON register

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SRST</td>
<td>Software reset. Causes a software reset. Has the same effect as the RESET pin.</td>
</tr>
</tbody>
</table>
| 1   | BSO  | Boot select override.  
     |                  | 1=Activate BMS to read from boot EPROM  
     |                  | Activated only during external port DMA transfers.  
     |                  | Deactivates the BSEL lines. Enables processor to read its boot EPROM when no longer in boot mode and to read additional code or data from its EPROM after completing booting. |
| 2   | IIVT | Internal interrupt vector table (no boot mode—BSEL =0, BMS=0).  
     |                  | Specifies the location of the interrupt vector table when processor configured for “no boot” mode.  
     |                  | 0= in external memory at 0x0002 0000.  
     |                  | 1= in internal memory at 0x0000 8000.  
     |                  | After reset, initialized to zero, placing the interrupt vector table in external memory for “no boot” mode.  
     |                  | When the processor is configured for one of the boot modes, the internal interrupt vector table always resides in internal memory, regardless of the value of this bit. |
| 3   | Reserved | |

ADSP-21065L SHARC DSP Technical Reference E-101
## IOP Registers

### Table E-25. SYSCON register (Cont’d)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 4-5 | HBW   | Host bus width.  
      |       | Specifies the external word width of the host  
      |       | bus for host accesses to the processor’s EPBx  
      |       | IOP registers.  
      |       | 00 = 32 bit host bus  
      |       | 01 = 16-bit host bus  
      |       | 10 = 8-bit host bus  
      |       | 11 = reserved  
      |       | Host accesses to all other IOP registers are  
      |       | always 32 bits, regardless of the value of  
      |       | this bit. |
| 6   | HMSWF | Host packing order.  
      |       | Specifies the packing order for host  
      |       | accesses.  
      |       | 0 = LSW first  
      |       | 1 = MSW first  
      |       | This bit ignored for 32- to 48-bit packing. |
| 7   | HPFLSH| Host packing status flush.  
      |       | Resets the host packing status.  
      |       | 1 = flush packing status  
      |       | This bit always reads as 0.  
      |       | Host must not access the IOP registers while  
      |       | the core writes this bit. |
Control and Status Registers

Table E-25. SYSCON register (Cont’d)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 8   | IMDWO  | Internal memory block 0 data width.  
|     |        | Specifies the data word width of internal memory, block 0.  
|     |        | 0= 32-bit data  
|     |        | 1= 40-bit data  
|     |        | Applications can store 48-bit instructions in block 0 regardless of the value of this bit.  
|     |        | For details, see Chapter 5, Memory, in ADSP-21065L SHARC DSP User’s Manual. |
| 9   | IMDW1  | Internal memory block 1 data width.  
|     |        | Specifies the data word width of internal memory, block 1.  
|     |        | 0= 32-bit data  
|     |        | 1= 40-bit data  
|     |        | Applications can store 48-bit instructions in block 1 regardless of the value of this bit.  
|     |        | For details, see Chapter 5, Memory, in ADSP-21065L SHARC DSP User’s Manual. |
| 10  | ADREDY | Active drive REDY.  
|     |        | Changes the REDY signal to an active drive output.  
|     |        | 0= open drain (o/d)  
|     |        | 1= active drive (a/d) |
### Table E-25. SYSCON register (Cont’d)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>BHD</td>
<td>Buffer hang disable. Enables/disables the hang condition that occurs when the processor’s core or an external device tries to read an empty buffer or write a full buffer. 0 = enable buffer hang 1 = disabled buffer hang After reset, this bit is enabled. Disabling this bit is useful for debugging applications.</td>
</tr>
<tr>
<td>12-15</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>16-17</td>
<td>EBPR</td>
<td>External bus priority. Specifies which of the processor’s three internal buses (PM, DM, and I/O) has priority when accessing the external ADDR\textsubscript{23-0} and DATA\textsubscript{31-0} buses. The processor’s internal buses are multiplexed together at the external port. 00 = even priority, alternating core and I/O accesses 01 = processor’s core (PM and DM) buses 10 = I/O processor’s I/O bus Eliminates contention at the external port when both the processor’s core and IOP try to read or write off-chip during the same cycle Not related to the function of the CPA pin (core priority access).</td>
</tr>
</tbody>
</table>
Control and Status Registers

Table E-25. SYSCON register (Cont'd)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>DCPR</td>
<td>DMA channels 8 and 9 priority. Specifies how the processor prioritizes accesses of the external ADDR\textsubscript{23-0} and DATA\textsubscript{31-0} buses between DMA channels 8 and 9 when both attempt to read or write off-chip during the same cycle. 0=sequential Send entire block of data from one DMA channel before servicing the next one, starting with channel 8. 1=rotating Send one data word per cycle, alternating between each DMA channel, starting with channel 8.</td>
</tr>
<tr>
<td>19-31</td>
<td>Reserved</td>
<td></td>
</tr>
</tbody>
</table>
IOP Registers

**SYSTAT**

System Status Register

The SYSTAT register provides status information on system functions, primarily for multiprocessor systems.

For details on using the SYSTAT register, in *ADSP-21065L SHARC DSP User's Manual* see:

- Chapter 7, Multiprocessing
- Chapter 8, Host Interface

In this manual, see Appendix A, Instruction Set Reference.

The SYSTAT register is memory-mapped in internal memory at address 0x0003.

After reset, all bits in SYSTAT, except IDC (1:0) and CRBM (1:0), are initialized to zero (0) as shown in Figure E-19. After reset, IDC (1:0) is equal to the value of the processor’s ID_{1:0} inputs, and CRBM (1:0) is equal to the ID of the current bus master.
Control and Status Registers

Figure E-19. SYSTAT register bits
Table E-26 lists and describes the individual bits of the SYSTAT register.

### Table E-26. SYSTAT register

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 0   | HSTM | Host mastership.  
  Indicates whether or not the host processor is the current bus master.  
  0= bus slave  
  1= bus master |
| 1   | BSYN | Bus synchronization.  
  Indicates whether or not bus arbitration logic is synchronized.  
  0= unsynchronized  
  1= synchronized |
| 2-3 | Reserved | |
| 4-5 | CRBM | Current bus master.  
  Identifies the ID code of the ADSP-21065L that is the current bus master.  
  If CRBM = ID of this processor, this processor is the current bus master.  
  CRBM is valid only for ID2-0 > 0.  
  When ID2-0 = 000, CRBM is always 1. |
| 6-7 | Reserved | |
# Control and Status Registers

Table E-26. SYSTAT register (Cont’d)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-9</td>
<td>IDC</td>
<td>ID code (ID1:0) of the processor. Identifies the IDx code of this processor. 00=reserved for single-processor systems only 01=ID1 10=ID2 11=reserved</td>
</tr>
<tr>
<td>10-11</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>SWPD</td>
<td>Slave write pending data. Indicates whether valid data is pending in the slave write FIFO. 0= No data pending 1= Data pending Cleared after the processor transfers data in the slave write FIFO to the target IOP register. Set when the slave write FIFO receives new data.</td>
</tr>
<tr>
<td>13</td>
<td>VIPD</td>
<td>Vector interrupt pending. Indicates whether or not a vector interrupt is pending. 0= none pending 1= pending</td>
</tr>
</tbody>
</table>
### IOP Registers

Table E-26. SYSTAT register (Cont'd)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 14  | HPS  | Host packing status.  
Indicates the progress of the host access packing procedure.  
0= fully packed  
1= partially packed |
| 15-31 | Reserved | |
Control and Status Registers

WAIT

External Memory Wait State Control Register

Applications use the WAIT register to set up external memory wait states and the processor’s response to the ACK signal.

For details on using the WAIT register, in ADSP-21065L SHARC DSP User’s Manual see:

- Chapter 5, Memory
- Chapter 6, DMA
- Chapter 7, Multiprocessing
- Chapter 12, System Design

In this manual, see Appendix A, Instruction Set Reference.

The WAIT register is memory-mapped in internal memory at address 0x0002.

After reset, the WAIT register is initialized to 0x21AD 6B5A as shown in Figure E-20 on page E-112. This configures the processor for:

- Six internal wait states.
- Dependence on ACK for all external memory banks.
- Multiprocessor memory space wait state enabled.
### IOP Registers

Figure E-20. WAIT register bits
Table E-27 lists and describes the individual bits of the WAIT register.

Table E-27. WAIT register

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 0-1 | EB0WM | External bank 0 wait state mode  
00= external acknowledge only (ACK)  
01= internal wait states only  
10= both internal and external acknowledge required  
11= either internal or external acknowledge required |
| 2-4 | EBOWS | External bank 0 number of wait states.  
000=0 wait states; no bus idle cycle\(^1\); no hold time cycle\(^2\)  
001=1 wait state; a bus idle cycle; no hold time cycle  
010=2 wait states; a bus idle cycle; no hold time cycle  
011=3 wait states; a bus idle cycle; no hold time cycle  
100=4 wait states; no bus idle cycle; a hold time cycle  
101=5 wait states; no bus idle cycle; a hold time cycle  
110=6 wait states; no bus idle cycle; a hold time cycle  
111=0 wait states; a bus idle cycle; no hold time cycle |
| 5-6 | EB1WM | External bank 1 wait state mode.  
For parameter values, see EBOWM parameter on page E-113. |
Table E-27. WAIT register  (Cont’d)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 7-9 | EB1WS | External bank 1 number of wait states.  
For parameter values, see EBOWS parameter on page E-113. |
| 10-11 | EB2WM | External bank 2 wait state mode.  
For parameter values, see EBOWM parameter on page E-113. |
| 12-14 | EB2WS | External bank 2 number of wait states.  
For parameter values, see EBOWS parameter on page E-113. |
| 15-16 | EB3WM | External bank 3 wait state mode.  
For parameter values, see EBOWM parameter on page E-113. |
| 17-19 | EB3WS | External bank 3 number of wait states.  
For parameter values, see EBOWS parameter on page E-113. |
| 20-21 | RBWM | ROM boot wait mode.  
Controls the wait mode for accesses that use the BMS pin. See the BSO bit in Table E-25 on page E-101.  
For parameter values, see EBOWM parameter on page E-113. |
| 22-24 | RBWS | ROM boot wait state.  
Controls the wait state for accesses that use the BMS pin. See the BSO bit in Table E-25 on page E-101.  
For parameter values, see EBOWS parameter on page E-113. |
### Control and Status Registers

Table E-27. WAIT register (Cont'd)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>25-28</td>
<td>Reserved</td>
<td></td>
</tr>
</tbody>
</table>
| 29   | MMSWS | Multiprocessor memory space wait state.  
Single wait state for multiprocessor memory space accesses. |
| 30   | HIDMA | Handshake idle cycle for DMA.  
Single idle cycle for DMA handshake. |
| 31   | Reserved |             |

1. Bus idle cycle = an inactive bus cycle the processor automatically generates to avoid bus driving conflicts. For devices with slow disable time, enable bus idle cycle generation with EBxWS parameter. Does not apply to SDRAM accesses.

2. Hold time cycle = an inactive bus cycle the processor automatically generates at the end of a read or write operation to provide a longer hold time for address and data. When enabled, the address and data remain unchanged and driven for one cycle after the read or write strobes are deasserted. Does not apply to SDRAM accesses.

Both the bus idle cycle and the hold time cycle occur if programmed, regardless of the wait state mode. For example, the ACK-only wait state mode can have a hold time cycle programmed for it.
To program the IOP registers, you write to the appropriate address in memory. You can use the symbolic names of the registers and individual bits in your application software—the file `def21065L.h`, which is provided in the INCLUDE directory of the ADSP-21000 Family Development Software—contains the `#define` definitions for these symbols.

Listing E.6 is the `def21065L.h` file, provided here for reference.

**Listing E.6. def21065L.h**

```c
/*————————————————————————————————————————*/
def21065L.h—SYSTEM AND IOP REGISTER BIT AND ADDRESS DEFINITIONS FOR ADSP-21065L

Last Modification on: June 26, 1998

This include file contains a list of macro defines to enable the programmer to use symbolic names for all of the system register bits for the ADSP-21065L. It also contains macros for the IOP register addresses and some bit fields.

/*————————————————————————————————————————*/
/* MODE1 register */
#define BR8 0x00000001 /* Bit 0: Bit-reverse for I8 */
#define BR0 0x00000002 /* Bit 1: Bit-reverse for I0 (uses DMS0-only) */
#define SRCU 0x00000004 /* Bit 2: Alt. reg. select for comp. units */
#define SRD1H 0x00000008 /* Bit 3: DAG1 alt. register select (7-4) */
#define SRD1L 0x00000010 /* Bit 4: DAG1 alt. register select (3-0) */
#define SRD2H 0x00000020 /* Bit 5: DAG2 alt. reg. select (15-12) */
#define SRD2L 0x00000040 /* Bit 6: DAG2 alt. register select (11-8) */
#define SRRFH 0x00000080 /* Bit 7: Reg. File alt. select - R(15-8) */
#define SRRFL 0x00000400 /* Bit 8: Reg. File alt. select - R(7-0) */
#define NESTM 0x00000800 /* Bit 11: Interrupt nesting enable */
#define IRPTEN 0x00001000 /* Bit 12: Global interrupt enable */
#define ALUSAT 0x00002000 /* Bit 13: Enable ALU fixed-pt. saturation */
#define SSE 0x00004000 /* Bit 14: Enable short word sign exten. */
#define TRUNC 0x00008000 /* Bit 15: 1=flt-pt. trunc. 0=Rnd to near */
#define RND32 0x00010000 /* Bit 16: 1=32b flt-pt.round. 0=40b rnd */
#define CSEL 0x00060000 /* Bit 17-18: CSelect: Bus Mastership */
```
Control and Status Registers

/* MODE2 register */
#define IRQ0E 0x00000001 /* Bit 0: IRQ0- 1=edge sens. 0=level sens. */
#define IRQ1E 0x00000002 /* Bit 1: IRQ1- 1=edge sens. 0=level sens. */
#define IRQ2E 0x00000004 /* Bit 2: IRQ2- 1=edge sens. 0=level sens. */
#define PERIOD_CNT0 0x00000008 /* Bit 3: Enable Period Count */
#define CADIS 0x00000010 /* Bit 4: Cache disable */
#define TIMEN0 0x00000020 /* Bit 5: Timer0 enable */
#define BUSLK 0x00000040 /* Bit 6: External bus lock */
#define PWMOUT0 0x00000080 /* Bit 7: PWMOUT/WIDTH_CNT control-Timer0 */
#define INT_HI0 0x00000100 /* Bit 8: Interrupt Vector location */
#define PULSE_HI0 0x00000200 /* Bit 9: Pulse transition edge select */
#define PERIOD_CNT1 0x00000400 /* Bit 10: Enable Period Count */
#define TIMEN1 0x00000800 /* Bit 11: Timer0 enable */
#define PWMOUT1 0x00001000 /* Bit 12: PWMOUT/WIDTH_CNT ctrl-Timer1 */
#define INT_HI1 0x00002000 /* Bit 13: Interrupt Vector location */
#define PULSE_HI1 0x00004000 /* Bit 14: Pulse transition edge select */
#define FLG0O 0x00008000 /* Bit 15: FLAG0 1=output 0=input */
#define FLG1O 0x00010000 /* Bit 16: FLAG1 1=output 0=input */
#define FLG2O 0x00020000 /* Bit 17: FLAG2 1=output 0=input */
#define FLG3O 0x00040000 /* Bit 18: FLAG3 1=output 0=input */
#define CAFRZ 0x00080000 /* Bit 19: Cache freeze */

/* ASTAT register */
#define AZ 0x00000001 /* Bit 0: ALU result 0 or flt-pt. undrflw */
#define AV 0x00000002 /* Bit 1: ALU overflow */
#define AN 0x00000004 /* Bit 2: ALU result negative */
#define AC 0x00000008 /* Bit 3: ALU fixed-pt. carry */
#define AS 0x00000010 /* Bit 4: ALU X input sign (ABS & MANT ops) */
#define AI 0x00000020 /* Bit 5: ALU fltg-pt. invalid operation */
#define MN 0x00000040 /* Bit 6: Multiplier result negative */
#define MV 0x00000080 /* Bit 7: Multiplier overflow */
#define MU 0x00000100 /* Bit 8: Multiplier flt-pt. underflow */
#define MI 0x00000200 /* Bit 9: Multiplier flt-pt. invalid op. */
#define AF 0x00000400 /* Bit 10: ALU fltg-pt. op. */
#define SV 0x00000800 /* Bit 11: Shifter overflow */
#define SZ 0x00001000 /* Bit 12: Shifter result zero */
#define SS 0x00002000 /* Bit 13: Shifter input sign */
#define BTF 0x00040000 /* Bit 18: Bit test flag for system regs. */
#define FLG0 0x00080000 /* Bit 19: FLAG0 value */
SYMBOL DEFINITIONS FILE (def21065L.h)

#define FLG1 0x00100000 /* Bit 20: FLAG1 value */
#define FLG2 0x00200000 /* Bit 21: FLAG2 value */
#define FLG3 0x00400000 /* Bit 22: FLAG3 value */
#define CACC0 0x01000000 /* Bit 24: Compare Accumulation Bit 0 */
#define CACC1 0x02000000 /* Bit 25: Compare Accumulation Bit 1 */
#define CACC2 0x04000000 /* Bit 26: Compare Accumulation Bit 2 */
#define CACC3 0x08000000 /* Bit 27: Compare Accumulation Bit 3 */
#define CACC4 0x10000000 /* Bit 28: Compare Accumulation Bit 4 */
#define CACC5 0x20000000 /* Bit 29: Compare Accumulation Bit 5 */
#define CACC6 0x40000000 /* Bit 30: Compare Accumulation Bit 6 */
#define CACC7 0x80000000 /* Bit 31: Compare Accumulation Bit 7 */

/* STKY register */
#define AUS 0x00000001 /* Bit 0: ALU flt-pt. underflow */
#define AVS 0x00000002 /* Bit 1: ALU flt-pt. overflow */
#define AOS 0x00000004 /* Bit 2: ALU fixed-pt. overflow */
#define AIS 0x00000020 /* Bit 5: ALU flt-pt. invalid operation */
#define MOS 0x00000040 /* Bit 6: Multiplier fixed-pt. overflow */
#define MVS 0x00000080 /* Bit 7: Multiplier flt-pt. overflow */
#define MUS 0x00000100 /* Bit 8: Multiplier flt-pt. underflow */
#define MIS 0x00000200 /* Bit 9: Multiplier flt-pt. invalid op. */
#define CB7S 0x00020000 /* Bit 17: DAG1 circular buffer 7 overflow */
#define CB15S 0x00040000 /* Bit 18: DAG2 circular buffer 15 overflow */
#define PCFL 0x00200000 /* Bit 21: PC stack full */
#define PCEM 0x00400000 /* Bit 22: PC stack empty */
#define SSOV 0x00800000 /* Bit 23: Status stack overflow (MODE1&ASTAT) */
#define SSEM 0x01000000 /* Bit 24: Status stack empty */
#define LSOV 0x02000000 /* Bit 25: Loop stack overflow */
#define LSEM 0x04000000 /* Bit 26: Loop stack empty */

/* IRPTL and IMASK and IMASKP registers */
#define RSTI 0x00000002 /* Bit 1: Offset: 04: Reset */
#define SOVFI 0x00000008 /* Bit 3: Offset: 0c: Stack overflow */
#define TMZHI 0x00000010 /* Bit 4: Offset: 10: Timer=0 (high prir.) */
#define VIRPTI 0x00000020 /* Bit 5: Offset: 14: Vector interrupt */
#define IRQ2I 0x00000040 /* Bit 6: Offset: 18: IRQ2- asserted */
#define IRQ1I 0x00000080 /* Bit 7: Offset: 1c: IRQ1- asserted */
#define IRQ0I 0x00000100 /* Bit 8: Offset: 20: IRQ0- asserted */
#define SPR0I 0x00000400 /* Bit 10: Offset: 28: SPORT0 receive */
#define SPR1I 0x00000800 /* Bit 11: Offset: 30: SPORT0 transmit */
#define SPT0I 0x00001000 /* Bit 12: Offset: 34: SPORT1 receive */
#define SPT1I 0x00002000 /* Bit 13: Offset: 38: SPORT1 transmit */
#define EP0I 0x00010000 /* Bit 16: Offset: 40: Ext. port chn 0 DMA */
#define EP1I 0x00020000 /* Bit 17: Offset: 44: Ext. port chn 1 DMA */
Control and Status Registers

#define CB7I 0x00200000 /* Bit 21: Offset: 54: Cir. buff 7 ovrflw */
#define CB15I 0x00400000 /* Bit 22: Offset: 58: Cir. buff 15 ovrflw */
#define TMZLI 0x00800000 /* Bit 23: Offset: 5c: Timer=0 (low prir.) */
#define FIXI 0x01000000 /* Bit 24: Offset: 60: Fixed-pt. overflow */
#define FLTOI 0x02000000 /* Bit 25: Offset: 64: fltg-pt. overflow */
#define FLTUI 0x04000000 /* Bit 26: Offset: 68: fltg-pt. underflow */
#define FLTII 0x08000000 /* Bit 27: Offset: 6c: fltg-pt. invalid */
#define SFT0I 0x10000000 /* Bit 28: Offset: 70: user software int 0 */
#define SFT1I 0x20000000 /* Bit 29: Offset: 74: user software int 1 */
#define SFT2I 0x40000000 /* Bit 30: Offset: 78: user software int 2 */
#define SFT3I 0x80000000 /* Bit 31: Offset: 7c: user software int 3 */

/* SYSCON Register */
#define SYSCON 0x00 /* Memory mapped System Config. Reg. */
#define SRST 0x00000001 /* Soft Reset */
#define BSO 0x00000002 /* Boot Select Override */
#define IIVT 0x00000004 /* Internal Interrupt Vector Table */
#define HBW00 0x00000000 /* Host Bus Width: 32bit */
#define HBW01 0x00000010 /* Host Bus Width: 16bit */
#define HBW10 0x00000020 /* Host Bus Width: 8bit */
#define HMSWF 0x00000040 /* Host packing order (0=LSW first, 1=MSW) */
#define HPFLSH 0x00000080 /* Host pack flush */
#define IMDW0X 0x00000100 /* Int. memory blk0, extended data (40b) */
#define IMDW1X 0x00000200 /* Int. memory blk1, extended data (40b) */
#define EBPR00 0x00000000 /* Ext. bus priority: Even */
#define EBPR01 0x00010000 /* Ext. bus priority: Core has priority */
#define EBPR10 0x00020000 /* Ext. bus priority: IO has priority */
#define DCPR 0x00040000 /* Sel. rotating access prir. - DMA8-DMA9 */

/* SYSTAT Register */
#define SYSTAT 0x03 /* Memory mapped System Status Register */
#define HSTM 0x00000001 /* Host is the Bus Master */
#define BSYN 0x00000002 /* Bus arbitration logic is synchronized */
#define CRBM 0x00000030 /* Current ADSP21065L Bus Master */
#define IDC 0x00000300 /* ADSP21065L ID Code */
#define SWPD 0x00001000 /* Slave write FIFO data pending */
#define VIPD 0x00002000 /* Vector interrupt pending (1 = pending) */
#define HPS 0x00004000 /* Host pack status */

/* SYSTEM registers */
#define SYSCON 0x00 /* System configuration register */
#define VIRPT 0x01 /* Vector interrupt table */
#define WAIT 0x02 /* Wait state config. for ext. memory */
#define SYSTAT 0x03 /* System status register */
SYMBOL DEFINITIONS FILE (def21065L.h)

DMA BUFFER registers

#define EPB0 0x04 /* External port DMA buffer 0 */
#define EPB1 0x05 /* External port DMA buffer 1 */

MESSAGE registers

#define MSGR0 0x08 /* Message register 0 */
#define MSGR1 0x09 /* Message register 1 */
#define MSGR2 0x0a /* Message register 2 */
#define MSGR3 0x0b /* Message register 3 */
#define MSGR4 0x0c /* Message register 4 */
#define MSGR5 0x0d /* Message register 5 */
#define MSGR6 0x0e /* Message register 6 */
#define MSGR7 0x0f /* Message register 7 */

MISCELLANEOUS registers

#define BMAX 0x18 /* Bus timeout maximum */
#define BCNT 0x19 /* Bus timeout counter */

DMAC registers

#define DMAC0 0x1c /* DMA 8 control register */
#define DMAC1 0x1d /* DMA 9 control register */

SDRAM & Timer registers

#define SDRDIV 0x20 /* SDRAM refresh counter specification */
#define TPERIOD0 0x28 /* Timer 0 period register */
#define TPWIDTH0 0x29 /* Timer 0 pulse width register */
#define TCOUNT0 0x2a /* Timer 0 counter */
#define TPERIOD1 0x2b /* Timer 1 period register */
#define TPWIDTH1 0x2c /* Timer 1 pulse width register */
#define TCOUNT1 0x2d /* Timer 1 counter */
#define IOCTL 0x2e /* SDRAM and gen. purpose I/O control reg. */
#define IOSTAT 0x2f /* Gen. purpose I/O status register */

DMA ADDRESS registers

#define IIR0A 0x60 /* DMA channel 0 index reg. */
#define IMR0A 0x61 /* DMA channel 0 modify reg. */
#define CROA 0x62 /* DMA channel 0 count reg. */
#define CPR0A 0x63 /* DMA channel 0 chain pointer reg. */
#define GPR0A 0x64 /* DMA channel 0 general purpose reg. */
Control and Status Registers

#define IIR0B 0x30 /* DMA channel 1 index reg. */
#define IMR0B 0x31 /* DMA channel 1 modify reg. */
#define CR0B 0x32 /* DMA channel 1 count reg. */
#define CPR0B 0x33 /* DMA channel 1 chain pointer reg. */
#define GPR0B 0x34 /* DMA channel 1 general purpose reg. */
#define DMASTAT 0x37 /* DMA channel status register */

#define IIR1A 0x68 /* DMA channel 2 index reg. */
#define IMR1A 0x69 /* DMA channel 2 modify reg. */
#define CR1A 0x6A /* DMA channel 2 count reg. */
#define CPR1A 0x6B /* DMA channel 2 chain pointer reg. */
#define GPR1A 0x6C /* DMA channel 2 general purpose reg. */

#define IIR1B 0x38 /* DMA channel 3 index reg. */
#define IMR1B 0x39 /* DMA channel 3 modify reg. */
#define CR1B 0x3A /* DMA channel 3 count reg. */
#define CPR1B 0x3B /* DMA channel 3 chain pointer reg. */
#define GPR1B 0x3C /* DMA channel 3 general purpose reg. */

#define IIT0A 0x70 /* DMA channel 4 index reg. */
#define IMT0A 0x71 /* DMA channel 4 modify reg. */
#define CT0A 0x72 /* DMA channel 4 count reg. */
#define CPT0A 0x73 /* DMA channel 4 chain pointer reg. */
#define GPT0A 0x74 /* DMA channel 4 general purpose reg. */

#define IIT0B 0x50 /* DMA channel 5 index reg. */
#define IMT0B 0x51 /* DMA channel 5 modify reg. */
#define CT0B 0x52 /* DMA channel 5 count reg. */
#define CPT0B 0x53 /* DMA channel 5 chain pointer reg. */
#define GPT0B 0x54 /* DMA channel 5 general purpose reg. */

#define IIT1A 0x78 /* DMA channel 6 index reg. */
#define IMT1A 0x79 /* DMA channel 6 modify reg. */
#define CT1A 0x7A /* DMA channel 6 count reg. */
#define CPT1A 0x7B /* DMA channel 6 chain pointer reg. */
#define GPT1A 0x7C /* DMA channel 6 general purpose reg. */

#define IIT1B 0x58 /* DMA channel 7 index reg. */
#define IMT1B 0x59 /* DMA channel 7 modify reg. */
#define CT1B 0x5A /* DMA channel 7 count reg. */
#define CPT1B 0x5B /* DMA channel 7 chain pointer reg. */
#define GPT1B 0x5C /* DMA channel 7 general purpose reg. */
#define IIEP0 0x40 /* DMA channel 8 index reg. */
#define IMEP0 0x41 /* DMA channel 8 modify reg. */
#define CEP0 0x42 /* DMA channel 8 count reg. */
#define CPEP0 0x43 /* DMA channel 8 chain pointer reg. */
#define GPEP0 0x44 /* DMA channel 8 general purpose reg. */
#define EIEP0 0x45 /* DMA channel 8 external index reg. */
#define EMEP0 0x46 /* DMA channel 8 external modify reg. */
#define ECEP0 0x47 /* DMA channel 8 external count reg. */

#define IIEP1 0x48 /* DMA channel 9 index reg. */
#define IMEP1 0x49 /* DMA channel 9 modify reg. */
#define CEP1 0x4A /* DMA channel 9 count reg. */
#define CPEP1 0x4B /* DMA channel 9 chain pointer reg. */
#define GPEP1 0x4C /* DMA channel 9 general purpose reg. */
#define EIEP1 0x4D /* DMA channel 9 external index reg. */
#define EMEP1 0x4E /* DMA channel 9 external modify reg. */
#define ECEP1 0x4F /* DMA channel 9 external count reg. */

/* Serial Port registers */
#define STCTL0 0xe0 /* SPORT 0 transmit control reg. */
#define SRCTL0 0xe1 /* SPORT 0 receive control reg. */
#define TX0 0xe2 /* SPORT 0 transmit data buffer */
#define RX0 0xe3 /* SPORT 0 receive data buffer */
#define TDIV0 0xe4 /* SPORT 0 transmit divisor reg. */
#define TCNT0 0xe5 /* SPORT 0 transmit count reg. */
#define RDIV0 0xe6 /* SPORT 0 receive divisor reg. */
#define RCNT0 0xe7 /* SPORT 0 receive count reg. */
#define MTCS0 0xe8 /* SPORT 0 multichannel xmit selector */
#define MRCS0 0xe9 /* SPORT 0 multichannel rcv selector */
#define MTCCS0 0xea /* SPORT 0 multichannel xmit compand selector */
#define MRCCS0 0xeb /* SPORT 0 multichannel rcv compand selector */
#define KEYWD0 0xec /* SPORT 0 keyword register */
#define IMASK0 0xed /* SPORT 0 keyword mask register */
#define STCTL1 0xf0 /* SPORT 1 transmit control reg. */
#define SRCTL1 0xf1 /* SPORT 1 receive control reg. */
#define TX1 0xf2 /* SPORT 1 transmit data buffer */
#define RX1 0xf3 /* SPORT 1 receive data buffer */
#define TDIV1 0xf4 /* SPORT 1 transmit divisor reg. */
#define TCNT1 0xf5 /* SPORT 1 transmit count reg. */
#define RDIV1 0xf6 /* SPORT 1 receive divisor reg. */
#define RCNT1 0xf7 /* SPORT 1 receive count reg. */
#define MTCS1 0xf8 /* SPORT 1 multichannel xmit selector */
#define MRCS1 0xf9 /* SPORT 1 multichannel xmit compand selector */
Control and Status Registers

#define MTCCS1 0xfa /* SPORT 1 multichn rcv compand selector */
#define MRCCS1 0xfb /* SPORT 1 multichn rcv compand selector */
#define KEYWD1 0xfc /* SPORT 1 keyword register */
#define IMASK1 0xfd /* SPORT 1 keyword mask register */

/* ———————————————————— Aliases for TX and Rx ———————————————————— */
#define TX0_A 0xe2 /* SPORT 0 transmit data buffer A */
#define RX0_A 0xe3 /* SPORT 0 receive data buffer A */
#define TX1_A 0xf2 /* SPORT 1 transmit data buffer A */
#define RX1_A 0xf3 /* SPORT 1 receive data buffer A */
#define TX0_B 0xee /* SPORT 0 transmit data buffer B */
#define RX0_B 0xef /* SPORT 0 receive data buffer B */
#define TX1_B 0xfe /* SPORT 1 transmit data buffer B */
#define RX1_B 0xff /* SPORT 1 receive data buffer B */
SYMBOL DEFINITIONS FILE (def21065L.h)
F  INTERRUPT VECTOR ADDRESSES

Table F-1 lists all processor interrupts according to their bit position in the IRPTL and IMASK registers. Four memory locations separate each interrupt vector. For each vector, Table F-1 also lists the address, mnemonic (not required by the assembler), and priority.

The addresses in the vector table represent offsets from a base address. For an interrupt vector table in internal memory, the base address is 0x0000 8000, the beginning of Block 0. For an interrupt vector table in external memory, the base address is 0x0002 0000.

Table F-1. IRPTL/IMASK interrupt vectors and priorities

<table>
<thead>
<tr>
<th>Bit</th>
<th>Address</th>
<th>Name</th>
<th>Description</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x00</td>
<td>Reserved</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0x04</td>
<td>RSTI</td>
<td>Reset (read-only, non-maskable)</td>
<td>Highest</td>
</tr>
<tr>
<td>2</td>
<td>0x08</td>
<td>Reserved</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0x0C</td>
<td>SOVFI</td>
<td>Status stack or loop stack overflow or PC full</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0x10</td>
<td>TMZHI</td>
<td>Timer high priority option</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0x14</td>
<td>VIRPTI</td>
<td>Vector interrupt</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0x18</td>
<td>IRQ2I</td>
<td>IRQ2 asserted</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0x1C</td>
<td>IRQ1I</td>
<td>IRQ1 asserted</td>
<td></td>
</tr>
</tbody>
</table>
Table F-1. IRPTL/IMASK interrupt vectors and priorities (Cont’d)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Address</th>
<th>Name</th>
<th>Description</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0x20</td>
<td>IRQ0I</td>
<td>IRQ0 asserted</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0x24</td>
<td></td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0x28</td>
<td>SPROI</td>
<td>DMA channel 0/1; SPORT0 receive A&amp;B</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0x2C</td>
<td>SPR1I</td>
<td>DMA channel 2/3; SPORT1 receive A&amp;B</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0x30</td>
<td>SPT0I</td>
<td>DMA channel 4/5; SPORT0 transmit A&amp;B</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>0x34</td>
<td>SPT1I</td>
<td>DMA channel 6/7; SPORT1 transmit A&amp;B</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>0x38</td>
<td></td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>0x3C</td>
<td></td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>0x40</td>
<td>EP0I</td>
<td>DMA channel 8; Ext. port buffer 0</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>0x44</td>
<td>EP1I</td>
<td>DMA channel 9; Ext. port buffer 1</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>0x48</td>
<td></td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>0x4C</td>
<td></td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>0x50</td>
<td></td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>0x54</td>
<td>CB7I</td>
<td>Circular buffer 7 overflow</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>0x58</td>
<td>CB15I</td>
<td>Circular buffer 15 overflow</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>0x5C</td>
<td>TMZLI</td>
<td>Timer low priority option</td>
<td></td>
</tr>
</tbody>
</table>
Interrupt Vector Addresses

Table F-1. IRPTL/IMASK interrupt vectors and priorities  (Cont’d)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Address</th>
<th>Name</th>
<th>Description</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>0x60</td>
<td>FIXI</td>
<td>Fixed-point overflow</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>0x64</td>
<td>FLTOI</td>
<td>Floating-point overflow exception</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>0x68</td>
<td>FLTUI</td>
<td>Floating-point underflow exception</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>0x6C</td>
<td>FLTII</td>
<td>Floating-point invalid exception</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>0x70</td>
<td>SFT0I</td>
<td>User software interrupt 0</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>0x74</td>
<td>SFT1I</td>
<td>User software interrupt 1</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>0x78</td>
<td>SFT2I</td>
<td>User software interrupt 2</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>0x7C</td>
<td>SFT3I</td>
<td>User software interrupt 3</td>
<td>Lowest</td>
</tr>
</tbody>
</table>

When an external source boots the processor’s on-chip SRAM, the interrupt vector table is located in internal memory. When the processor is in “no boot” mode because it will execute from off-chip memory, the interrupt vector table must be located in the off-chip memory. When an external EPROM or host boots the processor’s SRAM, the processor automatically sets bit 16 of IMASK (the EP0I interrupt for DMA channel 8) to 1 following reset to enable the DMA done interrupt for channel 8. It initializes IRPTL to all 0s following reset.

Applications can use the IIVT bit in the SYSCON control register to override the booting mode, which determines the location of the interrupt vector table. If the processor is in “no boot” mode, setting IIVT to 1 selects an internal vector table, and setting IIVT to 0 selects an external vector table. IIVT has no effect when an external source boots the processor while it is in other than “no boot” mode.
Figure F-1 on page F-5 shows the bit values in the IRPTL and IMASK registers. The default values are valid for the IMASK register only; the processor clears IRPTL after reset. For IMASK, 1 = unmasked (enabled), and 0 = masked (enabled).
Interrupt Vector Addresses

Figure F-1. IRPTL/IMASK register bit values
INDEX

Symbols
“Group I Instructions (Compute & Move)” on page A-28 A-2

Numerics
32- and 48-bit memory words, using 5-30
32-bit data starting memory address 5-35

A
AC (ALU fixed-point carry) bit 2-16
described 2-18
fixed-point logic operations and 2-18
setting and clearing 2-18
AC condition 3-13
Access address fields for external
memory space 5-26
Access restrictions
for internal buses 5-27
memory space 5-27
Access timing
external memory space 5-65
bus master reads 5-66
bus master writes 5-67
diagram of 5-65
external bus control 5-65
multiprocessor memory space 5-65,
5-67
diagram of 5-68
Accessing data over the PM bus 5-10
ACK
EPROM booting 12-52
extending off-chip memory accesses 5-53
external memory space interface and
5-47
IOP register writes and 7-26
multiprocessing and 12-52
pin definition 12-7
single-word EPBx data transfers and
7-28
state after reset 12-22
Act command 10-30
Address boundaries
external memory space 5-19
internal memory space 5-19
multiprocessor memory space 5-19
reserved addresses 5-19
Address decoding table for memory
accesses 5-20
Address ranges for instructions and data
5-34
INDEX

Address regions of internal memory space 5-23
Addressing
  32-bit data starting memory address 5-35
  data accesses of external memory space 5-52
direct 5-11
immediate 5-11
indirect 5-11
ADDRx
  and host accesses 8-11
EPROM booting and 12-54
external memory space interface and 5-44
generating addresses outside the address range of external memory space 6-30
parallel SDRAM refresh command 10-28
pin definition 12-4
state after reset 12-22
ADI product information, sources of -xix, -xiii
ADI product literature -xxiv, -xviii
ADREbDY bit (active drive REDY) switching between open and active-drain output 8-12
ADSP-21065L block diagram 6-2
AF (ALU floating-point operation) bit 2-16
described 2-19
AI (ALU floating-point invalid operation) bit 2-16
described 2-19
setting 2-19
AIS (ALU floating-point invalid operation) bit 2-17
described 2-19
setting 2-19
Alternate DAG registers 4-3
architecture 4-4
described 4-3
diagram of 4-4
MODE1 control bits for 4-5
SRD1H (DAG1 alternate register select 7-4) 4-5
SRD1L (DAG1 alternate register select 3-0) 4-5
SRD2H (DAG2 alternate register select 15-12) 4-5
SRD2L (DAG2 alternate register select 11-8) 4-5
Alternate register file registers 2-11
context switching 2-11
described 2-11
control bits 2-11
effect latency of activation 2-11
selecting the active sets 2-11
SRRFH 2-11
SRRFL 2-11
ALU
data formats 2-12
described 2-1
instruction set summary 2-21
instruction types 2-12
operating modes 2-14
INDEX

see ALU operating modes
status flags 2-16
see ALU status flags
ALU fixed-point saturation mode
2-14
ALU overflow flag and 2-15
described 2-14
negative overflows 2-15
positive overflows 2-14
ALU floating-point rounding
boundary 2-15
32-bit IEEE results 2-15
40-bit results 2-15
fixed-to-floating-point conversion
2-15
floating-point results, format of
2-15
ALU floating-point rounding
modes 2-15
round-to-nearest 2-15
round-to-zero 2-15
ALU instruction set, summary of
2-21
ALU operating modes 2-14
ALUSAT (ALU saturation mode)
bit 2-14
fixed-point saturation mode 2-14
see ALU fixed-point saturation
mode
MODE1 control bits 2-14
RND32 (floating-point rounding
boundary) bit 2-14
TRUNC (floating-point
rounding mode) bit 2-14
ALU operation
CACC status flag updates 2-16
compare accumulate operations
2-19
fixed-to-floating-point conversion
2-15
fixed-point results 2-13
floating-point rounding boundary
2-15
see ALU floating-point rounding
boundary
floating-point rounding modes
2-15
see ALU floating-point rounding
modes
status flag updating 2-17
ALU operations
and the register file 2-13
fixed-point inputs 2-13
fixed-point results, storing 2-13
instruction set summary 2-21
operands 2-13
ALU single-function compute
operations
COMP (Fx, Fy) B-31
COMP (Rx, Ry) B-11
described B-2
fixed-point, summary of B-3
floating-point, summary of B-4
Fn= -Fx B-32
Fn=(Fx+Fy)/2 B-30
Fn=ABS (Fx+Fy) B-28
Fn=ABS (Fx-Fy) B-29
Fn=ABS Fx B-33
INDEX

Fn=CLIP Fx BY Fy B-49
Fn=FLOAT Rx B-41
Fn=FLOAT Rx BY Ry B-41
Fn=Fx COPYSIGN Fy B-46
Fn=Fx+Fy B-26
Fn=Fx-Fy B-27
Fn=MAX (Fx, Fy) B-48
Fn=MIN (Fx, Fy) B-47
Fn=PASS Fx B-34
Fn=RECIPS Fx B-42
Fn=RND Fx B-35
Fn=RSQRTS Fx B-44
Fn=SCALB Fx BY Ry B-36
Rn= -Rx B-16
Rn=(Rx-Ry)/2 B-10
Rn=ABS Rx B-17
Rn=CLIP Rx BY Ry B-25
Rn=FIX Fx B-39
Rn=FIX Fx BY Ry B-39
Rn=LOGB Fx B-38
Rn=MANT Fx B-37
Rn=MAX (Rx, Ry) B-24
Rn=MIN (Rx, Ry) B-23
Rn=NOT Rx B-22
Rn=PASS Rx B-18
Rn=Rx AND Ry B-19
Rn=Rx OR Ry B-20
Rn=Rx XOR Ry B-21
Rn=Rx+1 B-14
Rn=Rx+Cl B-12
Rn=Rx+Cl-1 B-13
Rn=Rx+Ry B-6
Rn=Rx+Ry+Cl B-8
Rn=Rx-1 B-15
Rn=Rx-Ry B-7
Rn=Rx-Ry+Cl B-9
Rn=TRUNC Fx B-39
Rn=TRUNC Rx BY Ry B-39
ALU status flags 2-16
AC (ALU fixed-point carry) 2-16
AF (ALU floating-point operation) 2-16
AI (ALU floating-point invalid operation) 2-16
AIS (ALU floating-point invalid operation) 2-17
AN (ALU result negative) 2-16
AOS (ALU fixed-point overflow) 2-17
AS (ALU x input sign) 2-16
ASTAT status bits, summary of 2-16
AUS (ALU floating-point underflow) 2-17
AV (ALU overflow) 2-16
AVS (ALU floating-point overflow) 2-17
AZ (ALU result 0 or floating-point underflow) 2-16
CACC (compare accumulation register) 2-16
CACC update timing 2-16
dual add/subtract (fixed-point) B-96
dual add/subtract (floating-point) B-98
fixed-point carry flag 2-18
floating- to fixed-point
INDEX

conversions and 2-17
floating-point operation flag 2-19
invalid flag 2-19
negative flag 2-18
overflow flags 2-18
sign flag 2-19
state of 2-16
status register writes, priority of 2-17
sticky status flags 2-16
STKY status bits, summary of 2-17
underflow flags 2-17
updating 2-17
zero flag 2-17
ALUSAT (ALU saturation mode) bit 2-14
AN (ALU result negative) bit 2-16
described 2-18
AOS (ALU fixed-point overflow) bit 2-17
described 2-18
Arithmetic exceptions 3-38
Arithmetic logic (ALU) unit see ALU
Arithmetic status register, see ASTAT register
Array signal processing 5-29
AS (ALU x input sign) bit 2-16
ABS and MANT operations 2-19
described 2-19
Assembler instruction mnemonics 3-12
ASTAT register 2-16
AC 2-16
AF 2-16
AI 2-16
ALU status flags, summary of 2-16
AN 2-16
AS 2-16
AV 2-16
AZ 2-16
BFT E-6
bit definitions E-10
bitwise operations and 11-14
BTF 3-12
CACC 2-16
conditional instructions and 3-12
CRBM 7-11
default bit values, diagram of E-9
described E-8
flag status updates 12-31
FLAG_{3:0} 11-14, 12-29
FLAG_{3:0} inputs 12-31
FLAG_{3:0} outputs 12-33
FLAGx status bits 12-32
FLAGxO status bits 12-32
initialization value E-8
MI 2-34
MN 2-34
MU 2-34
multiplier status bits, summary of 2-34
multiplier status flags 2-34
multiprocessing and 7-11
MV 2-34
preserved current values of 3-49
INDEX

RTI instruction and 3-16
Shifter status bits, summary of 2-45
signaling external devices with
FLAGx bits 12-33
SS 2-45
status stack pushes and pops 12-34
status stack save and restore operations 3-48
SV 2-45
SZ 2-45
Asynchronous external interrupts described 3-51
guarantee sampling 3-51
Asynchronous host transfers 8-9
and SDRAM 8-9
broadcast writes 8-23
see Broadcast writes
CS 8-9
host driven signals 8-9
Asynchronous inputs 12-3, 12-27
signal recognition phase 12-27
synchronization delay 12-27
Asynchronous transfer timing, see Host asynchronous accesses
AUS (ALU floating-point underflow) bit 2-17
described 2-17
floating- to fixed-point conversions and 2-17
setting 2-18
Automatic wait state option 5-62
AV (ALU overflow) bit 2-16
ALU fixed-point saturation mode and 2-15
described 2-18
AV condition 3-13
AVS (ALU floating-point overflow) bit 2-17
described 2-18
AZ (ALU result 0 or floating-point underflow) bit 2-16
described 2-17
floating- to fixed-point conversions and 2-17
setting 2-17
underflow status 2-18

B
B (DAG base address) registers 4-2
circular data buffers and 4-11
Bank activate command (SDRAM), see Act command
BCNT register
BTC and 7-18
bus lock and 7-18
bus mastership timeout counter 7-18
HBR and 7-18
master processor operation 7-18
BHD (buffer hang disable) bit 8-19, 9-86
single-word EPBx data transfers 7-29
SPORT data buffer read/write results 9-7
SPORT data buffer reads/writes
and 9-15
Bit reversal
bit-reverse instruction 4-14
see Bit-reverse instruction
bit-reverse mode 4-13
see Bit-reverse mode
data addressing 4-13
BIT SET instruction
software interrupts, activating 3-49
Bit test flag bit, see BTF bit
Bit-reverse instruction
BITREV 4-14
described 4-14
index (I) registers and 4-14
operation sequence 4-14
Bit-reverse mode
control bits, summary of 4-14
DAG1 operation 4-13
DAG2 operation 4-13
described 4-13
effect timing 4-14
postmodify addressing operations 4-14
BM condition 3-13, 3-14
BMAX register
bus mastership timeout 7-17
maximum value of 7-17
BMS 5-53
boot mode 12-50
EPROM boot mode 5-53, 12-51
EPROM boot sequence after reset 12-53
EPROM chip select 12-53
host booting 12-56
multiprocessing 12-51
multiprocessor EPROM booting 12-59
multiprocessor host booting 12-59
pin connection 5-53
pin definition 12-13
state after reset 12-23
BMSTR
pin definition 12-13
state after reset 12-22
Boot hold off 12-52
Boot master output, see BMSTR
Boot memory select (BSEL, BMS)
described 5-53
EPROM boot mode 5-53
pin connections 5-53
Boot memory select, see BMS
Boot mode pins
BMS 12-50
BSEL 12-50
configurations 12-51
Boot modes
boot sequence and kernel loading 12-54
data packing 12-49
EPROM 5-53, 12-49, 12-51
see EPROM boot mode
host 12-49, 12-56
see Host boot mode
interrupt vector table address 5-30
no boot 5-30, 12-49, 12-60
see No boot mode
INDEX

pins, see Boot mode pins
selecting 12-50
when IIIVT=1 5-30
Boot select override, see BSO (boot select override)
Boot sequence and kernel loading 12-51
Booting 12-49
described 12-49
host boot sequence 12-58
loading an entire program 12-49
loading routine 12-49
modes, see Boot modes
multiprocessor systems 12-58
selecting 12-50
Branch instructions
call 3-16
delayed, see Delayed branches
described 3-16
jump 3-16
nondelayed, see Nondelayed branches
parameters 3-16
program memory data accesses 3-11
RTI 3-16
RTS 3-16
Broadcast writes
CS 8-23
defined 8-23
implementing 8-23
REDY 8-23
\textbf{BR}x
BTC and 7-12, 8-8
connection in a multiprocessor system 7-3
multiprocessor bus arbitration 7-10
pin definition 12-14
state after reset 12-22
system bus acquisition 7-12
BSEL 5-53
boot mode 12-50
EPROM boot mode 5-53
host booting 12-56
multiprocessor host booting 12-59
pin connection 5-53
pin definition 12-14
state after reset 12-23
BSO (boot select override)
accessing EPROM after bootstrap 12-55
overriding BMS 12-55
writing to BMS memory space 12-56
Bstop command 10-30
defined 10-5
BSYN (bus synchronization) bit 7-22, 7-42, 8-40
BTC
\textbf{BR}x and 7-12, 8-8
bus mastership timeout and 7-18
defined 8-5
external accesses and 7-14
external bus in 7-13
multiprocessing events that trigger a 7-12
INDEX

multiprocessing transfer sequence 7-13
multiprocessor bus arbitration 7-12
without CPA 7-19
BTF (bit test flag) bit
conditional instruction use E-7
system register bit manipulation instruction E-6
test operation results E-6
XOR operation results E-7
BTST Rx BY ‹data8› operation described B-73
shifter status flags B-73
BTST Rx BY Ry operation described B-73
shifter status flags B-73
Buffer hang disable bit, see BHD (buffer hang disable) bit
Burst stop command (SDRAM), see Bstop command
burst type (SDRAM), defined 10-5
Bus arbitration synchronization after reset 7-21
BSYN bit 7-22
bus synchronization scheme 7-21
described 7-21
individual processor reset 7-23
multiprocessor configuration 7-21
processor ID1 operation during 7-23
SRST 7-21
synchronization sequence 7-22

Bus arbitration, multiprocessing 7-10, 7-12
Bus connections
EPBx buffers 8-18
on-chip memory 5-7
Bus hold time cycle
described 5-60
diagram of 5-61
Bus idle cycle
described 5-58
diagram of 5-59
EBxWS bit values 5-60
with following SDRAM access 5-59
Bus lock and semaphores 7-34
bus lock feature 7-34
BUSLK (bus lock) bit, requesting bus lock 7-34
current bus master, identifying 7-34
read-write-modify operations 7-35
read-write-modify operations on semaphores 7-34
requesting bus lock 7-34
semaphore locations 7-34
semaphore, described 7-34
SWPD bit 7-35
Bus lock feature 7-18, 7-34
Bus master condition, see BM condition
Bus mastership timeout
BCNT register 7-18
BMAX register 7-17
INDEX

BTC and 7-18
configuring 7-17
Bus slave, defined 8-4
Bus synchronization
multiprocessor systems 7-11
scheme 7-21
Bus transition cycle, see BTC
BUSLK bit and bus mastership
timeout 7-18

C
C (DMA count register) 6-31
DMA interrupts and 6-9
CACC (compare accumulation
register) bit 2-16
described 2-19
update timing 2-16
Cache hit
defined 3-58
LRU bit and 3-59
triggering 3-59
Cache miss
defined 3-58
LRU bit and 3-59
memory accesses over PM bus
5-10
triggering 3-59
with DAG2 transfers 5-10
Call instructions
conditional branching 3-16
delayed and nondelayed 3-17
described 3-16
indirect, direct, and PC-relative
3-17
program memory data accesses
3-11
CAS
pin definition 12-10
state after reset 12-22
CAS before RAS automatic refresh
mode, see CBR
CAS latency, defined 10-5
CBR, defined 10-6
Changing external port DMA
channel priority assignment,
example of 6-38
Channel selection registers
architecture 9-72
channel slot operation 9-72
channel slot/register bit
correspondence 9-72
companding 9-72
MRCCSx 9-72
MRCSx 9-72
MTCCSx 9-72
MTCSx 9-72
operation 9-72
summary of 9-72
Channel slots
capabilities, summary of 9-67
companding 9-72
described 9-67
individual slots,
enabling/disabling 9-72
number of 9-67
operation parameters 9-72
synchronization 9-69
INDEX

CHEN (DMA chaining enable) bit
  6-14, 6-39
  described 6-15
Chip select, see CS
CHNL (current channel selected)
  bits 9-17, 9-38
  defined 9-26
  described 9-38, 9-71
Circular buffer addressing
  address wraparound 4-9
  architecture of circular data
    buffers 4-9
  buffer overflow interrupts 4-12
    see Circular buffer overflow interrupts
  circular buffer operation 4-10
    see Circular buffer operation
  circular buffer registers 4-11
    see Circular buffer registers
  index (I) registers and 4-9
  modify (M) registers 4-9
  postmodify addressing 4-9
  premodify addressing 4-9
  stepping through each buffer
    location 4-9
Circular buffer operation 4-10
  B register, loading 4-10
  data overflows 3-38
  first postmodify access 4-10
  I (index) register value, updating 4-10
  initializing buffer size (number of
    locations) 4-10
  initializing I (index) register value 4-10
  L (locations) register initialization
    4-10
  loading base address of buffer
    4-10
  set up in assembly language 4-10
Circular buffer overflow interrupts 4-12
  address wraparound 4-12
  implementing routines that swap I/O buffer pointers 4-12
  instructions that generate 4-12
  masking 4-13
  source of 4-12
  STKY register and 4-13
  summary of 4-12
Circular buffer registers 4-11
  B (DAG base address) registers
    4-11
  I (DAG index) registers 4-11
  L (DAG locations) registers 4-11
  M (DAG modify) registers 4-11
Circular data buffers 4-1
  addressing 4-9
    see Circular buffer addressing
  architecture 4-9
  assembly language set up 4-10
  base address 4-2, 4-9
  diagram of 4-9
  number of locations in 4-2
  operation 4-10
    see Circular buffer operation
  postmodify addressing operations and 4-7
INDEX

registers 4-11
see Circular buffer registers
Cjump/Rframe (type 24)
  instruction
described A-81
opcode (Rframe) A-82
opcode (with direct branch) A-82
opcode (with PC-relative branch) A-82
operations, summary of A-81
syntax summary A-10
CKRE (frame sync clock edge) bit
  9-16, 9-21
clock signal options 9-50
defined 9-26
  described 9-55
receive data and frame syncs 9-55
transmit data and frame syncs 9-55
Clear interrupt (CI) modifier 3-44
clearing the current interrupt for reuse 3-49
  example code using 3-50
Clear MR register 2-30
Clearing extra DMA requests 6-64
CLKin
  and XTAL 12-26
  enabling the internal clock generator 12-27
frequencies and processor cycles 12-26
JTAG connection 12-40
out
phase lock, achieving 12-27
pin definition 12-14
SPORT clock and frame sync frequencies 9-41
state after reset 12-23
CLKIN frequencies
  FLAGx operations 12-26
  host accesses 12-26
  IRQx operations 12-26
  multiprocessing operations 12-26
  of master processor operations 12-26
  processor cycles and 12-26
  SDRAM operations 12-26
  SPORT operations 12-27
wait state programming 12-27
Clock distribution 12-43
  controlled impedance transmission line 12-43
  end-of-line termination 12-43
  propagation delay 12-43
  source termination
  guidelines for using 12-44
  source termination, see Source termination
Clock in, see CLKin
Clock jitter 12-42
Clock skew 12-40, 12-43
Cluster bus
defined 8-5
described 8-44
Cluster multiprocessing 7-6
  application of 7-7
  configuration 7-7
described 7-7
diagram of 7-7

1-12  ADSP-21065L SHARC DSP User’s Manual
INDEX

CMOS input inverter 12-41
CNT_EXPx (timer counter expired) bit 11-6
CNT_OVFx (timer counter overflowed) bit 11-6
COMP (Fx, Fy) (floating-point) operation
ALU status flags B-31
described B-31
COMP (Rx, Ry) (fixed-point) operation
ALU status flags B-11
ASTAT register and B-11
described B-11
Companding 9-45
described 9-45
expanding in place 9-47
formats 9-44
in place 9-46
multichannel SPORT mode 9-67
operation 9-46
receive comparison enabled and 9-74
standard SPORT mode 9-59
supported algorithms 9-45
Computation units alternate register file registers 2-11
see Alternate register file registers
ALU 2-12
see ALU
ALU data formats 2-12
ALU instruction types 2-12
ALU operating modes 2-14
see ALU operating modes
ALU operations, see ALU operations
architecture 2-2
data formats 2-4
described 2-1
diagram of 2-2
extended-precision floating-point operations 2-5
fixed-point format 2-7
floating-point exception handling 2-6
interface with internal data buses 2-9
multifunction operations 2-50
see Multifunction operations
multiplier 2-1, 2-26
see Multiplier unit
register file and 2-9
rounding modes 2-7
Shifter unit 2-1, 2-41
see Shifter unit
short word floating-point 2-5
single-precision floating-point format 2-4
temporary data storage 2-2
Compute (type 2) instruction described A-32
example A-32
opcode A-32
syntax summary A-4
Compute and move/modify instructions compute (type 2) instructions A-4
compute/dreg⇒DM|PM,
immediate modify (type 4) instructions A-5
compute/ureg⇔DM|PM, register modify (type 3) instructions A-4
compute/ureg⇔ureg (type 5) instructions A-5
IF COND A-4
immediate Shift/dreg⇔DM|PM (type 6) instructions A-5
summary of A-4
Compute operation reference
compute operations B-1
multifunction operations B-94
see Multifunction operations
multiplier operations B-50
see Multiplier operations
shifter operations B-63
see Shifter operations
single-function operations
see Single-function compute operations
compute operation reference
single-function operations B-2
Compute operations described B-1
types B-1
Compute/dreg⇔DM/dreg⇔PM (type 1) instruction described A-30 example A-30 opcode A-30
Compute/dreg⇔DM|PM, immediate modify (type 4) instruction described A-35 example A-35 opcode A-36 syntax summary A-5
Compute/modify (type 7) instruction example A-42 opcode A-42
Compute/ureg⇔DM|PM, register modify (type 3) instruction example A-33 opcode A-34 syntax summary A-4
Compute/ureg⇔ureg (type 5) instruction described A-37 example A-37 opcode A-37 syntax summary A-5
Concurrent DMA accesses of external memory space 6-74
Concurrent DMA accesses of internal memory space 6-74
Condition codes 3-12
AC 3-13
AV 3-13
BM 3-14
EQ 3-13
FLAG0_IN 3-13
FLAG1_IN 3-13
FLAG2_IN 3-13
INDEX

FLAG3_IN 3-13
FOREVER 3-15
GE 3-14
GT 3-14
LCE 3-14
LE 3-13
LT 3-13
MN 3-13
MV 3-13
NE 3-14
NOT AC 3-14
NOT AV 3-14
NOT BM 3-15
NOT FLAG0_IN 3-14
NOT FLAG1_IN 3-14
NOT FLAG2_IN 3-14
NOT FLAG3_IN 3-14
NOT ICE 3-14
NOT MS 3-14
NOT MV 3-14
NOT SV 3-14
NOT SZ 3-14
NOT TF 3-15
summary of 3-13, A-13
SV 3-13
SZ 3-13
TF 3-14
TRUE 3-15

Conditional instructions
ASTAT register 3-12
bit test flag (BTF) 3-12
branches 3-16
condition codes 3-12
condition codes, summary of

3-13, A-13
conditions 3-12
CRBM in 7-12
executing 3-12
FLAGx bit states and 12-32
FOREVER condition 3-12
IF NOT LCE 3-13
instruction set syntax A-3
LCE condition 3-12
memory writes 5-49
memory writes and decoded
memory address lines 5-49
MODE1 register 3-12
NOT LCE condition 3-12
opcode components 3-12
termination codes 3-12
TRUE condition 3-12
Conditions that generate DMA and
I/O interrupts 6-47
Configuring SDRAM operation
10-13
Context switching 4-3
alternate register file registers and
2-11
MR registers and 2-29
Control and status registers
bit states E-1
described E-1
IOP registers E-1, E-31
see IOP registers
symbol definitions file
(DEF21065L.H) E-116
system registers E-1
see System registers
INDEX

Controlled impedance transmission line 12-43
Conventions of notation, global -xxv
Core accesses
  FLAGx and system bus accesses 8-48
  MSx and system bus accesses 8-48
  of the system bus 8-48
  over the PM bus 5-10
  type 10 instruction and system bus accesses 8-48
Core controlled interrupt-driven I/O 6-46
implementing 8-20
Core hang
  avoiding 9-15
  BHD (buffer hang disable) bit 7-29, 8-19, 9-86
defined 8-19
  reads/writes of RX/TX buffer and 9-86
  single-word data transfers 7-29
Core priority access
  described 7-18
  pin 7-11, 12-16
  slave processor external bus access sequence 7-19, 7-20
timing diagram 7-19
Counter-based loops
  CURLCNTR 3-35
  interrupt processing in 3-29
  overhead in 3-29
  pipelined one-instruction three-iteration 3-28
  pipelined one-instruction two-iteration (2 cycles of overhead) 3-29
  restrictions 3-28
CP (chain pointer) register
  and PCI bit, diagram of 6-40
  memory address field 6-39
  PCI (program controlled interrupts) bit 6-40
  symbolic address restriction 6-44
CPA
  core priority access timing, diagram of 7-19
  interrupting DMA transfers 7-18
  multiprocessor bus arbitration 7-11
  nonmultiprocessing system 7-19
  pin definition 12-16
  state after reset 12-23
CRBM (current bus master) bit
  7-11, 7-42, 8-40
  conditional instructions and 7-12
  Crosstalk, reducing 12-45
INDEX

Crystal oscillator terminal, see XTAL
CS
accessing a processor 8-11
EPROM boot mode 12-51
implementing broadcast writes 8-23
multiprocessor booting 12-59
pin definition 12-8
state after reset 12-23
CURLCNTR 3-12, 3-34
decrementing 3-34
described 3-34
LCNTR and 3-35
reading the 3-34
value while no loop executing 3-35
write restrictions 3-35
writing to 3-35
Current loop count, see CURLNCTR
Current loop counter, see CURLCNTR
Cycles, CLKIN frequencies and 12-26

D
DAG address output and modification 4-6
address offset modifier 4-6
immediate modifier value 4-6
immediate modifiers 4-8
M (DAG modify) registers 4-6
modify instructions 4-7
postmodify operations 4-6
premodify operations 4-6
DAG modify instructions 4-7
DAG operation 4-6
address output and modification 4-6
see DAG address output and modification
bit reversal and 4-13
bit-reverse instruction 4-14
see Bit-reverse instruction
bit-reverse mode 4-13
see Bit-reverse mode
circular buffer addressing 4-9
see Circular buffer addressing
dual data accesses and PM and DM bus addresses 5-8
generating internal bus addresses 5-26
generating memory addresses 5-11
indirect addressing 5-11
short word addresses and 4-6
summary of operations 4-6
DAG register transfers 4-15
between DAGs and DM data bus 4-15
data alignment with DM bus 4-15
described 4-15
diagram of 4-15
unsupported instruction sequences 4-16
DAG registers 4-1
alternate registers 4-3
INDEX

see Alternate DAG registers
architecture 4-2
base address (B) registers 4-2
circular data buffers 4-1, 4-2
DAG1 4-1
DAG2 4-1
described 4-2
diagram of 4-3
index (I) registers 4-2
locations (L) registers 4-2
MODE1 control bits for alternate
register set 4-5
modify (M) registers 4-2
operation 4-6
see DAG operation
pointer increment value 4-2
pointer to memory 4-2
see also DAG1, DAG2
subregister types, summary of 4-2
transfers with 4-15
see DAG register transfers
DAG1
bit-reverse mode 4-13
described 4-1
immediate I (index) register
modifier values 4-8
indirect addressing and the DM
bus 5-11
program sequencing 3-7
transfers with the DM data bus
4-15
Data accesses
conversion between short and
normal words 5-41
MSW/LSW of 32-bit words 5-41
of 40-bit data with 48-bit word
5-40
short word 5-41
word width and RND32 5-41
Data address generators, see DAG
operation
Data addresses 5-11
direct 5-11
immediate 5-11
indirect 5-11
Data addressing
address output and modification,
see DAG address output and
modification
circular buffer operation 4-10
see Circular buffer operation
circular data buffers 4-9
see Circular buffer addressing
DAG register transfers 4-15
see DAG register transfers
data address generators, see DAG
registers 4-1
described 4-1
INDEX

postmodify operations 4-6
  see Postmodify addressing operations
premodify operations 4-6
  see Premodify addressing operations
premodify vs. postmodify addressing, diagram of 4-7
Data bandwidth bottlenecks 7-6
Data delays, latencies, and throughput 12-62
cycles per 12-62
defined 12-62
summary of 12-62
Data flow multiprocessing 7-6
  application of 7-6
described 7-6
diagram of 7-6
Data formats 9-44
  ALU 2-12
  computations 2-4
  justification 9-44
Data memory data bus, see DM bus
Data receive (DRx_X) pins 9-4,
  12-11
Data segments, invalid addresses 5-52
Data storage, capacity
  mixed words 5-43
  packed words 5-43
Data storage, configuration
  32- and 40-bit data 5-40
  changing word width 5-40
  IMDWx bit (SYSCON) 5-40
Data throughput, defined 12-62
Data transfers
  48-bit accesses of program memory 5-14
  address sources 5-11
  between DM data bus and external memory 5-14
  between DM data bus and internal memory 5-14
  between memory and registers 5-12
  between memory and SPORTS 9-77
  between PX1 and PM data bus 5-12, 5-14
  between PX2 and DM data bus 5-14
  between PX2 and PM data bus 5-12
  example code for 48-bit program memory access 5-14
  multiprocessing
    see Multiprocessing data transfers
  multiprocessing DMA 7-30
  multiprocessing IOP register reads, see IOP register reads
  of 40-bit DM data bus 5-14
  over DM bus 5-11
  over PM bus 5-11
  over the external bus 5-43
  packed data 5-43
  PM bus destinations 5-11
  PX register data alignment 5-12
  PX register transfers, diagram of
INDEX

5-13
single-cycle, number of 5-17
universal register-to-register 5-12
with memory 5-7
Data transmit (DTx_X) pins 9-4
DATAx
and host accesses 8-30
EPROM boot mode and 12-51
EPROM boot sequence after reset 12-54
external memory space interface and 5-45
external port data alignment, diagram of 8-31
host booting and 12-58
pin definition 12-4
state after reset 12-23
DB modifier 3-18
Decode address register 3-6
Decode cycle 3-4
Decoded memory address lines (MSx) 5-49
Decoding table for memory addresses 5-20
Decoupling capacitors and ground planes 12-46
power plane 12-46
VDD pins 12-46
def21065L.h file 9-12
complete listing E-116
Delayed branches 3-18
call return address 3-19
DB modifier and 3-18
defined 3-19
instructions following, restriction 3-20
interrupt processing and 3-23
pipelined stages of jumps/calls 3-19
pipelined stages of returns 3-20
reading PC stack/PC stack pointer and 3-24
DEN (DMA enable) bit 6-9, 6-14, 8-28
described 6-15
enabling/disabling DMA 8-20
single-word EPBx data transfer control 7-29
single-word, non-DMA EPBx transfers 7-29, 8-20
Denormal operands 2-36
Design recommendations 12-45
crosstalk, reducing 12-45
reflections, reducing 12-46
Design resource references 12-47
Direct addressing 5-11
absolute address A-18
PC-relative address A-18
Direct jump|call (type 8) instruction described A-45
example A-46
opcode (with direct branch) A-46
opcode (with PC-relative branch) A-47
syntax summary A-6
Direction of DMA data transfers 6-15
EXTERN 6-16
INDEX

TRAN 6-16
Disable SDCLK0 bit, see DSDCTL bit
Disable SDCLK1 bit, see DSDCK1 bit
DITFS (data-independent TFS) bit 9-16
continuous TFS and 9-58
defined 9-26
described 9-57
DM bus
address bits, diagram of 5-8
and EPBx buffers 8-18
data storage 5-8
data transfer destinations 5-11
data transfer types 5-11
data transfers 5-7
defined 5-3
generating addresses for 5-11, 5-26
memory accesses 5-27
memory connection 5-7
PX register accesses 5-28
transferring data to the PM bus 5-12
transfers with DAG registers 4-15
DMA
address generators 6-75
asynchronous requests and DMARx 6-66
C (count) register initialization 6-29
chain insertion 6-44
channel active status 6-24, 6-26
channel chaining status 6-24
channel data buffers 6-28
channel parameter registers 6-28
channel status 6-24
channels 6-4
concurrent DMA accesses of on-chip memory space 6-74
control and data paths, diagram of 6-3
control registers, see DMACx registers
cycle, defined 6-28
data buffers 6-4
data packing through the EPBx buffers 6-51
data transfers, see DMA data transfers
disabling chaining 6-39
enabling 6-9
external port FIFO buffers (EPBx), see EPBx buffers
flushing the request counter (FLSH) 6-18
grant outputs 6-3, 6-64
see also DMAGx
II (index) register overflow 6-29
interrupts 6-45
maximum number of requests without a grant 6-64
mode configurations, summary of 6-56
operation 6-27
operation modes 6-11
overall throughput of multiple
INDEX

DMA channel memory accesses
6-74
packing sequence for download of
processor instructions from a
16-bit bus 6-53
packing sequence for download of
processor instructions from a
32-bit bus 6-52
packing sequence for host to
processor (8- to 48-bit words)
6-54
parameter registers, see DMA
parameter registers
polling for DMA status,
restrictions on 6-26
request inputs 6-3
see also DMARx
sequence 6-29, 6-39, 6-48, 6-49
system configurations for
interprocessor operation 6-70
summary of 6-70
TCB chain loading, see TCB
chain loading
transfer control block (TCB), see
TCB
DMA chain insertion mode 6-15
described 6-44
restrictions 6-45
setting up 6-44
DMA chaining 6-8
active status E-65
and the CP (chain pointer)
register 6-39
automatic 6-15
chain insertion mode, see DMA
chain insertion mode
chain pointer register, see CP
(chain pointer) register
channel status 6-24
described 6-39
disabling 6-15, 6-39
disabling DMA interrupts 6-40
DMA general purpose register and
see GP (DMA general purpose)
register
enabling 6-15, 6-39
enabling and disabling DMA
interrupts 6-46
initiating data transfers 6-39
inserting a high priority chain in
an active DMA chain 6-44
location of last DMA sequence
transferred 6-41
pointing to the next set of DMA
parameters in internal memory
6-39
prioritizing external DMA
accesses 6-37
priority of TCB chain loading
6-37
restrictions on 6-39
serial port channels 9-85
setting up and starting DMA data
transfers 6-43
status update latency 6-26
stopping a DMA sequence 6-49
storing the address of the
previously used buffer 6-30
TCB chain loading, see TCB chain loading
DMA channel parameter registers
C (count) 6-29
count register initialization 6-29
II (index) 6-28
IM (modify) 6-28
index register overflow 6-29
modify register value 6-28
offset before use 6-28
summary of 6-32
DMA channel status register, see DMASTATx register
DMA channels
active status 6-24, E-65
chaining status 6-24
channel active status bit 6-26
channel parameter registers, see DMA channel parameter registers
components of 6-27
control and status registers 6-27
data buffers 6-28
determining the state of 6-18
disabling 6-67
DMA interrupts 6-45
DMASTATx status register, see DMASTATx register
external port 6-12, 6-27, 6-30
I/O bus access priority, summary of 6-36
I/O transfer rate 6-74
inactive status E-65
index register, initializing 6-28
memory setup for EPBx DMA
channels 6-43
paced master mode DMA 6-58
parameter registers 6-28
priority of interprocessor I/O bus accesses 6-36
re-enabling 6-67
reinitialization 6-18
setting priority of 6-35
setting up master mode DMA 6-58
slave mode DMA, see Slave mode DMA
SPORT channel assignments 6-22
SPORT DMA channels 6-27
status of 6-24
TCB chain loading, see TCB chain loading
DMA control registers, see DMACx registers
DMA controller 6-7
address generator 6-33, 6-34
autoinitializing 6-39
channel priority logic 6-27
data packing order (MSWS) 6-17
data transfer rate 6-65
data transfer types 6-7
data transfers
external port block data, see External port DMA
serial port data I/O, see SPORT DMA
data transfers between external devices and external memory
INDEX

6-8
DMA control parameters 6-11
DMACx register bits
  see DMACx registers
EPROM booting 12-54
extending accesses until valid data
  in EPBx buffers 8-22
external port DMA channels 6-27
  external to internal transfer
    sequence, slave mode 6-60
  generating external memory access
    cycles in external handshake
    mode 6-69
  generating memory addresses
    6-28, 6-55
  hardware interface example,
    diagram of 6-72
host booting 12-58
host DMA transfers 8-18
I/O bus operations 6-27
incrementing and decrementing
  the modify register 6-28
internal to external memory
  transfers 6-75
internal to external transfer
  sequence, slave mode 6-61
operating modes 6-11
operation 6-7, 6-27
prioritizing external direct accesses
  to internal memory 6-37
prioritizing requests 6-35
prioritizing TCB chain loading
  6-37
priority of I/O bus accesses 6-74
redefining priority for external
  port channels 6-38
request and grant 6-35
request timing 6-65
rotating priority for external port
  channels 6-37
SPORT DMA chaining 9-85
SPORT DMA channels 6-27, 9-77, 9-78
system bus access deadlock
  resolution 8-50
system bus accesses 8-50
three-cycle pipeline 6-64
DMA data packing
  48-bit internal words 6-53
  DATAx lines used for 32-bit
    DMA data 6-53
  EPBx buffers DMA 6-51
    LSWF packing format 6-52
    MSWF packing format 6-52
    PMODE and HBW combinations, summary of 6-52
  flushing partially packed data
    6-18
  HMSWF (host packing order) bit
    6-54
host boot mode 12-57
host data transfers 8-22
in external handshake mode
  DMA 6-70
multiprocessing DMA transfers
  7-31
order of DMA transfers 6-17
packing sequence for download of
processor instructions from a 16-bit bus 6-53
packing sequence for download of processor instructions from a 32-bit bus 6-52
packing sequence for host to processor (8- to 48-bit words) 6-54
PMODE bit 6-17, 6-51, 7-31
SPORT DMA data transfers 6-22
status 6-54
status (PS) 6-16
DMA data transfers 6-7
address generation 6-28, 6-55
between external devices and external memory 6-8
between host and on-chip memory 8-21
between processors 7-30
blocked EPBx buffers 6-67
chaining 6-8
clock cycles per transfer 6-74
concurrent DMA accesses of on-chip memory space 6-74
data packing, see DMA data packing
data source and destination selection in handshake mode 6-63
DATAx lines for 32-bit data 6-53
direction of 6-15
SPORT transfers 6-22
EPBx buffers DMA 8-22
external handshake mode, see External handshake mode DMA
external port block data 6-7
external to internal transfer sequence in slave mode DMA 6-60
external transfers and the ECEPxx register 6-63
external transfers and the MSx lines 6-63
from internal to external memory space 6-75
hardware handshake signals 6-62
host block data 8-18
host EPBx transfers 8-22
host interface and 8-5
host transfers and data packing 8-22
I/O transfer rate, see DMA I/O transfer rate
initiating with chaining enabled 6-39
internal to external transfer sequence in slave mode 6-61
multiprocessing 7-25, 7-27, 7-30
non-DMA, single-word through the external port 6-50
overall throughput of multiple DMA channel memory accesses 6-74
packing order (MSWF) 6-17
packing status 6-16
priority of DMA channel accesses of the I/O bus 6-74
request timing 6-65
INDEX

request/grant latency, handling  6-65
responding to DMAGx  6-65
SDRAM controller commands and  10-36
SDRAM operation  10-24
serial port transfers  6-22
see SPORT DMA
setting up  6-9
setting up host DMA transfers to on-chip memory  8-21, 8-22
SPORT DMA block transfers  9-77
SPORT DMA channels  9-77
starting a DMA chain  6-43
through the host interface  8-5
transfer rate  6-65
types  6-7
word width of  6-16
DMA done interrupt  12-58
DMA grant x, see DMAGx
DMA handshake mode
asynchronous requests and  6-66
described  6-62
DMAGx  6-62
DMARx  6-62, 6-63
enabling  6-63
handshake timing  6-63
hardware handshake signals  6-62
DMA handshake single wait state (HIDMA)  5-57
DMA hardware interface  6-72
DMA I/O transfer rate
and uncompleted external transfers  6-74
external port DMA channels  6-74
serial port DMA channels  6-74
DMA interrupts  6-9, 8-20
and non-DMA I/O port transfers  6-46
C (count) register  6-45
C and ECEPx count registers and causes  6-47
core controlled interrupt-driven I/O  6-46
DEN (DMA enable) bit  6-17, 7-29
described  6-45
disabling  6-40
disabling in external handshake mode DMA  6-69
ECEP (external count) register  6-45
enabling and disabling, with chaining enabled  6-46
EPBx single-word transfers  6-17
generation  6-45, 7-29, 8-20
IMASK register  6-40, 6-45, 7-29
INTIO (DMA single-word interrupt enable) bit  6-17, 7-29
IRPTL register  6-9, 6-45, 7-29
masking  8-20
master mode DMA  6-45
PCI bit  6-40
program controlled  6-40
single-word EPBx transfers  7-29, 8-20
INDEX

DEN 8-20
  generation 8-20
  interrupt-driven I/O 8-20
INTIO 8-20
  masking 8-20
single-word non-DMA transfers 6-46
SPORT receive (RX) 6-23
SPORT transmit (TX) 6-23
vectors and priority, summary of 6-45
DMA modes 6-7
  chain insertion 6-44
  chaining disabled, DMA disabled 6-15
  chaining disabled, DMA enabled 6-15
configuration bit combinations, summary of 6-56
configurations 6-20
DEN and CHEN bit combinations 6-15
described 6-55
external handshake mode, see External handshake mode DMA
external port 6-55
handshake mode 6-20, 7-31
  see DMA handshake mode
initiating transfers 6-55
master mode 6-30
  see also Master mode DMA
paced master mode 6-21
  see Paced master mode DMA
slave mode 6-20, 7-31
  see also Slave mode DMA
DMA most significant word first for packing, see MSWF packing format
DMA operation
  ACK 6-69
  address generation, diagram of 6-34
  asynchronous requests and
    DMARx 6-66
    chaining, see DMA chaining
clearing extra requests 6-64
clock cycles per data transfer 6-74
concurrent DMA accesses of on-chip memory space 6-74
CP register symbolic address restriction 6-44
data packing, see DMA data packing 6-70
data transfer rate 6-65
direction of data transfers 6-15
DMA address generators (EIEPx and EMEPx registers) 6-75
DMA chain insertion mode, see DMA chain insertion mode
DMA enable (DEN) bit 6-9
DMA interrupts, see DMA interrupts
DMA request/grant latency, handling 6-65
DMA transfer types 6-7
DMAGx grant outputs 6-64
DMARx 6-63, 6-68
DMARx and DMAGx
INDEX

in external handshake mode 6-69
timing, diagram of 6-73
external handshake mode, see
External handshake mode DMA
external transfers and the ECEPx
register 6-63
flushing the DMA request counter
(FLSH) 6-18
handshake mode DMA, see
Handshake mode DMA
handshake timing 6-63
with asynchronous requests
6-66
hardware handshake signals 6-62
hardware interface example,
diagram of 6-72
I/O transfer rate, see DMA I/O
transfer rate
input requests 6-63, 6-64
internal request and grant 6-35
master mode DMA, see Master
mode DMA
MSx lines and external DMA
transfers 6-63
multiprocessing system
configuration for interprocessor
DMA 6-70
overall throughput of multiple
DMA channel memory accesses
6-74
paced master mode DMA, see
Paced master mode DMA
prioritizing external direct accesses
to internal memory 6-37
prioritizing requests 6-35
prioritizing TCB chain loading
6-37
priority of I/O bus accesses 6-74
between processors 6-36
summary of 6-36
program controlled interrupts
6-40
redefining priority for external
port channels 6-38
REDY signal and DMA write
operations through the EPBx
buffers 6-61
request timing 6-65
rotating priority for external port
channels 6-37
setting DMA channel
prioritization 6-35
setting up DMA transfers 6-9
slave mode 6-59
starting a new sequence 6-9, 6-29
starting and stopping a sequence
6-48
three-cycle pipeline and DMARx
6-64
DMA packing order and EPBx
buffers 6-54
DMA parameter registers 6-5
C (count) 6-9, 6-31
channel parameter registers, see
DMA channel parameter registers
CP (chain pointer) 6-30, 6-31,
6-39, 6-40
<table>
<thead>
<tr>
<th>Symbolic Address Restriction</th>
<th>DMA Transfer Modes, See DMA Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defined 6-5</td>
<td>Dmac0 Register</td>
</tr>
<tr>
<td>Ecep (External Count)</td>
<td>Eeprom Booting and 12-49</td>
</tr>
<tr>
<td>Eiep (External Index)</td>
<td>Host Booting and 12-49, 12-57</td>
</tr>
<tr>
<td>EmeP (External Modify)</td>
<td>Initialization After Reset 12-52, 12-56</td>
</tr>
<tr>
<td>External Index Overflow</td>
<td>Parameter Registers Initialization 12-53, 12-57</td>
</tr>
<tr>
<td>Gp (General Purpose) Register</td>
<td>Dmacx Control Registers, See Dmacx Registers</td>
</tr>
<tr>
<td>Ii (Index)</td>
<td>Dmacx Registers 6-5, 6-12</td>
</tr>
<tr>
<td>IM (Modify)</td>
<td>Accessing 6-11</td>
</tr>
<tr>
<td></td>
<td>Address of E-54</td>
</tr>
<tr>
<td>DMA Programming</td>
<td>Bit Definitions 6-14, E-56</td>
</tr>
<tr>
<td>DMA Registers</td>
<td>Chen 6-14, 6-15, 6-39</td>
</tr>
<tr>
<td>Buffer 6-11</td>
<td>Default Bit Values, Diagram of 6-13, E-55</td>
</tr>
<tr>
<td>Control 6-11</td>
<td>Defined 6-5</td>
</tr>
<tr>
<td>DMACx Bit Values, Diagram Of</td>
<td>Den 6-14, 6-15, 7-29, 8-20, 8-28</td>
</tr>
<tr>
<td>6-13</td>
<td>Described 6-11, E-54</td>
</tr>
<tr>
<td>External Port (Dmacx)</td>
<td>Dma Mode Configuration Bit</td>
</tr>
<tr>
<td>Parameter 6-11</td>
<td>Combinations 6-56</td>
</tr>
<tr>
<td>Summary Of 6-11</td>
<td>Dtype 6-14, 6-16</td>
</tr>
<tr>
<td>DMA Request X, See Dmacx</td>
<td>Extern 6-14, 6-19, 6-55, 6-75, 8-21, 8-22</td>
</tr>
<tr>
<td>DMA Sequence</td>
<td>Flsh 6-14, 6-18, 7-29, 8-19</td>
</tr>
<tr>
<td>Defined 6-39</td>
<td>Fs 6-14, 6-18</td>
</tr>
<tr>
<td>Events That Start A 6-48</td>
<td>Host Data Packing Control Bits</td>
</tr>
<tr>
<td>Events That Stop A 6-49</td>
<td>8-28</td>
</tr>
<tr>
<td>Starting A New 6-49</td>
<td>Host Data Transfers And 8-16</td>
</tr>
<tr>
<td>Starting And Stopping 6-48</td>
<td>Host Interface And 8-5</td>
</tr>
<tr>
<td>With Chaining Disabled 6-48</td>
<td>Hshake 6-14, 6-19, 6-55, 8-21, 8-21</td>
</tr>
<tr>
<td>With Chaining Enabled 6-48</td>
<td></td>
</tr>
</tbody>
</table>

---

INDEX
INDEX

8-22
initialization value E-54
INTIO 6-14, 6-17, 6-46, 7-29,
8-20
MASTER 6-14, 6-19, 6-30, 6-55,
8-21, 8-22
MSWF 6-14, 6-17
multiprocessing 7-4
multiprocessing DMA transfers to
internal memory space 7-30
multiprocessing operation 7-25
PMODE 6-14, 6-16, 7-31, 8-22,
8-24, 8-28
PS 6-14, 6-16, 6-54
TRAN 6-14, 6-15, 8-20, 8-28
DMA-driven data transfer mode
9-65
described 9-65
interrupt vector 9-65
DMAGx 6-3, 8-21
DMA grant outputs 6-64
eexternal handshake mode DMA
6-68
handshake for DMA transfers to
external memory space 7-32
handshake mode DMA 6-62,
7-31
host DMA transfers 8-21, 8-22
host DMA transfers through the
EPBx buffers 8-22
multiprocessing 6-68
multiprocessing DMA transfers
7-30
pin definition 12-5
responding to 6-65
state after reset 12-22
DMARx 6-3, 8-21
and external DMA requests 6-63
delay after enabling handshake
DMA 6-68
DMA handshake timing 6-63
extending the DMA grant cycle
6-63
external handshake mode DMA
6-68
handshake for DMA transfers to
external memory space 7-32
handshake mode DMA 6-62,
7-31
host DMA transfers 8-21, 8-22
multiprocessing DMA transfers
7-30
pin definition 12-5
setup time 6-63
state after reset 12-23
three-cycle pipeline 6-64
DMARx and DMAGx timing 6-73
DMASTATx register 6-24
address of E-64
bit definitions 6-24, E-66
default bit values, diagram of E-65
described E-64
DMA chaining status and chain
insertion mode E-64
DMA controller operation and
E-64
initialization value E-65
polling 6-26, 6-47
Index

polling restrictions 6-26, 6-48
reinitializing DMA channels and 6-18
status changes on master processor E-64
status write latency E-64
DO FOREVER instruction 3-12
Do until (type 13) instruction described A-60
example A-60
opcode (relative addressing) A-60
syntax summary A-7
Do until counter expired (type 12) instruction described A-58
example A-58
opcode (with immediate loop counter load) A-58
opcode (with loop counter load from a UREG) A-58
syntax summary A-7
DO UNTIL instruction 3-25
described 3-25
execution sequence 3-25
Instruction pipeline and 3-25
LCE condition 3-12
LCNTR value and 3-35
loop address stack and 3-33
PC stack and 3-25
pipelined loop termination operation 3-26
pipelined loopback operation 3-26
termination condition testing

DQM
defined 10-6
operation 10-27
pin definition 12-10
state after reset 12-22
DRx_X pins 9-4
pin definition 12-11
SPORT loopback mode 9-88
state after reset 12-24
DTx_X pins 9-4
high impedance state 9-67, 9-69
multichannel SPORT mode and 9-69
SPORT loopback mode 9-88
state after reset 12-24
DTYPE (data type) bits 6-14, 9-15, 9-21
and data word width 6-16
and the IMDW bit 6-16
companding format 9-44
data justification 9-44
defined 9-27
described 6-16, 9-44
multichannel operation data formats 9-44
transmit and receive sign extension 9-45
Dual add and subtract instructions, summary of 2-50
Dual add/subtract (fixed-point) ALU status flags B-96
compute field B-96
described B-96
INDEX

Dual add/subtract (floating-point)
   ALU status flags B-98
   compute field B-98
   described B-98
Dual data accesses 5-8
   cache miss 5-10
   DAG1 5-8
   DAG2 5-8
   digital filters 5-9
   DSP applications 5-9
   FFTs 5-9
   instruction cache 5-9
   instruction fetches 5-8
   modified Harvard architecture 5-8
   PM and DM bus addresses 5-8
   PM bus conflicts 5-10
single-cycle execution efficiency 5-9
single-cycle, parallel accesses 5-9

E
   Early frame sync mode
      described 9-56
   ECEPx (DMA external count register) 6-30, 6-32
      and external transfers 6-63
   DMA interrupts 6-9
   EPROM booting and 12-54
   MASTER mode 6-30
Effect latency
   activation of alternate register file
      register sets 2-11
      defined E-4
   SPORT control registers 9-13
      system registers E-4
   EIEPx (DMA external index register) 6-30, 6-31
      address generator 6-75
      generating external addresses for
         DMA transfers 6-55
      overflow 6-30
   EMEPx (DMA external modify register) 6-30, 6-32
      address generator 6-75
      generating external addresses for
         DMA transfers 6-55
   EMU
      pin definition 12-19
      state after reset 12-25
   Emulation status, see EMU
   Enabling DMA operation 6-9
   Enabling standard SPORT mode 9-59
End-of-line termination 12-43
   diagram of 12-43
   propagation delay 12-43
Entering and exiting self-refresh
   mode 10-28
   EP0I interrupt
      function and priority 9-6
      host booting 12-58
   EP1I interrupt
      function and priority 9-6
   EPBx buffers 6-5
      additional parameter registers
         6-30
      architecture 7-27
INDEX

associated DMA channels 6-50
blocked condition in DMA reads
and writes 6-67
bus connections 6-50, 8-18
clearing 6-50, 7-29
core hang 7-29
core read/write restrictions 6-50
DATAx lines used for 32-bit
DMA data 6-53
defined 6-5, 8-5
DMA data packing 6-51
LSWF packing format 6-52
MSWF packing format 6-52
packing logic 6-51
status 6-54
DMA packing modes, summary of 6-52
DMA transfer rate 6-50
DMA transfers to internal
memory space 7-30
ECEP (external count) register 6-30
EIEP (external index) register 6-30
overflow 6-30
EMEP (external modify) register 6-30
EPB0 and host booting 12-58
extending DMA access of internal
memory space 7-30
external port DMA 6-50
external port DMA channels and
DMA transfers to external
memory space 7-31
flushing (FLSH) 6-18, 6-51, 8-19
generating external addresses 6-55
handshake mode 6-62
host data transfers, see Host data transfers
host DMA transfers
see Host DMA transfers
host interface 8-5
host IOP register writes 8-17
host reads of an empty buffer 8-19
host writes 8-16, 8-18
HSHAKE 8-21
internal bus connections 7-27
MASTER 8-21, 8-22
multiprocessing 7-4, 7-26, 7-30
see also Multiprocessing EPBx transfers
non-DMA, single-word transfers 6-50
number of short words currently packed in 6-54
packing 48-bit internal words 6-53
packing status of DMA data transfers 6-16
packing/unpacking individual data words
HBW 8-19
ports 6-50, 8-18
processor writes to a full buffer 8-19
PS (DMACx registers) DMA packing status 6-54
INDEX

reading from an empty buffer 7-28
REDY signal and DMA write operations 6-61
setting up DMA transfers to internal memory space 8-21
size of 6-50
slave mode DMA 8-21
write latency 8-17, 8-18
writing to a full buffer 7-28
EPBX data packing
16- to 48-bit packing 8-35
32- to 48-bit packing 8-34
32-bit data 8-31
48-bit instructions 8-34
8- to 48-bit packing 8-35
DMACx control bits 8-28
summary of 8-28
host reads of 32-bit data 8-31
host writes of 32-bit data 8-33
SYSCON control bits 8-25
EPROM boot mode
ADDRx 12-54
BMS 12-51, 12-53
boot sequence and kernel loading 12-51, 12-54
bootstrapping 256 word instructions 12-52
BSEL 12-51
CS 12-51
data bus alignment 5-53
DATA<7:0> 12-51
described 12-51
DMAC0 register 12-49, 12-52,
12-53
EPROM chip select 12-53
external memory space address of first instruction 12-49
external port data (EPD) lines 12-53
generating EPROM addresses 5-53
MSx chip select line 5-53
multiprocessing 12-51
pin configuration 12-51
pin connections 5-53, 12-51
program counter address at reset 12-53
reset start-up sequence 12-53
RTI instructions 12-54
see also Host booting
wait states and 12-52
wait states configuration 5-53
EPROM boot select, see BSEL
EPROM booting
accessing EPROM after bootstrap 12-55
ACK 12-52
ADDRx 12-54
BMS and 12-53
boot hold off 12-52
BSO bit 12-55
DATAx 12-54
DMA controller operation 12-54
DMA count register and 12-54
EPROM chip select 12-53
external port data (EPD) lines 12-53
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt vector table, locating</td>
<td>12-61</td>
</tr>
<tr>
<td>Loading remaining EPROM data</td>
<td>12-55</td>
</tr>
<tr>
<td>Multiprocessing</td>
<td>12-59</td>
</tr>
<tr>
<td>Overriding BMS</td>
<td>12-55</td>
</tr>
<tr>
<td>Program counter address at reset</td>
<td>12-53</td>
</tr>
<tr>
<td>Reset start-up sequence</td>
<td>12-53</td>
</tr>
<tr>
<td>RTI instructions</td>
<td>12-54</td>
</tr>
<tr>
<td>Writing to BMS memory space</td>
<td>12-56</td>
</tr>
<tr>
<td>EQ condition</td>
<td>3-13</td>
</tr>
<tr>
<td>Execute cycle</td>
<td>3-4</td>
</tr>
<tr>
<td>Executing program from external memory space</td>
<td>5-50</td>
</tr>
<tr>
<td>40-bit data accesses</td>
<td>5-52</td>
</tr>
<tr>
<td>Aligning internal addresses with external memory space</td>
<td>5-50</td>
</tr>
<tr>
<td>Data access addressing</td>
<td>5-52</td>
</tr>
<tr>
<td>Data packing</td>
<td>5-49</td>
</tr>
<tr>
<td>Described</td>
<td>5-49</td>
</tr>
<tr>
<td>Example addresses for</td>
<td>5-50</td>
</tr>
<tr>
<td>External memory address generation scheme</td>
<td>5-51</td>
</tr>
<tr>
<td>Generating instruction addresses in external memory space</td>
<td>5-50</td>
</tr>
<tr>
<td>Invalid segment addresses</td>
<td>5-52</td>
</tr>
<tr>
<td>Mapping 64K memory space to 128K memory space</td>
<td>5-51</td>
</tr>
<tr>
<td>Multiple program segments, using</td>
<td>5-51</td>
</tr>
<tr>
<td>PM bus address restriction</td>
<td>5-52</td>
</tr>
<tr>
<td>Program segment alignment in external memory space</td>
<td>5-51</td>
</tr>
<tr>
<td>Storing instructions in internal memory space</td>
<td>5-50</td>
</tr>
<tr>
<td>Execution stalls</td>
<td>12-66</td>
</tr>
<tr>
<td>Extended-precision, floating-point format</td>
<td>C-4</td>
</tr>
<tr>
<td>Diagram of</td>
<td>C-4</td>
</tr>
<tr>
<td>Significant, size of</td>
<td>C-4</td>
</tr>
<tr>
<td>Size of</td>
<td>C-4</td>
</tr>
<tr>
<td>EXTERN (DMA external handshake mode enable) bit</td>
<td>6-14, 6-75, 8-21, 8-22</td>
</tr>
<tr>
<td>And the direction of DMA transfers</td>
<td>6-16</td>
</tr>
<tr>
<td>Described</td>
<td>6-19</td>
</tr>
<tr>
<td>DMA transfers to on-chip memory</td>
<td>7-31, 7-32</td>
</tr>
<tr>
<td>External (off-chip) memory external memory space</td>
<td>5-43</td>
</tr>
<tr>
<td>External port and</td>
<td>5-43</td>
</tr>
<tr>
<td>Interface pins</td>
<td>5-43</td>
</tr>
<tr>
<td>Interfacing with</td>
<td>5-43</td>
</tr>
<tr>
<td>External bus ADDRx and DATAx</td>
<td>8-2</td>
</tr>
<tr>
<td>Defined</td>
<td>8-5</td>
</tr>
<tr>
<td>Host interface and</td>
<td>8-2</td>
</tr>
<tr>
<td>Multiprocessing and</td>
<td>7-4</td>
</tr>
<tr>
<td>External bus address, see ADDRx</td>
<td></td>
</tr>
<tr>
<td>External bus data, see DATAx</td>
<td></td>
</tr>
<tr>
<td>External handshake mode DMA ACK</td>
<td>6-69</td>
</tr>
<tr>
<td>Configuration</td>
<td>6-20, 6-69, 7-32</td>
</tr>
<tr>
<td>Configuring DMA channels</td>
<td>6-69</td>
</tr>
</tbody>
</table>
INDEX

data packing 6-70
described 6-55, 6-68
disabling DMA interrupts 6-69
DMA transfers to external memory space 7-32
DMARx and DMAGx 6-69
handshaking signals 6-68
EXTERN bit 7-32, 8-22
external memory access, behavior of 6-69
generating external memory access cycles 6-69
generating external memory address and word count 6-69
host transfers to external memory space 8-22
HSHAKE bit 7-32, 8-22
MASTER bit 7-32, 8-22
MSx, RD, and WR 6-69
multiprocessing DMA transfers to external memory space 7-31
transfers between an external device and external memory space 6-75
External interrupts
asynchronous 3-51
edge-triggered 3-51
level-sensitive 3-50
programmable timer pins and sample timing 3-50
sensitivity option, setting 3-51
validity of 3-50
External memory banks
address locations 5-49
address space 5-48
bank 0 address space limitation 5-48
bus hold time cycle, see Bus hold time cycle
bus idle cycle, see Bus idle cycle
conditional memory write instructions 5-49
decoded memory address lines 5-49
described 5-48
DMA handshake wait state 5-57
EBxWS bit values 5-60
MSx lines 5-48
multiprocessor memory space wait state 5-57
number of wait states (EBxWS) 5-56
peripheral chip selects (MSx) 5-49
ROM boot wait mode 5-56
ROM boot wait state 5-57
running code from 5-48
SDRAM and wait states, see also SDRAM interface
SDRAM mapping 5-48
WAIT register, see WAIT register
wait state configuration 5-55
wait state generator 5-48
wait state mode (EBxWM) bits 5-56, 5-61
wait state modes 5-61
external memory banks vs. memory blocks 5-49
External memory space
access address fields 5-26
access timing 5-65
bus master reads 5-66
bus master writes 5-67
diagram of 5-65
external bus control 5-65
address boundaries 5-19
address of first instruction, no boot mode 12-49
address space 5-44
banks, see External memory banks
bus idle cycle, see Bus idle cycle
bus master writes 5-67
concurrent DMA accesses and wait states 6-74
defined 5-3
described 5-26
diagram of 5-26
DMA data transfers between external devices and external memory 6-8
DMA transfers 8-21
EPROM booting 12-49
host booting 12-49
host DMA transfers 8-21, 8-22
interface with external memory devices 5-43
mixed word storage 5-43
multiprocessor DMA transfers to 7-31
off-chip memory access extension 5-53, 5-54
packed word storage 5-43
program execution, see Program execution
response to ACK 5-53
running code from 5-48
SDRAM, see SDRAM interface
setting up host DMA transfers 8-22
wait states and acknowledge
see Wait states and acknowledge
external memory space
memory blocks vs. memory banks 5-49
suspending bus three-state (SBTS) and SDRAMs 5-63
External memory space accesses
DM bus 5-26
EP bus 5-26
external port 5-26
internal buses 5-44
PM bus 5-26
External memory space interface signals 5-44
ACK 5-47
ADDRx 5-44
DATAx 5-45
MSx 5-45
RD 5-45
SWx 5-46
WR 5-46
External memory wait state control register, see WAIT register
External port 12-4
ADDRx 12-4
buses 8-18
INDEX

data alignment, diagram of 12-21
data lines (EPD) and EPROM
  boot sequence after reset 12-53
DATAx 12-4
defined 5-3
DMA data transfers 7-30
DMAGx 12-5
DMARx 12-5
host interface and 8-2
MSx 12-5
multiprocessing data transfers
  7-25, 7-27
pin definitions 12-4
SBTS 12-6
SW 12-6
External port buffer 0 interrupt 9-6
External port buffer 1 interrupt 9-6
External port DMA
  block data transfers 6-7
  buffer size 6-50
  changing DMA channel priority
    assignment, example of 6-38
  channels 6-30, 6-50
  clearing EPBx buffers 6-50
  connection to internal memory
    space 6-27
  control bit definitions 6-14
  core read/write of EPBx buffers,
    restrictions 6-50
  data packing 6-51
    LSWF packing format 6-52
    MSWF packing format 6-52
    packing logic 6-51
  PMODE and HBW combina-
  tions, summary of 6-52
  described 6-50
  disabling 6-67
  DMA registers 6-12
  EPBx buffers 6-50
  fixed channel priority 6-38
  internal DMA request and grant
    6-35
  interrupts 6-45
  master mode DMA interrupts, see
    DMA interrupts 6-45
  modes, see DMA modes 6-55
  non-DMA, single-word transfers
    6-50
  priority of TCB chain loading, see
    TCB chain loading
  redefining DMA channel priority
    6-38
  re-enabling 6-67
  rotating channel priority 6-37,
    6-38
  transfer rate 6-50
External port DMA control
  registers, see DMACx registers
External port FIFO buffers, see
  EPBx buffers
EZ-ICE emulator
  board-level testing 12-38
  CLKIN connection 12-40
  connection requirements 12-36
  described 12-36
  executing synchronous
    multiprocessor operations
    12-40

I-38        ADSP-21065L SHARC DSP User’s Manual
INDEX

JTAG interface and 12-36
pin connections in nontesting
environments 12-38
probe 12-36
scan path, diagram of 12-40
signal termination 12-39
target board connector 12-36
see EZ-ICE target board connector
EZ-ICE target board connector
diagram of 12-38
pin strip header 12-37
specifications 12-37

F
FDEP bit field deposit instruction
2-43
bit field, diagram of 2-43
example, diagram of 2-44
Fetch address register 3-6
Fetch cycle 3-4
FEXT bit field extract instruction
2-43
example, diagram of 2-45
FEXT Rx BY Ry operation
described B-82
example B-83
shifter status flags B-83
Fixed priority for external port
channels 6-38
Fixed-point formats 2-7
32-bit formats, diagram of C-8
64-bit signed products, diagram of C-10
64-bit unsigned products,
diagram of C-9
ALU data and C-9
described C-8
fractional format C-8
multiplier data C-9
types C-8
Fixed-point MR register operations
clear MR register 2-30
described 2-30
rounding MR register 2-30
saturate MR register 2-31
Fixed-point multiplier results, see Multiplier fixed-point results
Fixed-point multiply and accumulate instructions,
summary of 2-51
Fixed-point operations
ALU inputs 2-13
ALU results 2-13
ALU single-function compute operations, summary of B-3
operands and results, format of 2-13
results, format of 2-13
Fixed-point saturation 2-14
Fixed-point to floating-point conversions 2-15
Flag inputs, see FLAGx 12-31
Flag outputs, see FLAGx 12-33
Flag pins, see FLAGx
FLAG0_IN condition 3-13
FLAG1_IN condition 3-13
FLAG2_IN condition 3-13
INDEX

FLAG3_IN condition 3-13
FLAGx
and core accesses of the system bus 8-48
bit states and conditional instructions 12-32
core and status registers 12-29, 12-30
inputs 12-31
operation cycles 12-26
output timing, diagram of 12-34
outputs 12-33
pin definition 12-16
programming the direction of FLAG11-4, 12-30
programming the direction of FLAG3_0 12-29
signaling external devices 12-33
single-bit signaling and 12-28
state after reset 12-24
status updates 12-31
Floating point DSP 1-8
dynamic range 1-8
ease-of-use 1-8
precision 1-8
signal-to-noise ratio 1-8
Floating-point data rounding bit, see RND32 bit
Floating-point formats
exception handling 2-6
extended-precision 2-5
extended-precision width 2-5
IEEE 754/854 standard
compatibility exceptions 2-4
short word 2-5
short word conversions from 32-bit words 2-5
short word using gradual underflow 2-6
single-precision 2-4
single-precision NAN inputs 2-4
single-precision rounding modes 2-5
single-precision, IEEE 754/854 standard 2-4
Floating-point multiply and ALU instructions, summary of 2-51
Floating-point operation exception handling
immediate corrections with interrupts 2-6
monitoring a single operation with ASTAT flags 2-6
monitoring results from multiple operations with STKY flags 2-7
Floating-point operations
ALU single-function compute operations, summary of B-4
exception handling 2-6
extended precision 2-5
FLSH (DMA flush buffers and status) bit 6-14
clearing extra DMA requests 6-64
described 6-18
flushing the EPBx buffers 6-51, 7-29
restriction 8-19
Fn= -Fx (floating-point) operation
ALU status flags B-32
  described B-32
Fn=(Fx+Fy)/2 (floating-point)
  operation
ALU status flags B-30
  described B-30
Fn=ABS (Fx+Fy) (floating-point)
  operation
ALU status flags B-28
  described B-28
Fn=ABS (Fx-Fy) (floating-point)
  operation
ALU status flags B-29
  described B-29
Fn=ABS Fx (floating-point)
  operation
ALU status flags B-33
  described B-33
Fn=CLIP Fx BY Fy operation
  ALU status flags B-49
  described B-49
Fn=FLOAT Rx BY Ry operation
  ALU status flags B-41
  described B-41
Fn=FLOAT Rx operation
  ALU status flags B-41
  described B-41
Fn=FUNPACK Rx operation
  described B-92
  gradual underflow B-92
results of B-92
  shifter status flags B-93
Fn=Fx COPYSIGN Fy operation
  ALU status flags B-46
  described B-46
Fn=Fx*Fy operation
  described B-62
multiplier status flags B-62
Fn=Fx+Fy (floating-point)
  operation
ALU status flags B-26
  described B-26
Fn=Fx-Fy (floating-point)
  operation
ALU status flags B-27
  described B-27
Fn=MAX (Fx, Fy) operation
  ALU status flags B-48
  described B-48
Fn=MIN (Fx, Fy) operation
  ALU status flags B-47
  described B-47
Fn=PASS Fx (floating-point)
  operation
ALU status flags B-34
  described B-34
Fn=RECIPS Fx operation
  ALU status flags B-43
  described B-42
Fn=RND Fx (floating-point)
  operation
ALU status flags B-35
  described B-35
Fn=RSQRTS Fx operation
  ALU status flags B-45
  described B-44
Fn=SCALB Fx BY Ry
  (floating-point) operation
INDEX

ALU status flags B-36
  described B-36
FOREVER condition 3-12, 3-15
FPACK instruction C-5
  conversion results C-6
  overflow condition, effects of C-7
Frame sync active level
  operation with LTFS/RTFS cleared 9-55
  operation with LTFS/RTFS set 9-55
Frame sync active state 9-55
Frame sync clock edge 9-55
Frame sync configuration 9-59
  both transmitters transmitting simultaneously 9-60
  continuous simultaneous transmission 9-60
  described 9-59
  enabling simultaneous transmission 9-60
  FS_BOTH values 9-59
Frame sync data dependency 9-57
  described 9-57
  operation with DITFS cleared 9-58
  operation with DITFS set 9-58
  timing of internally-generated TFS 9-57
Frame sync insert 9-56
  early frame sync mode 9-56
  frame signal timing modes, example of 9-57
  multichannel SPORT mode 9-56
  normal vs. alternate frame, diagram of 9-57
  operation with LAFS cleared 9-56
Frame sync options 9-52
  I²S SPORT mode 9-63
  multichannel SPORT mode 9-69
  word select signals 9-63
Frame sync requirement
  continuous output and 9-52
  described 9-52
  DMA chaining and 9-53
  framed serial transfers, example of 9-53
  framed vs. unframed data, diagram of 9-54
  initiating communications and 9-53
  operation with RFSR/TFSR cleared 9-53
  operation with RFSR/TFSR set 9-52
Frame sync source
  described 9-54
  frame sync divisors 9-54
  operation with ITFS/RTFS cleared 9-54
  operation with ITFS/RTFS set 9-54
Frame synchronization 9-5
  FS (DMA external port buffer status) bits 6-14
  described 6-18
  status values 6-19

I-42       ADSP-21065L SHARC DSP User’s Manual
INDEX

FS_BOTH (frame sync both) bit
9-17
defined 9-28
described 9-63
standard SPORT mode 9-59
word select signal 9-64
Full-page burst length (SDRAM) 10-18
FUNPACK instruction C-5
conversion results C-6

G
GE condition 3-14
General loop restrictions 3-27
last three instructions in 3-27
nested loops 3-27
Generating addresses for the PM and DM buses 5-11
Generating addresses outside the address range of external memory space 6-30
Generating internal and external addresses for DMA transfers 6-55
GND, pin definition 12-20
GP (DMA general purpose) register 6-31
and DMA sequences 6-41
loading 6-41
Gradual underflow C-7
Ground planes 12-46
Group I (compute and move) instructions
compute (type 2) instruction A-32
compute (type 2) instructions A-28
compute/dreg⇔DM/dreg⇔PM (type 1) instruction A-30
compute/dreg⇔DM/dreg⇔PM (type 1) instructions A-28
compute/dreg⇔DM|PM, immediate modify (type 4) instruction A-35
compute/dreg⇔DM|PM, immediate modify (type 4) instructions A-28
compute/modify (type 7) instruction A-42
compute/modify (type 7) instructions A-29
compute/ureg⇔DM|PM, register modify (type 3) instruction A-33
compute/ureg⇔DM|PM, register modify (type 3) instructions A-28
compute/ureg⇔ureg (type 5) instruction A-37
compute/ureg⇔ureg (type 5) instructions A-28
IF COND A-29
immediate Shift/dreg⇔DM|PM (type 6) instruction A-39
immediate Shift/dreg⇔DM|PM (type 6) instructions A-28
summary A-28
Group II (program flow control) instructions
INDEX

direct jump|call (type 8) instruction A-45
do until (type 13) instruction A-60
do until counter expired (type 12) instruction A-58
IF COND A-44
indirect jump or compute/dreg⇒DM (type 10) instruction A-52
indirect jump|call/compute (type 9) instruction A-48
return from subroutine|interrupt/compute (type 11) instruction A-55
summary A-44
Group III (immediate move) instructions
immediate data⇒DM|PM (type 16) instruction A-67
immediate data⇒ureg (type 17) instruction A-69
summary A-62
ureg⇒DM|PM (direct addressing) (type 14) instruction A-63
ureg⇒DM|PM (indirect addressing) (type 15) instruction A-65
Group IV (miscellaneous) instructions
Cjump/Rframe (type 24) instruction A-81
IDLE (type 22) instruction A-78
IDLE16 (type 23) instruction A-79
NOP (type 21) instruction A-77
pop stacks/flush cache (type 20) instruction A-75
register modify/bit-reverse (type 19) instruction A-73
summary A-70
system register bit manipulation (type 18) instruction A-71
GT condition 3-14

H
Handshake mode DMA 6-20
configuration 7-31
data source and destination selection 6-63
described 6-55, 6-62
DMAGx 6-62, 7-31, 8-22
DMARx 6-62, 7-31, 8-22
enabling 6-63
EXTERN bit 7-31, 8-22
external transfers and the ECEPx register 6-63
hardware handshake signals 6-62
host data transfers to internal memory space 8-22
HSHAKE bit 7-31, 8-22
MASTER bit 7-31, 8-22
multiprocessing DMA accesses of internal memory 7-31
Hardware SPORT reset 9-8
HBG
and host signal buffers 8-9
INDEX

host interface  8-8
multiprocessor bus arbitration 7-10
pin definition  12-8
state after reset 12-22

HBR
BCNT register and 7-18
host booting 12-58
host interface 8-8
maintaining host bus mastership 8-10
multiprocessor booting 12-59
multiprocessor bus arbitration 7-10
pin definition 12-9
relinquishing the bus 8-11
resolving system bus access
deadlock 8-49
signal glitches, avoiding 8-46
state after reset 12-24

HBW (host bus width) bits 8-22, 8-24, 8-26
changing the initialization-after-reset value 8-26
changing the packing mode 12-57
EPBx packing modes 6-16
external port DMA packing mode 6-51
host boot mode 12-57
host data transfers 8-24
host EPBx packing modes 8-19
host EPBx transfers 8-24
packing individual data words 8-19

High frequency design issues 12-42
clock distribution 12-43
clock specifications and jitter 12-42
clock with two frequency inputs,
diagram of 12-42
controlled impedance
transmission line 12-43
crosstalk, reducing 12-45
decoupling capacitors and ground planes, see Decoupling capacitors and ground planes
end-of-line termination, see End-of-line termination
oscilloscope probes, see Oscilloscope probes
point-to-point connections on serial ports, see Point-to-point connections on serial ports
propagation delay 12-43
reflections, reducing 12-46
signal integrity, see Signal integrity
source termination, see Source termination

HMSWF (host packing order) bit 6-54, 8-27
and 48-bit DMA words 6-53

Host asynchronous accesses
broadcast writes
see Broadcast writes
buses used for 8-16
CS 8-11
host interface buffers 8-12
INDEX

in multiprocessor systems 8-14
initiating 8-16
maximum throughput, reads 8-15
rate of 8-15
read cycle sequence 8-15
read/write example timing,
diagram of 8-13
REDY, see REDY
timing 8-11
\( t_{\text{RDYHG}} \) switching characteristic
and transfer timing 8-12
write cycle sequence 8-14
Host boot mode
boot sequence and kernel loading
12-51, 12-54
booting sequence 12-58
described 12-56
DMAC0 register 12-49, 12-57
external memory space address of
first instruction 12-49
HBR 12-58
pin configuration 12-51
see also Host booting
Host booting 12-56
BMS 12-56
boot sequence 12-58
BSEL 12-56
DATA\(_{15:0}\) 12-58
DMA controller operation 12-58
DMA data packing 12-57
DMA done interrupt 12-58
DMAC0 initialization after reset
12-56
HBR 12-58
interrupt vector table, locating
12-61
multiprocessing 12-59
pin configuration 12-56
reset boot sequence 12-56
RTI instruction 12-58
slave processor mode 12-56
writing directly to EPB0 12-58
writing to the IOP registers 12-58
Host bus acknowledge, see REDY
Host bus acquisition 8-8
accessing the processor 8-8
BRx 8-8
example timing, diagram of 8-10
HBG 8-8
HBR 8-8
host signal buffers 8-9
HTC 8-8
restrictions 8-10
SBTS 8-11
Host bus grant, see HBG
Host bus mastership
avoiding temporary loss of 8-10
HBG 8-8
HBR 8-8, 8-10
REDY 8-8
relinquishing the bus 8-11
Host bus request, see HBR
Host control of processor
asynchronous transfers 8-9
and SDRAM 8-9
\( CS \) 8-9
host driven signals 8-9
relinquishing the bus 8-11
INDEX

Host data packing 8-24
  16- to 48-bit packing 8-35
  32- to 48-bit packing 8-34
  32-bit data 8-31
  32-bit data reads 8-31
  32-bit data writes 8-33
  48-bit instructions 8-34
  8- to 48-bit packing 8-35
  changing the value of HBW 8-26
  diagram of 8-32
  for all IOP register accesses, except
    the EPBx buffers 8-24
  for EPBx accesses 8-24
  for non-EPBx IOP registers 8-24
  HBW 8-24
  individual data words 8-24
  packing/unpacking individual
    data words 8-19
  PMODE 8-19, 8-22, 8-24
  specifying host bus width 8-24
Host data transfers 8-16
  accessing the processor (CS) 8-11
  addressing 8-16
  addressing an IOP register 8-11
  ADDRx bits host must drive 8-11
  BHD (buffer hang disable) bit
    8-19
  communication with processor’s
    core 8-16
  control and configuration of
    processor operation 8-16
  core hang 8-19
  CS 8-16
  data packing, see Host data

packing
  defined 8-5
DMA transfers, see Host DMA
  transfers
DMA transfers, setting up 8-16
EPBx buffers, see EPBx buffers
EPBx writes 8-18
full speed asynchronous writes
  8-15
functions 8-16
handshake mode DMA 8-22
HSHAKE 8-21, 8-22
initiating 8-16
IOP register reads 8-17
IOP register writes 8-16
  cycles to complete 8-17
  maximum throughput 8-17
  slave write FIFO 8-16
MASTER 8-21, 8-22
maximum throughput, reads 8-15
rate of asynchronous writes 8-15
read cycle sequence 8-15
read/write cycle example timing,
  diagram of 8-13
resynchronizing previously
  written words 8-15
single-word 8-18, 8-19, 8-20
single-word, non-DMA 8-20
slave mode DMA 8-21
slave write FIFO 8-15
through the EPBx buffers, see
  Host EPBx transfers
transferring data 8-16
  types 8-18
INDEX

with on-chip memory 8-21
write cycle sequence 8-14
Host DMA transfers 8-21
block data 8-18
data packing and PMODE 8-22
DMAGx 8-21
DMARx 8-21
EXTERN 8-21
external memory space accesses 8-21, 8-22, 8-23
handshake mode DMA 8-22
HSHAKE 8-21, 8-22
internal memory space accesses 8-21, 8-22
MASTER 8-21, 8-22
setting up DMA transfers to on-chip memory 8-21, 8-22
slave mode DMA 8-21
Host EPBx transfers 8-18
BHD (buffer hang disable) bit 8-19
broadcast writes 8-23
see Broadcast writes
core hang 8-19
data packing, see Host data packing
DATAx and 8-30, 8-31
DMA data packing
see also Host data packing
DMA transfers
see also Host DMA transfers
DMACx packing control bits, summary of 8-28
HBW bit values, changing 8-29
host reads of an empty buffer 8-19
packing mode
bit combinations, summary of 8-24
HBW 8-24
PMODE 8-24
processor writes to a full buffer 8-19
setting up DMA transfers to internal memory space 8-21
single-word 8-18, 8-19, 8-20
single-word non-DMA 8-20
slave write FIFO 8-18
types 8-18
write latency 8-18
Host interface 8-1
accesses and operation cycles 12-26
accessing a processor 8-8, 8-11
accessing slave processors over the cluster bus 8-44
ACK 12-7
arbitration for control of the system bus 8-44
asynchronous transfer timing
see also Host asynchronous accesses
asynchronous writes, rate of 8-15
basic system bus/cluster bus interface, diagram of 8-45
bidirectional system bus interface, diagram of 8-47
BRx and host bus acquisition 8-8
buffers 8-12
INDEX

bus acquisition example timing, diagram of 8-10
cluster bus 8-44
core accesses of the system bus 8-48
CS 12-8
data bus lines and host bus width 8-30
data packing, see Host data packing
data transfers, see Host data transfers
diagram of 8-2
DMA data transfers, see Host DMA transfers
DMACx registers, see DMACx registers
external bus accesses 8-2, 8-5
external port and 8-2
features 8-1
HBG signal 8-8, 12-8
HBR signal 8-8, 12-9
HBW bit values, changing 8-29
host control
see Host bus acquisition
see Host bus mastership
host transfers, see Host data transfers, Host DMA transfers, and Host EPBx transfers
HTC (host transition cycle) 8-6
immediate high-priority interrupt 8-36
interprocessor messages 8-36
see Interprocessor messages
interrupt service routine 8-36
IOP registers, see IOP registers
local bus, defined 8-6
maximum throughput, reads 8-15
memory mapping 8-2
message passing 8-36
multiprocessor memory space and 8-6
physical connection to 8-2
pin definitions 12-7
processor, defined 8-6
REDY 12-9
SBTS and 8-4
signal glitches on the HBR line, avoiding 8-46
single-word data transfers, defined 8-6
slave processor, defined 8-7
suspending a processor’s active access of the system bus 8-50
SYSCON, see SYSCON register
system access of slave processors 8-46
system bus access deadlock, see System bus access deadlock
system clock cycle, references to 8-7
vector interrupts 8-36
Host interface pins
CS 8-3
HBR 8-3
REDY 8-4
summary of 8-3
INDEX

Host interface signals
  chip select 8-3
  host bus acknowledge 8-4
  host bus grant 8-3
  host bus request 8-3
  summary of 8-3
Host to processor, 8- to 48-bit word packing 6-54
Host transition cycle, see HTC
Host vector interrupts 8-38
  generating 8-38
  interrupt service routine 8-38
  interrupt service routines 8-38
  servicing 8-38
Host, defined 8-5
HPFLSH (host packing status flush) bit 8-27
HPS (host packing status) bit 7-43, 8-42
HSHAKE (DMA handshake mode enable) bit 6-14, 8-21, 8-22
  described 6-19
  DMA transfers to on-chip memory 7-31, 7-32
HSTM (host mastership) bit 7-41, 8-40
HTC 8-8, 8-9
  defined 8-6
Hysteresis
  described 12-41
  RESET 12-41

I
I (DAG index) registers 4-2

bit-reverse instruction and 4-14
  circular buffer addressing and 4-9
  circular data buffers and 4-11
  immediate modifiers 4-8
  postmodify addressing operations 4-7
  using without a circular data buffer but with circular buffer overflow interrupts enabled 4-13

I/O bus
  and DMA operations 6-27
  and the EPBx buffers 8-18
  data transfers with memory 5-7
  defined 5-4
  generating addresses for 32-bit addresses 5-26
  memory accesses 5-27
I/O interrupts, causes of 6-47
I/O processor 7-26
  see IOP registers
I²S SPORT mode
  control bits 9-62
  data word length and the frame sync divisor 9-63
  data word length capability 9-63
  default bit values, diagram of 9-19, 9-24
  default channel order 9-63
  described 9-41
  DITFS 9-26
  DTYPE 9-28
  dual transmitter operation 9-64
  enabling 9-62
INDEX

frame sync data dependency in
9-57
I²S bus architecture 9-61
Inter-IC sound bus protocol 9-61
L_FIRST 9-30
loopback mode 9-88
MSTR 9-31
operation capabilities, summary of
9-61
OPMODE 9-32, 9-36
PACK 9-32
RCLKDIV 9-62
receive control bits 9-21
ROVF 9-33
RXS 9-33
SCHEN 9-34
SDEN 9-34
setting the frame sync options
9-63
see Frame sync options
setting the internal serial clock rate
9-61
setting the transmit and receive
channel order 9-63
setting the word length 9-63
SLEN 9-35
SPEN 9-35
SPL 9-36
SPORT DMA enabling 9-65
see also SPORT_DMA
SPORT master mode, enabling
9-64
TCLKDIV 9-62
transmit control bits 9-15
TUVF 9-36
TXS 9-37
word select timing, diagram of
9-66
ICLK (transmit and receive clock
sources) bit 9-16, 9-21
clock signal options 9-50
IDC (ID code) bit 7-42, 8-41
IDLE (type 2) instruction 3-1
described 3-56
execution sequence 3-56
exiting 3-56
internal clock and timer operation
during 3-56
interrupt servicing and 3-38
IDLE (type 22) instruction
described A-78
opcode A-78
IDLE16 (type 23) instruction
application exits A-79
application software exits from
3-56
described 3-56, A-79
DMA transfers A-79
execution sequence 3-56
exiting 3-56
host accesses A-79
internal clock and timer operation
during 3-56
interrupt servicing and 3-38
multiprocessing A-79
nonsupported accesses A-80
opcode A-80
restrictions 3-56
INDEX

unsupported operations 3-57
IDx
bus synchronization and 7-11
connections in a multiprocessor system 7-3
multiprocessor bus arbitration 7-10
pin definition 12-16
state after reset 12-24
IEEE rounding modes 2-15
IF NOT LCE instruction 3-13
IF TRUE instruction 3-12
II (DMA index register) 6-31
IIVT (internal interrupt vector table) bit
boot modes 12-61
overriding the boot mode F-3
IM (DMA modify register) 6-31
IMASK register 3-46, 6-40, 9-9
accessing through the external port 6-47
and the VIRPT register 6-47
bit definitions E-14
bit values after reset 3-46
default bit values, diagram of E-13, F-5
described 9-73, E-12
disabling DMA interrupts 6-40, 6-69, 7-29
disabling interrupts 8-20
DMA interrupts 6-45
EP0I 3-46
host booting and 12-58
initialization value E-12
interrupt vectors and priorities F-1
IRPTL register bits and 3-44
masking interrupts 3-46
memory-mapped address and reset value 9-11, 9-12
multichannel receive comparison mask 9-73
RESET restrictions 3-46
IMASKP register 3-46
bit definitions 3-47
generation of new temporary interrupt masks 3-47
interrupt priority and 3-47
nesting interrupts 3-46
RTI instruction and 3-16
temporary masks for nested interrupts 3-47
IMAT (receive comparison accept data) bit 9-22
defined 9-28
receive comparisons and 9-74
IMDWx (internal memory block data width) bit 8-27
and word width of DMA data transfers 6-16
changing value of 5-40
RND32 and 5-41
Immediate addressing 5-11
Immediate DAG modifiers 4-8
instructions with parallel operations 4-8
magnitude of values 4-8
Immediate data⇒DM|PM (type 16) instruction
  described A-67
  example A-67
  opcode A-67
  syntax summary A-8
Immediate data⇒ureg (type 17) instruction
  described A-69
  example A-69
  opcode A-69
  syntax summary A-8
Immediate high-priority interrupt,
  see Vector interrupts
Immediate modifier value
  postmodify addressing operations 4-7
  premodify addressing operations 4-6
  width of 4-7
Immediate move instructions
  immediate data⇒DM|PM (type 16) instructions A-8
  immediate data⇒ureg (type 17) instructions A-8
  summary A-8
  ureg⇒DM|PM (direct addressing) (type 14) instructions A-8
  ureg⇒DM|PM (indirect addressing) (type 15) instructions A-8
Immediate Shift/dreg⇒DM|PM (type 6) instruction
  example A-39
  opcode (with data access) A-40
  opcode (without data access) A-40
  syntax summary A-8
IMODE (receive comparison enable) bit 9-21
defined 9-29
receive comparisons and 9-74
Indirect addressing 5-11
  DAG1 and the DM bus 5-11
  DAG2 and the PM bus 5-11
  postmodify with immediate value A-18
  postmodify with M register, update Iregister A-18
  premodify with immediate value A-18
  premodify with M register, update I register A-18
Indirect jump or compute/dreg⇒DM (type 10) instruction
  described A-52
  example A-53
  IF COND A-52
  opcode (with indirect jump) A-53
  opcode (with PC-relative jump) A-53
  syntax summary A-6
Indirect jump|call/compute (type 9) instruction
  described A-48
  example A-49
  opcode (with indirect branch)
INDEX

A-50
opcode (with PC-relative branch)
A-50

Individual register file registers
assembly language prefix identifier
2-10
described 2-10
fixed-point computations 2-10
floating-point computations 2-10
Input signal conditioning 12-41
input inverter and 12-41
Input synchronization delay 12-27
Inserting a high priority DMA chain
in an active DMA chain 6-44
Instruction addresses 3-6
Instruction cache
architecture, see Instruction cache
architecture
cache hit 3-58
cache miss 3-58, 5-10
defined 5-3
described 3-58
disable and freeze 3-61
see, Instruction cache disable and
freeze
dual data accesses 5-9
efficiency 3-60
see Instruction cache efficiency
instruction fetches 5-10
operation 3-58, 5-10
operation after reset 3-62
PM bus conflict 5-10
PM data bus accesses 5-10
program memory data accesses
3-10
program sequencing 3-7
size of 3-58
three-instruction pipeline and
3-58
Instruction cache architecture
addressing entry sets 3-59
cache hit 3-59
cache miss 3-59
described 3-58
diagram of 3-59
entry 3-58
entry sets 3-59
entry valid bit 3-59
instruction address mapping 3-59,
3-60
LRU bit 3-59
see LRU (least recently used) bit
Instruction cache disable and freeze
3-61
CADIS 3-62
CAFRZ 3-62
disabling 3-61
freezing 3-61
program memory data access
restrictions and 3-62
Instruction cache efficiency 3-60
bit rate and 3-60
cache misses and 3-60
described 3-60
example of cache-inefficient code
3-60
Instruction cycle 3-4
clock rate 3-4
decode 3-4
execute 3-4
fetch 3-4
pipelined execution cycles 3-5
pipelining 3-4
processing rate 3-4
Instruction fetches 5-8, 5-10
dual data accesses 5-8
over the PM data bus 5-10
PM bus conflict 5-10
through the instruction cache
5-10
word width of 5-28
Instruction pipeline 3-19
DO UNTIL instruction and 3-25
instruction cache and 3-58
loop restrictions and 3-27
short loops and 3-28
Instruction set notation A-11
Instruction set reference
compute and move/modify A-4
see Compute and move/modify
instructions
condition and termination codes,
summary of A-13
conditional instructions A-3
group I instructions
see Group I (compute and move)
instructions A-28
group II (program flow control)
instructions
see also Group II (program flow
control) instructions
summary A-44
group III (immediate move)
instructions A-62, A-70
see Group III (immediate move)
instructions
see Group IV (miscellaneous) in-
structions
group IV instructions A-9
immediate move instructions A-8
see Immediate move instructions
instruction summary A-2
instruction types A-2
map 1 system registers A-25
map 1 universal register codes
A-26
map 1 universal registers A-24
map 2 universal register codes
A-25, A-27
memory addressing A-18
see Memory addressing
miscellaneous instructions A-9
see Miscellaneous instructions
notation summary A-11
opcode notation, summary of
A-19
program flow control A-6
see Program flow control instruc-
tions
register types, summary of A-15
universal register codes, summary
of A-24
Instructions
conditional 3-12
conditional and FLAGx bit states
12-32
INDEX

conditional memory writes 5-49
internal memory storage 5-50
pipeline 3-19
INT_HIx (timer interrupt vector location) bit
described) 11-9
latching timer status bits 11-9
mapping programmable timer interrupts 5-45
Interface with the system bus 8-44
accessing slave processors over the cluster bus 8-44
arbitration for control of 8-44
basic system bus/cluster bus interface, diagram of 8-45
bidirectional system bus interface, diagram of 8-47
cluster bus 8-44
core accesses of 8-48
FLAGx 8-48
master processor accesses of 8-46
MSx 8-48
signal glitches on the HBR line 8-46
system access of slave processors 8-46
uniprocessor to microprocessor interface 8-51
Inter-IC sound bus protocol 9-61
Internal buses
access restrictions 5-27
and the external ADDRx data bus 5-12
control of 5-7
DM bus 5-7, 5-12
I/O bus 5-7, 5-12, 7-25
memory, connection to 5-7
PM bus 5-7, 5-12
Internal clock generator 12-26
enabling 12-27
multiprocessing and 12-26
phase lock 12-27
Internal interrupt vector table (IIVT) bit 5-30
Internal memory block data width, see IMDWx (internal memory block data width) bit
Internal memory map
IOP registers 5-23
normal word 5-24
short word 5-24
Internal memory space
address boundaries 5-19
address regions 5-23
concurrent DMA accesses of 6-74
defined 5-4
described 5-23
diagram of 5-23
DMA transfers and DMAGx 8-21
and DMARx 8-21
extending DMA access to 7-30
external port connection 6-27
handshake mode DMA accesses 7-31
host data transfers through the EPBx buffers 8-18
host DMA transfers 8-21
interrupt vector table, address of 5-24
low-level organization 5-35
map of 5-17
multiprocessor DMA transfers to 7-30
prioritizing external DMA accesses 6-37
reserved addresses 5-19
setting up host DMA transfers 8-21
slave mode DMA accesses 7-31
SPORT connection 6-27
unusable locations 5-24
Interprocessor communications overhead 7-6
see Interprocessor messages
Interprocessor messages 7-36, 7-37, 8-36
described 7-36
host vector interrupts, see Host vector interrupts
immediate high-priority interrupt 8-36
interrupt service routines 7-38, 7-39, 8-36
IOP registers 8-36
message passing, see Message passing
MSGRx registers 7-36, 8-36
types 8-36
vector interrupts 7-36, 7-38, 8-36
VIRPT register 7-36, 8-36
Interrupt controller 3-7
Interrupt latch register, see IRPTL register
Interrupt latency 3-40
branch and following cycle 3-43
branching to the vector cycles 3-40
first cycle in fetch/decode of first instruction in interrupt service routine 3-44
first two cycles of a program memory data access 3-43
interrupt priority and 3-43
IRQx and multiprocessor vector standard 3-43
last iteration of one-instruction loop 3-43
multicycle operations 3-43
pipelined delayed branch 3-42
pipelined program memory data access with cache miss 3-41
pipelined single-cycle instruction 3-40
processor access of external memory space during a host bus grant or while bus slave 3-44
recognition cycle 3-40
synchronization and latching cycle 3-40
third to last iteration of one-instruction loop 3-43
wait states for external memory space accesses 3-44
writes to IRPTL 3-40
Interrupt mask and latch registers, see IMASK register and IRPTL register
Interrupt mask pointer register, see IMASKP register
Interrupt masking and control 3-46
   IMASK register 3-46
      see also IMASK register
   IMASKP register 3-46
      see also IMASKP register
Interrupt priority 3-45
   arithmetic interrupts 3-45
      described 3-45
   INT_HIx bit and programmable timer interrupts 3-45
   nested interrupts and 3-45
   programmable timer interrupts 3-45
      ranking 3-45
      STKY flags and 3-45
Interrupt request lines, see IRQx
Interrupt service routine 7-39
   pushing ASTAT on the status stack 12-33, 12-34
   pushing IOSTAT on the status stack 12-33, 12-34
   reducing to normal subroutine 3-50
RTI instruction 3-39
   see also Vector interrupts
   servicing vector interrupts 7-38
   VIPD bit, checking 7-39
Interrupt servicing stages 3-40
   branching to the vector 3-40
   recognition 3-40
   synchronization and latching 3-40
Interrupt vector addresses
   described F-1
   external EPROM booting and location of interrupt vector table F-3
   external source booting and location of interrupt vector table F-3
   IIVT bit and selecting the location of the interrupt vector table F-3
   IMASK register bit values, diagram of F-5
   IRPTL and IMASK interrupt vectors and priorities F-1
   IRPTL register bit values, diagram of F-5
   no boot mode and location of interrupt vector table F-3
   offsets from base addresses F-1
Interrupt vector table 3-44
   address of 8-38
   IRPTL 3-44
      location for external EPROM booting F-3
      location for external source booting F-3
      location for no boot mode F-3
   VIRPT 3-44
Interrupt vector table address
   boot mode 5-30
   internal memory space 5-24
   locating 12-61
no boot mode 5-30
when IIVT=0 12-61
when IIVT=1 5-30, 12-61
Interrupt-driven data transfer mode 9-65
described 9-65
interrupts 9-65
Interrupting DMA transfers over the external bus 7-18
Interrupts
circular buffer overflow 4-12
clearing the current one for reuse 3-49
I²S interrupt-driven data transfer mode 9-65
IRPTL write timing 3-40
latency 3-40
loop address stack overflow 3-33
multiple SPORT single-word transfers 9-87
nesting and IMASKP 3-46
packed serial data word transfers 9-48
PC stack interrupt 3-24
processing 3-42
processing and delayed branches 3-23
processing in counter-based loops 3-29
program sequencer, see Program sequencer interrupts
program structures 3-1
programmable timer 3-45, 11-3, 11-6
serial port 9-6
see also SPORT interrupts
servicing restrictions 3-38
servicing sequence 3-39
servicing stages 3-40
software, see Software interrupts
SPORT packed-word transfers 9-87
SPORT single-word transfers 9-79, 9-86
stack overflow 3-24
timing and sensitivity of external 3-50
vector addresses F-1
vector table 3-44
INTIO (DMA single-word interrupt enable) bit 6-14
described 6-17
enabling interrupt-driven I/O 8-20
single-word EPBx data transfer control 7-29
IOCTL register 10-6
address of 11-14, E-68
bit definitions E-70
default bit values, diagram of 10-12, E-69
described E-68
DSDCK1 10-9, 10-15
DSDCTL 10-9, 10-15
FLAG1 12-30
FLAG1 control bits 12-30
FLAG1 direction 12-30
FLAG1 value after reset 12-31
INDEX

initialization value E-68
SDBN 10-11
SDBS 10-11
SDBUF 10-11, 10-17
SDCL 10-10
SDPGS 10-10, 10-18
SDPM 10-10
SDPSS 10-11
SDRAM configuration
  parameters, summary of 10-13
SDRAM control bit definitions
  10-9
SDRAM interface control 10-9
SDRAM power-up sequence and
  10-20
SDRDIV register and 10-14
SDSRF 10-10, 10-20
SDTRAS 10-10
SDTRP 10-10
IOP register reads
  described 7-27
  maximum throughput 7-27
IOP register writes
  ACK 7-26
  described 7-26
  maximum pipeline throughput
    7-27
  slave write FIFO 7-26
  throughput 7-26
  write latency 7-26
IOP registers
  access restrictions E-40
  address region 5-23
  addresses, reset values, and groups
  E-43
  addressing for host transfers 8-11
  bit wise operations and 12-30
  defined 5-4, 7-5, 8-6, E-1
  described E-31
  DM bus accesses E-41
  DMA registers, summary of E-33,
    E-35
  DMACx registers 6-11, 8-16,
    E-54
  DMASTATx register E-64
  external memory space wait states
    5-53
  external port bus accesses E-41
  group access contention E-41
  host booting and 12-58
  host data transfers and 8-16
  host interface and 8-6
  host reads of 8-17
  host writes to 8-16
  I/O bus accesses E-41
  initialization values after reset,
    summary of E-40
  internal DMA transfers to E-41
  internal memory address region
    5-23
  interprocessor messages and 8-36
  IOCTL 11-14, E-68
  IOSTAT 11-13, 11-14, E-75
  mode and control bit write
    latencies E-43
  MSGRx 8-16, 8-36
  multiprocessing and 7-25
  multiprocessing writes to 7-26
INDEX

PM bus accesses E-41
programmable timer registers,
addresses of 11-12
RDIVx E-78
resolving group access contention E-42
SDRDIV register 10-13
serial port registers, summary of E-35
SRCTLx E-81
STCTLx E-90
summary of E-31
SYSCON 8-16, E-99
SYSTAT 8-16, E-106
system control registers, summary of E-32
TCOUNTx 11-6
TDIVx E-78
TPERIODx 11-6
TPWIDTHx 11-6
VIRPT 8-36
WAIT E-111
write latencies E-42
IOSTAT register
address of 11-14, E-75
bit definitions E-76
default bit values, diagram of 11-13, E-76
described E-75
flag status updates 12-31
FLAG^{11,4} 12-30
FLAG^{11,4} inputs 12-31
FLAG^{11,4} outputs 12-33
FLAGx status bit permissions 12-32
FLAGxO status bits 12-32
initialization value E-75
programmable I/O ports and 11-13
signaling external devices with FLAGx bits 12-33
status stack pushes and pops 12-34
IRFS (RFS source) bit 9-21
defined 9-29
described 9-54
IRPTEN (global interrupt enable) bit
interrupt request validity and 3-38
IRPTL register
bit definitions E-14
BIT SET instruction and 3-49
bit values during interrupt service routine execution 3-44
clearing 3-44
clearing current interrupt for reuse 3-49
default bit values, diagram of E-13, F-5
described 3-44, E-12
disabling DMA interrupts and 7-29
DMA interrupts 6-9, 6-45
forced interrupt timing 3-40
IMASK register bits and 3-44, 3-46
initialization value E-12
interrupt priority 3-45
interrupt vector table and 3-44
interrupt vectors and priorities
F-1
programmable timer interrupts
and 11-9
RESET 3-44
reusing an interrupt the processor
is processing 3-44
RTI instruction and 3-16
size of 3-44
software interrupts, activating
3-49
updating 3-48

IRQ
external interrupt and timer pins
12-28
operation cycles 12-26
pin definition 12-17
program sequencer interrupts
3-38
standard latency 3-43
state after reset 12-24
task-on-demand control 12-28
timing and sensitivity 3-50
validity of edge-triggered 3-51
validity of level-sensitive 3-50

IRQxE (external interrupt mode)
bits, configuration values 3-51
ITFS (TFS source) bit 9-16
defined 9-29
described 9-54

J
JTAG boundary register D-6
described D-6
scan path positions
definitions D-6
latch type and function D-6
size of D-6

JTAG instruction register D-3
Bypass register D-4
described D-3
instruction binary code D-3
loading D-3
serial scan paths D-4
diagram of D-5
size of D-3
test instructions, summary of D-3

JTAG test access port
additional references D-29
BIST (built-in self-test
instructions) D-28
boundary register D-6
see JTAG boundary register
boundary scan D-1
described D-1
device identification register D-28
instruction register D-3
see JTAG instruction register
latches D-1
private instructions D-29
serial test access port D-1
serial-shift register path D-1
TAP (test access port) D-2
see TAP (JTAG test access port)
JTAG test clock, see TCK
JTAG test data input, see TDJ
JTAG test data output, see TDO
INDEX

JTAG test mode select, see TMS
JTAG test reset, see TRST
JTAG/emulator
  accessing on-chip emulation features 12-34
  boundary scans 12-34
  CLKin connection 12-40
  clock skew 12-40
  EMU 12-19
  executing synchronous multiprocessor operations 12-40
  EZ-ICE emulator, see EZ-ICE emulator
  interface pins 12-34
  pin definitions 12-19
  pin states after reset 12-25
  scan path, diagram of 12-40
  signal termination 12-39
  TCK 12-19
  TDI 12-19
  TDO 12-20
  test access port 12-34
  TMS 12-20
  TRST 12-20, 12-35
JUMP (CI) instruction
  clearing the current interrupt for reuse 3-49
  status stack restore of ASTAT 3-48
  status stack restore of MODE1 3-48
JUMP (LA)
  aborting noncounter-based loops prematurely 3-30
  automatic loop abort 3-17
  restriction in loops 3-17
Jump instructions 3-1
  automatic loop abort 3-17
  CI modifier 3-44
  conditional branching 3-16
  delayed and nondelayed 3-17
  described 3-16
  indirect, direct, and PC-relative 3-17
  program memory data accesses 3-11

K
KEYWDX register 9-9
  described 9-73
  memory-mapped address and reset value 9-10, 9-12
  multichannel receive comparison 9-73

L
L (DAG locations) registers 4-2
  circular data buffers and 4-11
  initialization and postmodify behavior 4-7
  values, restrictions on 4-11
L_FIRST (left/right channel transmit/receive first) bit 9-16, 9-21
  default setting 9-63
  defined 9-30
  described 9-63
INDEX

LADDR register
- described 3-33
- loop address stack pointer and 3-33
- value when loop address stack empty 3-33

LAFS (late TFS/RFS) bit 9-16, 9-22
- defined 9-29
- described 9-56
- late frame sync mode 9-56

Late frame sync mode 9-56
- described 9-56

Latencies and throughput
- summary of 12-65
- system registers effect and read latencies E-4

Latency between DMA request and DMA grant signals, handling 6-65

LCE condition 3-12, 3-14
- CURLCNTR (current loop count) and 3-12
- DO UNTIL instruction 3-12
- IF NOT LCE instruction and 3-13

LCNTR 3-25, 3-34
- CURLCNTR and 3-35
- described 3-35
- last loop iteration 3-35
- loop counter stack and 3-35
- nested loops, setting up count value for 3-35
- reads of 3-37

LE condition 3-13

Least significant word (LSW) format 5-29

Loading routine, see Booting

Local bus, host interface and 8-6

Loop abort (LA) modifier 3-34

Loop address stack 3-7
- described 3-32
- DO UNTIL instruction and 3-33
- empty state 3-33
- layout 3-32
- loop abort (LA) modifier and 3-34
- overflow 3-33
- PUSH LOOP instruction and 3-33
- pushing and popping 3-7
- stack pointer and the LADDR register 3-33
- STKY register and 3-33

Loop counter stack 3-34
- LCNTR value and 3-35
- pushing for nested loops, diagram of 3-36
- see LCNTR

Loop counters and stack 3-34
- current loop counter, see CURLCNTR

Loop counter stack 3-34
- loop counter, see LCNTR

Loop instructions 3-1
- counter-based loops 3-28
- DO FOREVER 3-12
- DO UNTIL 3-11
- see DO UNTIL instruction 3-25

instruction pipeline 3-27
INDEX

JUMP (LA) and automatic loop abort 3-17
loop address stack, see Loop address stack
loop counters and stack 3-34
noncounter-based loops 3-29
program memory data accesses
restrictions 3-27, 3-28, 3-29
see also General loop restrictions 3-27
short loops 3-27, 3-28
simple loop, example code 3-25
termination conditions 3-33
Loop stacks
empty flag 3-54
flags 3-54
overflow flag 3-54
Loop termination instructions 3-12
LRFS (active state RFS) bit 9-21
defined 9-30
LRU (least recently used) bit
described 3-59
values 3-59
LSEM bit 3-54
LSOV bit 3-54
LSWF packing format 6-52
LT condition 3-13
LTFS (active state TFS) bit 9-16
defined 9-30
described 9-55

M
M (DAG modify) registers 4-2
circular buffer addressing and 4-9
circular data buffers and 4-11
postmodify addressing operations 4-7
premodify addressing operations 4-6
MASTER (DMA master mode enable) bit 6-14, 6-30, 8-21, 8-22
described 6-19
DMA memory transfers 7-31, 7-32
DMA transfers to on-chip memory 7-31
Master mode DMA 6-21, 6-30, 6-55
described 6-55, 6-58
initiating transfers 6-55
operation examples 6-58
placing a channel in 6-58
Master processor
accesses and operation cycles 12-26
accesses of the system bus 8-46
data transfers with the slave processor 7-25
defined 7-5, 8-6
external bus arbitration 7-16
host interface and 8-6
MCE (multichannel mode enable) bit 9-22
defined 9-31
described 9-70
effect latency 9-70
INDEX

I$_2$S SPORT mode, enabling 9-62
OPMODE and 9-70
standard SPORT mode, enabling 9-59

Memory
32- and 40-bit data, configuration for 5-40
32- and 48-bit words, using 5-30
access restrictions 5-27
access timing of multiprocessor
memory space 5-67
ACK 5-47
address boundaries 5-19
address decoding table 5-20
ADDRx pin 5-44
architecture, diagram of 5-2
bandwidth 5-1
boot modes 5-53
bus idle cycle, see Bus idle cycle
bus master accesses of external
memory space 5-66, 5-67
cache miss 5-10
core accesses
internal memory space through
multiprocessor memory space 5-25
over the PM bus 5-10
DAG operation, see DAG operation
data transfers 5-7
48-bit accesses of program
memory 5-14
address sources 5-11
between memory and registers 5-12
between universal registers 5-12
example code for 48-bit pro-
gram memory access 5-14
over DM bus 5-11
over PM bus 5-11
PX register transfers, diagram of 5-13
single-cycle, number of 5-17
with the Register File 5-11
DATAx 5-45
DM bus, see DM bus
dual data accesses, see Dual data accesses
EPROM boot mode 5-53
see EPROM boot mode
executing program from external
memory space 5-49
extending off-chip memory
accesses 5-53
external memory address space 5-44
external memory banks and
SDRAM, see SDRAM interface
external memory banks, see External memory banks
external memory space access
address fields 5-26
external memory space access
timing 5-65
external memory space, see External memory space
external port access, see External port
external SDRAM memory, see
SDRAM interface
features 5-1
fine tuning accesses 5-35
generating memory addresses
5-11
I/O bus, see I/O bus
indirect addressing 5-11
Instruction cache, see Instruction cache
instruction fetches, see Instruction fetches
interface signals for external memory space 5-44
interface with off-chip devices 5-43
internal bus connections, see Internal buses
internal bus control, see Internal buses
internal memory space, see Internal memory space
interrupt vector table address 5-30
invalid multiprocessor memory space addresses 5-25
IOP registers, see IOP registers
low-level physical mapping 5-35
memory blocks, see Memory blocks
modified Harvard architecture 5-8
MSx 5-45
multiprocessor memory space access address fields 5-25
multiprocessor memory space, see
Multiprocessor memory space normal vs. short word addressing 5-29
off-chip devices and the external port 5-43
off-chip interface pins 5-43
ordering of 16-bit short words within 32- and 48-bit words 5-32
organization 5-16
organization vs. address, diagram of 5-32
packing external memory program data 5-49
PM and DM bus address bits, diagram of 5-8
PM and DM bus addresses 5-8
PM bus accesses of external memory space 5-26
PM bus, see PM bus
PM data accesses through the instruction cache 5-10
Program sequencer, see Program sequencer
PX registers, see PX registers
RD 5-45
Register File, see Register File
reserved addresses 5-19
same block, same cycle access conflicts 5-15
SDRAM, see SDRAM interface
shadow write FIFO 5-39
short word accesses 5-41
short word addresses, diagram of
INDEX

Memory block configuration
changing word width 5-40
for 16-bit short words 5-31
for 32-bit data words 5-31
for 48-bit instruction words 5-30
IMDWx bit and 5-40
starting address of 32-bit data
  5-35
word width 5-30
Memory block organization 5-16
block 0 5-16
block 1 5-16
columns 5-32
diagram of 5-16
low-level 5-35
physical mapping 5-35
Memory block storage capacity
48-bit words 5-31
for 16-bit words 5-31
for 32-bit words 5-31
Memory blocks
32- and 40-bit data, configuration
  for 5-40
32- and 48-bit words, configuring
  5-30
block 0 address ranges for
  instructions and data, example
  of 5-34
block 1 invalid addresses 5-29
defined 5-4
described 5-16
invalid addresses 5-39
normal word addresses 5-29
ordering of 16-bit short words
  within 32- and 48-bit words
  5-32
reads and writes of the same block
  3-10
short word addresses, see Short
  word addressing
single-cycle transfers, number of
  5-17
storage capacity 5-17
storing mixed words in the same
  block, see Mixed word storage
word size and 5-28
word types 5-28
memory blocks
  vs. memory banks 5-49
Memory map
  external memory space 5-26
  internal memory space 5-17
  multiprocessor memory space
  5-24
Memory organization 5-16
Memory read strobe, see RD
Memory select lines, see MSx
Memory write strobe, see WR
Message passing 7-36, 8-37
described 7-37
FLAGx pins and 7-37
host software protocols 8-37
host vector interrupts 8-38
interprocessor communication
  8-36
MSGRx registers 8-37
register handshake protocol 7-37,
  8-37
register write-back protocol 7-38, 8-38
software protocols for 7-37
vector interrupt-driven protocol 7-37, 8-37
MFD (multichannel frame delay) bits 9-16
defined 9-31
described 9-71
MI (multiplier floating-point invalid operation) bit 2-34
floating-point multiplication and 2-36
MIS (multiplier floating-point invalid operation) bit 2-34
Miscellaneous instructions
Cjump/Rframe (type 24) instructions A-10
NOP (type 21) instructions A-9
push|pop stacks/flush cache (type 20) instructions A-9
summary A-9
Mixed word storage
diagram of 5-38
invalid addresses 5-39
rules for 5-32
same block 5-32
same block restrictions 5-36
same block, example of 5-33
starting addresses for contiguous 32-bit data 5-37
Mixed words
example, diagram of 5-33
fine tuning accesses 5-35
MMSWS (multiprocessor memory space wait state) bit
automatic wait state option 5-62
MN (multiplier result negative) bit 2-34
described 2-35
MN condition 3-13
MOD1 multiplier operations
options described B-52
summary of B-53
MOD2 multiplier operations
options described B-51
summary of B-52
Mode register set command
(SDRAM), see MRS command
MODE1 register
alternate register file register control bits 2-11
alternate register file registers, activating 2-11
ALU operation bits 2-14
ALUSAT 2-14
bit definitions E-18
bit-reverse mode control bits 4-14
BM (bus master condition) 3-13
conditional instructions and 3-12
DAG register control bits,
summary of 4-5
default bit values, diagram of E-17
described E-16
effect latency of activation of alternate register file register sets
INDEX

2-11
floating-point operating mode
status bits, summary of 2-32
floating-point operation status
bits 2-32
initialization value E-16
IRPTEN 3-38
nested interrupts 3-46
NESTM 3-46
preserved current values of 3-49
program sequencing interrupts
and 3-38
RND32 2-14, 2-32, 5-41
RTI instruction and 3-16
sign extending short word
addresses 5-30
sign extension enable (SSE) bit
5-30, 5-42
SRCU 2-29
SRD1H 4-5
SRD1L 4-5
SRD2H 4-5
SRD2L 4-5
SRRFH 2-11
SRRFL 2-11
status stack save and restore
operations 3-48
TRUNC 2-14, 2-32
zero-filling short word addresses
5-30
MODE2 register
bit definitions E-23
BUSLK 7-34
CADIS and CAFRZ bit
definitions 3-62
default bit values, diagram of E-22
described E-21
diagram of 11-10
FLAG[3:0] control bits 12-29
initialization value E-21
instruction cache
disabling/freezing 3-62
instruction cache mode bits, value
after reset 3-62
INT_HIx 3-45, 11-9
interrupt mode bits 3-51
interrupt sensitivity configuration
3-51
IRQxE 3-51
mapping programmable timer
interrupts 3-45
PERIOD_CNTx 11-6, 11-8
programmable I/O ports and
11-13
programmable timer enable 11-1
PULSE_HIx 11-6, 11-8
PWMOUTx 11-3, 11-5, 11-8
TIMENx 11-1, 11-8
Modified Harvard architecture 5-8
MOS (multiplier fixed-point
overflow) bit 2-34
described 2-35
MR register values and 2-35
Most significant word (MSW)
format 5-29
MR=Rn/Rn=MR operation
compute field B-60
described B-60
multiplier status flags B-61
MRB=0 operation
   described B-59
   multiplier status flags B-59
MRB=MRB+Rx*Ry mod2 operation
   described B-55
   multiplier status flags B-55
MRB=MRB-Rx*Ry mod2 operation
   described B-56
   multiplier status flags B-56
MRB=RND MRB mod1 operation
   described B-58
   multiplier status flags B-58
MRB=Rx*Fy mod2 operation
   described B-54
   multiplier status flags B-54
MRB=SAT MRB mod1 operation
   described B-57
   multiplier status flags B-57
MRCCSx register 9-9, 9-72
   defined 9-72
   memory-mapped address and reset value 9-10, 9-12
   multichannel companding formats 9-45
MRCSx register 9-9, 9-72
   defined 9-72
   memory-mapped address and reset value 9-10, 9-11
MRF=0 operation
   described B-59
   multiplier status flags B-59
MRF=MRF+Rx*Ry mod2 operation
   described B-55
   multiplier status flags B-55
MRF=MRF-Rx*Ry mod2 operation
   described B-56
   multiplier status flags B-56
MRF=RND MRF mod1 operation
   described B-58
   multiplier status flags B-58
MRF=Rx*Ry mod2 operation
   described B-54
   multiplier status flags B-54
MRF=SAT MRF mod1 operation
   described B-57
   multiplier status flags B-57
MRS command 10-31
MSGRx registers 8-36
   host data transfers and 8-16
   host interface and 7-36
   host software protocols and 8-37
   interprocessor messages 7-36, 8-36
   message passing 7-36, 8-37
   multiprocessing data transfers 7-25
   shared-bus multiprocessing 8-36
MSTR (SPORT transmit and receive master mode) bit 9-16, 9-21
   defined 9-31
   described 9-64
MSWF packing format 6-14, 6-52
and 48-bit DMA words 6-53
described 6-17

MSx
and core accesses of the system bus 8-48
chip selects for peripheral devices 5-49
conditional memory write instructions and 5-49
decoded memory address lines 5-49
external DMA data transfers 6-63
external memory bank addresses and 5-48
external memory space interface and 5-45
pin definition 12-5
state after reset 12-22

MTCCSx register 9-9, 9-72
defined 9-72
memory-mapped address and reset value 9-10, 9-11
multichannel companding formats 9-45
receive comparison disabled and 9-74

MTCSx register 9-9, 9-72
defined 9-72
memory-mapped address and reset value 9-10, 9-11
MU (multiplier underflow) bit 2-34
described 2-36
Multichannel frame delay described 9-71

MFD values in multiprocessor system 9-71
T1 devices, interface with 9-71
Multichannel frame syncs 9-69
described 9-69
Multichannel receive comparison feature 9-74
Multichannel receive comparison mask registers
IMASK register 9-73
Multichannel receive comparison registers
described 9-73
example application 9-75
IMAT and 9-74
IMODE and 9-74
KEYWDx register 9-73
operation 9-73
SDEN and 9-74
SRCTLx register control bits for receive comparisons 9-74
Multichannel SPORT mode channel selection registers 9-72
see Channel selection registers
channel slot capabilities 9-67
channel slot synchronization 9-69
channel slots 9-67
CHNL (current channel selected) 9-26
CKRE (frame sync clock edge) 9-26
companding 9-67
companding formats 9-45
control bits 9-69
INDEX

see Multichannel SPORT mode control bits
current channel selected status 9-71
data justification 9-45
data word formats 9-44
data word selection 9-72
default bit values, diagram of 9-20, 9-25
described 9-67
DITFS 9-26
DMA operation and 9-69
DTYPE 9-27
early frame sync mode and 9-56
enable effect latency 9-70
enabling 9-70
frame sync data dependency in 9-57
frame sync logic level, configuring 9-55
frame sync source 9-69
frame syncs 9-69
see Multichannel frame syncs
IMAT 9-28
IMODE 9-29
late frame sync mode and 9-56
linear transfers 9-45
LRFS 9-30
LTFS 9-30
MCE 9-31
MFD 9-31
multichannel compand select registers 9-45
multichannel frame delay 9-71
multichannel operation, diagram of 9-68
NCH 9-32
number of channel slots, setting 9-71
OPMODE 9-32, 9-36
PACK 9-32
primary and secondary channels, configuration of 9-70
receive comparison registers 9-73
see Multichannel receive comparison registers
receive control bits 9-21
RFSx pin connections 9-69
ROVF 9-33
RXS 9-33
SCHEN 9-34
SDEN 9-34
SENDN 9-35
SLEN 9-35
TCLK 9-28
TCLKx and RCLKx pin connections in 9-67
TFSx pin connections 9-69
timing reference 9-69
transfer timing characteristics, example of 9-68
transmit control bits 9-15
transmit data valid signal 9-69
TUVF 9-36
TXS 9-37
TXx_z data buffer 9-69
Multichannel SPORT mode control bits 9-69
INDEX

CHNL 9-71
MCE 9-70
MFD 9-71
NCH 9-71
operation mode 9-70
summary of 9-69
Multifunction operations 2-50
described 2-50, B-94
dual add and subtract
instructions, summary of 2-50
dual add/subtract (fixed-point) B-96
dual add/subtract (floating-point) B-98
fixed-point multiply and
accumulate instructions, summary of 2-51
floating-point multiply and ALU
instructions, summary of 2-51
input operand constraints B-94
input operand locations,
restrictions 2-50
input registers, diagram of 2-52
multiplication and dual add and
subtract instructions, summary of 2-51
parallel multiplier and ALU
(fixed-point) B-100
parallel multiplier and ALU
(floating-point) B-101
parallel multiplier and dual
add/subtract B-104
Register File and B-94
single-operation functions vs.
2-50
types B-94
valid input registers, diagram of B-95
Multiplication and dual add and
subtract instructions, summary of 2-51
Multiplier fixed-point results 2-28
fractions 2-28
MR registers 2-28
see Multiplier MR registers
overflow status flags 2-35
placement, diagram of 2-28
Register File transfers 2-28
underflow status flags 2-37
Multiplier floating-point operating
modes 2-32
described 2-32
fixed-point rounding restriction 2-32
MODE1 status bits 2-32
RND32 (floating-point rounding boundary) 2-32
rounding boundary 2-33
rounding mode 2-33
TRUNC (floating-point rounding) 2-32
Multiplier instruction set summary 2-38
Multiplier MR register operations
valid maximum saturation values 2-31
Multiplier MR registers 2-28
activation of 2-29
INDEX

architecture 2-28
context switching 2-29
data alignment 2-29
data transfers and 2-29
described 2-28
fixed-point accumulation
instructions and 2-29
fixed-point integer and fraction results 2-36
MR transfer formats, diagram of 2-29
overflow status flags for
fixed-point results 2-35
parallel accumulators, use as 2-29
Register File transfers 2-30
Multiplier operations 2-27, B-50
denormal operands 2-36
described 2-27, B-50
fixed-point 2-27
fixed-point operand format 2-27
fixed-point results 2-28
see Multiplier fixed-point results
floating-point 2-27
floating-point operating modes 2-32
see Multiplier floating-point operating modes
Fn=Fx*Fy B-62
input/output rate 2-27
MOD1 options
described B-52
summary of B-53
MOD2 options
described B-51

MR registers and fixed-point results 2-28
MR=Rn/Rn=MR B-60
MRB=0 B-59
MRB=MRB+Rx*Ry mod2 B-55
MRB=MRB-Rx*Ry mod2 B-56
MRB=RND MRB mod1 B-58
MRB=Rx*Fy mod2 B-54
MRB=SAT MRB mod1 B-57
MRF=0 B-59
MRF=MRF+Rx*Ry mod2 B-55
MRF=MRF-Rx*Ry mod2 B-56
MRF=RND MRF mod1 B-58
MRF=Rx*Fy mod2 B-54
MRF=SAT MRF mod1 B-57
Register File 2-27
Rn=MRB+Rx*Ry mod2 B-55
Rn=MRB-Rx*Ry mod2 B-56
Rn=MRF+Rx*Ry mod2 B-55
Rn=MRF-Rx*Ry mod2 B-56
Rn=RND MRB mod1 B-58
Rn=RND MRF mod1 B-58
Rn=Rx*Fy mod2 B-54
Rn=SAT MRB mod1 B-57
Rn=SAT MRF mod1 B-57
status flag update 2-34
status of most recent 2-34
summary of B-50
Multiplier registers
summary of A-17
Multiplier status flags 2-34
ASTAT status bits, summary of 2-34

I-76 ADSP-21065L SHARC DSP User’s Manual
described 2-34
fixed-point underflow results 2-37
floating-point invalid operation 2-36
MI floating-point invalid operation 2-34
MIS floating-point invalid operation 2-34
MN result negative 2-34
MOS fixed-point overflow 2-34
MR register values and 2-35
MU underflow 2-34
MUS underflow 2-34
MV overflow 2-34
MVS floating-point overflow 2-34
negative flag 2-35
overflow flags 2-35
STKY status bits, summary of 2-34
underflow flags 2-36
updating 2-34
Multiplier unit 2-26
described 2-1, 2-26
fixed-point instructions 2-26
floating-point instructions 2-26
instruction set summary 2-38
instruction types, summary of 2-26
multifunction computations and operations 2-26
operations, see Multiplier operations
status flags, see Multiplier status flags
Multiprocessing 7-1
ACK 12-52
basic system, diagram of 7-2
BM condition and 3-13
BMS and 12-51
booting, see Multiprocessor booting
broadcast writes
see Broadcast writes
BRx pins 7-3
bus arbitration, see Multiprocessor bus arbitration
bus lock and semaphores, see Bus lock and semaphores
bus master 7-1
clock skew 12-43
configurations for interprocessor DMA, summary of 6-70
data transfers, see Multiprocessing data transfers
DMACx registers 7-4
emulating synchronous operations with CLKIN 12-40
EPBx buffers 7-4
EPROM boot mode and 12-51
external bus 7-1, 7-4
features 7-1
host accesses of both processors 8-14
host interface 8-6
host interface with the system bus 8-44
IDx pin connections 7-3
INDEX

immediate high-priority interrupt 8-36
internal clock generation and 12-26
interprocessor messages 7-36, 8-36
interrupt service routine 8-36
IOP registers 7-4, 7-5
master processor 7-5
multichannel SPORT mode and 9-71
multiprocessor memory space 7-4, 7-5
multiprocessor system 7-5
operation cycles 12-26
pin connections between two processors 7-3
SDRAM accesses and bus arbitration 7-17
SDRAM operation 10-25
shared-bus 8-36
sharing a common boot EPROM 12-51
sharing the DMAGx signal 6-68
single-word data transfers 7-5
slave processor 7-5
SYSTAT register status bits 7-40
see also SYSTAT register
system architecture, see Multiprocessing system architecture
system clock rate 7-5
system configuration for interprocessor DMA 6-70

Multiprocessing bus requests 7-10, 12-14
Multiprocessing data transfers 7-25
ACK 7-26
addressing 7-25
communication with slave processor’s core 7-25
data 7-25
DMA operations 7-25
DMA transfers, see Multiprocessing DMA transfers
DMACx registers 7-25
EPBx buffer writes, see EPBx buffers
EPBx transfers, see Multiprocessing EPBx transfers
external port 7-25
internal I/O bus 7-25
IOP register reads, see IOP register reads
IOP register writes, see IOP register writes
IOP registers 7-25
MSGRx registers 7-25
multiprocessor memory space accesses and wait states 7-25
shadow write FIFO 7-32
slave processor configuration 7-25
slave write FIFO 7-26
SYSCON register 7-25
SYSTAT register 7-25
types 7-25
vector interrupts and 7-25
VIRPT register 7-25
Multiprocessing DMA transfers
7-30
described 7-30
DMA packing 7-31
DMAGx 7-30
DMARx 7-30
extending internal memory space
access 7-30
external handshake mode DMA
configuration 7-32
external port DMA channels and
7-30
handshake mode DMA
configuration 7-31
slave mode DMA configuration
7-31
to on-chip memory 7-30, 7-31
types 7-30
Multiprocessing EPBx transfers
7-27
core hang 7-29
DEN (DMA enable) bit 7-29
DMA block transfers 7-27
external port buffers 7-27
FLSH (DMA flush buffers and
status) bit 7-29
flushing the EPBx buffers 7-29
interrupts 7-29
single-word transfers 7-27, 7-28
single-word, non-DMA transfers
7-29
types 7-27
writing to a full buffer 7-28
Multiprocessing ID, see IDx
Multiprocessing system architecture
cluster multiprocessing, see
Cluster multiprocessing
data bandwidth bottlenecks 7-6
data flow multiprocessing, see
Data flow multiprocessing
interprocessor communication
overhead 7-6
nodes 7-6
shared global memory 7-6
Multiprocessor booting 12-58
BEL 12-59
BMS 12-59
CS 12-59
EPROM boot sequence 12-59
from one EPROM, diagram of
12-60
HBR 12-59
host boot pin configuration 12-59
host boot sequence 12-59
Multiprocessor bus arbitration 7-10
acquiring the bus 7-12
BRx 7-10
BTC 7-12
bus request and read/write timing,
diagram of 7-15
bus synchronization operation
7-11
core priority access, see Core
priority access
CPA 7-11
described 7-10
DMA transfers and 7-17
HBRG 7-10
INDEX

HBR 7-10
IDx 7-10
pin definitions 7-10
protocol 7-12
SDRAM and 7-17
timing diagram 7-13
Multiprocessor memory space 7-4
access address fields 5-25
access timing 5-67, 5-68
address boundaries 5-19
address range of IDx processor 5-24
automatic wait state option 5-62
core accesses of internal memory space through 5-25
defined 5-5, 7-5
described 5-24
diagram of 5-24
host interface and 8-6
invalid addresses 5-25
map of 5-24
multiprocessing data transfers 7-25
single wait state (MMSWS) 5-57
wait states and acknowledge 5-61
Multiprocessor system 7-1
BRx pins 7-3
bus arbitration, see Multiprocessor bus arbitration
data transfers, see Multiprocessing data transfers
defined 7-5
determining the current bus master 7-11
diagram of 7-2
IDx pin connections 7-3
pin connections between two processors 7-3
processor self-configuration 7-11
Multiprocessor vector interrupts 3-52
described 3-52
minimum latency 3-52
VIPD bit 3-52
VIRPT 3-52
MUS (multiplier underflow) bit 2-34
described 2-36
MV (multiplier overflow) bit 2-34
described 2-35
MR register values and 2-35
MV condition 3-13
MVS (multiplier floating-point overflow) bit 2-34

N
NC, pin definition 12-20
NCH (number of channel slots) bit 9-22
defined 9-32
described 9-71
NE condition 3-14
Nested interrupt routines 3-7
Nested interrupts enabling and disabling 3-47
IMASKP register and 3-46
IMASKP register and temporary interrupt masks 3-47
interrupt priority and latency 3-43
MODE1 register and 3-46
NESTM bit 3-47
RTI instruction and 3-47
Nested loops 3-7
    noncounterbased loops and 3-30
    setting up a count value for 3-35
NESTM (nesting mode) bit
described 3-46
No boot mode
    address of initial instruction fetch
        12-60
described 12-60
    external memory address of first
        instruction 12-49
    interrupt vector table, address of
        5-30
    pin configuration 12-51
Noncounter-based loops
    aborting executing prematurely
        3-30
described 3-29
    instruction pipeline and 3-30
    nested loops and 3-30
    pipelined two-instruction
        one-iteration (2 cycles of
        overhead) 3-32
    pipelined two-instruction
        two-iteration 3-31
    restrictions 3-29
    termination condition 3-30, 3-33
Nondelayed branches 3-18
    call decode address 3-18
    call return address 3-18
    DB modifier and 3-18
defined 3-18
    pipelined stages of jumps/calls
        3-18
    pipelined stages of returns 3-19
Nonsequential program operations
    3-5
NOP (type 21) instruction
described A-77
    opcode A-77
    syntax summary A-9
Normal SBTS operation (HBR
deasserted) 5-63
Normal word addresses
    block 0 invalid addresses 5-29
    block 0, range of 5-29
    block 1 invalid addresses 5-29
    block 1, range of 5-29
    internal memory address region
        5-24
    interrupt vector table, address of
        5-24
    range of 5-29
    vs. short word addresses 5-29
    word width of 5-28
Normalized numbers C-2
    fields C-2
    hidden bit C-2
    unsigned exponent value range
        C-2
NOT AC condition 3-14
NOT AV condition 3-14
NOT BM condition 3-15
NOT FLAG0_IN condition 3-14
INDEX

NOT FLAG1_IN condition 3-14
NOT FLAG2_IN condition 3-14
NOT FLAG3_IN condition 3-14
NOT ICE condition 3-14
NOT LCE condition 3-12
NOT MS condition 3-14
NOT MV condition 3-14
NOT SV condition 3-14
NOT SZ condition 3-14
NOT TF condition 3-15
Notation conventions for Chapter 6, DMA 6-6
Number of external DMA bus transfers, specifying the 6-30
Numeric formats
described C-1
extended-precision, floating-point C-4
see Extended-precision, floating-point format
fixed-point C-8
see Fixed-point formats
short word, floating-point C-5
see Short word, floating-point format
single-precision, floating-point C-2
see Single-precision, floating-point format

Off-chip memory access extension method
either (ACK or WAIT register) method 5-54
external (ACK and WAIT register) method 5-54
external (ACK) method 5-53
internal (WAIT register) method 5-54
Opcode notation summary A-19
OPMODE (SPORT operation mode) bit 9-16, 9-21
defined 9-32
I²S SPORT mode, enabling 9-62
multichannel SPORT mode 9-70
standard mode, enabling 9-59
Oscilloscope probes 12-47
ground clip type 12-47
loading 12-47
recommended 12-47
standard ground clips 12-47
Overriding BMS 12-55

P
Paced master mode DMA 6-21
described 6-58
extending accesses 6-59
PACK (packing) bit 9-16, 9-21
defined 9-32
packing and unpacking serial data 9-47
SPORT DMA block transfers and 9-78

Off-chip memory access extension
method
either (ACK or WAIT register) method 5-54
external (ACK and WAIT register) method 5-54
external (ACK) method 5-53
internal (WAIT register) method 5-54
Opcode notation summary A-19
OPMODE (SPORT operation mode) bit 9-16, 9-21
defined 9-32
I²S SPORT mode, enabling 9-62
multichannel SPORT mode 9-70
standard mode, enabling 9-59
Oscilloscope probes 12-47
ground clip type 12-47
loading 12-47
recommended 12-47
standard ground clips 12-47
Overriding BMS 12-55

P
Paced master mode DMA 6-21
described 6-58
extending accesses 6-59
PACK (packing) bit 9-16, 9-21
defined 9-32
packing and unpacking serial data 9-47
SPORT DMA block transfers and 9-78
Packing sequence for downloading instructions from a 16-bit bus 6-53

Parallel multiplier and ALU
syntax and opcodes, summary of B-102

Parallel multiplier and ALU
(fixed-point)
compute field B-100
described B-100

Parallel multiplier and ALU
(floating-point)
compute field B-101
described B-101
Valid sources of input operands, summary of B-101

Parallel multiplier and dual add/subtract operations
compute field B-104
described B-104
Valid sources of input operands, summary of B-105

PC stack 3-6
almost full state 3-24
described 3-24
DO UNTIL instruction and 3-25
empty status 3-24
events that pop 3-24
flags 3-54
full state 3-24
full status 3-24
interrupt generation 3-24
Interrupt service routine push of 3-24
overflow status 3-24
reading and delayed branches 3-24
size of 3-24
stack full interrupt 3-24
STKY register and 3-24
PC stack empty flag 3-54
PC stack full flag 3-54
PC stack pointer, see PCSTKP
PCEM (PC stack empty) bit 3-54
PCFL (PC stack full) bit 3-54
PCI (program controlled interrupts) bit 6-40
CP (chain pointer) register and 6-40
described 6-40
disabling DMA interrupts 6-40
enabling and disabling DMA interrupts 6-46
restrictions 6-40
PCSTKP
data values 3-24
described 3-24
empty value 3-24
overflow value 3-24
pushing and popping 3-7
reading and delayed branches 3-24
write latency 3-24
PERIOD_CNTx (timer period count enable) bit 11-6
described 11-8
Pin definitions 12-3
ACK 12-7
INDEX

ADDRx 12-4
asynchronous inputs 12-3
BMS 12-13
BMSTR 12-13
BRx 7-10, 12-14
BSEL 12-14
CAS 12-10
CLKIN 12-14
CPA 7-11, 12-16
CS 12-8
DATAx 12-4
DMAGx 12-5
DMARx 12-5
DQM 12-10
DRx_X 12-11
DTx_X 12-11
EMU 12-19
external port 12-4
FLAGx 12-16
GND 12-20
HBR 12-9
host interface 12-7
IDx 7-10, 12-16
IRQx 12-17
JTAG/emulator 12-19
miscellaneous 12-20
MSx 12-5
multiprocessor bus arbitration
  7-10
NC 12-20
PWM_EVENTx 12-17
RAS 12-10
RCLKx 12-12
RD 12-17
REDY 12-9
RESET 12-18
RFSx 12-12
SBTS 12-6
SDA10 12-10
SDCKE 12-11
SDCLKx 12-10
SDRAM interface 12-10
SDWE 12-11
serial port 12-11
SW 12-6
synchronous inputs 12-3
system control 12-13
TCK 12-19
TCLKx 12-12
TDI 12-19
TDO 12-20
TFSx 12-12
TMS 12-20
TRST 12-20
unused inputs 12-3
VDD 12-20
WR 12-18
XTAL 12-19
Pin operation 12-26
asynchronous inputs 12-27
CLKIN frequencies, see CLkin frequencies
external interrupt and timer pins 12-28
EZ-ICE emulator, see EZ-ICE emulator
Flag inputs, see FLAGx 12-31
Flag outputs, see `FLAGx` 12-33
FLAGx 12-28
input synchronization delay 12-27
internal clock and phase lock 12-27
internal clock generation 12-26
JTAG interface pins, see `JTAG/emulator`
signal recognition phase 12-27
single-bit signaling 12-28
synchronization delay 12-27
XTAL and CLKin 12-26
Pin states after reset 12-22
ACK 12-22
ADDRx 12-22
BMS 12-23
BMSTR 12-22
BRx 12-22
BSEL 12-23
bus master driven pins 12-22
CAS 12-22
CLKIN 12-23
CPA 12-23
CS 12-23
DATAx 12-23
DMAGx 12-22
DMARx 12-23
DQM 12-22
DRx_X 12-24
DTx_X 12-24
EMU 12-25
FLAGx 12-24
HBG 12-22
HBR 12-24
IDx 12-24
IRQx 12-24
JTAG/emulator 12-25
MSx 12-22
PWM_EVENTx 12-24
RAS 12-23
RCLKx 12-24
RD 12-23
REDY 12-24
RESET 12-24
RFSx 12-24
SBTS 12-24
SDA10 12-23
SDCKE 12-23
SDCLKx 12-23
SDWE 12-23
serial port pins 12-24
SW 12-23
TCK 12-25
TCLKx 12-24
TDI 12-25
TDO 12-25
TFSx 12-24
TMS 12-25
TRST 12-25
WR 12-23
XTAL 12-24
Pipelining 3-19
described 3-4
execution cycles 3-5
system register writes and 3-8
Placing all SDRAM signals in a high impedance state 10-9
Placing the SDCLK1 signal only in a high impedance state 10-9
PM bus
   address bits, diagram of 5-8 and EPBx buffers 8-18
   connection to memory 5-7
   core memory accesses 5-10
   data storage 5-8
   data transfer destinations 5-11
   data transfer types 5-11
   data transfers with memory 5-7
   defined 5-5
dual data access conflicts 5-10
   generating 24-bit addresses 5-26
   generating addresses for 5-11
   instruction fetches 5-10
   memory accesses 5-27
   program segment address restriction 5-52
   PX register accesses 5-28
   Register File transfers 5-11
   transferring data to the DM bus 5-12
   PMODE (DMA packing mode enable) bit 6-14, 8-22, 8-24, 8-28
   and HBW bit combinations 6-52
   described 6-16
   EPBx packing mode bit values 6-17, 6-51
   EPBx packing modes 6-16
   external port DMA packing mode 6-51
   host EPBx packing modes 8-19
   host EPBx transfers 8-22, 8-24
   multiprocessing DMA transfers 7-31
   packing individual data words 8-19
   packing modes for EPBx buffers 6-17, 6-51
   values for EPBx buffer packing modes 6-17, 6-51
   PMWOUT 3-53
   Polling to determine the status of a DMA transfer 6-26
   Postmodify addressing operations 4-6
   compared to premodify addressing, diagram of 4-7
   immediate modifier value 4-6
   index (I) register value 4-6
   modify (M) register value 4-6
   uninitialized locations (L) registers and 4-7
   without circular data buffers 4-7
   Power and ground
   GND 12-20
   NC 12-20
   pin definitions 12-20
   VDD 12-20
   Power plane, decoupling capacitors and 12-46
   Power supply return, see GND
   Power supply, see VDD
   Powering up SDRAM after reset 10-28
   Power-up procedures
INDEX

TRST 12-35
Pre command 10-7, 10-32
Precharge command (SDRAM), see Pre command
Premodify addressing operations 4-6
compared to postmodify addressing, diagram of 4-7
immediate modifier value 4-6
index (I) registers and 4-6
locations (L) registers and 4-6
M (DAG modify) registers 4-6
modulo logic and 4-6
offset modifier 4-6
restrictions on using 4-6
Preprocessing 16-bit short word addresses, diagram of 5-36
Processor
defined 8-6
host control of 8-8
Processor architecture 1-9
booting 1-20
comprehensive instruction set 1-15
computation units 1-10
context switching 1-15
data address generators 1-11
DMA controller 1-19
DSP core 1-9
DSP core buses 1-13
dual-ported memory 1-16
external port interface 1-17
general-purpose I/O ports 1-14
host interface 1-17
I/O processor 1-18
Instruction cache 1-18
interrupts 1-15
Program sequencer 1-11
programmable timers 1-14
Register File 1-11
serial ports 1-18
summary of features 1-9
Processor benefits 1-5
Processor features 1-1, 1-5
40-bit extended precision 1-6
additional Literature 1-24
arithmetic 1-5
balanced performance 1-24
data flow 1-5
development tools 1-20
dual address generators 1-6
processor layout, diagram of 1-3
program sequencing, diagram of 1-3
summary of 1-22
super Harvard architecture, diagram of 1-2
Processor reset, see RESET
Processor synchronization, described 7-21
Processor system-level enhancements 1-6
high-level languages 1-7
IEEE formats 1-7
serial scan and emulation 1-7
Program controlled DMA interrupts 6-40
Program counter address after reset 12-53, 12-56
INDEX

Program counter stack pointer, see PCSTKP
Program counter stack, see PC stack
Program counter, see PC stack
Program execution
  40-bit data accesses 5-52
  address generation scheme 5-51
  aligning internal addresses with external memory space 5-50
  data access addressing 5-52
  data packing 5-49
  described 5-49
  example addresses for 5-50
  generating instruction addresses in external memory space 5-50
  invalid data segment addresses 5-52
  invalid program segment addresses 5-52
  mapping 64K memory space to 128K memory space 5-51
  multiple program segments, using 5-51
  PM bus address restriction 5-52
  program segment alignment in external memory space 5-51
  stalls 12-66
  storing instructions in internal memory space 5-50
Program flow control instructions
direct jump|call (type 8) instructions A-6
do until (type 13) instructions A-7
do until counter expired (type 12) instructions A-7
  indirect jump or compute/dreg⇔DM (type 10) instructions A-6
  summary of A-6
Program memory data accesses 3-10
  branch instructions, see Branch instructions
  instruction cache 3-10
  loop instructions, see Loop instructions
Program segments
  alignment in external memory space 5-51
  invalid external memory addresses 5-52
  multiple, using 5-51
Program sequencer
  architecture, see Program sequencer architecture
  conditional instructions and loop termination conditions evaluation 3-7
  defined 5-5
  generating 24-bit PM bus addresses 5-26
  generating 32-bit DM bus addresses 5-26
  generating memory addresses 5-11
  operation, see Program sequencer operation
  sources of fetch addresses 3-6
  summary of functions 3-2
INDEX

Program sequencer architecture 3-6
  decode address register 3-6
  diagram of 3-6
  fetch address register 3-6
  instruction cache, see Instruction cache
  interrupt controller 3-7
  loop address stack 3-7
    see Loop address stack
  PC stack 3-6
    see also PC stack
  program counter 3-6
  status stack 3-7
  system registers 3-7
    see also Program sequencer registers
  universal registers 3-7
Program sequencer interrupts 3-38
  arithmetic exceptions 3-38
  circular buffer data overflows 3-38
    described 3-38
    external 3-38
    internal 3-38
  interrupt servicing stages 3-40
    IRQx, see IRQx
    latency 3-40
  MODE1 register and 3-38
  RTI instruction and 3-38, 3-39
  servicing 3-38
  servicing sequence 3-39
  stack overflows 3-38
  valid status 3-38
Program sequencer operation 3-10
  branch instructions, see Branch
  instructions
    condition codes, summary of 3-13
    conditional instruction execution,
      see Conditional instructions
    CURLCNTR value and loop iterations 3-34
    evaluating conditions 3-12
    IDLE and IDLE16 instructions 3-56
    instruction cache 3-10
      see Instruction cache
    interrupt latency 3-40
    interrupt servicing stages 3-40
    interrupt vector table and 3-44
    interrupts, see Program sequencer interrupts
    loop address stack, see Loop address stack
    loop instructions, see Loop instructions
    multiprocessor vector interrupts 3-52
    nested interrupt servicing 3-48
    program memory data accesses 3-10
    reads and writes of the same memory block 3-10
    sequential program flow 3-10
    software interrupts 3-49
    status stack save and restore 3-48
    Program sequencer registers 3-7
    LADDR, see LADDR register
    loop address stack 3-7
      see also Loop address stack
overflow interrupts 3-24
PC stack pointer 3-7
pipelining effects on writes to 3-8
program counter stack pointer, see
    PCSTKP
read and effect latencies, summary
    of 3-8
readable registers 3-7
stack flags 3-54
status stack 3-7
system register bit manipulation
    instruction and 3-7
update timing 3-8
writable registers 3-7
Program sequencing 3-1
    clearing current interrupt for reuse
    3-49
clock rate 3-4
DAG2 3-7
external interrupt timing and
    sensitivity 3-50
IDLE and IDLE16 instructions
    3-56
IDLE instruction 3-1
instruction cache 3-7
instruction cycle, see Instruction cycle
instruction processing rate 3-4
interrupt latency 3-40
interrupt masking and control
    3-46
interrupt priority 3-45
    see also Interrupt priority
interrupts 3-1
jump instructions 3-1
loop instructions 3-1
multiprocessor vector interrupts
    3-52
nested interrupt routines 3-7
nested loops 3-7
nonsequential program operations
    3-5
pipelining 3-4, 3-8
program sequencer, see Program sequencer
program structures 3-1
programmable timers and 3-53
saving and restoring the status stack 3-48
software interrupts 3-49
subroutines 3-1
variation in program flow,
    diagram of 3-3
vector interrupt feature, using 3-52
Program structures 3-1
    branches 3-11
IDLE 3-1
interrupts 3-1
jumps 3-1
loops 3-1, 3-11
subroutines 3-1
Programmable I/O and SDRAM
    control register, see IOCTL register
Programmable I/O ports
    bitwise operations on 11-13
described 11-13
INDEX

FLAG11-4 11-13
functionality 11-13
IOSTAT register and 11-13
MODE2 register and 11-13
Programmable I/O status register, see IOSTAT register
Programmable timer pins, see PWM_EVENTx
Programmable timers 3-53
control bits and interrupt vectors 11-8
see Timer control bits and interrupt vectors
counters, maximum period of 3-53
enabling 11-1
features 3-53
functions 11-1
I/O pins 3-53
input/output pin 11-1
interrupts and the status stack 11-9
see Timer interrupts and the status stack
pulse width count/capture 11-1
see WIDTH_CNT timer mode
pulse width waveform generation 11-1
see PWMOUT timer mode
PWM_EVENTx pins 3-53
registers 11-1
TCOUNTx register 3-53
timer counter mode, see PWMOUT
timer counters, size of 11-1
timer register default values 11-11
timer/disable timing, diagram of 11-2
TPERIODx register 3-53
TPWIDTHx registers 3-53
Programming and memory 13-9
16-bit short words, reading 13-10
dual data accesses, performing 13-9
memory access space, restrictions 13-10
Programming and the computation units 13-6
compute operations 13-6
restrictions on delayed branching 13-7
writing twice to the same Register File location 13-7
Programming and the DAGs 13-8
illegal DAG register transfers 13-8
initializing circular buffers 13-9
Programming considerations
component-specific operations 13-6
computation units 13-6
see Programming and the computation units
DAG register writes 13-4
DAGs 13-8
see Programming and the DAGs
extra cycle conditions 13-1
loop accesses of program memory data 13-2
memory

see Programming and memory

nondelayed branches 13-1
one- and two-instruction loops, using 13-4
program memory data accesses with cache miss 13-2
summary of 13-1
wait state programming 13-5
PS (DMA pack status) bit 6-14
described 6-16
values for EPBx packing status 6-16
Pulse capture timer mode, see WIDTH_CNT timer mode
PULSE_CAPx (timer pulse captured) bit 11-6
PULSE_HIx (timer leading edge select) bit
described 11-8
WIDTH_CNT timer mode 11-6
PUSH LOOP instruction
loop address stack and 3-33
 Push|pop stacks/flush cache (type 20) instruction
described A-75
example A-75
opcode A-75
syntax summary A-9
PWM output/capture, see PWM_EVENTx
PWM_EVENTx 11-1
external interrupt and timer pins 12-28
pin definition 12-17
programmable timer I/O 3-53
PWMOUT timer mode 11-3
state after reset 12-24
task-on-demand control 12-28
WIDTH_CNT timer mode 11-5
PWMOUT timer mode 11-1
avoiding unpredictable results from the PWM_EVENTx signal 11-3
described 11-3
PWM_EVENTx operation 11-3
PWM_EVENTx timer pin and 11-3
PWMOUTx (timer mode control) bit 11-3
selecting 11-3
timer flow diagram 11-4
timer interrupts 11-3
TPERIODx register and 11-3
TPWIDTHx register and 11-3
PWMOUTx (timer mode control) bit
described 11-8
PWMOUT timer mode 11-3
WIDTH_CNT timer mode 11-5
PX bus connection 5-5, 5-11
PX data transfers
40-bit DM data bus 5-14
48-bit accesses of program memory 5-14
between DM data bus and external memory 5-14
between DM data bus and
INDEX

internal memory 5-14
between memory and registers 5-12
between PM and DM data buses 5-11
between PX1 and PM data bus 5-12, 5-14
between PX2 and DM data bus 5-14
between PX2 and PM data bus 5-12
data alignment 5-12
diagram of 5-13
example code for 48-bit program memory access 5-14
universal register-to-register 5-12
PX registers
40-bit data accesses with 48-bit words 5-40
architecture 5-12
bus connection 5-5
diagram of 5-12
PX1 alignment 5-12
PX2 alignment 5-12
subregister alignment 5-12
using 5-12
word width of internal bus accesses 5-28

RBWM
avoiding boot hold off 12-52
EPROM booting 12-52

RBWS
EPROM booting 12-52
RCLKDIV receive clock divisor 9-41
described 9-41
I²S SPORT mode 9-62
SPORT clock source and 9-50
RCLKx 9-4, 9-5
clock signal options 9-50
connection in multichannel
SPORT mode 9-67
pin definition 12-67
SPORT loopback mode 9-88
state after reset 12-24

RD
external memory space interface and 5-45
pin definition 12-17
state after reset 12-23
RDIVx register 9-5, 9-9
address of E-78
bit definitions E-80
clock and frame sync frequencies 9-39
default bit values, diagram of E-79
described E-78
divisor bit fields 9-41
memory-mapped address and reset value 9-10, 9-11
RCLKDIV 9-41
reset and E-78

RAS
pin definition 12-10
state after reset 12-23
RAS to CAS delay 10-41
INDEX

RFS signal frequencies 9-5
RFSDIV 9-41
Read (SDRAM) command 10-33
Read and effect latencies of system registers 3-8
Read latency
defined E-4
system registers E-4
Reading the IOP registers 7-27
Reads of a slave processor’s IOP registers 8-17
Receive clock (RCLKx) pins 9-4
Receive frame sync (RFSx) pins 9-4
Receive overflow status bit, see
ROVF (receive overflow) status bit
Receive shift register 9-5
Redefining priority for external port DMA channels 6-38
READY
assertion restrictions 8-12
changing to active-drain output 8-12
host interface 8-8
host IOP register reads 8-17
implementing broadcast writes 8-23
open-drain output 8-12
pin definition 12-9
response to CS, delay 8-11
state after reset 12-24
writes to a full slave write FIFO buffer and 8-17
Ref command 10-38
Refresh command (SDRAM), see
Ref command
Register File 2-9
access characteristics 2-9
alternate registers 2-11
see Alternate register file registers
computation units and 2-9
data writes, sources of 2-10
defined 5-6
fields for Shifter bit field deposit and extract operations, diagram of 2-42
fields for Shifter instructions, diagram of 2-42
individual data registers 2-10
see Individual register file registers
MR register transfers 2-30
multifunction operation operands and 2-50
multifunction operations and B-94
multiplier fixed-point results 2-28
PM data bus transfers and 5-11
shifter operations and B-63
shifter output 2-41
SPORT control registers and 9-12
structural and functional characteristics 2-9
system register bit manipulation instruction and E-6
register file
ALU operations and 2-13
Register handshake message passing protocol 7-37, 8-37
Register modify/bit-reverse (type 19) instruction
described A-73
example A-73
opcode (with bit reverse) A-74
opcode (without bit-reverse) A-73
Register types
  multiplier registers A-17
  summary of A-15
  universal registers A-15
Register write-back message passing
  protocol 7-38, 8-38
Reinitializing DMA channels
  (FLSH) 6-18
  latency 6-18
  restrictions 6-18
Requesting bus lock 7-34
RESET
  bit write restriction 3-44
  bus arbitration synchronization after 7-21
  input hysteresis
  pin definition 12-18
  programmable timer register
    initialization values 11-11
    state after reset 12-24
Reset initialization values of the
  WAIT register 5-55
Resource sharing 7-34
Return from interrupt, see RTI
  instruction 3-16
Return from subroutine, see RTS
  instruction 3-16
Return from subroutine|interrupt/compute (type 11) instruction
described A-55
example A-56
opcode (return from interrupt) A-57
opcode (return from subroutine) A-56
Return instructions 3-16
  conditional branching 3-16
  return from interrupt (RTI), see RTI instruction 3-16
  return from subroutine (RTS), see RTS instruction 3-16
Reusing the current interrupt 3-49
RFS signal 9-5
RFSDIV receive frame sync divisor 9-41
described 9-42
  frame sync source and 9-54
RFSR (receive frame sync requirement) bit 9-21
defined 9-33
  described 9-52
RFSx 9-4
  connection in multichannel
    SPORT mode 9-69
  I2S word select 9-63
  multichannel SPORT mode
    frame sync source 9-69
    multichannel timing reference 9-69
    pin definition 12-12
INDEX

SPORT loopback mode 9-88
state after reset 12-24
Rn= -Rx (fixed-point) operation
ALU status flags B-16
described B-16
Rn=(Rx-Ry)/2 (fixed-point)
operation
ALU status flags B-10
described B-10
Rn=ABS Rx (fixed-point) operation
ALU status flags B-17
described B-17
Rn=ASHIFT Rx BY ⟨data8⟩
operation
described B-67
shifter status flags B-67
Rn=ASHIFT Rx BY Ry operation
described B-67
shifter status flags B-67
Rn=BCLR Rx BY ⟨data8⟩ operation
described B-70
shifter status flags B-70
Rn=BCLR Rx BY Ry operation
described B-70
shifter status flags B-70
Rn=BSET Rx BY ⟨data8⟩ operation
described B-71
shifter status flags B-71
Rn=BSET Rx BY Ry operation
described B-71
shifter status flags B-71
Rn=BTGL Rx BY ⟨data8⟩ operation
described B-72
shifter status flags B-72
Rn=BTGL Rx BY Ry operation
described B-72
shifter status flags B-72
Rn=CLIP Rx BY Ry (fixed-point)
operation
ALU status flags B-25
described B-25
Rn=EXP Rx operation
described B-86
shifter status flags B-86
Rn=EXP Rx(EX) operation
described B-87
shifter status flags B-87
Rn=FDEP Rx BY ⟨bit6⟩⟨len6⟩
operation
described B-74
example B-75
shifter status flags B-75
Rn=FDEP Rx BY ⟨bit6⟩⟨len6⟩(SE)
operation
described B-78
example B-79
shifter status flags B-79
Rn=FDEP Rx BY Ry (SE) operation
described B-78
example B-79
shifter status flags B-79
Rn=FDEP Rx BY Ry operation
described B-74
example B-75
shifter status flags B-75
Rn=FEXT Rx BY ⟨bit6⟩⟨len6⟩(SE)
operation
described B-84
INDEX

element B-84
shifter status flags B-85
Rn=FEXT Rx BY \langle\text{bit6}\rangle:\langle\text{len6}\rangle
operation
described B-82
example B-83
shifter status flags B-83
Rn=FEXT Rx BY Ry (SE) operation
described B-84
example B-84
shifter status flags B-85
Rn=FIX Fx BY Ry operation
ALU status flags B-40
described B-39
Rn=FIX Fx operation
ALU status flags B-40
described B-39
Rn=FPACK Fx operation
described B-90
gradual underflow B-90
results of B-90
shifter status flags B-91
short float data format B-90
Rn=LEFT0 Rx operation
described B-89
shifter status flags B-89
Rn=LEFTZ Rx operation
described B-88
shifter status flags B-88
Rn=LOGB Fx operation
ALU status flags B-38
described B-38
Rn=LSHIFT Rx BY \langle\text{data8}\rangle
operation
described B-65
shifter status flags B-65
Rn=LSHIFT Rx BY Ry operation
described B-65
shifter status flags B-65
Rn=MANT Fx operation
ALU status flags B-37
described B-37
Rn=MAX (Rx, Ry) (fixed-point)
operation
ALU status flags B-24
described B-24
Rn=MIN (Rx, Ry) (fixed-point)
operation
ALU status flags B-23
described B-23
Rn=MRB+Rx*Ry mod2 operation
described B-55
Rn=MRB+Rx*Ry mod2 operation
multiplier status flags B-55
Rn=MRB-Rx*Ry mod2 operation
described B-56
multiplier status flags B-56
Rn=MRF+Rx*Ry mod2 operation
described B-55
multiplier status flags B-55
Rn=MRF-Rx*Ry mod2 operation
described B-56
multiplier status flags B-56
Rn=NOT Rx (fixed-point)
operation
ALU status flags B-22
described B-22
INDEX

Rn=PASS Rx (fixed-point) operation
   ALU status flags B-18
described B-18
Rn=Rn OR ASHIFT Rx BY (data8) operation
described B-68
   shifter status flags B-68
Rn=RN OR ASHIFT Rx BY Ry operation
described B-68
   shifter status flags B-68
Rn=Rn OR FDEP Rx BY (bit6):(len6) (SE) operation
described B-80
   example B-80
   shifter status flags B-81
Rn=Rn OR FDEP Rx BY (bit6):(len6) operation
described B-76
   example B-76
   shifter status flags B-77
Rn=Rn OR FDEP Rx BY Ry (SE) operation
described B-80
   example B-80
   shifter status flags B-81
Rn=Rn OR FDEP Rx BY Ry operation
described B-76
   example B-76
   shifter status flags B-77
Rn=Rn OR LSHIFT Rx BY Ry operation
described B-66
   shifter status flags B-66
   Rn=Rn OR LSHIFT Rx BY (data8) operation
described B-66
   shifter status flags B-66
Rn=RND MRB mod1 operation
described B-58
   multiplier status flags B-58
Rn=RND MRF mod1 operation
described B-58
   multiplier status flags B-58
Rn=ROT Rx BY (data8) operation
described B-69
   shifter status flags B-69
Rn=ROT Rx BY Ry operation
described B-69
   shifter status flags B-69
Rn=Rx AND Ry (fixed-point) operation
   ALU status flags B-19
described B-19
Rn=Rx OR Ry (fixed-point) operation
   ALU status flags B-20
described B-20
Rn=Rx XOR Ry (fixed-point) operation
   ALU status flags B-21
described B-21
Rn=Rx*Ry mod2 operation
   multiplier status flags B-54
Rn=Rx*Ry mode2 operation
   described B-54
INDEX

Rn=Rx+1 (fixed-point) operation
ALU status flags B-14
described B-14

Rn=Rx+Cl (fixed-point) operation
ALU status flags B-12
described B-12
saturation mode B-12

Rn=Rx+Cl-1 (fixed-point) operation
ALU status flags B-13
described B-13
saturation mode B-13

Rn=Rx+Ry (fixed-point) operation
ALU status flags B-6
described B-6
saturation mode B-6

Rn=Rx+Ry+Cl (fixed-point) operation
ALU status flags B-8
described B-8
saturation mode B-8

Rn=Rx-1 (fixed-point) operation
ALU status flags B-15
described B-15

Rn=Rx-Ry (fixed-point) operation
ALU status flags B-7
described B-7
saturation mode B-7

Rn=Rx-Ry+Cl (fixed-point) operation
ALU status flags B-9
described B-9
saturation mode B-9

Rn=SAT MRB mod1 operation
described B-57
multiplier status flags B-57

Rn=SAT MRF mod1 operation
described B-57
multiplier status flags B-57

Rn=TRUNC Fx BY Ry operation
ALU status flags B-40
described B-39

Rn=TRUNC Fx operation
ALU status flags B-40
described B-39

RND32 (floating-point rounding boundary) bit 2-14
32-bit data in 40-bit systems, using 5-41
32-bit IEEE results 2-15
40-bit results 2-15
multiplier floating-point operation 2-32, 2-33
vs. IMDWx 5-41

ROM boot wait mode (RBWM) 5-56
ROM boot wait state (RBWS) 5-57
Rotating priority for external port DMA channels 6-37
DCPR bit 6-37
described 6-37
vs. fixed priority 6-38
vs. SPORT channel priorities 6-38

Rounding modes
described 2-7
round-toward-zero 2-7
Rounding MR register 2-30
INDEX

Round-toward-zero rounding mode 2-7
ROVF (receive overflow status) bit 9-22, 9-38
defined 9-33
described 9-38
RTFS (active state RFS) bit described 9-55
RTI instruction 8-38
ASTAT register and 3-16
described 3-16
EPROM booting 12-54
host booting 12-58
IMASKP register and 3-16
IRPTL register and 3-16
MODE1 register and 3-16
nested interrupts 3-47
program sequencing interrupts and 3-38
status stack pop and 3-16
status stack restore of ASTAT 3-48
status stack restore of MODE1 3-48
RTS instruction
described 3-16
LR modifier and reusing the current interrupt 3-50
RXS (receive data buffer status) bits 9-22, 9-38
defined 9-33
described 9-38
SPORT reset and 9-7
RXx_z data buffer 9-9
data formats and 9-44
described 9-13
interrupts 9-14
memory-mapped address and reset value 9-10, 9-11, 9-12
operation, see RXx_z data buffer operation
read/write restrictions 9-15
reading/writing 9-14
reads of an empty buffer 9-14
receive overflow condition
ROVF (receive overflow) status bit 9-14
receive shift buffer 9-13, 9-44
size of 9-13
SPORT reset and 9-7
RXx_z data buffer operation 9-14
architecture 9-14
described 9-14
storage capacity 9-14

S
Saturate MR register 2-31
valid maximum saturation values 2-31
SBTS (suspend bus three-state)
host bus acquisition 8-11
pin definition 12-6
state after reset 12-24
system bus access deadlock, resolving 8-49
SBTS and HBR combination
applying 8-49
restrictions 8-49
INDEX

SCHEN (SPORT DMA chaining)
  bit 6-23, 9-16, 9-22
  defined 9-34
  enabling chaining on a SPORT
  DMA channel 9-85
  setting up DMA on SPORT
  channels 9-79
SDA10
  pin definition 12-10
  state after reset 12-23
SDCKE
  pin definition 12-11
  state after reset 12-23
SDCLKx
  pin definition 12-10
  state after reset 12-23
SDEN (SPORT DMA enable) bit
  6-23, 9-16, 9-22
  defined 9-34
  I²S SPORT mode 9-65
  multichannel receive comparisons
  and 9-74
  setting up DMA on SPORT
  channels 9-79
SDRAM 2x clock output, see
  SDCLKx
SDRAM A10 pin, see SDA10
SDRAM access 10-26
  A11 pin and 16M devices 10-27
  DQM pin operation 10-27
  mapping ADDRx bits 10-26
  multiplexed 32-bit SDRAM
  address, diagram of 10-26
SDRAM bank select bit, see
  SDRAM configuration
SDRAM burst stop command, see
  Bstop command
SDRAM clock enable, see SDCKE
SDRAM column access strobe, see
  CAS
SDRAM configuration 10-13
  active command delay 10-21
  buffering option 10-17
  CAS latency value 10-18
  clock enables and non-SDRAM
  systems 10-15
  clock enables for heavy clock loads
  10-16
  clock enables for minimal clock
  loads 10-15
  configuration parameters,
  summary of 10-13
DSDCK1 10-9, 10-15
DSDCTL 10-9, 10-15
external memory bank mapping
  10-16
IOCTL control bits 10-9
IOCTL register 10-9, 10-13
IOCTL register default bit values,
  diagram of 10-12
mapping processor addresses to
  SDRAM addresses 10-18
number of banks 10-16
page size 10-18
page size and device organization
  10-19
page size and number of banks
power-up mode 10-19
power-up sequence 10-9
power-up sequence and SDPM bit 10-20
power-up sequence and SDRDIV register 10-20
precharge delay 10-21
refresh counter equation variables 10-14
SDBN 10-11, 10-16
SDBS 10-11, 10-16
SDBUF 10-11, 10-17
SDCL 10-10, 10-18
SDPGS 10-10, 10-18
SDPM 10-10, 10-19
SDPSS 10-11, 10-20
SDRDIV register 10-13, 10-14
SDSRF 10-10, 10-20
SDTRAS 10-10, 10-21
SDTRP 10-10, 10-21
setting the clock enables 10-15
setting the refresh counter value 10-14
starting self-refresh mode 10-20
starting the power-up sequence 10-20
IOCTL register and 10-20
timing requirements 10-9
SDRAM control
controller commands, see SDRAM controller commands
SDBN 10-16

SDRAM control register, see IOCTL register
SDRAM controller commands 10-29
Act 10-30
Bstop 10-30
DMA transfers and 10-36
MRS 10-31
Pre 10-32
Read 10-33
Ref 10-38
Sref 10-28, 10-39
Write 10-35
SDRAM controller operation 10-18
accessing SDRAM devices, see SDRAM access
ADDRx 10-28
data throughput rates 10-23
described 10-23
DMA accesses 10-24
entering and exiting self-refresh mode 10-28
executing a parallel refresh command 10-27
mapping processor addresses to SDRAM addresses 10-18
multiprocessing accesses 10-25
powering up after reset 10-28
SDA10 10-27
SDSRF bit 10-28
SDRAM data mask, see DQM
SDRAM interface 5-6, 10-1
and asynchronous host transfers with the processor 8-9
INDEX

automatic refresh mode 10-6
bank active command 10-5
burst length 10-5
burst stop command 10-5
burst type 10-5
CAS 12-10
CAS latency 10-5
configuration parameters, see
SDRAM configuration
control register, see IOCTL register
controller commands, see
SDRAM controller commands
controller operation, see SDRAM controller operation
data mask I/O function 10-6
data transfer rate 10-1
diagram of 10-2
DQM 12-10
external memory devices 5-63
features 10-1
full-page burst length 10-18
IOCTL register 10-6
meeting multidevice timing requirements 10-17
memory mapping 5-48
mode register 10-6
multiple SDRAM banks, connection to 10-3
multiprocessor bus arbitration and 7-17
normal SBTS operation (HBR deasserted) 5-63
operation cycles 12-26
page size 10-6
pin definitions 12-10
see SDRAM interface pin definitions
precharge command 10-7
RAS 12-10
SDA10 12-10
SDCKE 12-11
SDCLKx 12-10
SDRDIV register 10-7
SDWE 12-11
self-refresh 10-7
self-refresh mode 10-20
setting SDRAM page size 10-18
suspending bus three-state (SBTS) 5-63
system with multiple SDRAM devices, diagram of 10-3
terminology 10-5
timing specifications, see SDRAM timing specifications
\(t_{\text{RAS}}\) active command time 10-7
\(t_{\text{RC}}\) bank cycle time 10-7
\(t_{\text{RCD}}\) RAS to CAS delay 10-8
\(t_{\text{RP}}\) precharge time 10-8
wait states and 5-48
SDRAM interface pin definitions 10-4
CAS 10-4
DQM 10-4
MSx 10-4
RAS 10-4
SDA10 10-4
SDCKE 10-4
INDEX

SDCLK0 10-4
SDCLK1 10-4
SDWE 10-4
SDRAM parallel refresh command 10-27
SDRAM pins, see SDRAM interface pin definitions
SDRAM refresh counter register, see SDRDIV register
SDRAM row access strobe, see RAS
SDRAM timing requirements 10-17
SDRAM timing specifications 10-41
   bank cycle time 10-41
   RAS to CAS delay 10-41
SDRAM write enable, see SDWE
SDRDIV register 10-7, 10-13
   refresh counter equation variables 10-14
   SDRAM power-up sequence and 10-20
      setting the refresh counter value 10-14
      setting the value 10-14
SDWE
   pin definition 12-11
   state after reset 12-23
Self-refresh command (SDRAM), see Sref command
Semaphore, described 7-34
SENDN (endian data word format) bit 9-15, 9-21
   defined 9-35
described 9-48
Sequential program flow 3-10
Serial communication synchronization 9-4
Serial port connections
data receive (DRx_X) pins 9-4
data transmit (DTx_X) pins 9-4
   pins, summary of 9-4
   receive clock (RCLKx) pins 9-4
   receive frame sync (RFSx) pins 9-4
   transmit clock (TCLKx) pins 9-4
   transmit frame sync (TFSx) pins 9-4
Serial ports 9-1
   clock and frame sync frequencies 9-39
   clock signal options, see SPORT clock signal options
   companding, see Companding
   connections, see Serial port connections
   control register status bits 9-38
   control registers 9-9
      see also SPORT control registers
   data buffers 9-9
      see also SPORT data buffers
   data packing and unpacking 9-47
      see also SPORT data packing and unpacking
   data receive inputs 9-4
   data transfer synchronization 9-4
   data transfers between SPORTs and memory
      see SPORT memory transfers
INDEX

data transmit outputs 9-4
data type and nonmultichannel operation 9-44
data word formats 9-44
diagram of 9-3
DMA operation 9-79
driver considerations 9-88
DRx_X 12-11
DTx_X 12-11
features 9-1
frame sync logic level 9-55
frame sync options 9-52
see SPORT frame sync options
frame synchronization 9-5
I2S mode 9-61
see I2S SPORT mode
internally-generated clock frequencies 9-5
interrupts, see SPORT interrupts
loopback mode 9-88
MSB/LSB data word format 9-48
multichannel mode 9-67
see Multichannel SPORT mode
operation cycles 12-27
operation summary 9-5
pin definitions 12-11
pin states after reset 12-24
point-to-point connections on
programming examples 9-89
RCLKx 12-12
RDIVx register 9-5
receive clock signal (RCLKx) 9-5
receive frame sync signal (RFS)
receive shift register 9-5
register and control parameter symbolic names 9-37
reset, see SPORT RESET
RFSx 12-12
RS-232 devices and 9-5
serial data word length 9-48
SPORT data buffer read/write results 9-7
standard mode, see Standard SPORT mode
TCLKx 12-12
TDIVx register 9-5
TFSx 12-12
transmit clock signal (TCLKx)
transmit frame sync signal (TFS)
transmit shift register 9-5
TXx_\_ data buffer 9-5
UARTs and 9-5
Serial RESET, see SPORT RESET
Series termination resistors 12-45
Series-terminated transmission line 12-43
Setting DMA channel prioritzation 6-35
Setting up DMA transfers 6-9
loading the C (count) register 6-9
see also DMA parameter registers
loading the II (index) register 6-9
see also DMA parameter registers
loading the IM (modify) register
INDEX

6-9
see also DMA parameter registers
writing the DMA control registers 6-9
see also DMACx registers
writing the DMA parameter registers 6-9
see also DMA parameter registers
Setting up multiple DMA operations 6-39
Setting up SPORT DMA transfers 6-23
see SPORT DMA
Shadow write FIFO 5-39, 7-32
Shared-bus multiprocessing 8-36
Shifter bit field deposit and extract operations 2-42
bit field definitions 2-43
described 2-42
FDEP bit field deposit instruction example, diagram of 2-44
FDEP instruction 2-43
FDEP instruction bit field, diagram of 2-43
FEXT bit field extract instruction 2-43
element, diagram of 2-45
Register Field fields for, diagram of 2-42
Y-input 2-42
Shifter instruction set summary 2-47
Shifter operations 2-41
bit field deposit and extract 2-42
see Shifter bit field deposit and extract operations
BTST Rx BY (data8) B-73
BTST Rx BY Ry B-73
data transfers 2-41
described B-63
FDEP field alignment, diagram of B-78
FDEP, diagram of B-74
FEXT field alignment, diagram of B-82
FEXT Rx BY Ry B-82
Fn=FUNPACK Rx B-92
instruction set summary 2-47
operands 2-41
output 2-41
Register File and 2-41, B-63
Register File fields for instructions, diagram of 2-42
results 2-42
Rn=ASHIFT Rx BY (data8) B-67
Rn=ASHIFT Rx BY Ry B-67
Rn=BCLR Rx BY (data8) B-70
Rn=BCLR Rx BY Ry B-70
Rn=BSET Rx BY (data8) B-71
Rn=BSET Rx BY Ry B-71
Rn=BTGL Rx BY (data8) B-72
Rn=BTGL Rx BY Ry B-72
Rn=EXP Rx B-86
Rn=EXP Rx(EX) B-87
Rn=FDEP Rx BY (bit6):(len6) B-74
Rn=FDEP Rx BY (bit6):(len6)(SE) B-78
INDEX

Rn=FDEP Rx BY Ry B-74
Rn=FDEP Rx BY Ry (SE) B-78
Rn=FEXT Rx BY (bit6):(len6) B-82
Rn=FEXT Rx BY (bit6):(len6) (SE) B-84
Rn=FEXT Rx BY Ry (SE) B-84
Rn=FPACK Fx B-90
Rn=LEFT0 Rx B-89
Rn=LEFTZ Rx B-88
Rn=LSHIFT Rx BY (data8) B-65
Rn=LSHIFT Rx BY Ry B-65
Rn=Rn OR ASHIFT Rx BY
   (data8) B-68
Rn=Rn OR ASHIFT Rx BY Ry
   B-68
Rn=Rn OR FDEP Rx BY
   (bit6):(len6) B-76
Rn=Rn OR FDEP Rx BY
   (bit6):(len6) (SE) B-80
Rn=Rn OR FDEP Rx BY Ry B-76
Rn=Rn OR FDEP Rx BY Ry (SE)
   B-80
Rn=Rn OR LSHIFT Rx BY Ry
   B-66
Rn=Rn OR LSHIFT Rx
   BY(data8) B-66
Rn=ROT Rx BY (data8) B-69
Rn=ROT Rx BY Ry B-69
single-function compute
   operations B-2
summary of B-63
Shifter status flags 2-45
   overflow flag 2-46
   sign flag 2-46
   SS input sign 2-45
summary of 2-45
SV overflow bits left of MSB 2-45
SZ result 0 2-45
zero flag 2-46
Shifter unit 2-41
   conversion between 16- and
      32-bit floating-point words C-5
described 2-1
instruction set summary 2-47
operations, see Shifter operations
status flags 2-45
   see Shifter status flags
Short loops
   described 3-28
instruction pipeline and 3-28
Short word accesses
   addresses, diagram of 5-42
   arithmetic shifting 5-42
   SSE bit 5-42
Short word addresses
   address region 5-24
   DAG operation on 4-6
diagram of 5-42
   internal memory address region
      5-24
Short word addressing 5-41
   and array signal processing 5-29
   and sign extension 5-30
   and zero-filling 5-30
   arithmetic shifting 5-42
   block 0 noncontiguous addresses
      5-29
INDEX

block 1 address range 5-29
MSW/LSW format 5-29
MSW/LSW of 32-bit words 5-41
normal word conversions 5-41
SSE bit 5-42
vs. normal word addressing 5-29
word width of 5-28
Short word memory accesses 5-41
MSW/LSW of 32-bit words 5-41
normal word conversions 5-41
Short word, floating-point format
described C-5
fields C-5
gradual underflow C-7
results of FPACK and FUNPACK conversion operations C-6
Shifter instructions and C-5
Sign extension of 16-bit short word addresses 5-30, 5-42
Signal glitches 8-46
Signal integrity 12-45
reducing capacitance load 12-45
reducing ringing 12-45
signal paths, adding damping resistance to 12-45
Signal recognition phase 12-27
Signal reflections
reducing 12-46
Signal ringing 12-42
reducing 12-45
Single-bit signaling 12-28
Single-cycle memory accesses, number of 5-17
Single-cycle, parallel accesses 5-9
Single-function compute operations
ALU operations B-2
see ALU single-function compute operations
compute field B-2
CU (computation unit) field B-2
described B-2
OPCODE field B-2
shifter operations B-2
Single-precision, floating-point format
data types, summary of C-3
diagram of C-2
fields C-2
hidden bit C-2
IEEE standard 754/854 C-2
infinity C-3
NAN C-3
normal C-3
normalized numbers C-2
unsigned exponent value range C-2
zero C-3
Single-processor system, diagram of 12-2
Single-word data transfers
defined 7-5
host interface 8-6
Single-word EPBx data transfers
ACK 7-28
core hang 7-29
DEN (DMA enable) bit and 7-29
  described 7-28
DMA interrupts 7-29
  multiprocessing and 7-27
non-DMA transfers 7-29
  reading from an empty buffer 7-28
  writing to a full buffer 7-28
Single-word, non-DMA
  interrupt-driven transfers
INTIO bit 6-46
  performing 6-46
Slave mode DMA 6-20
  configuration 6-59, 7-31
  described 6-59
  extended accesses of EPBx buffers 8-22
EXTERN bit 7-31, 8-21
  external to internal transfer sequence 6-60
HBW bit 8-22
  host data transfers to internal memory space 8-21
HSHAKE bit 7-31, 8-21
  initiating transfers 6-59
  internal to external transfer sequence 6-61
MASTER bit 7-31, 8-21
  multiprocessing DMA transfers 7-31
PMODE bit 8-22
  restriction 6-62
  system-level considerations 6-61
Slave processor
  defined 7-5
  external bus acquisition for read/writes 7-16
  host interface 8-7
  host writes to 8-16
  mode 12-56
Slave write FIFO 7-26, 8-15
  host EPBx writes 8-18
  host read delay 8-17
  writes to a full 8-16
SLEN (serial word length) bits 9-16, 9-21
  defined 9-35
  described 9-48
I2S SPORT mode 9-63
Soft processor reset, see SRST
Software interrupts 3-49
  activating 3-49
  IRPTL register 3-49
Software SPORT reset 9-8
Source termination 12-43
  diagram of 12-44
  guidelines for using 12-44
SPEN (SPORT enable) bit 9-15, 9-21
  defined 9-35
SPL (SPORT loopback mode) bit 9-22
  defined 9-36
  described 9-88
SPORT clock and frame sync frequencies 9-39
CLKIN 9-41
  clock divisor value, equation for
INDEX

calculating 9-42
frame sync divisor value, limitation 9-43
maximum clock rate restrictions 9-43
number of serial clock cycles between frame sync pulses, equation for calculating 9-42
serial clock frequency equation 9-42
value of frame sync divisor, equation for calculating 9-42
SPORT clock signal options 9-50
CKRE 9-50
clock edge 9-50
clock source 9-50
frequency 9-50
ICLK 9-50
internal vs. external clocks 9-50
see also SPORT clock source
RCLKx 9-50
single clock for input and output, use of 9-50
TCLKx 9-50
SPORT clock source 9-50
external 9-51
ICLK 9-50
internal clock 9-50
RCLKx 9-50
serial clock divisor value 9-50
serial clock divisors and external clock source 9-51
TCLKx 9-50
SPORT control registers 9-9
accesses by external devices 9-12
bit definitions 9-26
changing operation mode 9-13
control and status bit active state 9-12
core updates of status bits 9-15
IMASK 9-9, 9-11, 9-12
KEYWDx 9-9, 9-10, 9-12
memory-mapped addresses and reset values, summary of 9-10
MRCCSx 9-9, 9-10, 9-12
MRCSx 9-9, 9-10, 9-11
MTCCSx 9-9, 9-10, 9-11
MTCSx 9-9, 9-10, 9-11
programming 9-12
RDIVx 9-9, 9-10, 9-11
reading/writing 9-12
SRCTLx 9-9, 9-10, 9-11
status bits 9-38
STCTLx 9-9, 9-10, 9-11
summary of 9-9
symbolic names 9-12
TDIVx 9-9, 9-10, 9-11
transmit and receive 9-15
see also STCTLx register and SRCTLx register
write and effect latency 9-13
SPORT data buffers 9-9
core hang condition 9-15
described 9-13
memory-mapped addresses and reset values, summary of 9-10
read/write restrictions 9-15
reads/write of 9-14
receive data buffer operation 9-14
receive shift register 9-13
RXx_z 9-9, 9-10, 9-11, 9-12
size of 9-13
summary of 9-9
transmit data buffer operation
9-13
TXx_z 9-9, 9-10, 9-11, 9-12
SPORT data packing and
unpacking 9-47
data justification 9-47
interrupts 9-48
short word space addresses and
9-48
SPORT data word formats 9-44
companding
see Companding
data type 9-44
see also DTYPE (data type) bits
SPORT divisor registers, see RDIVx
register and TDIVx register
SPORT DMA 9-65
channel assignments 6-22
channels 6-22
connection to internal memory
space 6-27
control bits 6-23
control registers 6-22
data transfers 6-7, 6-22
and the STCTLx and SRCTLx
registers 6-23
data packing 6-22
direction of 6-7, 6-22
SCHEN DMA control bit 6-23
setting up 6-23
DMA-driven data transfer mode
9-65
see DMA-driven data transfer
mode enabling 9-65
internal DMA request and grant
6-35
interrupt-driven data transfer
mode 9-65
see Interrupt-driven data transfer
mode interrupts 6-23
SDEN DMA control bit 6-23
SPORT DMA block transfers
channel priorities 9-78
described 9-77
DMA channels 9-77
DMA interrupts with packing
enabled 9-79
packing 9-78
word size 9-78
SPORT DMA chaining 9-85
chain pointer register and 9-85
described 9-85
see also DMA chaining
SPORT DMA channels 9-77
SPORT DMA interrupts
EP0I 6-23
EP1I 6-23
SPR0I 6-23
SPR1I 6-23
SPT0I 6-23
SPT1I 6-23
INDEX

SPORT DMA operation 9-79
  count register and interrupts 9-81
DMA chaining, enabling 9-79
DMA parameter registers 9-79
  see SPORT DMA parameter registers
  enabling 9-79
RX buffer transfers 9-80
SCHEN 9-79
SDEN 9-79
TX buffer transfers 9-80
SPORT DMA parameter registers 9-79
  architecture 9-81
  chain pointer register 9-82
  count register 9-81
CPRx_X 9-80
CPTx_X 9-80
CRx_X 9-80
CTx_X 9-80
described 9-81
GPRx_X 9-80
GPTx_X 9-80
IIRx_X 9-80
IITx_X 9-80
IMRx_X 9-80
IMTx_X 9-80
index register 9-81
internal memory data buffer and
  9-81
interrupts 9-81
loading 9-80
modify register 9-81
register addresses, summary of

SPORT frame sync options 9-52
described 9-52
frame sync active state 9-55
frame sync clock edge 9-55
frame sync data dependency 9-57
frame sync insert 9-56
frame sync logic level 9-55
frame sync requirement 9-52
frame sync source 9-54
ITFS 9-54
RFSR 9-52
RTFS 9-54
summary of 9-52
TFSR 9-52
SPORT interrupts 9-6
described 9-6
EP0I 9-6
EP1I 9-6
receive DMA interrupt 9-6
SPR0I 9-6
SPR1I 9-6
SPT0I 9-6
SPT1I 9-6
summary of 9-6
timing 9-6
transmit DMA interrupt 9-6
with DMA disabled 9-6
SPORT loopback mode 9-88
described 9-88
SPL bit 9-88
SPORT master mode 9-64
SPORT memory transfers 9-77

I-112   ADSP-21065L SHARC DSP User’s Manual
INDEX

DMA block transfers 9-77
see SPORT DMA block transfers
interrupts 9-77
single-word transfers, see SPORT single-word transfers
transfer methods 9-77
SPORT MSB/LSB data word format 9-48
SPORT multichannel receive companding select register, see MRCCSx register
SPORT multichannel receive select register, see MRCSx register
SPORT multichannel transmit compand select register, see MTCCSx register
SPORT multichannel transmit select register, see MTCSx register
SPORT pin driver considerations 9-88
SPORT programming examples 9-89
DMA transfers with interrupts 9-93
single-word transfers with interrupts 9-91
single-word transfers without interrupts 9-89
SPORT receive clock and frame sync divisors register, see RDIVx register
SPORT receive comparison mask register, see IMASK register
SPORT receive comparison register, see KEYWDx register
SPORT receive control register, see SRCTLx register
SPORT receive data buffer, see RXx_z data buffer
SPORT RESET
data buffer read/write results 9-7
data buffer status bits and 9-7
described 9-7
hardware method 9-8
methods 9-7
RXS (receive data buffer status) bits 9-7
RXx_z data buffer 9-7
software method 9-8
transmit/receive operability 9-8
TXS (transmit data buffer status) bits 9-7
TXx_z data buffer 9-7
SPORT serial word length 9-48
described 9-48
DMA chaining and 9-49
RXx_z buffer operation 9-49
SLEN bit value 9-48
TXx_z buffer operation 9-49
SPORT single-word transfers
BHD (buffer hang disable) bit 9-86
core hang condition and 9-86
core updates of STCTLx and SRCTLx register status bits 9-86
described 9-86
interrupt-driven I/O, implementing 9-86
interrupts 9-86
SPORT transmit clock and frame sync divisors register, see TDIVx register
SPORT transmit control register, see STCTLx register
SPORT transmit data buffer, see TXx_z data buffer
SPORT0 receive DMA channel 0/1 interrupt 9-6
SPORT0 transmit DMA channel 4/5 interrupt 9-6
SPORT1 receive DMA channel 2/3 interrupt 9-6
SPORT1 transmit DMA channel 6/7 interrupt 9-6
SPR0I interrupt
  function and priority 9-6
SPR1I interrupt
  function and priority 9-6
SPT0I interrupt
  function and priority 9-6
SPT1I interrupt
  function and priority 9-6
SRCTLx register 6-23, 9-9, 9-15
  address of E-81
  bit definitions E-85
  CKRE 9-21, 9-26
  control bit definitions 9-26
  control bits, summary of 9-21
  core updates of status bits 9-15
  default bit values (I2S mode), diagram of 9-24, E-83
default bit values (multichannel mode), diagram of 9-25, E-84
default bit values (standard mode), diagram of 9-23, E-82
described E-81
DTYPE 9-21, 9-27, 9-44
effect latency 9-13
I2S mode control bits 9-21, 9-62
ICLK 9-21
IMAT 9-22, 9-28, 9-74
IMODE 9-21, 9-29, 9-74
initialization value E-81
IRFS 9-21, 9-29
L_FIRST 9-21, 9-30
LAFS 9-22, 9-29
LRFS 9-21, 9-30
MCE 9-22, 9-31
memory-mapped address and reset value 9-11
MSTR 9-21, 9-31
multichannel control bits 9-69
multichannel mode control bits 9-21
NCH 9-22, 9-32
OPMODE 9-21, 9-32
PACK 9-21, 9-32
receive comparison control bits 9-74
RFSR 9-21, 9-33
ROVF 9-22, 9-33
RXS 9-22, 9-33
SCHEN 9-22, 9-34
SDEN 9-22, 9-34
INDEX

SENDN 9-21, 9-35
setting up SPORT DMA data transfers 6-23
SLEN 9-21, 9-35
SPEN 9-21, 9-35
SPL 9-22, 9-36
SPORT DMA chaining enable (SCHEN) bit 6-23
SPORT DMA control bits 6-23
SPORT DMA enable (SDEN) bit 6-23
SRCTL0 memory-mapped address and reset value 9-10
standard mode control bits 9-21
status bits 9-38
TCLK 9-28
write latency 9-13
SRCU (alternate register select, computation units) bit
context switching 2-29
MR registers 2-29
SRD1H (DAG1 alternate register select 7-4) bit 4-5
SRD1L (DAG1 alternate register select 3-0) bit 4-5
SRD2H (DAG2 alternate register select 15-12) bit 4-5
SRD2L (DAG2 alternate register select 11-8) bit 4-5
Sref command 10-39
entering and exiting self-refresh mode 10-28
SRRFH (register file alternate select R15-R8/F15-F8) bit 2-11
SRRFL (register file alternate select R7-R0/F7-F0) bit 2-11
SRST (soft reset) bit 7-23
bus arbitration synchronization after 7-21
SS (Shifter input sign) bit 2-45
described 2-46
SSE bit 5-30, 5-42
SSEM bit 3-54
SSOV bit 3-54
Stack overflows 3-38
Standard SPORT mode channel configuration 9-59
CKRE (frame sync clock edge) 9-26
companding 9-59
companding formats 9-44
continuous simultaneous transmissions 9-59
data justification 9-44
data reception 9-59
default bit values, diagram of 9-18, 9-23
described 9-59
DITFS 9-26
DMA requests and interrupts 9-59
DTYPE 9-27, 9-28, 9-44
enabling 9-59
frame sync clock edge 9-55
frame sync configuration 9-59
see Frame sync configuration
frame sync data dependency in 9-57
INDEX

frame sync insert and 9-56
frame sync logic level, configuring 9-55
IMODE 9-29
ITFS 9-29
LAFS 9-29
loopback mode 9-88
LRFS 9-30
MCE 9-31
OPMODE 9-32, 9-36
PACK 9-32
receive control bits 9-21
RFSR 9-33
ROVF 9-33
RXS 9-33
SCHEN 9-34
SDEN 9-34
SENDN 9-35
setting the serial clock frequency 9-60
SLEN 9-35
SPEN 9-35
SPL 9-36
TCLK 9-28
TFS 9-30
TFSR 9-36
transmit configuration 9-59
transmit control bits 9-15
TUVF 9-36
TXS 9-37
using both transmitters simultaneously 9-59
Starting a new DMA sequence 6-9, 6-29
Starting address for contiguous 32-bit data 5-37
Starting address of 32-bit data, equations for 5-35
Starting and stopping DMA sequences 6-48
Status stack 3-7
current values of ASTAT and MODE1 3-49
flags 3-54
programmable timer interrupts and 11-9
pushing and popping 3-7
pushing and popping ASTAT 12-34
pushing and popping IOSTAT 12-34
RTI pop of 3-16
size of 3-48
stack pointer status 3-49
Status stack empty flag 3-54
Status stack flags 3-54
access of 3-54
empty 3-55
overflow and full 3-54
setting 3-54
summary of 3-54
Status stack overflow flag 3-54
Status stack pointer moving 3-49
status stack, pushes and pops of 3-49
Status stack save and restore 3-48
ASTAT register 3-48

INDEX

described 3-48
FLAG_{3:0} bit values 3-48
interrupts that automatically push the status stack 3-48
JUMP (CI) instruction 3-48
MODE1 register 3-48
RTI instruction 3-48
status and control bit preservation 3-48
status and mode contexts 3-48
STCTLx register 6-23, 9-9, 9-15
address of E-90
bit definitions E-94
CHNL 9-17, 9-26
CKRE 9-16, 9-26
control bit definitions 9-26
control bits, summary of 9-15
core updates of status bits 9-15
default bit values (I^{2}S mode), diagram of 9-19, E-92
default bit values (multichannel mode), diagram of 9-20, E-93
default bit values (standard mode), diagram of 9-18, E-91
described E-90
DITFS 9-16, 9-26
DTYPE 9-15, 9-27, 9-44
effect latency 9-13
FS\_BOTH 9-17, 9-28, 9-59
I^{2}S mode control bits 9-15, 9-62
ICLK 9-16
initialization value E-90
ITFS 9-16, 9-29
L\_FIRST 9-16, 9-30

LAFS 9-16, 9-29
LTFS 9-16, 9-30
memory-mapped address and reset value 9-10, 9-11
MFD 9-16, 9-31, 9-71
MSTR 9-16, 9-31
multichannel mode control bits 9-15, 9-69
OPMODE 9-16, 9-32
PACK 9-16, 9-32
SCHEN 9-16, 9-34
SDEN 9-16, 9-34
SENDN 9-15, 9-35
setting up SPORT DMA transfers 6-23
SLEN 9-16, 9-35
SPEN 9-15, 9-35
SPORT DMA chaining enable (SCHEN) bit 6-23
SPORT DMA control bits 6-23
SPORT DMA enable (SDEN) bit 6-23
standard mode control bits 9-15
status bits 9-38
TCLK 9-28
TFSR 9-16, 9-36
TUVF 9-17, 9-36
TXS 9-17, 9-37
write latency 9-13
Sticky bit
defined E-29
Sticky status register, see STKY register
STKY register 2-16
INDEX

AIS 2-17
ALU status flags, summary of 2-17
AOS 2-17
arithmetic exception interrupts and 3-42
arithmetic interrupts, priority of 3-45
AUS 2-17
AVS 2-17
bit definitions E-29
circular buffer overflow interrupts and 4-13
CNT_EXPx 11-6
CNT_OVFx 11-6
default bit values, diagram of E-28 described E-27
initialization value E-27
loop address stack and 3-33
LSEM 3-54
LSOV 3-54
MIS 2-34
MOS 2-34
multiplier status bits, summary of 2-34
MUS 2-34
MVS 2-34
PC stack flags 3-54
PC stack status flags 3-24
PCEM 3-54
PCFL 3-54
programmable timer overflow status 11-6
programmable timer status bits, summary of 11-11
PULSE_CAPx 11-6
ROVF 9-14
SSEM 3-54
SSOV 3-54
status stack flags 3-54
sticky bit, defined E-29
TUVF 9-14
Storage capacity of on-chip memory 5-17
Subroutines 3-1
call instructions 3-16
Super Harvard architecture, diagram of 1-2
Suspending bus three-state (SBTS) 12-6
see also SBTS
SV (Shifter overflow bits left of MSB) bit 2-45
described 2-46
SV condition 3-13
SW
external memory space interface and 5-46
pin definition 12-6
state after reset 12-23
SWPD (slave write pending data) bit 7-42
semaphore read-write-modify operations and 7-35
Symbol definitions file (def21065L.h) E-116
Synchronization sequence 7-22
Synchronous inputs 12-3
Synchronous write select, see SW
SYSCON register
  address of E-99
ADREDY 8-12
BHD 7-29, 8-19, 9-7, 9-15, 9-86
  bit definitions E-101
  data packing control bits,
  summary of 8-26
  default bit values, diagram of
    8-25, E-100
  described E-99
HBW 6-51, 8-22, 8-24, 8-26
HMSWF 6-54, 8-27
host data packing control bits
  8-25
HPFLSH 8-27
IIVT F-3
IMDW1 8-27
INDW0 8-27
initialization value 8-26, E-99
  internal interrupt vector table
    (IIVT) bit 5-30
  multiprocessing data transfers and
    7-25
SRST 7-21, 7-23
SYSTAT register
  address of E-106
  bit definitions 8-40, E-108
BSYN 7-22, 7-42, 8-40
CRBM 7-42, 8-40
default bit values, diagram of
  7-41, 8-43, E-107
  described E-106
HPS 7-43, 8-42
HSTM 7-41, 8-40
IDC 7-42, 8-41
initialization value E-106
multiprocessing data transfers and
  7-25
multiprocessing status
  information 8-40
  status bits 7-40
SWPD 7-35, 7-42
VIPD 3-52, 7-39, 7-42, 8-38,
  8-42
System bus
  arbitrating for control of 8-44
  arbitration unit 8-44, 8-51
  core accesses of 8-48
  host interface with 8-44
  ISA 8-44
  master processor accesses of 8-46
  PCI 8-44
System bus access deadlock
  HBR 8-49
  resolving
  SBTS 8-49
  SBTS and HBR combination
    8-49
System clock
  cycle reference for host interface
  operations 8-7
  frequencies of operations 12-26
System configuration register, see
  SYSCON register
System configurations for
  interprocessor DMA 6-70
System control
<table>
<thead>
<tr>
<th>BMS</th>
<th>12-13</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMSTR</td>
<td>12-13</td>
</tr>
<tr>
<td>BRx</td>
<td>12-14</td>
</tr>
<tr>
<td>BSEL</td>
<td>12-14</td>
</tr>
<tr>
<td>CLarkin</td>
<td>12-14</td>
</tr>
<tr>
<td>CPA</td>
<td>12-16</td>
</tr>
<tr>
<td>FLAGx</td>
<td>12-16</td>
</tr>
<tr>
<td>IDx</td>
<td>12-16</td>
</tr>
<tr>
<td>IRQx</td>
<td>12-17</td>
</tr>
<tr>
<td>pin definitions</td>
<td>12-13</td>
</tr>
<tr>
<td>PWM_EVENTx</td>
<td>12-17</td>
</tr>
<tr>
<td>RD</td>
<td>12-17</td>
</tr>
<tr>
<td>RESET</td>
<td>12-18</td>
</tr>
<tr>
<td>WR</td>
<td>12-18</td>
</tr>
<tr>
<td>XTAL</td>
<td>12-19</td>
</tr>
</tbody>
</table>

System design 12-1
accessing on-chip emulation features 12-34
asynchronous inputs 12-3, 12-27
basic single-processor system, diagram of 12-2
boot modes, see Boot modes
booting, see Booting
CLKIN frequencies 12-26
data delays and throughput summary 12-62
data delays, latencies, and throughput 12-62
decoupling capacitors and ground planes 12-46
described 12-1
design recommendations 12-45
enabling the internal clock generator 12-27
executing boundary scans 12-34
execution stalls 12-66
external interrupt and timer pins 12-28
external port data alignment, diagram of 12-21
EZ-ICE emulator, see EZ-ICE emulator
flag inputs 12-31
flag outputs 12-33
Flag pins and 12-28
FLAGx output timing, diagram of 12-34
FLAGxO status bits 12-32
high frequency design issues, see High frequency design issues
input signal conditioning, see Input signal conditioning
input synchronization delay 12-27
internal clock generation 12-26
JTAG interface pins 12-34
latencies and throughput, summary of 12-65
oscilloscope probes 12-47
pin definitions 12-3, 12-4
host interface 12-7
JTAG/emulator 12-19
miscellaneous 12-20
SDRAM interface 12-10
serial port 12-11
system control 12-13
pin operation 12-26
pin states after reset 12-22
INDEX

point-to-point connections on serial ports 12-45
recommended reference literature 12-47
reducing capacitance load 12-45
reducing ringing 12-45
RESET input hysteresis 12-41
  see also RESET
signal integrity 12-45
signal paths, adding damping resistance to 12-45
synchronous inputs 12-3
task-on-demand controls 12-28
test access port 12-34
unused inputs 12-3
XTAL and CLKIN operation 12-26
System register bit manipulation (type 18) instruction 3-7
BIT TST 3-12
BIT XOR 3-12
described A-71, E-5
example A-71
opcode A-72
operations E-6
restricted use E-6
result E-6
  see also BTF (bit test flag) bit
System registers
application access of E-2
ASTAT E-8
bit test flag E-6
defined E-1
described E-2
effect and read latencies E-4
IMASK E-12
initialization values after reset E-3
IRPTL E-12
MODE1 E-16
MODE2 E-21
program sequencer 3-7
read and effect latencies, summary of 3-8, E-5
STKY E-27
summary of E-2
system register bit manipulation instruction
  see System register bit manipulation (type 18) instruction
System status register, see SYSTAT register
SZ (Shifter result 0) bit 2-45
described 2-46
SZ condition 3-13

T
TAP (JTAG test access port)
  ABSDL (boundary scan description language) file D-3
described D-2
  TCK input D-2
  TDI input D-2
  TDO output D-2
  TMS input D-2
  TRST input D-2
TCB 6-5
  and chain loading 6-41
  and the chain pointer 6-41
defined 6-5, 6-39
memory setup for external port
DMA channels 6-43
storage locations 6-41
TCB chain loading 6-5, 6-26
defined 6-5, 6-39
described 6-41
prioritization of DMA channels
6-37
priority of external port DMA
channels 6-37
request prioritization 6-42
request procedure 6-42
see also TCB
sequence summary 6-41
TCB-to-register sequence 6-41
TCK
pin definition 12-19
state after reset 12-25
TCLK (transmit and receive clock
sources) bit
defined 9-28
TCLKDIV transmit clock divisor
9-40
described 9-41
I²S SPORT mode 9-62
SPORT clock source and 9-50
TCLKx 9-4, 9-5
clock signal options 9-50
connection in multichannel
SPORT mode 9-67
pin definition 12-12
SPORT loopback mode 9-88
state after reset 12-24
TCOUNTx register 11-1
reset values 11-11
size of 11-1
TDI
pin definition 12-19
state after reset 12-25
TDIVx register 9-5, 9-9
address of E-78
bit definitions E-80
clock and frame sync frequencies
9-39
default bit values, diagram of E-79
described E-78
divisor bit fields 9-40
memory-mapped address and
reset value 9-10, 9-11
reset and E-78
TCLKDIV 9-40
TFS signal frequencies 9-5
TFSDIV 9-40
TDO
pin definition 12-20
state after reset 12-25
Technical and customer support,
contacting -xx, -xiv
Termination
end-of-line 12-43
propagation delay 12-43
series-terminated transmission
line 12-43, 12-45
source 12-43, 12-44
Termination codes 3-12
see also Condition codes
INDEX

Termination conditions for noncounter-based loops 3-30
TF condition 3-12, 3-14
TFS signal 9-5
TFSDIV transmit frame sync divisor 9-40
described 9-42
frame sync source and 9-54
TFSR (transmit frame sync requirement) bit 9-16
defined 9-36
described 9-52
TFSx pins 9-4
connection in multichannel SPORT mode 9-69
I²S word select 9-63
multichannel SPORT mode transmit data valid signal 9-69
pin definition 12-12
SPORT loopback mode 9-88
state after reset 12-24
TIMENx (timer enable) bit 11-1
described) 11-8
Timer control bits and interrupt vectors
INT_HIx (timer interrupt vector location) 11-9
PERIOD_CNTx (timer period count enable) 11-8
PULSE_HIx (timer leading edge select) 11-8
PWMOUTx (timer mode control) 11-8
TIMENx (timer enable) 11-8

Timer counter timer mode, see \textit{PWMOUT}
Timer interrupts and the status stack 11-9
described 11-9
logical OR of both timer interrupts 11-9
TMZHI and 11-9
Timer pins, see \textit{PWM\_EVENT}x
Timer registers
IOP register addresses of 11-12
TCOUNTx 11-11
TPERIODx 11-11
TPWIDTHx 11-11
TMS
pin definition 12-20
state after reset 12-25
TPERIODx register 11-1
PWMOUT timer mode 11-3
reset values 11-11
size of 11-1
TPWIDTHx register 11-1
PWMOUT timer mode 11-3
reset values 11-11
size of 11-1
TRAN (DMA transfer direction) bit 6-14, 8-28
described 6-15
direction of DMA transfers 6-15
single-word EPBx transfers 8-20
Transfer control block, see \textit{TCB}
Transfer timing example
multichannel SPORT mode 9-68
INDEX

Transferring data between the PM and DM buses 5-12
Transferring data to and from memory 5-7
Transmit clock (TCLKx) pins 9-4
Transmit frame sync (TFSx) pins 9-4
Transmit shift register 9-5
Transmit underflow status, see TUVF (transmit underflow status) bit
\( t_{RAS} \) active command time 10-7
bank cycle time and 10-41
\( t_{RC} \) bank cycle time 10-7
\( t_{RCD} \) RAS to CAS delay 10-8
\( t_{RP} \) precharge time 10-8
TRST
pin definition 12-20
power-up procedures and 12-35
state after reset 12-25
TRUE condition 3-12, 3-15
TRUNC (floating-point rounding mode) bit 2-14
multiplier floating-point operation 2-32
multiplier floating-point operations 2-33
round-to-nearest 2-15
round-to-zero 2-15
\( t_{TRDYHG} \) switching characteristic 8-12
TUVF (transmit underflow status) bit 9-14, 9-17, 9-38
defined 9-36
described 9-39
TXS (transmit data buffer status)
bits 9-17, 9-38
defined 9-37
described 9-39
SPORT reset and 9-7
TXx_z data buffer 9-5, 9-9
data formats and 9-44
described 9-13
memory-mapped address and reset value 9-10, 9-11, 9-12
multichannel operation with DMA enabled 9-69
multichannel TFS operation 9-69
operation, see TXx_z data buffer operation
read/write restrictions 9-15
reading/writing 9-14
size of 9-13
SPORT reset and 9-7
transmit shift buffer 9-44
writes to a full buffer 9-14
TXx_z data buffer operation 9-13
architecture 9-13
described 9-13
interrupts 9-14
storage capacity 9-14
transmit underflow condition 9-14
Type 10 instruction 8-48
and core accesses of the system bus 8-48

U
Unconditional instructions
INDEX

IF TRUE 3-12
Uniprocessor to microprocessor bus interface 8-51
Universal registers A-15
  and bitwise operations 12-29
  and bitwise operations 11-13
ASTAT 11-14
DAG registers 4-15
  data transfers, between 5-12
IMASK 6-47, F-1
IRPTL F-1
  list of A-15
  map 1 register codes A-26
  map 1 registers A-24
  map 1 system registers A-25
  map 2 register codes A-27
  map 2 registers A-25
  program sequencer 3-7
  summary of A-15
  system registers and E-2
Unusable internal memory space addresses 5-24
Unused inputs 12-3
Unused pins 12-20
Ureg⇒DM|PM (direct addressing)
  (type 14) instruction
  described A-63
  example A-63
  opcode A-66
  syntax summary A-8
Ureg⇒DM|PM (indirect addressing) (type 15)
  instruction
  described A-65

V
VDD
  decoupling capacitors and ground planes 12-46
  pin definition 12-20
Vector interrupt table
  VIRPT 7-39
Vector interrupt-driven message passing protocol 7-37, 8-37
Vector interrupts 7-38
  addresses of F-1
  DMA done interrupt 12-58
  generating 8-38
  host 8-38
  host booting and 12-58
I²S DMA-driven data transfer mode 9-65
  immediate high-priority interrupt 8-36
  interprocessor communication 8-36
  interrupt service routines 7-39,
    8-36, 8-38
  address of 8-38
  data for 8-38
  RTI instruction and 8-38
  interrupt vector table 3-44, 7-39,
    8-38
  minimum latency 3-52, 7-39,
    8-38
INDEX

RTI (return from interrupt)
  instruction 8-38
  servicing 7-38, 8-38
  using 3-52, 7-39
VIPD bit 3-52
VIRPT register 7-38
VIPD (vector interrupt pending) bit
  7-42, 8-38, 8-42
  interprocessor messages 7-39
  multiprocessor vector interrupts 3-52
VIRPT register 6-47, 8-36
host booting and 12-58
host interface and 7-36
host interrupt service routines 8-38
host vector interrupts and 8-38
generating 8-38
  servicing 8-38
initialization at reset 8-38
interprocessor messages 7-36, 7-39, 8-36
interrupt service routine 8-38
interrupt vector table and 3-44
minimum latency 3-43
multiprocessing data transfers 7-25
multiprocessor vector interrupts 3-52
shared-bus multiprocessing 8-36
status of 3-52, 8-38
  vector interrupts 7-36, 7-38
VIPD 8-38

W
  WAIT register
  address of E-111
  bit definitions 5-56, E-113
default bit values, diagram of 5-58, E-112
described E-111
  EBxWM 5-56, 5-61
  EBxWS 5-56, 5-60
extending access to off-chip memory 5-54
HIDMA 5-57
initialization value 5-55, E-111
MMSWS 5-57, 5-62
RBWM 5-56, 12-52
RBWS 5-57, 12-52
wait state configuration features 5-55
Wait state modes 5-61
Wait states
  DMA transfers between processor’s internal and external memory 6-74
  EPROM booting 12-52
  multiprocessing data transfers 7-25
  programming clock cycles 12-27
Wait states and acknowledge
  automatic wait state option 5-62
  bus hold time cycle 5-60, 5-61
  bus idle cycle 5-58, 5-59, 5-60
  external memory banks and 5-48
  external memory space 5-53
  IOP control registers 5-53
INDEX

multiprocessor memory space 5-61
off-chip memory access extension 5-53
both (ACK and WAIT register) method 5-54
either (ACK or WAIT register) method 5-54
internal (WAIT register) method 5-54
WAIT register, see WAIT register
WIDTH_CNT timer mode 3-53, 11-1
capture mode 11-6
defining the leading and trailing edges of the PWM_EVENTx signal 11-6
described 11-5
pulse period capture 11-6
pulse width capture 11-6
PWM_EVENTx timer pin operation 11-5
PWMOUTx (timer mode control) bit 11-5
selecting 11-5
timer flow diagram 11-7
timer interrupts 11-6
timer overflow 11-6
timing, diagram of 11-5
TPERIODx and TPWIDTHx registers 11-6
Word select signal described 9-64
FS_BOTH and 9-64
I²S SPORT mode 9-63
timing in I²S SPORT mode, diagram of 9-66
Word size and memory block organization 5-28
Word types, memory 5-28
Word width and memory block organization 5-30
DMA data transfers 6-16
external words 6-52
HBW bits 6-52
instruction fetches 5-28
internal words 6-52
memory accesses 5-28
normal word addressing 5-28
PMODE bits 6-52
PX register over DM bus 5-28
PX register over PM bus 5-28
RND32 and 5-41
short word addressing 5-28
WR
external memory space interface and 5-46
pin definition 12-18
state after reset 12-23
Write (SDRAM) command 10-35
Write latencies
IOP register mode and control bits E-43
IOP registers 7-26, E-42
SPORT control registers 9-13
Writes to a slave processor’s EPBx buffers 8-18
INDEX

Writes to a slave processor’s IOP registers 8-16
Writing the IOP registers
multiprocessing data transfers 7-26
Writing to BMS memory space and BSO 12-56
and CLKin 12-26
enabling the internal clock generator 12-27
internal clock generation 12-26
pin definition 12-19
state after reset 12-24

Z
Zero-filling 16-bit short word addresses 5-30, 5-42

X
XTAL